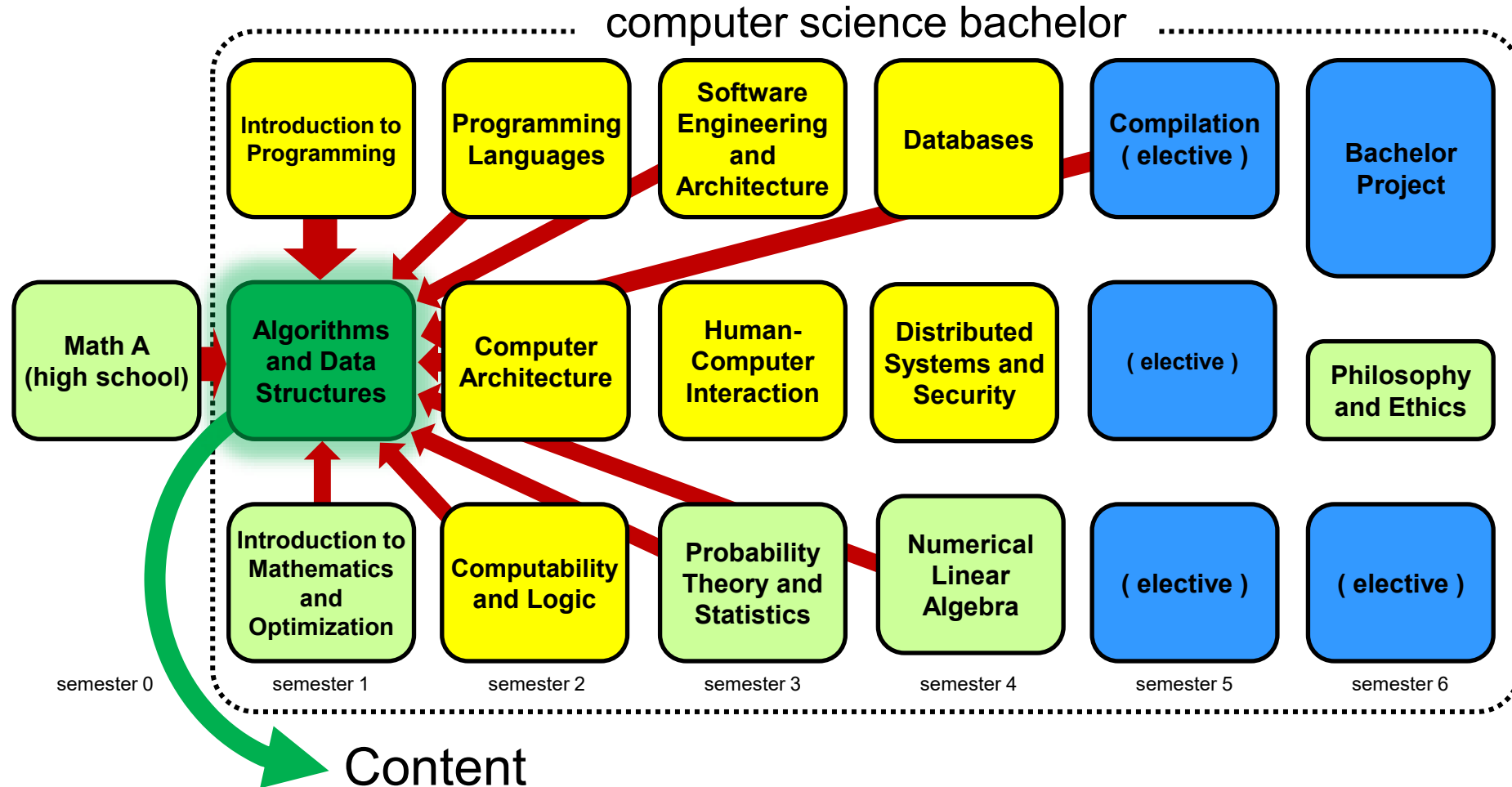


Algorithms and Data Structures

Gerth Stølting Brodal
gerth@cs.au.dk

➔ scientific prerequisites
(ideal world)

Exam : January
Reexam : May



- **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

Previous exams and practice exercises on course webpage

Language

English

Exercise sessions and study café, also Danish

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

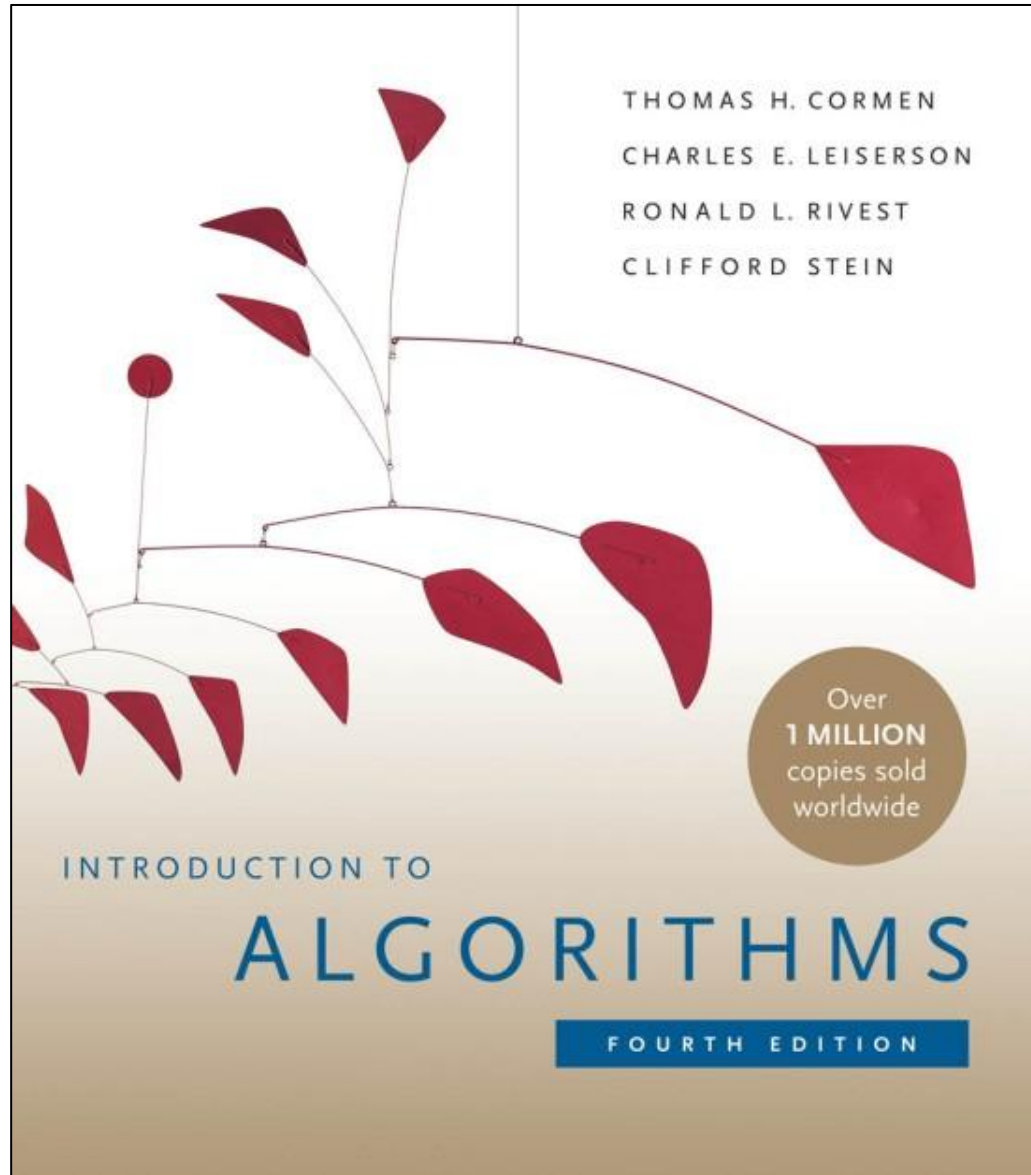
A set of exercises for each TØ

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 misunderstandings 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

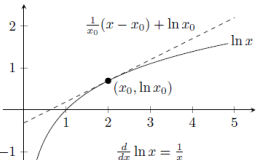
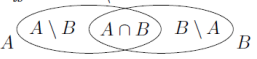
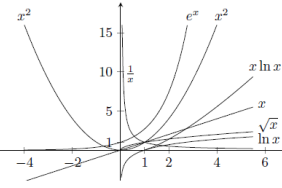
Gerth Stølting Brodal

(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																				
$\mathbb{N} = \{1, 2, 3, \dots\}$ natural numbers $\mathbb{R} =$ real numbers $\mathbb{R}^+ = \{x \in \mathbb{R} \mid x > 0\}$ <i>Floor</i> $\lfloor x \rfloor$, <i>ceiling</i> $\lceil x \rceil$ Associativity $(a \cdot b) \cdot c = a \cdot (b \cdot c)$, $(a + b) + c = a + (b + c)$ Commutativity $a \cdot b = b \cdot a$, $a + b = b + a$ Distributivity $a \cdot (b + c) = a \cdot b + a \cdot c$ Powers $a^0 = 1$, $a^1 = a$, $a^{b+c} = a^b \cdot a^c$ $(a^b)^c = a^{b \cdot c}$, $a^{b^c} = (a^b)^c$, $a^{-b} = \frac{1}{a^b}$ $(a \cdot b)^c = a^c \cdot b^c$, $a^{b \cdot c} = (a^b)^c$ Roots $\sqrt[n]{a} = \sqrt[n]{a} = a^{1/n}$, $\sqrt[n]{a} = a^{1/n}$ $\sqrt[n]{a \cdot b} = \sqrt[n]{a} \cdot \sqrt[n]{b}$, $\sqrt[n]{a/b} = \sqrt[n]{a}/\sqrt[n]{b}$ Fractions $\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$ $\frac{a/b}{c/d} = \frac{a \cdot d}{b \cdot c}$ $\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + b \cdot c}{b \cdot d}$ Logarithms $\log_b a = c \Leftrightarrow b^c = a$, $b^{\log_b a} = a$ <i>Natural logarithm</i> $\ln a = \log_e a$, $e = 2.71828 \dots$ <i>Binary logarithm</i> $\log_2 a = \lg a$ $\log_b 1 = 0$, $\log_b b = 1$ $\log_b (a \cdot c) = \log_b a + \log_b c$ $\log_b (a/c) = \log_b a - \log_b c$ $\log_b (a^c) = c \cdot \log_b a$ $\log_b a = \frac{\log_c a}{\log_c b}$, $\log_b^c a = (\log_b a)^c$ $a^{\log_b c} = 2^{\log_2 a \cdot \log_2 c \cdot \frac{1}{\log_2 b}} = c^{\log_b a}$  Matrices ^(1,4) $A = \begin{pmatrix} a_{11} & \dots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{pmatrix} \quad m \times n \text{ matrix}$ <i>Matrix addition</i> ($m \times n$ og $m \times n$) $C = A + B$, $c_{ij} = a_{ij} + b_{ij}$ <i>Matrix multiplication</i> ($m \times n$ og $n \times p$) $C = AB$, $c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$ <i>Constant multiplication</i> ($m \times n$) $C = cA$, $c_{ij} = c \cdot a_{ij}$ $(A + B) + C = A + (B + C)$ $A + B = B + A$ $c(dA) = (cd)A$ $c(A + B) = (cA) + (cB)$ $(AB)C = A(BC)$ $A(B + C) = (AB) + (AC)$ $(A + B)C = (AC) + (BC)$ 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 <i>et al.</i>, Introduction to Algorithms, 4th Edition, appendix A-C, 2022 Steve Seiden, Theoretical computer science cheat sheet, ACM SIGACT News, 27(4), 1996	Sets $Set \ A = \{a_1, a_2, \dots, a_n\}$ <i>Size</i> A <i>Membership</i> $x \in A$, <i>non-member</i> $x \notin A$ <i>Empty set</i> $\emptyset$, $\emptyset = 0$ <i>Subset</i> $A \subseteq B$, i.e. $x \in A \Rightarrow x \in B$ <i>Set intersection</i> $A \cap B$ <i>Set union</i> $A \cup B$ <i>Set difference</i> $A \setminus B$ or $A - B$  $A \cup B = (A \setminus B) \cup (A \cap B) \cup (B \setminus A)$ <i>Commutativity</i> $A \cap B = B \cap A$, $A \cup B = B \cup A$ <i>Associativity</i> $A \cup (B \cup C) = (A \cup B) \cup C$ $A \cap (B \cap C) = (A \cap B) \cap C$ <i>Distributivity</i> $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ <i>DeMorgan's laws</i> $A \setminus (B \cap C) = (A \setminus B) \cup (A \setminus C)$ $A \setminus (B \cup C) = (A \setminus B) \cap (A \setminus C)$ <i>Idempotence</i> $A \cup A = A = A \cap A$ <i>Empty set</i> $A \cup \emptyset = A$, $A \cap \emptyset = \emptyset$ <i>Complement</i> $\bar{A}$ wrt. <i>universe</i> U $\bar{\bar{A}} = A$ where $A \subseteq U$, $\bar{\bar{A}} = A$ $\overline{A \cap B} = \bar{A} \cup \bar{B}$, $\overline{A \cup B} = \bar{A} \cap \bar{B}$ A and B are <i>disjoint</i> $\Leftrightarrow A \cap B = \emptyset$ <i>Cross product / Cartesian product</i> $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$ Sums ⁽²⁾ $\sum_{i=1}^n a_i = a_1 + a_2 + \dots + a_n$ $\sum_{i=1}^n (a_i + b_i) = \sum_{i=1}^n a_i + \sum_{i=1}^n b_i$ $\sum_{i=1}^n (c \cdot a_i) = c \cdot \sum_{i=1}^n a_i$ $\sum_{i=0}^n 2^i = 1 + 2^1 + \dots + 2^n = 2^{n+1} - 1$ $\sum_{i=0}^n a^i = \frac{a^{n+1} - 1}{a - 1}$ for $a \neq 1$ $\sum_{i=0}^{\infty} a^i = \lim_{n \rightarrow \infty} \sum_{i=0}^n a^i = \frac{1}{1-a}$, $a < 1$ $\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$ $\sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}$ $\sum_{i=1}^n i \cdot a^i = \frac{a}{(1-a)^2} (1 - a^n - n \cdot a^n + n \cdot a^{n+1})$ $\sum_{i=1}^{\infty} i \cdot a^i = \frac{a}{(1-a)^2}$ for $a < 1$ <i>Polynomial</i> $P(x) = \sum_{i=0}^k c_i \cdot x^i$ <i>Telescoping sum</i> $\sum_{i=0}^{n-1} (a_{i+1} - a_i) = a_n - a_0$ <i>n-th harmonic number</i> $H_n = \sum_{i=1}^n \frac{1}{i} = \frac{1}{1} + \frac{1}{2} + \dots + \frac{1}{n}$ $\ln n + \frac{1}{n} \leq H_n \leq \ln n + 1$ $\lim_{n \rightarrow \infty} (H_n - \ln n) = \gamma = 0.577215 \dots$ γ = Euler-Mascheroni constant Products $\prod_{i=1}^n x_i = x_1 \cdot x_2 \cdot \dots \cdot x_n$ <i>Factorial</i> $n! = \prod_{i=1}^n i = 1 \cdot 2 \cdot \dots \cdot n$ $\ln(\prod_{i=1}^n x_i) = \sum_{i=1}^n \ln(x_i)$ $0 \leq \ln(n!) - (n \ln n - n + 1) \leq \ln n$	Probability theory ⁽³⁾ <i>Mean</i> $\bar{x} = \frac{x_1 + x_2 + \dots + x_n}{n}$ <i>Binomial coefficient</i> $\binom{n}{k} = \frac{n!}{(n-k)! \cdot k!}$ (number of ways to select k elements among n, independent on the order) <i>Linearity of expectation</i> $E[\sum_{i=1}^k c_i \cdot X_i] = \sum_{i=1}^k c_i \cdot E[X_i]$ <i>Bernoulli distribution</i> $X \sim \text{Bern}(p)$ $\Pr[X = 1] = p$, $\Pr[X = 0] = 1 - p$ <i>Binomial distribution</i> $X \sim \text{Bin}(n, p)$ (sum of n Bernoulli trials) $\Pr[X = k] = \binom{n}{k} \cdot p^k \cdot (1-p)^{n-k}$ Expected value $\mu = E[X] = p \cdot n$ <i>Geometric distribution</i> $X \sim \text{Geom}(p)$ (number of Bernoulli trials before 1) $\Pr[X = k] = p \cdot (1-p)^{k-1}$ $E[X] = \sum_{k=1}^{\infty} k \cdot \Pr[X = k] = \frac{1}{p}$ Logical expressions <i>Or / disjunction</i> $U \vee V$ <i>And / conjunction</i> $U \wedge V$ <i>Not / negation</i> $\neg U$ <i>Implication / conditional</i> $U \Rightarrow V$ <i>Equivalent / biconditional</i> $U \Leftrightarrow V$ <i>Exists</i> $\exists x : U(x)$ <i>For all</i> $\forall x : U(x)$ <table><tr><td>$\vee$</td><td>F</td><td>T</td></tr><tr><td>$\wedge$</td><td>F</td><td>T</td></tr><tr><td>$\neg$</td><td>F</td><td>T</td></tr><tr><td>$\vee$</td><td>F</td><td>T</td></tr><tr><td>$\wedge$</td><td>F</td><td>T</td></tr><tr><td>$\neg$</td><td>F</td><td>T</td></tr></table> $\Rightarrow$ F T T $\Leftrightarrow$ F T T XOR F T T F T T F T F F F F T T F T T F T T T F F = false, T = true Asymptotic notation ⁽¹⁾ $f(x) = O(g(x)) \Leftrightarrow \exists c, x_0 \forall x \geq x_0 : f(x) \leq c \cdot g(x)$ $f(x) = \Omega(g(x)) \Leftrightarrow \exists c > 0, x_0 \forall x \geq x_0 : f(x) \geq c \cdot g(x)$ $f(x) = \Theta(g(x)) \Leftrightarrow f(x) = O(g(x)) \wedge f(x) = \Omega(g(x))$ 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 	$\vee$	F	T	$\wedge$	F	T	$\neg$	F	T	$\vee$	F	T	$\wedge$	F	T	$\neg$	F	T
$\vee$	F	T																		
$\wedge$	F	T																		
$\neg$	F	T																		
$\vee$	F	T																		
$\wedge$	F	T																		
$\neg$	F	T																		

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

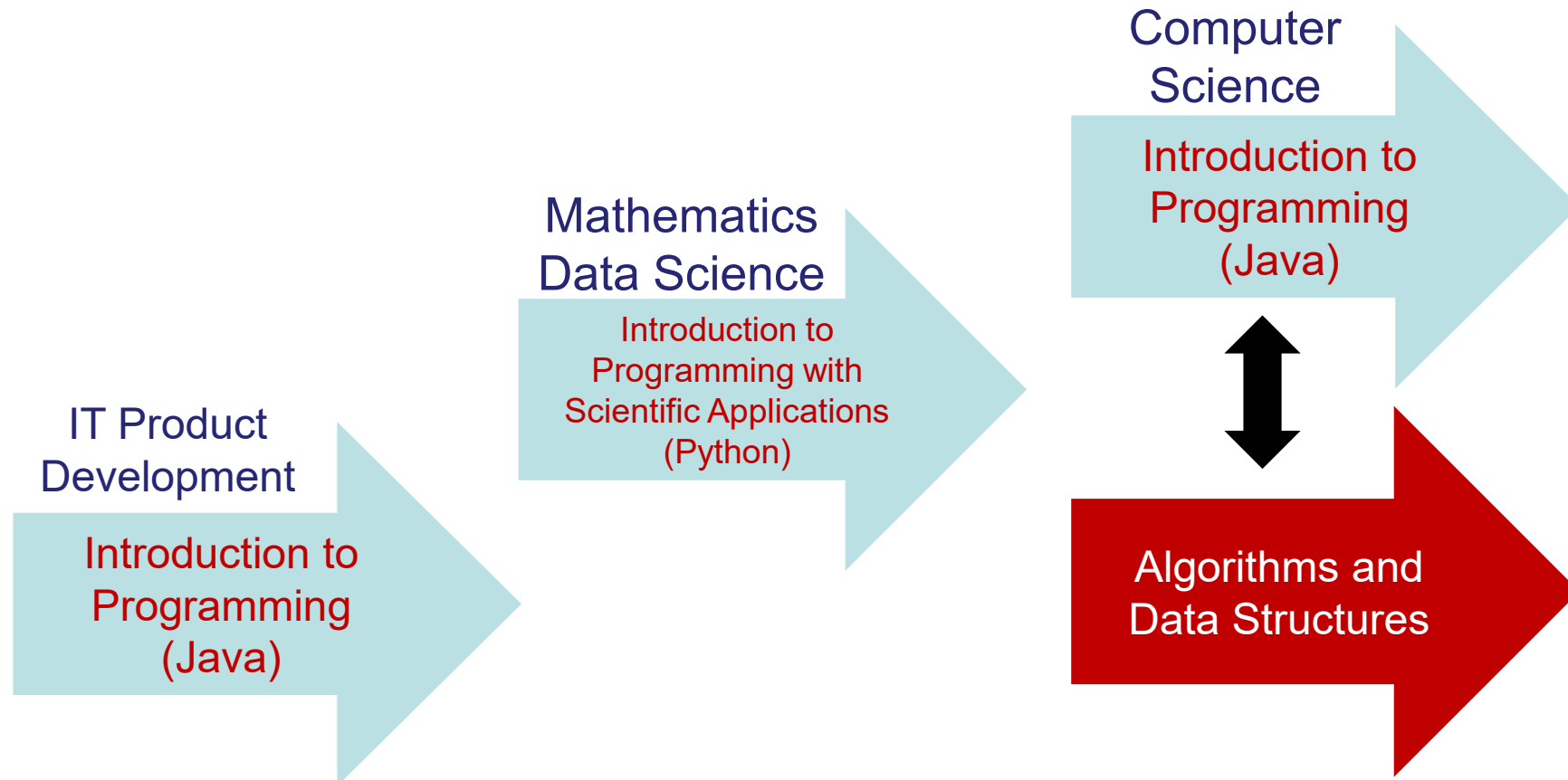
- a) Computer science (1st year)
- b) Data science (2nd year)
- c) IT product development (3rd year)
- d) Mathematics (usually 3rd year)
- e) Physics (usually 3rd year)
- f) other (usually supplement in programming)

Programming experience

(for the programming language you know the best)

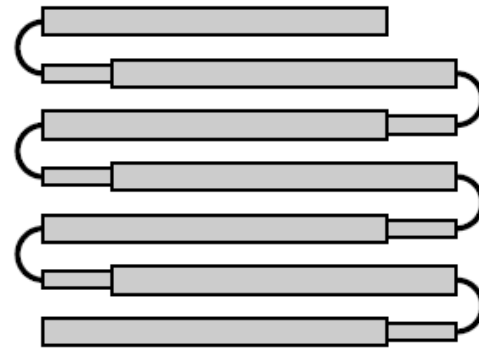
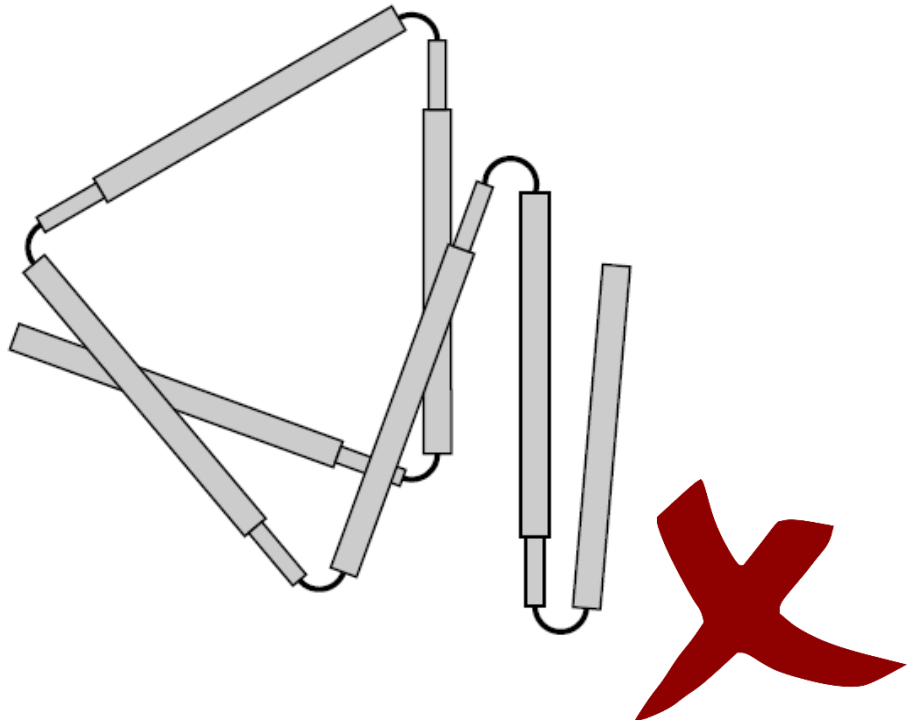
- a) None
- b) Basic knowledge
- c) Foundational knowledge
- d) Advanced
- e) Expert

- 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



”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

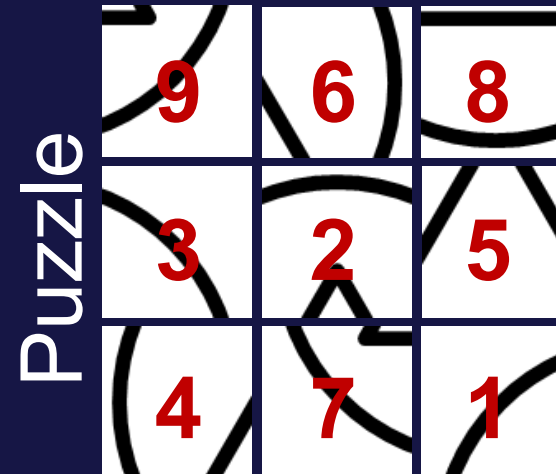
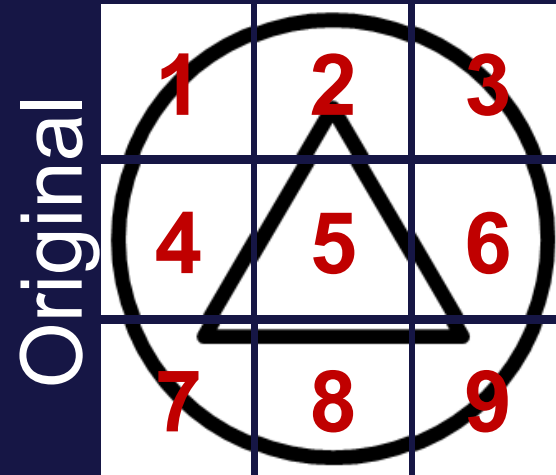
b) 6

c) 7

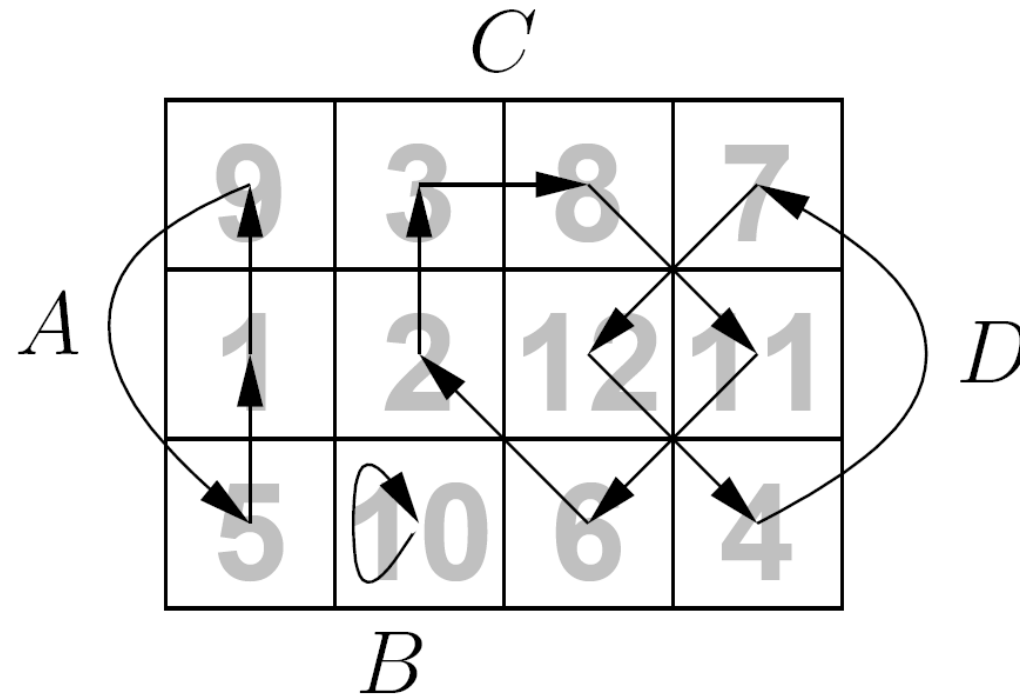
d) 8

e) 9

f) Don't know



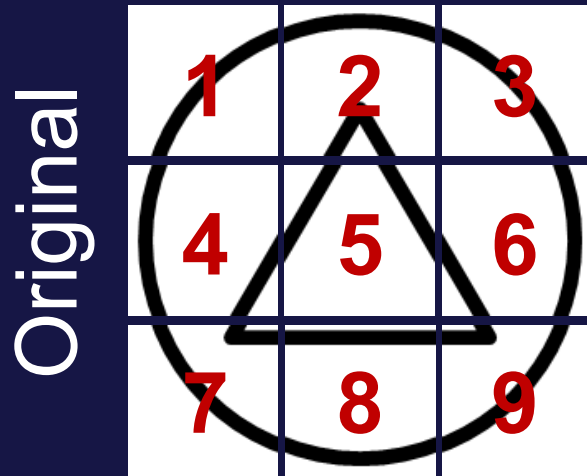
Cycles (Permutations)



Each arrow points to the piece's correct position

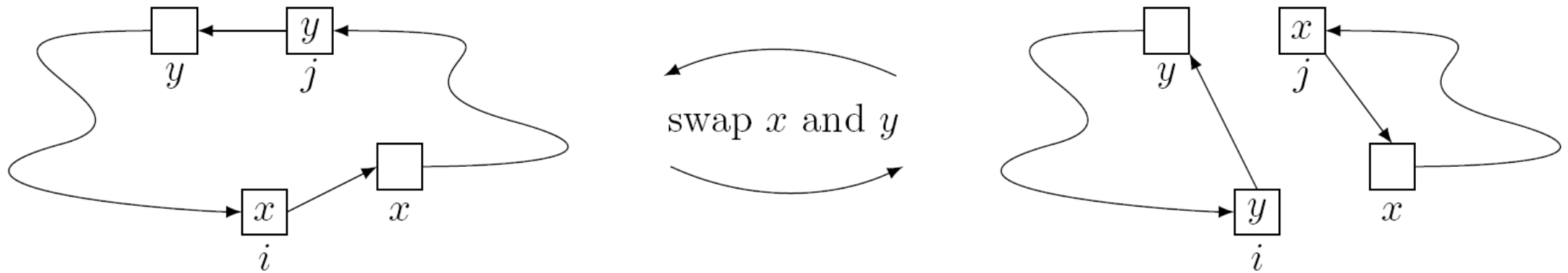
The arrows define a set of cycles (e.g. cycles A,B,C,D)

Optimal number of swaps ?



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

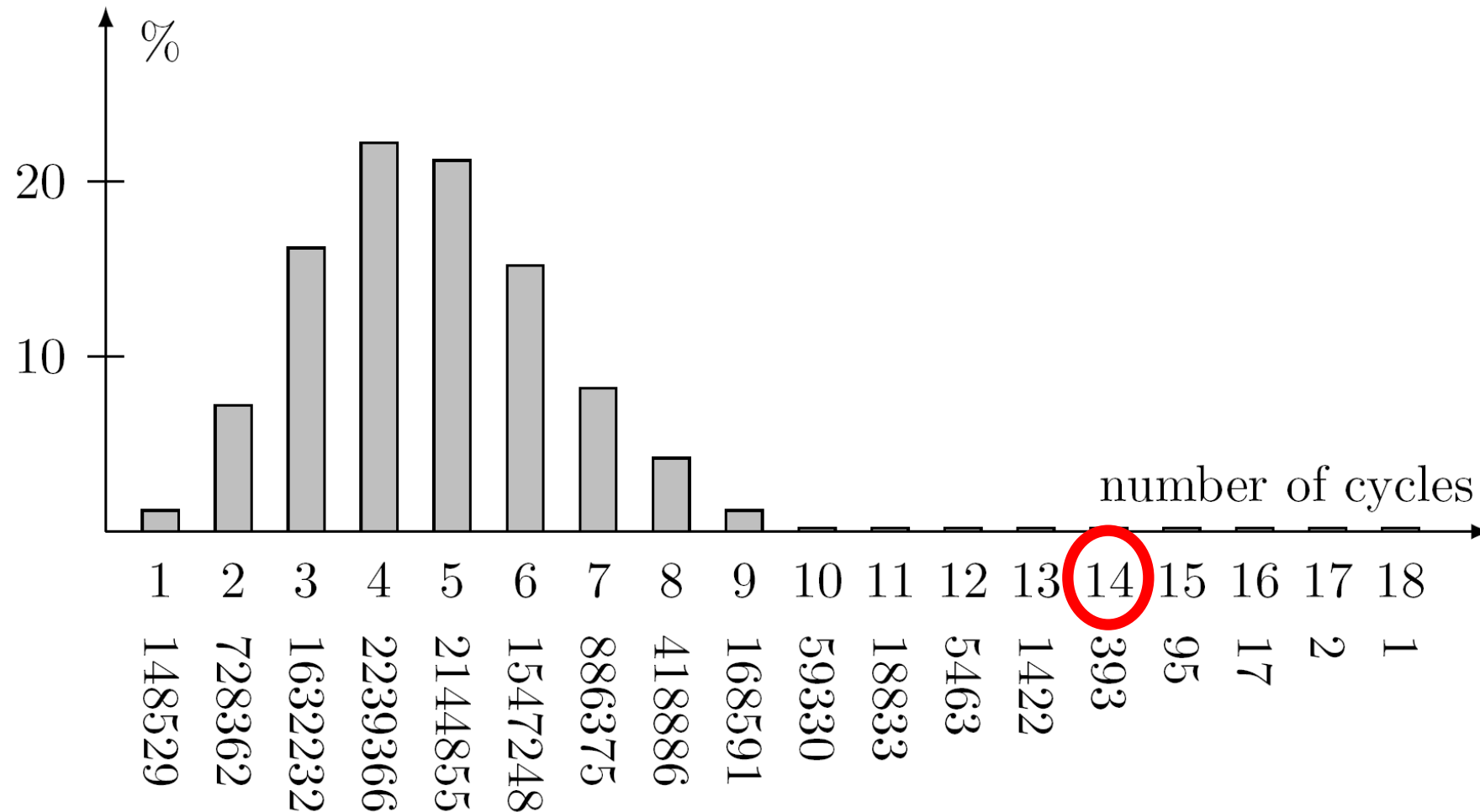
The algorithm is **optimal** since the number of swaps is best possible (the proved lower and upper bounds are identical)

Theorem

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

Distribution of number of cycles

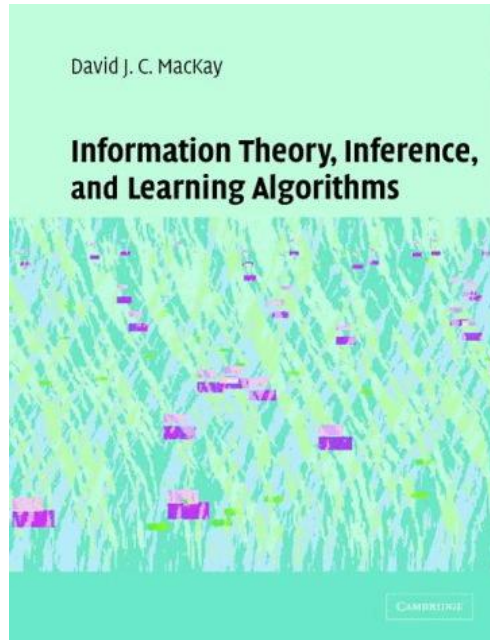
$n = 64$, 10.000.000 permutations



Algorithmic insights...

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

Random permutations...

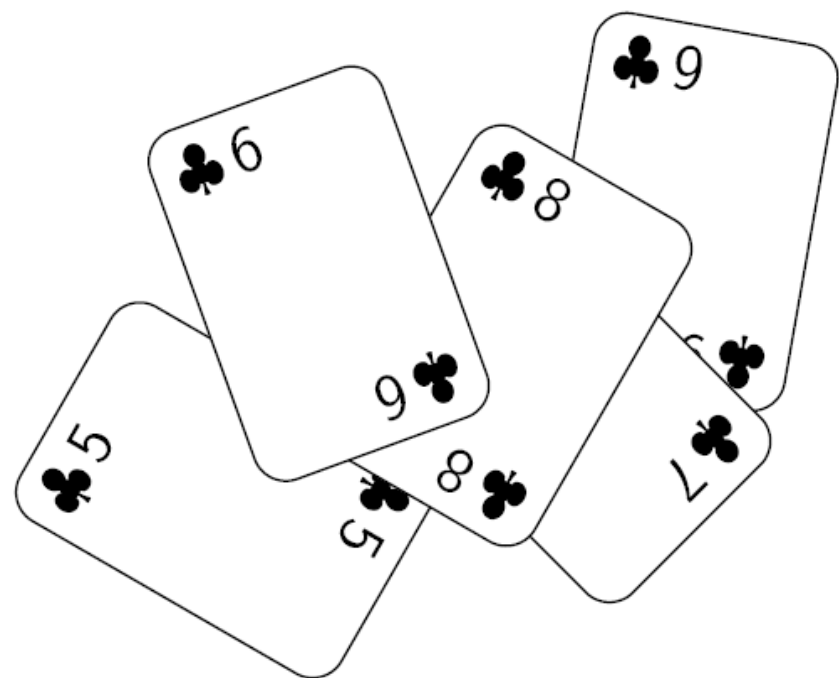


More information on random permutations can be found in David J.C. MacKay, appendix to *Information Theory, Inference, and Learning Algorithms*, on "Random Permutations", 4 pages.

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

SelectionSort

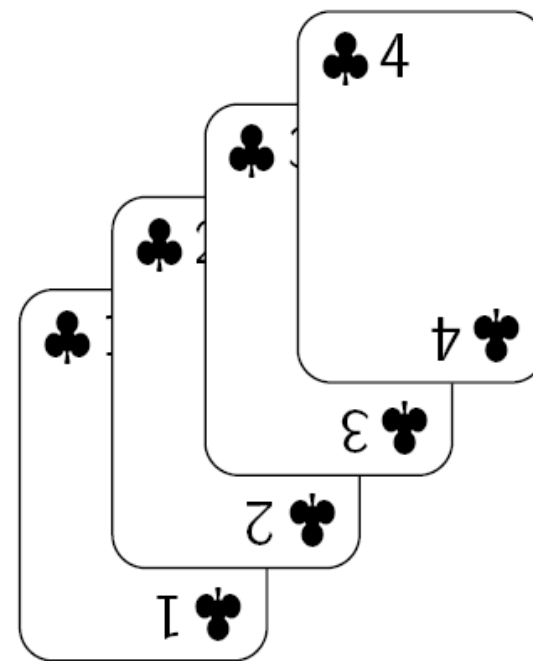
Move smallest card



Unsorted

C

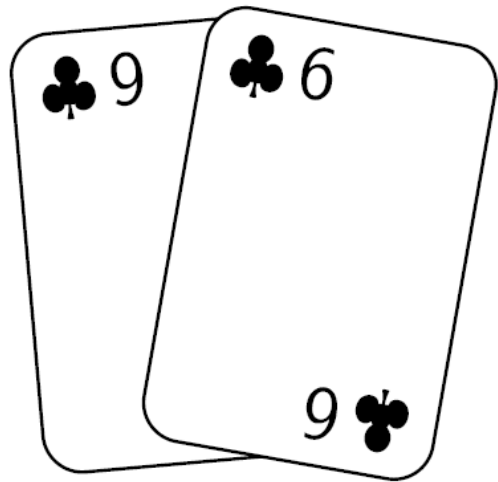
S



Sorted

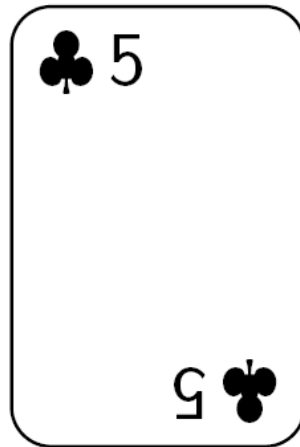
Invariant S sorted and $C \geq S$

Discarded



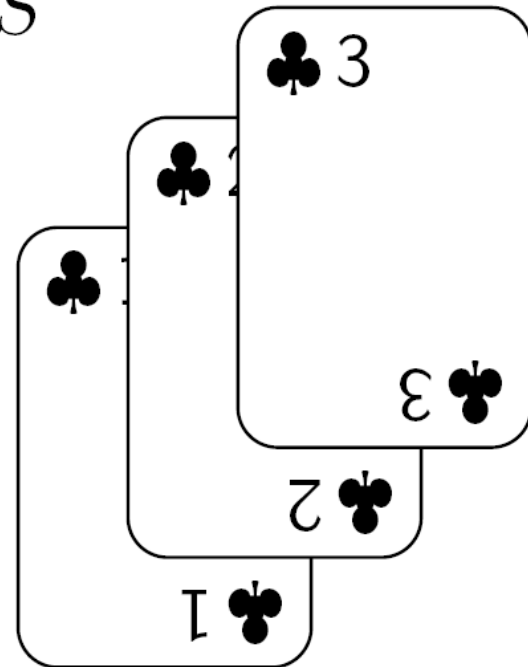
L

Minimum
candidate

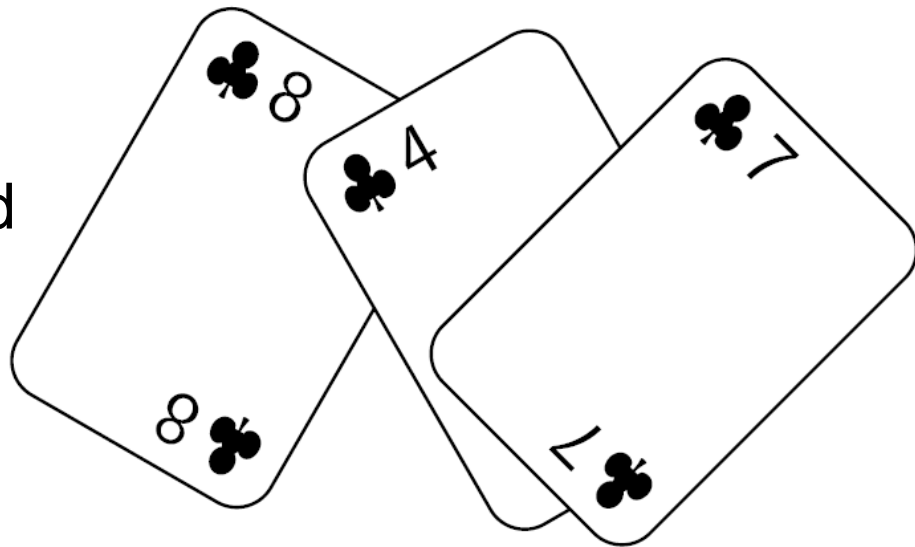


M

S



Not
considered
yet



U

Invariant

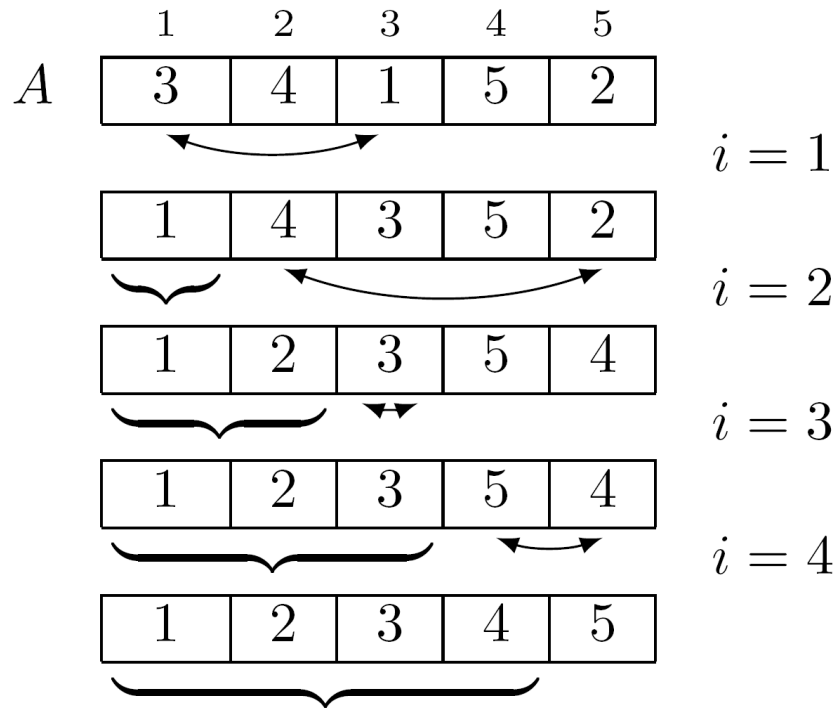
S sorted, $U \cup L \cup M \geq S$, $L \geq M$, $|M| \leq 1$, and $|L| \geq 1 \Rightarrow |M| = 1$

Implicit SelectionSort

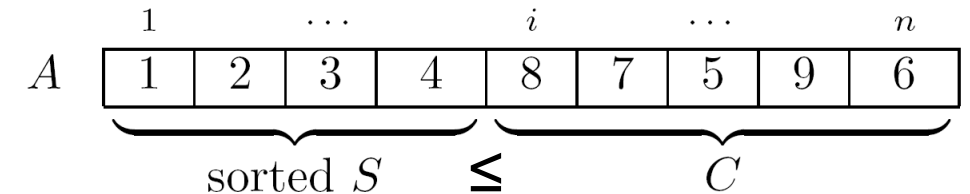
Algorithm SELECTIONSORTABSTRACT(A)

```

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

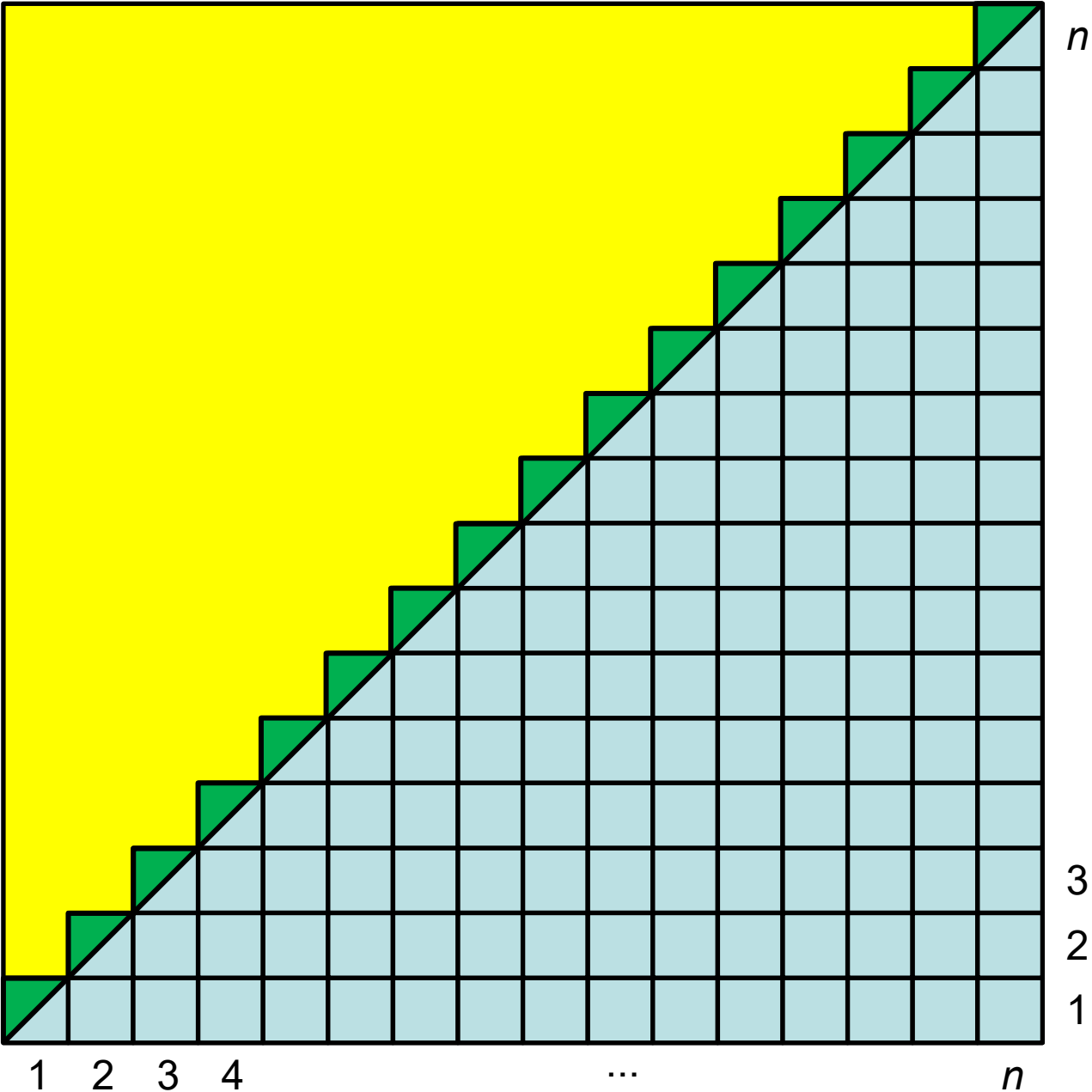
Lemma

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

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



Algorithms and Data Structures

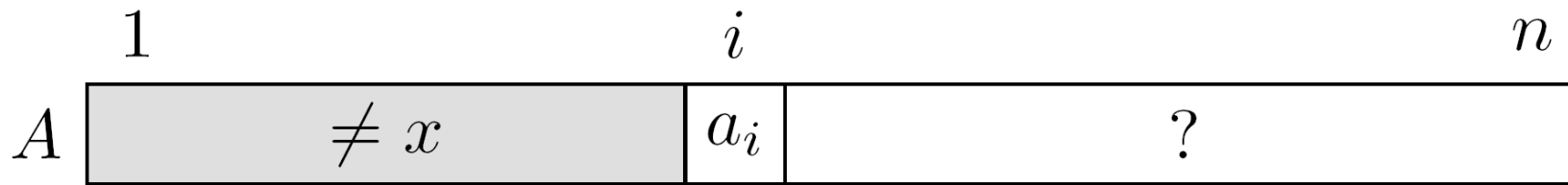
Binary and linear search, logarithms,
longest increasing subsequence, Erdős Szekeres theorem

Searching an unsorted list

24 11 3 7 32 47 5 2 89 12

Find x

**Visual
invariant**



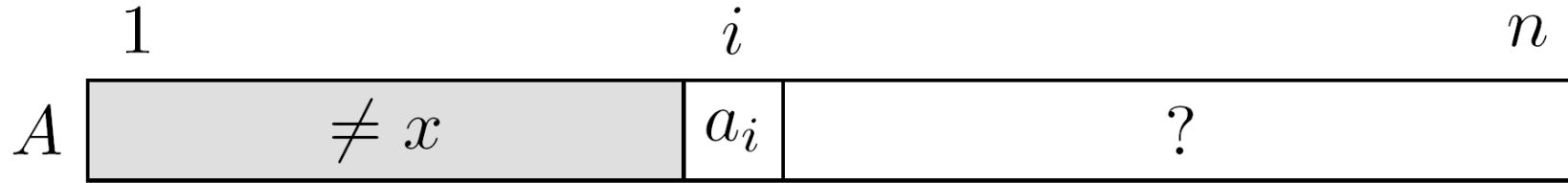
**Formal
invariant**

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

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 $x \notin A$

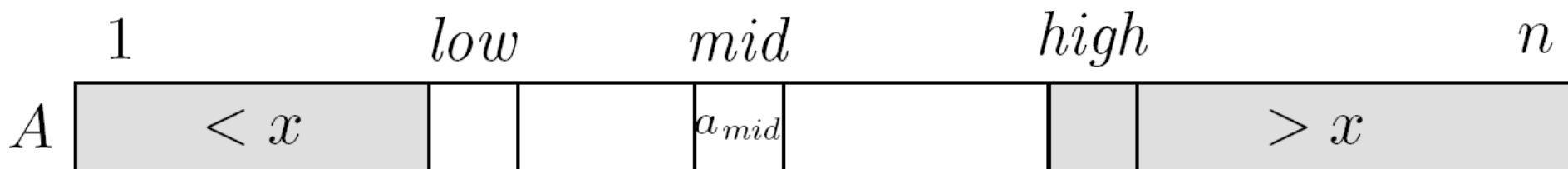
```
1   $i = 1$ 
2  while  $i \leq |A|$  do
3      if  $A[i] = x$  then
4          return  $i$ 
5       $i = i + 1$ 
6  return  $-1$ 
```


Searching a sorted list

3 7 9 11 13 27 33 37 42 89

Find x

**Visual
invariant**



**Formal
invariant**

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

Algorithm BINARYSEARCH(A, x)

Input Sorted array $A[1..n]$ and search key x

Output Index i where $A[i] = x$; -1 if $x \notin A$

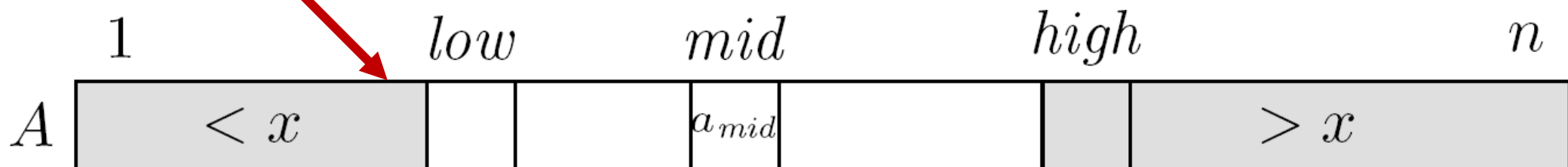
```
1   $low = 1$ 
2   $high = n + 1$ 
3  while  $low < high$  do
4       $mid = \lfloor (low + high) / 2 \rfloor$ 
5      if  $x = A[mid]$  then
6          return  $mid$ 
7      else if  $x < A[mid]$  then
8           $high = mid$ 
9      else if  $x > A[mid]$  then
10          $low = mid + 1$ 
11 return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

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 $\leq : n \rightarrow \lfloor n/2 \rfloor$

$$n \rightarrow \lfloor n/2 \rfloor \rightarrow \lfloor \lfloor n/2 \rfloor / 2 \rfloor \rightarrow \lfloor \lfloor \lfloor n/2 \rfloor / 2 \rfloor / 2 \rfloor \rightarrow \dots \rightarrow 0$$

After k comparisons $\leq n/2^k$ candidates
done the latest when $n/2^k < 1 \Leftrightarrow n < 2^k \Leftrightarrow \log_2 n < k \Leftrightarrow 1 + \lfloor \log_2 n \rfloor \leq k$

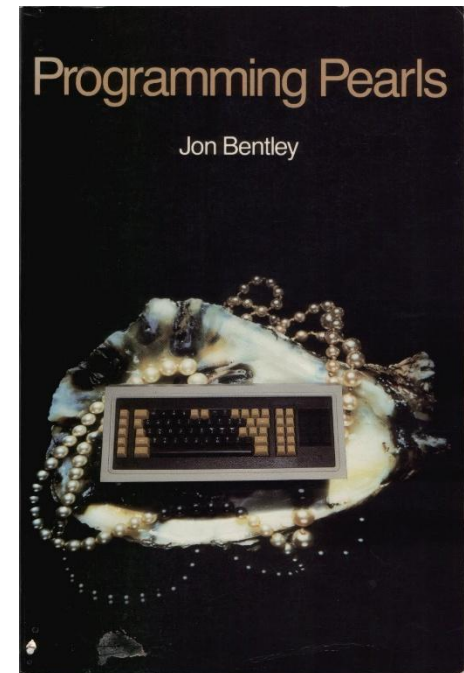
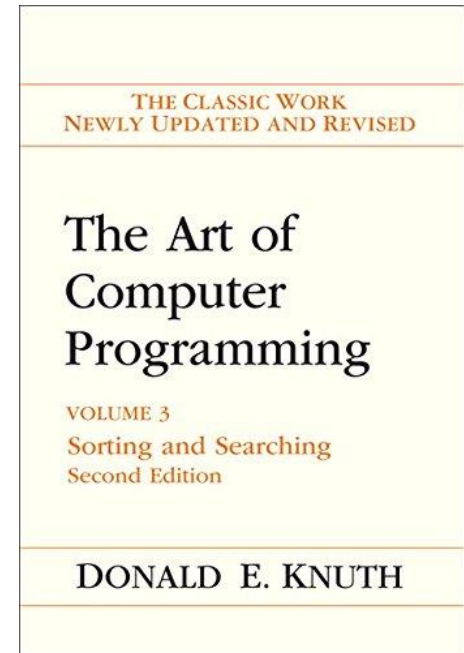
at most $1 + \lfloor \log_2 n \rfloor$ comparisons

Binary search in the literature

- According to Knuth first discussed in the literature
 - John Mauchly i [*Theory and techniqueus 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



EXAMPLES OF ALGOL PROCEDURES

43. *Program for Binary Search*

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

```
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:
```

Binary search in standard libraries

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



Searching a sorted list – lower bound

3	7	9	11	13	27	33	37	42	89
---	---	---	----	----	----	----	----	----	----

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

$$n \rightarrow \lfloor n/2 \rfloor \rightarrow \lfloor \lfloor n/2 \rfloor / 2 \rfloor \rightarrow \lfloor \lfloor \lfloor n/2 \rfloor / 2 \rfloor / 2 \rfloor \rightarrow \dots \rightarrow 0$$

$n \geq 2^{\lfloor \log_2 n \rfloor} \Rightarrow$ 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*

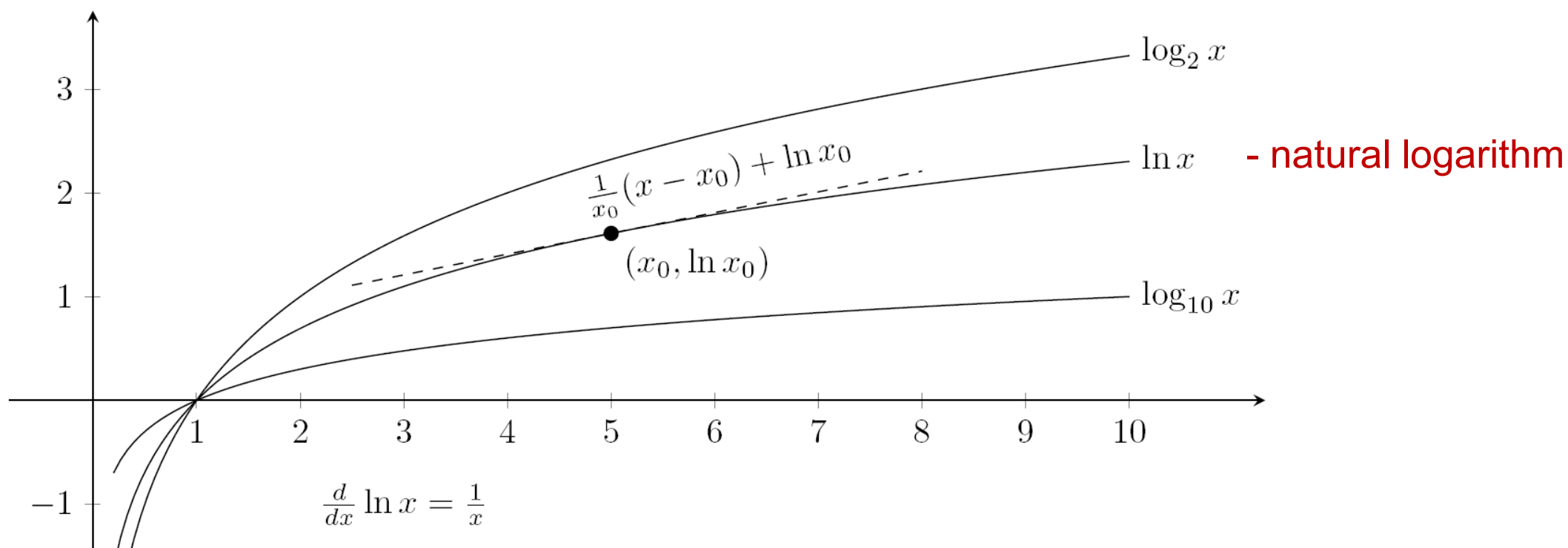
X	3	7	9	11	13	15	33	37	42	89
Y	4	6	8	18	23	27	51			

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

(the 9th smallest element is 15)

Logarithms

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



Logarithms – rules of arithmetic

$$\log_b(x \cdot y) = \log_b x + \log_b y$$

$$\log_b(x/y) = \log_b x - \log_b y$$

$$\log_b(x^p) = p \cdot \log_b x$$

$$\log_b x = \frac{\log_a x}{\log_a b}$$

$$\log_b(b^p) = p$$

$$\frac{d}{dx} \ln x = \frac{1}{x}$$

$$\ln y = \int_1^y \frac{1}{x} dx$$

$$H_n - \ln n \rightarrow \gamma \quad \text{for } n \rightarrow \infty$$

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

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

An algorithmic example (using binary search)

Patience Diff & Longest Increasing Subsequences

Shell command: diff

A.c		B.c
<pre>#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); } return 1; } int main(int argc, char *argv) { frobnitz(fact(1)); } }</pre>	<pre>\$ 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</pre>	<pre>#include <stdio.h> int fib(int n) { if(n < 2) return 1; return fib(n-1) + fib(n-2); } int main(int argc, char *argv) { printf("Your answer is: "); printf("%d\n", *argv); }</pre>

Recent Diffs

Unsaved diff

[Clear diffs](#)Recent diffs are deleted
on refresh

Saved Diffs

No diffs yet

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.

23. int main(int argc, char **argv)

24. {

25. frobnitz(fib(10));

26. }

Text Compare!

Email this comparison

```
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 }
27
```

```
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 }
26
```

Edit texts ...

Switch texts

Compare!

Clear all



```

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 }
27

```

```

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 }
26

```


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)

A.c

```
00 #include <stdio.h>
01
11 // Frobs foo heartily
12 int frobnitz(int foo)
04 {
14     int i;
15     for(i = 0; i < 10; i++)
07     {
08         printf("Your answer is: ");
17     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
22 int main(int argc, char *
23 {
24     frobnitz(fact(10));
25 }
```

B.c

```
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     }
19 }
```

Patience Diff

- 1) Find lines **occurring exactly once** in both texts
- 2) Find a **longest increasing (common) subsequence**

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)

~~30 83 73 80 59 63 41 78 88 82 53 31 22 74 6 38 99 57 43 80~~

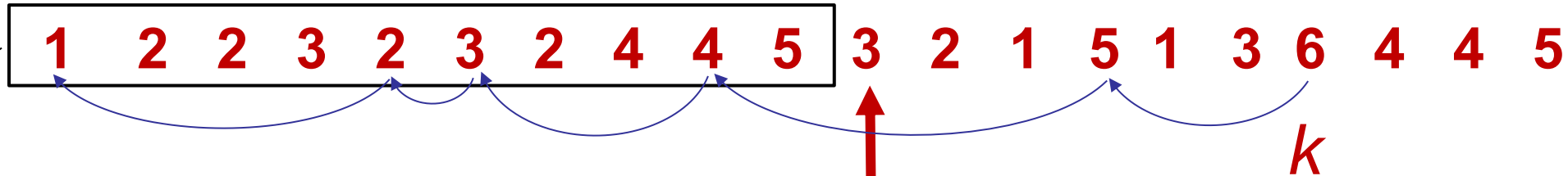
Problem

Erase as few numbers as possible,
so that the remaining numbers 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



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 $\lceil \sqrt{n} \rceil$.

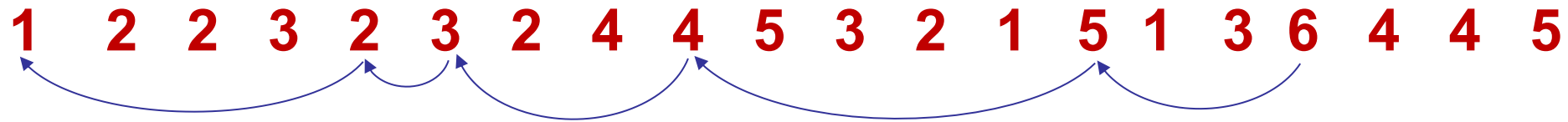
3 1 4 17 6 42 10 8 13 11 28 (increasing)

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

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



The diagram shows a sequence of 20 numbers. Below each number is a red number representing its rank in the sequence. Blue arrows connect the red numbers 1, 2, 3, 4, 5, 1, 3, 6, 4, 4, 5, indicating a decreasing subsequence of the original sequence.

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

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

For every small integer we have a special symbol

Base-10 system ($10 = 9 + 1$)

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

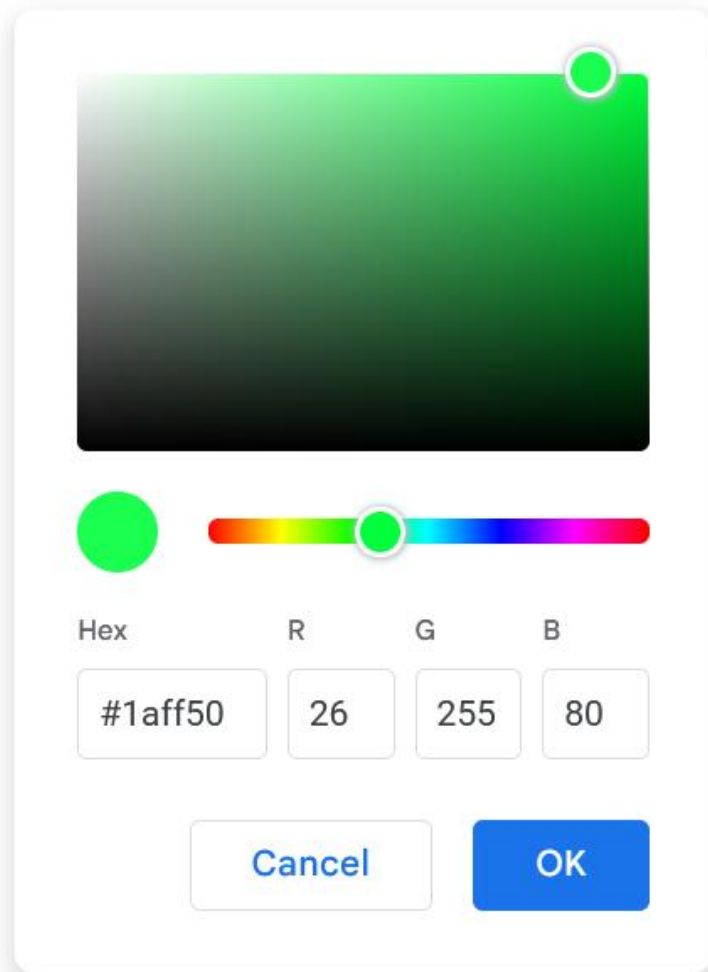
$$\sum_{i=0}^{n-1} d_i \cdot b^i$$

Example: $1849_{10} = \overset{\overset{a}{\downarrow}}{\overset{\overset{b}{\downarrow}}{\overset{\overset{c}{\downarrow}}{\overset{\overset{d}{\downarrow}}{11100111001}}}_2 = 3471_8 = \overset{\overset{x}{\downarrow}}{\overset{\overset{y}{\downarrow}}{\overset{\overset{z}{\downarrow}}{739}}}_{16}$

$1 \cdot 10^3 + 8 \cdot 10^2 + 4 \cdot 10^1 + 9 \cdot 10^0$ $2^{10} + 2^9 + 2^8 + 2^5 + 2^4 + 2^3 + 2^0$ $3 \cdot 8^3 + 4 \cdot 8^2 + 7 \cdot 8^1 + 1 \cdot 8^0$ $7 \cdot 16^2 + 3 \cdot 16^1 + 9 \cdot 16^0$

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 ?

a) 14_{10}

b) 26_{10}

c) 37_{10}

d) 42_{10}

e) 84_{10}

f) Don't know

42_{10} in base 5 ?

a) 42_5

b) 82_5

c) 132_5

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

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

Must (recursively) add
numbers with multiple digits

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

Binary addition

+	0	1
0	0	1
1	1	10

$$\begin{array}{r}
 1 \ 1 \ 1 \\
 1 \ 1 \ 1 \ 0 \ 2 \\
 + \quad 1 \ 0 \ 1 \ 1 \ 2 \\
 \hline
 1 \ 1 \ 0 \ 0 \ 1 \ 2
 \end{array}$$

School method

Addition of several numbers

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

Generalized school method

...carry can have multiple digits

	1	11	10	
			9	9
+			9	9
+			9	9
+			9	9
+			9	9
+			9	9
+			9	9
+			9	9
+			9	9
+			9	9
	1	1	8	8

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

Repeated addition of two numbers

Subtraction – school method

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

$$\begin{array}{rcccccc}
 & -1 & -1 & & -1 & & \\
 & 1 & 0 & 0 & 1 & 0 & 1_2 \\
 - & & 1 & 1 & 0 & 1 & 0_2 \\
 \hline
 & 0 & 0 & 1 & 0 & 1 & 1_2
 \end{array}$$

$$4 - 6 = -2 = -10 + 8$$

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

$$\begin{array}{r} 2 1 4 \\ 6 \cdot 427 \\ \hline 2562 \end{array}$$

Multiplication with
single digit

$$\begin{array}{r} 365 \cdot 427 \\ \hline 1281 \\ 2562 \\ 2135 \\ \hline 155855 \end{array}$$

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

Binary multiplication

.	0	1
0	0	0
1	0	1

Binary multiplication

$$\begin{array}{r} 1011_2 \cdot 1010_2 \\ \hline 1010_2 \\ 0000_2 \\ 1010_2 \\ 1010_2 \\ 1010_2 \\ \hline 1111001_2 \end{array}$$

The diagram illustrates the binary multiplication of 1011_2 and 1010_2 . The multiplicand 1011_2 and multiplier 1010_2 are at the top. A horizontal line separates them from the partial products. The partial products are shown below the line, aligned to the right. The first partial product is 1010_2 , the second is 0000_2 , and the third is 1010_2 . The final result is 1111001_2 , which is the sum of all partial products. Red dashed lines and arrows indicate the shifting of the partial products to the left by one position for each bit in the multiplier.

Binary multiplication of blocks with 1s

$$\overbrace{11111}^j \overbrace{00000}^i {}_2 = 1 \overbrace{000000000000}^{i+j} {}_2 - 1 \overbrace{00000}^i {}_2$$

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

Division

$$\begin{array}{r}
 \overbrace{1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0}_x \text{ }_2 / \overbrace{1\ 0\ 1\ 0\ 1}_y \text{ }_2 = \overbrace{1\ 0\ 1\ 1\ 1}_i \text{ }_2 \\
 \underline{1\ 1\ 1\ 1\ 1\ 0\ 1\ 0\ 0} \\
 -\ 1\ 0\ 1\ 0\ 1\ \underline{\hspace{1cm}}\ \textcolor{red}{p} \\
 \underline{1\ 0\ 1\ 0\ 0\ 1\ 0\ 0} \\
 -\ 0\ 0\ 0\ 0\ 0\ \hspace{1cm} \\
 \underline{1\ 0\ 1\ 0\ 0\ 1\ 0\ 0} \\
 -\ 1\ 0\ 1\ 0\ 1\ \hspace{1cm} \\
 \underline{1\ 0\ 1\ 0\ 0\ 0\ 0\ 0} \\
 -\ 1\ 0\ 1\ 0\ 1\ \hspace{1cm} \\
 \underline{1\ 0\ 0\ 1\ 1\ 0\ 0\ 0} \\
 -\ 1\ 0\ 1\ 0\ 1\ \hspace{1cm} \\
 \underline{1\ 0\ 0\ 0\ 1\ \textcolor{red}{r}}
 \end{array}$$

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  $y \cdot 2^{p+1} \leq x$ 
3       $p = p + 1$ 
4   $i = 0$ 
5   $r = x$ 
6  # Invariant:  $x = i \cdot y + r$  and  $r < y \cdot 2^{p+1}$ 
7  while  $y \leq r$ 
8      if  $y \cdot 2^p \leq r$ 
9           $i = i + 2^p$  # set position  $p$  in  $i = 1$ 
10          $r = r - y \cdot 2^p$ 
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 $d_0, d_1, \dots$ of the b -ary representation of x

```
1   $p = 0$ 
2  while  $x > 0$  do
3       $i = \lfloor x/b \rfloor$  ← INTEGERDIVISION( $x, b$ )
4       $d_p = x - i \cdot b$  ← remainder
5       $x = i$ 
6       $p = p + 1$ 
```

$x = 42 \quad b = 2$

p	x	i	$i \cdot b$	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				

Redundant number systems

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

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

$$42_{10} = 101010_2 = \underset{\substack{\parallel \\ 2 \cdot 2^4 + 1 \cdot 2^3 + 2 \cdot 2^0}}{2100}_2 = \underset{\substack{\parallel \\ 2 \cdot 2^4 + 2 \cdot 2^3 + (-1) \cdot 2^2 + (-1) \cdot 2^1}}{22-1-10}_2 \Rightarrow \text{not unique}$$

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

Algorithms and Data Structures

Induction and invariants

Proof by induction

- We wish to prove an infinite sequence of statements

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

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

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

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

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

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

□

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:

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

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

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 $n = 0$:

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

- 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:

$$\begin{aligned} \sum_{i=0}^{n+1} x^i &= x^{n+1} + \sum_{i=0}^n x^i \stackrel{\text{i.h.}}{=} x^{n+1} + \frac{x^{n+1}-1}{x-1} \\ &= \frac{(x-1)x^{n+1} + x^{n+1} - 1}{x-1} = \frac{x^{n+2} - 1}{x-1} = \frac{x^{(n+1)+1} - 1}{x-1} \quad \square \end{aligned}$$

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

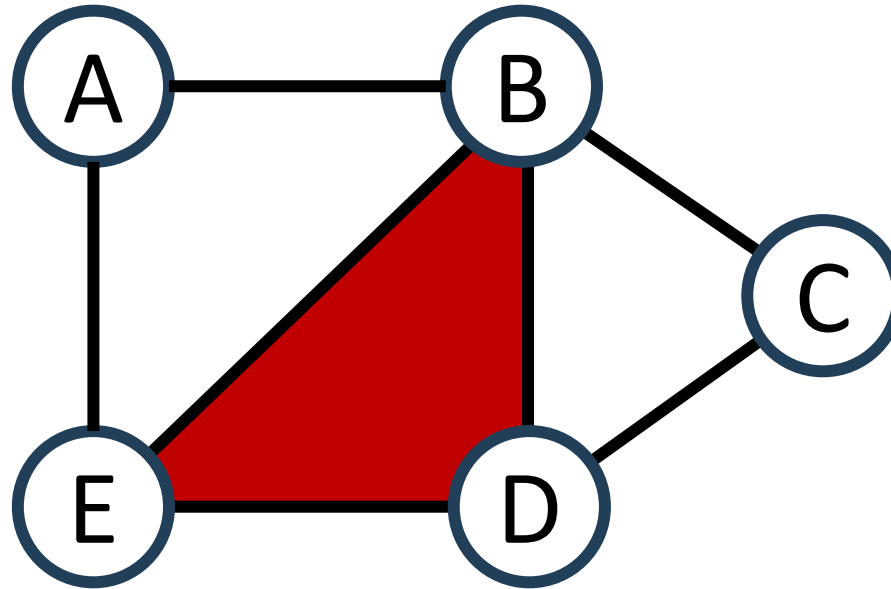
We must prove: $\sum_{i=1}^{n+1} i^2 = \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6}$

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

$$\begin{aligned} &\stackrel{\text{i.h.}}{=} (n+1)^2 + \frac{n(n+1)(2n+1)}{6} = \frac{6(n+1)^2 + n(n+1)(2n+1)}{6} \\ &= \frac{(n+1)(2n^2 + 7n + 6)}{6} = \frac{(n+1)(n+2)(2n+3)}{6} \\ &= \frac{(n+1)((n+1)+1)(2(n+1)+1)}{6} \end{aligned}$$

□

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)

[Dictionary](#)[Thesaurus](#)[Games](#)[Word of the Day](#)

Dictionary

Definition

[Synonyms](#)[Example Sentences](#)[Word History](#)[Phrases Containing](#)[Rhymes](#)[Entries Near](#)

invariant adjective

in·vari·ant (,)in-'ver-ē-ənt

[Synonyms of *invariant*](#) >

: **CONSTANT, UNCHANGING**

specifically : unchanged by specified mathematical or physical operations or transformations

| *invariant* factor

invariant noun

$i \leftarrow 0$

$x \leftarrow 100$

while $i \leq 10$ **do**

$x \leftarrow x + 7$

$i \leftarrow i + 1$

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

Examples for **I**

- $i \geq 0$
- $x \geq 100$
- $i \leq x$

{ I }

$x = 100 + 7 * i \wedge i \leq 11$
 $\wedge x$ and i are integers

$i \leftarrow 0$

$x \leftarrow 100$

while $i \leq 10$ **do**

$x \leftarrow x + 7$

$i \leftarrow i + 1$

x' and i' are the new values

$i=0 \wedge x=100 \rightarrow x = 100+7*i \wedge i \leq 11$

$x=100+7*i \wedge i \leq 11 \wedge i \leq 10$
 $\wedge x'=x+7 \wedge i'=i+1$

$\rightarrow x'=(100+7*i)+7 = 100+7*i' \wedge i' \leq 11$

$\neg(i \leq 10) \wedge (x = 100+7*i \wedge i \leq 11) \rightarrow i = 11 \wedge x = 177$

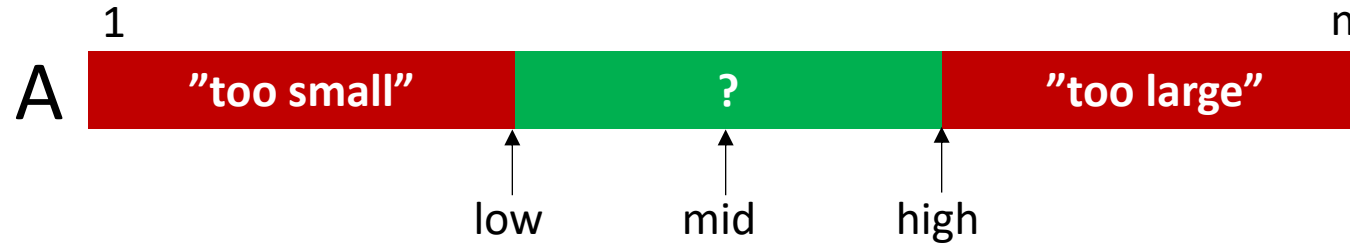
An invariant **I should satisfy**

- 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

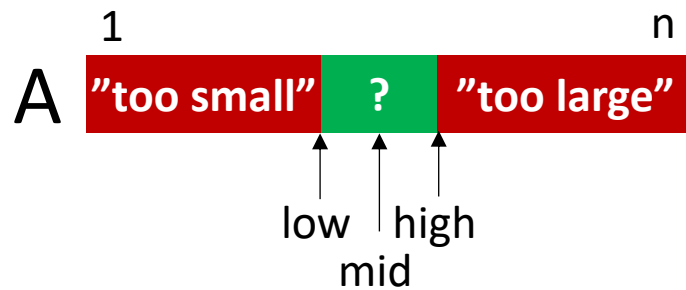
Use **I** and that the loop condition is false when the loop terminations to draw a conclusion

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  
     $mid \leftarrow \langle\langle f(low, high) \rangle\rangle$   
    << update >>
```

I_1 : $(A[i] < x \text{ for } 1 \leq i \leq low) \wedge (x < A[i] \text{ for } high \leq i \leq n)$

I_2 : $(A[i] < x \text{ for } 1 \leq i \leq low) \wedge (x \leq A[i] \text{ for } high \leq i \leq n)$

I_3 : $(A[i] < x \text{ for } 1 \leq i < low) \wedge (x \leq A[i] \text{ for } high < i \leq n)$

I_4 : $(A[i] < x \text{ for } 1 \leq i < low) \wedge (x \leq A[i] \text{ for } high \leq i \leq n)$

...

<< initialize >>

- a) $\text{low} \leftarrow 0; \text{high} \leftarrow n+1$
- b) $\text{low} \leftarrow 1; \text{high} \leftarrow n$
- c) $\text{low} \leftarrow 0; \text{high} \leftarrow n$
- d) $\text{low} \leftarrow 1; \text{high} \leftarrow n+1$
- e) Don't know

I: $(A[i] < x \text{ for } 1 \leq i < \text{low}) \wedge (x \leq A[i] \text{ for } \text{high} < i \leq n)$

<< loop condition >>

- a) $\text{low} = \text{high}$
- b) $\text{low} \neq \text{high}$
- c) $\text{low} < \text{high}$
- d) $\text{low} \leq \text{high}$
- e) $\text{low} > \text{high}$
- f) Don't know

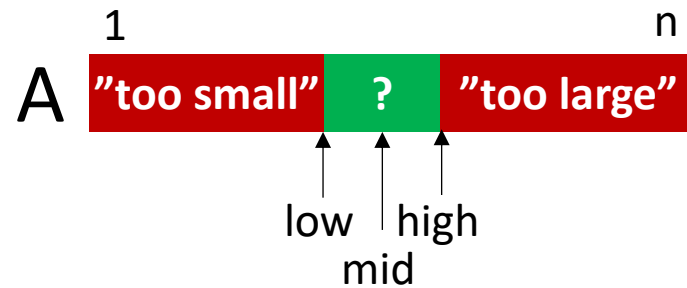
I: $(A[i] < x \text{ for } 1 \leq i < \text{low}) \wedge (x \leq A[i] \text{ for } \text{high} < i \leq n)$

<< update >>

- a) if $A[\text{mid}] < x$ then $\text{low} \leftarrow \text{mid}$ else $\text{high} \leftarrow \text{mid}$
- b) if $A[\text{mid}] < x$ then $\text{low} \leftarrow \text{mid}+1$ else $\text{high} \leftarrow \text{mid}$
- c) if $A[\text{mid}] < x$ then $\text{low} \leftarrow \text{mid}$ else $\text{high} \leftarrow \text{mid}+1$
- d) if $A[\text{mid}] < x$ then $\text{low} \leftarrow \text{mid}+1$ else $\text{high} \leftarrow \text{mid}-1$
- e) Don't know

I: $(A[i] < x \text{ for } 1 \leq i < \text{low}) \wedge (x \leq A[i] \text{ for } \text{high} < i \leq n)$

Binary search (in a sorted array)



```
low  $\leftarrow$  1; high  $\leftarrow$  n
{ I } while low  $\leq$  high do
    mid  $\leftarrow$   $\lfloor \text{low} + (\text{high} - \text{low}) / 2 \rfloor$ 
    if  $A[\text{mid}] < x$  then
        low  $\leftarrow$  mid+1
    else
        high  $\leftarrow$  mid-1
```

I: $(A[i] < x \text{ for } 1 \leq i < \text{low}) \wedge (x \leq A[i] \text{ for } \text{high} < i \leq n)$

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: $(A[i] < x \text{ for } 1 \leq i < \text{low}) \wedge (x \leq A[i] \text{ for } \text{high} < i \leq n)$

Algorithm 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
$p \leq p_0$	☒	☐
$x^p = r$	☐	☒
$x \leq x_0$	☐	☒
$r \cdot x_0^{p_0} = x^p$	☐	☒
$x_0^{p_0} = r \cdot x^p$	☒	☐

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
$1 \leq r < p$	☒	☐
$2p \leq n$	☐	☒
$p = 2^r$	☒	☐
$p = 2r$	☐	☒
$p = 2^{\text{intlog}(p)}$	☒	☐

Invariants

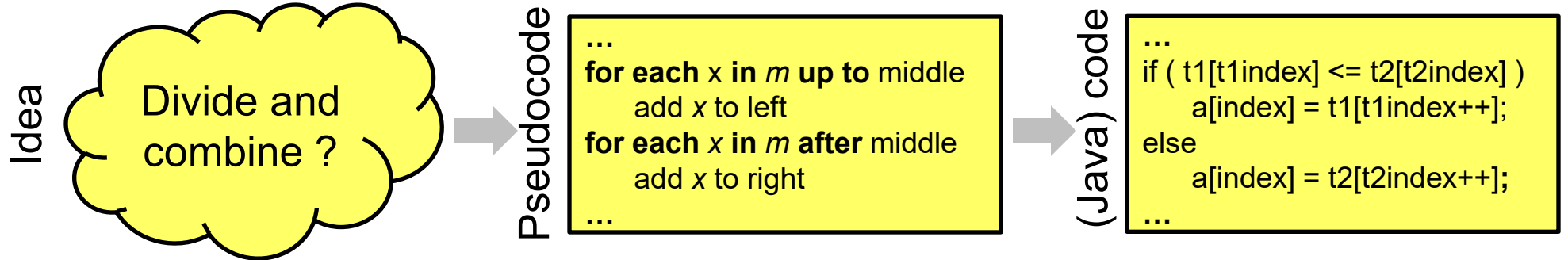
- A tool to analyze the states of a loop in an algorithm
- Design tool "Invariant $\rightarrow$ Code"
- Can be used to identify central properties of the state of a data structure

Algorithms and Data Structures

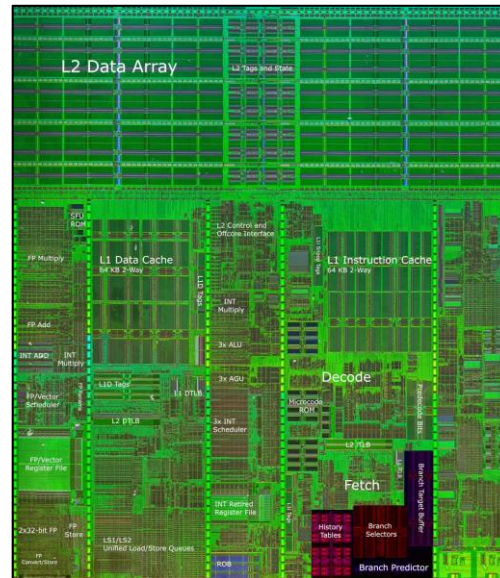
Analysis tools, RAM model,
Insertion-Sort, O-notation
[CLRS, Chapter 1-3.2]

What is the running time of an algorithm?

From idea to program execution

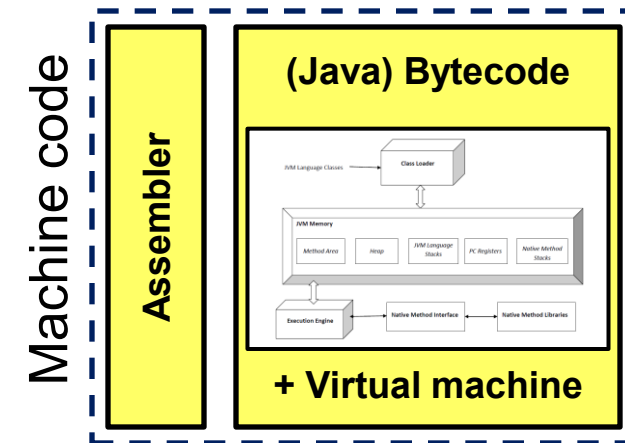


Microcode
Virtual memory
TLB
L1, L2,... cache
Branch Prediction
Pipelining
...



AMD Athlon (2006)
chipsandcheese.com

Program
excursion



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

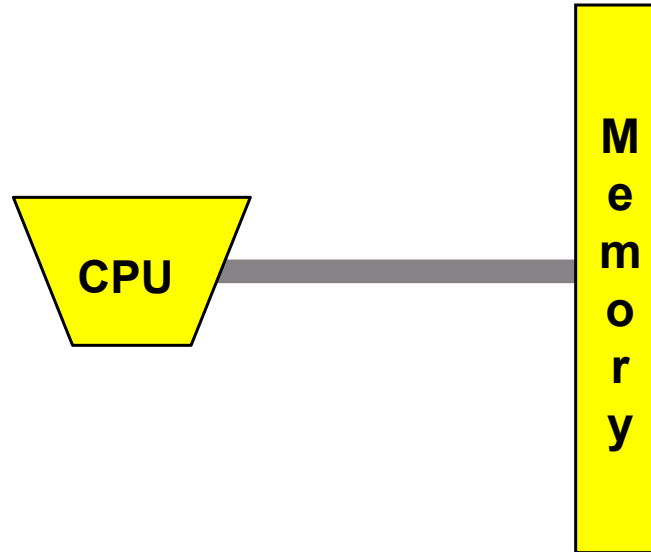
- the algorithm **terminates** on all inputs
- output **correct** on all inputs

Efficiency

- Optimize algorithms to use **minimal** time
- space, additions,... or **maximal** parallelism...
- $\sim n^2$ is better than $\sim n^3$: **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) $\sim n$
- b) $\sim n \log n$
- c) $\sim n^2$
- d) $\sim n^3$
- 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       $A[j + 1] = key$ 
```

Input n $n-1$ $n-2$... 4 3 2 1

Example

INSERTION-SORT(A, n)

$6n - 2$ 1 **for** $i = 2$ **to** n

$$(n-1) \times 4 \quad 2 \quad key = A[i]$$

```
0 3 // Insert  $A[i]$  into the sorted subarray  $A[1 : i - 1]$ 
```

$$(n-1) \times 3 \quad 4 \quad j = i - 1$$
$$\sum_{i=2}^n \begin{cases} \leq i \times 7 & 5 \\ \leq (i-1) \times 5 & 6 \\ \leq (i-1) \times 3 & 7 \end{cases} \quad \begin{array}{l} \mathbf{while} \ j > 0 \text{ and } A[j] > key \\ \quad A[j + 1] = A[j] \\ \quad j = j - 1 \end{array}$$

$(n-1) \times 5 \quad 8 \quad A[j + 1] = key$

```
1  $i = 2$  // write 2 to  $i$ : 1 instruction
```

```
 $n \times 3$  while  $i \leq n$  // read  $i$ , read  $n$ , compare : 3 instructions
```

• • • •

$(n-1) \times 3$ $i = i + 1$ // read i , add 1, write i : 3 instructions

```
// read i, add address of A to get address of A[i], read A[i],  
write to key : 4 instructions
```

```
// read  $i$ , subtract -1, write  $j$ : 3 instructions
```

```
// 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
```

```
// 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
```

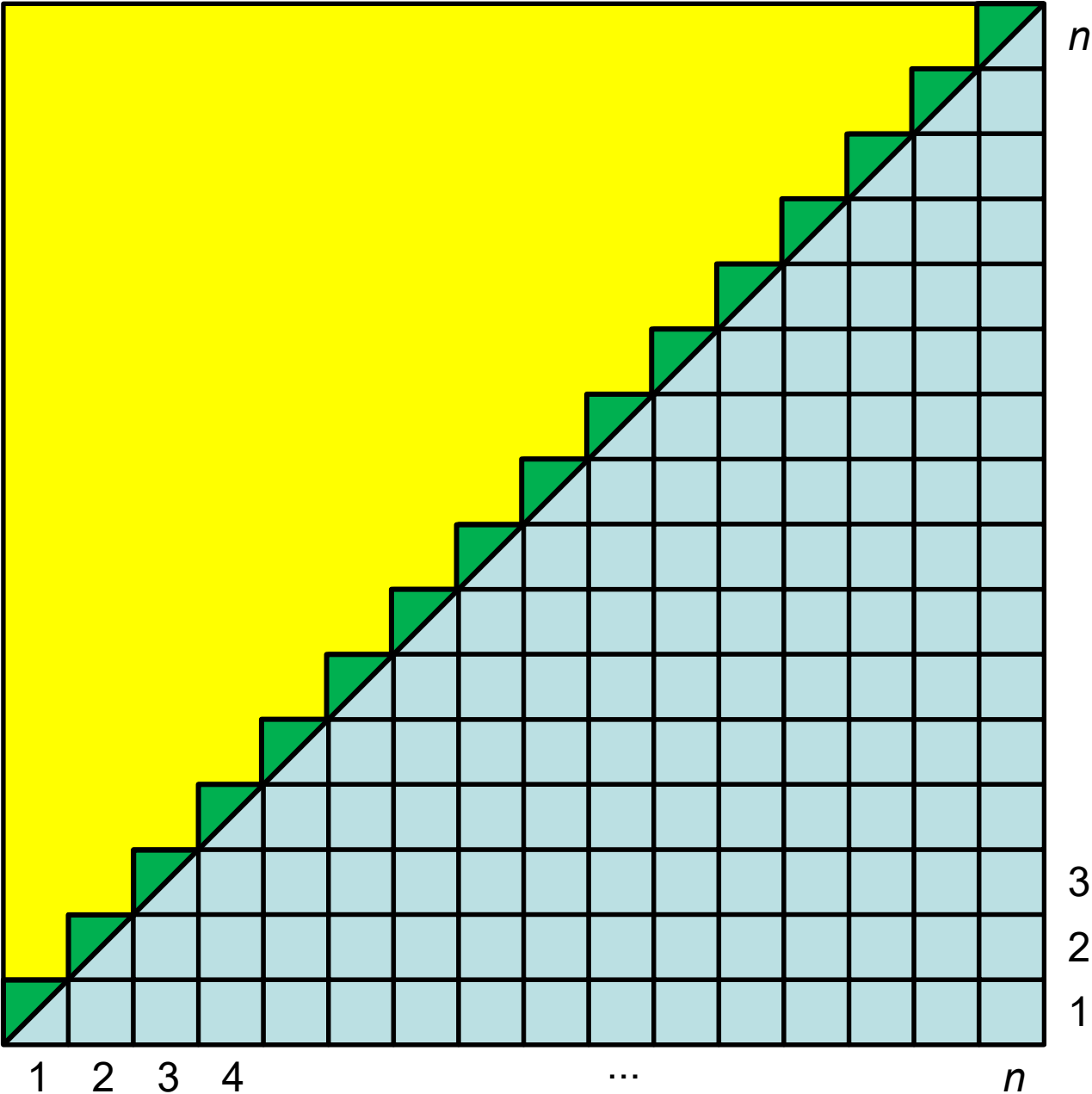
```
// read  $j$ , subtract -1, write  $j$  : 3 instructions
```

```
// read j, add +1, add address of A to get address of A[j+1],  
read key, write to A[j+1] : 5 instructions
```

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

[illegible]

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



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

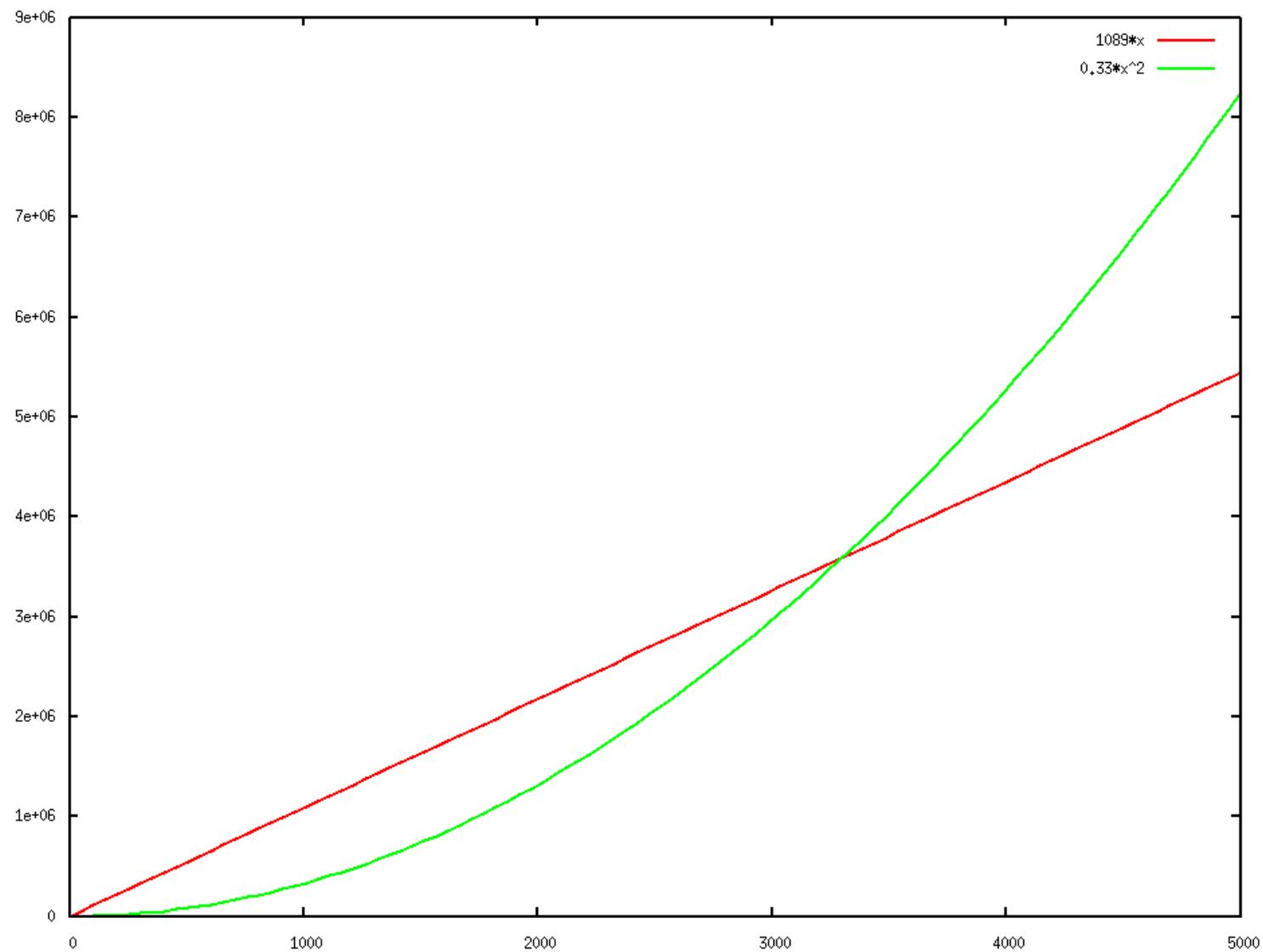
- Example of **pseudocode**
- Detailed analysis – cumbersome work
- Time: **worst-case** ($\sim n^2$) and **best-case** ($\sim n$) very different
- Time: **average** ($\sim n^2$)
- Faster on $\sim$ sorted input: **adaptive**

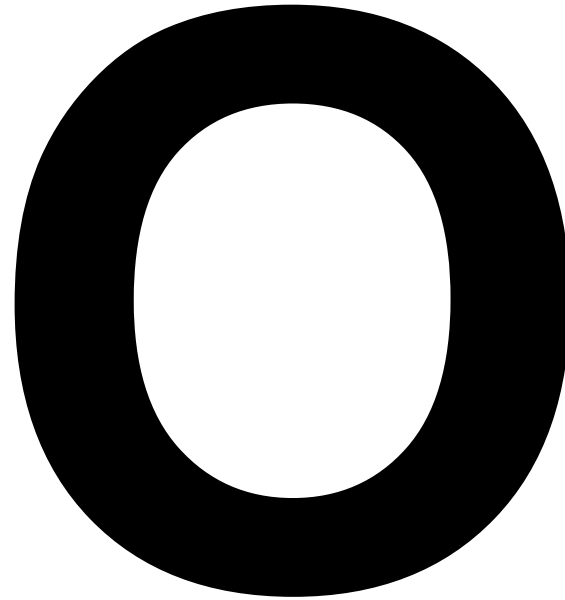
Asymptotic notation

- Basic assumptions:
 - $\sim n^2$ is better than $\sim n^3$
 - constants are not essential
- **Mathematical precise** way to work with " $\sim$ "
- Examples:

$$\begin{array}{rcl} 87 \cdot n^2 & " \leq " & 12 \cdot n^3 \\ 1089 \cdot n & " \leq " & 0.33 \cdot n^2 \\ 7 \cdot n^2 + 25 \cdot n & " \leq " & n^2 \end{array}$$

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





- notation

... and friends

Ω (big omega)

Θ (theta)

ω (small omega)

o (small o)

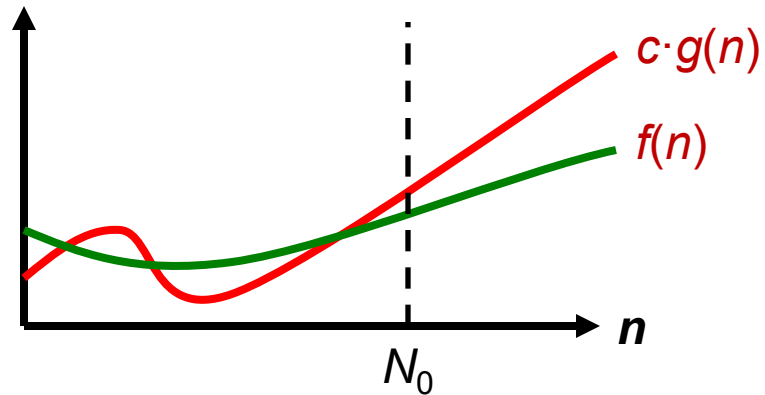
O-notation

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

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

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

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

(for positive monotone functions)

$$f_1(x) = O(g_1(x)) \text{ and } f_2(x) = O(g_2(x))$$

implies

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

- a) Yes
- b) No
- c) Don't know

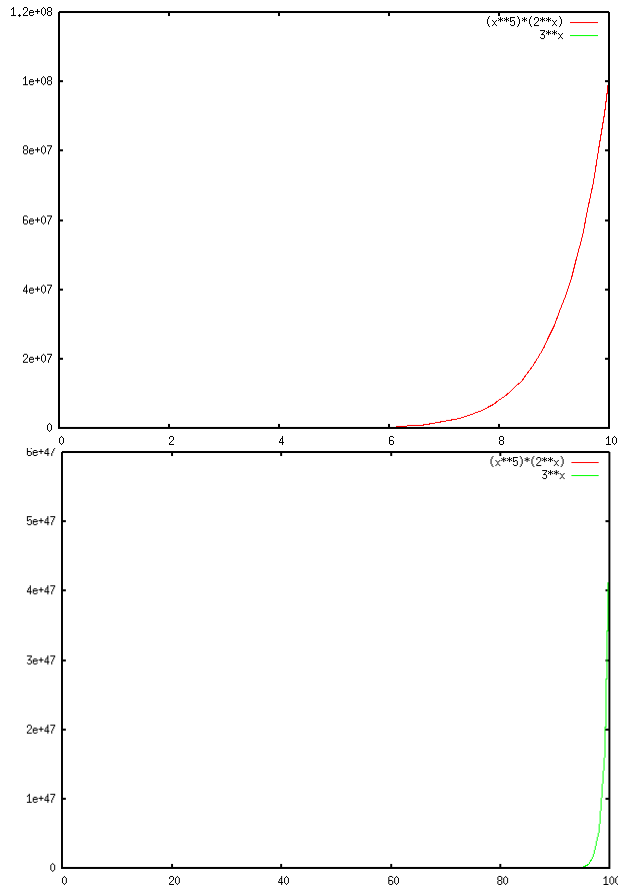
Examples : O

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

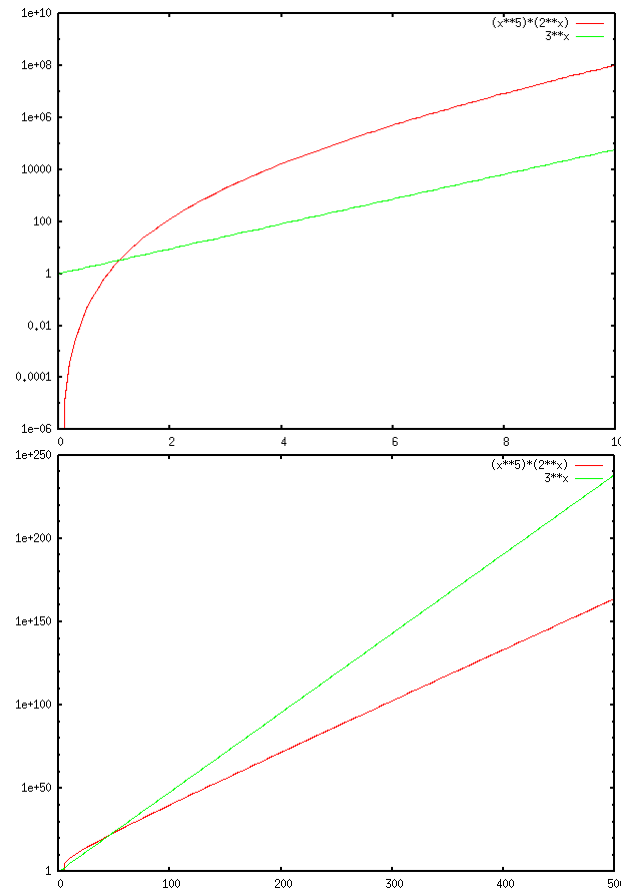
Visual test of $n^5 \cdot 2^n = O(3^n)$?



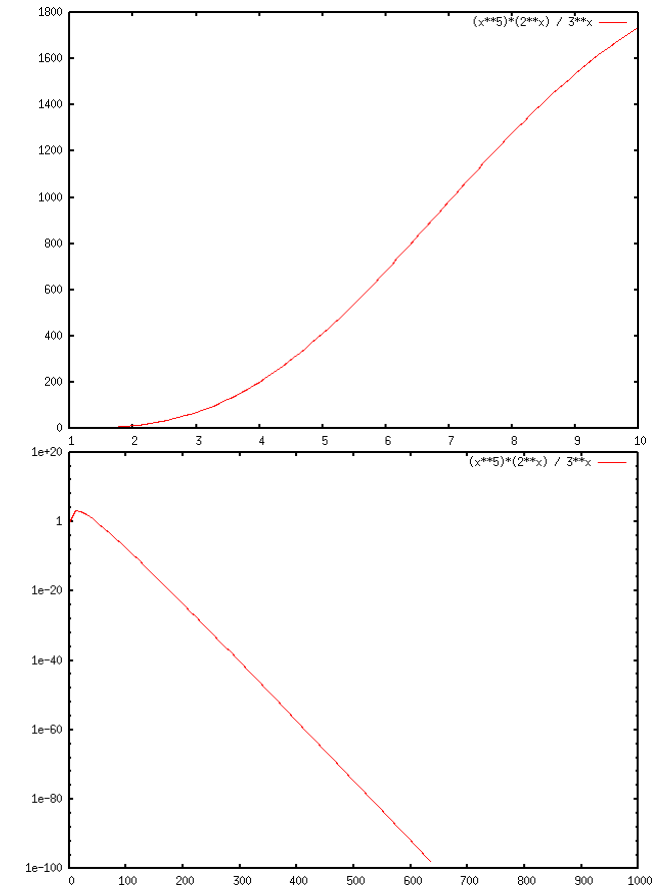
Plot of two functions
– not very informative



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 $n^5 \cdot 2^n \leq c \cdot 3^n$ for all $n \geq N_0$, for appropriate choices of c and N_0

Proof:

$$(5/\log_2 (3/2))^2 \leq n \quad \text{for } n \geq 73$$

$$\Downarrow \\ 5/\log_2 (3/2) \leq \sqrt{n} = n/\sqrt{n} \leq n/\log_2 n$$

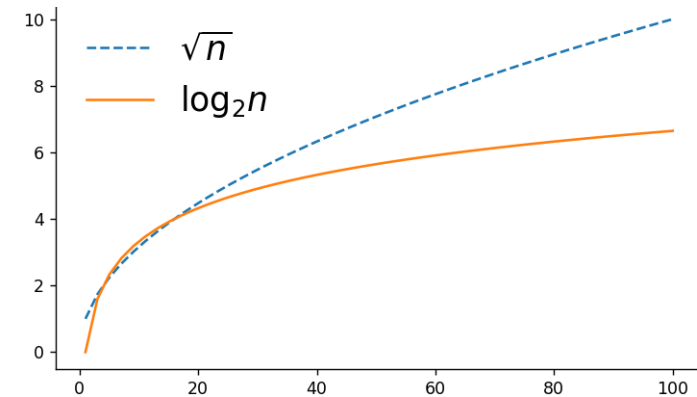
$$\Downarrow \\ 5 \cdot \log_2 n \leq n \cdot \log_2 (3/2)$$

$$\Downarrow \\ \log_2 (n^5) \leq \log_2 (3/2)^n$$

$$\Downarrow \\ n^5 \leq (3/2)^n = 3^n / 2^n$$

$$\Downarrow \\ n^5 \cdot 2^n \leq 3^n$$

since $\sqrt{n} \geq \log_2 n$ for $n \geq 17$



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

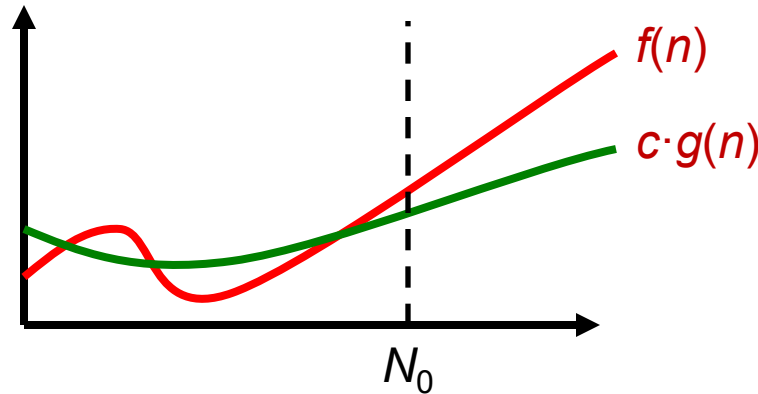
Theorem $\text{poly}(n) \cdot a^n = O(b^n)$ for all $1 \leq a < b$

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 $n \geq N_0$:

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

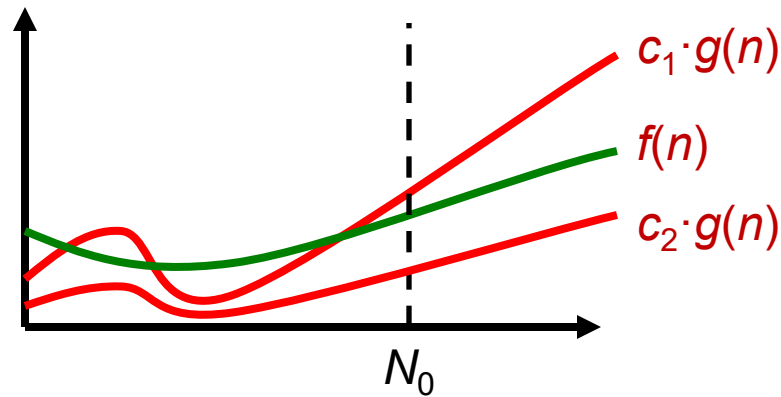


Intuitively: $f(n)$ is "larger than or equal to" $g(n)$, or $g(n)$ is "dominated by" $f(n)$

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" $g(n)$

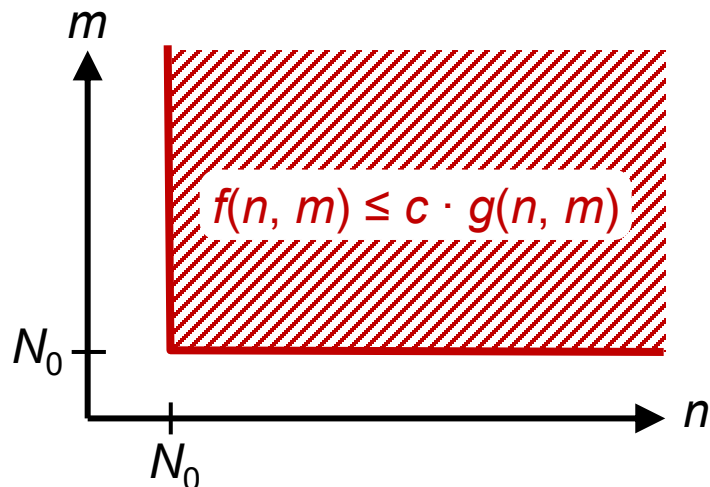
Multi-variable O-notation

We would like to be able to write

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

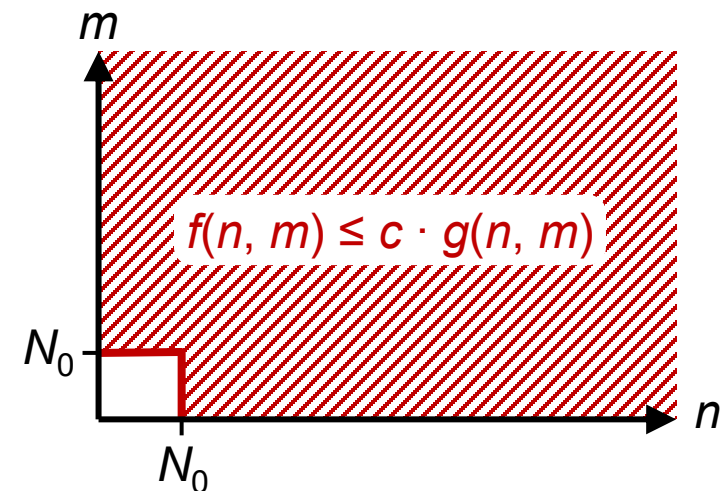
The literature is not precise/consistent...

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



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

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



CommitStrip.com

Algorithms and Data Structures

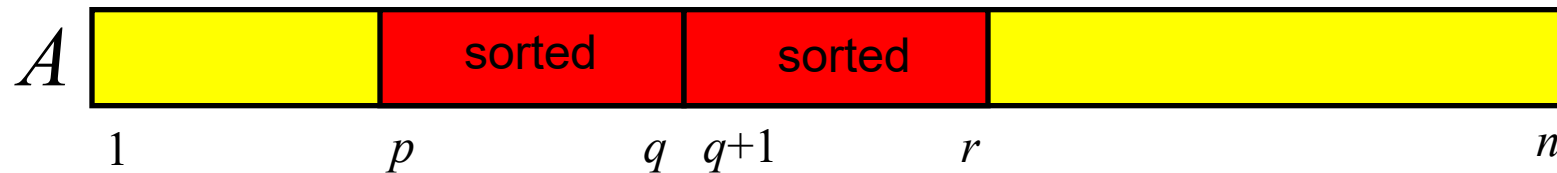
Merge-Sort [CLRS, Chapter 2.3]
Heaps [CLRS, Chapter 6]

Merge-Sort

(example of the divide-and-conquer technique)

MERGE-SORT(A, p, r)

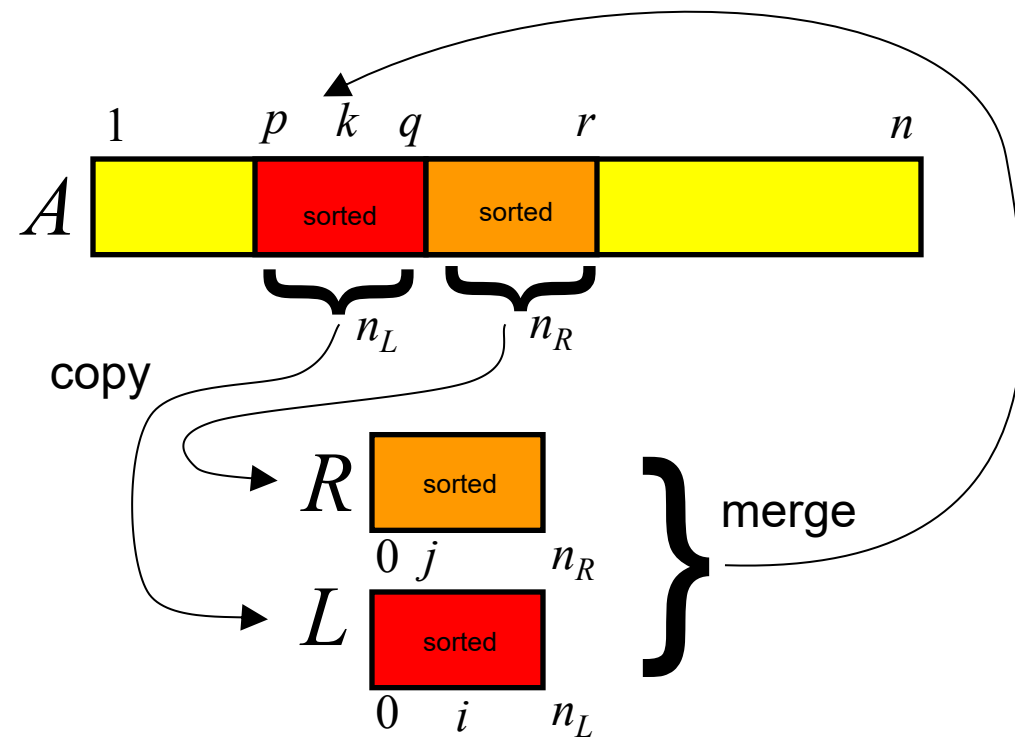
```
1  if  $p \geq r$                                 // zero or one element?
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$                     // midpoint of  $A[p : r]$ 
4  MERGE-SORT( $A, p, q$ )                          // recursively sort  $A[p : q]$ 
5  MERGE-SORT( $A, q + 1, r$ )                      // 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($A, 1, n$)

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,
12 //   copy the smallest unmerged element back into  $A[p : r]$ .
13 while  $i < n_L$  and  $j < n_R$ 
14     if  $L[i] \leq R[j]$ 
15          $A[k] = L[i]$ 
16          $i = i + 1$ 
17     else  $A[k] = R[j]$ 
18          $j = j + 1$ 
19      $k = k + 1$ 
20 // Having gone through one of  $L$  and  $R$  entirely, copy the
21 //   remainder of the other to the end of  $A[p : r]$ .
22 while  $i < n_L$ 
23      $A[k] = L[i]$ 
24      $i = i + 1$ 
25      $k = k + 1$ 
26 while  $j < n_R$ 
27      $A[k] = R[j]$ 
28      $j = j + 1$ 
29      $k = k + 1$ 
```

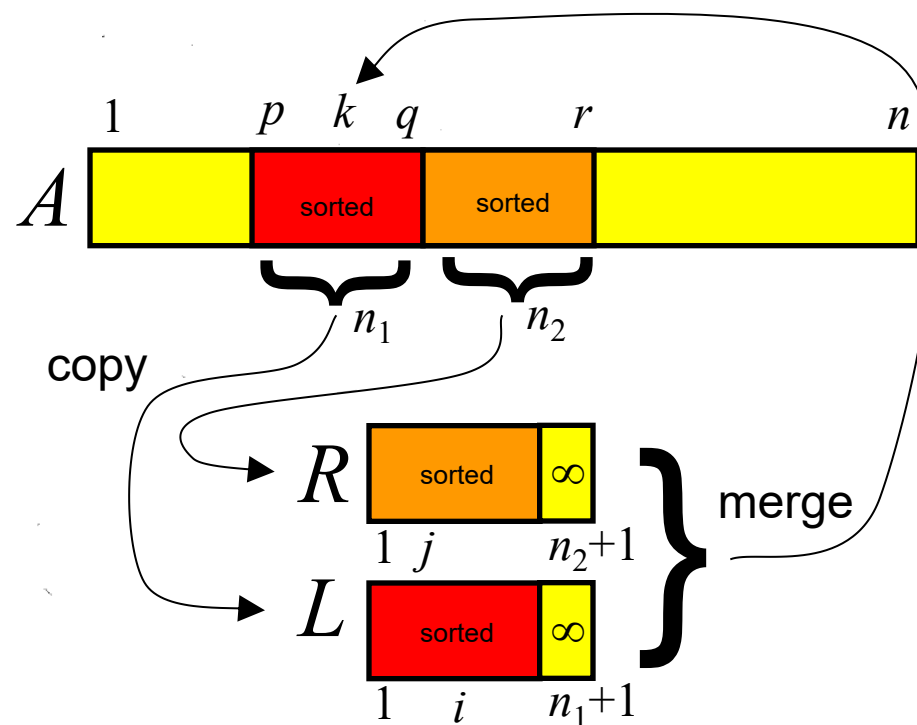


MERGE(A, p, q, r)

```

1   $n_1 = q - p + 1$ 
2   $n_2 = r - q$ 
3  let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays
4  for  $i = 1$  to  $n_1$ 
5       $L[i] = A[p + i - 1]$ 
6  for  $j = 1$  to  $n_2$ 
7       $R[j] = A[q + j]$ 
8   $L[n_1 + 1] = \infty$ 
9   $R[n_2 + 1] = \infty$ 
10  $i = 1$ 
11  $j = 1$ 
12 for  $k = p$  to  $r$ 
13     if  $L[i] \leq R[j]$ 
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)

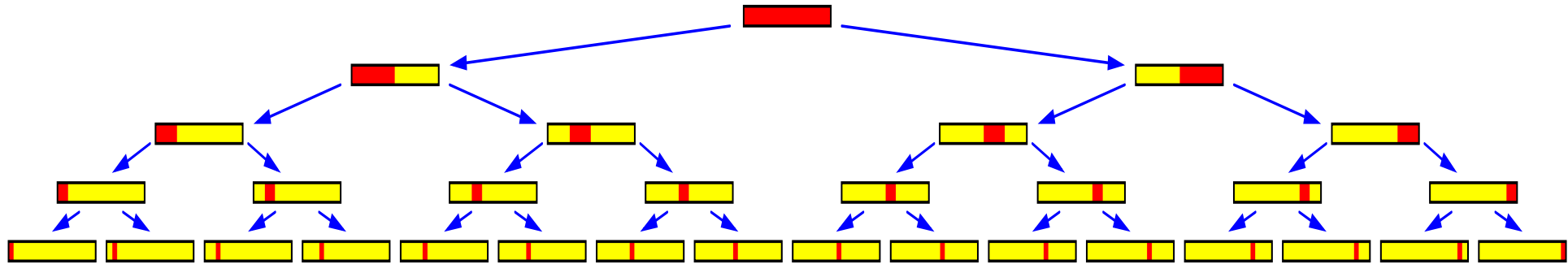
- a) $O(\log n)$
- b) $O(n)$
- c) $O(n \log n)$
- d) $O(n^2)$
- e) Don't know

```
MERGE-SORT( $A, p, r$ )  
1  if  $p \geq r$                                 // zero or one element?  
2      return  
3   $q = \lfloor (p + r)/2 \rfloor$                         // 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 : $O(n \cdot \# \text{ levels}) = O(n \cdot \log_2 n)$

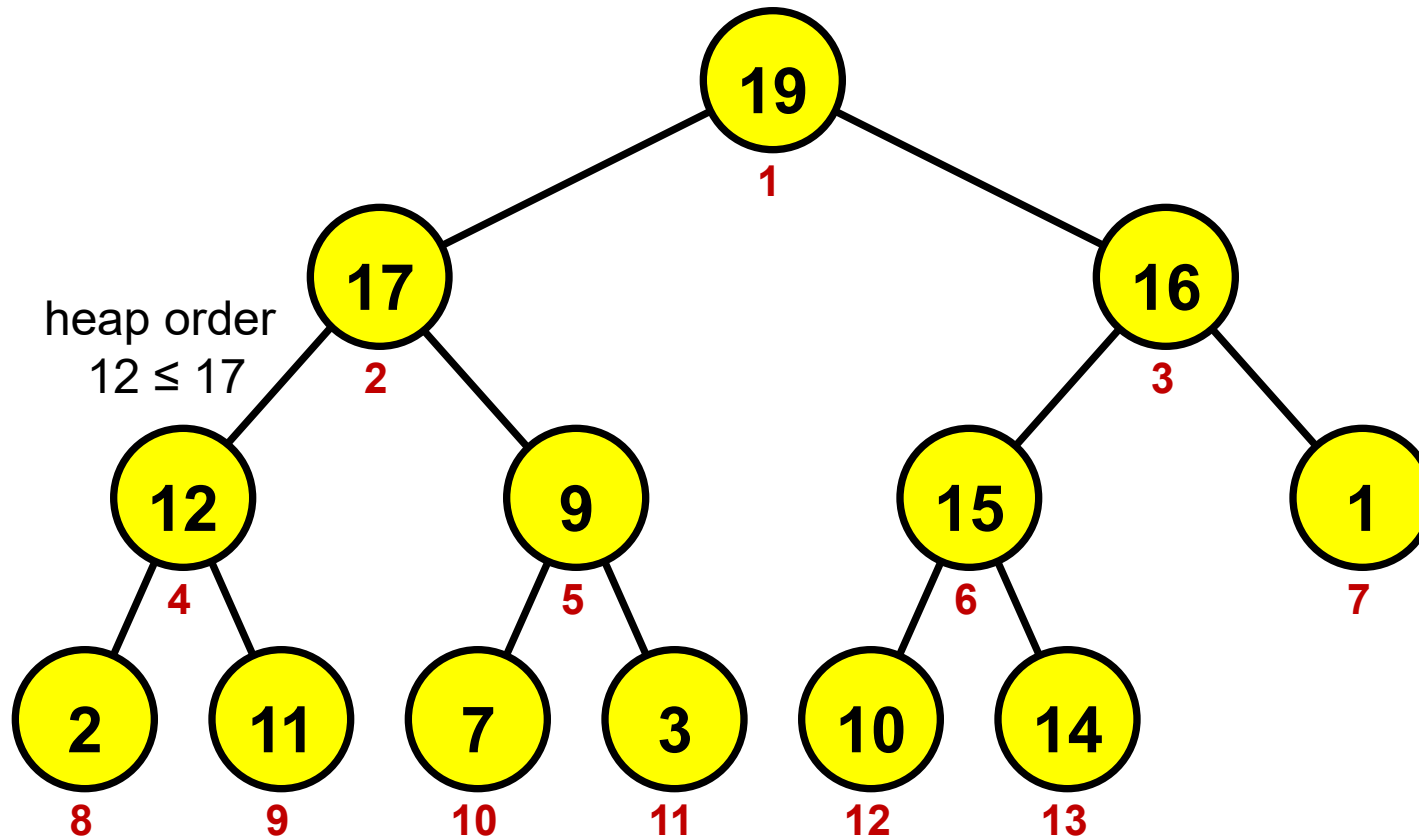
MERGE-SORT(A, p, r)

```
1  if  $p \geq r$                                 // zero or one element?
2      return
3   $q = \lfloor (p + r) / 2 \rfloor$                     // 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

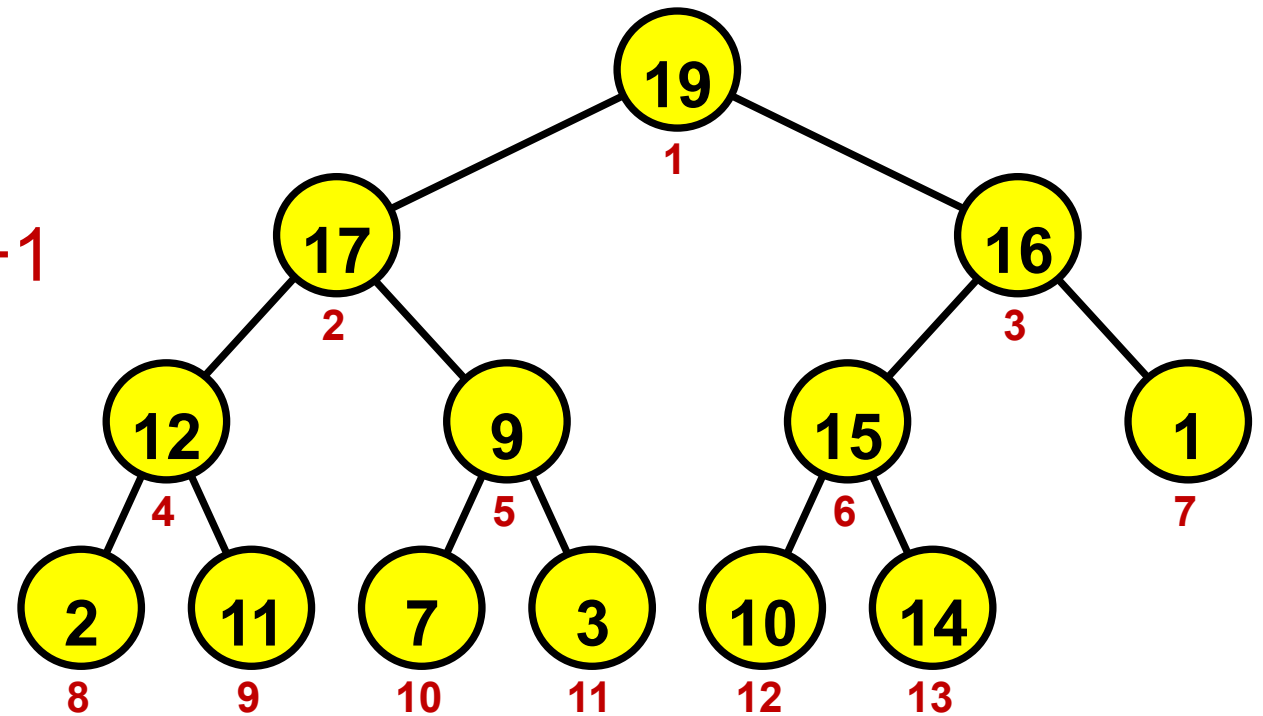
Valid Max-Heap ?

- 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

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

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)

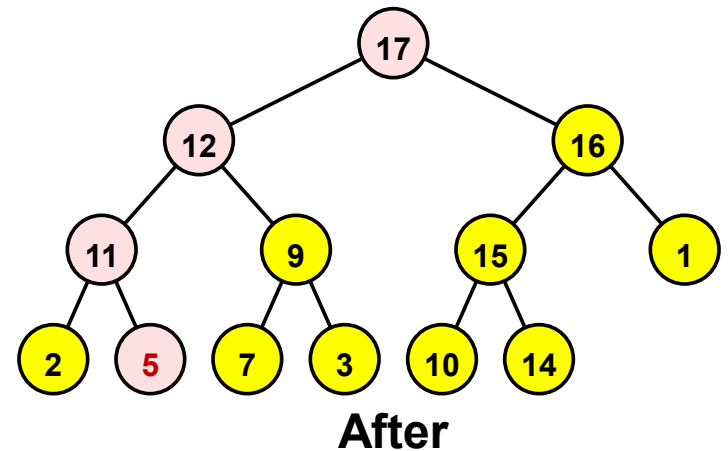
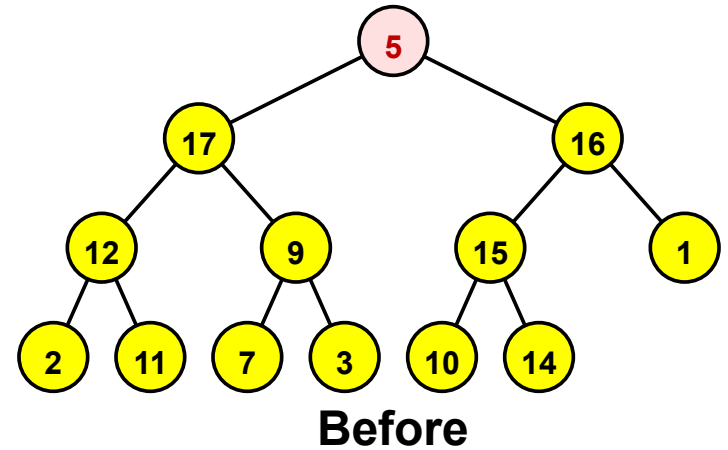


(**Exercise** How to compute parent/children when nodes are numbered 0,1,2,...?)

Max-Heapify

MAX-HEAPIFY(A, i)

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

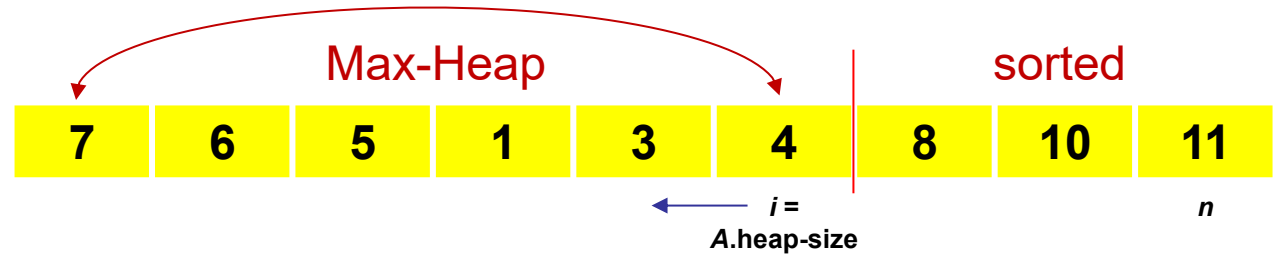


Time $O(\log n)$

Heap-Sort

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

```
1   $A.heap-size = n$ 
2  for  $i = \lfloor n/2 \rfloor$  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
 $O(n \cdot \log n)$

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

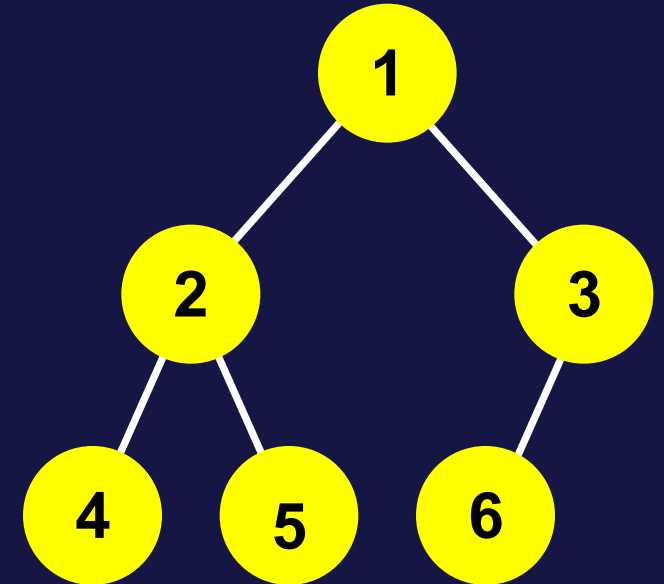
c)

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

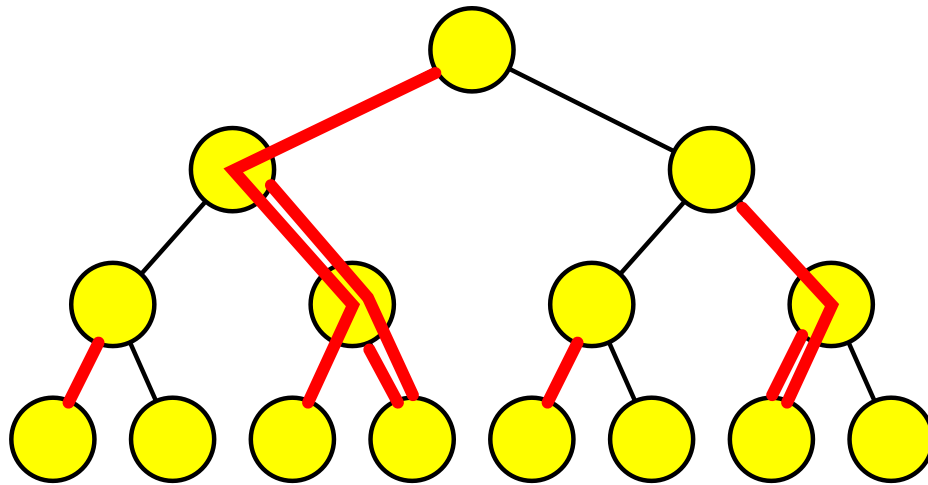
d)

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

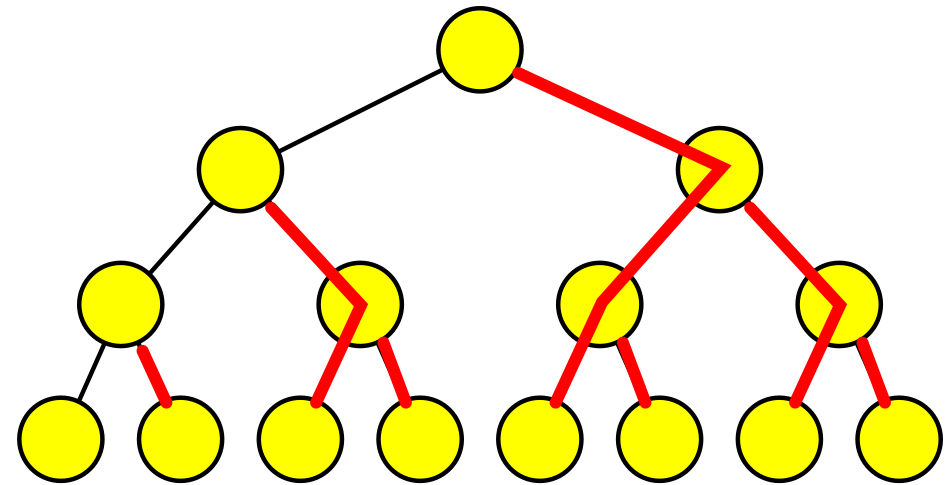
e) Don't know



Build-Max-Heap



Max-Heapify paths (example)



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

**Time for Build-Max-Heap
= $\sum$ time for Max-Heapify
= **# red edges****

$\leq$ **# red edges**
 $= n - \text{depth}$
 $= O(n)$

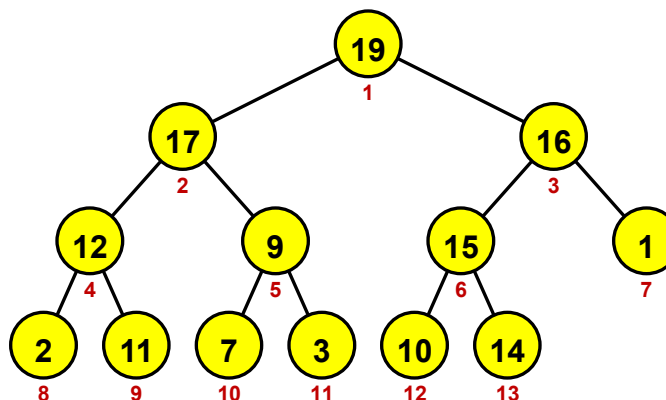
Time $O(n)$

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

Sorting algorithms

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

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)

```

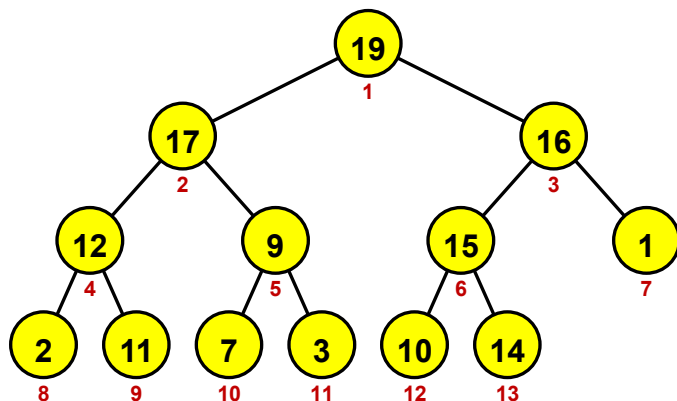
1  $A.heap-size = A.heap-size + 1$ 
2  $A[A.heap-size] = -\infty$ 
3 HEAP-INCREASE-KEY( $A, A.heap-size, key$ )
  
```

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    $i = PARENT(i)$ 
  
```

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   $max = \text{MAX-HEAP-MAXIMUM}(A)$ 
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   $x.key = -\infty$ 
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  $A[\text{PARENT}(i)].key < A[i].key$ 
6      exchange  $A[i]$  with  $A[\text{PARENT}(i)]$ , updating the information that maps
       priority queue objects to array indices
7   $i = \text{PARENT}(i)$ 

```

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
 $\frac{1}{2}$

Experiment

Throw a coin

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



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



Quicksort

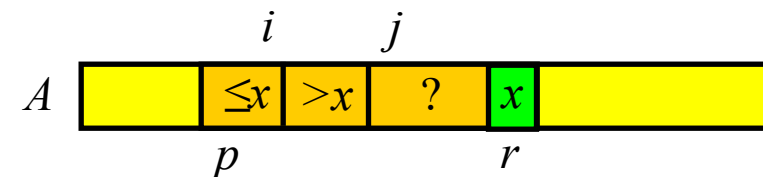
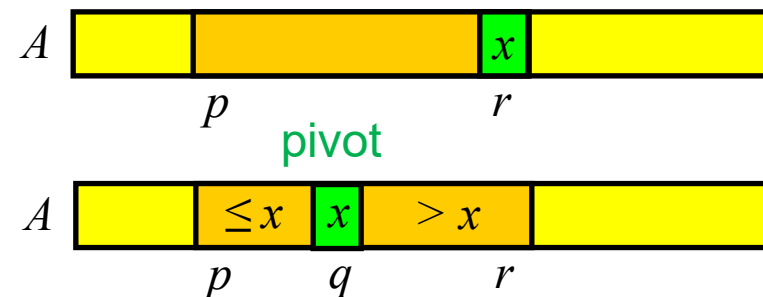
Sort $A[p..r]$

QUICKSORT(A, p, r)

```
1  if  $p < r$ 
2      // Partition the subarray around the pivot, which ends up in  $A[q]$ 
3       $q = \text{PARTITION}(A, p, r)$ 
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  $A[j] \leq x$  // 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 $O(n^2)$

Quicksort on 23 elements

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
18	10	5	4	20	11	22	15	3	17	14	16	19	8	6	21	7	13	9	12	23	2	1

1	10	5	4	20	11	22	15	3	17	14	16	19	8	6	21	7	13	9	12	23	2	18
---	----	---	---	----	----	----	----	---	----	----	----	----	---	---	----	---	----	---	----	----	---	----

10	5	4	11	15	3	17	14	16	8	6	7	13	9	12	2	18	20	21	23	19	22
----	---	---	----	----	---	----	----	----	---	---	---	----	---	----	---	----	----	----	----	----	----

2	5	4	11	15	3	17	14	16	8	6	7	13	9	12	10	20	21	19	22	23
---	---	---	----	----	---	----	----	----	---	---	---	----	---	----	----	----	----	----	----	----

5	4	3	8	6	7	9	10	15	11	17	13	14	12	16	19	21	20	23
---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----	----	----	----

5	4	3	8	6	7	9	15	11	13	14	12	16	17	20	21
---	---	---	---	---	---	---	----	----	----	----	----	----	----	----	----

5	4	3	6	7	8	11	12	13	14	15	17	21
---	---	---	---	---	---	----	----	----	----	----	----	----

5	4	3	6	8	11	13	14	15
---	---	---	---	---	----	----	----	----

3	4	5	13	14
---	---	---	----	----

4	5	13
---	---	----

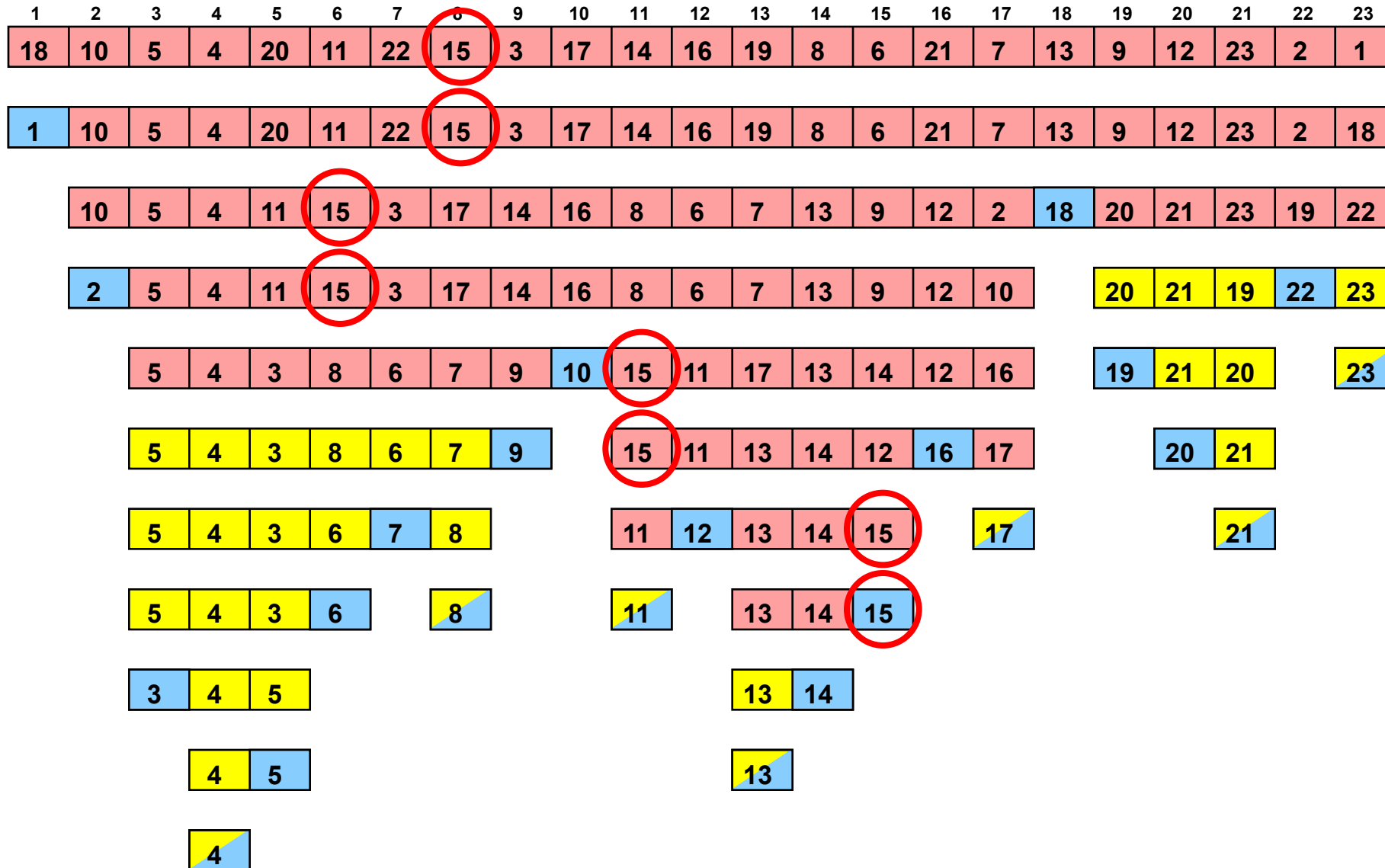
4

Number of calls to Partition ?

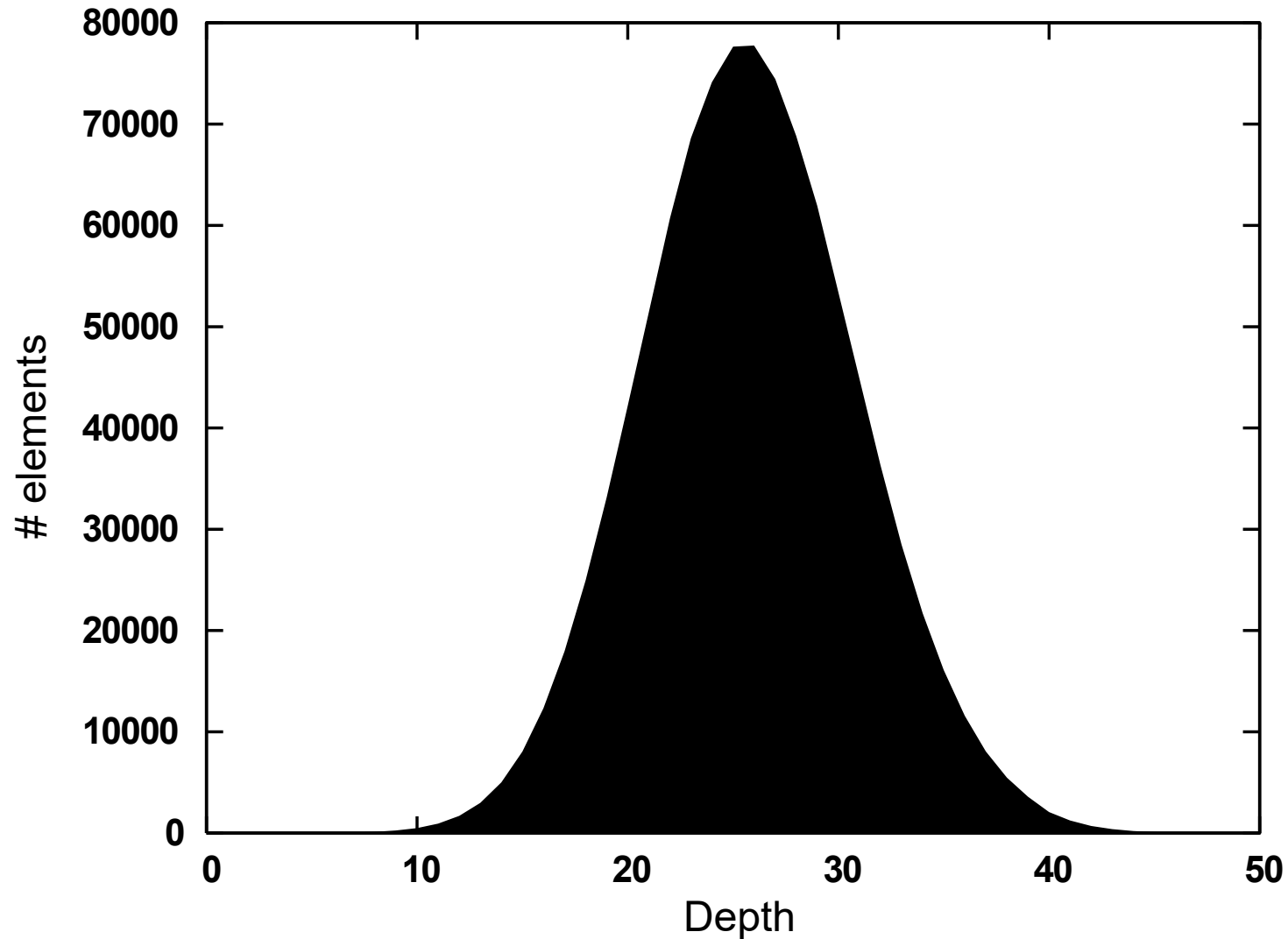
```
QUICKSORT( $A, p, r$ )  
1  if  $p < r$   
2      // Partition the subarray around the pivot, which ends up in  $A[q]$   
3       $q = \text{PARTITION}(A, p, r)$   
4      QUICKSORT( $A, p, q - 1$ ) // recursively sort the low side  
5      QUICKSORT( $A, q + 1, r$ ) // recursively sort the high 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  $i = \text{RANDOM}(p, r)$ 
2 exchange  $A[r]$  with  $A[i]$ 
3 return PARTITION( $A, p, r$ )
```

RANDOMIZED-QUICKSORT(A, p, r)

```
1 if  $p < r$ 
2      $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
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

Randomized-Quicksort – analysis



- An array is in level j if and only if it has $n(3/4)^{j+1} < \text{length} \leq n(3/4)^j$
- A partition is **good** if both subproblems $\leq 3/4$ 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 $x_i \leq 2 \cdot \log_{4/3} n$**

Randomized-Quicksort – analysis

Expected time for randomized quicksort

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

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

$$= O(n \cdot \log n)$$



Sorting algorithms

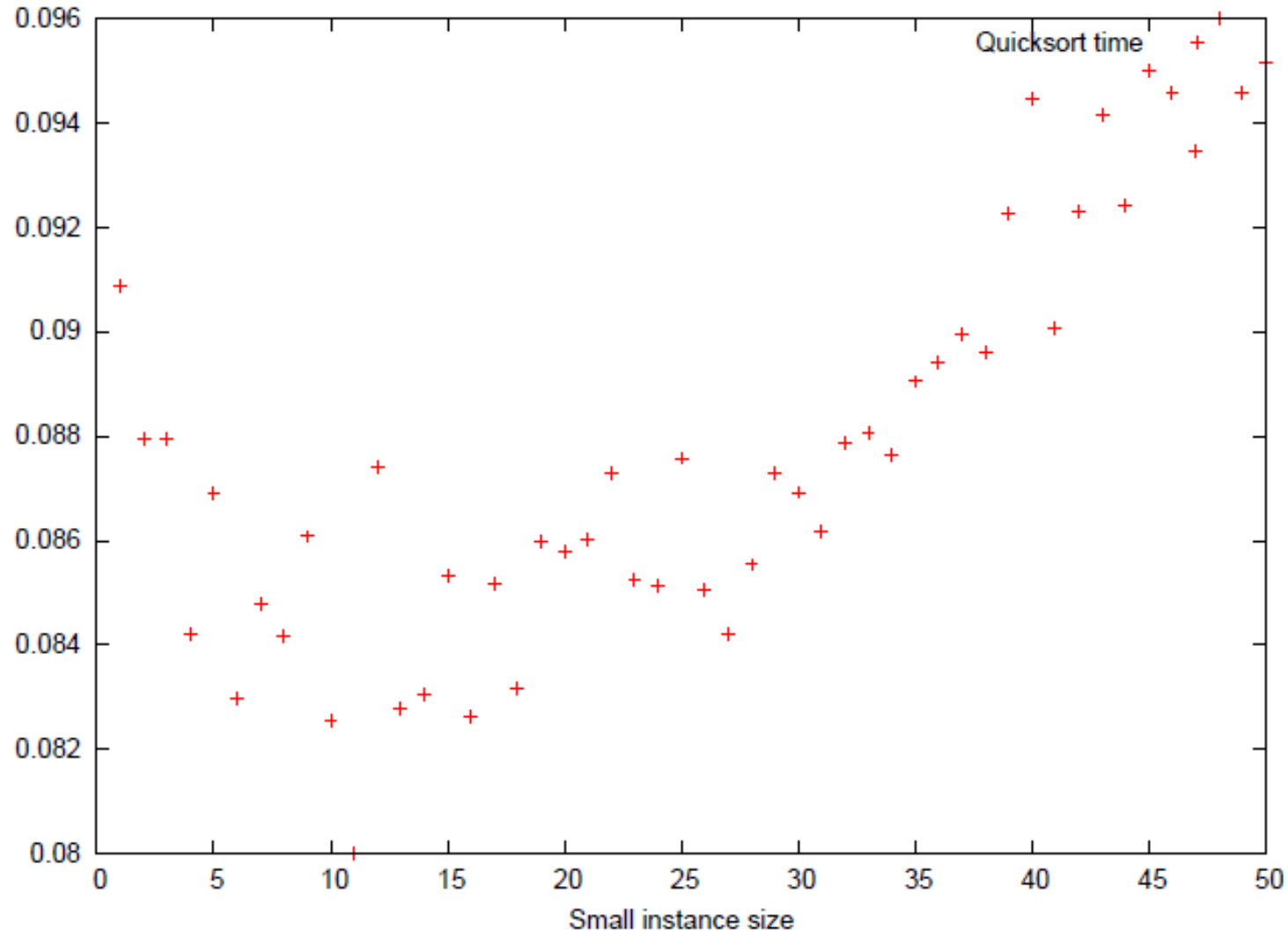
Algorithm	Worst-Case Time
Heap-Sort Merge-Sort	$O(n \cdot \log n)$
Insertion-Sort Selection-Sort Quicksort (Deterministic and randomized)	$O(n^2)$

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

Sorting

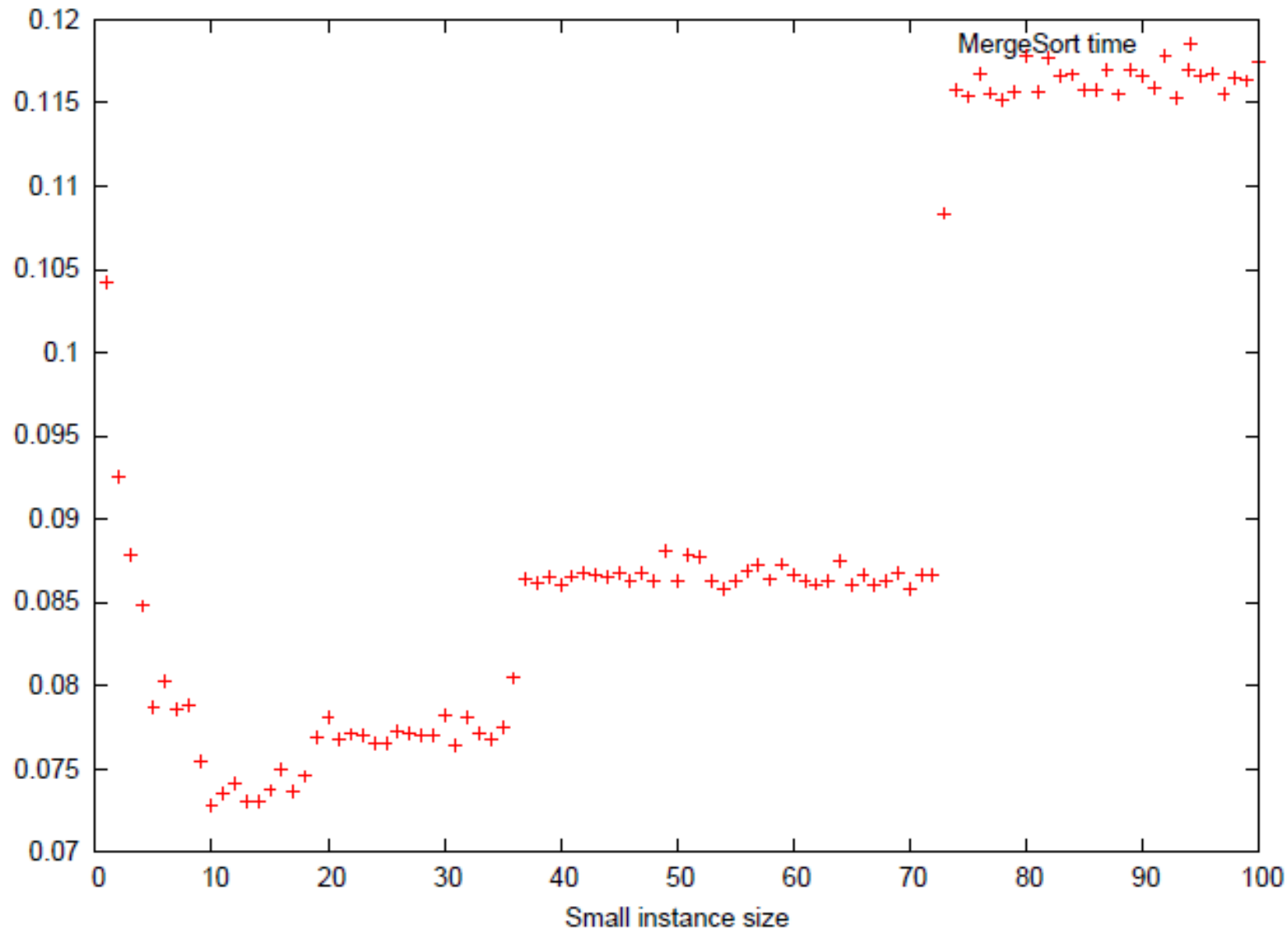
- some experimental results**

Quicksort + Insertion-sort



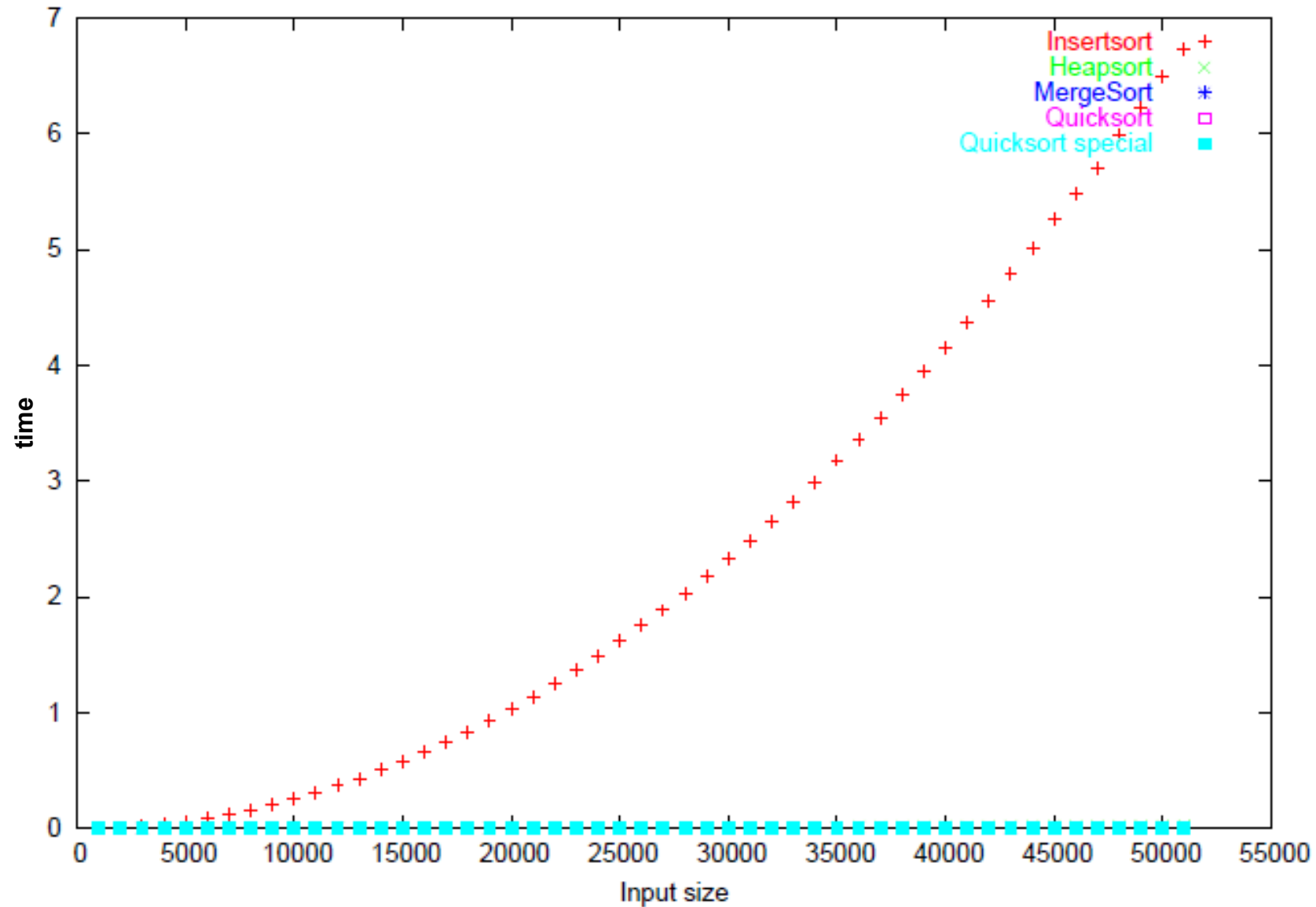
Use **insertion-sort** for small subproblems $n = 300.000$ random elements

Merge-sort + Insertion-sort

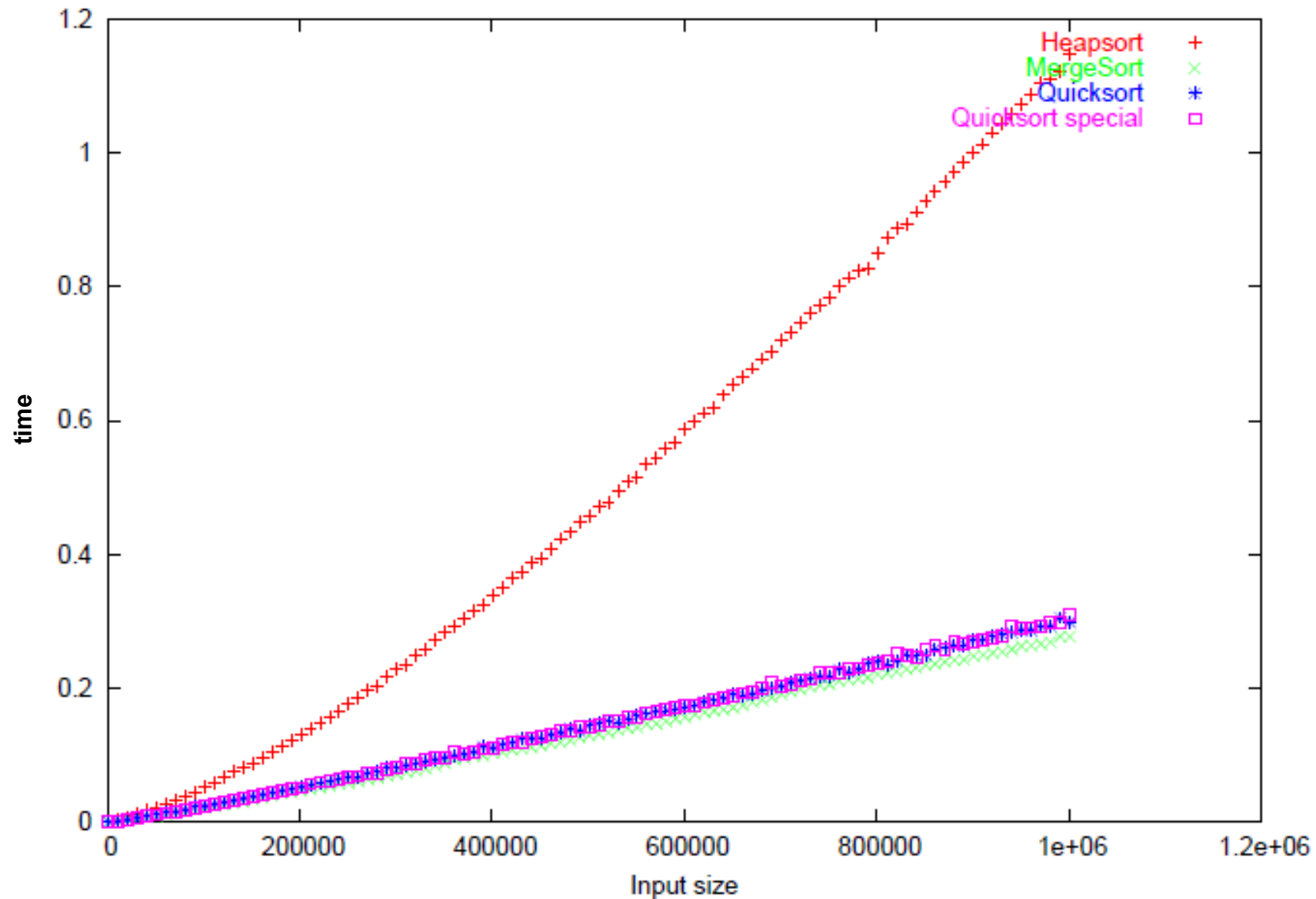


Use **insertion-sort** for small subproblems $n = 300.000$ random elements

Sorting algorithms



Sorting algorithms



Sorting in practice

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

- Inplace (requires no additional arrays)
- Fast

...with some twists

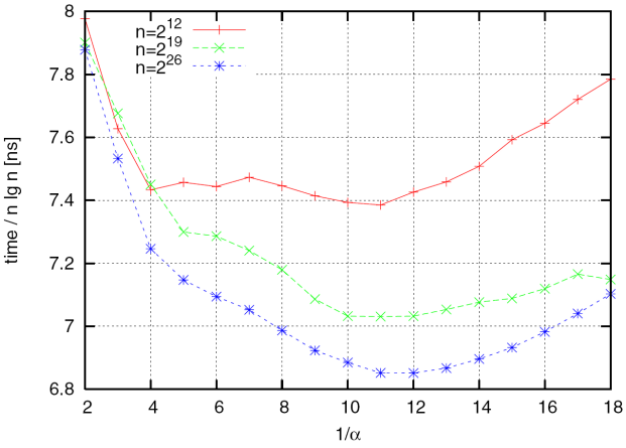
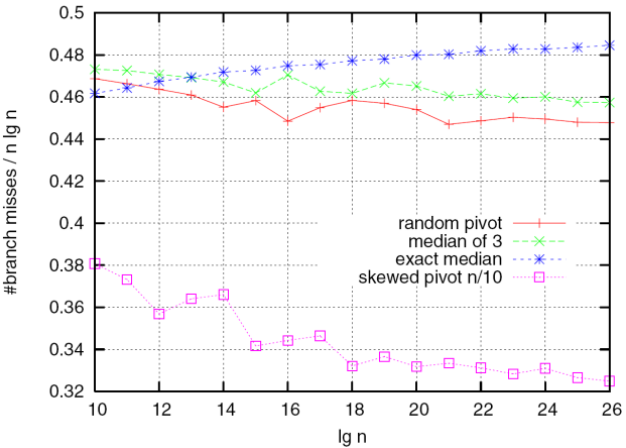
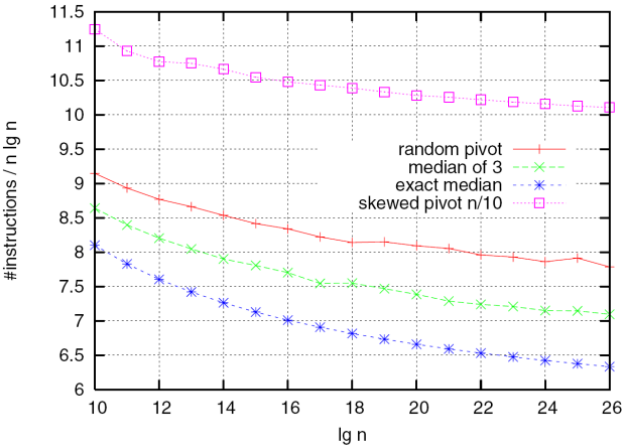
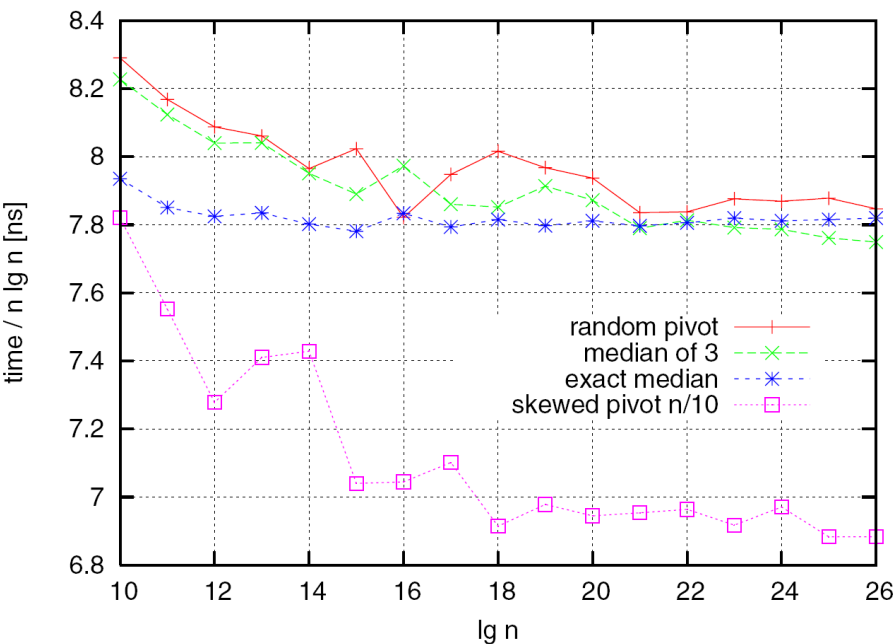
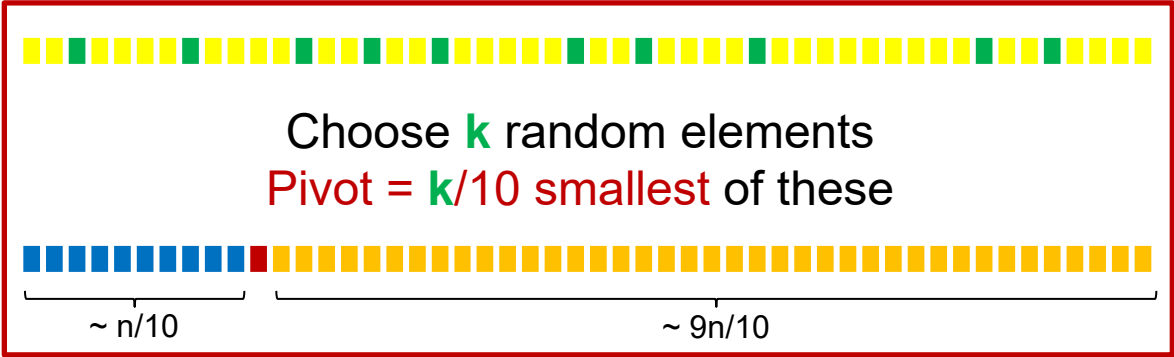
- use **Insertion-Sort** for small subproblems
- use **Merge-Sort** if QuickSort takes too long time
- Select pivot as the median of $O(1)$ (random) elements

How Branch Mispredictions Affect Quicksort

Kanela Kaligosi and Peter Sanders

14th Annual European Symposium on Algorithms (ESA 2006)

doi: [10.1007/11841036_69](https://doi.org/10.1007/11841036_69)

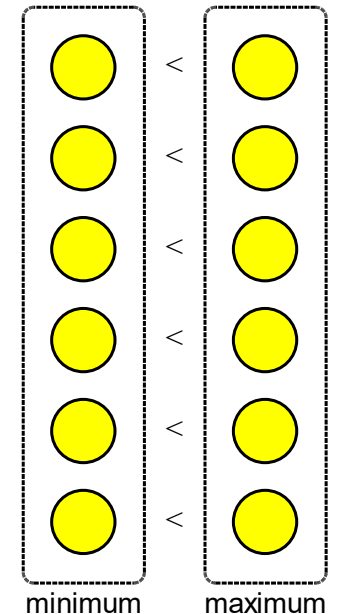


Algorithms and Data Structures

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 $3/2 \cdot n - 2$ comparisons

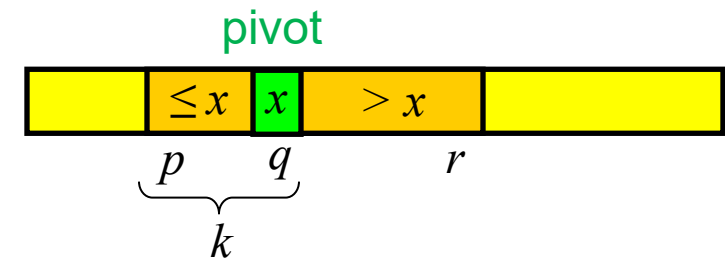


Randomized-Select

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

RANDOMIZED-SELECT(A, p, r, i)

```
1  if  $p == r$ 
2      return  $A[p]$       //  $1 \leq i \leq r - p + 1$  when  $p == r$  means that  $i = 1$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
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



1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
18	10	5	4	20	11	22	15	3	17	14	16	19	8	6	21	7	13	9	12	23	2	1
1	10	5	4	20	11	22	15	3	17	14	16	19	8	6	21	7	13	9	12	23	2	18
10	5	4	11	15	3	17	14	16	8	6	7	13	9	12	2	18	20	21	23	19	22	
2	5	4	11	15	3	17	14	16	8	6	7	13	9	12	10							
5	4	3	8	6	7	9	10	15	11	17	13	14	12	16								
15	11	13	14	12	16	17																
11	12	13	14	15																		
13	14	15																				

Worst-case time for Randomized-Select ?

```
RANDOMIZED-SELECT( $A, p, r, i$ )
1  if  $p == r$ 
2      return  $A[p]$            //  $1 \leq i \leq r - p + 1$  when  $p == r$  means that  $i = 1$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
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



Expected time for randomized select

$$= O(\sum_j \text{expected time for level } j)$$

$$= O(\sum_j n \cdot (3/4)^j \cdot \text{expected \# arrays in level } j)$$

$$= O(\sum_j n \cdot (3/4)^j \cdot 2) \quad \leftarrow \sum_{i=0}^{\infty} c^i = \frac{1}{1-c} \text{ for } 0 < c < 1$$

$$= \mathbf{O(n)}$$



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

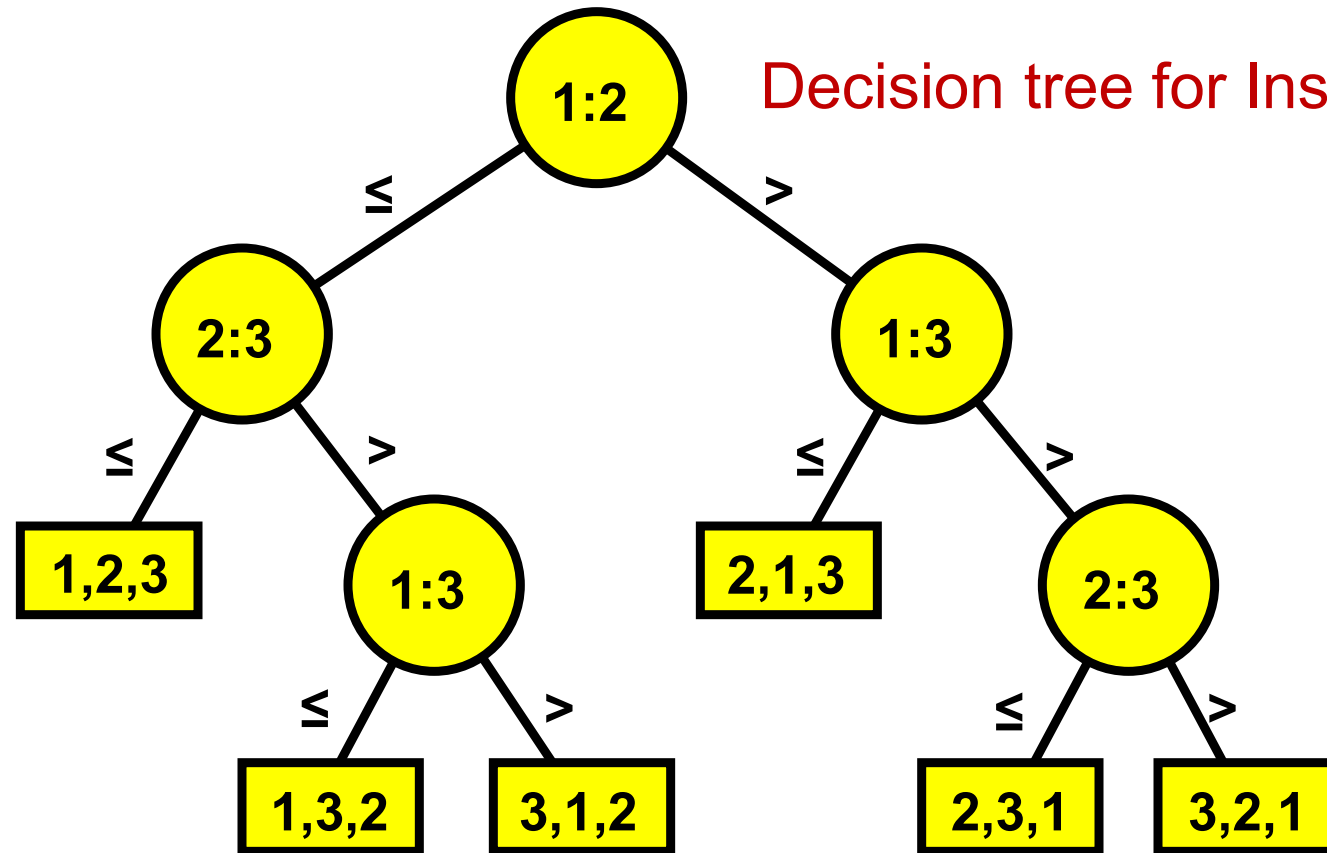
Comparisons for Insertion-Sort

1	2	3
x_1	x_2	x_3

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



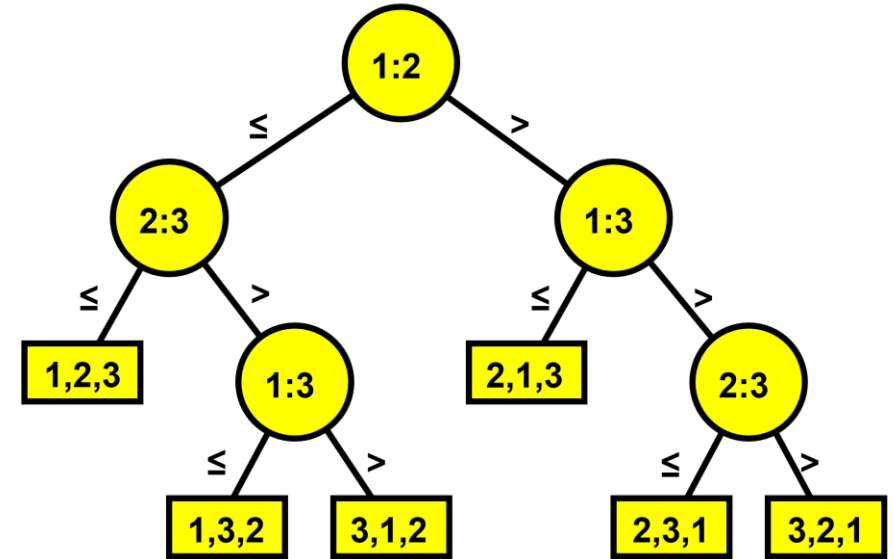
Internal node $i : j$ comparison of x_i and x_j
Leaves describe output **permutation**

Comparison-based sorting: Lower bound

$n!$ different outputs $\leq$ leaves

tree of height h has $\leq 2^h$ leaves

$$n! \leq 2^h$$



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

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

for $n \geq 4$

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
$\geq$ 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

n		2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	...	28	...	191		
Upper bounds	MergeSort / Binary search	1	3	5	8	11	14	17	21	25	29	33	37	41	45	49	54	59	64	69	74	79	84	89	94	...	109	...	1273		
	Ford-Johnson 1959	1	3	5	7	10	13	16	19	22	26	30	34	38	42	46	50	54	58	62	66	71	76	81	86	...	101	...	1192		
	Manacher 1979	optimal																													
	"The Ford-Johnson Sorting Algorithm Is Not Optimal"	1191																													
Lower bounds	Exhaustiv search											30	34	38	42	46	50	54	58						71						99
	$\lceil \log_2(n!) \rceil$	1	3	5	7	10	13	16	19	22	26	29	33	37	41	45	49	53	57	62	66	70	75	80	84	...	98	...	1177		
												Wells 1966	Kasai et al. 1994	Peczarski 2004	Peczarski 2006	Weiß, Stober 2023	Weiß, Stober 2023	Weiß, Stober 2023	Cheng et al. 2007						Peczarski 2004						Weiß, Stober 2023

Sorting integers

...exploit that integers are binary numbers

Counting-Sort:

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

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 not stable ?

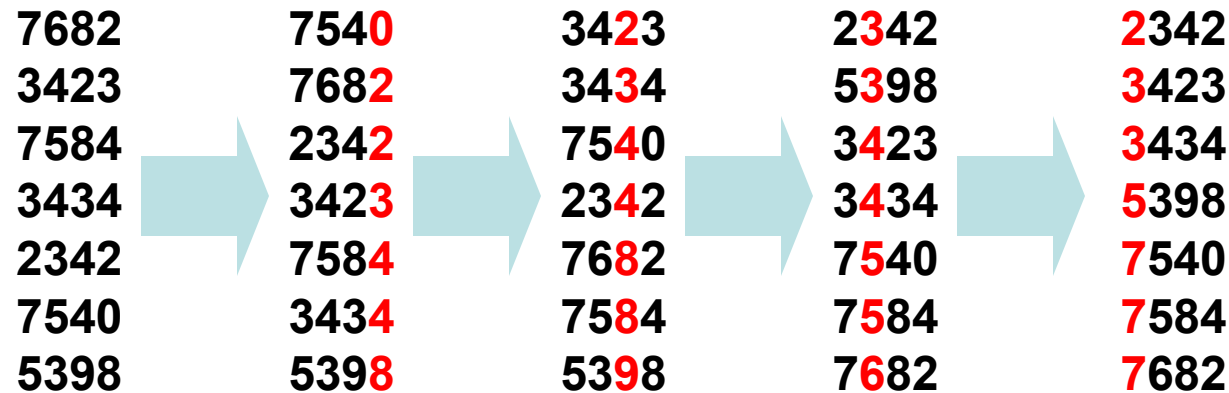
- a) Insertion-Sort
- b) Heap-Sort
- c) Quicksort
- d) Merge-Sort
- e) Heap-Sort and Quicksort
- f) Insertion-Sort and Heap-Sort
- g) Merge-Sort and Quicksort
- h) Insertion-Sort and Quicksort
- i) Don't know

Radix-Sort:

Input: array A , integers with d digits from $\{0, 1, \dots, k\}$

$\text{RADIX-SORT}(A, n, d)$

```
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

Gerth Stølting Brodal
Aarhus University
Dept. of Computer Science
Åbogade 34, DK-8200 Aarhus N

Radix-Sort at work at



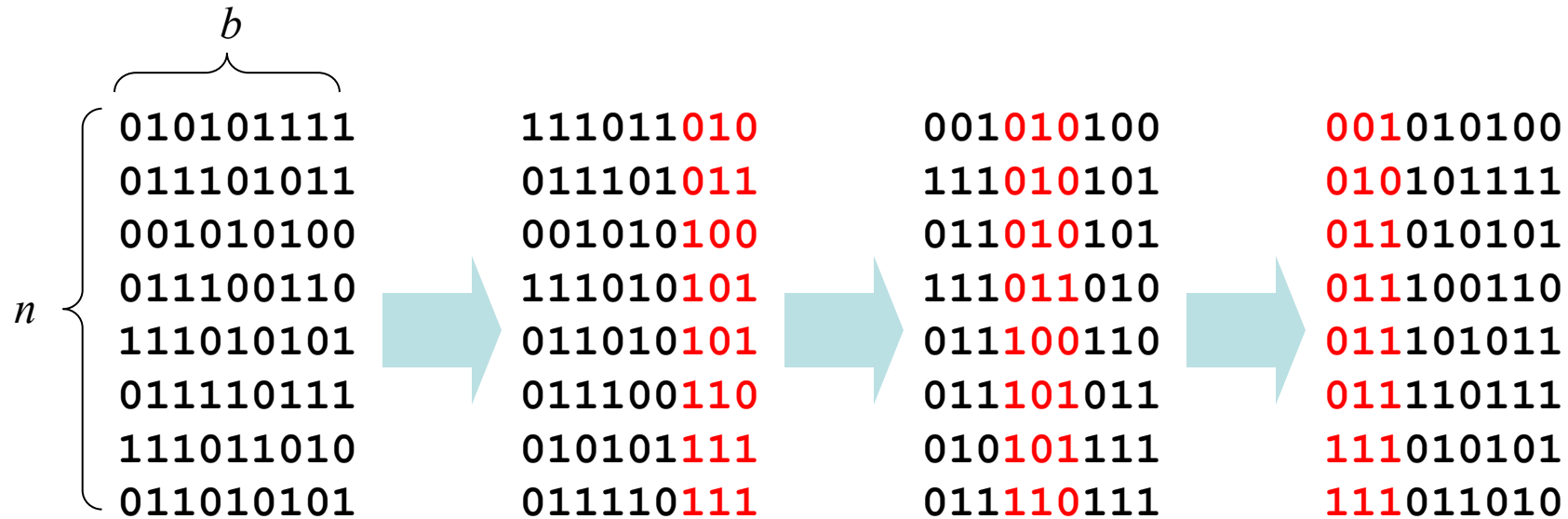
Step 1: Sort by number



Step 2: Sort by route

Radix-Sort:

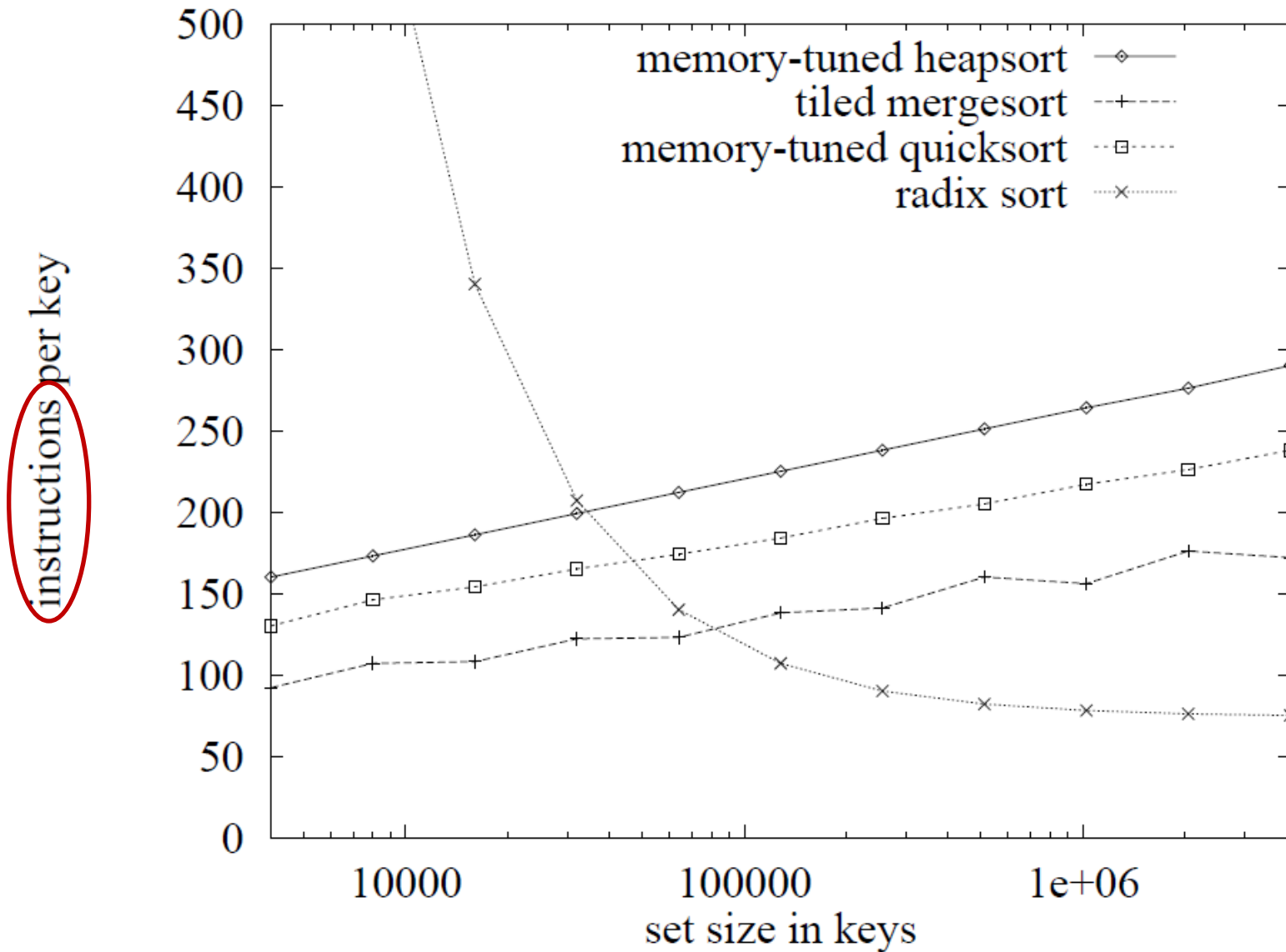
Input: array A , n integers with b bits



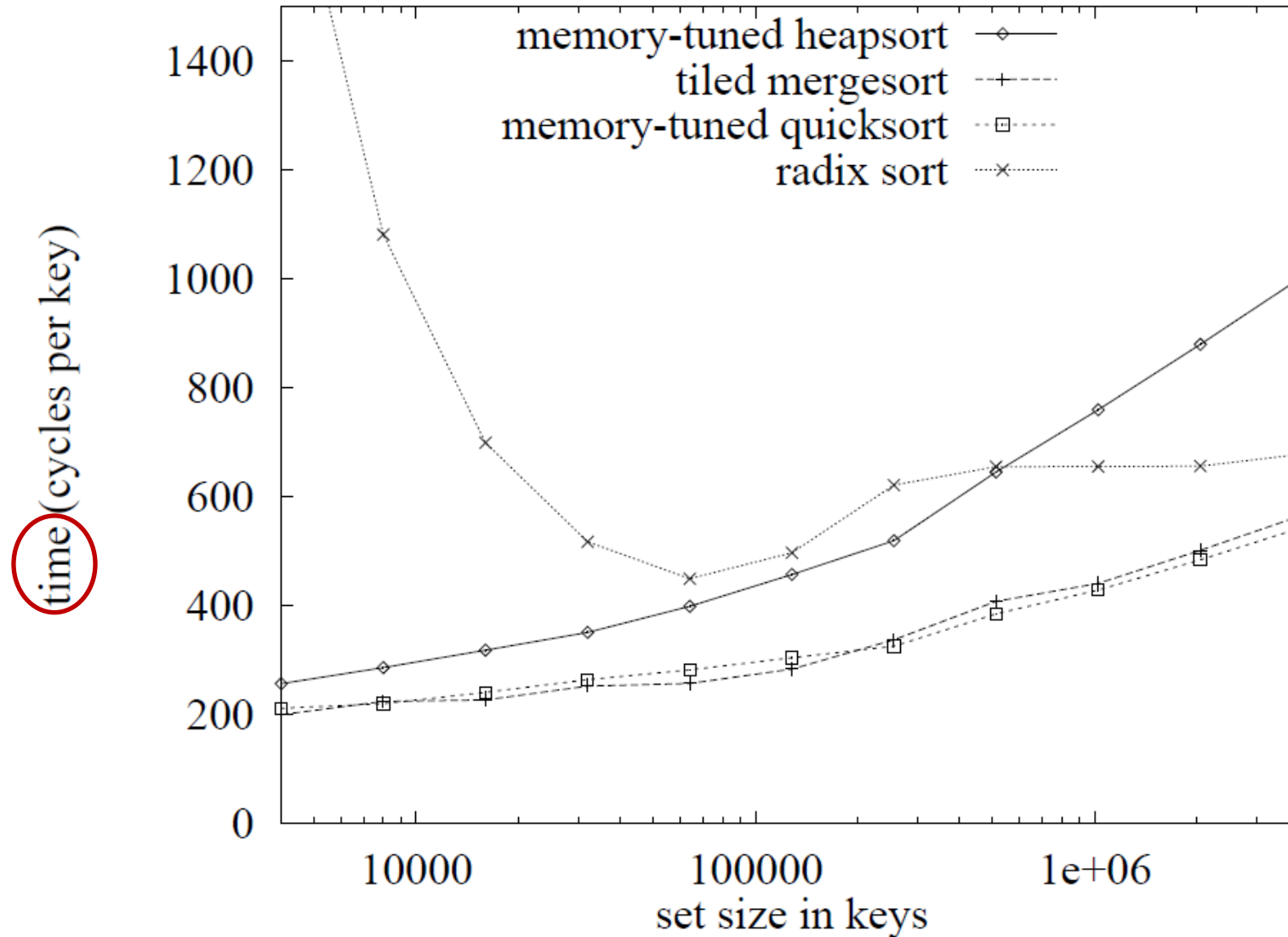
$n = 8$, $k = 7$ (3 bits = $\log n$ bits), $d = 3$ (3 x 3 bits)

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

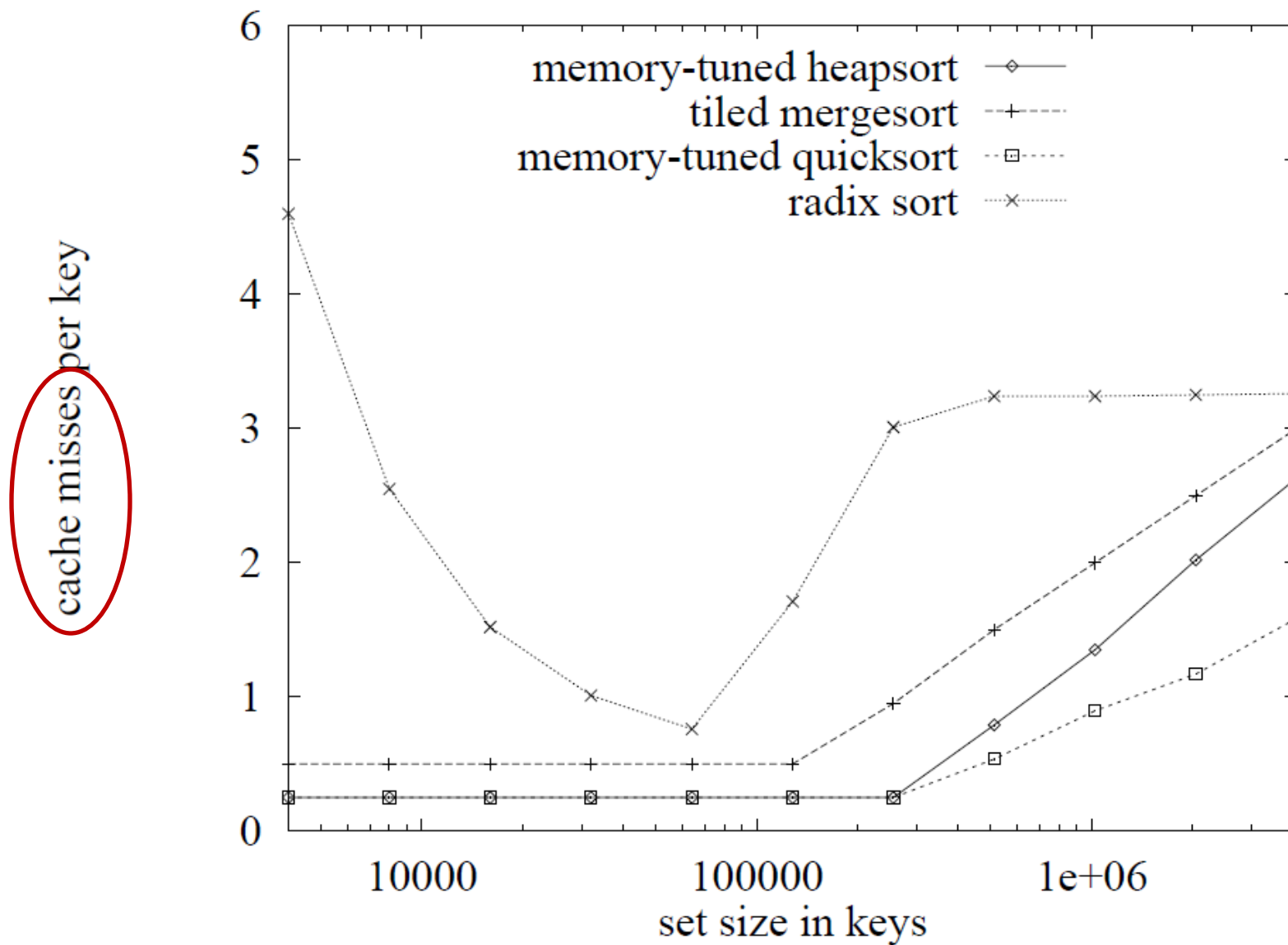
Radix-Sort: Experiments



Radix-Sort: Experiments



Radix-Sort: Experiments

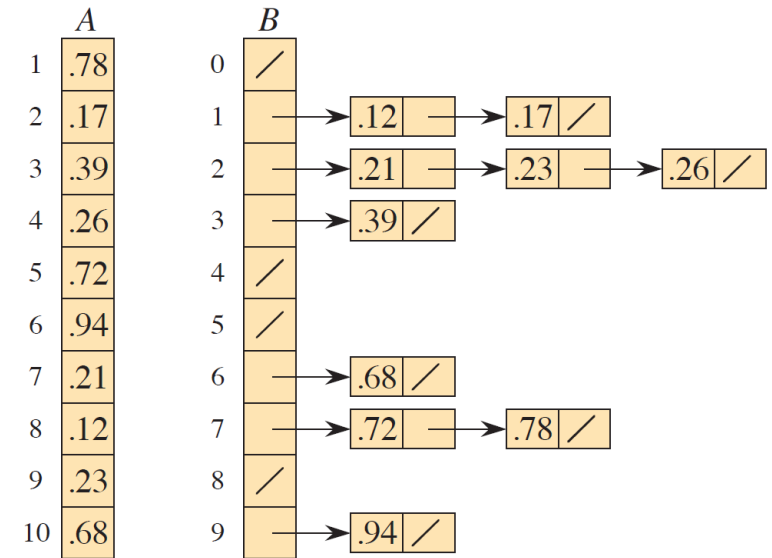


Bucket-Sort:

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  $B[\lfloor n \cdot A[i] \rfloor]$ 
6  for  $i = 0$  to  $n-1$ 
7      sort list  $B[i]$  with insertion sort
8  concatenate the lists  $B[0], B[1], \dots, B[n-1]$  together in order
9  return the concatenated lists
```



Expected time $O(n)$ – for random inputs

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.

$O(n \log \log n)$ is known. Is it possible to get expected $O(n)$ time?

With randomization and assuming word-size $\Omega(\log^2 n \cdot \log \log n)$ 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](https://doi.org/10.1109/SFCS.2002.1181890)

Andersson, Hagerup, Nilsson, Raman, *Sorting in linear time?*, STOC 1995, doi: [10.1145/225058.225173](https://doi.org/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](https://doi.org/10.1007/978-3-319-08404-6_3)

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 $2 \cdot \log n$) otherwise change to Heap-Sort Insertion-Sort ≤ 16 elements	Quicksort Not recursive
Source code	Install JDK from www.oracle.com/java/technologies/downloads/ File 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

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

Algorithms and Data Structures

Stacks, queues
[CLRS, Chapter 10]

[CLRS, Part 3] : Data structures

Maintain a structure for a
dynamic set of data

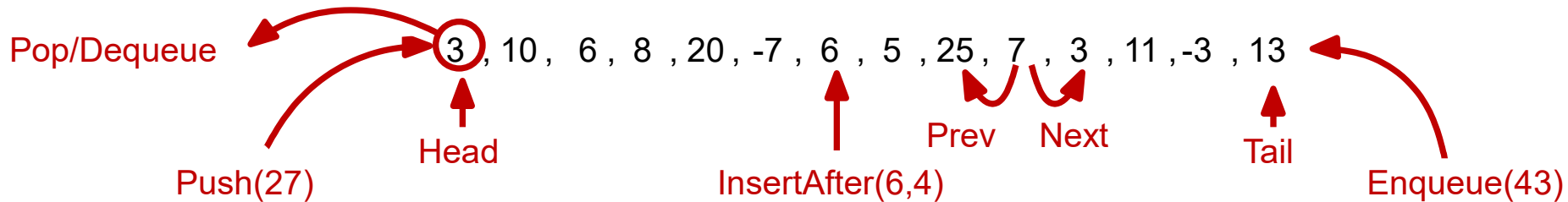
Abstract data structures for sets

Operation		Return value			
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			

- Min-priority-queue
- Max-priority-queue
- Dictionary

Abstract data structures for lists

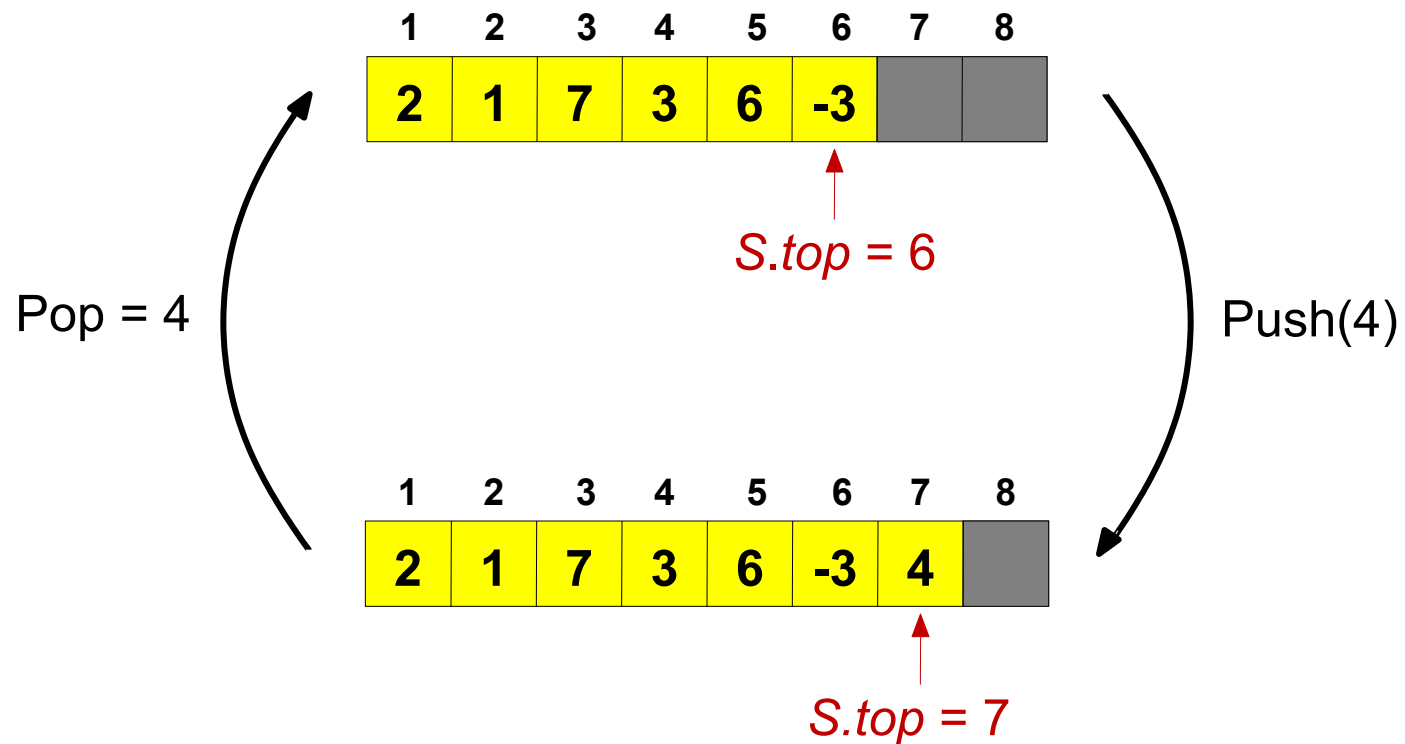
Operation		Return value	-Stack	-Queue
Queries	Empty(S)	True or False	●	●
	Head(S), Tail(S)	pointer to element		
	Next(S, x), Prev(S, x)	pointer to element		
	Search(S, x)	pointer to element		
Updates	Push(S, x)	-	●	
	Pop/Dequeue(S)	element	●	●
	Enqueue(S, x)	-		●
	Delete(S, x)	element		
	InsertAfter(S,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

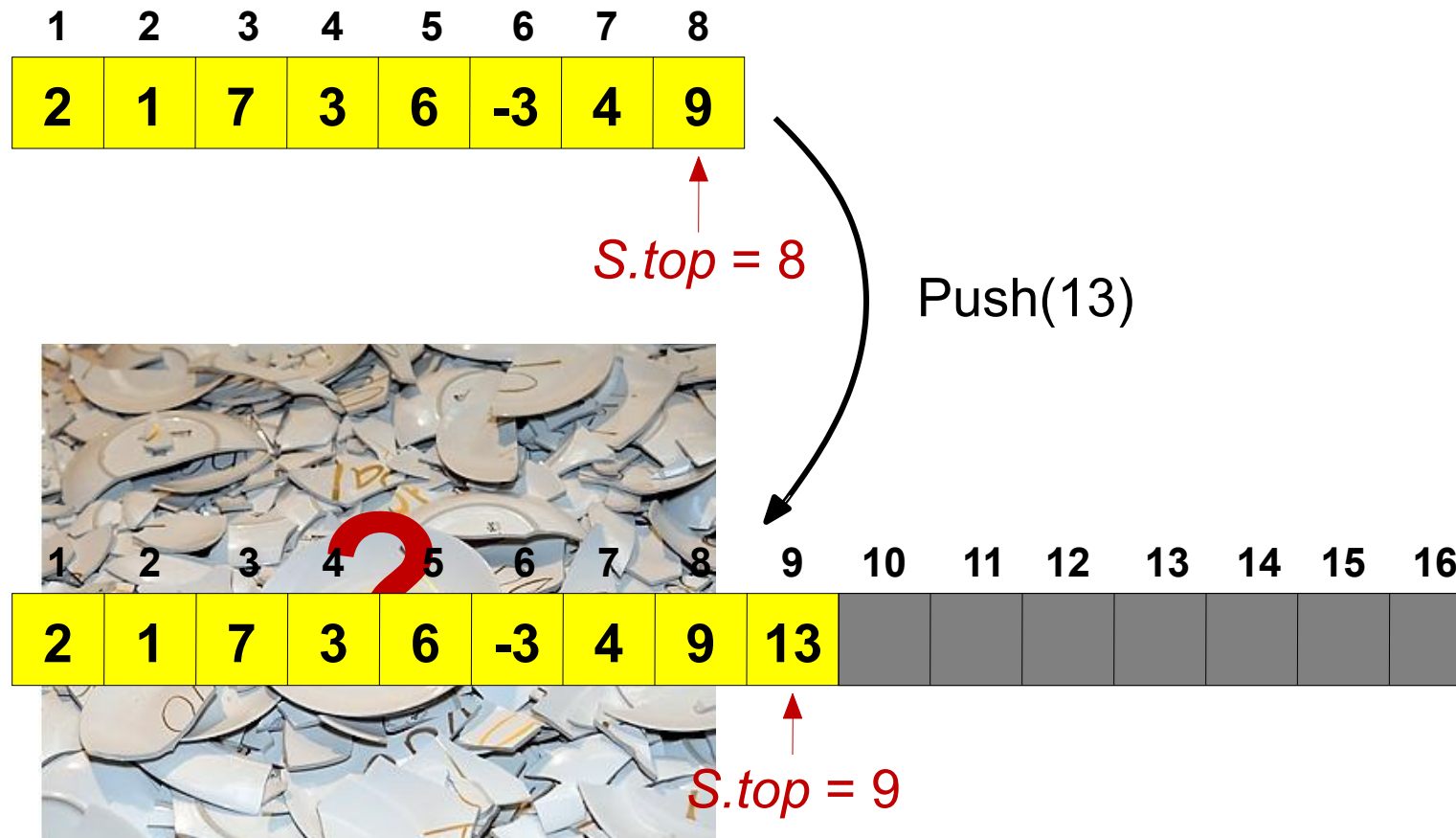
```
1  if  $S.top == S.size$ 
2      error "overflow"
3  else  $S.top = S.top + 1$ 
4       $S[S.top] = x$ 
```

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

1

2

4

8

16

32

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

Halve the array when it is $<1/4$ full

32

16

16

8

8

16

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

Array doubling + halving

– a general technique

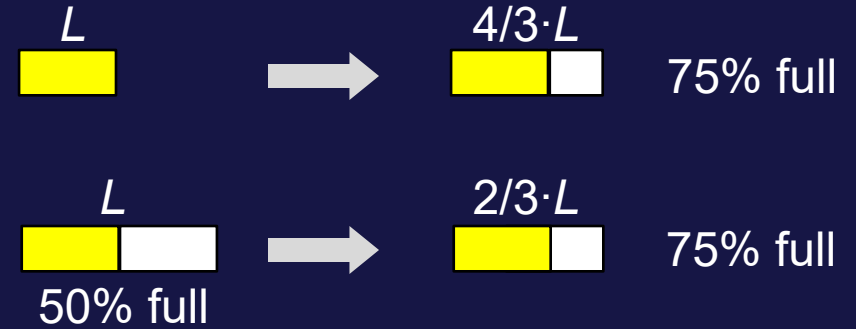
Time for n extensions/reductions is $O(n)$

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) $3/2 \cdot n$
- b) $4/3 \cdot n$
- c) $2 \cdot n$
- d) $8/3 \cdot n$
- e) Don't know

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 <code>trimToSize</code>)	< 50 %	No (can call <code>shrink_to_fit</code>)
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;

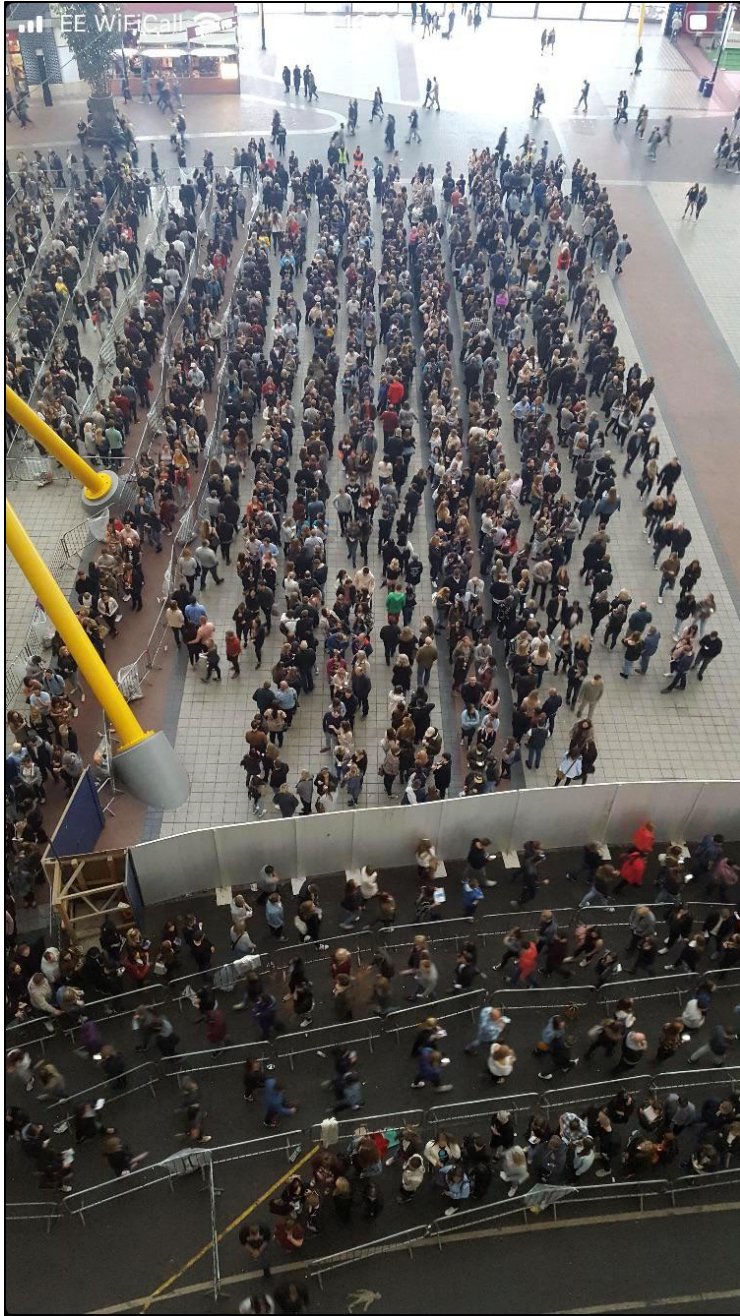
    if (allocated >= newsize && newsize >= (allocated >> 1)) {
        assert(self->ob_item != NULL || newsize == 0);
        Py_SET_SIZE(self, newsize);
        return 0;
    }

    new_allocated = ((size_t)newsize + (newsize >> 3) + 6) & ~(size_t)3;
    ...
}
```

C++ vector

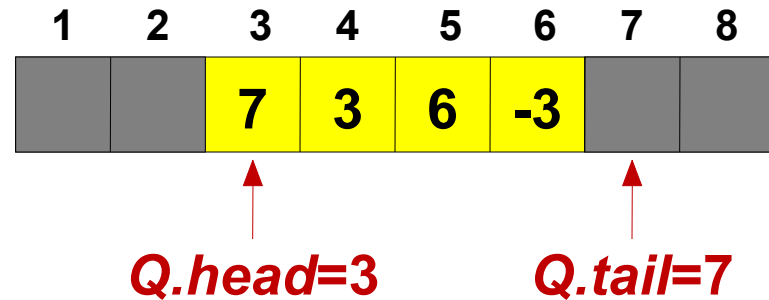
```
size_type
_M_check_len(size_type __n, const char* __s) const
{
    if (max_size() - size() < __n)
        __throw_length_error(__N(__s));

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

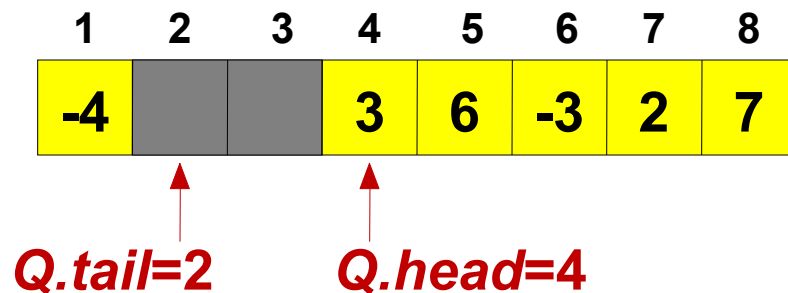


Queues

Queue : array Implementation



Enqueue(2)
Enqueue(7)
Enqueue(-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

1	2	3	4	5	6	7	8
-4	5		3	6	-3	2	7

$Q.tail=3$

$Q.head=4$

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

Overflow : array doubling/halving

Enqueue(9)

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

$Q.head=1$

$Q.tail=9$

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)	$O(1)^*$

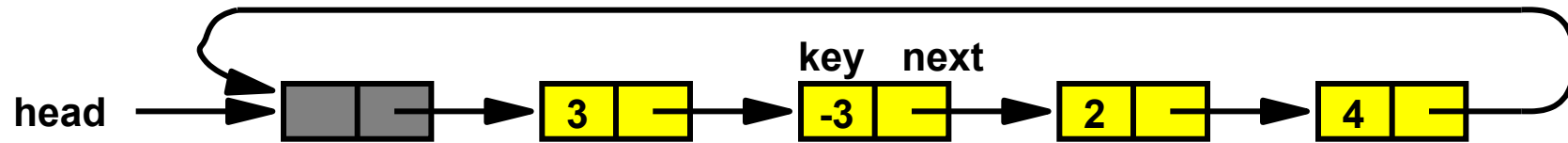
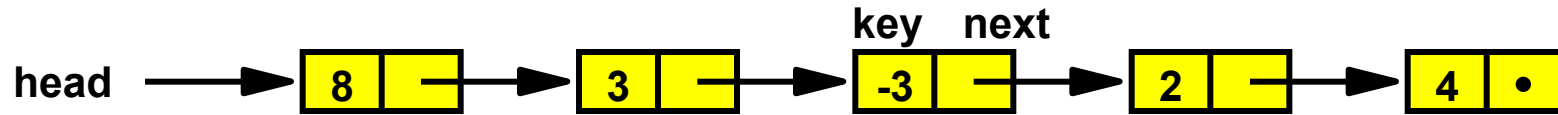
* Worst-case without doubling/halving
Amortized ([CLRS, Chapter 16]) with doubling/halving



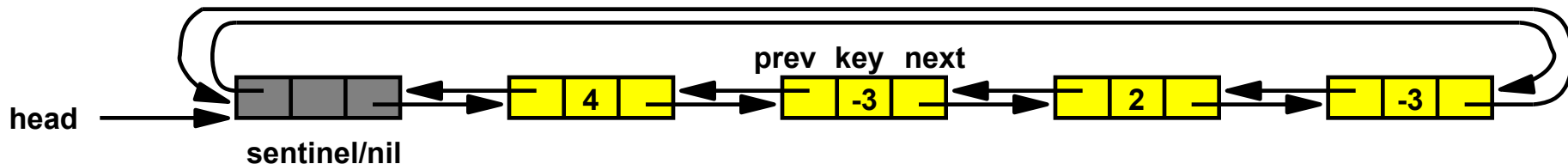
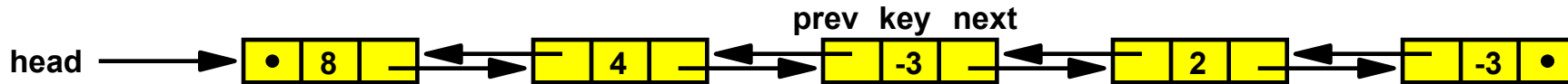
Linked lists

Linked lists

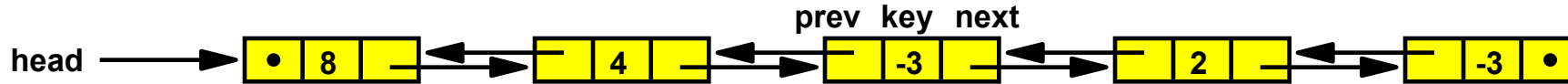
Singly linked (non-circular and circular)



Doubly linked (non-circular and circular)



Doubly linked lists



LIST-SEARCH(L, k)

```
1  $x = L.head$ 
2 while  $x \neq \text{NIL}$  and  $x.key \neq k$ 
3      $x = x.next$ 
4 return  $x$ 
```

LIST-PREPEND(L, x)

```
1  $x.next = L.head$ 
2  $x.prev = \text{NIL}$ 
3 if  $L.head \neq \text{NIL}$ 
4      $L.head.prev = x$ 
5  $L.head = x$ 
```

LIST-INSERT(x, y)

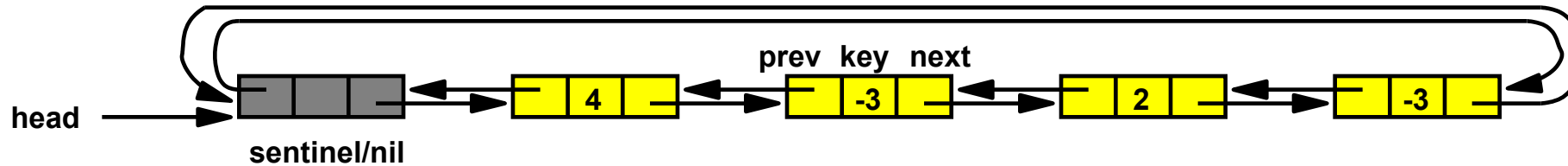
```
1  $x.next = y.next$ 
2  $x.prev = y$ 
3 if  $y.next \neq \text{NIL}$ 
4      $y.next.prev = x$ 
5  $y.next = x$ 
```

LIST-DELETE(L, x)

```
1 if  $x.prev \neq \text{NIL}$ 
2      $x.prev.next = x.next$ 
3 else  $L.head = x.next$ 
4 if  $x.next \neq \text{NIL}$ 
5      $x.next.prev = x.prev$ 
```

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

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  $x.key \neq k$ 
4       $x = x.next$ 
5  if  $x == L.nil$ 
6      return NIL
7  else return  $x$ 

```

[CLRS3]

LIST-SEARCH'(L, k)

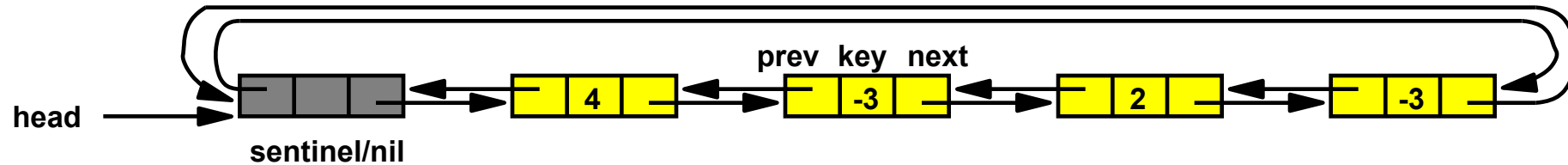
```

1   $x = L.nil.next$ 
2  while  $x \neq L.nil$  and  $x.key \neq k$ 
3       $x = x.next$ 
4  return  $x$ 

```

List-Search'	$O(n)$
List-Insert'	$O(1)$
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

$$L[R[x]] \leftarrow L[x], \quad R[L[x]] \leftarrow R[x] \quad (1)$$

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

$$L[R[x]] \leftarrow x, \quad R[L[x]] \leftarrow x \quad (2)$$

will put x back into the list again.

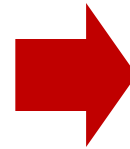


Donald E. Knuth (1938-)

"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 *)

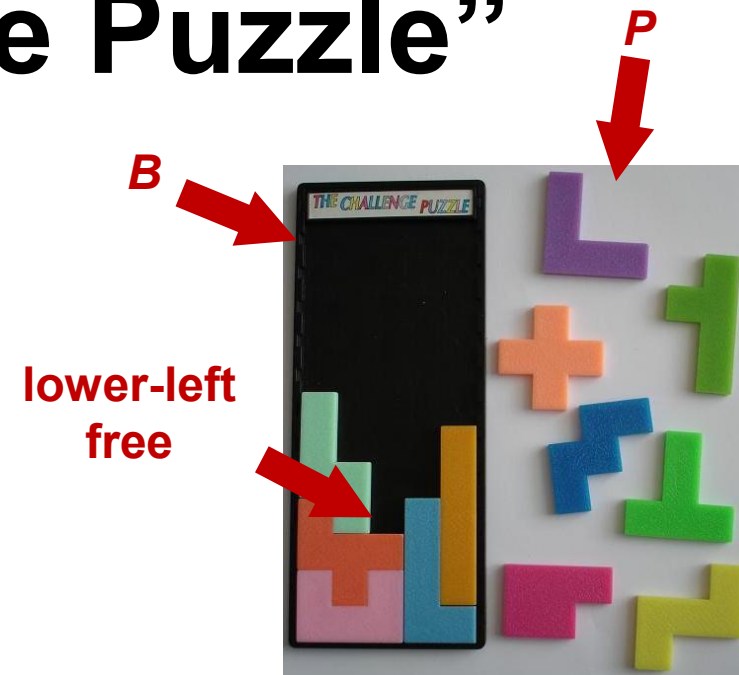
if p can be placed in lower-left free cell **then**

 insert p in B

 Solve(B, P)

 remove p from B

reinsert p in P



Before



After

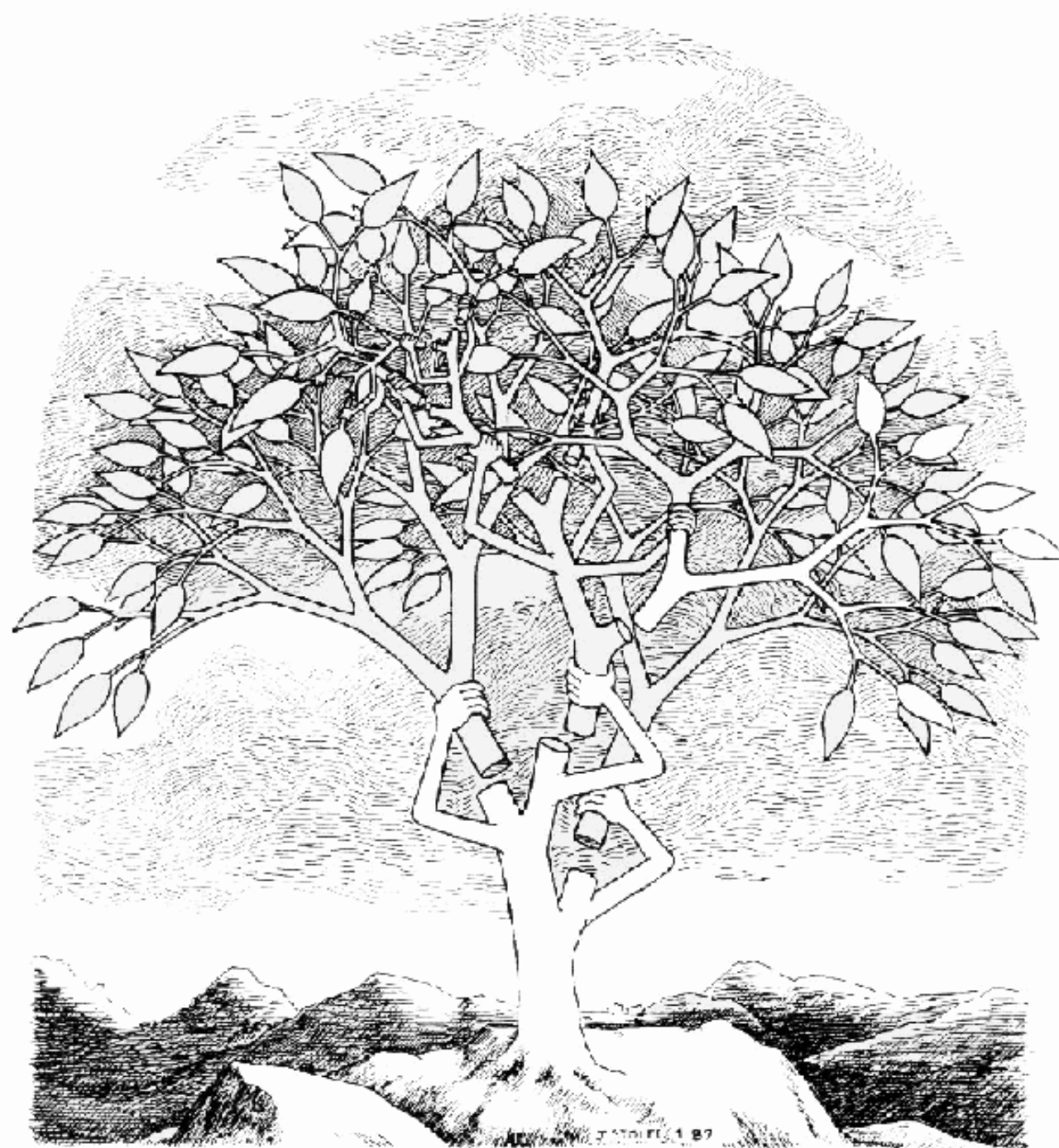
”The Challenge Puzzle”



4.040 solutions

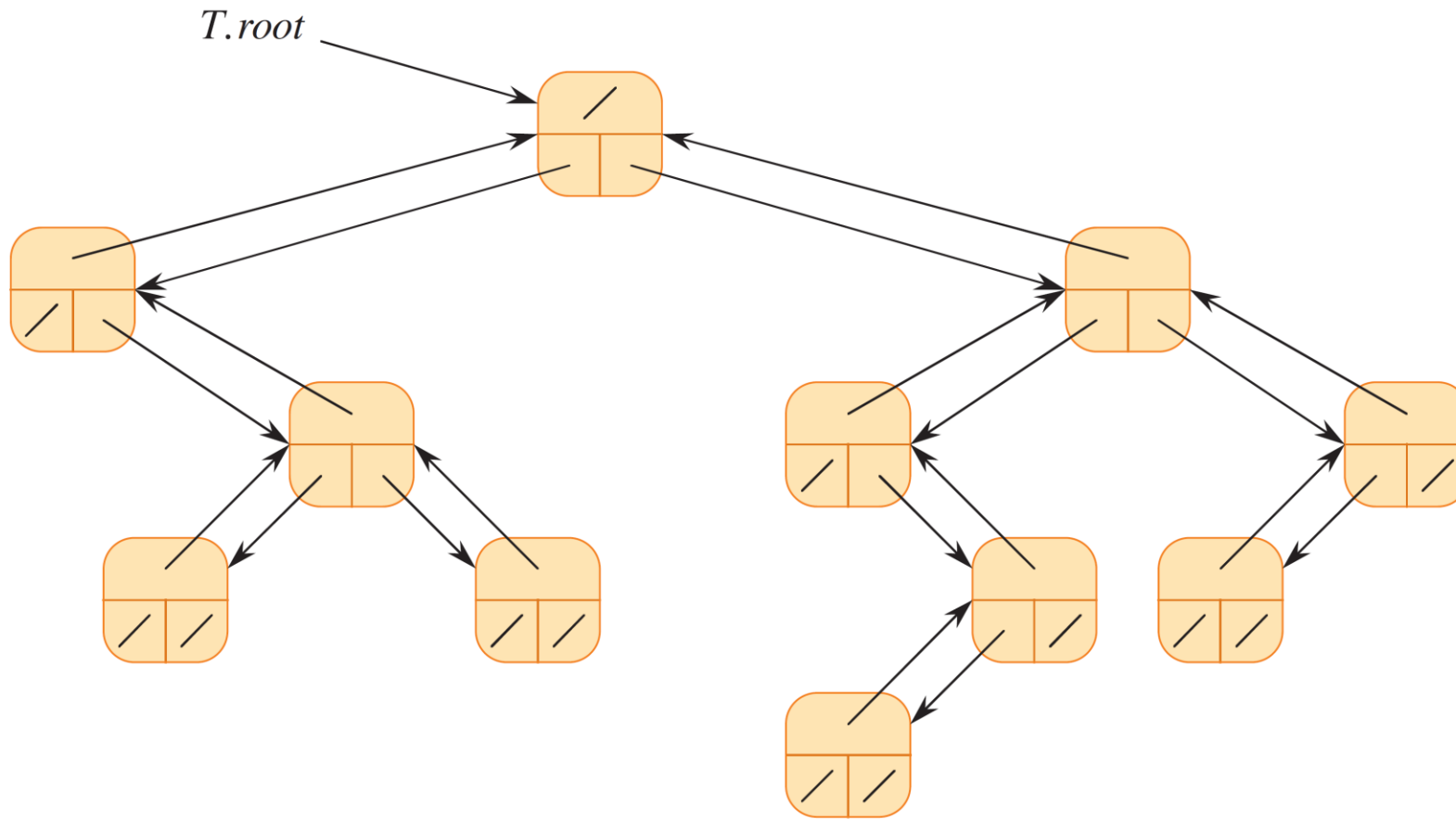
Solve function places

8.387.259 pieces



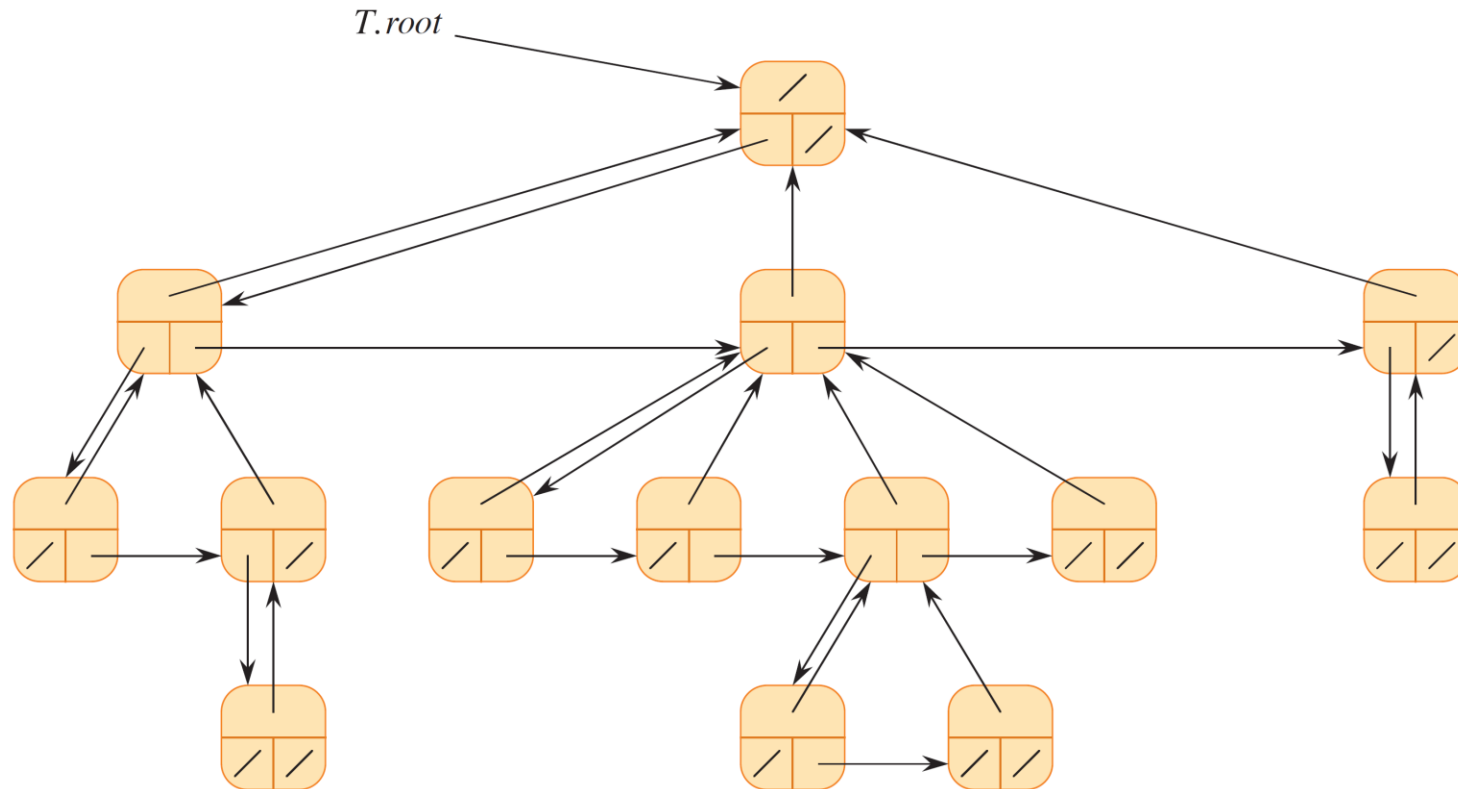
(Jorge Stolfi)

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]



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**

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

Direct addressing: keys $\{0, 1, 2, \dots, m-1\}$

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

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 $T[x.key] = \text{NIL}$

- + 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

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

$$m \ll |U|$$

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

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

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) $6^{|U|}$

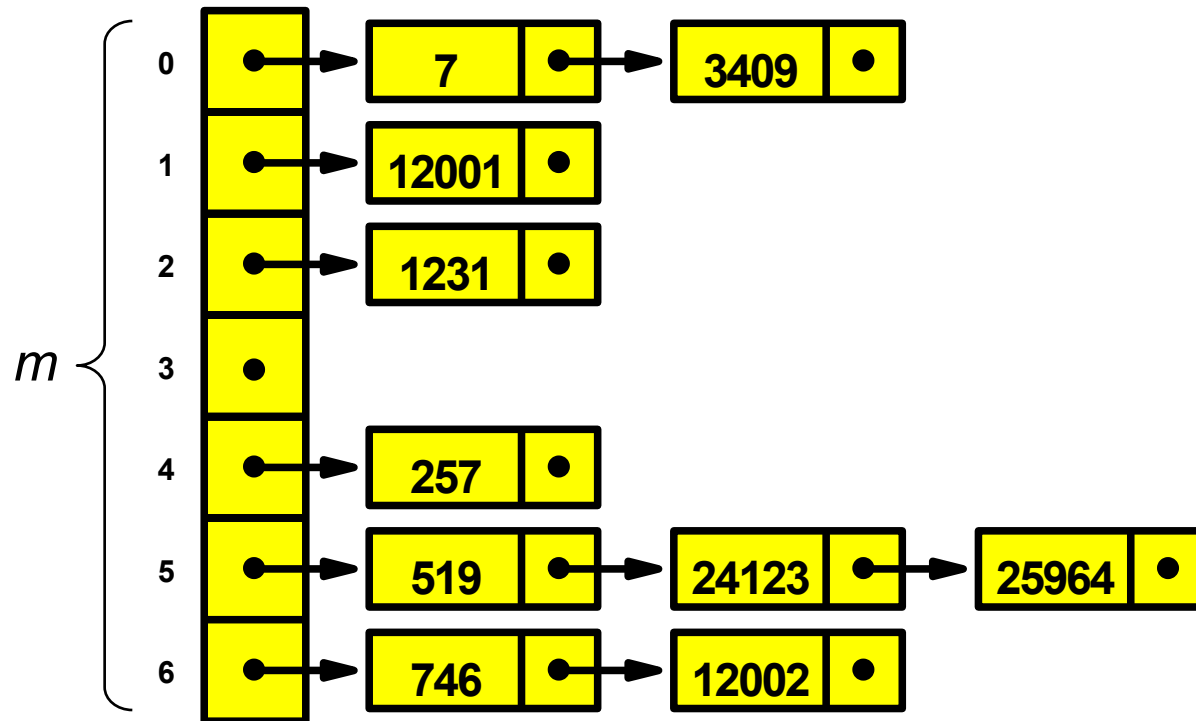
d) $6 \cdot m$

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

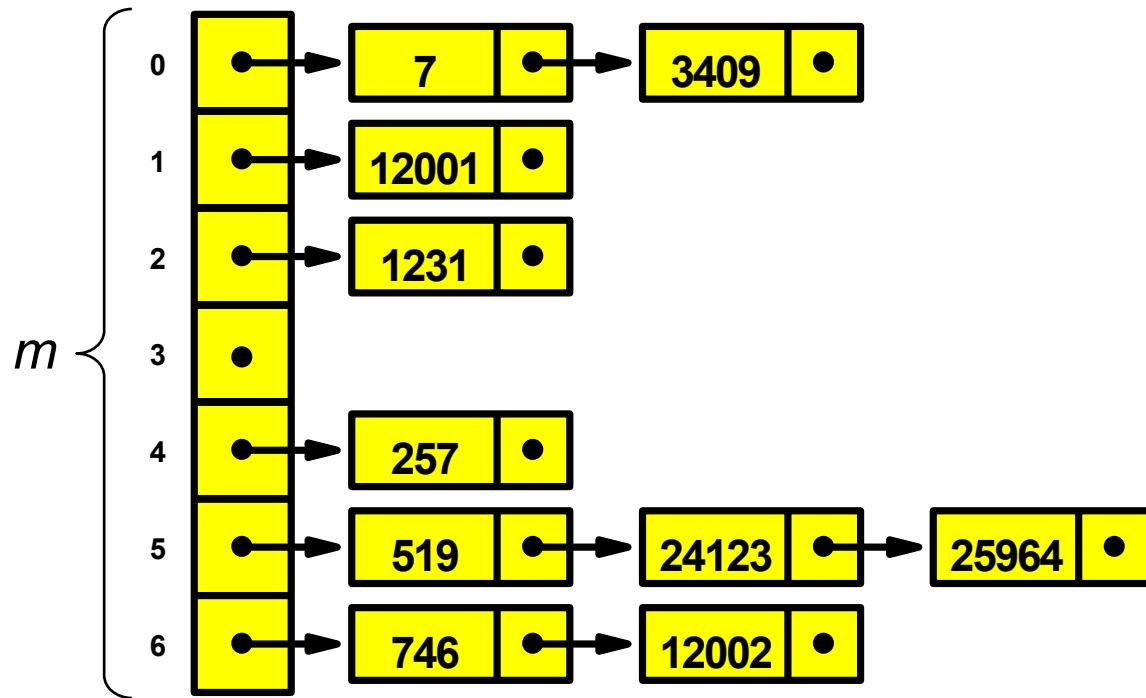


x	$h(x)$
7	0
257	4
519	5
746	6
1231	2
3409	0
12001	1
12002	6
24123	5
25964	5

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

- Store set of keys K
- Choose **random** hash function $h : U \rightarrow \{0, 1, \dots, m-1\}$
- 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:

$$h(\dots x_3 x_2 x_1 10101111) = h(\dots y_3 y_2 y_1 10101111)$$

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

$$h("c_3 c_2 c_1") = h("c_1 c_3 c_2")$$

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

($k = w$ -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

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

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...

$$(s \text{ bits}) \quad \begin{matrix} X_{s/w-1} & & X_1 & & X_0 \\ (w \text{ bits}) & & (w \text{ bits}) & & (w \text{ bits}) \end{matrix}$$

$$X = (b_{s-1}b_{s-2}\dots b_{s-w} \mid b_{s-w-1} \dots b_{2w} \mid b_{2w-1} \dots b_{w+1}b_w \mid b_{w-1} \dots b_2b_1b_0)_2$$

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

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

$p > 2^w$ prime
 $0 < a < p$ random

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

Polynomial evaluation $h_a(x)$

$y = 0$

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

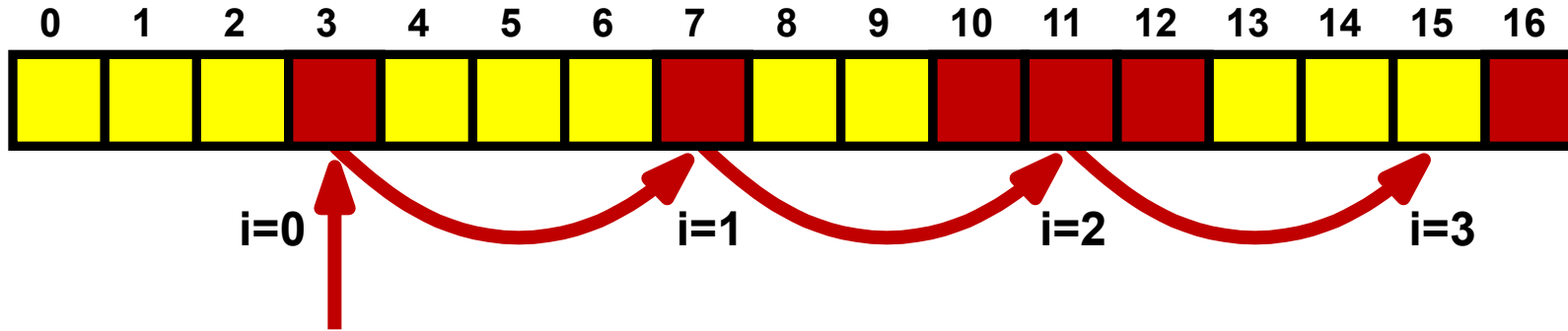
$y = (y \cdot a + x_i) \bmod p$

$h_a(x) = y$

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

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

Open addressing



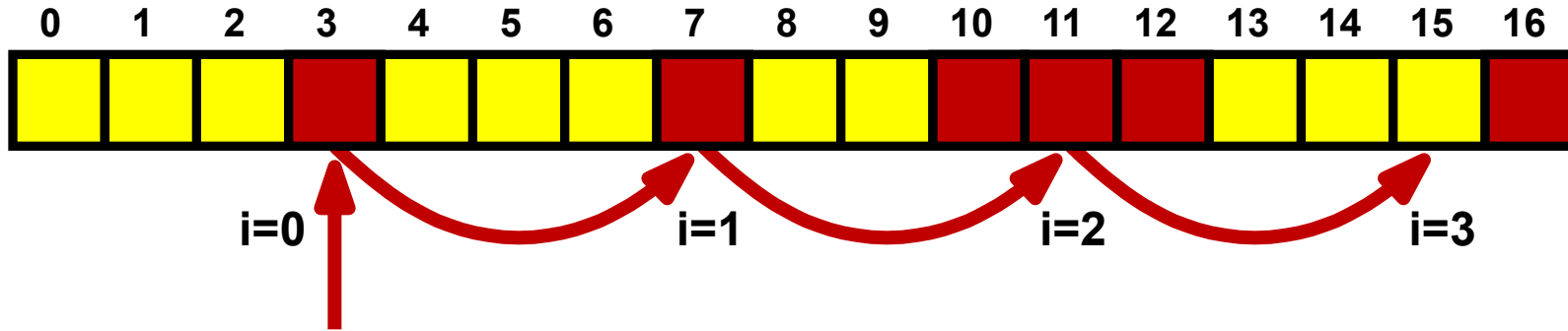
$\text{HASH-INSERT}(T, k)$

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

$\text{HASH-SEARCH}(T, k)$

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

Open addressing – analysis



Uniform hashing

$h(k, 0), h(k, 1), h(k, 2), \dots$ 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

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

for $i = 0, 1, 2, \dots$

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

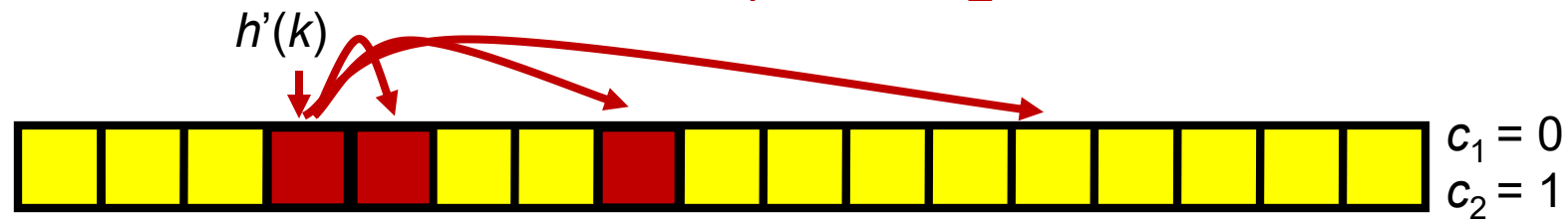
$$h'(x) = (2 \cdot x) \bmod 17$$

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

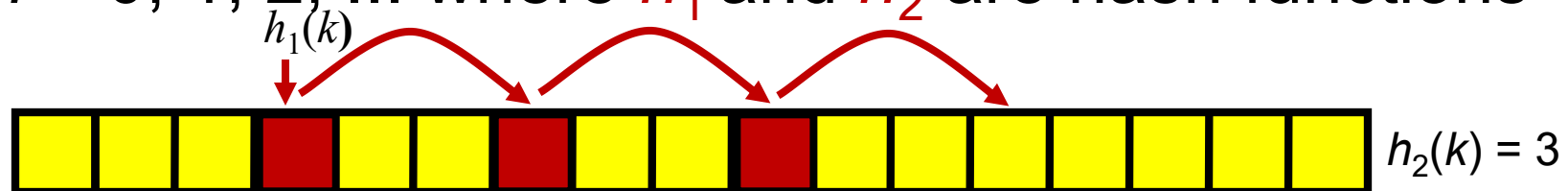


Double hashing

Insert k in first available slot

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

for $i = 0, 1, 2, \dots$ 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

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

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

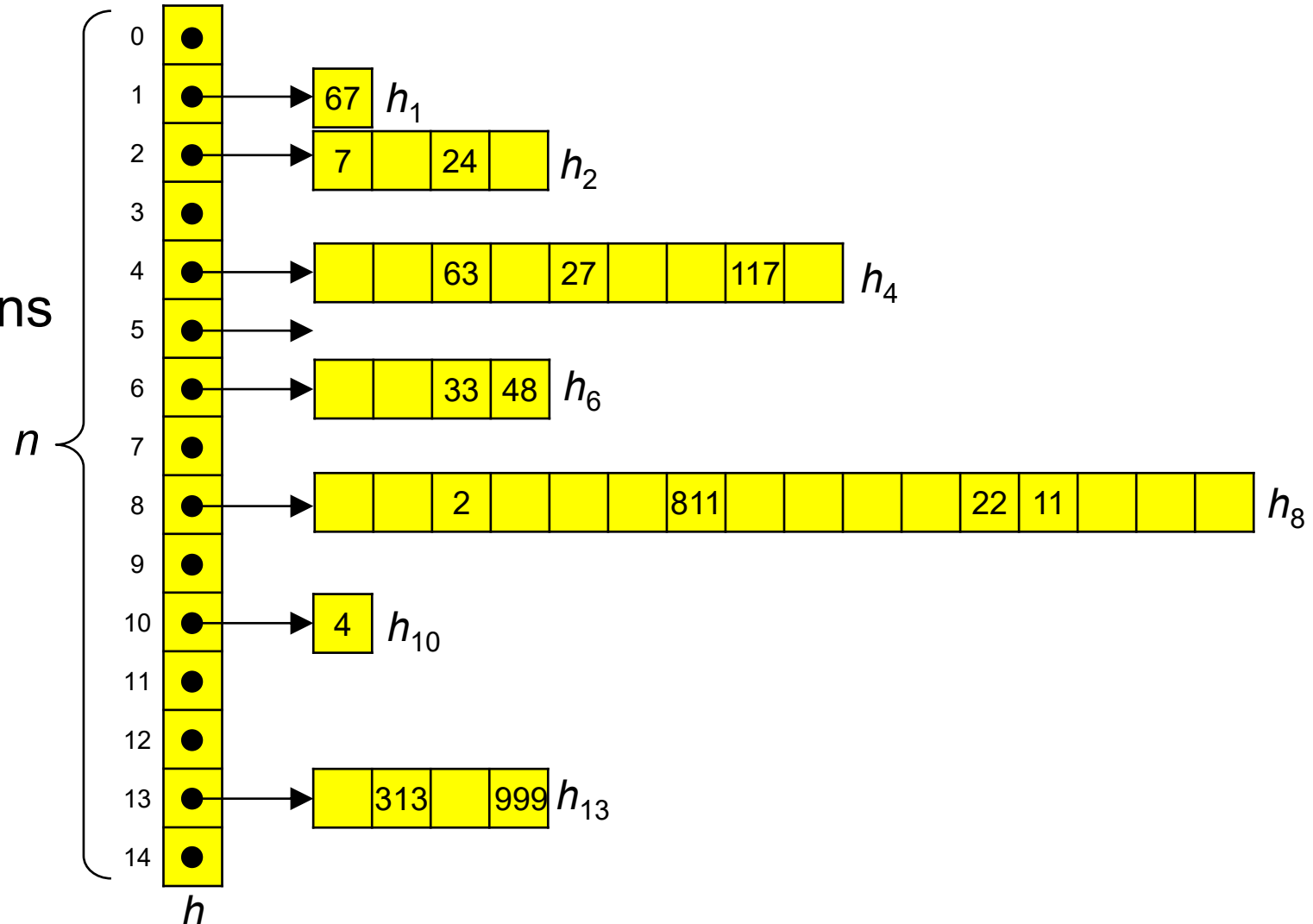
0	1	2	3	4	5	6	7	8	9
5		6	3	2	1	8		7	

exam question August 2010

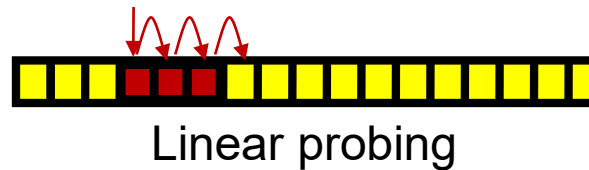
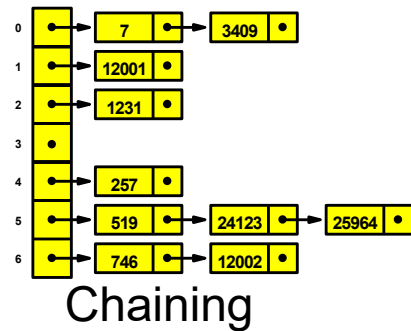
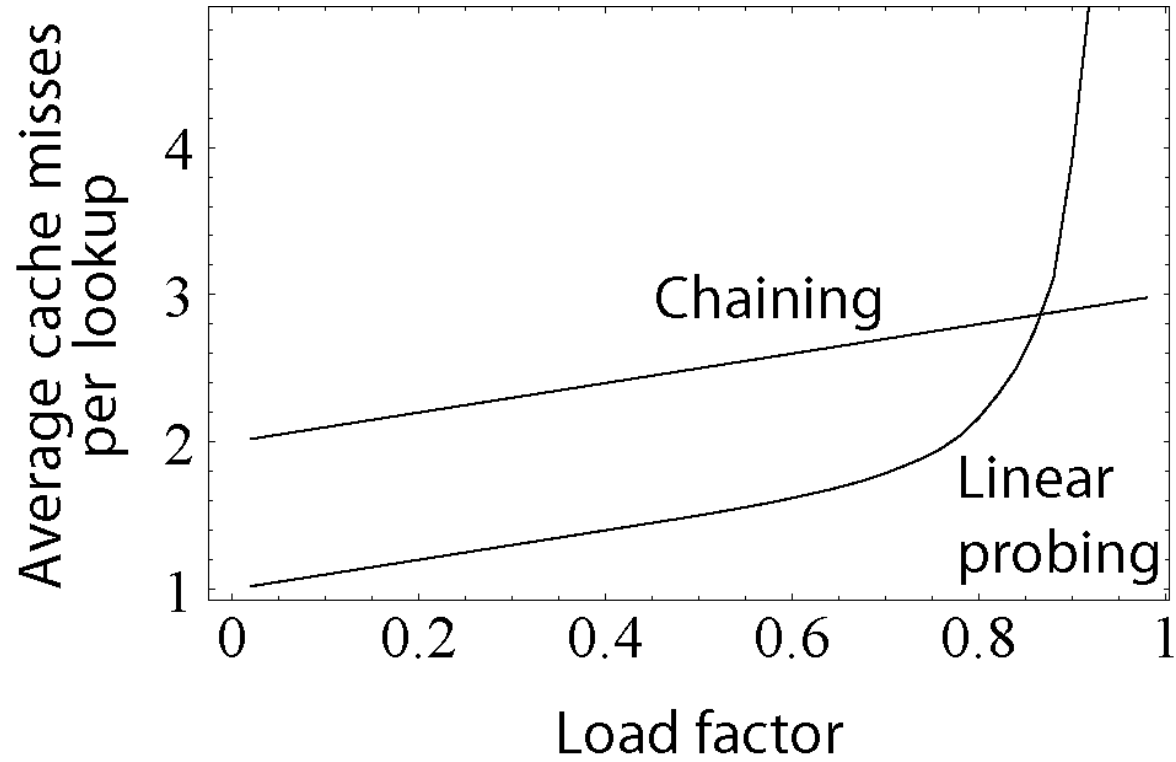
Perfect hashing (static)

- 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

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



Experimental comparison



Hashing

- **Choice of hash function**

- Try different options...
- Universal hash functions

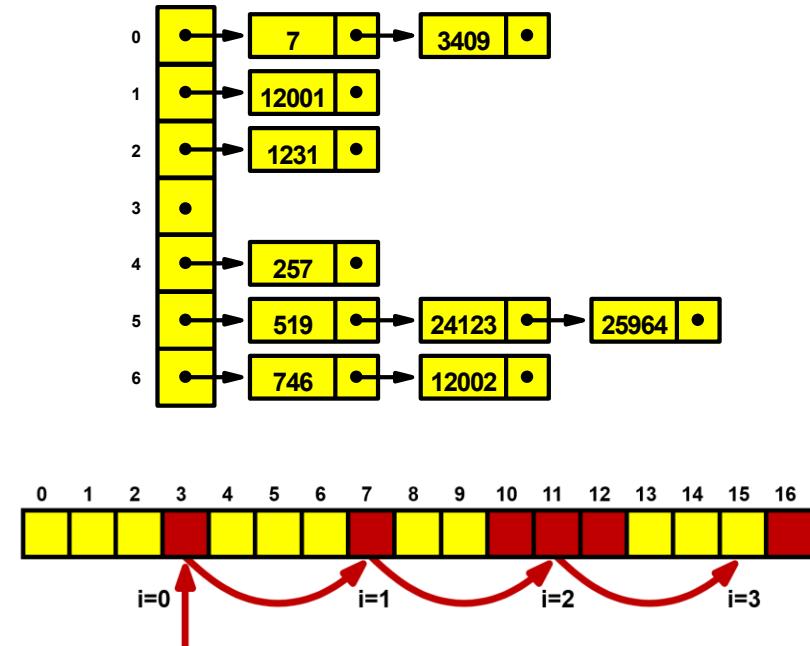
$$h_{a,b}(k) = ((a \cdot k + b) \bmod p) \bmod m$$

- **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**

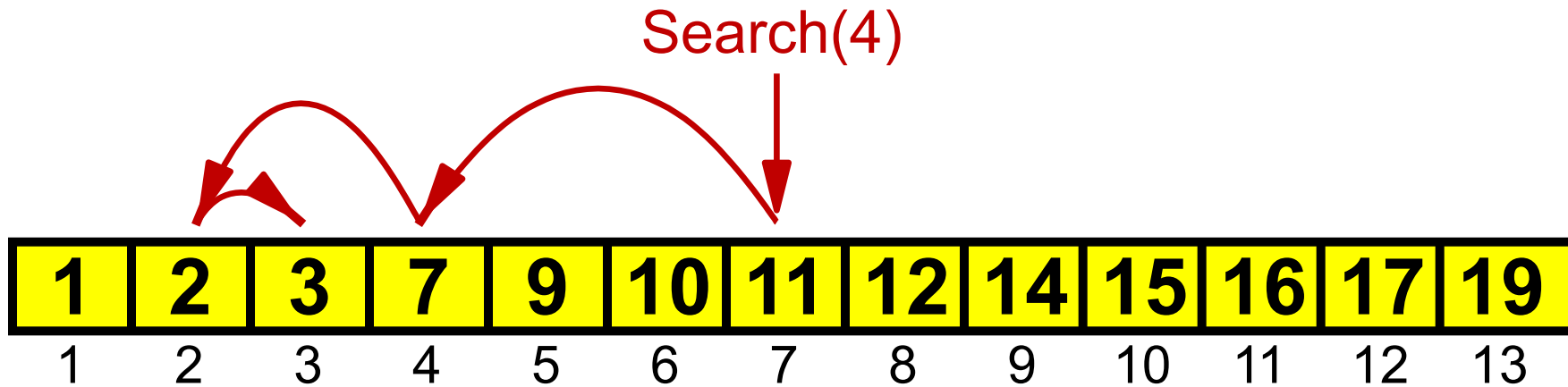
Search(S, k)

Insert(S, x)

Delete(S, x)

Elements can only be compared using " $\leq$ "

(Static) dictionary: sorted array



Search(S, x)	$O(\log n)$
Insert(S, x) Delete(S, x)	$O(n)$



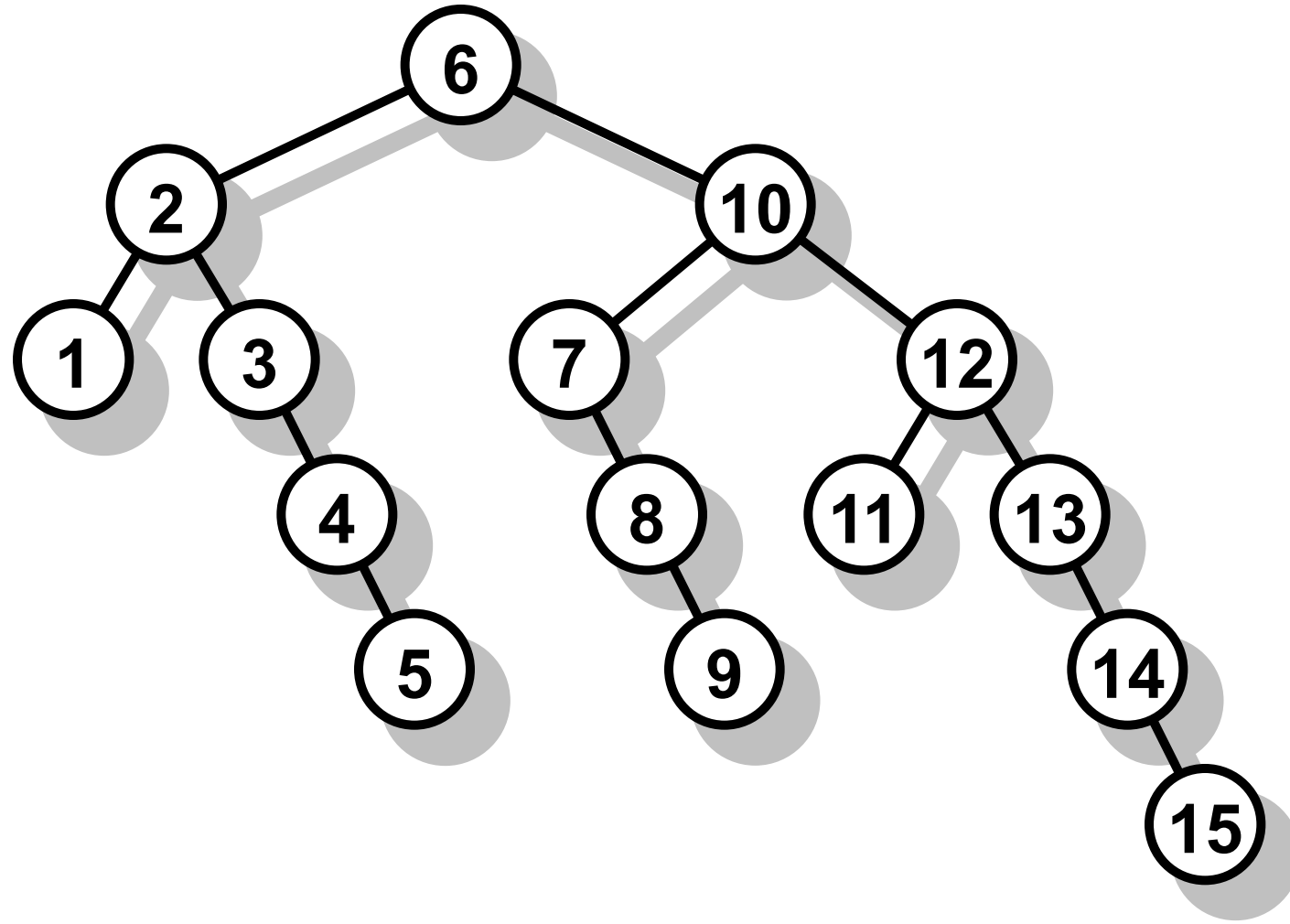
Worst-case search time

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

1	2	3	4	5	6	7	8
2	5	7	9	11	13	42	71

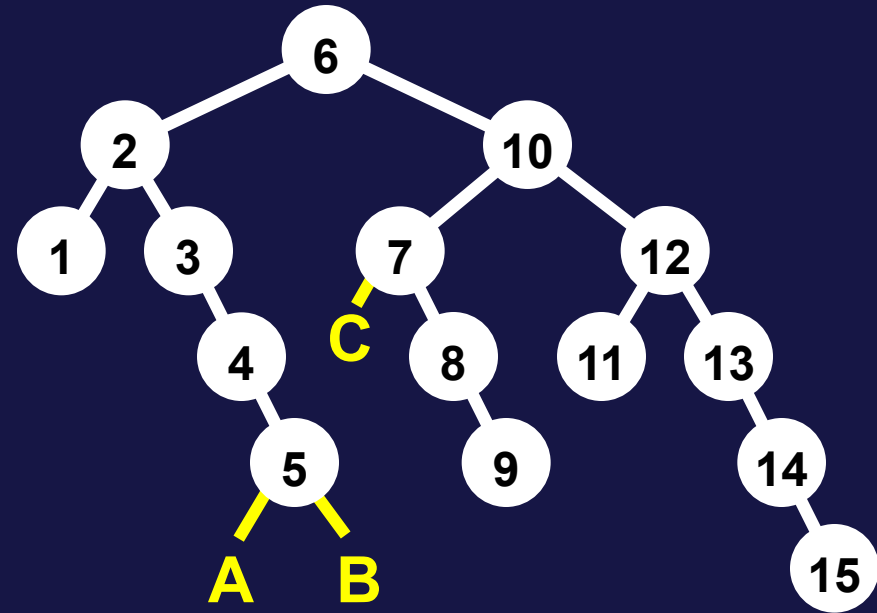
Binary search tree



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)

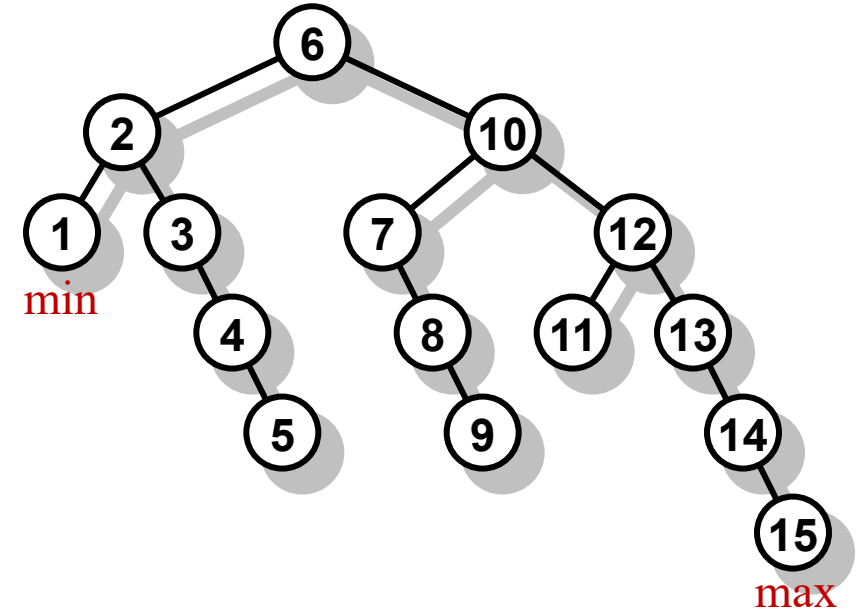
```
1  if  $x == \text{NIL}$  or  $k == x.\text{key}$ 
2      return  $x$ 
3  if  $k < x.\text{key}$ 
4      return TREE-SEARCH( $x.\text{left}, k$ )
5  else return TREE-SEARCH( $x.\text{right}, k$ )
```

ITERATIVE-TREE-SEARCH(x, k)

```
1  while  $x \neq \text{NIL}$  and  $k \neq x.\text{key}$ 
2      if  $k < x.\text{key}$ 
3           $x = x.\text{left}$ 
4      else  $x = x.\text{right}$ 
5  return  $x$ 
```

TREE-SUCCESSOR(x)

```
1  if  $x.\text{right} \neq \text{NIL}$ 
2      return TREE-MINIMUM( $x.\text{right}$ )
3  else
4       $y = x.p$ 
5      while  $y \neq \text{NIL}$  and  $x == y.\text{right}$ 
6           $x = y$ 
7           $y = y.p$ 
8  return  $y$ 
```



INORDER-TREE-WALK(x)

```
1  if  $x \neq \text{NIL}$ 
2      INORDER-TREE-WALK( $x.\text{left}$ )
3      print  $x.\text{key}$ 
4      INORDER-TREE-WALK( $x.\text{right}$ )
```

TREE-MINIMUM(x)

```
1  while  $x.\text{left} \neq \text{NIL}$ 
2       $x = x.\text{left}$ 
3  return  $x$ 
```

TREE-MAXIMUM(x)

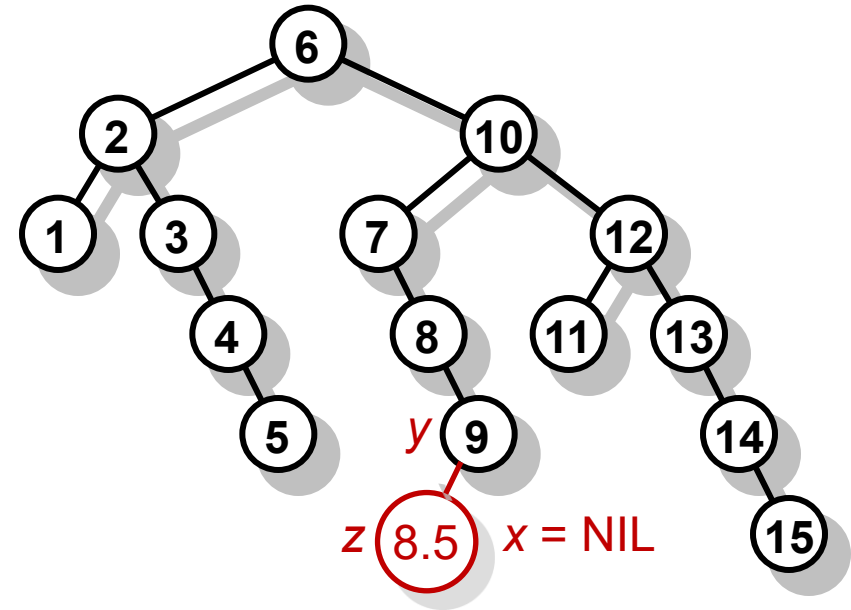
```
1  while  $x.\text{right} \neq \text{NIL}$ 
2       $x = x.\text{right}$ 
3  return  $x$ 
```

Insertions

TREE-INSERT(T, z)

Iterative search {

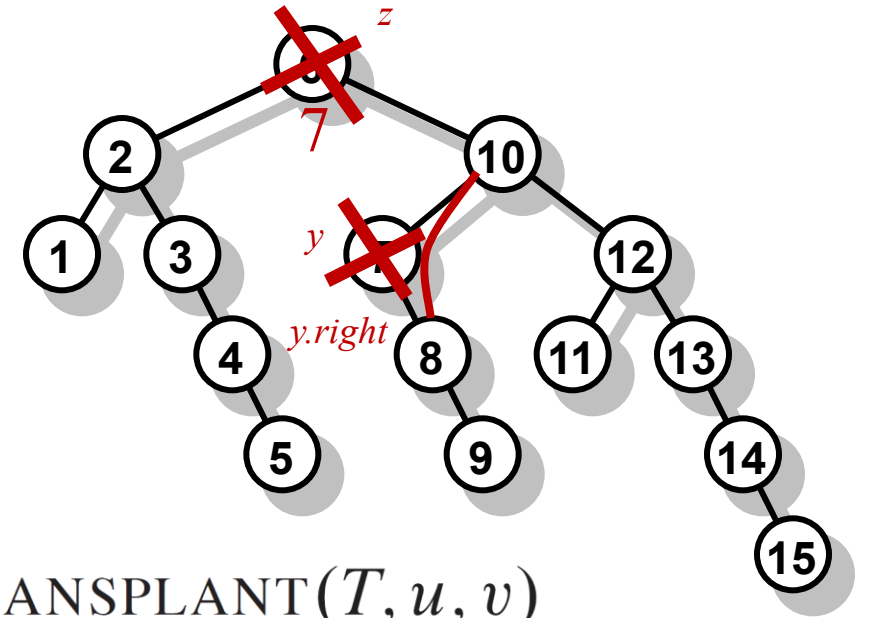
```
1   $x = T.root$ 
2   $y = NIL$ 
3  while  $x \neq NIL$ 
4       $y = x$ 
5      if  $z.key < x.key$ 
6           $x = x.left$ 
7      else  $x = x.right$ 
8   $z.p = y$ 
9  if  $y == NIL$ 
10      $T.root = z$ 
11  elseif  $z.key < y.key$ 
12      $y.left = z$ 
13  else  $y.right = z$ 
```



Deletions

TREE-DELETE(T, z)

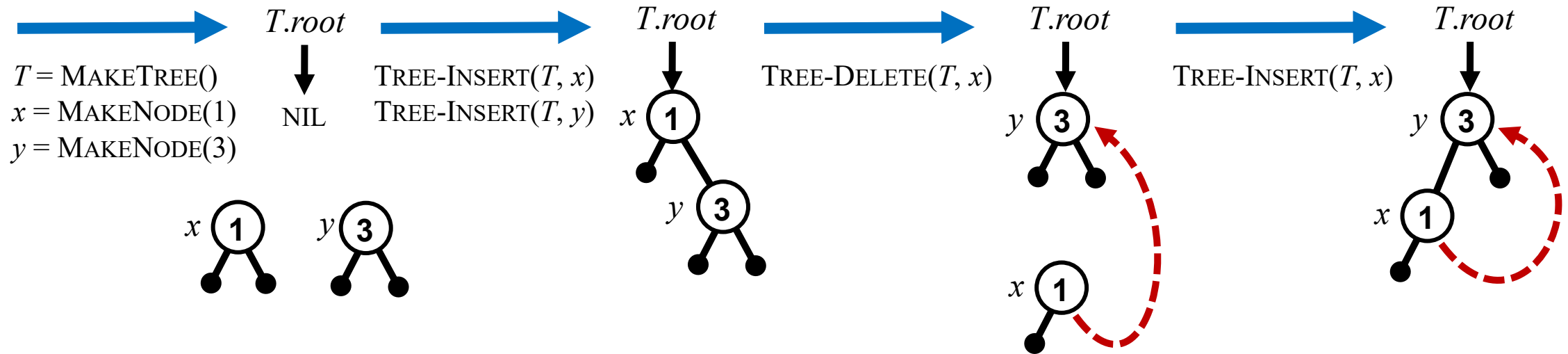
```
1  if  $z.left == \text{NIL}$ 
2      TRANSPLANT( $T, z, z.right$ )
3  elseif  $z.right == \text{NIL}$ 
4      TRANSPLANT( $T, z, z.left$ )
5  else  $y = \text{TREE-MINIMUM}(z.right)$ 
6      if  $y \neq z.right$ 
7          TRANSPLANT( $T, y, y.right$ )
8           $y.right = z.right$ 
9       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



TRANSPLANT(T, u, v)

```
1  if  $u.p == \text{NIL}$ 
2       $T.root = v$ 
3  elseif  $u == u.p.left$ 
4       $u.p.left = v$ 
5  else  $u.p.right = v$ 
6  if  $v \neq \text{NIL}$ 
7       $v.p = u.p$ 
```

Warning



- CLRS pseudocode does not reset the deleted nodes pointers, and assumes the inserted node's pointers are NIL
- $\text{TREE-SEARCH}(T.\text{root}, 2)$ infinite loop 😞

Binary search trees

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

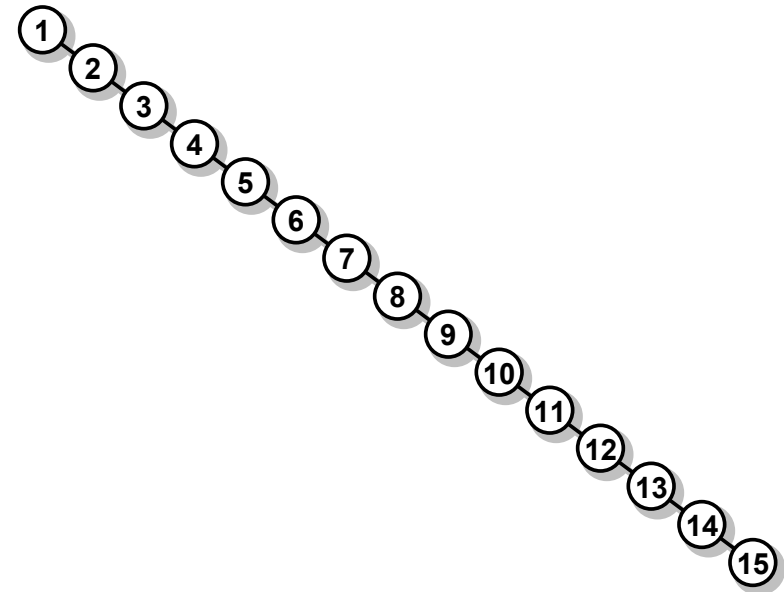
h = height, n = # elements

Height
of a binary search tree?

Largest and smallest heights

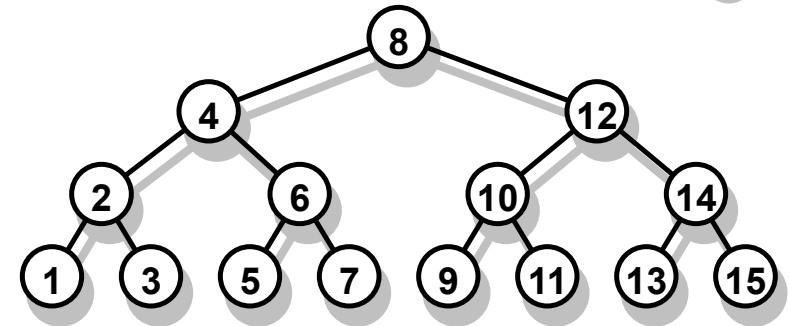
Insert in increasing order

– Height n



Perfect balanced
(smallest possible height)

– Height $1 + \lfloor \log_2 n \rfloor$



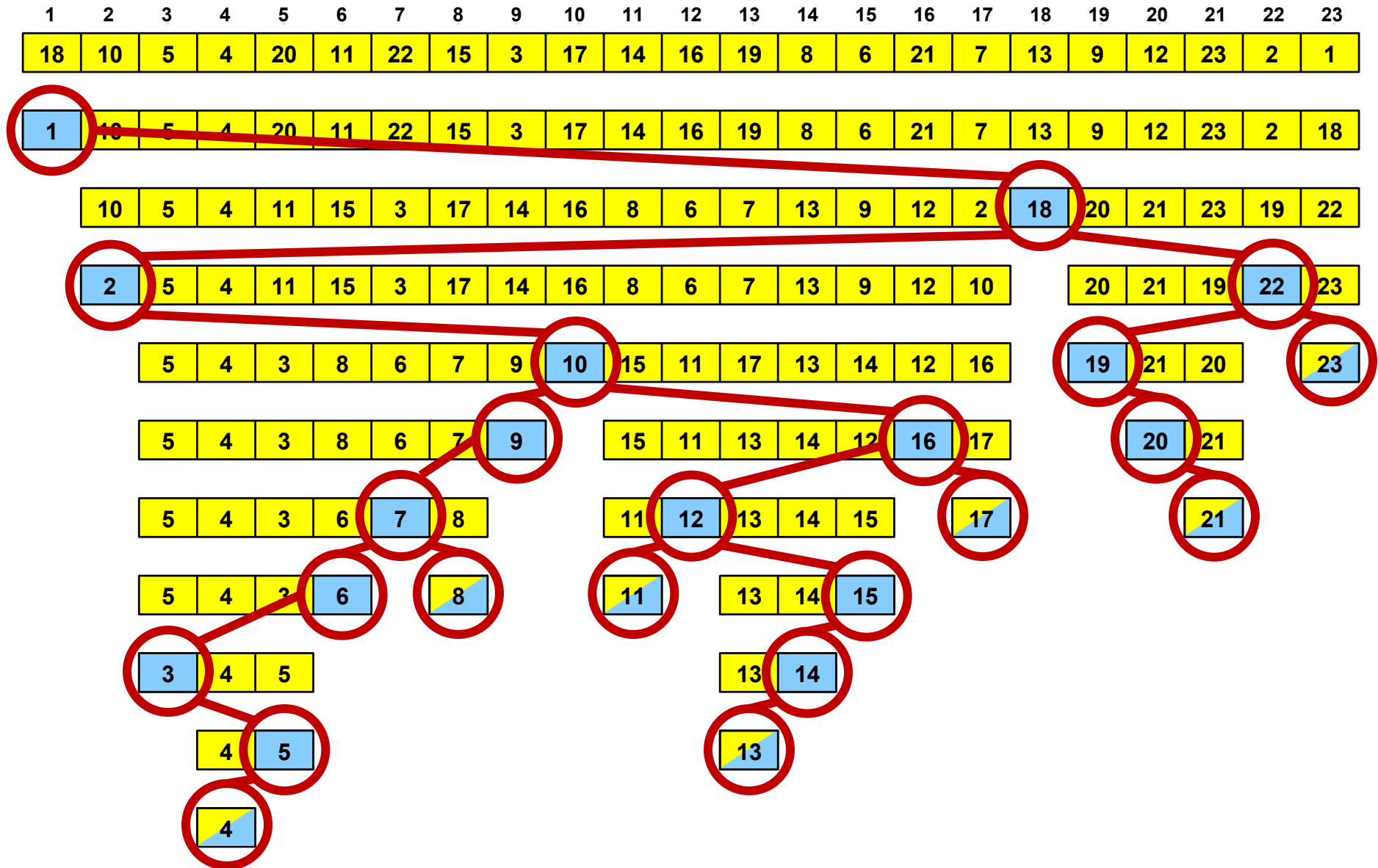
Random insertions into binary search trees

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 $\equiv$ 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

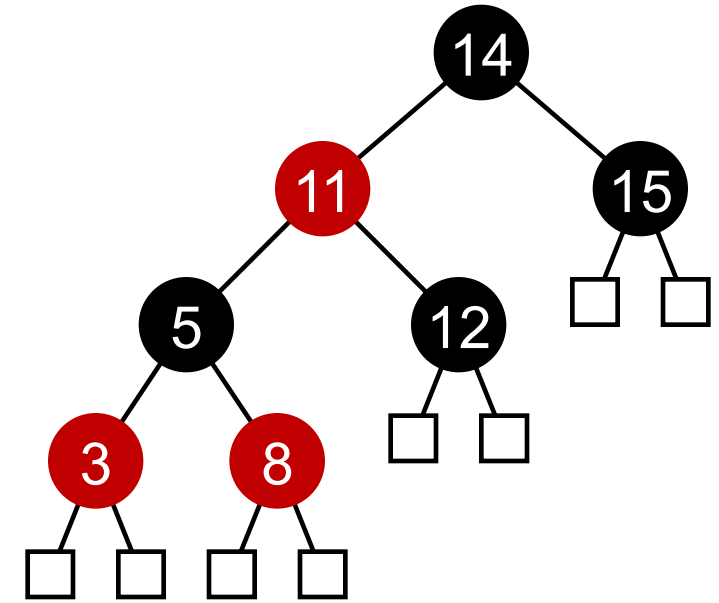
Binary search trees with **height** $O(\log n)$

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**

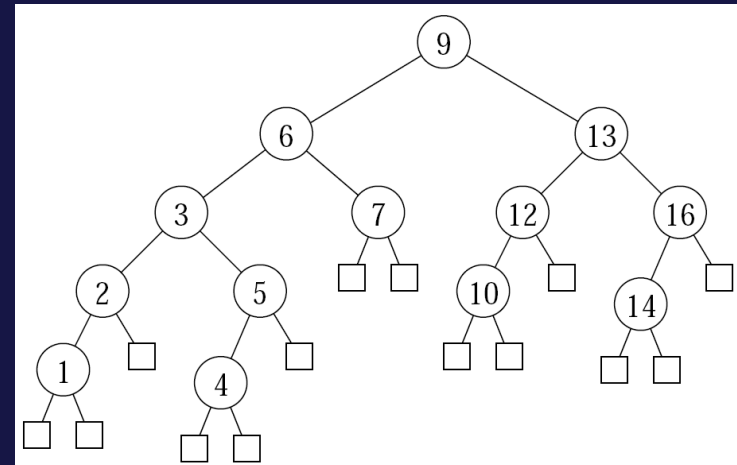
Theorem

A red-black tree has height $\leq 2 \cdot \log (n+1)$

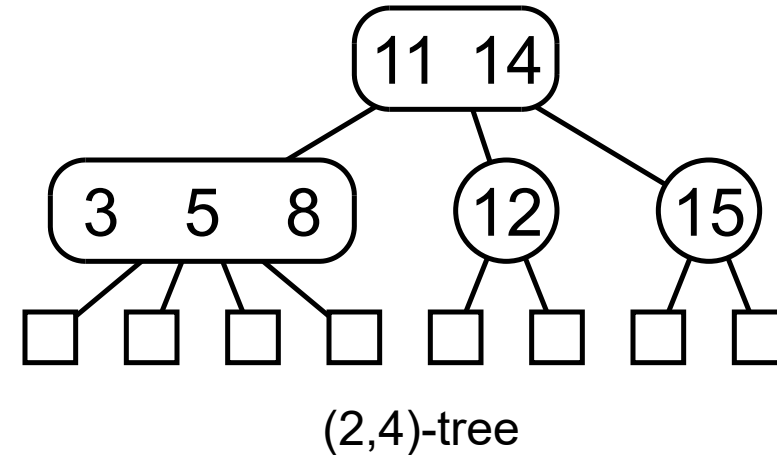
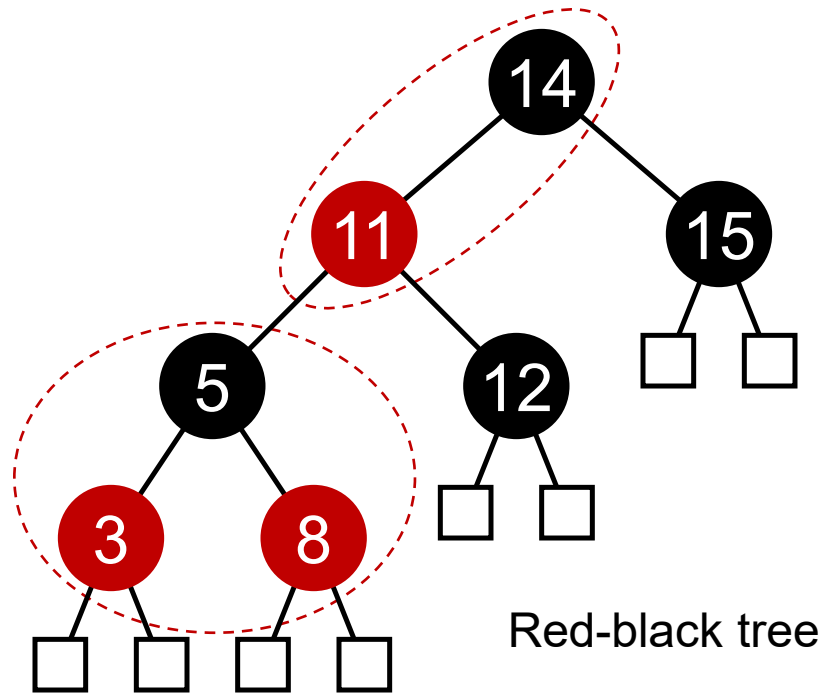


Which nodes should be colored red to make the tree a valid red-black tree ?

- 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



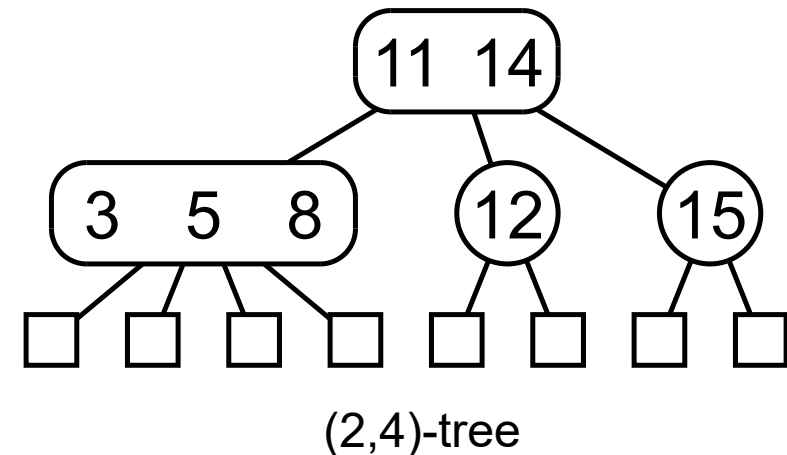
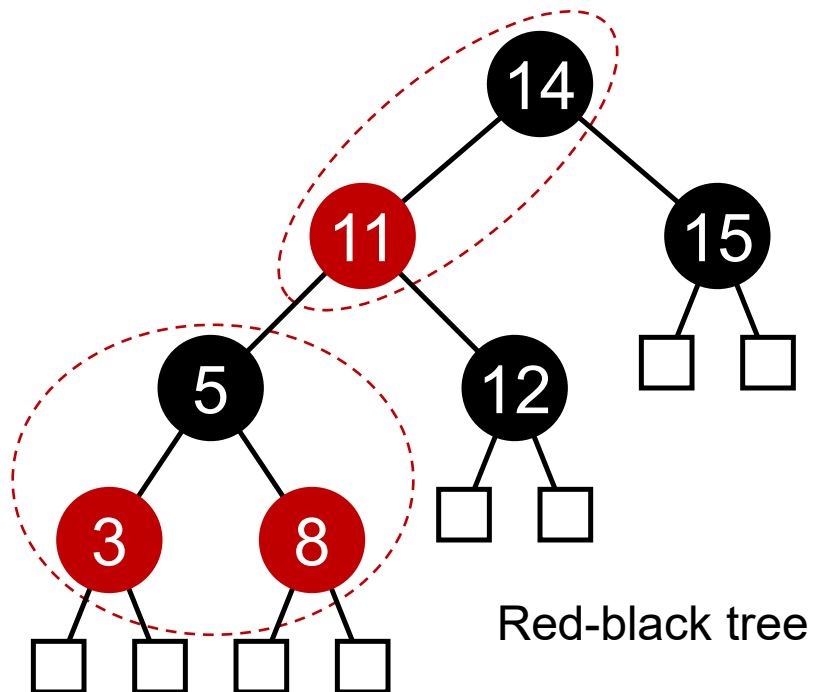
Red-black vs (2,4)-trees



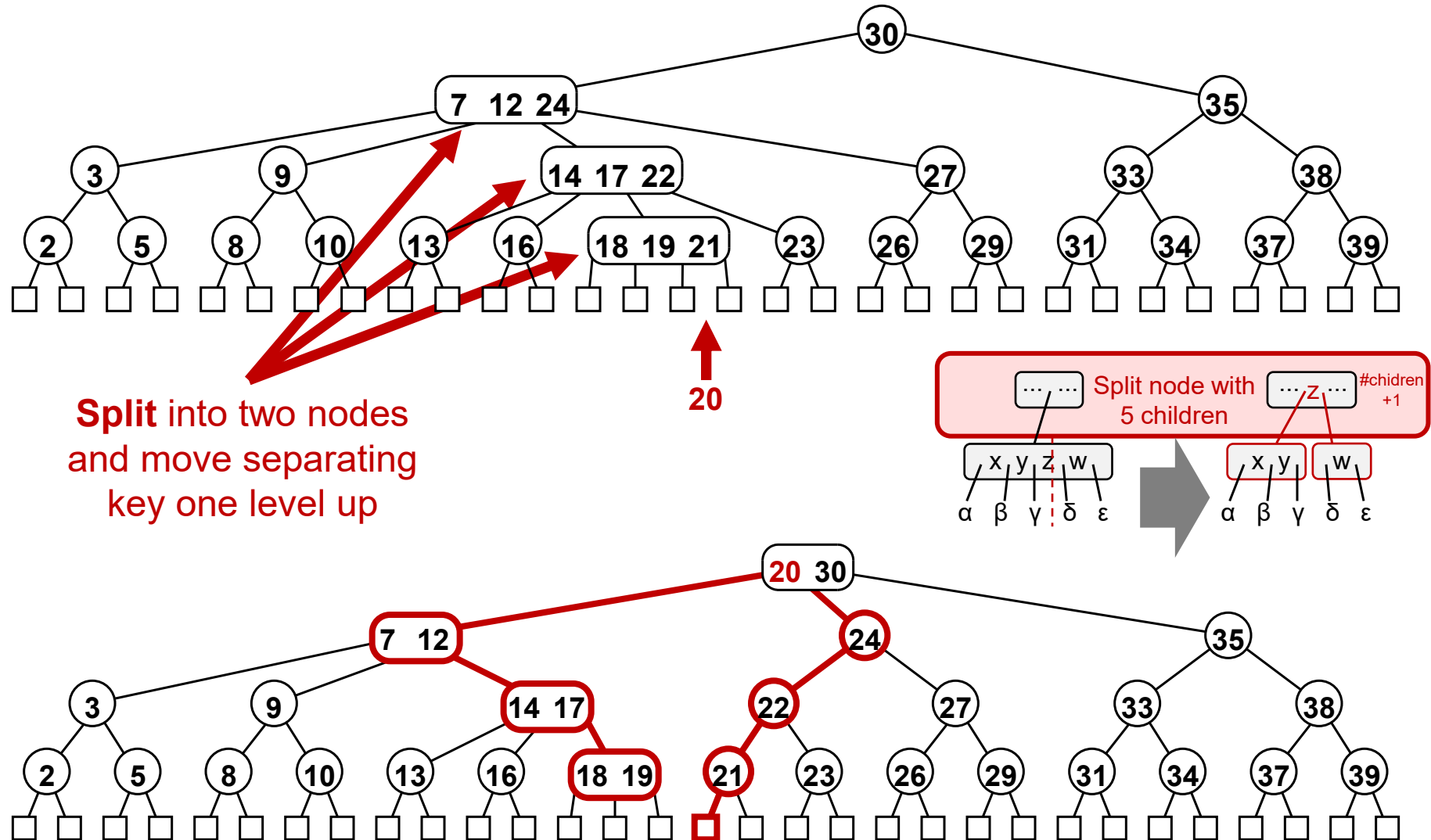
(2,4)-tree $\equiv$ red-black tree
where all red nodes are merged with parent

(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



(2,4)-tree insertion

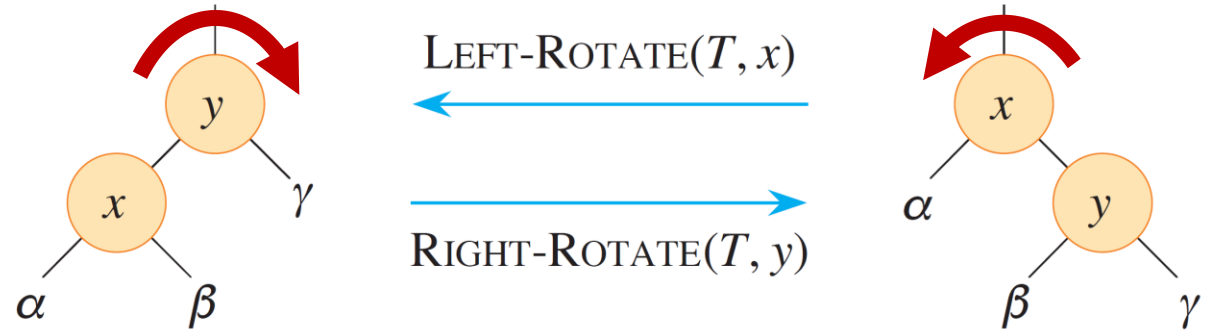


Red-black tree updates

Rotations

LEFT-ROTATE(T, x)

```
1   $y = x.right$ 
2   $x.right = y.left$ 
3  if  $y.left \neq T.nil$ 
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

Unbalanced
insertion

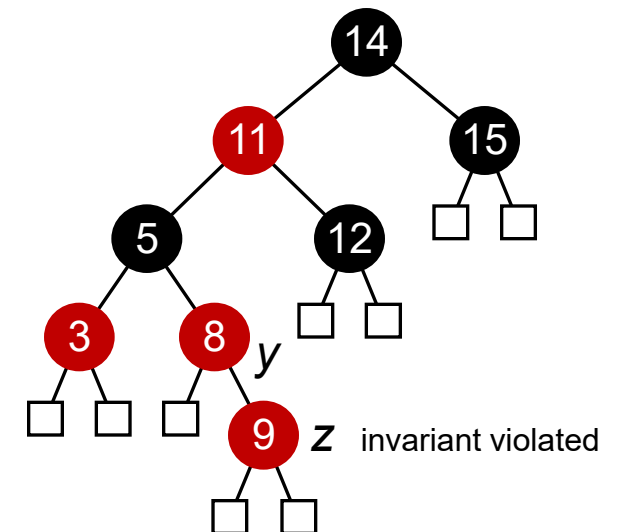
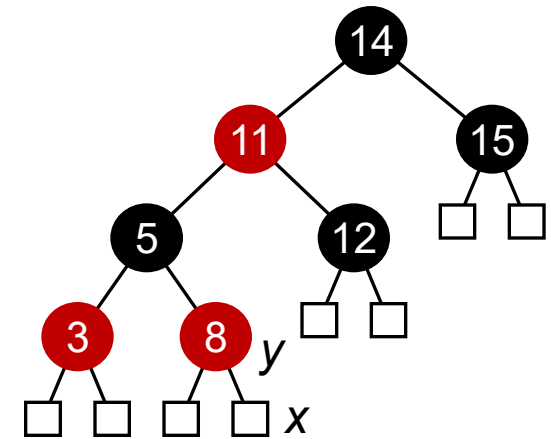
search

create z

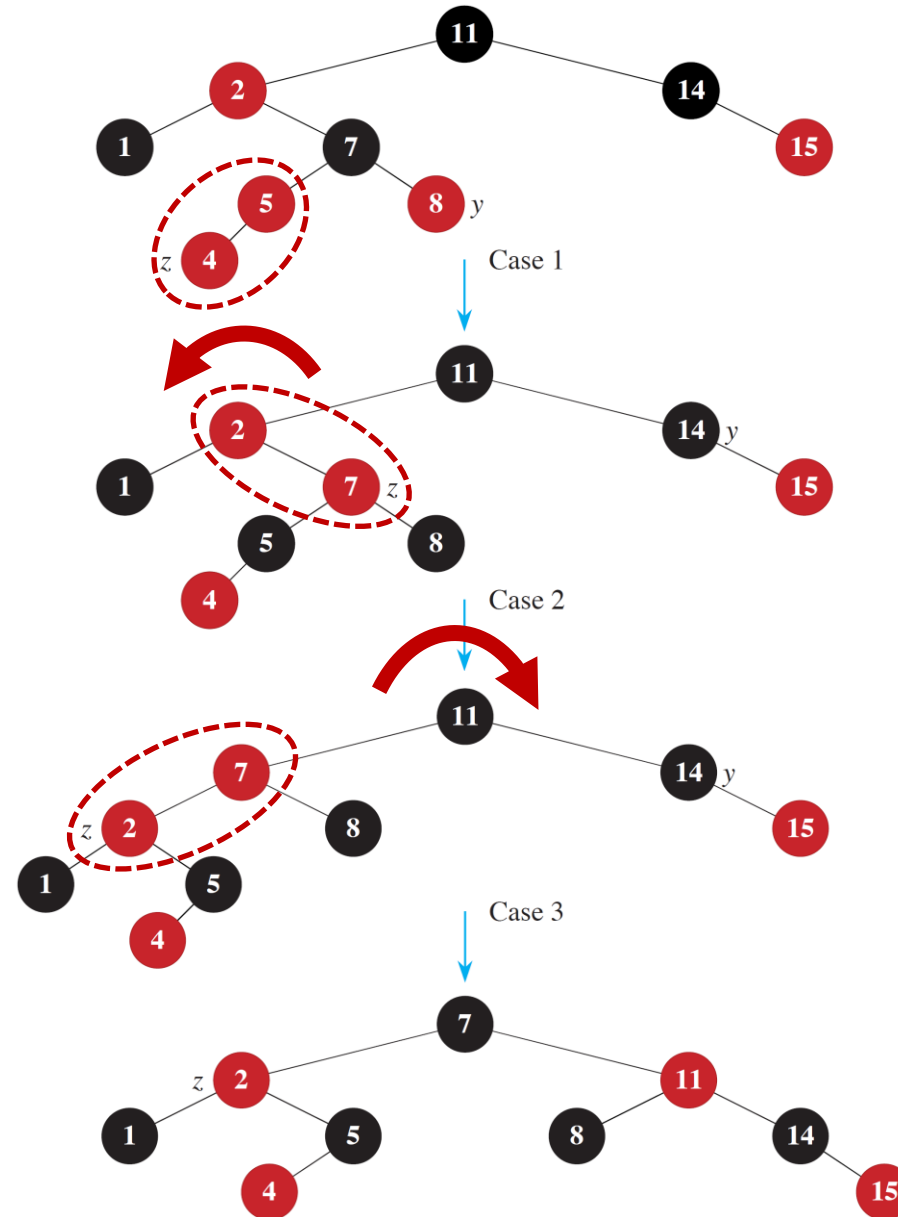
RB-INSERT(T, z)

```
1   $x = T.root$ 
2   $y = T.nil$ 
3  while  $x \neq T.nil$ 
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( $T, z$ )
```

Insert(9)



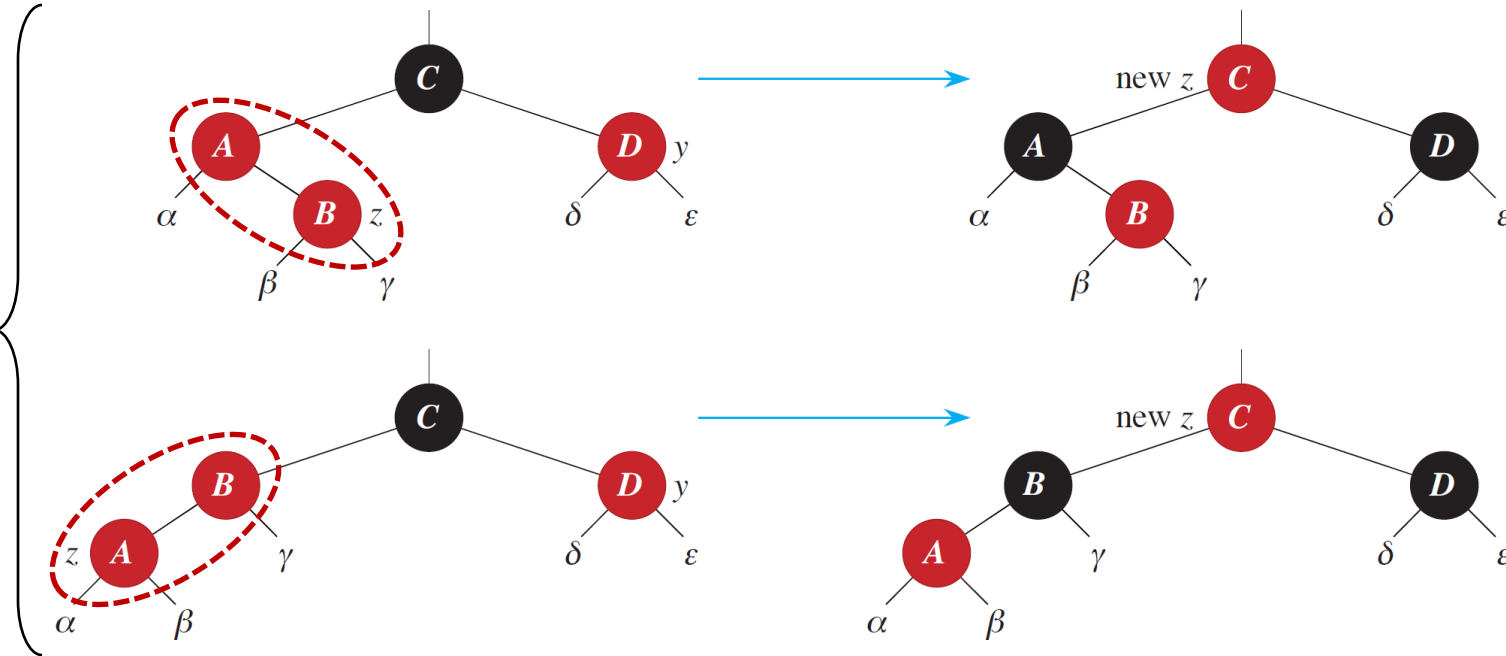
Example : Rebalancing after insertion



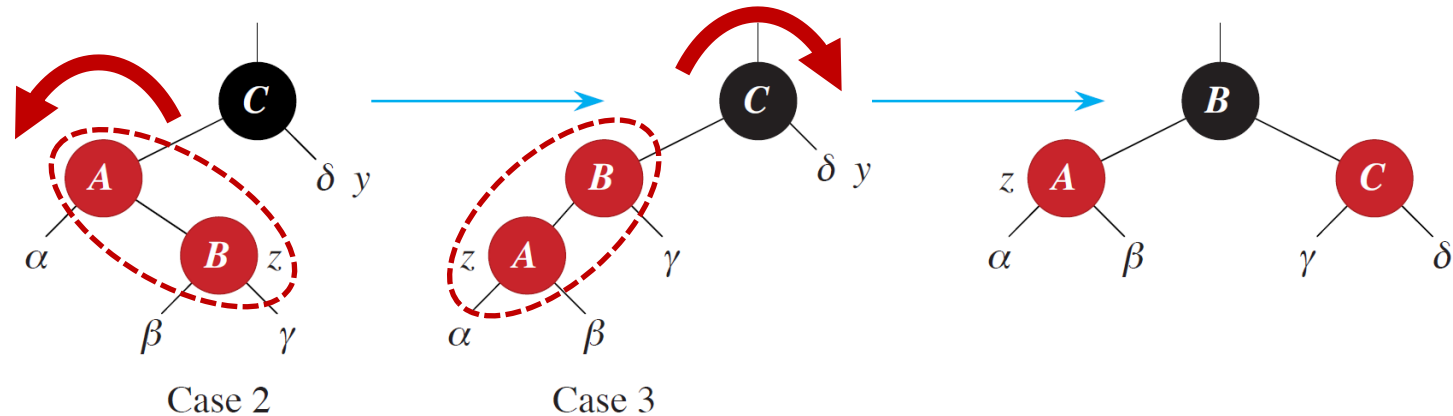
Red-black tree insert rebalancing

Recolor C and its two red children

Case 1



Final rebalancing

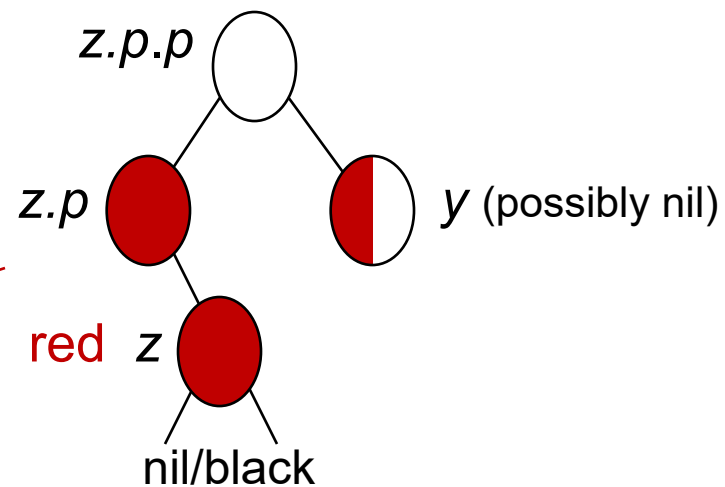


RB-INSERT-FIXUP(T, z)

```

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

```



case 1

case 2

case 3

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

- a) 1
- b) 2
- c) $\log n$
- d) $2 \cdot \log n$
- e) Don't know

Delete

Unbalanced deletion

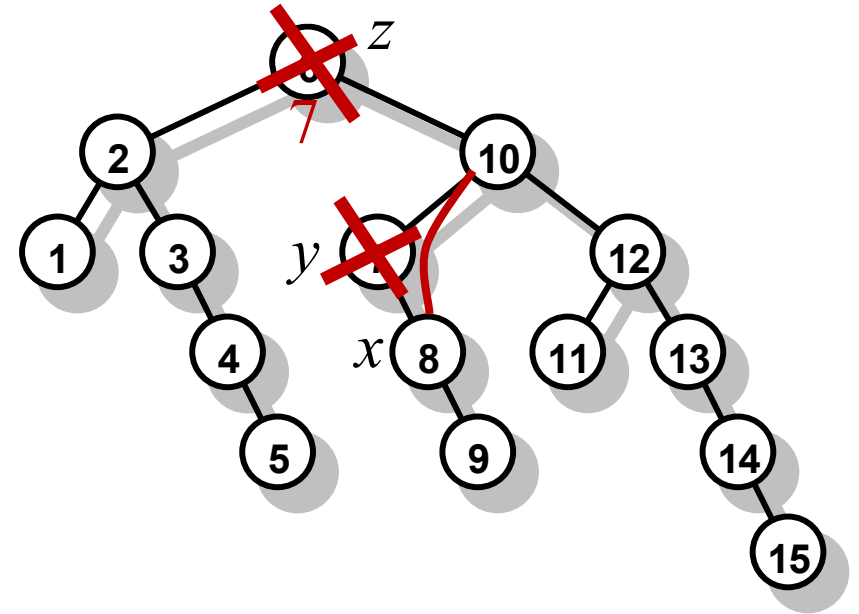
RB-DELETE(T, z)

```

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

```

x and $y.\text{right}$ can be $T.\text{nil}$



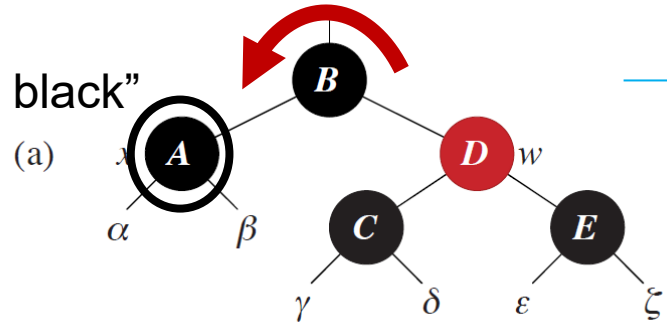
RB-TRANSPLANT(T, u, v)

```

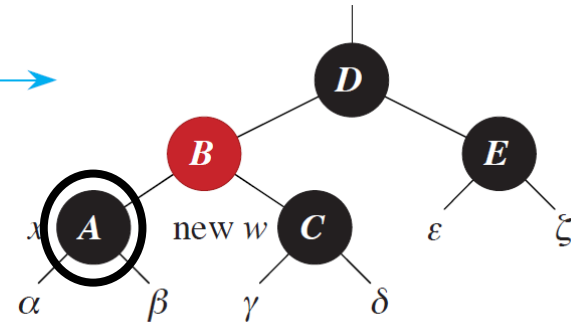
1  if  $u.p == T.\text{nil}$ 
2       $T.\text{root} = v$ 
3  elseif  $u == u.p.\text{left}$ 
4       $u.p.\text{left} = v$ 
5  else  $u.p.\text{right} = v$ 
6   $v.p = u.p$ 

```


"double black"

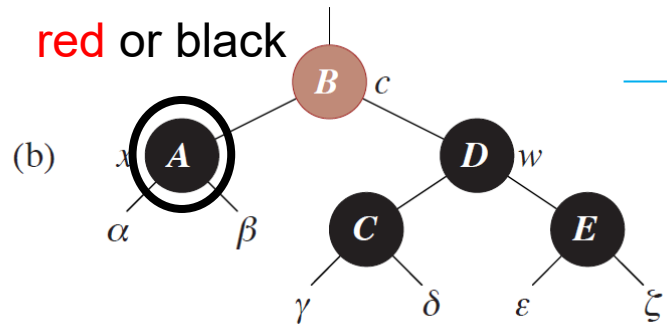


Case 1

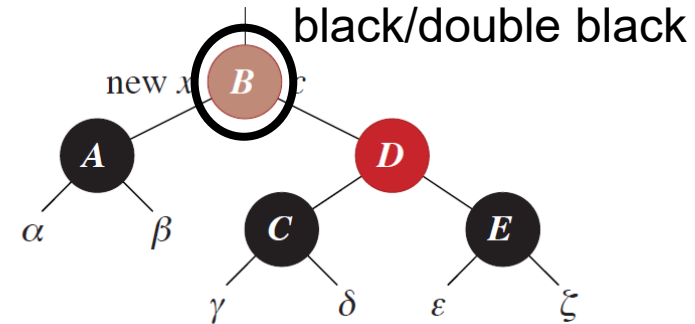


Case 1 $\rightarrow$ 2,3,4

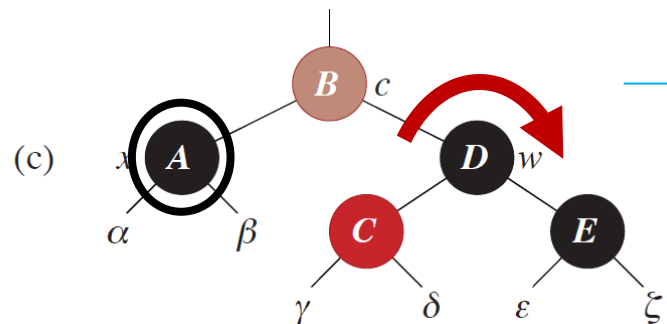
red or black



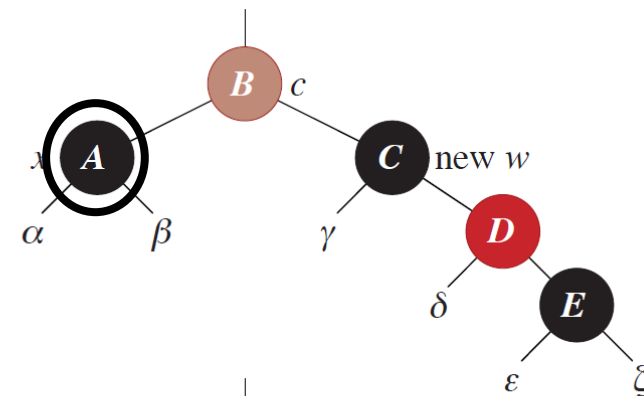
Case 2



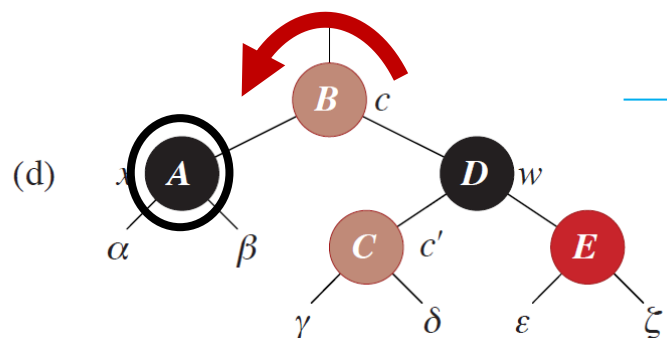
Case 2 $\rightarrow$ Problem x moves closer to the root or done



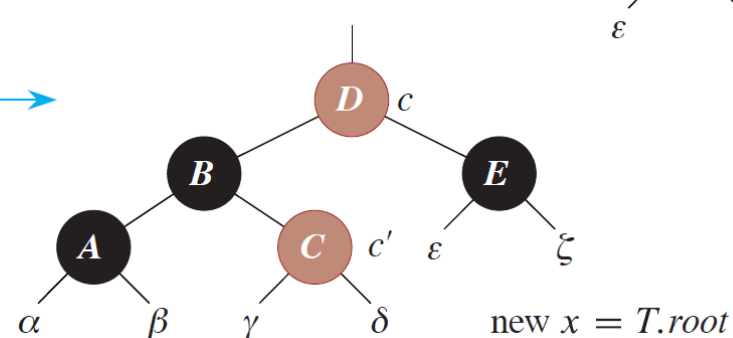
Case 3



Case 3 $\rightarrow$ 4
(and done)



Case 4



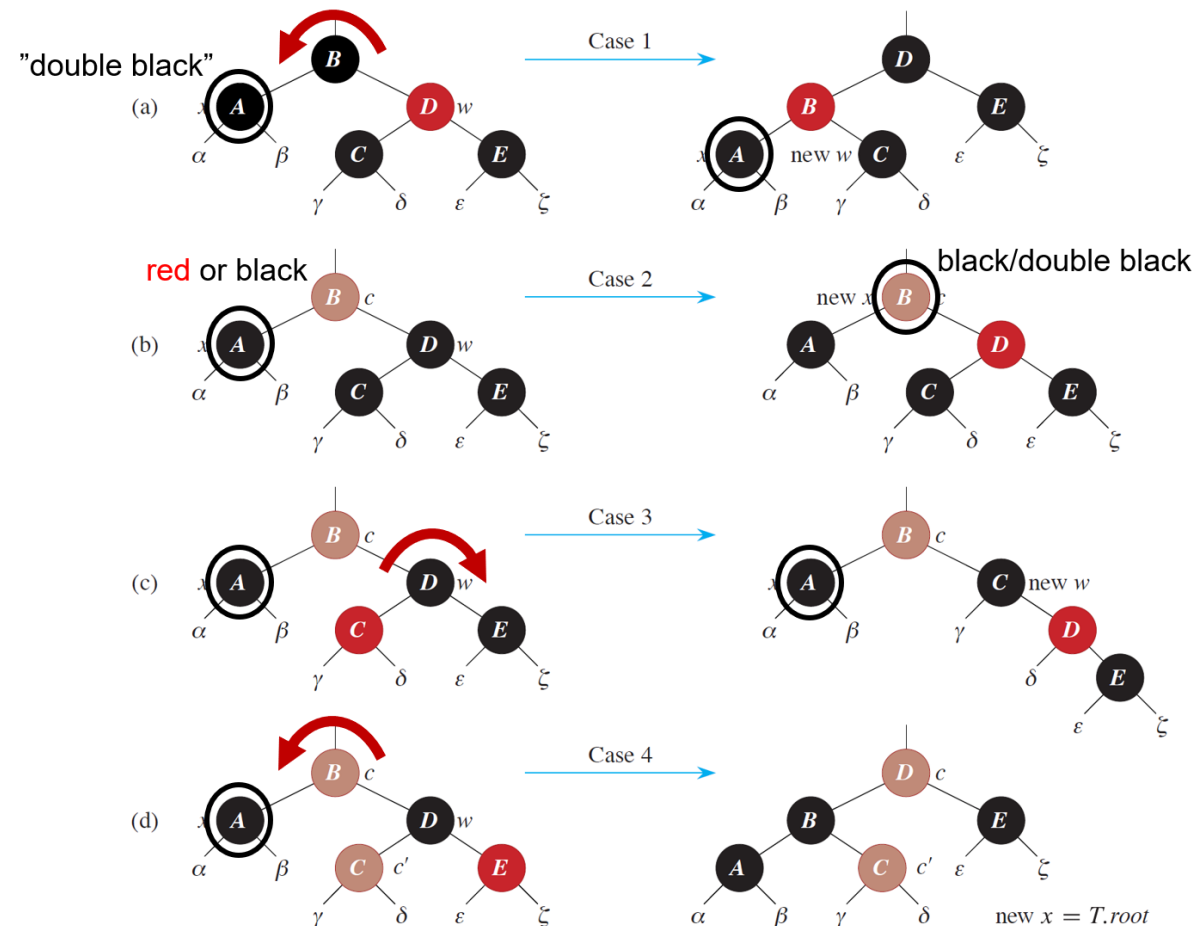
Case 4 $\rightarrow$ done

RB-DELETE-FIXUP(T, x)

```

1  while  $x \neq T.root$  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( $T, x.p$ )
8               $w = x.p.right$ 
9          if  $w.left.color == BLACK$  and  $w.right.color == BLACK$ 
10              $w.color = RED$ 
11              $x = x.p$ 
12         else
13             if  $w.right.color == BLACK$ 
14                  $w.left.color = BLACK$ 
15                  $w.color = RED$ 
16                 RIGHT-ROTATE( $T, w$ )
17                  $w = x.p.right$ 
18              $w.color = x.p.color$ 
19              $x.p.color = BLACK$ 
20              $w.right.color = BLACK$ 
21             LEFT-ROTATE( $T, x.p$ )
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) $O(\log n)$

Delete(S, x)

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

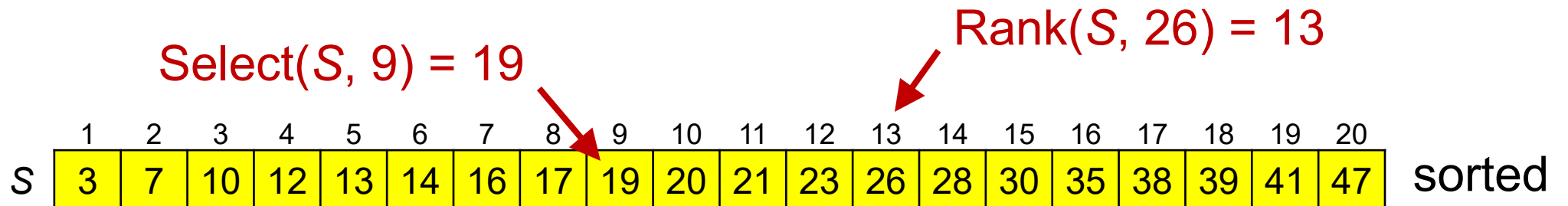
Augmented search trees: Dynamic rank, interval trees
[CLRS, Chapter 17]

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

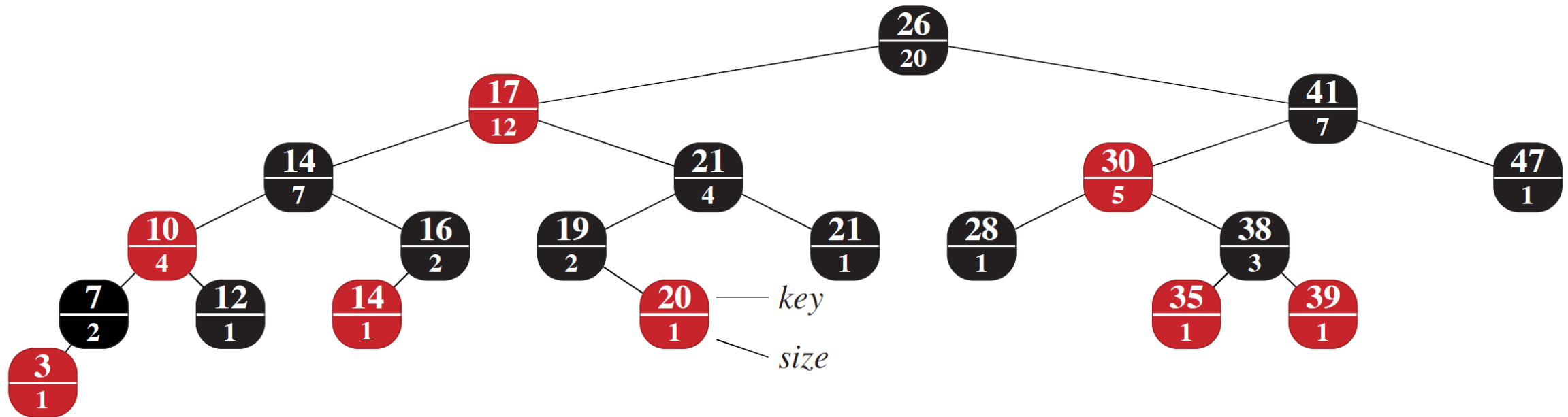
Dynamic rank (order statistics)

Insert(S, x)
Delete(S, x)
Select(S, i)
Rank(S, x)

$O(\log n)$



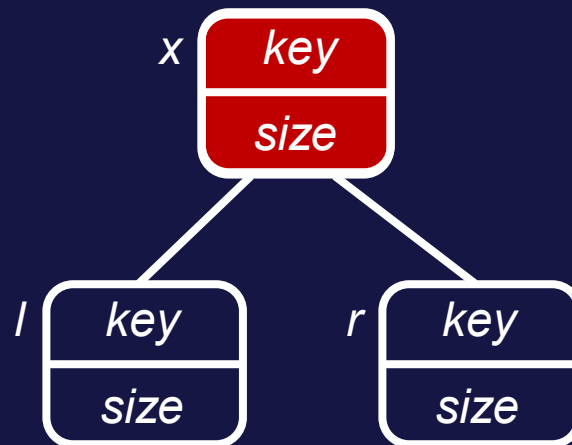
Dynamic rank



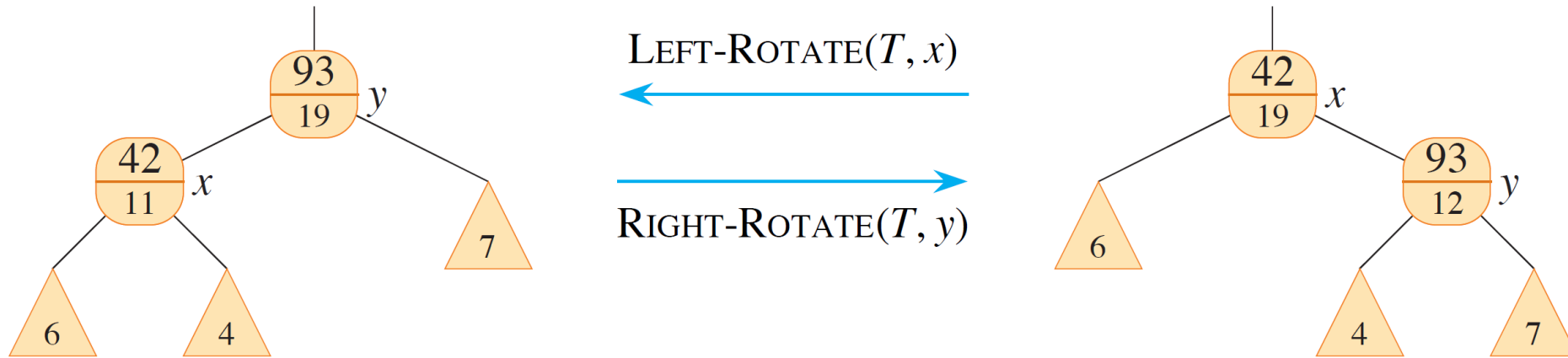
- Find the i^{th} **smallest**, insert, delete
- Maintain **red-black** tree
- Augment each node with the **size of the subtree**

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



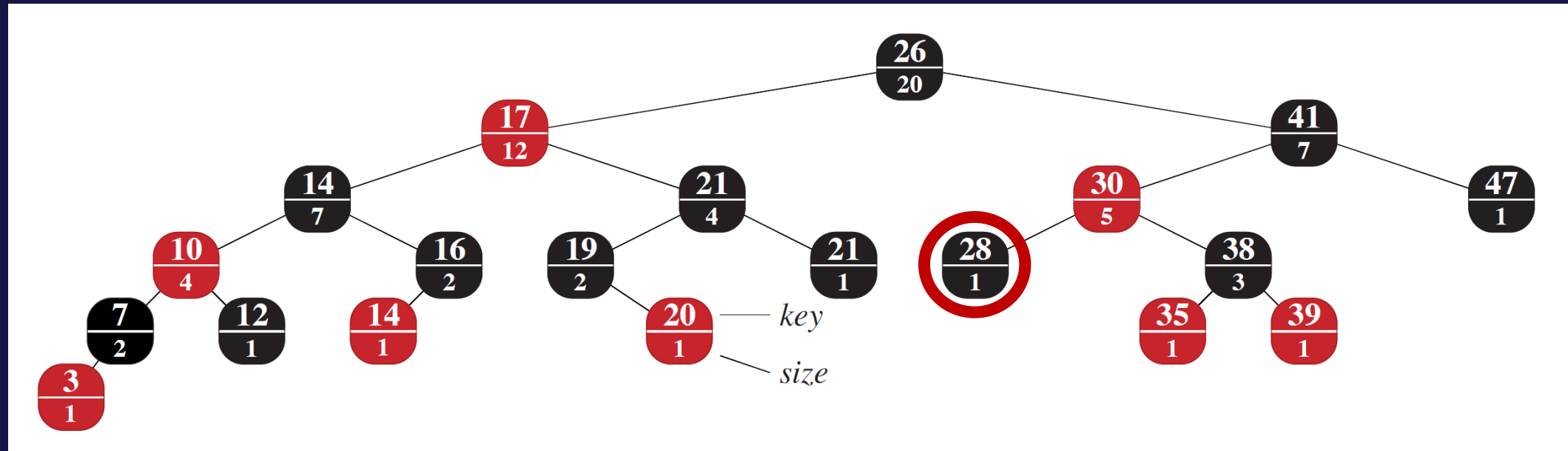
Dynamic rank



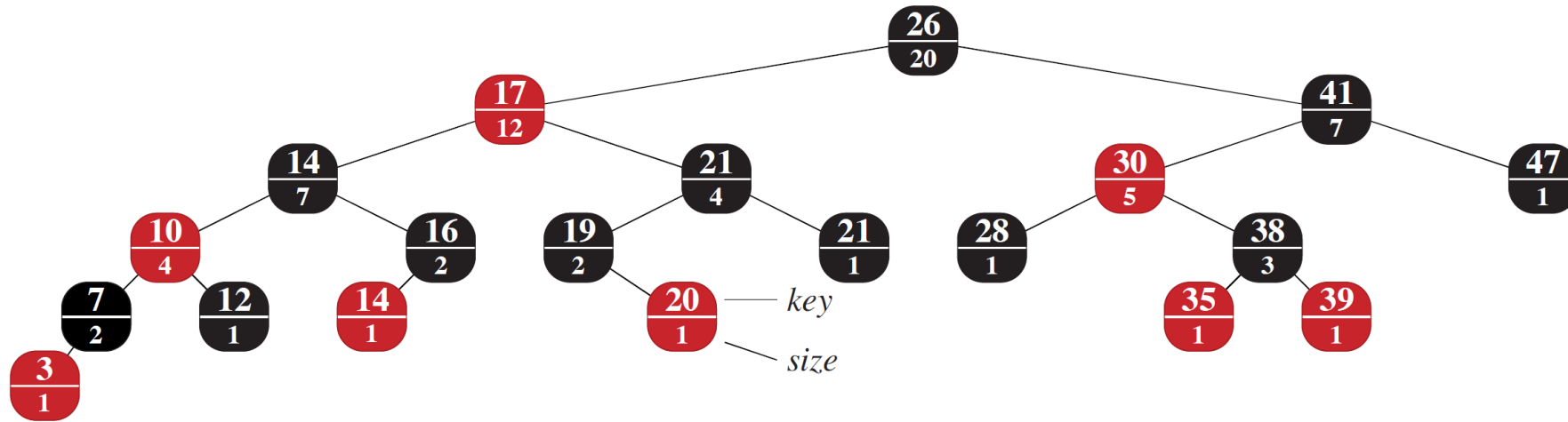
- Insert / delete: update **size** on path to root
- Under **rotations** in **red-black** update **size**

Rank of 28 ?

- a) 13
- b) 14
- 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  $y \neq T.root$ 
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)

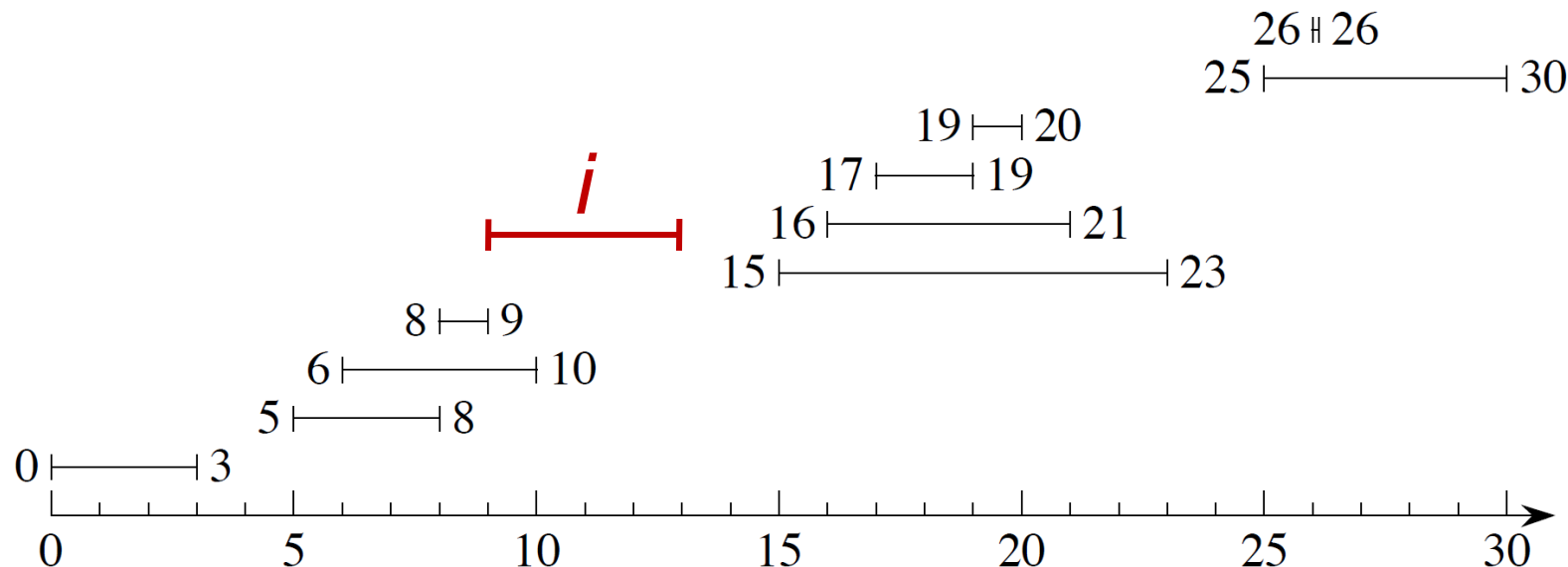
Select(S, i)

Rank(S, x)

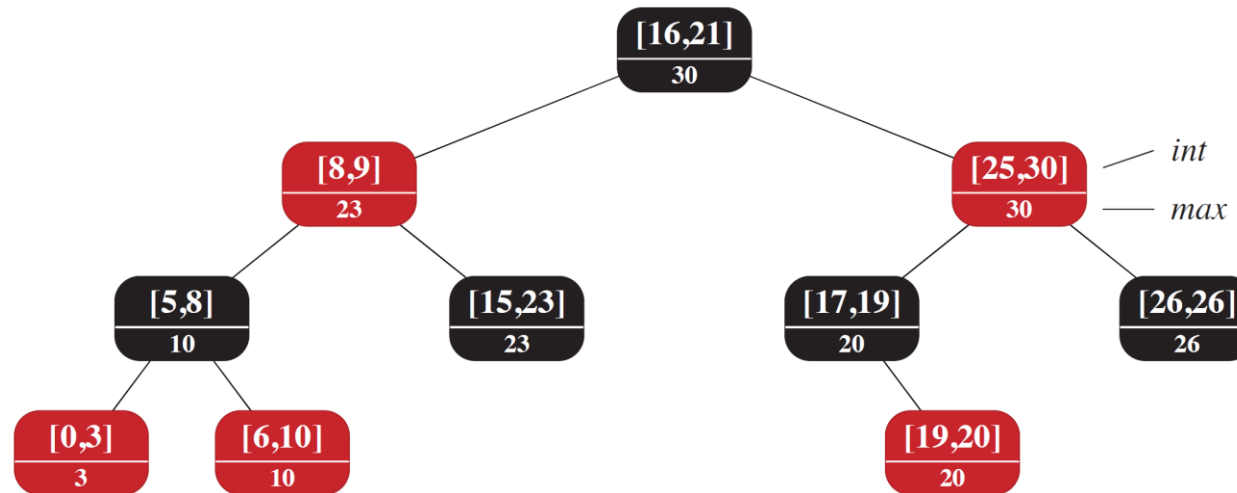
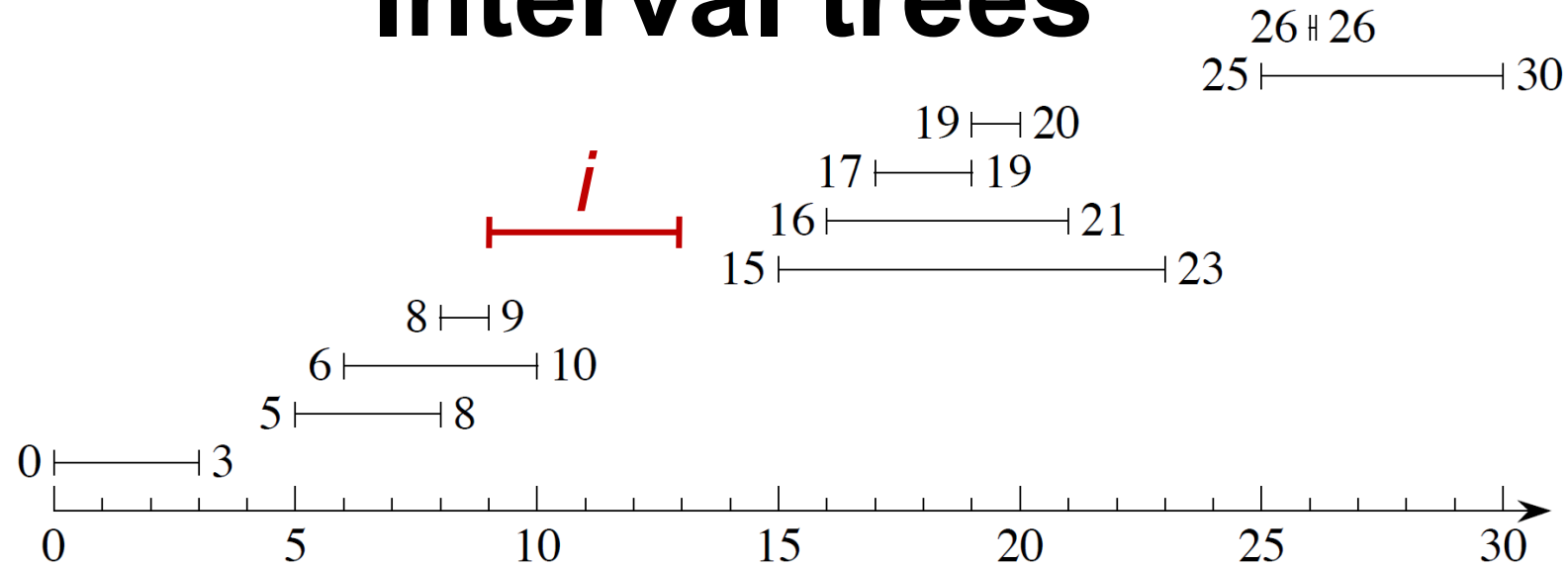
$O(\log n)$

Interval trees

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



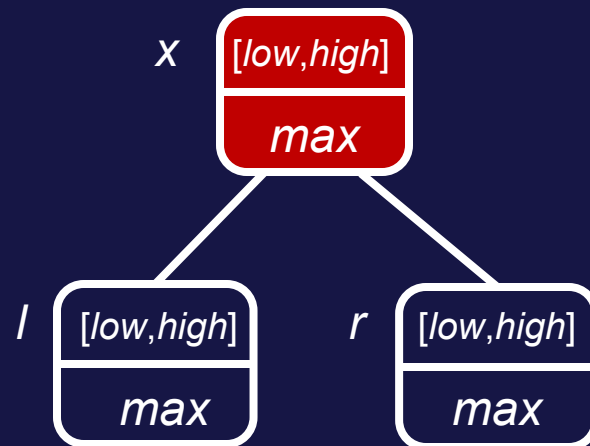
Interval trees



- **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

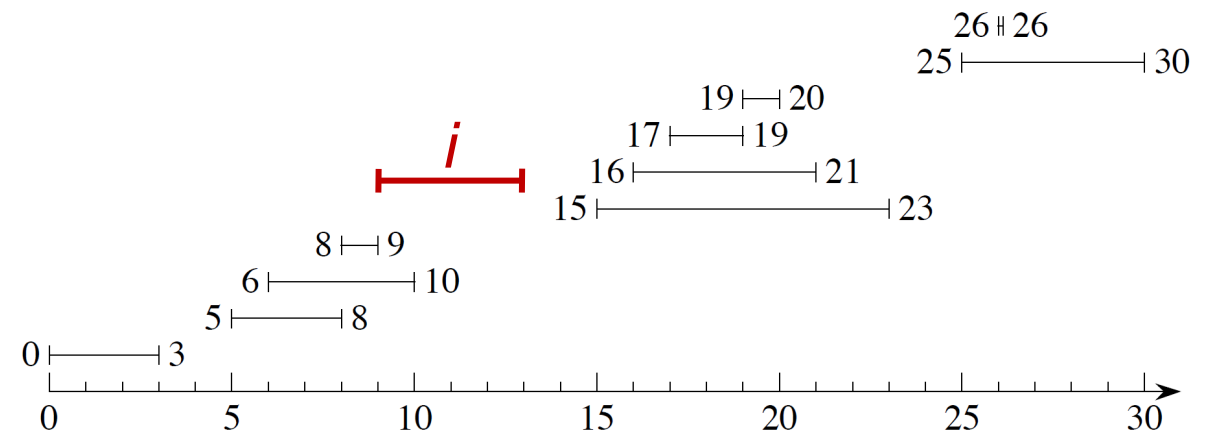
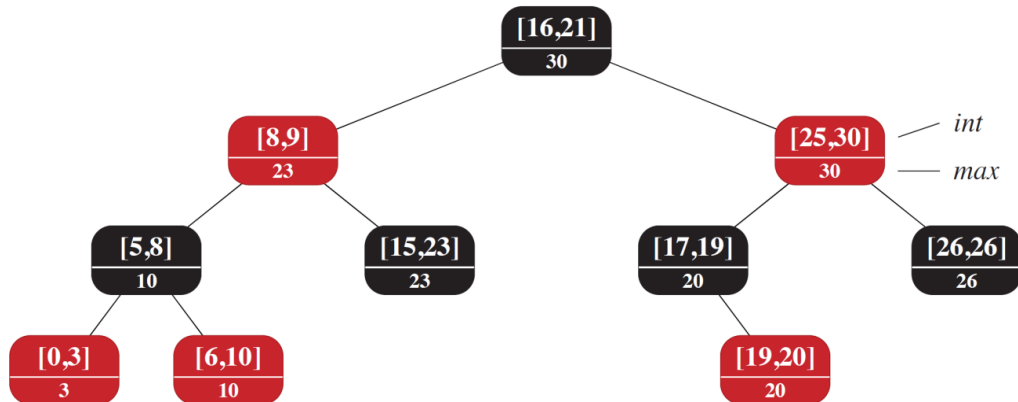


$i.\text{low} \leq x.\text{high}$ and $x.\text{low} \leq i.\text{high}$

INTERVAL-SEARCH(T, i)

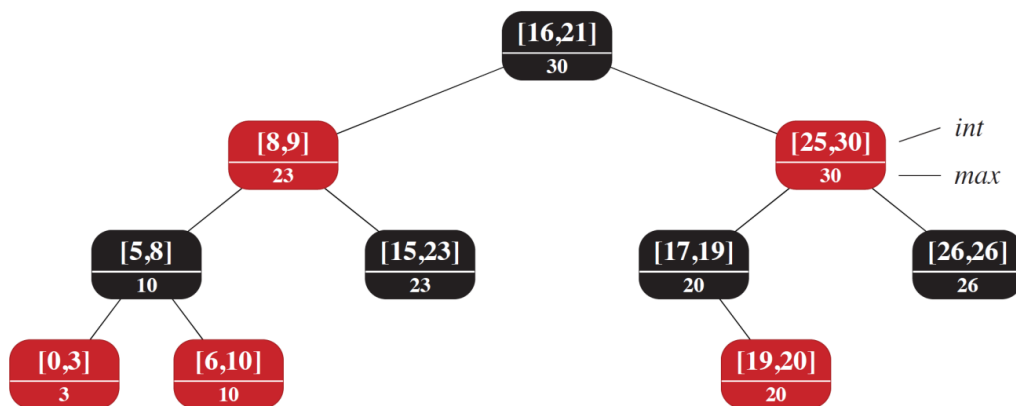
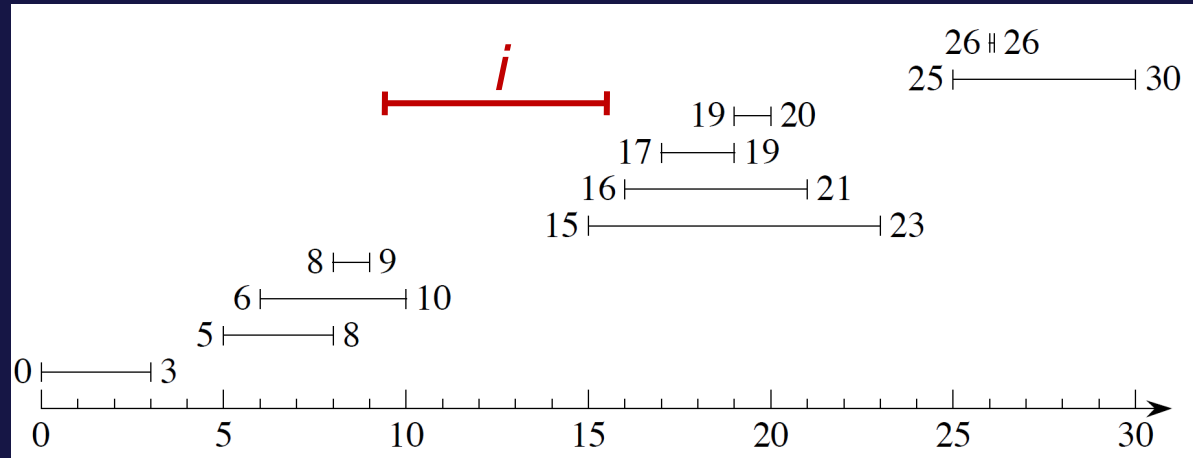
```
1  $x = T.\text{root}$ 
2 while  $x \neq T.\text{nil}$  and  $i$  does not overlap  $x.\text{int}$ 
3   if  $x.\text{left} \neq T.\text{nil}$  and  $x.\text{left}.\text{max} \geq i.\text{low}$ 
4      $x = x.\text{left}$ 
5   else  $x = x.\text{right}$ 
6 return  $x$ 
```

tricky



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

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

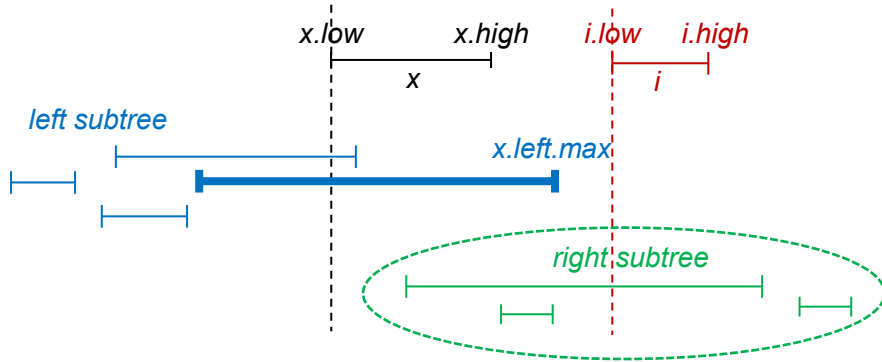


INTERVAL-SEARCH(T, i)

- 1 $x = T.root$
- 2 **while** $x \neq T.nil$ and i does not overlap $x.int$
- 3 **if** $x.left \neq T.nil$ and $x.left.max \geq i.low$
- 4 $x = x.left$
- 5 **else** $x = x.right$
- 6 **return** x

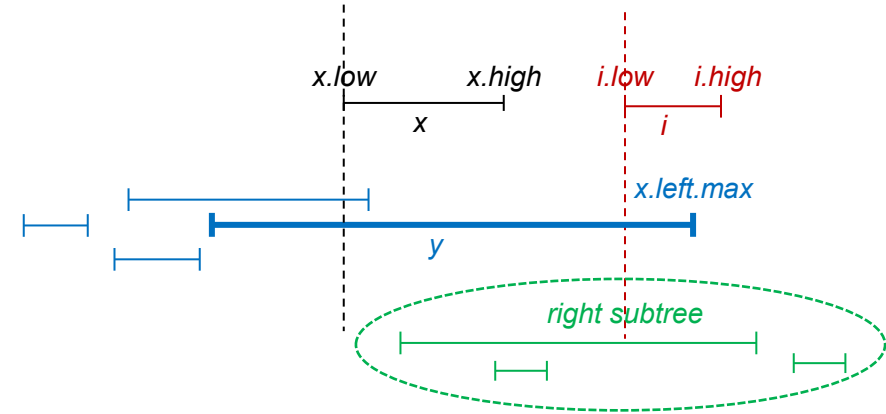
Interval trees – correctness

x left of i and $x.\text{left.max} < i.\text{low}$



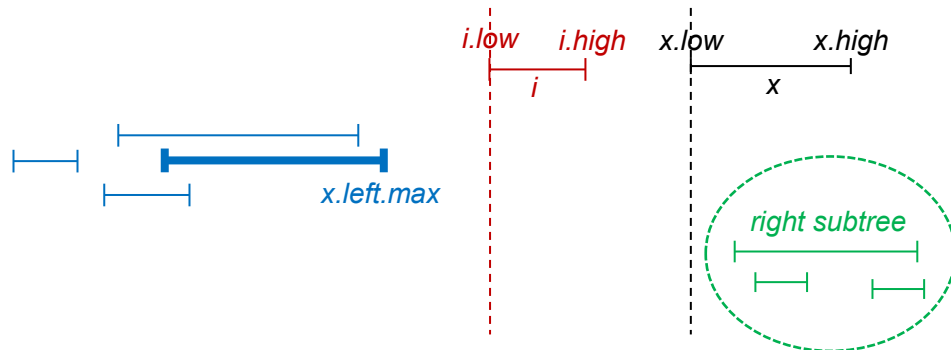
No overlap with i in left subtree, safe to **go right**

x left of i and $x.\text{left.max} \geq i.\text{low}$



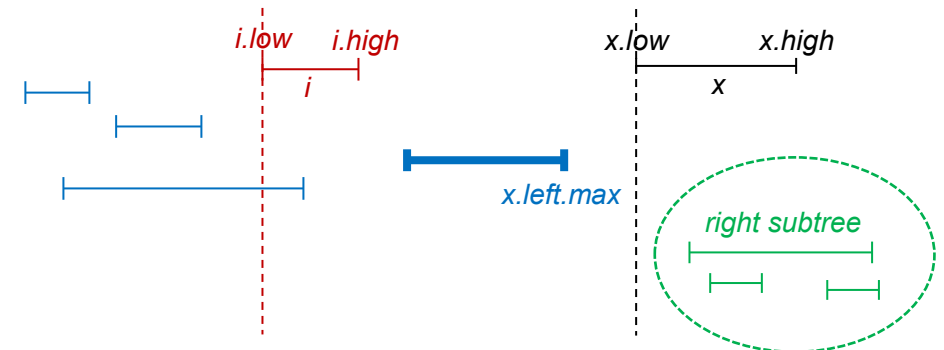
i overlaps with interval y in left subtree, safe to **go left**

i left of x and $x.\text{left.max} < i.\text{low}$



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

i left of x and $x.\text{left.max} \geq i.\text{low}$



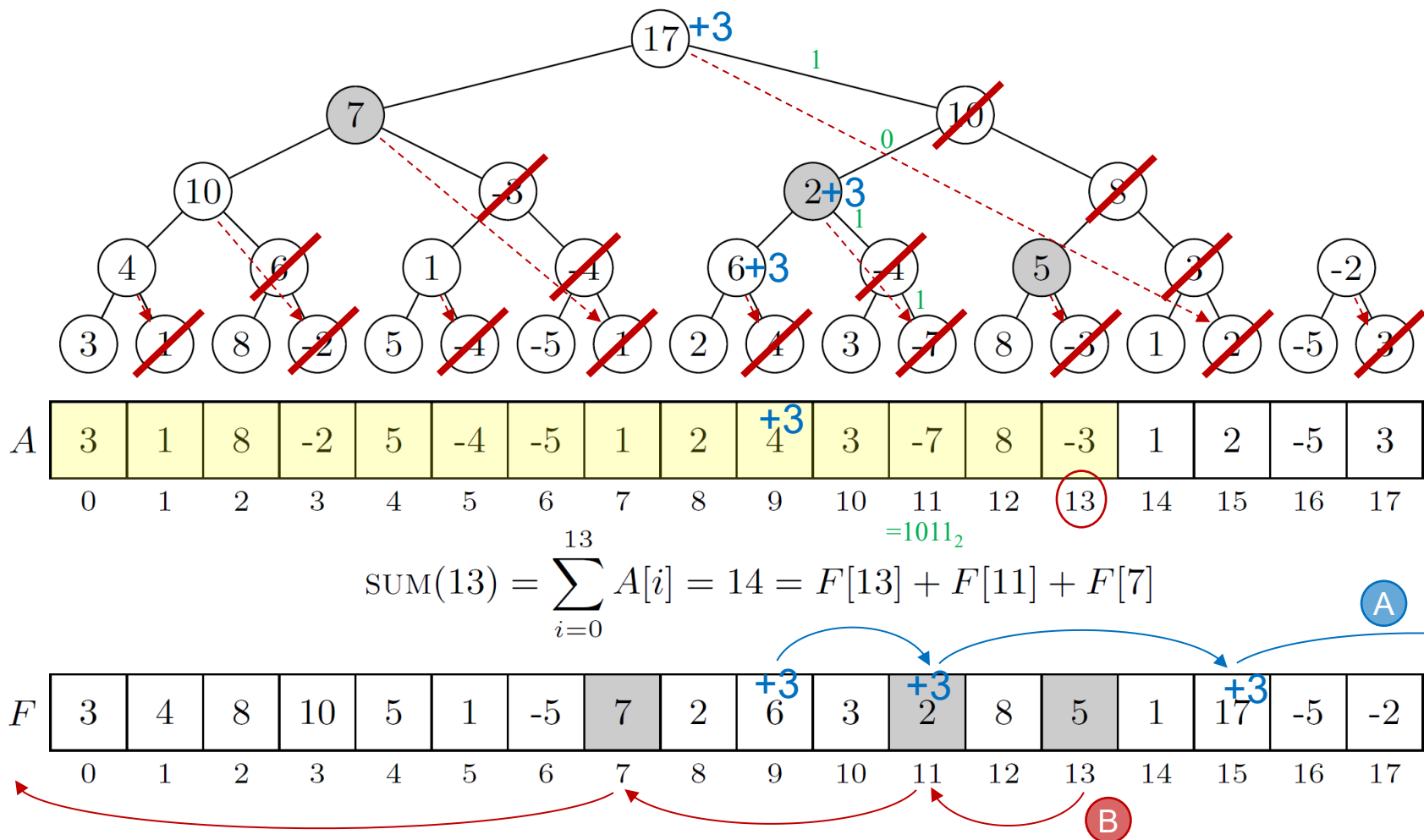
i no overlap with right subtree, safe to **go left**

Interval trees

Insert(S, i)
Delete(S, i) $O(\log n)$
Search(S, i)

Fenwick trees

Dynamic prefix sums in $O(\log n)$



Algorithm INIT(n)

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

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

```
1  while  $i < n$  do
```

$$2 \quad F[i] = F[i] + d$$
$$3 \quad i = i \mid (i + 1)$$

Algorithm SUM(i)

$$1 \quad s = 0$$

```
2  while  $i > 0$  do
```

$$3 \quad s = s + F[i]$$

4 $i = (i \ \& \ (i + 1)) - 1$ B

```
5 return s
```

A	B
9 = 01001 ₂	13 = 1101 ₂
11 = 010 <u>11</u> ₂	11 = 1 <u>011</u> ₂
15 = 01 <u>111</u> ₂	7 = <u>0111</u> ₂
31 = 11111 ₂	-1

Algorithms and Data Structures

Union-find

[CLRS, Chapter 19.1-19.3]

Union-find

MakeSet(x)

Create set $S = \{ x \}$

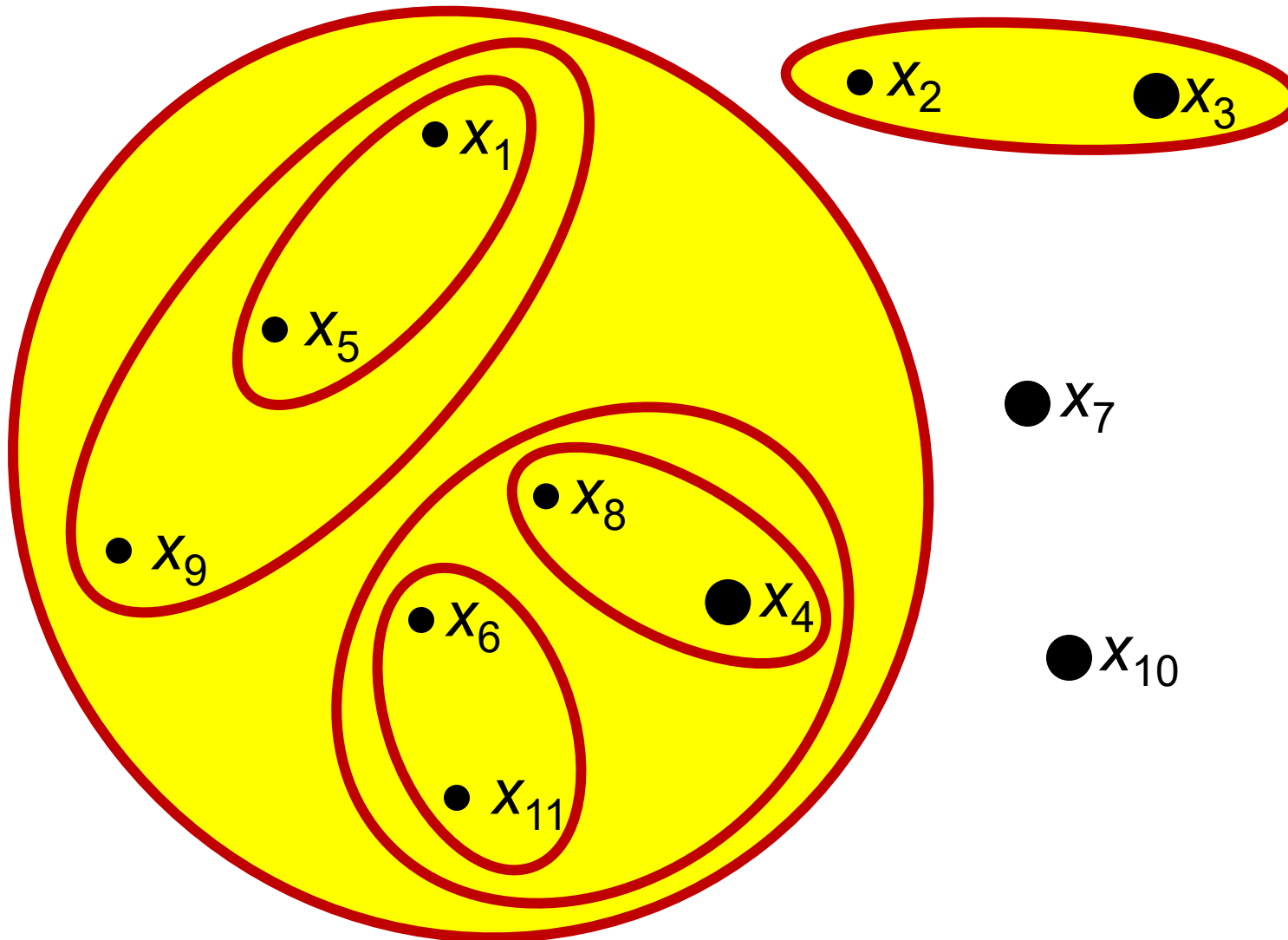
Union(x, y)

Replace $S_x = \{ \dots, x, \dots \}$ and $S_y = \{ \dots, y, \dots \}$ with $S_x \cup S_y$ (assuming x and y are in different sets)

FindSet(x)

Return a *representative* for $S_x = \{ \dots, x, \dots \}$
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)

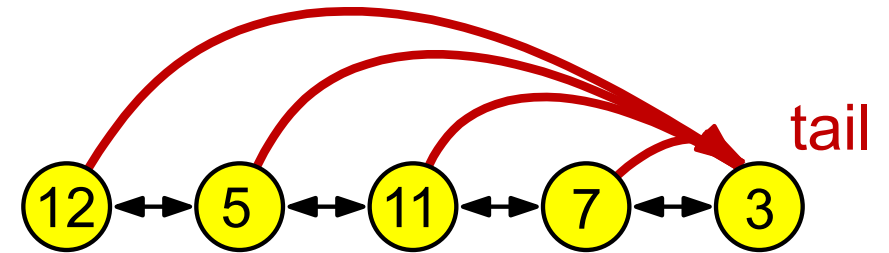
- Set = doubly-linked list
- **MakeSet** = create new node
- **FindSet** = return rightmost node (tail)
- **Union** = concatenate lists



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)

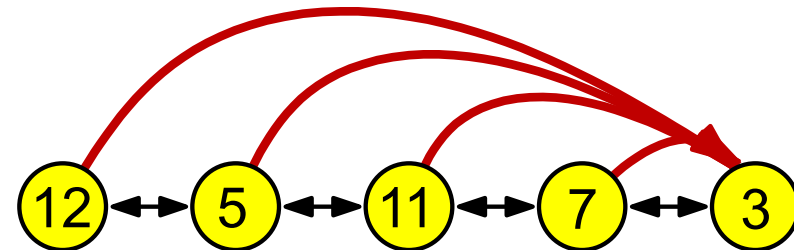
sequence of unions

Theorem

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

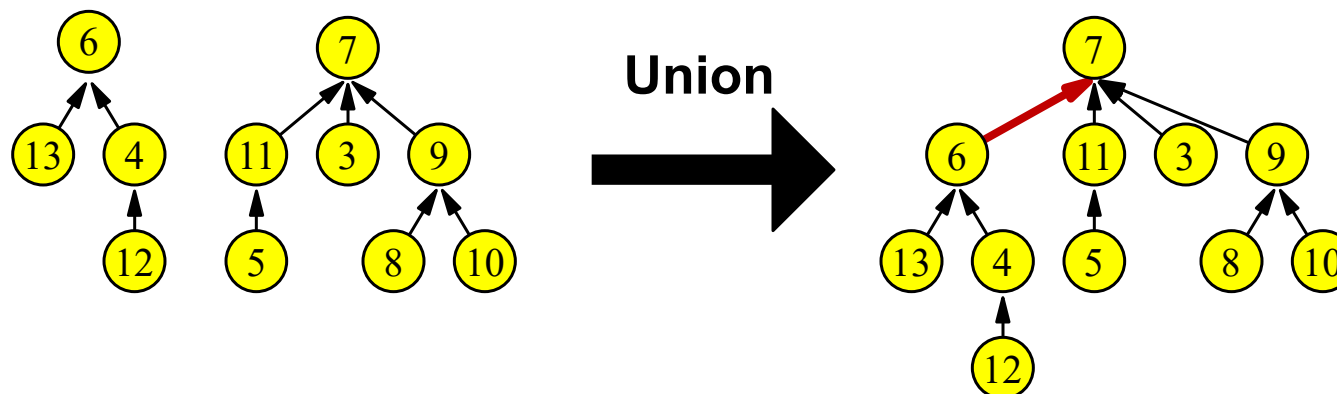
Proof

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)

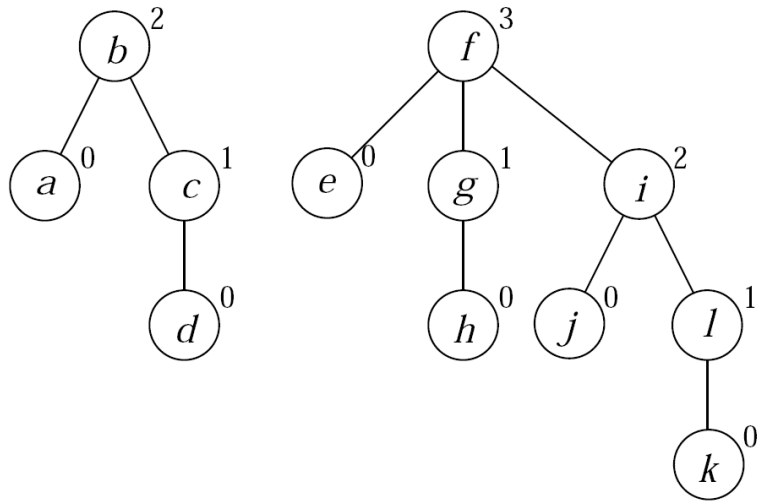


Set representation (III)

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** $x \neq x.p$
- 2 $x.p = \text{FIND-SET}(x.p)$ path compression
- 3 **return** $x.p$

Set representation (III)

analysis linking by rank

Lemma

$$\text{height}[\text{root}] \leq \text{rank}[\text{root}]$$

$$\text{size}[\text{root}] \geq 2^{\text{rank}[\text{root}]}$$

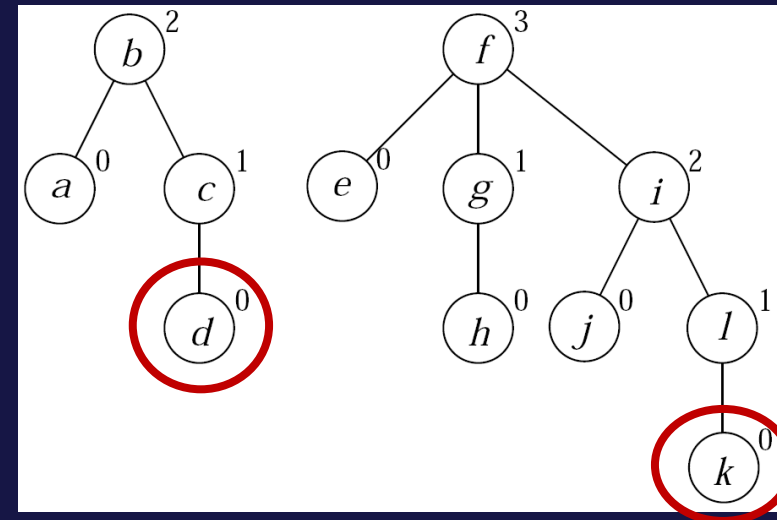
Proof Induction

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

Set representation (III)

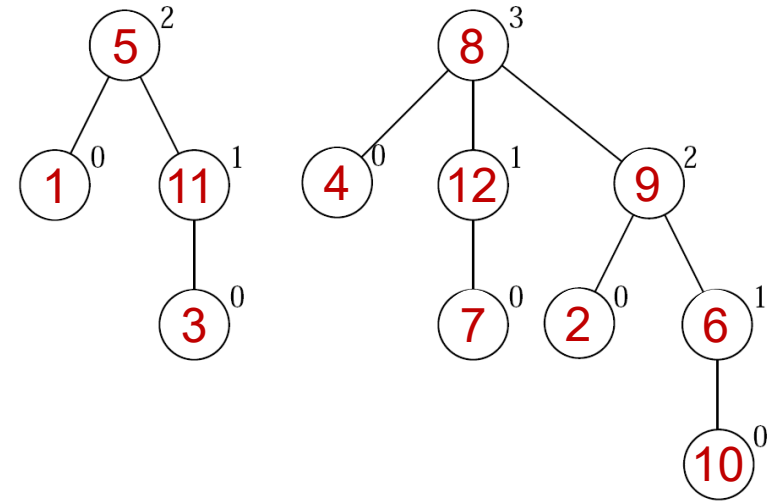
analysis linking-by-rank and path compression

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

Compact representation

- Elements = integers $1, \dots, n$
- *rank* and *p* arrays



	1	2	3	4	5	6	7	8	9	10	11	12
<i>p</i>	5	9	11	8	5	9	12	8	8	6	5	8
<i>rank</i>	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  $root \neq p[root]$  do
3       $root = p[root]$ 
4  while  $p[x] \neq root$  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  $root \neq p[root]$  do
3       $root = p[root]$ 
4  while  $x \neq root$  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

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( $y$ ))
```

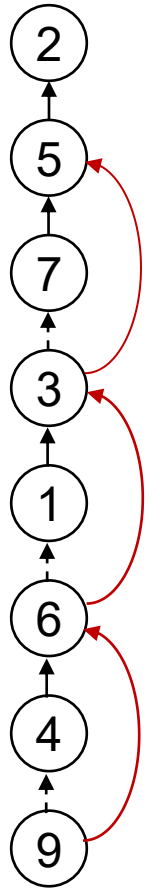
LINK(x, y)

```
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  $x \neq x.p$  do
2       $p[x] = p[p[x]]$ 
3       $x = p[x]$ 
4  return  $x$ 
```

FindSet



- **FindSet** makes 1 pass of path
- Shortcut every second edge
- m operations total time $O(m \cdot \alpha(n))$

Algorithms and Data Structures

Amortized analysis
[CLRS, Chapter 16]

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.)

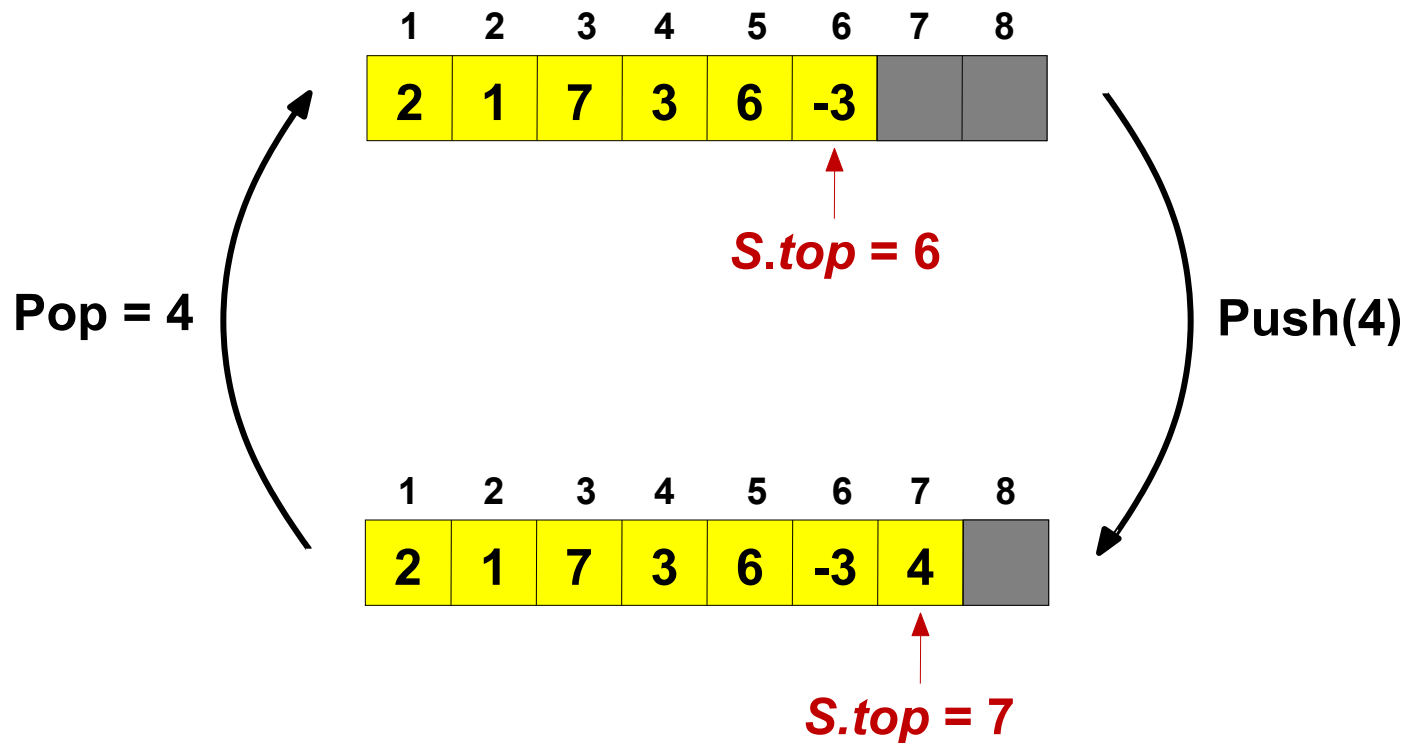
The size, isEmpty, get, set, getFirst, getLast, removeLast, iterator, listIterator, and reversed operations run in constant time. The add, and addLast operations runs in *amortized constant time*, that is, adding n elements requires O(n) time. All of the other operations run in linear time (roughly speaking).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

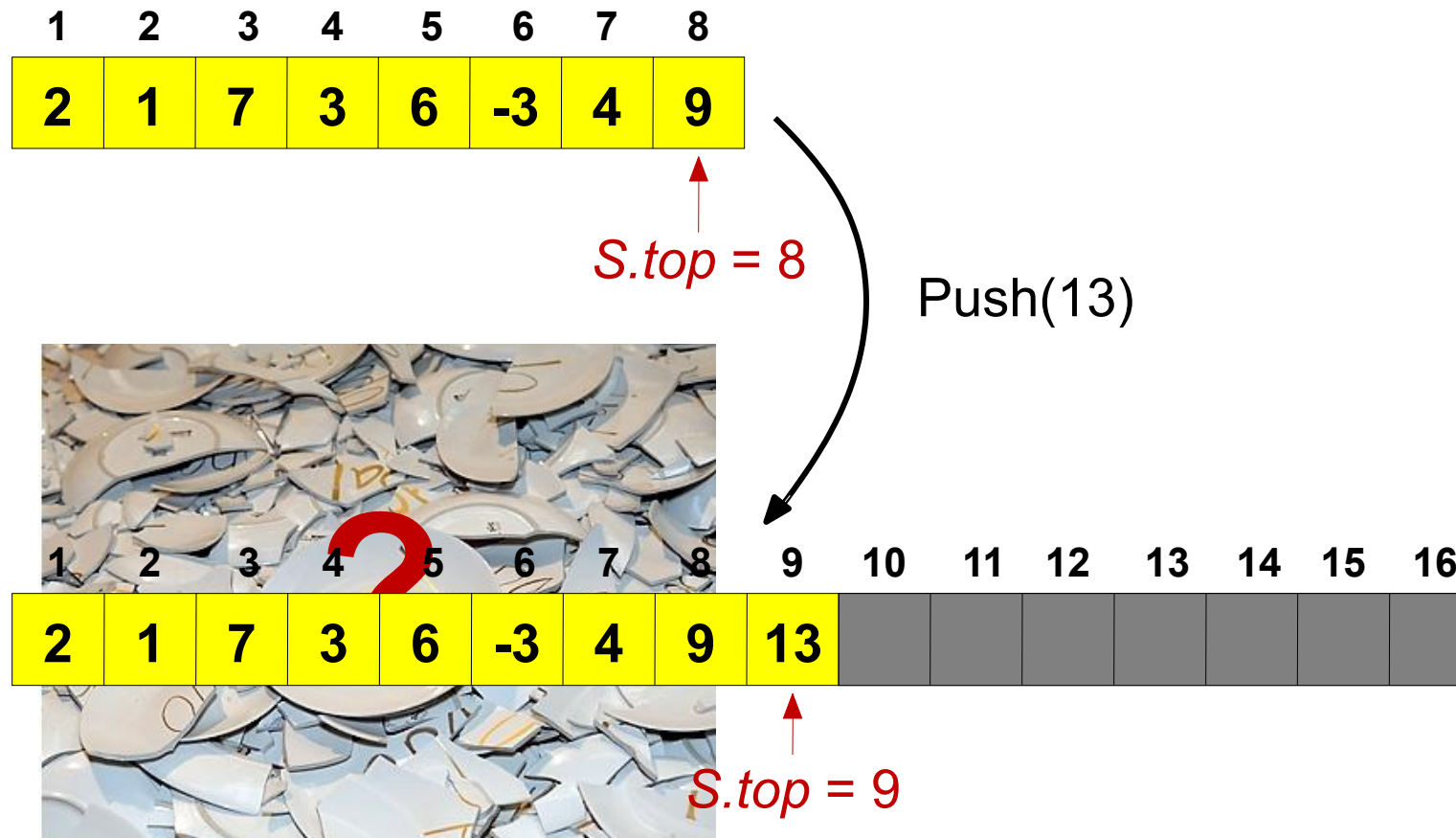
```
1  if  $S.top == S.size$ 
2      error "overflow"
3  else  $S.top = S.top + 1$ 
4       $S[S.top] = x$ 
```

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

1

2

4

8

16

32

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

Halve the array when it is $<1/4$ full

32

16

16

8

8

16

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 $O(n)$

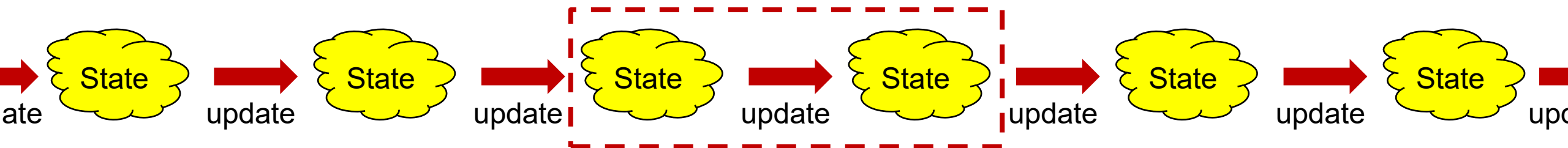
Wish list for analysis technique...

Requirements

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

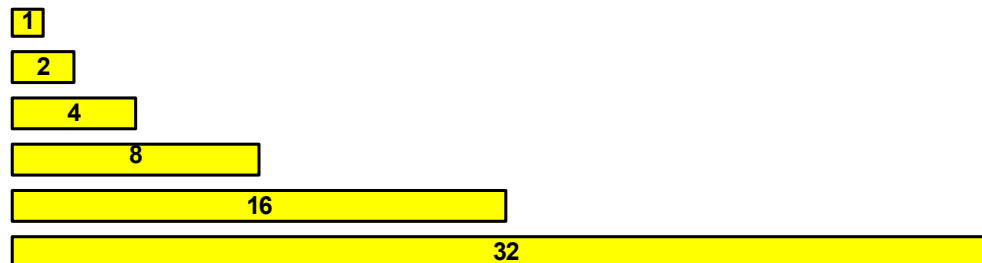
Advantages

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



Intuition

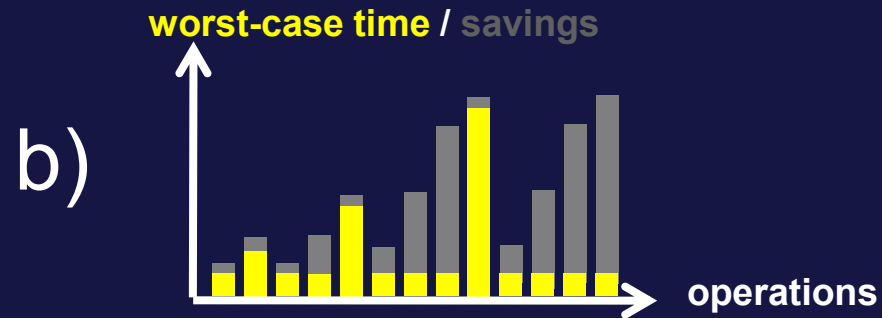
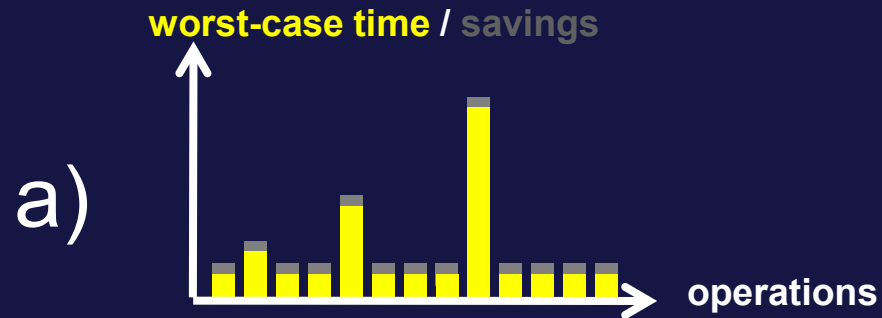
- There are **good/balanced** and **bad/unbalanced** states
- Going from a **bad** to a **good** state is **expensive**
- Requires many operations to get from a **good** to a **bad** state
- The (many) **cheap** operations *pay* slightly more to make later **expensive** operations nearly for **free**



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

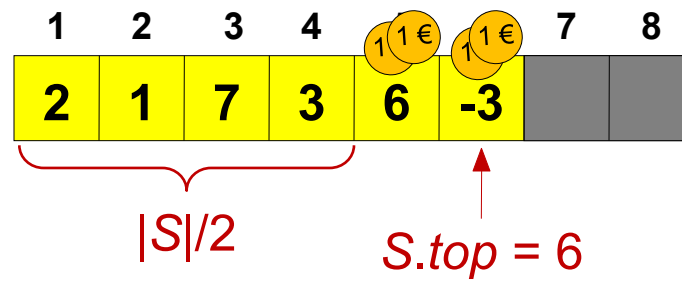
Relationship between worst-case time and savings Φ



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

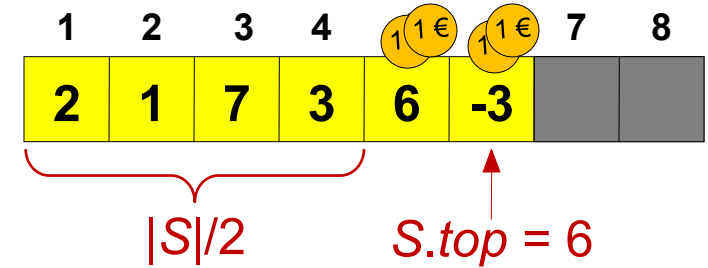


Example – stack

push = amortized 3€

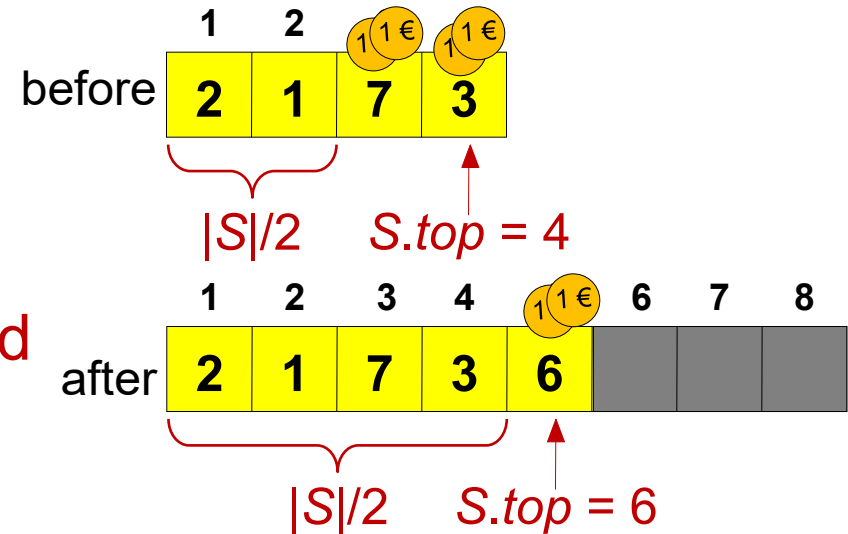
■ Push, no copying

- One new element: 1 €
- $\Phi = ||S|/2 - \text{top}[S]|$ increases by at most 1, i.e., invariant satisfied if we save up 2 €
- Amortized time: $1+2 = 3$ €

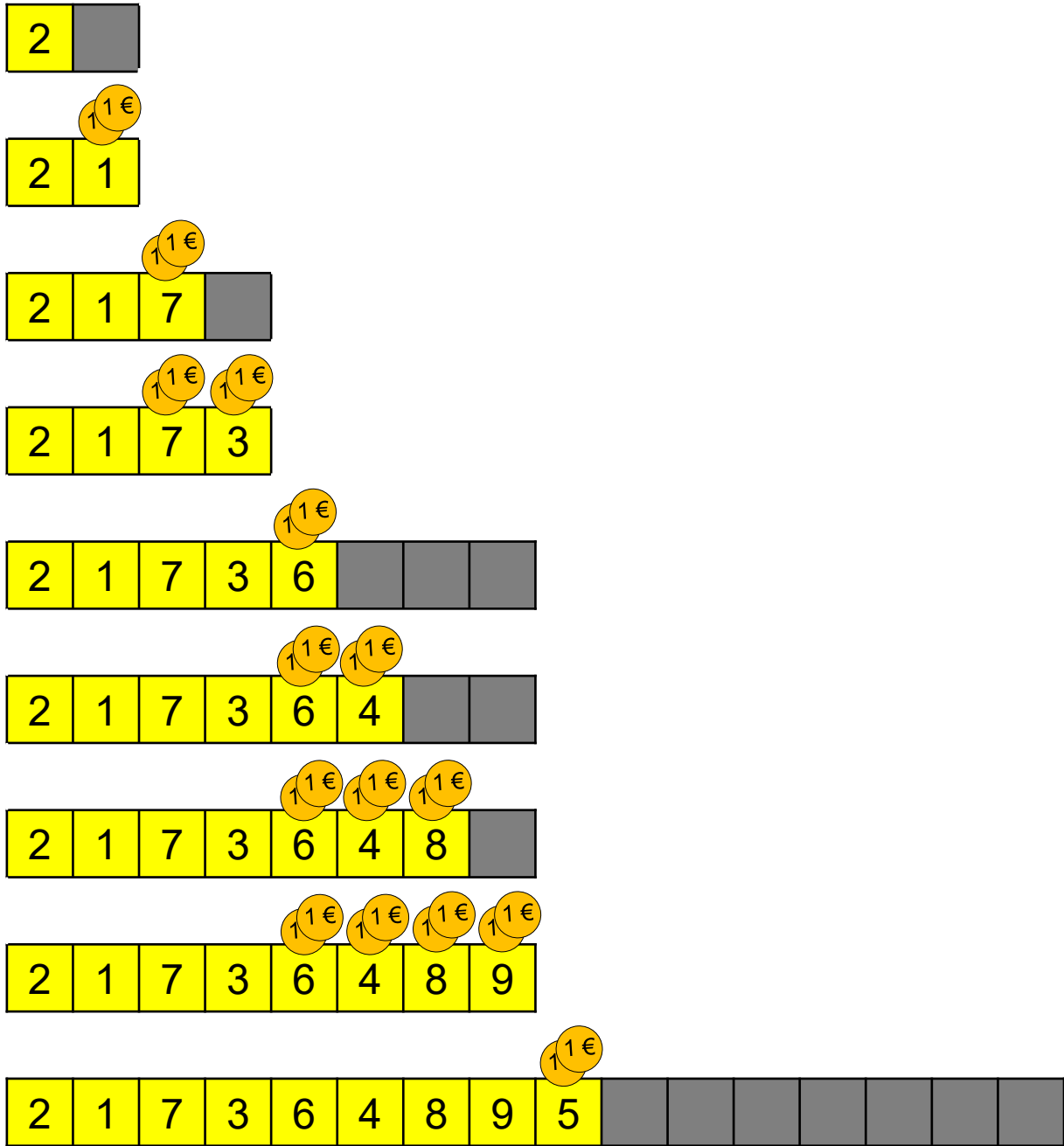


■ Push, with copying

- Copy S: $|S|$ €
- Insert new element: 1 €
- Φ before = $|S|$, Φ after = 2, i.e., $|S|-2$ € released
- Amortized time: $|S|+1-(|S|-2) = 3$ €



$$\text{Invariant : } \Phi = 2 \cdot |S.top - |S|/2|$$



push(1)
push(7)
push(4)
push(6)
push(4)
push(8)
push(9)
push(5)

potential	potential increase	real time	amortized time
0			
	2	1	$2+1 = 3$
2	0	$2+1$	$0+2+1 = 3$
2	2	1	$2+1 = 3$
4	-2	$4+1$	$-2+4+1 = 3$
2	2	1	$2+1 = 3$
4	2	1	$2+1 = 3$
6	2	1	$2+1 = 3$
8	-6	$8+1$	$-6+8+1 = 3$
2			

Binary counter

0000000000000000
0000000000000001 ← 1 bit
0000000000000010 ← 2 bits
0000000000000011 ← 1
0000000000000100 ← 3
0000000000000101 ← 1
0000000000000110 ← 2
0000000000000111 ← 1
0000000000001000 ← 4

...

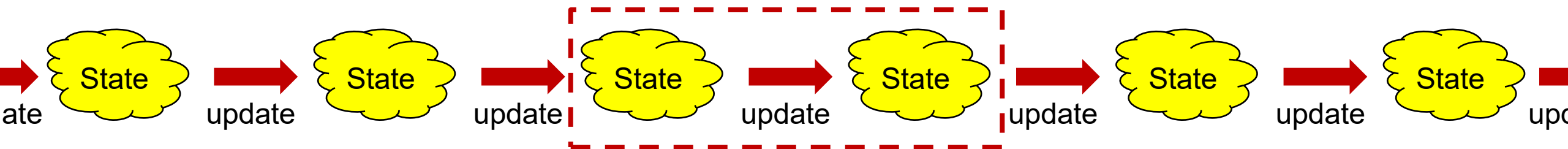
10101110111111
10101111000000 ← 7

**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



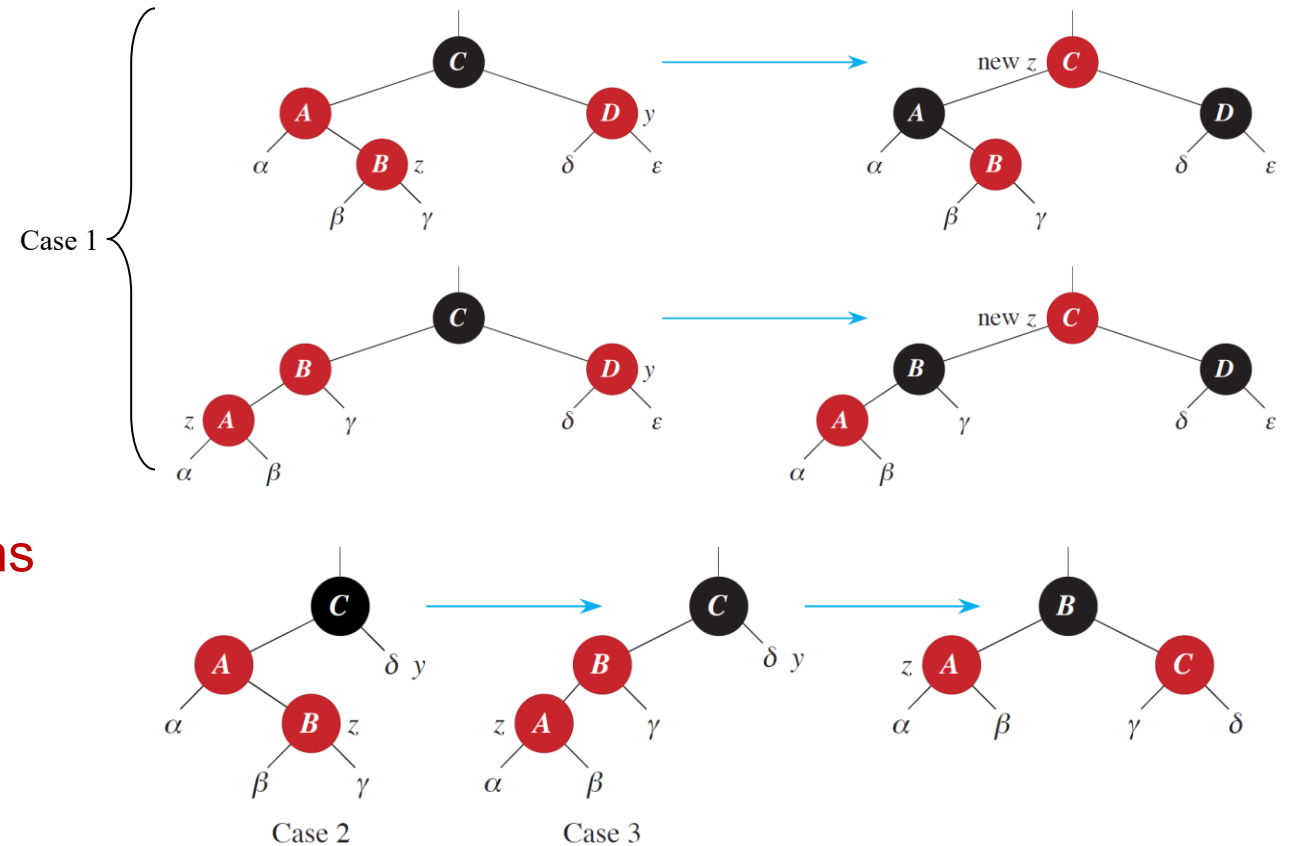
Example – red-black trees

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

transitions = *amortized* $O(1)$

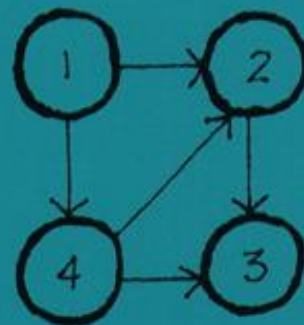
Φ = # red nodes

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

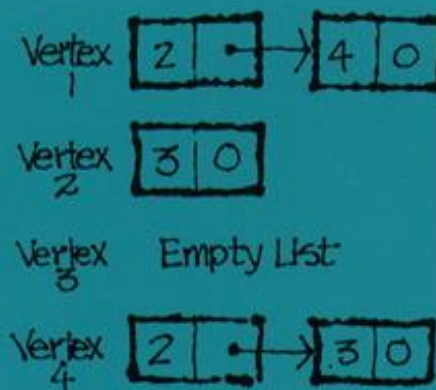


The Design and Analysis of Computer Algorithms

AHO | HOPCROFT | ULLMAN



	1	2	3	4
1	0	1	0	1
2	0	0	1	0
3	0	0	0	0
4	0	1	1	0



	Head	Next
Vertices 1		5
2		7
3		0
4		8
Edges 5	2	6
6	4	0
7	3	0
8	2	9
9	3	0

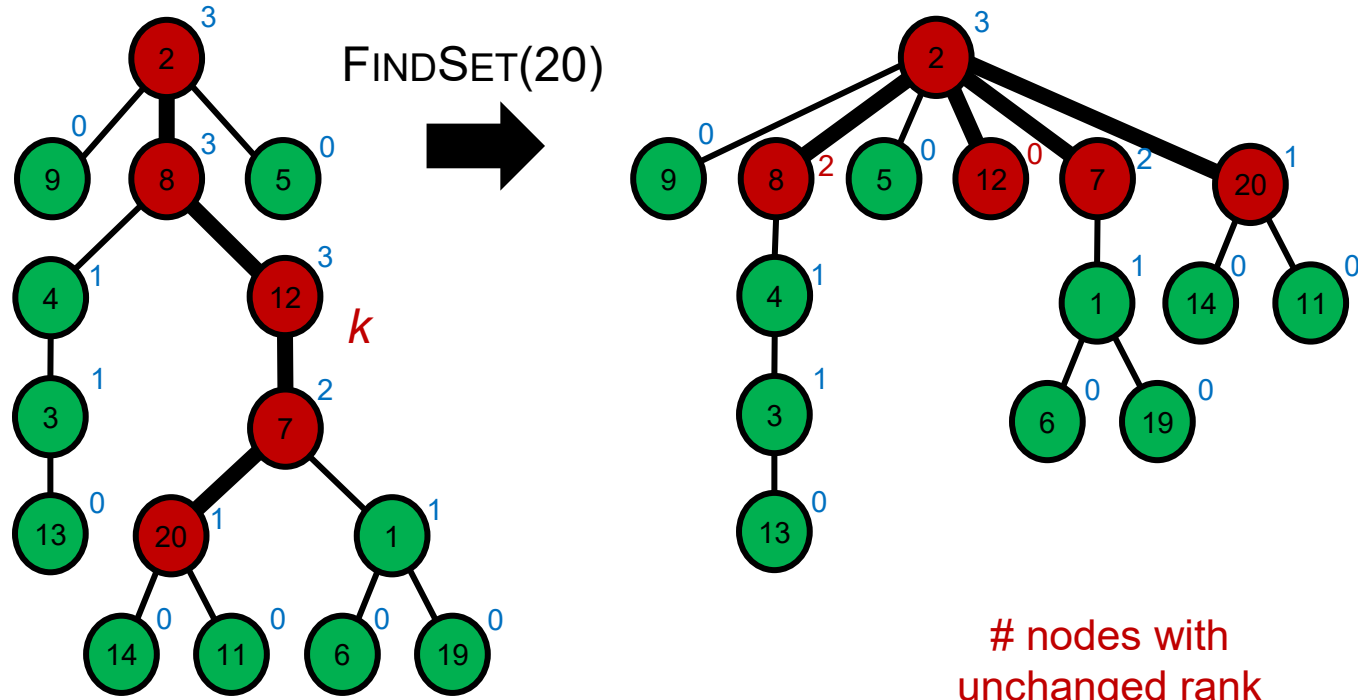
Union-Find with path compression (without rank)

```
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 $\text{rank}(x) = \lfloor \log_2 |T(x)| \rfloor$

Lemma $\text{rank}(x.p) \geq \text{rank}(x)$

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 $< r$

	Real time	$\Delta\Phi$	Amortized time (real time + $\Delta\Phi$)
MAKESET	$O(1)$	0	$O(1)$
LINK	$O(1)$	$\leq \log n$	$O(\log n)$
FINDSET	k	$\leq -k + \underbrace{2 + \log n}_{\text{# nodes with unchanged rank}}$	$O(\log n)$
UNION = 2 x FINDSET + LINK			$O(\log n)$

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

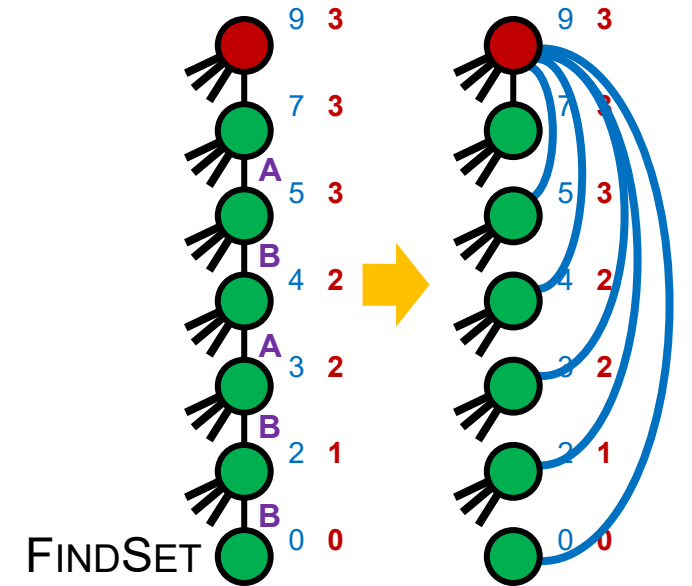
Theorem n MAKESET and m FINDSET and UNION operations take time $O((n + m) \cdot \log^* n)$

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

$$\begin{array}{ll} r_0 = 1 & r_3 = 2^4 = 16 \\ r_1 = 2^1 = 2 & r_4 = 2^{16} = 65536 \\ r_2 = 2^2 = 4 & r_5 = 2^{65536} \end{array}$$

Proof

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



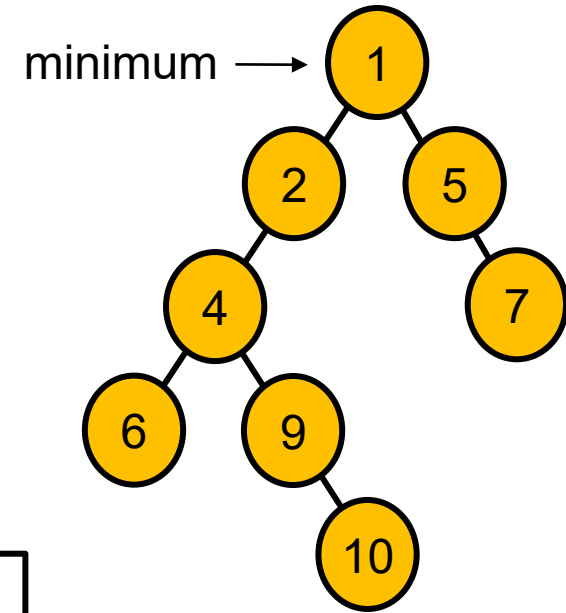
$$O(n + m + \underbrace{m \cdot \log^* n}_{\text{(B) parent is already in different layer}} + \underbrace{\sum_{\text{layer } i} \sum_{r=r_{i-1}+1..r_i} n / 2^r \cdot r_i}_{\substack{\text{(A) parent is in same layer} \\ \text{before parent change}}}) = O(n + m \cdot \log^* n + \sum_{\text{layer } i}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

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

times a node in layer i can have a parent in layer i

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)

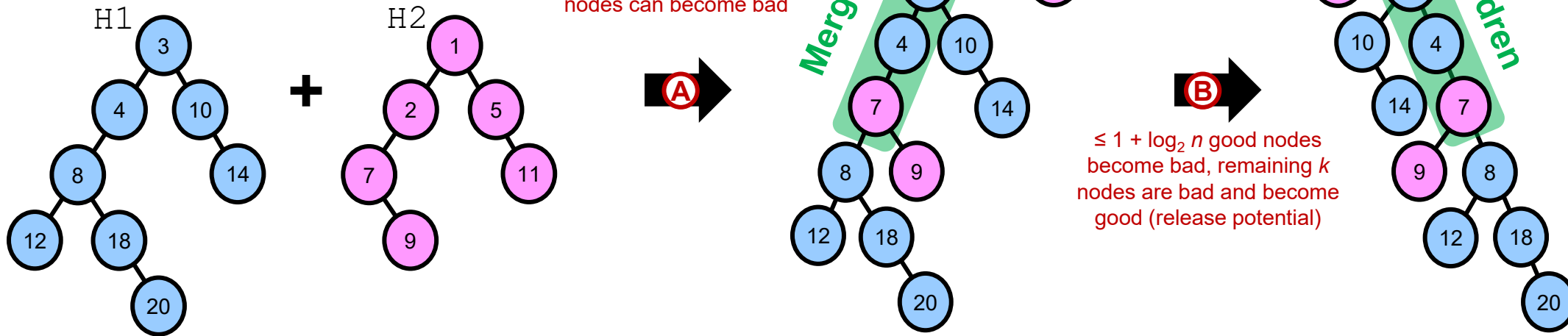
proc DeleteMin(H)
    return Meld(H.left, H.right)
```

Meld next slide

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 : $|v.\text{left}| \leq |v| / 2$

Lemma # good nodes on left path $\leq 1 + \log_2 n$

Potential $\Phi = \# \text{ bad nodes}$

Meld **amortized time**

$$\begin{aligned}
 & \underbrace{\Delta\Phi \text{ from (A)}}_{\leq 2 + 2 \cdot \log_2 n} + \underbrace{\Delta\Phi \text{ from (B)}}_{1 + \log_2 n - k} + \underbrace{\text{real time}}_{k + 1 + \log_2 n} = O(\log n) \\
 & \Rightarrow \text{All operations amortized time } O(\log n)
 \end{aligned}$$

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

$O_A \equiv$ amortized time

Fibonacci heaps

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

Splay trees

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

Algorithms and Data Structures

Divide and conquer

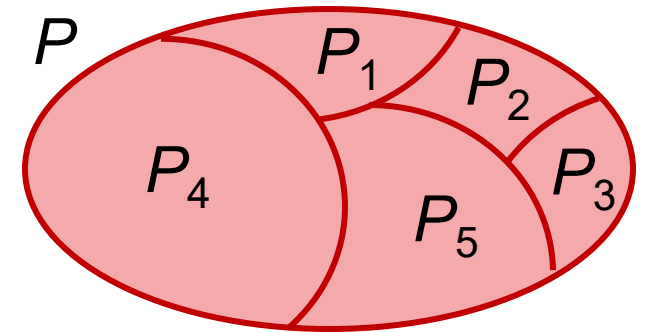
[CLRS, Chapter 2.3, 4.1-4.5, Problem 30.1.c]

Divide and conquer

Algorithm design technique

Applicable to some problems (but far from all)

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



Example – Merge-Sort

MERGE-SORT(A, p, r)

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

return

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

② Solve recursively

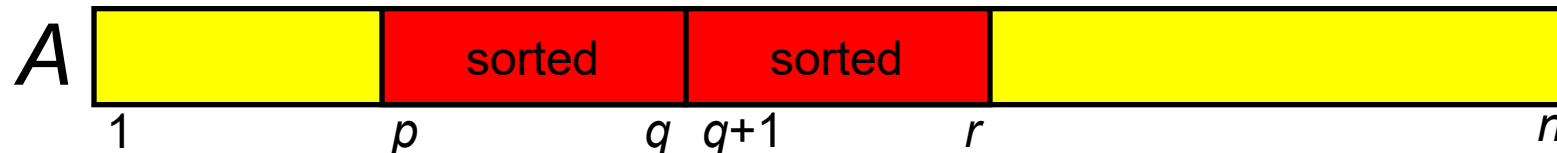
MERGE-SORT(A, p, q)

MERGE-SORT($A, q + 1, r$)

③ Combine

MERGE(A, p, q, r)

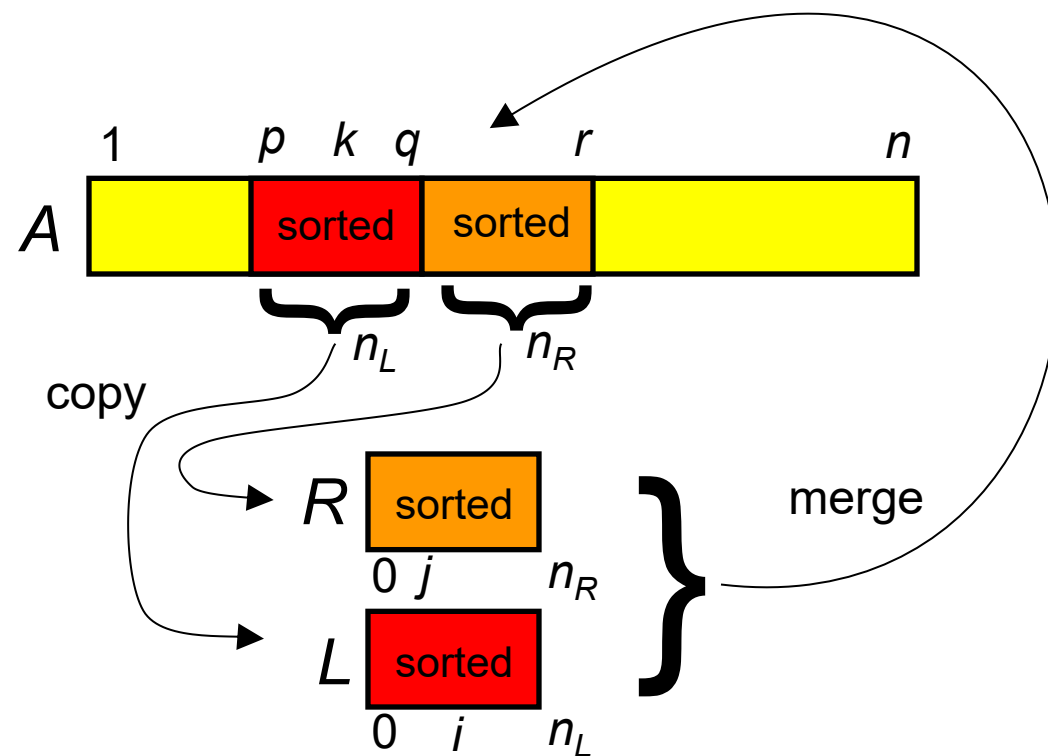
① Two smaller subproblems



MERGE(A, p, q, r)

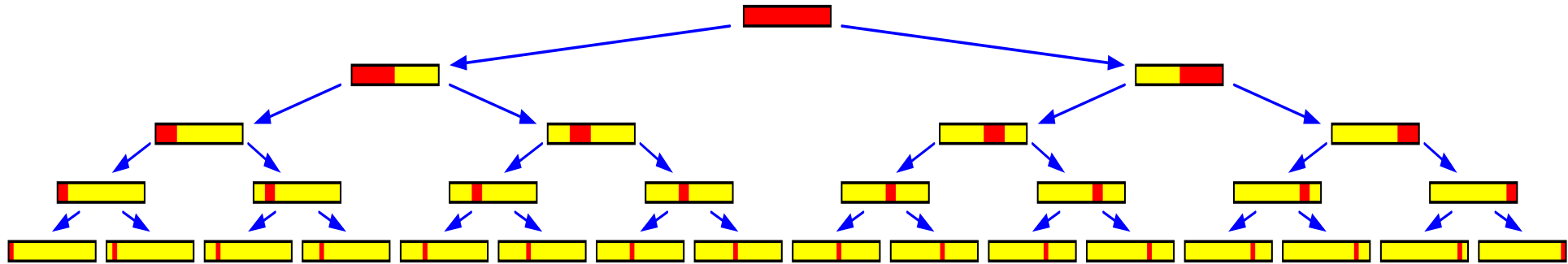
```
1   $n_L = q - p + 1$       // length of  $A[p : q]$ 
2   $n_R = r - q$           // length of  $A[q + 1 : r]$ 
3  let  $L[0 : n_L - 1]$  and  $R[0 : n_R - 1]$  be new arrays
4  for  $i = 0$  to  $n_L - 1$  // copy  $A[p : q]$  into  $L[0 : n_L - 1]$ 
5       $L[i] = A[p + i]$ 
6  for  $j = 0$  to  $n_R - 1$  // 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,
12 // copy the smallest unmerged element back into  $A[p : r]$ .
13 while  $i < n_L$  and  $j < n_R$ 
14     if  $L[i] \leq R[j]$ 
15          $A[k] = L[i]$ 
16          $i = i + 1$ 
17     else  $A[k] = R[j]$ 
18          $j = j + 1$ 
19      $k = k + 1$ 
20 // Having gone through one of  $L$  and  $R$  entirely, copy the
21 // remainder of the other to the end of  $A[p : r]$ .
22 while  $i < n_L$ 
23      $A[k] = L[i]$ 
24      $i = i + 1$ 
25      $k = k + 1$ 
26 while  $j < n_R$ 
27      $A[k] = R[j]$ 
28      $j = j + 1$ 
29      $k = k + 1$ 
```

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

$$\begin{array}{ll} T(n) = a & \text{if } n \leq c \\ T(n) = 2 \cdot T(n / 2) + a \cdot n & \text{otherwise} \end{array}$$

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



For $n \geq 3 \dots$

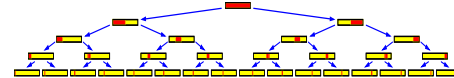
- a) $T(n) = 3 \cdot T(n) + a \cdot n^3$
- b) $T(n) = 3 \cdot T(n) + a \cdot n$
- c) $T(n) = 3 \cdot T(n/3) + a \cdot n$
- d) $T(n) = 3 \cdot T(n/2) + a \cdot n$
- e) Don't know

$\dots$ and $T(n) = c$ for $n < 3$

Solving a recurrence relation

$$\begin{array}{ll} T(n) = a & \text{if } n \leq c \\ T(n) = 2 \cdot T(n / 2) + a \cdot n & \text{otherwise} \end{array}$$

- 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 ?

$$\begin{aligned} T(n) &= 4 \cdot T(n / 2) + n^3 && \text{for } n > 1 \\ T(n) &= c && \text{for } n = 1 \end{aligned}$$

- 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 ?

$$\begin{aligned} T(n) &= 4 \cdot T(n / 2) + n && \text{for } n > 1 \\ T(n) &= c && \text{for } n = 1 \end{aligned}$$

- 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 ?

$$\begin{aligned} T(n) &= 5 \cdot T(n / 4) + n^2 && \text{for } n > 1 \\ T(n) &= c && \text{for } n = 1 \end{aligned}$$

- 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 ?

$$\begin{aligned} T(n) &= 9 \cdot T(n / 3) + n^2 && \text{for } n > 1 \\ T(n) &= c && \text{for } n = 1 \end{aligned}$$

- 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 ?

$$T(n) = 7 \cdot T(n/2) + c \cdot n^2 \quad \text{for } n \geq 2$$

$$T(n) = c \quad \text{for } n = 1$$

- 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

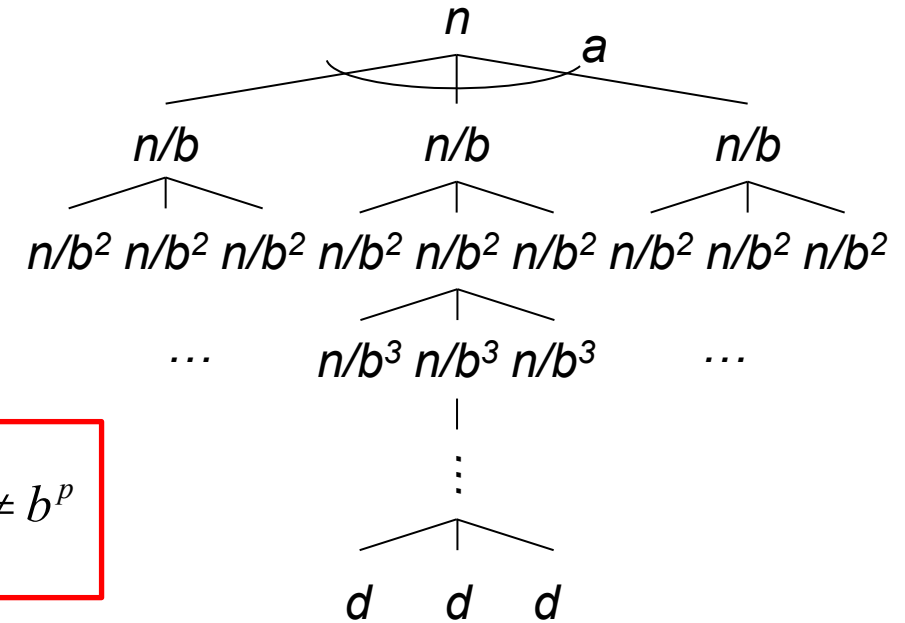
Depth of recursion ?

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

- a) $\log n$
- b) $\log_b n$
- c) $\log_b (n/d)$
- d) $\log_d (n/b)$
- e) Don't know

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	$a^i \cdot c \cdot (n/b^i)^p$	$c \cdot a^{\log_b (n/d)}$

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



$$T(n) = \Theta \left(\underbrace{c \cdot a^{\log_b (n/d)}}_{\text{(bottom of recursion)}} + \sum_{i=0}^{\log_b (n/d)-1} \underbrace{a^i \cdot c \cdot (n/b^i)^p}_{\text{(level } i = 0..\log_b (n/d) - 1)} \right)$$

$$= \Theta \left(c \cdot a^{\log_b n} + c \cdot n^p \cdot \sum_{i=0}^{\log_b n-1} (a/b^p)^i \right) \leftarrow \boxed{\frac{(a/b^p)^{\log_b n} - 1}{a/b^p - 1} \text{ for } a \neq b^p}$$

$$= \Theta \left(n^{\log_b a} + n^p \cdot \begin{cases} 1 & \text{for } a < b^p \\ \log n & \text{for } a = b^p \\ (a/b^p)^{\log_b n} & \text{for } a > b^p \end{cases} \right) = \Theta \left(\begin{cases} n^p & \text{for } a < b^p \\ n^p \cdot \log n & \text{for } a = b^p \\ n^{\log_b a} & \text{for } a > b^p \end{cases} \right)$$

$$\boxed{a^{\log_b n} = n^{\log_b a}}$$

$$\boxed{(b^p)^{\log_b n} = n^p}$$

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

$$I = \begin{array}{|c|c|} \hline I_h & I_l \end{array} \quad J = \begin{array}{|c|c|} \hline J_h & J_l \end{array}$$

$\overbrace{\hspace{1.5cm}}^{n/2}$

$$I \cdot J = I_h \cdot J_h \cdot 2^n + (I_h \cdot J_l + I_l \cdot J_h) \cdot 2^{n/2} + I_l \cdot J_l$$

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 $n = 1$

Multiplication of long integers

[CLRS, Problem 30.1.c]

$$I = \begin{array}{|c|c|} \hline I_h & I_l \\ \hline \end{array} \quad J = \begin{array}{|c|c|} \hline J_h & J_l \\ \hline \end{array}$$

$\overbrace{\hspace{1.5cm}}^{n/2}$

- I and J both n -bit integers
- Naïve implementation $O(n^2)$ bit operations
- $I \cdot J = I_h \cdot J_h \cdot 2^n + ((I_h - I_l) \cdot (J_l - J_h) + I_l \cdot J_l + I_h \cdot J_h) \cdot 2^{n/2} + I_l \cdot J_l$

$$T(n) = 3 \cdot T(n/2) + c \cdot n \quad \text{for } n \geq 2$$

$$T(n) = c \quad \text{for } n = 1$$

- $T(n) = O(n^{\log_2 3}) = O(n^{1.58})$

Multiplication of long integers

+4000 years	$O(n^2)$
Divide and conquer Karatsuba 1960	$O(n^{\log_2 3})$
Schönhage-Strassen 1971	$O(n \cdot \log n \cdot \log \log n)$
<u>Fürer 2007</u> <u>Harvey, Hoeven 2018</u> <u>Harvey, Hoeven 2019</u>	$O(n \cdot \log n \cdot 2^{O(\log^* n)})$ $O(n \cdot \log n \cdot 2^{2 \cdot \log^* n})$ $O(n \cdot \log n)$

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](https://doi.org/10.4007/annals.2021.193.2.4)

Matrices

$$\begin{array}{c}
 \text{represents linear} \\
 \text{transformations } \mathbb{R}^3 \rightarrow \mathbb{R}^4
 \end{array}
 \begin{array}{c}
 m = 3 \\
 \text{columns}
 \end{array}
 \begin{array}{c}
 n = 4 \\
 \text{rows}
 \end{array}
 \begin{pmatrix} 3 & 2 & -1 \\ 5 & 0 & 4 \\ 7 & -2 & 0 \\ 1 & 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3x_1 + 2x_2 - x_3 \\ 5x_1 + 4x_3 \\ 7x_1 - 2x_2 \\ x_1 + 3x_2 + 2x_3 \end{pmatrix}$$

$n \times m$ 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               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
  
```

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 $n \times n$ matrix $n \times n$ matrix

$$\begin{aligned} I &= A \cdot E + B \cdot G \\ J &= A \cdot F + B \cdot H \\ K &= C \cdot E + D \cdot G \\ L &= C \cdot F + D \cdot H \end{aligned}$$

- $A, B, \dots, K, L$ are $n/2 \times n/2$ -matrices
- I, J, K, L can be computed using **8 recurrence relation multiplikations** and **4 matrix additions** on $n/2 \times n/2$ -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

$$\begin{pmatrix} I & J \\ K & L \end{pmatrix} = \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} E & F \\ G & H \end{pmatrix}$$

$$\begin{aligned} I &= S_5 + S_6 + S_4 - S_2 \\ &= (A + D)(E + H) + (B - D)(G + H) + D(G - E) - (A + B)H \\ &= AE + DE + AH + DH + BG - DG + BH - DH + DG - DE - AH - BH \\ &= AE + BG. \end{aligned}$$

$$\begin{aligned} J &= S_1 + S_2 \\ &= A(F - H) + (A + B)H \\ &= AF - AH + AH + BH \\ &= AF + BH. \end{aligned}$$

$$\begin{aligned} K &= S_3 + S_4 \\ &= (C + D)E + D(G - E) \\ &= CE + DE + DG - DE \\ &= CE + DG. \end{aligned}$$

$$\begin{aligned} L &= S_1 - S_7 - S_3 + S_5 \\ &= A(F - H) - (A - C)(E + F) - (C + D)E + (A + D)(E + H) \\ &= AF - AH - AE + CE - AF + CF - CE - DE + AE + DE + AH + DH \\ &= CF + DH. \end{aligned}$$

$$\begin{aligned} S_1 &= A \cdot (F - H) \\ S_2 &= (A + B) \cdot H \\ S_3 &= (C + D) \cdot E \\ S_4 &= D \cdot (G - E) \\ S_5 &= (A + D) \cdot (E + H) \\ S_6 &= (B - D) \cdot (G + H) \\ S_7 &= (A - C) \cdot (E + F) \end{aligned}$$

7 recursive multiplications

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

MATHEMATICIANS ARE WEIRD

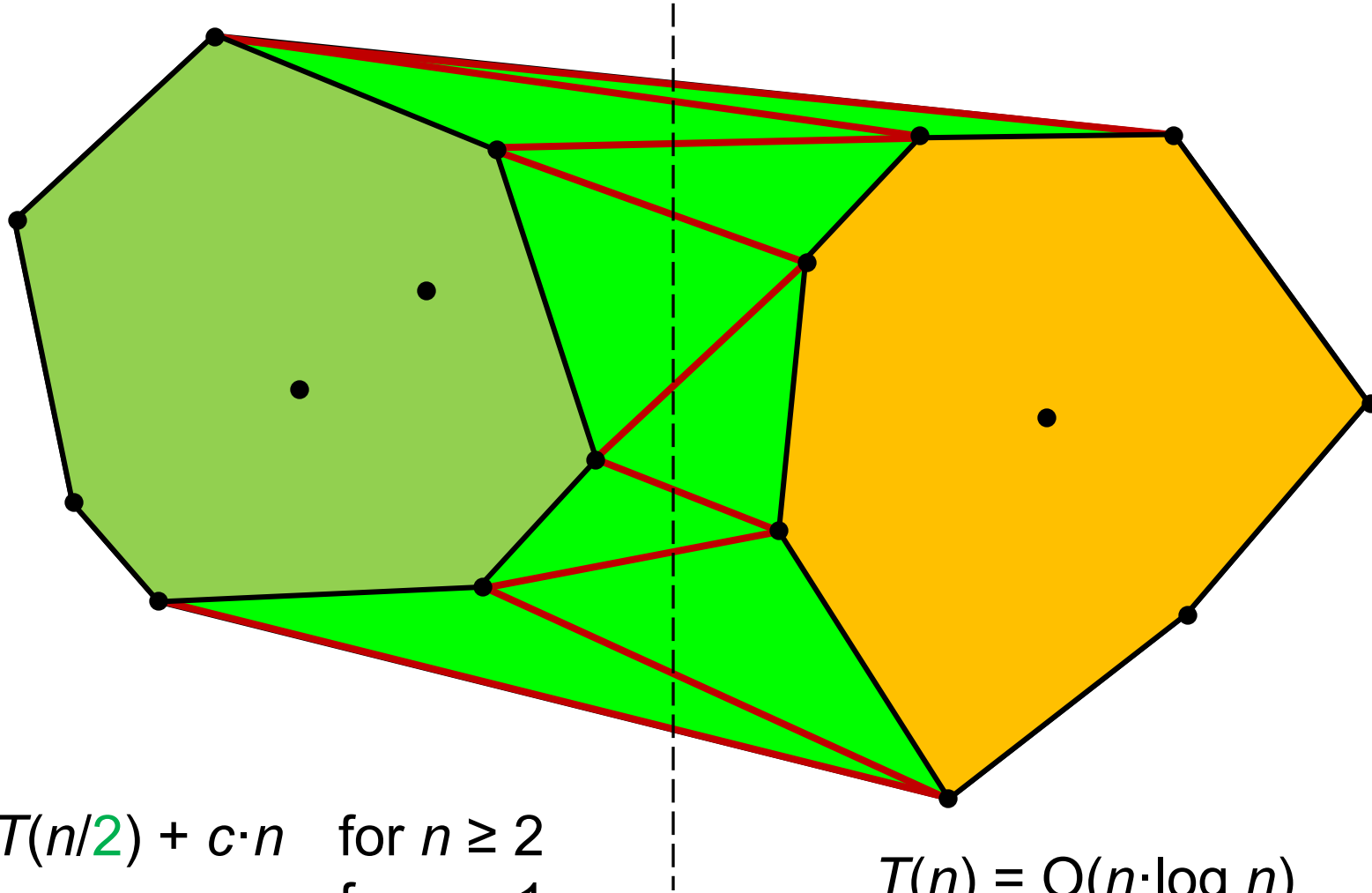
YOU KNOW
THAT THING THAT
WAS 2.3728639?

YES?

WE GOT
IT DOWN TO
2.3728596

Thunderous applause

Convex hull

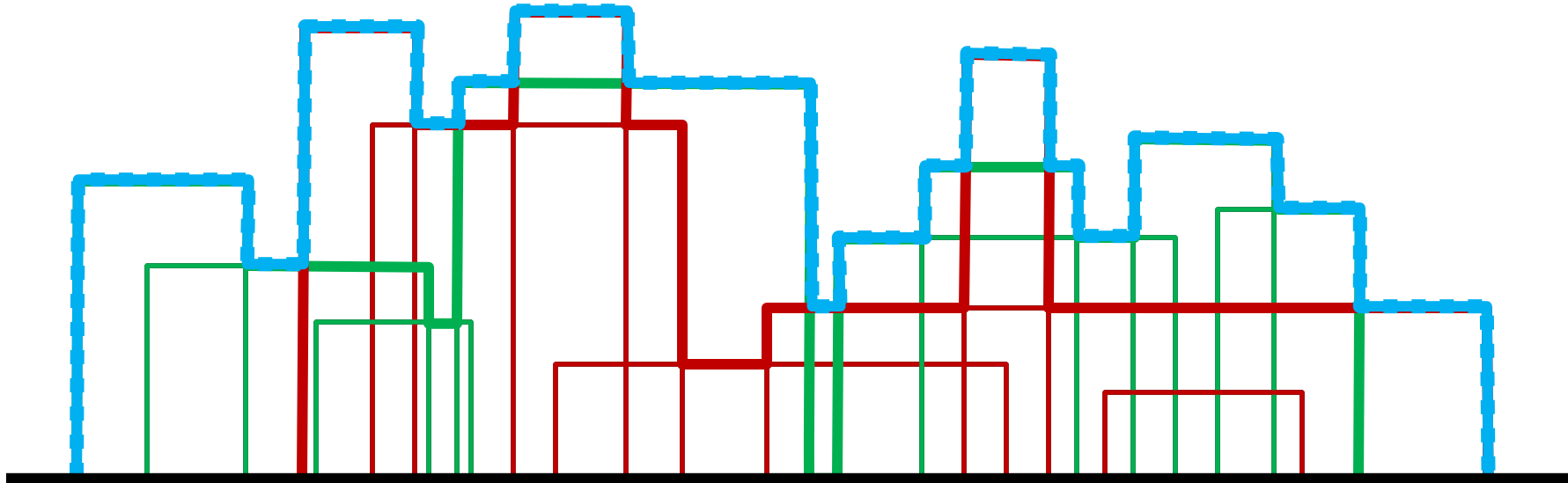


$$\begin{aligned} T(n) &= 2 \cdot T(n/2) + c \cdot n & \text{for } n \geq 2 \\ T(n) &= c & \text{for } n = 1 \end{aligned}$$

$$T(n) = O(n \cdot \log n)$$

Skyline

(assignment)



$$T(n) = ? \cdot T(n/?) + ? \quad \text{for } n \geq 2$$
$$T(n) = c \quad \text{for } n = 1$$

Algorithms and Data Structures

Selection in worst-case linear time
[CLRS, Chapter 9.3]

Selection

Find the i^{th} smallest element in a list

$L =$

10	5	12	3	1	7	42	9	15
----	---	----	---	---	---	----	---	----

$\text{SELECT}(L, 5) = 9$

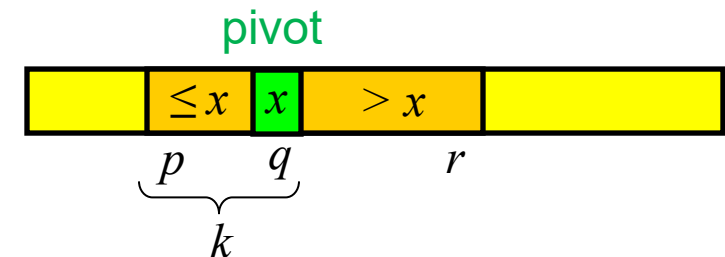
Algorithm	Time
Randomized-Select [CLRS, Chapter 9.2]	$O(n)$ expected $O(n^2)$ worst-case
Deterministic-Select [CLRS, Chapter 9.3]	$O(n)$ worst-case

Randomized-Select

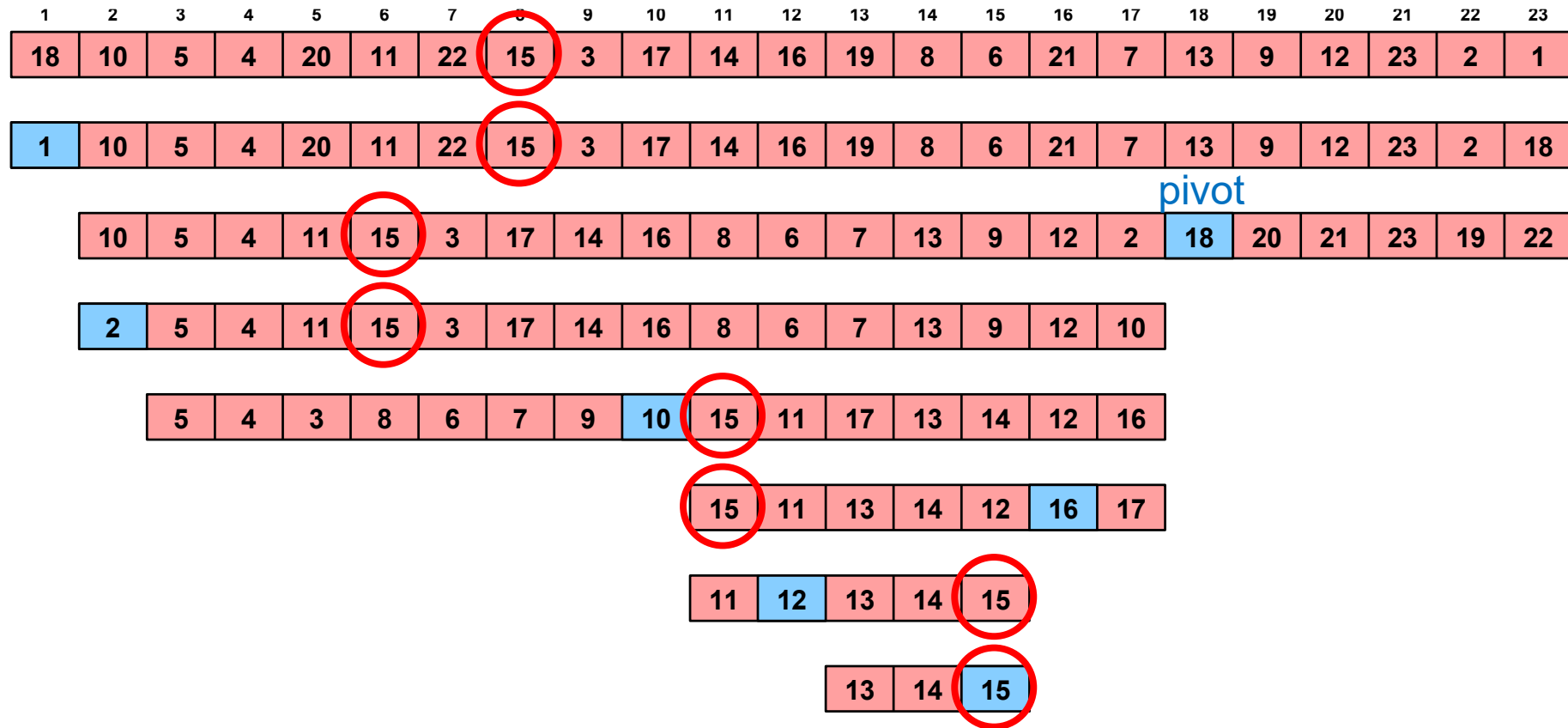
Find the i^{th} **smallest element** in $A[p..r]$ ($1 \leq i \leq r-p+1$)

RANDOMIZED-SELECT(A, p, r, i)

```
1  if  $p == r$ 
2      return  $A[p]$            //  $1 \leq i \leq r - p + 1$  when  $p == r$  means that  $i = 1$ 
3   $q = \text{RANDOMIZED-PARTITION}(A, p, r)$ 
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

SELECT(A, i)

small input

1 **if** $|A| \leq 5$ **then**
2 sort A and return i 'th element

compute pivot

3 partition A into $G_1, \dots, G_{\lceil n/5 \rceil}$ where $|G_i| \leq 5$
4 $medians = \{ g_i \mid g_i \text{ median of } G_i \}$
5 $pivot = \text{SELECT}(medians, \lfloor |medians|/2 \rfloor)$

make 1 recursive call
(like randomized select)

6 partition A w.r.t. $pivot$ into $A_{<}$, $A_{=}$ and $A_{>}$
7 **if** $i \leq |A_{<}|$ **then**
8 **return** **SELECT**($A_{<}, i$)
9 **if** $i > |A_{<}| + |A_{=}|$ **then**
10 **return** **SELECT**($A_{>}, i - |A_{<}| - |A_{=}|$)
11 **return** $pivot$

Example

$A = 30, 37, 91, 78, 34, 76, 22, 72,$
 $99, 63, 57, 57, 83, 97, 78, 44, 3,$
 $25, 44, 86, 44, 82, 52, 26, 53,$
 $90, 70, 17, 9, 56, 76, 89, 9, 37,$
 $39, 80, 84, 23, 42, 97, 72, 26$

G_1	G_2	G_3	G_4	G_5	...			$G_{\lceil n/5 \rceil}$
30	76	57	44	44	90	76	80	
37	22	57	3	82	70	89	84	72
91	72	83	25	52	17	9	23	26
78	99	97	44	26	9	37	42	
34	63	78	86	53	56	39	97	

consider input as $\lceil n/5 \rceil$ groups

sort each group independently

recursively find median

30	22	57	3	26	9	9	23	
34	63	57	25	44	17	37	42	26
37	72	78	44	52	56	39	80	72
78	76	83	44	53	70	76	84	
91	99	97	86	82	90	89	97	

$pivot = median(medians)$

medians

Quality of *pivot* ?

groups permuted such that
medians are partitioned w.r.t. *pivot*

$\leq pivot$

30	3	26	9	9	22	57	23	
34	25	44	37	17	63	57	42	26
37	44	52	39	56	72	78	80	72
78	44	53	76	70	76	83	84	
91	86	82	89	90	99	97	97	

$\sim \frac{3}{10}$ of A

$\geq pivot$

Maximum size of $A_{<}$?

$A_{<}$ are all elements in A smaller than the pivot

a) $\sim 3/10 \cdot |A|$

b) $\sim 1/4 \cdot |A|$

c) $\sim 1/2 \cdot |A|$

d) $\sim 7/10 \cdot |A|$

e) $\sim 3/4 \cdot |A|$

f) Don't know

$\leq \text{pivot}$

30	3	26	9	9	22	57	23	
34	25	44	37	17	63	57	42	26
37	44	52	39	56	72	78	80	72
78	44	53	76	70	76	83	84	
91	86	82	89	90	99	97	97	

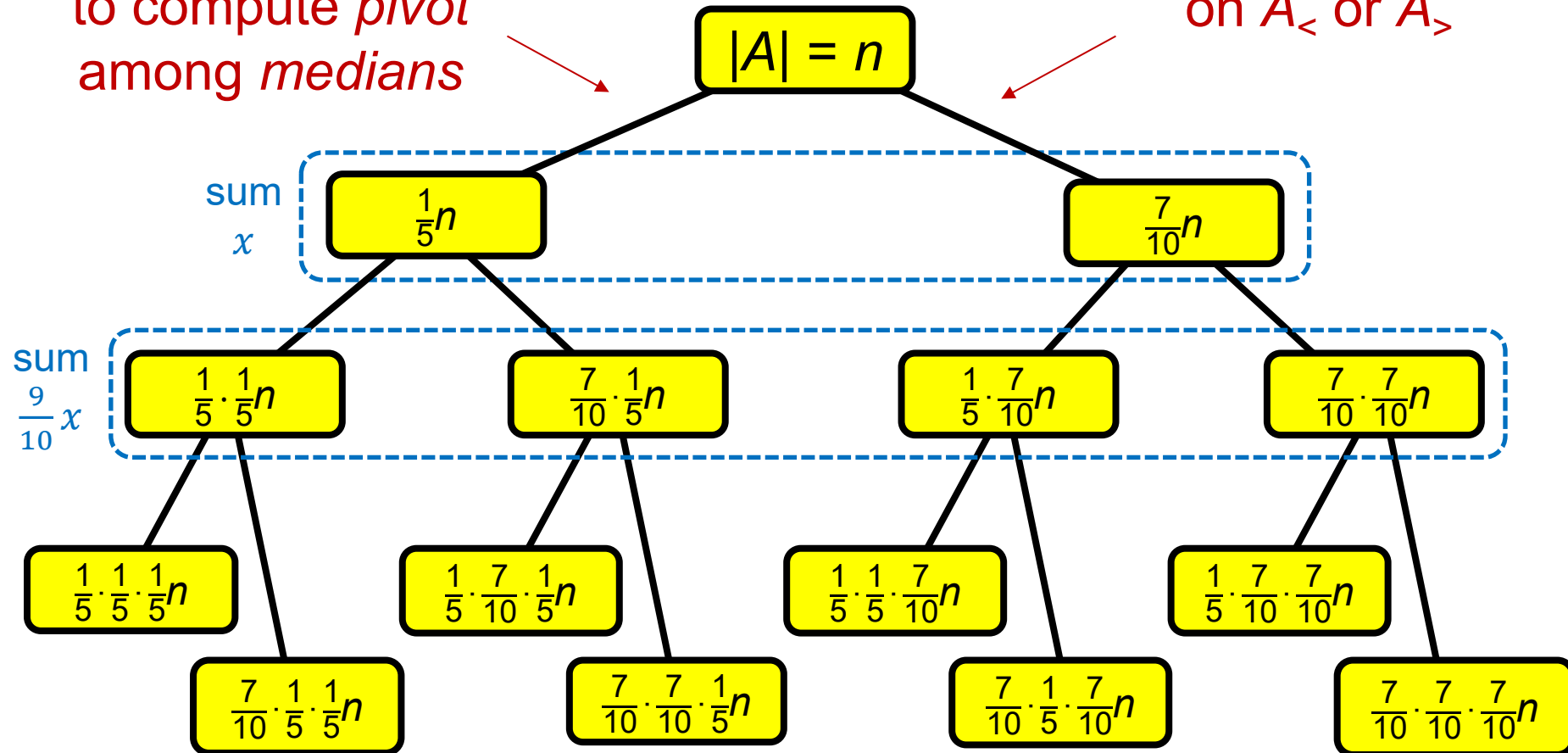
$\geq \text{pivot}$

$\sim \frac{3}{10}$ of A

Recursion tree $\text{SELECT}(A, i)$

Left recursive call
to compute *pivot*
among *medians*

right recursive call
on $A_{<}$ or $A_{>}$



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 $A_{>}$

$$T(n) \leq \begin{cases} T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } 1 \leq n \leq 5 \end{cases}$$

time to sort ≤ 5 elements

Note: Proof ignores that there can be $O(1)$ additional elements to each recursive call

Analysis (considering recursion tree)

$$T(n) \leq \begin{cases} T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } 1 \leq n \leq 5 \end{cases}$$

Note: The sum of subproblem sizes from depth $i + 1$ to depth i decreases by at least a factor $\frac{1}{5} + \frac{7}{10} = \frac{9}{10}$

$$T(n) \leq \sum_{i=0}^{\infty} c \cdot n \cdot \left(\frac{9}{10}\right)^i = c \cdot n \cdot \frac{1}{1 - 9/10} = 10 \cdot c \cdot n$$

Note: Proof ignores that there can be $O(1)$ additional elements to each recursive call

Analysis (by induction)

$$T(n) \leq \begin{cases} T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } 1 \leq n \leq 5 \end{cases}$$

Prove $T(n) \leq k \cdot n$ for some k

Proof

Base case $1 \leq n \leq 5$:

$$T(n) = c \leq k \cdot n \text{ if } c \leq k$$

Induction step $n > 5$:

$$T(n) \leq T\left(\frac{1}{5}n\right) + T\left(\frac{7}{10}n\right) + c \cdot n \stackrel{\text{i.h.}}{\leq} k \cdot \frac{1}{5}n + k \cdot \frac{7}{10}n + c \cdot n = \left(k \frac{9}{10} + c\right)n \leq k \cdot n$$

$$\text{if } k \frac{9}{10} + c \leq k \iff c \leq k \left(1 - \frac{9}{10}\right) \iff 10 \cdot c \leq k$$

Note: Proof ignores that there can be $O(1)$ additional elements to each recursive call

$$T(n) \leq 10 \cdot c \cdot n$$

Precise analysis

(tighter analysis than CLRS)

|medians|

$\max\{|A_{<}|, |A_{>}|\}$

$$T(n) \leq \begin{cases} T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(n - 3 \left\lfloor \frac{\lceil n/5 \rceil}{2} \right\rfloor + 2\right) + c \cdot n & \text{for } n > 5 \\ c & \text{for } n \leq 5 \end{cases}$$

Solution $T(n) \leq c \cdot \max\{1, 10n - 30\}$

Proof: For $1 \leq n \leq 15$ compute recurrence

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T(n) / c \leq$	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)

$$T(k) \leq c \cdot \max\{1, 10k - 30\} \text{ for } 1 \leq k \leq n - 1$$

Induction step (prove for n)

$$\begin{aligned} T(n) &\leq T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(n - 3 \left\lfloor \frac{\lceil n/5 \rceil}{2} \right\rfloor + 2\right) + c \cdot n \\ &\leq c \cdot \left(10 \left\lceil \frac{n}{5} \right\rceil - 30 + 10 \left(n - 3 \left\lfloor \frac{\lceil n/5 \rceil}{2} \right\rfloor + 2 \right) - 30 + n \right) \\ &\leq c \cdot \left(10 \left(\frac{n}{5} + 1 \right) + 10 \left(n - 3 \frac{n/5}{2} + 2 \right) - 60 + n \right) \\ &= c \cdot (10n - 30) \end{aligned}$$

□

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

$$T(n) \leq \begin{cases} T\left(\left\lceil \frac{n}{5} \right\rceil\right) + T\left(n - 3 \left\lceil \frac{\lceil n/5 \rceil}{2} \right\rceil + 2\right) + 7 \left\lceil \frac{n}{5} \right\rceil + n - 1 & \text{for } n > 5 \\ \frac{n}{T(n) \leq} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 \\ 0 & 1 & 3 & 5 & 7 \end{array} & \text{for } n \leq 5 \end{cases}$$

compute medians compute $|A_{<}|$, $|A_{=}|$ and $|A_{>}|$

Solution $T(n) \leq \max\{n, 24n - 72\}$

Proof: $n \leq 15$ check manually, $n \geq 16$ by induction.

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
$T(n) \leq$	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	$\leq 2.95n$ $\geq (2 + \varepsilon)n \quad \varepsilon \approx 2^{-80}$

C. A. R. Hoare, *Algorithm 65: Find*, Communications of the ACM, 1961, doi: [10.1145/366622.366647](https://doi.org/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](https://doi.org/10.1016/S0022-0000(73)80033-9)

Dorit Dor, Uri Zwick, *Median Selection Requires $(2+\varepsilon)n$ Comparisons*, SIAM Journal of Discrete Mathematics, 2001, doi: [10.1137/S0895480199353895](https://doi.org/10.1137/S0895480199353895)

Dorit Dor, Uri Zwick, *Selecting the Median*, SIAM Journal of Computing, 1999, doi: [10.1137/S0097539795288611](https://doi.org/10.1137/S0097539795288611)

Algorithms and Data Structures

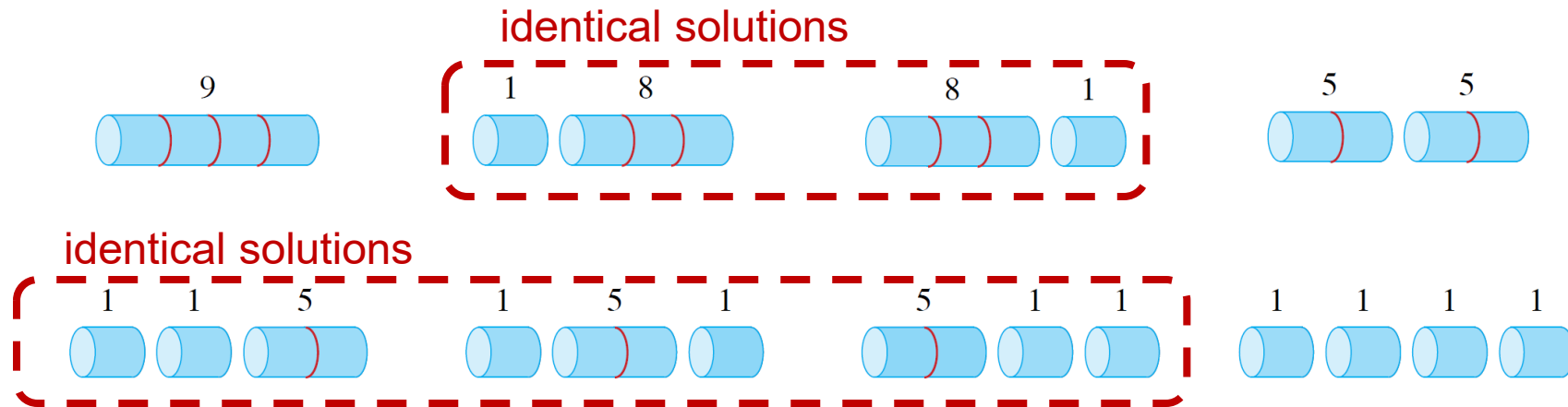
Dynamic programming
[CLRS, Chapter 14]

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

$$\begin{aligned}r_1 &= 1 \\r_2 &= 5 \\r_3 &= 8 \\r_4 &= 10 \\r_5 &= 13 \\r_6 &= 17 \\r_7 &= 18 \\r_8 &= 22 \\r_9 &= 25 \\r_{10} &= 30\end{aligned}$$

cutting

$$\begin{aligned}1 &= 1 \quad (\text{no cuts}) , \\2 &= 2 \quad (\text{no cuts}) , \\3 &= 3 \quad (\text{no cuts}) , \\4 &= 2 + 2 , \\5 &= 2 + 3 , \\6 &= 6 \quad (\text{no cuts}) , \\7 &= 1 + 6 \text{ or } 7 = 2 + 2 + 3 , \\8 &= 2 + 6 , \\9 &= 3 + 6 , \\10 &= 10 \quad (\text{no cuts}) .\end{aligned}$$

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   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max \{q, p[i] + \text{CUT-ROD}(p, n - i)\}$ 
6  return  $q$ 
```

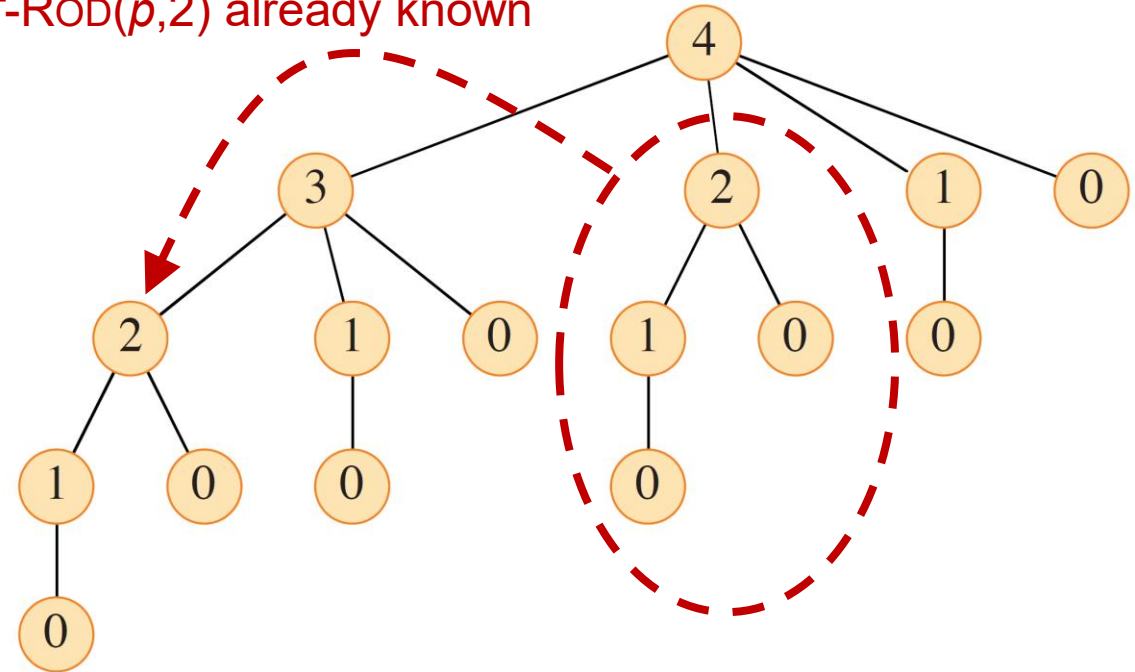
Optimal rod cutting – recursive solution

$$r_n = \begin{cases} 0 & \text{if } n = 0 \\ \max_{i=1..n} (p_i + r_{n-i}) & \text{otherwise} \end{cases}$$

CUT-ROD(p, n)

```
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max \{q, p[i] + \text{CUT-ROD}(p, n - i)\}$ 
6  return  $q$ 
```

CUT-ROD($p, 2$) already known



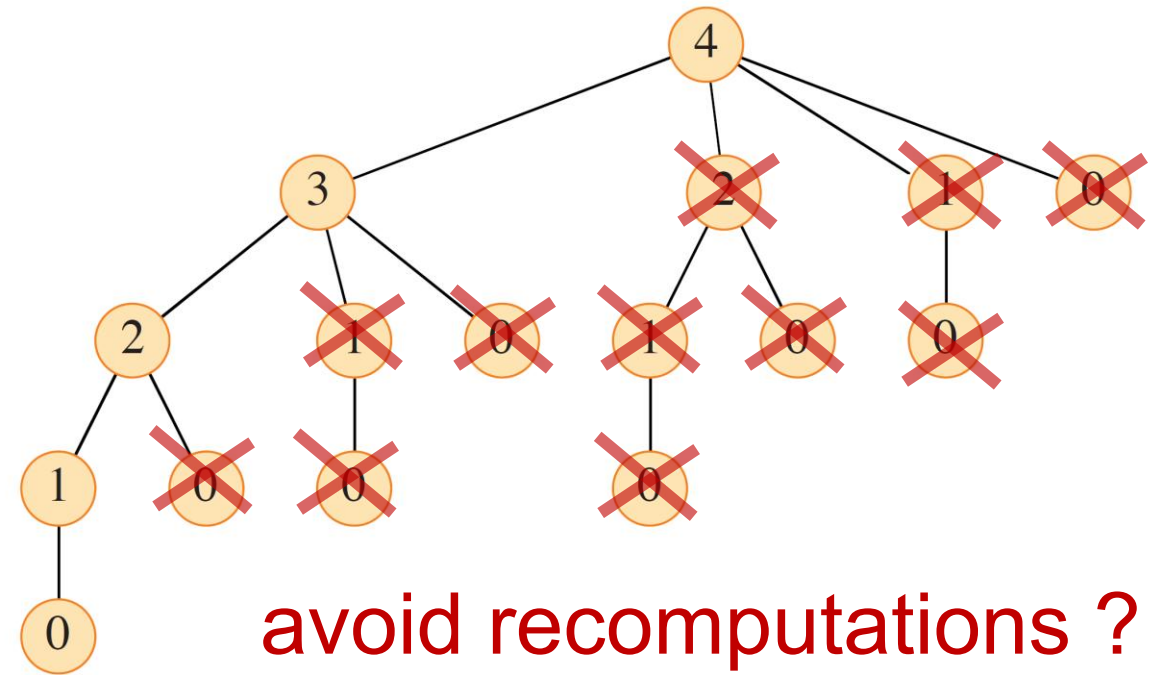
Time $O(2^n)$

Optimal rod cutting – recursive solution

$$r_n = \begin{cases} 0 & \text{if } n = 0 \\ \max_{i=1..n} (p_i + r_{n-i}) & \text{otherwise} \end{cases}$$

CUT-ROD(p, n)

```
1  if  $n == 0$ 
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$ 
5       $q = \max \{q, p[i] + \text{CUT-ROD}(p, n - i)\}$ 
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       $r[i] = -\infty$ 
4  return MEMOIZED-CUT-ROD-AUX( $p, n, r$ )
```

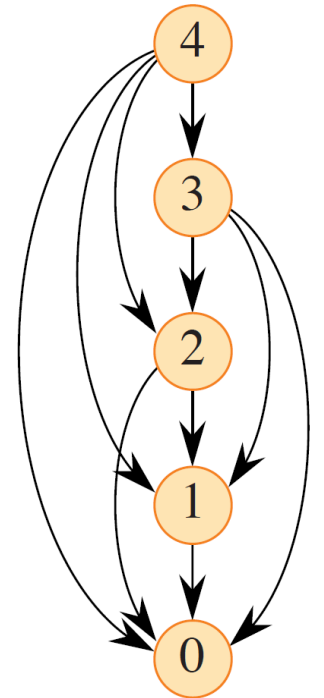
MEMOIZED-CUT-ROD-AUX(p, n, r)

```
1  if  $r[n] \geq 0$ 
2      return  $r[n]$ 
3  if  $n == 0$ 
4       $q = 0$ 
5  else  $q = -\infty$ 
6      for  $i = 1$  to  $n$ 
7           $q = \max \{q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r)\}$ 
8   $r[n] = q$ 
9  return  $q$ 
```

recurrence

remember result

subproblem
dependency



Time $O(n^2)$

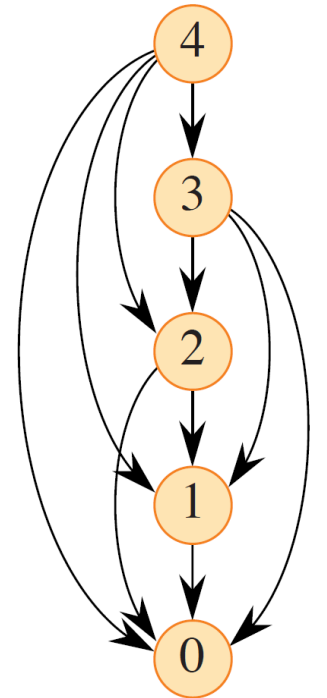
Optimal rod cutting – systematic tabulation

BOTTOM-UP-CUT-ROD(p, n)

```
1  let  $r[0:n]$  be a new array
2   $r[0] = 0$ 
3  for  $j = 1$  to  $n$  ← order crucial !
4       $q = -\infty$ 
5      for  $i = 1$  to  $j$ 
6           $q = \max \{q, p[i] + r[j - i]\}$ 
7       $r[j] = q$ 
8  return  $r[n]$ 
```

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

subproblem
dependency



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       $q = -\infty$ 
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   $(r, s) = \text{EXTENDED-BOTTOM-UP-CUT-ROD}(p, n)$ 
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 $(r, s) = \text{EXTENDED-BOTTOM-UP-CUT-ROD}(p, n)$

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 $X = x_1, \dots, x_n$ and $W = w_1, \dots, w_n$ 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 $i_1 < i_2 < \dots < i_l$ where $x_{i_1} < x_{i_2} < \dots < x_{i_l}$ and $w_{i_1} + w_{i_2} + \dots + w_{i_l}$ 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 .

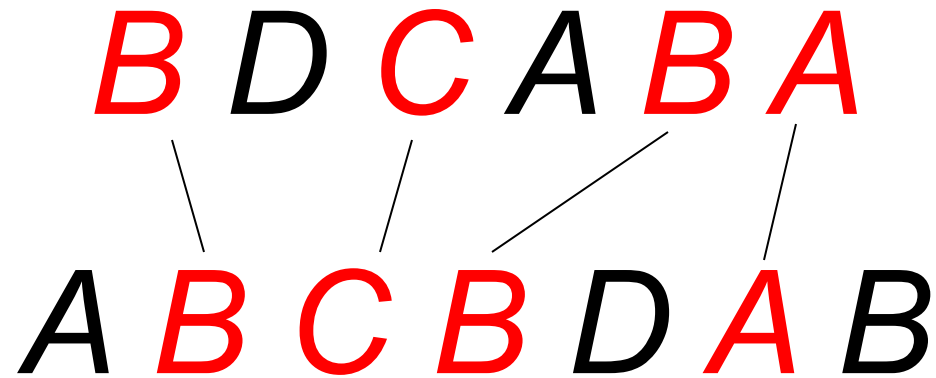
$$MEH(i) = \begin{cases} w_i + \max\{MEH(j) \mid 1 \leq j < i \wedge x_j < x_i\} & \text{if there exists } 1 \leq j < i \wedge x_j < x_i \\ w_i & \text{otherwise} \end{cases}$$

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
$MEH(i)$	1	2	3	4	1	?			

Exam, august 2008, question 4

- 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

Longest common subsequence

$Y =$ **B** D **C** A **B** ^{j} **A**

$X =$ A **B** **C** **B** D **A** ^{i} **B**

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max \{c[i, j - 1], c[i - 1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

length of longest common subsequence of $x_1 \cdots x_i$ and $y_1 \cdots y_j$

$c[5,3]$?

$Y = ABABGACBABAD$
 $X = BACABAABABC$

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

$Y =$ $B D C A B$ A
 $X =$ $A B C B$ $D A B$

j i

		j	0	1	2	3	4	5	6	
				y_j	B	D	C	A	B	A
i										
0	x_i		0	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	

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j, \\ \max\{c[i, j-1], c[i-1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

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              $b[i, j] = \nwarrow$ 
11         elseif  $c[i - 1, j] \geq c[i, j - 1]$ 
12              $c[i, j] = c[i - 1, j]$ 
13              $b[i, j] = \uparrow$ 
14         else  $c[i, j] = c[i, j - 1]$ 
15              $b[i, j] = \leftarrow$ 
16  return  $c$  and  $b$ 
    
```

		j	0	1	2	3	4	5	6	
				y_j	B	D	C	A	B	A
i	x_i									
0	x_i		0	0	0	0	0	0	0	0
1	A		0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\swarrow$ 1	$\leftarrow$ 1	$\swarrow$ 1	
2	B		0	$\swarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\swarrow$ 2	$\leftarrow$ 2	
3	C		0	$\uparrow$ 1	$\uparrow$ 1	$\swarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	
4	B		0	$\swarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\swarrow$ 3	$\leftarrow$ 3	
5	D		0	$\uparrow$ 1	$\swarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3	
6	A		0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\swarrow$ 3	$\uparrow$ 3	$\swarrow$ 4	
7	B		0	$\swarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\swarrow$ 4	$\uparrow$ 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  $b[i, j] == \nwarrow$ 
4      PRINT-LCS( $b, X, i - 1, j - 1$ )
5      print  $x_i$ 
6  elseif  $b[i, j] == \uparrow$ 
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	x_i	y_j		B	D	C	A	B	A
0		x_i	0	0	0	0	0	0	0
1	A		0	$\uparrow$ 0	$\uparrow$ 0	$\uparrow$ 0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ 1
2	B		0	$\nwarrow$ 1	$\leftarrow$ 1	$\leftarrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2
3	C		0	$\uparrow$ 1	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ 2	$\uparrow$ 2
4	B		0	$\nwarrow$ 1	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\leftarrow$ 3
5	D		0	$\uparrow$ 1	$\nwarrow$ 2	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\uparrow$ 3
6	A		0	$\uparrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\nwarrow$ 3	$\uparrow$ 3	$\nwarrow$ 4
7	B		0	$\nwarrow$ 1	$\uparrow$ 2	$\uparrow$ 2	$\uparrow$ 3	$\nwarrow$ 4	$\uparrow$ 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

		<i>j</i>						
		0	1	2	3	4	5	6
<i>i</i>	<i>y_j</i>	<i>B</i>	<i>D</i>	<i>C</i>	<i>A</i>	<i>B</i>	<i>A</i>	
0	<i>x_i</i>							
1	<i>A</i>	0	↑	↑	↑	↖	←	↖
2	<i>B</i>	0	↖	←	←	↑	↖	←
3	<i>C</i>	0	↑	↑	↖	←	↑	↑
4	<i>B</i>	0	↖	↑	↑	↑	↖	←
5	<i>D</i>	0	↑	↖	↑	↑	↑	↑
6	<i>A</i>	0	↑	↑	↑	↖	↑	↖
7	<i>B</i>	0	↖	↑	↑	↑	↖	↑

$m = |X|$ $n = |Y|$

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 $O(n \cdot m)$

Space $O(n + m)$

Example of **divide and conquer** used inside a **dynamic programming** algorithm

		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
i	x_i								
0			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	↖ 0 1	↑ 2 1	↑ 3 2	↑ 4 2	↖ 4 3	← 4 3
5	D		- 0	↑ 0 1	↖ 0 2	↑ 3 2	↑ 4 2	↑ 4 3	↑ 4 3
6	A		- 0	↑ 0 1	↑ 0 2	↑ 3 2	↖ 3 3	↑ 4 3	↖ 4 4
7	B		- 0	↖ - 1	↑ 0 2	↑ 3 2	↑ 3 3	↖ 3 4	↑ 4 4

$m = |X|$ $n = |Y|$

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 $O(n^{2-\epsilon})$ time ?

– Would break with the "Strong Exponential Time Hypothesis"

Hunt-Szymanski algorithm

		0	1	2	3	4	5	6	
	y_j	B	A	B	C	A	B		L
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**

$$r = |\{ (i, j) \mid x_i = y_j \}| \ll n \cdot m$$

$$r = \overset{A}{2 \cdot 2} + \overset{B}{2 \cdot 3} + \overset{C}{1 \cdot 1} = 11$$

(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:** $\text{LCS}(X, Y) = \text{LIS}(\text{Matches}(X, Y))$, LIS = longest increasing subsequence
Sort increasing i , decreasing j ; time $O((m + n + r) \cdot \log \min(n, m))$, space $O(r + m + n)$
- $|\text{LCS}|$: 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 $O(\min(n, m))$

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

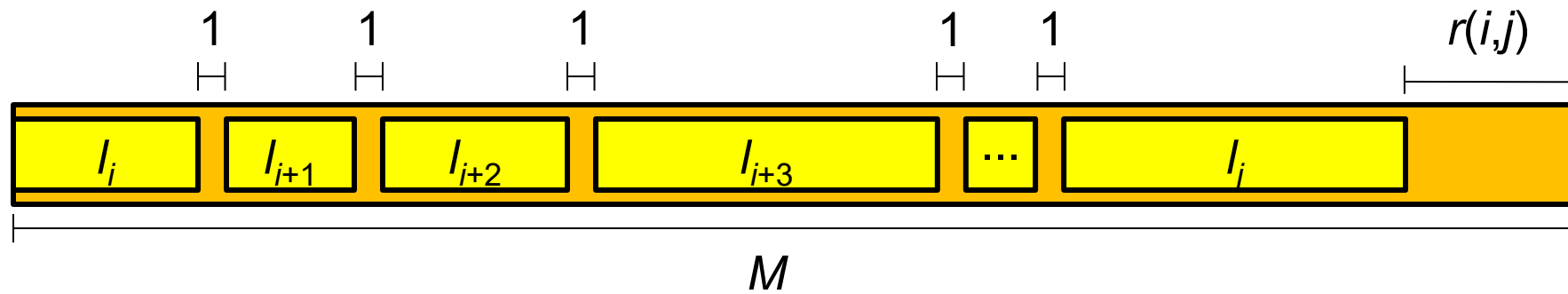
Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and nothing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. 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]

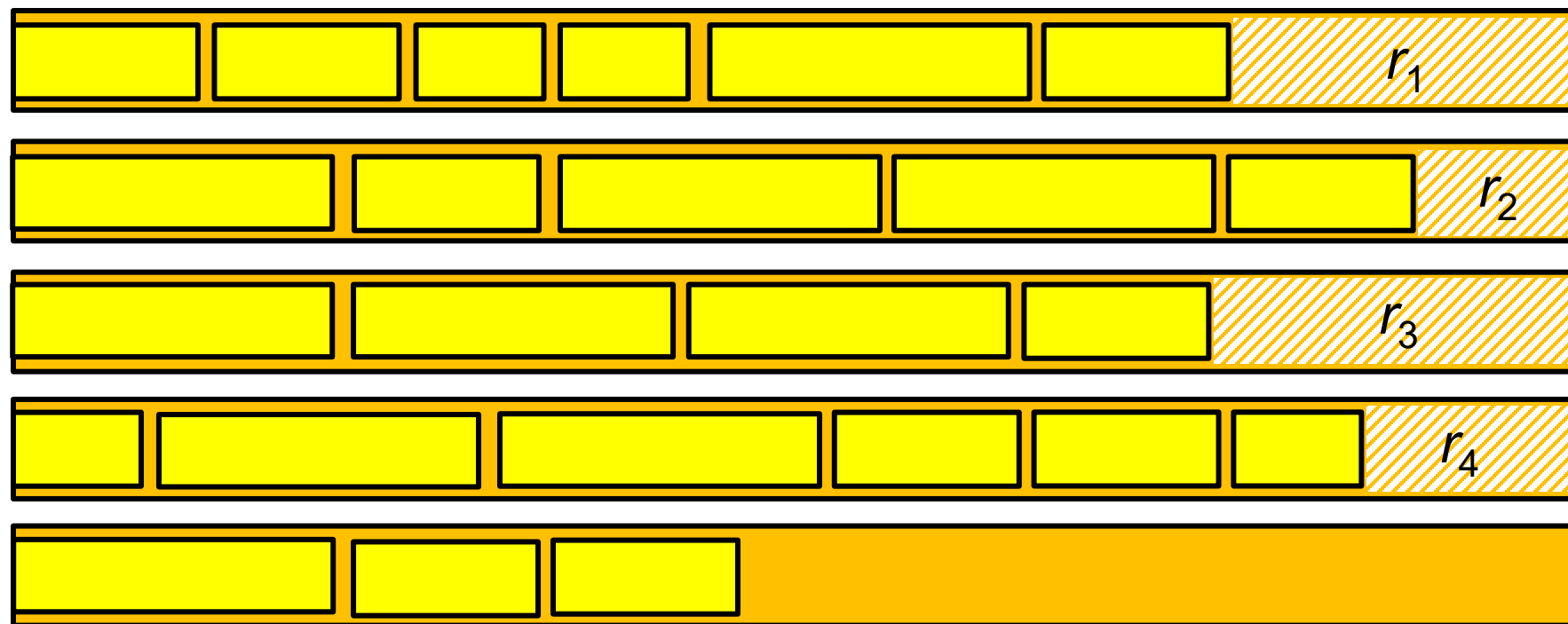
“Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \dots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of “neatness” is as follows. If a given line contains words i through j , where $i \leq j$, 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.”



$$r(i, j) = M - j + i - \sum_{k=i..j} l_k$$



$$\text{penalty} = r_1^3 + r_2^3 + r_3^3 + r_4^3$$

Printing neatly

Place the first j words on full lines

$$C[j] = \begin{cases} 0 & \text{if } j = 0 \\ \min\{C[i-1] + r(i, j)^3 \mid 1 \leq i \leq j \wedge r(i, j) \geq 0\} & \text{if } j > 0 \end{cases}$$

Best solution

$$\min\{C[j] \mid 0 \leq j < n \wedge r(j+1, n) \geq 0\}$$

remaining words $j+1, j+2, \dots, n$
on last line without penalty

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 $O(n^2)$

$$C[j] = \begin{cases} 0 & \text{if } j = 0 \\ \min\{C[i-1] + r(i,j)^3 \mid 1 \leq i \leq j \wedge r(i,j) \geq 0\} & \text{if } j > 0 \end{cases}$$

$$\min\{C[j] \mid 0 \leq j < n \wedge r(j+1, n) \geq 0\}$$

Matrix multiplication

$$\begin{matrix} & \textcolor{red}{C} \\ \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1r} \\ c_{21} & \textcolor{red}{c_{22}} & \cdots & c_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ c_{p1} & c_{p2} & \cdots & c_{pr} \end{pmatrix} & = & \begin{matrix} \textcolor{red}{A} \\ \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1q} \\ \textcolor{red}{a_{21}} & \textcolor{red}{a_{22}} & \cdots & \textcolor{red}{a_{2q}} \\ \vdots & \vdots & \ddots & \vdots \\ a_{p1} & a_{p2} & \cdots & a_{pq} \end{pmatrix} \end{matrix} & \begin{matrix} \textcolor{red}{B} \\ \begin{pmatrix} b_{11} & \textcolor{red}{b_{12}} & \cdots & b_{1r} \\ b_{21} & \textcolor{red}{b_{22}} & \cdots & b_{2r} \\ \vdots & \vdots & \ddots & \vdots \\ b_{q1} & \textcolor{red}{b_{q2}} & \cdots & b_{qr} \end{pmatrix} \end{matrix} \\ \textcolor{red}{p \times r} & & \textcolor{red}{p \times q} & & \textcolor{red}{q \times r} \end{matrix}$$

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               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
```

Multiplication of two matrices A and B with dimensions $p \times q$ and $q \times r$ takes time $O(p \cdot q \cdot r)$

Matrix-chain multiplication

$(A \cdot B) \cdot C$ 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) $(A \cdot B) \cdot C$
- b) $A \cdot (B \cdot C)$
- 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 $p_{i-1} \times p_i$ matrix

Note: There are $\Omega(4^n / n^{3/2})$ possible ways to place n pairs of parenthesis (n 'th Catalan number)

Matrix-chain multiplication

$m[i, j]$ = minimum number of (primitive) multiplications to compute $A_i \cdots A_j$

$$\begin{array}{c} p_{i-1} \times p_k \quad p_k \times p_j \\ (A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j) \end{array}$$

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j : i \leq k < j\} & \text{if } i < j \end{cases}$$

RECURSIVE-MATRIX-CHAIN(p, i, j)

```

1  if  $i == j$ 
2      return 0
3   $m[i, j] = \infty$ 
4  for  $k = i$  to  $j - 1$ 
5       $q = \text{RECURSIVE-MATRIX-CHAIN}(p, i, k)$ 
         $+ \text{RECURSIVE-MATRIX-CHAIN}(p, k + 1, j)$ 
         $+ p_{i-1}p_kp_j$ 
6      if  $q < m[i, j]$ 
7           $m[i, j] = q$ 
8  return  $m[i, j]$ 
```

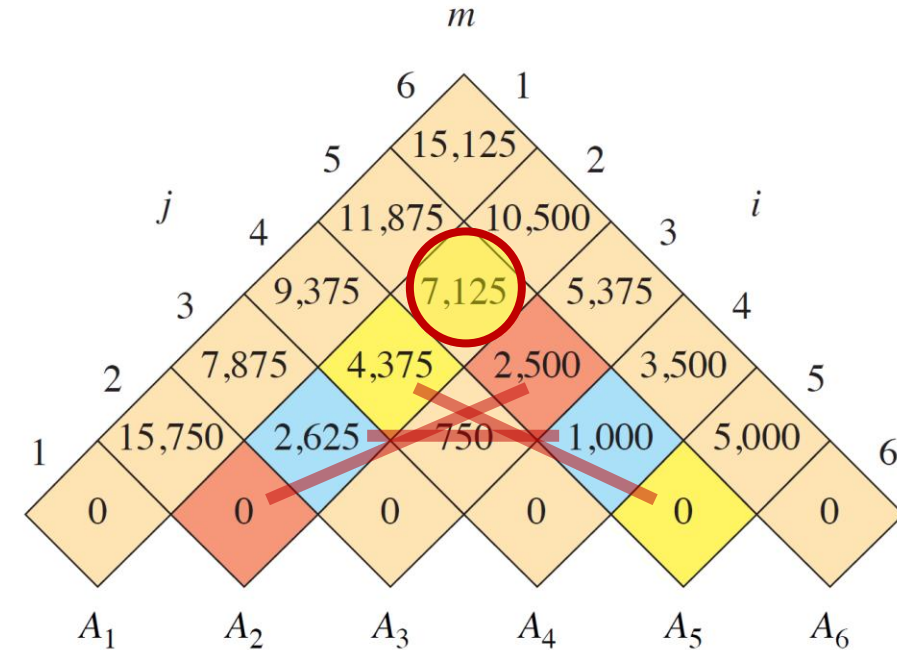
Time $\Omega(4^n/n^{3/2})$

Matrix-chain multiplication

MATRIX-CHAIN-ORDER(p, n)

```

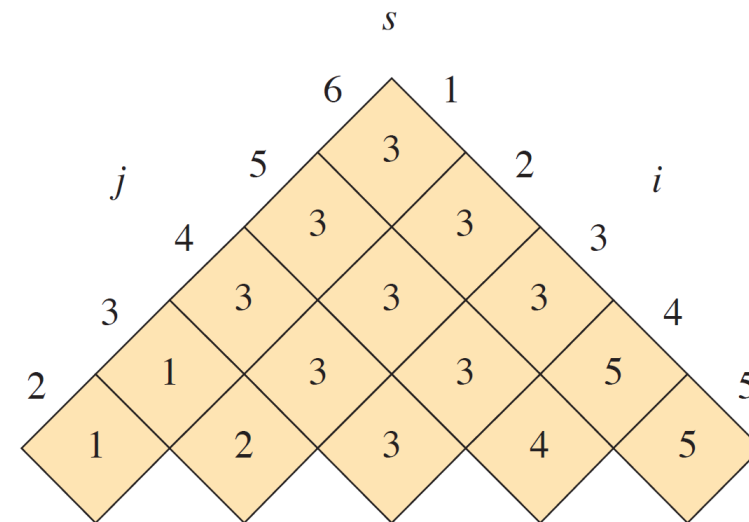
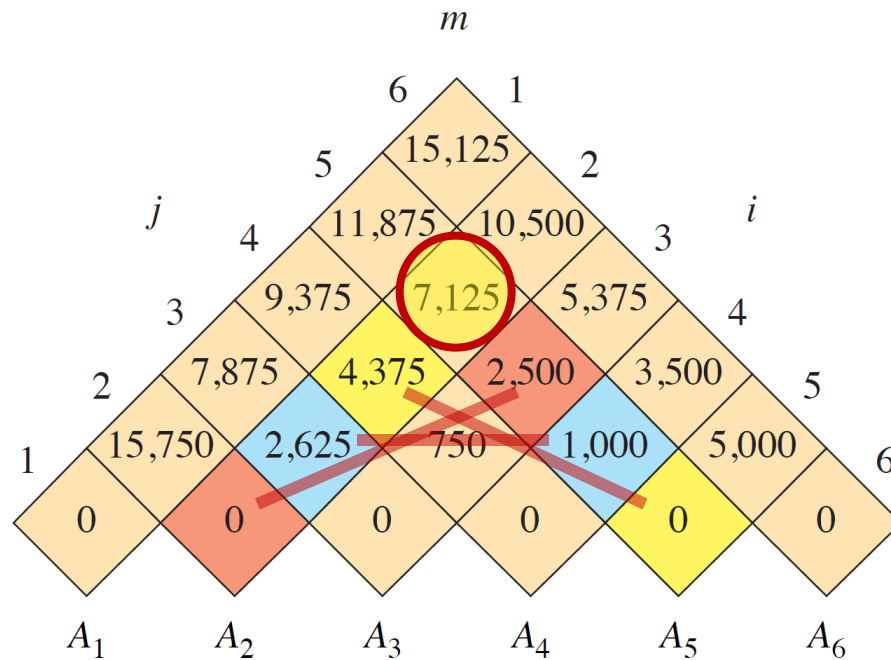
1  let  $m[1:n, 1:n]$  and  $s[1:n-1, 2:n]$  be new tables
2  for  $i = 1$  to  $n$ 
3       $m[i, i] = 0$ 
4  for  $l = 2$  to  $n$ 
5      for  $i = 1$  to  $n - l + 1$ 
6           $j = i + l - 1$ 
7           $m[i, j] = \infty$ 
8          for  $k = i$  to  $j - 1$ 
9               $q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$ 
10             if  $q < m[i, j]$ 
11                  $m[i, j] = q$ 
12                  $s[i, j] = k$ 
13  return  $m$  and  $s$ 
    
```



Time $O(n^3)$

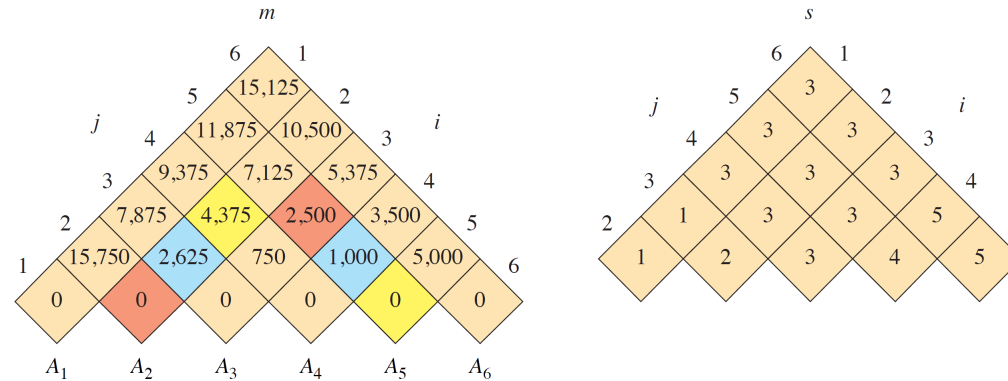
$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j : i \leq k < j\} & \text{if } i < j \end{cases}$$

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 ")"
    
```

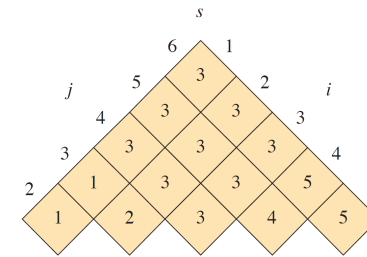
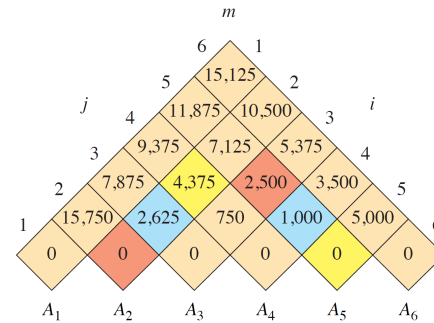
Time $O(n)$

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           $m[i, j] = \infty$ 
5  return LOOKUP-CHAIN( $m, p, 1, n$ )
```

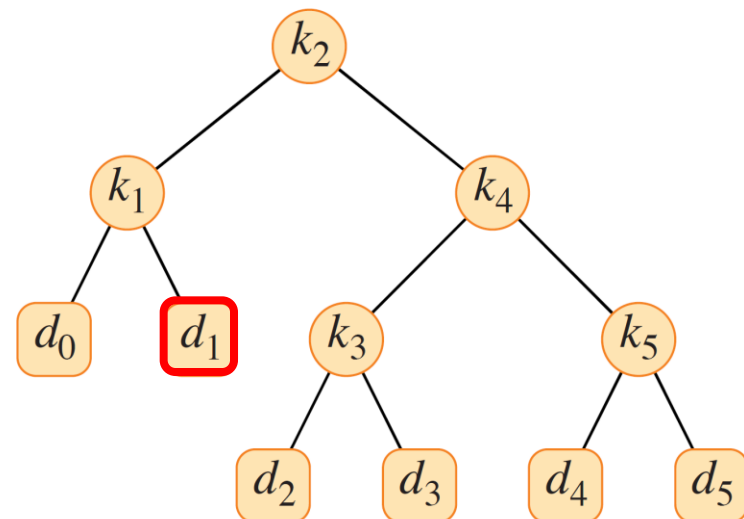
$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j : i \leq k < j\} & \text{if } i < j \end{cases}$$

LOOKUP-CHAIN(m, p, i, j)

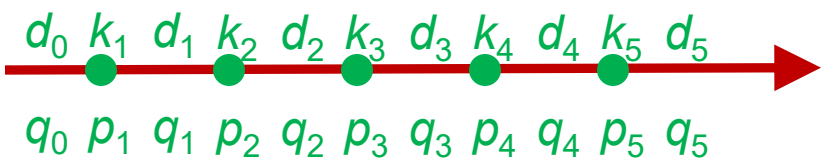
```
1  if  $m[i, j] < \infty$ 
2      return  $m[i, j]$ 
3  if  $i == j$ 
4       $m[i, j] = 0$ 
5  else for  $k = i$  to  $j - 1$ 
6       $q = \text{LOOKUP-CHAIN}(m, p, i, k)$ 
        +  $\text{LOOKUP-CHAIN}(m, p, k + 1, j) + p_{i-1}p_kp_j$ 
7      if  $q < m[i, j]$ 
8           $m[i, j] = q$ 
9  return  $m[i, j]$ 
```

Time $O(n^3)$

Optimal binary search trees



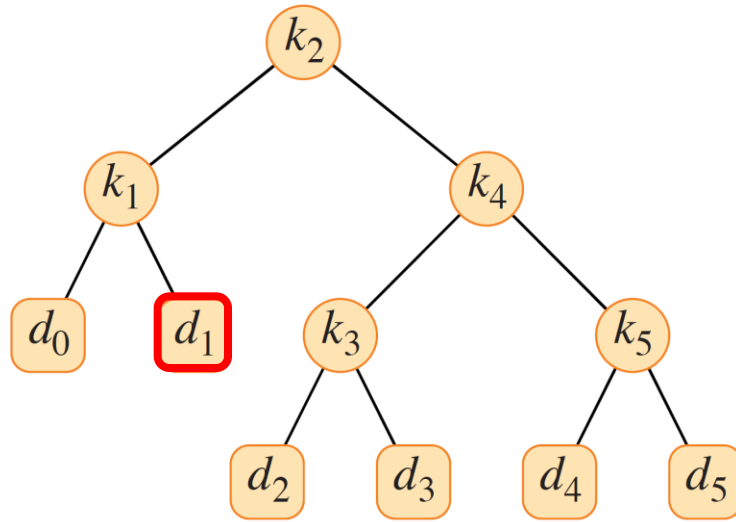
Expected search time 2.80



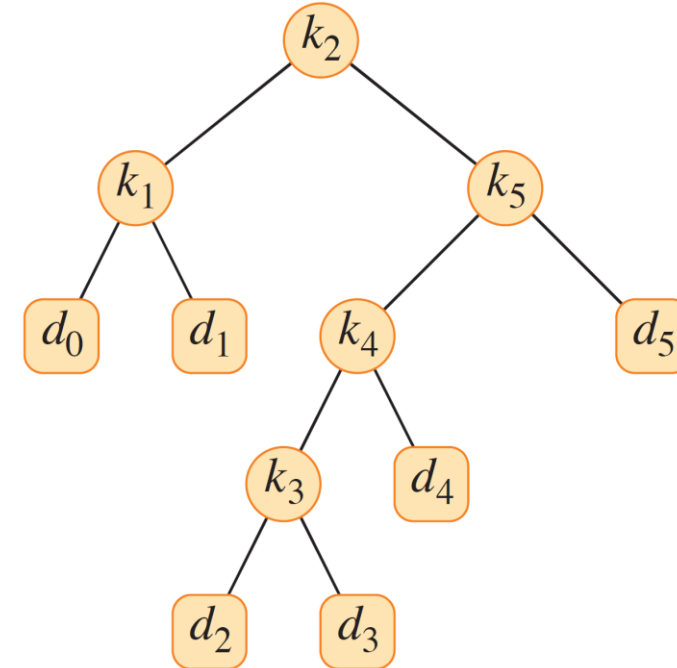
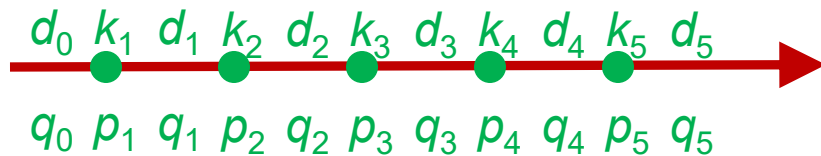
node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	$(2 + 1) \times$	0.10	$= 0.30$
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

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



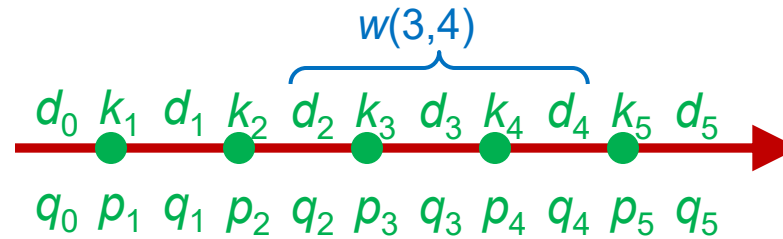
Expected search time 2.75

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

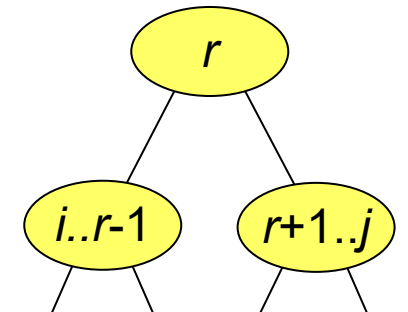
$$\sum_{i=1}^n p_i + \sum_{i=0}^n q_i = 1$$

$$w(i, j) = \sum_{l=i}^j p_l + \sum_{l=i-1}^j q_l$$



$$\begin{aligned} E[\text{search cost in } T] &= \sum_{i=1}^n (\text{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^n (\text{depth}_T(d_i) + 1) \cdot q_i \\ &= 1 + \sum_{i=1}^n \text{depth}_T(k_i) \cdot p_i + \sum_{i=0}^n \text{depth}_T(d_i) \cdot q_i \end{aligned}$$

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min \{e[i, r - 1] + e[r + 1, j] + w(i, j) : i \leq r \leq j\} & \text{if } i \leq j \end{cases}$$



optimal expected time for a search tree containing $k_i, \dots, k_j$ and $d_{i-1}, \dots, d_j$

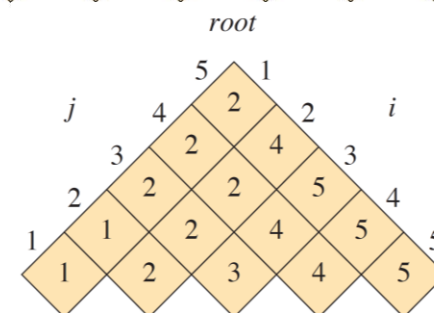
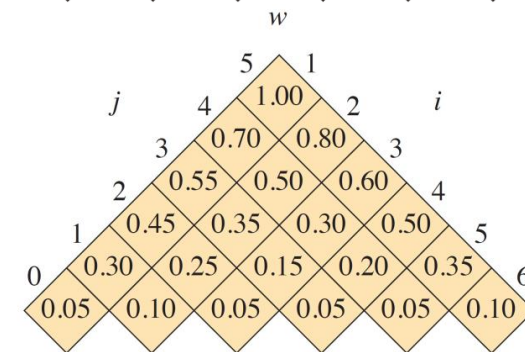
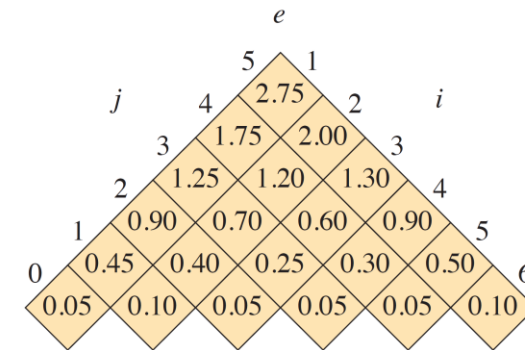
Optimal binary search trees

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           $e[i, j] = \infty$ 
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  $root$ 
    
```

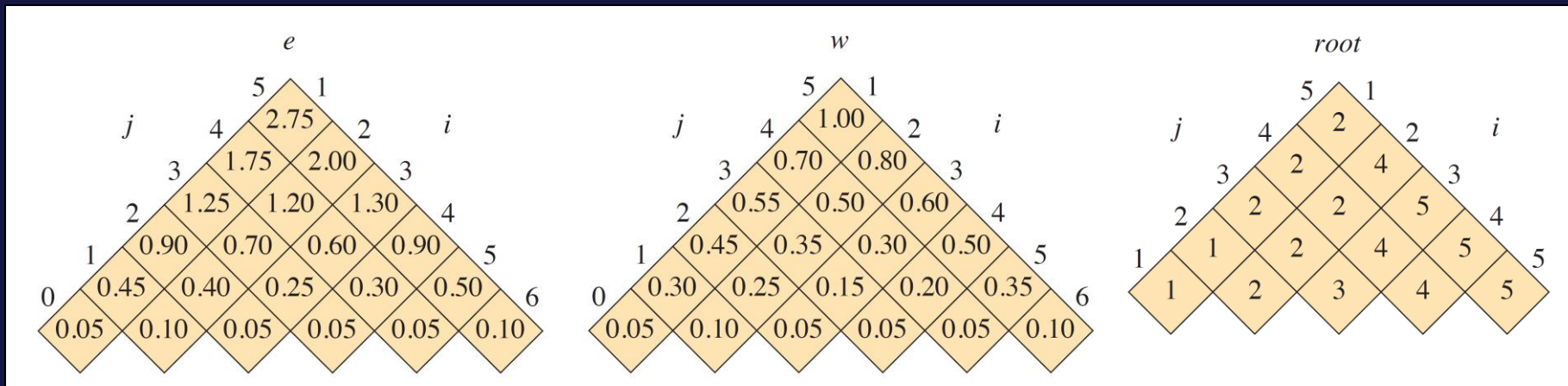
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min \{e[i, r - 1] + e[r + 1, j] + w(i, j) : i \leq r \leq j\} & \text{if } i \leq j \end{cases}$$



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 **B**ACBABA a subsequence of
ABABGACBABAD ?

Possible occurrence **AB****B**GACBABAD

Observation If there is an occurrence, then there also exists an occurrence matching the first **B** in the string (**greedy-choice property**)

ABABGACBABAD

Find recursively **ACBABA**

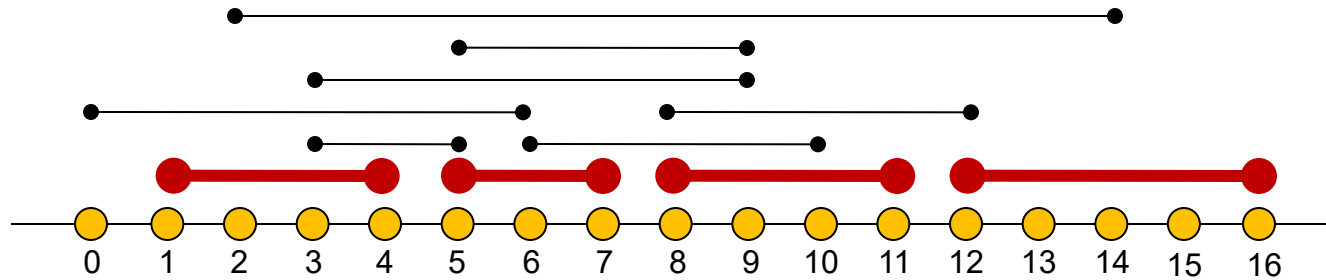
Time $O(n)$

Solution found by the greedy algorithm ?

- a) *ABABGACBABAD*
- b) *ABABGACBABAD*
- c) *ABABGACBABAD*
- 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

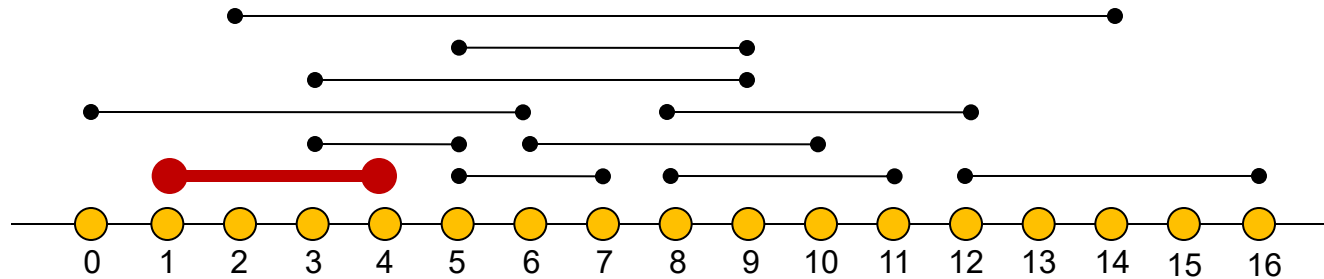


Input n activities with start time s_i and finishing time f_i

Output Maximal subset of non-overlapping activities

Activity selection

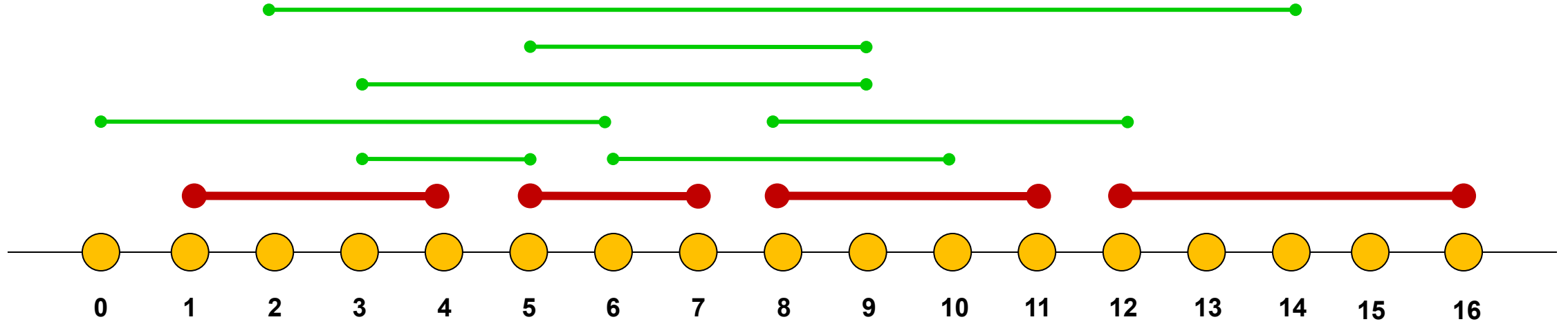
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	7	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16



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

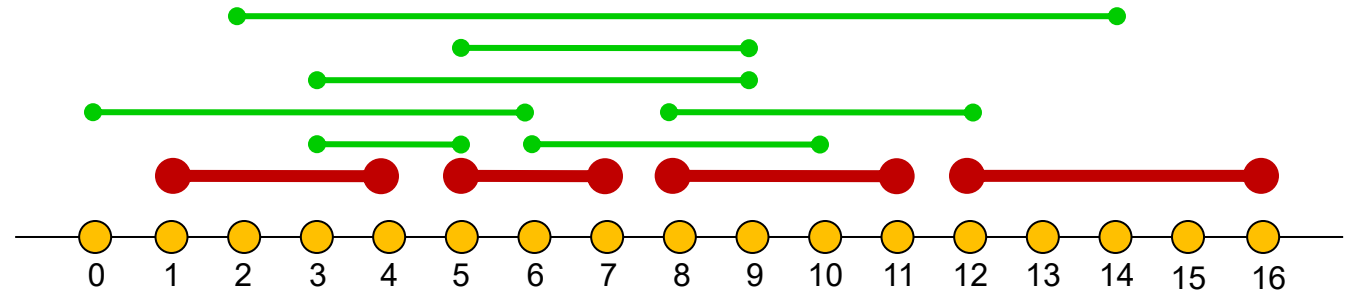
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	7	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16

← sorted

GREEDY-ACTIVITY-SELECTOR(s, f, n)

```

1   $A = \{a_1\}$ 
2   $k = 1$ 
3  for  $m = 2$  to  $n$ 
4      if  $s[m] \geq f[k]$ 
5           $A = A \cup \{a_m\}$ 
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) 00010010101101
- b) 11000011110101
- c) 10000011010101
- d) 00011100101110
- 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

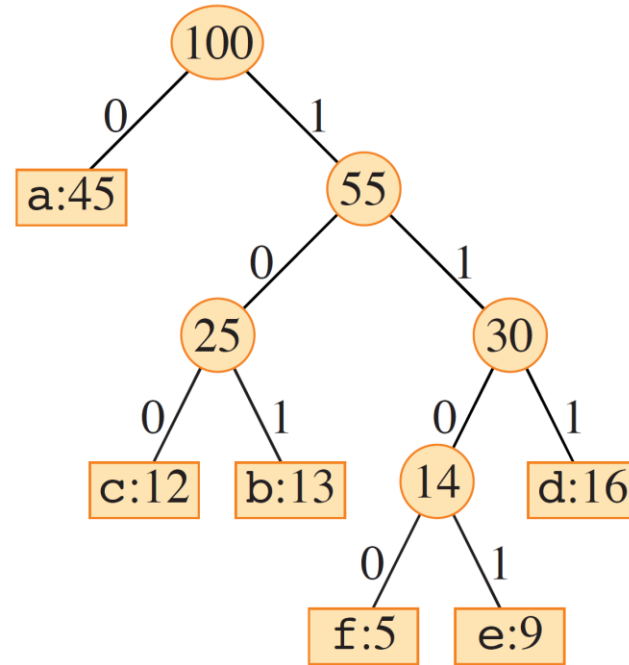
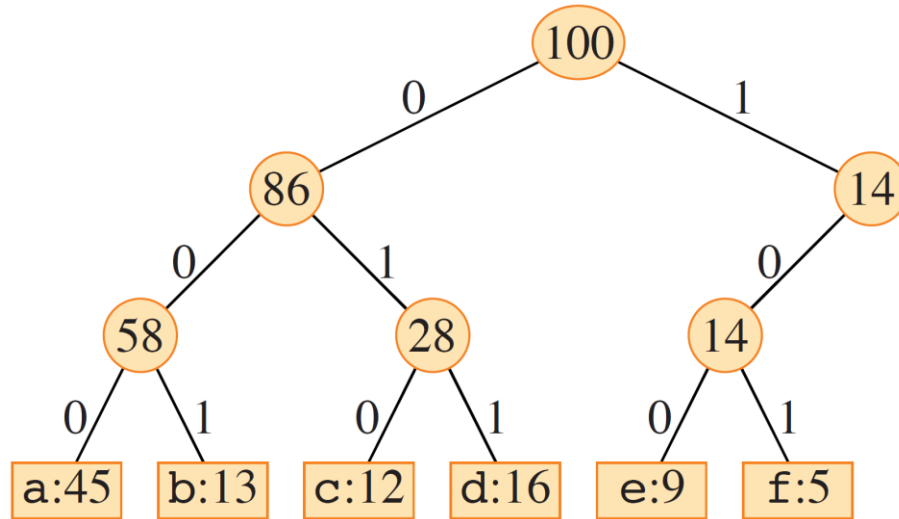
Compression

Given the frequencies 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

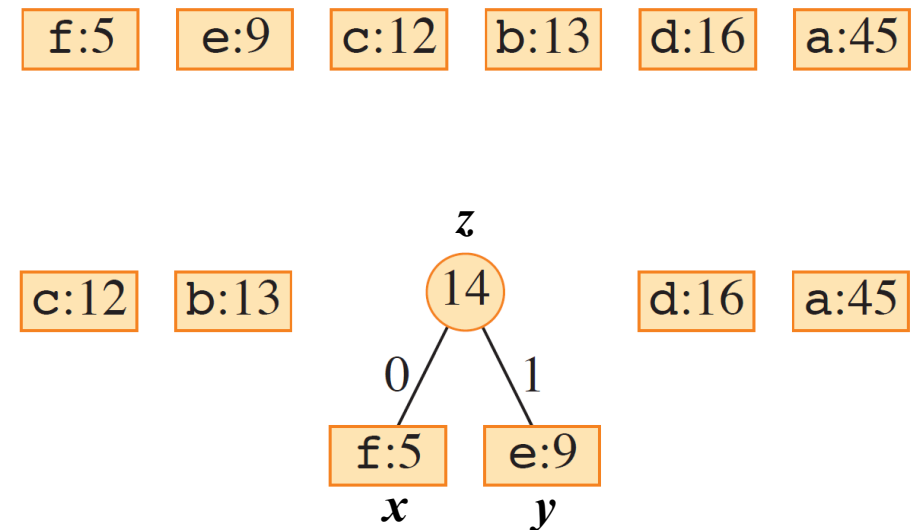


	a	b	c	d	e	f	
Frequency (in thousands)	45	13	12	16	9	5	
Fixed-length codeword	000	001	010	011	100	101	$3 * (45+13+12+16+9+5) = 300 \text{ bits}$
Variable-length codeword	0	101	100	111	1101	1100	$1*45+3*(13+12+16)+4*(9+5) = 224 \text{ bits}$

Huffman codes

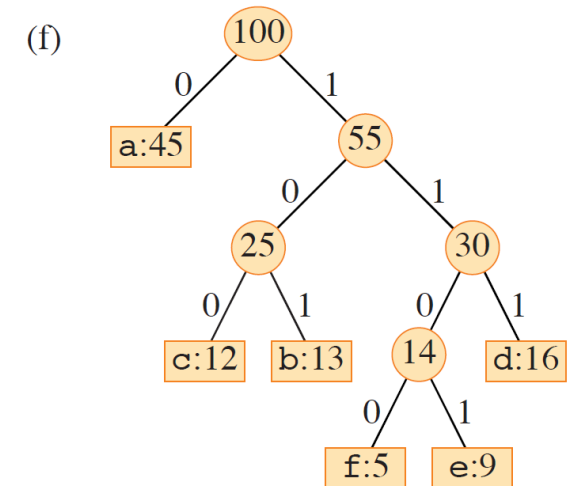
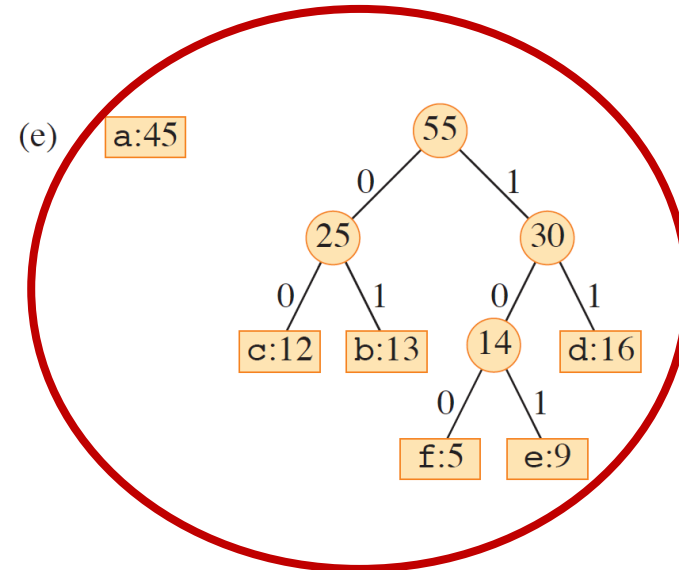
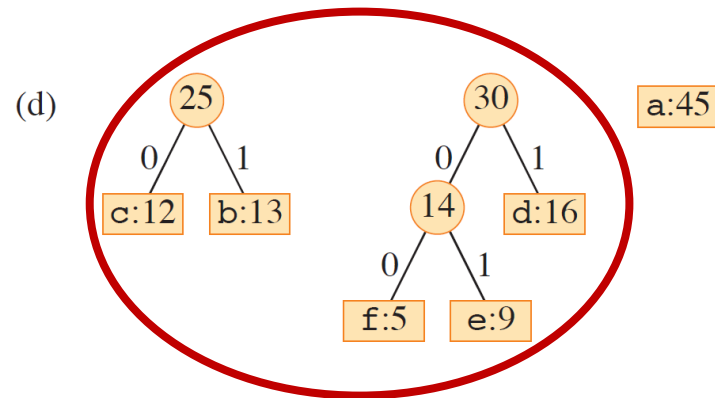
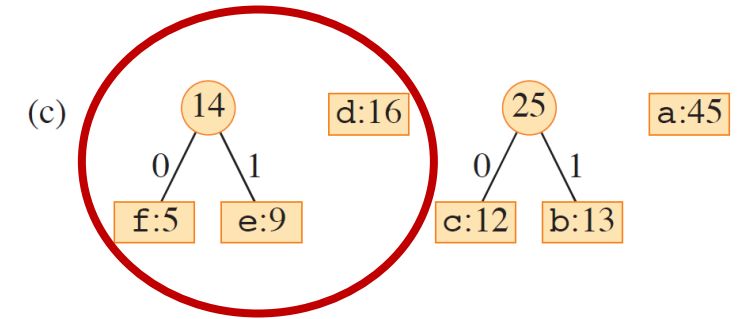
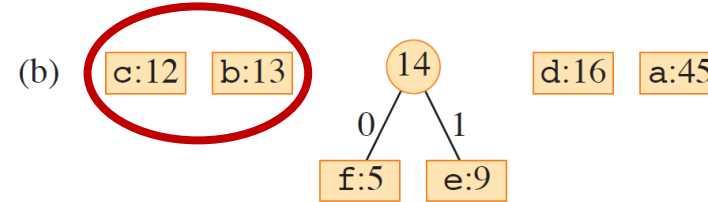
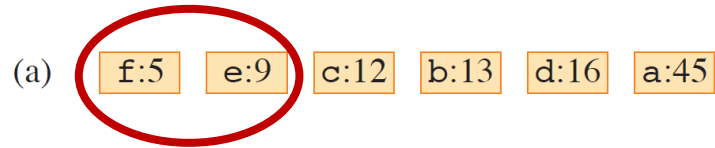
HUFFMAN(C)

```
1   $n = |C|$ 
2   $Q = C$ 
3  for  $i = 1$  to  $n - 1$ 
4      allocate a new node  $z$ 
5       $x = \text{EXTRACT-MIN}(Q)$ 
6       $y = \text{EXTRACT-MIN}(Q)$ 
7       $z.\text{left} = x$ 
8       $z.\text{right} = y$ 
9       $z.\text{freq} = x.\text{freq} + y.\text{freq}$ 
10      $\text{INSERT}(Q, z)$ 
11 return  $\text{EXTRACT-MIN}(Q)$ 
```



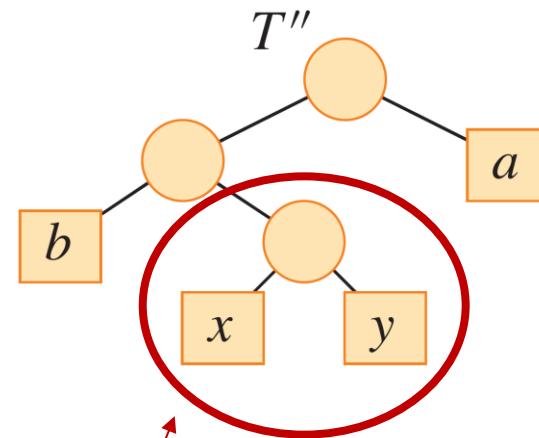
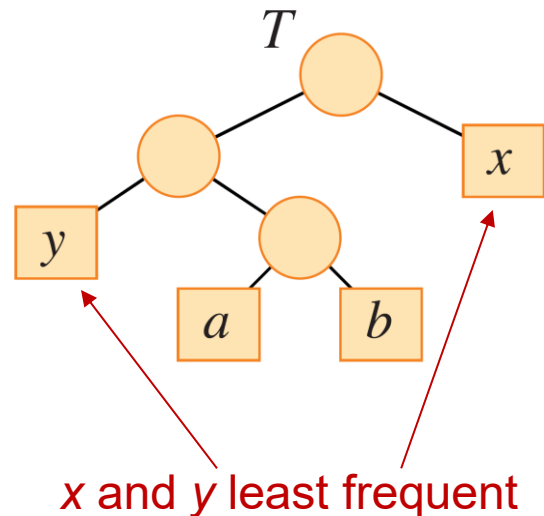
Time $O(n \cdot \log n)$

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

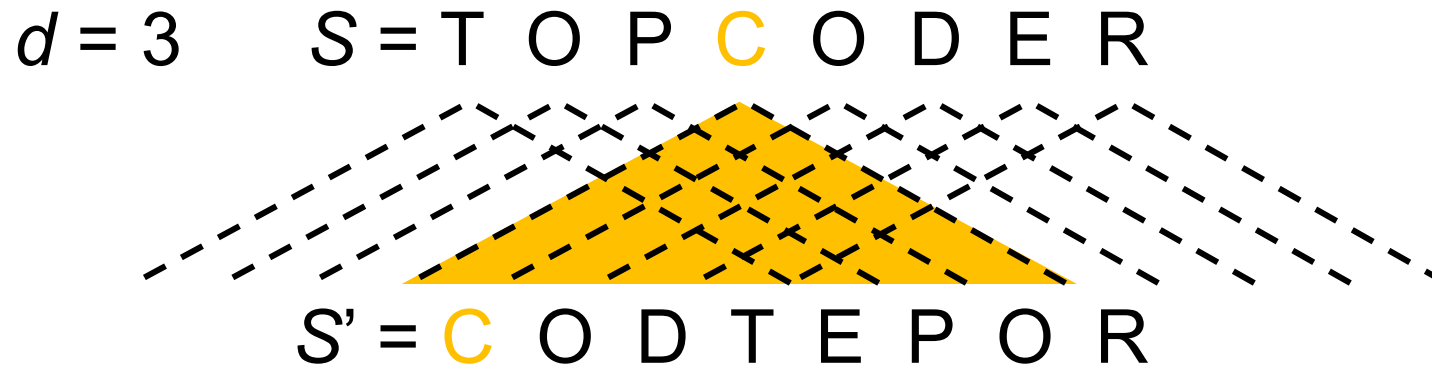


solve the problem with this as a leaf

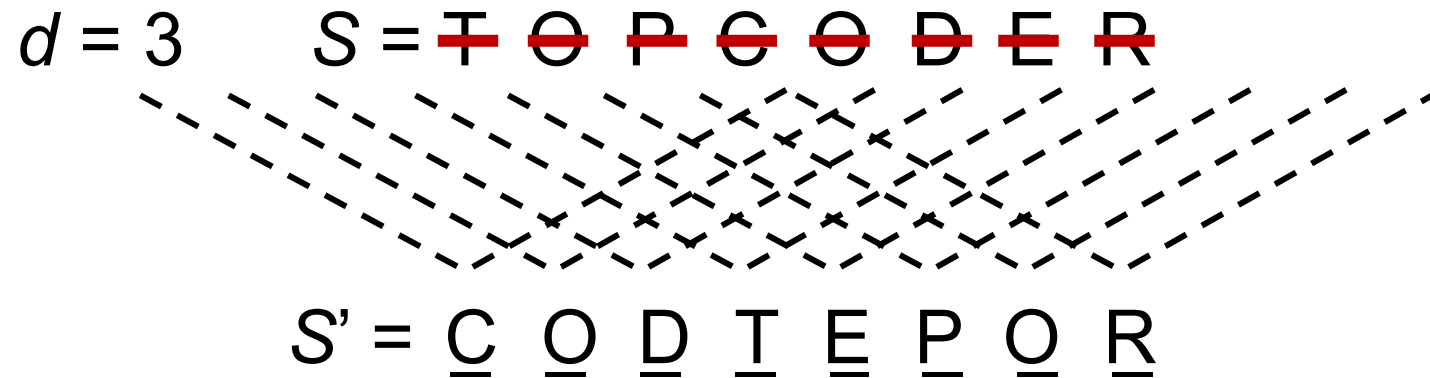
[tco14] TopCoder Open 2014 – Round 1A (500 point)

Input A string S , distance d

Output A string S' = lexicographically smallest permutation of S , where no character is moved more than d positions



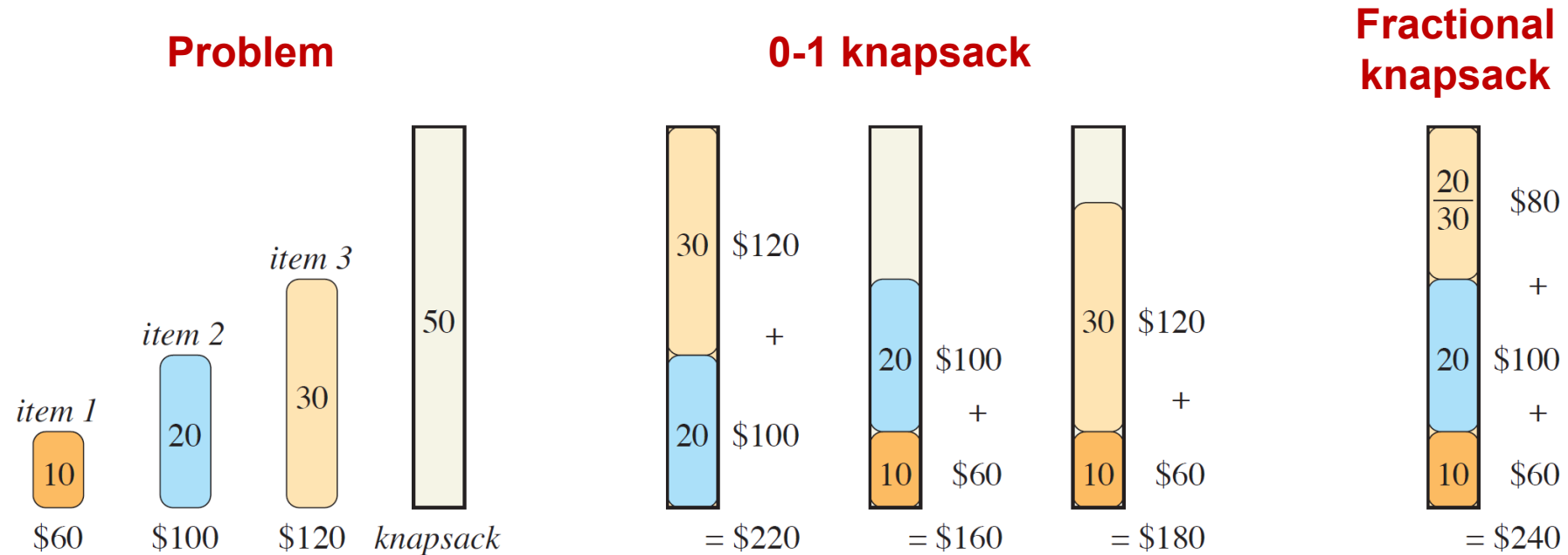
Greedy solution



Algorithm Construct S' from left to right

- take the character furthest to the left in the window, if not used
- otherwise take lexicographically smallest symbol in window (leftmost is multiple)

Dynamic programming vs greedy

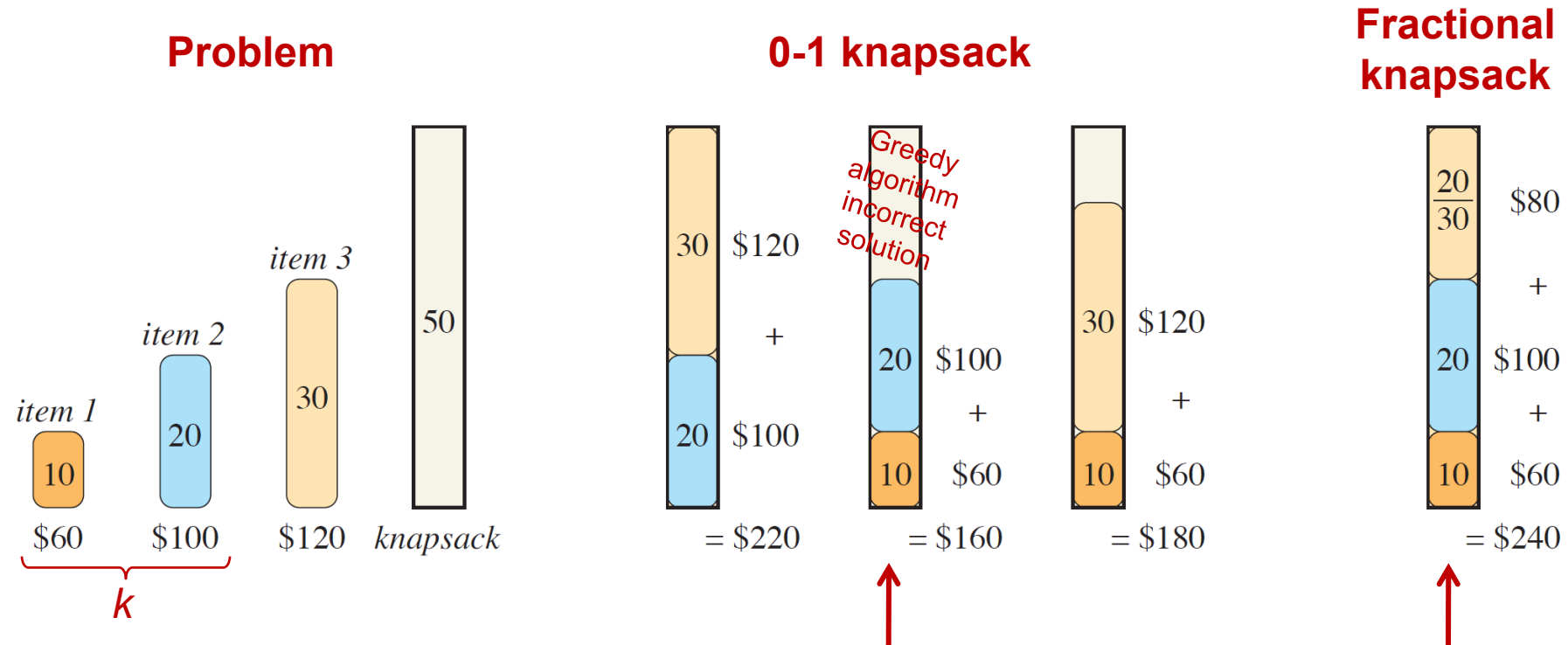


0-1 knapsack – maximum value for volume 11 ?

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

Dynamic programming vs greedy



$c(k, v)$ = best value among k first items for volume v

$$= \begin{cases} 0 & \text{if } k = 0 \\ c(k-1, v) & \text{if volume}_k > v \\ \max \{ c(k-1, v), \text{value}_k + c(k-1, v - \text{volume}_k) \} & \text{if volume}_k \leq v \end{cases}$$

Greedy fill by decreasing value/volume

0-1-knapsack

Volume	2	3	4	5
Value	3	5	4	6

$c(k,v)$		v											
		0	1	2	3	4	5	6	7	8	9	10	11
k	0	0	0	0	0	0	0	0	0	0	0	0	0
	1	0	0	3	3	3	3	3	3	3	3	3	3
	2	0	0	3	5	5	8	8	8	8	8	8	8
	3	0	0	3	5	5	8	8	9	9	12	12	12
	4	0	0	3	5	5	8	8	9	11	12	14	14

Systematic
tabulation

$c(k,v)$		v											
		0	1	2	3	4	5	6	7	8	9	10	11
k	0	0	0	0	0	0	0	0	0	0	0	-	0
	1	-	-	3	3	3	-	3	3	3	-	-	3
	2	-	-	3	-	-	-	8	8	-	-	-	8
	3	-	-	-	-	-	-	8	-	-	-	-	12
	4	-	-	-	-	-	-	-	-	-	-	-	14

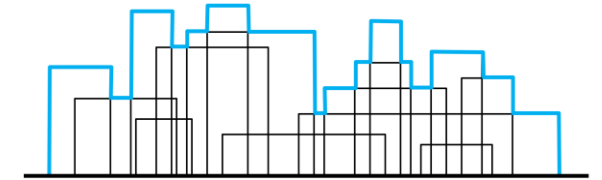
Recursion +
memoization

General algorithm design techniques

■ Divide and conquer

- Disjoint subproblems

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



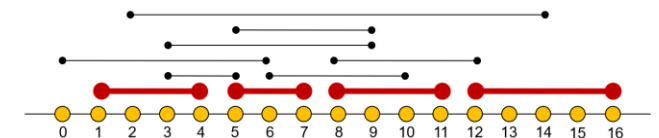
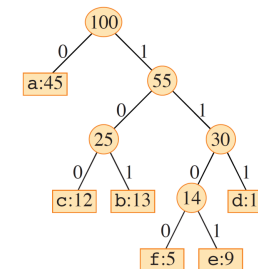
■ Dynamic programming

- Subproblems overlap
- Remember already computed results for subproblems
- Systematic tabulation or recursion with memoization

	j	0	1	2	3	4	5	6
i	y _j	B	D	C	A	B	A	
0	x _j	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

■ Greedy algorithms

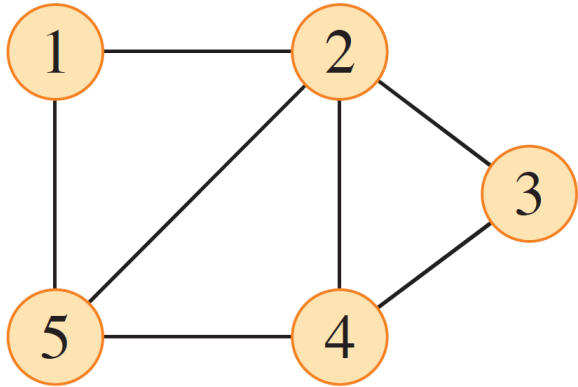
- Only one subproblem



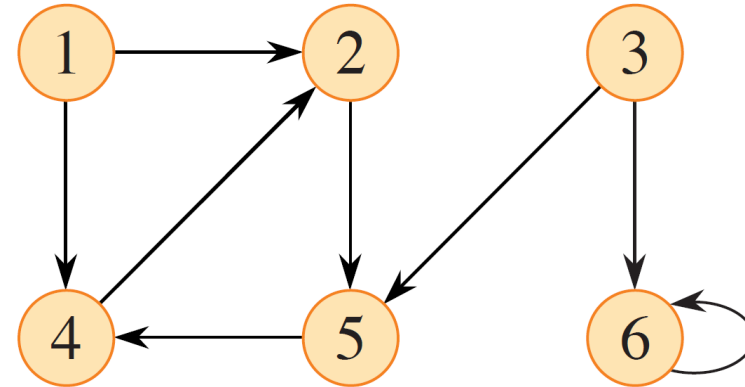
Algorithms and Data Structures

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

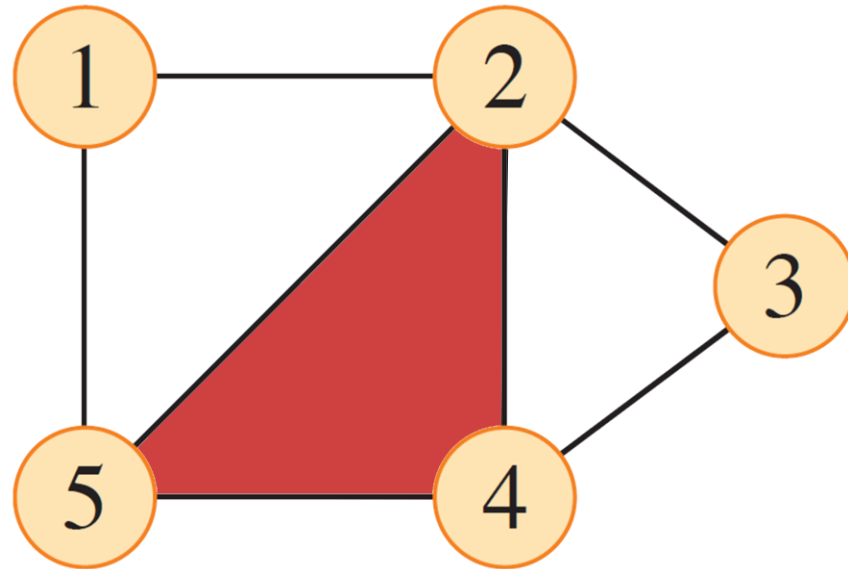
$E : \{u, v\}$ an undirected edge between u and v

(u, v) a directed edge from u to v

$n = |V| = \# \text{ nodes}$

$m = |E| = \# \text{ edges (connections between nodes)}$

Planar graphs – Euler's formula



$$\begin{aligned} |V| &= 5 \\ |E| &= 7 \\ \# \text{ faces} &= 4 \end{aligned}$$

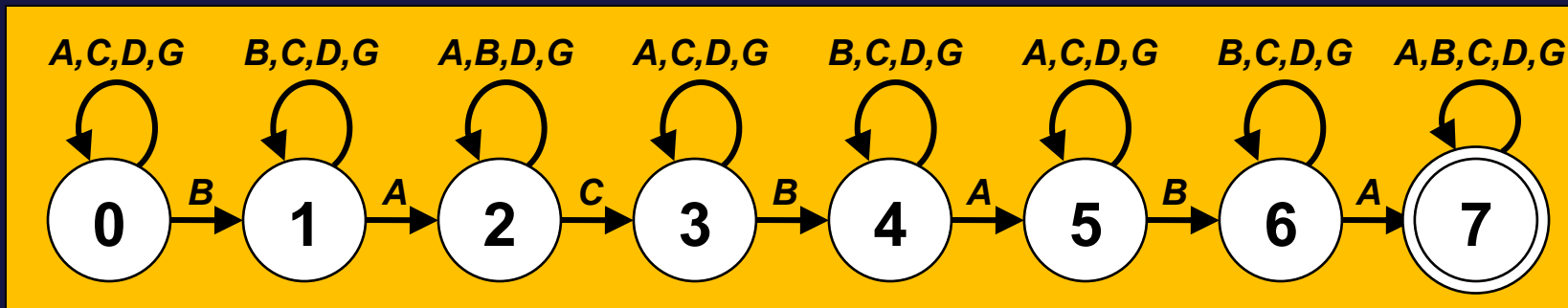
For a connected planar graph:

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



SUM

✖✔fx

=D9+C10-B10

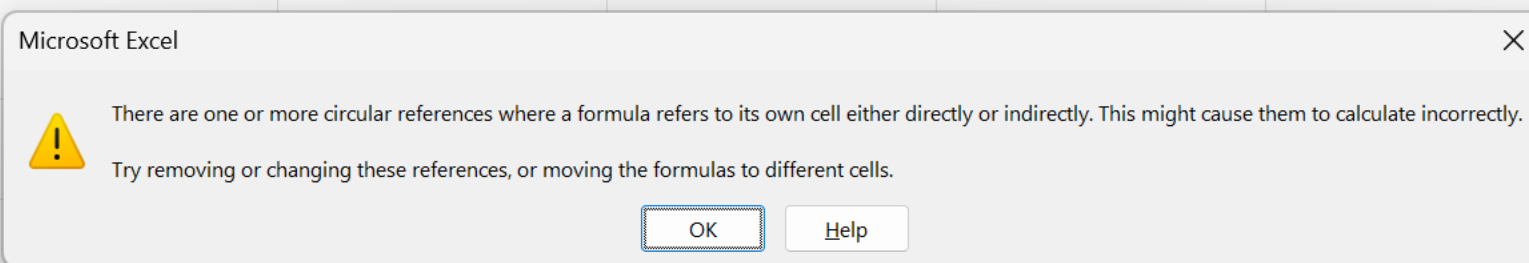
	A	B	C	D	E	F	G	H
1	Initial loan:	10000						
2	Yearly interest rate:	5%						
3	Yearly payment:	800						
4								
5	Payment number	Payment	Interest	Balance				
6				10000				
7	1	800	500	9700				
8	2	800	485	9385				
9	3	800	469	9054				
10	4	800	453	=D9+C10-B10				
11	5	800	435	8342				

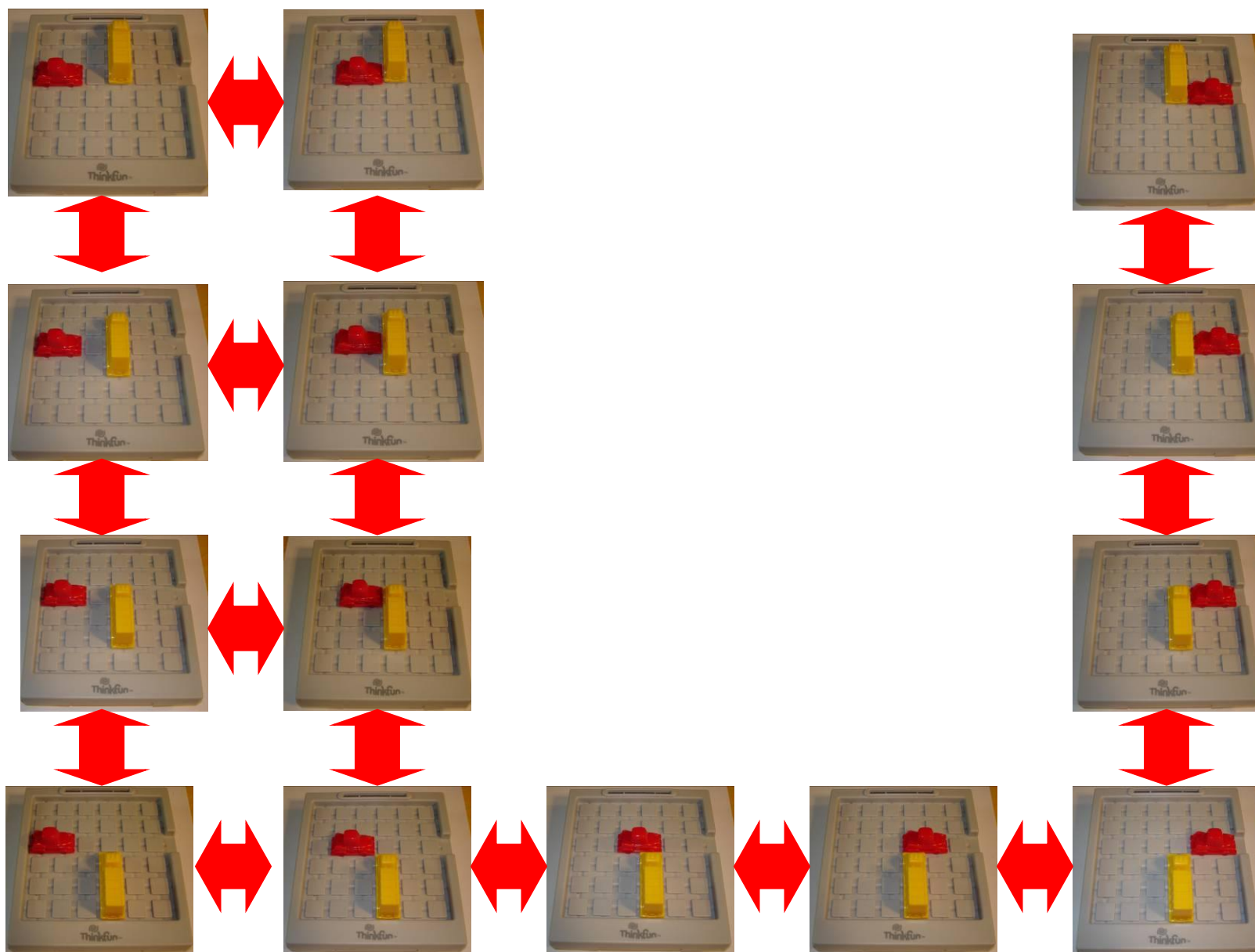
Which computation order ?

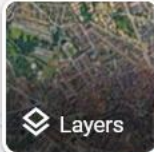
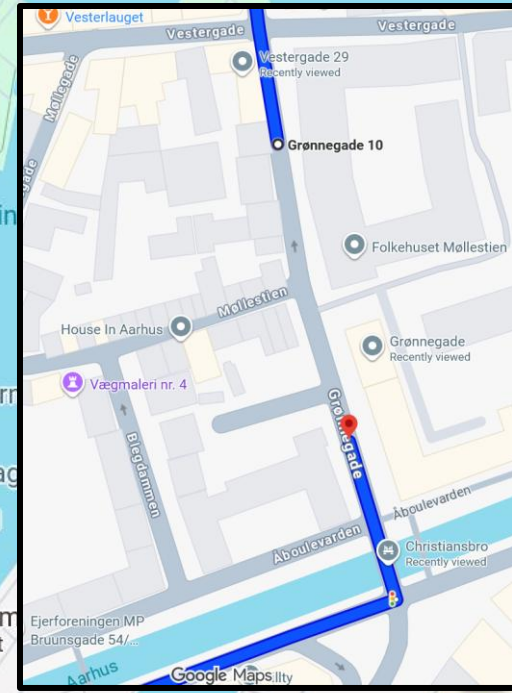
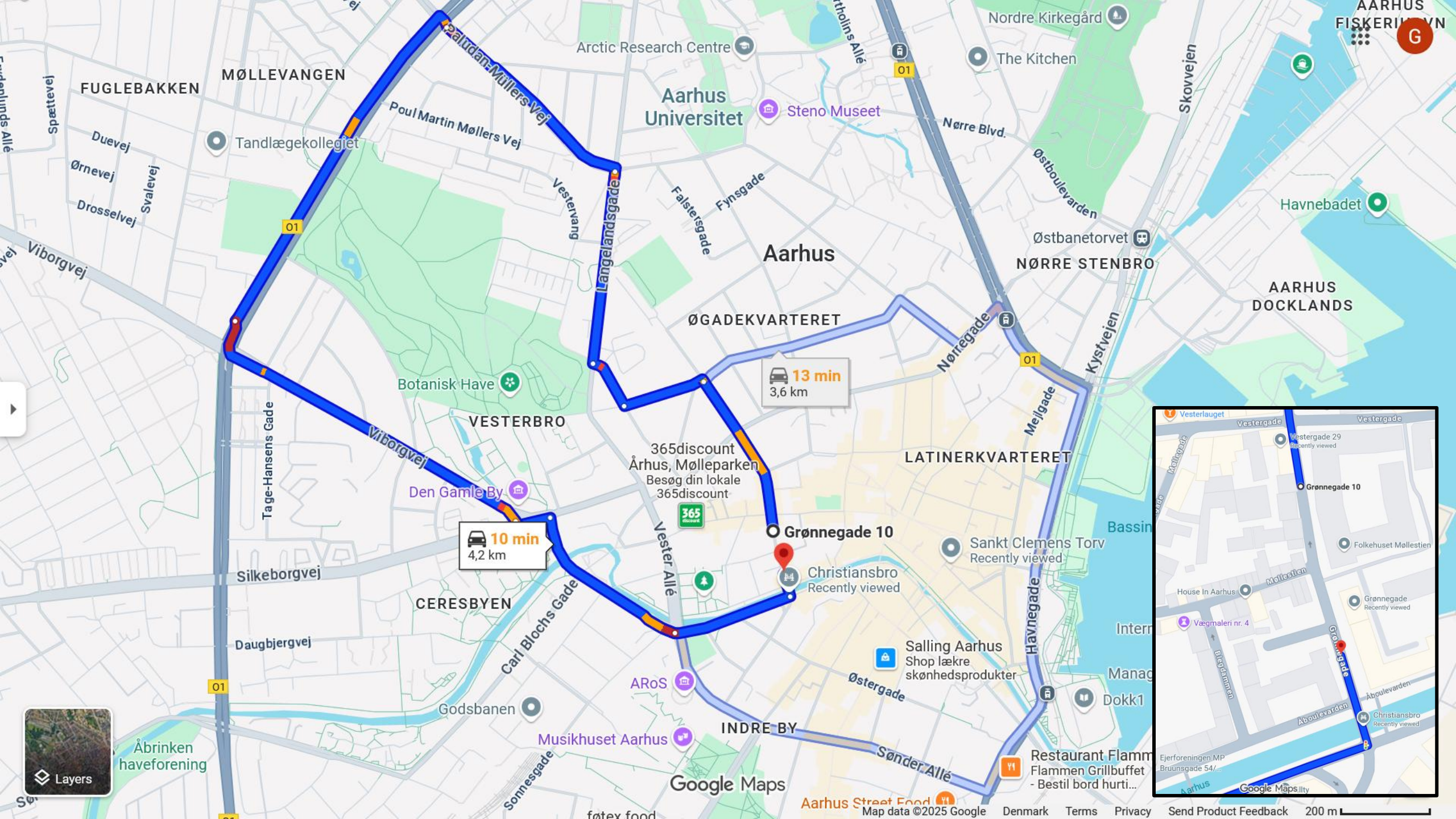
	A	B	C
1	10	20	= A1+B1
2	50	30	= A2+B2
3	= (A1+A2)/C3	= (B1+B2)/C3	= C1+C2

- a) C1 C2 A3 B3 C3
- b) A3 B3 C2 C1 C3
- c) C2 C1 C3 B3 A3
- d) Don't know

A1 $\downarrow$ $\vdots$ $\times$ $\checkmark$ f_x $\downarrow$ =B3+1







Layers

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



Search

Where is this?

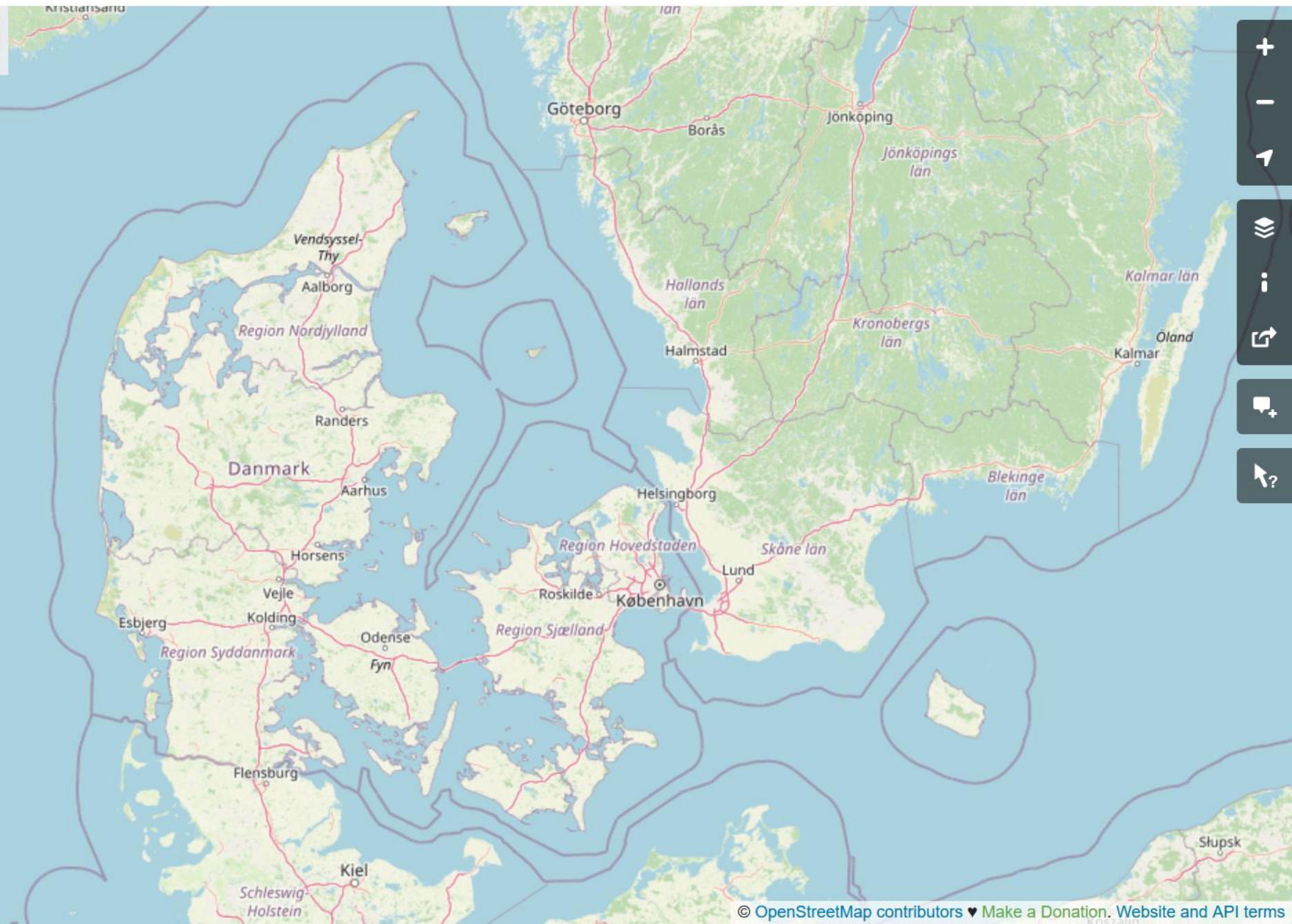


OpenStreetMap

Street network of Denmark

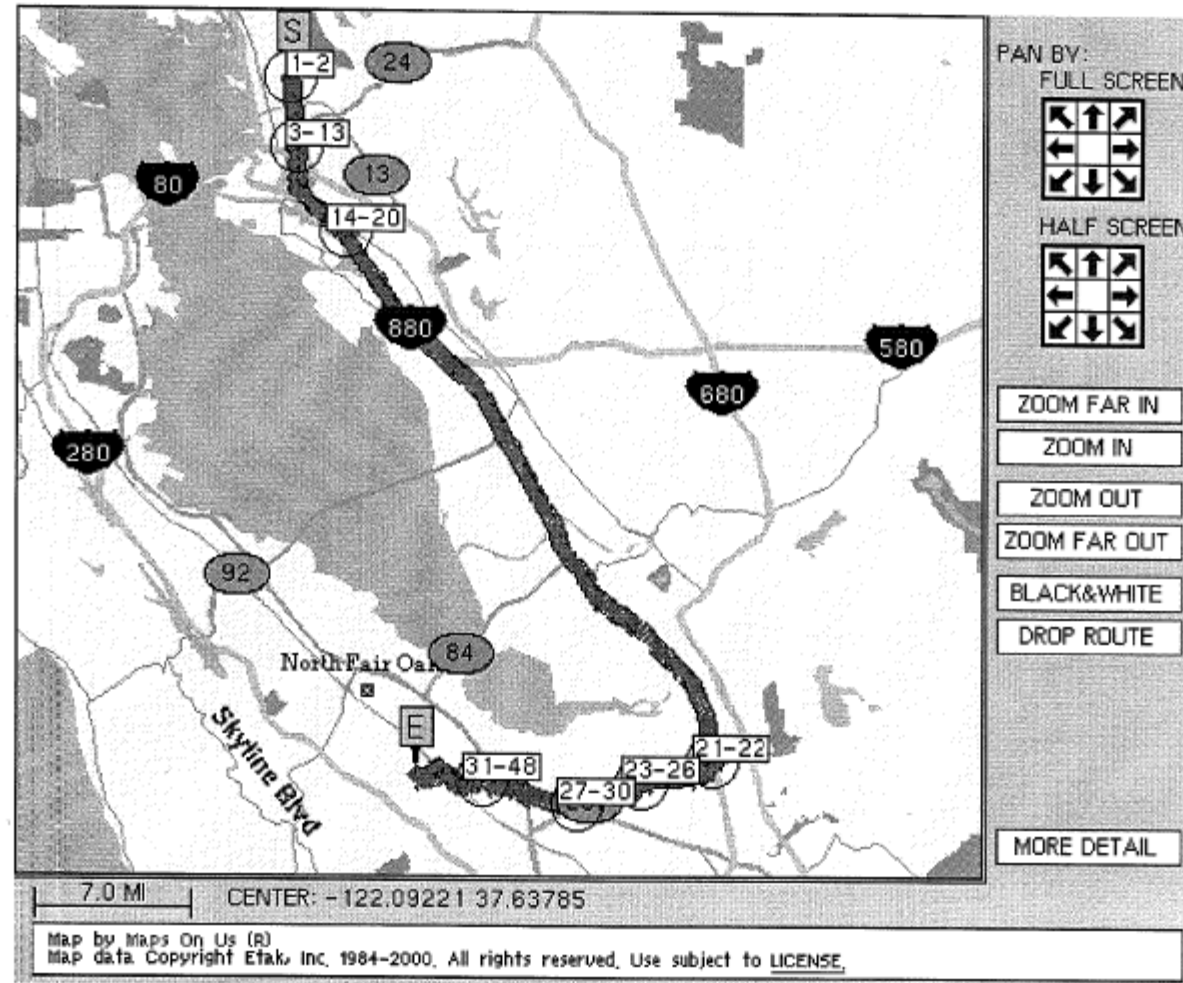
≈ 2 M vertices

≈ 4 M edges



50 km

30 mi



“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

Find connection: in Denmark 

A Skagen St. 

Departure 10.11 08:30 

B Rødby Færge 

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 

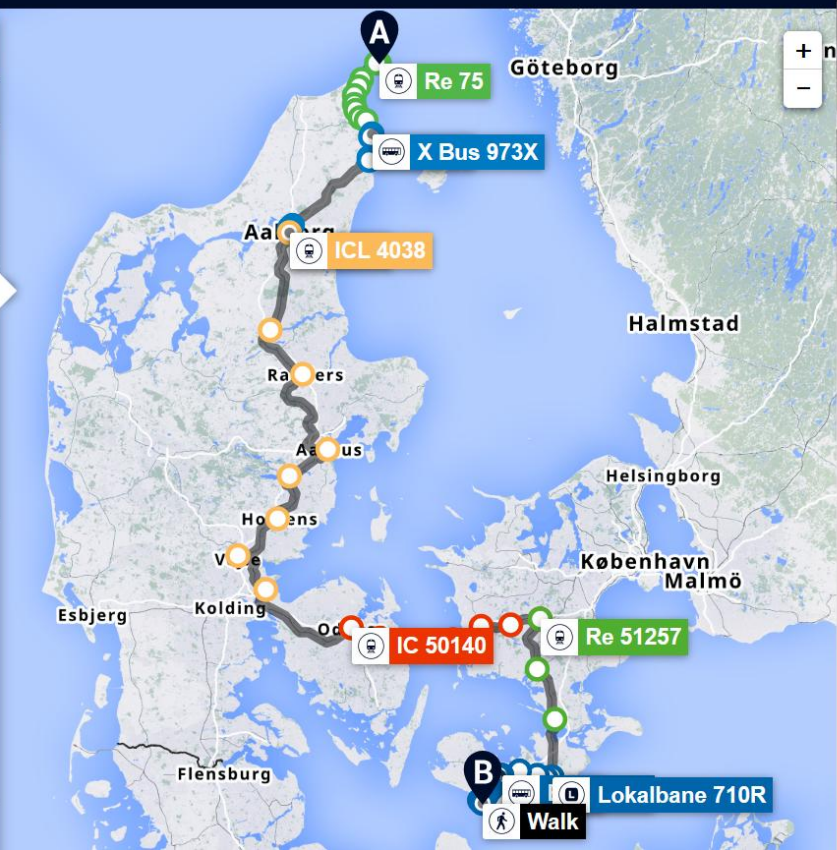
08:42          18:19

75 973X IC 4... IC 5... Re ... Re ... 710R 720R

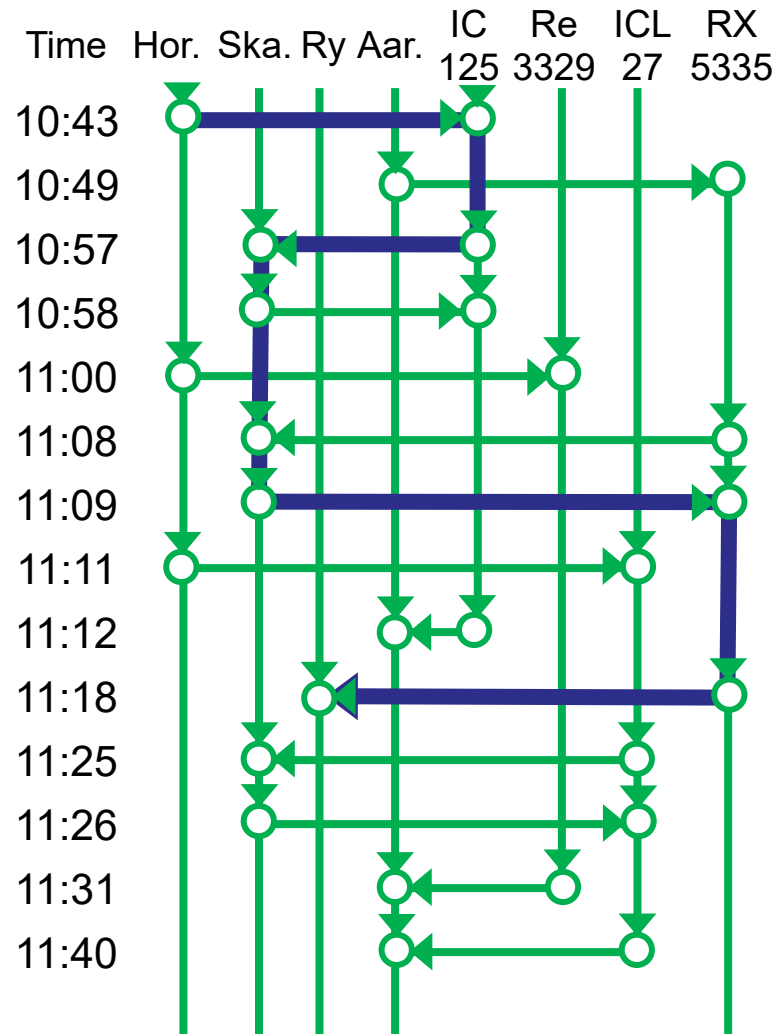
9 h 37 min., 7 Transfers

Price & buy

Details 



Travel plan (Horsens to Ry)



Algorithm

Find earliest node for **Ry** reachable from a given start node in **Horsens**

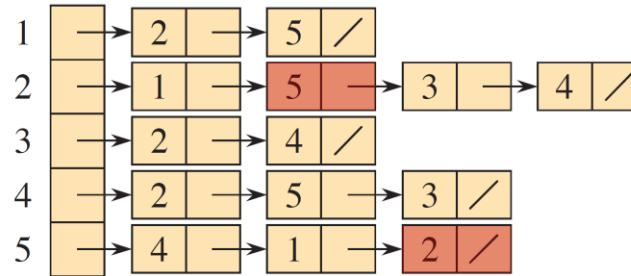
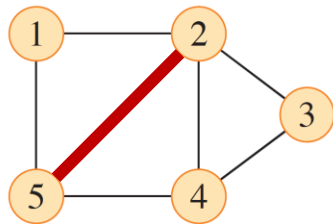
Train	Arr	Dep	Station
		10:43	Horsens
IC125	10:57	10:58	Skanderborg St
	11:12		Aarhus H
Re3329		11:00	Horsens
	11:31		Aarhus H
		11:11	Horsens
ICL27	11:25	11:26	Skanderborg St
	11:40		Aarhus H
		10:49	Aarhus H
RX5335	11:08	11:09	Skanderborg St
	11:18		Ry St



part of train schedule

Graph representations

- adjacency-lists and adjacency-matrices

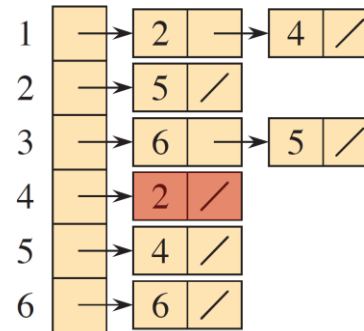
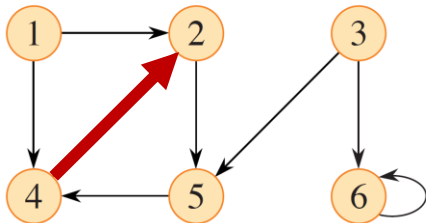


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Undirected graph

Space $O(n+m)$

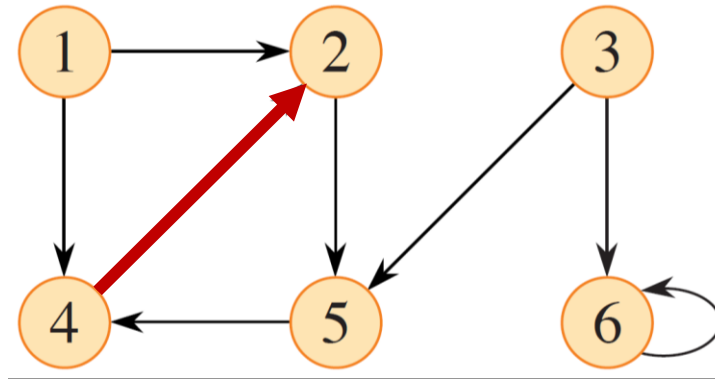
Space $O(n^2)$



	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)

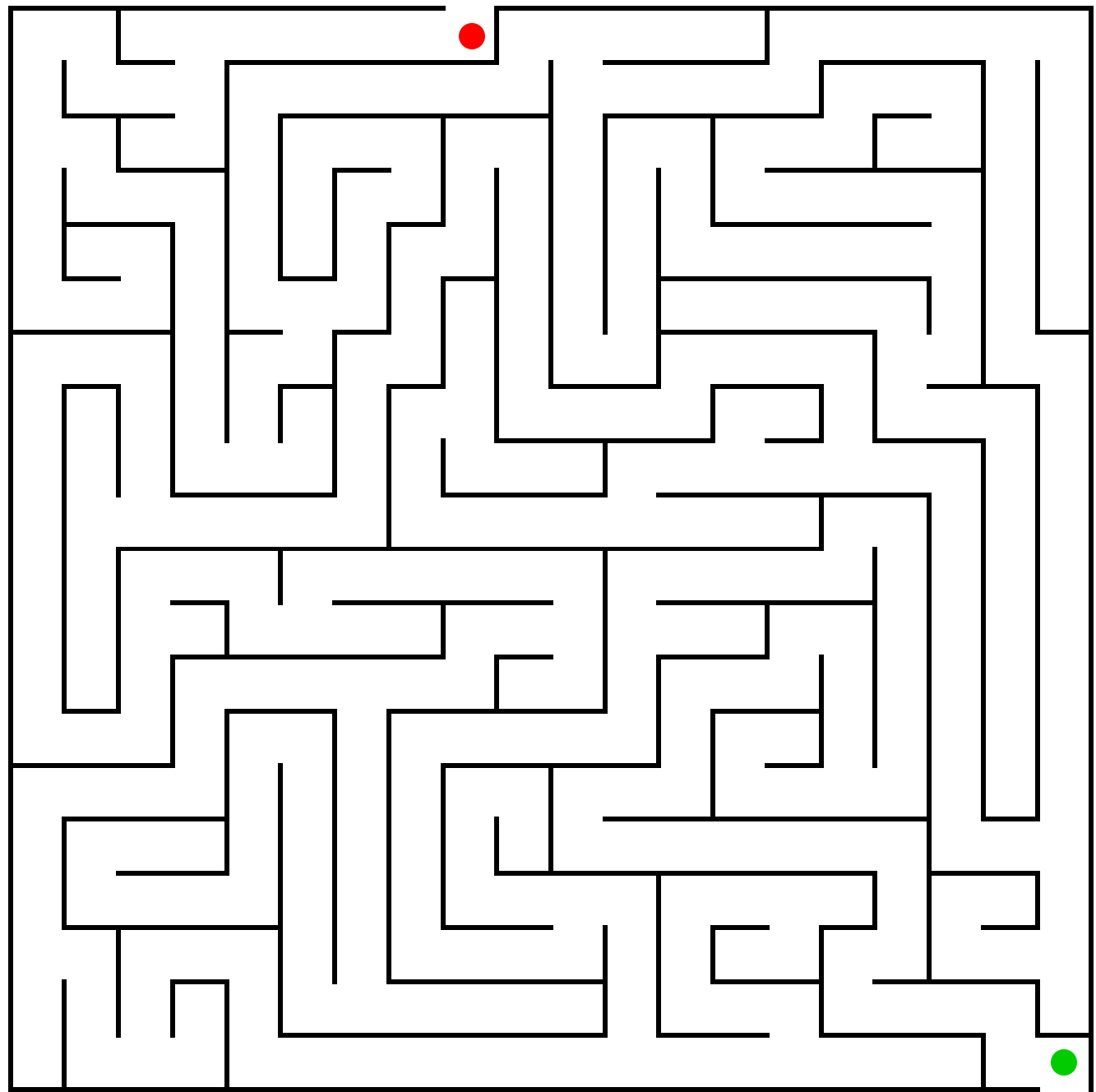
(1, 2)	(1, 4)	(2, 5)	(3, 5)	(3, 6)	(4, 2)	(5, 4)	(6, 6)
--------	--------	--------	--------	--------	--------	--------	--------

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

Red arrows indicate mappings from V to E: 1→2, 3→4, 4→5, 6→5, 7→6, 8→2.



Breadth first search (BFS)

BFS(G, s)

```

1  for each vertex  $u \in G.V - \{s\}$ 
2     $u.color = \text{WHITE}$ 
3     $u.d = \infty$ 
4     $u.\pi = \text{NIL}$ 
5   $s.color = \text{GRAY}$ 
6   $s.d = 0$ 
7   $s.\pi = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11    $u = \text{DEQUEUE}(Q)$ 
12   for each vertex  $v$  in  $G.Adj[u]$ 
13     if  $v.color == \text{WHITE}$ 
14        $v.color = \text{GRAY}$ 
15        $v.d = u.d + 1$ 
16        $v.\pi = u$ 
17       ENQUEUE( $Q, v$ )
18    $u.color = \text{BLACK}$ 
    
```

$u.color$

WHITE = nodes not visited yet

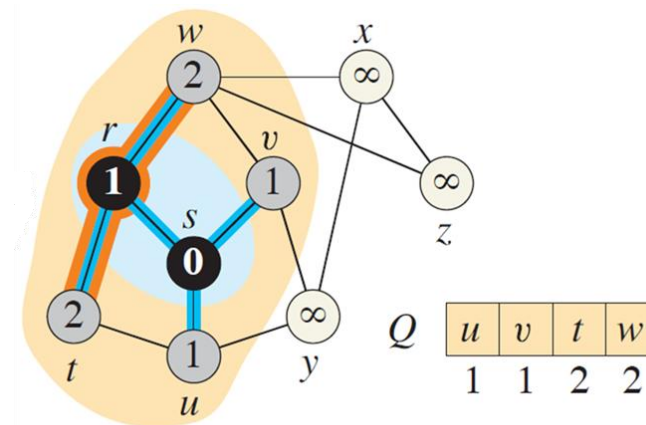
GRAY = nodes in queue Q

BLACK = nodes visited

$u.d$ = distance to s

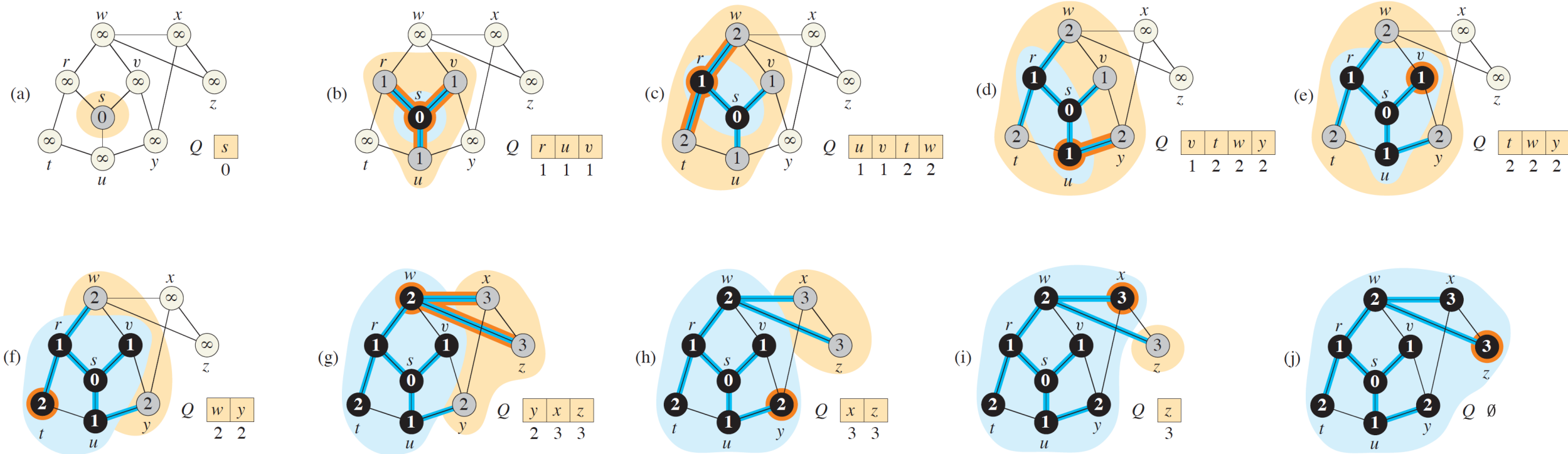
$u.\pi$ = parent to u in BFS tree

Q = queue with grey nodes
(which are connected to black nodes)



Time $O(n + m)$

BFS example

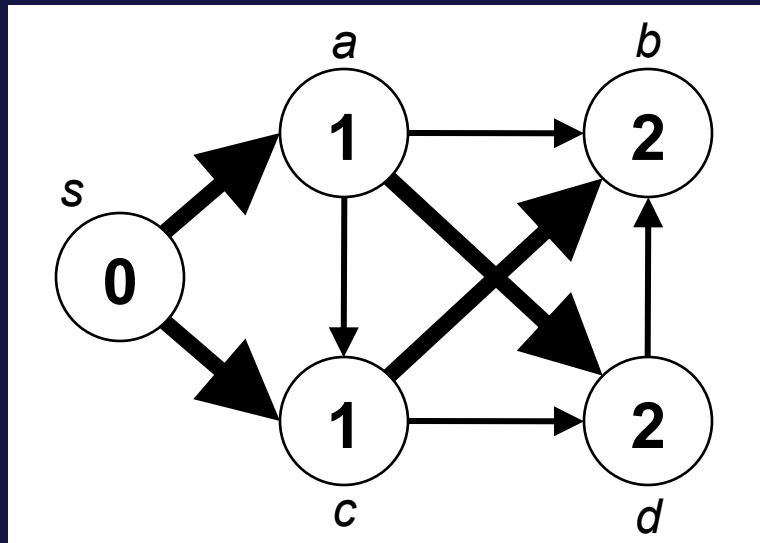


Is the below a valid BFS tree?

a) Yes

b) No

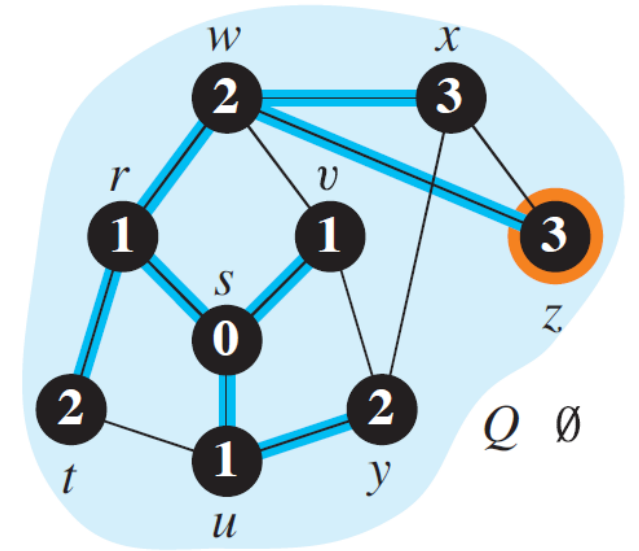
c) Don't know



BFS – print path from s to v

PRINT-PATH(G, s, v)

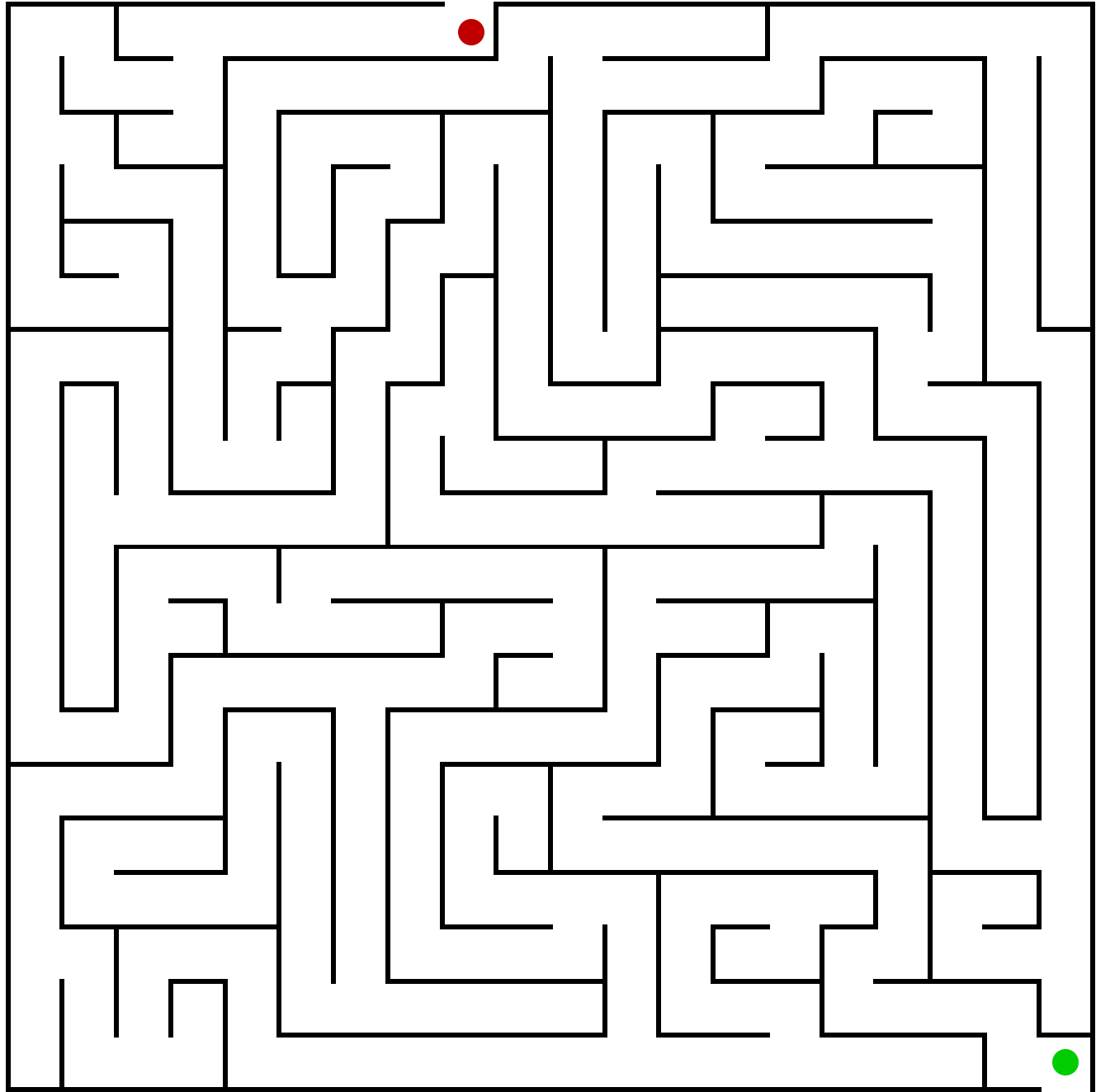
```
1  if  $v == s$ 
2      print  $s$ 
3  elseif  $v.\pi == \text{NIL}$ 
4      print “no path from”  $s$  “to”  $v$  “exists”
5  else PRINT-PATH( $G, s, v.\pi$ )
6      print  $v$ 
```



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  $u \in G.V$ 
2     $u.color = WHITE$ 
3     $u.\pi = NIL$ 
4   $time = 0$ 
5  for each vertex  $u \in G.V$ 
6    if  $u.color == WHITE$ 
7      DFS-VISIT( $G, u$ )
```

$u.color$

WHITE = nodes not yet visited

GRAY = nodes on recursion stack

BLACK = nodes visited

$u.\pi$ = parent of u in DFS tree

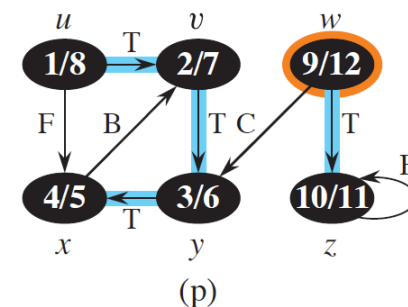
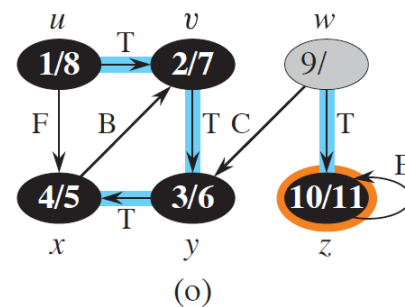
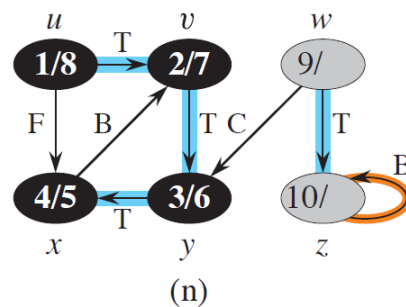
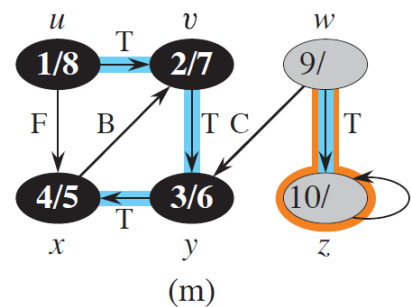
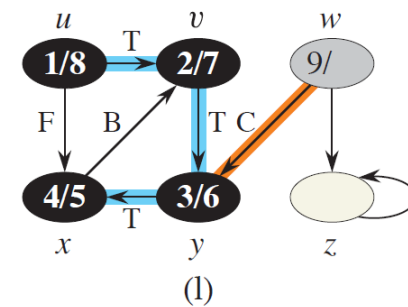
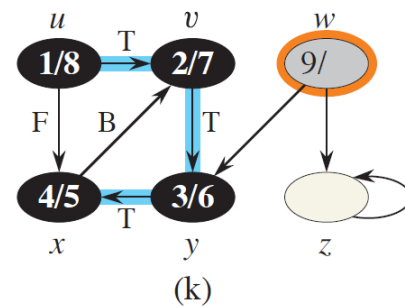
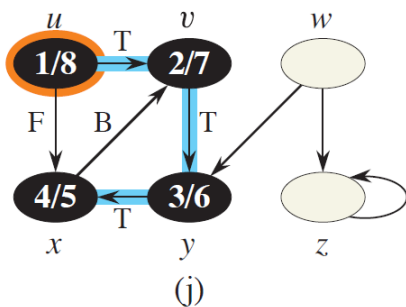
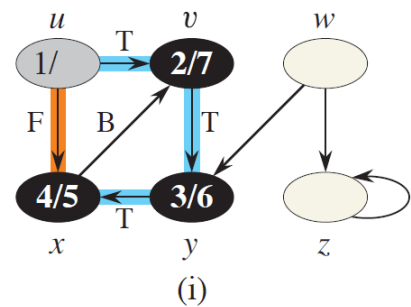
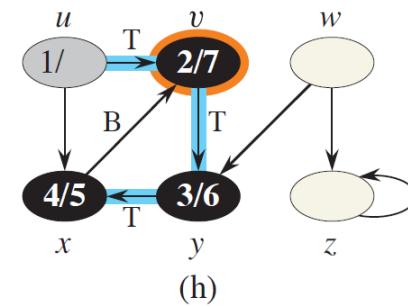
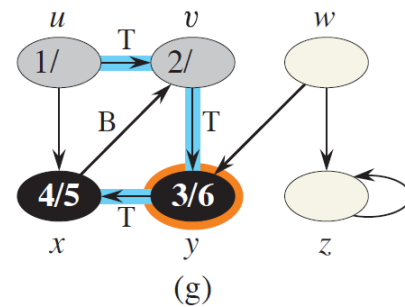
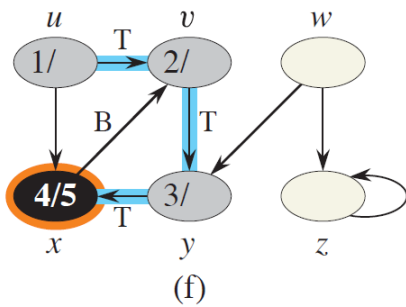
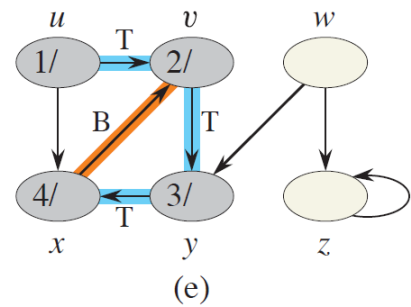
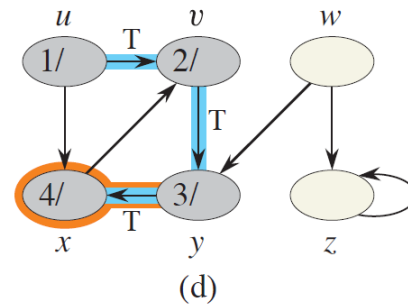
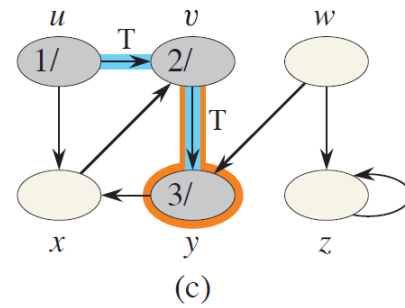
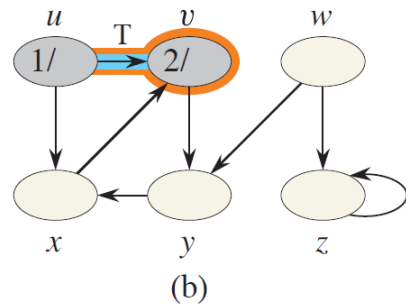
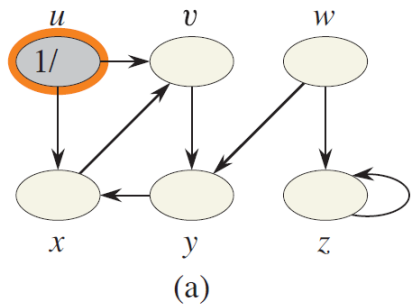
$u.d$ = discovery time for u

$u.f$ = finishing time for u

DFS-VISIT(G, u)

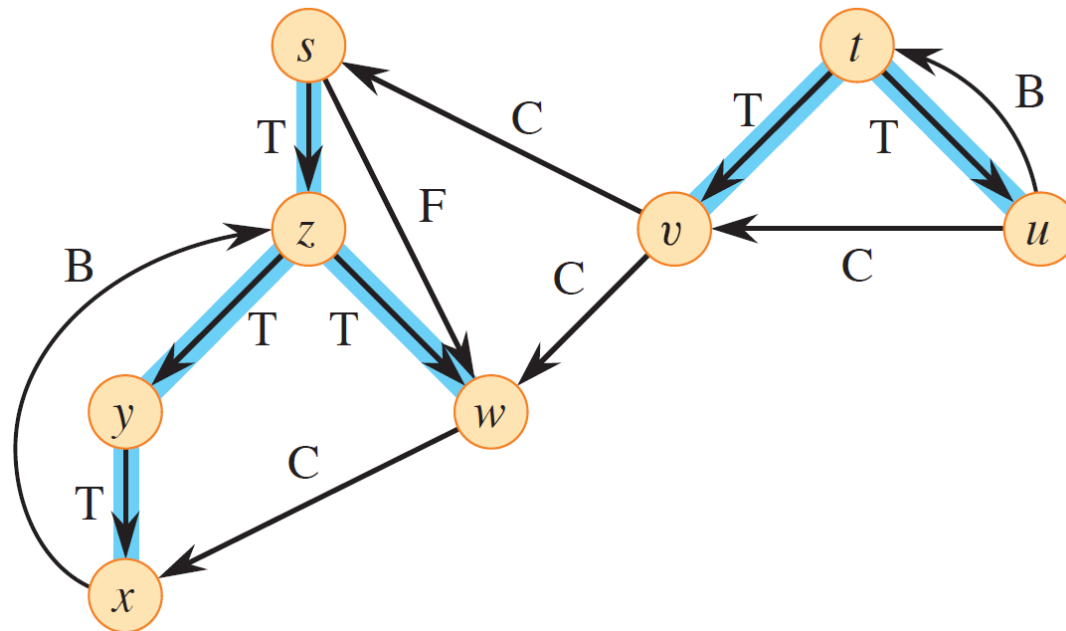
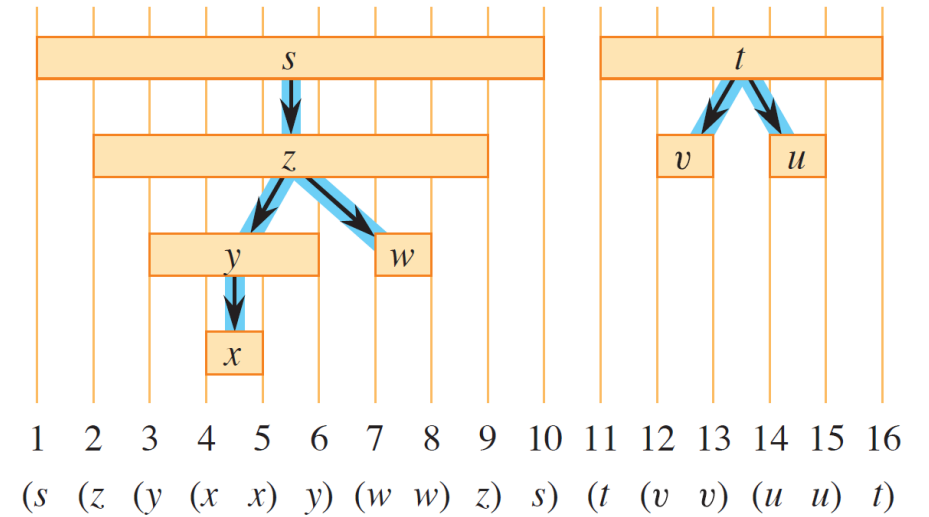
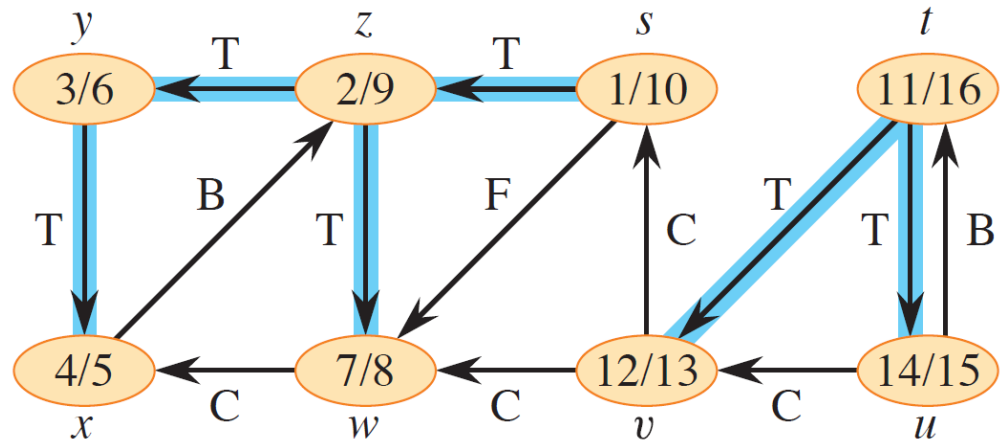
```
1   $time = time + 1$ 
2   $u.d = time$ 
3   $u.color = GRAY$ 
4  for each vertex  $v$  in  $G.Adj[u]$ 
5    if  $v.color == WHITE$ 
6       $v.\pi = u$ 
7      DFS-VISIT( $G, v$ )
8   $time = time + 1$ 
9   $u.f = time$ 
10  $u.color = BLACK$ 
```

Time $O(n + m)$



**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

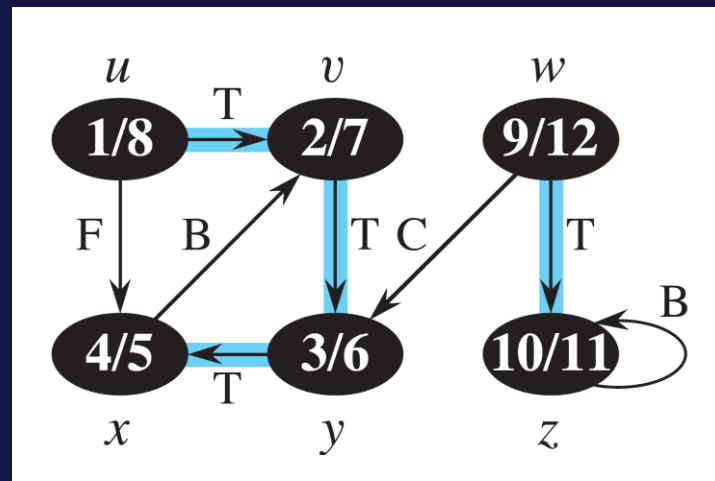
Find distances from start node
(distance = # edges, e.g., RushHour state space)

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



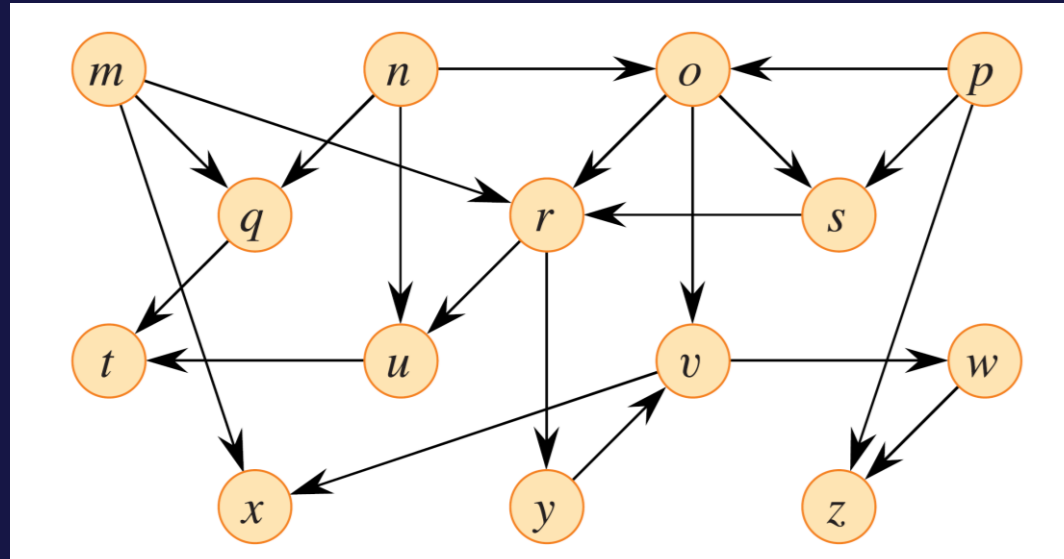
Assume there exists a path $a \rightsquigarrow b$ in a directed graph
without cycles

Is it always true that $a.f > b.f$?

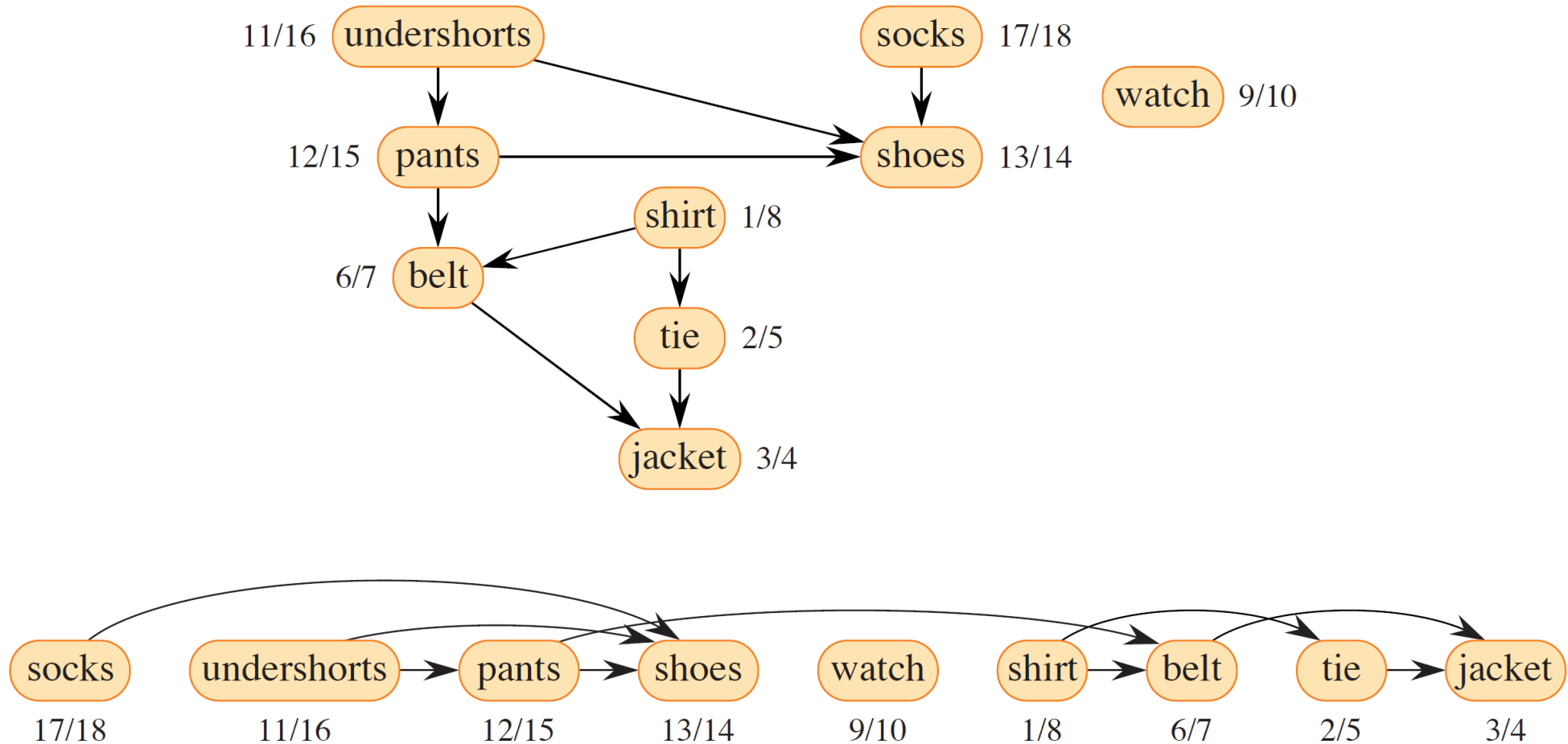
a) Yes

b) No

c) Don't know



Topological sorting of acyclic graphs



All edges are left-to-right

loan.xlsx • Saved

Search

File Home Insert Draw Page Layout Formulas Data Review View Automate Help Acrobat

Comments Share

SUM : $\times$ $\checkmark$ f_x =D9+C10-B10

	A	B	C	D	E	F	G	H
1	Initial loan:	10000						
2	Yearly interest rate:	5%						
3	Yearly payment:	800						
4								
5	Payment number	Payment	Interest	Balance				
6				10000				
7		1	800	500	9700			
8		2	800	485	9385			
9		3	800	469	9054			
10		4	800	453	=D9+C10-B10			
11		5	800	435	8342			

loan

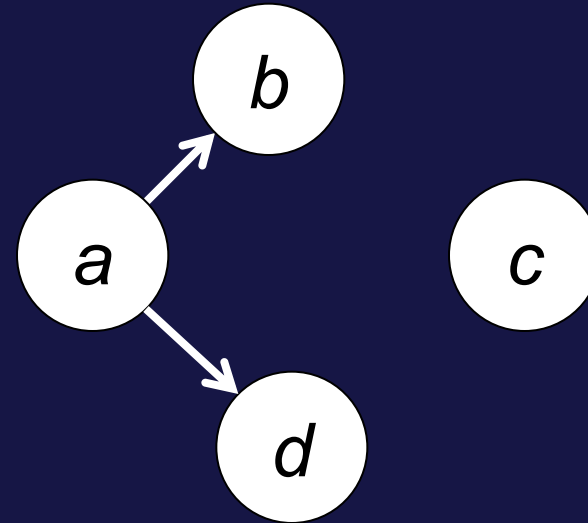
Edit Accessibility: Good to go

230%

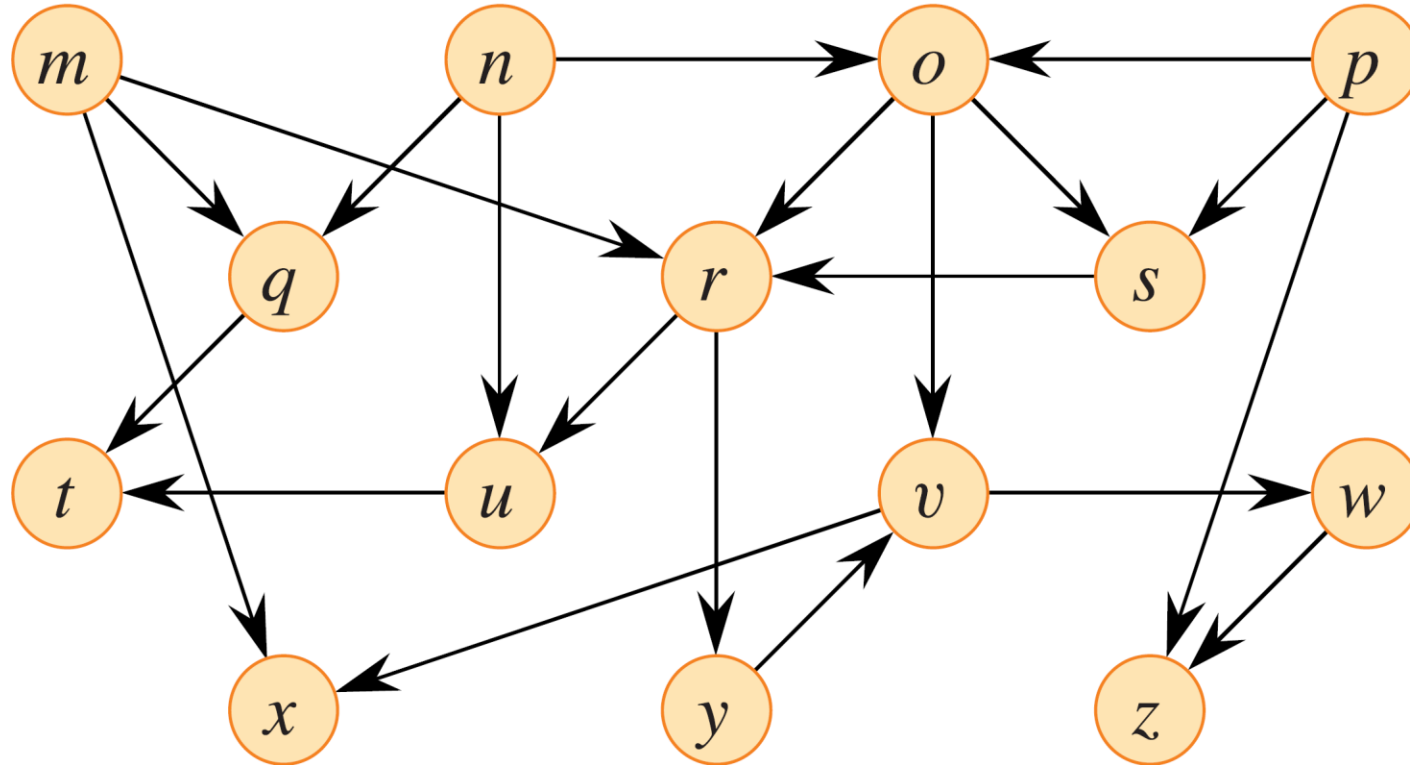
Topological sorting = a valid order to compute cells values

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



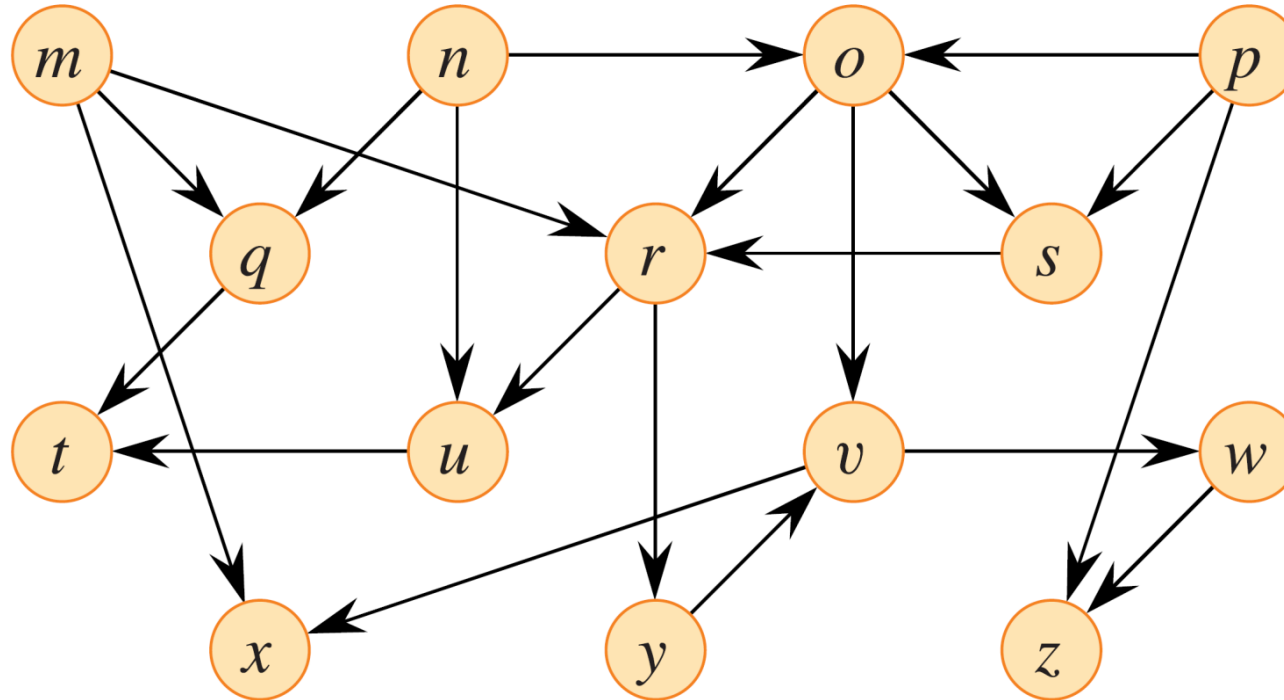
Topological sorting (I)



Algorithm Greedily remove a node of indegree 0 (and its outgoing edges) and add the node last in the topological order

Time $O(m + n)$

Topological sorting (II)



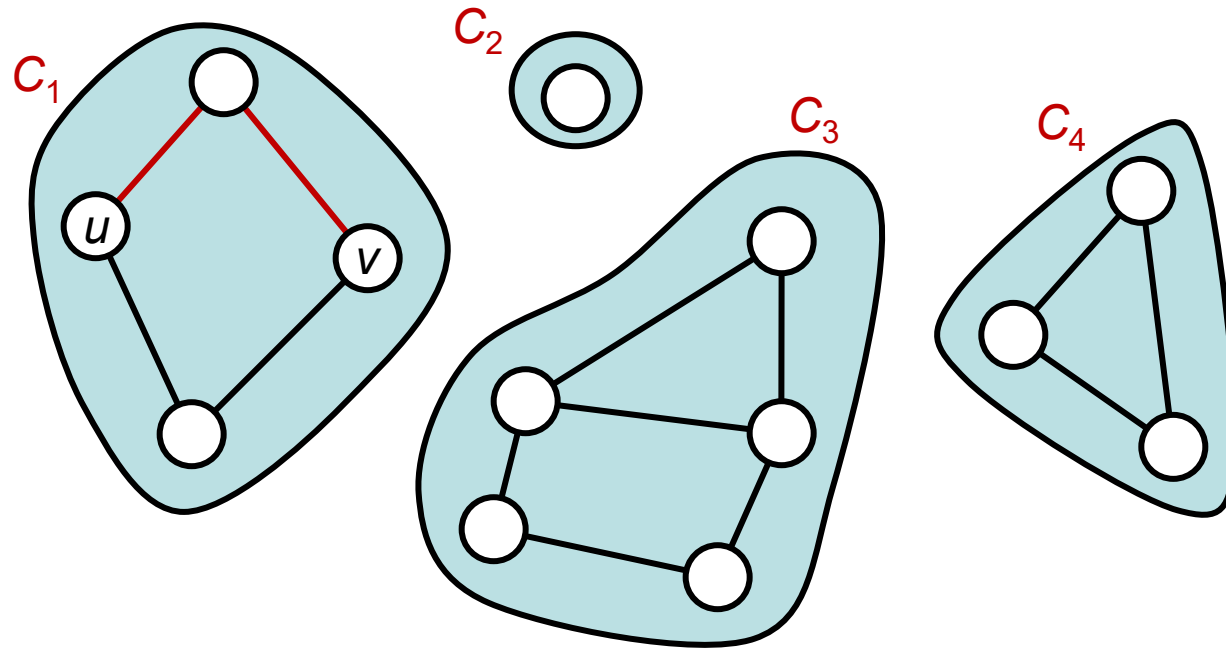
TOPOLOGICAL-SORT(G)

- 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 $O(m + n)$

Connected components

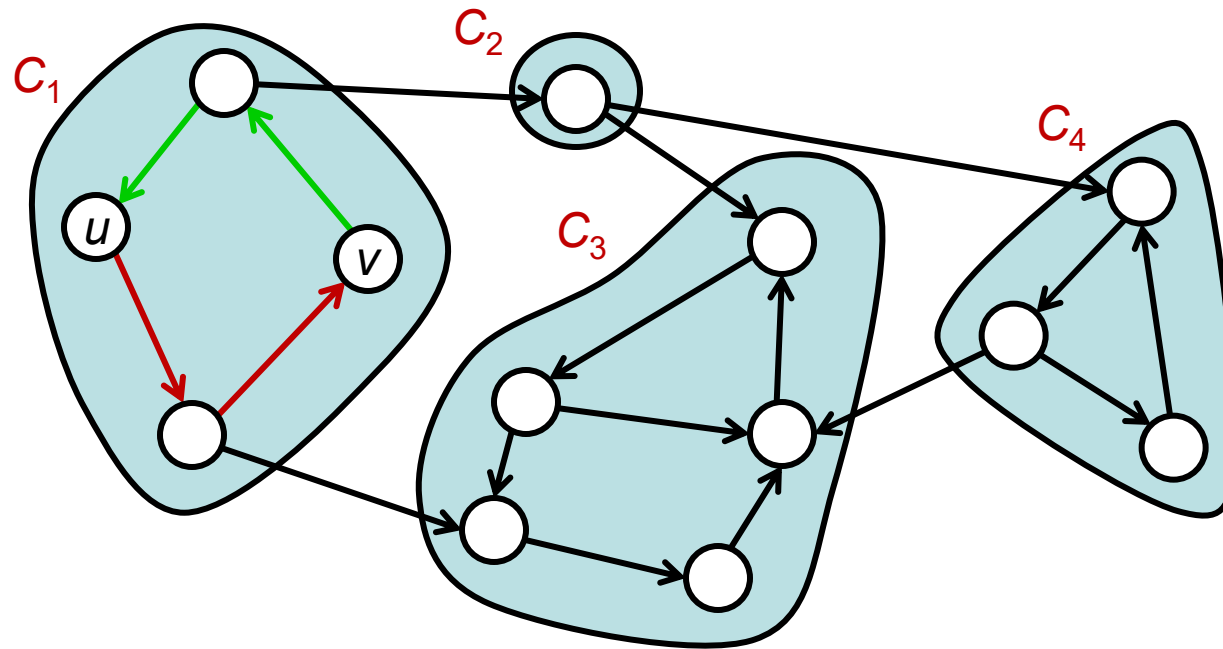
Partitioning of the nodes of an **undirected** graph into **components** $C_1, \dots, C_k$, such that u and v are in the same component $\Leftrightarrow$ there is a **path** between u and v



DFS/BFS time $O(m + n)$

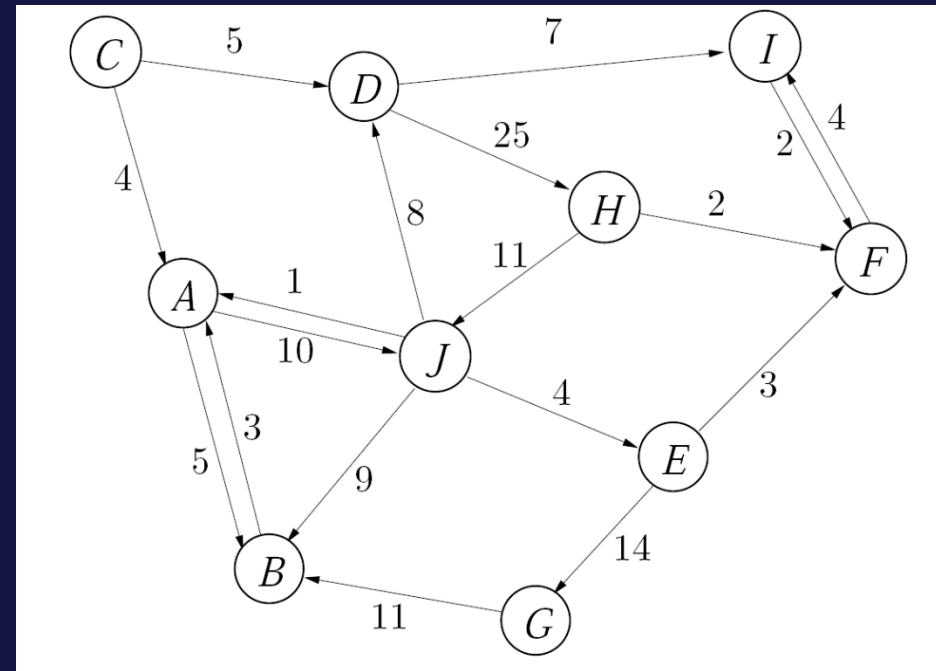
Strongly connected components

Partitioning of the nodes of a **directed** graph into **components** $C_1, \dots, C_k$, such that u and v are in the same component $\Leftrightarrow$ 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)

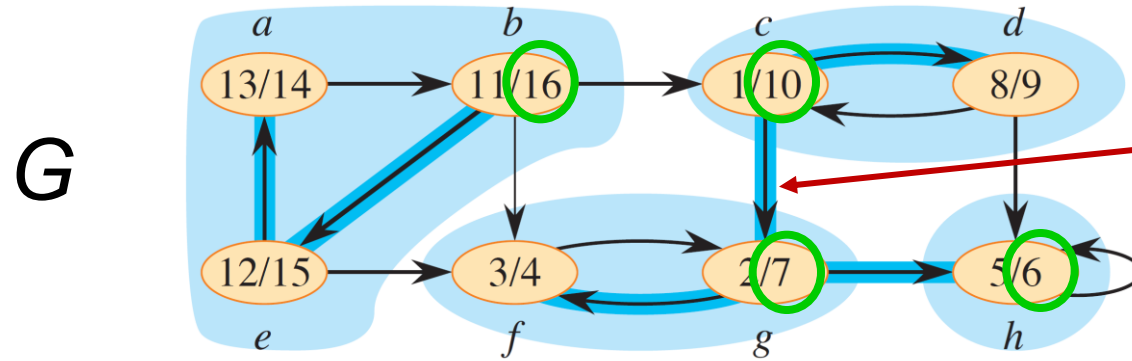
```
1 for each vertex  $u \in G.V$ 
2    $u.color = \text{WHITE}$ 
3    $u.\pi = \text{NIL}$ 
4  $time = 0$ 
5 for each vertex  $u \in G.V$ 
6   if  $u.color == \text{WHITE}$ 
7     DFS-VISIT( $G, u$ )
```

DFS-VISIT(G, u)

```
1  $time = time + 1$ 
2  $u.d = time$ 
3  $u.color = \text{GRAY}$ 
4 for each vertex  $v$  in  $G.Adj[u]$ 
5   if  $v.color == \text{WHITE}$ 
6      $v.\pi = u$ 
7     DFS-VISIT( $G, v$ )
8  $time = time + 1$ 
9  $u.f = time$ 
10  $u.color = \text{BLACK}$ 
```

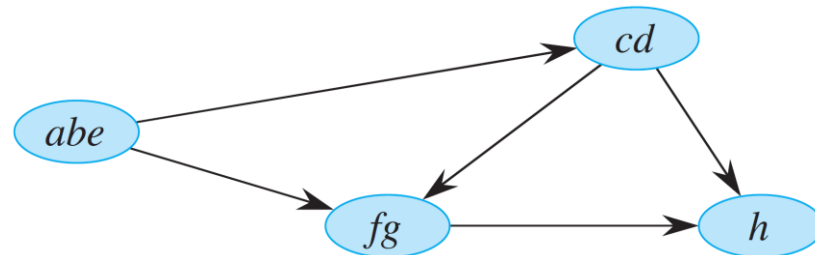
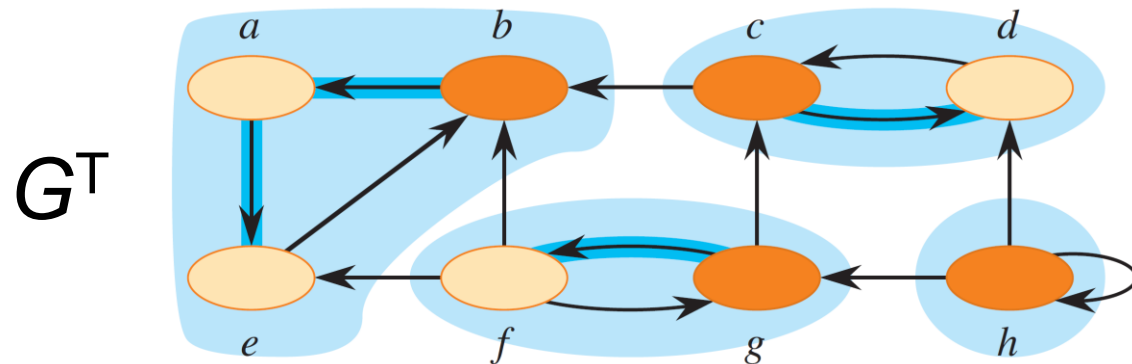
Time $O(m + n)$

Strongly connected components



DFS tree edges between two strongly connected components

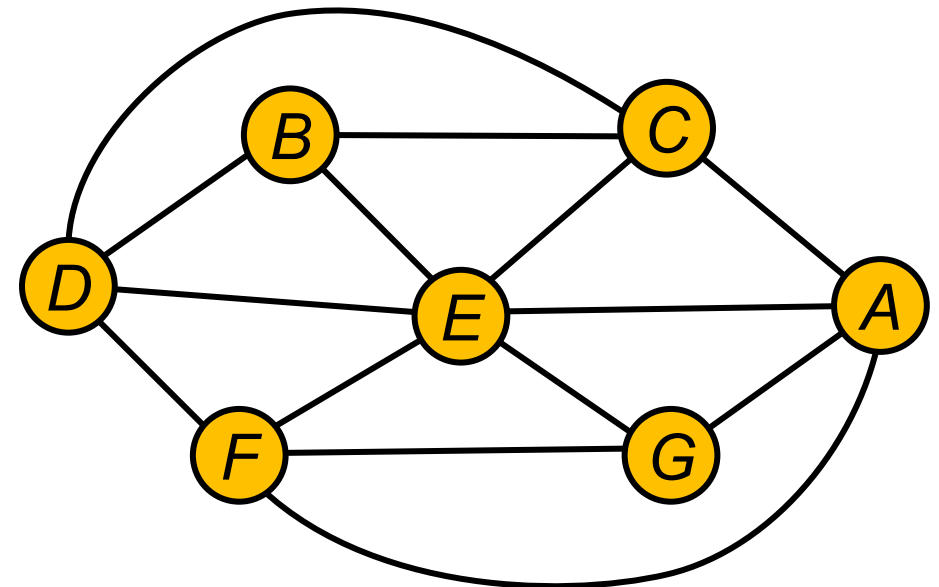
The largest finishing times in each component form a (reversed) topological sorting of the components

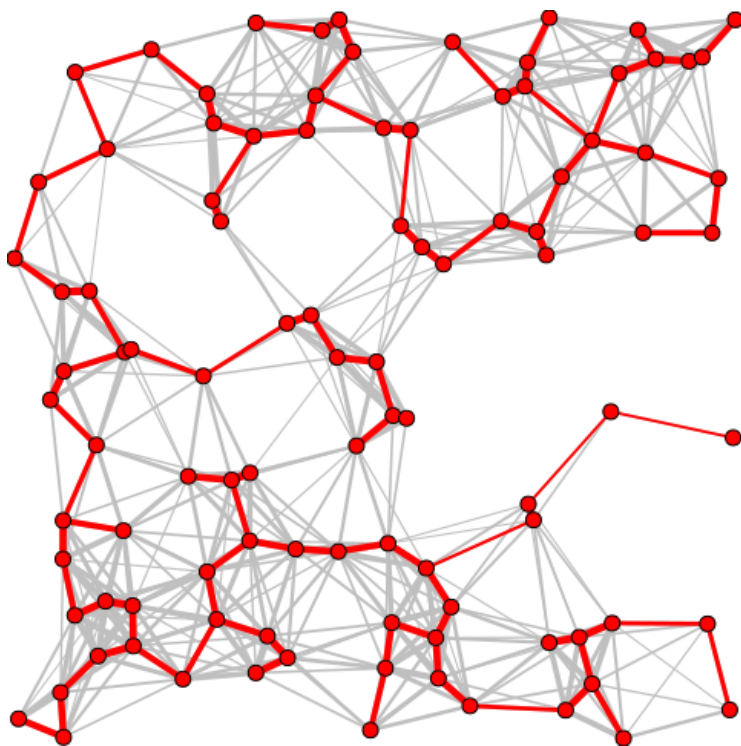


Planarity testing

- Decide if an undirected graph is planar, i.e., can be drawn in the plane without crossing edges
- Can be solved using DFS (nontrivial)

$E = \{\{A, C\}, \{A, E\}, \{A, F\}, \{A, G\}, \{B, C\},$
 $\{B, D\}, \{B, E\}, \{C, D\}, \{C, E\}, \{D, E\},$
 $\{D, F\}, \{E, F\}, \{E, G\}, \{F, G\}\}$





Algorithms and Data Structures

Minimum spanning trees (MST)
[CLRS, Chapter 21]

Dr. OTAKAR BORŮVKA:

Příspěvek k řešení otázky ekonomické stavby elektrovodných sítí.

Ve své práci „O jistém problému minimálním“*) odvodil jsem obecnou větu, již jest ve zvláštním případě řešena tato úloha:

V rovině (v prostoru) jest dáno n bodů, jejichž vzájemné vzdálenosti jsou vesměs různé. Jest je spojití sítí tak, aby:

příkladem vyložím. Čtenáře, jenž by se o věc blíže zajímal, odkazuji na citované pojednání.

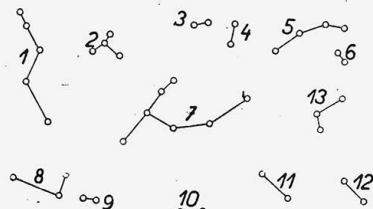
Řešení úlohy provedu v případě 40 bodů daných v obr. 1.

Každý z daných bodů spojím s bodem nejbližším. Tedy na př. bod 1 s bodem 2, bod 2 s bodem 3, bod 3



Obr. 1.

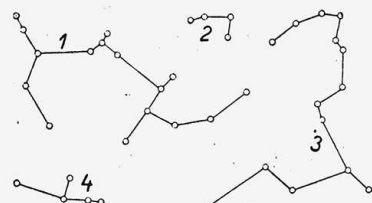
1. každé dva body byly spojeny buď přímo anebo prostřednictvím jiných,
2. celková délka sítě byla co nejmenší.



Obr. 2.

Jest zřejmé, že řešení této úlohy může mít v elektrotechnické praxi při návrzích plánů elektrovodných sítí jistou důležitost; z toho důvodu je zde stručně na

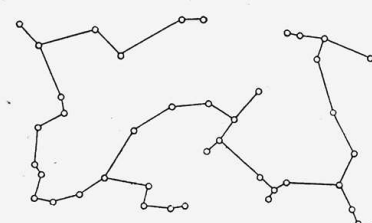
*) Vyjde v nejbližší době v Pracích Moravské přírodovědecké společnosti.



Obr. 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).

Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy na př. 1 s tahem 2, (tah 2 s ta-



Obr. 4.

Orazec převrácen.

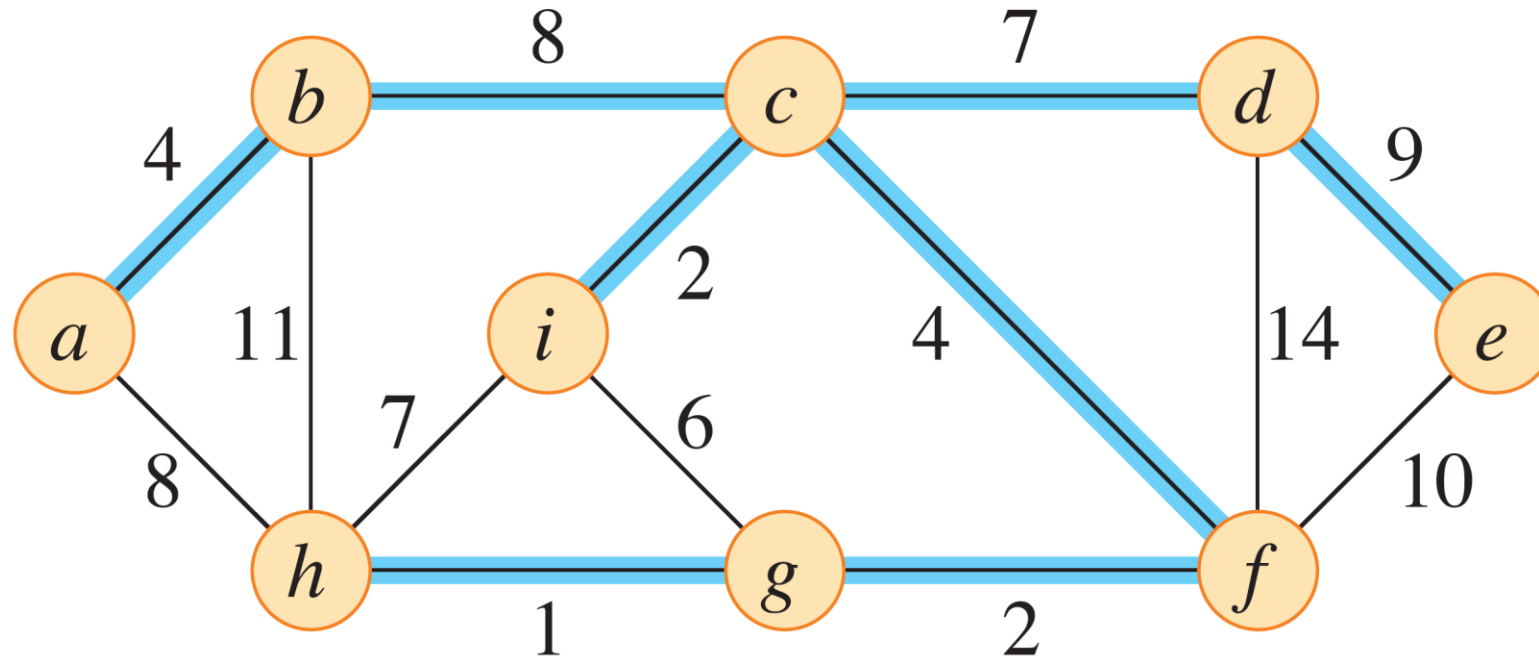
hem 1), tah 3 s tahem 4, (tah 4 s tahem 3) atd. Obdržím řadu polygonálních tahů 1, 2, ..., 4 (obr. 3).

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.

Otakar Borůvka, *Příspěvek k řešení otázky ekonomické stavby elektrovodných 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](https://doi.org/10.1016/S0012-365X(00)00224-7)

Minimum spanning tree (MST)

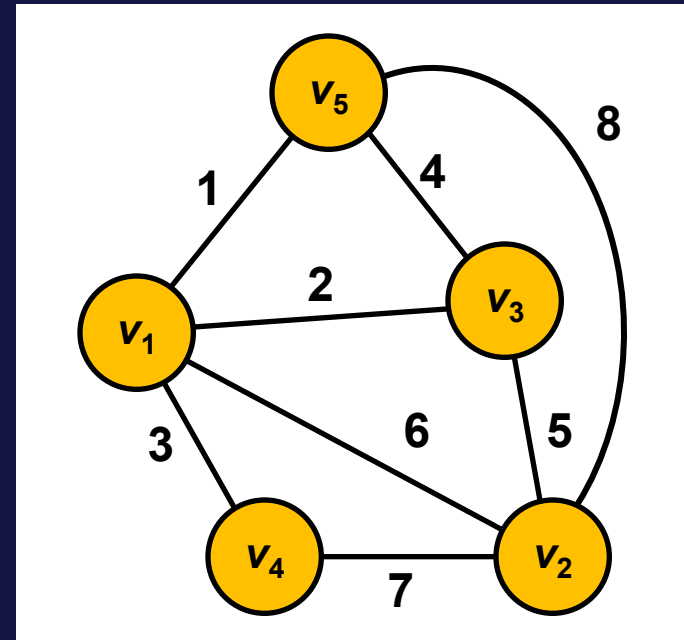


Problem

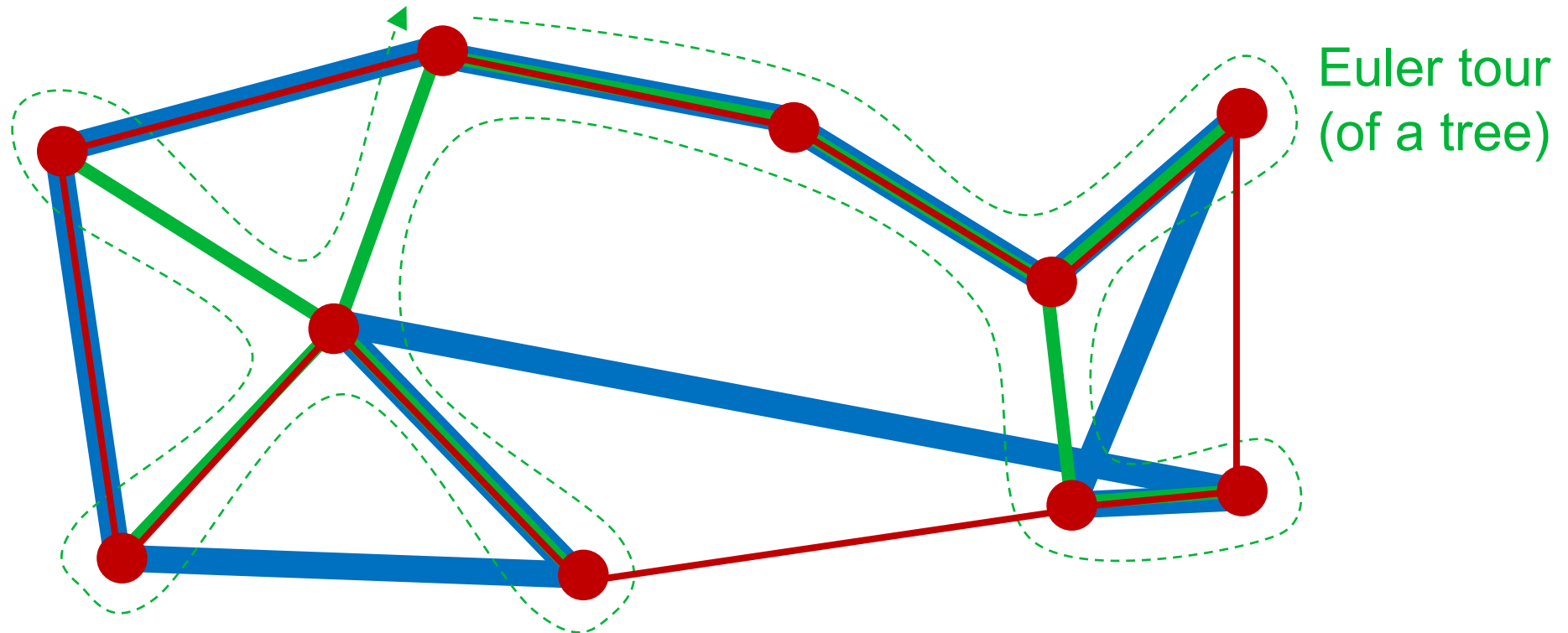
Find a **spanning tree** for a **connected undirected weighted** graph such that the **sum of the trees edge weights is smallest possible**

Weight of a minimum spanning tree?

- a) 10
- b) 11
- c) 12
- d) 13
- e) 14
- f) 15
- g) 16
- h) Don't know



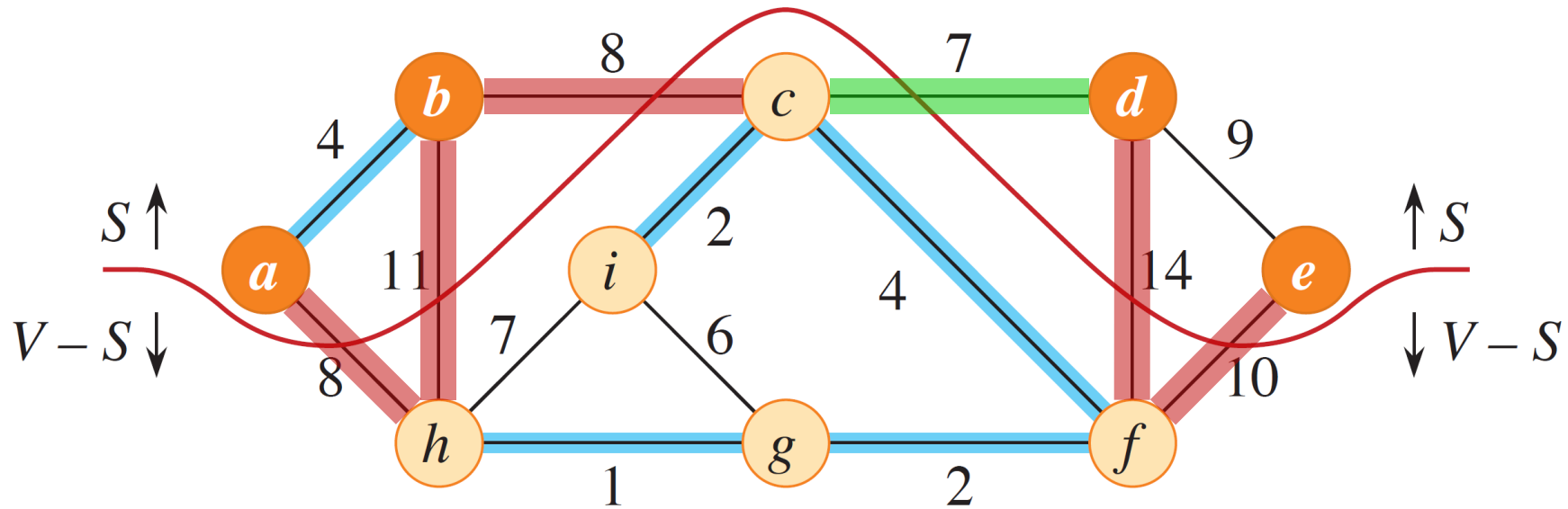
MST application – Euclidian shortest Hamiltonian cycle



Theorem : **Tour** found using an **MST** is a 2-approximation to the **shortest cycle**

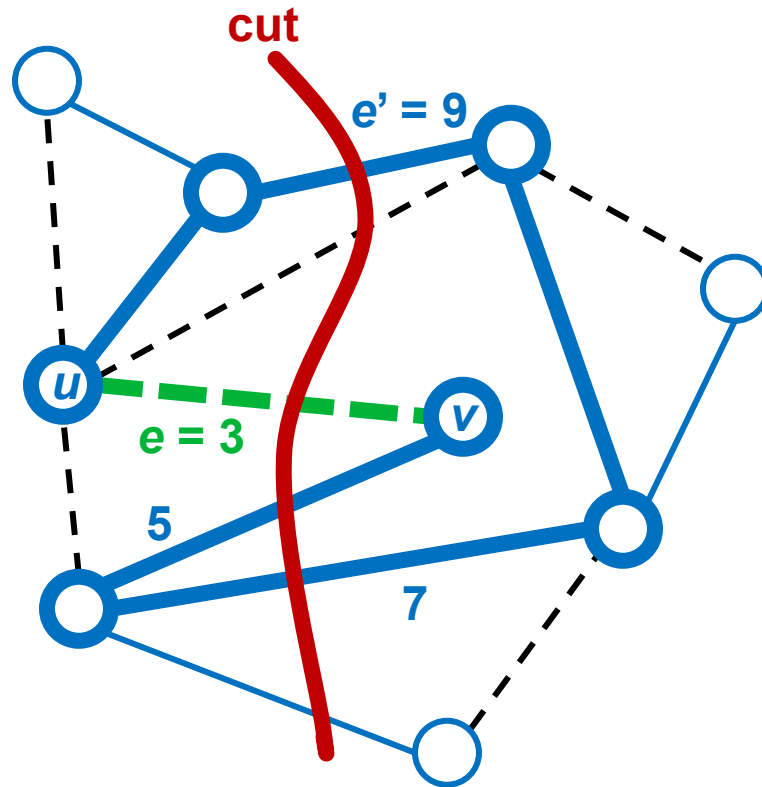
$$(|\text{Tour}| \leq 2 \cdot |\text{MST}| \quad \text{and} \quad |\text{MST}| \leq |\text{shortest cycle}|)$$

Minimum spanning tree – cut

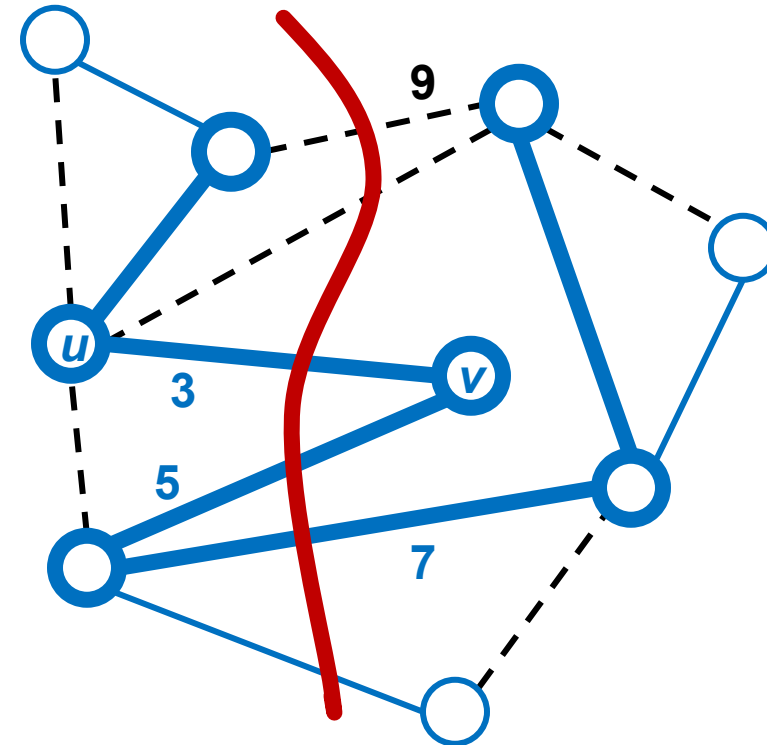
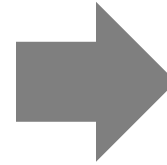


Theorem

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**



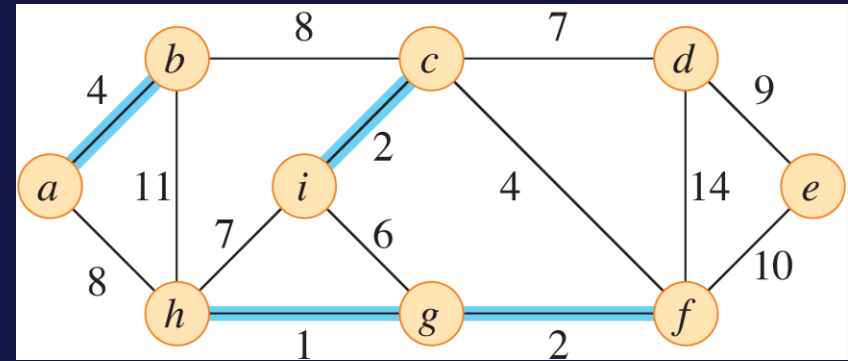
New spanning tree with smaller weight ⚡

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

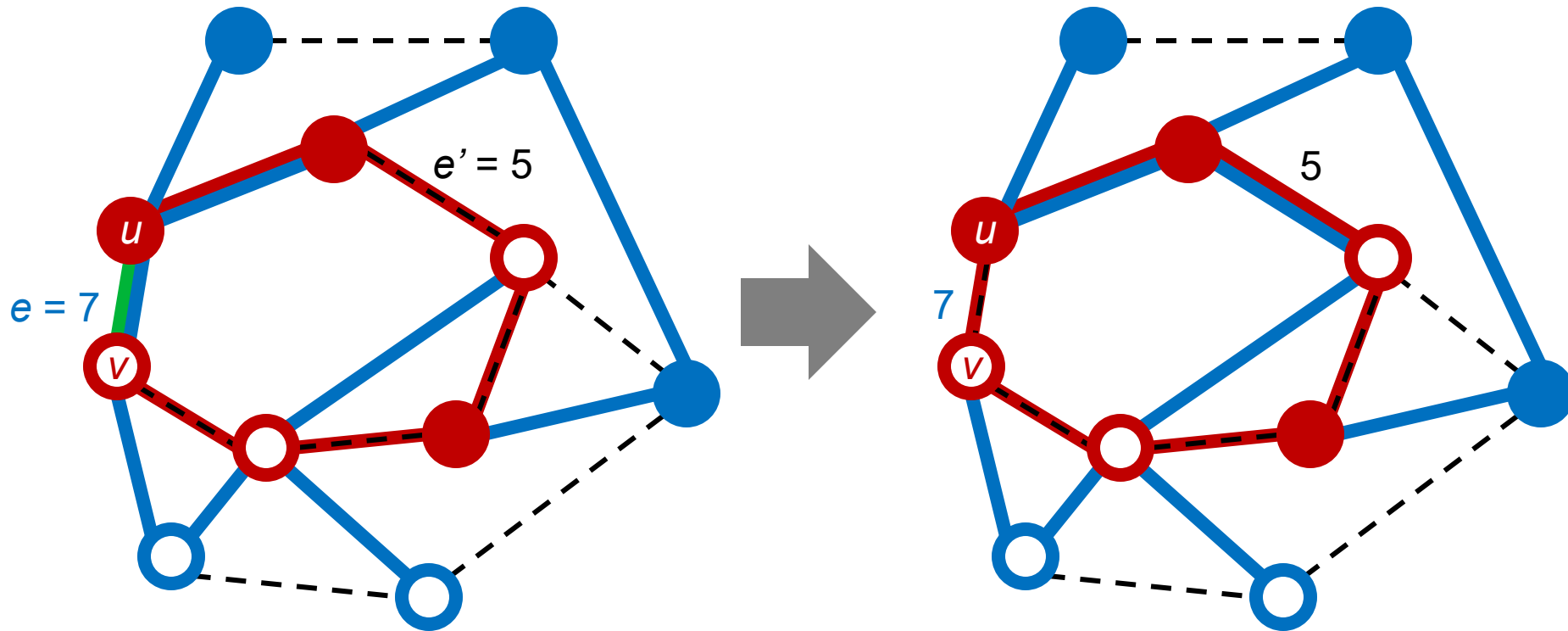


Theorem

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)

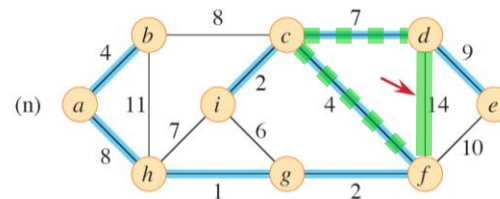
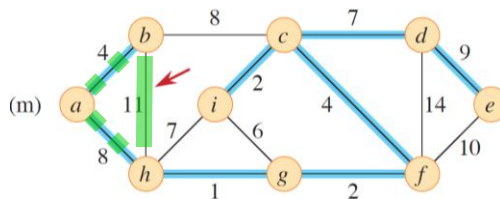
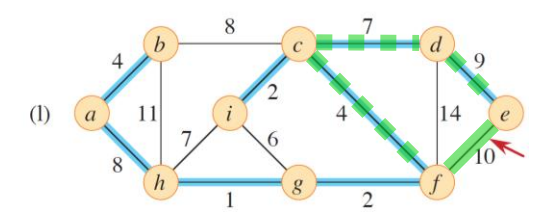
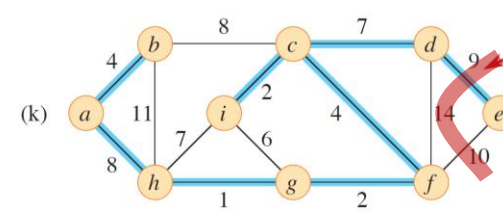
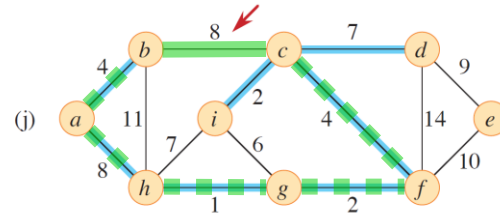
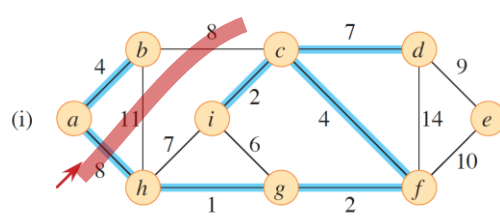
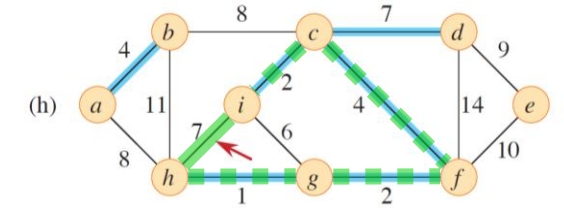
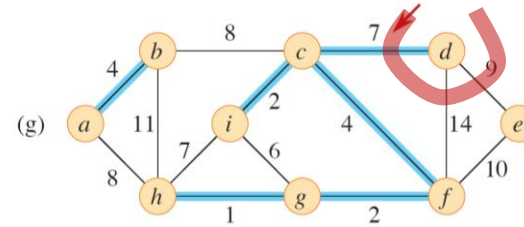
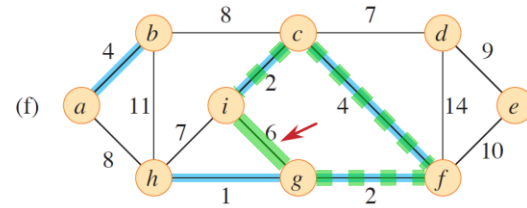
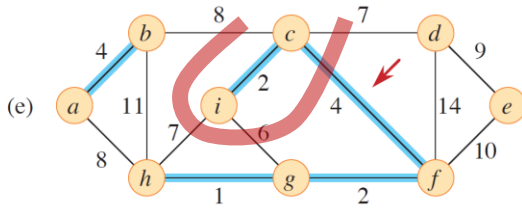
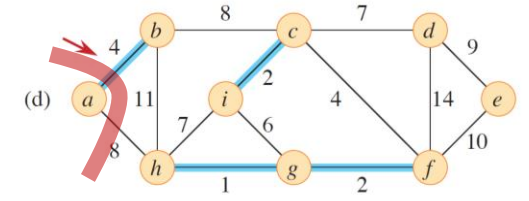
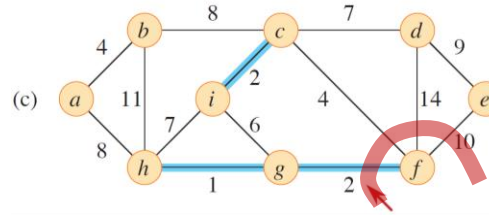
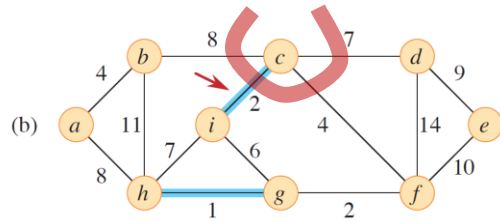
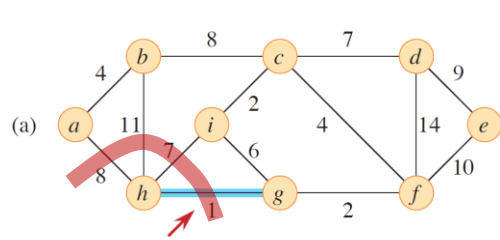
```
1   $A = \emptyset$ 
2  for each vertex  $v \in G.V$ 
3      MAKE-SET( $v$ )
4  create a single list of the edges in  $G.E$ 
5  sort the list of edges into monotonically increasing order by weight
6  for each edge  $(u, v)$  taken from the sorted list in order
7      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
8           $A = A \cup \{(u, v)\}$ 
9          UNION( $u, v$ )
10 return  $A$ 
```

bottleneck

union-find data
structure

Time $O(m \cdot \log n)$

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)

$$\text{MST-PRIM}(G, w, r)$$

```

1  for each vertex  $u \in G.V$ 

```

2 $u.key = \infty$

$$3 \quad \mathcal{U}.\pi = \text{NIL}$$
$$4 \quad r.key = 0$$
$$5 \quad Q = \emptyset$$

```

6  for each vertex  $u \in G.V$ 

```

7 $\text{INSERT}(Q, u)$ $\times n$

```
8  while  $Q \neq \emptyset$ 
```

9 $u = \text{EXTRACT-MIN}(Q)$ $\mathbf{x} \ n$

```

10   for each vertex  $v$  in  $G.Adj[u]$ 

```

```

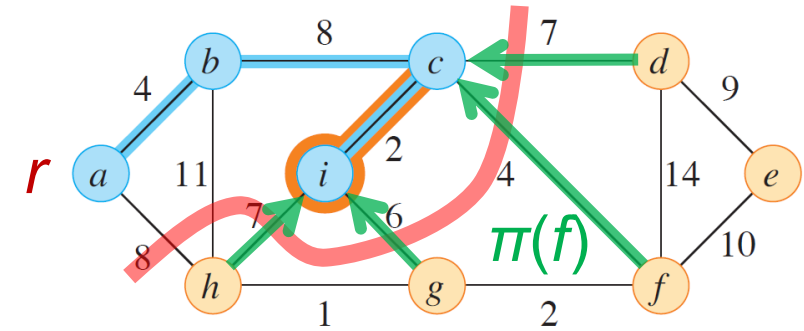
11      if  $v \in Q$  and  $w(u, v) < v.key$ 

```

$$12 \qquad v.\pi = u$$

13 $v.key = w(u, v)$

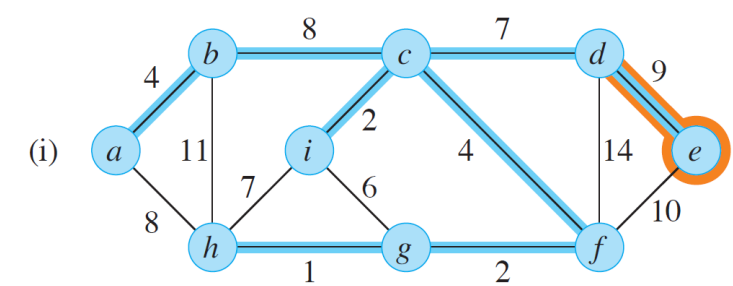
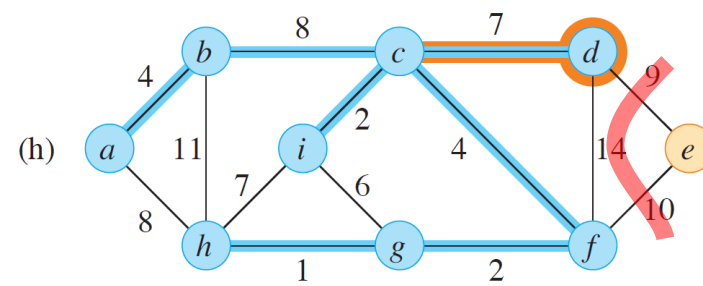
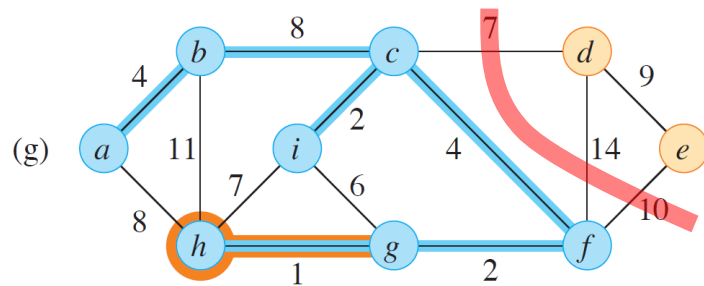
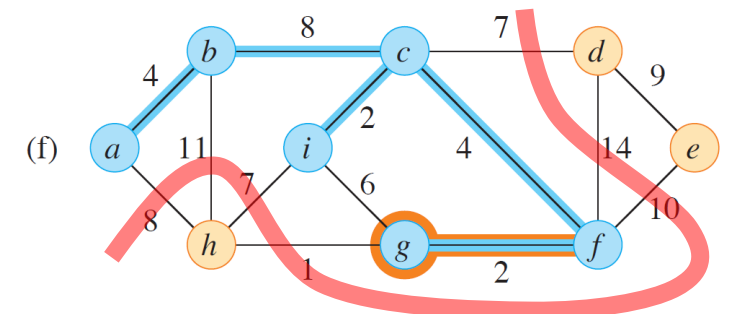
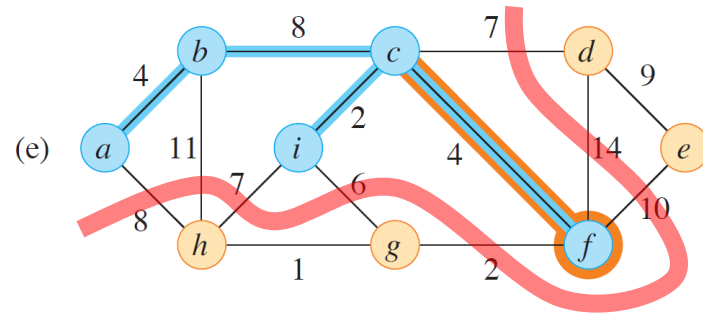
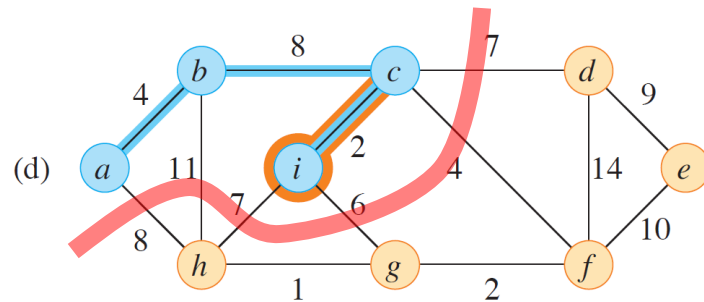
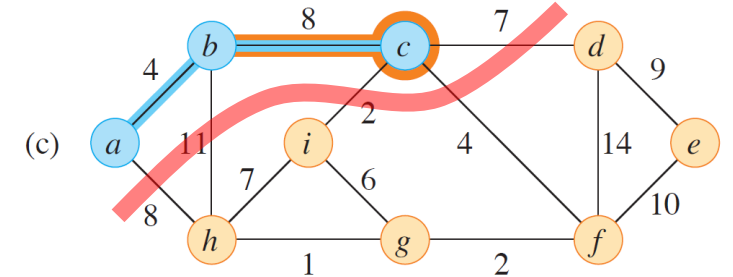
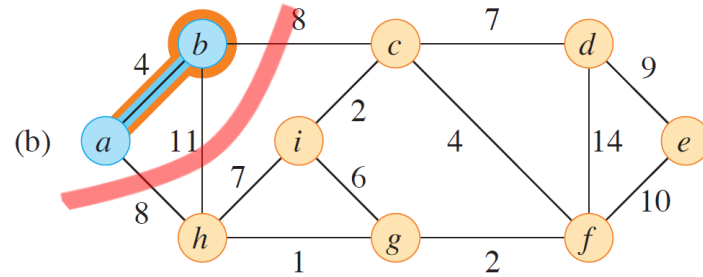
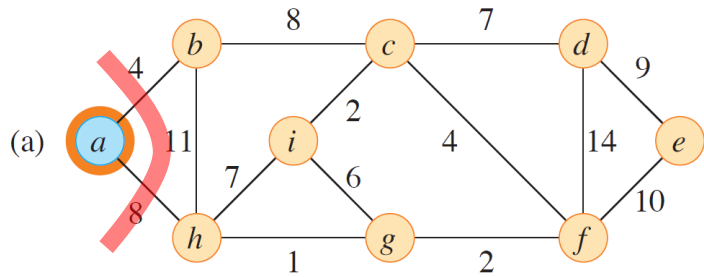
14 DECREASE-KEY($Q, v, w(u, v)$) $\times m$



bottleneck – priority queue

Time $O(m \cdot \log n)$

Prim's algorithm – example



Borůvka's algorithm

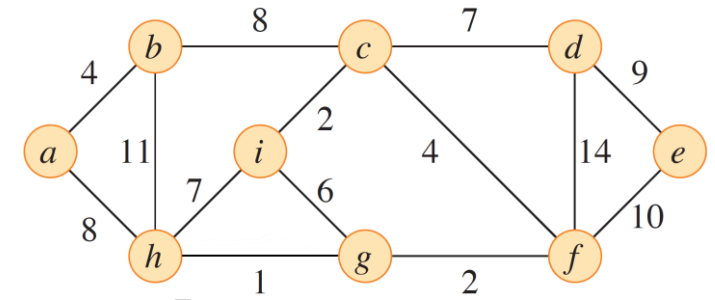
MST-BORUVKA(G, w)

```

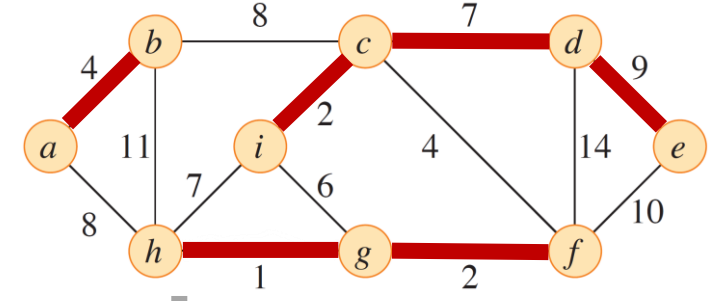
1   $A = \emptyset$ 
2  for each vertex  $v \in G.V$ 
3    MAKE-SET( $v$ )
3  while  $|A| < n - 1$ 
4     $B = \emptyset$ 
5    for each set  $S$ 
6      let  $(u, v)$  be lightest edge incident to  $S$ , i.e.  $u \in S$  and  $v \notin S$ 
7      if such  $(u, v)$  exists
8         $B = B \cup \{(u, v)\}$ 
8    for each edge  $(u, v) \in B$ 
9      if FIND-SET( $u$ )  $\neq$  FIND-SET( $v$ )
10        $A = A \cup \{(u, v)\}$ 
11       UNION( $u, v$ )
12 return  $A$ 
  
```

After i **while** iterations
each set size $\geq 2^i$ (or spanning)

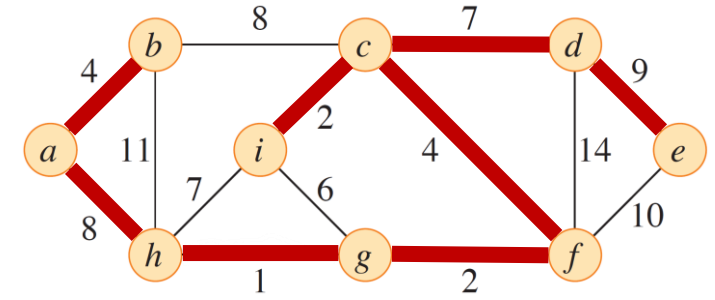
not curriculum



each node selects
lightest incident edge



each tree selects
lightest incident edge



Time $O(m \cdot \log n)$

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, 3 rd 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](https://doi.org/10.1090/S0002-9939-1956-0078686-7)

V. Jarník, *O jistém problému minimálním*, Práce Moravské Přírodovědecké Společnosti, 1930, hdl: [10338.dmlcz/500726](https://hdl.handle.net/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](https://doi.org/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](https://doi.org/10.1145/28869.28874)

Bernard Chazelle, *A Faster Deterministic Algorithm for Minimum Spanning Trees*, FOCS, 1997, doi [10.1109/SFCS.1997.646089](https://doi.org/10.1109/SFCS.1997.646089)

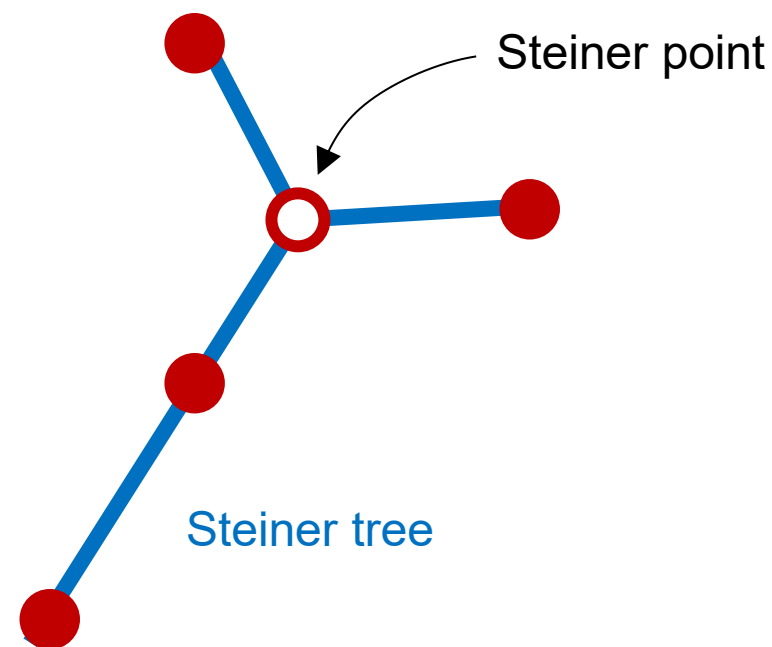
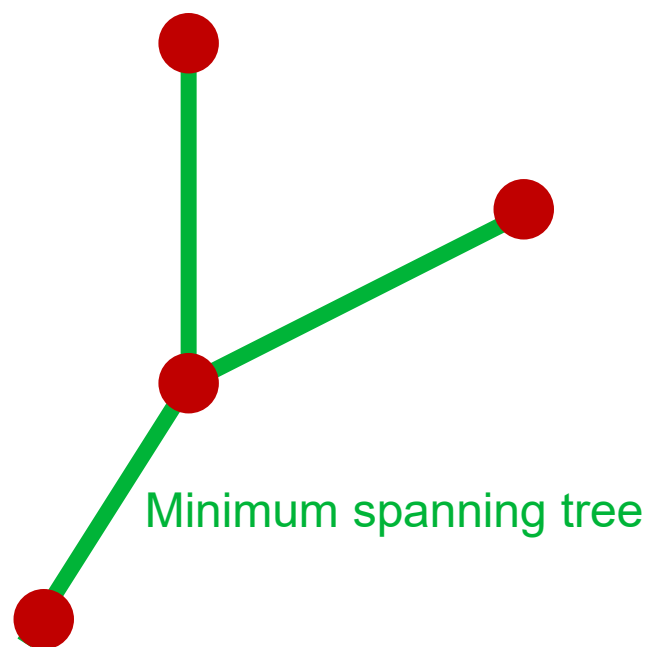
Seth Pettie, Vijaya Ramachandran, *An Optimal Minimum Spanning Tree Algorithm*, ICALP, 2000, doi: [10.1007/3-540-45022-X_6](https://doi.org/10.1007/3-540-45022-X_6)

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](https://doi.org/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](https://doi.org/10.1016/S0022-0000(05)80064-9)

Euclidian Steiner trees

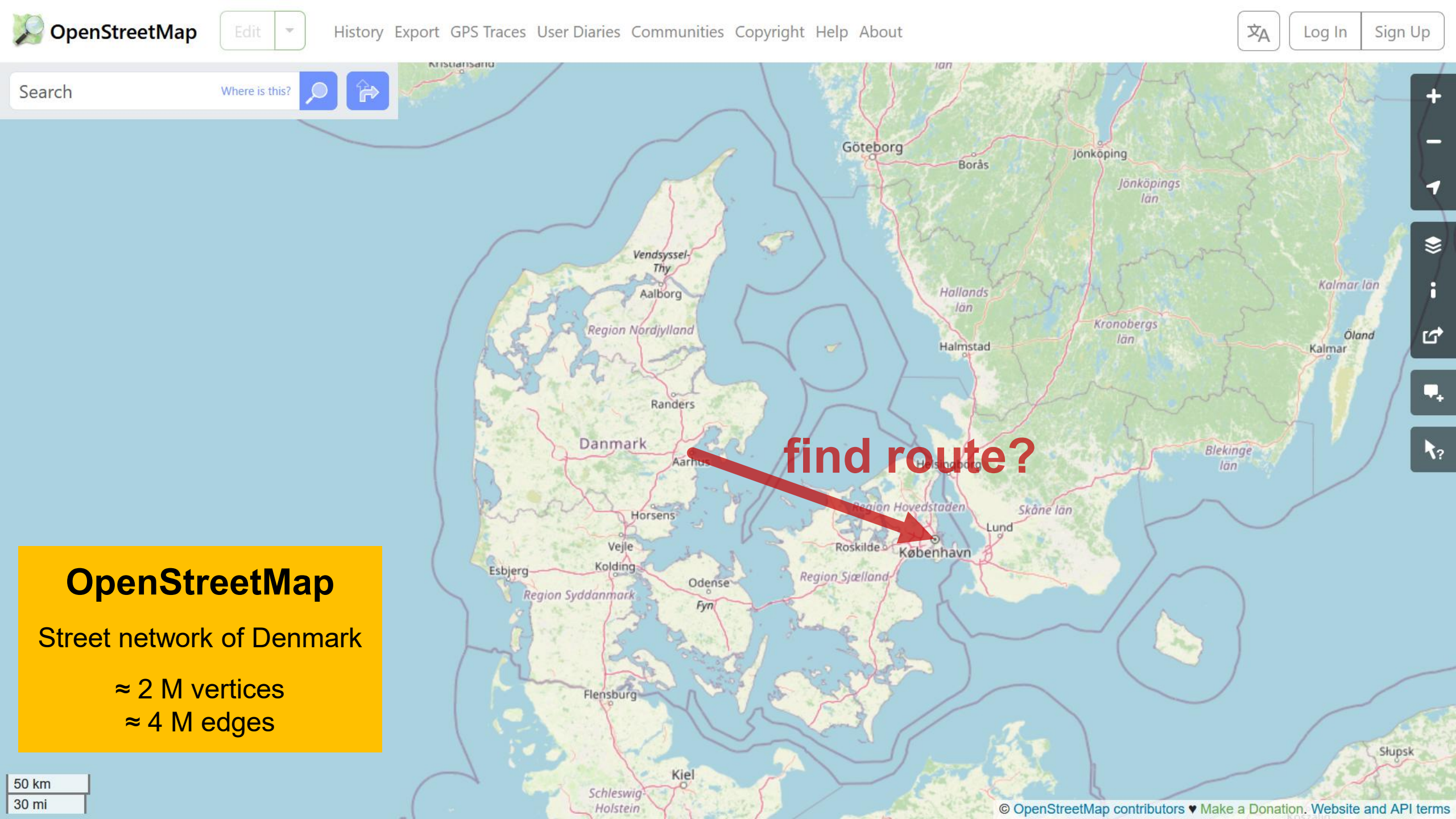
Given a set of points, find a tree with minimum weight connecting all points, possibly adding **new points**



Conjecture (Steiner Ratio): $|MST| \leq \frac{2}{\sqrt{3}} \cdot |\text{Steiner Tree}|$
NP hard; polynomial-time approximation scheme (PTAS)

Algorithms and Data Structures

Single source shortest paths (SSSP)
[CLRS, Chapter 22]

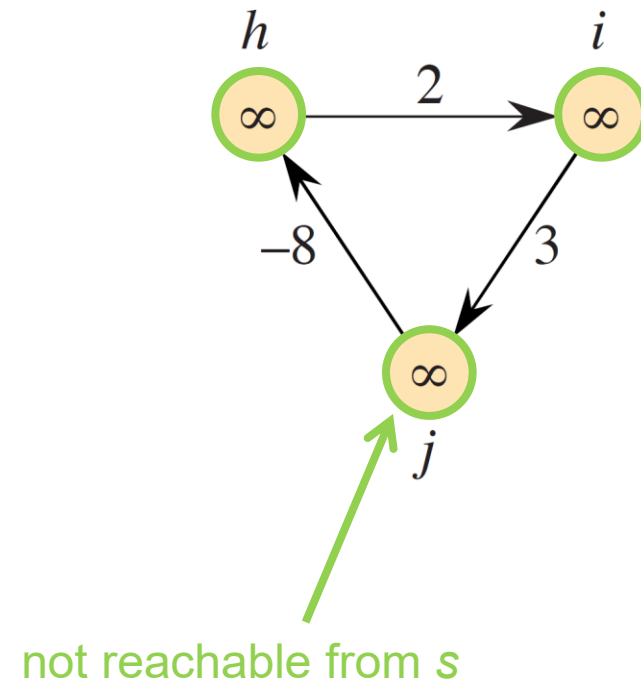
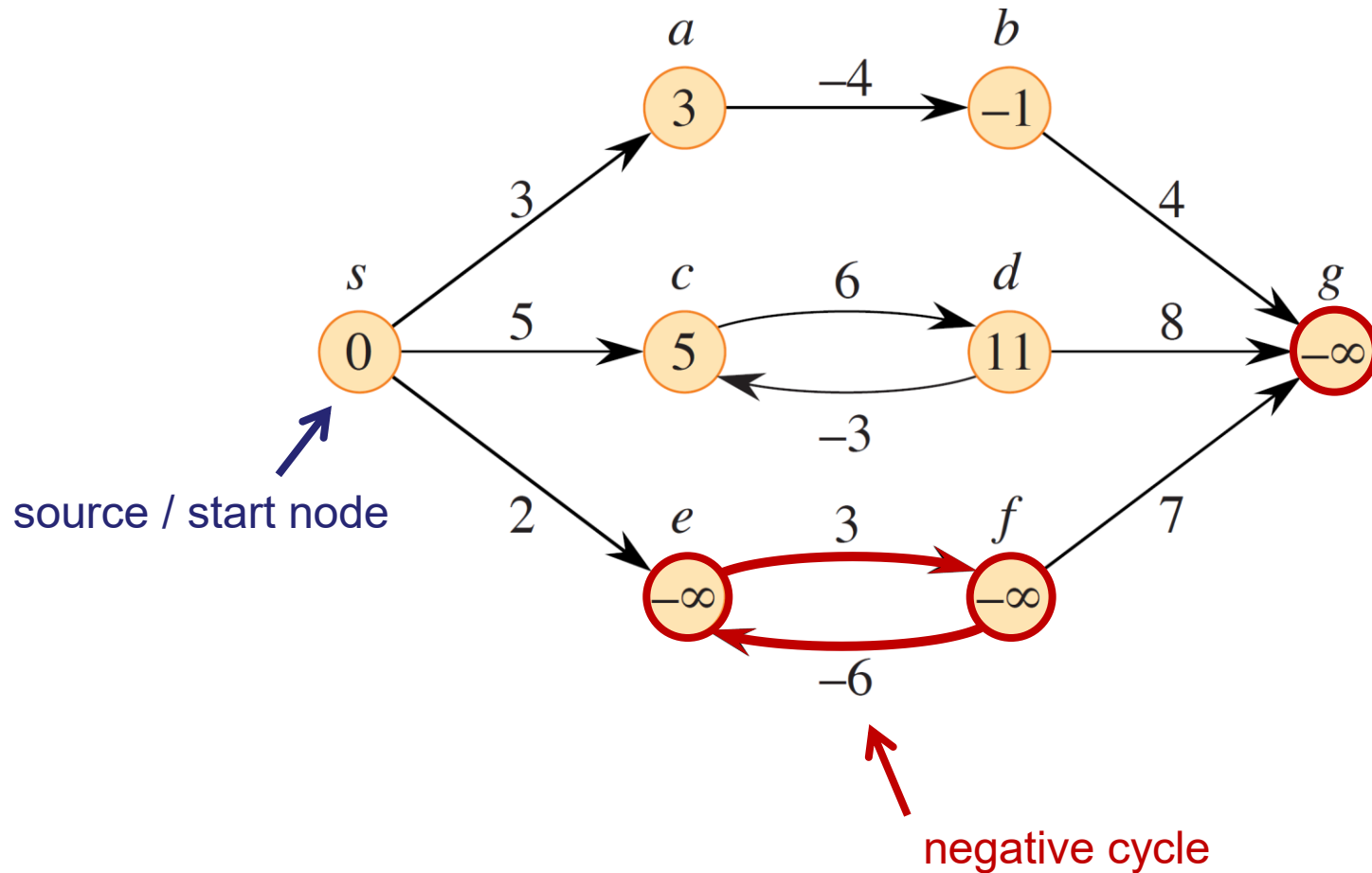


Search Where is this?

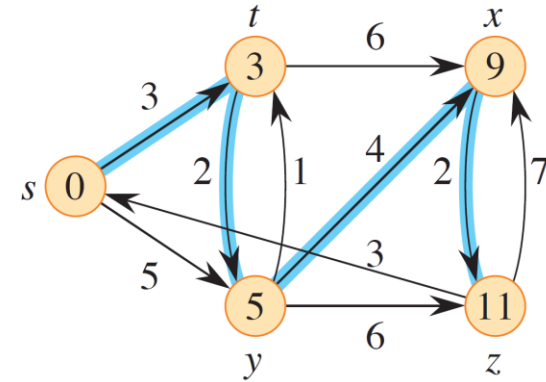
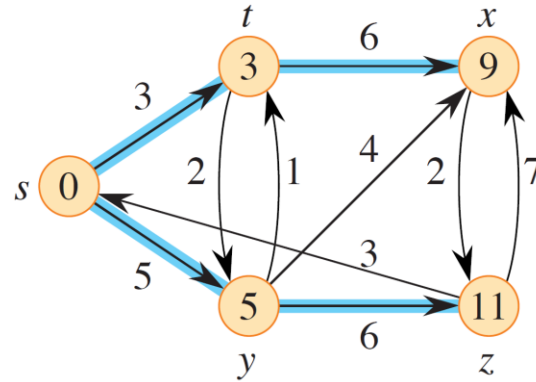
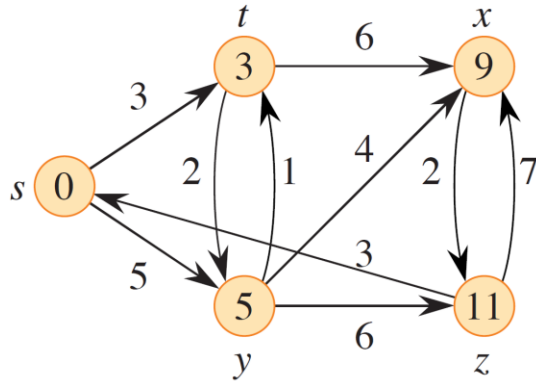
OpenStreetMap
Street network of Denmark
 ≈ 2 M vertices
 ≈ 4 M edges

50 km
30 mi

Shortest paths from s



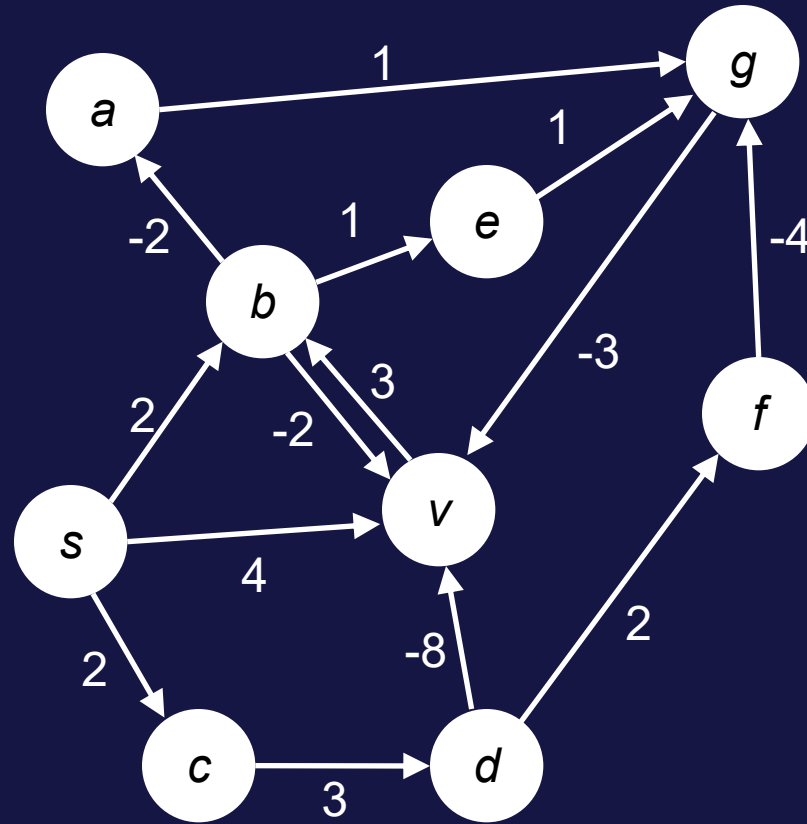
Shortest path trees



two different shortest path trees
representing paths from s of equal length

Length of a shortest path from s to v ?

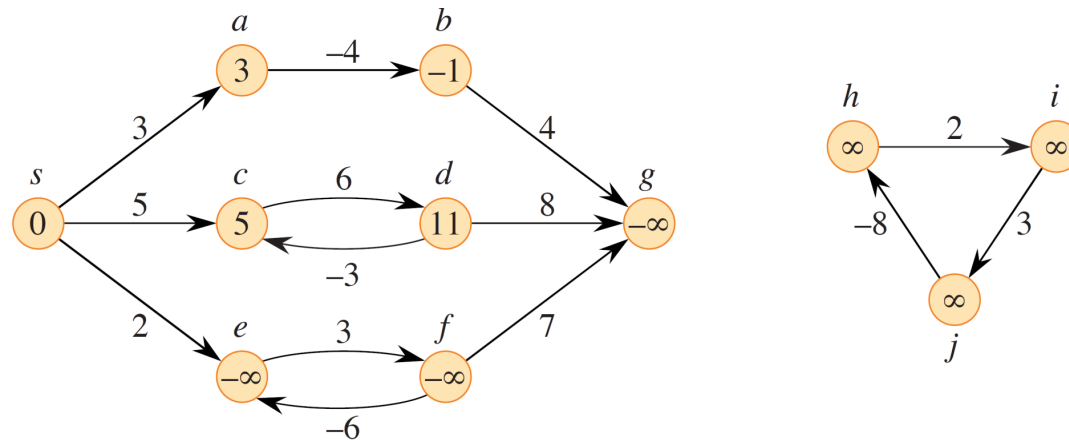
- a) $-\infty$
- b) -3
- c) -1
- d) 0
- e) 4
- f) $+\infty$
- g) Don't know



Shortest path estimates – initialization

INITIALIZE-SINGLE-SOURCE(G, s)

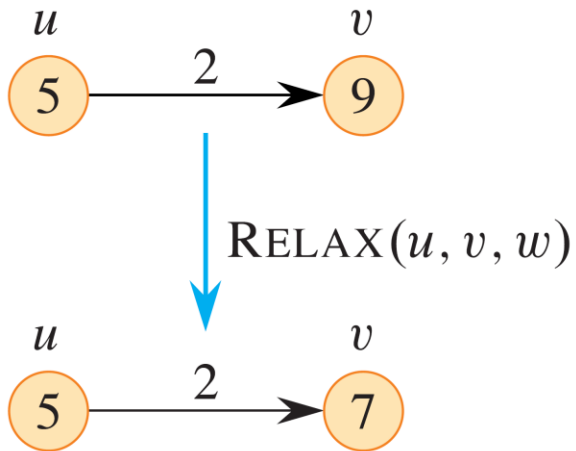
- 1 **for** each vertex $v \in G.V$
- 2 $v.d = \infty$
- 3 $v.\pi = \text{NIL}$
- 4 $s.d = 0$ ← parent in shortest path tree



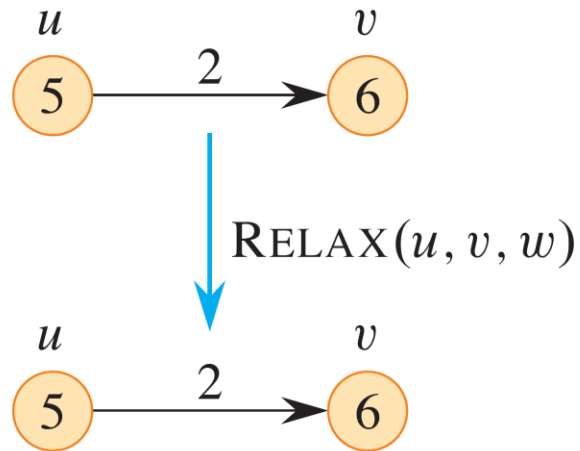
Relaxing edges

$\text{RELAX}(u, v, w)$

- 1 **if** $v.d > u.d + w(u, v)$
- 2 $v.d = u.d + w(u, v)$
- 3 $v.\pi = u$



Shorter distance found to v
(update $v.d$ and $v.\pi = u$)



No shorter distance
found to v

Invariant

If $v.d < \infty$, then a path exists from s to v with distance $v.d$ and $v.\pi$ as the second to last node

$v.d$ after $\text{Relax}(b, v, w)$?
(inside the nodes is the current d value)

a) -2

b) -1

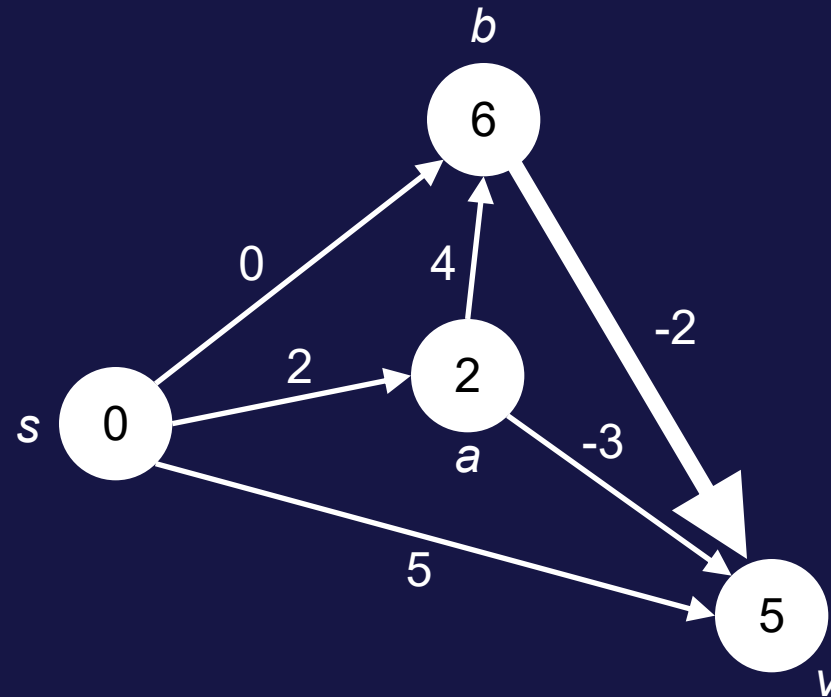
c) 0

d) 2

e) 4

f) 5

g) Don't know



Worst-case # calls to Relax ?

a) 10

b) 15

c) 31

d) 93

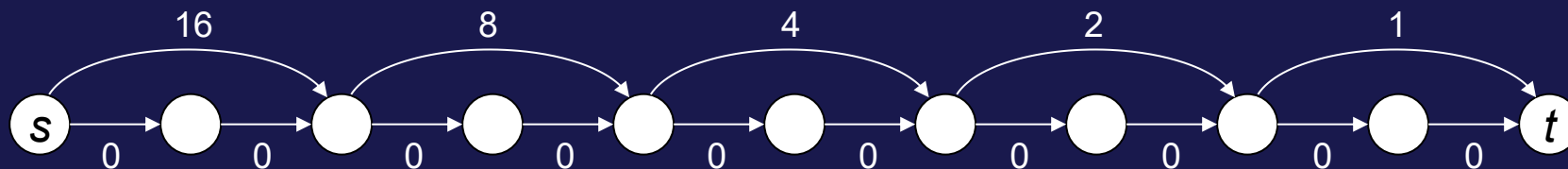
e) Don't know

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)



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 $(u, v) \in G.E$

4 RELAX(u, v, w)

5 **for** each edge $(u, v) \in G.E$

6 **if** $v.d > u.d + w(u, v)$

7 **return** FALSE

8 **return** TRUE

check for
negative cycle

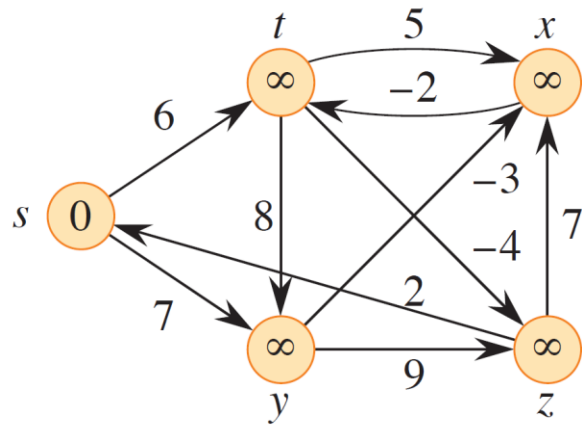
Time $O(n \cdot m)$

Alfonso Shimbel, *Structure in communication nets*, Proceedings of the Symposium on Information Networks, 1955

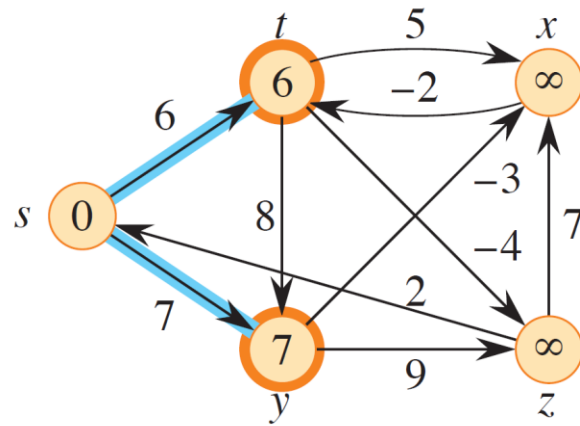
Richard Bellman, *On a routing problem*, Quarterly of Applied Mathematics, 1958, doi: [10.1090/qam/102435](https://doi.org/10.1090/qam/102435)

Lester R. Ford, Jr., *Network Flow Theory*, paper P-923, RAND Corporation, 1956

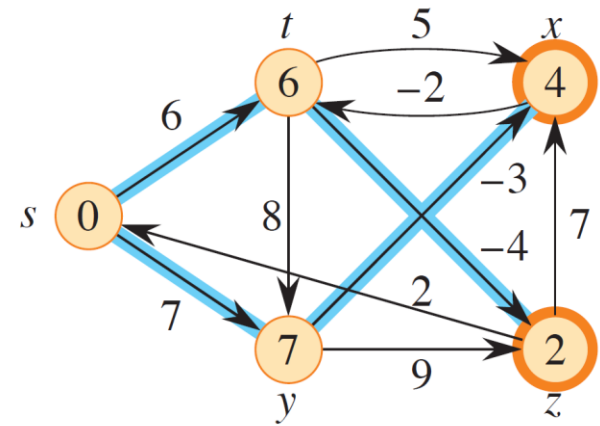
Bellman-Ford



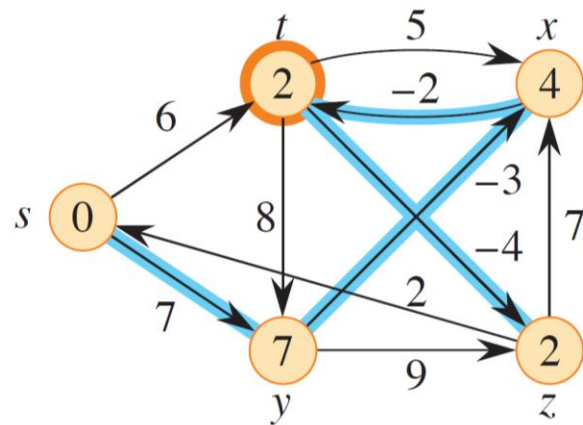
(a)



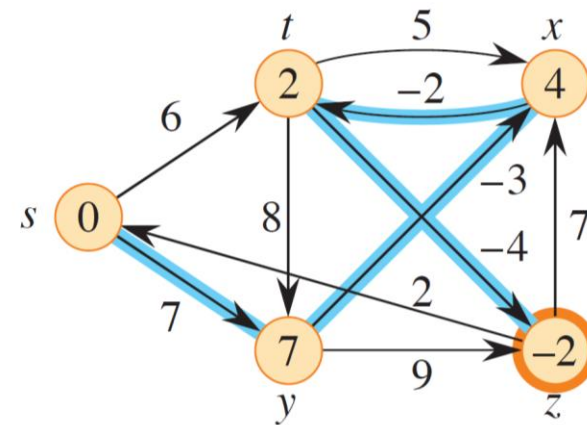
(b)



(c)



(d)



(e)

Assume there exists a **negative cycle** $v_1, v_2, \dots, v_k$ reachable from s

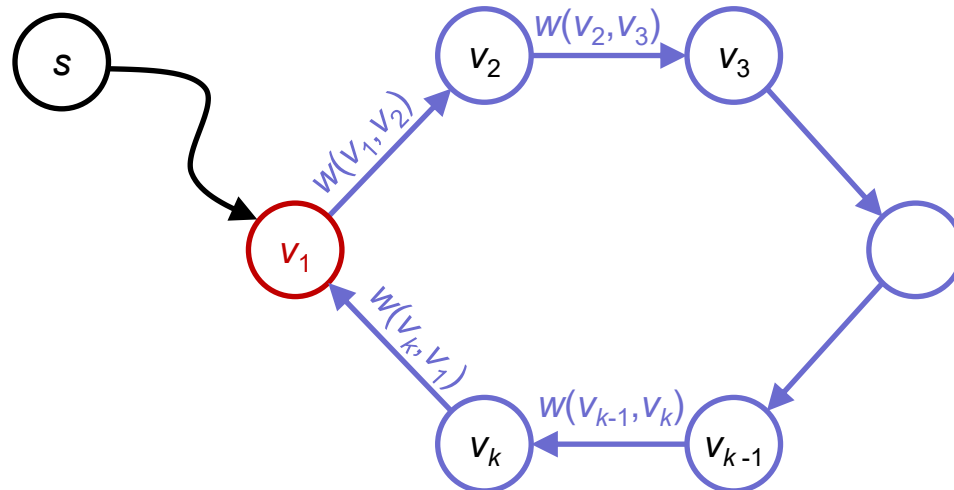
$$w(v_1, v_2) + w(v_2, v_3) + \dots + w(v_{k-1}, v_k) + w(v_k, v_1) < 0$$

and it is **not** possible to **RELAX** any edge on the cycle

$$d[v_{i+1}] \leq d[v_i] + w(v_i, v_{i+1})$$

This implies the **contradiction**

$$\begin{aligned} d[v_1] &\leq d[v_k] + w(v_k, v_1) \leq (d[v_{k-1}] + w(v_{k-1}, v_k)) + w(v_k, v_1) \leq \\ \dots &\leq d[v_1] + \underbrace{w(v_1, v_2) + w(v_2, v_3) + \dots + w(v_{k-1}, v_k) + w(v_k, v_1)}_{< 0} < d[v_1] \end{aligned}$$



Theorem

Consider an (unknown) shortest path tree T with an edge (u,v) .

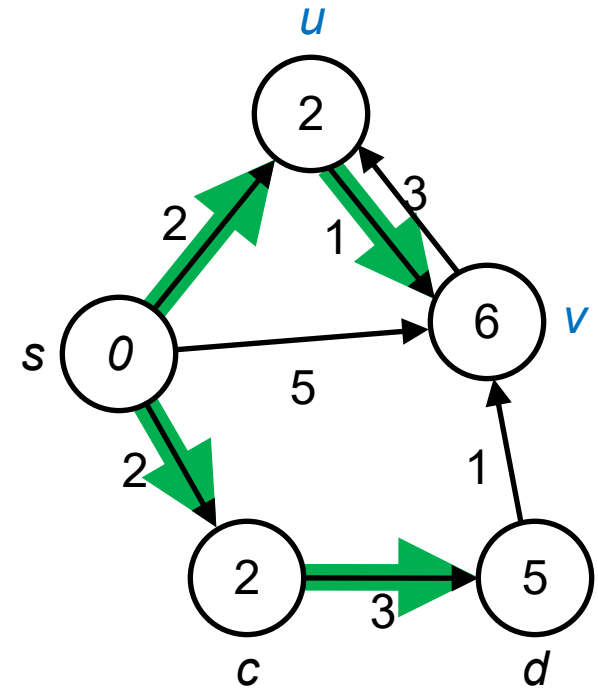
Assume $d[u]$ is the shortest distance to u .

After $\text{RELAX}(u,v,w)$ 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



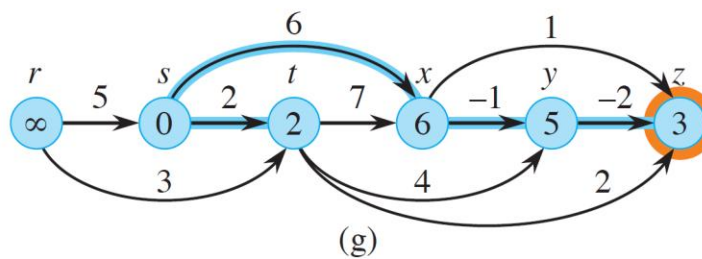
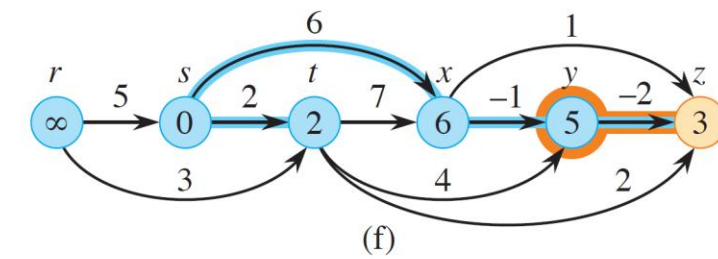
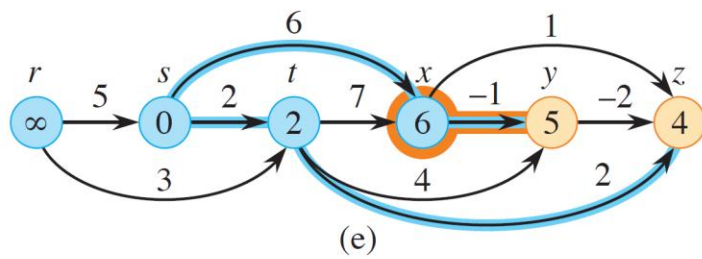
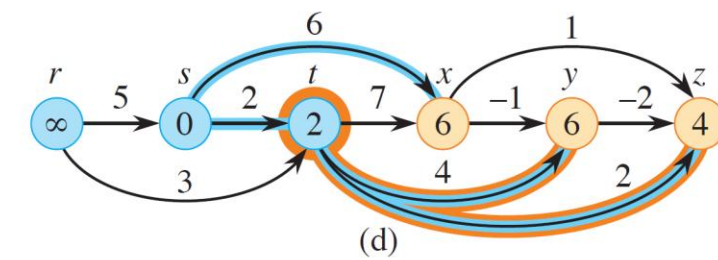
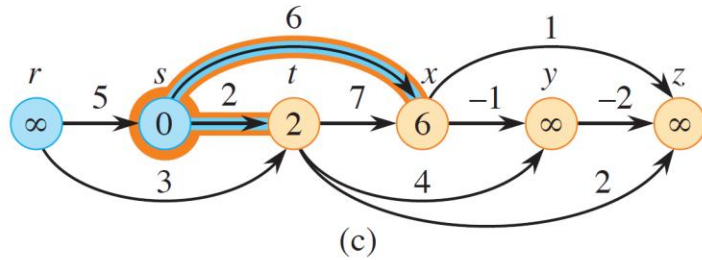
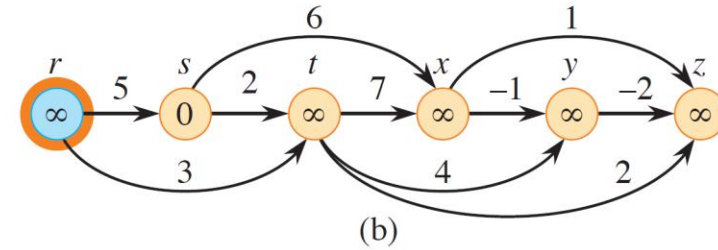
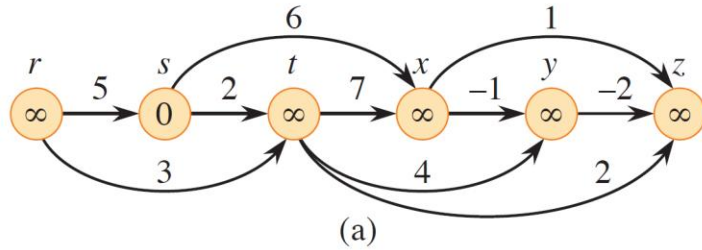
Shortest paths in acyclic graphs

DAG-SHORTEST-PATHS(G, w, s)

- 1 topologically sort the vertices of G
- 2 INITIALIZE-SINGLE-SOURCE(G, s)
- 3 **for** each vertex $u \in G.V$, 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



Dijkstra's algorithm (non-negative weights)

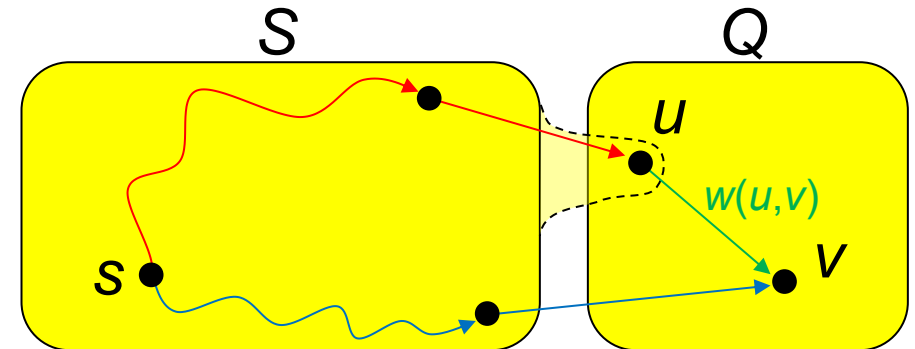
DIJKSTRA(G, w, s)

```
1  INITIALIZE-SINGLE-SOURCE( $G, s$ )
2   $S = \emptyset$ 
3   $Q = \emptyset$ 
4  for each vertex  $u \in G.V$ 
5      INSERT( $Q, u$ )
6  while  $Q \neq \emptyset$ 
7       $u = \text{EXTRACT-MIN}(Q)$ 
8       $S = S \cup \{u\}$ 
9      for each vertex  $v$  in  $G.\text{Adj}[u]$ 
10         RELAX( $u, v, w$ )
11         if the call of RELAX decreased  $v.d$ 
12             DECREASE-KEY( $Q, v, v.d$ )
```

- Visits nodes in increasing distance from s
- Q = priority queue with d as priority

Invariants

- $d[v]$ = shortest distance from s to v only visiting **intermediate nodes in S**
- $\forall p \in S, q \in Q : d[p] \leq d[q]$

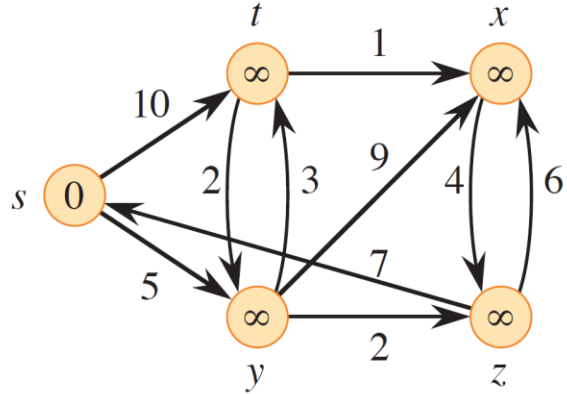


Update priority
of v in Q

Time $O((n + m) \cdot \log n)$
or $O(n^2 + m)$

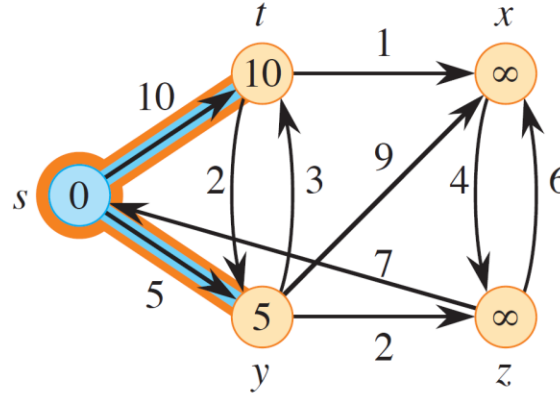
Dijkstra's algorithm

$Q = (s, 0) (t, \infty) (x, \infty) (y, \infty) (z, \infty)$



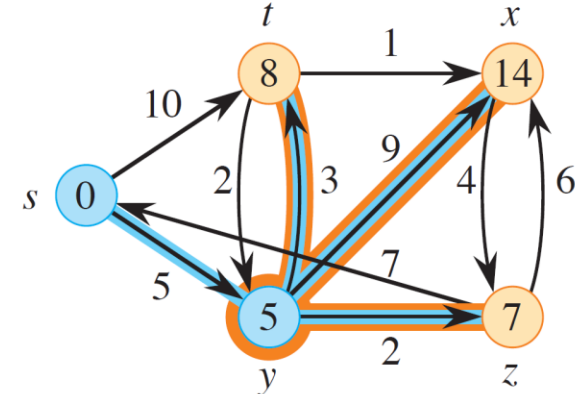
(a)

$Q = (y, 5) (t, 10) (x, \infty) (z, \infty)$



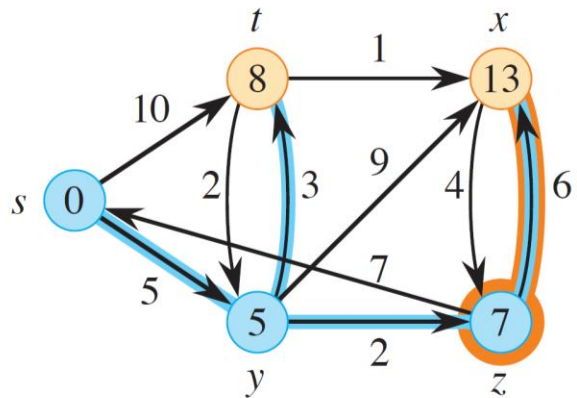
(b)

$Q = (z, 7) (t, 8) (x, 14)$



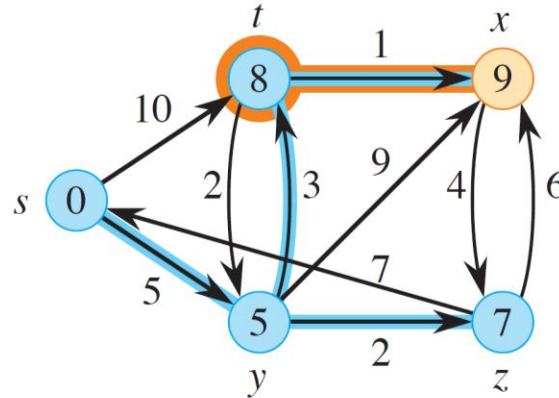
(c)

$Q = (t, 8) (x, 13)$



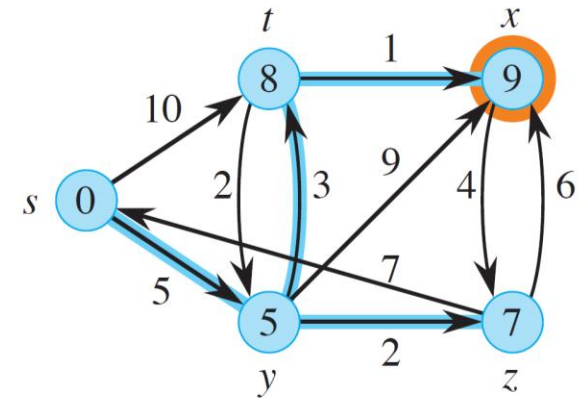
(d)

$Q = (x, 9)$



(e)

$Q = \emptyset$



(f)

Dijkstra's algorithm using repeated insertions

DIJKSTRA(G, w, s)

1 INITIALIZE-SINGLE-SOURCE(G, s)

2 $S = \emptyset$

(priority, node) pair $\rightarrow$ 3 $Q = \{(0, s)\}$

4 **while** $Q \neq \emptyset$

5 $(\delta, u) = \text{EXTRACT-MIN}(Q)$

skip outdated entries
where $\delta > u.d$ $\rightarrow$ 6 **if** $\delta = u.d$

7 $S = S \cup \{u\}$

8 **for** each vertex $v \in G.\text{Adj}[u]$

9 **if** $v.d > u.d + w(u, v)$

10 $v.d = u.d + w(u, v)$

11 $v.\pi = u$

the same node can be
inserted multiple times but
with decreasing priorities
(priority size increases
from $O(n)$ to $O(m)$)

$\rightarrow$ 12 $\text{INSERT}(Q, (v.d, v))$

} RELAX(u, v, w)

If priority queue
supports INSERT and
EXTRACT-MIN but not
DECREASE-KEY (like
Java's PriorityQueue)

Time $O((n + m) \cdot \log n)$

Single source shortest paths (SSSP)

Acyclic graphs (positive and negative weights)		$O(n + m)$
General graphs	only positive weights	Dijkstra $O((n + m) \cdot \log n)$ $O(n^2 + m)$ $O(m + n \cdot \log n)$
		Duan et al. $O(m \cdot \log^{2/3} n)$
	positive and negative weights	Bellman-Ford $O(m \cdot n)$

Relaxes every
edge exactly one

Edsger W. Dijkstra, *A note on two problems in connexion with graphs*, Numerische Mathematik, 1959, doi: [10.1007/BF01386390](https://doi.org/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](https://doi.org/10.1090/qam/102435)

Lester R. Ford, Jr., *Network Flow Theory*, paper P-923, RAND Corporation, 1956

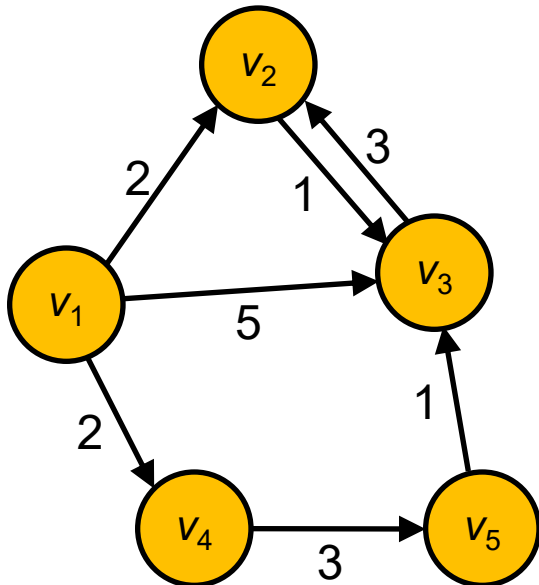
Duan, Mao, Mao, Shu, Yin, *Breaking the Sorting Barrier for Directed Single-Source Shortest Paths*, STOC, 2025, doi: [10.1145/3717823.3718179](https://doi.org/10.1145/3717823.3718179)

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}

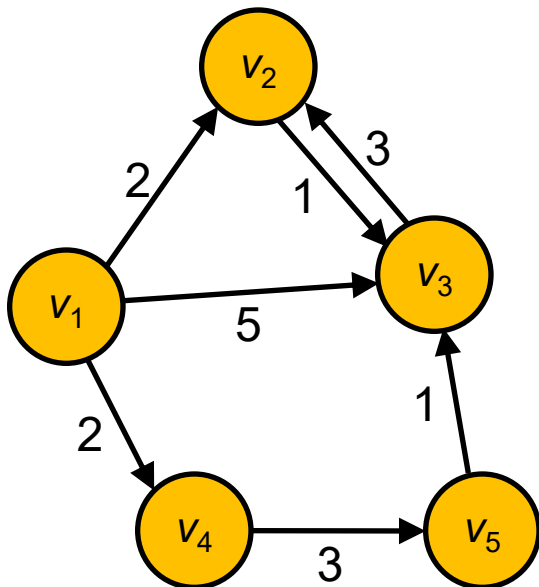
	1	2	3	4	5
1	0	2	3	2	5
2	$+\infty$	0	1	$+\infty$	$+\infty$
3	$+\infty$	3	0	$+\infty$	$+\infty$
4	$+\infty$	7	4	0	3
5	$+\infty$	4	1	$+\infty$	0

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	$+\infty$	0	1	$+\infty$	$+\infty$
3	$+\infty$	3	0	$+\infty$	$+\infty$
4	$+\infty$	7	4	0	3
5	$+\infty$	4	1	$+\infty$	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

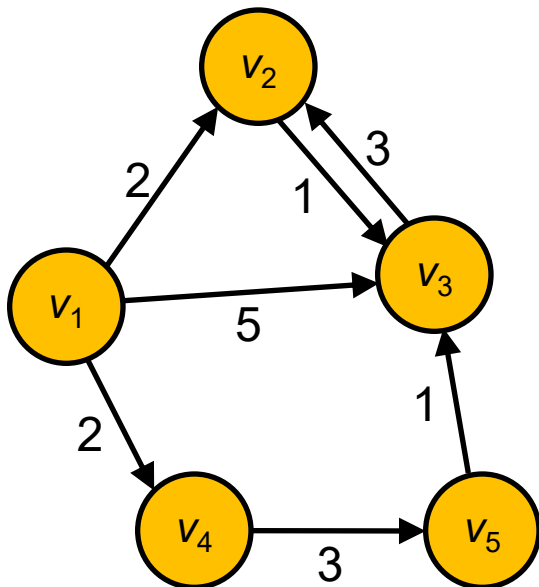
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)

```

1  if  $i == j$ 
2      print  $i$ 
3  elseif  $\pi_{ij} == \text{NIL}$ 
4      print “no path from”  $i$  “to”  $j$  “exists”
5  else PRINT-ALL-PAIRS-SHORTEST-PATH( $\Pi, i, \pi_{ij}$ )
6      print  $j$ 

```

Time $O(n)$



d_{ij}

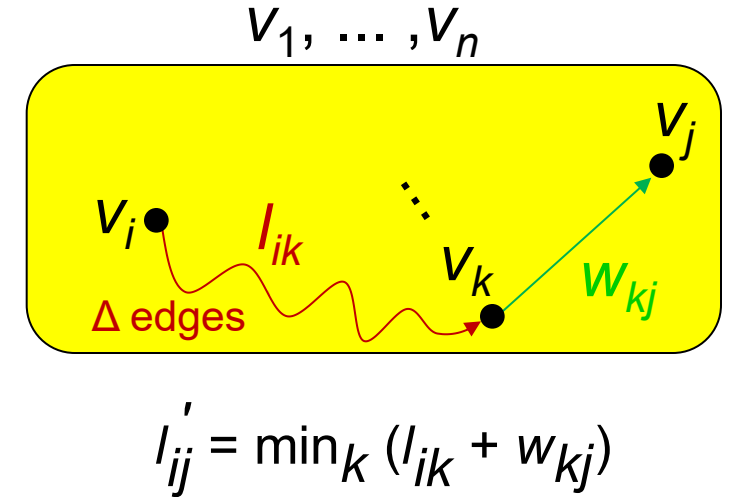
	1	2	3	4	5
1	0	2	3	2	5
2	$+\infty$	0	1	$+\infty$	$+\infty$
3	$+\infty$	3	0	$+\infty$	$+\infty$
4	$+\infty$	7	4	0	3
5	$+\infty$	4	1	$+\infty$	0

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  $\infty$ 
2  for  $i = 1$  to  $n$ 
3      for  $j = 1$  to  $n$ 
4          for  $k = 1$  to  $n$ 
5               $l_{ij}^{(r)} = \min \{ \underset{\text{red arrow}}{l_{ij}^{(r)}}, \underset{\text{green arrow}}{l_{ik}^{(r-1)} + w_{kj}} \}$ 
```



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 $\Delta+1$ 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               $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$ 
```

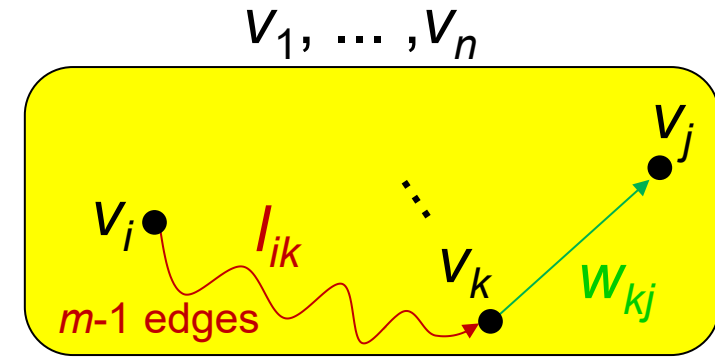
Time $O(n^3)$

SLOW-APSP($W, L^{(0)}, n$)

```

1  let  $L = (l_{ij})$  and  $M = (m_{ij})$  be new  $n \times n$  matrices
2   $L = L^{(0)}$  ← diagonal 0, elsewhere  $\infty$ 
3  for  $r = 1$  to  $n - 1$ 
4       $M = \infty$  // initialize  $M$ 
5      // Compute the matrix “product”  $M = L \cdot W$ 
6      EXTEND-SHORTEST-PATHS( $L, W, M, n$ )
7       $L = M$  → →
8  return  $L$  ↖ Copy content of matrix

```



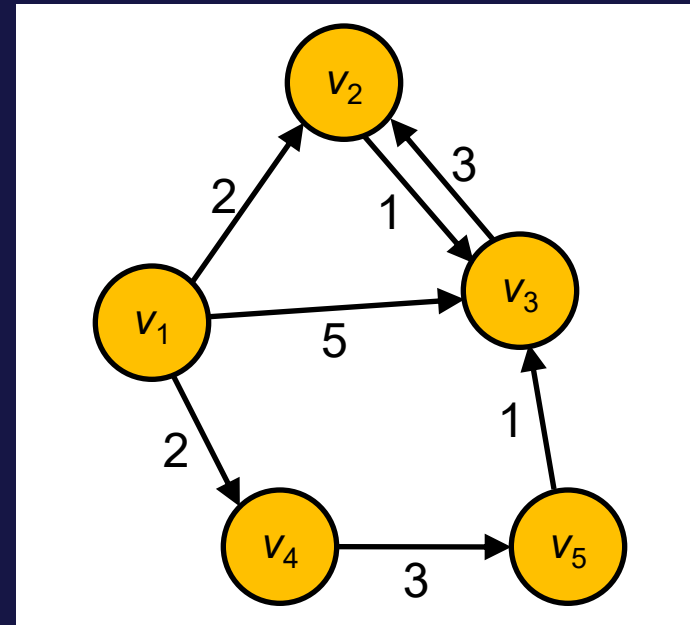
W = weight matrix

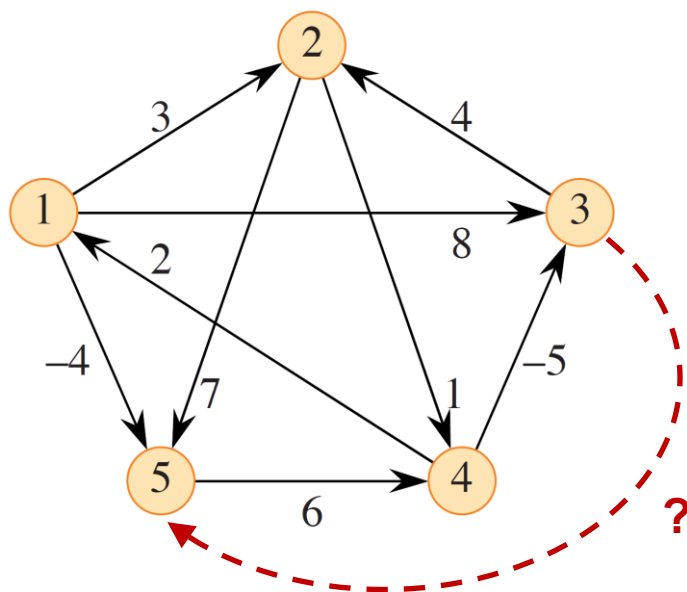
$L_{ij}^{(m)}$ = shortest distance from i to j for paths with m edges

Time $O(n^4)$

$$L_{1,3}^{(3)} \text{ ?}$$

- a) 0
- b) 3
- c) 5
- d) 6
- e) 9
- f) $+\infty$
- g) Don't know





input graph

$$L^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad L^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 2 & -4 \\ 3 & 0 & -4 & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & \infty & 1 & 6 & 0 \end{pmatrix}$$

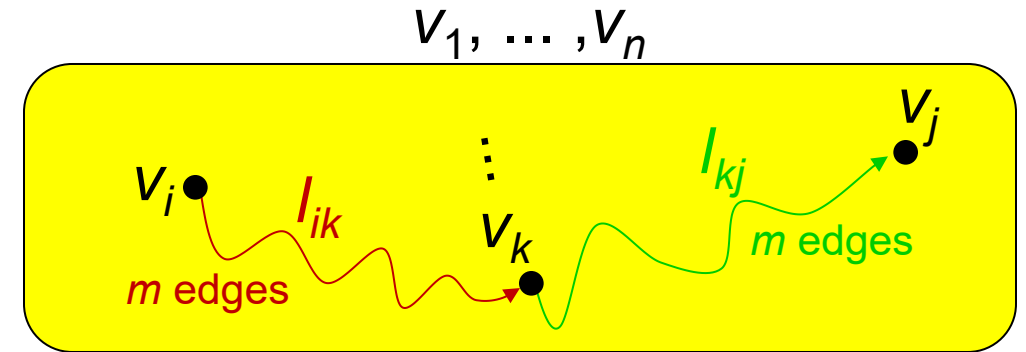
$$L^{(3)} = \begin{pmatrix} 0 & 3 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad L^{(4)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix}$$

FASTER-APSP(W, n)

```

1  let  $L$  and  $M$  be new  $n \times n$  matrices
2   $L = W$   $\leftarrow$  assumes diagonal 0
3   $r = 1$ 
4  while  $r < n - 1$   $\leftarrow$  Invariant:  $L = L^{(r)}$ 
5       $M = \infty$ 
6      EXTEND-SHORTEST-PATHS( $L, L, M, n$ )
7       $r = 2r$ 
8       $L = M$   $\leftarrow$  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

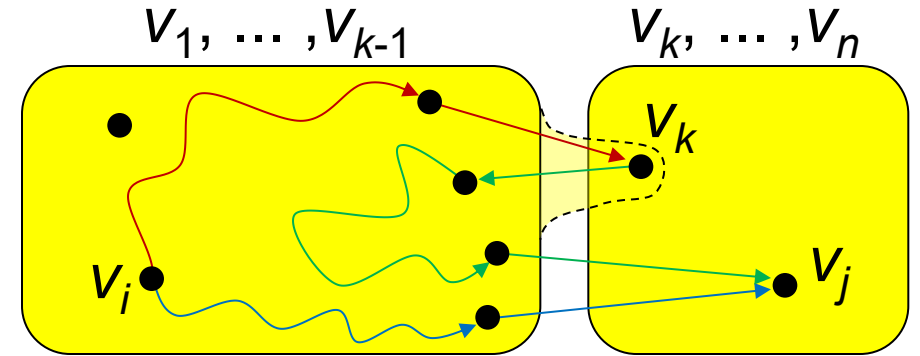
FLOYD-WARSHALL(W, n)

```

1   $D^{(0)} = W$ 
2  for  $k = 1$  to  $n$ 
3      let  $D^{(k)} = (d_{ij}^{(k)})$  be a new  $n \times n$  matrix
4      for  $i = 1$  to  $n$ 
5          for  $j = 1$  to  $n$ 
6               $d_{ij}^{(k)} = \min \{ \underbrace{d_{ij}^{(k-1)}}_{\text{blue arrow}}, \underbrace{d_{ik}^{(k-1)}}_{\text{red arrow}} + \underbrace{d_{kj}^{(k-1)}}_{\text{green arrow}} \}$ 
7  return  $D^{(n)}$ 

```

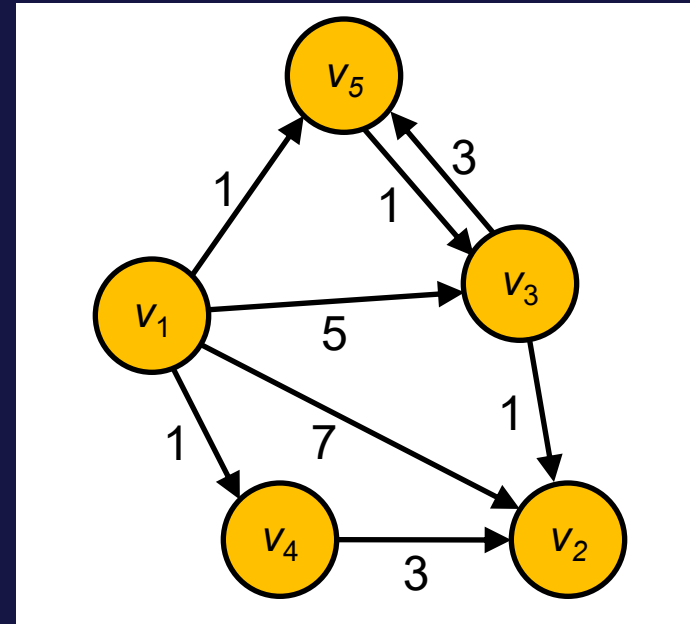
Time $O(n^3)$



$d_{ij}^{(k)}$ = shortest distance from i to j only using **intermediate nodes 1..k**

$$d_{1,2}^{(3)} \text{ ?}$$

- a) 3
- b) 4
- c) 6
- d) 7
- e) $+\infty$
- f) Don't know



Transitive closure

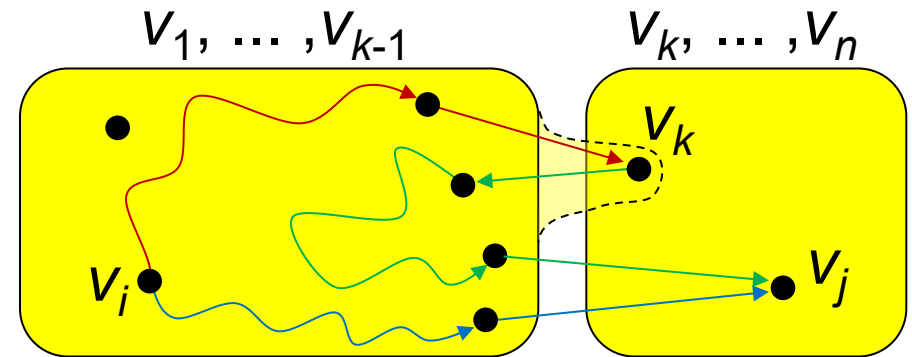
(Floyd-Warshall's algorithm simplified)

TRANSITIVE-CLOSURE(G, n)

```

1  let  $T^{(0)} = (t_{ij}^{(0)})$  be a new  $n \times n$  matrix
2  for  $i = 1$  to  $n$ 
3      for  $j = 1$  to  $n$ 
4          if  $i == j$  or  $(i, j) \in G.E$ 
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  $n \times n$  matrix
9      for  $i = 1$  to  $n$ 
10         for  $j = 1$  to  $n$ 
11              $t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)})$ 
12  return  $T^{(n)}$ 

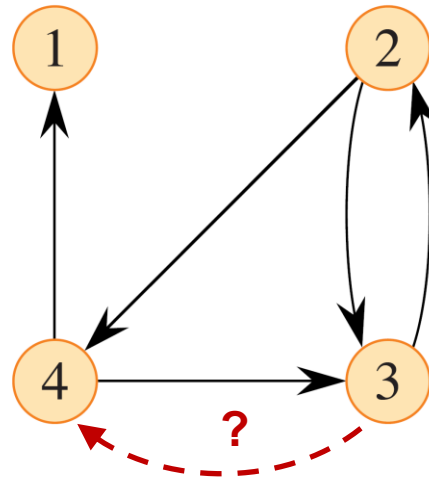
```



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

$$T^{(0)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(1)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(2)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 0 & 1 & 1 \end{pmatrix}$$

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$$T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Johnson's APSP algorithm

- **Sparse** graphs, i.e., $m \ll n^2$, with positive and **negative** weights

Without negative weights $\Rightarrow$ n x Dijkstra : $O(n \cdot (n \cdot \log n + m)) \ll O(n^3)$

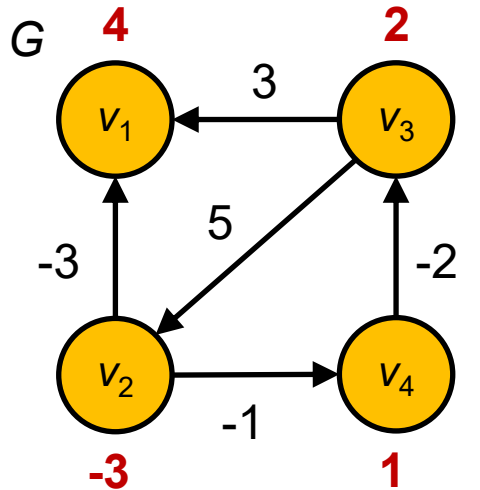
With negative weights $\Rightarrow$ Floyd-Warshall : $O(n^3)$

Johnson's algorithm

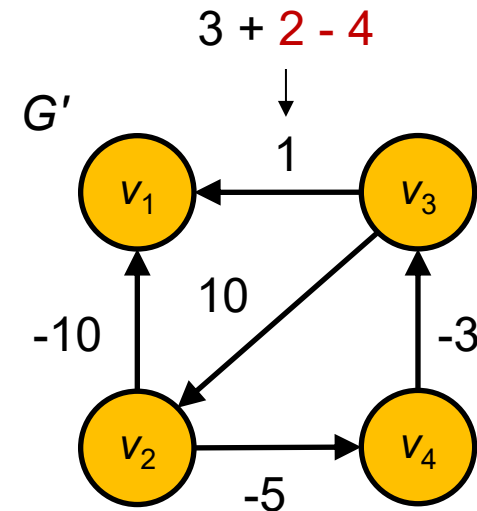
Idea Run Dijkstra's algorithm on a graph G' without negative edge weights, but having the same shortest paths as in G

Height transformation

associate every node u
an arbitrary **height** $h(u)$,
a.k.a. heuristic



$$w'(u,v) = w(u,v) + h(u) - h(v)$$



$d(v_2, v_3) =$
 $d'(v_2, v_3) + h(v_3) - h(v_2)$
 $= -8 + 2 - (-3)$

d	v_1	v_2	v_3	v_4
v_1	0	$+\infty$	$+\infty$	$+\infty$
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

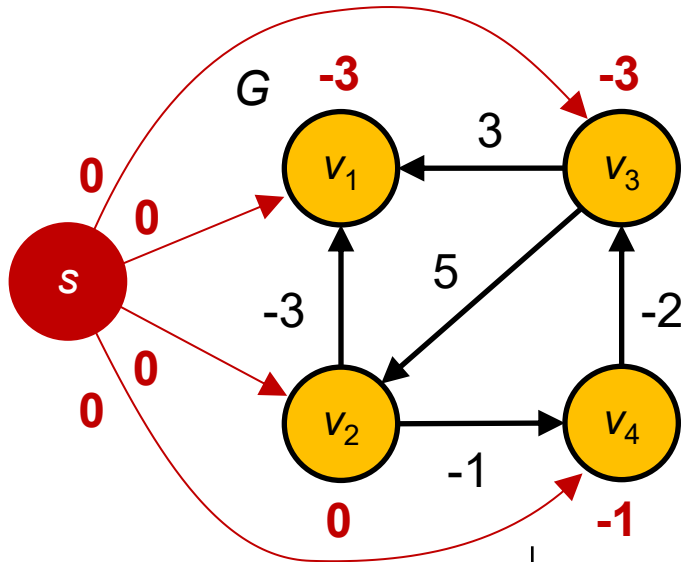
$$l'(v_{i_1} v_{i_2} \dots v_{i_k}) = l(v_{i_1} v_{i_2} \dots v_{i_k}) + h(v_{i_1}) - h(v_{i_k})$$

Shortest path in G
 $\Leftrightarrow$ shortest path in G'



d'	v_1	v_2	v_3	v_4
v_1	0	$+\infty$	$+\infty$	$+\infty$
v_2	-10	0	-8	-5
v_3	0	10	0	5
v_4	-3	7	-3	0

Johnson's algorithm

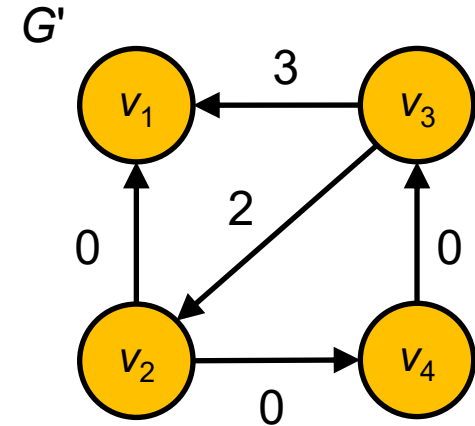


$$w'(u, v) = w(u, v) + h(u) - h(v)$$



$$w'(u, v) = w(u, v) + h(u) - h(v) \geq 0$$

since $h(v) \leq h(u) + w(u, v)$



d	v_1	v_2	v_3	v_4
v_1	0	$+\infty$	$+\infty$	$+\infty$
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	$+\infty$	$+\infty$	$+\infty$
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  $G'.V = G.V \cup \{s\}$ ,  
    $G'.E = G.E \cup \{(s, v) : v \in G.V\}$ , and  
    $w(s, v) = 0$  for all  $v \in G.V$   
2  if BELLMAN-FORD( $G', w, s$ ) == FALSE  
3      print “the input graph contains a negative-weight cycle”  
4  else for each vertex  $v \in G'.V$   
5      set  $h(v)$  to the value of  $\delta(s, v)$   
        computed by the Bellman-Ford algorithm  
6  for each edge  $(u, v) \in G'.E$   
7       $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$   
8  let  $D = (d_{uv})$  be a new  $n \times n$  matrix  
9  for each vertex  $u \in G.V$   
10     run DIJKSTRA( $G, \hat{w}, u$ ) to compute  $\hat{\delta}(u, v)$  for all  $v \in G.V$   
11     for each vertex  $v \in G.V$   
12          $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$   
13  return  $D$ 
```

Time $O(n^2 \cdot \log n + n \cdot m)$

Shortest paths in graphs

Single source
shortest paths
(SSSP)

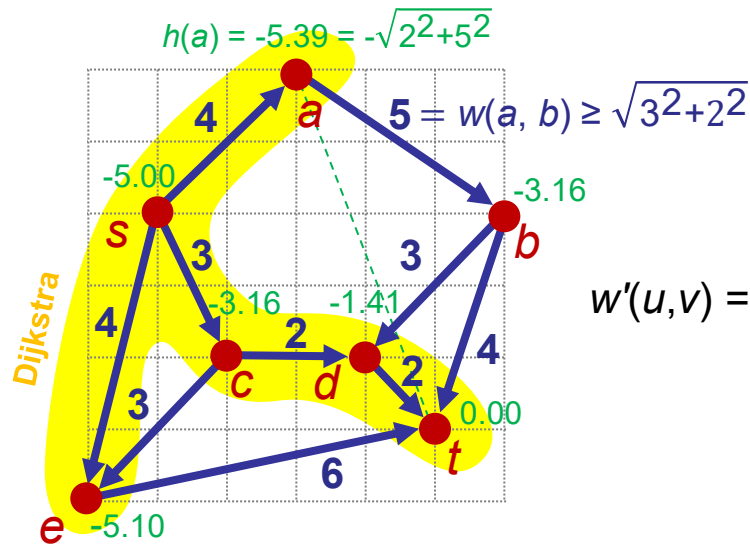
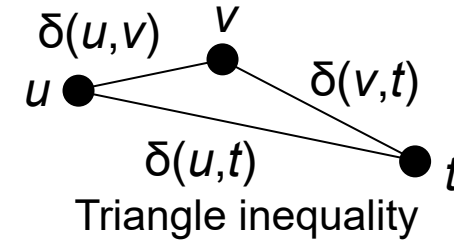
All pairs
shortest paths
(APSP)

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)	$n \times$ Dijkstra $O(n^2 \cdot \log n + n \cdot m)$ (2) $O(n^3)$ (3)
		$O(n \cdot \log n + m)$ (2)	
		$O(n^2)$ (3)	
	Positive and negative weights	Duan et al. $O(m \cdot \log^{2/3} n)$	Floyd-Warshall $O(n^3)$ Johnson $O(n^2 \cdot \log n + n \cdot m)$ (2)
		Bellman-Ford $O(n \cdot m)$	

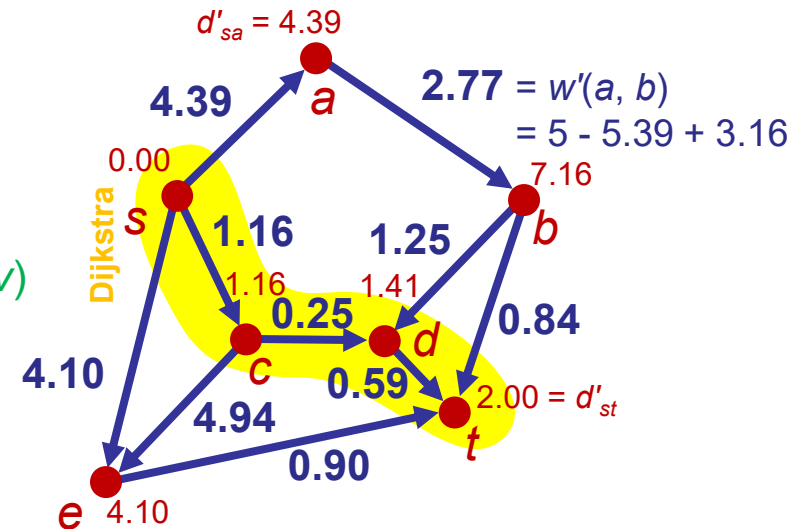
Different priority queues: (1) binary heap, (2) Fibonacci heap, (3) array

A*-algorithm – shortest path $s \rightsquigarrow t$

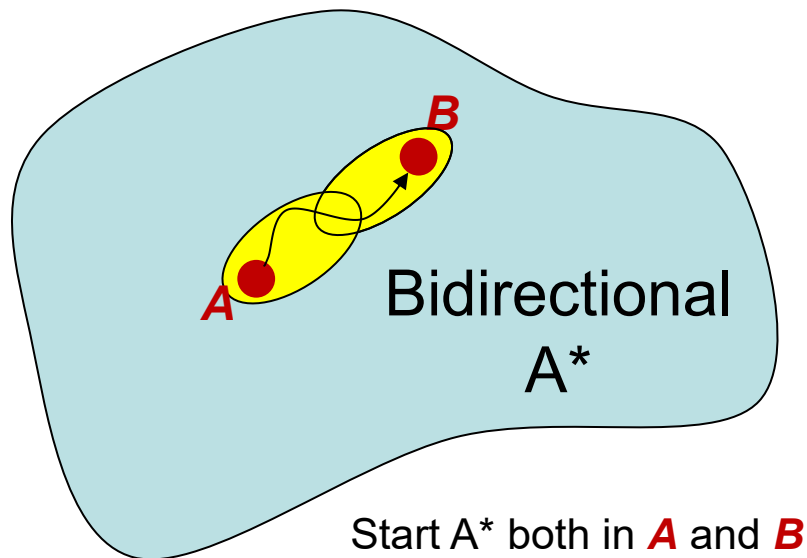
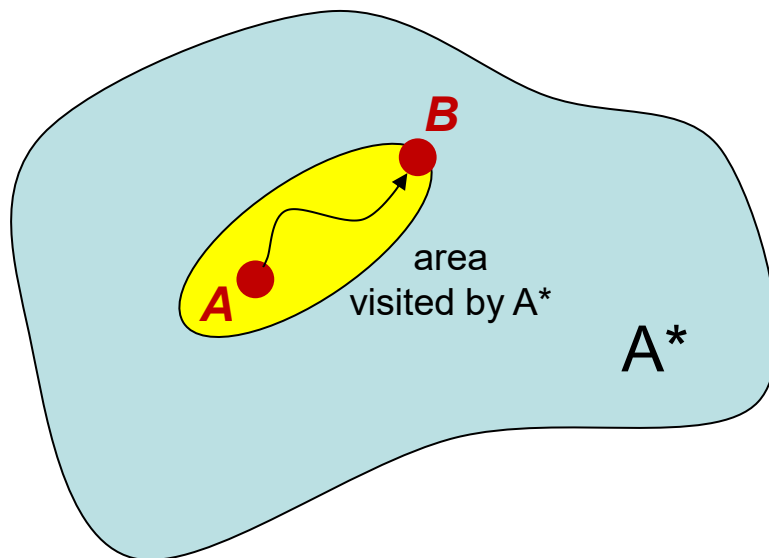
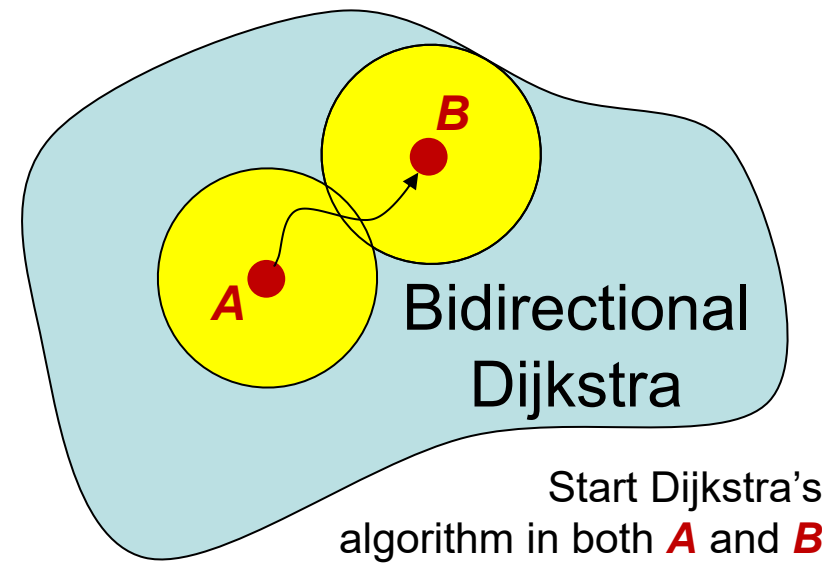
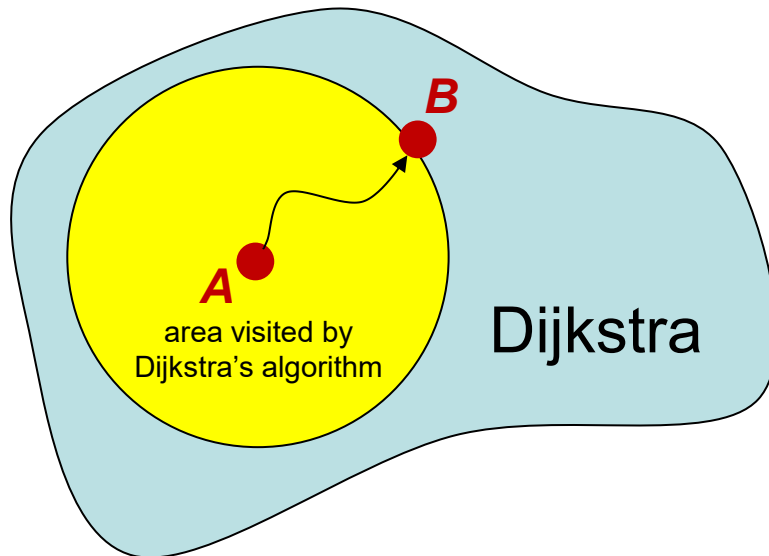
- Assume each node u has a **position** (u_x, u_y) and $w(u, v) \geq \delta(u, v) = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2}$
- $h(u) = -\delta(u, t) \Rightarrow w'(u, v) = w(u, v) + h(u) - h(v) \geq \delta(u, v) - \delta(u, t) + \delta(v, t) \geq 0$
- $d_{st} = d'_{st} - h(s) + h(t) = 2.00 + 5.00 - 0.00 = 7.00$



$$w'(u, v) = w(u, v) + h(u) - h(v)$$



In practice A* reduces the number of nodes visited by Dijkstra's algorithm

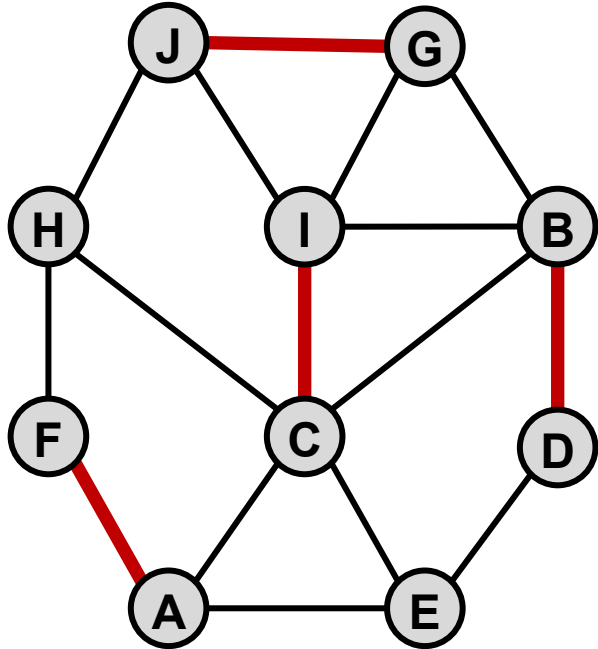


Algorithms and Data Structures

Matchings in graphs, maximum independent set,
minimum vertex cover

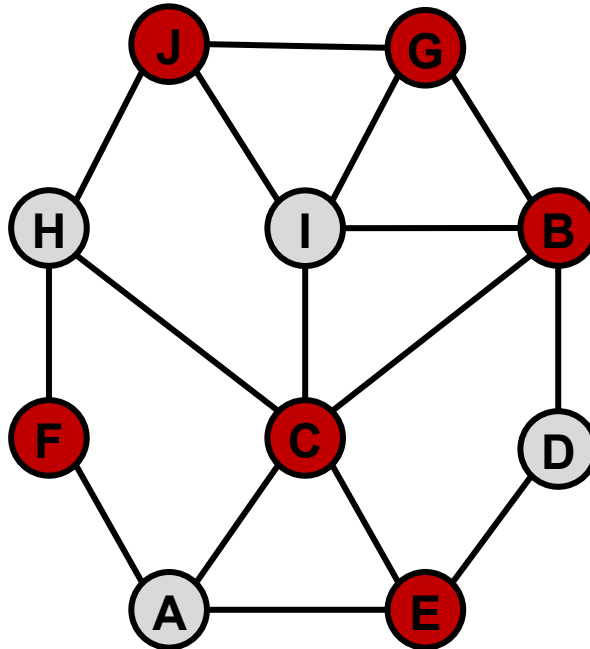
[CLRS, Chapters 24.3, 25.2, 34.4, 35.1]

Matching



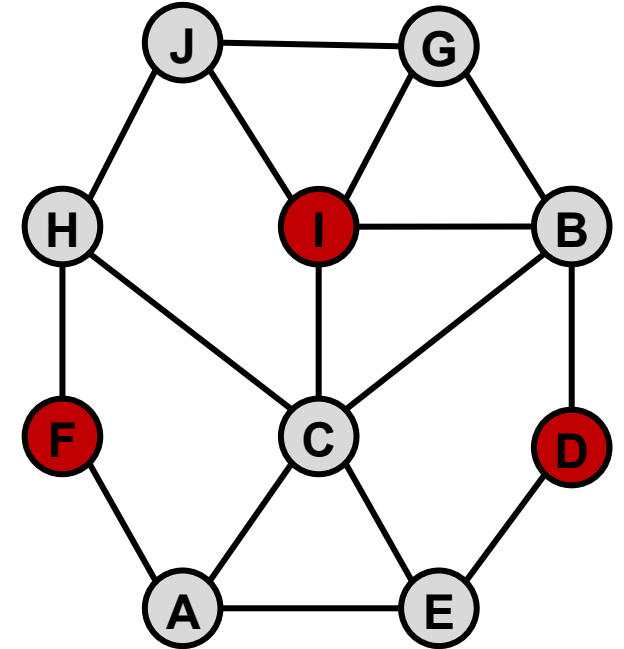
Subset of **edges**, where each node is part of at most one edge of the matching

Vertex cover



Subset of **nodes (a.k.a. vertices)**, where all edges in the graph contain at least one node from the cover

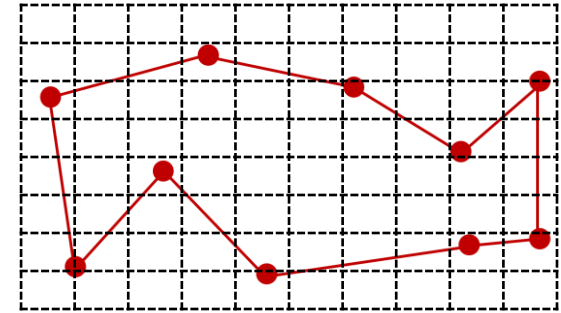
Independent set



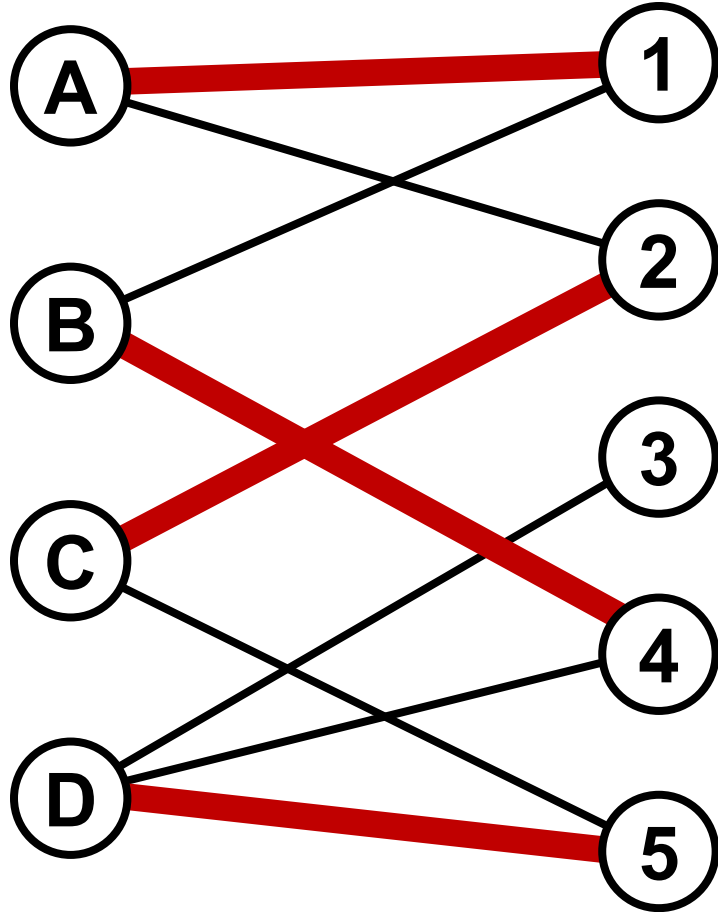
Subset of **nodes**, where no pair of nodes are connected by an edge

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 Prize 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**



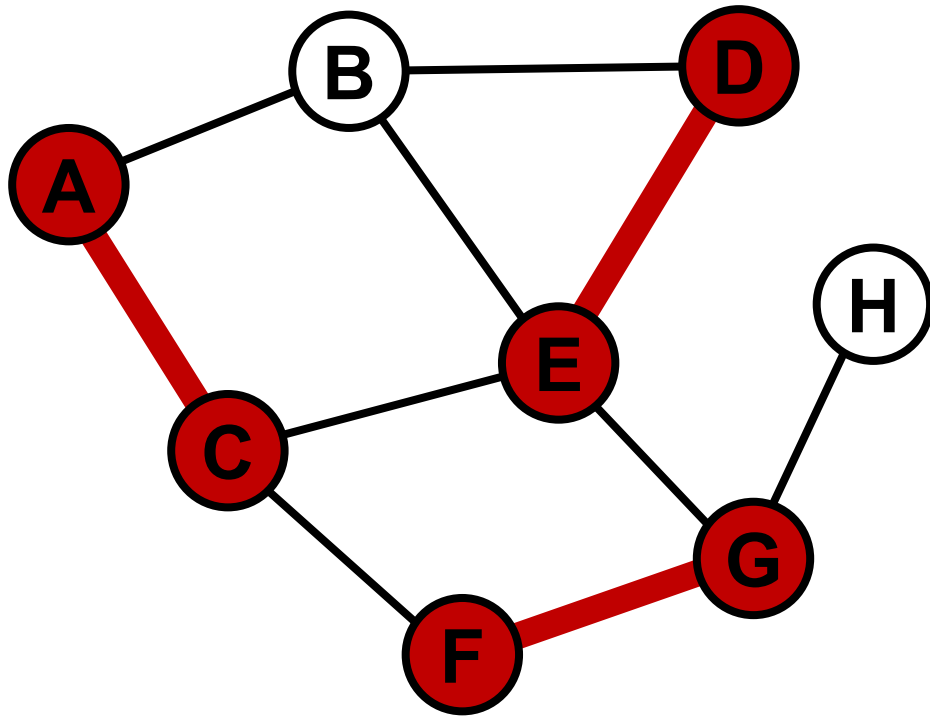
Matchings in bipartite graphs (special case)



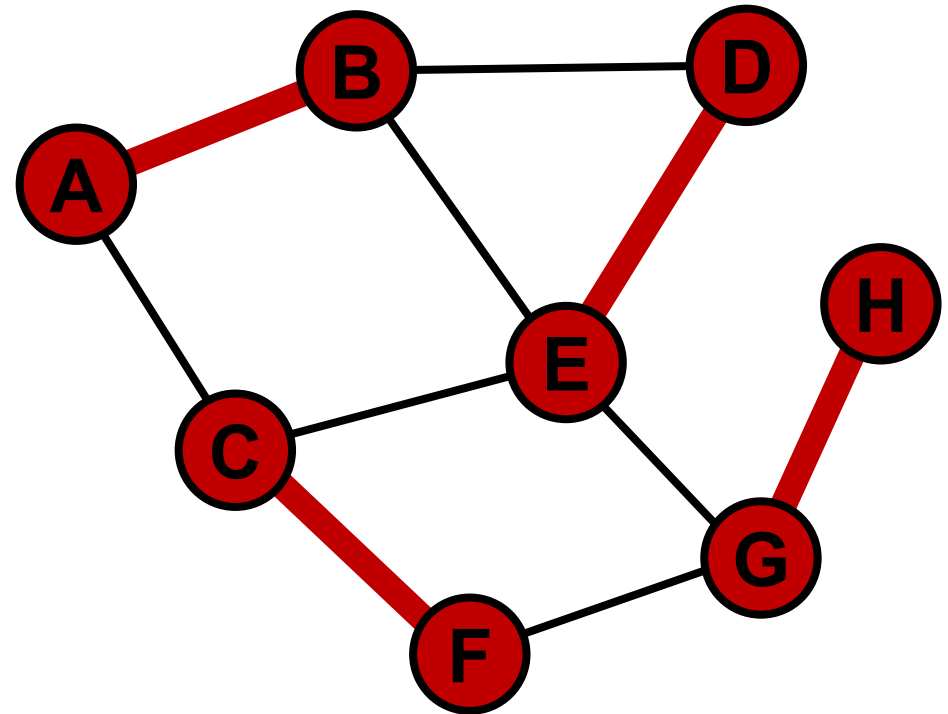
Examples

- **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



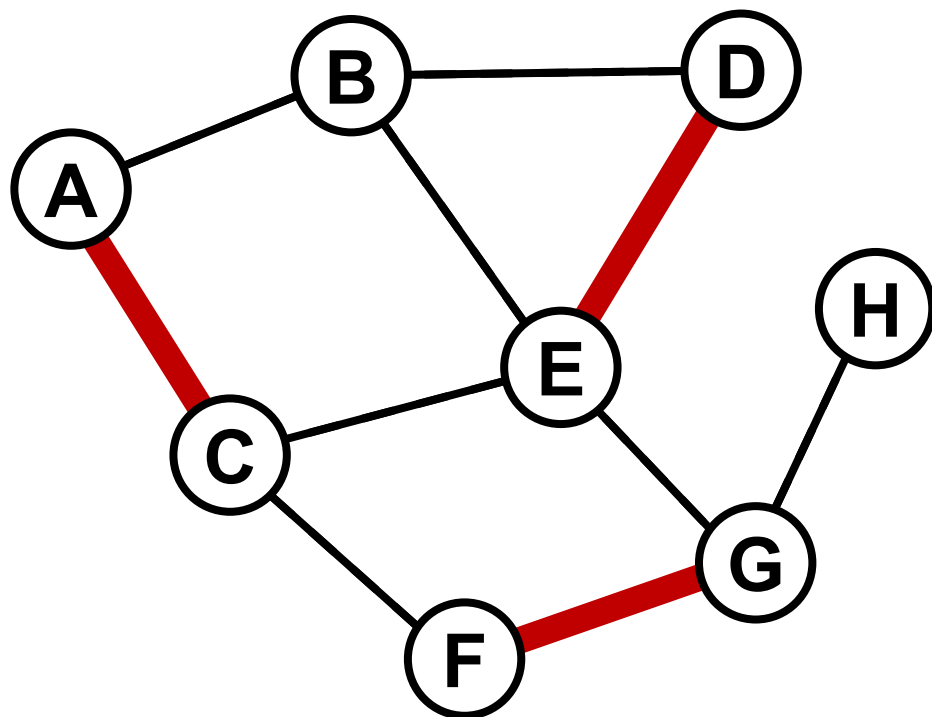
Maximal
cannot be extended



Maximum
largest possible matching

Lemma $|\text{Maximum matching}| / 2 \leq |\text{maximal matching}| \leq |\text{maximum matching}|$

Maximal matching construction

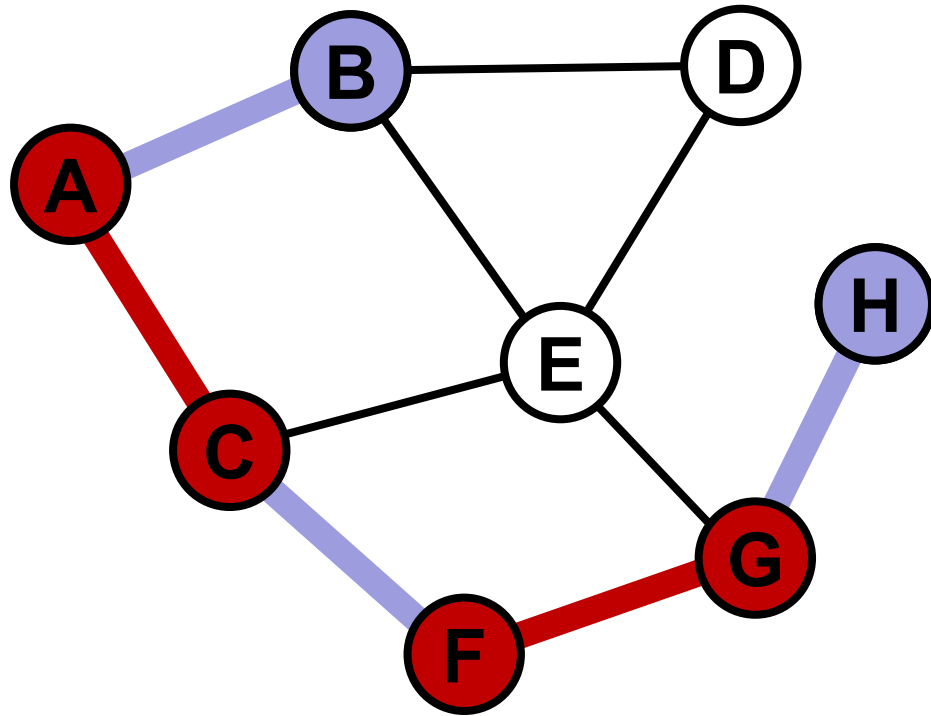


Greedy algorithm

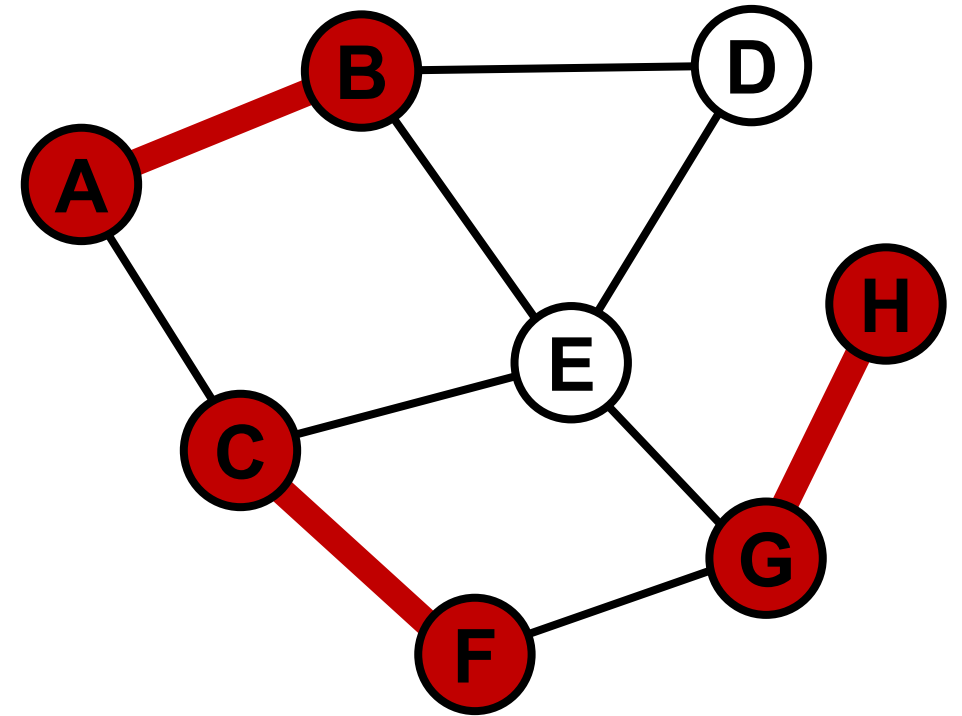
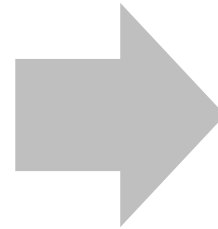
Continue adding arbitrary edges to the matching as long as possible

Time $O(n + m)$

Construction of a maximum matching – augmenting paths



Path between two unmatched nodes,
where every second edge in matching



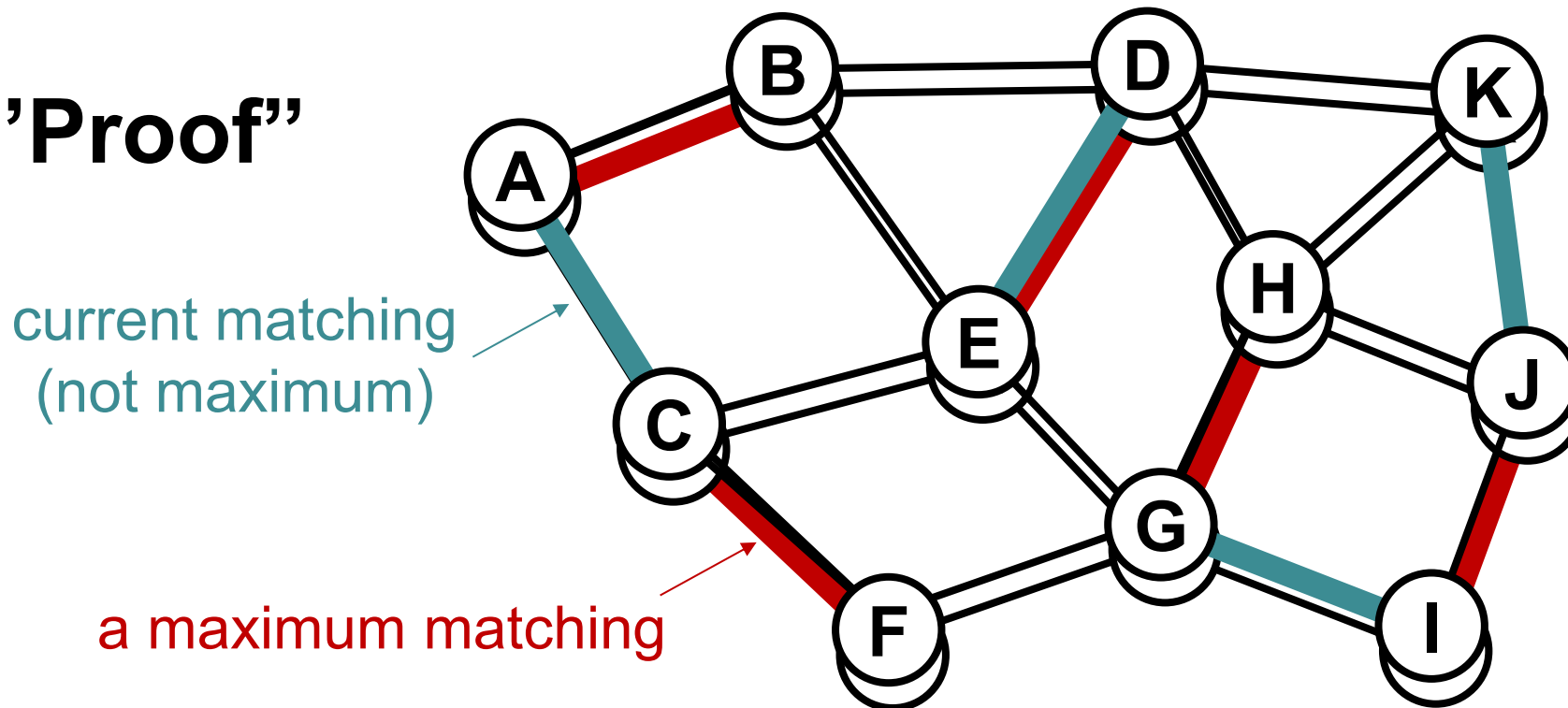
size of matching increased by one

Petersen 1891 / König 1931 / Berge 1957

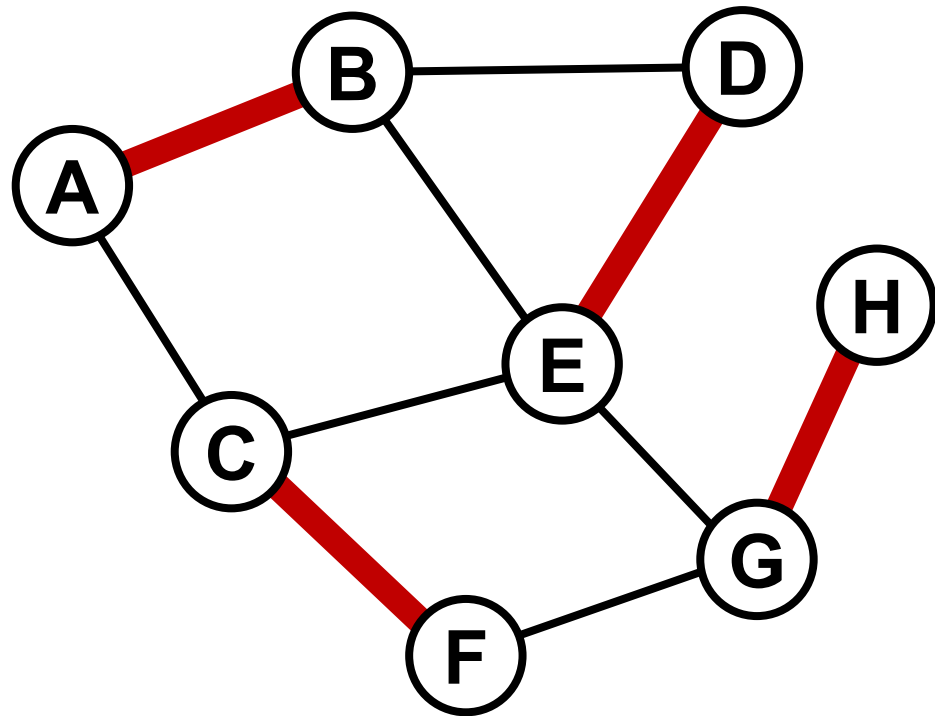
Theorem

A matching is maximum $\Leftrightarrow$ no augmenting path exists

”Proof”



Construction of a maximum matching



Greedy algorithm

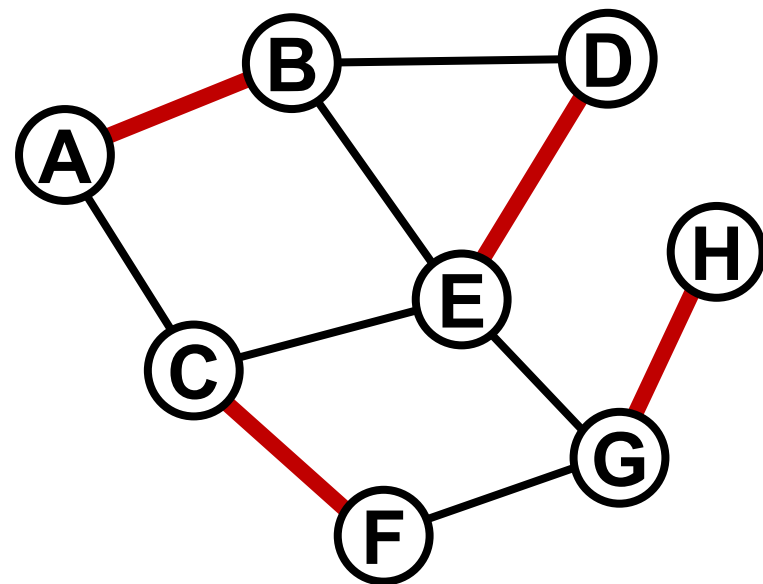
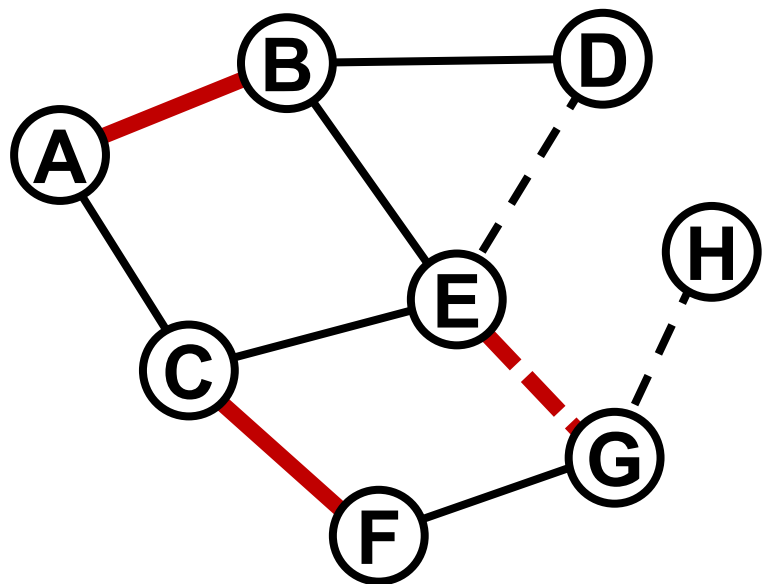
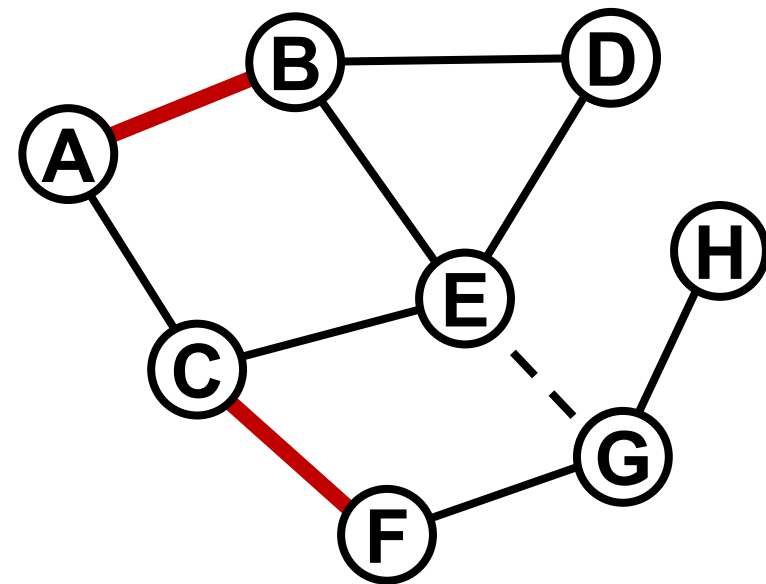
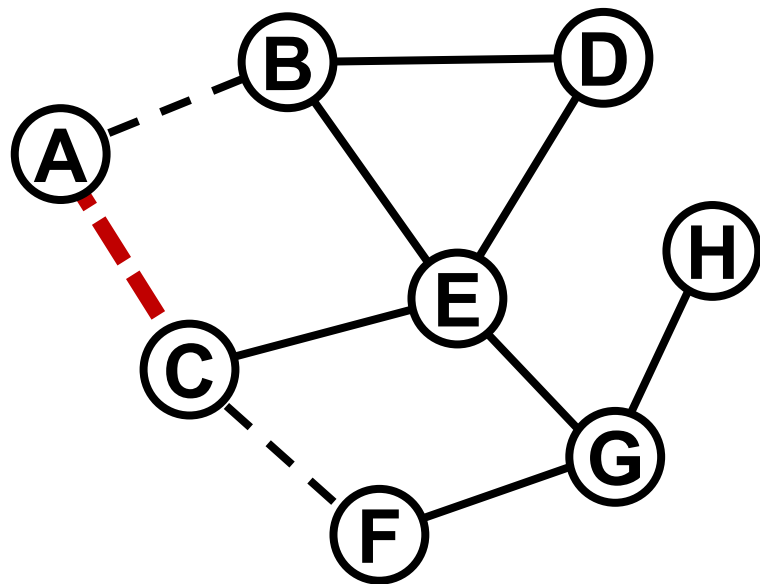
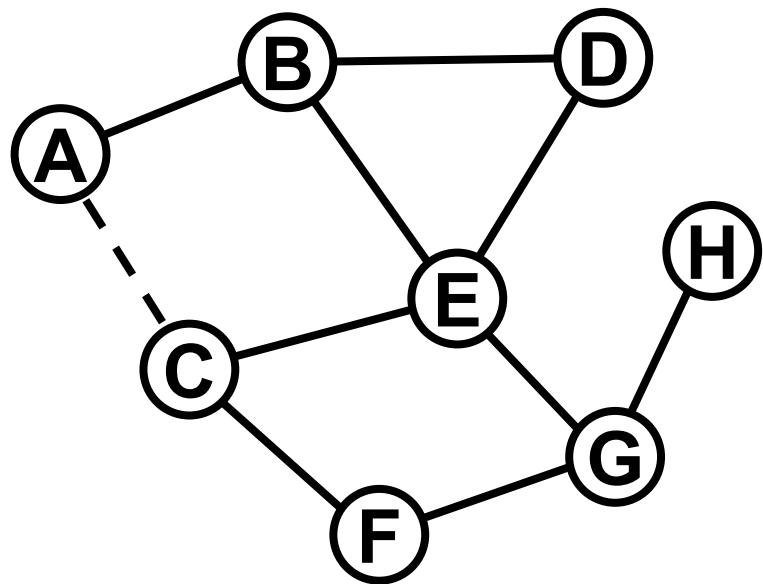
Apply augmenting paths
while possible

Edmonds 1961 $O(m \cdot n^2)$

Micali, Vazirani 1980 $O(m \cdot n^{1/2})$

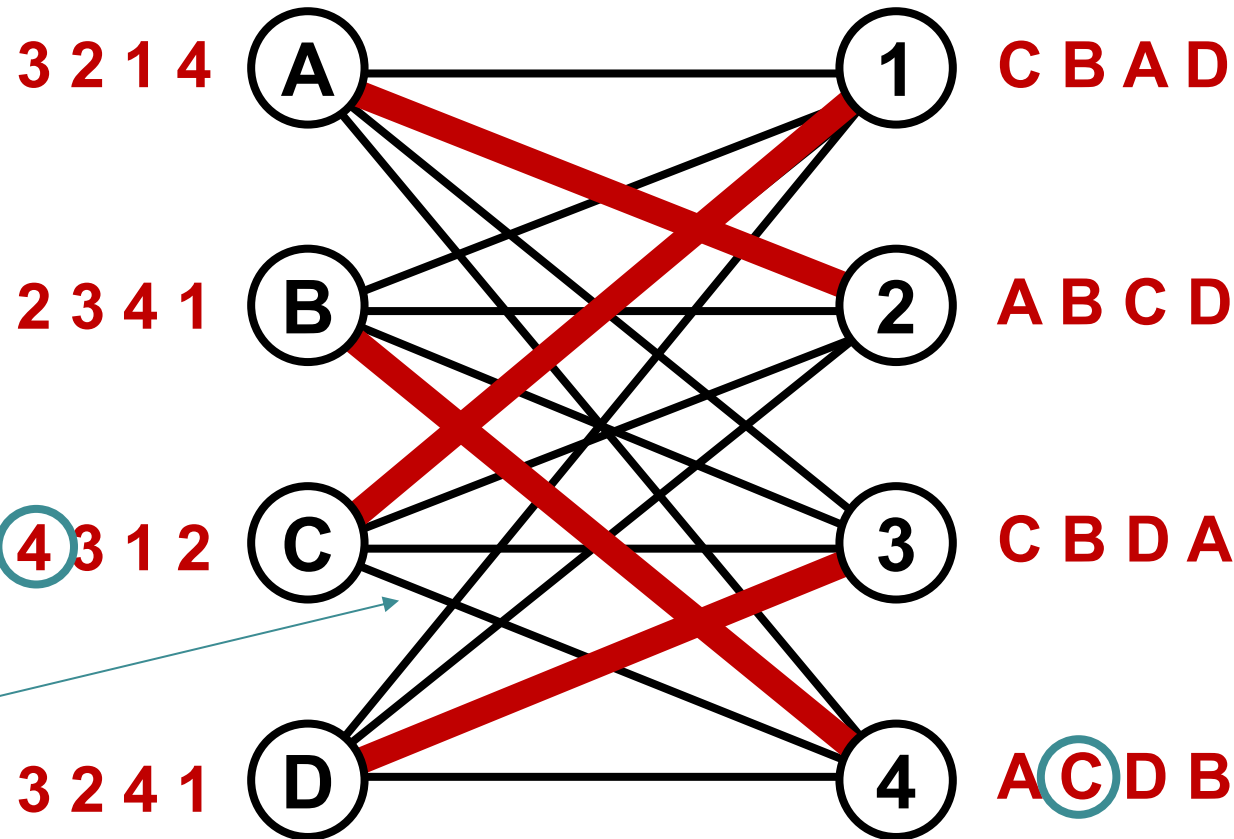
Jack Edmonds, *Paths, trees, and flowers*, Canadian Journal of Mathematics, 1965, doi: [10.4153/CJM-1965-045-4](https://doi.org/10.4153/CJM-1965-045-4)

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](https://doi.org/10.1109/SFCS.1980.12)



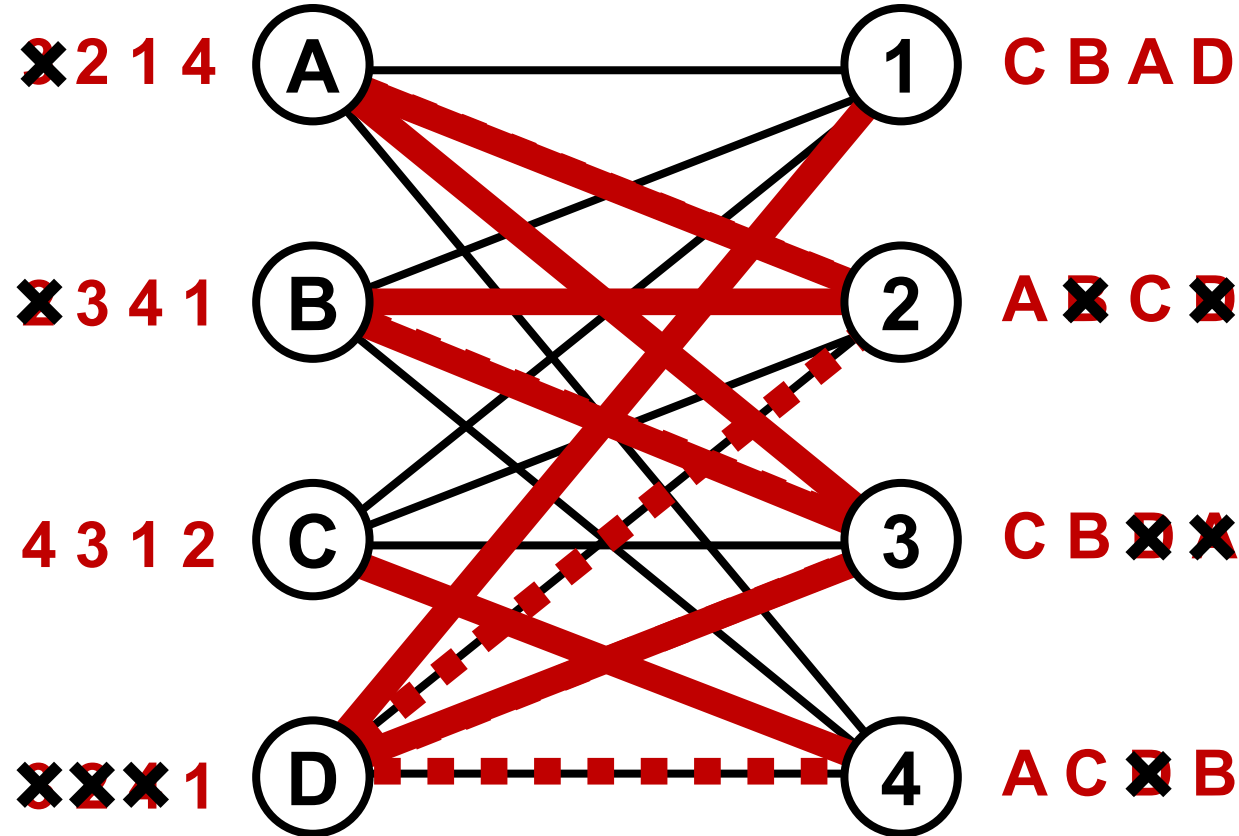
Stable marriage problem

preferences



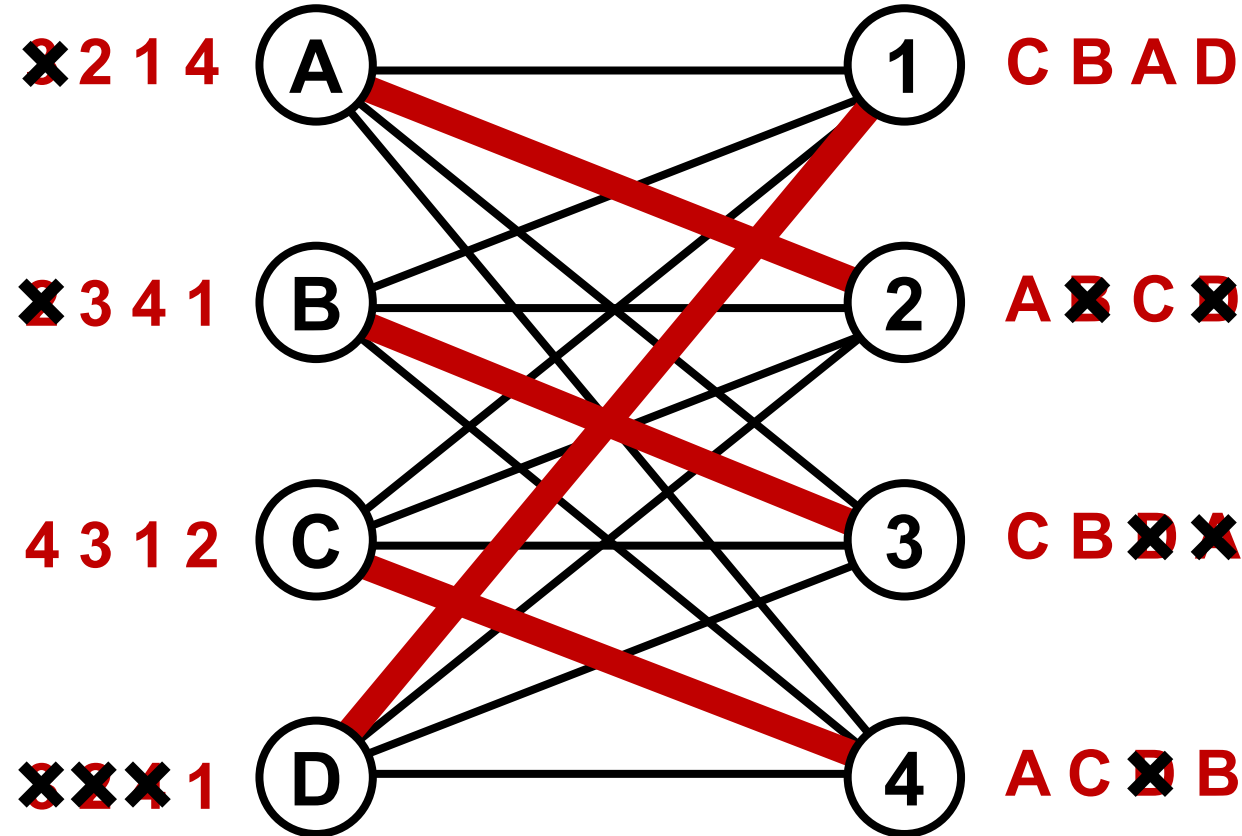
complete bipartite graph with equal
number of nodes on each side

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

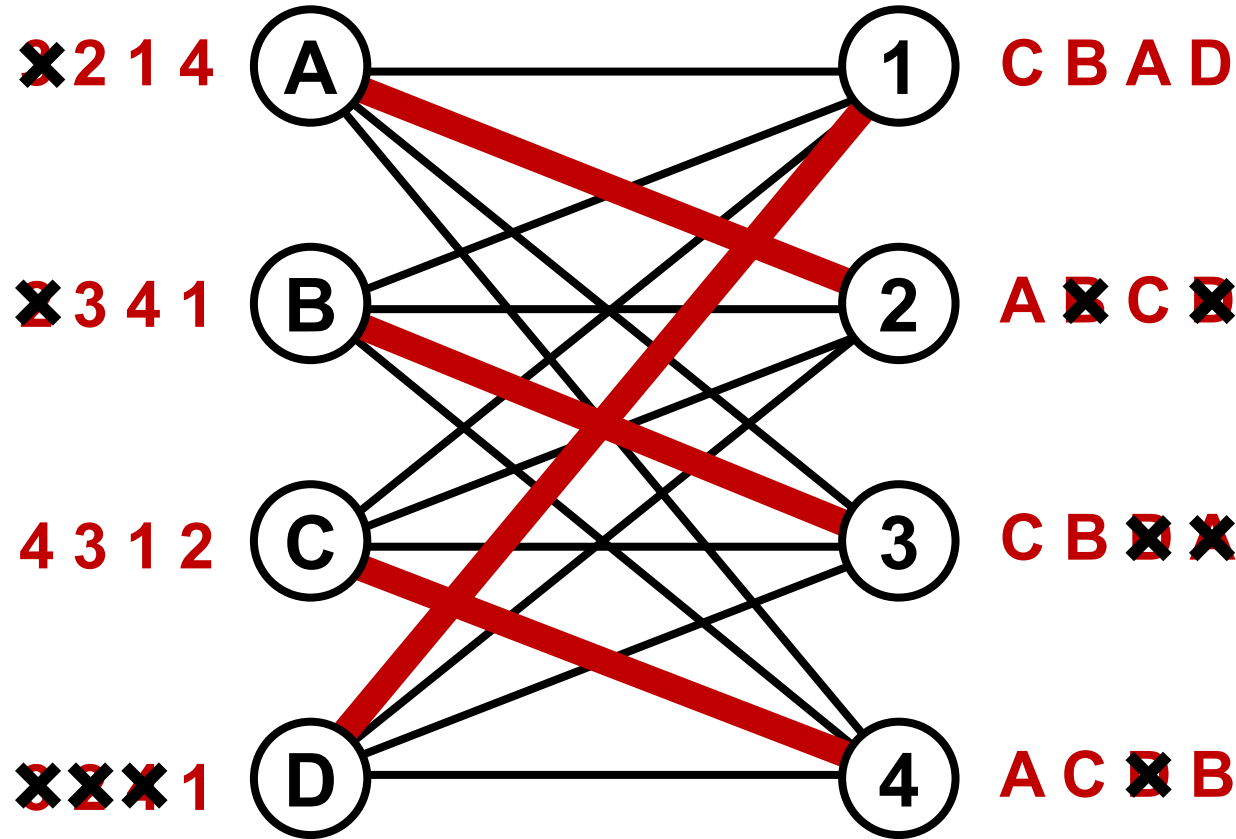


If left X rejected by right i , then i is matched with some Y preferred over X
 X is matched with the most preferred, not yet rejected X

Time $O(n^2)$

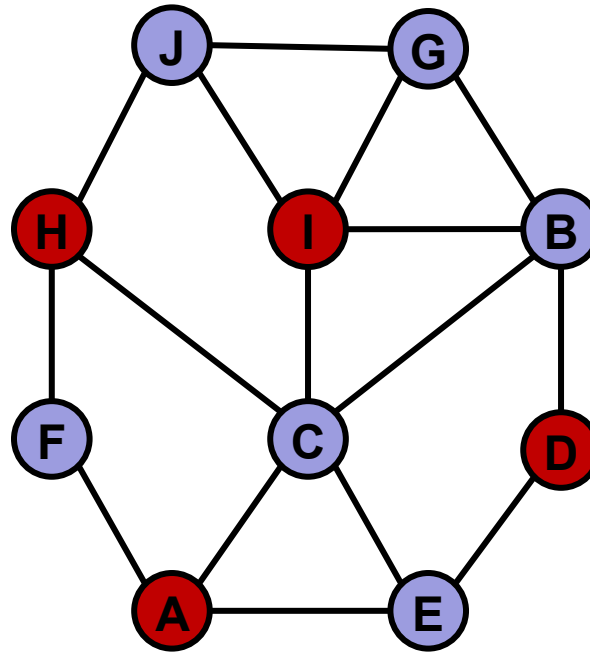
Theorem (Gale, Shapley 1962)

A stable marriage always exist



(Shapley got the Nobel price in economics for the result in 2012)

Independent set vs vertex cover



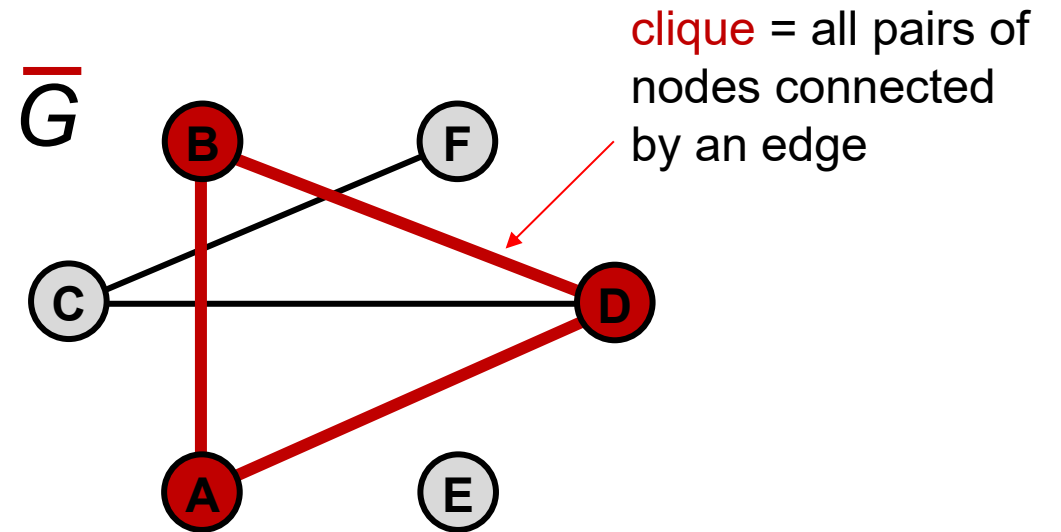
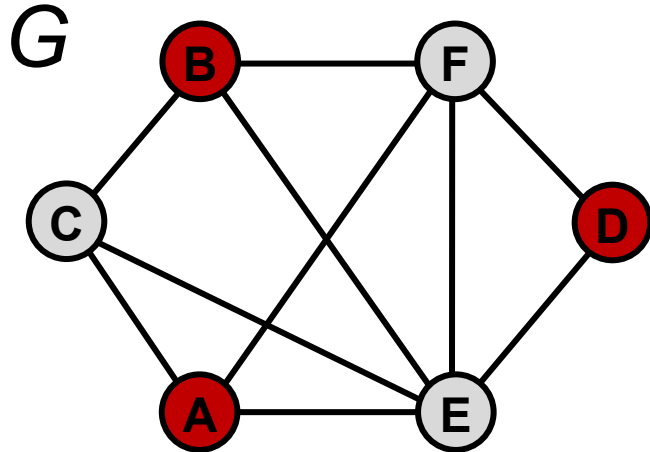
Lemma

$I \subseteq V$ independent set $\Leftrightarrow V \setminus I$ covers all edges

Theorem

$I \subseteq V$ maximum independent set $\Leftrightarrow V \setminus I$ minimum vertex cover

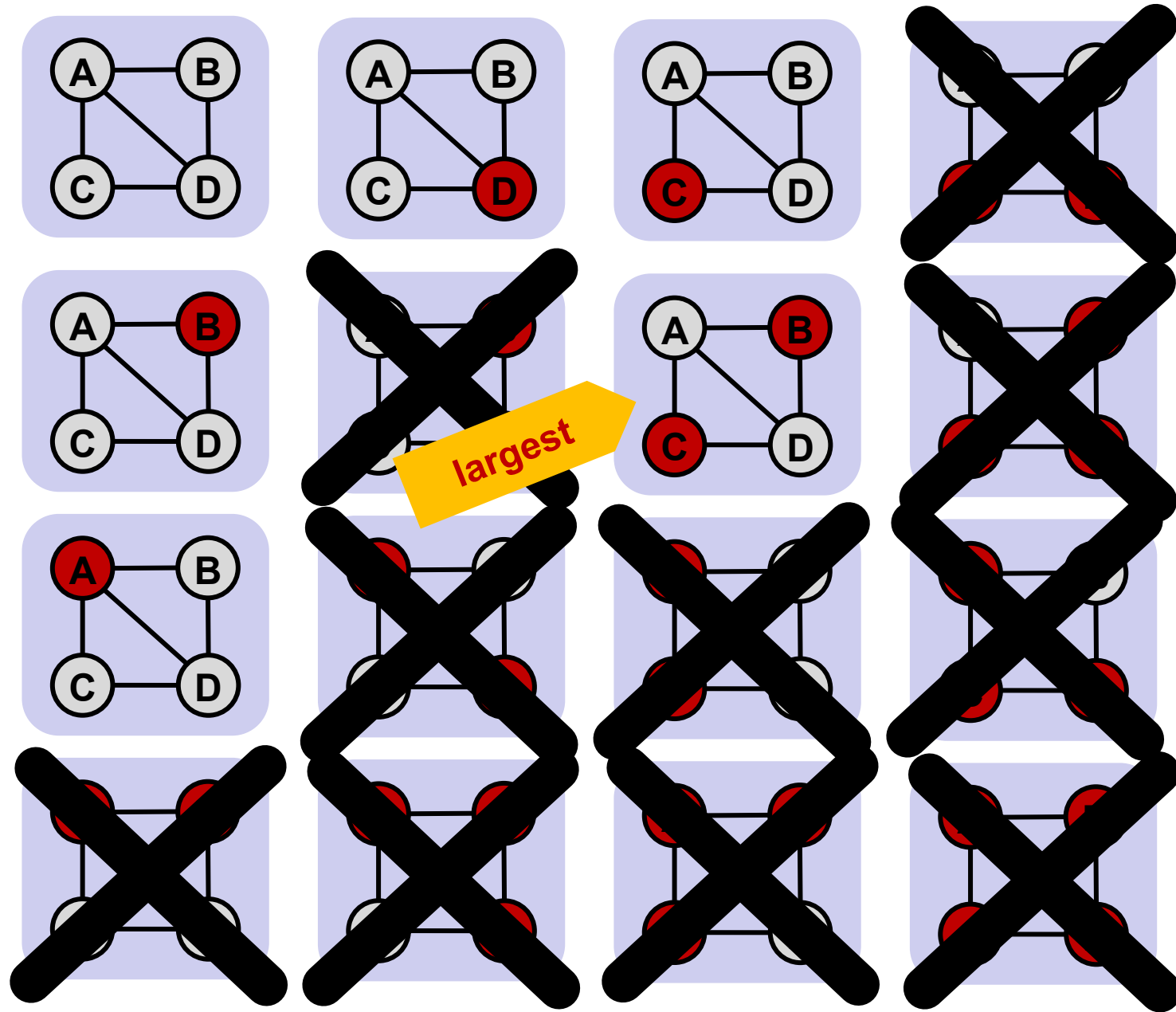
Independent set vs clique



Theorem

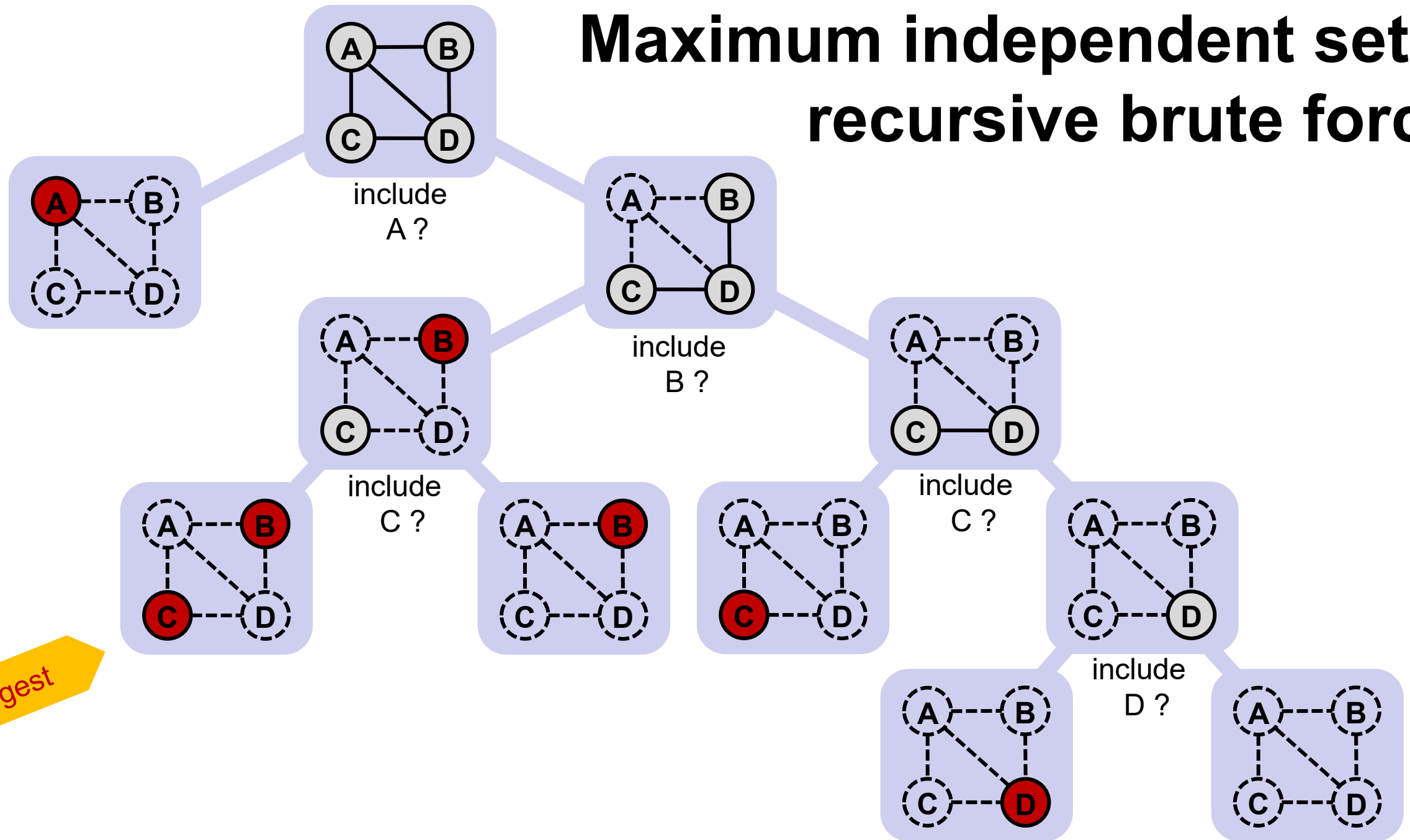
$I \subseteq V$ maximum independent set in $G \iff I$ maximum clique in $\overline{G}$

Maximum independent set – try all 2^n subsets



Time $O(m \cdot 2^n)$

Maximum independent set – recursive brute force

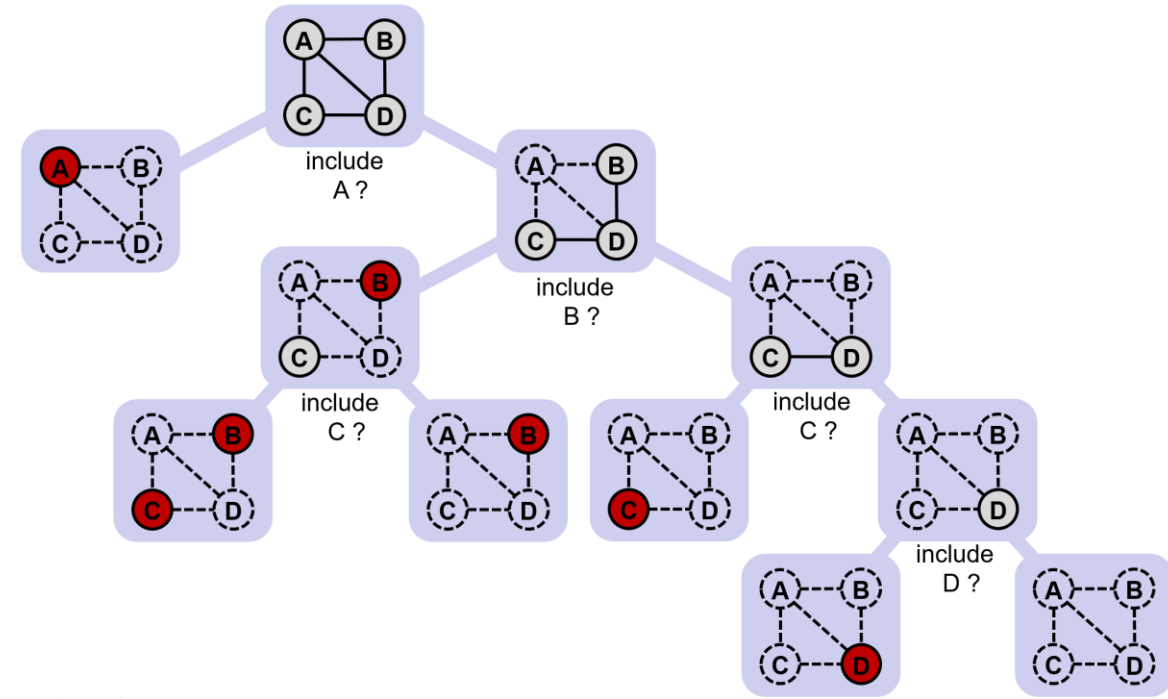


Maximum independent set

Algorithm MIS1(G)

```

1  if  $V_G = \emptyset$  then
2    return  $\emptyset$ 
3  let  $v \in V_G$  be an arbitrary vertex
4   $I = \text{MIS1}(G \setminus \{v\})$ 
5   $I' = \{v\} \cup \text{MIS1}(G \setminus \{v\} \setminus \text{adj}_G(v))$ 
6  if  $|I'| > |I|$  then
7     $I = I'$ 
8  return  $I$ 
    
```

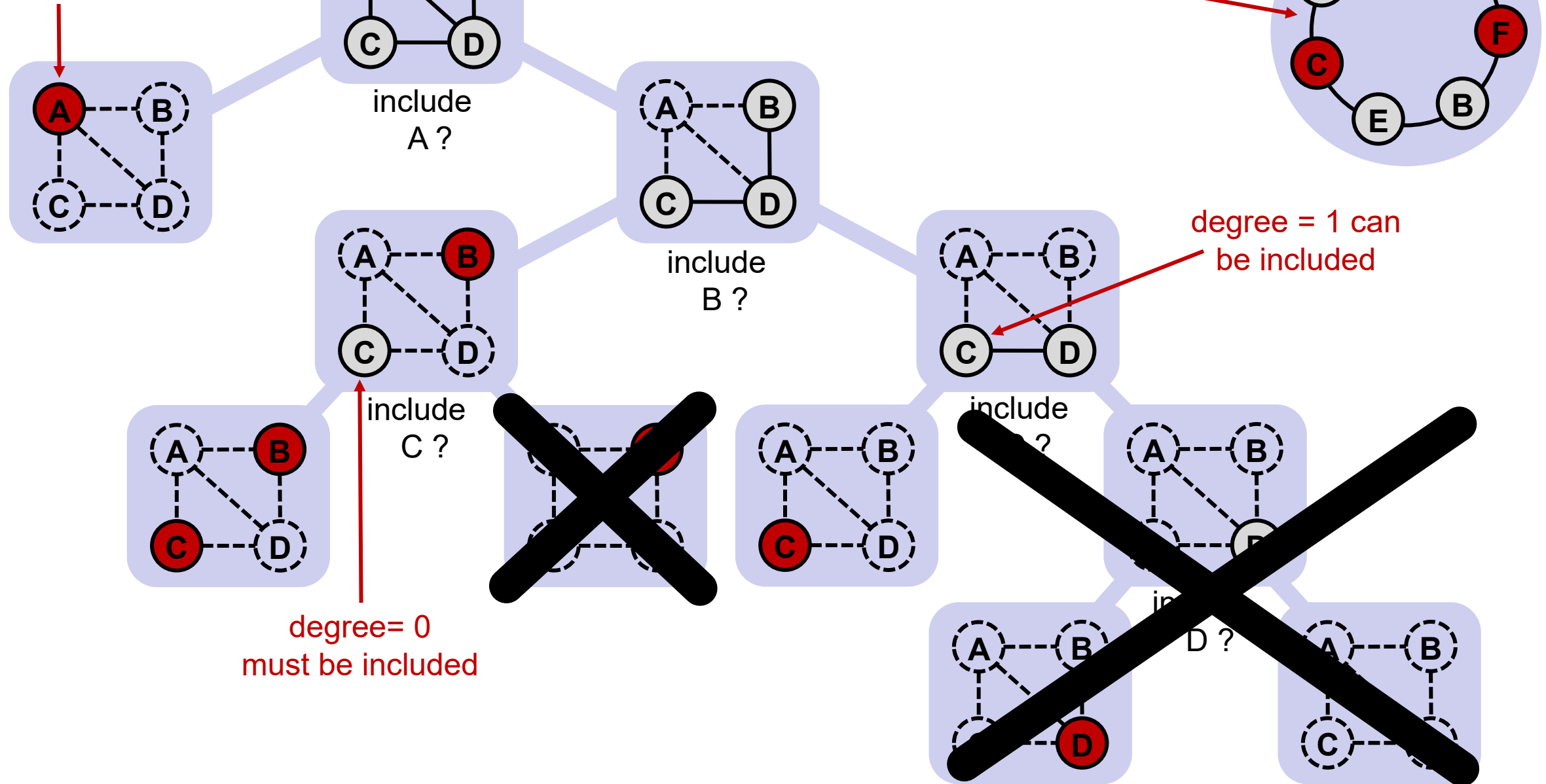


Analysis

recursive calls

$$T(n) \leq \begin{cases} 1 & \text{if } n = 0 \\ 1 + 2 \cdot T(n-1) & \text{otherwise} \end{cases} \Rightarrow T(n) \leq 2^{n+1} - 1$$

high degree removes
many neighbors from
one recursive call



Algorithm MIS2 (G)

```
1   $I = \emptyset$ 
2  while  $\exists v \in V_G : |\text{adj}_G(v)| \leq 1$  do
3       $I = I \cup \{v\}$ 
4       $G = G \setminus \{v\} \setminus \text{adj}_G(v)$ 
5  # all vertices have degree  $\geq 2$ 
6  if  $\exists v \in V_G : |\text{adj}_G(v)| \geq 3$  then
7       $I_1 = I \cup \text{MIS2}(G \setminus \{v\})$ 
8       $I_2 = I \cup \{v\} \cup \text{MIS2}(G \setminus \{v\} \setminus \text{adj}_G(v))$ 
9      return  $I_1$  if  $|I_1| \geq |I_2|$  otherwise  $I_2$ 
10 # all remaining vertices have degree 2
11 while  $\exists$  cycle  $C = (c_1, c_2, \dots, c_k)$  in  $G$  do
12      $I = I \cup \{c_1, c_3, c_5, \dots, c_{2\lfloor k/2 \rfloor}\}$ 
13      $G = G \setminus C$ 
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 $T(n) \leq \begin{cases} 1 & \text{if } n = 0 \\ 1 + T(n-1) + T(n-4) & \text{otherwise} \end{cases} \Rightarrow T(n) = O(1.38028^n)$

Reduction – 3-SAT to independent set

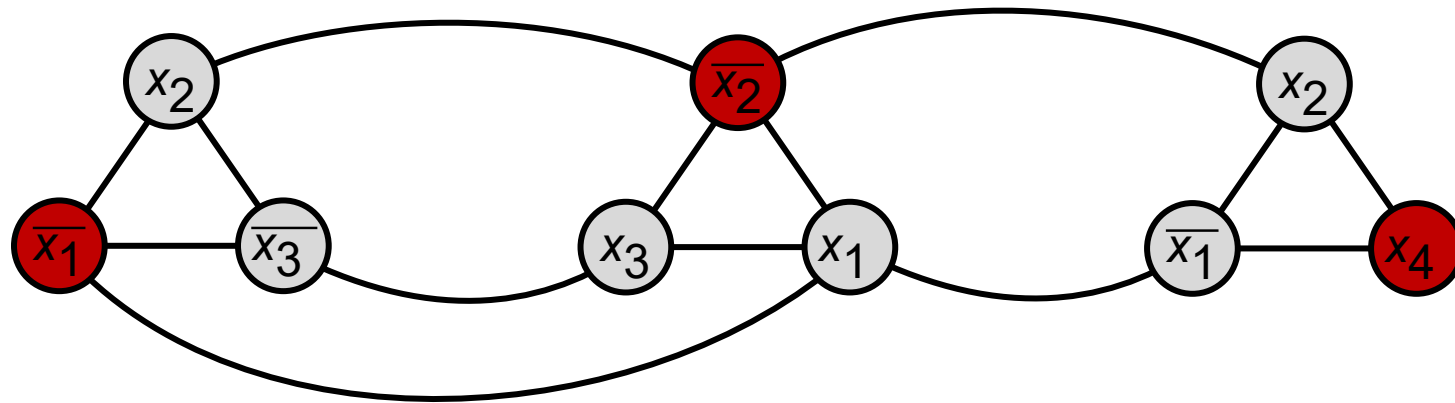
- **3-SAT** Given a Boolean formula in conjunctive normal form (CNF), where all n clauses have three literals, e.g.

$$\varphi = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

- Exists assignment of true/false to the literals $x_1, x_2, \dots$ such that the equation is true?

$$x_1 = x_2 = \text{false}, \quad x_3 = x_4 = \text{true}$$

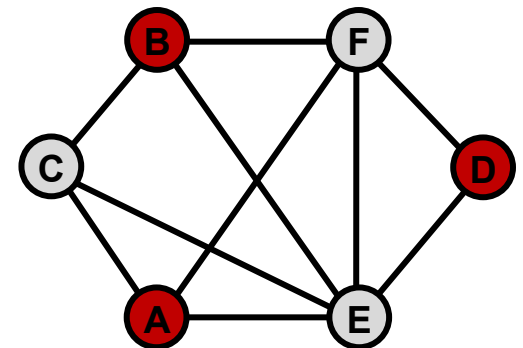
- **Reduction** Create one triangle per clause, one node per literal. Add edges $(\overline{x_i}, x_i)$.



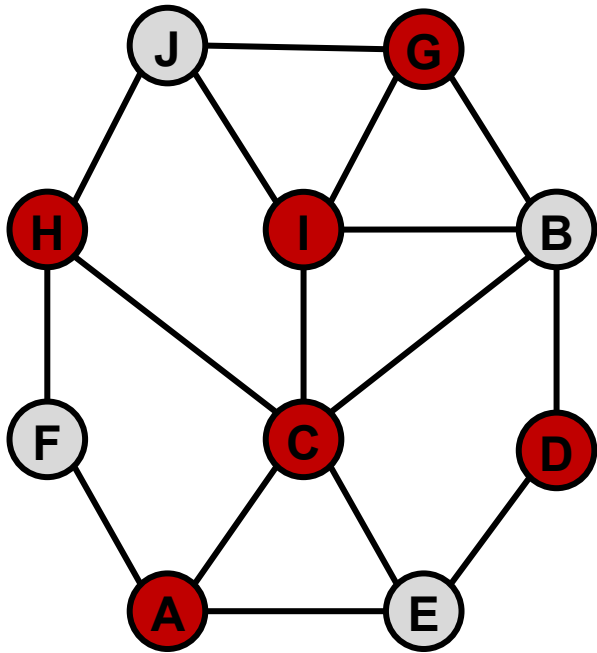
- **Theorem** 3-SAT formula satisfiable $\Leftrightarrow$ maximum independent set of size n

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]



2-approximation of minimum vertex cover



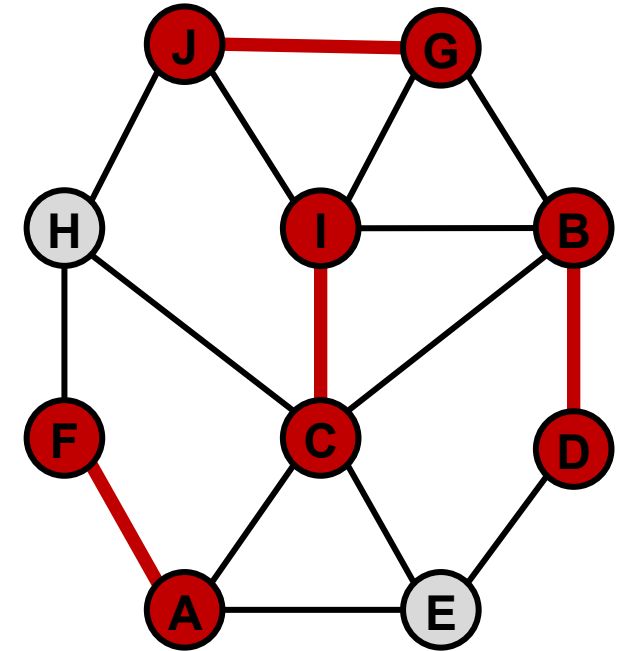
Minimum
vertex cover C_{optimal}

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 $\mathbf{P} \neq \mathbf{NP}$ (the big open problem in computer science)
- factor 2, provided the "unique games conjecture" is true (i.e., the above is best possible)

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

Fedor V. Fomin
Dieter Kratsch

Exact Exponential Algorithms

 Springer

Explains common algorithmic techniques for developing exact (exponential time) algorithms for NP-hard problems

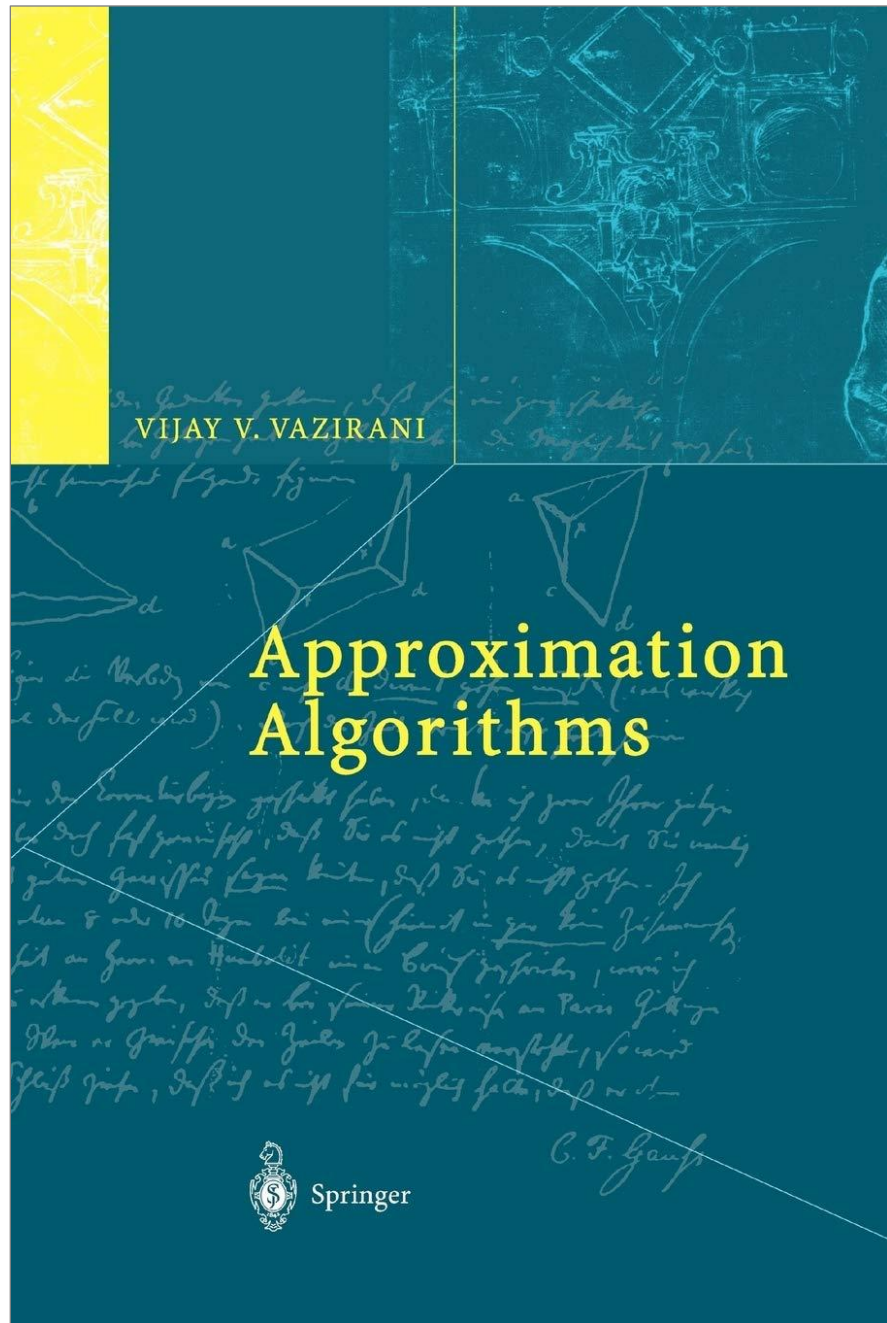
Exact Exponential Algorithms

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](https://doi.org/10.1007/978-3-642-16533-7)



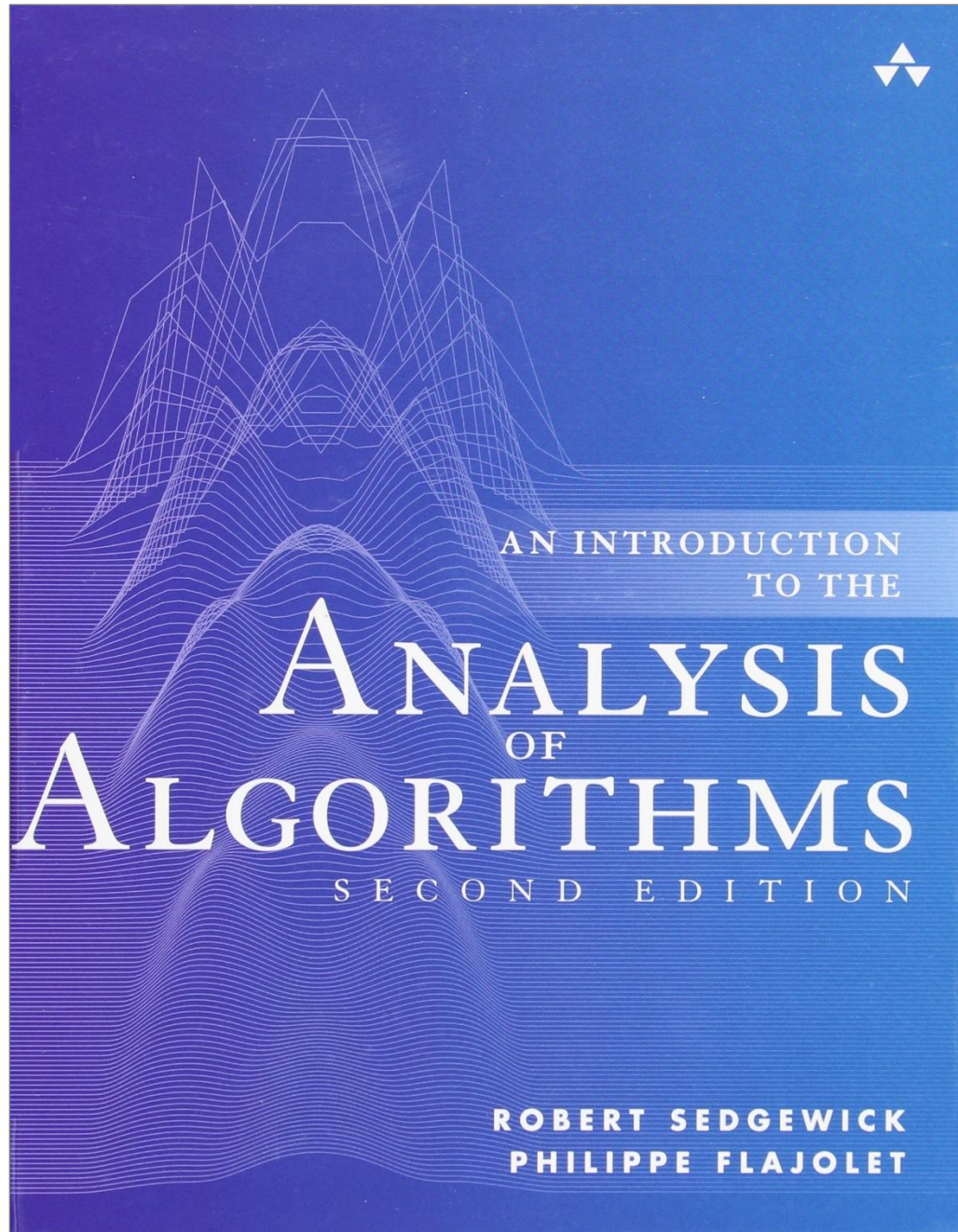
Explains the main theoretical approaches to find approximate solutions to hard combinatorial optimization and enumeration problems

Approximation Algorithms

Vijay V. Vazirani

XIX - 380, Springer-Verlag Berlin Heidelberg 2003

doi: [10.1007/978-3-662-04565-7](https://doi.org/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

Algorithms and Data Structures (520251E001)		Participant index			
Monday, January 5, 2026, 10:00–12:00					
1.1	☐ A ☐ B	13.1	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G		
1.2	☐ A ☐ B	13.2	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G		
1.3	☐ A ☐ B	13.3	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G		
1.4	☐ A ☐ B	13.4	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G		
1.5	☐ A ☐ B	13.5	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G		
1.6	☐ A ☐ B	14	☐ A ☐ B ☐ C ☐ D		
1.7	☐ A ☐ B	15.1	☐ A ☐ B		
1.8	☐ A ☐ B	15.2	☐ A ☐ B		
1.9	☐ A ☐ B	15.3	☐ A ☐ B		
1.10	☐ A ☐ B	15.4	☐ A ☐ B		
1.11	☐ A ☐ B	15.5	☐ A ☐ B		
1.12	☐ A ☐ B	16.1	☐ A ☐ B ☐ C ☐ D		
2.1	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H	16.2	☐ A ☐ B ☐ C ☐ D		
2.2	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H	16.3	☐ A ☐ B ☐ C ☐ D		
2.3	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H	16.4	☐ A ☐ B ☐ C ☐ D		
2.4	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H	16.5	☐ A ☐ B ☐ C ☐ D		
3.1	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	17	☐ A ☐ B ☐ C ☐ D		
3.2	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	18	☐ A ☐ B ☐ C ☐ D ☐ E		
3.3	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	19.1	☐ A ☐ B		
3.4	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	19.2	☐ A ☐ B		
3.5	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	19.3	☐ A ☐ B		
4	☐ A ☐ B ☐ C ☐ D ☐ E	19.4	☐ A ☐ B		
5	☐ A ☐ B ☐ C ☐ D ☐ E	19.5	☐ A ☐ B		
6	☐ A ☐ B ☐ C ☐ D	20.1	☐ A ☐ B ☐ C ☐ D		
7	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F	20.2	☐ A ☐ B ☐ C ☐ D		
8.1	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	20.3	☐ A ☐ B ☐ C ☐ D		
8.2	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	20.4	☐ A ☐ B ☐ C ☐ D		
8.3	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	20.5	☐ A ☐ B ☐ C ☐ D		
8.4	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	21.1	☐ A ☐ B		
8.5	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	21.2	☐ A ☐ B		
9.1	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	21.3	☐ A ☐ B		
9.2	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	21.4	☐ A ☐ B		
9.3	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	21.5	☐ A ☐ B		
9.4	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	22.1	☐ A ☐ B		
9.5	☐ A ☐ B ☐ C ☐ D ☐ E ☐ F ☐ G ☐ H ☐ I ☐ J ☐ K	22.2	☐ A ☐ B		
10.1	☐ A ☐ B	23.1	☐ A ☐ B		
10.2	☐ A ☐ B	23.2	☐ A ☐ B		
10.3	☐ A ☐ B	23.3	☐ A ☐ B		
10.4	☐ A ☐ B	24.1	☐ A ☐ B		
10.5	☐ A ☐ B	24.2	☐ A ☐ B		
11	☐ A ☐ B ☐ C ☐ D	24.3	☐ A ☐ B		
12	☐ A ☐ B ☐ C ☐ D				

Anonymous
Participant index

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

1.1	☐ A	☒ B	☐ C	☐ D	☐ E
1.2	☐ A	☐ B	☐ C	☒	☐ E

Invalid markings

2.1	☐ A	☐ B	☒	☒	☐ E
2.2	☒	☐ B	☐ C	☐ D	☐ E
2.3	☐ A	☐ B	☐ C	☒	☐ E
2.4	☐ A	☒	☐ C	☐ D	☐ E
2.5	☐ A	☐ B	☐ C	☐ D	☒

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.