

## Kap. 3: Sortieren (2)



Professor Dr. Petra Mutzel  
Lehrstuhl für Algorithm Engineering, LS11  
Fakultät für Informatik, TU Dortmund

5. VO DAP2 SS 2009 28. April 2009

## Motivation

„Warum soll ich hier bleiben?“

Wir analysieren Merge-Sort

„Was ist daran denn besonders?“

Wir lösen eine Rekursionsgleichung

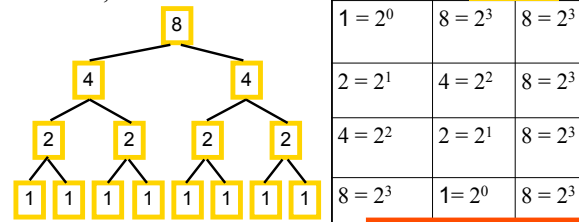
## Überblick

- Analyse von Merge-Sort
- Eigenschaften von Merge-Sort

- Quick-Sort
- Analyse von Quick-Sort

## Herleitung der Laufzeitfunktion (Abschätzung)

Sei  $n=2^k$  für ein beliebiges  $k$   
hier:  $n=8, k=3$ :



Anzahl der Instanzen	Zeit pro Instanz	Gesamtzeit
1 = 2 <sup>0</sup>	8 = 2 <sup>3</sup>	8 = 2 <sup>3</sup>
2 = 2 <sup>1</sup>	4 = 2 <sup>2</sup>	8 = 2 <sup>3</sup>
4 = 2 <sup>2</sup>	2 = 2 <sup>1</sup>	8 = 2 <sup>3</sup>
8 = 2 <sup>3</sup>	1 = 2 <sup>0</sup>	8 = 2 <sup>3</sup>

Aufwand in jeder Stufe ca.  $n=2^k$ .

Es gibt  $k+1=\log n + 1$  solcher Stufen

Gesamtaufwand:  
 $T(n) = n(1 + \log n) = \Theta(n \log n)$

## Worst-Case Analyse von MergeSort

- **Teile:**  $\Theta(c)$ ,  $c$  konstant **jetzt formal**
- **Eroberer:** Lösen zweier Teilprobleme der Größe  $\lceil n/2 \rceil$  bzw.  $\lfloor n/2 \rfloor$ :  $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$ .
- **Kombiniere:** Merge() kostet  $\Theta(n)$

Rekursionsgleichung der Laufzeitfunktion:

$$T(n) = \begin{cases} \Theta(1), & \text{für } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + \Theta(n), & \text{für } n > 1 \end{cases}$$

**Lösung:**  $T(n) = \Theta(n \log n)$

## Worst-Case MergeSort: Beweis

- Wir zeigen:  $T(n) = O(n \log n)$
- Aus der Rekursionsgleichung folgt: Es existiert ein  $a > 0$ , so dass gilt

$$T(n) \leq \begin{cases} a, & \text{für } n = 1 \\ T(\lceil \frac{n}{2} \rceil) + T(\lfloor \frac{n}{2} \rfloor) + an, & \text{für } n > 1 \end{cases}$$

Wir zeigen mit Induktion, dass mit  $c=3a$  für alle  $n \geq 3$  gilt:  $T(n) \leq cn \log(n-1)$

### Induktionsanfang:

- $n=3$ :  $T(3) \leq T(1) + T(2) + 3a$   
 $\leq T(1) + 2T(1) + 2a + 3a$   
 $\leq 3T(1) + 5a$   
 $\leq 8a = \frac{8}{3}c$   
 $\leq 3c = c3 \log(3 - 1)$
- $n=4$ :  $T(4) \leq T(2) + T(2) + 4a$   
 $\leq 4T(1) + 8a$   
 $\leq 12a$   
 $= 4c \leq c4 \log(4 - 1)$

### Induktionsanfang:

- $n=5$ :  
 $T(5) \leq T(2) + T(3) + 5a$   
 $\leq 2T(1) + 2a + T(2) + T(1) + 3a + 5a$   
 $\leq 3T(1) + 10a + 2T(1) + 2a$   
 $= 5T(1) + 12a$   
 $\leq 17a = \frac{17}{3}c$   
 $\leq 10c = c5 \log(5 - 1)$

### Induktionsschluss

- Ann: gilt für alle Instanzen kleiner als  $n$ :
- $n \geq 6$ :

$$T(n) \leq T\left(\left\lceil \frac{n}{2} \right\rceil\right) + T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + an$$

$$\leq c \left\lceil \frac{n}{2} \right\rceil \log\left(\left\lceil \frac{n}{2} \right\rceil - 1\right) + c \left\lfloor \frac{n}{2} \right\rfloor \log\left(\left\lfloor \frac{n}{2} \right\rfloor - 1\right) + an$$

$$\leq c \frac{n+1}{2} \log\left(\frac{n+1}{2} - 1\right) + c \frac{n}{2} \log\left(\frac{n}{2} - 1\right) + an$$

$$\leq c \frac{n+1}{2} \log\left(\frac{n-1}{2}\right) + c \frac{n}{2} \log\left(\frac{n-2}{2}\right) + an$$

$$= \frac{1}{2}c(n+1) \log(n-1) + \frac{1}{2}cn \log(n-2) - c \frac{n+1}{2} - c \frac{n}{2} + an$$

$\log \frac{Z}{N} = \log Z - \log N$

### Induktionsschluss ff

$$= \frac{1}{2}c(n+1) \log(n-1) + \frac{1}{2}cn \log(n-2) - c \frac{n+1}{2} - c \frac{n}{2} + an$$

$$\leq \frac{1}{2}cn \log(n-1) + \frac{1}{2}c \log(n-1) + \frac{1}{2}cn \log(n-1) - cn - \frac{c}{2} + an$$

$$= cn \log(n-1) + \frac{c}{2} \log(n-1) - cn - \frac{c}{2} + \frac{c}{3}n$$

$$\leq cn \log(n-1) - \frac{c}{2}(2n+1 - \log(n-1)) + \frac{c}{2}n$$

$$\leq cn \log(n-1) - \frac{c}{2}n + \frac{c}{2}n$$

$$= cn \log(n-1) \text{ 😊}$$

• zu zeigen:  $T(n) = \Omega(n \log n)$  → zuhause

### Worst-Case Analyse von MergeSort

- Anzahl der Schlüsselvergleiche (Z. 4):

$$C_{\text{best}}(n) = C_{\text{avg}}(n) = C_{\text{worst}}(n) = \Theta(n \log n)$$

- Anzahl der Datenbewegungen (Z. 6,8,10):

$$M_{\text{best}}(n) = M_{\text{avg}}(n) = M_{\text{worst}}(n) = \Theta(n \log n)$$

### Eigenschaften von MergeSort

- Eigenschaften:

- in situ ? ⚡
- adaptiv ? ⚡
- stabil ? 😊

- Besonderheit:

- MergeSort verarbeitet Daten sequentiell
- deshalb sind verkettete Listen gut geeignet
- gutes externes Verfahren

### 3.1.4 Quick-Sort

Idee folgt dem „Divide and Conquer“-Prinzip:

- **Teile:** Wähle Pivotelement  $k$  von  $A$ , teile  $A$  ohne  $k$  in Teilfolgen  $A_1$  und  $A_2$  mit
  - $A_1$  enthält nur Elemente  $\leq k$
  - $A_2$  enthält nur Elemente  $\geq k$
- **Erobere:** QuickSort( $A_1$ ); QuickSort( $A_2$ ); danach sind  $A_1$  und  $A_2$  sortiert
- **Kombiniere:** Bilde  $A$  durch Hintereinanderfügen in der Reihenfolge  $A_1, k, A_2$

1960 von C.A.R. Hoare entwickelt

### QuickSort(ref A,l,r)

- (1) **procedure** QuickSort(ref A,l,r)
- (2) **var** Index p
- (3) **if**  $l < r$  **then** {
- (4) p = Partition(A,l,r)
- (5) QuickSort(A,l,p-1)
- (6) QuickSort(A,p+1,r)
- (7) }

Aufruf: QuickSort(A,1,n)

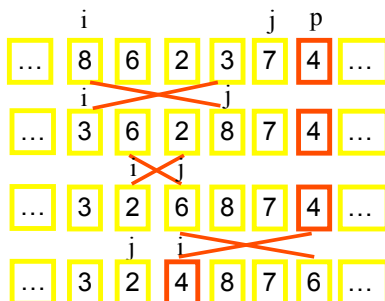
### Partition(ref A,l,r)

- (1) **function** Partition(ref A,l,r)
- (2) **var** Indizes i,j; Schlüsselwert x
- (3)  $x := A[r].key$
- (4)  $i := l - 1; j := r$
- (5) **repeat**
- (6) **repeat**  $i := i + 1$  **until**  $A[i].key \geq x$
- (7) **repeat**  $j := j - 1$  **until**  $j < l$  or  $A[j].key < x$
- (8) **if**  $i < j$  **then** vertausche  $A[i]$  und  $A[j]$
- (9) **until**  $i \geq j$
- (10) vertausche  $A[i]$  und  $A[r]$
- (11) **return** i

### Korrektheit

- QuickSort ist offensichtlich korrekt, wenn er terminiert.
- **Terminierung von Partition:**
  - **Zeile 6:** klar wegen Wahl des Pivotelements ganz rechts
  - **Zeile 7:** bricht spätestens ab, wenn der linke Rand überlaufen wird.

### Ablauf von QuickSort

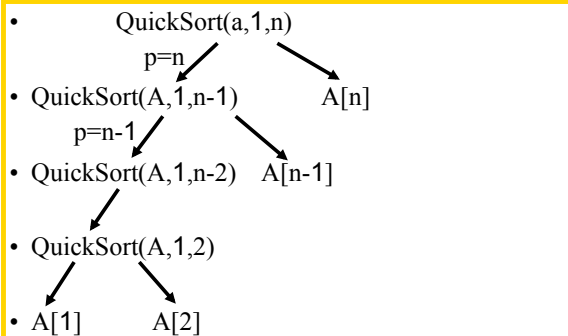


### Analyse von QuickSort

- **Teile:** Partition() kostet  $\Theta(n)$
- **Erobere:** Lösen zweier Teilprobleme der Größe  $|A_1|$  bzw.  $|A_2|$
- **Kombiniere:** konstant

Worst Case Fall?

## Aufrufbaum für sortierte Folge



## Analyse von QuickSort: Worst-Case

**Worst-Case:** z.B. falls A bereits sortiert ist:

- Aufrufbaum hat lineare Tiefe
- Partitionierung: n Vergleiche über Index i und 1 Vergleich über Index j beim ersten Durchgang, n-1 Vergleiche über i beim zweiten Durchgang,...

$$C_{worst}(n) = \sum_{k=2}^n (k+1) = \sum_{k=3}^{n+1} k = \frac{(n+1)(n+2)}{2} - 3 = \Theta(n^2)$$

- **Anzahl der Datenbewegungen:**

In diesem Fall:  $M(n)=\Theta(n)$  und im schlechtesten Fall:  $M_{worst}(n)=\Theta(n \log n)$  (ohne Beweis)

## Analyse von QuickSort: Best-Case

**Best-Case:** die beiden Folgen  $A_1$  und  $A_2$  haben immer ungefähr gleiche Länge

- Aufrufbaum hat Tiefe  $\Theta(\log n)$  (s. MergeSort)
- Partitionierung: n Vergleiche per Stufe

$$C_{best}(n) = \Theta(n \log n)$$

- **Anzahl der Datenbewegungen:**

In diesem Fall:  $M(n)=\Theta(n \log n)$  und im besten Fall:  $M_{best}(n)=\Theta(n)$