Algorithmen und Datenstrukturen 1

4. Vorlesung

Peter F. Stadler

Universität Leipzig
Institut für Informatik
studla@bioinf.uni-leipzig.de

4. 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)

Sortier-Wettbewerbe

Sortieren: Anwendungen

Sortierung ist fundamentale Operation in zahllosen System- und Anwendungsprogrammen: 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

Anwendungen : Duplikaterkennung

Naive Implementierung: Vergleiche jeden Datensatz mit jedem anderen.

Algorithmus für Duplikate in Feld A der Länge n:

```
Für i=1 bis n-1
   Für j=i+1 bis n
    falls A(i) = A(j), dann markiere A(j) als Duplikat
```

Aufwand: $O(n^2)$

Zweiter Versuch:

1. Schritt: Liste sortieren (danach stehen Duplikate unmittelbar hintereinander.)
2. Schritt: Liste einmal linear durchlaufen und Duplikate markieren.

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

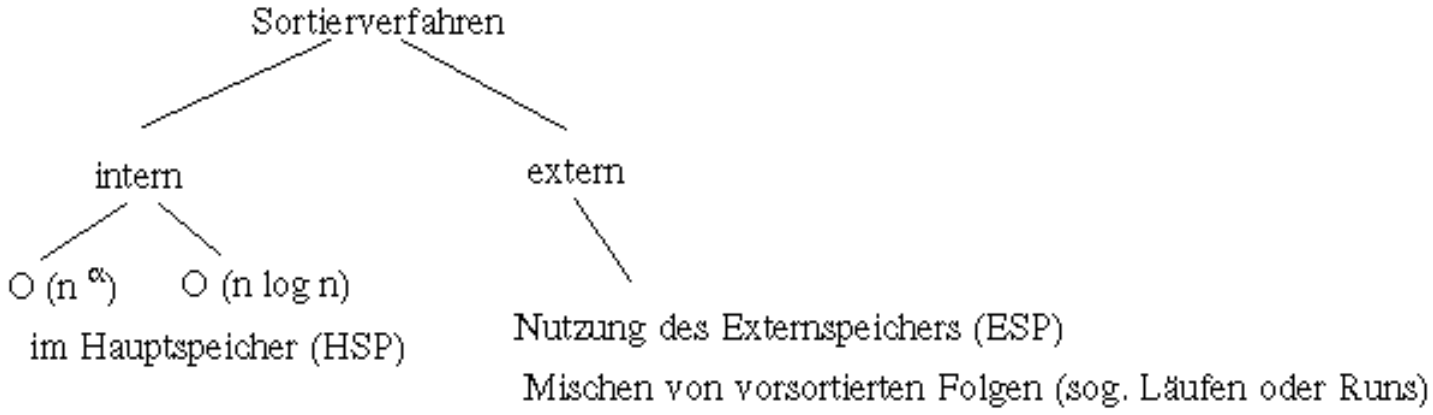
Beschreibung von Sortierverfahren I

Allgemeine Problemstellung

- Gegeben: Folge von Sätzen S_1, ... S_n, wobei jeder Satz S_i Schlüssel K_i besitzt
- Gesucht: Permutation der Zahlen von 1 bis n, welche aufsteigende Schlüsselreihenfolge ergibt:

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

Interne vs. externe Sortierverfahren



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_{avg})
  - #Satzbewegungen M (M_{min}, M_{max}, M_{avg})

Wie schnell kann man sortieren?

Voraussetzung für jedes Sortieren:

Auf den zu sortierenden Objekten muss eine Ordnung definiert sein.

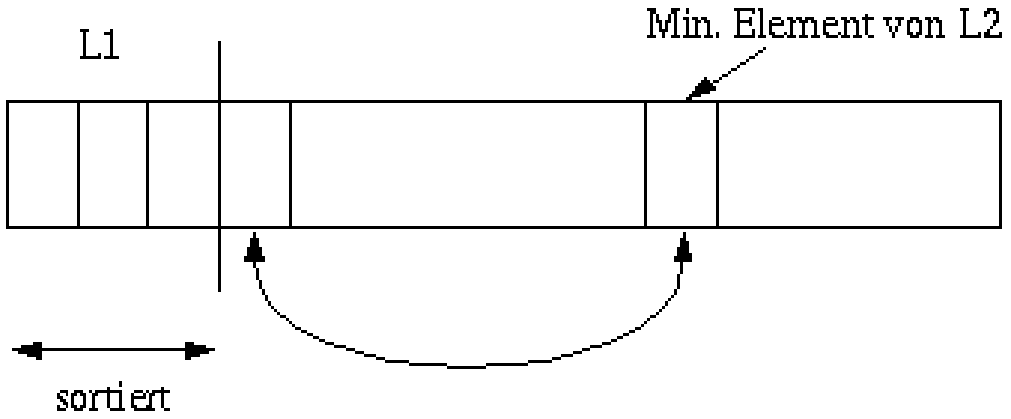
Satz:

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

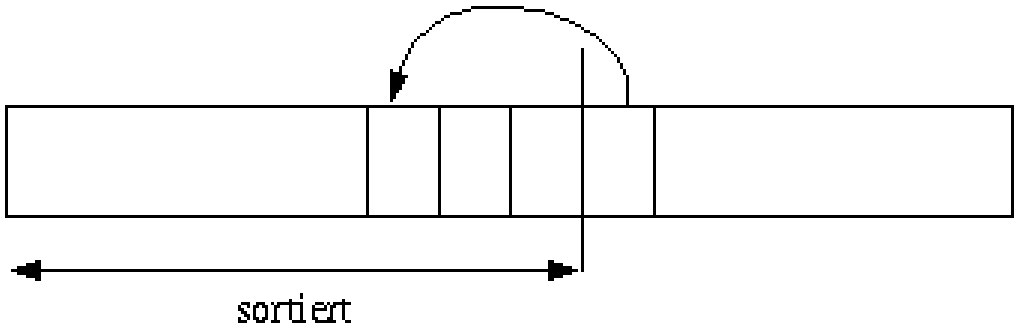
Klassifizierung von Sortiertechniken I

Sortieren durch ...

1) Auswählen



2) Einfügen

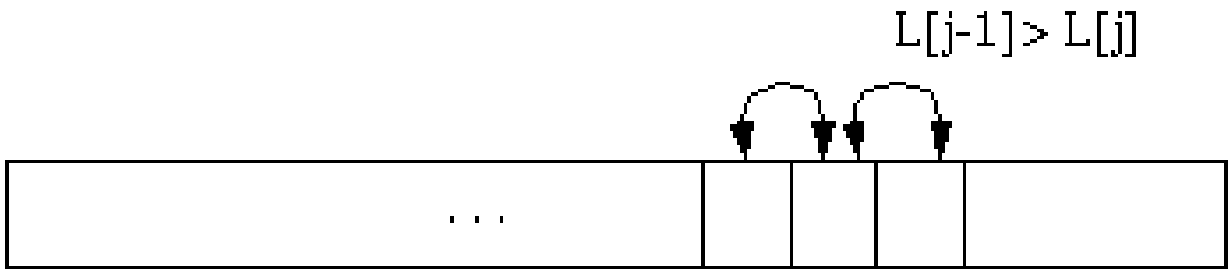


Klassifizierung von Sortiertechniken II

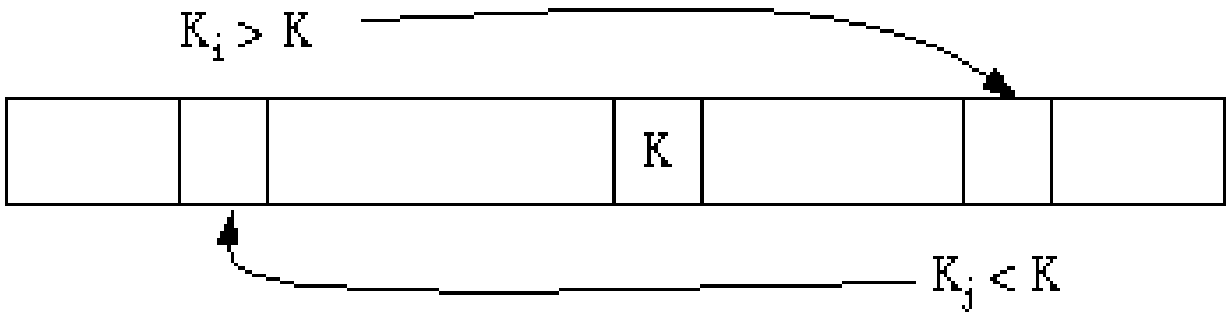
Sortieren durch ...

3) Austauschen

a) lokal



b) entfernt

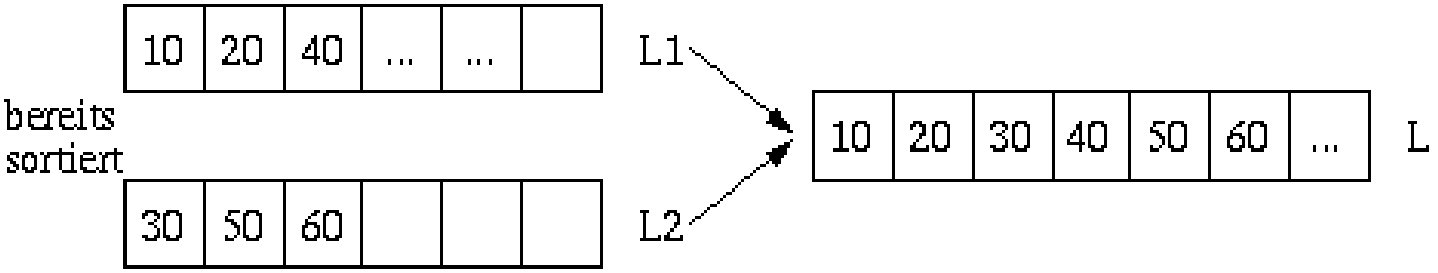


Klassifizierung von Sortiertechniken III

Sortieren durch ...

4) Mischen

(„Reißverschlussverfahren“)



Klassifizierung von Sortiertechniken IV

Sortieren durch ...

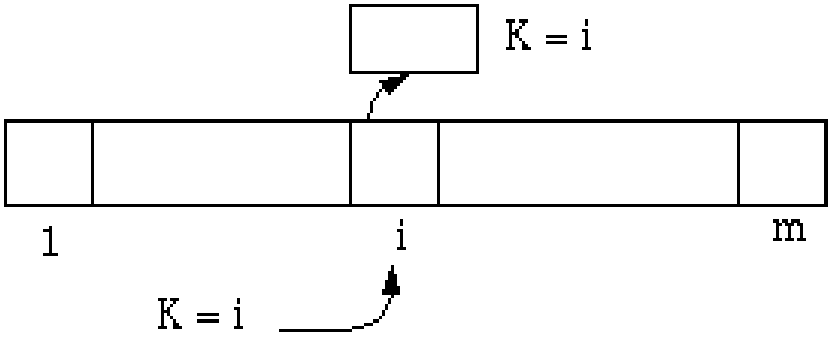
5. Streuen und Sammeln

   begrenzter Schlüsselbereich m, z. B. 1 - 1000

   relativ dichte Schlüsselbelegung n ≤ m

   Duplikate möglich (n > m)

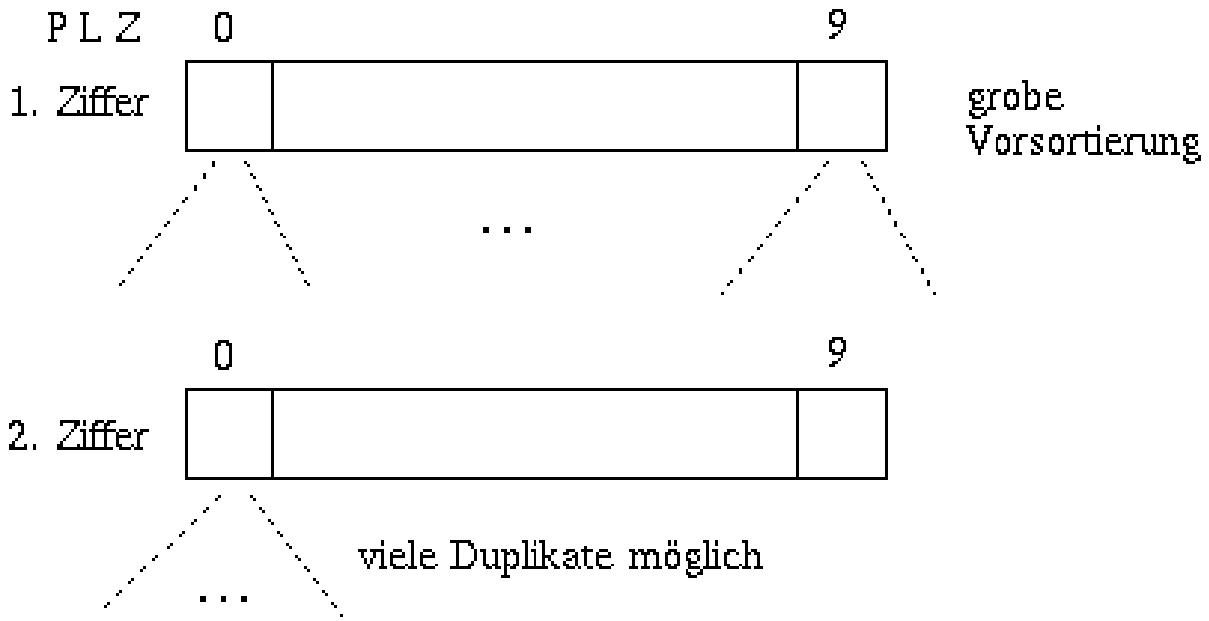
   lineare Sortierkosten !



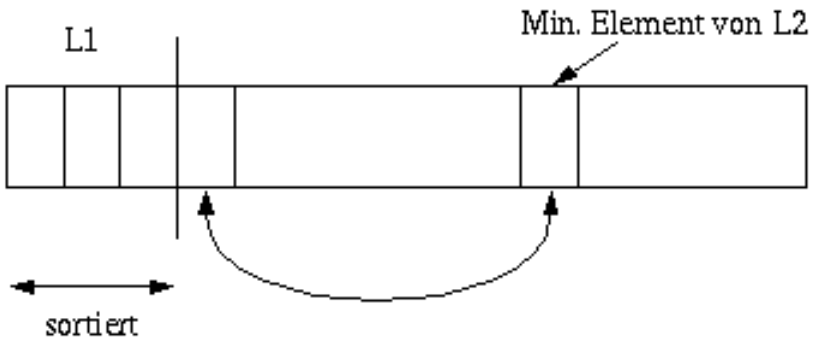
Klassifizierung von Sortiertechniken V

Sortieren durch ...

6. Fachverteilen (z. B. Poststelle)



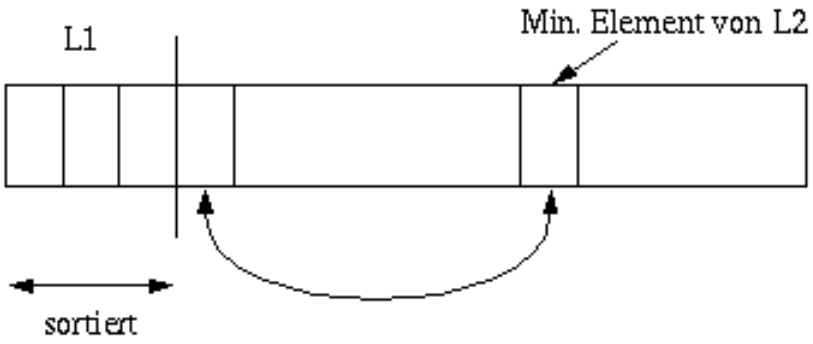
Sortieren durch Auswählen (Selection Sort) I



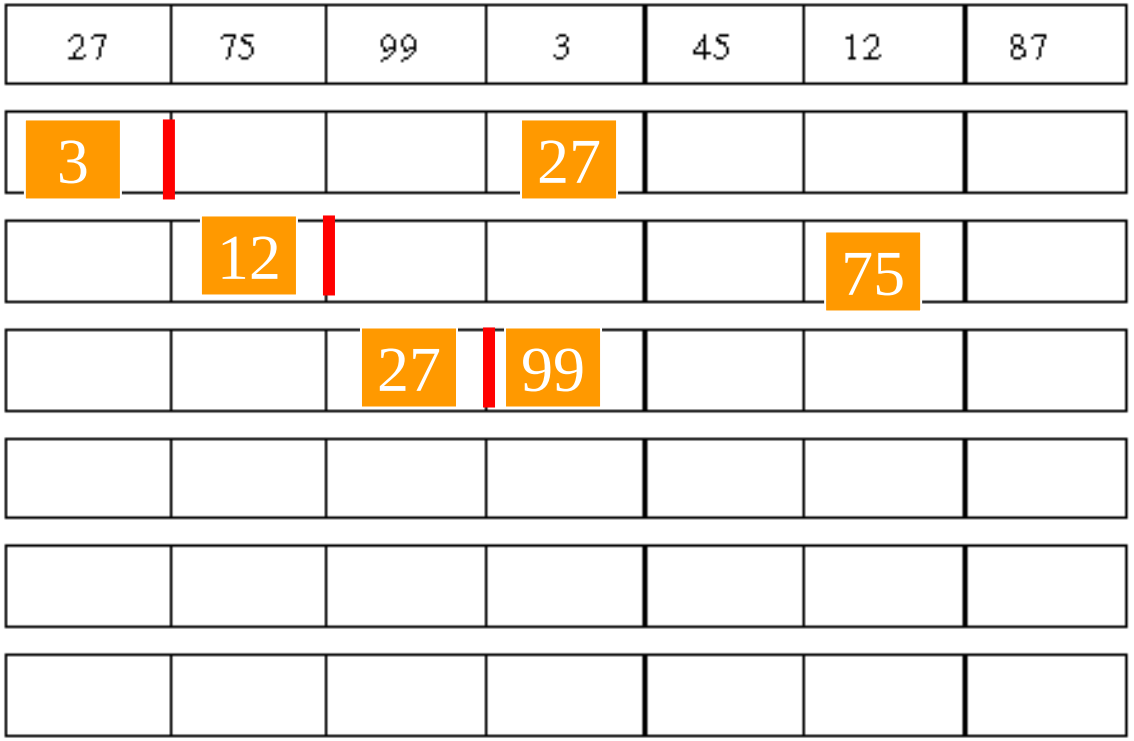
Algorithmus:

1. Start: Unsortierte Teilliste (UT) ist die ganze Liste. Falls UT nicht leer:
2. Auswahl des kleinsten Elementes in UT
3. Austausch mit dem ersten Element von UT.
4. Reduziere UT um dieses Element.
5. Abbruch falls UT nur aus einem Element besteht.
6. Gehe zu 2.

Selection Sort – Beispiel



Beispiel:



Sortieren durch Auswählen (Selection Sort) II

Algorithmus

```
Äußere Schleife: Für i=1 bis n-1
   Setze min = i
   Innere Schleife: Für j=i+1 bis n
       Falls A(j) < A(min)
              setze min = j
   Tausche A(i) und 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

Sortieren durch Vertauschen (Bubble Sort) I

Idee

- Vertauschen benachbarter Elemente, die nicht in Sortierordnung
- 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)

Sortieren durch Vertauschen (Bubble Sort) II

Algorithmus 1

```
Äußere Schleife: Für i=n-1 bis 1
   Innere Schleife: Für j=1 bis i
       Falls A(j) > A(j+1)
             Tausche A(j) und A(j+1)
```

Eigenschaften

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

Anzahl Satzbewegungen (durch Swap):

$M_{min}(n) = 0$

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

$M_{avg}(n) = M_{max}/2$

In-situ-Verfahren, stabil. Vorsortierung kann teilweise genutzt werden

Sortieren durch Vertauschen (Bubble Sort) III

Algorithmus 2 (mit Abbruchkontrolle)

```
Äußere Schleife: solange noch Vertauschungen auftreten
   Innere Schleife: Für i=1 bis n-1
       Falls A(i) > A(i+1)
             Tausche A(i) und A(i+1)
```

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 gut genutzt werden

Bubble Sort Beispiel

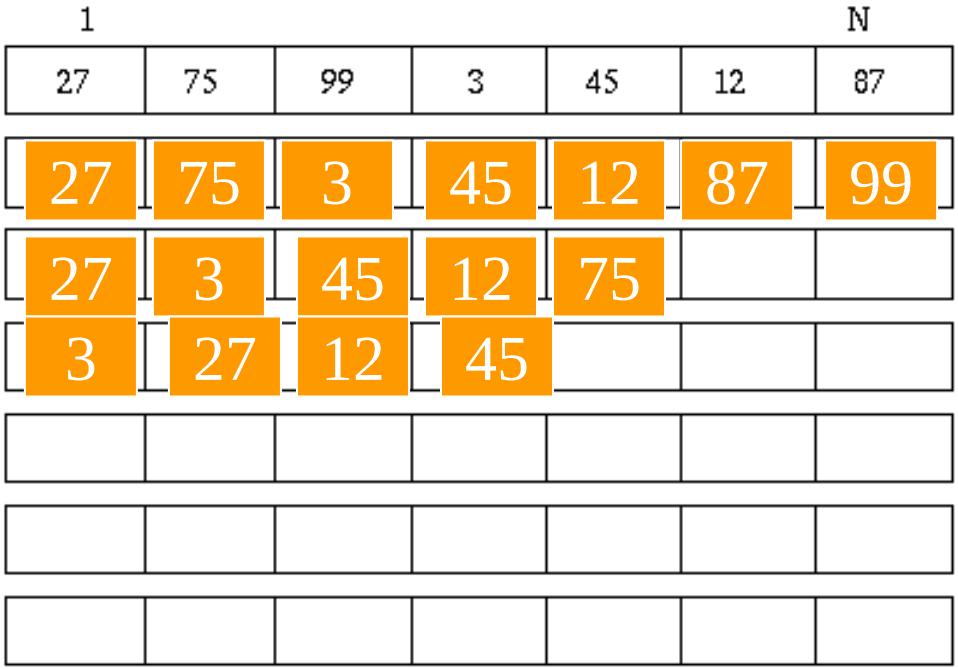
Algorithmus 2 (mit Abbruchkontrolle)

Äußere Schleife: solange noch Vertauschungen auftreten

Innere Schleife: Für i = 1 bis n-1

Falls A(i) > A(i+1)

Tausche A(i) und A(i+1)

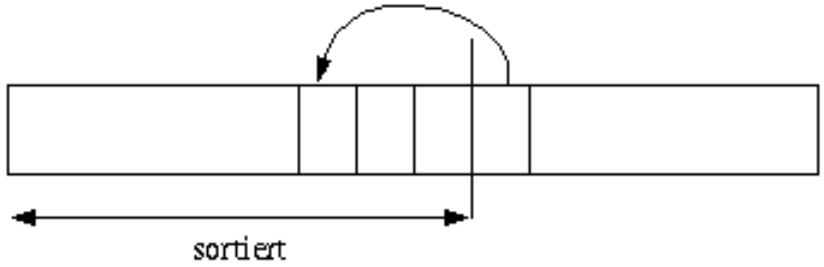


Sortieren durch Einfügen (Insertion Sort) I

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

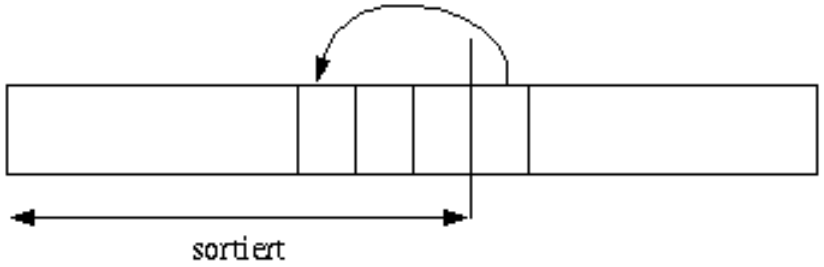
Illustration:



Algorithmus 1

```
Äußere Schleife:                    Für i = 2 bis n
        Einfügeschritt initialisieren:   temp = A(i); j = i-1;
        Platz schaffen:                      while ( j>0 and A(j) > temp )
                                                 A(j+1) = A(j); j--;
        Dort einfügen:                       A(j+1) = temp;
```

Sortieren durch Einfügen (Insertion Sort) II




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

Eigenschaften

Anzahl Schlüsselvergleiche: $C_{min}(n) =$

$C_{max}(n) =$

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

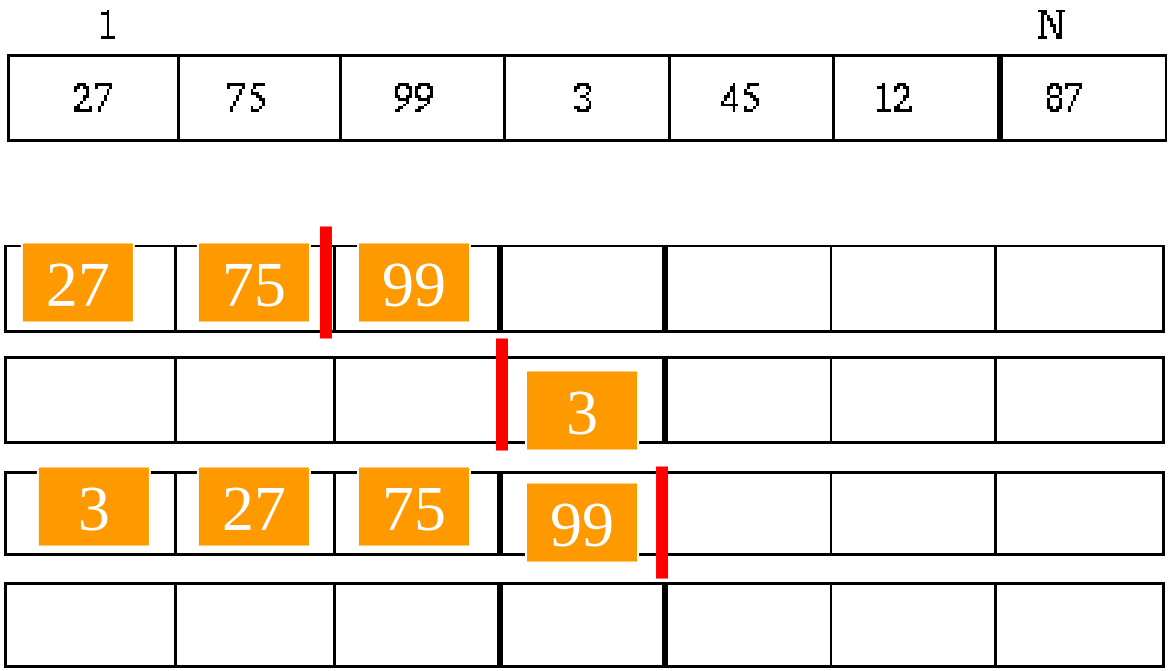
$M_{max}(n) =$

$M_{avg}(n) =$

In-situ-Verfahren, stabil

Insertion Sort Beispiel

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



Vergleich der einfachen Sortierverfahren

Schlüsselvergleiche

	C_{min}	C_{avg}	C_{max}
Direktes Auswählen	$n (n - 1) / 2$	$n (n - 1) / 2$	$n (n - 1) / 2$
Bubble Sort	$n (n - 1) / 2$	$n (n - 1) / 2$	$n (n - 1) / 2$
Einfügen	$n - 1$	$(n^2 + 3n + 4) / 4$	$(n^2 + n - 2) / 2$

Satzbewegungen

	M_{min}	M_{avg}	M_{max}
Direktes Auswählen	$3 (n - 1)$	$3 (n - 1)$	$3 (n - 1)$
Bubble Sort	0	$3n (n - 1) / 4$	$3n (n - 1) / 2$
Einfügen	$3 (n - 1)$	$(n^2 + 11n + 12) / 4$	$(n^2 + 5n - 6) / 2$

Sortieren von großen Datensätzen (Indirektes Sortieren)

Indirektes Sortieren erlaubt, Kopieraufwand für jedes Sortierverfahrens 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) I

(Shell, 1957)

Idee

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

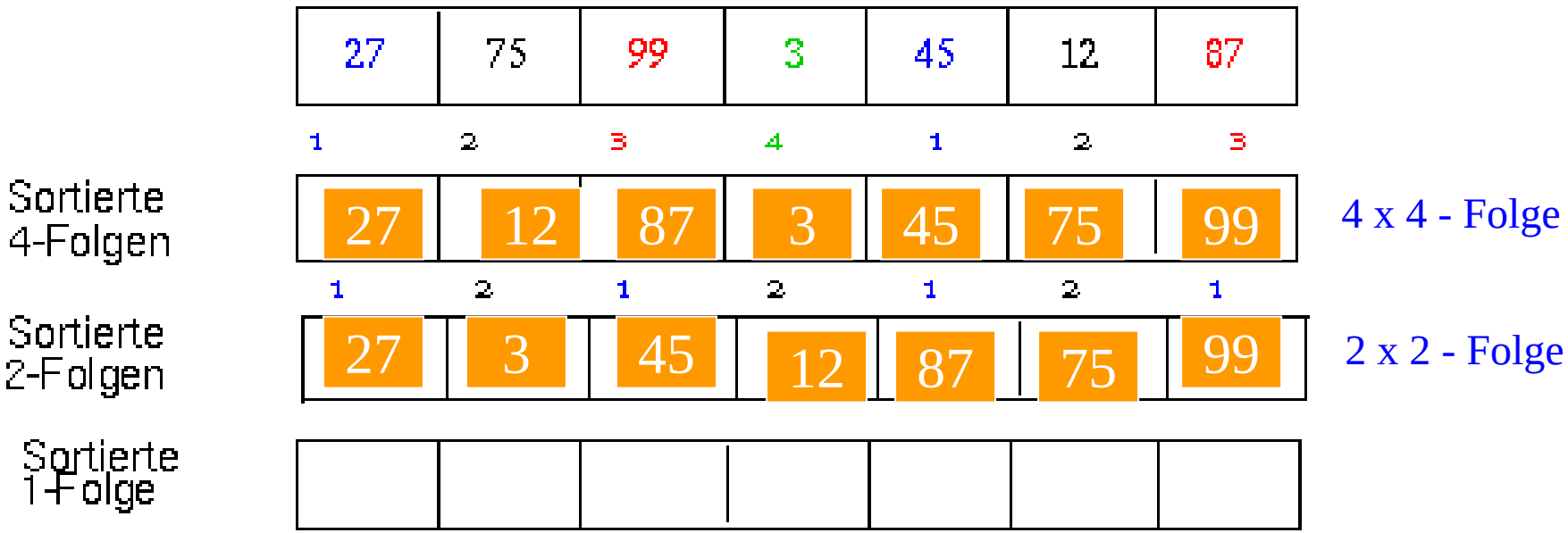
Vorgehensweise:

- Festlegung von t Inkrementen (Elementabständen) h_i mit $h_1 > h_2 > ... > h_t = 1$. 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

Shell-Sort (Sortieren mit abnehmenden Inkrementen) II

Beispiel:

$h_1 = 4; h_2 = 2; h_3 = 1$



Shell-Sort (Sortieren mit abnehmenden Inkrementen) III

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 ... mit $h_{i-1} = 2 \cdot h_i + 1$, $h_t = 1$ und $t = [\log_2 n] - 1$
- Aufwand $O(n^{1.2})$

Andere Vorschläge erreichen $O(n \log_2 n)$

- Inkremente der Form $2^p 3^q$