

Diese Vorlesung handelt von Algorithmen.

Was ist ein Algorithmus?

Beispiel: Sortierproblem

Eingabe: Die Zahlenfolge

$$\langle a_1, \dots, a_n \rangle$$

Ausgabe: Eine Permutation (Umordnung)

$$\langle a'_1, \dots, a'_n \rangle$$

derselben Folge mit der Eigenschaft

$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

Zum Beispiel

Eingabe:  $\langle 31, 41, 59, 26, 41, 58 \rangle$

Ausgabe  $\langle 26, 31, 41, 41, 58, 59 \rangle$

Die Eingabe ist eine Instanz des Sortierproblems.

Der Algorithmus ist korrekt, wenn er das

Der Algorithmus ist korrekt, wenn er das Problem für alle Instanzen löst.


Der folgende Algorithmus löst das Problem


Ich erläutere ihn zuerst an einem Beispiel

1	2	3	4	5	6
31	41	59	26	41	58

$j=4$

key = 26

31	41	59		41	58
					

	31	41	59	41	58
	key = 26				

### Insertion sort

for  $j \leftarrow 2$  to  $\text{length}[A]$

do key  $\leftarrow A[j]$

$i \leftarrow j-1$

while  $i > 0 \wedge A[i] > \text{key}$   
do  $A[i+1] \leftarrow A[i]$   
 $i \leftarrow i-1$

$A[i+1] \leftarrow \text{key}.$

Korrektheitsbeweis:

Schleifeninvariante: Zu Beginn der for-Schleife besteht die Teilfolge  $A[1, \dots, i-1]$  aus den Elementen der Original-Teilfolge und ist sortiert.

Der Korrektheitsbeweis ist ein Induktionsbeweis.

Man beweist die Invariante für die Initialisierung.

Die Invariante wird beim Durchlauf der Schleife erhalten, d.h.)  
Man zeigt: wenn die Invariante vor einer Iteration der Schleife korrekt ist, dann auch danach.

Zum Schluss zeigt man: Bei Beendigung der Schleife folgt aus der Invariante die Korrektheit des Algorithmus.

Initialisierung ✓

Erhaltung: erfordert bei formalem Beweis die Einführung

## Abschluss

einer weiteren Variante

Analyse: Wie gut?  
Wie schnell?  
<Wie viel Platz?>  
<Wie viel Code?>

Man benötigt ein Berechnungsmodell.

RAM: Instruktionen werden  
nacheinander abgearbeitet

Auf alle Speicherplätze  
kann gleich schnell zu-  
gegriffen werden.

Laufzeit: Anzahl der Schritte  
Die Ausführung <sup>zeit</sup> einer Zeile  
im Pseudocode ist eine  
Konstante.

Eingabelänge: Anzahl der zu sortieren-  
den Objekte.

Ziel: Laufzeit als Funktion der  
Eingabelänge bestimmen.

Achtung: auch andere Definitionen  
von Laufzeit und Eingabelänge  
möglich.

Auch der Speicherplatz, der benötigt wird, spielt eine wichtige Rolle. Damit nicht.

Insertion sort

for  $j \leftarrow 2$  to  $\text{length}[A]$

do  $\text{key} \leftarrow A[j]$

$i \leftarrow j-1$

while  $i > 0 \wedge A[i] > \text{key}$

$i \leftarrow i-1$   
 $A[i+1] \leftarrow \text{key}$

Kosten #

$C_1$   $n$

$C_2$   $n-1$

$C_4$   $n-1$

$C_5$   $\sum_{j=2}^n t_j$

$C_7$   $\sum_{j=2}^n (t_j - 1)$

$C_8$   $n-1$

$$T(n) = C_1 n + C_2 (n-1) + C_4 (n-1)$$

$$+ C_5 \sum_{j=2}^n t_j$$

$$+ C_6 \sum_{j=2}^n (t_j - 1)$$

$$+ C_7 \sum_{j=2}^n (t_j - 1) + C_8 (n-1)$$

Best case: Folge geordnet:  $\forall j: t_j = 1$

$$T(n) = C_1 n + C_2 (n-1) + C_4 (n-1) + C_5 (n-1) + C_8 (n-1)$$

$$= (c_1 + c_2 + c_4 + c_5 + c_8) n - (c_2 + c_3 + c_5)$$

der Eingabelänge.

Worst case: Folge in umgekehrte Reihenfolge sortiert.

$$t_j = j : j = 2, 3, \dots, n$$

$$\sum_{j=2}^n (j-1) = \frac{n(n-1)}{2} \quad \sum_{j=2}^n j = \frac{n(n+1)}{2}$$

$$T(n) = \left( \frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left( c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) n - (c_2 + c_4 + c_5 + c_8)$$

quadratisch.