

# ADS: Algorithmen und Datenstrukturen

Teil  $\lceil \pi \rceil$

Peter F. Stadler & Konstantin Klemm

Bioinformatics Group, Dept. of Computer Science & Interdisciplinary Center for  
Bioinformatics, **University of Leipzig**

4. November 2009

# Sortierverfahren

## Elementare Sortierverfahren

- Sortieren durch direktes Auswählen (Straight Selection Sort)
- Sortieren durch Vertauschen (Bubble Sort)
- Sortieren durch Einfügen (Insertion Sort)
- Shell-Sort (Sortieren mit abnehmenden Inkrementen)
- Quick-Sort Auswahl-Sortierung (Turnier-Sortierung, Heap-Sort)
- Sortieren durch Streuen und Sammeln
- Merge-Sort, Externe Sortierverfahren
- Ausgeglichenes  $k$ -Wege-Merge-Sort
- Auswahl-Sortierung mit Ersetzung (Replacement Selection)

# Sortieren: Anwendungen

**Sortierung** ist fundamentale Operation in zahllosen System- und Anwendungsprogrammen: Oft **dominierender Anteil** in Programmlaufzeiten bei großen Datenmengen.

Beispiele:

- Wörter in Lexikon
- Bibliothekskatalog
- Kontoauszug (geordnet nach Datum der Kontobewegung)
- Sortierung von Studenten nach Namen, Notendurchschnitt, Semester, ...
- Sortierung von Adressen / Briefen nach Postleitzahlen, Ort, Straße, ...
- Bestandteile komplexer Anwendungen: Datenbanken, Datamining, Information Retrieval, Bioinformatik, ...

# Anwendungen: Duplikaterkennung

Naive Implementierung: Vergleiche jeden Datensatz mit jedem anderen. Algorithmus für Duplikate in Feld  $A$  der Länge  $n$ :

```
for (i=1 .. n-1)
  for ( j=i+1 .. n)
    if ( A[i] == A[j] ) mark_as_duplicate( A[j] )
```

Aufwand:  $O(n^2)$

Zweiter Versuch:

- Liste sortieren (danach stehen Duplikate unmittelbar hintereinander).
- Liste einmal linear durchlaufen und Duplikate markieren.

Falls wir besser als in  $O(n^2)$  sortieren können, ist der zweite Versuch effektiver.

# Beschreibung von Sortierverfahren I

Allgemeine Problemstellung:

- Gegeben: Folge von Sätzen  $S_1, \dots, S_n$ , wobei jeder Satz  $S_i$  Schlüssel  $K_i$  besitzt.

Gesucht: Permutation der Zahlen von 1 bis  $n$ , welche aufsteigende Schlüsselreihenfolge ergibt:

$$K_{(1)} \leq K_{(2)} \leq \dots \leq K_{(k)}.$$

- Sortierverfahren:
  - intern (im Hauptspeicher)
    - $O(n^\alpha)$
    - $O(n \log n)$
  - extern (Nutzung von Externspeicher)
    - Mischen von vorsortierten Folgen

# Beschreibung von Sortierverfahren II

- einfache vs. spezielle Sortierverfahren
  - Annahmen über Datenstrukturen (z.B. Array) oder Schlüsseleigenschaften
- Wünschenswert: stabile Sortierverfahren, bei denen die relative Reihenfolge gleicher Schlüsselwerte bei der Sortierung gewahrt bleibt
- Speicherplatzbedarf am geringsten für Sortieren am Ort (*in situ*)
- Weitere Kostenmaße:
  - # Schlüsselvergleiche  $C$  ( $C_{\min}$ ,  $C_{\max}$ ,  $C_{\text{avg}}$ )
  - # Satzbewegungen  $M$  ( $M_{\min}$ ,  $M_{\max}$ ,  $M_{\text{avg}}$ )

# Wie schnell kann man sortieren?

**Voraussetzung** für jedes Sortieren:

Auf den zu sortierenden Objekten muss eine Ordnung definiert sein.

**Satz.** Jedes allgemeine Sortierverfahren, welches zur Sortierung nur Vergleichsoperationen zwischen Schlüsseln (sowie Tauschoperationen) verwendet, benötigt sowohl im mittleren als auch im schlechtesten Fall wenigstens  $\Omega(n \log n)$  Schlüsselvergleiche.

# Klassifizierung von Sortiertechniken

Sortieren durch ...

① Auswählen

*	*	*	z	-	...	-	X	-	...	-
*	*	*	X	-	...	-	z	-	...	-

② Einfügen

*	*	*	*	-	...	-	X	-	...	-
*	*	X	*	*	-	...	-	-	...	-

③ Lokales Austauschen: vertausche lokale Fehlordnungen  $x > y$

-	-	-	-	-	...	x	y	-	...	-
-	-	-	-	-	...	y	x	-	...	-

④ Nicht-Lokales Austauschen:

$$K_i > K \dots K \dots K_j < K$$

$$K_j < K \dots K \dots K_i > K$$

# Klassifizierung von Sortiertechniken II

## Sortieren durch ...

- Mischen (“Reißverschlussverfahren”)  
 $L_1 = [10, 20, 40, 70, \dots]$   
 $L_2 = [30, 50, 60, 90, \dots]$   
Mischen:  
 $L = [10, 20, 30, 40, 50, 60, 70, \dots]$
- Streuen & Sammeln
  - begrenzter Schlüsselbereich  $m$ , z.B. 1 – 1000
  - relativ dichte Schlüsselbelegung  $n \leq m$
  - Duplikate möglich ( $n > m$ )
  - **lineare Sortierkosten !!!!**

# Klassifizierung von Sortiertechniken III

Sortieren durch ...

- Fachverteilen (z.B. Poststelle, z.B. Postleitzahlen in 10 Fächer nach der 1. Stelle)  
macht Sinn für VIELE Duplikate
- dann Sortierung der "Fächer"

# Sortieren durch Auswählen (Selection Sort) I

Algorithmus:

- 1 **Start:** Unsortierte Teilliste (UT) ist die ganze Liste
  - 2 **WHILE**( Liste UT enthält mehr als ein Element ):
  - 3 Auswahl des kleinsten Elementes  $x$  in UT
  - 4 Austausch von  $x$  mit dem **ersten** Element von UT
  - 5 Reduziere UT um dieses Element
- ENDWHILE**

Beispiel: 3 12 27 99 27 75 → TAFEL

# Selection Sort II

```
for i=1 ... n-1
  min = i
  for j=i+1 bis n
    if (A(j) < A(min)) min = j
  exchange A(i) and A(min)
```

Eigenschaften:

Anzahl Schlüsselvergleiche:  $C_{\min}(n) = C_{\max}(n) = n(n-1)/2$

Anzahl Satzbewegungen (durch Swap):

$M_{\min}(n) = M_{\max}(n) = 3(n-1)$

*In-situ*-Verfahren, nicht stabil

# Bubble Sort I

Idee:

- Vertauschen benachbarter Elemente, die nicht in Sortierordnung sind
  - pro Durchgang wandert größtes Element der noch unsortierten Teilliste nach “oben”
  - Sortierung endet, wenn in einem Durchgang keine Vertauschung mehr erfolgte
- Methode: “Sortieren durch lokale Vertauschung”  
Variation: Shaker-Sort (Durchlaufrichtung wechselt bei jedem Durchgang)

# Bubble Sort II

Algorithmus 1:

```
for i=n-1 .. 1
  for j=1 .. i
    if ( A(j) > A(j+1) ) exchange A(j) and A(j+1)
```

Eigenschaften:

Anzahl Schlüsselvergleiche:

$C_{\min}(n) = C_{\max}(n) = C_{\text{avg}}(n) = n(n-1)/2$  Anzahl

Satzbewegungen (durch Swap):  $M_{\min}(n) = 0$ ,

$M_{\max}(n) = 3n(n-1)/2$ ,  $M_{\text{avg}}(n) = M_{\max}/2$

*In-situ*-Verfahren, stabil.

Vorsortierung kann teilweise genutzt werden

# Bubble Sort III

## Algorithmus 2 (mit Abbruchkontrolle)

```
Vertauschungen = true
while( Vertauschungen )
    Vertauschungen = false
    for i=1 .. n-1
        if ( A(i) > A(i+1) )
            exchange A(i) and A(i+1);
            Vertauschungen = true
```

Eigenschaften:

Anzahl Schlüsselvergleiche:  $C_{\min}(n) = n - 1$ ,  $C_{\max}(n) = n(n - 1)/2$

Anzahl Satzbewegungen (durch Swap):  $M_{\min}(n) = 0$ ,

$M_{\max}(n) = 3n(n - 1)/2$

*In-situ*-Verfahren, stabil.

Vorsortierung kann teilweise genutzt werden

Beispiel: 27 75 99 3 45 12 87      **Tafel**

# Sortieren durch Einfügen (Insertion Sort) I

Idee:

- $i$ -tes Element der Liste  $x$  (1. Element der unsortierten Teilliste) wird an der richtigen Stelle der bereits sortierten Teilliste (1 bis  $i - 1$ ) eingefügt
- Elemente in sortierter Teilliste mit höherem Schlüsselwert als  $x$  werden verschoben

```
for(i=2 .. n)
    temp = A(i); j = i-1;
    while ( j>0 and A(j) > temp ) A(j+1) = A(j); j--
    A(j+1) = temp;
```

Beispiel 27 75 99 3 45 12 87 → TAFEL  
EINFACH ZUM NACHDENKEN: Eigenschaften

# Vergleich der einfachen Sortierverfahren

## Schlüsselvergleiche

Algorithm	$C_{\min}$	$C_{\text{avg}}$	$C_{\max}$
Selection	$n(n-1)/2$	$n(n-1)/2$	$n(n-1)/2$
Bubble	$n(n-1)/2$	$n(n-1)/2$	$n(n-1)/2$
Insertion	$n-1$	$(n^2 + 3n + 4)/4$	$(n^2 + n - 2)/2$

## Satzbewegungen

Algorithm	$C_{\min}$	$C_{\text{avg}}$	$C_{\max}$
Selection	$3(n-1)$	$3(n-1)$	$3(n-1)$
Bubble	0	$3n(n-1)/4$	$3n(n-1)/2$
Insertion	$3(n-1)$	$(n^2 + 11n + 12)/4$	$(n^2 + 5n + 6)/2$

# Sortieren von großen Datensätzen

**Indirektes Sortieren** erlaubt, Kopieraufwand für jedes Sortierverfahren auf lineare Kosten  $O(n)$  zu beschränken  
Führen eines Hilfsfeldes von Indizes auf das eigentliche Listenfeld  
Liste:  $A[1..n]$   
Pointerfeld  $P[1..n]$  (Initialisierung  $P[i] = i$ )  
Schlüsselzugriff: statt  $A[i]$  immer  $A[P[i]]$   
Austausch: statt  $swap(A, i, j)$  nur  $swap(P, i, j)$   
Sortierung erfolgt lediglich auf Indexfeld  
abschließend erfolgt linearer Durchlauf zum Umkopieren der Sätze

# Shell Sort

Sortieren mit abnehmenden Inkrementen (Shell, 1957)

Idee

- Sortierung in mehreren Stufen “von grob bis fein”
- Vorsortierung reduziert Anzahl von Tauschvorgängen

Vorgehensweise:

- Festlegung von  $t$  Inkrementen (Elementabständen)  $h_i$  mit  $h_1 > h_2 > \dots > h_t = 1$ . Wir bezeichnen als  $h_i$ -Folge eine Folge von Elementen aus der Liste mit Abstand  $h_i$
- Im  $i$ -ten Schritt erfolgt unabhängiges Sortieren aller  $h_i$ -Folgen (mit Insertion Sort)
- Eine Elementbewegung bewirkt Sprung um  $h_i$  Positionen
- Im letzten Schritt erfolgt “normales” Sortieren durch Einfügen

BESPIEL:  $h_1 = 4$ ,  $h_2 = 2$ ,  $h_3 = 1$

27 75 99 3 45 12 87 → TAFEL

# Shell-Sort II

Aufwand wesentlich von Wahl der Anzahl und Art der Schrittweiten abhängig

Insertion Sort ist Spezialfall ( $t = 1$ ).

Knuth [Kn73] empfiehlt':

- Schrittweitenfolge von 1, 3, 7, 15, 31 ...  
 $h_{i-1} = 2h_i + 1$ ,  $h_t = 1$  und  $t = \lceil \log_2 n \rceil - 1$
- Aufwand  $O(n^{1.2})$
- Andere Vorschläge erreichen  $O(n \log_2 n)$  — Inkremente der Form  $2^p 3^q$ .