

Konventionen Pseudocode

- Blockstruktur wird nur durch Einrücken gekennzeichnet (keine Klammern).
- Schleifenkonstrukte **while** und **repeat** wie üblich.
- Bei **for** bleibt Wert nach Verlassen der Schleife erhalten.
- Alles nach “//” ist Kommentar.
- Mehrfachzuweisung $x \leftarrow y \leftarrow z$ bedeutet $y \leftarrow z; x \leftarrow y$.
- Variablen sind lokal (local).
- Zugriff auf Feldelemente: $A[i]$ das i -te Element.
- Datenattribute z.B. $länge(A)$.
- Parameter einer Prozedur: **call by value**.
- “und” und “oder” sind träge (lazy) Operatoren.



Das Sortierproblem

- **Eingabe:** Eine Folge von n Zahlen $\langle a_1, a_2, \dots, a_n \rangle$.
- **Ausgabe:** Eine Permutation $\langle b_1, b_2, \dots, b_n \rangle$ der Eingabefolge mit $b_1 \leq b_2 \leq \dots \leq b_n$.



Insertion-Sort

```
0  INSERTION-SORT(A)
1  for  $j \leftarrow 2$  to  $\text{länge}[A]$ 
2  do  $\text{schlüssel} \leftarrow A[j]$ 
3  // Fuege  $A[j]$  in die sortierte Sequenz  $A[1..j-1]$  ein.
4   $i \leftarrow j-1$ 
5  while  $i > 0$  und  $A[i] > \text{schlüssel}$ 
6  do  $A[i+1] \leftarrow A[i]$ 
7   $i \leftarrow i-1$ 
8   $A[i+1] \leftarrow \text{schlüssel}$ 
```



Analyse von Algorithmen

- Obere asymptotische Schranke

$$O(g(n)) = \{f(n) \mid \text{es gibt } c, n_0 > 0 \text{ mit } 0 \leq f(n) \leq cg(n) \\ \text{für alle } n \geq n_0\}$$

- Untere asymptotische Schranke

$$\Omega(g(n)) = \{f(n) \mid \text{es gibt } c, n_0 > 0 \text{ mit } 0 \leq cg(n) \leq f(n) \\ \text{für alle } n \geq n_0\}$$

- Asymptotisch scharfe Schranke

$$\Theta(g(n)) = \{f(n) \mid \text{es gibt } c_1, c_2, n_0 > 0 \text{ mit} \\ 0 \leq c_1g(n) \leq f(n) \leq c_2g(n) \text{ für alle } n \geq n_0\}$$



Analyse von Insertion-Sort

0	<i>INSERTION-SORT(A)</i>	<i>Kosten</i>	<i>Zeit</i>
1	for $j \leftarrow 2$ to $\text{länge}[A]$	c_1	n
2	do $\text{schlüssel} \leftarrow A[j]$	c_2	$n - 1$
3	// setze $A[j]$ ein ...	0	$n - 1$
4	$i \leftarrow j - 1$	c_4	$n - 1$
5	while $i > 0$ und $A[i] > \text{schlüssel}$	c_5	$\sum_{j=2}^n t_j$
6	do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^n (t_j - 1)$
7	$i \leftarrow i - 1$	c_7	$\sum_{j=2}^n (t_j - 1)$
8	$A[i+1] \leftarrow \text{schlüssel}$	c_8	$n - 1$



Aufwandsanalyse

Durch Summieren der Produkte aus Kosten und Zeit:

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5 \sum_{j=2}^n t_j +$$

$$c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n - 1)$$

- Günstigster Fall: Das Feld ist schon sortiert

$$\begin{aligned} T(n) &= c_1n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1) \\ &= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_6) \end{aligned}$$

↪ lineare Laufzeit



Aufwandsanalyse “worst case”

- Schlechtester Fall: Das Feld ist in umgekehrter Reihenfolge sortiert

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5 \left(\frac{n(n+1)}{2} - 1 \right) \\ &\quad + c_6 \left(\frac{n(n+1)}{2} - 1 \right) + c_7 \left(\frac{n(n+1)}{2} - 1 \right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2} \right) n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8 \right) \\ &\quad - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

↪ quadratische Laufzeit



Analyse II

Im Folgenden werden wir meistens nur die Laufzeit im schlechtesten Fall analysieren, denn

- der schlechteste Fall bietet eine obere Schranke für die maximale Laufzeit,
- für einige Algorithmen tritt der schlechteste Fall häufig auf: Z.B. Suche in einer Datenbank,
- der “mittlere Fall” ist oft annähernd genauso schlecht wie der schlechteste Fall.



Methode: Teile und Beherrsche

- **Teile** das Problem in eine Anzahl von Teilproblemen auf
- **Beherrsche** die Teilprobleme durch rekursives Lösen bis sie so klein sind, daß sie direkt gelöst werden können.
- **Verbinde** die Lösungen der Teilprobleme zur Lösung des Ausgangsproblems.



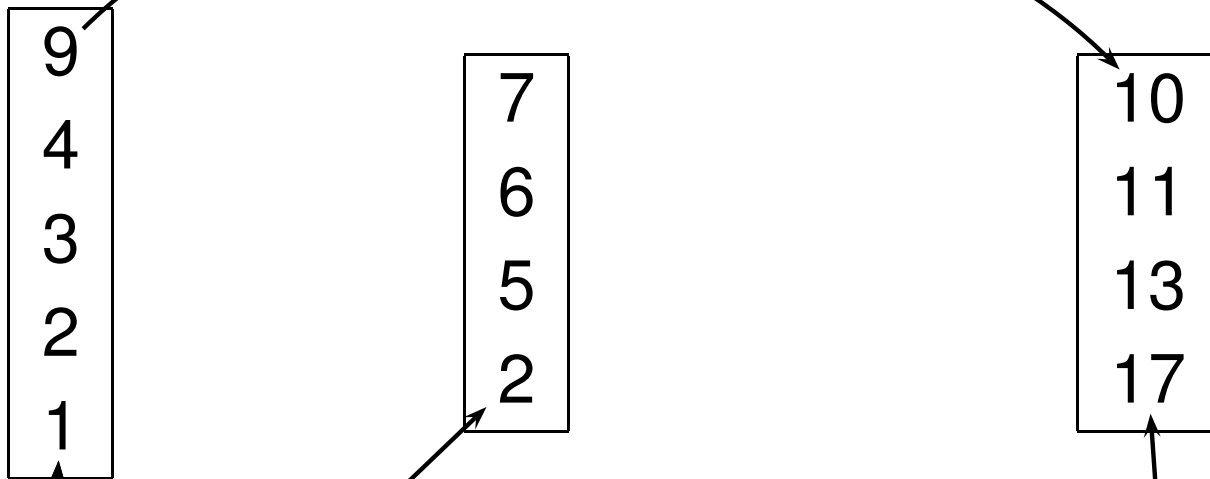
Beispiel: Merge-Sort

- **Teile** die zu sortierende Sequenz der Länge n in zwei Teilsequenzen der Länge $\frac{n}{2}$.
- **Beherrsche** durch rekursives Anwenden von Merge-Sort auf die zwei Teilsequenzen.
- **Verbinde** die zwei Teilsequenzen durch Mischen (merge).



Merge

$$9 \geq 7$$



Vorsortierte Sequenzen

Ergebnis-Sequenz



Pseudocode Merge

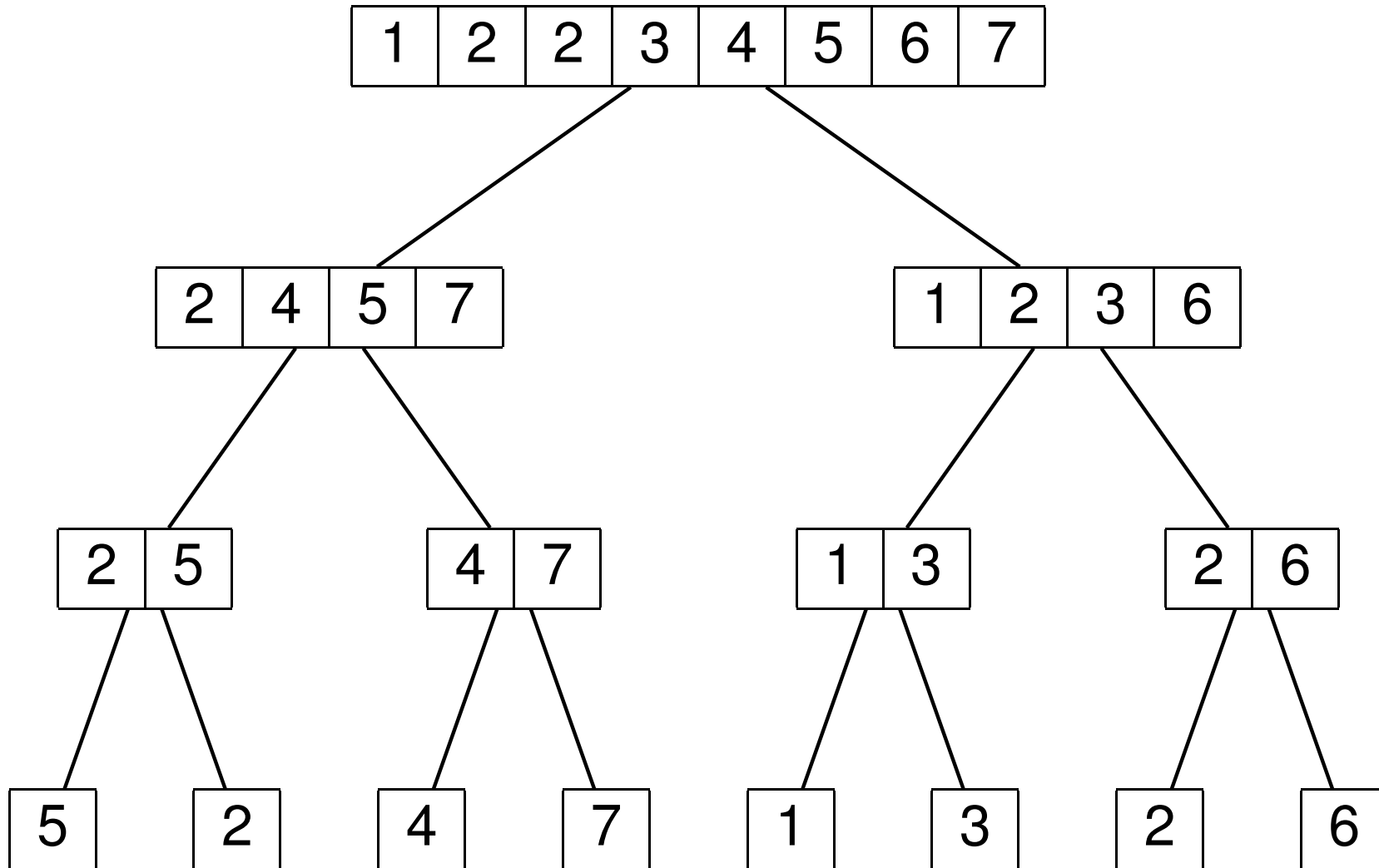
```
0 MERGE(A, p, q, r)
1    $n_1 \leftarrow q - p + 1$ 
2    $n_2 \leftarrow r - q$ 
3   erzeuge die Felder  $L[1..n_1 + 1]$  und  $R[1..n_2 + 1]$ 
4   for  $i \leftarrow 1$  to  $n_1$  do  $L[i] \leftarrow A[p + i - 1]$ 
5   for  $j \leftarrow 1$  to  $n_2$  do  $R[j] \leftarrow A[q + j]$ 
6    $L[n_1 + 1] \leftarrow R[n_2 + 1] \leftarrow \infty$  // Wächter
7    $i \leftarrow j \leftarrow 1$ 
8   for  $k \leftarrow p$  to  $r$ 
9     do if  $L[i] \leq R[j]$ 
10        then  $A[k] \leftarrow L[i]$ 
11             $i \leftarrow i + 1$ 
12        else  $A[k] \leftarrow R[j]$ 
13             $j \leftarrow j + 1$ 
```

Pseudocode Merge-Sort

```
0  MERGE-SORT( $A, p, r$ )
1  if  $p < r$ 
2      then  $q \leftarrow \lfloor (p + r) / 2 \rfloor$ 
3          MERGE-SORT( $A, p, q$ )
4          MERGE-SORT( $A, q+1, r$ )
5          MERGE( $A, p, q, r$ )
```



Beispiel



Laufzeitanalyse

- Im allgemeinen Teile- und Beherrsche-Fall gilt: Sei $T(N)$ die Laufzeit für ein Problem der Größe n . Ist n hinreichend klein $n \leq c$, dann benötigt die direkte Lösung eine konstante Zeit $\Theta(1)$. Führt die Aufteilung des Problems zu a Teilproblemen der Größe $1/b$ und braucht die Aufteilung $D(n)$ Zeit und das Verbinden zum ursprünglichen Problem die Zeit $C(n)$ so gilt:

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n \leq c \\ a(T(n/b)) + D(n) + C(n) & \text{sonst} \end{cases}$$

- Im Fall von Merge-Sort ist $a = b = 2$ und $c = 1$, also

$$T(n) = \begin{cases} \Theta(1) & \text{falls } n = 1 \\ 2(T(n/2)) + dn & \text{sonst} \end{cases}$$

Laufzeitanalyse 2

- Man kann die Problemgröße nur $\log_2(n)$ oft aufteilen.
- Beim i -ten Aufteilen hat man 2^i Teillisten der Größe $n/2^i$ zu lösen und benötigt dafür dn Zeit
- Somit braucht man insgesamt $dn \log_2 n + dn$ Zeit.

