Premaster Course Algorithms 1

SS 2019

Prof. Dr. Christian Scheideler

Basic Information

Lectures:

- Mo 6 – 8 pm F1.110

Exam:

- Oral exam at end of course

Course Webpage:

- http://cs.uni-paderborn.de/ti/lehre/veranstaltungen/ss-2019

Office hours:

- Thu, 4-5 pm, F2.326

Basic Information

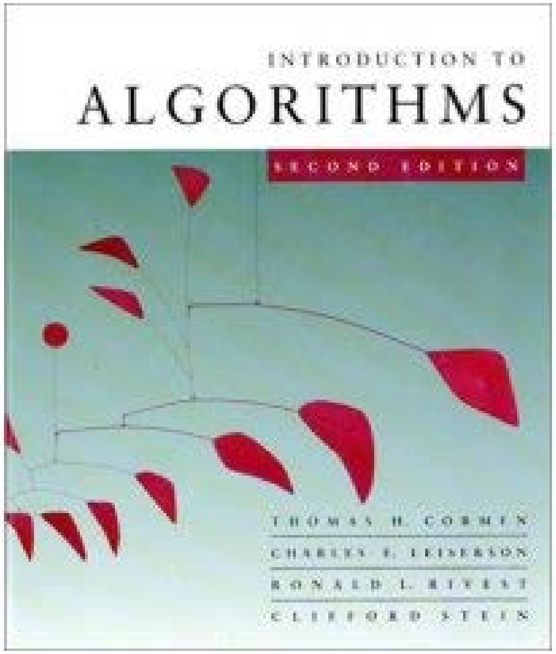
Literature:

Cormen, Leiserson, Rivest, Stein:

Introduction to Algorithms, 3rd ed.

MIT Press/McGraw-Hill

ISBN 0-262-53305-8



Basic Information

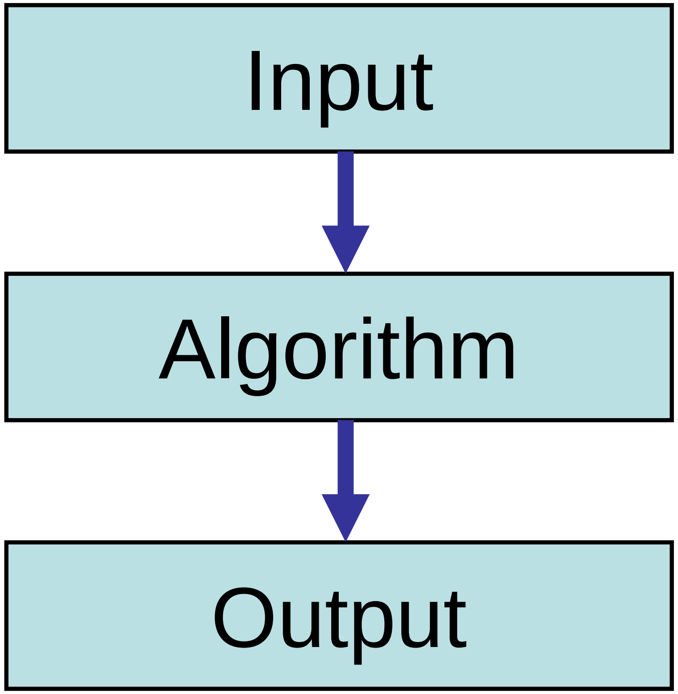
Contents of the course:

- April 8: Chapters 1-4 (Intro and runtime analysis)
- April 15: Chapters 6-7 (Sorting)
- April 29: Chapters 10-12 (Elementary data structures)
- May 6: Chapters 24-25 (Basic graph algorithms)
- May 13: Chapters 24-25 (Shortest paths)
- May 20: Chapter 26 (Network flow)

What is an Algorithm?

Definition 1.1: An *algorithm* is a concise description of a procedure to solve a certain class of problems.

Here:



Input

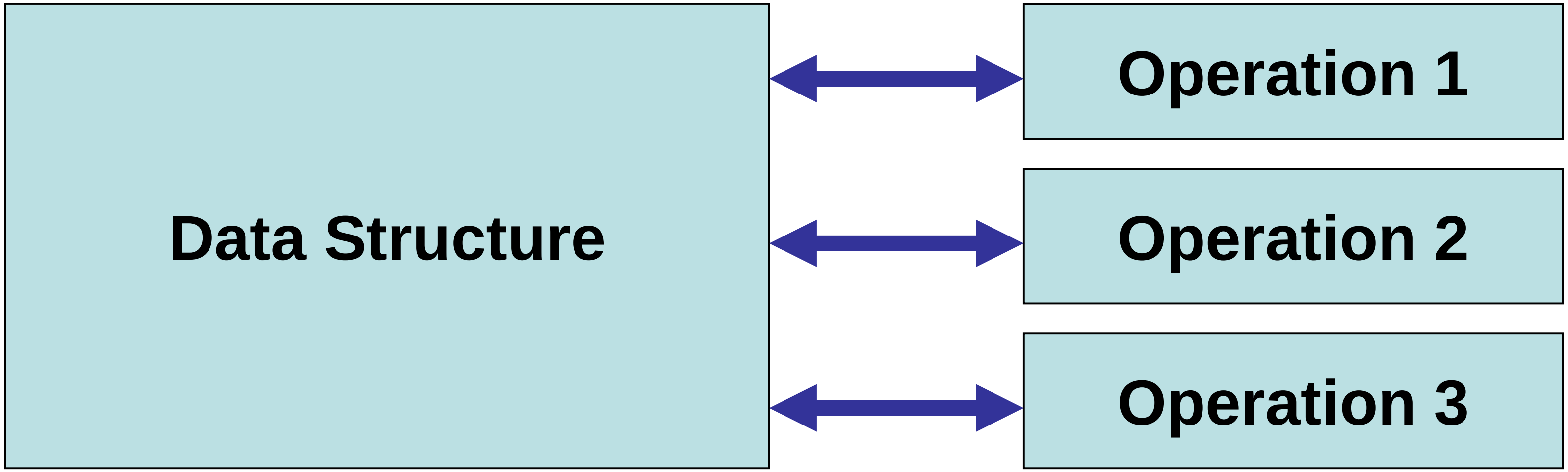
Algorithm

Output

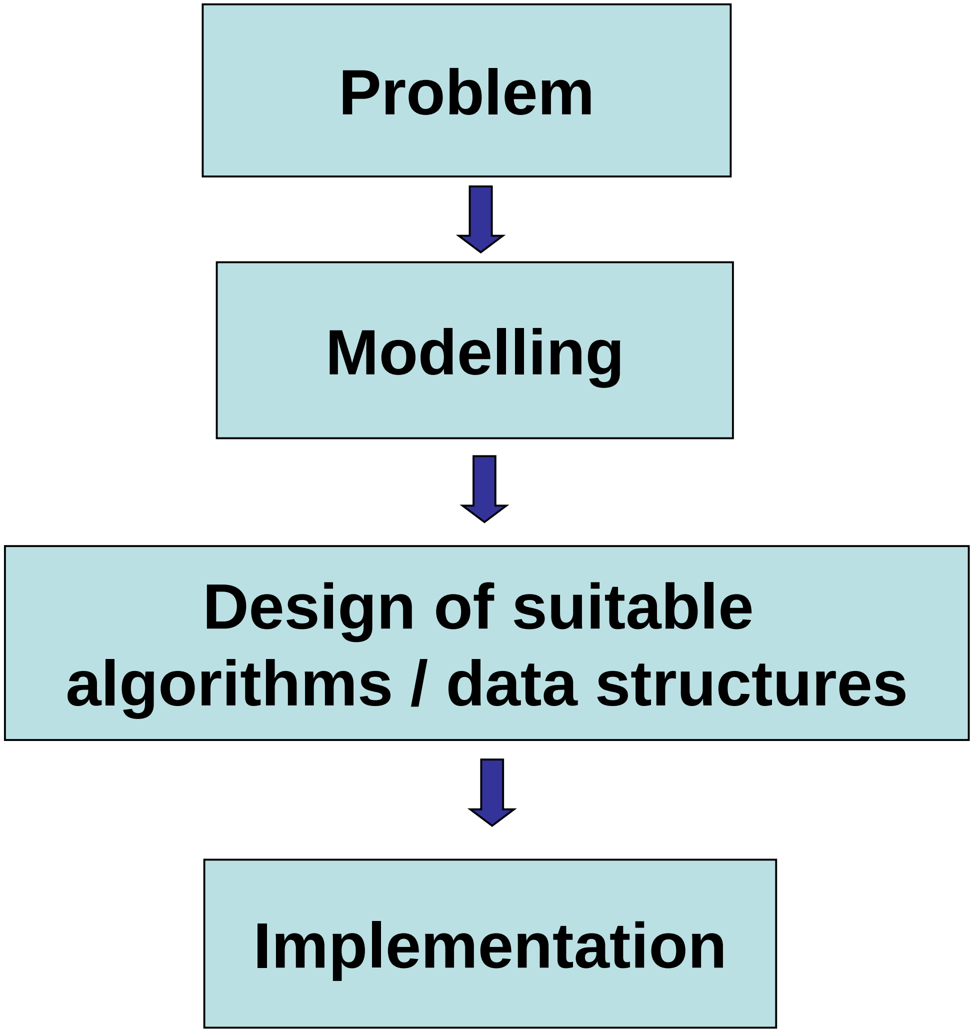
What is a Data Structure?

Definition 1.2: A *data structure* is a specific way of organizing data in the memory of a computer to facilitate operations like *Seach, Insert, and Delete*.

Here:



Software Development



Problem

Modelling

Design of suitable algorithms / data structures

Implementation

Important Criteria

- ➢ Algorithms / data structures must be correct.

  ⇒ *Correctness proofs.*

- ➢ Algorithms / data structures should work efficiently.

  ⇒ *Analytical methods for time and space analysis*.

- ➢ For *guarantees*, analytical methods *cannot* rely on empirical studies but must be based on a *mathematical analysis*.

Design of Algorithms

A rigorous algorithmic approach requires:

1. *Formal description of algorithm (in pseudo-code)*
2. *Correctness proof*
3. *Formal time and/or space analysis*

Notation

Pseudocode:

- Loops (for, while, repeat)
- Branching (if – then – else)
- Returning from procedure call (return)
- Assignment (:=)
- Block structure via indentation

Time needed for elementary operations (like assignments and checks): constant

Example: Minimum Search

Input: sequence A of n numbers $(a_1,a_2,\dots,a_n)$

Output: index i with $a_i \le a_j$ for all $1 \le j \le n$.

Algorithm:

Min-Search(A):

```
min:=1
for j:=2 to length(A)
    if A[j]<A[min] then min:=j
return min
```

Example:

Input: (31,41,59,26,51,48)

Output: 4

Example: Sorting

Input: sequence A of n numbers $(a_1,a_2,\dots,a_n)$
Output: permutation of $(a_1,a_2,\dots,a_n)$ into $(b_1,b_2,\dots,b_n)$ with $b_1 \le b_2 \le \dots \le b_n$.

Algorithm:

Insertion-Sort(A):

```
for j:=2 to length(A)
    key:=A[j]; i:=j-1  // insert A[j] into A[1..j-1]
    while i>0 and A[i]>key
        A[i+1]:=A[i]
        i:=i-1
    A[i+1]:=key
return A
```

Example: Sorting

Runtime analysis:

```
Insertion-Sort(A):                      cost     times
1 for j:=2 to length(A)                 $c_1$    $n$
2     key:=A[j]; i:=j-1                 $c_2$    $n-1$
3     while i>0 and A[i]>key            $c_3$    $\Sigma_{j=2}^{n} t_j$
4         A[i+1]:=A[i]                  $c_4$    $\Sigma_{j=2}^{n} (t_j-1)$
5         i:=i-1                        $c_5$    $\Sigma_{j=2}^{n} (t_j-1)$
6     A[i+1]:=key                       $c_6$    $n-1$
7 return A                              $c_7$    $1$
```

Worst case: $t_j=j$ (A[j] has to be placed into A[1])

Worst case runtime: $T(n) \approx c \cdot n^2$ for some constant c

Example: Sorting

Input: sequence A of n numbers $(a_1,a_2,\ldots,a_n)$
Output: permutation of $(a_1,a_2,\ldots,a_n)$ into $(b_1,b_2,\ldots,b_n)$ with $b_1 \le b_2 \le \ldots \le b_n$.

Algorithm: Merge-Sort(A,1,n)

Merge-Sort(A,p,r):

```
if $p<r$
    $q:=\lfloor (p+r)/2 \rfloor$  // compute middle position
    Merge-Sort(A,p,q)
    Merge-Sort(A,q+1,r)
    Merge(A,p,q,r)
```

Merge(A,p,q,r) merges sorted A[p..q] and A[q+1…r] to sorted A[p..r].

Illustration of Merge-Sort (1)

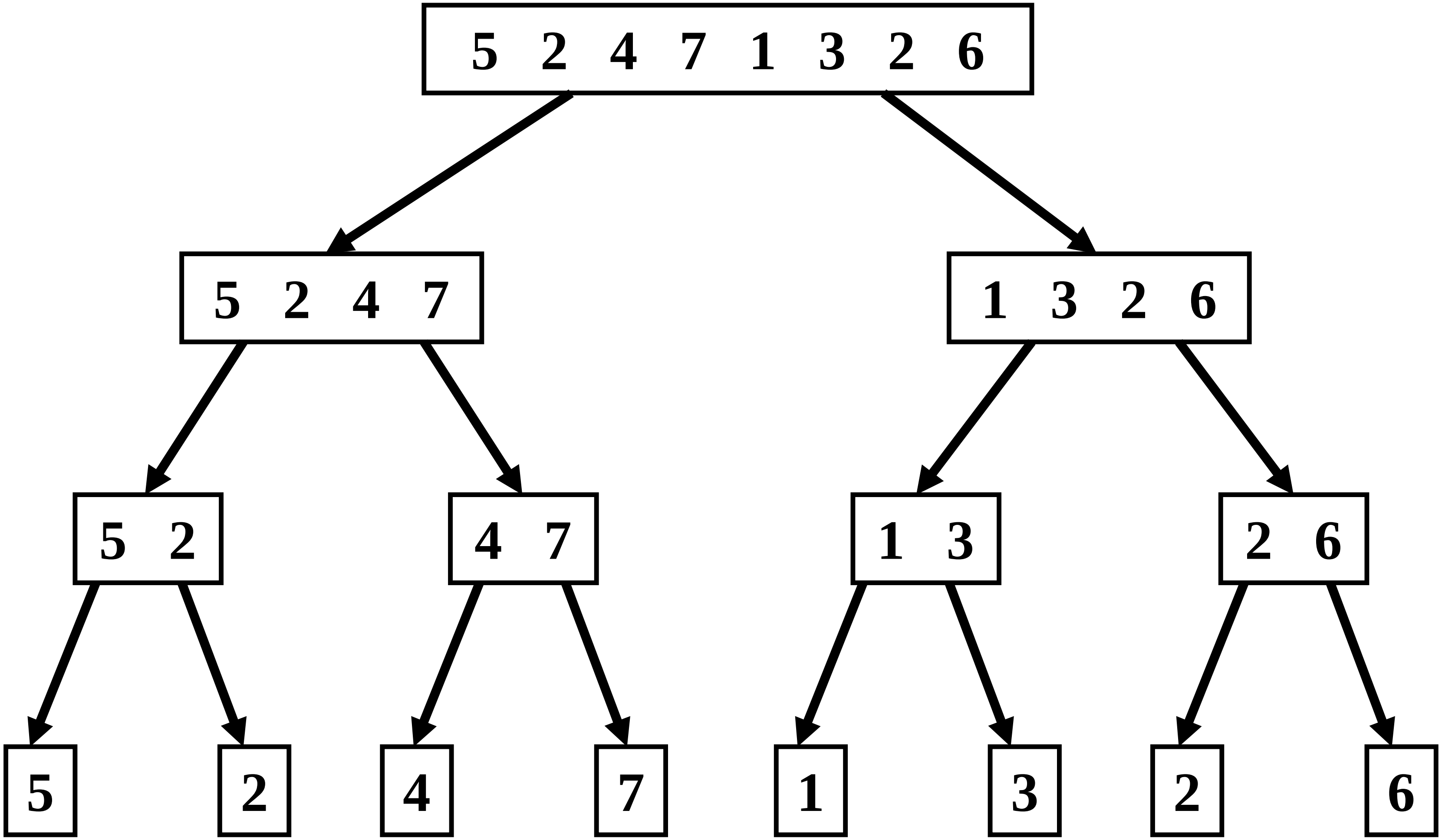
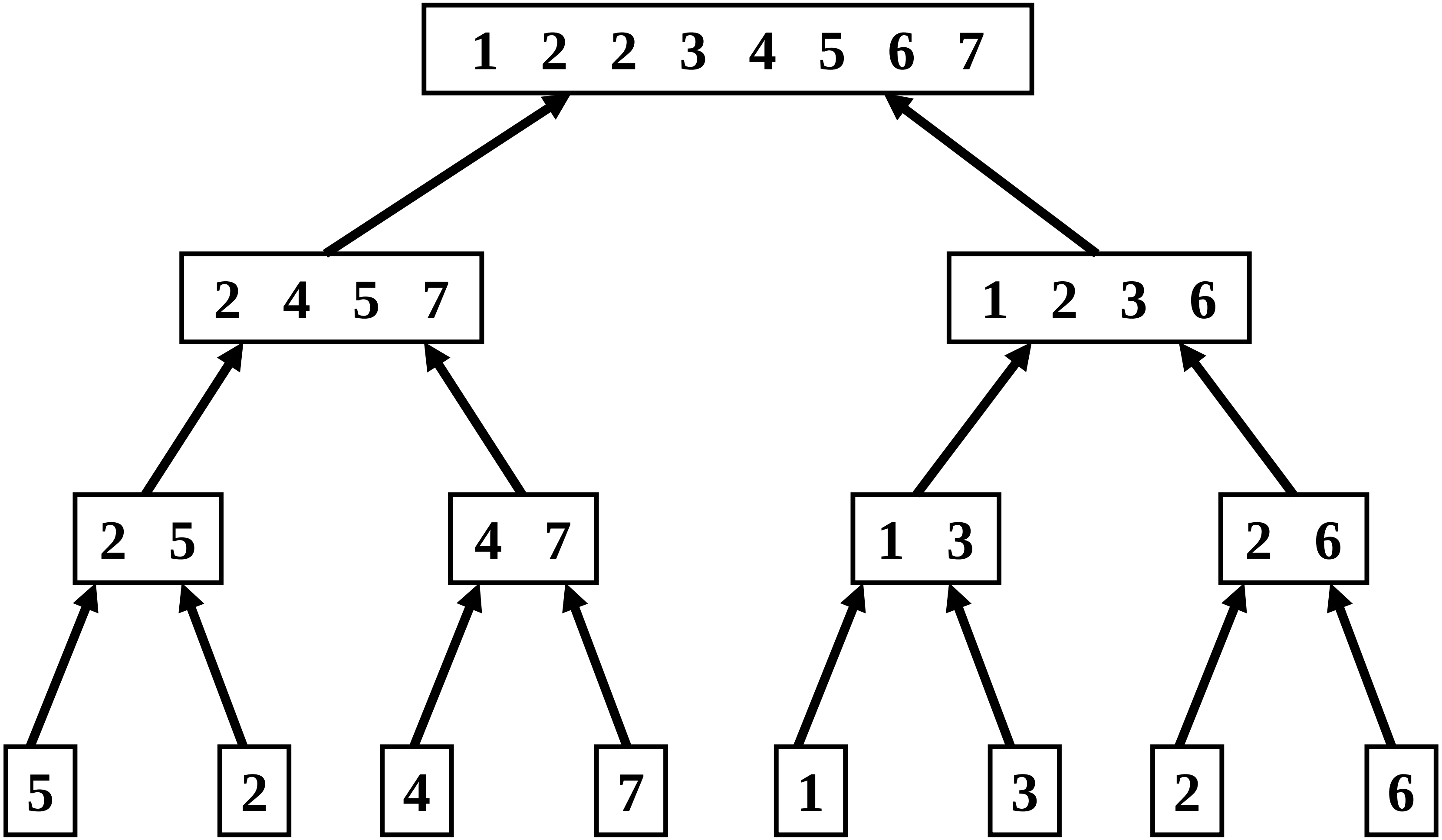


Illustration of Merge-Sort (2)



Example: Sorting

Runtime analysis:

- $n=r-p+1$: number of elements
- $T(n)$: runtime of Merge-Sort on n elements

Merge-Sort(A,p,r):	cost	times
if $p<r$	c_1	1
$q:=\lfloor (p+r)/2 \rfloor$	c_2	1
Merge-Sort(A,p,q)	$T(q-p+1)$	1
Merge-Sort(A,q+1,r)	$T(r-q)$	1
Merge(A,p,q,r)	$c_3 \cdot n$	1

Example: Sorting

Runtime analysis:

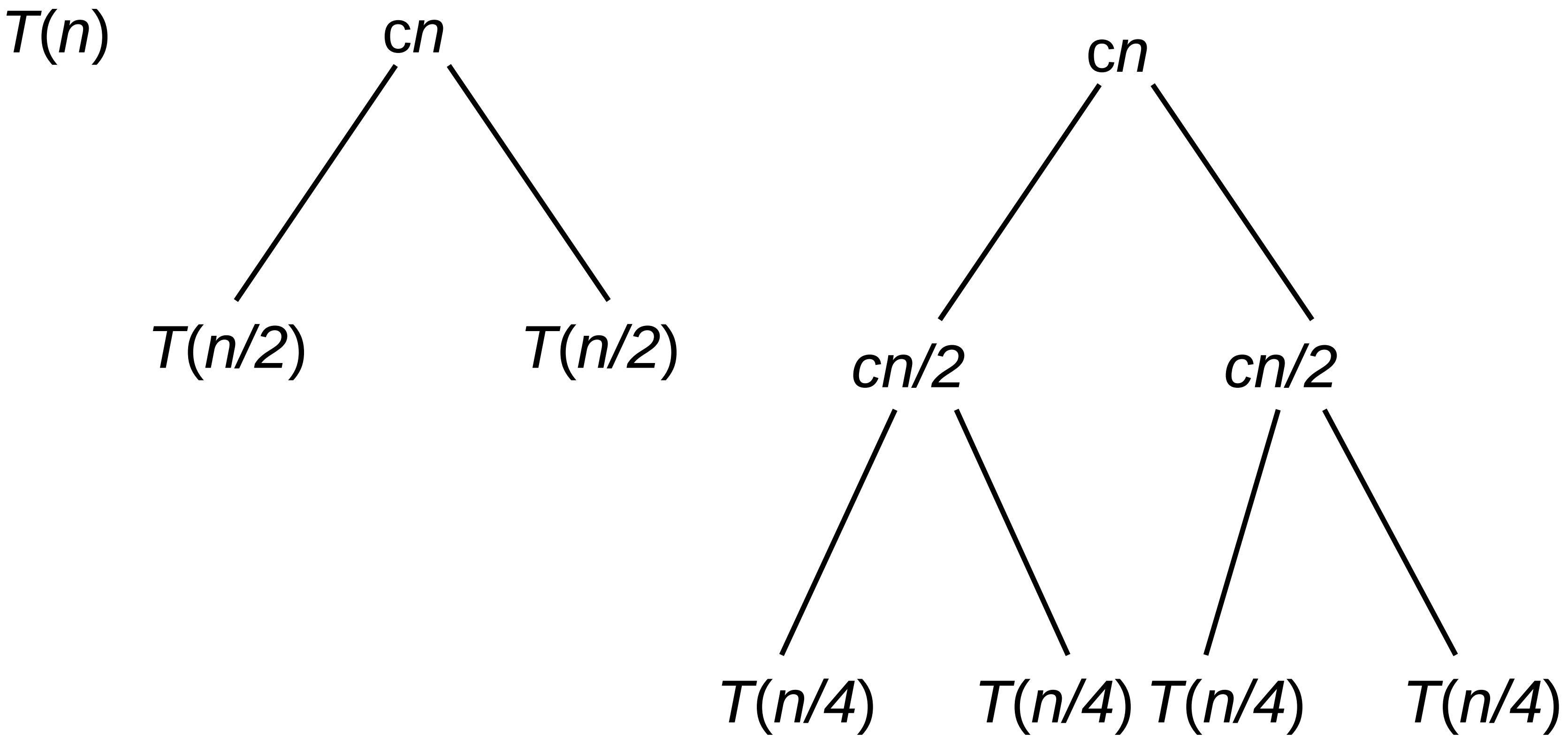
Merge-Sort(A,p,r):	cost	times
if $p<r$	c_1	1
$q:=\lfloor (p+r)/2 \rfloor$	c_2	1
Merge-Sort(A,p,q)	$T(q-p+1)$	1
Merge-Sort(A,q+1,r)	$T(r-q)$	1
Merge(A,p,q,r)	$c_3 \cdot n$	1

Merge(A,p,q,r) merges sorted A[p..q] and A[q+1…r] to sorted A[p..r]. Suppose that n is a power of 2. Then:

Runtime:

$$T(n) = \begin{cases} 2 \cdot T(n/2)+c_3 \cdot n+c_1+c_2 & \text{if } n>1 \\ c_1 & \text{if } n=1 \end{cases}$$

Runtime of Merge-Sort



$T(n)$

cn

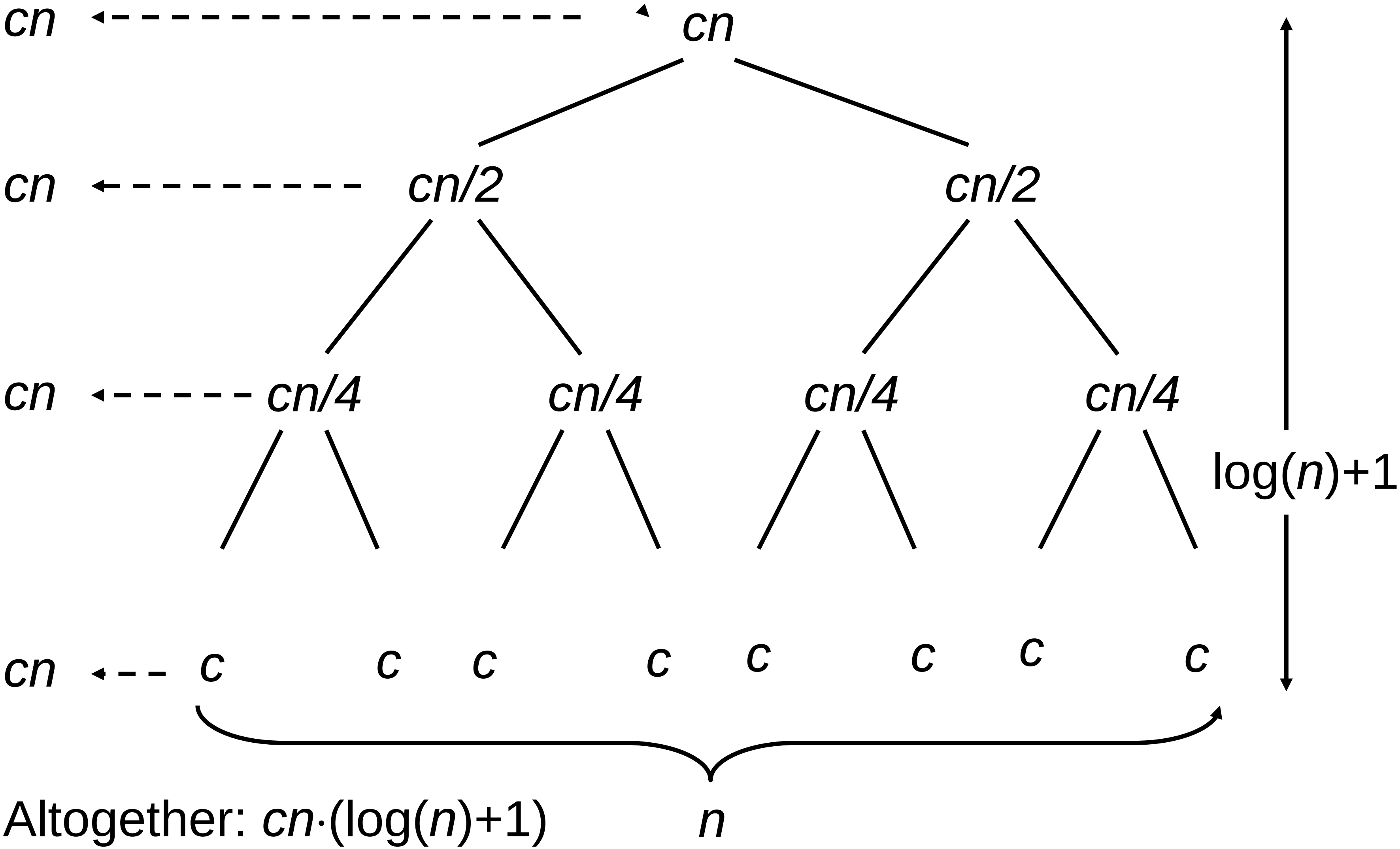
$T(n/2)$ $T(n/2)$

cn

$cn/2$ $cn/2$

$T(n/4)$ $T(n/4)$ $T(n/4)$ $T(n/4)$

Runtime of Merge-Sort


cn

cn

cn/2 cn/2

cn

cn/4 cn/4 cn/4 cn/4

cn

c c c c c c c c

log(*n*)+1

n

Altogether: $cn \cdot (\log(n)+1)$

Asymptotic Notation

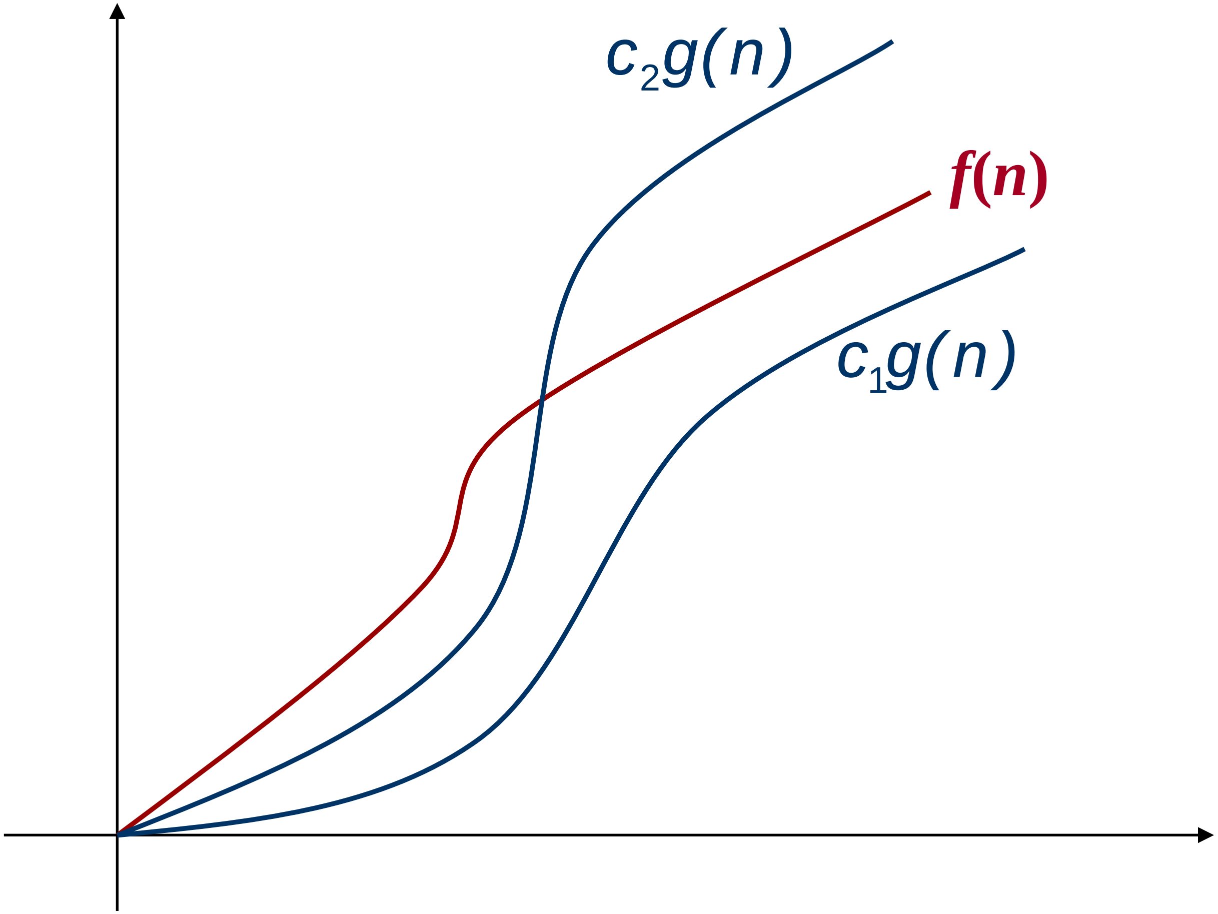
In theory, often just the asymptotic runtime of algorithms is of interest.

Asymptotic runtime ignores constants and just specifies the growth of functions.

Instead of $T(n)=c_1 \cdot n^2+c_2 \cdot n-c_3$ just $T(n)=\Theta(n^2)$.
(Θ: „grows as fast as“)

Formally,
$\Theta(g(n))$ = { f(n) | there exist positive constants $c_1 \leq c_2$ and n_0 such that for all $n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ }

Illustration of Θ(*g*(*n*))



Asymptotic Notation

$\Theta(g(n))$ = { f(n) | there exist positive constants $c_1 \leq c_2$ and n_0 such that for all $n \geq n_0$, $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ }

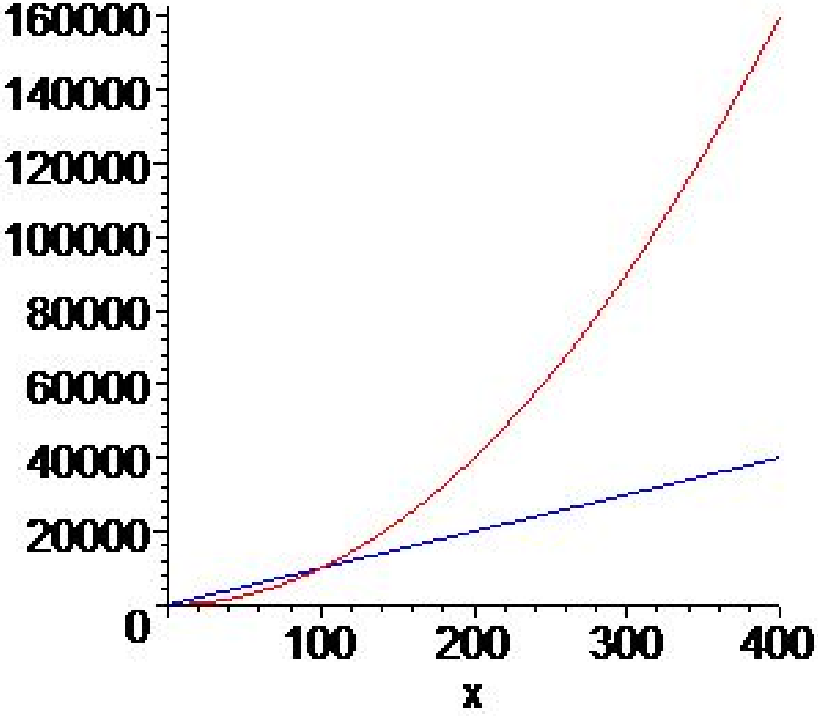
$O(g(n))$ = { f(n) | there exist positive constants c and n_0 such that for all $n \geq n_0$, $0 \leq f(n) \leq c \cdot g(n)$ }

$\Omega(g(n))$ = { f(n) | there exist positive constants c and n_0 such that for all $n \geq n_0$, $0 \leq c \cdot g(n) \leq f(n)$ }

If $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$ then $f(n) \in \Theta(g(n))$.

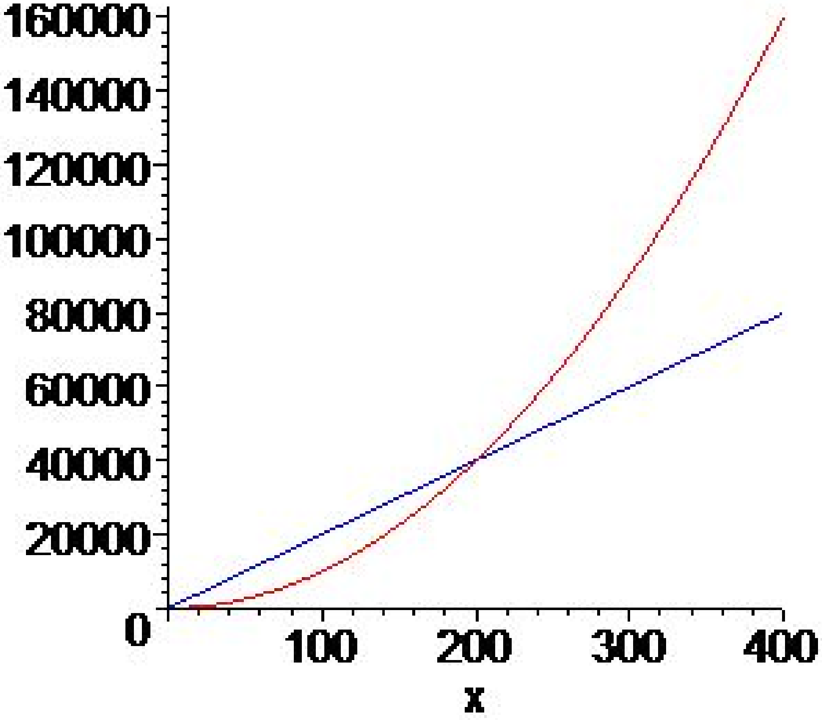
By abuse of notation, one often writes $f(n)=O(g(n))$ instead of $f(n) \in O(g(n))$.

Illustration of O(*g*(*n*))



$g(x)=x^2$

$f(x)=100x$



$g(x)=x^2$

$f(x)=200x$

Asymptotic Notation

Transitivity:

- If $f(n)=\Theta(g(n))$ and $g(n)=\Theta(h(n))$ then $f(n)=\Theta(h(n))$.
- If $f(n)=O(g(n))$ and $g(n)=O(h(n))$ then $f(n)=O(h(n))$.
- If $f(n)=\Omega(g(n))$ and $g(n)=\Omega(h(n))$ then $f(n)=\Omega(h(n))$.

Reflexivity:

- $f(n)=\Theta(f(n))$, $f(n)=O(f(n))$, and $f(n)=\Omega(f(n))$

Symmetry:

- $f(n)=\Theta(g(n))$ if and only if $g(n)=\Theta(f(n))$.

Transpose symmetry:

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

Asymptotic Notation

Theorem 1.3: Let $p(n) = \sum_{i=0}^{k} a_i \cdot n^i$ with $a_k > 0$. Then we have $p(n) = \Theta(n^k)$.

Proof:

To show: $p(n) = O(n^k)$ and $p(n) = \Omega(n^k)$.

- $p(n) = O(n^k)$: For all $n \geq 1$,

  $$p(n) \leq \sum_{i=0}^{k} |a_i|\, n^i \leq n^k \sum_{i=0}^{k} |a_i|$$

- Hence, definition of $O()$ is satisfied with $c=\sum_{i=0}^{k} |a_i|$ and $n_0=1$.

- $p(n) = \Omega(n^k)$: For all $n \geq 2k \cdot A/a_k$ and $A=\max_i |a_i|$,

  $p(n) \geq a_k \cdot n^k - \sum_{i=0}^{k-1} A \cdot n^i \geq a_k\, n^k - k \cdot A\, n^{k-1} \geq a_k\, n^k/2$

- Hence, definition of $\Omega()$ is satisfied with $c=a_k/2$ and $n_0=2kA/a_k$.

Asymptotic Runtime Analysis

Worst-case runtime:

- $T(I)$: worst-case runtime of instruction I
- $T(\text{elementary command}) = O(1)$
- $T(\text{return } x) = O(1)$
- $T(I;I') = T(I) + T(I')$
- $T(\text{if } C \text{ then } I \text{ else } I') = T(C) + \max\{T(I),T(I')\}$
- $T(\text{for } i:=a \text{ to } b \text{ do } I) = \sum_{i=a}^{b} (O(1)+ T(I))$
- $T(\text{repeat } I \text{ until } C) = \sum_{i=1}^{k} (T(C)+T(I))$
  (k: number of iterations)
- $T(\text{while } C \text{ do } I) = \sum_{i=1}^{k} (T(C)+T(I))$

Runtime analysis difficult for while- und repeat-loops since we need to determine k, which is sometimes not so easy!

Example: Computation of Sign

Input: number $x \in \mathbb{R}$

```
Signum(x):
    if $x<0$ then return $-1$        $O(1)$
    if $x>0$ then return $1$         $O(1)$
    return $0$                       $O(1)$
```

total runtime: $O(1+1+1) = O(1)$

Example: Minimum

Input: array of numbers A[1],…,A[n]

```
Minimum(A):
min := 1                              $O(1)$
for i:=1 to n do                      $\sum_{i=1}^{n} (O(1)+T(I))$
    if A[i]<min then min:=A[i]        $O(1)$
return min                            $O(1)$
```

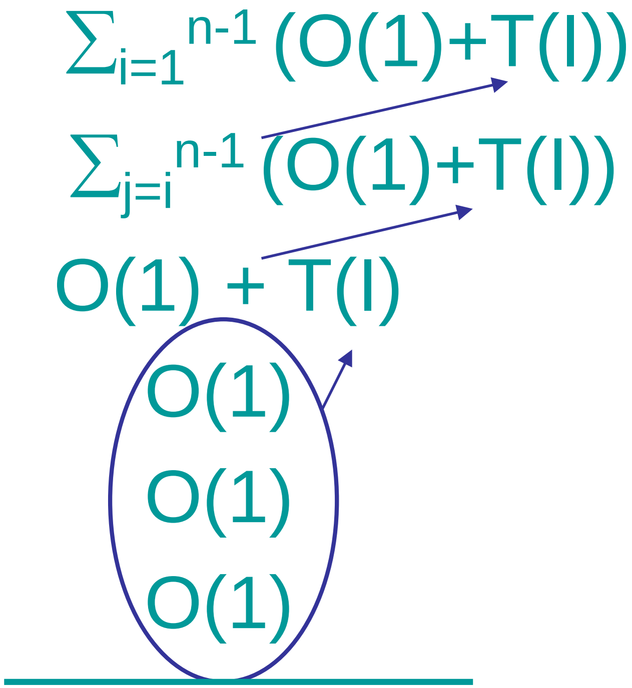
runtime: $O(1) + \sum_{i=1}^{n} O(1) + O(1) = O(n)$

Example: Sorting

Input: array of numbers A[1],….,A[n]

Bubblesort(A):

```
for i:=1 to n-1 do
    for j:= n-1 downto i do
        if A[j]>A[j+1] then
            x:=A[j]
            A[j]:=A[j+1]
            A[j+1]:=x
```



$\sum_{i=1}^{n-1} (O(1)+T(I))$

$\sum_{j=i}^{n-1} (O(1)+T(I))$

$O(1) + T(I)$

$O(1)$

$O(1)$

$O(1)$

runtime: $\sum_{i=1}^{n-1} \sum_{j=i}^{n-1} O(1) = O(n^2)$

Master Theorem

Theorem 1.4: For some positive constants a,b,c with $n=b^k$ for some natural number k let

$$T(n) = c \qquad \text{if } n \leq 1$$

$$T(n) = c \cdot n + a \cdot T(n/b) \qquad \text{if } n > 1$$

Then it holds that

$$T(n) = \Theta(n) \qquad \text{if } a<b$$

$$T(n) = \Theta(n \log n) \qquad \text{if } a=b$$

$$T(n) = \Theta(n^{\log_b a}) \qquad \text{if } a>b$$

Master Theorem

Proof:

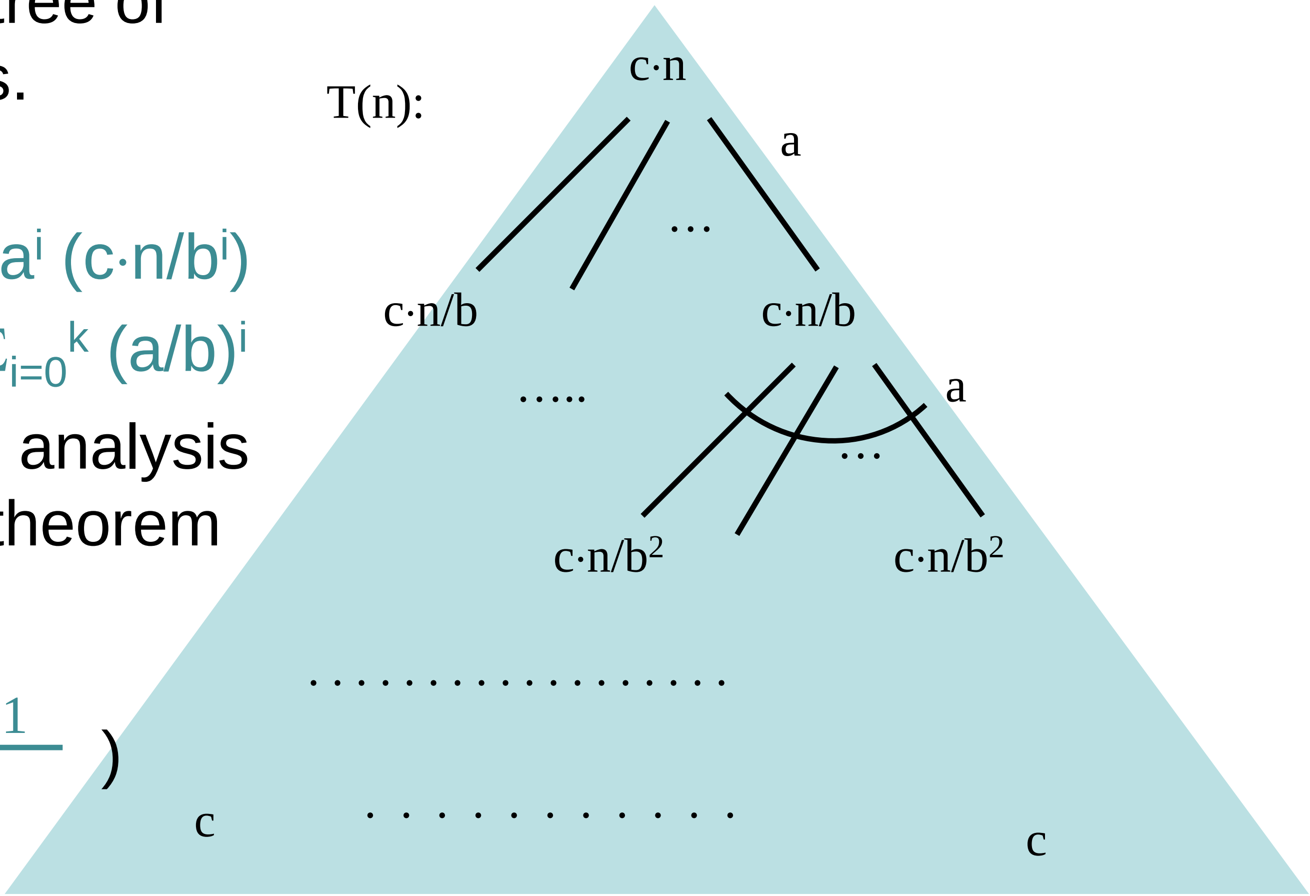
- Consider the tree of recursive calls.
- It holds:

  $T(n) \leq \Sigma_{i=0}^{k} a^i (c\cdot n/b^i)$

  $\leq c\cdot n\ \Sigma_{i=0}^{k} (a/b)^i$

- Case-by-case analysis results in the theorem (use for $a \neq b$

  $\Sigma_{i=0}^{k} z^i = \frac{z^{i+1}-1}{z-1}$)



Example: Matrix Multiplication

$$\begin{pmatrix} 3 & 7 & 5 & 4 \\ 0 & 3 & 2 & 4 \\ 10 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 3 \\ 3 & 1 & 1 & 0 \\ 2 & 3 & 2 & 2 \end{pmatrix} = \begin{pmatrix} \quad & \quad & \quad & \quad \\ & & & \\ & & & \\ & & & \end{pmatrix}$$

Matrix Multiplication

$A=(a_{ij})_{1\leq i,j\leq n}$ $B=(b_{ij})_{1\leq i,j\leq n}$ $C=(c_{ij})_{1\leq i,j\leq n}$

$$\begin{pmatrix} 3 & 7 & 5 & 4 \\ 0 & 3 & 2 & 4 \\ 10 & 2 & 1 & 0 \\ 1 & 0 & 1 & 0 \end{pmatrix} \bullet \begin{pmatrix} 2 & 1 & 1 & 0 \\ 0 & 0 & 1 & 3 \\ 3 & 1 & 1 & 0 \\ 2 & 3 & 2 & 2 \end{pmatrix} = \begin{pmatrix} 29 & 20 & 23 & 29 \\ 14 & 14 & \dots & \\ \dots & & & \\ \dots & & & \end{pmatrix}$$

$c_{ij}= \Sigma_{k=1}^{n} a_{ik} \cdot b_{kj}$

Matrix Multiplication

Problem: Compute product of two n×n matrices

- Input: matrices X,Y
- Output: matrix Z = X·Y

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & x_{1,3} & x_{1,4} \\ x_{2,1} & x_{2,2} & x_{2,3} & x_{2,4} \\ x_{3,1} & x_{3,2} & x_{3,3} & x_{3,4} \\ x_{4,1} & x_{4,2} & x_{4,3} & x_{4,4} \end{pmatrix}, \quad Y = \begin{pmatrix} y_{1,1} & y_{1,2} & y_{1,3} & y_{1,4} \\ y_{2,1} & y_{2,2} & y_{2,3} & y_{2,4} \\ y_{3,1} & y_{3,2} & y_{3,3} & y_{3,4} \\ y_{4,1} & y_{4,2} & y_{4,3} & y_{4,4} \end{pmatrix}$$

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)
**new** array Z[1,..,n][1,..,n]
**for** i←1 **to** n **do**
    **for** j ← 1 **to** n **do**
        Z[i][j] ← 0
        **for** k ←1 **to** n **do**
            Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
**return** Z
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)
    new array Z[1,..,n][1,..,n]
    for i ← 1 to n do
        for j ← 1 to n do
            Z[i][j] ← 0
            for k ← 1 to n do
                Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
    return Z
```

Runtime:

$\Theta(n^2)$

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                          Runtime:
new array Z[1,..,n][1,..,n]                                  $\Theta(n^2)$
for $i \leftarrow 1$ to n do                                 $\Theta(n)$
    for $j \leftarrow 1$ to n do
        $Z[i][j] \leftarrow 0$
        for $k \leftarrow 1$ to n do
            $Z[i][j] \leftarrow Z[i][j] + X[i][k] \cdot Y[k][j]$
return Z
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                      Runtime:
new array Z[1,..,n][1,..,n]                              $\Theta(n^2)$
for i←1 to n do                                          $\Theta(n)$
    for j ← 1 to n do                                    $\Theta(n^2)$
        Z[i][j] ← 0
        for k ←1 to n do
            Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
return Z
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                         Runtime:
    new array Z[1,..,n][1,..,n]                             $\Theta(n^2)$
    for i←1 to n do                                         $\Theta(n)$
        for j ← 1 to n do                                   $\Theta(n^2)$
            Z[i][j] ← 0                                     $\Theta(n^2)$
            for k ←1 to n do
                Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
    return Z
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                      Runtime:
new array Z[1,..,n][1,..,n]                              $\Theta(n^2)$
for i ← 1 to n do                                        $\Theta(n)$
    for j ← 1 to n do                                    $\Theta(n^2)$
        Z[i][j] ← 0                                      $\Theta(n^2)$
        for k ← 1 to n do                                $\Theta(n^3)$
            Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]
return Z
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                          Runtime:
    new array Z[1,..,n][1,..,n]                              $\Theta(n^2)$
    for i $\leftarrow$ 1 to n do                                   $\Theta(n)$
        for j $\leftarrow$ 1 to n do                               $\Theta(n^2)$
            Z[i][j] $\leftarrow$ 0                                 $\Theta(n^2)$
            for k $\leftarrow$ 1 to n do                           $\Theta(n^3)$
                Z[i][j] $\leftarrow$ Z[i][j] + X[i][k] $\cdot$ Y[k][j]      $\Theta(n^3)$
    return Z
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                                 Runtime:
    new array Z[1,..,n][1,..,n]                                     $\Theta(n^2)$
    for i ← 1 to n do                                               $\Theta(n)$
        for j ← 1 to n do                                           $\Theta(n^2)$
            Z[i][j] ← 0                                             $\Theta(n^2)$
            for k ← 1 to n do                                       $\Theta(n^3)$
                Z[i][j] ← Z[i][j] + X[i][k] · Y[k][j]               $\Theta(n^3)$
    return Z                                                        $\Theta(1)$
```

Matrix Multiplication

```
MatrixMultiplication(Array X, Y, n)                               Runtime:
new array Z[1,..,n][1,..,n]                                       $\Theta(n^2)$
for $i \leftarrow 1$ to n do                                      $\Theta(n)$
    for $j \leftarrow 1$ to n do                                  $\Theta(n^2)$
        $Z[i][j] \leftarrow 0$                                    $\Theta(n^2)$
        for $k \leftarrow 1$ to n do                              $\Theta(n^3)$
            $Z[i][j] \leftarrow Z[i][j] + X[i][k] \cdot Y[k][j]$  $\Theta(n^3)$
return Z                                                          $\Theta(1)$
                                                                  ____________
                                                                  $\Theta(n^3)$
```

Matrix Multiplication

Recursive approach:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Recursive calls:

- 8 multiplications of n/2×n/2 matrices
- 4 additions of n/2×n/2 matrices

Matrix Multiplication

Recursive approach:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Recursive calls:

- 8 multiplications of n/2×n/2 matrices
- 4 additions of n/2×n/2 matrices

Runtime:

- $T(n) = 8 \cdot T(n/2) + \Theta(n^2)$

Matrix Multiplication

Runtime:

- $T(n) = \underset{\overset{\uparrow}{a}}{8} \cdot T(n/\underset{\overset{\uparrow}{b}}{2}) + \underset{\overset{\uparrow}{f(n)}}{k \cdot n^2}$

Extended Master Theorem (see book):

- $f(n) = k \cdot n^2$
- $a=8$, $b=2$
- Case 1: Runtime $\Theta(n^{\log_b a}) = \Theta(n^3)$
- Not better than elementary algorithm!

Matrix Multiplication

Strassen‘s Algorithm:

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \times \begin{pmatrix} E & F \\ G & H \end{pmatrix} = \begin{pmatrix} AE + BG & AF + BH \\ CE + DG & CF + DH \end{pmatrix}$$

Trick:

$P_1 = A \cdot (F-H)$

$P_2 = (A+B) \cdot H$

$P_3 = (C+D) \cdot E$

$P_4 = D \cdot (G-E)$

$P_5 = (A+D) \cdot (E+H)$

$P_6 = (B-D) \cdot (G+H)$

$P_7 = (A-C) \cdot (E+F)$

$AE+BG = P_5 + P_4 - P_2 + P_6$

$AF+BH = P_1 + P_2$

$CE+DG = P_3 + P_4$

$CF+DH = P_5 + P_1 - P_3 - P_7$

7 Multiplications!!!

Matrix Multiplication

Runtime:

- $T(n) = \underset{\overset{\uparrow}{a}}{7} \cdot T(n/\underset{\overset{\uparrow}{b}}{2}) + \underset{\overset{\uparrow}{f(n)}}{k \cdot n^2}$

Extended Master Theorem:

- $f(n) = k \cdot n^2$
- $a=7$, $b=2$
- Case 1: Runtime $\Theta(n^{\log_b a})=\Theta(n^{\log_2 7}) = \Theta(n^{2.81})$
- Better than elementary approach!

Summary

We covered:

- Pseudo code
- Asymptotic runtime analysis

Next lecture:

- Sorting