

Grundlagen der Programmierung (Vorlesung 18)

Ralf Möller, FH-Wedel

■ Vorige Vorlesung

- Asymptotische Komplexität von Algorithmen

■ Inhalt dieser Vorlesung

- Sortieralgorithmen und deren Analyse

■ Lernziele

- Grundlagen der Analyse von Algorithmen

Danksagung

- Das Material ist angelehnt an die Materialien aus der Vorlesung Datenstrukturen und Algorithmen von Prof. Dr. M. Jarke, M. Gebhardt, T. v. d. Maßen, A. Nowack, Dr. J.-C. Töbermann, RWTH Aachen
- <http://www-i5.informatik.rwth-aachen.de/lehrstuhl/lehre/DA01/>

Vorabbemerkungen (1)

Definition: Partielle Ordnung

Es sei \mathcal{M} eine nicht leere Menge und $\leq \subseteq \mathcal{M} \times \mathcal{M}$ eine binäre Relation auf \mathcal{M} . Das Paar $\langle \mathcal{M}, \leq \rangle$ heißt eine *partielle Ordnung* auf \mathcal{M} , genau dann wenn \leq die folgenden Eigenschaften erfüllt:

- **Reflexivität:** $\forall x \in \mathcal{M} : x \leq x$
- **Transitivität:** $\forall x, y, z \in \mathcal{M} : x \leq y \wedge y \leq z \rightarrow x \leq z$
- **Antisymmetrie:** $\forall x, y \in \mathcal{M} : x \leq y \wedge y \leq x \rightarrow x = y$

Vorabbemerkungen (2)

Definition: Strikter Anteil einer Ordnungsrelation

Für eine partielle Ordnung \leq auf einer Menge \mathcal{M} definieren wir die Relation $<$ durch:

$$x < y := x \leq y \wedge x \neq y$$

Die Relation $<$ heißt auch der *strikte Anteil* von \leq .

Definition: Totale Ordnung

Es sei \mathcal{M} eine nicht leere Menge und $\leq \subseteq \mathcal{M} \times \mathcal{M}$ eine binäre Relation über \mathcal{M} . \leq heißt eine *totale Ordnung* auf \mathcal{M} , genau dann wenn gilt:

- $< \mathcal{M}, \leq >$ ist eine partielle Ordnung und
- **Trichotomie:** $\forall x, y \in \mathcal{M} : \quad x < y \vee x = y \vee y < x$

Sortierung von Reihenungen

■ Sortierproblem - Definition:

- Gegeben sei eine Reihung a der Form
array [1..n] of M und eine totale Ordnung
 \leq definiert auf M .
- Gesucht in eine Reihung b : array [1..n] of M , so daß gilt:
 $\forall 1 \leq i < n . (b[i] \leq b[i+1] \wedge \exists j \in \{1, \dots, n\} . (a[j] = b[i]))$

Unterscheidungskriterien (1)

- M kann eine Menge zusammengesetzter Objekte sein
 - Die Elemente aus M können wieder Reihungen (Arrays) sein
 - Andere Beispiele sind (Name, Vorname) usw.
- M kann auch N_0 sein

Unterscheidungskriterien (2)

- Üblicherweise wird für die Ordnungsrelation bei zusammengesetzten Reihungselementen nur eine Teilkomponente als Schlüssel verwendet

Liste	Schlüsselement	Ordnung
Telefonbuch	Nachname	lexikographische Ordnung
Klausurergebnisse	Punktezahl	\leq auf \mathbb{Q}
Lexikon	Stichwort	lexikographische Ordnung
Studentenverzeichnis	Matrikelnummer	\leq auf \mathbb{N}
Entfernungstabelle	Distanz	\leq auf \mathbb{R}
Fahrplan	Abfahrtszeit	„früher als“

Unterscheidungskriterien (3)

■ Stabilität

- Ein Sortierverfahren heißt stabil, wenn sich in der Ergebnisreihung die Reihenfolge gleicher Elemente nicht ändert
- Relevant ist dieses für Reihungen mit zusammengesetzten Elementen

Vereinfachung

- Wir betrachten hier zur Vereinfachung nur $M = N_0$
- Wir suchen eine Sortierfunktion, d.h. das Eingabearray wird nicht verändert. Der Algorithmus arbeitet auf einer vorher erstellten Kopie der Eingabe
- Wird als Ergebnis die gleiche Reihung verwendet, spricht man von In-situ-Sortieren

Abkürzung: For-Schleife

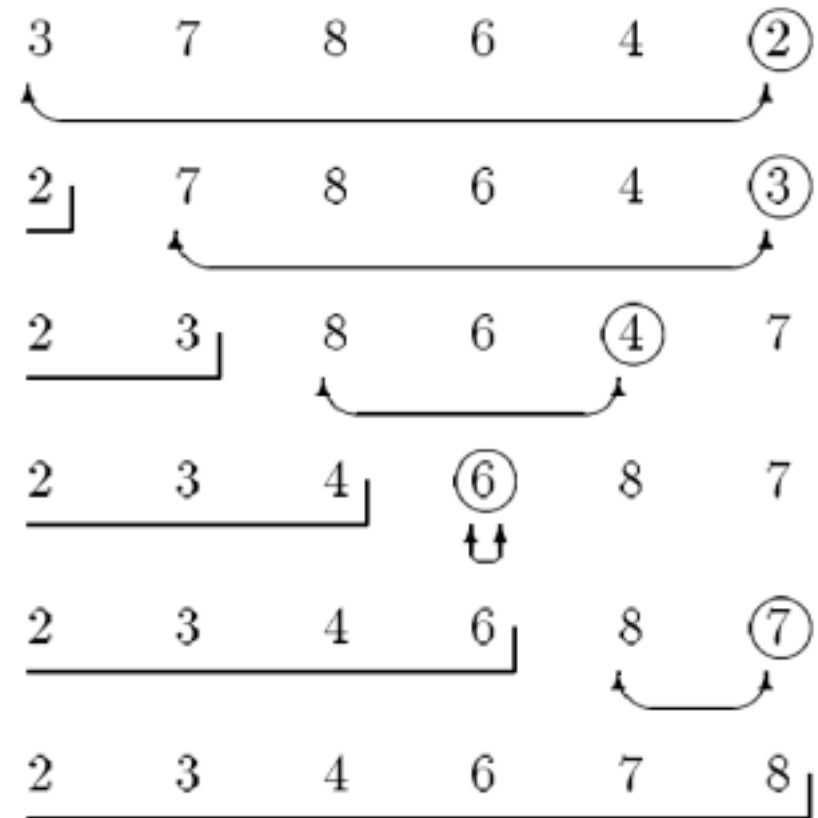
■ for $i := a$ to b by X do $\langle \text{Rumpf} \rangle$ end for

steht für:

■ $i := a$;
while $\neg(i = b)$ do
 $\langle \text{Rumpf} \rangle$;
 $i := i + X$
end while

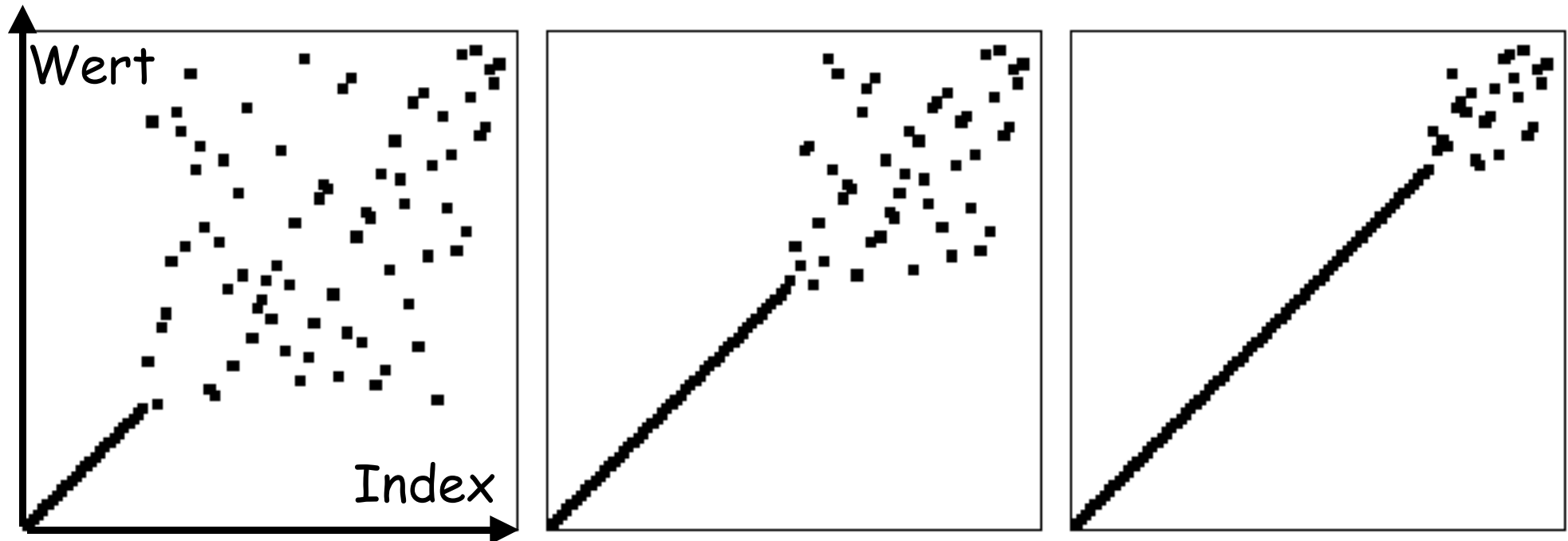
Sortieren durch Auswahl: selection-sort

```
■ selection-sort(a : array [1..n] of  $N_0$ ) : array [1..n] of  $N_0$ 
begin
  var i, j, min :  $N_0$ ;
  for i := 1 to n-1 do
    min := i;
    for j := i + 1 to n do
      if a[j] < a[min]
        then min := j
      end if
    end for;
    a[i], a[min] := a[min], a[i]
  end for;
  a
end
```



Ein Beispiellauf

- Ein Kasten zeigt das Array a zu einem Zeitpunkt
- Ein Punkt kennzeichnet einen Arraywert
(Wert durch Höhe kodiert, Index durch Position)
- Annahme: Zufällige Sortierung am Anfang



Komplexitätsanalyse

Komplexitätsanalyse

Zum Sortieren der gesamten Folge $a[1], \dots, a[N]$ werden $N - 1$ Durchläufe benötigt. Pro Schleifendurchgang i gibt es eine Vertauschung, die sich aus je drei Bewegungen und $N - i$ Vergleichen zusammensetzt, wobei $N - i$ die Anzahl der noch nicht sortierten Elemente ist. Insgesamt ergeben sich:

- $3 \cdot (N - 1)$ Bewegungen und
- $(N - 1) + (N - 2) + \dots + 2 + 1 = \frac{N \cdot (N - 1)}{2}$ Vergleiche.

Da die Anzahl der Bewegungen nur linear in der Anzahl der Datensätze wächst, ist SelectionSort besonders für Sortieraufgaben geeignet, in denen die einzelnen Datensätze sehr groß sind.

Sortieren durch Einfügen: insertion-sort

■ insertion-sort(a : array $[1..n]$ of N_0): array $[1..n]$ of N_0

begin

var i, j, w : N_0 ;

for $i := 2$ to n do

$w := a[i]$;

$j := i$;

 while $1 < j \wedge a[j-1] > w$ do

$a[j] := a[j-1]$;

$j := j - 1$

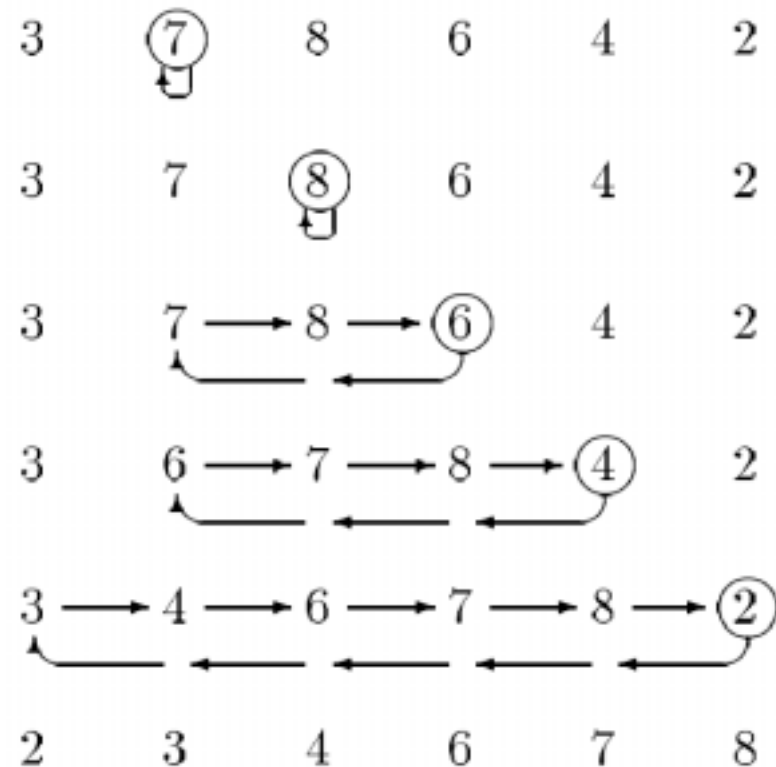
 end ;

$a[j] := w$;

end ;

a

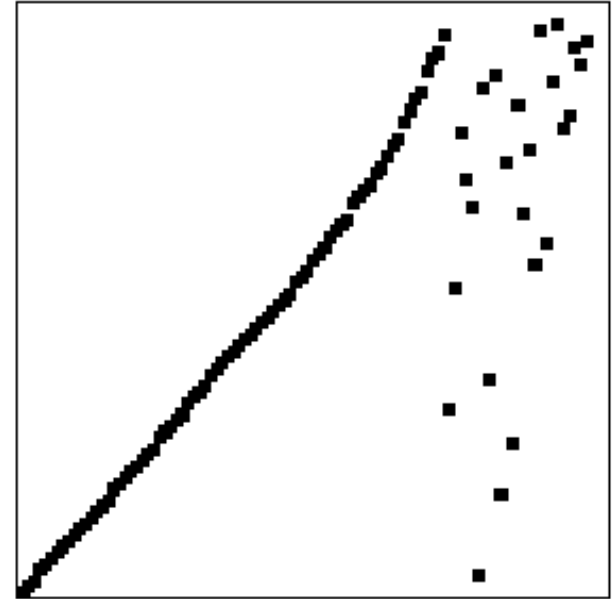
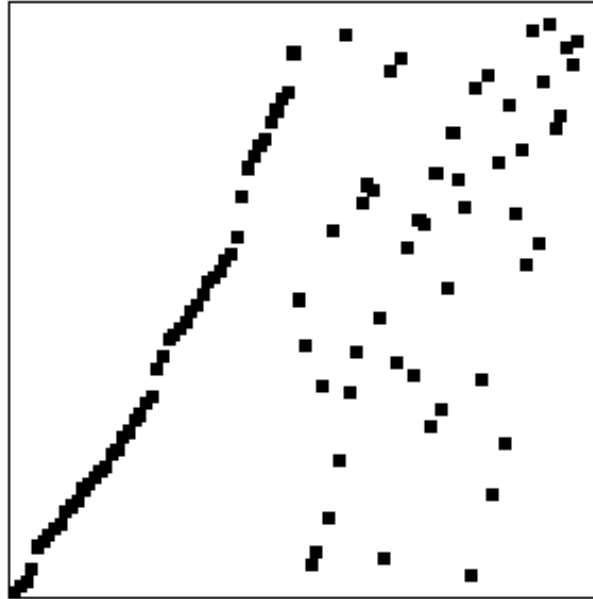
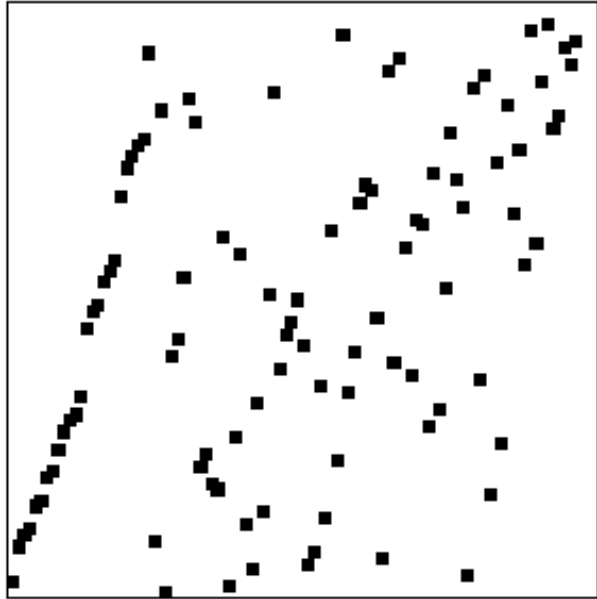
end



Bemerkung

- Nicht-strikte Auswertung von \wedge sichert, daß im Falle $j = 1$ nicht auf $a[0]$ zugegriffen wird.

Ein Beispiellauf



Komplexitätsanalyse (1)

Vergleiche: Das Einfügen des Elements $a[i]$ in die bereits sortierte Anfangsfolge $a[1], \dots, a[i-1]$ erfordert mindestens einen Vergleich, höchstens jedoch i Vergleiche. Im Mittel sind dies $i/2$ Vergleiche, da bei zufälliger Verteilung der Schlüssel die Hälfte der bereits eingefügten Elemente größer ist als das Element $a[i]$.

- *Best Case* – Bei vollständig vorsortierten Folgen ergeben sich $N - 1$ Vergleiche.
- *Worst Case* – Bei umgekehrt sortierten Folgen gilt für die Anzahl der Vergleiche:

$$\begin{aligned}\sum_{i=2}^N i &= \left(\sum_{i=1}^N i \right) - 1 = \frac{N(N+1)}{2} - 1 \\ &= \frac{N^2}{2} + \frac{N}{2} - 1\end{aligned}$$

- *Average Case* – Die Anzahl der Vergleiche ist etwa $N^2/4$

Komplexitätsanalyse (2)

Bewegungen: Im Schleifendurchgang i ($i = 2, \dots, N$) wird bei bereits sortierter Anfangsfolge $a[1], \dots, a[i-1]$ das einzufügende Element $a[i]$ zunächst in die Hilfsvariable v kopiert (eine Bewegung) und anschließend mit höchstens i Schlüsseln

wenigstens jedoch mit einem Schlüssel und im Mittel mit $i/2$ Schlüsseln verglichen. Bis auf das letzte Vergleichselement werden die Datensätze um je eine Position nach rechts verschoben (jeweils eine Bewegung). Anschließend wird der in v zwischengespeicherte Datensatz an die gefundene Position eingefügt (eine Bewegung). Für den Schleifendurchgang i sind somit mindestens zwei Bewegungen, höchstens jedoch $i + 1$ und im Mittel $i/2 + 2$ Bewegungen erforderlich.

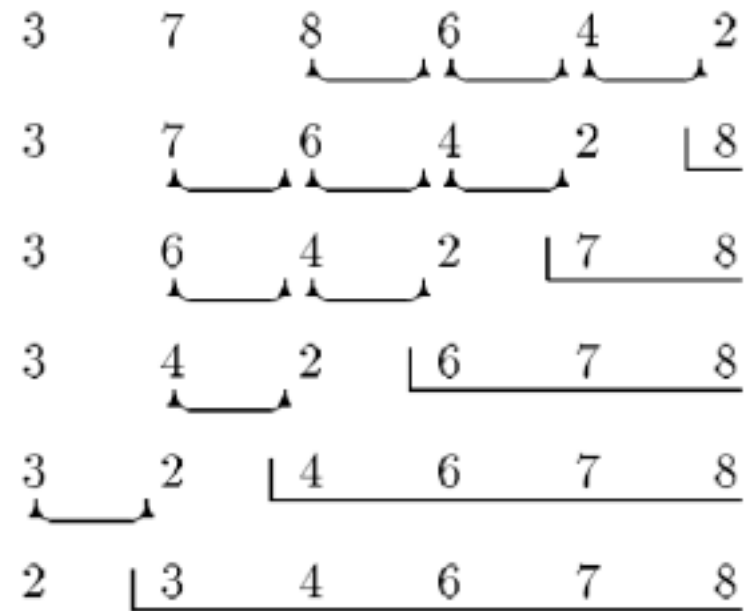
Komplexitätsanalyse (3)

- *Best Case* – Bei vollständig vorsortierten Folgen $2(N - 1)$ Bewegungen
- *Worst Case* – Bei umgekehrt sortierten Folgen $N^2/2$ Bewegungen
- *Average Case* – $\sim N^2/4$ Bewegungen

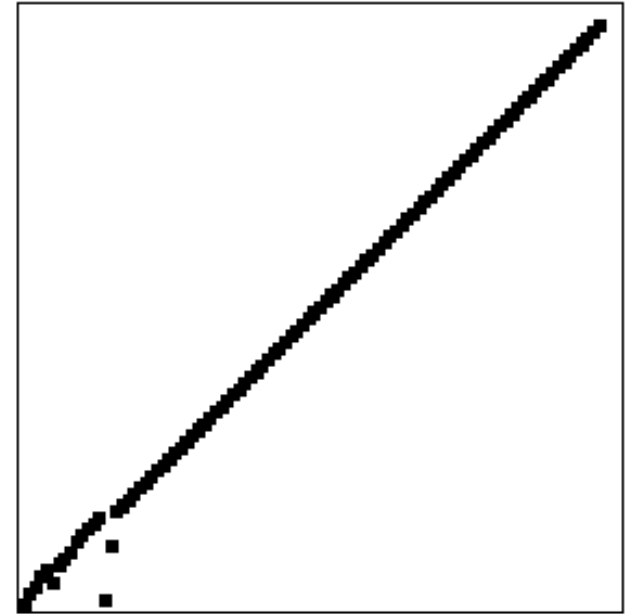
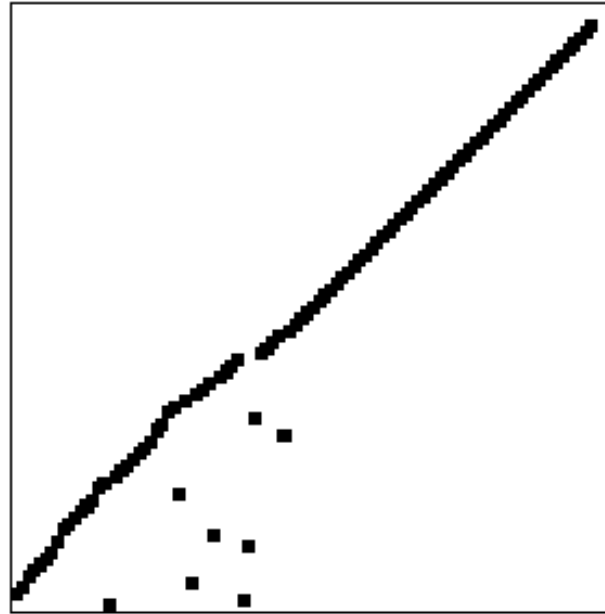
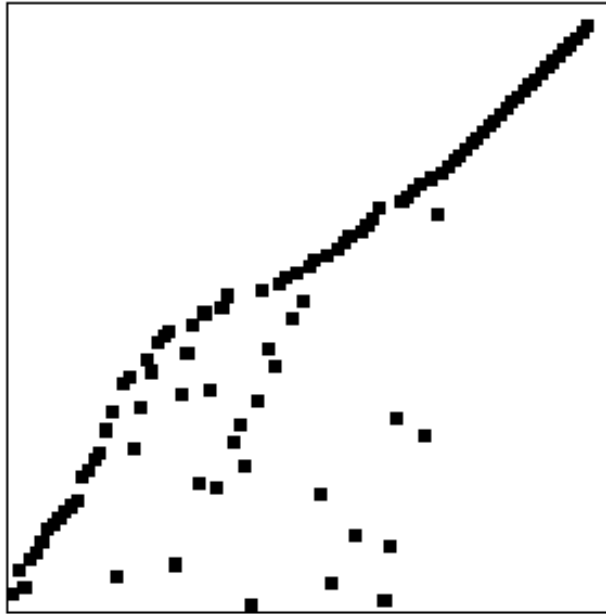
Für „fast sortierte“ Folgen verhält sich InsertionSort nahezu linear. Im Unterschied zu SelectionSort vermag InsertionSort somit eine in der zu sortierenden Datei bereits vorhandene Ordnung besser auszunutzen.

Bubblesort

```
■ function bubble-sort (a : array [1..n] of  $N_0$ ) : array [1..n] of  $N_0$ 
begin
  var i, j :  $N_0$ ;
  for i := n to 1 by -1 do
    for j := 2 to i do
      if  $a[j-1] > a[j]$ 
        then  $a[j-1], a[j] := a[j], a[j-1]$ 
      end
    end
  end
end ;
a
end
```



Eine Beispiellauf



Komplexitätsanalyse (1)

Vergleiche: Die Anzahl der Vergleiche ist unabhängig vom Vorsortierungsgrad der Folge. Daher sind der *worst case*, *average case* und *best case* identisch, denn es werden stets alle Elemente der noch nicht sortierten Teilfolge miteinander verglichen. Im i -ten Schleifendurchgang ($i = N, N - 1, \dots, 2$) enthält die noch unsortierte Anfangsfolge $N - i + 1$ Elemente, für die $N - i$ Vergleiche benötigt werden.

Um die ganze Folge zu sortieren, sind $N - 1$ Schritte erforderlich. Die Gesamtzahl der Vergleiche wächst damit quadratisch in der Anzahl der Schlüsselemente:

$$\begin{aligned}\sum_{i=1}^{N-1} (N - i) &= \sum_{i=1}^{N-1} i \\ &= \frac{N(N - 1)}{2}\end{aligned}$$

Komplexitätsanalyse (2)

Bewegungen: Aus der Analyse der Bewegungen für den gesamten Durchlauf ergeben sich:

- im *Best Case*: 0 Bewegungen
- im *Worst Case*: $\sim \frac{3N^2}{2}$ Bewegungen
- im *Average Case*: $\sim \frac{3N^2}{4}$ Bewegungen.

Vergleich elementarer Sortiervverfahren

■ Anzahl der Vergleiche:

Verfahren	Best Case	Average Case	Worst Case
SelectionSort	$N^2/2$	$N^2/2$	$N^2/2$
InsertionSort	N	$N^2/4$	$N^2/2$
BubbleSort	$N^2/2$	$N^2/2$	$N^2/2$

■ Anzahl der Bewegungen:

Verfahren	Best Case	Average Case	Worst Case
SelectionSort	$3(N - 1)$	$3(N - 1)$	$3(N - 1)$
InsertionSort	$2(N - 1)$	$N^2/4$	$N^2/2$
BubbleSort	0	$3N^2/4$	$3N^2/2$

Folgerungen

BubbleSort: ineffizient, da immer $N^2/2$ Vergleiche

InsertionSort: gut für fast sortierte Folgen

SelectionSort: gut für große Datensätze aufgrund konstanter Zahl der Bewegungen, jedoch stets $N^2/2$ Vergleiche

Fazit: InsertionSort und SelectionSort sollten nur für $N \leq 50$ eingesetzt werden.

Höhere Sortiervverfahren: Quicksort

QuickSort wurde 1962 von C.A.R. Hoare entwickelt.

Prinzip: Das Prinzip folgt dem Divide-and-Conquer-Ansatz:

Gegeben sei eine Folge F von Schlüsselementen.

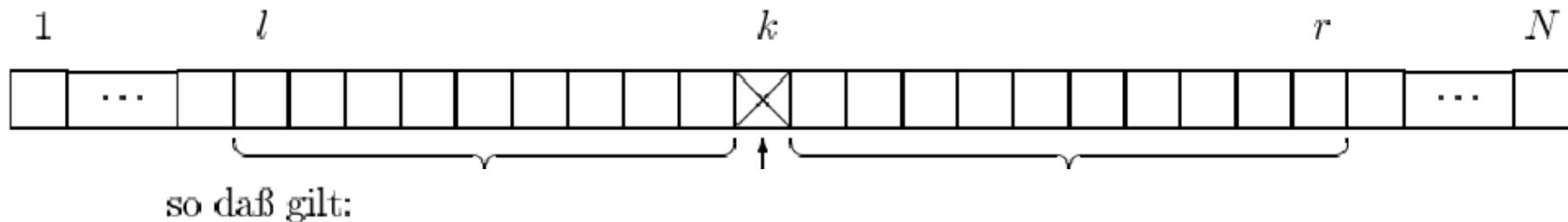
1. Zerlege F bzgl. eines partitionierenden Elementes (engl.: *pivot* = Drehpunkt) $p \in F$ in zwei Teilfolgen F_1 und F_2 , so daß gilt:

$$\begin{aligned}x_1 &\leq p & \forall x_1 \in F_1 \\p &\leq x_2 & \forall x_2 \in F_2\end{aligned}$$

2. Wende dasselbe Schema auf jede der so erzeugten Teilfolgen F_1 und F_2 an, bis diese nur noch höchstens ein Element enthalten.

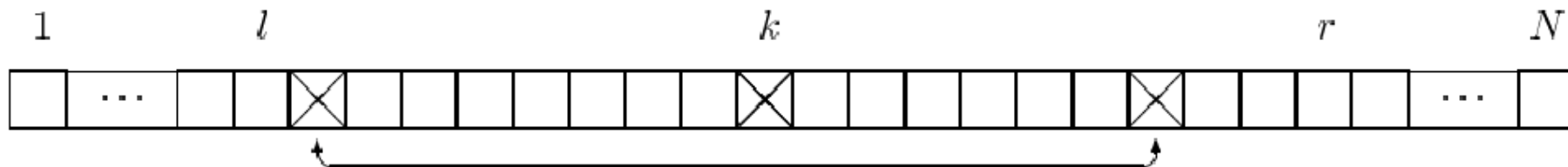
Quicksort: Kernidee

- **Ziel:** Zerlegung (Partitionierung) des Arrays $a[l..r]$ bzgl. eines Pivot-Elementes $a[k]$ in zwei Teilarrays $a[l..k-1]$ und $a[k+1..r]$



$$\begin{aligned}\forall i \in \{l, \dots, k-1\} : & \quad a[i] \leq a[k] \\ \forall j \in \{k+1, \dots, r\} : & \quad a[k] \leq a[j]\end{aligned}$$

- **Methode:** Austausch von Schlüsseln zwischen beiden Teilarrays



- **Rekursion:** linkes Teilarray $a[l..k-1]$ und rechtes Teilarray $a[k+1..r]$ bearbeiten

Quicksort

- $\text{quicksort}(a : \text{array } [1..n] \text{ of } N_0) : \text{array } [1..n] \text{ of } N_0$
 $\text{quicksort}'(a, 1, n)$
- $\text{quicksort}'(a : \text{array } [1..n] \text{ of } N_0; p, r : N_0) :$
 $\text{array } [1..n] \text{ of } N_0$

```
begin
  var q : NO;
  if p < r
    then q := partition(a, p, r);
         quicksort'(quicksort'(a, p, q), q+1, r)
    end if;
  a
end
```

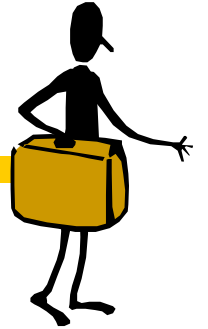
Partition

```
■ partition(a : array [1..n] of N0; p, r : N0) : N0
begin
  var x, i, j, result : N0;
  x, i, j, result := a[p], p - 1, r + 1, -1;
  while result < 0 do
    repeat j := j - 1 until a[j] <= x end repeat;
    repeat i := i + 1 until a[i] >= x end repeat;
    if i < j
      then a[i], a[j] := a[j], a[i]
      else result := j
    end if
  end while;
  result
end
```

Komplexitätsabschätzung

1. Schritt	<div style="border: 1px solid black; width: 100%; height: 60px; display: flex; align-items: center; justify-content: center;">N</div>
2. Schritt	<div style="display: flex; justify-content: space-between;"><div style="border: 1px solid black; width: 48%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{2}$</div><div style="border: 1px solid black; width: 4%; height: 60px;"></div><div style="border: 1px solid black; width: 48%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{2}$</div></div>
3. Schritt	<div style="display: grid; grid-template-columns: repeat(4, 1fr); gap: 2px;"><div style="border: 1px solid black; width: 24%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{4}$</div><div style="border: 1px solid black; width: 24%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{4}$</div><div style="border: 1px solid black; width: 24%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{4}$</div><div style="border: 1px solid black; width: 24%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{4}$</div></div>
4. Schritt	<div style="display: grid; grid-template-columns: repeat(8, 1fr); gap: 2px;"><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div><div style="border: 1px solid black; width: 12%; height: 60px; display: flex; align-items: center; justify-content: center;">$\frac{N}{8}$</div></div>
\vdots	\vdots
(ld N)-ter Schritt	<div style="display: flex; align-items: center; justify-content: space-around;"><div style="display: grid; grid-template-columns: repeat(6, 1fr); gap: 2px;"><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div></div><div style="flex-grow: 1; text-align: center; font-size: 2em;">...</div><div style="border: 1px solid black; width: 16.6%; height: 60px; display: flex; align-items: center; justify-content: center;">1</div></div>

Zusammenfassung, Kernpunkte



■ Einfache Sortierverfahren

- Sortieren durch Auswahl
- Sortieren durch Einfügen
- Sortieren durch paarweises Vertauschen (Bubblesort)

■ Höhere Sortierverfahren

- Quicksort

■ Komplexitätsabschätzung

- n^2 vs. $n \log n$
- Teile-und-herrsche-Prinzip

Was kommt beim nächsten Mal?



- Abstrakte Maschinen für spezielle Aufgaben
- Automatentheorie und Formale Sprachen