

# Kapitel 3: Sortieren ff

Professor Dr. Petra Mutzel  
 Lehrstuhl für Algorithm Engineering, LS11

5. VO

18. April 2006

## Überblick

• Analyse von MergeSort

• Quick-Sort

## Motivation

„Warum soll ich hier bleiben?“

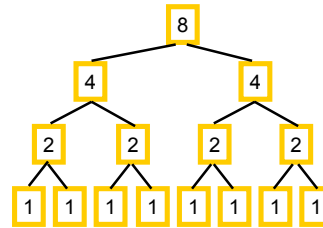
Wir lernen Quick-Sort

„Was ist daran denn besonders?“

Quick-Sort gehört zu den „Top 10“ wichtigsten Algorithmen

## Herleitung der Laufzeitfunktion

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



Anzahl der Instanzen	Zeit pro Instanz	Gesamtzeit
$1 = 2^0$	$8 = 2^3$	$8 = 2^3$
$2 = 2^1$	$4 = 2^2$	$8 = 2^3$
$4 = 2^2$	$2 = 2^1$	$8 = 2^3$
$8 = 2^3$	$1 = 2^0$	$8 = 2^3$

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

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

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

## Worst-Case Analyse von MergeSort

- **Teile:**  $\Theta(c)$ ,  $c$  konstant
- **Erobere:** 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:

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

10

### Induktionsanfang:

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

11

### Induktionsschluss

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

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

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

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

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

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

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

12

### Induktionsschluss ff

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

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

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

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

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

$$= cn \log(n-1) \text{ 😊}$$

- zu zeigen:  $T(n) = \Omega(n \log n) \rightarrow$  zuhause

13

### Worst-Case Analyse von MergeSort

- Anzahl der Schlüsselvergleiche (Z. 3):

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




- Anzahl der Datenbewegungen (Z. 5+7):

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

14

### Eigenschaften von MergeSort

- Eigenschaften:

- in situ ? 
- adaptiv ? 
- stabil ? 

- Besonderheit:

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

15

**• Wie Computer sind auch Algorithmen Technologie!**

	Algorithmus	Implementierung	Geschwindigkeit: ops/sec
Super-computer	InsertionSort	$2n^2$	1000 Mio
langsamer Computer	MergeSort	$50n \log n$	10 Mio

- Sortieren von 1 Mio. Zahlen:
  - InsertSort:  $2(10^6)^2 \text{ops} / 10^9 \text{ops/sec} = 2000 \text{ sec} \approx 33 \text{ min}$
  - MergeSort:  $50 \cdot 10^6 \log 10^6 \text{ops} / 10^7 \text{ops/sec} \approx 100 \text{ sec}$
  - gleiche Rechner: 33 min vs. 1 sec

### 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)

**procedure** QuickSort(ref A,l,r)

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

Aufruf: QuickSort(A,1,n)

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

**procedure** Partition(ref A,l,r)

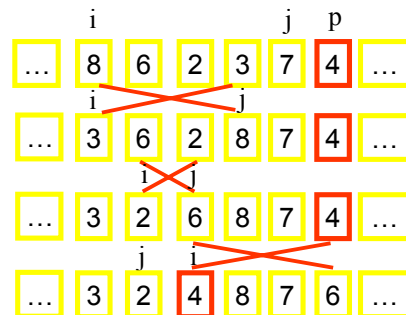
- (1) **var** Indizes i,j; Schlüsselwert x
- (2)  $x := A[r].\text{key}$
- (3)  $i := l-1; j := r$
- (4) **repeat**
- (5)   **repeat**  $i := i+1$  **until**  $A[i].\text{key} \geq x$
- (6)   **repeat**  $j := j-1$  **until**  $A[j].\text{key} \leq x$  or  $j < l$
- (7)   **if**  $i < j$  **then** vertausche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

#### Korrektheit

- QuickSort ist offensichtlich korrekt, wenn er terminiert.
- Terminierung von Partition:**
  - **Zeile 5:** klar wegen Wahl des Pivotelements ganz rechts
  - **Zeile 6:** bricht spätestens ab, wenn der linke Rand überlaufen wird.

20

#### Ablauf von QuickSort



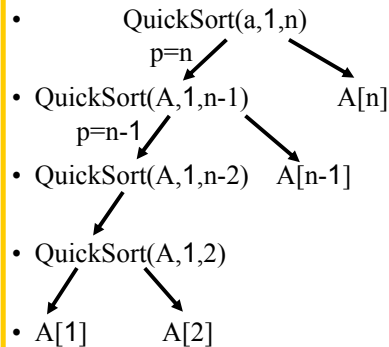
21

## Analyse von QuickSort

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

22

## Aufrufbaum für sortierte Folge



23

## 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

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

- **Anzahl der Datenbewegungen:**

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

24

## 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

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



- **Anzahl der Datenbewegungen:**

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

25

## Analyse von QuickSort: Average

- Grundannahmen
  - alle Schlüssel sind verschieden
  - o.B.d.A.  $A[j] \in \{1, 2, \dots, n\}$
  - alle Permutationen sind gleichwahrscheinlich

-  Jede Zahl  $k \in \{1, 2, \dots, n\}$  tritt mit gleicher Wahrscheinlichkeit  $1/n$  an Position n auf
- Pivotelement k erzeugt zwei Folgen der Längen k-1 und n-k
- Diese Folgen sind wieder zufällig:  gilt

## Analyse von QuickSort: Average

- Teilt man sämtliche Folgen mit n Elementen mit dem Pivotelement, so erhält man sämtliche Folgen der Länge k-1 und n-k.

Rekursionsgleichung der Laufzeitfunktionen:

$$T(n) \leq \begin{cases} a, & \text{für } n = 1 \\ \frac{1}{n} \sum_{k=1}^n (T(k-1) + T(n-k)) + bn & \text{für } n \geq 2 \end{cases}$$

wobei die Summe über alle n möglichen Aufteilungen läuft und  $bn$  der Aufteilungsaufwand für eine Folge der Länge n ist.

26

## Analyse von QuickSort: Average

Einsetzen von  $T(0)=0$ :

$$T(n) \leq \begin{cases} a, & \text{für } n = 1 \\ \frac{2}{n} \sum_{k=1}^{n-1} T(k) + bn & \text{für } n \geq 2 \end{cases}$$

Lösung der Rekursionsgleichung:

$$T(n) = \Theta(n \log n)$$

28

## Beweis von Average-Case

- Wir zeigen:  $T(n) = O(n \log n)$
- $T(n) = \Omega(n \log n)$  ähnlich (o.Bw.)

Wir zeigen durch vollständige Induktion:

es existieren  $c, n_0 > 0$ , so dass für alle  $n \geq n_0$ :

$$T(n) \leq cn \log n$$

Wir wählen:  $n_0 = 2$ .

Induktionsanfang:  $n=2$ :  $T(2) = T(1) + bn = a + 2b$

somit ist  $T(2) \leq c \cdot 2 \log 2$  g.d.w.  $c \geq (a+2b)/2 = a/2 + b$

## Beweis von Average-Case ff

Induktionsschluss: Sei nun  $n \geq 3$ :

$$\begin{aligned} T(n) &\leq \frac{2}{n} \sum_{k=1}^{n-1} T(k) + bn \\ &\leq \frac{2}{n} \sum_{k=1}^{n-1} (ck \log k) + bn \\ &\leq \dots \leq cn \log n - \frac{c}{4}n - \frac{c}{2} + bn \\ &\leq cn \log n \end{aligned}$$

↑  
gilt, wenn z.B.  $c=4b$

Wähle insgesamt  $c = \max \{a/2 + b, 4b\}$

30

## Eigenschaften von QuickSort

- **Anzahl der Schlüsselvergleiche:**

$$C_{\text{best}}(n) = C_{\text{avg}}(n) = \Theta(n \log n); \quad C_{\text{worst}}(n) = \Theta(n^2)$$

- **Eigenschaften:**

- in situ? ⚡  $\Theta(n)$  zusätzlichen Speicher, falls nicht-rekursiv:  $\Theta(\log n)$
- adaptiv? ⚡
- stabil? ⚡

- es exist. viele Varianten (z.B. zufällig)
- sehr gut in der Praxis