

## Kapitel 3: Sortieren

Professor Dr. Petra Mutzel  
Lehrstuhl für Algorithm Engineering, LS11

4. VO

13. April 2006

## Überblick

- ADT Dictionary
- Einführung in das Sortierproblem
- Insertion-Sort
- Selection-Sort
- Merge-Sort

2

## Motivation

„Warum soll ich hier bleiben?“  
Sortierverfahren sind **WICHTIG!!!**

„Ich kann doch schon sortieren.“  
**ABER ES GEHT SCHNELLER!**

3

## Der ADT Dictionary

„Dictionary“: Wörterbuch

- **Wertebereich:**  $D \subseteq K \times V$ , wobei  $K$  Schlüssel (key) bezeichnet und  $V$  die Werte. Dabei ist jedem  $k \in K$  höchstens ein  $v \in V$  zugeordnet.

- **Operationen:**
  - Einfügen, Entfernen, Suchen (s. nächste Folie)
  - Im folgenden sei  $Q$  ein Dictionary vor Anwendung der Operationen.

4

## Operationen des ADT Dictionary

- **INSERT( $K$   $k$ ,  $V$   $v$ )**
  - Falls  $k$  nicht schon in  $D$  ist, dann wird ein neuer Schlüssel  $k$  mit Wert  $v$  in  $D$  eingefügt, andernfalls wird der Wert des Schlüssels  $k$  auf  $v$  geändert.

$\text{INSERT}(k,v)$ : Falls  $k$  neu:  $D := D \cup (k,v)$ , sonst  
 $D := D \setminus (k,v') \cup (k,v)$

- **DELETE( $K$   $k$ )**
  - Entfernt Schlüssel  $k$  mit Wert aus  $D$  (falls  $k$  in  $D$ )

- **SEARCH( $K$   $k$ ):  $V$** 
  - Gibt den bei Schlüssel  $k$  gespeicherten Wert zurück (falls  $k$  in  $D$ )

5

## Der ADT Dictionary

- **Wörterbuchproblem:** Finde eine Datenstruktur mit möglichst effizienten Implementierungen für die Operationen eines Dictionary.

- **Naive Lösung:** als Paar von Feldern:
  - Lineare Laufzeit für **alle Operationen**

- **Im Laufe der Vorlesung:** einige weitaus bessere Verfahren!

6

## Kapitel 3: Sortieren

7

## „Unser“ Sortierproblem

**Eingabe:** Folge von Datensätzen  $s_1, s_2, \dots, s_n$  mit Schlüsseln  $k_1, k_2, \dots, k_n$ , auf denen eine Ordnungsrelation „ $\leq$ “ definiert ist.

**Ausgabe:** Permutation  $\pi : \{1, 2, \dots, n\} \rightarrow \{\pi(1), \pi(2), \dots, \pi(n)\}$ , so dass die Umordnung der Datensätze gemäß  $\pi$  die Schlüssel in aufsteigende Reihenfolge bringt:  $k_{\pi(1)} \leq k_{\pi(2)} \leq \dots \leq k_{\pi(n)}$

Speicherung in Feld:  $A[1], \dots, A[n]$

Ansprechbar: Schlüssel:  $A[i].key$

Informationsfeld:  $A[i].info$

8

## Laufzeitmessung

- Anzahl der durchgeführten Schlüsselvergleiche („Comparisons“) für Best-Case, Worst-Case und Average-Case:

•  $C_{best}(n), C_{worst}(n), C_{avg}(n)$

- Anzahl der durchgeführten Bewegungen („Movements“) von Datensätzen für Best-Case, Worst-Case und Average-Case:

•  $M_{best}(n), M_{worst}(n), M_{avg}(n)$

9

## Eigenschaften von Sortierverfahren

- **Intern/Extern:** Geht das Verfahren davon aus, dass alle Daten im Hauptspeicher sind, dann  $\rightarrow$  **intern**
- Manchmal müssen Daten aus Platzgründen ausgelagert werden (Platte); Verfahren, die hier gut geeignet sind  $\rightarrow$  **extern**
- **In situ:** Benötigt ein Sortieralgorithmus zusätzlich zur Eingabe höchstens konstant viel zusätzlichen Speicherplatz, dann  $\rightarrow$  **in situ**

10

## Eigenschaften von Sortierverfahren

- **Adaptiv:** Laufzeit abhängig von dem Grad der Vorsortierung der Daten; falls besser für vorsortierte Daten, dann  $\rightarrow$  **adaptiv**

- **Stabil:** gleiche Reihenfolge von Datensätzen mit gleichem Schlüssel vor und nach dem Sortieren, dann  $\rightarrow$  **stabil**

11

## 3.1 Allgemeine Sortierverfahren

- **Voraussetzung:** je zwei Schlüssel  $k_i$  und  $k_j$  sind vergleichbar, also entweder gilt  $k_i \leq k_j$  oder  $k_j \leq k_i$ .

12

### 3.1.1 Insertion-Sort / Analyse

- Anzahl der Schlüsselvergleiche:

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

13

### 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) }
```

14

### 3.1.1 Insertion-Sort / Analyse

- Anzahl der Schlüsselvergleiche:

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

- Anzahl der Datenbewegungen:

$$M_{\text{best}}(n) = \Theta(n) \text{ und } M_{\text{avg}}(n) = M_{\text{worst}}(n) = \Theta(n^2)$$

- Eigenschaften:

- in situ? 😊
- adaptiv? 😊
- stabil? 😊

15

### Inversionen

In einer Permutation  $\pi = \{\pi_1, \pi_2, \dots, \pi_n\}$  heißt ein Paar  $(\pi_i, \pi_j)$  eine Inversion, wenn gilt:  $i < j$  und  $\pi_i > \pi_j$ .

- Die Anzahl der Inversionen einer Folge  $\pi$  heißt Inversionszahl und ist ein Maß für die Vorsortierung einer Folge.

- Es gilt: Eine Folge ist sortiert g.d.w. die Anzahl ihrer Inversionen gleich 0 ist.
- Im schlimmsten Fall besitzt eine Folge  $\sum(N-i) = \Theta(n^2)$  viele Inversionen.

16

### Inversionen

- Die Anzahl der Schlüsselvergleiche und Datenbewegungen in InsertionSort hängt eng mit der Anzahl der Inversionen der Folge zusammen:

- Die Anzahl der Inversionen der Folge ist gleich  $\sum_{k=2..n} (s_k - 1)$ .

17

### 3.1.2 Selection-Sort

Idee von „Sortieren durch Auswahl“:

- Bestimme Position  $i_1 \in \{1, 2, \dots, n\}$  zu der das Element mit minimalem Schlüssel auftritt; vertausche  $A[1]$  mit  $A[i_1]$ ;
- Bestimme  $i_2 \in \{2, \dots, n\}$  ..., vertausche  $A[2]$  mit  $A[i_2]$ ; etc.

Beispiel: s. VO und Skript

18

### SelectionSort(ref A)

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

```

(1) for j:=1,2,...,n-1 {
(2)   minpos:=j
(3)   for i:=j+1,...,n {
(4)     if A[i].key < A[minpos].key then
(5)       minpos:=i
(6)   }
(7)   if minpos > j then
(8)     Vertausche A[minpos] mit A[j]
(9) }
    
```

1

### Analyse von SelectionSort

- **Anzahl der Schlüsselvergleiche (Z. 4):**

$$C_{best}(n) = C_{avg}(n) = C_{worst}(n) = \Theta(n^2)$$

denn:

$$C_{best}(n) = \sum_{j=1}^{n-1} (n-j) = \sum_{j=1}^{n-1} j = \frac{n(n-1)}{2} = \theta(n^2)$$

- **Anzahl der Datenbewegungen (Z. 8):**

$$M_{best}(n) = 0$$

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

20

### Eigenschaften von SelectionSort

- **Eigenschaften:**

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

- **Einsatz von SelectionSort, wenn:**

- Bewegungen von Datensätzen teuer
- Vergleiche zwischen Schlüsseln billig

21

### 3.1.3 Merge-Sort

Idee folgt dem „Divide and Conquer“-Prinzip („Teile und Eroberer“):

- **Teile** das Problem in Teilprobleme auf.
- **Eroberer** die Teilprobleme durch rekursives Lösen. Wenn sie klein genug sind, löse sie direkt.
- **Kombiniere** die Lösungen der Teilprobleme zu einer Lösung des Gesamtproblems.

wichtiges Algorithmen-Entwurfsprinzip!

22

### 3.1.3 Merge-Sort

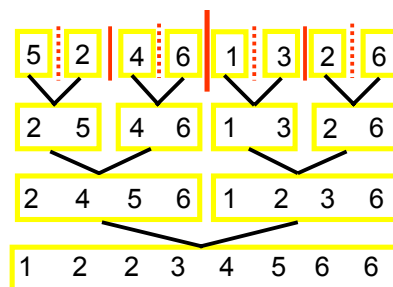
Idee von „Sortieren durch Mischen“:

- **Teile:** Teile die Folge in der Mitte in zwei Teilfolgen.
- **Eroberer:** Sortiere beide Teilfolgen rekursiv. Für 1-elementige Teilfolgen ist nichts zu tun.
- **Kombiniere:** Verschmelze die sortierten Teilfolgen zu einer Gesamtfolge.

Merge Sort: älteste für den Computer entwickelte Sortieralgorithmus (John von Neumann 1945)

23

### Ablauf von MergeSort



24

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

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

- (1) **var** Index m
- (2) **if** l < r **then** {
- (3)   m:= $\lfloor (l+r)/2 \rfloor$
- (4)   MergeSort(A,l,m)
- (5)   MergeSort(A,m+1,r)
- (6)   Merge(a,l,m,r)
- (7) }

Aufruf: MergeSort(A,1,n)

### Merge(ref A,l,m,r)

**procedure** Merge(ref A,l,m,r)

- (1) i:=l; j:=m+1
- (2) **for** k:=1,...,r {
- (3)   **if** (i>m) **or** ((j≤r) **and** (A[i].key>A[j].key))
- (4)   **then**
- (5)     B[k]:=A[j]; j:=j+1
- (6)   **else**
- (7)     B[k]:=A[i]; i:=i+1
- (8) }
- (9) Schreibe sort. Folge zurück von B nach A

MergeSort(A,1,8)

MergeSort(A,1,4)

MergeSort(A,1,2)

Rekursive Aufrufe  
unseres Beispiels

MergeSort(A,1,1)

MergeSort(A,2,2)

Merge(A,1,1,2)

MergeSort(A,3,4)

MergeSort(A,3,3)

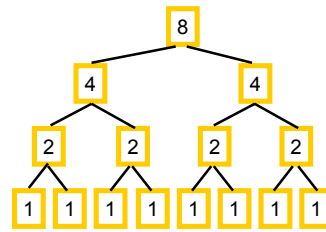
MergeSort(A,4,4)

Merge(A,3,3,4)

Merge(A,1,4,2)   ...u.s.w.

### Herleitung der Laufzeitfunktion

Sei  $n=2^k$  für ein beliebiges k



Anzahl der Instanzen	Zeit pro Instanz	Gesamtzeit
1	8	8
2	4	8
4	2	8
8	1	8

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

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

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