

# Algorithmen und Datenstrukturen

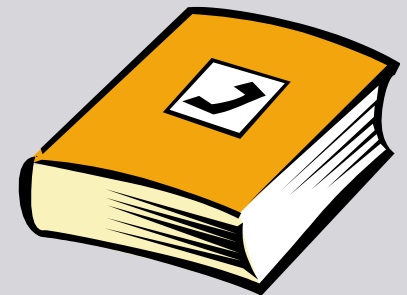
## Teil 3 - Algorithmen

Prof. Dr. Peter Jüttner

# Komplexe Algorithmen

## Sortierverfahren

- ***Sortieren*** bedeutet allgemein der Prozess des Anordnens einer gegebenen Menge von Daten in einer bestimmten ***Ordnung***.
- Sortiert wird, um zu einem späteren Zeitpunkt schnell nach einem bestimmten Element der Menge suchen zu können.
- Beispiele sortierter Datenmengen: Telefonbücher, Indexe, Wörterbücher



# Komplexe Algorithmen

## Sortiervverfahren

- gibt es in zahlreichen Variationen
- zeigen (exemplarisch) wie ein Problem auf vielfältige Weise gelöst werden kann
- bieten ein gutes Beispiel, die Leistung verschiedener Algorithmen miteinander zu vergleichen

# Komplexe Algorithmen

## Ordnung (Definition)

**Gegeben sei ein Array eines Types T der Länge n,  $n > 0$  (in C: `T array[n]`). Das Array ist bzgl. einer Ordnungsfunktion  $f(T)$  geordnet, wenn gilt:**

$$f(\text{Array}[0]) \leq f(\text{Array}[1]) \leq \dots \leq f(\text{Array}[n-1])$$

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Allgemeines

- Ausgangspunkt: Array der Länge  $n$ , das bzgl. einer Ordnung  $\leq$  zu sortieren ist.
  - Effizienz eines Sortieralgorithmus:
    - Anzahl der notwendigen Vergleiche
    - Anzahl der notwendigen Bewegungen (Austausch von Arrayelementen)
- als Funktion von  $n$  (Anzahl der Elemente des Arrays)

# Komplexe Algorithmen

## Sortiervverfahren für Arrays → Allgemeines

3	4	7	9	2	10	5	8	6	1
---	---	---	---	---	----	---	---	---	---



1	2	3	4	5	6	7	8	9	10
---	---	---	---	---	---	---	---	---	----

# Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

- **Sortieren durch Einfügen**
- **Sortieren durch Auswählen**
- **Sortieren durch Austauschen**

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

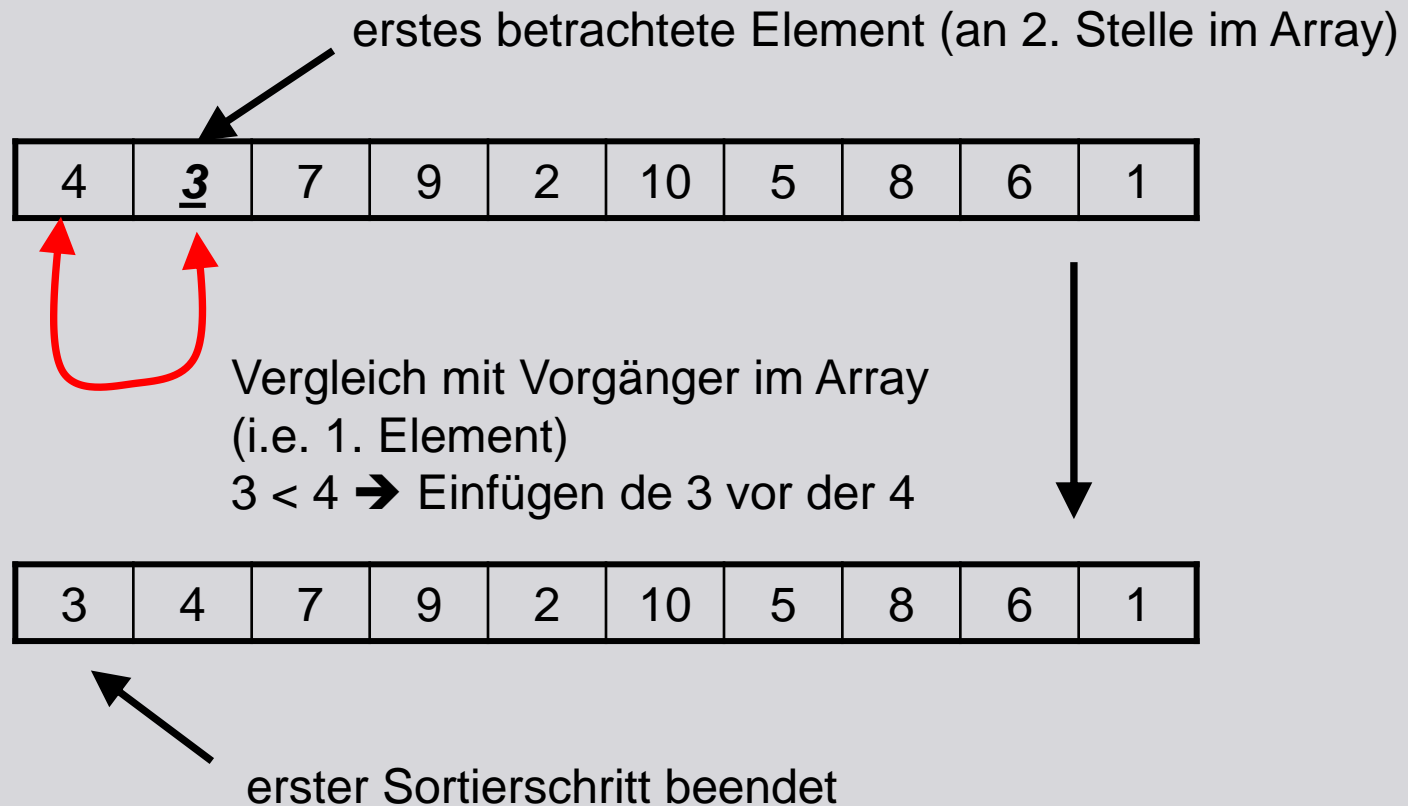
- **Sortieren durch Einfügen**
  - **Durchlaufe das Array elementweise, beginnend mit dem 2. bis zum n-ten Element.**
  - **Betrachte dabei jedes Element und sortiere es in die Sequenz der schon zuvor betrachteten Elemente ein. Dabei wird das einzusortierende Element mit den Vorgängern verglichen so lange die Vorgänger (Elemente mit kleinerem Arrayindex) größer sind oder der Anfang des Arrays erreicht ist.**
  - **Das Ende des kompletten Sortiervorgangs ist erreicht, wenn das letzte Element des Arrays betrachtet und ggf. einsortiert wurde.**



# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

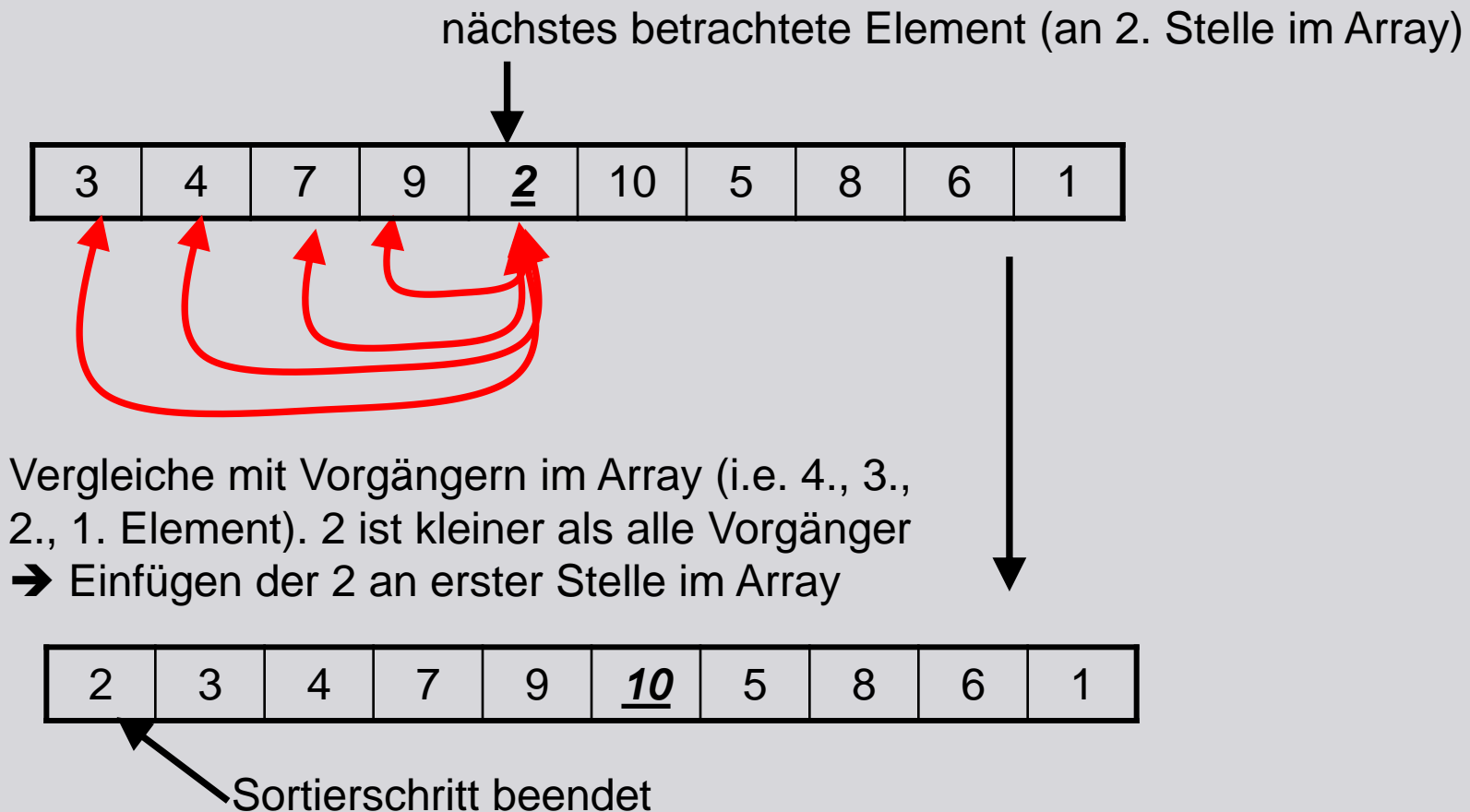
- Sortieren durch Einfügen → Beispiel



## Komplexe Algorithmen

### Sortierv Verfahren für Arrays → Einfache Verfahren

- Sortieren durch Einfügen → Beispiel



# Komplexe Algorithmen

## Sortiervverfahren für Arrays → Einfache Verfahren

### • Sortieren durch Einfügen → Beispiel



# Komplexe Algorithmen

## Sortierverfahren für Arrays → Einfache Verfahren

### • Sortieren durch Einfügen → Algorithmus

```
void sortieren_durch_einfuegen (int f[maxindex])
{ int laufindex; /* läuft durch Feld ab Index 2, bezeichnet das einzusortierende Element */
  int vergleichsindex; /* Index für Elemente, mit denen Element[laufindex] verglichen wird */
  int zwischenspeicher; /* zum Merken beim Verschieben */
  for (laufindex=2;laufindex<maxindex;laufindex++)
  { zwischenspeicher = f[laufindex]; f[0] = zwischenspeicher; /* Wert an Marke zuweisen */
    vergleichsindex = laufindex-1; /* Vergleich startet mit dem nächstkleineren Index */
    while (zwischenspeicher< f[vergleichsindex])
    { f[vergleichsindex+1] = f[vergleichsindex]; /* Verschieben der größeren Elemente */
      vergleichsindex = vergleichsindex-1;
    };
    f[vergleichsindex+1] = zwischenspeicher; /* Einsetzen des einzusortierenden Elements */
  };
}
```

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

- **Sortieren durch Einfügen → Algorithmus**

**Modifikation: Hinzufügen eines 0-ten Elements an das Arrays als Marke, die jeweils mit dem einzusortierenden Element besetzt wird. Dies erspart das Abfragen auf den Arrayindex bei jedem Vergleichsvorgang**

M	4	3	7	9	2	10	5	8	6	1
---	---	---	---	---	---	----	---	---	---	---

**→ Array wird um ein Element länger**

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

- **Sortieren durch Einfügen → Analyse des Algorithmus**
  - **Zahl der Vergleiche  $V[i]$  beim  $i$ -ten Durchlauf ist maximal  $i$  und minimal 1.**  
**Annahme: alle Permutationen des Felds sind gleich wahrscheinlich → im Mittel werden  $i/2$  Vergleiche durchgeführt.**
  - **Die Zahl der Bewegungen  $B[i]$  von Elementen beim  $i$ -ten Durchlauf ist  $V[i]+2$  (einschließlich des Zwischenspeicherns)**

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

• **Sortieren durch Einfügen → Analyse des Algorithmus**

$$\begin{aligned} \rightarrow V_{\text{minimal}} &= n-1 & B_{\text{minimal}} &= 3 \cdot (n-1) \\ \rightarrow V_{\text{maximal}} &= (n^2+n)/2 - 1 & B_{\text{maximal}} &= (n^2+5 \cdot n-6)/2 \end{aligned}$$

- **kleinste Werte, wenn das Feld bereits geordnet ist, größte Werte, falls das Feld in umgekehrter Reihenfolge sortiert ist.**

$$\rightarrow V_{\emptyset} = (n^2+3n-4)/4$$

**→ Aufwand der Ordnung  $n^2$**

## Komplexe Algorithmen

### Sortierv Verfahren für Arrays → Einfache Verfahren

- **Sortieren durch Einfügen → Analyse des Algorithmus (Herleitung)**

→  $V_{\text{minimal}} = n-1$ , da  $n-1$  Zeichen einsortiert werden und für jedes Zeichen minimal ein Vergleich anfällt.

⇒ Anzahl der Minimalen Bewegungen ist  $= (V_{\text{min}}[i] + 2) * (n-1) = 3 * (n-1)$

→  $V_{\text{maximal}} = (n^2 + n) / 2 - 1$ , da  $n-1$  Zeichen einsortiert werden und für für das  $i$ -te Zeichen maximal  $i$  Vergleiche anfallen ( $1 \leq i \leq n$ )

⇒  $V_{\text{maximal}} = 1 + 2 + 3 + 4 + \dots + n-1 = (n^2 + n) / 2 - 1$



## Komplexe Algorithmen

### Sortierv Verfahren für Arrays → Einfache Verfahren

- Sortieren durch Einfügen → Analyse des Algorithmus (Herleitung)

→  $B_{\text{maximal}}$  = Summe der maximalen Bewegungen für jedes Zeichen =  
Summe der maximalen Vergleiche für jedes Zeichen + 2 \* (Anzahl  
der einsortierten Zeichen)  
 $= (n^2+n)/2 - 1 + 2*n-2 = (n^2+n-2+4*n-4)/2 = (n^2+5*n-6)/2$

# Übung

## Rekursive Implementierung des Sortierens durch Einfügen

### Beschreibung:

In der Vorlesung wurde eine nicht-rekursive Version des Sortierens durch Einfügen vorgestellt.

### Aufgabe:

Entwickeln Sie eine rekursive Version des Algorithmus. Überlegen Sie dazu zunächst wie das Sortieren rekursiv gelöst werden kann. Implementieren Sie dann den entsprechenden Algorithmus in C.



## Komplexe Algorithmen

Zum Schluss dieses Abschnitts ...

**Noch Fragen ??**

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

- Sortieren durch binäres Einfügen
  - Verbesserung des direkten Einfügens durch Nutzen der Tatsache, dass die Sequenz, in die das gerade betrachtete Element eingefügt wird, bereits sortiert ist.

4	<u>3</u>	7	9	2	10	5	8	6	1
3	4	<u>7</u>	9	2	10	5	8	6	1
3	4	7	<u>9</u>	2	10	5	8	6	1
3	4	7	9	<u>2</u>	10	5	8	6	1
2	3	4	7	9	<u>10</u>	5	8	6	1
2	3	4	7	9	10	<u>5</u>	8	6	1
2	3	4	5	7	9	10	<u>8</u>	6	1
2	3	4	5	7	8	9	10	<u>6</u>	1
2	3	4	5	6	7	8	9	10	<u>1</u>
1	2	3	4	5	6	7	8	9	10

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

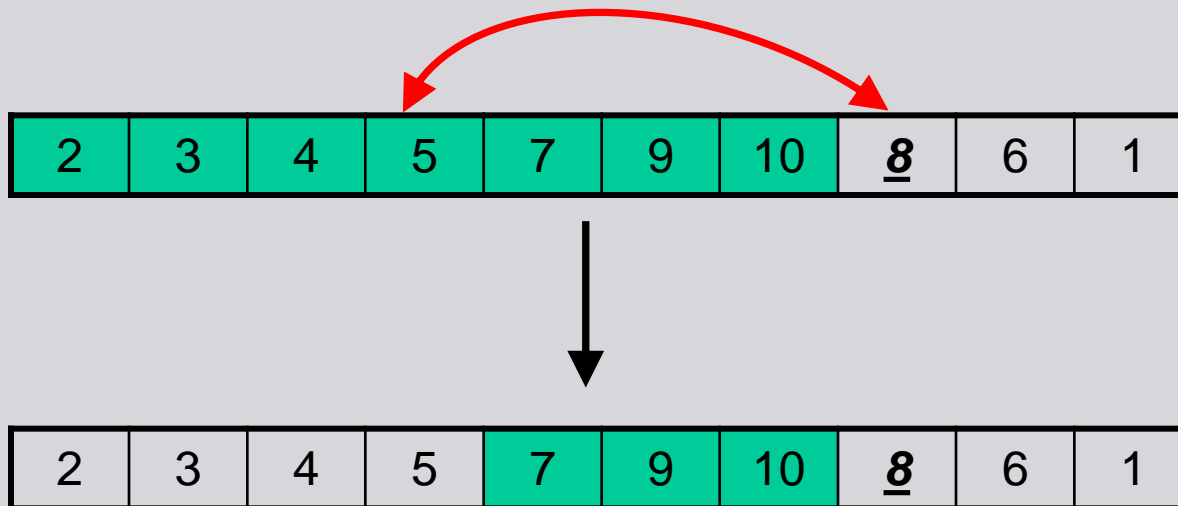
- Sortieren durch binäres Einfügen
  - Binäre Suche der Einfügestelle anstatt lineare Suche, d.h. Vergleich mit dem mittleren Element der bereits sortierten Sequenz. Falls einzufügendes Element kleiner, dann binäre Suche im vorderen Teil, sonst binäre Suche im hinteren Teil.


2	3	4	5	7	9	10	<u>8</u>	6	1
---	---	---	---	---	---	----	----------	---	---

# Komplexe Algorithmen

## Sortierverfahren für Arrays → Einfache Verfahren

- Sortieren durch binäres Einfügen
  - Beispiel: Einfügen des 8. Elements (mit dem Wert 8) in die bereits sortierte Frequenz der Elemente 1..7
  - 1. Schritt: Vergleich mit dem 4. Element (Wert 5) → Einfügen in der Sequenz 5. bis 7. Element

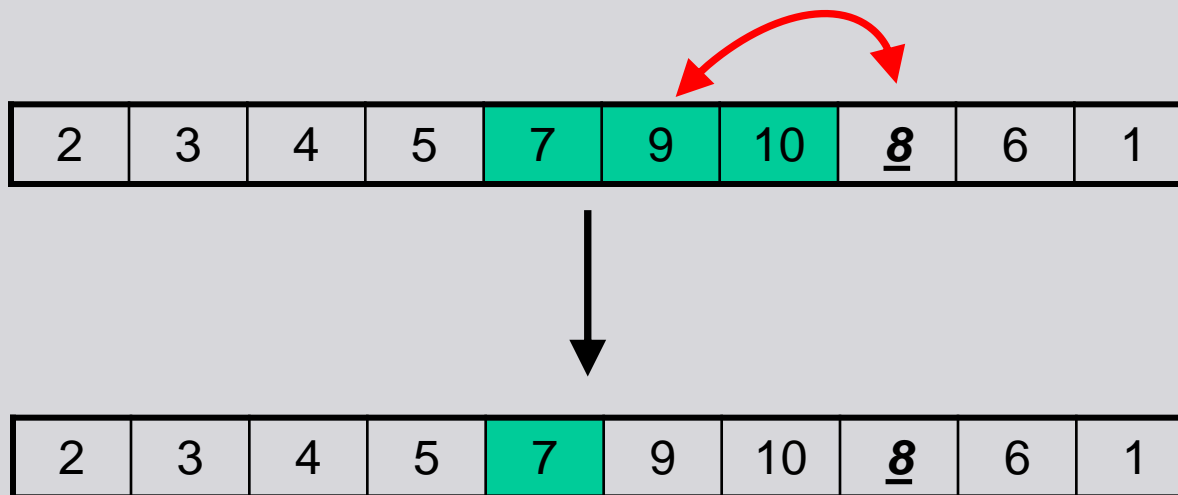



 = Teilsequenz, in die einsortiert wird.

# Komplexe Algorithmen

## Sortiervverfahren für Arrays → Einfache Verfahren

- Sortieren durch binäres Einfügen
  - Beispiel: Einfügen des 8. Elements (mit dem Wert 8) in die bereits sortierte Frequenz der Elemente 1..7
  - 2. Schritt: Vergleich mit dem 6. Element (Wert 9) → Einfügen in der Sequenz 5. bis 5. Element

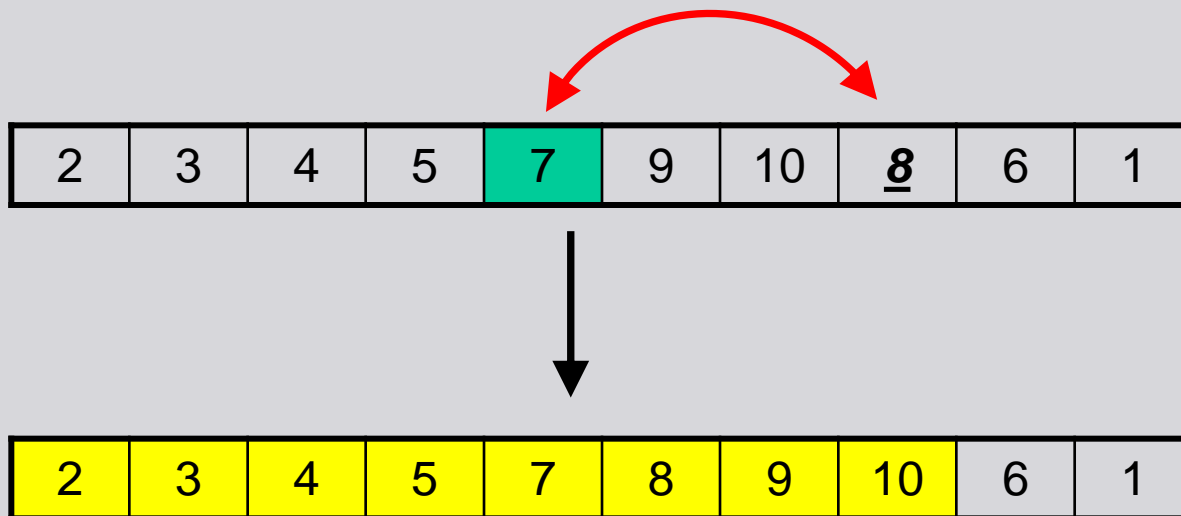



 = Teilsequenz, in die einsortiert wird.


# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

- Sortieren durch binäres Einfügen
    - Beispiel: Einfügen des 8. Elements (mit dem Wert 8) in die bereits sortierte Frequenz der Elemente 1..7
3. Schritt: Vergleich mit Element 5 (Wert 7)  
→ Einfügen nach dem 5. Element



 = Teilsequenz, in die einsortiert wird.

 = neue bereits sortierte Teilsequenz



# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

### • Sortieren durch binäres Einfügen → Algorithmus

```
void sortieren_mit_binärem_einfügen (int f[maxindex])
{ int laufindex; int index; int zwischenspeicher; /* zum Merken beim Verschieben */
  int links; int rechts; /* Grenzen des Teilarrays in das binär einsortiert wird */
  for (laufindex=1;laufindex<maxindex;laufindex++)
  { zwischenspeicher = f[laufindex];
    links = 0; rechts = laufindex-1;
    while (links<=rechts)
    { index = (links+rechts) / 2;
      if (zwischenspeicher < f[index])
        rechts = index-1; else links = index+1;
    };
    for (index=laufindex-1;index>=links;index--) f[index+1] = f[index];
    f[links]= zwischenspeicher;
  };
};
```

}

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

- **Sortieren durch binäres Einfügen → Analyse des Algorithmus**

- **Zahl der Vergleiche beim i-ten Durchlauf:  $\text{ganz}(\log_2 i)$  (ganz = nächstgrößere ganze Zahl)**

**→ Gesamtzahl der Vergleiche**

$$\sum_{i=1}^n \text{ganz}(\log_2 i) \approx \int_1^n \log_2 x dx = n(\log_2 n - c) + c$$

**mit  $c = 1/\ln 2$**

**→ Anzahl der Vergleiche wächst mit dem Produkt aus  $n$  und  $\log_2 n$**

## Komplexe Algorithmen

**Sortierverfahren für Arrays → Einfache Verfahren**

- **Sortieren durch binäres Einfügen → Analyse des Algorithmus**
  - **Die Zahl der Bewegungen  $B[i]$  von Elementen beim  $i$ -ten Durchlauf ist  $V[i]+2$  (einschließlich des Zwischenspeicherns)**
  - **Aufwand für Bewegungen wie zuvor, wächst mit  $n$  im Quadrat**
  - **Aufwand für Bewegungen überwiegt gegenüber dem Aufwand für Vergleiche**
  - **Gesamtaufwand wächst mit  $n$  im Quadrat**

## Komplexe Algorithmen

Zum Schluss dieses Abschnitts ...

**Noch Fragen ??**

## Komplexe Algorithmen

### Sortierverfahren für Arrays → Einfache Verfahren

- Sortieren durch direktes Auswählen
  - Prinzip:
    - Auswahl des kleinsten Elements im noch nicht sortierten Teil des Arrays
    - Austausch mit dem ersten Element des noch nicht sortierten Teilarrays
    - Analog mit dem Rest des Arrays


# Komplexe Algorithmen

Sortiervverfahren für Arrays → Einfache Verfahren

• Sortieren durch direktes Auswählen → Beispiel

Sortierschritte

4	3	7	9	2	10	5	8	6	1
1	4	3	7	9	2	10	5	8	6
1	2	4	3	7	9	10	5	8	6
1	2	3	4	7	9	10	5	8	6
1	2	3	4	5	7	9	10	8	6
1	2	3	4	5	6	7	9	10	8
1	2	3	4	5	6	7	9	10	8
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

 = bereits sortierte Folge

## Komplexe Algorithmen

### Sortierverfahren für Arrays → Einfache Verfahren

- **Sortieren durch direktes Auswählen**
  - „Gegenstück“ zum Sortieren durch Einfügen
  - **Sortieren durch Auswählen**
    - Auswahl aus allen zu sortierenden Elementen
    - Einfügen am Anfang des zu sortierenden Arrays
- **Sortieren durch Einfügen**
  - Auswahl am Anfang des zu sortierenden Arrays
  - Einfügen in die Sequenz der bereits sortierten Elemente

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

- Sortieren durch direktes Auswählen

## Sortieren durch Auswählen vs. Sortieren durch Einfügen

4	3	7	9	2	10	5	8	6	1
1	4	3	7	9	2	10	5	8	6
1	2	4	3	7	9	10	5	8	6
1	2	3	4	7	9	10	5	8	6
1	2	3	4	5	7	9	10	8	6
1	2	3	4	5	6	7	9	10	8
1	2	3	4	5	6	7	9	10	8
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	8	9	9	10
1	2	3	4	5	6	7	8	9	10

4	<u>3</u>	7	9	2	10	5	8	6	1
3	4	<u>7</u>	9	2	10	5	8	6	1
3	4	7	<u>9</u>	2	10	5	8	6	1
3	4	7	9	<u>2</u>	10	5	8	6	1
2	3	4	7	9	<u>10</u>	5	8	6	1
2	3	4	7	9	10	<u>5</u>	8	6	1
2	3	4	5	7	9	10	<u>8</u>	6	1
2	3	4	5	7	8	9	10	<u>6</u>	1
2	3	4	5	6	7	8	9	10	<u>1</u>
1	2	3	4	5	6	7	8	9	10



# Komplexe Algorithmen

## Sortierverfahren für Arrays → Einfache Verfahren

### • Sortieren durch direktes Auswählen → Algorithmus

```
void sortieren(int f[maxindex])
{ int lindex; /* läuft durch Feld ab Position 1 */
  int sindex; /* läuft durch das noch zu sortierende Feld */
  int zspeicher; /* speichert das kleinste Element beim Durchlauf */
  int zindex; /* speichert den Index des kleinsten Elements */
  for (lindex=0;lindex<maxindex;lindex++)
  { zspeicher = f[lindex]; zindex = lindex;
    for (sindex=lindex+1; sindex<maxindex;sindex++)
    { if (f[sindex]<zspeicher)
        { zspeicher = f[sindex]; zindex = sindex; };
    };
    zspeicher = f[lindex]; f[lindex] = f[zindex]; f[zindex] = zspeicher;
  };
};
```

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

- **Sortieren durch direktes Auswählen → Analyse des Algorithmus**
  - **Zahl der Vergleiche beim i-ten Durchlauf: n-i**  
**n-1 Durchläufe werden durchgeführt**  
**→ Anzahl der Vergleiche:**

$$\sum_{i=1}^{n-1} \left( \sum_{j=i+1}^n 1 \right) = \sum_{i=1}^{n-1} (n-i) = \sum_{i=1}^{n-1} n - \sum_{i=1}^{n-1} i =$$

$$(n-1)*n - ((n-1)^2 + (n-1))/2$$

**→  $(n^2-n)/2$  Vergleiche sind notwendig**

## Komplexe Algorithmen

**Sortierverfahren für Arrays → Einfache Verfahren**

- **Sortieren durch direktes Auswählen → Analyse des Algorithmus**
    - **Zahl der Bewegungen**  
minimal  $3 \cdot (n-1)$ , falls Array bereits geordnet  
maximal  $\lceil n^2/4 \rceil + 3 \cdot (n-1)$ , falls das Array umgekehrt sortiert ist  
Mittel  $\sim n \cdot (\ln n + g)$  (g Eulersche Konstante) \*)
- das Verfahren der direkten Auswahl ist günstiger als Einfügeverfahren**

\*) die Herleitung dieser Formel ist komplex

## Komplexe Algorithmen

Zum Schluss dieses Abschnitts ...

**Noch Fragen ??**

## Komplexe Algorithmen

### Sortierv Verfahren für Arrays → Einfache Verfahren

- Sortieren durch direktes Austauschen
  - Prinzip:
    - Mehrfaches Durchlaufen des Arrays (wie bisher)
    - Fortgesetztes Austauschen nebeneinander liegender Elemente im Array falls das hintere Element kleiner als das vordere ist.

Anmerkung: Wird das zu sortierende Array vertikal angeordnet (kleinstes Element oben), dann „steigen“ die kleineren Element wie Luftblasen in einer Flüssigkeit nach oben. Deshalb heißt dieses Verfahren auch

**Bubblesort**

# Komplexe Algorithmen

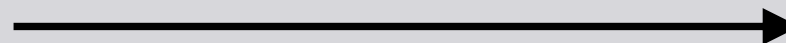
## Sortiervverfahren für Arrays → Einfache Verfahren

### • Sortieren durch direktes Austauschen → Beispiel

4	1	1	1	1	1
3	4	2	2	2	2
7	3	4	3	3	3
9	7	3	4	4	4
2	9	7	5	5	5
10	2	9	7	6	6
5	10	5	9	7	7
8	5	10	6	9	8
6	8	6	10	8	9
1	6	8	8	10	10



= Elemente,  
die „nach  
oben“  
getauscht“  
werden



Sortierschritte

# Komplexe Algorithmen

## Sortierverfahren für Arrays → Einfache Verfahren

### • Sortieren durch direktes Austauschen (Bubblesort) → Algorithmus

```
void bubblesort(int f[maxindex])
{ int laufindex; /* läuft durch Feld ab Position 1 */
  int suchindex; /* läuft durch das noch zu sortierende Feld */
  int zwischenspeicher; /* speichert das kleinste Element beim Durchlauf */
  for (laufindex=1; laufindex<maxindex; laufindex++)
  { for (suchindex=maxindex-1; suchindex>=laufindex; suchindex--)
    { if (f[suchindex]<f[suchindex-1])
      { zwischenspeicher = f[suchindex];
        f[suchindex] = f[suchindex-1];
        f[suchindex-1] = zwischenspeicher;
      };
    };
  };
};
```

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Einfache Verfahren**

- **Sortieren durch direktes Austauschen (Bubblesort) →**

**Analyse des Algorithmus**

- **Zahl der Vergleiche beim i-ten Durchlauf:  $n-i$   
 $n-1$  Durchläufe werden durchgeführt**

**→ Anzahl der Vergleiche:  $(n^2-n)/2$**

- **Anzahl der Bewegungen**

- **minimal 0**

- **maximal  $3*(n^2-n)/2$**

- **im Durchschnitt  $3*(n^2-n)/4$**

**→ Sortieren durch direktes Austauschen ist den vorherigen Verfahren unterlegen!**



## Komplexe Algorithmen

Zum Schluss dieses Abschnitts ...

**Noch Fragen ??**

# Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Komplexe Verfahren**

- **Quicksort**
- **Mergesort**

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Komplexe Verfahren**

- **Sortieren durch Zerlegen (Partitionieren) → Quicksort**
  - **Prinzip:**
    - **Wähle ein beliebiges Element  $e$  aus dem Array**
    - **Zerlege das zu sortierende Array in zwei Teile, so dass im linken Teil alle Elemente kleiner  $e$  sind und im rechten Teil alle Elemente größer oder gleich  $e$ .**
    - **Bearbeite auf diese Weise nun die Teilarrays**
    - **Führe diesen Prozeß fort, bis die Teilarrays nur noch ein Element besitzen, damit ist die Sortierung beendet**

# Komplexe Algorithmen

Sortierv Verfahren für Arrays → Komplexe Verfahren

- Sortieren durch Zerlegen (Partitionieren) → Quicksort, Beispiel für Methode

Sortierschritte ↓

4	3	7	9	<u>2</u>	10	5	8	6	1
1	2	7	9	3	<u>10</u>	5	8	6	4
1	2	7	9	<u>3</u>	4	5	8	6	10
1	2	3	9	7	<u>4</u>	5	8	6	10
1	2	3	4	7	9	<u>5</u>	8	6	10
1	2	3	4	5	9	<u>7</u>	8	6	10
1	2	3	4	5	6	7	8	9	10

i = Vergleichselement für nächste Zerlegung eines Teilarrays

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Komplexe Verfahren

### • Sortieren durch Zerlegen → Quicksort

```
void quicksort(int f[maxindex],int links, int rechts) /* Startwerte Arraygrenzen für links und rechts */  
{ int w,x; int i,j;  
  i=links; j=rechts;  
  x=f[(links+rechts)/2];  
  do { while (f[i]<x) i++; while (x<f[j]) j--;  
      if (i<=j)  
      { w=f[i]; f[i]=f[j]; f[j]=w;  
        i++; j--;  
      };  
  } while (i<=j);  
  if (links < j) quicksort(f,links,j); /* Sortieren Teilarray, falls notwendig */  
  if (i<rechts) quicksort(f,i,rechts); /* Sortieren Teilarray, falls notwendig */  
};
```

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Komplexe Verfahren**

- **Sortieren durch Zerlegen → Quicksort Analyse des Algorithmus**

- **Anzahl Vergleiche:**

- **schlechtester Fall:  $n+2$  Vergleiche bei einem Durchlauf und Partitionierung immer an den Rand gelegt → Aufwand  $O(n^2)$** 
  - **Quicksort in diesem Fall ineffizient**

## Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Komplexe Verfahren**

- **Sortieren durch Zerlegen → Quicksort, Analyse des Algorithmus**
  - **Anzahl Vergleiche:**
    - **durchschnittlicher („normaler“) Fall Aufwand  $O(n \ln n)$**

## Übung

# Rekursive Implementierung des Sortierens durch Einfügen

### Beschreibung:

In der Vorlesung wurde die Sortiermethode Quicksort besprochen.

### Aufgabe:

Erweitern Sie den Algorithmus von Quicksort so, dass bei der Sortierung die Anzahl der Vergleiche und Vertauschungen mitgezählt und am Ende der Sortierung ausgegeben werden.

1. Sortieren Sie die folgenden Arrays mit 10 Elementen mit Quicksort und vergleichen Sie die Ergebnisse:

[1,2,3,4,5,6,7,8,9,10]

[10,9,8,7,6,5,4,3,2,1]

[1,1,1,1,1,1,1,1,1,1]

[3,6,4,2,10,8,9,1,7,5,]

2. Sortieren Sie diese Arrays mit einer der anderen Sortiermethoden und vergleichen Sie die Ergebnisse mit dem Ergebnis von Quicksort





## Komplexe Algorithmen

Zum Schluss dieses Abschnitts ...

**Noch Fragen ??**

## Komplexe Algorithmen

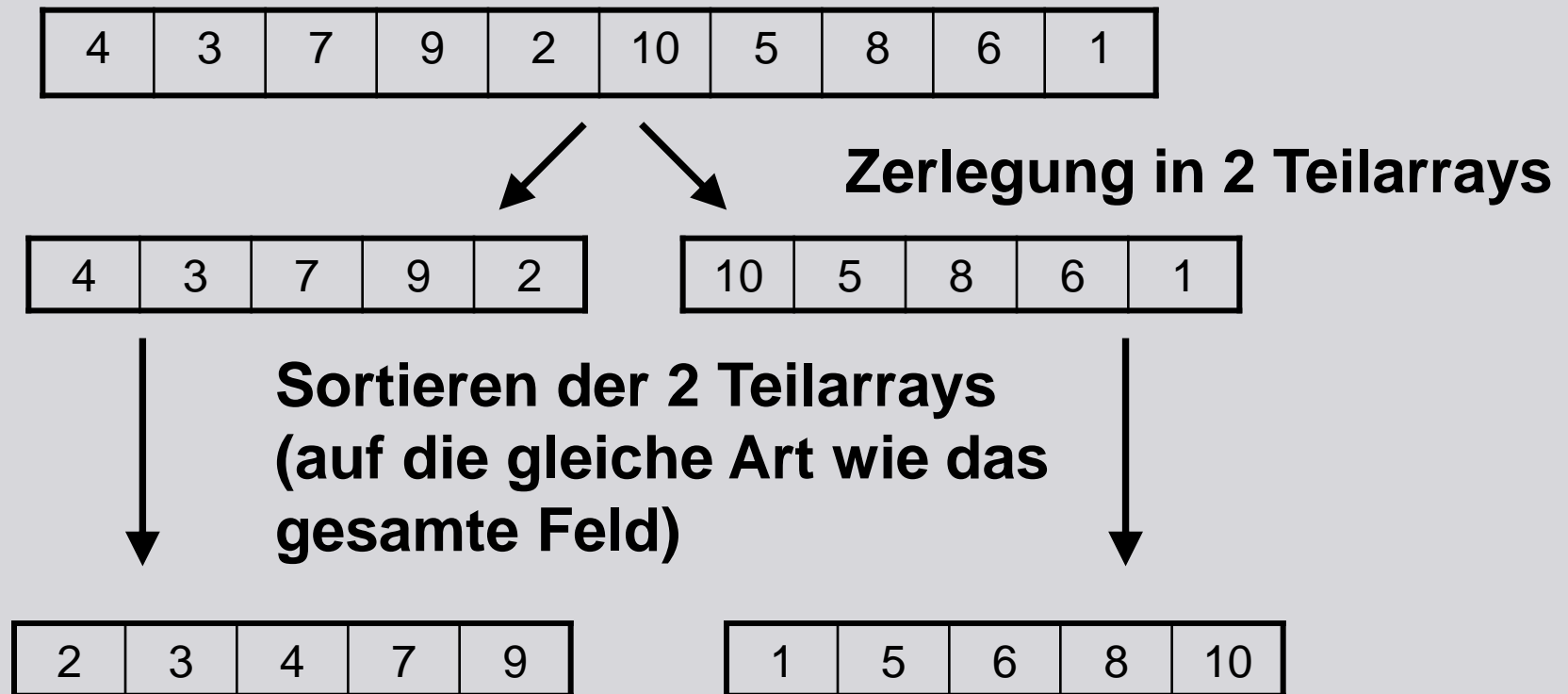
**Sortierverfahren für Arrays → Komplexe Verfahren**

- **Sortieren durch Zerlegen (Partitionieren) → Mergesort**
  - **Prinzip:**
    - **Zerlege das zu sortierende Array in zwei Teile, sortiere diese durch weitere Zerlegung**
    - **Füge die sortierten Teile elementweise zusammen, so dass die neue Sequenz sortiert ist**
    - **Führe diesen Prozeß fort, bis die Teilarrays nur noch ein Element besitzen, damit ist die Sortierung beendet**

# Komplexe Algorithmen

**Sortierv Verfahren für Arrays → Komplexe Verfahren**

- **Sortieren durch Zerlegen (Partitionieren) → Mergesort**  
**Beispiel**

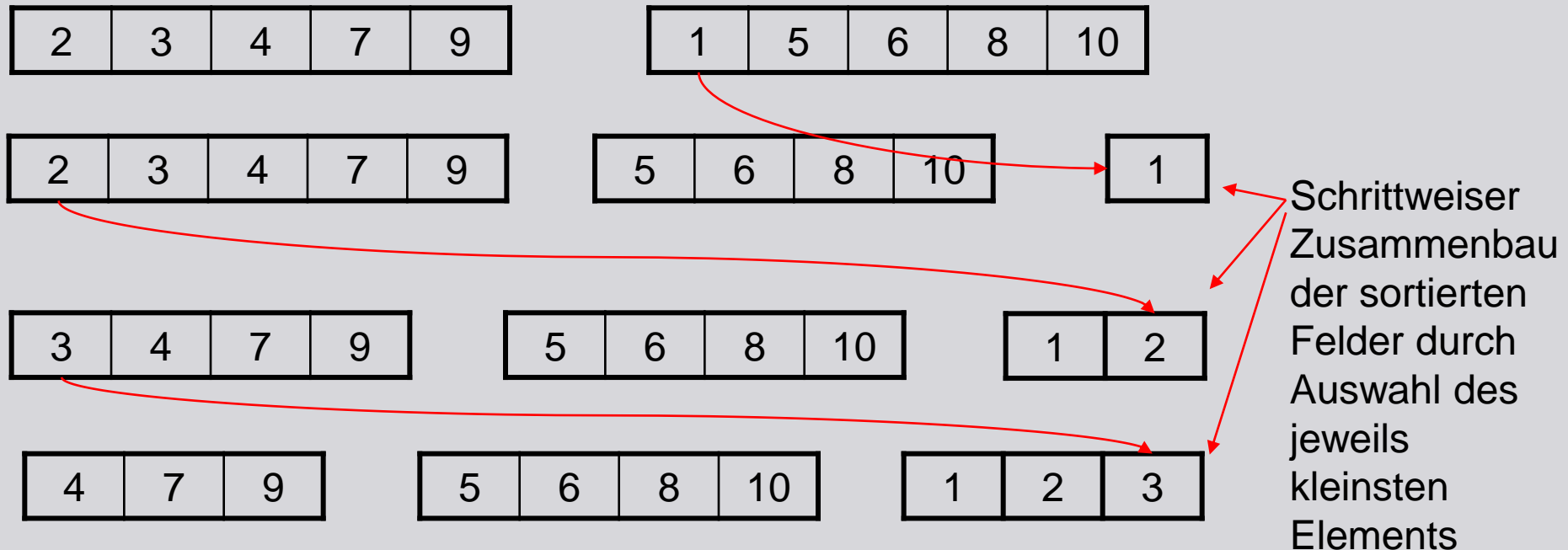


# Komplexe Algorithmen

## Sortierverfahren für Arrays → Komplexe Verfahren

- Sortieren durch Zerlegen (Partitionieren) → Mergesort

### Beispiel



# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

### • Sortieren durch Zerlegen (Partitionieren) → Mergesort

```
void mergesort(int f[maxindex])  
{ int hilfsfeld[maxindex]; /* dient zum Zwischenspeichern */  
  mergesort2(f,hilfsfeld,0,maxindex-1);  
};
```

```
void mergesort2(int f[maxindex], int h[maxindex], int links, int rechts)  
{ int mitte;  
  if (links<rechts)  
  { mitte = (links+rechts)/2;  
    mergesort2(f,h,links,mitte);  
    mergesort2(f,h,mitte+1,rechts);  
    merge(f,h,links,mitte,rechts);  
  };  
};
```

# Komplexe Algorithmen

## Sortierv Verfahren für Arrays → Einfache Verfahren

### • Sortieren durch Zerlegen (Partitionieren) → Mergesort

```
void merge(int f[maxindex], int h[maxindex], int links, int mitte, int rechts)
{
    int i = links; int j = mitte+1; int h_position = links;
    while ((i<=mitte) && (j<=rechts))
    {
        if (f[i] <= f[j])
        {
            h[h_position] = f[i]; i++;
        }
        else
        {
            h[h_position] = f[j]; j++; h_position++;
        }
    }
    while (i<=mitte)
    {
        h[h_position] = f[i]; i++; h_position++;
    }
    while (j<=rechts)
    {
        h[h_position] = f[j]; j++; h_position++;
    }
    for (h_position = links; h_position<=rechts; h_position++)
        f[h_position] = h[h_position];
};
```

## Komplexe Algorithmen

**Sortierverfahren für Arrays → Komplexe Verfahren**

- **Sortieren durch Zerlegen (Partitionieren) → Mergesort**

**Analyse des Algorithmus**

**Anzahl der Vergleiche:  $O(n \log n)$**

## Komplexe Algorithmen

**Sortierverfahren für Arrays → Komplexe Verfahren**

**Anmerkung**

**Sowohl Quicksort als auch Mergesort zerlegen (rekursiv) eine Array in sortierte Teilarrays, die anschließend wieder zusammengefügt werden.**

**Bei Quicksort erfordert das Zerlegen Aufwand, während das Zusammenfügen einfach ist.**

**Im Gegensatz dazu ist bei Mergesort das Zerlegen einfach, aber der Zusammenbau ist aufwändig.**



## Komplexe Algorithmen

Zum Schluss dieses Abschnitts ...

**Noch Fragen ??**