

# Datenstrukturen, Algorithmen und Programmierung 2

Professor Dr. Petra Mutzel

Lehrstuhl für Algorithm Engineering, LS11

1. VO

4. April 2006

# Motivation

„Warum soll ich in DAP2 gehen?“

**MERKE: DAP2 IST WICHTIG!!!**

„Ich kann doch schon programmieren.“

**ABER NICHT IMMER EFFIZIENT!**

# Überblick

Einführung

Organisatorisches zur

- Vorlesung
- Übung

# Kapitel 1: Einführung

1.1 Grundbegriffe

1.2 Beispiel: Sortierproblem

1.3 Analyse von Algorithmen

Konkrete Formulierung abstrakter Algorithmen, die sich auf bestimmte Darstellungen wie Datenstrukturen stützen

=

Summe aus Algorithmen und Datenstrukturen

Algorithmus

Datenstruktur

Programm

Rechenvorschrift

Eingabe

Berechnung

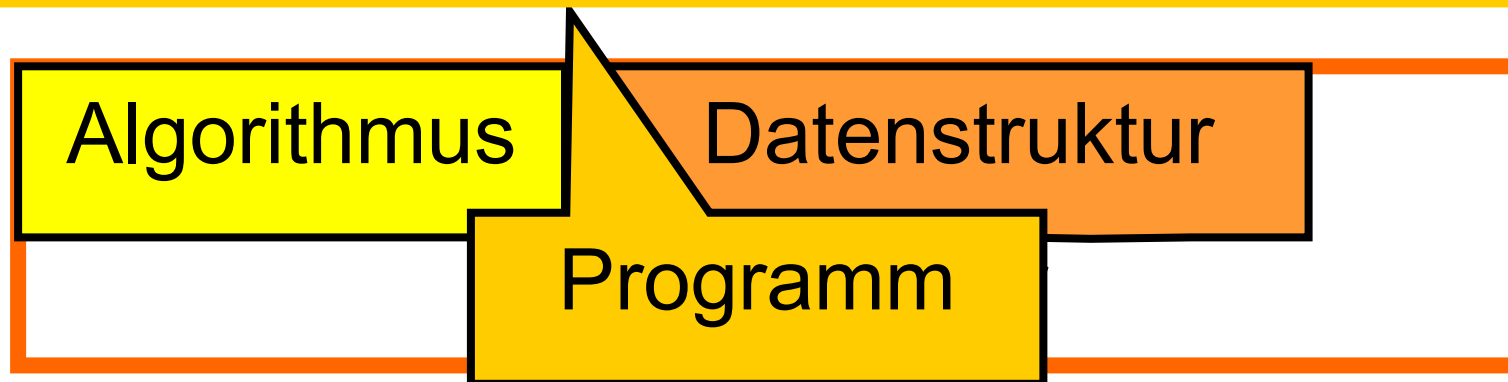
Ausgabe

Schema zur Repräsentation der durch einen Algorithmus behandelten Daten

Konkrete Formulierung abstrakter Algorithmen, die sich auf bestimmte Darstellungen wie Datenstrukturen stützen

=

Summe aus Algorithmen und Datenstrukturen



## Darstellung

- als Text (Deutsch, Englisch,...)
- als Computerprogramm (Java, C++,...)
- als Hardwaredesign
- als Pseudocode

DAP2

# Pseudocode

Vereinfachung real existierender  
Programmiersprachen

C / Java

```
int tmp = i;  
i = j;  
j = tmp;
```

Pseudocode

vertausche i mit j



```
for (int i=0;i<9;i++) {  
    float f =A[i]; ...  
}
```

```
for f:=A[0],...,A[8] {  
    ...  
}
```



# Beispiel: Sortierproblem

Eingabe: Folge von  $n$  Zahlen  $\langle a_1, a_2, \dots, a_n \rangle$

Ausgabe: Permutation  $\langle a'_1, a'_2, \dots, a'_n \rangle$   
der Eingabefolge, so dass  $a'_1 \leq a'_2 \leq \dots \leq a'_n$

Jede konkrete Zahlenfolge ist eine Instanz des  
Sortierproblems, z.B.:

$\langle 5, 3, 11, 2, 17 \rangle \rightarrow \langle 2, 3, 5, 11, 17 \rangle$

Gesucht: Korrekter Algorithmus für Problem

# Sortieren durch Einfügen

Eingabe: zu sortierende Zahlenfolge (key)

Ausgabe: sortierte Zahlenfolge

Setze Index  $k$  auf 2.-tes Element

Setze Index  $i$  auf  $(k-1)$ -tes Element

Solange  $i \neq 0$  und  $(\text{key}(k) < \text{key}(i))$

    Setze Index  $i$  auf  $(i-1)$ -tes Element

Platziere  $k$ -tes Element zwischen  $i$ -tes und  $(i+1)$ -tes Element

Analyse: Datenstruktur: ARRAY oder LISTE?

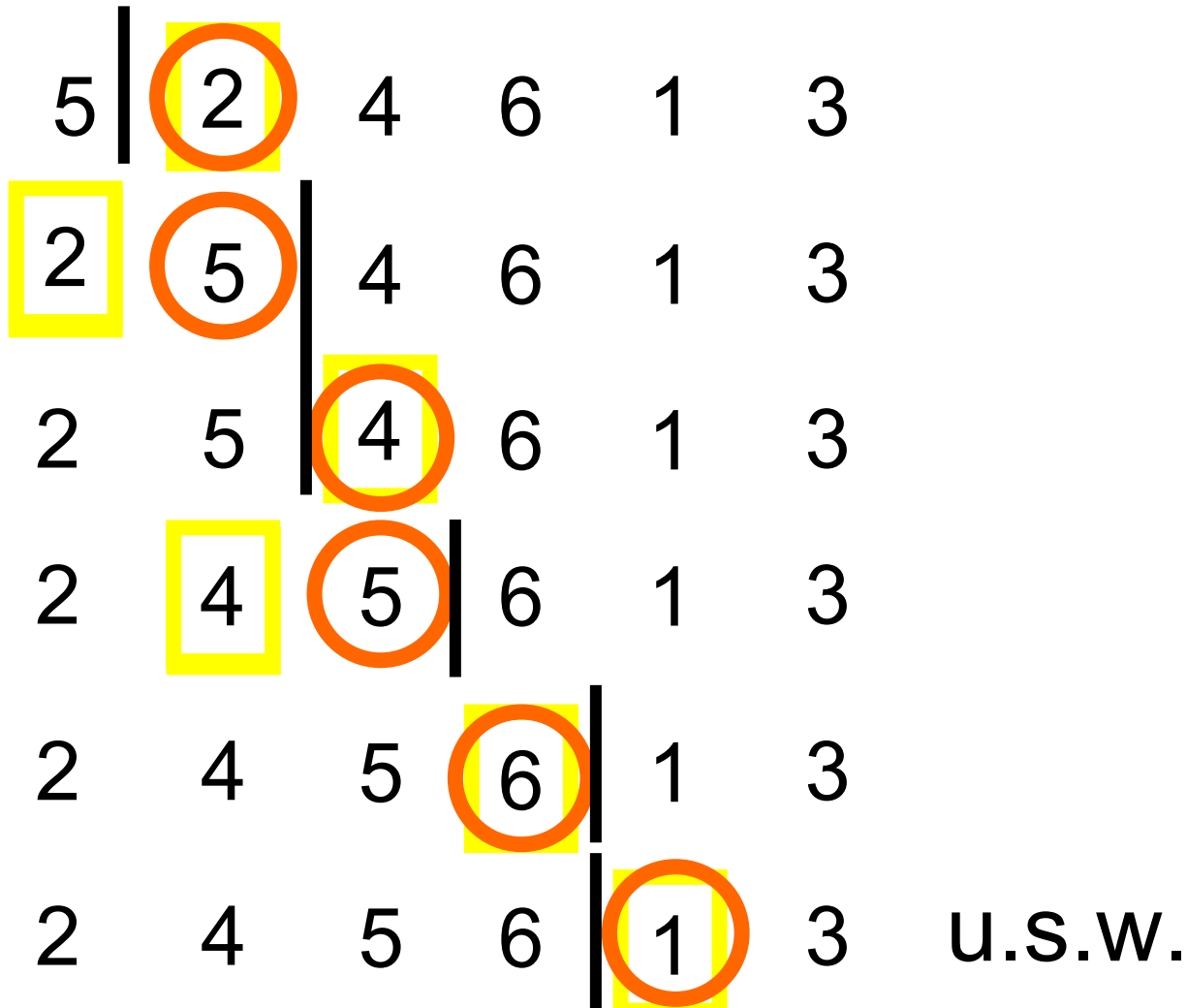
# InsertionSort(ref A)

Eingabe/Ausgabe: Zahlenfolge in Feld  $A[1..n]$

```
(1) for k:=2,...,n {  
(2)   key:=A[k]  
(3)   i:=k  
(4)   while i>1 and A[i-1]>key {  
(5)     A[i]:=A[i-1]  
(6)     i:=i-1  
(7)   }  
(8)   A[i]:=key  
(9) }
```

# Ablauf von InsertionSort(ref A)

Zahlenfolge: k=○ i=□



# Analyse von Algorithmen

JETZT AUFPASSEN: sehr sehr WICHTIG!

Maße für die Effizienz eines Algorithmus:

- Laufzeit 
- benötigter Speicherplatz
- Anzahl der Vergleichsoperationen
- Anzahl der Datenbewegungen

# RAM-Maschinenmodell

Eigenschaften der „Random Access Machine“:

- Es gibt genau einen Prozessor, der das Programm sequentiell abarbeitet
- Jede Zahl, die wir in unserem Programm benutzen paßt in eine Speichereinheit
- Alle Daten liegen in einem direkt zugreifbaren Speicher
- Alle Speicherzugriffe dauern gleich lang
- Alle primitiven Operationen benötigen konstante Zeit

# Primitive Operationen

- Zuweisungen ( $a:=b$ )
- arithmetische Operationen (Addition, Multiplikation, Modulo-Op., Wurzel-Op.)
- logische Operationen (and, or, not)
- Vergleichsoperationen ( $\leq, \geq, \neq$ )
- Befehle zur Ablaufsteuerung (if-then)

Die Laufzeit eines Algorithms ist die Anzahl der bei einer Berechnung durchgeführten primitiven Operationen.

# Laufzeit eines Algorithmus

Laufzeit als Funktion der Eingabegröße

– z.B. für Sortierprobleme:  $n$

Laufzeit abhängig von spezieller Instanz,

– z.B. für Sortierprobleme: sortierte Eingabefolge  
eventuell schneller als für unsortierte Folge

Deswegen: Best-Case, Worst-Case und  
Average-Case Analyse

# Analyse von InsertionSort(ref A)

$s_k$ : Anzahl der Durchführungen von (4)

```
(1) for k:=2,...,n {  
(2)   key:=A[k]  
(3)   i:=k  
(4)   while i>1 and A[i-1]>key {  
(5)     A[i]:=A[i-1]  
(6)     i:=i-1  
(7)   }  
(8)   A[i]:=key  
(9) }
```

Zeit	Wie oft?
$t_1$	n
$t_2$	n-1
$t_3$	n-1
$t_4$	$\sum s_k$
$t_5$	$\sum (s_k - 1)$
$t_6$	$\sum (s_k - 1)$
$t_7$	$\sum (s_k - 1)$
$t_8$	n-1
$t_9$	n-1

# Analyse von InsertionSort(ref A)

$$\begin{aligned}
 T(n) = & t_1 n + t_2(n-1) + t_3(n-1) \\
 & + t_4 \sum_{k=2}^n s_k + t_5 \sum_{k=2}^n (s_k - 1) \\
 & + t_6 \sum_{k=2}^n (s_k - 1) \\
 & + t_7 \sum_{k=2}^n (s_k - 1) \\
 & + t_8(n-1) + t_9(n-1)
 \end{aligned}$$

→ abhängig von  $s_k$

Zeit	Wie oft?
$t_1$	$n$
$t_2$	$n-1$
$t_3$	$n-1$
$t_4$	$\sum s_k$
$t_5$	$\sum (s_k - 1)$
$t_6$	$\sum (s_k - 1)$
$t_7$	$\sum (s_k - 1)$
$t_8$	$n-1$
$t_9$	$n-1$

# Best-Case Analyse

Kürzeste mögliche Laufzeit über alle möglichen Eingabe-Instanzen bei vorgegebener Eingabegröße.

Sortierte Folge:  $s_k=1$

$$\begin{aligned} T(n) &= (t_1 + t_2 + t_3 + t_4 + t_8 + t_9)n \\ &\quad - (t_2 + t_3 + t_4 + t_8 + t_9) \\ &= an + b \end{aligned}$$

$T(n)$ =Lineare Funktion in  $n$  mit Konstanten  $a$  und  $b$

Zeit	Wie oft?
$t_1$	$n$
$t_2$	$n-1$
$t_3$	$n-1$
$t_4$	$\sum s_k$
$t_5$	$\sum (s_k - 1)$
$t_6$	$\sum (s_k - 1)$
$t_7$	$\sum (s_k - 1)$
$t_8$	$n-1$
$t_9$	$n-1$

# Worst-Case Analyse

Längste mögliche Laufzeit über alle möglichen Eingabe-Instanzen bei vorgegebener Eingabegröße.

Umgekehrt sortierte Folge:  $s_k = k$

$$\sum_{k=2}^n (k - 1) = \sum_{k=1}^{n-1} k = \frac{n(n-1)}{2}$$

Daraus folgt  $T(n) = an^2 + bn + c$

$T(n)$  = Quadratische Funktion in  $n$  mit Konstanten  $a$ ,  $b$  und  $c$

Zeit	Wie oft?
$t_1$	$n$
$t_2$	$n-1$
$t_3$	$n-1$
$t_4$	$\sum s_k$
$t_5$	$\sum (s_k - 1)$
$t_6$	$\sum (s_k - 1)$
$t_7$	$\sum (s_k - 1)$
$t_8$	$n-1$
$t_9$	$n-1$

# Average-Case Analyse

Durchschnittliche Laufzeit über alle möglichen Eingabe-Instanzen bei vorgegebener Eingabegröße.

Problem: Was ist eine „durchschnittliche“ Eingabe

Hier:  $s_k = k/2$

$T(n)$ =Quadratische Funktion in  $n$  mit Konstanten  $a$ ,  $b$  und  $c$

# Laufzeit-Analyse

Genauere Laufzeitberechnung ist sehr aufwändig, deswegen Vereinfachung

Wir betrachten nur die Ordnung der Laufzeit, wobei  $n$  als beliebig groß angenommen wird

Wir sagen: „ $f(n)=an+b$  ist in  $\Theta(n)$ “  
und „ $g(n)=an^2+bn+c$  ist in  $\Theta(n^2)$ “

Wir sagen: „ $f(n)$  wächst linear für große  $n$ “  
und „ $g(n)$  wächst quadratisch für große  $n$ “

# Organisatorisches zur Vorlesung

Inhalte der Vorlesung

Literatur zur Vorlesung

Klausur

Organisatorisches zur Übung

# Inhalte der Vorlesung

1. Einführung (O-Notation)
2. Abstrakte Datentypen und Datenstrukturen
3. Sortieralgorithmen
4. Suchalgorithmen (Binärsuche, B- und AVL-Bäume, Skiplisten)
5. Hashing
6. Graphenalgorithmen (BFS, DFS, Zshgskomp, MST, kürzeste Wege)
7. Optimierung (Heuristiken, B&B, Dyn. Prog.)
8. Geometrische Algorithmen

# Literatur zur Vorlesung

- VO-Folien auf Web:

ls11-[www.cs.uni-dortmund.de/lehre/SoSe06/dap2.jap](http://www.cs.uni-dortmund.de/lehre/SoSe06/dap2.jap)

- Skript auf Web (in Raten)

- Bücher:

- R. Sedgewick: Algorithmen, Pearson Studium 2002, 2. Auflage
- T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein: Introduction to Algorithms, MIT Press 2001, 2. Auflage
- T. Ottmann und P. Widmayr: Algorithmen und Datenstrukturen, Spektrum Akademischer Verlag 2001, 4. Auflage

# Klausur

Klausurtermin: Freitag, 21. Juli 2006

Nebentermin: Montag, 9. Oktober 2006

90 Min. , Inhalte der Vorlesung, Übung hilfreich!

Aktuelle Informationen:

[ls11-www.cs.uni-dortmund.de/lehre/SoSe06/dap2.jap](http://ls11-www.cs.uni-dortmund.de/lehre/SoSe06/dap2.jap)

# Organisatorisches zur Übung

→ Markus Chimani