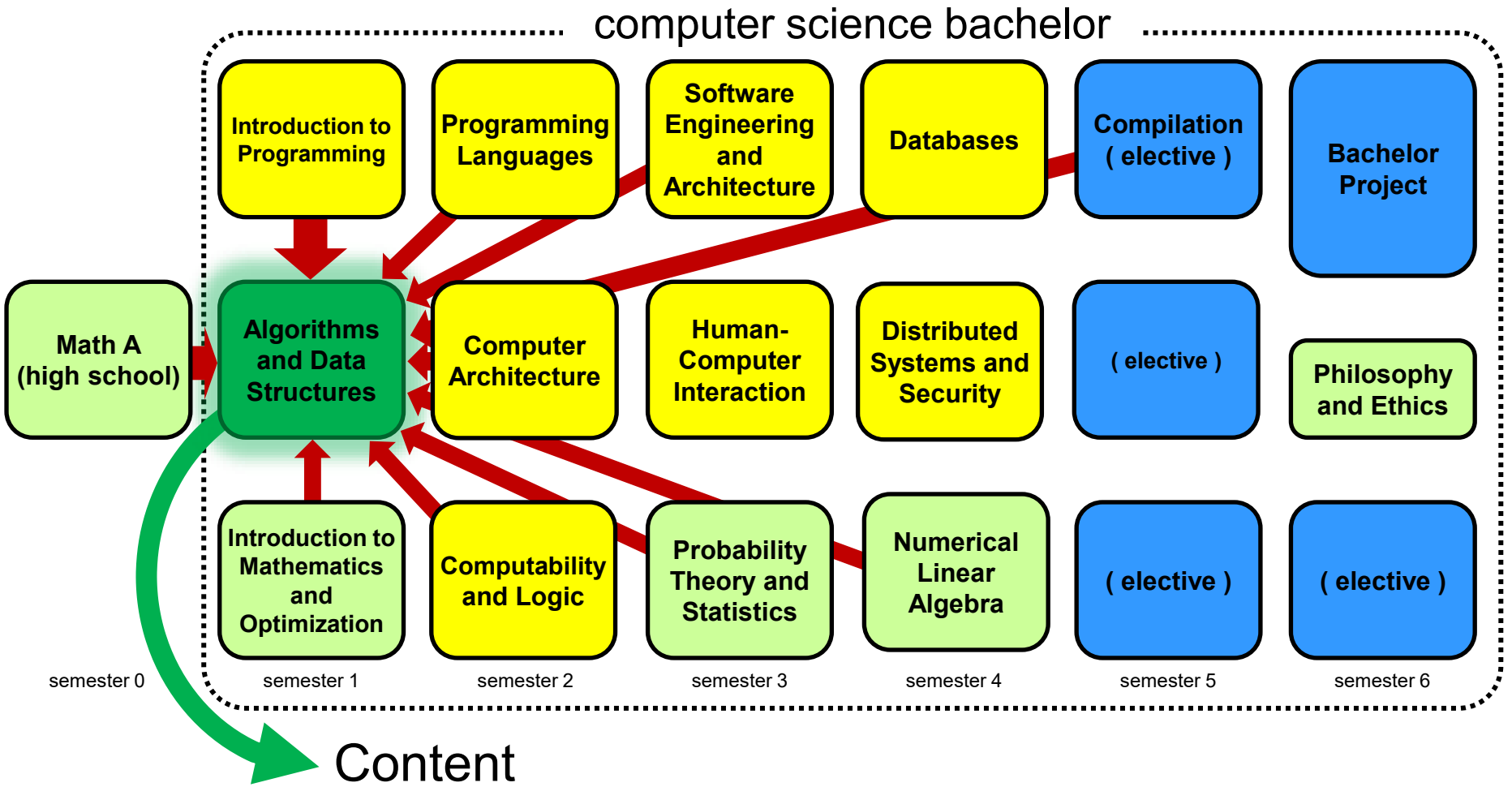
Algorithms and Data Structures

Gerth Stølting Brodal
gerth@cs.au.dk

scientific prerequisites
(ideal world)

Exam : January
Reexam : May



Content

- **Introduction to algorithms and data structures**
- Examples of applications
- Advanced algorithms and data structures
- Research results

ambitious

Course elements

Teaching forms
Lectures 2 + 2 hours/week
Exercise sessions (“TØ” = Teoretiske Øvelser) 3 hours/week
Study café 5 x 2 hours/week
Online discussion forums [Brightspace]

Lectures cover the material
Exercise sessions *work* with the material

Teaching assistant help with exercises and assignments (cs.au.dk/studiecafe)

Assignments (groups, 1-3 students, must be approved)
9 theoretical assignments
4 weeks with programming exercises

Practice algorithmic formulations
Handed in on Brightspace
Deadline set by you and your TA
Resubmission (positive) = TA feedback and possibility to improve
Recommended groups with 2-3 students for ongoing discussion of the problems

Exam
2 hours multiple choice, no aids, 7-scale grading

Language
English
Exercise sessions and study café, also Danish

Previous exams and practice exercises on course webpage

Work load	hours x weeks
Lectures	4 x 14 = 56
Exercise sessions (TØ)	3 x 14 = 42
Study café	1 x 14 = 14
Assignments	3 x 14 = 42
Preparation lectures	2 x 14 = 28
Preparation TØ	2 x 14 = 28
Preparation exam	45
Exam	2
Total hours	**257**

3 hours each week with a teaching assistant

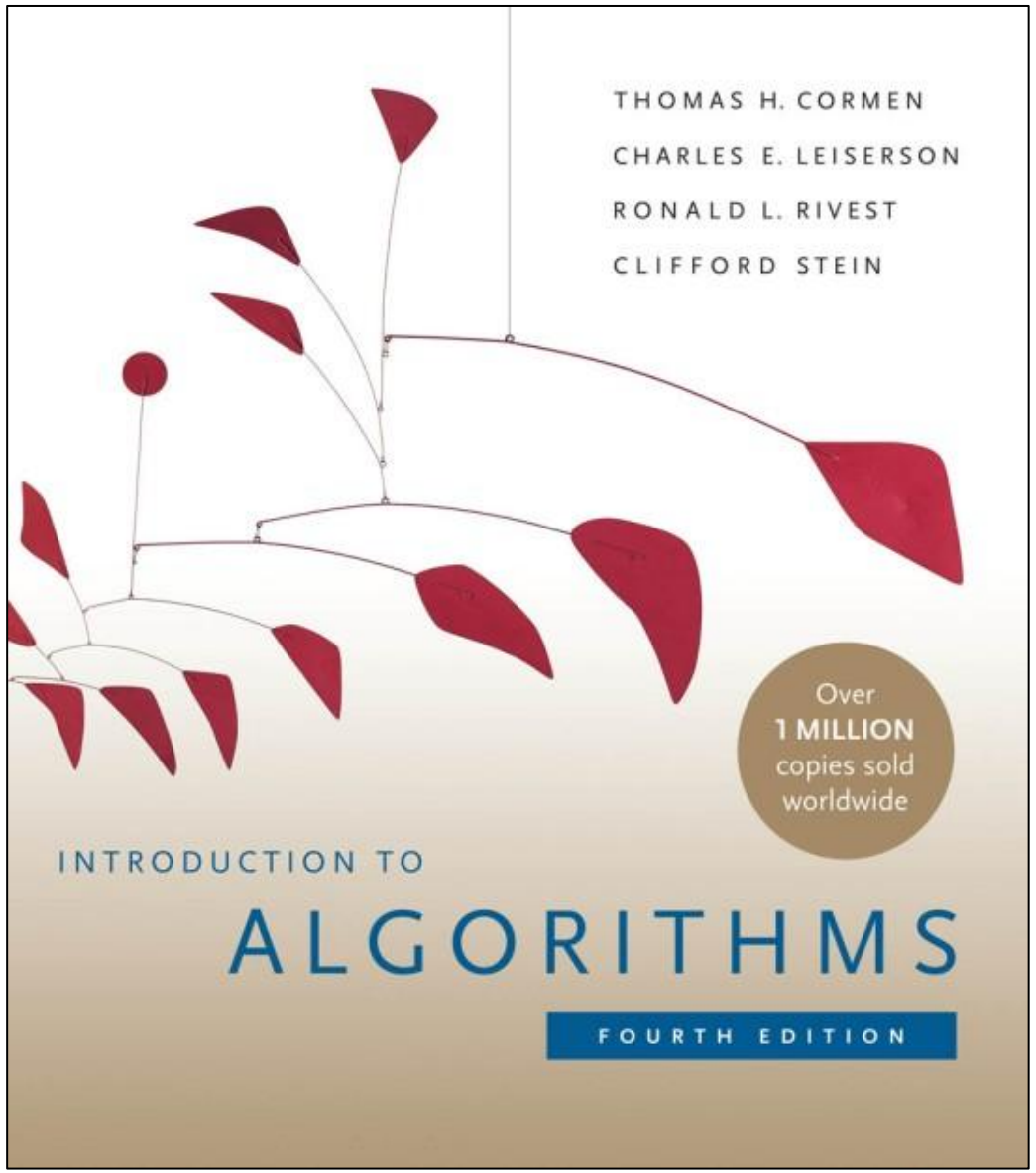
A set of exercises for each TØ

Exercises on “Course plan” associated with the lectures since your last TØ

Format

1. Prepare before TØ with your study group and try to understand and solve all exercises
   Use the study café & discussion forum to get help
2. At TØ students present solutions to the exercises
   The TA helps fixing minunderstandings and emphasizing important points

Make sure to have an ongoing dialog with your TA about the best usage of your TØ hours



Primary textbook

Supplementary Lecture notes

Algorithms and Data Structures

Gerth Stølting Brodal

(updated as the course progresses)

If 0 is considered a natural number depends who you ask !

A "cheat sheet" with mathematical notation and formulas used in the course can be found in "Supplementary Lecture Notes"

Mathematics Cheat Sheet

$\mathbb{N} = \{1, 2, 3, \ldots\}$ natural numbers
$\mathbb{R}$ = real numbers
$\mathbb{R}^+ = \{x \in \mathbb{R} \mid x > 0\}$
Floor $\lfloor x \rfloor$, *ceiling* $\lceil x \rceil$

Associativity
$(a \cdot b) \cdot c = a \cdot (b \cdot c), \quad (a+b)+c = a+(b+c)$

Commutativity
$a \cdot b = b \cdot a, \quad a + b = b + a$

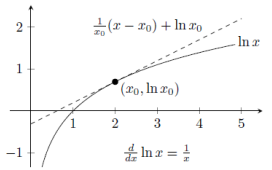
Distributivity
$a \cdot (b + c) = a \cdot b + a \cdot c$

Powers
$a^0 = 1, \quad a^1 = a, \quad a^{b+c} = a^b \cdot a^c$
$(a^b)^c = a^{b \cdot c}, \quad a^{bc} = a^{(b^c)}, \quad a^{-b} = \frac{1}{a^b}$
$(a \cdot b)^c = a^c \cdot b^c, \quad a^{b-c} = a^b / a^c$

Roots
$\sqrt{a} = \sqrt[2]{a} = a^{1/2}, \quad \sqrt[n]{a} = a^{1/n}$
$\sqrt[n]{a \cdot b} = \sqrt[n]{a} \cdot \sqrt[n]{b}, \quad \sqrt[n]{a/b} = \sqrt[n]{a} / \sqrt[n]{b}$

Fractions
$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$
$\frac{a}{b} / \frac{c}{d} = \frac{a \cdot d}{b \cdot c}$
$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + b \cdot c}{b \cdot d}$

Logarithms
$\log_b a = c \Leftrightarrow b^c = a, \quad b^{\log_b a} = a$
Natural logarithm
$\ln a = \log_e a, \quad e = 2.71828\ldots$
Binary logarithm $\log_2 a = \lg a$
$\log_b 1 = 0, \quad \log_b b = 1$
$\log_b(a \cdot c) = \log_b a + \log_b c$
$\log_b(a/c) = \log_b a - \log_b c$
$\log_b(a^c) = c \cdot \log_b a$
$\log_b a = \frac{\log_c a}{\log_c b}, \quad \log_b^c a = (\log_b a)^c$
$a^{\log_b c} = 2^{\log_2 a \cdot \log_2 c \cdot \frac{1}{\log_2 b}} = c^{\log_b a}$

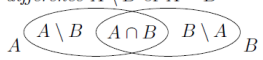


Matrices (1,4)

$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \quad m \times n \text{ matrix}$$

Matrix addition ($m \times n$ og $m \times n$)
$C = A + B, \quad c_{ij} = a_{ij} + b_{ij}$
Matrix multiplication ($m \times n$ og $n \times p$)
$C = AB, \quad c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$
Constant multiplication ($m \times n$)
$C = cA, \quad c_{ij} = c \cdot a_{ij}$
$(A + B) + C = A + (B + C)$
$A + B = B + A$
$c(dA) = (cd)A$
$c(A + B) = (cA) + (cB)$
$(AB)C = A(BC)$
$A(B + C) = (AB) + (AC)$
$(A + B)C = (AC) + (BC)$

Sets
Set $A = \{a_1, a_2, \ldots, a_n\}$
Size $|A|$
Membership $x \in A$, *non-member* $x \notin A$
Empty set $\emptyset, \quad |\emptyset| = 0$
Subset $A \subseteq B$, i.e. $x \in A \Rightarrow x \in B$
Set intersection $A \cap B$
Set union $A \cup B$
Set difference $A \setminus B$ or $A - B$



$A \cup B = (A \setminus B) \cup (A \cap B) \cup (B \setminus A)$
Commutativity
$A \cap B = B \cap A, \quad A \cup B = B \cup A$
Associativity
$A \cup (B \cup C) = (A \cup B) \cup C$
$A \cap (B \cap C) = (A \cap B) \cap C$
Distributivity
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
DeMorgan's laws
$A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$
$A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$
Idempotence $A \cup A = A = A \cap A$
Empty set $A \cup \emptyset = A, \quad A \cap \emptyset = \emptyset$
Complement $\overline{A}$ wrt. *universe* U
$\overline{A} = U \setminus A$ where $A \subseteq U, \quad \overline{\overline{A}} = A$
$\overline{A \cup B} = \overline{A} \cap \overline{B}, \ \overline{A \cap B} = \overline{A} \cup \overline{B}$
A and B are *disjoint* $\Leftrightarrow A \cap B = \emptyset$
Cross product/Cartesian product
$A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$

Sums (2)
$\sum_{i=1}^{n} a_i = a_1 + a_2 + \cdots + a_n$
$\sum_{i=1}^{n} (a_i + b_i) = \sum_{i=1}^{n} a_i + \sum_{i=1}^{n} b_i$
$\sum_{i=1}^{n} (c \cdot a_i) = c \cdot \sum_{i=1}^{n} a_i$
$\sum_{i=0}^{n} 2^i = 1 + 2^1 + \cdots + 2^n = 2^{n+1} - 1$
$\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ for $a \neq 1$
$\sum_{i=0}^{\infty} a^i = \lim_{n \to \infty} \sum_{i=0}^{n} a^i = \frac{1}{1-a}, \ |a|<1$
$\sum_{i=1}^{n} i = 1 + 2 + 3 + \cdots + n = \frac{n(n+1)}{2}$
$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
$\sum_{i=1}^{n} i \cdot a^i = \frac{a}{(1-a)^2}(1 - a^n - n \cdot a^n + n \cdot a^{n+1})$
$\sum_{i=1}^{\infty} i \cdot a^i = \frac{a}{(1-a)^2}$ for $|a| < 1$
Polynomial $P(x) = \sum_{i=0}^{k} c_i \cdot x^i$
Telescoping sum
$\sum_{i=0}^{n-1} (a_{i+1} - a_i) = a_n - a_0$
n-te *harmonic* number $H_n = \sum_{i=1}^{n} \frac{1}{i}$
$= \frac{1}{1} + \frac{1}{2} + \cdots + \frac{1}{n}$
$\ln n + \frac{1}{n} \leq H_n \leq \ln n + 1$
$\lim_{n \to \infty} (H_n - \ln n) = \gamma = 0.577215\ldots$
$=$ Euler–Mascheroni constant

Products
$\prod_{i=1}^{n} x_i = x_1 \cdot x_2 \cdots x_n$
Factorial $n! = \prod_{i=1}^{n} i = 1 \cdot 2 \cdots n$
$\ln\left(\prod_{i=1}^{n} x_i\right) = \sum_{i=1}^{n} \ln(x_i)$
$0 \leq \ln(n!) - (n \ln n - n + 1) \leq \ln n$

Probability theory (3)
Mean $\overline{x} = \frac{x_1 + x_2 + \cdots + x_n}{n}$
Binomial coefficient $\binom{n}{k} = \frac{n!}{(n-k)! \, k!}$
(number of ways to select k elements among n, indepedent on the order)
Linearity of expectation
$\mathrm{E}[\sum_{i=1}^{k} c_i \cdot X_i] = \sum_{i=1}^{k} c_i \cdot \mathrm{E}[X_i]$
Bernoulli distribution $X \sim Bern(p)$
$\Pr[X = 1] = p, \quad \Pr[X = 0] = 1 - p$
Binomial distribution $X \sim Bin(n, p)$
(sum of n Bernoulli trials)
$\Pr[X = k] = \binom{n}{k} \cdot p^k \cdot (1 - p)^{n-k}$
Expected value $\mu = \mathrm{E}[X] = p \cdot n$
Geometric distribution $X \sim Geom(p)$
(number of Bernoulli trials before 1)
$\Pr[X = k] = p \cdot (1 - p)^{k-1}$
$\mathrm{E}[X] = \sum_{k=1}^{\infty} k \cdot \Pr[X = k] = \frac{1}{p}$

Logical expressions
Or/disjunction $U \vee V$
And/conjunction $U \wedge V$
Not/negation $\neg U$
Implication/conditional $U \Rightarrow V$
Equivalent/biconditional $U \Leftrightarrow V$
Exists $\exists x : U(x)$
For all $\forall x : U(x)$

$\vee$	F	T
F	F	T
T	T	T

$\wedge$	F	T
F	F	F
T	F	T

	$\neg$
F	T
T	F

$\Rightarrow$	F	T
F	T	T
T	F	T

$\Leftrightarrow$	F	T
F	T	F
T	F	T

XOR	F	T
F	F	T
T	T	F

F = false, T = true

Asymptotic notation (1)
$f(x) = \mathrm{O}(g(x)) \Leftrightarrow$
$\exists c, x_0 \forall x \geq x_0 : f(x) \leq c \cdot g(x)$
$f(x) = \Omega(g(x)) \Leftrightarrow$
$\exists c > 0, x_0 \forall x \geq x_0 : f(x) \geq c \cdot g(x)$
$f(x) = \Theta(g(x)) \Leftrightarrow$
$f(x) = \mathrm{O}(g(x)) \wedge f(x) = \Omega(g(x))$

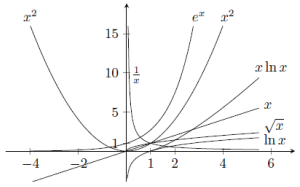
Master Theorem (1)
Constants $a, c, d > 0$, $p \geq 0$ and $b > 1$

$$T(n) = \begin{cases} c & \text{for } n \leq d \\ a \cdot T(n/b) + c \cdot n^p & \text{for } n > d \end{cases}$$

$\Downarrow$

$$T(n) = \begin{cases} \Theta(n^p) & \text{for } a < b^p \\ \Theta(n^p \cdot \log_b n) & \text{for } a = b^p \\ \Theta(n^{\log_b a}) & \text{for } a > b^p \end{cases}$$

Growth of functions



Literature

Mathematical formulas and terms (primary school), Ministry of Education, June 2017 [in Danish]
Mathematical formulas (high school, stx A), Undervisningsministeriet, May 2018 [in Danish]
Thomas H. Cormen *et al.*, Introduction to Algorithms, 4th Edition, appendix A–C, 2022
Steve Seiden, Theoretical computer science cheat sheet, ACM SIGACT News, 27(4), 1996

Covered in (1)this course, (2)introduction to mathematics course, (3)probability course, (4)linear algebra course

Learning outcomes and competences (from course description)

At the end of the course, the participants will be able to

- Formulate and execute algorithms and data structures in the form of **pseudo code**.
- Construct, implement and analyze algorithms using **standard algorithm paradigms**.
- Identify and compare **data structures** and **graph algorithms** for solving algorithmic problems.
- **Construct**, **implement** and **evaluate** the performance of algorithms for simple algorithmic problems.
- Analyze and compare the **time** and **space usage** of algorithms and data structures.
- Identify valid **invariants** and prove the correctness of simple algorithms.

Questions ?

See Brightspace and slides for info!

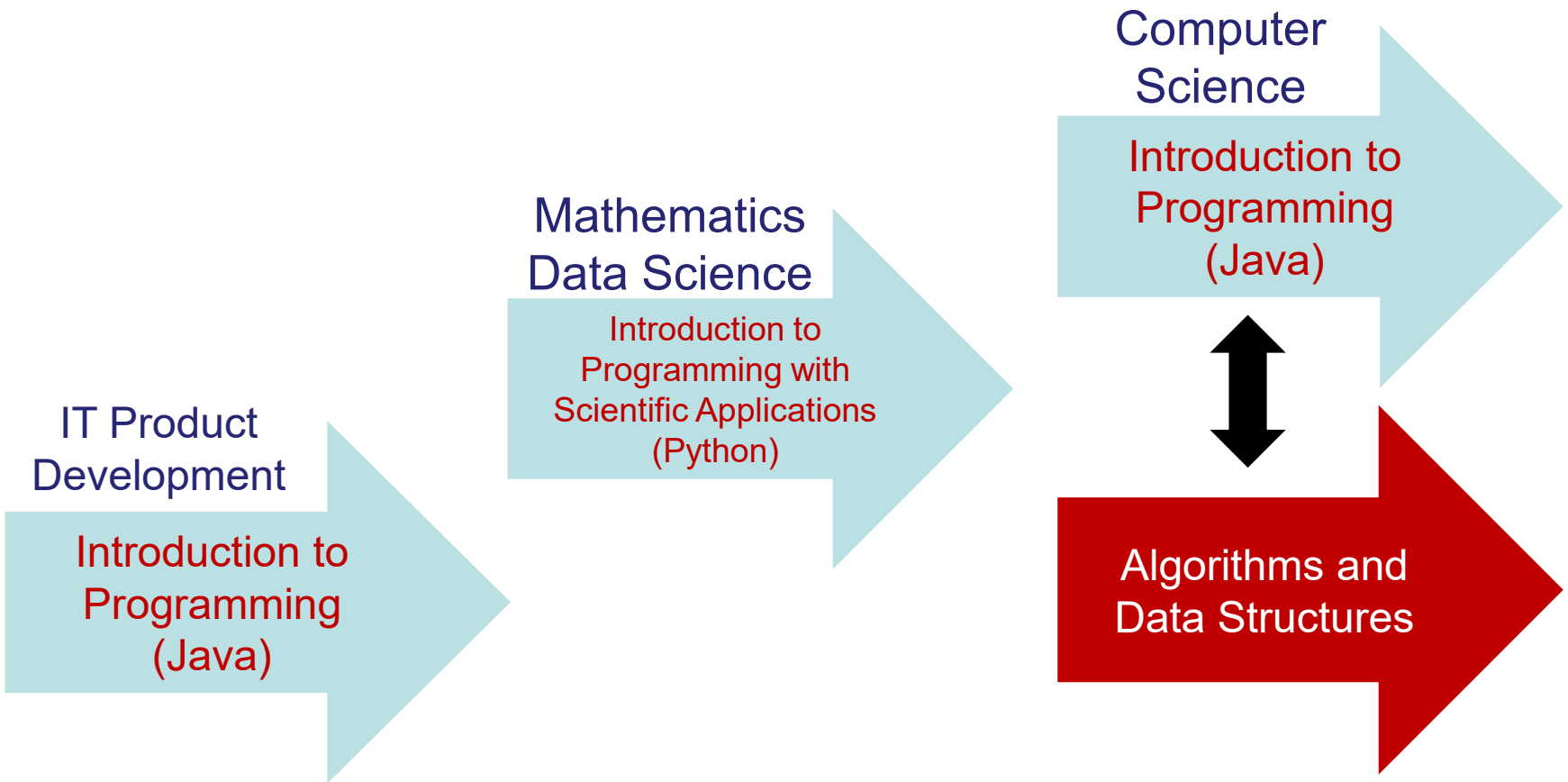
Study

a) Computer science (1st year)
b) Data science (2nd year)
c) IT product development (3rd year)
d) Mathematics (usually 3rd year)
e) Physics (usually 3rd year)
f) other (usually supplement in programming)

Programming experience
(for the programming language you know the best)

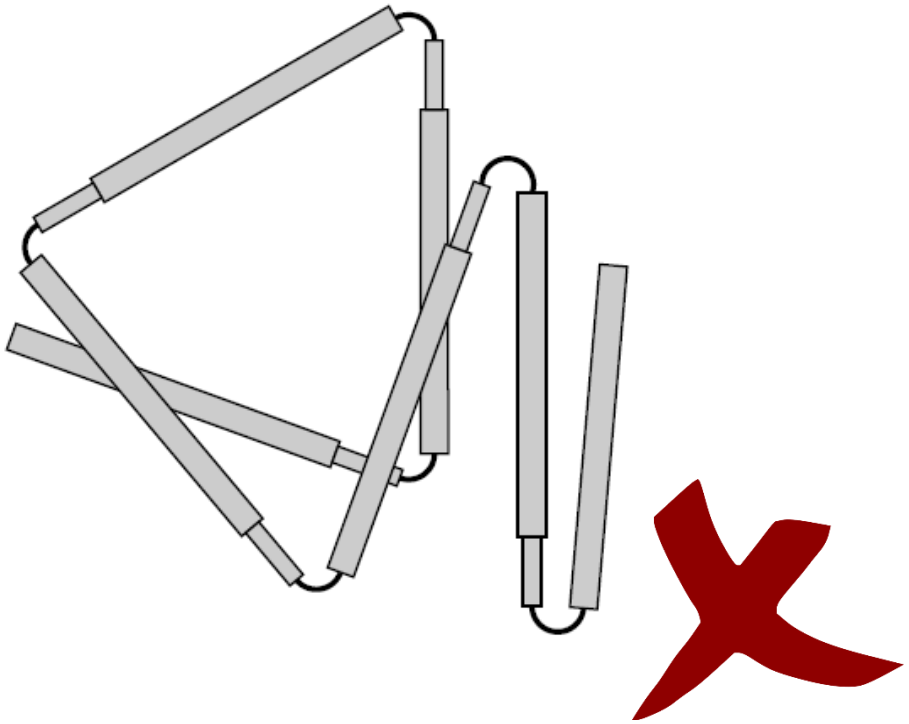
a) None
b) Basic knowledge
c) Foundational knowledge
d) Advanced
e) Expert

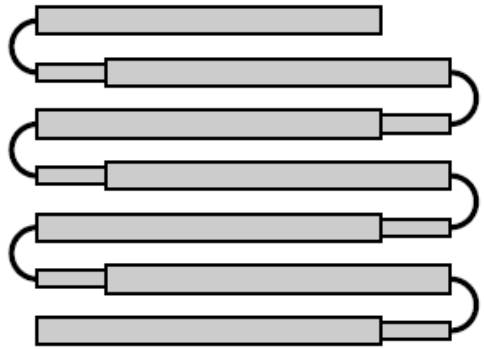
- The course Algorithms and Data Structures is about the efficiency of computer programs
- Really requires you can program...



- Programming exercises will be basic Java

Tent pole problem





Algorithms and Data Structures

Puzzle, SelectionSort


TV2
500
JUL I VALHAL
2005

”Lokes Høj”

- 64 pieces
- Score = 500 – # of swaps
- High score 450, i.e., 50 swaps

How do you achieve a low number of swaps – luck or cleverness ?

Optimal number of swaps ?

a) 5

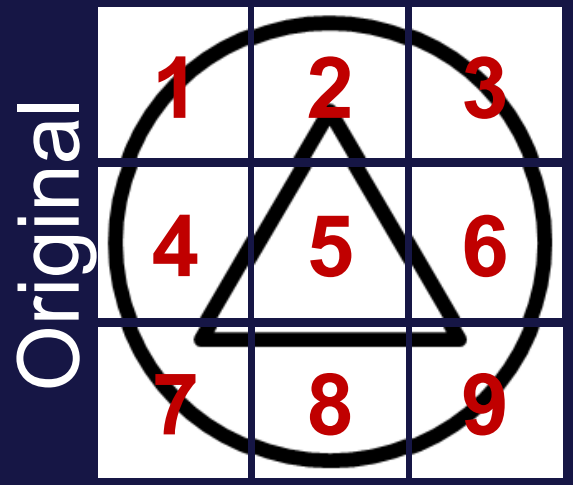
b) 6

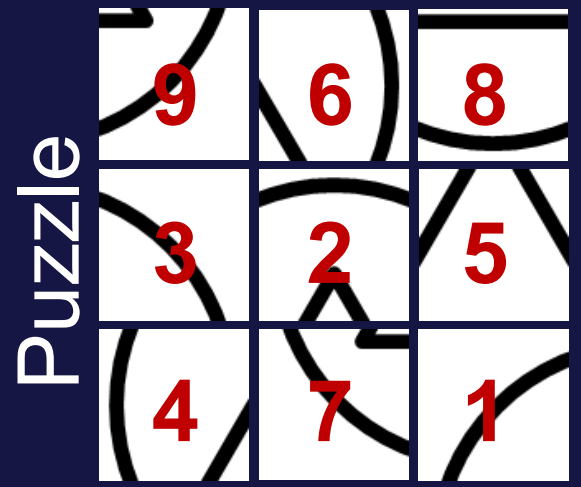
c) 7

d) 8

e) 9

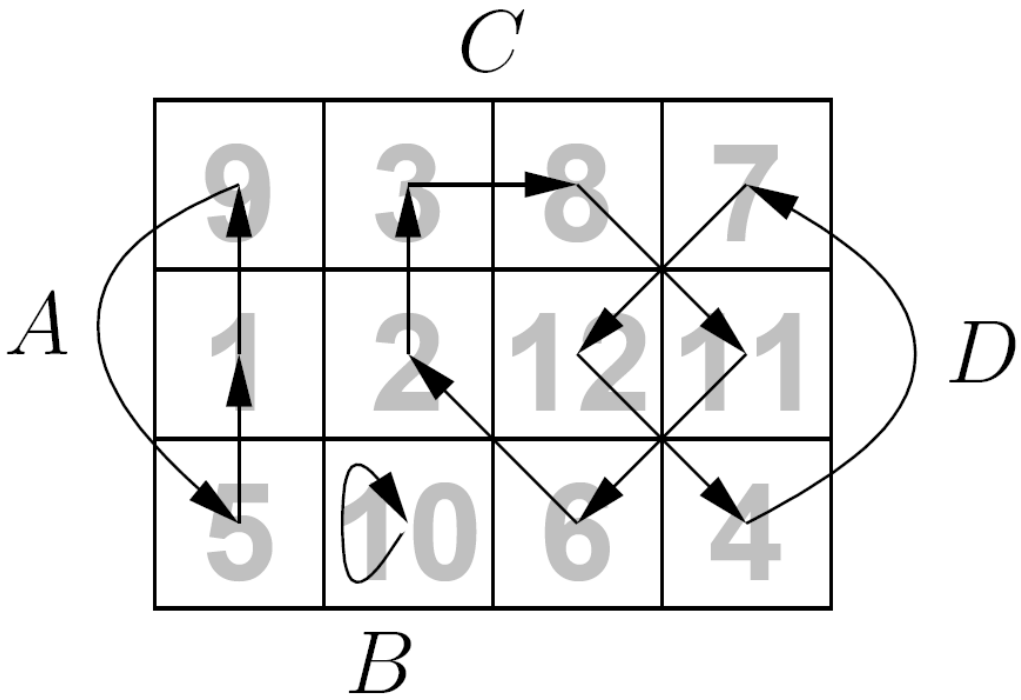
f) Don't know





One solution: **1-9 2-6 5-6 8-3 4-8 8-7**

Cycles (Permutations)

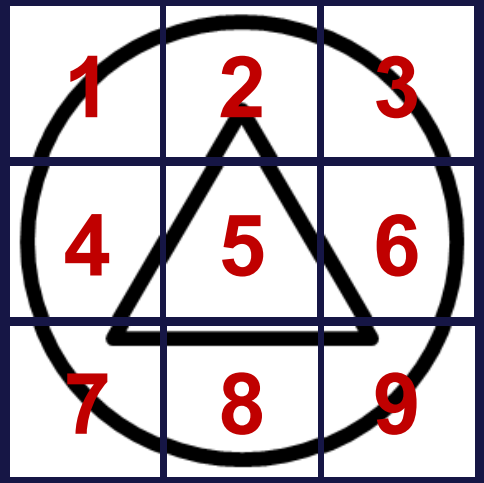


Each arrow points to the piece's correct position

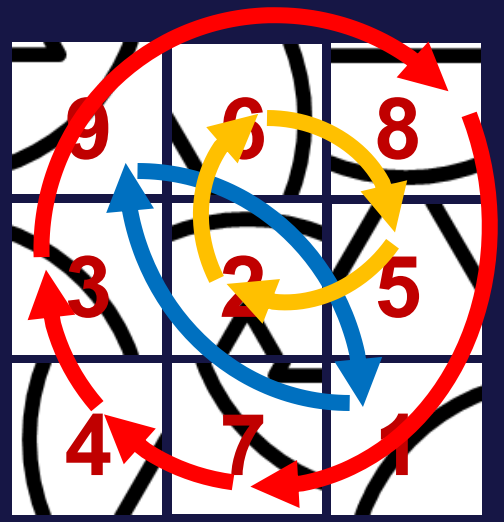
The arrows define a set of cycles (e.g. cycles A,B,C,D)

Optimal number of swaps ?

Original

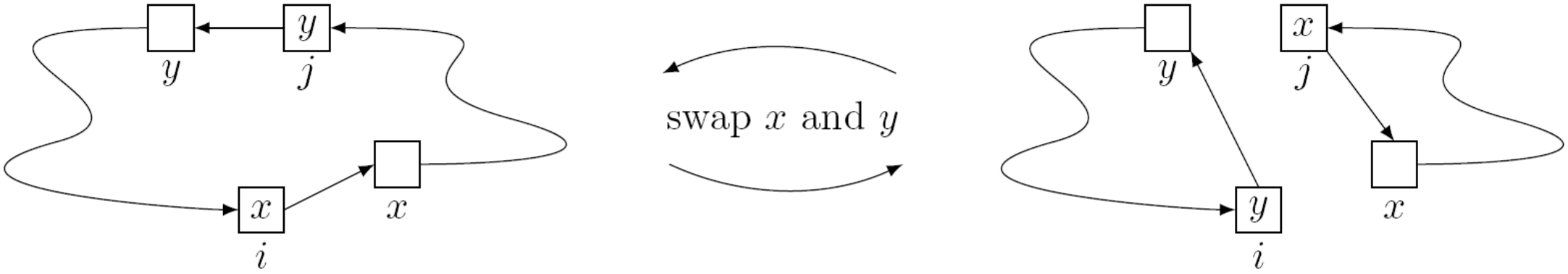


Puzzle



One solution: 1-9 2-6 5-6 8-3 4-8 8-7

Swaps and cycles



Lemma

- Swapping two pieces in the **same cycle** increases the number of cycles by one.
- Swapping two pieces in two **distinct cycle** decreases the number of cycles by one.

Lemma
When all n pieces are placed correctly there are exactly n cycles.

Lemma
To solve a puzzle with n pieces and initially k cycles requires $\geq n - k$ swaps.

We have proved a lower bound for

ALL algorithms solving the problem

A (greedy) algorithm

Algorithm PUZZLE

1 **while** there exists a misplaced piece x **do**
2 Let y be the piece at x's correct position
3 swap x and y

Lemma
The algorithm never swaps correctly placed pieces.

Lemma
The algorithm performs $\leq n - 1$ swaps

Lemma
To solve a puzzle with n pieces and initially k cycles the algorithm performs exactly $n - k$ swaps.

We have proved an upper bound for a specific algorithm

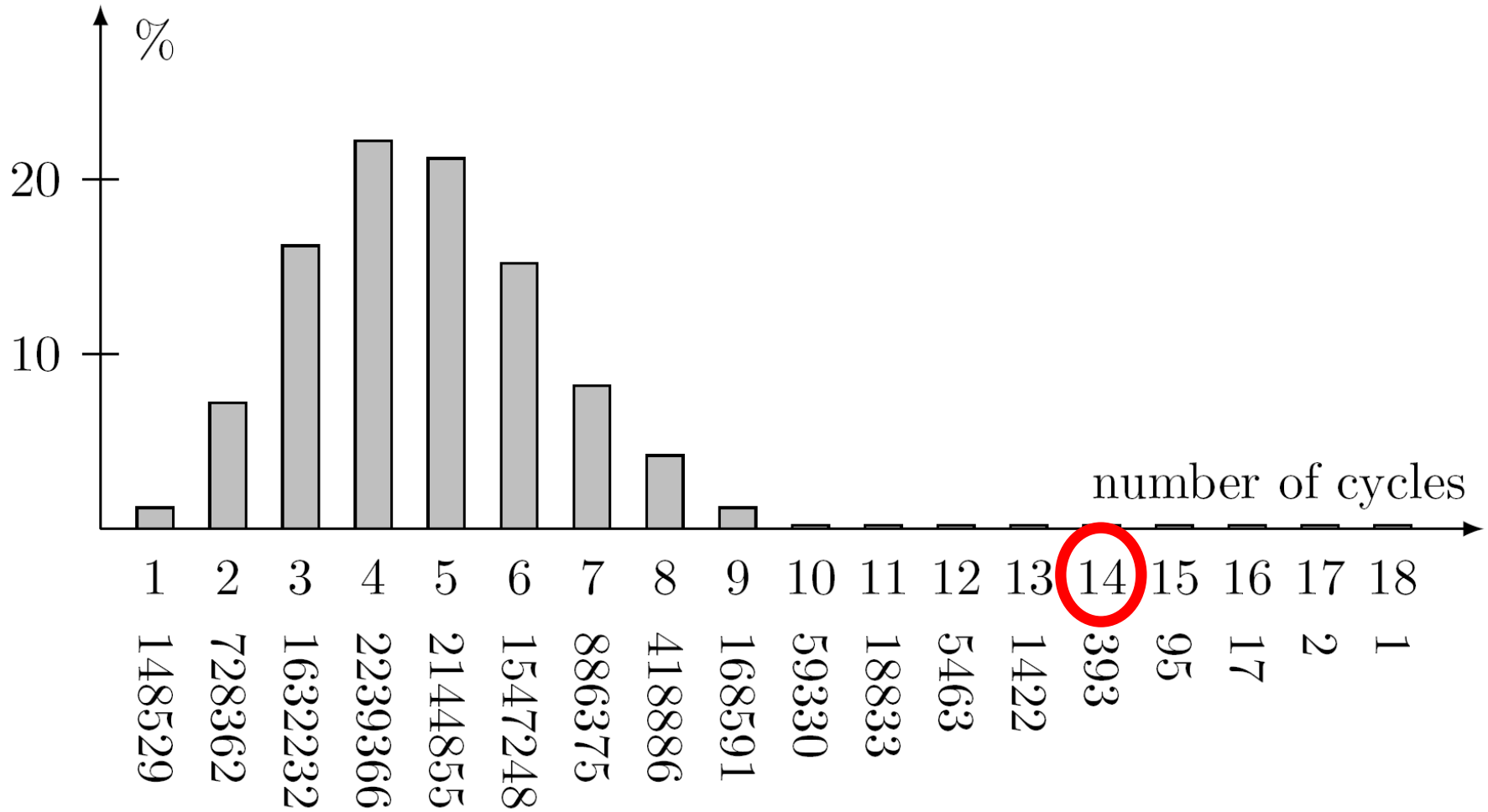
The algorithm is optimal since the number of swaps is best possible (the proved lower and upper bounds are identical)

Theorem

To solve a puzzle with n pieces and initially k cycles requires exactly $n - k$ swaps

Distribution of number of cycles

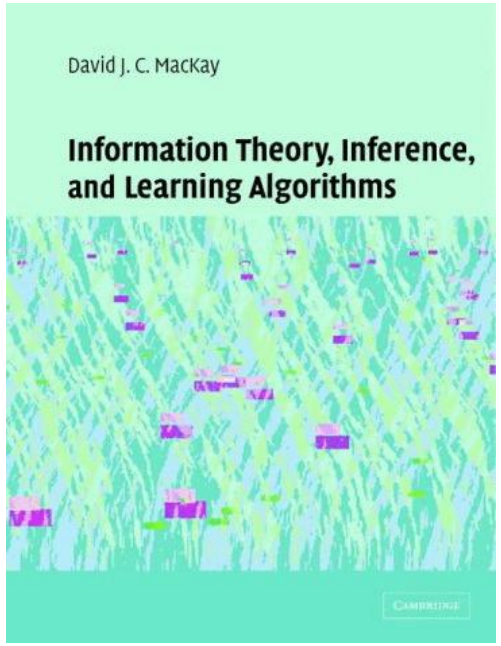
n = 64, 10.000.000 permutations



Algorithmic insights…

- **Mathematical insights** (cycles)
- **Resource usage** (number of swaps)
- **Lower bound** (≥ n - k swaps)
- **Greedy algorithm**
- **Analyzed algorithm** (≤ n - k swaps)
- **Optimal algorithm** (argued best possible)
- **Input dependent resource usage**
  (size and number of cycles in input)

Random permutations…



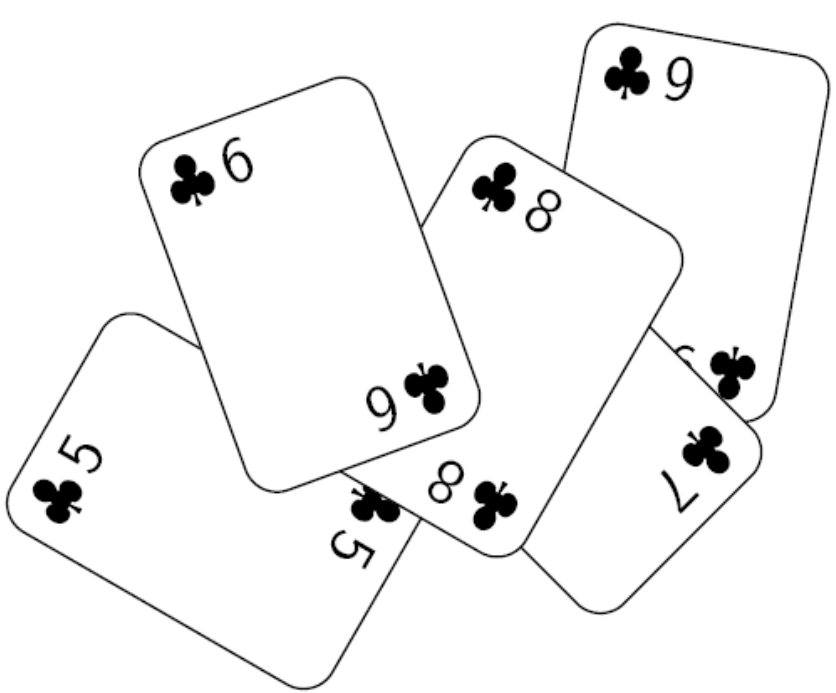
More information on random permutations can be found in David J.C. MacKay, appendix to *Information Theory, Inference, and Learning Algorithms*, on

"Random Permutations“, 4 pages.

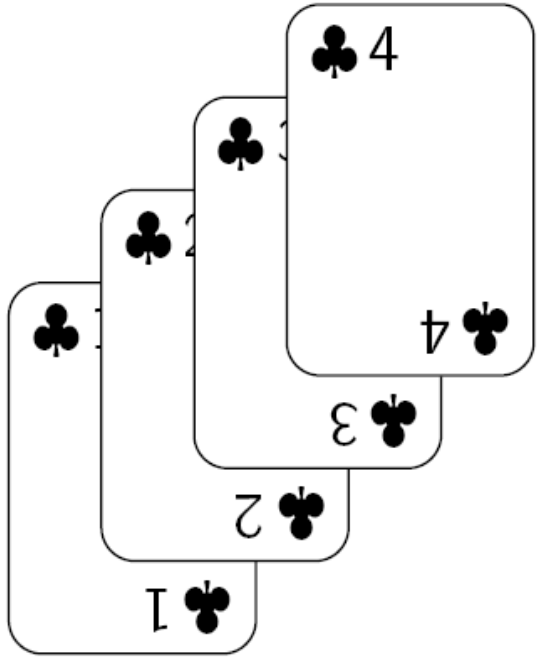
www.inference.phy.cam.ac.uk/mackay/itila/cycles.pdf

SelectionSort

Move smallest card



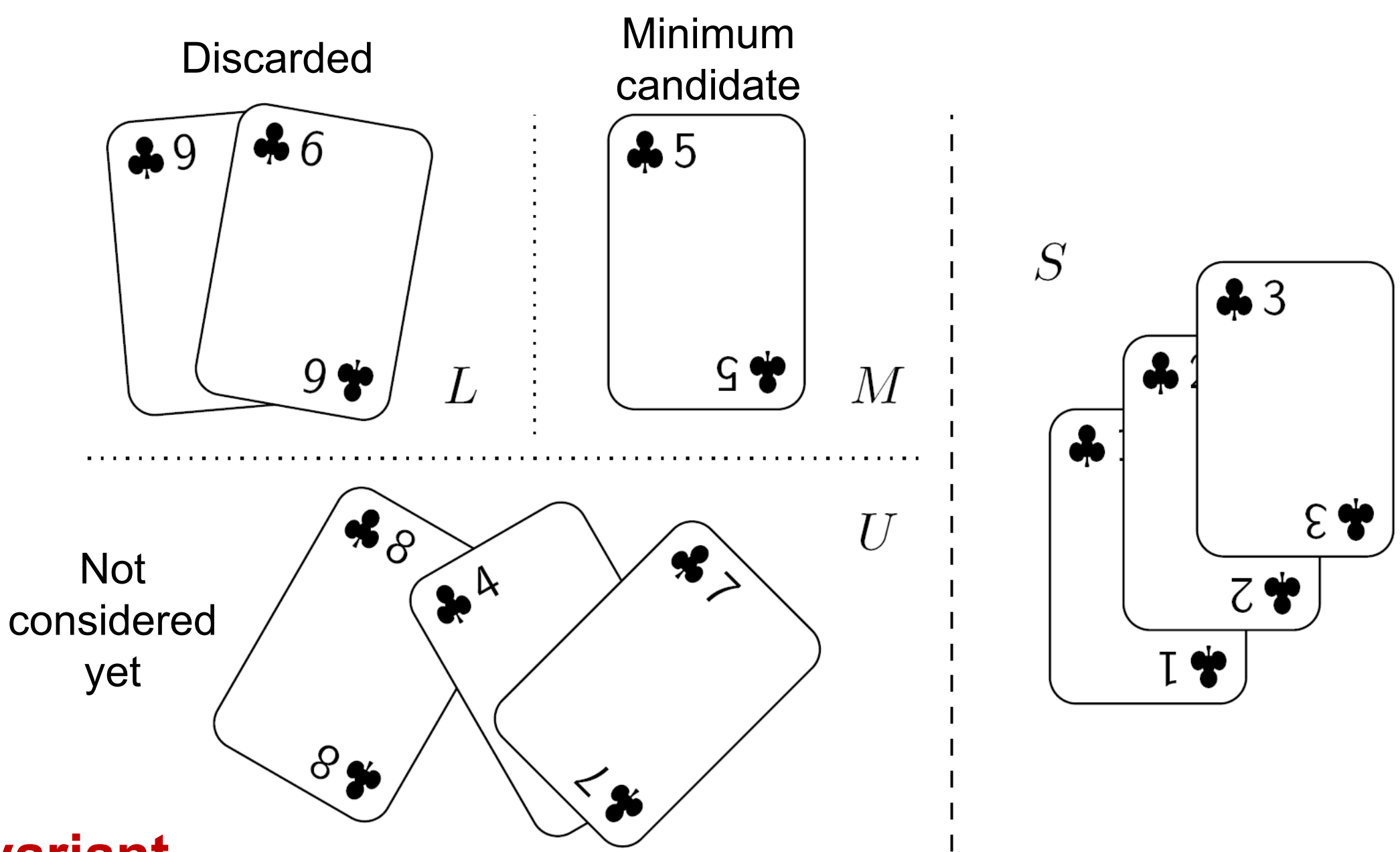
C | S



Unsorted

Sorted

Invariant S sorted and $C \geq S$



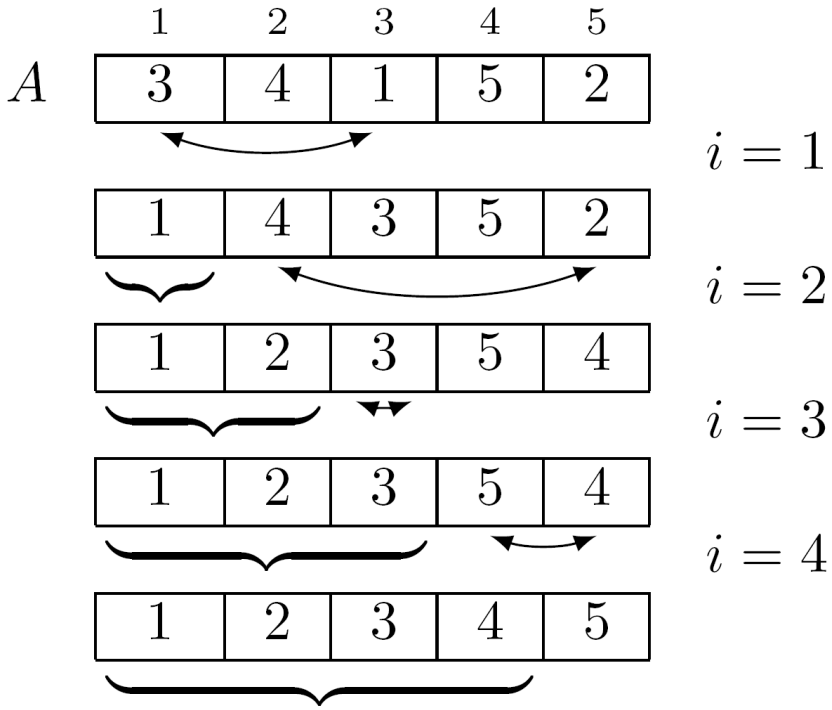
Invariant

S sorted, $U \cup L \cup M \geq S$, $L \geq M$, $|M| \leq 1$, and $|L| \geq 1 \Rightarrow |M| = 1$

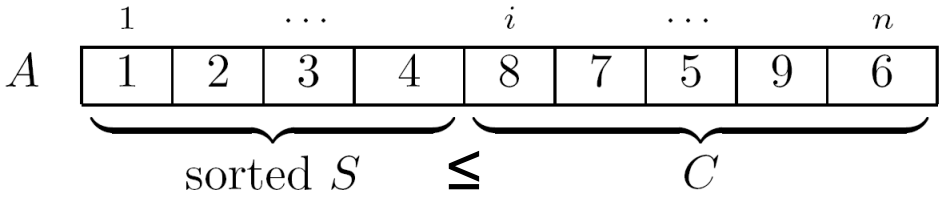
Implicit SelectionSort

Algorithm SelectionSortAbstract(A)

```
1  for i = 1 to |A| − 1 do
2      swap A[i] and minimum of A[i..|A|]
```



Invariant



Algorithm SelectionSort(A)

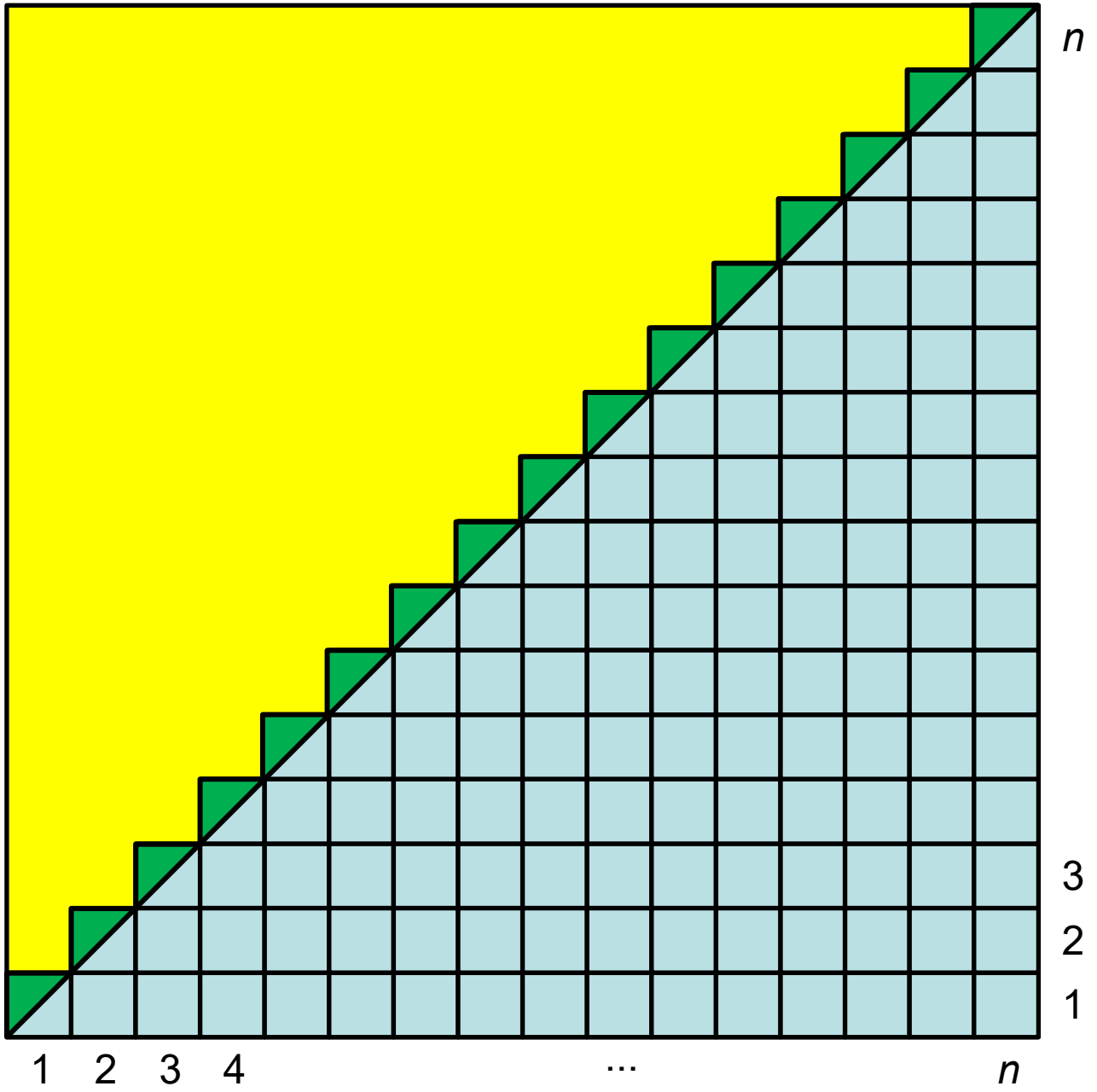
```
1  for i = 1 to |A| − 1 do
2      k = i
3      for j = i + 1 to |A| do
4          # A[k] = min A[i..j − 1]
5          if A[j] < A[k] then
6              k = j
7      # A[k] = min A[i..|A|]
8      tmp = A[i]
9      A[i] = A[k]
10     A[k] = tmp
```

Analysis SelectionSort

Lemma
To find the smallest card among k cards requires $k - 1$ comparisons

Lemma
SelectionSort on n cards makes $(n - 1) + (n - 2) + \cdots + 1 = n \cdot (n - 1) / 2$ comparisons

$$1 + 2 + \cdots + n = n^2/2 + n/2 = \frac{n(n+1)}{2}$$



Algorithms and Data Structures

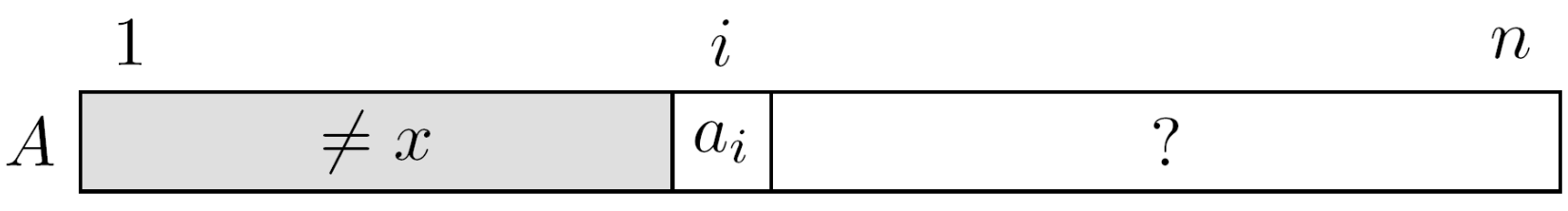
Binary and linear search, logarithms,
longest increasing subsequence, Erdös Szekeres theorem

Searching an unsorted list

24	11	3	7	32	47	5	2	89	12

Find x

Visual invariant



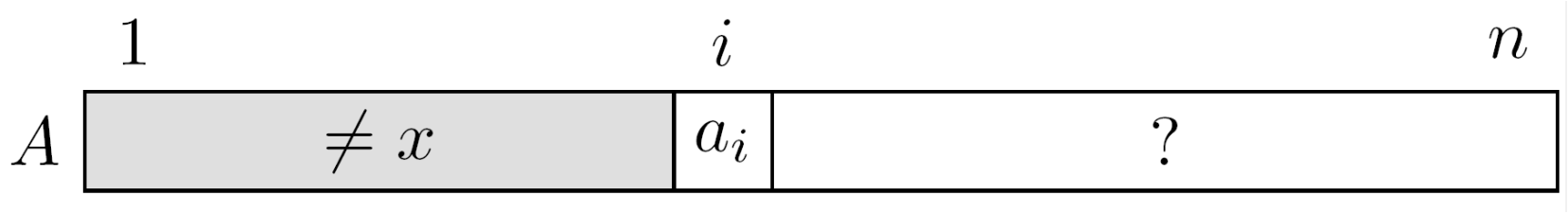
Formal invariant

$$1 \leq i \leq n+1 \ \wedge \ a_j \neq x \text{ for all } 1 \leq j < i$$

at most *n* comparisons

Searching an unsorted list

Visual invariant



Algorithm LINEARSEARCH(A, x)
Input Array $A[1..n]$ and search key x
Output Index i where $A[i] = x$; -1 if $x \notin A$

```
1  i = 1
2  while i ≤ |A| do
3      if A[i] = x then
4          return i
5      i = i + 1
6  return −1
```

Searching a sorted list

3	7	9	11	13	27	33	37	42	89

Find x

Visual invariant

Formal invariant

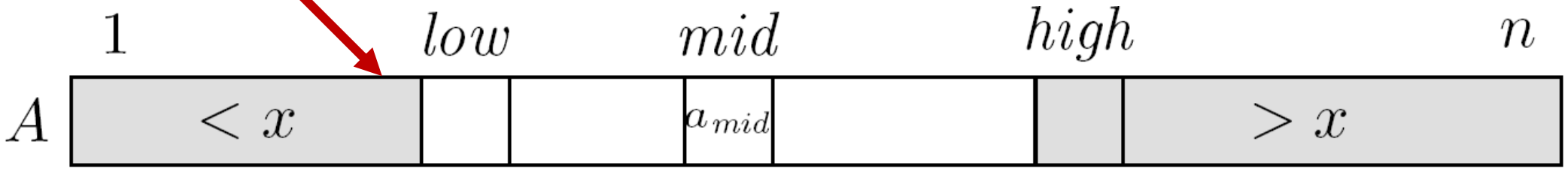
$$(1 \leq low \leq high \leq n+1) \ \wedge$$
$$(a_j < x \text{ for all } 1 \leq j < low) \ \wedge$$
$$(x < a_j \text{ for all } high \leq j \leq n)$$

Algorithm BINARYSEARCH(A, x)
Input Sorted array $A[1..n]$ and search key x
Output Index i where $A[i] = x$; -1 if $x \notin A$

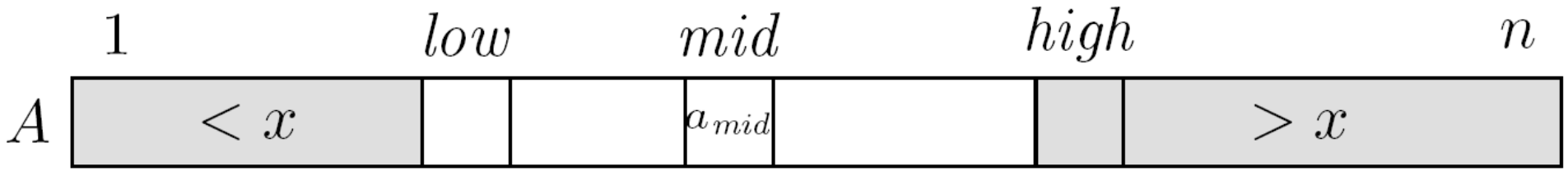
```
1   low = 1
2   high = n + 1
3   while low < high do
4       mid = ⌊(low + high)/2⌋
5       if x = A[mid] then
6           return mid
7       else if x < A[mid] then
8           high = mid
9       else if x > A[mid] then
10          low = mid + 1
11  return −1
```

Exercise
What happens if floor is replaced by ceil?

Exercise
Modify algorithm to let *low* point here



Binary search – number of comparisons



Number of candidates after one comparison ≤ : $n \to \lfloor n/2 \rfloor$

$$n \to \lfloor n/2 \rfloor \to \lfloor \lfloor n/2 \rfloor /2 \rfloor \to \lfloor \lfloor \lfloor n/2 \rfloor /2 \rfloor /2 \rfloor \to \cdots \to 0$$

After k comparisons ≤ $n/2^k$ candidates

done the latest when $n/2^k < 1 \iff n < 2^k \iff \log_2 n < k \iff 1 + \lfloor \log_2 n \rfloor \leq k$

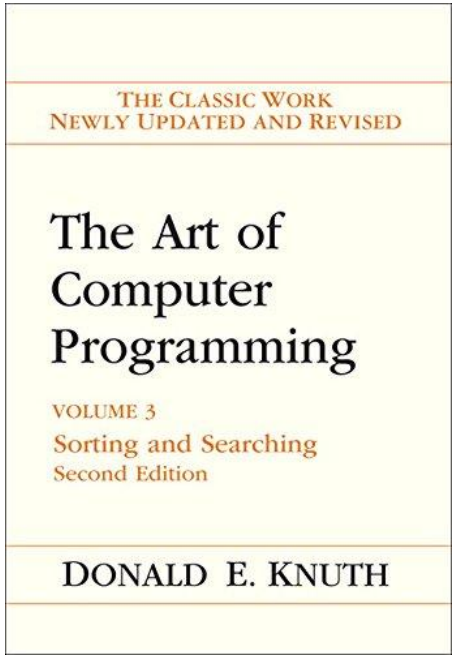
at most 1 + $\lfloor \log_2 n \rfloor$ comparisons

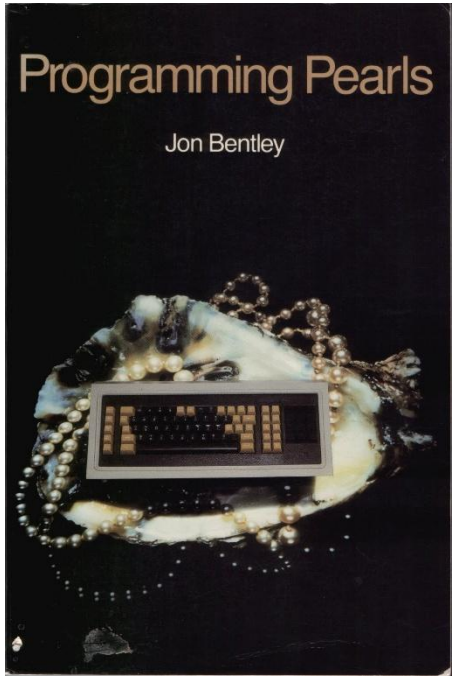
Binary search in the literature

- According to Knuth first discussed in the literature
  - John Mauchly i [*Theory and techniqeus fór the design of electronic digital computers*, ed. G. W. Patterson, 3 (1946), 22.8-22.9] for $n = 2^k - 1$.
  - H. Bottenbruch [Structure and Use of ALGOL 60, *Journal of the ACM*, 9 (1962), 214], for all n

- Bentley's experience from Bell Labs and IBM in the beginning of the 80s, where he asked peopler to implement binary search:

  …*given ample time, only about ten percent of professional programmers were able to get this small program right...*





Donald E. Knuth, *The Art of Computer Programming: Volume 3, Sorting and Searching*, Chapter 6.2.1, 1973

Jon Bentley, *Programming Pearls*, Chapter 4.1, 1986

43. *Program for Binary Search*

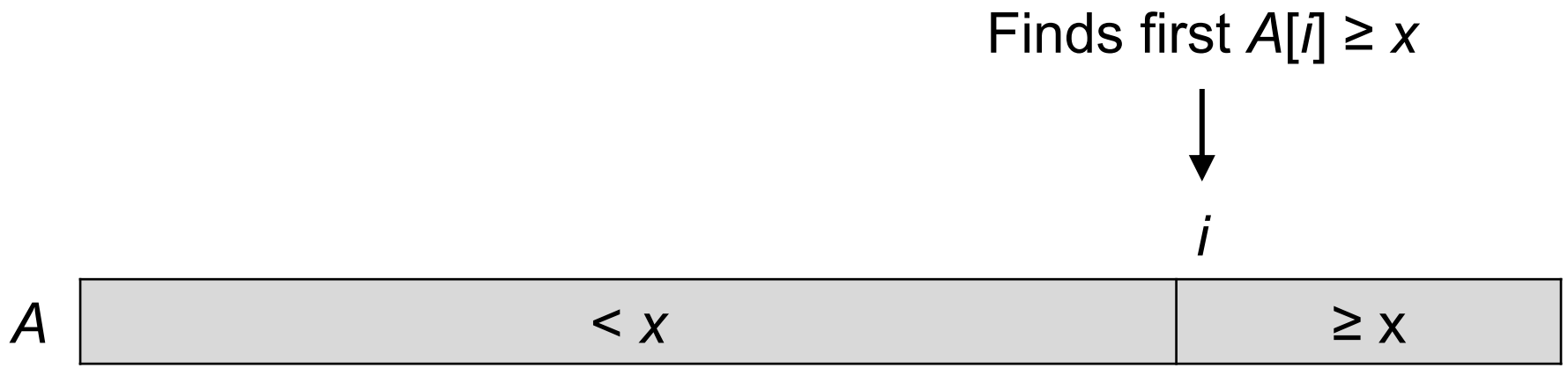
The following procedure assumes that the elements of an array a are arranged in descending order, that is, $a[1] \geqq a[2] \geqq a[3] \geqq \cdots$. Given a number b and a subscript j such that $a[1] \geqq b > a[j]$ the program determines in $1 + \log_2 j$ comparisons a subscript p for which $a[p] \geqq b > a[p + 1]$.

```
procedure binary search (a, b, j, p);
   value j, b;  integer j, p;  real b;  real array a;
begin integer p1;
                  p := 1;
test for end:     if j − p = 1 then go to M;
                  p1 := (j + p) ÷ 2;
                  if a[p1] ≧ b then p := p1 else j := p1;
                  go to test for end;
              M:
```

H. Bottenbruch, *Structure and Use of ALGOL 60*, Journal of the ACM, 1962, doi: 10.1145/321119.321120

Binary search in standard libaries

- Java `java.util.Collections.binarySearch`
- C++ `std::lower_bound`
- Python `bisect.bisect_left`



Searching a sorted list – lower bound

For any algorithm, an *adversary* can answer the comparison such that number of candidates is at least

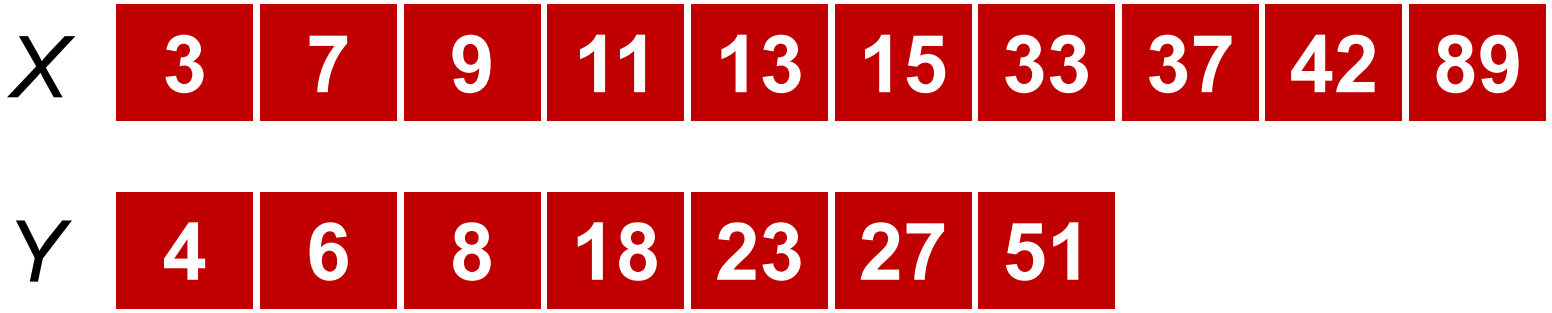
$$n \to \lfloor n/2 \rfloor \to \lfloor \lfloor n/2 \rfloor /2 \rfloor \to \lfloor \lfloor \lfloor n/2 \rfloor /2 \rfloor /2 \rfloor \to \cdots \to 0$$

$n \geq 2^{\lfloor \log_2 n \rfloor}$ $\Rightarrow$ after k comparisons $\geq 2^{\lfloor \log_2 n \rfloor - k}$ candidates

$\Rightarrow$ after $\lfloor \log_2 n \rfloor$ comparisons at least 1 candidate remains

at least 1 + $\lfloor \log_2 n \rfloor$ comparisons (in the worst case)

Problem 1.9*

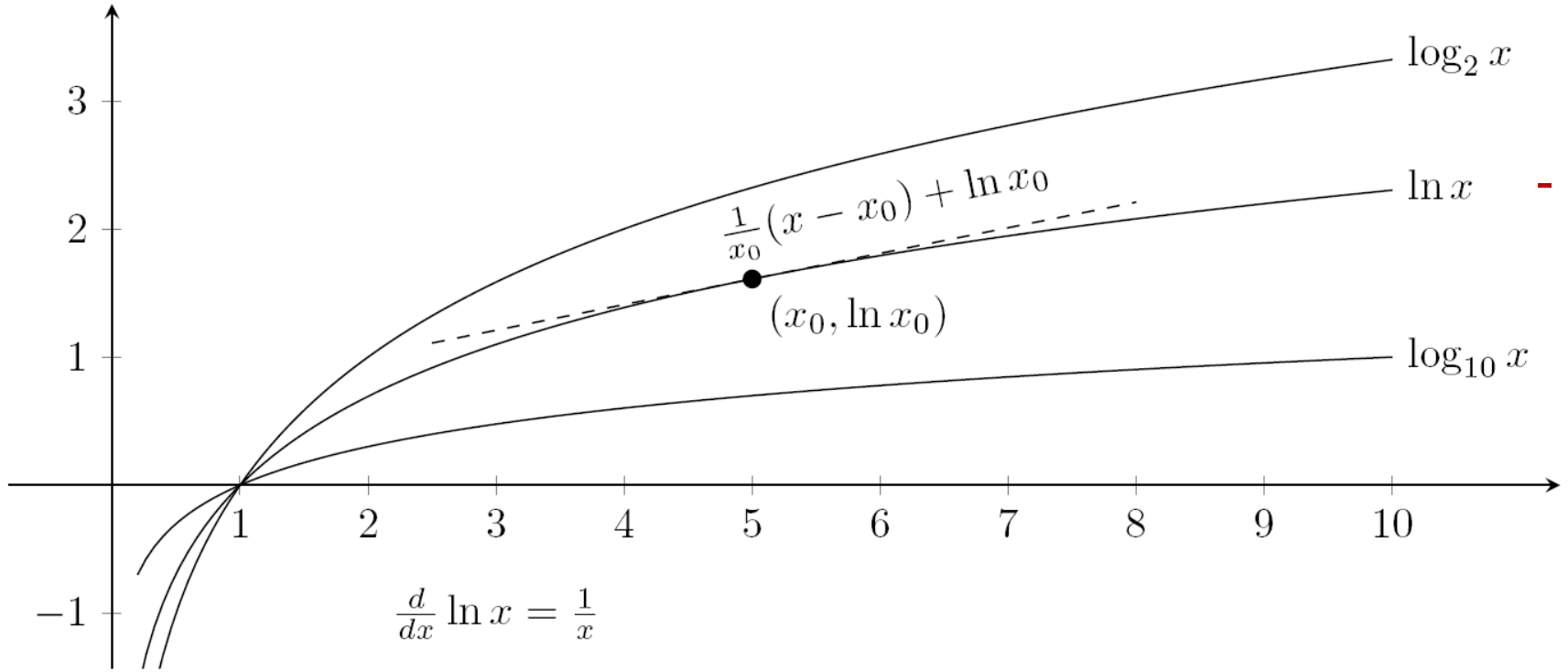


Given X and Y sorted, find the r^{th} smallest in $X \cup Y$

(the 9^{th} smallest element is 15)

Logarithms

Definition $y = \log_b x \Leftrightarrow b^y = x$



- natural logarithm

Logarithms – rules of arithmetic

$$\begin{aligned}
\log_b(x \cdot y) &= \log_b x + \log_b y \\
\log_b(x/y) &= \log_b x - \log_b y \\
\log_b(x^p) &= p \cdot \log_b x \\
\log_b x &= \frac{\log_a x}{\log_a b} \\
\log_b(b^p) &= p \\
\frac{d}{dx} \ln x &= \frac{1}{x} \\
\ln y &= \int_1^y \frac{1}{x} dx \\
H_n - \ln n &\to \gamma \quad \text{for} \quad n \to \infty
\end{aligned}$$

Harmonic number $H_n = \sum_{i=1}^{n} \frac{1}{i}$

Euler-Mascheroni constant $\gamma = 0.577215664901\ldots$

An algorithmic example (using binary search)

Patience Diff
&
Longest Increasing Subsequences

Shell command: diff

A.c

```
#include <stdio.h>

// Frobs foo heartily
int frobnitz(int foo)
{
    int i;
    for(i = 0; i <
    {
        printf("You
        printf("%d\
    }
}

int fact(int n)
{
    if(n > 1)
    {
        return fact
    }
    return 1;
}

int main(int argc,
{
    frobnitz(fact(1
}
```

B.c

```
#include <stdio.h>

int fib(int n)
{
                                  fib(n-2);
                                  +)
                                  o);
                                  *argv)
```

```
$ diff A.c B.c
3,4c3
< // Frobs foo heartily
< int frobnitz(int foo)
---
> int fib(int n)
6,7c5
<     int i;
<     for(i = 0; i < 10; i++)
---
>     if(n > 2)
9,10c7
<         printf("Your answer is: ");
<         printf("%d\n", foo);
---
>         return fib(n-1) + fib(n-2);
11a9
>     return 1;
14c12,13
< int fact(int n)
---
> // Frobs foo heartily
```

wikipedia.org/wiki/Diff

11 removals 10 additions

```
1. #include <stdio.h>
2.
3. // Frobs foo heartily
4. int frobnitz(int foo)
5. {
6.     int i;
7.     for(i = 0; i < 10; i++)
8.     {
9.         printf("Your answer is: ");
10.        printf("%d\n", foo);
11.    }
12. }
13.
14. int fact(int n)
15. {
16.     if(n > 1)
17.     {
18.         return fact(n-1) * n;
19.     }
20.     return 1;
21. }
22.
23. int main(int argc, char **argv)
24. {
25.     frobnitz(fact(10));
26. }
```

```
1. #include <stdio.h>
2.
3. int fib(int n)
4. {
5.     if(n > 2)
6.     {
7.         return fib(n-1) + fib(n-2);
8.     }
9.     return 1;
10. }
11.
12. // Frobs foo heartily
13. int frobnitz(int foo)
14. {
15.     int i;
16.     for(i = 0; i < 10; i++)
17.     {
18.         printf("%d\n", foo);
19.     }
20. }
21.
22. int main(int argc, char **argv)
23. {
24.     frobnitz(fib(10));
25. }
```

Text Compare!

Email this comparison

```
#include <stdio.h>

// Frobs foo heartily
int frobnitz(int foo)
{
    int i;
    for(i = 0; i < 10; i++)
    {
        printf("Your answer is: ");
        printf("%d\n", foo);
    }
}

int fact(int n)
{
    if(n > 1)
    {
        return fact(n-1) * n;
    }
    return 1;
}

int main(int argc, char **argv)
{
    frobnitz(fact(10));
}
```

```
#include <stdio.h>

int fib(int n)
{
    if(n > 2)
    {
        return fib(n-1) + fib(n-2);
    }
    return 1;
}

// Frobs foo heartily
int frobnitz(int foo)
{
    int i;
    for(i = 0; i < 10; i++)
    {

        printf("%d\n", foo);
    }
}

int main(int argc, char **argv)
{
    frobnitz(fib(10));
}
```

Edit texts ... | Switch texts | Compare! | Clear all

File Left Right View Options Mergely Social

```
#include <stdio.h>

// Frobs foo heartily
int frobnitz(int foo)
{
    int i;
    for(i = 0; i < 10; i++)
    {
        printf("Your answer is: ");
        printf("%d\n", foo);
    }
}

int fact(int n)
{
    if(n > 1)
    {
        return fact(n-1) * n;
    }
    return 1;
}

int main(int argc, char **argv)
{
    frobnitz(fact(10));
}
```

```
#include <stdio.h>

int fib(int n)
{
    if(n > 2)
    {
        return fib(n-1) + fib(n-2);
    }
    return 1;
}

// Frobs foo heartily
int frobnitz(int foo)
{
    int i;
    for(i = 0; i < 10; i++)
    {
        printf("%d\n", foo);
    }
}

int main(int argc, char **argv)
{
    frobnitz(fib(10));
}
```

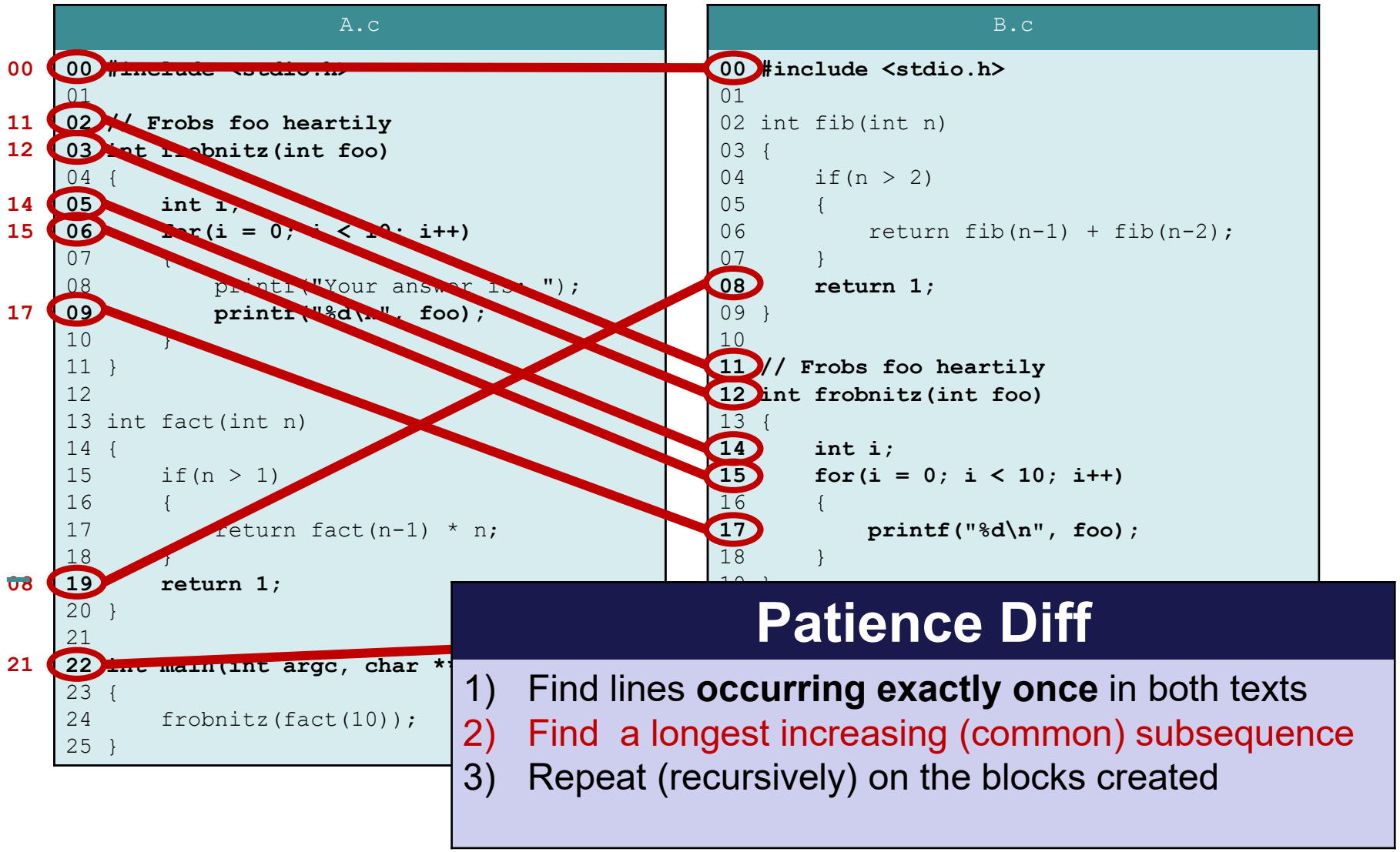
Patience Diff

(Bram Cohen, inventor of BitTorrent)

- Tries to create **readable and meaningful output** – opposed to minimal output
- Used in the Bazaar version control system

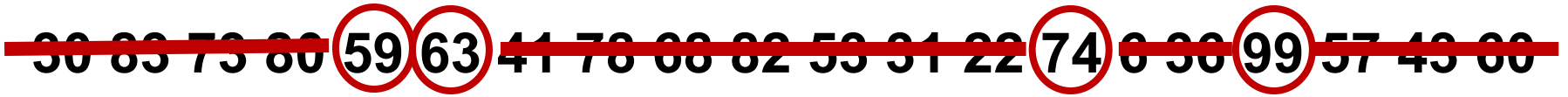
Finding a smallest possible edit sequence is covered later in the course (topic Dynamic Programming)

alfedenzo.livejournal.com: *Patience Diff, a brief summary*


A.c
B.c
00 00 #include <stdio.h>
01
11 02 // Frobs foo heartily
12 03 int frobnitz(int foo)
04 {
14 05 int i;
15 06 for(i = 0; i < 10; i++)
07 {
08 printf("Your answer is: ");
17 09 printf("%d\n", foo);
10 }
11 }
12
13 int fact(int n)
14 {
15 if(n > 1)
16 {
17 return fact(n-1) * n;
18 }
08 19 return 1;
20 }
21
21 22 int main(int argc, char *
23 {
24 frobnitz(fact(10));
25 }
00 #include <stdio.h>
01
02 int fib(int n)
03 {
04 if(n > 2)
05 {
06 return fib(n-1) + fib(n-2);
07 }
08 return 1;
09 }
10
11 // Frobs foo heartily
12 int frobnitz(int foo)
13 {
14 int i;
15 for(i = 0; i < 10; i++)
16 {
17 printf("%d\n", foo);
18 }
Patience Diff
1) Find lines occurring exactly once in both texts
2) Find a longest increasing (common) subsequence
3) Repeat (recursively) on the blocks created

Longest increasing subsequence

(Problem 1.2)

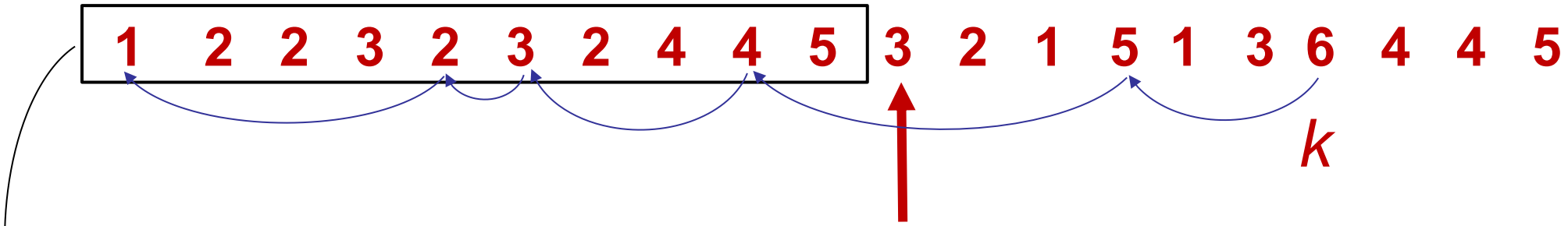


Problem

Erase as few numbers as possible,
so that the remaining number are in increasing order

Longest increasing subsequence
(Problem 1.2)

30 83 73 80 59 63 41 78 68 82 53 31 22 74 6 36 99 57 43 60



longest increasing subsequence ending with 53

Length	Smallest last element
1	30
2	41
3	~~63~~ 53
4	68
5	82

= binary search

$\leq n \cdot (1 + \lfloor \log_2 k \rfloor)$ comparisons

Theorem (Erdős and Szekeres, 1935)

Any sequence of n numbers has an increasing or decreasing subsequence of length at least $\lceil \sqrt{n} \rceil$.

3 1 4 17 6 42 10 8 13 11 28 (increasing)

24 3 12 16 7 14 26 8 20 2 1 (decreasing)

Paul Erdős, George Szekeres, *A combinatorial problem in geometry*, Compositio Mathematica, 1935, doi: 10.1007/978-0-8176-4842-8_3

Theorem (Erdős and Szekeres, 1935)

30 83 73 80 59 63 41 78 68 82 53 31 22 74 6 36 99 57 43 60

1 2 2 3 2 3 2 4 4 5 3 2 1 5 1 3 6 4 4 5

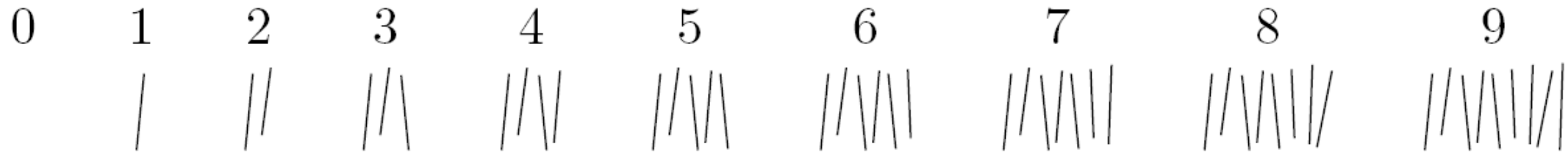
Length	Smallest last element
1	30 22 6
2	83 73 59 41 31
3	80 63 53 36
4	78 68 57 43
5	82 74 60
6	99

Either many ($\geq \sqrt{n}$) rows (long increasing subsequence) or
at least one long ($\geq \sqrt{n}$) row (long decreasing subsequence)

Algorithms and Data Structures

Integer arithmetic: Binary and decimal numbers, addition, subtraction, multiplication, division

Digits



For every small integer we have a special symbol

Base-10 system (10 = 9 + 1)

Value of $d_{n-1}d_{n-2}\cdots d_1 d_0$ where $d_i \in \{0,1,2,3,4,5,6,7,8,9\}$

$$\sum_{i=0}^{n-1} d_i \cdot 10^i = d_{n-1}\cdot 10^{n-1} + d_{n-2}\cdot 10^{n-2} + \cdots + d_1 \cdot 10^1 + d_0 \cdot 10^0$$

Example: $1849_{10} = 1\cdot 10^3 + 8\cdot 10^2 + 4\cdot 10^1 + 9\cdot 10^0$

Representation base *b*

Value of $d_{n-1}d_{n-2}\cdots d_1d_0$

$$\sum_{i=0}^{n-1} d_i \cdot b^i$$

Example: $1849_{10} = \underbrace{1}_{a}\underbrace{110}_{b}\underbrace{01}_{c}\underbrace{1001}_{d}{}_2 = 3471_8 = 739_{16}$

$\underbrace{1}_{x}\underbrace{110}_{y}\underbrace{01}_{z}\underbrace{1001}_{d}{}_2$

$1 \cdot 10^3 + 8 \cdot 10^2 + 4 \cdot 10^1 + 9 \cdot 10^0$

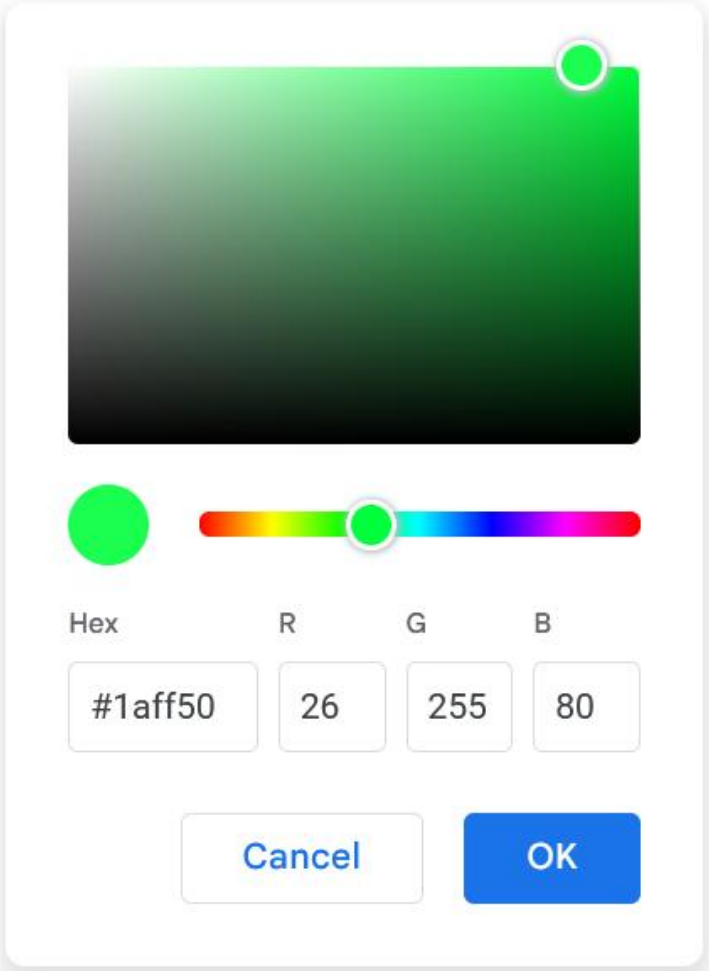
$2^{10} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^0$

$3 \cdot 8^3 + 4 \cdot 8^2 + 7 \cdot 8^1 + 1 \cdot 8^0$

$7 \cdot 16^2 + 3 \cdot 16^1 + 9 \cdot 16^0$

Decimal value	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Binary symbol	0	1														
Octal symbol	0	1	2	3	4	5	6	7								
Hexadecimal symbol	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

Color picker



docs.google.com

- RGB = **R**ed-**G**reen-**B**lue system
- A typical value consists of three values R, G and B, each value consisting of 8 bits, i.e., a value in the decimal range 0..255 and hexadecimal 00..FF
- R: $1A_{16} = 26_{10}$
- G: $FF_{16} = 255_{10}$
- B: $50_{16} = 80_{10}$

101010_2 in base 10 ?

5 4 3 2 1 0

a) 14_{10}

b) 26_{10}

c) 37_{10}

d) $42_{10} = 2^5 + 2^3 + 2^1 = 32 + 8 + 2$

e) 84_{10}

f) Don't know

42_{10} in base 5 ?

a) 42_5

b) 82_5

c) 132_5 $= 1 \cdot 5^2 + 3 \cdot 5^1 + 2 \cdot 5^0$

$= 25 + 15 + 2$

d) 134_5

e) 210_5

f) Don't know

Addition

+	0	1	2	3	4	5	6	7	8	9
0	0	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9	10
2	2	3	4	5	6	7	8	9	10	11
3	3	4	5	6	7	8	9	10	11	12
4	4	5	6	7	8	9	10	11	12	13
5	5	6	7	8	9	10	11	12	13	14
6	6	7	8	9	10	11	12	13	14	15
7	7	8	9	10	11	12	13	14	15	16
8	8	9	10	11	12	13	14	15	16	17
9	9	10	11	12	13	14	15	16	17	18

Addition – school method

$$
\begin{array}{rrrr}
 & 1 & 1 & \\
 & 8 & 4 & 3 \\
+ & 5 & 7 & 2 \\
\hline
1 & 4 & 1 & 5
\end{array}
$$

$$
\begin{aligned}
843 + 572 &= (\mathbf{8} \cdot 10^2 + \mathbf{4} \cdot 10^1 + \mathbf{3} \cdot 10^0) + (\mathbf{5} \cdot 10^2 + \mathbf{7} \cdot 10^1 + \mathbf{2} \cdot 10^0) \\
&= (\mathbf{8} + \mathbf{5}) \cdot 10^2 + (\mathbf{4} + \mathbf{7}) \cdot 10^1 + (\mathbf{3} + \mathbf{2}) \cdot 10^0 \\
&= (8 + 5) \cdot 10^2 + (4 + 7) \cdot 10^1 + \mathbf{5} \cdot 10^0 \\
&= (8 + 5) \cdot 10^2 + \mathbf{11} \cdot 10^1 + 5 \cdot 10^0 \\
&= (8 + 5) \cdot 10^2 + (\mathbf{1} \cdot 10 + \mathbf{1}) \cdot 10^1 + 5 \cdot 10^0 \\
&= (8 + 5) \cdot 10^2 + (\mathbf{1} \cdot 10^2 + \mathbf{1} \cdot 10^1) + 5 \cdot 10^0 \\
&= ((8 + 5) + \mathbf{1}) \cdot 10^2 + \mathbf{1} \cdot 10^1 + 5 \cdot 10^0 \\
&= (\mathbf{13} + 1) \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0 \\
&= \mathbf{14} \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0 \\
&= (\mathbf{1} \cdot 10 + \mathbf{4}) \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0 \\
&= \mathbf{1} \cdot 10^3 + \mathbf{4} \cdot 10^2 + 1 \cdot 10^1 + 5 \cdot 10^0 \\
&= 1415
\end{aligned}
$$

Must (recursively) add numbers with multiple digits

Binary addition

+	0	1
0	0	1
1	1	10

$$
\begin{array}{rr}
 & {\scriptstyle 1\ 1\ 1} \\
 & 1\ 1\ 1\ 0_{\,2} \\
+ & 1\ 0\ 1\ 1_{\,2} \\
\hline
 & 1\ 1\ 0\ 0\ 1_{\,2}
\end{array}
$$

School method

Addition of several numbers

$$\begin{array}{rcccc} & {\scriptstyle 3} & {\scriptstyle 3} & {\scriptstyle 2} & \\ & & 2 & 7 & 7 \\ + & & 9 & 5 & 5 \\ + & & 7 & 7 & 6 \\ + & & 4 & 9 & 1 \\ + & 1 & 5 & 2 & 4 \\ + & & 3 & 3 & 6 \\ \hline & 4 & 3 & 5 & 9 \end{array}$$

Generalized school method

$$\begin{array}{rcccc} & {\scriptstyle 1} & {\scriptstyle 11} & {\scriptstyle 10} & \\ & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ + & & & 9 & 9 \\ \hline & 1 & 1 & 8 & 8 \end{array}$$

...carry can have multiple digits

$$\begin{array}{rcccc} & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & & 2 & 7 & 7 \\ + & & 9 & 5 & 5 \\ \hline & {\scriptstyle 1} & {\scriptstyle 1} & & \\ & 1 & 2 & 3 & 2 \\ + & & 7 & 7 & 6 \\ \hline & 2 & 0 & 0 & 8 \\ + & & 4 & 9 & 1 \\ \hline & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & \\ & 2 & 4 & 9 & 9 \\ + & 1 & 5 & 2 & 4 \\ \hline & 4 & 0 & 2 & 3 \\ + & & 3 & 3 & 6 \\ \hline & 4 & 3 & 5 & 9 \end{array}$$

Repeated addition of two numbers

Subtraction – school method

$$\begin{array}{r} \overset{-1}{5}\ \overset{-1}{3}\ 4\ 7 \\ -\ 1\ 8\ 6\ 3 \\ \hline 3\ 4\ 8\ 4 \end{array}$$

4 – 6 = -2 = -10 + 8

$$\begin{array}{r} \overset{-1}{1}\ \overset{-1}{0}\ 0\ \overset{-1}{1}\ 0\ 1_2 \\ -\ \ 1\ 1\ 0\ 1\ 0_2 \\ \hline 0\ 0\ 1\ 0\ 1\ 1_2 \end{array}$$

Multiplication

·	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	1	2	3	4	5	6	7	8	9
2	0	2	4	6	8	10	12	14	16	18
3	0	3	6	9	12	15	18	21	24	27
4	0	4	8	12	16	20	24	28	32	36
5	0	5	10	15	20	25	30	35	40	45
6	0	6	12	18	24	30	36	42	48	54
7	0	7	14	21	28	35	42	49	56	63
8	0	8	16	24	32	40	48	56	64	72
9	0	9	18	27	36	45	54	63	72	81

Multiplication

$$\begin{array}{r} {\scriptstyle 2\ 1\ 4} \\ 6 \cdot 4\,2\,7 \\ \hline 2\,5\,6\,2 \end{array}$$

Multiplication with single digit

$$\begin{array}{r} 3\,6\,5 \cdot 4\,2\,7 \\ \hline 1\,2\,8\,1 \\ 2\,5\,6\,2 \\ 2\,1\,3\,5 \\ \hline 1\,5\,5\,8\,5\,5 \end{array}$$

$$\begin{aligned} 365 \cdot 427 &= (3 \cdot 10^2 + 6 \cdot 10^1 + 5 \cdot 10^0) \cdot 427 \\ &= 3 \cdot 427 \cdot 10^2 + 6 \cdot 427 \cdot 10^1 + 5 \cdot 427 \cdot 10^0 \\ &= 1281 \cdot 10^2 + 2562 \cdot 10^1 + 2135 \cdot 10^0 \\ &= 128100 + 25620 + 2135 \\ &= 155855 \end{aligned}$$

Binary multiplication

$\cdot$	0	1
0	0	0
1	0	1

Binary multiplication

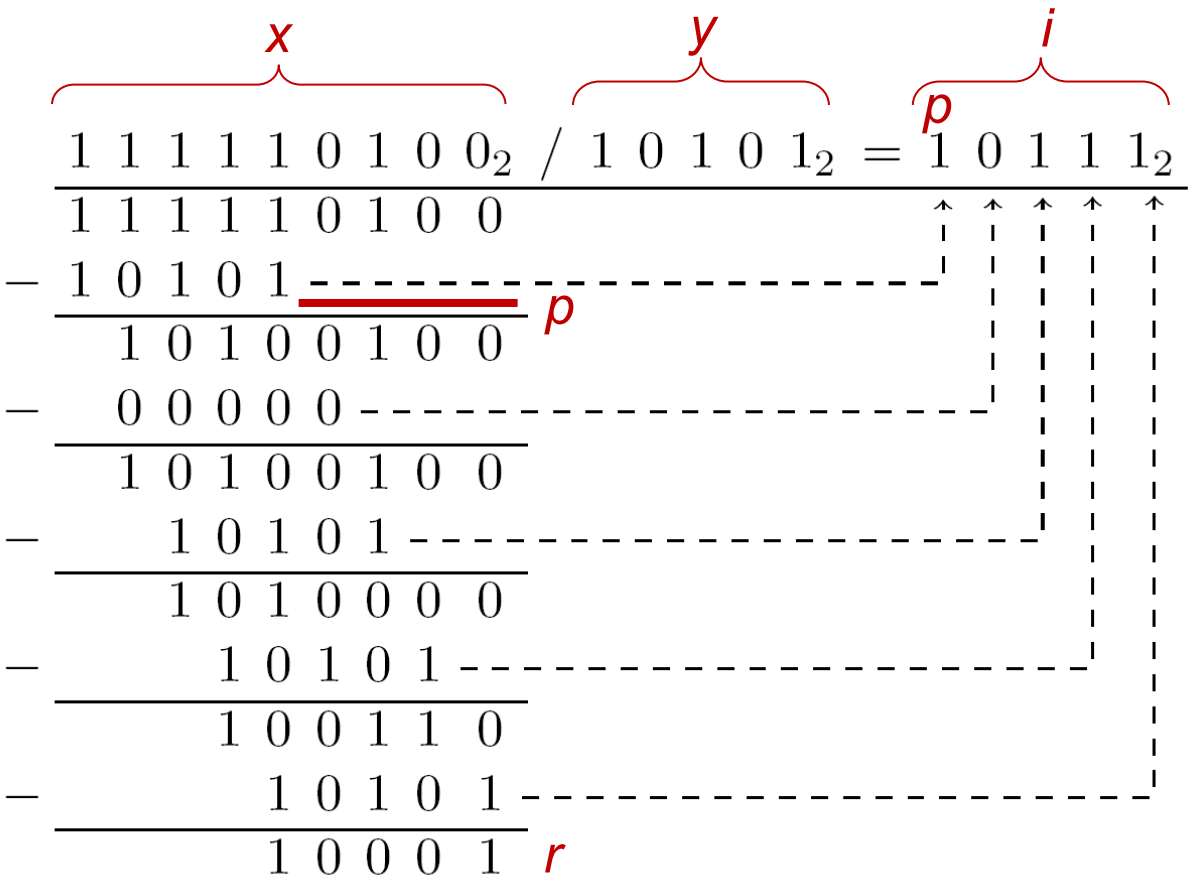
$$
\begin{array}{r}
1\ 0\ 1\ 1\ 1_2 \cdot 1\ 0\ 1\ 0\ 1_2 \\
\hline
1\ 0\ 1\ 0\ 1_2 \\
0\ 0\ 0\ 0\ 0_2 \\
1\ 0\ 1\ 0\ 1_2 \\
1\ 0\ 1\ 0\ 1_2 \\
1\ 0\ 1\ 0\ 1_2 \\
\hline
1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1_2
\end{array}
$$

Binary multiplication of blocks with 1s

$$\overbrace{11111}^{j}\overbrace{00000}^{i}{}_2 = 1\overbrace{0000000000}^{i+j}{}_2 - 1\overbrace{00000}^{i}{}_2$$

$$\begin{array}{r}
1\ 0\ \overbrace{1\ 1\ 1}_2 \cdot 1\ 0\ 1\ 0\ 1_2 \\
\hline
1\ 0\ 1\ 0\ 1_2 \\
\left\{\begin{array}{ll} + & 1\ 0\ 1\ 0\ 1_2 \\ - & 1\ 0\ 1\ 0\ 1_2 \end{array}\right. \\
\hline
1\ 1\ 1\ 1\ 0\ 0\ 0\ 1\ 1_2
\end{array}$$

Division



Algorithm IntegerDivision(x, y)
Input Integers $x \geq 0$ and $y \geq 1$
Output Integer $i = \lfloor x/y \rfloor$, i.e. $0 \leq x - i \cdot y < y$

```
1   p = 0
2   while y · 2^(p+1) ≤ x
3       p = p + 1
4   i = 0
5   r = x
6   # Invariant: x = i · y + r and r < y · 2^(p+1)
7   while y ≤ r
8       if y · 2^p ≤ r
9           i = i + 2^p      # set position p in i = 1
10          r = r − y · 2^p
11      p = p − 1
12  return i
```

Conversion to base *b*

Algorithm BASEREPRESENTATION(x, b)
Input Integers $x \geq 0$ and base $b \geq 2$
Output Digits $d_0, d_1, \ldots$ of the b-ary representation of x

```
1   p = 0
2   while x > 0 do
3       i = ⌊x/b⌋
4       d_p = x − i · b
5       x = i
6       p = p + 1
```

INTEGERDIVISION(x, b)

remainder

$x = 42 \quad b = 2$

p	x	i	$i*b$	p_d	base b
0	42	21	42	0	0
1	21	10	20	1	10
2	10	5	10	0	010
3	5	2	4	1	1010
4	2	1	2	0	01010
5	1	0	0	1	101010
6	0				

not curriculum

Redundant number systems

$$\sum_{i=0}^{n-1} d_i \cdot b^i$$

- Normally one assumes $0 \leq d_i < b$ ⇒ unique representation
- Redundant number systems also allow other digits, e.g. …,-2,-1,b,b+1,…

$$42_{10} = 101010_2 = 21002_2 = 22\text{-}1\text{-}10_2$$

⇒ *not* unique

$2*2^4+1*2^3+2*2^0$ $\quad$ $2*2^4+2*2^3+-1*2^2+-1*2^1$

- Advantage can avoid cascaded carries under addition
- Idea used in hardware and many algorithms and data structures

Amr Elmasry, Jyrki Katajainen, *Regular numeral systems for data structures*, Acta Informatica, 2022, doi: 10.1007/s00236-021-00407-9

Algorithms and Data Structures

Induction and invariants

Proof by induction

- We wish to prove an infinite sequence of statements

$$U_1, U_2, U_3, U_4, U_5, \ldots, U_n, U_{n+1}, \ldots.$$

- E.g., U_n could be the statement $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

The statements can be **proved by using the induction principle**, by proving the following two statements:

1. **Base case**:
   Prove that U_1 is true
2. **Induction step**:
   Assume for an $n \geq 1$ that U_n is true (**induction hypothesis**)
   and prove that then U_{n+1} is true, i.e., prove that $U_n \Rightarrow U_{n+1}$

Example: Prove $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

[CLRS A.1] (A.1)

- Base case $n = 1$: $\sum_{i=1}^{1} i = 1 = \frac{1(1+1)}{2}$
- Induction step:

  Induction hypothesis: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

  We must prove: $\sum_{i=1}^{n+1} i = \frac{(n+1)\big((n+1)+1\big)}{2}$

  Proof: $\sum_{i=1}^{n+1} i = (n+1) + \sum_{i=1}^{n} i \overset{\text{i.h.}}{=} n + 1 + \frac{n(n+1)}{2}$

  $= \frac{2(n+1) + n(n+1)}{2} = \frac{(2+n)(n+1)}{2}$

  $= \frac{(n+1)\big((n+1)+1\big)}{2}$

  $ac + bc = (a+b)c$

□

Example: Prove $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

[CLRS A.1] (A.6)

- Base case $n = 0$: $\sum_{i=0}^{0} 2^i = 2^0 = 1 = 2^{0+1} - 1$
- Induction step:

  Induction hypothesis: $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

  We must prove: $\sum_{i=0}^{n+1} 2^i = 2^{(n+1)+1} - 1$

  Proof:

$$\sum_{i=0}^{n+1} 2^i = 2^{n+1} + \sum_{i=0}^{n} 2^i \overset{\text{i.h.}}{=} 2^{n+1} + 2^{n+1} - 1$$

$$= 2 \cdot 2^{n+1} - 1 = 2^{n+2} - 1 = 2^{(n+1)+1} - 1 \qquad \square$$

Example: Prove $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$ for $x \neq 1$

[CLRS A.1] (A.6)

- Base case $n = 0$:

  $\sum_{i=0}^{0} x^i = x^0 = 1 = \frac{x^{0+1}-1}{x-1}$ $\qquad x \neq 1$

- Induction step:

  Induction hypothesis: $\sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$

  We must prove: $\sum_{i=0}^{n+1} x^i = \frac{x^{(n+1)+1}-1}{x-1}$

  Proof:

$$\sum_{i=0}^{n+1} x^i = x^{n+1} + \sum_{i=0}^{n} x^i \overset{\text{i.h.}}{=} x^{n+1} + \frac{x^{n+1}-1}{x-1}$$

$$= \frac{(x-1)x^{n+1}+x^{n+1}-1}{x-1} = \frac{x^{n+2}-1}{x-1} = \frac{x^{(n+1)+1}-1}{x-1} \quad \square$$

Example: Prove $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

[CLRS A.1] (A.4)

- Base case $n = 1$: $\sum_{i=1}^{1} i^2 = 1 = \frac{1(1+1)(2\cdot 1+1)}{6}$
- Induction step:

  Induction hypothesis: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

  We must prove: $\sum_{i=1}^{n+1} i^2 = \frac{(n+1)\big((n+1)+1\big)(2(n+1)+1)}{6}$

Proof $\sum_{i=1}^{n+1} i^2 = (n+1)^2 + \sum_{i=1}^{n} i^2$

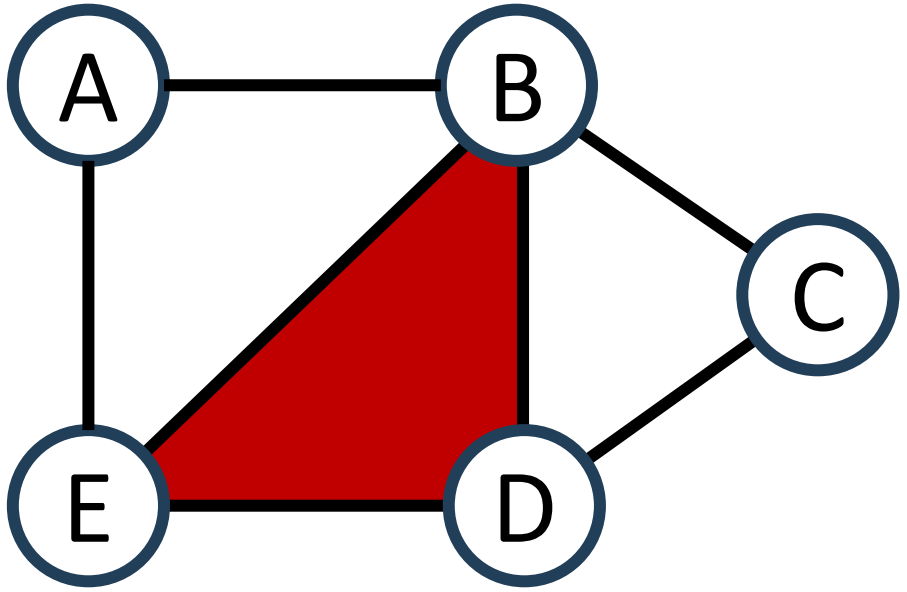
$$\overset{\text{i.h.}}{=} (n+1)^2 + \frac{n(n+1)(2n+1)}{6} = \frac{6(n+1)^2 + n(n+1)(2n+1)}{6}$$

$$= \frac{(n+1)(2n^2+7n+6)}{6} = \frac{(n+1)(n+2)(2n+3)}{6}$$

$$= \frac{(n+1)\big((n+1)+1\big)(2(n+1)+1)}{6}$$

□

Planar graphs – Euler's formula



$|V| = 5$ nodes
$|E| = 7$ edges
faces = 4

A connected planar graph satisfies

Euler's formula: $|V| - |E| + \#\text{ faces} = 2$

Corollary: $|E| \leq 3|V| - 6$

(for $|V| \geq 3$, no self-loops, no parallel edges)

www.merriam-webster.com

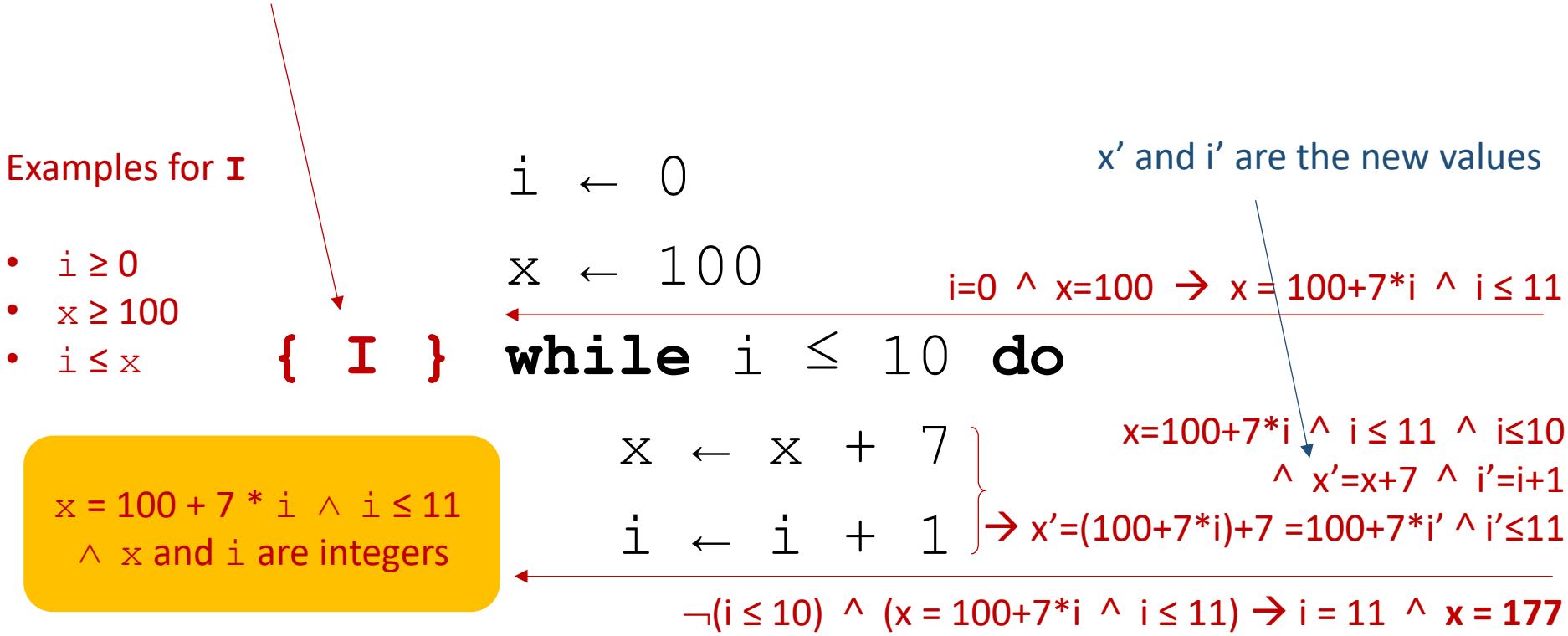
```
i ← 0
x ← 100
while i ≤ 10 do
    x ← x + 7
    i ← i + 1
```

What is the value of `x` when the program terminates ?

```
i ← 0
x ← 100
while i ≤ 10 do
    x ← x + 7
    i ← i + 1
```

a) 100 b) 107 c) 110 d) 111 e) 170 f) 177 g) Don’t know

Loop invariant = statement



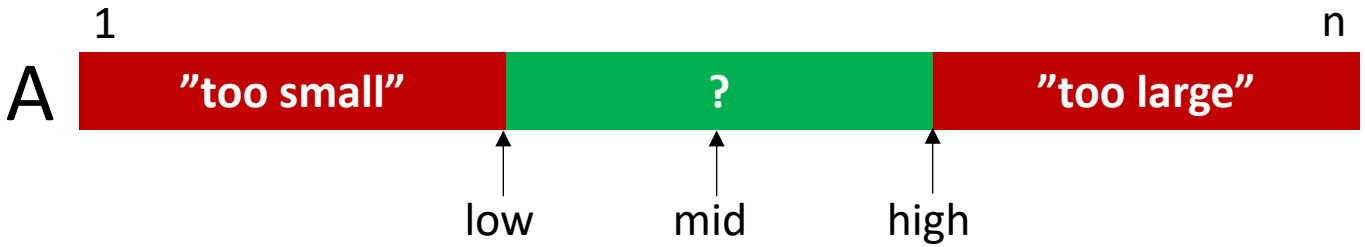
An invariant I should satisfy

1) When the loop is reached the first time, **I** is satisfied
2) If **I** is satisfied before the loop, then **I** is satisfied after an iteration of the loop

I is automatically satisfied when we get out of the loop

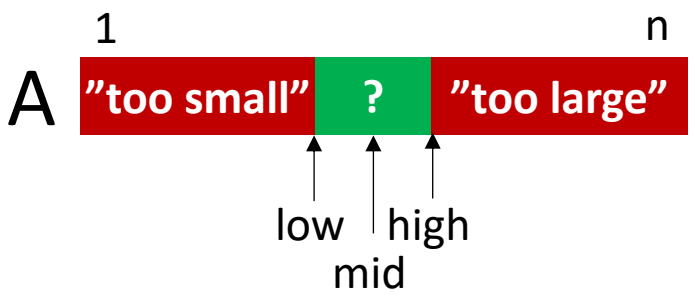
Use **I** and that the loop condition is false when the loop terminations to draw a conclusion

Binary search (in a sorted array)




```
        << initialize >>
{ I } while << loop condition >> do
            mid ← << f(low,high) >>
            << update >>
```

Binary search (in a sorted array)




```
                << initialize >>
{ I } while << loop condition >> do
        mid ← << f(low,high) >>
        << update >>
```

I_1: (A[i] < x for $1 \le i \le$ low) ^ (x < A[i] for high $\le i \le n$)

I_2: (A[i] < x for $1 \le i \le$ low) ^ (x $\le$ A[i] for high $\le i \le n$)

I_3: (A[i] < x for $1 \le i <$ low) ^ (x $\le$ A[i] for high $< i \le n$)

I_4: (A[i] < x for $1 \le i <$ low) ^ (x $\le$ A[i] for high $\le i \le n$)

...

<< initialize >>

a) low ← 0; high ← n+1
b) low ← 1; high ← n
c) low ← 0; high ← n
d) low ← 1; high ← n+1
e) Don't know

I: (A[i] < x for 1 ≤ i < low) ^ (x ≤ A[i] for high < i ≤ n)

<< loop condition >>

a) low = high
b) low ≠ high
c) low < high
d) low ≤ high
e) low > high
f) Don't know

I: (A[i] < x for 1 ≤ i < low) ^ (x ≤ A[i] for high < i ≤ n)

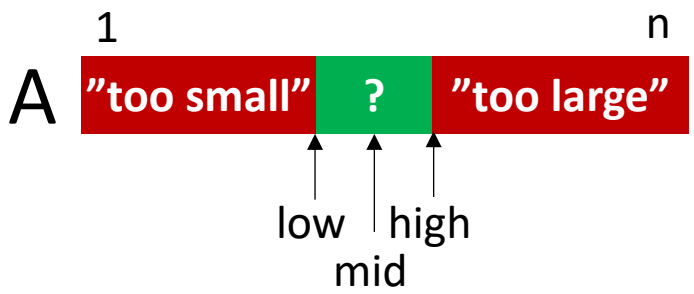
<< update >>

a) if A[mid] < x then low ← mid else high ← mid
b) if A[mid] < x then low ← mid+1 else high ← mid
c) if A[mid] < x then low ← mid else high ← mid+1

d) if A[mid] < x then low ← mid+1 else high ← mid-1
e) Don't know

I: (A[i] < x for 1 ≤ i < low) ^ (x ≤ A[i] for high < i ≤ n)

Binary search (in a sorted array)




```
        low ← 1; high ← n
{ I } while low ≤ high  do
          mid ← ⌊low+(high-low)/2⌋
          if A[mid] < x then
             low ← mid+1
          else
             high ← mid-1
```

I: (A[i] < x for 1 ≤ i < low) ^ (x ≤ A[i] for high < i ≤ n)

Where is the answer located ?

a) low - 1

b) low

c) low + 1

d) high - 1

e) high

f) high + 1

g) Don't know

I: (A[i] < x for 1 ≤ i < low) ^ (x ≤ A[i] for high < i ≤ n)

Algorithm $\mathrm{POWER}(x, p)$

Input : Integer $x \geq 0$ and $p \geq 0$
Output : $r = x^p$
Method : $r \leftarrow 1;$
$\{I\}$ **while** $p \geq 1$ **do**
if p even **then**
$x \leftarrow x * x;\ p \leftarrow p/2$
else
$r \leftarrow r * x;\ p \leftarrow p - 1$

p_0 = initial value of p

	Yes	No
$p \leq p_0$	☒	☐
$x^p = r$	☐	☒
$x \leq x_0$	☐	☒
$r \cdot x_0^{p_0} = x^p$	☐	☒
$x_0^{p_0} = r \cdot x^p$	☒	☐

Exam, Algorithms and Data Structures, March 2017, problem 21

Algorithm LOG2(n)
Input : Integer $n \geq 2$
Output : $r = \text{intlog}(n) = \lfloor \log_2 n \rfloor$
Method : $i \leftarrow 1;$
$r \leftarrow 1;$
$p \leftarrow 2;$
$\{I\}$ **while** $2p \leq n$ **do**
if $p * p \leq n$ **then**
$p \leftarrow p * p;$
$r \leftarrow 2 * r$
else
$p \leftarrow 2 * p;$
$r \leftarrow r + 1$

	Yes	No
$1 \leq r < p$	☒	☐
$2p \leq n$	☐	☒
$p = 2^r$	☒	☐
$p = 2r$	☐	☒
$p = 2^{\text{intlog}(p)}$	☒	☐

Exam, Algorithms and Data Structures, March 2015, problem 21

Invariants

- A tool to analyze the states of a loop in an algorithm
- Design tool ”Invariant → Code”
- Can be used to identify central properties of the state of a data structure

Algorithms and Data Structures

Analysis tools, RAM model,
Insertion-Sort, O-notation
[CLRS, Chapter 1-3.2]

What is the running time of an algorithm?

From idea to program execusion

Idea

Divide and combine ?

Pseudocode

```
...
for each x in m up to middle
    add x to left
for each x in m after middle
    add x to right
...
```

(Java) code

```
...
if ( t1[t1index] <= t2[t2index] )
    a[index] = t1[t1index++];
else
    a[index] = t2[t2index++];
...
```

Compiler

Microcode

Virtual memory

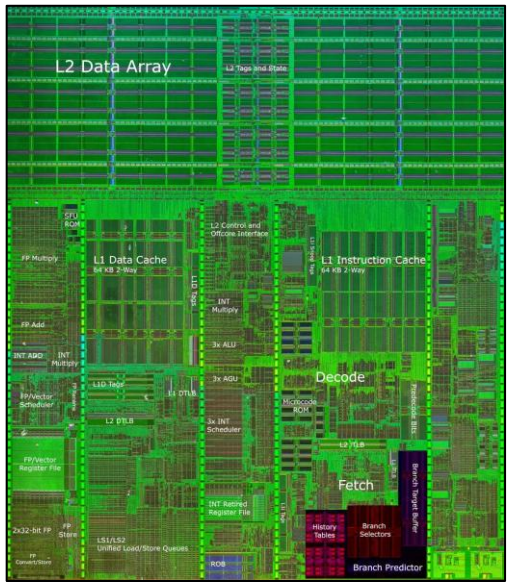
TLB

L1, L2,… cache

Branch Prediction

Pipelining

…



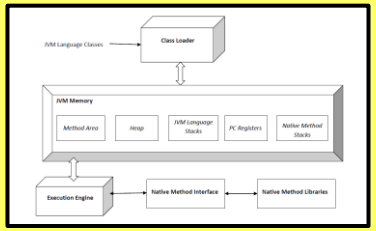
AMD Athlon (2006)
chipsandcheese.com

Program execursion

Machine code

Assembler

(Java) Bytecode



+ Virtual machine

wikipedia.org/wiki/Java_virtual_machine

What is the running time of an ALGORITHM?

a) Time (min/sec) to run a program
b) # instructions executed
c) # memory cells read/written
d) # procedure calls
e) Other

Machines computation time vary...

Machine	Time (sec)
camel19	8.9
molotov	10.2
harald	26.2
gorm	7.8

Time to sort 65 MB of web log activity around year 2000 on various machines at the Department of Computer Science

Idea:

Argue about algorithms independently of machines used

Design of Algorithms

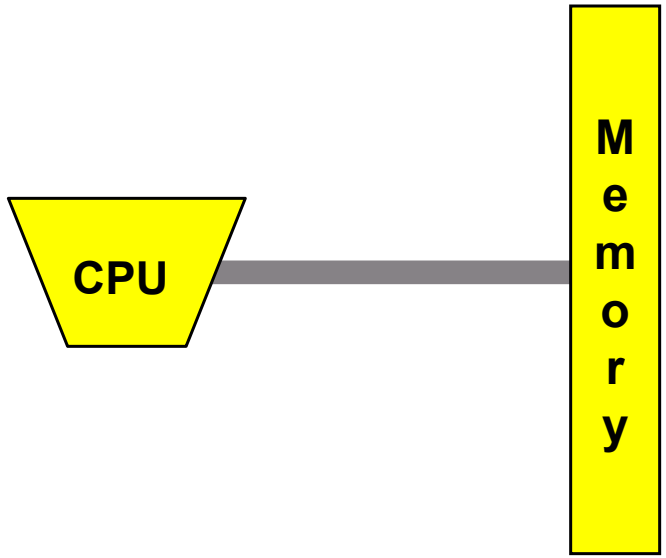
Correct algorithm

- the algorithm **terminates** on all inputs
- output **correct** on all inputs

Efficiency

- Optimize algorithms to use **minimal** time
- space, additions,... or **maximal** parallelism...
- ~ n^2 is better than ~ n^3 : **asymptotic time**
- Less important : **constants**
- Resource usage: **Worst-case** or **average**?

RAM Model
(Random Access Machine)



- Data is stored in the memory
- Computations happen in the CPU (Central Processing Unit)
- Basic operations take 1 time unit:

  +, -, *, AND, OR, XOR, get(i), set(i,v), ...

- A machine word contains $c \cdot \log n$ bits

Example

```
INSERTION-SORT(A, n)
1   for i = 2 to n
2       key = A[i]
3       // Insert A[i] into the sorted subarray A[1 : i − 1]
4       j = i − 1
5       while j > 0 and A[j] > key
6           A[j + 1] = A[j]
7           j = j − 1
8       A[j + 1] = key
```

How many times is line 6 executed ?

a) ~ n

b) ~ $n \log n$

c) ~ n^2

d) ~ n^3

e) Don't know

```
INSERTION-SORT(A, n)
1   for i = 2 to n
2       key = A[i]
3       // Insert A[i] into the sorted subarray A[1 : i − 1]
4       j = i − 1
5       while j > 0 and A[j] > key
6           A[j + 1] = A[j]
7           j = j − 1
8       A[j + 1] = key
```

Example

```
INSERTION-SORT(A, n)
```

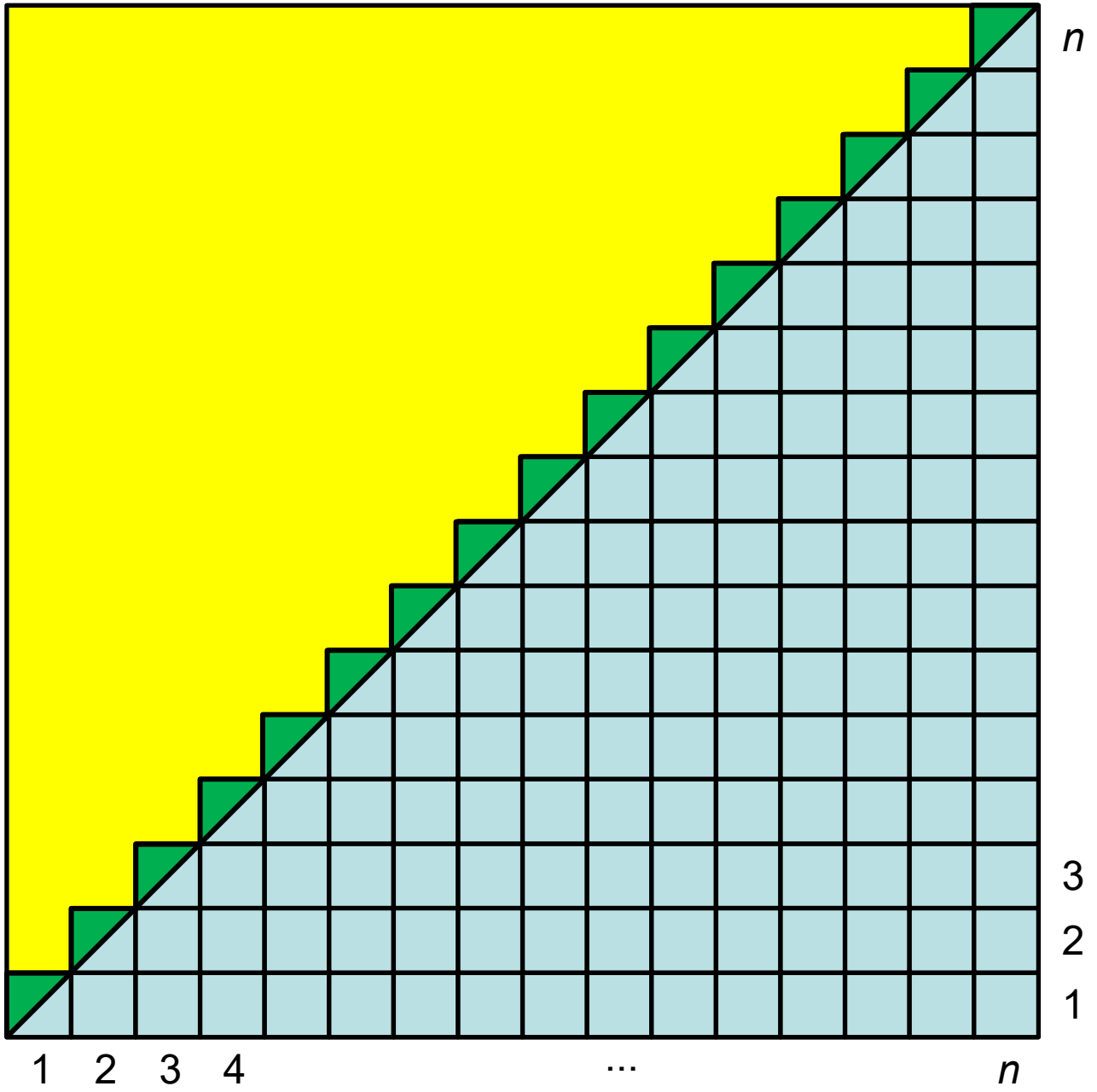
1	$i = 2$	// write 2 to i : 1 instruction
$n \times 3$	**while** $i \le n$	// read i, read n, compare : 3 instructions
	....	
$(n\text{-}1) \times 3$	$i = i + 1$	// read i, add 1, write i : 3 instructions

$6n - 2$	1 **for** $i = 2$ **to** n	
$(n\text{-}1) \times 4$	2 $key = A[i]$	// read i, add address of A to get address of $A[i]$, read A[i], write to key : 4 instructions
0	3 // Insert $A[i]$ into the sorted subarray $A[1 : i - 1]$	
$(n\text{-}1) \times 3$	4 $j = i - 1$	// read i, subtract -1, write j : 3 instructions
$\le i \times 7$	5 **while** $j > 0$ and $A[j] > key$	// read j, compare $j > 0$, add address of A to get address of $A[j]$, read A[j], read key, compare A[j] > key, and : 7 instructions
$\le (i\text{-}1) \times 5$	6 $A[j + 1] = A[j]$	// read j, add address of A to get address of $A[j]$, read A[j], add 1 to get address A[j+1], write to A[j+1] : 5 instructions
$\le (i\text{-}1) \times 3$	7 $j = j - 1$	// read j, subtract -1, write j : 3 instructions
$(n\text{-}1) \times 5$	8 $A[j + 1] = key$	// read j, add +1, add address of A to get address of A[j+1], read key, write to A[j+1] : 5 instructions

$\sum_{i=2}^{n}$

Total number of instructions $\le 18n - 14 + \sum_{i=2}^{n} (15i - 8) = 7.5\, n^2 + 17.5\, n - 21$

$$1 + 2 + \cdots + n = n^2/2 + n/2 = \frac{n(n+1)}{2}$$



Insertion-Sort (C)

```
insertion(int a[], int N)
 { int i, j, key;

   for(j=1; j < N; j++)
    { key = a[j];
      i = j-1;
      while( i>=0 && a[i] > key )
       { a[i+1] = a[i];
         i--;
       }
      a[i+1] = key;
    }
 }
```

```
insertion:
        pushl   %ebp
        movl    %esp, %ebp
        pushl   %edi
        pushl   %esi
        pushl   %ebx
        subl    $12, %esp
        cmpl    $1, 12(%ebp)
        jle     .L3
        movl    8(%ebp), %edx
        xorl    %ebx, %ebx
        movl    8(%ebp), %eax
        movl    $1, -16(%ebp)
        movl    4(%edx), %edx
        addl    $4, %eax
        movl    %eax, -20(%ebp)
        movl    %edx, -24(%ebp)
        .p2align 4,,7
.L6:
        movl    8(%ebp), %ecx
        leal    0(,%ebx,4), %esi
        movl    (%ecx,%ebx,4), %eax
        cmpl    -24(%ebp), %eax
        jle     .L8
        movl    %ecx, %edi
        leal    -4(%esi), %ecx
        leal    (%ecx,%edi), %edx
        jmp     .L9
        .p2align 4,,7
.L16:
        movl    (%edx), %eax
        movl    %ecx, %esi
        subl    $4, %edx
        subl    $4, %ecx
        cmpl    -24(%ebp), %eax
        jle     .L8
.L9:
        movl    -20(%ebp), %edi
        subl    $1, %ebx
        movl    %eax, (%edi,%esi)
        jns     .L16
.L8:
        movl    -16(%ebp), %edi
        movl    8(%ebp), %edx
        leal    (%edx,%edi,4), %eax
.L5:
        movl    -24(%ebp), %ecx
        movl    -20(%ebp), %edx
        addl    $1, -16(%ebp)
        movl    -16(%ebp), %edi
        movl    %ecx, (%edx,%ebx,4)
        cmpl    %edi, 12(%ebp)
        jle     .L3
        movl    4(%eax), %edx
        movl    %edi, %ebx
        addl    $4, %eax
        subl    $1, %ebx
        movl    %edx, -24(%ebp)
        jns     .L6
        jmp     .L5
.L3:
        addl    $12, %esp
        popl    %ebx
        popl    %esi
        popl    %edi
        popl    %ebp
        ret
```

Example: Insertion-Sort

- Example of **pseudocode**
- Detailed analysis – cumbersome work
- Time: **worst-case** (~ n^2) and **best-case** (~ n) very different
- Time: **average** (~ n^2)
- Faster on ~ sorted input: **adaptive**

Asymptotic notation

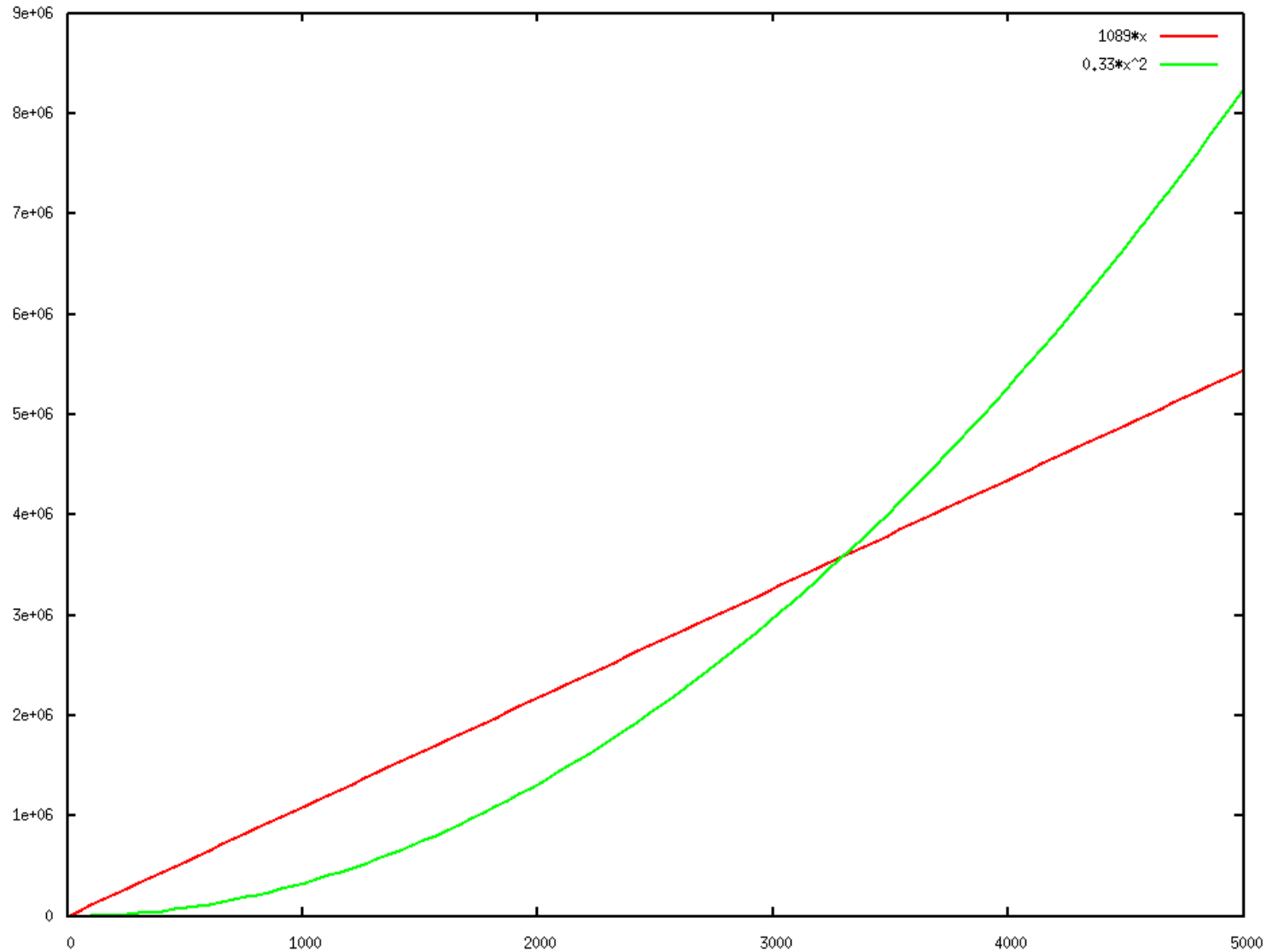
- Basic assumptions:
  - ~ n^2 is better than ~ n^3
  - constants are not essential
- **Mathematical precise** way to work with ”~”
- Examples:

$$87 \cdot n^2 \quad \text{”} \leq \text{”} \quad 12 \cdot n^3$$

$$1089 \cdot n \quad \text{”} \leq \text{”} \quad 0.33 \cdot n^2$$

$$7 \cdot n^2 + 25 \cdot n \quad \text{”} \leq \text{”} \quad n^2$$

$1089 \cdot x$ vs $0.33 \cdot x^2$



O - notation

... and friends

- Ω (big omega)
- Θ (theta)
- ω (small omega)
- o (small o)

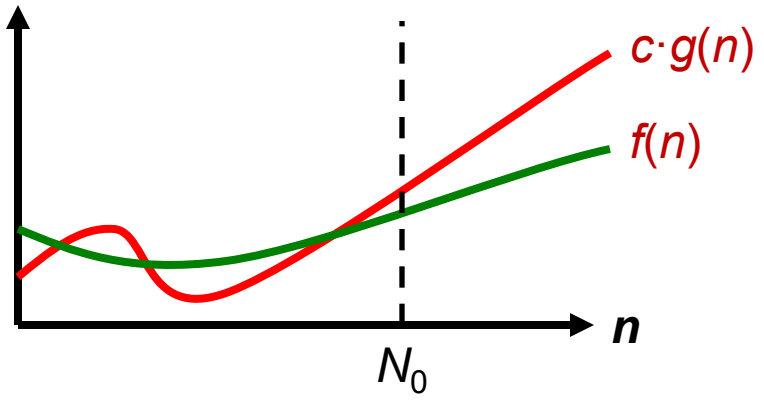
O-notation

Definition: $f(n) = O(g(n))$

if $f(n)$ and $g(n)$ are functions $N \rightarrow R$ and

there *exist constants* $c > 0$ and N_0 such that *for all* $n \geq N_0$:

$$f(n) \leq c \cdot g(n)$$



Intuitively: $f(n)$ is ”smaller or equal to” $g(n)$, or $g(n)$ ”dominates” $f(n)$

$10 \cdot n = O(n^2)$?

a) No

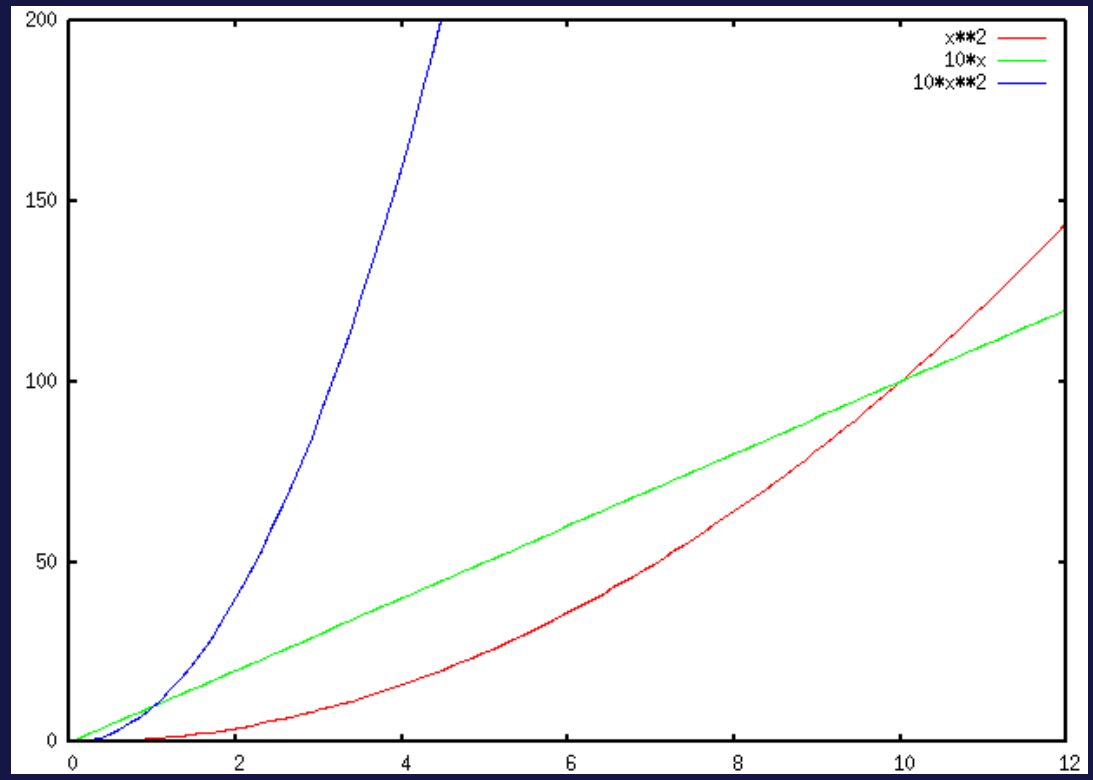
b) Yes – c=1 and N_0=1

c) Yes – c=10 and N_0=1

d) Yes – c=1 and N_0=10

e) Both c) and d)

f) Don't know



O-notation

O($g(n)$) is really the **set of functions** which are asymptotically bounded by $g(n)$

Computer science notation	Mathematical notation
$f(n) = O(g(n))$	$f(n) \in O(g(n))$
$O(f(n)) = O(g(n))$	$O(f(n)) \subseteq O(g(n))$
Examples: $n^2 = O(n^3)$ ~~$O(n^3) = n^2$~~ $O(n^2) = O(n^3)$ ~~$O(n^3) = O(n^2)$~~	

Example: Insertion-Sort

```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1]
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

Time $O(n^2)$

Example : O – rules

$f(n) = O(g(n)) \quad \rightarrow \quad c \cdot f(n) = O(g(n))$ for $c \geq 0$

$$\left.\begin{array}{l} f_1(n) = O(g_1(n)) \\ f_2(n) = O(g_2(n)) \end{array}\right\} \rightarrow \left\{\begin{array}{l} f_1(n) + f_2(n) = O(\max(g_1(n), g_2(n))) \\ f_1(n) \cdot f_2(n) = O(g_1(n) \cdot g_2(n)) \end{array}\right.$$

$$c_k \cdot n^k + c_{k-1} \cdot n^{k-1} + \cdots + c_2 \cdot n^2 + c_1 \cdot n + c_0 = O(n^k)$$

(for positive monotone functions)

$f_1(x) = O(g_1(x))$ and $f_2(x) = O(g_2(x))$

implies

$f_1(x) - f_2(x) = O(g_1(x) - g_2(x))$?

a) Yes

b) No

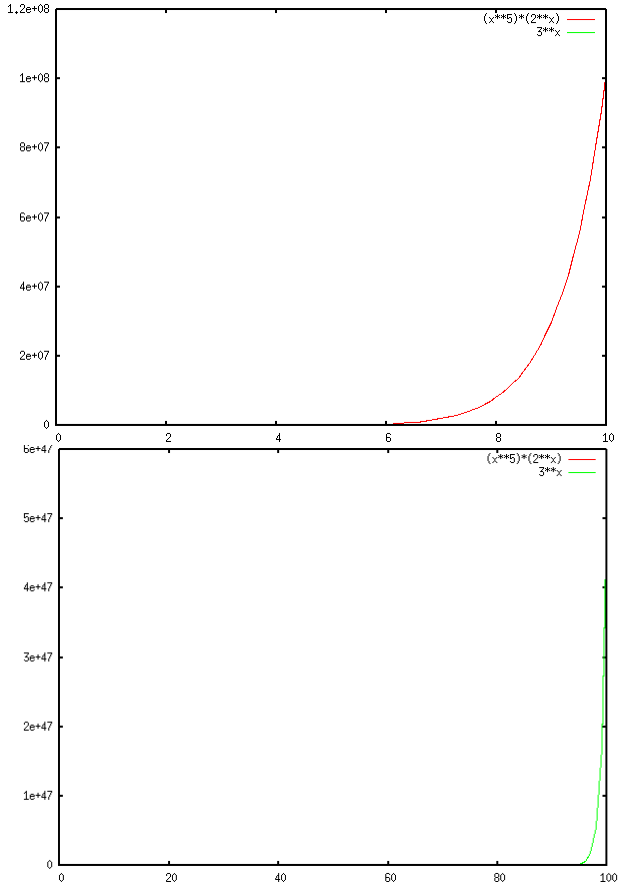
c) Don't know

Examples : O

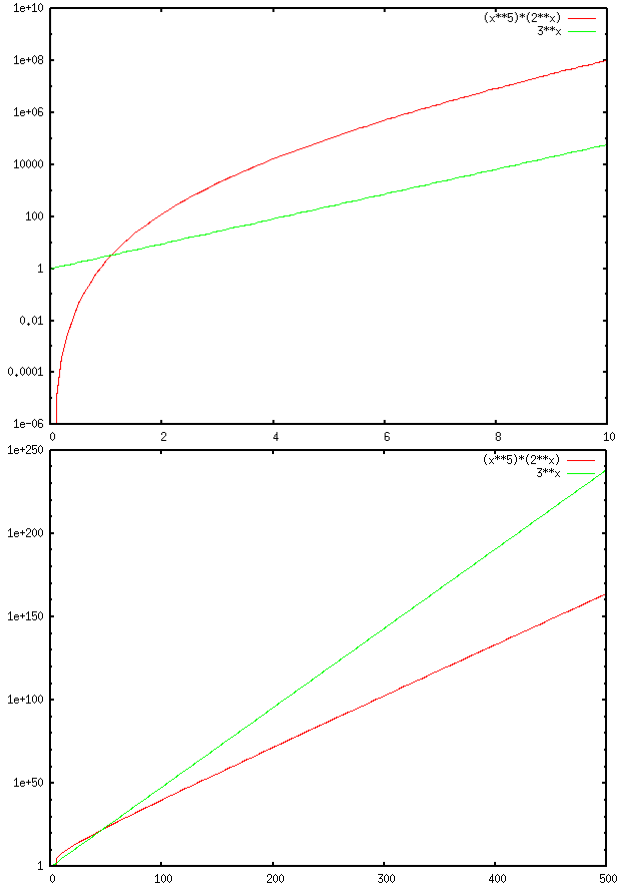
- $3 \cdot n^2 + 7 \cdot n = O(n^2)$
- $n^2 = O(n^3)$
- $\log_2 n = O(n^{0.5})$
- $(\log_2 n)^3 = O(n^{0.1})$
- $n^2 \cdot \log_2 n + 7 \cdot n^{2.5} = O(n^{2.5})$
- $2^n = O(3^n)$
- $n^5 \cdot 2^n = O(3^n)$

Visual test of $n^5 \cdot 2^n = O(3^n)$?

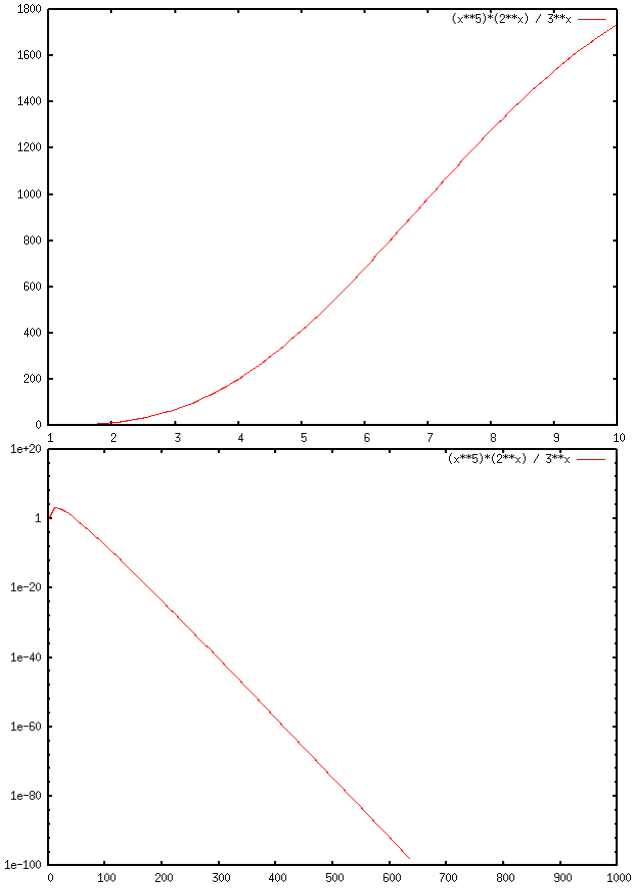
Plot of two functions
– not very informative



Plot of two functions
with logarithmic y-axis
– first plot misleading



Plot of fraction
between two functions
– first plot misleading



Proof of $n^5 \cdot 2^n = O(3^n)$

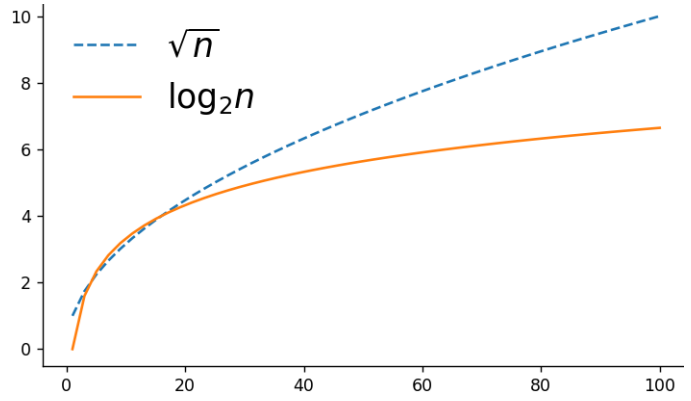
Prove $n^5 \cdot 2^n \le c \cdot 3^n$ for all $n \ge N_0$, for appropriate choices of c and N_0

Proof:

$(5/\log_2 (3/2))^2 \le n$ for $n \ge 73$

$\Downarrow$

$5/\log_2 (3/2) \le \sqrt{n} = n/\sqrt{n} \le n/\log_2 n$ since $\sqrt{n} \ge \log_2 n$ for $n \ge 17$



$\Downarrow$

$5 \cdot \log_2 n \le n \cdot \log_2 (3/2)$

$\Downarrow$

$\log_2 (n^5) \le \log_2 (3/2)^n$

$\Downarrow$

$n^5 \le (3/2)^n = 3^n / 2^n$

$\Downarrow$

$n^5 \cdot 2^n \le 3^n$

i.e., the desired statement holds for $c = 1$ and $N_0 = 73$. □

Theorem $\text{poly}(n) \cdot a^n = O(b^n)$ for all $1 \le a < b$

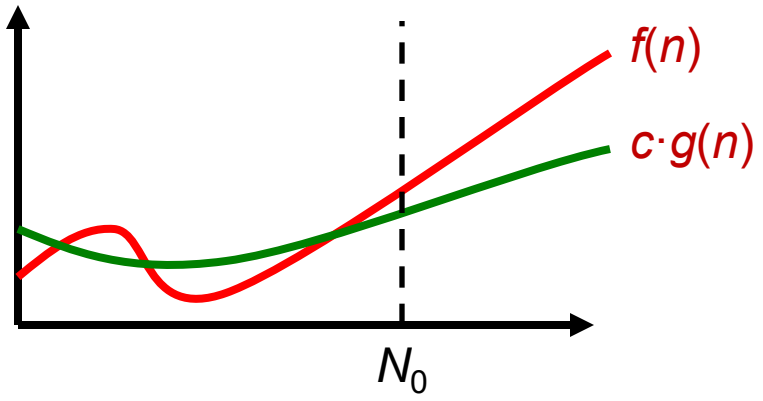
Ω –notation (capital Omega)

Definition: $f(n) = \Omega(g(n))$

if $f(n)$ and $g(n)$ are functions $N \rightarrow R$ and

there *exist constants* $c > 0$ and N_0 such that *for all* $n \geq N_0$:

$f(n) \geq c \cdot g(n)$

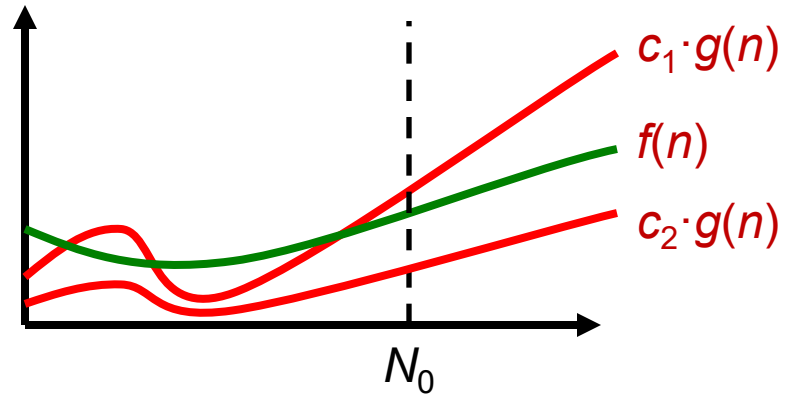


Intuitively: $f(n)$ is ”larger than or equal to” $g(n)$, or $g(n)$ is ”dominated by” $f(n)$

Θ-notation (Theta)

Definition: $f(n) = \Theta(g(n))$

if $f(n) = O(g(n))$ and $f(n) = \Omega(g(n)$



Intuitively: $f(n)$ and $g(n)$ are ”asymptotical identical”

o-notation (small o/omicron)

Definition: $f(n) = o(g(n))$

if $f(n)$ and $g(n)$ are functions $N \rightarrow R$ and

for all $c > 0$, there *exists* and N_0 such that *for all* $n \geq N_0$:

$$f(n) \leq c \cdot g(n)$$

Intuitively: $f(n)$ is ”strictly smaller than” $g(n)$

ω-notation (small omega)

Definition: $f(n) = \omega(g(n))$

if $f(n)$ and $g(n)$ are functions $N \rightarrow R$ and

for all $c > 0$, there *exists* an N_0 such that *for all* $n \geq N_0$:

$$f(n) \geq c \cdot g(n)$$

Intuitively: $f(n)$ is ”strictly larger than” $g(n)$

wikipedia.org/wiki/Big_O_notation#Multiple_variables

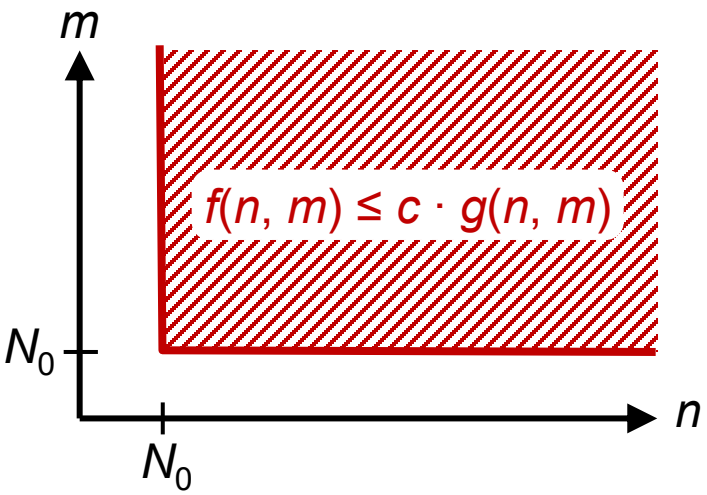
Multi-variable O-notation

We would like to be able to write

$$3n^2 \cdot m + 7n \cdot m = O(n^2 \cdot m)$$

The literature is not precise/consistent...

$$f(n, m) = O(g(n, m)) \ ?$$

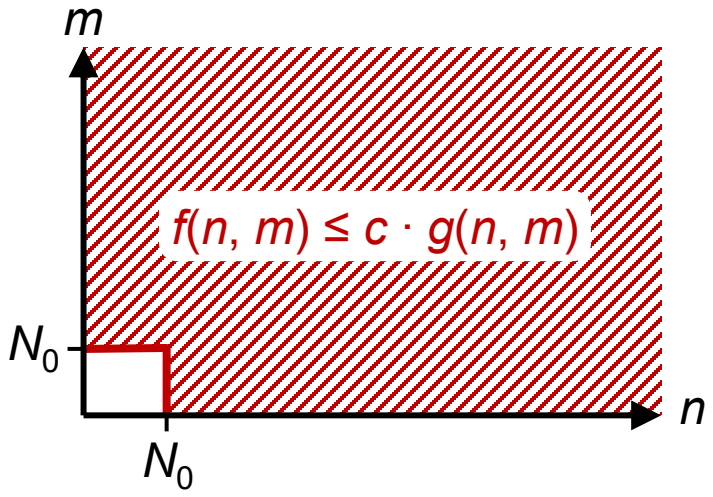


$\exists c\ \exists N_0\ \forall n\ \ \forall m$:

$n \geq N_0 \ \wedge \ m \geq N_0 \ \Rightarrow \ f(n,m) \leq c \cdot g(n,m)$

or

?



$\exists c\ \exists N_0\ \forall n\ \ \forall m$:

$n \geq N_0 \ \vee \ m \geq N_0 \ \Rightarrow \ f(n,m) \leq c \cdot g(n,m)$

Algorithm Analysis

- RAM model
- O-notation

... do not need to describe and analyze algorithms in full detail !

Algorithms and Data Structures

Programming exercises

Programming exercises

- See Brightspace for links to the exercises
- Approximately 14 days to solve the exercises
- Don't copy from others or let an AI assistant solve your exercises
- Usernames and passwords will be available at Brightspace > Grades
- Status of submitted solutions: domjudge.cs.au.dk
- Submitting a wrong solution after having submitted a correct solution does not remove the points
- For each round of programming exercises is stated ”Full points *X* / *Y*”, and a round is passed when you achieve at least *X* points

www.commitstrip.com/en/2018/05/09/progress/

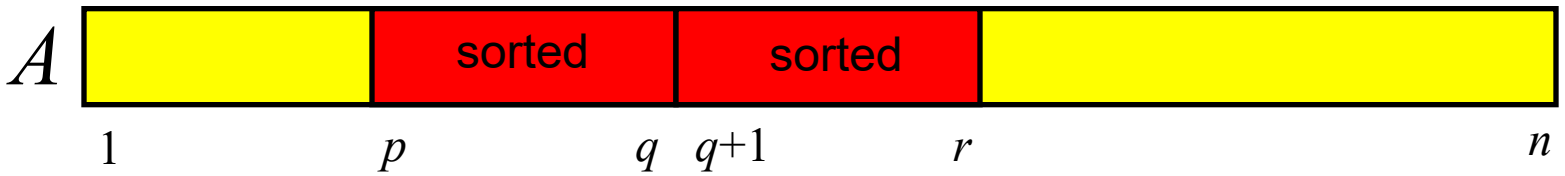
Algorithms and Data Structures

Merge-Sort [CLRS, Chapter 2.3]
Heaps [CLRS, Chapter 6]

Merge-Sort

(example of the divide-and-conquer technique)

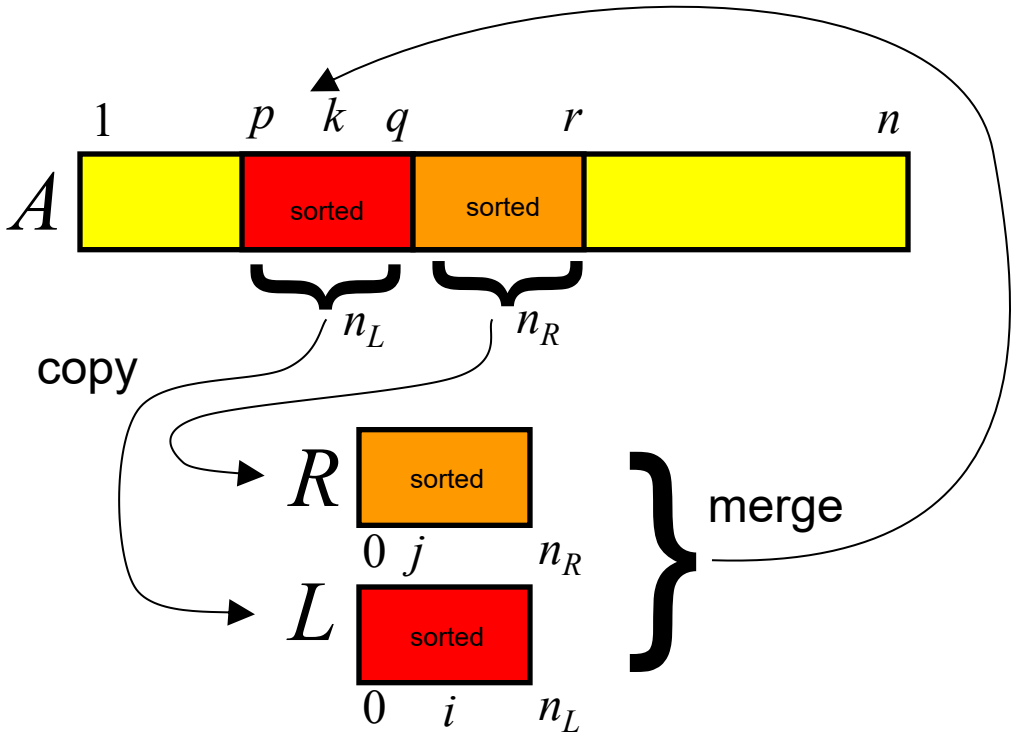
```
MERGE-SORT(A, p, r)
1  if p ≥ r                          // zero or one element?
2      return
3  q = ⌊(p + r)/2⌋                   // midpoint of A[p : r]
4  MERGE-SORT(A, p, q)               // recursively sort A[p : q]
5  MERGE-SORT(A, q + 1, r)           // recursively sort A[q + 1 : r]
6  // Merge A[p : q] and A[q + 1 : r] into A[p : r].
7  MERGE(A, p, q, r)
```



Initial call is MERGE-SORT(*A*,1,*n*)

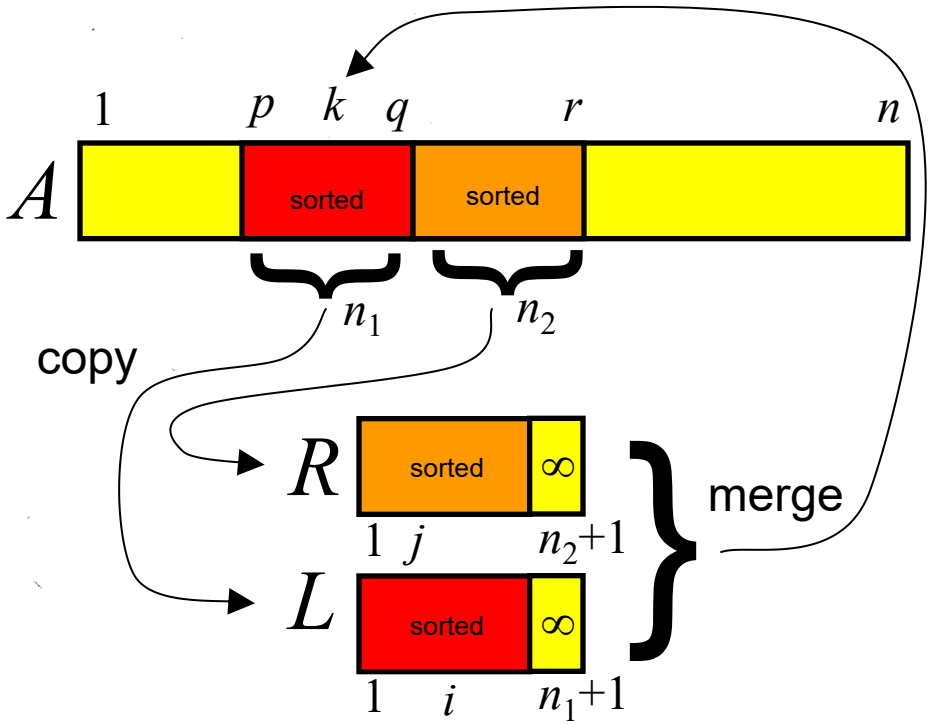
von Neumann, 1945

```
MERGE(A, p, q, r)
1  n_L = q − p + 1          // length of A[p : q]
2  n_R = r − q              // length of A[q + 1 : r]
3  let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
4  for i = 0 to n_L − 1     // copy A[p : q] into L[0 : n_L − 1]
5      L[i] = A[p + i]
6  for j = 0 to n_R − 1     // copy A[q + 1 : r] into R[0 : n_R − 1]
7      R[j] = A[q + j + 1]
8  i = 0                    // i indexes the smallest remaining element in L
9  j = 0                    // j indexes the smallest remaining element in R
10 k = p                    // k indexes the location in A to fill
11 // As long as each of the arrays L and R contains an unmerged element,
   //     copy the smallest unmerged element back into A[p : r].
12 while i < n_L and j < n_R
13     if L[i] ≤ R[j]
14         A[k] = L[i]
15         i = i + 1
16     else A[k] = R[j]
17         j = j + 1
18     k = k + 1
19 // Having gone through one of L and R entirely, copy the
   //     remainder of the other to the end of A[p : r].
20 while i < n_L
21     A[k] = L[i]
22     i = i + 1
23     k = k + 1
24 while j < n_R
25     A[k] = R[j]
26     j = j + 1
27     k = k + 1
```



[CLRS, 3rd edition]

```
MERGE(A, p, q, r)
 1  n1 = q − p + 1
 2  n2 = r − q
 3  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
 4  for i = 1 to n1
 5      L[i] = A[p + i − 1]
 6  for j = 1 to n2
 7      R[j] = A[q + j]
 8  L[n1 + 1] = ∞
 9  R[n2 + 1] = ∞
10  i = 1
11  j = 1
12  for k = p to r
13      if L[i] ≤ R[j]
14          A[k] = L[i]
15          i = i + 1
16      else A[k] = R[j]
17          j = j + 1
```



How many times can an element y be compared ?

(worst-case)

a) O(log n)

b) O(n)

c) O(n log n)

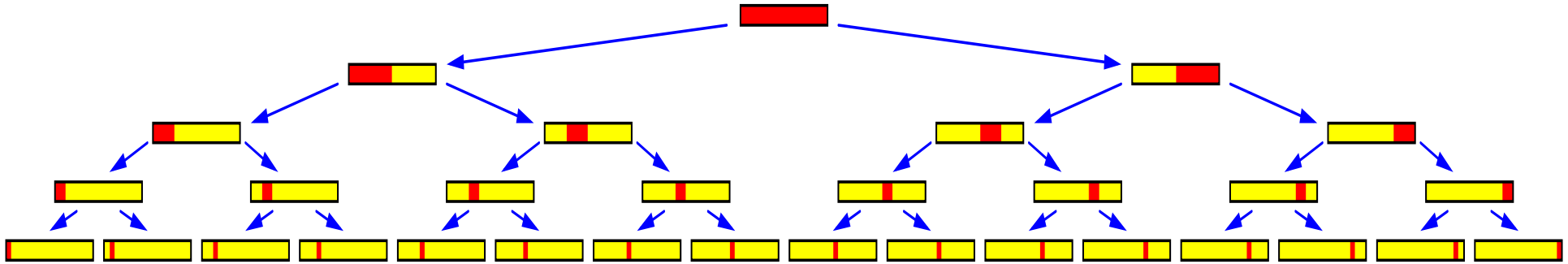
d) O(n^2)

e) Don't know

```
MERGE-SORT(A, p, r)
1  if p ≥ r                                  // zero or one element?
2      return
3  q = ⌊(p + r)/2⌋                           // midpoint of A[p : r]
4  MERGE-SORT(A, p, q)                       // recursively sort A[p : q]
5  MERGE-SORT(A, q + 1, r)                   // recursively sort A[q + 1 : r]
6  // Merge A[p : q] and A[q + 1 : r] into A[p : r].
7  MERGE(A, p, q, r)
```

Merge-Sort : Analysis

Recursion tree



Observation : Total work per level in the tree is O(n)

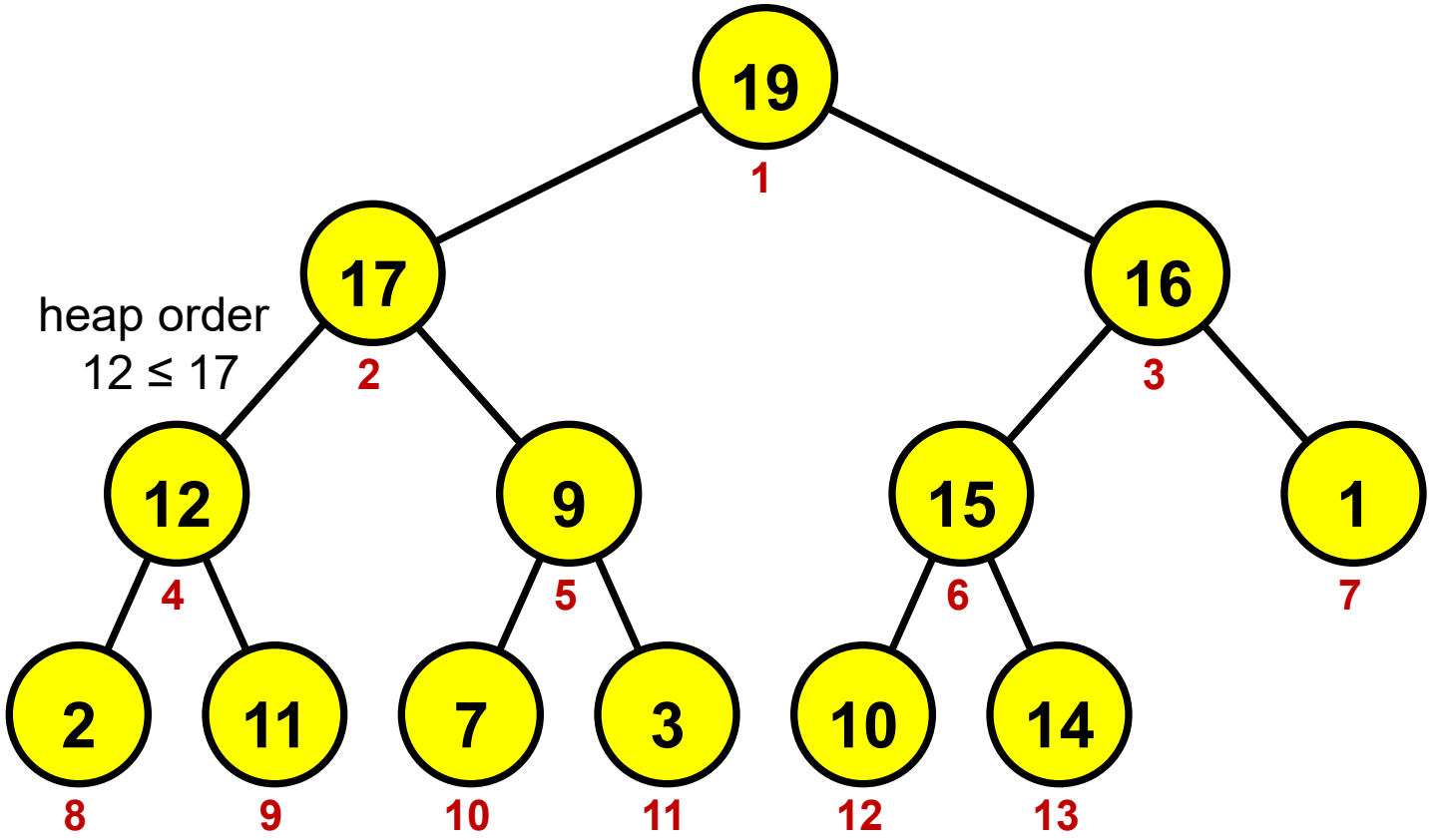
Total work : O(n · # levels) = O(n · $\log_2 n$)

```
MERGE-SORT(A, p, r)
1  if p ≥ r                                  // zero or one element?
2      return
3  q = ⌊(p + r)/2⌋                           // midpoint of A[p : r]
4  MERGE-SORT(A, p, q)                       // recursively sort A[p : q]
5  MERGE-SORT(A, q + 1, r)                   // recursively sort A[q + 1 : r]
6  // Merge A[p : q] and A[q + 1 : r] into A[p : r].
7  MERGE(A, p, q, r)
```

	# procedure calls	# element movements	# comparisons
a)	O(log n)	O(n)	O(n log n)
b)	O(log n)	O(n log n)	O(n log n)
c)	O(n)	O(n)	O(n log n)
d)	O(n)	O(n log n)	O(n log n)
e)	O(n log n)	O(n log n)	O(n log n)
f)		Don't know	

Heap-Sort

Binary (Max-)Heap

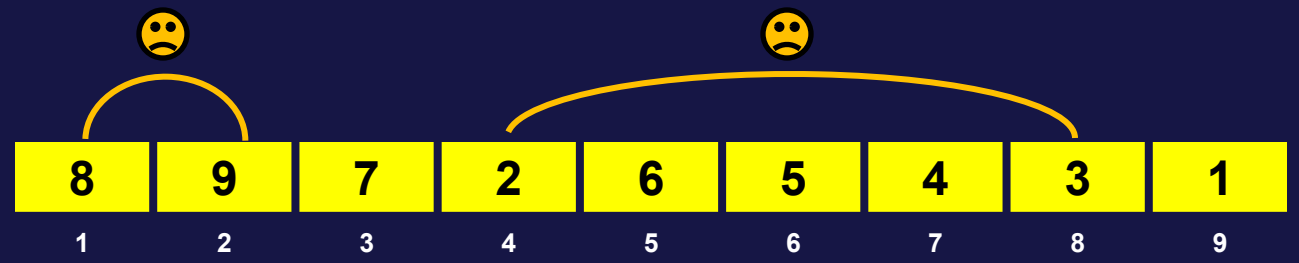


19	17	16	12	9	15	1	2	11	7	3	10	14
1	2	3	4	5	6	7	8	9	10	11	12	13

Williams, 1964

Valid Max-Heap ?

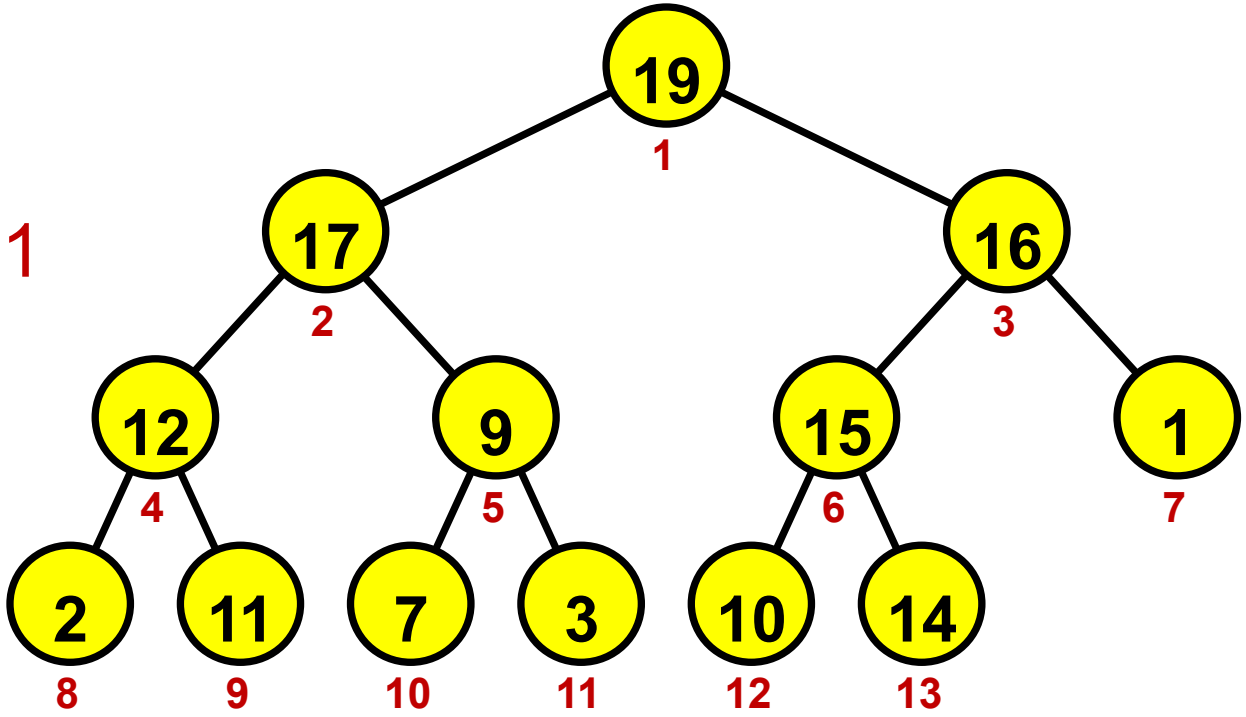
a) Yes
b) No – 1 element does not satisfy heap order
c) No – 2 elements do not satisfy heap order
d) No – 3 elements do not satisfy heap order
e) No – 4 elements do not satisfy heap order
f) Don't know



Max-heap : Properties

- Root : node 1
- Children of node i : $2i$ and $2i+1$
- Parent of node i : $\lfloor i / 2 \rfloor$
- Depth : $1+\lfloor \log_2 n \rfloor$

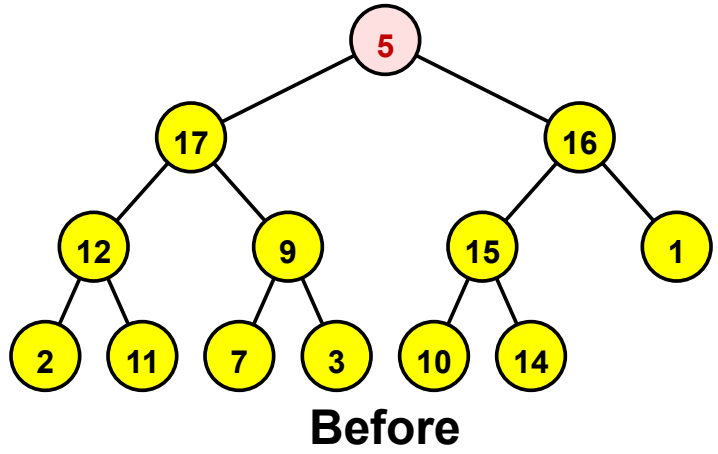
  (n = # elements)



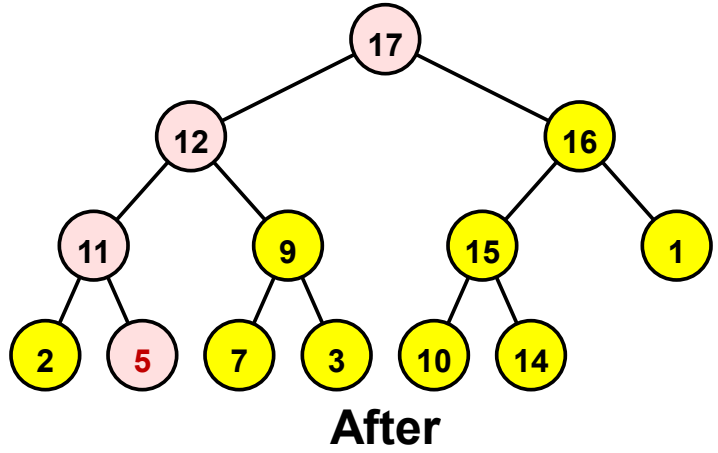
(**Exercise** How to compute parent/children when nodes are numbered 0,1,2,…?)

Max-Heapify

```
MAX-HEAPIFY(A, i)
1   l = LEFT(i)
2   r = RIGHT(i)
3   if l ≤ A.heap-size and A[l] > A[i]
4       largest = l
5   else largest = i
6   if r ≤ A.heap-size and A[r] > A[largest]
7       largest = r
8   if largest ≠ i
9       exchange A[i] with A[largest]
10      MAX-HEAPIFY(A, largest)
```



Before



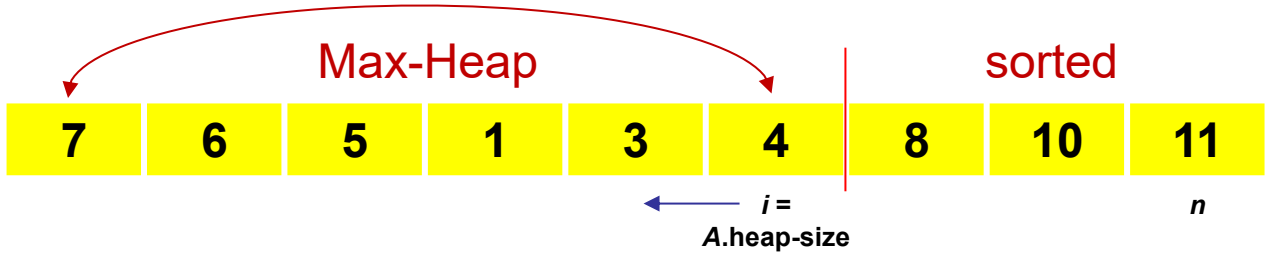
After

Time O(log *n*)

Heap-Sort

BUILD-MAX-HEAP(A, n) **Floyd, 1964**

```
1  A.heap-size = n
2  for i = ⌊n/2⌋ downto 1
3      MAX-HEAPIFY(A, i)
```



HEAPSORT(A, n) **Williams, 1964**

```
1  BUILD-MAX-HEAP(A, n)
2  for i = n downto 2
3      exchange A[1] with A[i]
4      A.heap-size = A.heap-size − 1
5      MAX-HEAPIFY(A, 1)
```

Time

$O(n \cdot \log n)$

J. W. J. Williams, *Algorithm 232: Heapsort,* Communications of the ACM, 1964, doi: 10.1145/512274.3734138

Robert W. Floyd, *Algorithm 245: Treesort,* Communications of the ACM, 1964, doi: 10.1145/355588.365103

Result of applying Build-Max-Heap ?

Input

1	2	3	4	5	6
1	2	3	4	5	6

a)

6	5	4	3	2	1
1	2	3	4	5	6

b)

6	5	3	4	2	1
1	2	3	4	5	6

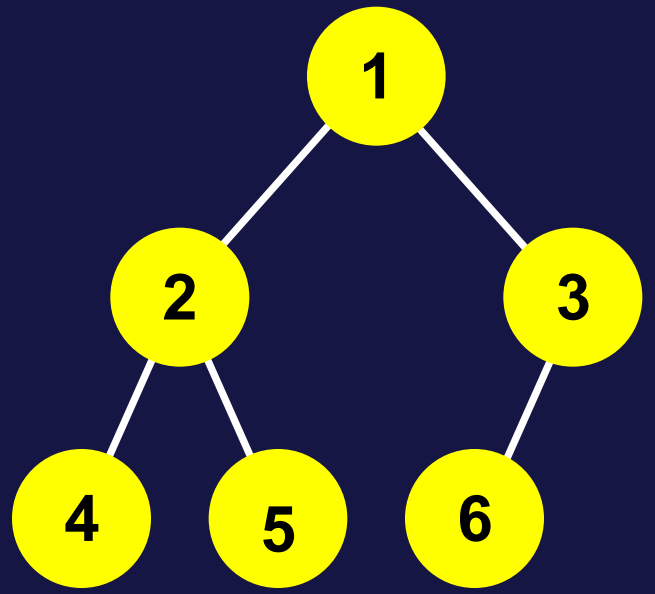
c)

6	5	4	1	2	3
1	2	3	4	5	6

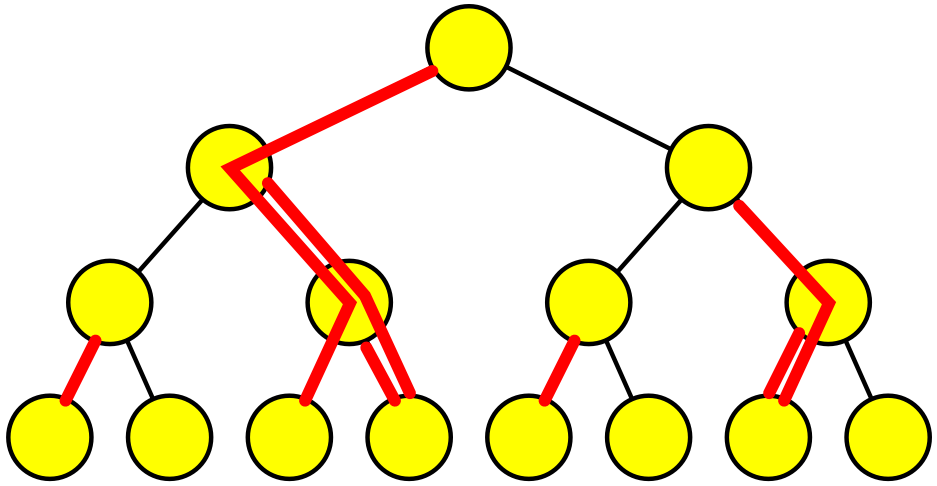
d)

6	5	1	4	2	3
1	2	3	4	5	6

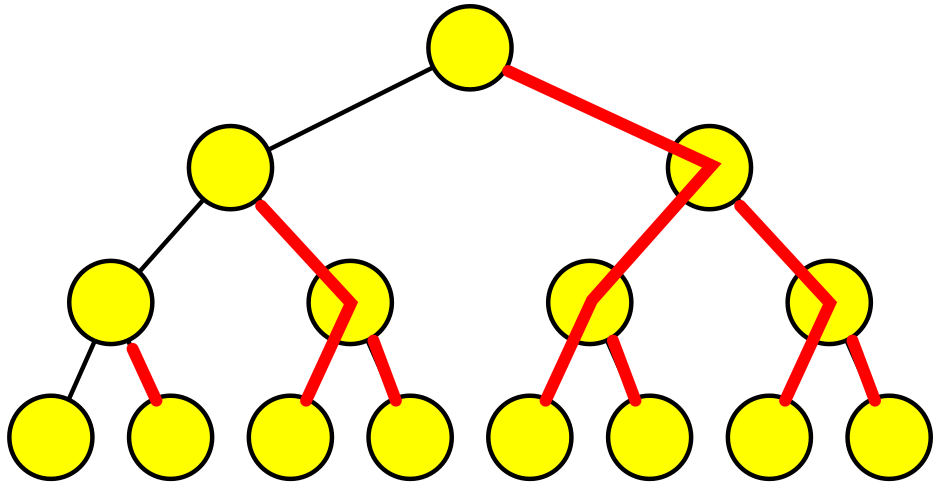
e) Don't know



Build-Max-Heap



Max-Heapify paths (example)



Non-overlapping paths with same # edges (right, left, left, left, ….)

Time for Build-Max-Heap

= Σ time for Max-Heapify

= # red edges

≤ # red edges

= *n* - depth

= O(*n*)

Time O(*n*)

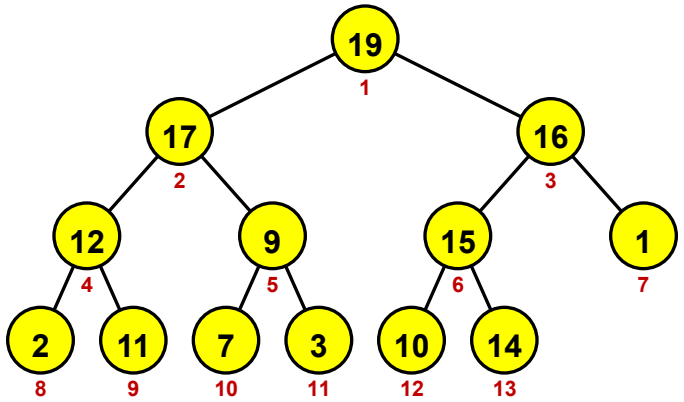
$$\sum_{i=1}^{\log n} i\frac{n}{2^i} \leq 2n$$

Sorting algorithms

Algorithm	Worst-Case Time
Heap-Sort Merge-Sort	$O(n \cdot \log n)$
Selection-Sort Insertion-Sort	$O(n^2)$

[CLRS, 3rd edition]

Max-Heap operations




```
HEAP-MAXIMUM(A)
1  return A[1]
```

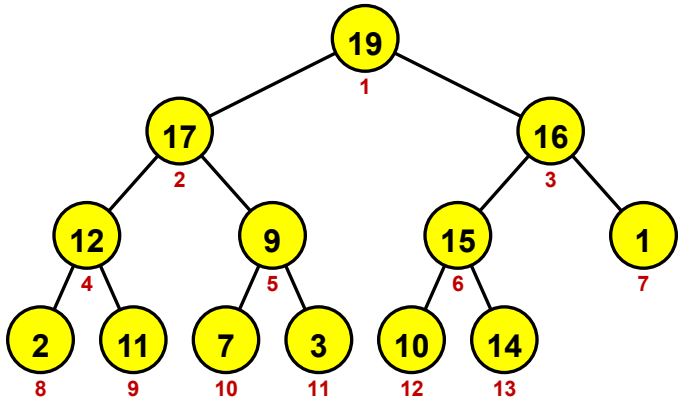
```
HEAP-EXTRACT-MAX(A)
1  if A.heap-size < 1
2       error "heap underflow"
3  max = A[1]
4  A[1] = A[A.heap-size]
5  A.heap-size = A.heap-size − 1
6  MAX-HEAPIFY(A, 1)
7  return max
```

```
MAX-HEAP-INSERT(A, key)
1  A.heap-size = A.heap-size + 1
2  A[A.heap-size] = −∞
3  HEAP-INCREASE-KEY(A, A.heap-size, key)
```

```
HEAP-INCREASE-KEY(A, i, key)
1  if key < A[i]
2       error "new key is smaller than current key"
3  A[i] = key
4  while i > 1 and A[PARENT(i)] < A[i]
5       exchange A[i] with A[PARENT(i)]
6       i = PARENT(i)
```

[CLRS, 4th edition]

Max-Heap operations




```
MAX-HEAP-MAXIMUM(A)
1  if A.heap-size < 1
2        error "heap underflow"
3  return A[1]
```

```
MAX-HEAP-EXTRACT-MAX(A)
1  max = MAX-HEAP-MAXIMUM(A)
2  A[1] = A[A.heap-size]
3  A.heap-size = A.heap-size − 1
4  MAX-HEAPIFY(A, 1)
5  return max
```

```
MAX-HEAP-INSERT(A, x, n)
1  if A.heap-size == n
2        error "heap overflow"
3  A.heap-size = A.heap-size + 1
4  k = x.key
5  x.key = −∞
6  A[A.heap-size] = x
7  map x to index heap-size in the array
8  MAX-HEAP-INCREASE-KEY(A, x, k)
```

New interface to INCREASE-KEY : An "object" instead of an index – needs a secondary structure (map) to keep track of where elements are in the heap

```
MAX-HEAP-INCREASE-KEY(A, x, k)
1  if k < x.key
2        error "new key is smaller than current key"
3  x.key = k
4  find the index i in array A where object x occurs
5  while i > 1 and A[PARENT(i)].key < A[i].key
6        exchange A[i] with A[PARENT(i)], updating the information that maps
           priority queue objects to array indices
7        i = PARENT(i)
```

Max-Heap operations

Operation	Worst-Case Time
Max-Heap-Insert Max-Heap-Extract-Max Max-Heap-Increase-Key	O(log n)
Max-Heap-Maximum	O(1)

n = current number of elements in the heap

Priority queue

A **priority queue** is an **abstract data structure** to store a set of **values** with associated **keys** and supporting the following operations:

- **Insert**(S, x)
- **Maximum**(S)
- **Extract-Max**(S)

Maximum is with respect to the associated keys.

A possible implementation of a priority queue is a heap.

Algorithms and Data Structures

Quicksort
[CLRS, Chapter 7]

Probability theory background?

a) None
b) High school
c) High school + University
d) University
e) Somewhere else
f) Don't know

head (en)
plat (dk)

tail (en)
krone (dk)

Probability for tail when flipping a coin

1/2

Experiment
Throw a coin

a) Head
b) Tail
c) Edgeways
d) No coin

random.org

Experiment
Number of throws before getting tail ?

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) 9 or more

Theorem
Expected number of throws to get tail = 2

Proof

$$\sum_{i=1}^{\infty} i \cdot \left(\frac{1}{2}\right)^{i-1} \cdot \frac{1}{2} = \sum_{i=1}^{\infty} i \cdot \left(\frac{1}{2}\right)^{i} = 2$$

probability i^{th} throw shows tail
probability i - 1 first throws show head
throws

□

Generalization to tail probability p:

$$\sum_{i=1}^{\infty} i \cdot (1-p)^{i-1} \cdot p = \frac{1}{p}$$

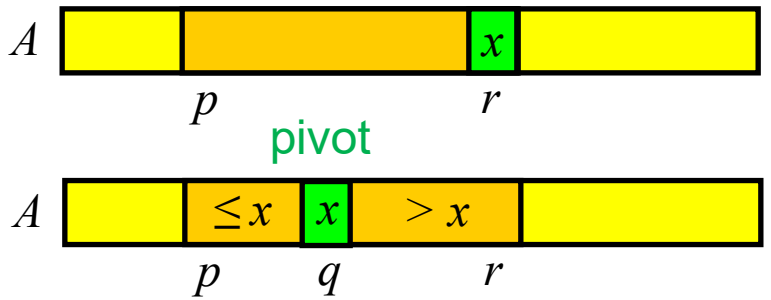
random.org

cs.au.dk/~gerth/slides/math.pdf

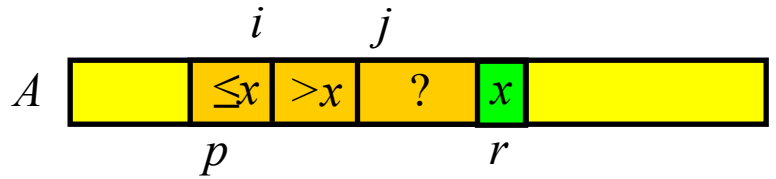
Quicksort

Sort $A[p..r]$

```
QUICKSORT(A, p, r)
1  if p < r
2      // Partition the subarray around the pivot, which ends up in A[q]
3      q = PARTITION(A, p, r)
4      QUICKSORT(A, p, q − 1)  // recursively sort the low side
5      QUICKSORT(A, q + 1, r)  // recursively sort the high side
```



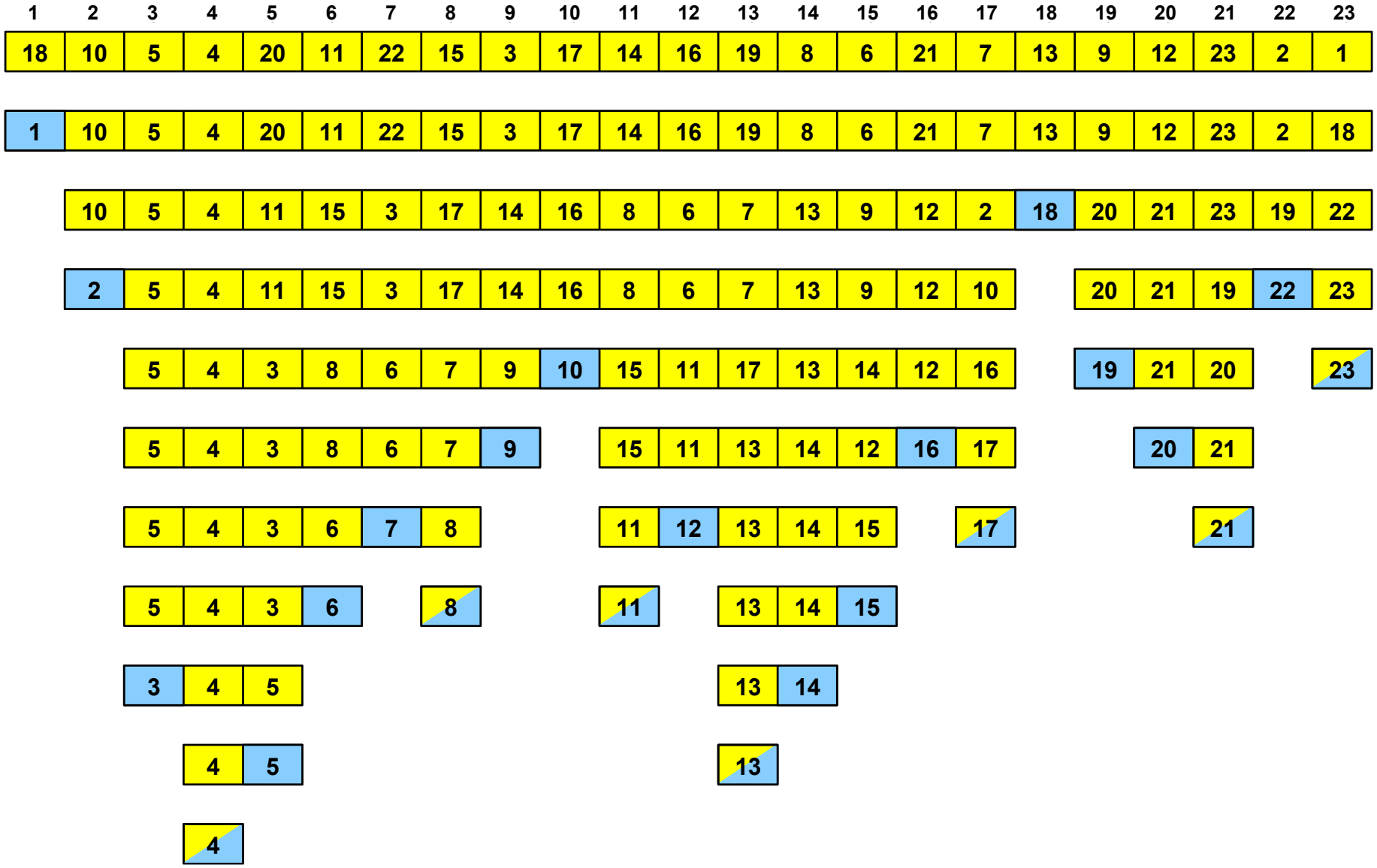

```
PARTITION(A, p, r)
1  x = A[r]                          // the pivot
2  i = p − 1                         // highest index into the low side
3  for j = p to r − 1                // process each element other than the pivot
4      if A[j] ≤ x                   // does this element belong on the low side?
5          i = i + 1                     // index of a new slot in the low side
6          exchange A[i] with A[j]       // put this element there
7  exchange A[i + 1] with A[r]       // pivot goes just to the right of the low side
8  return i + 1                      // new index of the pivot
```



Worst-case time $O(n^2)$

C. A. R. Hoare, *Algorithm 64: Quicksort,* Communications of the ACM, 1961, doi: 10.1145/366622.36664

Quicksort on 23 elements



Number of calls to Partition ?

```
QUICKSORT(A, p, r)
1   if p < r
2       // Partition the subarray around the pivot, which ends up in A[q]
3       q = PARTITION(A, p, r)
4       QUICKSORT(A, p, q − 1)  // recursively sort the low side
5       QUICKSORT(A, q + 1, r)  // recursively sort the high side
```

a) O(log n)

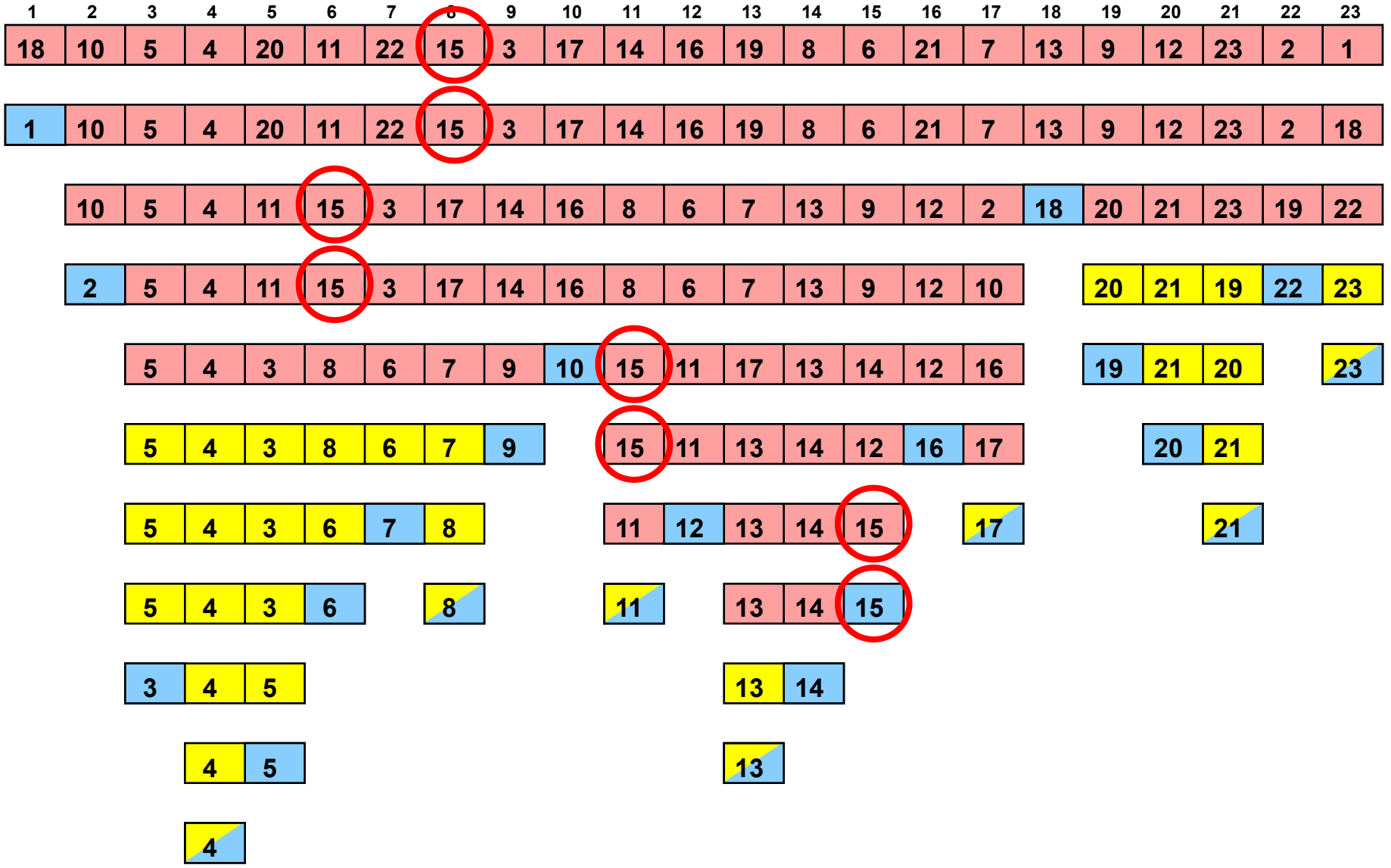
b) O(n)

c) O(n log n)

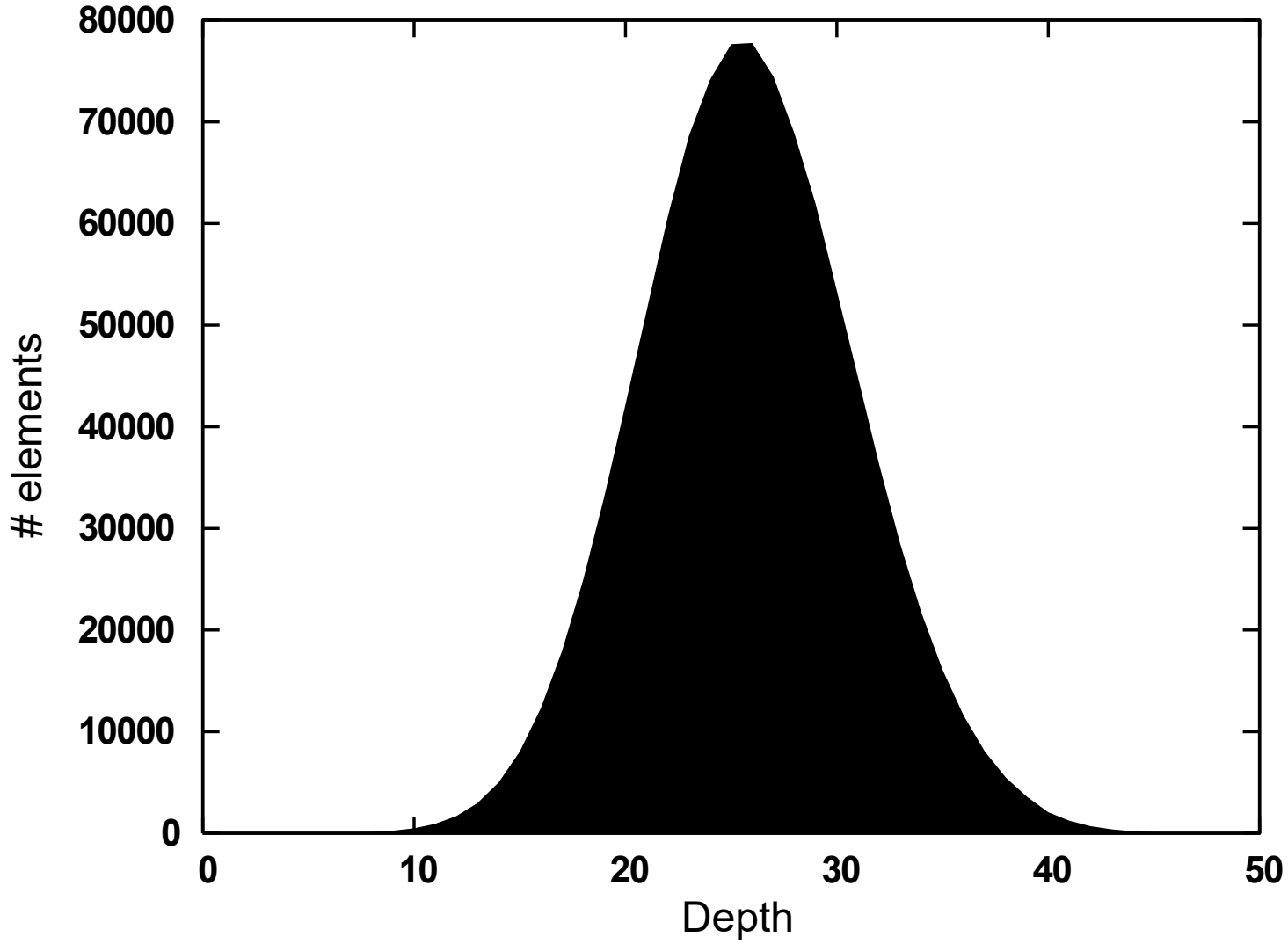
d) O(n^2)

e) Don't know

Quicksort – recursion for element 15



Quicksort – depths for an input of size $n \approx 2^{20}$



Randomized-Quicksort

```
RANDOMIZED-PARTITION(A, p, r)
1  i = RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)

RANDOMIZED-QUICKSORT(A, p, r)
1  if p < r
2       q = RANDOMIZED-PARTITION(A, p, r)
3       RANDOMIZED-QUICKSORT(A, p, q − 1)
4       RANDOMIZED-QUICKSORT(A, q + 1, r)
```

Expected time $O(n \cdot \log n)$

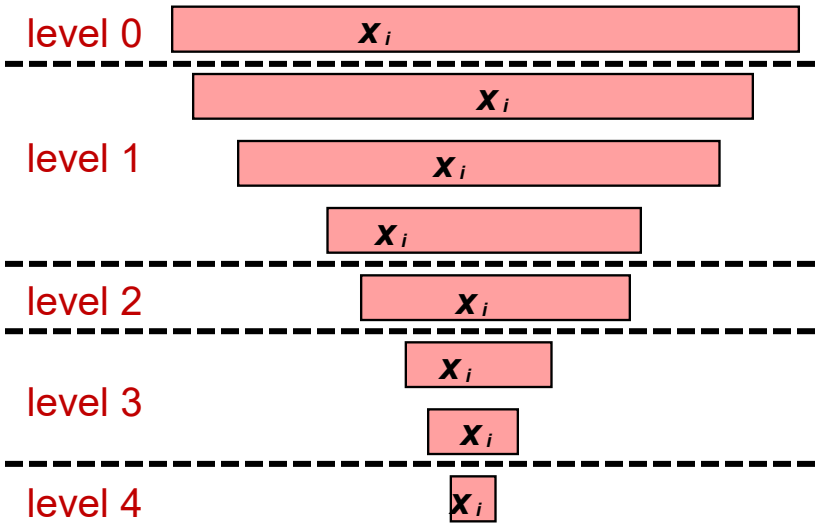
Expected number of coin-flips before you have seen 3 tails?

a) None of the below
b) Don't know
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) 9

Theorem

Expected number of coin flips to get ***k*** tails is ***2k***

Randomized-Quicksort – analysis



level j arrays of length $\le n(3/4)^j$

- An array is in level j if and only if it has $n(3/4)^{j+1} <$ length $\le n(3/4)^j$
- A partition is **good** if both subproblems ≤ 3/4 of the elements (i.e., both advance by at least one level) – probability ≥ 0.5
- x_i expected ≤ 2 times in each level (can also skip levels)
- Expected depth of $x_i \le 2 \cdot \log_{4/3} n$

Randomized-Quicksort – analysis

Expected time for randomized quicksort

= $O(\Sigma_{i=1..n}$ expected depth of input $x_i)$

= $O(\Sigma_{i=1..n} \log n)$

= $O(n \cdot \log n)$ □

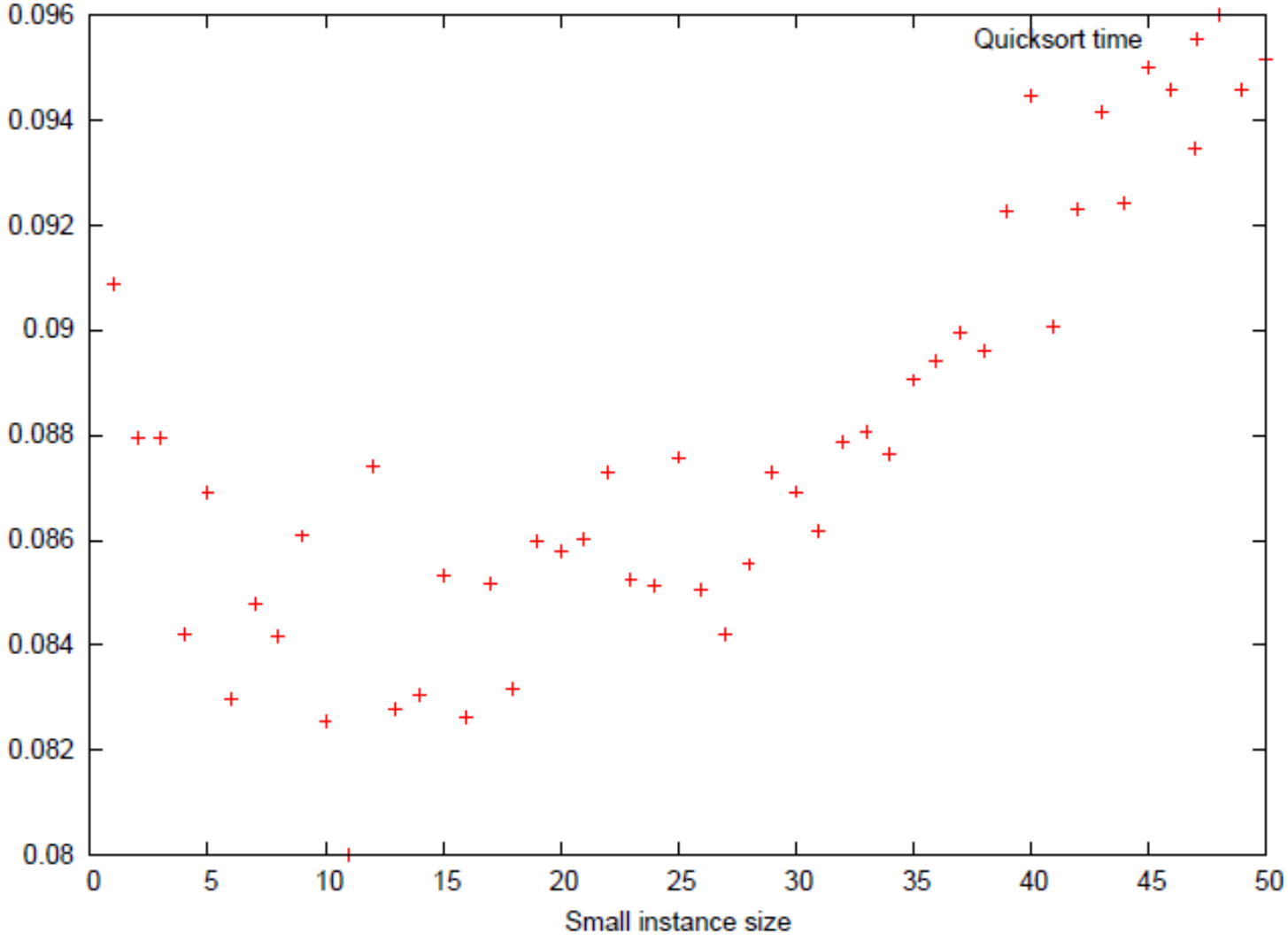
Sorting algorithms

Algorithm	Worst-Case Time
Heap-Sort Merge-Sort	$O(n \cdot \log n)$
Insertion-Sort Selection-Sort Quicksort (Deterministic and randomized)	$O(n^2)$

Algorithm	Expected time
Randomized-Quicksort	$O(n \cdot \log n)$

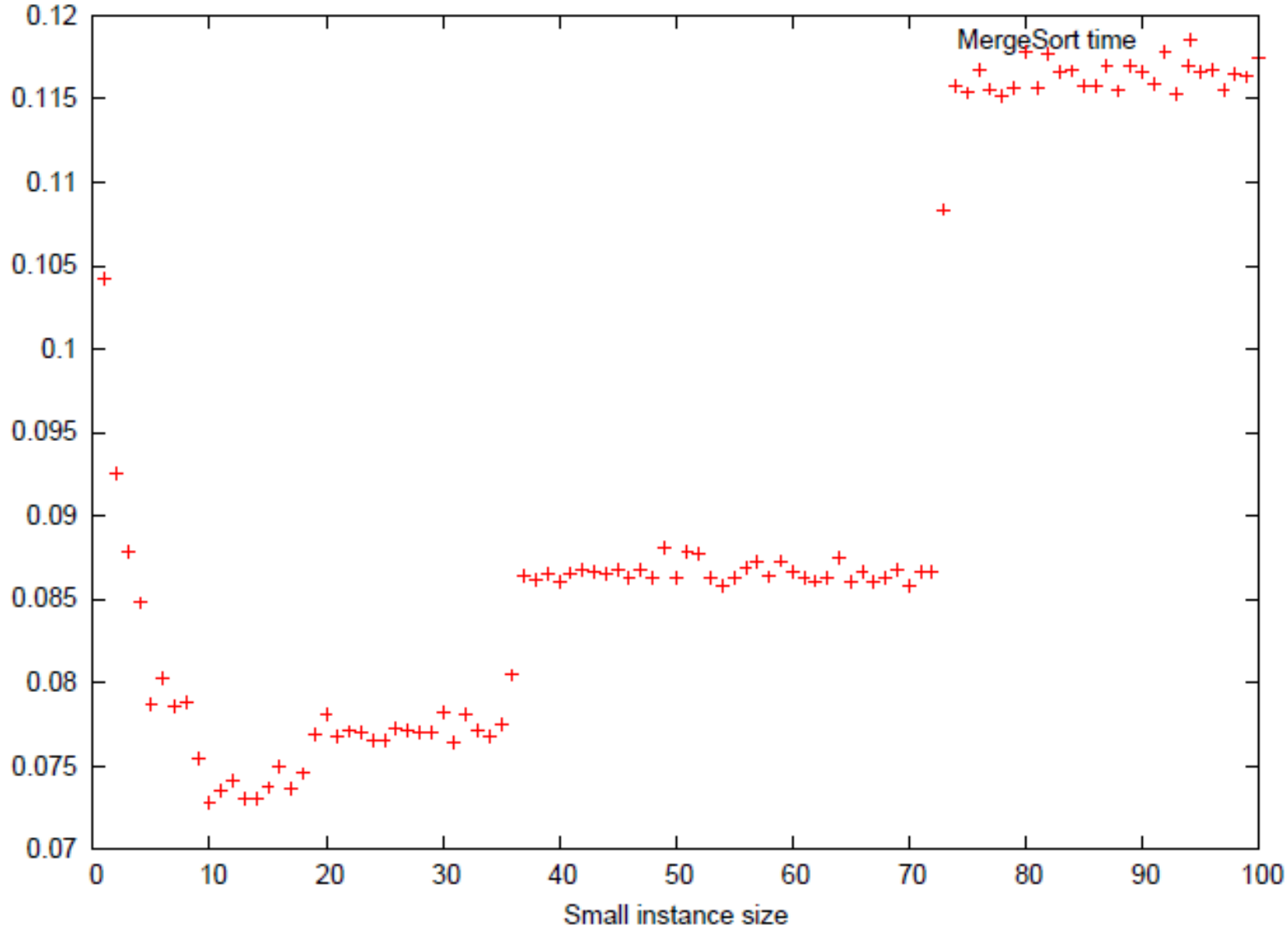
Sorting
– some experimental results

Quicksort + Insertion-sort



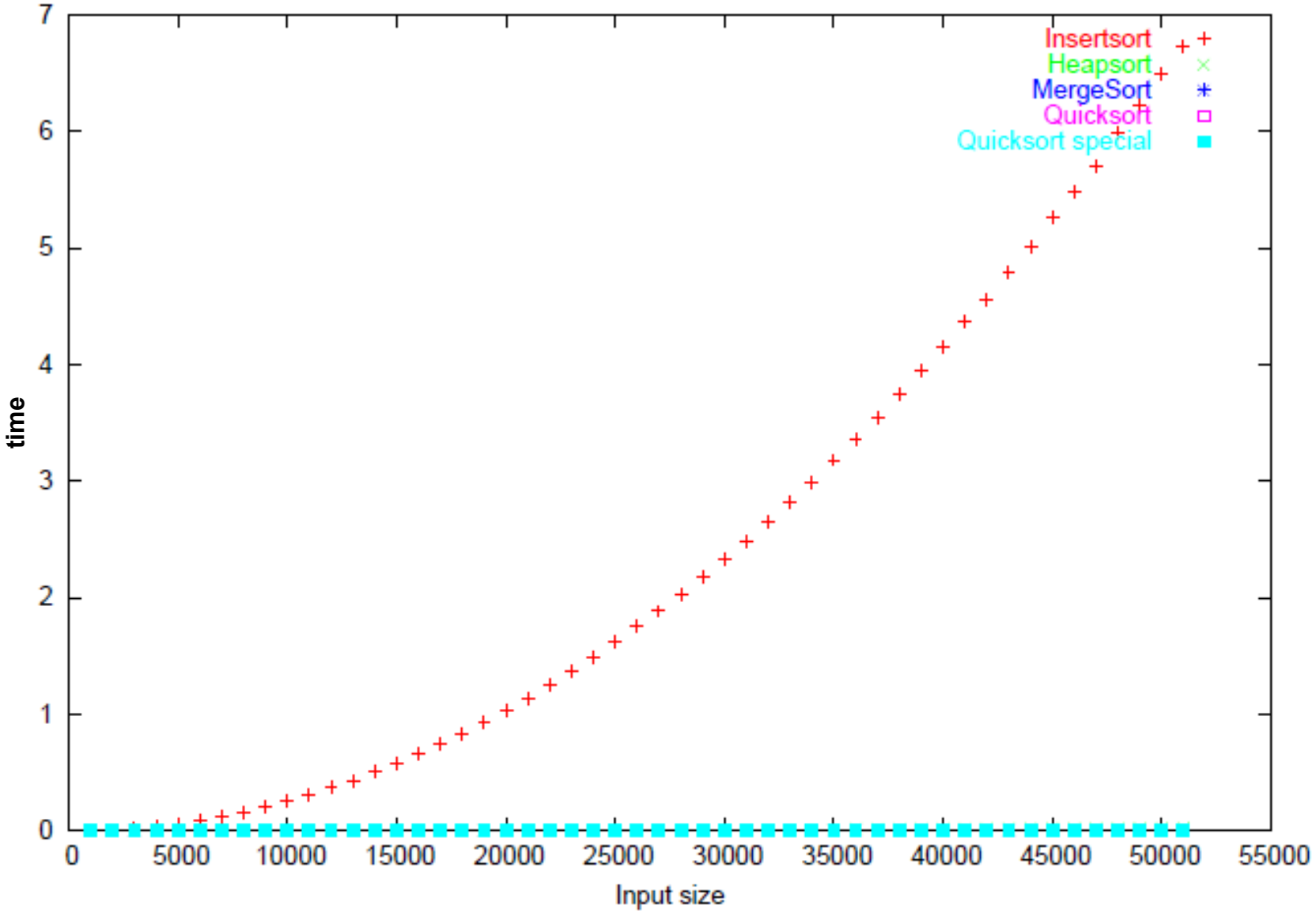
Use **insertion-sort** for small subproblems n = 300.000 random elements

Merge-sort + Insertion-sort

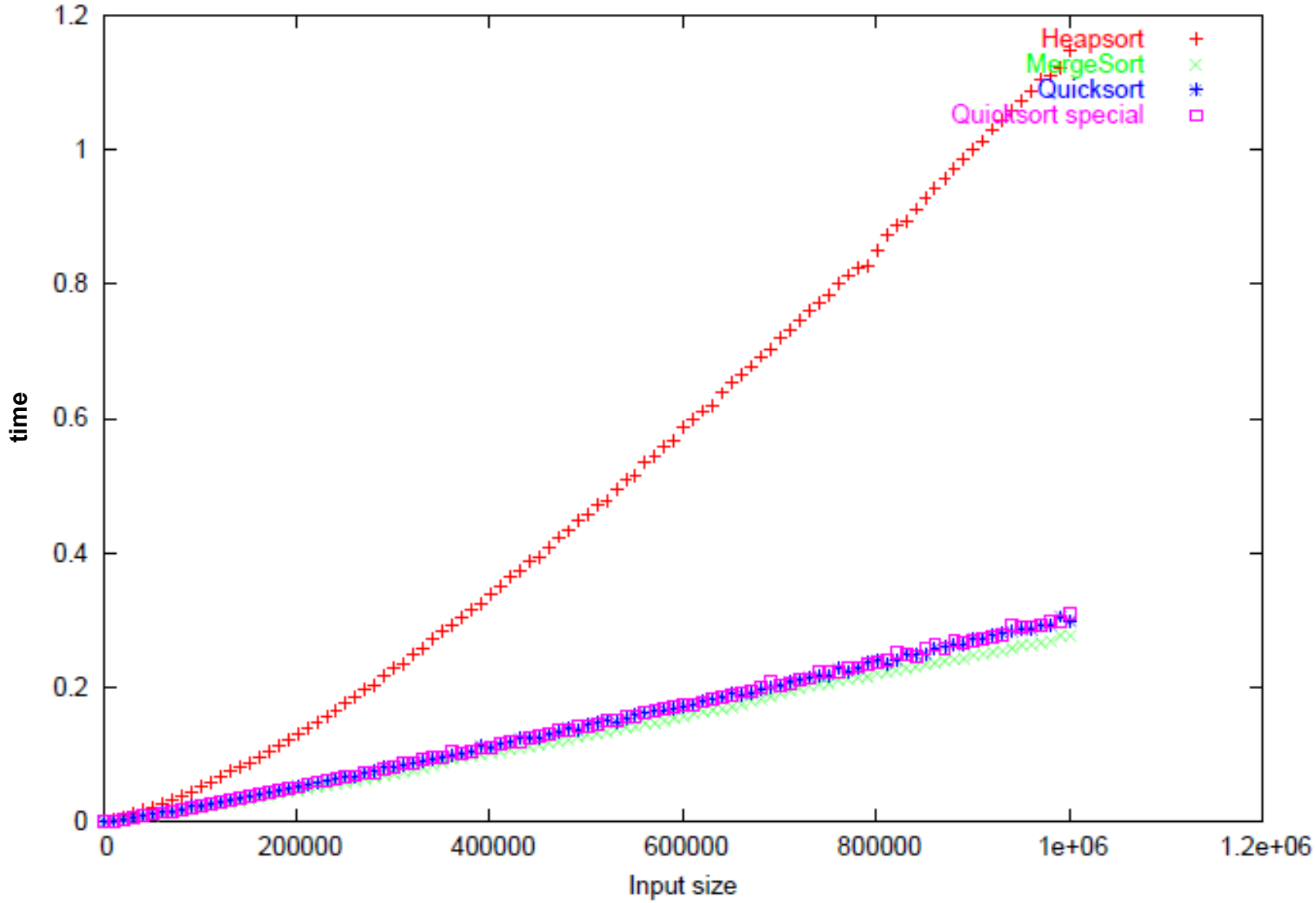


Use **insertion-sort** for small subproblems n = 300.000 random elements

Sorting algorithms



Sorting algorithms



Sorting in practice

Often **QuickSort** is the standard (C, C++, Java)

- Inplace (requires no additional arrays)
- Fast

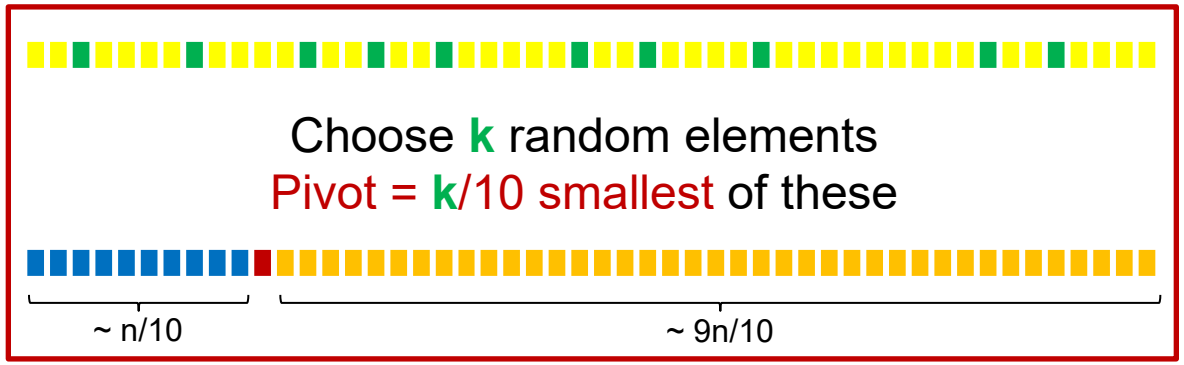
...with some twists

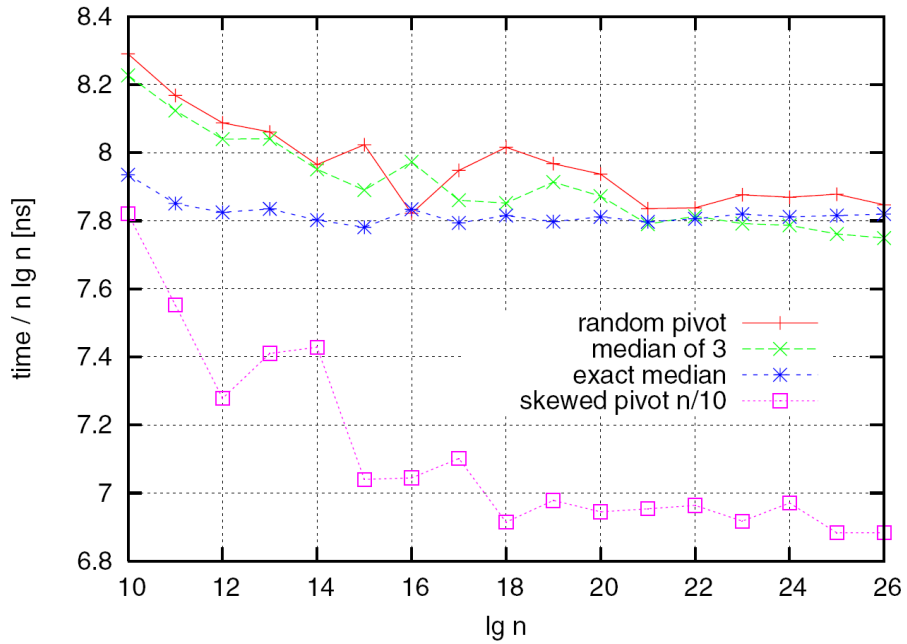
- use **Insertion-Sort** for small subproblems
- use **Merge-Sort** if QuickSort takes too long time
- Select pivot as the median of O(1) (random) elements

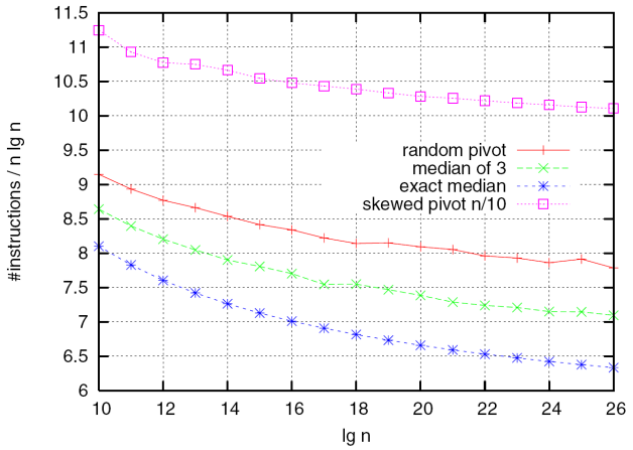
How Branch Mispredictions Affect Quicksort

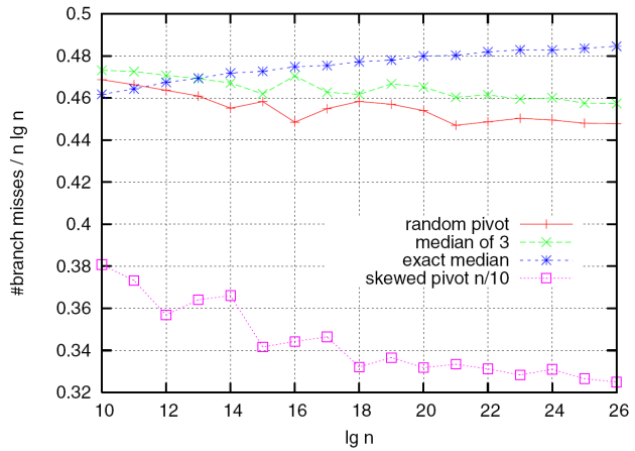
Kanela Kaligosi and Peter Sanders

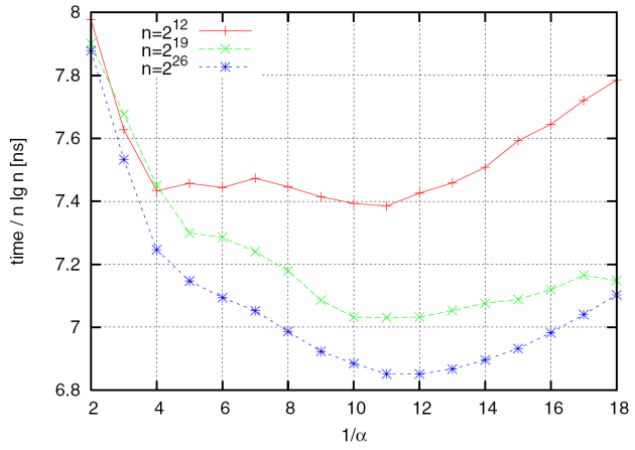
14th Annual European Symposium on Algorithms (ESA 2006)
doi: 10.1007/11841036_69











3GHz Pentium 4 Prescott, GCC 3.3, PAPI, avg. 100 runs with random permutations of [1..n]

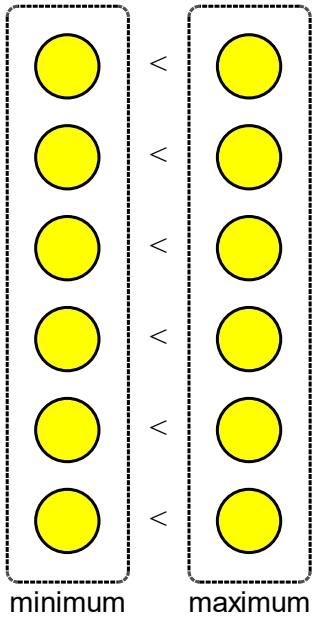
Algorithms and Data Structures

Randomized-select
[CLRS, Chapter 9.1-9.2]

Computation of minimum and maximum

- To find the *minimum* (or maximum) of n elements requires $n - 1$ comparisons

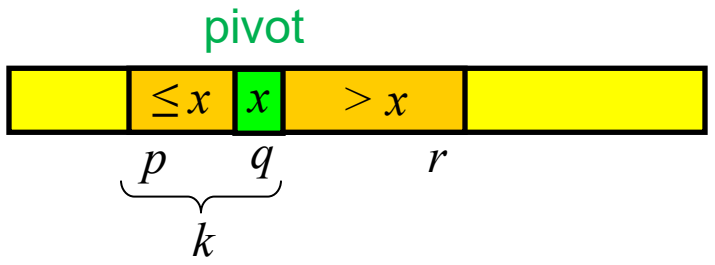
- To find both *the minimum and the maximum* of n elements requires $3/2 \cdot n - 2$ comparsisons



Randomized-Select

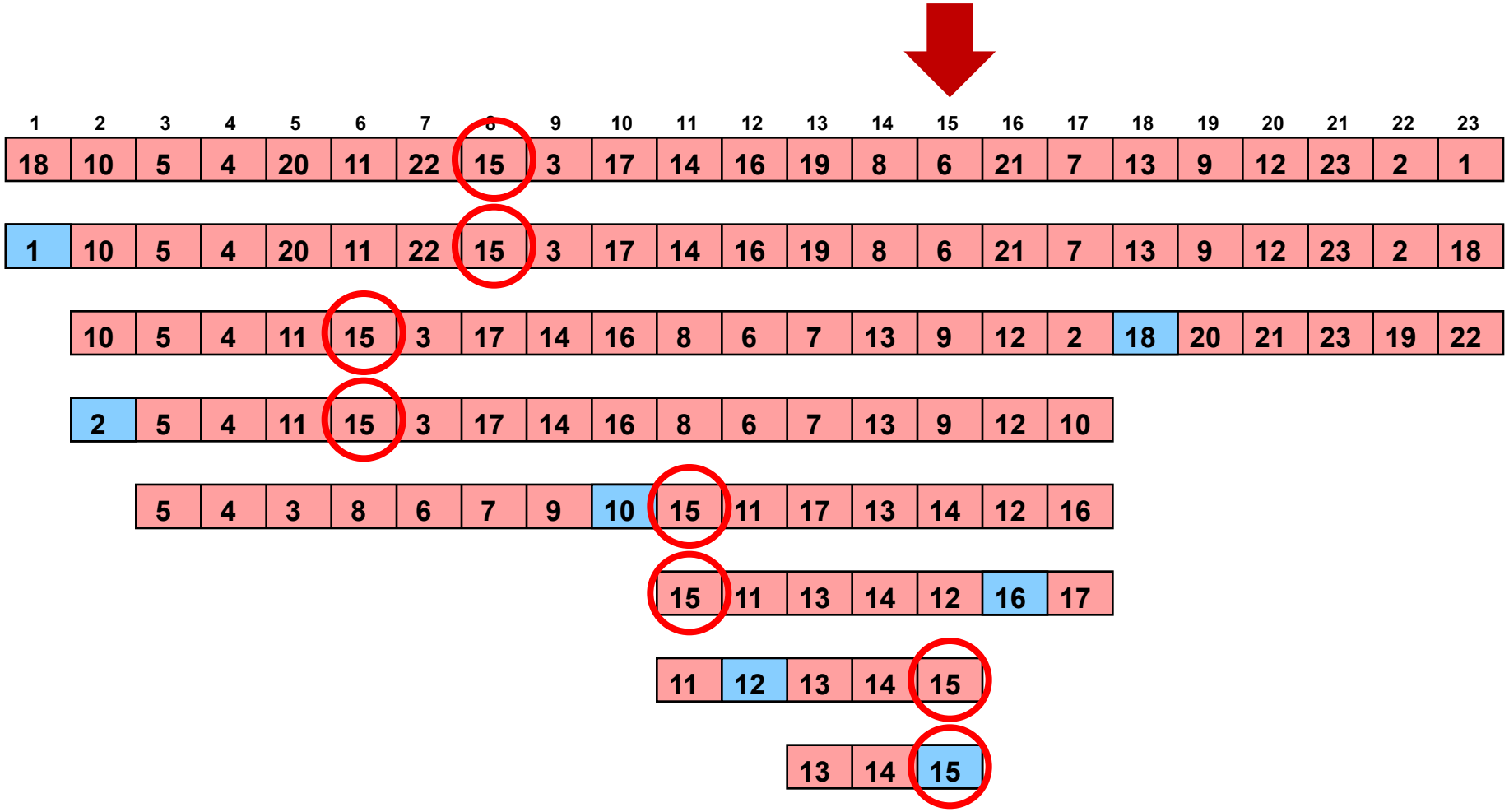
Find the i^{th} smallest element in $A[p..r]$ $(1 \le i \le r\text{-}p\text{+}1)$

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2        return A[p]        // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]        // the pivot value is the answer
7  elseif i < k
8        return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```



C. A. R. Hoare, *Algorithm 65: Find,* Communications of the ACM, 1961, doi: 10.1145/366622.366647

Randomized-Select 15

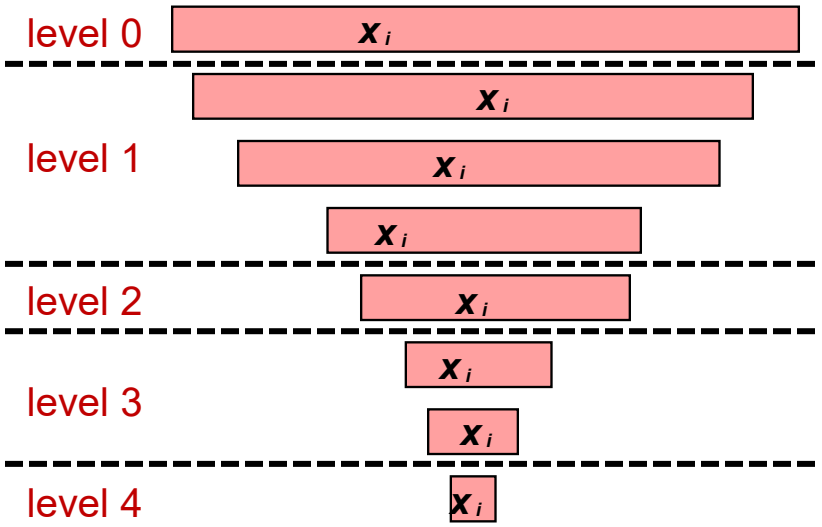


Worst-case time for Randomized-Select ?

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]        // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6       return A[q]        // the pivot value is the answer
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```

a) O(log n)
b) O(n)
c) O(n log n)
d) O(n^2)
e) Don't know

Randomized-Select – analysis



level j arrays of length $\le n(3/4)^j$

Expected time for randomized select

= O(Σ_j **expected time for** level j)

= O(Σ_j $n \cdot (3/4)^j$ · **expected # arrays** in level j)

= O(Σ_j $n \cdot (3/4)^j$ · 2) ← $\sum_{i=0}^{\infty} c^i = \frac{1}{1-c}$ for $0 < c < 1$

= **O(n)**

□

Selection

Algorithm	Time
Randomized-Select [CLRS, Chapter 9.2]	O(n) expected O(n^2) worst-case
Deterministic-Select [CLRS, Chapter 9.3]	O(n) worst-case

Algorithms and Data Structures

Lower bound for comparison-based sorting,
Counting-Sort, Radix-Sort, Bucket-Sort
[CLRS, Chapter 8]

Sorting algorithms
(comparison based)

Algorithm	Worst-Case Time
Heap-Sort Merge-Sort	$O(n \cdot \log n)$
Insertion-Sort Quicksort (Deterministic and randomized)	$O(n^2)$

Algorithm	Expected time
Randomized Quicksort	$O(n \cdot \log n)$

What is a sorting algorithm?

Deterministic algorithm

- On a given input the algorithm always does the same

Randomized algorithm

- Algorithm can make random choices
- Execution depends on both input *and* random choices

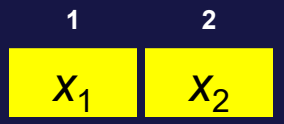
Output

- a permutation of the input

Comparison based algorithm

- output depends only on the outcome of comparing input

comparisons to correctly sort 2 elements ?



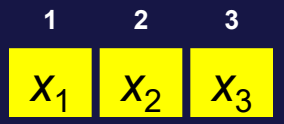
a) No comparisons

b) At least 1 comparison

c) At least 2 comparisons

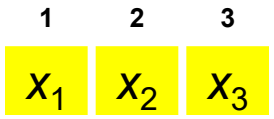
d) Don't know

comparisons to correctly sort 3 elements ?



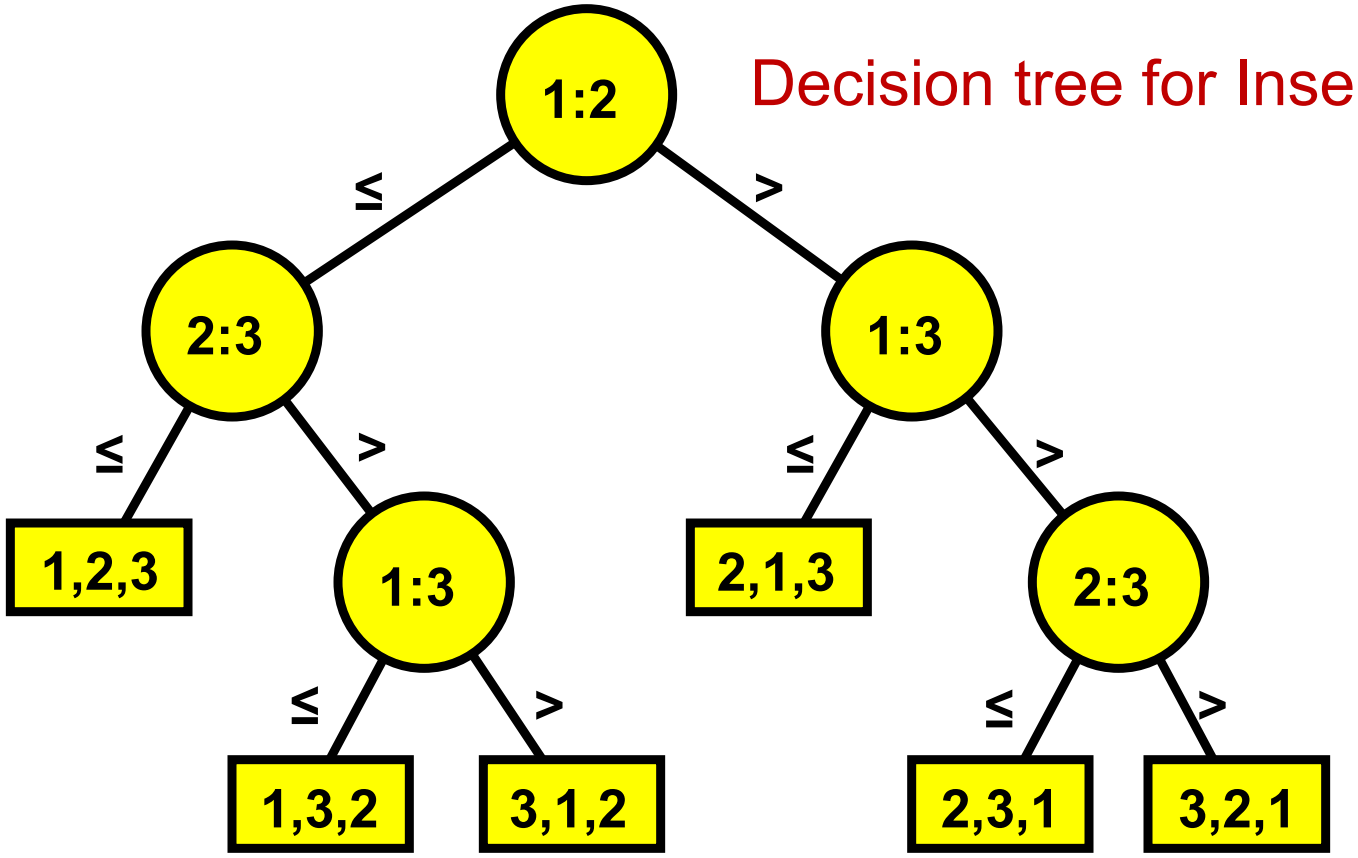
a) No comparisons
b) At least 1 comparison
c) At least 2 comparisons
d) At least 3 comparisons
e) At least 4 comparisons
f) At least 5 comparisons
g) Don't know

Comparisons for Insertion-Sort




```
INSERTION-SORT(A, n)
1  for i = 2 to n
2      key = A[i]
3      // Insert A[i] into the sorted subarray A[1 : i − 1]
4      j = i − 1
5      while j > 0 and A[j] > key
6          A[j + 1] = A[j]
7          j = j − 1
8      A[j + 1] = key
```

Comparison-based sorting: Lower bound



Decision tree for Insertion-Sort for $n = 3$

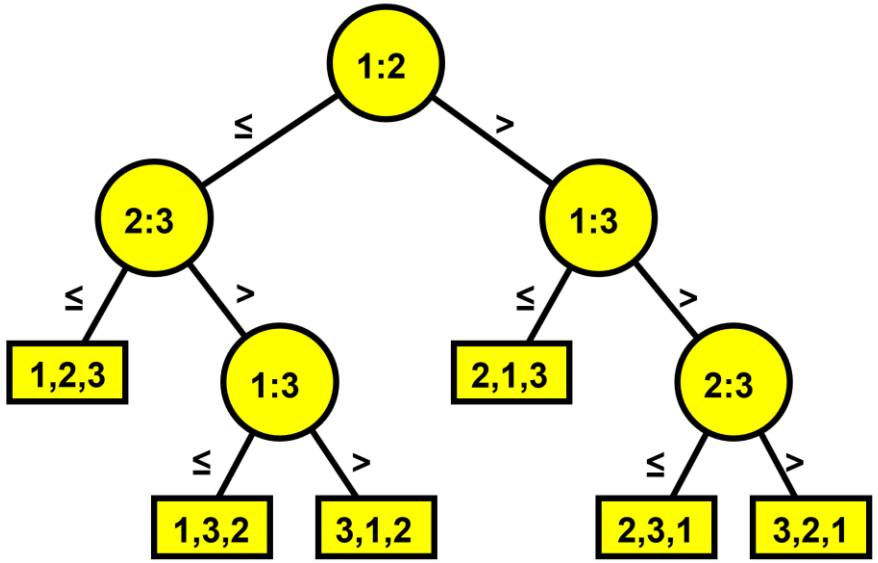
Internal node $i : j$ comparison of x_i and x_j

Leaves describe output permutation

Comparison-based sorting: Lower bound

$n!$ different outputs ≤ leaves

tree of height h has ≤ 2^h leaves



$$n! \le 2^h$$

$$h \ge \log(n!) \ge \log\left(\left(\frac{n}{2}\right)^{n/2}\right) = \frac{n}{2}\log\frac{n}{2} = \frac{n}{2}(\log n - 1) \ge \frac{n}{4}\log n$$

for $n \ge 4$

Worst-case $\Omega(n \cdot \log n)$ comparisons

comparisons to sort 10 elements ?

a) At least 9 comparisons
b) At least 10 comparisons
c) At least 21 comparisons
d) At least 22 comparisons
e) At least 23 comparisons
f) Don't know

n	1	2	3	4	5	6	7	8	9	10	11	12
$n!$	1	2	6	24	120	720	5.040	40.320	362.880	3.628.800	39.916.800	479.001.600

d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2^d	1	2	4	8	16	32	64	128	256	512	1.024	2.048	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576

d	21	22	23	24	25	26	27	28	29
2^d	2.097.152	4.194.304	8.388.608	16.777.216	33.554.432	67.108.864	134.217.728	268.435.456	536.870.912

comparisons to sort 1-12 elements ?

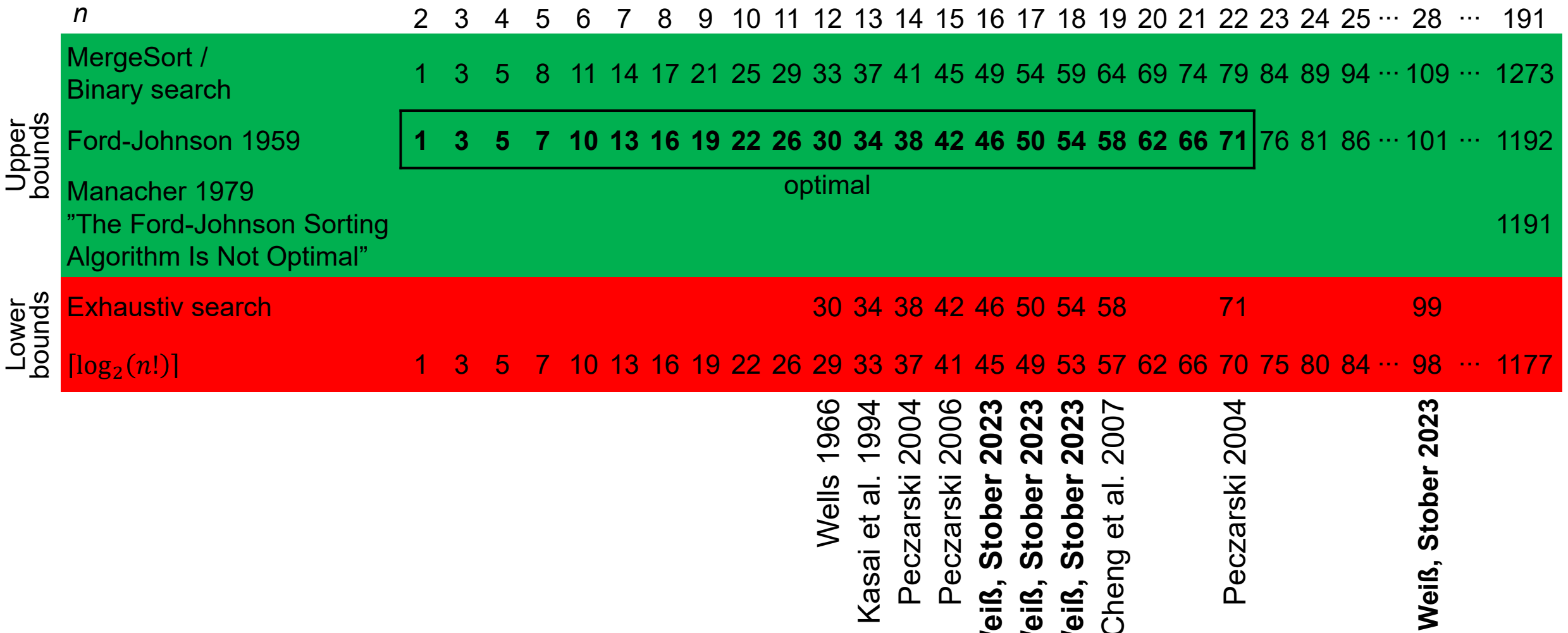
n	1	2	3	4	5	6	7	8	9	10	11	12
≥ comparisons	0	1	3	5	7	10	13	16	19	22	26	29

n	1	2	3	4	5	6	7	8	9	10	11	12
$n!$	1	2	6	24	120	720	5.040	40.320	362.880	3.628.800	39.916.800	479.001.600

d	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2^d	1	2	4	8	16	32	64	128	256	512	1.024	2.048	4.096	8.192	16.384	32.768	65.536	131.072	262.144	524.288	1.048.576

d	21	22	23	24	25	26	27	28	29
2^d	2.097.152	4.194.304	8.388.608	16.777.216	33.554.432	67.108.864	134.217.728	268.435.456	536.870.912

Sorting – Fewest # comparisons



Florian Stober, Armin Weiß, *Lower Bounds for Sorting 16, 17, and 18 Elements*, ALENEX 2023, doi: 10.1137/1.9781611977561.ch17

Sorting integers

...exploit that integers are binary numbers

Counting-Sort:

Input: A, output: B, integers from $\{0,1, \ldots,k\}$

```
COUNTING-SORT(A, n, k)
 1  let B[1 : n] and C[0 : k] be new arrays
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to n
 5      C[A[j]] = C[A[j]] + 1
 6  // C[i] now contains the number of elements equal to i.
 7  for i = 1 to k
 8      C[i] = C[i] + C[i − 1]
 9  // C[i] now contains the number of elements less than or equal to i.
10  // Copy A to B, starting from the end of A.
11  for j = n downto 1
12      B[C[A[j]]] = A[j]
13      C[A[j]] = C[A[j]] − 1    // to handle duplicate values
14  return B
```

Worst-case time O(n + k)

Which algorithms are <u>not</u> stable ?

a) Insertion-Sort
b) Heap-Sort
c) Quicksort
d) Merge-Sort

e) Heap-Sort and Quicksort
f) Insertion-Sort and Heap-Sort
g) Merge-Sort and Quicksort
h) Insertion-Sort and Quicksort
i) Don't know

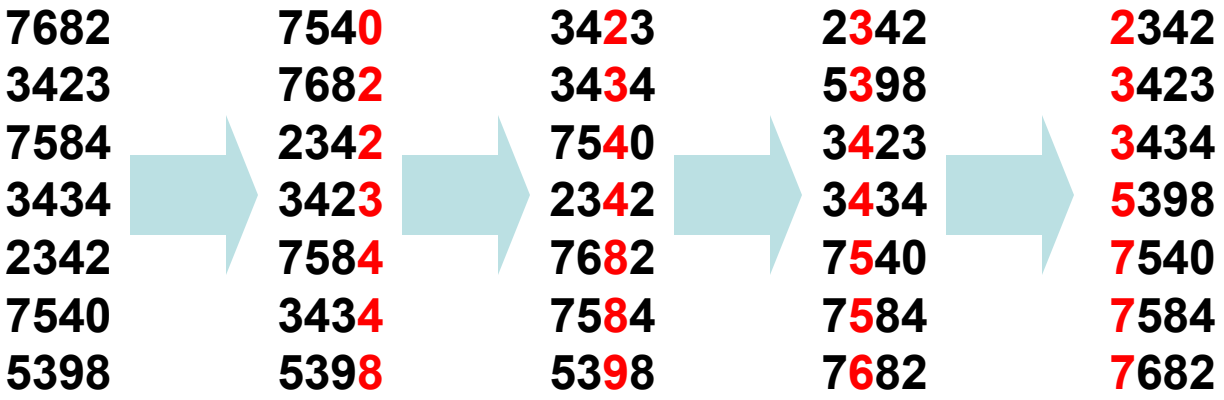
Radix-Sort:

Input: array *A*, integers with *d* digits from {0,1,...,*k*}

RADIX-SORT(A, n, d)

1 **for** $i = 1$ **to** d

2 use a stable sort to sort array $A[1:n]$ on digit i



Worst-case time $O(d \cdot (n + k))$

Sorting A5 window envelopes

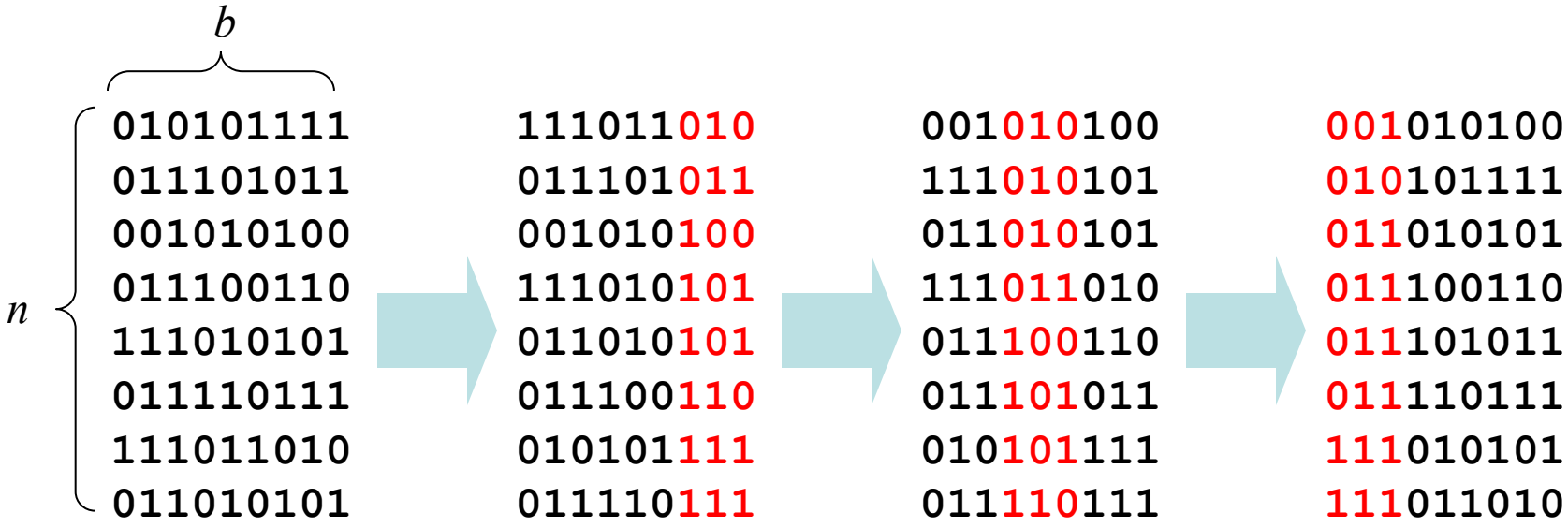
Radix-Sort at work at POST

Step 1: Sort by number

Step 2: Sort by route

Radix-Sort:

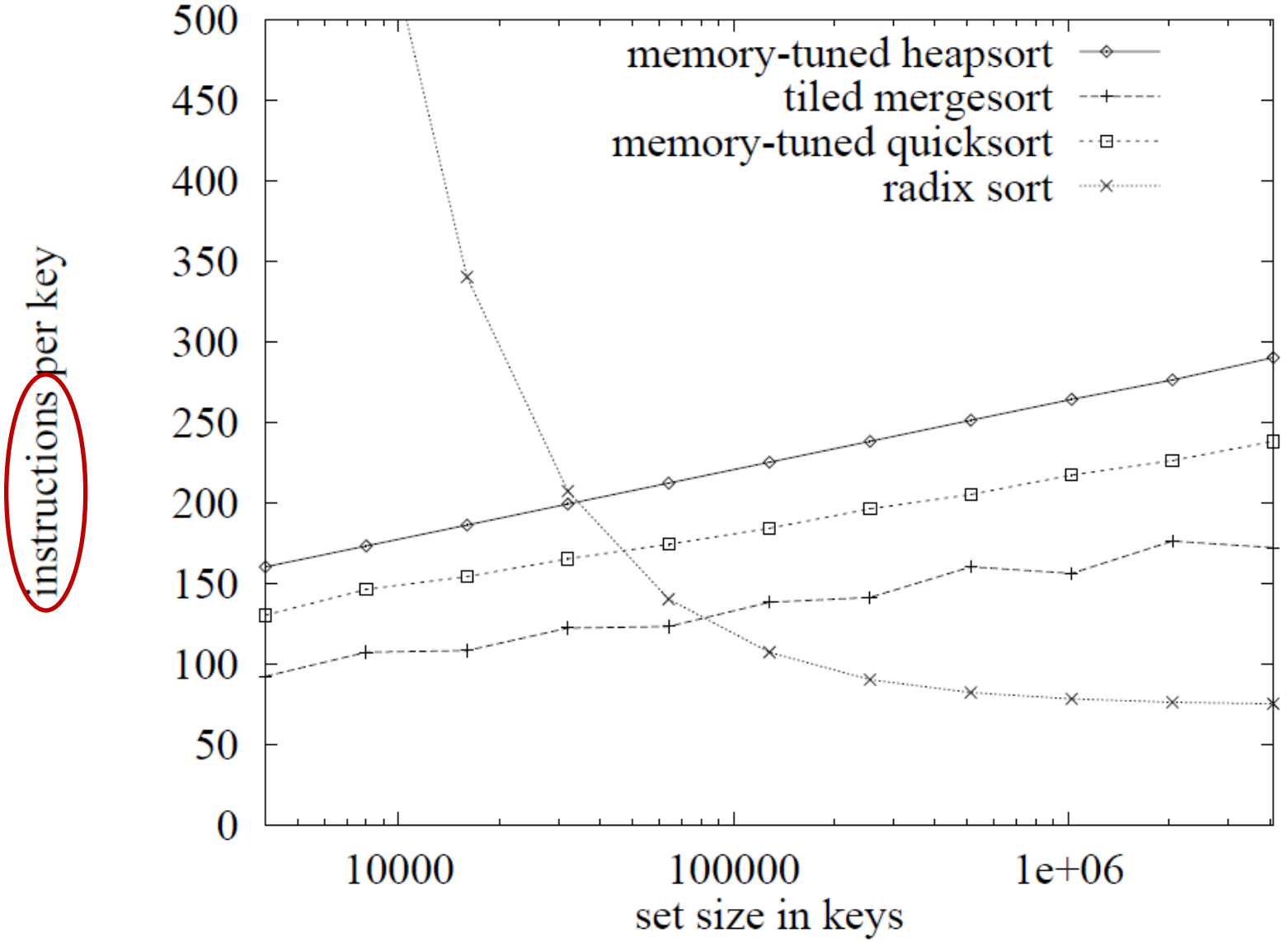
Input: array A, n integers with b bits



n = 8, k = 7 (3 bits = log n bits), d = 3 (3 x 3 bits)

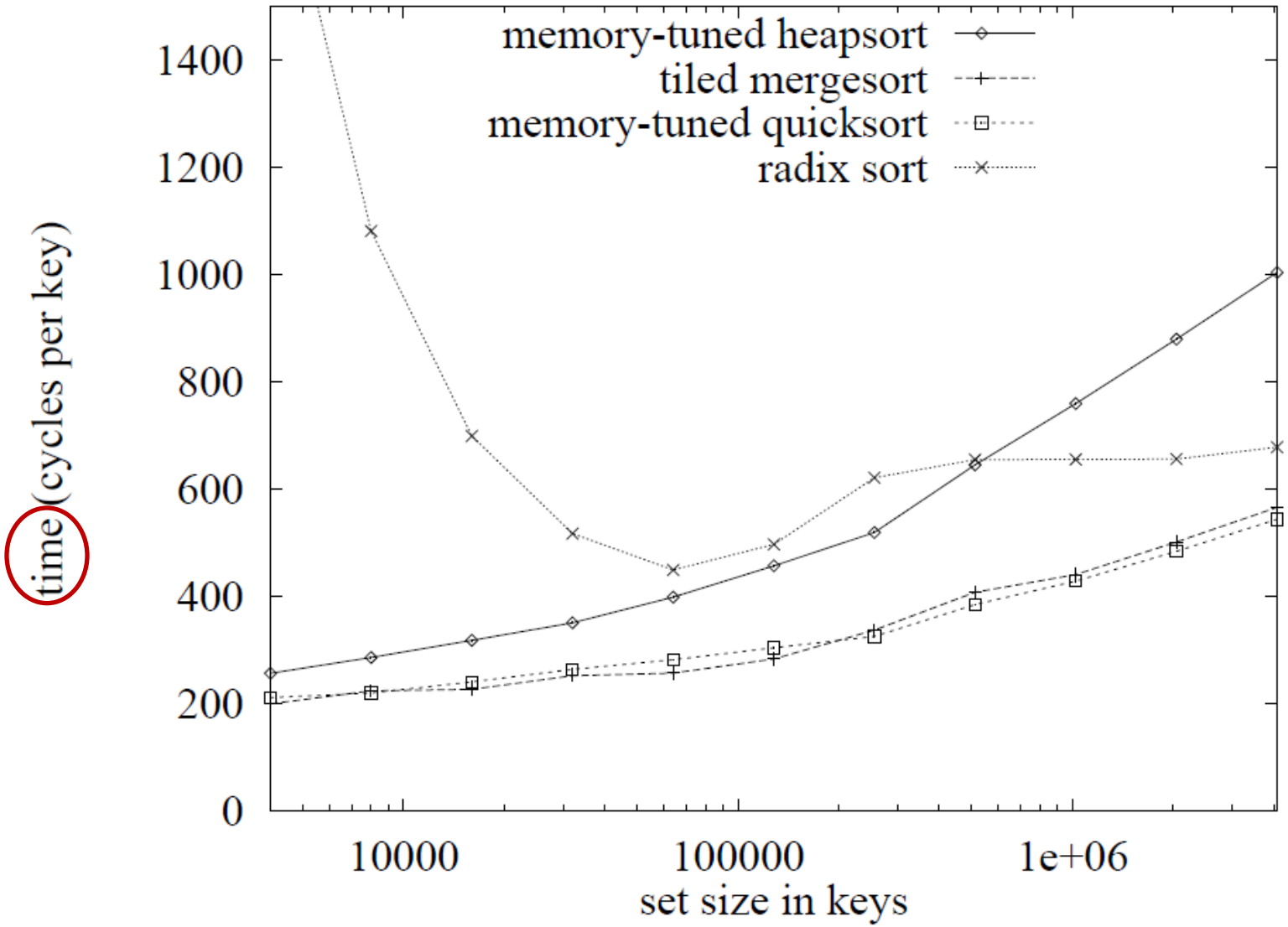
Input n integers with b bits: Worst-case time $O(n \cdot b / \log n)$

Radix-Sort: Experiments



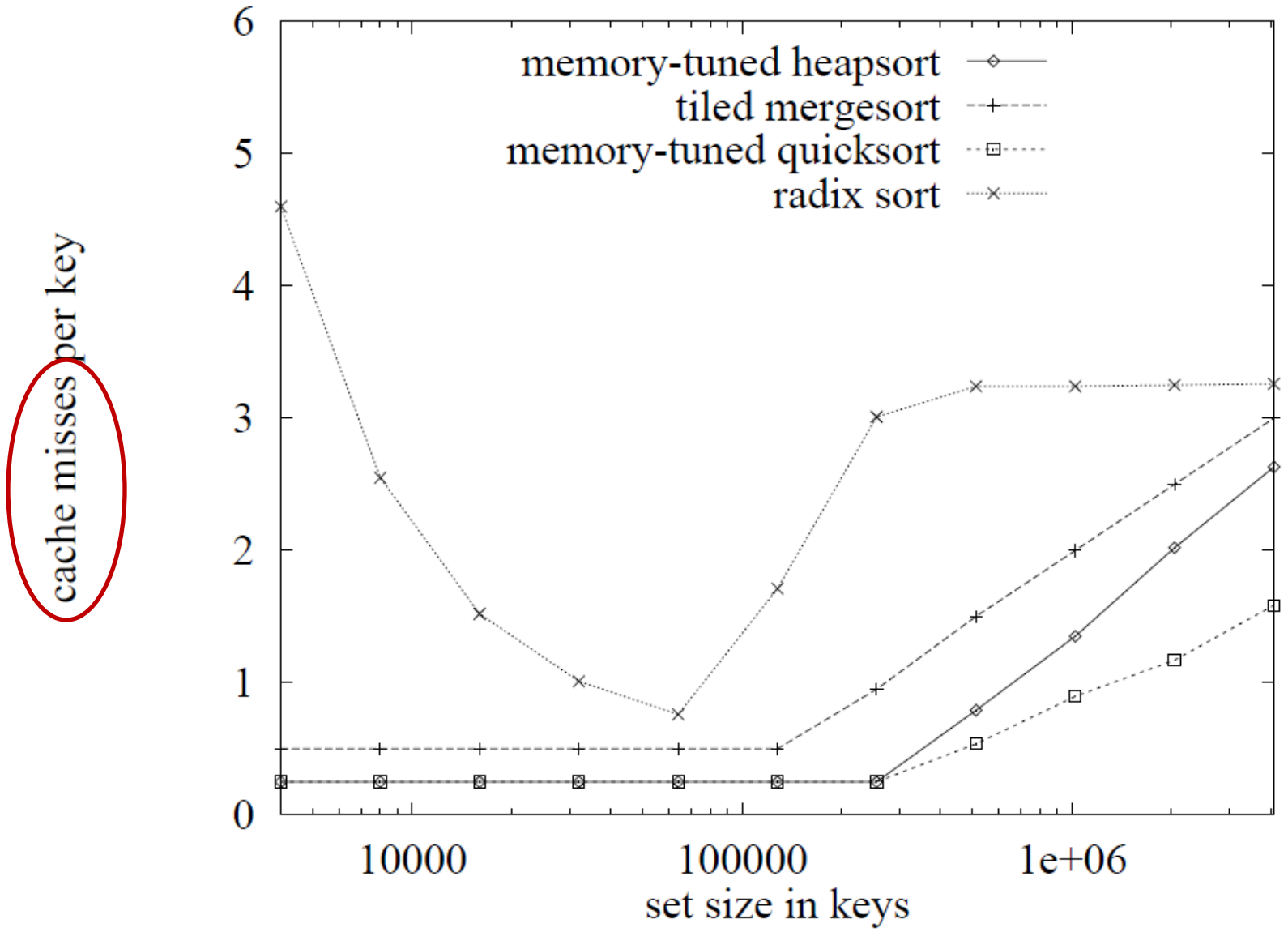
A. LaMarca, R. E. Ladner, *The Influence of Caches on the Performance of Sorting,* SODA 1997, dl.acm.org/doi/10.5555/314161.314324

Radix-Sort: Experiments



LaMarca, Ladner ’97

Radix-Sort: Experiments

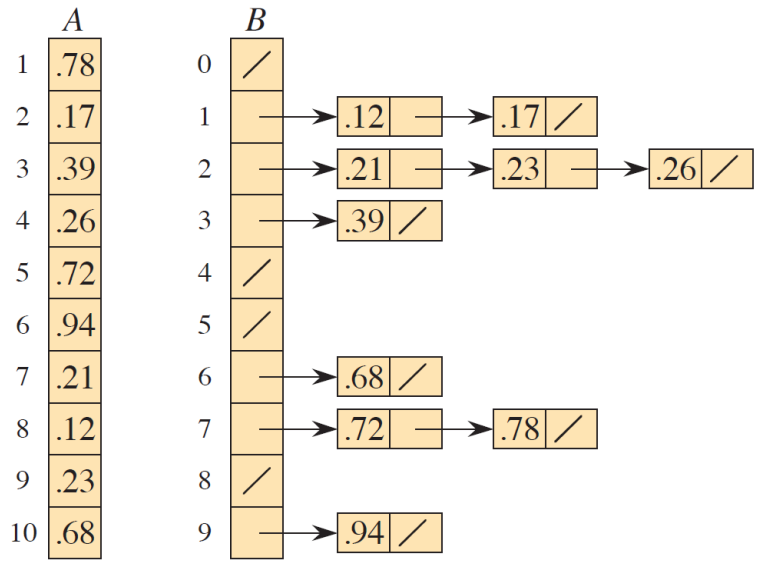


LaMarca, Ladner ’97

Bucket-Sort:

Input: A, real numbers from [0..1[

```
BUCKET-SORT(A, n)
1  let B[0 : n − 1] be a new array
2  for i = 0 to n − 1
3      make B[i] an empty list
4  for i = 1 to n
5      insert A[i] into list B[⌊n · A[i]⌋]
6  for i = 0 to n − 1
7      sort list B[i] with insertion sort
8  concatenate the lists B[0], B[1], . . . , B[n − 1] together in order
9  return the concatenated lists
```



Expected time O(n) – for random inputs

not curriculum

Sorting – state-of-the-art

Best known sorting bound on RAM model (with unlimited word size) is a randomized algorithm using O(n) space and has expected running time

$$O\left(n\sqrt{\log\log n}\right)$$

If this can be achieved without randomization is unknown.
O(n loglog n) is known. Is it possible to get expected O(n) time?
With randomization and assuming word-size $\Omega(\log^{2+\varepsilon} n)$ there exists a randomized algorithm with expected O(n) time.

Han, Thorup, *Integer Sorting in* $O\left(n\sqrt{\log\log n}\right)$ *Expected Time and Linear Space*, FOCS 2002, doi: 10.1109/SFCS.2002.1181890

Andersson, Hagerup, Nilsson, Raman, *Sorting in linear time?,* STOC 1995, doi: 10.1145/225058.225173

Belazzougui, Brodal, Nielsen, *Expected Linear Time Sorting for Word Size* $\Omega(\log^2 n \cdot \log\log n)$, SWAT 2014, doi: 10.1007/978-3-319-08404-6_3

not curriculum

Sorting algorithms in Java, Python, C++ and C

	Java (JDK SE 24.0.2)	Python (CPython 3.13.7)	C++ (GCC 15.2)	C (GNU C)
Method	Java.util.Arrays.sort	sorted	std::sort	qsort
Documentation	https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/Arrays.html#sort(int[])	https://docs.python.org/3/library/functions.html#sorted	http://www.cplusplus.com/reference/algorithm/sort/	https://sourceware.org/glibc/manual/latest/html_node/Array-Sort-Function.html
Algorithm	**Dual Pivot Quicksort** Insertion-Sort for few elements Merge-Sort for few sorted “runs” Counting-Sort for BYTE and SHORT input	**Power-Sort** Merge-Sort variant that identifies sorted “runs” to reduce the number of merges	**Introsort** Quicksort (max depth 2·log n) otherwise change to Heap-Sort Insertion-Sort ≤ 16 elements	**Quicksort** Not recursive
Source code	Install JDK from www.oracle.com/java/technologies/downloads/ Fil java.base\java\util\DualPivotQuicksort.java from C:\Program Files\Java\jdk-25\lib\src.zip	https://github.com/python/cpython/blob/master/Objects/listobject.c https://github.com/python/cpython/blob/master/Objects/listsort.txt	https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_algo.h	https://code.woboq.org/userspace/glibc/stdlib/qsort.c.html

not curriculum

Binary heaps in Java, Python, C++

	Java (JDK 25)	Python (CPython 3.13.7)	C++ (GCC 15.2)
Method	java.util.PriorityQueue	heapq	std::priority_queue
Documentation	https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/PriorityQueue.html	https://docs.python.org/3/library/heapq.html	http://www.cplusplus.com/reference/queue/priority_queue/
Source code	Install JDK from www.oracle.com/java/technologies/downloads/ File java.base\java\util\PriorityQueue.java from C:\Program Files\Java\jdk-25\lib\src.zip	https://github.com/python/cpython/blob/master/Lib/heapq.py	https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_heap.h

All three implementations use 0-indexed arrays, with

left(i) = $2 \cdot i + 1$ right(i) = $2 \cdot i + 2$ parent(i) = $\lfloor (i - 1) / 2 \rfloor$

Algorithms and Data Structures

Stacks, queues
[CLRS, Chapter 10]

[CLRS, Part 3] : Data structures

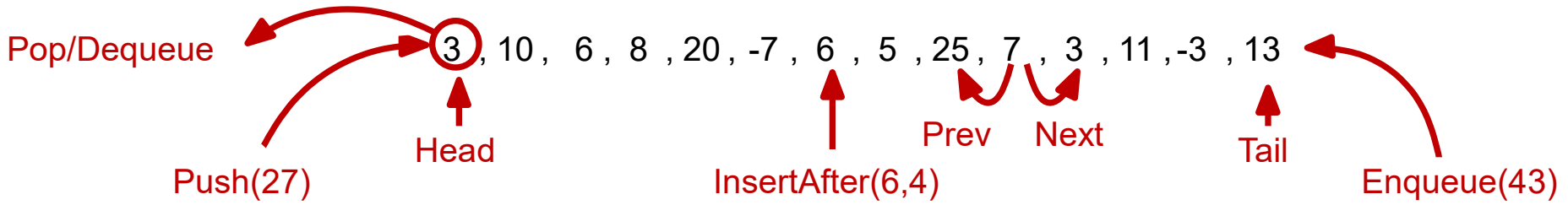
Maintain a structure for a
dynamic set of data

Abstract data structures for sets

	Operation	Return value	- Min-prioritety-queue	- Max-priority-queue	- Dictionary
Queries	**Minimum(*S*)**	pointer to element	●		
	Maximum(*S*)	pointer to element		●	
	Search(*S*, *x*)	pointer to element			●
	Member(*S*, *x*)	True or False			
	Successor(*S*, *x*)	pointer to element			
	Predecessor(*S*, *x*)	pointer to element			
Updates	**Insert(*S*, *x*)**	pointer to element	●	●	●
	Delete(*S*, *x*)	-			●
	DeleteMin(*S*)	element	●		
	DeleteMax(*S*)	element		●	
	Join(S_1, S_2)	set S			
	Split(*S*, *x*)	sets S_1 and S_2			

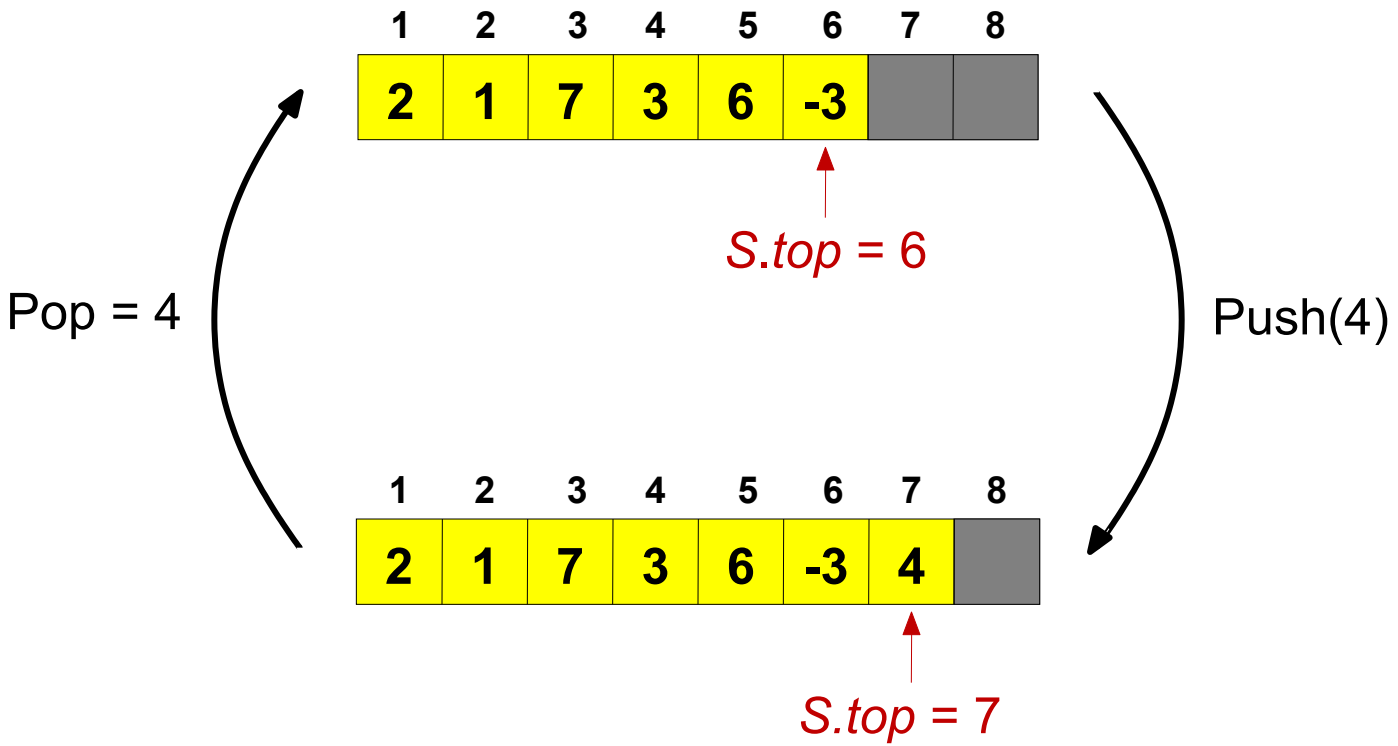
Abstract data structures for lists

	Operation	Return value	-Stack	-Queue
Queries	**Empty(*S*)**	True or False	●	●
	Head(*S*), Tail(*S*)	pointer to element		
	Next(*S*, *x*), Prev(*S*, *x*)	pointer to element		
	Search(*S*, *x*)	pointer to element		
Updates	**Push(*S*, *x*)**	-	●	
	Pop/Dequeue(*S*)	element	●	●
	Enqueue(*S*, *x*)	-		●
	Delete(*S*, *x*)	element		
	InsertAfter(*S*, *x*, *y*)	pointer to element		



Stack

Stack : array implementation



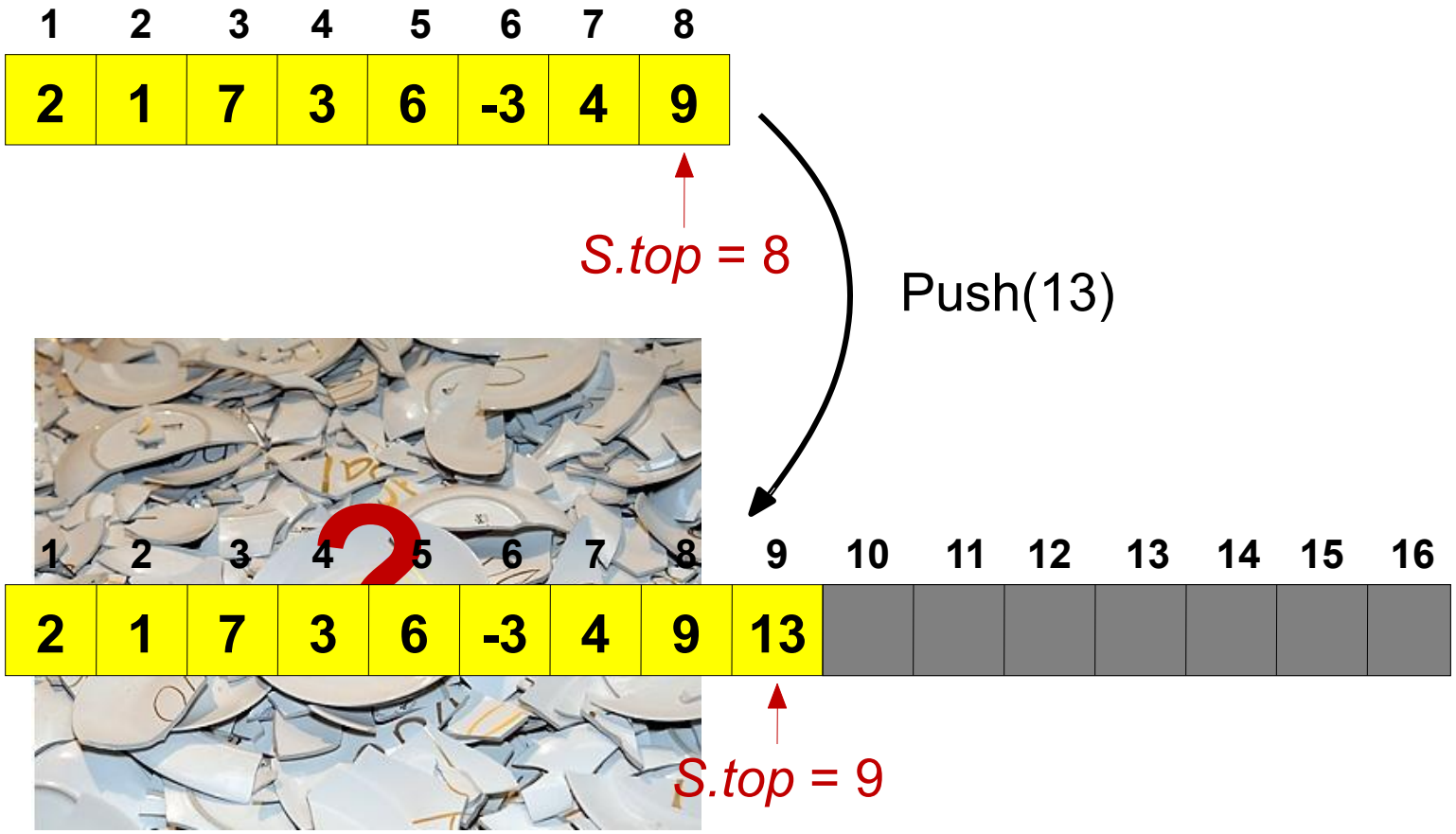

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

```
PUSH(S, x)
1  if S.top == S.size
2       error "overflow"
3  else S.top = S.top + 1
4       S[S.top] = x
```

```
POP(S)
1  if STACK-EMPTY(S)
2       error "underflow"
3  else S.top = S.top − 1
4       return S[S.top + 1]
```

Stack-Empty, Push, Pop : O(1) time

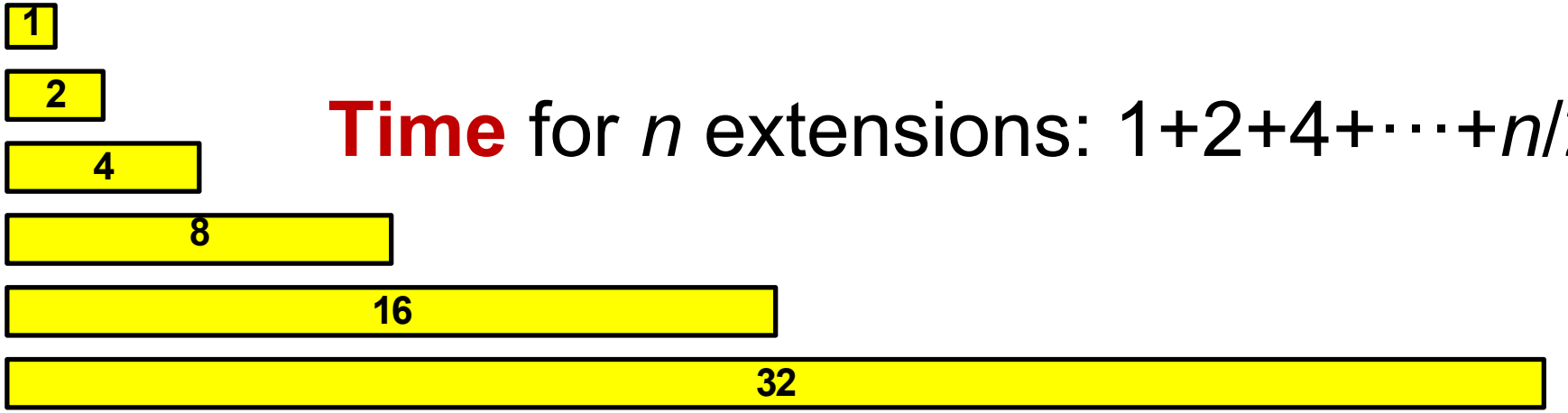
Stak : Overflow



Array doubling : O(*n*) time

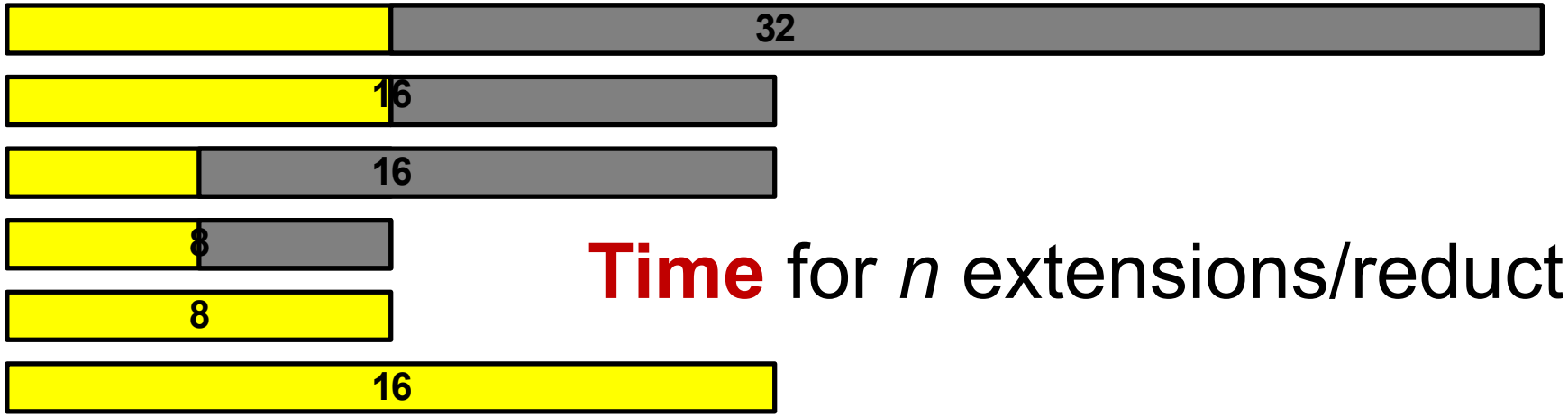
Array doubling

Double array when it is full



Time for n extensions: $1+2+4+\cdots+n/2+n = O(n)$

Halve the array when it is <1/4 full



Time for n extensions/reductions: $O(n)$

Array doubling + halving
– a general technique

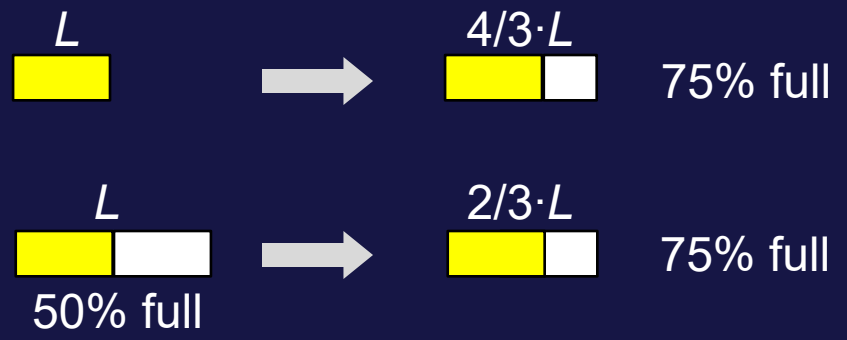
Time for n extensions/reductions is $O(n)$

Space $\leq 4 \cdot$ current number of elements

Array implementation of a stack:
n push and pop operations take time $O(n)$

Factor 4/3 increase

On overflow let the new size be 4/3 times the current size, and when the array becomes half full reduce the size to 2/3



What is maximal size of the array when it stores *n* elements ?

a) 3/2 · *n*
b) 4/3 · *n*
c) 2 · *n*
d) 8/3 · n
e) Don't know

not curriculum

Reallocation strategies in Java, Python and C++

Language	Java ArrayList (JDK SE 25)	Python list (CPython 3.15)	C++ vector (GCC 15.2)
On overflow	**+ 50 %**	**+ 12.5 %**	**+ 100 %**
Shrinking	No (can call trimToSize)	< 50 %	No (can call shrink_to_fit)
Documentation	https://docs.oracle.com/javase/25/docs/api/java/util/ArrayList.html	https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range	http://www.cplusplus.com/reference/vector/vector/
Source code	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\ArrayList.java fra C:\Program Files\Java\jdk-25\lib\src.zip	github.com/python/cpython/blob/master/Objects/listobject.c	github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_vector.h

Java ArrayList

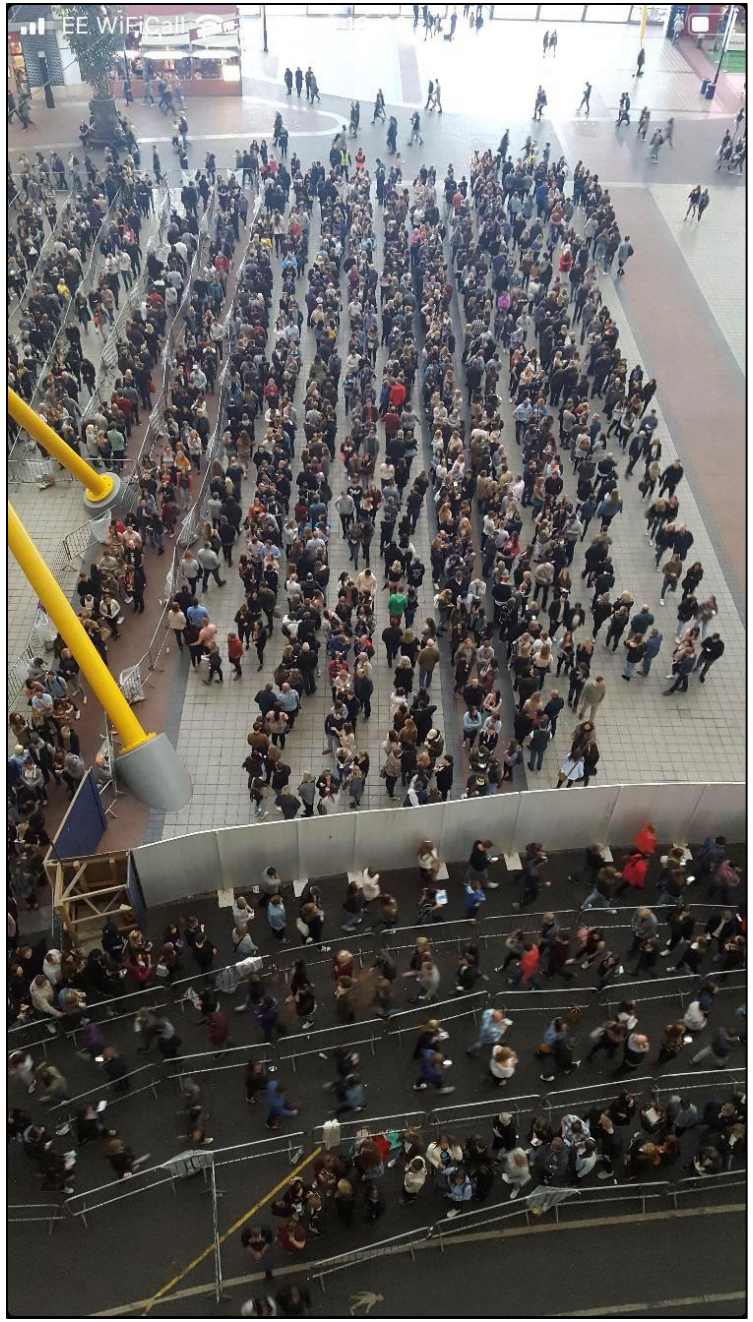
```
private Object[] grow(int minCapacity) {
    int oldCapacity = elementData.length;
    if (oldCapacity > 0 || elementData != DEFAULTCAPACITY_EMPTY_ELEMENTDATA) {
        int newCapacity = ArraysSupport.newLength(oldCapacity,
            minCapacity - oldCapacity, /* minimum growth */
            oldCapacity >> 1           /* preferred growth */);
        return elementData = Arrays.copyOf(elementData, newCapacity);
    } else {
        return elementData = new Object[Math.max(DEFAULT_CAPACITY, minCapacity)];
    }
}
```

Python list

```
static int
list_resize(PyListObject *self, Py_ssize_t newsize)
{
    size_t new_allocated, target_bytes;
    Py_ssize_t allocated = self->allocated;

    if (allocated >= newsize && newsize >= (allocated >> 1)) {
        assert(self->ob_item != NULL || newsize == 0);
        Py_SET_SIZE(self, newsize);
        return 0;
    }
    new_allocated = ((size_t)newsize + (newsize >> 3) + 6) & ~(size_t)3;
    …
}
```

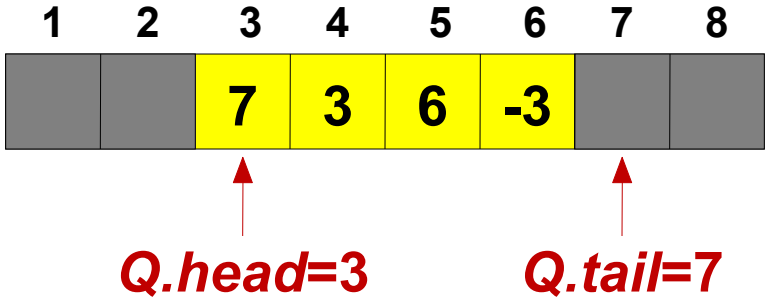
C++ vector

```
size_type
_M_check_len(size_type __n, const char* __s) const
{
    if (max_size() - size() < __n)
        __throw_length_error(__N(__s));

    const size_type __len = size() + (std::max)(size(), __n);
    return (__len < size() || __len > max_size())
        ? max_size()
        : __len;
}
```

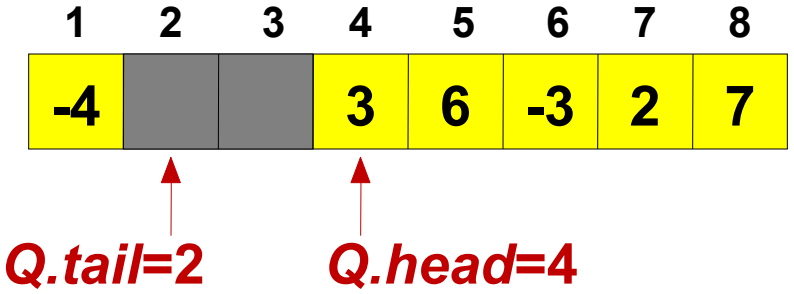


Queues

Queue : array Implementation



Enqueue(2)
Enqueue(7)
Engueue(-4)
Dequeue = 7



ENQUEUE(Q, x)

```
1  Q[Q.tail] = x
2  if Q.tail == Q.size
3        Q.tail = 1
4  else Q.tail = Q.tail + 1
```

DEQUEUE(Q)

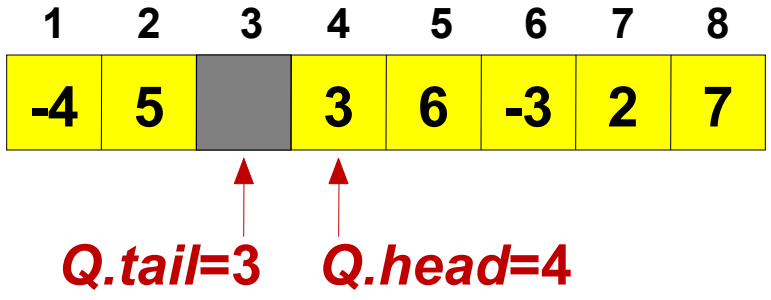
```
1  x = Q[Q.head]
2  if Q.head == Q.size
3        Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

Enqueue, Dequeue : O(1) time

Maximal capacity of a queue implemented in an array of size *n* ?

a) *n*
b) *n* – 1
c) *n* – 2
d) *n* / 2
e) Don't know

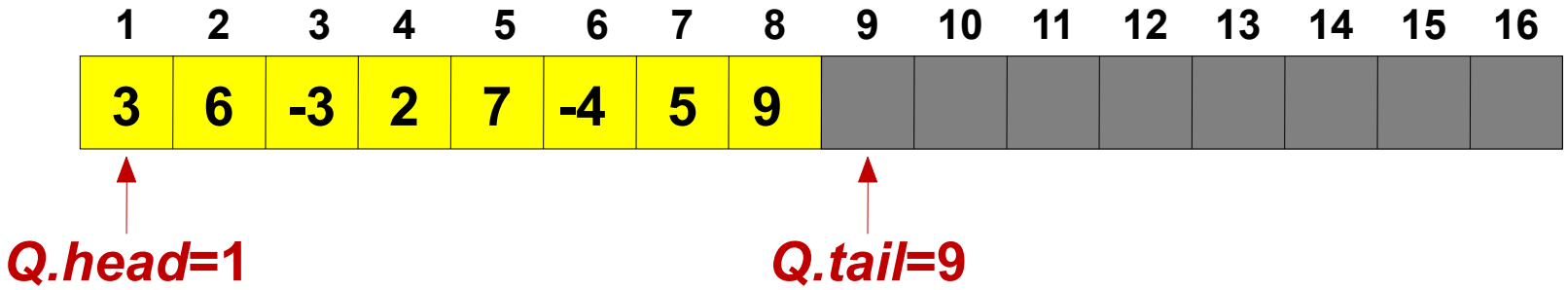
Queue : array implementation



Empty : *Q.tail* = *Q.head* ?

Overflow : array doubling/halving

Enqueue(9)



Array implementation of queue:
n enqueue and dequeue operations take time O(*n*)

Arrays (with doubling/halving)

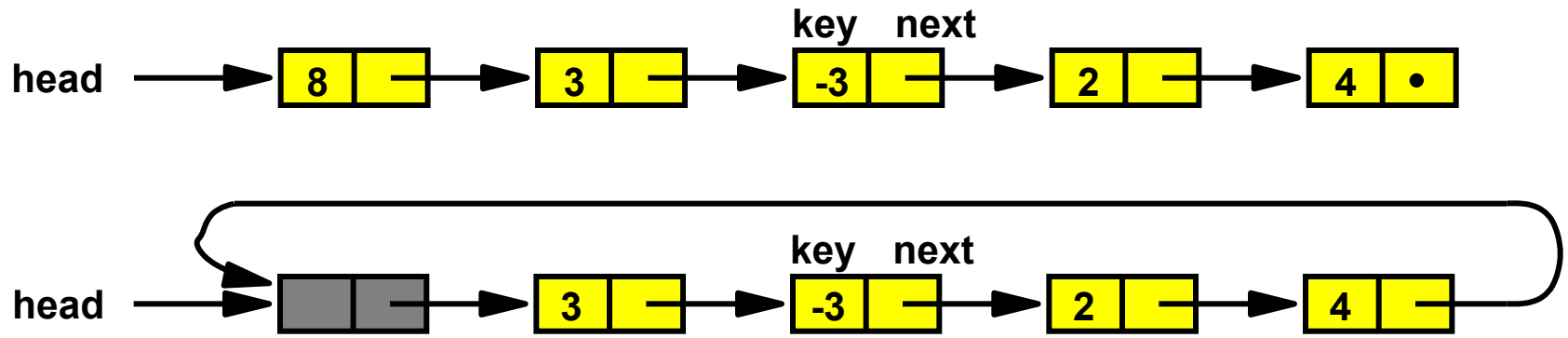
Stack	Push(S, x) Pop(S)	O(1)*
Queue	Enqueue(S, x) Dequeue(S)	O(1)*

* Worst-case without doubling/halving
  Amortized ([CLRS, Chapter 16]) with doubling/halving

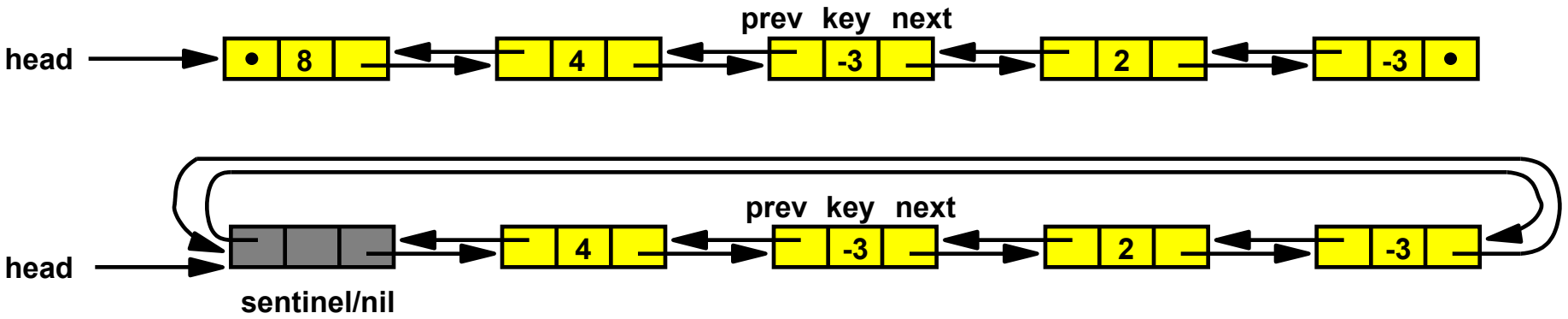
Linked lists

Linked lists

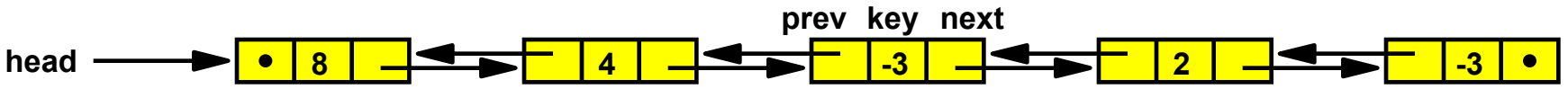
Singly linked (non-circular and circular)



Doubly linked (non-circular and circular)



Doubly linked lists




```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

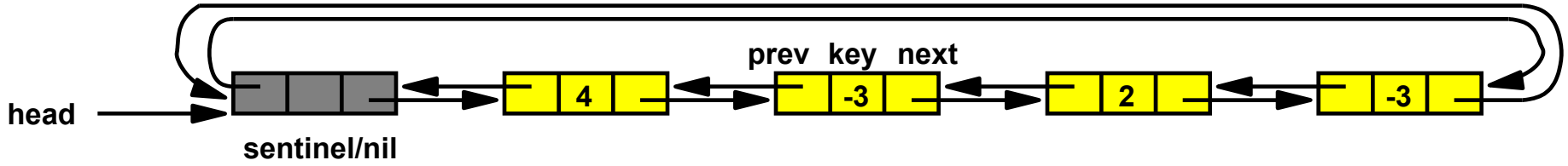
```
LIST-PREPEND(L, x)
1  x.next = L.head
2  x.prev = NIL
3  if L.head ≠ NIL
4      L.head.prev = x
5  L.head = x
```

```
LIST-INSERT(x, y)
1  x.next = y.next
2  x.prev = y
3  if y.next ≠ NIL
4      y.next.prev = x
5  y.next = x
```

```
LIST-DELETE(L, x)
1  if x.prev ≠ NIL
2      x.prev.next = x.next
3  else L.head = x.next
4  if x.next ≠ NIL
5      x.next.prev = x.prev
```

List-Search	O(n)
List-Prepend List-Insert List-Delete	O(1)

Doubly linked circular lists



LIST-INSERT′(x, y)

```
1  x.next = y.next
2  x.prev = y
3  y.next.prev = x
4  y.next = x
```

LIST-DELETE′(x)

```
1  x.prev.next = x.next
2  x.next.prev = x.prev
```

[CLRS4]

LIST-SEARCH′(L, k)

```
1  L.nil.key = k
2  x = L.nil.next
3  while x.key ≠ k
4        x = x.next
5  if x == L.nil
6        return NIL
7  else return x
```

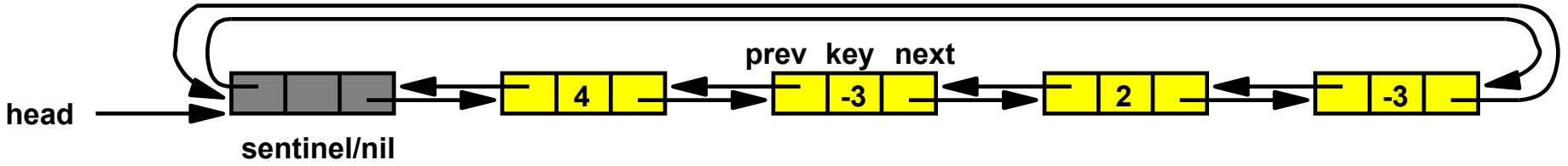
[CLRS3]

LIST-SEARCH′(L, k)

```
1  x = L.nil.next
2  while x ≠ L.nil and x.key ≠ k
3        x = x.next
4  return x
```

List-Search’	O(n)
List-Insert’ List-Delete’	O(1)

Doubly linked circular lists



Stack	Push(S, x) Pop(S)	O(1)
Queue	Enqueue(S, x) Dequeue(S)	O(1)

Dancing Links

Donald E. Knuth, Stanford University

My purpose is to discuss an extremely simple technique that deserves to be better known. Suppose x points to an element of a doubly linked list; let $L[x]$ and $R[x]$ point to the predecessor and successor of that element. Then the operations

$$L\big[R[x]\big] \leftarrow L[x], \qquad R\big[L[x]\big] \leftarrow R[x] \tag{1}$$

remove x from the list; every programmer knows this. But comparatively few programmers have realized that the subsequent operations

$$L\big[R[x]\big] \leftarrow x, \qquad R\big[L[x]\big] \leftarrow x \tag{2}$$

will put x back into the list again.

Donald E. Knuth (1938-)

Donald E. Knuth, *Dancing links,* Millennial Perspectives in Computer Science, 187–214, 2000, Palgrave. Also arXiv, doi: 10.48550/arXiv.cs/0011047

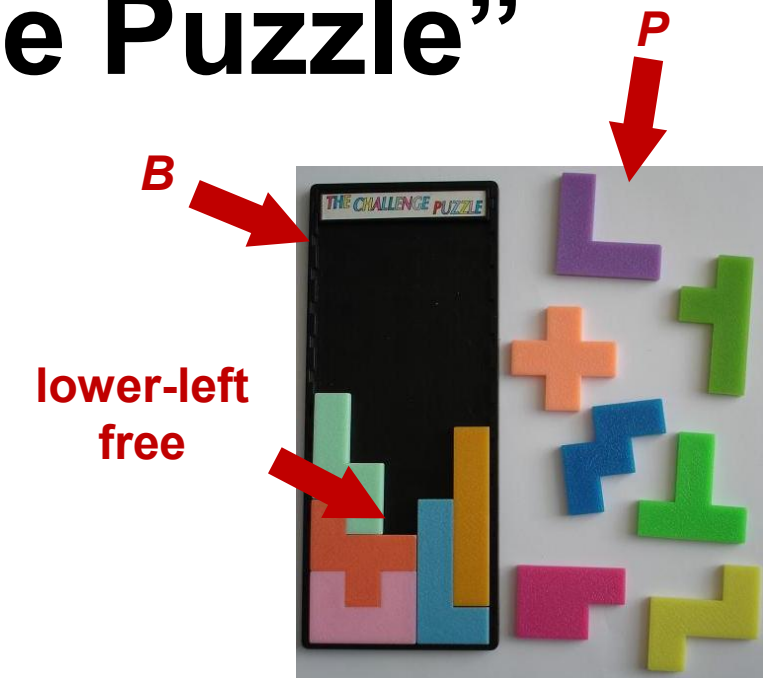
”The Challenge Puzzle”

?

”The Challenge Puzzle”

B := Empty board
P := All pieces
Solve(*B*, *P*)

```
procedure Solve(Partial solution B, Pieces P)
  if |P| = 0 then
    report solution B
  else
    for all p in P
      remove p from P
      for all orientations of p  (* max 8 distinct *)
        if p can be placed in lower-left free cell then
          insert p in B
          Solve(B, P)
          remove p from B
      reinsert p in P
```



Before

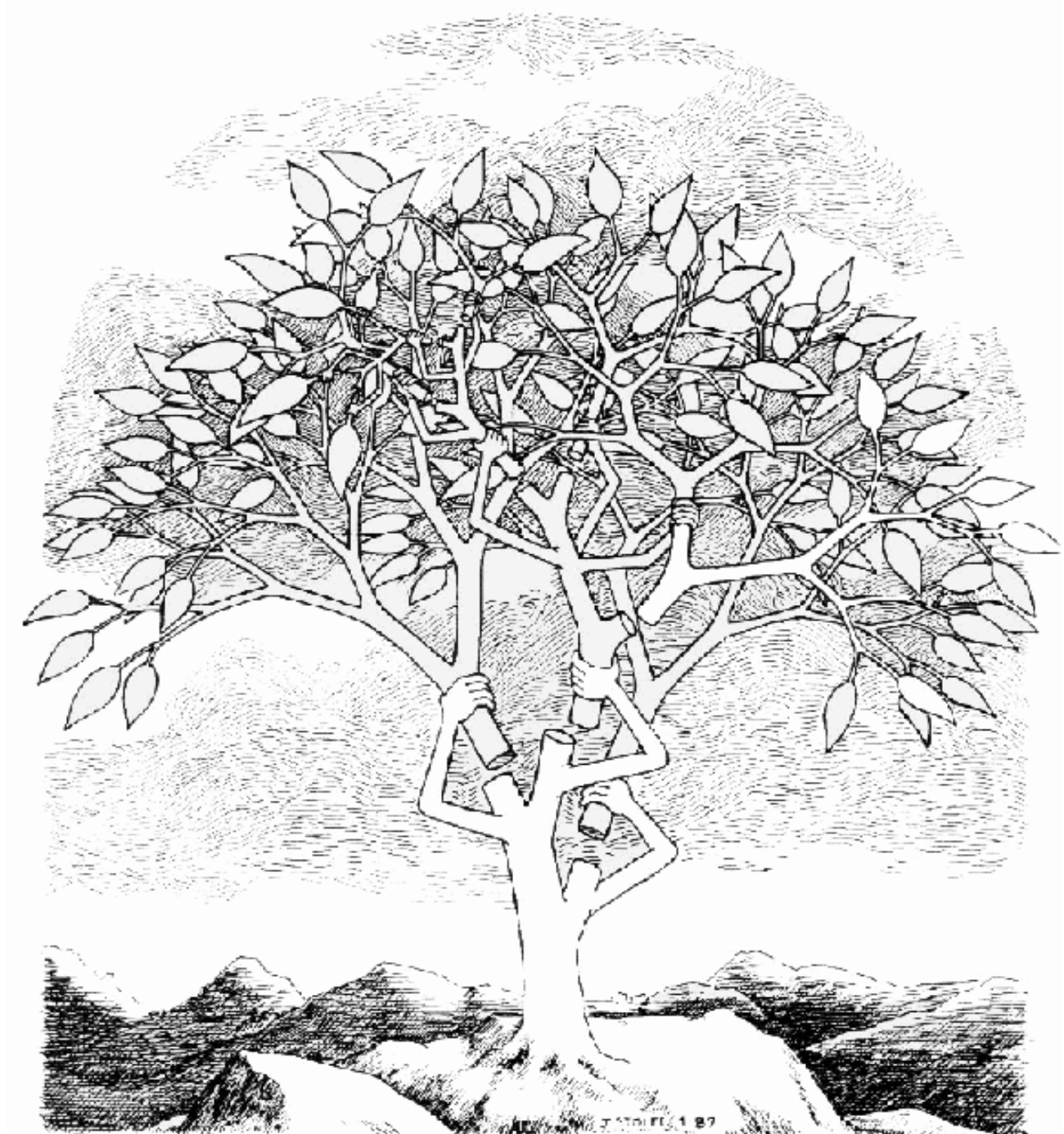
After

”The Challenge Puzzle”

4.040 solutions

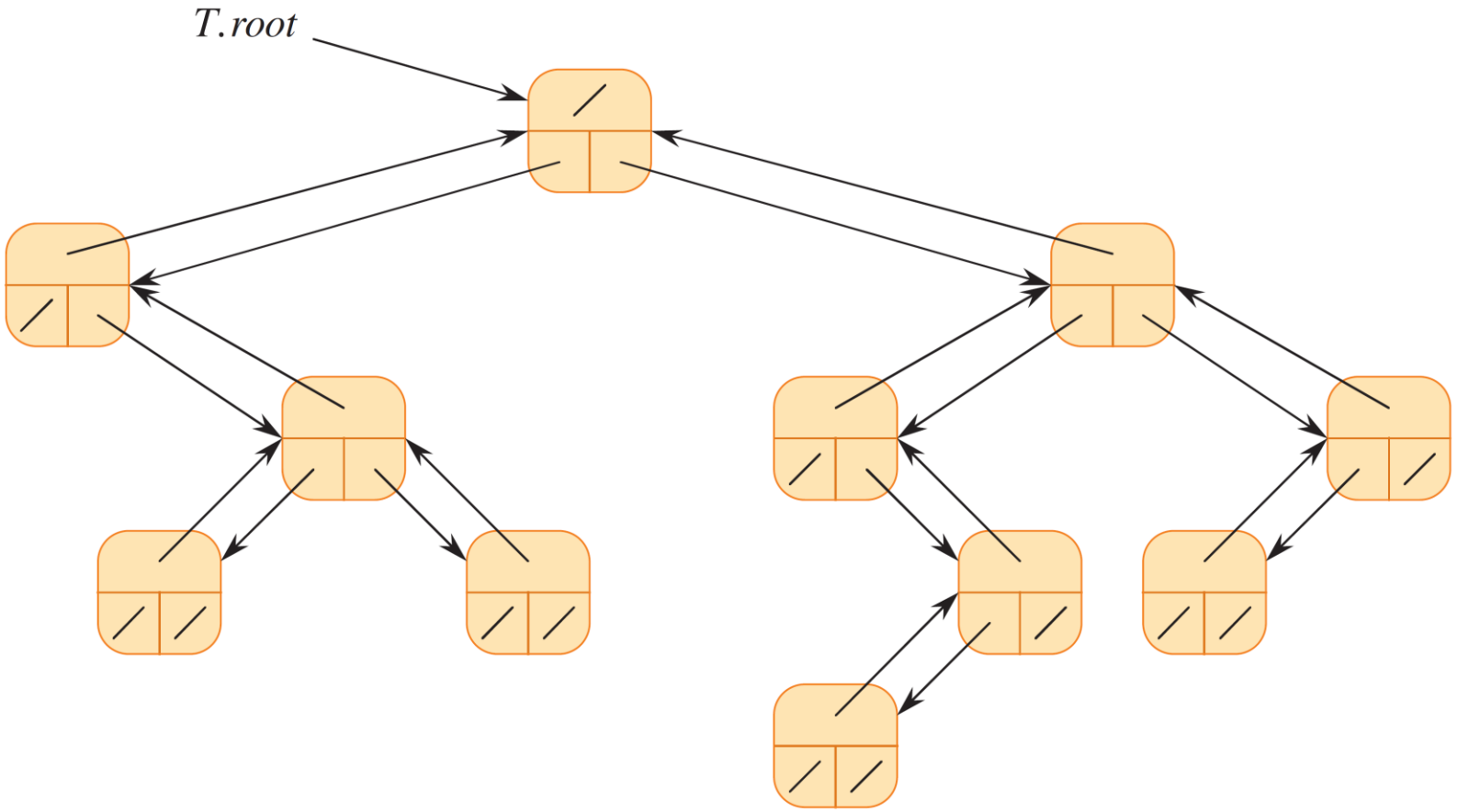
Solve function places

8.387.259 pieces



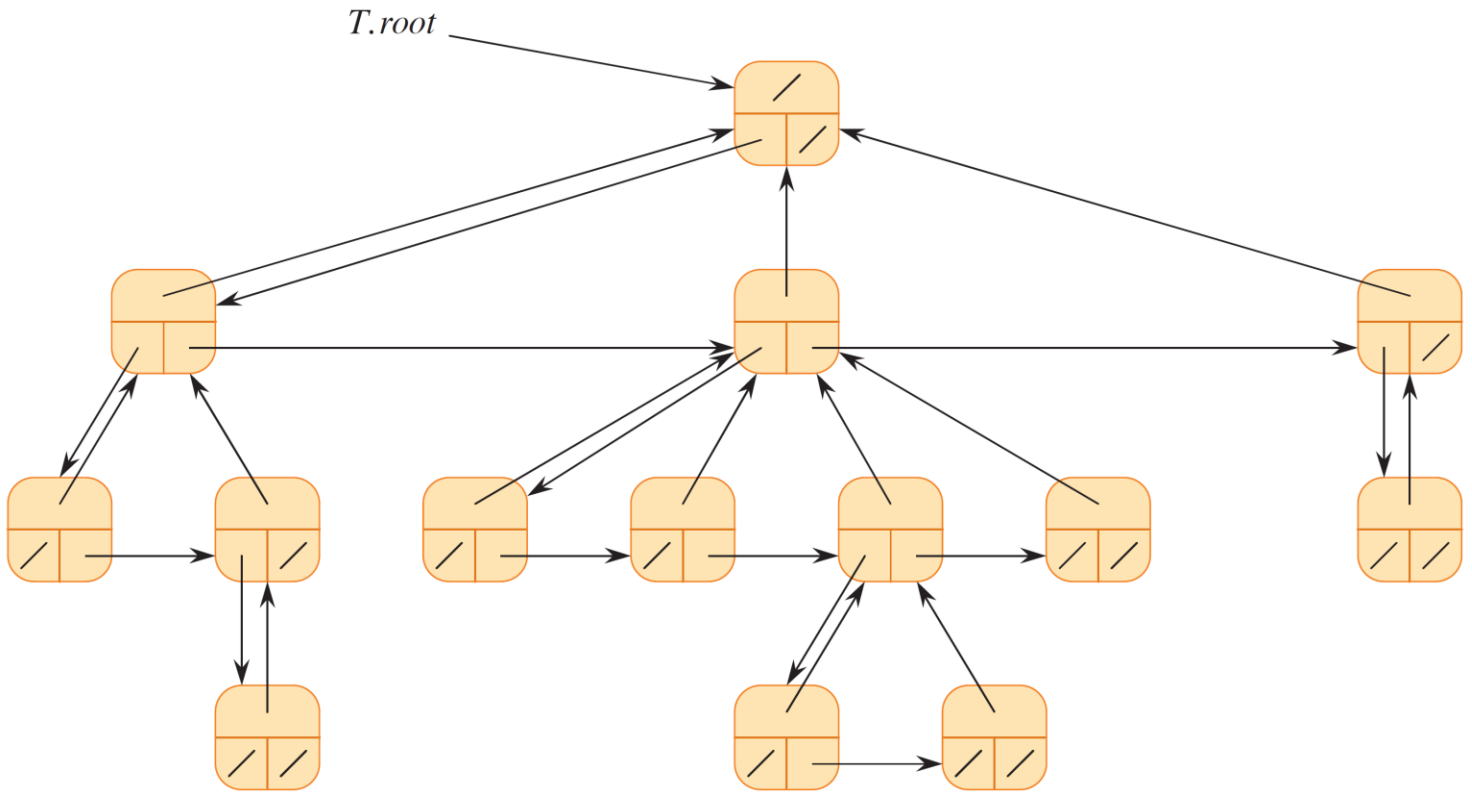
(Jorge Stolfi)

Binary tree representation



Data fields: **left child**, **right child**, **parent**

Tree representation



Data fields: **left child**, **right sibling**, **parent**

Time to access the *i*-th child of a node ?

a) O(1)
b) O(*i*)
c) O(log *i*)
d) Don't know

Algorithms and Data Structures

Hashing
[CLRS, Chapter 11.1-11.4]

Merriam-Webster

Dictionary Thesaurus

hash

hash 1 of 3 verb

hash

hashed; hashing; hashes

Synonyms of *hash* >

transitive verb

1 **a** : to chop (food, such as meat and potatoes) into small pieces

b : **CONFUSE**, **MUDDLE**

2 : to talk about : **REVIEW** → often used with *over* or *out*

hash over a problem

hashing out their differences

hash 2 of 3 noun (1)

1 : chopped food

specifically : chopped meat mixed with potatoes and browned

2 : a restatement of something that is already known

the same old *hash*

3 **a** : **HODGEPODGE**, **JUMBLE**

b : a confused muddle

made a *hash* of the whole project

4 : **POUND SIGN sense 2**

hash 3 of 3 noun (2)

: **HASHISH**

www.merriam-webster.com/dictionary/hash

Abstract data structure: dictionary

Search(S, k)
Insert(S, x)
Delete(S, x)

Can we exploit that x for all practical purposes is a **sequence of bits? Yes !**

All keys are numbers...

”Cat” = $01000011.01100001.01110100_2 = 4415860_{10}$

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

What number does ”AU” correspond to ?

a) 4155_{10}
b) 6175_{10}

c) 16725_{10}
d) 24949_{10}
e) Don't know

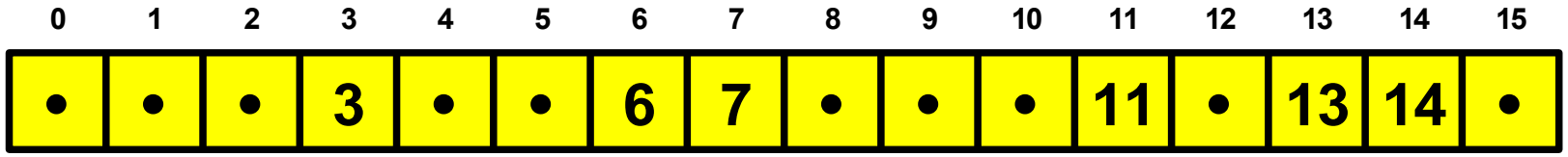
	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

$4 \cdot 16^3 + 1 \cdot 16^2 + 5 \cdot 16^1 + 5 \cdot 16^0 = 4 \cdot 4096 + 1 \cdot 256 + 5 \cdot 16 + 5$

↓

”AU” = 4155_{16} = 0100 0001 0101 0101_2 = 16725_{10}

Direct adressing: keys {0,1, 2, ..., *m*-1}




```
DIRECT-ADDRESS-SEARCH(T, k)
1  return T[k]

DIRECT-ADDRESS-INSERT(T, x)
1  T[x.key] = x

DIRECT-ADDRESS-DELETE(T, x)
1  T[x.key] = NIL
```

+ Good for small key universes
+ Can manually define keys to be 0, 1, 2, 3, ...
– Huge space overhead if only few keys used

Hash function

- (Huge) key universe ***U***
- Hash function

$$h : U \rightarrow \{0, 1, ..., m\text{-}1\}$$

$m << |U|$

x	$h(x)$
7	0
257	4
519	5
746	6
1231	2
3409	0
12001	1
12002	6
24123	5
25964	5

$h(k) = (5 \cdot k) \bmod 7$

\+ Easy to distribute keys evenly
– Multiple keys can hash to the same value
– Nearly identical keys can be arbitrarily distributed

possible (hash) functions

$h : U \rightarrow \{0, 1, \ldots, m\text{-}1\}$?

a) $|U| \cdot m$

b) $|U|$!

c) $2^{|U|}$

d) $m^{|U|}$

e) $|U|^m$

f) Don't know

possible (hash) functions describable by 6 characters ?

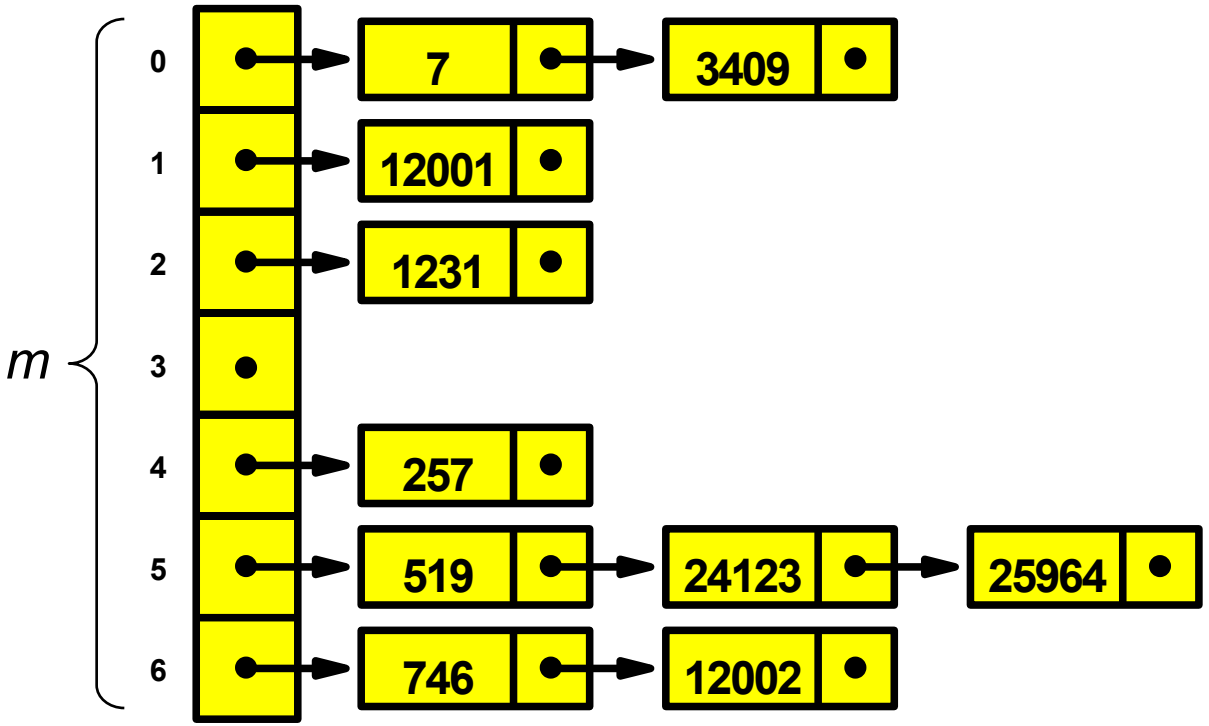
	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

a) $|U|$
b) 128^6
c) $6^{|U|}$
d) $6 \cdot m$
e) Don't know

x	$h(x)$
7	0
257	4
519	5
746	6
1231	2
3409	0
12001	1
12002	6
24123	5
25964	5

$h(k) = 5k \bmod 7$

Hash table with chaining

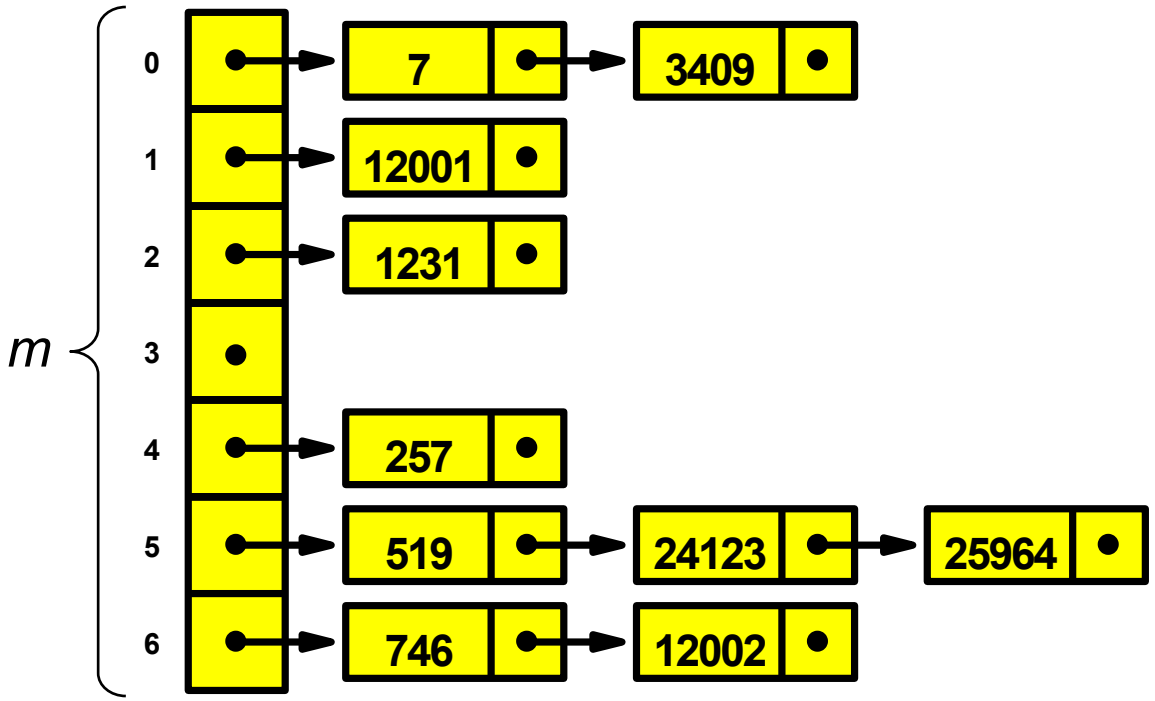


x	$h(x)$
7	0
257	4
519	5
746	6
1231	2
3409	0
12001	1
12002	6
24123	5
25964	5

$h(k) = (5 \cdot k) \bmod 7$

- Store set of keys K
- Choose **random** hash funktion $h : U \rightarrow \{0, 1, ..., m\text{-}1\}$
- Store keys in table by **hash value**
- **Collision lists** for keys with same hash value

Hash table with chaining



CHAINED-HASH-INSERT(T, x)

1 LIST-PREPEND$(T[h(x.key)], x)$

CHAINED-HASH-SEARCH(T, k)

1 **return** LIST-SEARCH$(T[h(k)], k)$

CHAINED-HASH-DELETE(T, x)

1 LIST-DELETE$(T[h(x.key)], x)$

- Choose **uniform random** hash function
- Expected # keys in one table entry chain $|K| / m$
- Insert, Delete, Search **expected time** $O(|K| / m)$, i.e., $O(1)$ for $m = \Omega(|K|)$

What is a good hash function ?

– For all hash functions there exist a bad set of keys, where all keys hash to the same value

\+ For each set there exists a good hash function distributing evenly the keys (if the hash function can be described succinctly is a different question)

Goal Find a small set of hash functions, where a random hash function works sufficiently well for a given set

Hash functions – examples

$h(k) = k \bmod m$ (typically m is a prime)

$m = 2^8$ ignores all except the last 8 bits:

$h(...x_3x_2x_1 10101111) = h(...y_3y_2y_1 10101111)$

$m = 2^8-1$ ignores all swaps of characters

$h(\text{“}c_3c_2c_1\text{”}) = h(\text{“}c_1c_3c_2\text{”})$

$h(k) = \lfloor k \cdot s/2^{w-t} \rfloor \bmod 2^t$ (k = w-bit, $h(k)$= t-bit)

$h(0101000010101010) = 01000$

$0101000010101010 \cdot 1001111000110111$

$= 0011000111011010\underbrace{01000}_{t}00010000110$

w

Universal hash functions

Find **prime** $p \geq |U|$.

Define $p \cdot (p-1)$ hash functions $h_{a,b}$, where $1 \leq a < p$ and $0 \leq b < p$

$$h_{a,b}(k) = ((a \cdot k + b) \bmod p) \bmod m$$

Theorem

For two keys $x \neq y$ and a random hash function $h_{a,b}$ we have

$$\Pr[h_{a,b}(x) = h_{a,b}(y)] \leq 1/m$$ **(Universal)**

Corollary

For a hash table using chaining and a **random hash function** $h_{a,b}$
Insert, Delete, and Search take **expected time** $O(|K| / m)$

Hash table with chaining

Search(S, k) **Insert**(S, x) **Delete**(S, x)	O(1) expected

Hashing of numbers with many bits...

(s bits) $x_{s/w-1}$ (w bits) x_1 (w bits) x_0 (w bits)

$$x = (\boxed{b_{s-1}b_{s-2}\ldots b_{s-w}}\ \boxed{b_{s-w-1}\ \ldots\ b_{2w}}\ \boxed{b_{2w-1}\ldots b_{w+1}b_w}\ \boxed{b_{w-1}\ldots b_2b_1b_0})_2$$

$$= x_{s/w-1}\cdot 2^{w(s/w-1)} + x_{s/w-2}\cdot 2^{w(s/w-2)} + \cdots + x_1\cdot 2^w + x_0$$

$$h_a(x) = (x_{s/w-1}\cdot a^{s/w-1} + x_{s/w-2}\cdot a^{s/w-2} + \cdots + x_1\cdot a^1 + x_0) \bmod p$$

$p > 2^w$ prime

$0 < a < p$ random

$$= ((\cdots((x_{s/w-1}\cdot a + x_{s/w-2})\cdot a + x_{s/w-3})\cdot a + \cdots + x_1)\cdot a + x_0) \bmod p$$

Polynomial evaluation $h_a(x)$

$y = 0$

for $i = s/w - 1$ downto 0

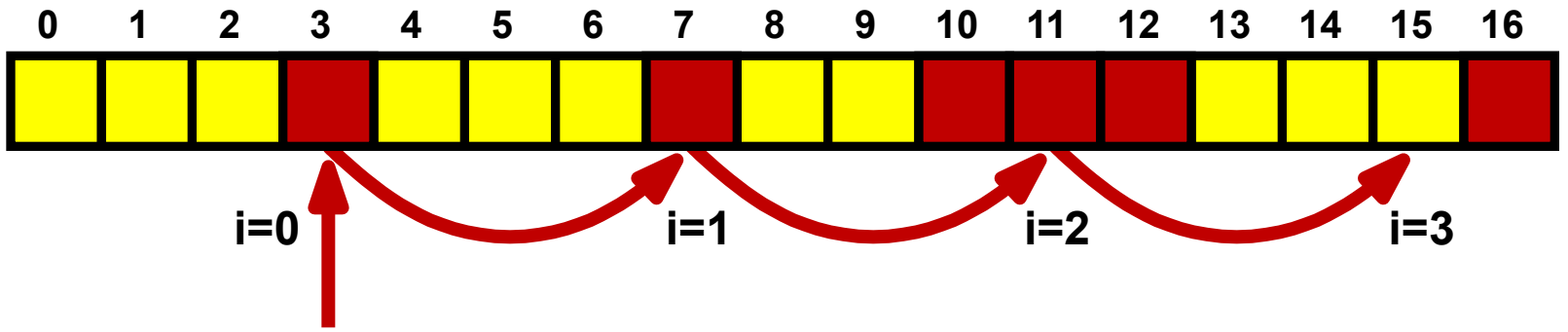
$y = (y \cdot a + x_i) \bmod p$

$h_a(x) = y$

$(a\cdot b) \bmod p = ((a \bmod p)\cdot b) \bmod p$

$(a+b) \bmod p = ((a \bmod p)+b) \bmod p$

Open addressing



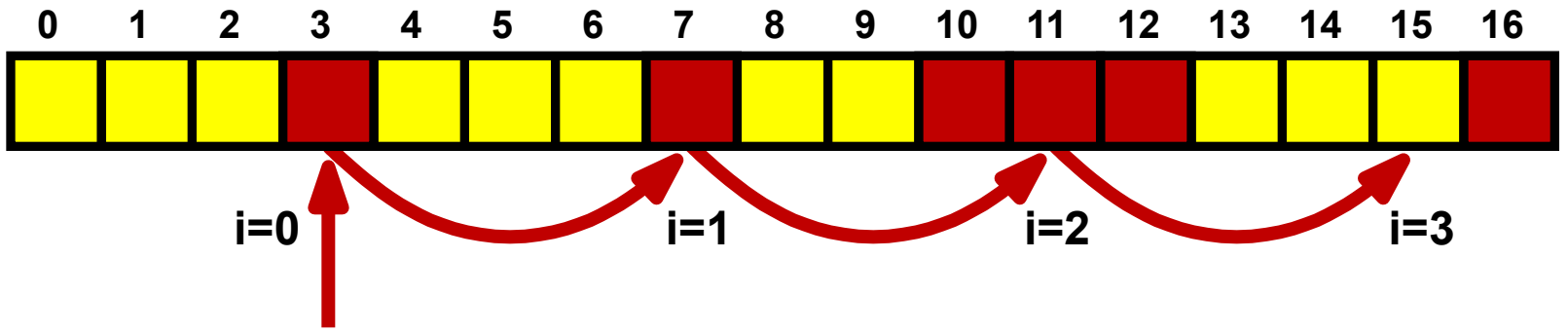
HASH-INSERT(T, k)

```
1  i = 0
2  repeat
3       q = h(k, i)
4       if T[q] == NIL
5            T[q] = k
6            return q
7       else i = i + 1
8  until i == m
9  error "hash table overflow"
```

HASH-SEARCH(T, k)

```
1  i = 0
2  repeat
3       q = h(k, i)
4       if T[q] == k
5            return q
6       i = i + 1
7  until T[q] == NIL or i == m
8  return NIL
```

Open addressing – analysis



Uniform hashing

$h(k, 0), h(k, 1), h(k, 2), \ldots$ is a uniform random sequence **(unrealistic)**

Theorem [CLRS Chapter 11.4, Corollary 11.7]

Uniform hashing requires expected $1/(1-\alpha)$ lookups for insertions, where $\alpha = |K|/m$ is the load factor

Linear probing

Insert k in first available slot

$$h(k, i) = (h'(k) + i) \mod m$$

for i = 0, 1, 2...

Example

Insert 9, 3, 20, 6, 12, 2, 19, 11, 5

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
	9			2	19	3	20	12	11	5		6				

x	$h'(x)$
2	4
3	6
5	10
6	12
9	1
11	5
12	7
19	4
20	6

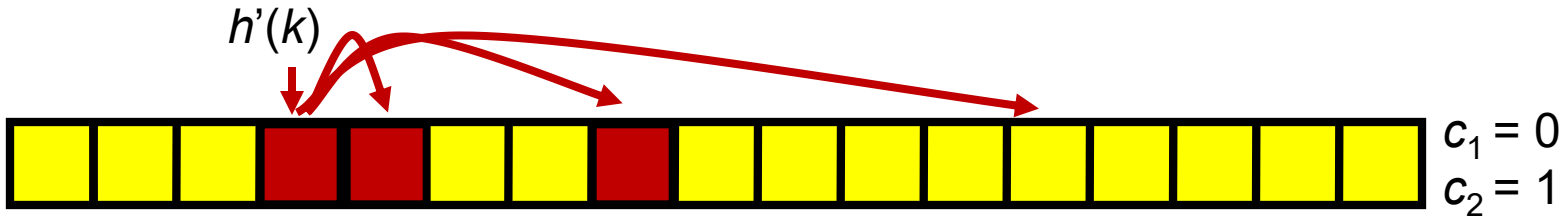
$h'(x) = (2 \cdot x) \mod 17$

Quadratic probing

Insert k in first available slot

$$h(k, i) = (h'(k) + c_1 \cdot i + c_2 \cdot i^2) \bmod m$$

for $i = 0, 1, 2, \ldots$ where c_1 and $c_2 \neq 0$ are constants

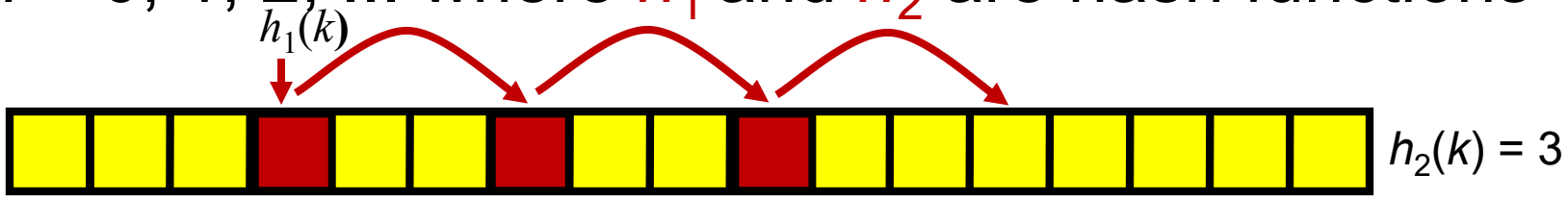


Double hashing

Insert k in first available slot

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \bmod m$$

for $i = 0, 1, 2, \ldots$ where h_1 and h_2 are hash functions



Double hashing – where is 1 inserted ?

a) 1
b) 5
c) 7
d) 9
e) Don't know

$h_1(k) = (2k) \bmod 10$

$h_2(k) = 1+((2k) \bmod 9)$

0	1	2	3	4	5	6	7	8	9
5		6	3	2	1	8		7	

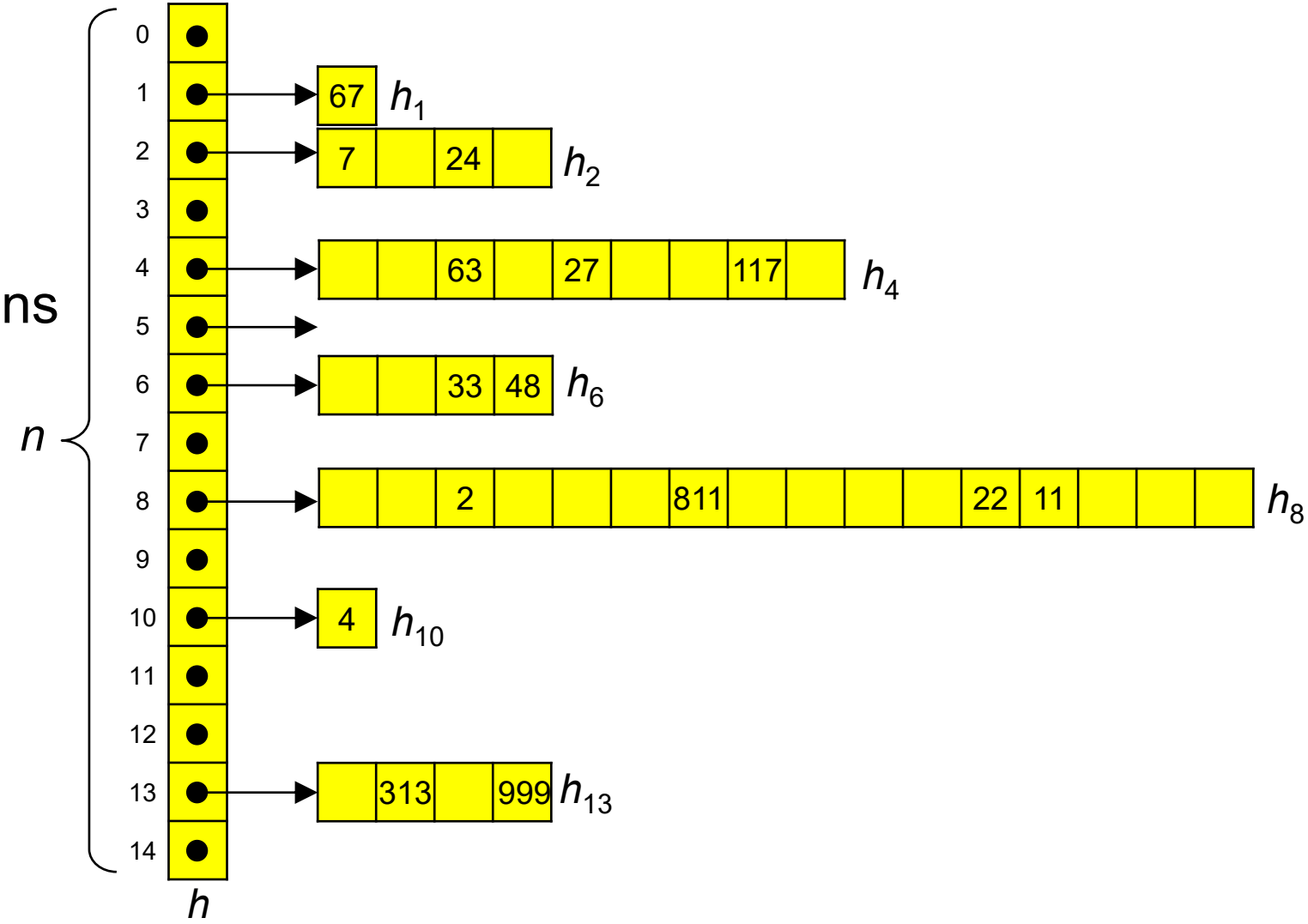
exam question August 2010

CLRS 3rd edition, Chapter 11.5
– not curriculum

Perfect hashing (static)

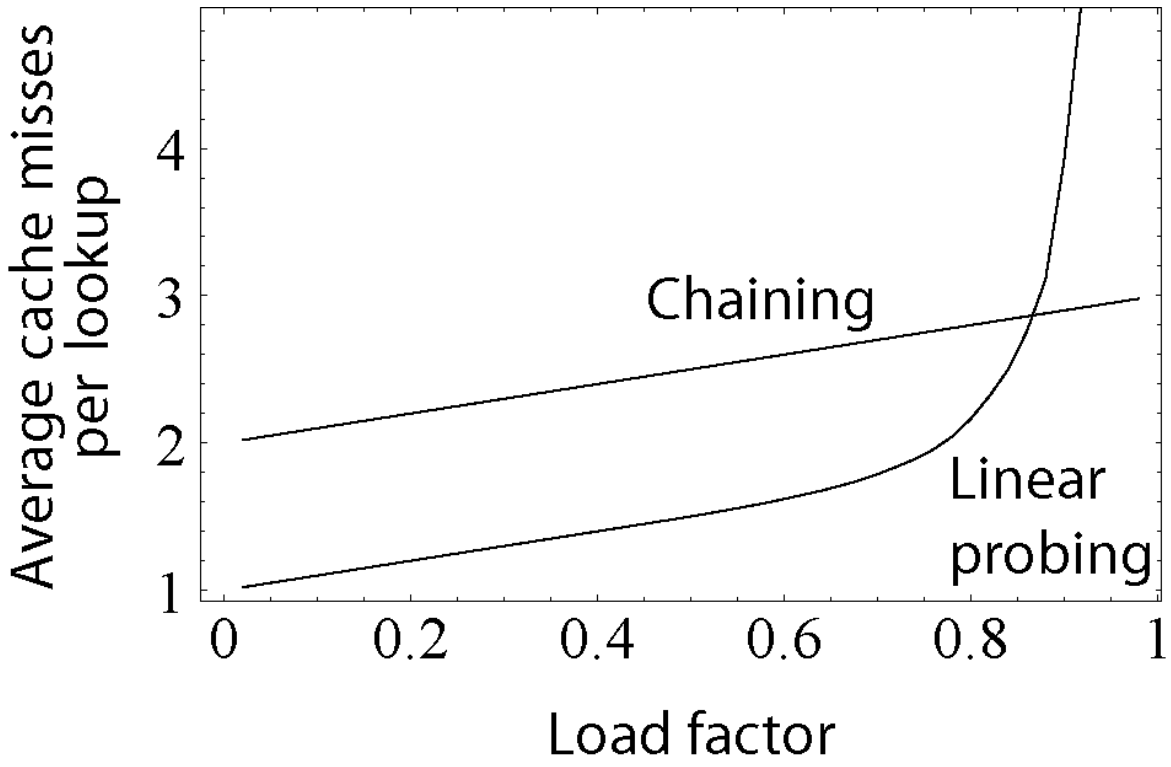
- n keys
- 2 levels of hash functions
- Buckets separate hash functions
- h hash table with n slots
- Bucket with n_i keys, n_i^2 slots
- Universal hash functions

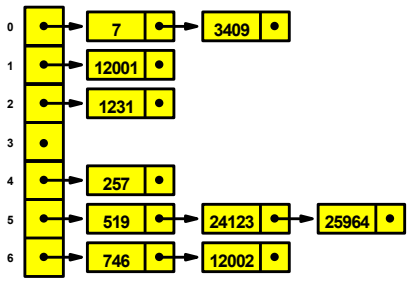
E[collisions] = $\binom{n}{2}$ / # slots

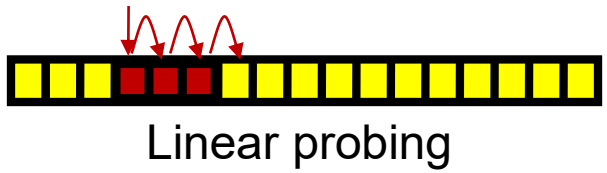


Fredman, Komlós, Szemerédi, *Storing a Sparse Table with 0(1) Worst Case Access Time*. Journal of the ACM, 1984, DOI: 10.1145/828.1884

Experimental comparison







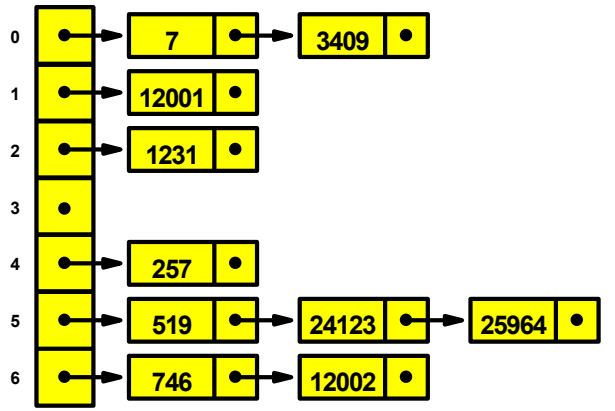
wikipedia.org

Hashing

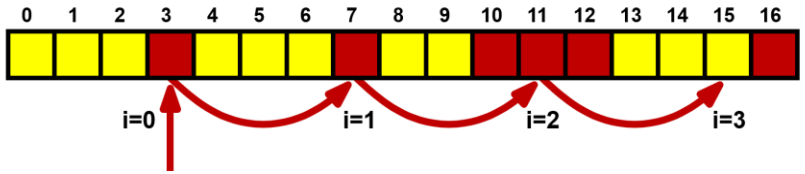
- **Choice of hash function**
  - Try different options...
  - Universal hash functions

$$h_{a,b}(k) = ((a \cdot k + b) \bmod p) \bmod m$$

- **Hash tables**
  - Collision lists (chaining)



- **Open addressing**
  - Linear probing
  - Quadratic probing
  - Double hashing



Algorithms and Data Structures

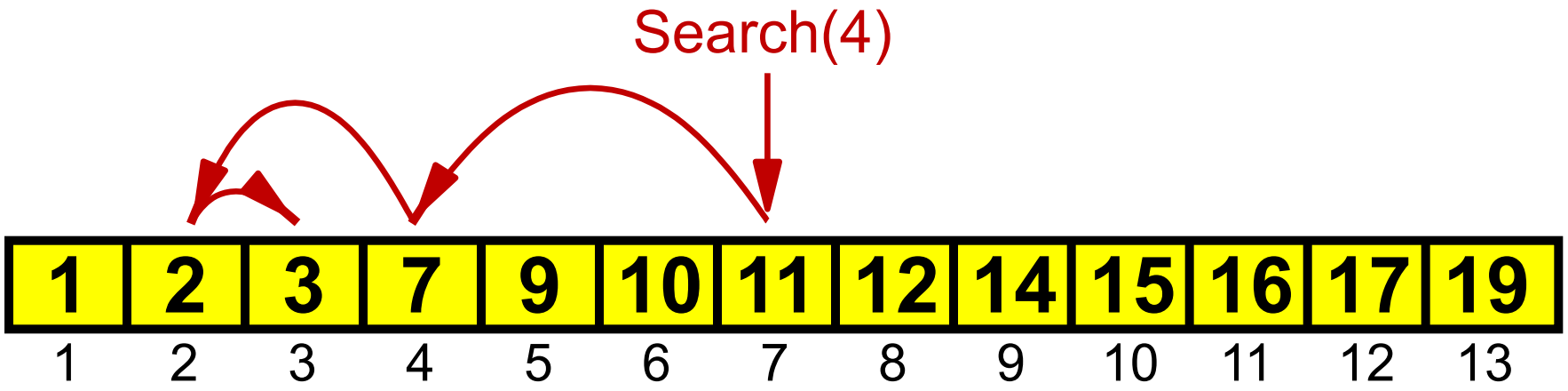
Binary search trees
[CLRS, Chapter 12]

Abstract data structure: dictionary

Search(S, k)
Insert(S, x)
Delete(S, x)

Elements can only be compared using ”≤”

(Static) dictionary: sorted array

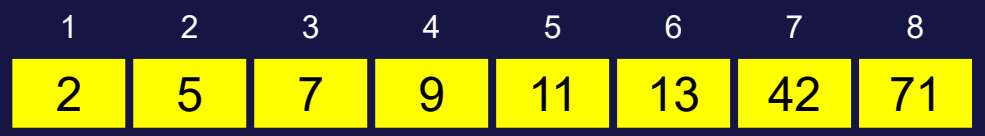


Search(S, x**)**	O(log n)
Insert(S, x**)** **Delete(**S, x**)**	O(n)

Worst-case search time

How many comparisons do a search after an element require in a sorted array with 8 elements, when elements can only be compared ?

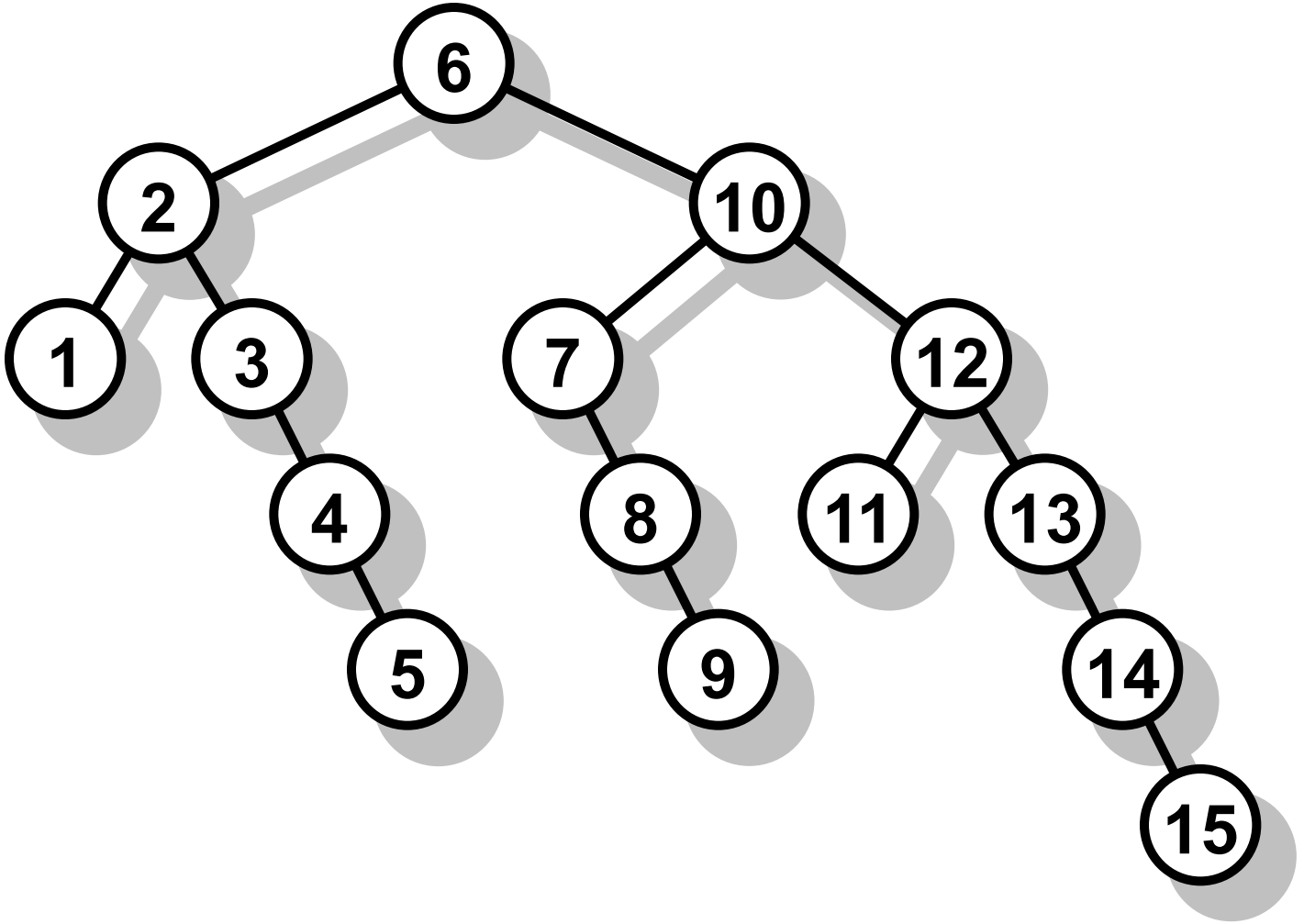
a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) Don't know



Theorem

Searching among ≥ 2^i elements requires at least $i + 1$ comparisons

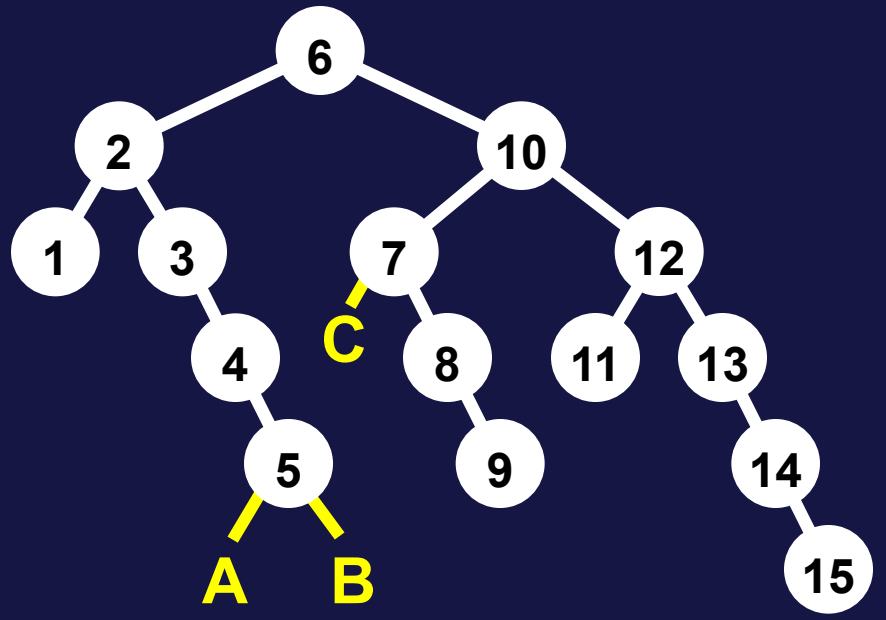
Binary search tree



Invariant Each node stores an element, and elements in the left (right) subtree are smaller (larger) than or equal to the element in a node

Where can 5.5 be inserted ?

a) A
b) B
c) C
d) A or B
e) B or C
f) Don't know

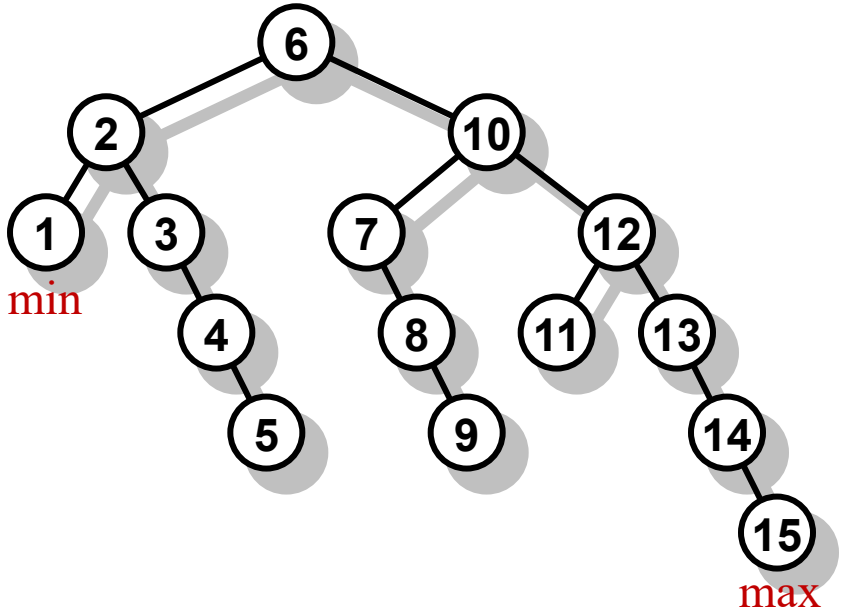


Searches

```
TREE-SEARCH(x, k)
1  if x == NIL or k == x.key
2       return x
3  if k < x.key
4       return TREE-SEARCH(x.left, k)
5  else return TREE-SEARCH(x.right, k)
```

```
ITERATIVE-TREE-SEARCH(x, k)
1  while x ≠ NIL and k ≠ x.key
2       if k < x.key
3            x = x.left
4       else x = x.right
5  return x
```

```
TREE-SUCCESSOR(x)
1  if x.right ≠ NIL
2       return TREE-MINIMUM(x.right)
3  else
4       y = x.p
5       while y ≠ NIL and x == y.right
6            x = y
7            y = y.p
8       return y
```




```
INORDER-TREE-WALK(x)
1  if x ≠ NIL
2       INORDER-TREE-WALK(x.left)
3       print x.key
4       INORDER-TREE-WALK(x.right)
```

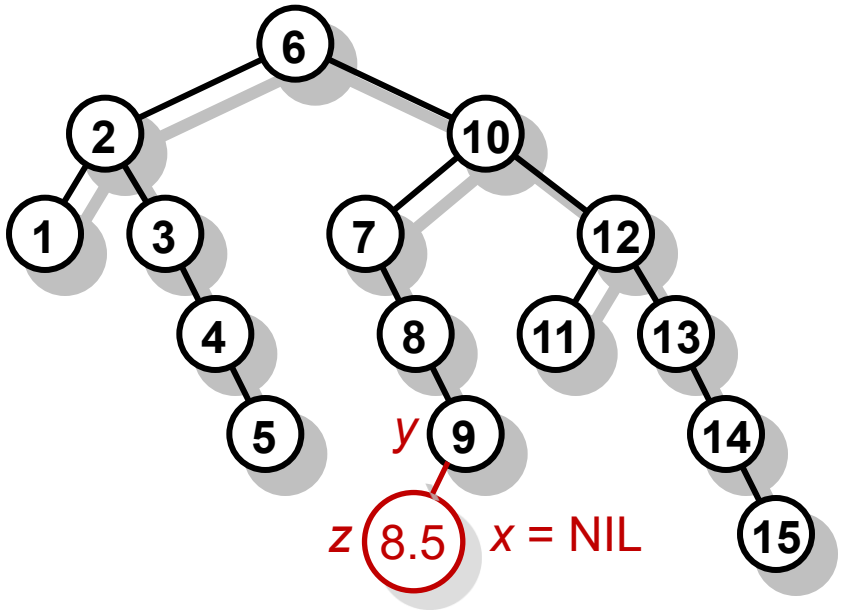
```
TREE-MINIMUM(x)
1  while x.left ≠ NIL
2       x = x.left
3  return x
```

```
TREE-MAXIMUM(x)
1  while x.right ≠ NIL
2       x = x.right
3  return x
```

Insertions

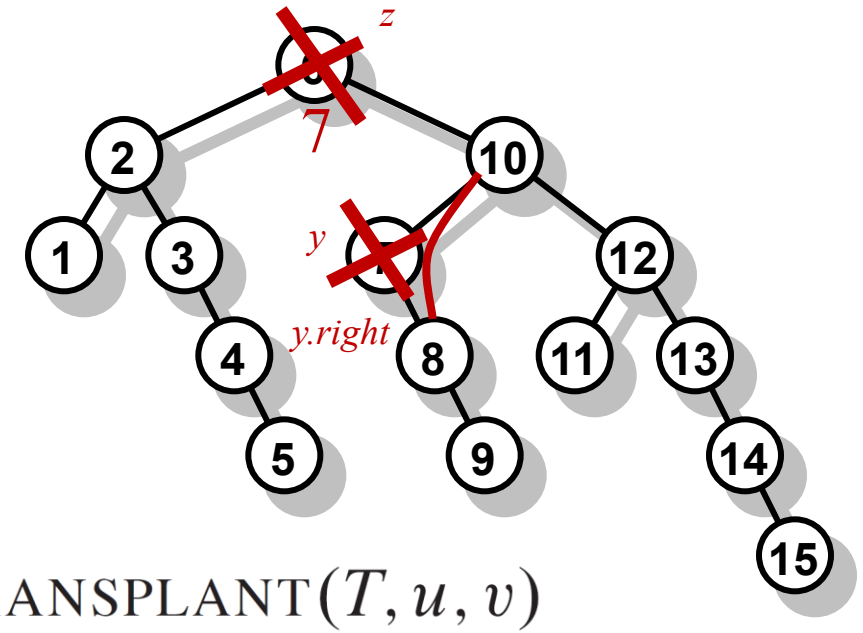
```
TREE-INSERT(T, z)
 1   x = T.root
 2   y = NIL
 3   while x ≠ NIL
 4         y = x
 5         if z.key < x.key
 6               x = x.left
 7         else x = x.right
 8   z.p = y
 9   if y == NIL
10         T.root = z
11   elseif z.key < y.key
12         y.left = z
13   else y.right = z
```

Iterative search



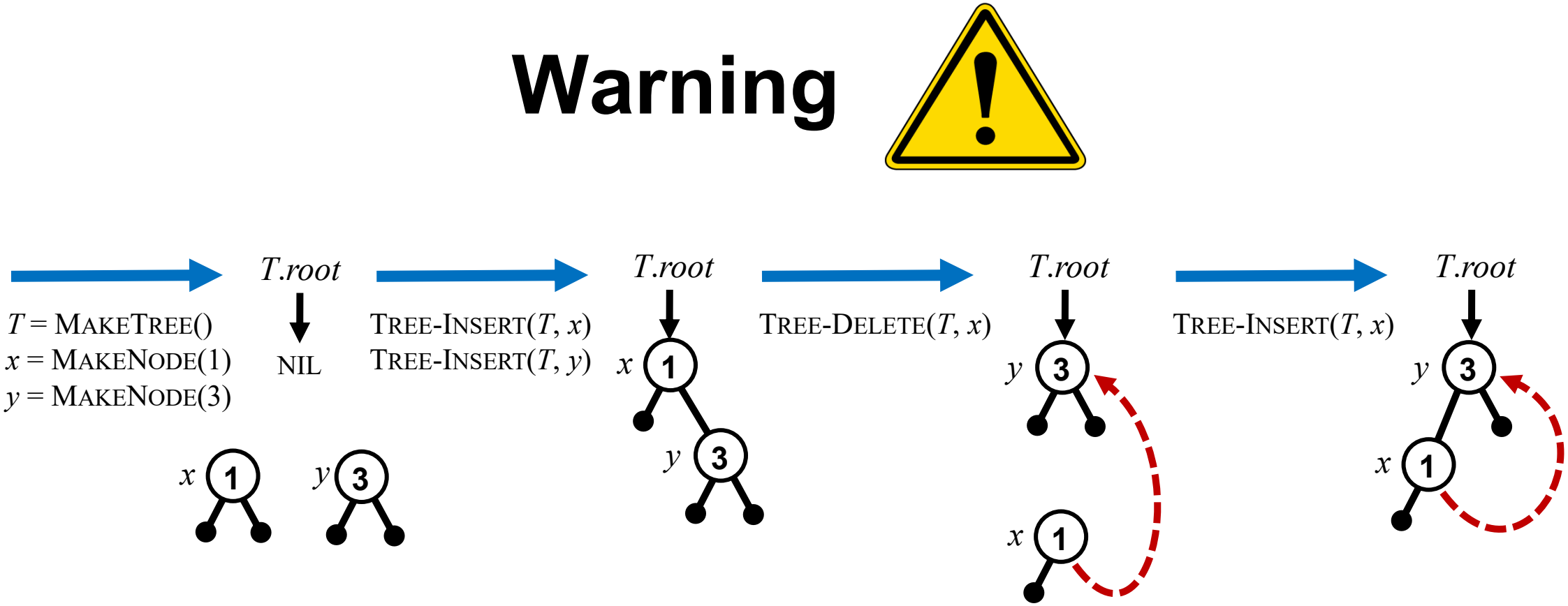
Deletions

```
TREE-DELETE(T, z)
 1  if z.left == NIL
 2       TRANSPLANT(T, z, z.right)
 3  elseif z.right == NIL
 4       TRANSPLANT(T, z, z.left)
 5  else y = TREE-MINIMUM(z.right)
 6       if y ≠ z.right
 7            TRANSPLANT(T, y, y.right)
 8            y.right = z.right
 9            y.right.p = y
10       TRANSPLANT(T, z, y)
11       y.left = z.left
12       y.left.p = y
```




```
TRANSPLANT(T, u, v)
1  if u.p == NIL
2       T.root = v
3  elseif u == u.p.left
4       u.p.left = v
5  else u.p.right = v
6  if v ≠ NIL
7       v.p = u.p
```

Warning



- CLRS pseudocode does not reset the deleted nodes pointers, and assumes the inserted node's pointers are NIL
- TREE-SEARCH(*T.root*, 2) infinite loop ☹

Binary search trees

Inorder-Tree-Walk	O(n)
Tree-Search **Iterative-Tree-Search** **Tree-Minimum** **Tree-Maximum** **Tree-Insert** **Tree-Delete**	O(h)

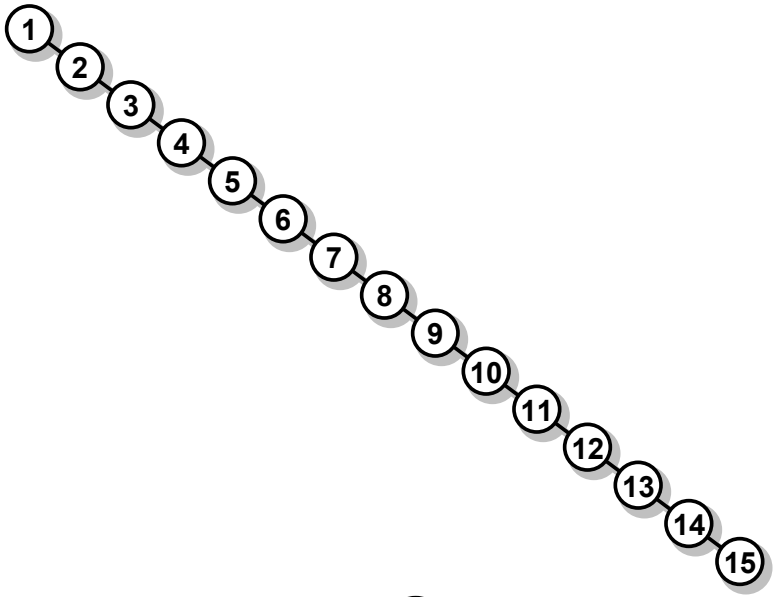
h = height, n = # elements

Height
of a binary search tree?

Largest and smallest heights

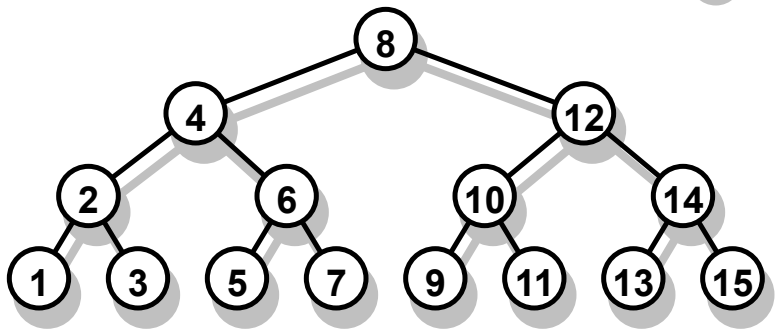
Insert in increasing order

– Height n



Perfect balanced
(smallest possible height)

– Height $1+\lfloor \log_2 n \rfloor$



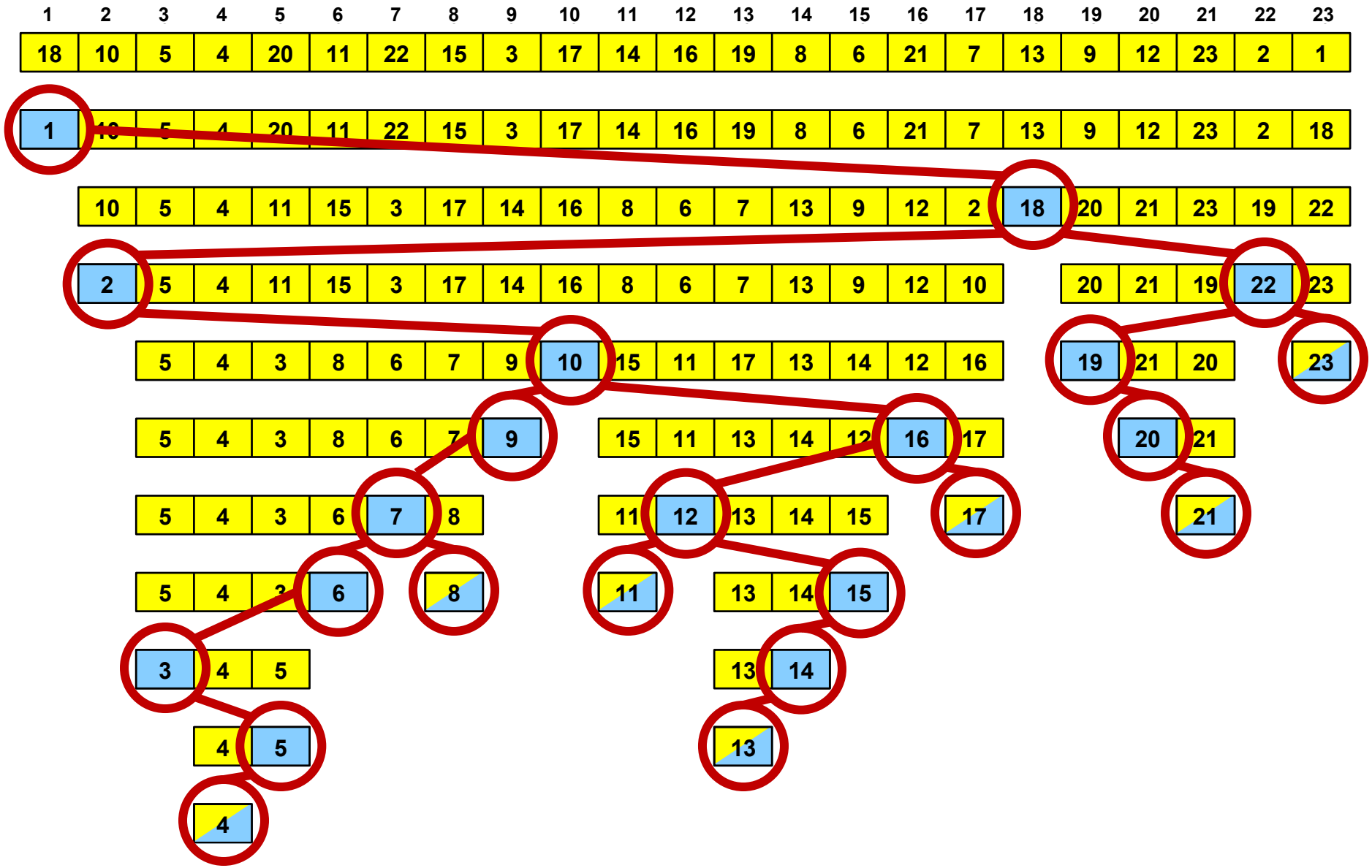
Random insertions into binary search trees

Algorithm

Insert n elements in random order

- As QuickSort argument, the **expected depth** of an element is $O(\log n)$ [CLRS, Problem 12.3]
- The expected **height of a tree** is $O(\log n)$, i.e., *all nodes* have expected depth $O(\log n)$ [CLRS 3rd edition, Chapter 12.4]

RandomizedQuicksort ≡ random insertions



Balanced search trees

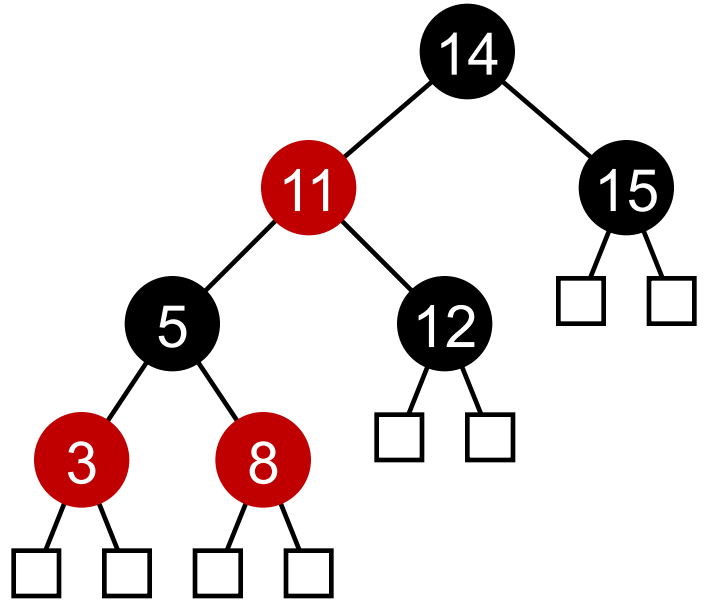
Balanced search trees restructure the trees during updates to guarantee $O(\log n)$ height

- AVL-trees [CLRS Pr. 13-3]
- BB[α]-trees
- Splay trees
- Locally balanced trees
- Red-black trees [CLRS Ch. 13]
- Randomized trees/treaps [CLRS 3ed Pr. 13-4]
- (2,3)-trees
- (2,4)-trees [CLRS Ch. 18.1, Pr. 18-2]
- B-trees [CLRS Ch. 18]
- Weight balanced B-trees
- ...

Algorithms and Data Structures

Balanced binary search trees: Red-black trees
[CLRS, Chapter 13]

Red-black trees



Goal

Binary search trees with height O(log n)

Invariants

- Every node is **red** or **black**
- Every **red** node has a **black** parent
- All paths from the root to an external leaf (nil) **have the same number of black nodes**

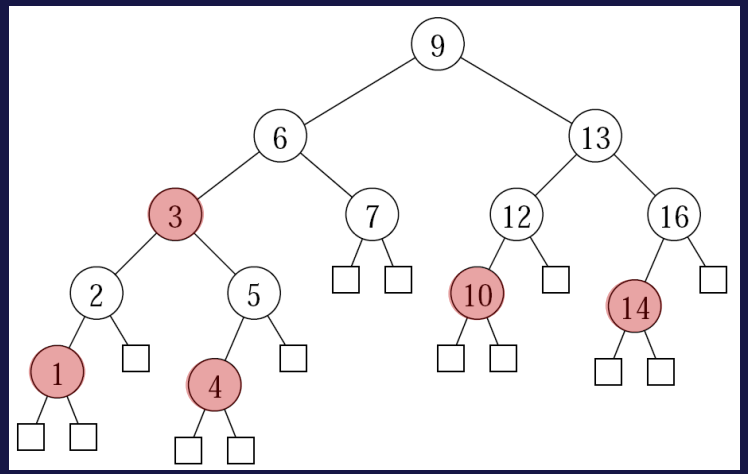
Theorem

A red-black tree has height ≤ 2·log (n+1)

Leo J. Guibas, Robert Sedgewick, *A Dichromatic Framework for Balanced Trees*, FOCS, 1978, doi: 10.1109/SFCS.1978.3

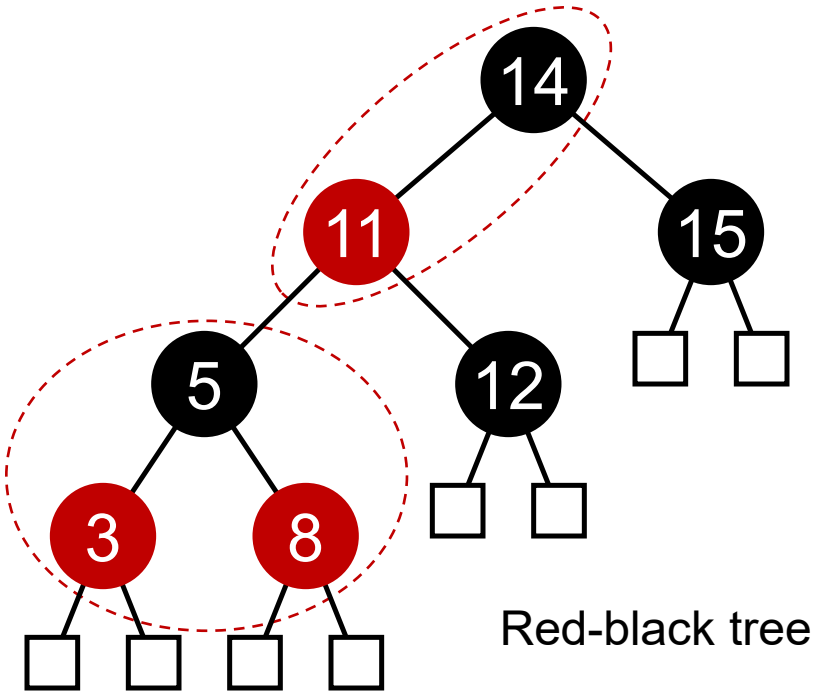
Which nodes should be colored red to make the tree a valid red-black tree ?

a) 1, 4, 10, 14
b) 2, 5, 10, 14
c) 1, 3, 4, 7, 12, 16
d) 1, 3, 4, 10, 14
e) None
f) Don't know

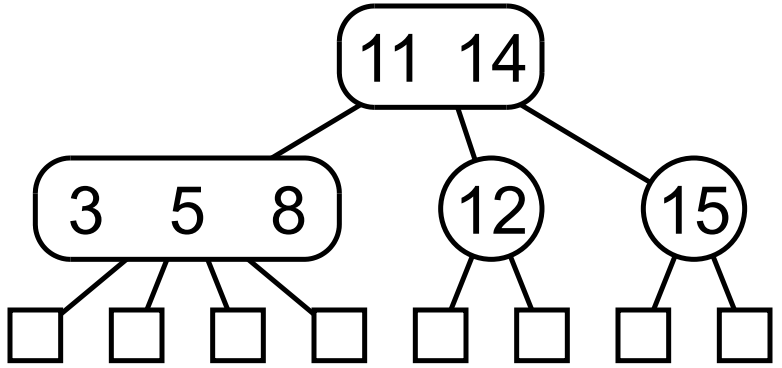


In general, not all binary search trees can be colored to become a valid red-black tree, and possible valid colorings are not necessarily unique

Red-black vs (2,4)-trees



Red-black tree



(2,4)-tree

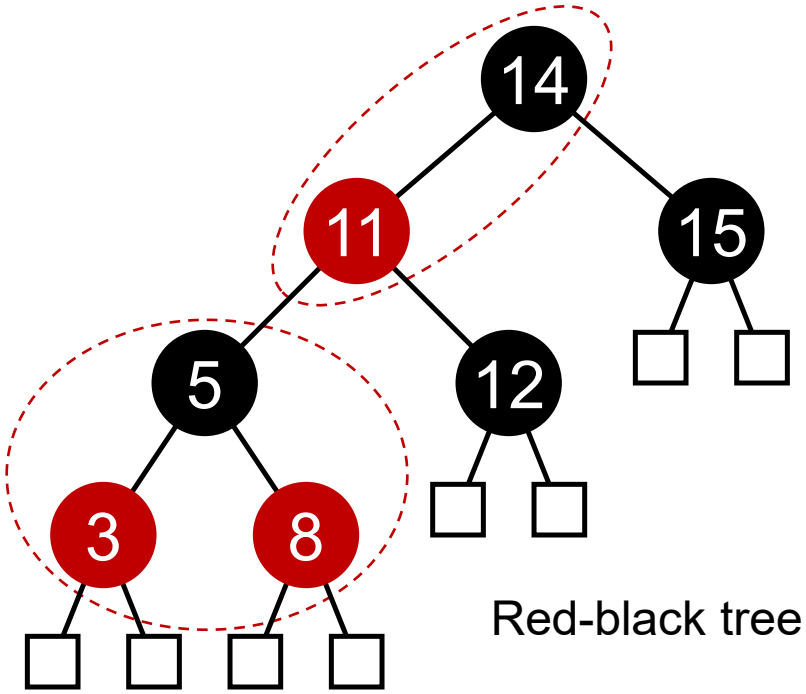
(2,4)-tree ≡ red-black tree

where all red nodes are merged with parent

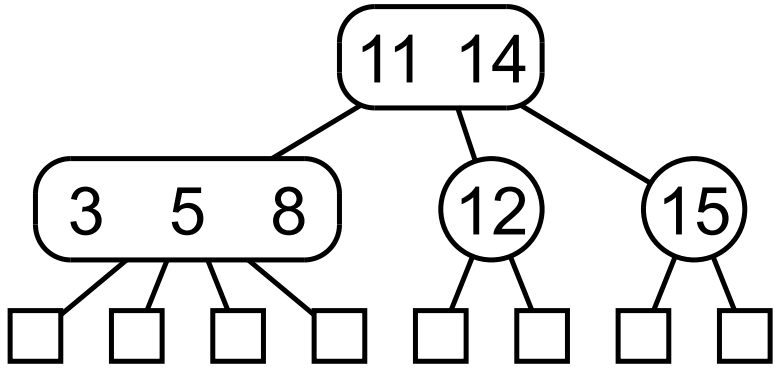
[(2,4)-trees = CLRS Chapter 18.1 & Problem 18-2]

(2,4)-tree properties

- All interne nodes **2 to 4 children**
- Node with *i* **children** stores *i*-1 **elements**
- Root-to-leaf paths all have equal length

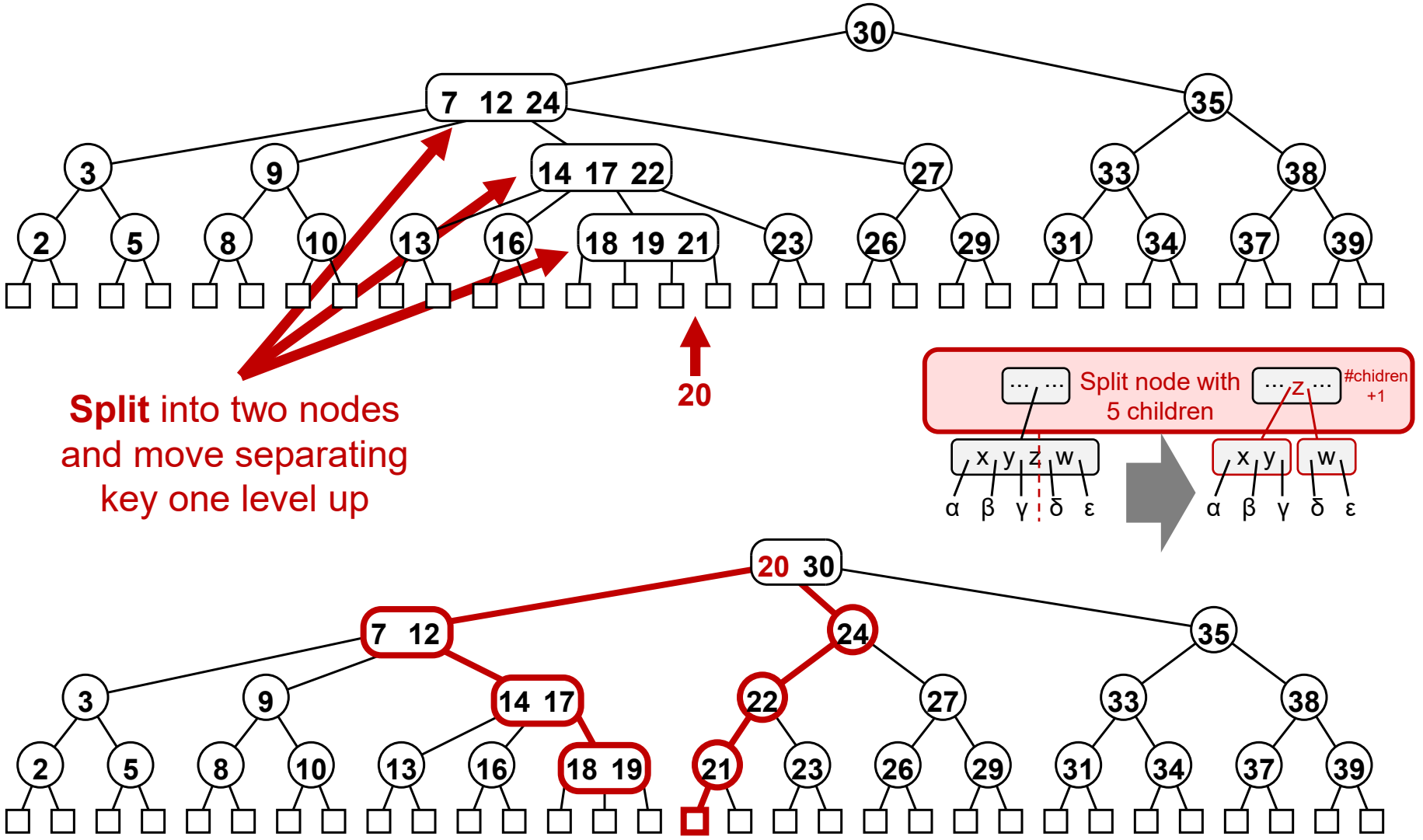


Red-black tree



(2,4)-tree

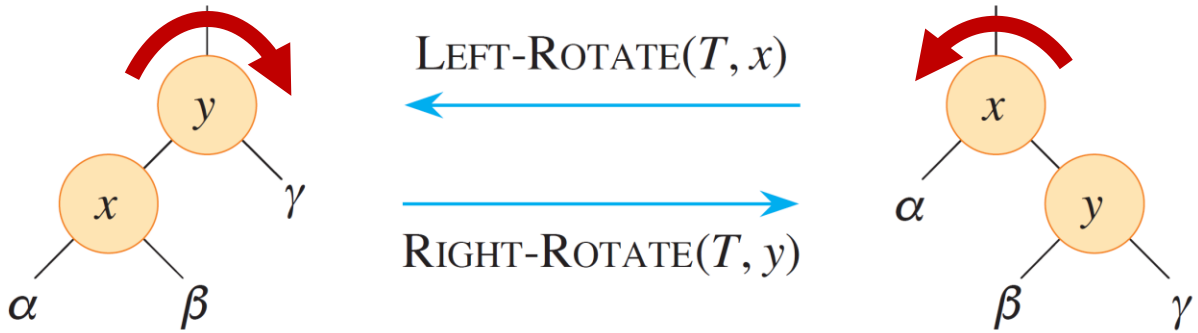
(2,4)-tree insertion



Red-black tree updates

Rotations

```
LEFT-ROTATE(T, x)
 1   y = x.right
 2   x.right = y.left
 3   if y.left ≠ T.nil
 4         y.left.p = x
 5   y.p = x.p
 6   if x.p == T.nil
 7         T.root = y
 8   elseif x == x.p.left
 9         x.p.left = y
10   else x.p.right = y
11   y.left = x
12   x.p = y
```



Insert

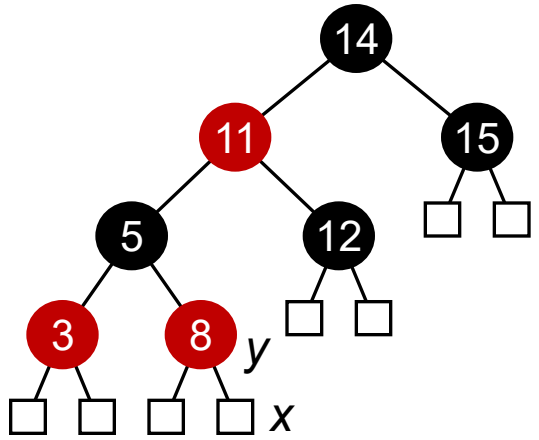
```
RB-INSERT(T, z)
1   x = T.root
2   y = T.nil
3   while x ≠ T.nil
4       y = x
5       if z.key < x.key
6           x = x.left
7       else x = x.right
8   z.p = y
9   if y == T.nil
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right = T.nil
16  z.color = RED
17  RB-INSERT-FIXUP(T, z)
```

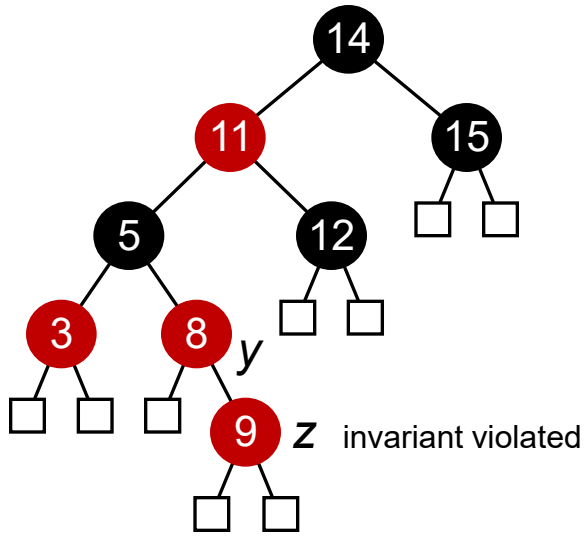
search

create z

Unbalanced insertion

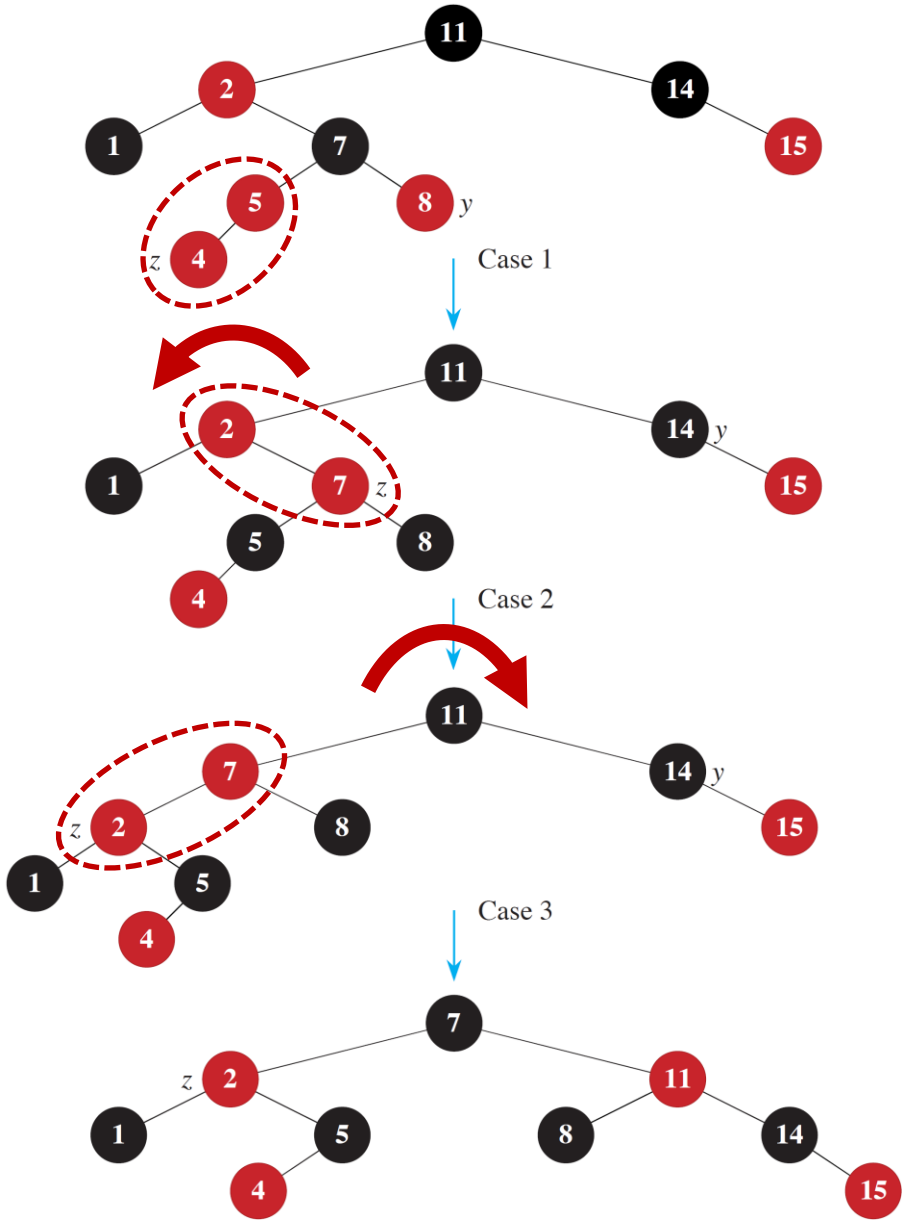
Insert(9)





invariant violated

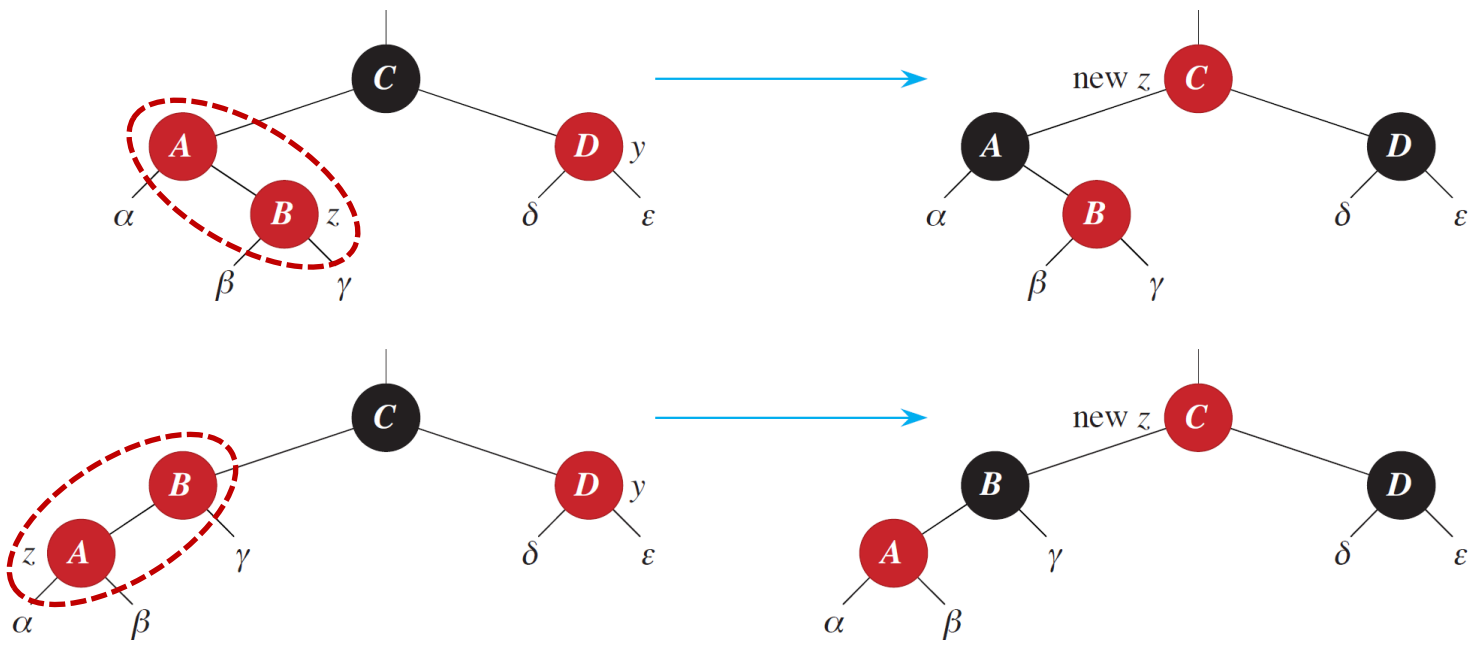
Example : Rebalancing after insertion



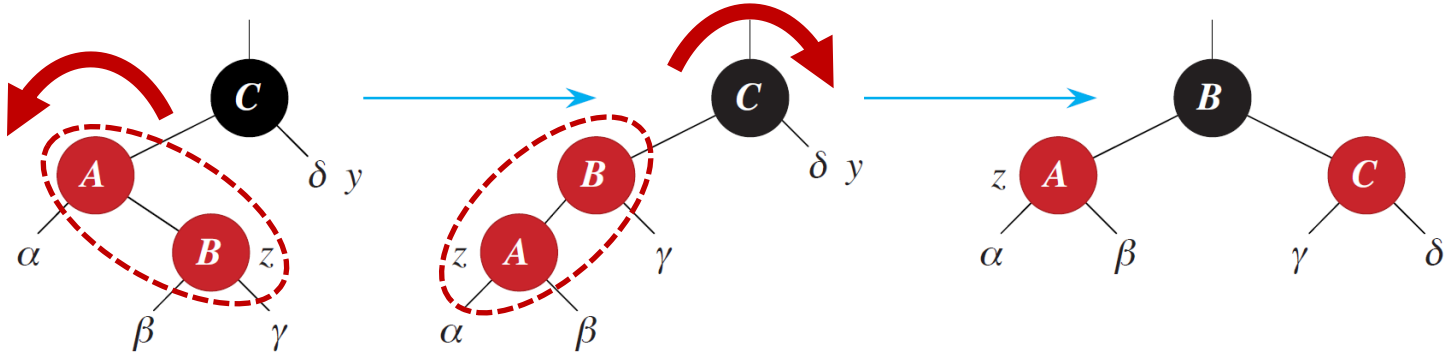
Red-black tree insert rebalancing

Recolor *C* and its two red children

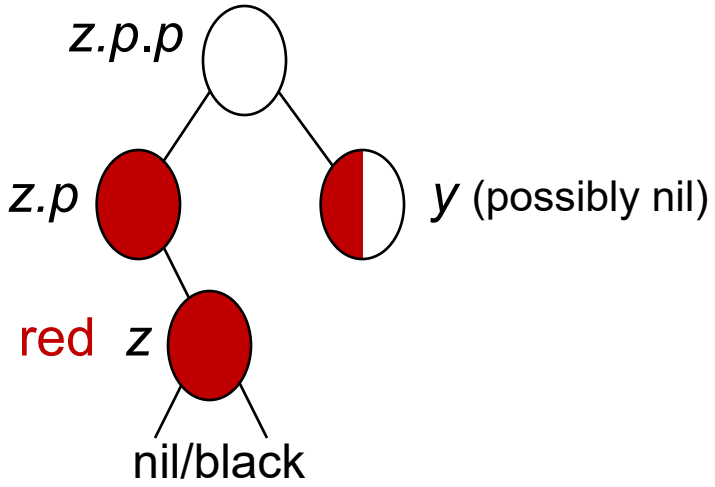
Case 1



Final rebalancing



```
RB-INSERT-FIXUP(T, z)
 1  while z.p.color == RED
 2      if z.p == z.p.p.left
 3          y = z.p.p.right
 4          if y.color == RED
 5              z.p.color = BLACK                    }
 6              y.color = BLACK                      }  case 1
 7              z.p.p.color = RED                    }
 8              z = z.p.p                            }
 9          else
10              if z == z.p.right
11                  z = z.p                          }
12                  LEFT-ROTATE(T, z)                }  case 2
13              z.p.color = BLACK                    }
14              z.p.p.color = RED                    }  case 3
15              RIGHT-ROTATE(T, z.p.p)               }
16      else // same as lines 3–15, but with "right" and "left" exchanged
30  T.root.color = BLACK
```



Worst-case # rotations during red-black tree insertion ?

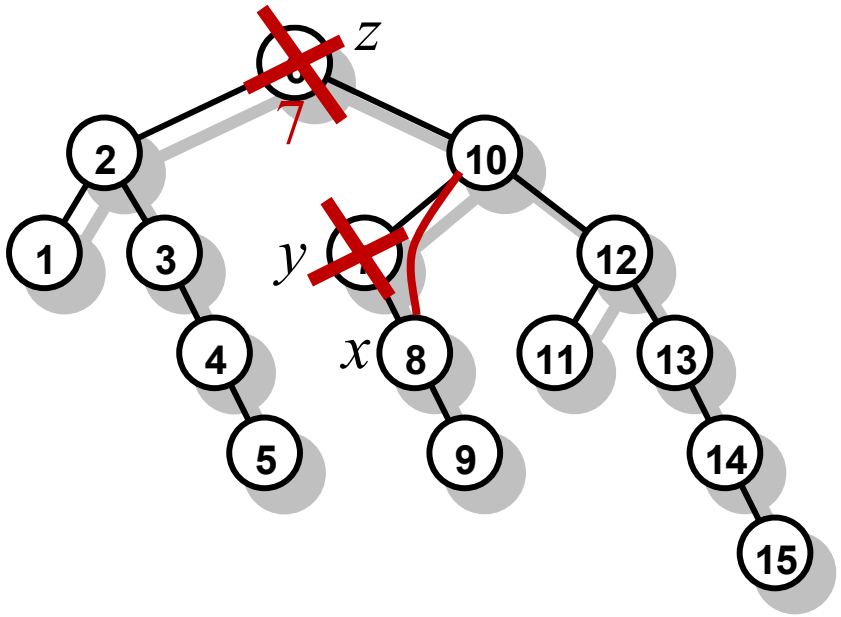
a) 1
b) 2
c) log *n*
d) 2·log *n*
e) Don't know

Delete

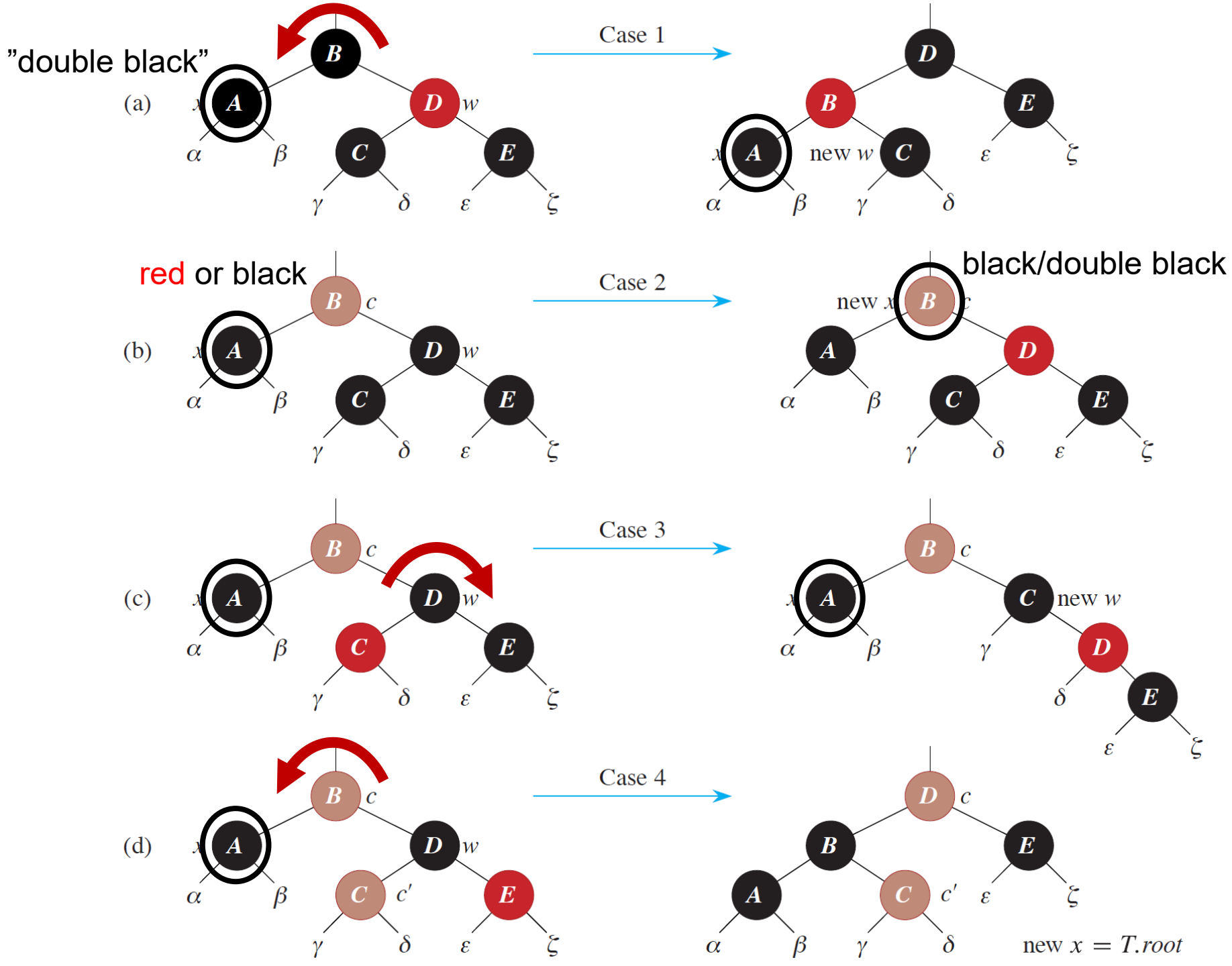
Unbalanced deletion

```
RB-DELETE(T, z)
 1  y = z
 2  y-original-color = y.color
 3  if z.left == T.nil
 4       x = z.right
 5       RB-TRANSPLANT(T, z, z.right)
 6  elseif z.right == T.nil
 7       x = z.left
 8       RB-TRANSPLANT(T, z, z.left)
 9  else y = TREE-MINIMUM(z.right)
10       y-original-color = y.color
11       x = y.right
12       if y ≠ z.right
13            RB-TRANSPLANT(T, y, y.right)
14            y.right = z.right
15            y.right.p = y
16       else x.p = y
17       RB-TRANSPLANT(T, z, y)
18       y.left = z.left
19       y.left.p = y
20       y.color = z.color
21  if y-original-color == BLACK
22       RB-DELETE-FIXUP(T, x)
```

x and *y.right* can be *T*.nil




```
RB-TRANSPLANT(T, u, v)
1  if u.p == T.nil
2       T.root = v
3  elseif u == u.p.left
4       u.p.left = v
5  else u.p.right = v
6  v.p = u.p
```



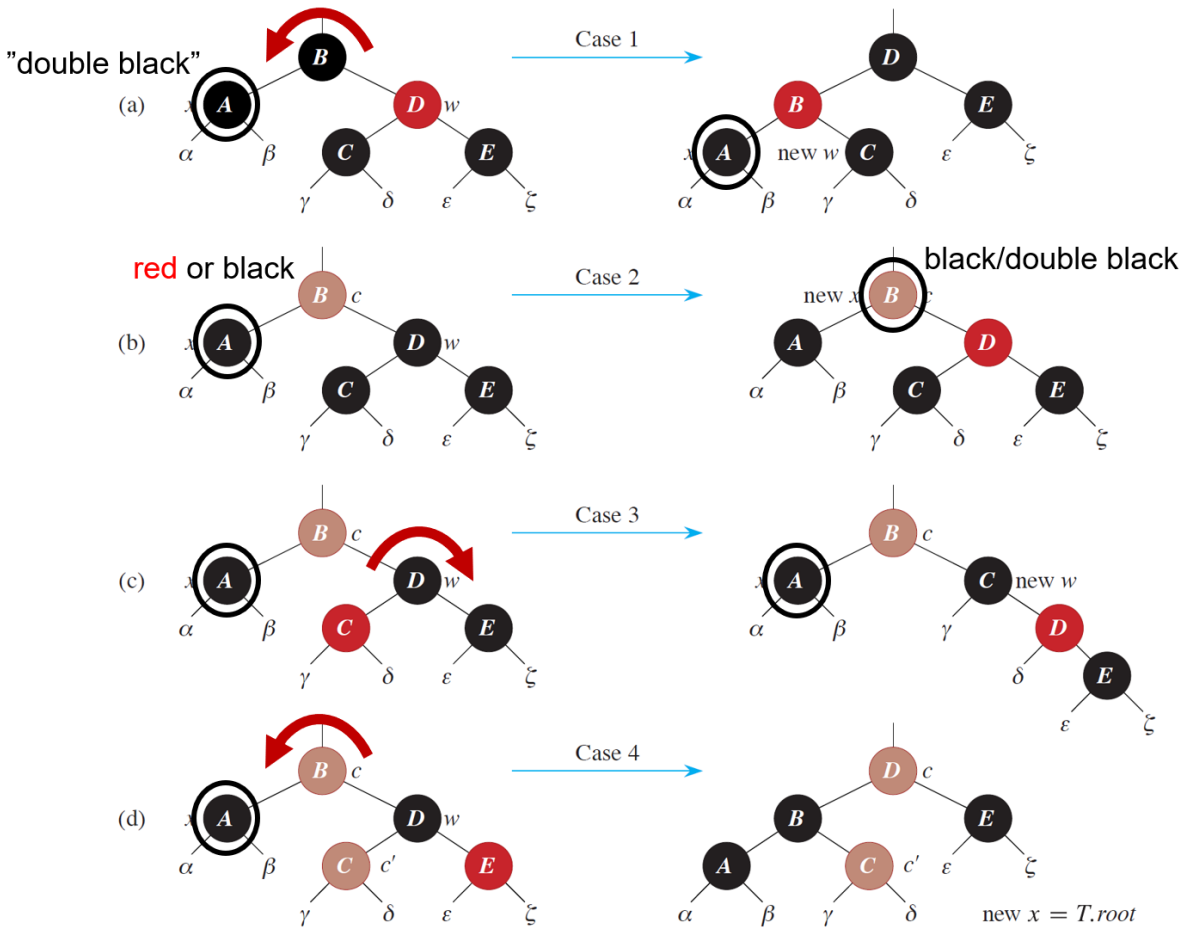
Case 1 → 2,3,4

Case 2 → Problem *x* moves closer to the root or done

Case 3 → 4 (and done)

Case 4 → done

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left                    // is x a left child?
 3          w = x.p.right                   // w is x's sibling
 4          if w.color == RED
 5              w.color = BLACK
 6              x.p.color = RED
 7              LEFT-ROTATE(T, x.p)                          } case 1
 8              w = x.p.right
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED
11              x = x.p                                      } case 2
12          else
13              if w.right.color == BLACK
14                  w.left.color = BLACK
15                  w.color = RED
16                  RIGHT-ROTATE(T, w)                       } case 3
17                  w = x.p.right
18              w.color = x.p.color
19              x.p.color = BLACK
20              w.right.color = BLACK
21              LEFT-ROTATE(T, x.p)                          } case 4
22              x = T.root
23      else // same as lines 3–22, but with "right" and "left" exchanged
44  x.color = BLACK
```



Dynamic dictionary : red-black trees

Search(S, x)
Insert(S, x)
Delete(S, x)

O(log n)

not curriculum

Red-black trees in Java and C++

	Java (JDK SE 25)	C++ (GCC 15.2)
Method	java.util.TreeMap	std::map
Documentation	https://docs.oracle.com/en/java/javase/25/docs/api/java.base/java/util/TreeMap.html	http://www.cplusplus.com/reference/map/map/
Source code	Install JDK from https://www.oracle.com/java/technologies/downloads/ File java.base\java\util\TreeMap.java from C:\ProgramFiles\Java\jdk-25\lib\src.zip	https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_tree.h

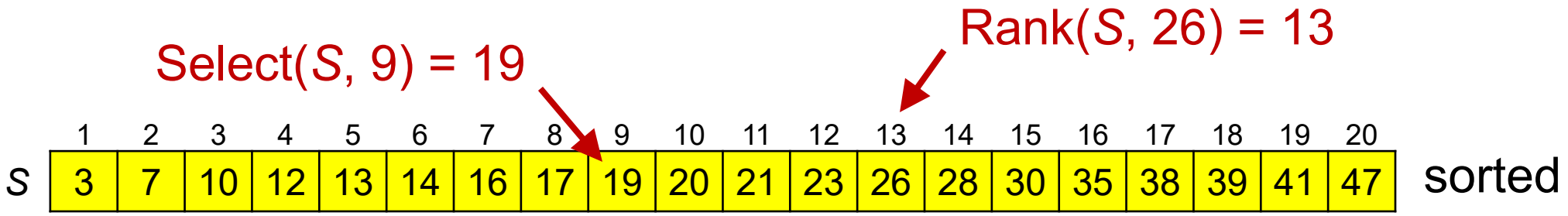
Algorithms and Data Structures

Augmented search trees: Dynamic rank, interval trees
[CLRS, Chapter 17]

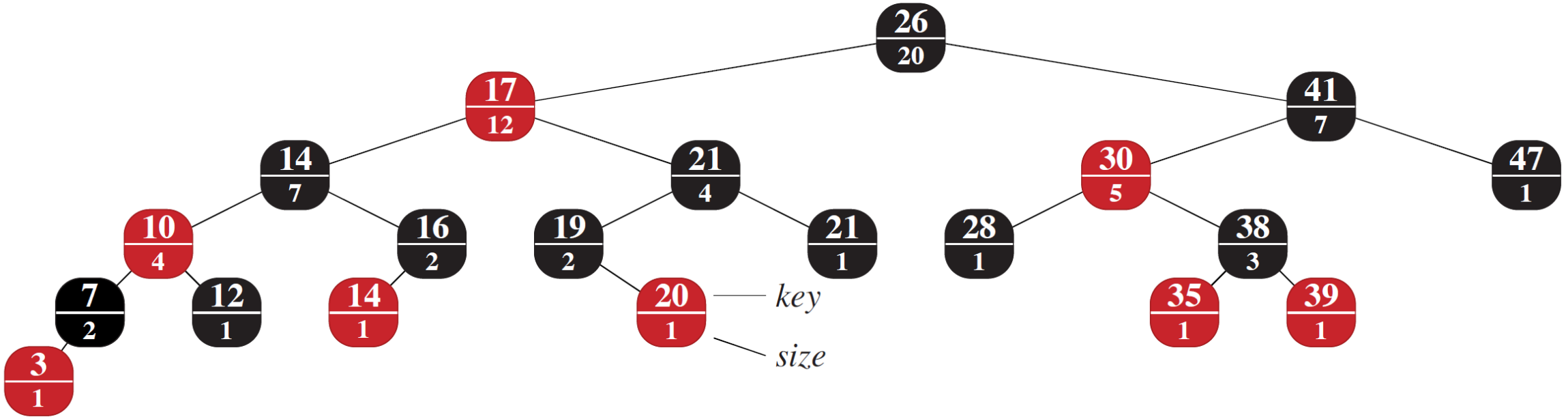
Fenwick trees (not curriculum)
[B, Chapter 2.1]

Dynamic rank (order statistics)

Insert(*S*, *x*) **Delete**(*S*, *x*) **Select**(*S*, *i*) **Rank**(*S*, *x*)	O(log *n*)



Dynamic rank



- Find the i^{th} smallest, insert, delete
- Maintain **red-black** tree
- Augment each node with the size of the subtree

Computation of augmentation *x.size* ?

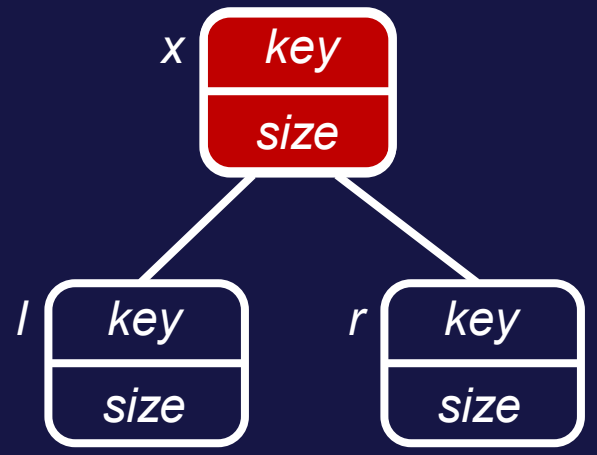
a) *x.size* = *l.size* + *r.size*

b) *x.size* = *l.size* + *r.size* + 1

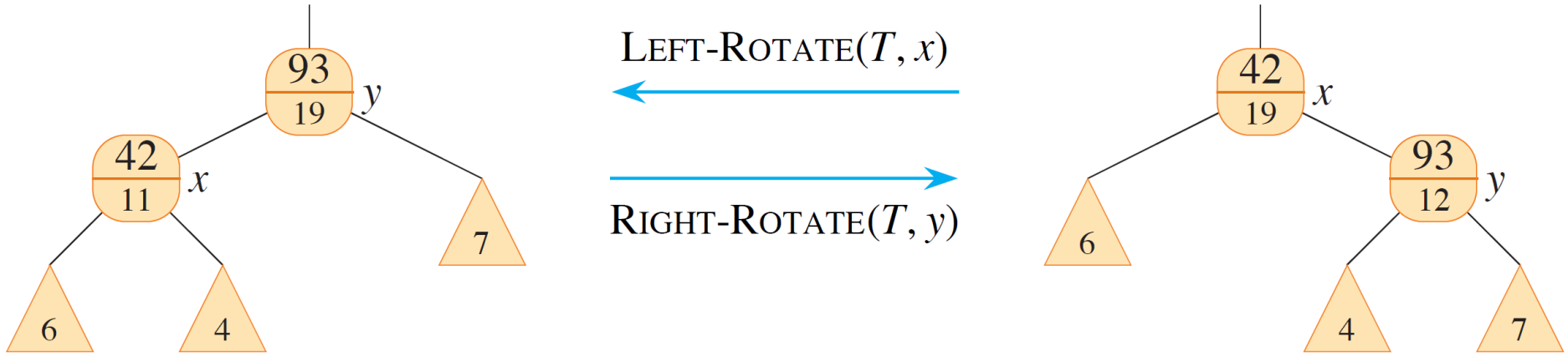
c) *x.size* = x.size + *l.size* + *r.size*

d) *x.size* = x.size + *l.size* + *r.size* + 1

e) Don't know



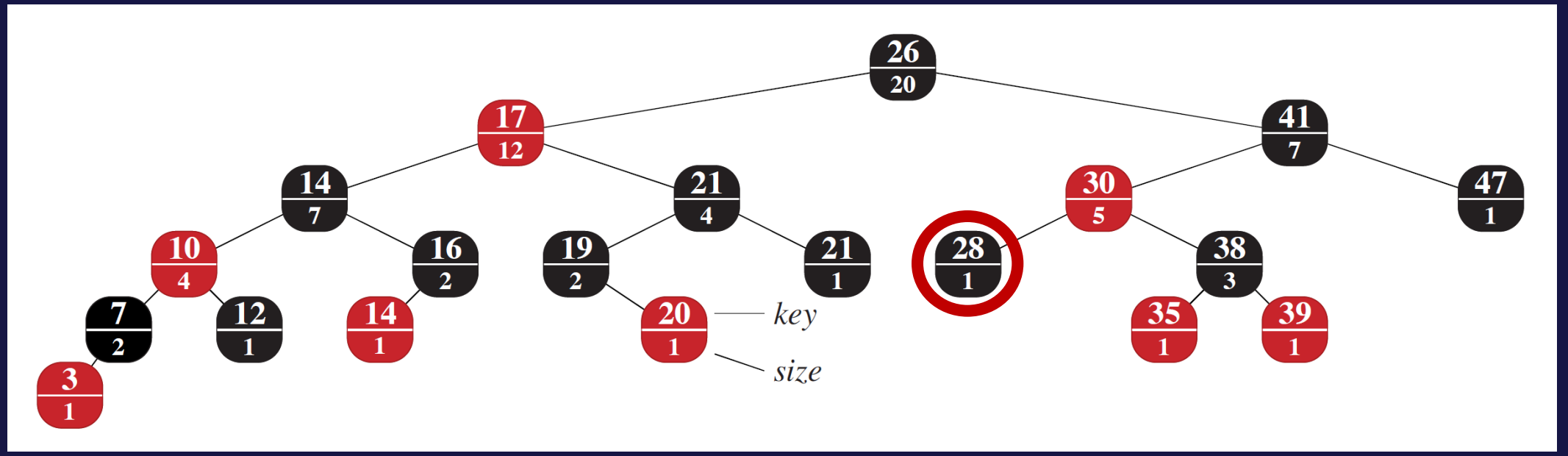
Dynamic rank



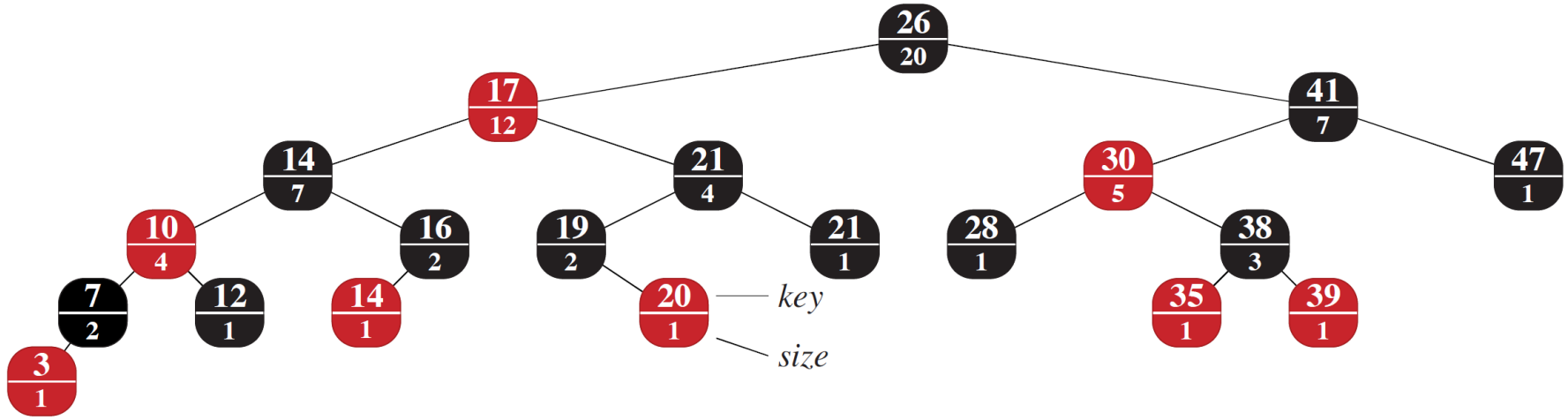
- Insert / delete: update size on path to root
- Under rotations in **red-black** update size

Rank of 28 ?

a) 13
b) 14 = 12 + 1 + 1
c) 16
d) 28
e) Don't know



Dynamic rank



OS-RANK(T, x)

```
1  r = x.left.size + 1
2  y = x
3  while y ≠ T.root
4        if y == y.p.right
5              r = r + y.p.left.size + 1
6        y = y.p
7  return r
```

OS-SELECT(x, i)

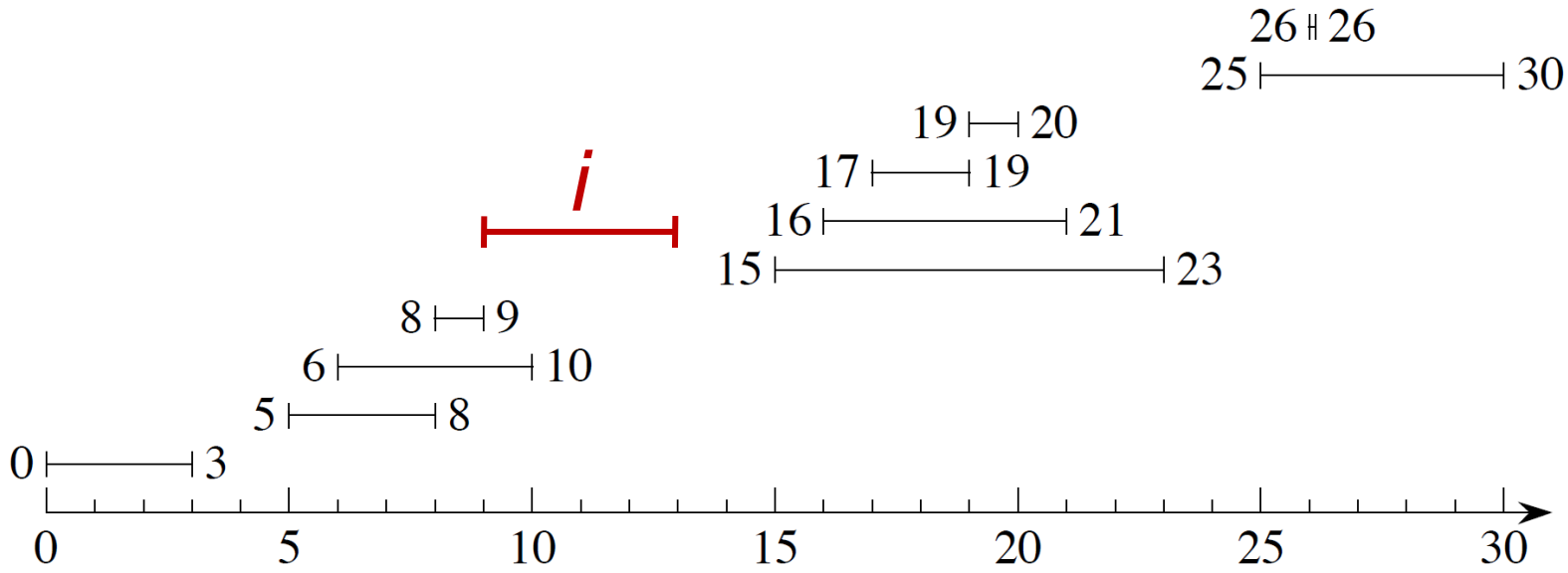
```
1  r = x.left.size + 1
2  if i == r
3        return x
4  elseif i < r
5        return OS-SELECT(x.left, i)
6  else return OS-SELECT(x.right, i − r)
```

Dynamic rank

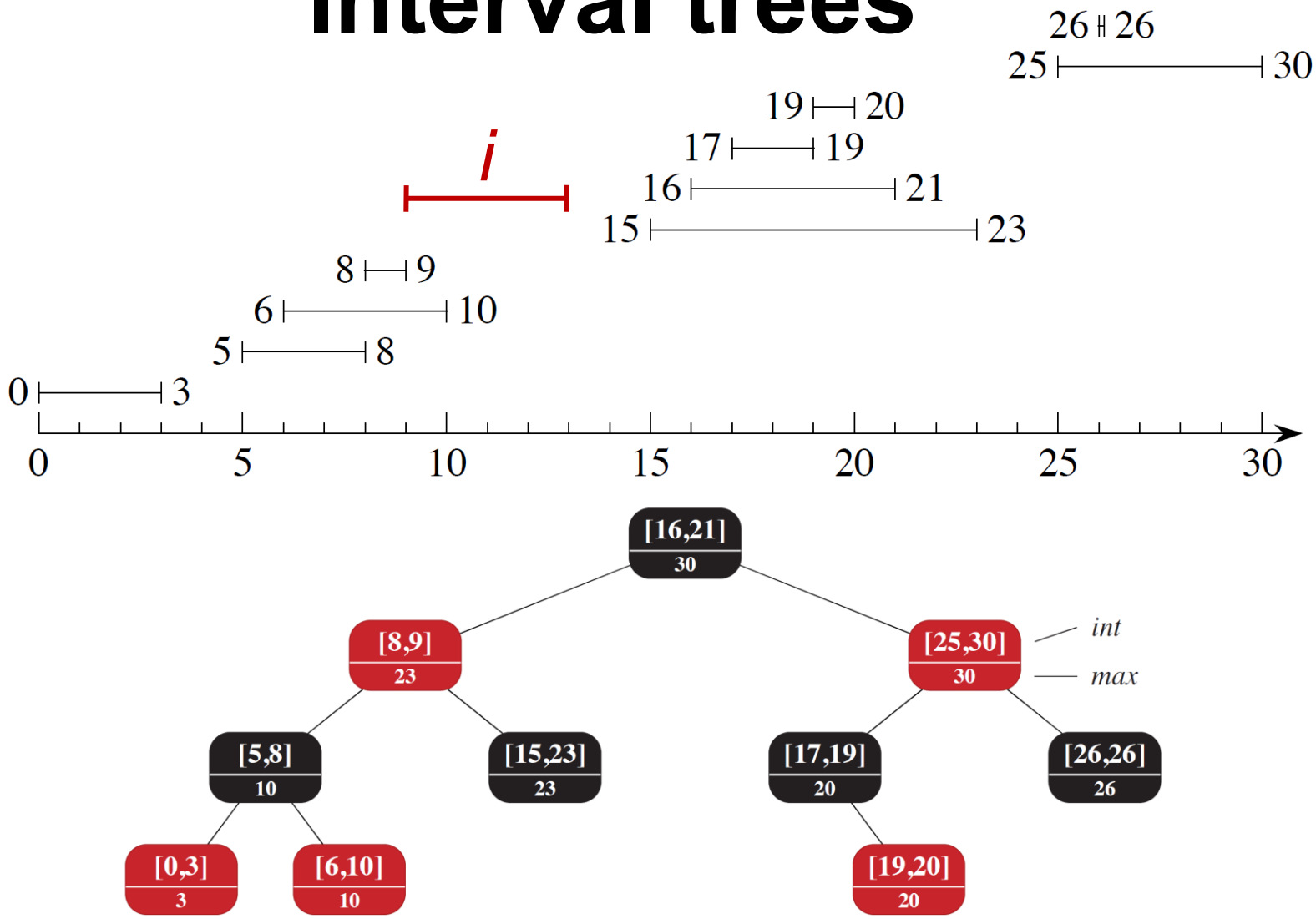
Insert(S, x) **Delete**(S, x) **Select**(S, i) **Rank**(S, x)	O(log n)

Interval trees

- Store a set of intervals
- Insert and delete intervals
- Find an interval with overlap with a query interval *i*



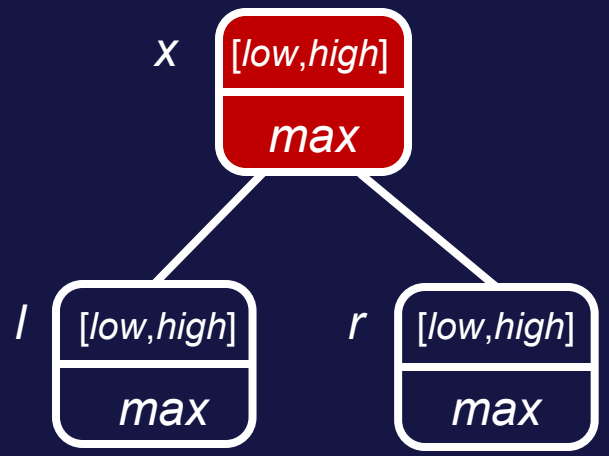
Interval trees



- **Red-black** tree ordered by left endpoint
- Augment nodes with maximum right endpoint in subtree

Computation of augmentation *x.max* ?

a) *x.max* = *r.max*
b) *x.max* = *r.high*
c) *x.max* = max(*r.max*, *x.high*)
d) *x.max* = max(*l.max*, *x.high*)
e) *x.max* = max(*l.max*, *r.max*, *x.high*)
f) Don't know



Interval trees

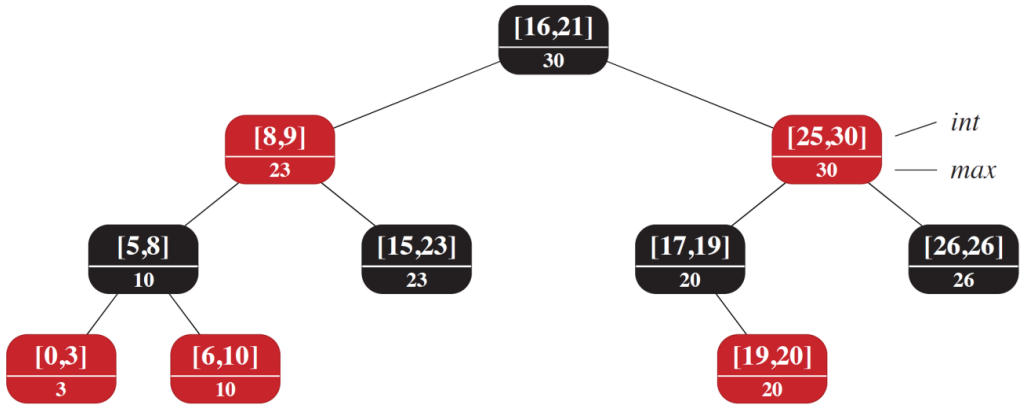
i overlaps *x*

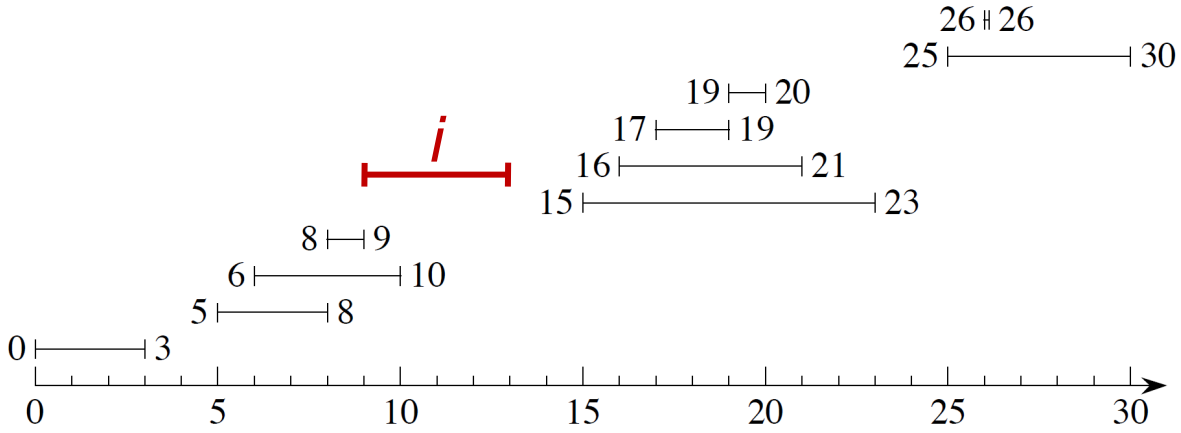
⇕

i.low ≤ *x.high* and *x.low* ≤ *i.high*

```
INTERVAL-SEARCH(T, i)
1  x = T.root
2  while x ≠ T.nil and i does not overlap x.int
3      if x.left ≠ T.nil and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

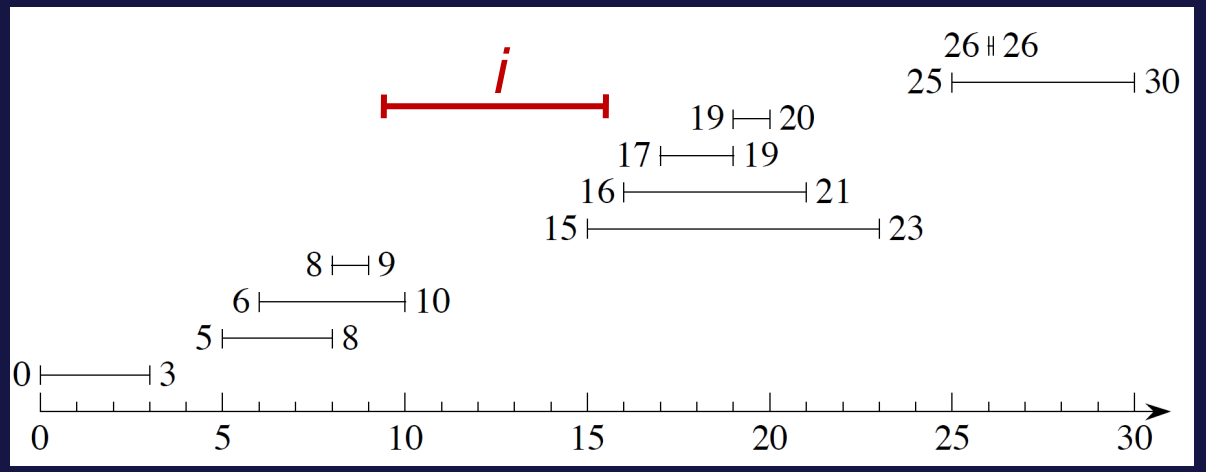
tricky

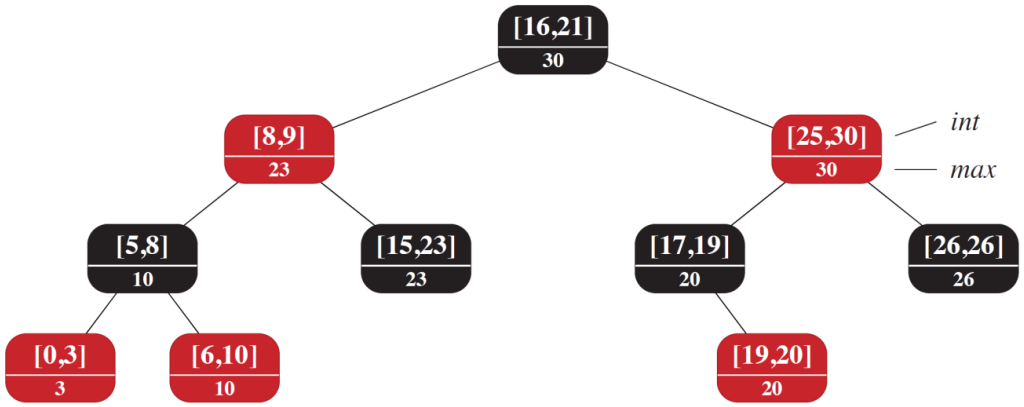




Interval-Search(*T*, [9.5,15.5]) ?

a) [6, 10]
b) [15, 23]
c) Don't know



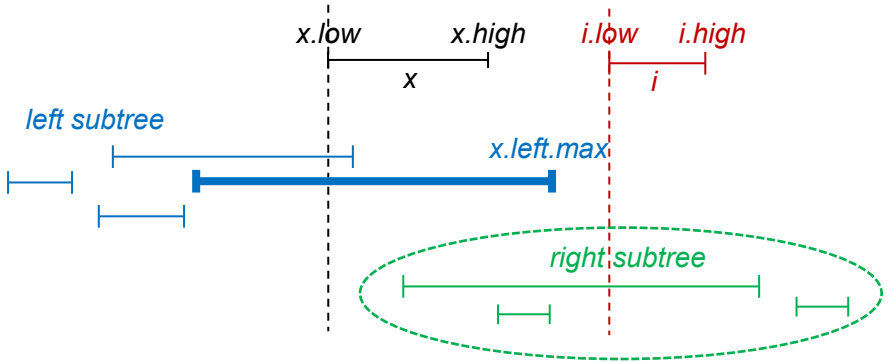


INTERVAL-SEARCH(T, i)

```
1  x = T.root
2  while x ≠ T.nil and i does not overlap x.int
3      if x.left ≠ T.nil and x.left.max ≥ i.low
4          x = x.left
5      else x = x.right
6  return x
```

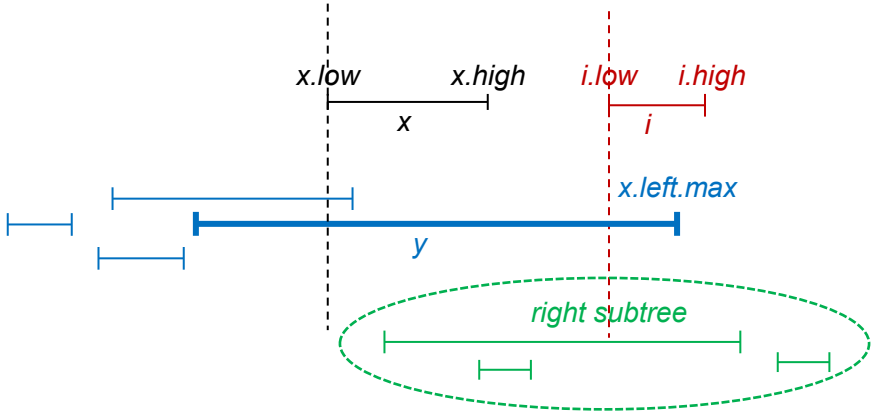
Interval trees – correctness

x left of *i* **and** *x.left.max* < *i.low*



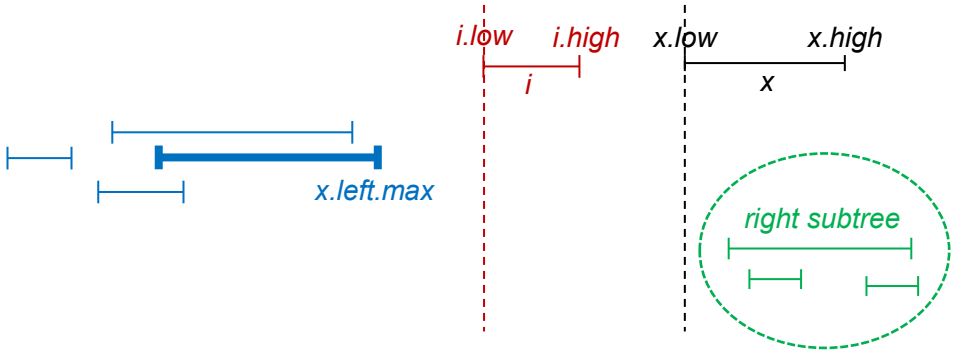
No overlap with *i* in left subtree, safe to **go right**

x left of *i* **and** *x.left.max* ≥ *i.low*



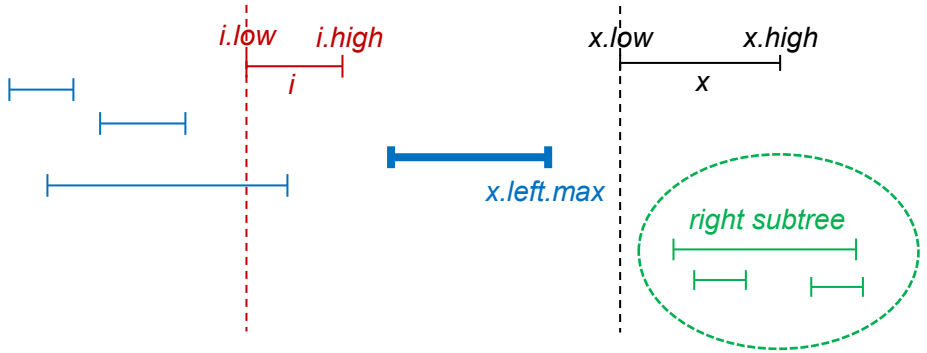
i overlaps with interval *y* in left subtree, safe to **go left**

i left of *x* **and** *x.left.max* < *i.low*



i does not overlap, safe to **go right**

i left of *x* **and** *x.left.max* ≥ *i.low*



i no overlap with right subtree, safe to **go left**

Interval trees

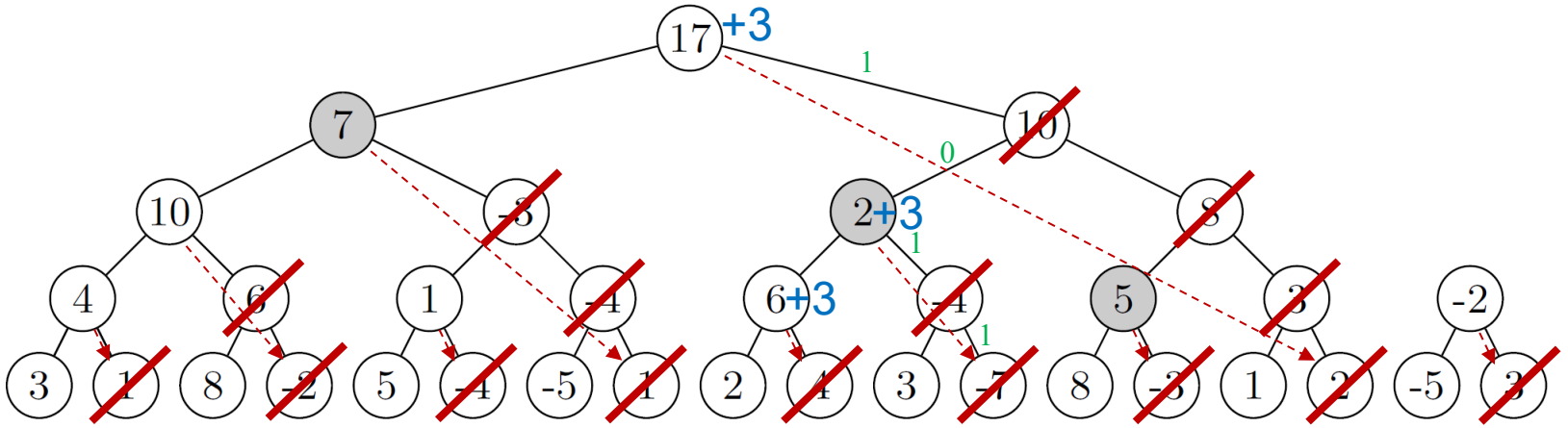
Insert(S, i)
Delete(S, i)
Search(S, i)

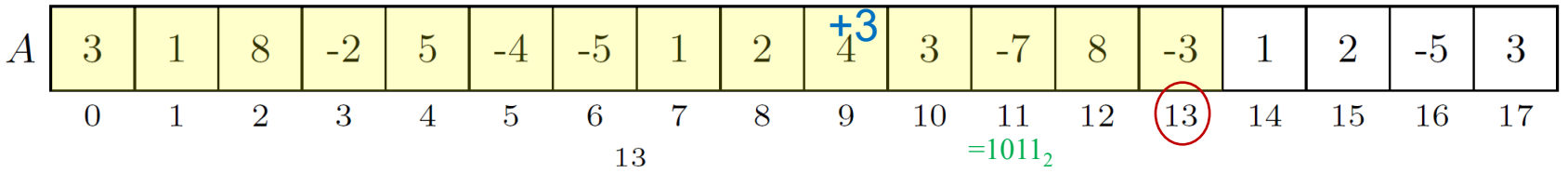
O(log n)

not curriculum

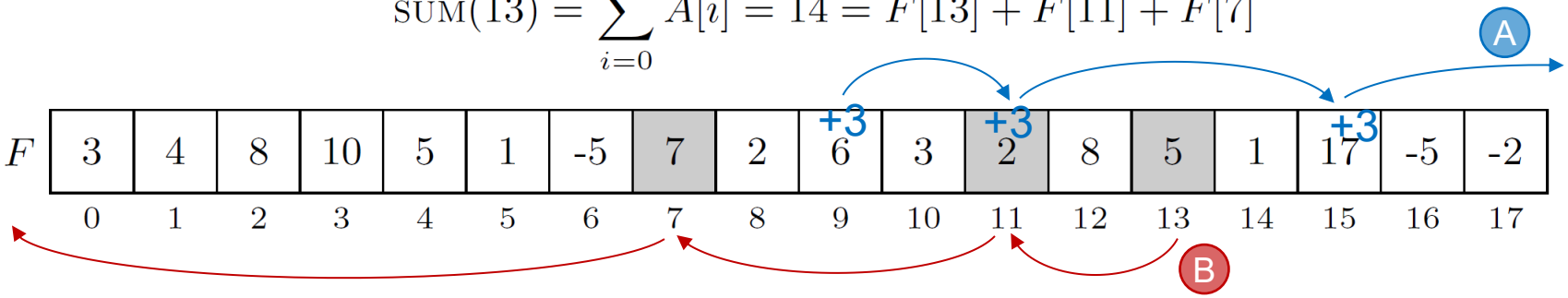
Fenwick trees

Dynamic prefix sums in O(log *n*)





$$\text{SUM}(13) = \sum_{i=0}^{13} A[i] = 14 = F[13] + F[11] + F[7]$$



Algorithm $\text{INIT}(n)$

```
1  F[0..n] = array with zeros
```

Algorithm $\text{INC}(i, d)$

```
1  while i ≤ n do
2      F[i] = F[i] + d
3      i = i | (i + 1)          A
```

Algorithm $\text{SUM}(i)$

```
1  s = 0
2  while i ≥ 0 do
3      s = s + F[i]
4      i = (i & (i + 1)) − 1    B
5  return s
```

A	B
9 = 01001₂	13 = 1101₂
11 = 01011₂	11 = 1011₂
15 = 01111₂	7 = 0111₂
31 = 11111₂	-1

Peter M. Fenwick, *A new data structure for cumulative frequency tables,* Software: Practice and Experience, 1994, :10.1002/spe.4380240306

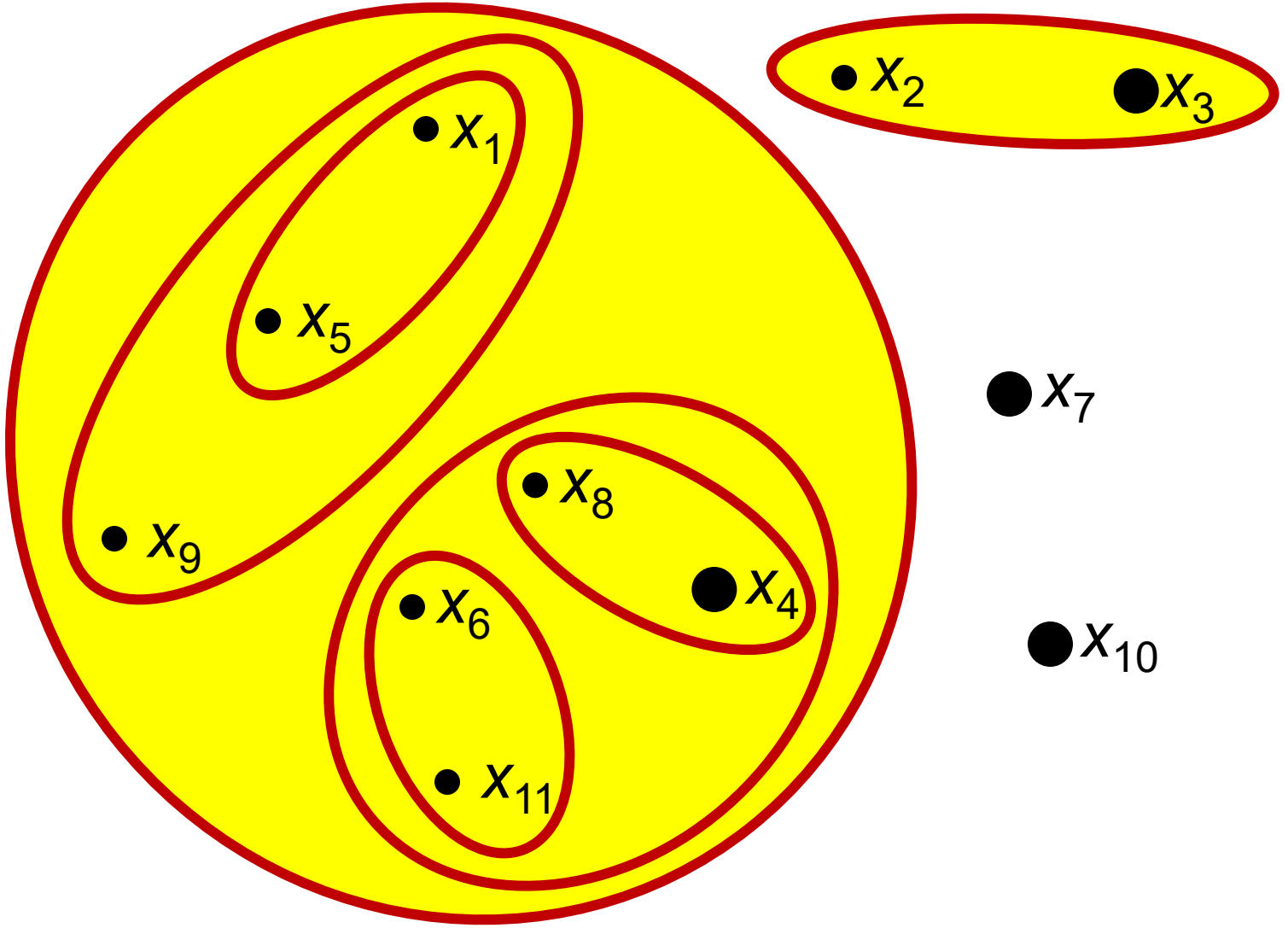
Algorithms and Data Structures

Union-find
[CLRS, Chapter 19.1-19.3]

Union-find

MakeSet(*x*)	Create set $S = \{ x \}$
Union(*x*, *y*)	Replace $S_x = \{..., x, ...\}$ and $S_y = \{..., y, ...\}$ with $S_x \cup S_y$ (assuming *x* and *y* are in different sets)
FindSet(*x*)	Return a *representative* for $S_x = \{ ..., x, ... \}$ FindSet(*x*) = FindSet(*y*) if and only if *x* and *y* are in the same set

Example : Union-find



x_i	FindSet(x_i)
x_1	x_4
x_2	x_3
x_3	x_3
x_4	x_4
x_5	x_4
x_6	x_4
x_7	x_7
x_8	x_4
x_9	x_4
x_{10}	x_{10}
x_{11}	x_4

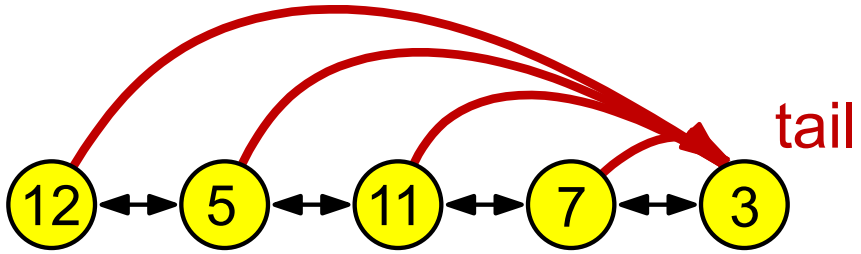
Set representation (I)

- Set = doubly-linked list
- **MakeSet** = create new node
- **FindSet** = return rightmost node (tail)
- **Union** = concatenate lists

MakeSet(S, x)	O(1)
FindSet(x)	O($\|S_x\|$)
Union(x, y)	O($\|S_x\| + \|S_y\|$)

Set representation (II)

- Set = doubly-linked list + tail pointers
- **MakeSet** = create new node
- **FindSet** = return tail node
- **Union** = concatenate lists and update tail pointers



MakeSet(S, x)	O(1)
FindSet(x)	O(1)
Union(x, y)	O(min($\|S_x\|$, $\|S_y\|$))

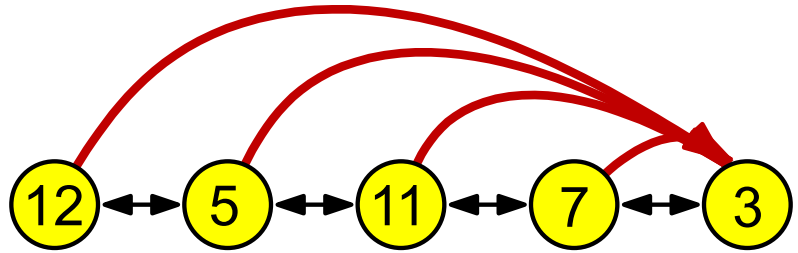
Set representation (II)
sequence of unions

Theorem

A sequence of n **Union** takes time $O(n \cdot \log n)$

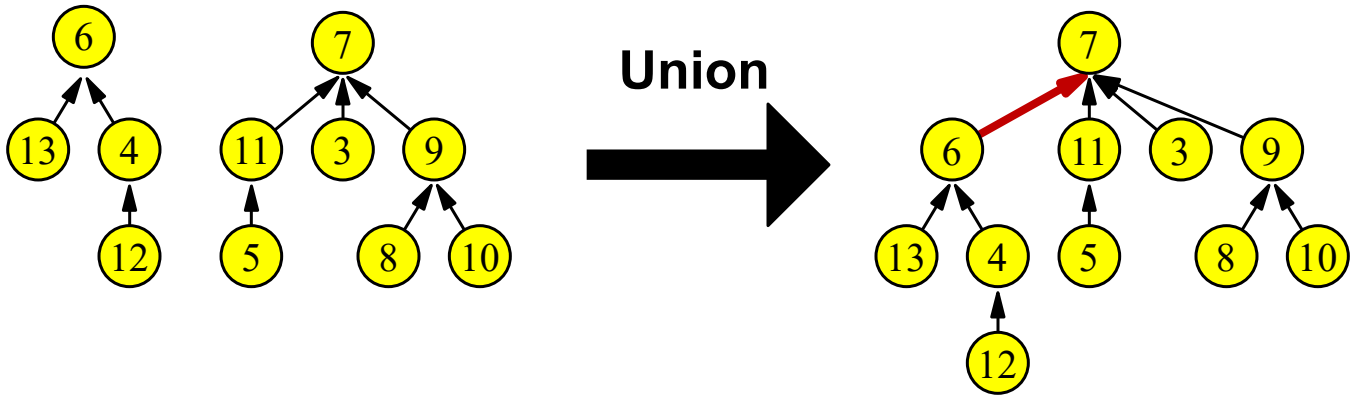
Proof

When a tail pointer is moved, the new list is at least twice the size, i.e., can be moved at most log n times



Set representation (III)

- Set = tree (only parent pointers, no child pointers)
- **MakeSet** = create new node
- **FindSet** = return root
- **Union** = attach ”small” tree below root of ”big” tree (size = # elements)



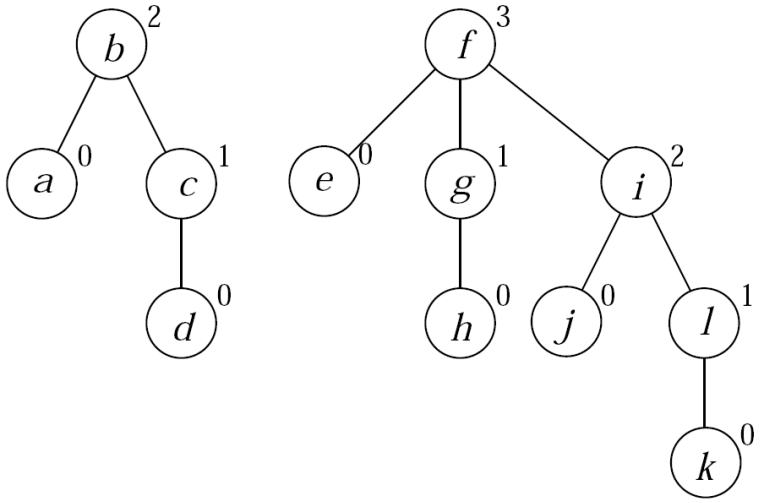
M. J. Fischer. *Efficiency of equivalence algorithms*, Symposium on the Complexity of Computer Computations, 1972, doi: 10.1007/978-1-4684-2001-2_14

Set representation (III)

Height of a tree representing a set of size n ?

a) O(1)
b) O(log n)
c) O($\sqrt{n}$)
d) O(n)
e) Don't know

Path compression




```
MAKE-SET(x)
1  x.p = x
2  x.rank = 0

UNION(x, y)
1  LINK(FIND-SET(x), FIND-SET(y))

LINK(x, y)
1  if x.rank > y.rank                    linking by rank
2       y.p = x
3  else x.p = y
4       if x.rank == y.rank
5            y.rank = y.rank + 1

FIND-SET(x)
1  if x ≠ x.p
2     x.p = FIND-SET(x.p)                path compression
3  return x.p
```

Robert Endre Tarjan, *Efficiency of a good but not linear set union algorithm,* Journal of the ACM, 1975. doi:10.1145/321879.321884

Set representation (III)
analysis linking by rank

Lemma

$$height[root] \le rank[root]$$
$$size[root] \ge 2^{rank[root]}$$

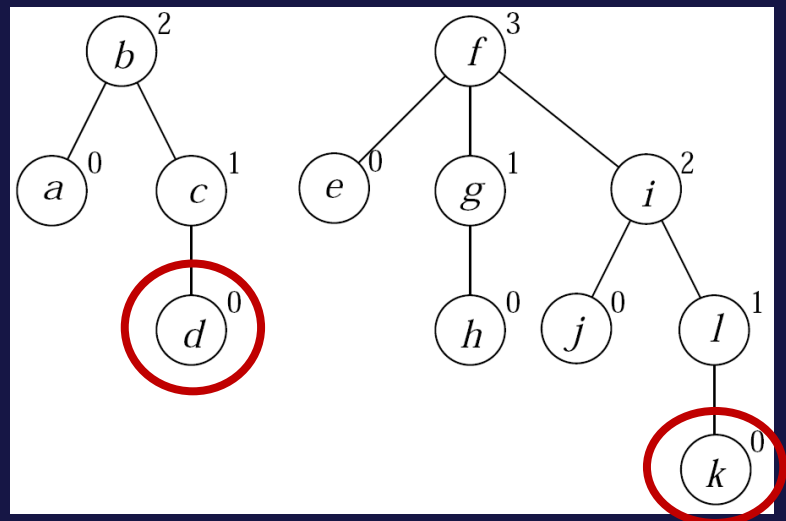
Proof Induction

MakeSet(S, x)	O(1)
Union(x, y)	O((log $\|S_x\|$) + (log $\|S_y\|$))
FindSet(x)	O(log $\|S_x\|$)

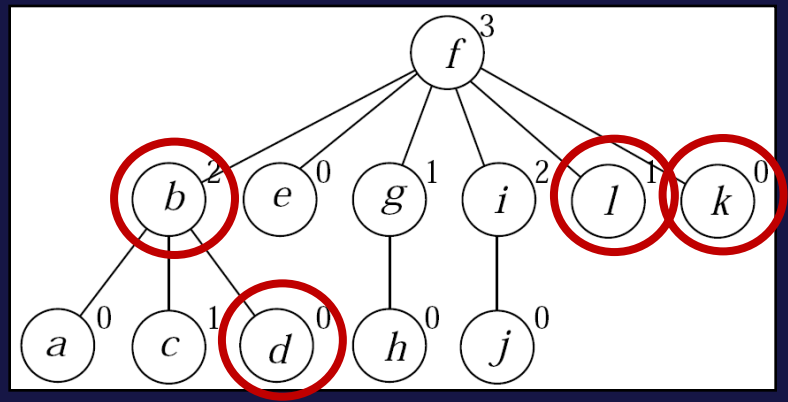
(above does not assume path compression)

children at the root after Union(*d*, *k*) when using linking-by-rank and path compression ?

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) Don't know



Exam summer 2009



proof not curriculum

Set representation (III)
analysis linking-by-rank and path compression

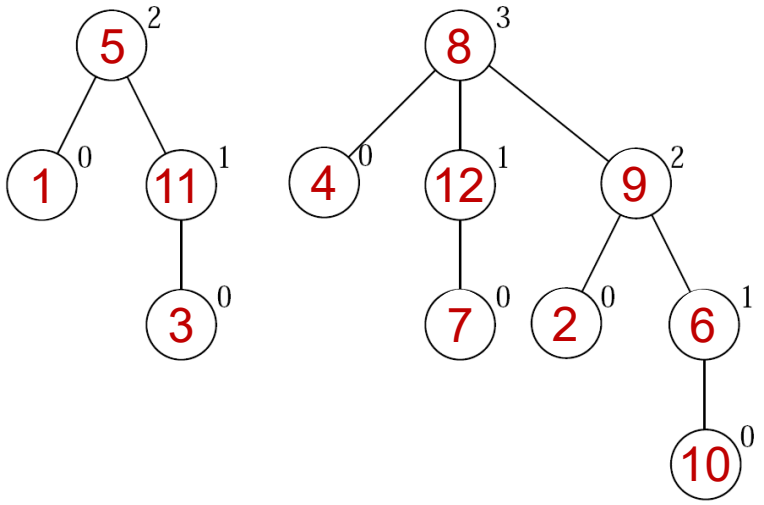
Theorem ([CLRS, Theorem 19.14])

n **MakeSet** and *m* **Union** and **Find** operations take total time $O(m \cdot \alpha(n))$ where $\alpha(n)$ is the inverse Ackerman function, where $\alpha(n) \leq 4$ for all practical values of *n*

Robert Endre Tarjan, *Efficiency of a good but not linear set union algorithm,* Journal of the ACM, 1975, doi: 10.1145/321879.321884

Compact representation

- Elements = integers 1,...,*n*
- *rank* and *p* arrays



	1	2	3	4	5	6	7	8	9	10	11	12
p	5	9	11	8	5	9	12	8	8	6	5	8

	1	2	3	4	5	6	7	8	9	10	11	12
rank	0	0	0	0	2	1	0	3	2	0	1	1

Non-recursive

```
MAKESETS(n)
1   Let p and rank be arrays of size n
2   for i = 1 to n
3       p[i] = i
4       rank[i] = 0

UNION(x, y)
1   LINK(FINDSET(x), FINDSET(y))

LINK(x, y)
1   if rank[x] > rank[y] then
2       p[y] = x
3   else
4       p[x] = y
5       if rank[x] = rank[y] then
6           rank[y] = rank[y] + 1
```

```
FINDSET(x)
1   root = x
2   while root ≠ p[root] do
3       root = p[root]
4   while p[x] ≠ root do
5       y = p[x]
6       p[x] = root
7       x = y
8   return root
```

- **FindSet** requires 2 passes

Only path compression

```
MAKESETS(n)
1   Let p and rank be arrays of size n
2   for i = 1 to n
3       p[i] = i
4       rank[i] = 0

UNION(x, y)
1   LINK(FINDSET(x), FINDSET(y))

LINK(x, y)
1   if rank[x] > rank[y] then
2       p[y] = x
3   else
4       p[x] = y
5       if rank[x] = rank[y] then
6           rank[y] = rank[y] + 1
```

```
FINDSET(x)
1   root = x
2   while root ≠ p[root] do
3       root = p[root]
4   while x ≠ root do
5       y = p[x]
6       p[x] = root
7       x = y
8   return x
```

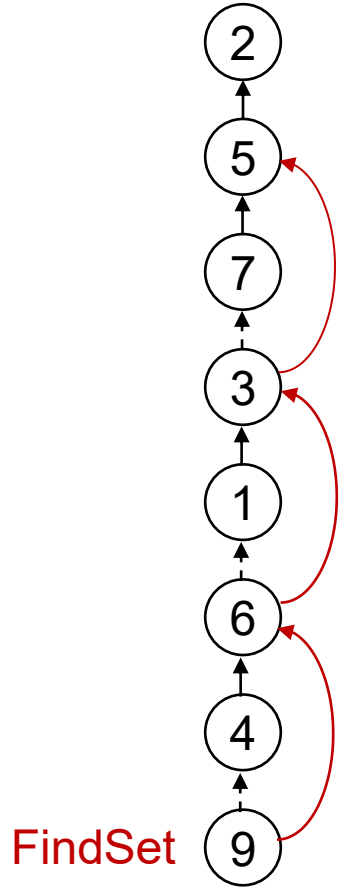
- **FindSet** requires 2 passes
- Only requires array p
- m operations $O(m \log n)$ time

Robert Endre Tarjan and Jan van Leeuwen, *Worst-case analysis of set union algorithms*, Journal of the ACM, 1984, doi: 10.1145/62.2160

Single pass path compression

```
MAKESETS(n)
1  Let p and rank be arrays of size n
2  for i = 1 to n
3      p[i] = i
4      rank[i] = 0

UNION(x, y)
1  LINK(FINDSET(x), FINDSET(y))

LINK(x, y)
1  if rank[x] > rank[y] then
2      p[y] = x
3  else
4      p[x] = y
5      if rank[x] = rank[y] then
6          rank[y] = rank[y] + 1
```

```
FINDSET(x)
1  while x ≠ x.p do
2      p[x] = p[p[x]]
3      x = p[x]
4  return x
```



- **FindSet** makes 1 pass of path
- Shortcut every second edge
- *m* operations total time O(*m*·α(*n*))

Robert Endre Tarjan and Jan van Leeuwen, *Worst-case analysis of set union algorithms*, Journal of the ACM, 1984, doi: 10.1145/62.2160

Algorithms and Data Structures

Amortized analysis
[CLRS, Chapter 16]

Robert Endre Tarjan, *Amortized computational complexity*, SIAM Journal on Algebraic Discrete Methods, 1985, doi: 10.1137/0606031

Class ArrayList<E>

java.lang.Object
  java.util.AbstractCollection<E>
    java.util.AbstractList<E>
      java.util.ArrayList<E>

Type Parameters:

E - the type of elements in this list

All Implemented Interfaces:

Serializable, Cloneable, Iterable<E>, Collection<E>, List<E>, RandomAccess, SequencedCollection<E>

Direct Known Subclasses:

AttributeList, RoleList, RoleUnresolvedList

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

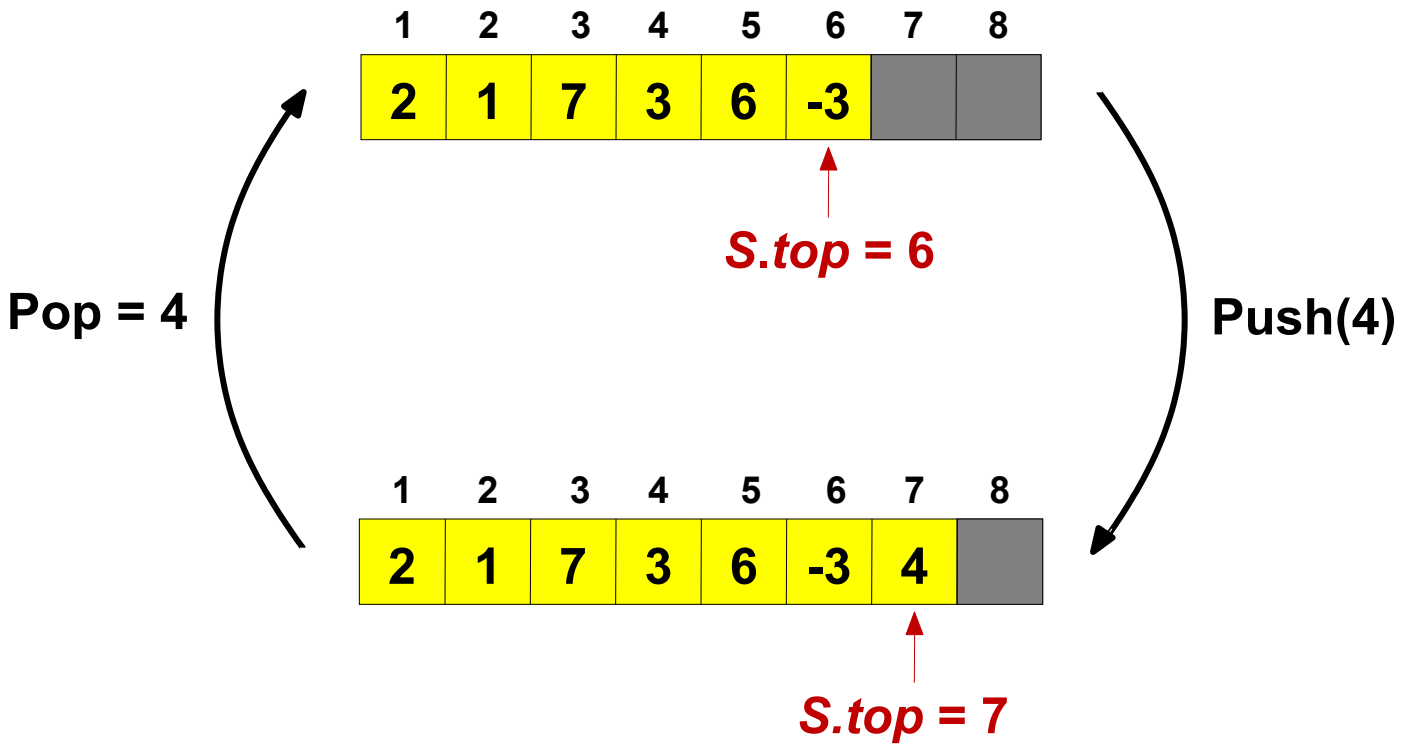
Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This class is roughly equivalent to `Vector`, except that it is unsynchronized.)

The `size`, `isEmpty`, `get`, `set`, `getFirst`, `getLast`, `removeLast`, `iterator`, `listIterator`, and `reversed` operations run in constant time. The `add`, and `addLast` operations runs in *amortized constant time*, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking). The constant factor is low compared to that for the `LinkedList` implementation.

to an empty list

Stack

Stack : array implementation



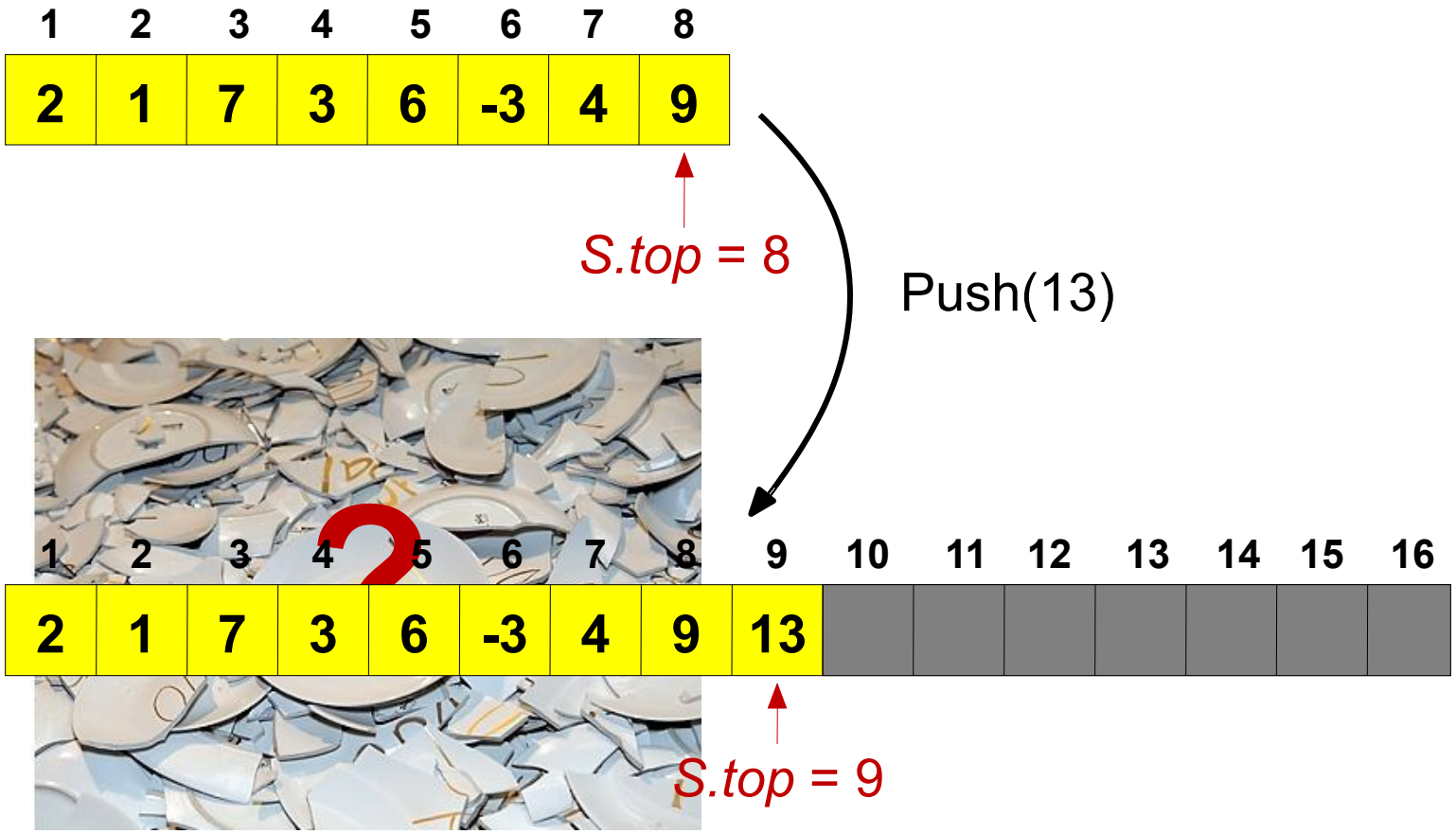

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

```
PUSH(S, x)
1  if S.top == S.size
2       error "overflow"
3  else S.top = S.top + 1
4       S[S.top] = x
```

```
POP(S)
1  if STACK-EMPTY(S)
2       error "underflow"
3  else S.top = S.top − 1
4       return S[S.top + 1]
```

Stack-Empty, Push, Pop : worst-case O(1) time

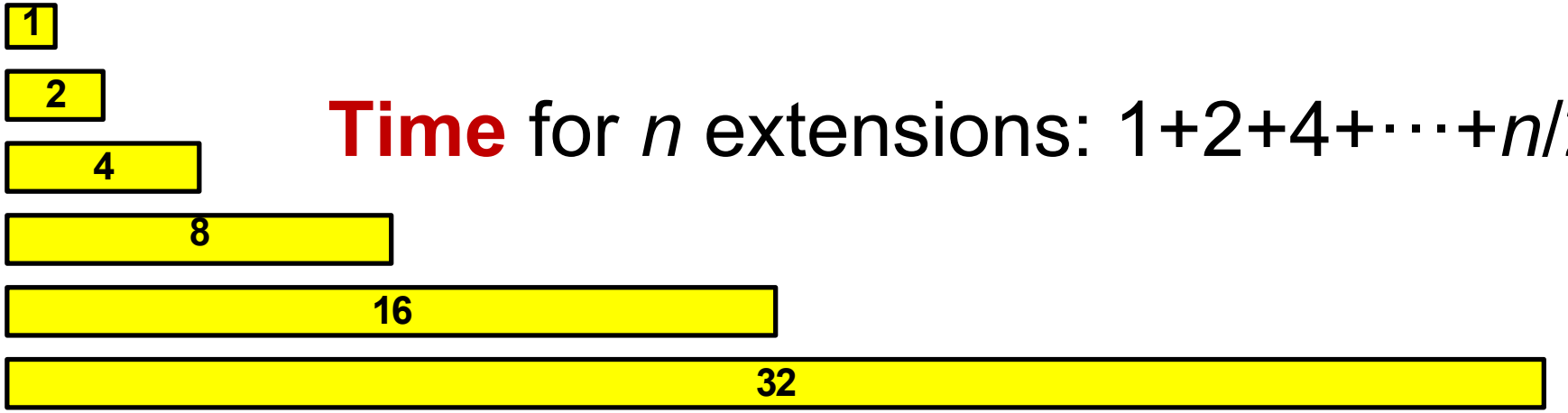
Stak : Overflow



Array doubling : O(*n*) time

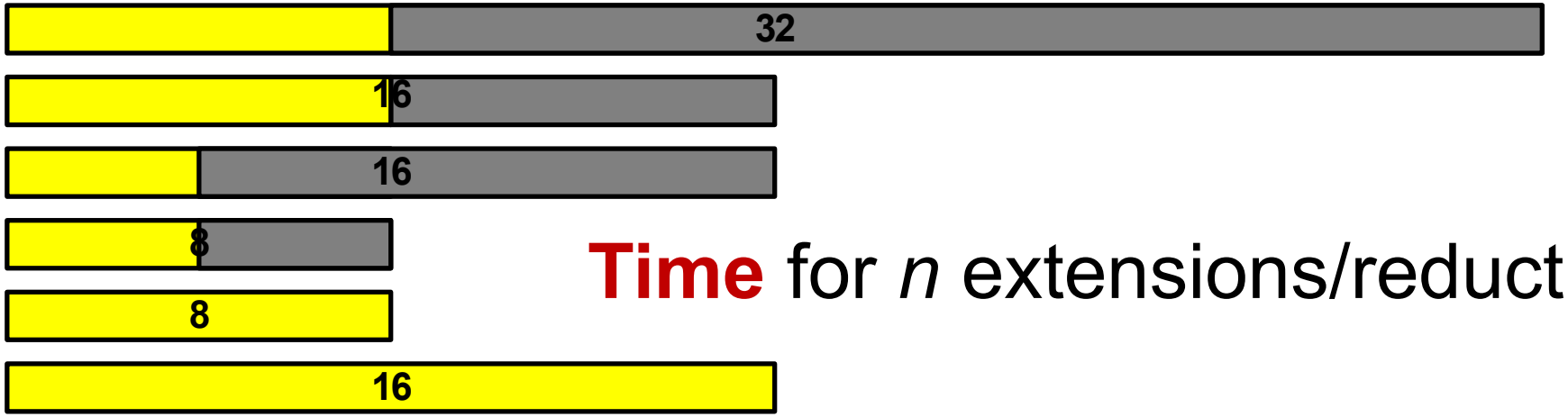
Array doubling

Double array when it is full



Time for n extensions: $1+2+4+\cdots+n/2+n = O(n)$

Halve the array when it is <1/4 full



Time for n extensions/reductions: $O(n)$

Array doubling + halving
– a general technique

Time for n extensions/reductions is $O(n)$

Space $\leq 4 \cdot$ current number of elements

Array implementation of a stack :
n push and pop operations take time $O(n)$

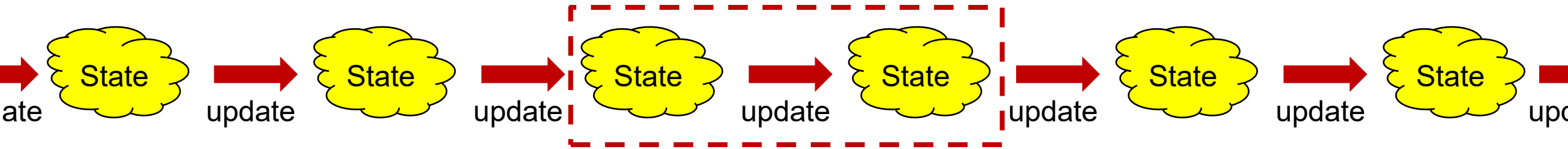
Wish list for analysis technique...

Requirements

- Analyze **worst-case** time for a **sequence** of operations
- Only analyze **single operations**

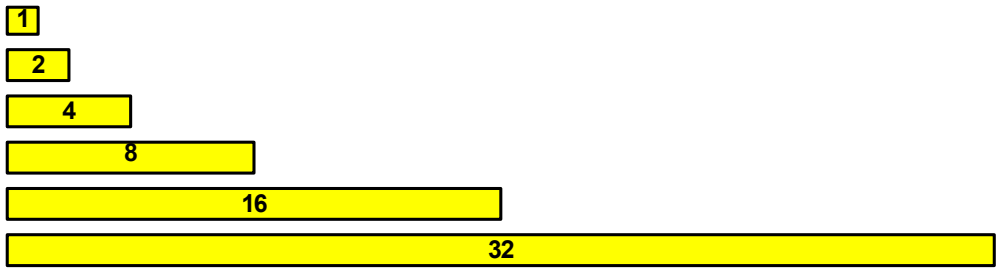
Advantages

- **Avoids** considering **interactions** between operations in the sequence
- Applies to all sequences with the given operations



Intuition

- There are **good**/**balanced** and **bad**/**unbalanced** states
- Going from a **bad** to a **good** state is **expensive**
- Requires many operations to get from a **good** to a **bad** state
- The (many) **cheap** operations *pay* slightly more to make later **expensive** operations nearly for **free**

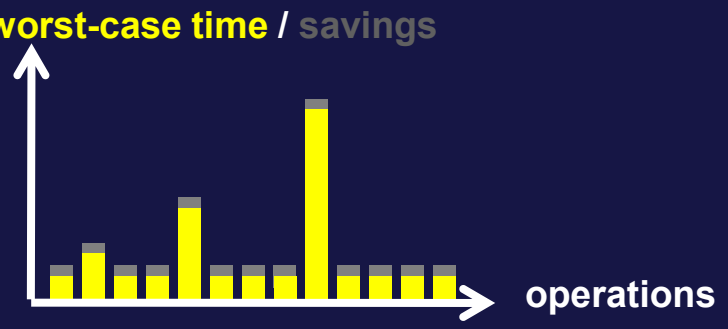


Amortized analysis

- 1 € can pay for O(1) work
- An operation taking O(t) time costs t €
- When to pay/saving up is not important – just need to have sufficient money when needed!
- (Bank) savings = potential = Φ
- Cannot borrow money, i.e., need to save money before spending money, $\Phi \geq 0$
- Amortized time for an operation = what we are willing to pay – but we need sufficient savings to be able to pay for the operation
- Use invariants to capture relationship between savings and state

Relationship between worst-case time and savings Φ

a)



b)

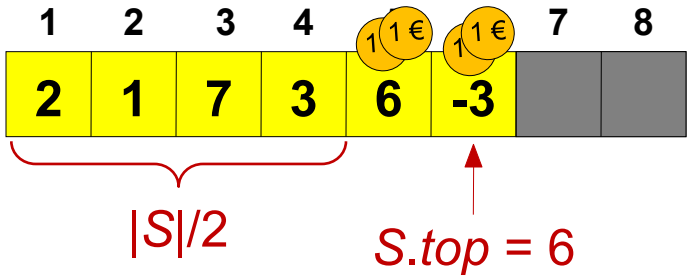
c)

d) Don't know

Example – stack

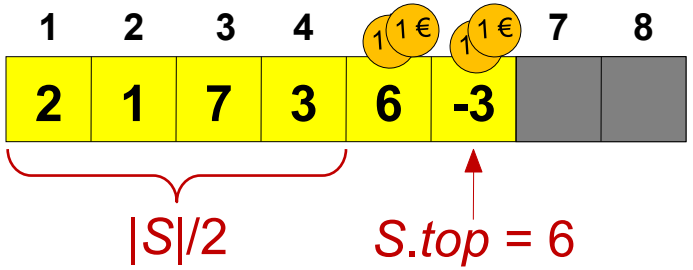
- A **good** stack is half full – no savings required
- Invariant : $\Phi = 2 \cdot |\, S.top - |S|/2 \,|$
- Assume: Cost 1 € per element inserted / copied
- Amortized time per push ≤ 3 € **?**

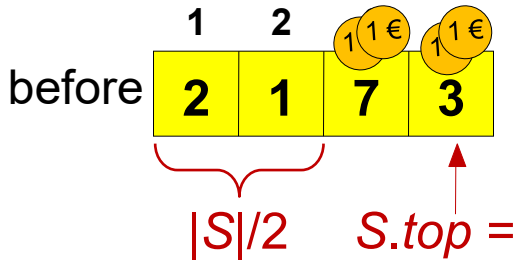
  (do we always have money to pay for the operation?)
- If yes: n push operations cost ≤ $3n$ €

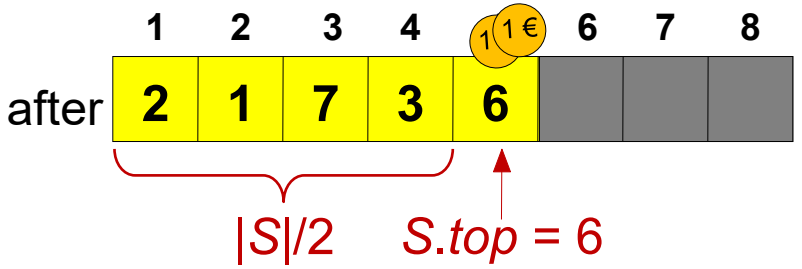


Example – stack
push = amortized 3€

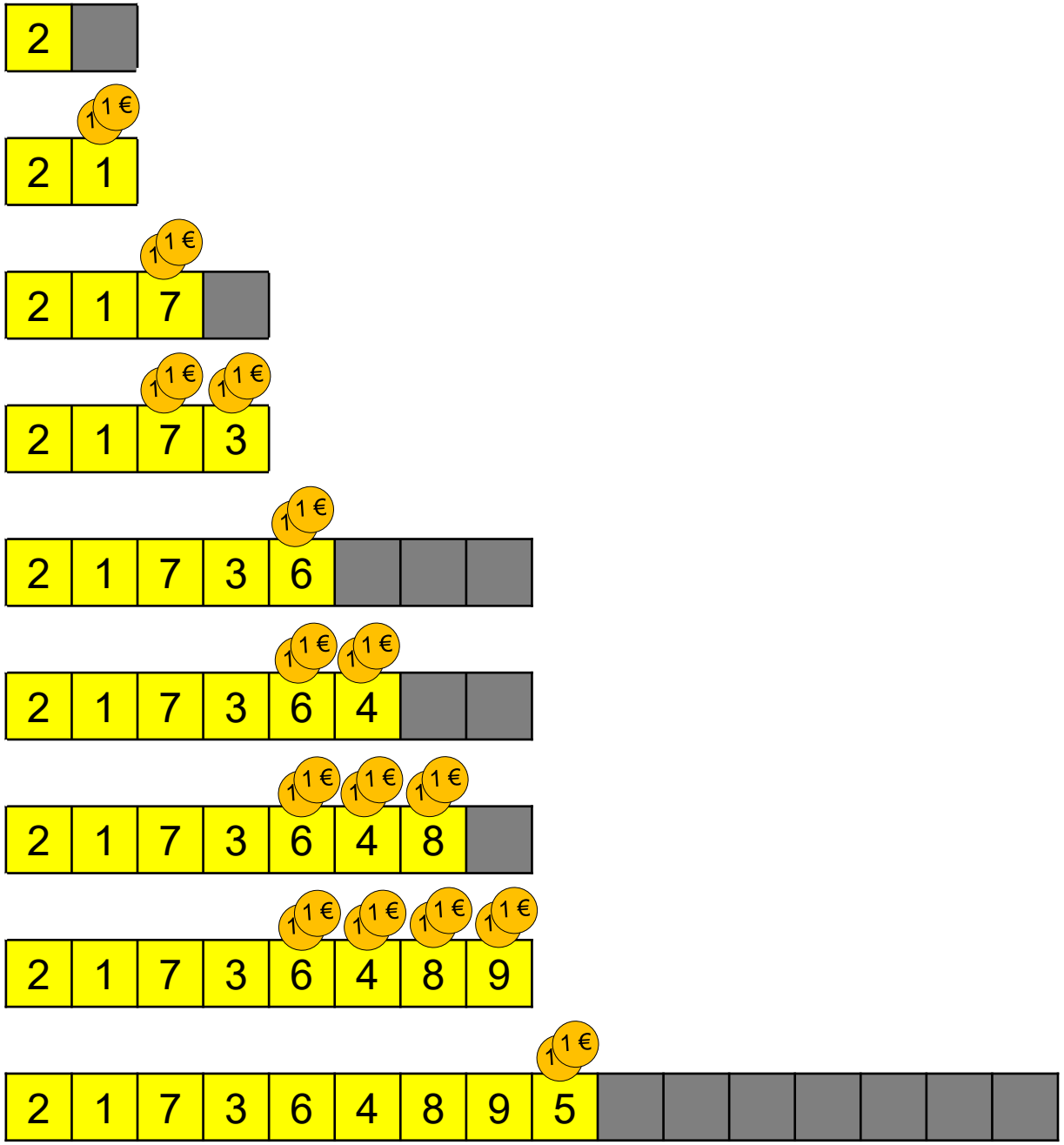
- **Push, no copying**
  - One new element: 1 €
  - Φ = ||*S*|/2 - top[*S*]| increases by at most 1, i.e., invariant satisfied if we save up 2 €
  - Amortized time: 1+2 = 3 €
- **Push, with copying**
  - Copy *S*: |*S*| €
  - Insert new element: 1 €
  - Φ before = |*S*|, Φ after = 2, i.e., |*S*|-2 € released
  - Amortized time: |*S*|+1-(|S|-2) = 3 €





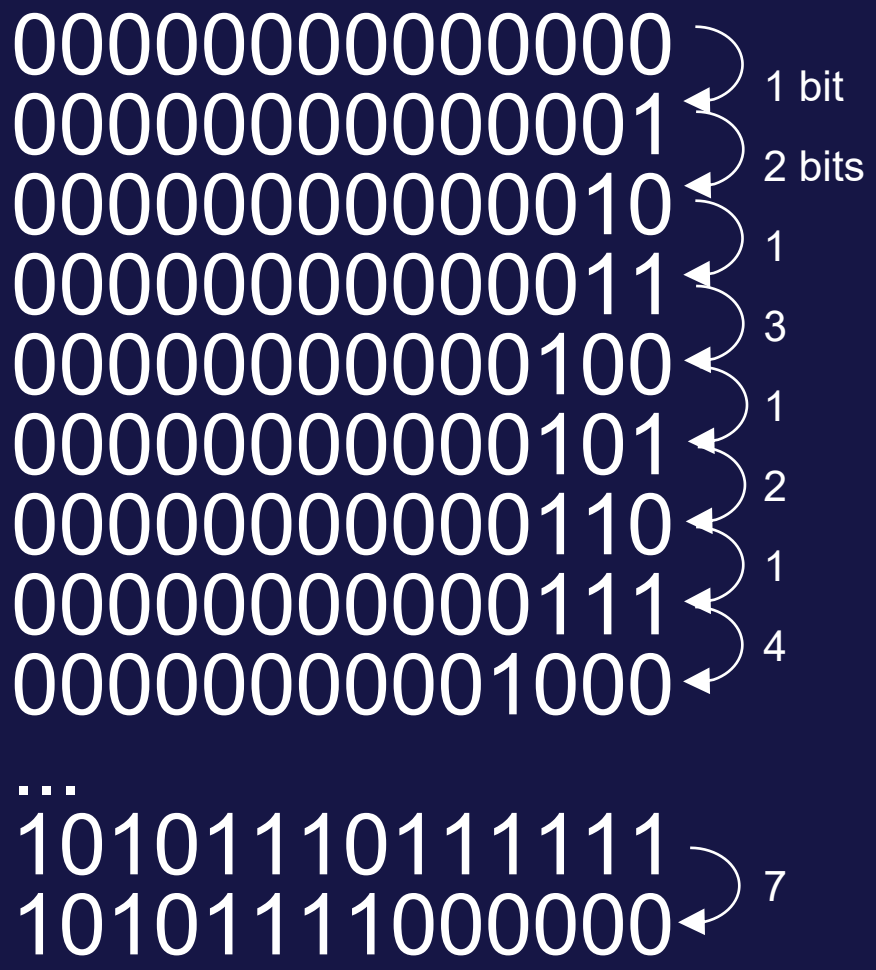


Invariant : Φ = 2·| *S*.top - |*S*|/2 |



	potential	potential increase	real time	amortized time
	0			
push(1)		2	1	2+1 = 3
	2			
push(7)		0	2+1	0+2+1 = 3
	2			
push(4)		2	1	2+1 = 3
	4			
push(6)		-2	4+1	-2+4+1 = 3
	2			
push(4)		2	1	2+1 = 3
	4			
push(8)		2	1	2+1 = 3
	6			
push(9)		2	1	2+1 = 3
	8			
push(5)		-6	8+1	-6+8+1 = 3
	2			

Binary counter



What is a good savings- / potential- / Φ-function ?

a) Position of most significant 1-bit

b) Position of the rightmost 0

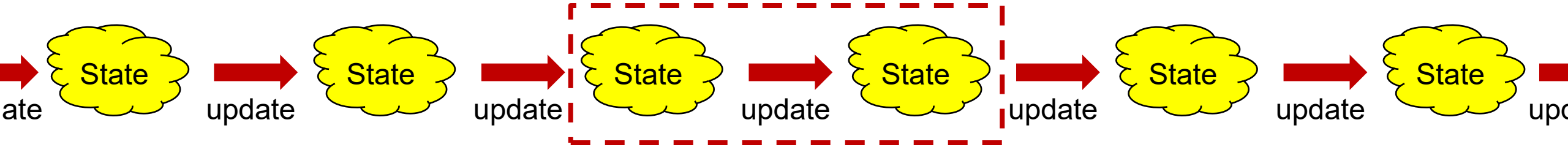
c) # 1's in the binary number

d) # 0's in the binary number

e) Don't know

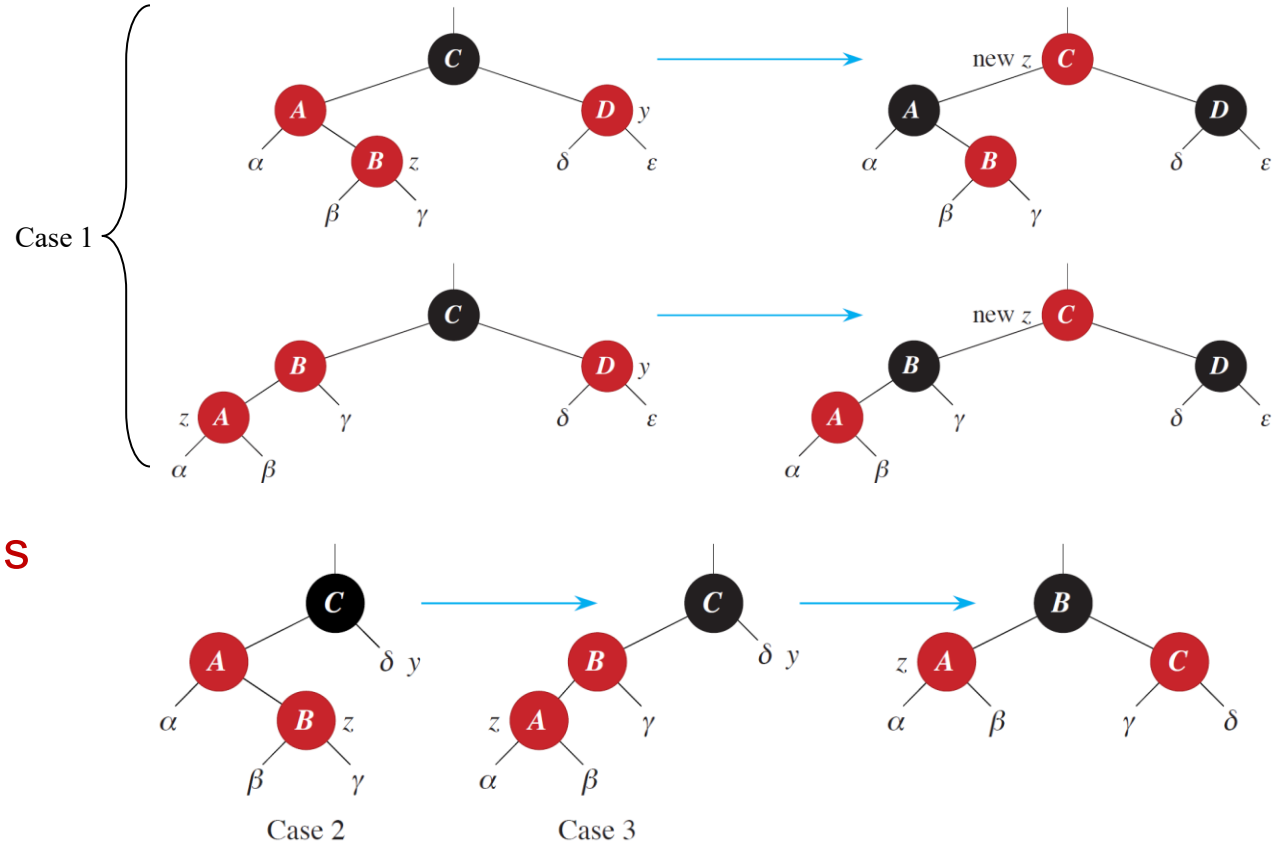
Amortized analysis

- Analysis technique to argue about the worst-case time for a sequence of operations
- Only need to analyze the operations individually
- **The Art** : Find a useful invariant for Φ



Example – red-black trees

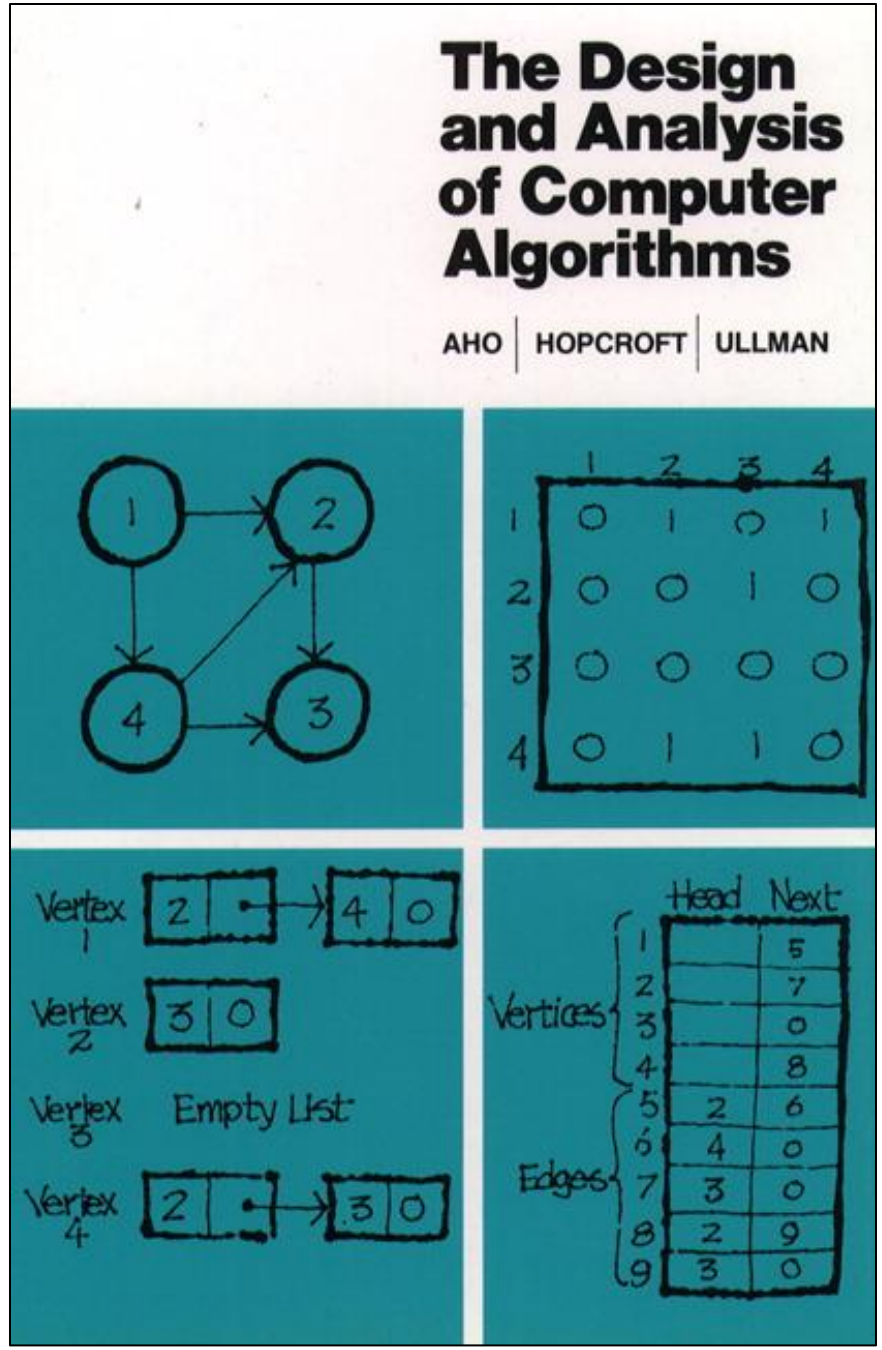
Insert(x)
=
Search ← O(log n)
+
Create new red leaf ← O(1)
+
Rebalance ← # transitions



transitions = *amortized* O(1)

Φ = # red nodes

Corollary: Insertion in a red-black tree takes *amortized* O(1) time, *if* insertion position is known


The Design and Analysis of Computer Algorithms
AHO
HOPCROFT
ULLMAN
1
2
4
3
1 2 3 4
1 0 1 0 1
2 0 0 1 0
3 0 0 0 0
4 0 1 1 0
Vertex 1
2
4
0
Vertex 2
3
0
Vertex 3
Empty List
Vertex 4
2
3
0
Head
Next
Vertices
Edges
1 5
2 7
3 0
4 8
5 2 6
6 4 0
7 3 0
8 2 9
9 3 0

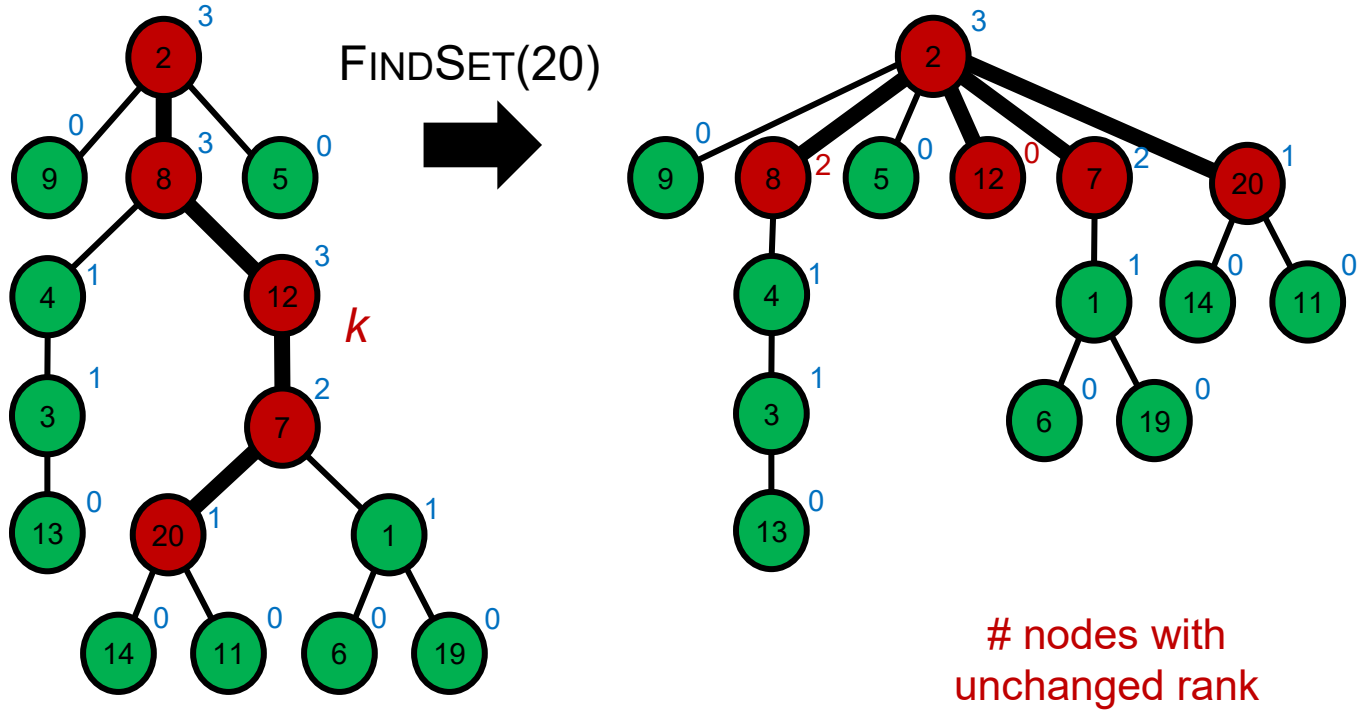
1974

Exercise **4.17 in Aho Hopcroft Ullman "The Design and Analysis of Computer Algorithms" (1974)

not curriculum

Union-Find with path compression (without rank)

```
MAKESET(x)
1   x.p = x

UNION(x, y)
1   LINK(FINDSET(x), FINDSET(y))

LINK(x, y)
1   y.p = x

FINDSET(x)
1   if x ≠ x.p do
2       x.p = FINDSET(x.p)
3   return x.p
```

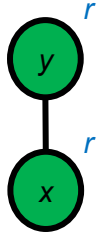


Amortized analysis

Definition rank(x) = $\lfloor \log_2 |T(x)| \rfloor$

Lemma rank(x.p) ≥ rank(x)

Potential $\Phi = \sum_x \text{rank}(x)$



Observation

Assume *x* and *y* rang *r*, i.e., both subtrees have size $[2^r, 2^{r+1}[$

If *x* is removed, the *y* gets rank < *r*

nodes with unchanged rank

	Real time	ΔΦ	Amortized time (real time + ΔΦ)
MAKESET	O(1)	0	O(1)
LINK	O(1)	≤ log *n*	O(log n)
FINDSET	*k*	≤ - *k* + 2 + log *n*	O(log n)
UNION = 2 x FINDSET + LINK			O(log n)

not curriculum

Union-Find with path compression and linking-by-rank

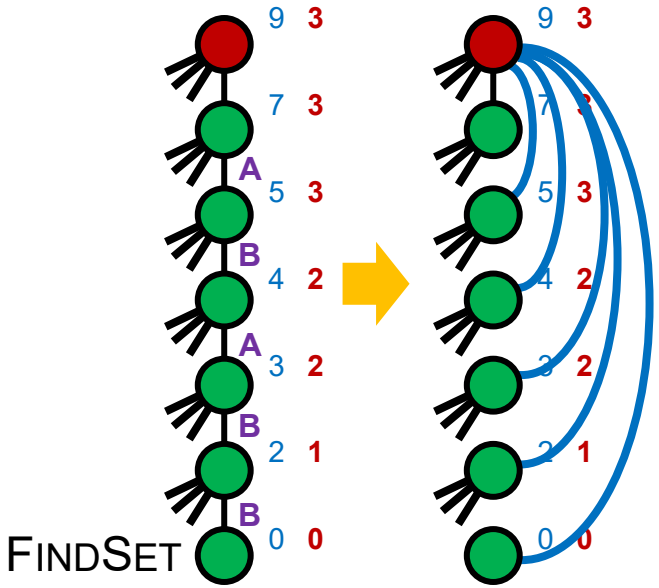
Theorem n MAKESET and m FINDSET and UNION operations take time $O((n + m) \cdot \log^* n)$

Definition $\log^* n$ = smallest i where $n \le r_i = 2^{2^{\cdot^{\cdot^{\cdot^{2}}}}}$ } tower of height i

$r_0 = 1$ $\quad$ $r_3 = 2^4 = 16$
$r_1 = 2^1 = 2$ $\quad$ $r_4 = 2^{16} = 65536$
$r_2 = 2^2 = 4$ $\quad$ $r_5 = 2^{65536}$

Proof

- Only the rank of a root can change (increase during link)
- A non-root x never becomes a root again and rank($x.p$) > rank(x)
- A root with rank r has subtree size ≥ 2^r, all non-roots rank < r
- **Lemma** # nodes with rank r ≤ $n / 2^r$
- Parent change increases rank difference to parent
- Define layer of a rank r as $\lambda(r) = \log^* r$
- Total time $O(n + m +$ # parent changes$)$



$$O(n + m + \underbrace{m \cdot \log^* n}_{\text{(B) parent is already in different layer}} + \underbrace{\sum_{\text{layer } i} \sum_{r=r_{i-1}+1..r_i} n / 2^r \cdot r_i}_{\text{(A) parent is in same layer before parent change}}) = O(n + m \cdot \log^* n + \sum_{\text{layer } i} n / 2^{r_{i-1}} \cdot r_i) = O((n + m) \cdot \log^* n)$$

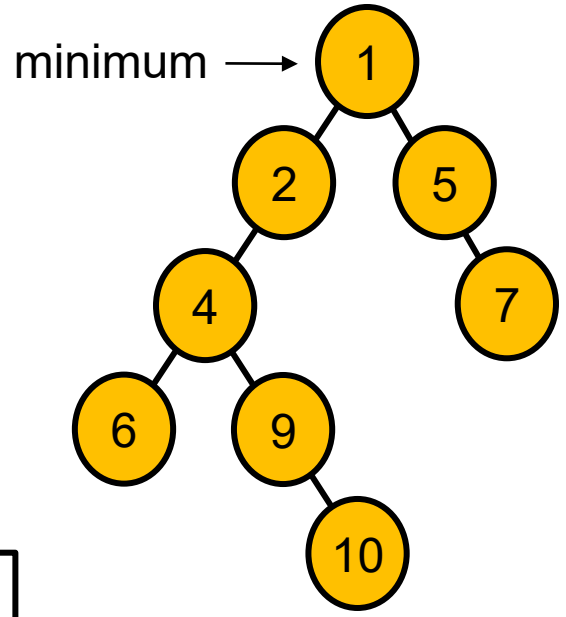
r_i: # times a node in layer i can have a parent in layer i

$2^{r_{i-1}} = 2^{r_{i-1}}$ (assumes $r_{-1} = 0$)

not curriculum

Skew heaps – ”self-balancing” heaps

- Priority queue
  - `FindMin`, `Insert`, `DeleteMin`, `Meld`
- Heap-ordered binary tree
  - *No requirements to structure or depth*
- Node: `left`, `right`, `element`
- Priority queue = pointer to root




```
proc FindMin(H)
    return H.element

proc Insert(H, e)
    v = new Node(e, Null, Null)
    return Meld(H, v)

proc DeleteMin(H)
    return Meld(H.left, H.right)
```

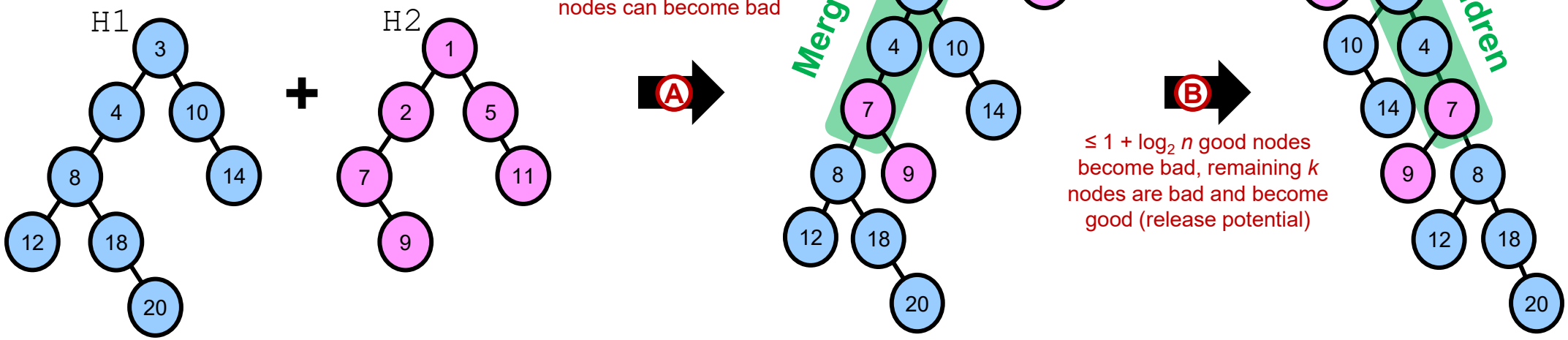
`Meld` next slide

Daniel Dominic Sleator and Robert Endre Tarjan, *Self-Adjusting Heaps*, SIAM Journal on Computing, 1986, doi: 10.1137/0215004

Skew heaps

not curriculum

Meld




```
proc Meld(H1, H2)
    if H1 = Null then return H2
    if H2 = Null then return H1
    if H1.element < H2.element then
        return new Node(H1.element, H1.right, Meld(H1.left, H2))
    else
        return new Node(H2.element, H2.right, Meld(H2.left, H1))
```

Amortized analysis

Definition Good node `v` : |`v.left`| ≤ |`v`| / 2
Lemma # good nodes on left path ≤ 1 + $\log_2 n$
Potential Φ = # bad nodes

`Meld` **amortized time**

ΔΦ from A + ΔΦ from B + real time

≤ $2 + 2\cdot\log_2 n + 1 + \log_2 n - k + k + 1 + \log_2 n = O(\log n)$

⇒ All operations amortized time $O(\log n)$

not curriculum

Self-balancing data structures

with amortized running times

	Skew heaps	Fibonacci heaps	Splay trees
Minimum	O(1)	O(1)	$O_A(1)^*$
Insert	$O_A(\log n)$	$O_A(1)$	$O_A(\log n)$
DeleteMin	$O_A(\log n)$	$O_A(\log n)$	$O_A(1)^*$
Delete		$O_A(\log n)$	$O_A(\log n)$
Meld	$O_A(\log n)$	$O_A(1)$	
DecreaseKey		$O_A(1)$	$O_A(\log n)$
Search			$O_A(\log n)$

Skew Heaps Sleator and Tarjan, SICOMP 1986

Fibonacci heaps Fredman and Tarjan, JACM 1987 [CLRS, 3rd edition, Chapter 19]

Splay trees Sleator and Tarjan, JACM 1985 + Cole*, SICOMP 2006

$O_A \equiv$ amortized time

Algorithms and Data Structures

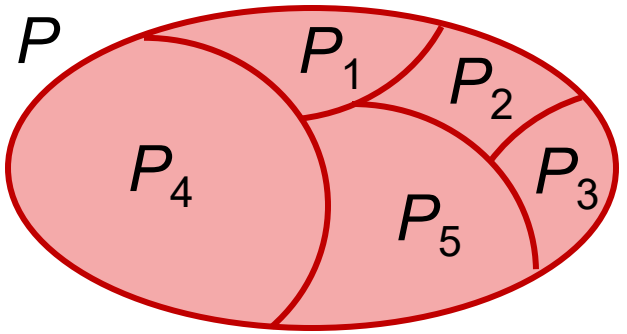
Divide and conquer
[CLRS, Chapter 2.3, 4.1-4.5, Problem 30.1.c]

Divide and conquer

Algorithm design technique

Applicable to some problems (but far from all)

- **Partition** a problem P into smaller problems $P_1,..,P_k$ of the same type which can be solved independently
- **Recursively** solve subproblems $P_1,..,P_k$ (small problems are solved directly)
- **Combine** solutions for $P_1,..,P_k$ to a solution for P



Example – Merge-Sort

MERGE-SORT(A, p, r)

Small subproblem (size ≤ 1) → **if** $p \geq r$

return

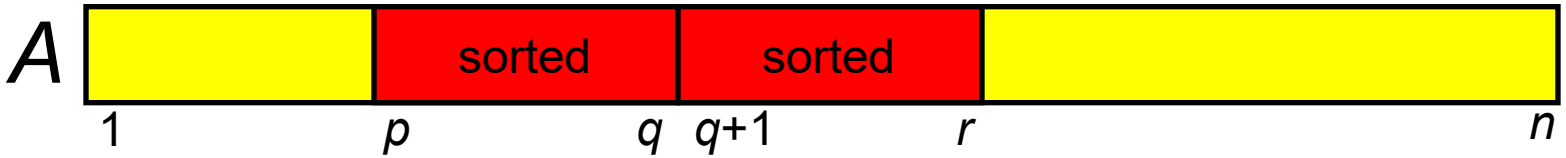
$q = \lfloor (p + r)/2 \rfloor$

1 Two smaller subproblems

2 Solve recursively → MERGE-SORT(A, p, q)

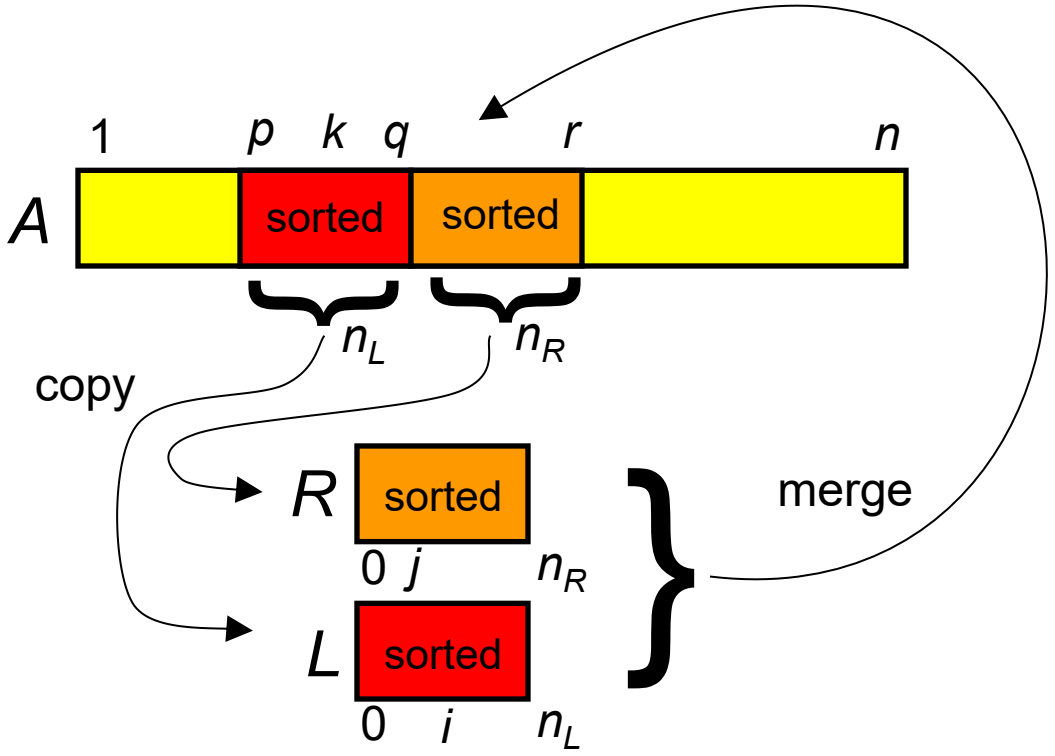
→ MERGE-SORT$(A, q + 1, r)$

3 Combine → MERGE(A, p, q, r)



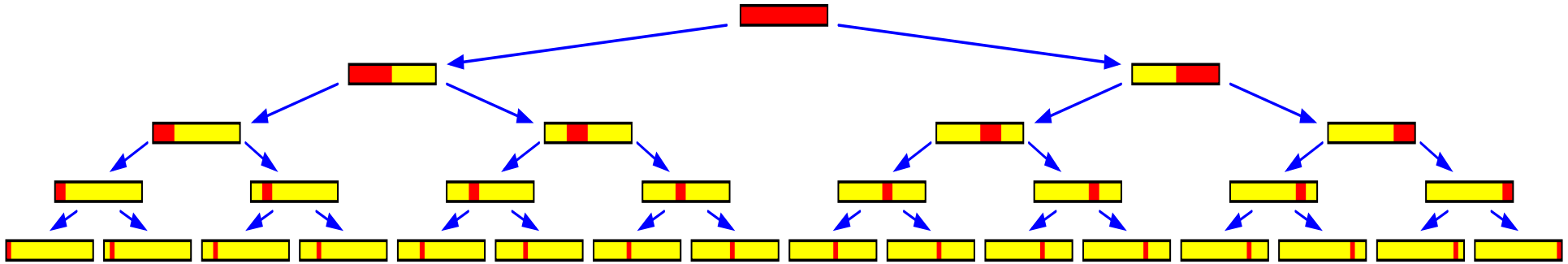
```
MERGE(A, p, q, r)
1  n_L = q − p + 1          // length of A[p : q]
2  n_R = r − q              // length of A[q + 1 : r]
3  let L[0 : n_L − 1] and R[0 : n_R − 1] be new arrays
4  for i = 0 to n_L − 1     // copy A[p : q] into L[0 : n_L − 1]
5      L[i] = A[p + i]
6  for j = 0 to n_R − 1     // copy A[q + 1 : r] into R[0 : n_R − 1]
7      R[j] = A[q + j + 1]
8  i = 0                    // i indexes the smallest remaining element in L
9  j = 0                    // j indexes the smallest remaining element in R
10 k = p                    // k indexes the location in A to fill
11 // As long as each of the arrays L and R contains an unmerged element,
   //     copy the smallest unmerged element back into A[p : r].
12 while i < n_L and j < n_R
13     if L[i] ≤ R[j]
14         A[k] = L[i]
15         i = i + 1
16     else A[k] = R[j]
17         j = j + 1
18     k = k + 1
19 // Having gone through one of L and R entirely, copy the
   //     remainder of the other to the end of A[p : r].
20 while i < n_L
21     A[k] = L[i]
22     i = i + 1
23     k = k + 1
24 while j < n_R
25     A[k] = R[j]
26     j = j + 1
27     k = k + 1
```

③ Combine



Merge-Sort – analysis

Recursion tree



Observation : Total work per level in the tree is $O(n)$

Total work : $O(n \cdot \#\text{ levels}) = O(n \cdot \log_2 n)$

Divide and conquer – examples

- **MergeSort**
  - Partition into two equally sized parts
  - Recursively sort
  - Combine = merge sorted lists
- **QuickSort**
  - Partition into two parts around random pivot (random partition)
  - Recursively sort
  - Combine = none (concatenation of left and right solutions)
- **QuickSelect**
  - Partition into two parts around random pivot (random partition)
  - One recursive call to select
  - Combine = none (return result from recursion)

Analysis of divide and conquer

= analysis of a recursive procedure

Essentially two approaches

- Argue directly about **recursion tree**
  (depth, #node at each level, work at the nodes/levels/tree)
- Solve a mathematical **recurrence relation**, e.g., by induction

$$T(n) = a \qquad \text{if} \quad n \le c$$
$$T(n) = 2 \cdot T(n / 2) + a \cdot n \qquad \text{otherwise}$$

What is the recurrence relation for MergeSort when sorting recursively 3 parts of size $n/3$?

For $n \geq 3$ …

a) $T(n) = 3 \cdot T(n) + a \cdot n^3$

b) $T(n) = 3 \cdot T(n) + a \cdot n$

c) $T(n) = 3 \cdot T(n/3) + a \cdot n$

d) $T(n) = 3 \cdot T(n/2) + a \cdot n$

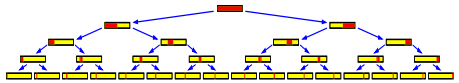
e) Don't know

… and $T(n) = c$ for $n < 3$

Solving a recurrence relation

$$T(n) = a \quad \text{if} \quad n \le c$$
$$T(n) = 2 \cdot T(n / 2) + a \cdot n \quad \text{otherwise}$$

- Unfold recurrence relation and argue about the **recursion tree**



- Guess a solution and prove by **induction** by increasing n

Recurrence relations – pitfalls

- Odd partitions ignored (n odd, recurrence relation calls typically $\lfloor n/2 \rfloor$ and $\lceil n/2 \rceil$) [CLRS, Chapter 4.7]

- Analysis typically only for $n = 2^k$

- Never use O-notation in recurrence relations – use constants (~~$T(n) = O(n) + O(T(n/3))$~~)

$$T(n) = c \cdot n + a \cdot T(n / 3)$$

Master theorem

simplification of [CLRS, Theorem 4.1]

Theorem

If a, b, c, d and p are constants, where a is an integer and b, c, d and p are real values, $a \geq 1$, $b > 1$, $c > 0$, $d \geq 1$ and $p \geq 0$, then the recurrence

$$T(n) = \begin{cases} c & \text{if } n \leq d \\ a \cdot T(n/b) + c \cdot n^p & \text{if } n > d \end{cases}$$

has the solutions

$$T(n) = \begin{cases} \Theta(n^p) & \text{if } a < b^p \\ \Theta(n^p \cdot \log n) & \text{if } a = b^p \\ \Theta(n^{\log_b a}) & \text{if } a > b^p \ . \end{cases}$$

Note that $n^{\log_b a} = a^{\log_b n}$.

Solution to the recurrence relation ?

$T(n) = 4 \cdot T(n / 2) + n^3$ for $n > 1$

$T(n) = c$ for $n = 1$

a) $T(n) = \Theta(n^3)$

b) $T(n) = \Theta(n^3 \log n)$

c) $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$

d) Don't know

Solution to the recurrence relation ?

$$T(n) = 4 \cdot T(n / 2) + n \quad \text{for } n > 1$$
$$T(n) = c \quad \text{for } n = 1$$

a) $T(n) = \Theta(n)$

b) $T(n) = \Theta(n \log n)$

c) $T(n) = \Theta(n^{\log_2 4}) = \Theta(n^2)$

d) Don't know

Solution to the recurrence relation ?

$T(n) = 5 \cdot T(n / 4) + n^2$ for $n > 1$

$T(n) = c$ for $n = 1$

a) $T(n) = \Theta(n^2)$

b) $T(n) = \Theta(n^2 \log n)$

c) $T(n) = \Theta(n^{\log_4 5}) = \Theta(n^{1.161})$

d) Don't know

Solution to the recurrence relation ?

$$T(n) = 9 \cdot T(n / 3) + n^2 \quad \text{for} \quad n > 1$$
$$T(n) = c \quad \text{for} \quad n = 1$$

a) $T(n) = \Theta(n^2)$

b) $T(n) = \Theta(n^2 \log n)$

c) $T(n) = \Theta(n^{\log_3 9}) = \Theta(n^2)$

d) Don't know

Solution to the recurrence relation ?

$T(n) = 7 \cdot T(n/2) + c \cdot n^2$ for $n \geq 2$

$T(n) = c$ for $n = 1$

a) $T(n) = \Theta(n^2)$

b) $T(n) = \Theta(n^2 \cdot \log n)$

c) $T(n) = \Theta(n^{7/2})$

d) $T(n) = \Theta(n^{\log_2 7}) = \Theta(n^{2.81})$

e) $T(n) = \Theta(n^7)$

f) Don't know

Depth of recursion ?

$$T(n) = \begin{cases} c & \text{if } n \leq d \\ a \cdot T(n/b) + c \cdot n^p & \text{if } n > d \end{cases}$$

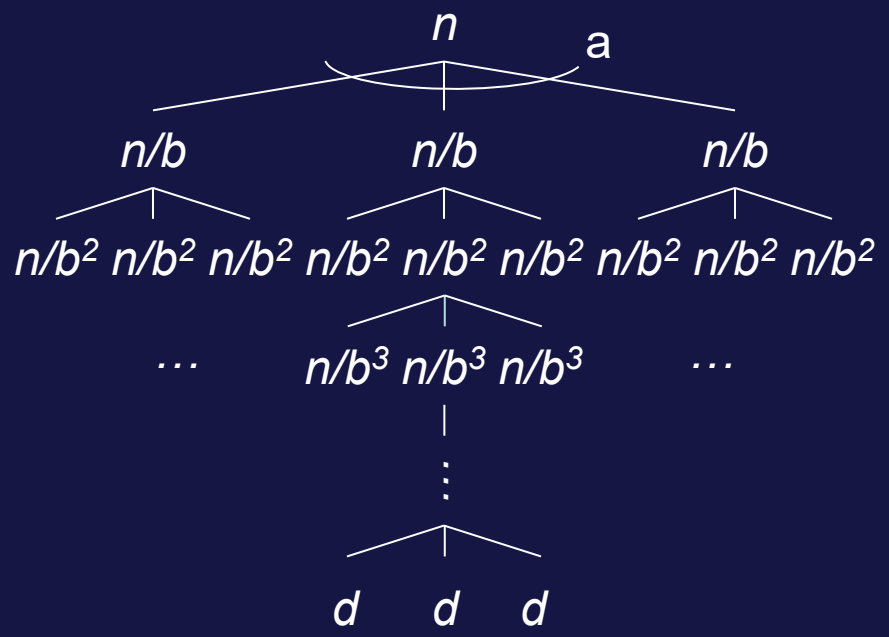
a) log *n*

b) $\log_b n$

c) $\log_b (n/d)$

d) $\log_d (n/b)$

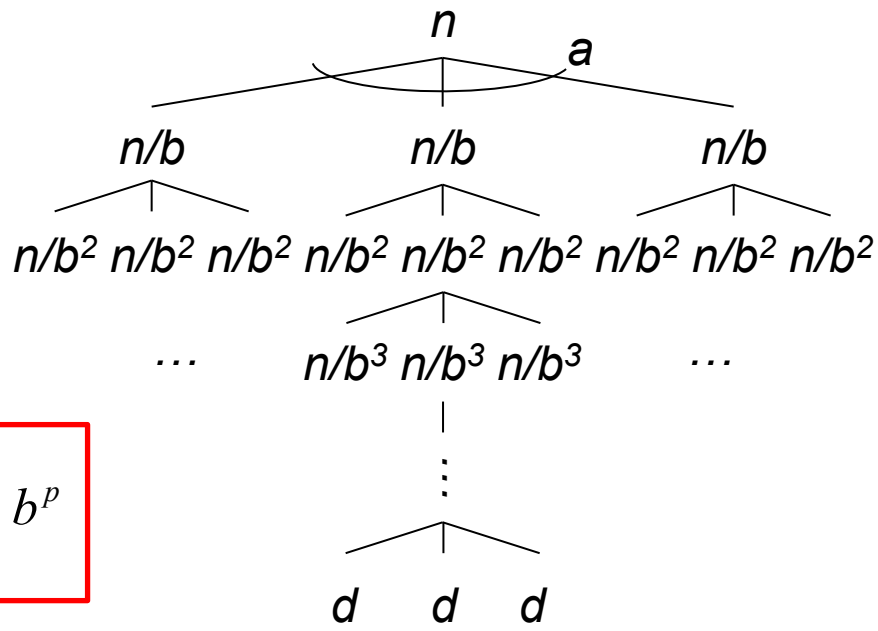
e) Don't know



$n / b^k \leq d$?

Depth	$i = 0..\log_b (n/d) - 1$	$\log_b (n/d)$
# subproblems	a^i	$a^{\log_b (n/d)}$
Size of subproblems	n/b^i	d
Contribution per subproblem	$c \cdot (n/b^i)^p$	c
Contribution per level	$a^i \cdot c \cdot (n/b^i)^p$	$c \cdot a^{\log_b (n/d)}$

$$T(n) = \begin{cases} c & \text{if } n \le d \\ a \cdot T(n/b) + c \cdot n^p & \text{if } n > d \end{cases}$$



$$T(n) = \Theta\left(\underbrace{c \cdot a^{\log_b (n/d)}}_{\text{(bottom of recursion)}} + \underbrace{\sum_{i=0}^{\log_b (n/d)-1} a^i \cdot c \cdot (n/b^i)^p}_{\text{(level } i = 0..\log_b (n/d) - 1\text{)}} \right)$$

$$= \Theta\left(c \cdot a^{\log_b n} + c \cdot n^p \cdot \sum_{i=0}^{\log_b n - 1} (a/b^p)^i \right)$$

$$\frac{(a/b^p)^{\log_b n} - 1}{a/b^p - 1} \quad \text{for} \quad a \neq b^p$$

$$= \Theta\left(n^{\log_b a} + n^p \cdot \begin{cases} 1 & \text{for } a < b^p \\ \log n & \text{for } a = b^p \\ (a/b^p)^{\log_b n} & \text{for } a > b^p \end{cases} \right) = \Theta\left(\begin{cases} n^p & \text{for } a < b^p \\ n^p \cdot \log n & \text{for } a = b^p \\ n^{\log_b a} & \text{for } a > b^p \end{cases} \right)$$

$$a^{\log_b n} = n^{\log_b a}$$

$$(b^p)^{\log_b n} = n^p$$

Multiplication of long integers

$$2387 * 3351$$

$$= (23 * 10^2 + 87) * (33 * 10^2 + 51)$$

$$= 23 * 33 * 10^4 + (23 * 51 + 87 * 33) * 10^2 + 87 * 51$$

$$= 759 * 10^4 + (1173 + 2871) * 10^2 + 4437$$

$$= 7590000$$

$$+ \ \ 117300$$

$$+ \ \ 287100$$

$$+ \ \ \ \ 4437$$

$$= \underline{\underline{7998837}}$$

Multiplication integers with *n* bits

$$I \cdot J = I_h \cdot J_h \cdot 2^n + (I_h \cdot J_l + I_l \cdot J_h) \cdot 2^{n/2} + I_l \cdot J_l$$

a) $T(n) = 2 \cdot T(n/2) + a \cdot n^2$

b) $T(n) = 4 \cdot T(n/4) + a \cdot n$

c) $T(n) = 4 \cdot T(n/2) + a \cdot n$

d) $T(n) = 4 \cdot T(n/2) + a \cdot n^2$

e) Don't know

… and $T(n) = c$ for $n = 1$ $\Longrightarrow$ $T(n) = O(n^2)$

Multiplication of long integers

[CLRS, Problem 30.1.c]

- I and J both n-bit integers
- Naïve implementation $O(n^2)$ bit operations
- $I \cdot J = I_h \cdot J_h \cdot 2^n + ((I_h - I_l) \cdot (J_l - J_h) + I_l \cdot J_l + I_h \cdot J_h) \cdot 2^{n/2} + I_l \cdot J_l$

$$T(n) = 3 \cdot T(n/2) + c \cdot n \quad \text{for } n \geq 2$$

$$T(n) = c \quad \text{for } n = 1$$

- $T(n) = O(n^{\log_2 3}) = O(n^{1.58})$

Anatolii A. Karatsuba 1960 (wikipedia.org/wiki/Karatsuba_algorithm)

Multiplication of long integers

+4000 years	$O(n^2)$
Divide and conquer Karatsuba 1960	$O(n^{\log_2 3})$
Schönhage-Strassen 1971	$O(n \cdot \log n \cdot \log\log n)$
Fürer 2007 Harvey, Hoeven 2018 Harvey, Hoeven 2019	$O(n \cdot \log n \cdot 2^{O(\log^* n)})$ $O(n \cdot \log n \cdot 2^{2 \cdot \log^* n})$ $O(n \cdot \log n)$

A paper on the history of multiplication,
Anatolii A. Karatsuba, *The Complexity of Computations*, Proceedings of the Steklov Institute of Mathematics, 1995 (Russian original)
David Harvey, Joris van der Hoeven, *Integer multiplication in time* O(*n* log *n*), Annals of Mathematics, 2021, doi: 10.4007/annals.2021.193.2.4

Matrices

$m = 3$ columns

represents linear transformations $\mathbb{R}^3 \longrightarrow \mathbb{R}^4$

$n = 4$ rows

$$\begin{pmatrix} 3 & 2 & -1 \\ 5 & 0 & 4 \\ 7 & -2 & 0 \\ 1 & 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3x_1 + 2x_2 - x_3 \\ 5x_1 + 4x_3 \\ 7x_1 - 2x_2 \\ x_1 + 3x_2 + 2x_3 \end{pmatrix}$$

$n \times m$ matrix

column vector
$m \times 1$ matrix

column vector
$n \times 1$ matrix

Rules

Matrix addition
($n \times m$ matrices)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 4 & 1 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 7 & 5 \\ 8 & 9 \end{pmatrix}$$

$(A + B) + C = A + (B + C)$

$A + B = B + A$

Matrix subtraction
($n \times m$ matrices)

$$\begin{pmatrix} 6 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} - \begin{pmatrix} 2 & 6 \\ 4 & 1 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 4 & -4 \\ -1 & 3 \\ 2 & 3 \end{pmatrix}$$

$A - (B + C) = (A - B) - C$

Multiplication by a constant

$$3\begin{pmatrix} 1 & 2 \\ -2 & 3 \\ 4 & -1 \end{pmatrix} = \begin{pmatrix} 3 & 6 \\ -6 & 9 \\ 12 & -3 \end{pmatrix}$$

$c(dA) = (cd)A$

$c(A + B) = (cA) + (cB)$

Matrix multiplication

= composition of linear transformations

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & c_{22} & \cdots & c_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p1} & c_{p2} & \cdots & c_{pr} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{q1} & b_{q2} & \cdots & b_{qr} \end{pmatrix}$$

$p \times r$ matrix — $p \times q$ matrix — $q \times r$ matrix

$=$

$$c_{ij} = \sum_{k=1..m} a_{ik} \cdot b_{kj}$$

Rules

$$(AB)C = A(BC)$$
$$A(B+C) = (AB)+(AC)$$
$$(A+B)C = (AC)+(BC)$$

$$AB \neq BA$$

$$\begin{pmatrix} 2 & 1 & 4 \\ 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 3 & 1 \\ 2 & 2 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 12 & 20 \\ 11 & 15 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 1 \\ 2 & 2 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} 2 & 1 & 4 \\ 1 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 7 & 6 & 14 \\ 6 & 8 & 12 \\ 6 & 13 & 12 \end{pmatrix}$$

Matrix Multiplication

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & c_{22} & \cdots & c_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p1} & c_{p2} & \cdots & c_{pr} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ a_{21} & a_{22} & \cdots & a_{2q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1r} \\ b_{21} & b_{22} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{q1} & b_{q2} & \cdots & b_{qr} \end{pmatrix}$$

$p \times r$ matrix — $p \times q$ matrix — $q \times r$ matrix

$=$

RECTANGULAR-MATRIX-MULTIPLY(A, B, C, p, q, r)

```
1  for i = 1 to p
2      for j = 1 to r
3          for k = 1 to q
4              c_ij = c_ij + a_ik · b_kj
```

Naïve implementation $O(pqr)$ time

Quadratic matrix multiplication

[CLRS, Chapter 4.1-4.2]

$\frac{n}{2} \times \frac{n}{2}$ matrix

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$n \times n$ matrix $\quad$ $n \times n$ matrix $\quad$ $n \times n$ matrix

$$\begin{aligned} I &= A \cdot E + B \cdot G \\ J &= A \cdot F + B \cdot H \\ K &= C \cdot E + D \cdot G \\ L &= C \cdot F + D \cdot H \end{aligned}$$

- A, B, ..., K, L are $n/2$ x $n/2$-matrices
- I, J, K, L can be computed using 8 recurrence relation multiplikations and 4 matrix additions on $n/2$ x $n/2$ -matricer
- $T(n) = 8 \cdot T(n/2) + c \cdot n^2$ for $n \geq 2$

  $T(n) = c$ for $n = 1$
- $T(n) = O(n^{\log_2 8}) = O(n^3)$

Strassen's matrix multiplication

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$\begin{aligned} I &= S_5 + S_6 + S_4 - S_2 \\ &= (A+D)(E+H) + (B-D)(G+H) + D(G-E) - (A+B)H \\ &= AE + DE + AH + DH + BG - DG + BH - DH + DG - DE - AH - BH \\ &= AE + BG. \end{aligned}$$

$$\begin{aligned} J &= S_1 + S_2 \\ &= A(F-H) + (A+B)H \\ &= AF - AH + AH + BH \\ &= AF + BH. \end{aligned}$$

$$\begin{aligned} K &= S_3 + S_4 \\ &= (C+D)E + D(G-E) \\ &= CE + DE + DG - DE \\ &= CE + DG. \end{aligned}$$

$$\begin{aligned} L &= S_1 - S_7 - S_3 + S_5 \\ &= A(F-H) - (A-C)(E+F) - (C+D)E + (A+D)(E+H) \\ &= AF - AH - AE + CE - AF + CF - CE - DE + AE + DE + AH + DH \\ &= CF + DH. \end{aligned}$$

$$\begin{aligned} S_1 &= A(F-H) \\ S_2 &= (A+B)H \\ S_3 &= (C+D)E \\ S_4 &= D(G-E) \\ S_5 &= (A+D)(E+H) \\ S_6 &= (B-D)(G+H) \\ S_7 &= (A-C)(E+F) \end{aligned}$$

7 recursive multiplications

Volker Strassen, *Gaussian Elimination is not Optimal,* Numerische Mathematik, 1969, doi:10.1007/BF02165411

Strassen's matrix multiplication

- Uses 18 matrix additions, time $O(n^2)$, and 7 recursive matrix multiplications

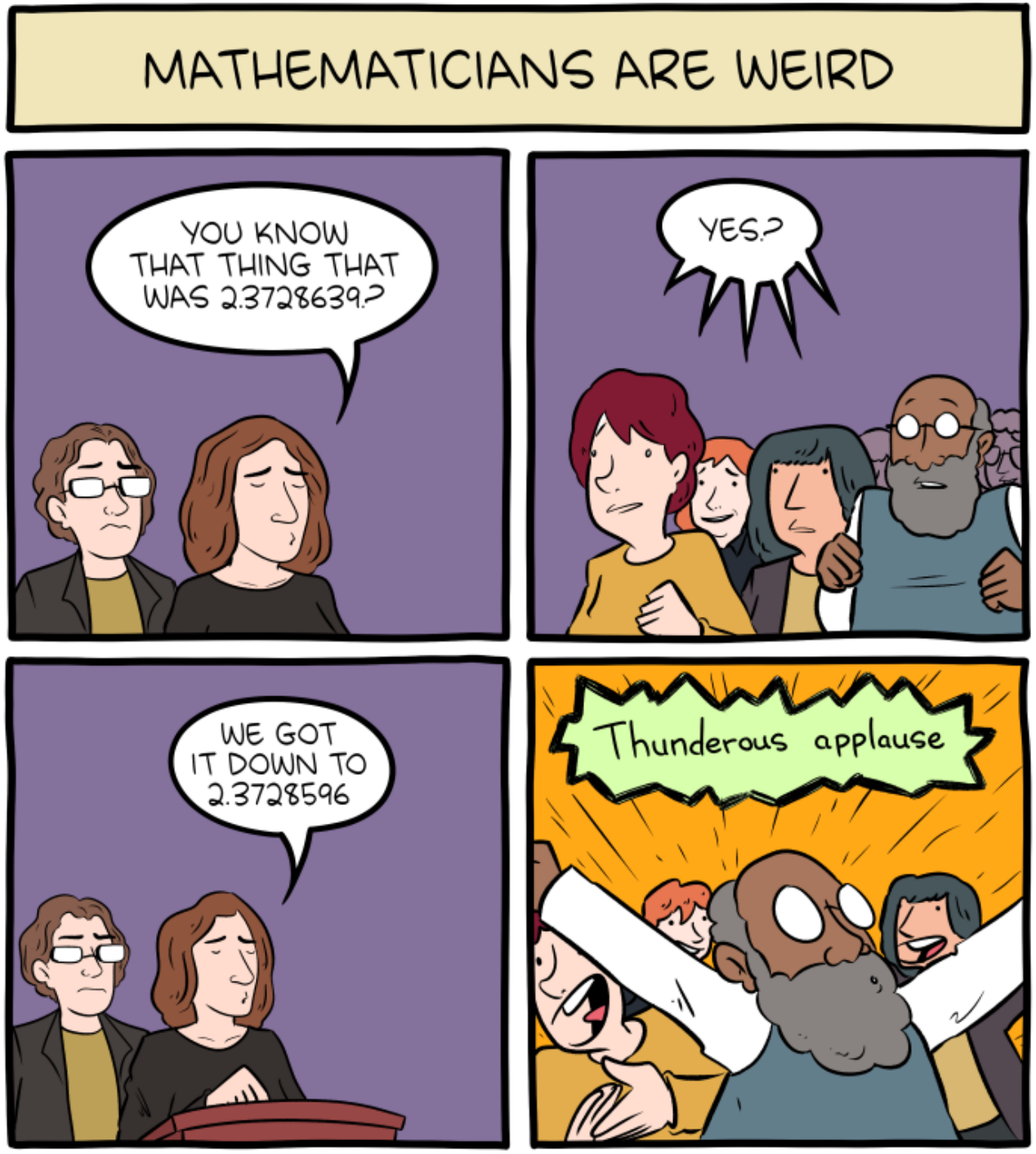
$$T(n) = 7 \cdot T(n/2) + c \cdot n^2 \quad \text{for } n \geq 2$$

$$T(n) = c \quad \text{for } n = 1$$

- $T(n) = O(n^{\log_2 7}) = O(n^{2.8074})$

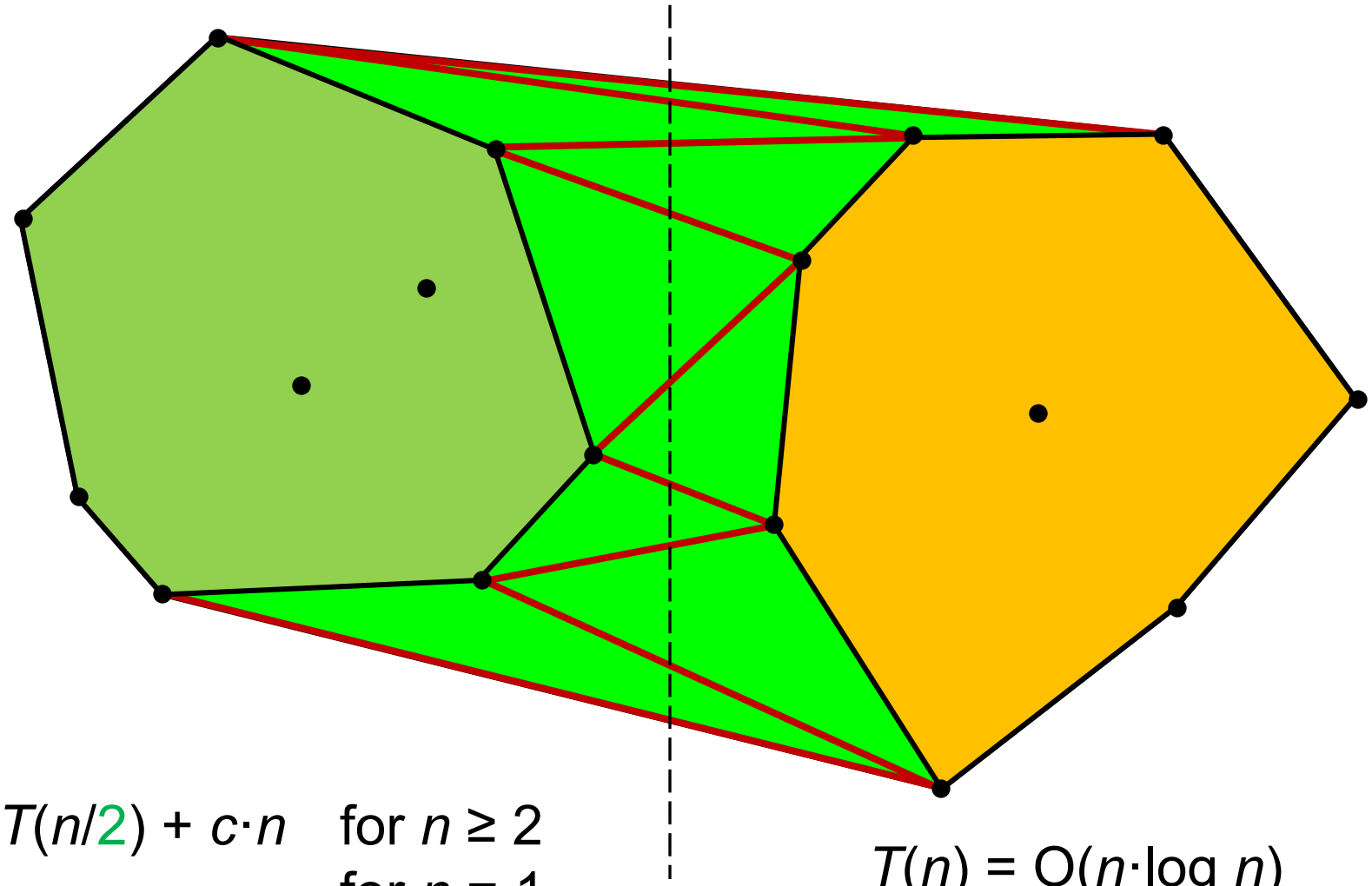
Year	ω, $O(n^\omega)$
1969	2.8074
1978	2.796
1979	2.780
1981	2.522
1981	2.517
1981	2.496
1986	2.479
1990	2.3755
2010	2.3737
2013	2.3729
2014	2.3728639
2020	2.3728596
2022	2.371866
2023	2.371552
2025	2.371339

en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication



www.smbc-comics.com/comic/mathematicians

Convex hull



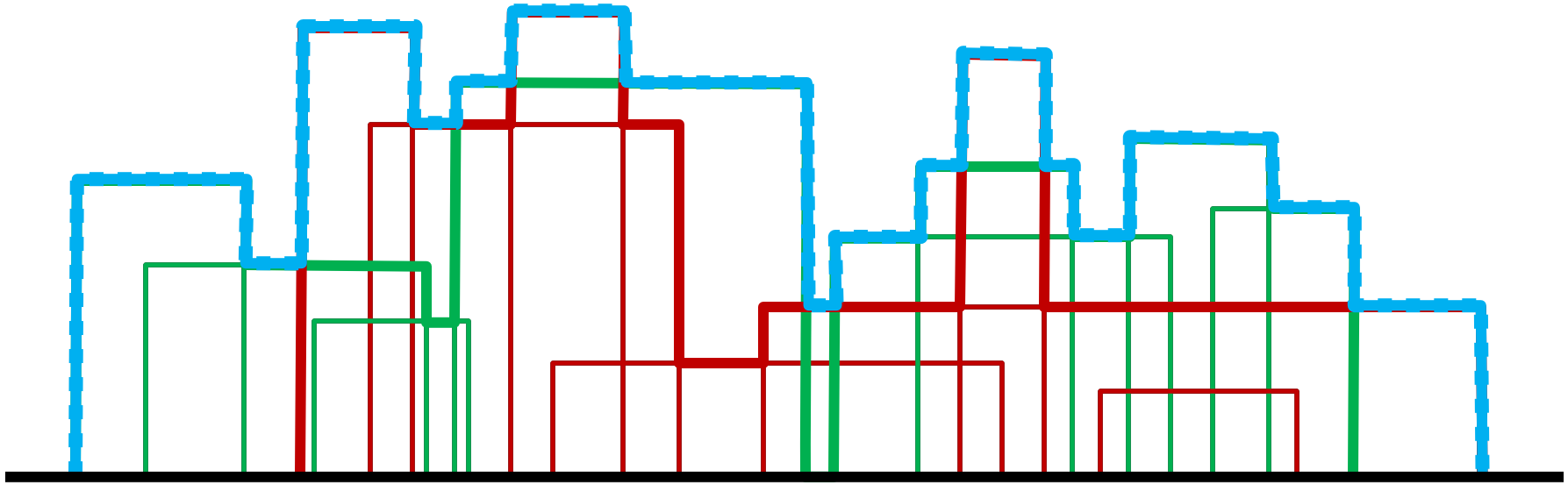
$T(n) = 2 \cdot T(n/2) + c \cdot n$ for $n \geq 2$

$T(n) = c$ for $n = 1$

$T(n) = O(n \cdot \log n)$

Preparata, Hong, *Convex Hulls of Finite Sets of Points in Two and Three Dimensions,* Communications of the ACM, 1977, doi: 10.1145/359423.359430

Skyline
(assignment)



$T(n) = ? \cdot T(n/?) + ?$ for $n \geq 2$

$T(n) = c$ for $n = 1$

Algorithms and Data Structures

Selection in worst-case linear time
[CLRS, Chapter 9.3]

M. Blum, R. W. Floyd, V. Pratt, R. L. Rivest, R. E. Tarjan, 1973, Journal of Computer and System Sciences, 1973, doi: 10.1016/S0022-0000(73)80033-9

Selection

Find the i^{th} smallest element in a list

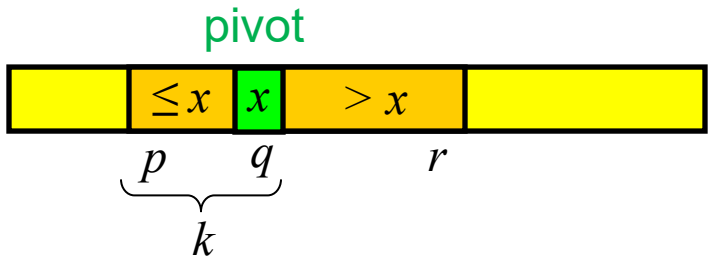
SELECT(L, 5) = 9

Algorithm	Time
Randomized-Select [CLRS, Chapter 9.2]	O(n) expected O(n^2) worst-case
Deterministic-Select [CLRS, Chapter 9.3]	O(n) worst-case

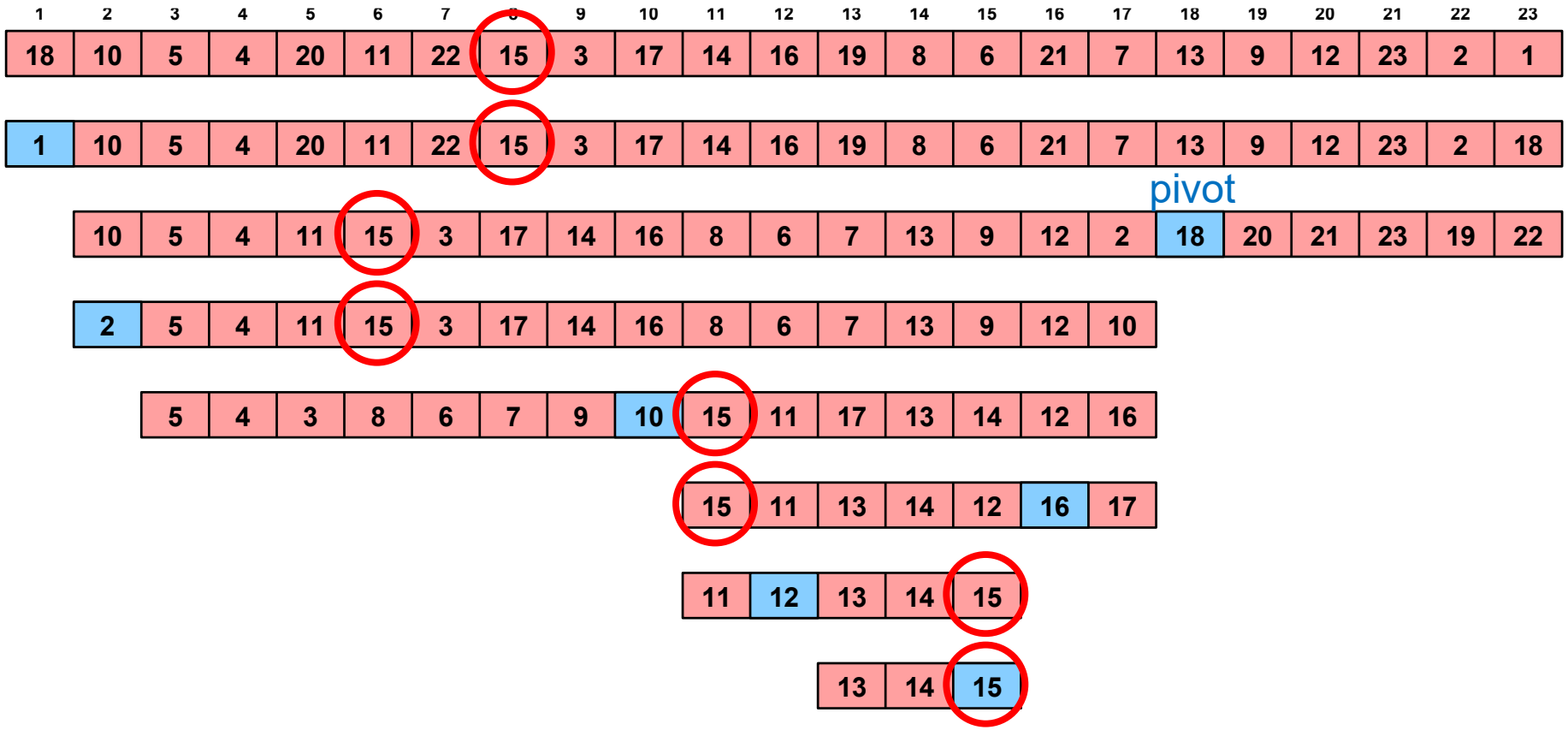
Randomized-Select

Find the i^{th} smallest element in $A[p..r]$ $(1 \le i \le r\text{-}p+1)$

```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2        return A[p]        // 1 ≤ i ≤ r − p + 1 when p == r means that i = 1
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k
6        return A[q]        // the pivot value is the answer
7  elseif i < k
8        return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q + 1, r, i − k)
```



Randomized-Select 15



Randomized-Select

- **Randomized** algorithm (pivot chosen randomly)
  - pivot is in the middle part with constant probability
- Example of **divide and conquer**
  - only 1 smaller subproblem is solved recursively
  - all time is spent in the partition
    (combine step simply returns result from the recursive call)
- Time: worst-case $O(n^2)$, **expected $O(n)$**
  - Analysis *cannot* be solved using the Master Theorem

Deterministic-Select

- Same ideas as Randomized-Select
  - Choose one element as pivot
  - Partition w.r.t. the pivot
  - Make at most one recursive call on elements < or > pivot
- New idea
  - Recursively use Select to find a **good pivot**
- Analysis
  - Divide and conquer

$$T(n) \leq T(a \cdot n) + T(b \cdot n) + c \cdot n$$

  - Cannot use Master Theorem (instead analyze directly)

Deterministic-Select

small input

compute pivot

make 1 recursive call
(like randomized select)

```
SELECT(A, i)
1    if |A| ≤ 5 then
2        sort A and return i'th element
3    partition A into G_1,...,G_⌈n/5⌉ where |G_i| ≤ 5
4    medians = { g_i | g_i median of G_i }
5    pivot = SELECT(medians, ⌊|medians|/2⌋)
6    partition A w.r.t. pivot into A_<, A_= and A_>
7    if i ≤ |A_<| then
8        return SELECT(A_<, i)
9    if i > |A_<| + |A_=| then
10       return SELECT(A_>, i − |A_<| − |A_=|)
11   return pivot
```

Example

A = 30, 37, 91, 78, 34, 76, 22, 72,
99, 63, 57, 57, 83, 97, 78, 44, 3,
25, 44, 86, 44, 82, 52, 26, 53,
90, 70, 17, 9, 56, 76, 89, 9, 37,
39, 80, 84, 23, 42, 97, 72, 26

consider input as $\lceil n/5 \rceil$ groups

sort each group independently

recursively find median →

pivot = median(*medians*)

medians

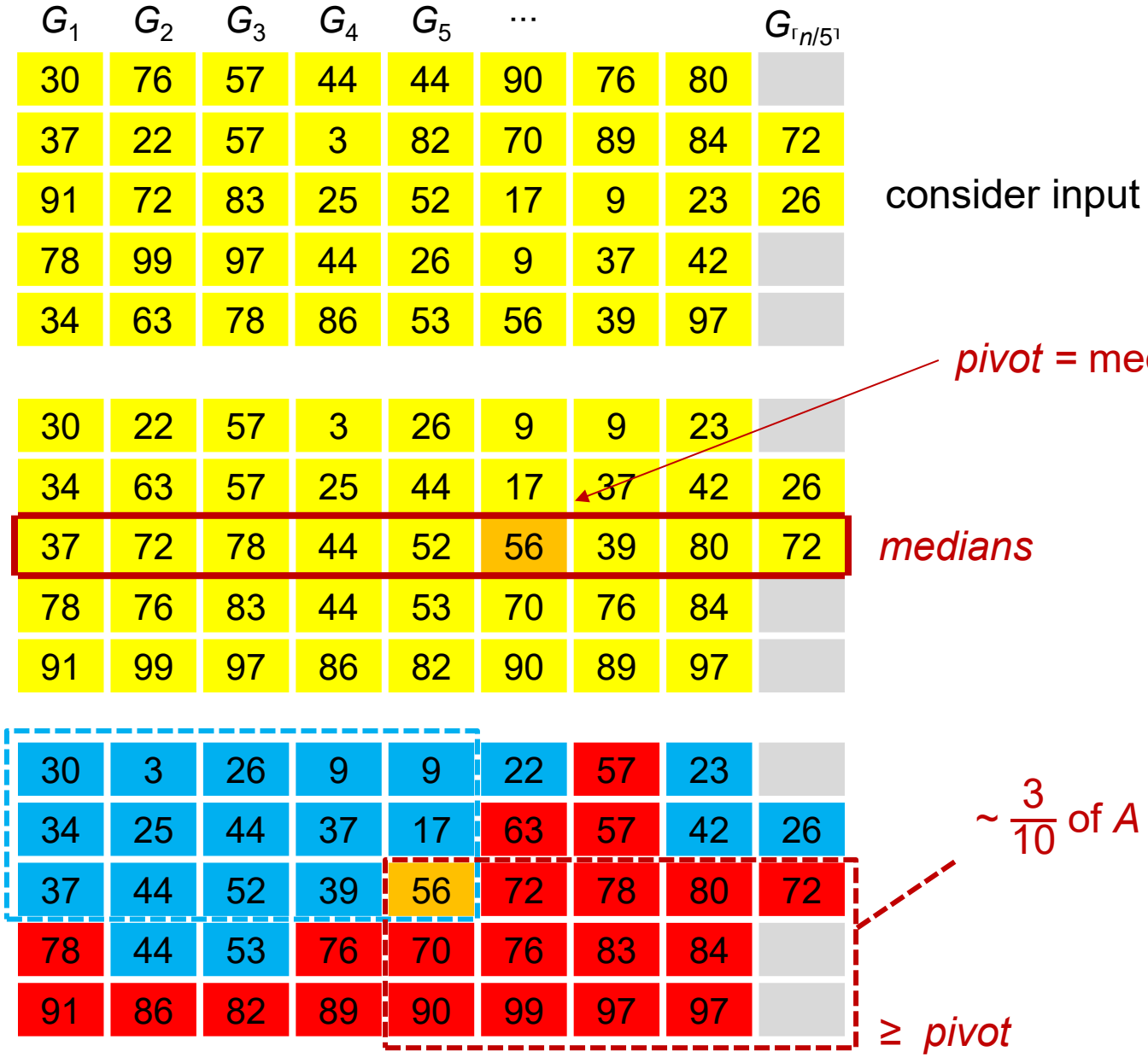
Quality of *pivot* ?

≤ *pivot*

groups permuted such that medians are partitioned w.r.t. *pivot*

~ $\frac{3}{10}$ of A

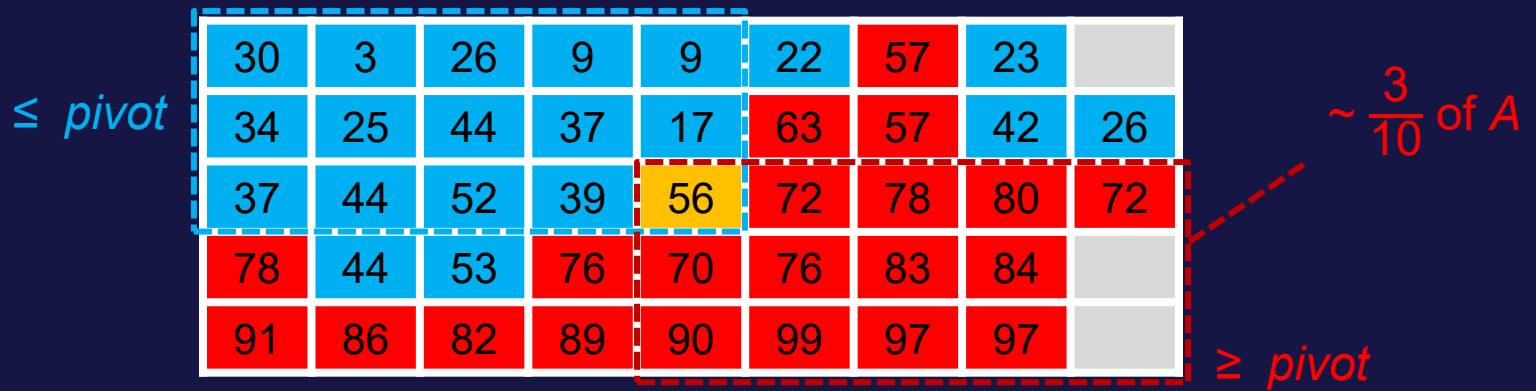
≥ *pivot*



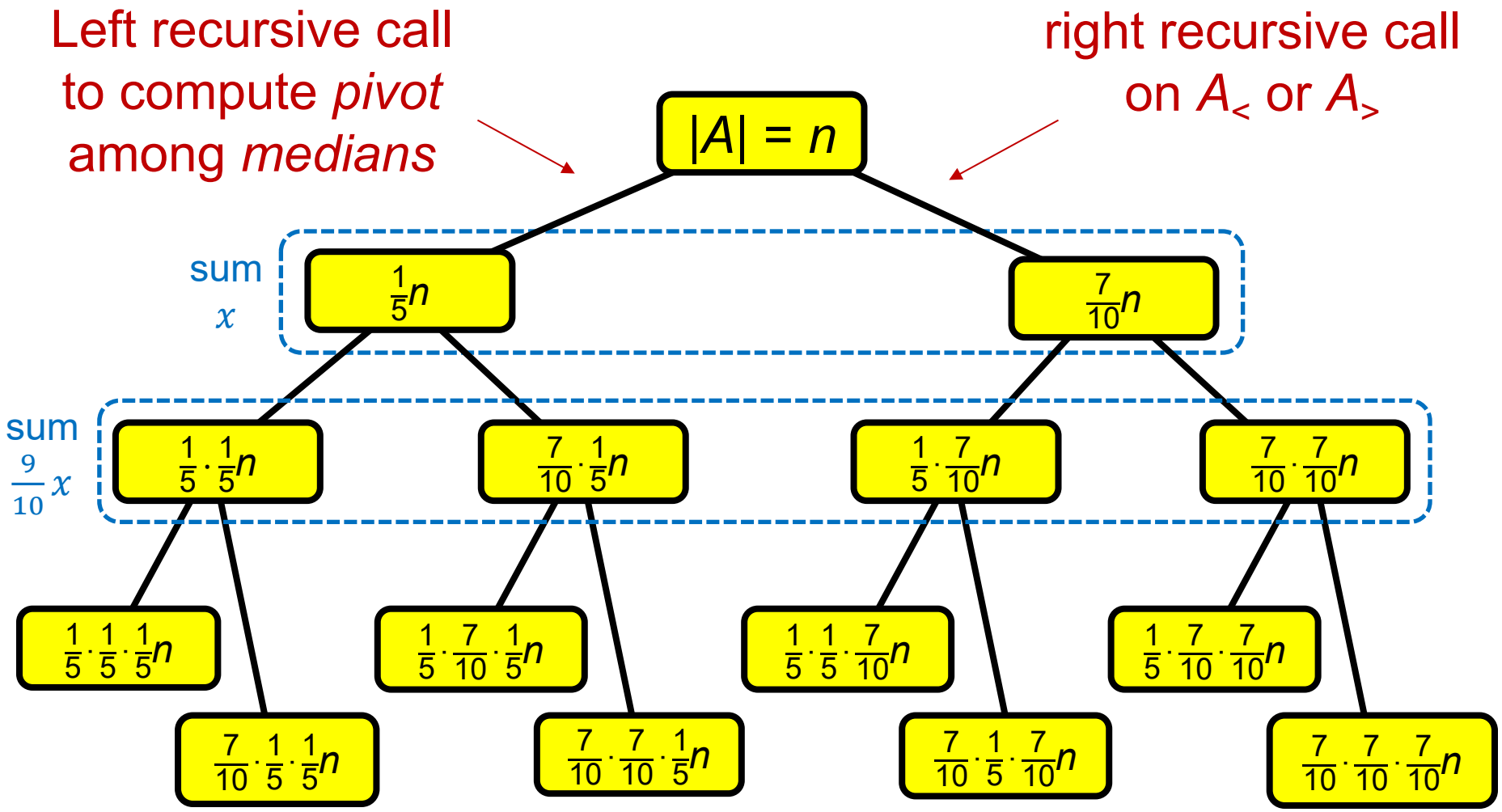
Maximum size of $A_<$?

$A_<$ are all elements in A smaller than the pivot

a) ~ 3/10 · |A|
b) ~ 1/4 · |A|
c) ~ 1/2 · |A|
d) ~ 7/10 · |A|
e) ~ 3/4 · |A|
f) Don't know



Recursion tree SELECT(A, i)



Note: Proof ignores that there can be O(1) additional elements to each recursive call

Analysis (recurrence)

recursive call to compute *pivot*

recursive call on $A_<$ or $A_>$ (upper bound)

time to compute the median of each group and to create partition $A_<$, $A_=$ and $A_>$

$$T(n) \le \begin{cases} T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } 1 \le n \le 5 \end{cases}$$

time to sort ≤ 5 elements

Note: Proof ignores that there can be O(1) additional elements to each recursive call

Analysis (considering recursion tree)

$$T(n) \le \begin{cases} T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } 1 \le n \le 5 \end{cases}$$

Note: The sum of subproblem sizes from depth $i + 1$ to depth i decreases by at least a factor $\frac{1}{5} + \frac{7}{10} = \frac{9}{10}$

$$T(n) \le \sum_{i=0}^{\infty} c \cdot n \cdot \left(\frac{9}{10}\right)^i = c \cdot n \cdot \frac{1}{1 - 9/10} = 10 \cdot c \cdot n$$

Note: Proof ignores that there can be O(1) additional elements to each recursive call

Analysis (by induction)

$$T(n) \le \begin{cases} T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } 1 \le n \le 5 \end{cases}$$

Prove $T(n) \le k \cdot n$ for some k

Proof

Base case $1 \le n \le 5$:

$T(n) = c \le k \cdot n$ if $c \le k$

$$T(n) \le 10 \cdot c \cdot n$$

Induction step $n > 5$:

$$T(n) \le T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n \overset{\text{i.h.}}{\le} k \cdot \frac{1}{5}n + k \cdot \frac{7}{10}n + c \cdot n = \left(k\frac{9}{10} + c\right)n \le k \cdot n$$

if $k\frac{9}{10} + c \le k \iff c \le k\left(1 - \frac{9}{10}\right) \iff 10 \cdot c \le k$

Note: Proof ignores that there can be O(1) additional elements to each recursive call

Precise analysis

(tighter analysis than CLRS)

|*medians*|

max{ $|A_<|$, $|A_>|$ }

$$T(n) \le \begin{cases} T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(n - 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil + 2\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } n \le 5 \end{cases}$$

Solution $T(n) \le c \cdot \max\{1, 10n - 30\}$

Proof: For $1 \le n \le 15$ compute recurrence

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T(n) / c \le$	1	1	1	1	1	8	16	25	35	46	28	38	49	61	44
$\max\{1, 10n - 30\}$	1	1	1	10	20	30	40	50	60	70	80	90	100	110	120

(continues)

Precise analysis (continued)

For $n \geq 16$ proof by **induction**

Induction hypothesis (assume already proved)

$$T(k) \leq c \cdot \max\{1, 10k - 30\} \quad \text{for} \quad 1 \leq k \leq n-1$$

Induction step (prove for n)

$$\begin{aligned} T(n) &\leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(n - 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil + 2\right) + c \cdot n \\ &\leq c \cdot \left(10\left\lceil \frac{n}{5} \right\rceil - 30 + 10\left(n - 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil + 2\right) - 30 + n\right) \\ &\leq c \cdot \left(10\left(\frac{n}{5} + 1\right) + 10\left(n - 3\frac{n/5}{2} + 2\right) - 60 + n\right) \\ &= c \cdot (10n - 30) \end{aligned}$$

□

Worst-case # comparisons for Select $n = 5$?

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6

g) 7
h) 8
i) 9
j) Don't know

Even more precise analysis – # comparisons

compute *medians*

compute $|A_<|$, $|A_=|$ and $|A_>|$

$$T(n) \le \begin{cases} T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(n - 3\left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil + 2\right) + 7\left\lceil \frac{n}{5} \right\rceil + n - 1 & \text{for } n > 5 \\ \begin{array}{c|ccccc} n & 1 & 2 & 3 & 4 & 5 \\ \hline T(n) \le & 0 & 1 & 3 & 5 & 7 \end{array} & \text{for } n \le 5 \end{cases}$$

Solution $T(n) \le \max\{n, 24n - 72\}$

Proof: $n \le 15$ check manually, $n \ge 16$ by induction.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T(n) \le$	0	1	3	5	7	21	38	57	79	103	66	88	112	138	104
$\max\{n, 24n - 72\}$	1	2	3	24	48	72	96	120	144	168	192	216	240	264	268

Selection

Algorithm	Time
Randomized-Select [CLRS, Chapter 9.2] Hoare 1961	$O(n)$ expected $O(n^2)$ worst-case
Deterministic-Select [CLRS, Chapter 9.3] Blum et al. 1973	$O(n)$ worst-case
Median worst-case comparisons Dor, Zwick 1995, 1996	$\leq 2.95n$ $\geq (2 + \varepsilon)n$ $\varepsilon \approx 2^{-80}$

C. A. R. Hoare, *Algorithm 65: Find,* Communications of the ACM, 1961, doi: 10.1145/366622.366647
Blum, Floyd, Pratt, Rivest, Tarjan, *Time bounds for selection*, Journal of Computer and System Sciences, 1973, doi: 10.1016/S0022-0000(73)80033-9
Dorit Dor, Uri Zwick, *Median Selection Requires (2+ε)n Comparisons*, SIAM Journal of Discrete Mathematics, 2001, doi: 10.1137/S0895480199353895
Dorit Dor, Uri Zwick, *Selecting the Median*, SIAM Journal of Computing, 1999, doi: 10.1137/S0097539795288611

Algorithms and Data Structures

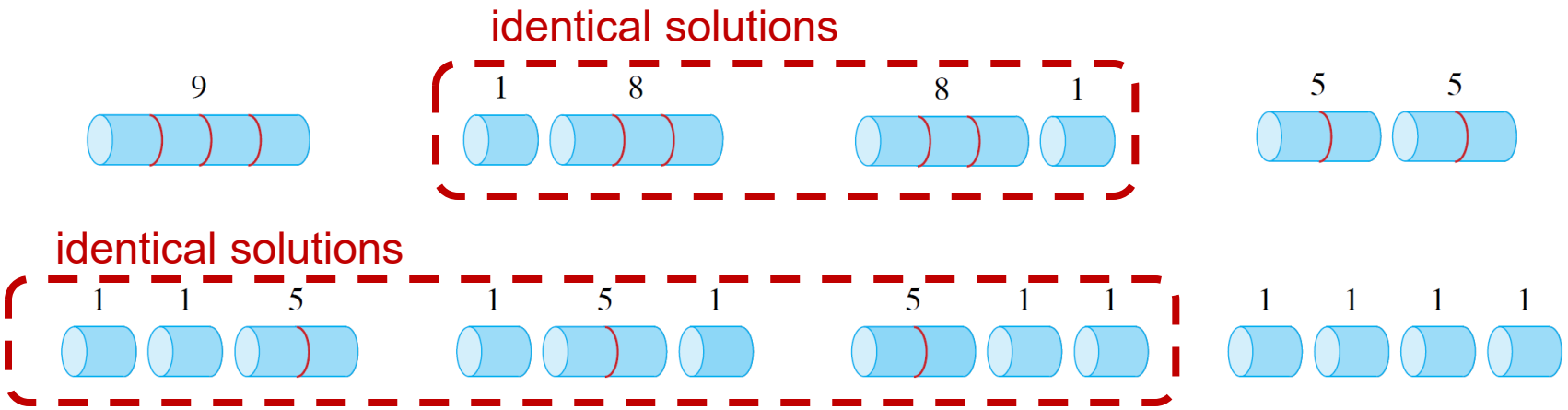
Dynamic programming
[CLRS, Chapter 14]

Dynamic programming

- **General algorithm design technique**
  - Works for a long sequence of problems
  - A solution can be computed from solutions to subproblems
- **Recursive solution**
  - Typical exponential time
- **Dynamic programming**
  - Reuse solutions to already considered subproblems
  - Approaches: recursive with memoization or systematic tabulation
  - Typically, polynomial time

Example – rod cutting

Problem: Cut a rod into smaller pieces, where each piece has a price, maximizing the sum of prices



A rod of length 4 can be partitioned in 8 ways – but only 5 different results

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Maximum price for cutting rod of length 12?

a) 12
b) 30
c) 32
d) 34

e) 35
f) 36
g) Don't know

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

Optimal cutting of rods of length 1..10

best price	cutting
$r_1 = 1$	$1 = 1$ (no cuts),
$r_2 = 5$	$2 = 2$ (no cuts),
$r_3 = 8$	$3 = 3$ (no cuts),
$r_4 = 10$	$4 = 2 + 2$,
$r_5 = 13$	$5 = 2 + 3$,
$r_6 = 17$	$6 = 6$ (no cuts),
$r_7 = 18$	$7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$	$8 = 2 + 6$,
$r_9 = 25$	$9 = 3 + 6$,
$r_{10} = 30$	$10 = 10$ (no cuts).

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

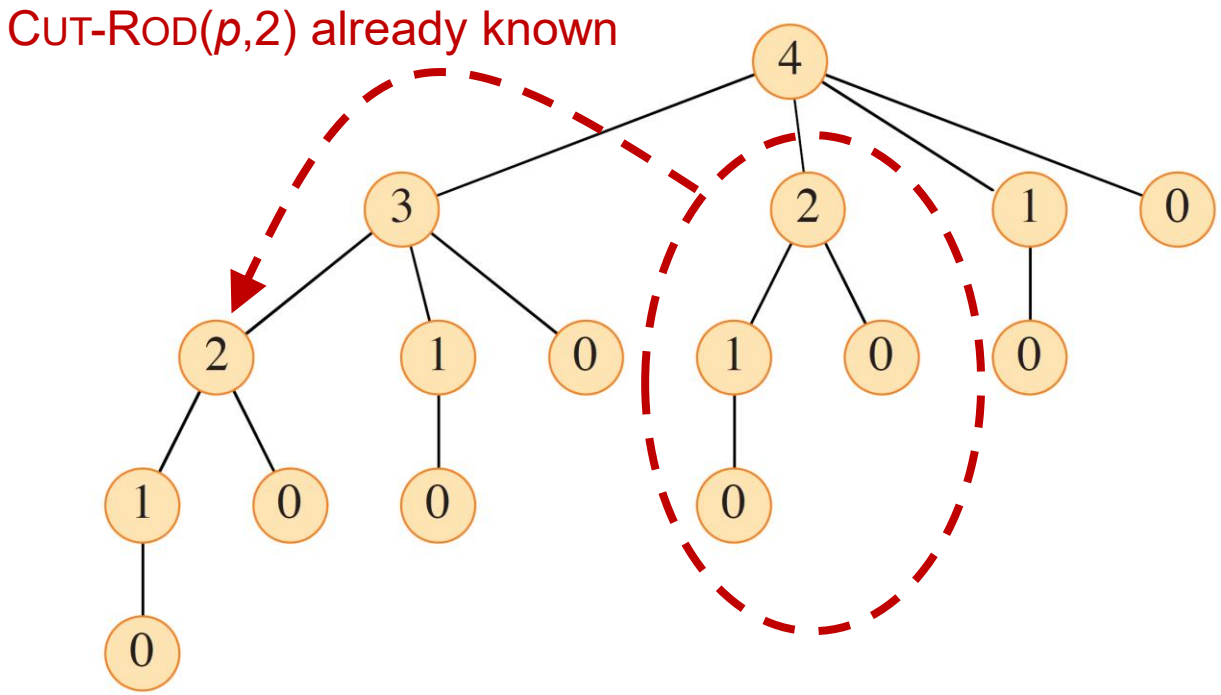
Optimal rod cutting – recursive solution

$$r_n = \begin{cases} 0 & \text{if } n = 0 \\ \max_{i=1..n}(p_i + r_{n-i}) & \text{otherwise} \end{cases}$$

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max{q, p[i] + CUT-ROD(p, n - i)}
6  return q
```

Optimal rod cutting – recursive solution

$$r_n = \begin{cases} 0 & \text{if } n = 0 \\ \max_{i=1..n}(p_i + r_{n-i}) & \text{otherwise} \end{cases}$$

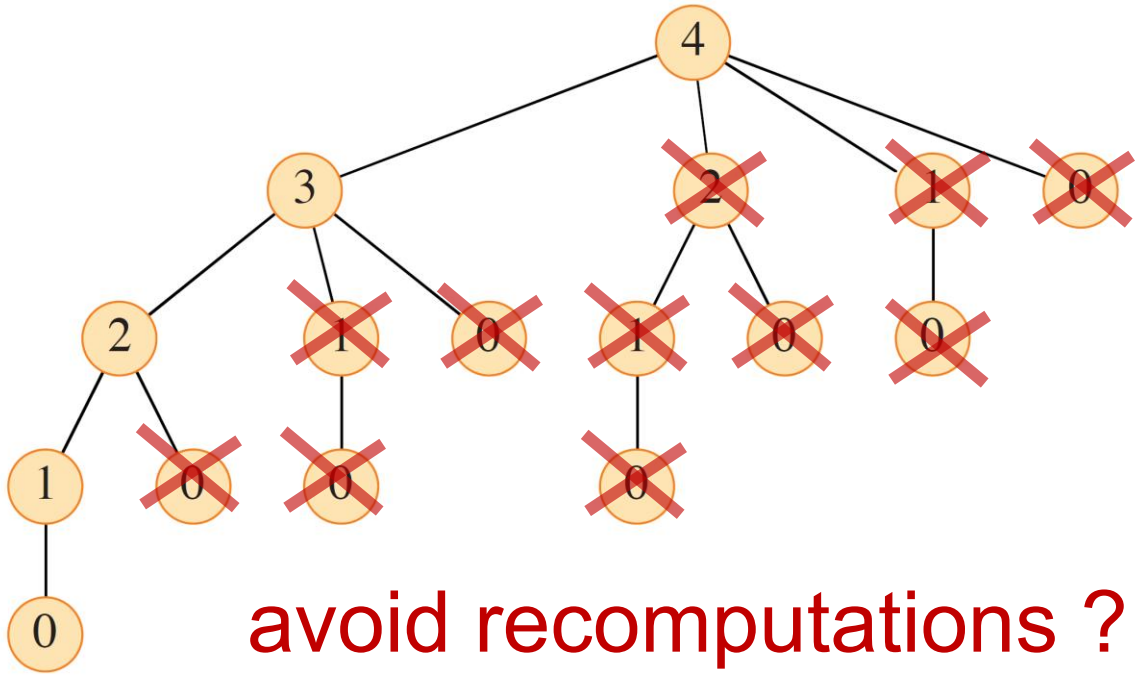



```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = −∞
4  for i = 1 to n
5      q = max{q, p[i] + CUT-ROD(p, n − i)}
6  return q
```

Time $O(2^n)$

Optimal rod cutting – recursive solution

$$r_n = \begin{cases} 0 & \text{if } n = 0 \\ \max_{i=1..n}(p_i + r_{n-i}) & \text{otherwise} \end{cases}$$



avoid recomputations ?

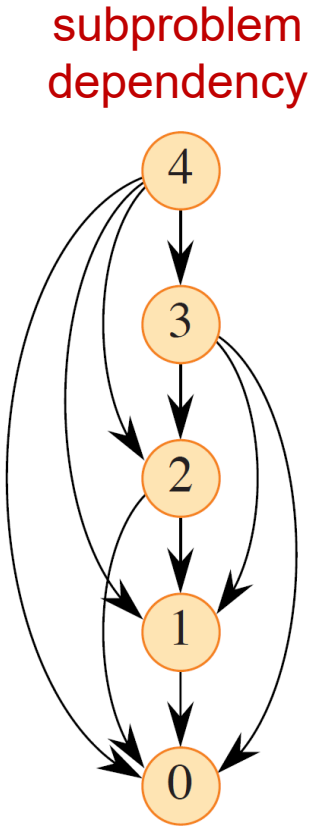
```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max{q, p[i] + CUT-ROD(p, n - i)}
6  return q
```

Optimal rod cutting – memoized recursion

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0:n] be a new array
2  for i = 0 to n
3      r[i] = −∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = −∞
6      for i = 1 to n
7          q = max{q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n − i, r)}
8  r[n] = q
9  return q
```

recurrence

remember result

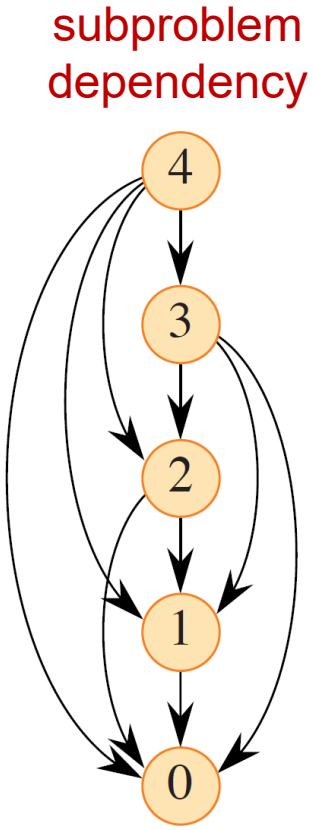


Time $O(n^2)$

Optimal rod cutting – systematic tabulation

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0 : n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = −∞
5      for i = 1 to j
6          q = max{q, p[i] + r[j − i]}
7      r[j] = q
8  return r[n]
```

order crucial !



i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30

Time $O(n^2)$

Optimal rod cutting – printing solution

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0:n] and s[1:n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

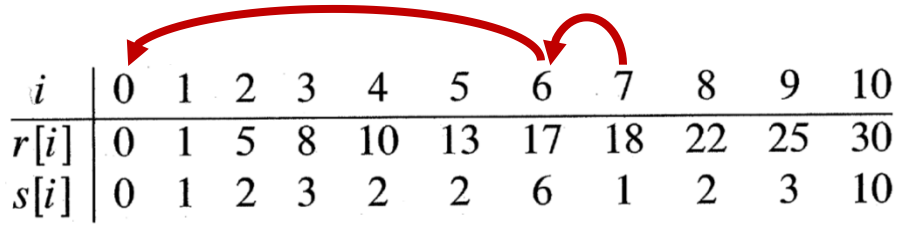
i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Time $O(n^2)$

Output for $n = 7$?

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n − s[n]
```



i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

a) 18 17

b) 17 18

c) 6 1

d) 1 6

e) 7 6

f) Don’t know

Maximum weight for an increasing subsequence?

Let $X = x_1,\ldots,x_n$ and $W = w_1,\ldots,w_n$ be sequences, where each element x_i is assigned a weight w_i.

Find an increasing subsequence of X with maximum weight, i.e., find positions $i_1<i_2<\cdots<i_l$ where $x_{i_1}<x_{i_2}<\cdots<x_{i_l}$ and $w_{i_1}+w_{i_2}+\cdots+w_{i_l}$ is maximized.

i	1	2	3	4	5	6	7	8	9
x_i	6	8	7	5	3	9	4	10	5
w_i	1	1	2	4	1	1	2	4	3

Exam, august 2008, question 4

a) 4
b) 5
c) 6
d) 7
e) 8
f) 9
g) 10
h) 11
i) 12
j) Don't know

MEH(6) ?

We let *MEH*(i) (max-ending-here) denote the maximum weight of an increasing subsequence ending with x_i.

$$MEH(i) = \begin{cases} w_i + \max\{MEH(j) \mid 1 \le j < i \wedge x_j < x_i\} & \text{if there exists } 1 \le j < i \wedge x_j < x_i \\ w_i & \text{otherwise} \end{cases}$$

i	1	2	3	4	5	6	7	8	9
$MEH(i)$	**1**	**2**	**3**	**4**	**1**	**?**			

i	1	2	3	4	5	6	7	8	9
x_i	6	8	7	5	3	9	4	10	5
w_i	1	1	2	4	1	1	2	4	3

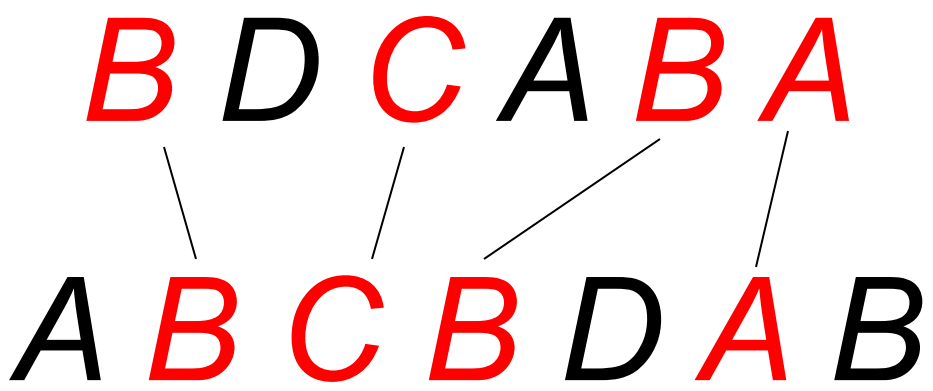
Exam, august 2008, question 4

(answer subquestion b)

```
msf = −∞
for i = 1 to n
  m = 0
  for j = 1 to i − 1
    if x_j < x_i and MEH(j) > m then m = MEH(j)
  MEH(j) = m + w_i
  if MEH(j) > msf then msf = MEH(j)
return msf
```

a) 1
b) 2
c) 3
d) 4
e) 5 
f) 6
g) 7
h) 8
i) 9
j) Don't know

Longest common subsequence



Longest common subsequence ?

ABABGACBABAD

BACABAABABC

a) BACBABA

b) BABBAB

c) BAABABA

d) BACABAB

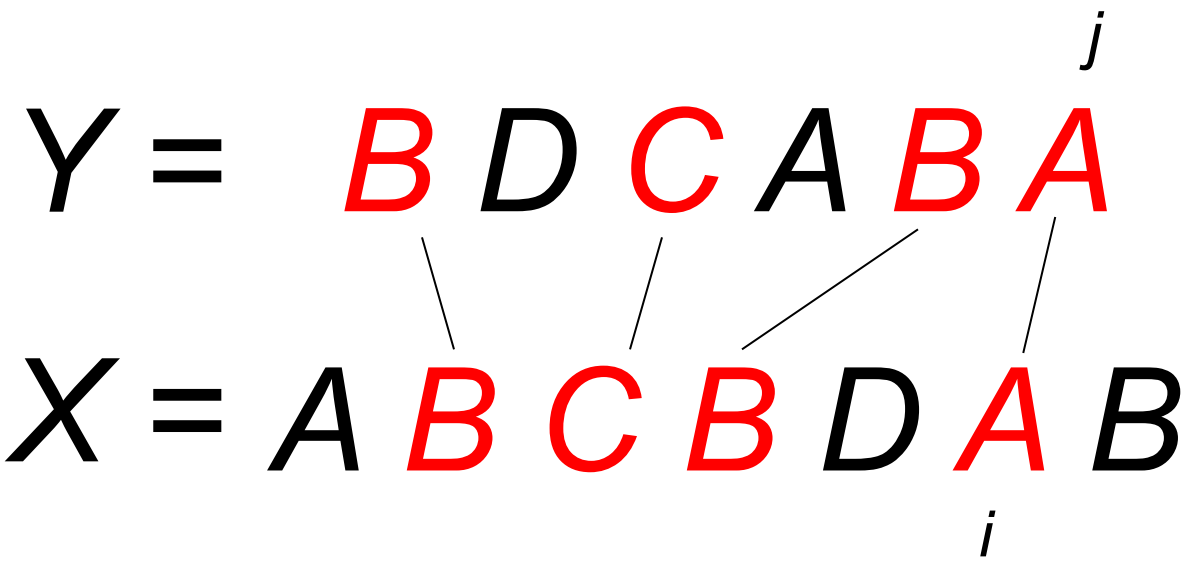
e) Don't know

Answer not unique

Easy to check subsequence (greedy)

Exponential many possible subsequences

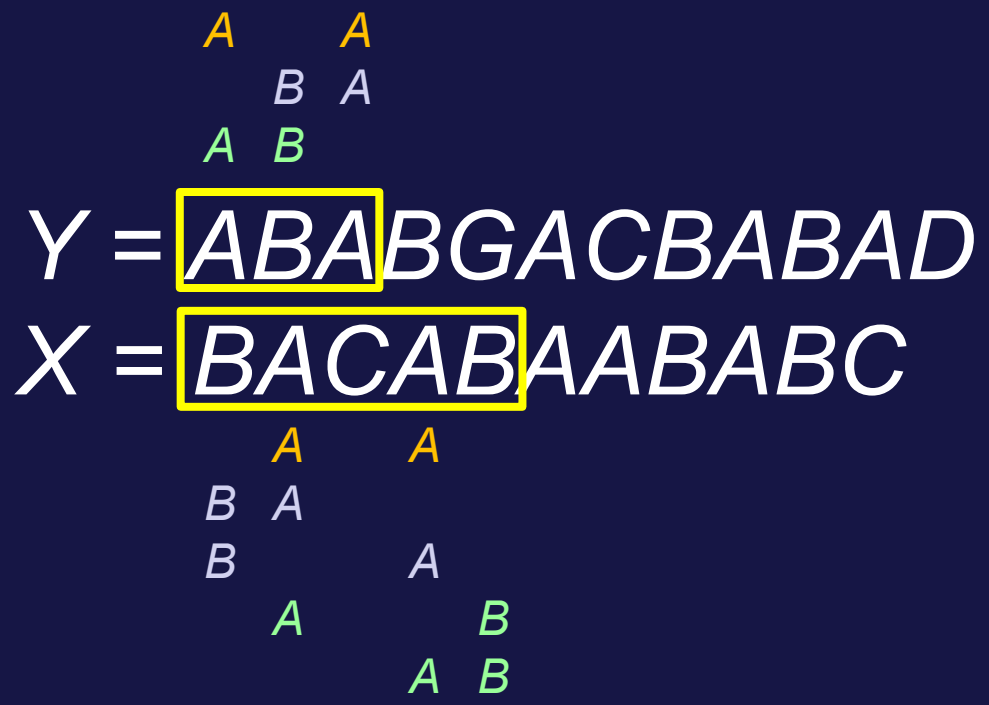
Longest common subsequence



$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1,j-1]+1 & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

length of longst common subsequence of $x_1 \cdots x_i$ and $y_1 \cdots y_j$

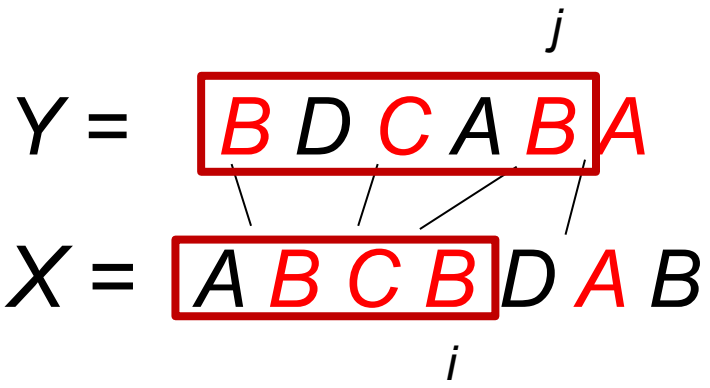
c[5,3] ?



a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) Don't know

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

Longest common subsequence



i		j	0	1	2	3	4	5	6
				y_j	B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0\,, \\ c[i-1,j-1] + 1 \;\nwarrow & \text{if } i,j > 0 \text{ and } x_i = y_j\,, \\ \max\{c[i,j-1], c[i-1,j]\} & \text{if } i,j > 0 \text{ and } x_i \neq y_j\,. \end{cases}$$

Longest common subsequence

```
LCS-LENGTH(X, Y, m, n)
 1  let b[1 : m, 1 : n] and c[0 : m, 0 : n] be new tables
 2  for i = 1 to m
 3      c[i, 0] = 0
 4  for j = 0 to n
 5      c[0, j] = 0
 6  for i = 1 to m
 7      for j = 1 to n
 8          if x_i == y_j
 9              c[i, j] = c[i − 1, j − 1] + 1
10              b[i, j] = "↖"
11          elseif c[i − 1, j] ≥ c[i, j − 1]
12              c[i, j] = c[i − 1, j]
13              b[i, j] = "↑"
14          else c[i, j] = c[i, j − 1]
15              b[i, j] = "←"
16  return c and b
```

		j	0	1	2	3	4	5	6
i			y_j	B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Time $O(n \cdot m)$

Longest common subsequence

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2        return
3  if b[i, j] == "↖"
4        PRINT-LCS(b, X, i − 1, j − 1)
5        print x_i
6  elseif b[i, j] == "↑"
7        PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

Time $O(n + m)$

Space usage for LCS algorithm to find an LCS ?

a) O($n + m$)

b) O($n \cdot m$)

c) Don't know

	j	0	1	2	3	4	5	6
i		y_j	B	D	C	A	B	A
0	x_i	0	0	0	0	0	0	0
1	A	0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B	0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C	0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B	0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D	0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A	0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B	0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

$m = |X|$ $n = |Y|$

LCS length

Fill out table row-by-row, only remember previous row

O(min(n, m)) space

not curriculum

Computation of an LCS in O(*n* + *m*) space

Idea

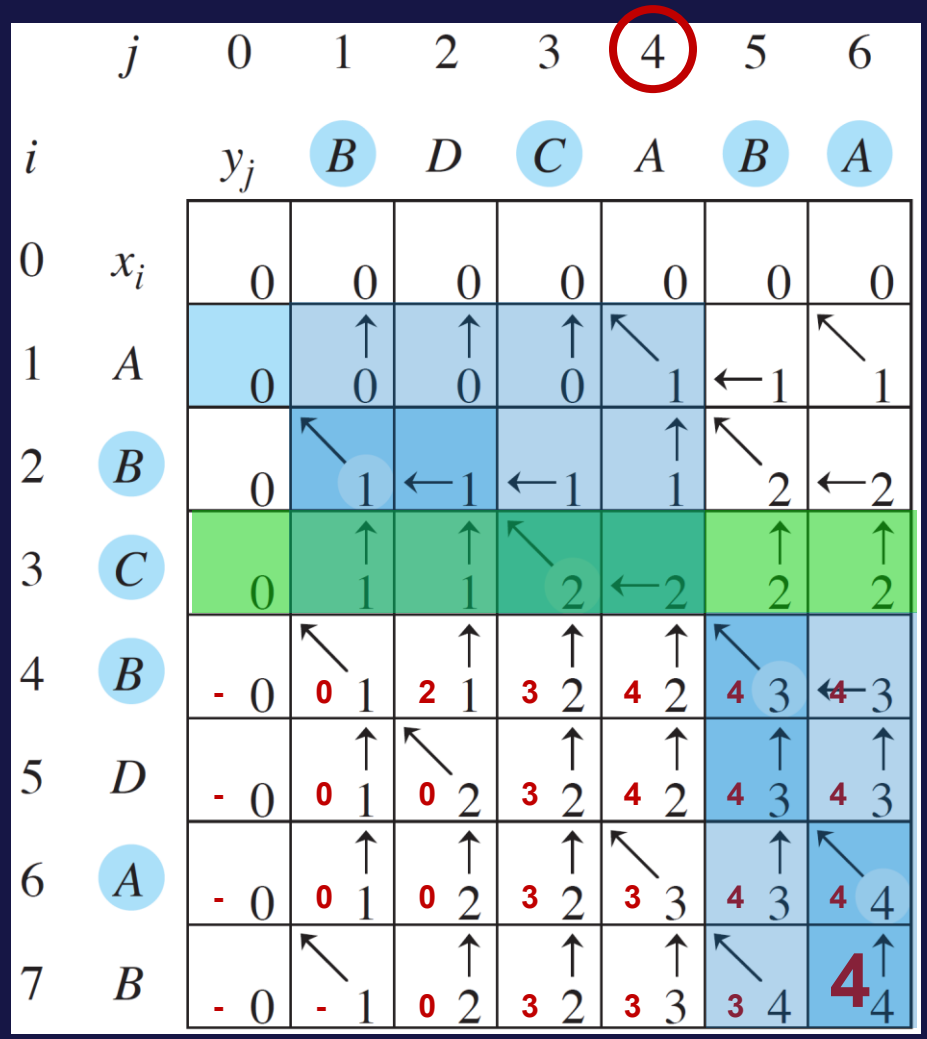
Compute path **without storing** the full dynamic programming table

1) Compute where the path crosses the middle row

2) Recursively solve the two subproblems

Time O(*n* · *m*) Space O(*n* + *m*)

Example of **divide and conquer** used inside a **dynamic programming** algorithm



m = |*X*| *n* = | *Y* |

D.S. Hirschberg, *A linear space algorithm for computing maximal common subsequences*, Communications of the ACM, 1975, doi: 10.1145/360825.360861

Longest common subsequence

$O(n \cdot m)$ time	dynamic programming
$O(n+m)$ space	dynamic programming and divide and conquer [Hirschberg 1975]

Open problem

Is it possible to compute the longest common subsequence of two strings of length n in worst-case $O(n^{2-\varepsilon})$ time ?

– Would break with the "Strong Exponential Time Hypothesis"

Abboud, Backurs, Williams, *Tight Hardness Results for LCS and Other Sequence Similarity Measures*, FOCS, 2015, doi: 10.1109/FOCS.2015.14
Bringmann, Künnemann, *Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping*, FOCS, 2015, doi: 10.1109/FOCS.2015.15

not curriculum

Hunt-Szymanski algorithm

		0	1	2	3	4	5	6	***L***
	y_j		B	A	B	C	A	B	
0	x_i	0	0	0	0	0	0	0	
1	A	0	0	1	1	1	1	1	[2]
2	B	0	1	1	2	2	2	2	[1, 3]
3	A	0	1	2	2	2	3	3	[1, 2, 5]
4	C	0	1	2	2	3	3	3	[1, 2, 4]
5	B	0	1	2	3	3	3	4	[1, 2, 3, 6]

Assume **few pairwise matches**

$$r = |\{ (i, j) \mid x_i = y_j \}| \ll n \cdot m$$

$$r = \overset{\text{A}}{2}\cdot 2 + \overset{\text{B}}{2}\cdot 3 + \overset{\text{C}}{1}\cdot 1 = 11$$

(1,5) (1,2) (2,6) (2,3) (2,1) (3,5) (3,2) (4,4) (5,6) (5,3) (5,1)

- LCS table *not* explicitly computed; only consider matches
- **Observation: LCS(*X*, *Y*) = LIS(Matches(*X*, *Y*))**, LIS = longest increasing subsequence
  Sort increasing *i*, decreasing *j*; time $O((m + n + r) \cdot \log \min(n, m))$, space $O(r + m + n)$
- |LCS|: Compute for each row ***L*** = first column with 1, 2, 3, …
  and only changes to ***L*** for each row – total *r* binary searches in ***L***
  Time $O((r + m + n) \cdot \log \min(n, m))$, space $O(\min(n, m))$

Hunt, Szymanski, *A fast algorithm for computing longest common subsequences,* Communications of the ACM, 1977, doi: 10.1145/359581.359603

Text formatting – line breaks

tex.stackexchange.com/questions/120271/alternatives-to-latex/120279

The first paragraph of Herman Melville's *Moby Dick* typeset using three different programs.
The text is set using Garamond Premier Pro 12/14 in a 5 cm wide column, fully justified.
Created by Roel Zinkstok of Zink Typography (www.zinktypografie.nl), January 2010

Microsoft Word 2008

Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a

Adobe InDesign CS4

Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral prin-

pdf-LaTeX 3.1415926

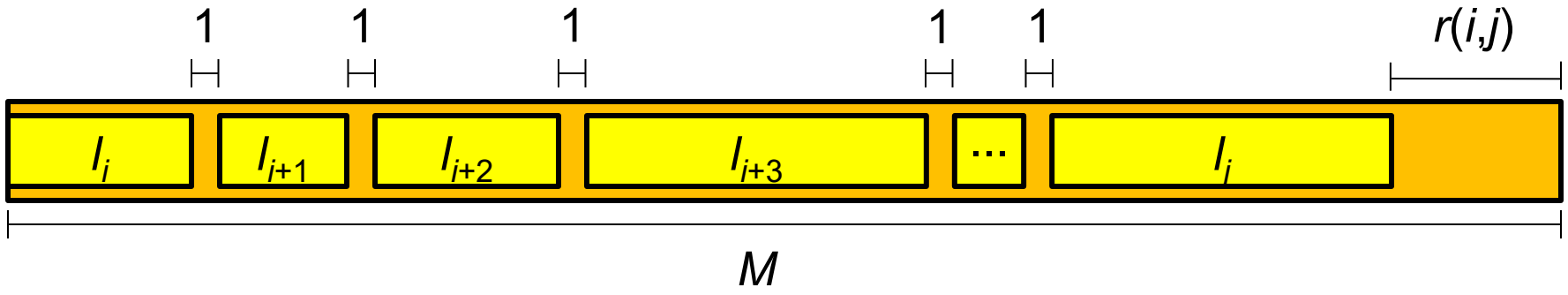
Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself involuntarily pausing before coffin warehouses, and bringing up the rear of every funeral I meet; and especially whenever my hypos get such an upper hand of me, that it requires a strong moral

Printing neatly [CLRS Problem 14-4]

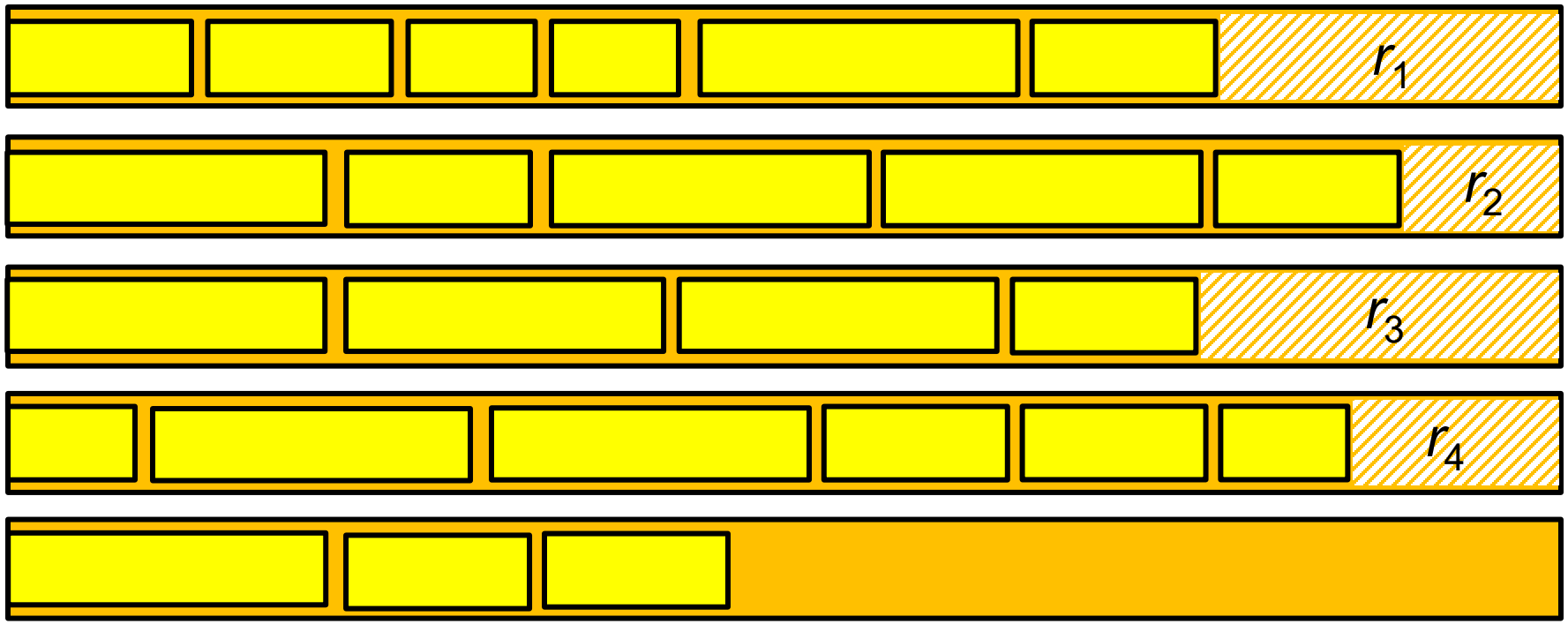
"Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths l_1, l_2,..., l_n, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \le j$, and we leave exactly one space between words, the number of *extra space characters* at the end of the line is

$$M - j + i - \sum_{k=i..j} l_k ,$$

which must be nonnegative so that the words fit on the line. We wish to **minimize** the sum, over all lines except the last, of **the cubes of the numbers of extra space characters at the ends of lines**. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm."



$$r(i, j) = M - j + i - \Sigma_{k=i..j}\, l_k$$



$$\text{penalty} \quad = \quad r_1^{\,3} + r_2^{\,3} + r_3^{\,3} + r_4^{\,3}$$

Printing neatly

Place the first j words on full lines

$$C[j] = \begin{cases} 0 & \text{if } j = 0 \\ \min\{C[i-1] + r(i,j)^3 \mid 1 \le i \le j \ \wedge \ r(i,j) \ge 0\} & \text{if } j > 0 \end{cases}$$

Best solution

$$\min\{C[j] \mid 0 \le j < n \ \wedge \ r(j+1, n) \ge 0\}$$

remaining words j+1, j+2, ..., n
on last line without penality

Printing neatly

```
 1 // compute r
 2 for i = 1 to n
 3   sum = 0
 4   for j = i to n
 5     sum = sum + l[j]
 6     r[i,j] = M − j + i - sum

 7 // compute C
 8 C[0] = 0
 9 for j = 1 to n
10   C[j] = +∞
11   for i = 1 to j
12     if r[i,j]>=0 and C[i-1] + r[i,j]^3<C[j] then
13       C[j] = C[i-1] + r[i,j]^3
14       I[j] = i
```

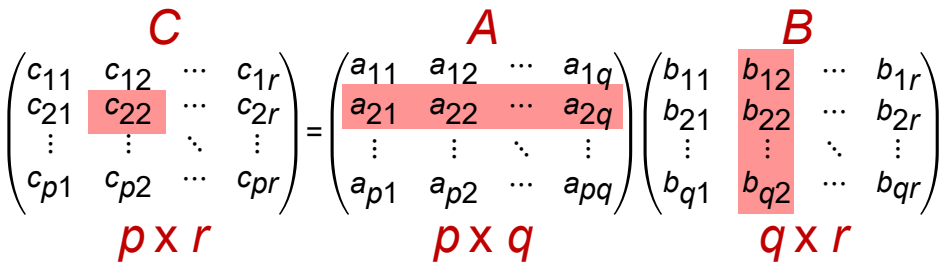
```
15 // print best solution
16 proc report(j)
17   if j > 0 then
18     i = I[j]
19     report(i - 1)
20     print word[i]..word[j] <newline>

21 k = n - 1
22 for j = 0 to n - 2
23   if r[j+1,n]>=0 and C[j]<C[k] then
24     k = j
25 report(k)
26 print word[k+1]..word[n] <newline>
```

Time and space $O(n^2)$

$$C[j] = \begin{cases} 0 & \text{if } j = 0 \\ \min\{C[i-1] + r(i,j)^3 \mid 1 \le i \le j \wedge r(i,j) \ge 0\} & \text{if } j > 0 \end{cases}$$

$$\min\{C[j] \mid 0 \le j < n \wedge r(j+1, n) \ge 0\}$$

Matrix multiplication




```
RECTANGULAR-MATRIX-MULTIPLY(A, B, C, p, q, r)
1  for i = 1 to p
2      for j = 1 to r
3          for k = 1 to q
4              c_ij = c_ij + a_ik · b_kj
```

Multiplication of two matrices A and B with dimensions p x q and q x r takes time $O(p \cdot q \cdot r)$

Matrix-chain multiplication

$$(A \cdot B) \cdot C \text{ or } A \cdot (B \cdot C) \ ?$$

Matrix multiplication is associative (can place parenthesis arbitrary), but not commutative (one cannot change the order of the matrices)

What multiplication order makes fewest (primitive) multiplications ?

a) $(A \cdot B) \cdot C$ $2 \cdot 10 \cdot 50 + 2 \cdot 50 \cdot 20 = 3000$

b) $A \cdot (B \cdot C)$ $10 \cdot 50 \cdot 20 + 2 \cdot 10 \cdot 20 = 10400$

c) Both do the same

d) Don't know

$A = 2 \times 10$ $B = 10 \times 50$ $C = 50 \times 20$

Matrix-chain multiplication

Problem: Find the best placement of parenthesis to multiply n matrices

$$A_1 \cdot A_2 \cdots A_n$$

where A_i is a p_{i-1} x p_i matrix

Note: There are $\Omega(4^n / n^{3/2})$ possible ways to place n pairs of parenthesis (n'th Catalan number)

en.wikipedia.org/wiki/Catalan_number

Matrix-chain multiplication

$m[i, j]$ = minimum number of (primitive) multiplications to compute $A_i \cdots A_j$

$p_{i-1} \times p_k$ $\quad$ $p_k \times p_j$

$(A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$

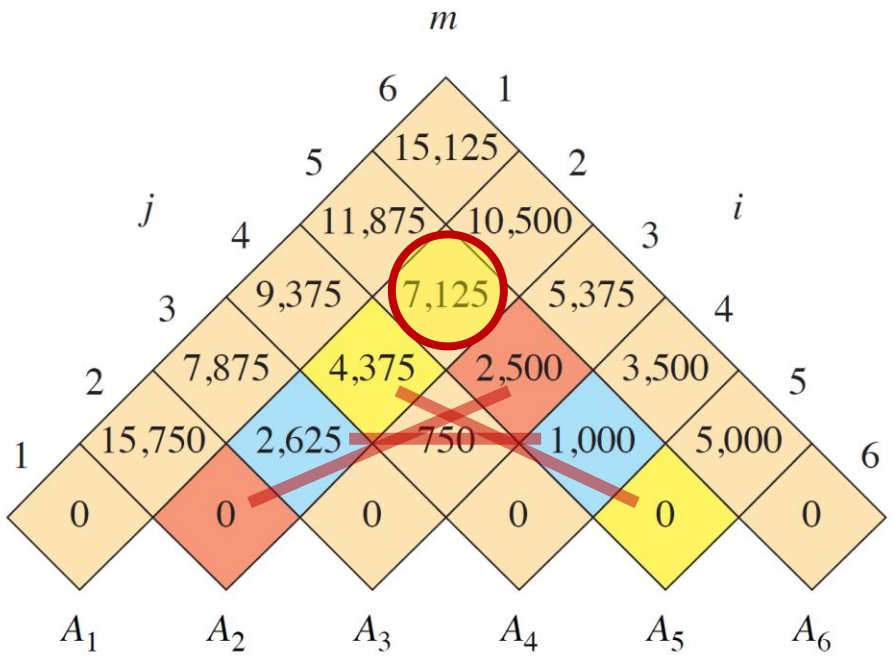
$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j : i \le k < j\} & \text{if } i < j \end{cases}$$

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i == j
2       return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5       q = RECURSIVE-MATRIX-CHAIN(p, i, k)
              + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
              + p_{i−1} p_k p_j
6       if q < m[i, j]
7            m[i, j] = q
8  return m[i, j]
```

Time $\Omega(4^n/n^{3/2})$

Matrix-chain multiplication

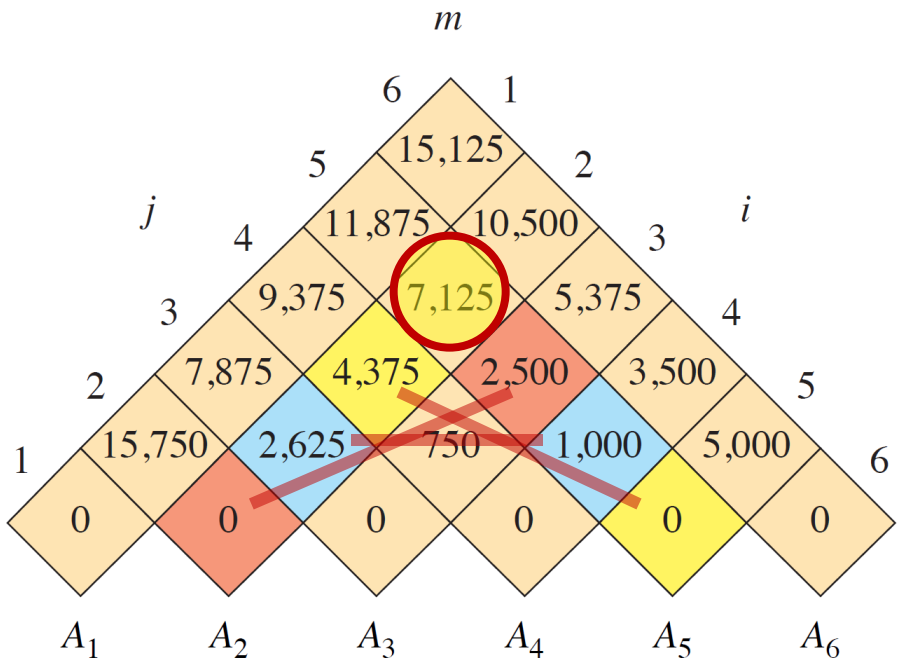
```
MATRIX-CHAIN-ORDER(p, n)
 1  let m[1:n, 1:n] and s[1:n−1, 2:n] be new tables
 2  for i = 1 to n
 3      m[i, i] = 0
 4  for l = 2 to n
 5      for i = 1 to n − l + 1
 6          j = i + l − 1
 7          m[i, j] = ∞
 8          for k = i to j − 1
 9              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
10              if q < m[i, j]
11                  m[i, j] = q
12                  s[i, j] = k
13  return m and s
```

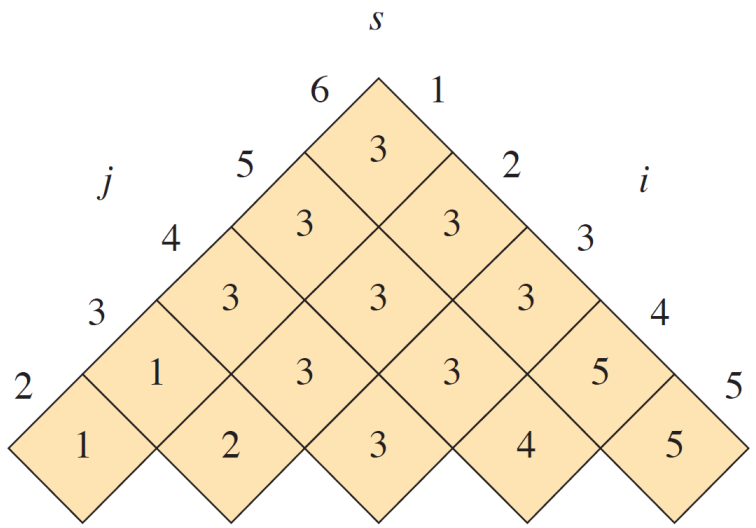


Time $O(n^3)$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j : i \le k < j\} & \text{if } i < j \end{cases}$$

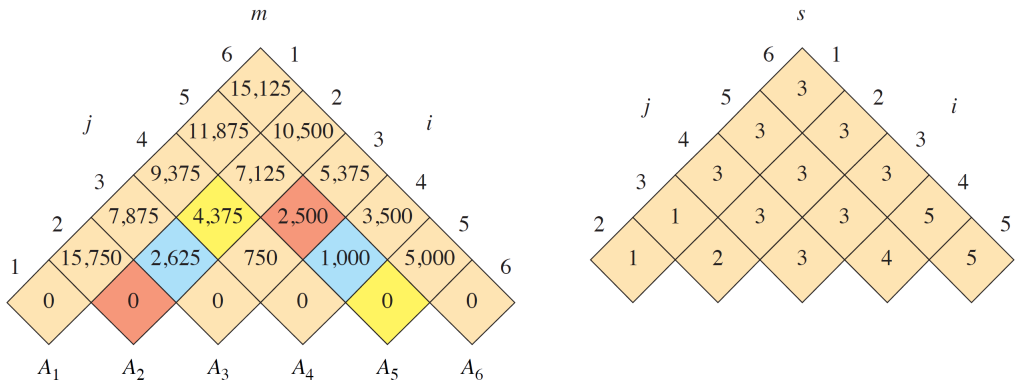
Matrix-chain multiplication





matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

Matrix-chain multiplication

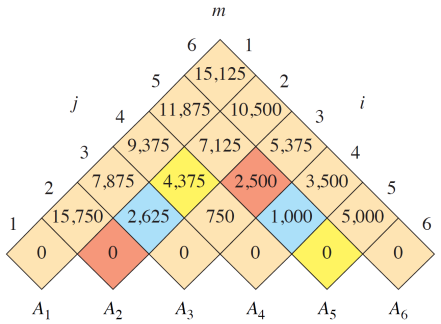


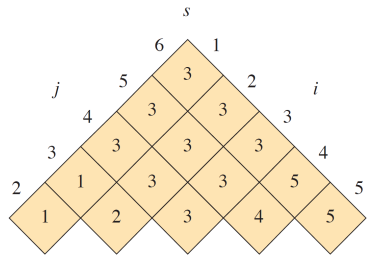

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2        print "A"_i
3  else print "("
4        PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5        PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6        print ")"
```

Time O(n)

What is printed for $i = 1$ and $j = 6$?

```
PRINT-OPTIMAL-PARENS(s, i, j)
1  if i == j
2        print "A"_i
3  else print "("
4        PRINT-OPTIMAL-PARENS(s, i, s[i, j])
5        PRINT-OPTIMAL-PARENS(s, s[i, j] + 1, j)
6        print ")"
```





a) $((A_1A_2)(A_3(A_4A_5)A_6)))$

b) $((A_1((A_2A_3)((A_4A_5)A_6))))$

c) $((A_1(A_2A_3))((A_4A_5)A_6))$

d) $((A_1(A_2\,(A_3\,(A_4(A_5A_6))))))$

e) Don't know

Memoized matrix-chain multiplication

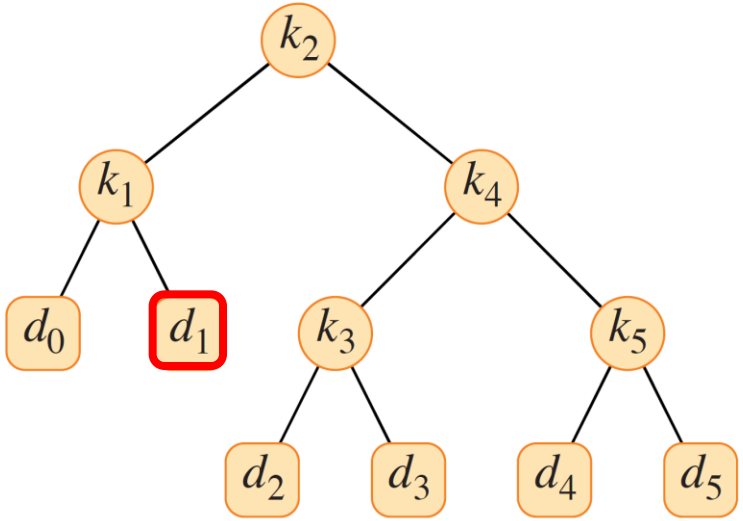
```
MEMOIZED-MATRIX-CHAIN(p, n)
1  let m[1:n, 1:n] be a new table
2  for i = 1 to n
3      for j = i to n
4          m[i, j] = ∞
5  return LOOKUP-CHAIN(m, p, 1, n)
```

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j : i \le k < j\} & \text{if } i < j \end{cases}$$

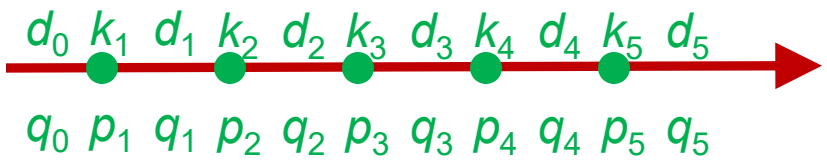
```
LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2      return m[i, j]
3  if i == j
4      m[i, j] = 0
5  else for k = i to j − 1
6          q = LOOKUP-CHAIN(m, p, i, k)
                + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7          if q < m[i, j]
8              m[i, j] = q
9  return m[i, j]
```

Time $O(n^3)$

Optimal binary search trees



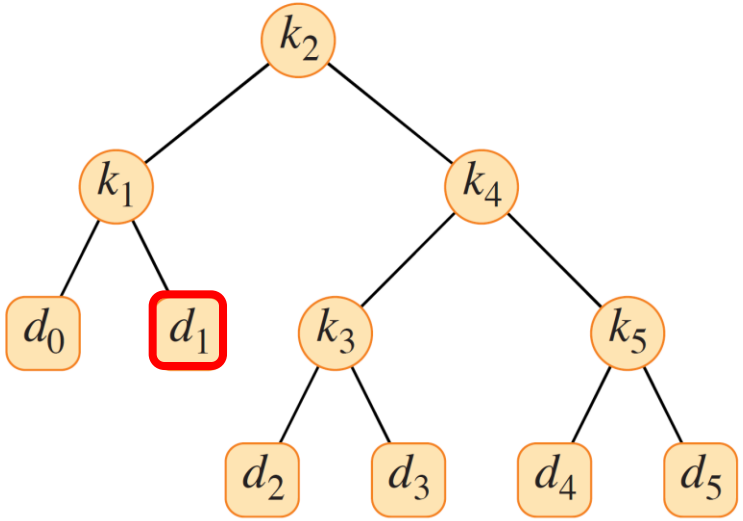
Expected search time 2.80



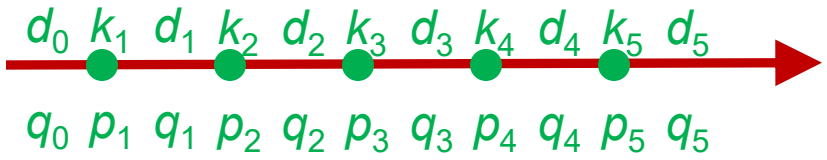
node	depth	probability		contribution
k_1	1	0.15		0.30
k_2	0	0.10		0.10
k_3	2	0.05		0.15
k_4	1	0.10		0.20
k_5	2	0.20		0.60
d_0	2	0.05		0.15
d_1	(2 + 1) x	0.10	=	0.30
d_2	3	0.05		0.20
d_3	3	0.05		0.20
d_4	3	0.05		0.20
d_5	3	0.10		0.40
Total				2.80

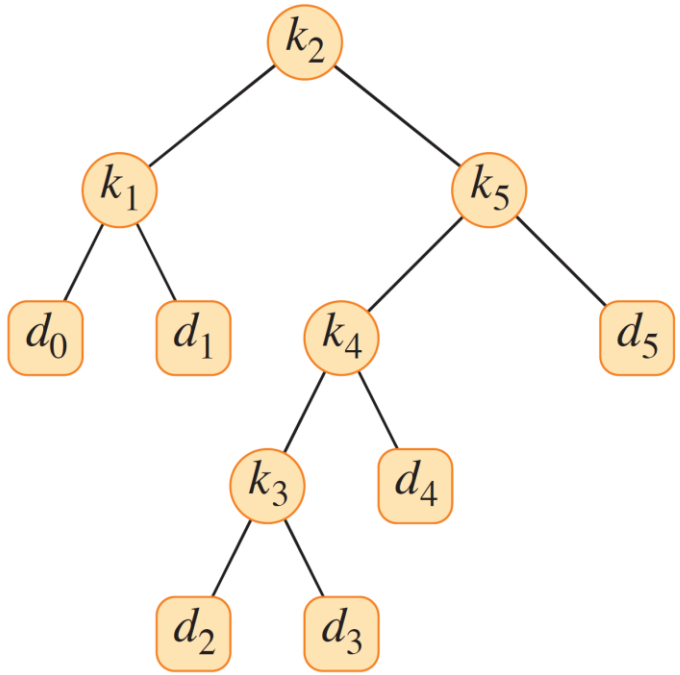
i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal binary search trees



Expected search time 2.80



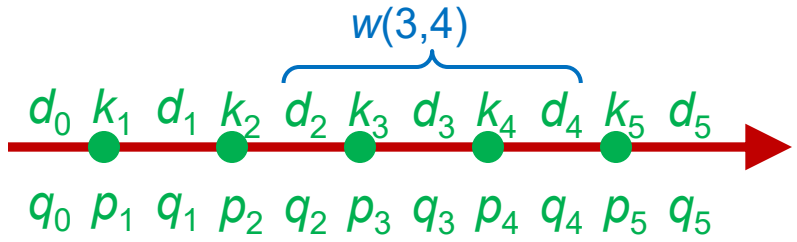


Expected search time 2.75

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimal binary search trees

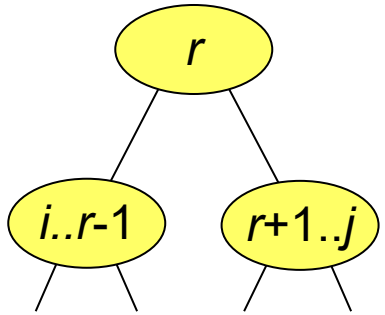
$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \qquad w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$



$$\begin{aligned}\text{E}[\text{search cost in } T] &= \sum_{i=1}^{n}(\text{depth}_T(k_i)+1)\cdot p_i + \sum_{i=0}^{n}(\text{depth}_T(d_i)+1)\cdot q_i \\ &= 1 + \sum_{i=1}^{n}\text{depth}_T(k_i)\cdot p_i + \sum_{i=0}^{n}\text{depth}_T(d_i)\cdot q_i\end{aligned}$$

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min\{e[i,r-1] + e[r+1,j] + w(i,j) : i \le r \le j\} & \text{if } i \le j \end{cases}$$

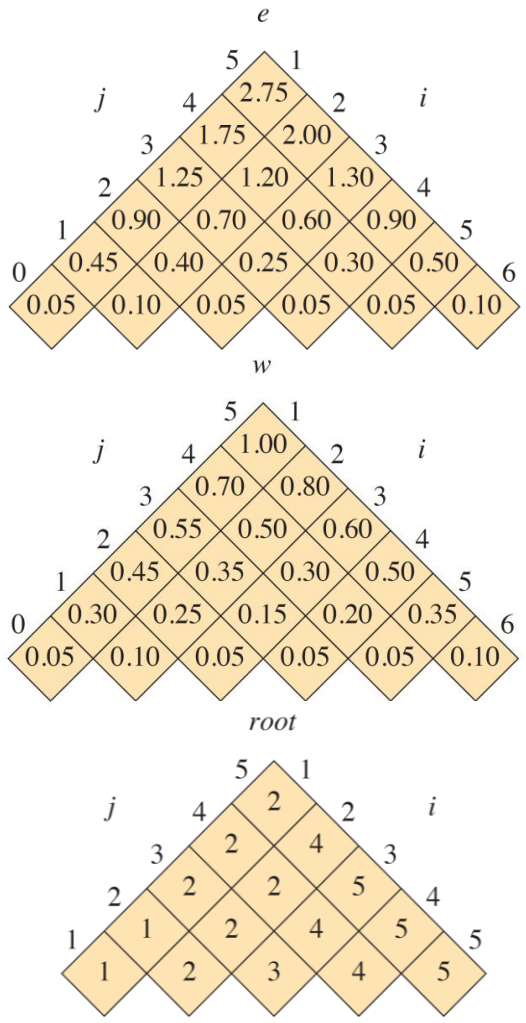
optimal expected time for a search tree containing $k_i,\ldots,k_j$ and $d_{i-1},\ldots,d_j$



Optimal binary search trees

$$e[i,j] = \begin{cases} q_{i-1} & \text{if } j = i-1 \\ \min\{e[i,r-1] + e[r+1,j] + w(i,j) : i \le r \le j\} & \text{if } i \le j \end{cases}$$

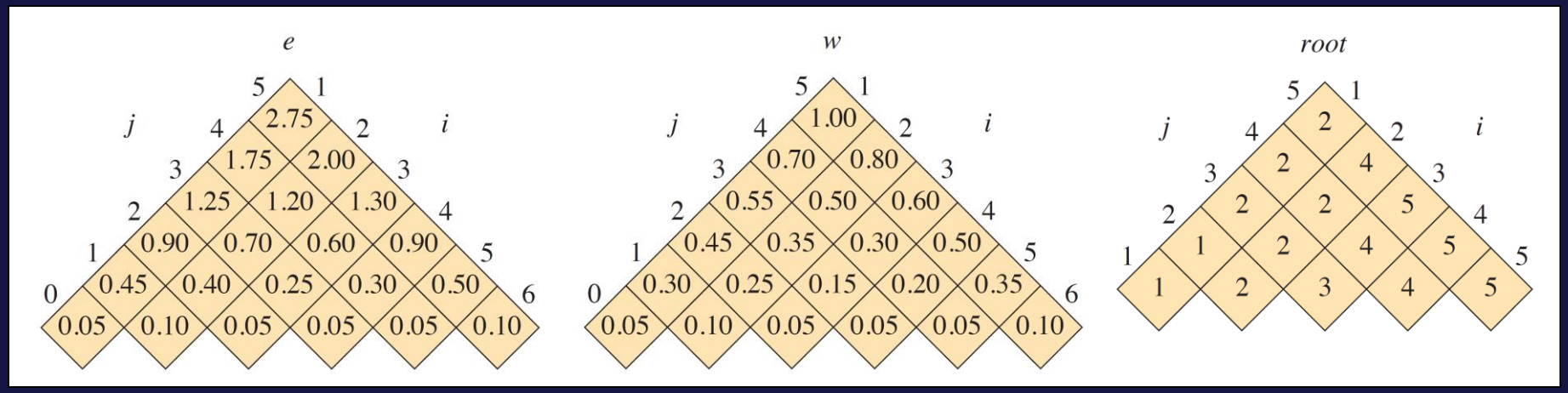
```
OPTIMAL-BST(p, q, n)
 1  let e[1:n+1, 0:n], w[1:n+1, 0:n],
          and root[1:n, 1:n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i-1] = q_{i-1}
 4      w[i, i-1] = q_{i-1}
 5  for l = 1 to n
 6      for i = 1 to n - l + 1
 7          j = i + l - 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j-1] + p_j + q_j
10          for r = i to j
11              t = e[i, r-1] + e[r+1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```



Time $O(n^3)$

Root of an optimal binary search tree ?

a) k_1
b) k_2
c) k_3
d) k_4
e) k_5
f) Don't know



Dynamic programming

- **General algorithm design technique**
  - Works for a long sequence of problems
  - A solution can be computed from solutions to subproblems
- **Recurrence**
- **Examples**
  - Rod cutting
  - Longest common subsequence
  - Printing neatly
  - Matrix-chain multiplication
  - Optimal binary search trees

Algorithms and Data Structures

Greedy algorithms
[CLRS, Chapter 15.1-15.3]

Greedy algorithms

- Problems where a solution can be constructed from the solution to one smaller subproblem

- Subproblem can be identified efficiently

(simpler than divide and conquer
and dynamic programming)

Longest common subsequence ?

ABABGACBABAD

BACABAABABC

a) BACBABA

b) BABBAB

c) BAABABA

d) BACABAB

e) Don't know

Answer not unique

Easy to check subsequence (greedy)

Exponential many possible subsequences

Is *BACBABA* a subsequence of *ABABGACBABAD* ?

Possible occurrence *ABABGACBABAD*

Observation If there is an occurrence, then there also exists an occurrence matching the first ***B*** in the string (greedy-choice property)

ABABABGACBABAD

Find recursively ***ACBABA***

Time O(*n*)

Solution found by the greedy algorithm ?

a) *AB<u>A</u>BG<u>ACBABA</u>D*

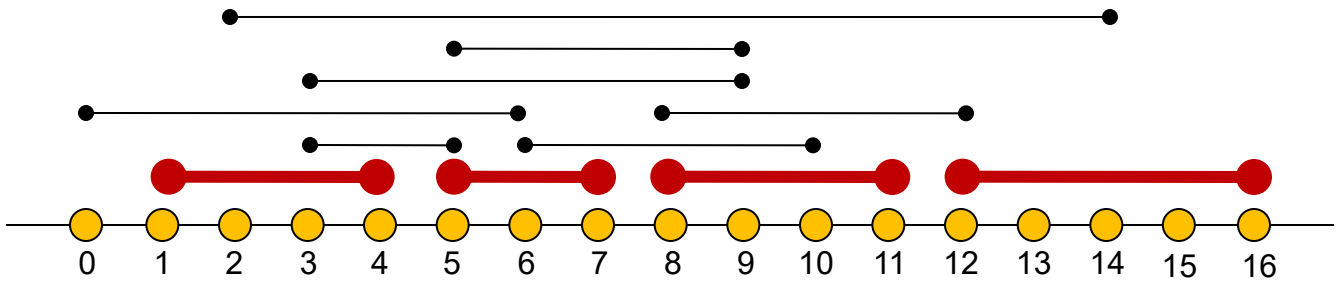
b) *A<u>BA</u>BGA<u>CBABA</u>D*

c) *ABA<u>B</u>G<u>ACBABA</u>D*

d) Don't know

Activity selection

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	7	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

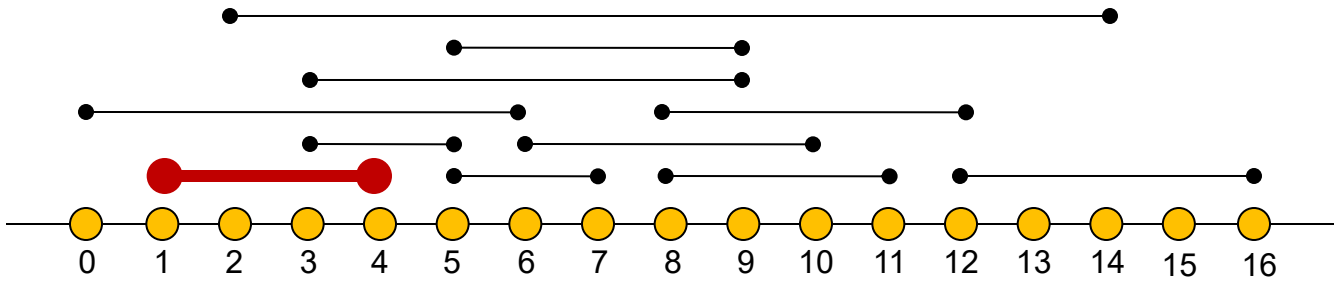


Input n activities with start time s_i and finishing time f_i

Output Maximal subset of non-overlapping activities

Activity selection

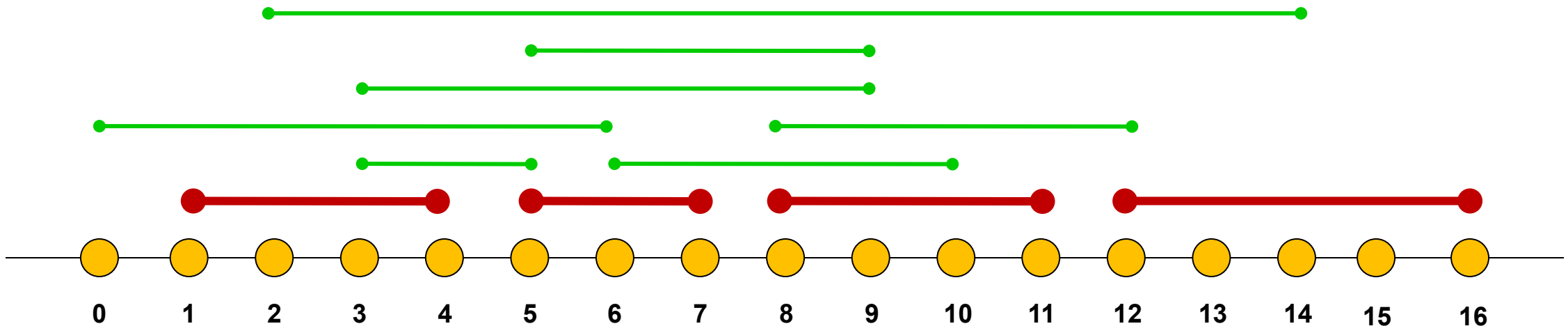
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	7	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16



Observation There always exists a maximal solution containing an activity with earliest finishing time (greed-choice property)

Preprocessing Sort activities by finishing time

Greedy deletions



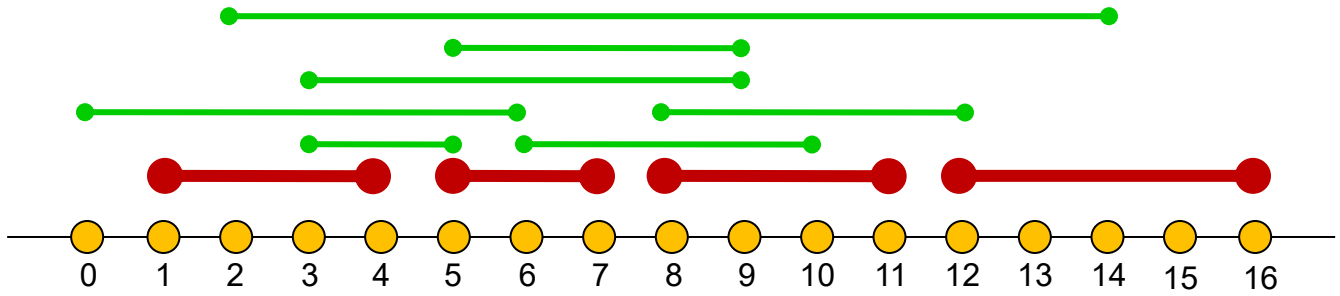
Activity selection

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	7	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

← sorted

GREEDY-ACTIVITY-SELECTOR(s, f, n)

```
1  A = {a_1}
2  k = 1
3  for m = 2 to n
4      if s[m] ≥ f[k]
5          A = A ∪ {a_m}
6          k = m
7  return A
```



Time $O(n \cdot \log n)$

Huffman codes

ASCII table

$83_{10} = 53_{16} = 101\ 0011_2$

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

The string ”AU” as binary ?

a) 000100101011101
b) 110000111101 01
c) 100000110101 01
d) 000111001011 10
e) Don't know

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

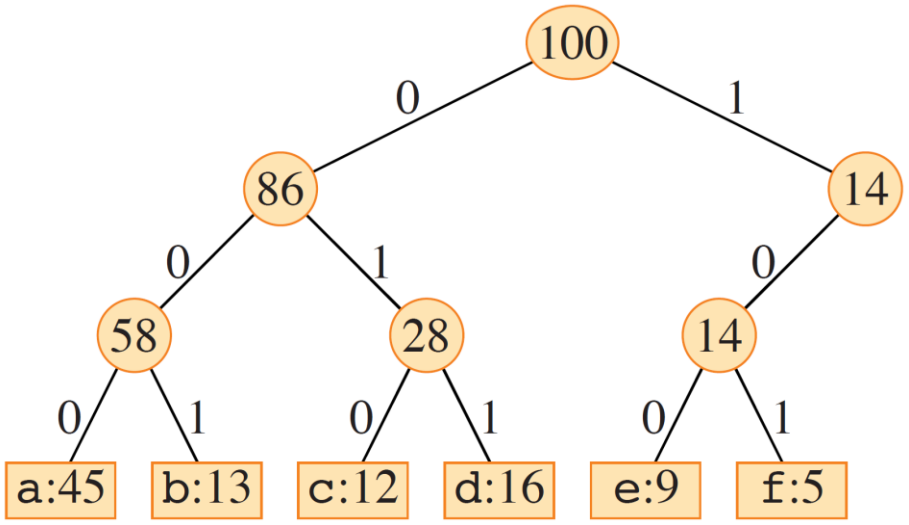
Compression

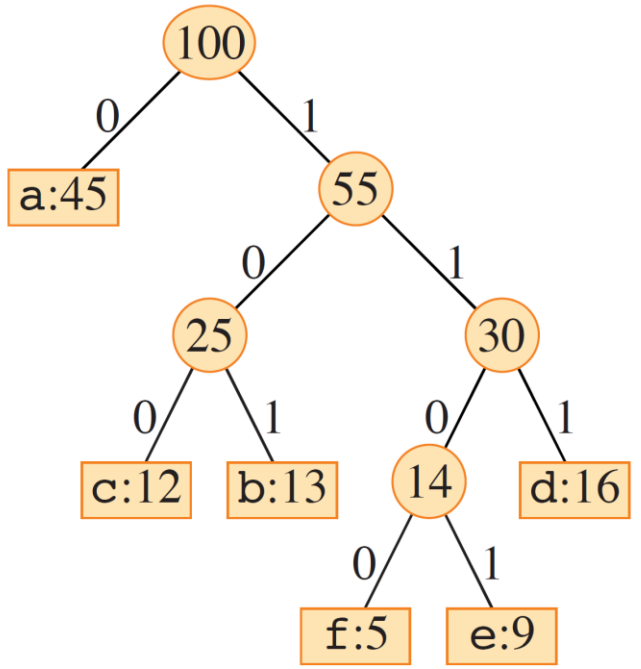
Given the frequences of the occurrences of the symbols in the input, replace input symbols with shorter bit strings (fixed-length or variable-length codes)

	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

Note Variable-length codes must be prefix free

Fixed-length vs variable-length codes





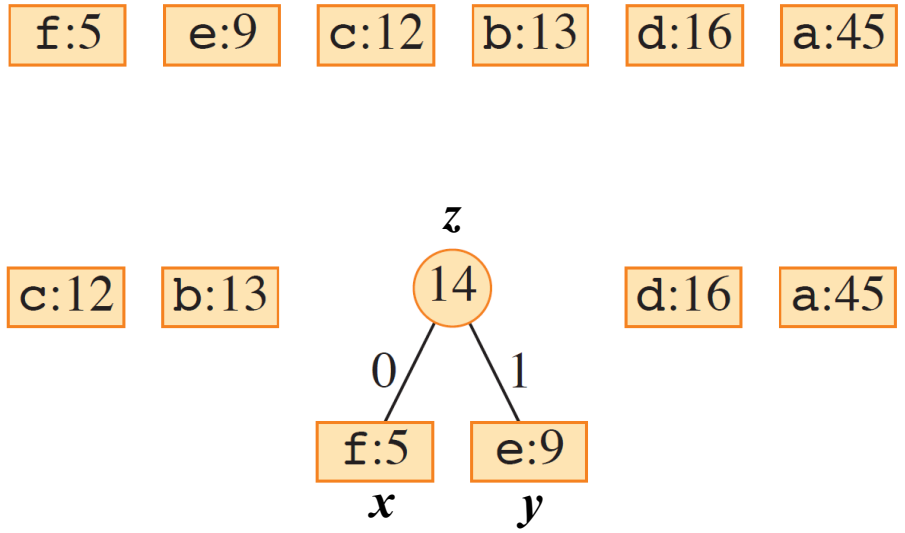
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

3 * (45+13+12+16+9+5) = 300 bits

1*45+3*(13+12+16)+4*(9+5) = 224 bits

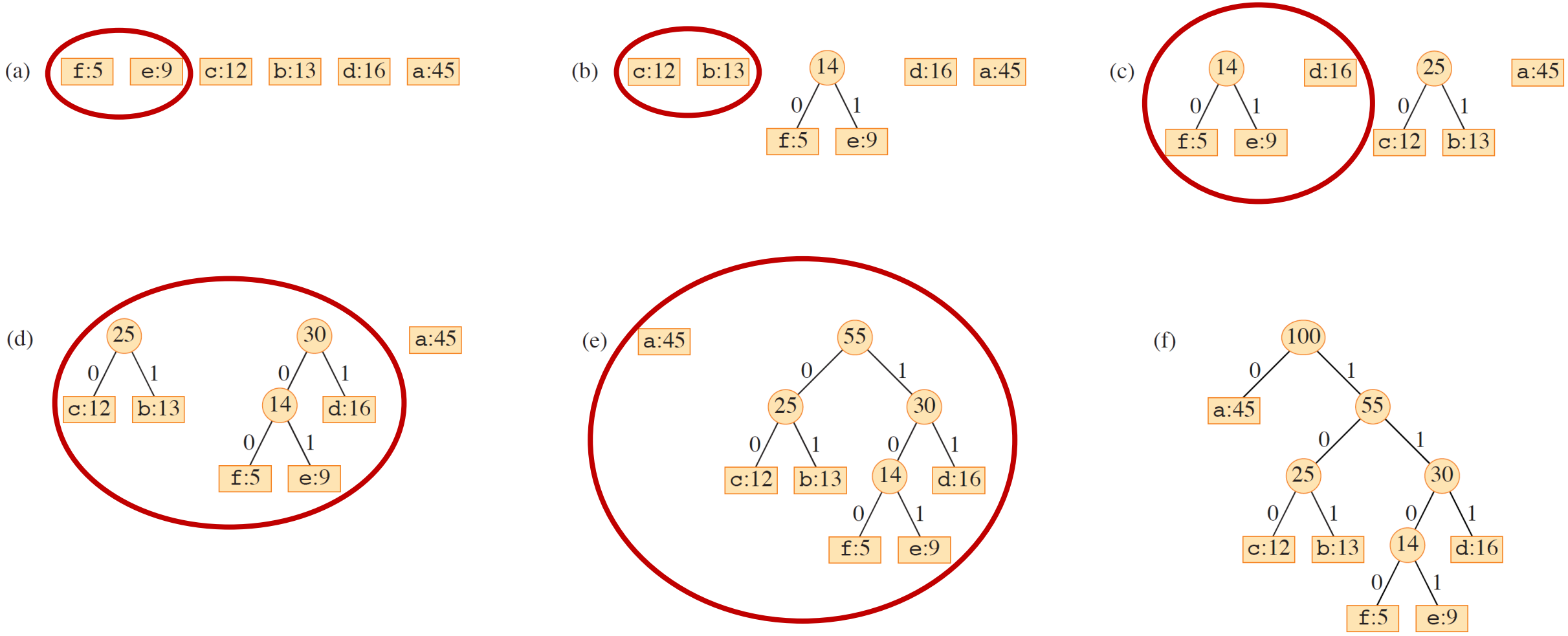
Huffman codes

```
HUFFMAN(C)
 1  n = |C|
 2  Q = C
 3  for i = 1 to n − 1
 4      allocate a new node z
 5      x = EXTRACT-MIN(Q)
 6      y = EXTRACT-MIN(Q)
 7      z.left = x
 8      z.right = y
 9      z.freq = x.freq + y.freq
10      INSERT(Q, z)
11  return EXTRACT-MIN(Q)
```



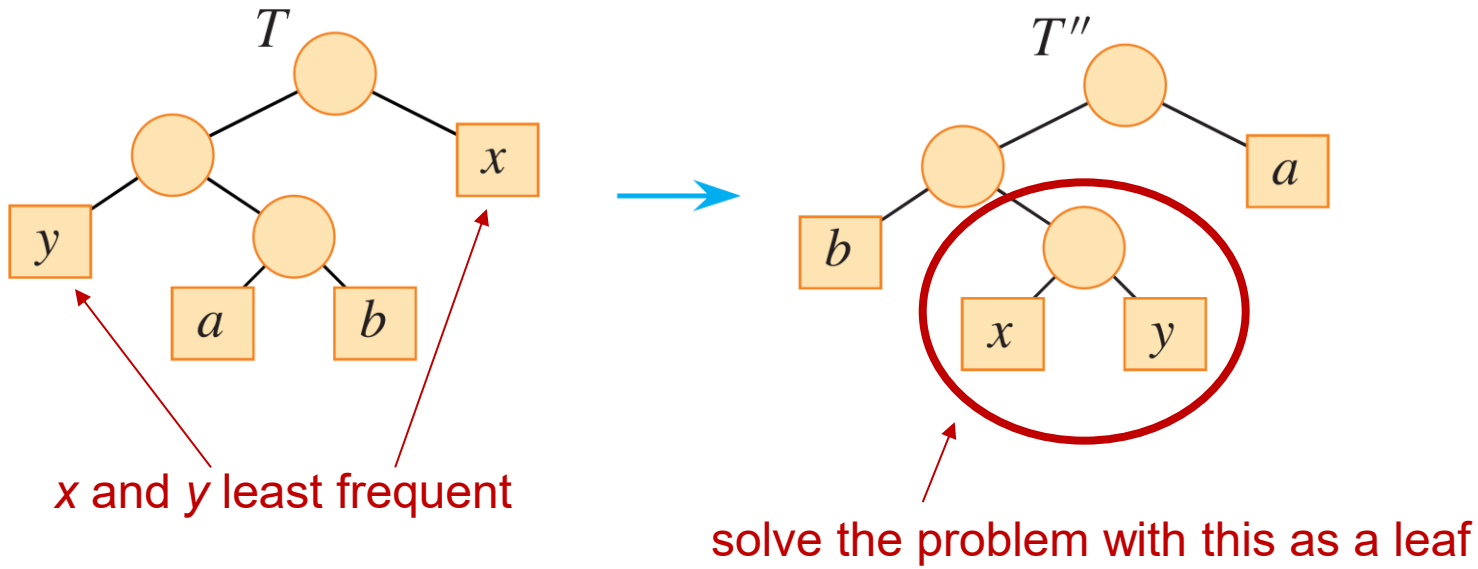
Time $O(n \cdot \log n)$

Huffman codes



Correctness of Huffman codes

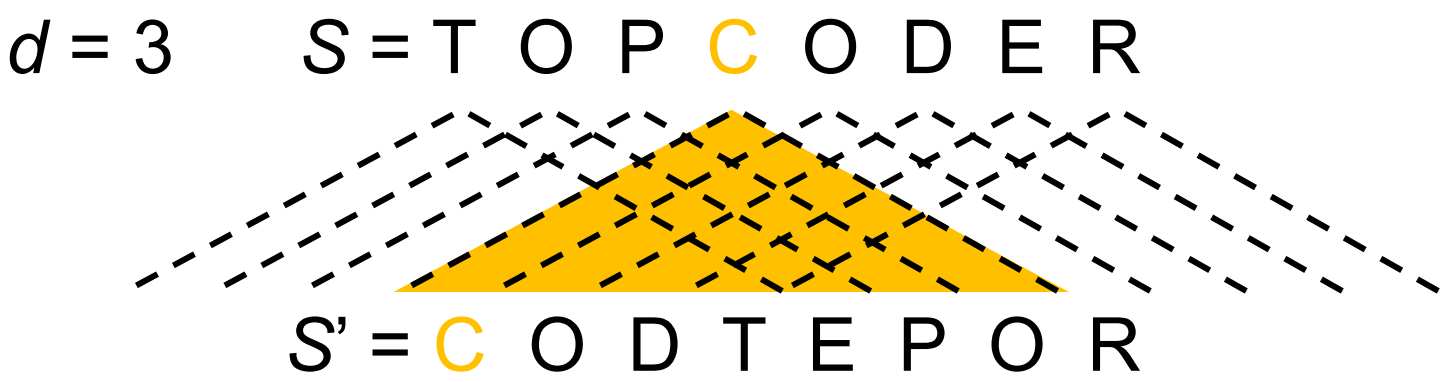
Theorem There always exist an optimal prefix code where the two least frequent symbols (x and y) have the same code length and are identical except for the last bit



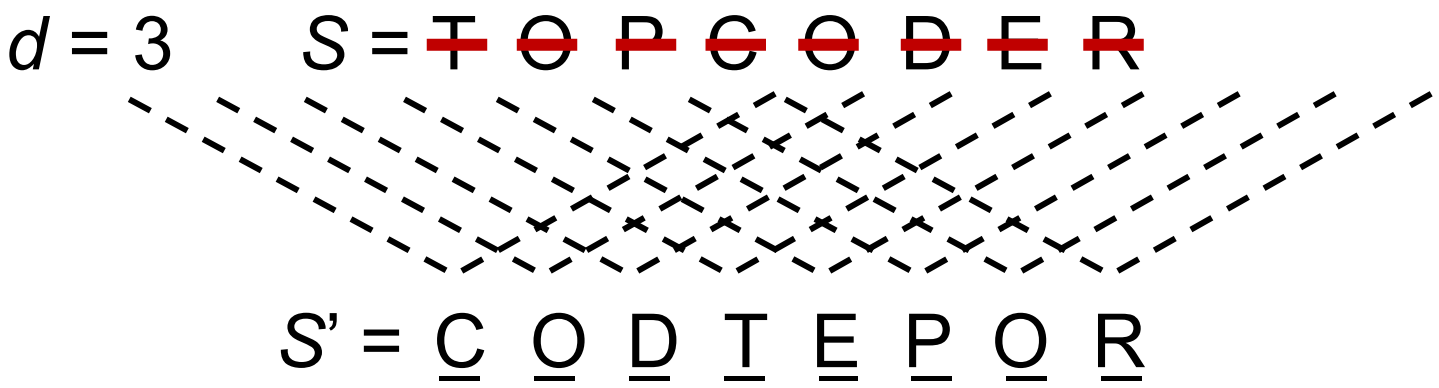
TopCoder Open 2014 – Round 1A (500 point)

Input A string *S*, distance *d*

Output A string *S*' = lexicographically smallest permutation of *S*, where no character is moved more than *d* positions

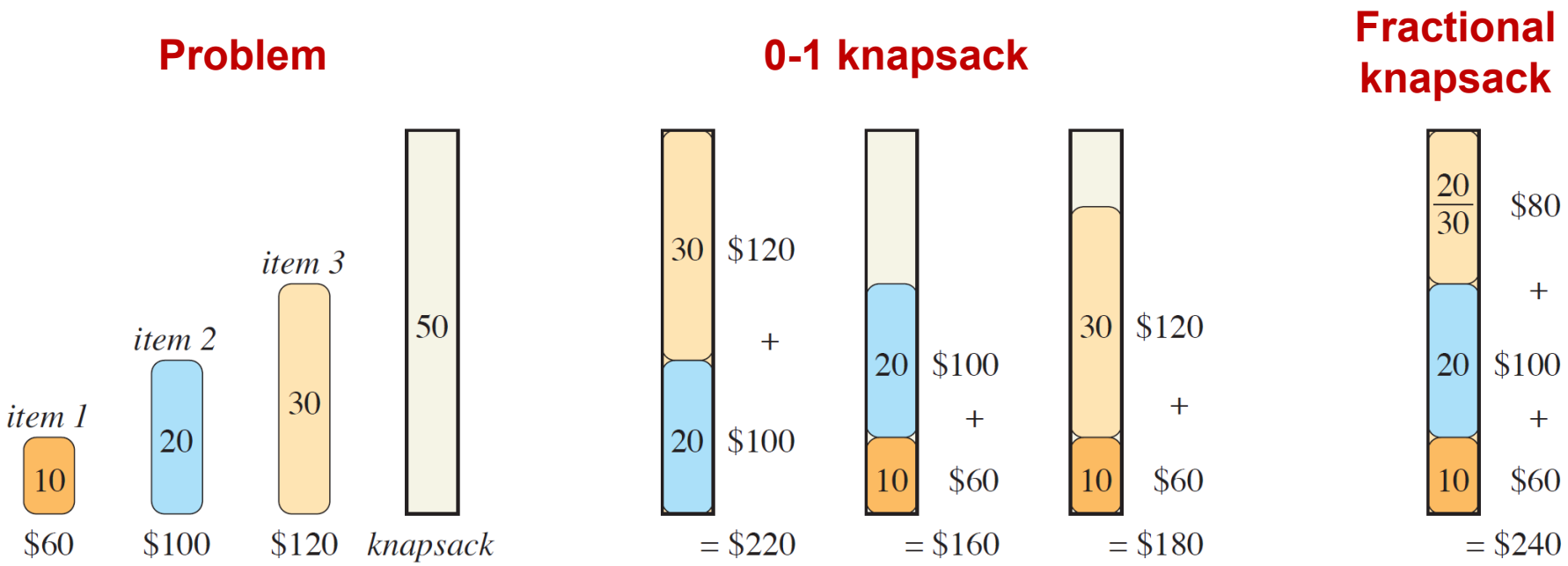


Greedy solution



Algorithm Construct *S'* from left to right
- take the character furthest to the left in the window, if not used
- otherwise take lexicographicaly smallest symbol in window (leftmost is multiple)

Dynamic programming vs greedy



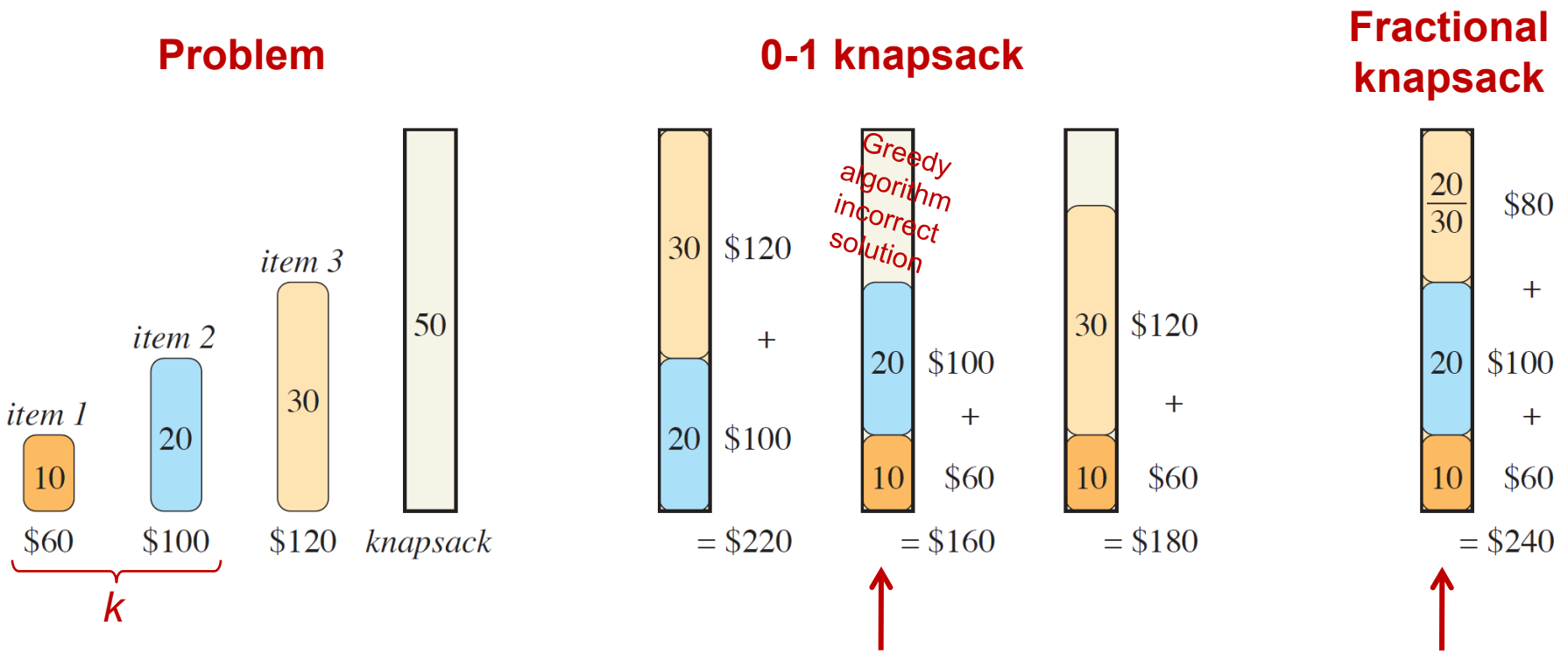
0-1 knapsack – maximum value for volume 11 ?

Volume	2	3	4	5
Value	3	5	4	6

a) 10
b) 11
c) 12
d) 13
e) 14
f) 15
g) Don't know

A solution with maximum value is not necessarily a solution with maximum volume

Dynamic programming vs greedy



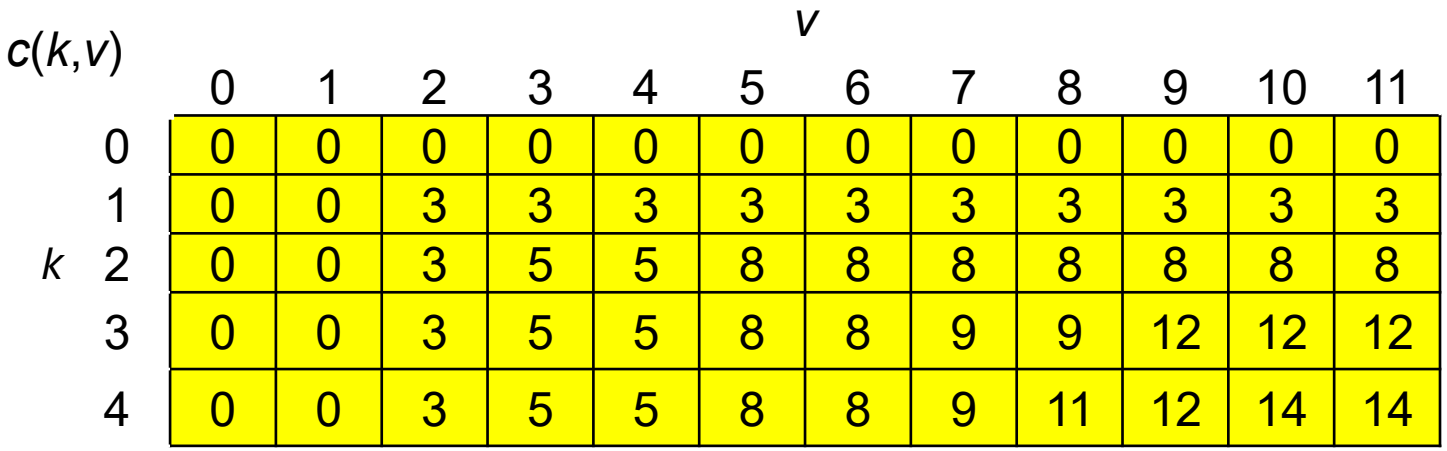
$c(k,v)$ = best value among k first items for volume v

$$= \begin{cases} 0 & \text{if } k = 0 \\ c(k-1, v) & \text{if } \text{volume}_k > v \\ \max\{c(k-1, v), \text{value}_k + c(k-1, v-\text{volume}_k)\} & \text{if } \text{volume}_k \le v \end{cases}$$

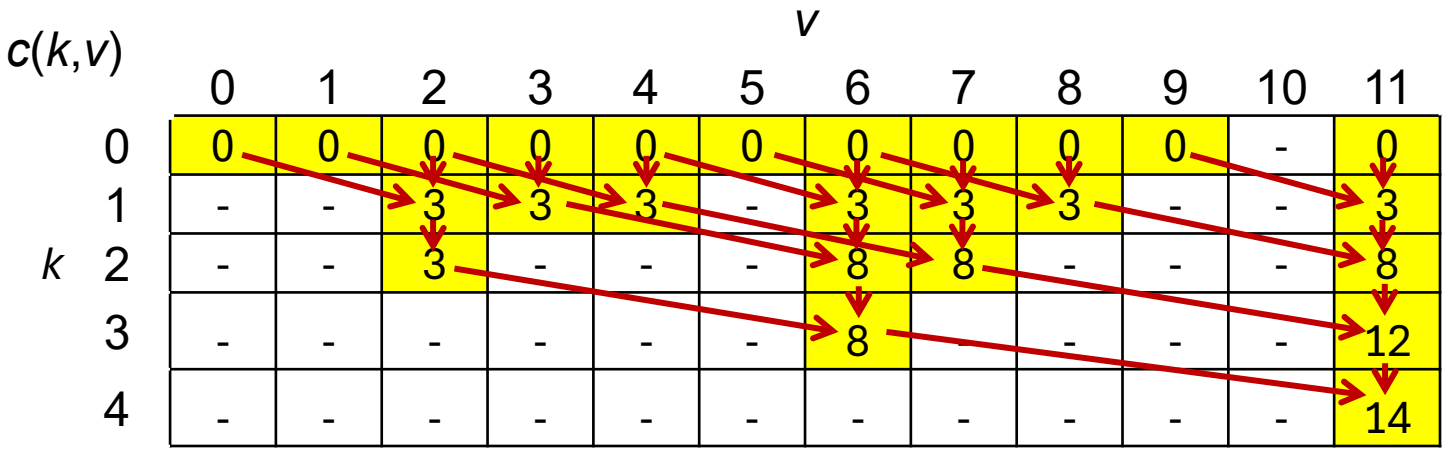
Greedily fill by decreasing value/volume

0-1-knapsack

Volume	2	3	4	5
Value	3	5	4	6



Systematic tabulation

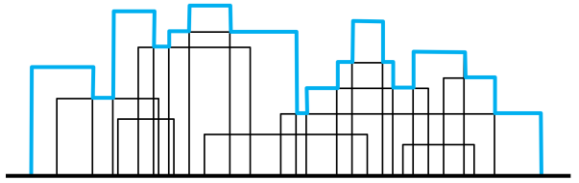


Recursion + memoization

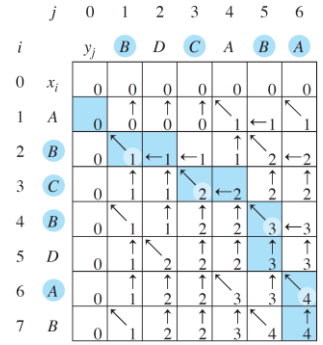
General algorithm design techniques

- **Divide and conquer**
  - Disjoint subproblems

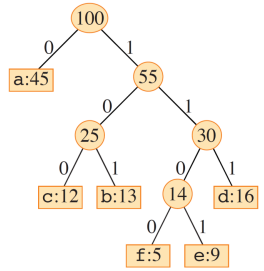
$$T(n) = \begin{cases} c & \text{if } n \le d \\ a \cdot T(n/b) + c \cdot n^p & \text{if } n > d \end{cases}$$

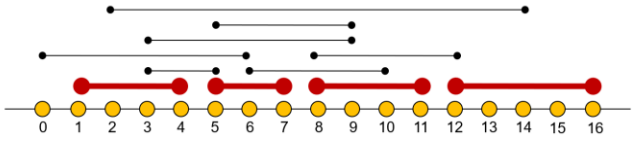


- **Dynamic programming**
  - Subproblems overlap
  - Remember already computed results for subproblems
  - Systematic tabulation or recursion with memoization



- **Greedy algorithms**
  - Only one subproblem

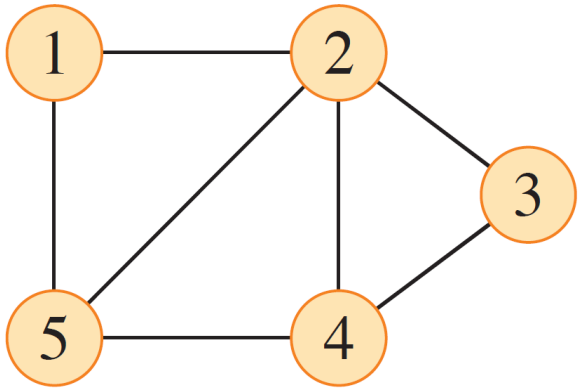




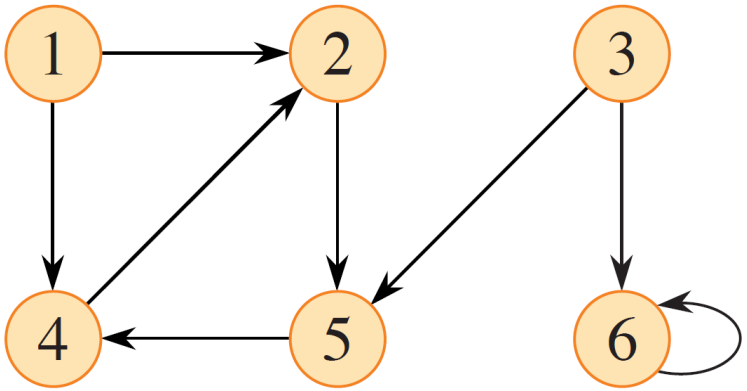
Algorithms and Data Structures

Graph representations, breadth first search (BFS)
[CLRS, Chapter 20.1-20.2]

Graphs



Undirected graph



Directed graph (a.k.a. digraph)

$G = (V, E)$ graph with nodes V and edges E

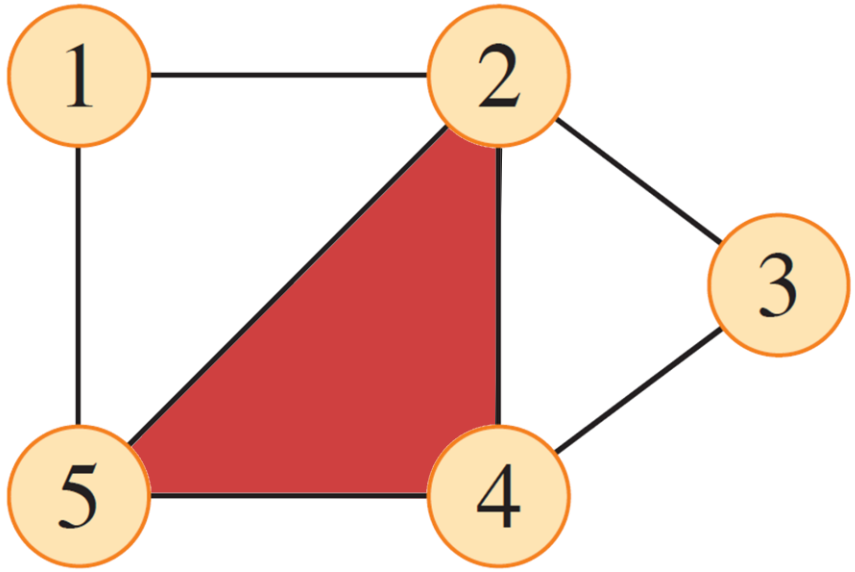
E : $\{u, v\}$ an undirected edge between u and v

(u, v) a directed edge from u to v

$n = |V|$ = # nodes

$m = |E|$ = # edges (connections between nodes)

Planar graphs – Euler's formula



$|V| = 5$
$|E| = 7$
faces = 4

For a connected planar graph:

Euler's formula: $|V| - |E| + \#\text{ faces} = 2$

Corollary: $|E| \leq 3|V| - 6$ (for $|V| \geq 3$, no self-loops, no parallel edges)

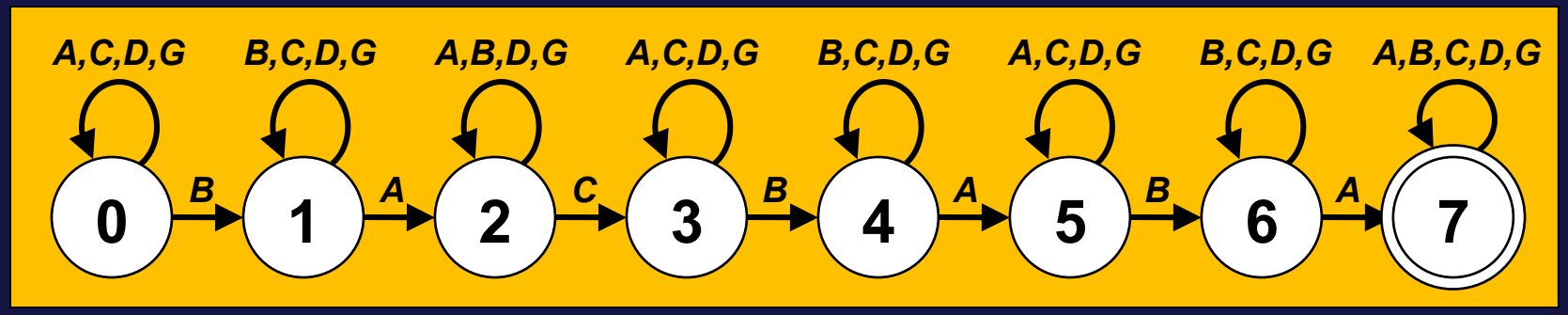
Solution found by the greedy algorithm ?

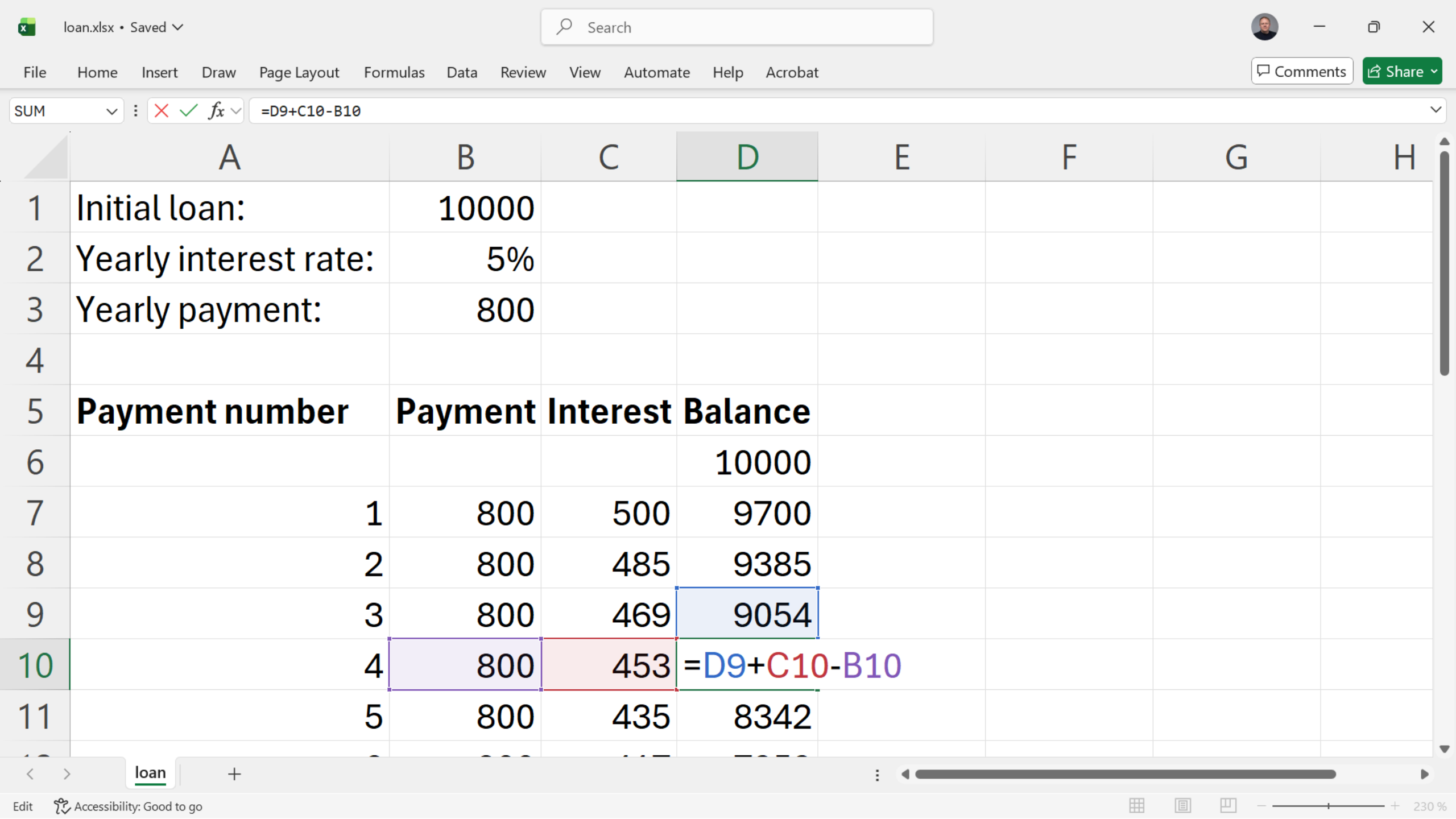
a) *ABABGACBABAD*

b) *ABABGACBABAD*

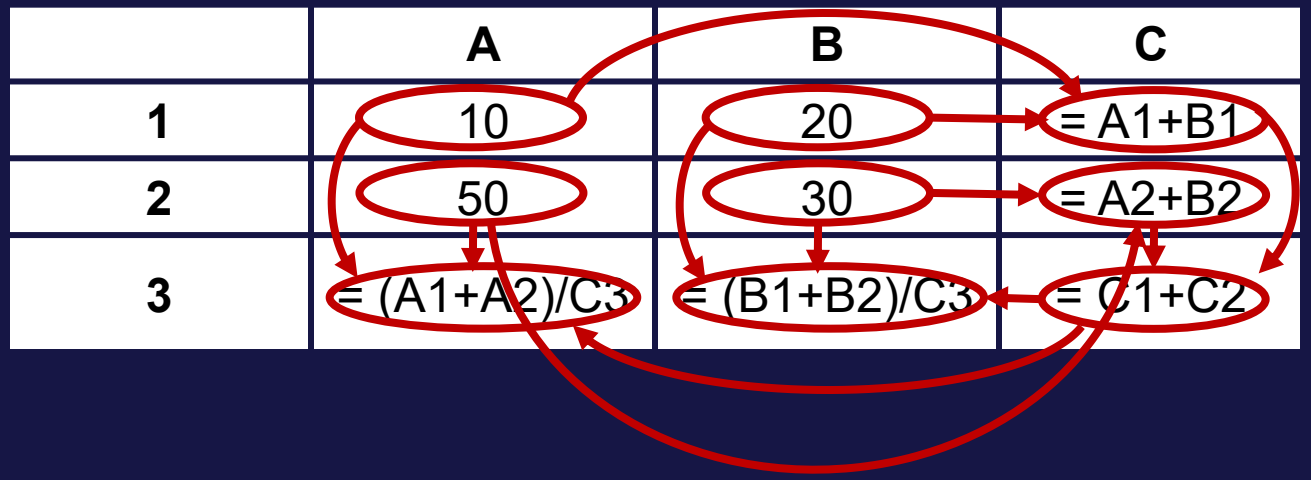
c) *ABABGACBABAD*

d) Don't know




loan.xlsx • Saved
Search
File Home Insert Draw Page Layout Formulas Data Review View Automate Help Acrobat
Comments
Share
SUM
=D9+C10-B10
Initial loan: 10000
Yearly interest rate: 5%
Yearly payment: 800
Payment number Payment Interest Balance
10000
1 800 500 9700
2 800 485 9385
3 800 469 9054
4 800 453 =D9+C10-B10
5 800 435 8342
loan
Edit
Accessibility: Good to go

Which computation order ?



a) C1 C2 A3 B3 C3

b) A3 B3 C2 C1 C3

c) C2 C1 C3 B3 A3

d) Don't know

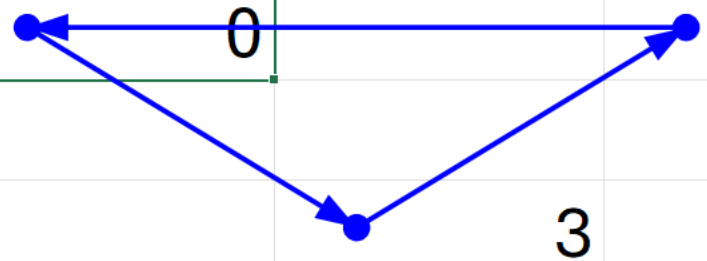
Book1 - Excel

Search

File Home Insert Draw Page Layout Formulas Data Review View Automate Help Acrobat

Comments Share

A1 =B3+1



Microsoft Excel

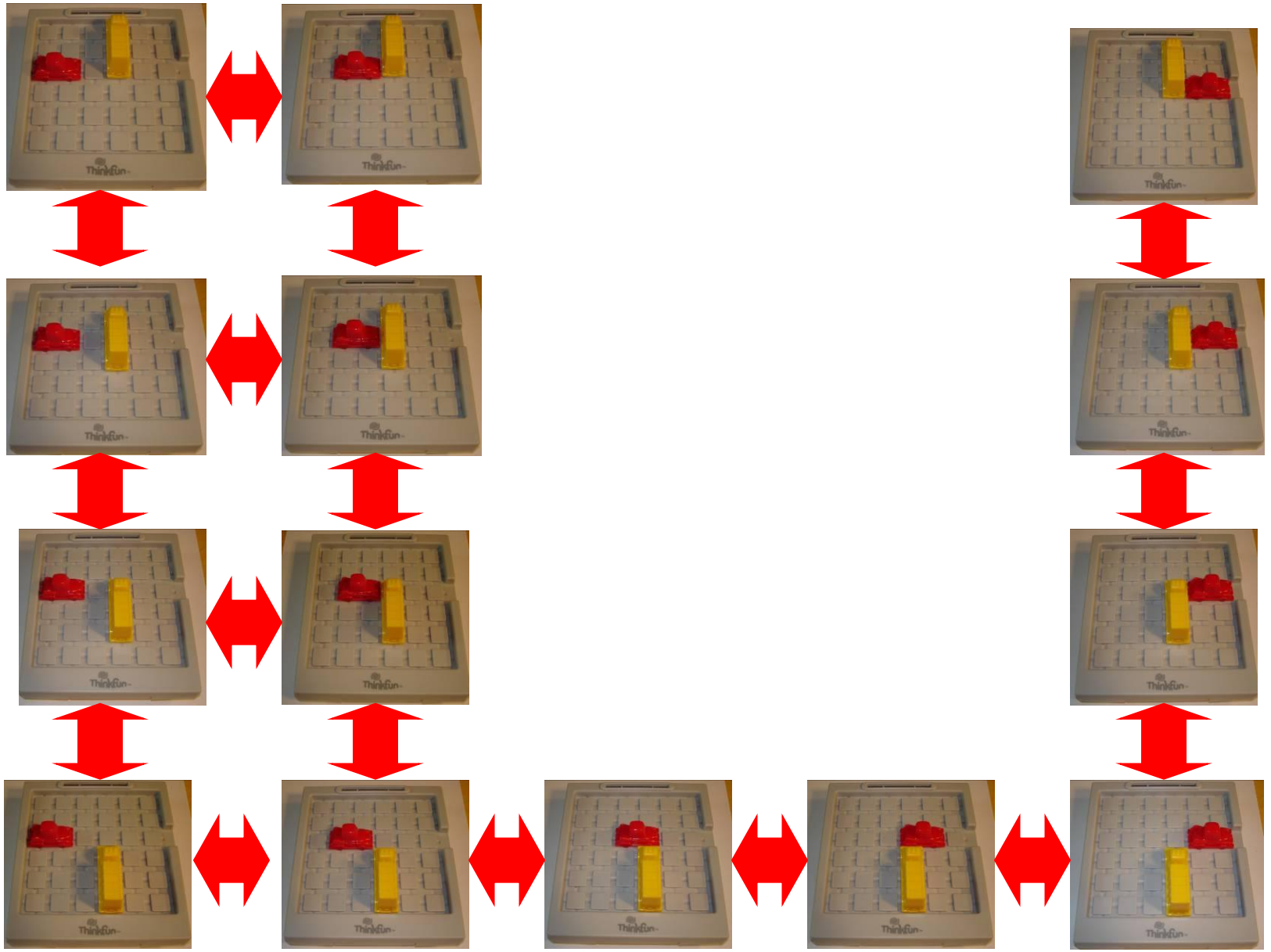
There are one or more circular references where a formula refers to its own cell either directly or indirectly. This might cause them to calculate incorrectly.

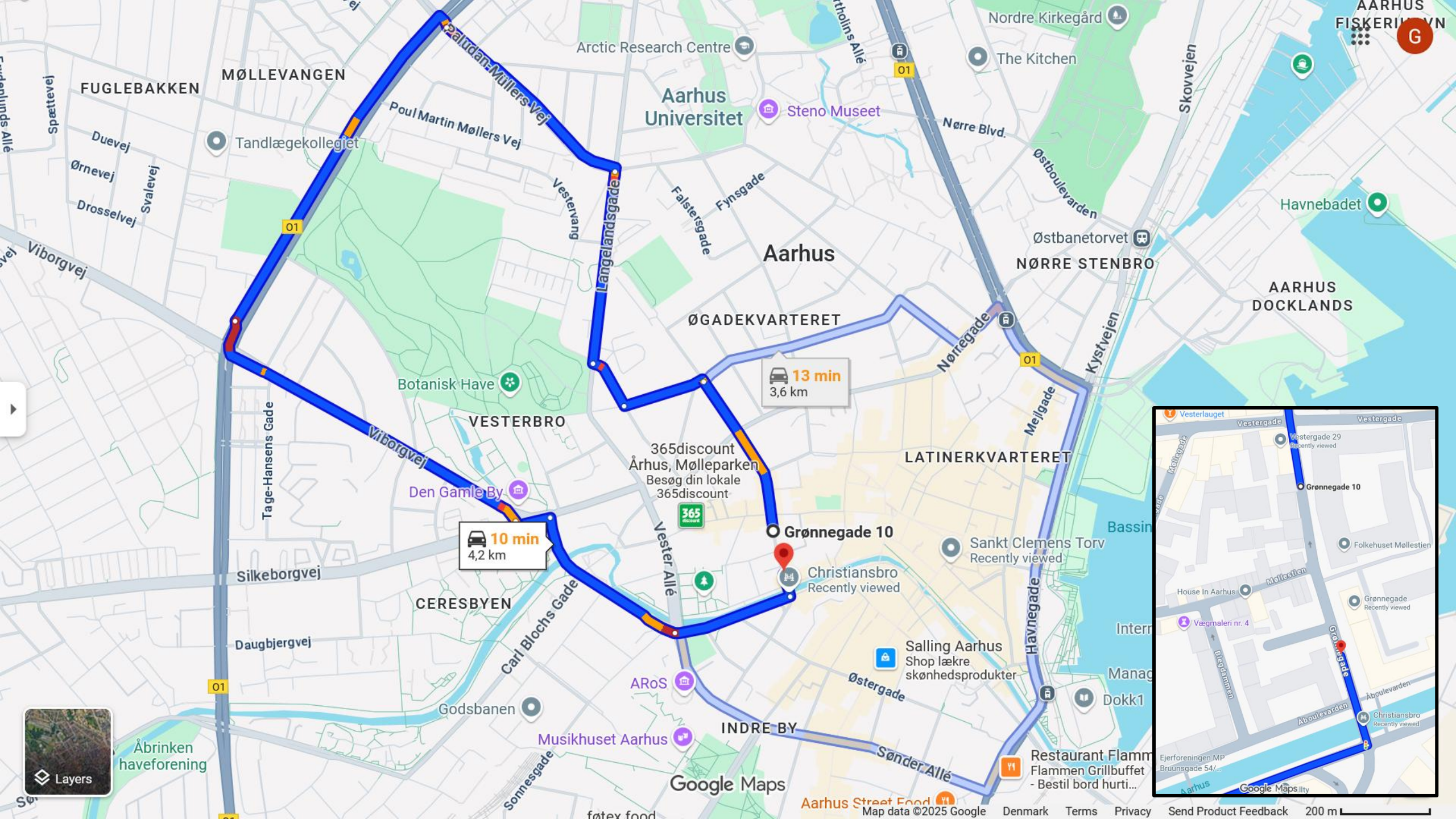
Try removing or changing these references, or moving the formulas to different cells.

OK Help

Sheet1

Ready Circular References: B3 Accessibility: Good to go 257 %

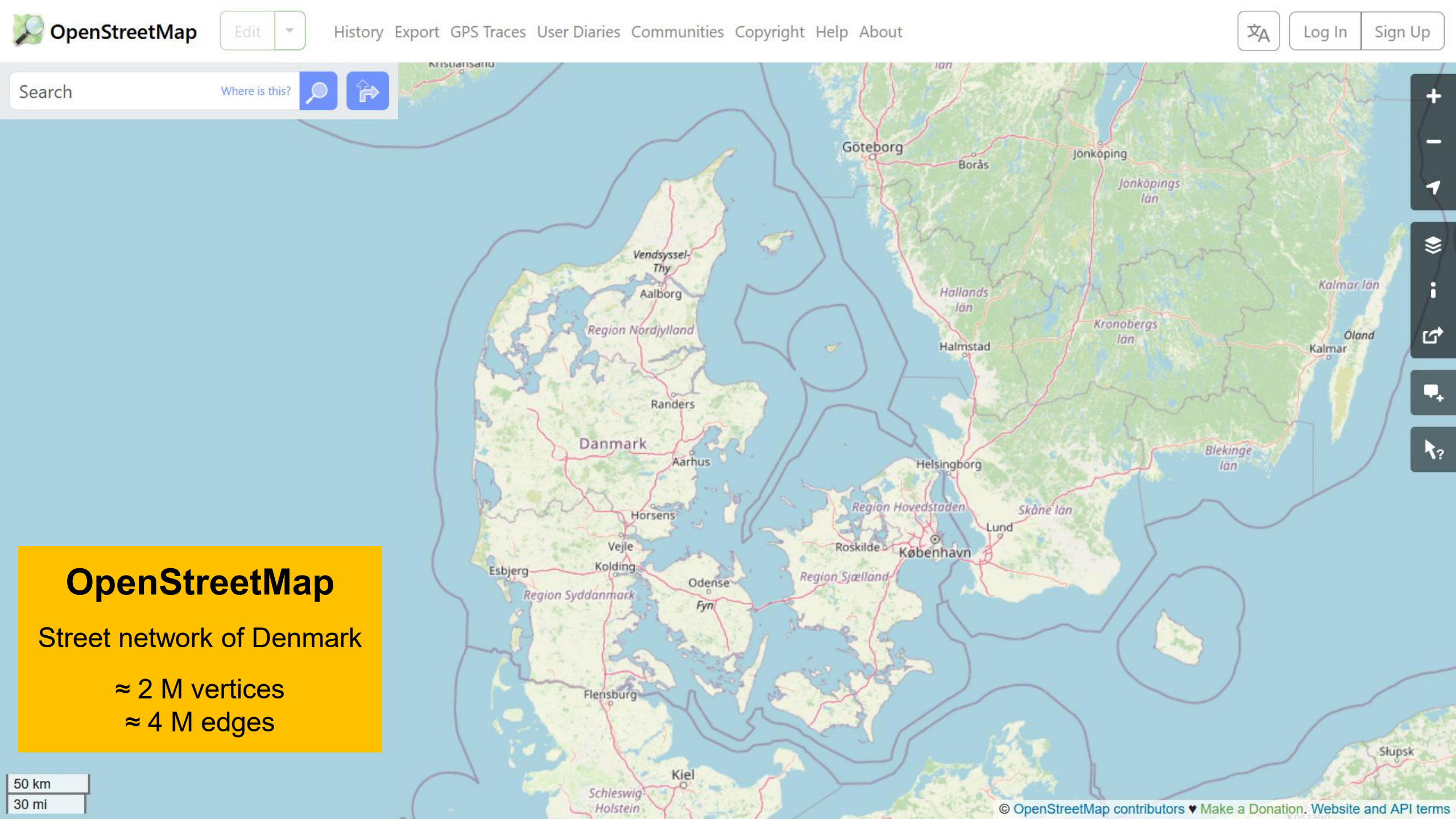



Poul Martin Møllers Vej
Vestervang
Langelandsgade
Falstersgade
Fynsgade
Nørre Blvd.
Østboulevarden
Skovvejen
Viborgvej
Tage-Hansens Gade
Vester Allé
Carl Blochs Gade
Nørregade
Kystvejen
Mejlgade
Havnegade
Østergade
Sønder Allé
Sonnesgade
Daugbjergvej
Silkeborgvej
Spættevej
Duevej
Ørnevej
Drosselvej
Svalevej
Tandlægekollegiet
Arctic Research Centre
Aarhus Universitet
Steno Museet
Nordre Kirkegård
The Kitchen
Havnebadet
Østbanetorvet
Botanisk Have
Den Gamle By
365discount Århus, Mølleparken
Besøg din lokale 365discount
Grønnegade 10
Christiansbro
Recently viewed
Sankt Clemens Torv
Salling Aarhus
Shop lækre skønhedsprodukter
Dokk1
ARoS
Godsbanen
Musikhuset Aarhus
Åbrinken haveforening
Restaurant Flamm
Flammen Grillbuffet - Bestil bord hurti...
Aarhus Street Food
13 min
3,6 km
10 min
4,2 km
FUGLEBAKKEN
MØLLEVANGEN
VESTERBRO
ØGADEKVARTERET
Aarhus
NØRRE STENBRO
AARHUS DOCKLANDS
LATINERKVARTERET
CERESBYEN
INDRE BY
Layers
Google Maps
Map data ©2025 Google
Denmark
Terms
Privacy
Send Product Feedback
200 m

How many nodes are required to represent a road crossing ?

a) 1
b) 2
c) 4
d) 5
e) 8
f) 9
g) 12
h) Don't know

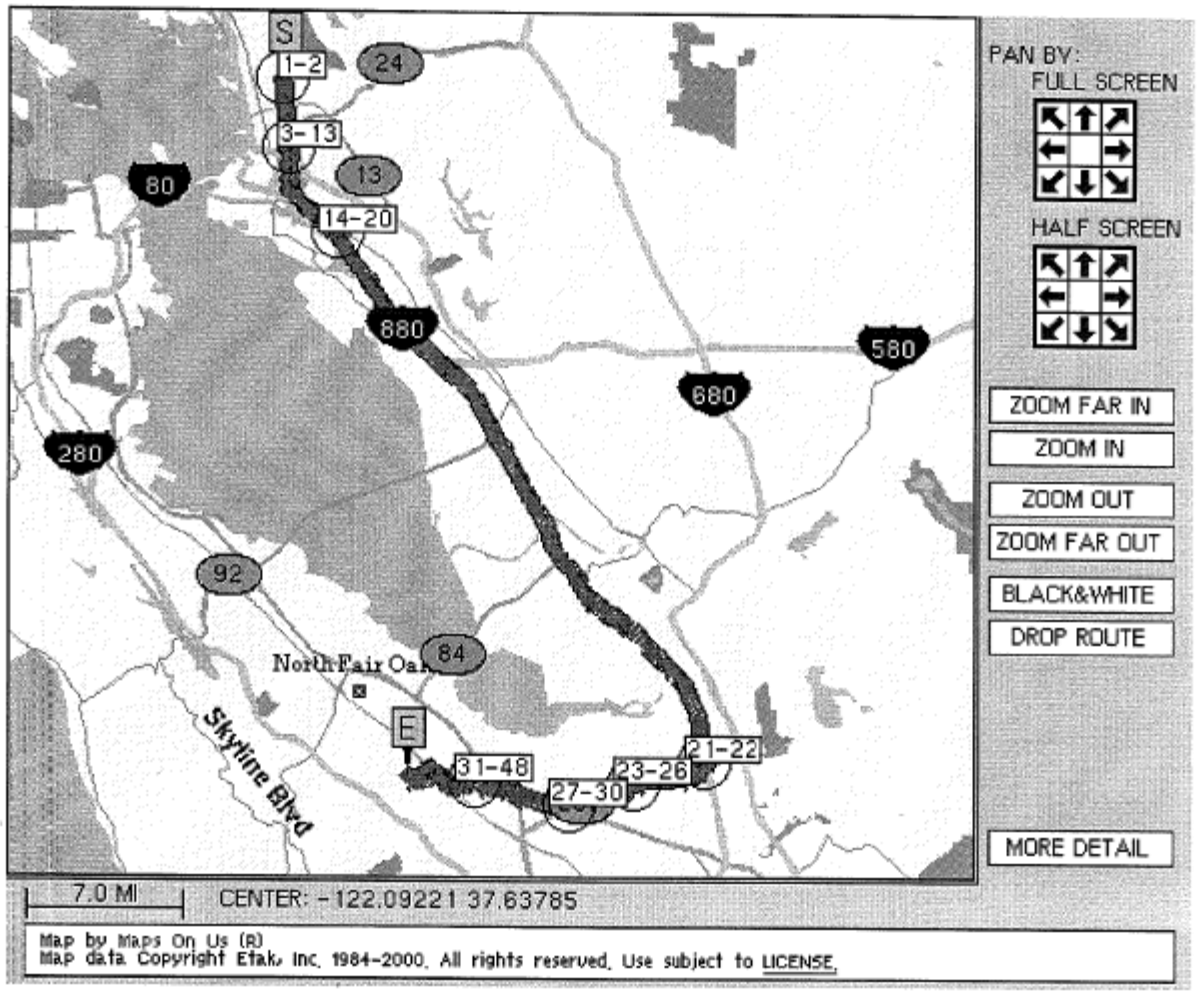
old picture of Vesterbro Torv



OpenStreetMap

Street network of Denmark

≈ 2 M vertices
≈ 4 M edges



"However, because of the size of the routing data, we have to use heuristics when planning routes. As a result, sometimes a Favor Highways route will be slightly faster than the Fastest route."

— MapsOnUs

REJSEPLANEN

Journey Planner | Departureboard | Try rejseplanen.dk beta

Find connection: in Denmark

Skagen St.

Rødby Færge

Departure 10.11 08:30

Options

Find

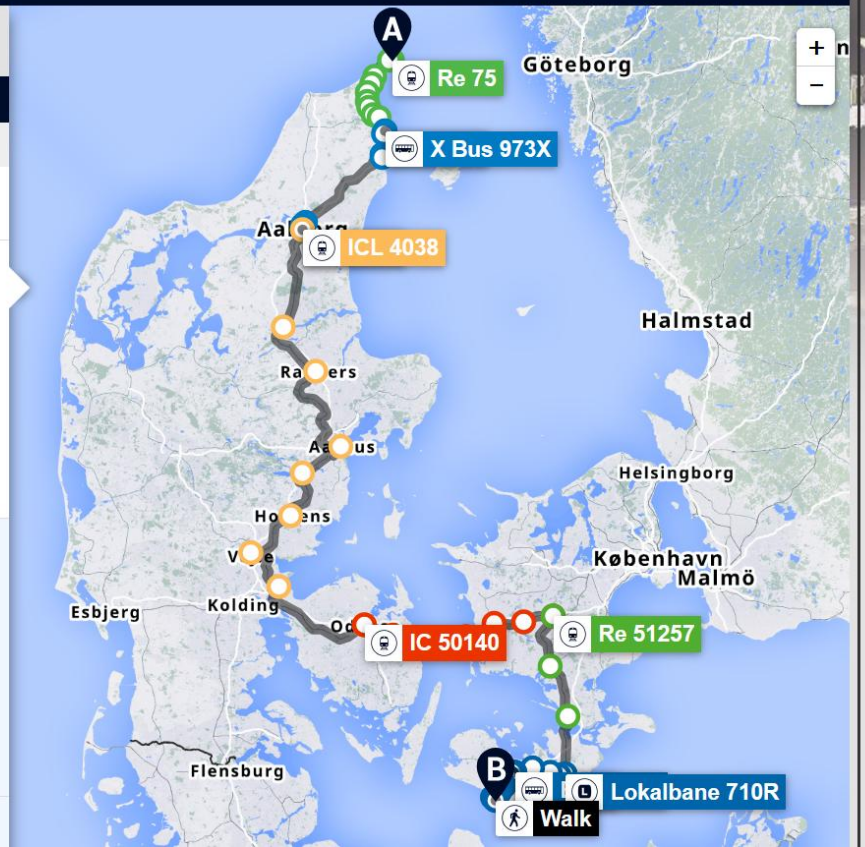
Public Transport

Mon. 11/10/2025

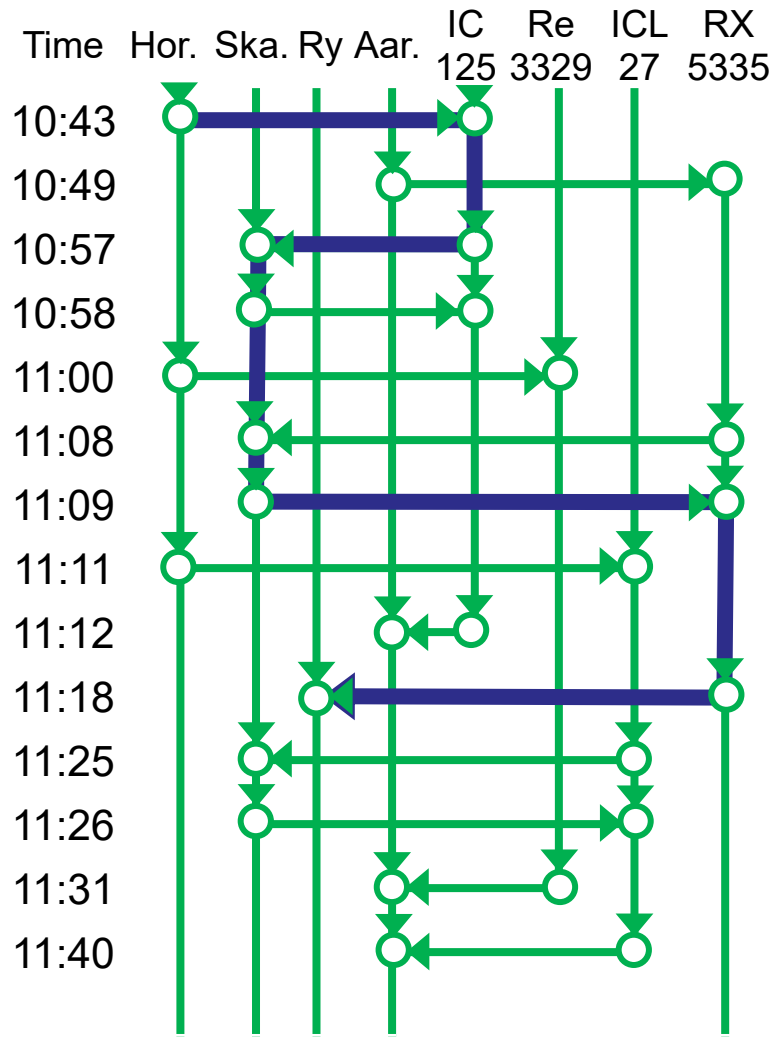
Earlier

08:42 — 17:49
75 973X ICL … IC 5… Re … 710R 720R
9 h 7 min., 6 Transfers
Price & buy | Details

08:42 — 18:19
75 973X IC 4… IC 5… Re … Re … 710R 720R
9 h 37 min., 7 Transfers
Price & buy | Details



Travel plan (Horsens to Ry)

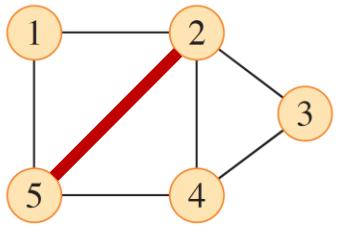


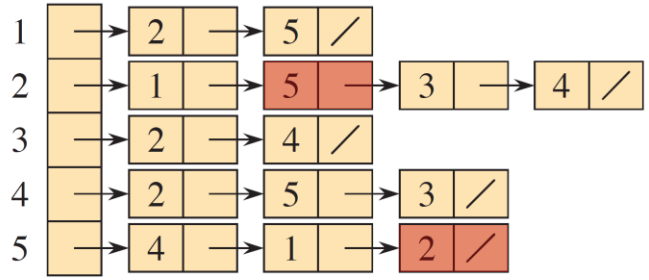
Algorithm
Find earliest node for Ry reachable from a given start node in Horsens

Train	Arr	Dep	Station
		10:43	Horsens
IC125	10:57	10:58	Skanderborg St
	11:12		Aarhus H
		11:00	Horsens
Re3329	11:31		Aarhus H
		11:11	Horsens
ICL27	11:25	11:26	Skanderborg St
	11:40		Aarhus H
		10:49	Aarhus H
RX5335	11:08	11:09	Skanderborg St
	11:18		Ry St

part of train schedule

Graph representations – adjacency-lists and adjacency-matrices



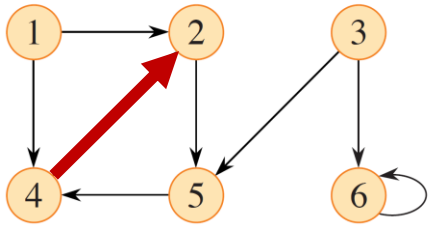


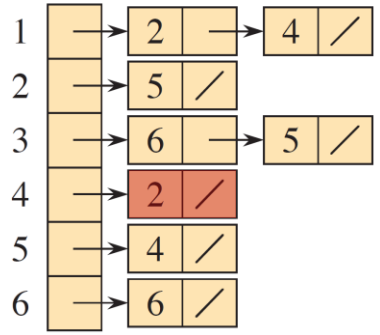
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Undirected graph

Space O(n+m)

Space O(n^2)

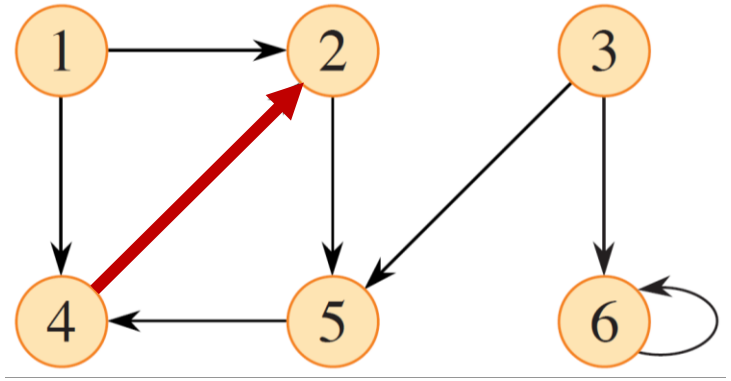




	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Directed graph (a.k.a. digraph)

Graph representations – more alternatives



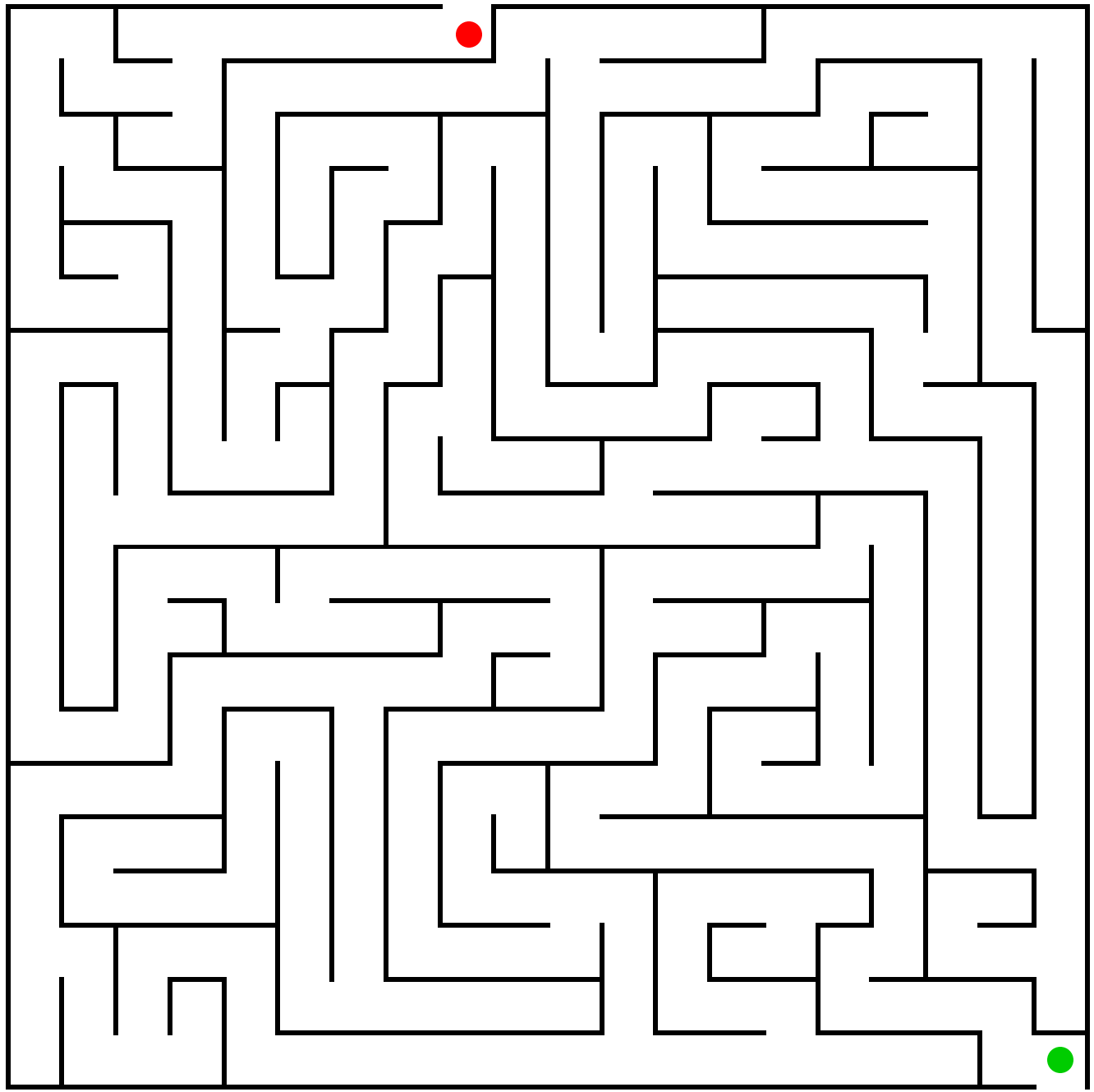
Edge list
(list/array with pairs of integers)

(1, 2)	(1, 4)	(2, 5)	(3, 5)	(3, 6)	(4, 2)	(5, 4)	(6, 6)

Compact adjacency lists
(two arrays with integers)

	1	2	3	4	5	6
V	1	3	4	6	7	8

E	2	4	5	5	6	2	4	6



Breadth first search (BFS)

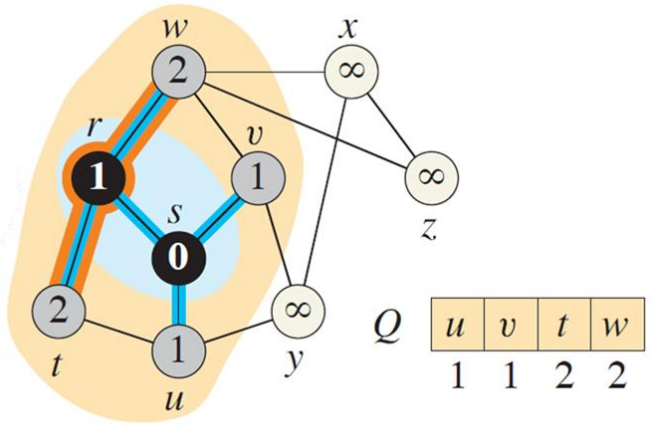
```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each vertex v in G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

u.color
WHITE = nodes not visited yet
GRAY = nodes in queue *Q*
BLACK = nodes visited

u.d = distance to *s*

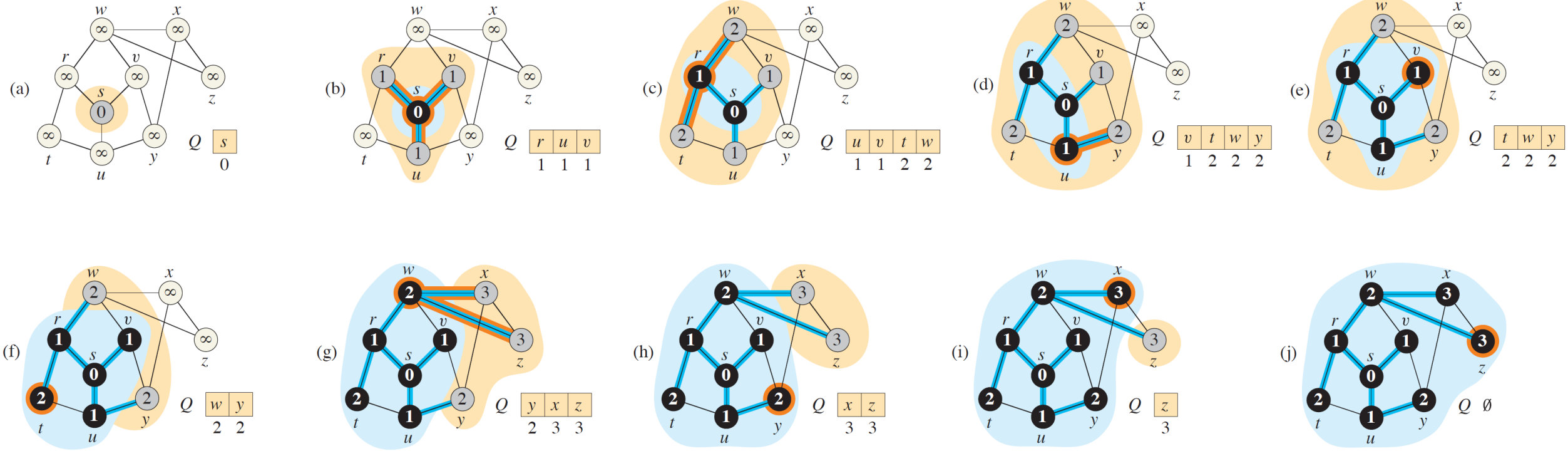
u.π = parent to *u* in BFS tree

Q = queue with grey nodes
(which are connected to black nodes)



Time O(*n* + *m*)

BFS example

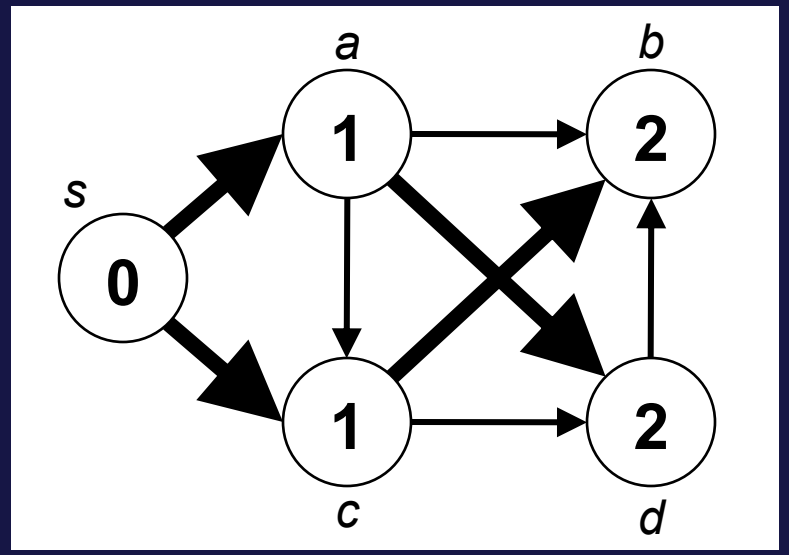

(a)
Q s
0
(b)
Q r u v
1 1 1
(c)
Q u v t w
1 1 2 2
(d)
Q v t w y
1 2 2 2
(e)
Q t w y
2 2 2
(f)
Q w y
2 2
(g)
Q y x z
2 3 3
(h)
Q x z
3 3
(i)
Q z
3
(j)
Q ∅

Is the below a valid BFS tree?

a) Yes

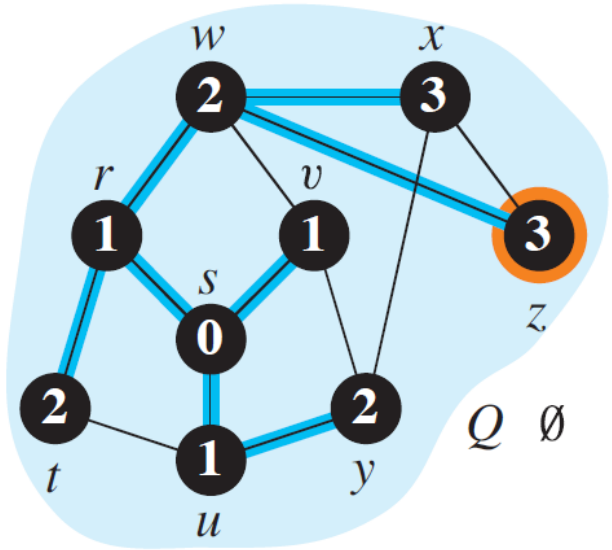
b) No

c) Don't know



BFS – print path from *s* to *v*

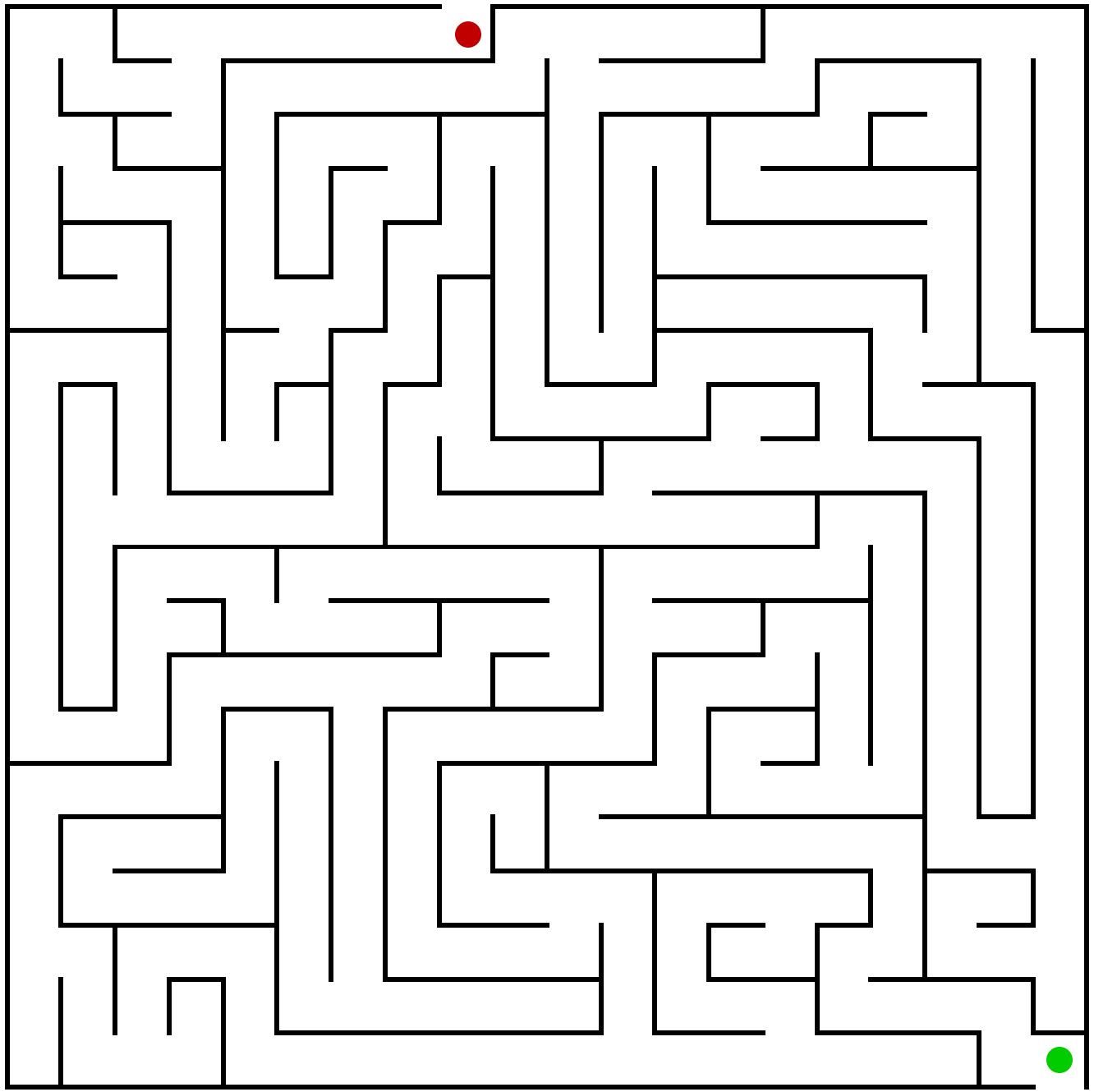
```
PRINT-PATH(G, s, v)
1  if v == s
2      print s
3  elseif v.π == NIL
4      print "no path from" s "to" v "exists"
5  else PRINT-PATH(G, s, v.π)
6      print v
```



Time O(*n*)

Algorithms and Data Structures

Depth first search (DFS), topological sorting,
strongly connected components
[CLRS, Chapter 20.3-20.5]



Depth first search (DFS)

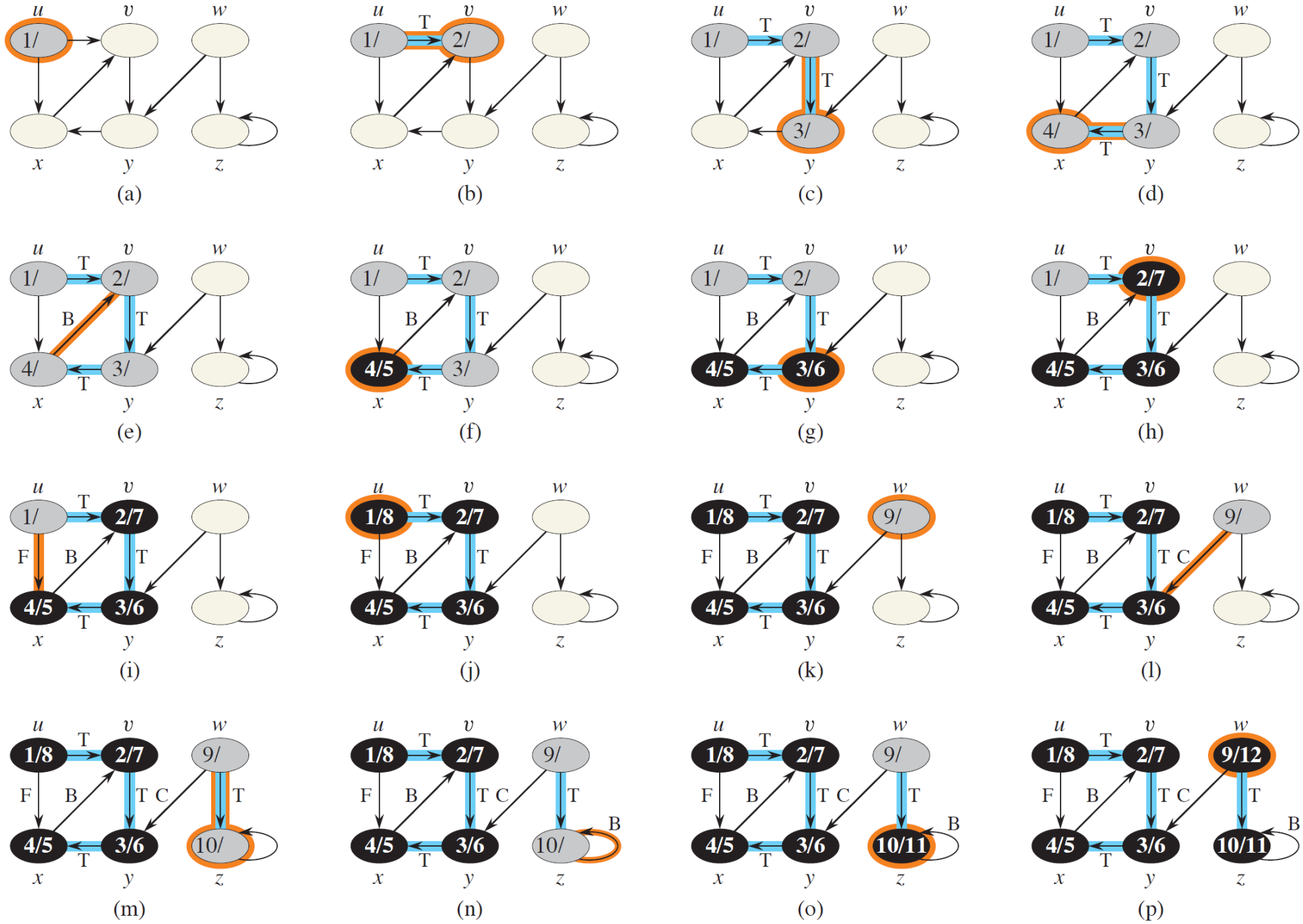
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each vertex v in G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  time = time + 1
9  u.f = time
10 u.color = BLACK
```

u.color
WHITE = nodes not yet visited
GRAY = nodes on recursion stack
BLACK = nodes visited

u.π = parent of *u* in DFS tree

u.d = discovery time for *u*
u.f = finishing time for *u*

Time O(*n* + *m*)

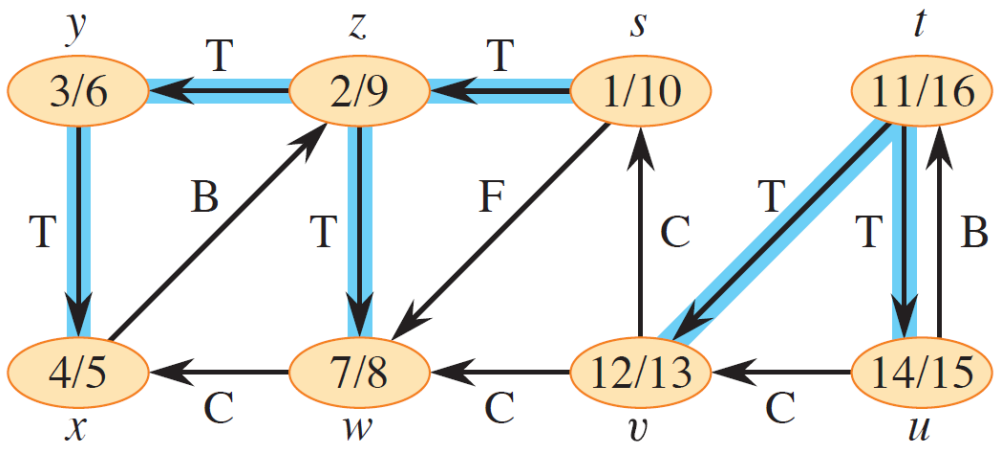

u v w
1/
x y z
(a)
T
2/
(b)
3/
(c)
4/
(d)
B
(e)
4/5
(f)
3/6
(g)
2/7
(h)
F
(i)
1/8
(j)
9/
(k)
C
(l)
10/
(m)
(n)
10/11
(o)
9/12
(p)

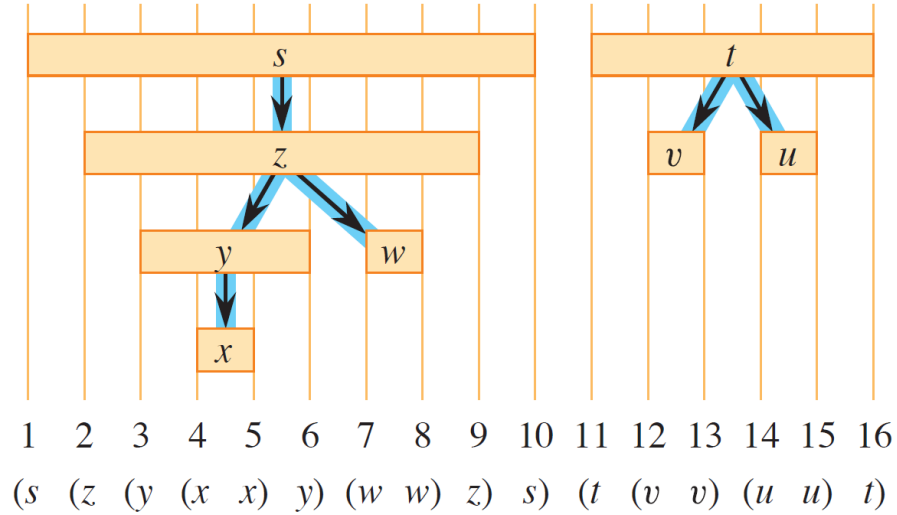
Can a node have 13/17 as DFS discovery/finishing times ?

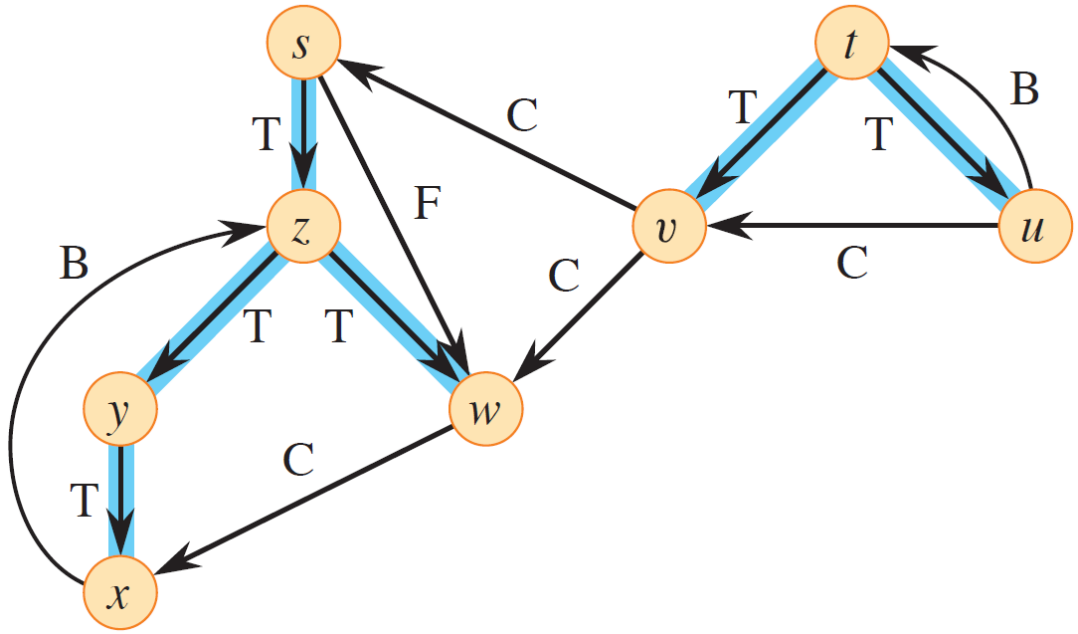
a) Yes

b) No

c) Don't know







= tree edge
B = back edge
C = cross edge
F = forward edge

Applications of BFS and DFS

BFS Find distances from start node
(distance = # edges, e.g., RushHour state space)

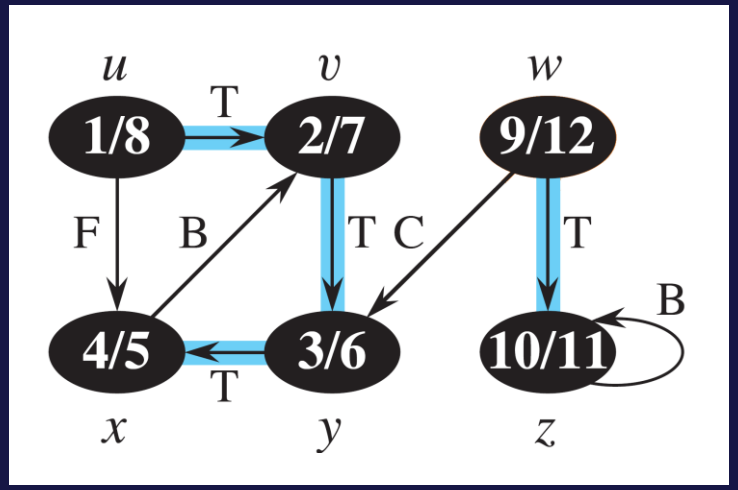
DFS *Topological sorting*
Strongly connected components
Planarity testing

Assume there exists a path $a \rightsquigarrow b$ in a directed graph.
Is it always true that $a.f > b.f$?

a) Yes

b) No

c) Don't know



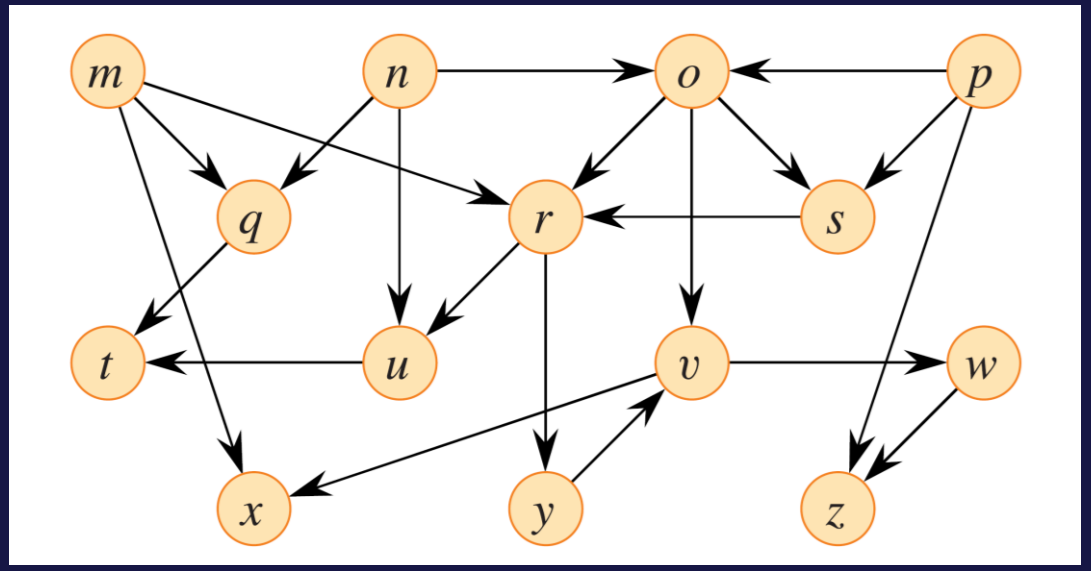
Assume there exists a path $a \rightsquigarrow b$ in a directed graph **without cycles**

Is it always true that $a.f > b.f$?

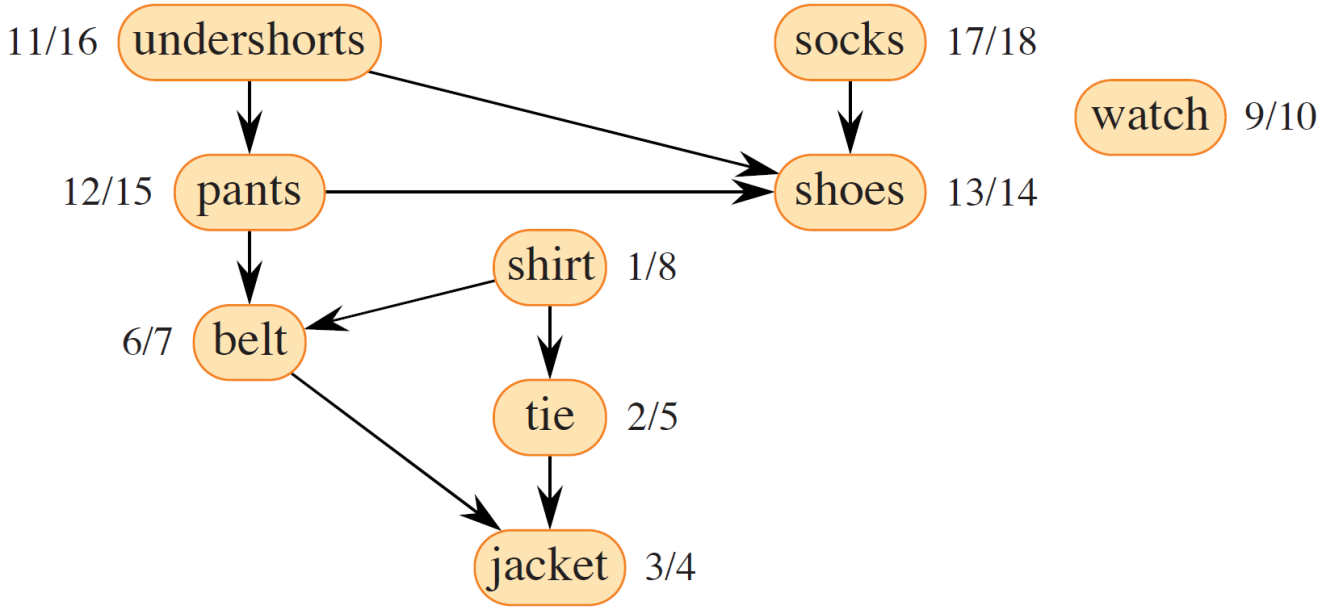
a) Yes

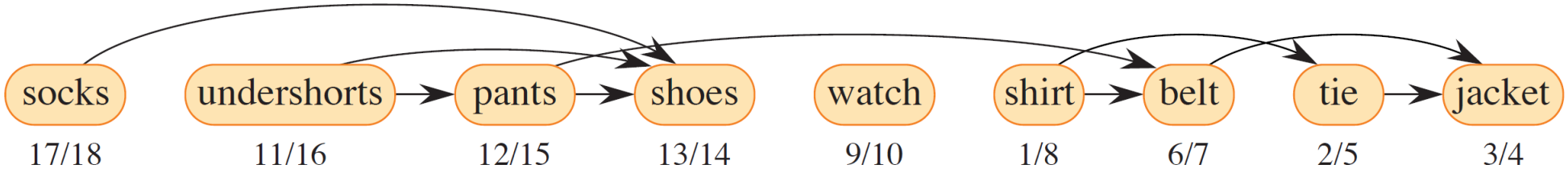
b) No

c) Don't know

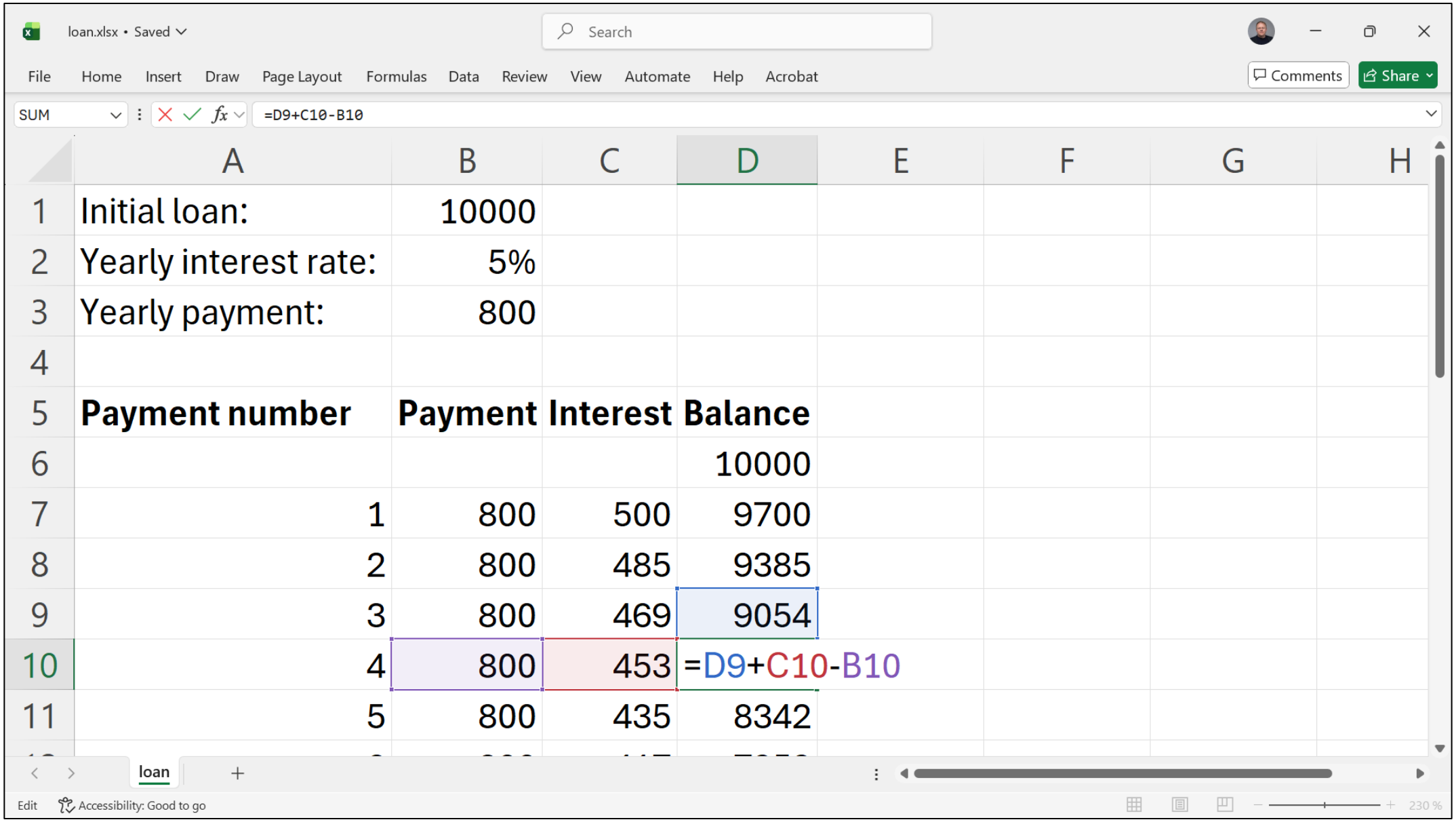


Topological sorting of acyclic graphs





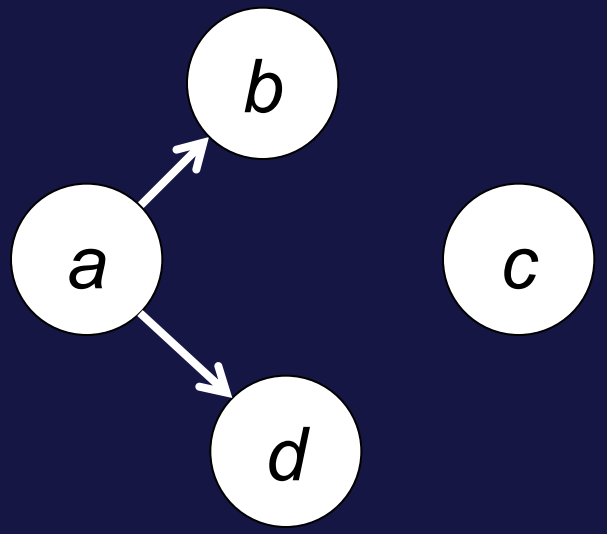
All edges are left-to-right



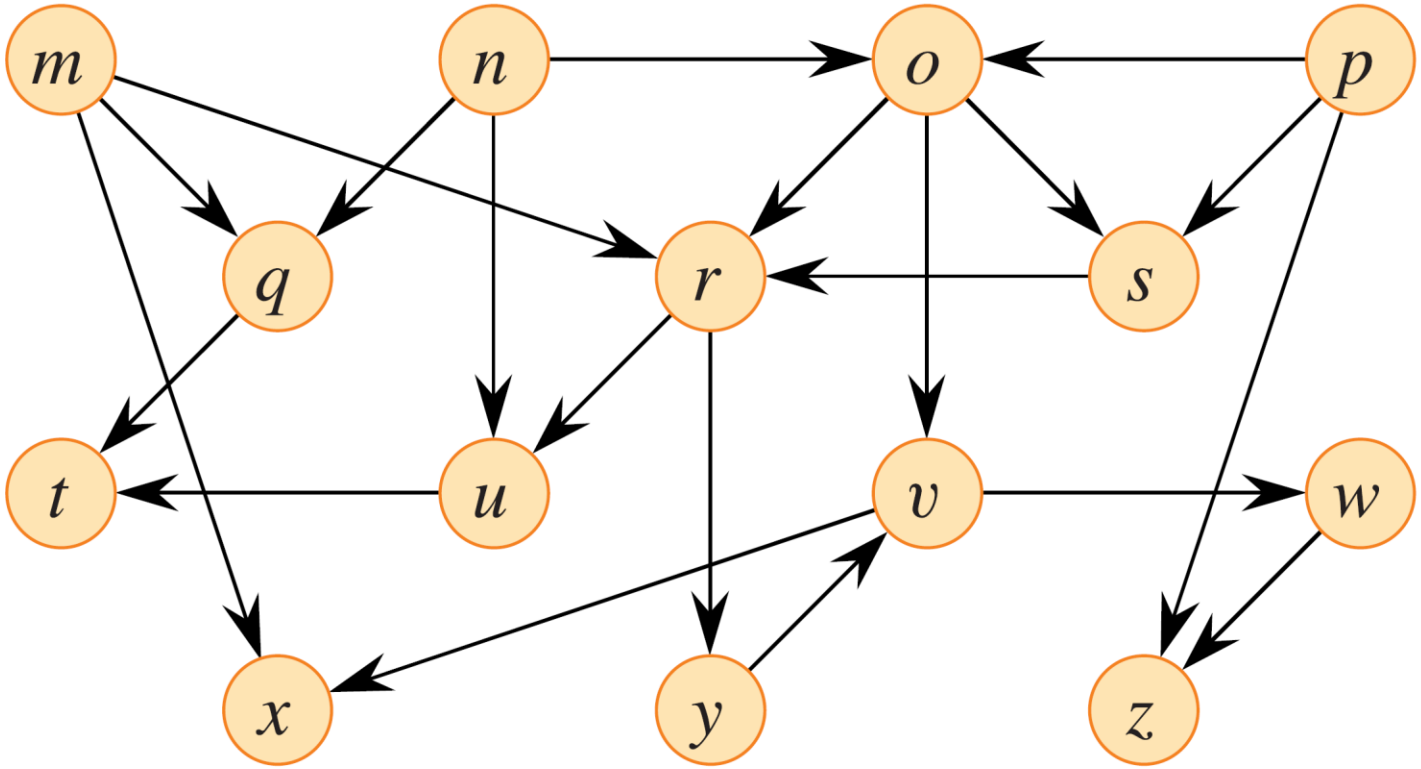
Topological sorting = a valid order to compute cells values

ways to topological sort the graph ?

a) 1
b) 2
c) 3
d) 4
e) 5
f) 6
g) 7
h) 8
i) 9
j) Don't know



Topological sorting (I)

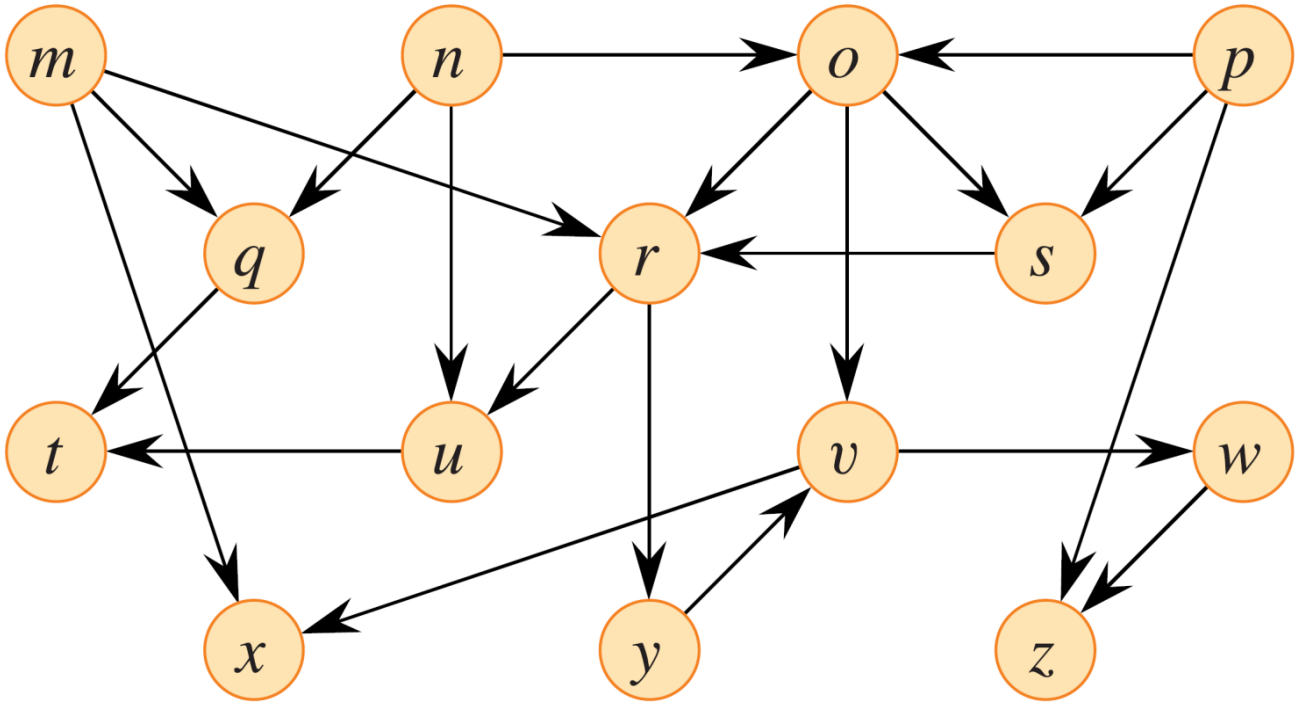


Algorithm Greedily remove a node of indegree 0 (and its outgoing edges) and add the node last in the topological order

Time O($m + n$)

Arthur B. Kahn, *Topological sorting of large networks*, Communications of the ACM, 1962, doi:10.1145/368996.369025

Topological sorting (II)



TOPOLOGICAL-SORT(G)

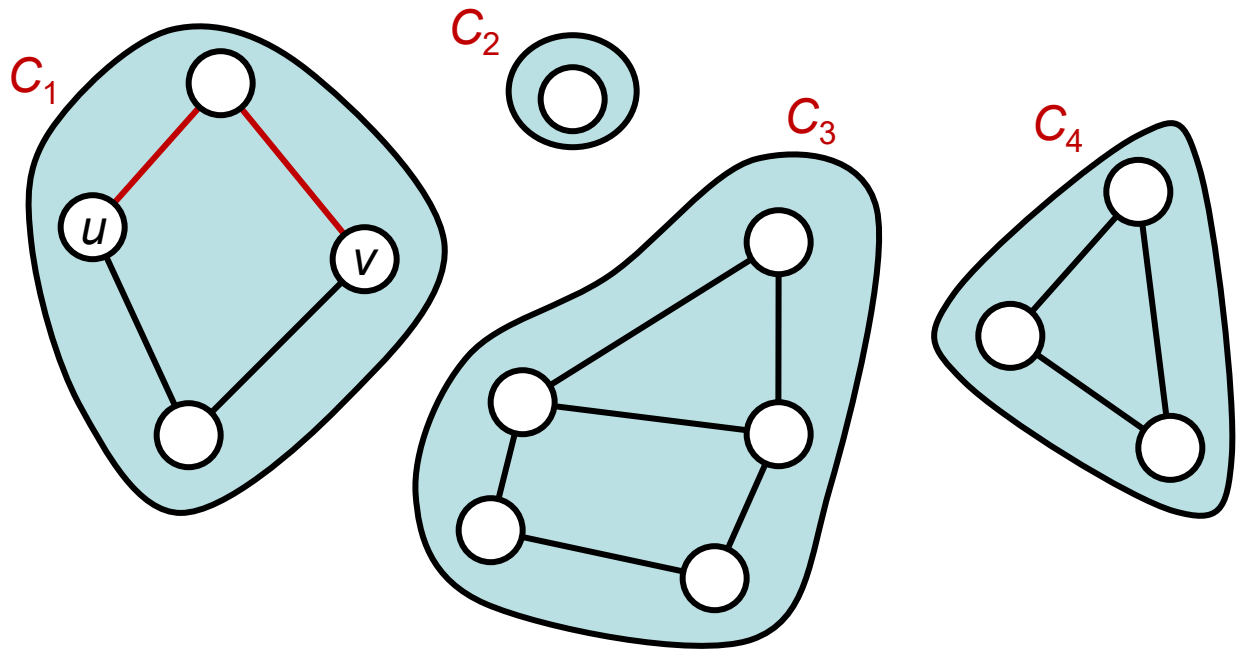
1 call DFS(G) to compute finish times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

Time O(m + n)

Robert E. Tarjan, *Edge-disjoint spanning trees and depth-first search*, Acta Informatica, 1976, doi: 10.1007/BF00268499

Connected components

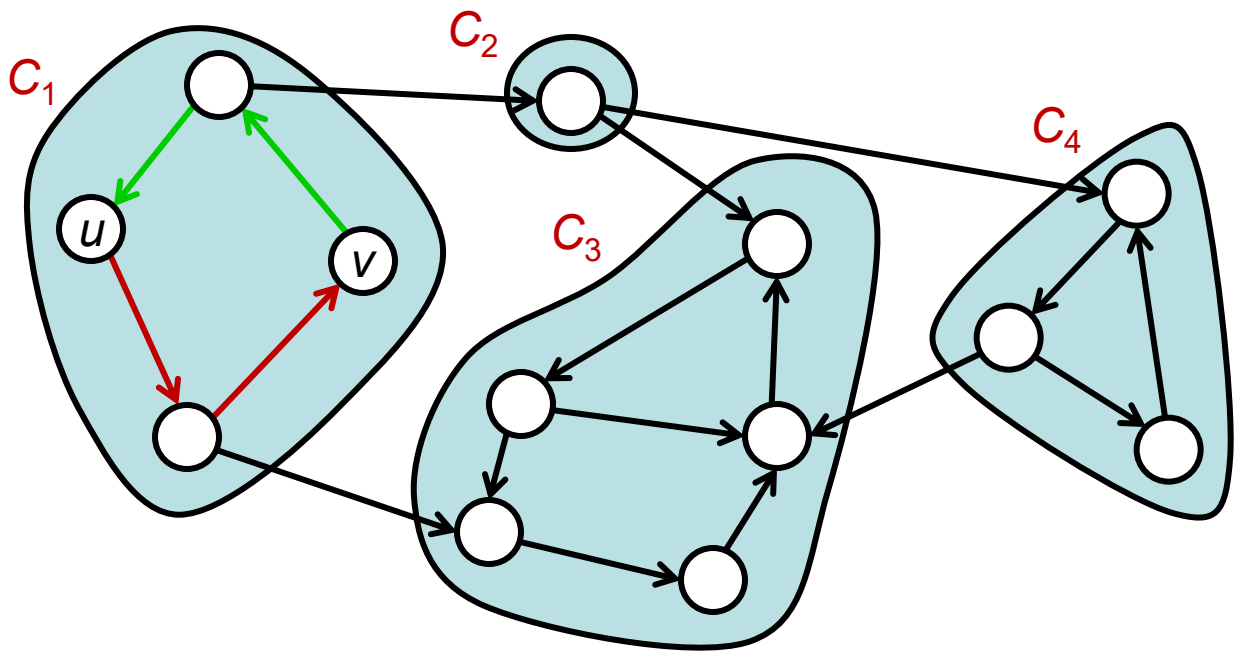
Partitioning of the nodes of an undirected graph into components $C_1,\ldots,C_k$, such that u and v are in the same component $\Leftrightarrow$ there is a path between u and v



DFS/BFS time O(m + n)

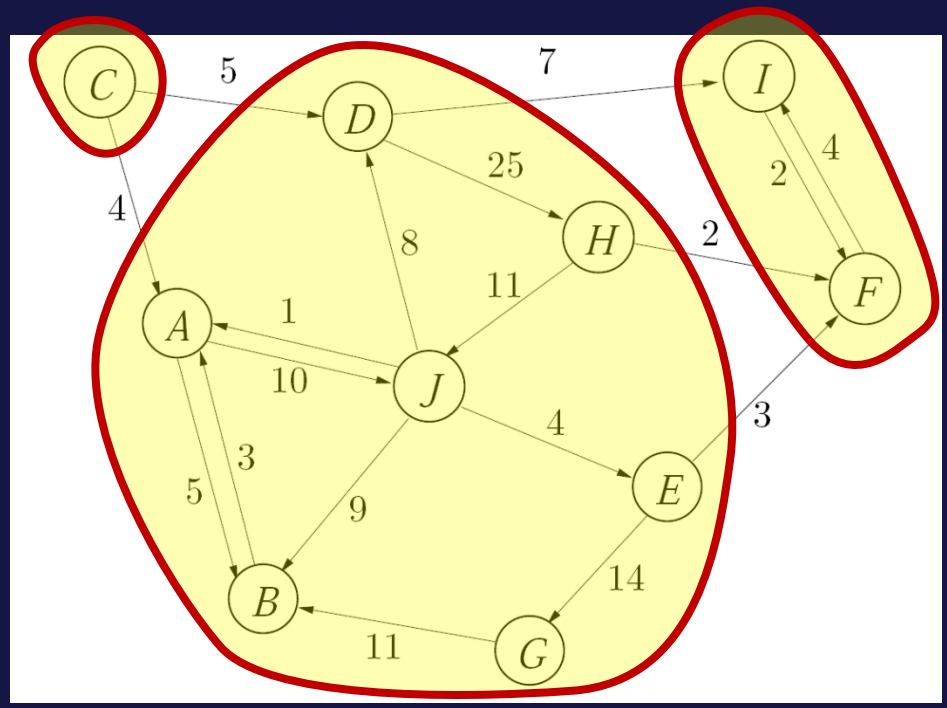
Strongly connected components

Partitioning of the nodes of a directed graph into components $C_1,\ldots,C_k$, such that u and v are in the same component ⇔ there is a path from u to v *and* a path from v to u



strongly connected components ?

a) 1
b) 2
c) 3
d) 4
e) 5
f) Don't know



Exam Summer 2009, Question 1(d)

{*A*, *B*, *D*, *E*, *G*, *H*, *J*}, {*C*}, {*F*, *I*}

Strongly connected components

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finish times $u.f$ for each vertex u
2 create G^T
3 call DFS(G^T), but in the main loop of DFS, consider the vertices
in order of decreasing $u.f$ (as computed in line 1)
4 output the vertices of each tree in the depth-first forest formed in line 3 as a
separate strongly connected component

```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

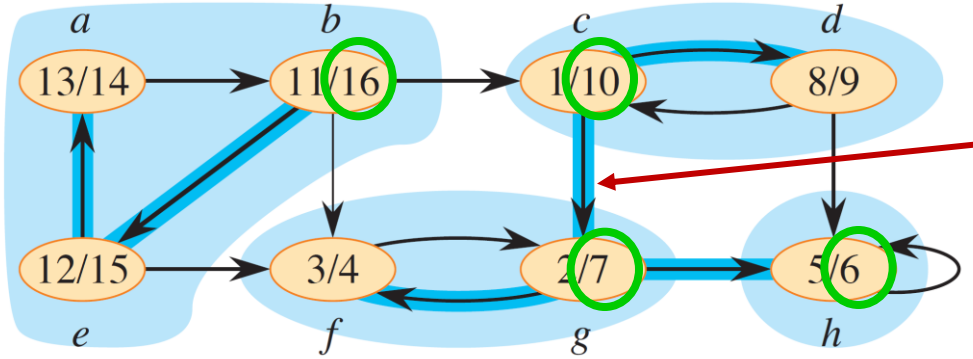
```
DFS-VISIT(G, u)
1  time = time + 1
2  u.d = time
3  u.color = GRAY
4  for each vertex v in G.Adj[u]
5      if v.color == WHITE
6          v.π = u
7          DFS-VISIT(G, v)
8  time = time + 1
9  u.f = time
10 u.color = BLACK
```

Time O(m + n)

Robert Endre Tarjan, *Depth-First Search and Linear Graph Algorithms*. SIAM Journal of Computing, 1972, doi: 10.1137/0201010

Strongly connected components

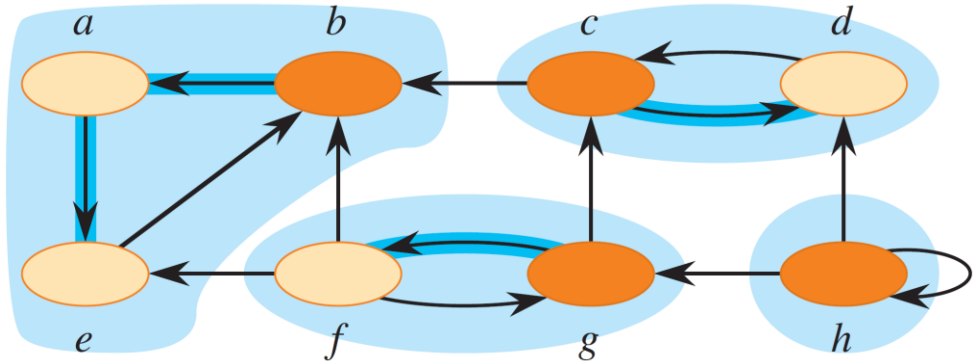
G

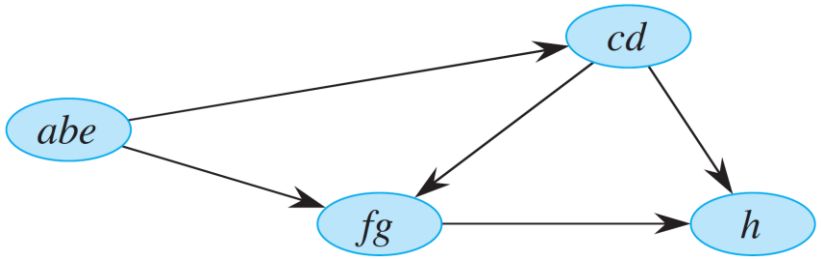


DFS tree edges between two strongly connected components

The largest finishing times in each component form a (reversed) topological sorting of the components

G^T



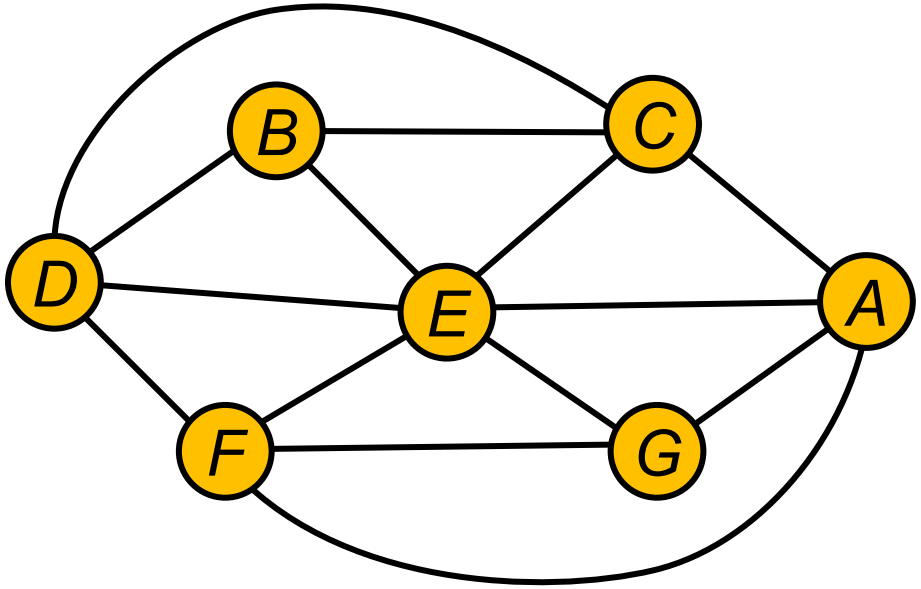


not curriculum

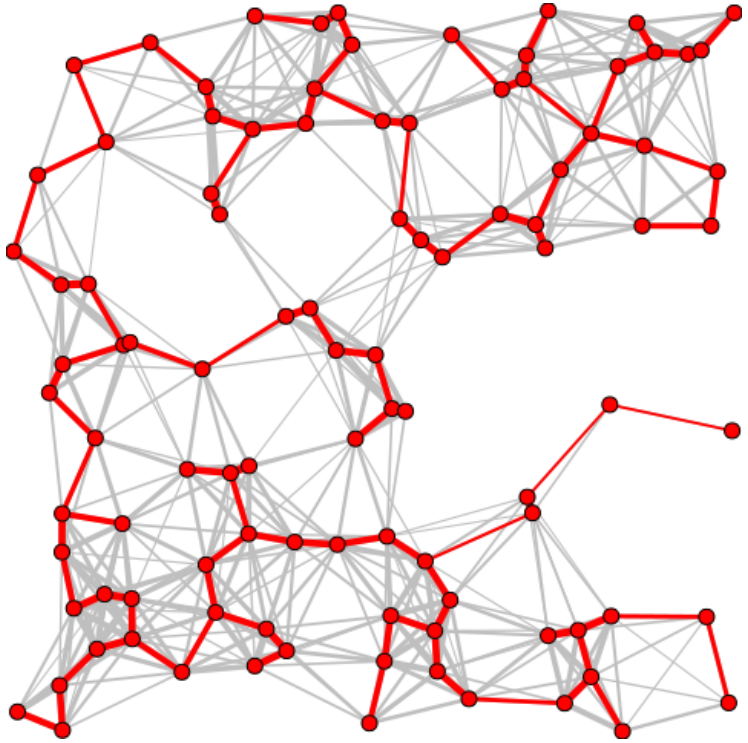
Planarity testing

- Decide if an undirected graph is planar, i.e., can be drawn in the plane without crossing edges
- Can be solved using DFS (nontrivial)

$E = \{\{A, C\}, \{A, E\}, \{A, F\}, \{A, G\}, \{B, C\},$
$\{B, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, E\},$
$\{D, F\}, \{E, F\}, \{E, G\}, \{F, G\}\}$



John Hopcroft, Robert Tarjan, *Efficient planarity testing*, Journal of the ACM, 1974, doi: 10.1145/21850.321852



Algorithms and Data Structures

Minimum spanning trees (MST)
[CLRS, Chapter 21]

Kontaimu p. prof. Dr V. Bydžovskému v Hlubokém 5.X.26. O. Borůvka.

ZVLÁŠTNÍ OTISK Z ČASOPISU „ELEKTROTECHNICKÝ OBZOR“
Roč. 15. Čís. 10. Praha III., Cihelná 102. 5. března 1926.

Dr. OTAKAR BORŮVKA:

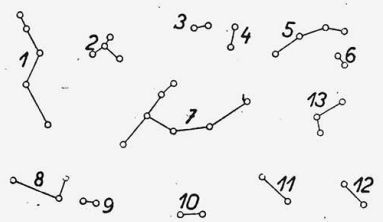
Příspěvek k řešení otázky ekonomické stavby elektrovodných sítí.

Ve své práci „O jistém problému minimálním“ *) odvodil jsem obecnou větu, jíž jest ve zvláštním případě řešena tato úloha:

V rovině (v prostoru) jest dáno n bodů, jejichž vzájemné vzdálenosti jsou vesměs různé. Jest je spojiti sítí tak, aby:

1. každé dva body byly spojeny buď přímo anebo prostřednictvím jiných,
2. celková délka sítě byla co nejmenší.

Obr. 1.



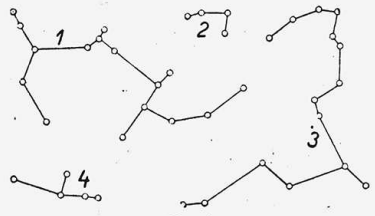
Obr. 2.

Jest zřejmé, že řešení této úlohy může míti v elektrotechnické praksi při návrzích plánů elektrovodných sítí jistou důležitost; z toho důvodu je zde stručně na příkladě vyložím. Čtenáře, jenž by se o věc blíže zajímal, odkazuji na citované pojednání.

*) Vyjde v nejbližší době v Pracích Moravské přírodovědecké společnosti.

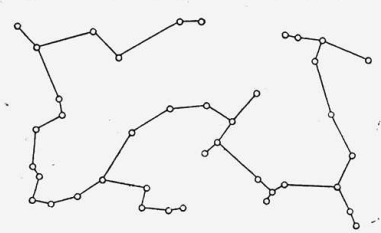
Řešení úlohy provedu v případě 40 bodů daných v obr. 1.

Každý z daných bodů spojím s bodem nejbližším. Tedy na př. bod 1 s bodem 2, bod 2 s bodem 3, bod 3 s bodem 4 (bod 4 s bodem 3), bod 5 s bodem 2, bod 6 s bodem 5, bod 7 s bodem 6, bod 8 s bodem 9, (bod 9 s bodem 8) atd. Obdržím řadu polygonálních tahů 1, 2,.... 13 (obr. 2).



Obr. 3.

Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy na př. 1 s tahem 2, (tah 2 s tahem 1), tah 3 s tahem 4, (tah 4 s tahem 3) atd. Obdržím řadu polygonálních tahů 1, 2,.... 4 (obr. 3).



Obr. 4.

Orazec převrácen.

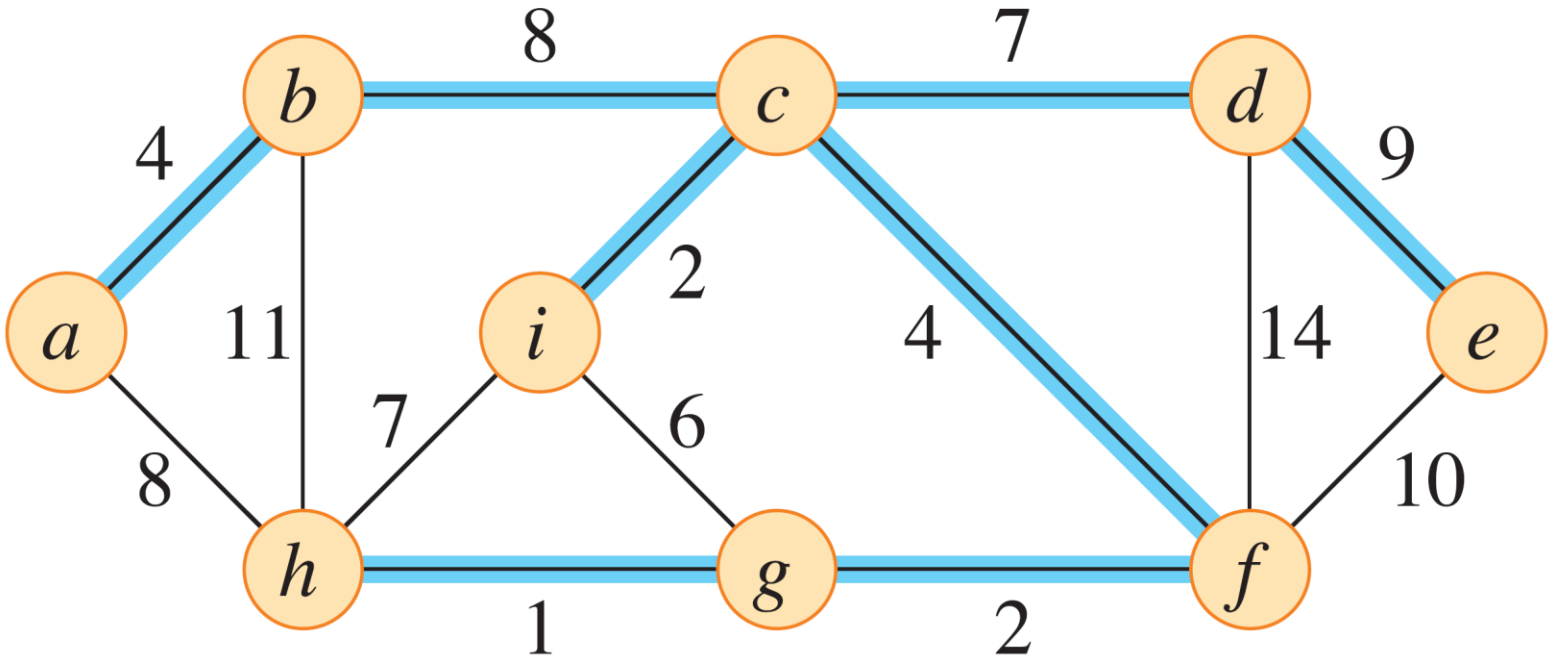
Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy tah 1 s tahem 3, tah 2 s tahem 3 (tah 3 s tahem 1), tah 4 s tahem 1. Obdržím konečně jediný polygonální tah (obr. 4), jenž řeší danou úlohu.

Matematický ústav Masarykovy university v Brně,
v lednu 1926.

Otakar Borůvka, *Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí (Contribution to the solution of a problem of economical construction of electrical networks)*, Elektronický Obzor, 15:153–154, 1926

Jaroslav Nešetřil, Eva Milková, Helena Nešetřilová, *Otakar Borůvka on minimum spanning tree problem Translation of both the 1926 papers, comments, history*, Discrete Mathematics, 2001, doi: 10.1016/S0012-365X(00)00224-7

Minimum spanning tree (MST)

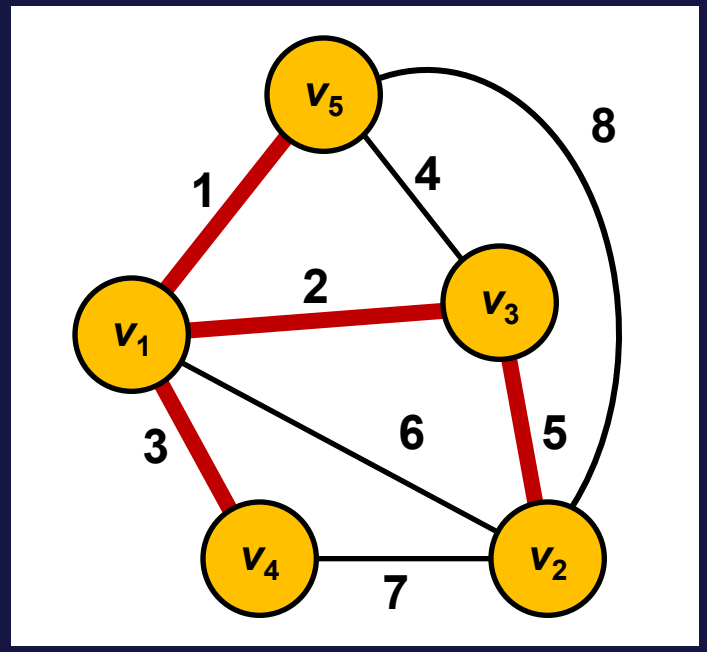


Problem

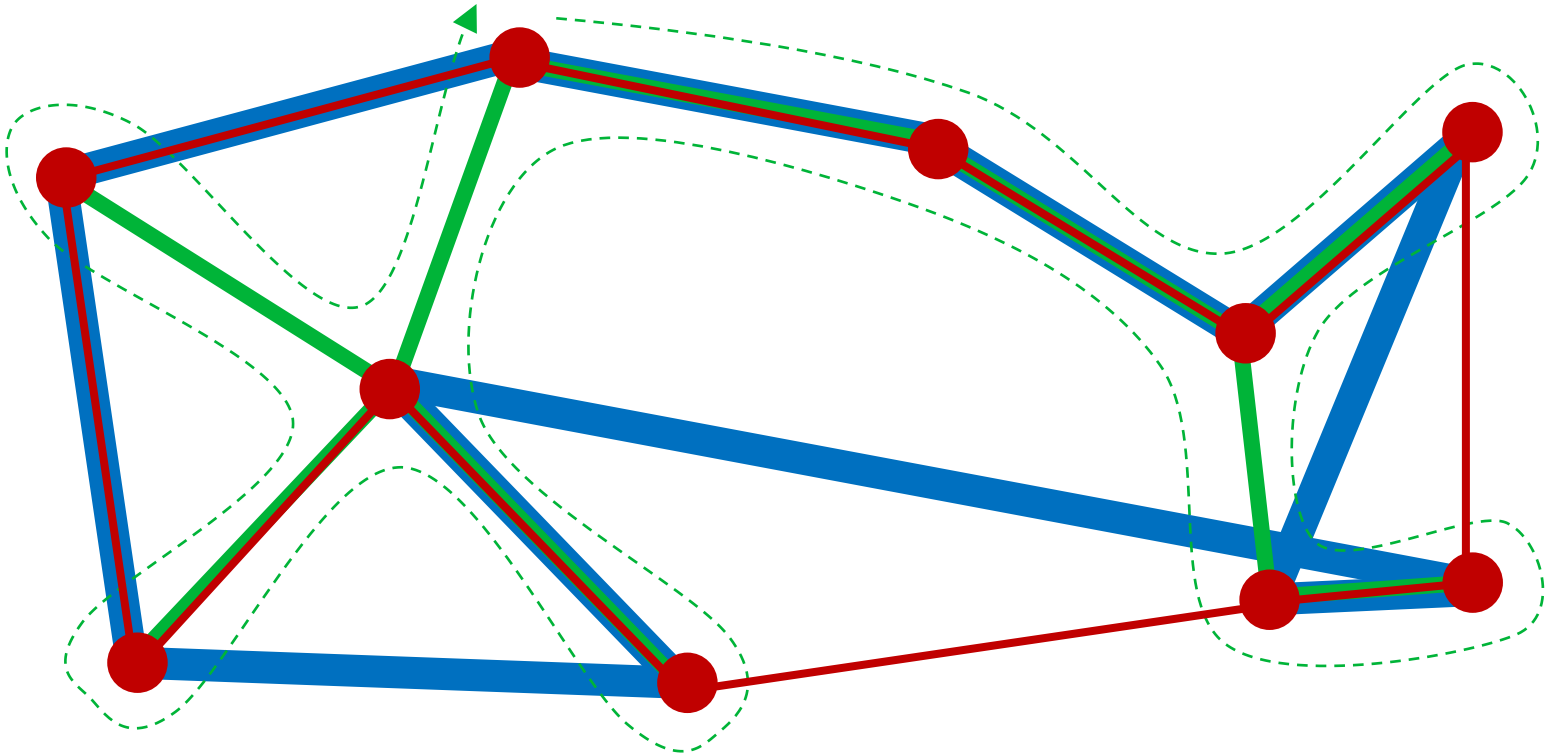
Find a **spanning tree** for a **connected undirected weighted** graph such that the **sum of the trees edge weights is smallest possible**

Weight of a minimum spanning tree?

a) 10
b) 11
c) 12
d) 13
e) 14
f) 15
g) 16
h) Don't know



MST application – Euclidian shortest Hamiltonian cycle

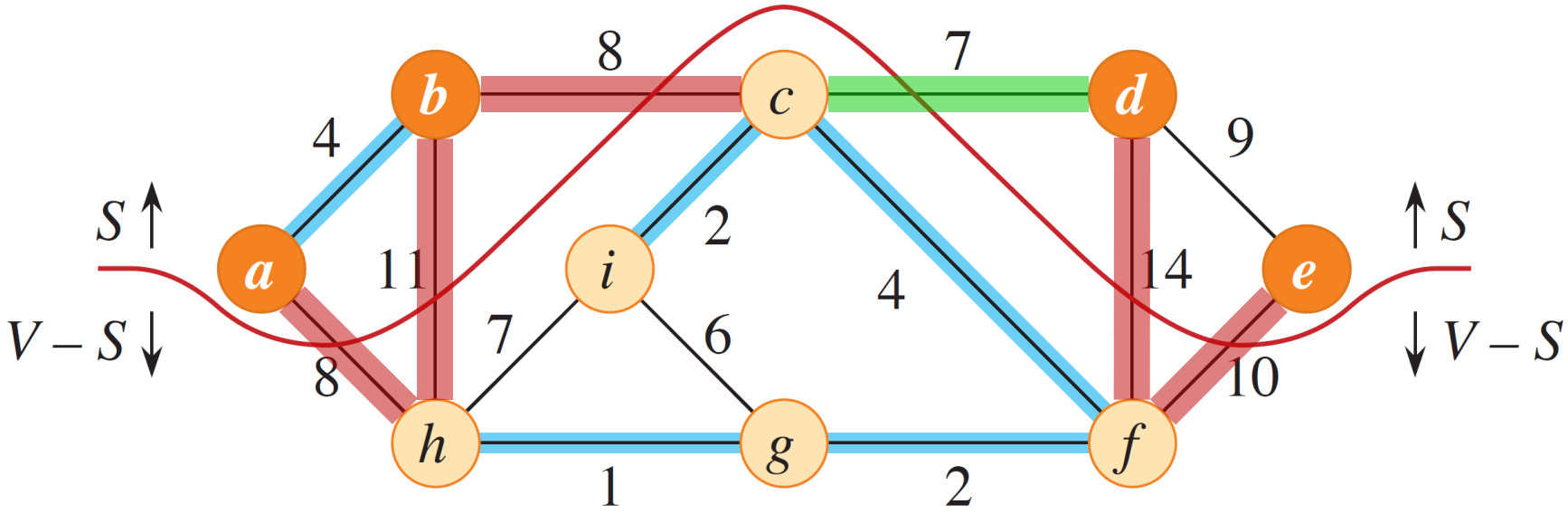


Euler tour
(of a tree)

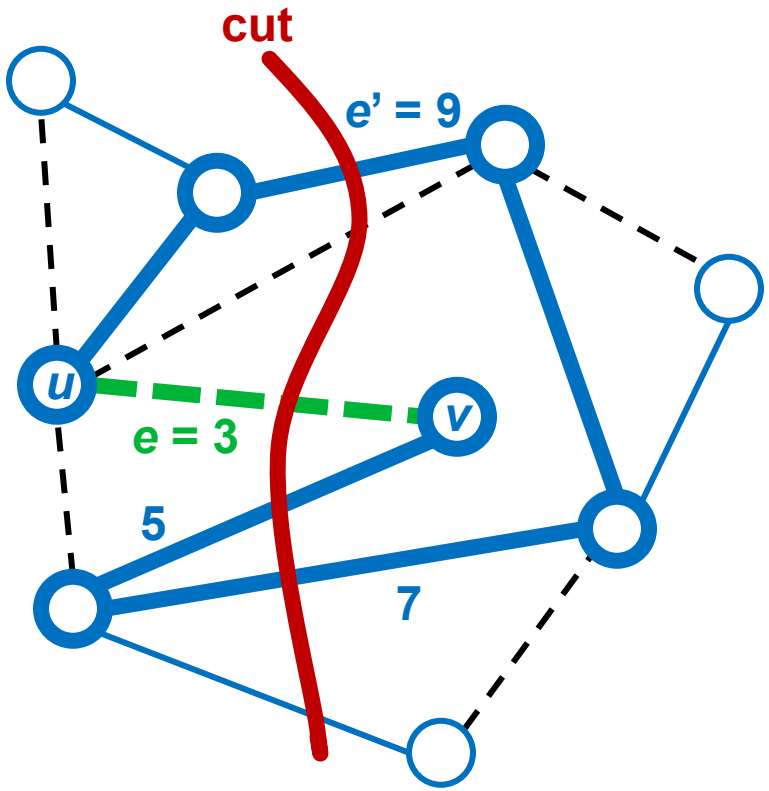
Theorem : Tour found using an **MST** is a 2-approximation to the **shortest cycle**

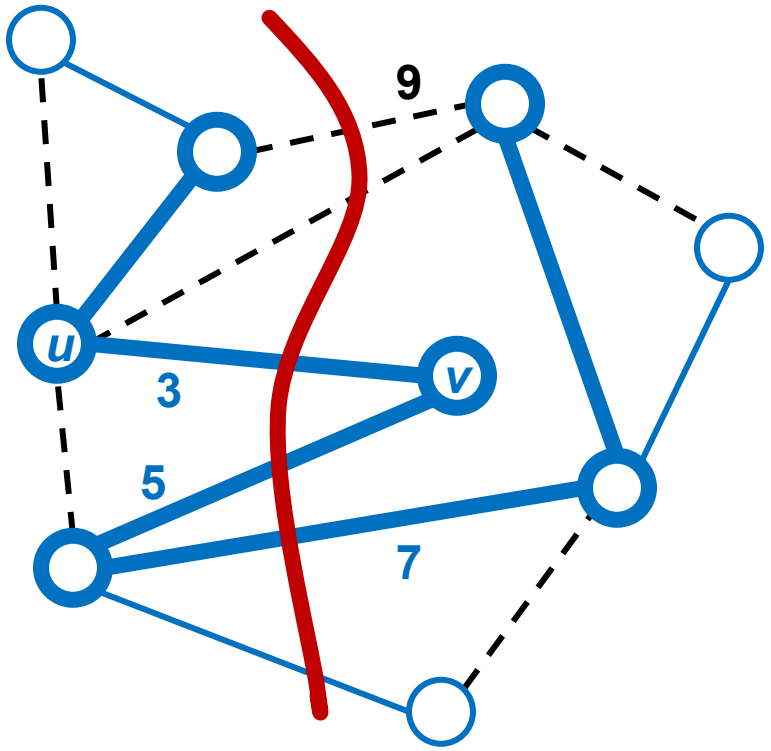
(|**Tour**| ≤ 2·|**MST**| and |**MST**| ≤ |**shortest cycle**|)

Minimum spanning tree – cut



Theorem
If all weights are distinct, then for every **cut(*S*, *V*-*S*)** the **lightest edge** crossing the cut is part of a minimum spanning tree





Assume for the sake of contradiction we have an MST and cut where the lightest edge *e* is not part of the MST

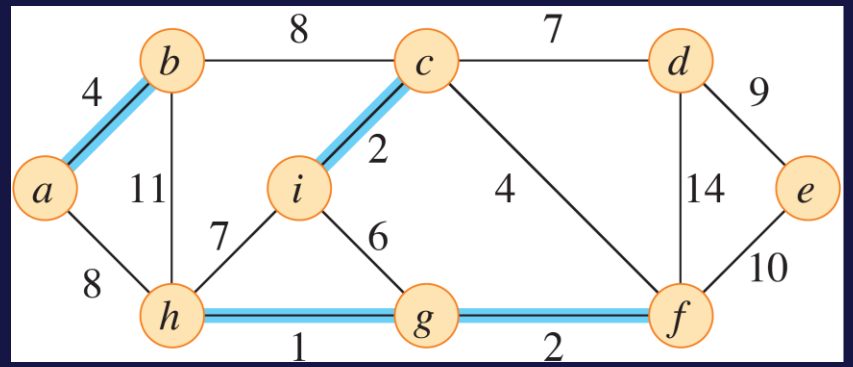
New spanning tree with smaller weight

Theorem

If all weights are distinct, then for every **cut(*S*, *V*-*S*)** the **lightest edge** crossing the cut is part of a minimum spanning tree

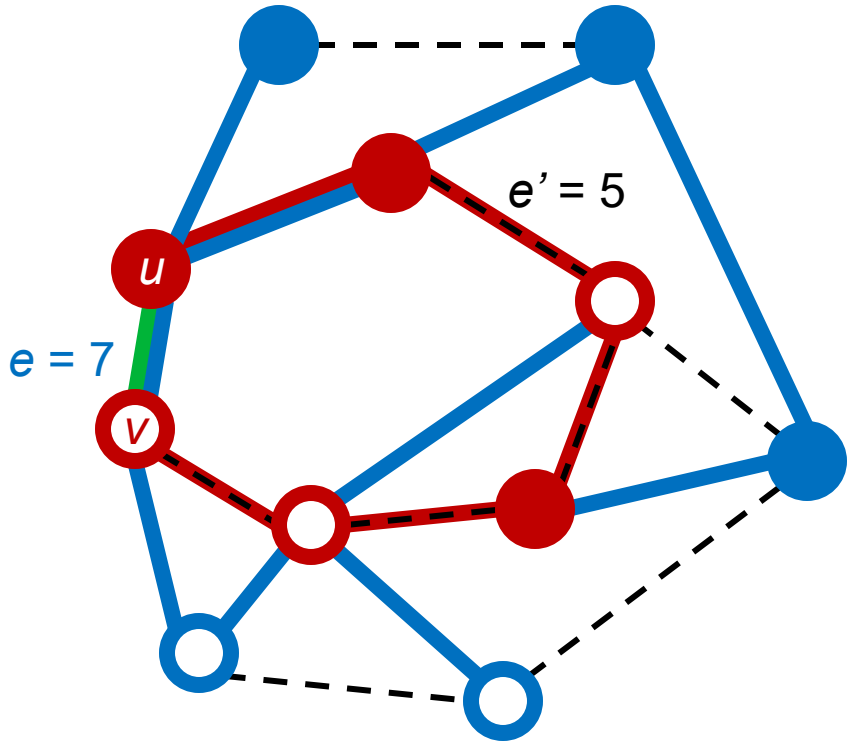
Can you always apply the cut theorem if you only have found a subset of the edges of an MST for a connected graph ?

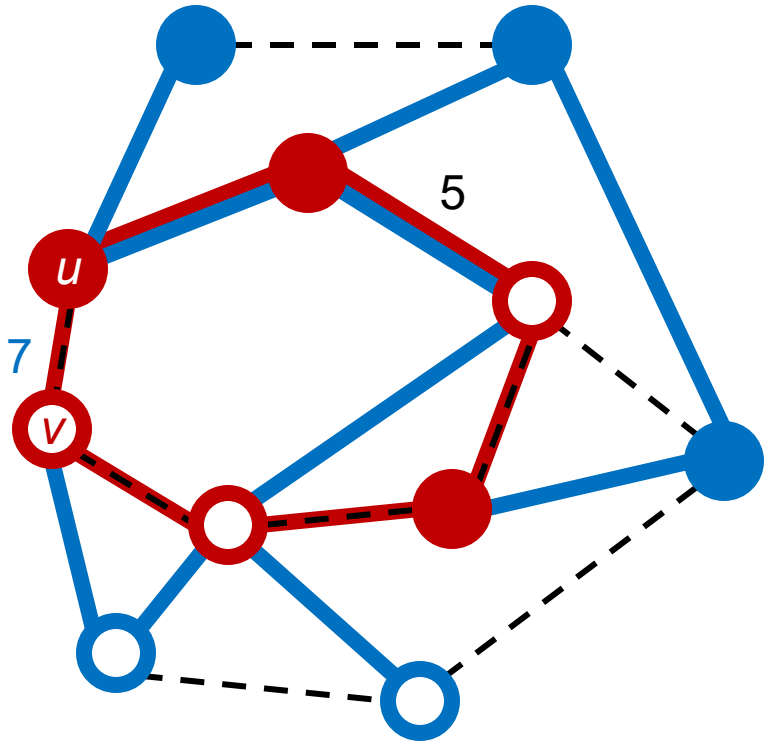
a) Yes
b) No
c) Don't know



Theorem
If all weights are distinct, then for every **cut(*S*, *V*-*S*)** the **lightest edge** crossing the cut is part of a minimum spanning tree

Theorem
If all weights are distinct, the **heaviest edge** on a **cycle** is not in a minimum spanning tree





Assume for the sake of contradiction we have an MST and a cycle where the heaviest edge *e* is part of the MST

New spanning tree with smaller weight

Minimum spanning tree – generic greedy algorithm

GENERIC-MST(G, w)

1 $A = \emptyset$
2 **while** A does not form a spanning tree
3 find an edge (u, v) that is safe for A
4 $A = A \cup \{(u, v)\}$
5 **return** A

A lightest edge over a cut including edges from *A*

Kruskal's algorithm

```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  create a single list of the edges in G.E
5  sort the list of edges into monotonically increasing order by weight
6  for each edge (u, v) taken from the sorted list in order
7       if FIND-SET(u) ≠ FIND-SET(v)
8            A = A ∪ {(u, v)}
9            UNION(u, v)
10 return A
```

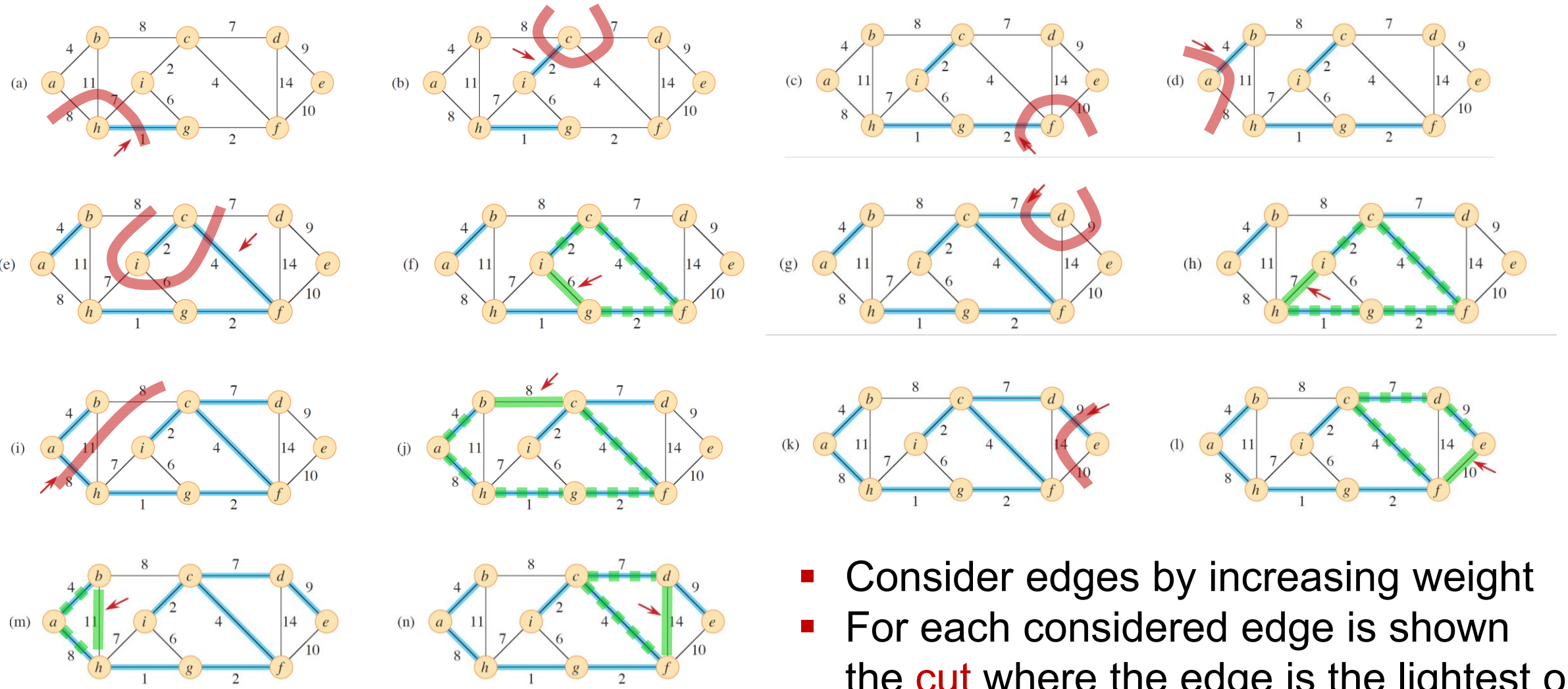
bottleneck

union-find data structure

Time $O(m \cdot \log n)$

J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem,* AMS, 1956, doi: 10.1090/S0002-9939-1956-0078686-7

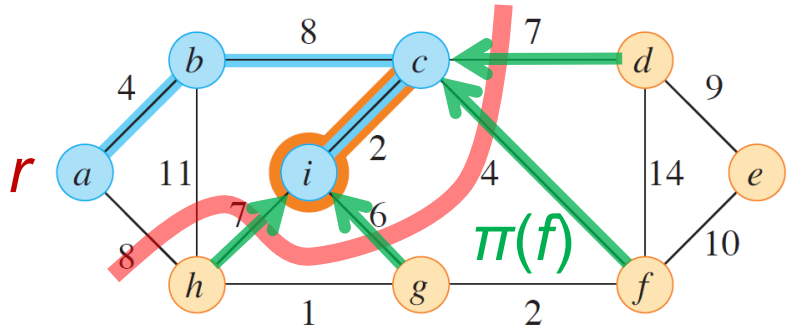
Kruskal's algorithm – example



- Consider edges by increasing weight
- For each considered edge is shown the cut where the edge is the lightest or the cycle where the edge is the heaviest

Prim's algorithm (a.k.a. Prim-Jarník's algorithm)

```
MST-PRIM(G, w, r)
 1  for each vertex u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = ∅
 6  for each vertex u ∈ G.V
 7      INSERT(Q, u)                          x n
 8  while Q ≠ ∅
 9      u = EXTRACT-MIN(Q)                    x n
10      for each vertex v in G.Adj[u]
11          if v ∈ Q and w(u, v) < v.key
12              v.π = u
13              v.key = w(u, v)
14              DECREASE-KEY(Q, v, w(u, v))   x m
```



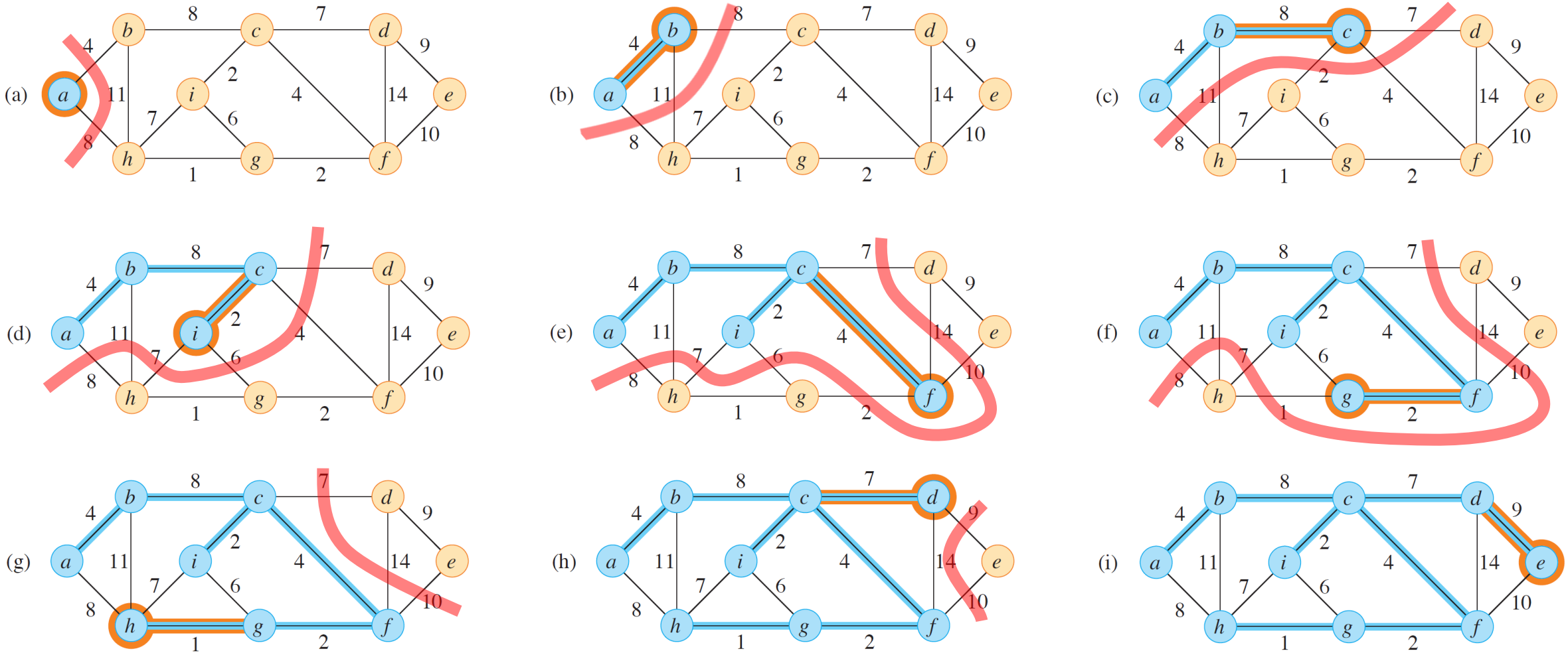
bottleneck – priority queue

Time O(m·log n)

V. Jarník, *O jistém problému minimálním*, Práce Moravské Přírodovědecké Společnosti, 1930, hdl: 10338.dmlcz/500726

R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Technical Journal, 1957 doi: 10.1002/j.1538-7305.1957.tb01515.x

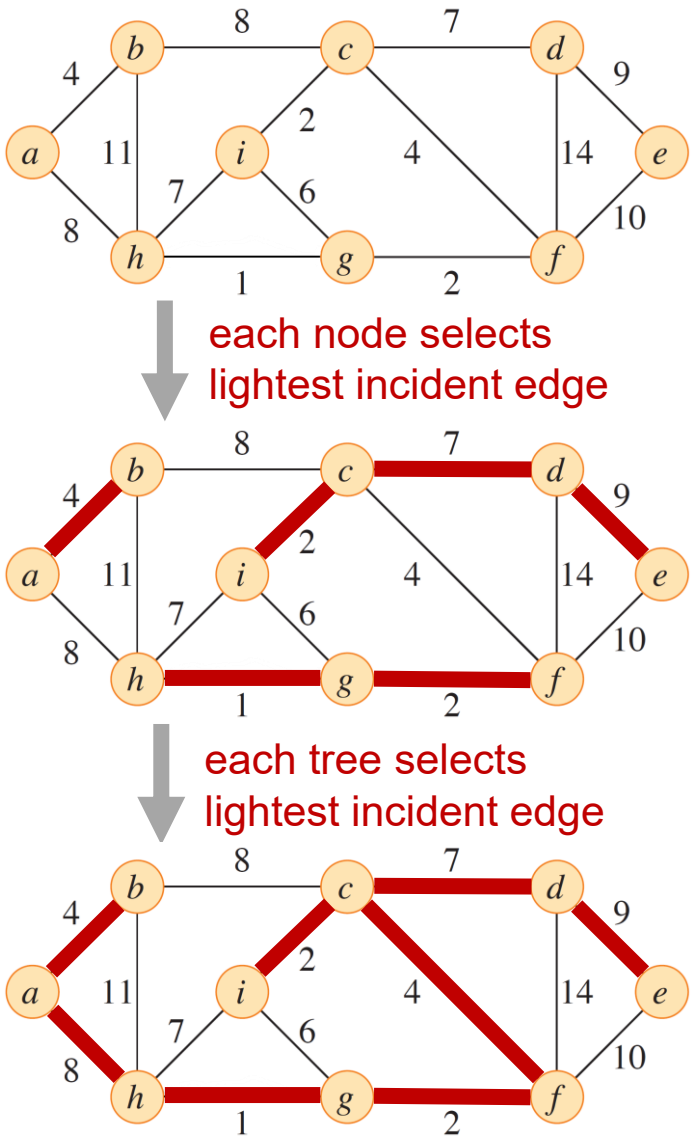
Prim's algorithm – example



not curriculum

Borůvka's algorithm

```
MST-BORUVKA(G, w)
1   A = ∅
2   for each vertex v ∈ G.V
3       MAKE-SET(v)
3   while |A| < n - 1
4       B = ∅
5       for each set S
6           let (u, v) be lightest edge incident to S, i.e. u ∈ S and v ∉ S
7           if such (u, v) exists
8               B = B ∪ {(u, v)}
8       for each edge  (u, v) ∈ B
9           if FIND-SET(u) ≠ FIND-SET(v)
10              A = A ∪ {(u, v)}
11              UNION(u, v)
12  return A
```



After *i* **while** iterations
each set size ≥ 2^i (or spanning)

Time O(*m*·log *n*)

Otakar Borůvka, *Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí*, Elektronický Obzor, 1926

Exam August 2005, Exercise 3(c)

Question 3(c)
Describe an algorithm, that given a weighted graph *G* and an edge *e* decides if *e* is contained in a minimum spanning tree for *G*.
It assumed that all edge weights are distinct. The running time must be O(*m*), where *m* is the number of edges in the graph.

Minimum spanning trees

Kruskal (1956) (many trees, sort edges increasing weight, union-find)	$O(m \cdot \log n)$
Prim-Jarník (1930) (one tree, priority queue with neighbor nodes)	$O(m \cdot \log n)$
	$O(m + n \cdot \log n)$ (Fibonnaci heaps [CLRS, 3rd Ed. Chapter 19] (1984))
Borůvka (1926) (many trees, contraction of trees)	$O(m \cdot \log n)$
Fredman, Tarjan (1984) (Borůvka (1926) + Fibonnaci heaps)	$O(m \cdot \log^* n)$
Chazelle (1997)	$O(m \cdot \alpha(m,n))$
Pettie, Ramachandran (2000)	? (optimal deterministic comparison based)
Karger, Klein, Tarjan (1995) (Randomized) Fredman, Willard (1994) (RAM)	$O(m)$

J. B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem,* AMS, 1956, doi: 10.1090/S0002-9939-1956-0078686-7

V. Jarník, *O jistém problému minimálním*, Práce Moravské Přírodovědecké Společnosti, 1930, hdl: 10338.dmlcz/500726

R. C. Prim, *Shortest connection networks and some generalizations*, Bell System Technical Journal, 1957 doi: 10.1002/j.1538-7305.1957.tb01515.x

Otakar Borůvka, *Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí*, Elektronický Obzor, 1926

M. L. Fredman, R. E. Tarjan, *Fibonacci heaps and their uses in improved network optimization algorithms,* J. ACM, 1987, doi: 10.1145/28869.28874

Bernard Chazelle, *A Faster Deterministic Algorithm for Minimum Spanning Trees*, FOCS, 1997, doi 10.1109/SFCS.1997.646089

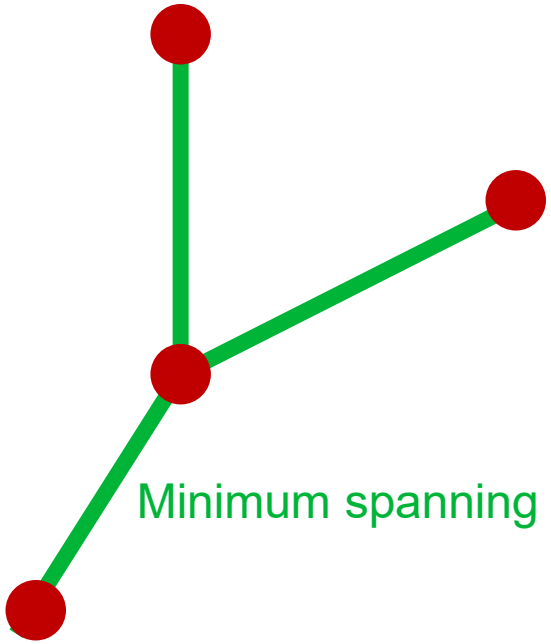
Seth Pettie, Vijaya Ramachandran, *An Optimal Minimum Spanning Tree Algorithm,* ICALP, 2000, doi: 10.1007/3-540-45022-X_6

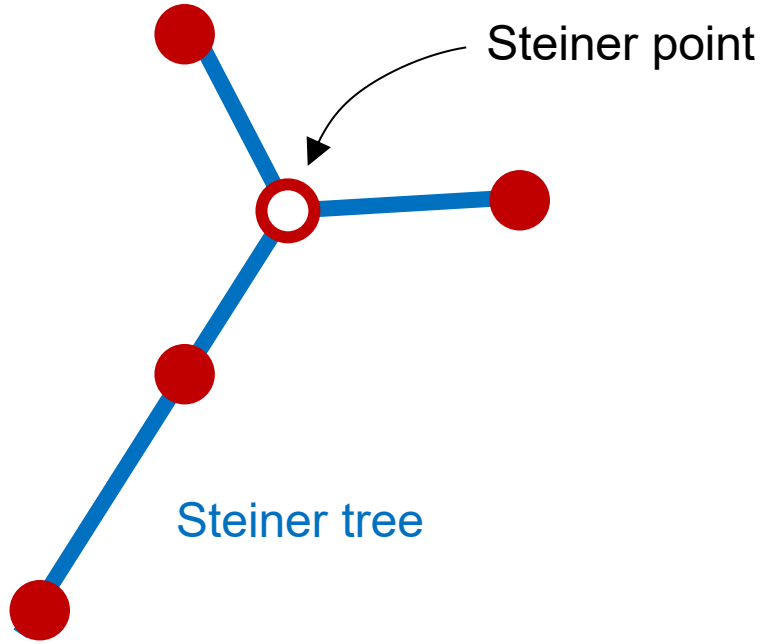
D.R. Karger, P.N. Klein, R.E. Tarjan, *A Randomized Linear-Time Algorithm to Find Minimum Spanning Trees*, J. ACM, 1995, doi: 10.1145/201019.201022

Fredman, Willard, *Trans-Dichotomous Algorithms for Minimum Spanning Trees and Shortest Paths*, JCSS, 1994, doi: 10.1016/S0022-0000(05)80064-9

Euclidian Steiner trees

Given a set of points, find a tree with minimum weight connecting all points, possibly adding **new points**





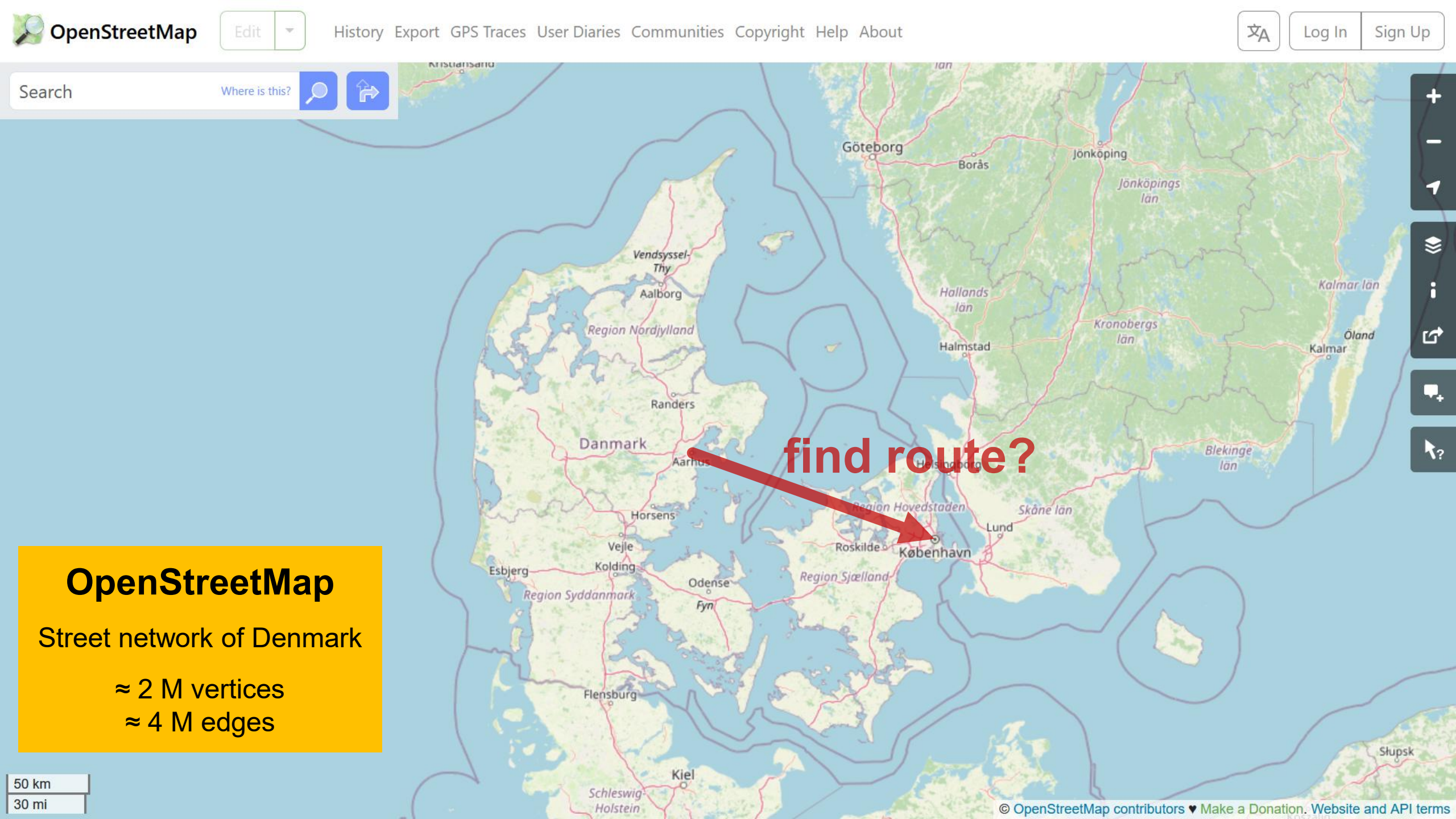
Conjecture (Steiner Ratio): $|\text{MST}| \leq \frac{2}{\sqrt{3}} \cdot |\text{Steiner Tree}|$

NP hard; polynomial-time approximation scheme (PTAS)

Jakob Steiner [1796-1863]

Algorithms and Data Structures

Single source shortest paths (SSSP)
[CLRS, Chapter 22]

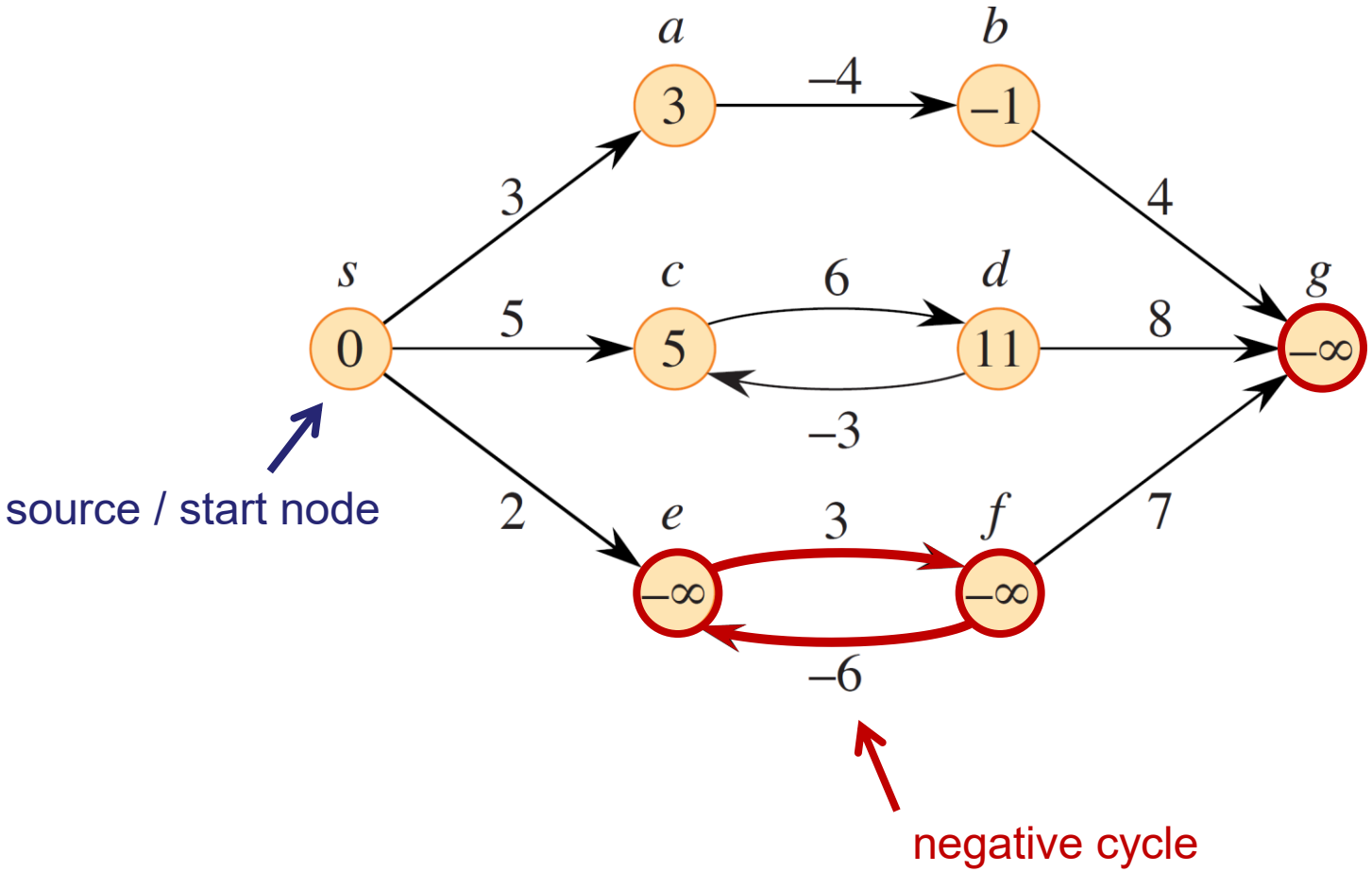


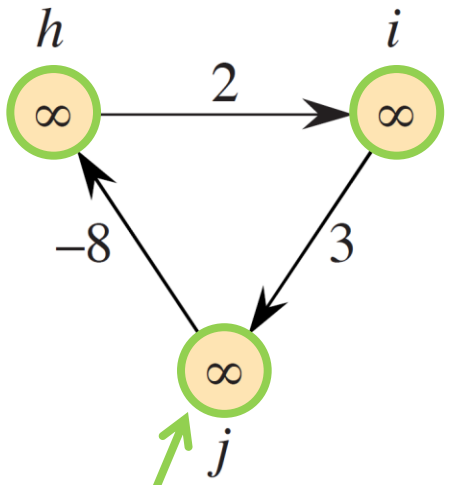
OpenStreetMap

Street network of Denmark

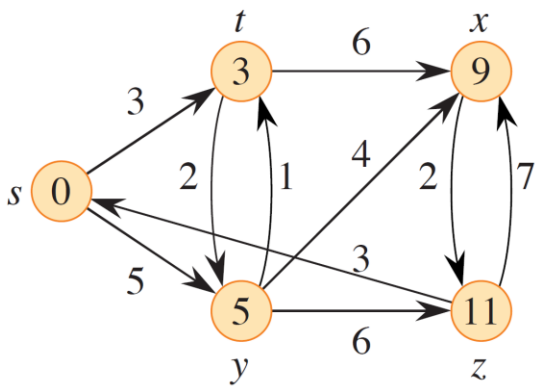
≈ 2 M vertices
≈ 4 M edges

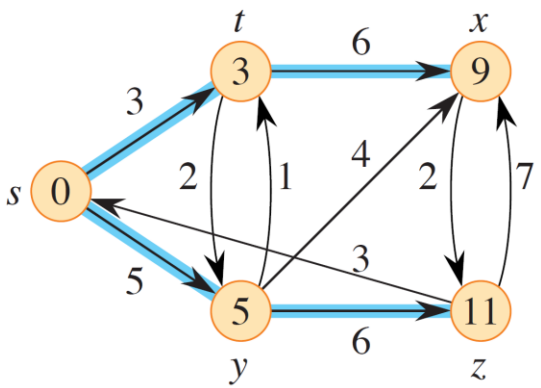
Shortest paths from *s*

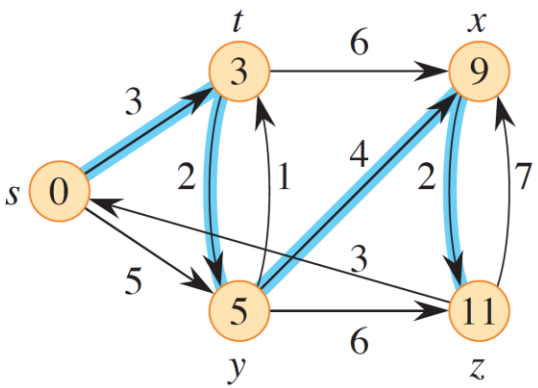




Shortest path trees



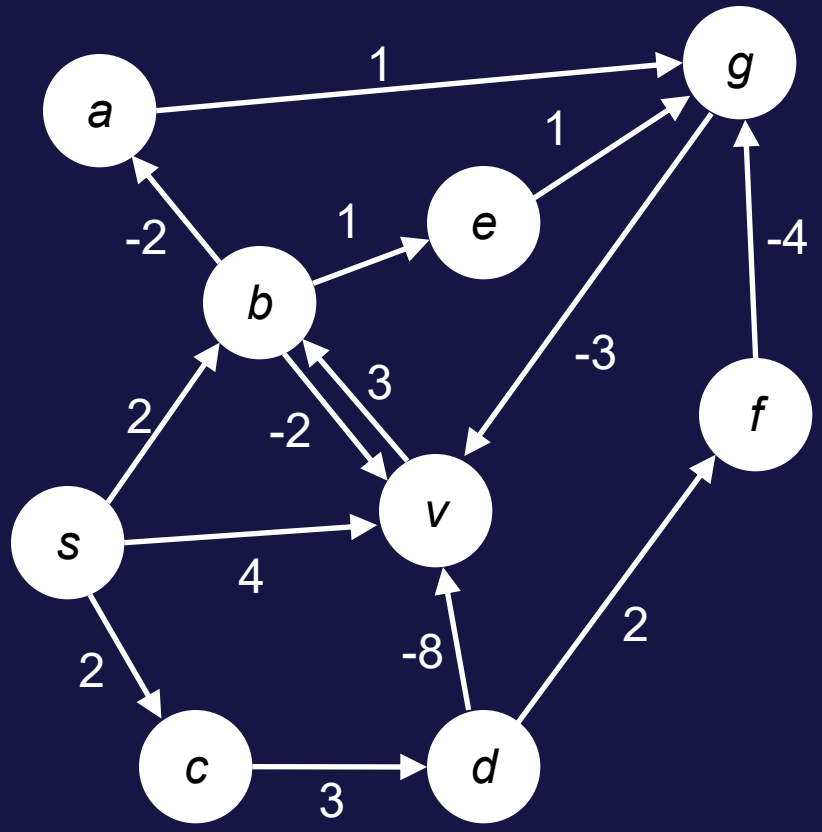




two different shortest path trees
representing paths from *s* of equal length

Length of a shortest path from *s* to *v* ?

a) -∞
b) -3
c) -1
d) 0
e) 4
f) +∞
g) Don't know



Shortest path estimates – initialization

INITIALIZE-SINGLE-SOURCE(G, s)

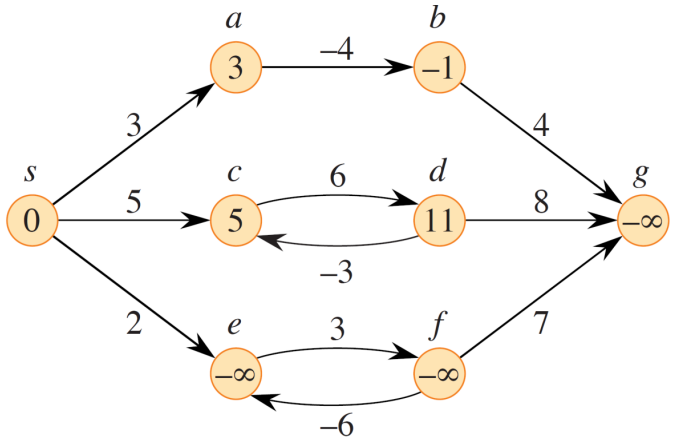
1 **for** each vertex $v \in G.V$

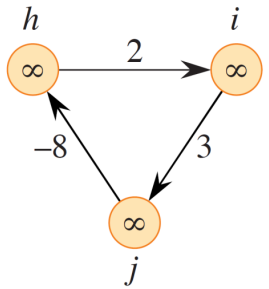
2 $\quad v.d = \infty$

3 $\quad v.\pi = \text{NIL}$

4 $s.d = 0$

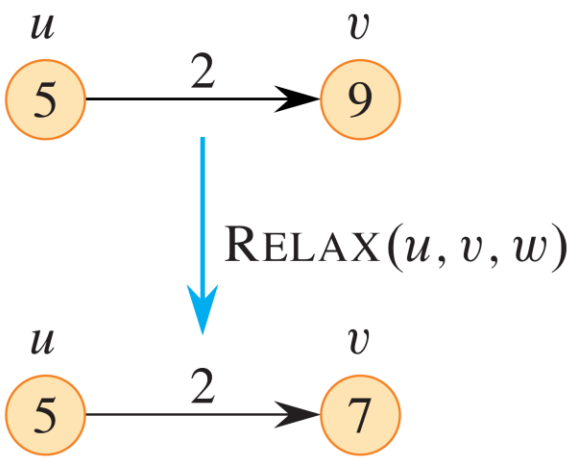
parent in shortest path tree



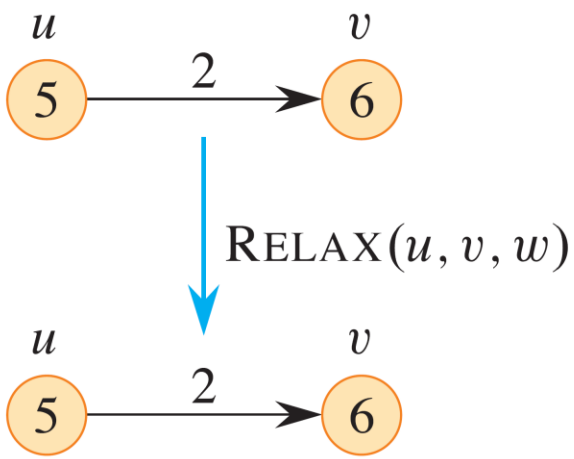


Relaxing edges

```
RELAX(u, v, w)
1   if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```



Shorter distance found to *v*
(update *v.d* and *v*.π = *u*)



No shorter distance
found to *v*

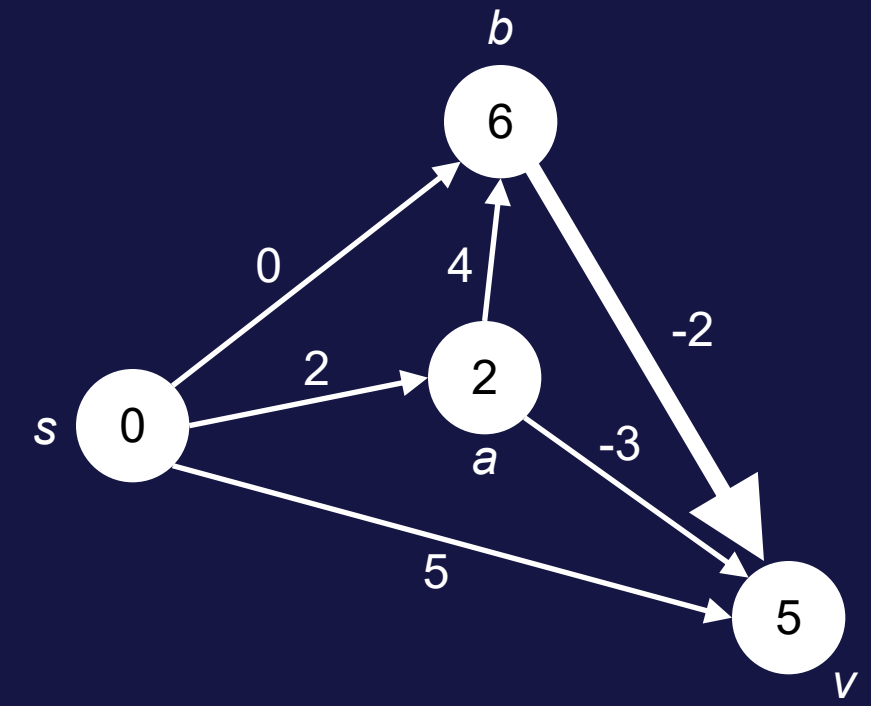
Invariant

If $v.d < \infty$, then a path exists from *s* to *v* with distance *v.d* and *v*.π as the second to last node

v.d after Relax(*b*, *v*, *w*) ?

(inside the nodes is the current *d* value)

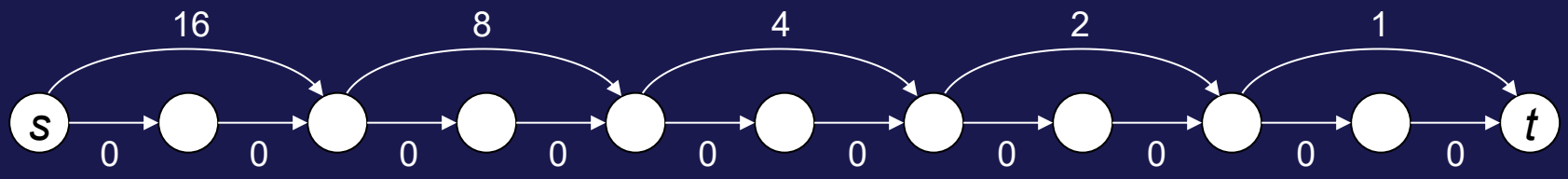
a) -2
b) -1
c) 0
d) 2
e) 4
f) 5
g) Don't know



Worst-case # calls to Relax ?

GREEDY-RELAX(G, w, s)
1 INITIALIZE-SINGLE-SOURCE(G, s)
2 **while** exist (u, v) : $v.d > u.d + w(u, v)$
3 RELAX(u, v, w)

a) 10
b) 15
c) 31
d) 93
e) Don't know



Bellman-Ford (supports negative edge weights)

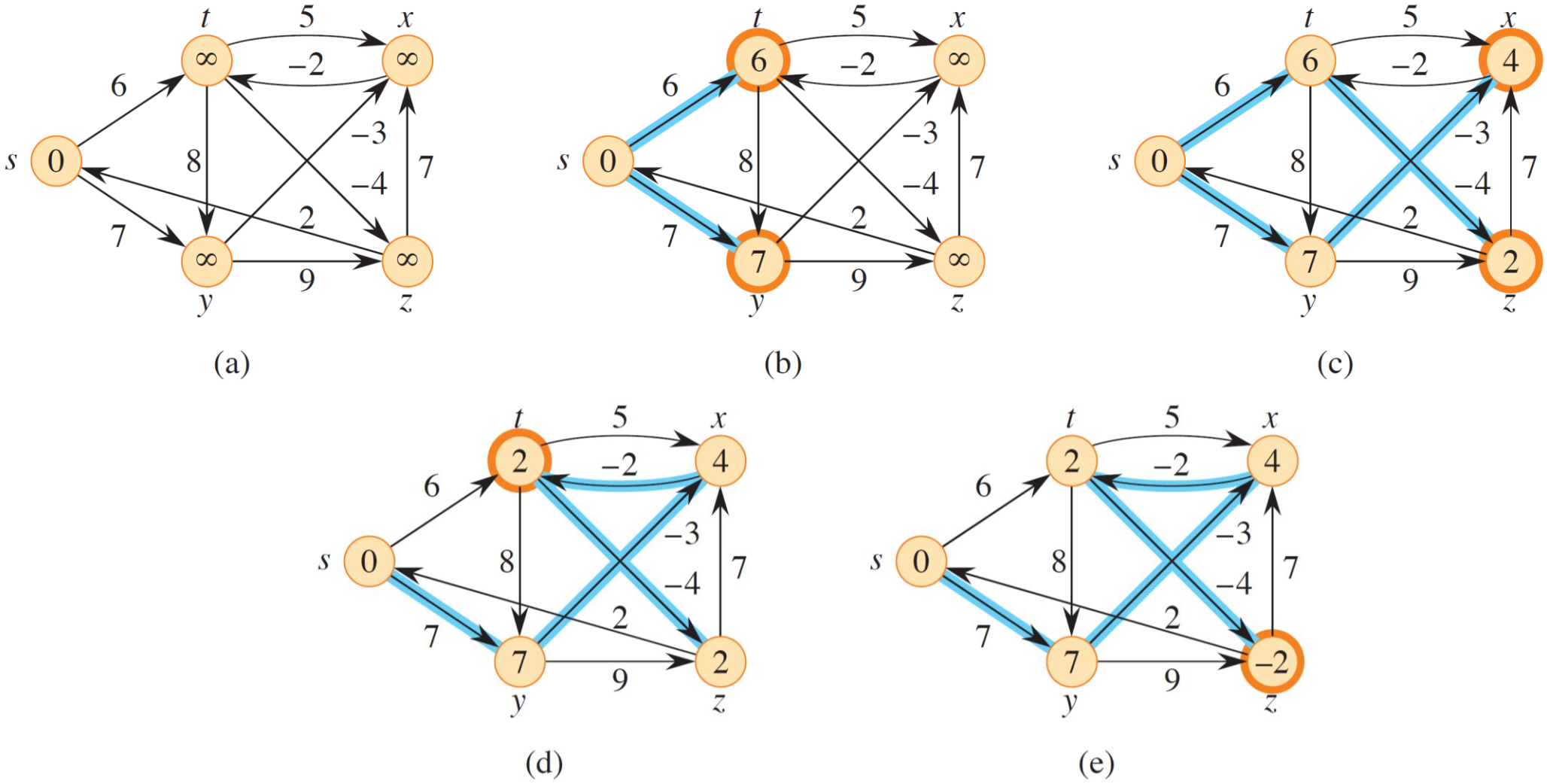
```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

check for negative cycle

Time $O(n \cdot m)$

Alfonso Shimbel, *Structure in communication nets,* Proceedings of the Symposium on Information Networks, 1955
Richard Bellman, *On a routing problem*, Quarterly of Applied Mathematics, 1958, doi: 10.1090/qam/102435
Lester R. Ford, Jr., *Network Flow Theory*, paper P-923, RAND Corporation, 1956

Bellman-Ford



Assume there exists a negative cycle $v_1, v_2, \ldots, v_k$ reachable from s

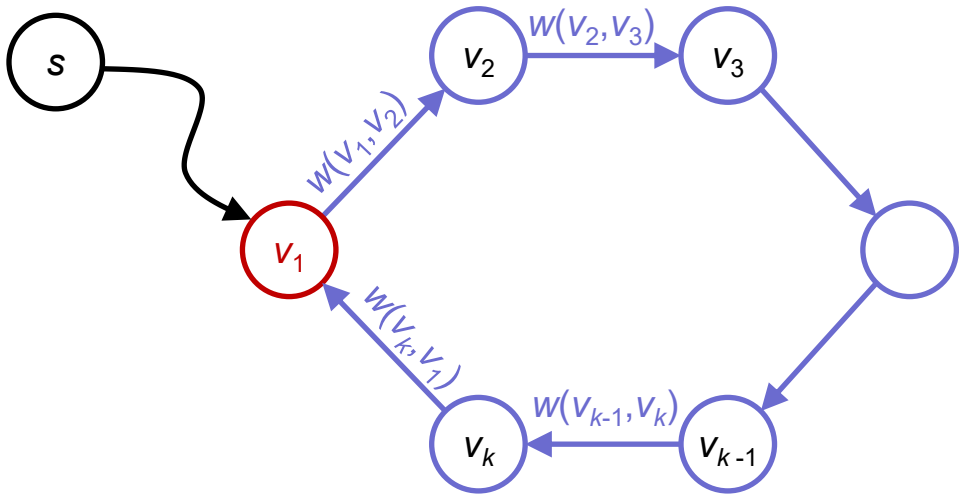
$$w(v_1,v_2) + w(v_2,v_3) + \cdots + w(v_{k-1},v_k) + w(v_k,v_1) < 0$$

and it is **not** possible to **RELAX** any edge on the cycle

$$d[v_{i+1}] \le d[v_i] + w(v_i,v_{i+1})$$

This implies the contradiction

$$d[v_1] \le d[v_k] + w(v_k,v_1) \le \big(d[v_{k-1}] + w(v_{k-1},v_k)\big) + w(v_k,v_1) \le$$
$$\cdots \le d[v_1] + \underbrace{w(v_1,v_2) + w(v_2,v_3) + \cdots + w(v_{k-1},v_k) + w(v_k,v_1)}_{< 0} < d[v_1]$$

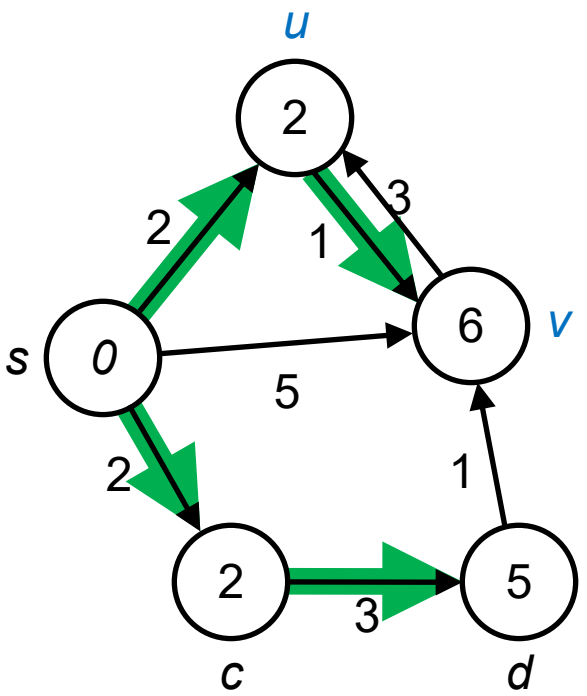


Theorem

Consider an (unknown) shortest path tree T with an edge (u,v).

Assume $d[u]$ is the shortest distance to u.

After RELAX(u,v,w) then $d[v]$ is the shortest distance to v.



⇓

Corollary

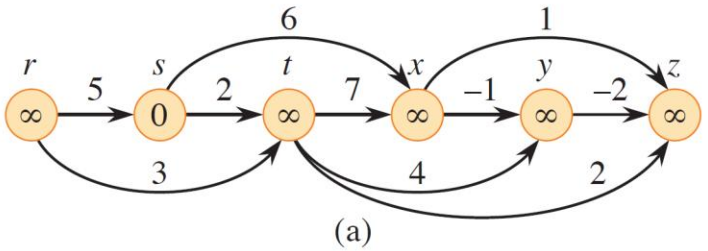
The Bellman-Ford algorithm has after relaxing all edges $n - 1$ times found the shortest distances

Shortest paths in acylic graphs

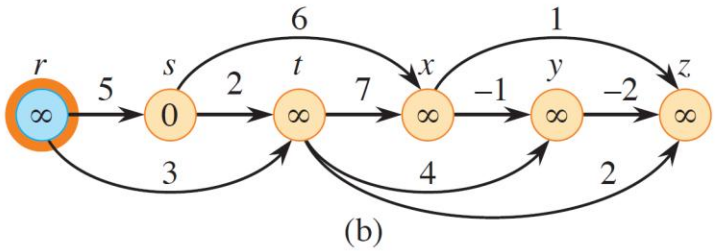
```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u ∈ G.V, taken in topologically sorted order
4       for each vertex v in G.Adj[u]
5            RELAX(u, v, w)
```

Time O($n + m$)

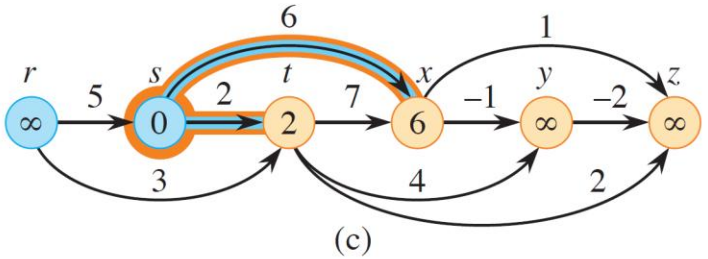
Acyclic graph



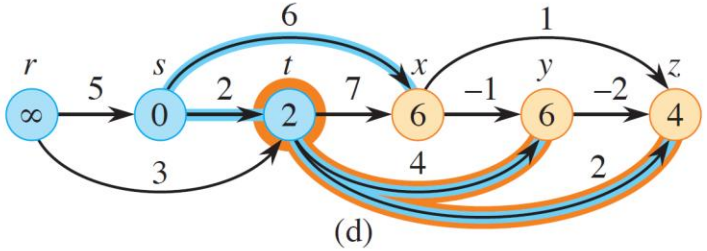
(a)



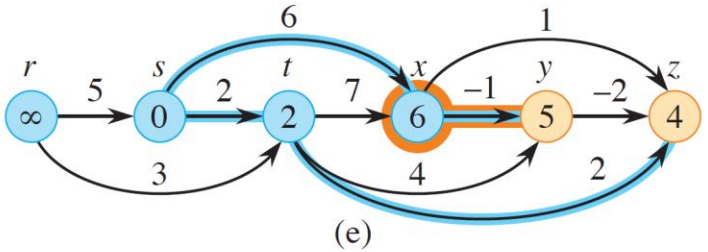
(b)



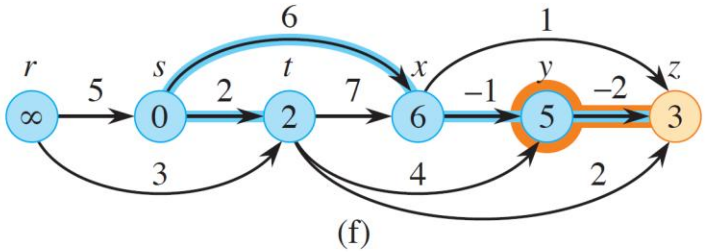
(c)



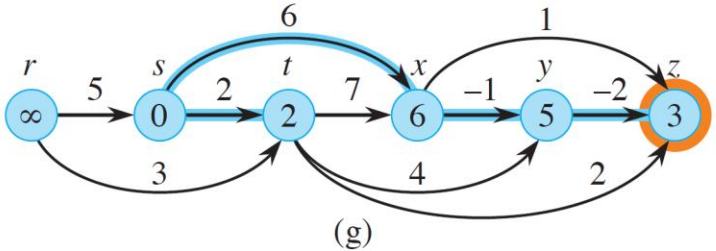
(d)



(e)



(f)



(g)

Dijkstra's algorithm (non-negative weights)

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = ∅
4  for each vertex u ∈ G.V
5      INSERT(Q, u)
6  while Q ≠ ∅
7      u = EXTRACT-MIN(Q)
8      S = S ∪ {u}
9      for each vertex v in G.Adj[u]
10         RELAX(u, v, w)
11         if the call of RELAX decreased v.d
12             DECREASE-KEY(Q, v, v.d)
```

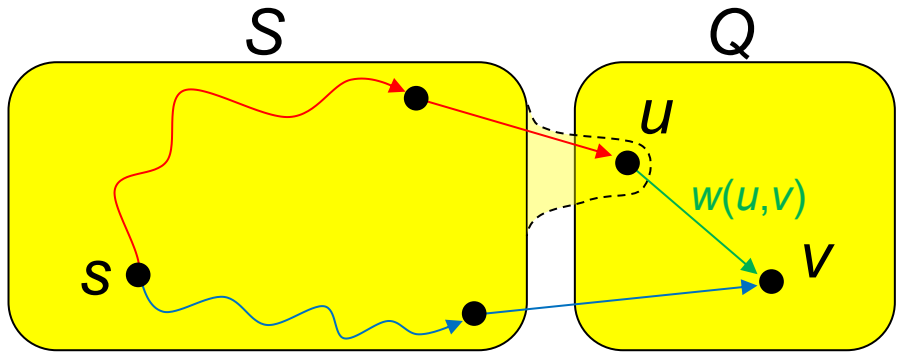
Update priority of *v* in Q

- Visits nodes in increasing distance from *s*
- *Q* = priority queue with *d* as priority

Invariants

i) *d*[*v*] = shortest distance from *s* to *v* only visiting **intermediate nodes in *S***

ii) $\forall p \in S, q \in Q : d[p] \leq d[q]$

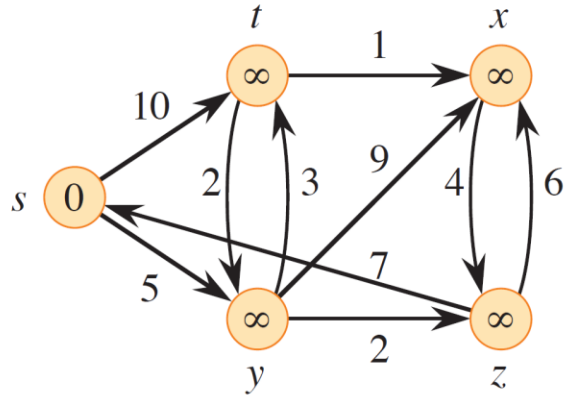


Time $O((n + m) \cdot \log n)$
or $O(n^2 + m)$

Edsger W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1959, doi: 10.1007/BF01386390

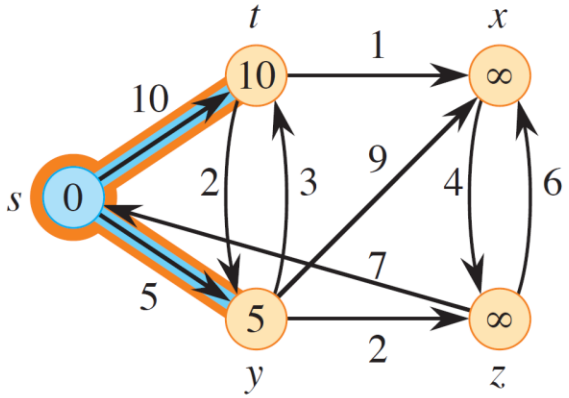
Dijkstra's algorithm

$Q = (s,0)\ (t,\infty)\ (x,\infty)\ (y,\infty)\ (z,\infty)$



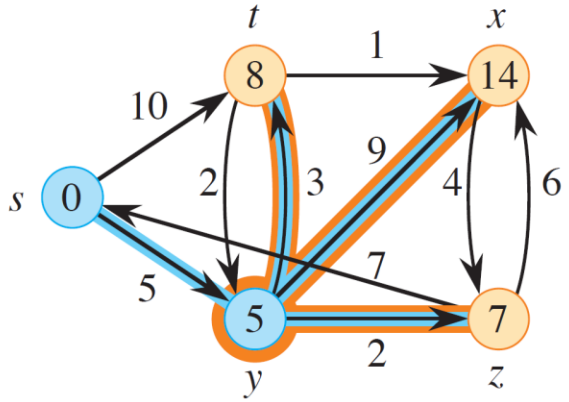
(a)

$Q = (y,5)\ (t,10)\ (x,\infty)\ (z,\infty)$



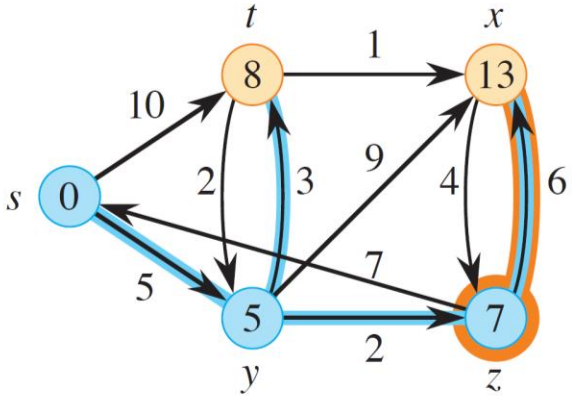
(b)

$Q = (z,7)\ (t,8)\ (x,14)$



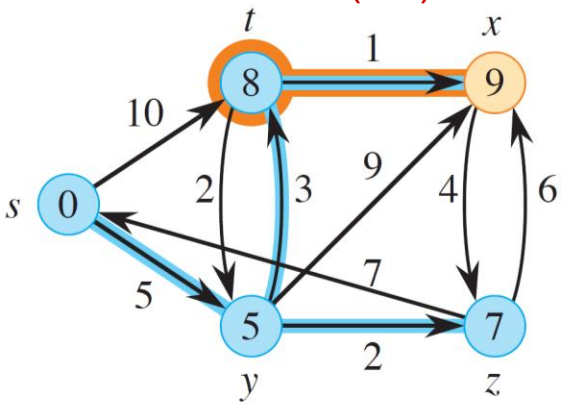
(c)

$Q = (t,8)\ (x,13)$



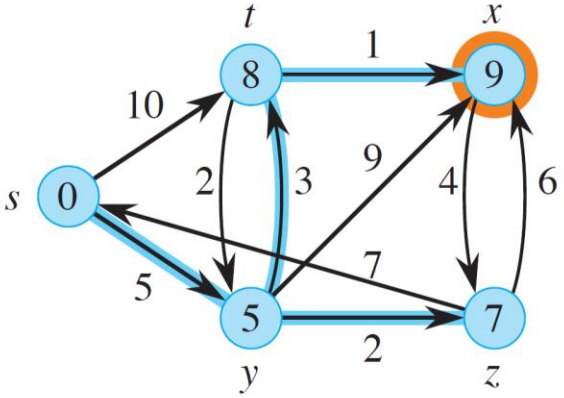
(d)

$Q = (x,9)$



(e)

$Q = \emptyset$



(f)

Dijkstra's algorithm using repeated insertions

```
DIJKSTRA(G, w, s)
1   INITIALIZE-SINGLE-SOURCE(G, s)
2   S = ∅
3   Q = {(0, s)}
4   while Q ≠ ∅
5       (δ, u) = EXTRACT-MIN(Q)
6       if δ = u.d
7           S = S ∪ { u }
8           for each vertex v ∈ G.Adj[u]
9               if v.d > u.d + w(u, v)
10                  v.d = u.d + w(u, v)
11                  v.π = u
12                  INSERT(Q, (v.d, v))
```

(priority, node) pair

skip outdated entries where δ > *u.d*

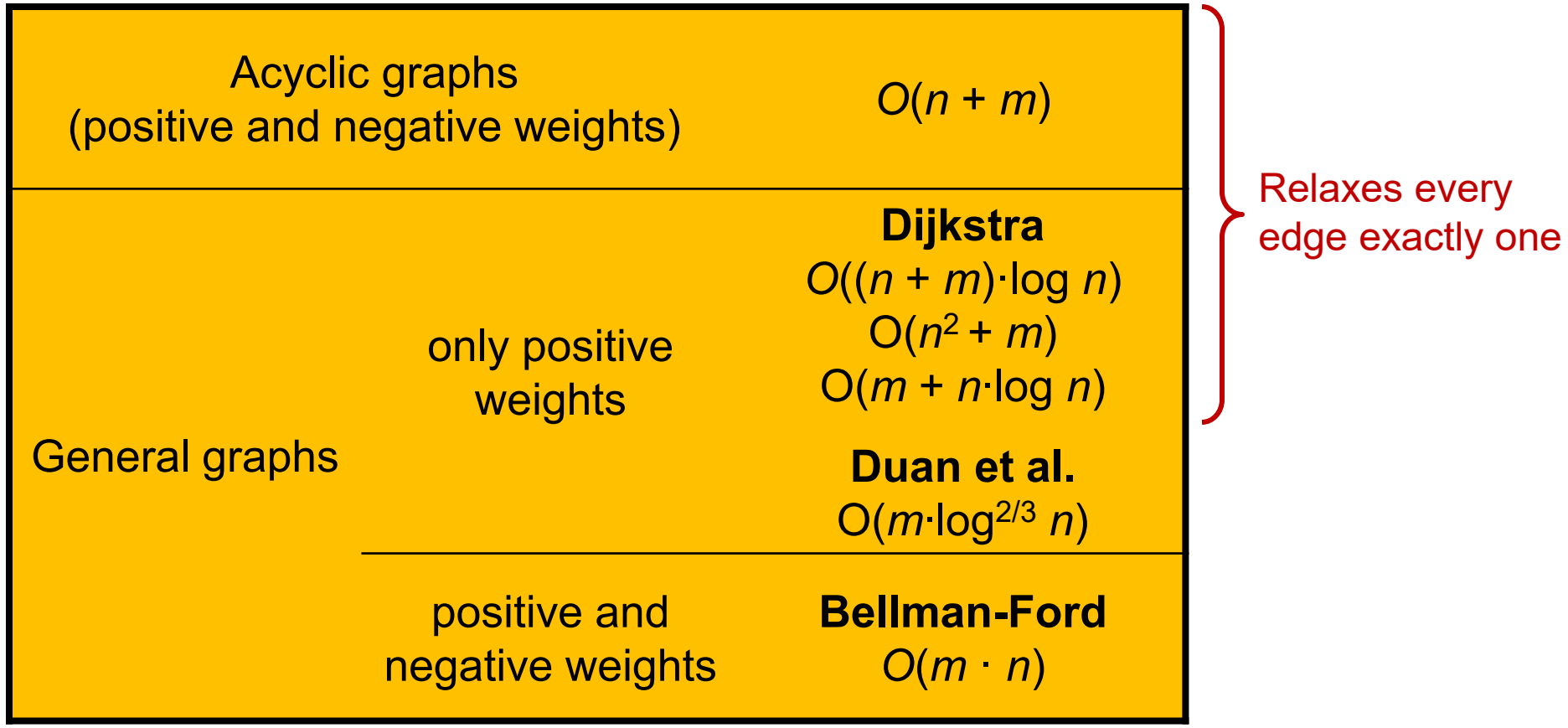
the same node can be inserted multiple times but with decreasing priorities (priority size increases from O(*n*) to O(*m*))

RELAX(*u*, *v*, *w*)

If priority queue supports INSERT and EXTRACT-MIN but not DECREASE-KEYo (like Java's PriorityQueue)

Time O((*n* + *m*)·log *n*)

Single source shortest paths (SSSP)



Edsger W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1959, doi: 10.1007/BF01386390

Alfonso Shimbel, *Structure in communication nets,* Proceedings of the Symposium on Information Networks, 1955

Richard Bellman, *On a routing problem*, Quarterly of Applied Mathematics, 1958, doi: 10.1090/qam/102435

Lester R. Ford, Jr., *Network Flow Theory*, paper P-923, RAND Corporation, 1956

Duan, Mao, Mao, Shu, Yin, *Breaking the Sorting Barrier for Directed Single-Source Shortest Paths*, STOC, 2025, doi: 10.1145/3717823.3718179

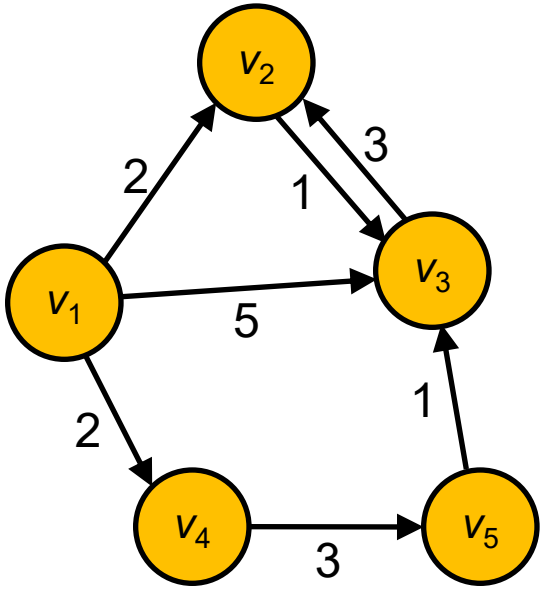
Algorithms and Data Structures

All pairs shortest paths (APSP)
[CLRS, Chapter 23]

All pairs shortest paths

If all weights positive – run Dijkstra's algorithm once with each v_i as source

$O(n \cdot m \cdot \log n)$



d_{ij}

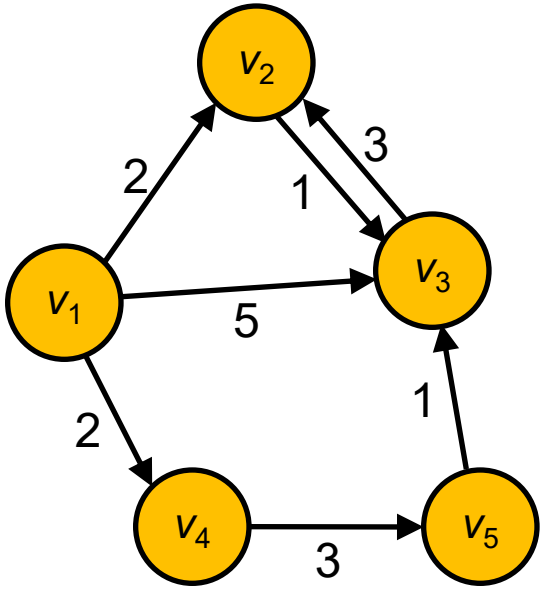
	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL		5	NIL	4
5	NIL	3	5	NIL	NIL

$\pi_{4,2}$?

a) 1
b) 2
c) 3
d) 4
e) 5
f) Don't know



d_{ij}

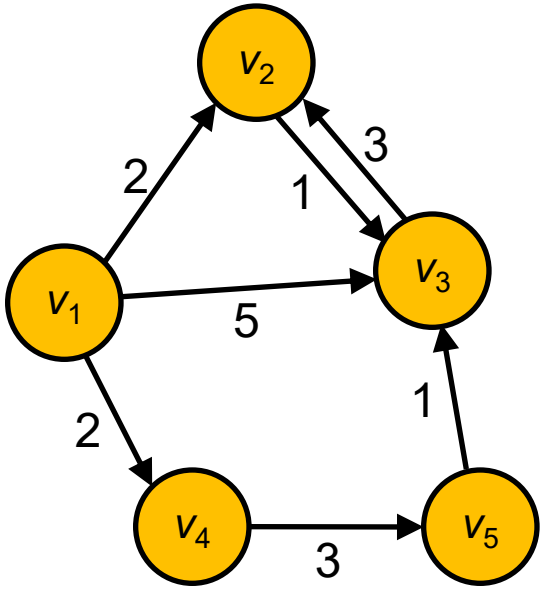
	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL	**3**	5	NIL	4
5	NIL	3	5	NIL	NIL

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  if i == j
2      print i
3  elseif π_ij == NIL
4      print "no path from" i "to" j "exists"
5  else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6      print j
```

Time O(n)



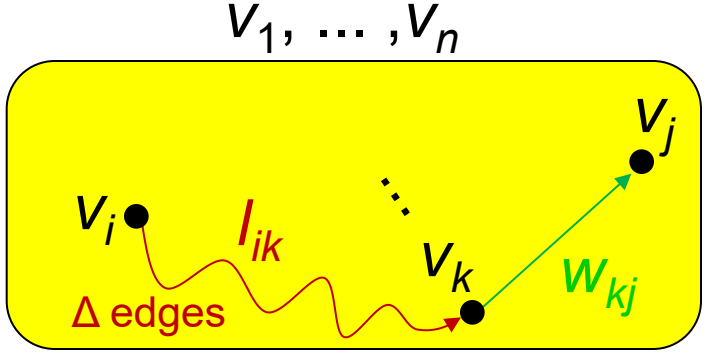
d_{ij}

	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL	3	5	NIL	4
5	NIL	3	5	NIL	NIL

```
EXTEND-SHORTEST-PATHS(L^(r-1), W, L^(r), n)
1  // Assume that the elements of L^(r) are initialized to ∞
2  for i = 1 to n
3      for j = 1 to n
4          for k = 1 to n
5              l_ij^(r) = min{l_ij^(r), l_ik^(r-1) + w_kj}
```



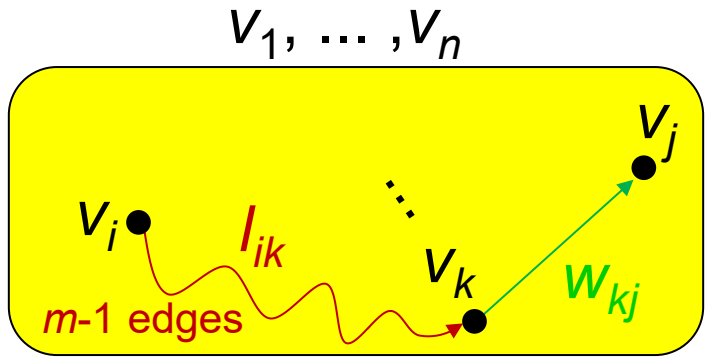
$l'_{ij} = \min_k (l_{ik} + w_{kj})$

W = weight matrix
L_{ij} = shortest distance from i to j for paths with Δ edges
L_{ij}' = shortest distance from i to j for paths with Δ+1 edges

Time $O(n^3)$

```
MATRIX-MULTIPLY(A, B, C, n)
1  for i = 1 to n
2      for j = 1 to n
3          for k = 1 to n
4              c_ij = c_ij + a_ik · b_kj
```

Time $O(n^3)$




```
SLOW-APSP(W, L^(0), n)
1  let L = (l_ij) and M = (m_ij) be new n × n matrices
2  L = L^(0)                                   ← diagonal 0, elsewhere ∞
3  for r = 1 to n − 1
4      M = ∞        // initialize M
5      // Compute the matrix "product" M = L · W
6      EXTEND-SHORTEST-PATHS(L, W, M, n)
7      L = M                                   ← Copy content of matrix
8  return L
```

W = weight matrix
$L_{ij}^{(m)}$ = shortest distance from i to j for paths with m edges

Time $O(n^4)$

$L_{1,3}^{(3)}$?

a) 0

b) 3

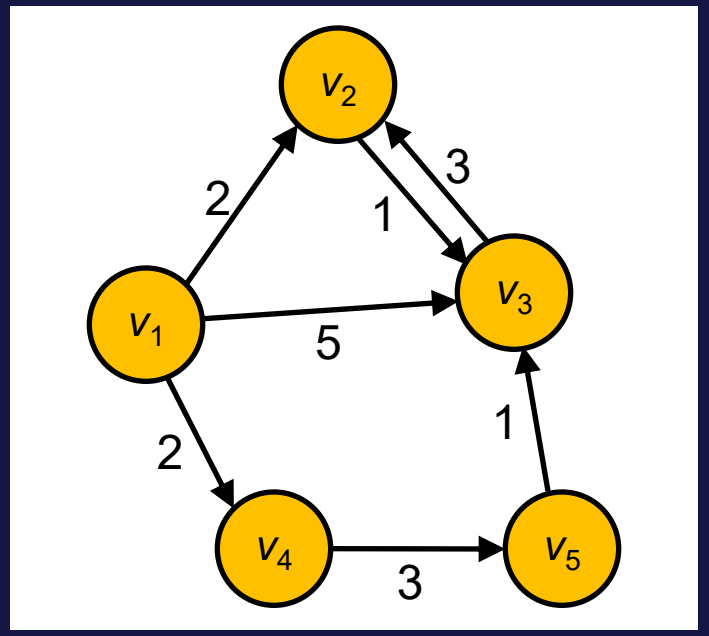
c) 5

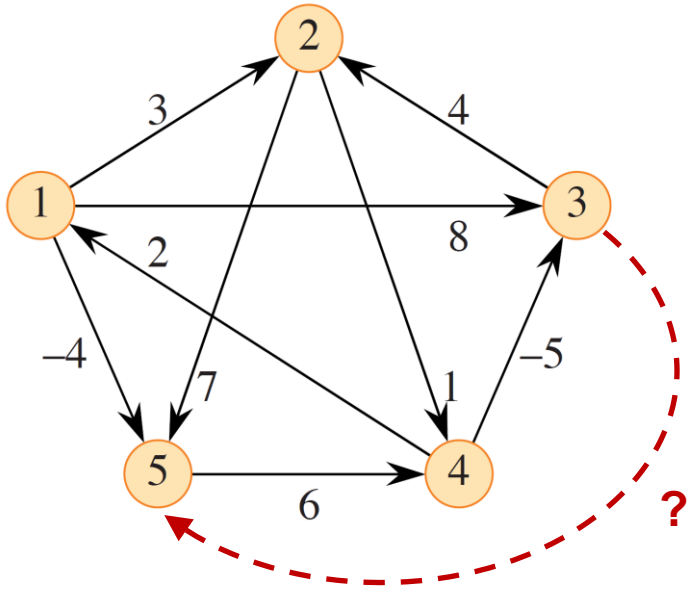
d) 6

e) 9

f) $+\infty$

g) Don't know



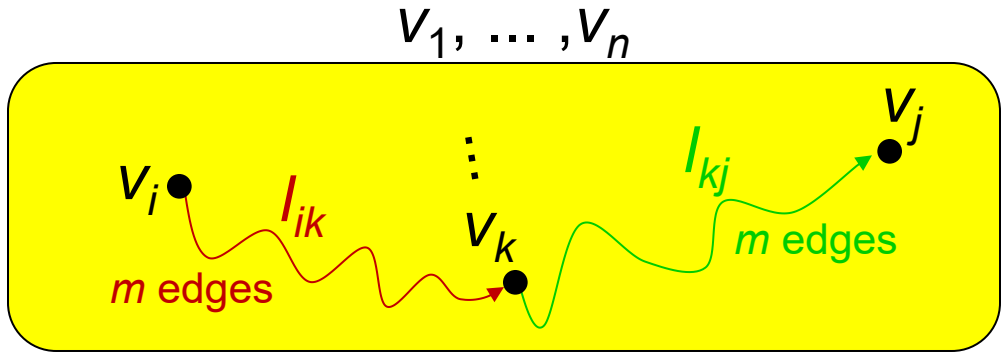


input graph

$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \qquad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \qquad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

```
FASTER-APSP(W, n)
1  let L and M be new n × n matrices
2  L = W          ← assumes diagonal 0
3  r = 1
4  while r < n − 1   ← Invariant: L = L^(r)
5      M = ∞
6      EXTEND-SHORTEST-PATHS(L, L, M, n)
7      r = 2r
8      L = M      ← Copy content of matrix
9  return L
```



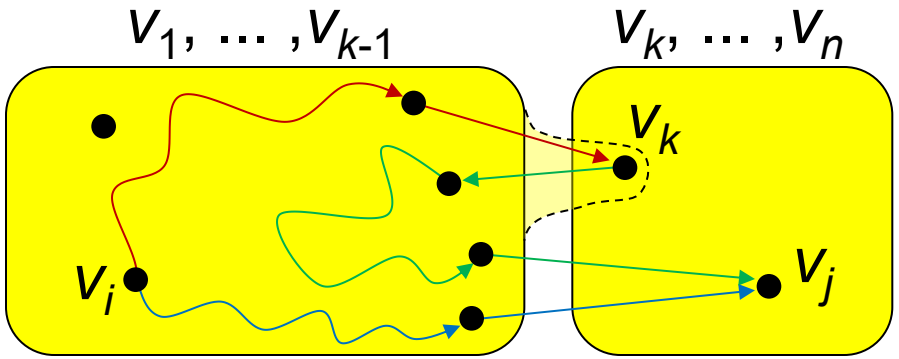
$l_{ij}^{(2m)} = \min_k (l_{ik}^{(m)} + l_{kj}^{(m)})$

W = weight matrix
$L_{ij}^{(m)}$ = shortest distance from i to j for paths with m edges

Time $O(n^3 \cdot \log n)$

Floyd-Warshall's algorithm

```
FLOYD-WARSHALL(W, n)
1  D^(0) = W
2  for k = 1 to n
3      let D^(k) = (d_ij^(k)) be a new n × n matrix
4      for i = 1 to n
5          for j = 1 to n
6              d_ij^(k) = min{d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1)}
7  return D^(n)
```



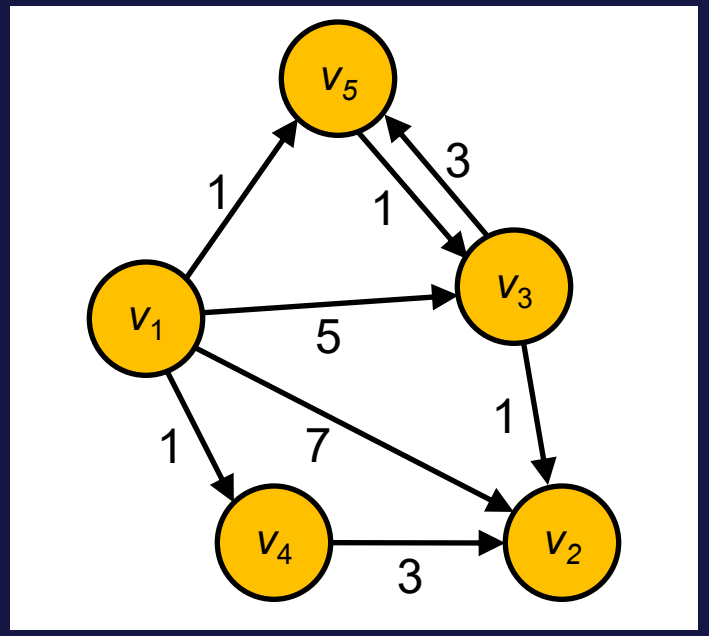
Time $O(n^3)$

$d_{ij}^{(k)}$ = shortest distance from *i* to *j* only using intermediate nodes 1..*k*

Robert W. Floyd, *Algorithm 97: Shortest Path,* Communications of the ACM, 1962, doi: 10.1145/367766.368168

$d_{1,2}^{(3)}$?

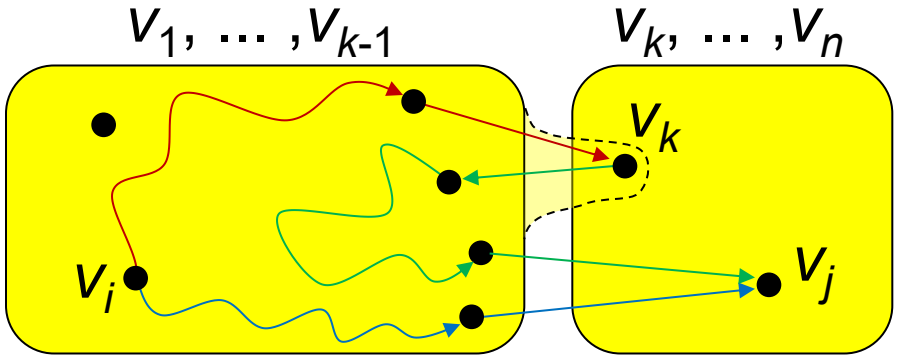
a) 3
b) 4
c) 6
d) 7
e) +∞
f) Don't know



Transitive closure

(Floyd-Warshall's algorithm simplified)

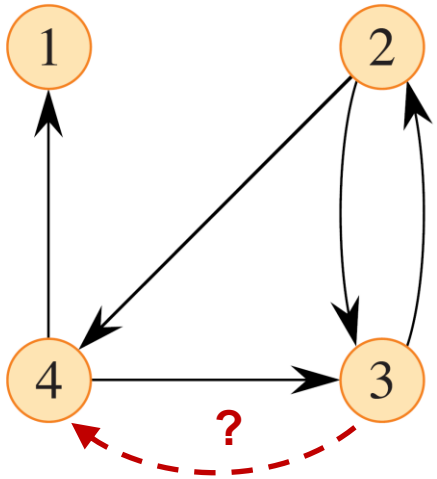
```
TRANSITIVE-CLOSURE(G, n)
1   let T^(0) = (t_ij^(0)) be a new n × n matrix
2   for i = 1 to n
3       for j = 1 to n
4           if i == j or (i, j) ∈ G.E
5               t_ij^(0) = 1
6           else t_ij^(0) = 0
7   for k = 1 to n
8       let T^(k) = (t_ij^(k)) be a new n × n matrix
9       for i = 1 to n
10          for j = 1 to n
11              t_ij^(k) = t_ij^(k−1) ∨ (t_ik^(k−1) ∧ t_kj^(k−1))
12  return T^(n)
```



Time $O(n^3)$

$t_{ij}^{(k)}$ = exists a path from *i* to *j* only using intermediate nodes 1..*k*

Transitive closure



input graph
+ self loops

$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix} \quad T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Johnson's APSP algorithm

- Sparse graphs, i.e., $m \ll n^2$, with positive and negative weights

Without negative weights ⇒ n x Dijkstra : $O(n \cdot (n \cdot \log n + m)) \ll O(n^3)$

With negative weights ⇒ Floyd-Warshall : $O(n^3)$

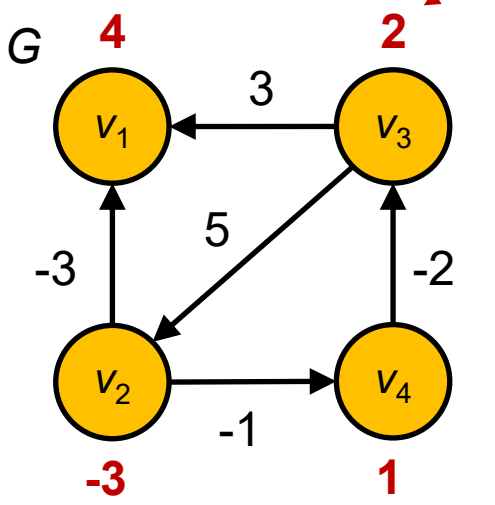
Johnson's algorithm

Idea Run Dijkstra's algorithm on a graph G' without negative edge weights, but having the same shortest paths as in G

Donald B. Johnson, *Efficient algorithms for shortest paths in sparse networks*, Journal of the ACM, 1977, doi: 10.1145/321992.321993

Height transformation

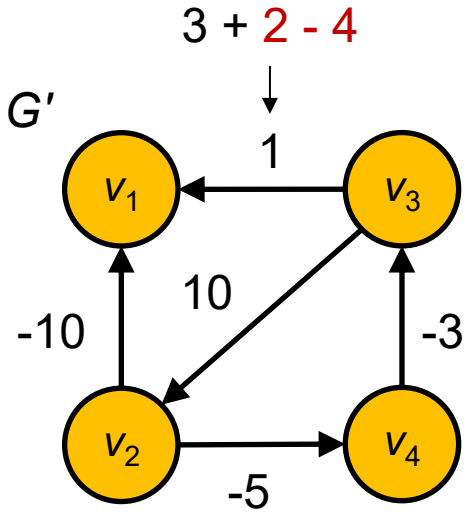
associate every node u an arbitrary height $h(u)$, a.k.a. heuristic



$$w'(u,v) = w(u,v) + h(u) - h(v)$$

$$l'(v_{i_1}\ v_{i_2} \cdots v_{i_k}) = l(v_{i_1}\ v_{i_2} \cdots v_{i_k}) + h(v_{i_1}) - h(v_{i_k})$$

Shortest path in G ⇔ shortest path in G'

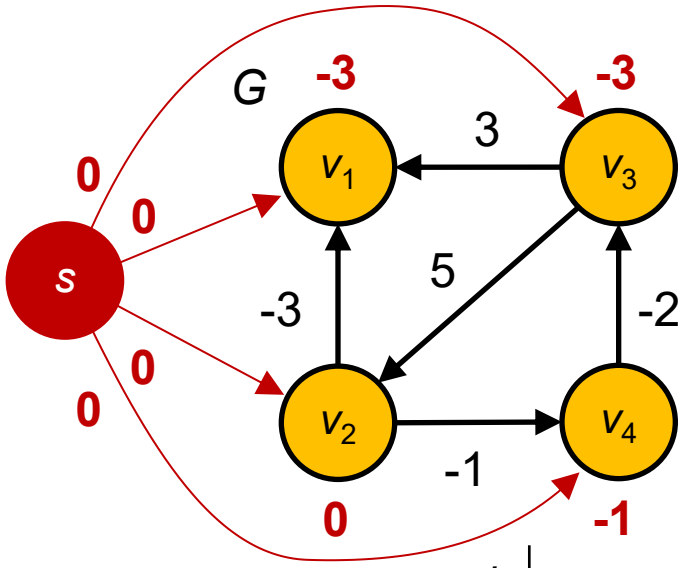


d	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

$d(v_2,v_3) = d'(v_2,v_3) + h(v_3) - h(v_2)$
$= -8 + 2 - -3$

d'	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-10	0	-8	-5
v_3	0	10	0	5
v_4	-3	7	-3	0

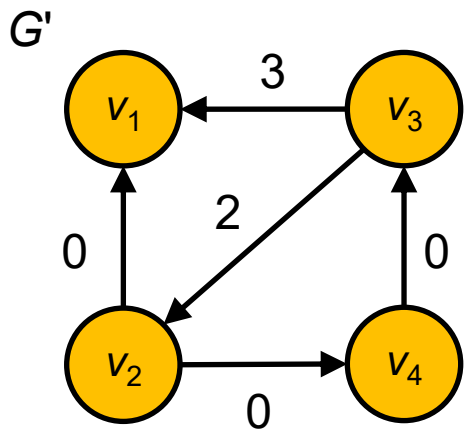
Johnson's algorithm



$$w'(u, v) = w(u, v) + h(u) - h(v)$$

$$w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$$

since $h(v) \leq h(u) + w(u, v)$



d	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

3 = 2 + 0 – (-1)

d'	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	0	0	0	0
v_3	2	2	0	2
v_4	2	2	0	0

- Add source node s with weight 0 edges to all nodes
- Let $h(u) = d_G(s, u)$ – SSSP Bellman-Ford in time $O(n \cdot m)$
- Run Dijkstra from each v_i in G' in time $O(n \cdot (n \cdot \log n + m))$
- Let $d_G(v_i, v_j) = d_{G'}(v_i, v_j) + h(v_j) - h(v_i)$

JOHNSON(G, w)

```
1  compute G', where G'.V = G.V ∪ {s},
       G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
       w(s, v) = 0 for all v ∈ G.V
2  if BELLMAN-FORD(G', w, s) == FALSE
3       print "the input graph contains a negative-weight cycle"
4  else for each vertex v ∈ G'.V
5            set h(v) to the value of δ(s, v)
                 computed by the Bellman-Ford algorithm
6       for each edge (u, v) ∈ G'.E
7            ŵ(u, v) = w(u, v) + h(u) − h(v)
8       let D = (d_uv) be a new n × n matrix
9       for each vertex u ∈ G.V
10           run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11           for each vertex v ∈ G.V
12                d_uv = δ̂(u, v) + h(v) − h(u)
13      return D
```

Time $O(n^2 \cdot \log n + n \cdot m)$

Shortest paths in graphs

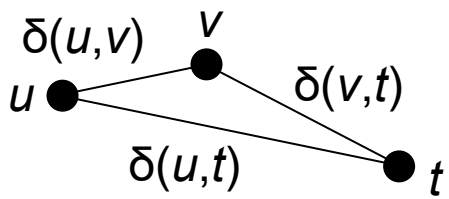
		Single source shortest paths (SSSP)	All pairs shortest paths (APSP)
Acyclic graphs (positive and negative weights)		$O(n + m)$	$O(n \cdot (n + m))$
General graphs	Only positive weights	**Dijkstra** $O((n + m) \cdot \log n)$ (1) $O(n \cdot \log n + m)$ (2) $O(n^2)$ (3) **Duan et al.** $O(m \cdot \log^{2/3} n)$	***n* x Dijkstra** $O(n^2 \cdot \log n + n \cdot m)$ (2) $O(n^3)$ (3)
	Positive and negative weights	**Bellman-Ford** $O(n \cdot m)$	**Floyd-Warshall** $O(n^3)$ **Johnson** $O(n^2 \cdot \log n + n \cdot m)$ (2)

Different priority queues: (1) binary heap, (2) Fibonacci heap, (3) array

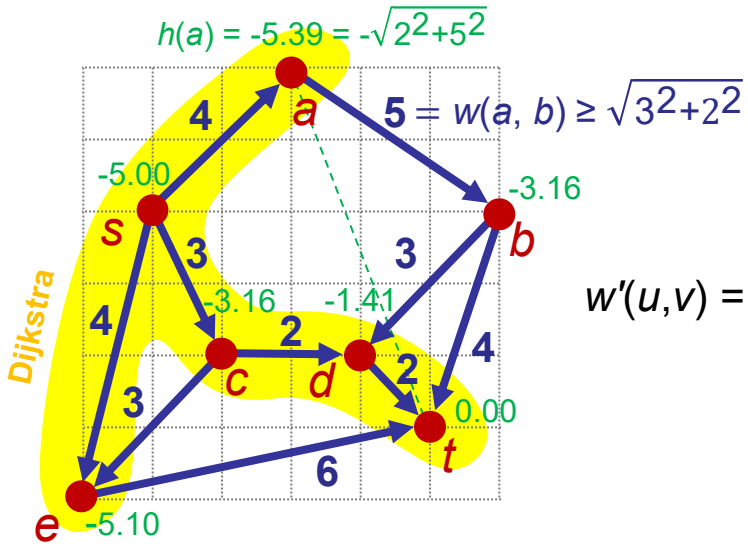
not curriculum

A*-algorithm – shortest path $s \rightsquigarrow t$

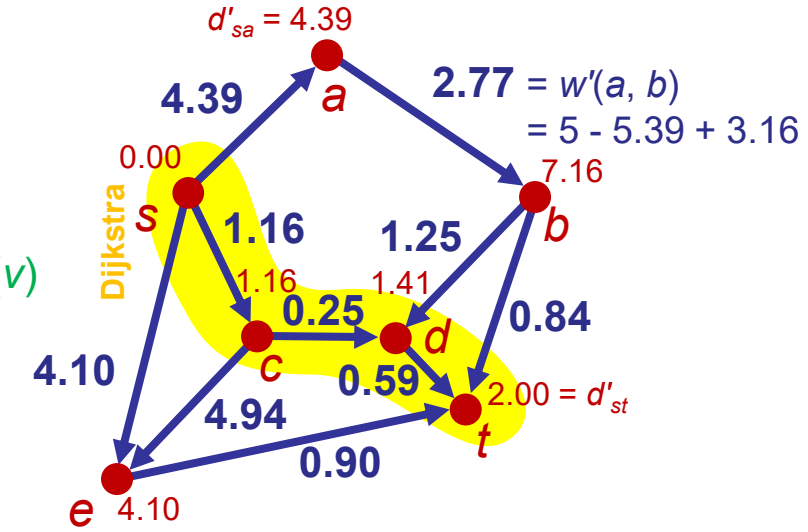
- Assume each node u has a position (u_x, u_y) and $w(u, v) \geq \delta(u, v) = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2}$
- $h(u) = -\delta(u, t) \Rightarrow w'(u, v) = w(u, v) + h(u) - h(v) \geq \delta(u, v) - \delta(u, t) + \delta(v, t) \geq 0$
- $d_{st} = d'_{st} - h(s) + h(t) = 2.00 + 5.00 - 0.00 = \mathbf{7.00}$



Triangle inequality



$w'(u,v) = w(u,v) + h(u) - h(v)$

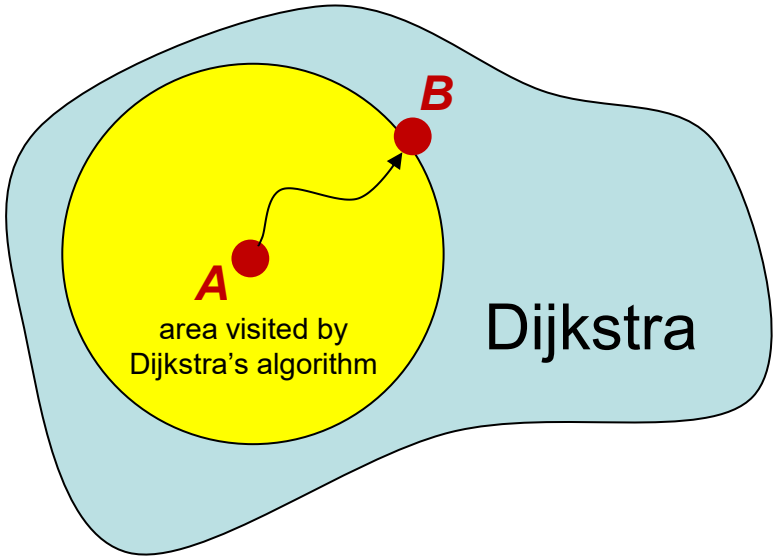


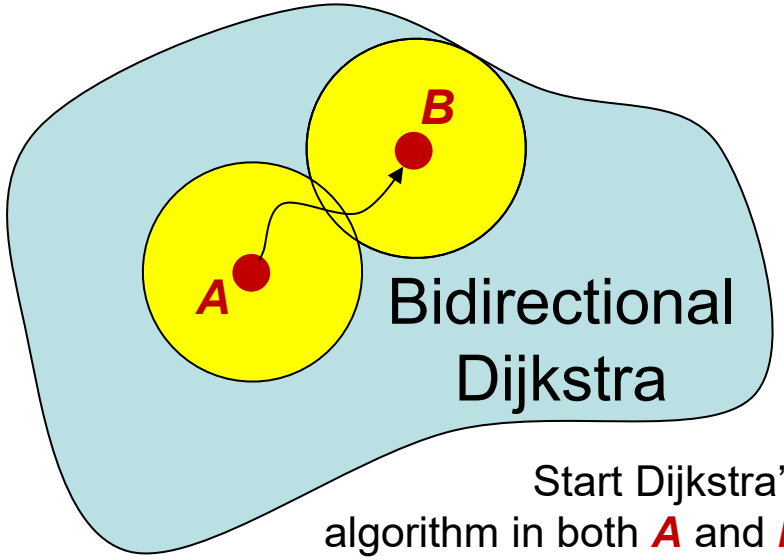
In practice A* reduces the number of nodes visited by Dijkstra's algorithm

Youtube: Compare A* with Dijkstra algorithm

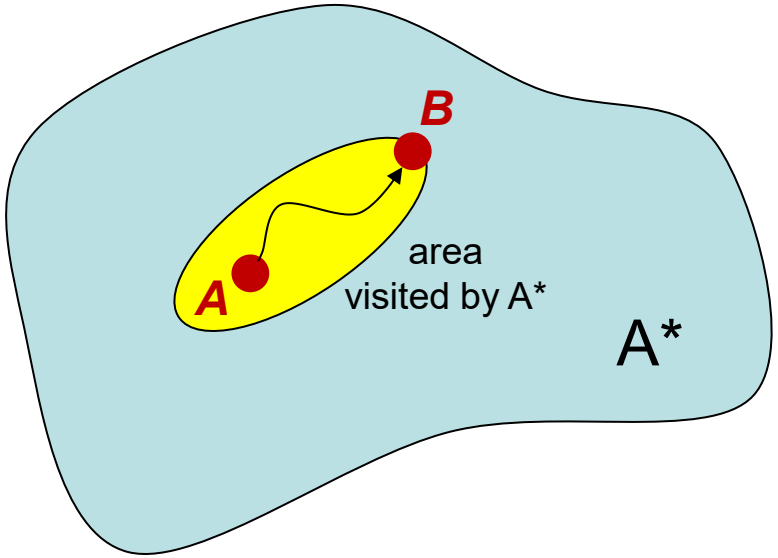
Hart, Nilsson, Raphael, *A Formal Basis for the Heuristic Determination of Minimum Cost Paths*
IEEE Transactions on Systems Science and Cybernetics, 1968, doi: 10.1109/TSSC.1968.300136

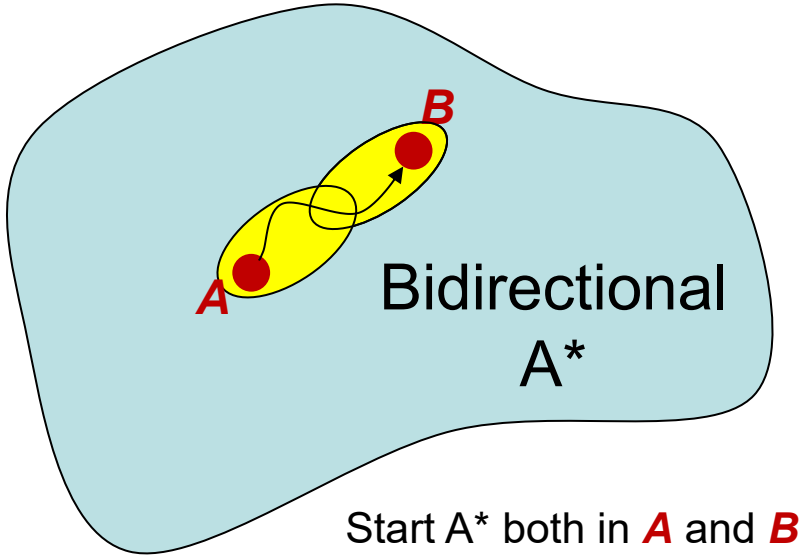
not curriculum





Start Dijkstra's
algorithm in both ***A*** and ***B***





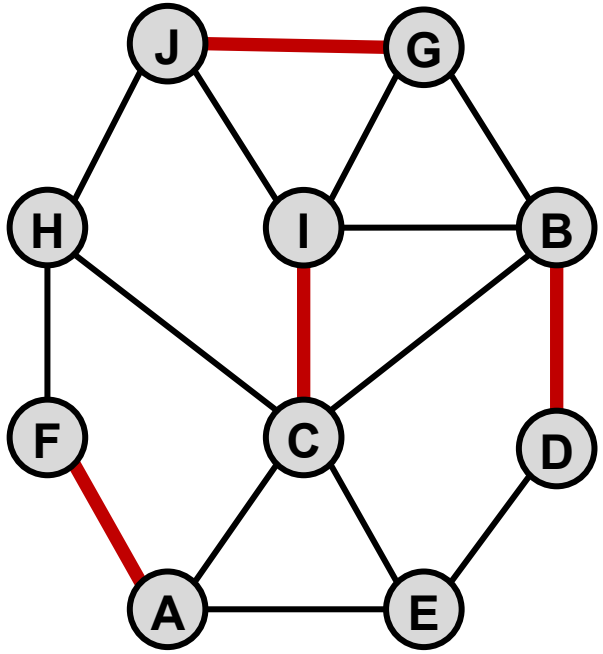
Start A* both in ***A*** and ***B***

not curriculum

Algorithms and Data Structures

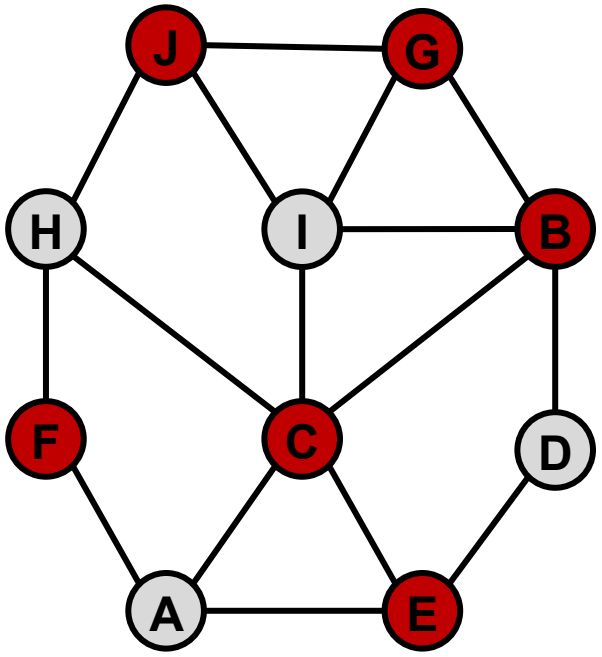
Matchings in graphs, maximum independent set, minimum vertex cover
[CLRS, Chapters 24.3, 25.2, 34.4, 35.1]

Matching



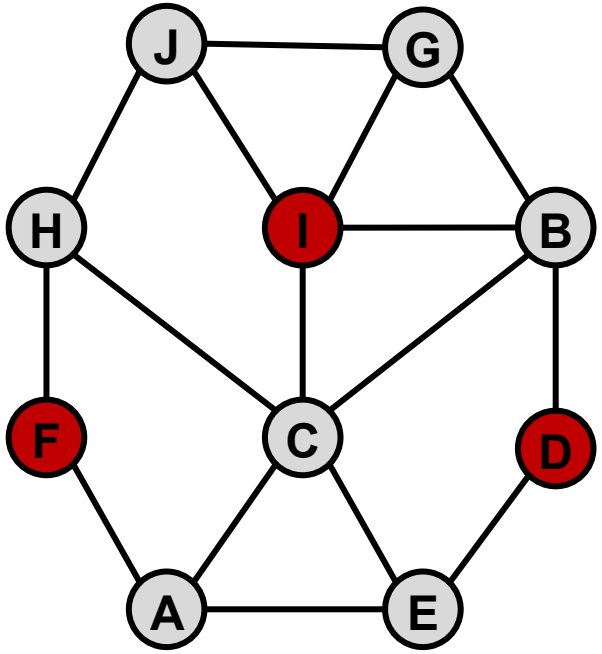
Subset of edges, where each node is part of at most one edge of the matching

Vertex cover



Subset of nodes (a.k.a. vertices), where all edges in the graph contain at least one node from the cover

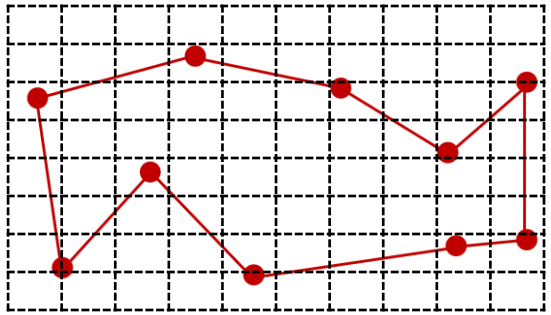
Independent set



Subset of nodes, where no pair of nodes are connected by an edge

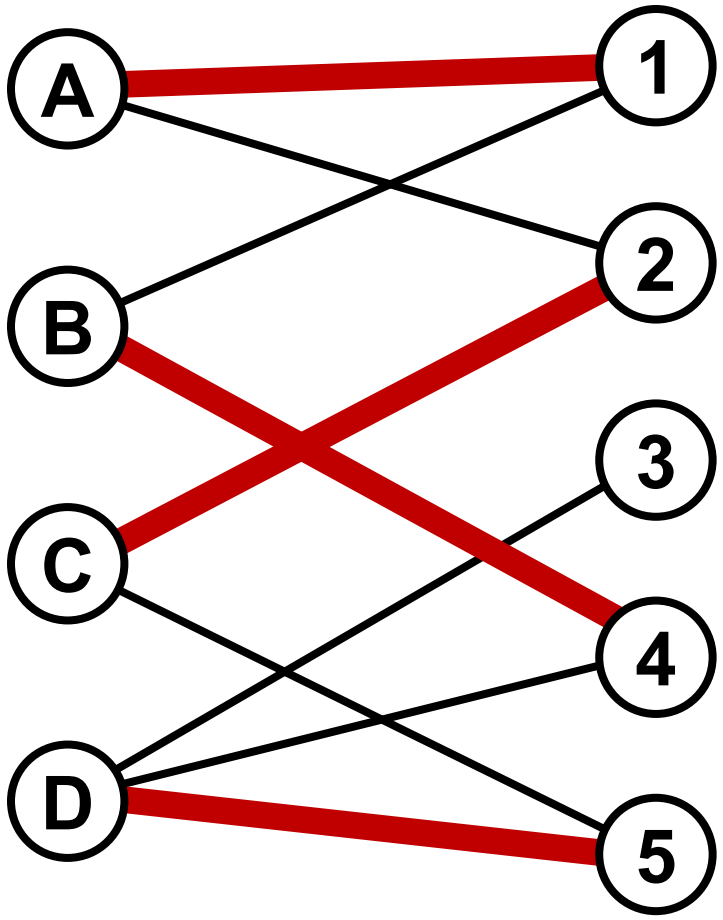
Not all graph problems are equally easy…

- Some problems can be solved efficiently in polynomial time
  DFS, **BFS**, **shortest paths**, **topological sorting**, **matchings**…
- Many problems are NP-hard
  - Exact solutions likely require exponential time
  - Some can be approximated arbitrary well
    (PTAS = ”polynomial time approximation scheme”)
    - e.g. **Euclidian traveling salesman** (Gödel Price 2010)



  - Some can only be approximated up to a constant (APX-hard)
    - e.g. **minimum vertex cover**
  - Some cannot be approximated within a constant efficiently
    - e.g. **maximum independent set**

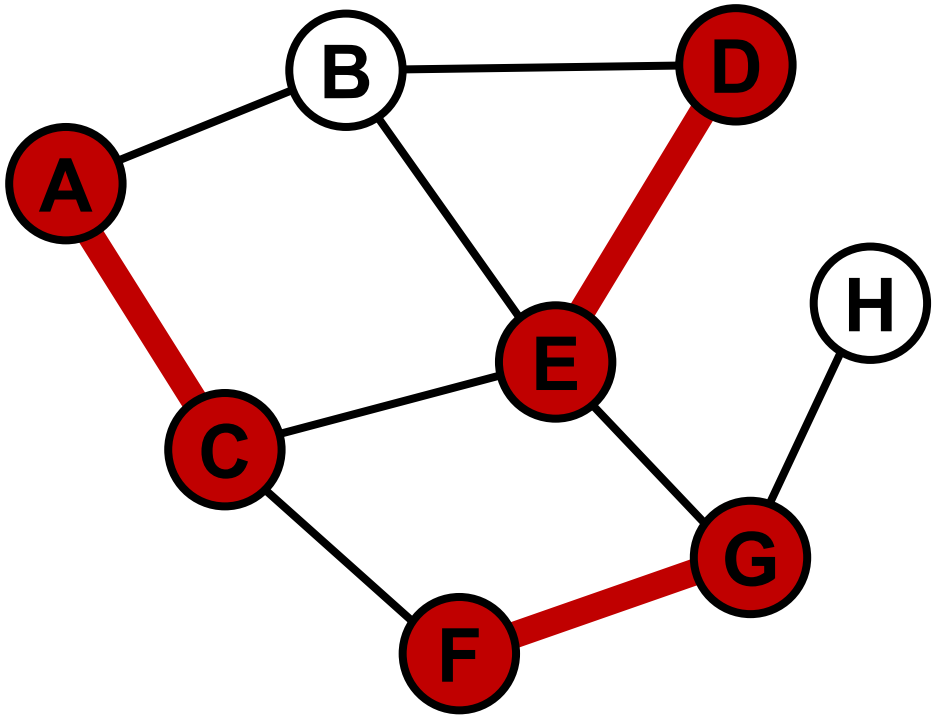
Matchings in bipartite graphs (special case)



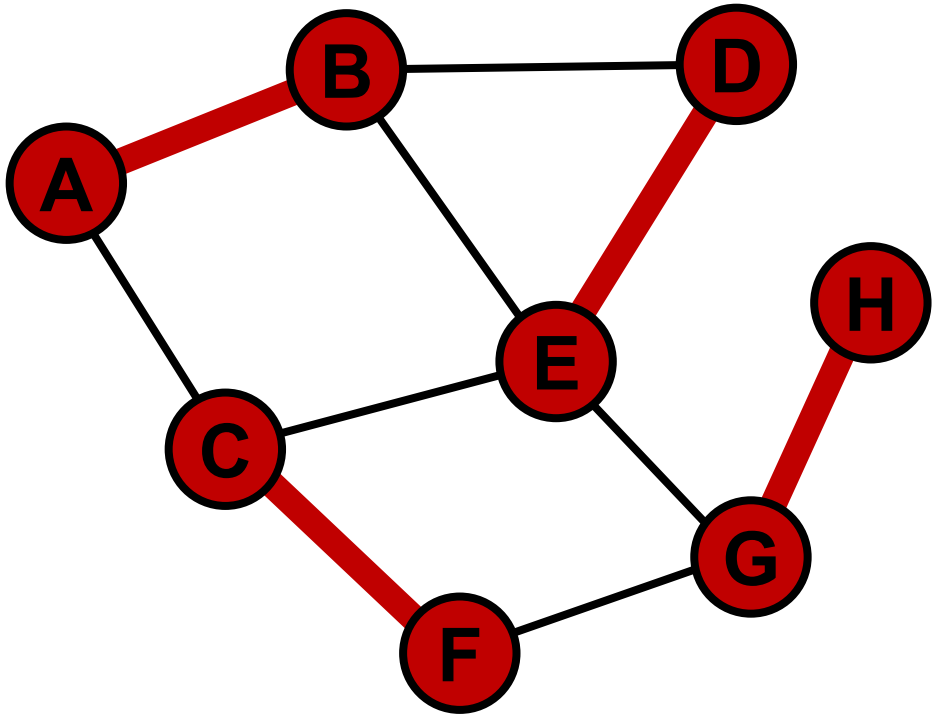
Examples

- **Busses vs routes**
  (edge = possible, sufficient capacity, range, electric vs non-electric)
- **People vs tasks**
  (can a task be handled by a person)
  Does there exist a matching where all people get assigned a task?
- **Jobs vs applications**
  (e.g. medicine students and hospitals)

Maximal and maximum matchings



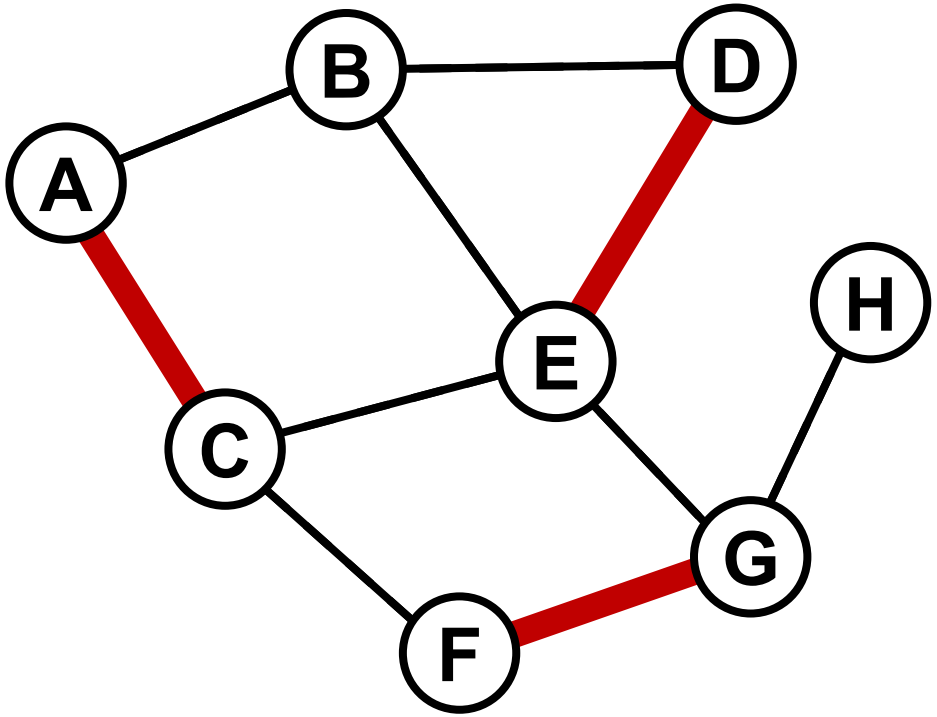
Maximal
cannot be extended



Maximum
largest possible matching

Lemma |Maximum matching| / 2 ≤ |maximal matching| ≤ |maximum matching|

Maximal matching construction

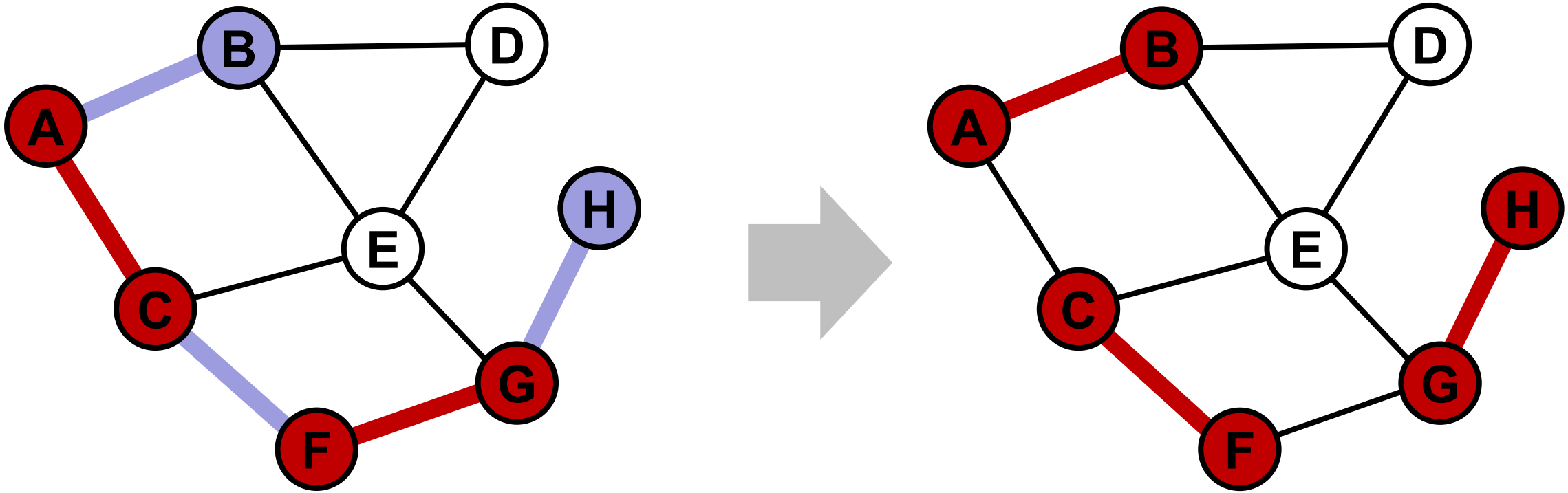


Greedy algorithm

Continue adding arbitrary edges to the matching as long as possible

Time $O(n + m)$

Construction of a maximum matching – augmenting paths



Path between two unmatched nodes, where every second edge in matching

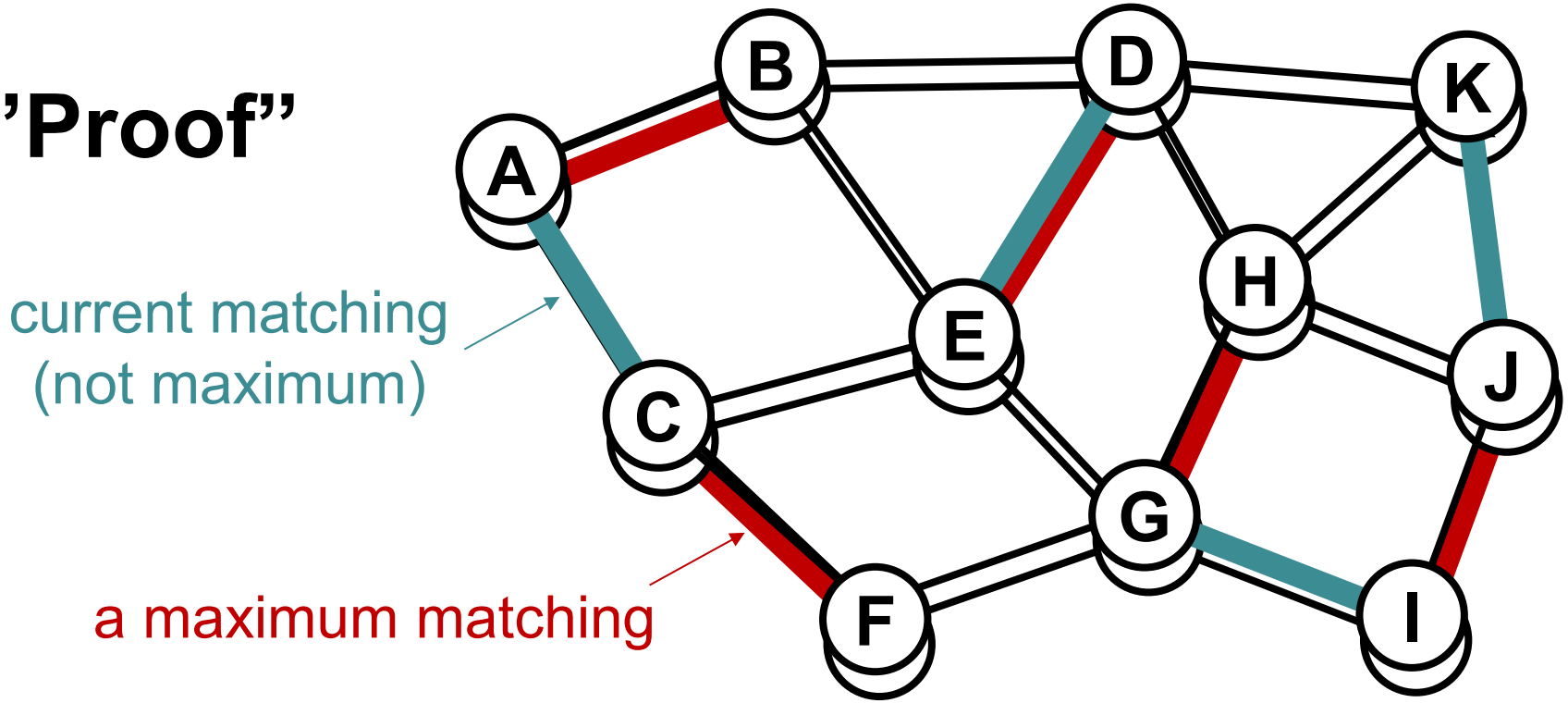
size of matching increased by one

Petersen 1891 / Kőnig 1931 / Berge 1957

Theorem

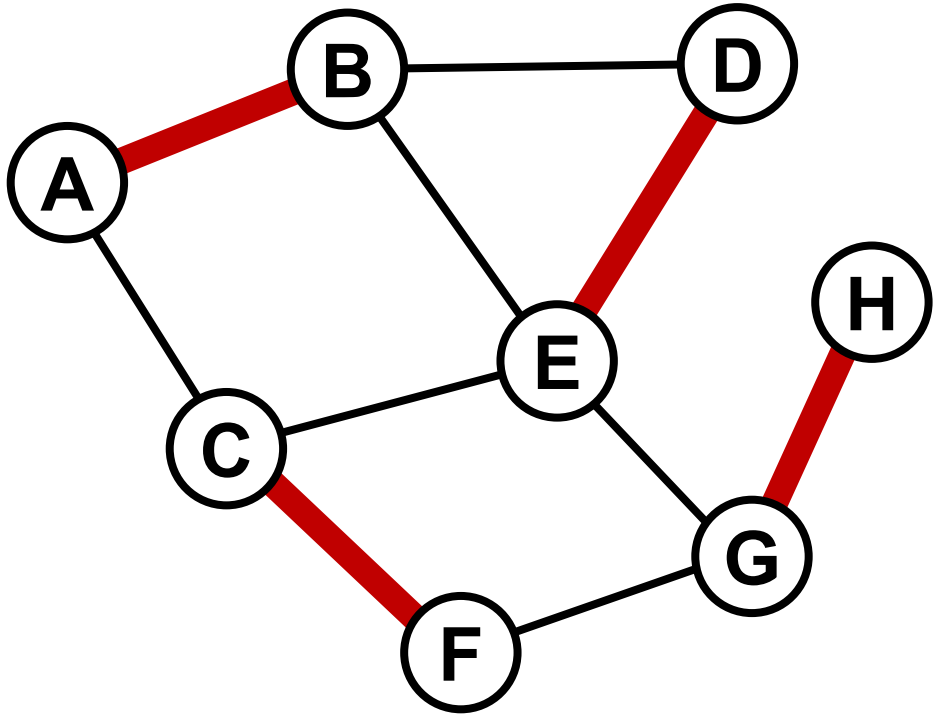
A matching is maximum ⇔ no augmenting path exists

”Proof”



Claude Berge, *Two theorems in graph theory*, Proc. National Academy of Sciences of the United States of America, 1957, doi: 10.1073/pnas.43.9.842

Construction of a maximum matching

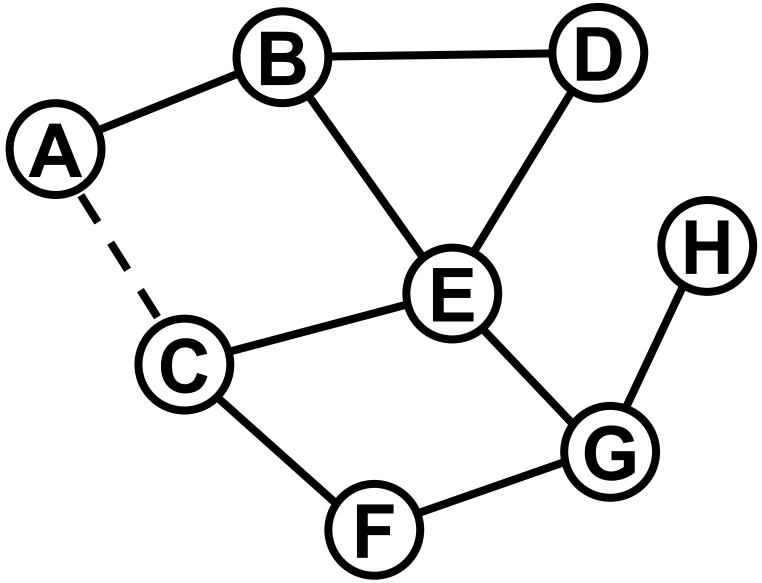


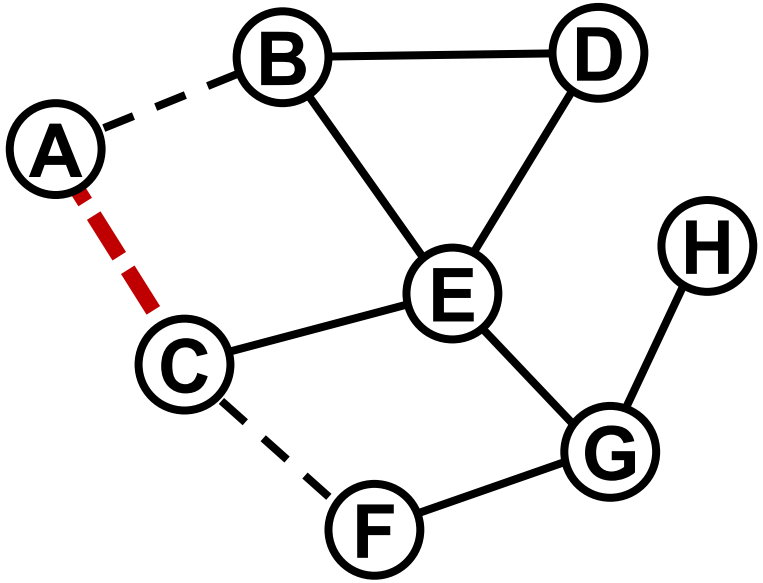
Greedy algorithm

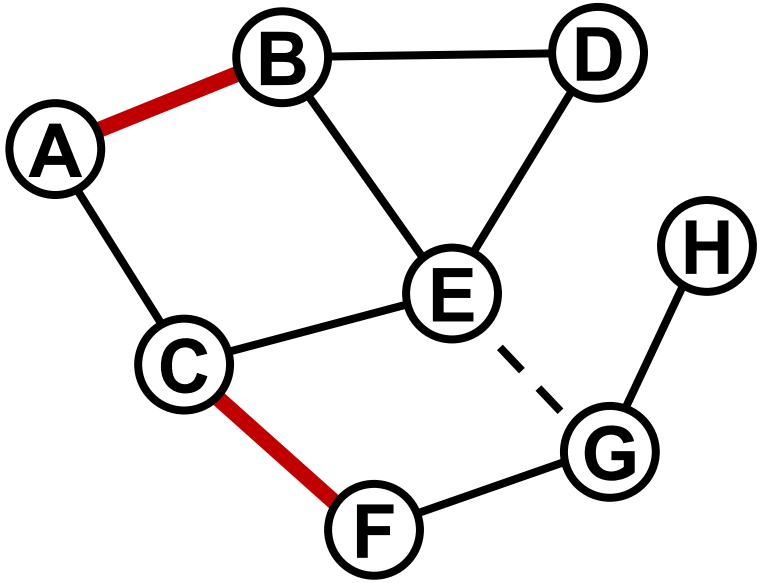
Apply augmenting paths while possible

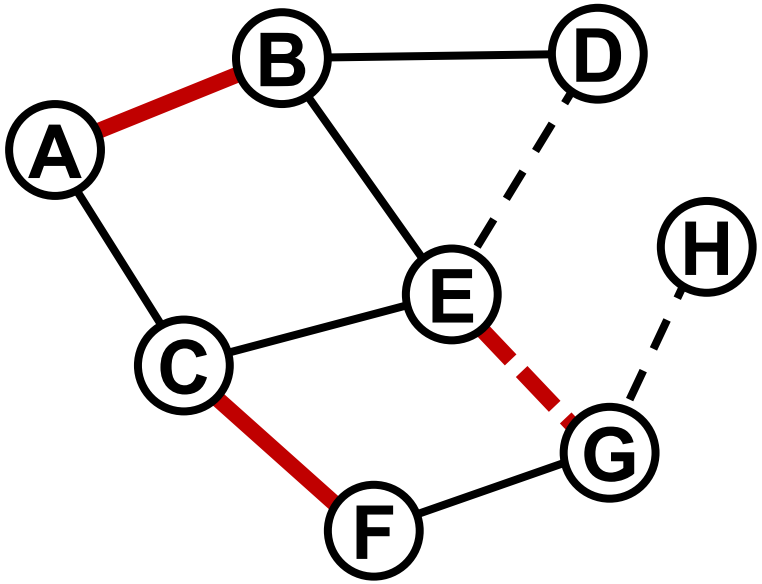
Edmonds 1961 $O(m \cdot n^2)$
Micali, Vazirani 1980 $O(m \cdot n^{1/2})$

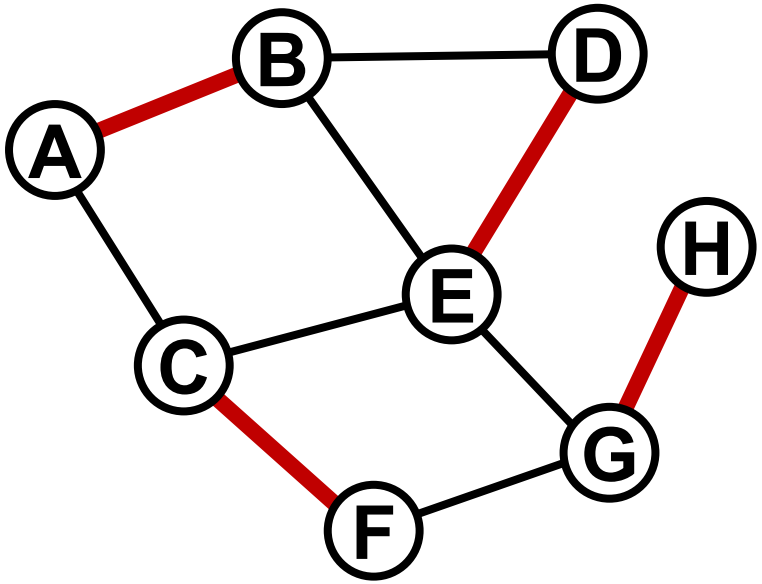
Jack Edmonds, *Paths, trees, and flowers*, Canadian Journal of Mathematics, 1965, doi: 10.4153/CJM-1965-045-4
Silvio Micali, Vijay V. Vazirani, *An $O(\sqrt{|V|}\cdot|E|)$ Algorithm for Finding Maximum Matching in General Graphs,* FOCS, 1980, doi: 10.1109/SFCS.1980.12


A
B
C
D
E
F
G
H

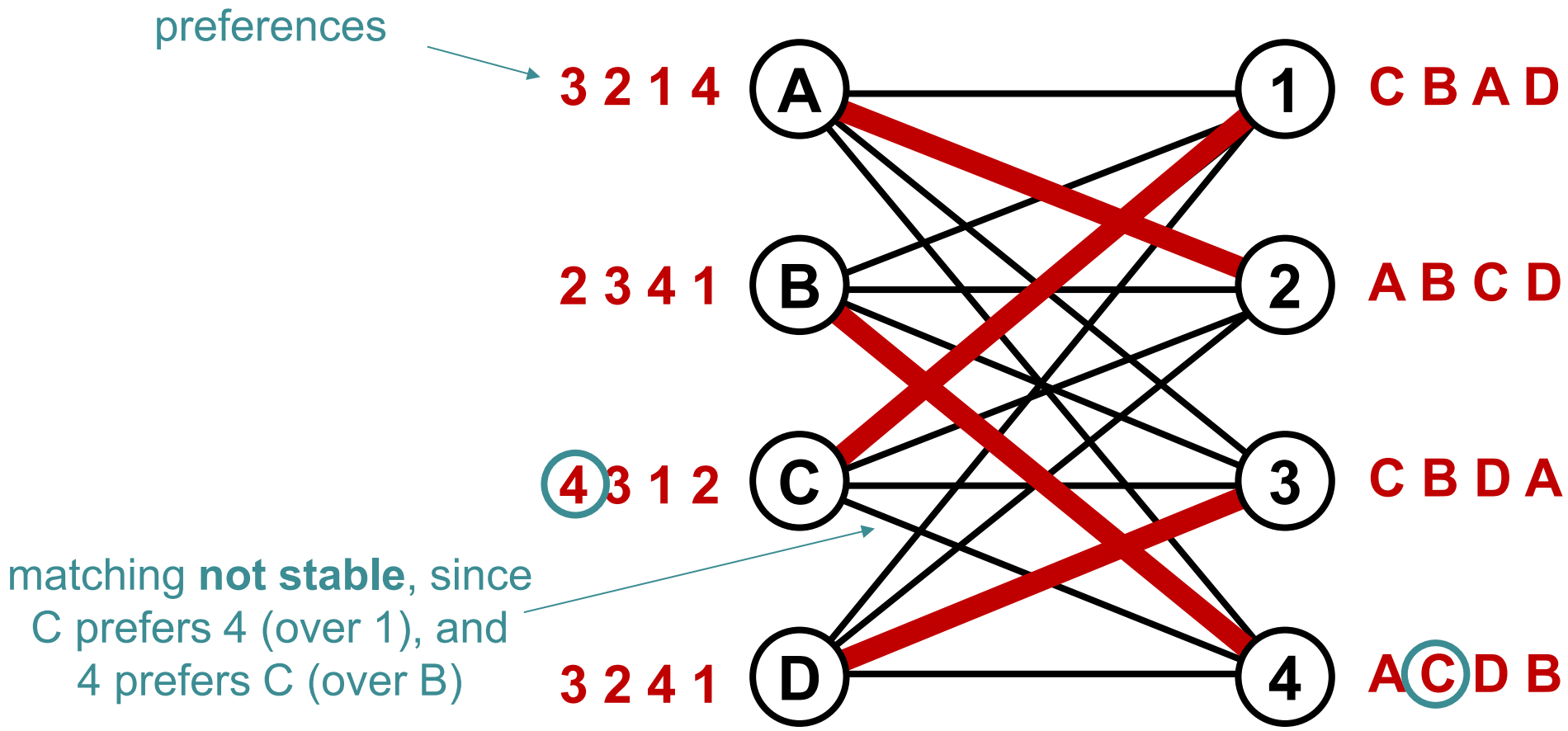

A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H

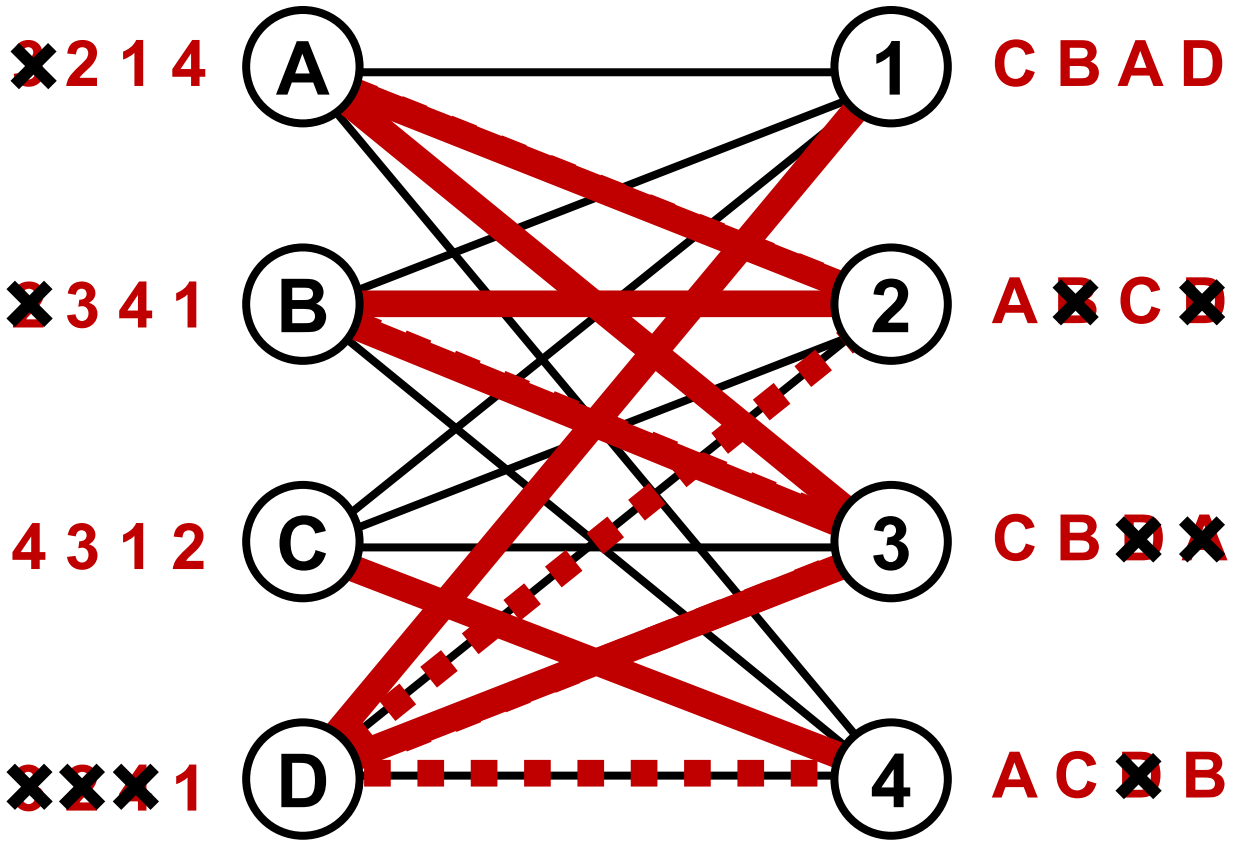
Stable marriage problem



complete bipartite graph with equal number of nodes on each side

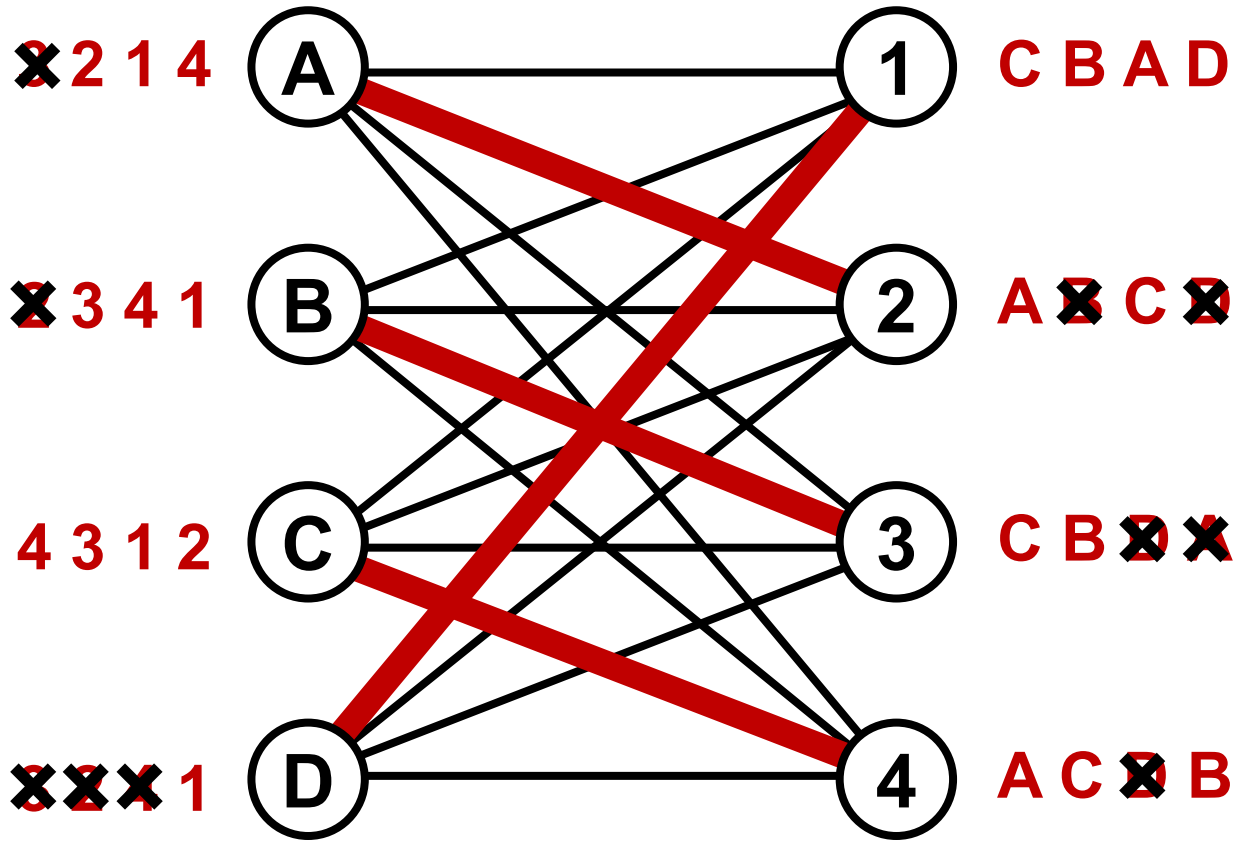
CLRS Chapter 25.2

Stable marriage – greedy algorithm



If left *X* rejected by right *i*, then *i* is matched with some *Y* preferred over *X*
X is matched with the most preferred, not yet rejected *X*

Stable marriage – greedy algorithm

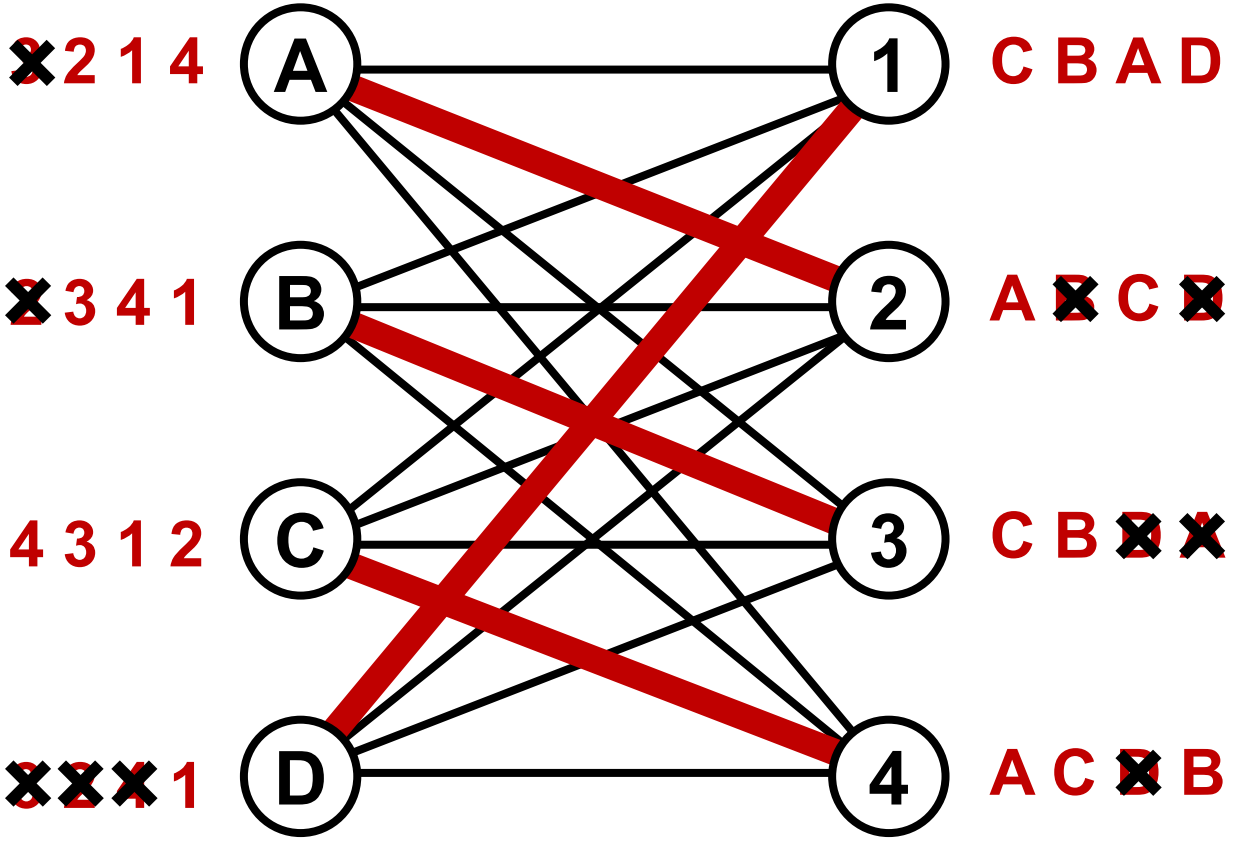


If left *X* rejected by right *i*, then *i* is matched with some *Y* preferred over *X*
X is matched with the most preferred, not yet rejected *X*

Time $O(n^2)$

Theorem (Gale, Shapley 1962)

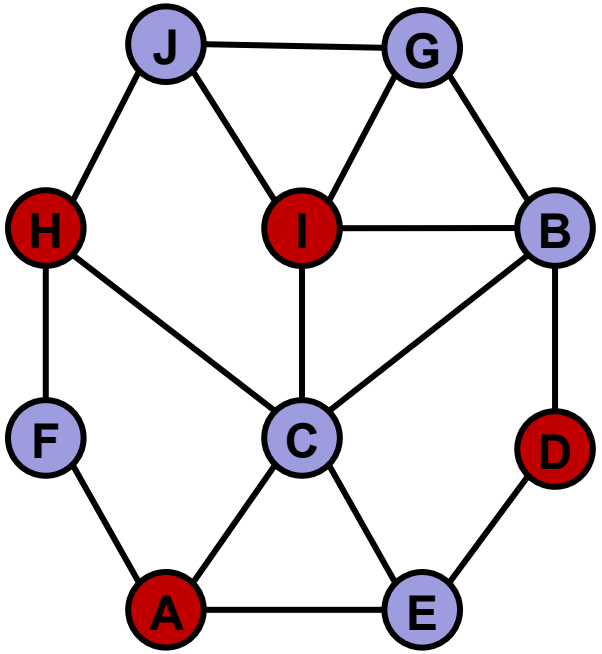
A stable marriage always exist



(Shapley got the Nobel price in economics for the result in 2012)

D. Gale, L. S. Shapley, *College admissions and the stability of marriage*, The American Mathematical Monthly, 1962, doi: 10.2307/2312726

Independent set vs vertex cover



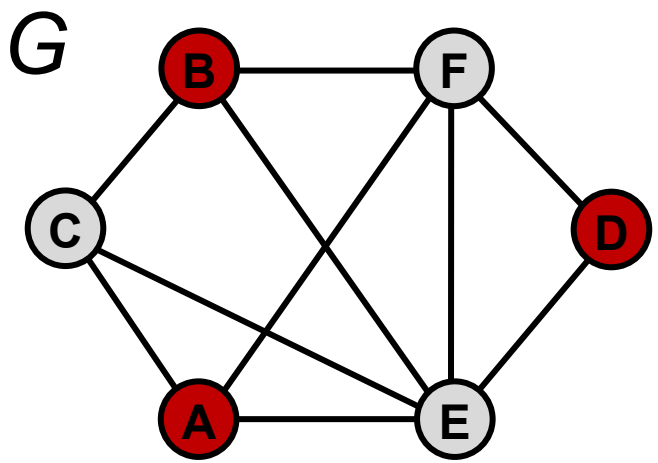
Lemma

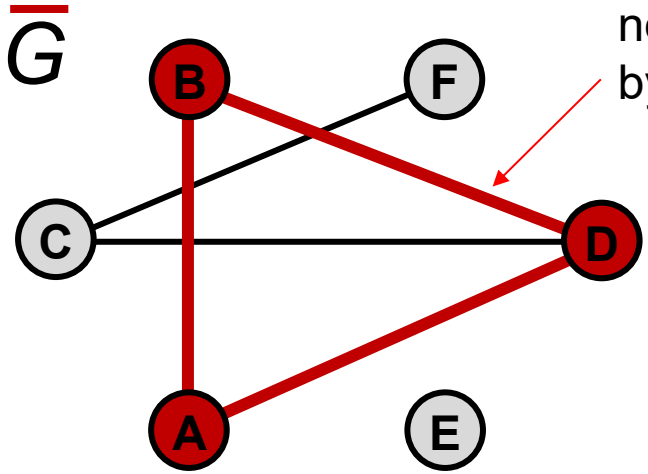
$I \subseteq V$ independent set $\Leftrightarrow$ $V \setminus I$ covers all edges

Theorem

$I \subseteq V$ maximum independent set $\Leftrightarrow$ $V \setminus I$ minimum vertex cover

Independent set vs clique



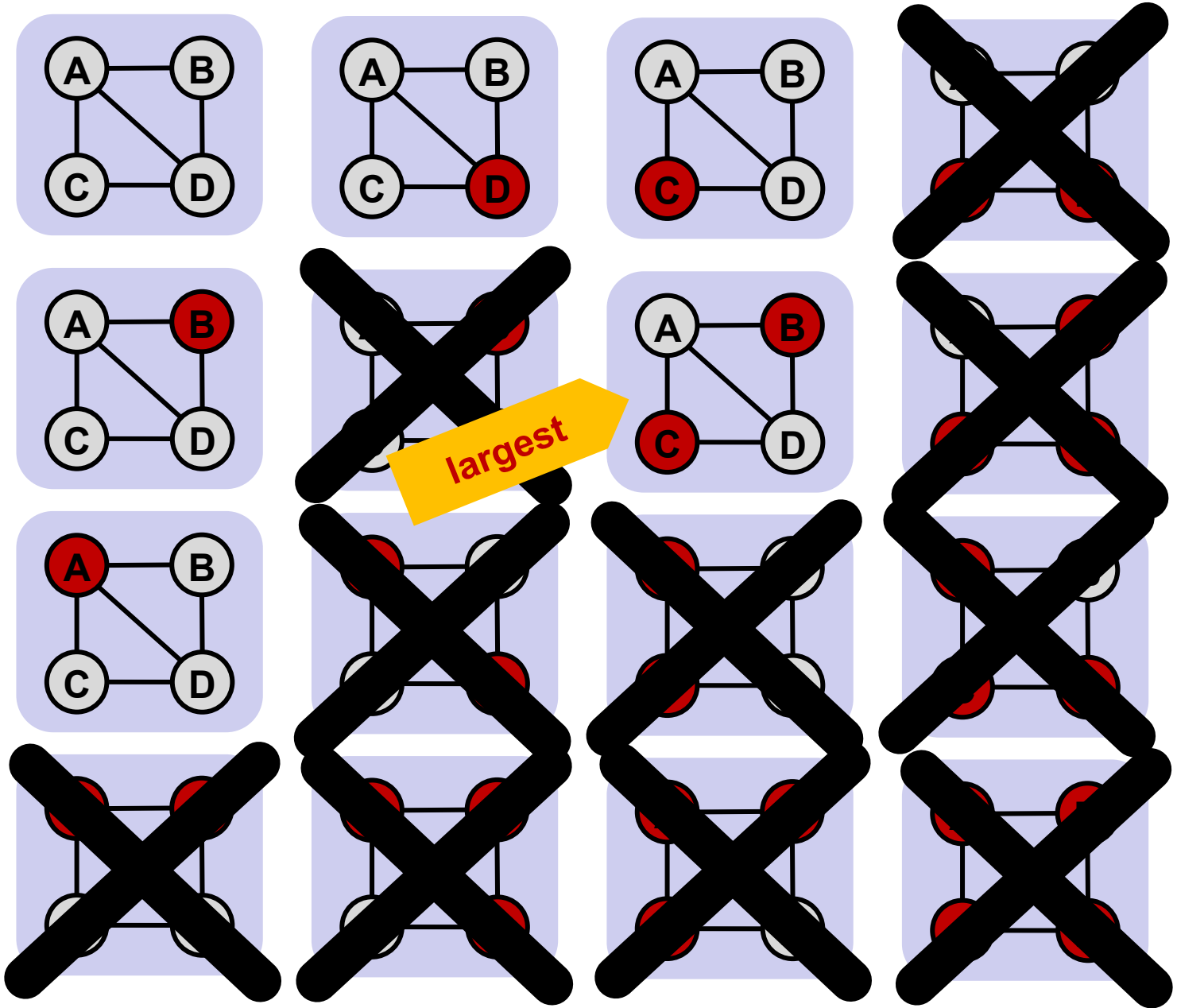


clique = all pairs of nodes connected by an edge

Theorem

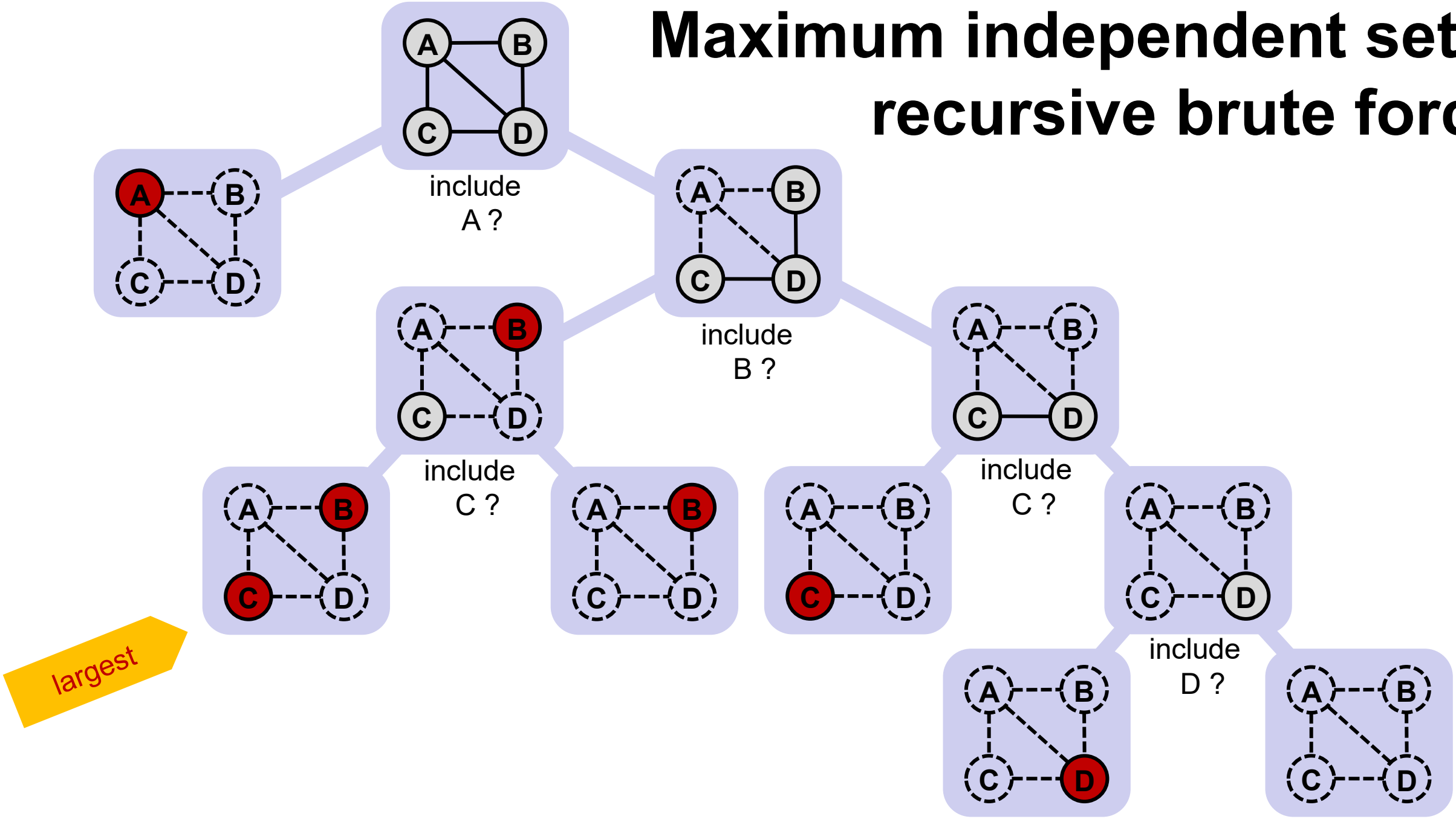
$I \subseteq V$ maximum independent set in $G \Leftrightarrow I$ maximum clique in $\overline{G}$

Maximum independent set – try all 2^n subsets



Time $O(m \cdot 2^n)$

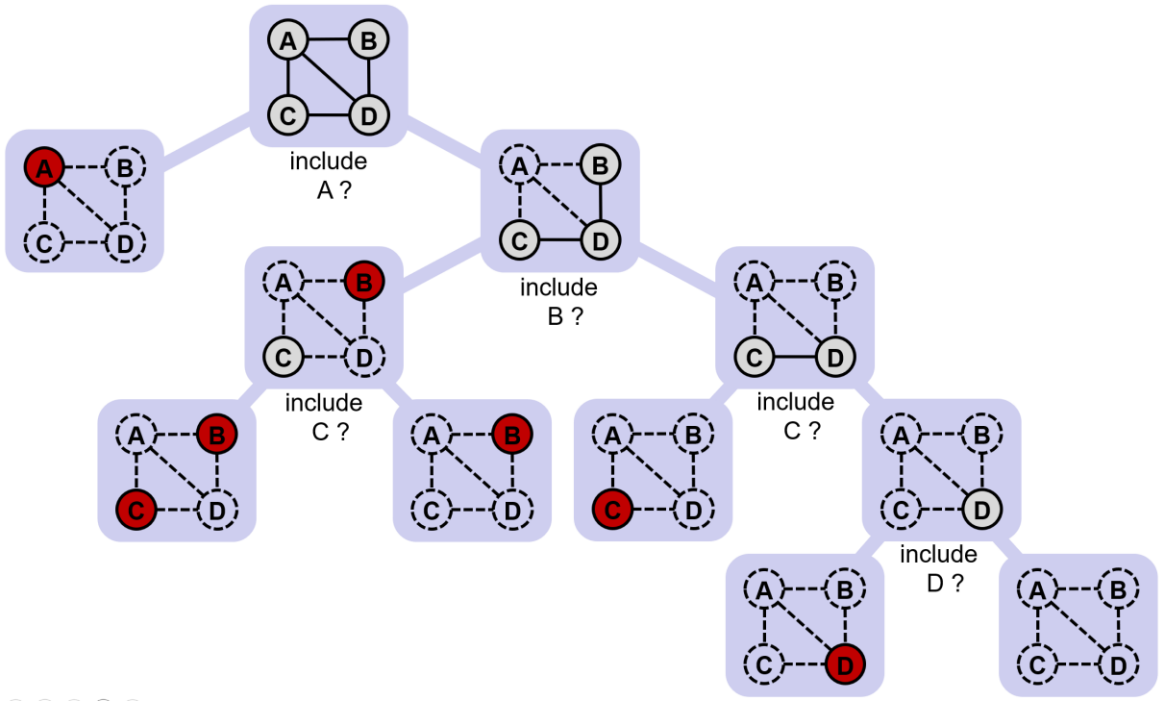
Maximum independent set – recursive brute force



Maximum independent set

Algorithm $\text{MIS1}(G)$

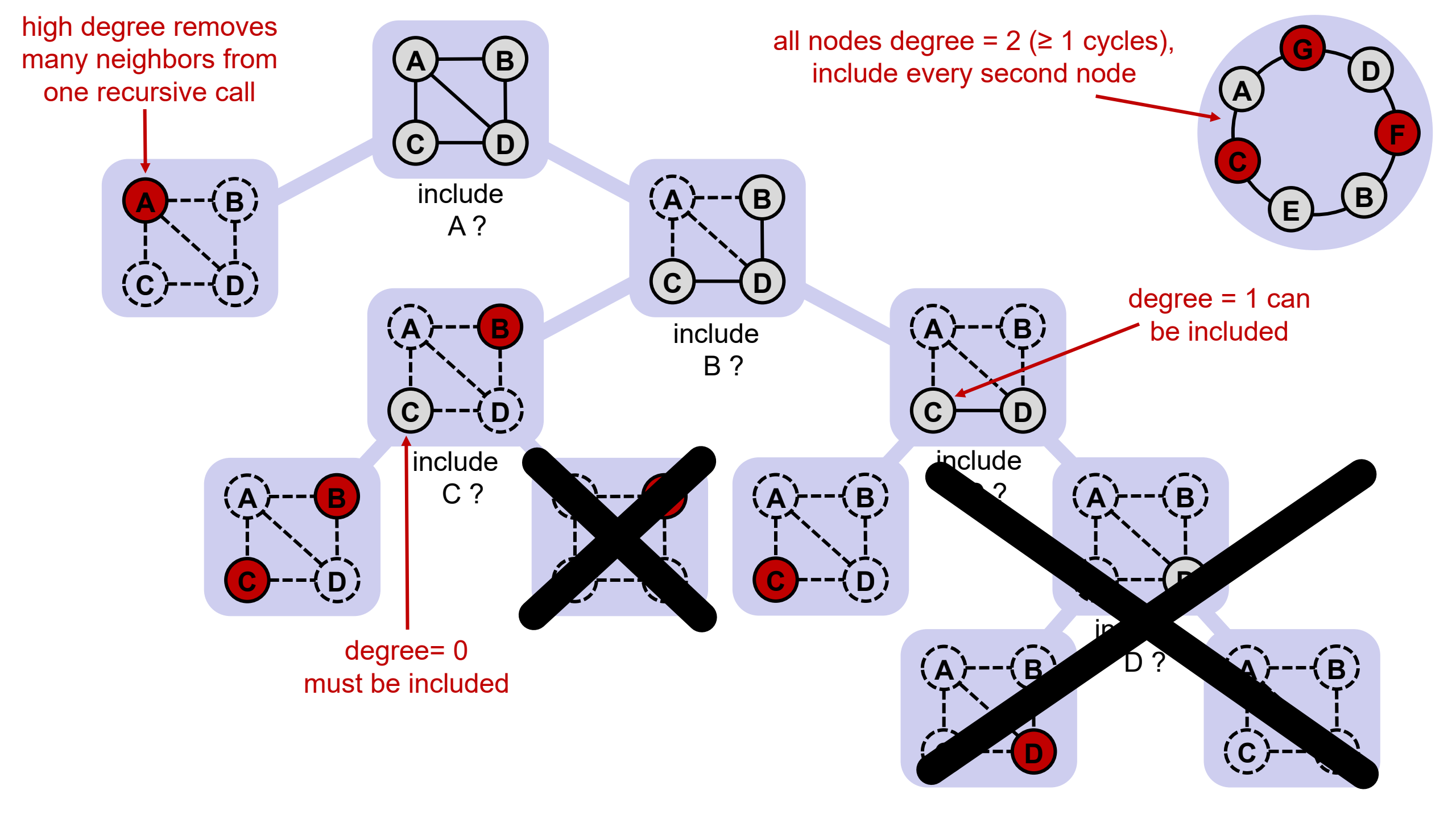
```
1  if V_G = ∅ then
2      return ∅
3  let v ∈ V_G be an arbitrary vertex
4  I = MIS1(G \ {v})
5  I' = {v} ∪ MIS1(G \ {v} \ adj_G(v))
6  if |I'| > |I| then
7      I = I'
8  return I
```



Analysis

recursive calls

$$T(n) \le \begin{cases} 1 & \text{if } n = 0 \\ 1+2\cdot T(n-1) & \text{otherwise} \end{cases} \quad \Rightarrow \quad T(n) \le 2^{n+1}-1$$


high degree removes many neighbors from one recursive call
A
B
C
D
include A ?
include B ?
include C ?
degree= 0 must be included
degree = 1 can be included
include D ?
all nodes degree = 2 (≥ 1 cycles), include every second node
G
D
A
F
C
E
B

Algorithm MIS2 (G)

```
1   I = ∅
2   while ∃ v ∈ V_G : |adj_G(v)| ≤ 1 do
3       I = I ∪ {v}
4       G = G \ {v} \ adj_G(v)
5   # all vertices have degree ≥ 2
6   if ∃ v ∈ V_G : |adj_G(v)| ≥ 3 then
7       I_1 = I ∪ MIS2 (G \ {v})
8       I_2 = I ∪ {v} ∪ MIS2 (G \ {v} \ adj_G(v))
9       return I_1 if |I_1| ≥ |I_2| otherwise I_2
10  # all remaining vertices have degree 2
11  while ∃ cycle C = (c_1, c_2, ..., c_k) in G do
12      I = I ∪ {c_1, c_3, c_5, ..., c_{2⌊k/2⌋}}
13      G = G \ C
14  return I
```

greedy include nodes with degree 0 and 1

like MIS1, except ≥ 4 nodes are removed in second call

greedy include nodes on cycles

Analysis

recursive calls

$$T(n) \le \begin{cases} 1 & \text{if } n = 0 \\ 1+T(n-1)+T(n-4) & \text{otherwise} \end{cases} \quad \Rightarrow \quad T(n) = O(1.38028^n)$$

Reduction – 3-SAT to independent set

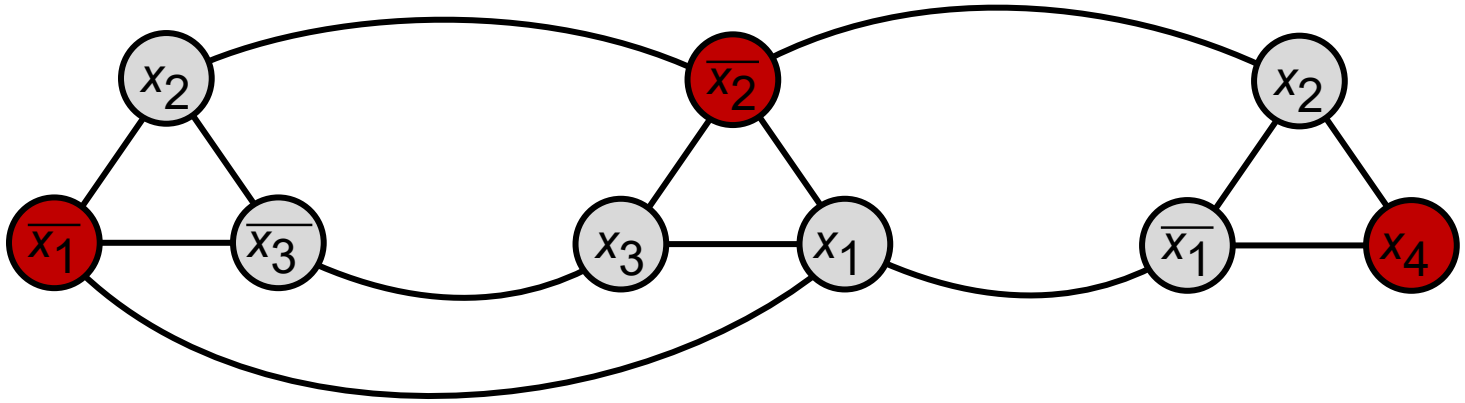
- **3-SAT** Given a Boolean formula in conjunctive normal form (CNF), where all *n* clauses have three literals, e.g.

$$\varphi = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

- Exists assignment of true/false to the literals $x_1, x_2, \ldots$ such that the equation is true?

$$x_1 = x_2 = \text{false}, \qquad x_3 = x_4 = \text{true}$$

- **Reduction** Create one triangle per clause, one node per literal. Add edges $(\overline{x_i}, x_i)$.



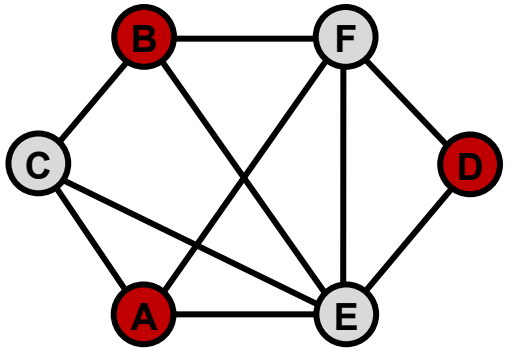
- **Theorem** 3-SAT formula satisfiable ⇔ maximum independent set of size *n*

Richard M. Karp, *Reducibility Among Combinatorial Problems,* Complexity of Computer Computations, 1972, doi: 10.1007/978-1-4684-2001-2_9

NP-hardness

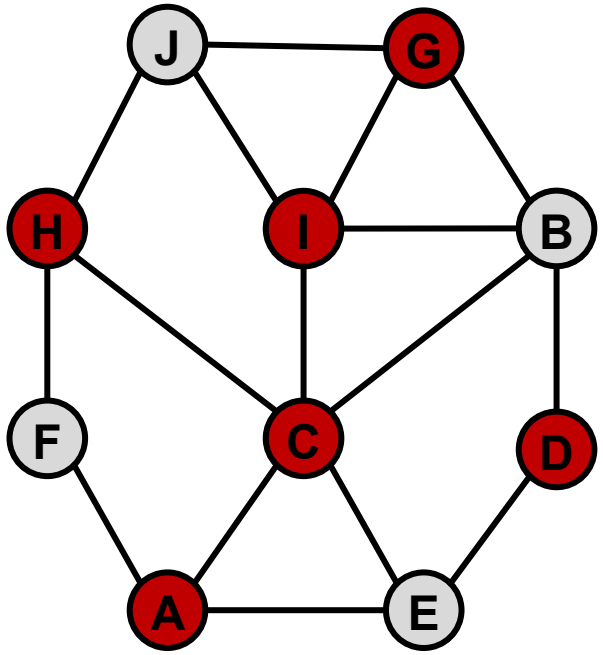
- 3-SAT is an NP-hard problem, implying that it is questionable if there exist better than exponential time algorithms
- The reduction implies that the following problems are also NP-hard
  - maximum independent set
  - maximum clique (= independent set in the negated graph)
  - minimum vertex cover (= negation of maximum independent set)

- Maximum independent set can be solved in time $O(1.1996^n)$ [Xiao & Nagamochi, 2017]



Richard M. Karp, *Reducibility Among Combinatorial Problems*, Complexity of Computer Computations, 1972, doi: 10.1007/978-1-4684-2001-2_9
Mingyu Xiao, Hiroshi Nagamochi, *Exact algorithms for maximum independent set*, Information and Computation, 2017, doi: 10.1016/j.ic.2017.06.001

2-approximation of minimum vertex cover



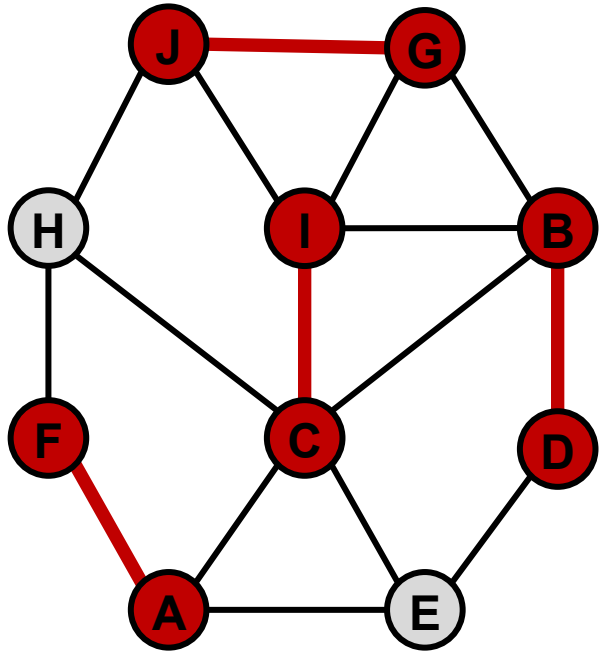
Minimum
vertex cover $C_{optimal}$

Algorithm

- Compute a **maximal** matching M
- Vertex cover C = all nodes in M

Analysis

- C is at most twice the minimum vertex cover $C_{optimal}$, since all edges in M must include a node from $C_{optimal}$
- Time $O(n + m)$



maximal matching M
$\Rightarrow$ vertex cover C

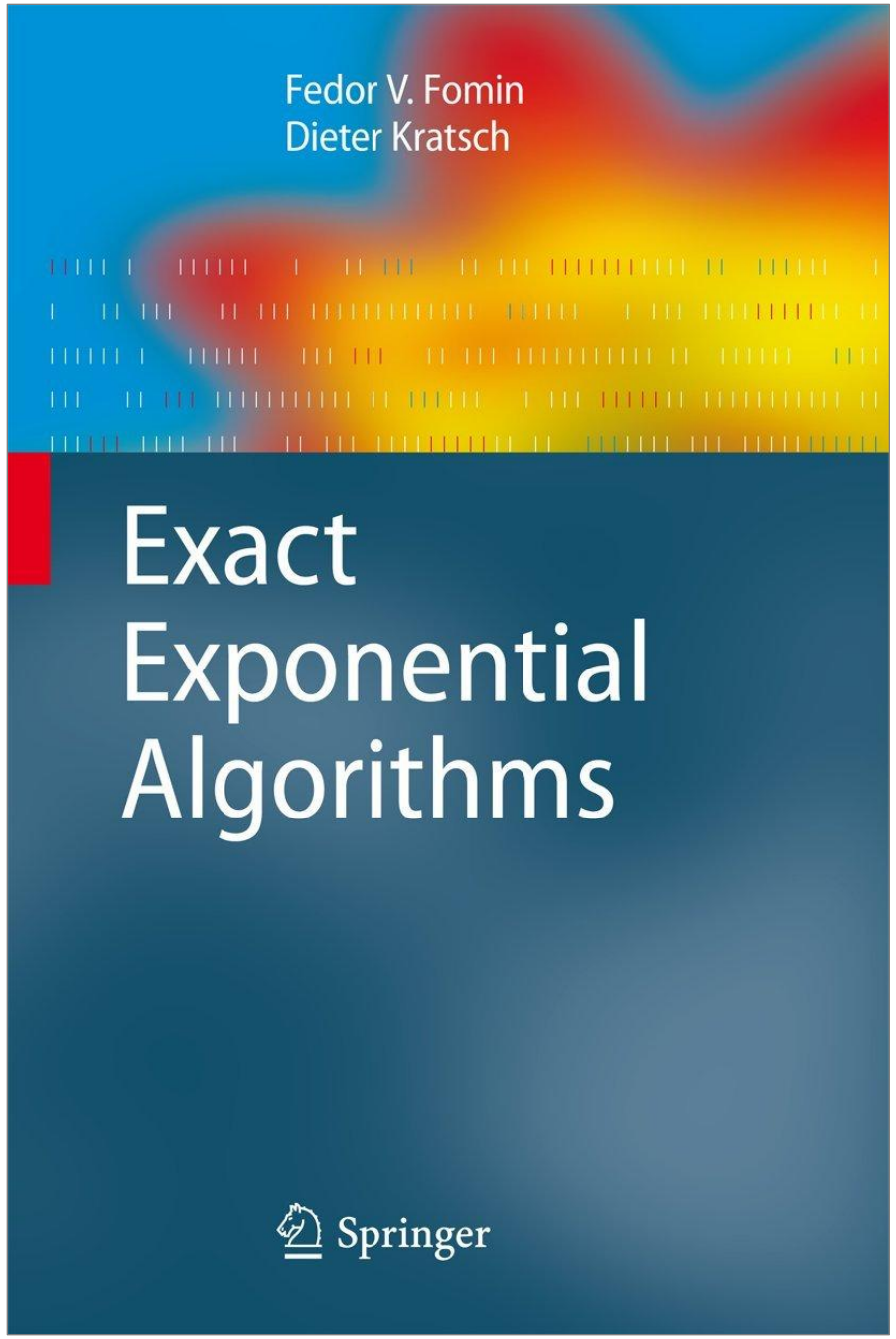
Minimum vertex cover cannot be approximated in polynomial time better than…

- factor $\sqrt{2} \approx 1.4142$, provided **P ≠ NP** (the big open problem in computer science)
- factor 2, provided the ”unique games conjecture” is true (i.e., the above is best possible)

Subhash Khot, *On the power of unique 2-prover 1-round games*, STOC 2002, doi:10.1145/509907.510017

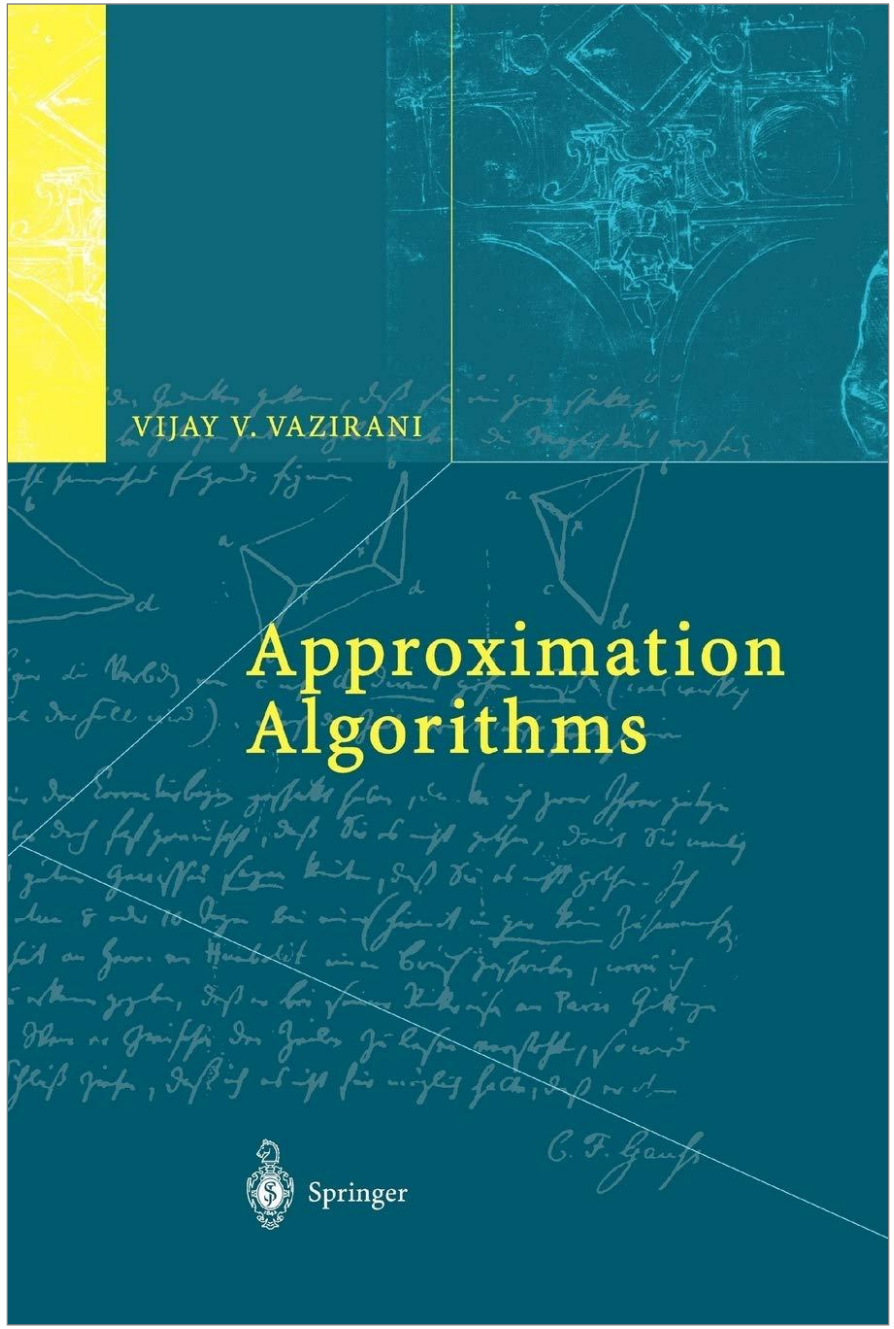
Summary

- Maximal and maximum matchings
- Stable marriage problem
- Maximum clique
- Maximum independent sets
- Minimum vertex cover
- 3-SAT (NP-hardness)
- Reductions
- Exponential time algorithms
- Approximation algorithms



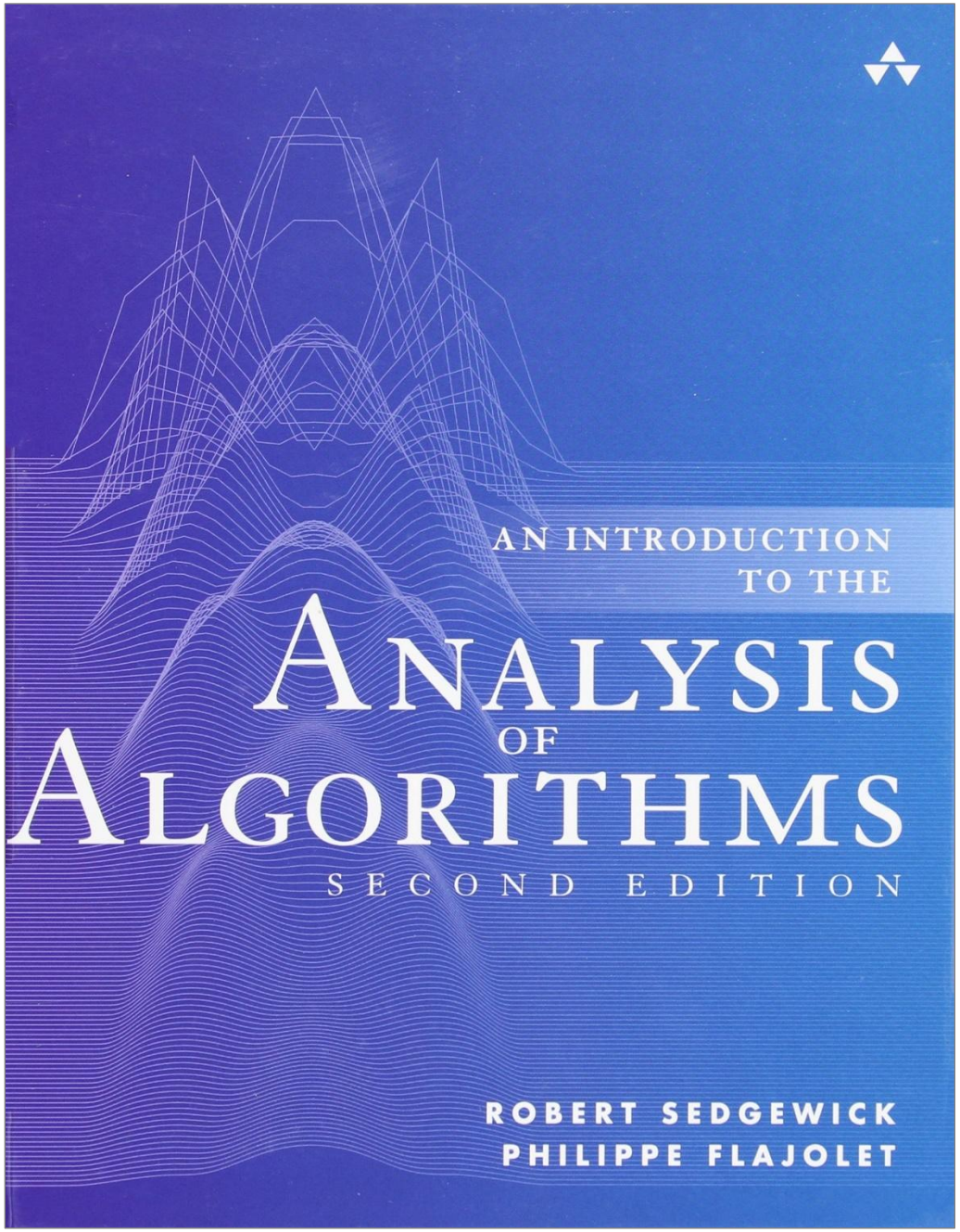
Explains common algorithmic techniques for developing exact (exponential time) algorithms for NP-hard problems

Exact Exponential Algorithms
Fedor V. Fomin, Dieter Kratsch
Texts in Theoretical Computer Science. An EATCS Series
203 sider, Springer-Verlag Berlin Heidelberg 2010
doi: 10.1007/978-3-642-16533-7



Explains the main theoretical approaches to find approximate solutions to hard combinatorial optimization and enumeration problems

Approximation Algorithms
Vijay V. Vazirani
XIX - 380, Springer-Verlag Berlin Heidelberg 2003
doi: 10.1007/978-3-662-04565-7



Detail mathematical background on how to solve recurrences and more

An Introduction to the Analysis of Algorithms
Robert Sedgewick, Philippe Flajolet
XVIII – 572, Addison-Wesley Professional 2013
ISBN: 78-0-321-90575-8

Algorithms and Data Structures

Exam

Exam

- Ordinary exam in January, reexam in May
- Multiple-choice on paper
- No aids allowed at the exam (also not textbooks or personal notes)
- 2 hours
- Preparation for the exam: Solve old exam exercises + Q&A

Max 1 mark per subquestion

Anonymous
Participant index

Algorithms and Data Structures (520251E001)

Monday, January 5, 2026, 10:00–12:00

Participant index: ☐☐☐☐

1.1 A B
1.2 A B
1.3 A B
1.4 A B
1.5 A B
1.6 A B
1.7 A B
1.8 A B
1.9 A B
1.10 A B
1.11 A B
1.12 A B

2.1 A B C D E F G H
2.2 A B C D E F G H
2.3 A B C D E F G H
2.4 A B C D E F G H

3.1 A B C D E F G H I J K
3.2 A B C D E F G H I J K
3.3 A B C D E F G H I J K
3.4 A B C D E F G H I J K
3.5 A B C D E F G H I J K

4 A B C D E

5 A B C D E

6 A B C D

7 A B C D E F

8.1 A B C D E F G H I J K
8.2 A B C D E F G H I J K
8.3 A B C D E F G H I J K
8.4 A B C D E F G H I J K
8.5 A B C D E F G H I J K

9.1 A B C D E F G H I J K
9.2 A B C D E F G H I J K
9.3 A B C D E F G H I J K
9.4 A B C D E F G H I J K
9.5 A B C D E F G H I J K

10.1 A B
10.2 A B
10.3 A B
10.4 A B
10.5 A B

11 A B C D

12 A B C D

13.1 A B C D E F G
13.2 A B C D E F G
13.3 A B C D E F G
13.4 A B C D E F G
13.5 A B C D E F G

14 A B C D

15.1 A B
15.2 A B
15.3 A B
15.4 A B
15.5 A B

16.1 A B C D
16.2 A B C D
16.3 A B C D
16.4 A B C D
16.5 A B C D

17 A B C D

18 A B C D E

19.1 A B
19.2 A B
19.3 A B
19.4 A B
19.5 A B

20.1 A B C D
20.2 A B C D
20.3 A B C D
20.4 A B C D
20.5 A B C D

21.1 A B
21.2 A B
21.3 A B
21.4 A B
21.5 A B

22.1 A B
22.2 A B

23.1 A B
23.2 A B
23.3 A B

24.1 A B
24.2 A B
24.3 A B

EXAM

Algorithms and Data Structures (520251E001)

Monday, January 5, 2026, 10:00–12:00

Institut for Datalogi, Naturvidenskabelige Fakultet, Aarhus Universitet

Number of pages (including front page): **14**

Aids: **None**

Only the separate answer sheet must be filled out and submitted.
This problem statement should *not* be submitted.

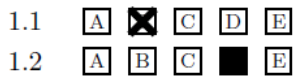
Information and Scoring

This exam set consists of a number of multiple-choice problems. The answers to the problems are indicated on the **separate answer sheet** which is to be submitted. For each problem, the share of the total exam set is indicated. Each subproblem has exactly one correct answer. For each subproblem, you may choose **at most one answer** by crossing out the corresponding box. A subproblem is graded as follows:

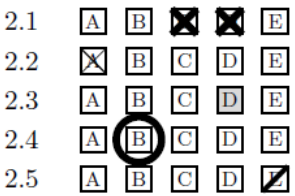
- If you tick the correct answer, you get 1 point.
- If you do not tick any answers, you get 0 points.
- If you tick an incorrect answer, you get $-\frac{1}{k-1}$ points, where k is the number of answer options.

For a problem with weight $v\,\%$ and with n subproblems, where you achieve a total of s points, your score for the problem is calculated as $s\,/\,n \cdot v\,\%$. Note that it is possible to get negative points for a problem.

Correct markings



Invalid markings



A marking must be a clear dark cross between the corners of a box, or a completely filled box. If you make a mistake, you can write next to the answer options what your answer.