Übung zu
Algorithmen und Datenstrukturen (für ET/IT)

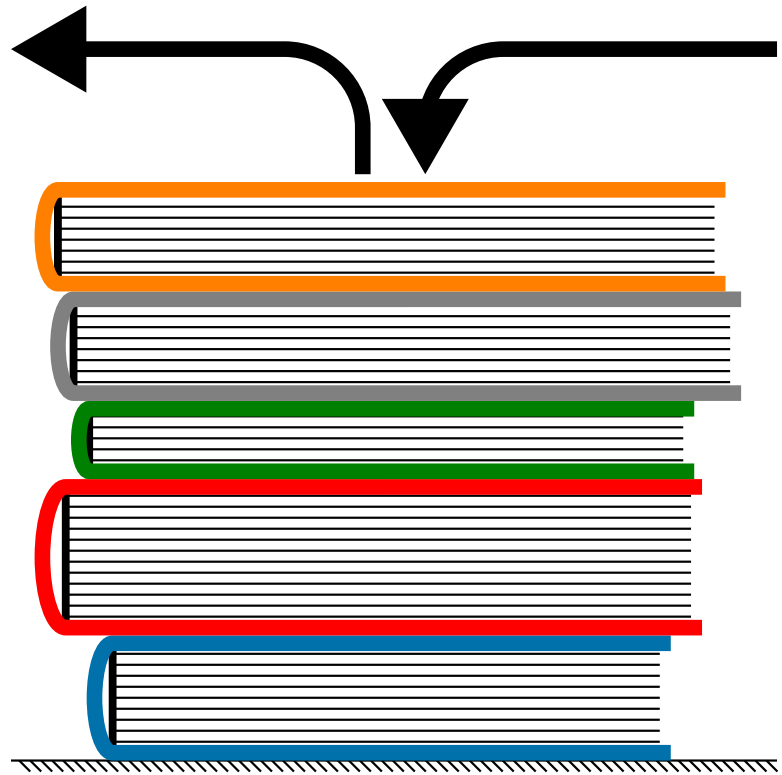
Sommersemester 2014

Jakob Vogel

Computer-Aided Medical Procedures
Technische Universität München

Stack

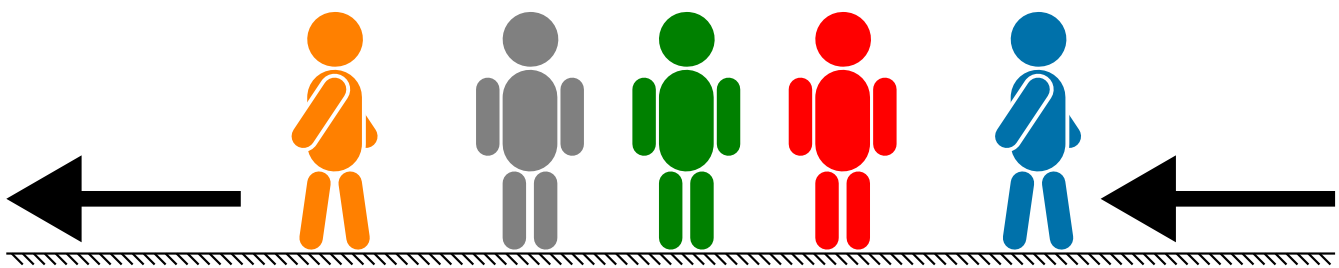
- Abstrakt definiert
- Interface-Funktionen
  - `PUSH`
  - `POP`
  - `TOP`
  - ...
- Constraint
  - LI-FO (Last In, First Out)
  - „Bücherstapel“
- Mögl. Implementation:
  - Array mit Schreibindex
  - Doppelt verkettete Liste
  - ...



Queue

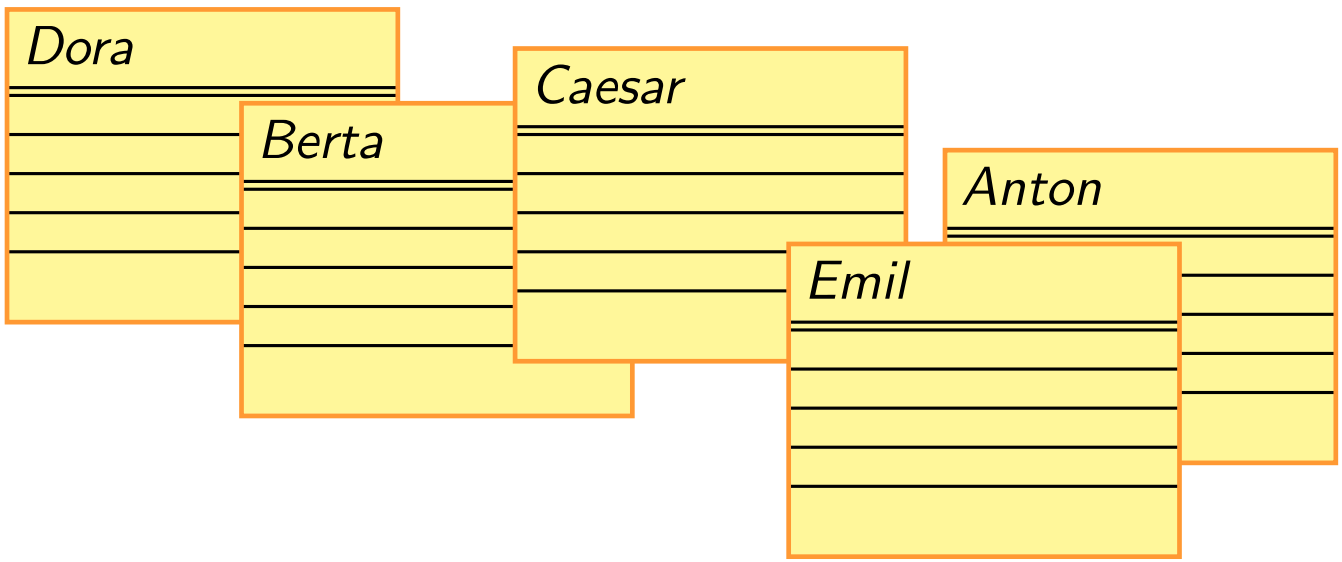
- Abstrakt definiert
- Interface-Funktionen
  - `ENQUEUE`
  - `DEQUEUE`
  - ...
- Constraint
  - FI-FO (First In, First Out)
  - „Warteschlange“

- Mögl. Implementation:
  - Verkettete Liste
  - Zwei Stacks
  - Ringpuffer
  - ...

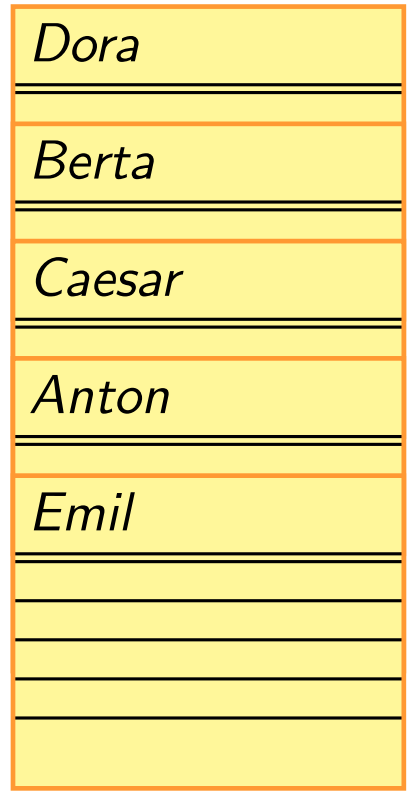


Sortierung: Beispielproblem

- Beispiel: Gustav notiert die Adressen seiner Bekannten auf Karteikarten, und hat den Karteikasten fallen lassen...

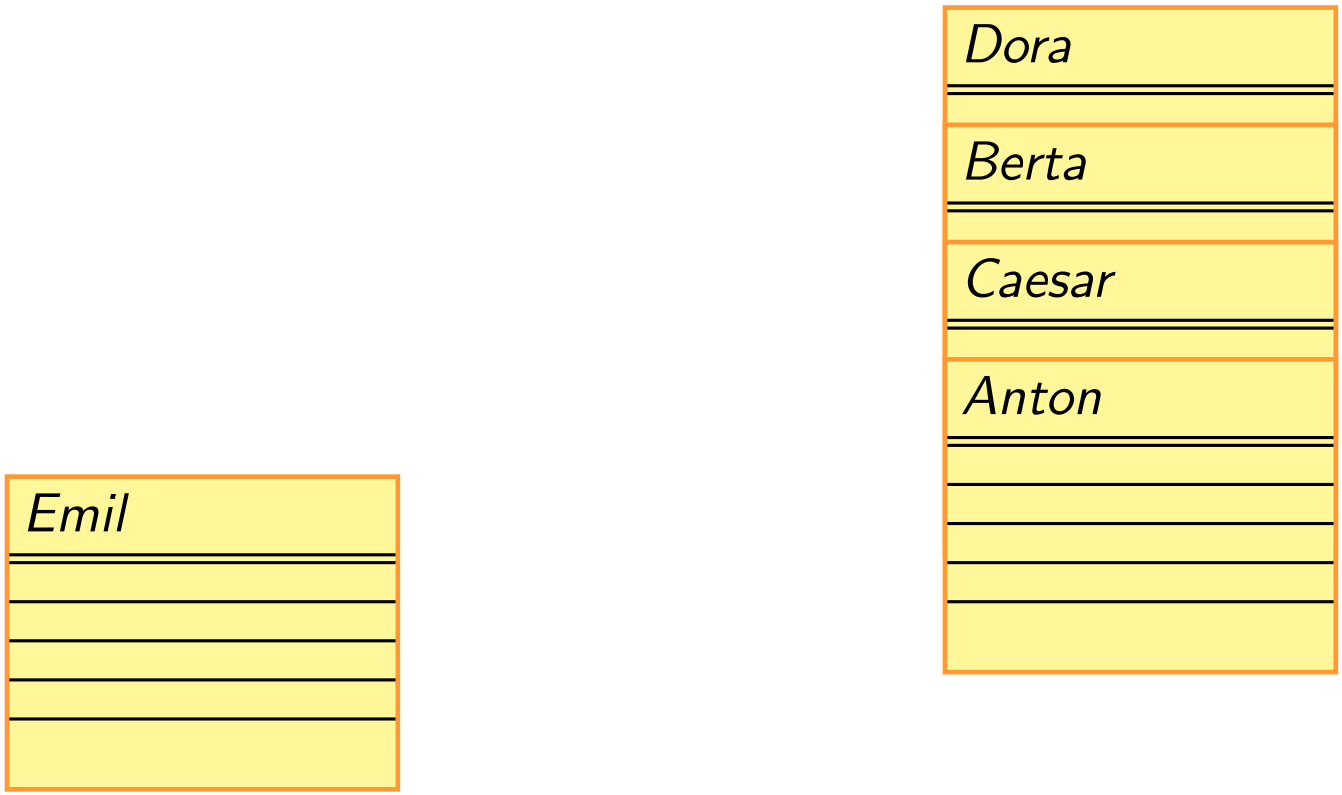


Insertion Sort



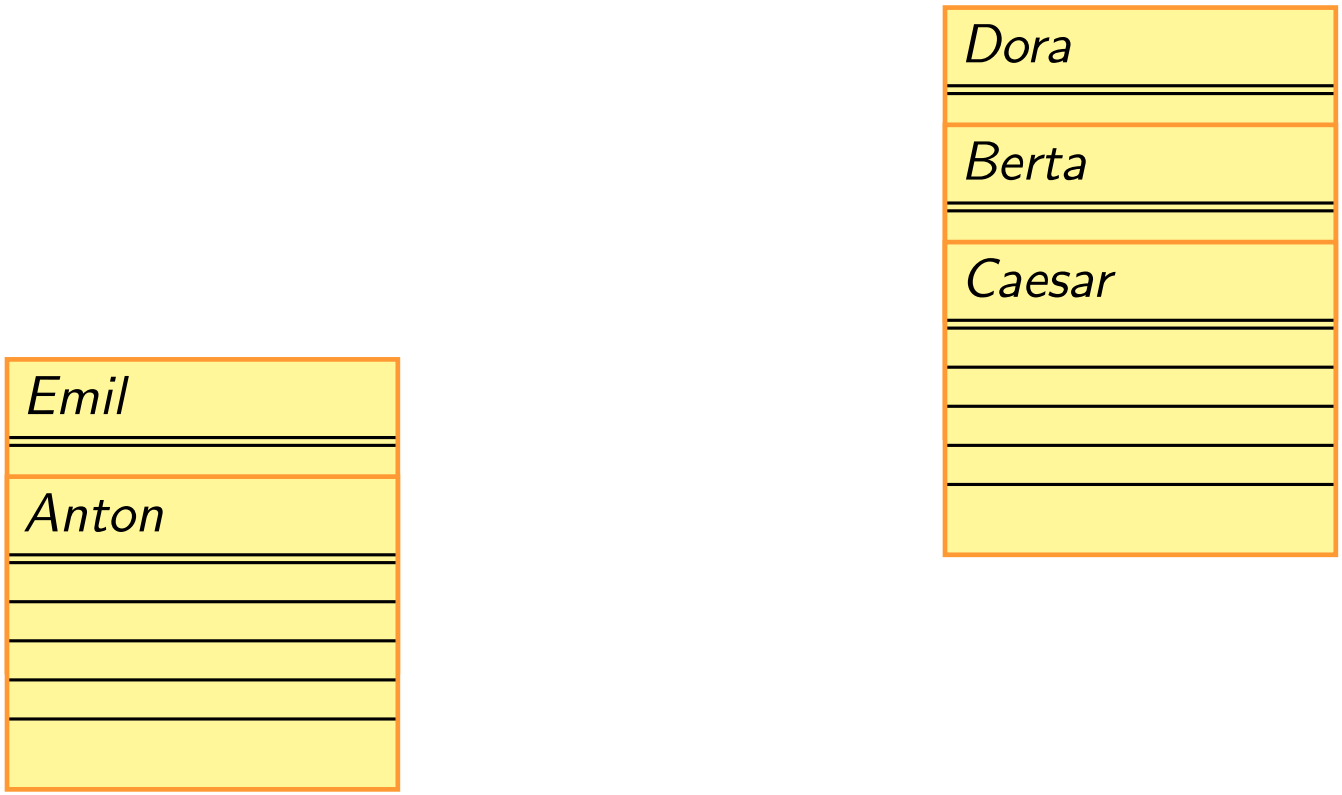
- Gustav nimmt die oberste Karte von *Emil*, und eröffnet damit den *sortierten* Stoss

Insertion Sort



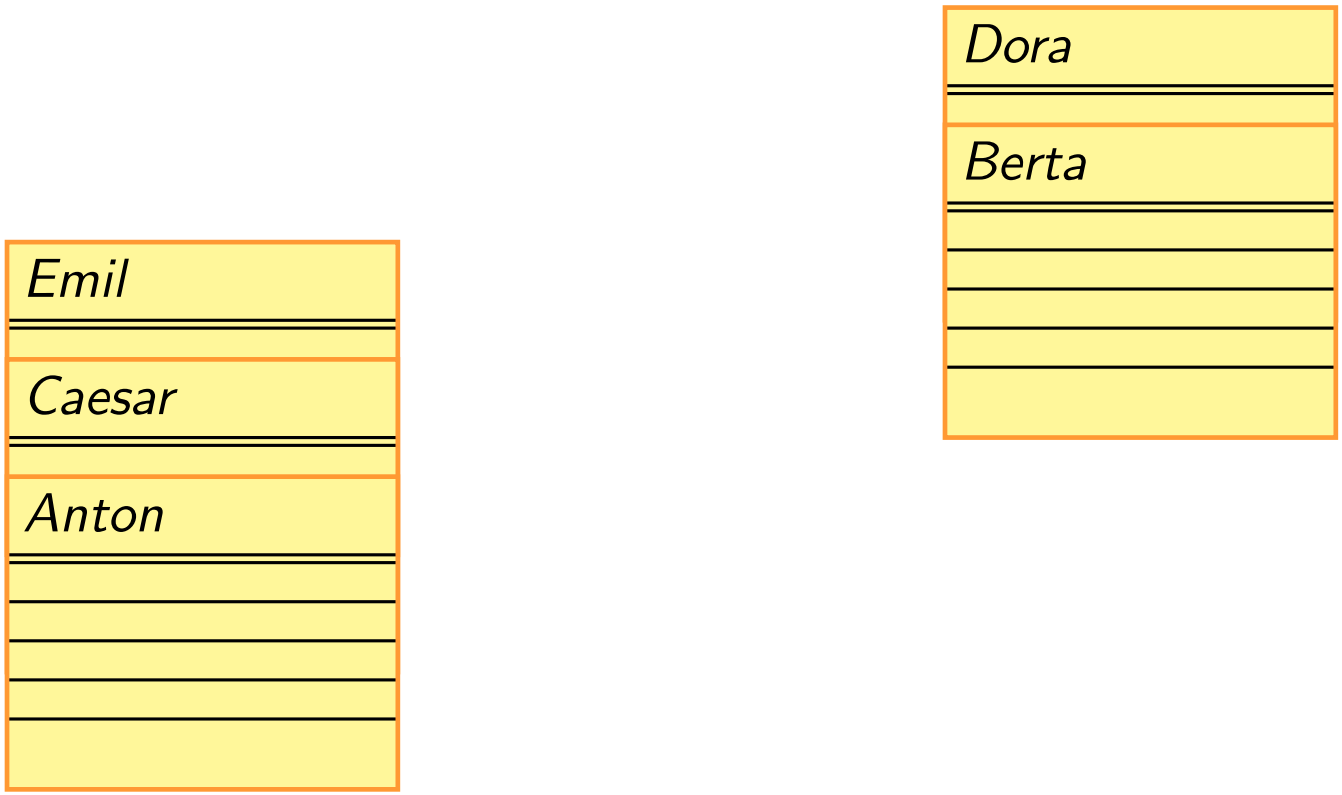
- Gustav nimmt die Karte von *Anton*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort



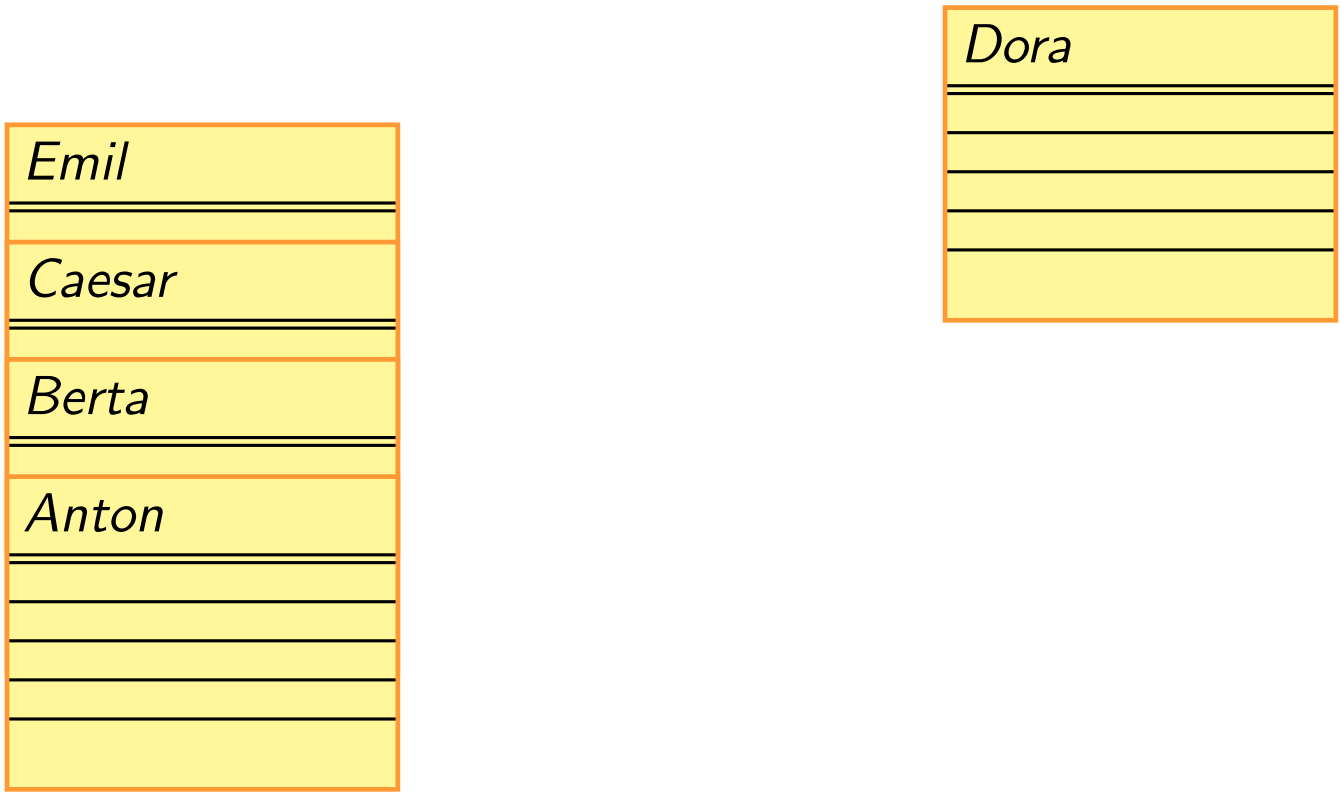
- Gustav nimmt die Karte von *Caesar*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort



- Gustav nimmt die Karte von *Berta*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort

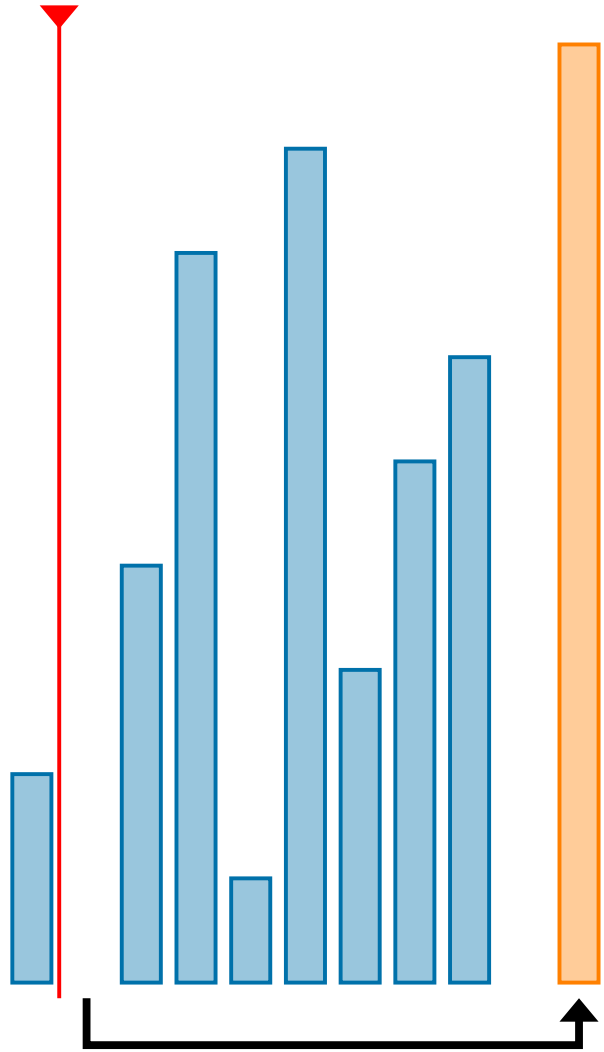


- Gustav nimmt die Karte von *Dora*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort

- Gustav hat jetzt alle seine Karteikarten geordnet, indem er stets die nächste Karte *einsortiert* hat

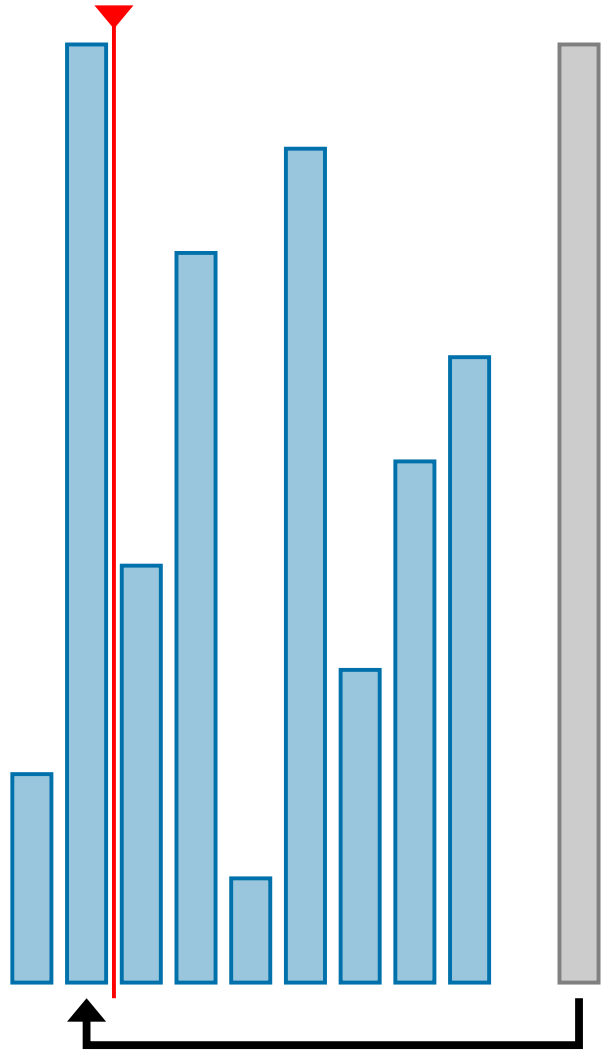
Insertion Sort



$j = 1$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

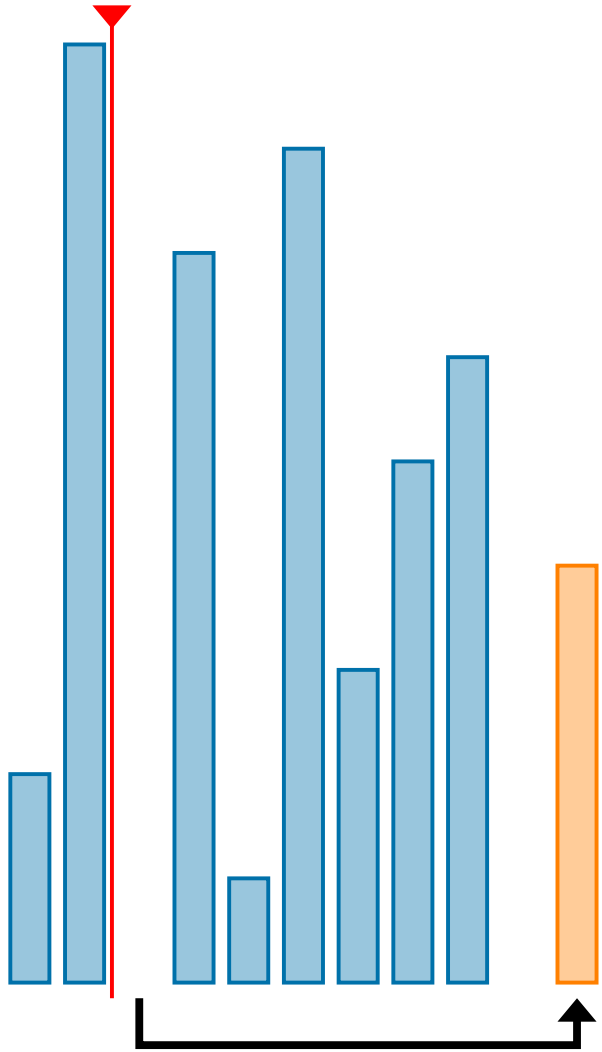
Insertion Sort



$j = 1$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

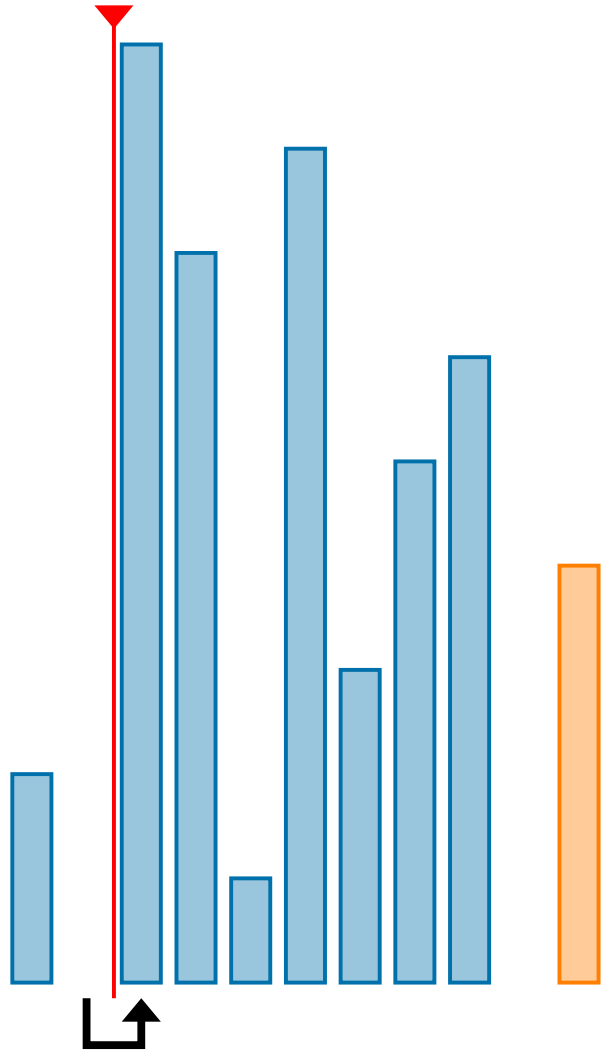
Insertion Sort



$j = 2$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

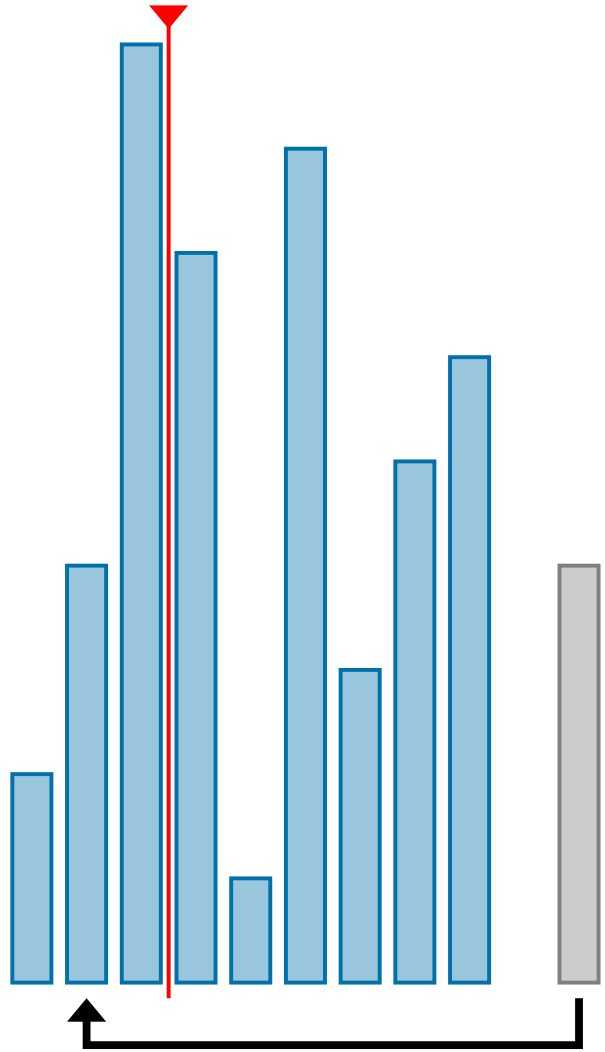
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 2$: Platz für Einfügung – **fertig**

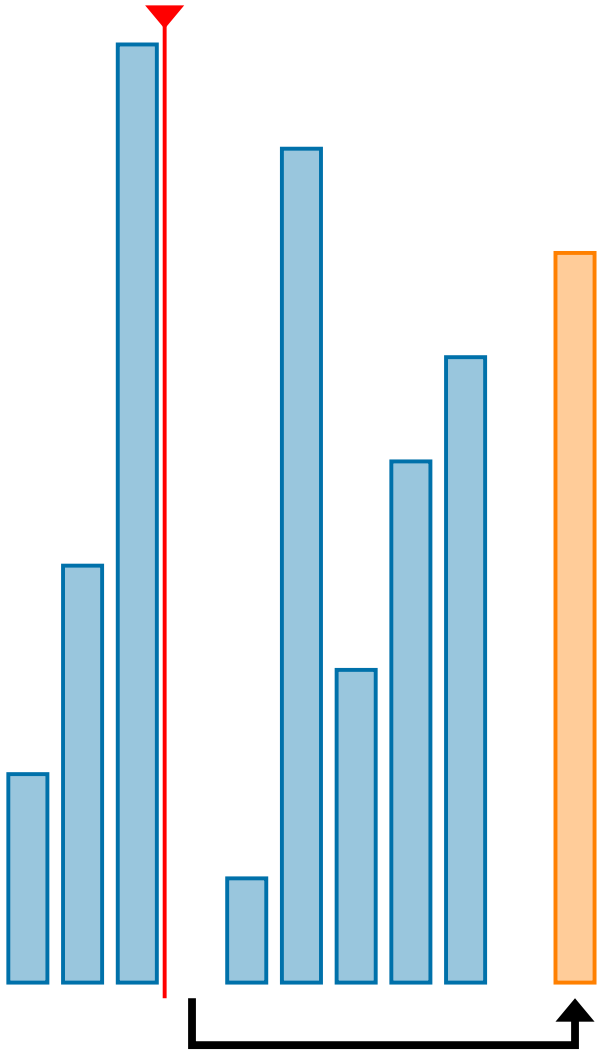
Insertion Sort



$j = 2$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

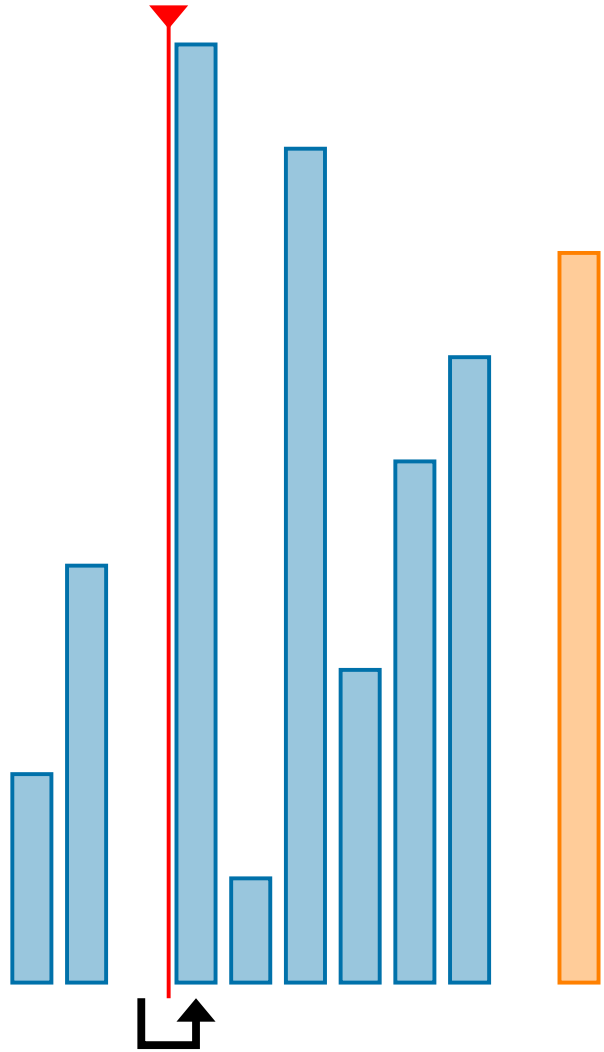
Insertion Sort



$j = 3$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

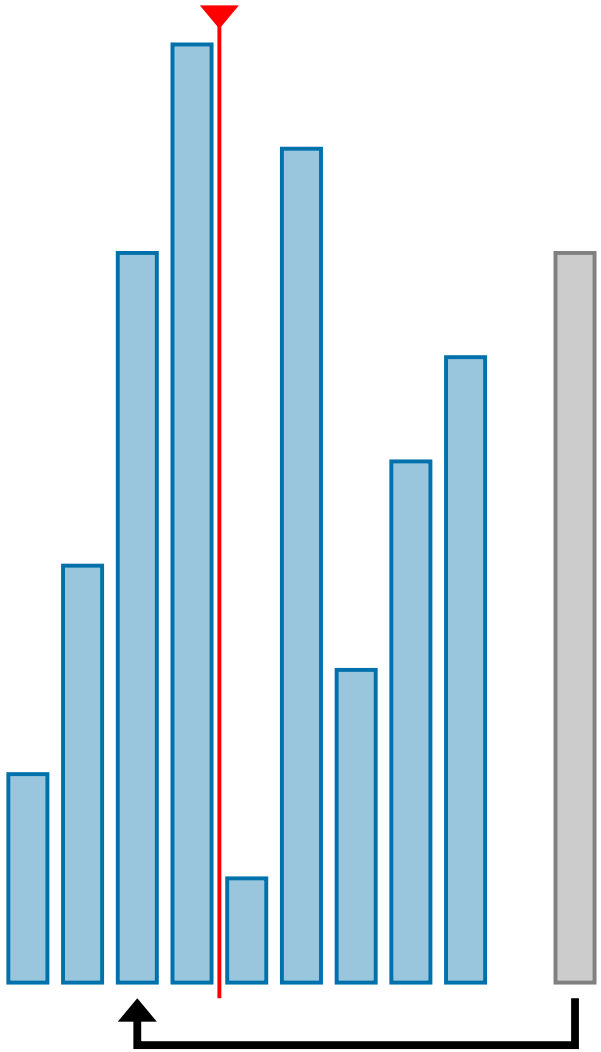
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 3$: Platz für Einfügung – **fertig**

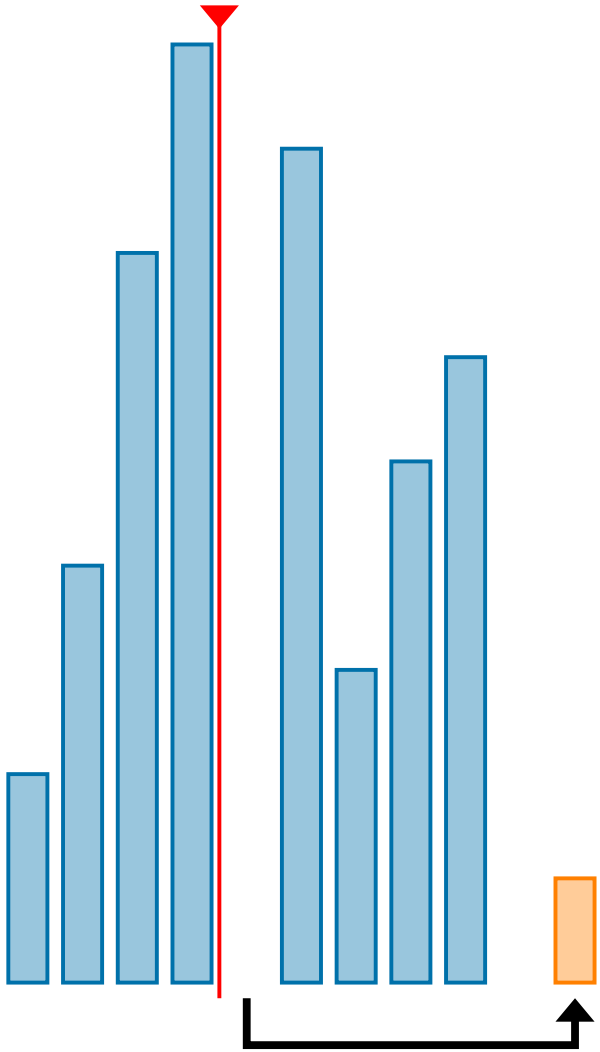
Insertion Sort



$j = 3$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

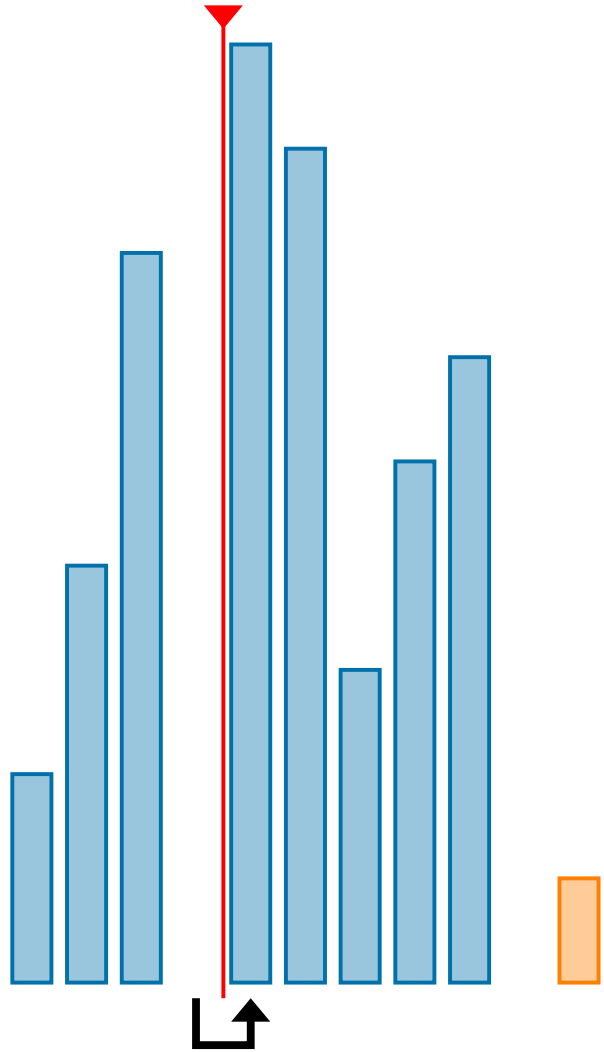
Insertion Sort



$j = 4$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

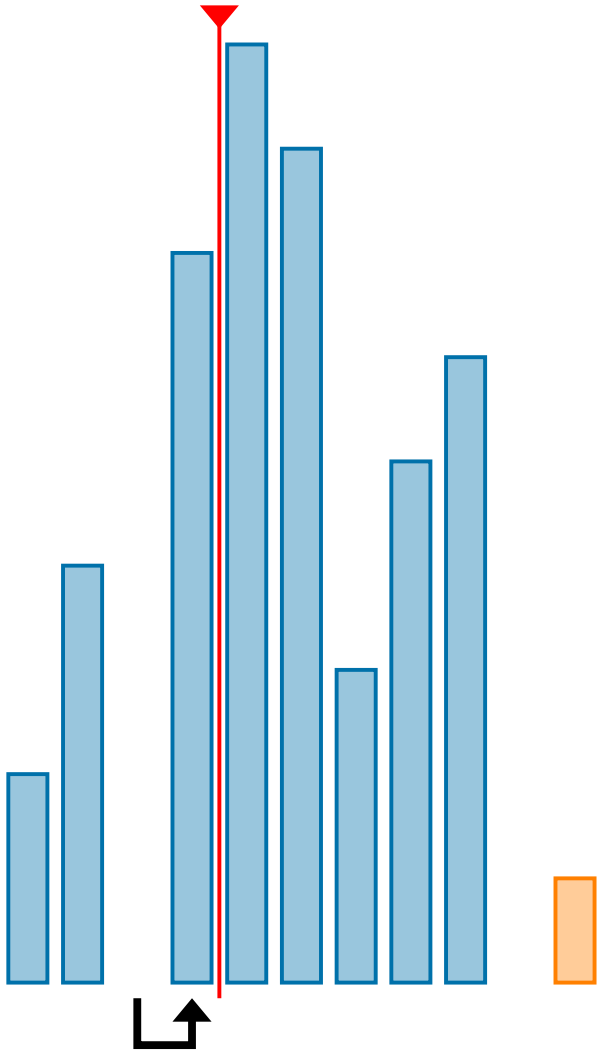
Insertion Sort



$j = 4$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

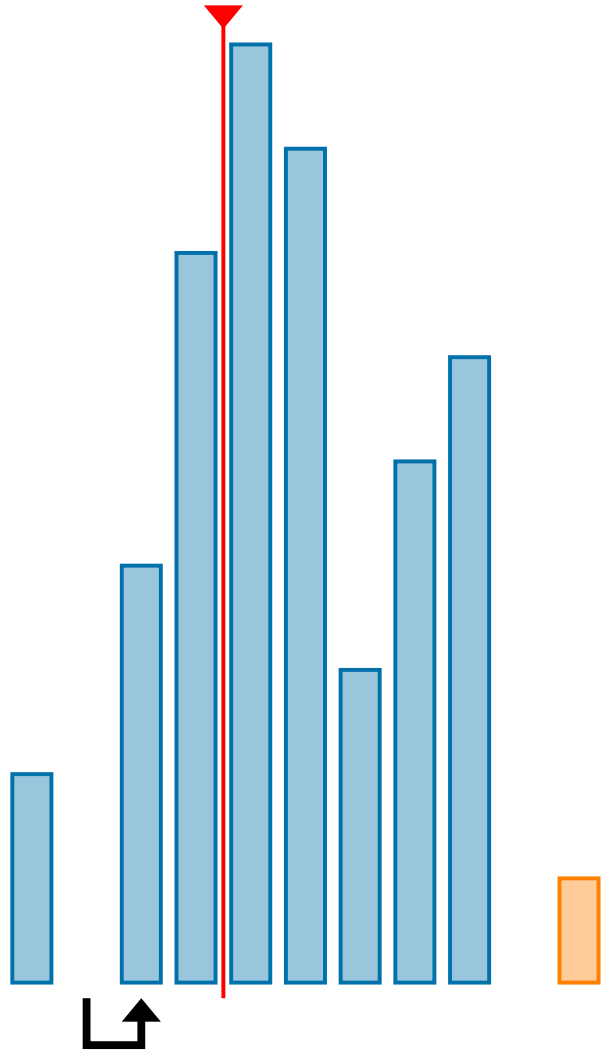
Insertion Sort



$j = 4$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

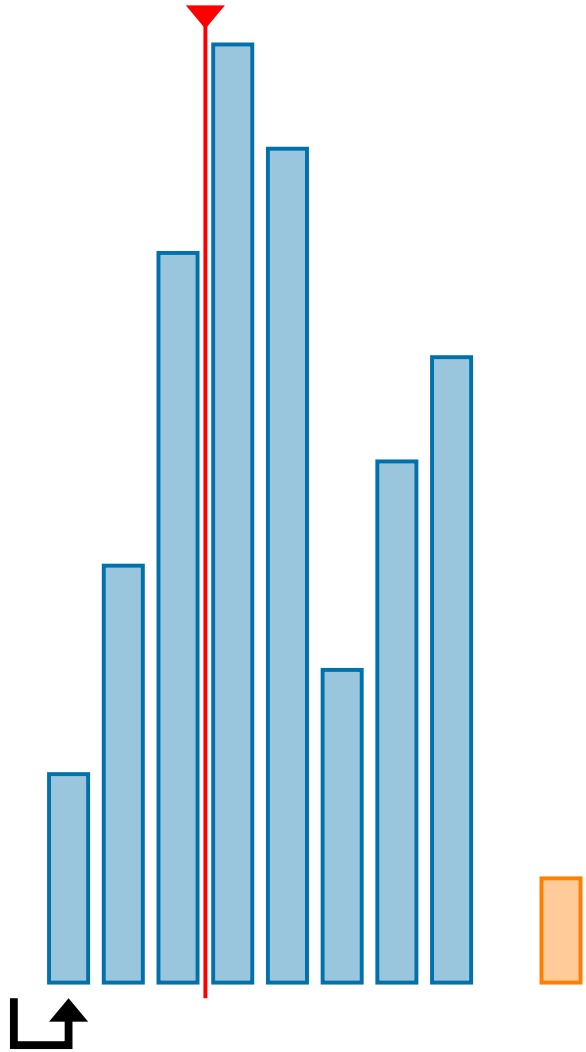
Insertion Sort



$j = 4$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

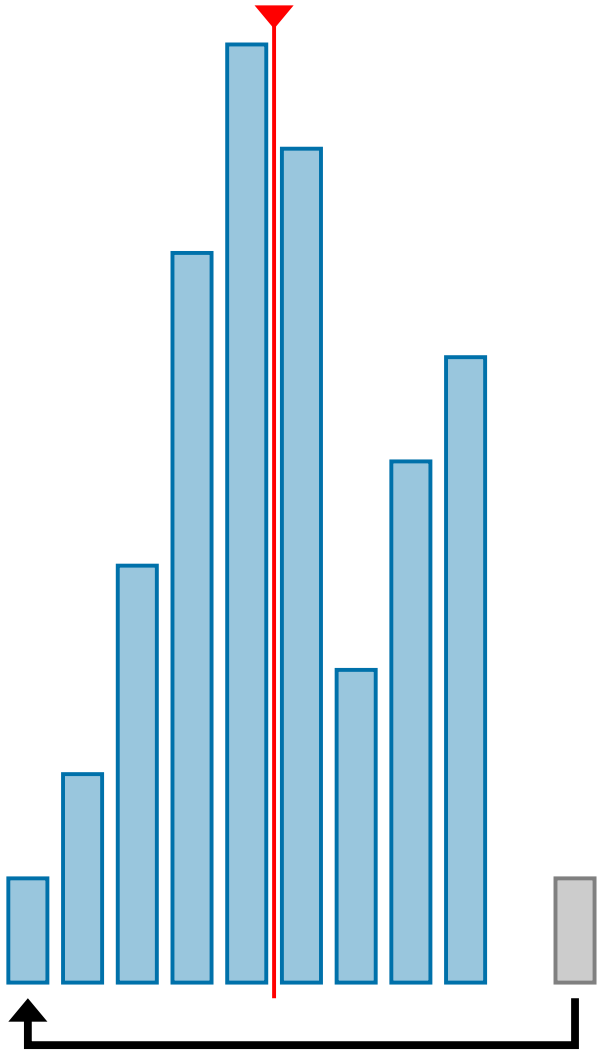
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 4$: Platz für Einfügung – **fertig**

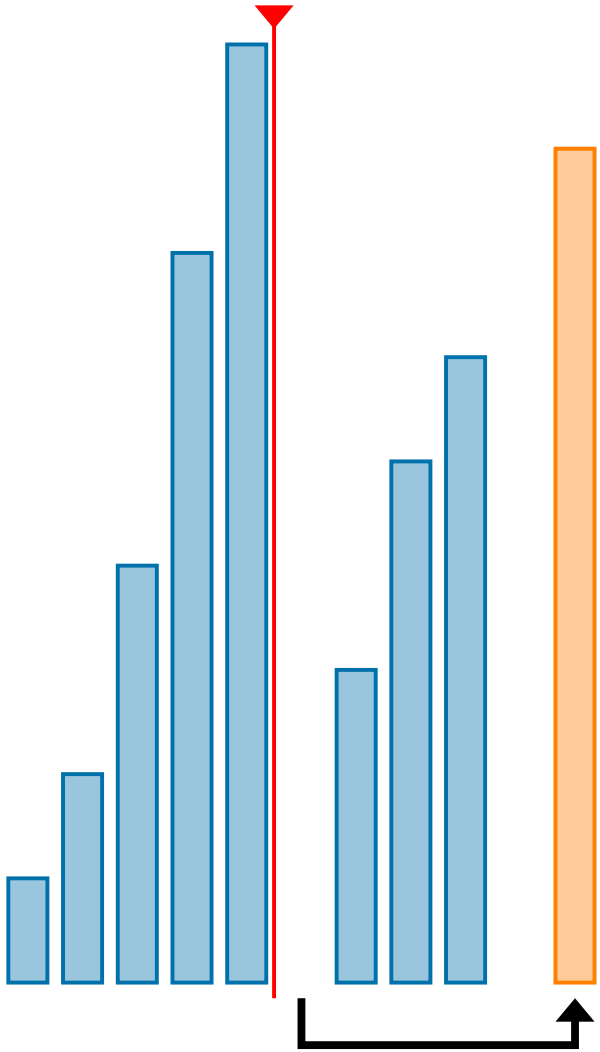
Insertion Sort



$j = 4$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

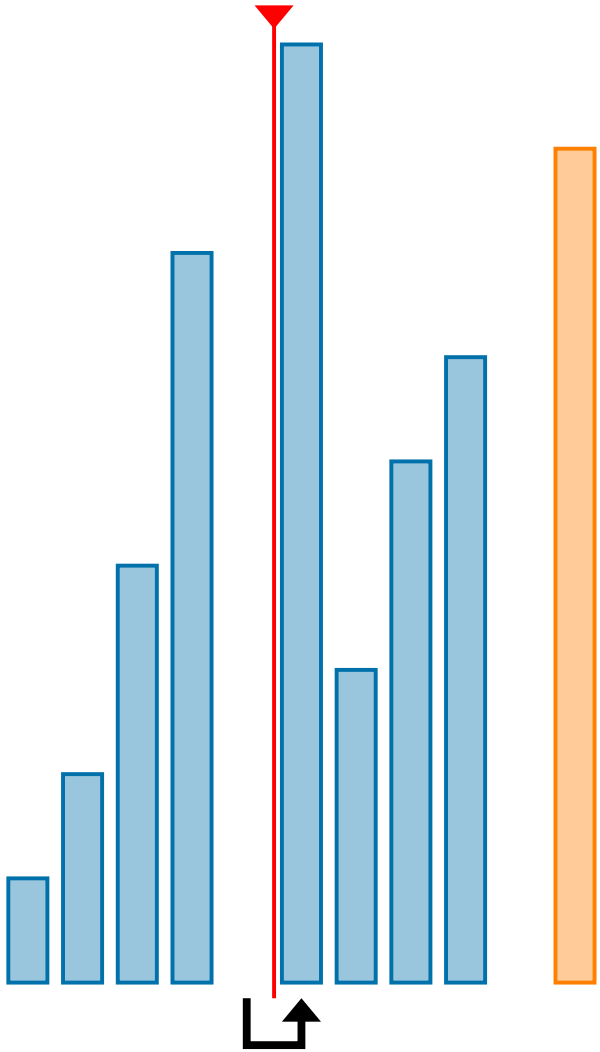
Insertion Sort



$j = 5$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

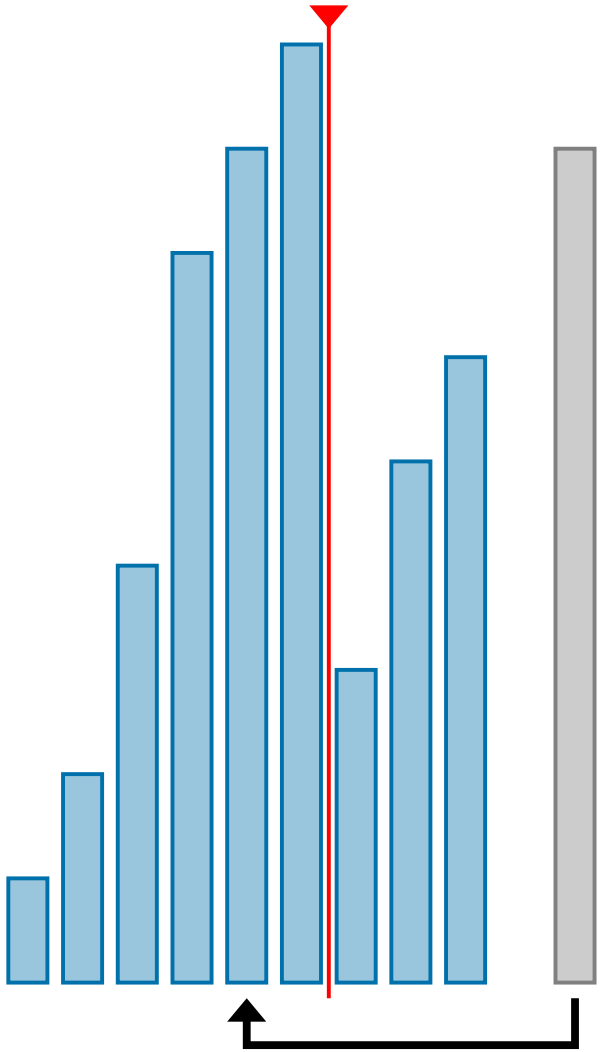
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 5$: Platz für Einfügung – **fertig**

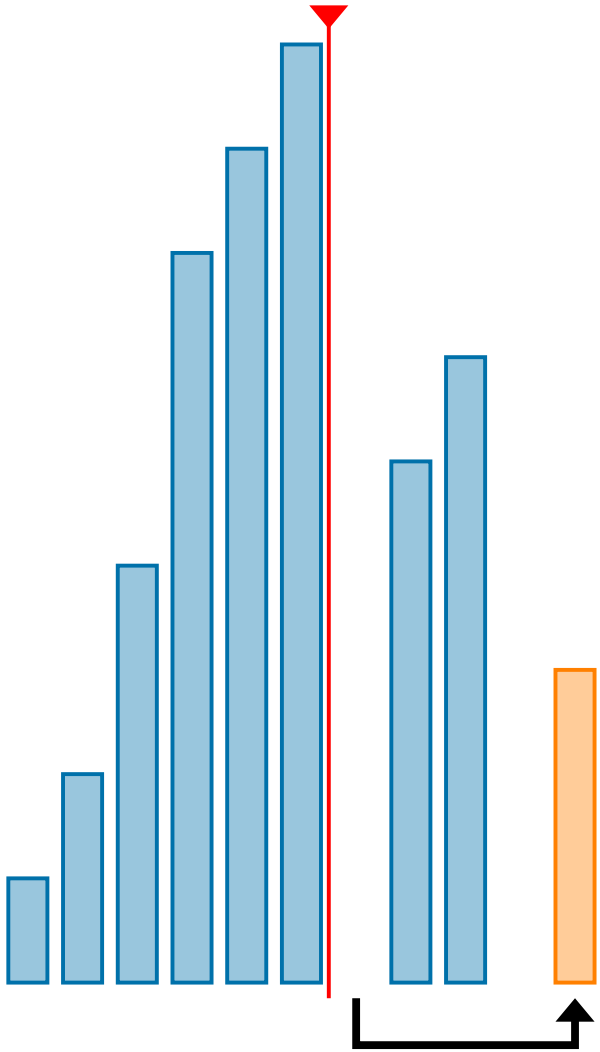
Insertion Sort



$j = 5$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

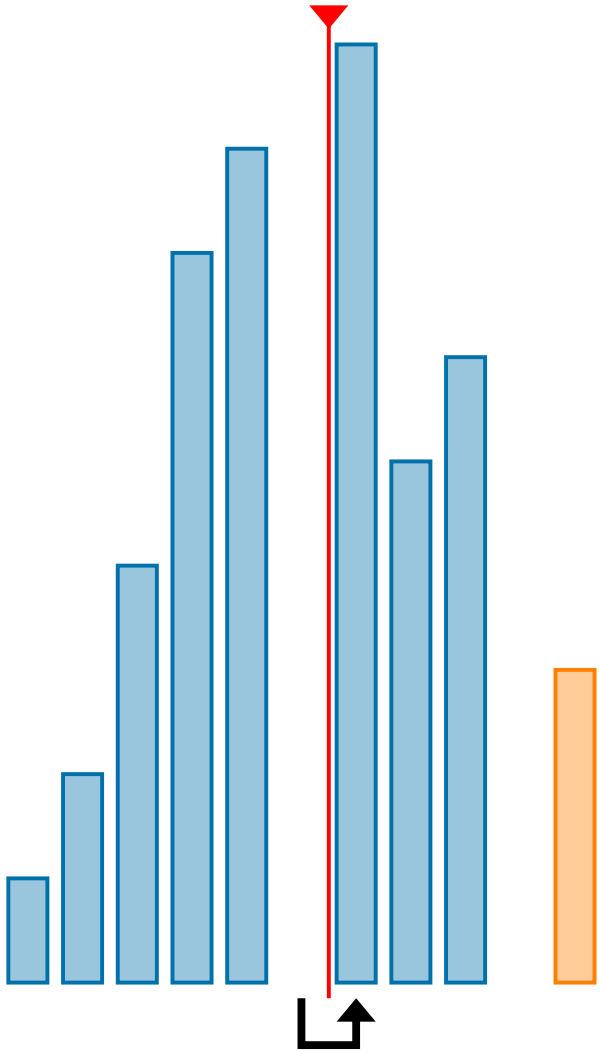
Insertion Sort



$j = 6$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

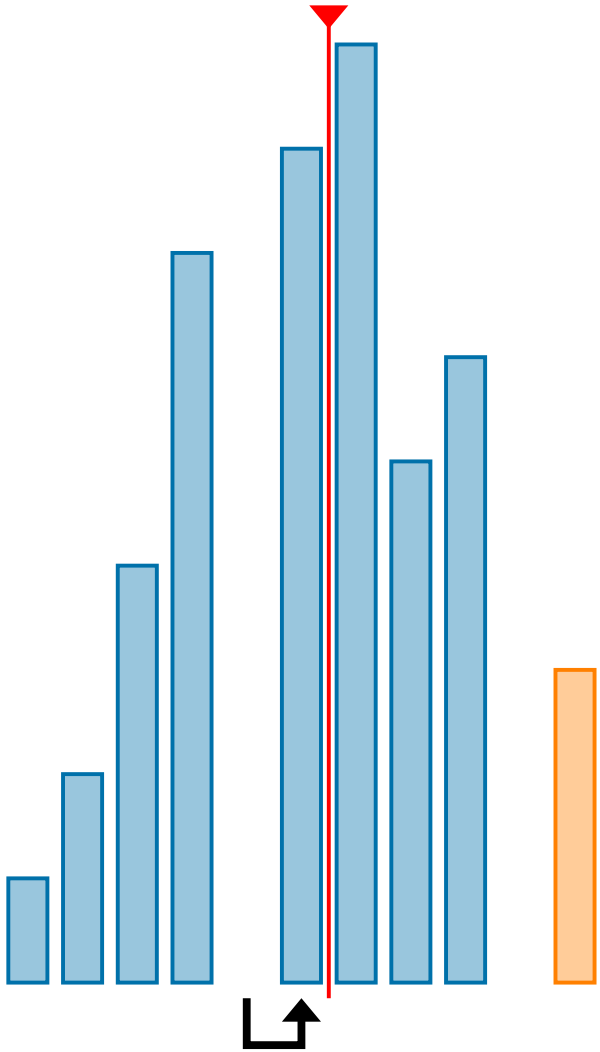
Insertion Sort



$j = 6$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

Insertion Sort

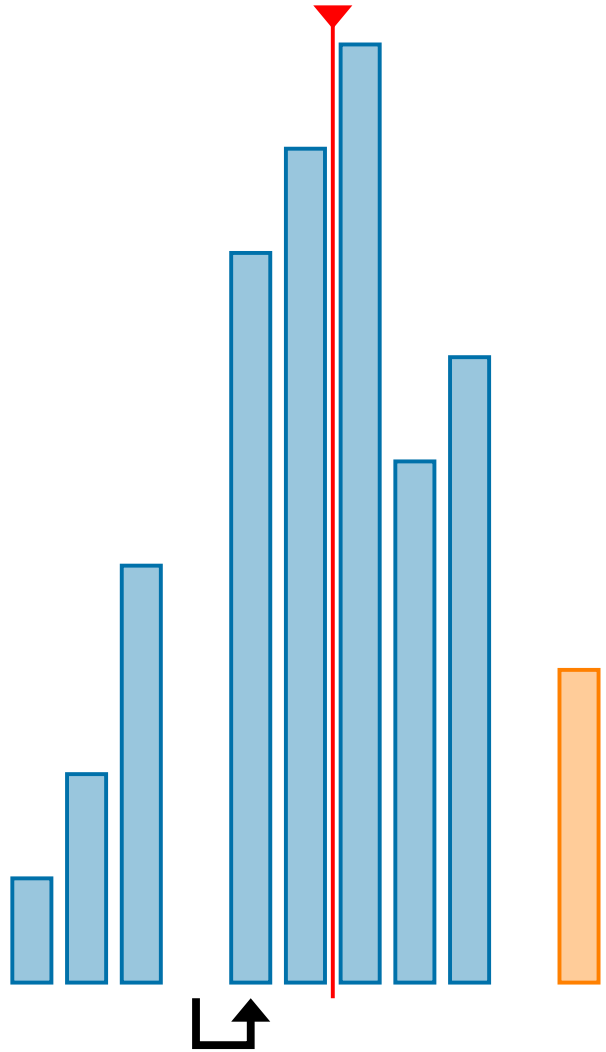


$j = 6$: Platz für Einfügung

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
InsertionSort(A):
    for j = 1 to n − 1 {
        key = A[j];
        i = j − 1;
        while (i ≥ 0 && A[i] > key) {
            A[i + 1] = A[i];
            i = i − 1;
        }
        A[i + 1] = key;
    }
```

Insertion Sort

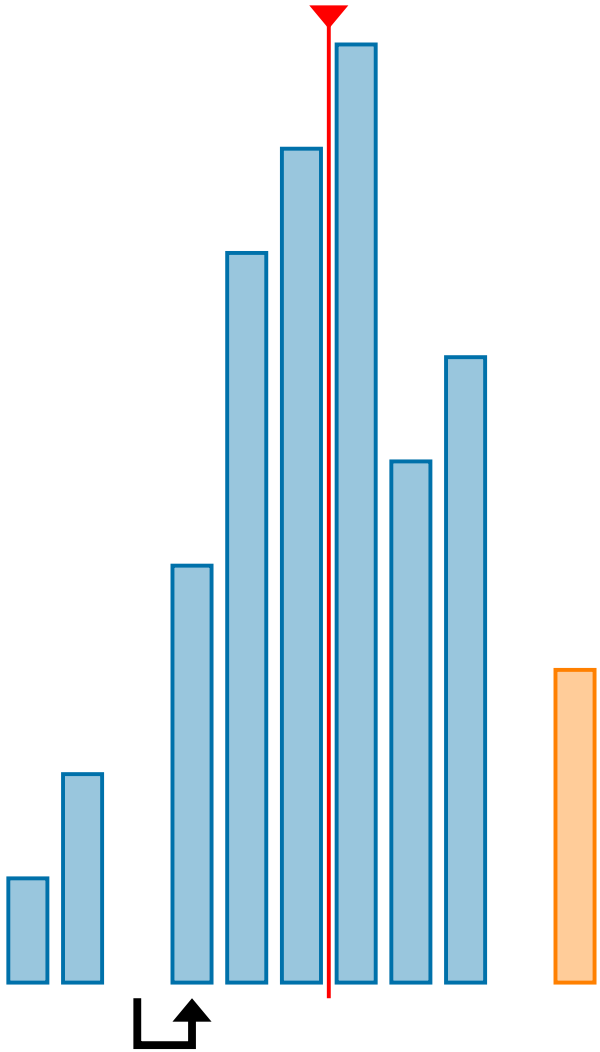


$j = 6$: Platz für Einfügung

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
InsertionSort(A):
    for j = 1 to n − 1 {
        key = A[j];
        i = j − 1;
        while (i ≥ 0 && A[i] > key) {
            A[i + 1] = A[i];
            i = i − 1;
        }
        A[i + 1] = key;
    }
```

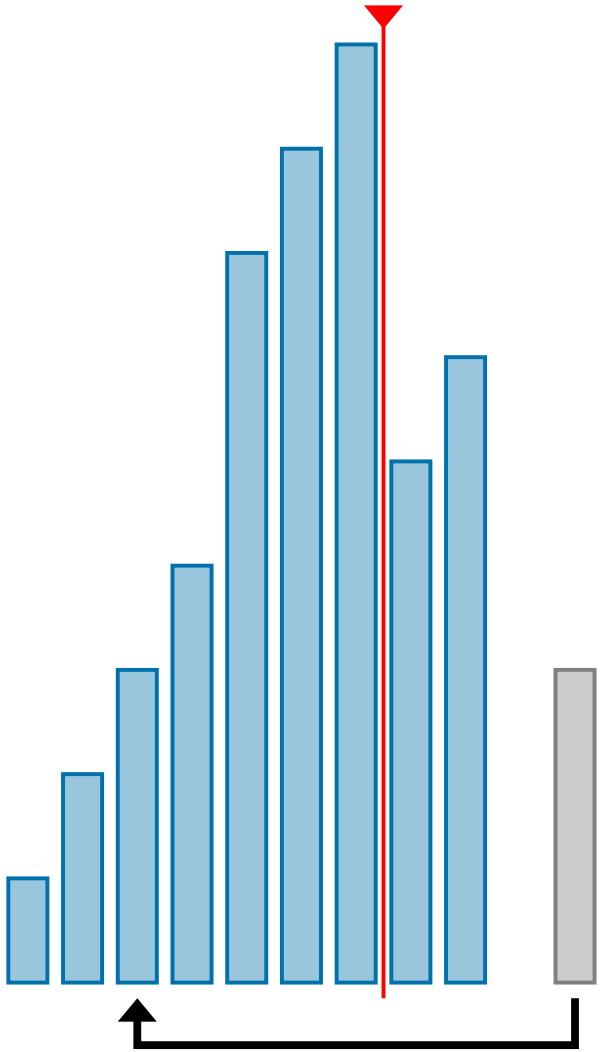
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 6$: Platz für Einfügung – **fertig**

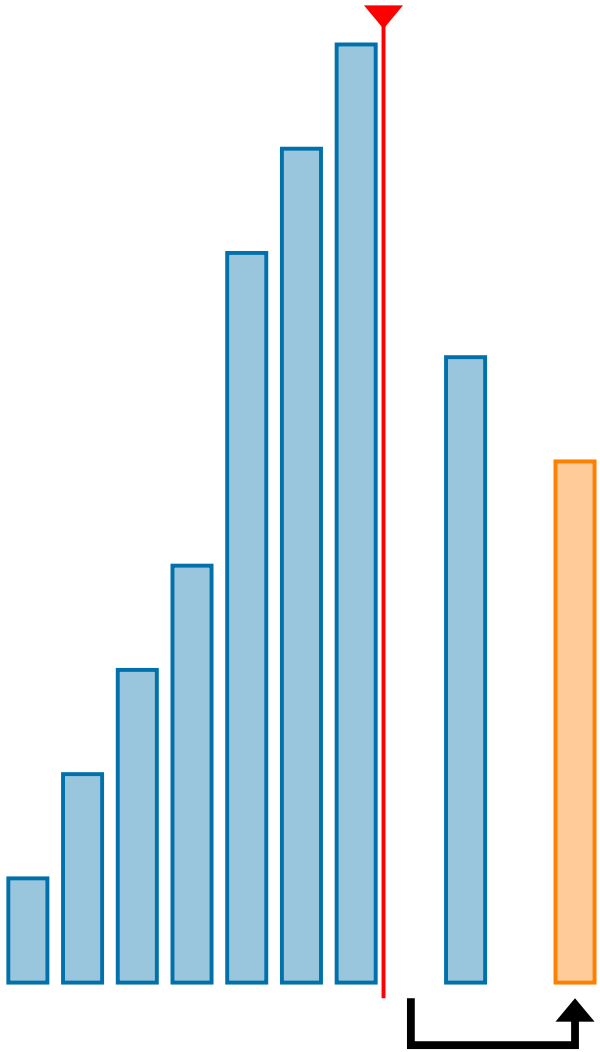
Insertion Sort



$j = 6$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

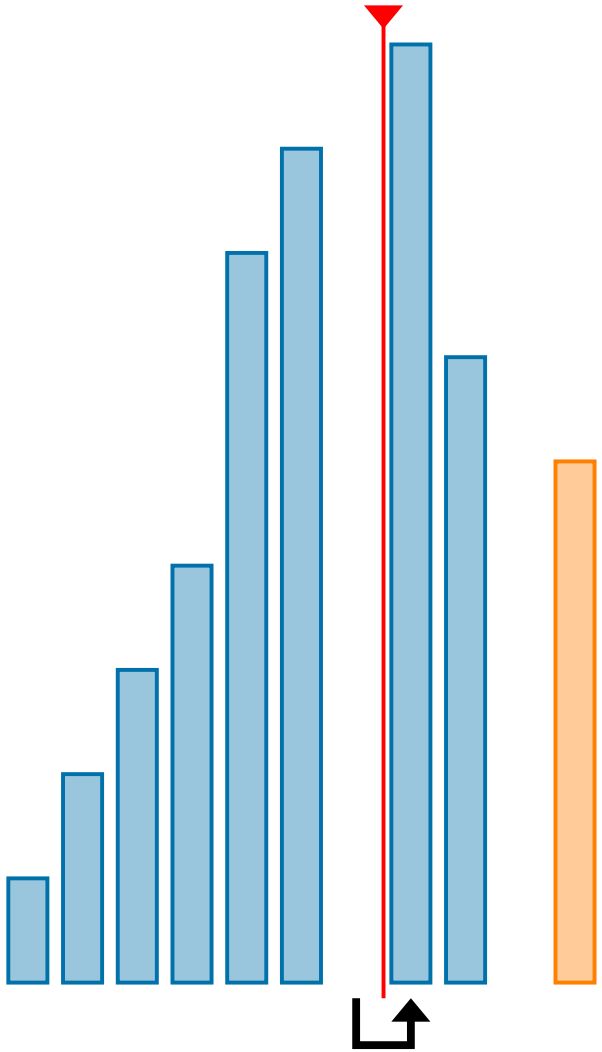
Insertion Sort



$j = 7$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

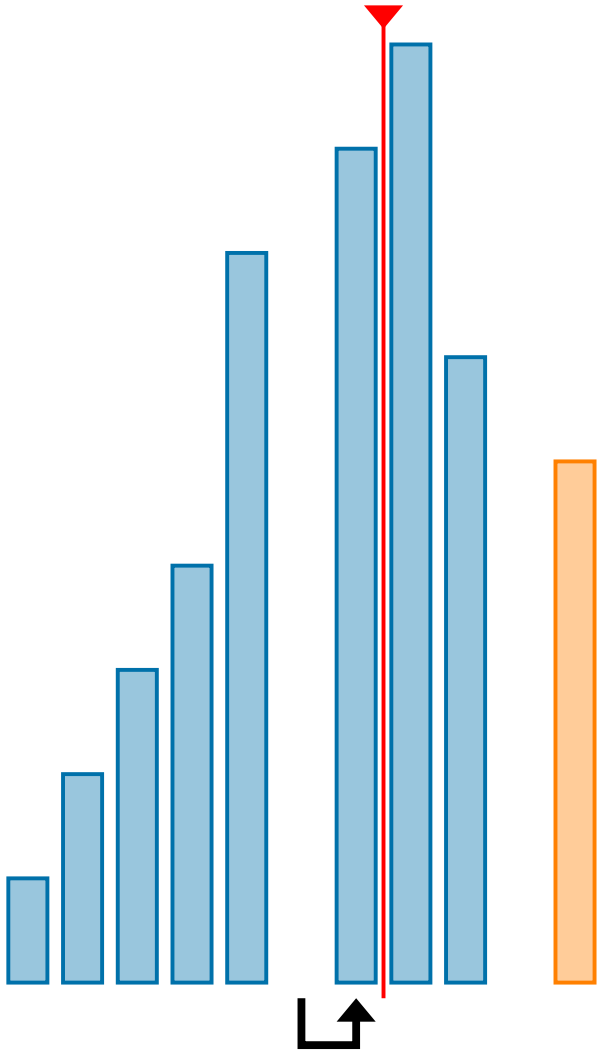
Insertion Sort



$j = 7$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

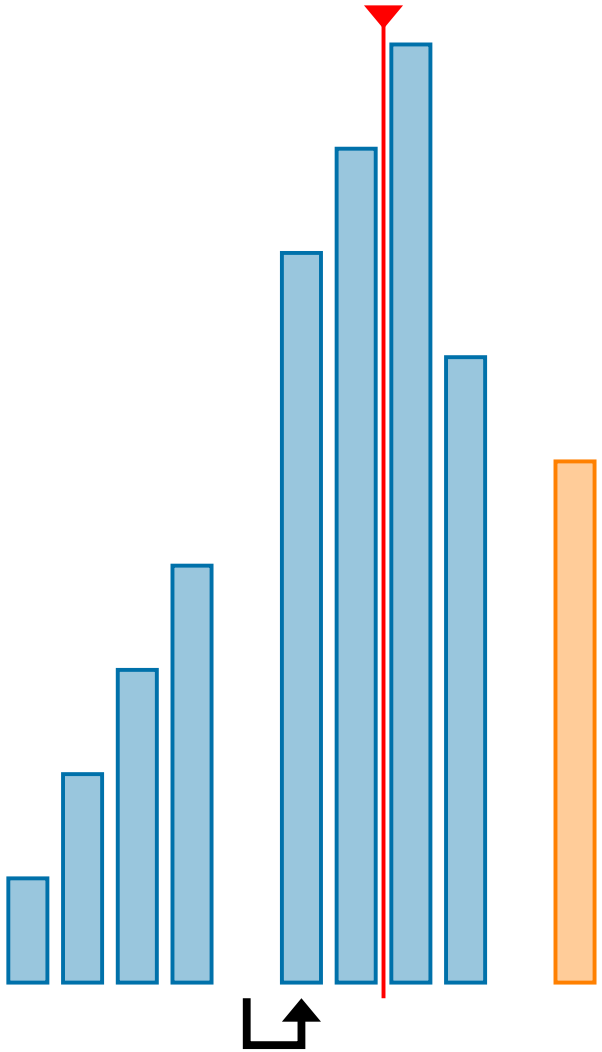
Insertion Sort



$j = 7$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

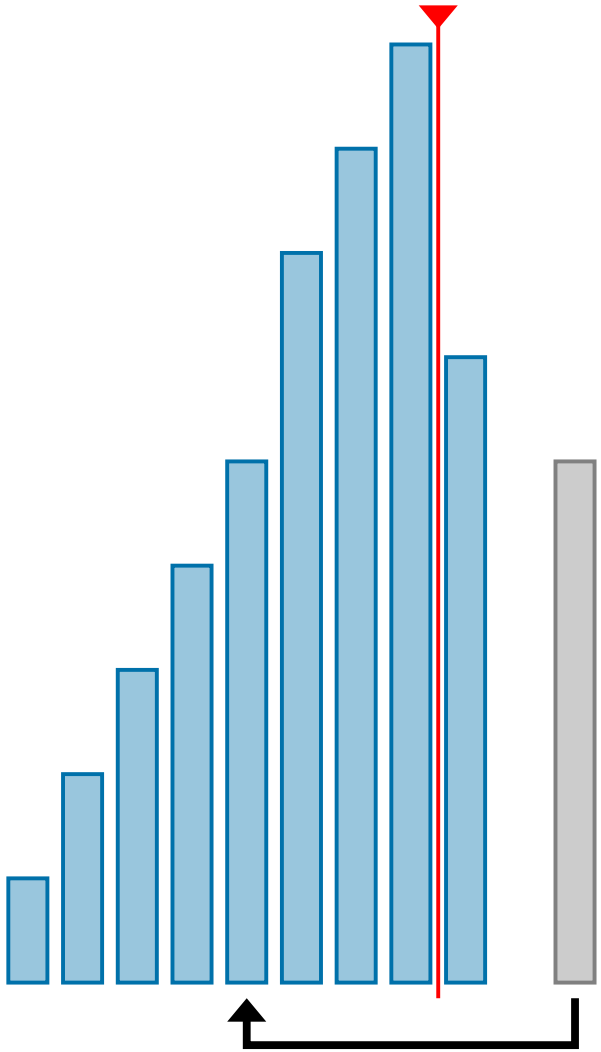
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 7$: Platz für Einfügung – **fertig**

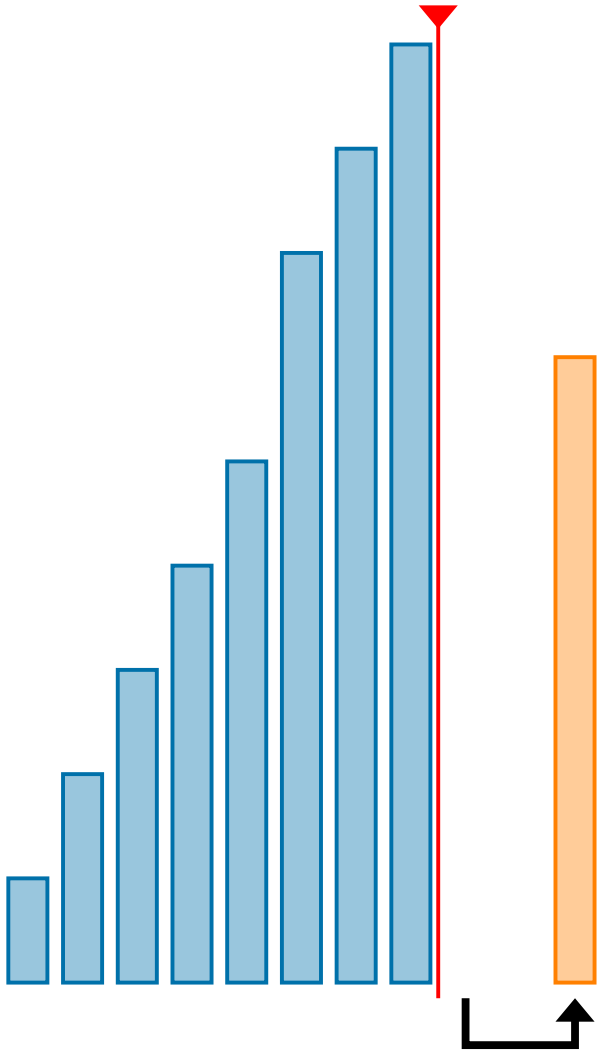
Insertion Sort



$j = 7$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

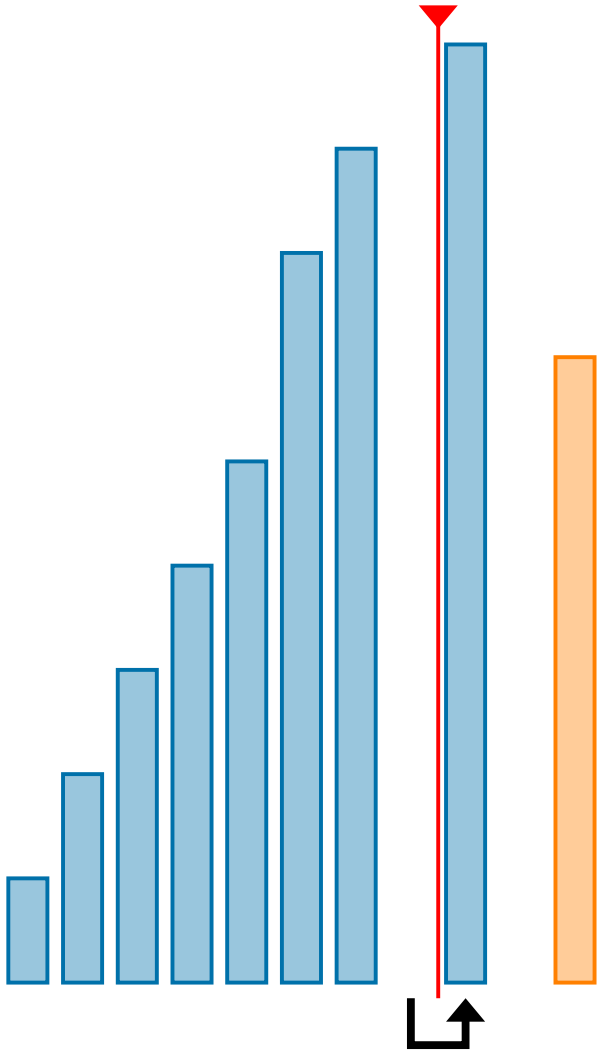
Insertion Sort



$j = 8$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

Insertion Sort

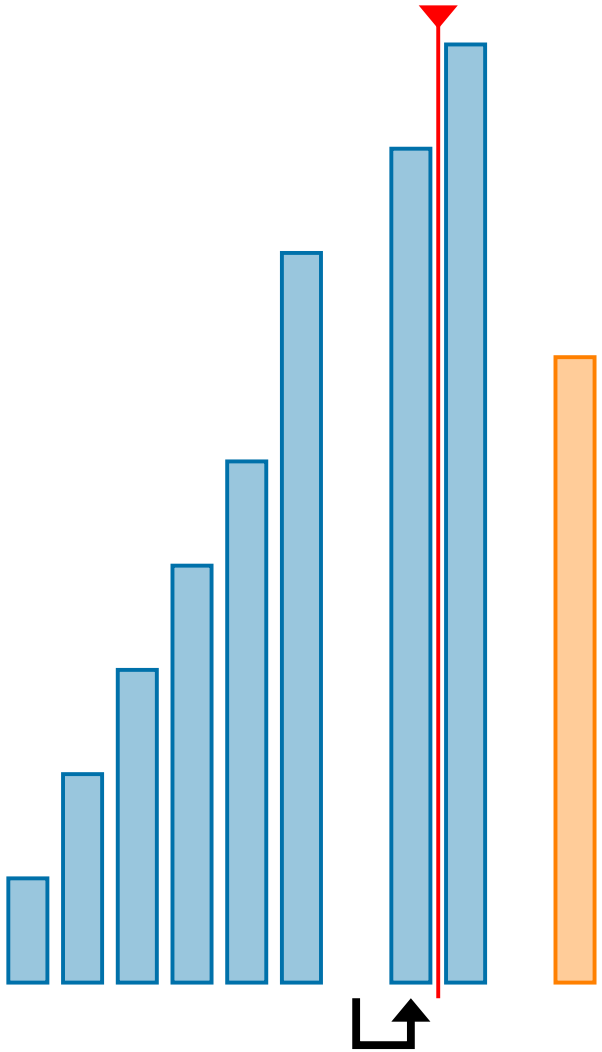


$j = 8$: Platz für Einfügung

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
InsertionSort(A):
    for j = 1 to n − 1 {
        key = A[j];
        i = j − 1;
        while (i ≥ 0 && A[i] > key) {
            A[i + 1] = A[i];
            i = i − 1;
        }
        A[i + 1] = key;
    }
```

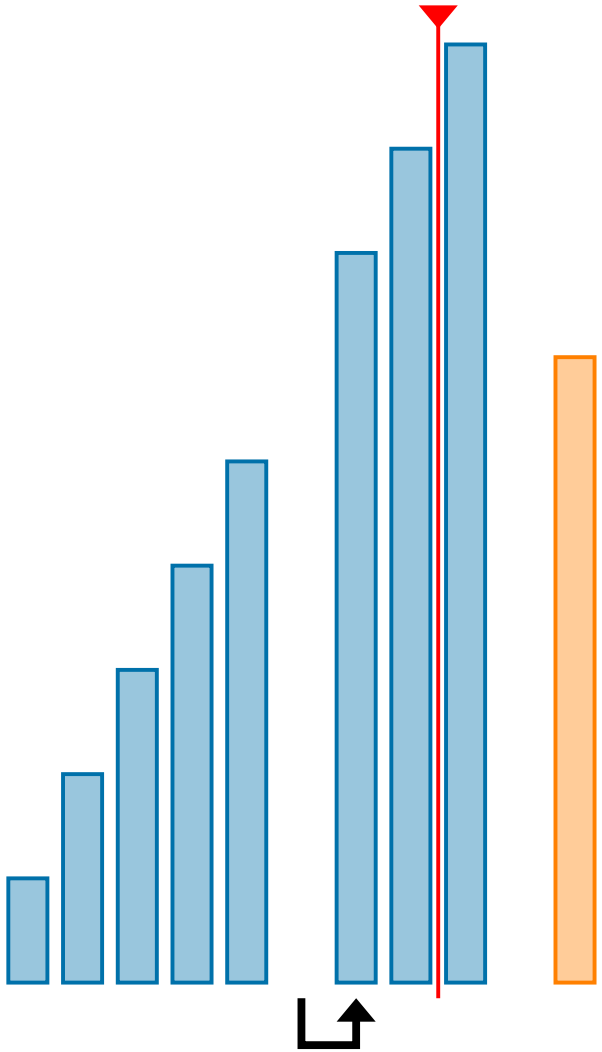
Insertion Sort



$j = 8$: Platz für Einfügung

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

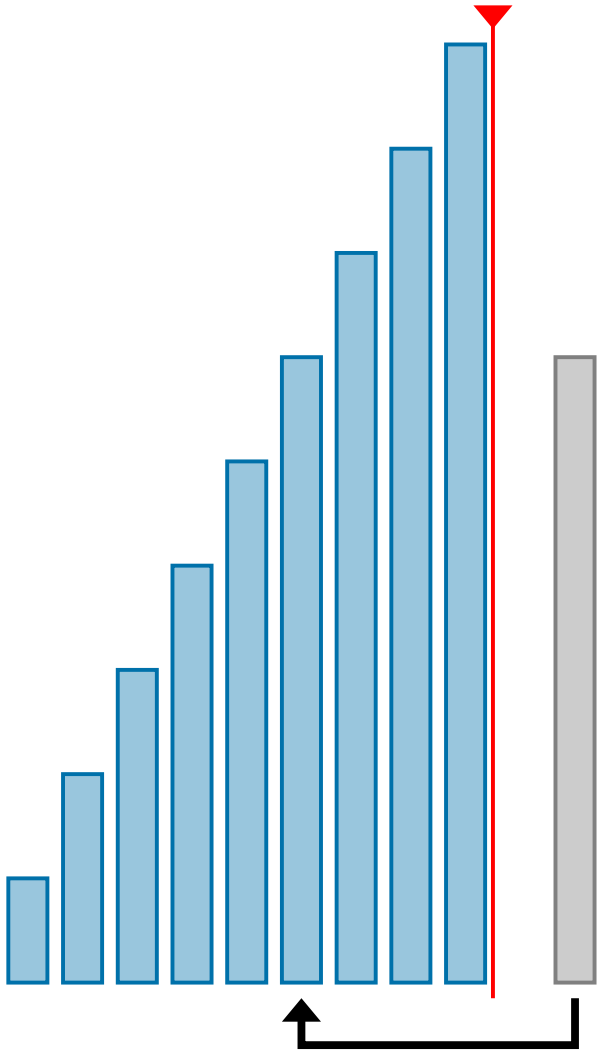
Insertion Sort



```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

$j = 8$: Platz für Einfügung – **fertig**

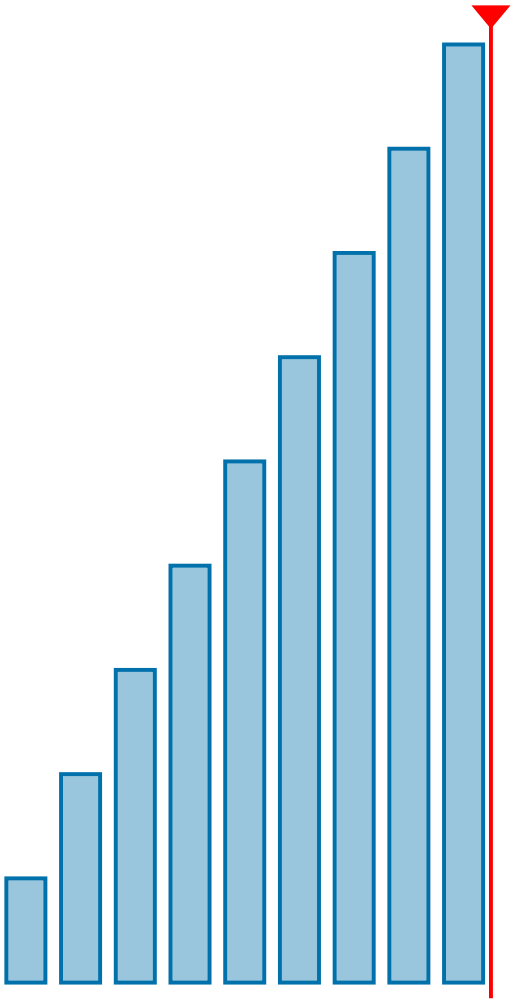
Insertion Sort



$j = 8$: Endzustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
InsertionSort(A):
        for j = 1 to n − 1 {
                key = A[j];
                i = j − 1;
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

Insertion Sort



Endzustand

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
InsertionSort(A):
    for j = 1 to n − 1 {
        key = A[j];
        i = j − 1;
        while (i ≥ 0 && A[i] > key) {
            A[i + 1] = A[i];
            i = i − 1;
        }
        A[i + 1] = key;
    }
```

Verifikation

- „Verfolgung“ logischer Bedingungen durch Code
  - Vorbedingung $\{VOR\}$
  - Nachbedingung $\{NACH\}$
- Kette von Anweisungen $\{A\}\ \alpha;\beta\ \{B\}$
  - $\{A\}\ \alpha\ \{C\}\ \beta\ \{B\}$
- Bedingte Verarbeitung $\{A\}$ **if** (C) $\{\alpha\}$ **else** $\{\beta\}$ $\{B\}$
  - $\{A \wedge C\}\ \alpha\ \{B\}$
  - $\{A \wedge \neg C\}\ \beta\ \{B\}$
- Schleifen $\{A\}$ **while** (C) $\{\beta\}$ $\{B\}$ mit Invariante P
  - P gilt am Anfang: $A \Rightarrow P$
  - P bleibt gültig: $\{P \wedge C\}\ \beta\ \{P\}$
  - $P \wedge \neg C \Rightarrow B$

Multiplikation durch Addition

```
Multiply(a, b):
    if (a = 0 ∨ b = 0) {
        c = 0;
    }
    else if (a < 0) {
        c = −Multiply(−a, b);
    }
    else {
        c = 0;
        i = 1;
        while (i ≤ a) {
            c = c + b;
            i = i + 1;
        }
    }
    return c;
```

Vor- und Nachbedingung

- Die Parameter sind zwei beliebige ganzzahlige Werte

$$VOR \quad = \quad a \in \mathbb{Z} \quad \wedge \quad b \in \mathbb{Z}$$

- Das Ergebnis c ist das Produkt dieser Werte:

$$NACH \quad = \quad (c = a \cdot b)$$

Abkürzungen

Für mehr Übersichtlichkeit ersetzen wir die Anweisungsblöcke zur Berechnung von c durch Symbole α, β und γ

Multiply(a, b)**:**

```
if (a = 0 ∨ b = 0) {
        c = 0;
}
else if (a < 0) {
        c = −Multiply(−a, b);
}
else {
        c = 0;
        i = 1;
        while (i ≤ a) {
                c = c + b;
                i = i + 1;
        }
}
return c;
```

Abkürzungen

Für mehr Übersichtlichkeit ersetzen wir die Anweisungsblöcke zur Berechnung von c durch Symbole α, β und γ

Multiply(a, b)**:**

if $(a = 0 \ \vee \ b = 0)$ {

α

}

else if $(a < 0)$ {

β

}

else {

γ

}

Korrektheit der **Multiply**: Fallunterscheidung

Zunächst verifizieren wir die Fallunterscheidung:

$\{VOR\}$
if $(a = 0 \ \vee \ b = 0)$ {
α
}
else if $(a < 0)$ {
β
}
else {
γ
}
$\{NACH\}$

Korrektheit der **Multiply**: Fallunterscheidung

Dazu betrachten wir die drei Fälle separat:

- $\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$
- $\{VOR \ \wedge \ \neg(a = 0 \ \vee \ b = 0) \ \wedge \ a < 0\} \ \beta \ \{NACH\}$
- $\{VOR \ \wedge \ \neg(a = 0 \ \vee \ b = 0) \ \wedge \ \neg(a < 0)\} \ \gamma \ \{NACH\}$

Korrektheit der **Multiply**: Fallunterscheidung

Dazu betrachten wir die drei Fälle separat:

- $\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$
- $\{VOR \ \wedge \ a < 0 \ \wedge \ b \neq 0\} \ \beta \ \{NACH\}$
- $\{VOR \ \wedge \ \neg(a = 0 \ \vee \ b = 0) \ \wedge \ \neg(a < 0)\} \ \gamma \ \{NACH\}$

Korrektheit der **Multiply**: Fallunterscheidung

Dazu betrachten wir die drei Fälle separat:

- $\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$
- $\{VOR \ \wedge \ a < 0 \ \wedge \ b \neq 0\} \ \beta \ \{NACH\}$
- $\{VOR \ \wedge \ a > 0 \ \wedge \ b \neq 0\} \ \gamma \ \{NACH\}$

Korrektheit der **Multiply**: Fall 1

$$\{VOR \ \wedge \ (a = 0 \ \vee \ b = 0)\} \ \alpha \ \{NACH\}$$

Es gilt ja:

$$a = 0 \ \vee \ b = 0 \quad \Rightarrow \quad a \cdot b = 0$$

α setzt in diesem Fall $c = 0$, also gilt $c = a \cdot b$, und die Nachbedingung ist erfüllt.

- Dieser Fall ist **korrekt**!

Korrektheit der **Multiply**: Fall 2

$$\{VOR \ \wedge \ a < 0 \ \wedge \ b \neq 0\} \ \beta \ \{NACH\}$$

Es gilt ja:

$$a < 0 \quad \Rightarrow \quad a = -|a| = -1 \cdot (-a)$$

Bei $a < 0$ und beliebigem b gilt weiter wegen Assoziativgesetzes:

$$a \cdot b = (-1 \cdot (-a)) \cdot b = -1 \cdot ((-a) \cdot b)$$

β setzt in diesem Fall $c = -\text{Multiply}(-a, b)$, also gilt $c = a \cdot b$, und die Nachbedingung ist erfüllt, falls die Methode im dritten Fall korrekt ist.

- Dieser Fall ist (wahrscheinlich) **korrekt**!

Korrektheit der **Multiply**: Fall 3

$$\{VOR \ \wedge \ a > 0 \ \wedge \ b \neq 0\} \ \gamma \ \{NACH\}$$

Für die Korrektheit der Schleife in γ benötigt man die *Invariante*:

$c = 0;$
$i = 1;$
while $(i \leq a)$ {
    $c = c + b;$
    $i = i + 1;$
}

Der bereits in c aufaddierte Teil und der noch aufzuaddierende Teil müssen zusammen das korrekte Ergebnis $a \cdot b$ ergeben:

$$P \quad = \quad (a \cdot b = c + (a - i + 1) \cdot b)$$

Bei Zählung ab 0 wäre das schöner!

Abkürzungen

Wieder ersetzen wir Anweisungsblöcke:

$\{C_0\}$
$c = 0;$
$i = 1;$
$\{C_1\}$
while $(i \leq a)$ {
$\{C_2\}$
$c = c + b;$
$i = i + 1;$
$\{C_3\}$
}
$\{NACH\}$

Abkürzungen

Wieder ersetzen wir Anweisungsblöcke:

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ $\{$

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

$\}$

$\{NACH\}$

Korrektheit der **Multiply**: Schleife

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ {

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

}

$\{NACH\}$

Die Korrektheit der Schleife wird in drei Schritten gezeigt:

- Schleifeneintritt: $C_1 \Rightarrow P$
- Schleifenkörper: $\{P \ \wedge \ (i \leq a)\} \ \gamma_{\text{körper}} \ \{P\}$
- Schleifenende: $P \ \wedge \ \neg(i \leq a) \Rightarrow NACH$

Korrektheit der **Multiply**: Schleifeneintritt

$$C_1 \Rightarrow P$$

- C_0 ergibt sich aus der Vorbedingung des dritten Falles

$$C_0 \quad := \quad (VOR \ \wedge \ a > 0 \ \wedge \ b \neq 0)$$

- Prüfe $\{C_0\}\ \gamma_{\text{init}}\ \{C_1\}$

$$C_1 \quad := \quad (C_0 \ \wedge \ c = 0 \ \wedge \ i = 1)$$

- Einfügung in die Formel aus P ergibt *wahre* Aussage

$$a \cdot b = c + (a - i + 1) \cdot b = 0 + (a - 1 + 1) \cdot b = a \cdot b$$

- Schleifeneintritt ist **korrekt**!

Korrektheit der **Multiply**: Schleife

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ {

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

}

$\{NACH\}$

Die Korrektheit der Schleife wird in drei Schritten gezeigt:

- Schleifeneintritt: $C_1 \Rightarrow P$
- Schleifenkörper: $\{P \ \wedge \ (i \leq a)\} \ \gamma_{\text{körper}} \ \{P\}$
- Schleifenende: $P \ \wedge \ \neg(i \leq a) \Rightarrow NACH$

Korrektheit der **Multiply**: Schleifenkörper

$$\{C_2 := (P \ \wedge \ (i \leq a))\} \ \gamma_{\text{körper}} \ \{C_3 := P\}$$

- In $\gamma_{\text{körper}}$ werden *neue* Werte gesetzt:

$$c' = c + b \qquad i' = i + 1$$

- Invariante galt *vorher*:

$$a \cdot b = c + (a - i + 1) \cdot b$$

- Prüfung der Invariante *nachher*:

$$\begin{aligned} c' + (a - i' + 1) \cdot b &= (c + b) + (a - (i + 1) + 1) \cdot b \\ &= c + b + (a - i) \cdot b \\ &= c + (a - i + 1) \cdot b \\ &= a \cdot b \end{aligned}$$

- Schleifenkörper ist **korrekt**!

Korrektheit der **Multiply**: Schleife

$\{C_0\}$

γ_{init}

$\{C_1\}$

while $(i \leq a)$ $\{$

$\{C_2\}$

$\gamma_{\text{körper}}$

$\{C_3\}$

$\}$

$\{NACH\}$

Die Korrektheit der Schleife wird in drei Schritten gezeigt:

- Schleifeneintritt: $C_1 \Rightarrow P$
- Schleifenkörper: $\{P \ \wedge \ (i \leq a)\} \ \gamma_{\text{körper}} \ \{P\}$
- Schleifenende: $P \ \wedge \ \neg(i \leq a) \Rightarrow NACH$

Korrektheit der **Multiply**: Schleifenende

$$P \ \wedge \ \neg(i \leq a) \Rightarrow \mathit{NACH}$$

- Wegen schrittweiser Erhöhung gilt sogar

$$i = a + 1$$

- Einfügen in Invariante ergibt *NACH*:

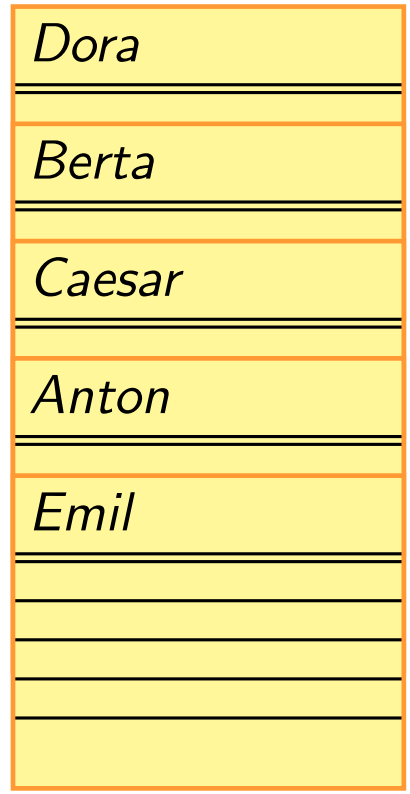
$$\begin{aligned} a \cdot b &= c + (a - i + 1) \cdot b \\ &= c + (a - (a + 1) + 1) \cdot b \\ &= c + (a - a - 1 + 1) \cdot b \\ &= c + 0 \cdot b \\ &= c \end{aligned}$$

- Schleifenende (und damit der ganze dritte Fall) **korrekt**!

Korrektheit der **Multiply**: Zusammenfassung

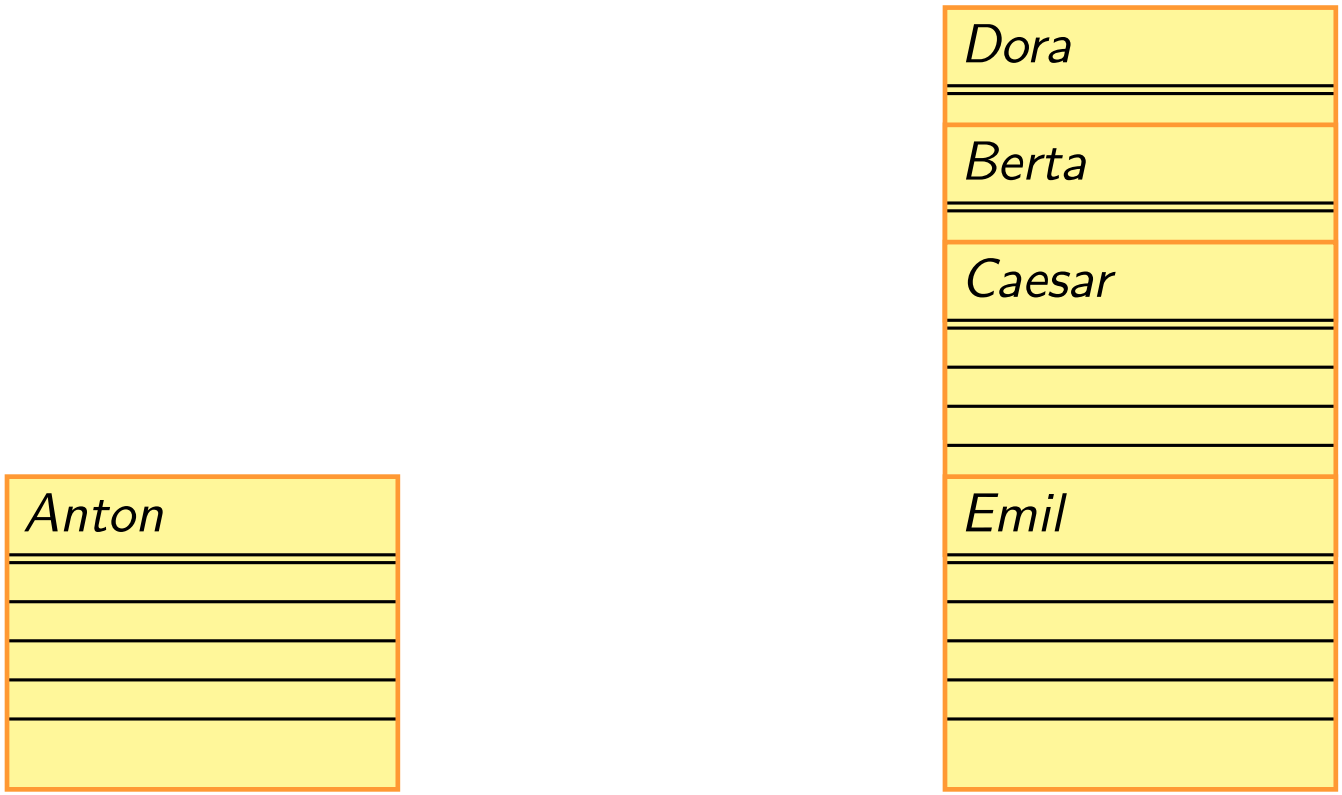
- Untersuchung der drei Fälle der Fallunterscheidung
- Letzter Fall erfordert Untersuchung der Schleife
  - Schleifeneintritt
  - Schleifenkörper
  - Schleifenende

Selection Sort



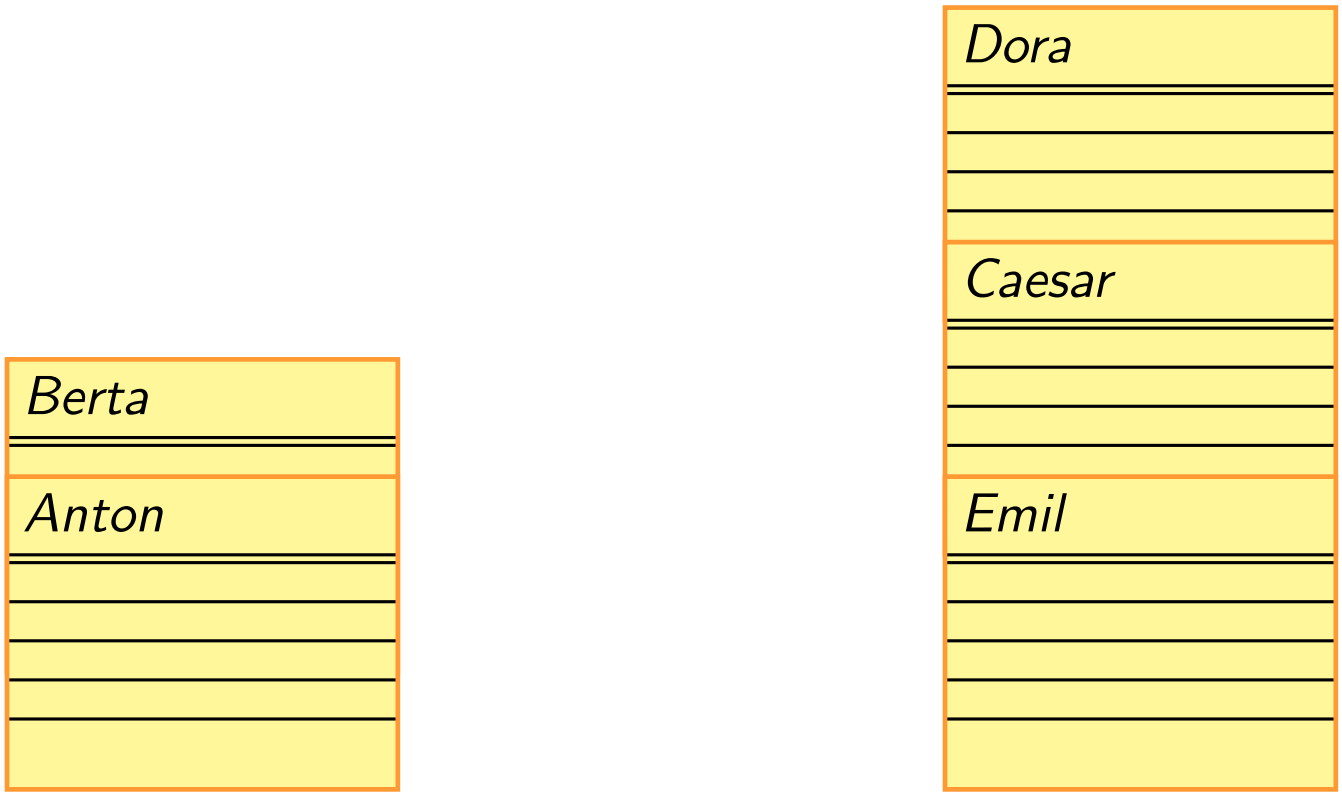
- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Anton*

Selection Sort



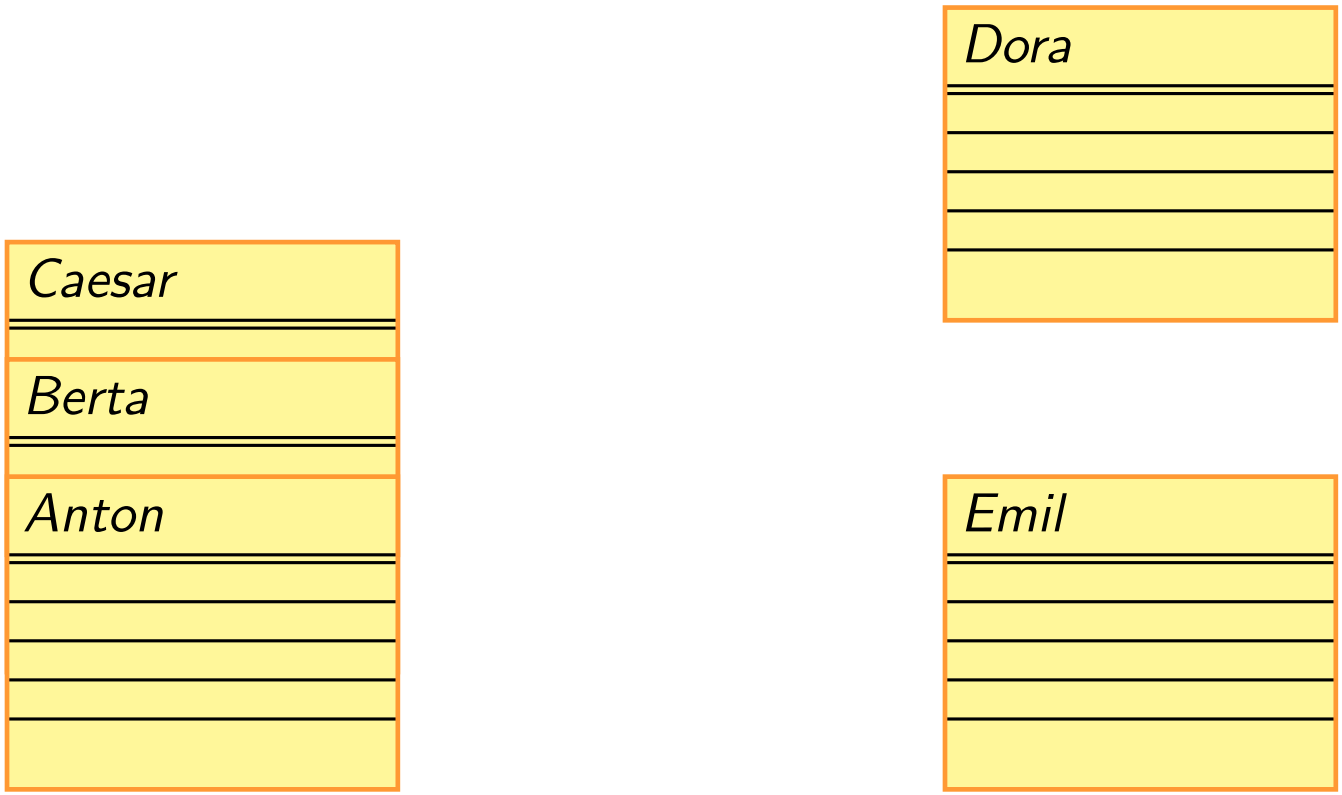
- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Berta*

Selection Sort



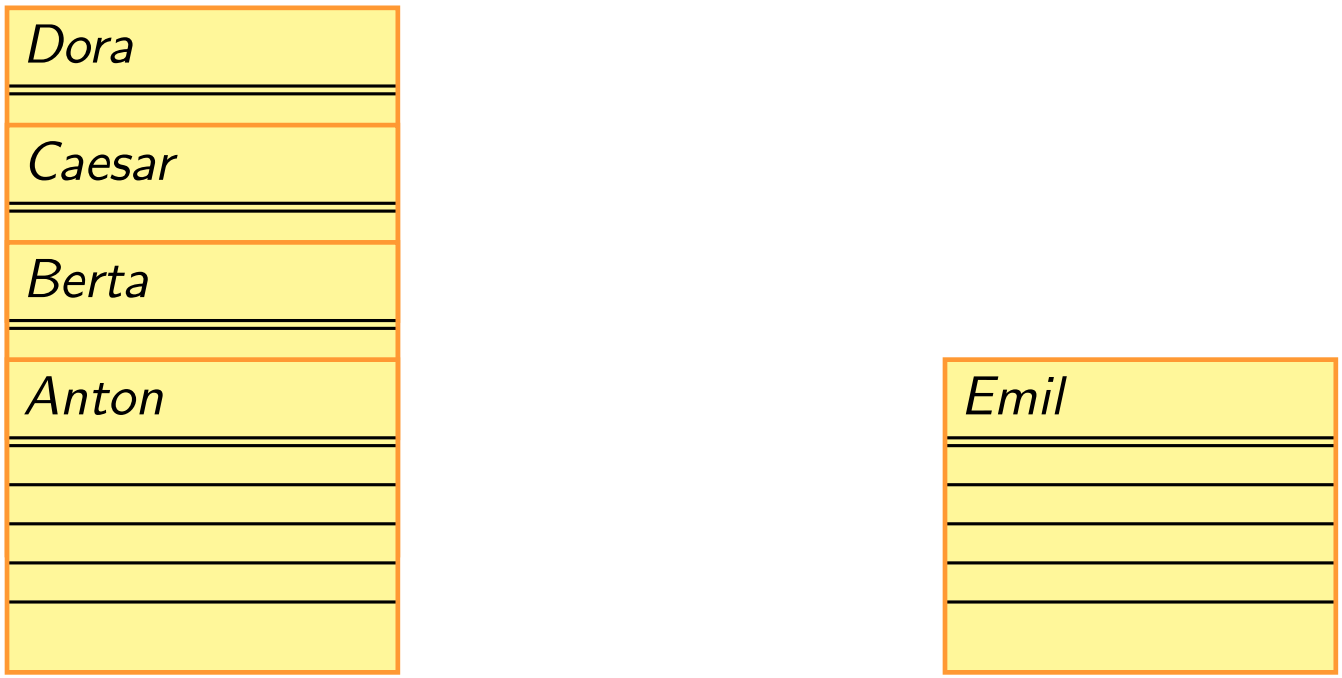
- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Caesar*

Selection Sort



- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Dora*

Selection Sort

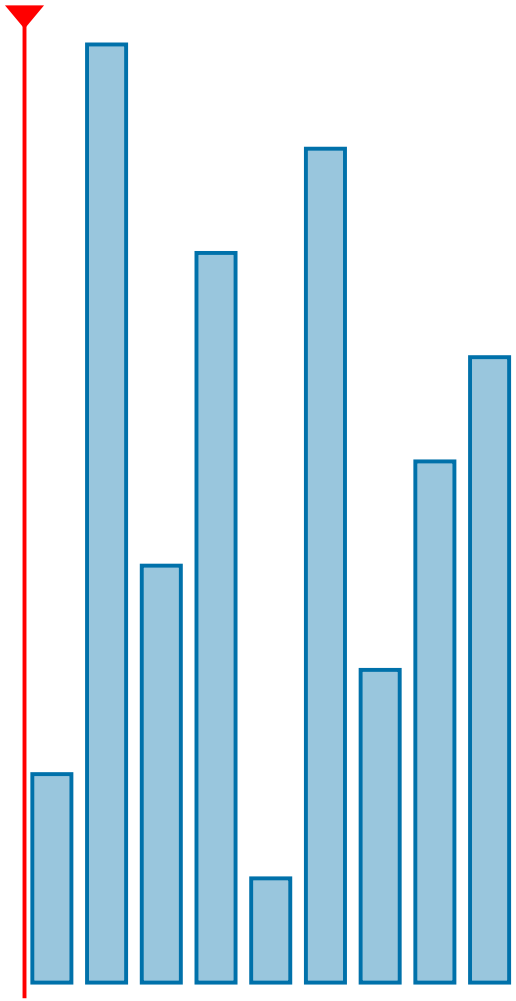


- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Emil*

Selection Sort

- Gustav hat jetzt alle seine Karteikarten geordnet, indem er stets die „kleinste“ Karte *angehängt* hat

Selection Sort



$j = 0$: Anfangszustand

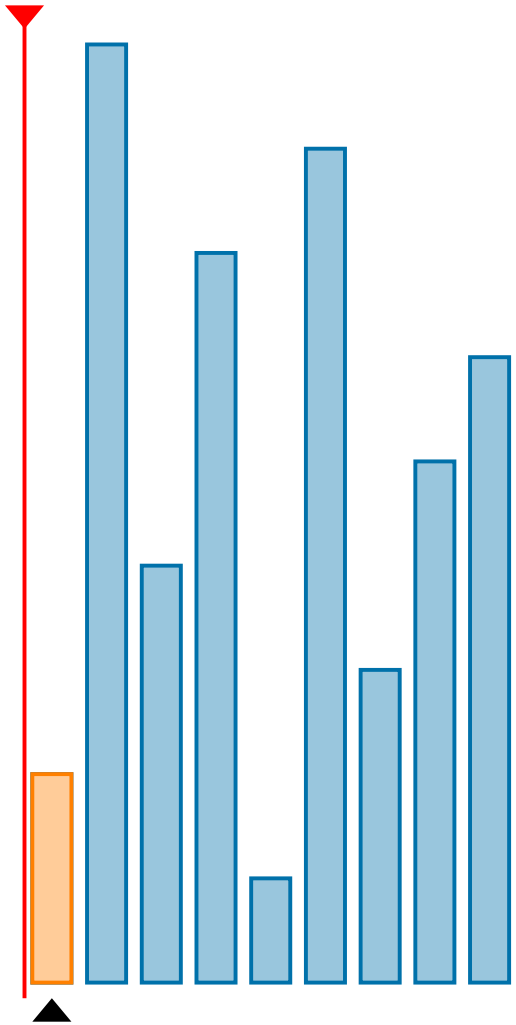
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

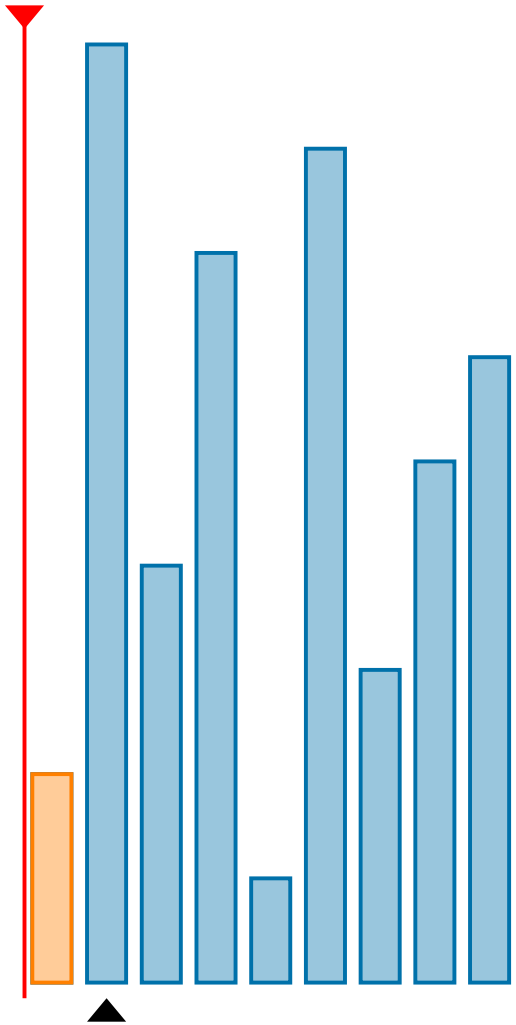


$j = 0$: Suche bei 0

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

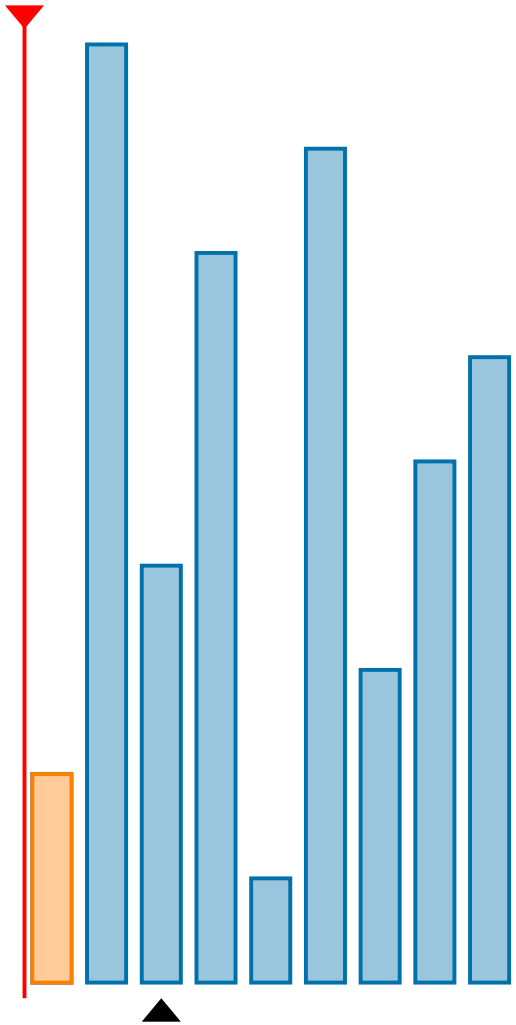


$j = 0$: Suche bei 1

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

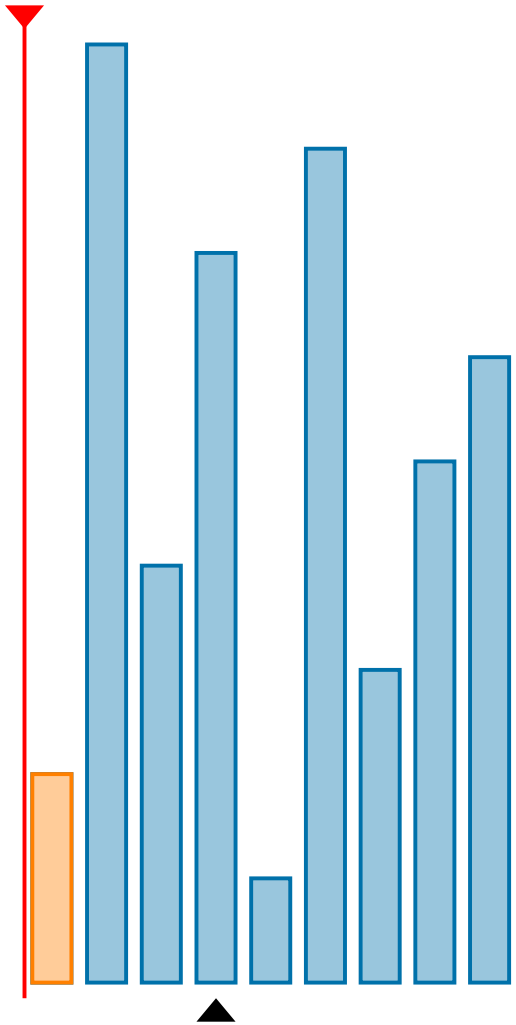


$j = 0$: Suche bei 2

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

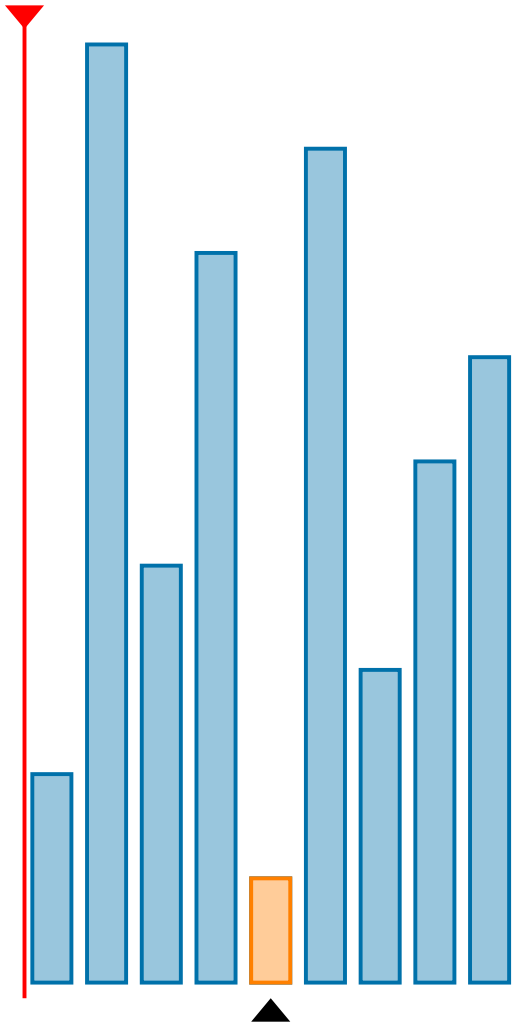


$j = 0$: Suche bei 3

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 0$: Suche bei 4

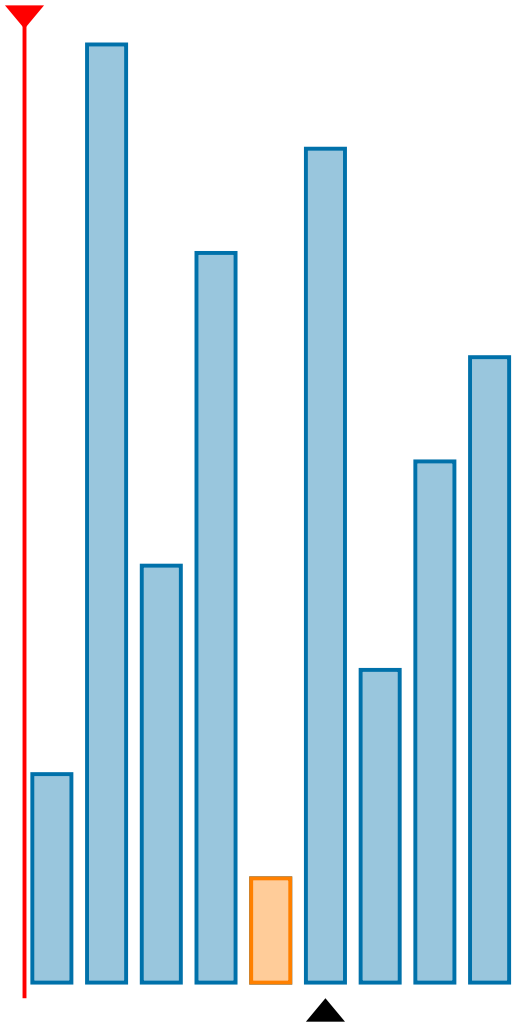
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

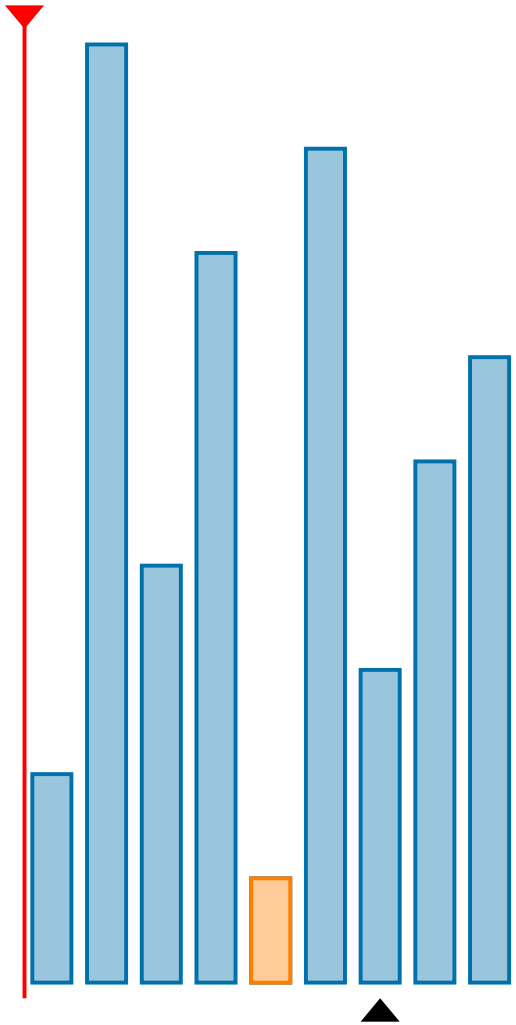


$j = 0$: Suche bei 5

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

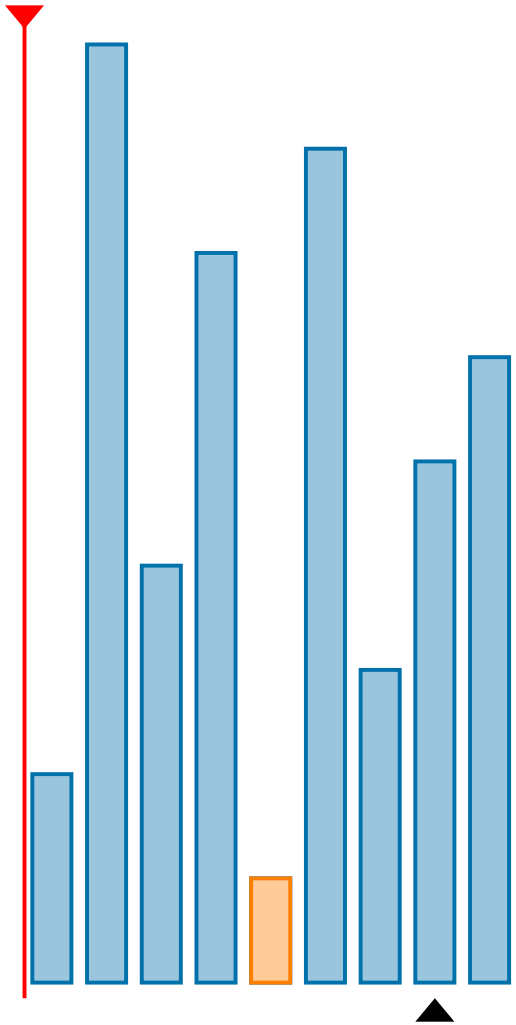


$j = 0$: Suche bei 6

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

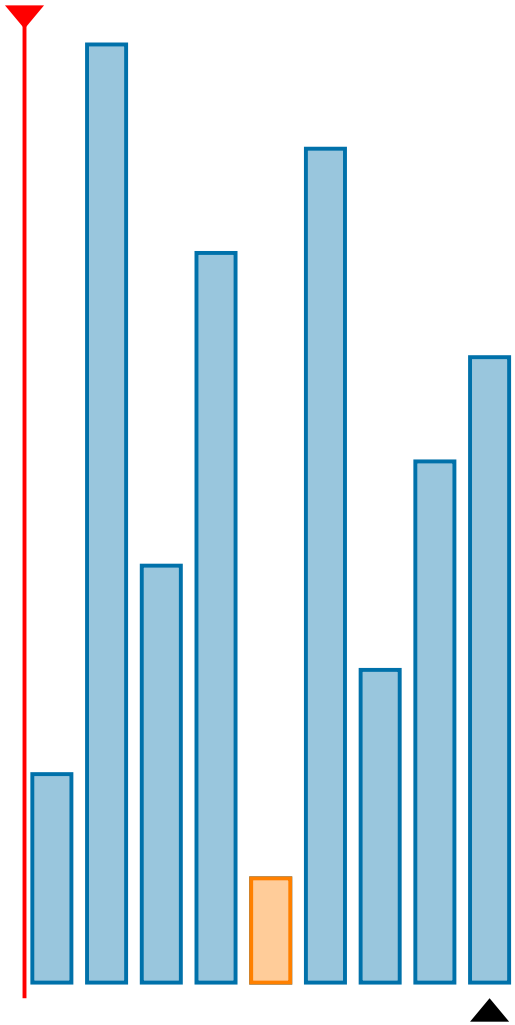


$j = 0$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

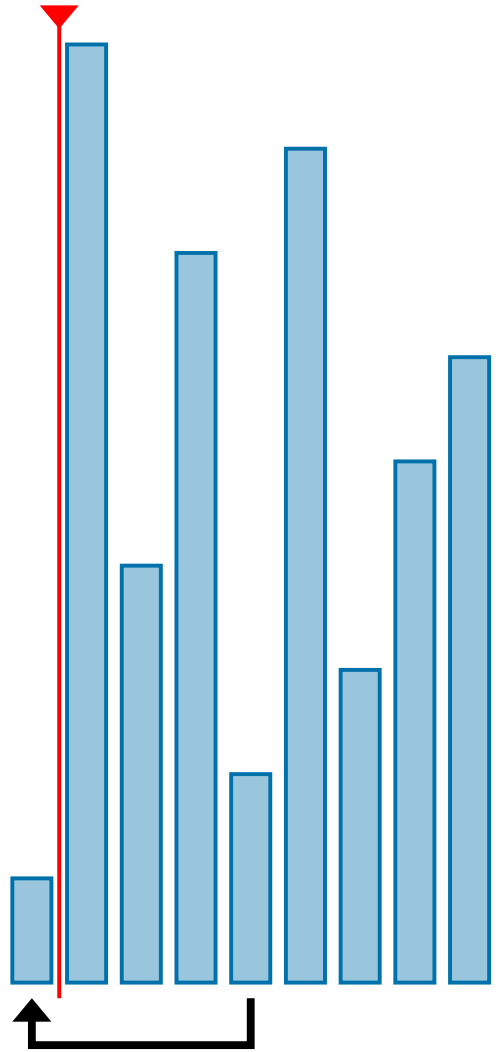


$j = 0$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

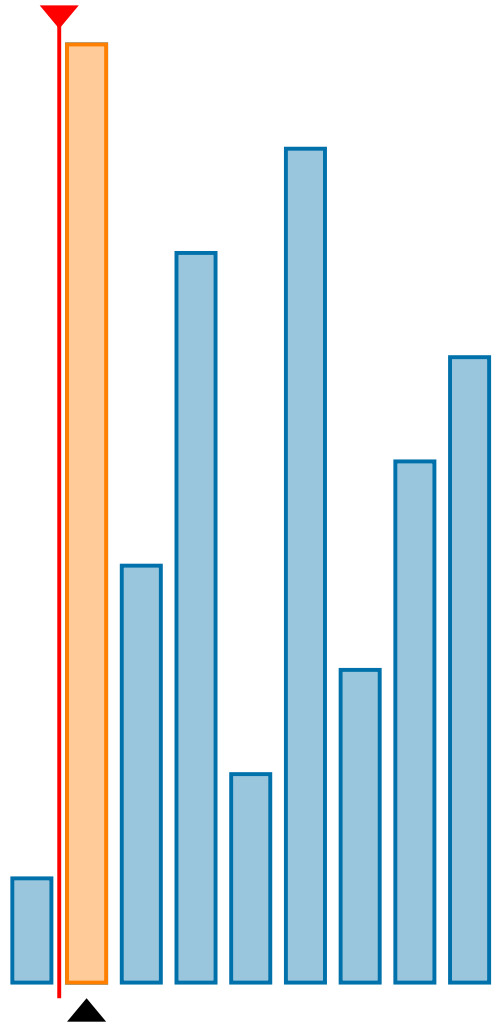


$j = 1$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

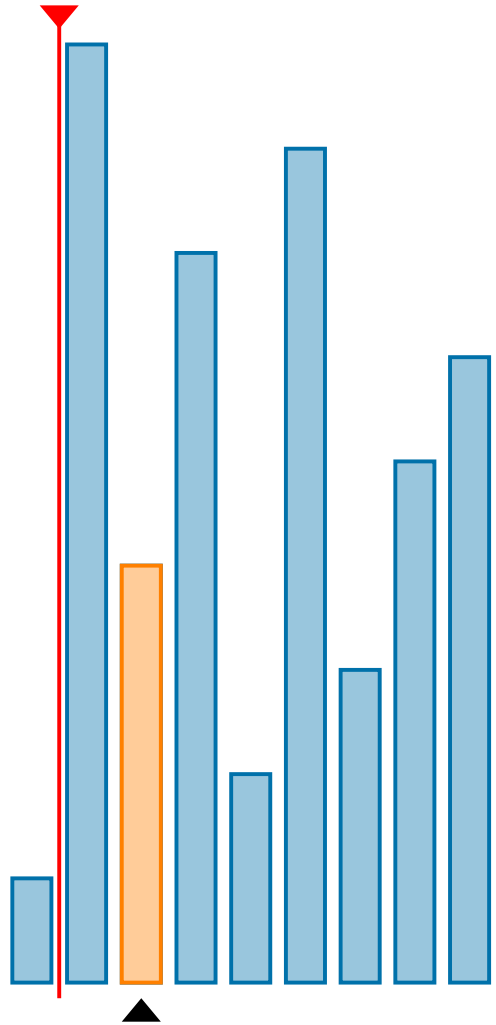


$j = 1$: Suche bei 1

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

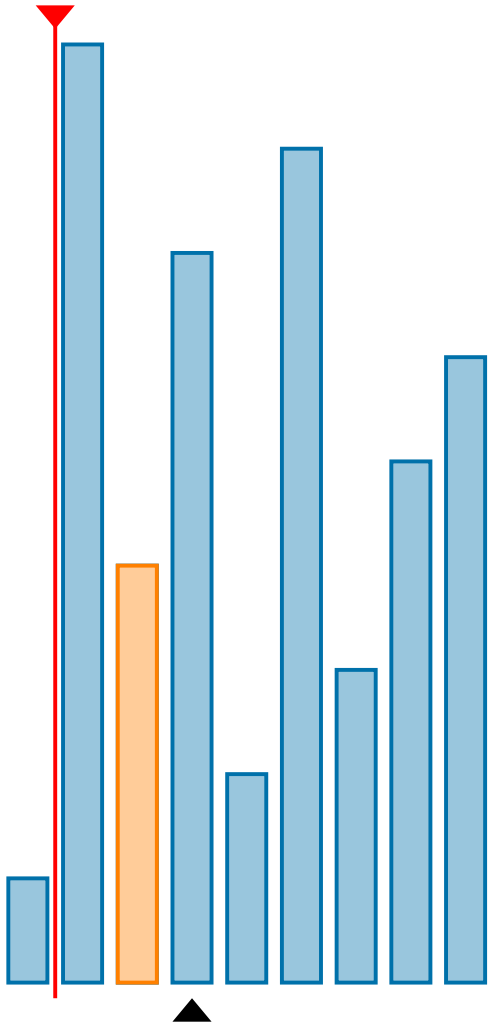


$j = 1$: Suche bei 2

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

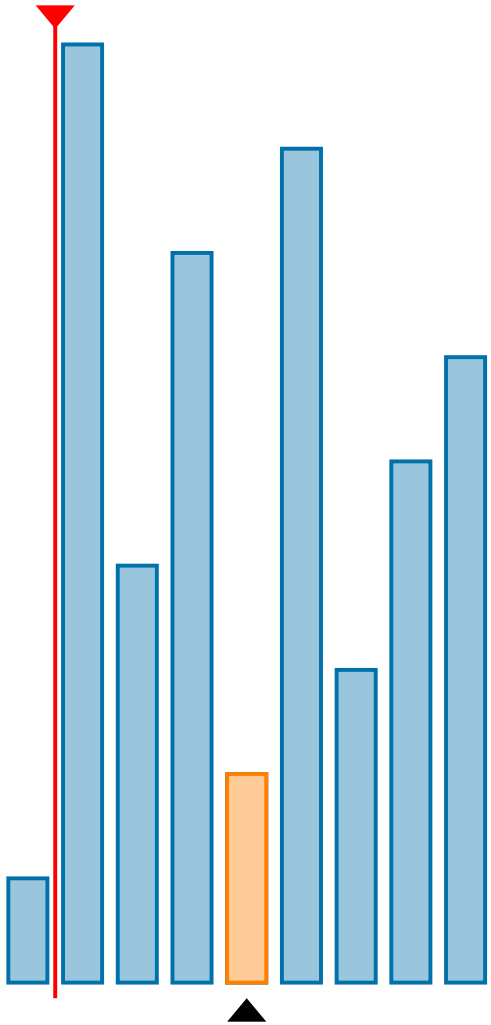


$j = 1$: Suche bei 3

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

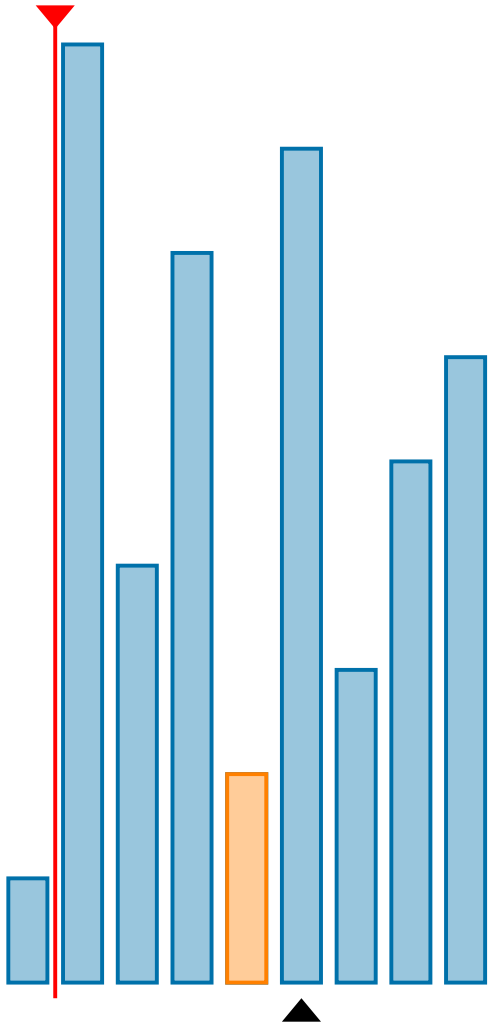


$j = 1$: Suche bei 4

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 1$: Suche bei 5

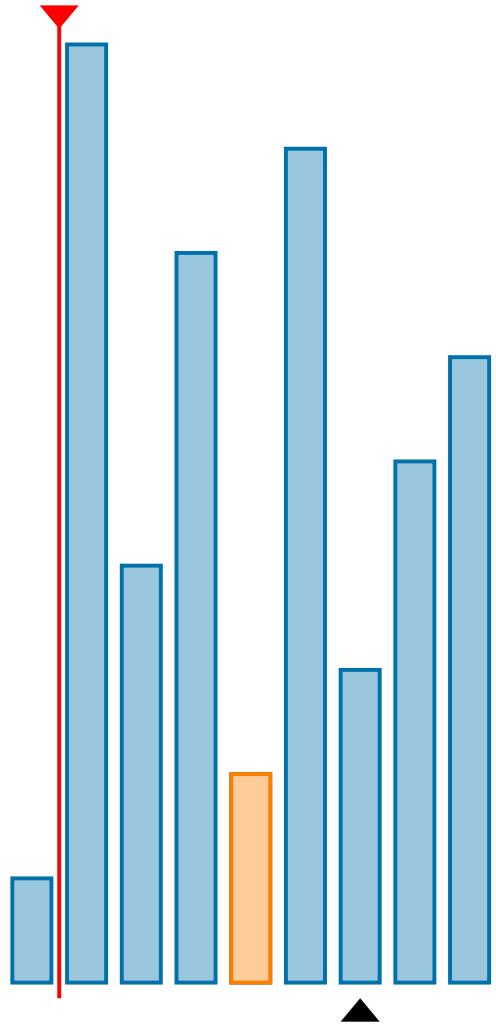
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

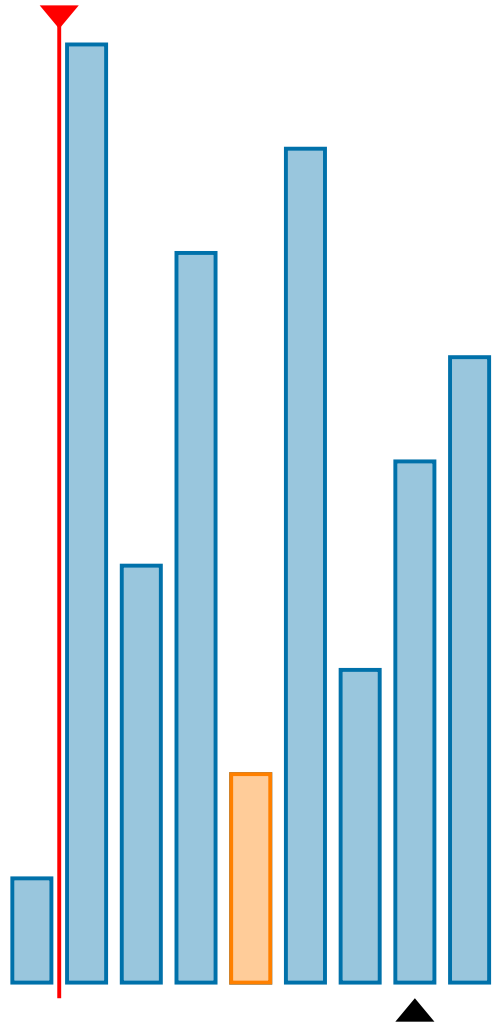


$j = 1$: Suche bei 6

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

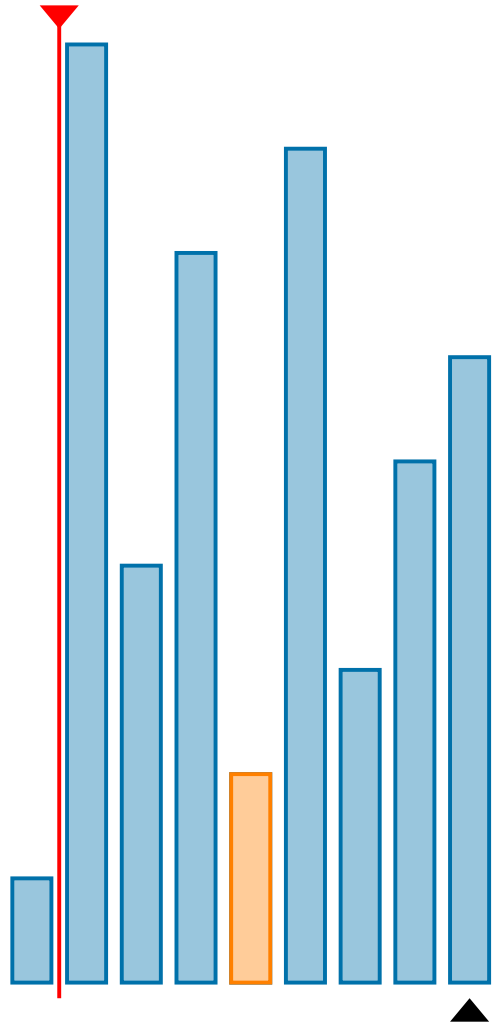


$j = 1$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

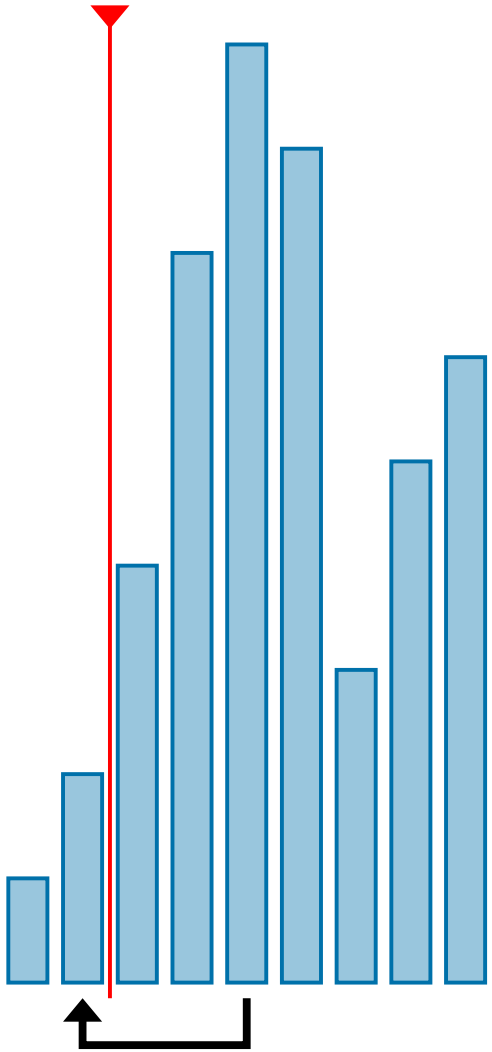


$j = 1$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

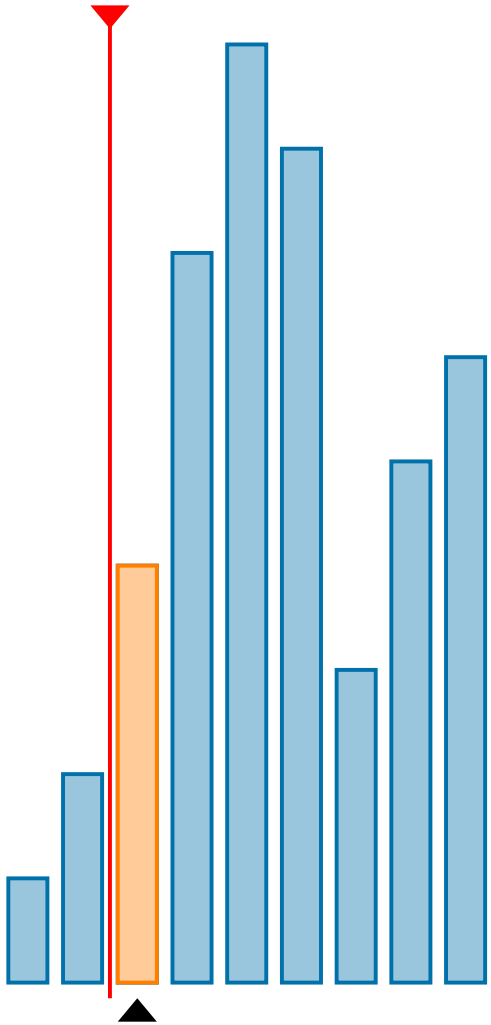


$j = 2$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 2$: Suche bei 2

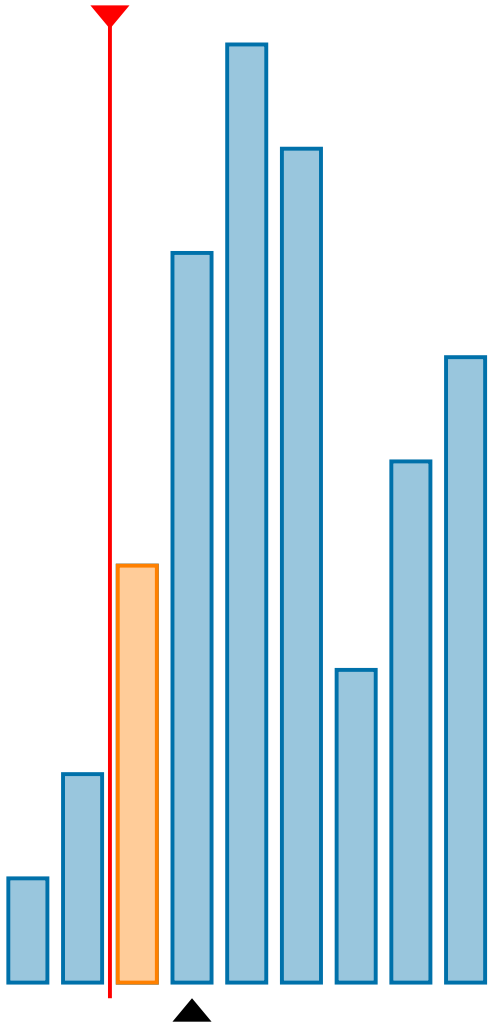
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

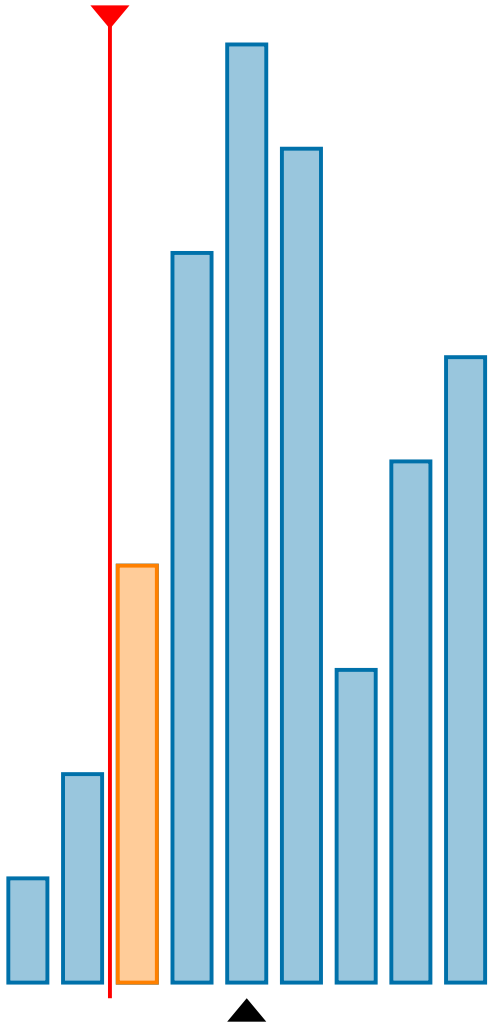


$j = 2$: Suche bei 3

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

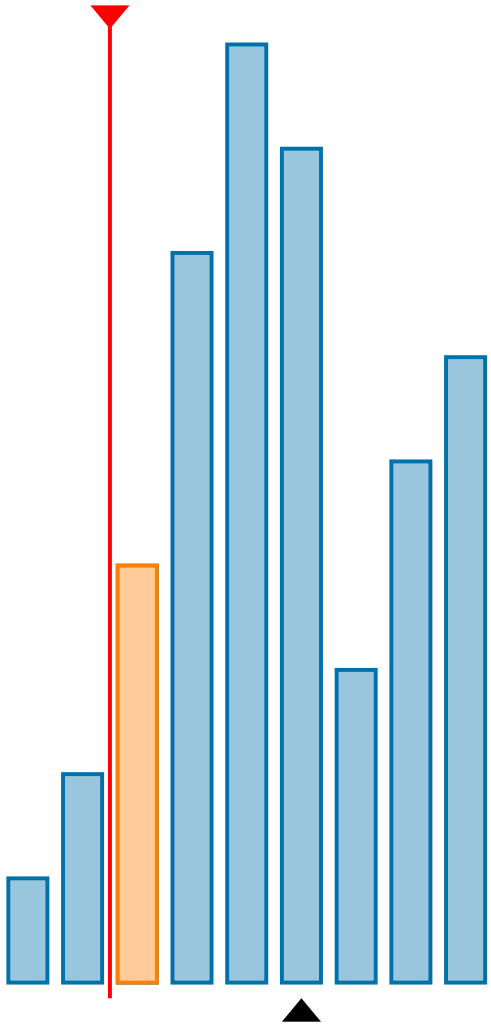


$j = 2$: Suche bei 4

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

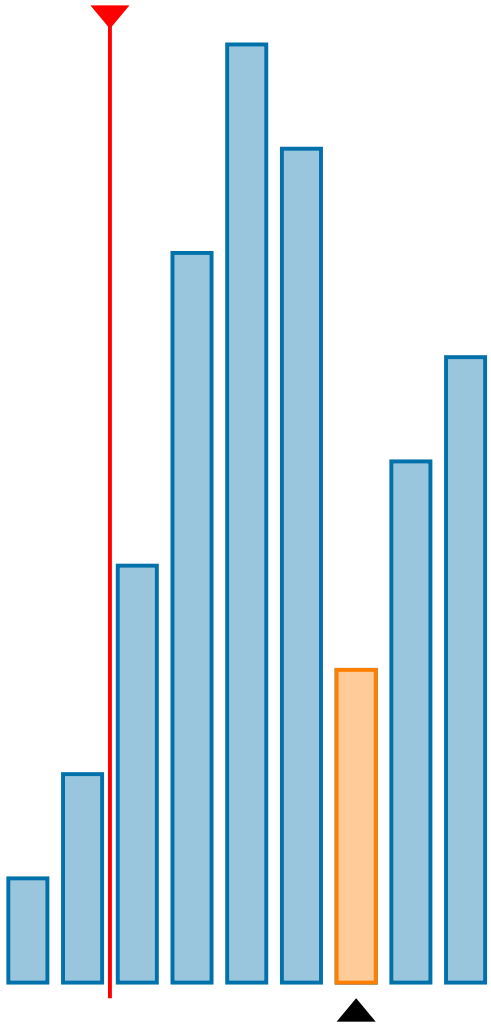


$j = 2$: Suche bei 5

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

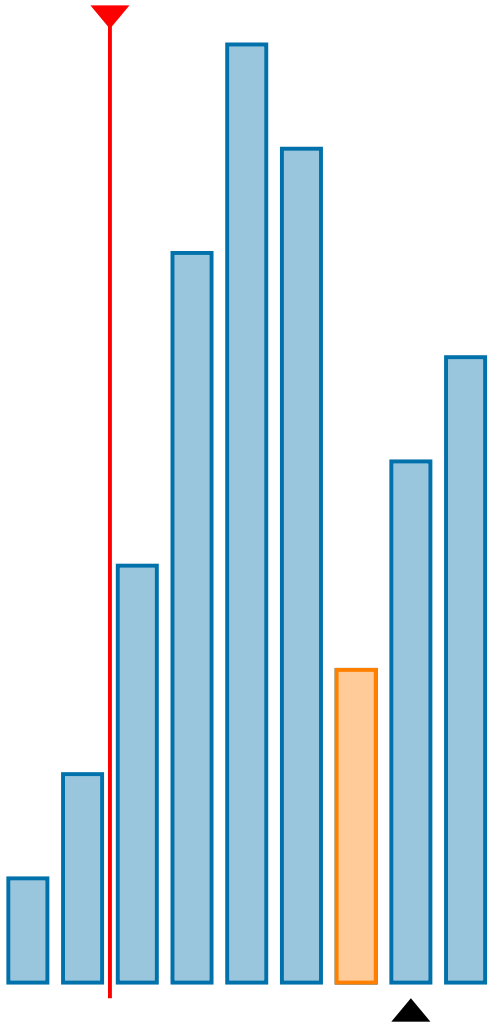


$j = 2$: Suche bei 6

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

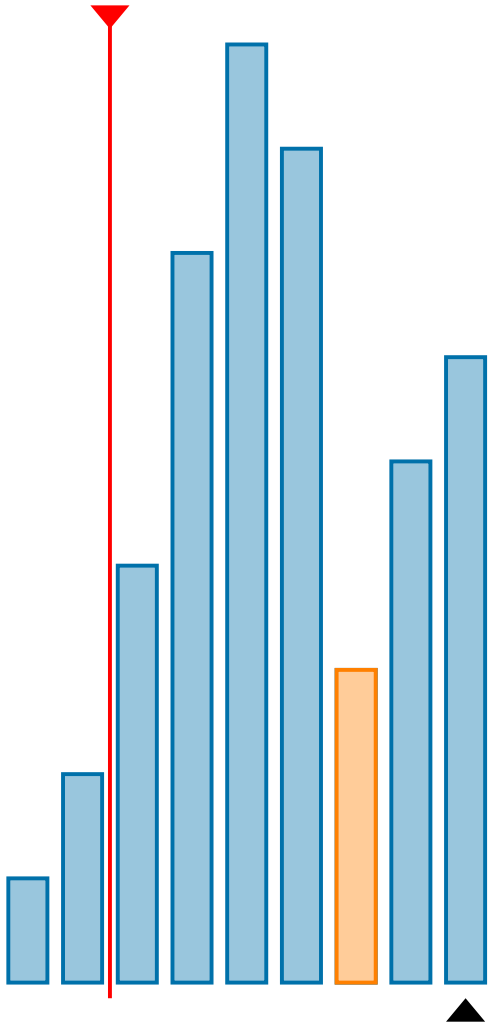


$j = 2$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

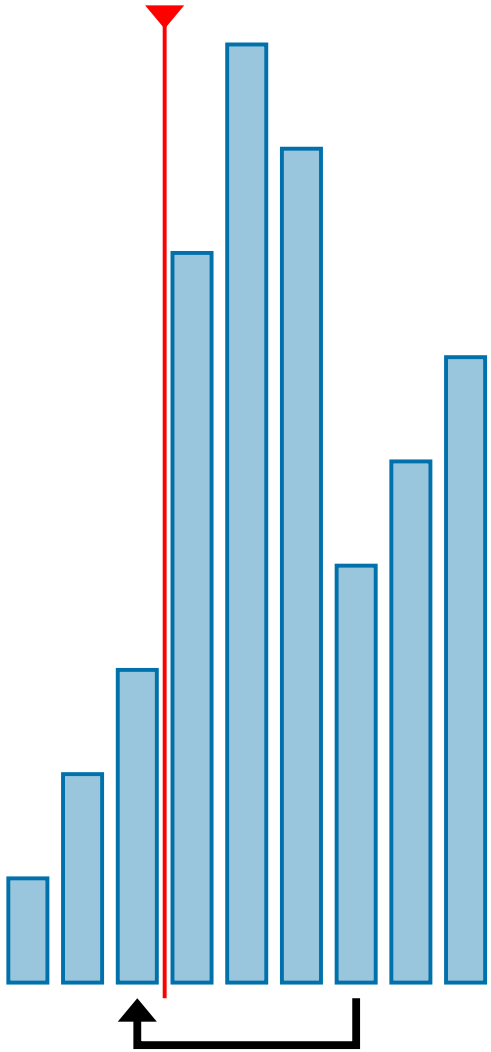


$j = 2$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 3$: Anfangszustand

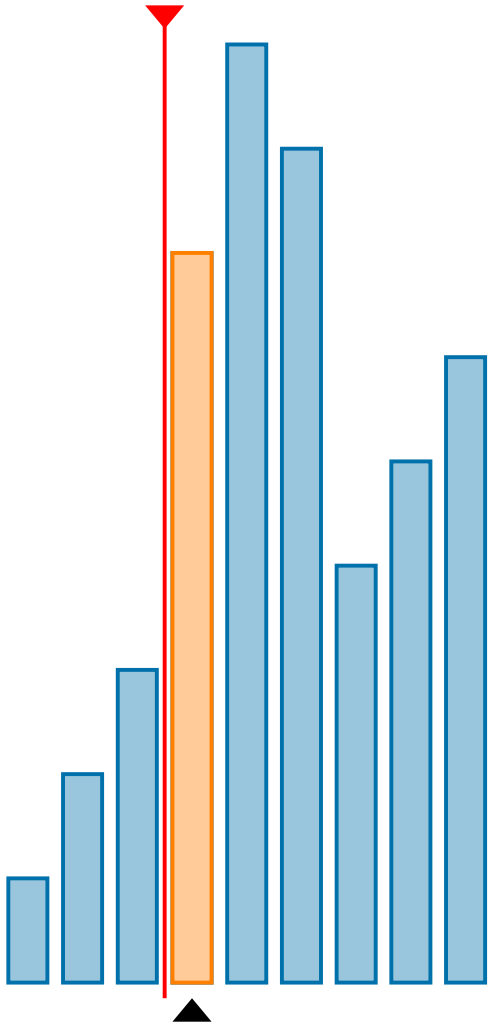
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

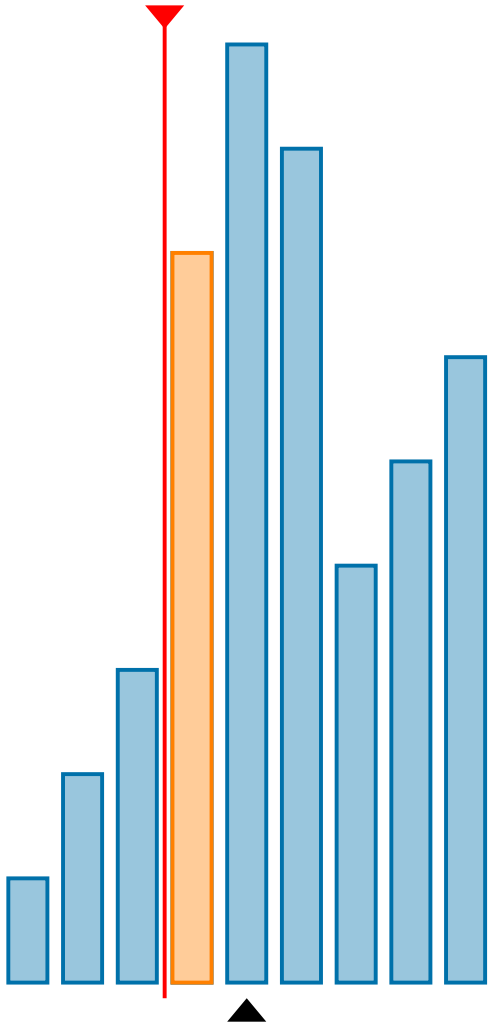


$j = 3$: Suche bei 3

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

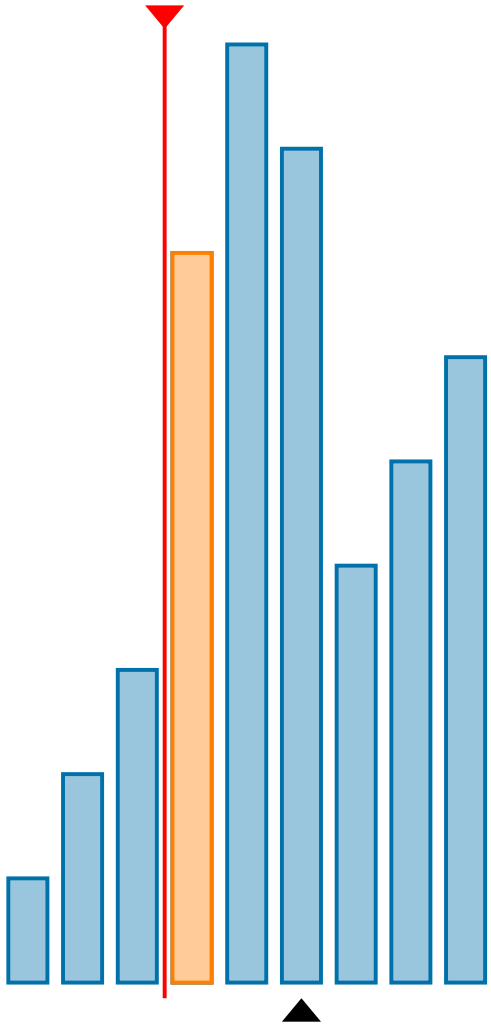


$j = 3$: Suche bei 4

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

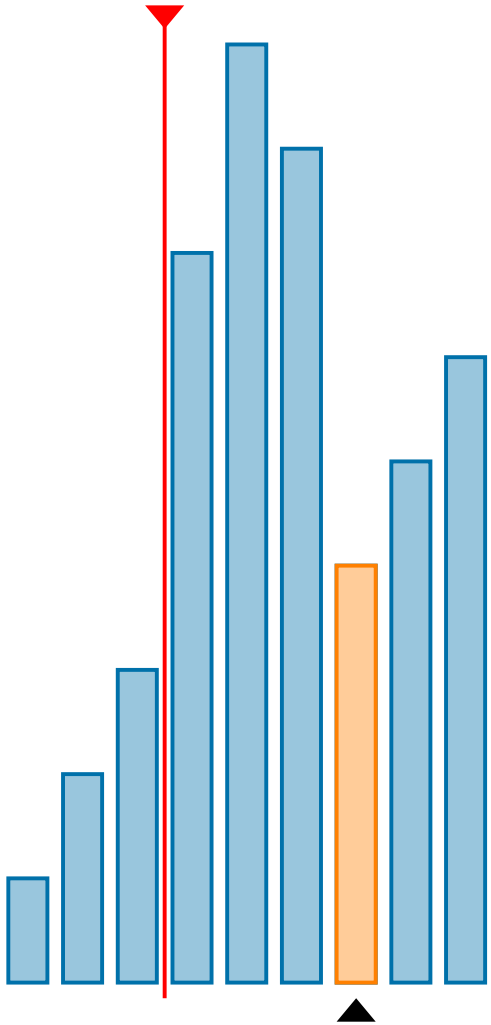


$j = 3$: Suche bei 5

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 3$: Suche bei 6

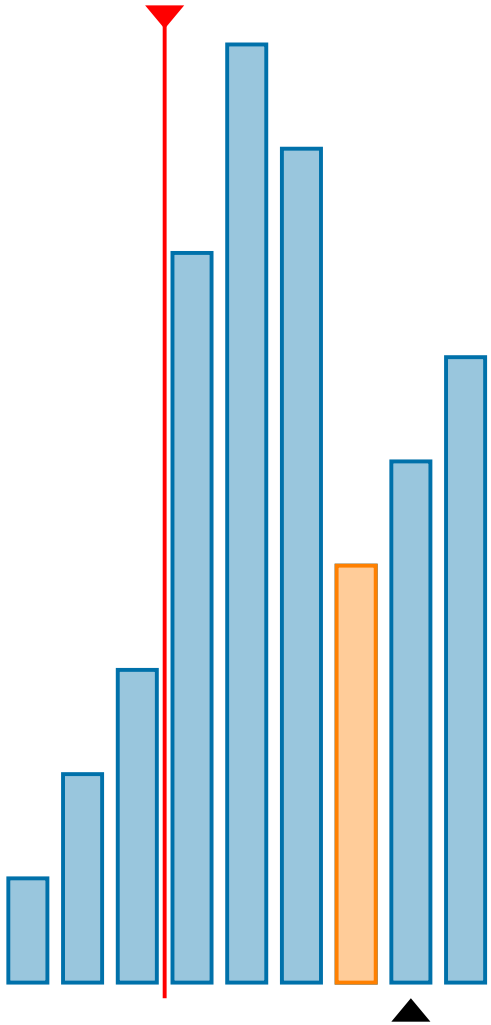
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 3$: Suche bei 7

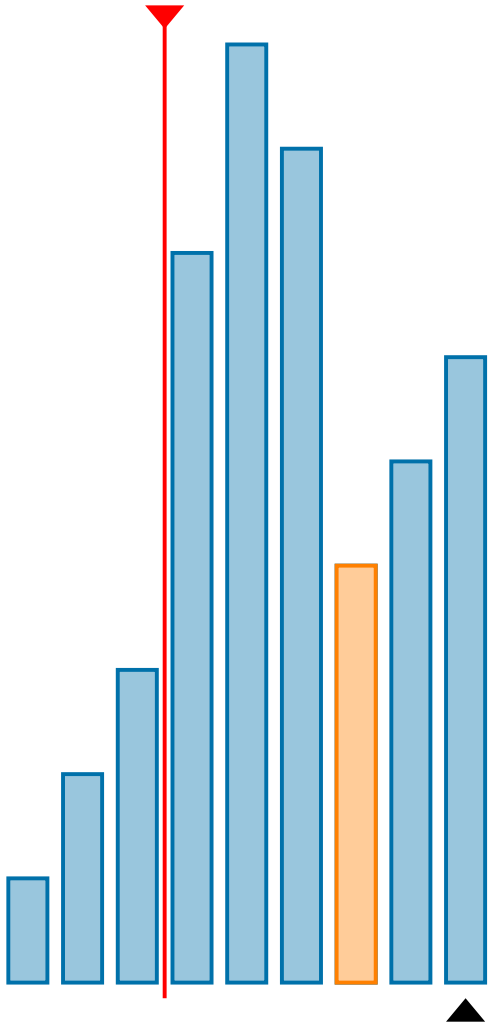
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

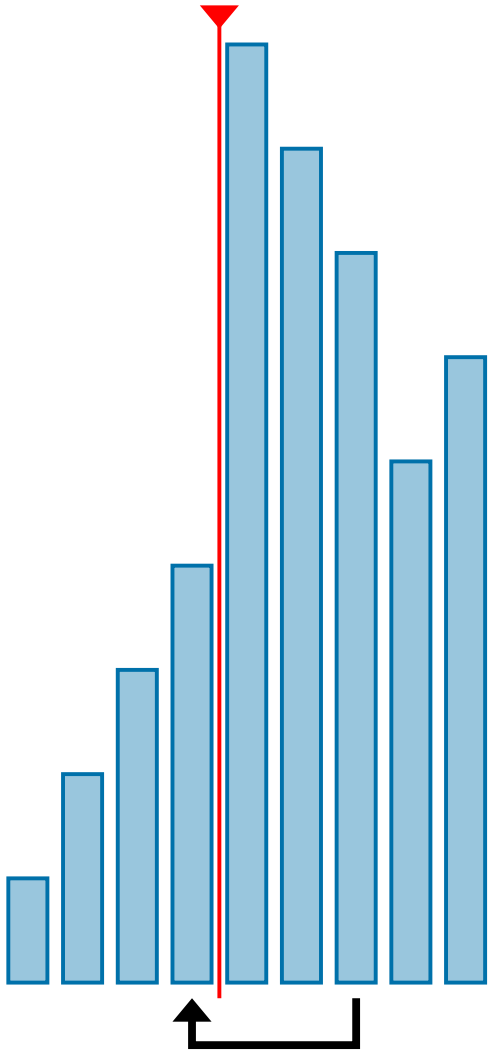


$j = 3$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 4$: Anfangszustand

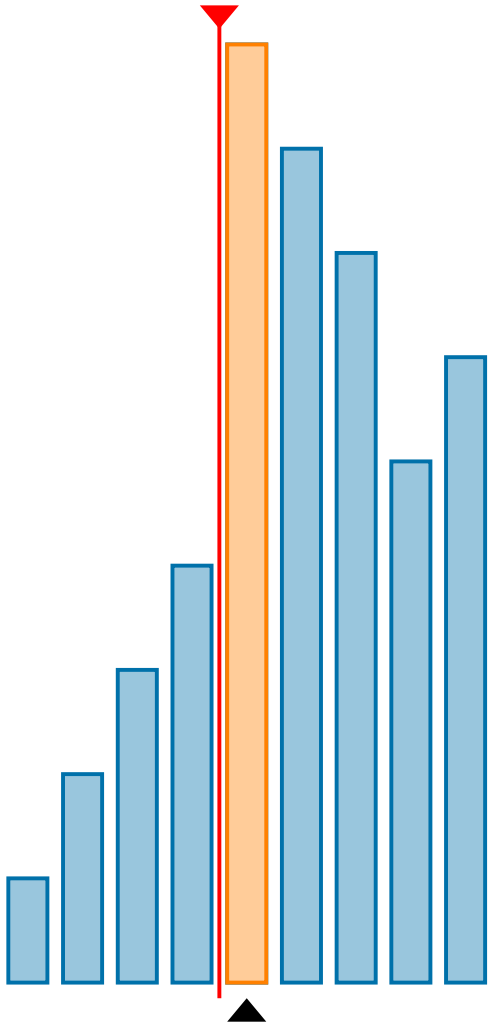
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

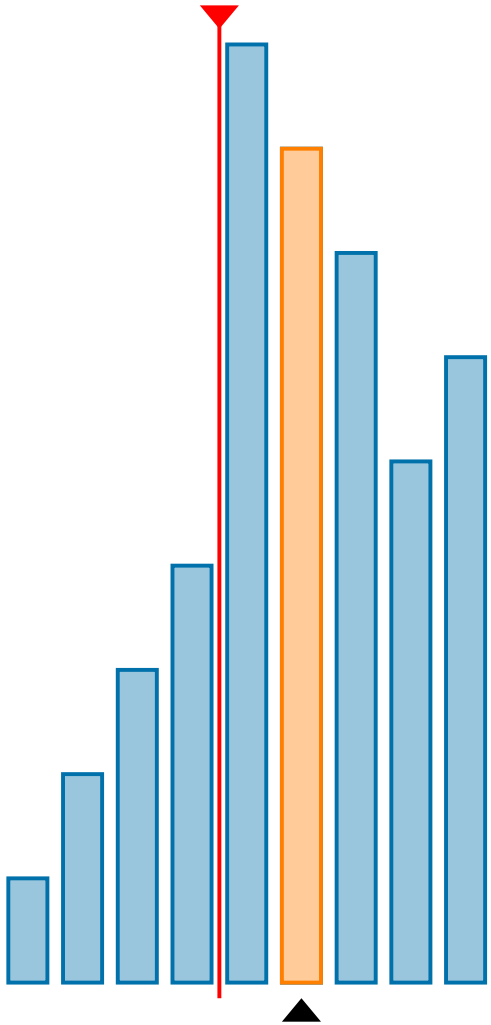


$j = 4$: Suche bei 4

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

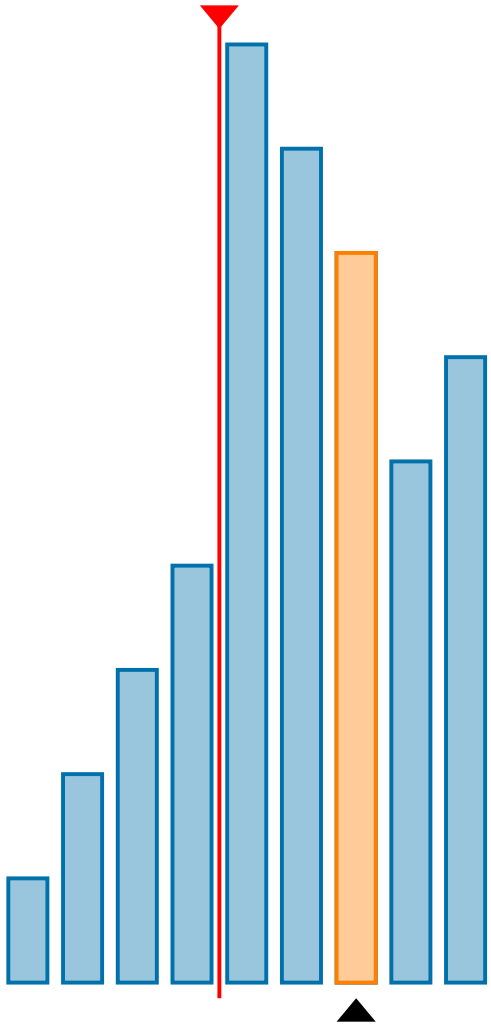


$j = 4$: Suche bei 5

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

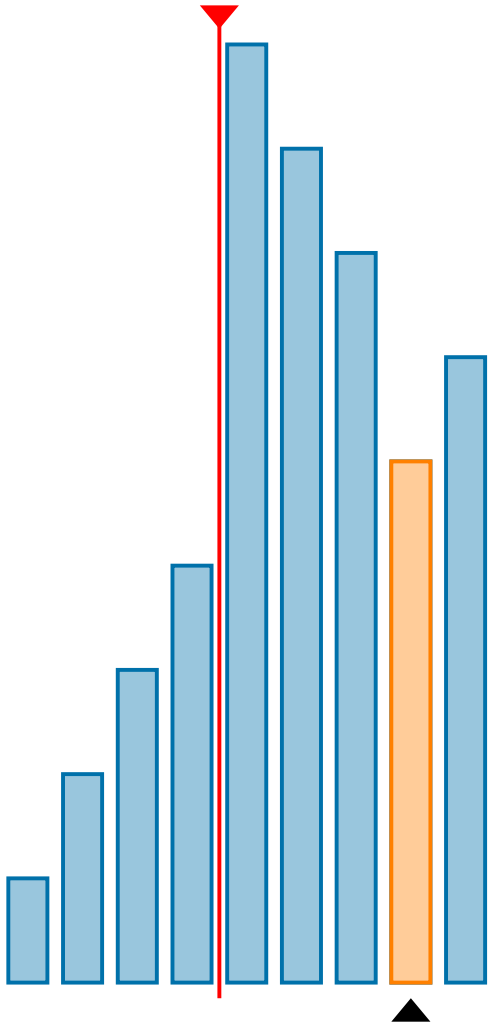


$j = 4$: Suche bei 6

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

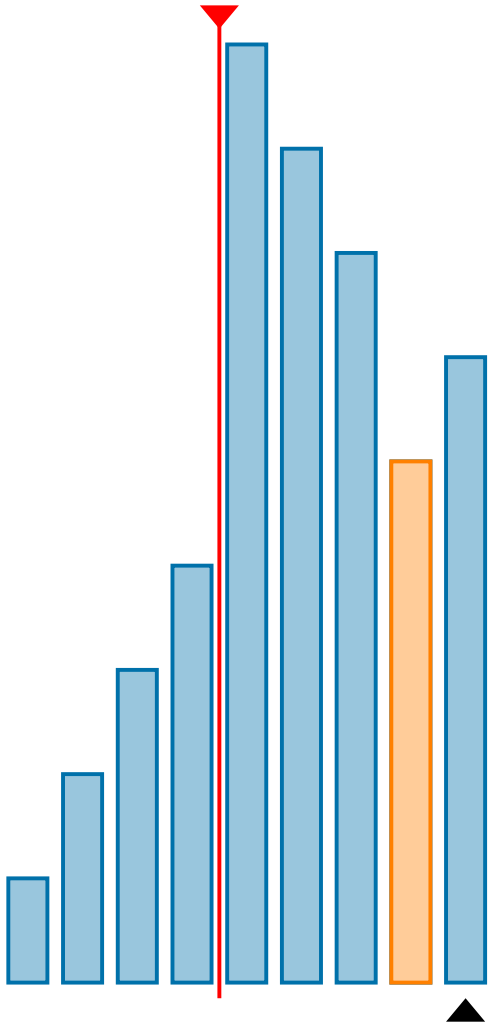


$j = 4$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

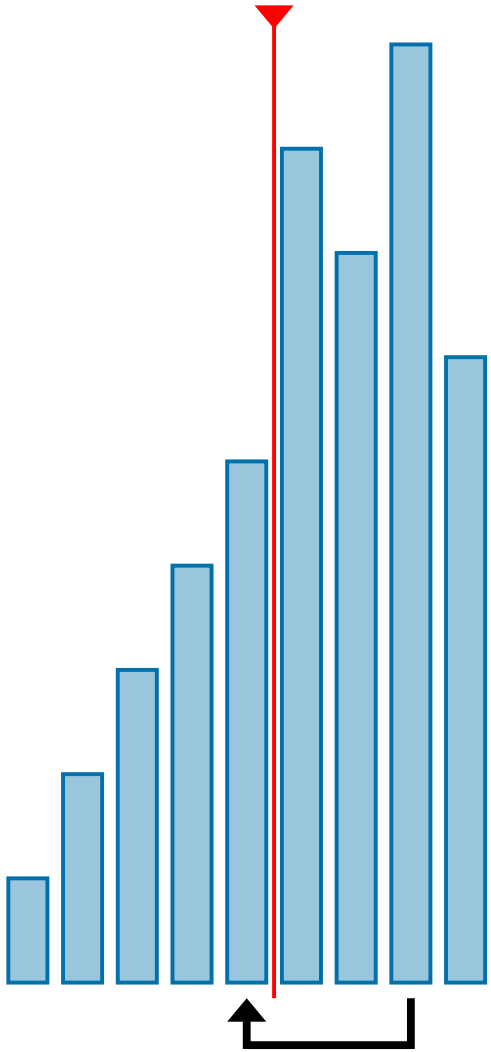


$j = 4$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 5$: Anfangszustand

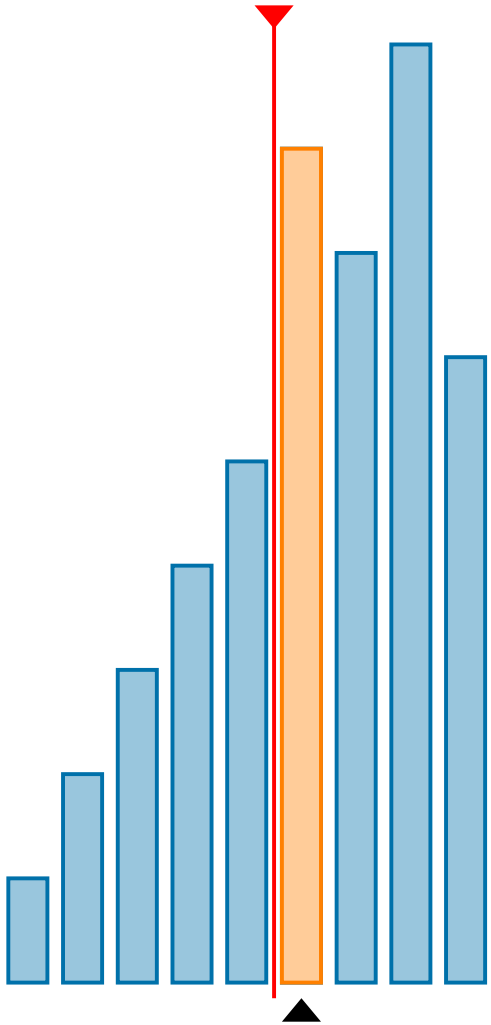
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

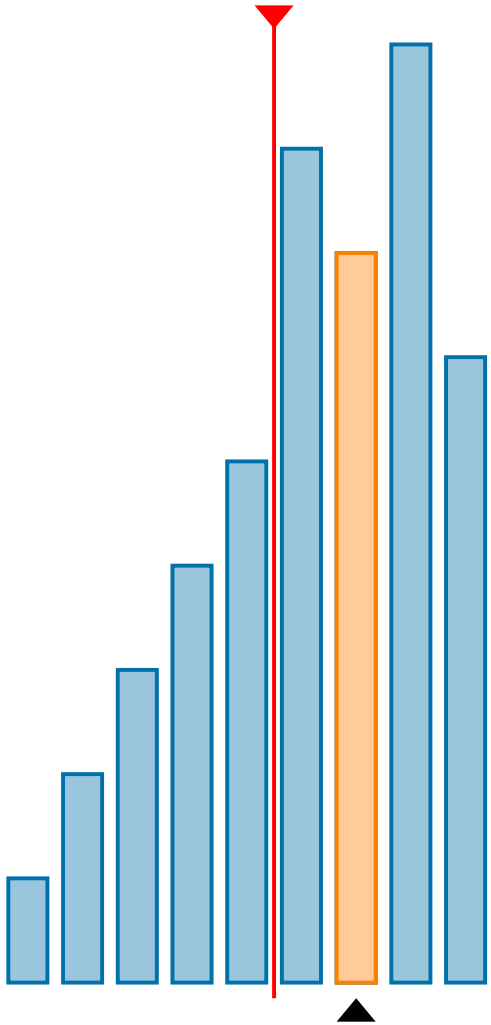


$j = 5$: Suche bei 5

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 5$: Suche bei 6

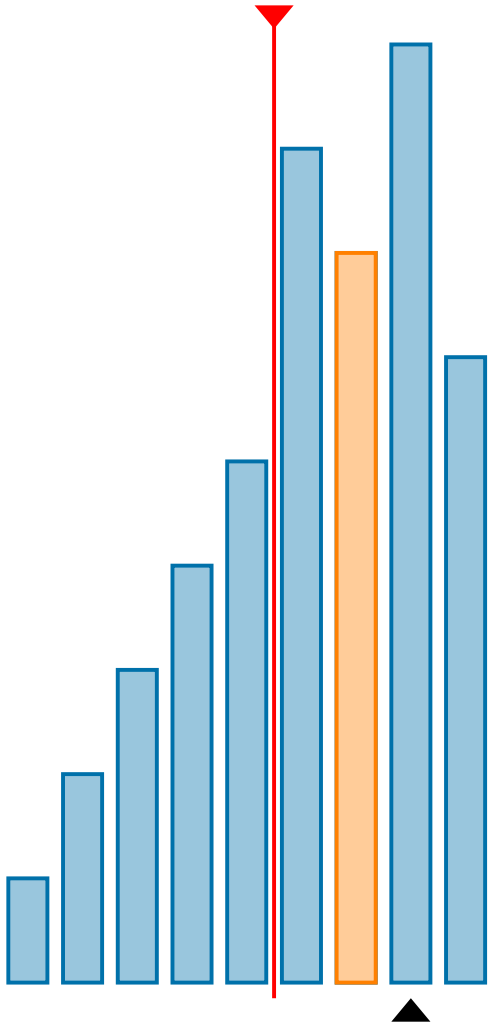
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

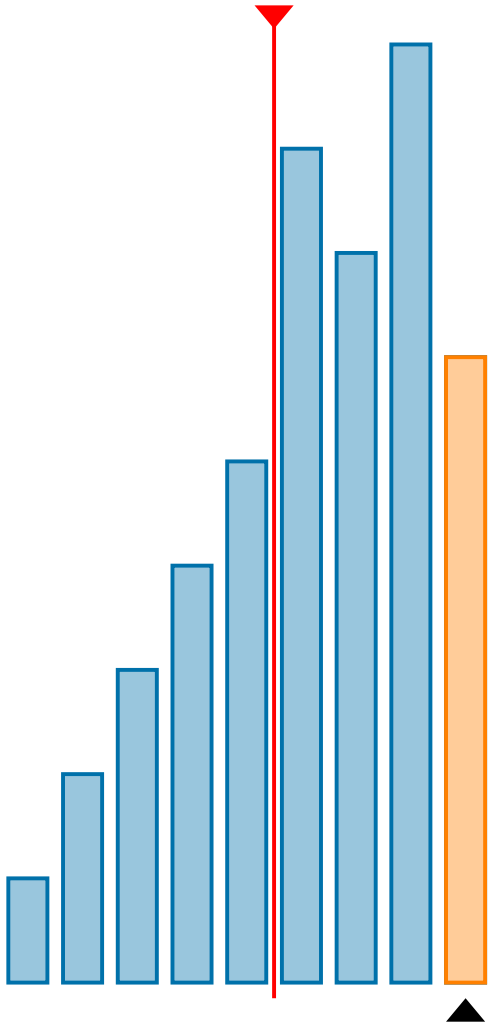


$j = 5$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

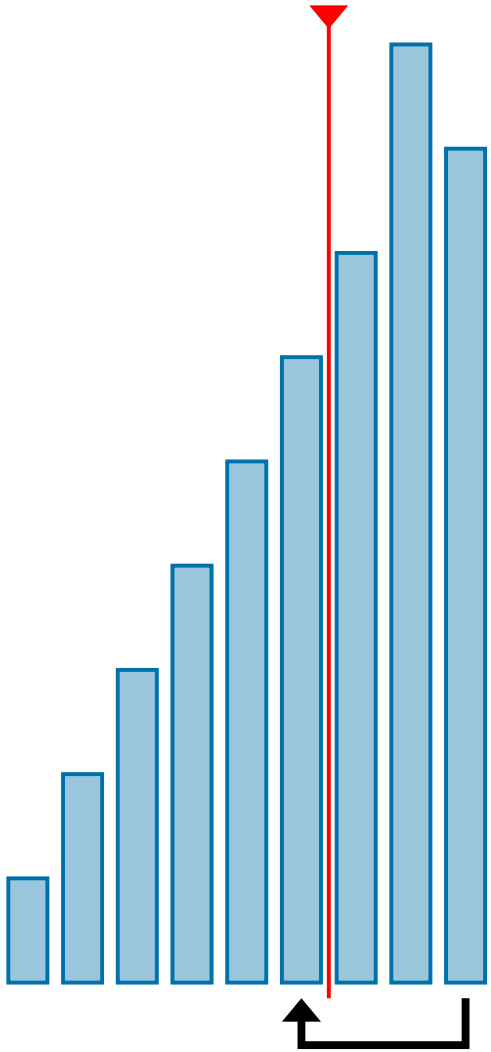


$j = 5$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 6$: Anfangszustand

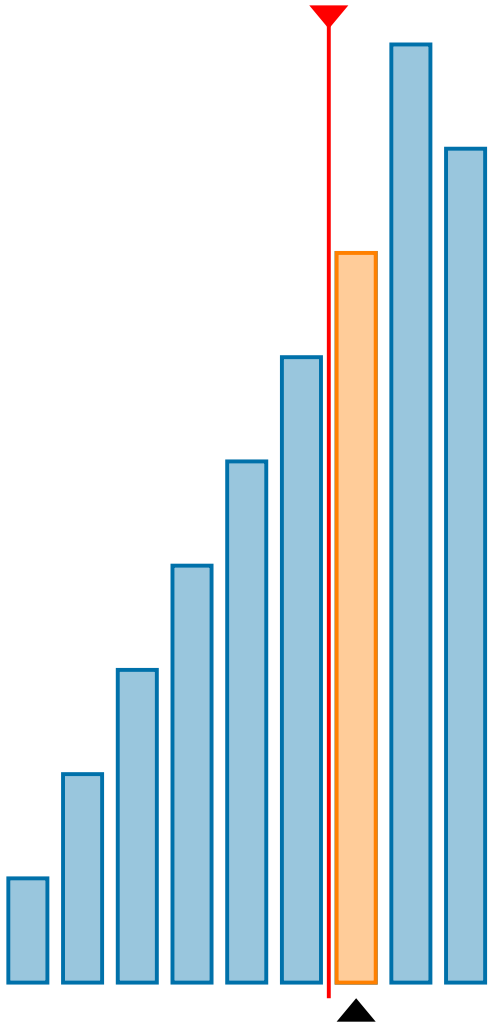
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

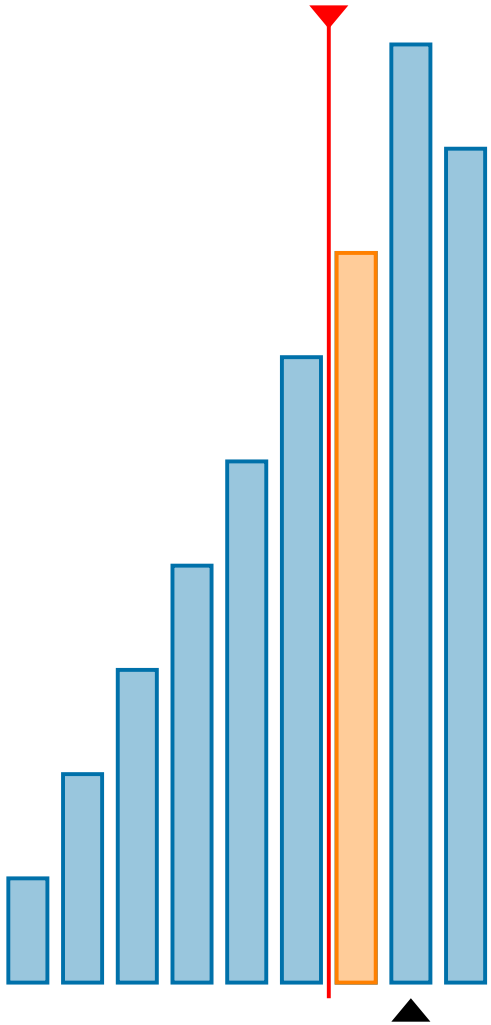


$j = 6$: Suche bei 6

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

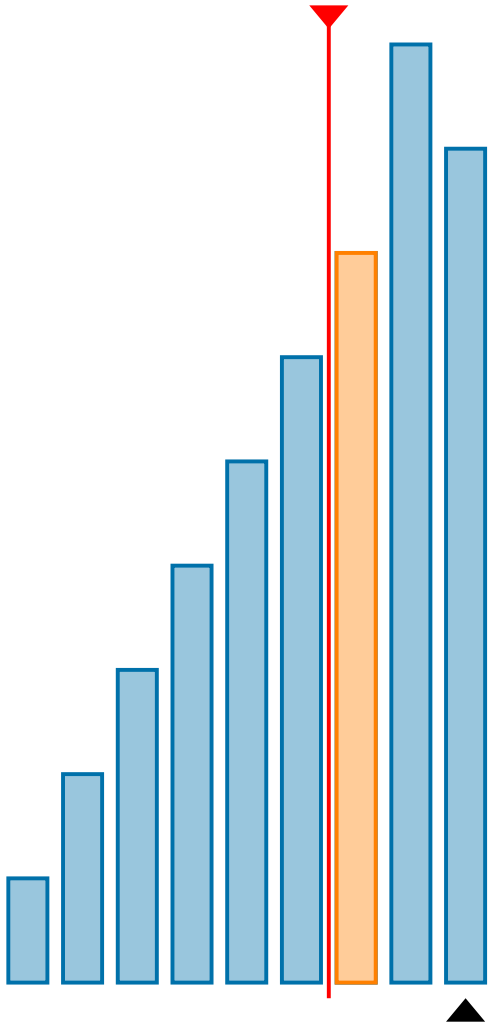


$j = 6$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

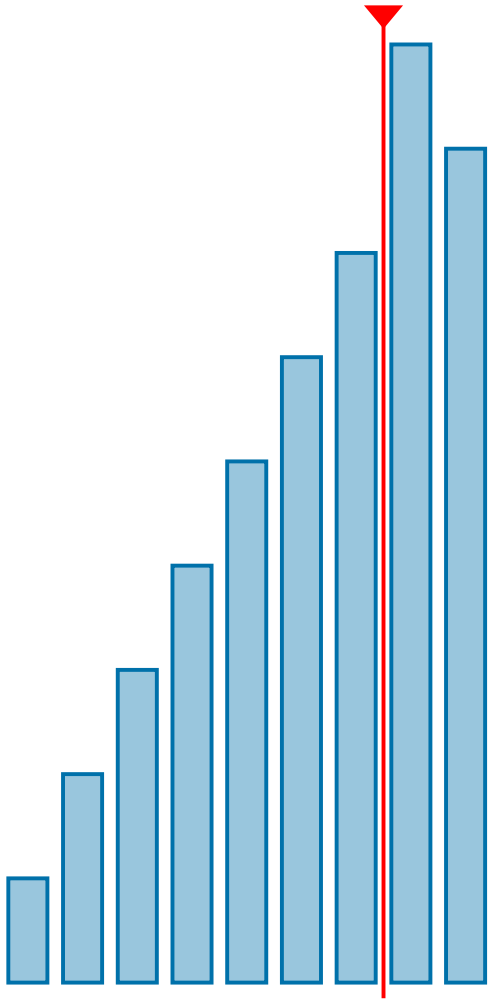


$j = 6$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



$j = 7$: Anfangszustand

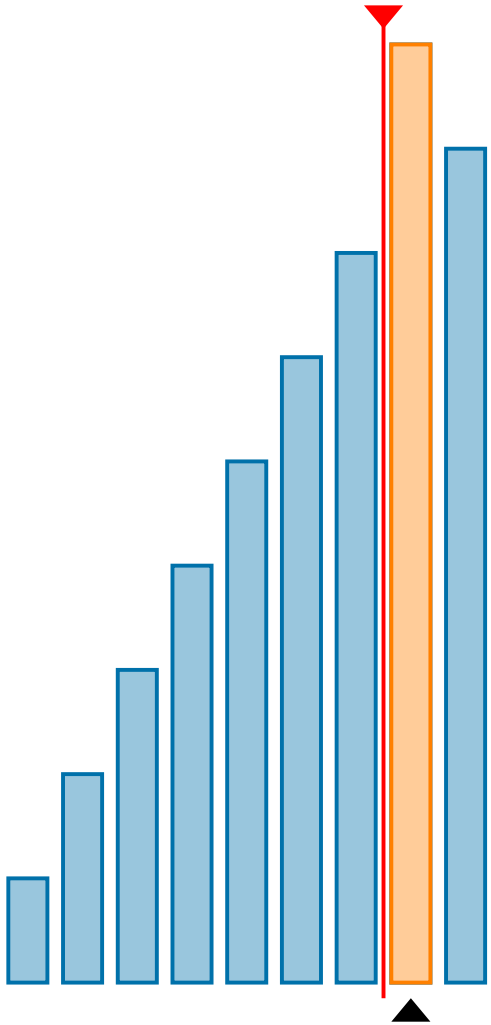
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

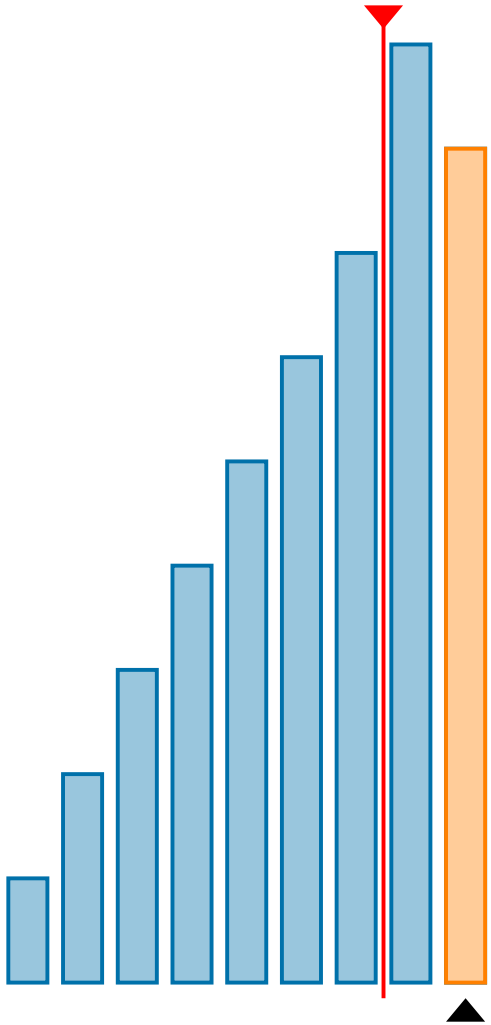


$j = 7$: Suche bei 7

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

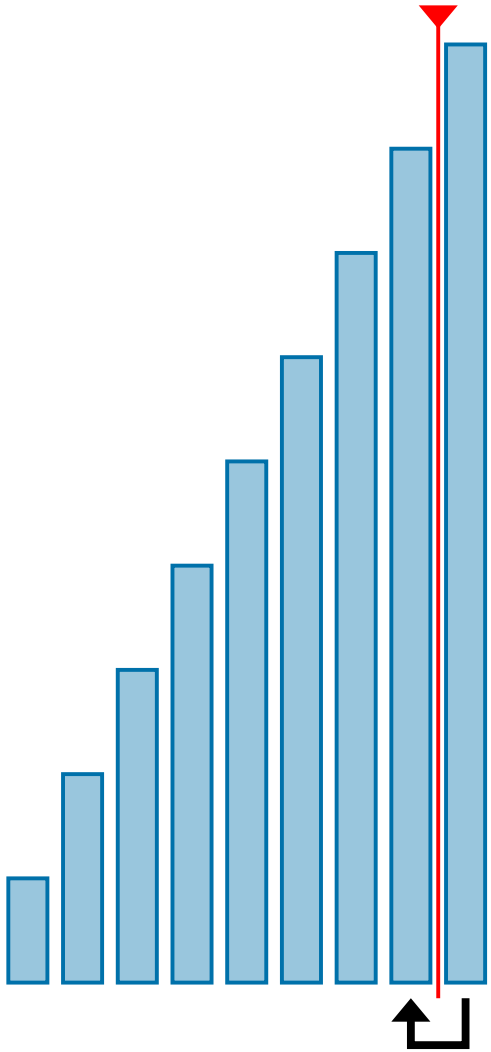


$j = 7$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

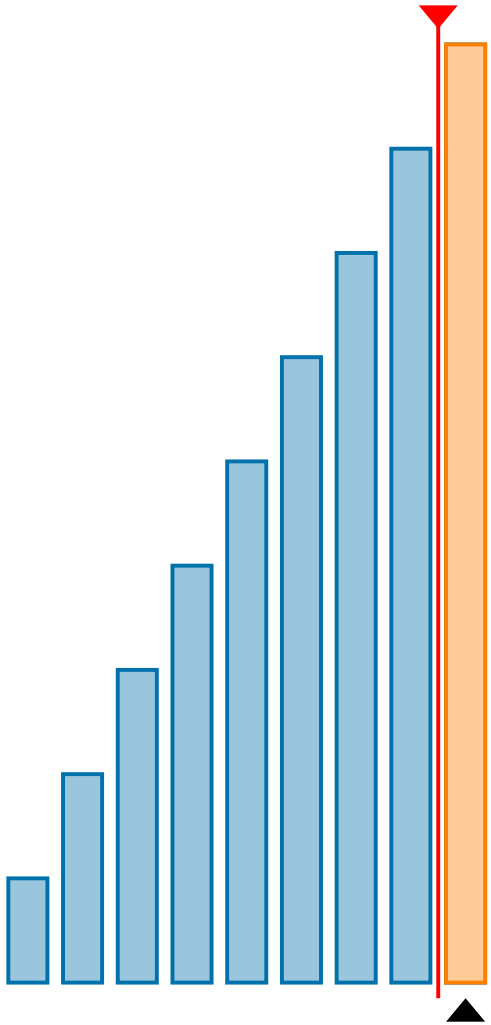


$j = 8$: Anfangszustand

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort

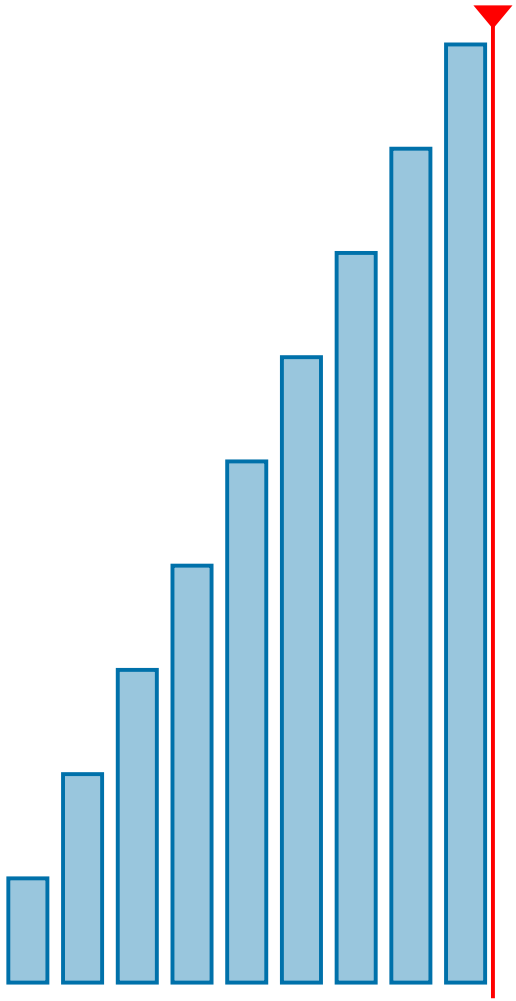


$j = 8$: Suche bei 8

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
       Startindex j
Output: Index i ≥ j des kleinsten Elements in A[j..n−1]
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

```
Input: Feld A[0..n−1] von n ≥ 0 natürlichen Zahlen
Output: Feld A aufsteigend sortiert
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Selection Sort



Endzustand

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Startindex j
Output: Index $i \geq j$ des kleinsten Elements in $A[j..n-1]$

```
IndexOfMin(A, j)
        i = j;
        Ai = A[j];
        for k = j + 1 to n − 1 {
                if (A[k] < Ai) {
                        i = k;
                        Ai = A[k];
                }
        }
        return i;
```

Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Feld A aufsteigend sortiert

```
SelectionSort(A):
        for j = 0 to n − 1 {
                i = IndexOfMin(A, j);
                if (i ≠ j)
                        Swap(A, i, j);
        }
```

Vor- und Nachbedingungen

Input: Array $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen
Output: Array A aufsteigend sortiert
SelectionSort(A)**:**

```
for j = 0 to n − 1 {
        i = IndexOfMin(A, j);
        if (j ≠ i) {
                Swap(A, i, j);
        }
}
```

$$
\begin{aligned}
VOR_{sort} &= n \geq 0 \\
NACH_{sort} &= A \text{ enthält Permutation der Originaldaten} \quad \wedge \\
&\quad A[0] \leq A[1] \leq ... \leq A[n-1]
\end{aligned}
$$

Vor- und Nachbedingungen

Input: Array $A[0..n-1]$ von $n > 0$ natürlichen Zahlen; Startindex j
Output: Index $i \geq j$ des kleinsten Elements im Rest $A[j..n-1]$
IndexOfMin(A, j)**:**

```
i = j;
for k = j + 1 to n − 1 {
        if (A[k] < A[i]) {
                i = k;
        }
}
```

$$VOR_{min} = n > 0 \quad \wedge \quad 0 \leq j < n$$

$$NACH_{min} = j \leq i < n \quad \wedge \quad A[i] \leq A[k] \quad \forall k \in \{j, ..., n-1\}$$

Vor- und Nachbedingungen

Input: Array $A[0..n-1]$ von $n > 0$ natürlichen Zahlen; Indices i, j
Output: Array A mit Zellen i und j vertauscht
Swap(A, i, j)**:**

```
k = A[j];
A[j] = A[i];
A[i] = k;
```

$$
\begin{aligned}
VOR_{swap} &= n > 0 \quad \wedge \quad 0 \leq i, j < n \quad \wedge \quad A[i] = a \quad \wedge \quad A[j] = b \\
NACH_{swap} &= A[i] = b \quad \wedge \quad A[j] = a \quad \wedge \\
&\quad A[k] \text{ unverändert } \quad \forall k \in \{0, ..., n-1\} \setminus \{i, j\}
\end{aligned}
$$

$$\forall a, b \in \mathbb{Z}$$

Korrektheit von Selection Sort

$\{VOR\}$

$j = 0;$
while $(j < n)$ {
  $i = \text{IndexOfMin}(A, j);$
  if $(j \neq i)$ {
    $\text{Swap}(A, i, j);$
  }
  $j = j + 1;$
}

$\{NACH\}$

- Schleifeninvariante P beschreibt sortierten ($A[0..j-1]$) und unsortierten Teil ($A[j..n-1]$)

$$P \quad = \quad A \text{ enthält Permutation der Originaldaten} \quad \wedge$$
$$A[0] \leq A[1] \leq \ldots \leq A[j-1] \leq A[j..n-1]$$

- Unterschied zur Verifikation des Insertion Sort!

Korrektheit von Selection Sort

$\{VOR\}$
$j = 0;$
$\{C_1\}$
while $(j < n)$ {
  $i = \mathsf{IndexOfMin}(A, j);$
  if $(j \neq i)$ {
    $\mathsf{Swap}(A, i, j);$
  }
  $j = j + 1;$
}
$\{NACH\}$

- Prüfung des Schleifeneintritts: Sortierter Teil ist leer, daher gilt P.

$$C_1 \quad = \quad VOR \quad \wedge \quad j = 0 \quad \wedge \quad P$$

Korrektheit von Selection Sort

$\{VOR\}$
$j = 0;$
while $(j < n)$ {
$\{C_2\}$
$i = \text{IndexOfMin}(A, j);$
if $(j \neq i)$ {
$\text{Swap}(A, i, j);$
}
$j = j + 1;$
}
$\{NACH\}$

- Prüfung des Schleifenkörpers I: Es gelten P und die Schleifenbedingung $j < n$, damit auch die Vorbedingung von **IndexOfMin**.

$$C_2 \quad = \quad 0 \leq j < n \quad \wedge \quad P \quad \wedge \quad ...$$

Korrektheit von Selection Sort

$\{VOR\}$
```
        j = 0;
        while (j < n) {
                i = IndexOfMin(A, j);
{C_3}
                if (j ≠ i) {
                        Swap(A, i, j);
                }
                j = j + 1;
        }
```
$\{NACH\}$

- Prüfung des Schleifenkörpers II: Es gelten C_2 (keine Änderungen!) und die Nachbedingungen von **IndexOfMin**.

$$\begin{aligned} C_3 \quad = \quad & 0 \leq j < n \quad \wedge \quad P \quad \wedge \quad ... \quad \wedge \\ & j \leq i < n \quad \wedge \quad A[i] \leq A[k] \quad \forall k \in \{j, ..., n-1\} \end{aligned}$$

Korrektheit von Selection Sort

$\{VOR\}$

```
        j = 0;
        while (j < n) {
                i = IndexOfMin(A, j);
                if (j ≠ i) {
                        Swap(A, i, j);
                }
{C4}
                j = j + 1;
        }
```

$\{NACH\}$

- Prüfung des Schleifenkörpers III: Verschiebe minimales unsortierte Element $A[i]$ auf Stelle j; trivial falls $i = j$, ansonsten **Swap**en.

$$C_4 \quad = \quad 0 \leq j < n \quad \wedge \quad P \quad \wedge \quad \ldots \quad \wedge \quad A[j] \leq A[(j+1)..(n-1)]$$

Korrektheit von Selection Sort

$\{VOR\}$

```
        j = 0;
        while (j < n) {
                i = IndexOfMin(A, j);
                if (j ≠ i) {
                        Swap(A, i, j);
                }
                j = j + 1;
```

$\{C_5\}$

```
        }
```

$\{NACH\}$

- Prüfung des Schleifenkörpers IV: Dadurch ist $A[j] \geq A[j-1]$ und minimal im Vergleich zum Rest, also bleibt P gültig.

$$C_5 \quad = \quad (0 \leq j < n \quad \vee \quad j \geq n) \quad \wedge \quad P \quad \wedge \quad \ldots$$

Korrektheit von Selection Sort

$\{VOR\}$

```
j = 0;
while (j < n) {
        i = IndexOfMin(A, j);
        if (j ≠ i) {
                Swap(A, i, j);
        }
        j = j + 1;
}
```

$\{NACH\}$

- Prüfung der Nachbedingung: „Trennindex“ j erreicht das Ende, der unsortierte Teil verschwindet, und es ergibt sich die Nachbedingung

$$j \geq n \quad \wedge \quad P \quad \Rightarrow \quad NACH$$

Korrektheit von Selection Sort

- Sonderfälle wurden vernachlässigt!
- Die Darstellung ist verkürzt!
- **IndexOfMin** und **Swap** wurden hier nicht verifiziert!

- Lösungsblatt nachbearbeiten!