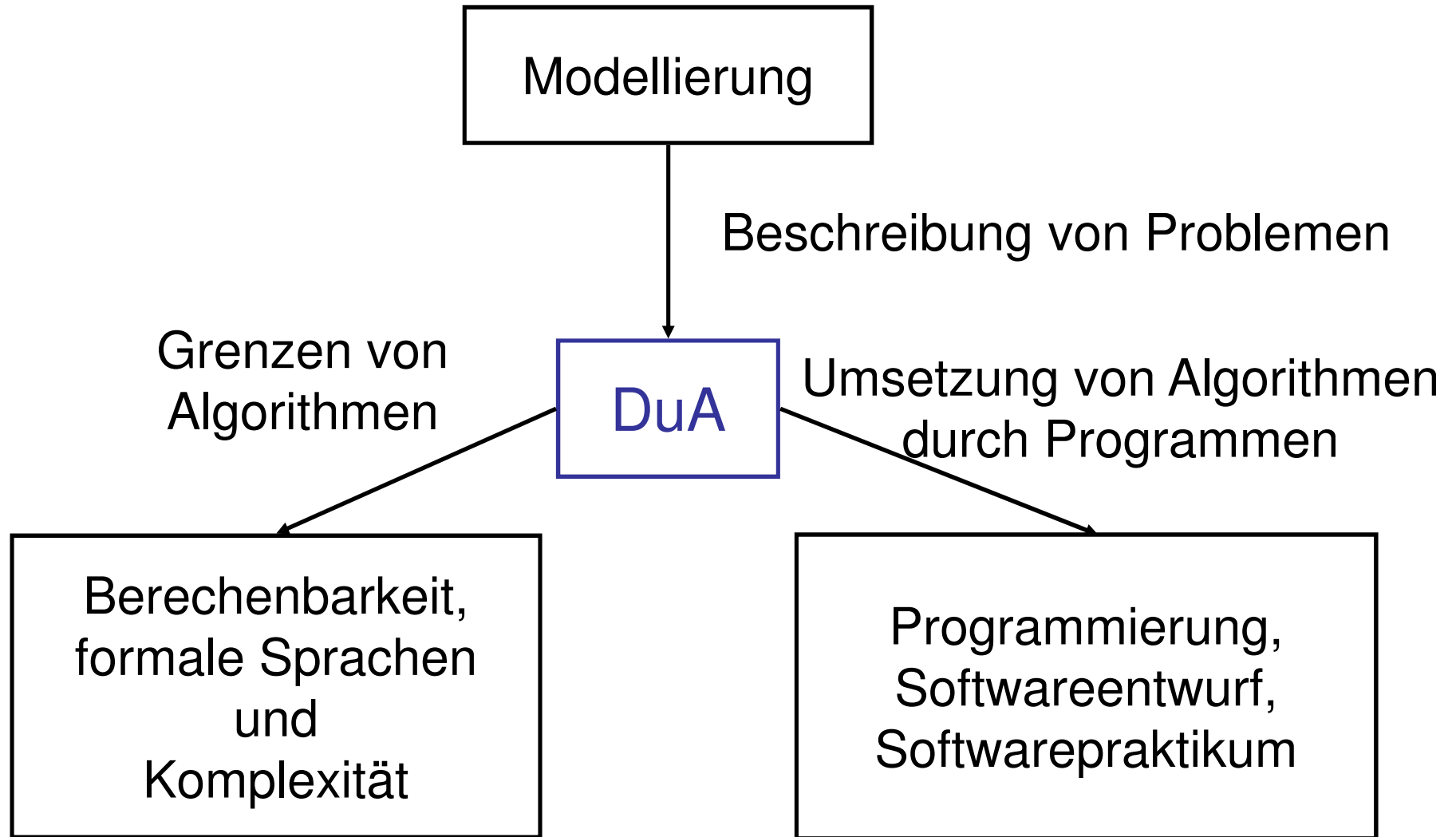


Wiederholung



Wiederholung

- Beschreibung von Algorithmen durch *Pseudocode*.
- Korrektheit von Algorithmen durch *Invarianten*.
- Laufzeitverhalten beschreiben durch *O-Notation*.
 - Wollen uns auf *asymptotische Laufzeit* konzentrieren.
 - Werden in Zukunft Laufzeiten immer mit Hilfe von O -, Ω -, Θ -Notation angeben.

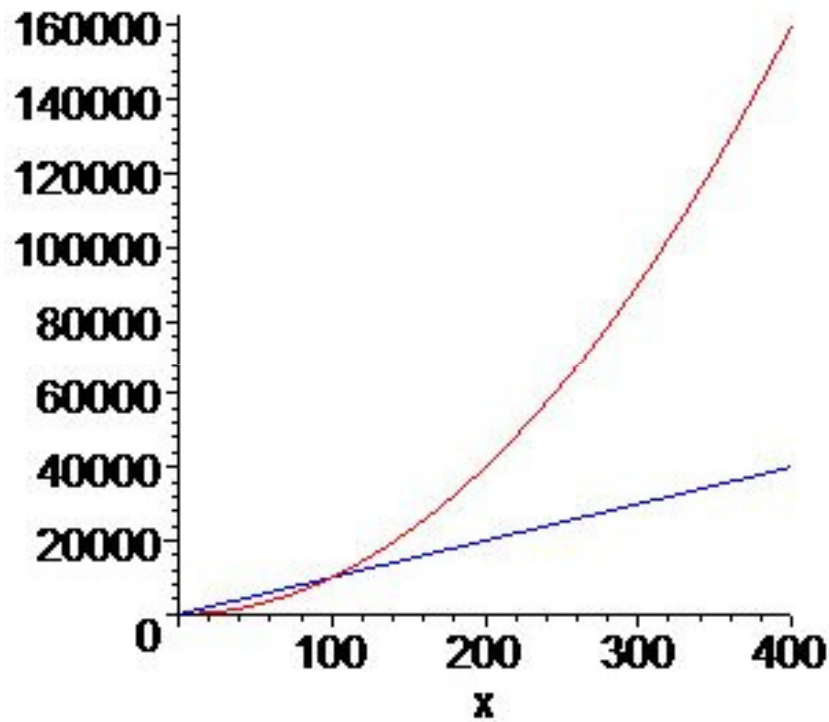
O-Notation

Definition 2.6.: Sei $g : \mathbb{N} \rightarrow \mathbb{R}^+$ eine Funktion. Dann bezeichnen wir mit $O(g(n))$ die folgende Menge von Funktionen

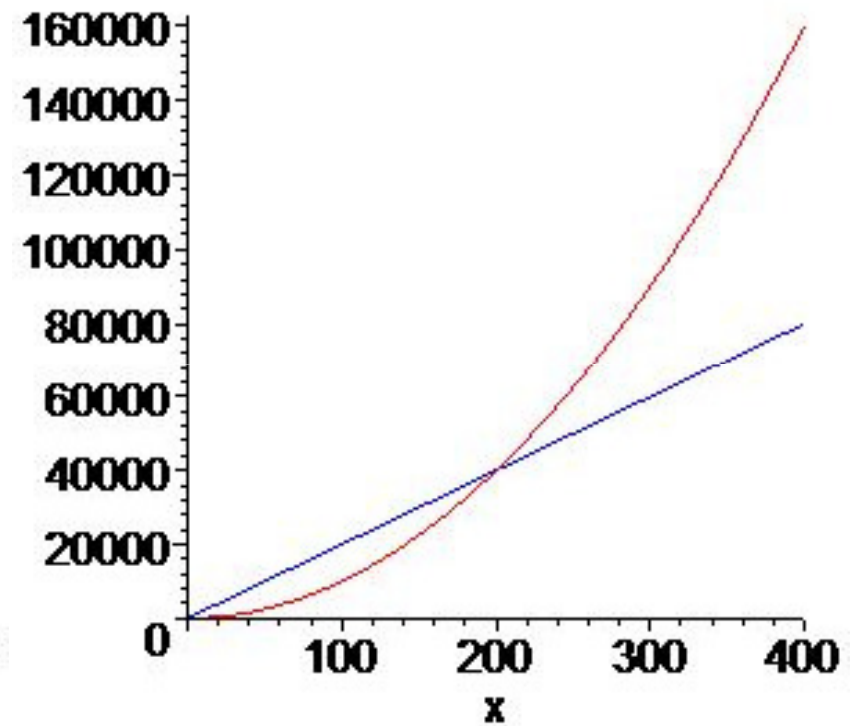
$$O(g(n)) := \left\{ \begin{array}{l} \text{Es existieren Konstanten } c > 0, n_0, \\ f(n) : \text{ so dass für alle } n \geq n_0 \text{ gilt} \\ 0 \leq f(n) \leq cg(n). \end{array} \right\}$$

- $O(g(n))$ formalisiert: Die Funktion $f(n)$ wächst asymptotisch nicht schneller als $g(n)$.
- Statt $f(n)$ in $O(g(n))$ in der Regel $f(n) = O(g(n))$

Illustration von $O(g(n))$



$$g(x) = x^2$$
$$f(x) = 100x$$



$$g(x) = x^2$$
$$f(x) = 200x$$

Ω -Notation

Definition 2.7: Sei $g : \mathbb{N} \rightarrow \mathbb{R}^+$ eine Funktion. Dann bezeichnen wir mit $\Omega(g(n))$ die folgende Menge von Funktionen

$$\Omega(g(n)) := \left\{ f(n) : \begin{array}{l} \text{Es existieren Konstanten } c > 0, n_0, \\ \text{so dass f\u00fcr alle } n \geq n_0 \text{ gilt} \\ 0 \leq cg(n) \leq f(n). \end{array} \right\}$$

- $\Omega(g(n))$ formalisiert: Die Funktion $f(n)$ w\u00e4chst asymptotisch mindestens so schnell wie $g(n)$.
- Statt $f(n)$ in $\Omega(g(n))$ in der Regel $f(n) = \Omega(g(n))$

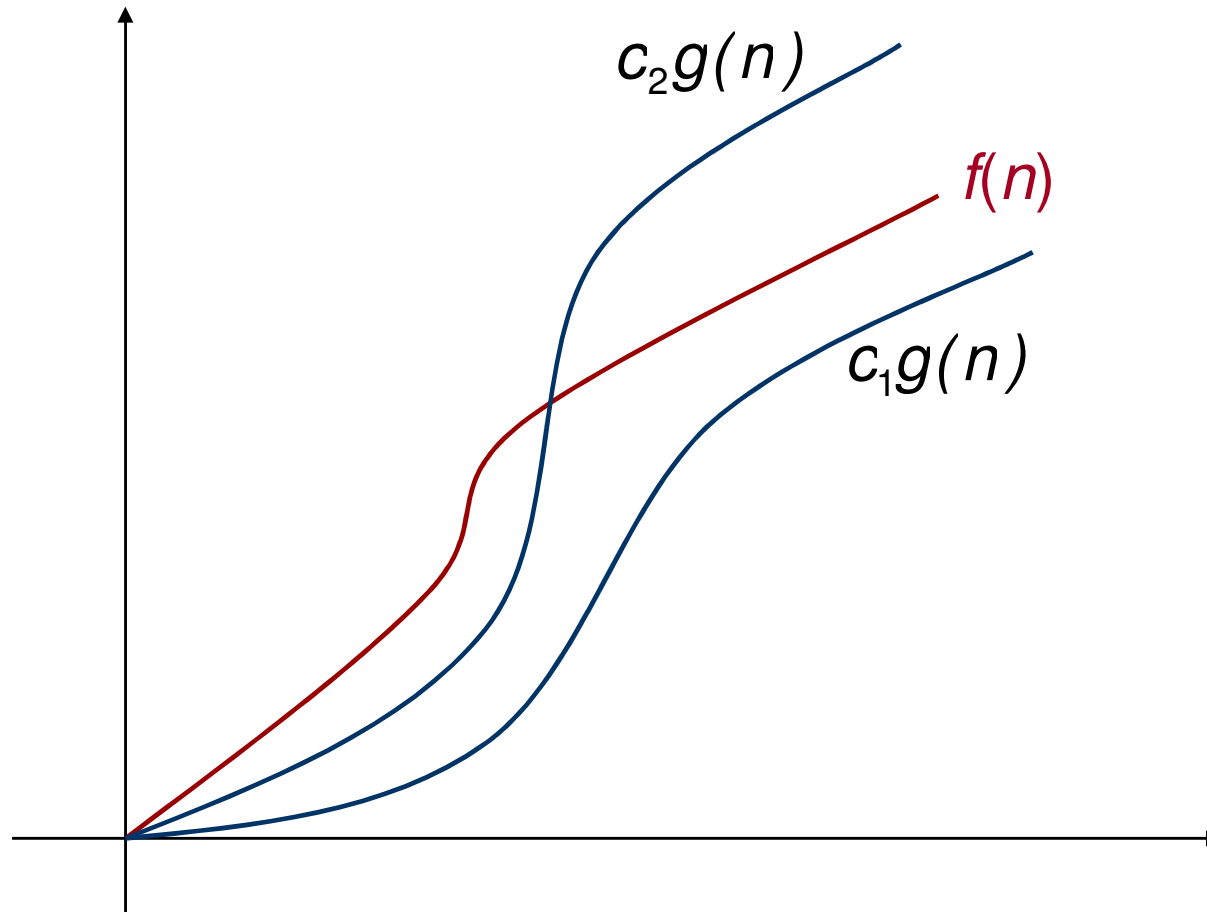
Θ -Notation

Definition 2.8: Sei $g : \mathbb{N} \rightarrow \mathbb{R}^+$ eine Funktion. Dann bezeichnen wir mit $\Theta(g(n))$ die folgende Menge von Funktionen

$$\Theta(g(n)) := \left\{ \begin{array}{l} \text{Es existieren Konstanten } c_1 > 0, c_2, n_0, \\ f(n) : \text{ so dass f\u00fcr alle } n \geq n_0 \text{ gilt} \\ 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n). \end{array} \right\}$$

- $\Theta(g(n))$ formalisiert: Die Funktion $f(n)$ w\u00e4chst asymptotisch genau so schnell $g(n)$.
- Statt $f(n)$ in $\Theta(g(n))$ in der Regel $f(n) = \Theta(g(n))$

Illustration von $\Theta(g(n))$



Regeln für Kalküle - Transitivität

O-, Ω - und Θ -Kalkül sind **transitiv**, d.h.:

➤ Aus $f(n) = O(g(n))$ und $g(n) = O(h(n))$ folgt $f(n) = O(h(n))$.

➤ Aus $f(n) = \Omega(g(n))$ und $g(n) = \Omega(h(n))$ folgt $f(n) = \Omega(h(n))$.

➤ Aus $f(n) = \Theta(g(n))$ und $g(n) = \Theta(h(n))$ folgt $f(n) = \Theta(h(n))$.

Regeln für Kalküle - Transitivität

$f(n) = O(g(n)) \leftrightarrow$ es gibt c', n'_0 , so dass

$$f(n) < c'g(n) \text{ für alle } n > n'_0.$$

$g(n) = O(h(n)) \leftrightarrow$ es gibt c'', n''_0 , so dass

$$g(n) < c''h(n) \text{ für alle } n > n''_0.$$

Sei $n_0 = \max\{n'_0, n''_0\}$ und $c = c' \cdot c''$. Dann gilt für $n > n_0$:

$$f(n) < c' \cdot g(n) < c' \cdot c'' \cdot h(n) = c \cdot h(n).$$

Regeln für Kalküle - Transitivität

$f(n) = \Omega(g(n)) \leftrightarrow$ es gibt c', n'_0 , so dass

$$f(n) > c'g(n) \text{ für alle } n > n'_0.$$

$g(n) = \Omega(h(n)) \leftrightarrow$ es gibt c'', n''_0 , so dass

$$g(n) > c''h(n) \text{ für alle } n > n''_0.$$

Sei $n_0 = \max\{n'_0, n''_0\}$ und $c = c' \cdot c''$. Dann gilt für $n > n_0$:

$$f(n) > c' \cdot g(n) > c' \cdot c'' \cdot h(n) = c \cdot h(n).$$

Regeln für Kalküle - Transitivität

$$f(n) = \Theta(g(n)) \rightarrow f(n) = O(g(n)) \text{ und } f(n) = \Omega(g(n))$$

$$g(n) = \Theta(h(n)) \rightarrow g(n) = O(h(n)) \text{ und } g(n) = \Omega(h(n))$$

$$f(n) = O(g(n)) \text{ und } g(n) = O(h(n)) \text{ impliziert } f(n) = O(h(n))$$

$$f(n) = \Omega(g(n)) \text{ und } g(n) = \Omega(h(n)) \text{ impliziert } f(n) = \Omega(h(n))$$

↓

$$f(n) = \Theta(h(n)).$$

Regeln für Kalküle - Reflexivität

- O-, Ω- und Θ-Kalkül sind **reflexiv**, d.h.:

$$f(n) = O(f(n))$$

$$f(n) = \Omega(f(n))$$

$$f(n) = \Theta(f(n))$$

- Θ-Kalkül ist **symmetrisch**, d.h.

$$f(n) = \Theta(g(n)) \text{ genau dann, wenn } g(n) = \Theta(f(n)).$$

Regeln für Kalküle

Satz 2.10: Sei $f : \mathbb{N} \rightarrow \mathbb{R}^+$ mit $f(n) \geq 1$ für alle n . Weiter sei $k, l \geq 0$ mit $k \geq l$. Dann gilt

1. $f(n)^l = O(f(n)^k)$.
2. $f(n)^k = \Omega(f(n)^l)$.

Satz 2.11: Seien $\varepsilon, k > 0$ beliebig. Dann gilt

1. $\log(n)^k = O(n^\varepsilon)$.
2. $n^\varepsilon = \Omega(\log(n)^k)$.

Beweis – Satz 2.11

- ▶ $\lim_{n \rightarrow \infty} \log n = \infty$
- ▶ Regel von L'Hospital: $I =]a, b[$ nichtleeres, offenes Intervall, $f, g : I \rightarrow \mathbf{R}$ mit $\lim_{x \rightarrow b} f(x) = \lim_{x \rightarrow b} g(x) \in \{0, \infty\}$.
Falls $g(x) \neq 0$ für alle $x \in I$ und

$$\lim_{x \rightarrow b} \frac{f'(x)}{g'(x)} \in \mathbf{R},$$

dann gilt

$$\lim_{x \rightarrow b} \frac{f(x)}{g(x)} = \lim_{x \rightarrow b} \frac{f'(x)}{g'(x)}.$$

Beweis – Satz 2.11 (2)

- ▶ Wir zeigen zunächst: $\ln(n) = O(n)$
- ▶ Regel von L'Hospital mit $f(x) = x$ und $g(x) = \ln(x)$ impliziert $\lim_{x \rightarrow \infty} \frac{x}{\ln(x)} = \lim_{x \rightarrow \infty} \frac{1}{1/x} = \infty$.
- ▶ \Rightarrow Für alle $c > 0$ gibt es $x_c > 0$, so dass für alle $x > x_c$

$$\frac{x}{\ln(x)} > c.$$

Beweis – Satz 2.11 (3)

- ▶ Analog gilt: Ersetze x durch $\ln(n)$; Für alle $c > 0$ gibt es $x_c > 0$, so dass für alle $\ln(n) > x_c$

$$\ln(n) > c \ln \ln(n).$$

- ▶ Dann gilt:

$$\ln(n)^k \leq e^{\frac{k}{\epsilon} \cdot \frac{\epsilon}{k} k \ln \ln(n)} \leq e^{\frac{\epsilon}{k} k \ln(n)} \leq n^\epsilon$$

für alle $\ln(n) > x_{k/\epsilon}$.

- ▶ Wir haben ausgenutzt: $\ln(n) > \frac{k}{\epsilon} \ln \ln(n)$ für alle $\ln(n) > x_{k/\epsilon}$.

3. Inkrementelle Algorithmen

Definition 3.1: Bei einem inkrementellen Algorithmus wird sukzessive die Teillösung für die ersten i Objekte aus der bereits bekannten Teillösung für die ersten $i-1$ Objekte berechnet, $i=1, \dots, n$.

Beispiel Min-Search:

- Objekte sind Einträge des Eingabearrays.
- Teilproblem bestehend aus ersten i Objekten bedeutet Minimum im Teilarray $A[1..i]$ bestimmen.

Sortieren und inkrementelle Algorithmen

Eingabe bei Sortieren: Folge von n Zahlen (a_1, a_2, \dots, a_n) .

Ausgabe bei Sortieren: Umordnung (b_1, b_2, \dots, b_n) der Eingabefolge, so dass $b_1 \leq b_2 \leq \dots \leq b_n$.

Sortieralgorithmus: Verfahren, das zu jeder Folge (a_1, a_2, \dots, a_n) sortierte Umordnung (b_1, b_2, \dots, b_n) berechnet.

Eingabe: $(31, 41, 59, 26, 51, 48)$

Ausgabe: $(26, 31, 41, 48, 51, 59)$

Insertion-Sort

Idee: Sukzessive wird eine Sortierung der Teilarrays $A[1..i]$, $1 \leq i \leq \text{length}(A)$ berechnet.

Insertion - Sort(A)

```
1 for  $j \leftarrow 2$  to  $\text{length}(A)$ 
2   do  $key \leftarrow A[j]$ 
3     ▷ Füge  $A[j]$  in die sortierte Folge  $A[1..j-1]$  ein.
4      $i \leftarrow j-1$ 
5     while  $i > 0$  and  $A[i] > key$ 
6       do  $A[i+1] \leftarrow A[i]$ 
7          $i \leftarrow i-1$ 
8    $A[i+1] \leftarrow key$ 
```

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

Beispiel:

| | | | | | | | |
|---|----|---|----|---|---|----|----|
| 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |
|---|----|---|----|---|---|----|----|

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | |
|---|----|---|----|---|---|----|----|
| 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |
|---|----|---|----|---|---|----|----|

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | |
|---|----|---|----|---|---|----|----|
| | 1 | | | | | | n |
| 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | | |
|--|---|----|---|----|---|---|----|----|
| | 1 | j | | | | | n | |
| | 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |

Insertion Sort

InsertionSort(Array A)

➤ Eingabegröße n

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**

➤ $(\text{length}(A)=n)$

2. $\text{key} \leftarrow A[j]$

3. $i \leftarrow j-1$

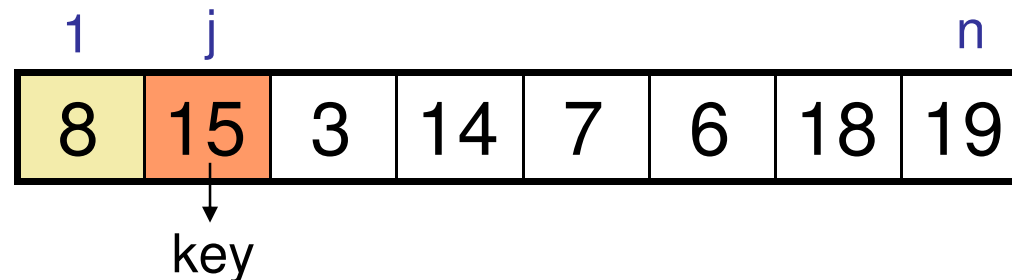
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**

5. $A[i+1] \leftarrow A[i]$

6. $i \leftarrow i-1$

7. $A[i+1] \leftarrow \text{key}$

Beispiel:



Insertion Sort

InsertionSort(Array A)

➤ Eingabegröße n

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**

➤ ($\text{length}(A)=n$)

2. $\text{key} \leftarrow A[j]$

3. $i \leftarrow j-1$

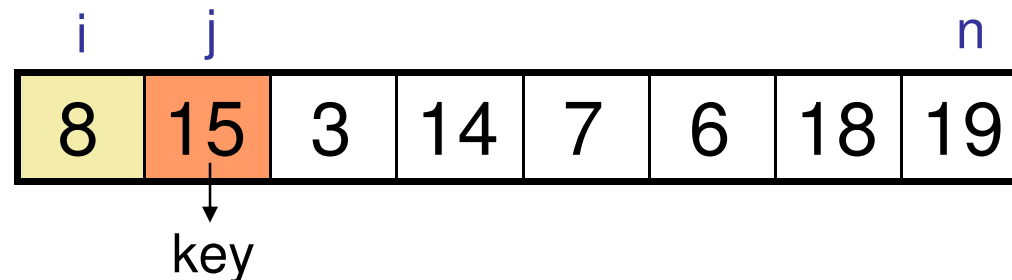
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**

5. $A[i+1] \leftarrow A[i]$

6. $i \leftarrow i-1$

7. $A[i+1] \leftarrow \text{key}$

Beispiel:



Insertion Sort

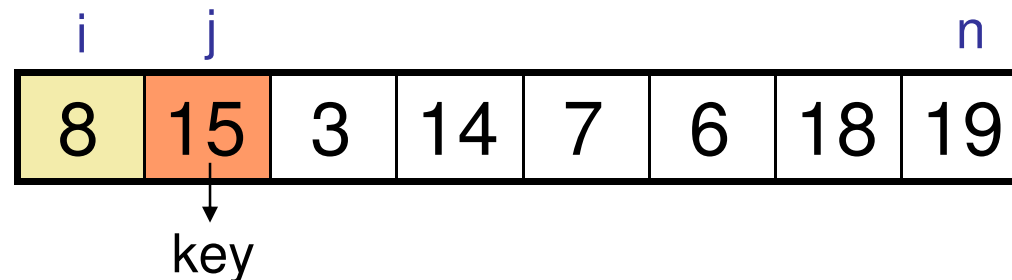
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



Insertion Sort

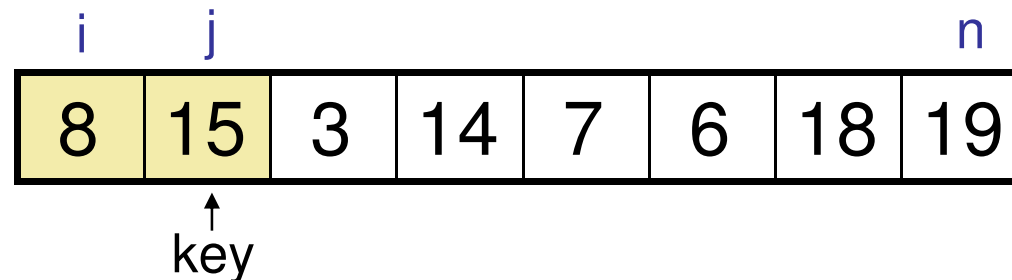
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | | |
|--|---|----|---|----|---|---|----|----|
| | 1 | | j | | | | n | |
| | 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |

Insertion Sort

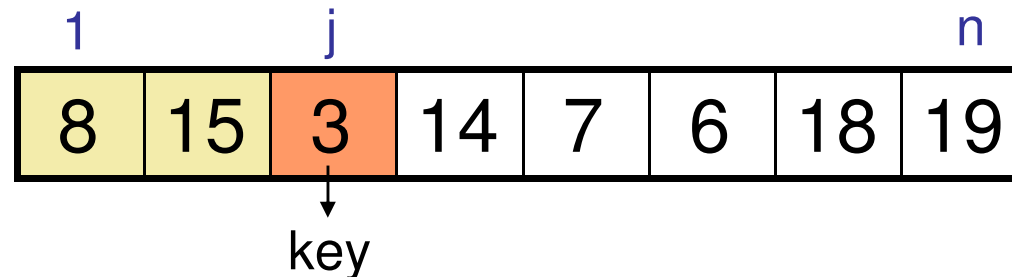
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



Insertion Sort

InsertionSort(Array A)

➤ Eingabegröße n

1. **for** $j \leftarrow 2$ **to** length(A) **do**

➤ (length(A)=n)

2. key \leftarrow A[j]

3. $i \leftarrow j-1$

4. **while** $i > 0$ and $A[i] > \text{key}$ **do**

5. $A[i+1] \leftarrow A[i]$

6. $i \leftarrow i-1$

7. $A[i+1] \leftarrow \text{key}$

Beispiel:

| | | | | | | | | |
|--|---|----|---|----|---|---|----|----|
| | 1 | i | j | | | | n | |
| | 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |

key=3

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | | |
|--|---|----|---|----|---|---|----|----|
| | 1 | i | j | | | | n | |
| | 8 | 15 | 3 | 14 | 7 | 6 | 18 | 19 |

key=3

Insertion Sort

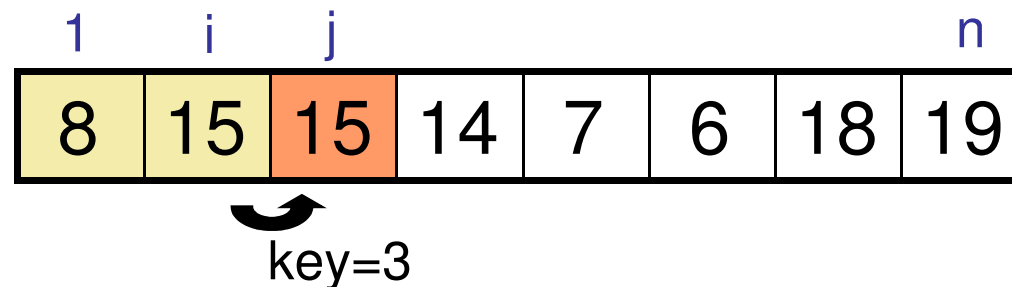
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | |
|---|-----|----|-----|---|---|----|-----|
| | i | | j | | | | n |
| 8 | 15 | 15 | 14 | 7 | 6 | 18 | 19 |

key=3

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:

| | | | | | | | |
|---|-----|----|-----|---|---|----|-----|
| | i | | j | | | | n |
| 8 | 15 | 15 | 14 | 7 | 6 | 18 | 19 |

key=3

Insertion Sort

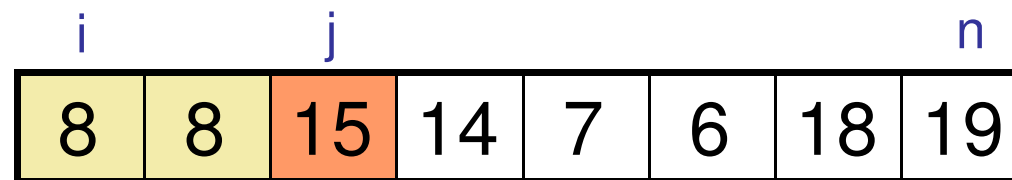
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



key=3

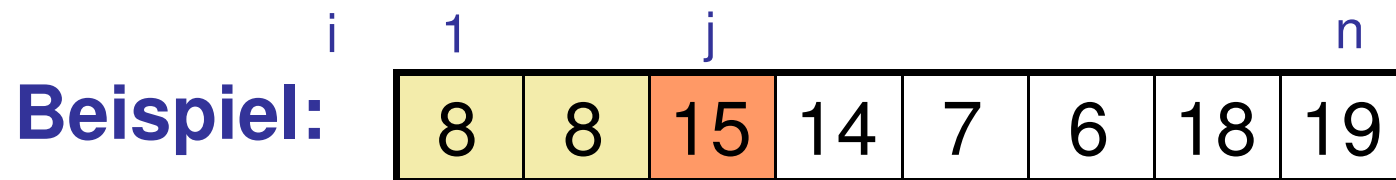
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$



key=3

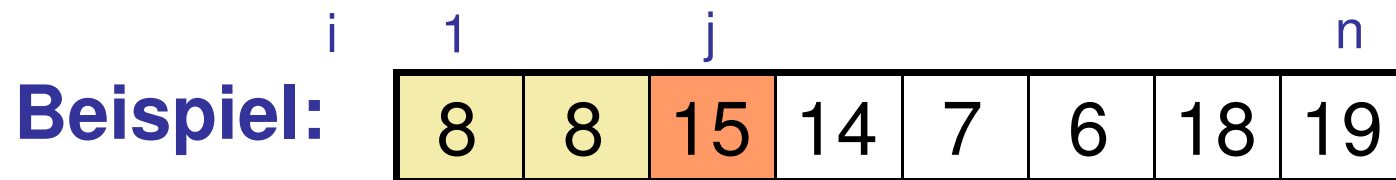
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$



key=3

Insertion Sort

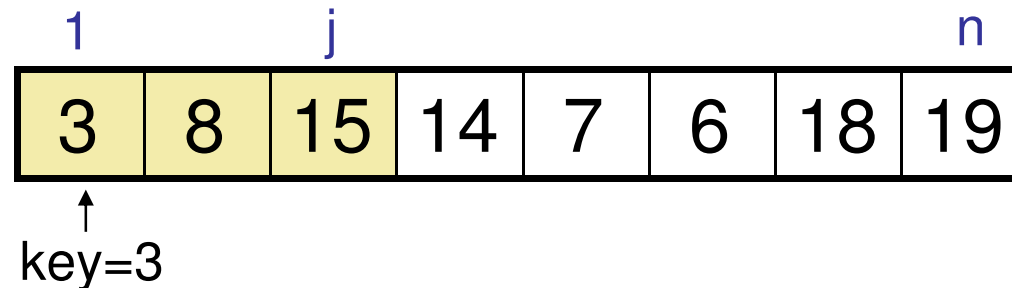
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



Insertion Sort

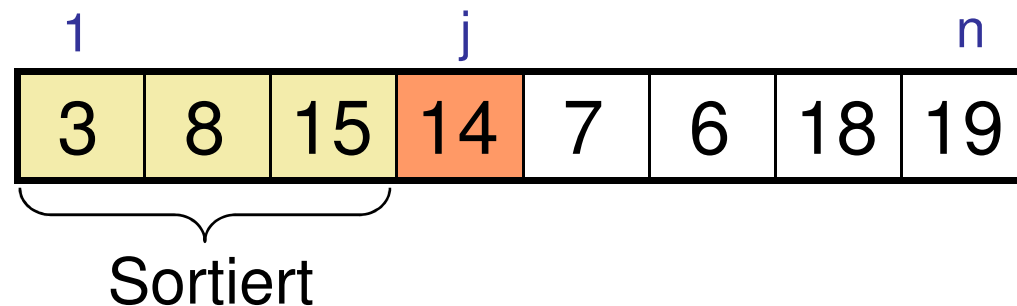
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



Insertion Sort

InsertionSort(Array A)

➤ Eingabegröße n

1. **for** j ← 2 **to** length(A) **do**

➤ (length(A)=n)

2. key ← A[j]

3. i ← j-1

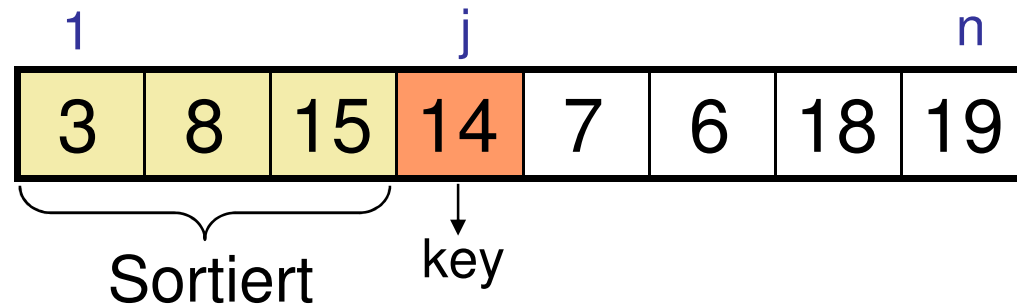
4. **while** i>0 and A[i]>key **do**

5. A[i+1] ← A[i]

6. i ← i-1

7. A[i+1] ← key

Beispiel:



Insertion Sort

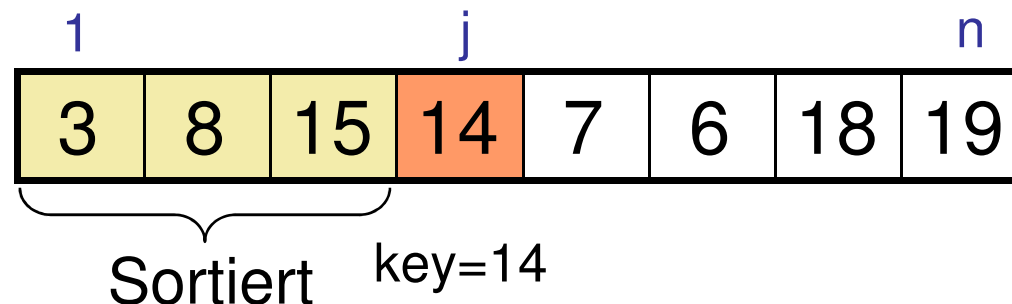
InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

➤ Eingabegröße n

➤ $(\text{length}(A)=n)$

Beispiel:



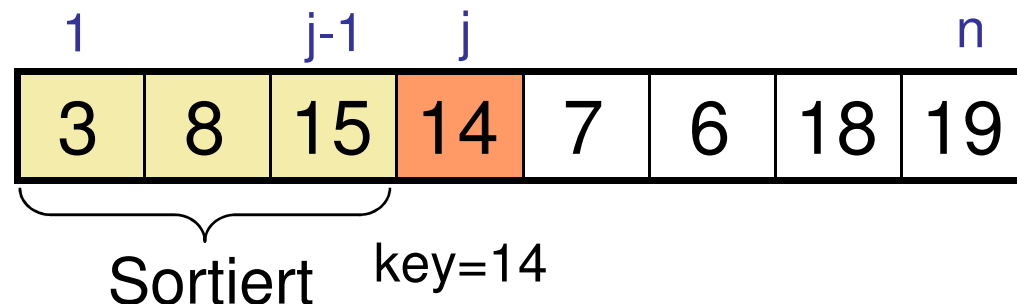
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts

Beispiel:



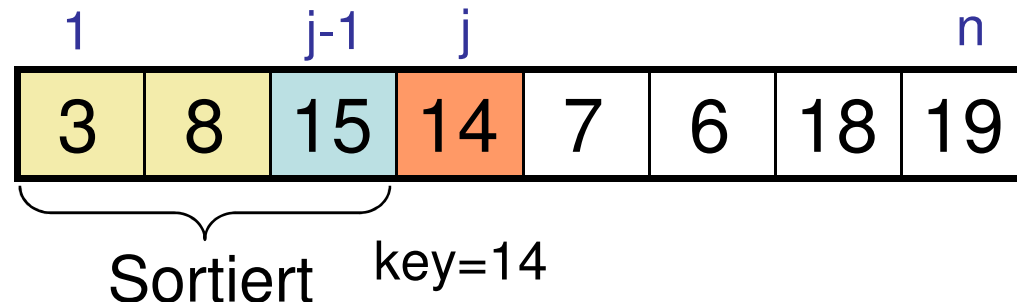
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts

Beispiel:



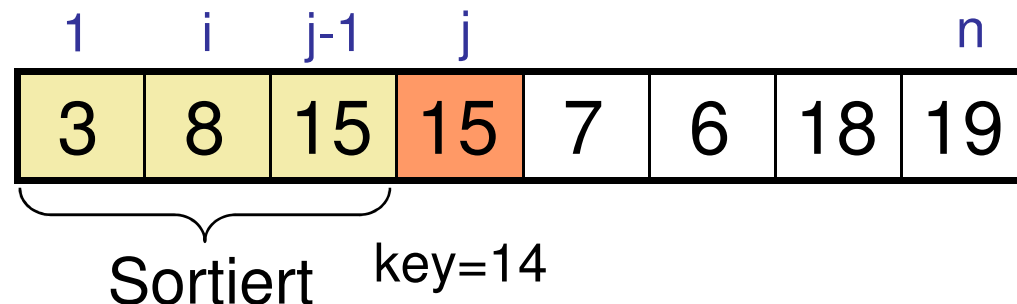
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts

Beispiel:



Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts

Beispiel:

| | | | | | | | | |
|--|---|---|-----|----|---|---|----|----|
| | 1 | i | i+1 | j | | | n | |
| | 3 | 8 | 15 | 15 | 7 | 6 | 18 | 19 |

key=14

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:

| | | | | | | | | |
|--|---|---|-----|----|---|---|----|----|
| | 1 | i | i+1 | j | | | n | |
| | 3 | 8 | 15 | 15 | 7 | 6 | 18 | 19 |

key=14

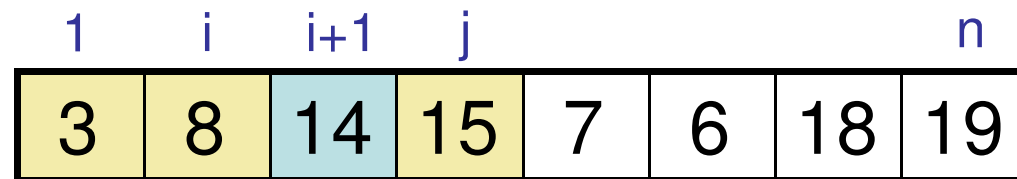
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



↑
key=14

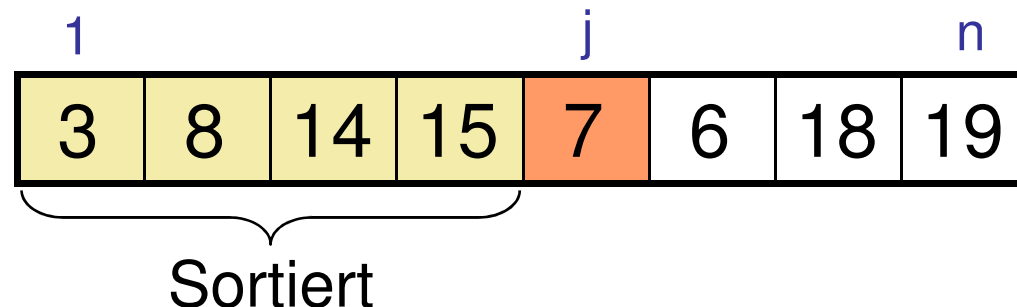
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



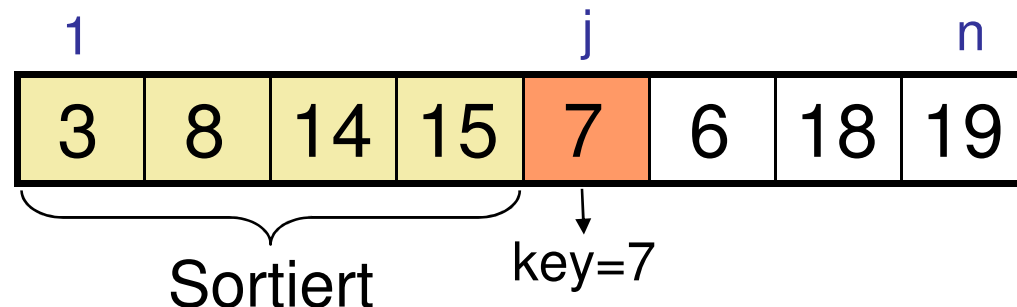
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



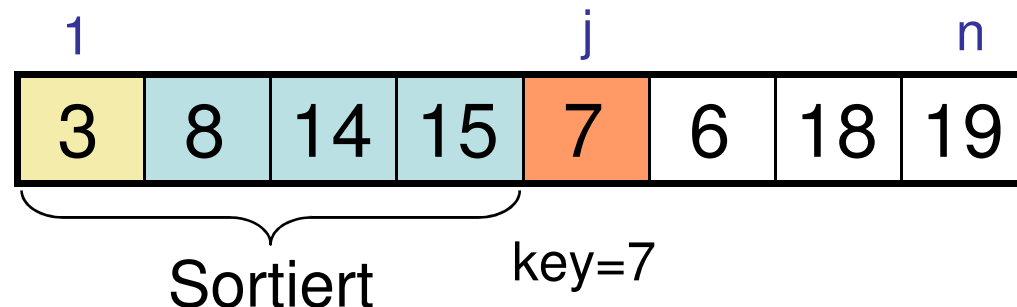
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $(\text{length}(A)=n)$
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:

| | | | | | | | |
|---|---|---|----|----|---|----|----|
| | 1 | | | j | | | n |
| 3 | 8 | 8 | 14 | 15 | 6 | 18 | 19 |

$\text{key}=7$

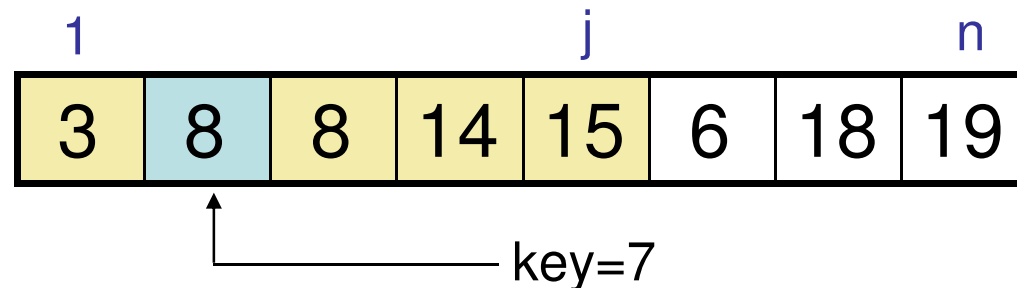
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



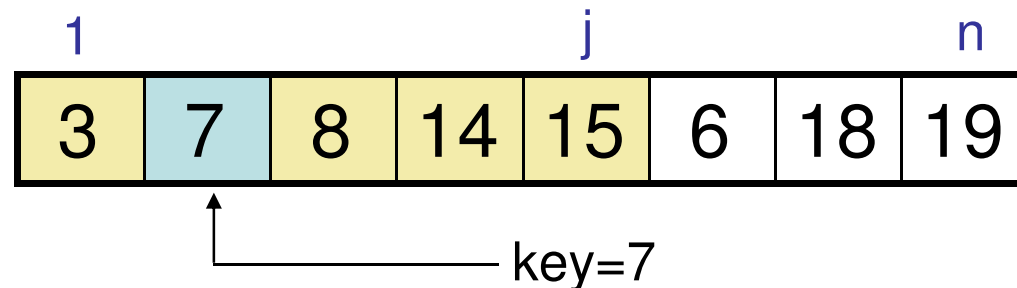
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



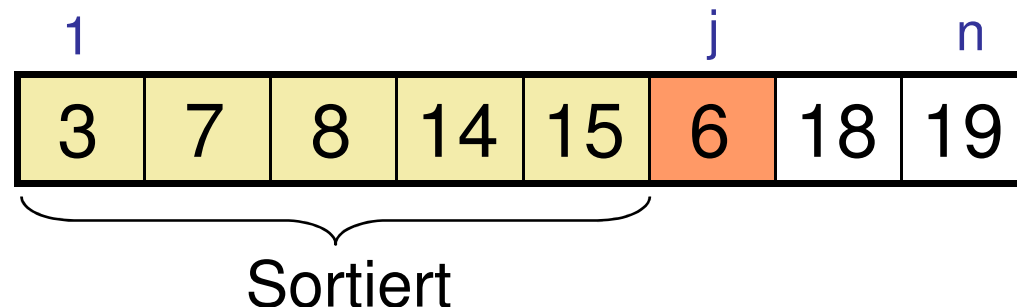
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



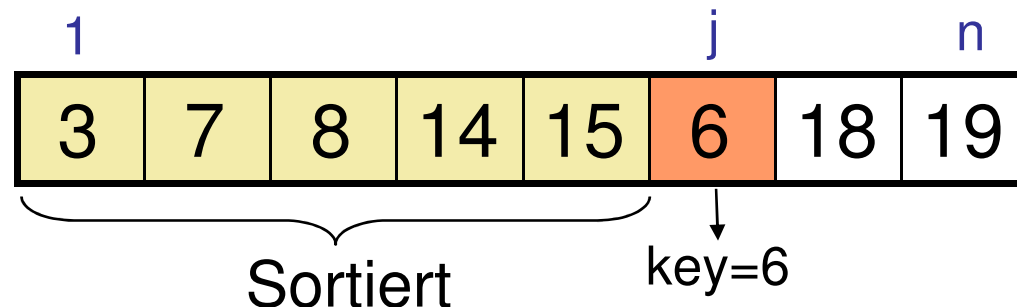
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:

| | | | | | | | | |
|--|---|---|---|----|----|---|----|----|
| | 1 | | | | | j | | n |
| | 3 | 7 | 8 | 14 | 15 | 6 | 18 | 19 |

key=6

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $(\text{length}(A)=n)$
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:

| | | | | | | | | |
|--|---|---|---|---|----|----|----|----|
| | 1 | | | | | j | | n |
| | 3 | 7 | 7 | 8 | 14 | 15 | 18 | 19 |

key=6

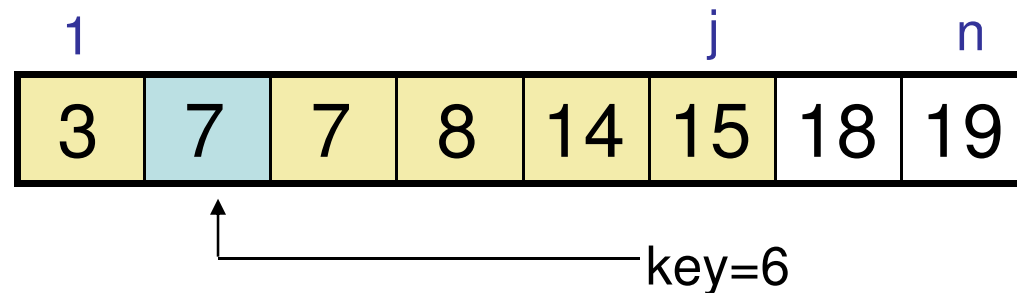
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



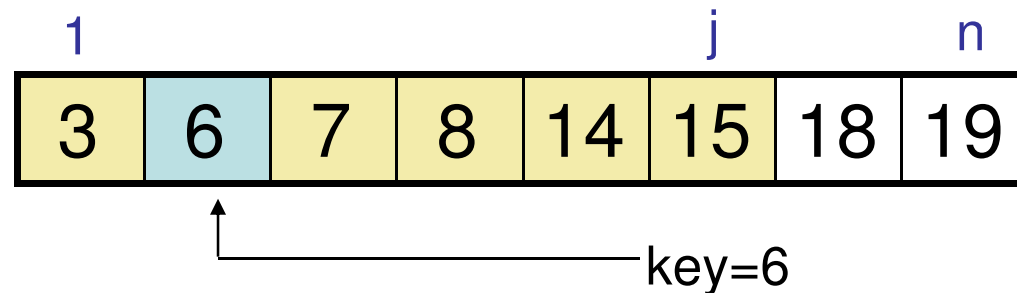
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



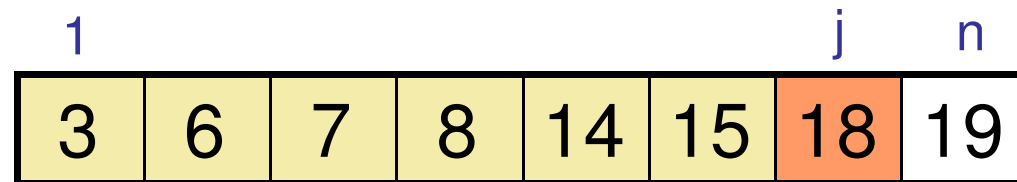
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $(\text{length}(A)=n)$
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



Sortiert

Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $(\text{length}(A)=n)$
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:

| | | | | | | | |
|---|---|---|---|----|----|----|----|
| | 1 | | | | | j | n |
| 3 | 6 | 7 | 8 | 14 | 15 | 18 | 19 |

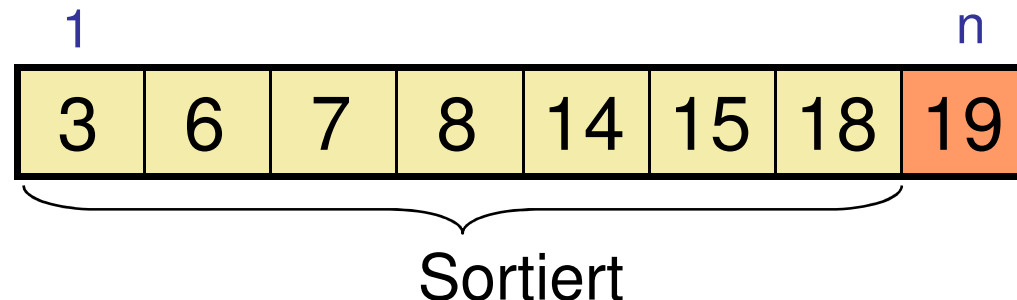
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- ($\text{length}(A)=n$)
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



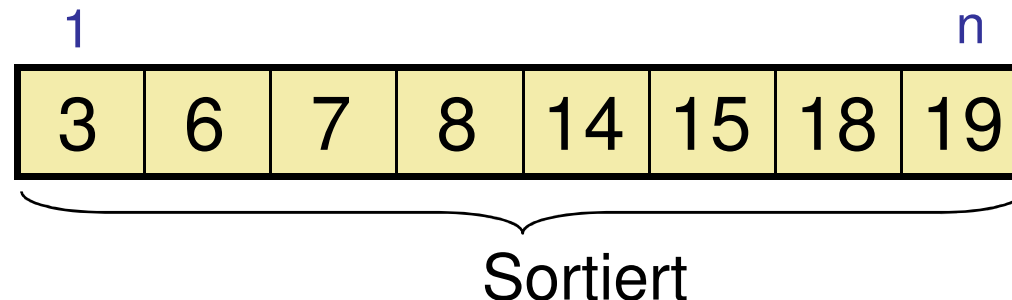
Insertion Sort

InsertionSort(Array A)

1. **for** $j \leftarrow 2$ **to** $\text{length}(A)$ **do**
2. $\text{key} \leftarrow A[j]$
3. $i \leftarrow j-1$
4. **while** $i > 0$ and $A[i] > \text{key}$ **do**
5. $A[i+1] \leftarrow A[i]$
6. $i \leftarrow i-1$
7. $A[i+1] \leftarrow \text{key}$

- Eingabegröße n
- $(\text{length}(A)=n)$
- verschiebe alle
- $A[1, \dots, j-1]$, die größer als
- key sind eine Stelle
- nach rechts
- Speichere key in „Lücke“

Beispiel:



Invariante bei Insertion-Sort

Invariante: Vor Durchlauf der for-Schleife (Zeilen 1-8) für Index j gilt, dass $A[1..j-1]$ die $j-1$ Eingabezahlen in sortierter Reihenfolge enthält.

Initialisierung: $j=2$ und $A[1]$ ist sortiert, da es nur eine Zahl enthält.

Erhaltung: while-Schleife (Zeilen 5-7) zusammen mit Zeile 8 sortiert $A[j]$ korrekt ein.

Terminierung: Vor Durchlauf mit $j=length(A)+1$ ist $A[1..length(A)]$ sortiert.

Invariante bei Insertion-Sort

Satz 1

- Insertion-Sort sortiert eine Folge von n Zahlen aufsteigend.

Beweis:

- Wir zeigen, dass die Schleifeninvariante erfüllt ist
- Da die Schleife mit $j=n+1$ terminiert, folgt aus der Invariante die Korrektheit des Algorithmus

Induktionsanfang ($j=2$):

- $A[1]$ ist sortiert ✓

Induktionsannahme(I.A.):

Invariante gilt für $j=N$

Invariante bei Insertion-Sort

Induktionsschritt ($N \rightarrow N+1$):

- Betrachte Durchlauf mit $j=N$
- Insertionsort merkt sich $A[N]$ in Variable key
- Sei $1 \leq k \leq N-1$ der größte Index mit $A[k] > \text{key}$ oder $k=N$, falls ein solcher nicht existiert
- Der Algorithmus verschiebt $A[k, \dots, N-1]$ nach $A[k+1, \dots, N]$
- Dann wird $A[k]$ auf den Wert key gesetzt
- Danach gilt:
 - (1) $A[1] \leq A[2] \leq \dots \leq A[k-1]$ nach I.A.
 - (2) $A[k-1] \leq A[k] \leq A[k+1]$ nach Ablauf der Schleife
 - (3) $A[k+1] \leq A[k+2] \leq \dots \leq A[N]$ nach I.A.
- Aus (1)-(3) folgt die Behauptung für $N+1$

Insertion-Sort – Analyse (1)

| Insertion - Sort(<i>A</i>) | cost | times |
|--|-------|--------------------------|
| 1 for $j \leftarrow 2$ to length(<i>A</i>) | C_1 | n |
| 2 do key $\leftarrow A[j]$ | C_2 | $n-1$ |
| 3 ▷ Einfügen von $A[j]$. | | |
| 4 $i \leftarrow j-1$ | C_4 | $n-1$ |
| 5 while $i > 0$ and $A[i] > key$ | C_5 | $\sum_{j=2}^n t_j$ |
| 6 do $A[i+1] \leftarrow A[i]$ | C_6 | $\sum_{j=2}^n (t_j - 1)$ |
| 7 $i \leftarrow i-1$ | C_7 | $\sum_{j=2}^n (t_j - 1)$ |
| 8 $A[i+1] \leftarrow key$ | C_8 | $n-1$ |

Hierbei ist t_j die Anzahl der Durchläufe der Abfrage in Zeile 5.

Insertion-Sort – Analyse (2)

Satz 3.2. Insertion-Sort besitzt Laufzeit $\Theta(n^2)$.

Zum Beweis wird gezeigt:

1. Es gibt ein c_2 , so dass die Laufzeit von Insertion - Sort bei allen Eingaben der Größe n immer höchstens c_2n^2 ist.
2. Es gibt ein c_1 , so dass für alle n eine Eingabe I_n der Größe n existiert bei der Insertion - Sort mindestens Laufzeit c_1n^2 besitzt.

4. Divide & Conquer – Merge-Sort

Definition 4.1: Divide&Conquer (Teile&Erobere) ist eine auf Rekursion beruhende Algorithmentechnik.

Eine Divide&Conquer-Algorithmus löst ein Problem in 3 Schritten:

- **Teile** ein Problem in mehrere Unterprobleme.
- **Erobere** jedes einzelne Unterproblem durch rekursive Lösung. Ausnahme sind kleine Unterprobleme, diese werden direkt gelöst.
- **Kombiniere** die Lösungen der Teilprobleme zu einer Gesamtlösung.

Divide&Conquer und Sortieren

- **Teile** ein Problem in Unterprobleme.
- **Erobere** jedes einzelne Teilproblem, durch rekursive Lösung.
- **Kombiniere** die Lösungen zu einer Gesamtlösung.

- Teile eine n -elementige Teilfolge auf in zwei Teilfolgen mit jeweils etwa $n/2$ Elementen.
- Sortiere die beiden Teilfolgen rekursiv.
- Mische die sortierten Teilfolgen zu einer sortierten Gesamtfolge.

Merge-Sort

Merge-Sort ist eine mögliche Umsetzung des Divide&Conquer-Prinzips auf das Sortierproblem.

Merge - Sort(A, p, r)

```
1 if  $p < r$ 
2   then  $q \leftarrow \lfloor (p+r)/2 \rfloor$ 
3       Merge - Sort( $A, p, q$ )
4       Merge - Sort( $A, q+1, r$ )
5       Merge( $A, p, q, r$ )
```

- Merge ist Algorithmus zum Mischen zweier sortierter Teilfolgen
- Aufruf zu Beginn mit Merge-Sort($A, 1, length[A]$).

Illustration von Merge-Sort (1)

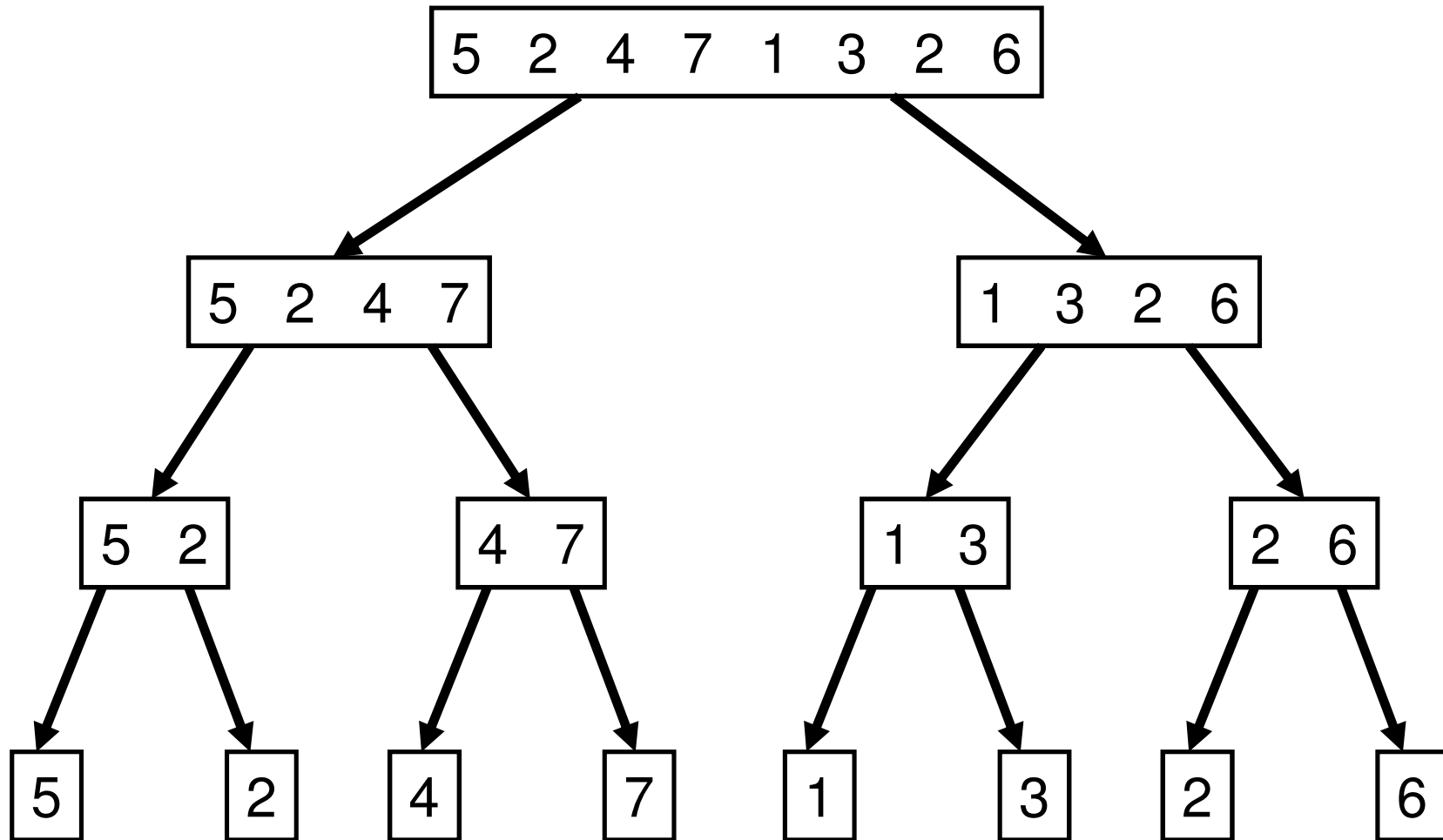
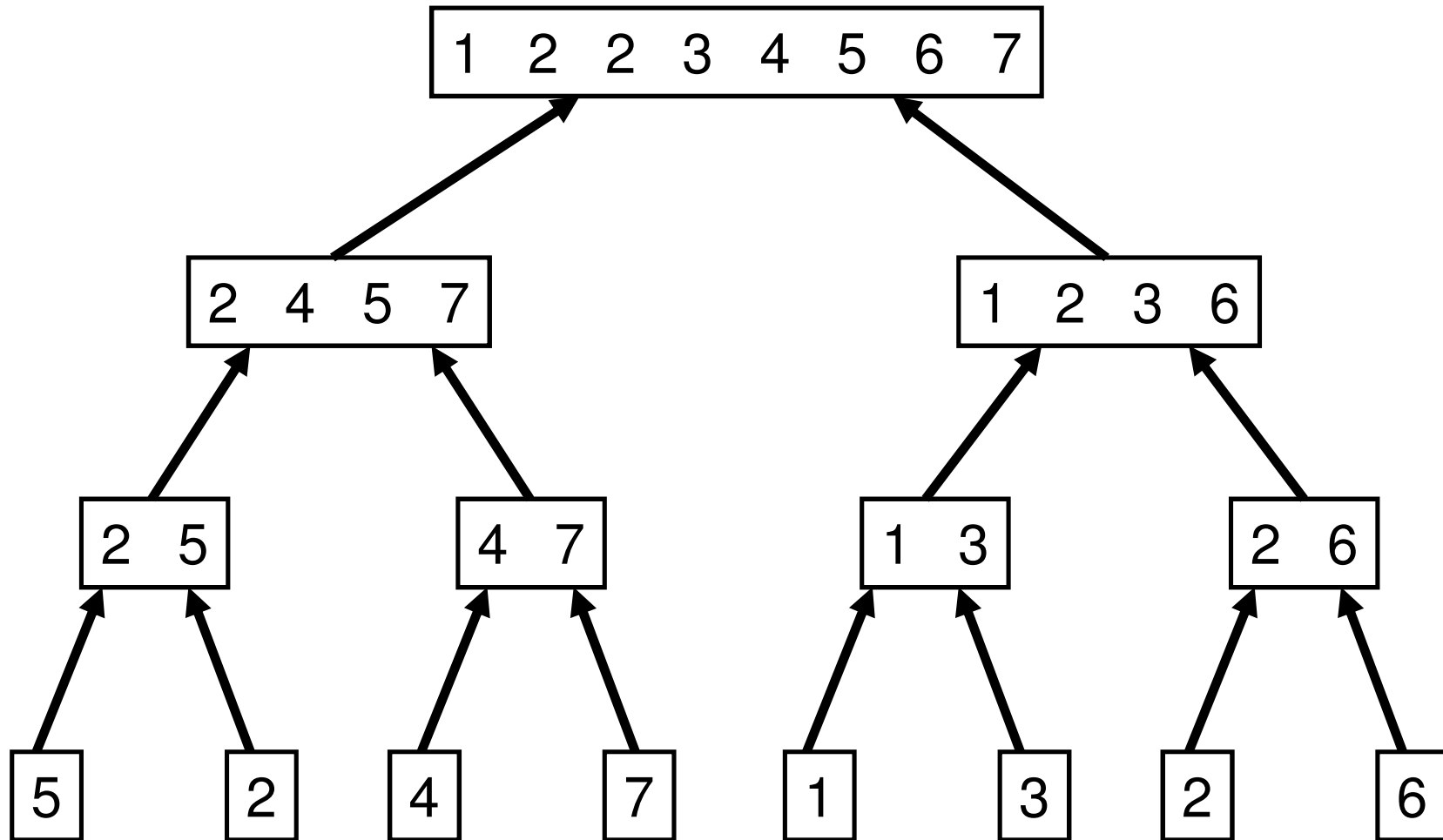


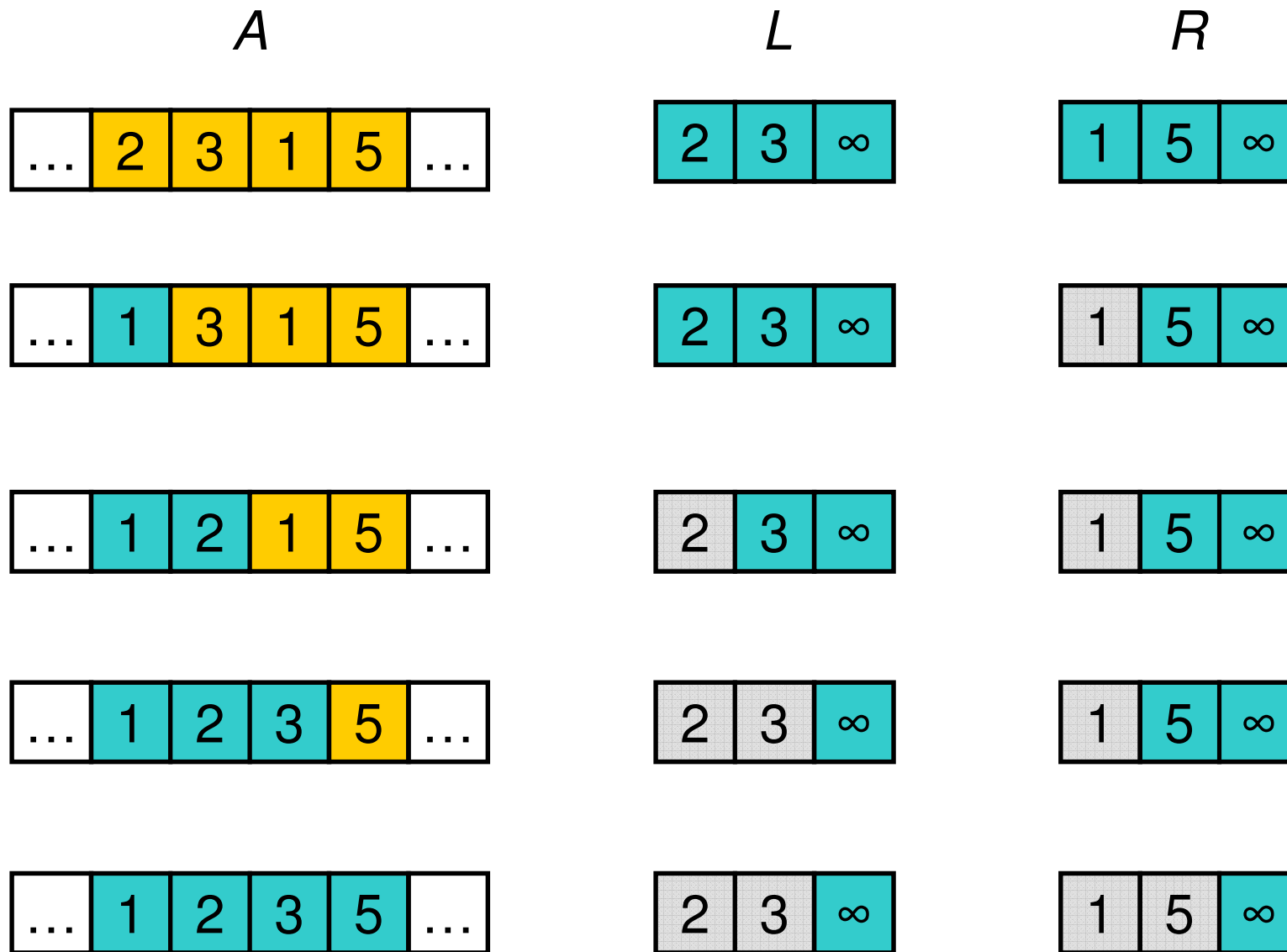
Illustration von Merge-Sort (2)



Kombination von Teilfolgen - Merge

```
Merge( $A, p, q, r$ )
1  $n_1 \leftarrow q - p + 1$ 
2  $n_2 \leftarrow r - q$ 
3  $\triangleright$  Erzeuge Arrays  $L[1..n_1 + 1], R[1..n_2 + 1]$ 
4 for  $i \leftarrow 1$  to  $n_1$ 
5     do  $L[i] \leftarrow A[p + i - 1]$ 
6 for  $j \leftarrow 1$  to  $n_2$ 
7     do  $R[j] \leftarrow A[q + j]$ 
8  $L[n_1 + 1] \leftarrow \infty$ 
9  $R[n_2 + 1] \leftarrow \infty$ 
10  $i \leftarrow 1$ 
11  $j \leftarrow 1$ 
12 for  $k \leftarrow p$  to  $r$ 
13     do if  $L[i] \leq R[j]$ 
14         then  $A[k] \leftarrow L[i]$ 
15              $i \leftarrow i + 1$ 
16         else  $A[k] \leftarrow R[j]$ 
17              $j \leftarrow j + 1$ 
```

Illustration von Merge



Korrektheit von Merge - Sort

Zeigen Korrektheit von Merge-Sort mit **vollständiger Induktion** über Größe des Eingabearrays A . Zeigen hierzu:

1. Ist $length[A]=1$, dann ist Merge-Sort korrekt.
(Induktionsverankerung)
2. Ist Merge-Sort für alle Arrays A mit $length[A]< n$ korrekt, dann ist Merge-Sort auch für Arrays A mit $length[A]=n$ korrekt. (Induktionsschluss)

zu 1. Ist $length[A]=1$, so gilt in Merge $p=r$. Array $A[p]$ wird nicht verändert und Merge-Sort ist korrekt.

zu 2. Folgt, wenn Merge aus sortierten Teilarrays $A[p, \dots, q]$, $A[q+1, \dots, r]$ sortiertes Array $A[p, \dots, r]$ berechnet.

Korrektheit von Merge - Invariante

Lemma 4.2: Erhält Algorithmus Merge als Eingabe ein Teilarray $A[p, \dots, r]$, so dass die beiden Teilarrays $A[p, \dots, q]$ und $A[q+1, \dots, r]$ sortiert sind, so ist enthält nach Durchlauf von Merge das Teilarray $A[p, \dots, r]$ ebenfalls sortiert.

Invariante: Vor Durchlauf der Schleife in Zeilen 12 -17 mit Index k enthält das Array $A[p..k-1]$ die $k-p$ kleinsten Zahlen aus den Arrays L und R in sortierter Reihenfolge. Außerdem sind $L[i]$ und $R[j]$ jeweils die kleinsten noch nicht einsortierten Elemente in den Arrays L bzw. R .

Korrektheit von Merge – 3 Schritte

Initialisierung: Vor Durchlauf mit $k=p$ ist das Array $A[p..k-1]$ leer. Daher ist die Invariante erfüllt.

Erhaltung: Es gelte $L[i] \leq R[j]$. Dann ist $L[i]$ kleinstes noch nicht einsortiertes Element. Damit enthält $A[p..k]$ die $k-p+1$ kleinsten Elemente. Zusammen mit Erhöhung der Zähler i, k garantiert dies Erhaltung der Invariante. Analoge Argumentation im Fall $L[i] > R[j]$.

Terminierung: Nach Ende der Schleife enthält $A[p..r]$ die $r-p+1$ kleinsten Elemente sortiert. Also sind dann alle Elemente sortiert.

Laufzeit von Merge

Lemma 4.3: Ist die Eingabe von Merge ein Teilarray der Größe n , so ist die Laufzeit von Merge $\Theta(n)$.