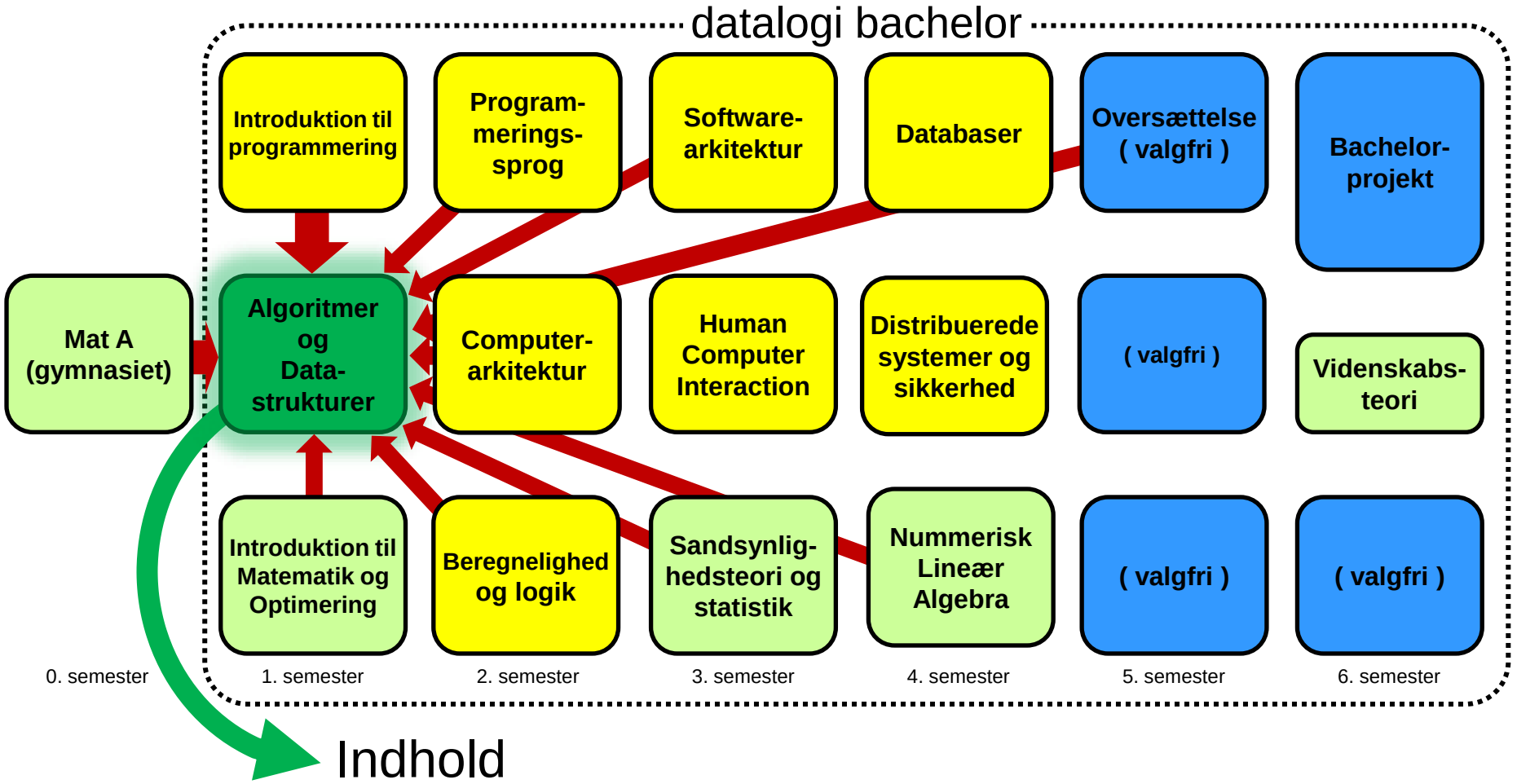
Algoritmer og Datastrukturer

Gerth Stølting Brodal
gerth@cs.au.dk

faglige forudsætninger
(ideelle verden)

Eksamen : januar
Reeksamen : maj



Indhold

- **Introduktion til algoritmer og datastrukturer**
- Eksempler på anvendelse
- Avancerede algoritmer og datastrukturer
- Forskningsresultater

ambitiøst

Kursuselementer

Undervisningsformer
Forelæsninger 2 + 2 timer/uge [fysisk + optages + efterfølgende video]
Teoretiske øvelser "TØ" 3 timer/uge [fysisk]
Studiecafé 5 x 2 timer/uge [fysisk]
Online diskussionsforum [Brightspace]

Forelæsningerne gennemgår stoffet
Til øvelserne *arbejder* man med stoffet

Hjælp til ugeopgaver og afleveringer af en instruktor (cs.au.dk/studiecafe)

Afleveringer (grupper 1-3 personer)
9 teoretiske afleveringer (skal godkendes)
4 ugers programeringsopgaver (del af karakteren)

Træner algoritmiske formuleringer
Afleveres på Brightspace
Frist aftales med instruktoren
Genaflevering (positivt) = instruktor feedback og mulighed for at forbedre sig
Anbefales grupper af 2-3 personer for løbende at kunne diskutere opgaver

Eksamen
2 timers multiple choice, uden hjælpemidler, 7-skala

Sprog
Forelæsninger og øvelser dansk
(materialle og kursusside dog engelsk)

Tidligere eksamensopgaver og træningsopgaver på kursussiden

Tidsforbrug	timer x uger
Forelæsninger	4 x 14 = 56
Teoretiske øvelser (TØ)	3 x 14 = 42
Studiecafé	1 x 14 = 14
Afleveringer	3 x 14 = 42
Forberedelse forelæsning	2 x 14 = 28
Forberedelse TØ	2 x 14 = 28
Forberedelse eksamen	45
Eksamen	2
Timer i alt	**257**

3 timer om ugen med en instruktor

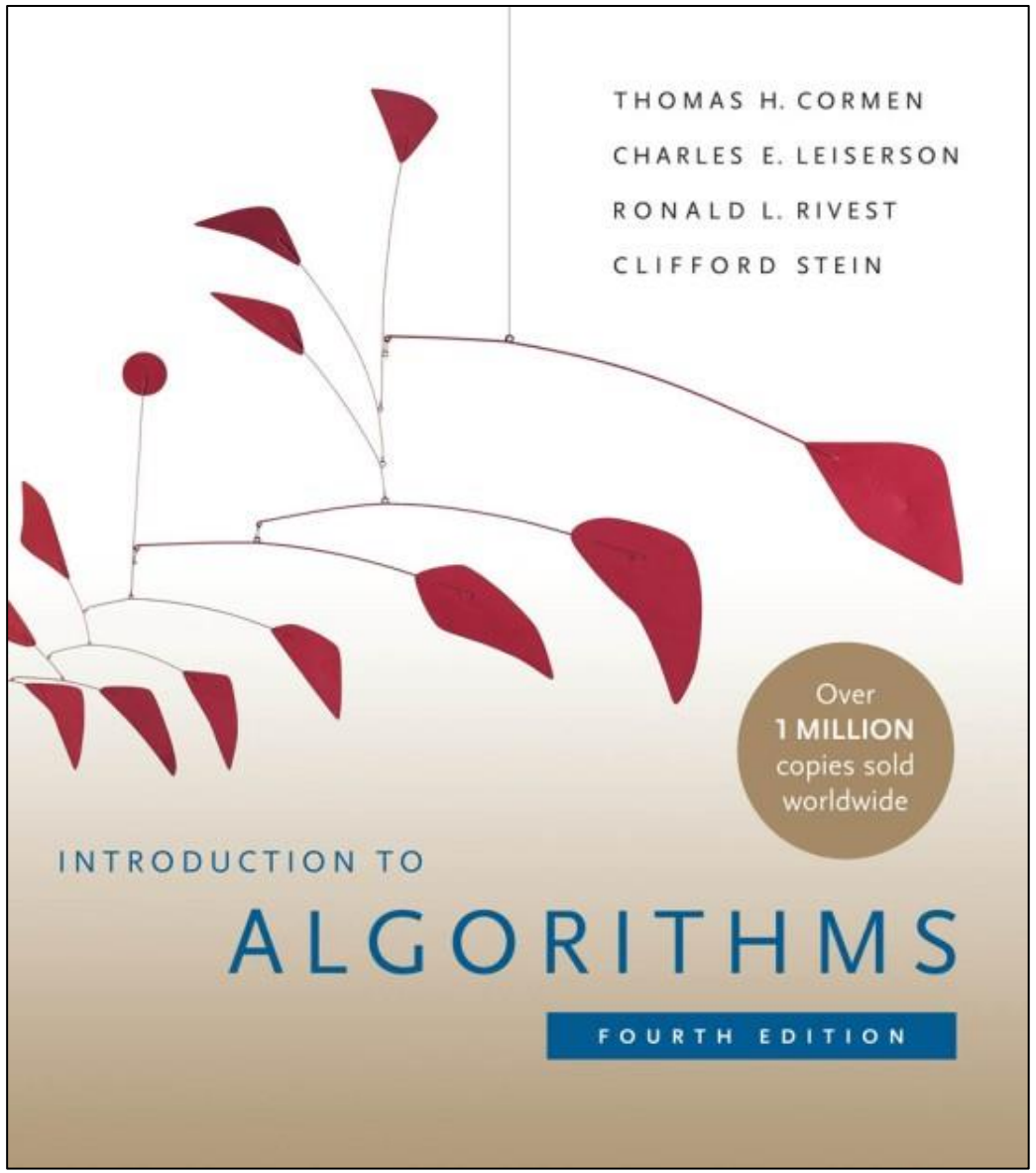
En række opgaver til hver gang

Opgaverne på "Course plan" hørende til forelæsninger *efter* jeres sidste TØ time, og *op til* denne TØ time

Format

1. I forbereder jer inden TØ i læsegrupperne og forsøger at forstå og løse alle opgaverne
   Brug studiecafé & diskussionsforummet til hjælp
2. Ved øvelserne gennemgåes opgaverne af de studerende, imens instruktoren hjælper med at rette misforståelser og fremhæve vigtige pointer

Sørg for løbende dialog med instruktoren om hvordan TØ timerne bruges bedst muligt



Primær lærebog

Supplementary Lecture notes

Algorithms and Data Structures

Gerth Stølting Brodal

Opdateres løbende igennem kurset

Om 0 er med i de naturlige tal
afhænger af hvem man spørger !

Et ”cheat sheet” med matematisk notation og regneregler der anvendes i kurset findes i ”Supplementary Lecture Notes”

Mathematics Cheat Sheet

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

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

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

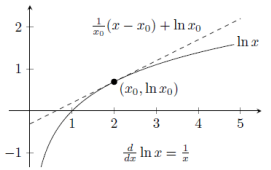
Distributivity
$a \cdot (b + c) = a \cdot b + a \cdot c$

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

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

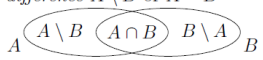
Fractions
$\frac{a}{b} \cdot \frac{c}{d} = \frac{a \cdot c}{b \cdot d}$
$\frac{a}{b} / \frac{c}{d} = \frac{a \cdot d}{b \cdot c}$
$\frac{a}{b} + \frac{c}{d} = \frac{a \cdot d + b \cdot c}{b \cdot d}$

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



Matrices (1,4)
$$A = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \quad m \times n \text{ matrix}$$
Matrix addition ($m \times n$ og $m \times n$)
$C = A + B, \quad c_{ij} = a_{ij} + b_{ij}$
Matrix multiplication ($m \times n$ og $n \times p$)
$C = AB, \quad c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$
Constant multiplication ($m \times n$)
$C = cA, \quad c_{ij} = c \cdot a_{ij}$
$(A + B) + C = A + (B + C)$
$A + B = B + A$
$c(dA) = (cd)A$
$c(A + B) = (cA) + (cB)$
$(AB)C = A(BC)$
$A(B + C) = (AB) + (AC)$
$(A + B)C = (AC) + (BC)$

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



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

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

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

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

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

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

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

	$\neg$
F	T
T	F

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

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

XOR	F	T
F	F	T
T	T	F

F = false, T = true

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

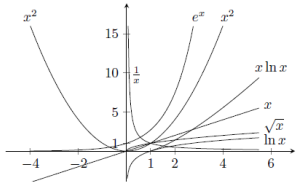
Master Theorem (1)
Constants $a, c, d > 0$, $p \geq 0$ and $b > 1$

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

$$\Downarrow$$

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

Growth of functions



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

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

Læringsmål fra kursusbeskrivelsen

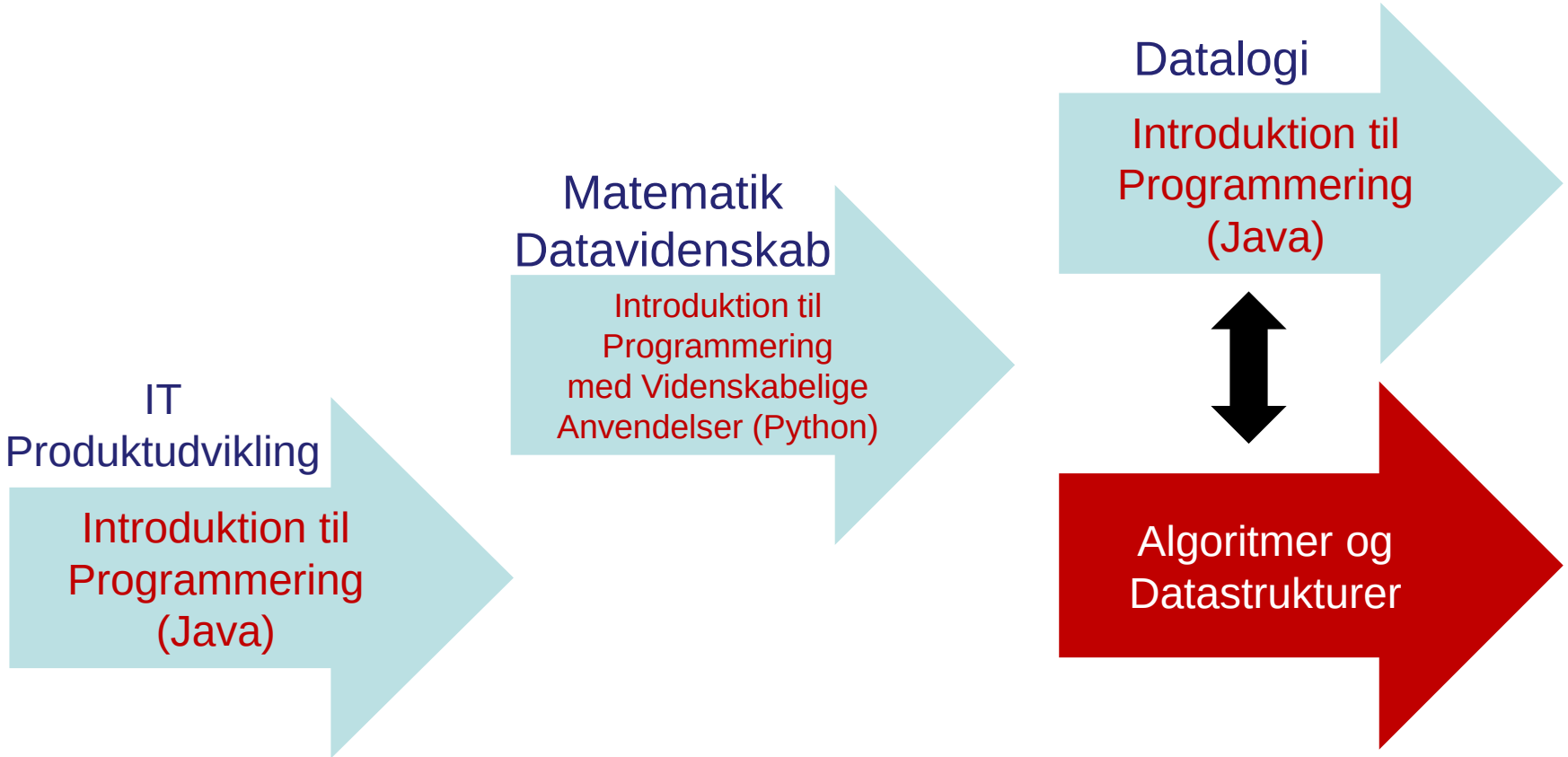
I slutningen af kurset vil deltagerne kunne

- **Formulere** og **udføre** algoritmer og datastrukturer i form af pseudokode
- **Konstruere**, **implementere** og **analysere** algoritmer ved hjælp af standard algoritme paradigmer
- **Identificere** og **sammenligne** datastrukturer og grafalgoritmer til løsning af algoritmiske problemer
- **Identificere** gyldige invarianter for en algoritme
- **Konstruere**, **implementere** og **evaluere** algoritmers ydeevne for simple algoritmiske problemer
- **Analysere** og **sammenligninge** tid og pladsforbrug af algoritmer og datastrukturer
- **Bevise** korrektheden af enkle programmer og transitionssystemer

Spørgsmål ?

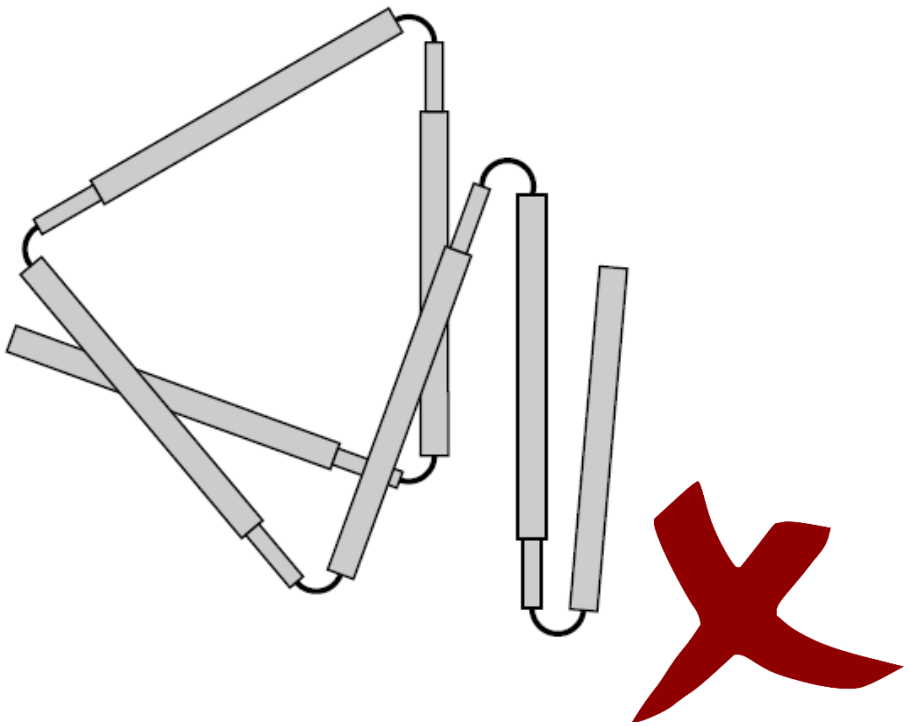
Se Brightspace for info, samt slides!

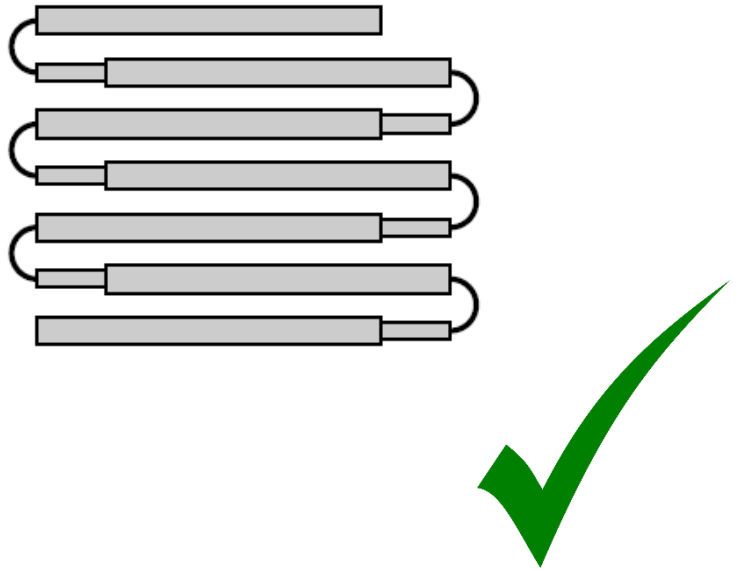
- Algoritmer og Datastrukturer omhandler effektivitet af programmer
- Kræver egentlig man kan programmere...



- Programmeringsafleveringerne vil være basal Java

Teltstangsproblemet



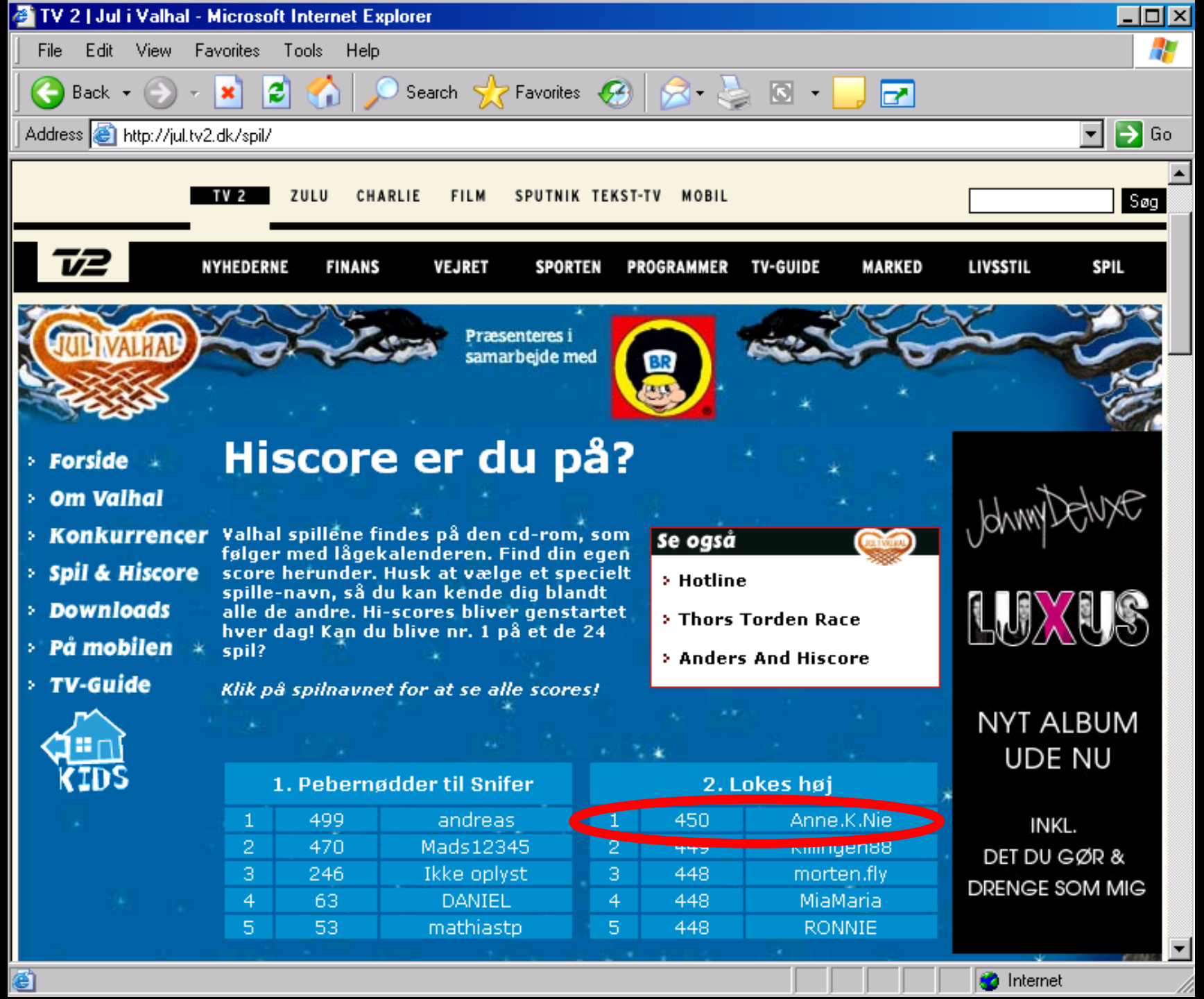


Algoritmer og Datastrukturer

Puzzle, SelectionSort

2005


TV2
500

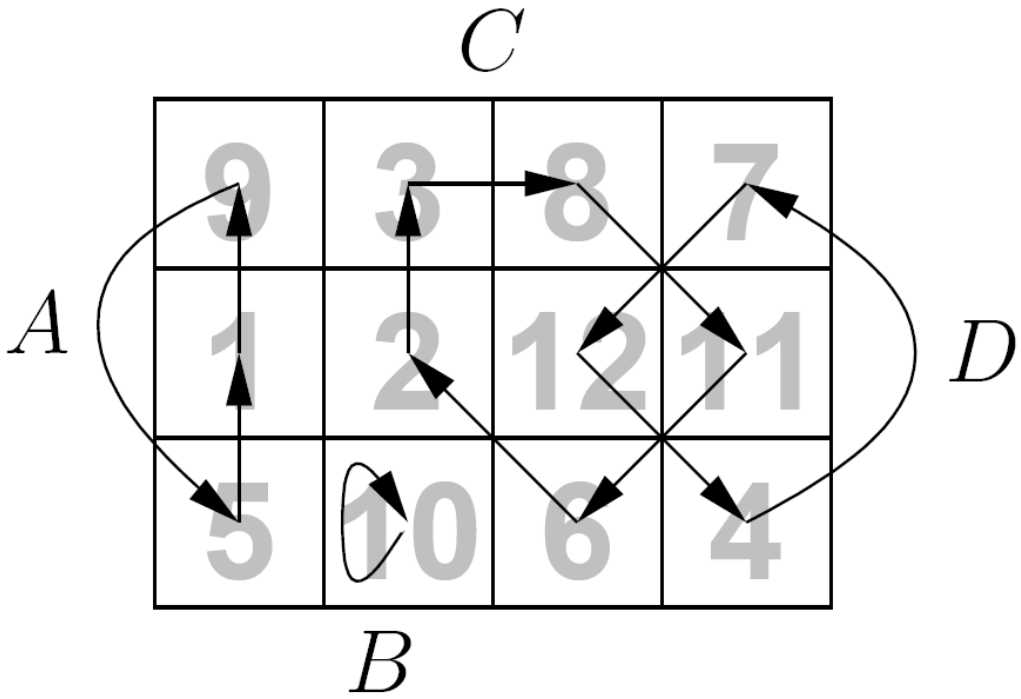

TV 2 | Jul i Valhal - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://jul.tv2.dk/spil/
Go
TV 2 ZULU CHARLIE FILM SPUTNIK TEKST-TV MOBIL
Søg
TV2
NYHEDERNE FINANS VEJRET SPORTEN PROGRAMMER TV-GUIDE MARKED LIVSSTIL SPIL
JUL I VALHAL
Præsenteres i samarbejde med
BR
Forside
Om Valhal
Konkurrencer
Spil & Hiscore
Downloads
På mobilen
TV-Guide
KIDS
Hiscore er du på?
Valhal spillene findes på den cd-rom, som følger med lågekalenderen. Find din egen score herunder. Husk at vælge et specielt spille-navn, så du kan kende dig blandt alle de andre. Hi-scores bliver genstartet hver dag! Kan du blive nr. 1 på et de 24 spil?
Klik på spilnavnet for at se alle scores!
Se også
Hotline
Thors Torden Race
Anders And Hiscore
1. Pebernødder til Snifer
1 499 andreas
2 470 Mads12345
3 246 Ikke oplyst
4 63 DANIEL
5 53 mathiastp
2. Lokes høj
1 450 Anne.K.Nie
2 449 Killingen88
3 448 morten.fly
4 448 MiaMaria
5 448 RONNIE
JohnnyDeluxe
LUXUS
NYT ALBUM UDE NU
INKL.
DET DU GØR &
DRENGE SOM MIG
Internet

”Lokes Høj”

- 64 brikker
- Hiscore 450
- Antal ombytninger 500 - 450 = 50

Hvordan opnår man et lavt antal ombytninger
– held eller dygtighed ?

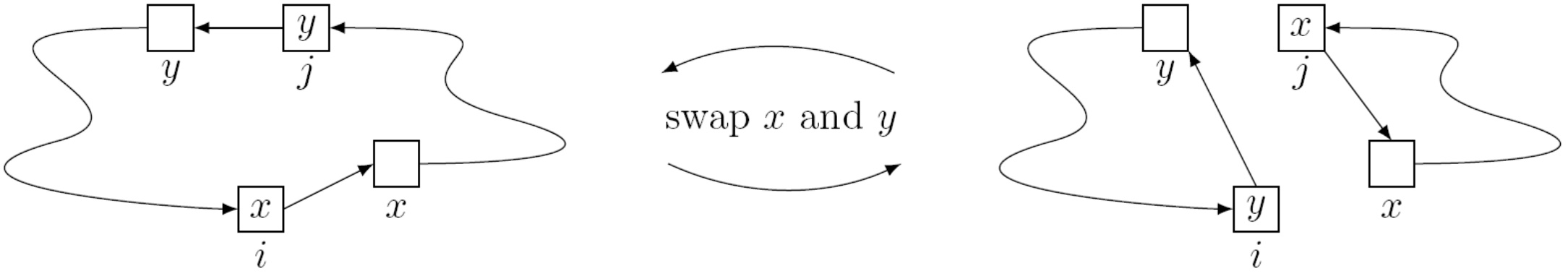
Cykler (Permutationer)



Hver pil peger på brikkens korrekte plads

Definerer en mængde af cykler (fx cyklerne A,B,C,D)

Ombytninger og Cykler



Lemma

- En ombytning af to brikker i **samme cykel** øger antallet af cykler med én.
- En ombytning af to brikker fra to **forskellige cykler** reducerer antallet af cykler med én.

Lemma
Når alle ***n*** brikker er korrekt placeret er der præcis ***n*** cykler.

Lemma
For at løse et puslespil med ***n*** brikker og ***k*** cykler i starten kræves **≥ *n* – *k*** ombytninger.

Har vist en nedre grænse for

ALLE algoritmer der løser problemet

En (grådig) algoritme

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
Algoritmen bytter aldrig om på brikker der står korrekt.

Lemma
Algoritmen udfører $\leq n - 1$ ombytninger

Lemma
For at løse et puslespil med n brikker og k cykler i starten udfører algoritmen præcis $n - k$ ombytninger.

Har vist en øvre grænse for en konkret algoritme

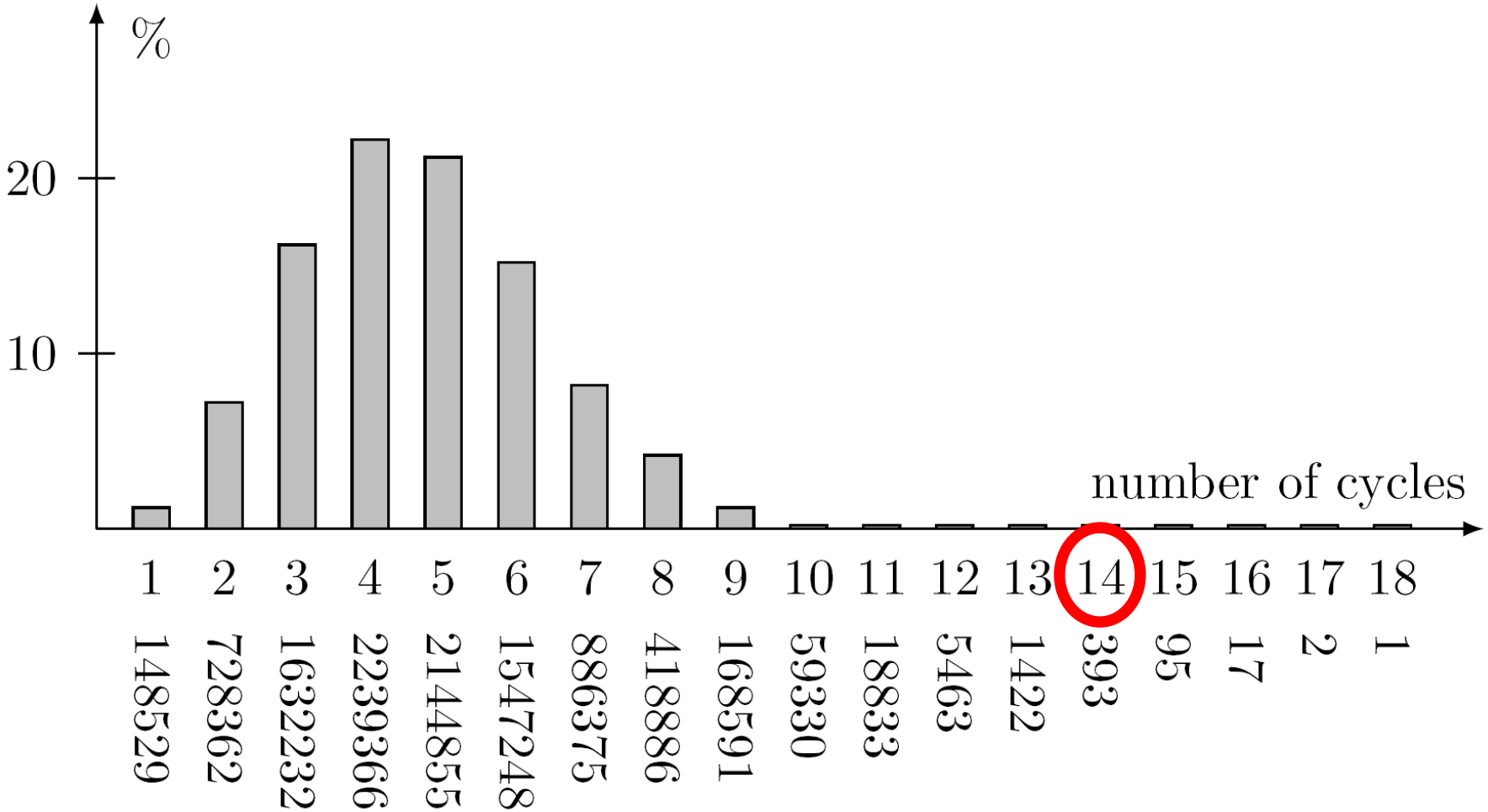
Algoritmen er optimal da antal ombytninger er bedst mulig (de viste nedre og øvre grænser er identiske)

Sætning

For at løse et puslespil med n brikker og k cykler i starten kræves præcis $n - k$ ombytninger

Fordelingen af antal cykler

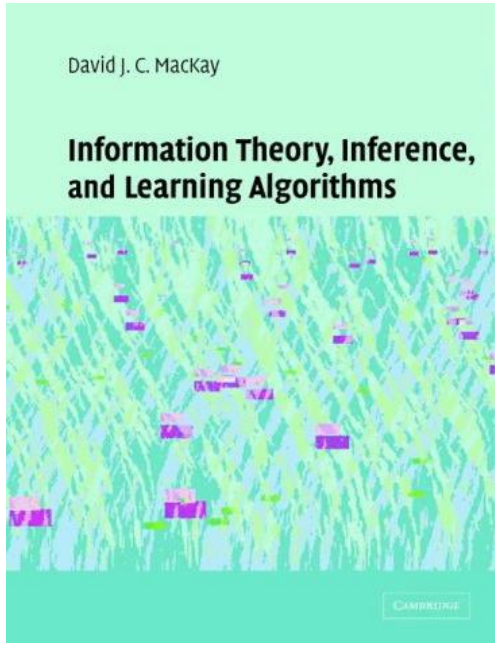
n = 64, 10.000.000 permutationer



Algoritmisk indsigt…

- **Matematisk indsigt** (cykler)
- **Resourceforbrug** (antal ombytninger)
- **Nedre grænse** (≥ *n* - *k* ombytninger)
- **Grådig algoritme**
- **Analyseret algoritmen** (≤ *n* - *k* ombytninger)
- **Optimal algoritme** (argumenteret bedst mulig)
- **Input afhængig resourceforbrug**

Tilfældige permutationer…



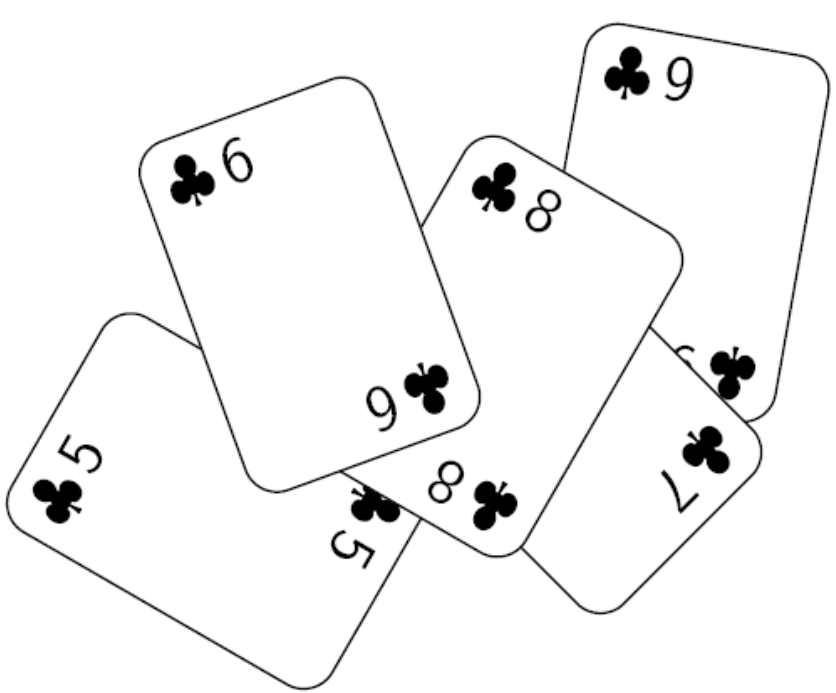
Yderligere information kan findes i David J.C. MacKay, tillæg til *Information Theory, Inference, and Learning Algorithms*, om

"Random Permutations“, 4 sider.

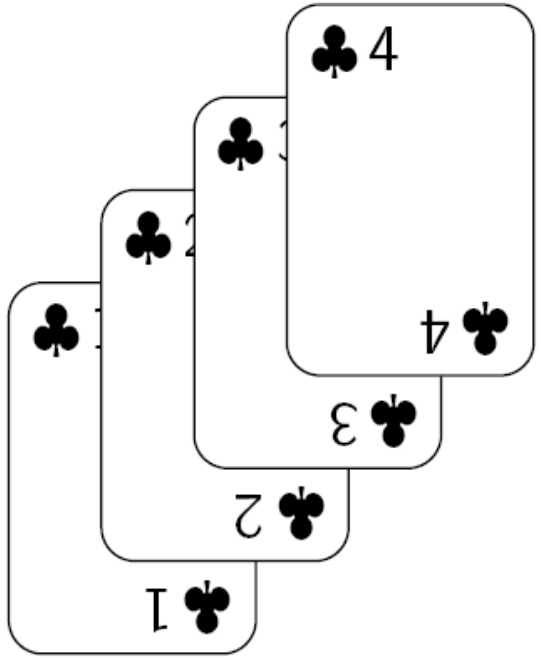
http://www.inference.phy.cam.ac.uk/mackay/itila/cycles.pdf

SelectionSort

Flyt mindste kort



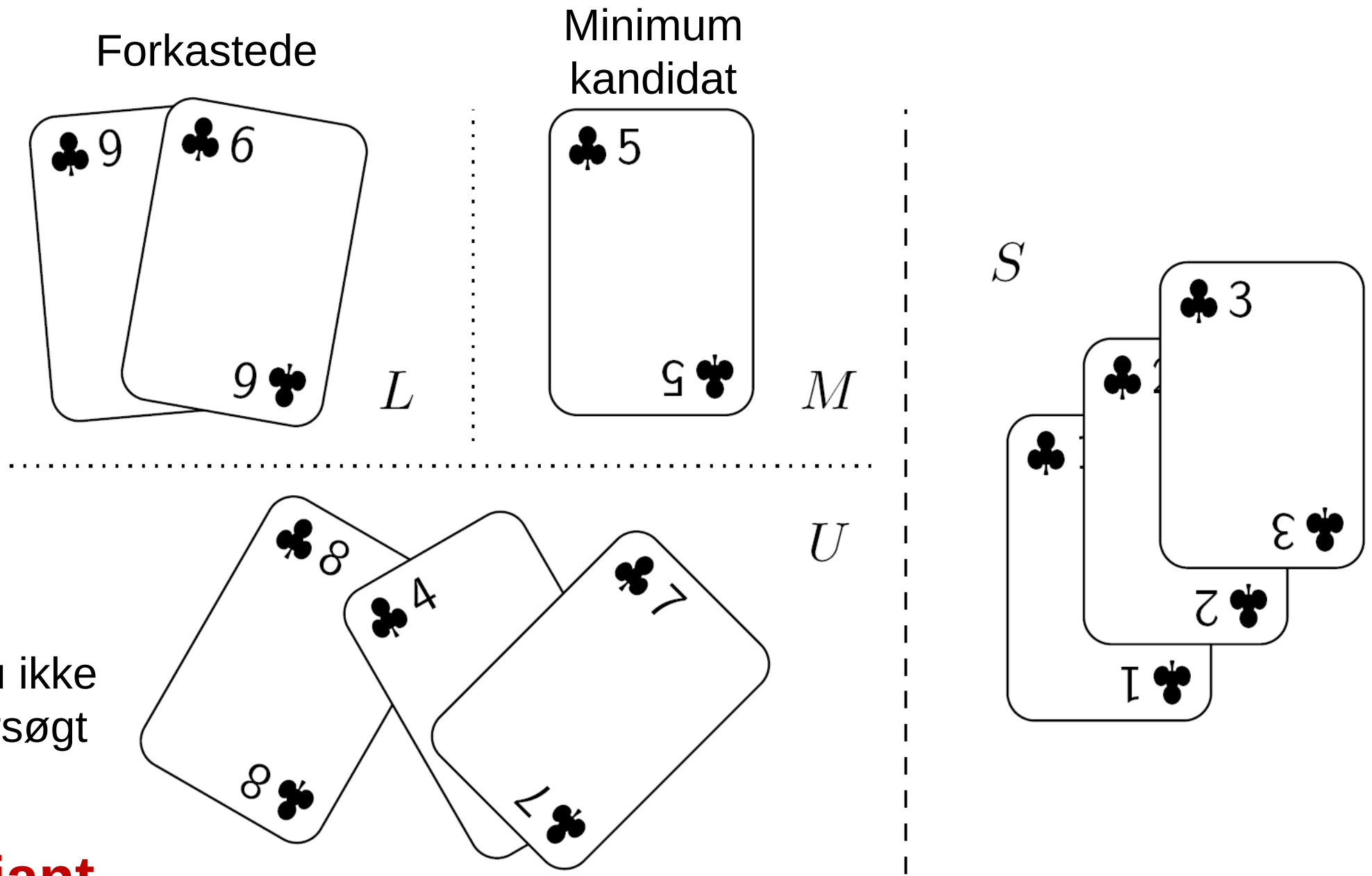
C S



Usorteret

Sorteret

Invariant S sorteret og $C \geq S$



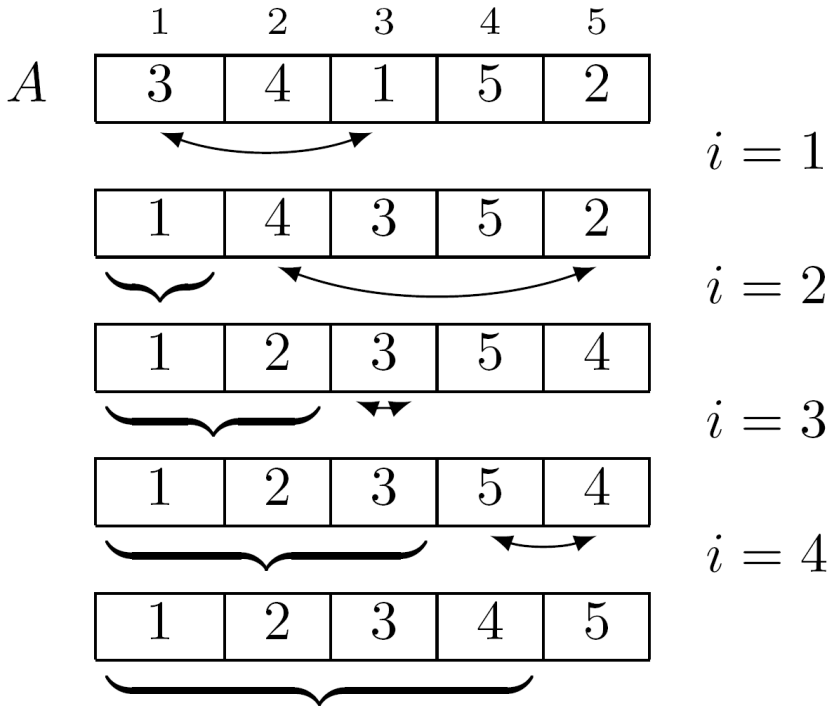
Invariant

S sorteret, $U \cup L \cup M \geq S$, $L \geq M$, $|M| \leq 1$, og $|L| \geq 1 \Rightarrow |M| = 1$

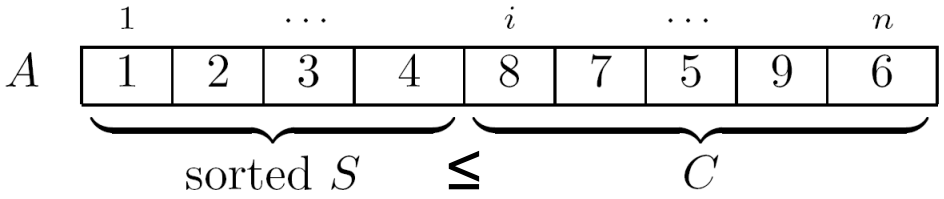
Implicit SelectionSort

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)

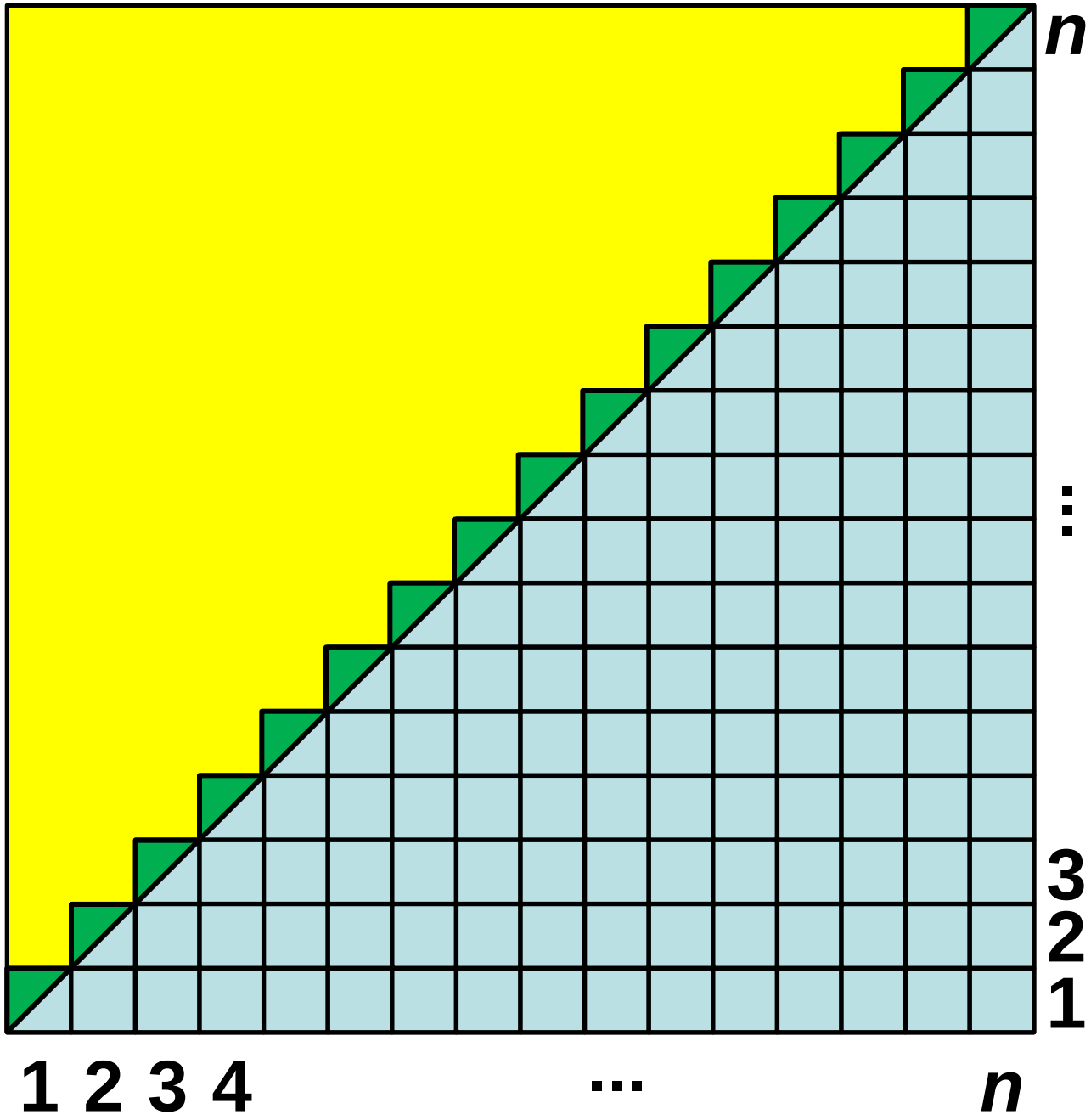
```
1  for i = 1 to |A| − 1 do
2      k = i
3      for j = i + 1 to |A| do
4          # A[k] = min A[i..j − 1]
5          if A[j] < A[k] then
6              k = j
7      # A[k] = min A[i..|A|]
8      tmp = A[i]
9      A[i] = A[k]
10     A[k] = tmp
```

Analyse SelectionSort

Lemma
At finde det mindste kort blandt k kort kræver $k - 1$ sammenligninger

Lemma
SelectionSort på n kort laver $(n - 1) + (n - 2) + \cdots + 1 = n \cdot (n - 1) / 2$ sammenligninger

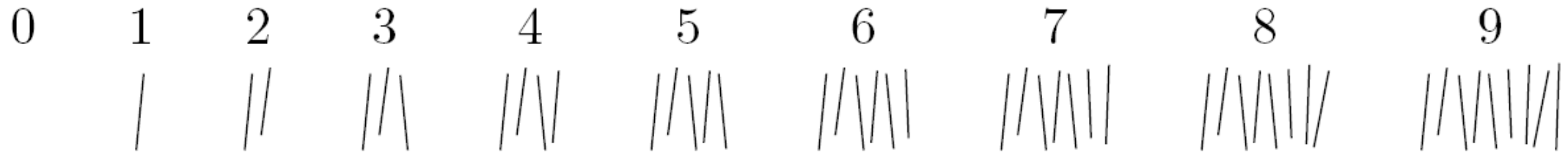
$$1 + 2 + \cdots + n$$
$$= n^2/2 + n/2$$
$$= n(n + 1)/2$$



Algoritmer og Datastrukturer

Heltalsaritmetik: Binære og decimal tal, addition, subtraktion, multiplikation, division

Cifre



For hvert lille tal har man et specielt symbol

10-tals systemet (10 = 9 + 1)

Værdien af $d_{n-1}d_{n-2}\cdots d_1 d_0$

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

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

Repræsentation base *b*

Værdien af $d_{n-1}d_{n-2}\cdots d_1d_0$

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

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

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

$1\cdot10^3+8\cdot10^2+4\cdot10^1+9\cdot10^0$ $\quad 2^{10}+2^9+2^8+2^5+2^4+2^3+2^0$ $\quad 3\cdot8^3+4\cdot8^2+7\cdot8^1+1\cdot8^0$ $\quad 7\cdot16^2+3\cdot16^1+9\cdot16^0$

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

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 - skolemetoden

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

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

Skal (rekursivt) addere flercifrede tal

Binær addition

$+$	0	1
0	0	1
1	1	10

$$
\begin{array}{rccccc}
 & {\scriptstyle 1} & {\scriptstyle 1} & {\scriptstyle 1} & & \\
 & 1 & 1 & 1 & 0 & _2 \\
+ & 1 & 0 & 1 & 1 & _2 \\
\hline
 & 1 & 1 & 0 & 0 & 1 \ _2
\end{array}
$$

Skolemetoden

Addition af flere tal

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

Generaliseret skolemetode

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

...menterne kan have flere cifre

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

Gentagen addition af to tal

Subtraktion – skolemetoden

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

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

4 – 6 = -2 = -10 + 8

Multiplikation

·	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

Multiplikation

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

Multiplikation med enkelt ciffer

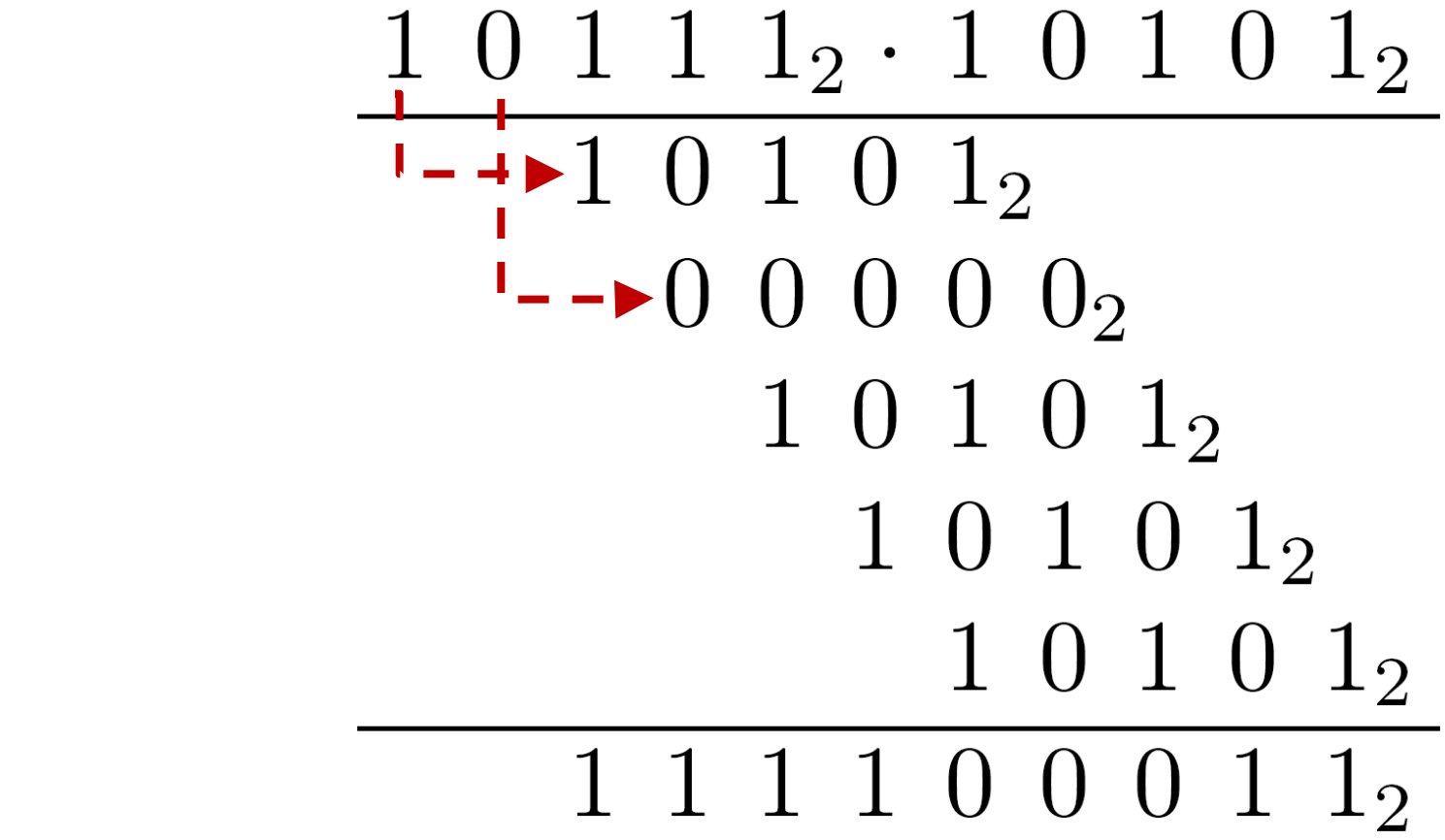
$$\begin{array}{r} 365 \cdot 427 \\ \hline 1281 \\ 2562 \\ 2135 \\ \hline 155855 \end{array}$$

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

Binær multiplikation

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

Binær multiplikation

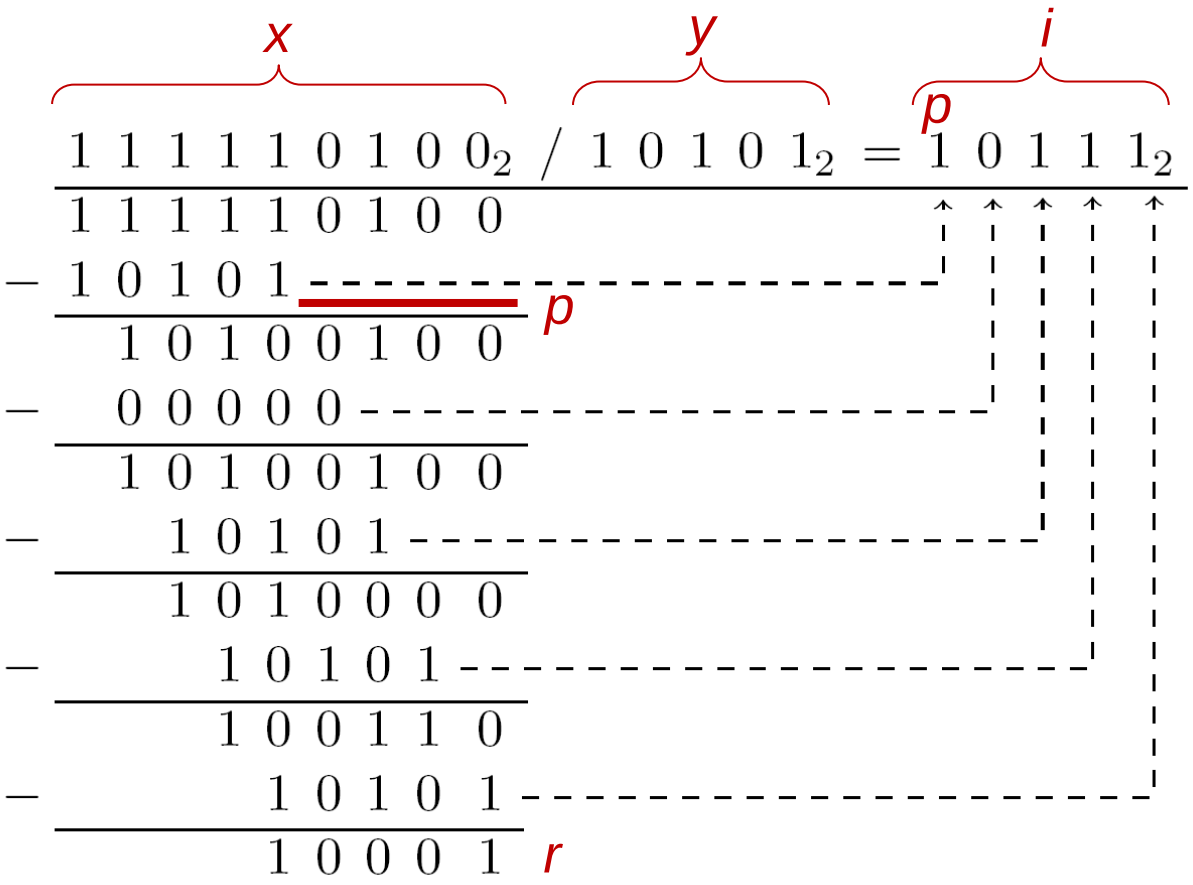


Binær multiplikation af blokke med 1

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

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

Division



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

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

Konvertering til base *b*

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

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

INTEGERDIVISION(x, b)

$x = 42 \quad b = 2$

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

Ikke pensum

Redundante talsystemer

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

- Normalt antages $0 \leq d_i < b$ $\Rightarrow$ entydig repræsentation
- Redundante talsystemer tillader f.eks. også cifre …, -2, -1, b, b+1, …

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

$\Rightarrow$ *ikke* entydig

21002_2 = $2*2^4+1*2^3+2*2^0$

$22\text{-}1\text{-}10_2$ = $2*2^4+2*2^3+-1*2^2+-1*2^1$

- Fordel undgår kaskader af menter under addition
- Anvendes i hardware og mange algoritmer og data strukturer

Se f.eks. Amr Elmasry, Jyrki Katajainen, *Regular numeral systems for data structures*, Acta Informatica 59 (2022), Doi 10.1007/s00236-021-00407-9

Algoritmer og Datastrukturer

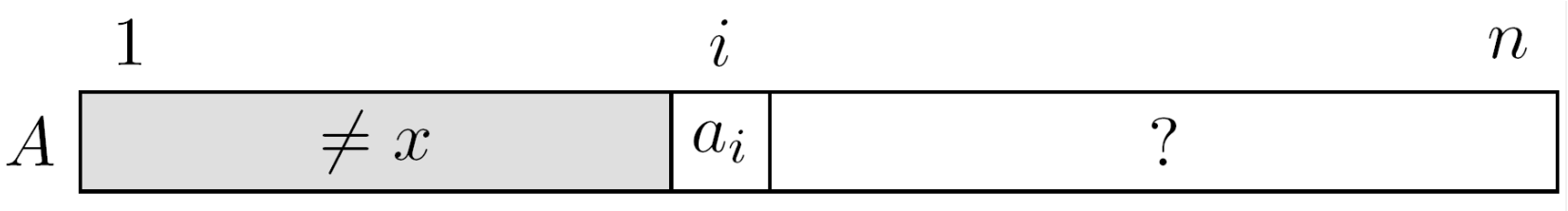
Binær og lineær søgning, logaritmer,
længste voksende delsekvens, Erdös Szekeres sætning

Søgning i usorteret liste

24	11	3	7	32	47	5	2	89	12

Find x

Visuel invariant



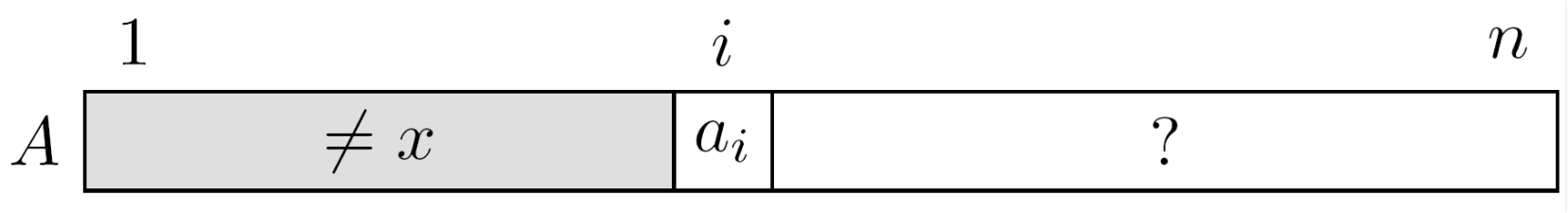
Formel invariant

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

højst *n* sammenligninger

Søgning i usorteret liste

Visuel invariant



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

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

Søgning i sorteret liste

3	7	9	11	13	27	33	37	42	89

Find x

Visuel invariant

Formel invariant

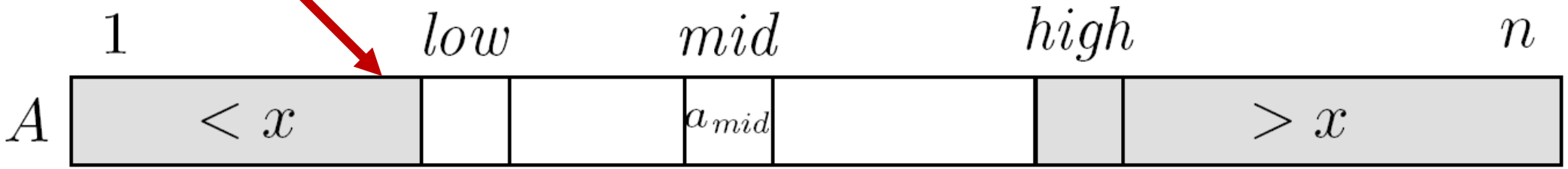
$$(1 \leq low \leq high \leq n+1) \ \wedge$$
$$(a_j < x \text{ for all } 1 \leq j < low) \ \wedge$$
$$(x < a_j \text{ for all } high \leq j \leq n)$$

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

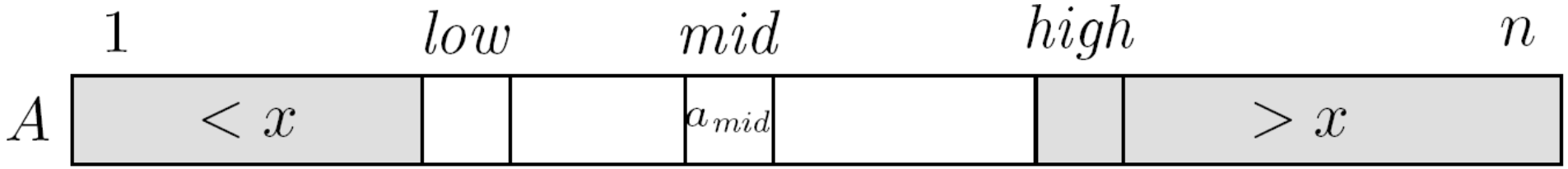
```
1   low = 1
2   high = n + 1
3   while low < high do
4       mid = ⌊(low + high)/2⌋
5       if x = A[mid] then
6           return mid
7       else if x < A[mid] then
8           high = mid
9       else if x > A[mid] then
10          low = mid + 1
11  return −1
```

Opgave
Hvad sker der når man runder op her?

Opgave
Modificer algoritmen til *low* peger her



Binær søgning – antal sammenligninger



Antal kandidater efter én sammenligning ≤ : $n \to \lfloor n/2 \rfloor$

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

Efter k sammenligninger ≤ $n/2^k$ kandidater

garanteret færdig når $n/2^k < 1 \Leftrightarrow n < 2^k \Leftrightarrow \log_2 n < k \Leftrightarrow 1 + \lfloor \log_2 n \rfloor \leq k$

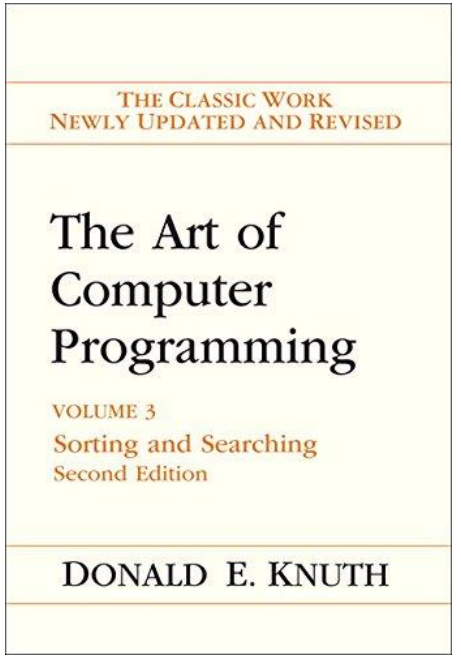
højst $1 + \lfloor \log_2 n \rfloor$ sammenligninger

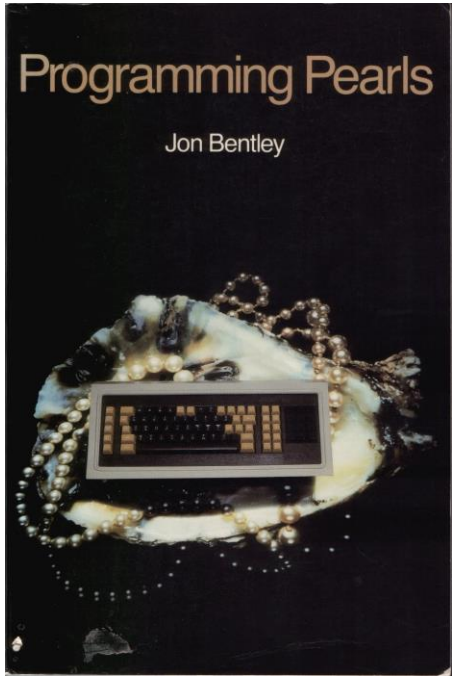
Binær søgning i litteraturen

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

- Bentley's erfaring fra Bell Labs og IBM i starten af 80'erne, hvor han bad folk implementere binær søgning:

  …*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, Kapitel 6.2.1, 1973
Jon Bentley, Programming Pearls, Kapitel 4.1, 1986

43. *Program for Binary Search*

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

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

H. Bottenbruch, Structure and Use of ALGOL 60, *Journal of the ACM*, 9 (1962), 214, DOI: 10.1145/321119.321120

Binære søgning i standard biblioteker

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

Søgning i sorteret liste – nedre grænse

3	7	9	11	13	27	33	37	42	89

For enhver algoritme kan man svare således at antallet af kandidater mindst er

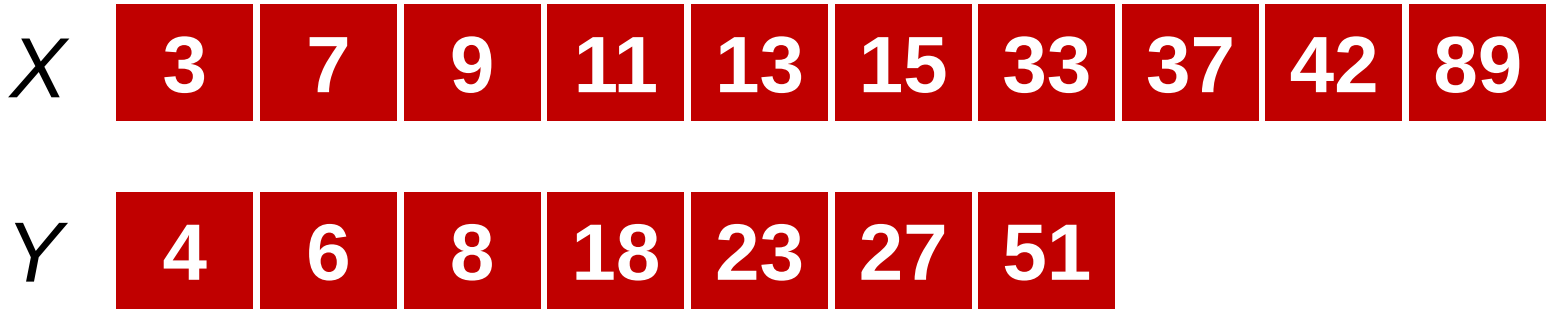
$$n \to \lfloor n/2 \rfloor \to \lfloor \lfloor n/2 \rfloor /2 \rfloor \to \lfloor \lfloor \lfloor n/2 \rfloor /2 \rfloor /2 \rfloor \to \cdots \to 0$$

$n \geq 2^{\lfloor \log_2 n \rfloor} \Rightarrow$ efter k sammenligninger $\geq 2^{\lfloor \log_2 n \rfloor - k}$ kandidater

$\Rightarrow$ efter $\lfloor \log_2 n \rfloor$ sammenligninger mindst 1 kandidat tilbage

mindst 1 + $\lfloor \log_2 n \rfloor$ sammenligninger

Problem 1.9

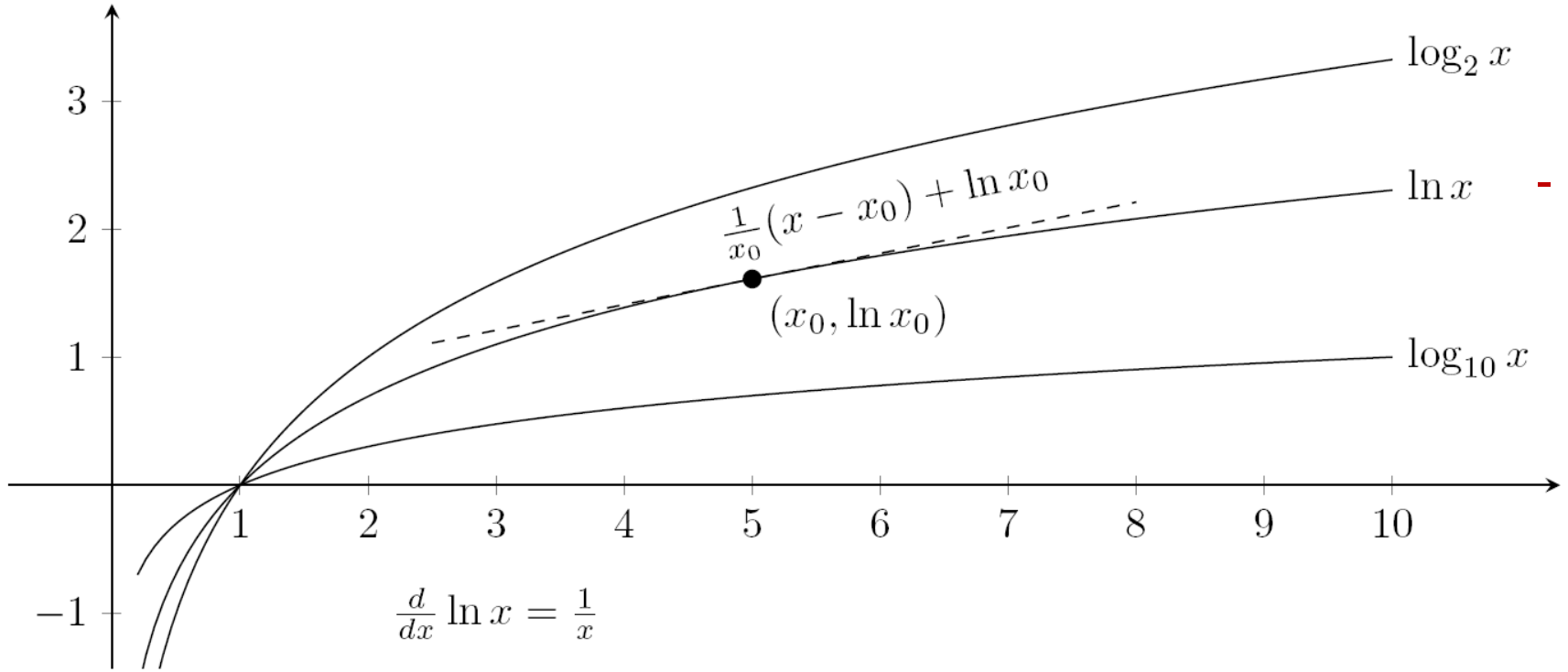


Givet X og Y sorteret, find det r-te mindste i $X \cup Y$

(det 9. mindste element er 15)

Logaritmer

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



- naturlige logaritme

Logaritmer – regneregler

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

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

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

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

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

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

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

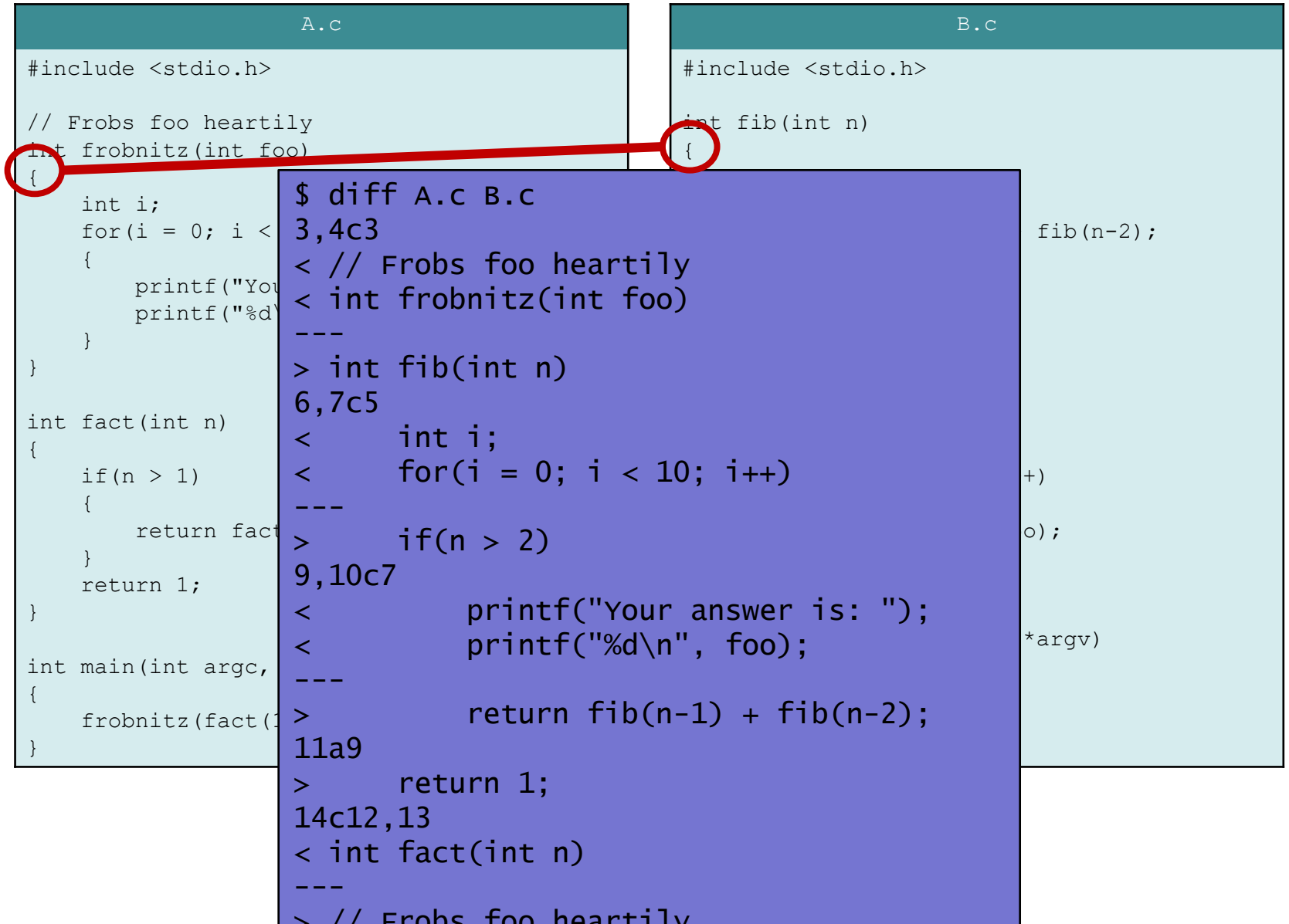
$$H_n - \ln n \to \gamma \quad \text{for} \quad n \to \infty$$

Harmoniske tal $H_n = \sum_{i=1}^{n} \frac{1}{i}$

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

Et algoritmisk eksempel (der anvender binær søgning)

Patience Diff
&
Længste Voksende Delsekvenser


A.c
#include <stdio.h>
// Frobs foo heartily
int frobnitz(int foo)
{
int i;
for(i = 0; i <
{
printf("You
printf("%d\
}
}
int fact(int n)
{
if(n > 1)
{
return fact
}
return 1;
}
int main(int argc,
{
frobnitz(fact(1
}
B.c
#include <stdio.h>
int fib(int n)
{
fib(n-2);
+)
o);
*argv)
$ diff A.c B.c
3,4c3
< // Frobs foo heartily
< int frobnitz(int foo)

> int fib(int n)
6,7c5
< int i;
< for(i = 0; i < 10; i++)

> if(n > 2)
9,10c7
< printf("Your answer is: ");
< printf("%d\n", foo);

> return fib(n-1) + fib(n-2);
11a9
> return 1;
14c12,13
< int fact(int n)

> // Frobs foo heartily

11 removals 10 additions

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

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

Text Compare!

Email this comparison

```
#include <stdio.h>

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

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

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

```
#include <stdio.h>

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

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

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

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

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

File Left Right View Options Mergely Social

```
#include <stdio.h>

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

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

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

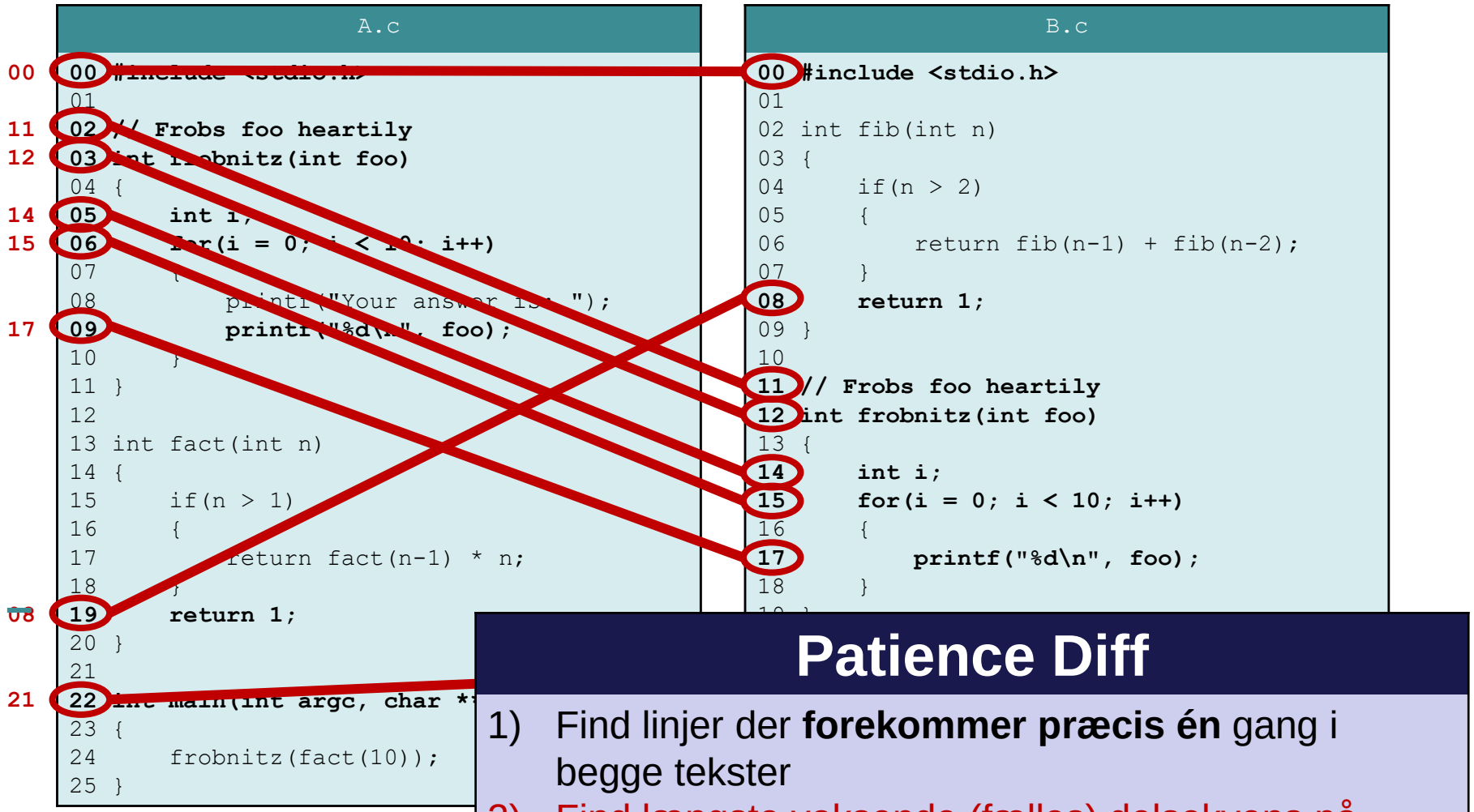
```
#include <stdio.h>

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

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

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

Patient Diff
(Bram Cohen, opfinder af BitTorrent)

- Forsøger at lave **læsbar og meningsfuldt output** – frem for mindst mulig

- Anvendes i Bazaar versionskontrolsystemet (bazaar-vcs.org)

Mindst mulig løsning findes senere i kurset vha. Dynamisk Programmering


A.c
B.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 return 1;
20 }
21
21 int main(int argc, char **
23 {
24 frobnitz(fact(10));
25 }
00 #include <stdio.h>
01
02 int fib(int n)
03 {
04 if(n > 2)
05 {
06 return fib(n-1) + fib(n-2);
07 }
08 return 1;
09 }
10
11 // Frobs foo heartily
12 int frobnitz(int foo)
13 {
14 int i;
15 for(i = 0; i < 10; i++)
16 {
17 printf("%d\n", foo);
18 }
Patience Diff
1) Find linjer der forekommer præcis én gang i begge tekster
2) Find længste voksende (fælles) delsekvens på disse
3) Gentag (rekursivt) på blokkende

Længste voksende delsekvens
(Problem 1.2)

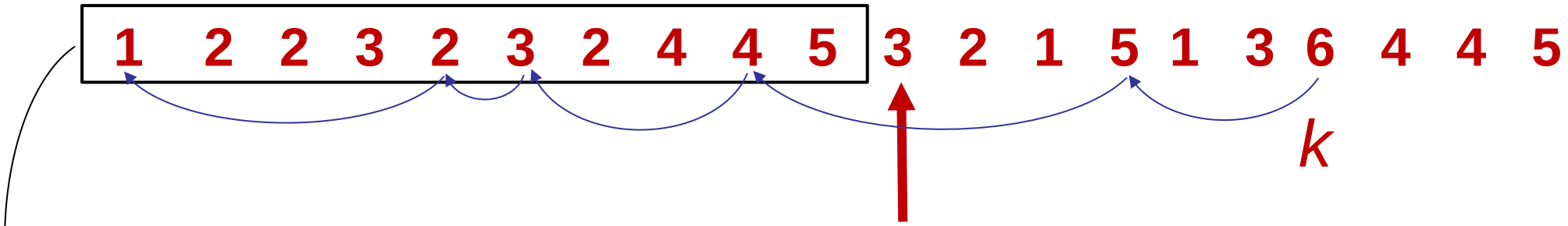
30 83 73 80 59 63 41 78 68 82 53 31 22 74 6 36 99 57 43 60

Opgave

Slet så få tal som muligt fra en liste af tal,
så de resterende tal står i voksende orden

Længste voksende delsekvens
(Problem 1.2)

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



længste voksende delsekvens der slutter på 53

Længde	Mindste sidste element
1	30
2	41
3	~~63~~ 53
4	68
5	82

= binær søgning

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

Sætning (Erdős og Szekeres, 1935)

Enhver sekvens af n tal har en voksende eller aftagende delsekvens af længde mindst $\lceil \sqrt{n} \rceil$.

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

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

Sætning (Erdős og 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

Længde	Mindste sidste 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

Enten mange ($\geq \sqrt{n}$) rækker (lang voksende delsekvens) eller
mindst én lang ($\geq \sqrt{n}$) række (lang aftagende delsekvens)

Algoritmer og Datastrukturer

Induktion og invarianter

Bevis ved induktion

- Vi ønsker at bevise en uendelig række udsagn

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

- F.eks. kan U_n være udsagnet: $\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$

Udsagnene kan **bevises v.h.a. induktionsprincipet**, ved at bevise følgende to udsagn:

1. **Basis**:
   Vis at U_1 gælder
2. **Induktionsskridt**:
   Antag for et $n \geq 1$ at U_n gælder (**induktionshypotese**)

   og vis så at U_{n+1} gælder, dvs. vis at $U_n \Rightarrow U_{n+1}$

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

[CLRS A.1] (A.1)

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

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

  Skal vise at: $\sum_{i=1}^{n+1} i = \frac{(n+1)\big((n+1)+1\big)}{2}$

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

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

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

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

□

Eksempel: Bevis $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

[CLRS A.1] (A.6)

- Basis $n = 0$: $\sum_{i=0}^{0} 2^i = 2^0 = 1 = 2^{0+1} - 1$
- Induktionsskridt:

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

  Skal vise at: $\sum_{i=0}^{n+1} 2^i = 2^{(n+1)+1} - 1$

  Bevis:

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

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

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

[CLRS A.1] (A.6)

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

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

  Skal vise at: $\sum_{i=0}^{n+1} x^i = \frac{x^{(n+1)+1}-1}{x-1}$

  Bevis:

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

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

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

[CLRS A.1] (A.4)

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

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

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

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

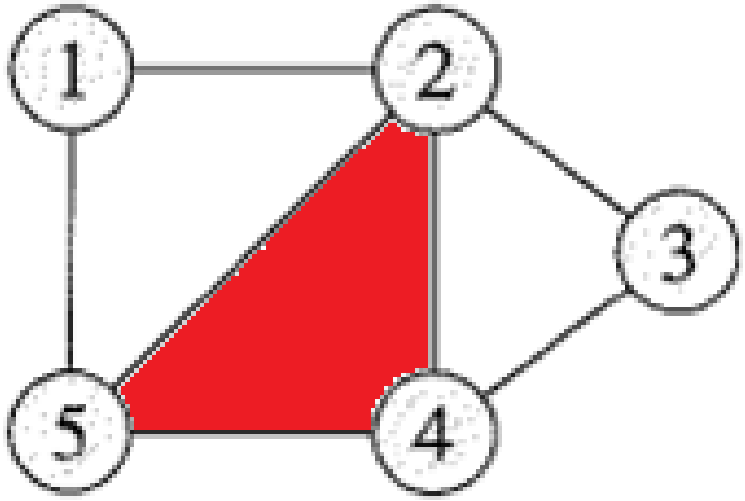
$$\overset{\text{i.h.}}{=} (n+1)^2 + \frac{n(n+1)(2n+1)}{6} = \frac{6(n+1)^2 + n(n+1)(2n+1)}{6}$$

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

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

□

Planar Grafer - Eulers formel



$|V| = 5$ knuder

$|E| = 7$ kanter

flader = 4

For en sammenhængende planar graf gælder:

Eulers formel: $|V| - |E| + \#\text{ flader} = 2$

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

(for $|V| \geq 3$,
ingen selvløkker,
ingen parallelle kanter)

invariant adjektiv

BØJNING -, -e

UDTALE ['enva͡i,anˀd]

OPRINDELSE første led latin *in-* i betydningen 'ikke-, u-'

Betydninger

uforanderlig; konstant – bl.a. inden for matematik

GRAMMATIK almindelig som substantiv fælleskøn

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

Løkke-invariant = udsagn

Eksempler på **I**

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

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

x' og i' er de nye værdier

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

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

$x = 100 + 7 * i \wedge i \leq 11 \wedge x$ og i er heltal

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

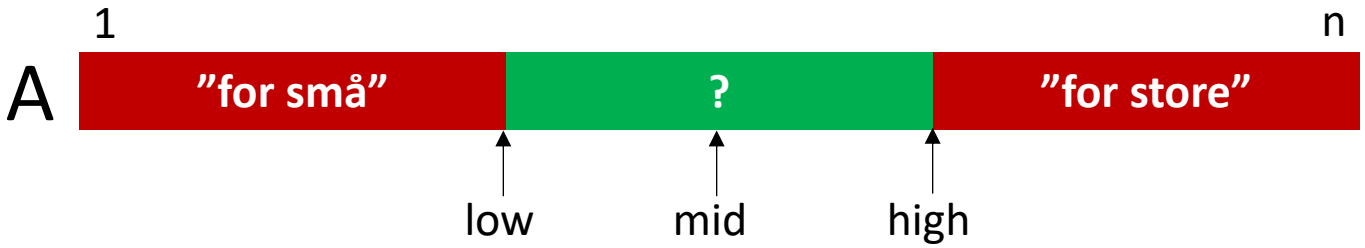
En invariant I skal opfylde

1) Når løkken nås første gang, så er **I** opfyldt
2) Hvis **I** er opfyldt før løkken udføres, så er **I** opfyldt efter en udførsel af løkken

I gælder automatisk når vi kommer ud af løkken

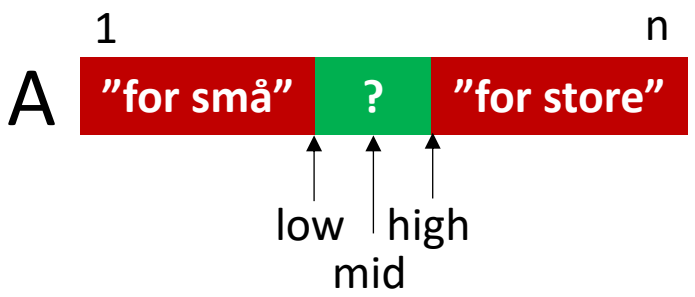
Udnyt **I** og at løkke-betingelsen er forkert når vi er færdig til at drage en konklusion

Binær søgning (i sorteret array)




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

Binær søgning (i sorteret array)




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

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

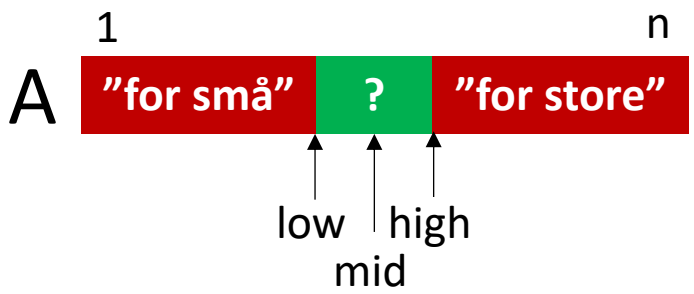
I_2: (A[i] < x for 1 ≤ i ≤ low) ^ (x ≤ A[i] for high ≤ i ≤ n)

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

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

…

Binær søgning (i sorteret array)




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

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

Algoritme POWER(x, p)
Inputbetingelse : Heltal $x \geq 0$ og $p \geq 0$
Outputkrav : $r = x^p$
Metode : $r \leftarrow 1;$
$\{I\}$ **while** $p \geq 1$ **do**
if p lige **then**
$x \leftarrow x * x;\ p \leftarrow p/2$
else
$r \leftarrow r * x;\ p \leftarrow p - 1$

p_0 = værdien af p i starten

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

Eksamensopgave, Algoritmer og Datastrukturer 1, marts 2017, opgave 21

Algoritme LOG2(n)
Inputbetingelse : Heltal $n \geq 2$
Outputkrav : $r = \text{intlog}(n) = \lfloor \log_2 n \rfloor$
Metode : $i \leftarrow 1;$
$r \leftarrow 1;$
$p \leftarrow 2;$
$\{I\}$ **while** $2p \leq n$ **do**
if $p * p \leq n$ **then**
$p \leftarrow p * p;$
$r \leftarrow 2 * r$
else
$p \leftarrow 2 * p;$
$r \leftarrow r + 1$

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

Eksamensopgave, Algoritmer og Datastrukturer 1, marts 2015, opgave 21

Invarianter

- Værktøj til analyse af tilstandene i en løkke i en algoritme
- Designværktøj ”Invariant → kode”
- Kan indfange essentielle egenskaber ved en datastrukturs tilstand

Algoritmer og Datastrukturer

Analyseværktøjer, RAM model,
Insertion-Sort, O-notation
[CLRS, 1-3.2]

Hvad er udførselstiden for en algoritme?

Fra Idé til Programudførelse

Idé

Del og Kombiner ?

Pseudokode

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

(Java-)kode

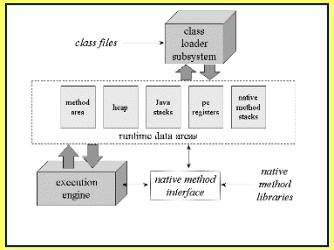
```
...
if ( t1[t1index] <= t2[t2index] )
      a[index] = t1[t1index++];
else
      a[index] = t2[t2index++];
...
```

Compiler

Maskinkode

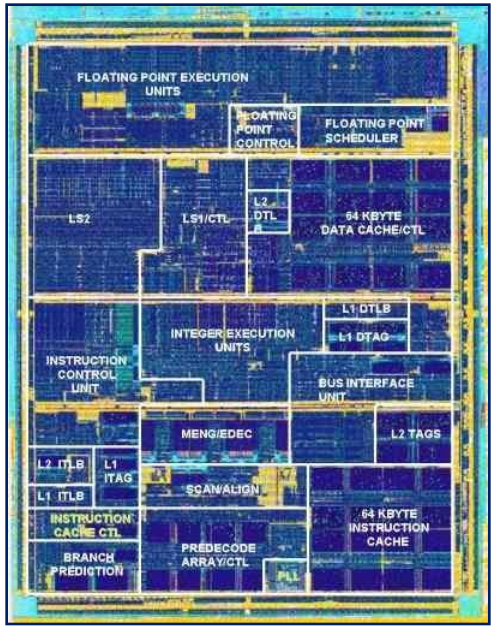
Assembler

(Java) Bytekode



+ Virtuel maskine

Program-udførsel



Mikrokode

Virtuel hukkomelse/ TLB

L1, L2,… cache

Branch Prediction

Pipelining

...

Maskiner har forskellig hastighed...

Maskine	Tid (sek)
camel19	8.9
molotov	10.2
harald	26.2
gorm	7.8

Tid for at sortere linierne i en 65 MB web log på forskellige maskiner på Institut for Datalogi

Idé:

Argumenter om algoritmer uafhængig af maskine

Design af Algoritmer

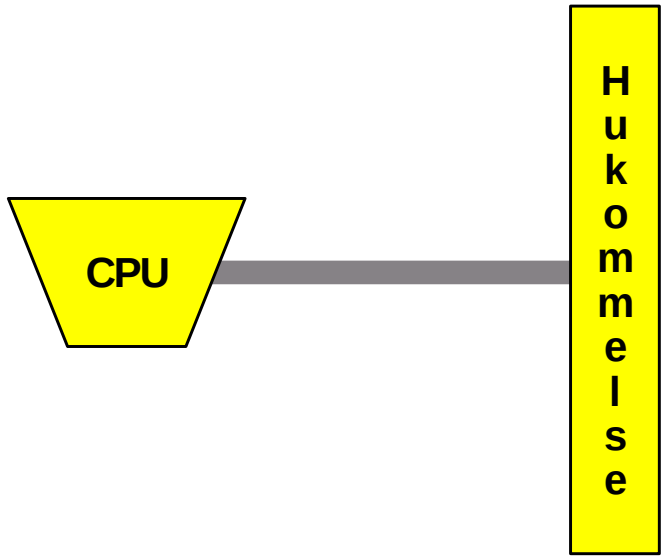
Korrekt algoritme

- algoritmen **standser** på alle input
- output er det **rigtige** på alle input

Effektivitet

- Optimer algoritmerne mod at bruge **minimal** tid, plads, additioner,... eller **maximal** parallellisme...
- $\sim n^2$ er bedre end $\sim n^3$: **asymptotisk tid**
- Mindre vigtig : **konstanterne**
- Resouceforbrug: **Worst-case** eller **gennemsnitlig**?

RAM Modellen
(Random Access Machine)



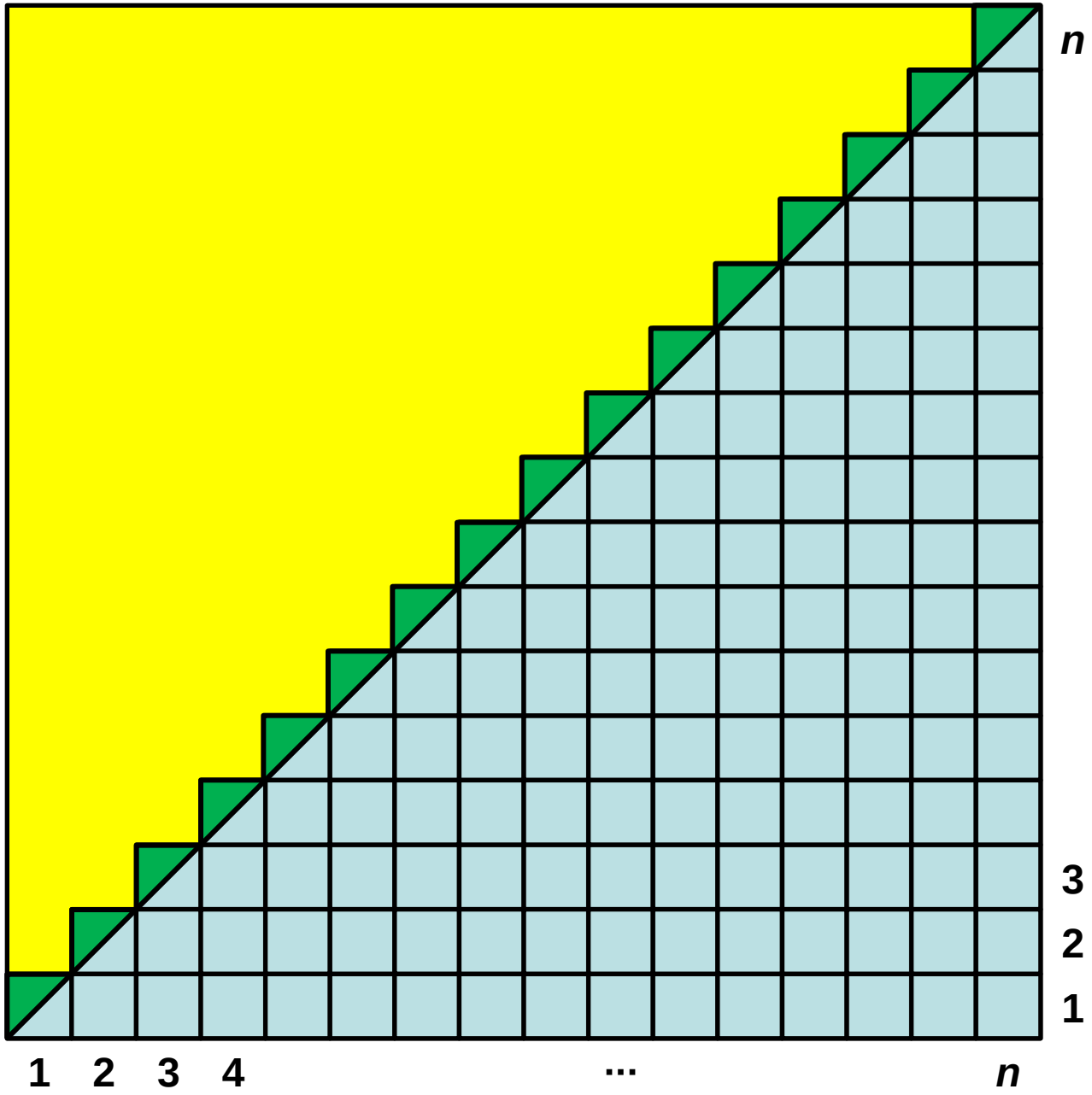
- Beregninger sker i CPU
- Data gemmes i hukommelsen
- Basale operationer tager **1 tidsenhed**:

  +, -, *, AND, OR, XOR, **get(i)**, **set(i,v)**, ...
- Et maskinord indeholder $c \cdot \log n$ **bits**

Eksempel: Insertion-Sort

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

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



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
```

Eksempel: Insertion-Sort

- Eksempel på **pseudo-kode**
- Detaljeret analyse – stort arbejde
- Tid: **worst-case** ($\sim n^2$) og **best-case** ($\sim n$) meget forskellige
- Tid: **gennemsnitlige** ($\sim n^2$)
- Hurtigere på ~ sorterede input: **adaptive**

Asymptotisk notation

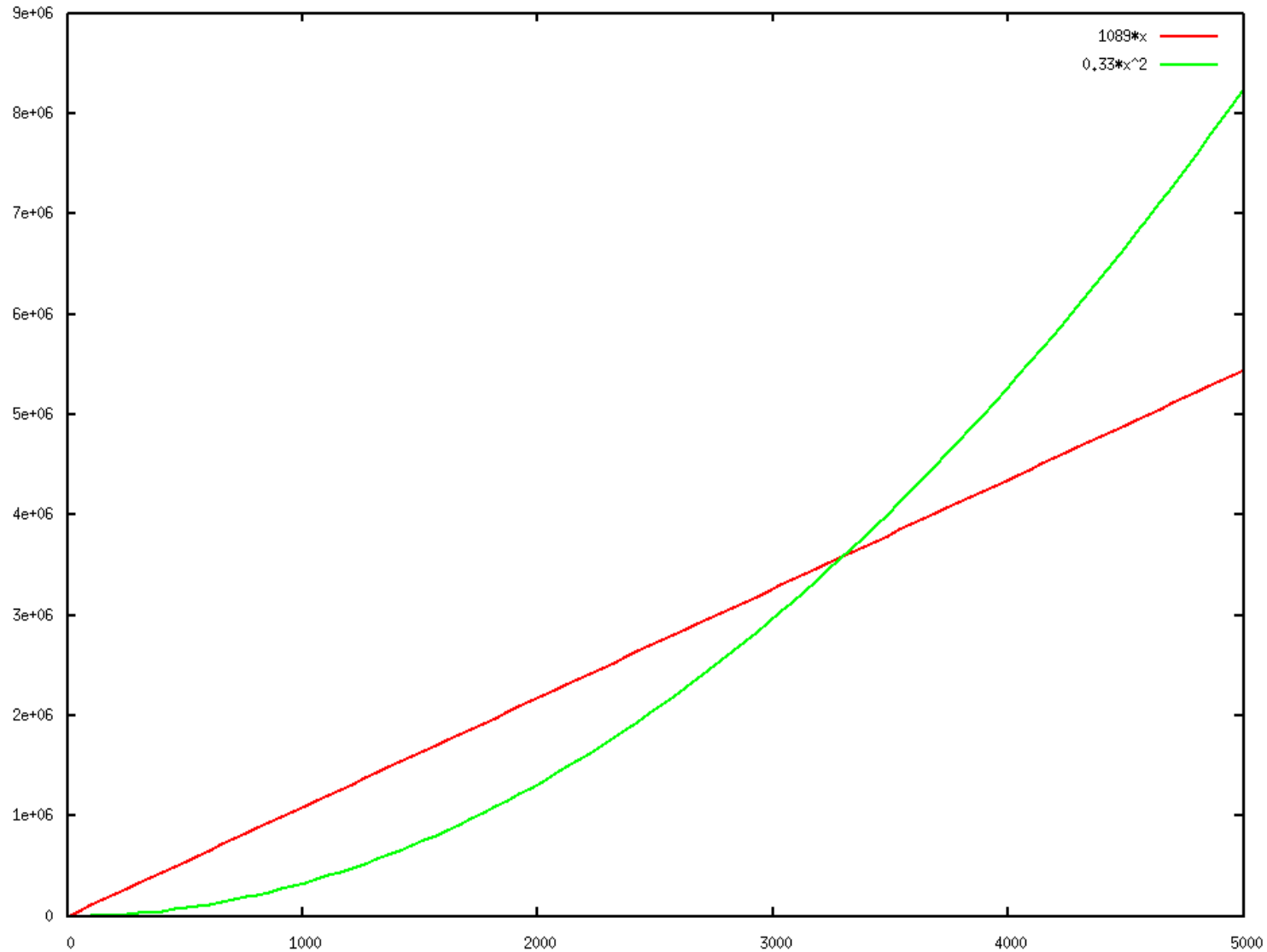
- Grundlæggende antagelse:
  - $\sim n^2$ er bedre end $\sim n^3$
  - Konstanter ikke vigtige
- **Matematisk formel** måde at arbejde med ”~”
- Eksempler:

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

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

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

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



O - notation

... og vennerne

Ω (store omega)
Θ (theta)
ω (lille omega)
o (lille o)

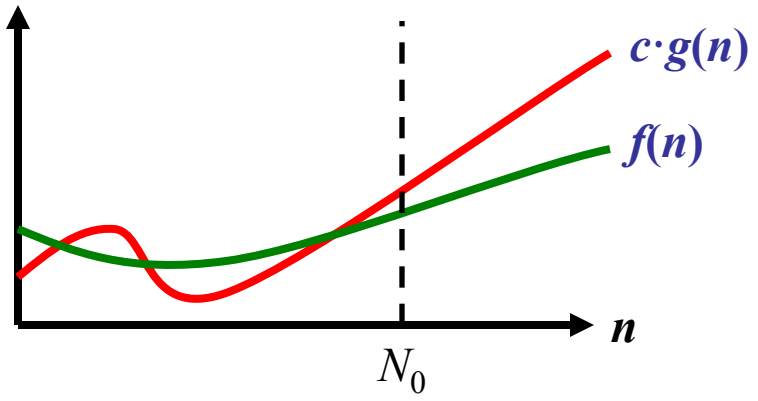
O-notation

Definition: $f(n) = \mathrm{O}(g(n))$

hvis $f(n)$ og $g(n)$ er funktioner $N \rightarrow R$ og

findes $c > 0$ og N_0 så *for alle* $n \geq N_0$:

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



Intuitivt: $f(n)$ er ”mindre end er lig med” $g(n)$, eller $g(n)$ ”dominerer” $f(n)$

O-notation

O($g(n)$) er mængden af funktioner der asymptotisk er begrænset af $g(n)$

Datalogi notation	Matematik notation
$f(n) = O(g(n))$	$f(n) \in O(g(n))$
$O(f(n)) = O(g(n))$	$O(f(n)) \subseteq O(g(n))$
Eksempler: • $n^2 = O(n^3)$ • ~~$O(n^3) = n^2$~~ • $O(n^2) = O(n^3)$ • ~~$O(n^3) = O(n^2)$~~	

Eksempel: Insertion-Sort

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

Tid $O(n^2)$

Eksempler : O - regneregler

$$f(n) = \mathrm{O}(g(n)) \quad \rightarrow \quad c \cdot f(n) = \mathrm{O}(g(n))$$

$$f_1(n) = \mathrm{O}(g_1(n)) \quad \text{og} \quad f_2(n) = \mathrm{O}(g_2(n)) \quad \rightarrow$$

$$f_1(n) + f_2(n) = \mathrm{O}(\max(g_1(n), g_2(n)))$$

$$f_1(n) \cdot f_2(n) = \mathrm{O}(g_1(n) \cdot g_2(n))$$

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

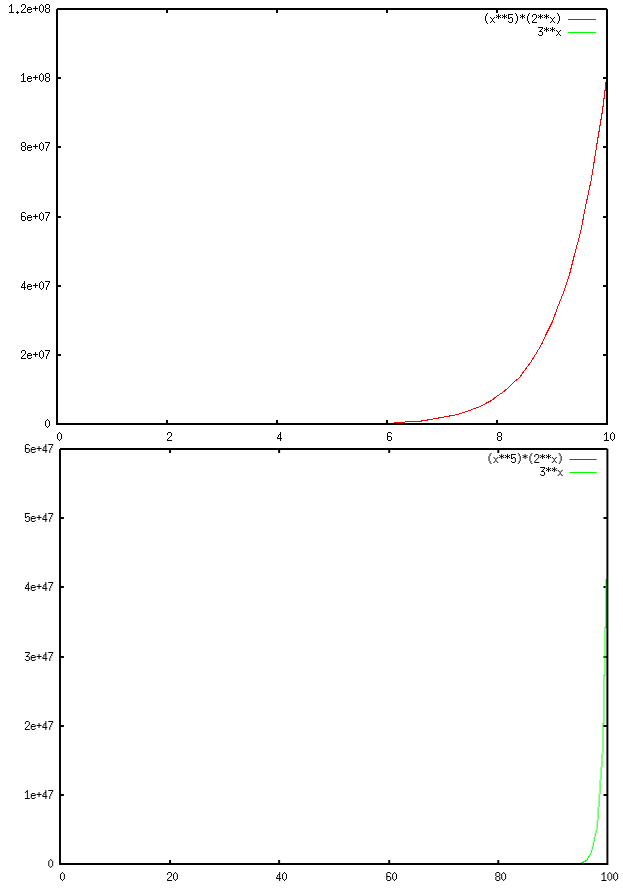
for positive monotone funktioner

Eksempler : O

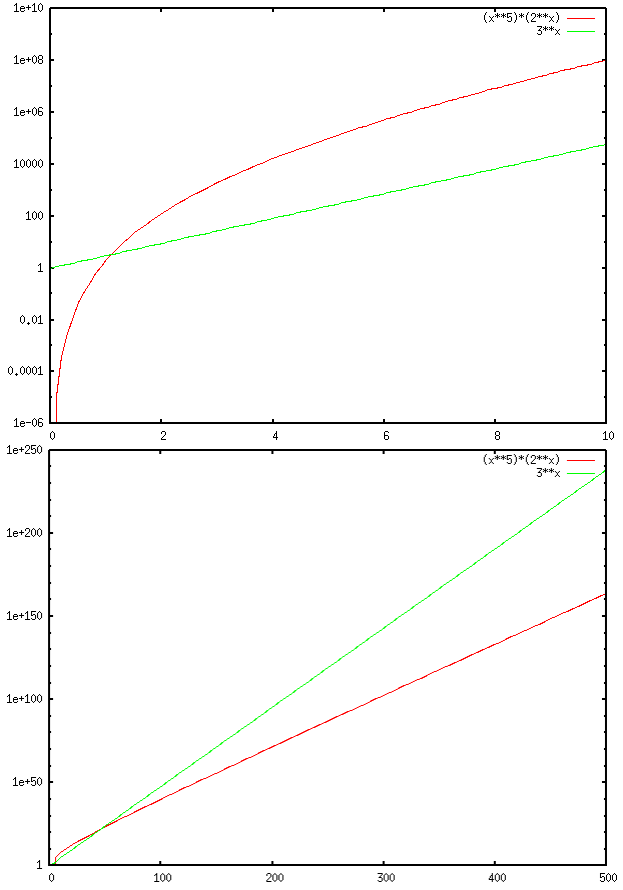
- $3 \cdot n^2 + 7 \cdot n = \mathrm{O}(n^2)$
- $n^2 = \mathrm{O}(n^3)$
- $\log_2 n = \mathrm{O}(n^{0.5})$
- $(\log_2 n)^3 = \mathrm{O}(n^{0.1})$
- $n^2 \cdot \log_2 n + 7 \cdot n^{2.5} = \mathrm{O}(n^{2.5})$
- $2^n = \mathrm{O}(3^n)$
- $n^5 \cdot 2^n = \mathrm{O}(3^n)$

Visuel test af $n^5 \cdot 2^n = \mathrm{O}(3^n)$?

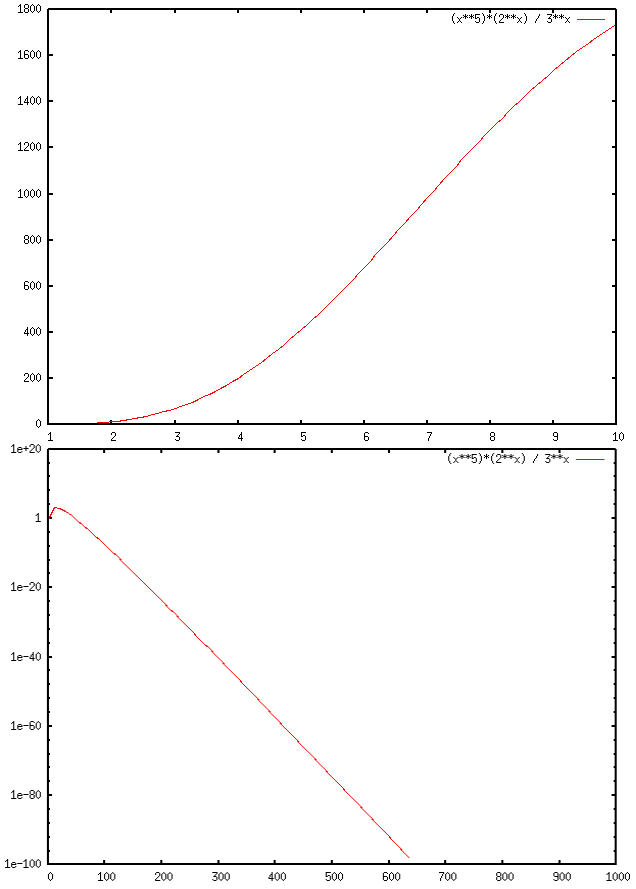
Plot af de to funktioner
– ikke særlig informativ



Plot af de to funktioner
med logaritmisk y-akse
– første plot misvisende



Plot af brøken mellem
de to funktioner
– første plot misvisende



Plots lavet med Gnuplot

Bevis for $n^5 \cdot 2^n = \mathrm{O}(3^n)$

Vis $n^5 \cdot 2^n \leq c \cdot 3^n$ for $n \geq N_0$ for passende valg af c og N_0

Bevis:

$(5/\log_2 (3/2))^2 \leq n$ for $n \geq 73$

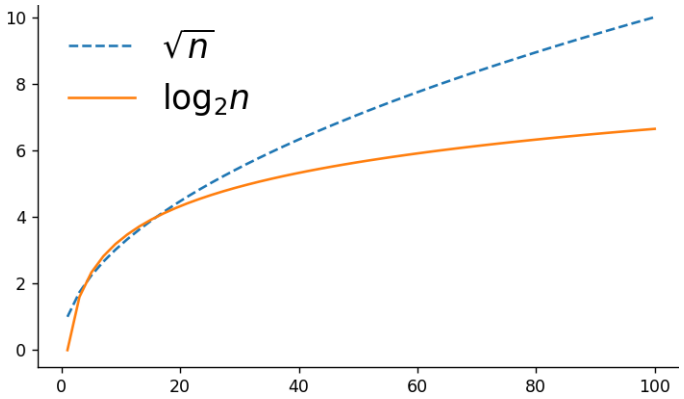
$\Downarrow$ $5/\log_2 (3/2) \leq \sqrt{n} = n/\sqrt{n} \leq n/\log_2 n$ da $\sqrt{n} \geq \log_2 n$ for $n \geq 17$

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

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

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

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



Dvs. det ønskede gælder for $c = 1$ og $N_0 = 73$. □

Sætning $\mathrm{poly}(n) \cdot a^n = \mathrm{O}(b^n)$ for alle $1 \leq a < b$

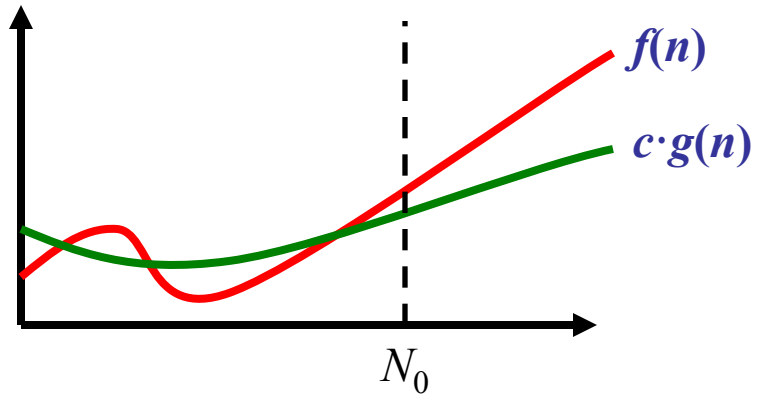
Ω –notation (Omega)

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

hvis $f(n)$ og $g(n)$ er funktioner $N \rightarrow R$ og

findes $c > 0$ og N_0 så *for alle* $n \geq N_0$:

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

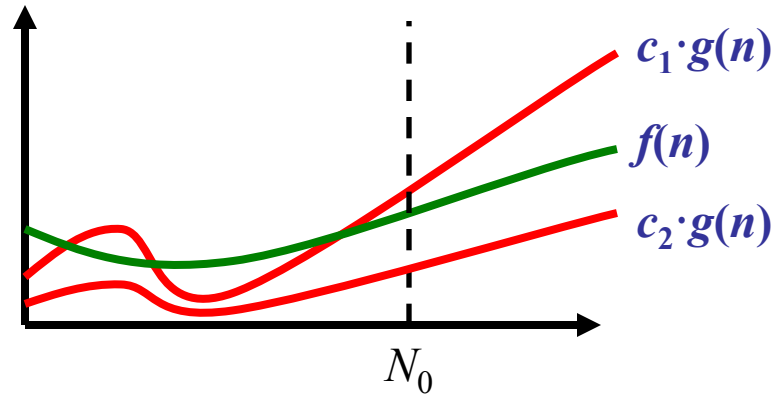


Intuitivt: $f(n)$ er ”større end er lig med” $g(n)$, eller $g(n)$ er ”domineret af” $f(n)$

Θ-notation (Theta)

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

hvis $f(n)=O(g(n))$ og $f(n)= \Omega(g(n)$



Intuitivt: $f(n)$ og $g(n)$ er ”asymptotisk ens”

o-notation (lille o)

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

hvis $f(n)$ og $g(n)$ er funktioner $N \rightarrow R$ og

for alle $c > 0$, *findes* N_0 så *for alle* $n \geq N_0$:

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

Intuitivt: $f(n)$ er ”skarpt mindre end” $g(n)$

ω-notation (lille omega)

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

hvis $f(n)$ og $g(n)$ er funktioner $N \rightarrow R$ og

for alle $c > 0$, *findes* N_0 så *for alle* $n \geq N_0$:

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

Intuitivt: $f(n)$ er ”skarpt større end” $g(n)$

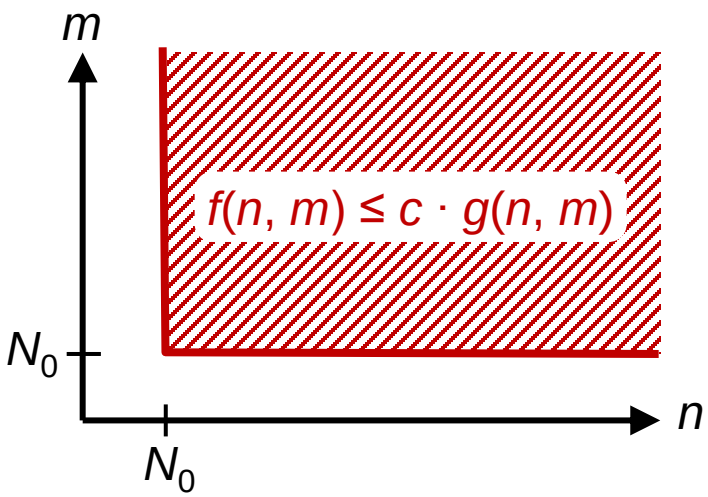
O-notation i flere variable

Vi vil gerne kunne skrive

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

Litteraturen er dog ikke præcis / konsistent...

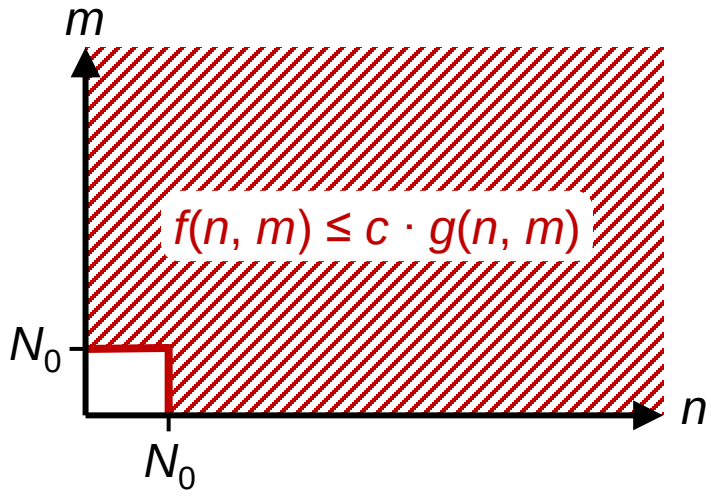
$$f(n, m) = O(g(n, m)) \quad ?$$



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

eller

?



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

Algoritme Analyse

- RAM model
- O-notation

... behøver ikke at beskrive og analysere algoritmer i detaljer !

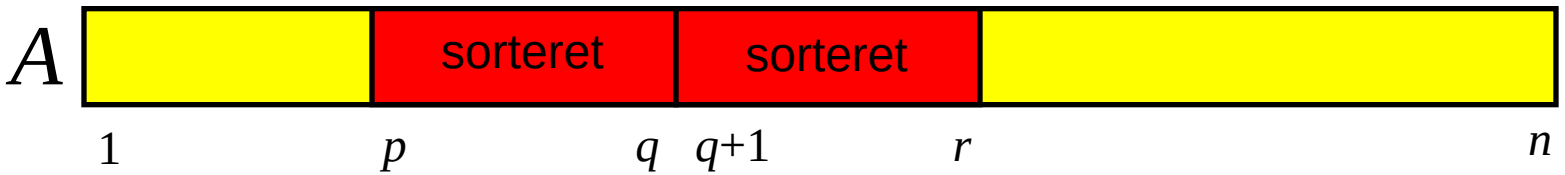
Algoritmer og Datastrukturer

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

Merge-Sort

(Eksempel på Del-og-kombiner)

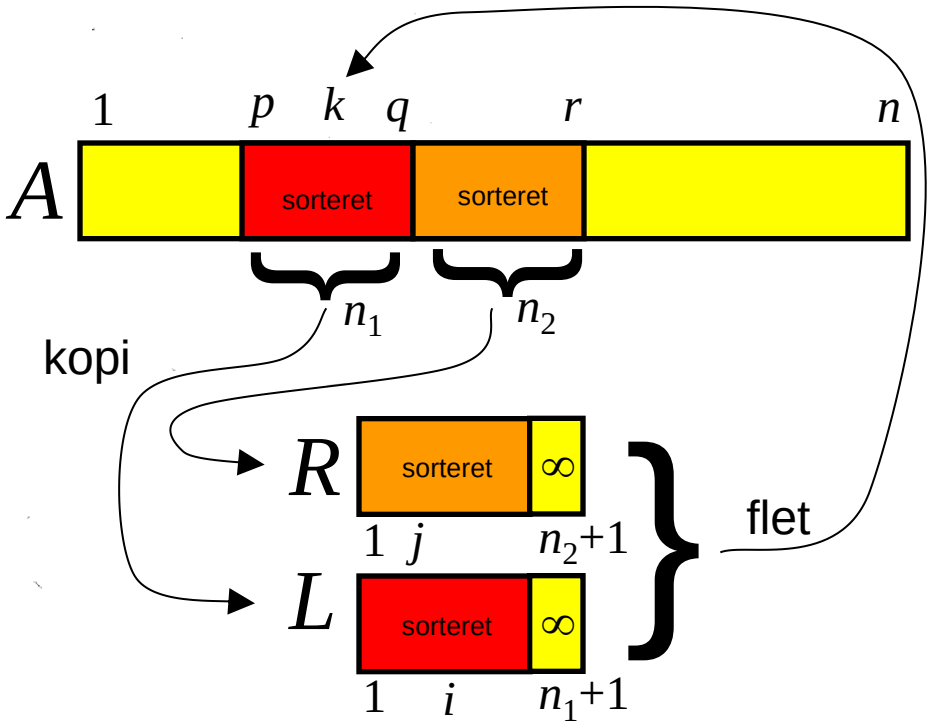
```
MERGE-SORT(A, p, r)
1  if p < r
2      q = ⌊(p + r)/2⌋
3      MERGE-SORT(A, p, q)
4      MERGE-SORT(A, q + 1, r)
5      MERGE(A, p, q, r)
```



I starten kaldes MERGE-SORT(A,1,n)

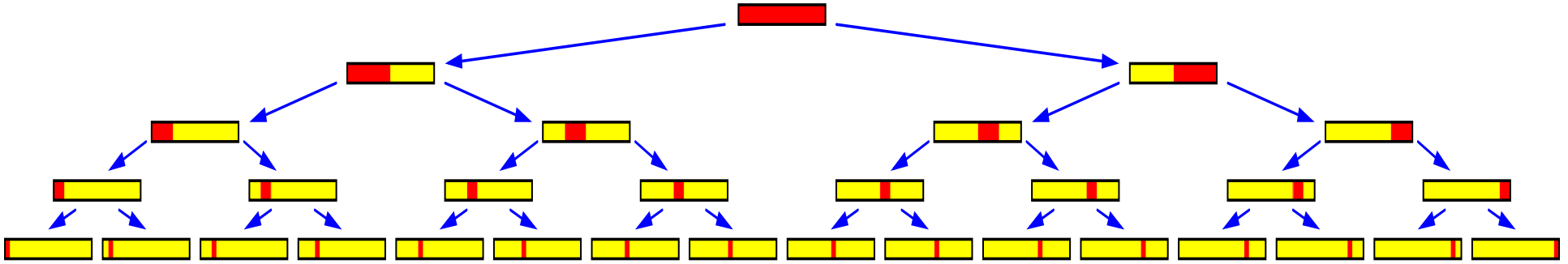
von Neumann, 1945

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



Merge-Sort : Analyse

Rekursionstræet



Observation

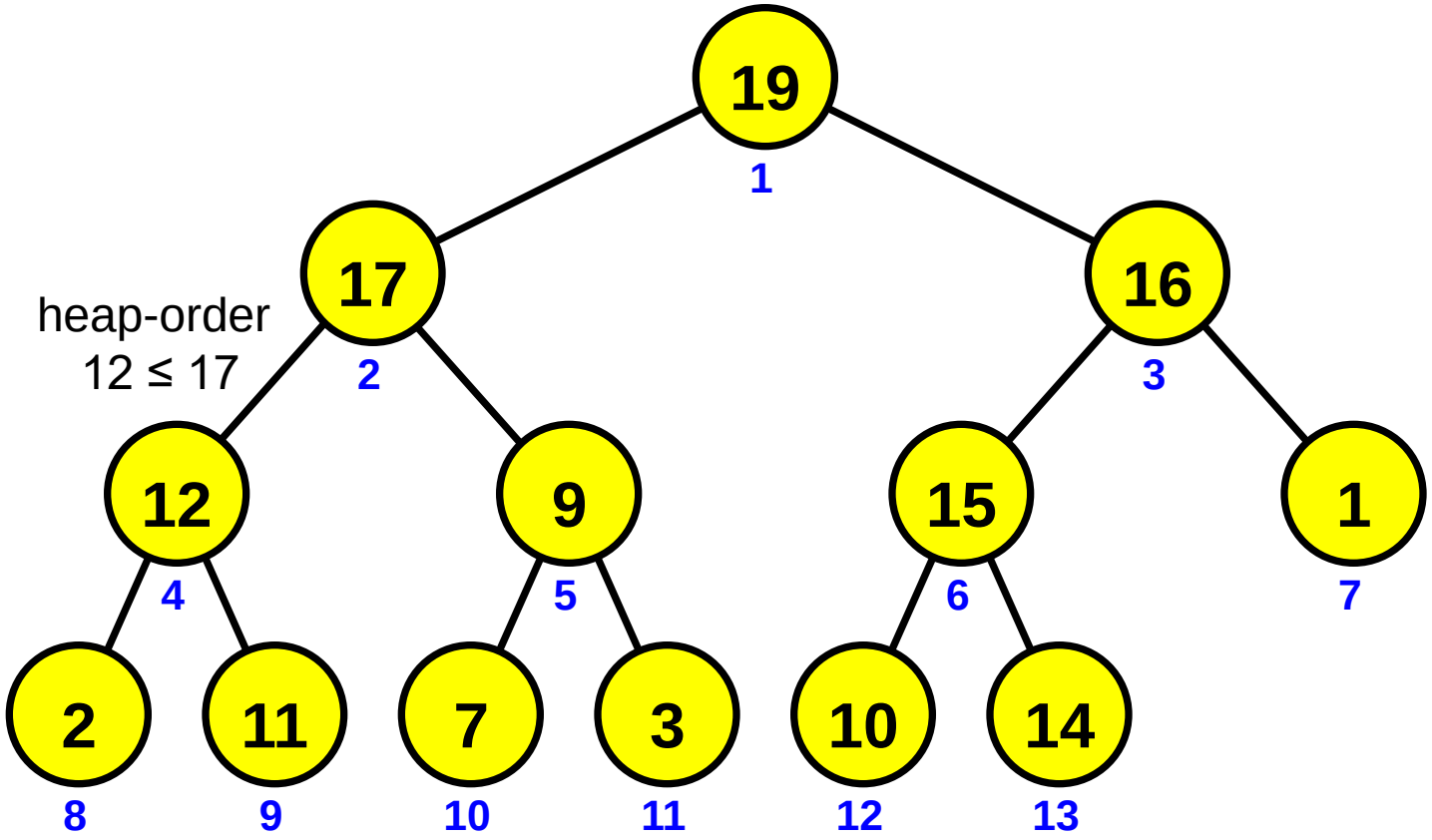
Samlet arbejde per lag er $\mathrm{O}(n)$

Arbejde

$$\mathrm{O}(n \cdot \#\,\mathrm{lag}) = \mathrm{O}(n \cdot \log_2 n)$$

Heap-Sort

Binær (Max-)Heap



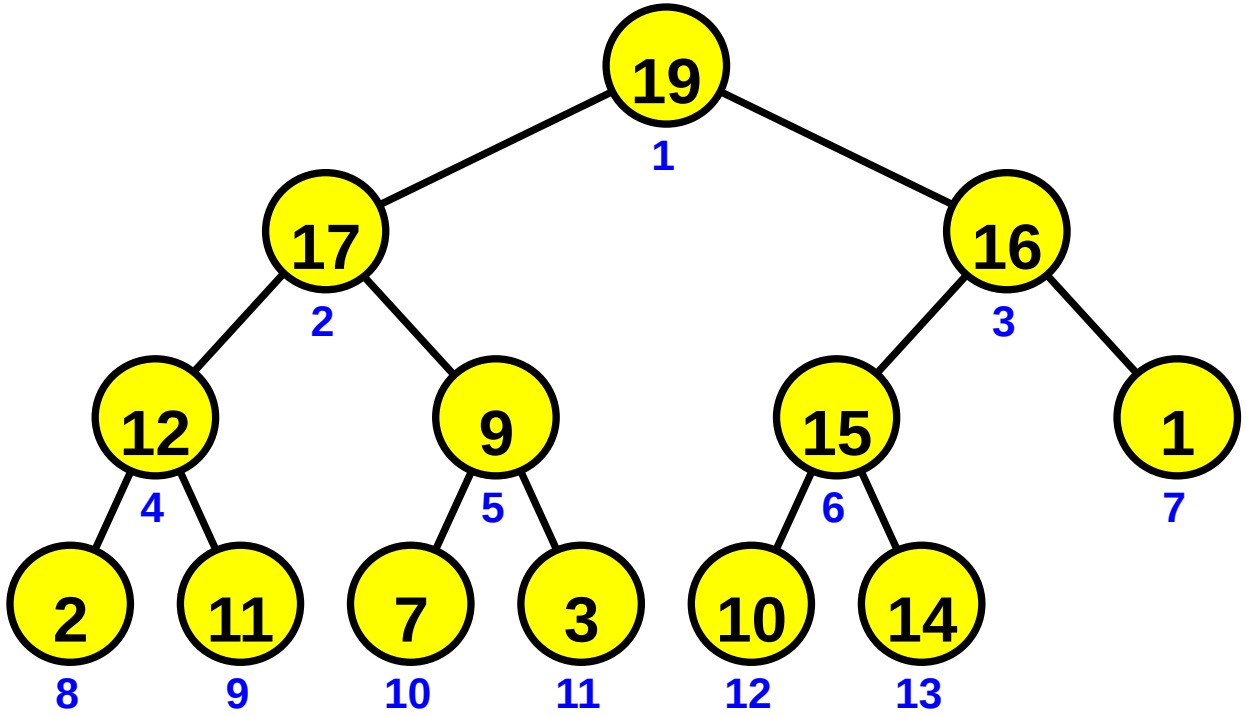
19	17	16	12	9	15	1	2	11	7	3	10	14
1	2	3	4	5	6	7	8	9	10	11	12	13

Williams, 1964

Max-heap : Egenskaber

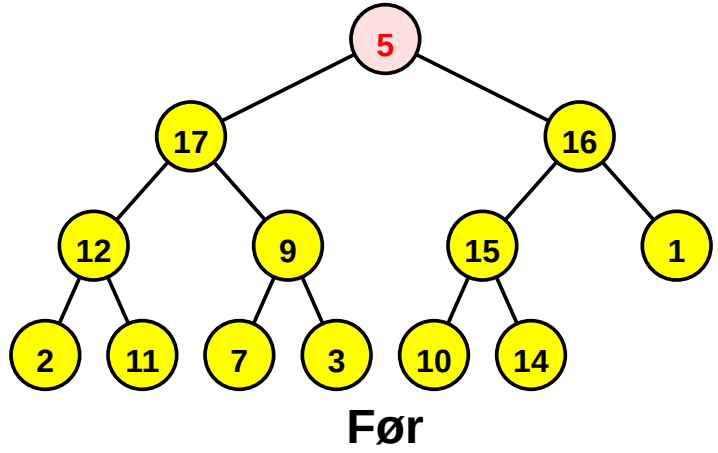
- Roden : knude 1
- Børn til knude i : $2i$ og $2i+1$
- Faren til knude i : $\lfloor i / 2 \rfloor$
- Dybde : $1+\lfloor \log_2 n \rfloor$

  (n = antal elementer)

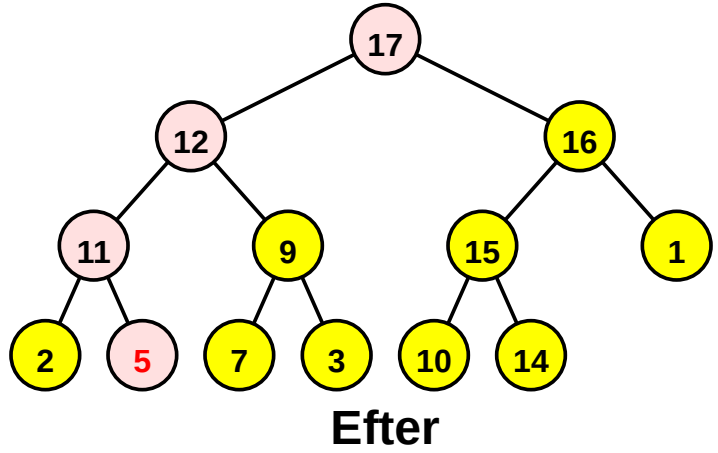


Max-Heapify

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



Før



Efter

Tid $O(\log n)$

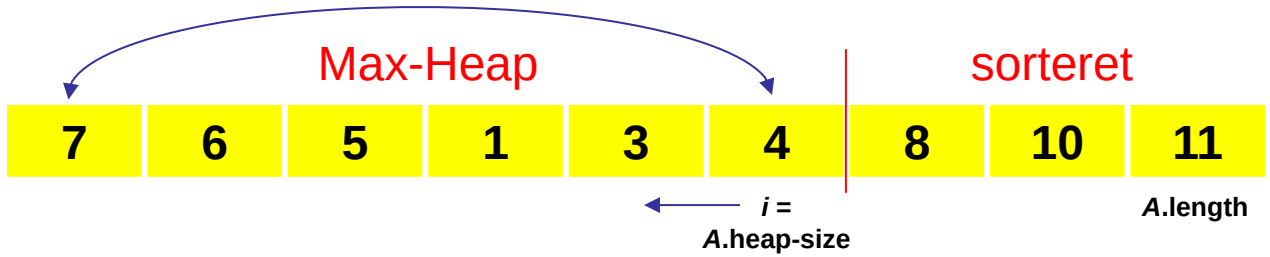
Heap-Sort

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

Floyd, 1964

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

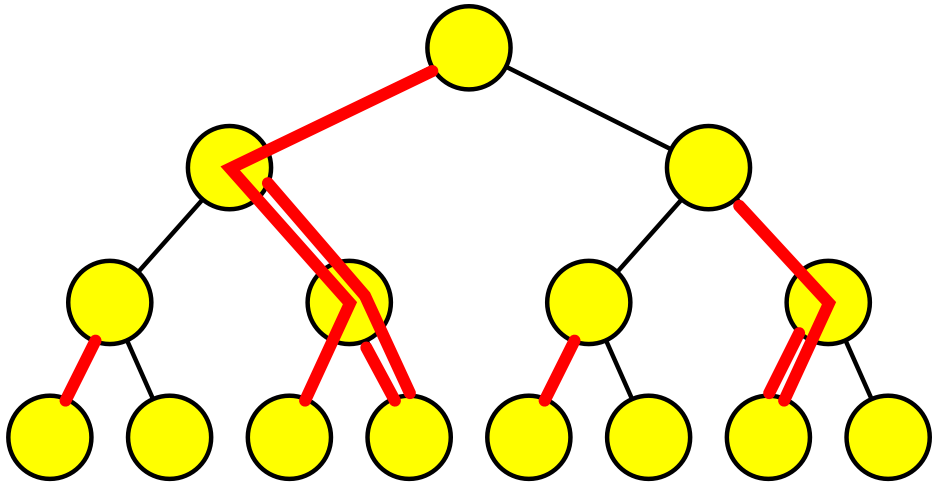
Williams, 1964



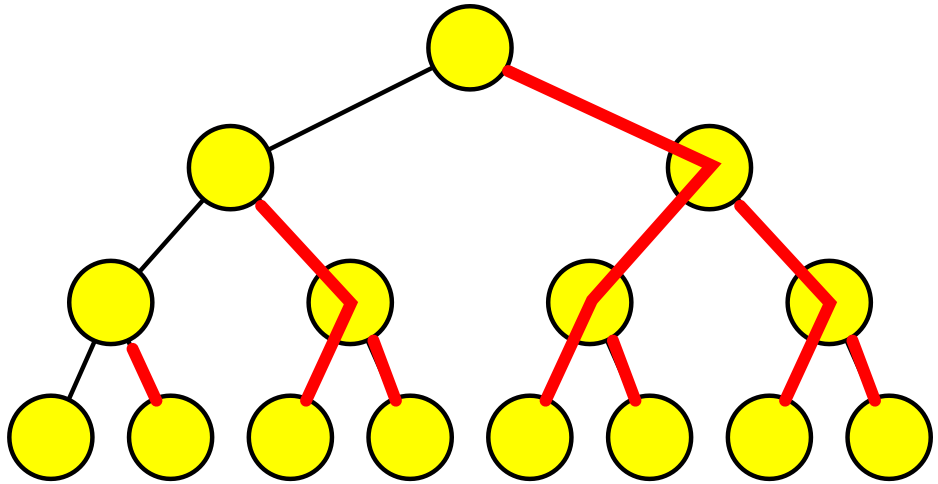
Tid

$O(n \cdot \log n)$

Build-Max-Heap



Max-Heapify stierne (eksempel)



Ikke-overlappende stier med samme #kanter (højre, venstre, venstre...)

Tid for Build-Max-Heap

= Σ tid for Max-Heapify

= # røde kanter

≤ # røde kanter

= n - dybde

= $O(n)$

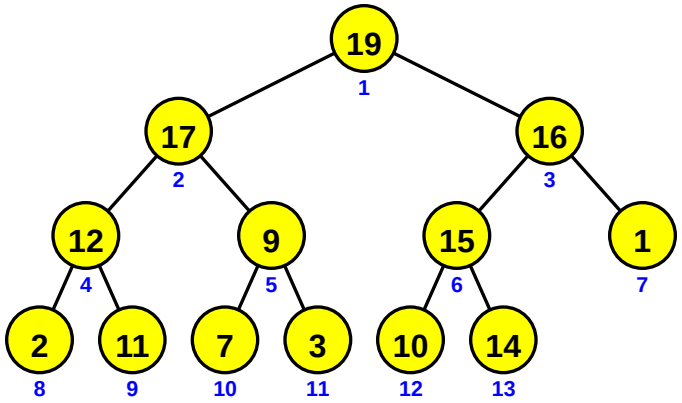
Tid $O(n)$

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

Sorterings-algoritmer

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

Max-Heap operationer




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

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

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

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

Max-Heap operation

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

n = aktuelle antal elementer i heapen

Prioritetskø

En **prioritetskø** er en **abstrakt datastruktur** der gemmer en mængde af **elementer** med tilknyttet **nøgle** og understøtter operationerne:

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

Maximum er med hensyn til de tilknyttede nøgler.

En mulig implementation af en prioritetskø er en heap.

Algoritmer og Datastrukturer

Programmeringsopgaver

Programmeringsopgaver

- Links til opgaver findes på Brightspace.
- 14 dage til at lave dem (normalt IKKE ved TØ).
- **Tæller 20% i endelig karakter.**
- Kopier ikke direkte fra andre grupper (eksamenssnyd).

- Brugernavn og password findes på Brightspace > Grades.

Domjudge

- Check status for indsendte løsninger:

  https://domjudge.cs.au.dk/public

- Man kan godt indsende løsninger efter deadline
  (og få dem godkendt af domjudge),
  men disse tæller IKKE med i karakter.
  Det er status ved afleveringsfristen der tæller.

- Hvis man indsender en forkert løsning efter man har indsendt en rigtig får man stadig fuld point.

Om scoring

- Det er ikke nødvendigt at få alle point i alle opgaver for at få maksimum karakter.
- Ved hver af de 4 runder af programmeringsopgaver står ”Full points *X* / *Y*”.
- Hvis der opnås flere end *X* point i en runde, tæller de ekstra point i intervallet]*X*, *Y*] kun 1/3 til rundens score.
- Endelig karakter baseres på SUM(runde scores) i forhold til SUM(full points).

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

Algoritmer og Datastrukturer

Quicksort
[CLRS, kapitel 7]

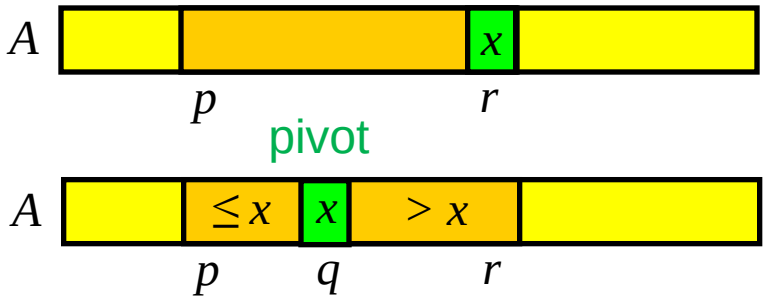
Sandsynligheden for at slå krone

1/2

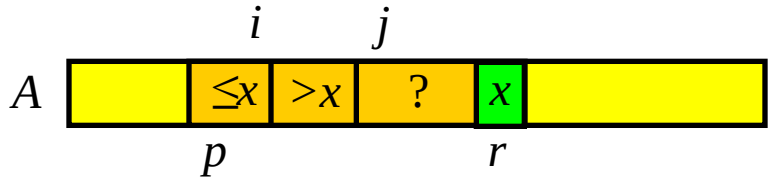
Quicksort:

Sorter $A[p..r]$

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



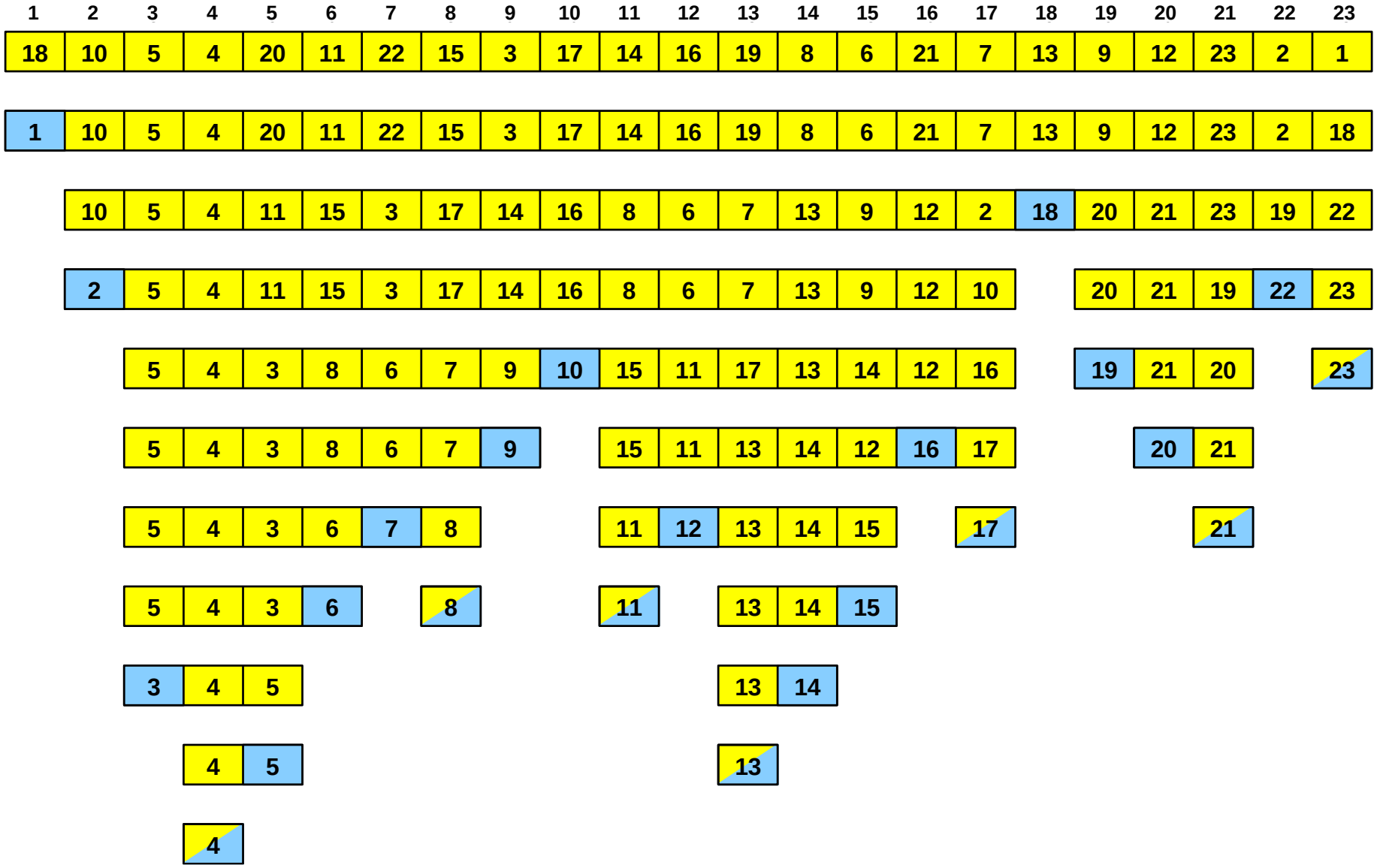

```
PARTITION(A, p, r)
1  x = A[r]
2  i = p − 1
3  for j = p to r − 1
4      if A[j] ≤ x
5          i = i + 1
6          exchange A[i] with A[j]
7  exchange A[i + 1] with A[r]
8  return i + 1
```



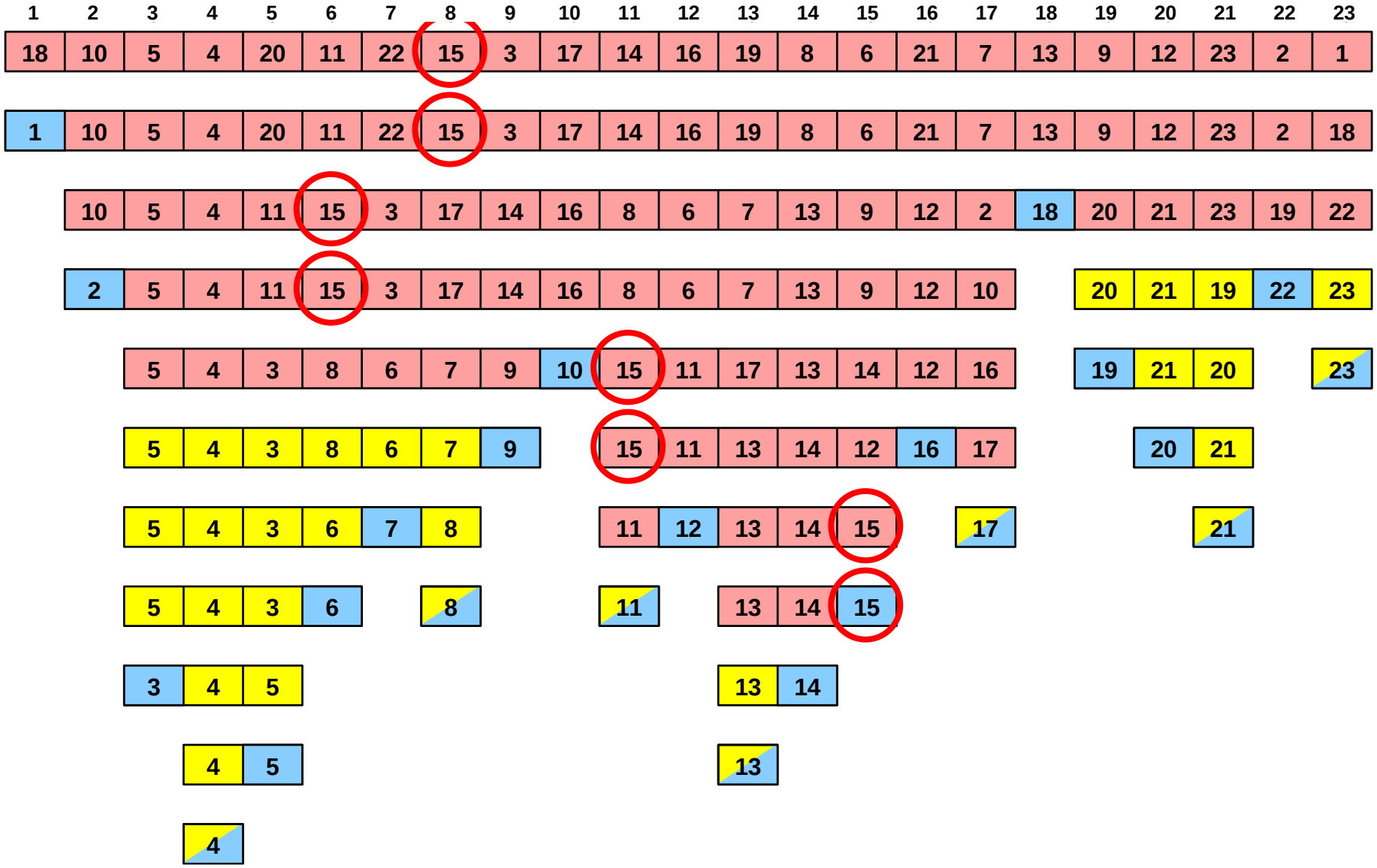
Worst-case tid $O(n^2)$

Hoare, 1961

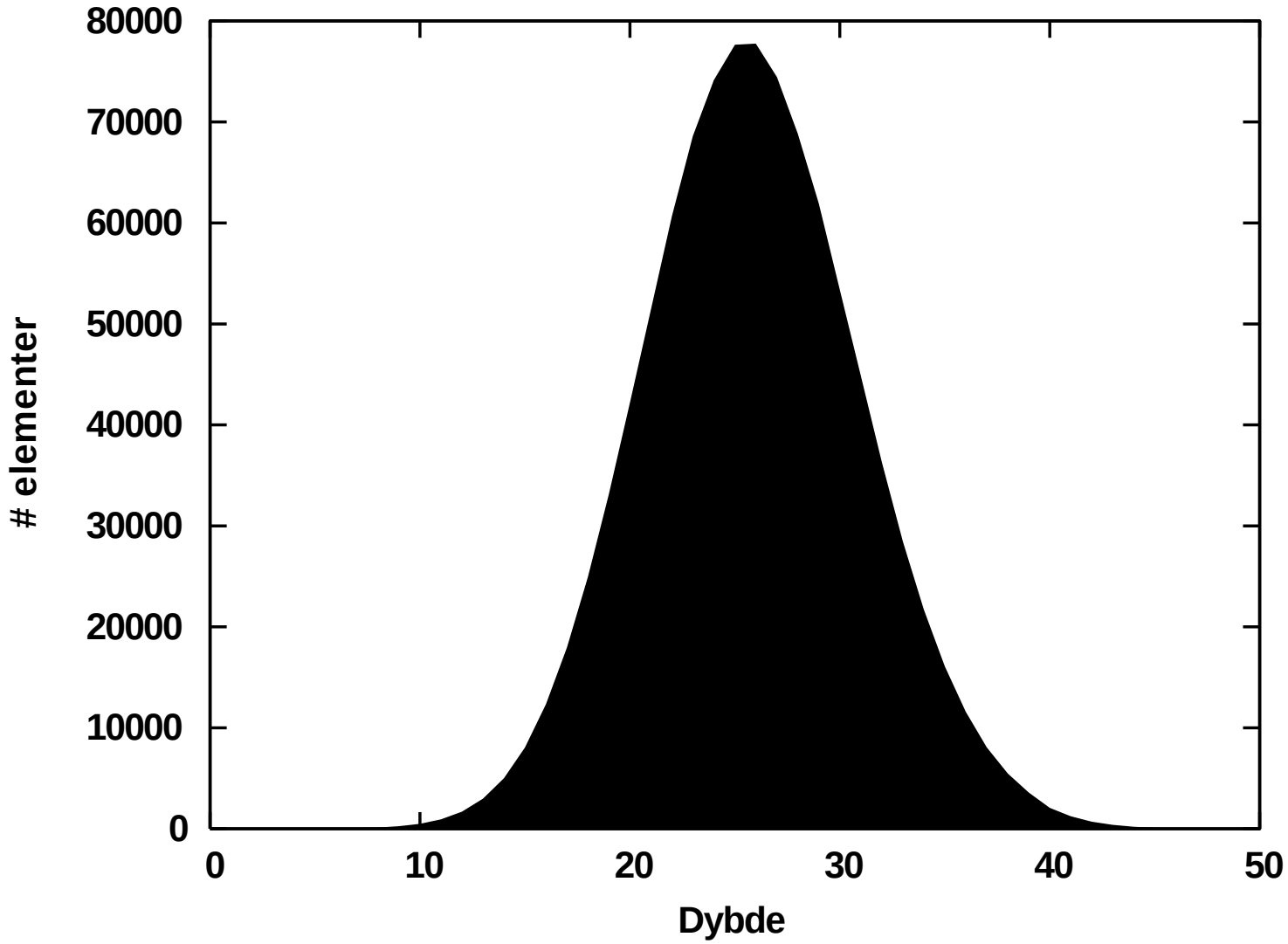
Quicksort på 23 elementer



Quicksort : Rekursionen for 15



Quicksort : Dybde ved $n \approx 2^{20}$



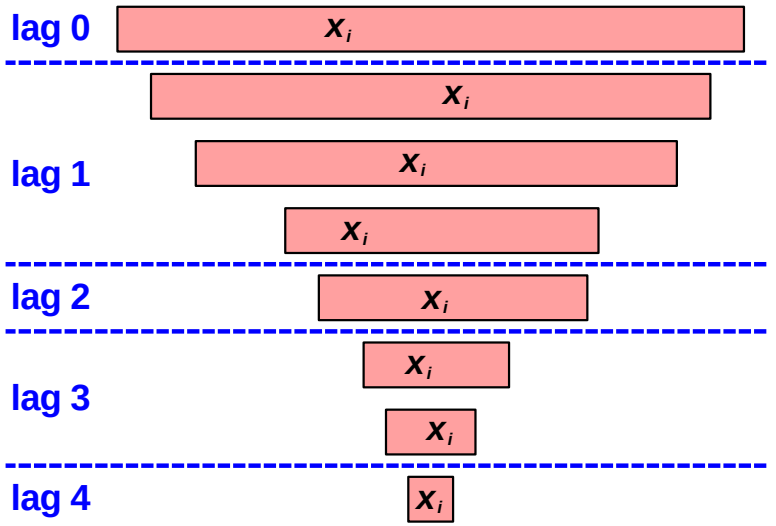
Randomized Quicksort

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

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

Forventet tid $O(n \cdot \log n)$

Randomized Quicksort : Analyse



Lag *i*
hvert array
længde ≤ $n(3/4)^i$

- Et array er i lag j hvis længde $n(3/4)^{j+1}$.. $n(3/4)^j$
- En opdeling er **god** hvis hver del $\leq ¾$ elementer (mindst +1 lag) – sker med sandsynlighed ≥ 0.5
- x_i forventes ≤ 2 gange i hvert lag
- **Forventede dybde af** $x_i \leq 2 \cdot \log_{4/3} n$

Randomized Quicksort : Analyse

Forventede tid for randomized quicksort

$= O(\Sigma_{i=1..n}$ **forventede dybde** af input $x_i)$

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

$= O(n \cdot \log n)$ □

Sorterings-algoritmer

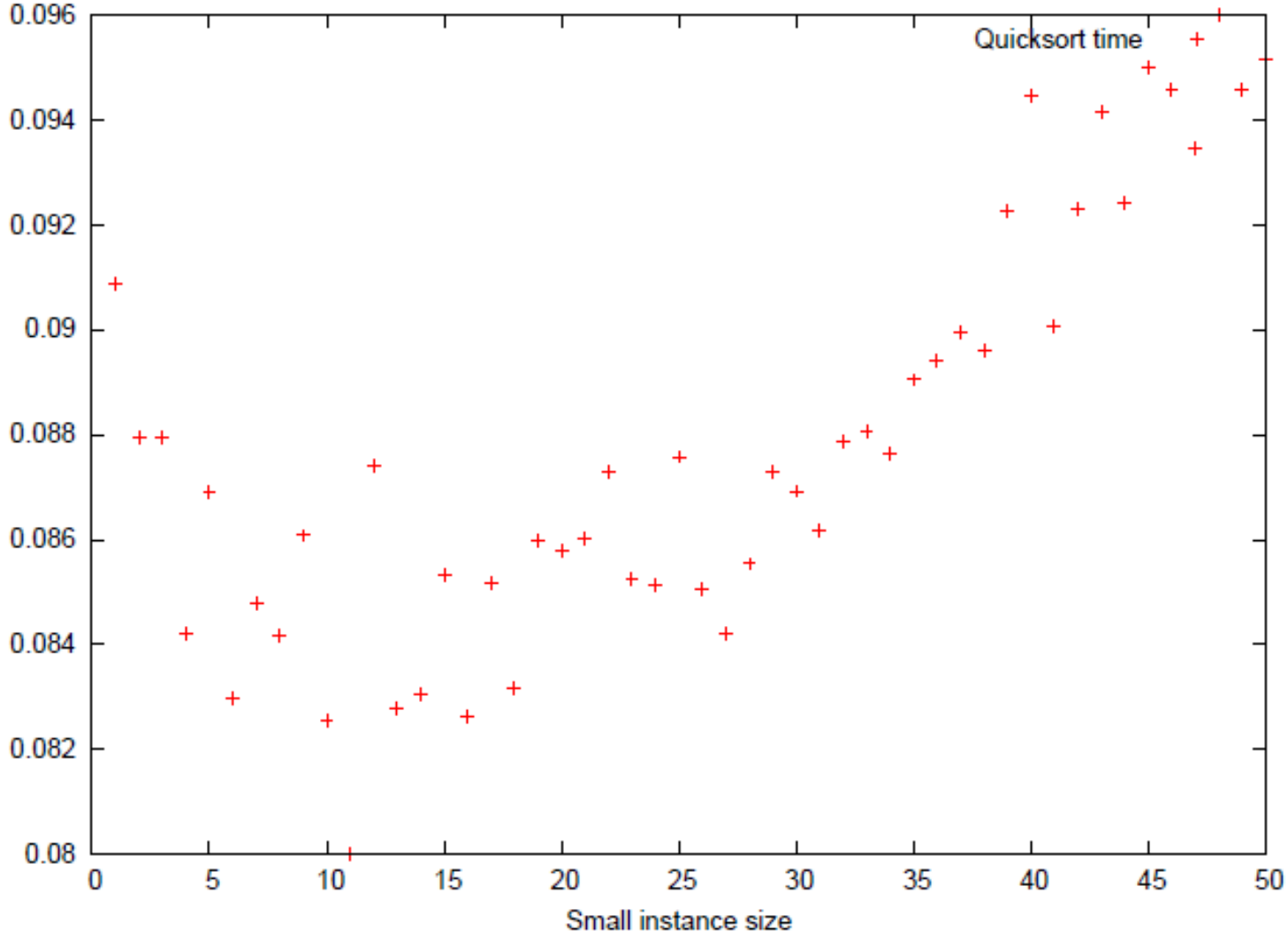
Algoritme	Worst-Case Tid
Heap-Sort	$O(n \cdot \log n)$
Merge-Sort	
Insertion-Sort	$O(n^2)$
Selection-Sort	
QuickSort (Deterministic og randomiseret)	

Algoritme	Forventet tid
Randomiseret QuickSort	$O(n \cdot \log n)$

Sortering:

Eksperimentelle resultater

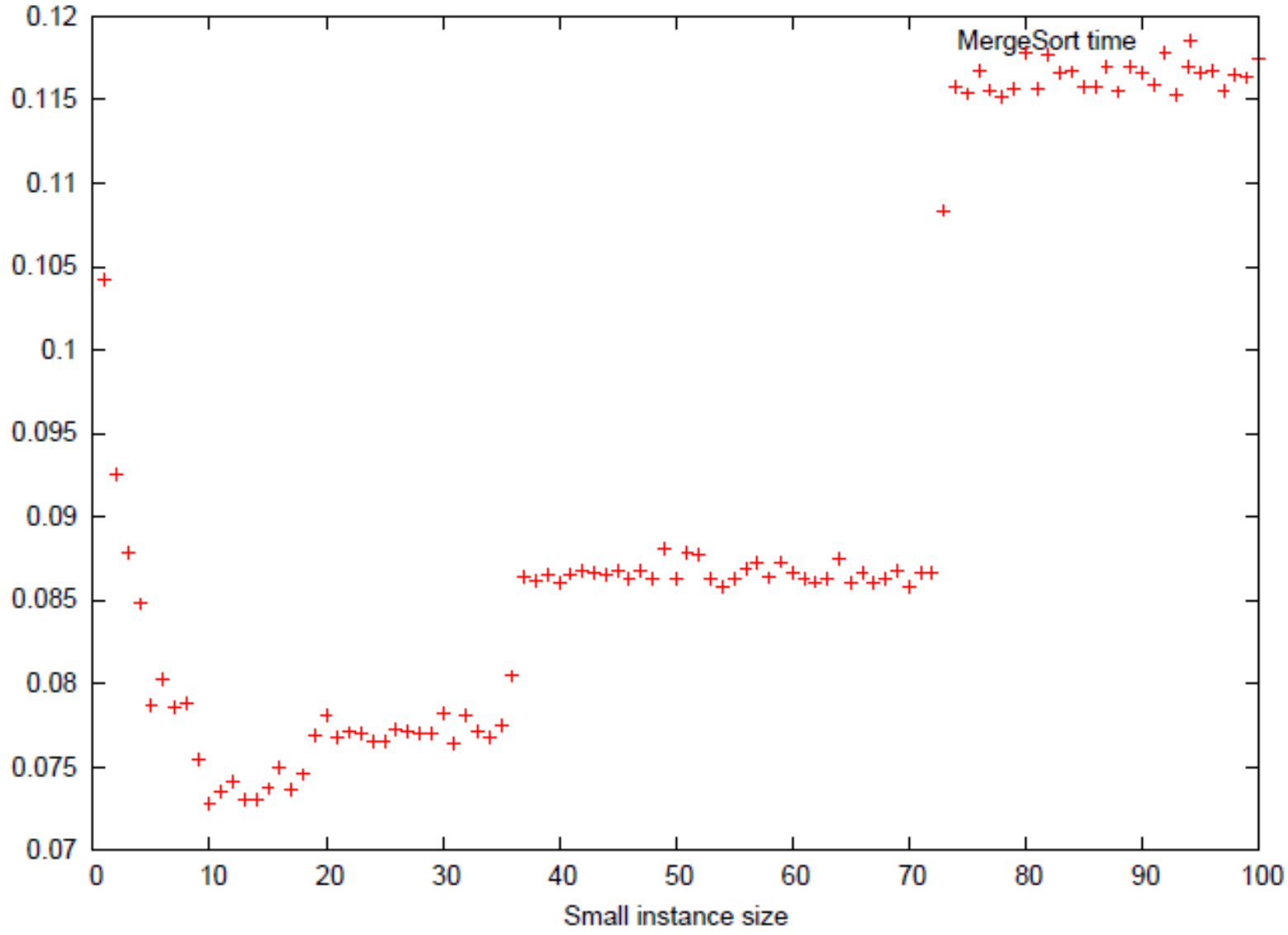
Quicksort med skift til Insertion-sort



Skift til insertion-sort ved små problemstørrelser

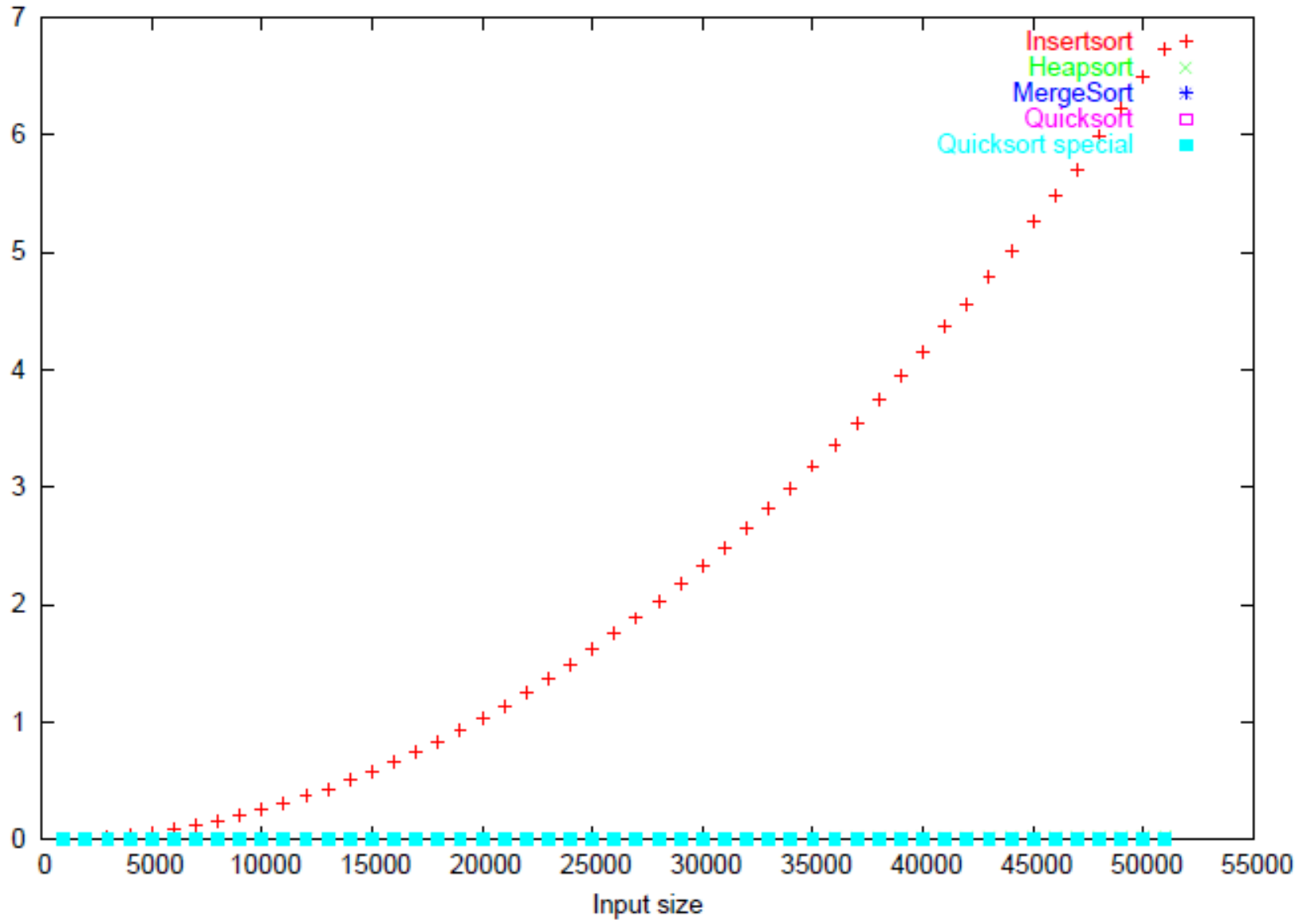
n = 300.000 tilfældige elementer

Mergesort med skift til Insertion-sort

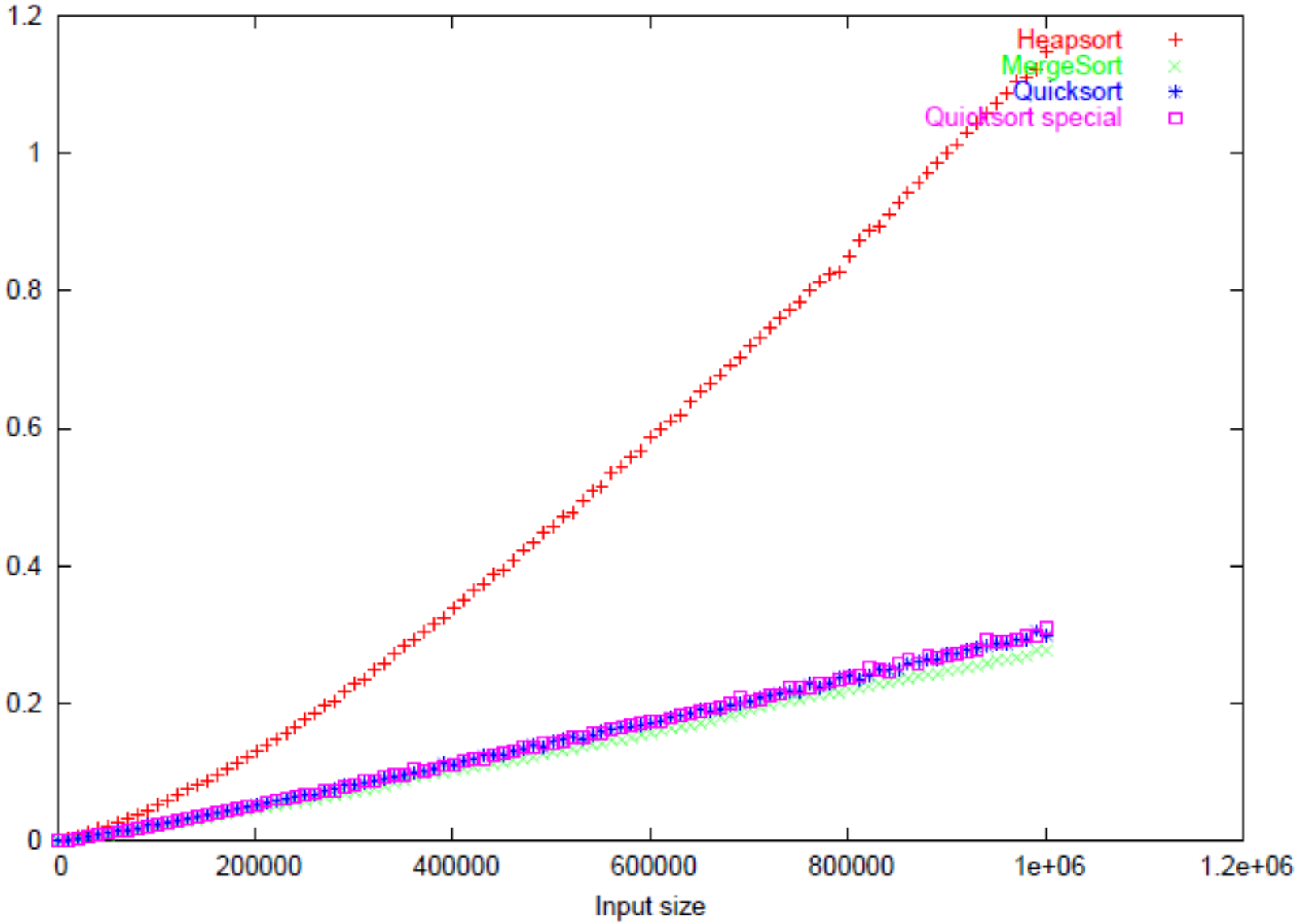


Skift til insertion-sort ved små problemstørrelser n = 300.000 elementer

Tiden for Sorterings Algoritmer



Tiden for Sorterings Algoritmer



Sortering i Praksis

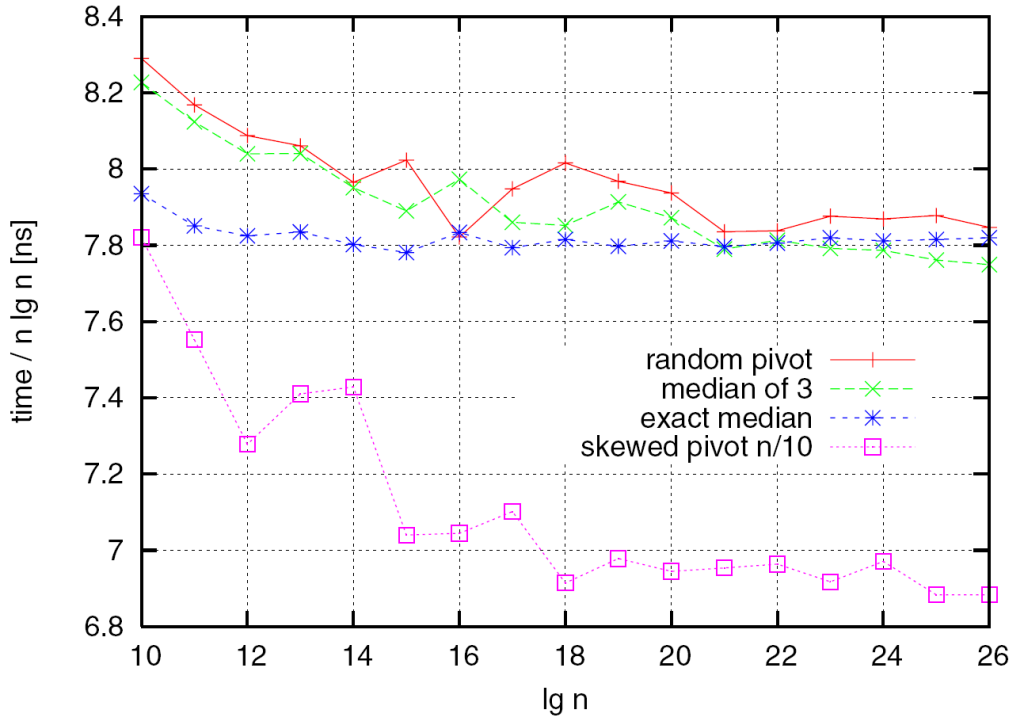
Typisk anvendes **QuickSort** (C, C++, Java)

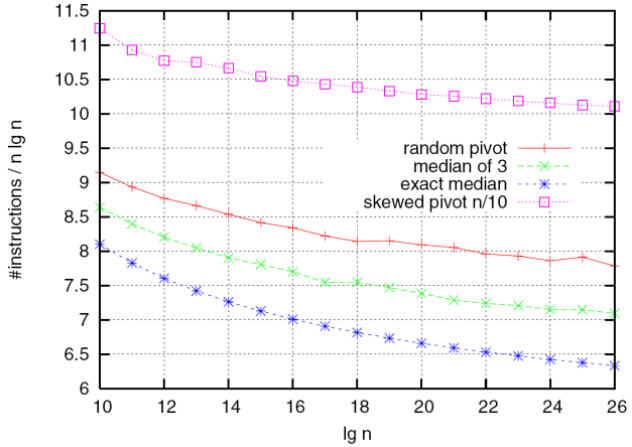
- Inplace (kræver ingen yderligere arrays)
- Hurtig

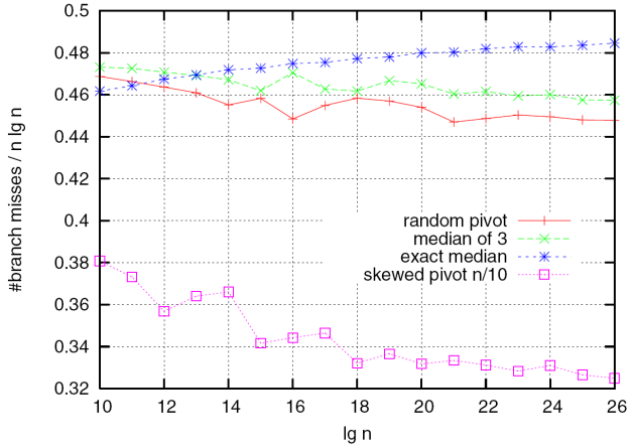
...dog med et par twists

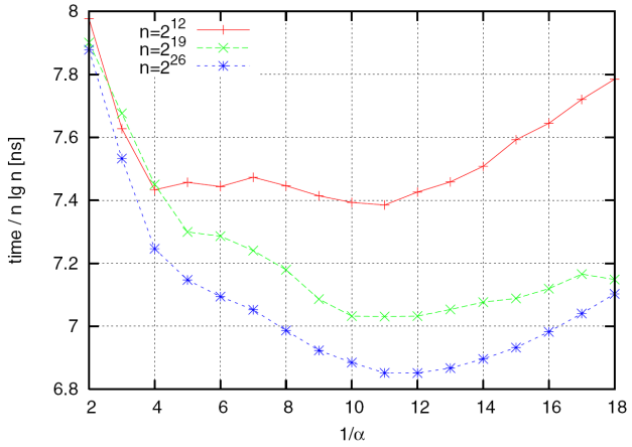
- skift til **InsertionSort** i bunden af rekursionen
- skift til **MergeSort** hvis QuickSort tager for lang tid
- Vælg Pivot'et som medianen ud af O(1) (tilfældige) elementer

“How Branch Mispredictions Affect Quicksort”
Kanela Kaligosi and Peter Sanders
14th Annual European Symposium on Algorithms (ESA 2006)









3GHz Pentium 4 Prescott, GCC 3.3, PAPI, gns. 100 kørsler med tilfældige permutationer af [1..n]

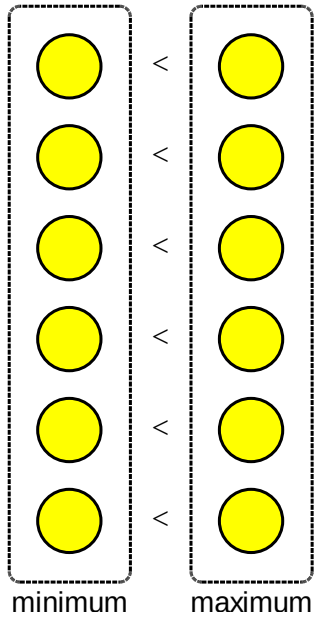
Algoritmer og Datastrukturer

Randomized-select
[CLRS, kapitel 9.1-9.2]

Beregning af Minimum of Maximum

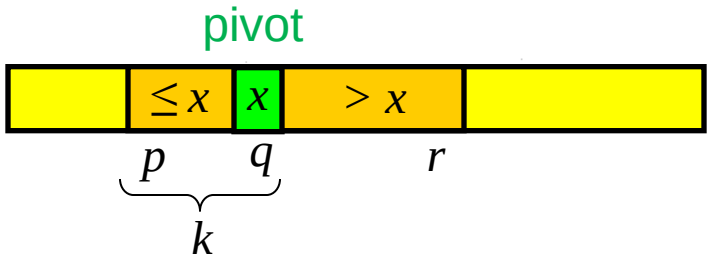
- At finde *minimum* af n elementer kræver n-1 sammenligninger

- At finde *minimum og maximum* af n elementer kræver $3/2 \cdot n$-2 sammenligninger



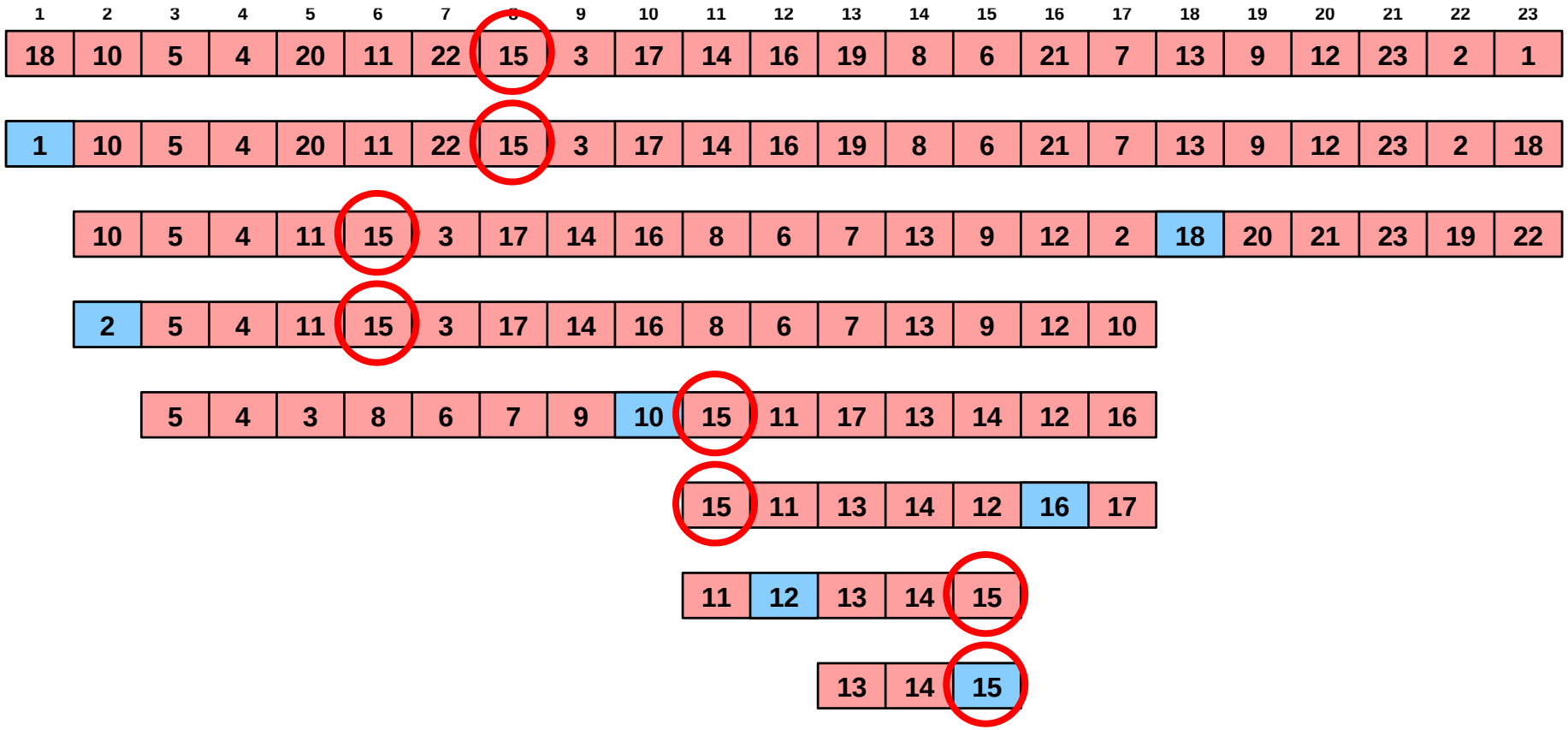
Randomized Select:

Find det i'te mindste element i $A[p..r]$ $(1 \le i \le r\text{-}p\text{+}1)$

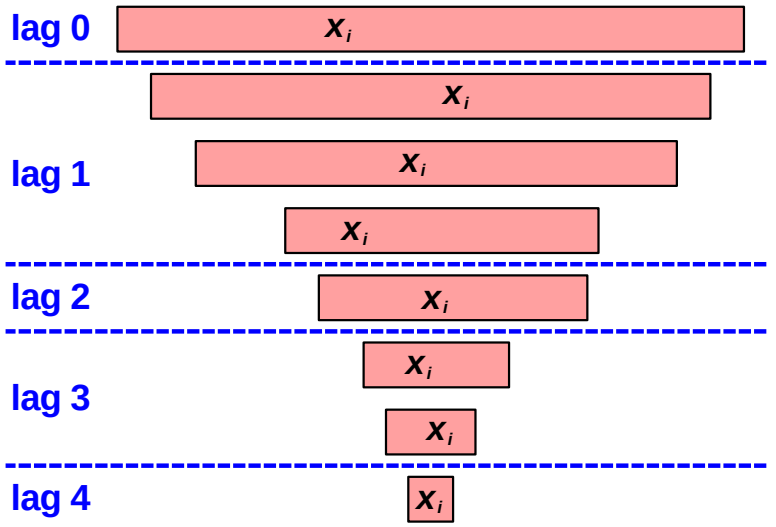



```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6       return A[q]
7  elseif i < k
8       return RANDOMIZED-SELECT(A, p, q − 1, i)
9  else return RANDOMIZED-SELECT(A, q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

Randomized-Select 15



Randomized Select : Analyse



Lag *i*
hvert array
længde ≤ $n(3/4)^i$

Forventede tid for randomized select

$= O(\Sigma_j$ **forventede tid** i lag $j)$

$= O(\Sigma_j\, n\cdot(3/4)^j \cdot$ # **forventede arrays** i lag $j)$

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

$= \mathbf{O(n)}$

□

Selektion

Algoritme	Tid
Randomized-Select [CLRS, Kap. 9.2]	$O(n)$ forventet $O(n^2)$ worst-case
Deterministic-Select [CLRS, Kap. 9.3]	$O(n)$ worst-case

Algoritmer og Datastrukturer

Nedre grænse for sammenligningsbaseret sortering,
Counting-Sort, Radix-Sort, Bucket-Sort
[CLRS, kapitel 8]

Sorterings-algoritmer

(sammenligningsbaserede)

Algoritme	Worst-Case Tid
Heap-Sort	$O(n \cdot \log n)$
Merge-Sort	
Insertion-Sort	$O(n^2)$
QuickSort (Deterministisk og randomiseret)	$O(n^2)$

Algoritme	Forventet tid
Randomiseret QuickSort	$O(n \cdot \log n)$

Hvad er en Sorterings algoritme?

Deterministisk algoritme

- For et givet input gør algoritmen altid det samme

Randomiseret sorterings algoritme

- Algoritmen kan lave tilfældige valg, algoritmens udførsel afhænger af både input og de tilfældige valg

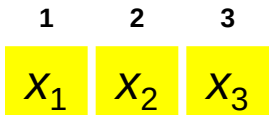
Output

- en permutation af input

Sammenligningsbaseret algoritme

- output afhænger kun af sammenligninger af input elementer

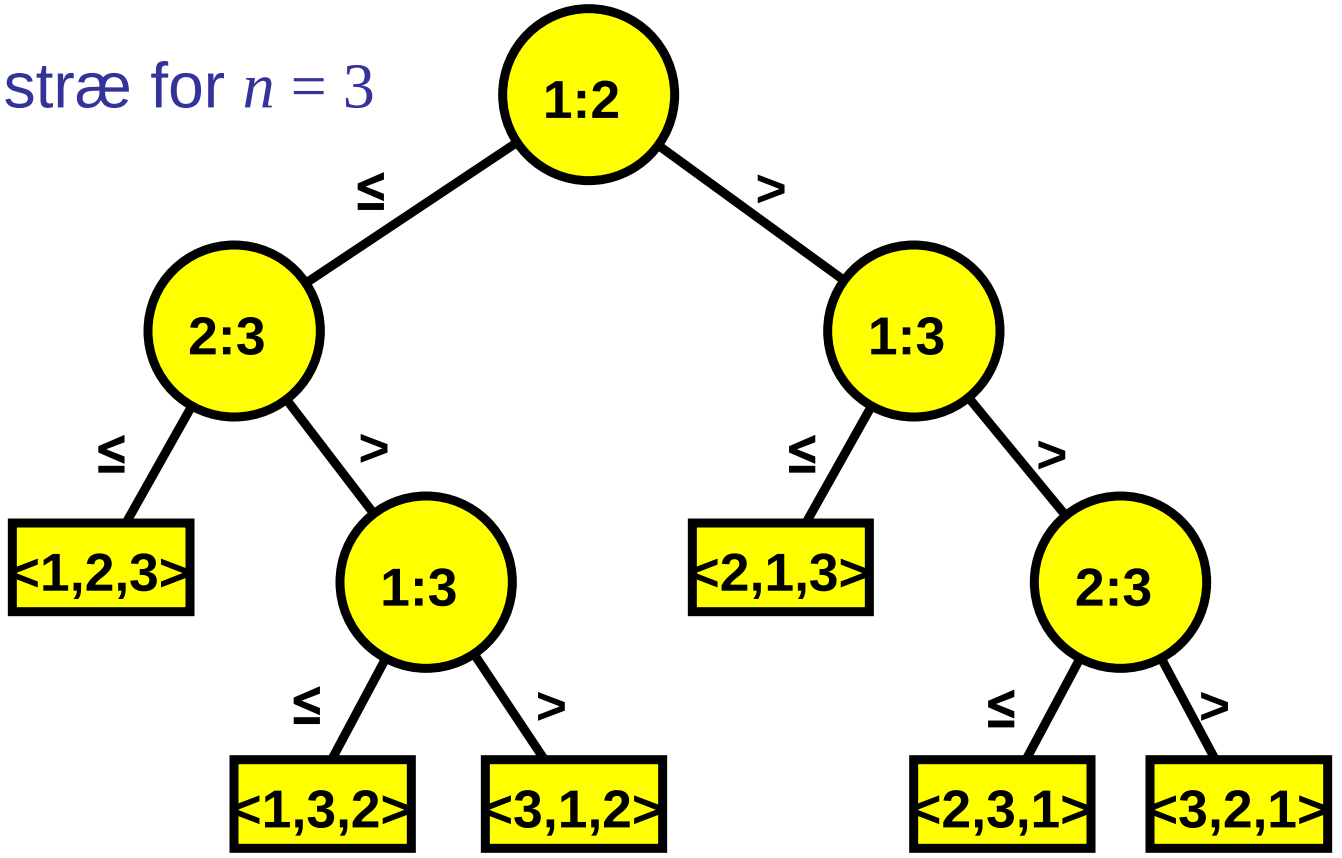
Sammenligninger for Insertion-Sort




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

Sortering ved sammmenligninger: Nedre grænse

Beslutningstræ for $n = 3$



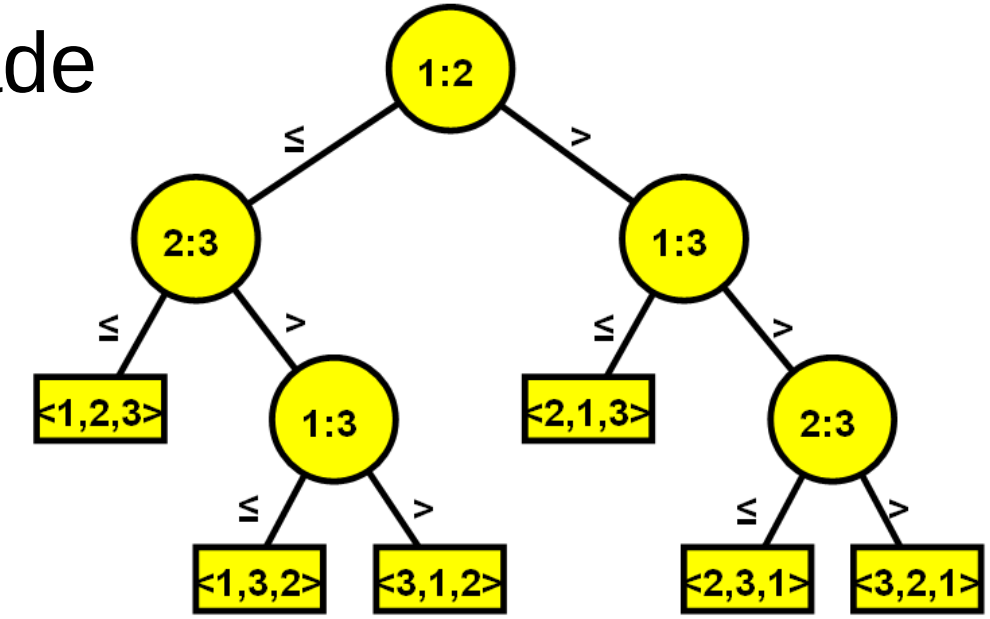
Interne knude $i{:}j$ sammenligner x_i og x_j

Blade beskriver output **permutation**

Sortering ved sammmenligninger: Nedre grænse

$n!$ forskellige output $\leq$ forskellige blade

træ af højde h har $\leq 2^h$ blade



$$n! \leq 2^h$$

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

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

Sortering – Mindste antal sammenligninger

	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
Øvre grænser	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 ”The Ford-Johnson Sorting Algorithm Is Not Optimal”	optimalt																											1191
Nedre grænser	Udtømmende søgning											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**		

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

Sortering af heltal

...udnyt at elementer er bitstrenge

Counting-Sort:

Input: A, ouput: B, tal fra $\{0,1, \ldots,k\}$

```
COUNTING-SORT(A, B, k)
 1  let C[0..k] be a new array
 2  for i = 0 to k
 3      C[i] = 0
 4  for j = 1 to A.length
 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  for j = A.length downto 1
11      B[C[A[j]]] = A[j]
12      C[A[j]] = C[A[j]] − 1
```

Worst-case tid $O(n+k)$

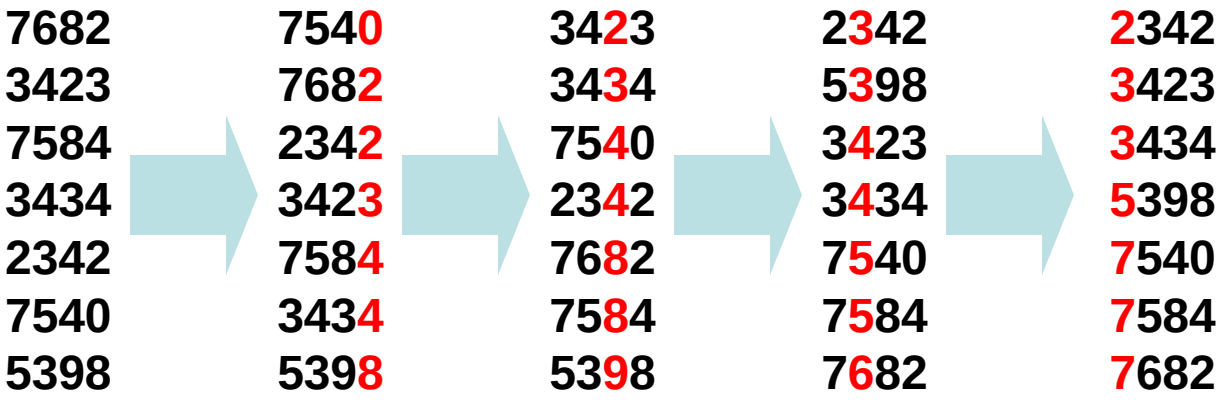
Radix-Sort:

Input: array A, tal med d cifre fra $\{0,1,...,k\}$

RADIX-SORT(A, d)

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

2 use a stable sort to sort array A on digit i



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

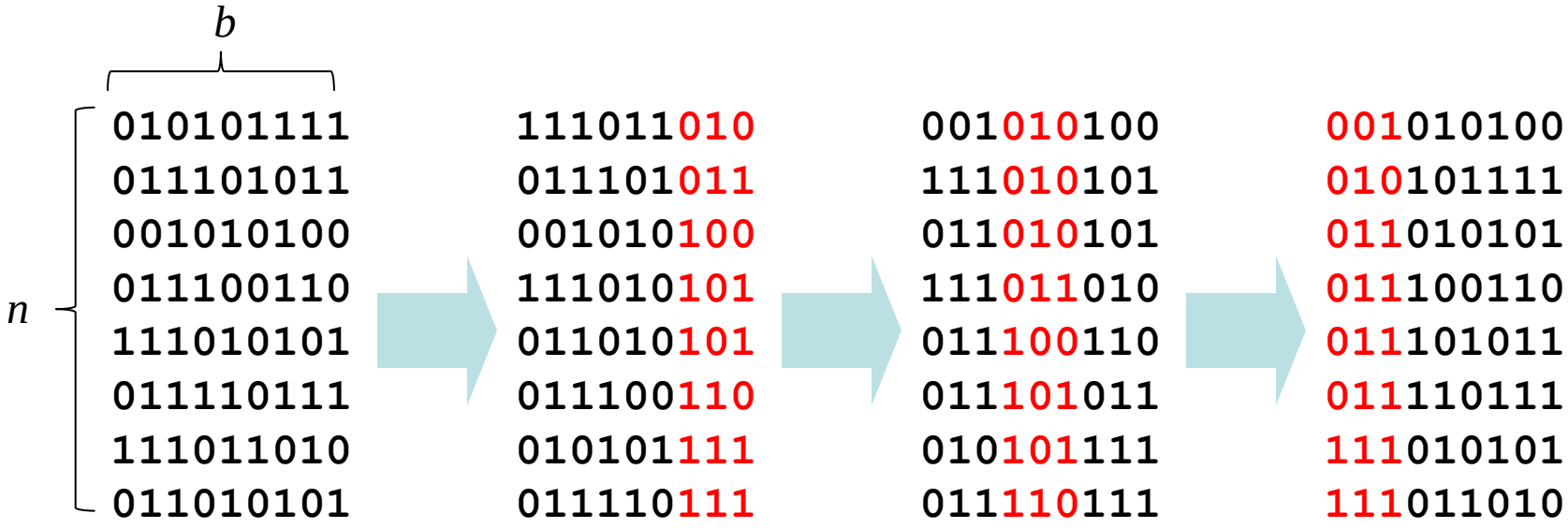
RadixSort hos POST

Trin 1: Sorter efter husnummer

Trin 2: Sorter efter rute

Radix-Sort:

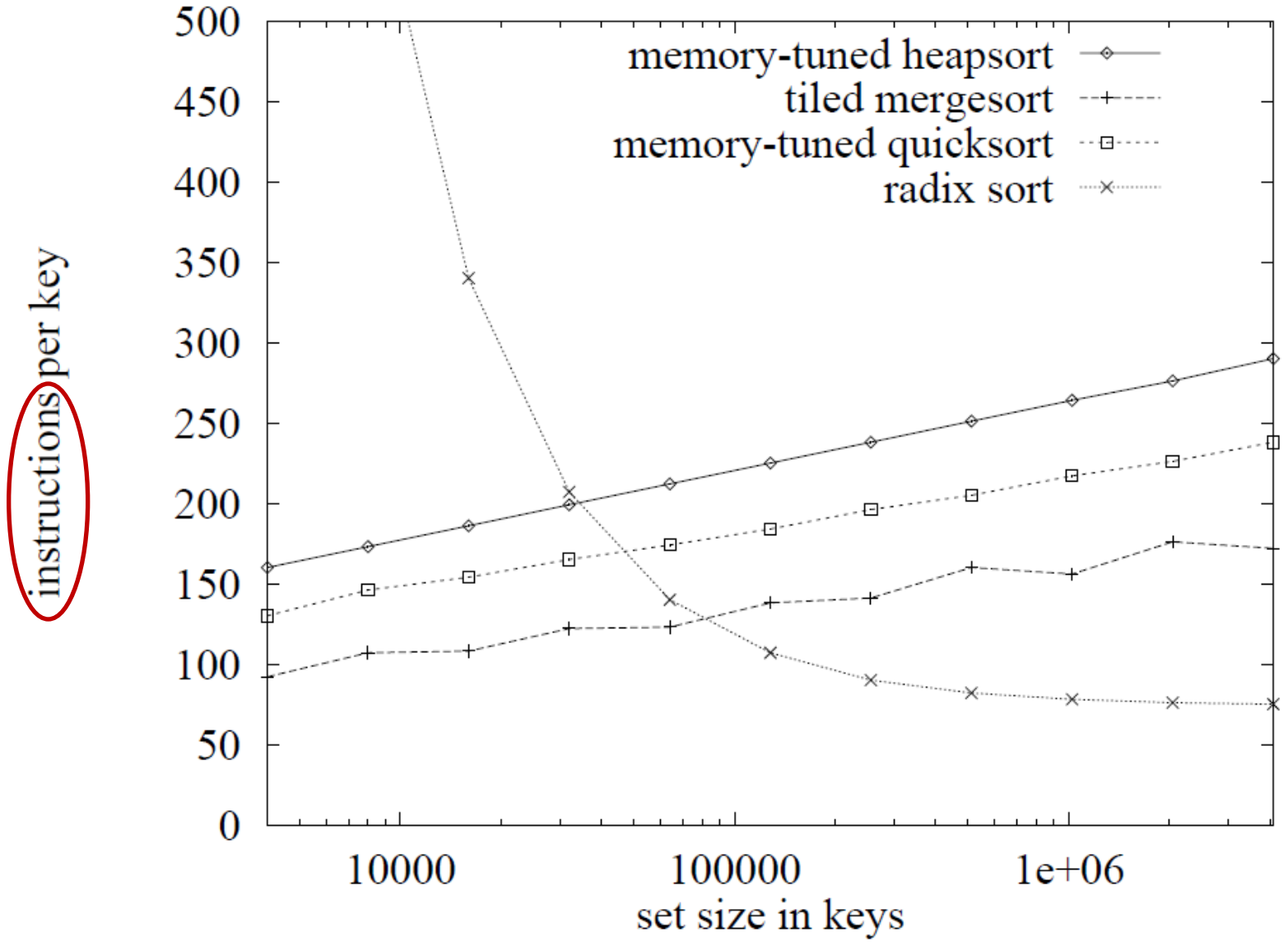
Input: array A, n tal med b bits



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

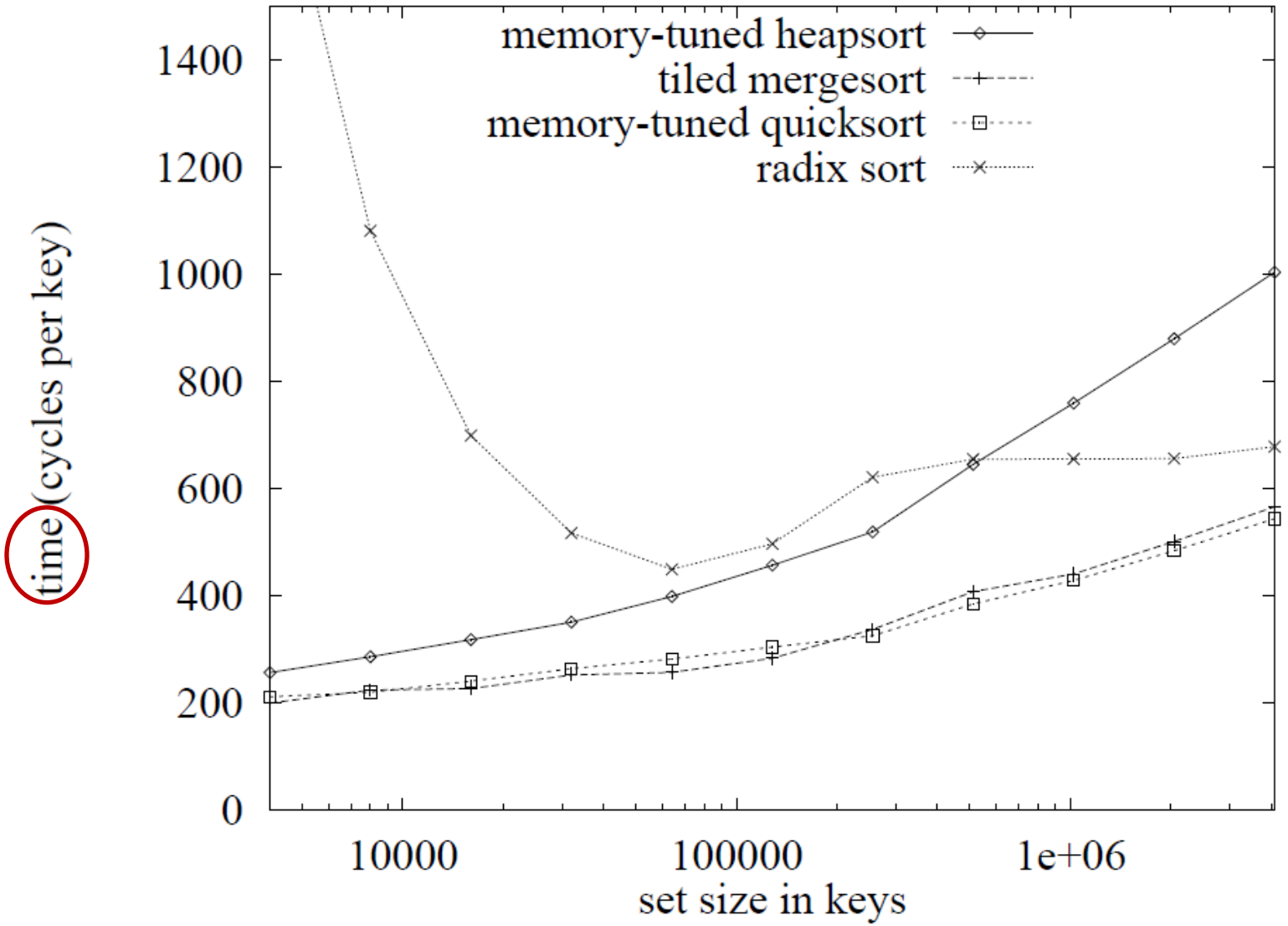
Input n tal med b bits: **Worst-case tid** $O(n \cdot b/\log n)$

Radix-Sort: Eksperimenter



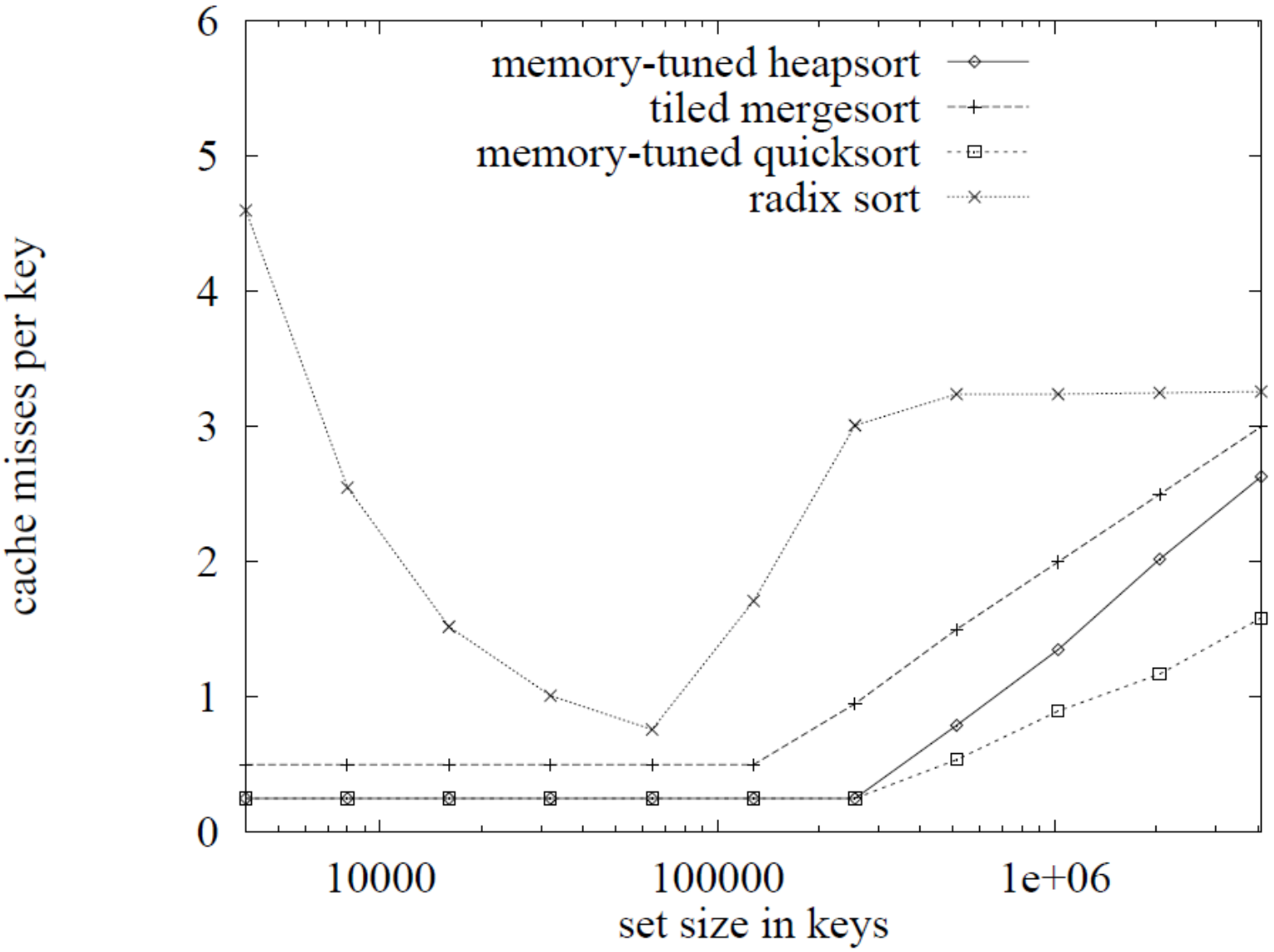
LaMarca, Ladner ’97: The influence of Caches on the Performance of Sorting

Radix-Sort: Eksperimenter



LaMarca, Ladner ’97

Radix-Sort: Eksperimenter

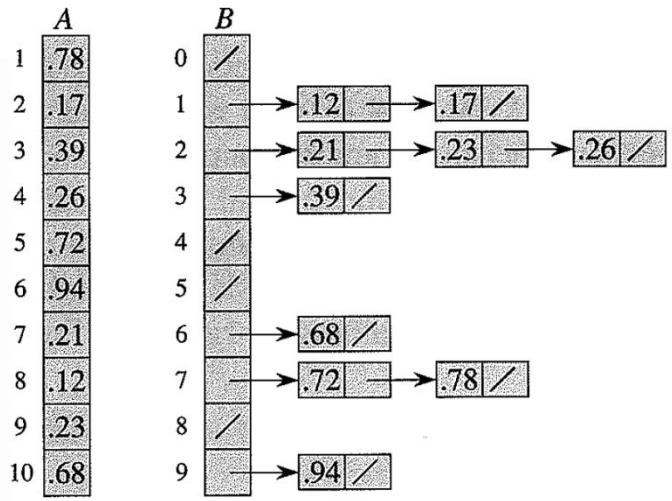


LaMarca, Ladner ’97

Bucket-Sort:

Input: A, reelle tal fra $[0..1[$

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



Forventet tid $O(n)$ – for tilfældigt input

Ikke pensum

Sortering – State-of-the-Art

Bedst kendte sorterings grænse for RAM modellen (med ubegrænset ordstørrelse) er en randomiseret algoritme der bruger $O(n)$ plads og har en forventet køretid på:

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

Om dette kan opnås uden randomisering er ukendt.
$O(n \log\log n)$ er kendt. Kan man opnå forventet $O(n)$ tid?
Med randomisering og for ordstørrelse $\Omega(\log^2 n \cdot \log\log n)$ findes en randomiseret algoritme med forventet $O(n)$ tid.

Yijie Han, Mikkel Thorup: *Integer Sorting in* $O(n\sqrt{\log\log n})$ *Expected Time and Linear Space*. Proc. 43rd Symposium on Foundations of Computer Science, 135-144, 2002.

Arne Andersson, Torben Hagerup, Stefan Nilsson, Rajeev Raman: *Sorting in linear time?* Proc. 27th ACM Symposium on Theory of Computing, 427-436, 1995.

Djamal Belazzougui, Gerth Stølting Brodal, and Jesper Sindahl Nielsen: *Expected Linear Time Sorting for Word Size* $\Omega(\log^2 n \cdot \log\log n)$. SWAT 2014.

Ikke pensum

Sorteringsalgoritmer i Java, Python, C++ og C

	Java (JDK SE 12.0.1)	Python (CPython 3.7)	C++ (GCC 9.1)	C (GNU C)
Metode	Java.util.Arrays.sort	sorted	std::sort	qsort
Dokumentation	https://docs.oracle.com/javase/10/docs/api/java/util/Arrays.html#sort(int[])	https://docs.python.org/3/library/functions.html#sorted	http://www.cplusplus.com/reference/algorithm/sort/	https://www.gnu.org/software/libc/manual/html_node/Array-Sort-Function.html
Algoritme	*Indbyggede typer (int, float, double...)* **Dual Pivot Quicksort** Insertion-Sort for ≤ 47 elementer Merge-Sort for input ≤ 67 sorterede “runs” Counting-Sort for stor BYTE og SHORT input *Object* **Tim Sort**	**Tim Sort** Merge-Sort variant der identificerer sorterede “runs” til at reducere antal fletninger	**Introsort** Quicksort (max dybde 2·log n) ellers skift til Heapsort Insertion-Sort ≤ 16 elements	**Quicksort** Ikke rekursiv
Kildekode	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\DualPivotQuicksort.java fra C:\ProgramFiles\Java\jdk-12.0.1\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

Ikke pensum

Binary Heaps i Java, Python, C++

	Java (JDK SE 12.0.1)	Python (CPython 3.7)	C++ (GCC 9.1)
Metode	java.util.PriorityQueue	heapq	std::priority_queue
Dokumentation	https://docs.oracle.com/javase/10/docs/api/java/util/PriorityQueue.html	https://docs.python.org/3.7/library/heapq.html	http://www.cplusplus.com/reference/queue/priority_queue/
Kildekode	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\PriorityQueue.java fra C:\ProgramFiles\Java\jdk-12.0.1\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

Alle tre implementation bruger 0-indekserede arrays hvor

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

Algoritmer og Datastrukturer

Stakke, køer
[CLRS, kapitel 10]

[CLRS, Del 3] : Datastrukturer

Opretholde en struktur for en **dynamisk** mængde data

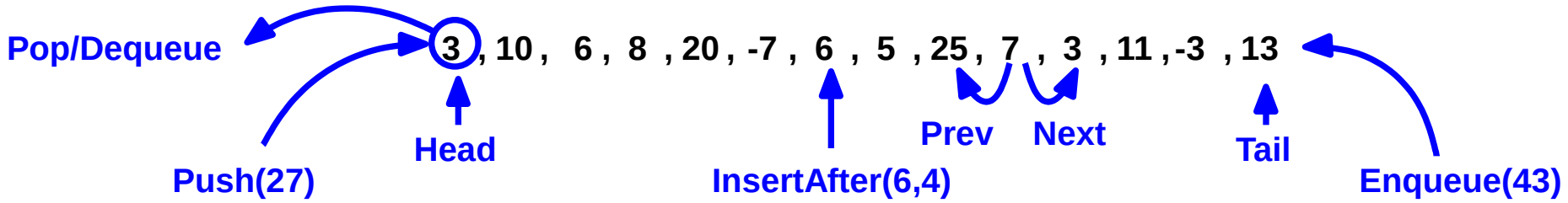
Abstrakte Datastrukturer for Mængder

-Min-prioritetskø
-Max-prioritetskø
- Ordbog

Forespørgsel	Minimum(S)	pointer til element	●		
	Maximum(S)	pointer til element		●	
	Search(S, x)	pointer til element			●
	Member(S, x)	TRUE eller FALSE			
	Successor(S, x)	pointer til element			
	Predecessor(S, x)	pointer til element			
Opdateringer	Insert(S, x)	pointer til element	●	●	●
	Delete(S, x)	-			●
	DeleteMin(S)	element	●		
	DeleteMax(S)	element		●	
	Join(S_1, S_2)	mængde S			
	Split(S, x)	mængder S_1 og S_2			

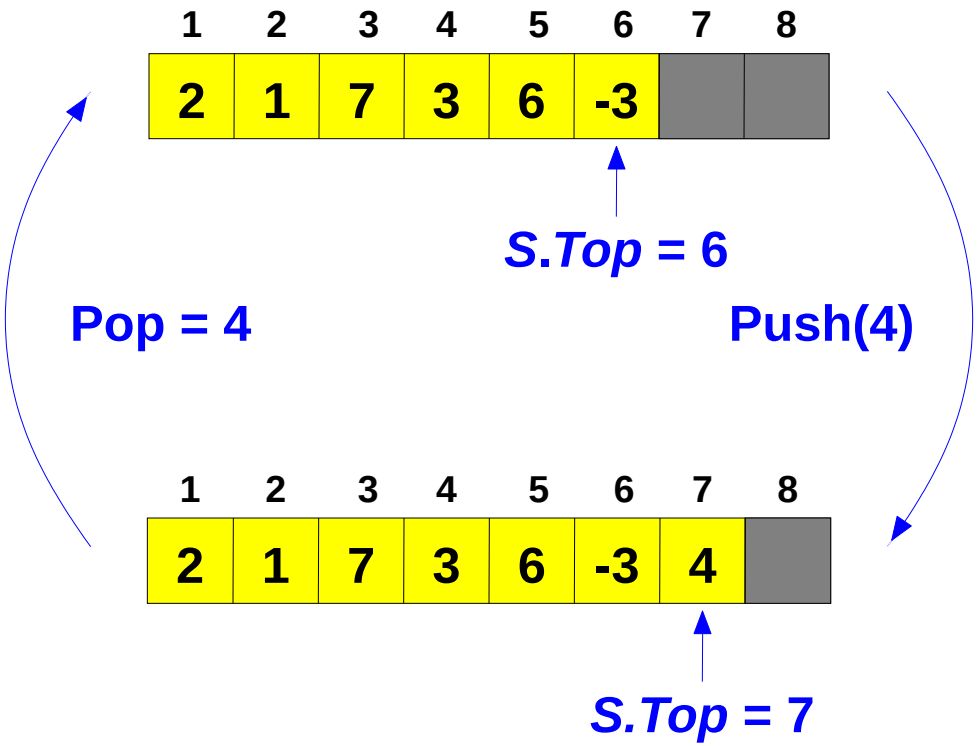
Abstrakte Datastrukturer for Lister

			-Stak	-Kø
Forespørgsel	Empty(S)	TRUE eller FALSE	●	●
	Head(S), Tail(S)	pointer til element		
	Next(S, x), Prev(S, x)	pointer til element		
	Search(S, x)	pointer til element		
Opdateringer	Push(S, x)	-	●	
	Pop/Dequeue(S)	element	●	●
	Enqueue(S, x)	-		●
	Delete(S, x)	element		
	InsertAfter(S, x, y)	pointer til element		



Stak

Stak : Array Implementation



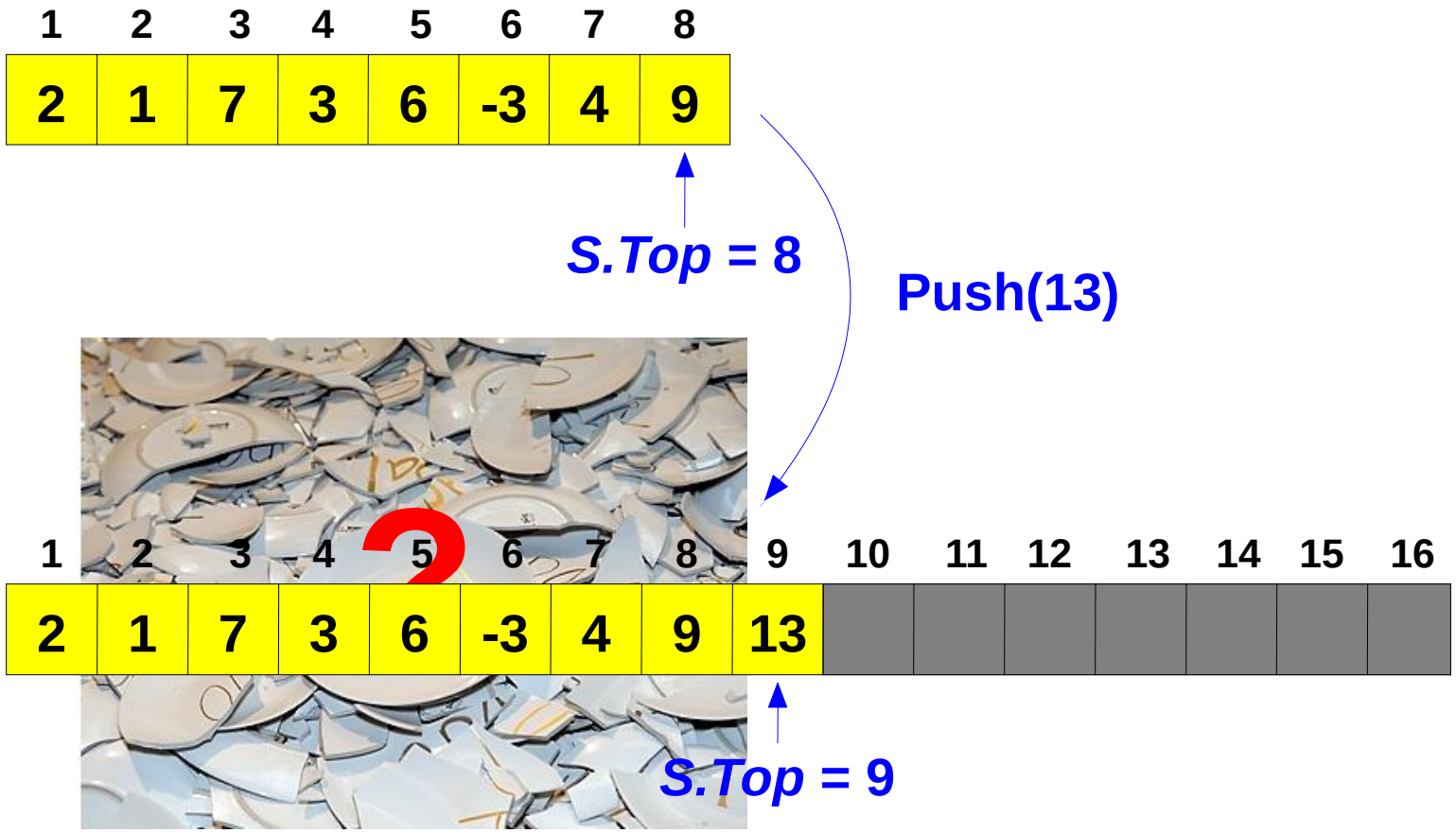

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

```
PUSH(S, x)
1  S.top = S.top + 1
2  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 : $O(1)$ tid

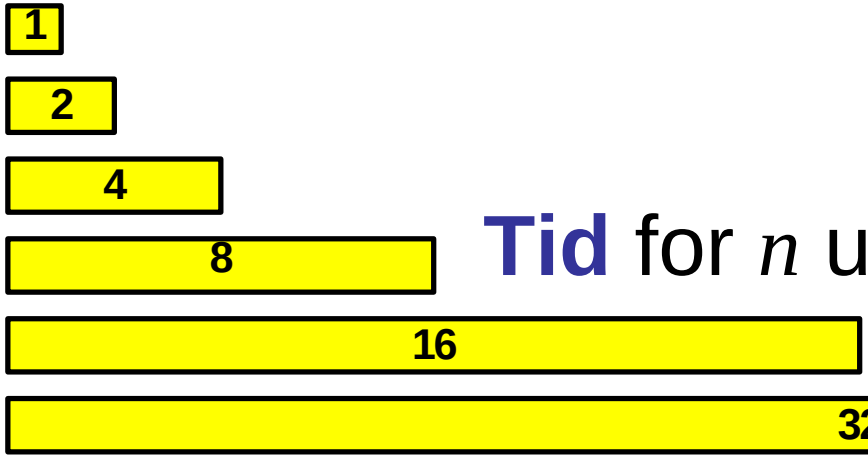
Stak : Overløb



Array fordobling : $O(n)$ tid

Array Fordobling

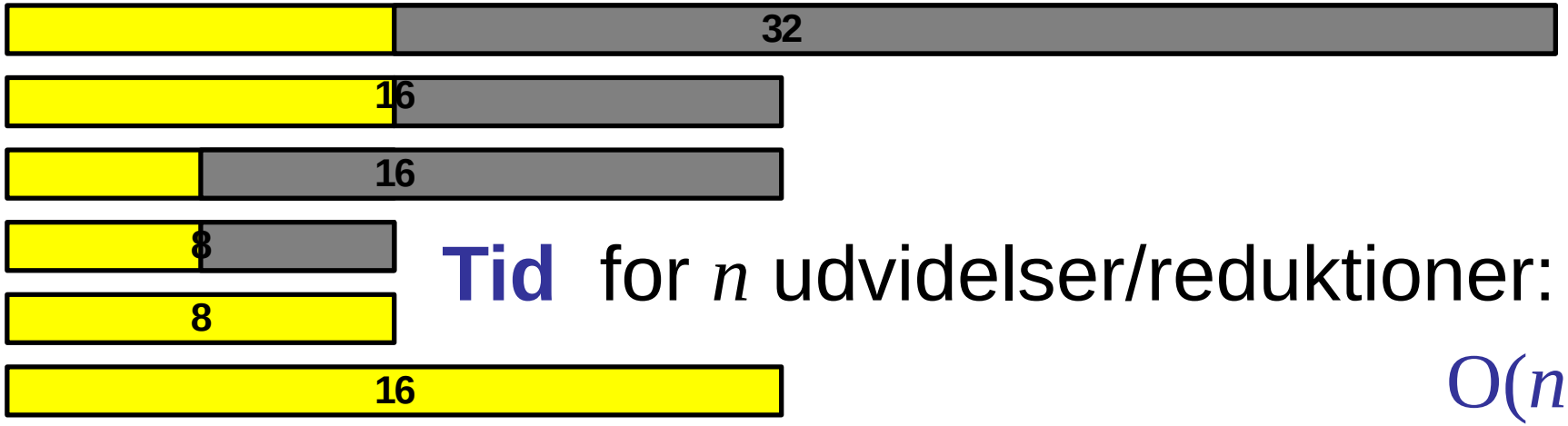
Fordoble arrayet når det er fuld



Tid for n udvidelser:

$1+2+4+\cdots+n/2+n = O(n)$

Halver arrayet når det er <1/4 fyldt



Tid for n udvidelser/reduktioner:

$O(n)$

Array Fordobling + Halvering

– en generel teknik

Tid for n udvidelser/reduktioner er $O(n)$

Plads $\leq 4 \cdot$ aktuelle antal elementer

Array implementation af Stak:

n push og pop operationer tager $O(n)$ tid

Ikke pensum

Reallokeringsstrategier i Java, Python og C++

Sprog	Java ArrayList (JDK SE 12.0.1)	Python list (CPython 3.7)	C++ vector (GCC 9.1)
Ved overløb	**+ 50 %**	**+ 12.5 %**	**+ 100 %**
Formindskelse	Nej (kan kalde trimToSize)	< 50 %	Nej (kan kalde shrink_to_fit)
Dokumentation	https://docs.oracle.com/javase/10/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/
Kildekode	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\ArrayList.java fra C:\Program Files\Java\jdk-12.0.1\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 int newCapacity(int minCapacity) {
    // overflow-conscious code
    int oldCapacity = elementData.length;
    int newCapacity = oldCapacity + (oldCapacity >> 1);
    if (newCapacity - minCapacity <= 0) {
        if (elementData == DEFAULTCAPACITY_EMPTY_ELEMENTDATA)
            return Math.max(DEFAULT_CAPACITY, minCapacity);
        if (minCapacity < 0) // overflow
            throw new OutOfMemoryError();
        return minCapacity;
    }
    return (newCapacity - MAX_ARRAY_SIZE <= 0)
        ? newCapacity
        : hugeCapacity(minCapacity);
}
```

Python list

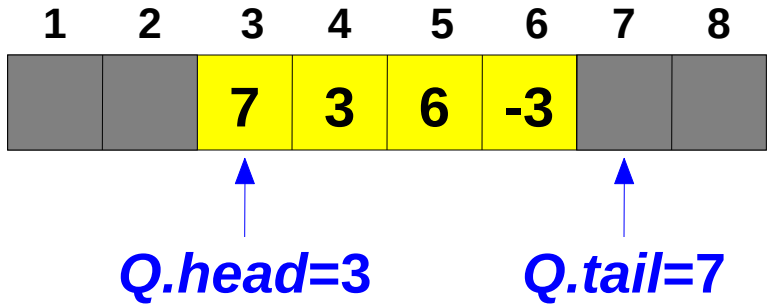
```
static int
list_resize(PyListObject *self, Py_ssize_t newsize)
{
    PyObject **items;
    size_t new_allocated, num_allocated_bytes;
    Py_ssize_t allocated = self->allocated;
    if (allocated >= newsize && newsize >= (allocated >> 1)) {
        assert(self->ob_item != NULL || newsize == 0);
        Py_SIZE(self) = newsize;
        return 0;
    }
    new_allocated =
      (size_t)newsize + (newsize >> 3) + (newsize < 9 ? 3 : 6);
    …
}
```

C++ vector

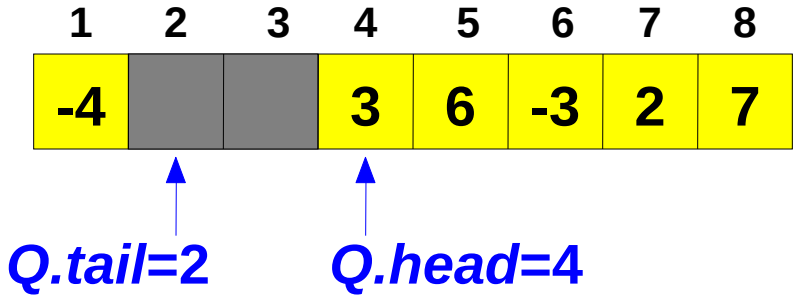
```
size_type
_M_check_len(size_type __n, const char* __s) const
{
    if (max_size() - size() < __n)
        __throw_length_error(__N(__s));

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

Kø

Kø : Array Implementation



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

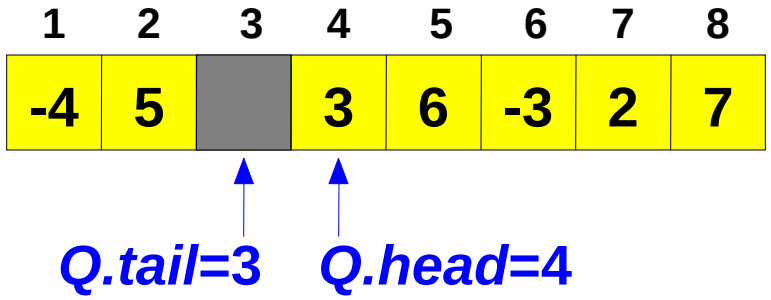



```
ENQUEUE(Q, x)
1  Q[Q.tail] = x
2  if Q.tail == Q.length
3       Q.tail = 1
4  else Q.tail = Q.tail + 1
```

```
DEQUEUE(Q)
1  x = Q[Q.head]
2  if Q.head == Q.length
3       Q.head = 1
4  else Q.head = Q.head + 1
5  return x
```

Enqueue, dequeue : $O(1)$ tid

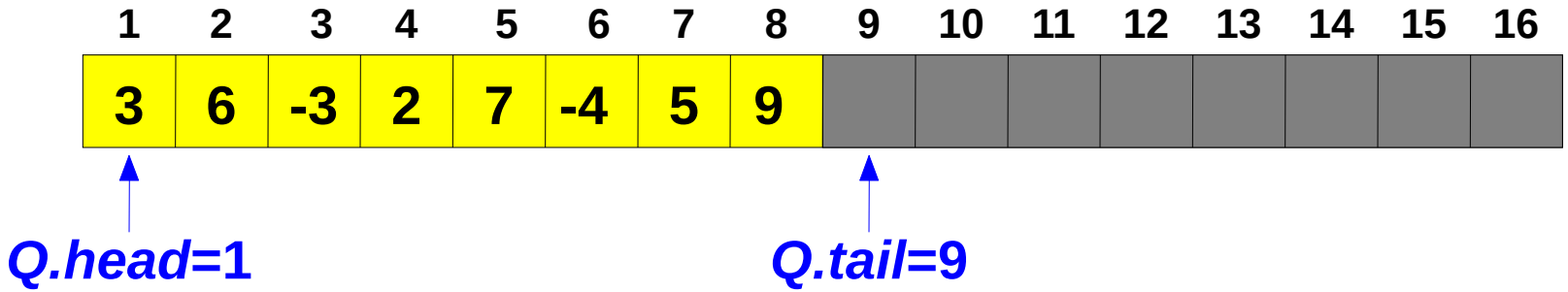
Kø : Array Implementation



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

Overløb : array fordobling/ halvering

Enqueue(9)



Array implementation af Kø:
n enqueue og dequeue operationer tager $O(n)$ tid

Arrays (med Fordobling/Halvering)

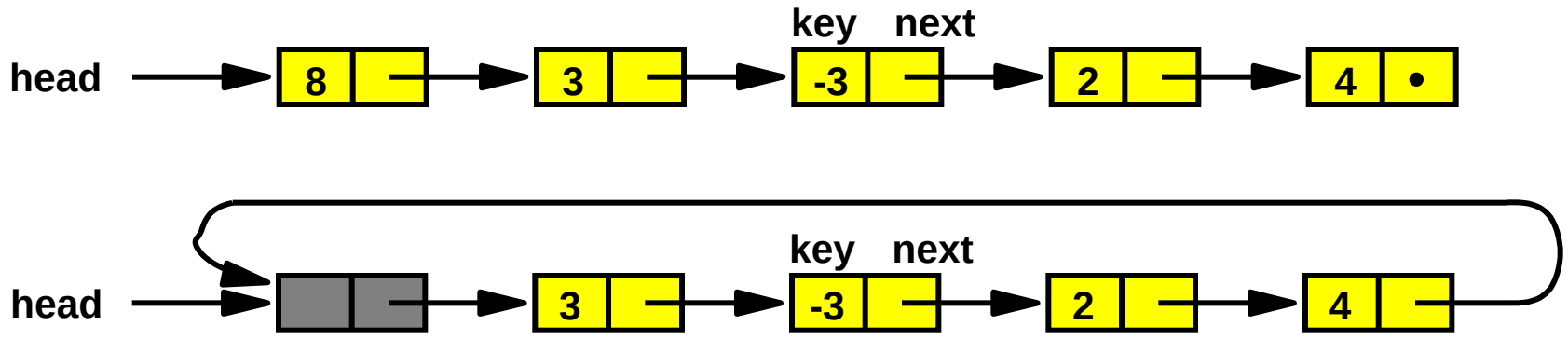
Stak	**Push(S, x)**	$O(1)$*
	Pop(S)	$O(1)$*
Kø	**Enqueue(S, x)**	$O(1)$*
	Dequeue(S)	$O(1)$*

* **Worst-case uden fordobling/halvering**
  Amortiseret ([CLRS, Kap. 16]) med fordobling/halvering

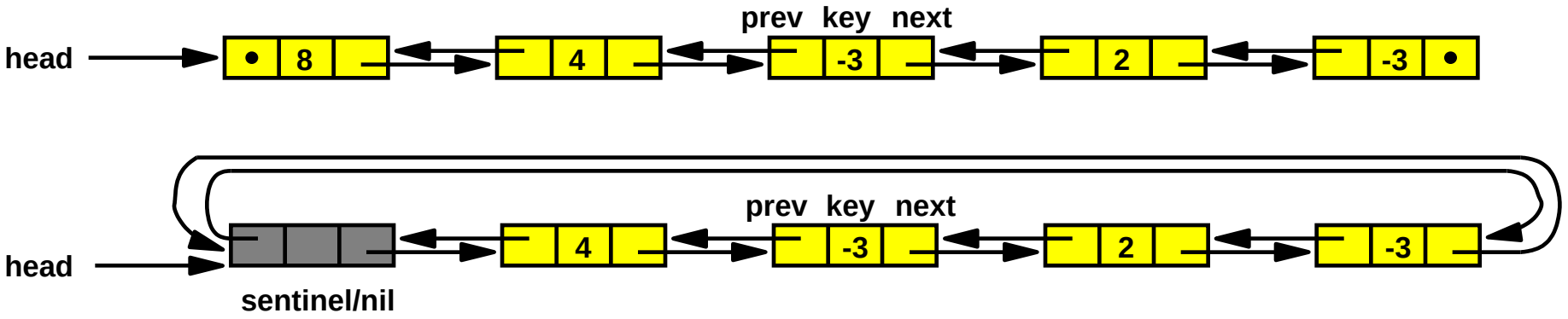
Kædede lister

Kædede Lister

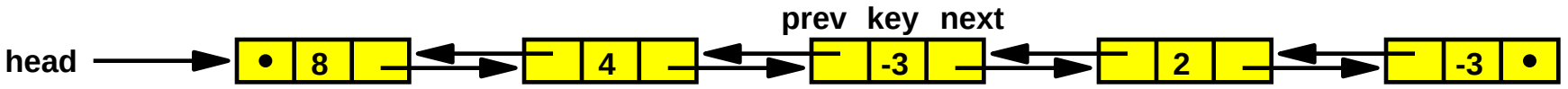
Enkelt kædede (ikke-cyklisk og cyklisk)



Dobbelt kædede (ikke-cyklisk og cyklisk)



Dobbelt Kædede Lister



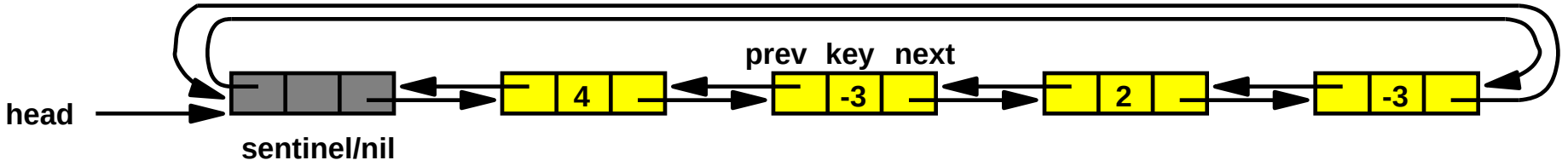

```
LIST-SEARCH(L, k)
1  x = L.head
2  while x ≠ NIL and x.key ≠ k
3      x = x.next
4  return x
```

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

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

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

Dobbelt Kædede Cykliske Lister



LIST-SEARCH′(L, k)

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

LIST-INSERT′(L, x)

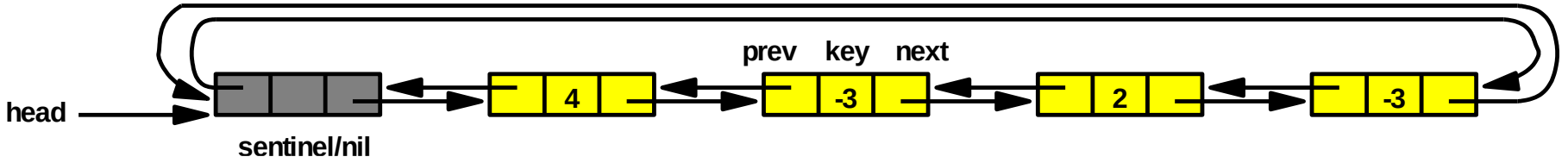
```
1  x.next = L.nil.next
2  L.nil.next.prev = x
3  L.nil.next = x
4  x.prev = L.nil
```

LIST-DELETE′(L, x)

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

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

Dobbelt Kædede Cykliske Lister



Stak	**Push(S, x)**	O(1)
	Pop(S)	O(1)
Kø	**Enqueue(S, x)**	O(1)
	Dequeue(S)	O(1)

Dancing Links

Donald E. Knuth, Stanford University

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

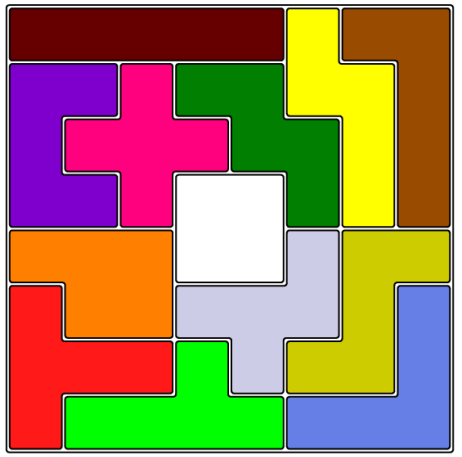
$$L\big[R[x]\big] \leftarrow L[x], \qquad R\big[L[x]\big] \leftarrow R[x] \tag{1}$$

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

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

will put x back into the list again.

Donald E. Knuth (1938-)



”The Challenge Puzzle”

?

”The Challenge Puzzle”

```
L := Tomt bræt
B := Alle brikker
Solve(L,B)

procedure Solve(Delløsning L, Brikker B)
  for alle b i B
    for alle orienteringer af b  (* max 8 forskellige *)
      if b kan placeres i nederste venstre fri then
        fjern b fra B
        indsæt b i L
        if |B| = 0 then
          rapporter L er en løsning
        else
          Solve(L,B)
        fi
        slet b fra L
        genindsæt b i B
      fi
```

Før

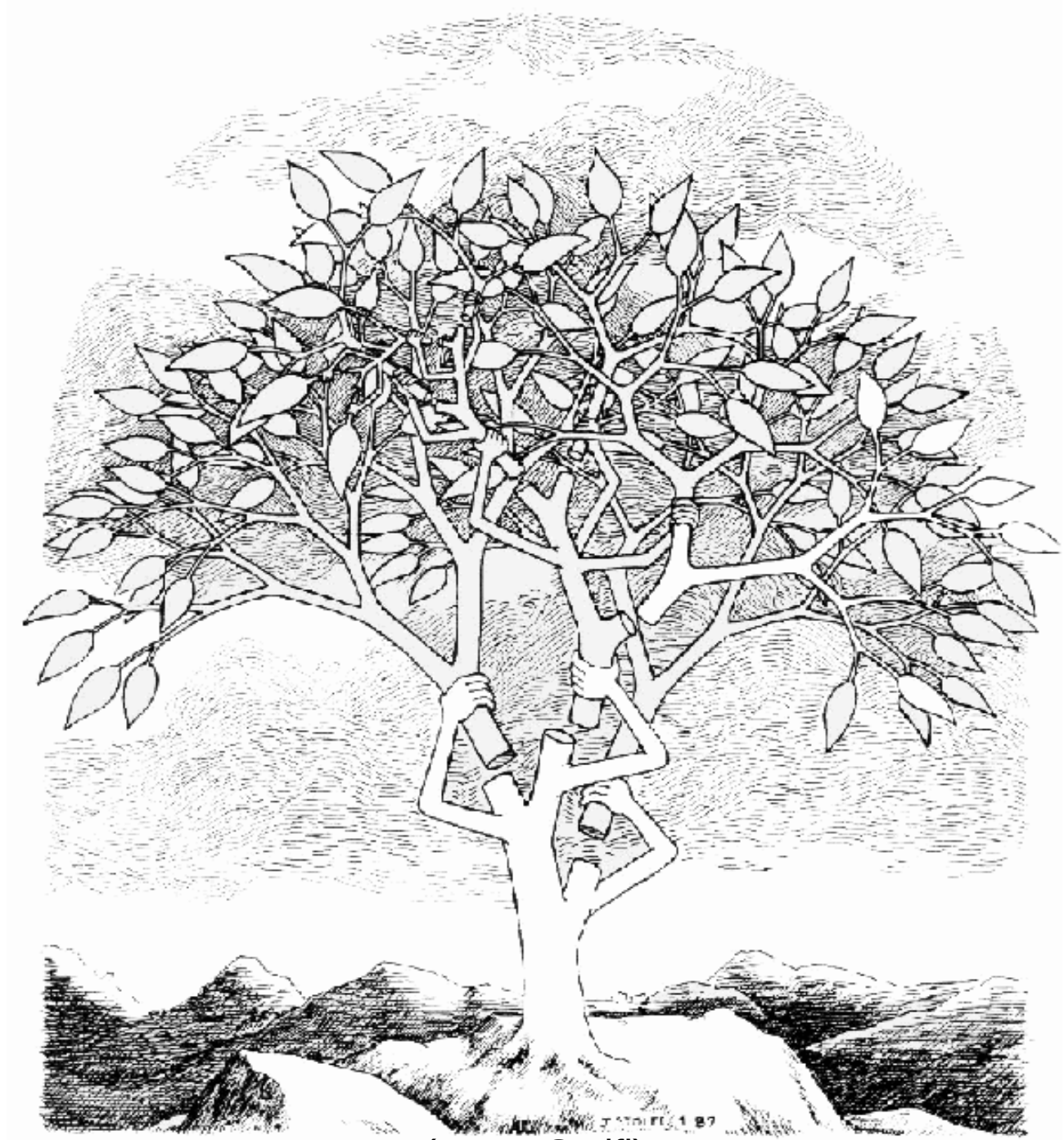
Efter

”The Challenge Puzzle”

4.040 løsninger

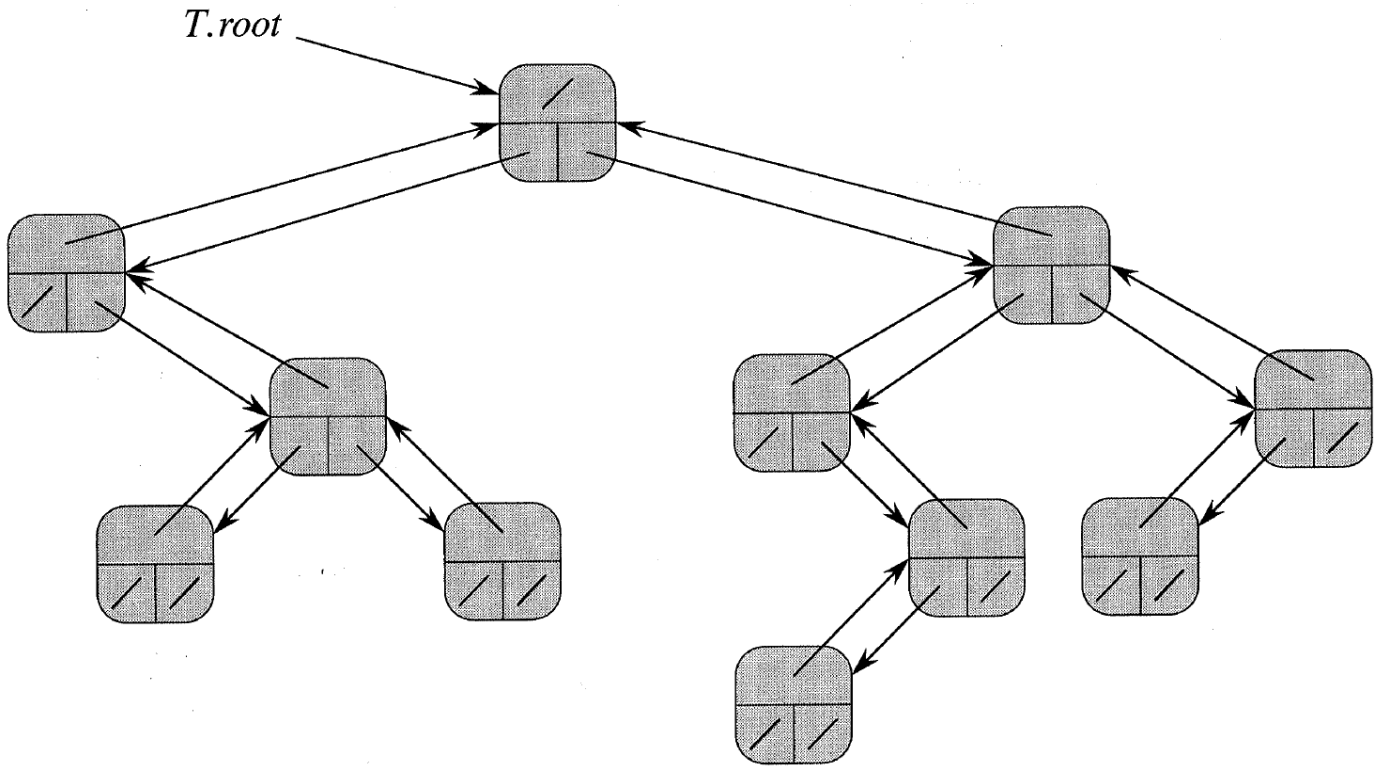
Solve placerer

8.387.259 brikker



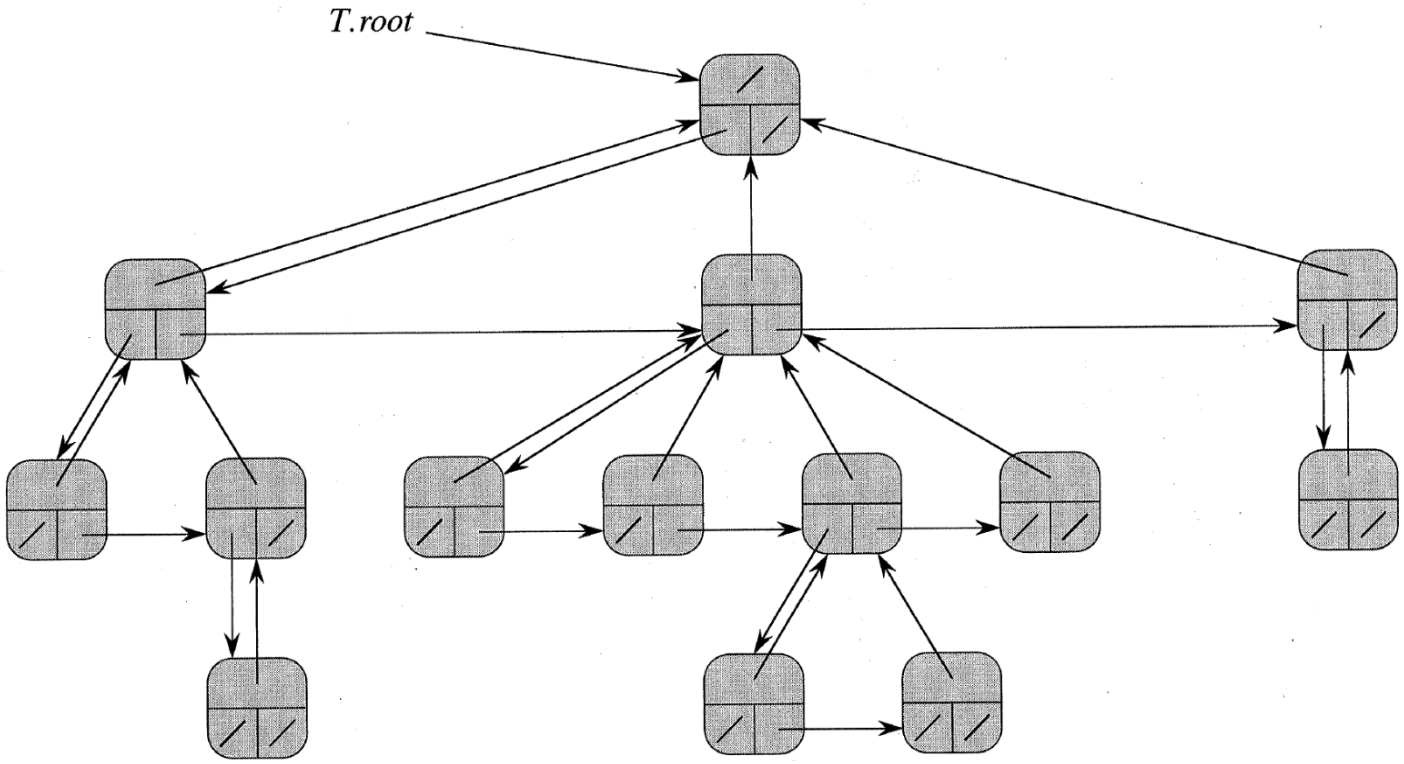
(Jorge Stolfi)

Binær Træ Repræsentation



Felter: **Left**, **right**, **parent**

Træ Repræsentation



Felter: **Left-child**, **right-sibling**, **parent**

Algoritmer og Datastrukturer

Hashing
[CLRS, kapitel 11.1-11.4]

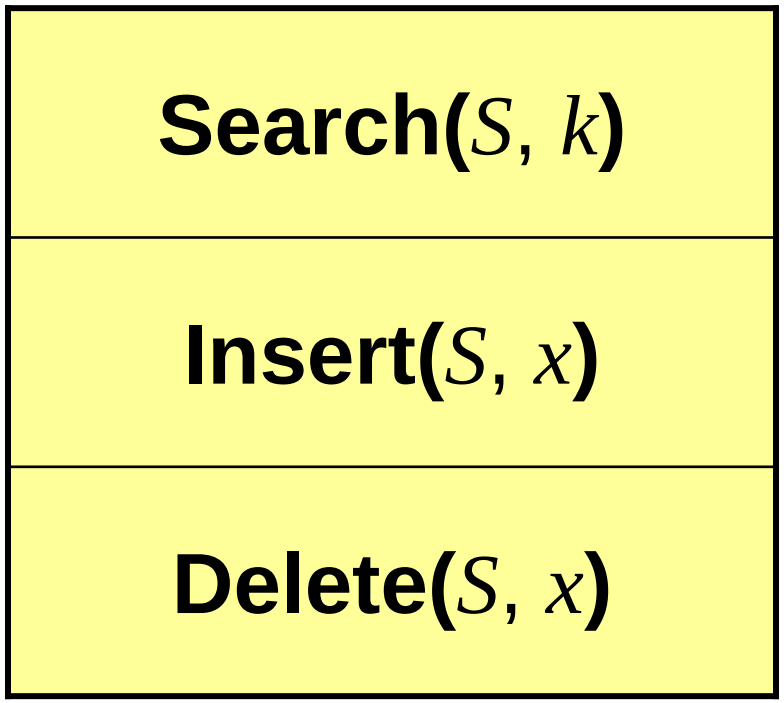
hash (Engelsk-Dansk)

1. (sb) (ret med kød og kartofler) biksemad (*fx* a meat and potato hash); *(fig.)* kludder; noget værre rod;
¤ *make a ~ of* forkludre; udføre på en elendig (el. kikset) måde; *settle somebody's ~* ordne nogen; få nogen ned med nakken;

2. (sb) (narko) hash (*fx* smoke hash);

3. (vb) hakke; skære i stykker; *(fig.)* forkludre; slippe (rigtigt) dårligt fra;
¤ *~ over* diskutere; drøfte (*fx* we can hash it over later); *~ up* forkludre; slippe dårligt fra.

Abstrakt Datastruktur: Ordbog



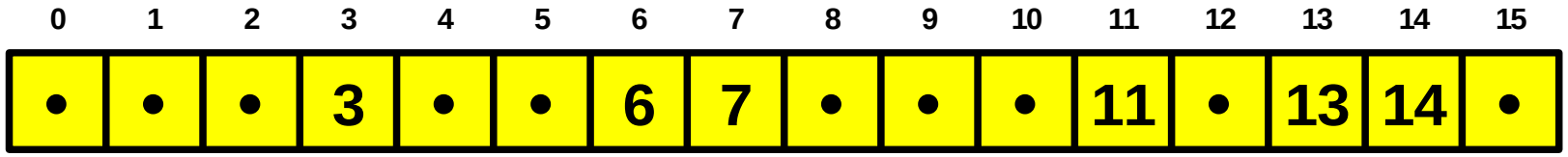
Kan vi udnytte at x for alle praktiske formål er en **sekvens af bits?** **JA !**

Alle nøgler er tal...

”Sko” = $01010011.01101011.01101111_2 = 5466991_{10}$

	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

Direkte Adressering :
Nøgler $\{0,1,2,\ldots,m\text{-}1\}$




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

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

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

\+ Godt ved små nøgle universer
\+ Selv kan generere nøglerne som 1,2,3,...
– Stort plads overforbrug når kun få nøgler brugt

Hash Funktion

- Nøgler U
- **Hash funktion**

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

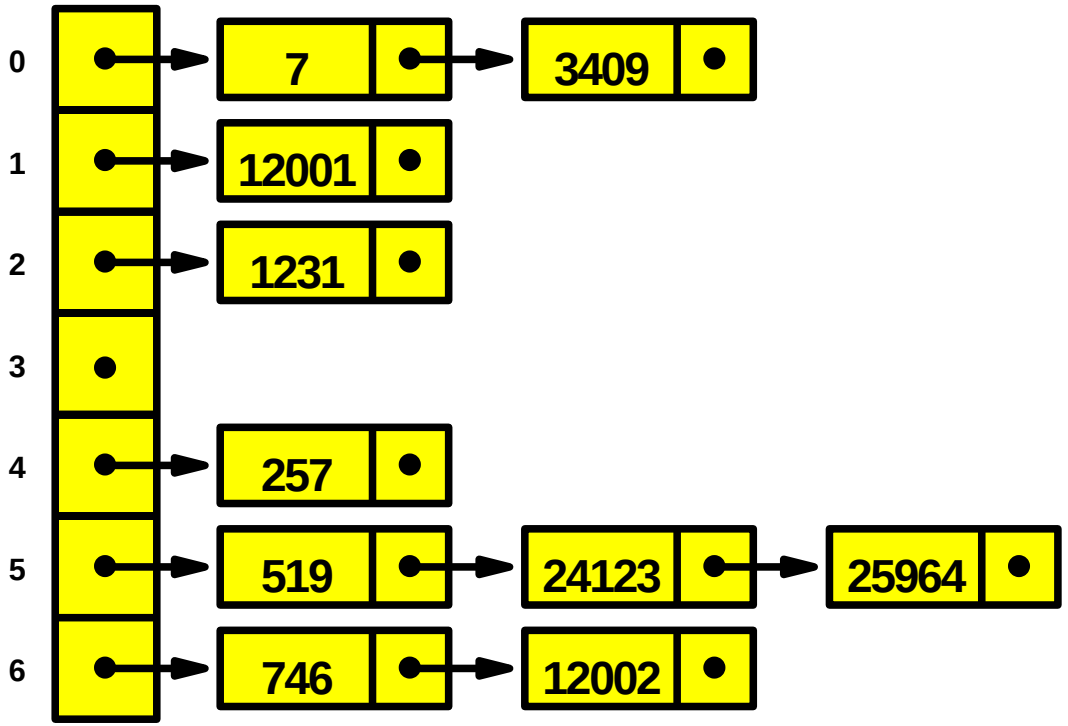
$$m << |U|$$

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

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

\+ Nemt at jævne nøglerne jævnt ud
– Flere nøgler kan hashes til samme værdi
– Næsten ens nøgler kan være vilkårligt spredt

Hash Tabel : Kollisionslister

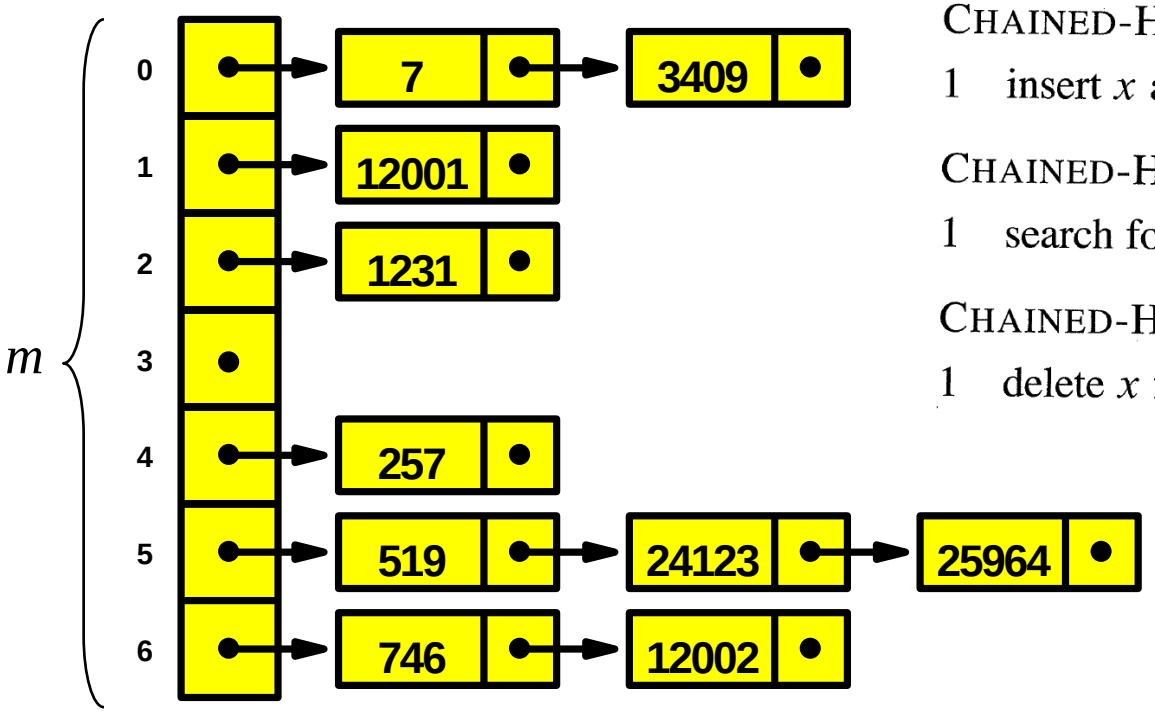


x	$h(x)$
7	0
257	4
519	5
746	6
1231	2
3409	0
12001	1
12002	6
24123	5
25964	5

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

- Gem mængde af nøgler K
- Vælg **tilfældig** hash funktion $h : U \rightarrow \{0,1,..., m\text{-}1\}$
- Gem nøglerne i tabel efter **hash værdi**
- **Kollisionslister** til nøgler med samme hashværdi

Hash Tabel : Kollisionslister



CHAINED-HASH-INSERT(T, x)

1 insert x at the head of list $T[h(x.key)]$

CHAINED-HASH-SEARCH(T, k)

1 search for an element with key k in list $T[h(k)]$

CHAINED-HASH-DELETE(T, x)

1 delete x from the list $T[h(x.key)]$

Vælg en uniform tilfældig hash funktion:

- Forventet antal nøgler i en indgang i tabellen $|K|/m$
- Insert, Delete, Search **forventet tid** $O(|K|/m)$, dvs. tid $O(1)$ hvis $m = \Omega(|K|)$

Hvad er en god Hash Funktion?

– For enhver funktion findes en dårlig mængde nøgler der hasher til samme værdi

\+ For enhver mængde nøgler findes en god hash funktion der jævner godt ud (om den kan beskrives kompakt er et andet spørgsmål)

Mål Find en lille **mængde af hash funktioner** hvor en tilfældig funktion virker rimelig godt på en given mængde

Hash Funktioner : Eksempler

$h(k) = k \bmod m$ **(typisk m et primtal)**

$m = 2^8$ ignorerer alt på nær de 8 sidste bit:

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

$m = 2^8\text{-}1$ ignorerer alle ombytninger af tegn:

$$h(\text{"}c_3c_2c_1\text{"}) = h(\text{"}c_1c_3c_2\text{"})$$

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

$h(0101000010101010) = 01000$

$0101000010101010 \cdot 1001111000110111$

$= 0011000111011010\underline{01000}00010000110$

Universelle Hash Funktioner

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

Definer $p \cdot (p\text{-}1)$ hash funktioner $h_{a,b}$, hvor $1 \leq a < p$ og $0 \leq b < p$

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

Sætning

For to nøgler $x \neq y$ og en tilfældig hash funktion $h_{a,b}$ gælder

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

Korollar

For en hash tabel med en **tilfældig hash funktion** $h_{a,b}$
tager Insert, Delete, Search **forventet tid** $O(|K|/m)$

Hash Tabel : Universel Hashing

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

Hashing af tal med mange bits...

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

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

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

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

$p > 2^w$ primtal

$0 < a < p$ tilfældig

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

Polynomium evaluering $h_a(x)$

$y = 0$

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

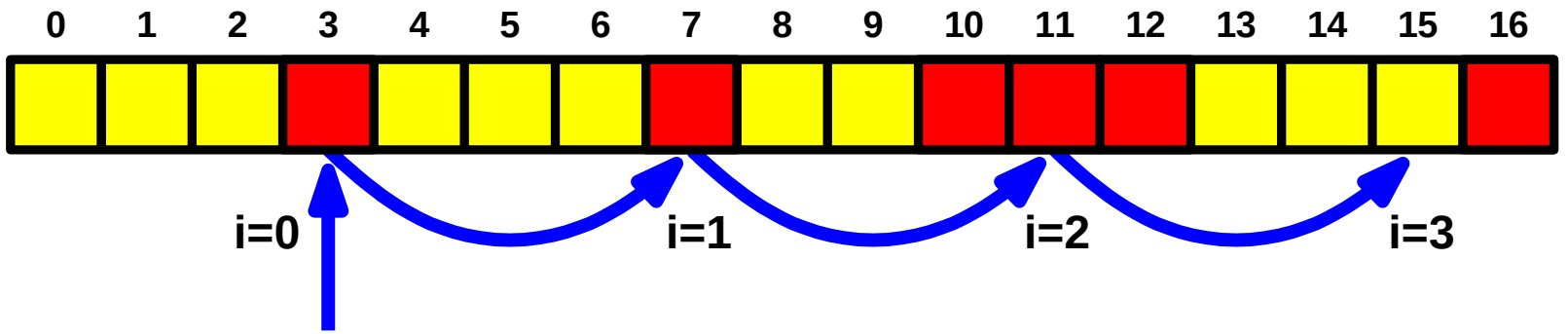
$y = (y \cdot a + x_i) \bmod p$

$h_a(x) = y$

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

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

Åben Adressering



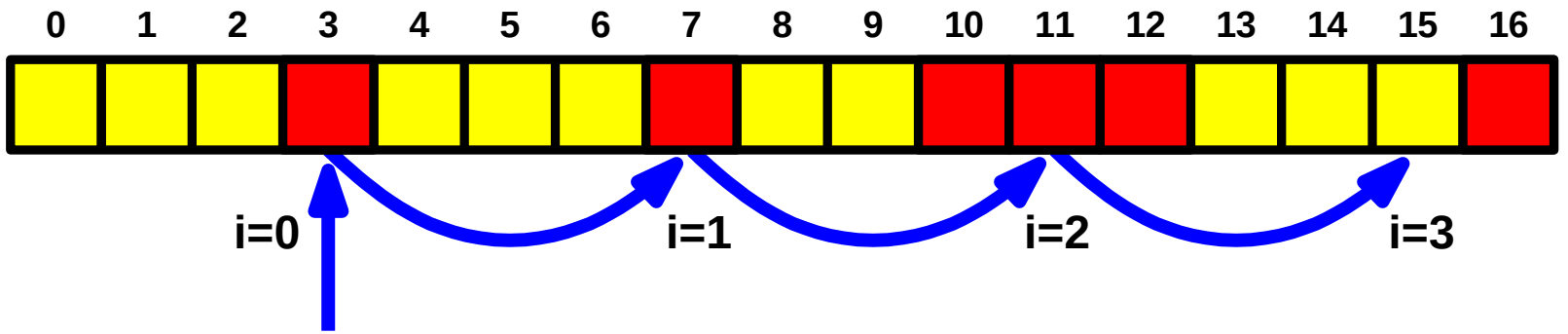
HASH-INSERT(T, k)

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

HASH-SEARCH(T, k)

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

Åben Adressering : Analyse



Uniform hashing

$h(k, 0), h(k, 1), h(k, 2)$,... er en **uniform tilfældig** rækkefølge

(urealistisk)

Sætning

Ved uniform hashing er det forvente antal lookups $1/(1-\alpha)$

hvor $\alpha = |K|/m$ er belastningsfaktoren / fyldningsgraden

Lineær Probing

Indsæt k på første ledige plads

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

for $i = 0,1,2...$

Eksempel :

Indsæt 9, 3, 20, 6, 12, 2, 19, 11, 5

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

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

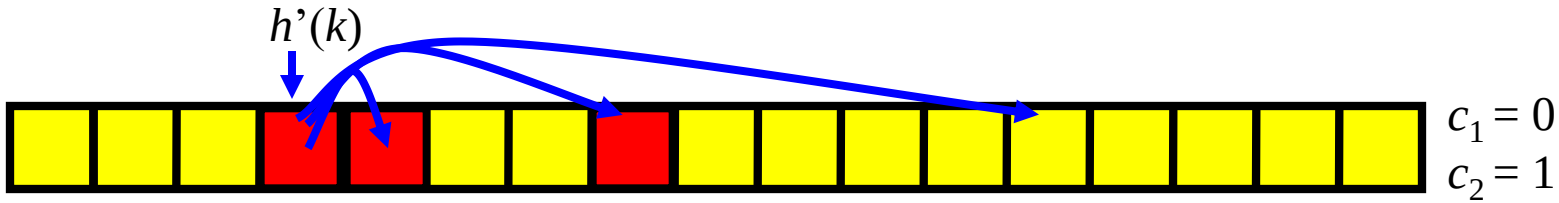
$h'(x)=(2 \cdot x) \mod 17$

Kvadratisk Probing

Indsæt k på første ledige plads

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

for $i = 0,1,2, \ldots$ hvor c_1 og $c_2 \neq 0$ er konstanter

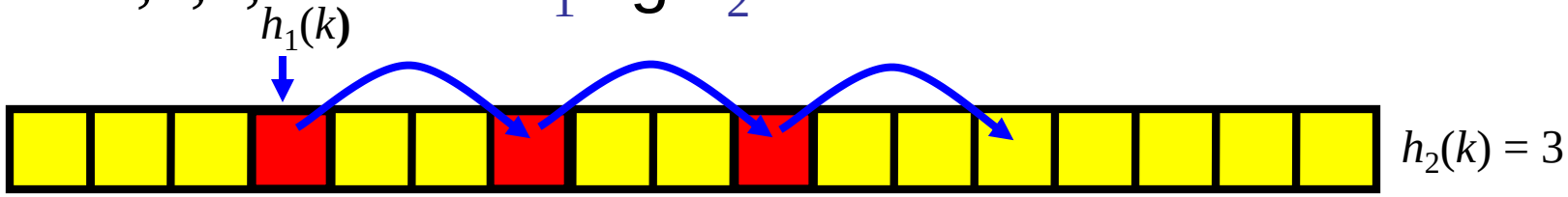


Dobbelt Hashing

Indsæt k på første ledige plads

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

for $i = 0,1,2, \ldots$ hvor h_1 og h_2 er hash funktioner

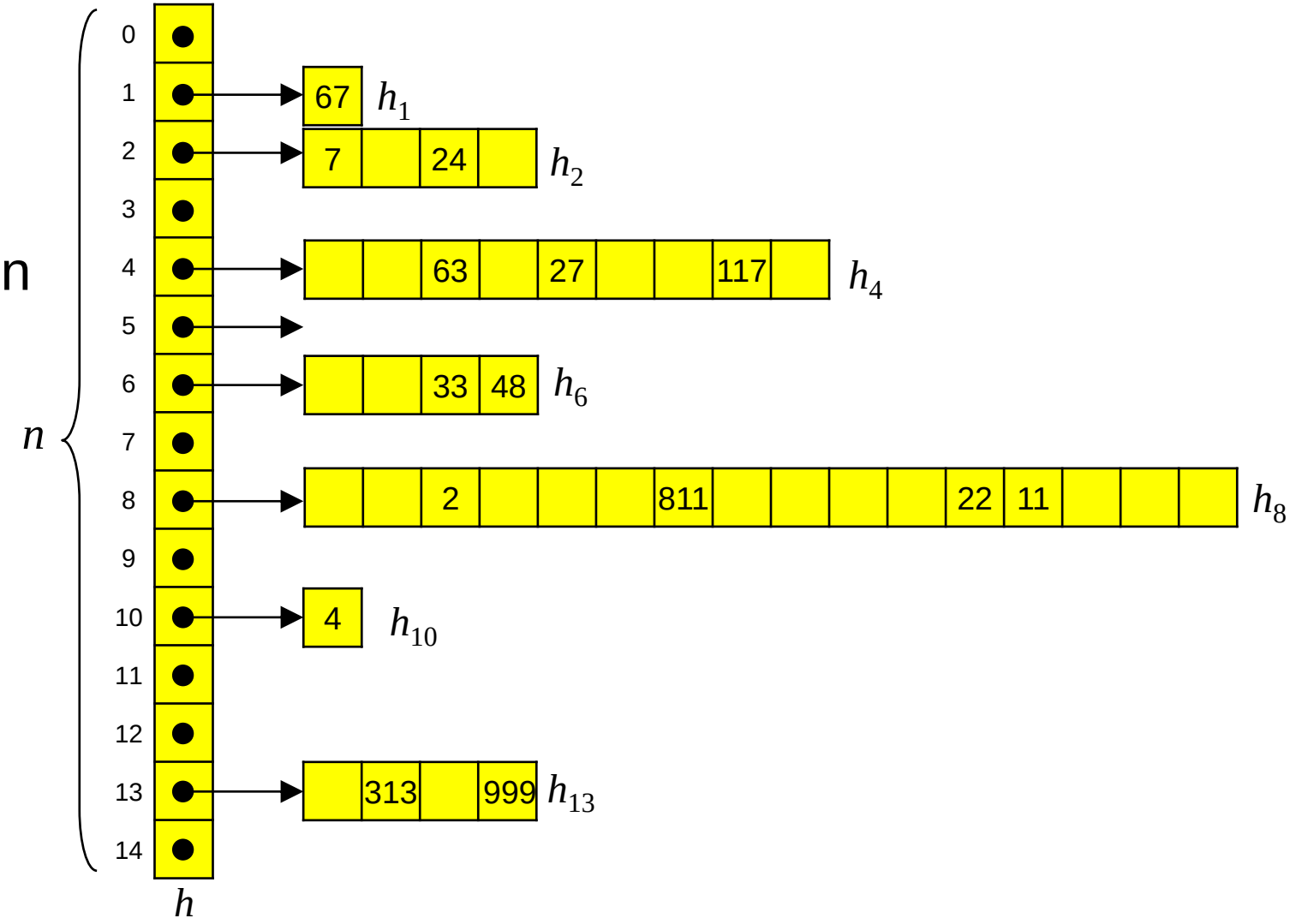


CLRS 3. udgave, Kapitel 11.5 – ikke pensum

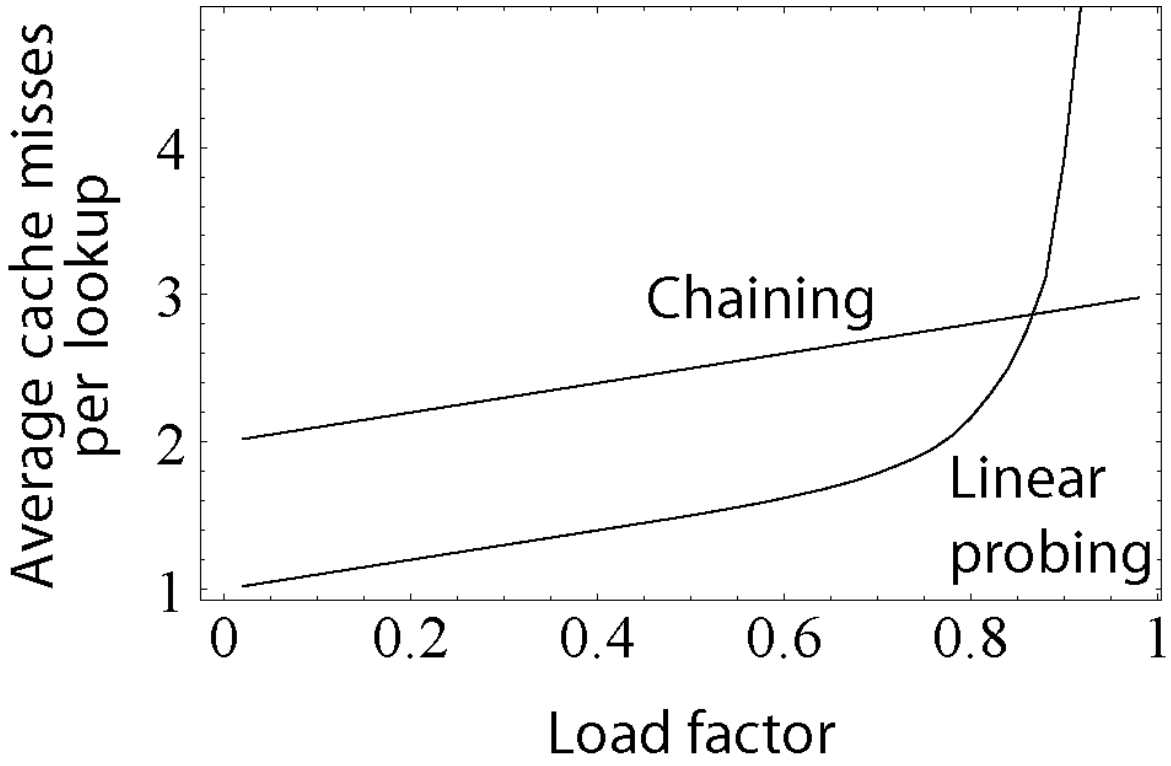
Perfekt hashing (statisk)

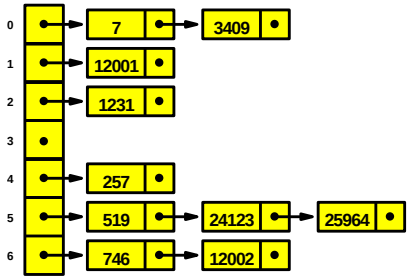
- n elementer
- 2 niveauer af hash funktioner
- hver spand egen hash funktion
- h hash tabel med n indgange
- Spand med n_i elementer størrelse n_i^2
- Universelle hash funktioner

E[kollisioner] = $\binom{n}{2}$ / # indgange



Eksperimentel Sammenligning





Chaining


Linear probing

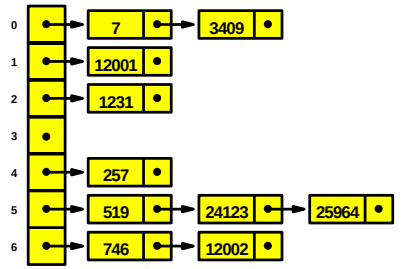
wikipedia.org

Hashing

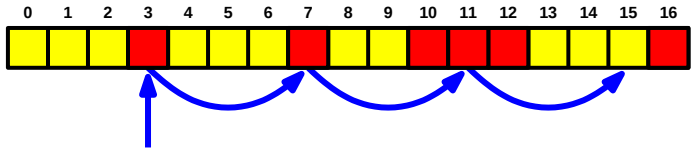
- **Valg af hash funktion**
  - **Prøv sig frem...**
  - **Universelle hash funktioner**

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

- **Hash tabeller**
  - **Kollisionslister (kædede lister)**



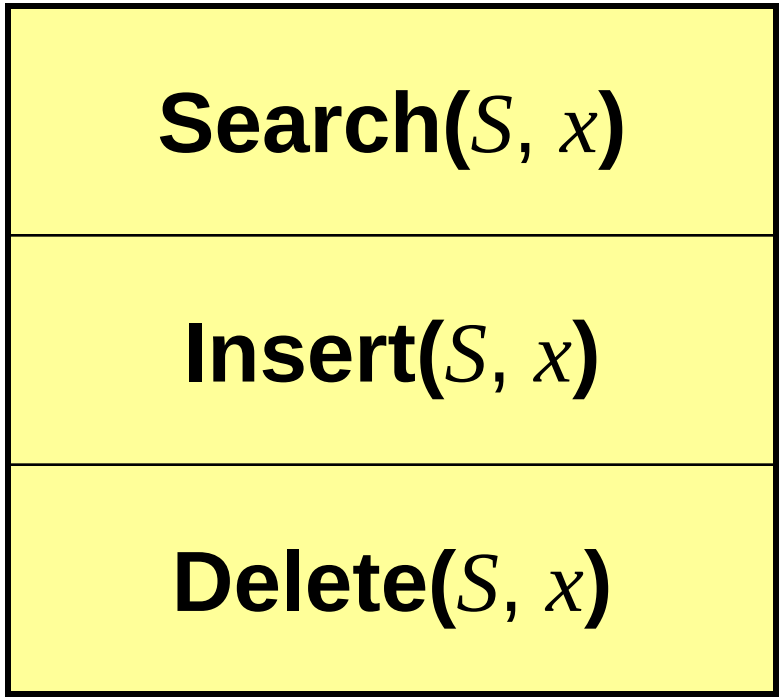
- **Åben adressering**
  - **Lineær probing**
  - **Kvadratisk probing**
  - **Dobbelt hashing**



Algoritmer og Datastrukturer

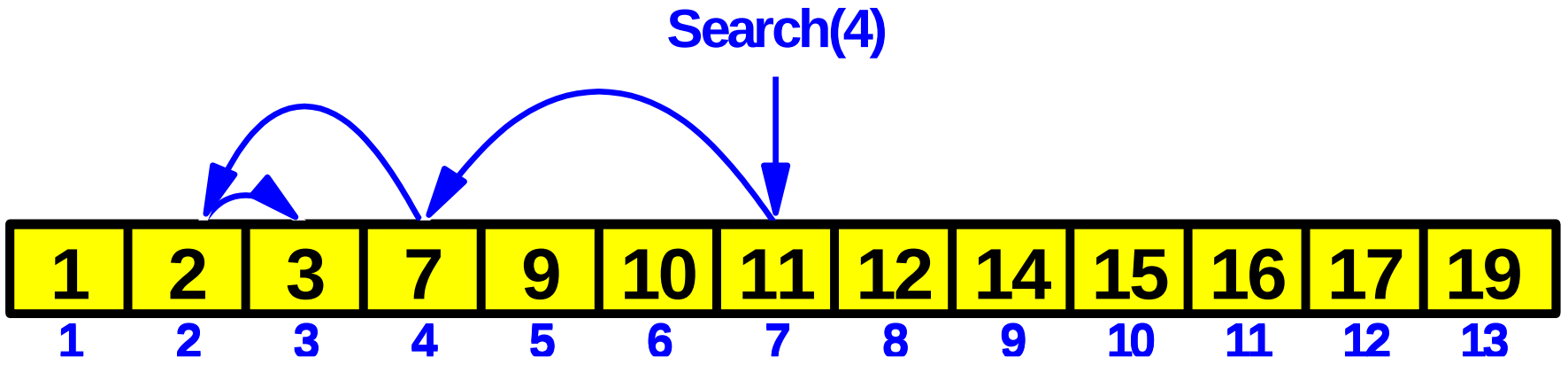
Binære søgetræer
[CLRS, kapitel 12]

Abstrakt Datastruktur: Ordbog



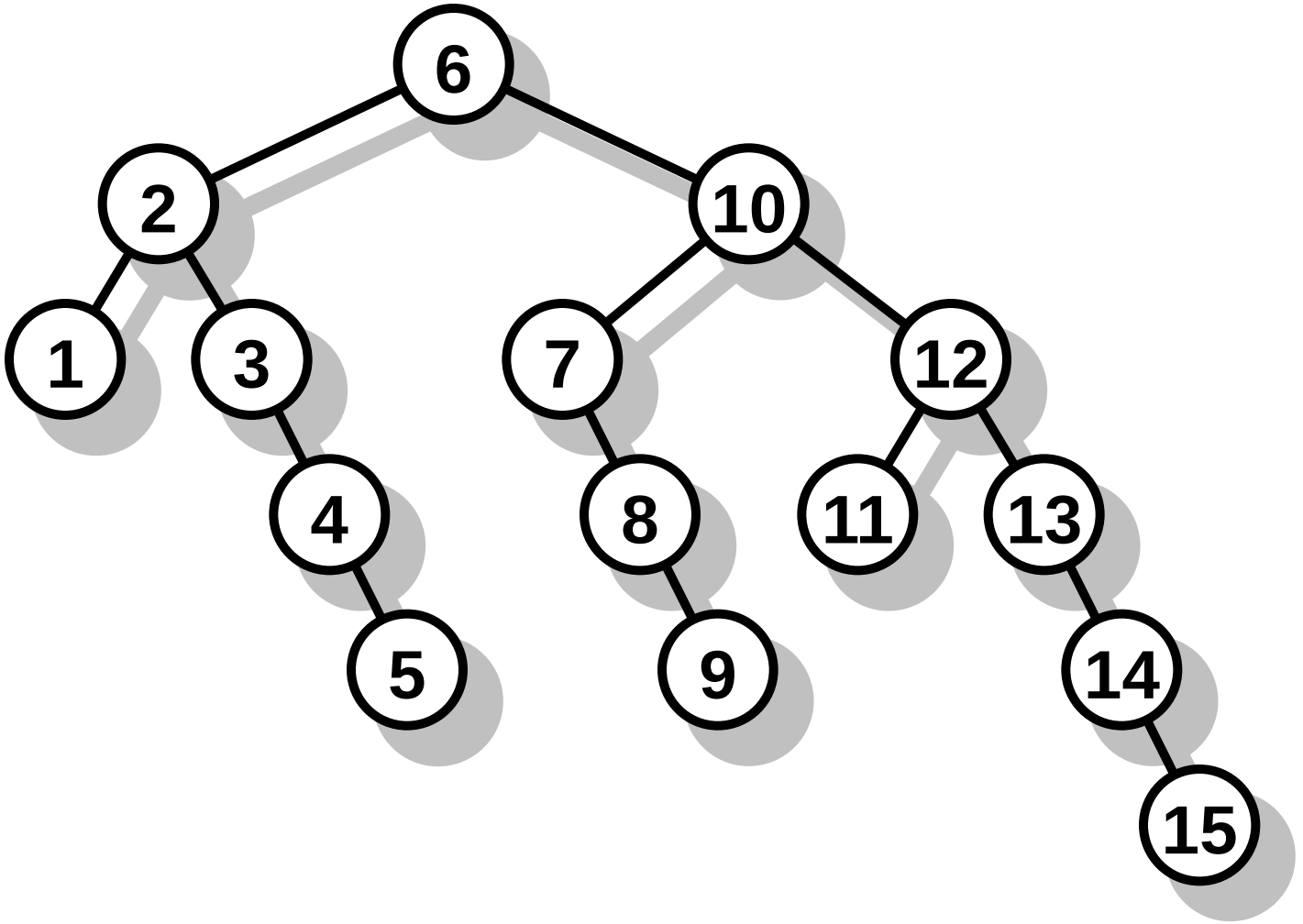
Elementer må kun sammenlignes med ”≤”

(Statisk) Ordbog : Sorteret Array



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

Søgetræ



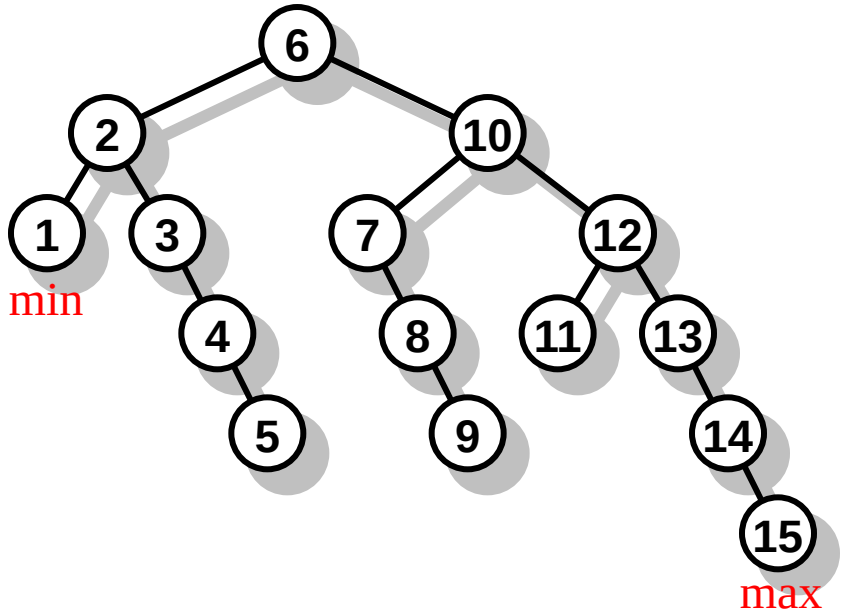
Invariant For alle knuder er elementerne i venstre (højre) undertræ mindre (større) end eller lig med elementet i knuden

Søgetræs søgninger

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

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

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




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

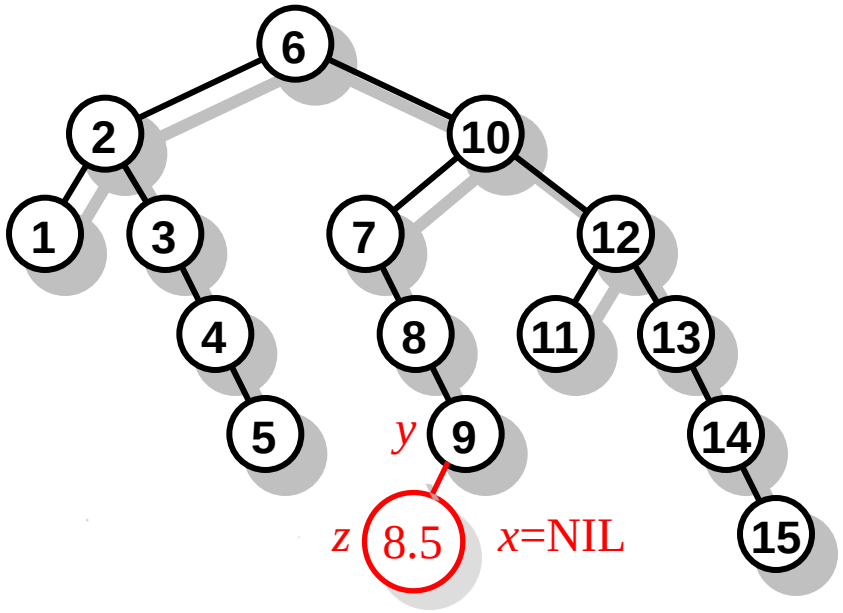
```
TREE-MINIMUM(x)
1  while x.left ≠ NIL
2        x = x.left
3  return x
```

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

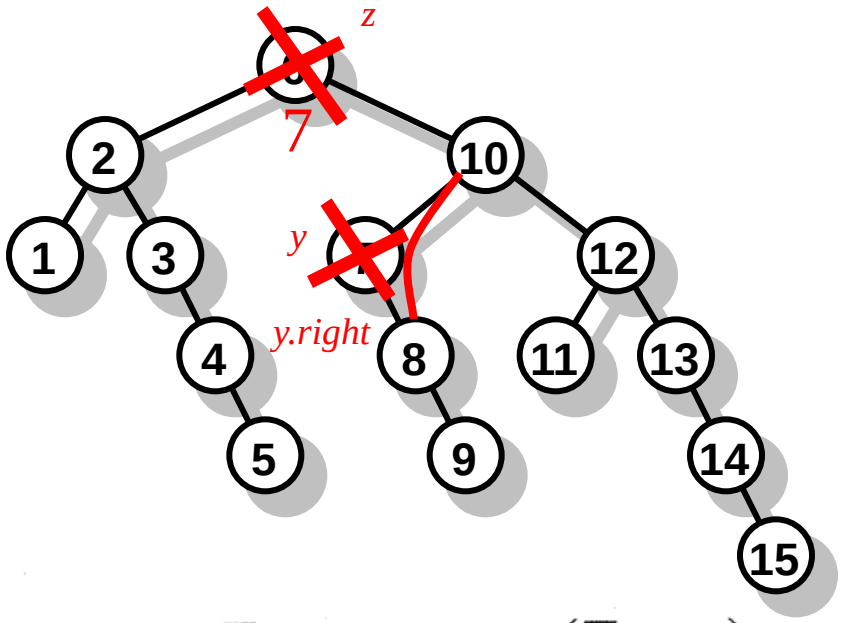
Indsættelse i Søgetræ

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

Iterativ søgning



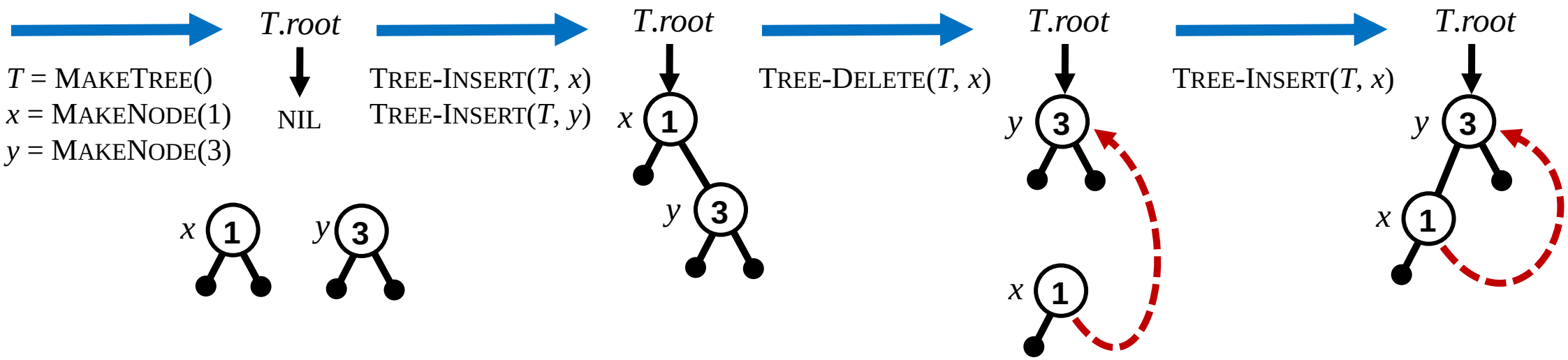
Slettelse fra Søgetræ




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

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

!!! Advarsel !!!



- CLRS pseudokoden nulstiller ikke slettede knuders pointers og antager at den indsatte knudes børn er NIL
- TREE-SEARCH(*T.root*, 2) vil gå i uendelig løkke 

Søgetræer

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

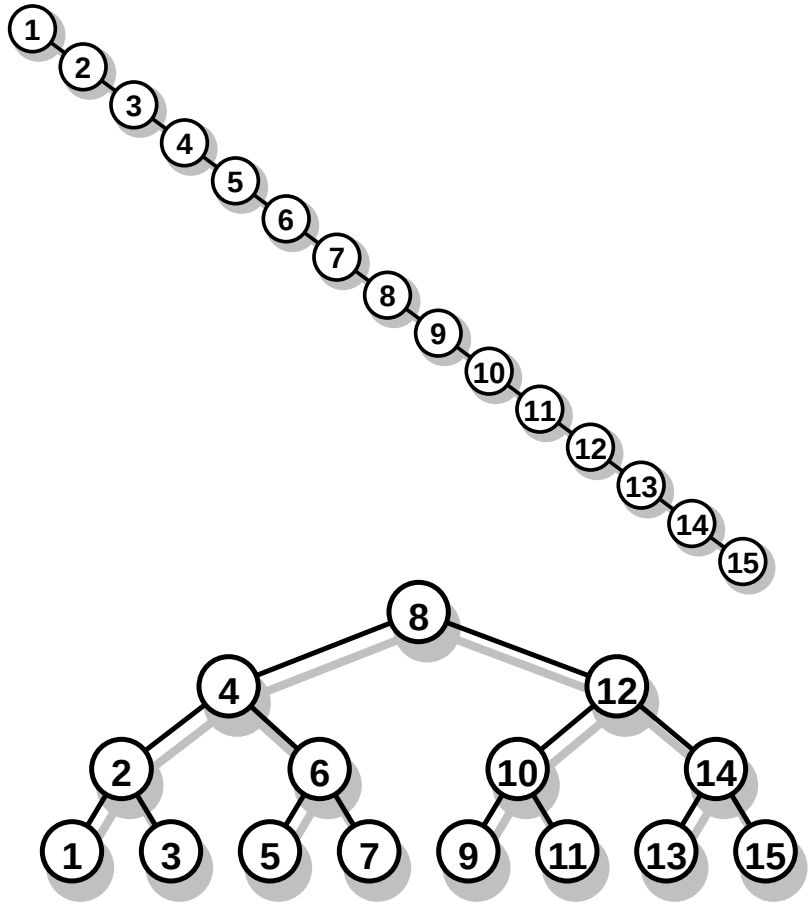
h = højden af træet, n = antal elementer

Højden af et Søgetræ ?

Største og Mindste Højde

Indsæt i stigende rækkefølge
- Højde n

Perfekt balanceret
(mindst mulig højde)
- Højde $1+\lfloor \log n \rfloor$



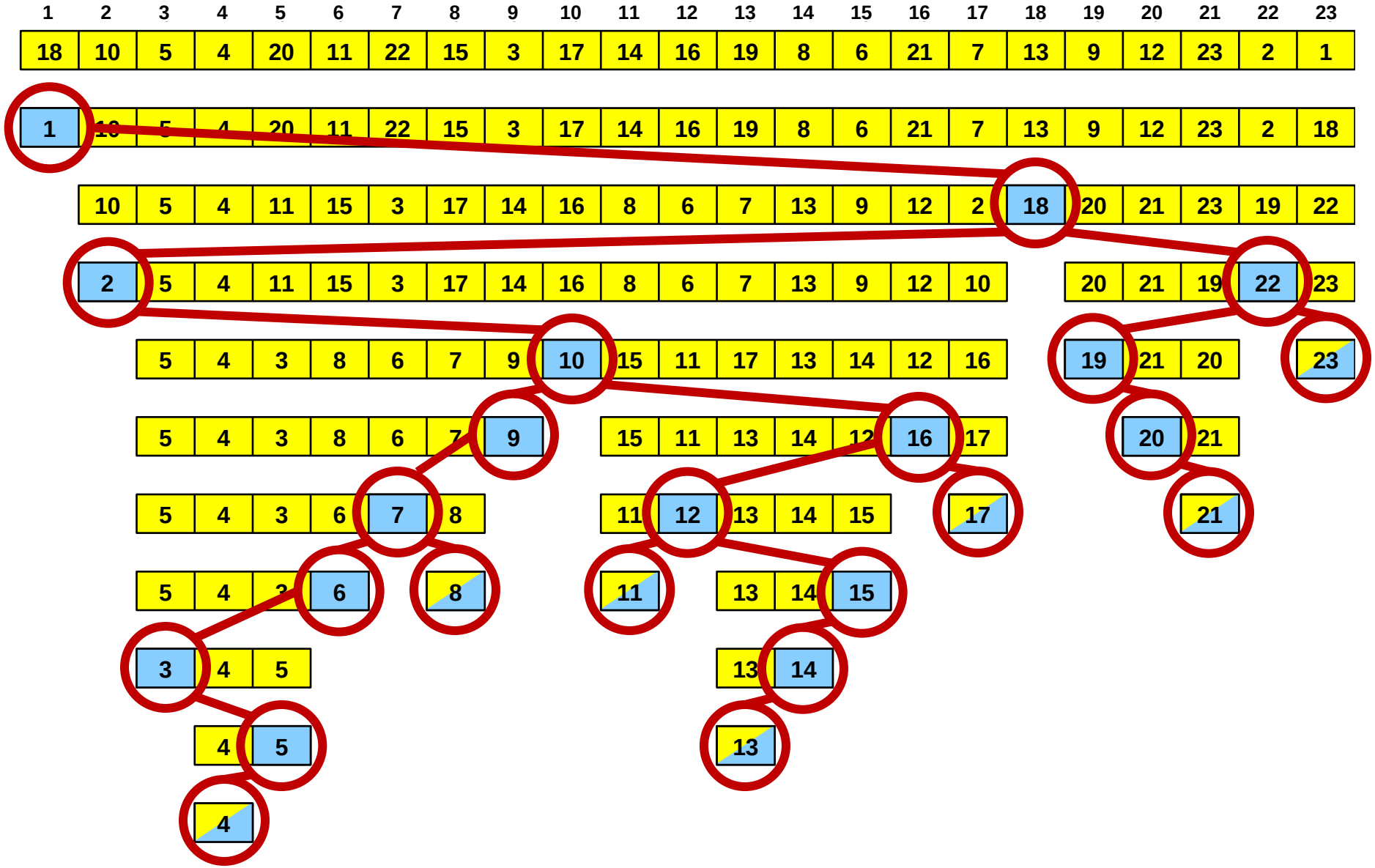
Tilfældige Indsættelser i et Søgetræ

Algoritme

Indsæt n elementer i tilfældige rækkefølge

- Ligesom ved QuickSort argumenteres at den f**orventede dybde** af et element er $O(\log n)$ [CLRS, problem 12.3]
- Den forventede **højde af træet** er $O(\log n)$, dvs. *alle knuder* har forventet dybde $O(\log n)$ [CLRS 3. udgave, kapitel 12.4]

Quicksort på 23 elementer



Balancerede Søgetræer

Ved **balancerede søgetræer** omstruktureres træet løbende så søgetiderne forbliver $O(\log n)$

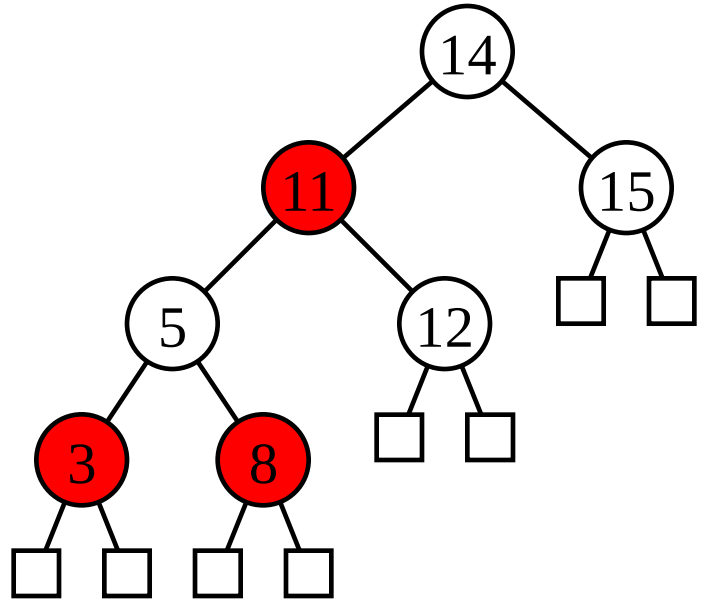
- AVL-træer [CLRS pr. 13-3]
- BB[α]-træer
- Splay træer
- Lokalt balancerede træer
- Rød-sorte træer [CLRS kap. 13]

- Randomized trees [CLRS 3.udg. pr. 13-4]
- (2,3)-træer
- (2,4)-træer [CLRS pr. 18-2]
- B-træer [CLRS kap. 18]
- Vægtbalancerede B-træer
- ...

Algoritmer og Datastrukturer

Balancerede søgetræer: Rød-sorte søgetræer
[CLRS, kapitel 13]

Rød-Sorte Søgetræer



Mål

Søgetræer med dybde $O(\log n)$

Invarianter

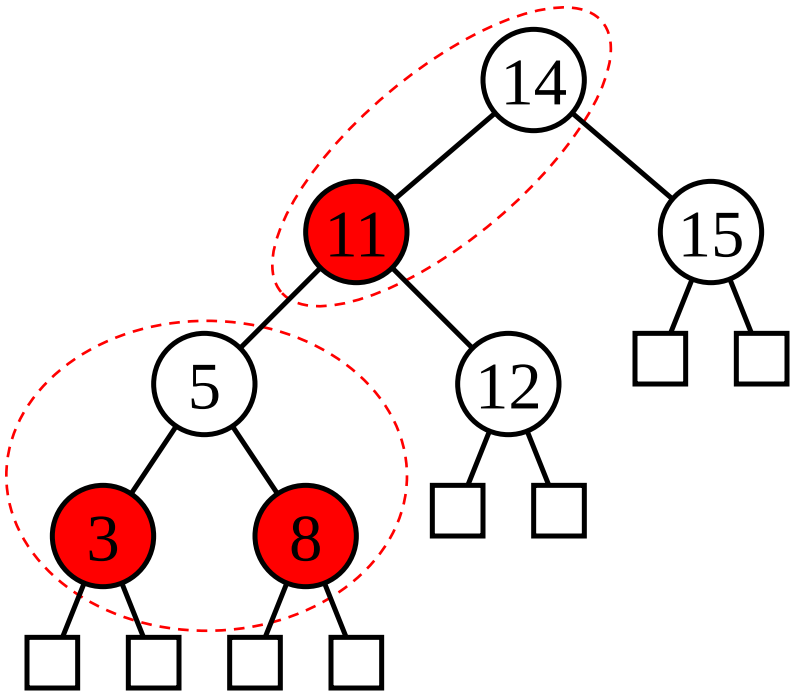
- Hver knude er enten **RØD** eller **SORT**
- Hver **RØD** knude har en **SORT** far
- Alle stier fra roden til et eksternt blad har **samme antal sorte knuder**

Sætning

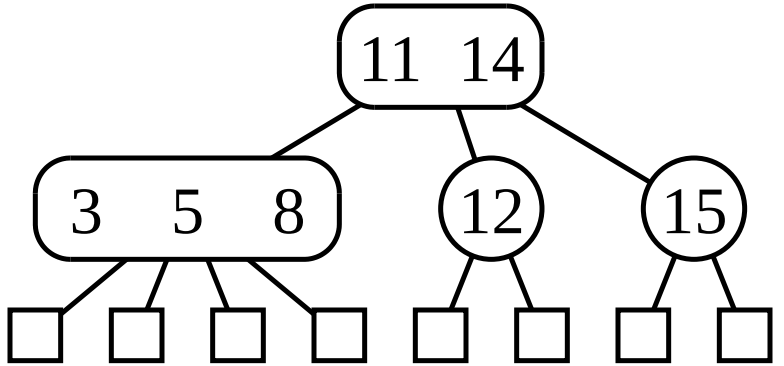
Et rød-sort træ har højde $\leq 2 \cdot \log (n+1)$

"A Dichromatic Framework for Balanced Trees", Guibas, Sedgewick, 1978

Rød-Sorte vs (2-4)-træer



Rød-sort træ



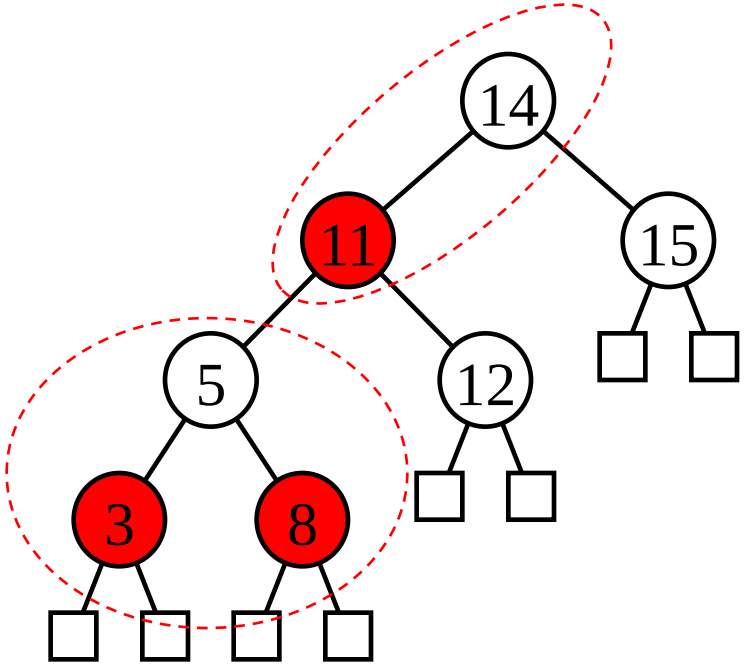
(2,4)-træ

(2,4)-træ ≡ **rød-sort træ** hvor alle røde knuder er slået sammen med faderen

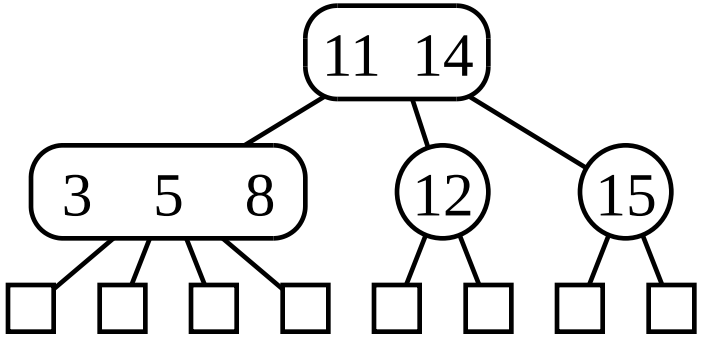
[2-4-træer = CLRS kapitel 18.1 & problem 18-2]

Egenskaber ved (2,4)-træer

- Alle interne knuder har mellem **2 og 4 børn**
- Knude med i **børn** gemmer i-1 **elementer**
- Stier fra roden til et eksternt blad er lige lange

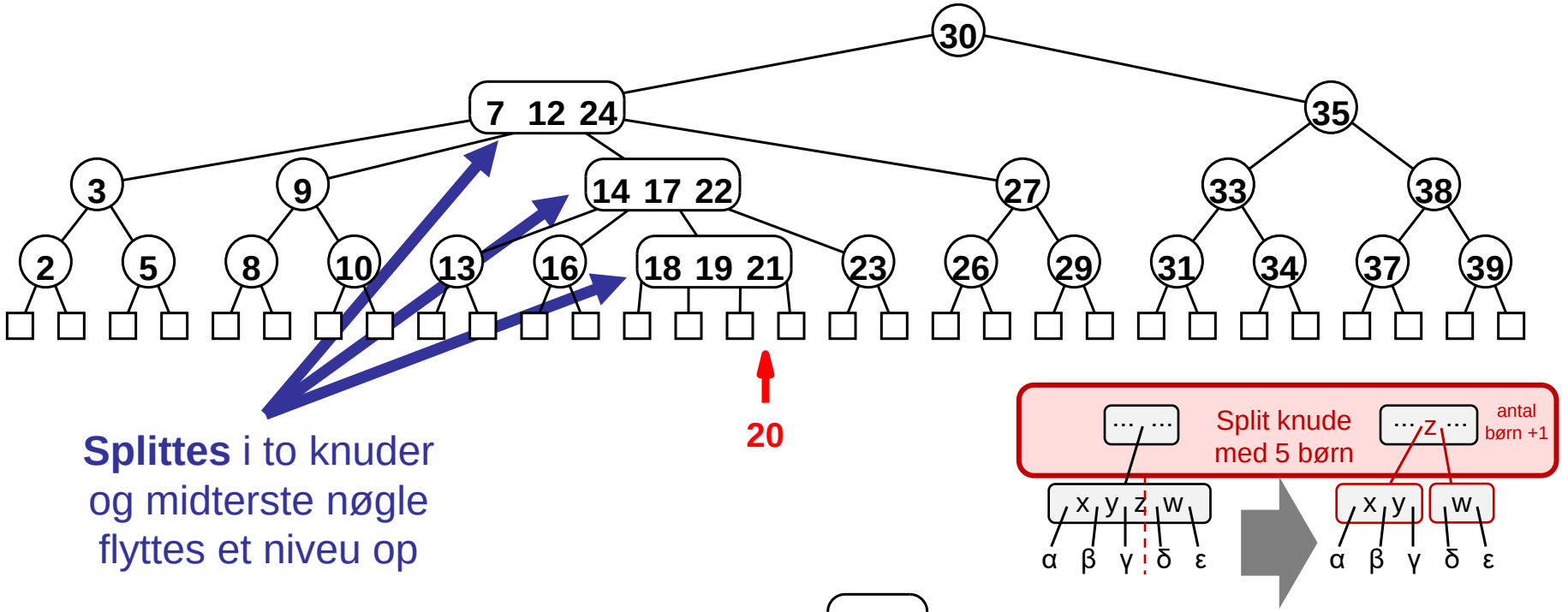


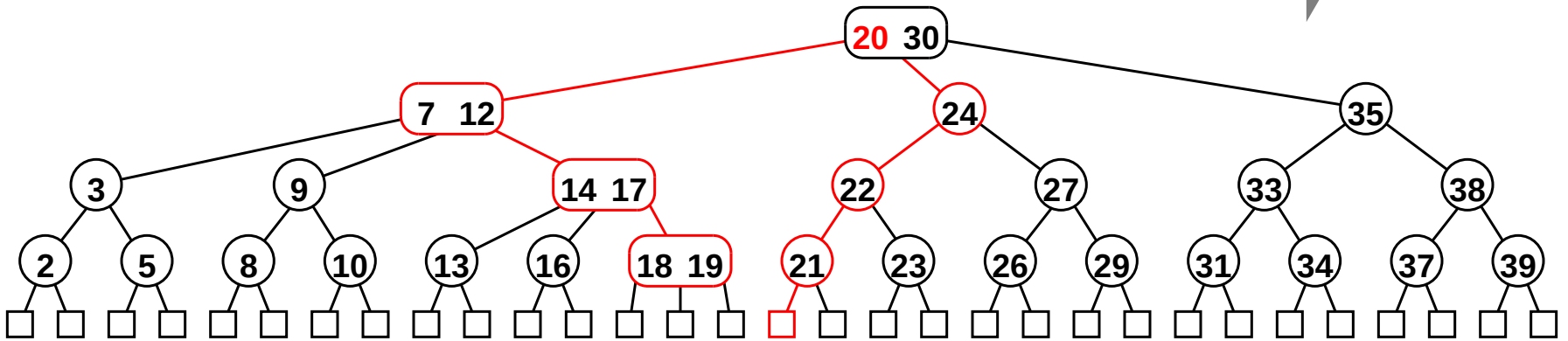
Rød-sort træ



(2,4)-træ

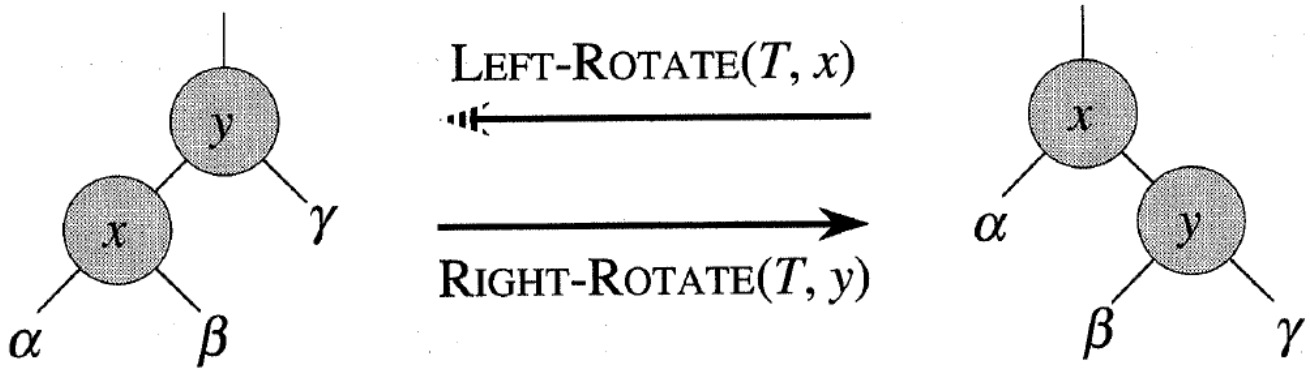
Indsættelse i et (2,4)-træ





Opdateringer af Rød-Sorte Træer

Rotationer




```
LEFT-ROTATE(T, x)
 1  y = x.right                  // set y
 2  x.right = y.left             // turn y's left subtree into x's right subtree
 3  if y.left ≠ T.nil
 4       y.left.p = x
 5  y.p = x.p                    // link x's parent to y
 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                   // put x on y's left
12  x.p = y
```

Insert

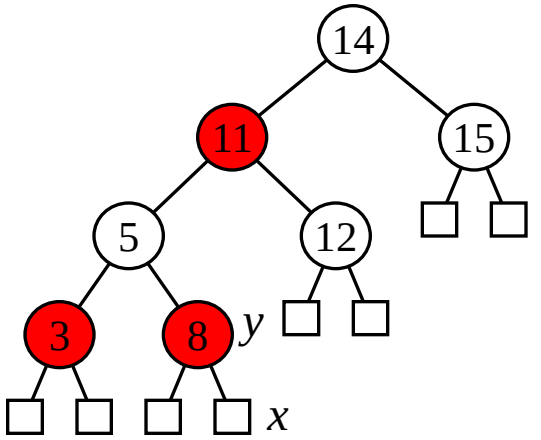
Ubalanceret indsættelse

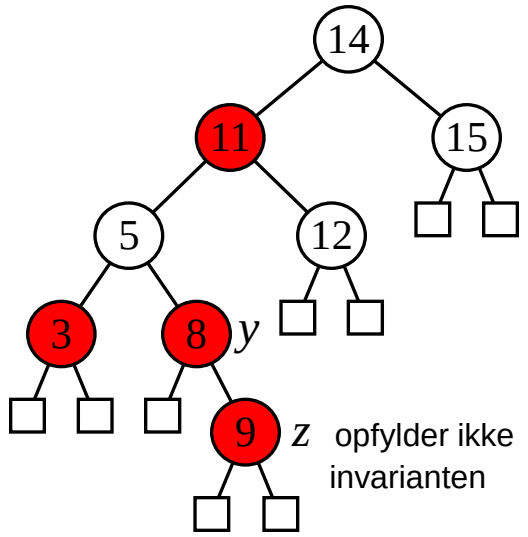
søg

opret z

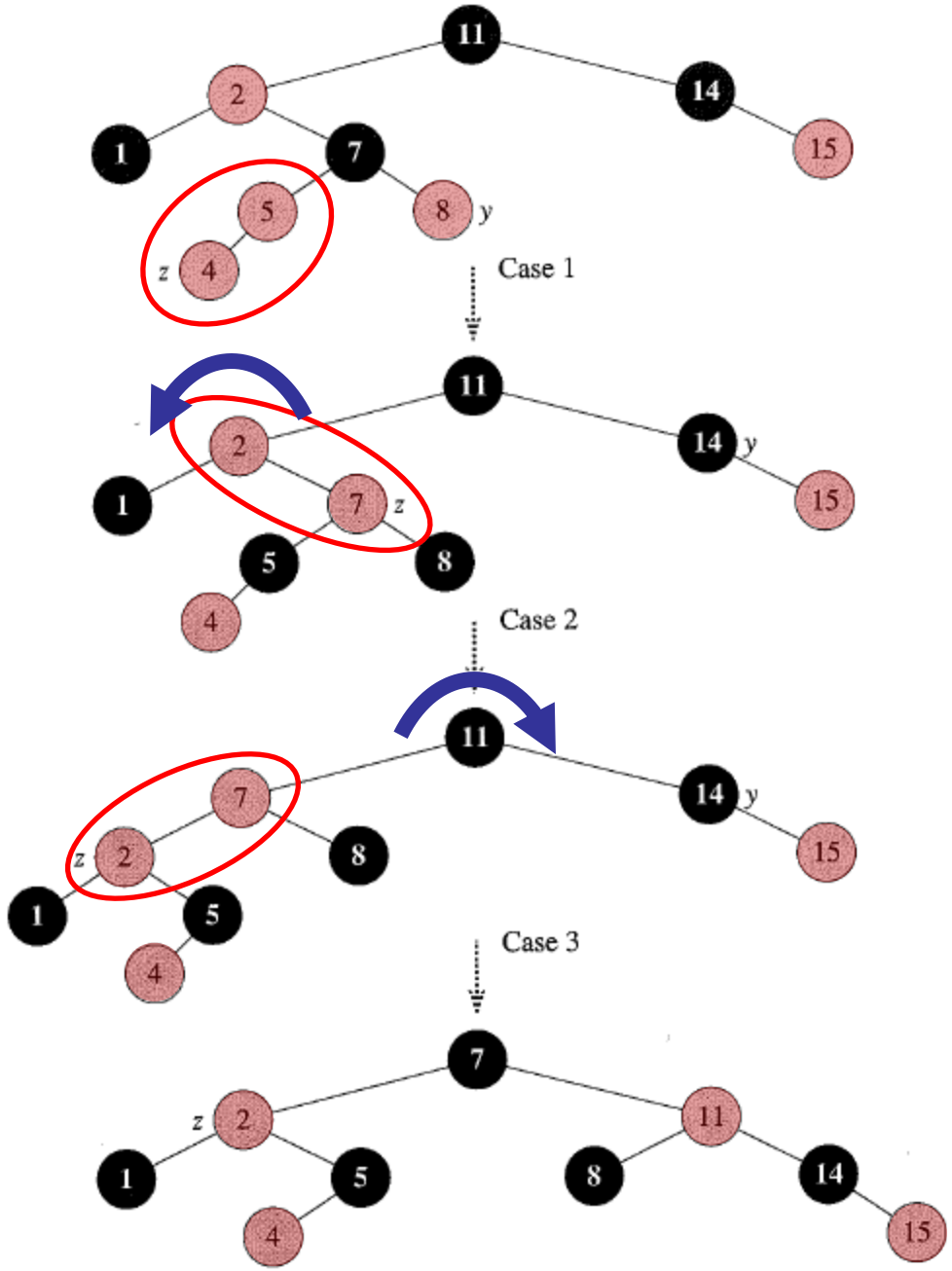
```
RB-INSERT(T, z)
 1  y = T.nil
 2  x = T.root
 3  while x ≠ T.nil
 4      y = x
 5      if z.key < x.key
 6          x = x.left
 7      else x = x.right
 8  z.p = y
 9  if y == T.nil
10      T.root = z
11  elseif z.key < y.key
12      y.left = z
13  else y.right = z
14  z.left = T.nil
15  z.right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

Insert(9)



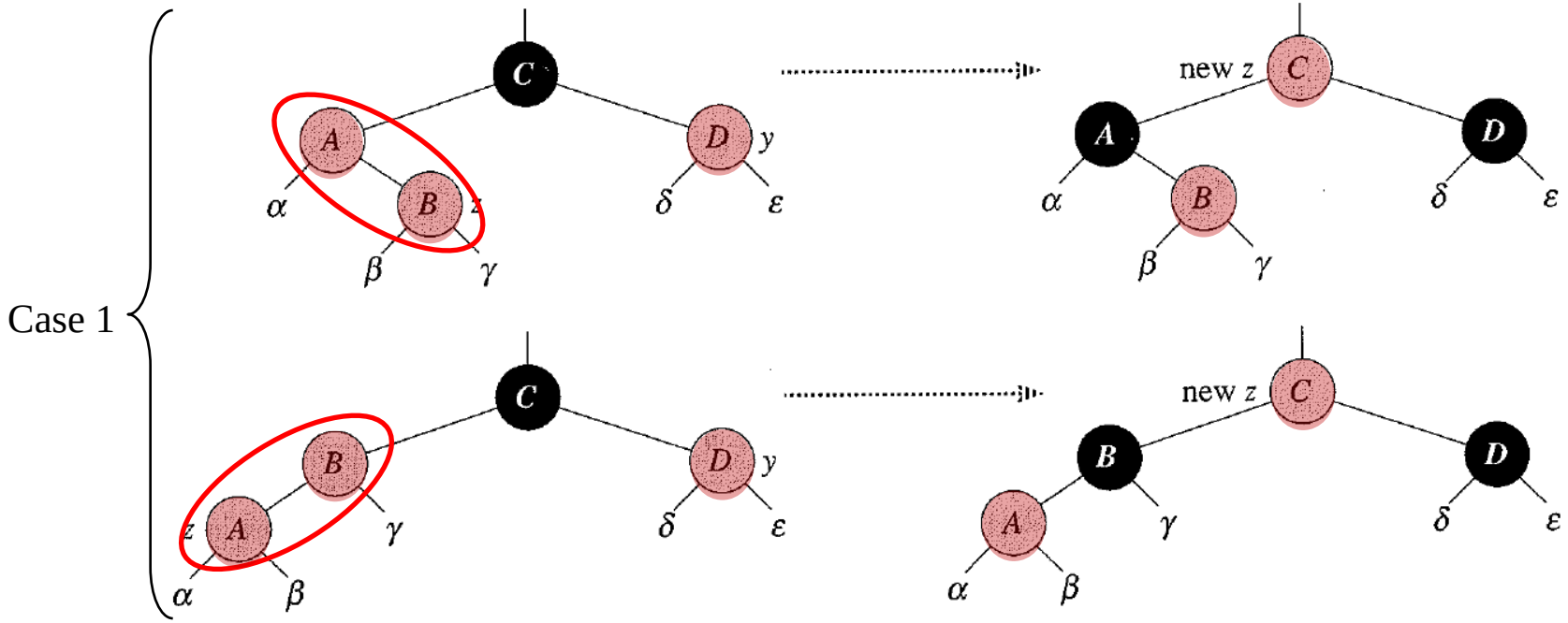


z opfylder ikke invarianten

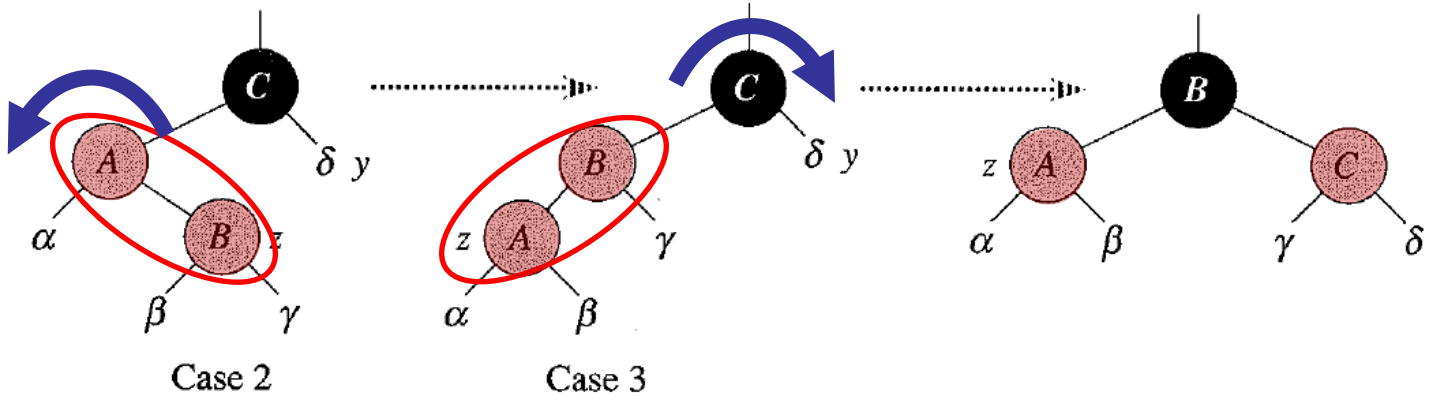

11
2
14
1
7
15
5
8
y
z
4
Case 1
11
2
14
y
1
7
z
15
5
8
4
Case 2
11
7
14
y
z
2
8
15
1
5
4
Case 3
7
z
2
11
1
5
8
14
4
15

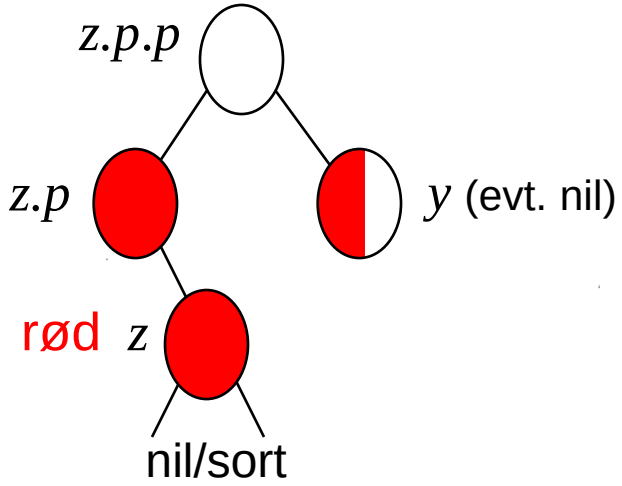
Indsættelse i rød-sorte træer: rebalancering

Omfarv *C* og dens to røde børn



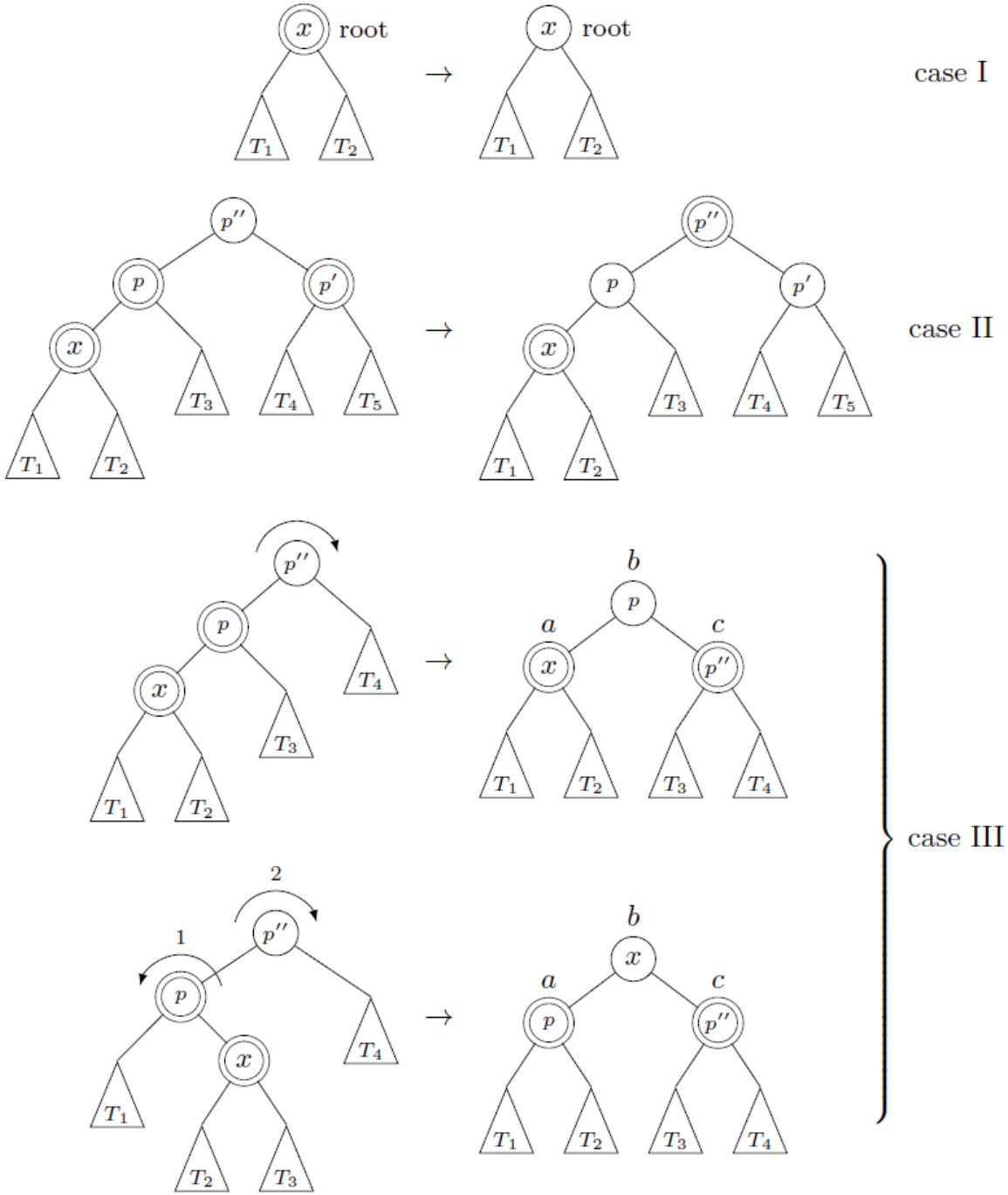
Afslut






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

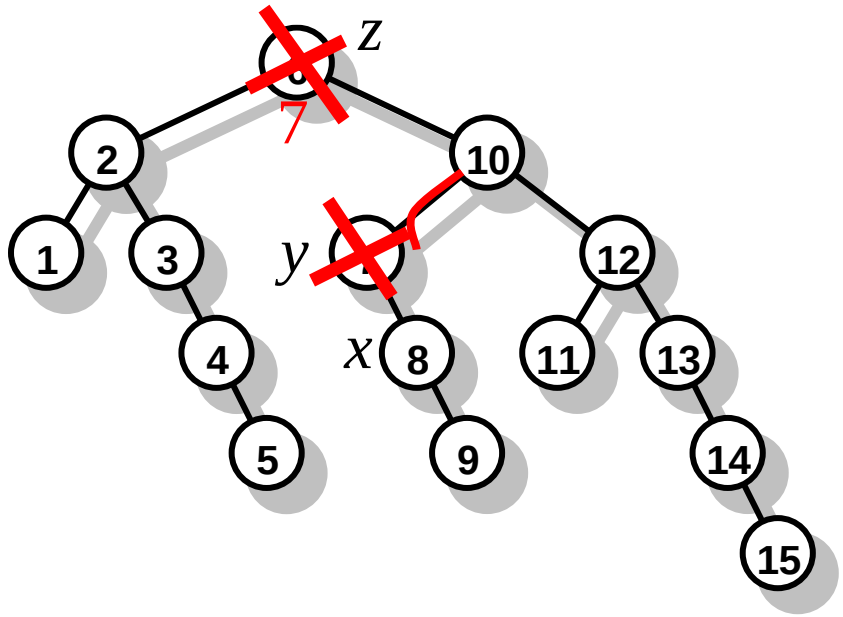
[B]



Delete

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

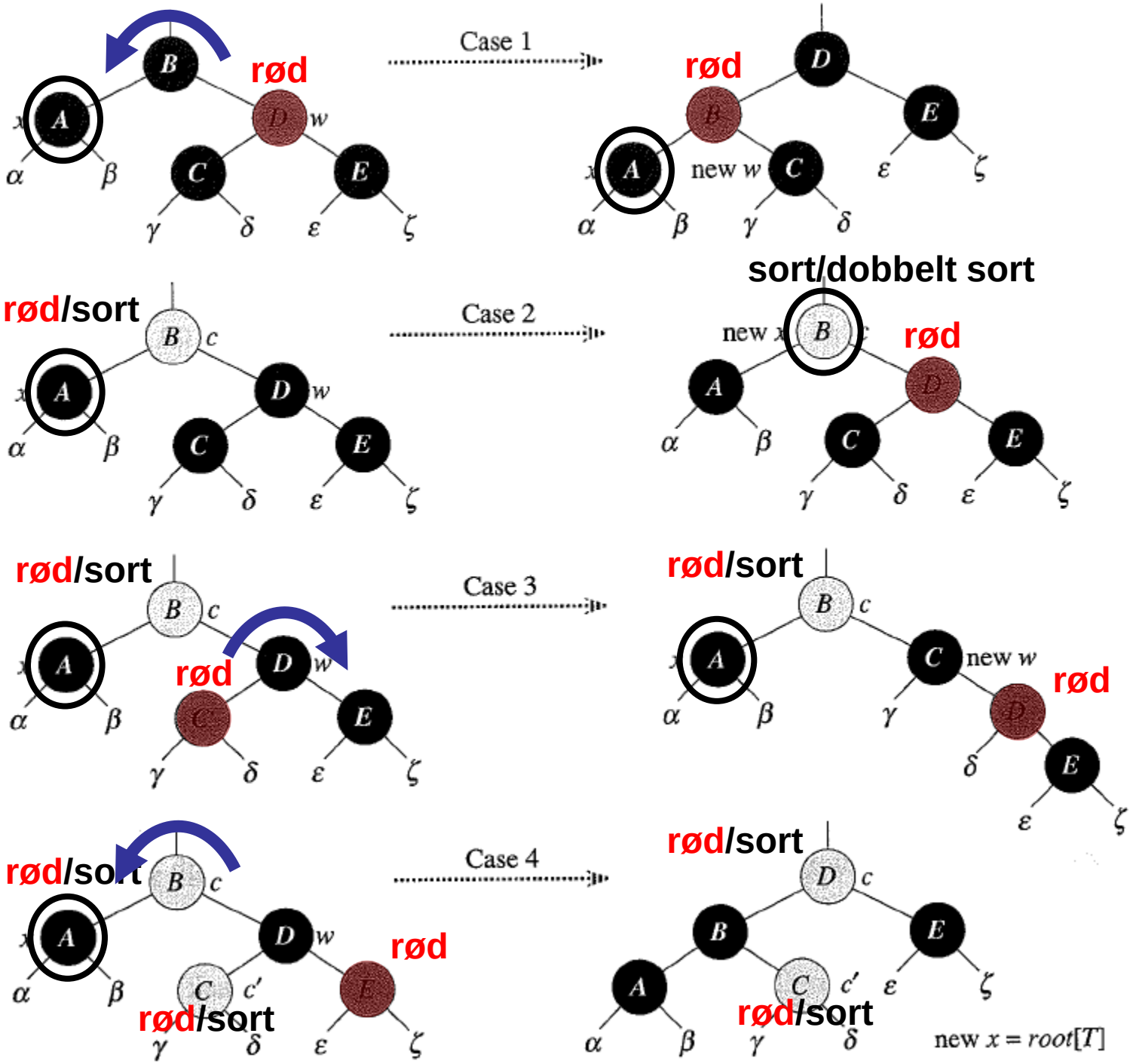
Ubalanceret slettelse



x og y.right kan være T.nil !!!

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

x
”dobbelt sort”



Case 1 → 2,3,4

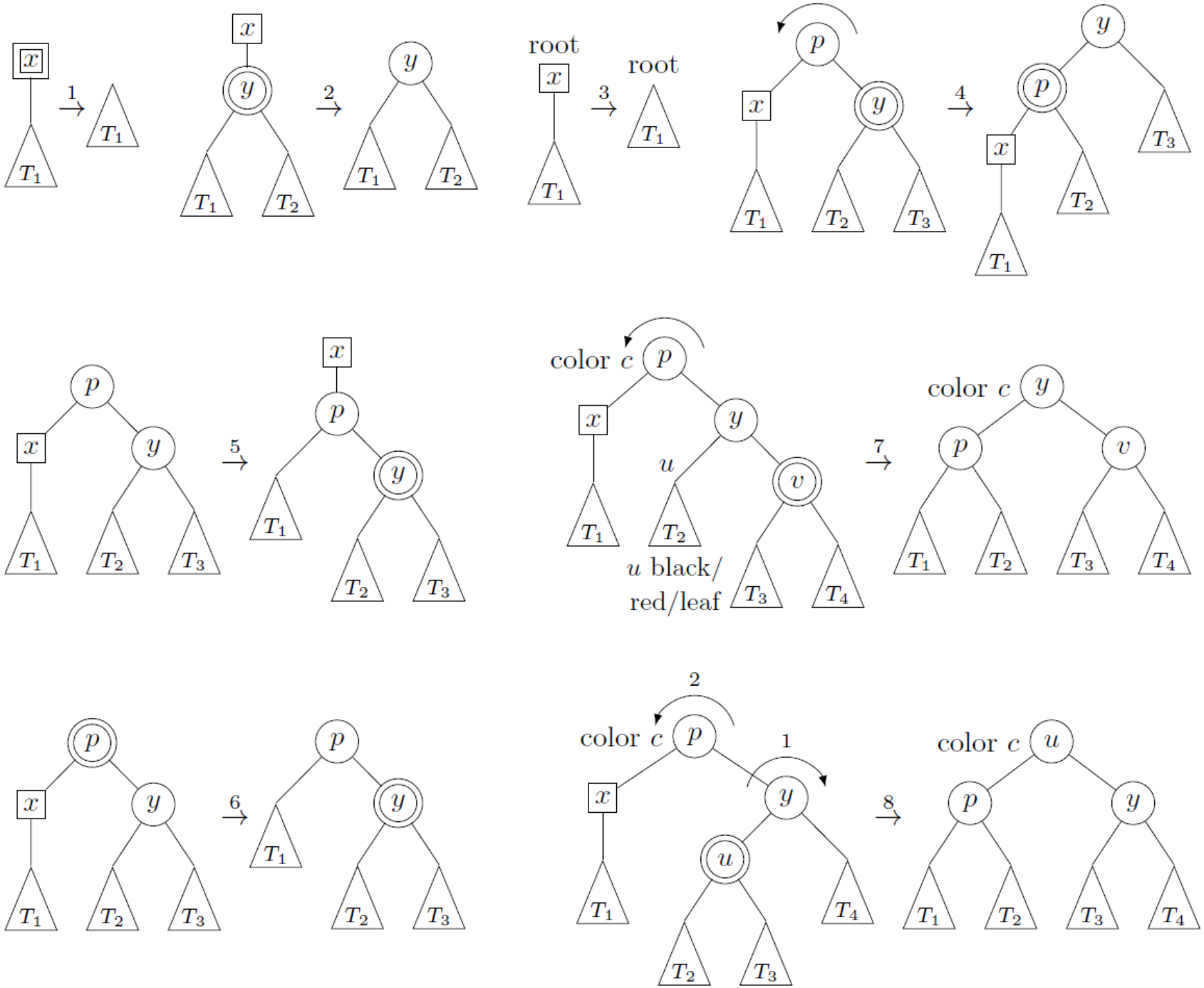
Case 2 → Problemet x flyttet tættere på roden eller færdig

Case 3 → 4 (og færdig)

Case 4 → Færdig

```
RB-DELETE-FIXUP(T, x)
 1  while x ≠ T.root and x.color == BLACK
 2      if x == x.p.left
 3          w = x.p.right
 4          if w.color == RED
 5              w.color = BLACK                          // case 1
 6              x.p.color = RED                          // case 1
 7              LEFT-ROTATE(T, x.p)                      // case 1
 8              w = x.p.right                            // case 1
 9          if w.left.color == BLACK and w.right.color == BLACK
10              w.color = RED                            // case 2
11              x = x.p                                  // case 2
12          else if w.right.color == BLACK
13                  w.left.color = BLACK                 // case 3
14                  w.color = RED                        // case 3
15                  RIGHT-ROTATE(T, w)                   // case 3
16                  w = x.p.right                        // case 3
17              w.color = x.p.color                      // case 4
18              x.p.color = BLACK                        // case 4
19              w.right.color = BLACK                    // case 4
20              LEFT-ROTATE(T, x.p)                      // case 4
21              x = T.root                               // case 4
22      else (same as then clause with "right" and "left" exchanged)
23  x.color = BLACK
```

[B]



Dynamisk Ordbog : Rød-Sorte Træer

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

Ikke pensum

Rød-Sorte Træer i Java og C++

	Java (JDK SE 12.0.1)	C++ (GCC 9.1)
Metode	java.util.TreeMap	std::map
Dokumentation	https://docs.oracle.com/javase/10/docs/api/java/util/TreeMap.html	http://www.cplusplus.com/reference/map/map/
Kildekode	Installer JDK fra www.oracle.com/technetwork/java/javase/downloads/ Fil java.base\java\util\TreeMap.java fra C:\ProgramFiles\Java\jdk-12.0.1\lib\src.zip	https://github.com/gcc-mirror/gcc/blob/master/libstdc++-v3/include/bits/stl_tree.h

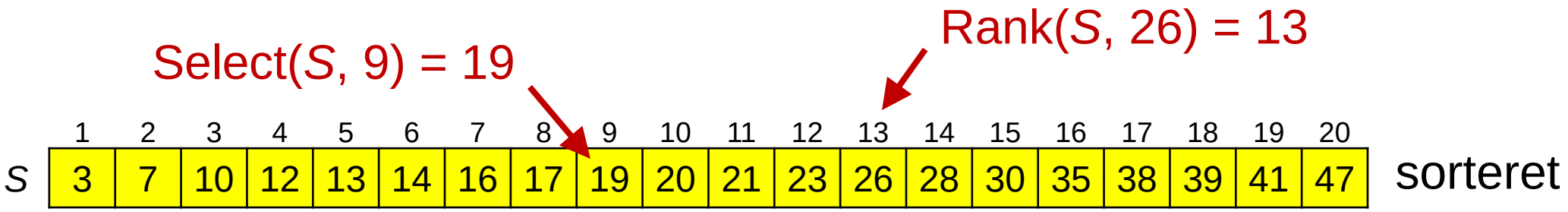
Algoritmer og Datastrukturer

Udvidede søgetræer: Dynamisk rang, interval træer
[CLRS, kapitel 17]

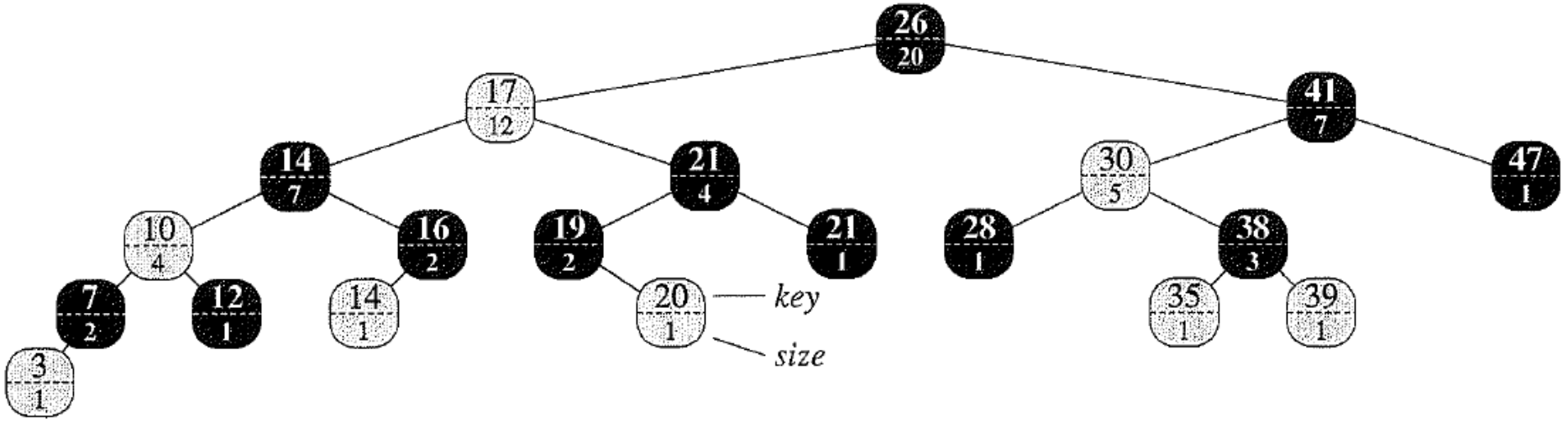
Fenwick træer (ikke pensum)
[B, kapitel 2.1]

Dynamisk Rang

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

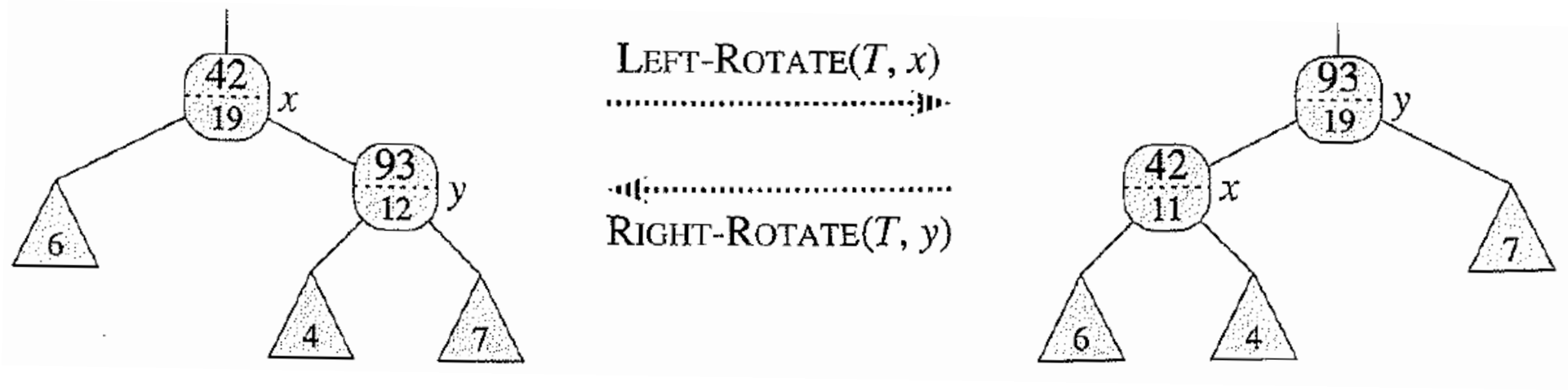


Dynamisk Rang



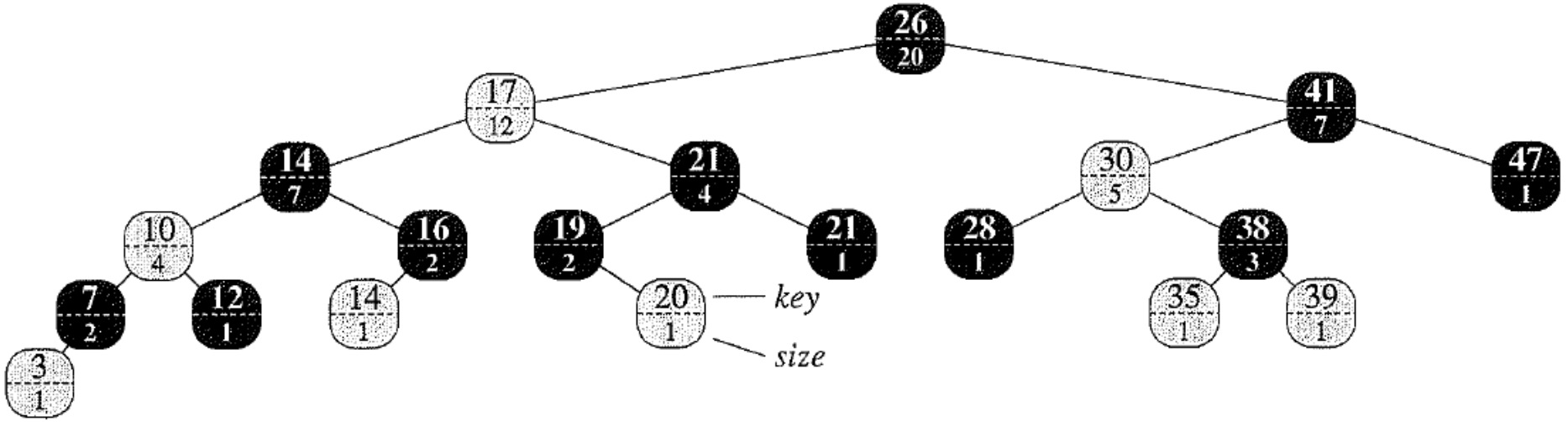
- Find det *i*'te mindste, indsættelser, slettelser
- Vedligehold i **rød**-**sort** søgetræ
- Udvid hver knude med størrelse af undertræerne

Dynamisk Rang



- Indsættelse/slettelse: opdater size på stien til roden
- Under rebalancering af det rød-sorte træ, vedligehold size under rotationer

Dynamisk Rang




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

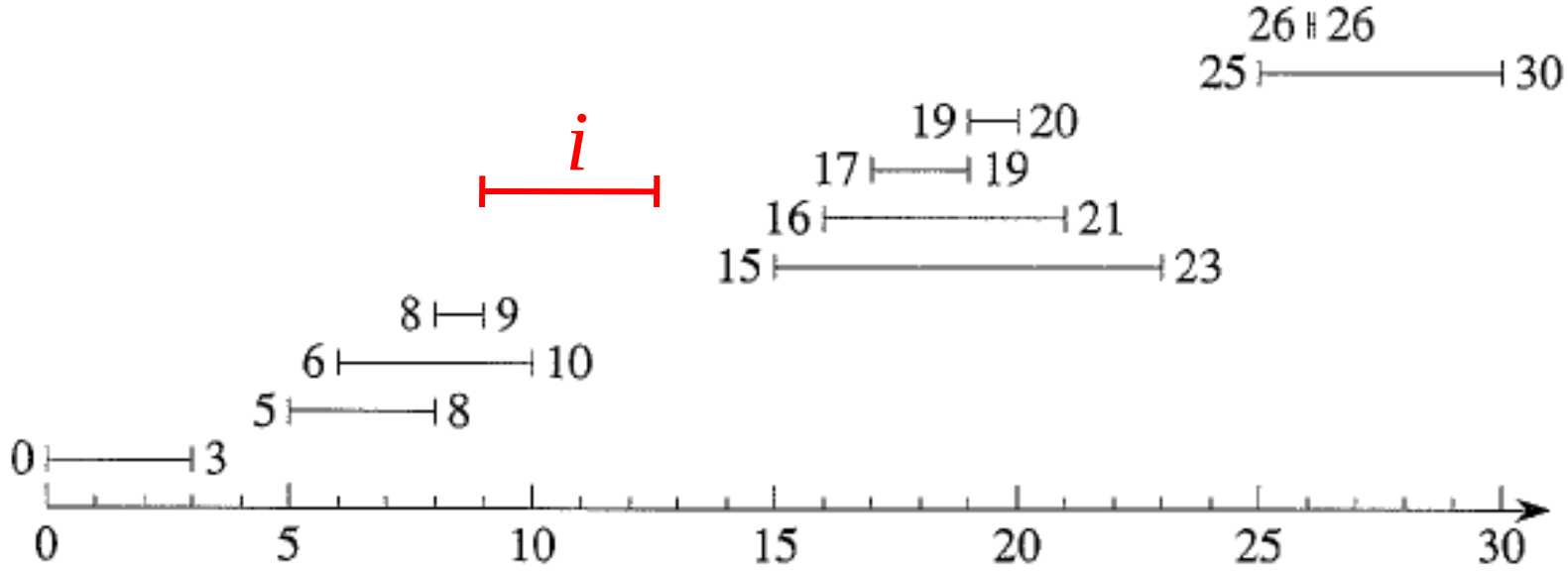
```
OS-SELECT(x, i)
1  r = x.left.size + 1
2  if i == r
3       return x
4  elseif i < r
5       return OS-SELECT(x.left, i)
6  else return OS-SELECT(x.right, i − r)
```

Dynamisk Rang

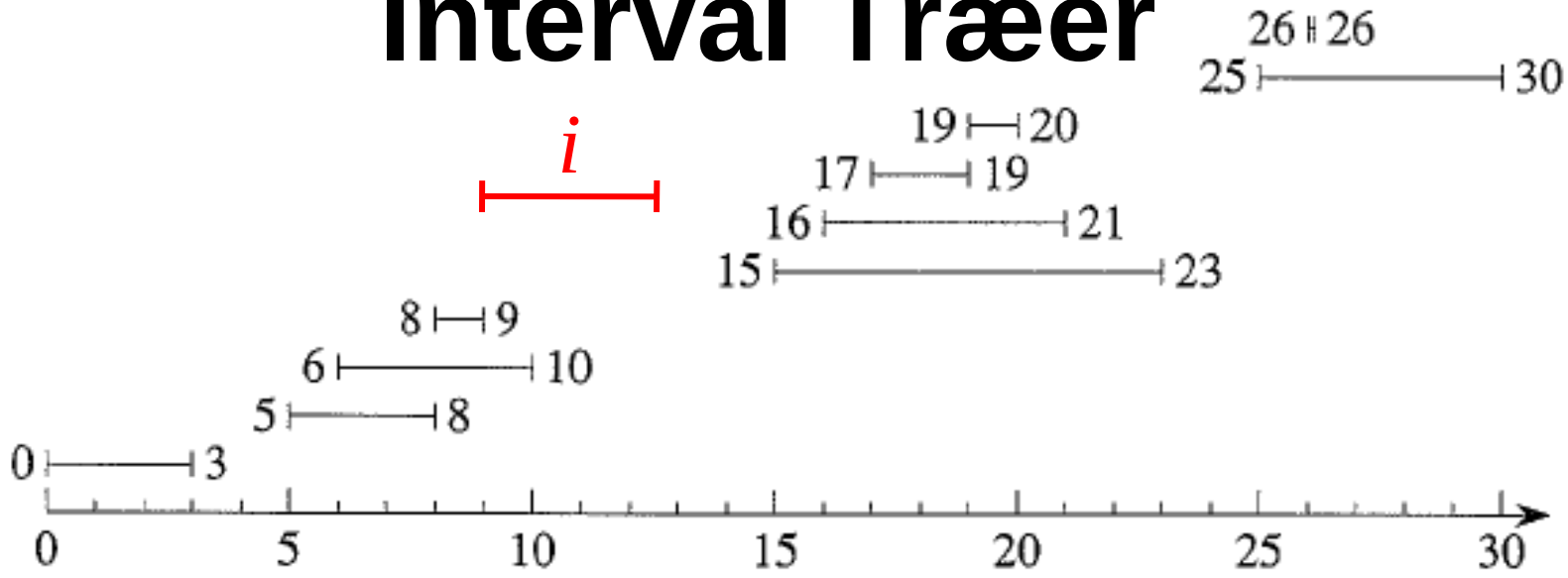
Select(S, i**)**	$O(\log n)$
Rank(S, x**)**	
Insert(S, x**)**	
Delete(S, x**)**	

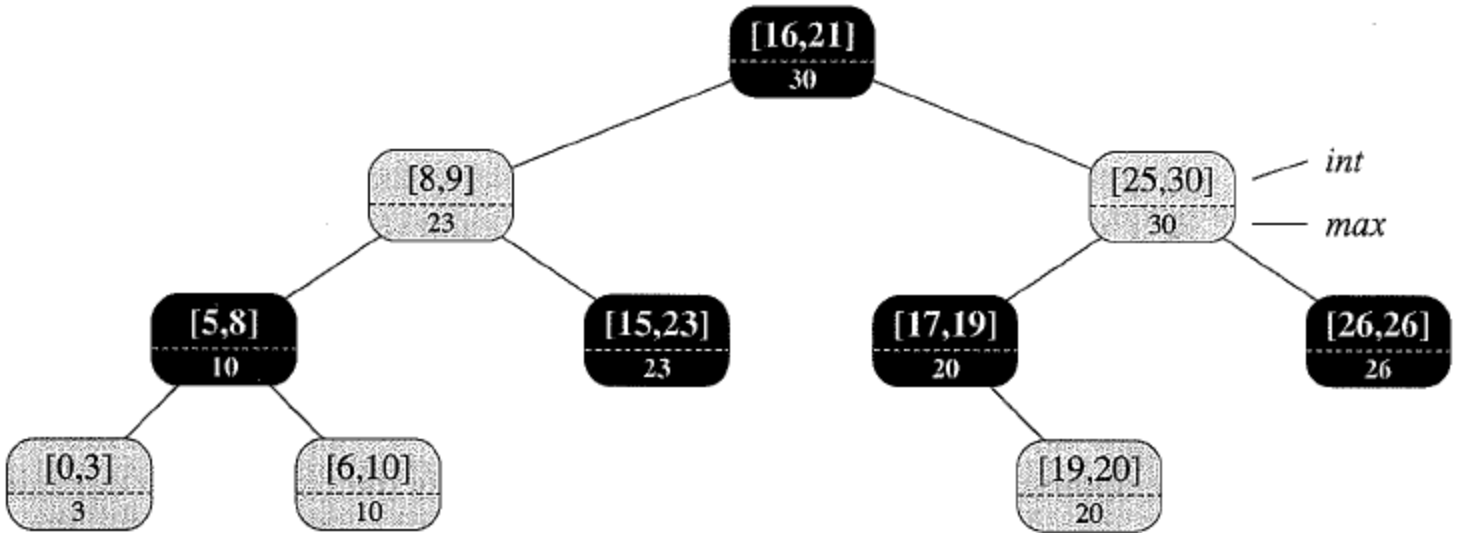
Interval Træer

- Vedligehold en mængde af intervaller
- Indsæt og slet indsatte intervaller
- Find et interval der overlapper med et givet interval



Interval Træer





- Søgetræ over intervallernes **venstre endepunkt**
- Hver knude gemmer yderligere **maximum højre endepunkt** for et interval i undertræet

Interval Træer

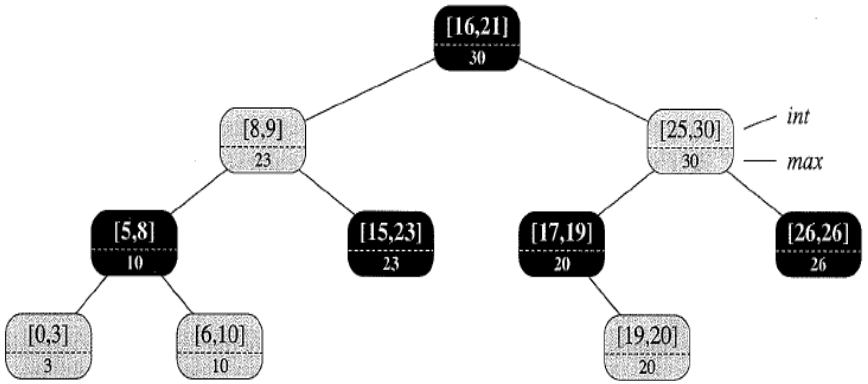
i overlapper *x*

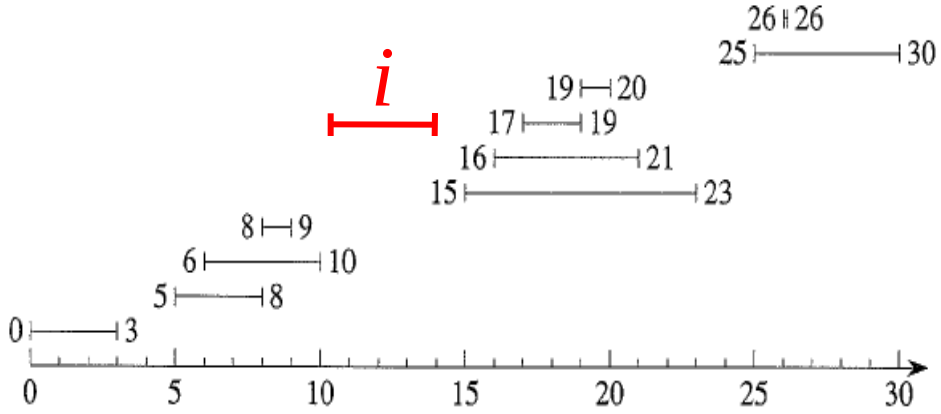
⇕

i.low ≤ *x.high* og *x.low* ≤ *i.high*

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

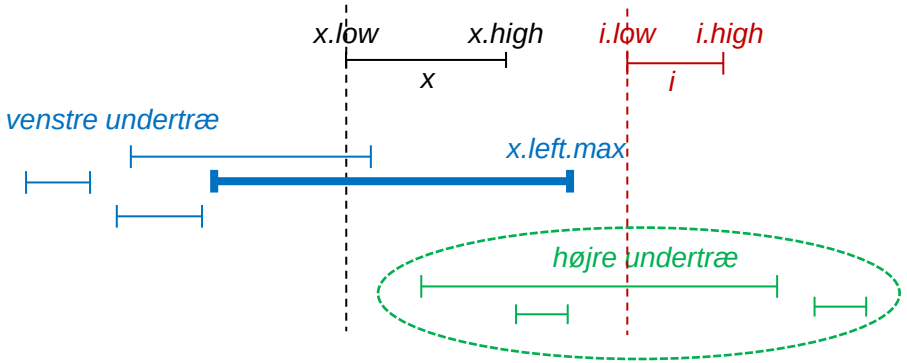
tricky





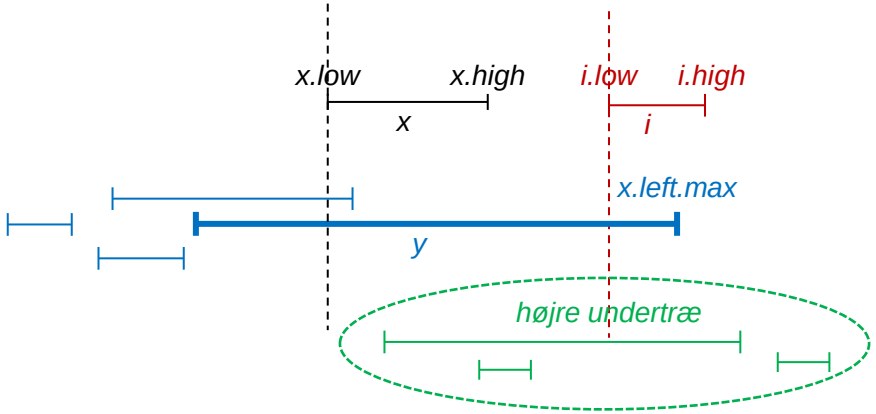
Interval Træer – Korrekthed

x til venstre for *i* **og** *x.left.max* < *i.low*



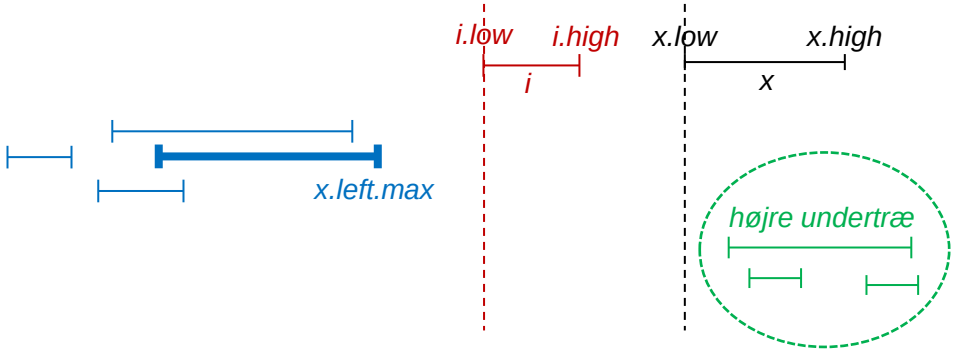
Kan ikke være overlap i venstre undertræ, så sikkert at **gå til højre**

x til venstre for *i* **og** *x.left.max* ≥ *i.low*



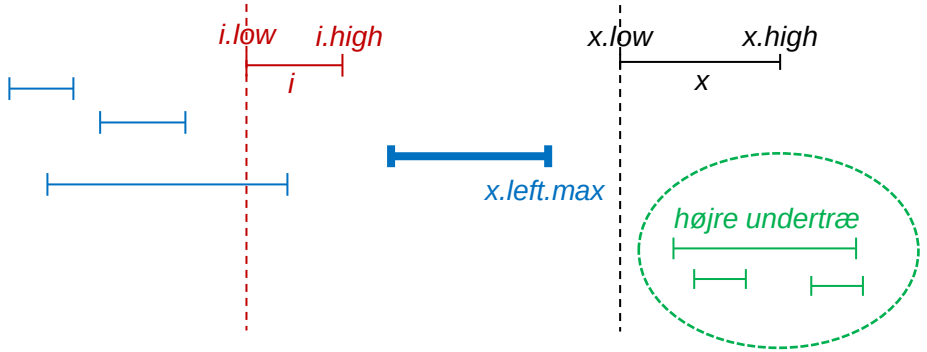
Findes overlappende interval *y* i venstre undertræ, så sikkert at **gå til venstre**

i til venstre for *x* **og** *x.left.max* < *i.low*



Ingen overlap
(overser ikke noget ved at **gå til højre)**

i til venstre for *x* **og** *x.left.max* ≥ *i.low*



Ingen overlap i højre undertræ,
så sikkert at **gå til venstre**

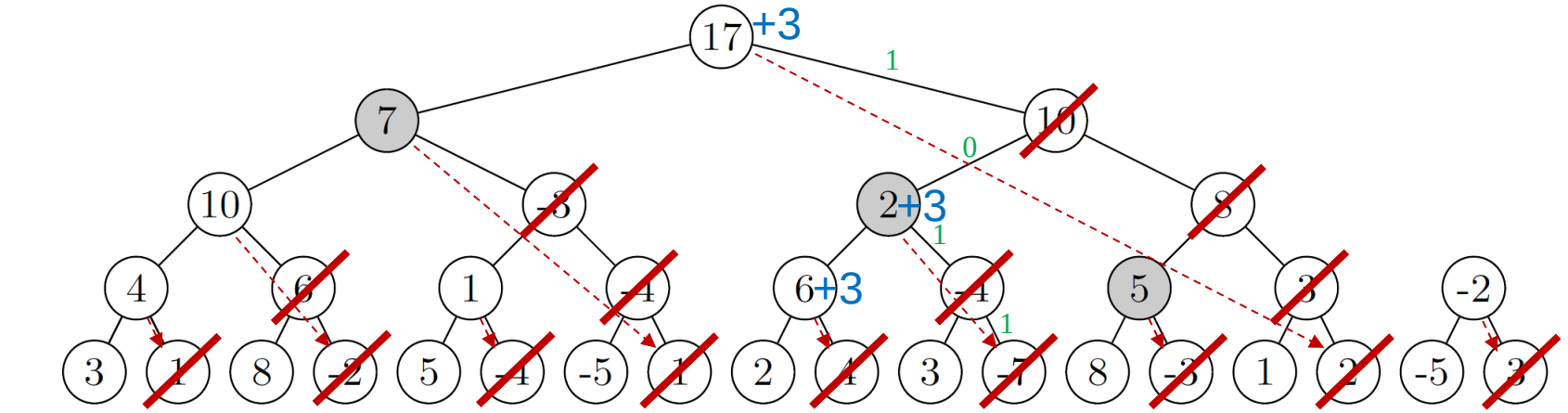
Interval Træer

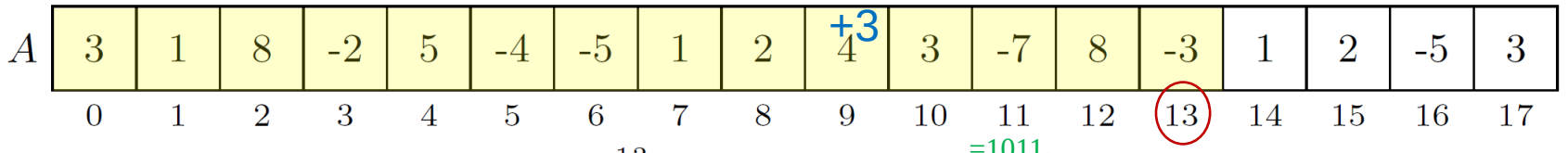
Search(T, i)	$O(\log n)$
Insert(T, i)	
Delete(T, i)	

Fenwick Træer

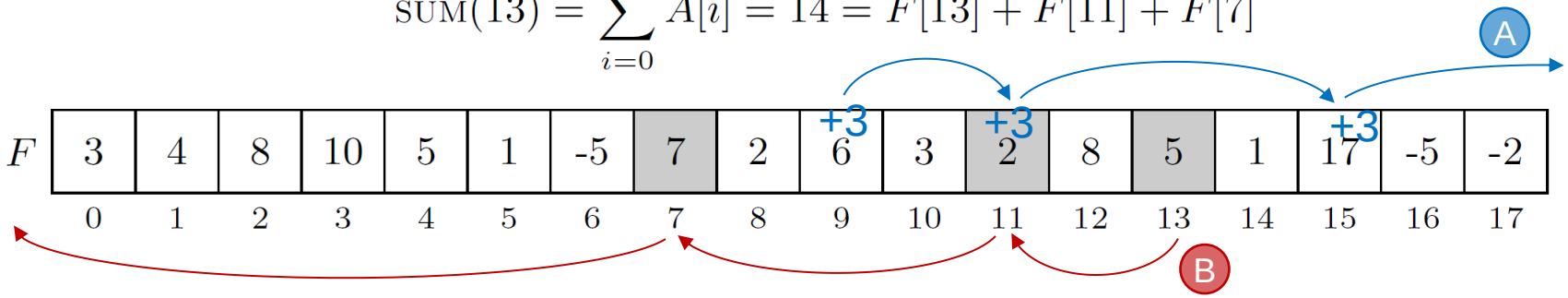
Ikke pensum

Dynamiske præfiks summer i O(log *n*)





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



Algorithm $\text{INIT}(n)$

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

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

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

Algorithm $\text{SUM}(i)$

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

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

Fenwick 1996

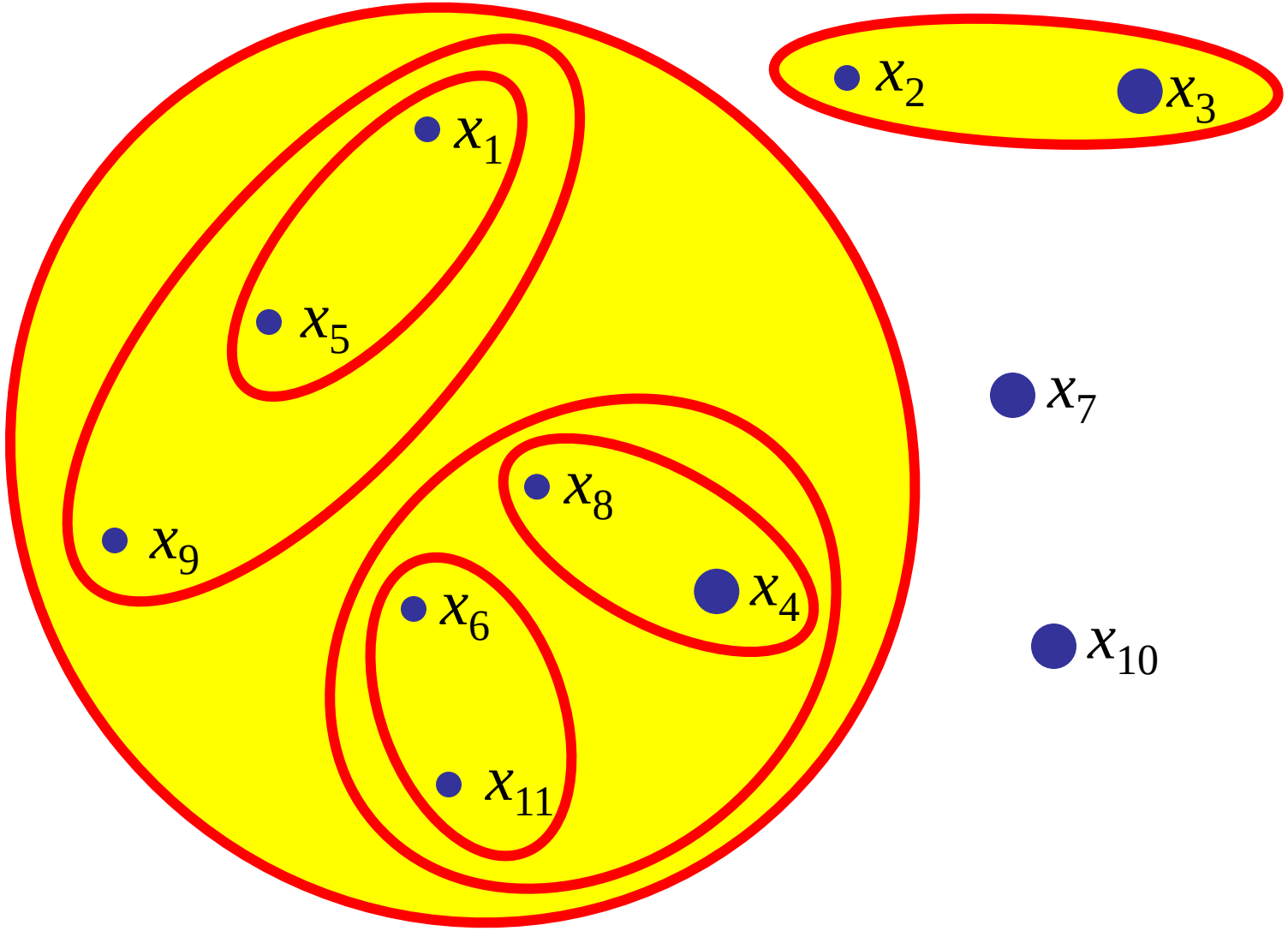
Algoritmer og Datastrukturer

Union-find
[CLRS, kapitel 19.1-19.3]

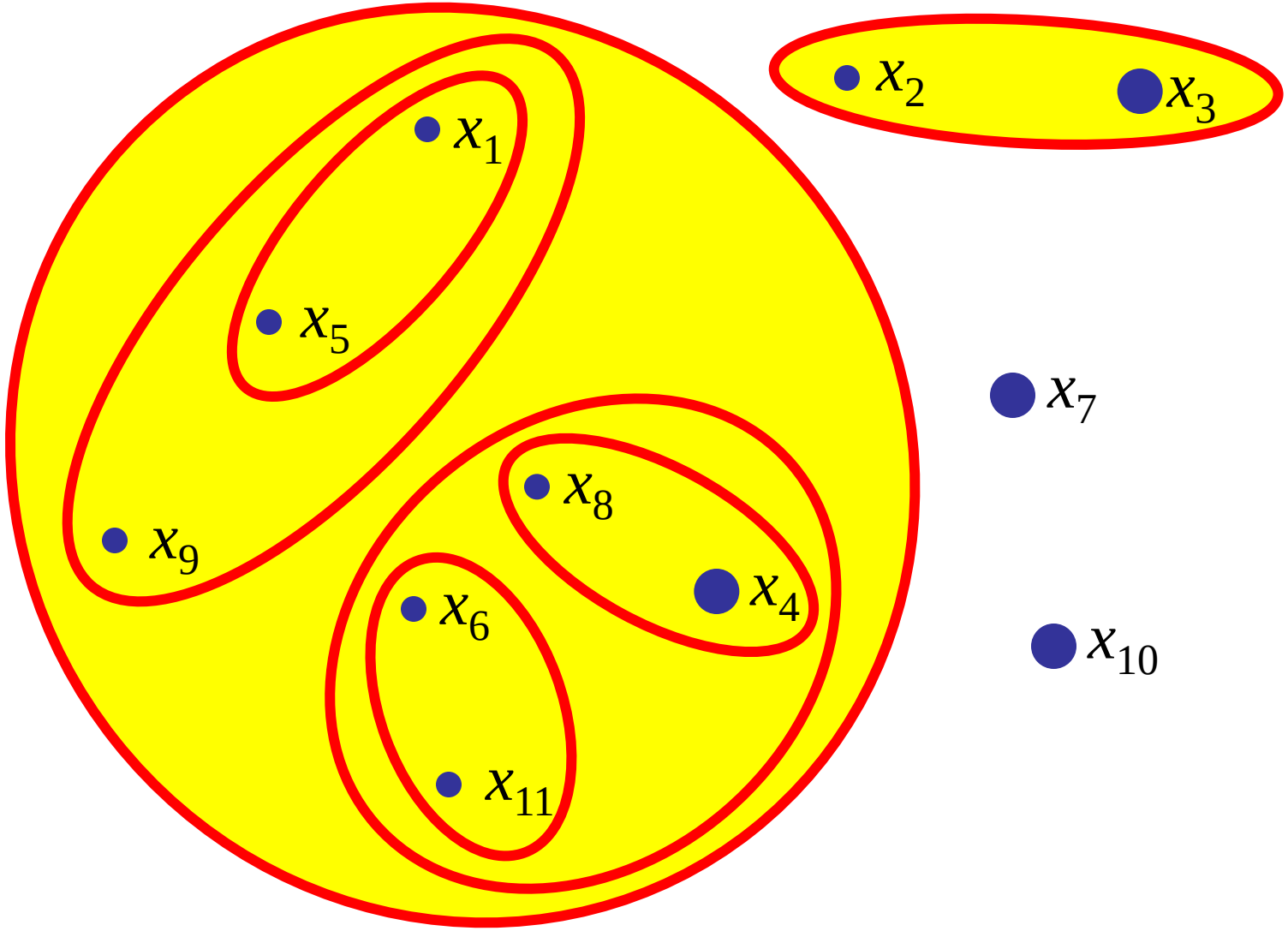
Union-Find

MakeSet(x)	Opret en ny mængde $S = \{ x \}$
Union(x, y)	Erstat $S_x = \{ \ldots, x, \ldots \}$ $S_y = \{ \ldots, y, \ldots \}$ med $S_x \cup S_y = \{ \ldots, x, \ldots, y, \ldots \}$ (det antages at x og y ligger i forskellige mængder)
FindSet(x)	Returner en ***repræsentant*** for $S_x = \{ \ldots, x, \ldots \}$ FindSet(x) = FindSet(y) hvis og kun hvis x og y er i samme mængde

Eksempel : Union-Find



Eksempel : 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

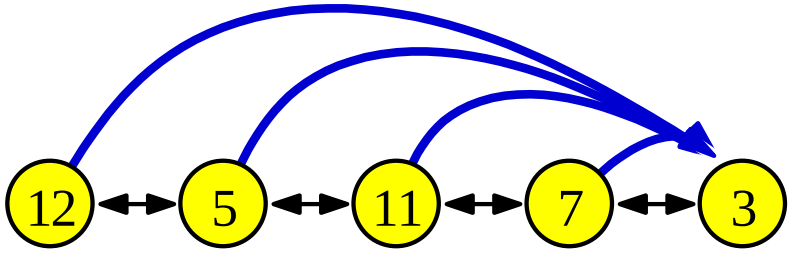
Trærepræsentation (I)

- Mængde = dobbeltkædet liste
- MakeSet = lav en ny knude
- FindSet = returner sidste knude
- Union = konkatener listerne

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

Trærepræsentation (II)

- Mængde = dobbeltkædet liste
- MakeSet = lav en ny knude
- FindSet = returner sidste knude
- Union = konkatener listerne og opdater pointerne



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

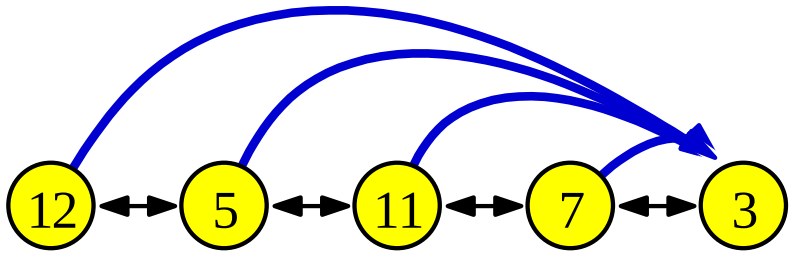
Trærepræsentation (II)
Sekvens af Union

Sætning

Et sekvens af n union operationer tager tid $O(n \cdot \log n)$

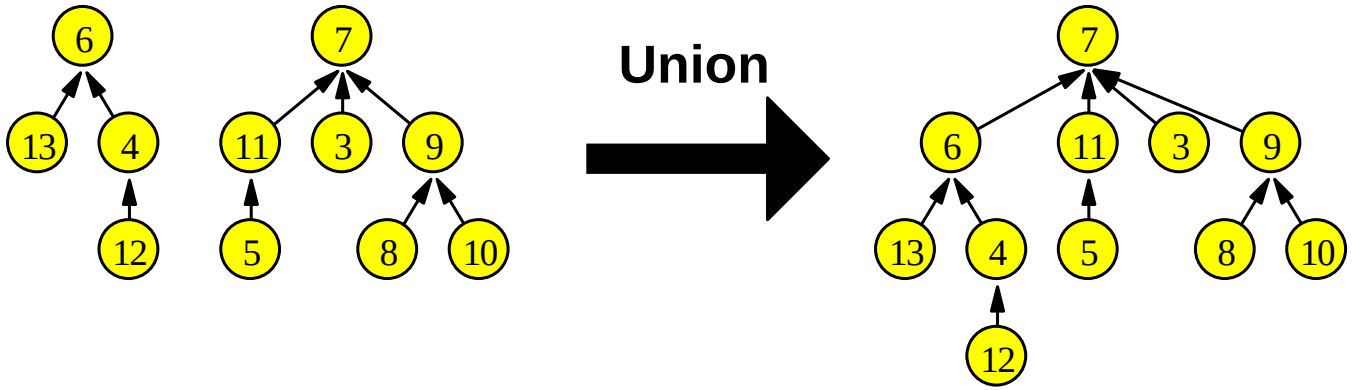
Bevis

Hver pointer flyttes højest log n gange -hver gang til en liste der mindst er dobbelt så stor

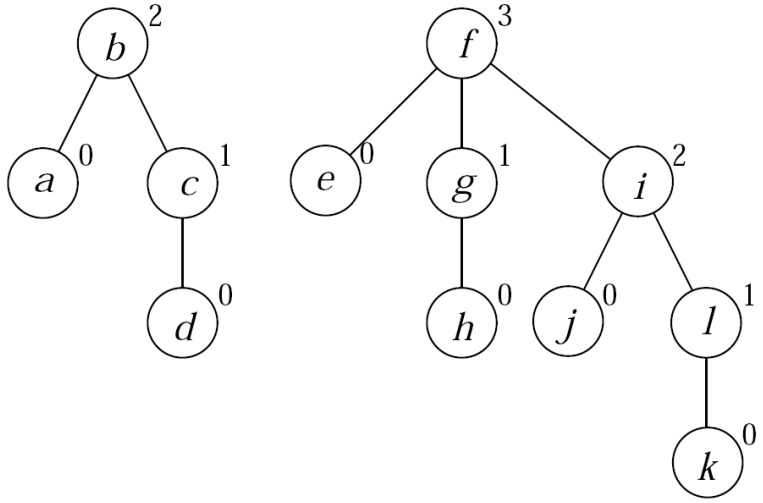


Trærepræsentation (III)

- Mængde = træ
- MakeSet = lav en ny knude
- FindSet = returner roden
- Union = Sæt det ”lille” træ under roden af det ”store” træ (størrelse = antal elementer i træet)



Stikomprimering




```
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
2       y.p = x
3  else x.p = y
4       if x.rank == y.rank
5            y.rank = y.rank + 1

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

linking ved rank

stikomprimering

Analyse af Trærepræsentation med Rank-Linkning

Lemma

$$height[\text{rod}] \leq rank[\text{rod}]$$
$$size[\text{rod}] \geq 2^{rank[\text{rod}]}$$

Bevis

Induktion.

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

(Ovenstående udnytter kun linking by rank)

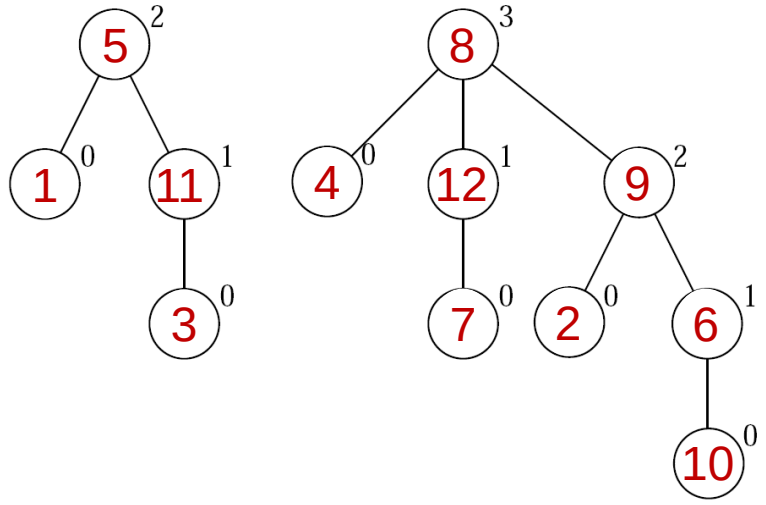
Analyse af Trærepræsentation med Stikomprimering

Sætning

En sekvens af m Union-Find operationer på n elementer tager tid $O(m \cdot \alpha(n))$ hvor $\alpha(n)$ er den Inverse til Ackerman funktionen ([CLRS], kapitel 19.4) hvor for alle praktiske formål $\alpha(n) \leq 4$.

Kompakt Repræsentation

- Elementer = heltal 1..*n*
- *rank* og *p* arrays



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

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

Ikke Rekursiv Løsning

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

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

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

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

- FINDSET kræver to gennemløb af stien

Kun Sti-Komprimering

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

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

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

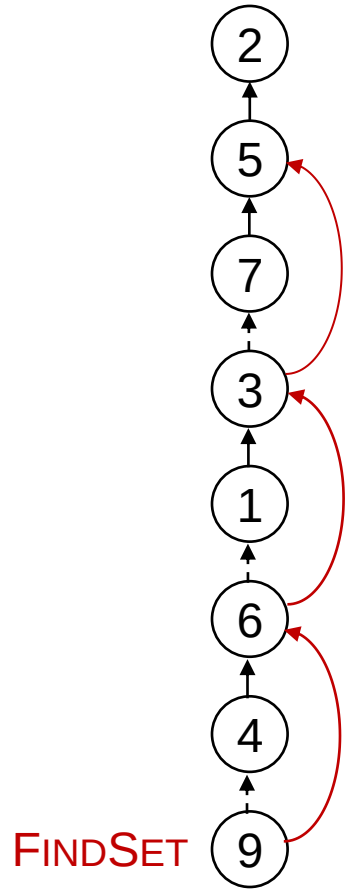
```
FINDSET(x)
1   root = x
2   while root ≠ p[root] do
3       root = p[root]
4   while x ≠ root do
5       y = p[x]
6       p[x] = root
7       x = y
8   return x
```

- FINDSET kræver to gennemløb af stien
- Gemmer kun *p* array
- *m* operationer tager tid O(*m* log *n*)

Alternativ Sti-Komprimering (Single Pass)

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

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

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

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



- FINDSET Kræver ét gennemløb af stien
- Hver anden knude peger på bedsteforældren
- m operationer tager tid $O(m \cdot \alpha(n))$

Algoritmer og Datastrukturer

Amortiseret analyse
[CLRS, kapitel 16]

compact1, compact2, compact3

java.util

Class ArrayList<E>

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

All Implemented Interfaces:

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

Direct Known Subclasses:

`AttributeList, RoleList, RoleUnresolvedList`

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

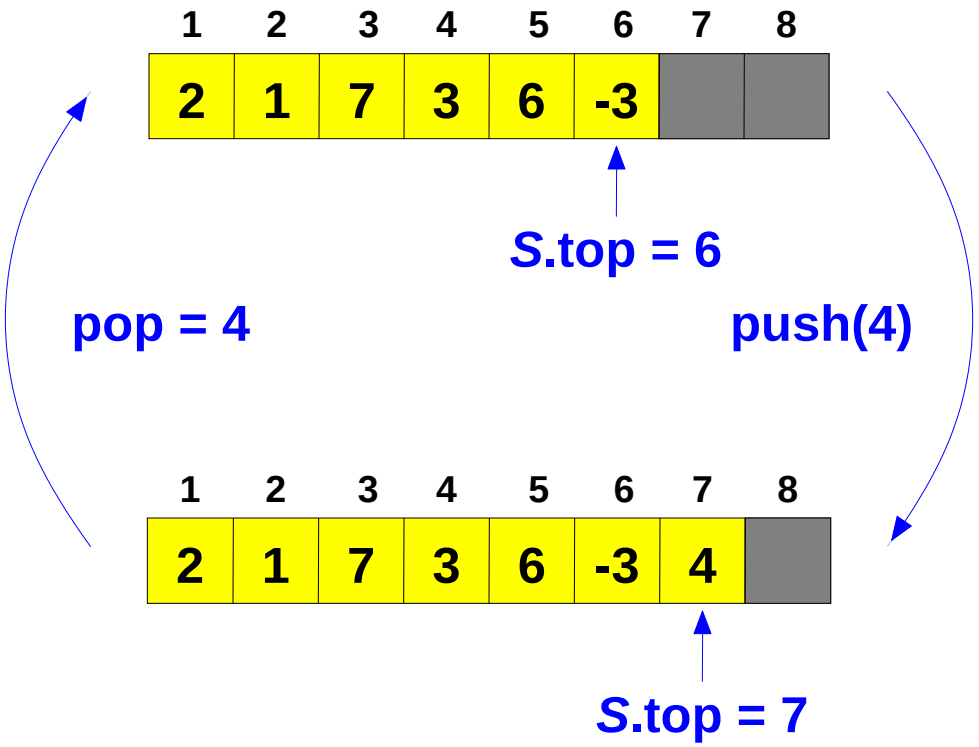
Resizable-array implementation of the `List` interface. Implements all optional list operations, and permits all elements, including `null`. In addition to implementing the `List` interface, this class provides methods to manipulate the size of the array that is used internally to store the list. (This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

The `size`, `isEmpty`, `get`, `set`, `iterator`, and `listIterator` operations run in constant time. The add operation 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

Stak

Stak : Array Implementation



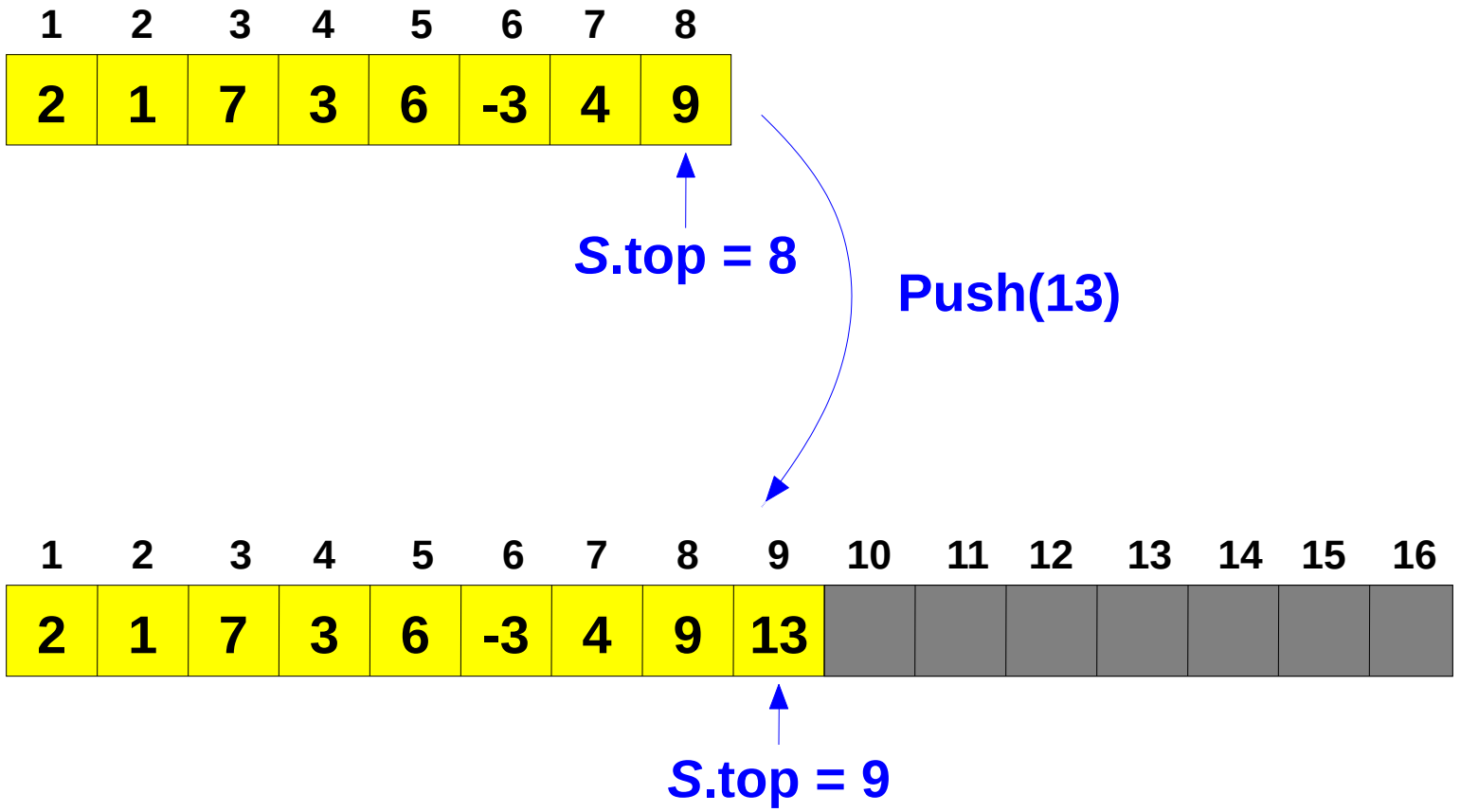

```
STACK-EMPTY(S)
1  if S.top == 0
2       return TRUE
3  else return FALSE
```

```
PUSH(S, x)
1  S.top = S.top + 1
2  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) tid

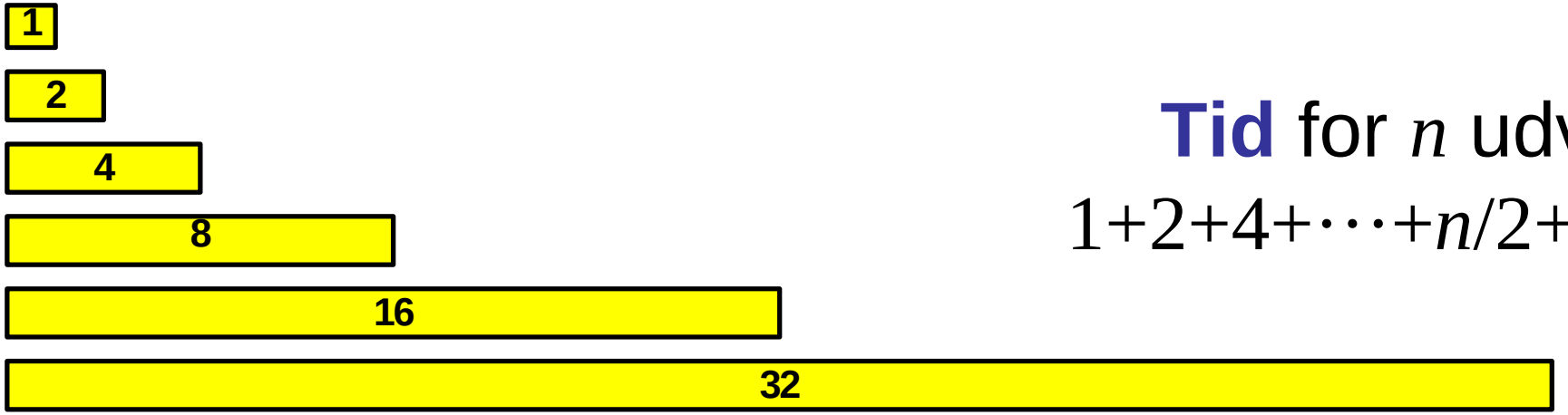
Stak : Overløb



Array fordobling : $O(n)$ tid

Array Fordobling

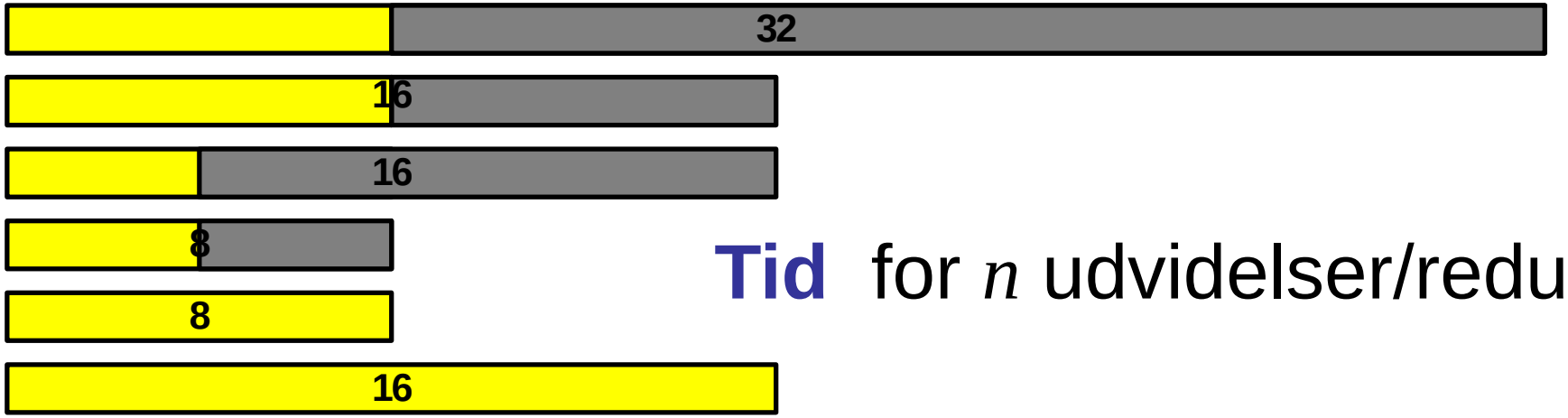
Fordoble arrayet når det er fuld



Tid for n udvidelser:

$1+2+4+\cdots+n/2+n = O(n)$

Halver arrayet når det er <1/4 fyldt



Tid for n udvidelser/reduktioner:

$O(n)$

Array Fordobling + Halvering

– en generel teknik

Tid for n udvidelser/reduktioner er $O(n)$

Plads $\leq 4 \cdot$ aktuelle antal elementer

Array implementation af Stak:

n push og pop operationer tager $O(n)$ tid

Analyse teknik ønskes...

Krav

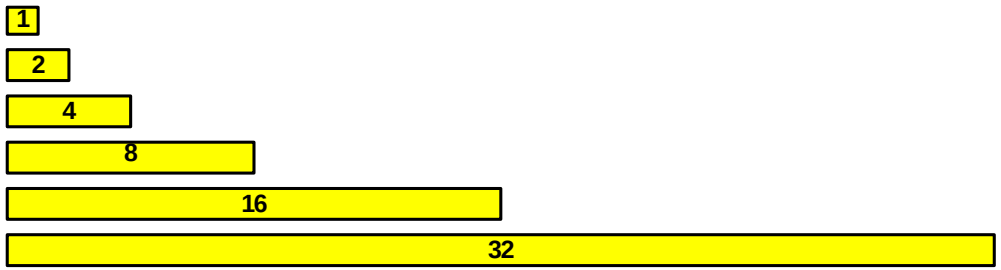
- Analysere **worst-case** tiden for en **sekvens** af operationer
- Behøver kun at analysere den **enkelte operation**

Fordel

- Behøver **ikke** overveje andre operationer i sekvens og deres **indbyrdes påvirkninger**
- Gælder for alle sekvenser med de givne operationer

Intuition

- Der findes ”**gode**”/”**balancerede**” tilstande og ”**dårlige**”/”**ubalancerede**”
- At komme fra en ”**dårlig**” tilstand til en ”**god**” tilstand er **dyrt**
- Det tager mange operationer fra en ”**god**” tilstand før man er i en ”**dårlig**”
- For de (mange) **billige** operationer ”betaler” vi lidt ekstra for senere at kunne lave en **dyr** operation næsten **gratis**



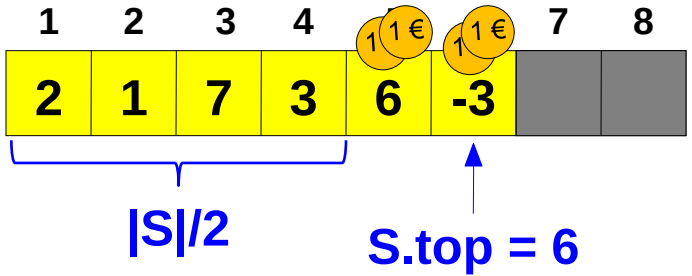
Amortiseret Analyse

- **1 €** kan betale for $O(1)$ **arbejde**
- En operation der tager tid $O(t)$ koster t €
- Hvornår vi betaler/sparer op er ligegyldigt – bare pengene er der når vi skal bruge dem!
- Opsparing = Potentiale = Φ
- Vi kan ikke låne penge, dvs. vi skal spare op før vi bruger pengene, $\Phi \geq 0$
- Amortiseret tid for en operation = hvad vi er **villige** til at betale – men vi skal have råd til operationen!
- Brug **invarianter** til at beskrive sammenhængen mellem **opsparingen** og **datastrukturens tilstand**

Eksempel: Stak

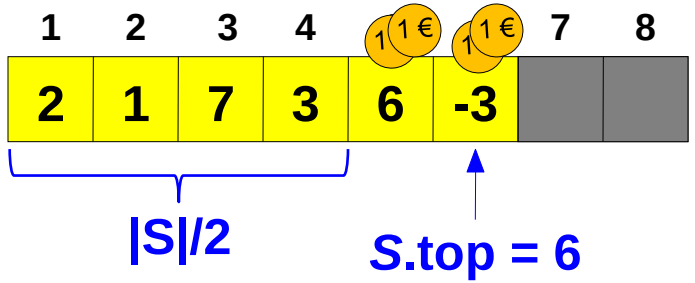
- En **god** stak er halv fuld – kræver ingen opsparing
- Invariant : $\Phi = 2 \cdot |\, S.top - |S|/2\,|$
- Antag: 1 € per element indsættelse/kopiering
- Amortiseret tid per push: 3 **€ ?**

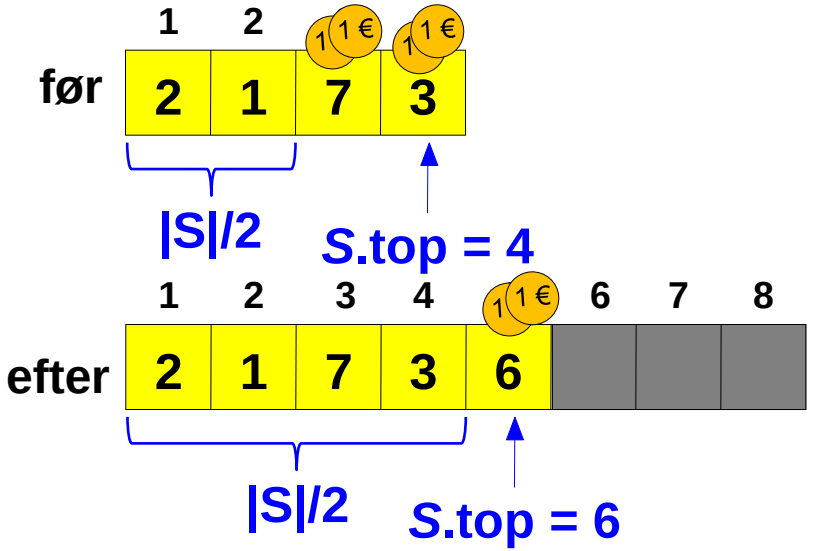
  (har vi altid penge til at udføre operationen?)
- Hvis ja: n push operationer koster ≤ $3n$ €



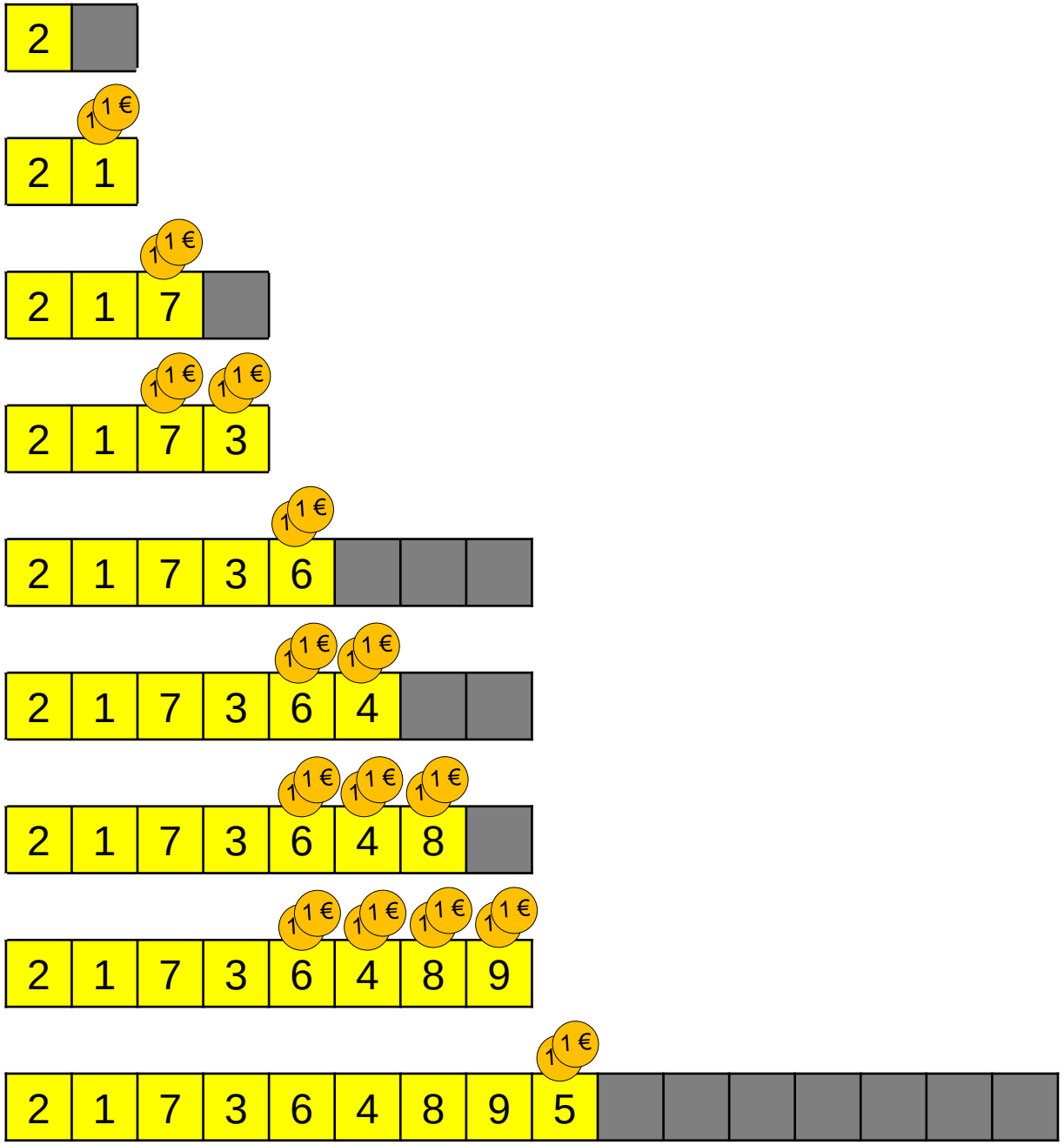
Eksempel: Stak
Push = Amortiseret 3€

- Push uden kopiering:
  - Et nyt element : **1 €**
  - $\Phi = ||S|/2\text{-top}[S]|$ vokser med højst 1,
    så invarianten holder hvis vi sparer **2 €** op
  - Amortiseret tid: $1+2 = \mathbf{3}$ **€**
- Push med kopiering
  - Kopier S: $|S|$ **€**
  - Indsæt nye element: **1 €**
  - Φ før = $|S|$, Φ efter = 2, dvs $|S|-2$ **€ frigives**
  - Amortiseret tid: $|S|+1-(|S|-2) = \mathbf{3}$ **€**



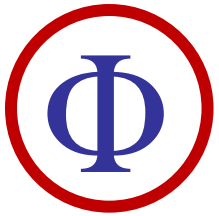


Invariant: $\Phi = 2 \cdot |\, S.\text{top}-|S|/2\,|$



	potentiale	potentiale vækst	faktisk tid	amortiseret tid
	0			
push(1)		2	1	2+1 = 3
	2			
push(7)		0	2+1	0+2+1 = 3
	2			
push(4)		2	1	2+1 = 3
	4			
push(6)		-2	4+1	-2+4+1 = 3
	2			
push(4)		2	1	2+1 = 3
	4			
push(8)		2	1	2+1 = 3
	6			
push(9)		2	1	2+1 = 3
	8			
push(5)		-6	8+1	-6+8+1 = 3
	2			

Amortiseret Analyse

- Teknik til at argumenter om **worst-case** tiden for en sekvens af operationer
- Behøver kun at analysere operationerne enkeltvis
- **Kunsten**: Find den rigtige invariant for 

Eksempel: Rød-Sorte Træer

Insert(x)

=

Søgning ← $O(\log n)$

+

Opret nyt rødt blad ← $O(1)$

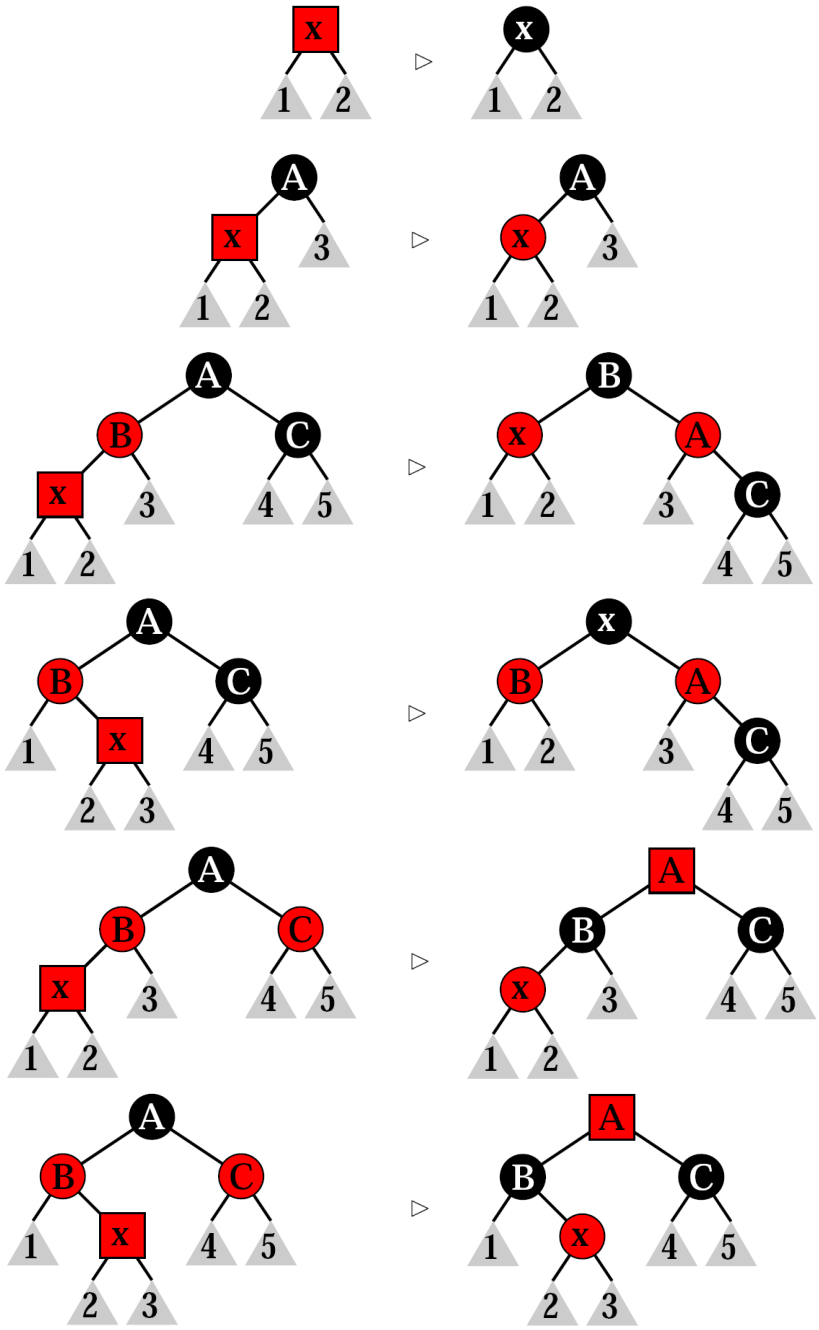
+

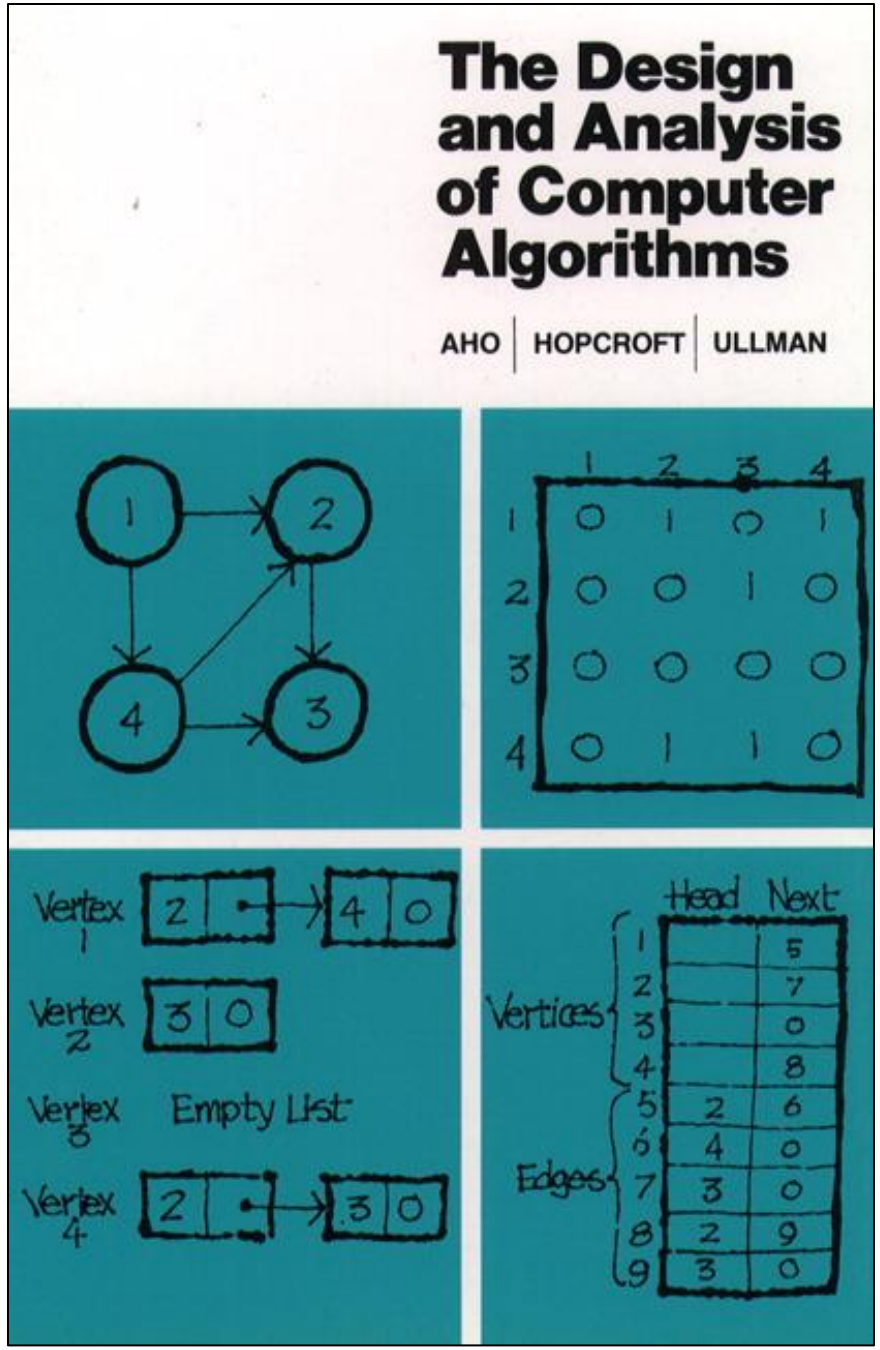
Rebalancering ← # transitioner

transitioner = *amortiseret* O(1)

Φ = # røde knuder

Korollar: Indsættelse i rød-sorte træer tager *amortiseret* O(1) tid, hvis indsættelsespositionen er kendt




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

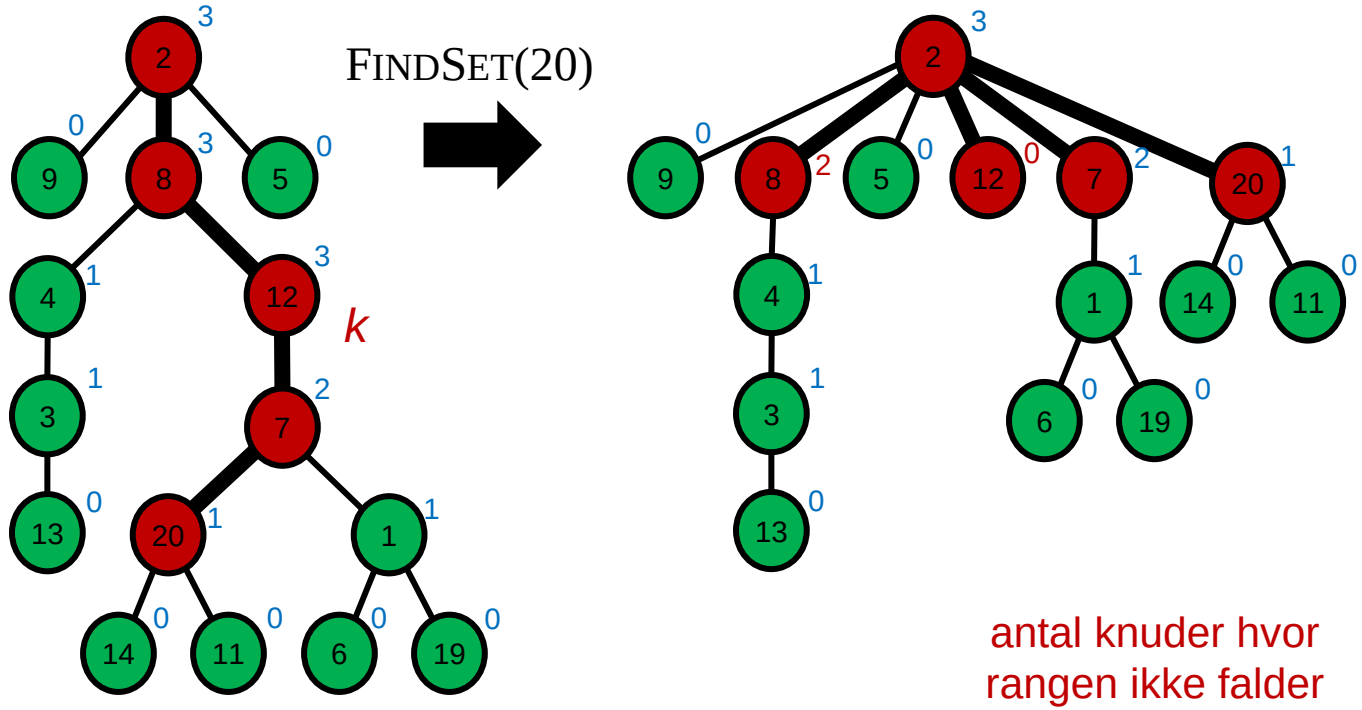
1974

Opgave **4.17 i Aho Hopcroft Ullman ”The Design and Analysis of Computer Algorithms” (1974)

Ikke pensum

Union-Find med Sti-Komprimering (uden 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
```



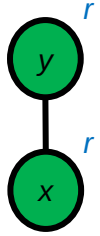
antal knuder hvor rangen ikke falder

Amortiseret analyse

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

Lemma rank(*x.p*) ≥ rank(*x*)

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



Observation

Antag *x* og *y* har rang *r*, dvs. begge undertræer har størrelse $[2^r, 2^{r+1}[$

Hvis *x* fjernes, så får *y* rang < *r*

	Faktisk arbejde	ΔΦ	Amortiseret arbejdet (faktisk arbejde + ΔΦ)
MAKESET	O(1)	0	O(1)
LINK	O(1)	≤ log *n*	O(log n)
FINDSET	*k*	≤ - *k* + 2 + log *n*	O(log n)
UNION = 2 x FINDSET + LINK			O(log n)

Ikke pensum

Union-Find med Sti-komprimering og Union-by-rank

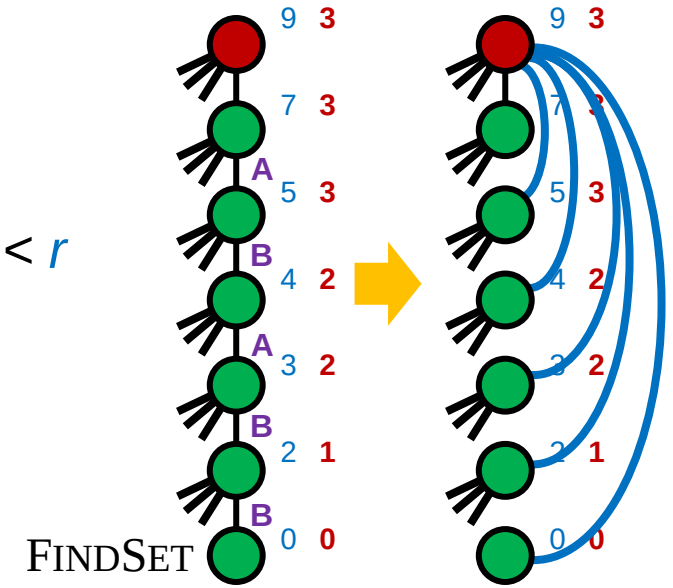
Sætning n MAKESET og m FINDSET og UNION operationer tager tid $O((n + m) \cdot \log^* n)$

Definition $\log^* n$ = mindste i hvor $n \le r_i = 2^{2^{2^2}}$ } tårn af højde i

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

Bevis

- Kun rødders rank kan ændre sig (vokse under link)
- En ikke-rod x bliver aldrig rod igen og rank($x.p$) > rank(x)
- En rod med rank r har ≥ 2^r knuder i undertræet, alle ikke-rødder rank < r
- **Lemma** # knuder med rank r ≤ $n / 2^r$
- En parent-ændring øger rank-forskellen til parent
- Definer laget af en rank r ved $\lambda(r) = \log^* r$
- Total tid $O(n + m +$ # parent-ændringer$) =$



$$O(n + m + \underbrace{m \cdot \log^* n}_{\text{(B) parent er allerede i et andet lag}} + \underbrace{\sum_{\text{lag } i} \sum_{r=r_{i-1}+1..r_i} n / 2^r \cdot r_i}_{\text{(A) parent er i samme lag før parent-ændring}}) = O(n + m \cdot \log^* n + \sum_{\text{lag } i} n / 2^{r_{i-1}} \cdot r_i) = O((n + m) \cdot \log^* n)$$

#gange en knude i lag i kan have en parent i lag i

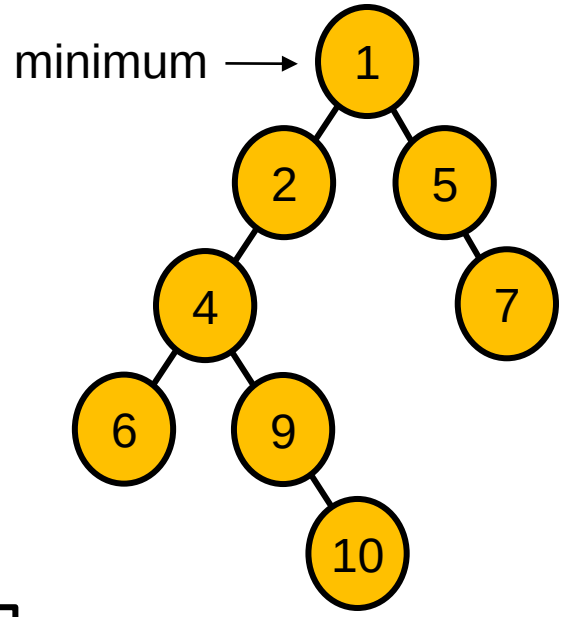
$r_i \le 2^{r_i - 1} = 2^{r_{i-1}}$ (antager $r_{-1} = 0$)

Ikke pensum

Skew heaps – ”selvbalancerende” heaps

Sleator og Tarjan, *Self-Adjusting Heaps*, SIAM Journal on Computing 15(1): 52–69, 1986

- Prioritetskø:
  - `FindMin`, `Insert`, `DeleteMin`, `Meld`
- Heap-ordnet binært træ
  - *Ingen krav til struktur eller dybde*
- Knude: `left`, `right`, `element`
- Prioritetskø = pointer til roden



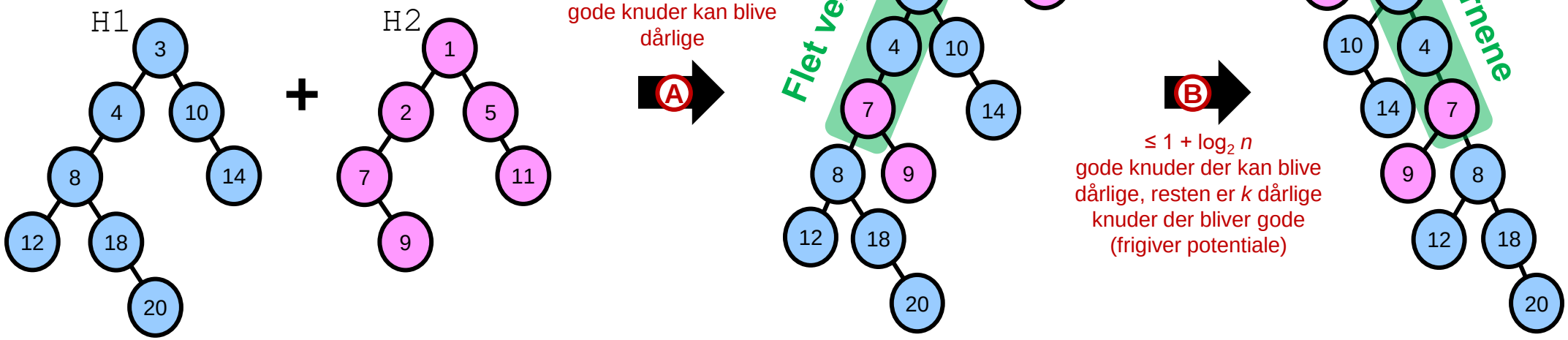

```
proc FindMin(H)
    return H.element

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

Ikke pensum

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))
```

Amortiseret analyse:

Definition God knude `v` : |`v.left`| ≤ |v| / 2
Lemma # gode knuder på venstre stien ≤ 1 + $\log_2 n$
Potentiale Φ = # dårlige knuder

`Meld` **amortiseret omkostning:**

ΔΦ fra A + ΔΦ fra B + faktisk arbejde

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

⇒ Alle operationer amortiseret time O(log n)

Ikke pensum

Selvbalancerende Datastrukturer

med amortiserede udførselstider

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

Skew Heaps Sleator og Tarjan, SICOMP 1986 O_{AM} ≡ amortiseret tid
Fibonacci heaps Fredman og Tarjan, JACM 1987 [CLRS, 3. udgave, kapitel 19]
Splay trees Sleator og Tarjan, JACM 1985 + Cole* , SICOMP 2006

Algoritmer og Datastrukturer

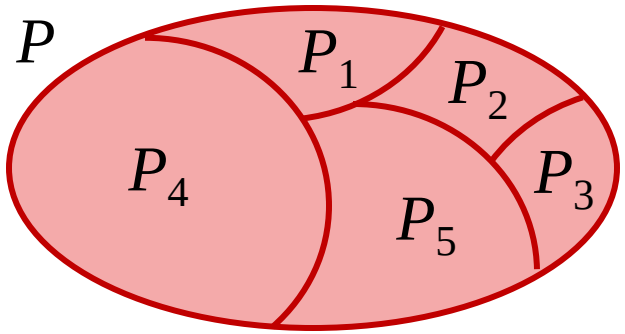
Del-og-kombiner
[CLRS, kapitel 2.3, 4.1-4.5, problem 30.1.c]

Del-og-Kombiner

Algoritme design teknik

Virker for mange problemer (men langt fra alle)

- **Opdel** et problem P i mindre problemer P_1,..,P_k, der kan løses uafhængigt
  (små problemer løses direkte)
- Løs delproblemerne P_1,..,P_k **rekursivt**
- **Kombiner** løsningerne for P_1,..,P_k til en løsning for P



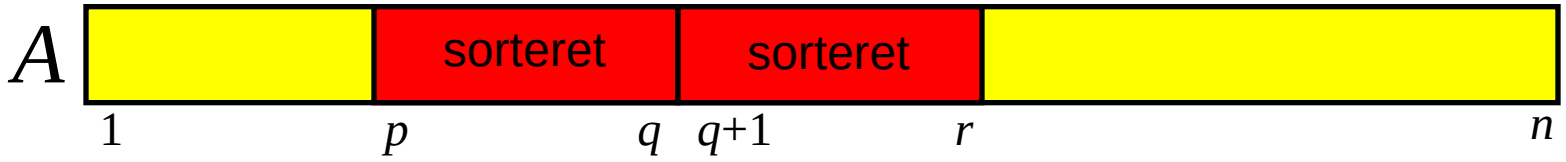
Eksempel: Merge-Sort

```
MERGE-SORT(A, p, r)
if p < r
    q = ⌊(p + r)/2⌋
    MERGE-SORT(A, p, q)
    MERGE-SORT(A, q + 1, r)
    MERGE(A, p, q, r)
```

① To mindre delproblemer

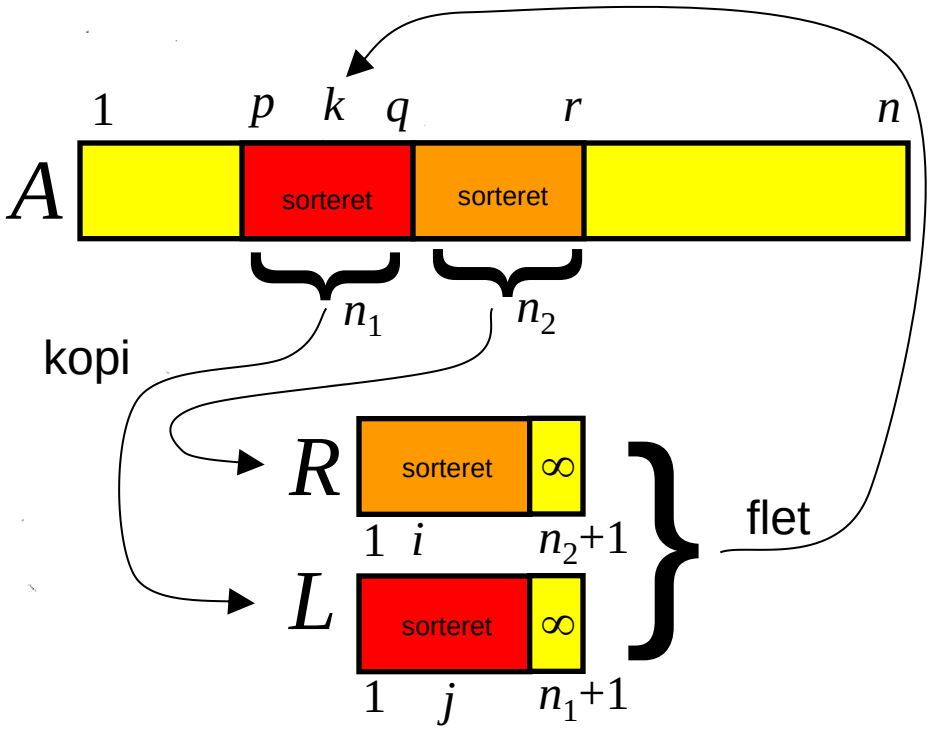
② Løs rekursivt

③ Kombiner



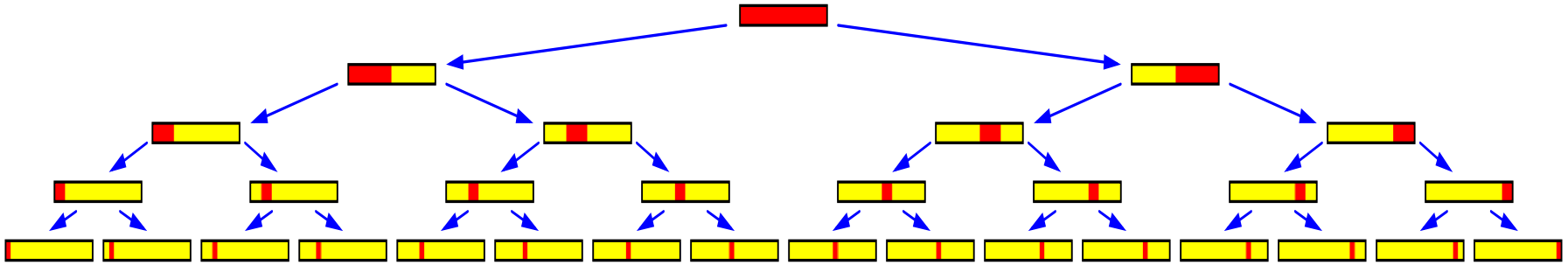
③ Kombiner

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



Merge-Sort : Analyse

Rekursionstræet



Observation

Samlet arbejde per lag er $O(n)$

Arbejde

$$O(n \cdot \#\,\text{lag}) = O(n \cdot \log_2 n)$$

Del-og-kombiner, eksempler:

- **MergeSort**
  - Del op i to lige store dele
  - Rekursiv sortering
  - Kombiner = fletning
- **QuickSort**
  - Opdel i to dele efter tilfældigt pivot (**tilfældig opdeling**)
  - Rekursiv sortering
  - Kombiner = ingen (konkatener venstre og højre)
- **QuickSelect**
  - Opdel efter tilfældigt pivot (**tilfældig opdeling**)
  - Et rekursivt kald til select
  - Kombiner = ingen

Analyse af Del-og-Kombiner

= analyse af en rekursiv procedure

Essentielt to forskellige måder:

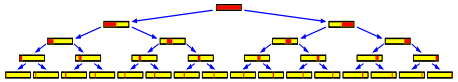
1. Argumenter direkte om **rekursionstræet** (analyser dybde, #knuder på hvert niveau, arbejde i knuderne/niveauerne/træet)
2. Løs en matematisk **rekursionsligning**, f.eks.

$$T(n) = a \quad \text{hvis} \quad n \leq c$$
$$T(n) = 2 \cdot T(n/2) + a \cdot n \quad \text{ellers}$$

Bevises f.eks. vha. induktion.

Løsning af rekursionsligninger

- Fold rekursionsligningen ud og argumenter om **rekursionstræet**



- Gæt en løsning og vis den ved induktion efter voksende n

$$T(n) = a \quad \text{hvis} \quad n \leq c$$
$$T(n) = 2 \cdot T(n/2) + a \cdot n \quad \text{ellers}$$

Rekursionsligninger: Faldgrubber

- Ulige opdelinger glemmes (n ulige, så er de rekursive kald typisk $\lfloor n/2 \rfloor$ og $\lceil n/2 \rceil$) [CLRS, kapitel 4.7]

- Analyserer typiske kun for $n = 2^k$

- Brug aldrig O-udtryk i rekursionsformlen – brug konstanter (~~$T(n) = O(n) + O(T(n/3))$~~)

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

Master Theorem

(Simplificering af [CLRS, Theorem 4.1])

Theorem

Hvis a, b, c, d og p er konstanter, hvor a er et heltal og b, c, d og p er reelle tal, $a \geq 1$, $b > 1$, $c > 0$, $d \geq 1$ og $p \geq 0$, så har rekursionsligningen

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

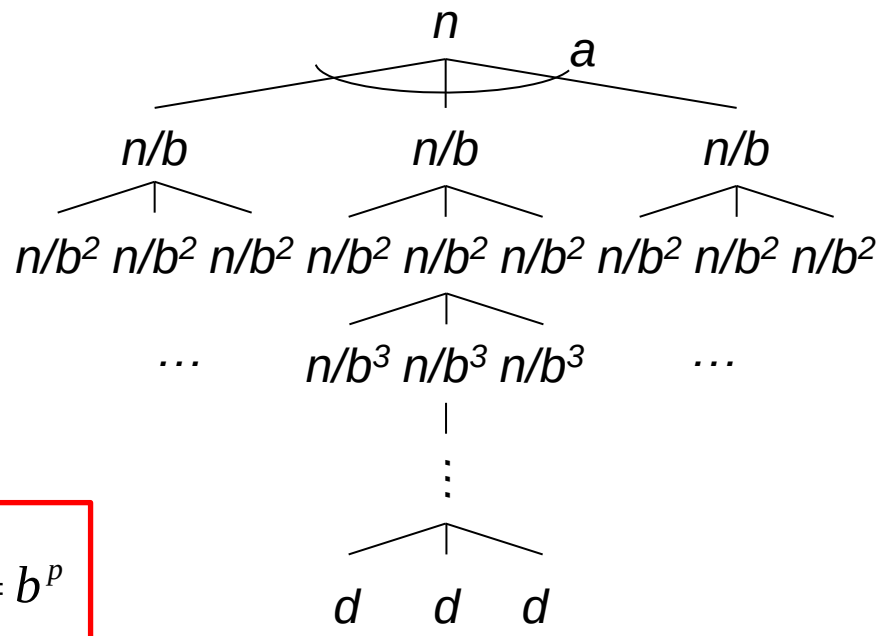
følgende løsning

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

Bemærk $n^{\log_b a} = a^{\log_b n}$.

Dybde	***i* = 0..$\log_b$ (*n*/*d*) - 1**	**$\log_b$ (*n*/*d*)**
# delproblemer	a^i	$a^{\log_b (n/d)}$
Størrelse af delproblemer	n/b^i	d
Tid per delproblem	$c \cdot (n/b^i)^p$	c
Tid per lag	$a^i \cdot c \cdot (n/b^i)^p$	$c \cdot a^{\log_b (n/d)}$

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



$$T(n) = \Theta\left(\underbrace{c \cdot a^{\log_b (n/d)}}_{\text{(bunden af rekursionen)}} + \underbrace{\sum_{i=0}^{\log_b (n/d)-1} a^i \cdot c \cdot (n/b^i)^p}_{\text{(lag } i = 0..\log_b (n/d) - 1)} \right)$$

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

$$\frac{(a/b^p)^{\log_b n} - 1}{a/b^p - 1} \quad \text{for} \quad a \neq b^p$$

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

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

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

Multiplikation af lange heltal

$$2387 * 3351$$

$$= (23 * 10^2 + 87) * (33 * 10^2 + 51)$$

$$= 23 * 33 * 10^4 + (23 * 51 + 87 * 33) * 10^2 + 87 * 51$$

$$= 759 * 10^4 + (1173 + 2871) * 10^2 + 4437$$

$$= 7590000$$

$$+ \ \ 117300$$

$$+ \ \ 287100$$

$$+ \ \ \ \ 4437$$

$$= \underline{\underline{7998837}}$$

Multiplikation af lange heltal

[CLRS, problem 30.1.c]

Karatsuba 1960

- I og J hver heltal med n bits
- Naive implementation kræver $O(n^2)$ bit operationer
- Lad $I = I_h \cdot 2^{n/2} + I_l$ og $J = J_h \cdot 2^{n/2} + J_l$
- $I \cdot J = I_h \cdot J_h \cdot 2^n + ((I_h - I_l) \cdot (J_l - J_h) + I_l \cdot J_l + I_h \cdot J_h) \cdot 2^{n/2} + I_l \cdot J_l$

$T(n) = 3 \cdot T(n/2) + c \cdot n$	for $n \geq 2$
$T(n) = c$	for $n = 1$

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

Multiplikation af lange heltal

+4000 år	$O(n^2)$
Del-og-kombiner Karatsuba 1960	$O(n^{\log_2 3})$
Schönhage-Strassen 1971	$O(n \cdot \log n \cdot \log\log n)$
Fürer 2007 Harvey, Hoeven 2018 Harvey, Hoeven 2019	$O(n \cdot \log n \cdot 2^{O(\log^* n)})$ $O(n \cdot \log n \cdot 2^{2 \cdot \log^* n})$ $O(n \cdot \log n)$

Ikke pensum: Anatolii A. Karatsuba har skrevet en artikel om historien af multiplikation. *The Complexity of Computations*, Proceedings of the Steklov Institute of Mathematics, 1995, vol. 211, 169–183 (russisk original)

Matricer

$m = 3$
søjler / kolonner

repræsenterer
lineær transformation $\mathbb{R}^3 \longrightarrow \mathbb{R}^4$

$n = 4$
rækker

$$\begin{pmatrix} 3 & 2 & -1 \\ 5 & 0 & 4 \\ 7 & -2 & 0 \\ 1 & 3 & 2 \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 3x_1 + 2x_2 - x_3 \\ 5x_1 + 4x_3 \\ 7x_1 - 2x_2 \\ x_1 + 3x_2 + 2x_3 \end{pmatrix}$$

$n \times m$ matrix

søjlevektor
$m \times 1$ matrix

søjlevektor
$n \times 1$ matrix

Regneregler

Matrix addition
($n \times m$ matricer)

$$\begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} + \begin{pmatrix} 5 & 6 \\ 4 & 1 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 6 & 8 \\ 7 & 5 \\ 8 & 9 \end{pmatrix}$$

$$(A + B) + C = A + (B + C)$$
$$A + B = B + A$$

Matrix subtraktion
($n \times m$ matricer)

$$\begin{pmatrix} 6 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} - \begin{pmatrix} 2 & 6 \\ 4 & 1 \\ 3 & 3 \end{pmatrix} = \begin{pmatrix} 4 & -4 \\ -1 & 3 \\ 2 & 3 \end{pmatrix}$$

$$A - (B + C) = (A - B) - C$$

Multiplikation med
konstant

$$3\begin{pmatrix} 1 & 2 \\ -2 & 3 \\ 4 & -1 \end{pmatrix} = \begin{pmatrix} 3 & 6 \\ -6 & 9 \\ 12 & -3 \end{pmatrix}$$

$$c(dA) = (cd)A$$
$$c(A + B) = (cA) + (cB)$$

Matrix Multiplikation

= komposition af lineære transformationer

$$\begin{matrix} & & & & & j & & & & & j & & & \\ i & \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} & \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix} & = & i & \begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix} \end{matrix}$$

$n \times m$ matrix $\quad$ $m \times p$ matrix $\quad$ $n \times p$ matrix

$$c_{ij} = \sum_{k=1..m} a_{ik} \cdot b_{kj}$$

Regneregler

$$(AB)C = A(BC)$$
$$A(B + C) = (AB) + (AC)$$
$$(A + B)C = (AC) + (BC)$$

$$AB \neq BA$$

$$\begin{pmatrix} 2 & 1 & 4 \\ 1 & 3 & 2 \end{pmatrix} \begin{pmatrix} 3 & 1 \\ 2 & 2 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 12 & 20 \\ 11 & 15 \end{pmatrix}$$

$$\begin{pmatrix} 3 & 1 \\ 2 & 2 \\ 1 & 4 \end{pmatrix} \begin{pmatrix} 2 & 1 & 4 \\ 1 & 3 & 2 \end{pmatrix} = \begin{pmatrix} 7 & 6 & 14 \\ 6 & 8 & 12 \\ 6 & 13 & 12 \end{pmatrix}$$

Matrix Multiplikation

$$\begin{pmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{np} \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1m} \\ a_{21} & a_{22} & \cdots & a_{2m} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nm} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \cdots & b_{mp} \end{pmatrix}$$

$n \times p$ matrix $\quad$ $n \times m$ matrix $\quad$ $m \times p$ matrix

$=$

```
MATRIX-MULTIPLY(A, B)
1  if A.columns ≠ B.rows
2      error "incompatible dimensions"
3  else let C be a new A.rows × B.columns matrix
4      for i = 1 to A.rows
5          for j = 1 to B.columns
6              c_ij = 0
7              for k = 1 to A.columns
8                  c_ij = c_ij + a_ik · b_kj
9      return C
```

Naive implementation: tid $O(npm)$

(Kvadratisk) Matrix Multiplikation

[CLRS, kapitel 4.1-4.2]

$\frac{n}{2} \times \frac{n}{2}$ matrix

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

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

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

- $A, B, ..., K, L$ er $n/2$ x $n/2$-matricer
- I, J, K, L kan beregnes med
  8 rekursive multiplikationer og
  4 matrix additioner
  på $n/2$ x $n/2$ -matricer
- $T(n) = 8 \cdot T(n/2) + c \cdot n^2$ for $n \geq 2$

  $T(n) = c$ for $n = 1$
- $T(n) = O(n^{\log_2 8}) = O(n^3)$

Strassen's Matrix Multiplikation

1969

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

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

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

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

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

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

7 rekursive multiplikationer

Strassen's Matrix Multiplikation

- Bruger 18 matrix additioner (tid $O(n^2)$) og 7 rekursive matrix multiplikationer

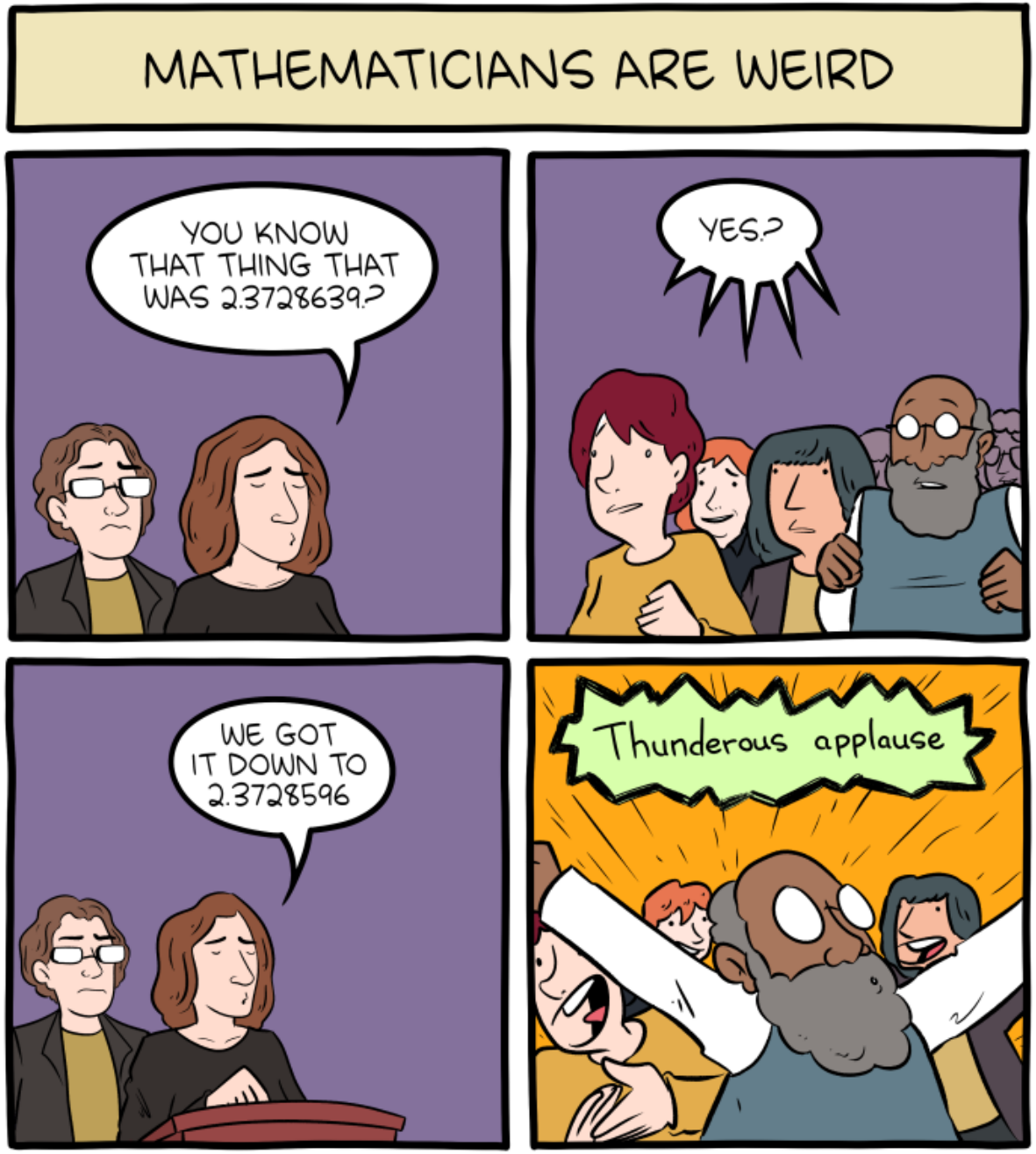
$$T(n) = 7 \cdot T(n/2) + c \cdot n^2 \quad \text{for } n \geq 2$$

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

- $T(n) = O(n^{\log_2 7}) = O(n^{2.8074})$

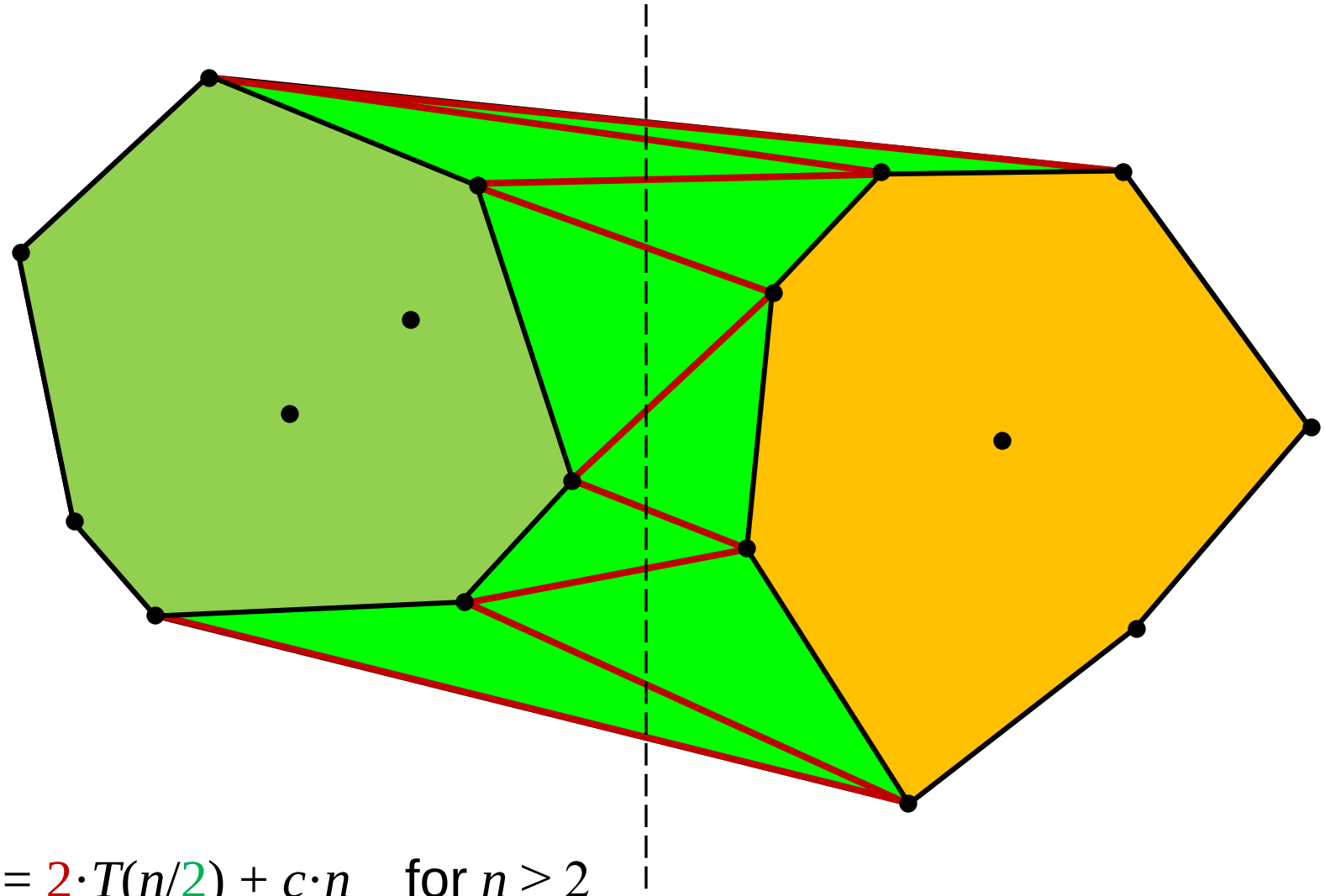
År	ω, $O(n^\omega)$
1969	2.8074
1978	2.796
1979	2.780
1981	2.522
1981	2.517
1981	2.496
1986	2.479
1990	2.3755
2010	2.3737
2013	2.3729
2014	2.3728639
2020	2.3728596
2022	2.371866
2023	2.371552
2025	2.371339

en.wikipedia.org/wiki/Computational_complexity_of_matrix_multiplication



https://www.smbc-comics.com/comic/mathematicians

Konveks Hylster

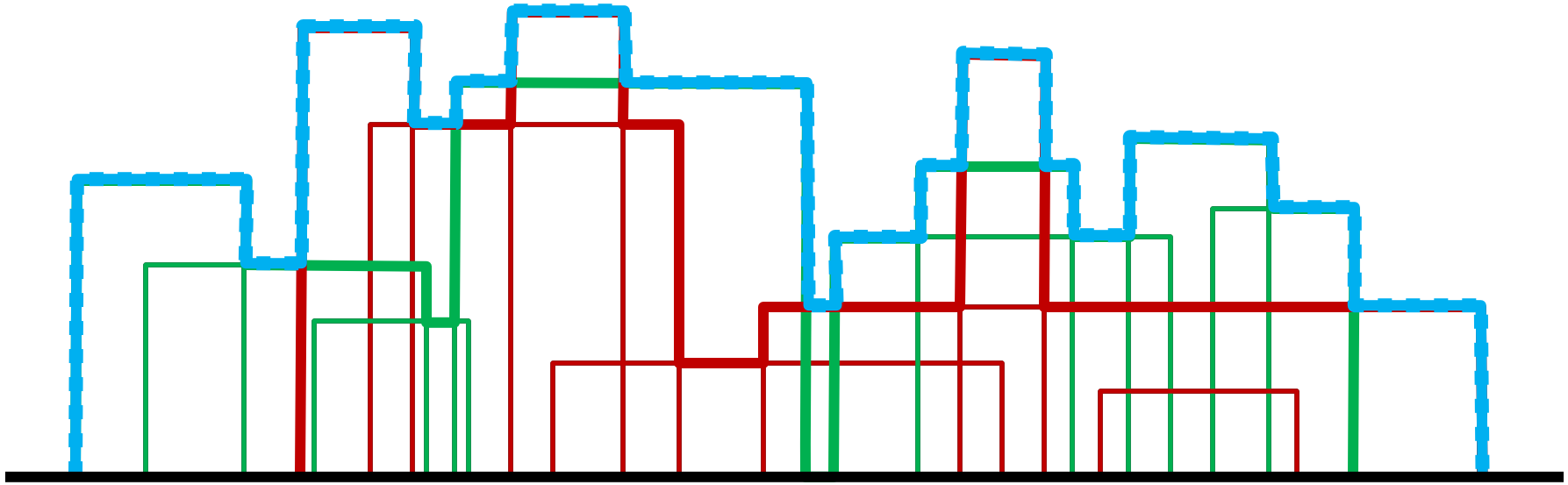


$T(n) = 2 \cdot T(n/2) + c \cdot n$ for $n \geq 2$

$T(n) = c$ for $n = 1$

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

Skyline
(afleveringsopgave)



$T(n) = ? \cdot T(n/?) + ?$ for $n \geq 2$

$T(n) = c$ for $n = 1$

Algoritmer og Datastrukturer

Selektion i worst-case lineær tid
[CLRS, kapitel 9.3]

Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan, 1973

Selektion

Find det i'te mindste element i en liste

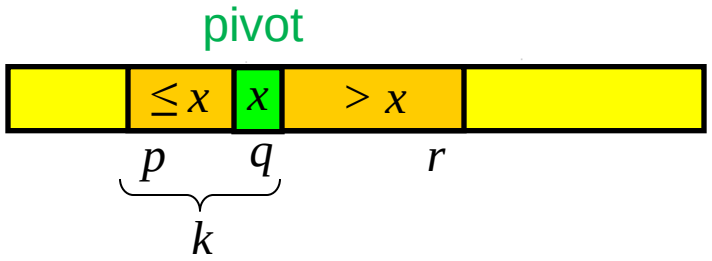
$L =$ | 10 | 5 | 12 | 3 | 1 | 7 | 42 | 9 | 15 |

$\textsc{Select}(L, 5) = 9$

Algoritme	Tid
Randomized-Select [CLRS, Kap. 9.2]	$O(n)$ forventet $O(n^2)$ worst-case
Deterministic-Select [CLRS, Kap. 9.3]	$O(n)$ worst-case

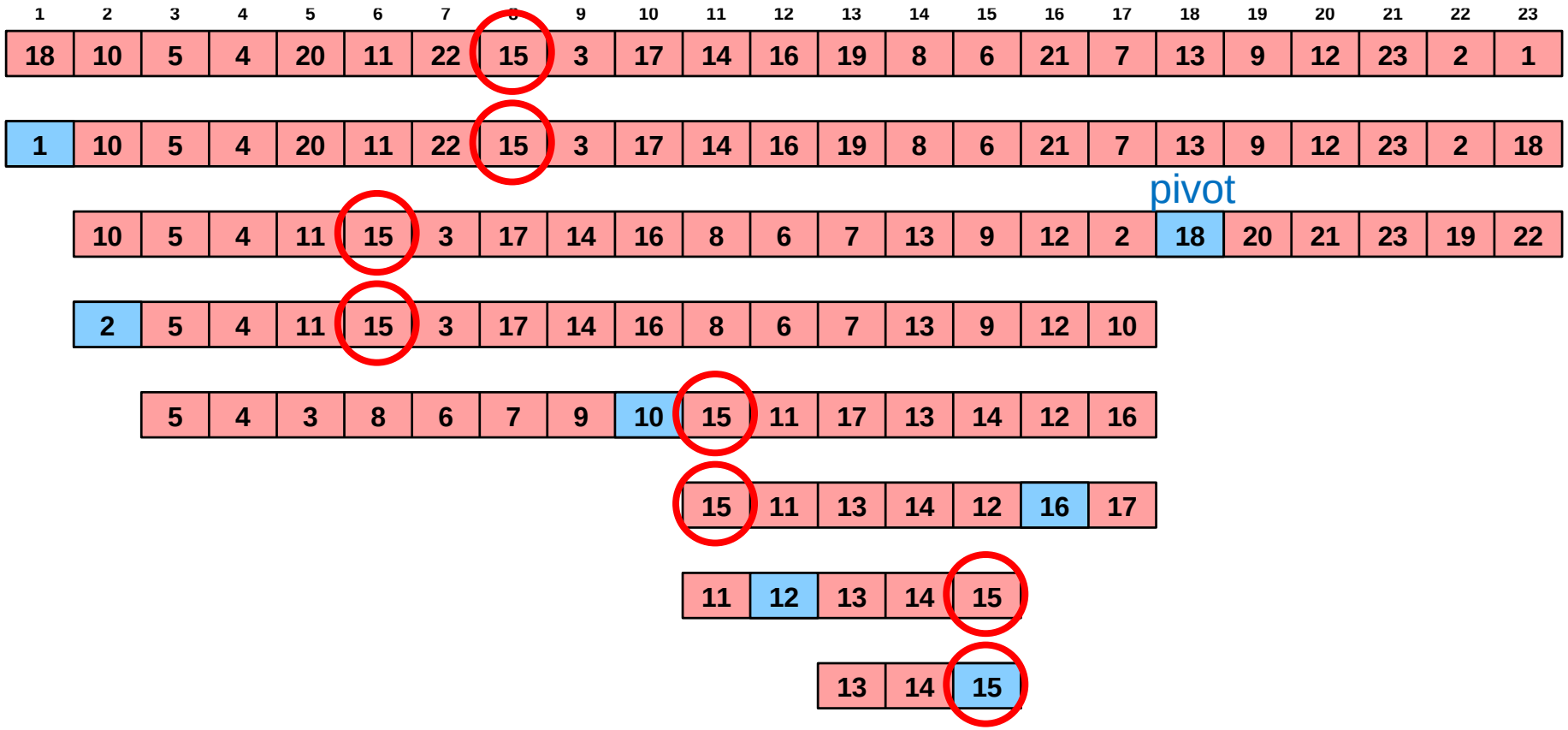
Randomized-Select:

Find det *i*'te mindste element i $A[p..r]$ $(1 \le i \le r\text{-}p+1)$




```
RANDOMIZED-SELECT(A, p, r, i)
1  if p == r
2       return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q − p + 1
5  if i == k            // the pivot value is the answer
6       return A[q]
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

- **Randomiseret** algoritme (vælger pivot tilfældig)
  - pivot vælges i midterste del med en vis sandsynlighed
- Eksempel på **del-og-kombiner**
  - kun 1 mindre delproblem løses rekursivt
  - hele tiden bruges i opdelingen
    (kombination returnerer blot resultatet fra rekursionen)
- Tid: worst-case $O(n^2)$, **forventet $O(n)$**
  - Analysen kan *ikke* anvende Master Teoremet

Deterministic-Select

- Samme idé som Randomized-Select
  - Vælg et element som pivot
  - Opdel m.h.t. pivot
  - Lav højst ét rekursivt kald på dem der er < eller > pivot
- Ny idé
  - Rekursivt brug Select til at finde god pivot
- Analyse
  - Del-og-kombiner

$$T(n) \leq T(a \cdot n) + T(b \cdot n) + c \cdot n$$

  - Kan ikke bruge Master teoremet ☹

Deterministic-Select

små input

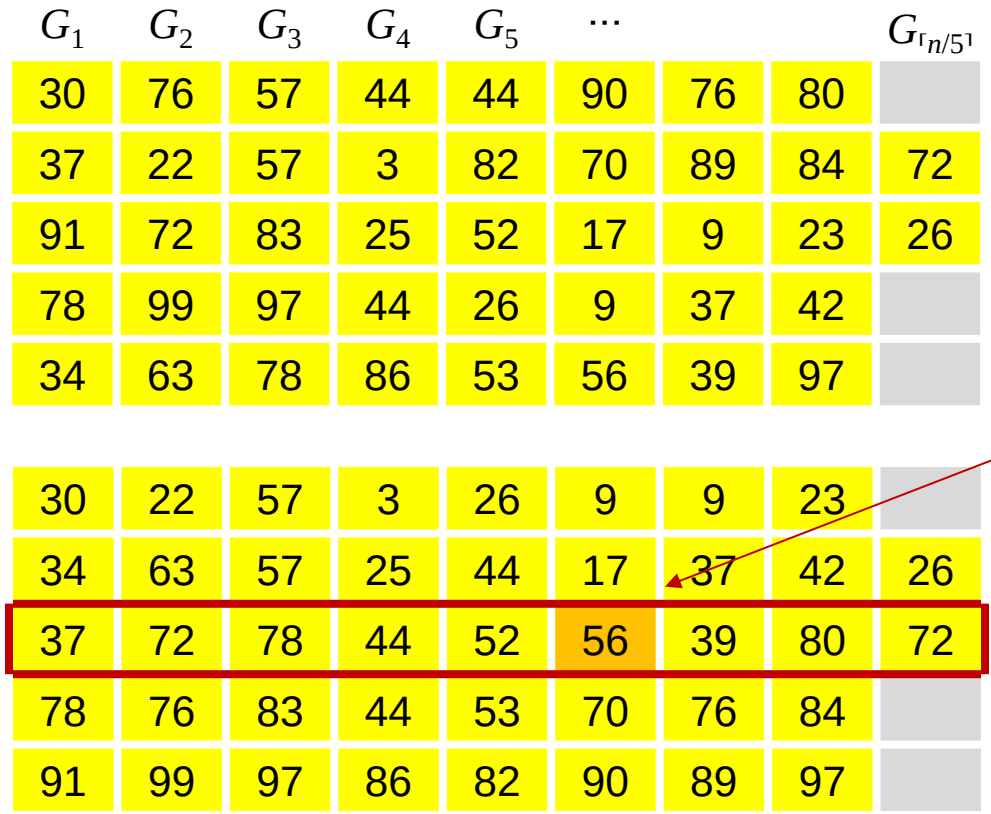
beregn pivot

max 1 rekursivt kald
(som randomized select)

```
SELECT(A, i)
1    if |A| ≤ 5 then
2        sort A and return i'th element
3    partition A into G_1,...,G_⌈n/5⌉ where |G_i| ≤ 5
4    medians = { g_i | g_i median of G_i }
5    pivot = SELECT(medians, ⌊|medians|/2⌋)
6    partition A w.r.t. pivot into A_< , A_= and A_>
7    if i ≤ |A_<| then
8        return SELECT(A_< , i)
9    if i > |A_<| + |A_=| then
10       return SELECT(A_> , i – |A_<| – |A_=|)
11   return pivot
```

Eksempel

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



betragt input som $\lceil n/5 \rceil$ grupper

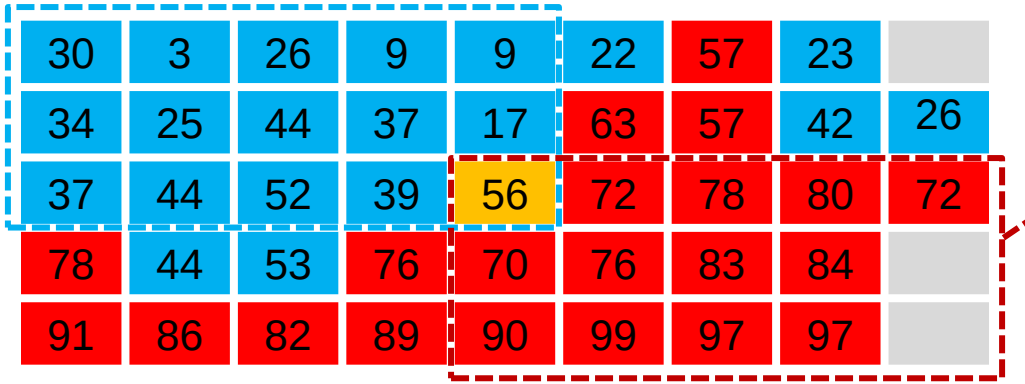
Sorter grupperne hver for sig

$pivot$ = median($medians$)

rekursivt find medianen

$medians$

Kvaliteten af *pivot* ?



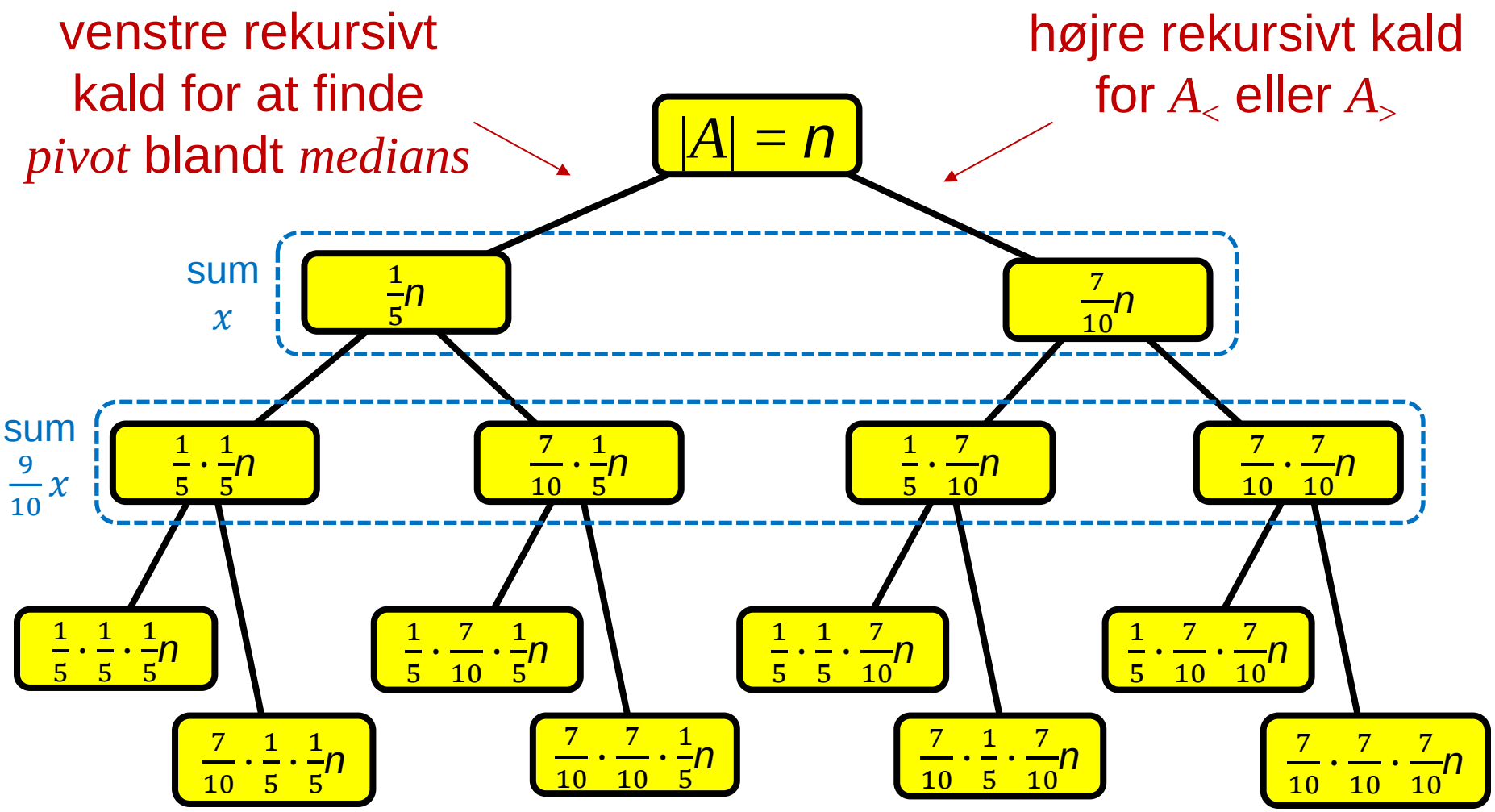
$\leq pivot$

grupperne permuteret så medianerne er opdelt m.h.t. *pivot*

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

$\geq pivot$

Rekursionstræ SELECT(A, i)



Note: Beviset ignorer at der til de rekursive kald kan være O(1) ekstra elementer når n ikke kan dividers med 5 og 10

Analyse

rekursivt fald for at bestemme *pivot*

rekursivt kald for $A_<$ eller $A_>$

tid for at finde medianen af hver af grupperne og at lave opdelingen i $A_<$, $A_=$ og $A_>$

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

tid for at sortere ≤ fem elementer

Note: Beviset ignorer at der til de rekursive kald kan være O(1) ekstra elementer når n ikke kan dividers med 5 og 10

Analyse

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

Bemærk i rekursionstræet er summen af størrelserne i dybde $i + 1$ højst $\frac{1}{5} + \frac{7}{10} = \frac{9}{10}$ gange størrelsen i dybde i

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

Note: Beviset ignorer at der til de rekursive kald kan være O(1) ekstra elementer når n ikke kan dividers med 5 og 10

Analyse (ved induktion)

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

Vis at $T(n) \leq k \cdot n$

Bevis

Basis $1 \leq n \leq 5$:

$T(n) \leq k \cdot n$ hvis $c \leq k$.

Induktionsskridt $n > 5$:

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

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

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

Note: Beviset ignorer at der til de rekursive kald kan være O(1) ekstra elementer når n ikke kan dividers med 5 og 10

Præcis Analyse

(tættere analyse end CLRS)

$|medians|$

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

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

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

Bevis: For $1 \leq n \leq 15$ udregn rekursionsligningen...

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

(fortsættes)

Præcis Analyse (fortsat)

For $n \geq 16$ bevis ved **induktion**.

Induktionshypotese (antag at vi allerede har bevist)

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

Induktionsskridt (vis for n)

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

□

Endnu mere Præcis Analyse :

Sammenligninger

beregne *medians*

beregne $|A_<|$, $|A_=|$ og $|A_>|$

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

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

Bevis: $n \leq 15$ check manuelt. $n \geq 16$ ved induktion.

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

Selektion

Algoritme	Tid
Randomized-Select [CLRS, Kap. 9.2] Hoare 1961	$O(n)$ forventet $O(n^2)$ worst-case
Deterministic-Select [CLRS, Kap. 9.3] Blum et al. 1973	$O(n)$ worst-case
Median worst-case sammenligninger Dor, Zwick 1995, 1996	$\leq 2.95n$ $\geq (2+\varepsilon)n$ $\varepsilon \approx 2^{-80}$

Algoritmer og Datastrukturer

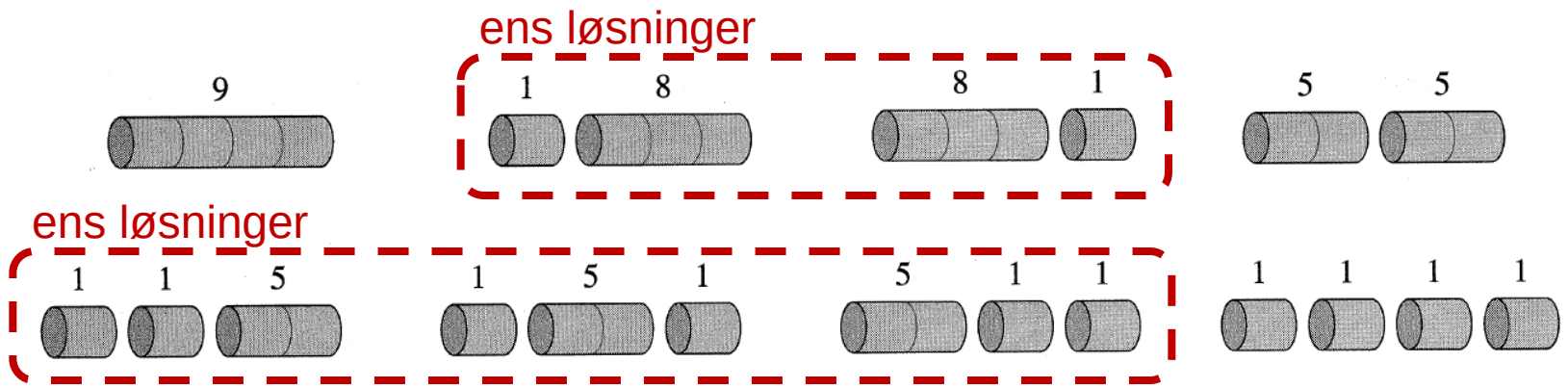
Dynamisk programmering
[CLRS, kapitel 14]

Dynamisk Programmering

- **Generel algoritmisk teknik** – virker for mange (men langt fra alle) problemer
- ***Krav***: ”Optimal delstruktur” – en løsning til problemet kan konstrueres ud fra optimale løsninger til ”delproblemer”
- **Rekursive løsning:**
  - Typisk eksponentiel tid
- **Dynamisk Programmering:**
  - Genbrug løsninger til delproblemer
  - Beregn delløsninger systematisk
  - Typisk polynomiel tid

Dynamisk Programmering: Optimal opdeling af en stang

Problem: Opdel en stang i dele, hvor hver del har en pris, således at den resulterende sum er maksimereret



En stang af længde 4 kan opdeles 8 forskellige måder – dog kun 5 forskellige resultater

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 opdeling af stænger af længde 1..10

Bedste pris	Opdeling
$r_1 = 1$	$1 = 1$ (no cuts),
$r_2 = 5$	$2 = 2$ (no cuts),
$r_3 = 8$	$3 = 3$ (no cuts),
$r_4 = 10$	$4 = 2 + 2$,
$r_5 = 13$	$5 = 2 + 3$,
$r_6 = 17$	$6 = 6$ (no cuts),
$r_7 = 18$	$7 = 1 + 6$ or $7 = 2 + 2 + 3$,
$r_8 = 22$	$8 = 2 + 6$,
$r_9 = 25$	$9 = 3 + 6$,
$r_{10} = 30$	$10 = 10$ (no cuts).

length i	1	2	3	4	5	6	7	8	9	10
price p_i	1	5	8	9	10	17	17	20	24	30

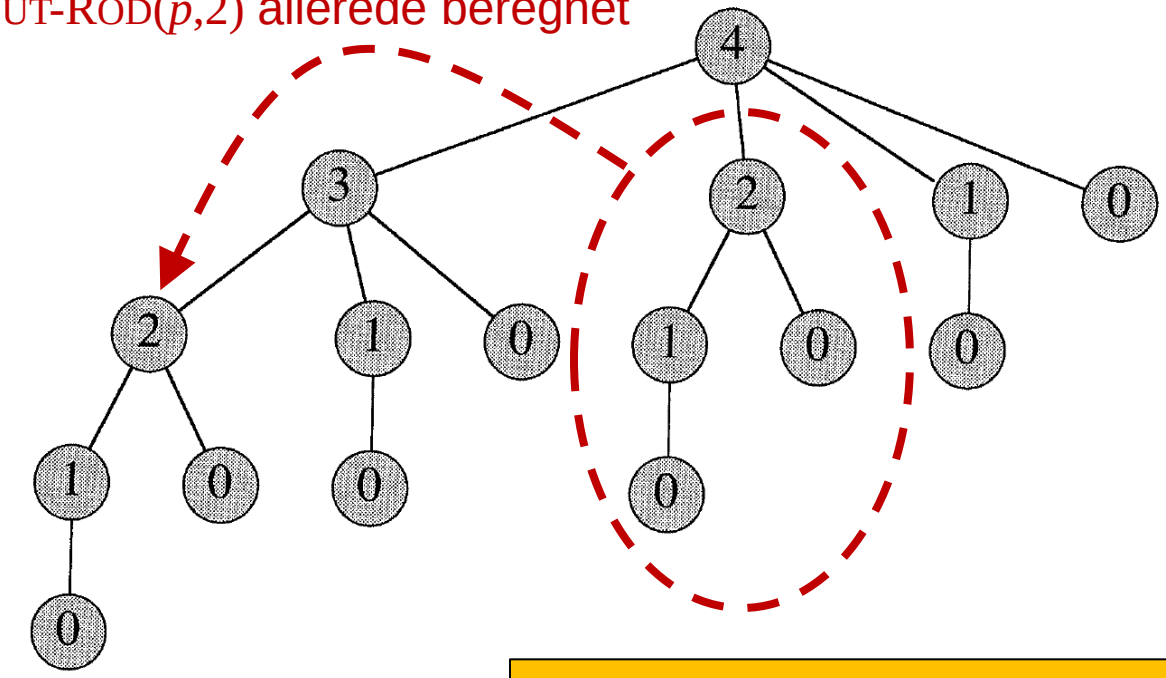
Optimal opdeling af en stang: Rekursiv løsning

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

```
CUT-ROD(p, n)
1  if n == 0
2      return 0
3  q = -∞
4  for i = 1 to n
5      q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

Optimal opdeling af en stang: Rekursiv løsning

CUT-ROD(p,2) allerede beregnet

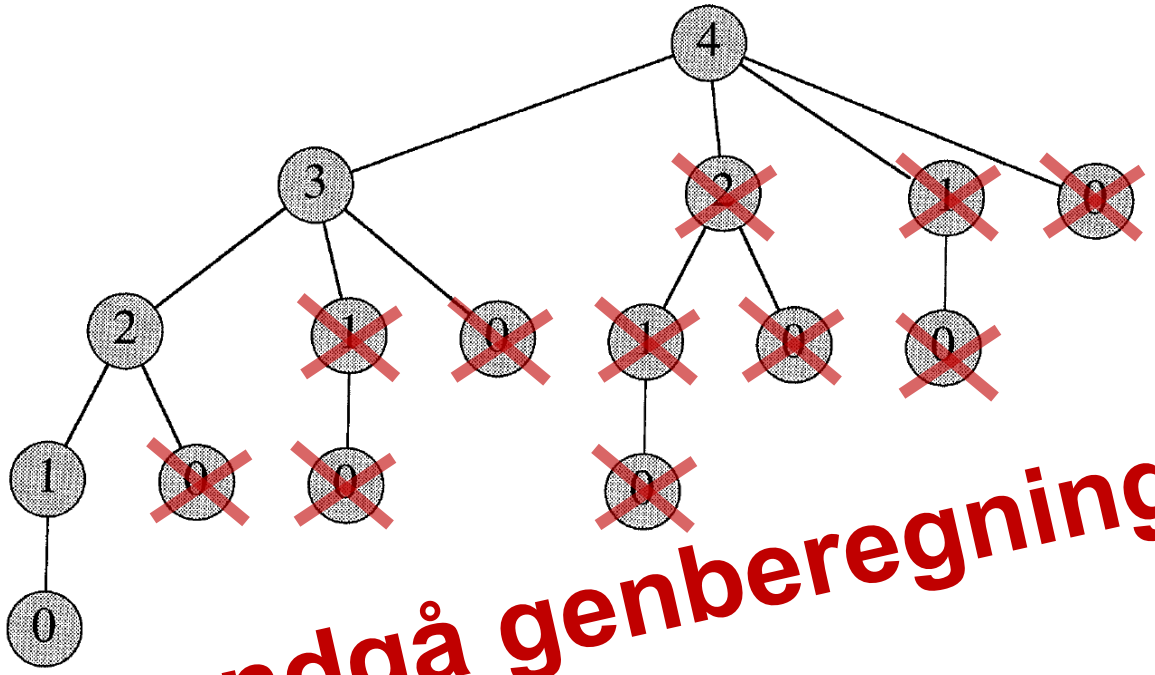



```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

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

Tid $O(2^n)$

Optimal opdeling af en stang: Rekursiv løsning



undgå genberegning ?

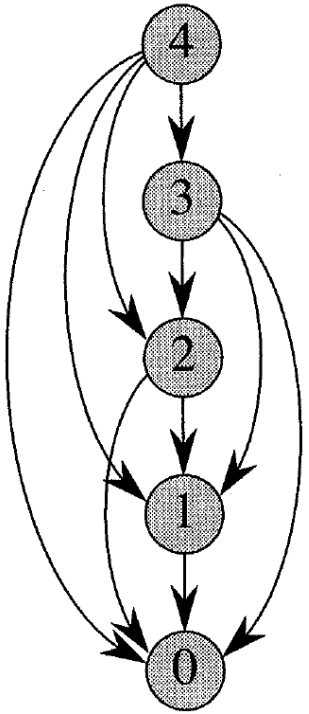
```
CUT-ROD(p, n)
1  if n == 0
2       return 0
3  q = -∞
4  for i = 1 to n
5       q = max(q, p[i] + CUT-ROD(p, n - i))
6  return q
```

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

Optimal opdeling af en stang: Rekursiv løsning + Memoization

```
MEMOIZED-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  for i = 0 to n
3      r[i] = -∞
4  return MEMOIZED-CUT-ROD-AUX(p, n, r)

MEMOIZED-CUT-ROD-AUX(p, n, r)
1  if r[n] ≥ 0
2      return r[n]
3  if n == 0
4      q = 0
5  else q = -∞
6      for i = 1 to n
7          q = max(q, p[i] + MEMOIZED-CUT-ROD-AUX(p, n - i, r))
8  r[n] = q
9  return q
```

husk resultatet !

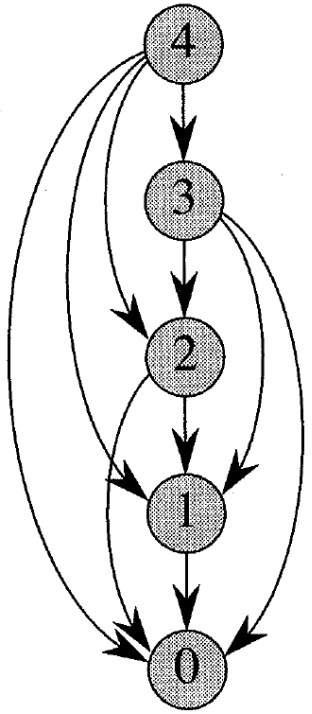


Tid $O(n^2)$

Optimal opdeling af en stang: Systematisk udfyldning

```
BOTTOM-UP-CUT-ROD(p, n)
1  let r[0..n] be a new array
2  r[0] = 0
3  for j = 1 to n
4      q = -∞
5      for i = 1 to j
6          q = max(q, p[i] + r[j - i])
7      r[j] = q
8  return r[n]
```

rækkefølgen vigtig !



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

Tid $O(n^2)$

Optimal opdeling af en stang: Udskrivning af løsningen

```
EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
 1  let r[0..n] and s[0..n] be new arrays
 2  r[0] = 0
 3  for j = 1 to n
 4      q = -∞
 5      for i = 1 to j
 6          if q < p[i] + r[j - i]
 7              q = p[i] + r[j - i]
 8              s[j] = i
 9      r[j] = q
10  return r and s
```

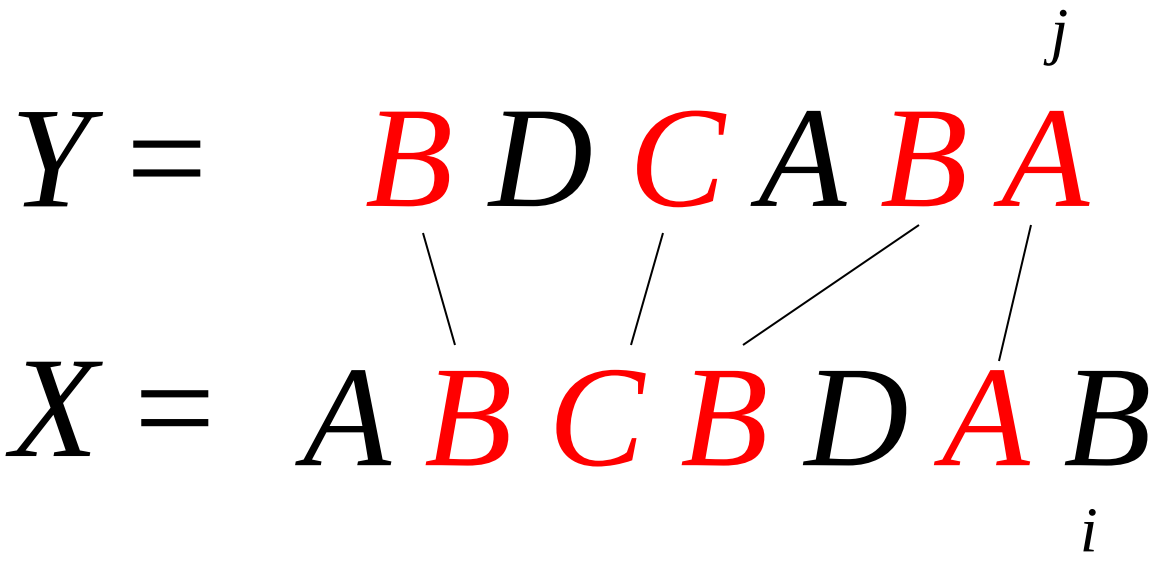
i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

```
PRINT-CUT-ROD-SOLUTION(p, n)
1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0
3      print s[n]
4      n = n - s[n]
```

Tid $O(n^2+n)$

Længste Fælles Delsekvens

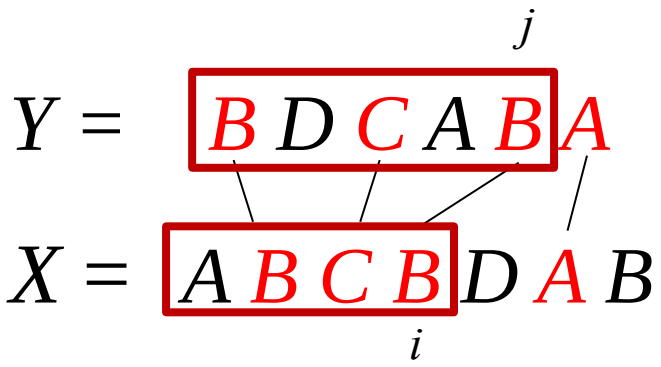
Længste Fælles Delsekvens



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

længden af en længste fælles delsekvens af $x_1x_2\cdots x_i$ og $y_1y_2\cdots y_j$

Længste Fælles Delsekvens

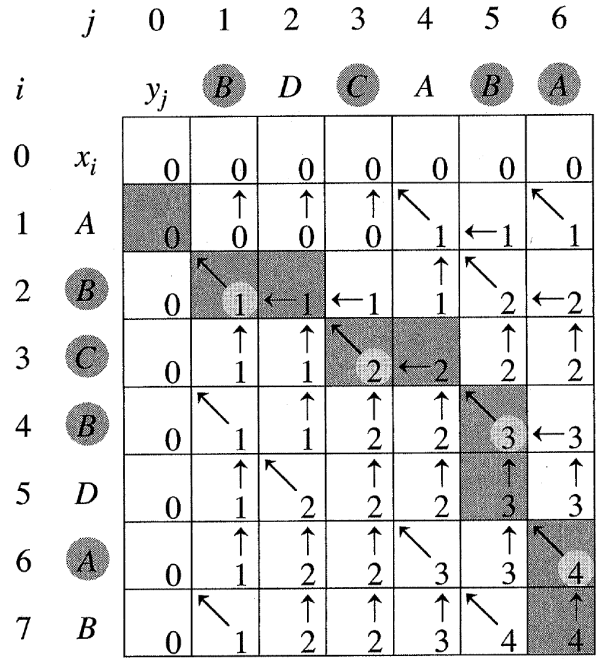


		j	0	1	2	3	4	5	6
i		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

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

Længste Fælles Delsekvens

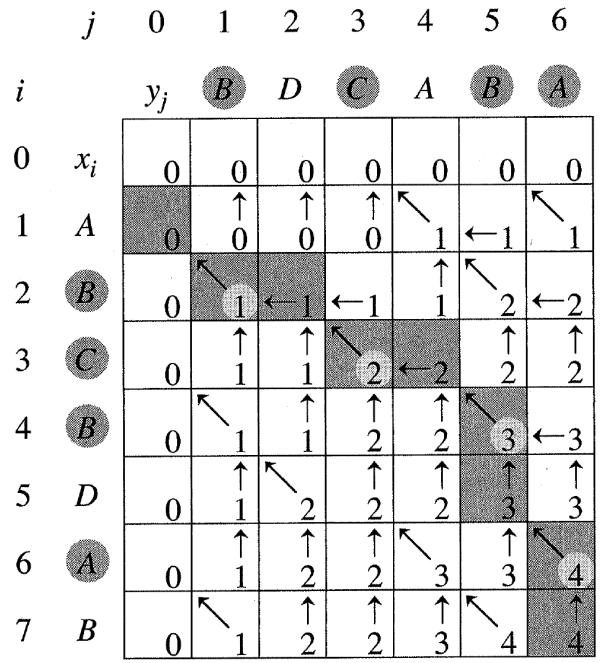
```
LCS-LENGTH(X, Y)
 1  m = X.length
 2  n = Y.length
 3  let b[1..m, 1..n] and c[0..m, 0..n] be new tables
 4  for i = 1 to m
 5      c[i, 0] = 0
 6  for j = 0 to n
 7      c[0, j] = 0
 8  for i = 1 to m
 9      for j = 1 to n
10          if x_i == y_j
11              c[i, j] = c[i − 1, j − 1] + 1
12              b[i, j] = "↖"
13          elseif c[i − 1, j] ≥ c[i, j − 1]
14              c[i, j] = c[i − 1, j]
15              b[i, j] = "↑"
16          else c[i, j] = c[i, j − 1]
17              b[i, j] = "←"
18  return c and b
```



Tid $O(nm)$

Længste Fælles Delsekvens

```
PRINT-LCS(b, X, i, j)
1  if i == 0 or j == 0
2       return
3  if b[i, j] == "↖"
4       PRINT-LCS(b, X, i − 1, j − 1)
5       print x_i
6  elseif b[i, j] == "↑"
7       PRINT-LCS(b, X, i − 1, j)
8  else PRINT-LCS(b, X, i, j − 1)
```



Tid $O(n+m)$

Ikke pensum

Beregning af en LCS i O(*n* + *m*) plads

Idé

Beregn stien **uden at huske** matricen

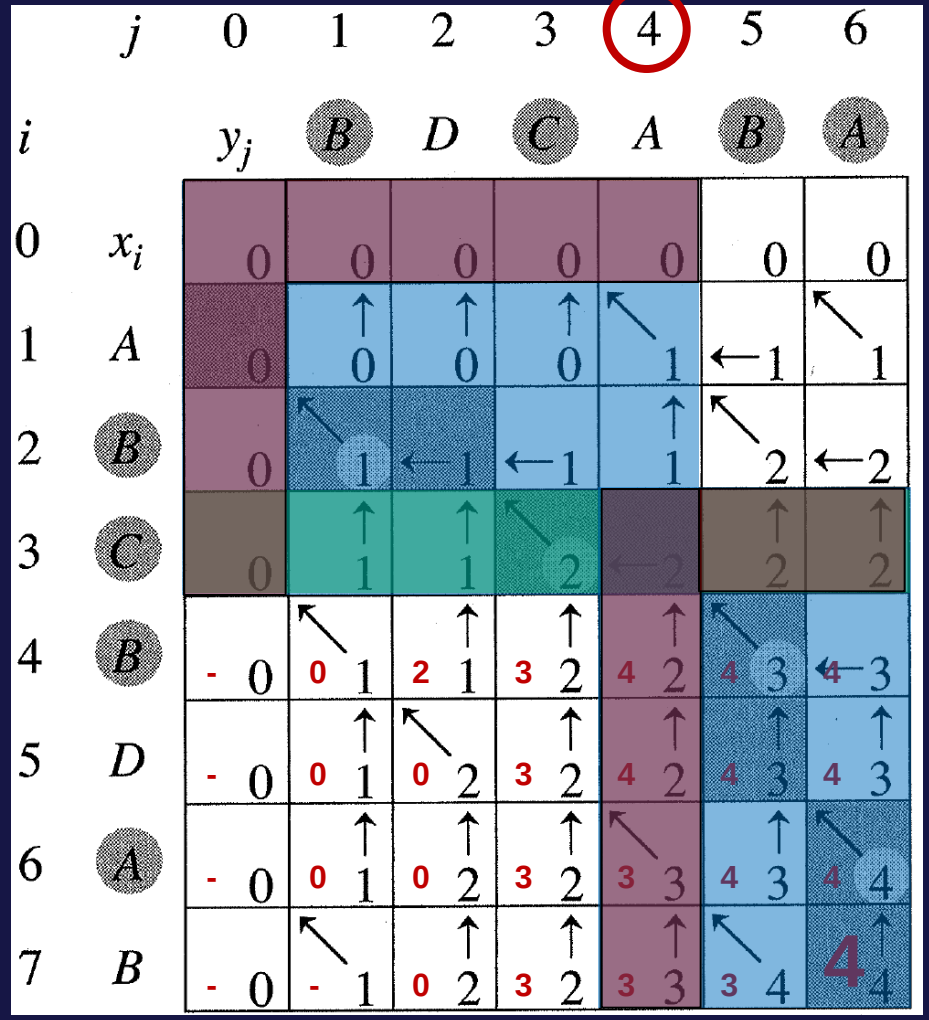
1) Beregn hvor stien krydser den midterste række

2) Beregn rekursivt de to del-stier

Tid O(*n*·*m*)

Plads O(*n*+*m*)

Eksempel på **del-og-kombiner** anvendt i en **dynamisk programmerings** algoritme



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

Hirschberg 1975

Længste Fælles Delsekvens

$O(n \cdot m)$ tid	dynamisk programmering
$O(n+m)$ plads	dynamisk programmering og del og kombiner [Hirschberg 1975]

Åben problem

Kan man løse længste fælles delsekvens problemet for to strenge af længde n i tid $O(n^{2-\varepsilon})$?

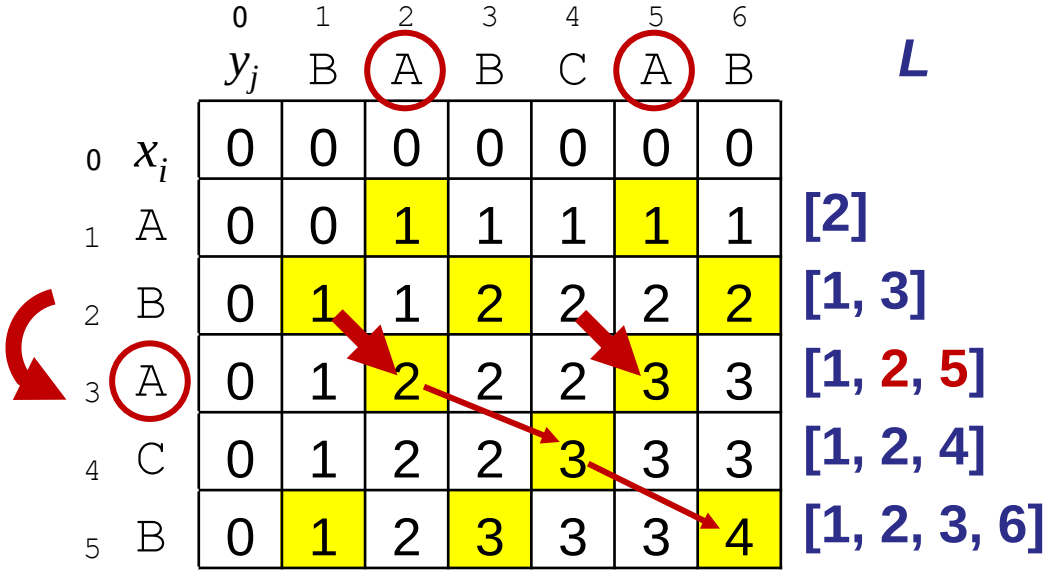
– Det ville bryde med den såkaldte Strong Exponential Time Hypothesis

Abboud, Backurs, Williams. Tight Hardness Results for LCS and Other Sequence Similarity Measures.
In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 59–78, 2015.

Bringmann and Künnemann. Quadratic Conditional Lower Bounds for String Problems and Dynamic Time Warping.
In *Proc. 56th Annual IEEE Symposium on Foundations of Computer Science (FOCS'15)*, pages 79–97, 2015.

Ikke pensum

LCS – Hunt-Szymanski (1976)



		0	1	2	3	4	5	6	**L**
		y_j	B	A	B	C	A	B	
0	x_i	0	0	0	0	0	0	0	
1	A	0	0	1	1	1	1	1	[2]
2	B	0	1	1	2	2	2	2	[1, 3]
3	A	0	1	2	2	2	3	3	[1, 2, 5]
4	C	0	1	2	2	3	3	3	[1, 2, 4]
5	B	0	1	2	3	3	3	4	[1, 2, 3, 6]

Antag **få parvise matches**

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

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

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

- Beregn *ikke* LCS tabellen eksplicit; kig kun på matches
- **LCS(*X*, *Y*) = LIS(Matches(*X*, *Y*))**
  Sorter voksende *i*, aftagende *j*; LIS = længste voksende delsekvens m.h.t. *j*
- |LCS|: Beregn for hver række ***L*** = den første kolonne indeholdende 1, 2, 3, …
  og kun ændring i ***L*** for hver række – totalt *r* binære søgninger i ***L***
  Tid O((*m* + *n* + *r*) · log min(*n*, *m*)), plads O(min(*n*, *m*))

Tekstformatering – Linieskift

https://tex.stackexchange.com/questions/120271/alternatives-to-latex/120279#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 noth-ing particular to interest me on shore, I thought I would sail about a little and see the wa-tery part of the world. It is a way I have of driving off the spleen, and regulating the cir-culation. Whenever I find my-self growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself invol-untarily pausing before coffin

Adobe InDesign CS4

Call me Ishmael. Some years ago – never mind how long pre-cisely – having little or no mon-ey 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 grow-ing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenev-er I find myself involuntarily pausing before coffin warehous-es, and bringing up the rear of

pdf-LaTeX 3.1415926

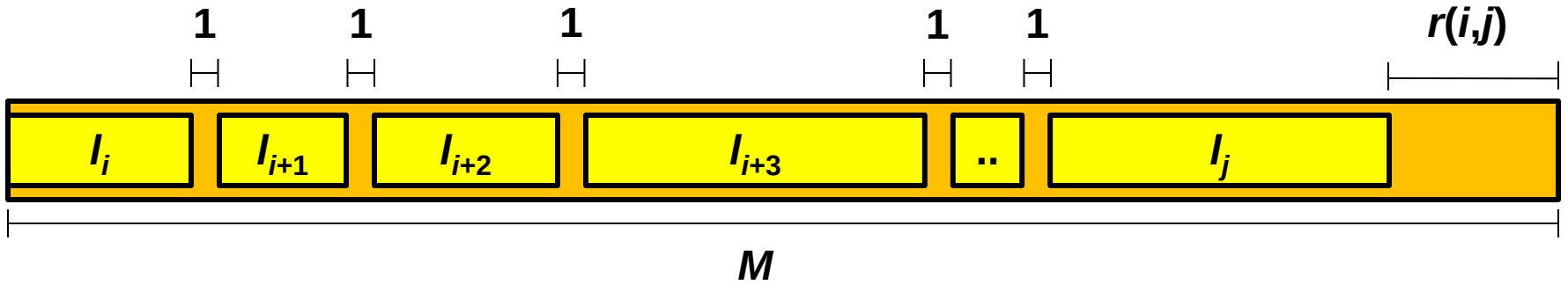
Call me Ishmael. Some years ago – never mind how long precisely – having little or no money in my purse, and noth-ing particular to interest me on shore, I thought I would sail about a little and see the watery part of the world. It is a way I have of driving off the spleen, and regulating the circulation. Whenever I find myself growing grim about the mouth; whenever it is a damp, drizzly November in my soul; whenever I find myself invol-untarily pausing before coffin warehouses, and bringing up the

Problem CLRS 14-4 Printing neatly

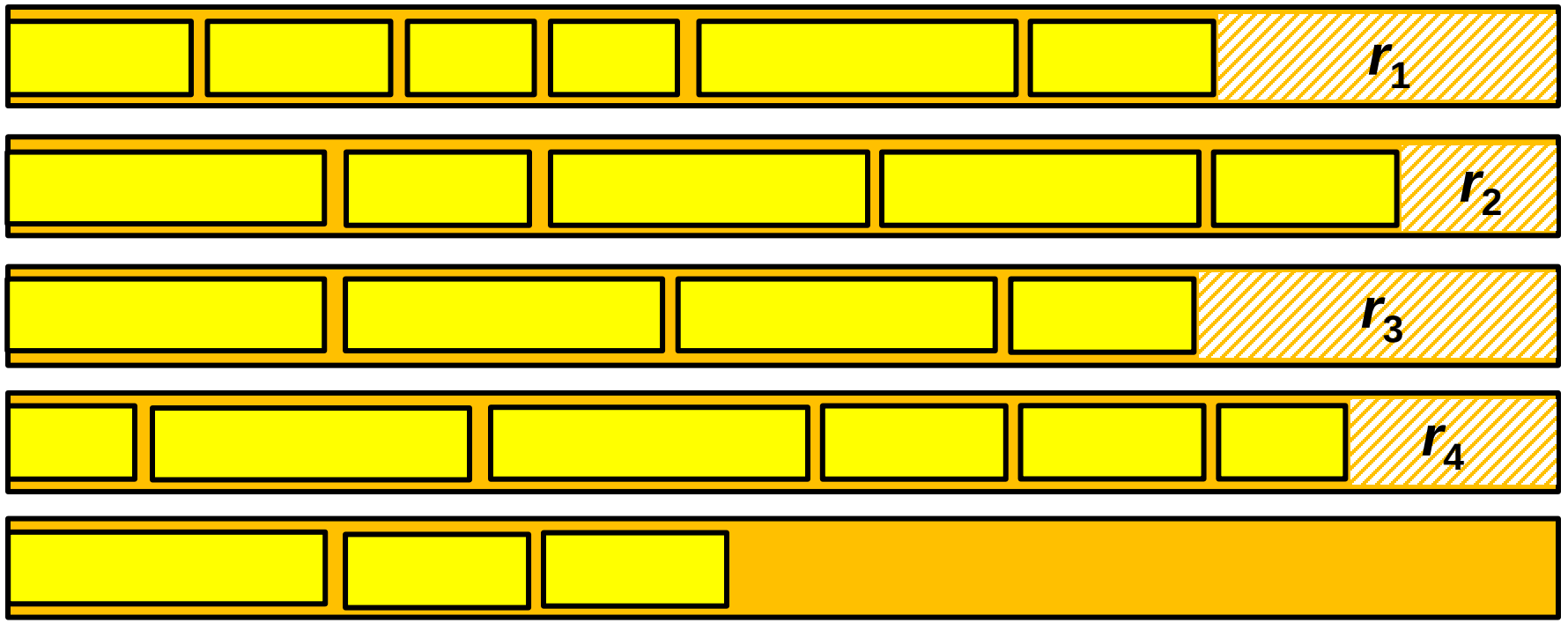
“Consider the problem of neatly printing a paragraph with a monospaced font (all characters having the same width) on a printer. The input text is a sequence of n words of lengths l_1, l_2,..., l_n, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum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

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

which must be nonnegative so that the words fit on the line. We wish to **minimize** the sum, over all lines except the last, of **the cubes of the numbers of extra space characters at the ends of lines**. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time and space requirements of your algorithm.“



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



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

Problem CLRS 14-4 Printing neatly

Pris for at udskrive de første j ord på hele linier:

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

Bedste løsning er:

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

ord j+1,j+2,...,n på sidste linie

Printing neatly

```
 1 // beregn 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 // beregn 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
```

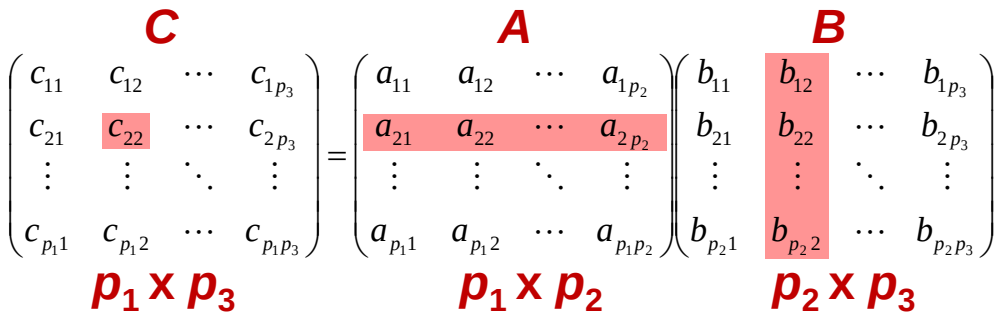
```
15 // udskriv bedste løsning
16 proc udskriv(j)
17   if j > 0 then
18     i = I[j]
19     udskriv(i - 1)
20     print ord[i]..ord[j] <newline>

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 udskriv(k)
26 print ord[k+1]..ord[n] <newline>
```

Tid og plads $O(n^2)$

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

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

Matrix Multiplikation




```
MATRIX-MULTIPLY(A, B)
1  if A.columns ≠ B.rows
2      error "incompatible dimensions"
3  else let C be a new A.rows × B.columns matrix
4      for i = 1 to A.rows
5          for j = 1 to B.columns
6              c_ij = 0
7              for k = 1 to A.columns
8                  c_ij = c_ij + a_ik · b_kj
9      return C
```

Multiplikation af to matricer A og B af størrelse p_1 x p_2 og p_2 x p_3 tager tid $O(p_1 \cdot p_2 \cdot p_3)$

Matrix-kæde Multiplikation

$$(A \cdot B) \cdot C \text{ eller } A \cdot (B \cdot C) \ ?$$

Matrix multiplikation er associativ (kan sætte parentesser som man vil) men ikke kommutativ (kan ikke bytte rundt på rækkefølgen af matricerne)

Matrix-kæde Multiplikation

Problem: Find den bedste rækkefølge (parentesser) for at gange n matricer sammen

$$A_1 \cdot A_2 \cdots A_n$$

hvor A_i er en $p_{i\text{-}1}$ x p_i matrix

NB: Der er $\Omega(4^n/n^{3/2})$ mulige måder for parentesserne

Matrix-kæde Multiplikation

$m[i, j]$ = minimale antal (primitive) multiplikationer for at beregne $A_i \cdot \cdots \cdot A_j$

$p_{i-1} \times p_k$ $\quad$ $p_k \times p_j$

$(A_i \cdots A_k) \cdot (A_{k+1} \cdots A_j)$

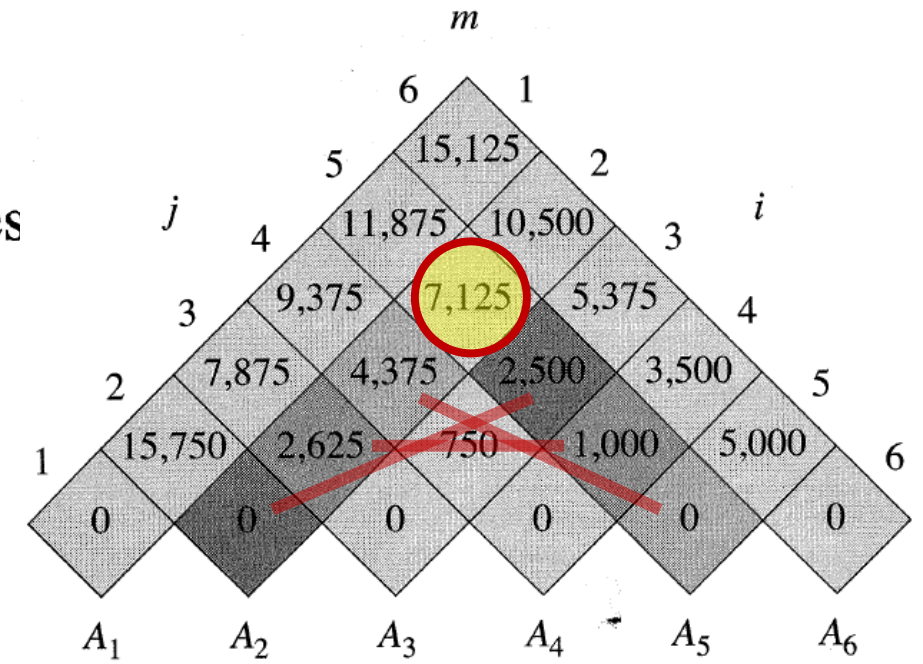
$$m[i, j] = \begin{cases} 0 & \text{if } i = j \,, \\ \min\limits_{i \le k < j} \{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\} & \text{if } i < j \,. \end{cases}$$

```
RECURSIVE-MATRIX-CHAIN(p, i, j)
1  if i == j
2       return 0
3  m[i, j] = ∞
4  for k = i to j − 1
5       q = RECURSIVE-MATRIX-CHAIN(p, i, k)
              + RECURSIVE-MATRIX-CHAIN(p, k + 1, j)
              + p_{i−1} p_k p_j
6       if q < m[i, j]
7            m[i, j] = q
8  return m[i, j]
```

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

Matrix-kæde Multiplikation

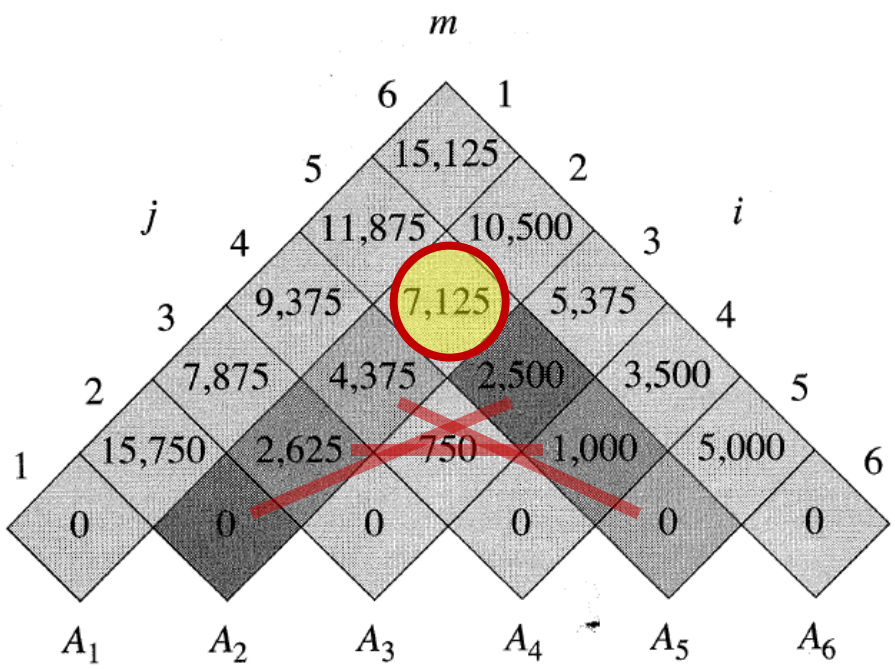
```
MATRIX-CHAIN-ORDER(p)
 1  n = p.length − 1
 2  let m[1..n, 1..n] and s[1..n − 1, 2..n] be new tables
 3  for i = 1 to n
 4      m[i, i] = 0
 5  for l = 2 to n                    // l is the chain length
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          m[i, j] = ∞
 9          for k = i to j − 1
10              q = m[i, k] + m[k + 1, j] + p_{i−1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] = q
13                  s[i, j] = k
14  return m and s
```

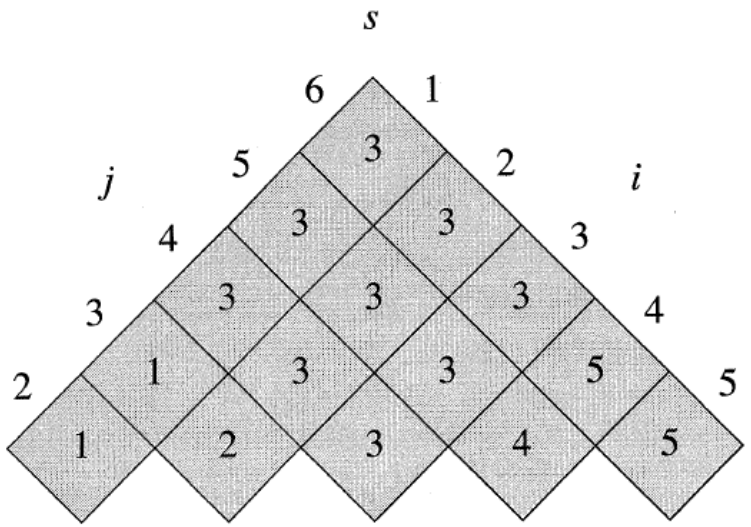


Tid $O(n^3)$

$$m[i,j] = \begin{cases} 0 & \text{if } i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

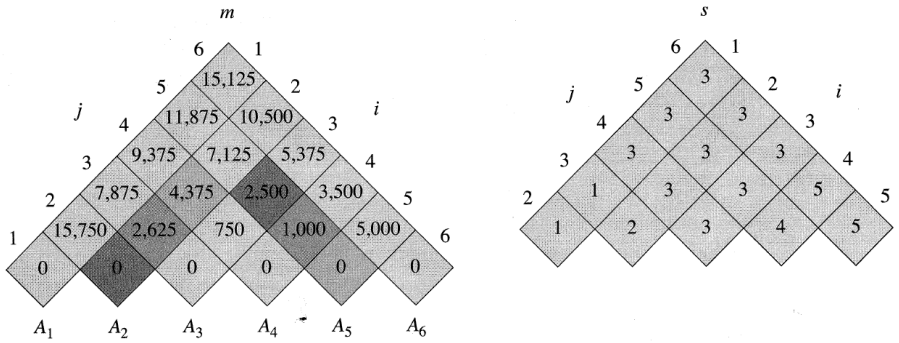
Matrix-kæde Multiplikation





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-kæde Multiplikation




```
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 ")"
```

Tid O(n)

”Memoized” Matrix-kæde Multiplikation

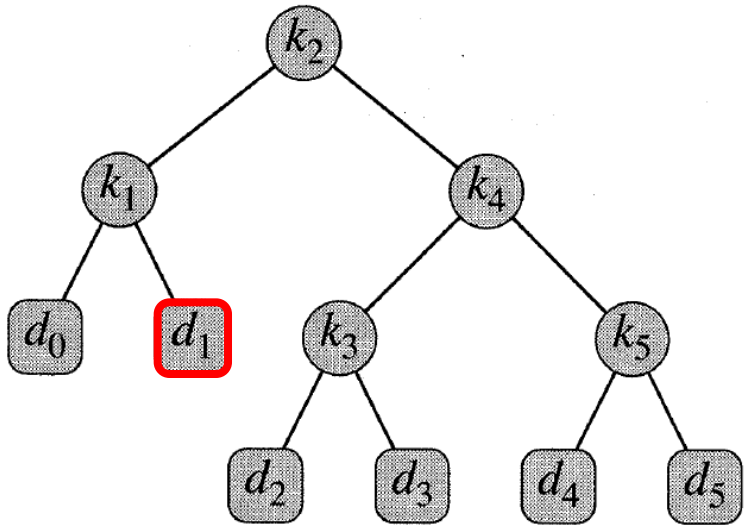
```
MEMOIZED-MATRIX-CHAIN(p)
1  n = p.length − 1
2  let m[1..n, 1..n] be a new table
3  for i = 1 to n
4      for j = i to n
5          m[i, j] = ∞
6  return LOOKUP-CHAIN(m, p, 1, n)
```

$$m[i,j] = \begin{cases} 0 & \text{if } i = j\,, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1} p_k p_j\} & \text{if } i < j\,. \end{cases}$$

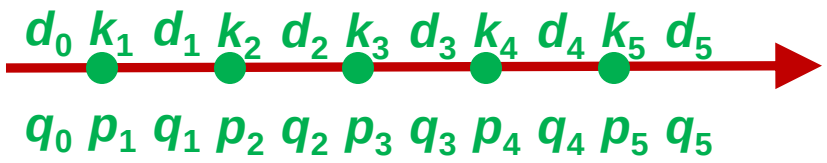
```
LOOKUP-CHAIN(m, p, i, j)
1  if m[i, j] < ∞
2      return m[i, j]
3  if i == j
4      m[i, j] = 0
5  else for k = i to j − 1
6          q = LOOKUP-CHAIN(m, p, i, k)
                + LOOKUP-CHAIN(m, p, k + 1, j) + p_{i−1} p_k p_j
7          if q < m[i, j]
8              m[i, j] = q
9  return m[i,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

Tid $O(n^3)$

Optimale Binære Søgetræer



Forventet søgetid 2.80

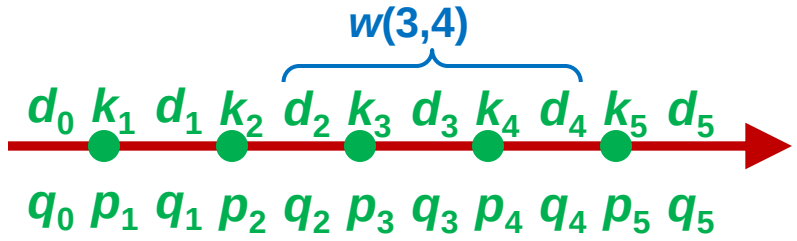


node	depth	probability	contribution
k_1	1	0.15	0.30
k_2	0	0.10	0.10
k_3	2	0.05	0.15
k_4	1	0.10	0.20
k_5	2	0.20	0.60
d_0	2	0.05	0.15
d_1	(2 + 1) x	0.10 =	0.30
d_2	3	0.05	0.20
d_3	3	0.05	0.20
d_4	3	0.05	0.20
d_5	3	0.10	0.40
Total			2.80

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

Optimale Binære Søgetræer

$$\sum_{i=1}^{n} p_i + \sum_{i=0}^{n} q_i = 1 \qquad w(i, j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l$$

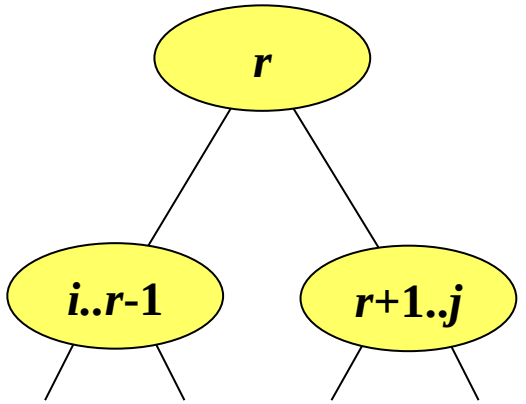


$$\mathrm{E}[\text{search cost in } T] = \sum_{i=1}^{n} (\mathrm{depth}_T(k_i) + 1) \cdot p_i + \sum_{i=0}^{n} (\mathrm{depth}_T(d_i) + 1) \cdot q_i$$

$$= 1 + \sum_{i=1}^{n} \mathrm{depth}_T(k_i) \cdot p_i + \sum_{i=0}^{n} \mathrm{depth}_T(d_i) \cdot q_i$$

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

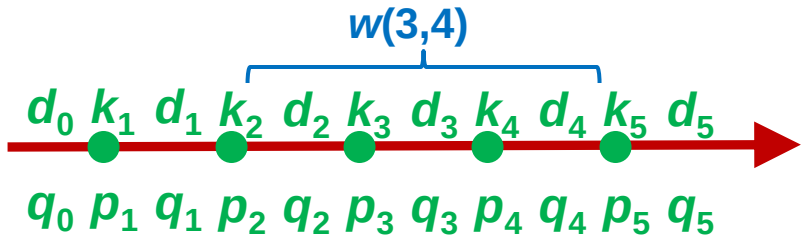
forventet optimal tid for et søgetræ indeholdende k_i...,k_j og d_{i-1}...,d_j

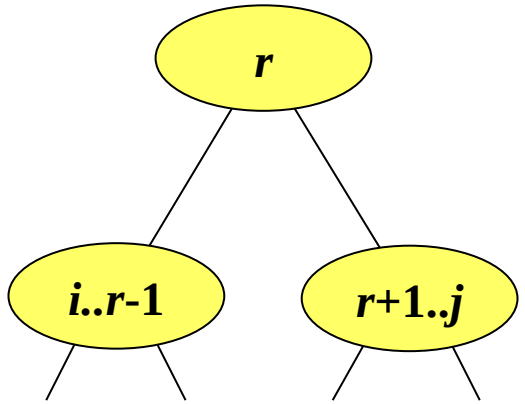


Optimale Binære Søgetræer

```
OPTIMAL-BST(p, q, n)
 1  let e[1..n+1, 0..n], w[1..n+1, 0..n],
        and root[1..n, 1..n] be new tables
 2  for i = 1 to n + 1
 3      e[i, i − 1] = q_{i−1}
 4      w[i, i − 1] = q_{i−1}
 5  for l = 1 to n
 6      for i = 1 to n − l + 1
 7          j = i + l − 1
 8          e[i, j] = ∞
 9          w[i, j] = w[i, j − 1] + p_j + q_j
10          for r = i to j
11              t = e[i, r − 1] + e[r + 1, j] + w[i, j]
12              if t < e[i, j]
13                  e[i, j] = t
14                  root[i, j] = r
15  return e and root
```

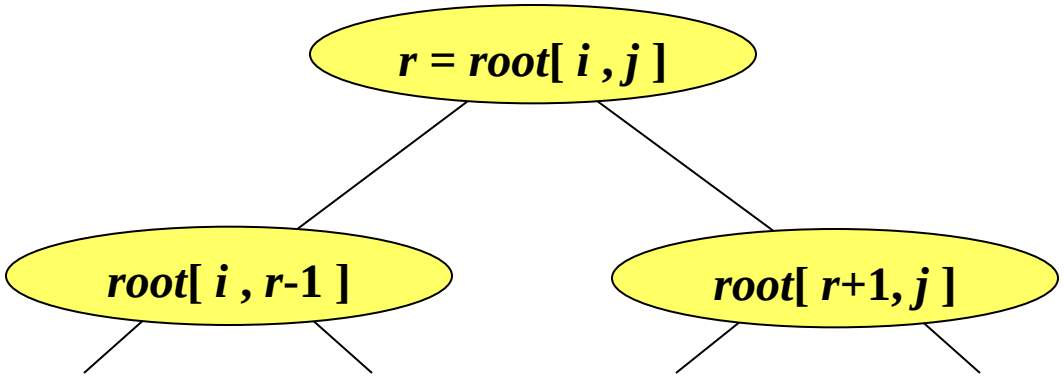
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1, \\ \min\limits_{i \le r \le j} \{e[i, r-1] + e[r+1, j] + w(i, j)\} & \text{if } i \le j. \end{cases}$$

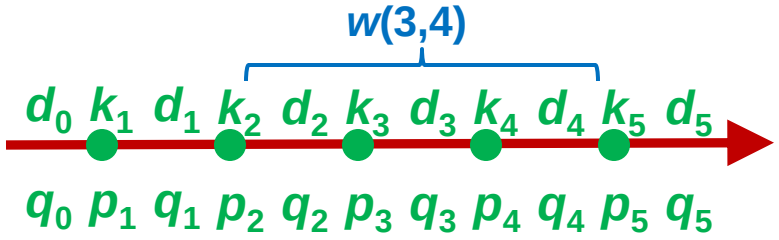


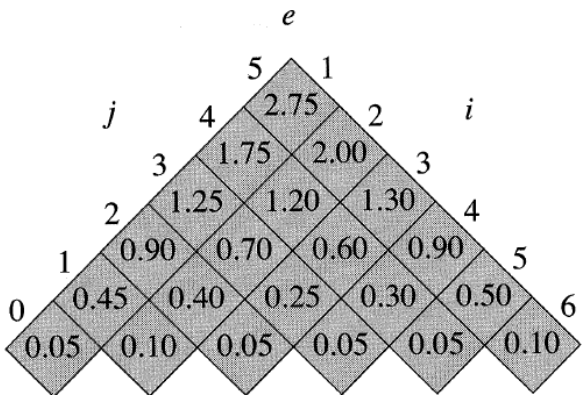


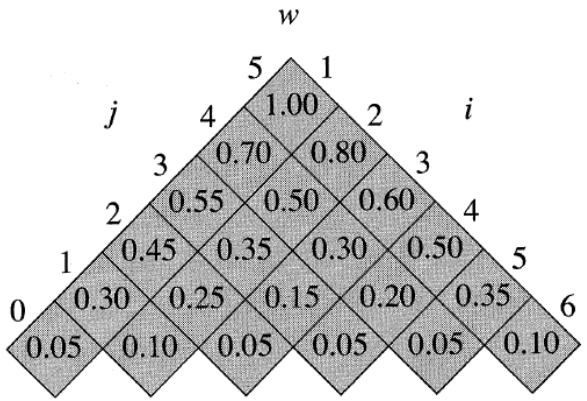
Tid $O(n^3)$

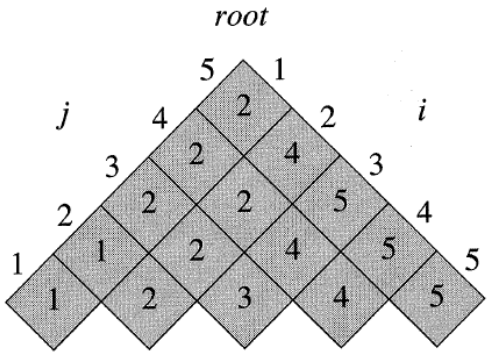
Konstruktion af Optimalt Binært Søgetræ











Dynamisk Programmering

- **Generel algoritmisk teknik**
- ***Krav***: ”Optimal delstruktur” – en løsning til problemet kan konstrueres ud fra optimale løsninger til ”delproblemer”
- **Rekursionsligning**
- **Eksempler**
  - Stang opdeling
  - Længste fælles delsekvens
  - ”Printing neatly”
  - Matrix-kæde multiplikation
  - Optimale søgetræer

Algoritmer og Datastrukturer

Grådige algoritmer
[CLRS, kapitel 15.1-15.3]

Grådige Algoritmer

- Problemer hvor en løsning kan konstrueres ud fra en løsning for kun **ét mindre delproblem**

- Delproblemet kan identificeres effektivt

(simplere end dynamisk programmering)

Er *BACBABA* en delsekvens af *ABABGACBABAD* ?

Mulig forekomst ***ABABGACBABAD***

Observation Hvis der findes en forekomst, så findes også en forekomst der matcher det første ***B*** i strengen

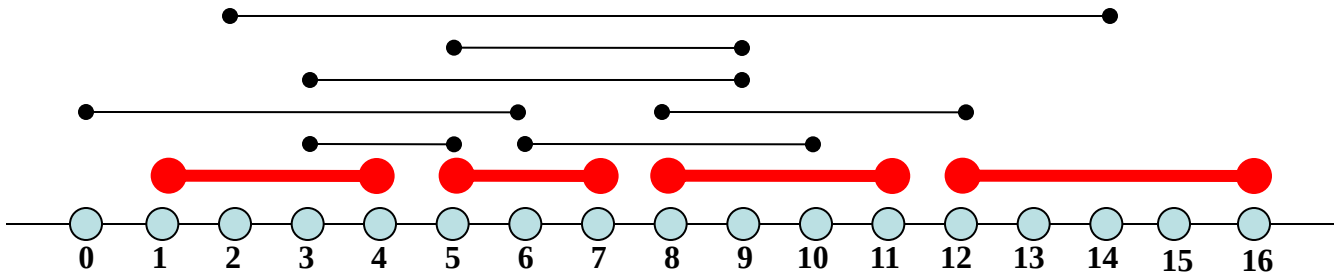
ABABGACBABAD

Find rekursivt ***ACBABA***

Tid $O(n)$

Udvælgelse af Aktiviteter

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

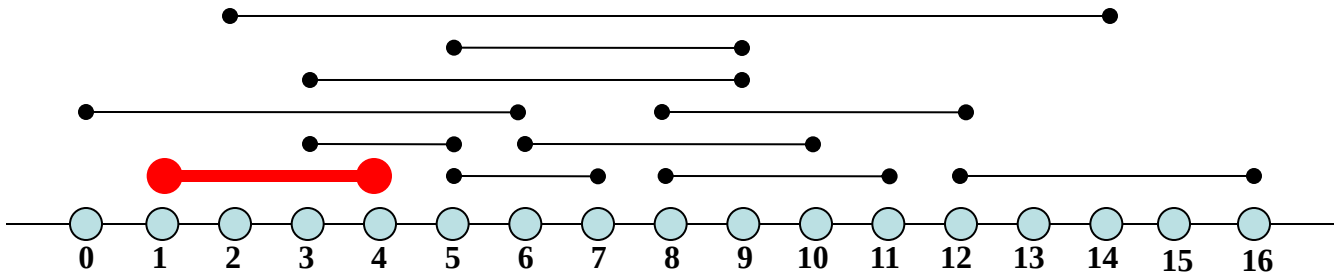


Input: n aktiviter med starttid s_i og sluttid f_i

Output: Maximal mængde ikke-overlappende aktiviteter

Udvælgelse af Aktiviteter

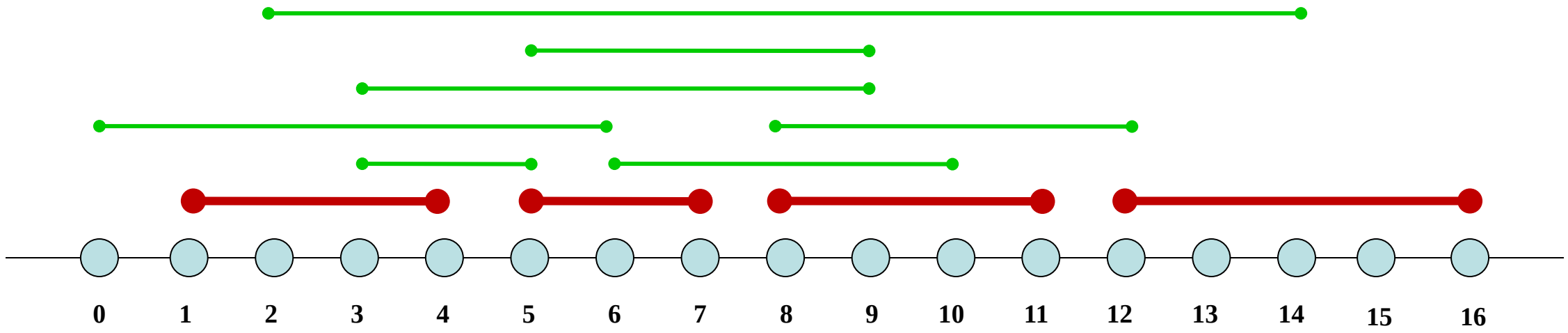
i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	9	9	10	11	12	14	16



Observation: Der findes altid en maximal løsning som indeholder aktiviteten med den *tidligste sluttid (grådig-valg egenskab)*

Preprocessering: Sorter aktiviteterne efter sluttidspunktet

Grådig sletning



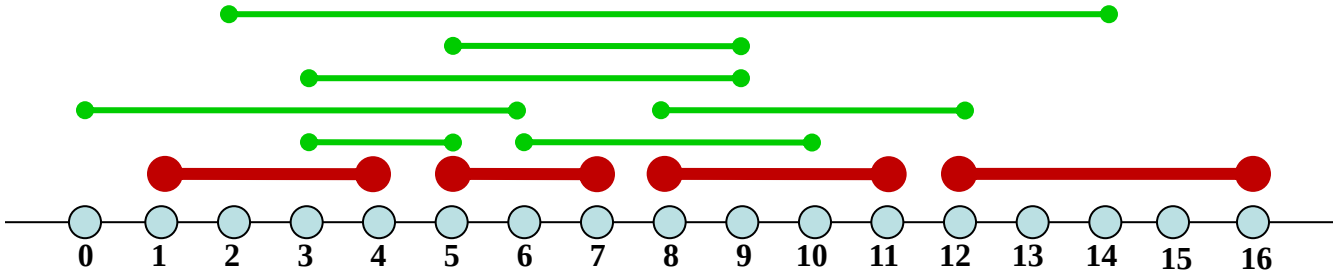
Udvælgelse af Aktiviteter

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

$\leftarrow$ sorteret

GREEDY-ACTIVITY-SELECTOR(s, f)

```
1  n = s.length
2  A = {a_1}
3  k = 1
4  for m = 2 to n
5      if s[m] ≥ f[k]
6          A = A ∪ {a_m}
7          k = m
8  return A
```



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

Huffman koder

Ascii Tabel

$83_{10} = 53_{16} = 101\ 0011_2$

	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

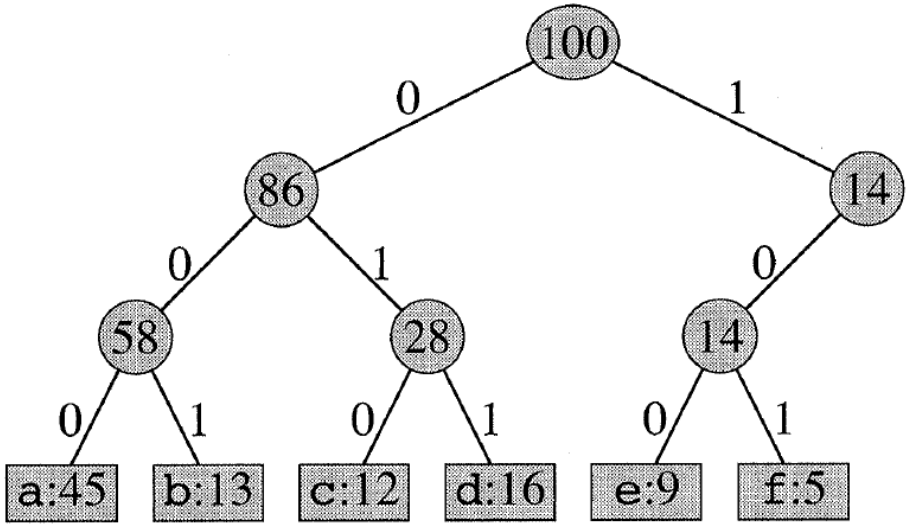
Komprimering

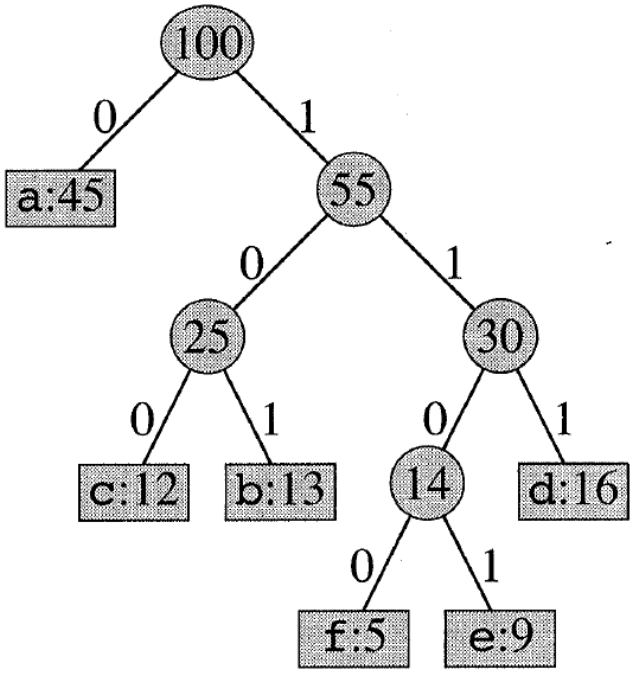
Givet frekvensen af symbolerne i input, erstat input symbolerne med **kortere bitstrenge** (fast-længde eller variabel-længde)

	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

NB: Variabel-længde skal være prefix-fri

Fixed-længde vs variabel-længde





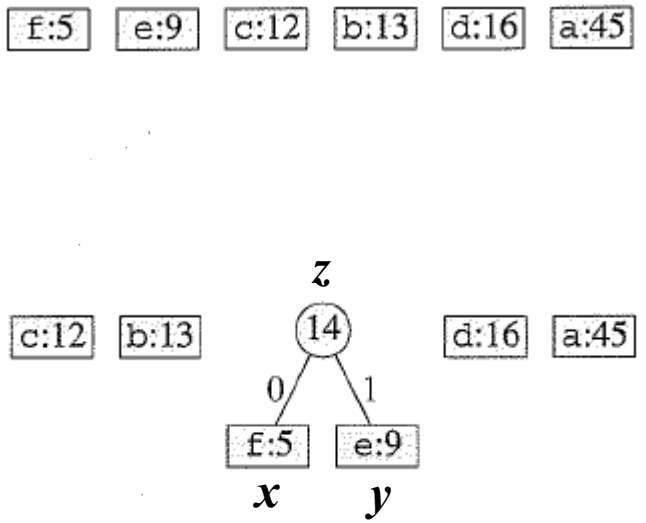
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

3 * (45+13+12+16+9+5)
= 300 bits

1*45+3*(13+12+16)+4*(9+5)
= 224 bits

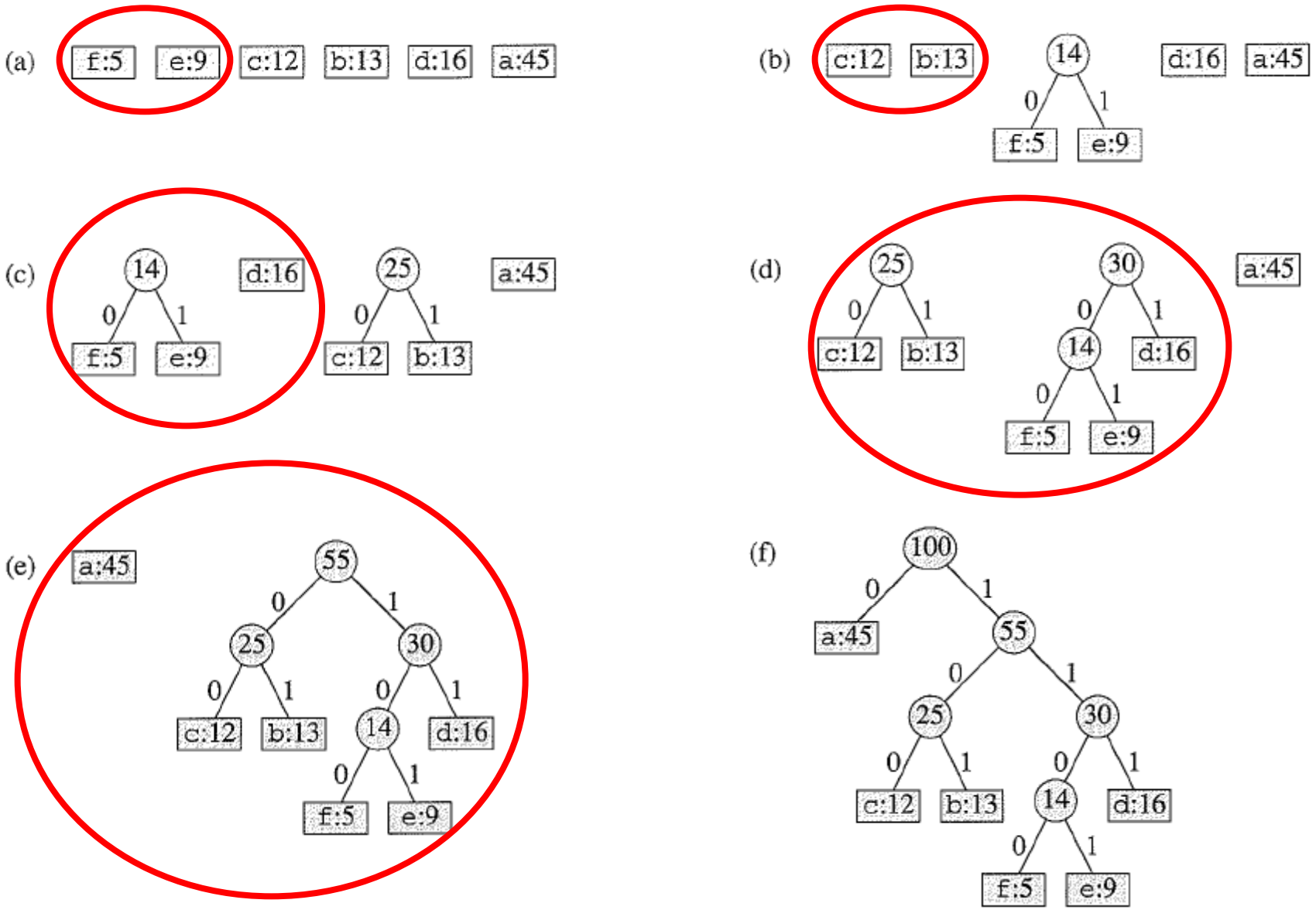
Huffman Koder

```
HUFFMAN(C)
1  n = |C|
2  Q = C
3  for i = 1 to n − 1
4       allocate a new node z
5       z.left = x = EXTRACT-MIN(Q)
6       z.right = y = EXTRACT-MIN(Q)
7       z.freq = x.freq + y.freq
8       INSERT(Q, z)
9  return EXTRACT-MIN(Q)      // return the root of the tree
```



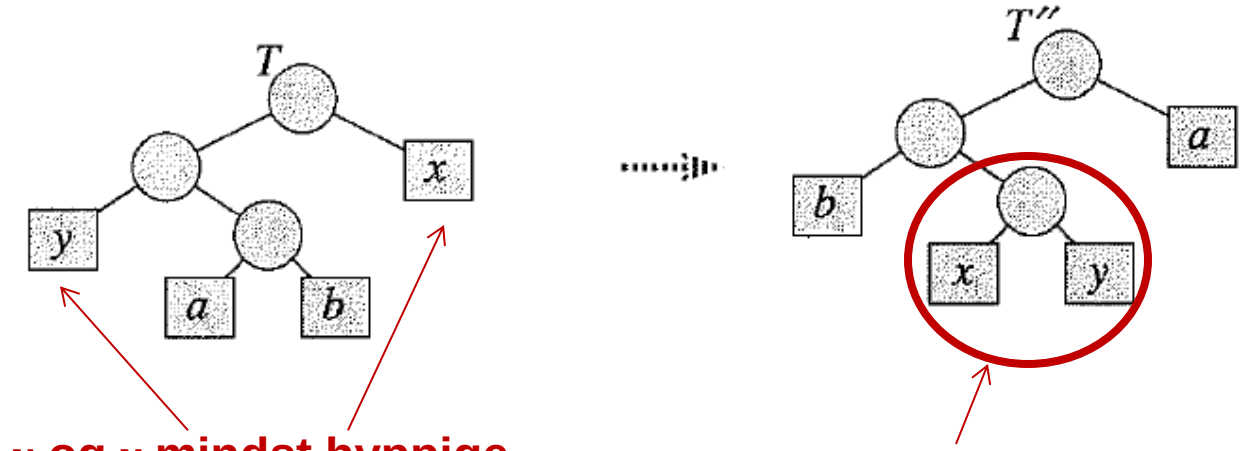
Tid $O(n \cdot \log n)$

Huffman Koder



Korrektheden af Huffman Koder

Sætning Der findes altid en optimal prefix kode hvor de to mindst hyppige symboler (x og y) har samme kodelængde og kun adskiller sig i sidste bit

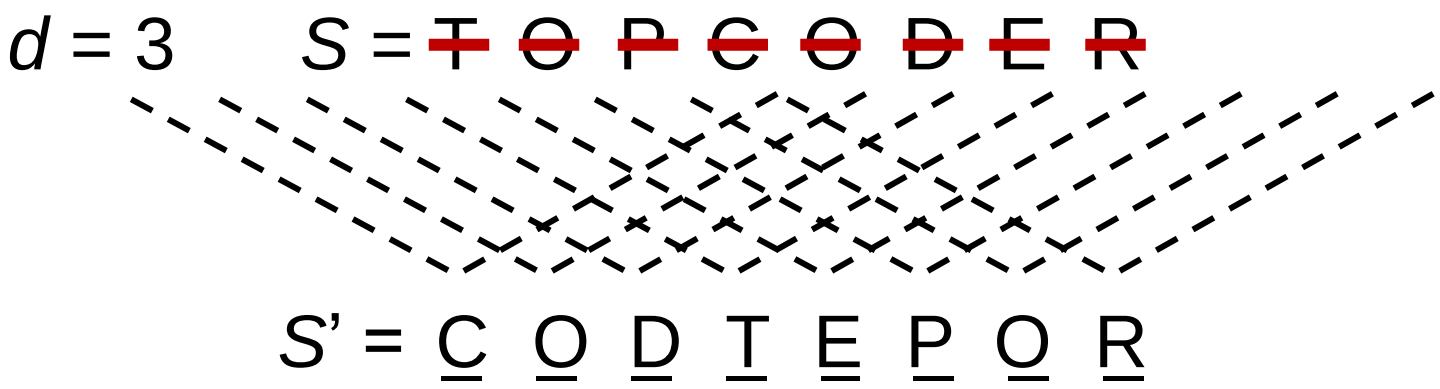


TopCoder Open 2014 – Round 1A (500 point)

Input: En streng *S*, afstand *d*

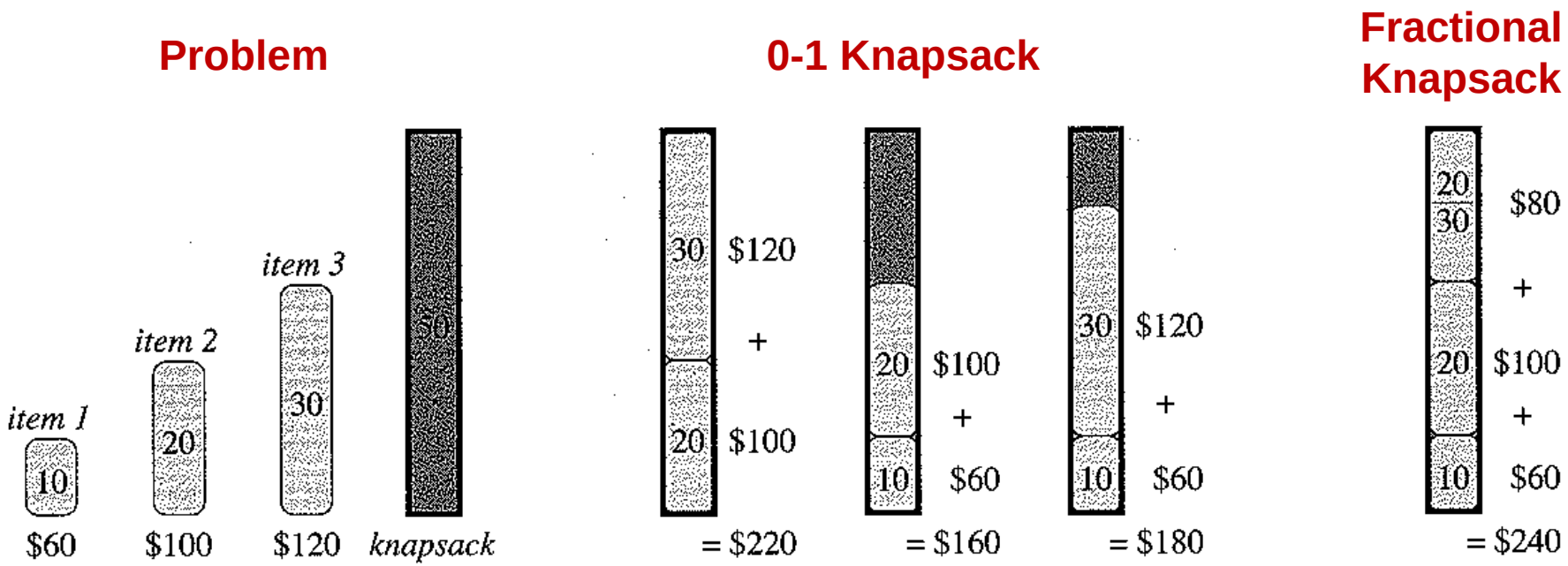
Ouput: En streng *S*' = leksikografisk mindste permutation af *S*, hvor ingen tegn flyttes mere end *d* positioner

Grådig løsning

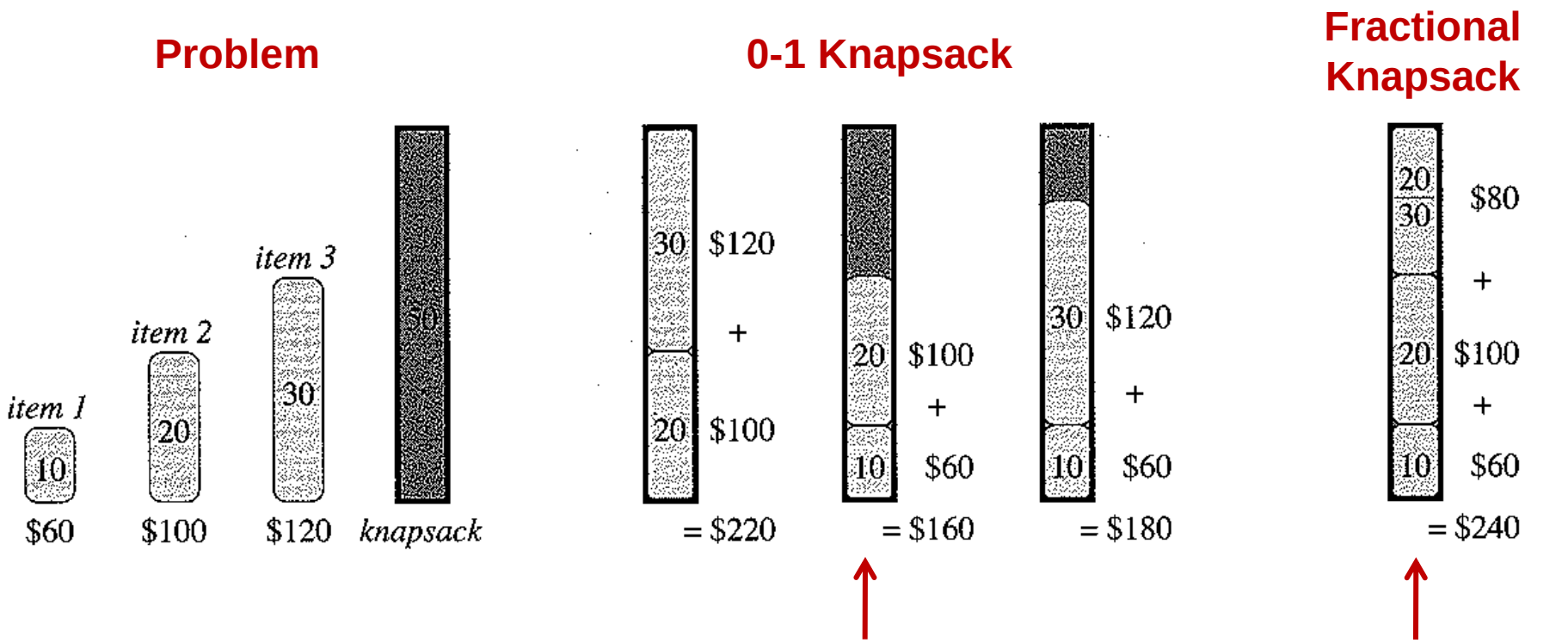


Algoritme Konstruer *S'* fra venstre mod højre
- tag det tegnet længst til venstre i vinduet, hvis ubrugt
- ellers tag leksikografisk mindste tegn (det længst til venstre hvis flere ens)

Dynamisk Programmering vs Grådig



Dynamisk Programmering vs Grådig



c(k,v) = bedste værdi blandt k første objekter ved volume v tilbage

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

Grådigt fyld i efter aftagende værdi/volume

0-1-Knapsack

Volume	2	3	4	5
Værdi	3	5	4	6

$c(k,v)$

k \ v	0	1	2	3	4	5	6	7	8	9	10	11
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

Systematisk tabeludfyldning

$c(k,v)$

k \ v	0	1	2	3	4	5	6	7	8	9	10	11
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

Rekursion + “memoization”

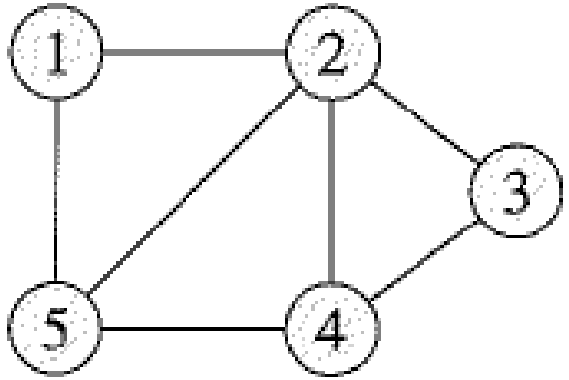
Generelle Algoritmiske Design Tenikker

- **Del-og-Kombiner**
  - Disjunkte delproblemer
- **Dynamisk Programmering**
  - Overlappende delproblemer
  - Husk allerede beregnede delresultater
  - Systematisk beregning af løsninger til alle mulige delproblemer (eller memoization)
- **Grådige Algoritmer**
  - Kun et delproblem

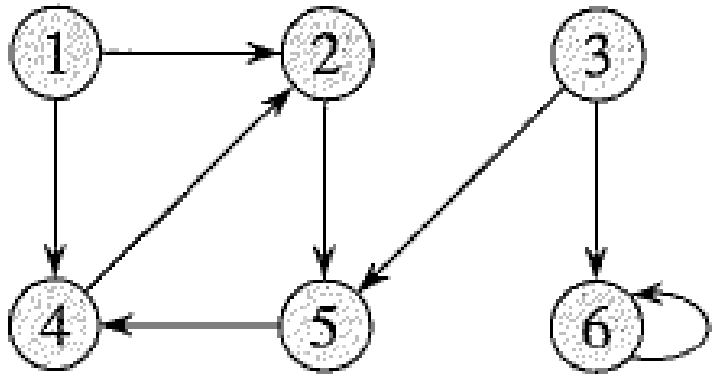
Algoritmer og Datastrukturer

Graf repræsentationer, bredde først søgning (BFS)
[CLRS, kapitel 20.1-20.2]

Grafer



Uorienterede grafer



Orienterede grafer

***G* = (*V*, *E*)** graf med knuder *V* og kanter *E*

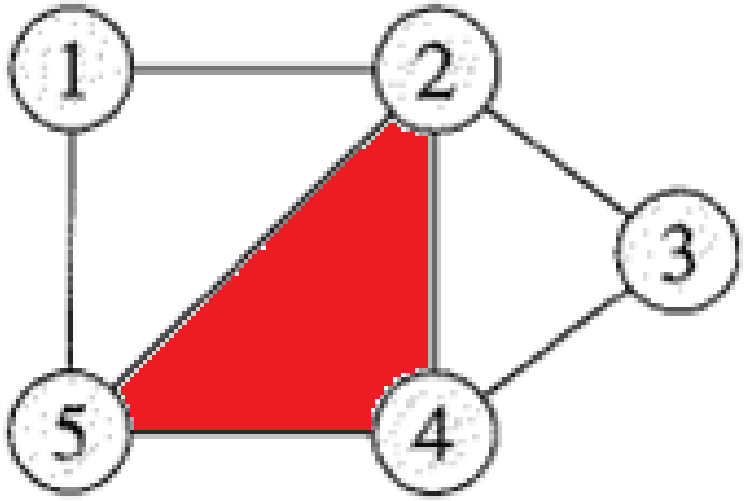
E : **{*u*, *v*}** kant mellem *u* og *v* i en uorienteret graf og

(*u*, *v*) en orienteret kant fra *u* til *v*

n = |*V*| = antal knuder

m = |E| = antal kanter (forbindelser mellem knuder)

Planar Grafer - Eulers formel

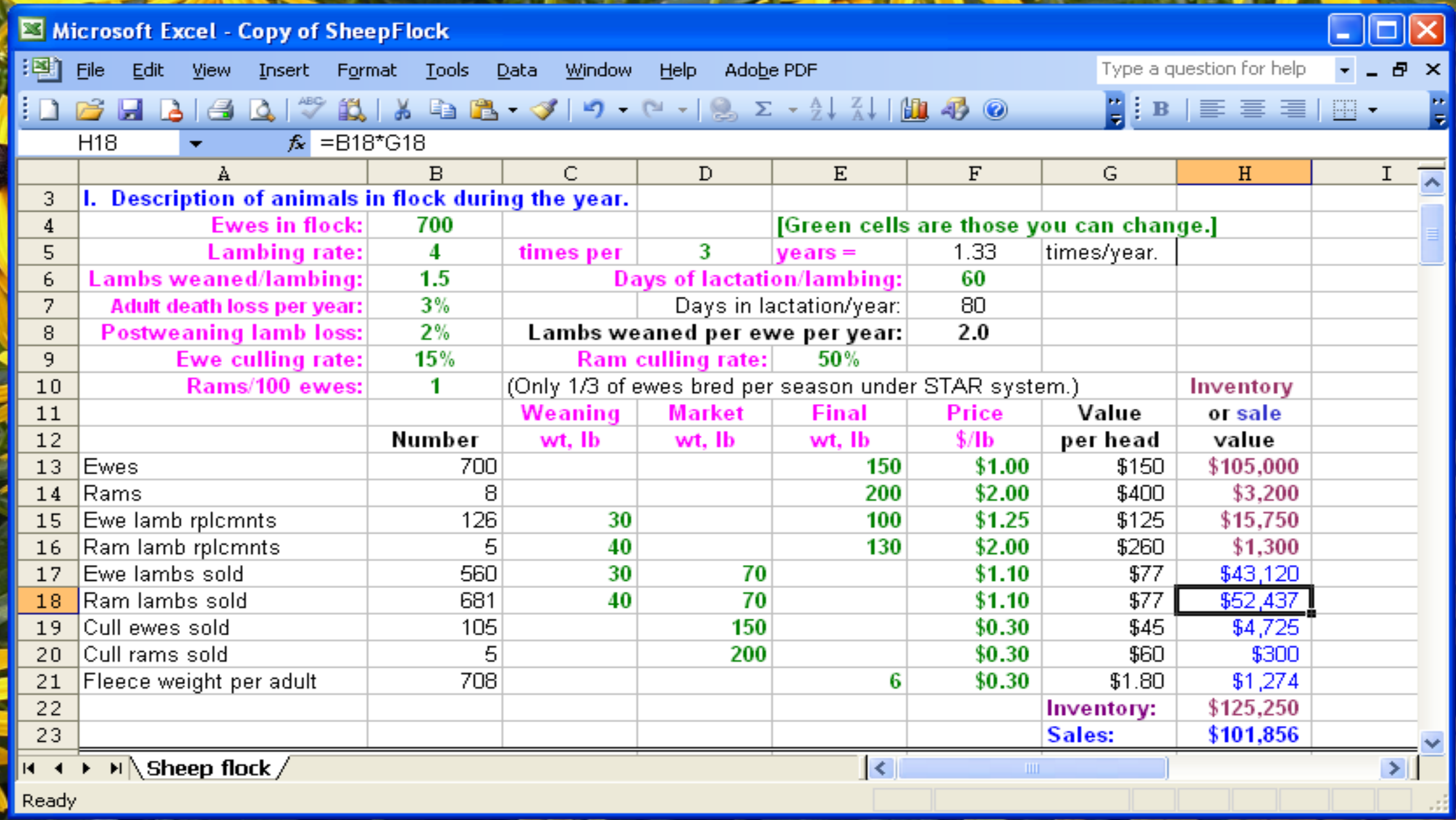


$|V| = 5$
$|E| = 7$
flader = 4

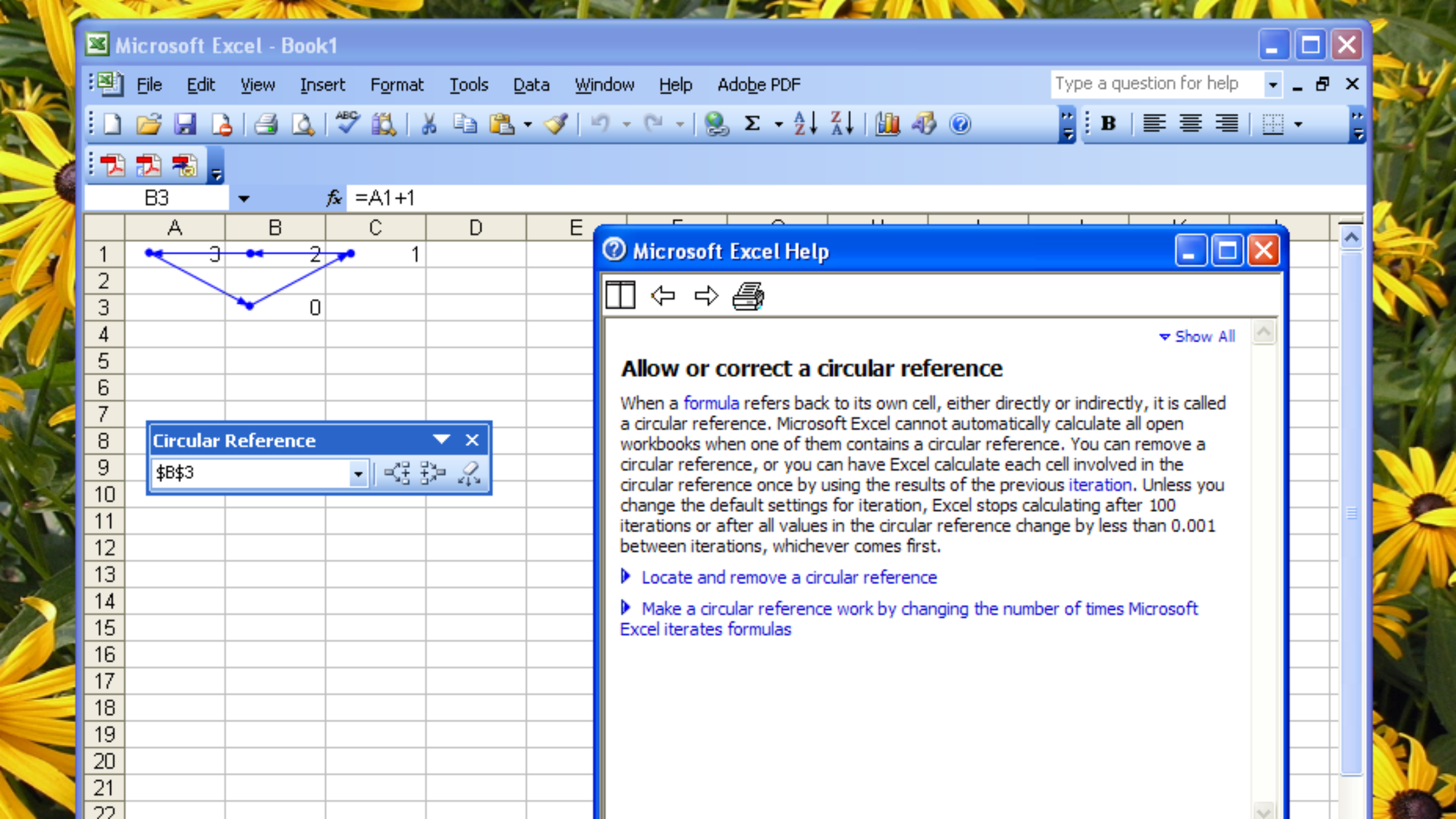
For en sammenhængende planar graf gælder:

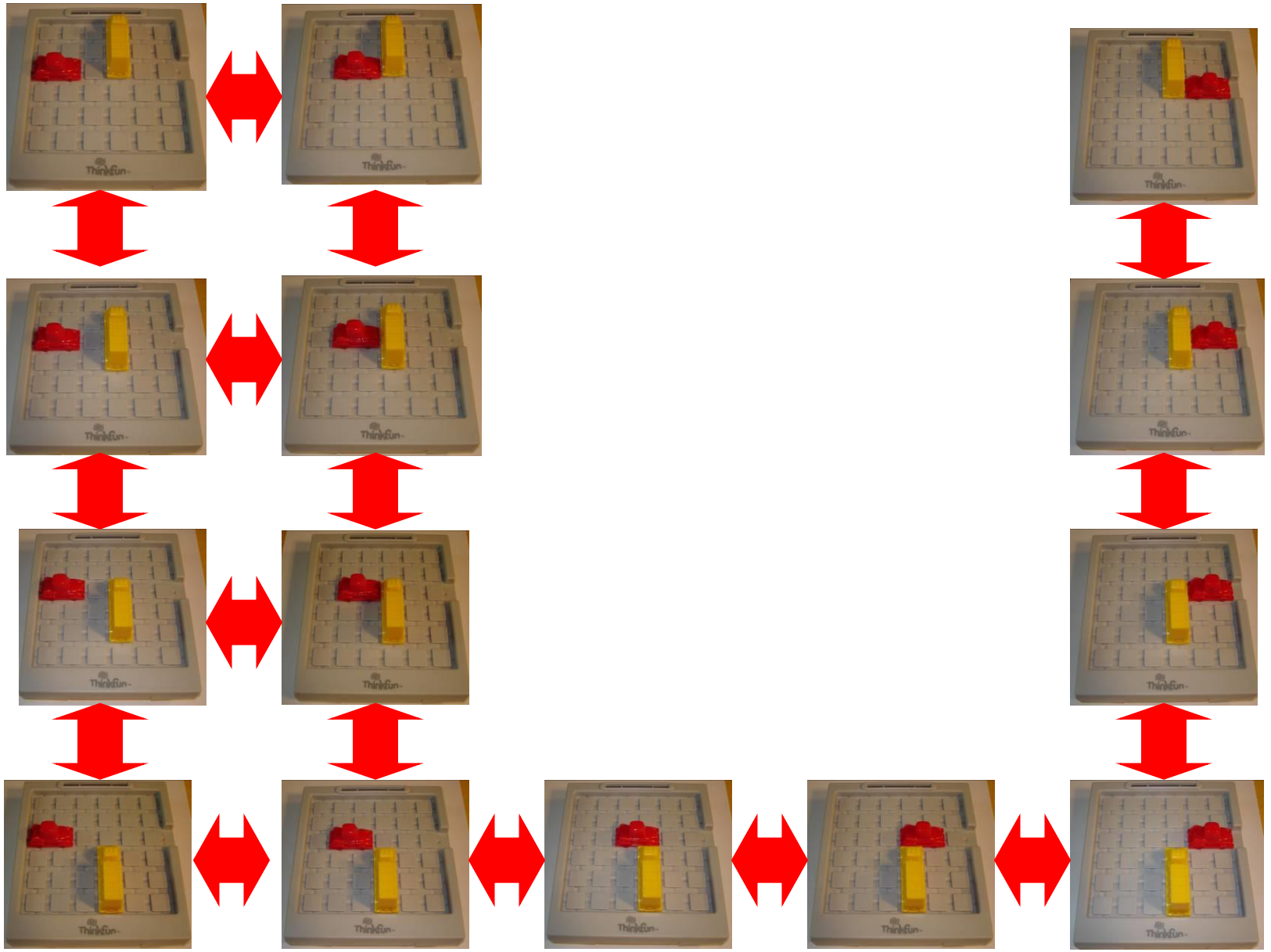
Eulers formel: $|V| - |E| + \#\text{ flader} = 2$

Korollar: $|E| \leq 3|V| - 6$ (for $|V| \geq 3$, ingen selvløkker, ingen parallelle kanter)



	A	B	C	D	E	F	G	H	I
3	**I. Description of animals in flock during the year.**								
4	**Ewes in flock:**	**700**			**[Green cells are those you can change.]**				
5	**Lambing rate:**	**4**	**times per**	**3**	**years =**	1.33	times/year.		
6	**Lambs weaned/lambing:**	**1.5**			**Days of lactation/lambing:**	**60**			
7	**Adult death loss per year:**	**3%**			Days in lactation/year:	80			
8	**Postweaning lamb loss:**	**2%**			**Lambs weaned per ewe per year:**	**2.0**			
9	**Ewe culling rate:**	**15%**		**Ram culling rate:**	**50%**				
10	**Rams/100 ewes:**	**1**	(Only 1/3 of ewes bred per season under STAR system.)					**Inventory**	
11			**Weaning**	**Market**	**Final**	**Price**	**Value**	**or sale**	
12		**Number**	**wt, lb**	**wt, lb**	**wt, lb**	**$/lb**	**per head**	**value**	
13	Ewes	700			**150**	**$1.00**	$150	**$105,000**	
14	Rams	8			**200**	**$2.00**	$400	**$3,200**	
15	Ewe lamb rplcmnts	126	**30**		**100**	**$1.25**	$125	**$15,750**	
16	Ram lamb rplcmnts	5	**40**		**130**	**$2.00**	$260	**$1,300**	
17	Ewe lambs sold	560	**30**	**70**		**$1.10**	$77	$43,120	
18	Ram lambs sold	681	**40**	**70**		**$1.10**	$77	$52,437	
19	Cull ewes sold	105		**150**		**$0.30**	$45	$4,725	
20	Cull rams sold	5		**200**		**$0.30**	$60	$300	
21	Fleece weight per adult	708			**6**	**$0.30**	$1.80	$1,274	
22							**Inventory:**	**$125,250**	
23							**Sales:**	**$101,856**	


Microsoft Excel - Book1
File Edit View Insert Format Tools Data Window Help Adobe PDF
Type a question for help
B3
=A1+1
A B C D E
1 3 2 1
2
3 0
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
Circular Reference
B3
Microsoft Excel Help
Show All
Allow or correct a circular reference
When a formula refers back to its own cell, either directly or indirectly, it is called a circular reference. Microsoft Excel cannot automatically calculate all open workbooks when one of them contains a circular reference. You can remove a circular reference, or you can have Excel calculate each cell involved in the circular reference once by using the results of the previous iteration. Unless you change the default settings for iteration, Excel stops calculating after 100 iterations or after all values in the circular reference change by less than 0.001 between iterations, whichever comes first.
Locate and remove a circular reference
Make a circular reference work by changing the number of times Microsoft Excel iterates formulas

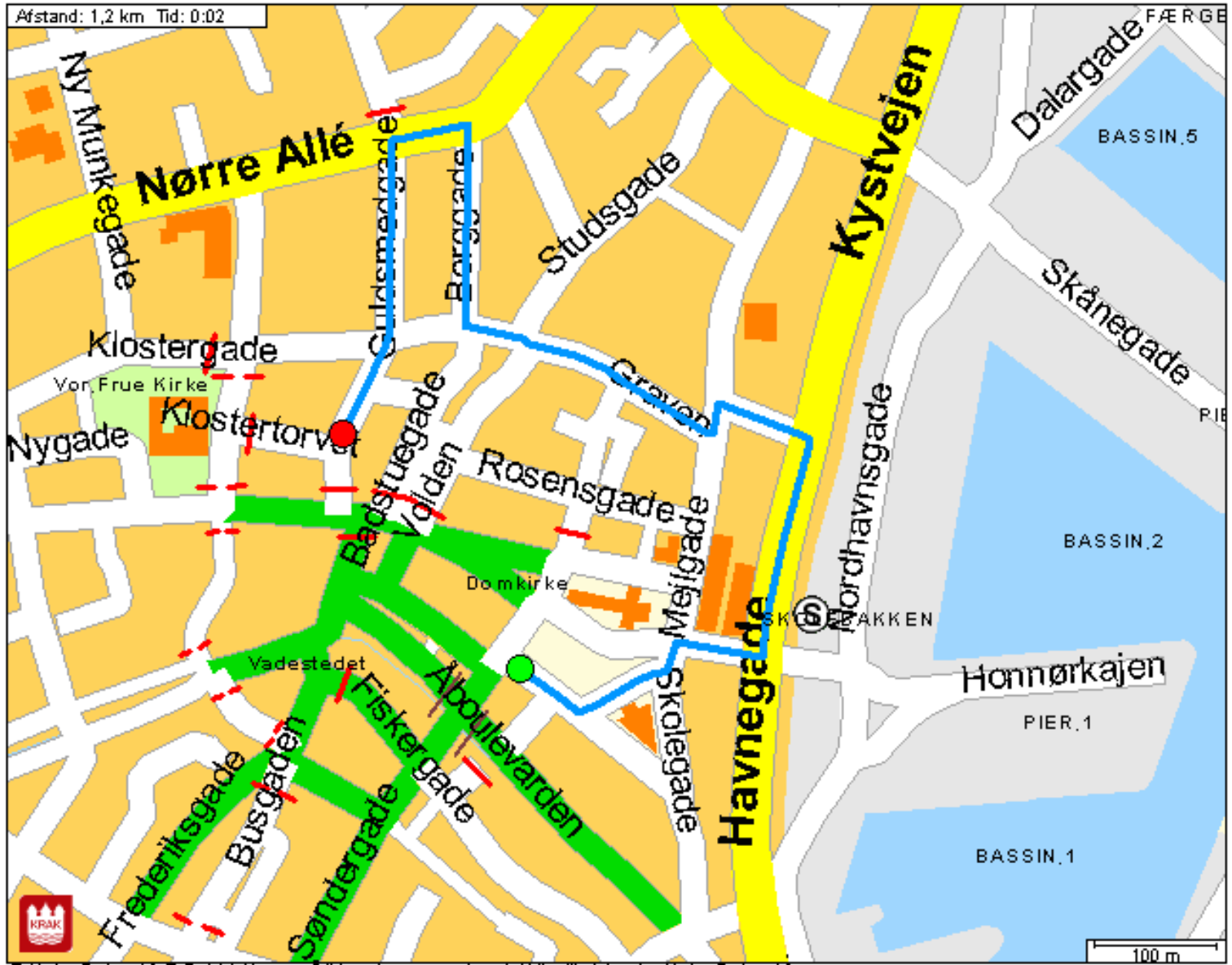


Rute på kort

Fra Kannikegade 1 , 8000 Århus C

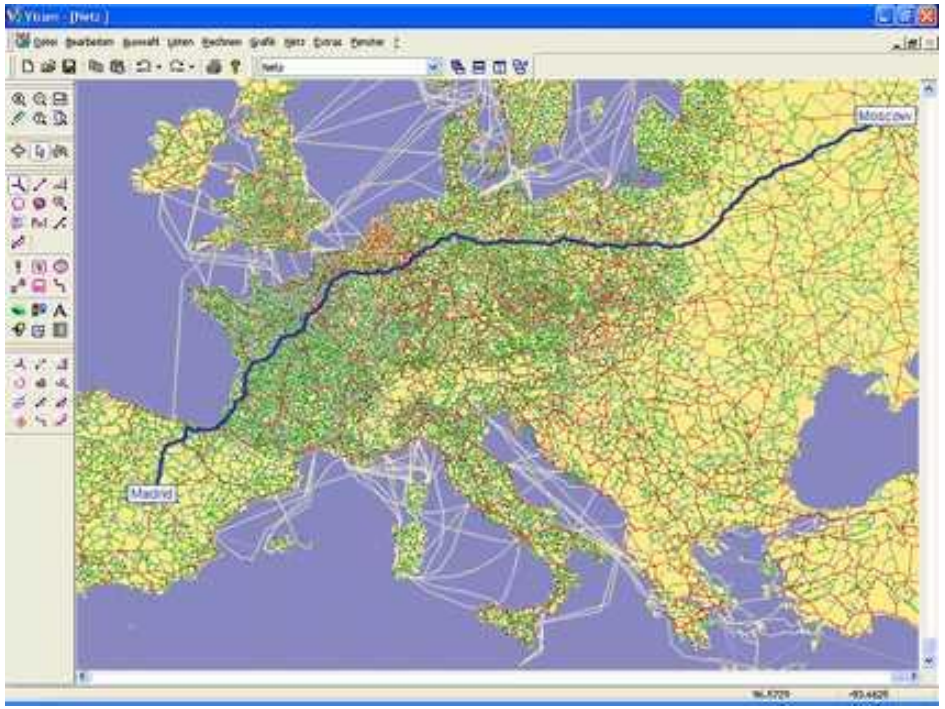
Til Guldsmedgade 1 , 8000 Århus C

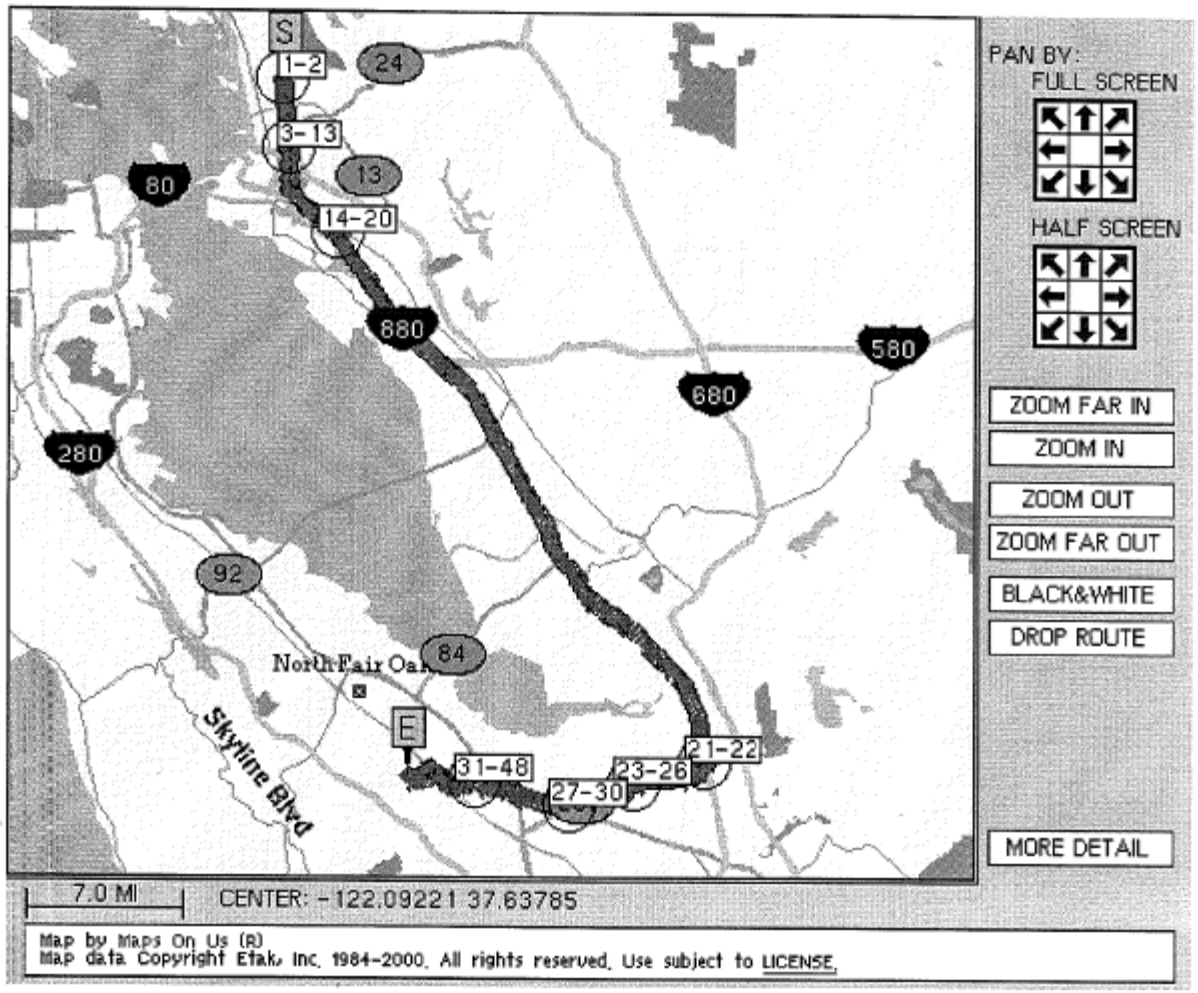
Via



Kort over Vest-Europa

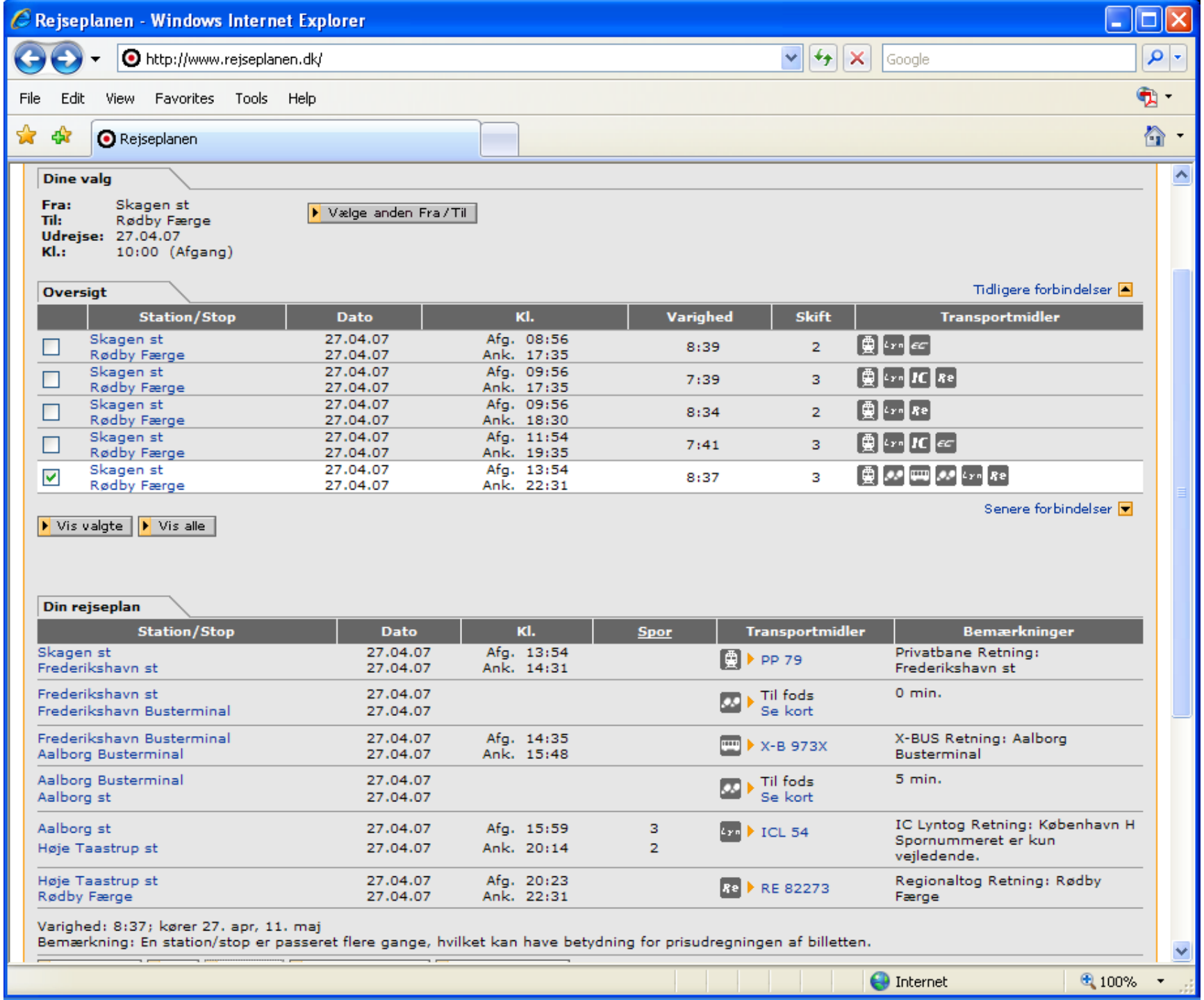
- **18.029.721 knuder**
- **42.199.587 orienterede kanter**



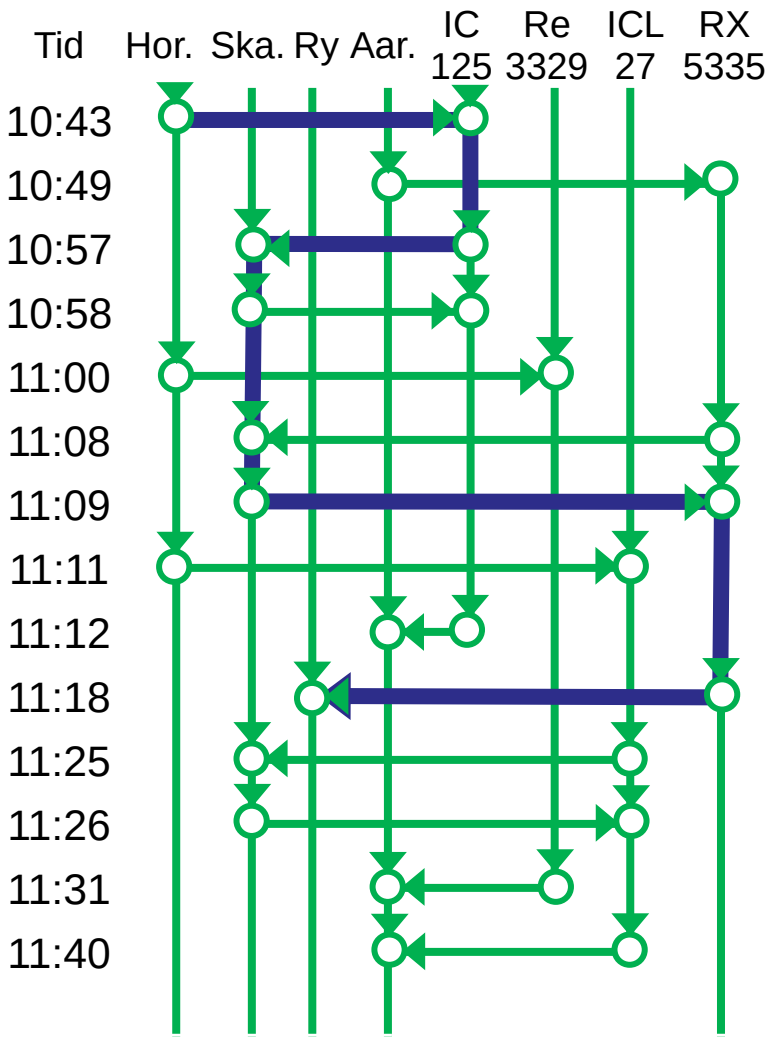


"However, because of the size of the routing data, we have to use heuristics when planning routes. As a result, sometimes a Favor Highways route will be slightly faster than the Fastest route."

— MapsOnUs


Rejseplanen - Windows Internet Explorer
http://www.rejseplanen.dk/
Google
File Edit View Favorites Tools Help
Rejseplanen
Dine valg
Fra: Skagen st
Til: Rødby Færge
Udrejse: 27.04.07
Kl.: 10:00 (Afgang)
Vælge anden Fra/Til
Oversigt
Tidligere forbindelser
Station/Stop | Dato | Kl. | Varighed | Skift | Transportmidler
Skagen st / Rødby Færge | 27.04.07 / 27.04.07 | Afg. 08:56 / Ank. 17:35 | 8:39 | 2
Skagen st / Rødby Færge | 27.04.07 / 27.04.07 | Afg. 09:56 / Ank. 17:35 | 7:39 | 3
Skagen st / Rødby Færge | 27.04.07 / 27.04.07 | Afg. 09:56 / Ank. 18:30 | 8:34 | 2
Skagen st / Rødby Færge | 27.04.07 / 27.04.07 | Afg. 11:54 / Ank. 19:35 | 7:41 | 3
Skagen st / Rødby Færge | 27.04.07 / 27.04.07 | Afg. 13:54 / Ank. 22:31 | 8:37 | 3
Senere forbindelser
Vis valgte
Vis alle
Din rejseplan
Station/Stop | Dato | Kl. | Spor | Transportmidler | Bemærkninger
Skagen st / Frederikshavn st | 27.04.07 / 27.04.07 | Afg. 13:54 / Ank. 14:31 | PP 79 | Privatbane Retning: Frederikshavn st
Frederikshavn st / Frederikshavn Busterminal | 27.04.07 / 27.04.07 | Til fods Se kort | 0 min.
Frederikshavn Busterminal / Aalborg Busterminal | 27.04.07 / 27.04.07 | Afg. 14:35 / Ank. 15:48 | X-B 973X | X-BUS Retning: Aalborg Busterminal
Aalborg Busterminal / Aalborg st | 27.04.07 / 27.04.07 | Til fods Se kort | 5 min.
Aalborg st / Høje Taastrup st | 27.04.07 / 27.04.07 | Afg. 15:59 / Ank. 20:14 | 3 / 2 | ICL 54 | IC Lyntog Retning: København H Spornummeret er kun vejledende.
Høje Taastrup st / Rødby Færge | 27.04.07 / 27.04.07 | Afg. 20:23 / Ank. 22:31 | RE 82273 | Regionaltog Retning: Rødby Færge
Varighed: 8:37; kører 27. apr, 11. maj
Bemærkning: En station/stop er passeret flere gange, hvilket kan have betydning for prisudregningen af billetten.
Internet
100%

Rejseplan (Horsens til Ry)

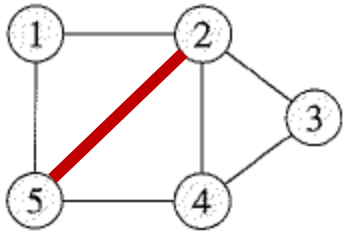


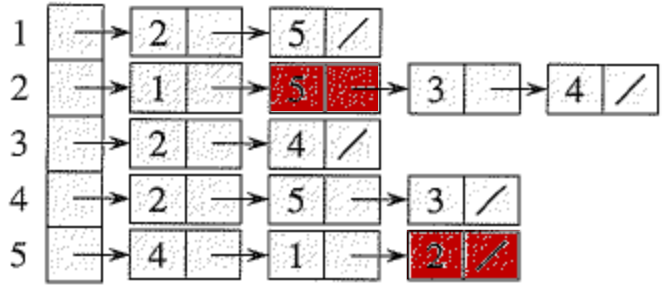
Algoritme
Find tidligste knude for Ry der kan nås fra en given start-knude i Horsens

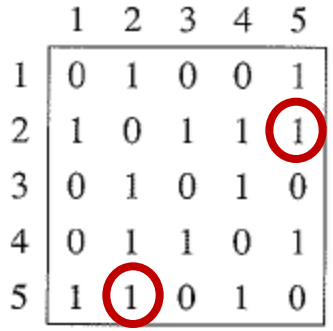
Tog	Ank	Afg	Station
		10:43	Horsens
IC125	10:57	10:58	Skanderborg St
	11:12		Aarhus H
		11:00	Horsens
Re3329	11:31		Aarhus H
		11:11	Horsens
ICL27	11:25	11:26	Skanderborg St
	11:40		Aarhus H
		10:49	Aarhus H
RX5335	11:08	11:09	Skanderborg St
	11:18		Ry St

uddrag af køreplaner

Graf repræsentationer: Incidenslister og incidensmatricer





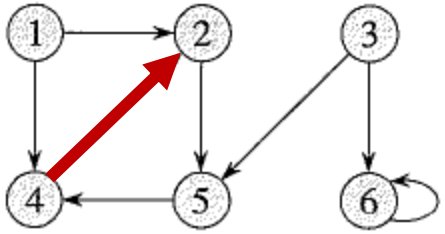


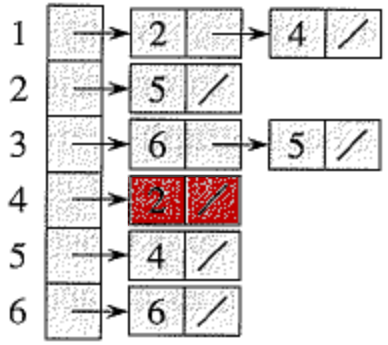
	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

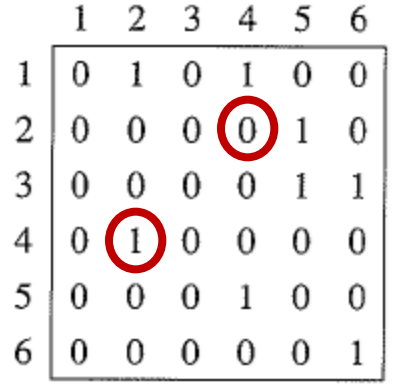
Uorienterede grafer

Plads O(n+m)

Plads O(n^2)



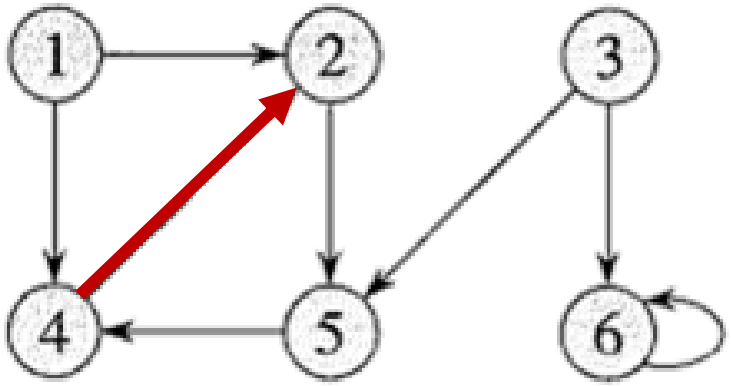




	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Orienterede grafer

Graf repræsentationer:
… et par flere alternativer



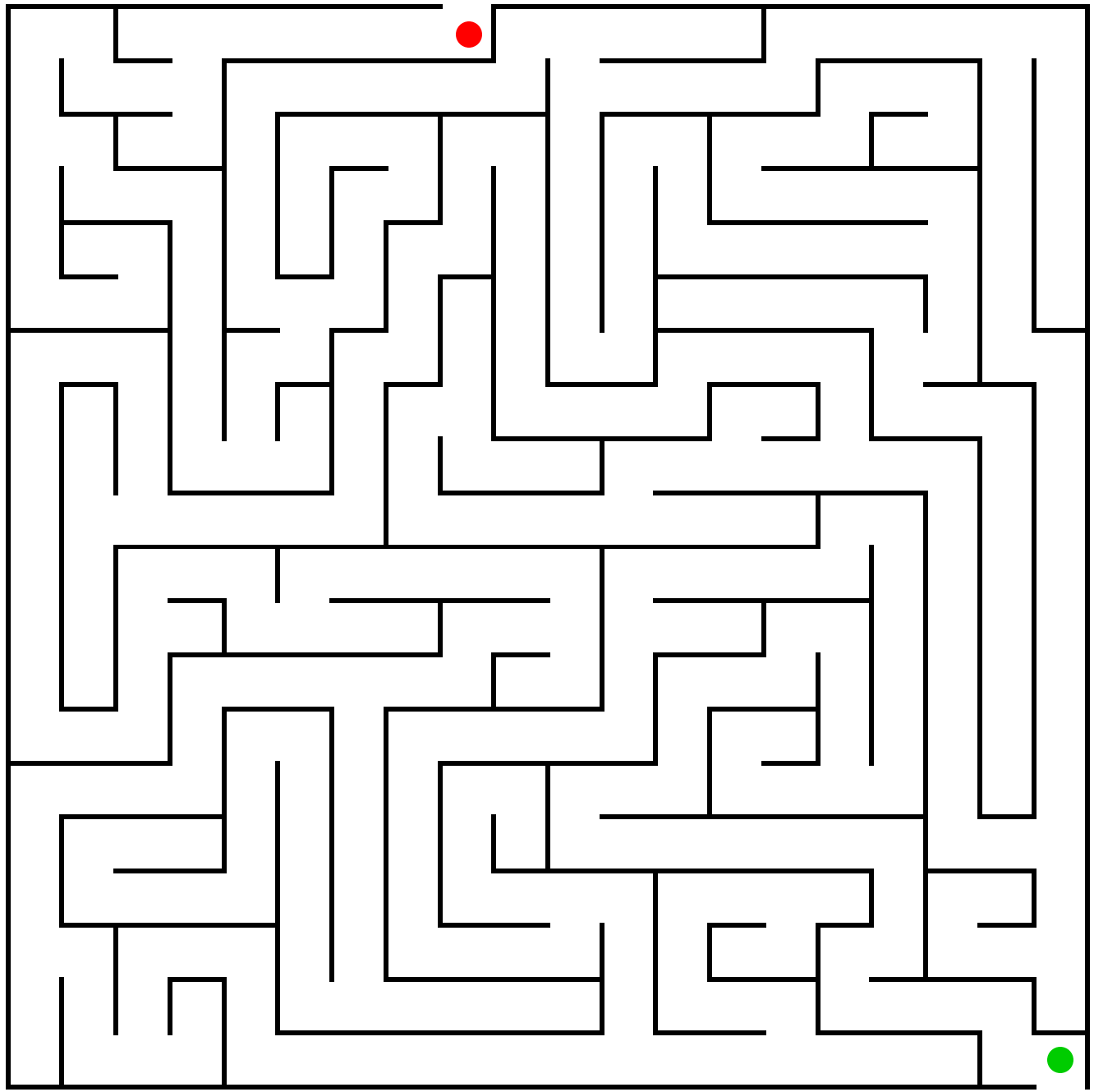
Kantliste
(list/array med par af heltal)

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

Kompakte incidenslister
(to arrays med heltal)

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

E	2	4	5	5	6	2	4	6



Bredde først søgning (BFS)

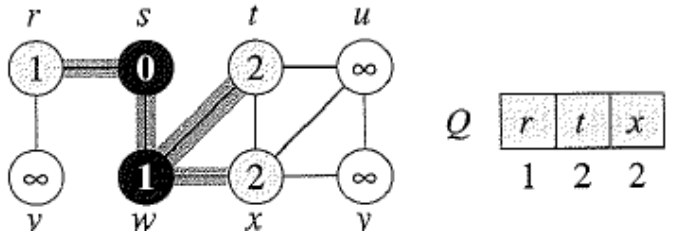
```
BFS(G, s)
 1  for each vertex u ∈ G.V − {s}
 2      u.color = WHITE
 3      u.d = ∞
 4      u.π = NIL
 5  s.color = GRAY
 6  s.d = 0
 7  s.π = NIL
 8  Q = ∅
 9  ENQUEUE(Q, s)
10  while Q ≠ ∅
11      u = DEQUEUE(Q)
12      for each v ∈ G.Adj[u]
13          if v.color == WHITE
14              v.color = GRAY
15              v.d = u.d + 1
16              v.π = u
17              ENQUEUE(Q, v)
18      u.color = BLACK
```

u.color:
WHITE = knuderne endnu ikke besøgt
GRAY = knuderne i køen Q
BLACK = knuderne besøgt

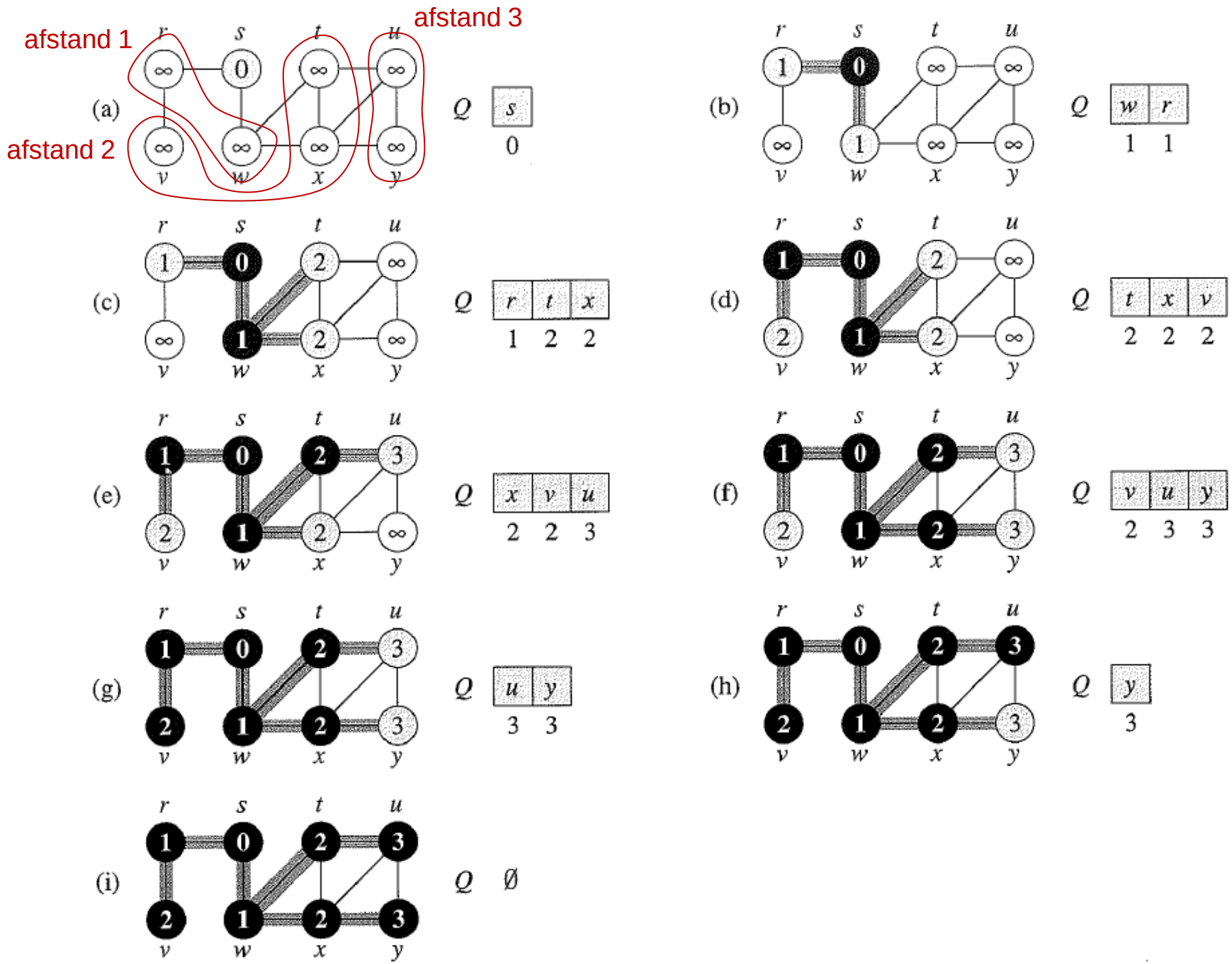
$u.d$ = afstand til s

$u.\pi$ = faderen til u i BFS træet

Q = kø af grå knuder
(som er forbundet til sorte knuder)



Tid $O(n+m)$

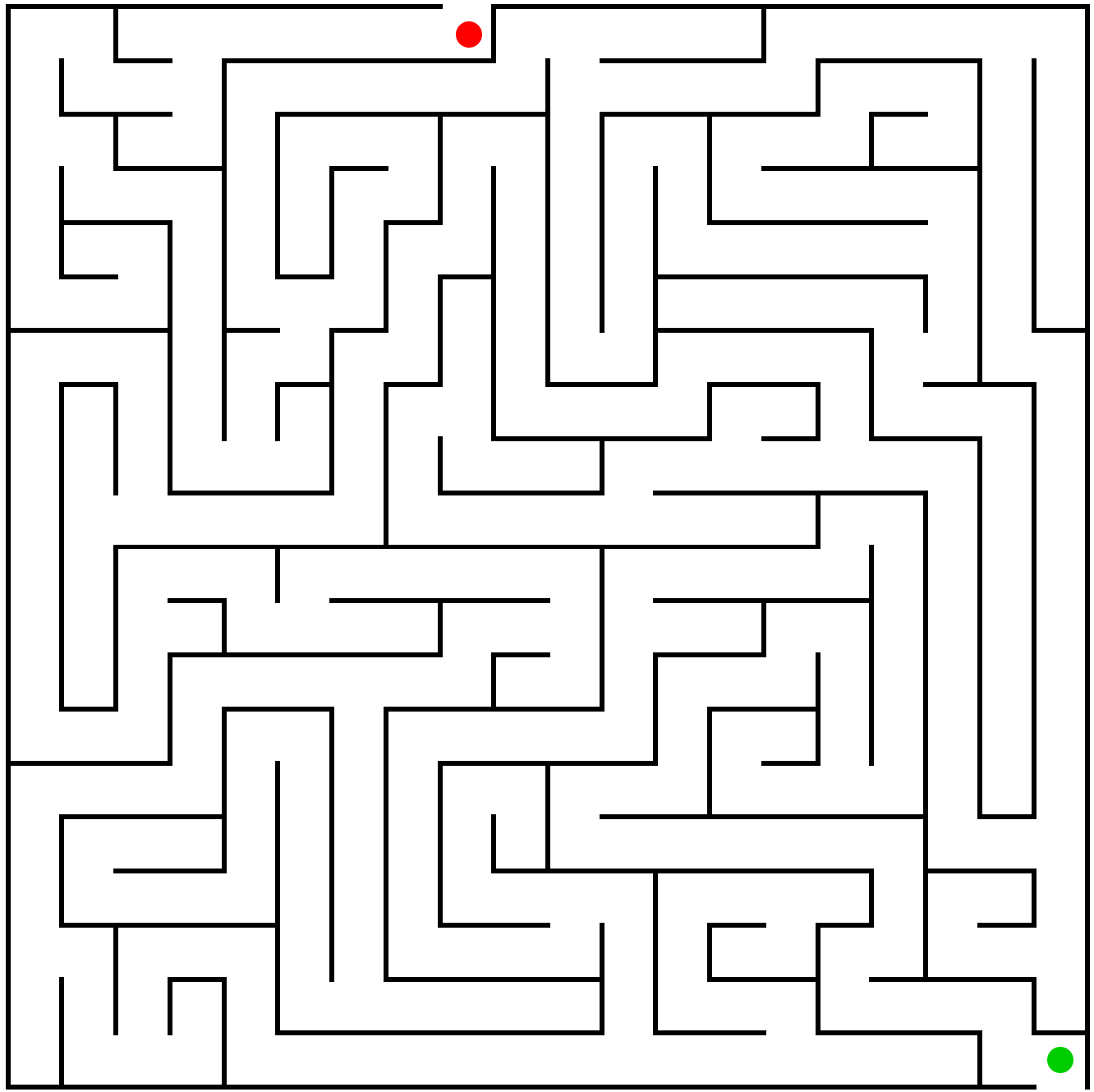

afstand 1
afstand 2
afstand 3
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
Q
s
0
w r
1 1
r t x
1 2 2
t x v
2 2 2
x v u
2 2 3
v u y
2 3 3
u y
3 3
y
3
Ø

BFS : Udskrivning af sti fra s til v

```
PRINT-PATH(G, s, v)
1   if v == s
2       print s
3   elseif v.π == NIL
4       print "no path from" s "to" v "exists"
5   else PRINT-PATH(G, s, v.π)
6       print v
```

Algoritmer og Datastrukturer

Dybde først søgning (DFS), Topologisk Sortering,
Stærke Sammenhængskomponenter
[CLRS, kapitel 20.3-20.5]



Dybde Først Søgning (DFS)

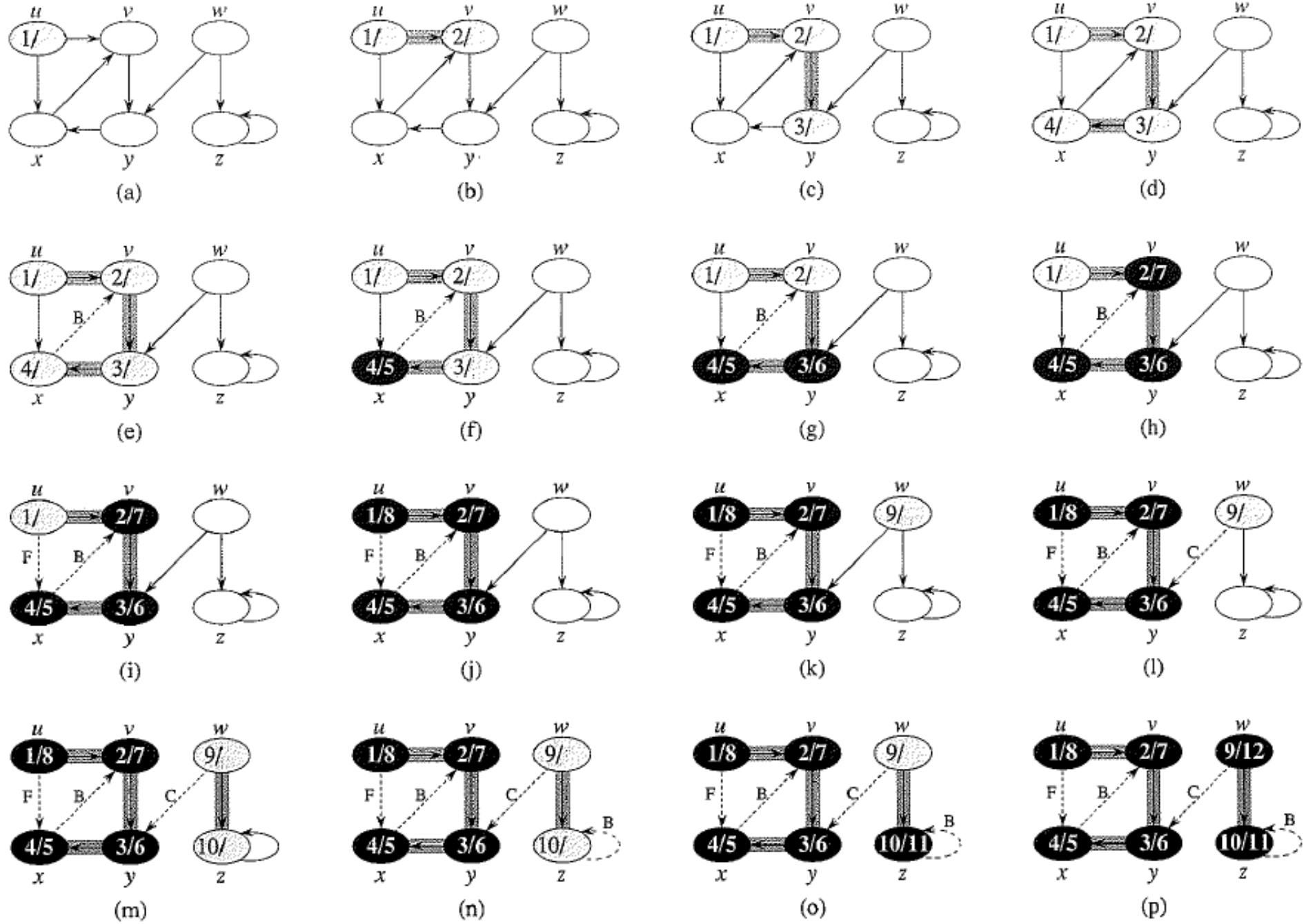
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)

DFS-VISIT(G, u)
 1  time = time + 1                  // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]            // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK                  // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

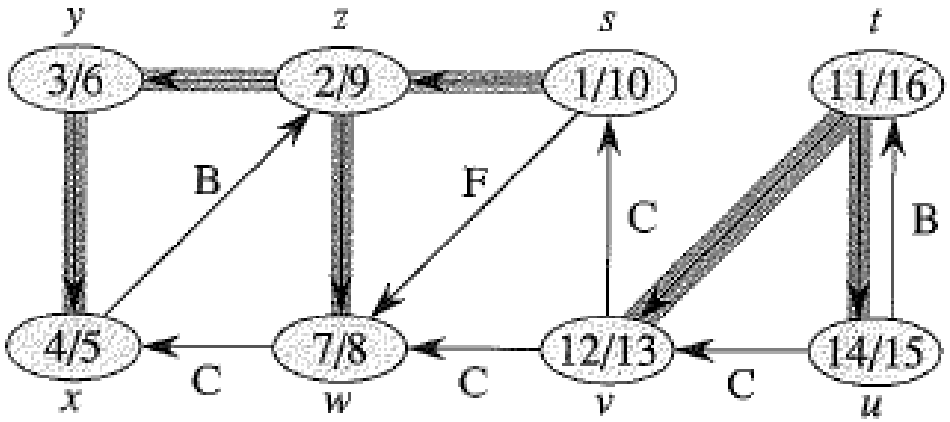
u.color
WHITE = knuderne endnu ikke besøgt
GRAY = knuder på rekursionsstakken
BLACK = knuderne besøgt

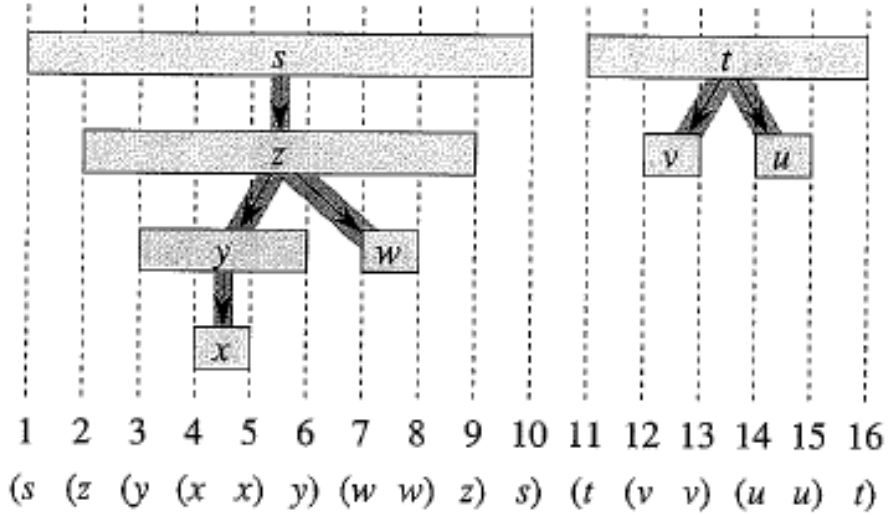
$u.\pi$ = faderen til u i DFS træet

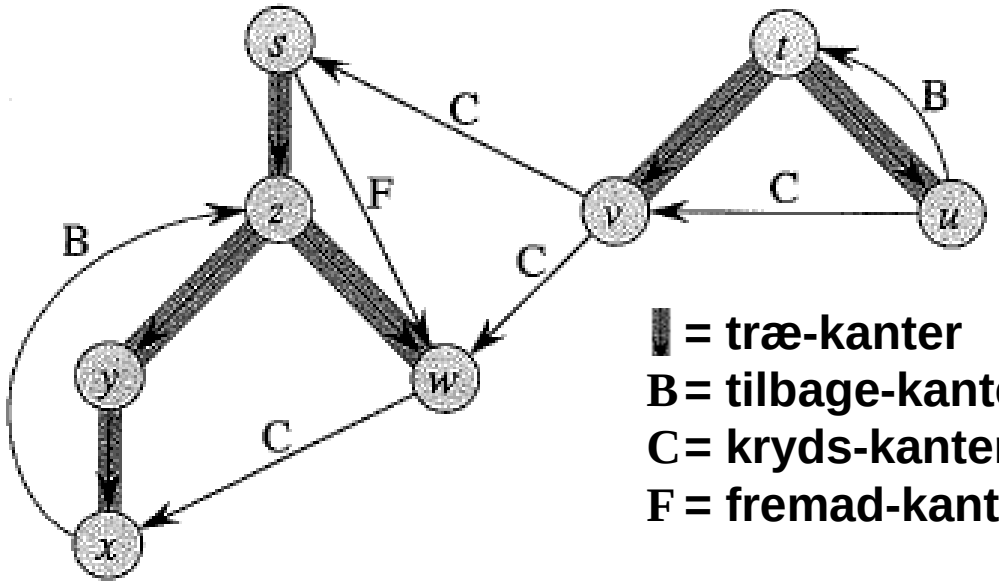
$u.d$ = "discover time" for u
$u.f$ = "finishing time" for u

Tid $O(n+m)$


u v w
1/
x y z
(a)
u v w
1/ 2/
x y z
(b)
u v w
1/ 2/ 3/
x y z
(c)
u v w
1/ 2/ 4/ 3/
x y z
(d)
u v w
1/ 2/ B 4/ 3/
x y z
(e)
u v w
1/ 2/ B 4/5 3/
x y z
(f)
u v w
1/ 2/ B 4/5 3/6
x y z
(g)
u v w
1/ 2/7 B 4/5 3/6
x y z
(h)
u v w
1/ 2/7 F B 4/5 3/6
x y z
(i)
u v w
1/8 2/7 F B 4/5 3/6
x y z
(j)
u v w
1/8 2/7 9/ F B 4/5 3/6
x y z
(k)
u v w
1/8 2/7 9/ F B C 4/5 3/6
x y z
(l)
u v w
1/8 2/7 9/ F B C 4/5 3/6 10/
x y z
(m)
u v w
1/8 2/7 9/ F B C 4/5 3/6 10/ B
x y z
(n)
u v w
1/8 2/7 9/ F B C 4/5 3/6 10/11 B
x y z
(o)
u v w
1/8 2/7 9/12 F B C 4/5 3/6 10/11 B
x y z
(p)







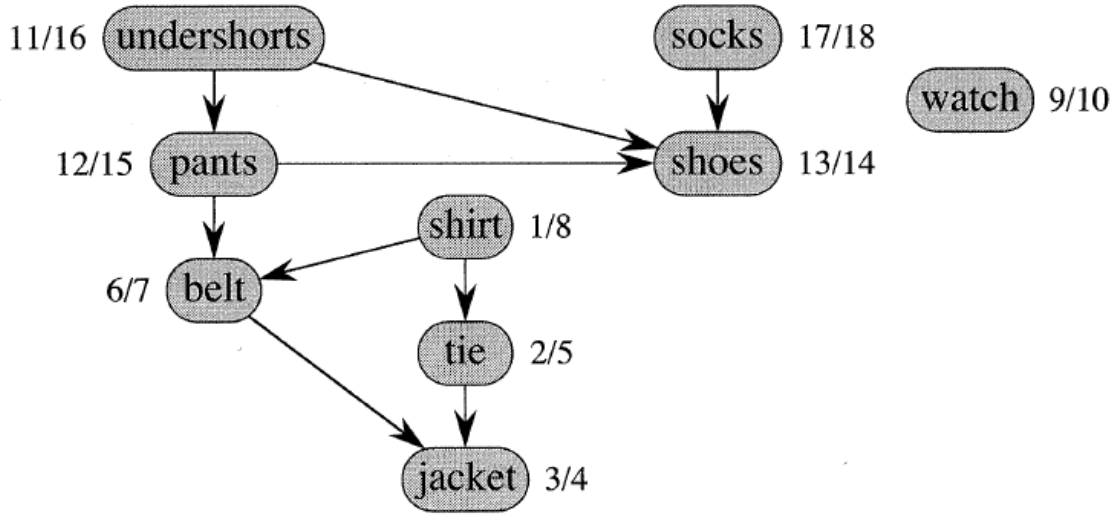
= træ-kanter
B = tilbage-kanter
C = kryds-kanter
F = fremad-kanter

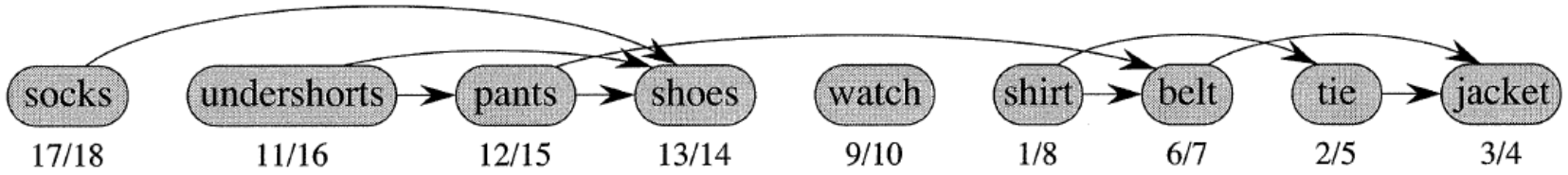
BFS og DFS anvendelser

BFS Finde afstande til startknuden
(afstand = antal kanter, f.eks. RushHour)

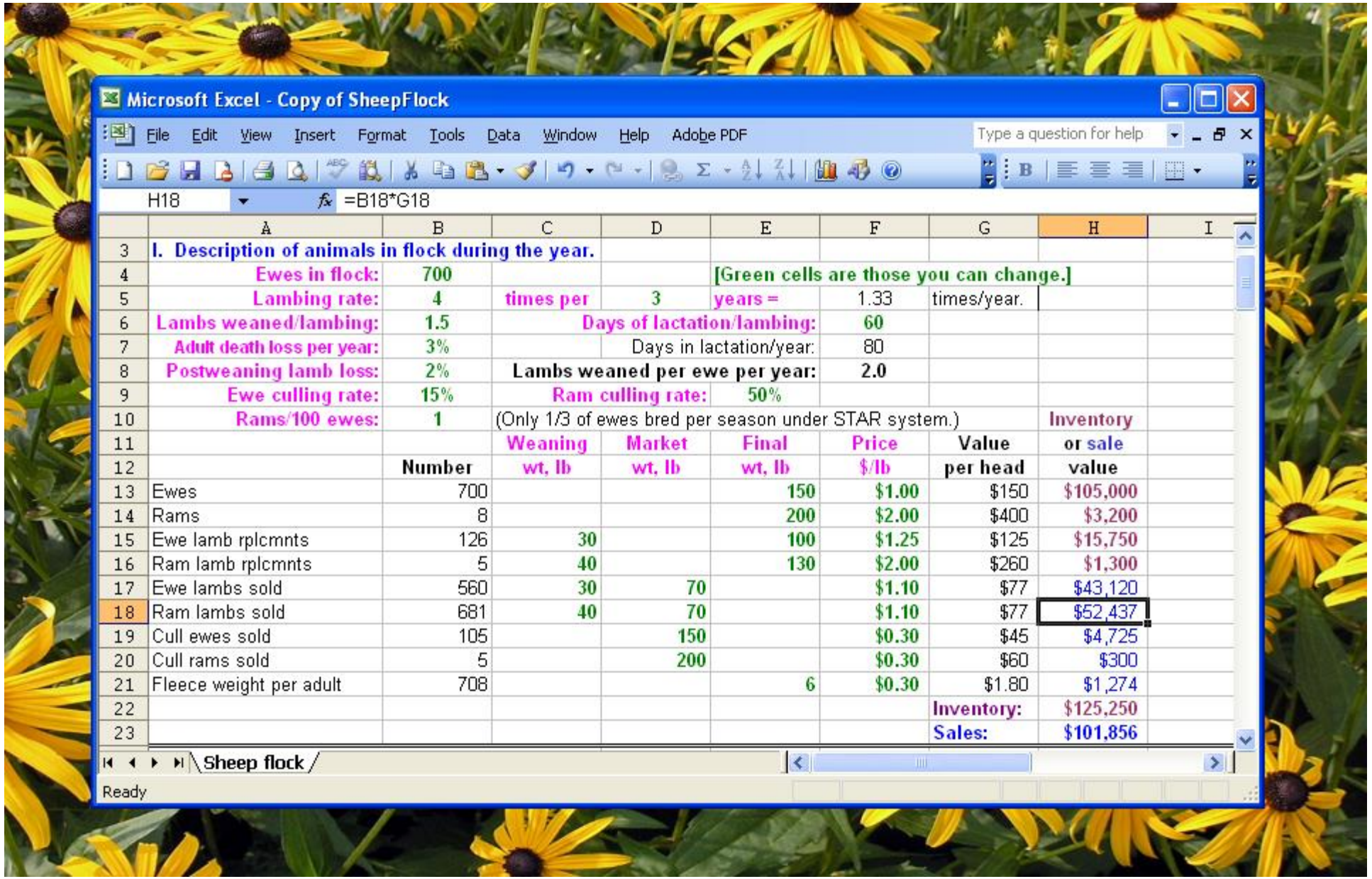
DFS *Topologisk sortering*
Stærke sammenhængskomponenter
(Planaritets testning)

Acykliske Grafer: Topologisk Sortering





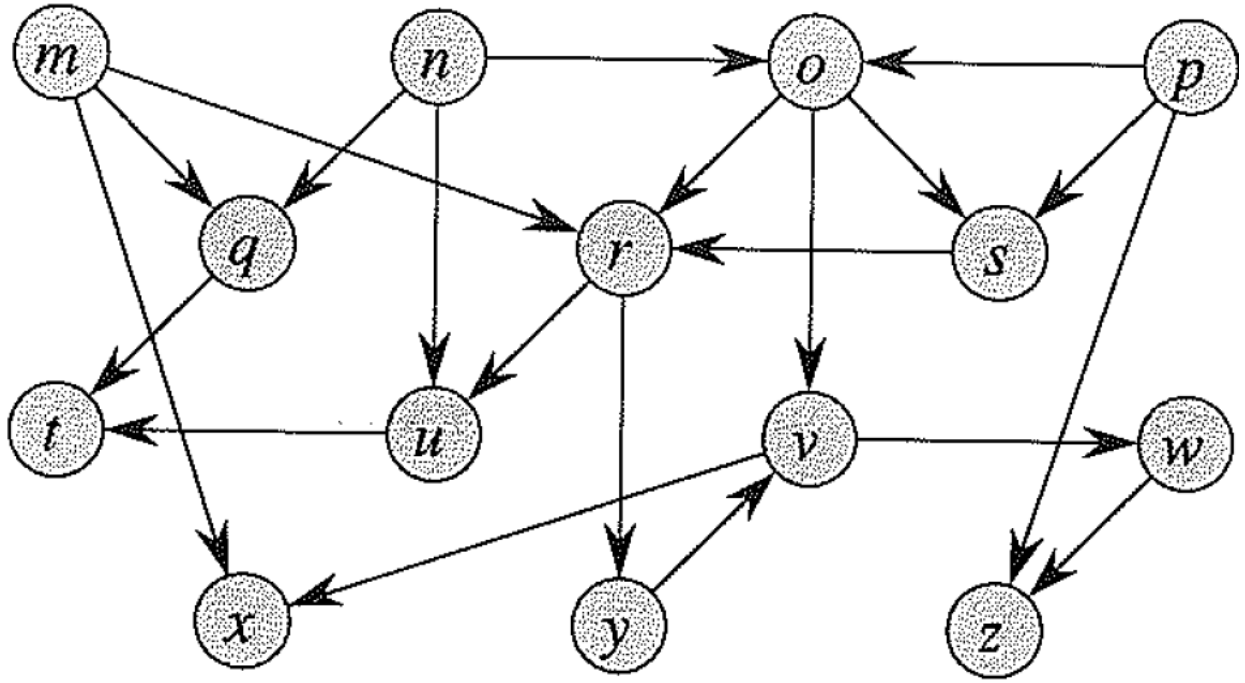
Alle kanter går fra venstre-mod-højre



Topologisk sortering = en rækkefølge hvor vi kan beregne cellernes indhold

Kahn 1962

Topologisk Sortering (I)

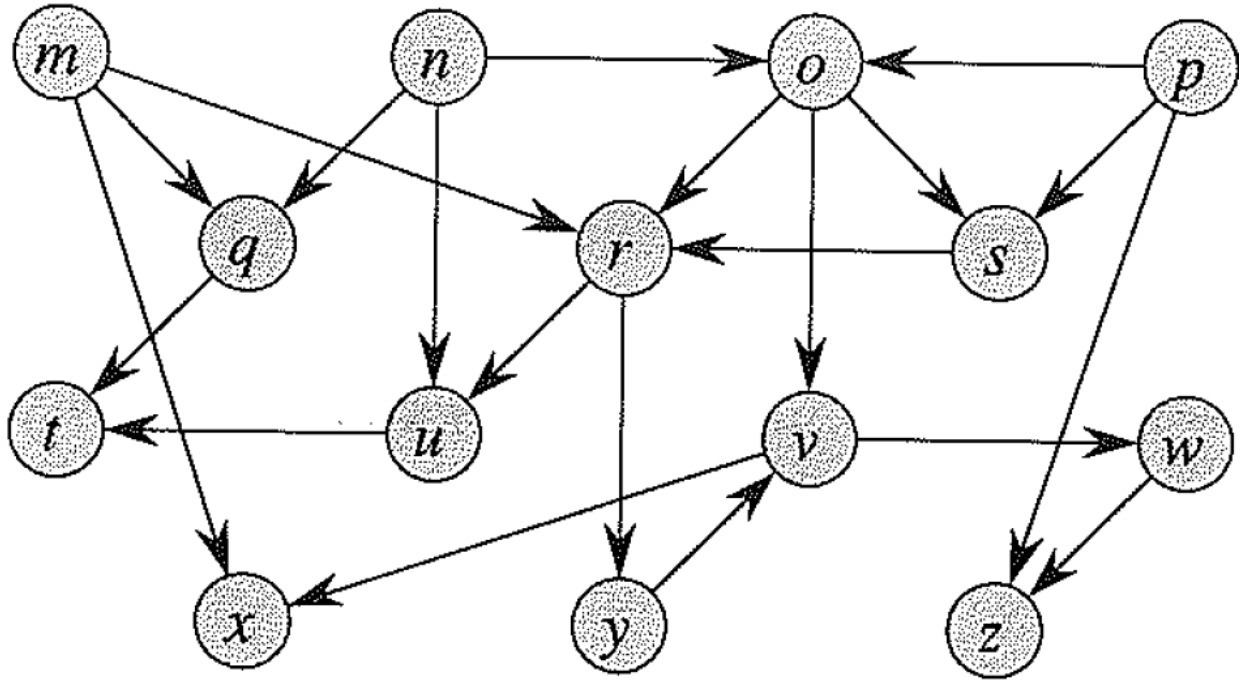


Algoritme: Grådigt slet en knude med indgrad 0 (og udgående kanter), og tilføj knuden sidst i den topologiske orden

Tid $O(m+n)$

Tarjan 1976

Topologisk Sortering (II)



TOPOLOGICAL-SORT(G)

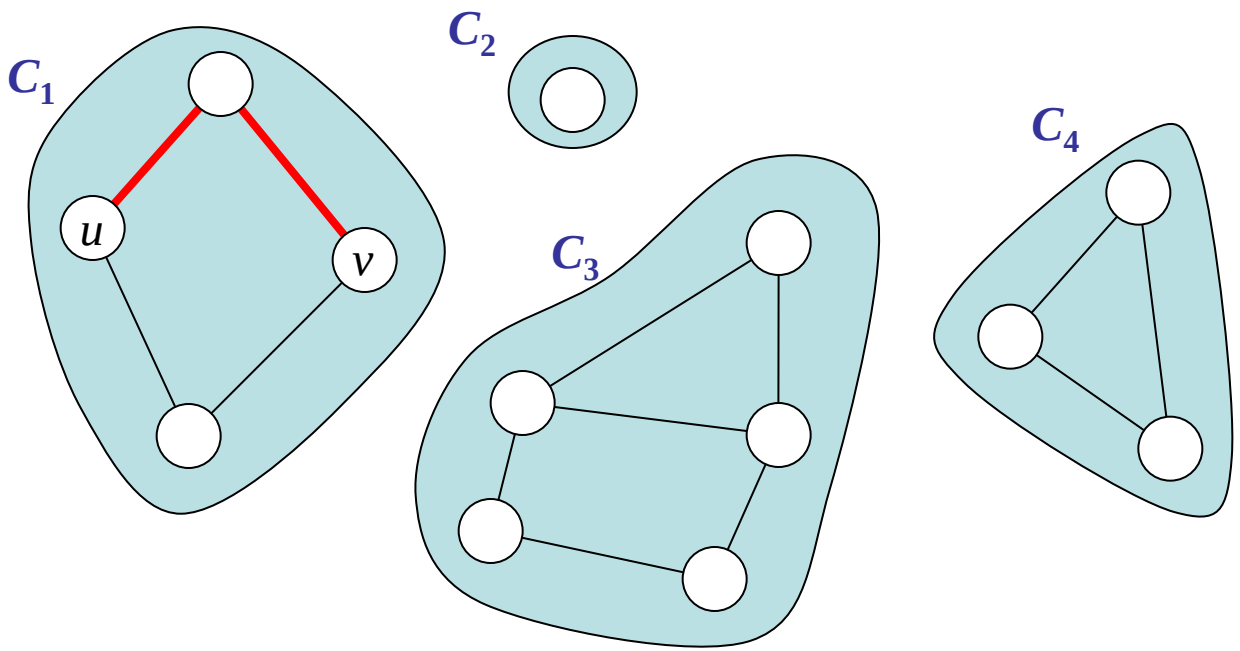
1 call DFS(G) to compute finishing times $v.f$ for each vertex v
2 as each vertex is finished, insert it onto the front of a linked list
3 **return** the linked list of vertices

Tid $O(m+n)$

Sammenhængskomponenter

Opdeling af knuderne i en uorienteret graf i **komponenter $C_1,\ldots,C_k$**, således at

u og v er i C_i hvis og kun hvis der er en sti mellem u og v



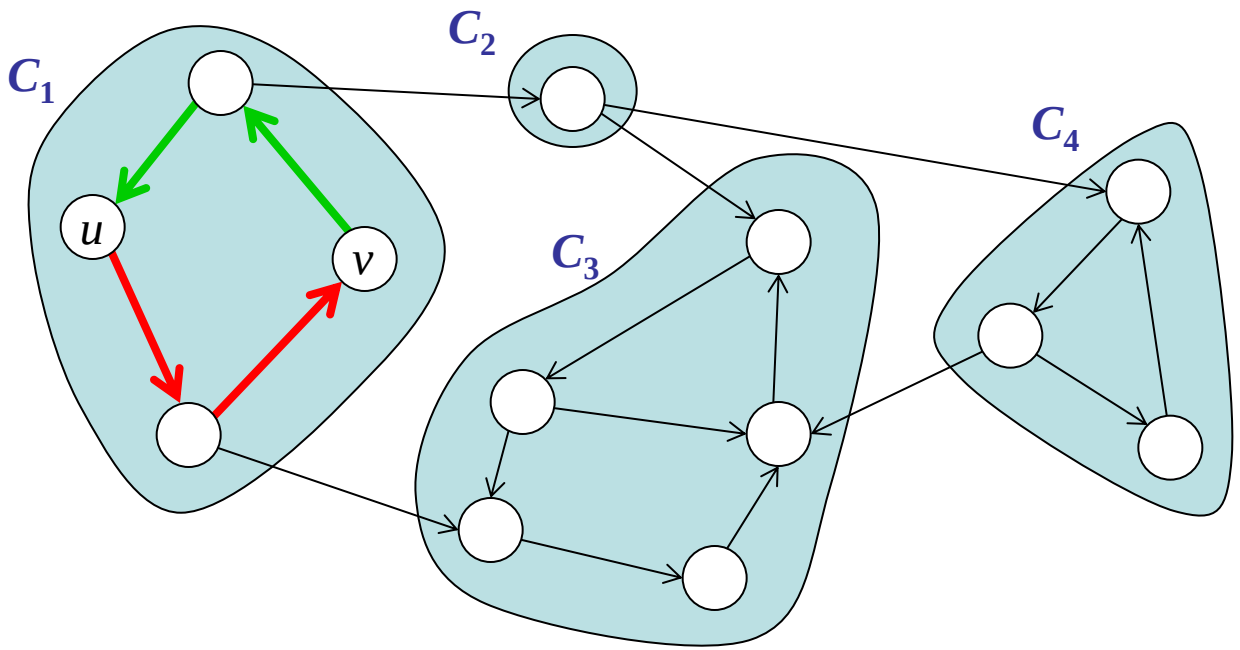
DFS/BFS,tid $O(m+n)$

Stærke Sammenhængskomponenter

Opdeling af knuderne i en orienteret graf i **komponenter $C_1,\ldots,C_k$**, således at

u og v er i C_i hvis og kun hvis der både er

- en sti fra u til v og
- en sti fra v til u



Tarjan 1972

Stærke Sammenhængskomponenter

STRONGLY-CONNECTED-COMPONENTS(G)

1 call DFS(G) to compute finishing times $u.f$ for each vertex u
2 compute 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 separate strongly connected component

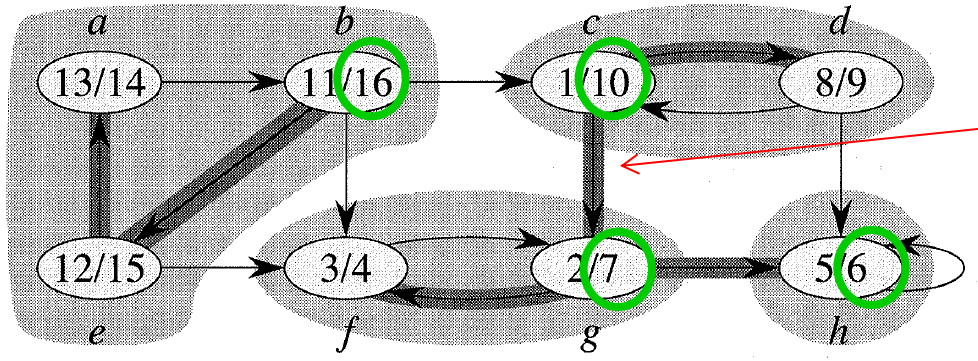
```
DFS(G)
1  for each vertex u ∈ G.V
2      u.color = WHITE
3      u.π = NIL
4  time = 0
5  for each vertex u ∈ G.V
6      if u.color == WHITE
7          DFS-VISIT(G, u)
```

```
DFS-VISIT(G, u)
 1  time = time + 1            // white vertex u has just been discovered
 2  u.d = time
 3  u.color = GRAY
 4  for each v ∈ G.Adj[u]      // explore edge (u, v)
 5      if v.color == WHITE
 6          v.π = u
 7          DFS-VISIT(G, v)
 8  u.color = BLACK            // blacken u; it is finished
 9  time = time + 1
10  u.f = time
```

Tid $O(m+n)$

Stærke Sammenhængskomponenter

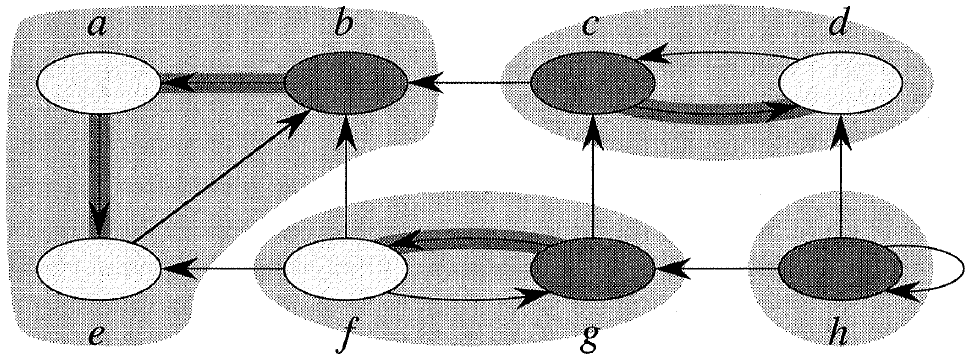
G

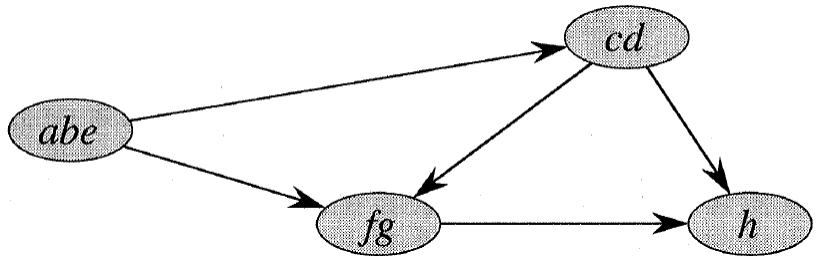


DFS trækanter mellem to stærke sammenhængs-komponenter

De største finising tider i hver komponent udgør en (omvendt) topologisk sortering af komponenterne

G^T



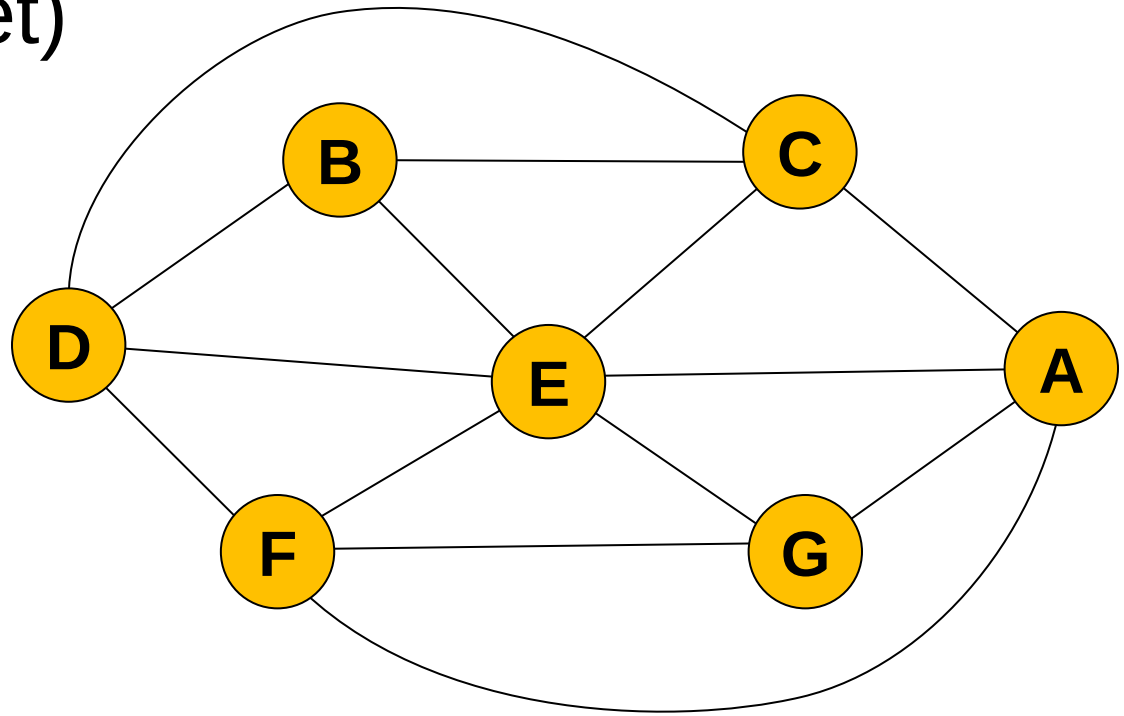


Ikke pensum

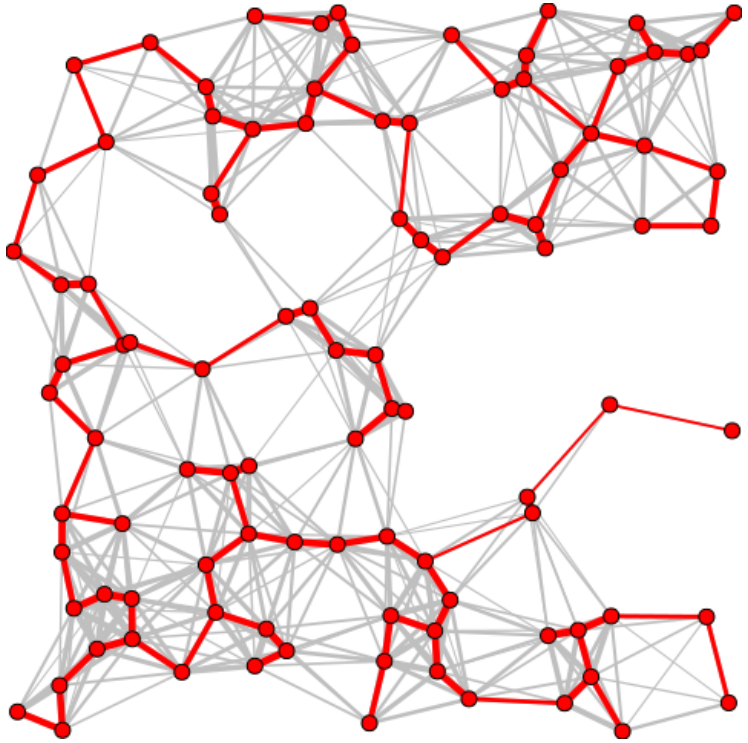
Planaritets test

- Planaritets test = givet en uorienteret graf, afgør om grafen er planar, dvs. kan tegnes i planen uden at kanterne krydser
- Kan løses vha DFS (kompliceret)

***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}}**



Hopcroft, Tarjan, *Efficient planarity testing*, JACM 1974



Algoritmer og Datastrukturer

Minimum udspændende træer (MST)
[CLRS, kapitel 21]

Kontaimu p. prof. Dr V. Bydžovskému v Hluboké 5.x.26. O. Borůvka.

ZVLÁŠTNÍ OTISK Z ČASOPISU „ELEKTROTECHNICKÝ OBZOR“
Roč. 15. Čís. 10. Praha III., Cihelná 102. 5. března 1926.

Dr. OTAKAR BORŮVKA:

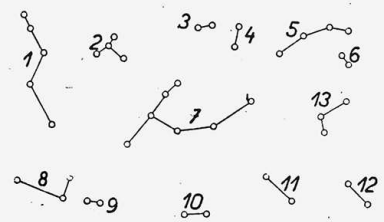
Příspěvek k řešení otázky ekonomické stavby elektrovodných sítí.

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

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

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

Obr. 1.

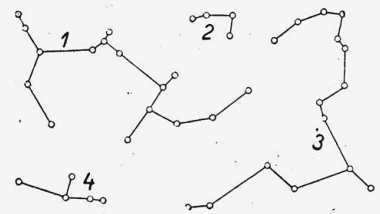


Obr. 2.

Jest zřejmé, že řešení této úlohy může míti v elektrotechnické praksi při návrzích plánů elektrovodných sítí jistou důležitost; z toho důvodu je zde stručně na příkladě vyložím. Čtenáře, jenž by se o věc blíže zajímal, odkazuji na citované pojednání.

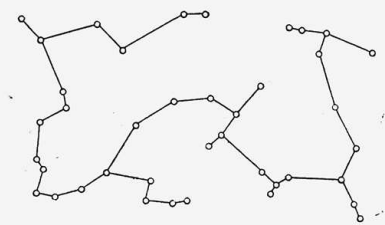
Řešení úlohy provedu v případě 40 bodů daných v obr. 1.

Každý z daných bodů spojím s bodem nejbližším. Tedy na př. bod 1 s bodem 2, bod 2 s bodem 3, bod 3 s bodem 4 (bod 4 s bodem 3), bod 5 s bodem 2, bod 6 s bodem 5, bod 7 s bodem 6, bod 8 s bodem 9, (bod 9 s bodem 8) atd. Obdržím řadu polygonálních tahů 1, 2,.... 13 (obr. 2).



Obr. 3.

Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy na př. 1 s tahem 2, (tah 2 s tahem 1), tah 3 s tahem 4, (tah 4 s tahem 3) atd. Obdržím řadu polygonálních tahů 1, 2,.... 4 (obr. 3).



Obr. 4.

Orazec převrácen.

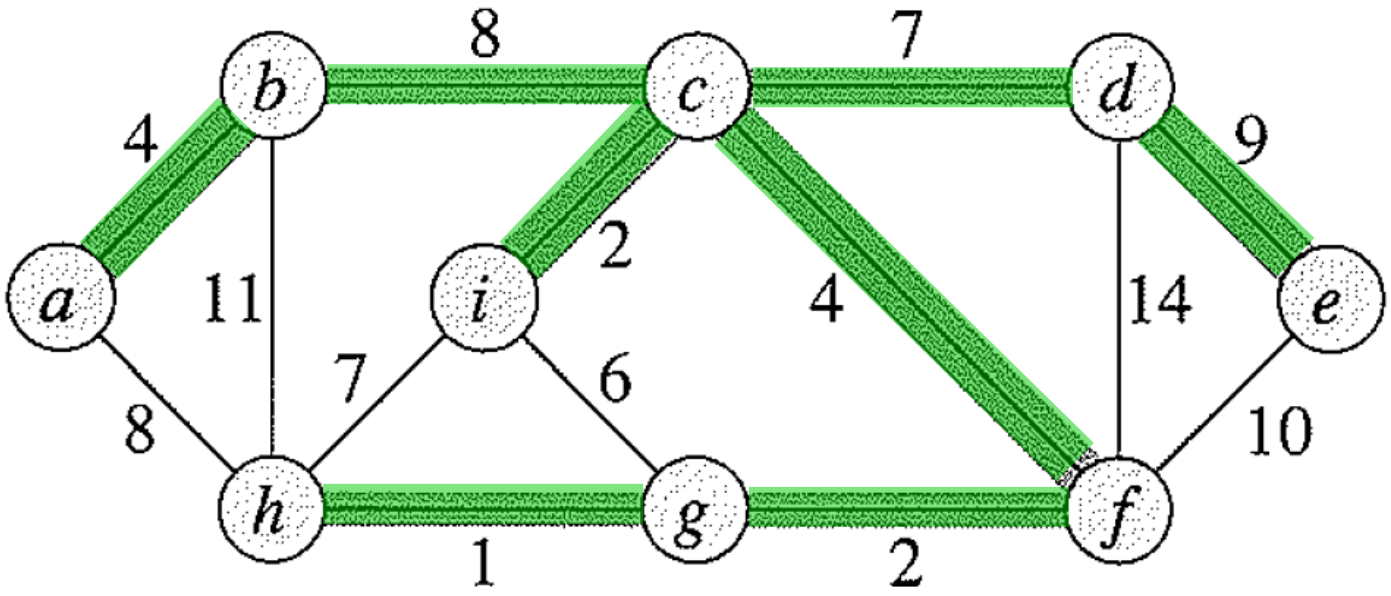
Každý z nich spojím nejkratším způsobem s tahem nejbližším. Tedy tah 1 s tahem 3, tah 2 s tahem 3 (tah 3 s tahem 1), tah 4 s tahem 1. Obdržím konečně jediný polygonální tah (obr. 4), jenž řeší danou úlohu.

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

Matematický ústav Masarykovy university v Brně, v lednu 1926.

Otakar Borůvka. *Příspěvek k řešení otázky ekonomické stavby elektrovodních sítí*. Elektronický Obzor, 15:153–154 (1926)

Minimum Udspændende Træ (MST)

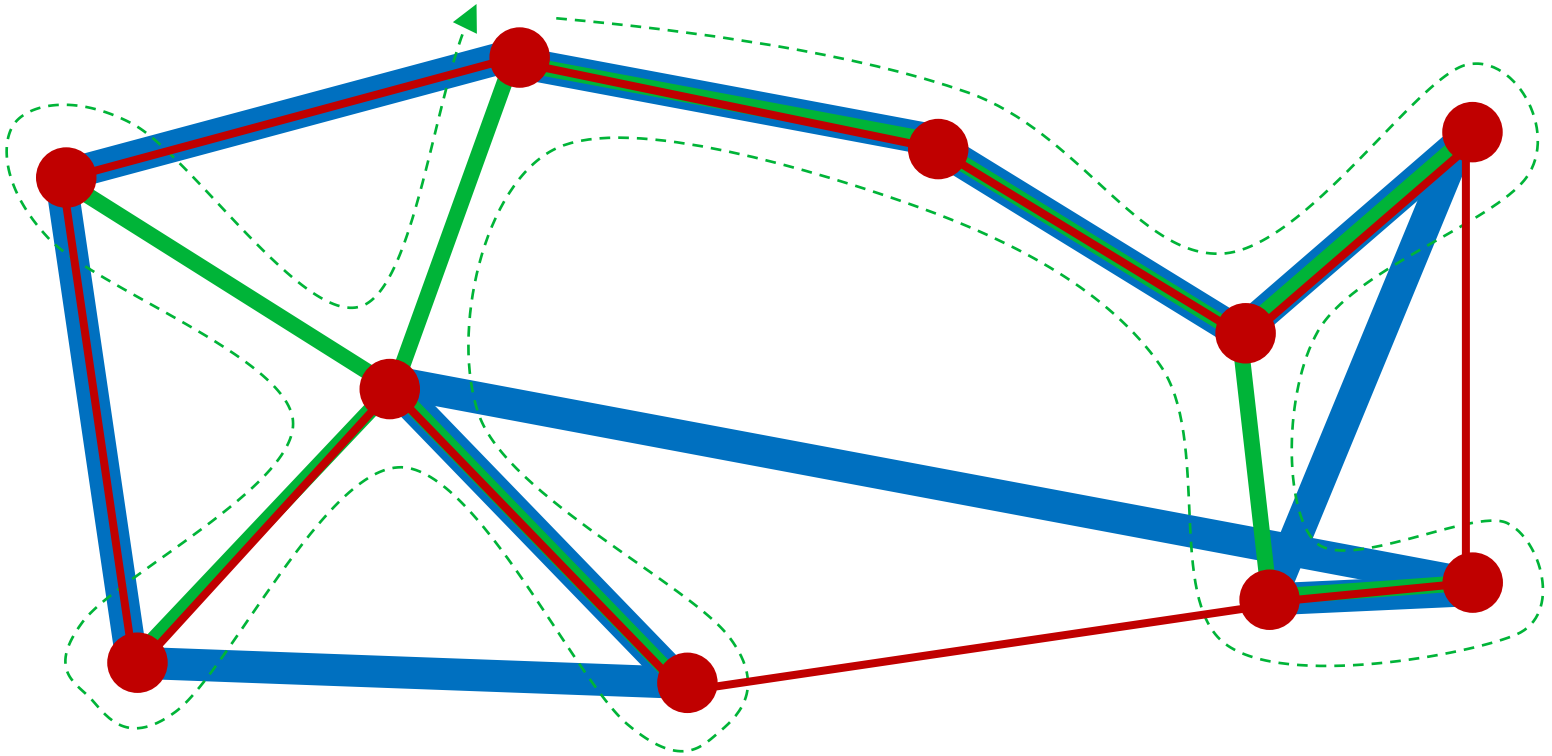


Problem

Find et udspændende træ for en sammenhængende uorienteret vægtet graf således at summen af kanterne er mindst mulig

IKKE PENSUM

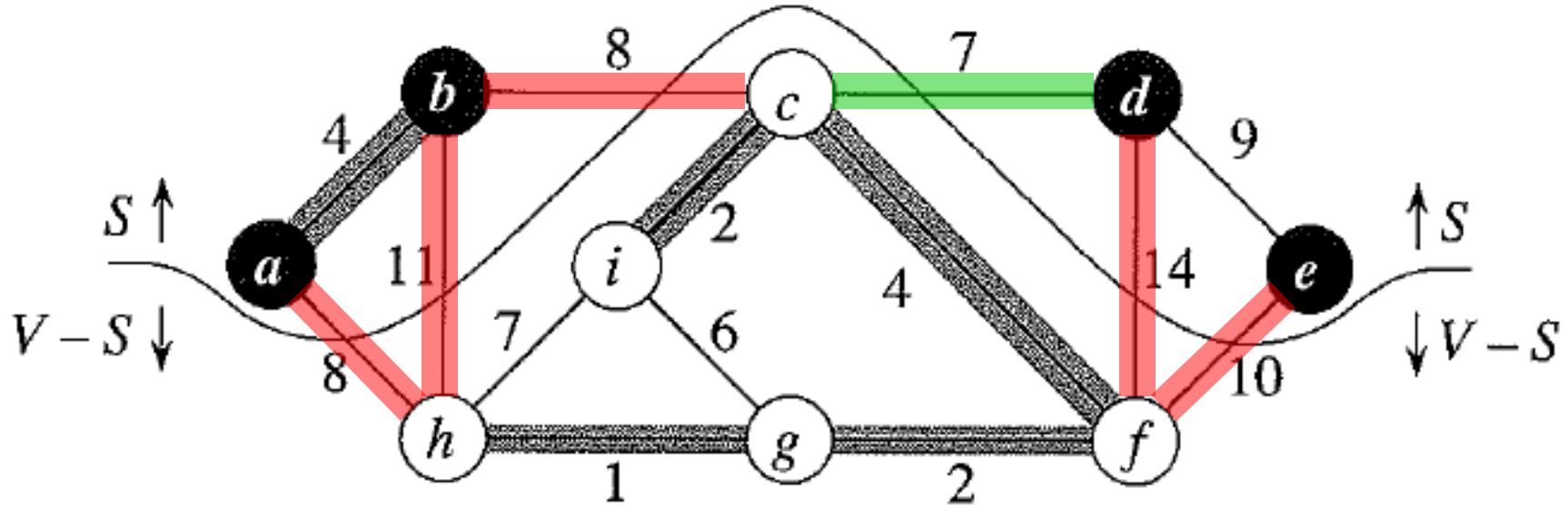
En anvendelse af MST :
(Euklidisk) Korteste Hamiltonske Cykel



Sætning : Touren fundet vha. et **MST**
er en 2-approximation til en **korteste cykel**

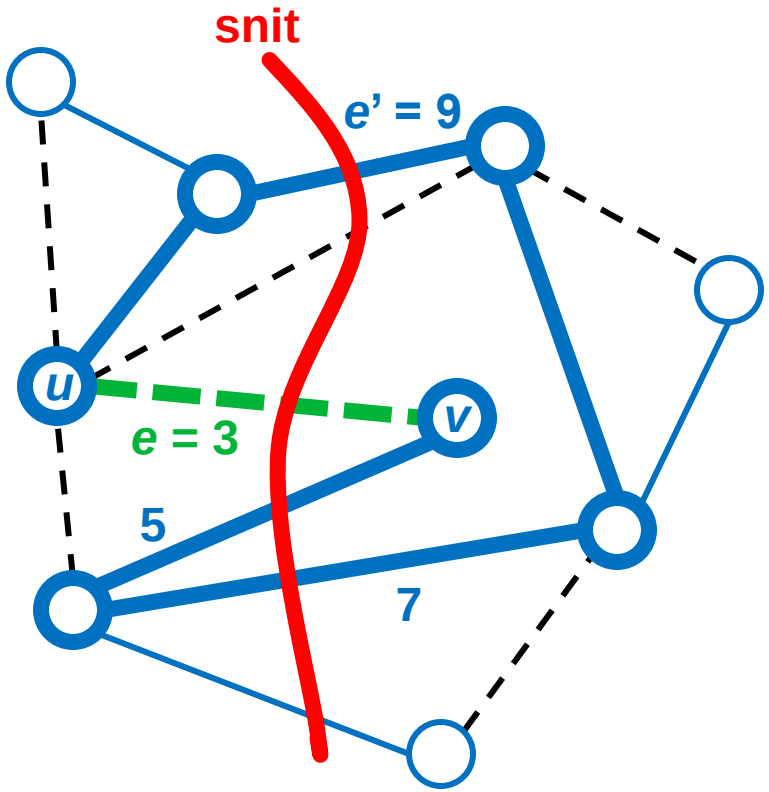
(|**Touren**| ≤ 2·|**MST**| og |**MST**| ≤ |**korteste cykel**|)

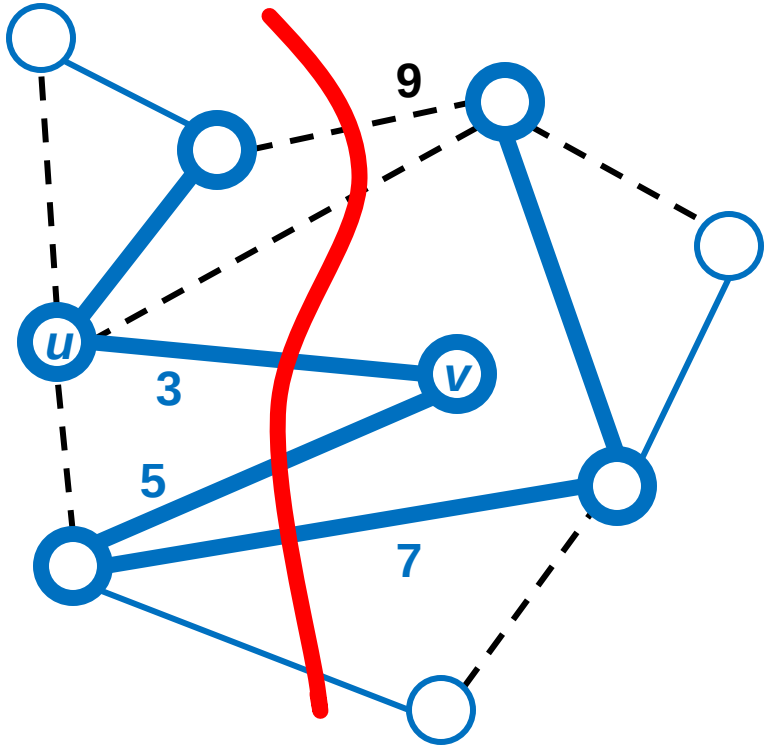
Minimum Udspændende Træer: Snit



Sætning

Hvis alle vægte er forskellige, så gælder der for *ethvert* **snit (*S*,*V*-*S*)** at den **letteste kant** der krydser snittet er med i et minimum udspændende træ





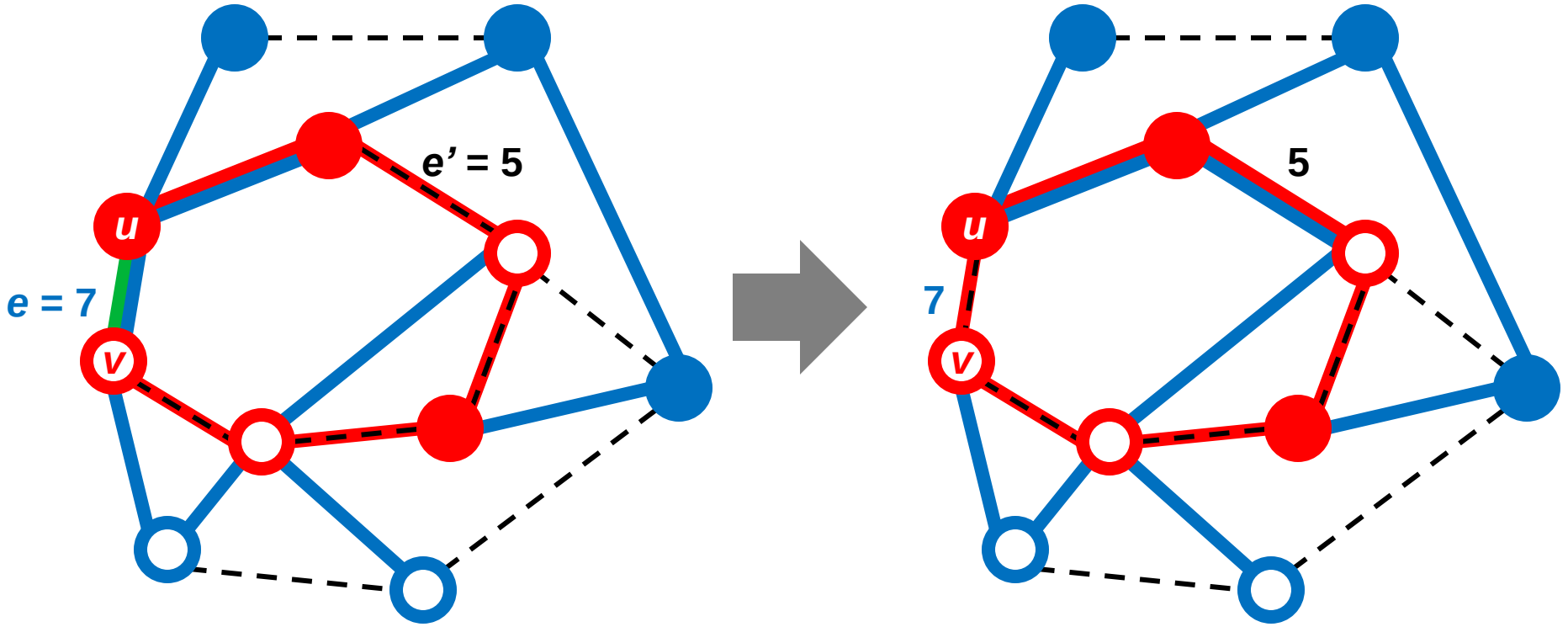
Antag modsætningsvis et MST med et snit
hvor mindste kant *e* ikke er med i MST

Nyt udspændende træ
med mindre vægt

Sætning

Hvis alle vægte er forskellige, så gælder der for *ethvert* **snit (*S*,*V*-*S*)** at den **letteste kant** der krydser snittet er med i et minimum udspændende træ

Sætning
Hvis alle vægte er forskellige, så gælder der for *enhver* **cykel** at den **tungeste kant i cyklen** ikke er med i et minimum udspændende træ



Antag modsætningsvis et MST og cykel
hvor tungeste kant *e* er med i MST

Nyt udspændende træ
med mindre vægt

Minimum Udspændende Træer: Grådig generel algoritme

```
GENERIC-MST(G, w)
1  A = ∅
2  while A does not form a spanning tree
3      find an edge (u, v) that is safe for A
4      A = A ∪ {(u, v)}
5  return A
```

En letteste kant over et snit (som ikke allerede indeholder kanter fra *A*)

Kruskall's Algoritme

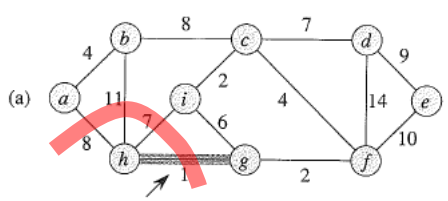
```
MST-KRUSKAL(G, w)
1  A = ∅
2  for each vertex v ∈ G.V
3       MAKE-SET(v)
4  sort the edges of G.E into nondecreasing order by weight w
5  for each edge (u, v) ∈ G.E, taken in nondecreasing order by weight
6       if FIND-SET(u) ≠ FIND-SET(v)
7            A = A ∪ {(u, v)}
8            UNION(u, v)
9  return A
```

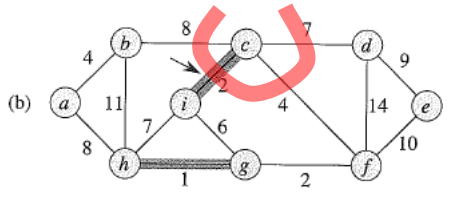
flaskehals i algoritmen

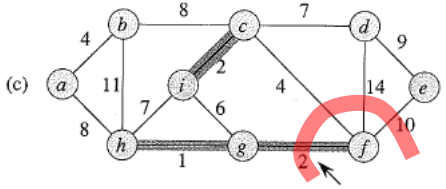
Union-Find datastruktur

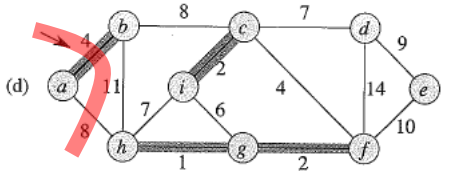
Tid $O(m \cdot \log n)$

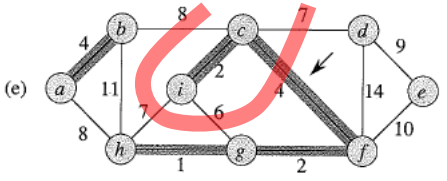
Kruskall's Algoritme: Eksempel

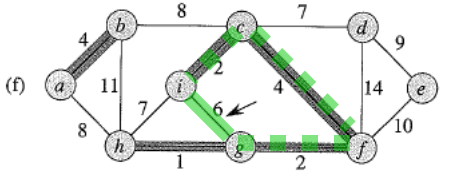


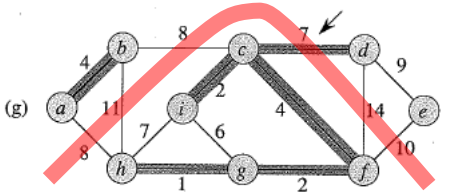


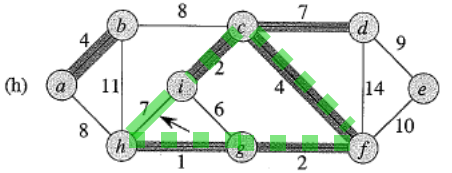


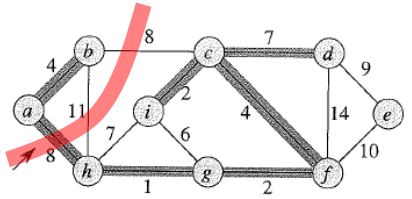


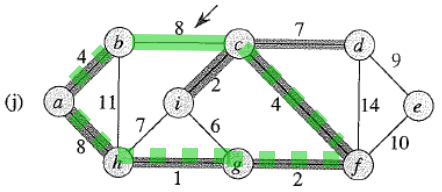


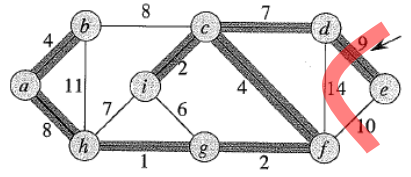


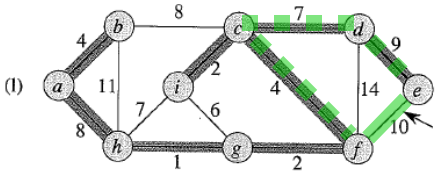


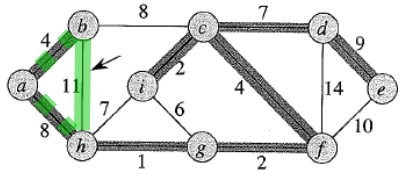


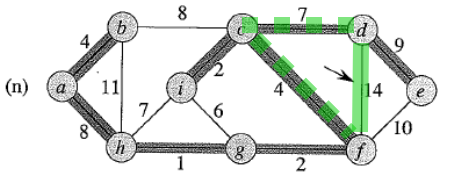










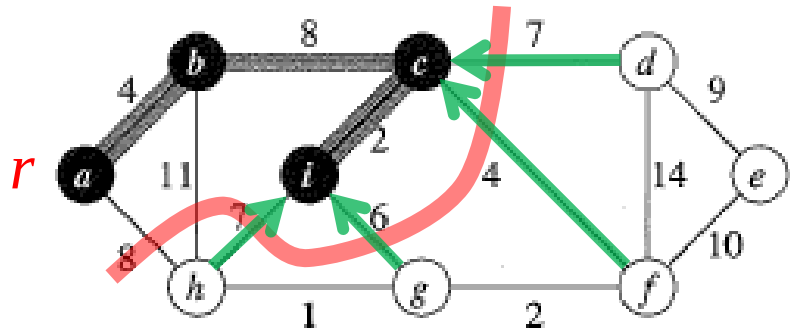


Kantene betrages efter stigende vægte (angivet med ↗)

For hver kant er angivet snittet hvor den er en letteste kant eller cyklen hvor den er en tungeste kant

Prim's Algoritme

```
MST-PRIM(G, w, r)
 1  for each u ∈ G.V
 2      u.key = ∞
 3      u.π = NIL
 4  r.key = 0
 5  Q = G.V
 6  while Q ≠ ∅
 7      u = EXTRACT-MIN(Q)
 8      for each v ∈ G.Adj[u]
 9          if v ∈ Q and w(u, v) < v.key
10              v.π = u
11              v.key = w(u, v)
```



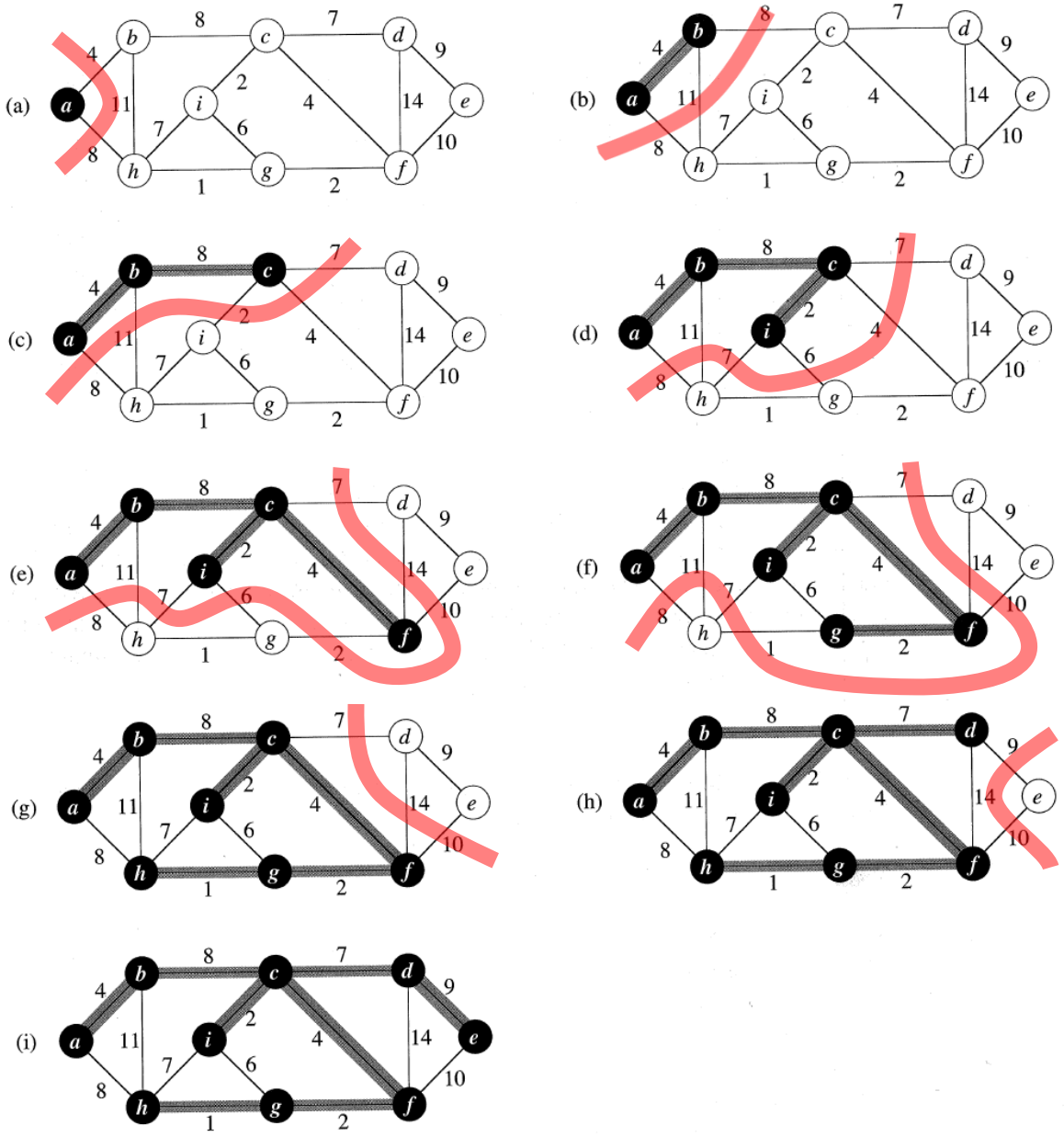
flaskehals i algoritmen - prioritetskø

x *n*

x *m*

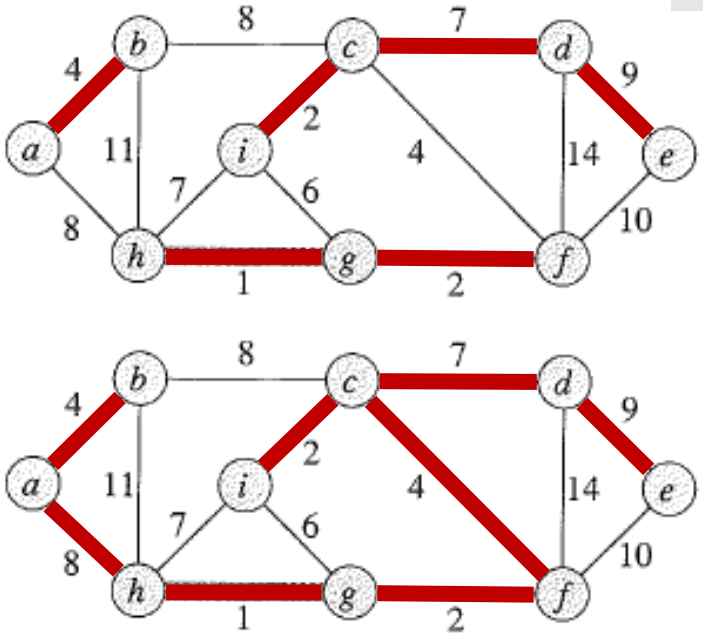
Tid $O(m \cdot \log n)$

Prim's Algoritme: Eksempel



IKKE PENSUM

Borůvka's Algoritme




```
MST-BORUVKA(G, w)
1   A = ∅
2   for each vertex v ∈ G.V
3       MAKE-SET(v)
3   while |A| < n - 1
4       B = ∅
5       for each set S
6           let (u, v) be lightest edge incident to S, i.e. u ∈ S and v ∉ S
7           if such (u, v) exists
8               B = B ∪ {(u, v)}
8       for each edge  (u, v) ∈ B
9           if FIND-SET(u) ≠ FIND-SET(v)
10              A = A ∪ {(u, v)}
11              UNION(u, v)
12  return A
```

Efter *i* iterationer af while **har hver mængde størrelse ≥ 2^i (eller er udspændende)**

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

Minimum Udspænding Træer

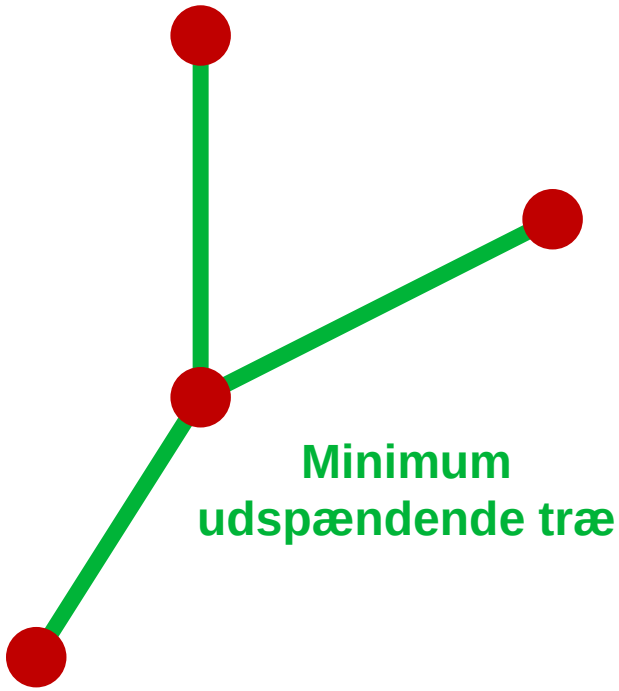
Kruskall (1956) (mange træer; sorterer kanterne)	$O(m \cdot \log n)$
Prim (1930) (et træ; prioritetskø over naboknuder)	$O(m \cdot \log n)$
	$O(m + n \cdot \log n)$ (Fibonnaci heaps [CLRS, 3. udg. kapitel 19] (1984))
Borůvka (1926) (mange træer samtidigt; kontraktion)	$O(m \cdot \log n)$
Fredman, Tarjan (1984) (Borůvka (1926) + Fibonnaci heaps)	$O(m \cdot \log^* n)$
Chazelle (1997)	$O(m \cdot \alpha(m,n))$
Pettie, Ramachandran (2000)	? (men optimal deterministisk sammenligningsbaseret)
Karger, Klein, Tarjan (1995) (Randomiseret) Fredman, Willard (1994) (RAM)	$O(m)$

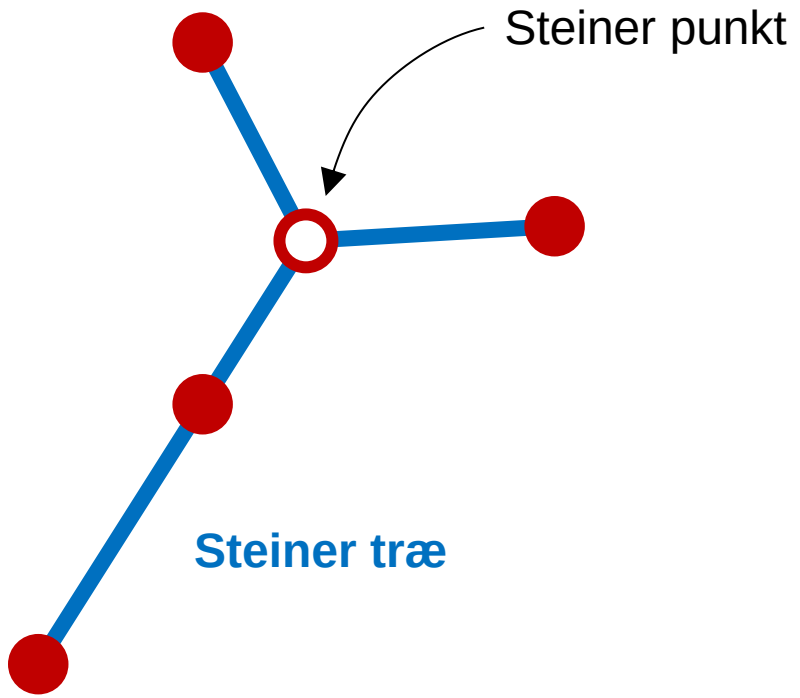
IKKE PENSUM

Jakob Steiner [1796-1863]

(Euklidiske) Steiner Træer

Givet en mængde punkter, find et træ med minimum vægt som forbinder alle knuder, muligvis ved at tilføje **nye punkter**





Conjecture (Steiner Ratio): $|\text{MST}| \leq \frac{2}{\sqrt{3}} \cdot |\text{Steiner Tree}|$

NP hårdt; polynomial-time approximation scheme (PTAS)

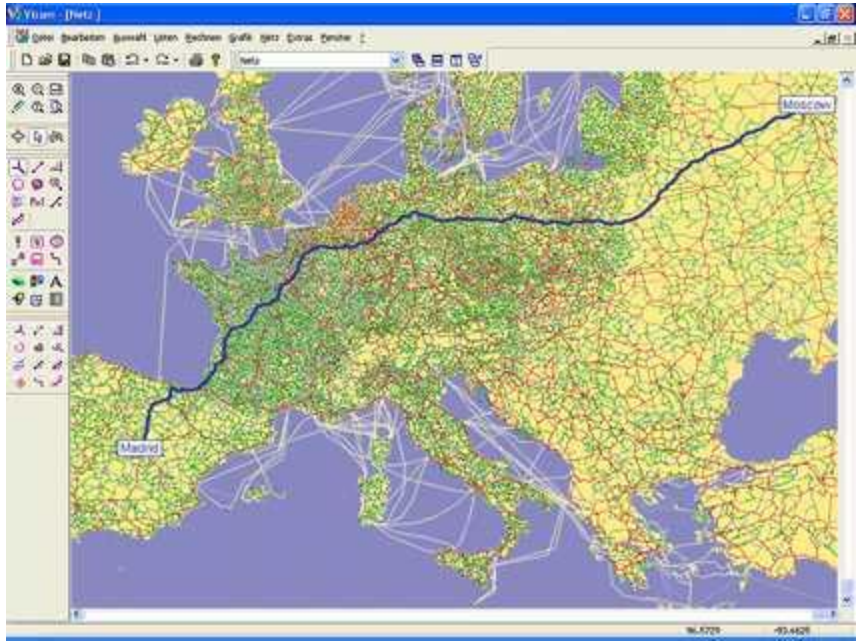
Algoritmer og Datastrukturer

Korteste veje – fra en knude (SSSP)
[CLRS, kapitel 22]

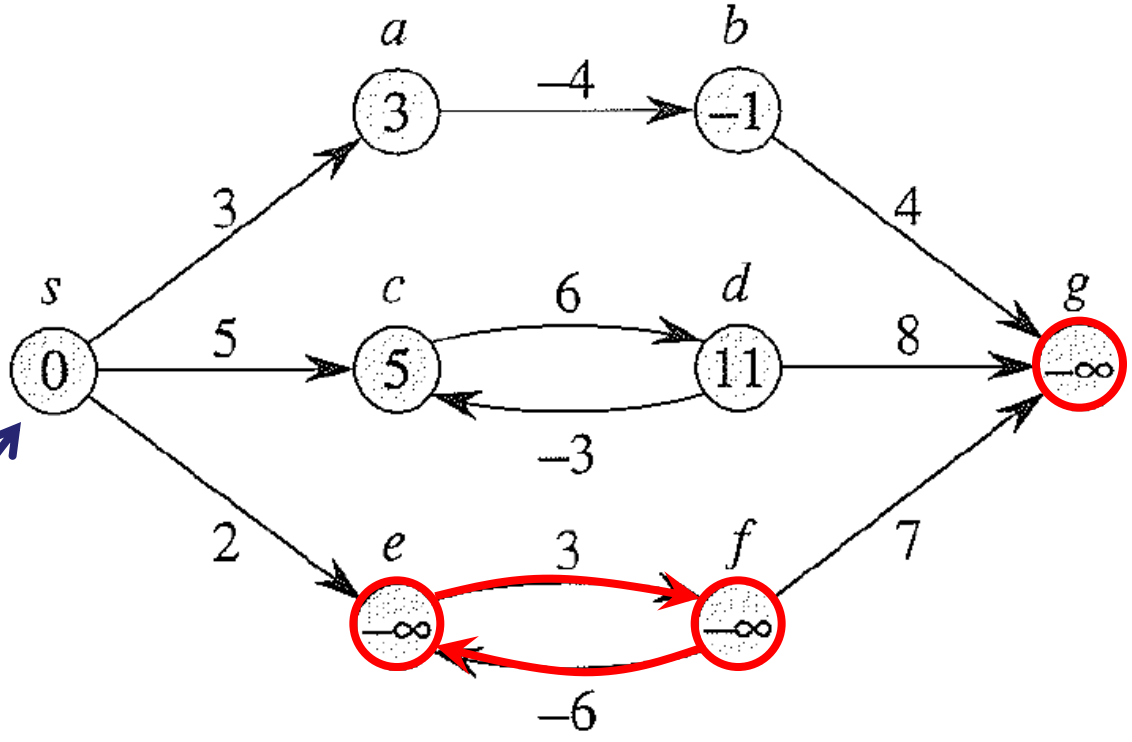
Kort over Vest-Europa

- **18.029.721 knuder**
- **42.199.587 orienterede kanter**

traffic mobility logistics.

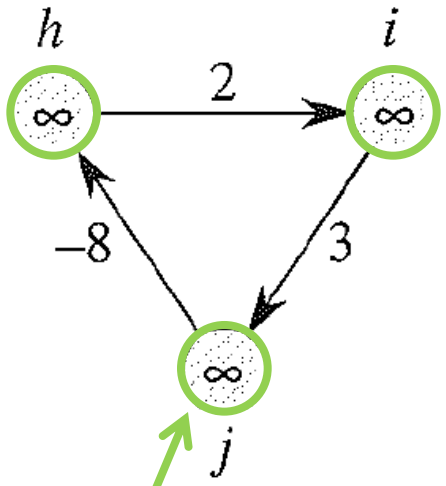


Eksempel: Korteste veje fra s



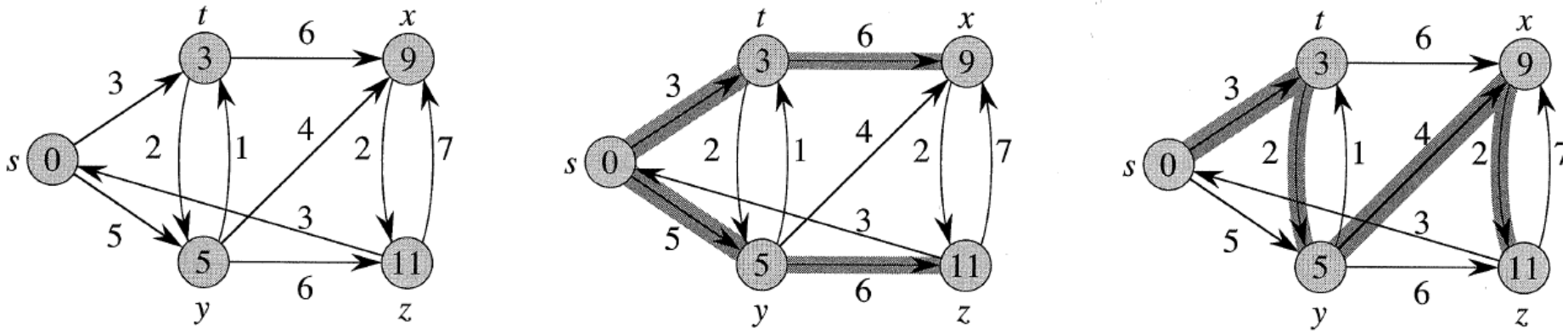
Start knude (kilde)

Negativ cykel



Uforbundet til s

Eksempel: Korteste veje træer

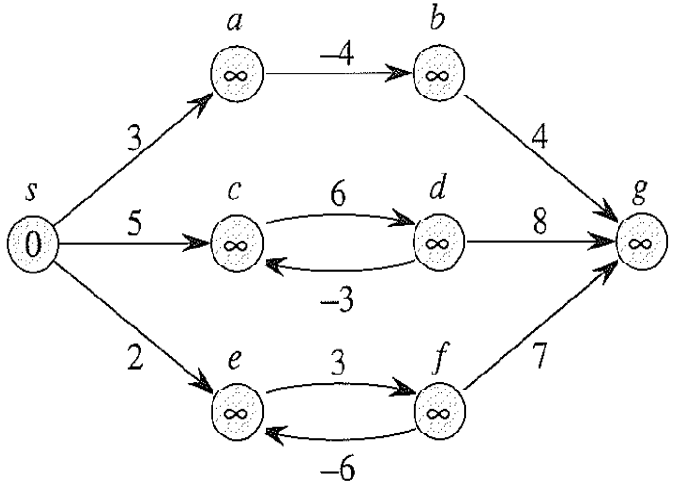


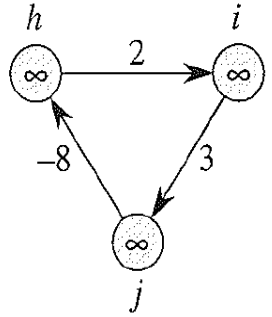
2 forskellige korteste veje træer der repræsenterer stier fra s med samme længde

Korteste Veje Estimater : Initialisering

INITIALIZE-SINGLE-SOURCE(G, s)

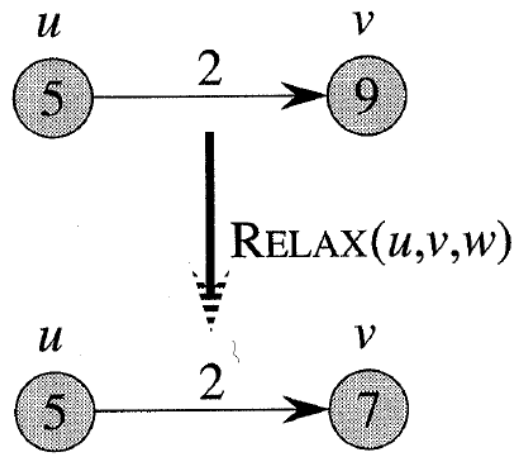
1 **for** each vertex $v \in G.V$
2 $\quad v.d = \infty$
3 $\quad v.\pi = \text{NIL}$
4 $s.d = 0$



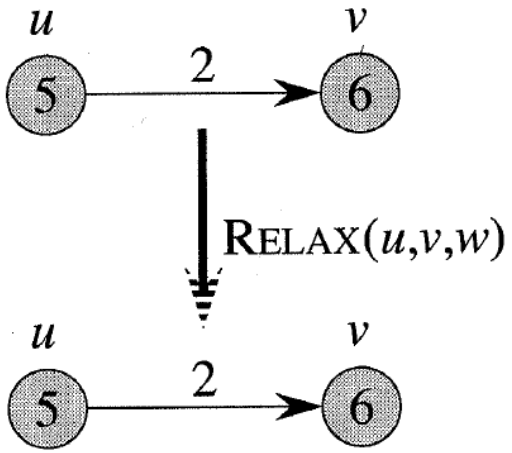


Korteste Veje Estimater : Relax

```
RELAX(u, v, w)
1   if v.d > u.d + w(u, v)
2       v.d = u.d + w(u, v)
3       v.π = u
```



Kortere afstand til *v* fundet (opdater *v.d* og *v.π = u*)



Forbedrer ikke afstanden til *v*

Invariant

Hvis $v.d < \infty$ så findes en sti fra *s* til *v* med afstand *v.d* og *v.π* som næstsidste knude

Bellman-Ford:

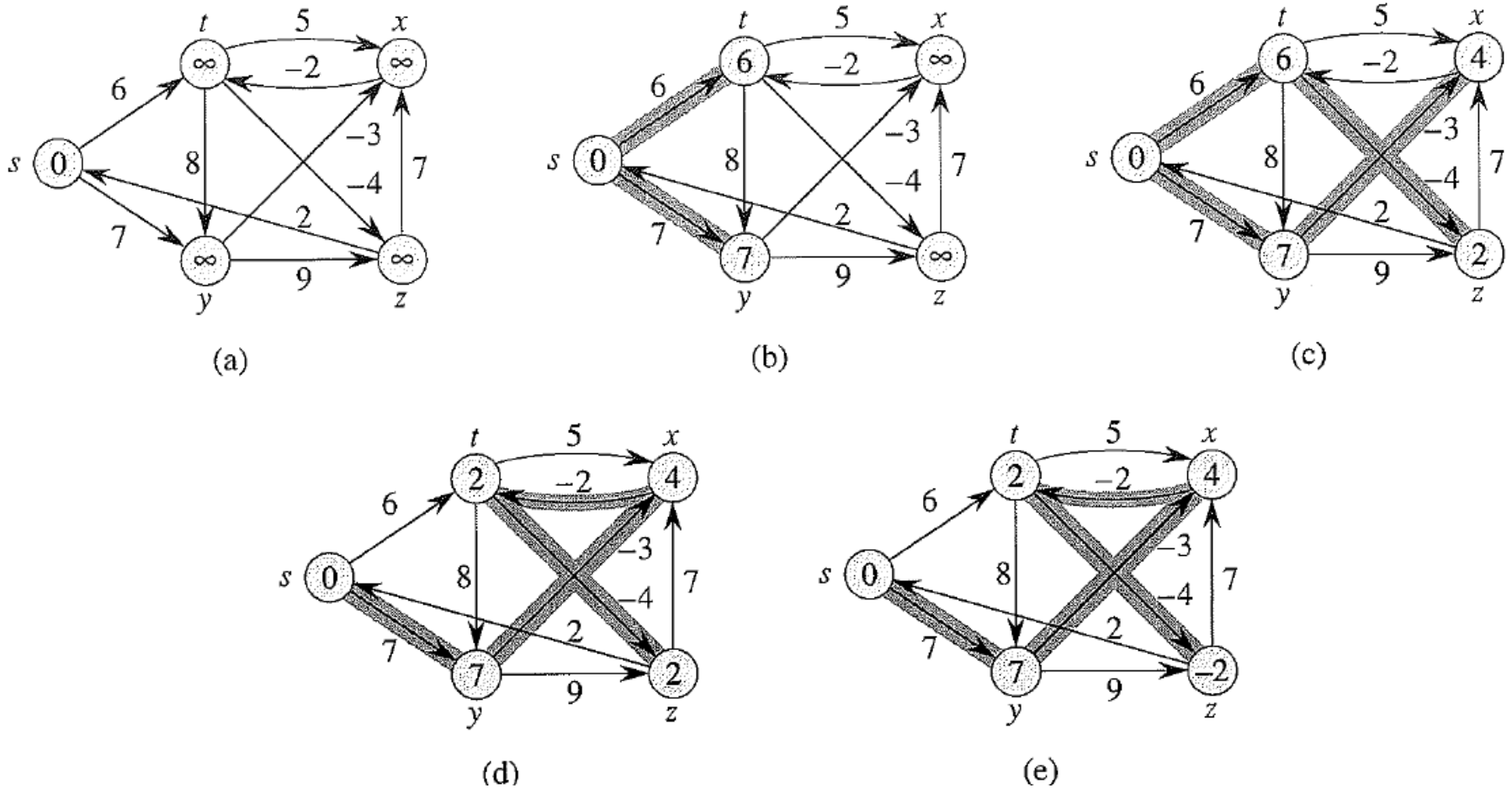
Korteste Veje i Grafer med Negative Vægte

```
BELLMAN-FORD(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  for i = 1 to |G.V| − 1
3      for each edge (u, v) ∈ G.E
4          RELAX(u, v, w)
5  for each edge (u, v) ∈ G.E
6      if v.d > u.d + w(u, v)
7          return FALSE
8  return TRUE
```

Check for negativ cykel

Tid $O(nm)$

Bellman-Ford: Eksempel



Antag der finds en **negative cykel**

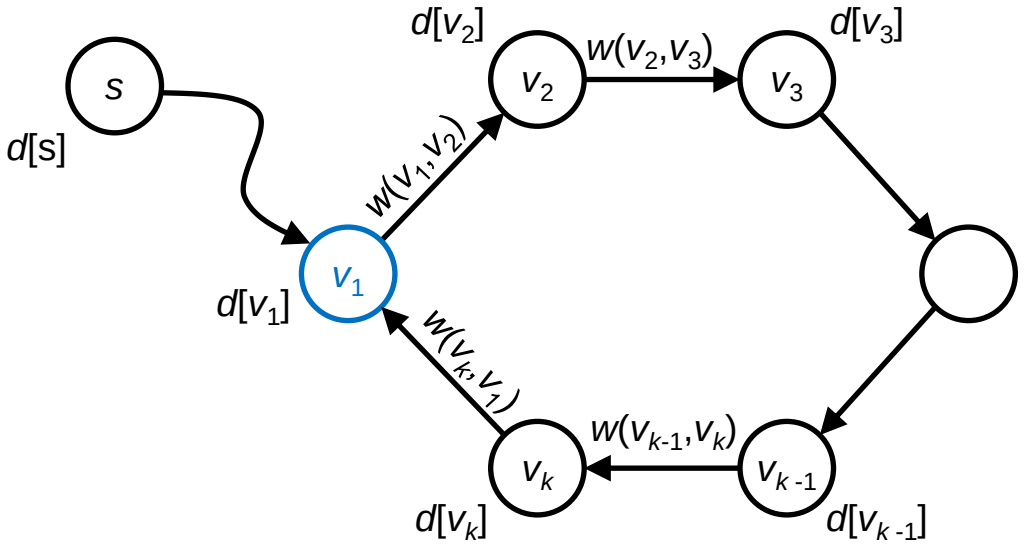
$$w(v_1,v_2) + w(v_2,v_3) + \cdots + w(v_{k-1},v_k) + w(v_k,v_1) < 0$$

og **ikke** muligt at lave **RELAX**

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

heraf følger **modstriden** (hvis ikke alle $d[v_i] = \infty$)

$$d[v_1] \leq d[v_k] + w(v_k,v_1) \leq \left(d[v_{k-1}] + w(v_{k-1},v_k)\right) + w(v_k,v_1) \leq$$
$$\cdots \leq d[v_1] + w(v_1,v_2) + w(v_2,v_3) + \cdots + w(v_{k-1},v_k) + w(v_k,v_1) < d[v_1]$$

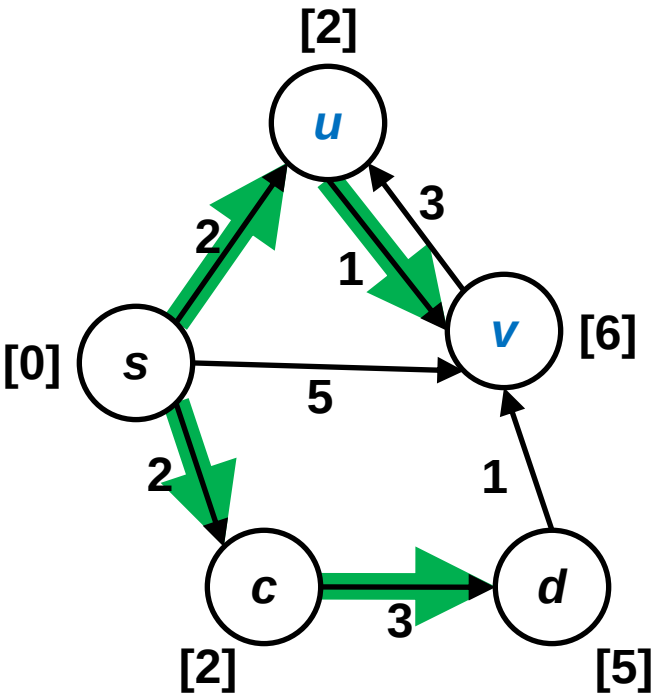


Sætning

Betragt et (ukendt) korteste veje træ T hvori (u,v) er en kant.

Antag den aktuelle $d[u]$ er den korteste afstand til u.

RELAX(u,v,w) medfører at $d[v]$ også er en kortest afstand til v (hvis den ikke allerede var det).



⇓

Korollar

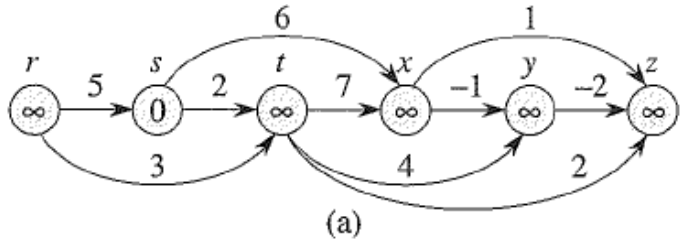
Bellman-Ford finder de rigtige korteste afstande efter $n - 1$ RELAX iterationer.

Korteste Veje i Acycliske Grafer

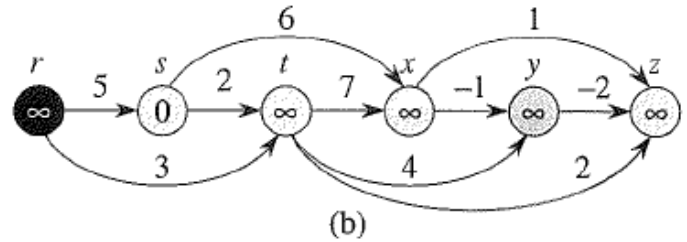
```
DAG-SHORTEST-PATHS(G, w, s)
1  topologically sort the vertices of G
2  INITIALIZE-SINGLE-SOURCE(G, s)
3  for each vertex u, taken in topologically sorted order
4       for each vertex v ∈ G.Adj[u]
5            RELAX(u, v, w)
```

Tid $O(n+m)$

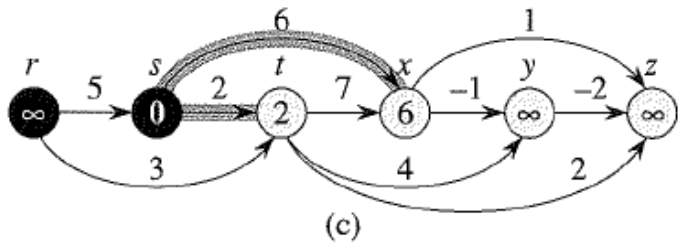
Acykliske Grafer : Eksempel



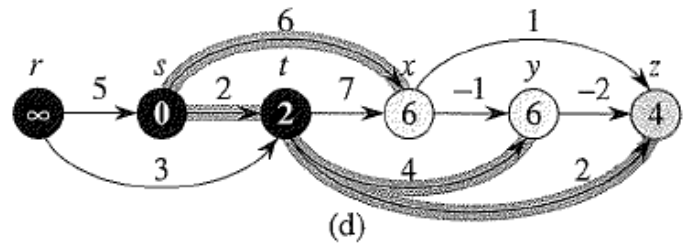
(a)



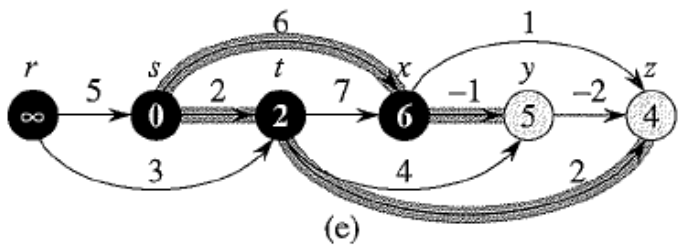
(b)



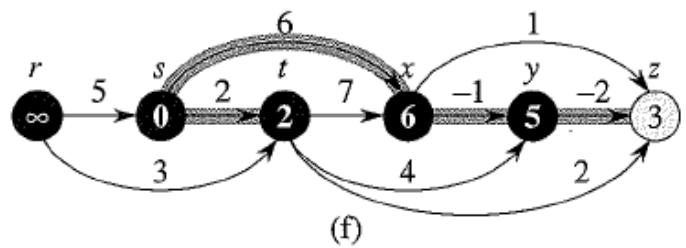
(c)



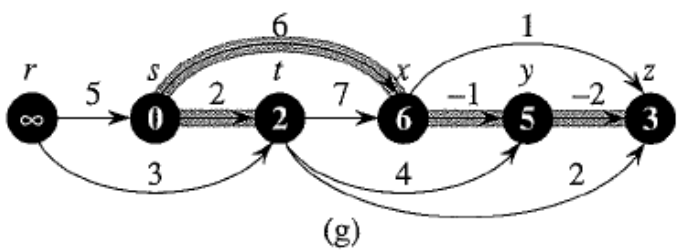
(d)



(e)



(f)



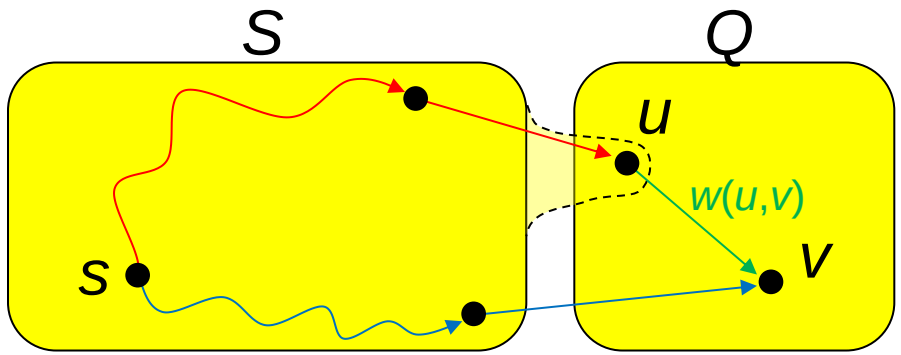
(g)

Dijkstra: Korteste Veje i Grafer uden Negative Vægte

Invarianter

i) $d[v]$ = korteste afstand fra s til v **via knuder i S**

ii) $\forall p \in S, q \in Q : d[p] \le d[q]$




```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S = ∅
3  Q = G.V
4  while Q ≠ ∅
5      u = EXTRACT-MIN(Q)
6      S = S ∪ {u}
7      for each vertex v ∈ G.Adj[u]
8          RELAX(u, v, w)
```

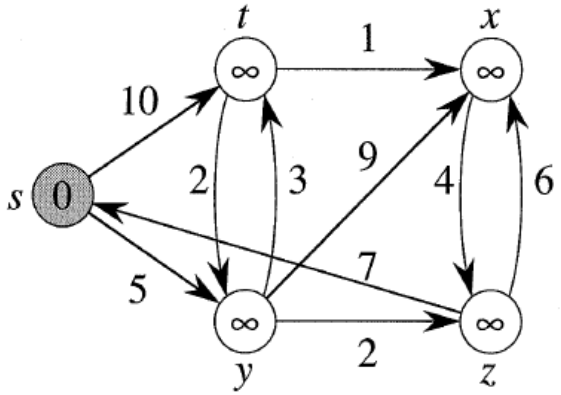
Q = prioritets kø, prioritet = *d*
(besøger knuderne efter stigende afstand fra *s*)

Opdaterer prioriteten af *v* i *Q*

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

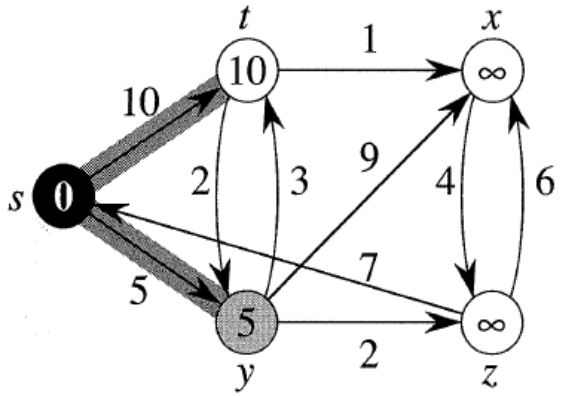
Dijkstra : Eksempel

Q = (s,0) (t,∞) (x,∞) (y,∞) (z,∞)



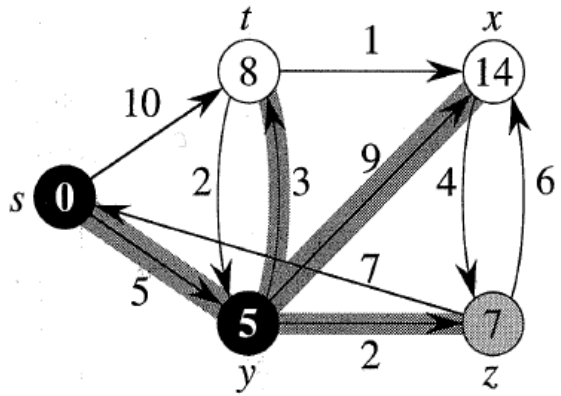
(a)

Q = (y,5) (t,10) (x,∞) (z,∞)



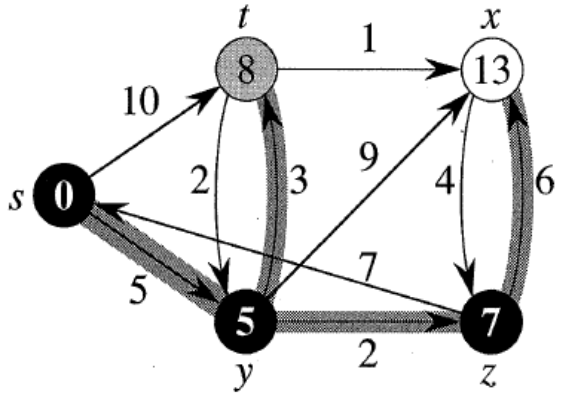
(b)

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



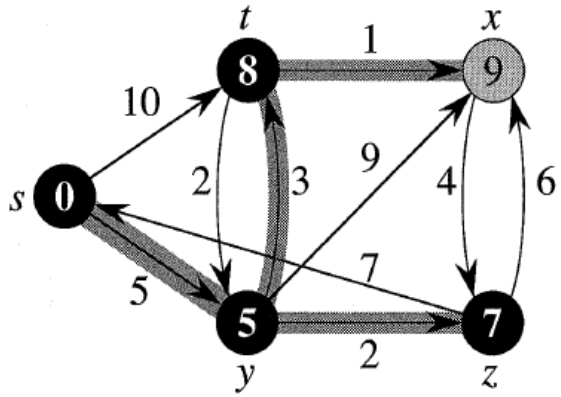
(c)

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



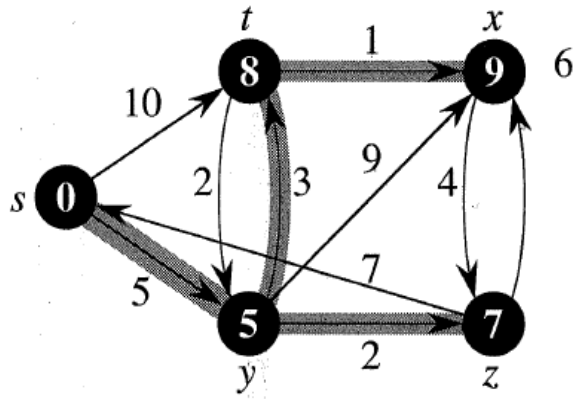
(d)

Q = (x,9)



(e)

Q =



(f)

Dijkstra Implementation:

Prioritetskø med gentagne indsættelser

```
DIJKSTRA(G, w, s)
1   INITIALIZE-SINGLE-SOURCE(G, s)
2   S = ∅
3   Q = {(s.d, s)}
2   while Q ≠ ∅
3       (δ, u) = EXTRACT-MIN(Q)
4       if δ = u.d
5           S = S ∪ { u }
6           for each vertex v ∈ G.Adj[u]
7               if v.d > u.d + w(u, v)
8                   v.d = u.d + w(u, v)
9                   v.π = u
10                  INSERT(Q, (v.d, v))
```

(prioritet, element) par

Ignorer alle u hvor $\delta > u.d$

RELAX(u, v, w)

Samme knude kan indsættes flere gange, men med aftagende prioriteter

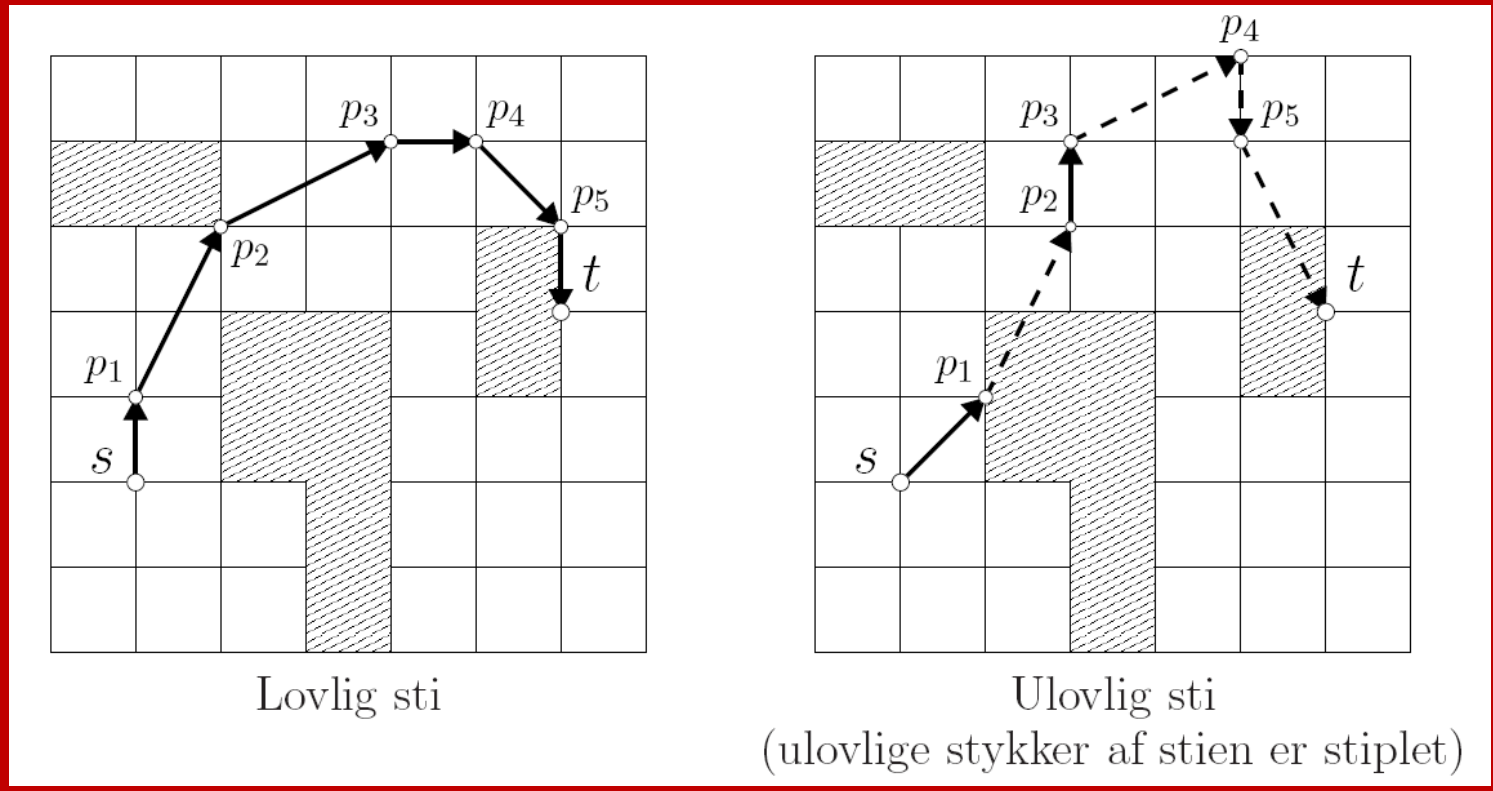
Opsummering

		SSSP En-til-alle korteste veje
Acykliske grafer (positive og negative vægte)		$O(n+m)$
Generelle grafer	Kun positive vægte	**Dijkstra** $O((n+m)\cdot\log n)$ $O(n^2+m)$
	Positive og negative vægte	**Bellman-Ford** $O(m\cdot n)$

Relaxer hver kant præcis én gang

Vektorrace

(find hurtigste vej fra *s* til *t*)



Eksamensopgave Sommeren 2009, opgave 3

Konstruer en uorienteret graf hvor man har en knude (i, j, x, y) for hvert par af et gitterpunkt (i, j) og hastighedsvektor (x, y), hvor hastighedsvektoren (x, y) fører til gitterpunktet (i, j) fra gitterpunktet $(i - x, j - y)$, og hvor $0 \leq i \leq n$, $0 \leq j \leq n$, $i - n \leq x \leq i$ og $j - n \leq y \leq j$. Dvs. grafen har $O(n^4)$ knuder. For alle par af knuder $v = (i, j, x, y)$ og $v' = (i', j', x', y')$ laver vi en kant imellem v og v' hvis og kun hvis følgende er opfyldt:

- $i' = i + x'$ og $j' = j + y'$
- $-1 \leq x' - x \leq 1$ og $-1 \leq y' - y \leq 1$
- alle cellerne der skæres af liniestykket fra (i, j) til (i', j') er frie

Givet (i, j, x, y) findes der højest 3^2 knuder (i', j', x', y') der opfylder de to første krav. Dvs. der er højest $O(n^4)$ potentielle kanter hvor det sidste krav kræver at vi for hver kant checker om $\leq 2n$ krydsende celler er fri. Grafen kan derfor konstrueres i tid $O(n^5)$. Endeligt checkes der vha. BFS om der en sti fra $s = (i_s, j_s, 0, 0)$ til $t = (i_t, j_t, 0, 0)$ i tid $O(n^4)$, hvor (i_s, j_s) og (i_t, j_t) er hhv. start og slut gitterpunktet. Total tid bliver $O(n^5)$.

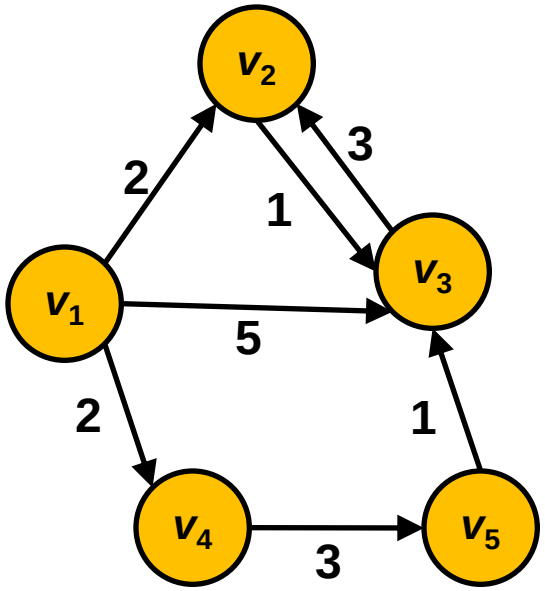
Note: Da man ikke må køre ud af gitteret og har begrænset acceleration og deacceleration har vi at den maksimale hastighed k i x og y-retningen skal opfylde $1 + 2 + \cdots + k \leq n$, dvs. $k = O(\sqrt{n})$. Dette medfører at antal knuder kan begrænses til $O(n \cdot n \cdot \sqrt{n} \cdot \sqrt{n}) = O(n^3)$ og konstruktionstiden til $O(n^3\sqrt{n})$.

Algoritmer og Datastrukturer

Korteste veje – mellem alle par af knuder (APSP)
[CLRS, kapitel 23]

Korteste Veje mellem alle Par af Knude

Hvis alle kanter har positive vægte
– kør Dijkstra's algoritme n gange,
én gang for hvert v_i som startknude
$O(n \cdot m \cdot \log n)$



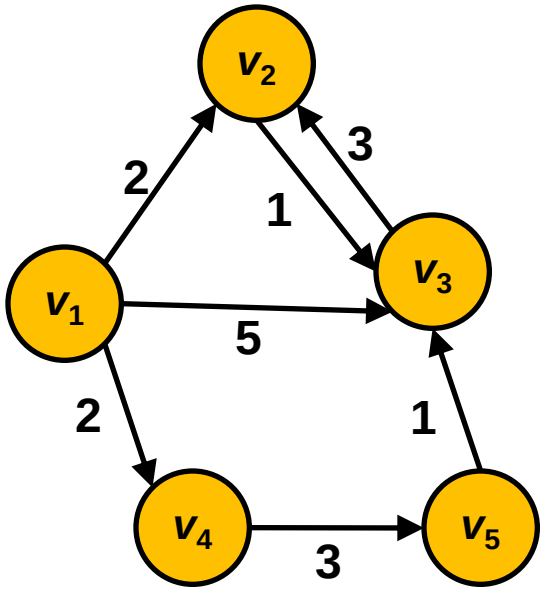
d_{ij}

	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL		5	NIL	4
5	NIL	3	5	NIL	NIL

```
PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, j)
1  if i == j
2        print i
3  elseif π_ij == NIL
4        print "no path from" i "to" j "exists"
5  else PRINT-ALL-PAIRS-SHORTEST-PATH(Π, i, π_ij)
6        print j
```



d_{ij}

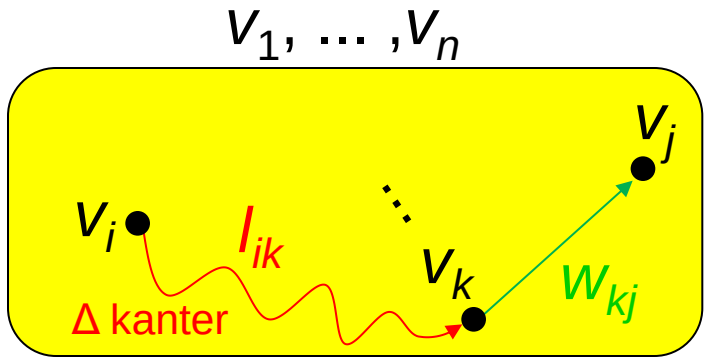
	1	2	3	4	5
1	0	2	3	2	5
2	+∞	0	1	+∞	+∞
3	+∞	3	0	+∞	+∞
4	+∞	7	4	0	3
5	+∞	4	1	+∞	0

π_{ij}

	1	2	3	4	5
1	NIL	1	2	1	4
2	NIL	NIL	2	NIL	NIL
3	NIL	3	NIL	NIL	NIL
4	NIL	3	5	NIL	4
5	NIL	3	5	NIL	NIL

Tid $O(n)$

```
EXTEND-SHORTEST-PATHS(L, W)
1  n = L.rows
2  let L' = (l'_ij) be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          l'_ij = ∞
6          for k = 1 to n
7              l'_ij = min(l'_ij, l_ik + w_kj)
8  return L'
```



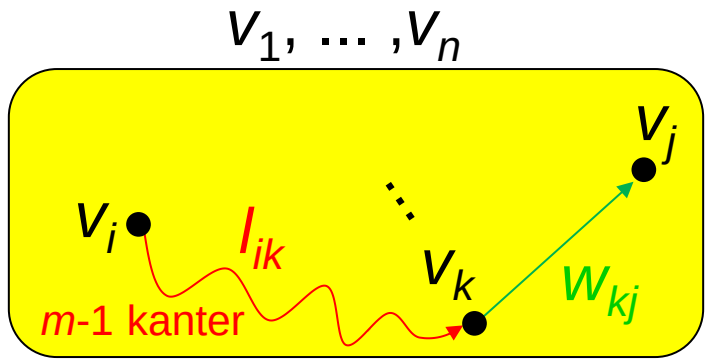
$$l'_{ij} = \min_k (l_{ik} + w_{kj})$$

L_{ij} = korteste afstand fra *i* til *j* for stier med Δ kanter
W = vægtmatrixen
L'_{ij} = korteste afstand fra *i* til *j* for stier med Δ+1 kanter

Tid $O(n^3)$

```
SQUARE-MATRIX-MULTIPLY(A, B)
1  n = A.rows
2  let C be a new n × n matrix
3  for i = 1 to n
4      for j = 1 to n
5          c_ij = 0
6          for k = 1 to n
7              c_ij = c_ij + a_ik · b_kj
8  return C
```

Tid $O(n^3)$

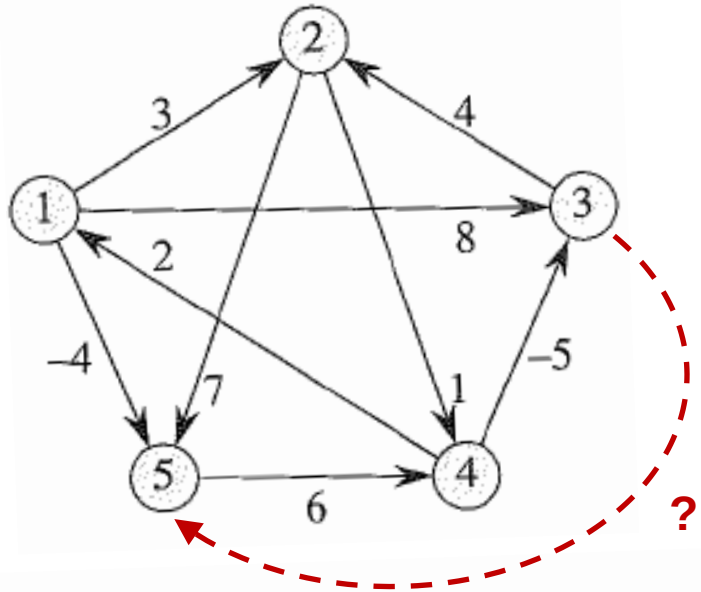



```
SLOW-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  for m = 2 to n − 1
4      let L^(m) be a new n × n matrix
5      L^(m) = EXTEND-SHORTEST-PATHS(L^(m−1), W)
6  return L^(n−1)
```

diagonalen = 0

$L^{(m)}_{ij}$ = korteste afstand fra i til j for stier med m kanter
W = vægtmatrixen

Tid $O(n^4)$

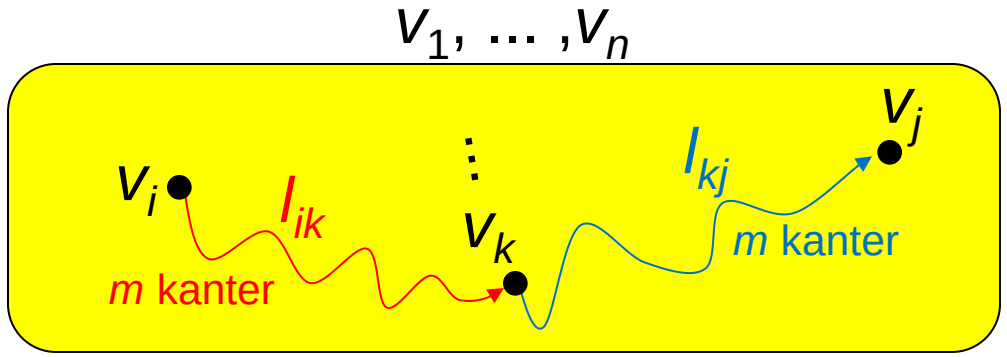


input graf

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

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

```
FASTER-ALL-PAIRS-SHORTEST-PATHS(W)
1  n = W.rows
2  L^(1) = W
3  m = 1
4  while m < n − 1
5      let L^(2m) be a new n × n matrix
6      L^(2m) = EXTEND-SHORTEST-PATHS(L^(m), L^(m))
7      m = 2m
8  return L^(m)
```



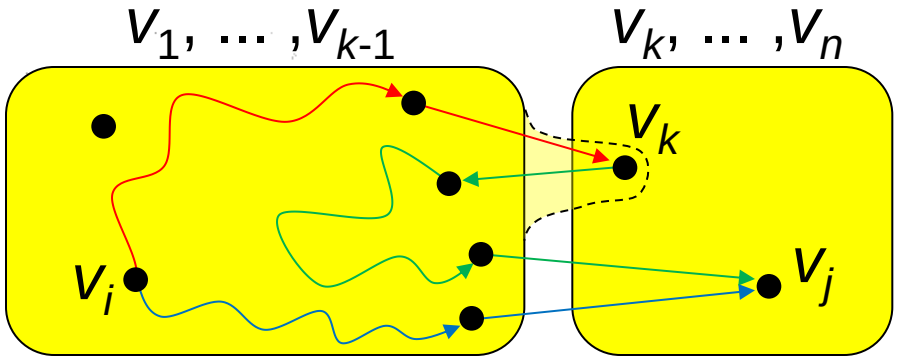
$$l_{ij}^{(2m)} = \min_k \left(l_{ik}^{(m)} + l_{kj}^{(m)}\right)$$

$L^{(m)}_{ij}$ = korteste afstand fra i til j for stier med m kanter
W = vægtmatrixen

Tid $O(n^3 \cdot \log n)$

Floyd-Warshall

```
FLOYD-WARSHALL(W)
1  n = W.rows
2  D^(0) = W
3  for k = 1 to n
4      let D^(k) = (d_ij^(k)) be a new n × n matrix
5      for i = 1 to n
6          for j = 1 to n
7              d_ij^(k) = min(d_ij^(k-1), d_ik^(k-1) + d_kj^(k-1))
8  return D^(n)
```



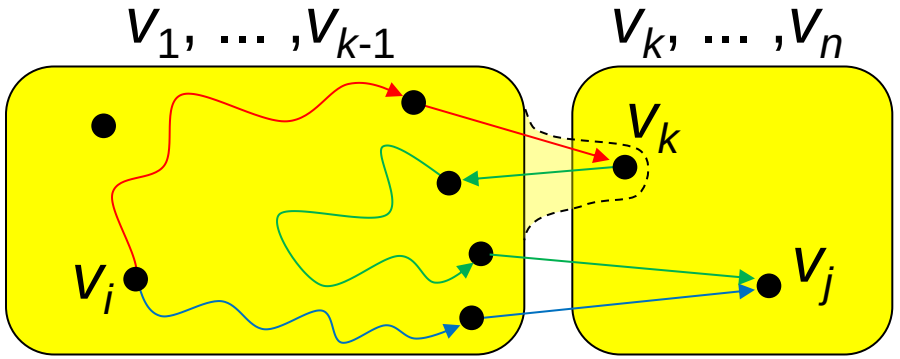
$d^{(k)}_{ij}$ = korteste vej fra *i* til *j* der kun går **via** 1..*k*

Tid $O(n^3)$

Transitive Lukning

(= Floyd-Warshall simplificeret)

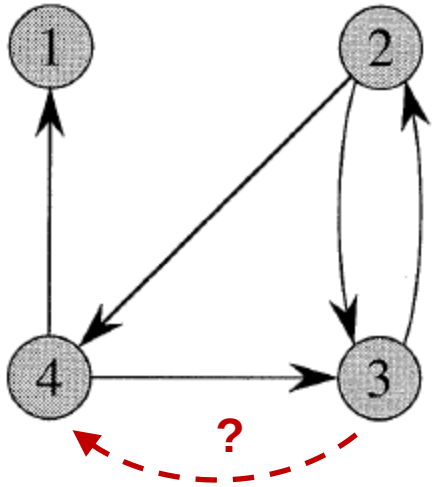
```
TRANSITIVE-CLOSURE(G)
 1  n = |G.V|
 2  let T^(0) = (t_ij^(0)) be a new n × n matrix
 3  for i = 1 to n
 4      for j = 1 to n
 5          if i == j or (i, j) ∈ G.E
 6              t_ij^(0) = 1
 7          else t_ij^(0) = 0
 8  for k = 1 to n
 9      let T^(k) = (t_ij^(k)) be a new n × n matrix
10      for i = 1 to n
11          for j = 1 to n
12              t_ij^(k) = t_ij^(k-1) ∨ (t_ik^(k-1) ∧ t_kj^(k-1))
13  return T^(n)
```



$t^{(k)}_{ij}$ = **findes en vej** fra *i* til *j* der kun går via 1..*k*

Tid $O(n^3)$

Transitive Lukning: Eksempel



input graf
+ selvløkker

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

$$T^{(3)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix} \quad T^{(4)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

Johnson's Algoritme

Korteste veje mellem alle par af knuder (APSP)

- Tynde grafer G, d.v.s. $m << n^2$,
- og postitive og negative vægtede kanter

Uden negative vægte ⇒ n x Dijkstra : $O(n \cdot (n \cdot \log n + m))$

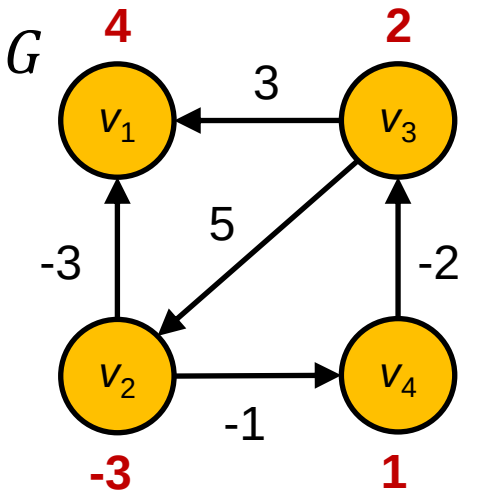
Med negative vægte ⇒ Floyd-Warshall : $O(n^3)$

Johnson's Algoritme

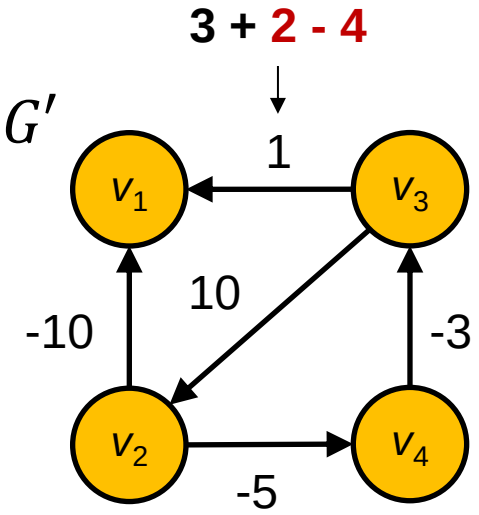
Ide: Kør Dijkstra på en graf G' uden negative kanter, som har de samme korteste veje som i G

Højdetransformation

tilknyt hver knude u en arbitrær højde $h(u)$



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



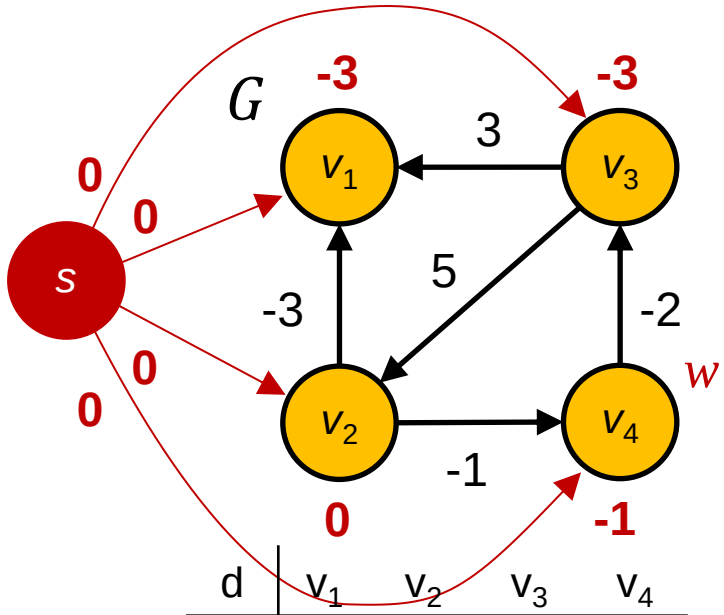
d	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

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

Korteste vej i G
⇔ korteste vej i G'

d'	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-10	0	-8	-5
v_3	0	10	0	5
v_4	-3	7	-3	0

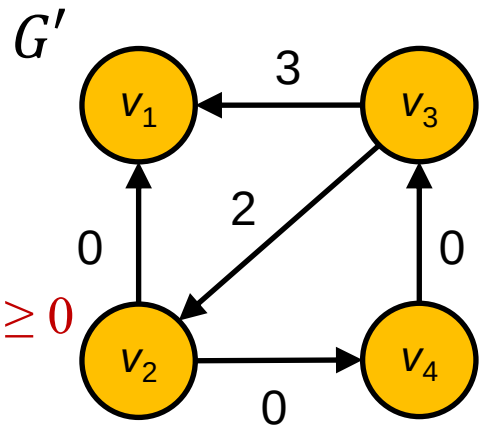
Johnson's Algoritme



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

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

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



d	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	-3	0	-3	-1
v_3	2	5	0	4
v_4	0	3	-2	0

d'	v_1	v_2	v_3	v_4
v_1	0	+∞	+∞	+∞
v_2	0	0	0	0
v_3	2	2	0	2
v_4	2	2	0	0

$3 = 2 + 0 - (-1)$

- Tilføj knude s og kanter fra s til alle knuder med vægt 0
- Lad $h(u) = d_G(s, u)$ – SSSP Bellman-Ford i tid $O(n \cdot m)$
- Kør Dijkstra fra hvert v_i i G' i tid $O(n \cdot (n \cdot \log n + m))$
- Lad $d_G(v_i, v_j) = d_{G'}(v_i, v_j) + h(v_j) - h(v_i)$

```
JOHNSON(G, w)
 1  compute G', where G'.V = G.V ∪ {s},
        G'.E = G.E ∪ {(s, v) : v ∈ G.V}, and
        w(s, v) = 0 for all v ∈ G.V
 2  if BELLMAN-FORD(G', w, s) == FALSE
 3      print "the input graph contains a negative-weight cycle"
 4  else for each vertex v ∈ G'.V
 5          set h(v) to the value of δ(s, v)
                computed by the Bellman-Ford algorithm
 6      for each edge (u, v) ∈ G'.E
 7          ŵ(u, v) = w(u, v) + h(u) − h(v)
 8      let D = (d_uv) be a new n × n matrix
 9      for each vertex u ∈ G.V
10          run DIJKSTRA(G, ŵ, u) to compute δ̂(u, v) for all v ∈ G.V
11          for each vertex v ∈ G.V
12              d_uv = δ̂(u, v) + h(v) − h(u)
13      return D
```

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

Korteste Veje

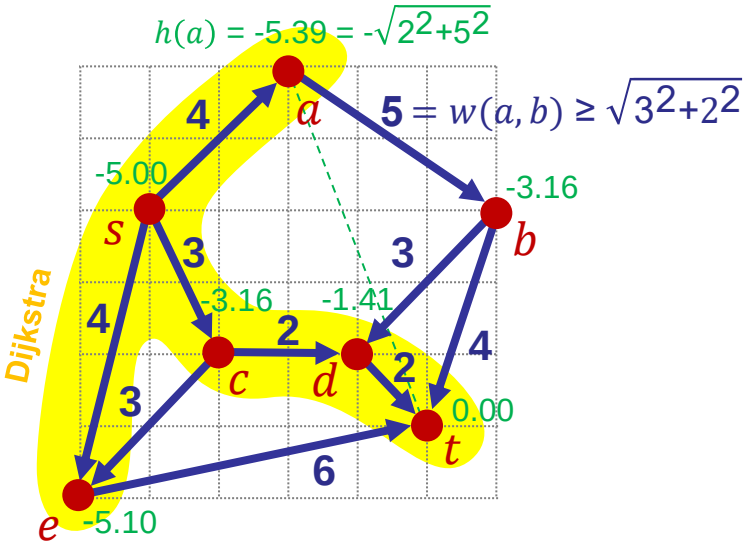
		En-til-alle korteste veje (SSSP)	Alle-til-alle korteste veje (APSP)
Acykliske grafer (positive og negative vægte)		$O(n+m)$	$O(n\cdot(n+m))$
Generelle grafer	Kun positive vægte	**Dijkstra** $O((n+m)\cdot\log n)$ (1) $O(n\cdot\log n + m)$ (2) $O(n^2)$ (3)	***n* x Dijkstra** $O(n^2\cdot\log n + n\cdot m)$ (2) $O(n^3)$ (3)
	Positive og negative vægte	**Bellman-Ford** $O(n\cdot m)$	**Floyd-Warshall** $O(n^3)$ **Johnson** $O(n^2\cdot\log n + n\cdot m)$ (2)

Forskellige prioritetskøer: (1) binær heap, (2) Fibonacci heap, (3) array

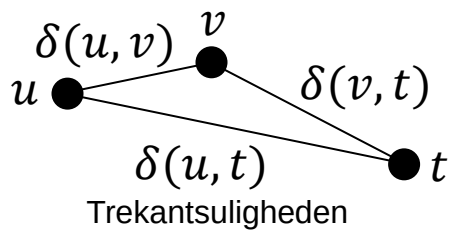
IKKE PENSUM

A*-algoritmen : korteste vej $s \rightsquigarrow t$

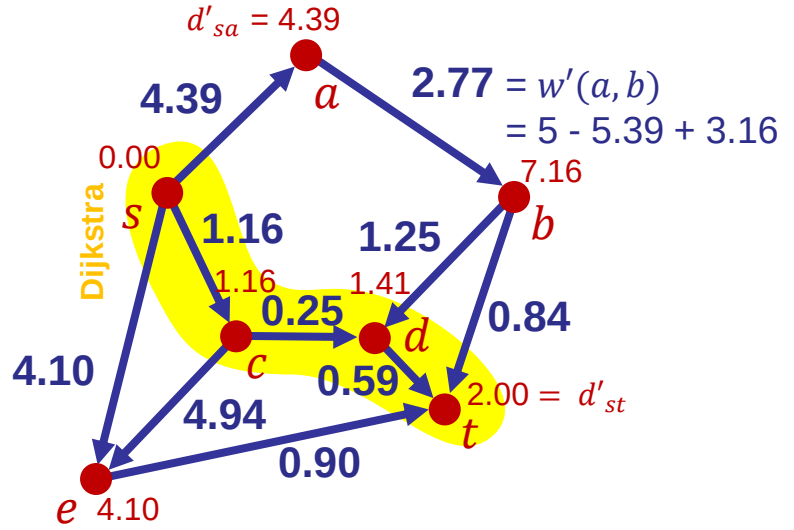
- **Antag** knude u har position (u_x, u_y) og $w(u,v) \geq \delta(u,v) = \sqrt{(v_x - u_x)^2 + (v_y - u_y)^2}$
- $\boxed{h(u) = -\delta(u,t)} \Rightarrow w'(u,v) = w(u,v) + h(u) - h(v) \geq \delta(u,v) - \delta(u,t) + \delta(v,t) \geq 0$
- $d_{st} = d'_{st} - h(s) + h(t) = 2.00 + 5.00 - 0.00 = \underline{\mathbf{7.00}}$



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



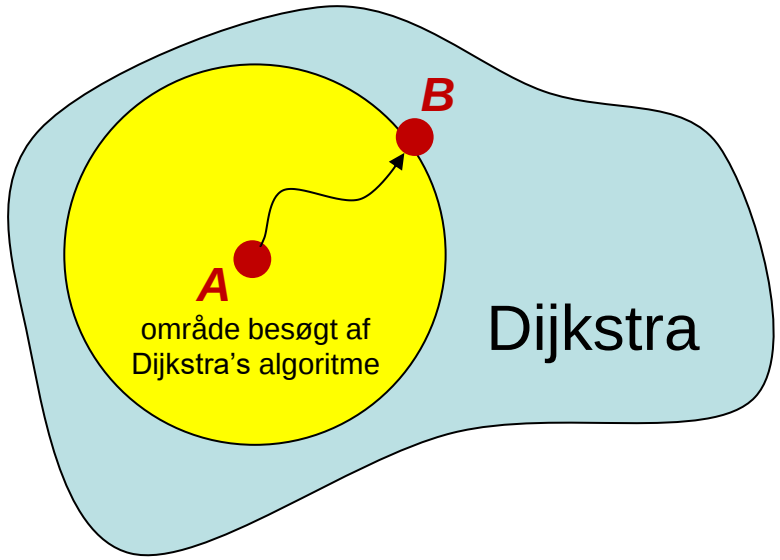
Trekantsuligheden

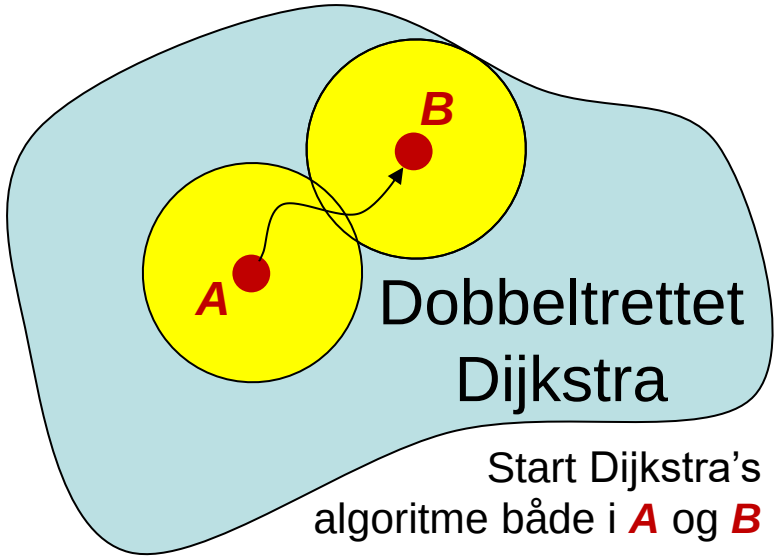


A* reducerer i praksis antallet af knuder Dijkstra's algoritme besøger

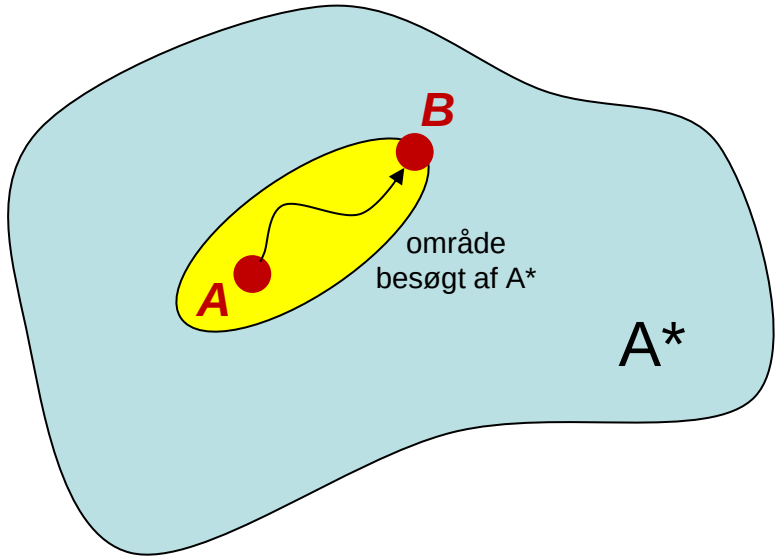
Youtube: Compare A* with Dijkstra algorithm

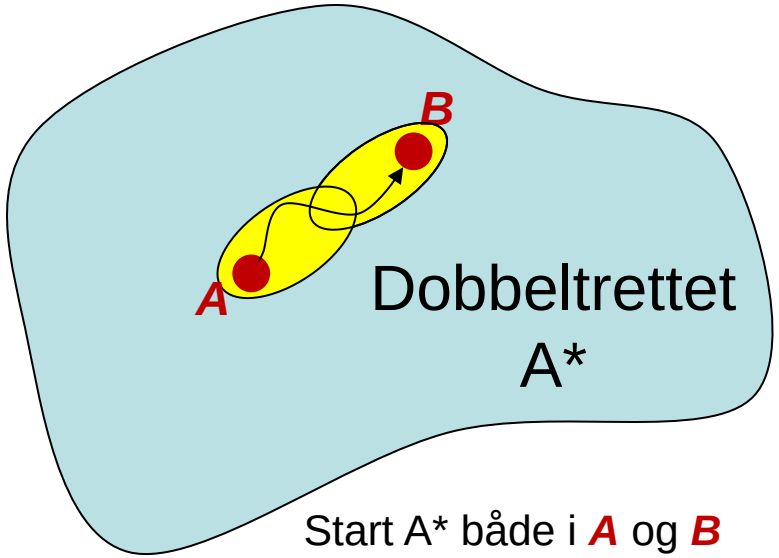
IKKE PENSUM





Start Dijkstra's
algoritme både i ***A*** og ***B***





Start A* både i ***A*** og ***B***

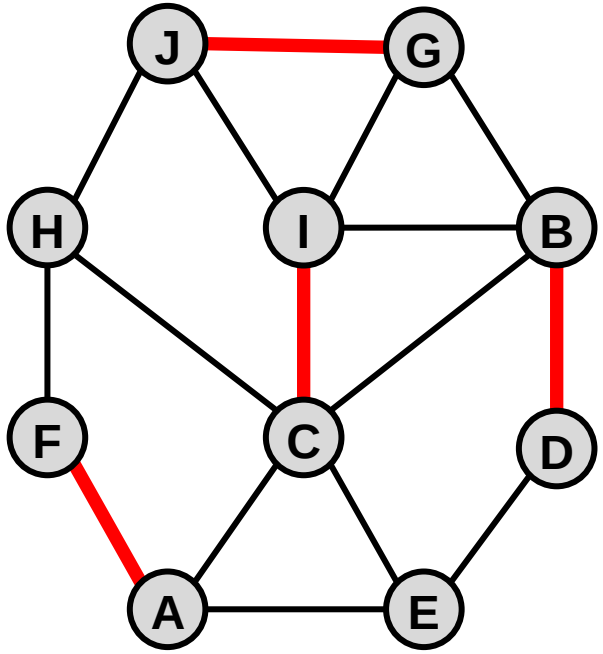
Intuitivt: Start Dijkstra's algoritme både i ***A*** og ***B***

IKKE PENSUM

Algoritmer og Datastrukturer

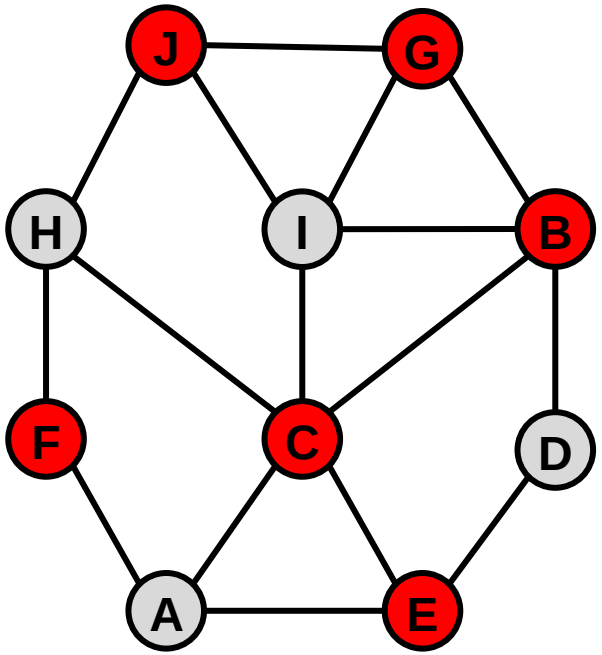
Parringer i grafer
Maximum uafhængige mængde
Minimum knudedækning
[CLRS, kapitel 24.3, 25.2, 34.4, 35.1]

Parring (Matching)



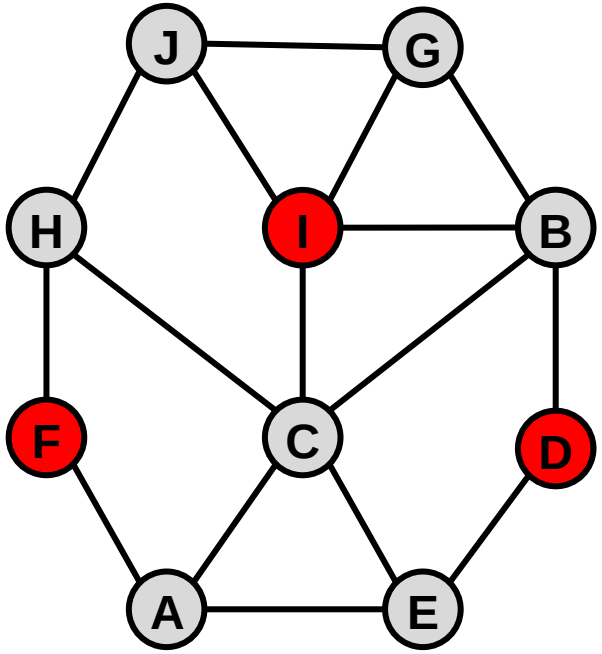
Delmængde af kanterne, hvor hver knude højst indgår i én kant i parringen

Knudedækning (Vertex cover)



Delmængde af knuderne, så alle kanter i grafen indeholder mindst en af disse

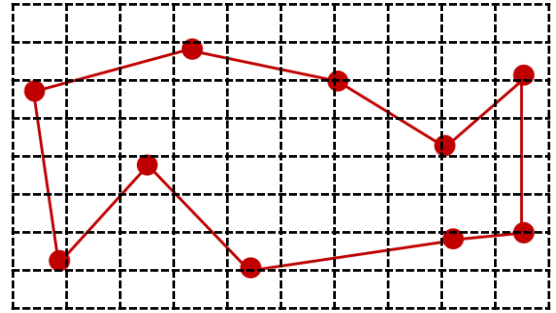
Uafhængig mængde (Independent set)



Delmængde af knuderne, hvor ingen par af knuder er forbundet med en kant i grafen

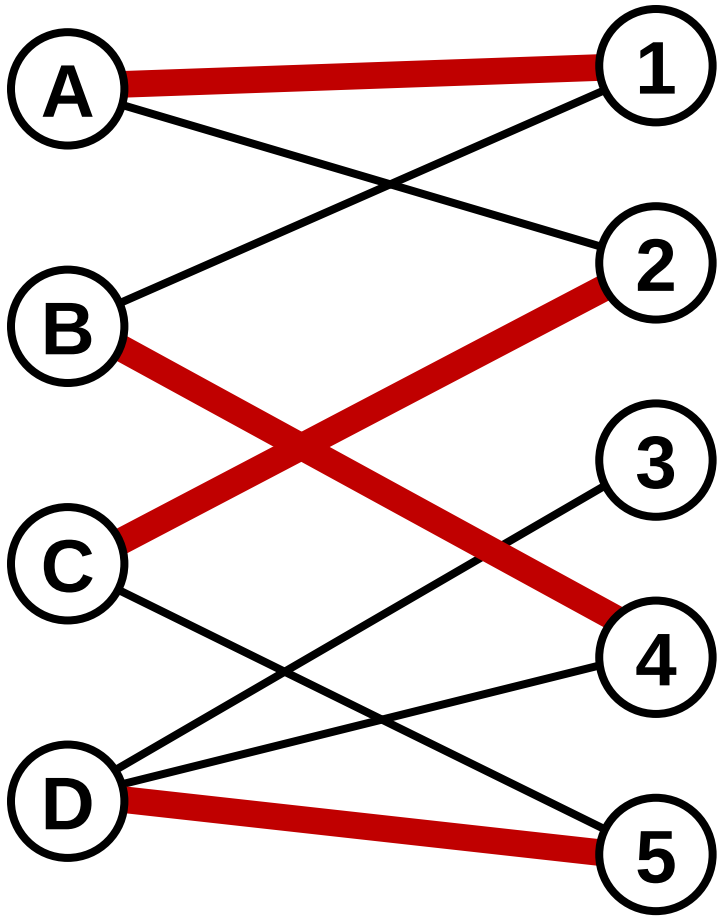
Ikke alle grafproblemer er lige lette…

- Nogle problemer kan løses effektivt i polynomial tid
  - **DFS**, **BFS**, **korteste veje**, **topologisk sortering**, **parringer**…
- Mange problemer er dog NP-hårde
  - Eksakte løsninger kræver sandsynligvis eksponentiel tid
  - Nogle kan approksimeres vilkårligt godt (PTAS = ”polynomial time approximation scheme”)
    - f.eks. **Euklidisk Traveling Salesman** (Gödel Prisen 2010)



  - Nogle kan kun approksimeres op til en konstant (APX-hårde)
    - f.eks. **Mindste knudedækning (Minimum Vertex Cover)**
  - Nogle kan slet ikke approksimeres inden for en konstant
    - f.eks. **Største uafhængige mængde (Maximum Independent Set)**

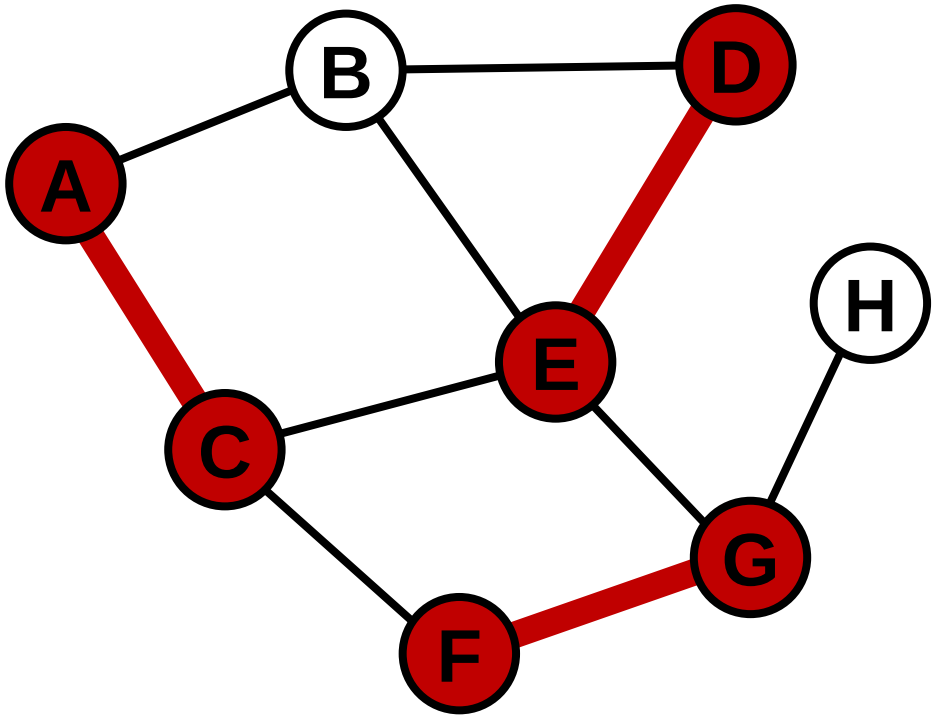
Parringer i todelte grafer (specialtilfælde)



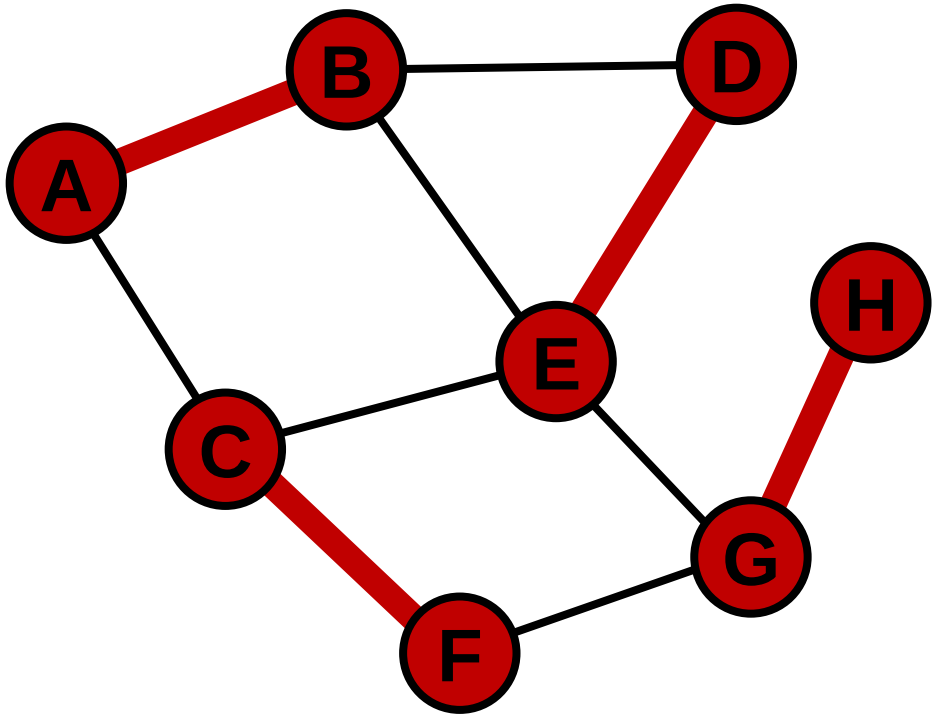
Eksempler

- Busruter vs Busser
  (kant angiver om det er muligt pga kapacitet, rækkevidde, elektrisk vs ikke-elektrisk)
- Personer vs Arbejdsopgaver
  (kant angiver om en given person ville være i stand til at udføre opgaven)
  Findes en matching hvor alle personer arbejder?
- Ansøgninger til jobs
  (fx studerende og sygehuse)

Maksimal og maksimum parring



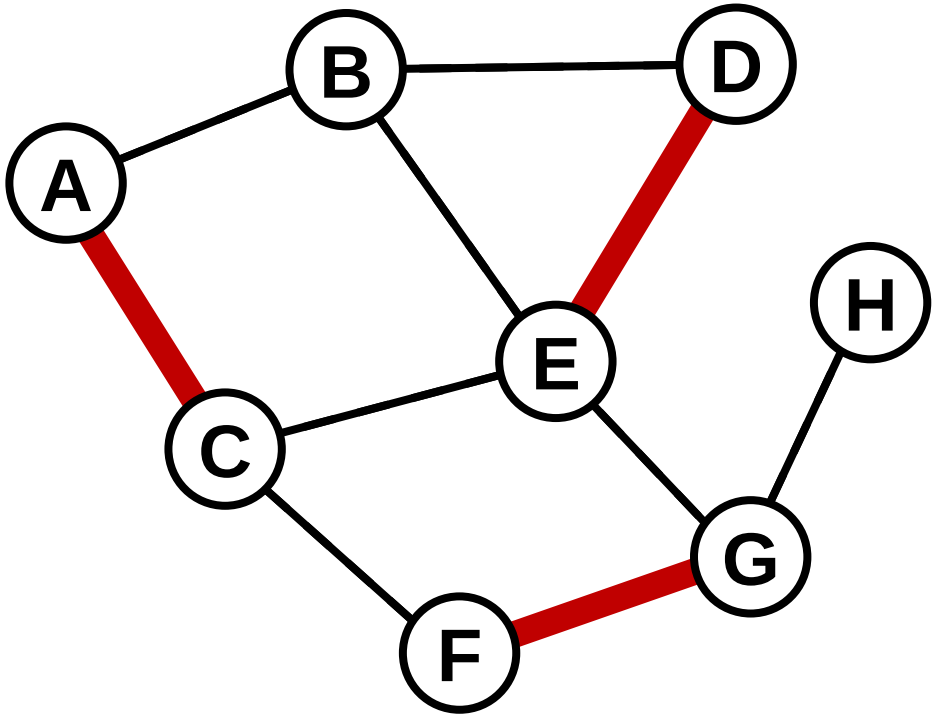
Maksimal
kan ikke udvides



Maksimum
størst mulig parring

Lemma |Maksimum parring| / 2 ≤ |Maksimal parring| ≤ |Maksimum parring|

Konstruktion af en maksimal parring

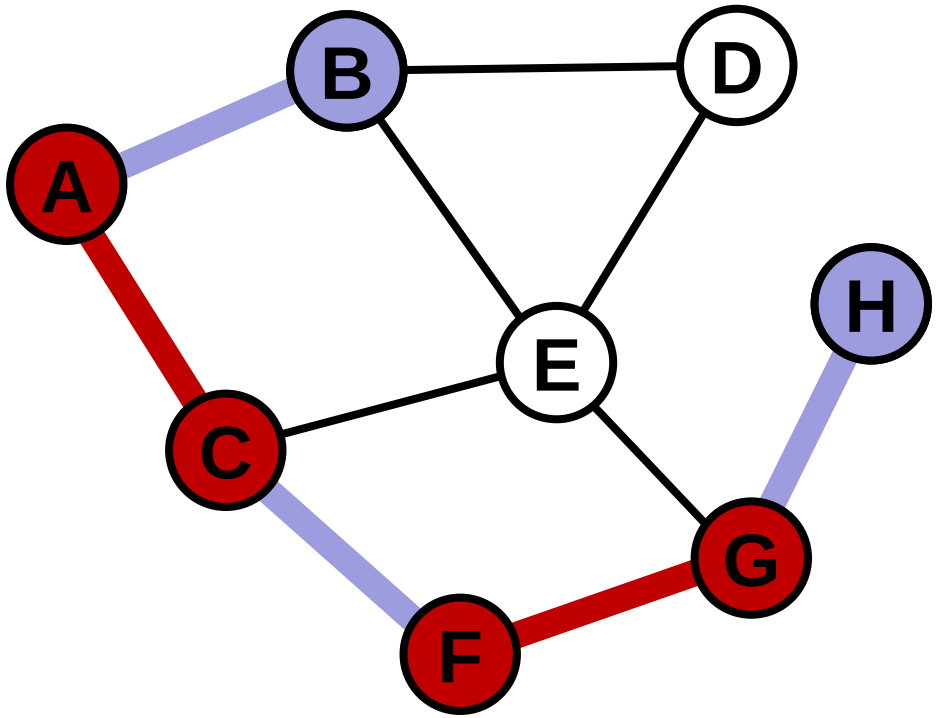


Grådig Algoritme

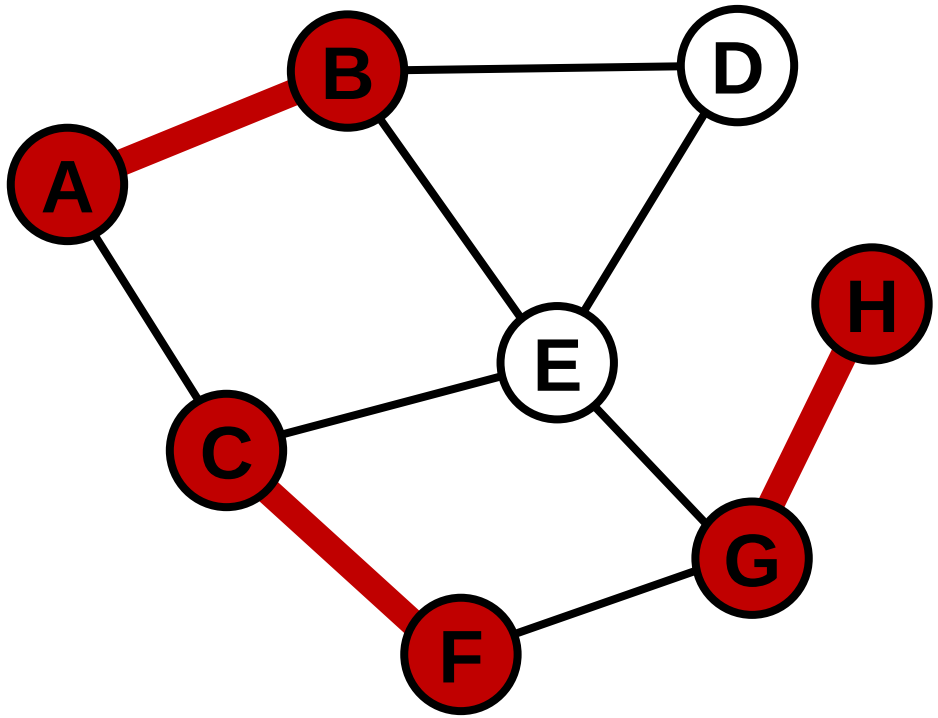
Tilføj en vilkårlig kant til parringen så længe det er muligt

$O(n + m)$

Konstruktion af en maksimum parring – forbedrende stier



Sti mellem to knuder ikke i parring,
hver anden kant i parring

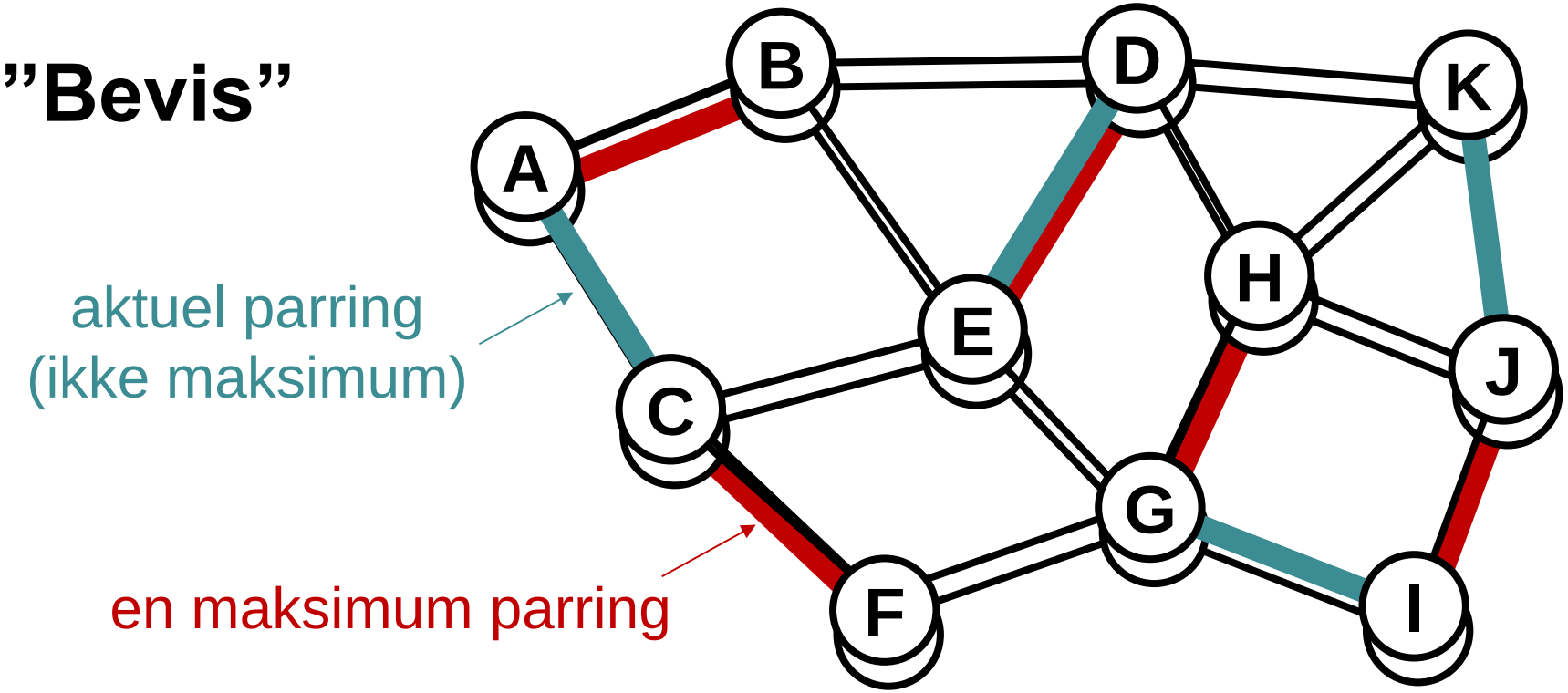


Parring øges med 1

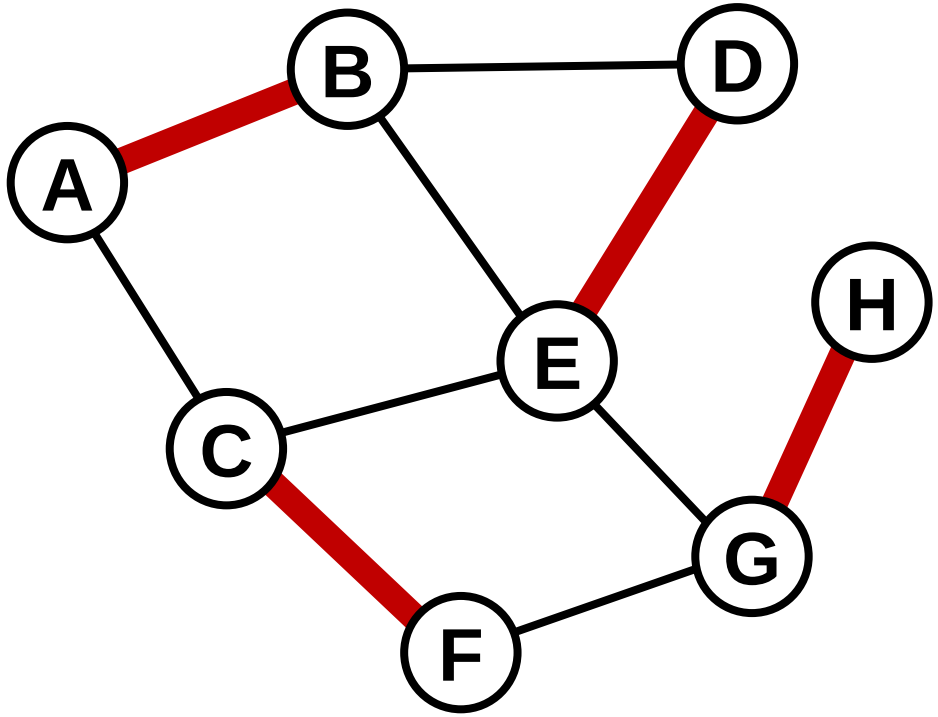
Petersen 1891 / Kőnig 1931 / Berge 1957

Sætning
En parring er maksimum ⇔ findes ingen forbedrende stier

”Bevis”



Konstruktion af en maksimum parring

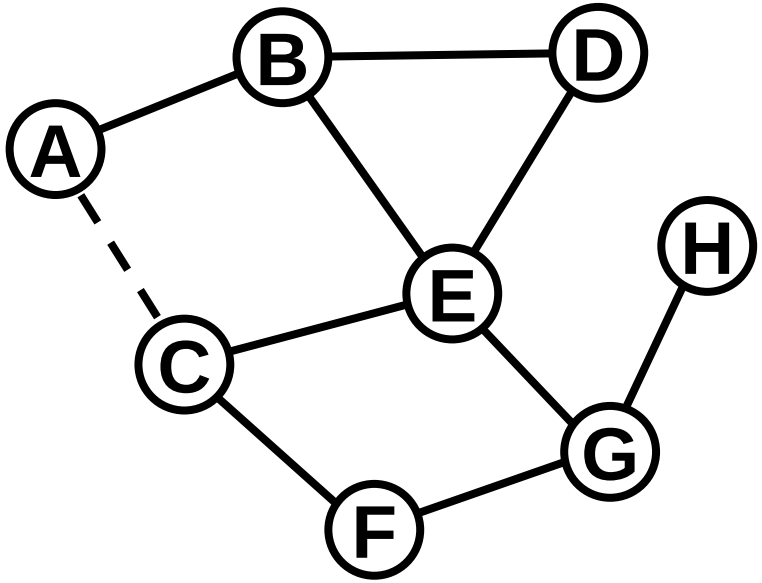


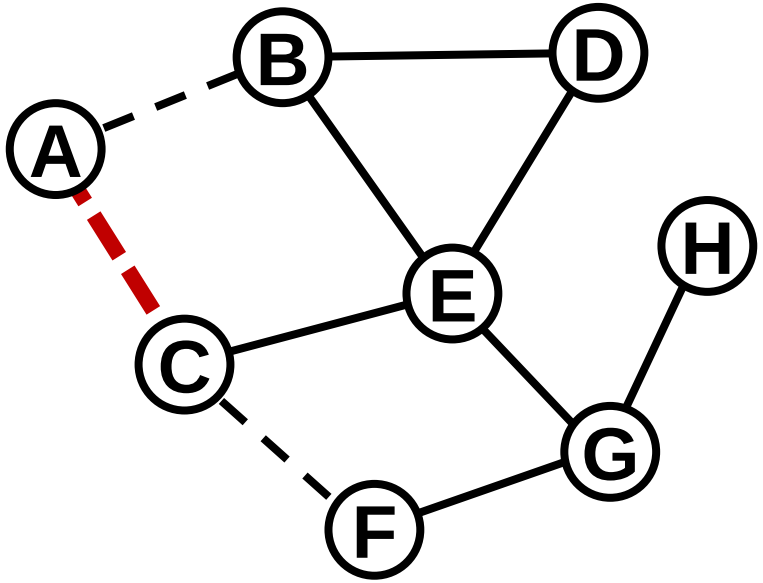
Algoritme

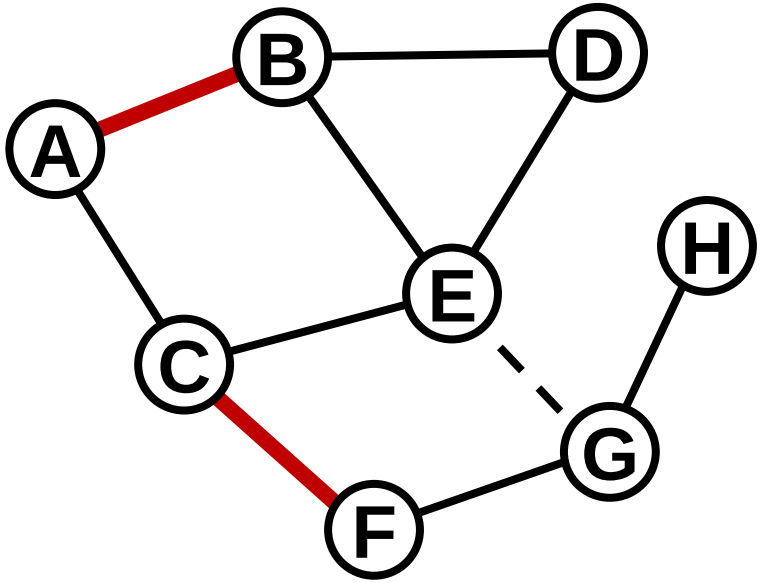
Anvend forbedrende stier så længe de findes

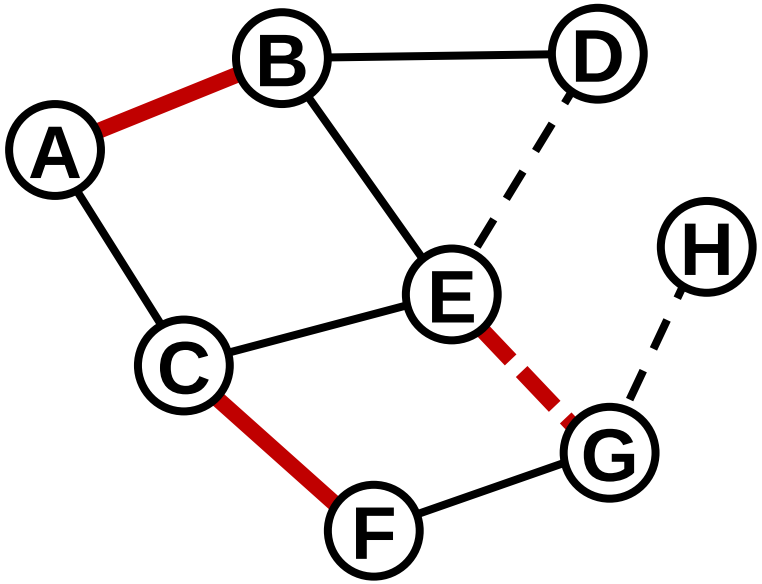
Edmonds 1961 $O(m \cdot n^2)$

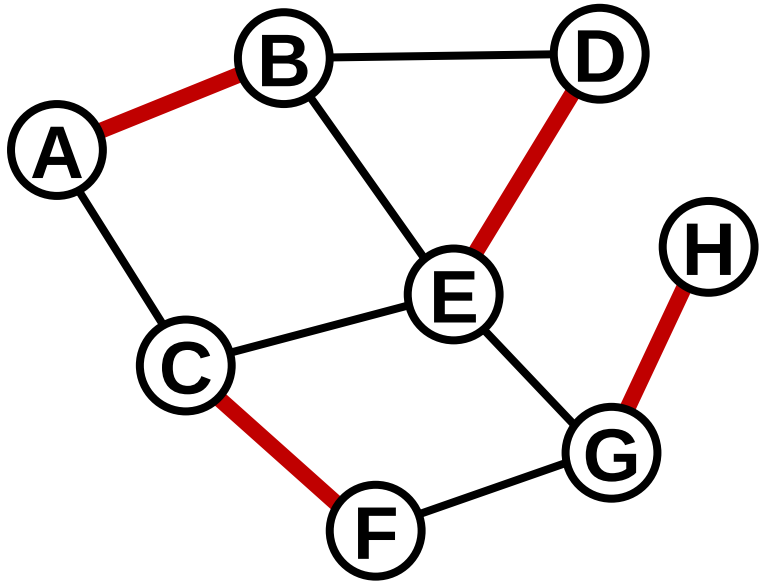
Micali, Vazirani 1980 $O(m \cdot n^{1/2})$


A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H

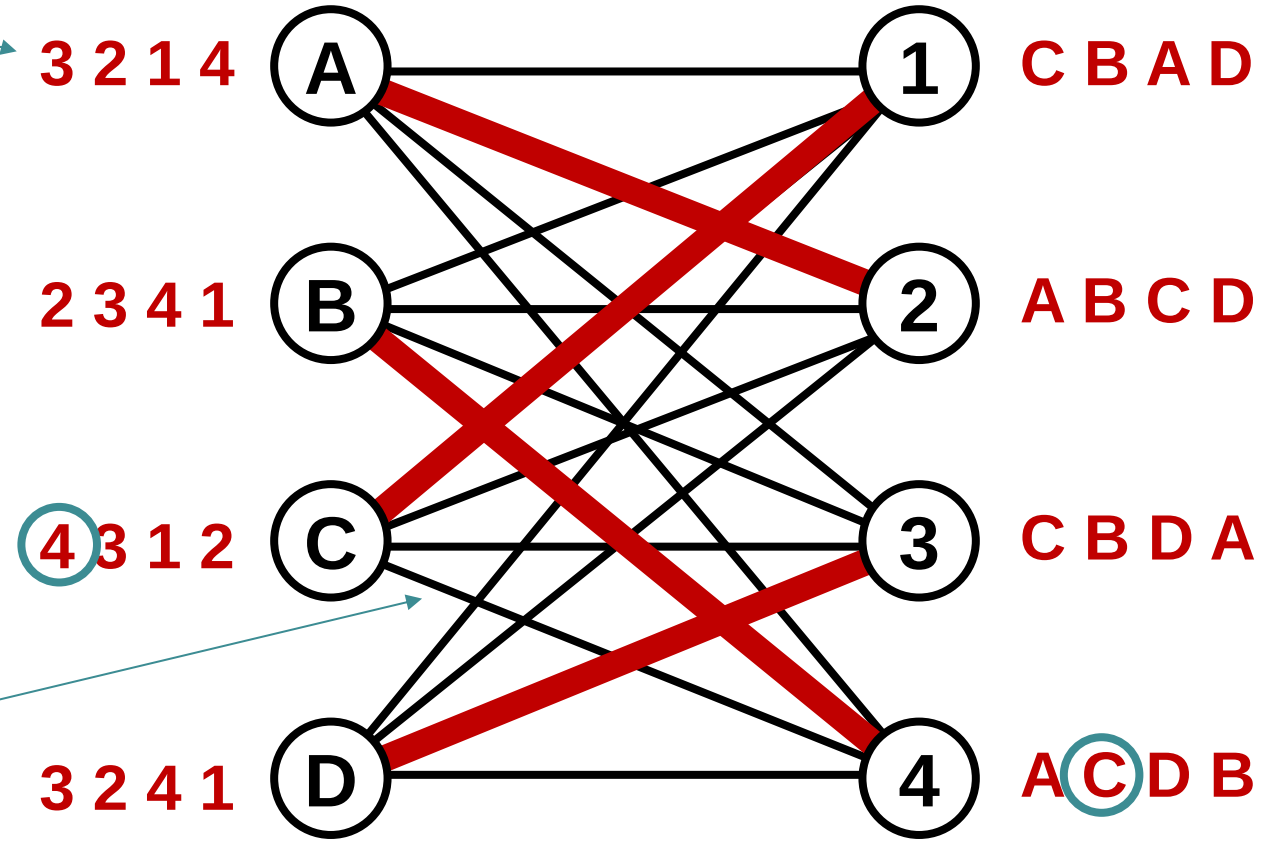

A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H


A
B
C
D
E
F
G
H

Stabil parring (stable marriage problem)

præference



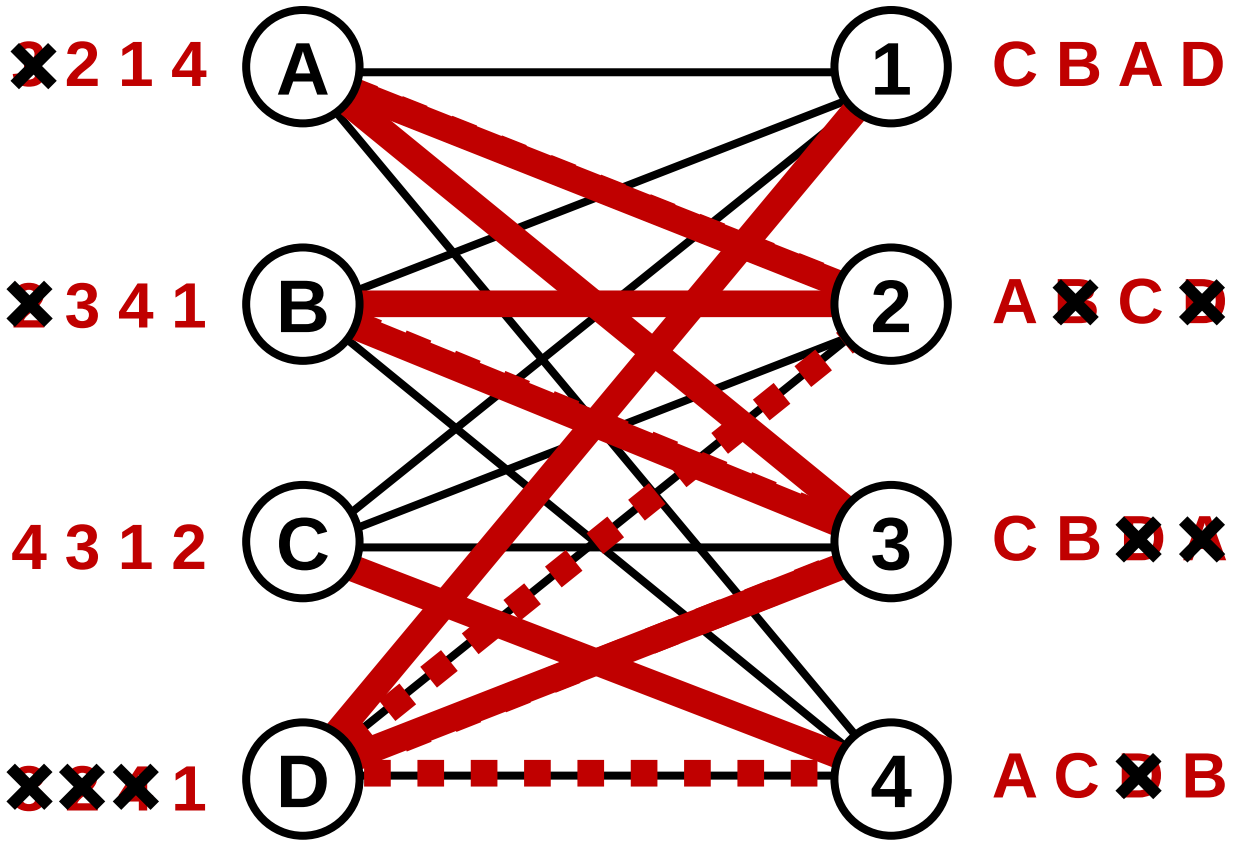
parringen er ikke stabil, da C hellere vil være sammen med 4 (end 1), og 4 heller med C (end B)

fuldstændig todelt graf med lige mange knuder på begge sider

CLRS 25.2

Stabil parring – Grådig algoritme

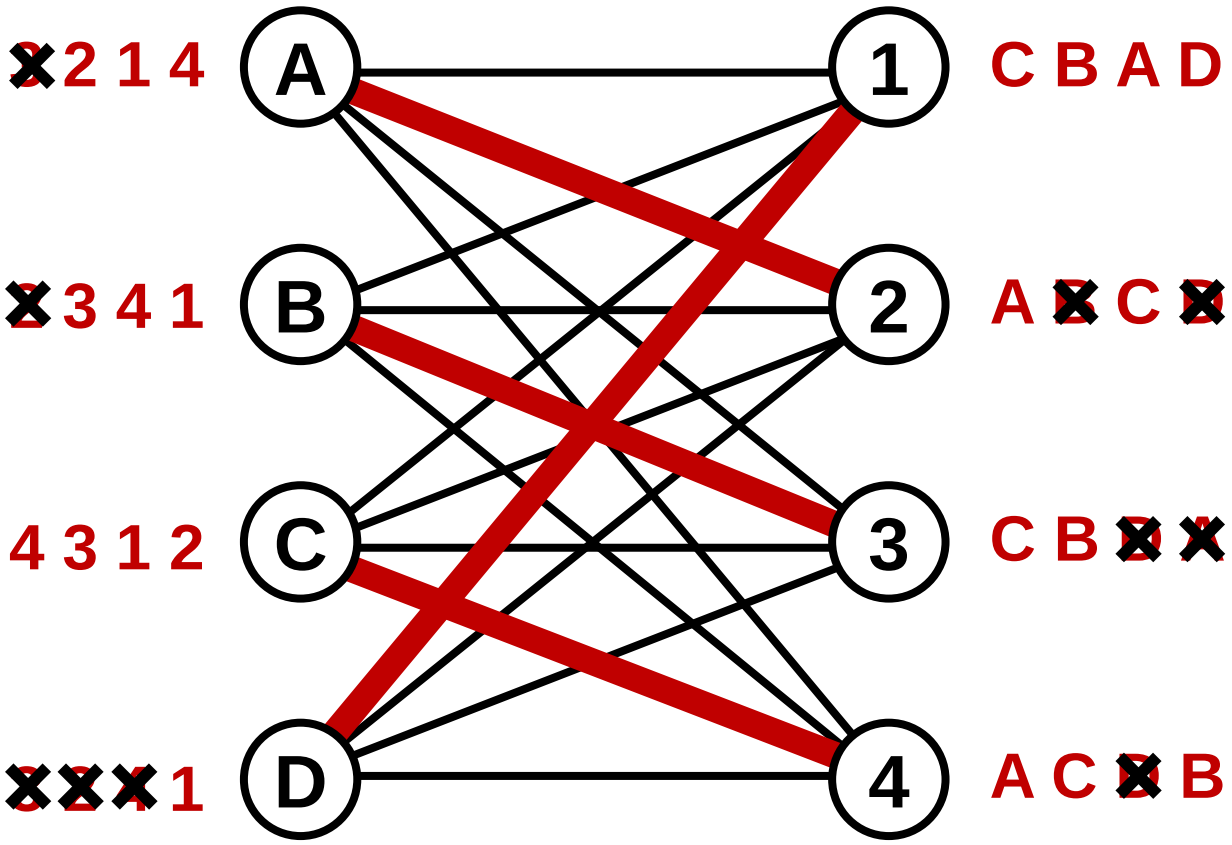
$O(n^2)$



Hvis venstre side *X* er afvist af højre side *i*, så er *i* parret med en som foretrækkes frem for *X*
X er parret med den mest foretrukne, som endnu ikke har afvist *X*

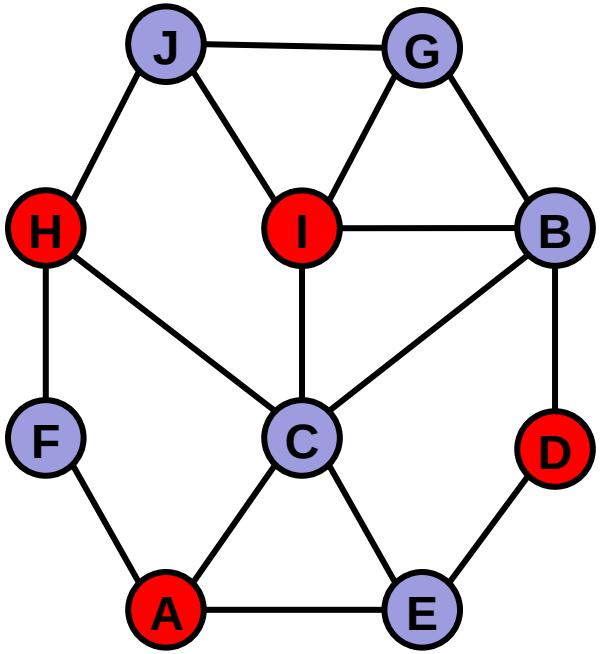
Sætning (Gale, Shapley 1962)

Der findes altid en stabil parring



(Shapley fik i 2012 Nobelprisen i økonomi for resultatet)

Uafhængig mængde vs Knudedækning
(Independent set vs Vertex cover)



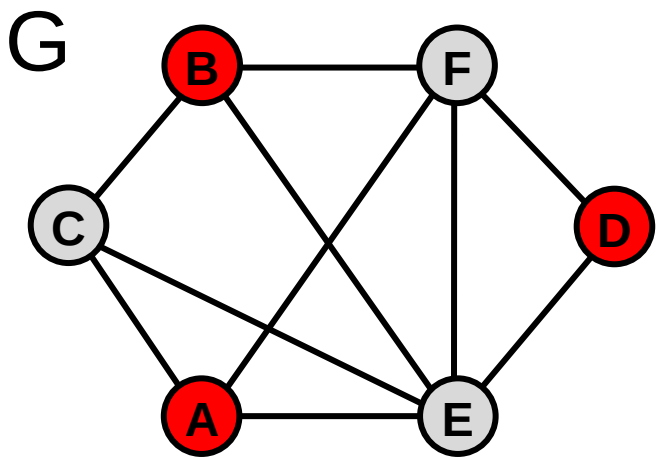
Lemma

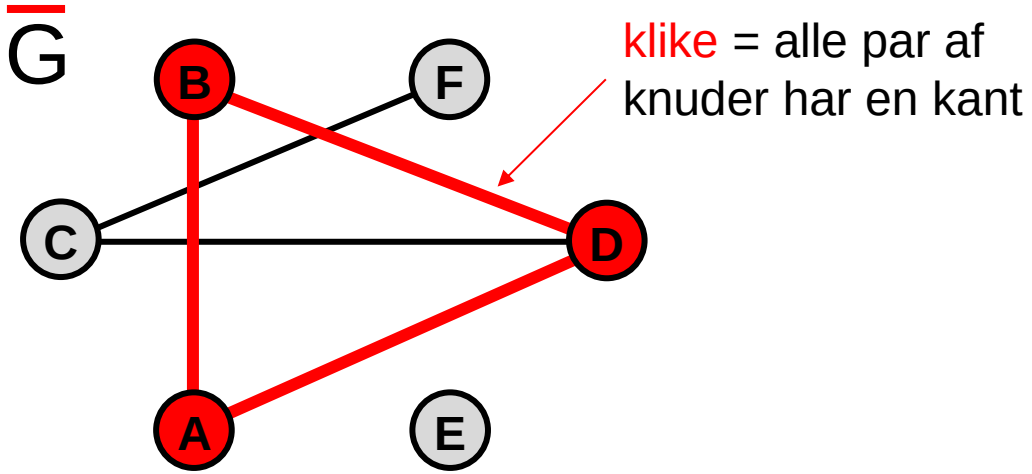
I ⊆ *V* uafhængig mængde ⇔ *V* \ *I* dækning af kanterne

Sætning

I ⊆ *V* maximum uafhængig mængde ⇔ *V* \ *I* minimum knudedækning

Uafhængig mængde vs Klike
(Independent set vs Clique)

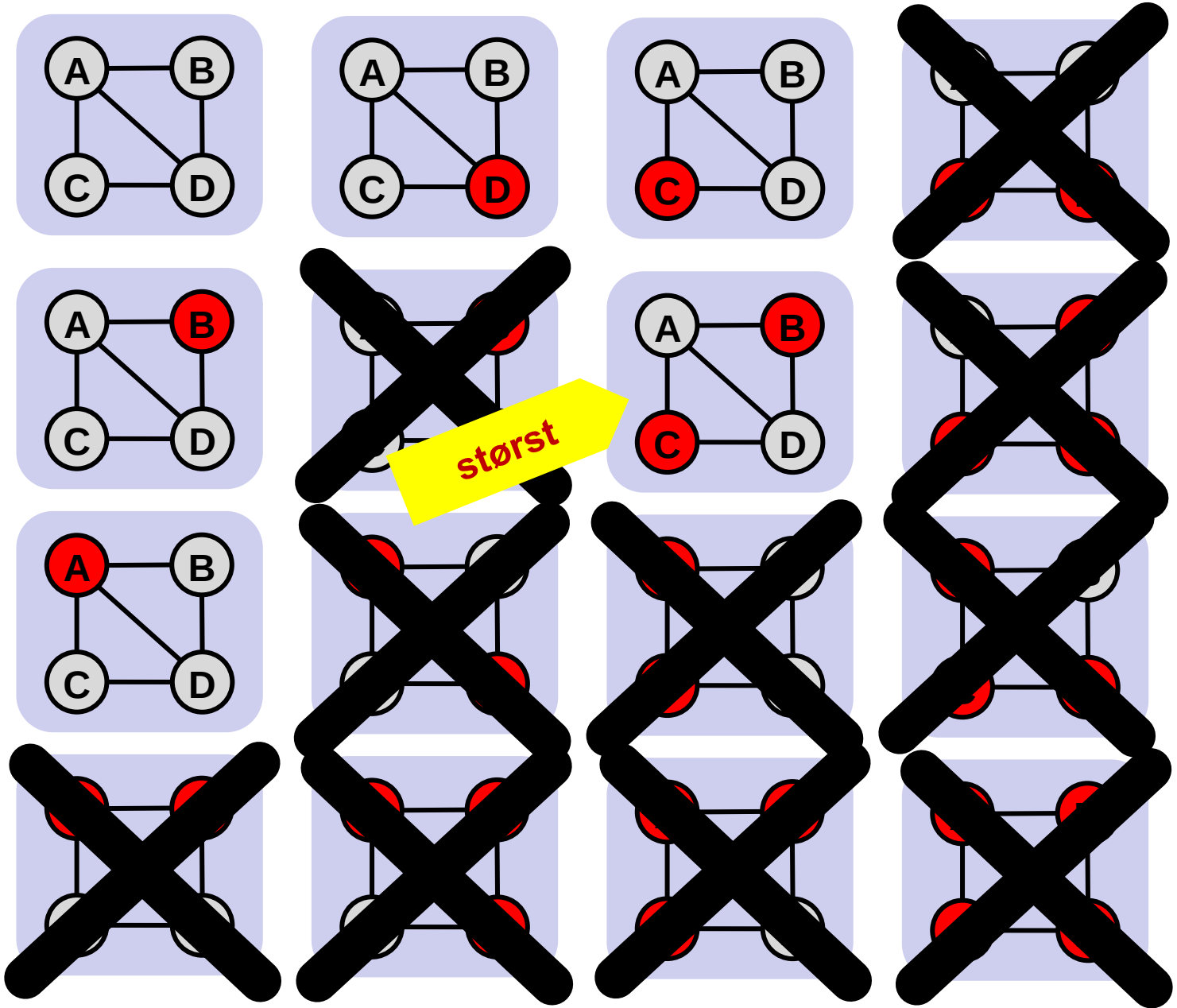




Sætning

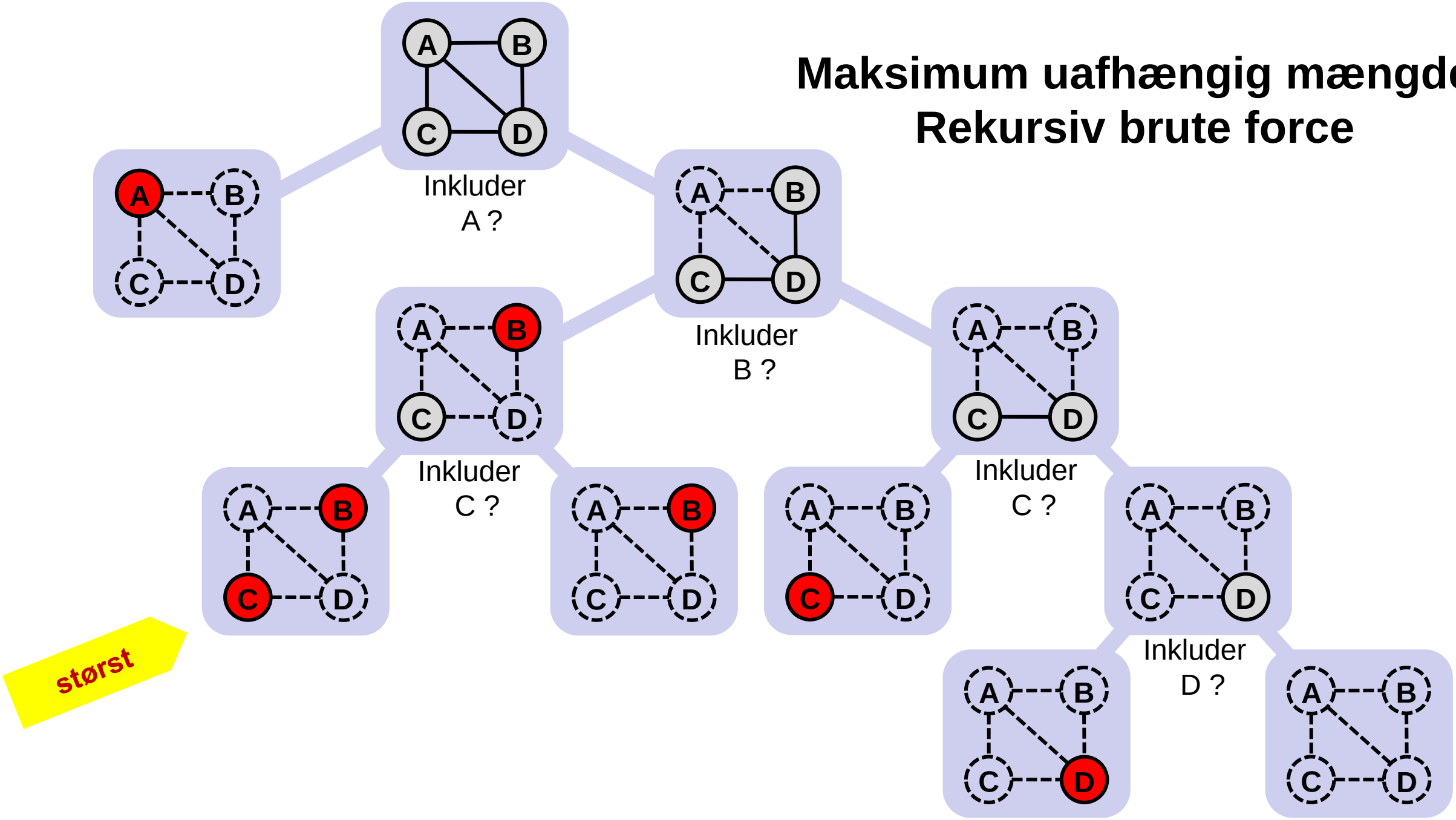
I ⊆ *V* maximum uafhængig mængde i G ⇔ *I* maximum klike i $\overline{G}$

Maksimum uafhængig mængde – prøv alle 2^n delmængder



$O(m \cdot 2^n)$

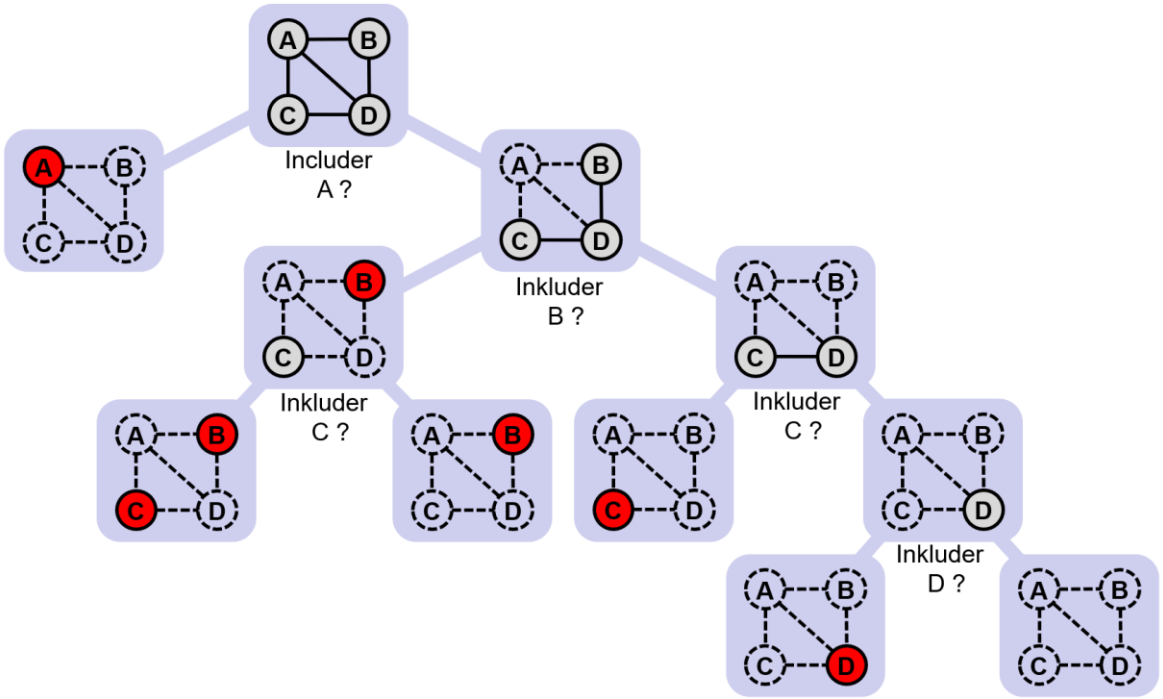
Maksimum uafhængig mængde
Rekursiv brute force



Maksimum uafhængig mængde

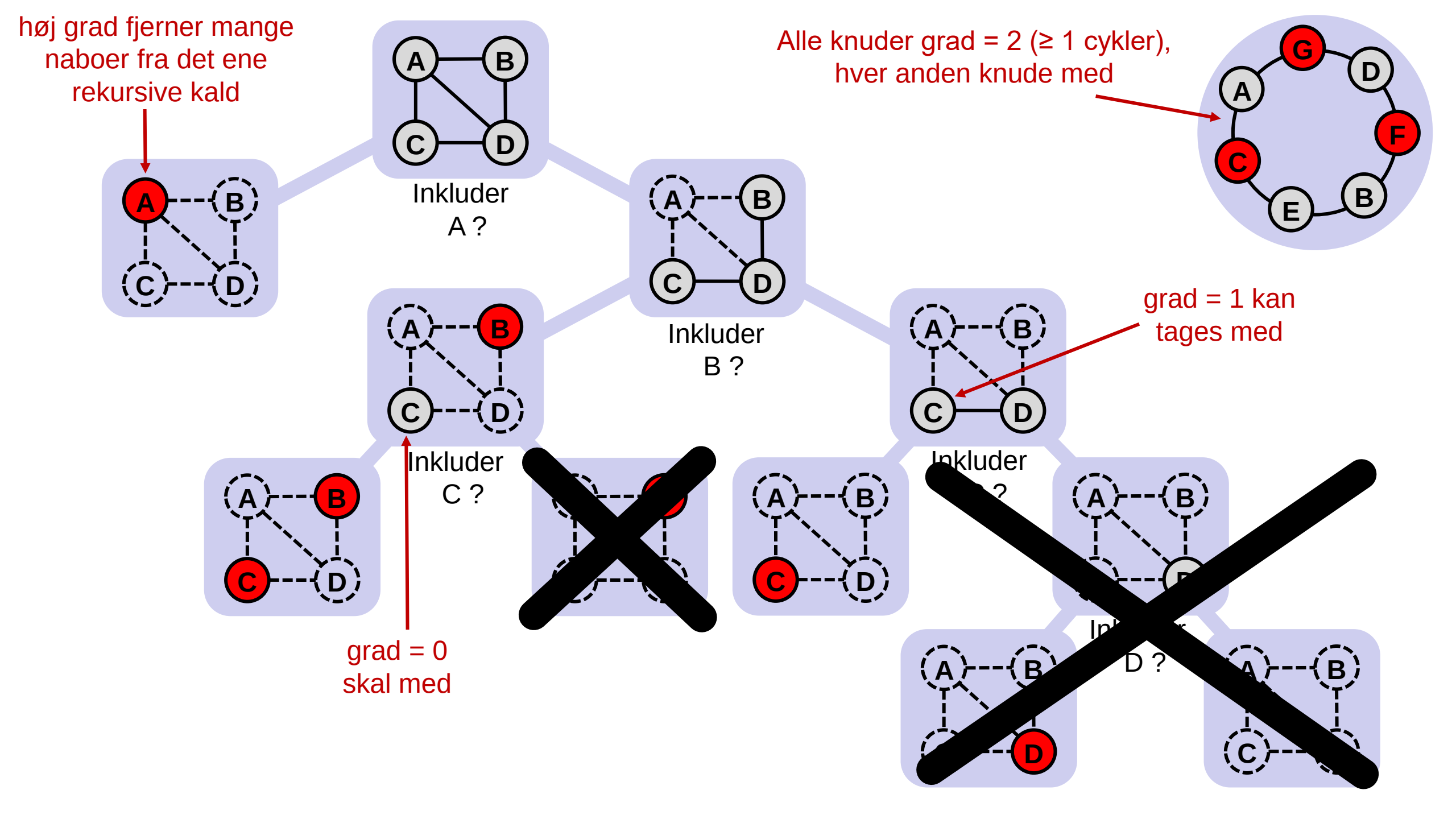
Algorithm $\text{MIS1}(G)$

```
1  if V_G = ∅ then
2      return ∅
3  let v ∈ V_G be an arbitrary vertex
4  I = MIS1(G \ {v})
5  I' = {v} ∪ MIS1(G \ {v} \ adj_G(v))
6  if |I'| > |I| then
7      I = I'
8  return I
```



Analyse

Antal rekursive kald $T(n) \leq \begin{cases} 1 & \text{hvis } n = 0 \\ 1 + 2 \cdot T(n-1) & \text{ellers} \end{cases} \Rightarrow T(n) \leq 2^{n+1} - 1$


høj grad fjerner mange naboer fra det ene rekursive kald
Alle knuder grad = 2 (≥ 1 cykler), hver anden knude med
grad = 1 kan tages med
Inkluder A ?
Inkluder B ?
Inkluder C ?
grad = 0 skal med
D ?

Algorithm MIS2 (G)

```
1   I = ∅
2   while ∃ v ∈ V_G : |adj_G(v)| ≤ 1 do
3       I = I ∪ {v}
4       G = G \ {v} \ adj_G(v)
5   # all vertices have degree ≥ 2
6   if ∃ v ∈ V_G : |adj_G(v)| ≥ 3 then
7       I_1 = I ∪ MIS2(G \ {v})
8       I_2 = I ∪ {v} ∪ MIS2(G \ {v} \ adj_G(v))
9       return I_1 if |I_1| ≥ |I_2| otherwise I_2
10  # all remaining vertices have degree 2
11  while ∃ cycle C = (c_1, c_2, ..., c_k) in G do
12      I = I ∪ {c_1, c_3, c_5, ..., c_{2⌊k/2⌋}}
13      G = G \ C
14  return I
```

grådigt inkluder knuder med grad 0 og 1

som MIS1, dog fjernes mindst 4 knuder i andet kald

grådigt inkluder knuder på cykler

Analyse

Antal rekursive kald $T(n) \leq \begin{cases} 1 & \text{hvis } n = 0 \\ 1 + T(n-1) + T(n-4) & \text{ellers} \end{cases} \Rightarrow T(n) = O(1.38028^n)$

Reduktion : 3-SAT til uafhængig mængde

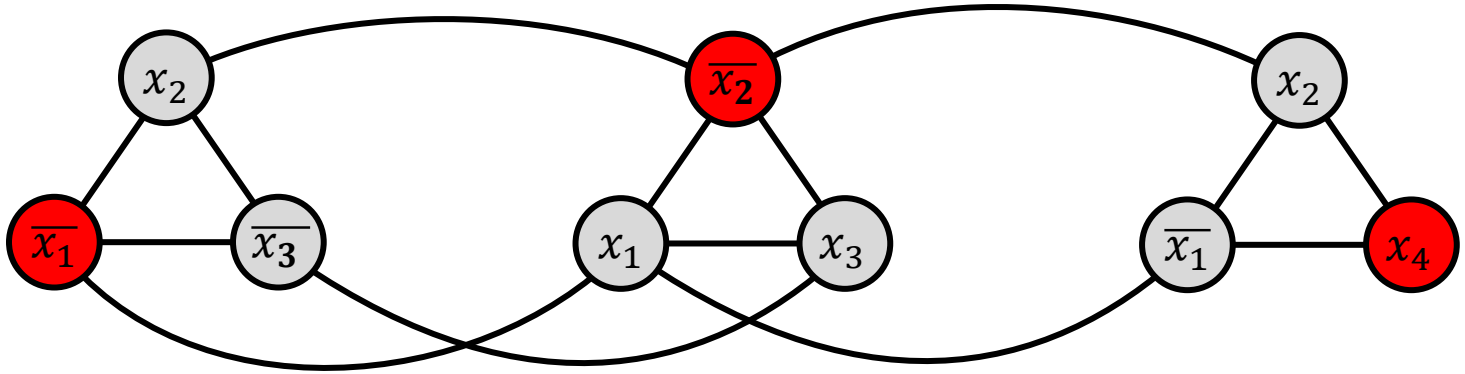
- **3-SAT** Givet en boolsk formel på konjunktiv normal form (CNF), hvor alle *n* klausuler har tre literaler, f.eks.

$$\varphi = (\overline{x_1} \vee x_2 \vee \overline{x_3}) \wedge (x_1 \vee \overline{x_2} \vee x_3) \wedge (\overline{x_1} \vee x_2 \vee x_4)$$

- Findes en tildeling af sand/falsk til literalerne $x_1, x_2, \ldots$ så formelen er sand?

$$x_1 = x_2 = \text{falsk}, \qquad x_3 = x_4 = \text{sand}$$

- **Reduktion** Lav en trekant per klausul, en knude per literal. Tilføj alle kanter $(\overline{x_i}, x_i)$.



- **Sætning** 3-SAT formel kan tilfredsstilles ⇔ maximum uafhængig mængde størrelse *n*

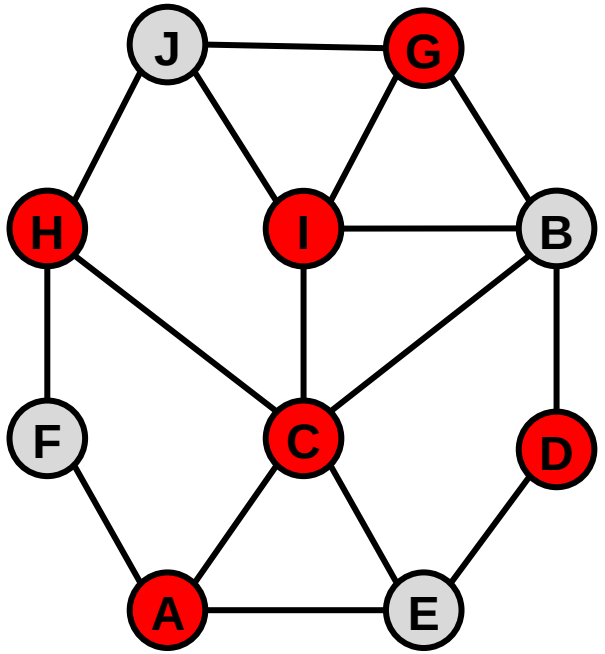
NP-hårdhed

- 3-SAT er et såkaldt NP-hårdt problem, hvilket medfører at det er tvivlsomt om der findes bedre end eksponentieltids algoritmer

- Reduktionen medfører at følgende også er NP-hårde
  - Maksimum uafhængig mængde
  - Maksimum klike (= uafhængig mængde i negerede graf)
  - Minimum knudedækning (= negerede mængde til uafhængig mængde)

- Uafhængig mængde kan løses i tid $O(1.1996^n)$
  [Xiao & Nagamochi, 2017]

2-approksimering af minimum knudedækning



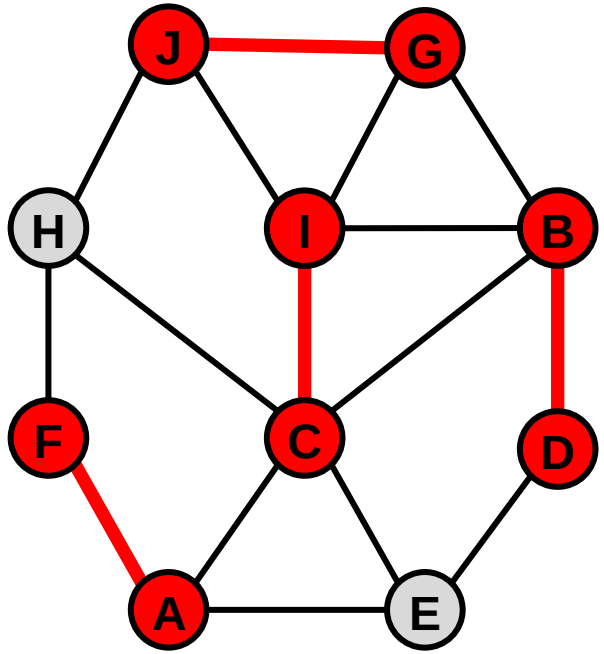
Minimum knudedækning $C_{optimal}$

Algoritme

- Beregn en maksimal parring M
- Knudedækning C = alle knuder i M

Analyse

- C er højst dobbelt så stor som en minimum knudedækning $C_{optimal}$, da alle kanter i M må have en knude i $C_{optimal}$
- Tid $O(n + m)$



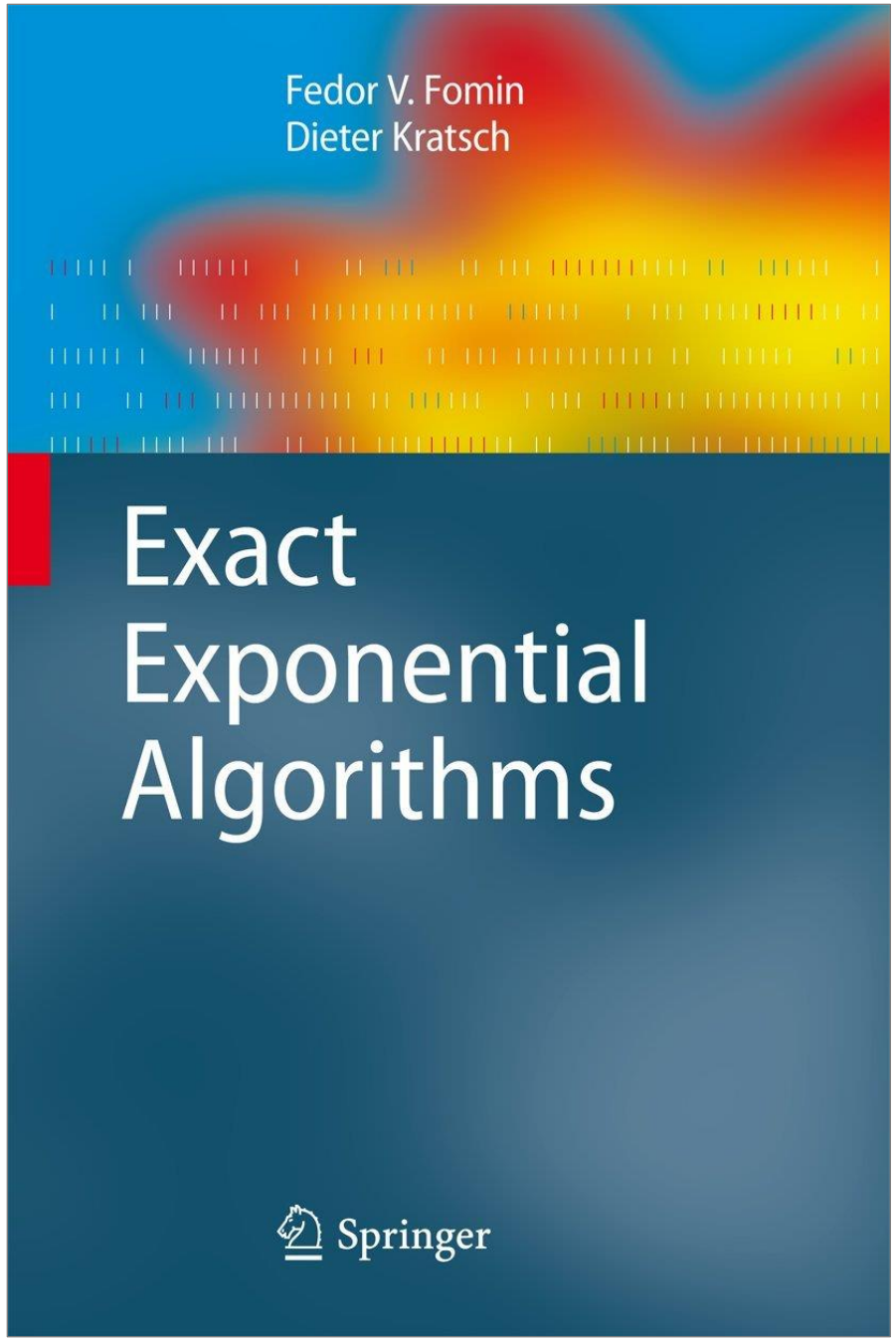
Maksimal parring M
⇒ knudedækning C

Minimum knudedækning kan i polynomiel tid ikke approksimeres bedre end…

- faktor $\sqrt{2} \approx 1.4142$, såfremt **P ≠ NP** (datalogiens store åbne spørgsmål)
- faktor 2, såfremt ”uniq games conjecture” er sandt (dvs. ovenstående er bedst mulig)

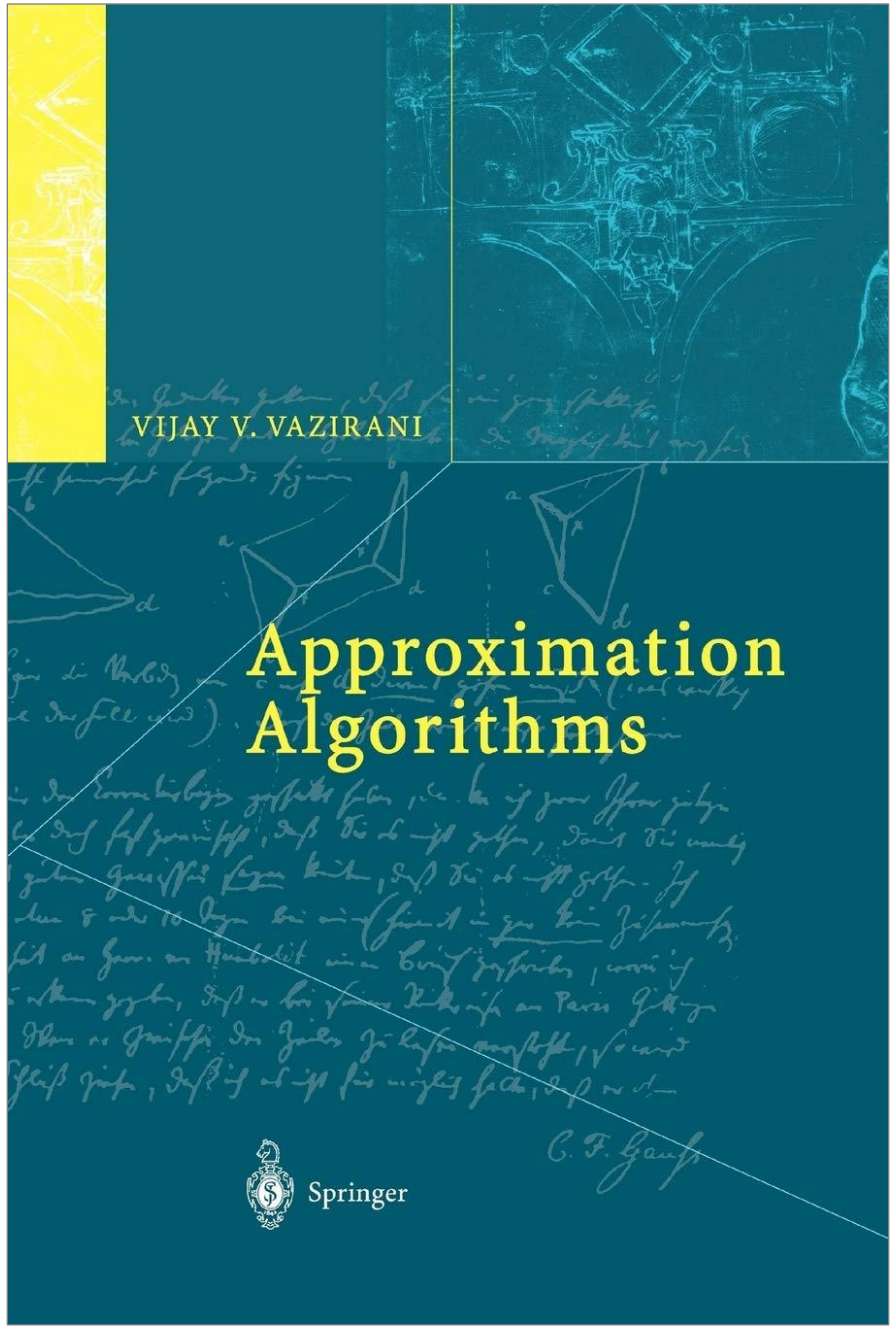
Opsummering

- Maksimal/maksimum/stabil parring
- Maksimum klike
- Maksimum uafhængige mængder
- Minimal knudedækning
- 3-SAT (NP-hårdhed)
- Reduktioner
- Eksponentieltids algoritmer
- Approksimations algoritmer



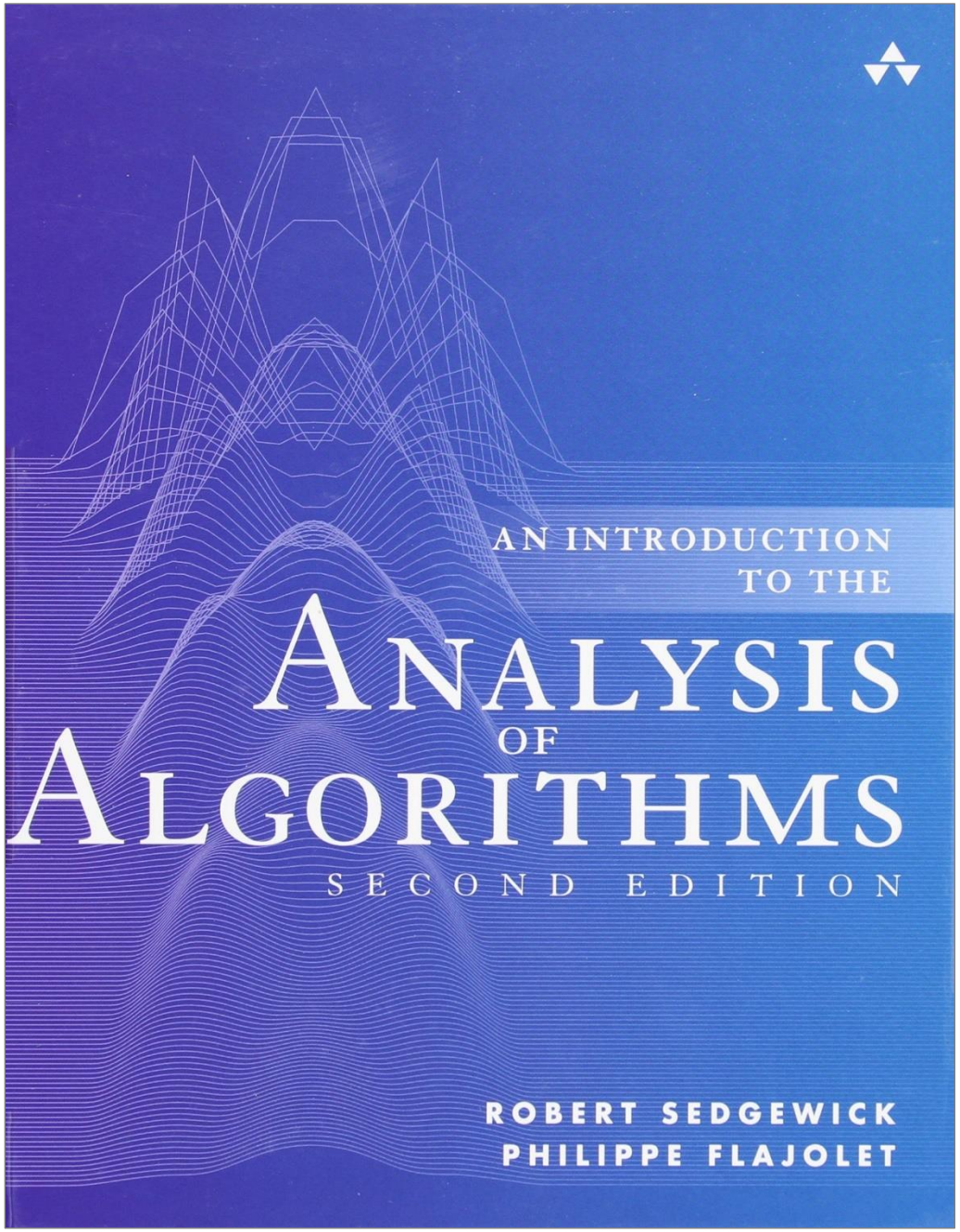
Forklarer de mest almindelige algoritmiske teknikker for at udvikle eksakte (eksponentiel tids) algoritmer for NP-hårde problemer

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



Forklarer de dominerende teoretiske tilgangsvinkler for at finde approksimative løsninger til hårde kombinatoriske optimerings og nummererings problemer.

Approximation Algorithms,
Vijay V. Vazirani.
XIX - 380, Springer-Verlag Berlin Heidelberg 2003.
DOI 10.1007/978-3-662-04565-7



Detaljeret gennemgang af matematikken for at løse rekursionsligninger m.m.

An Introduction to the Analysis of Algorithms
Robert Sedgewick, Philippe Flajolet.
XVIII – 572, Addison-Wesley Professional 2013.
ISBN: 78-0-321-90575-8

Algoritmer og Datastrukturer

Eksamen

Eksamen

- Ordinær eksamen januar, reeksamen til maj
- Ingen hjælpemidler
- 2 timer
- Forberedelse til eksamen: Løs gamle eksamensopgaver + Q&A
- Eksamensopgaverne meget lig med gamle eksamensopgaver i dADS1 (- transitionssystemer, termineringsfunktioner)
  dADS2 (- flow/netværkstrømninger, strengalgoritmer)
- Bemærk: Indtil foråret 2017 lå kurserne på 2. semester og man kunne have lærebogen med til eksamen

Navn

Studienummer *ikke* AU id

Max 1 kryds per delspørgsmål

Algoritmer og Datastrukturer

Onsdag 19. januar 2022, 9:00–11:00 Studienummer

Navn:

1.1 A B C D E
1.2 A B C D E
1.3 A B C D E
1.4 A B C D E
1.5 A B C D E
1.6 A B C D E
1.7 A B C D E
1.8 A B C D E

2.1 A B C D E F G H I J K
2.2 A B C D E F G H I J K
2.3 A B C D E F G H I J K
2.4 A B C D E F G H I J K
2.5 A B C D E F G H I J K

3.1 A B C D E F G
3.2 A B C D E F G
3.3 A B C D E F G
3.4 A B C D E F G
3.5 A B C D E F G
3.6 A B C D E F G

4.1 A B
4.2 A B
4.3 A B
4.4 A B
4.5 A B
4.6 A B

5 A B C D E

6.1 A B C D E F G H I J
6.2 A B C D E F G H I J
6.3 A B C D E F G H I J
6.4 A B C D E F G H I J
6.5 A B C D E F G H I J
6.6 A B C D E F G H I J
6.7 A B C D E F G H I J

7.1 A B C D E
7.2 A B C D E
7.3 A B C D E

8 A B C D E F

9.1 A B C D E F G H I J K
9.2 A B C D E F G H I J K
9.3 A B C D E F G H I J K
9.4 A B C D E F G H I J K
9.5 A B C D E F G H I J K

10.1 A B C D E F G H I J K L
10.2 A B C D E F G H I J K L
10.3 A B C D E F G H I J K L
10.4 A B C D E F G H I J K L
10.5 A B C D E F G H I J K L

11.1 A B C D E F G H I J K L
11.2 A B C D E F G H I J K L
11.3 A B C D E F G H I J K L

12 A B C D E

13 A B C D E F

14 A B C D E F

15.1 A B C
15.2 A B C

16.1 A B C D
16.2 A B C D
16.3 A B C D
16.4 A B C D

17.1 A B C D E F G H I J K L
17.2 A B C D E F G H I J K L
17.3 A B C D E F G H I J K L
17.4 A B C D E F G H I J K L

18 A B C D E F

19.1 A B C D E F G H I J K
19.2 A B C D E F G H I J K
19.3 A B C D E F G H I J K

20.1 A B C D E
20.2 A B C D E
20.3 A B C D E

21 A B C D E F

22 A B C D E F G

23 A B C D E F

24.1 A B C
24.2 A B C
24.3 A B C
24.4 A B C
24.5 A B C
24.6 A B C
24.7 A B C

25 A B C D E F

EKSAMEN

Algoritmer og Datastrukturer

Onsdag 19. januar 2021, 9:00–11:00

Institut for Datalogi, Naturvidenskabelige Fakultet, Aarhus Universitet

Antal sider i opgavesættet (incl. forsiden): 14

Tilladte medbragte hjælpemidler: **Ingen**

Kun det separate svarark skal udfyldes og afleveres.

Denne opgaveformulering skal *ikke* afleveres.

Vejledning og pointgivning

Dette eksamenssæt består af en mængde multiple-choice-opgaver.

Svarene på opgaverne angives på det **separate svarark som afleveres**.

For hver opgave er angivet opgavens andel af det samlede eksamenssæt.

Hvert delspørgsmål har præcist et rigtigt svar.

For hvert delspørgsmål må du vælge **max ét svar** ved at afkrydse den tilsvarende rubrik.

Et delspørgsmål bedømmes som følgende:

- Hvis du sætter kryds ved det rigtige svar, får du 1 point.
- Hvis du ikke sætter nogen krydser, får du 0 point.
- Hvis du sætter kryds ved et forkert svar, får du $-\frac{1}{k-1}$ point, hvor k er antal svarmuligheder.

For en opgave med vægt $v\,\%$ og med n delspørgsmål, hvor du opnår samlet s point, beregnes din besvarelse af opgaven som:

$$\frac{s}{n} \cdot v\,\%$$

Bemærk at det er muligt at få negative point for en opgave.