Übung zu
Algorithmen und Datenstrukturen (für ET/IT)

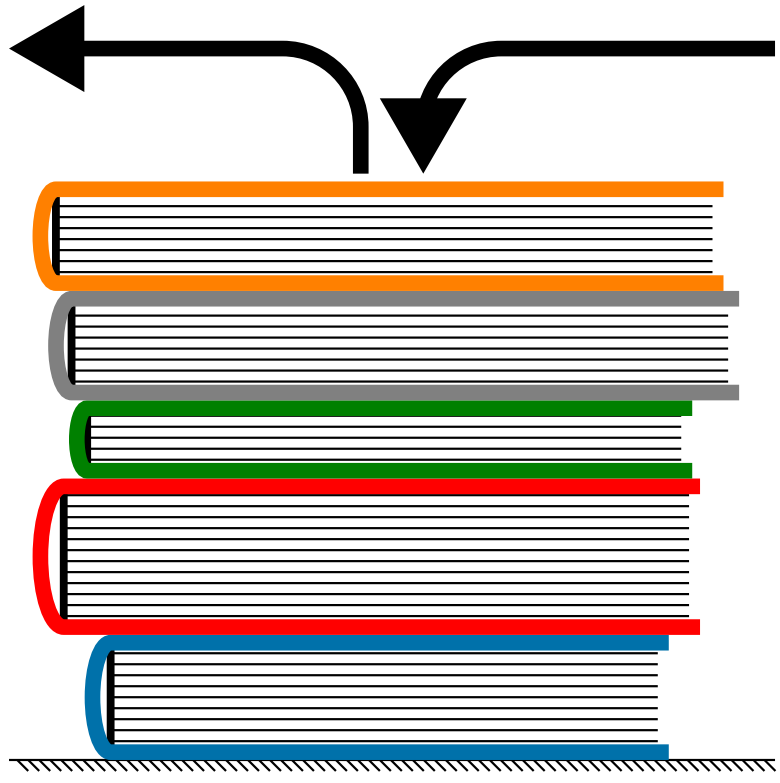
Sommersemester 2017

Rüdiger Göbl, Mai Bui

Computer Aided Medical Procedures
Technische Universität München

Stack

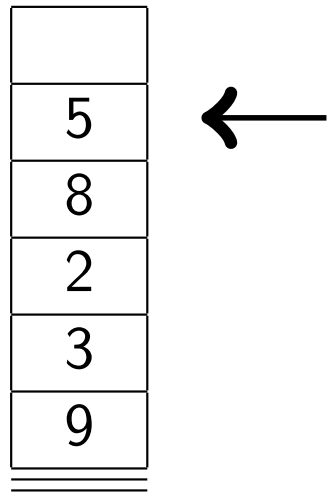
- Abstrakter Datentyp
- Operationen auf Stacks:
  - `PUSH`
  - `POP`
  - `TOP`
  - `isEmpty`
- Constraint
  - LI-FO (Last In, First Out)
  - „Bücherstapel“
- Mögl. Implementation:
  - Array mit Schreibindex
  - verkettete Liste



Stack

Rückgabe Term:

Zustand Stapel S:



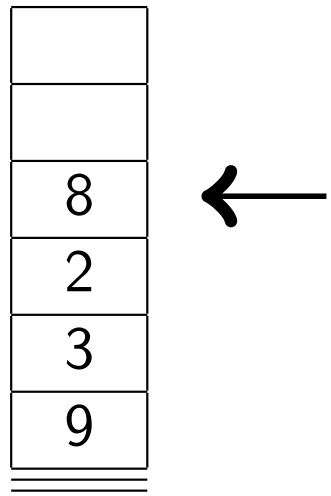
initialer Zustand

Stack

Rückgabe Term:

5

Zustand Stapel S:



pop(S)

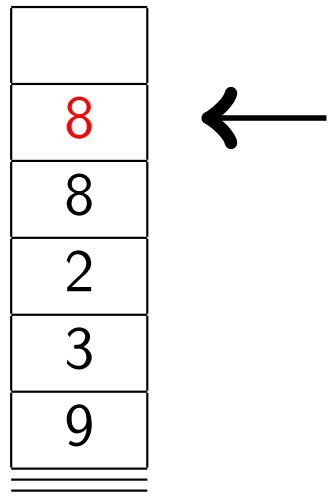
→ entferne das letzte Element vom Stack

→ liefert 5 zurück

Stack

Rückgabe Term:

Zustand Stapel S:



push(S,8)

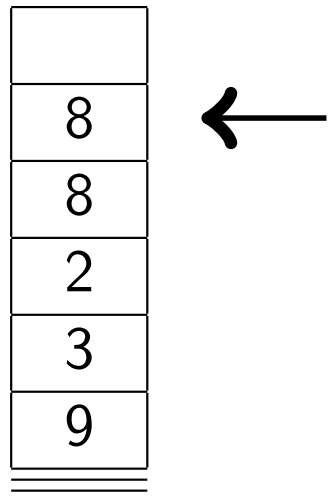
→ lege Element 8 auf den Stack

Stack

Rückgabe Term:

8

Zustand Stapel S:



top(S)

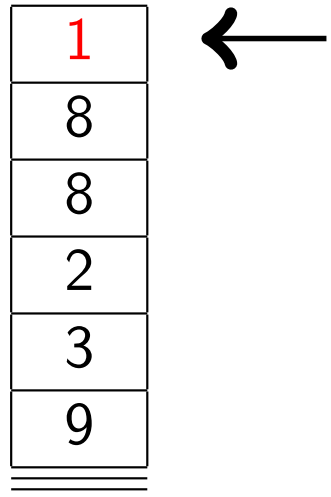
→ hole das letzte Element vom Stack

→ kein Löschen des Elementes aus dem Stack

Stack

Rückgabe Term:

Zustand Stapel S:



push(S,1)

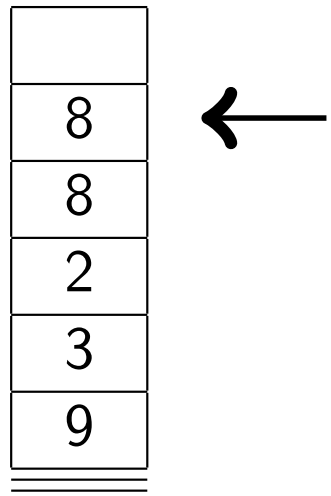
→ lege Element 1 auf den Stack

Stack

Rückgabe Term:

1

Zustand Stapel S:



pop(S)

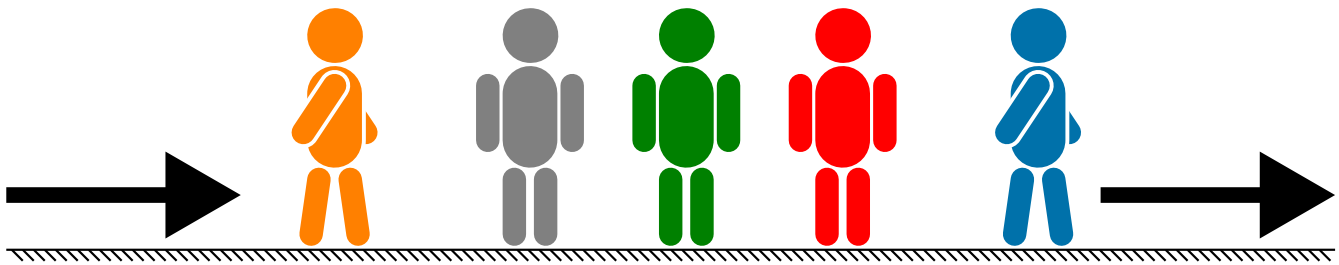
→ entferne das letzte Element vom Stack

→ liefert 5 zurück

Queue

- Abstrakter Datentyp
- Interface-Funktionen
  - `ENQUEUE`
  - `DEQUEUE`
- Constraint
  - FI-FO (First In, First Out)
  - „Warteschlange“

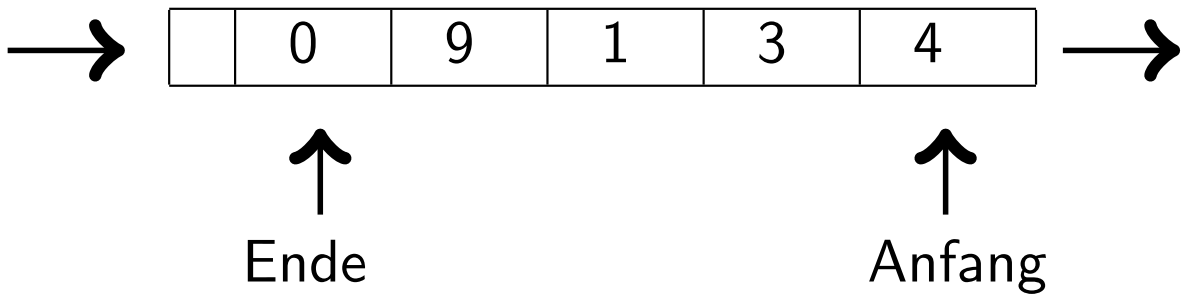
- Mögl. Implementation:
  - Verkettete Liste
  - Zwei Stacks
  - Ringpuffer



Queue

Rückgabe Term:

Zustand Queue Q:



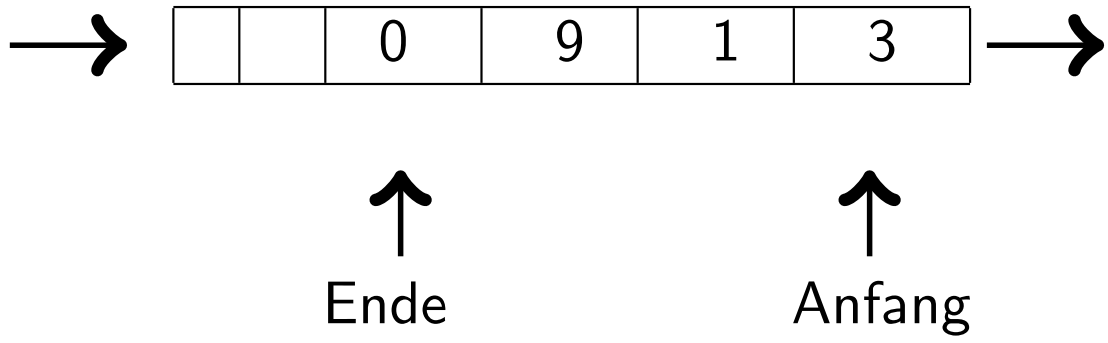
initialer Zustand

Queue

Rückgabe Term:

4

Zustand Queue Q:



dequeue(Q)

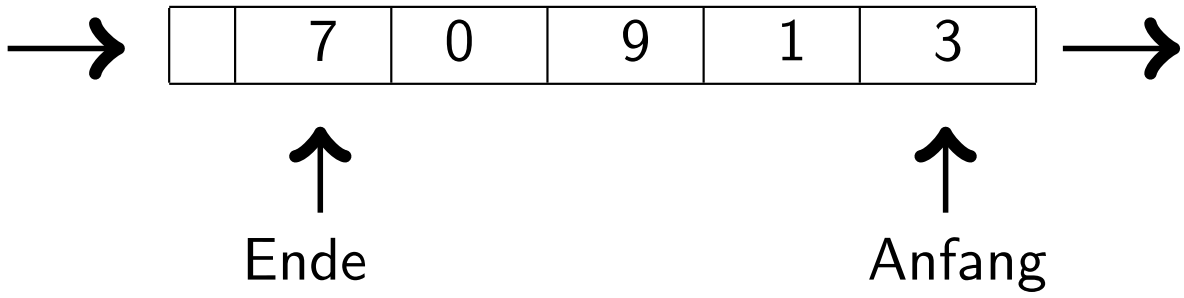
$\rightarrow$ Rückgabe des Wertes 4

$\rightarrow$ Update der Queue

Queue

Rückgabe Term:

Zustand Queue Q:



enqueue(Q,7)

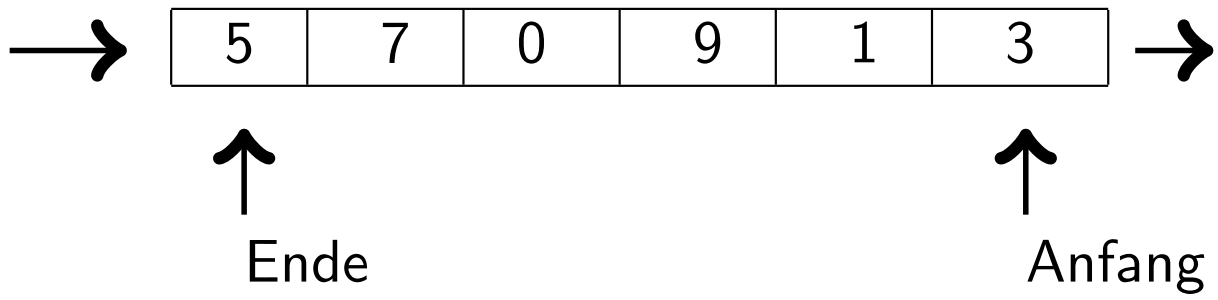
→ Hinzufügen des Wertes 7

→ Update der Queue

Queue

Rückgabe Term:

Zustand Queue Q:



enqueue(Q,5)

→ Hinzufügen des Wertes 5

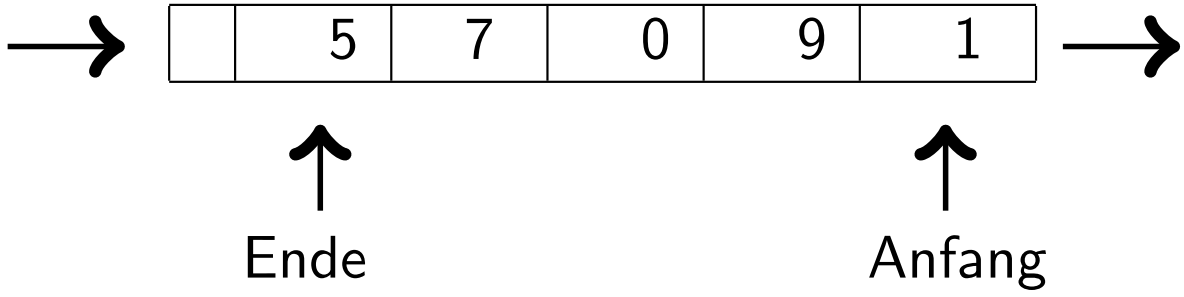
→ Update der Queue

Queue

Rückgabe Term:

3

Zustand Queue Q:



dequeue(Q)

→ Rückgabe des Wertes 3

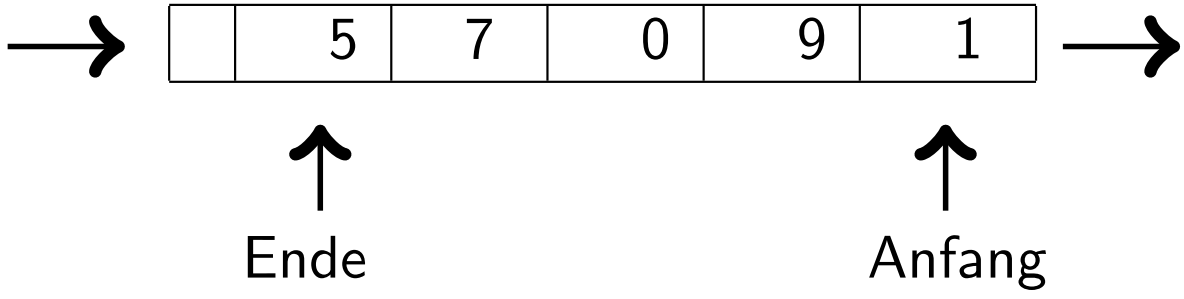
→ Update der Queue

Queue

Rückgabe Term:

False

Zustand Queue Q:

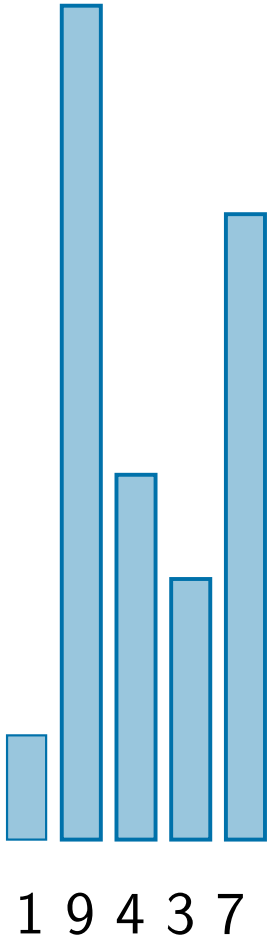


isEmtpy(Q)

→ Rückgabe des Wertes False

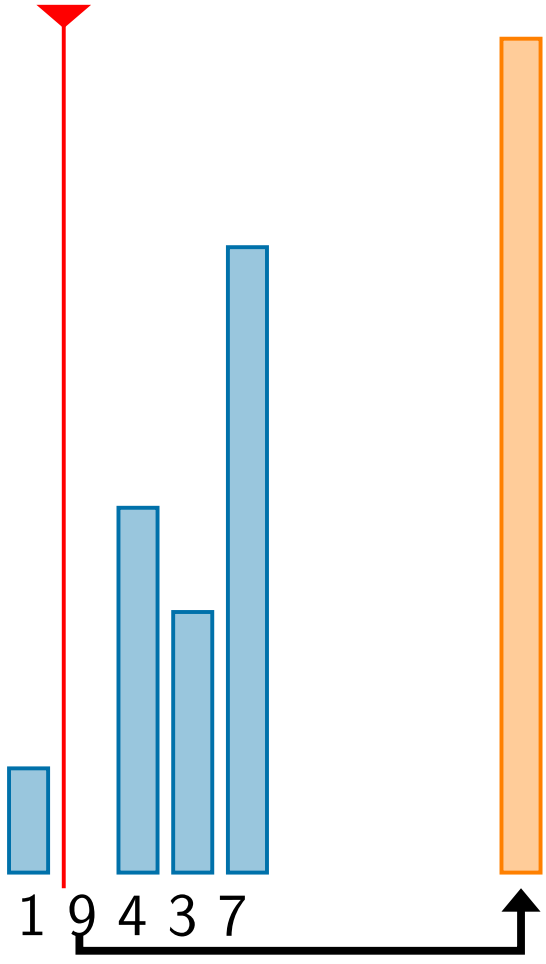
→ Überprüfen ob die Queue leer ist

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;
        }
```

Insertion Sort

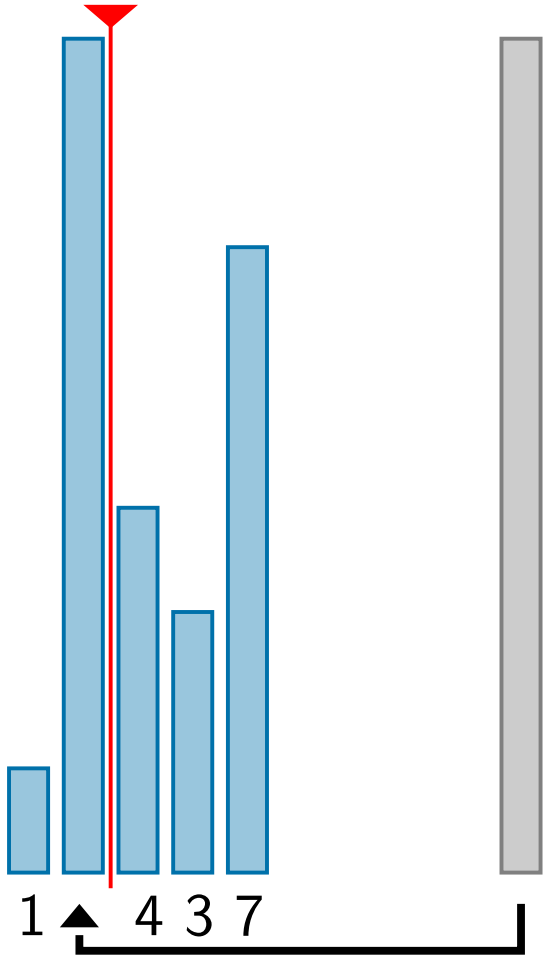


$j = 1$: Anfangszustand

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 { // j = 1
    key = A[j]; // key = 9
    i = j − 1; // i = 0
    while (i ≥ 0 && A[i] > key) {
        A[i + 1] = A[i];
        i = i − 1;
    }
    A[i + 1] = key;
}
```

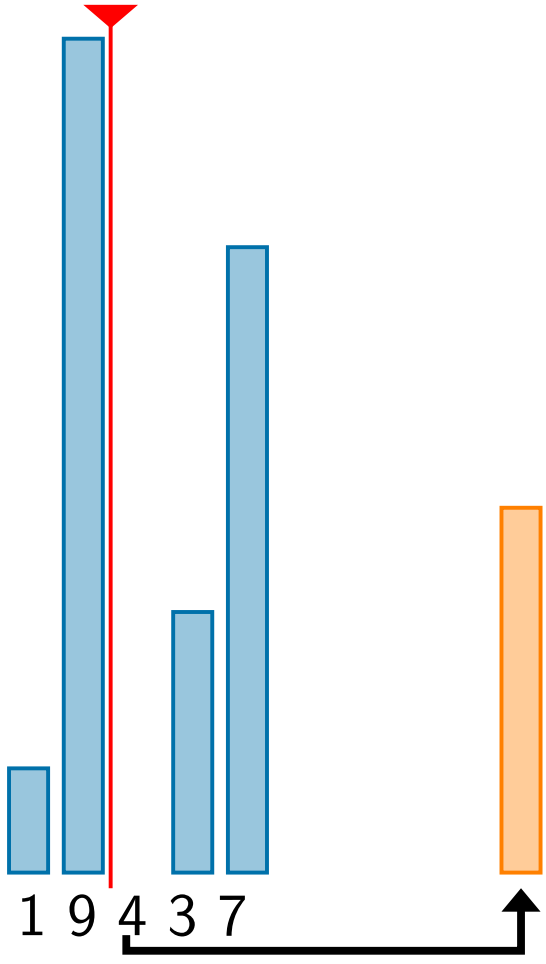
Insertion Sort



$j = 1$: 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; // A[1] = 9
        }
```

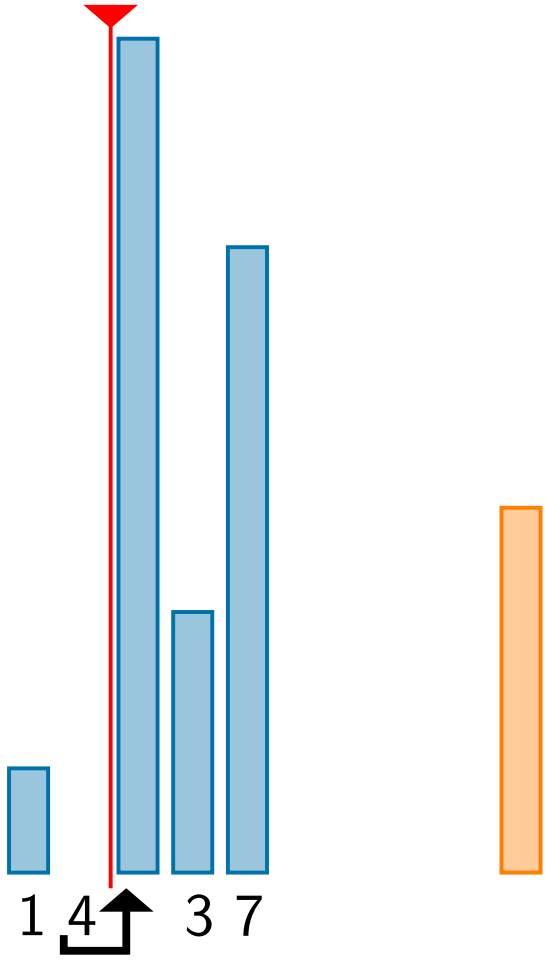
Insertion Sort



$j = 2$: 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 { // j = 2
                key = A[j]; // key = 4
                i = j − 1; // i = 1
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

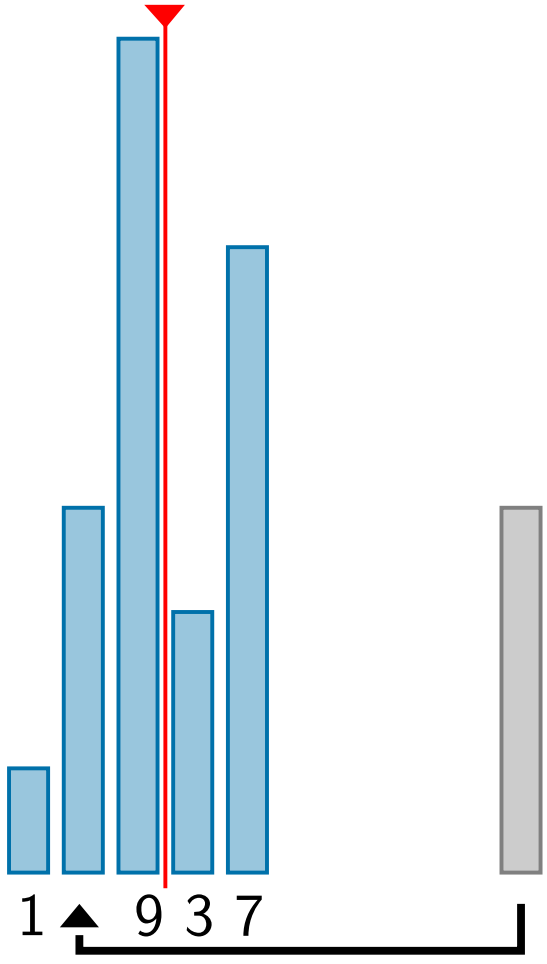
Insertion Sort




```
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];// A[2] = 9
                        i = i − 1;// i = 0
                }
                A[i + 1] = key;
        }
```

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

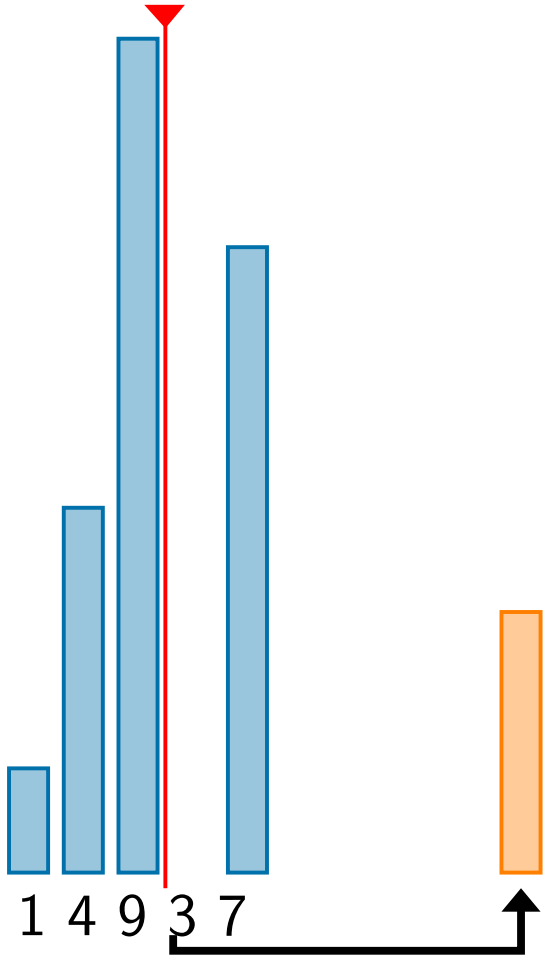
Insertion Sort



$j = 2$: 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; // A[1] = 4
        }
```

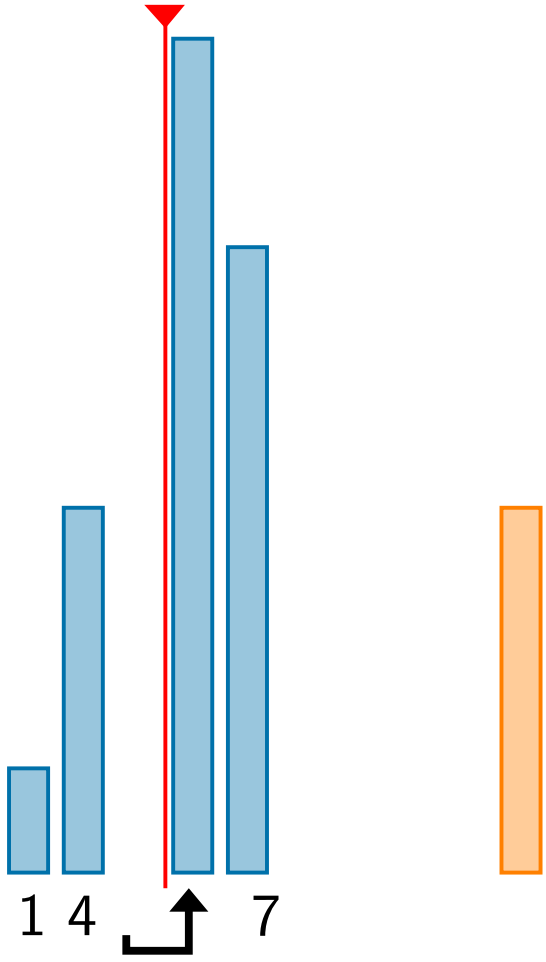
Insertion Sort



$j = 3$: 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 { // j = 3
                key = A[j]; // key = 3
                i = j − 1; // i = 2
                while (i ≥ 0 && A[i] > key) {
                        A[i + 1] = A[i];
                        i = i − 1;
                }
                A[i + 1] = key;
        }
```

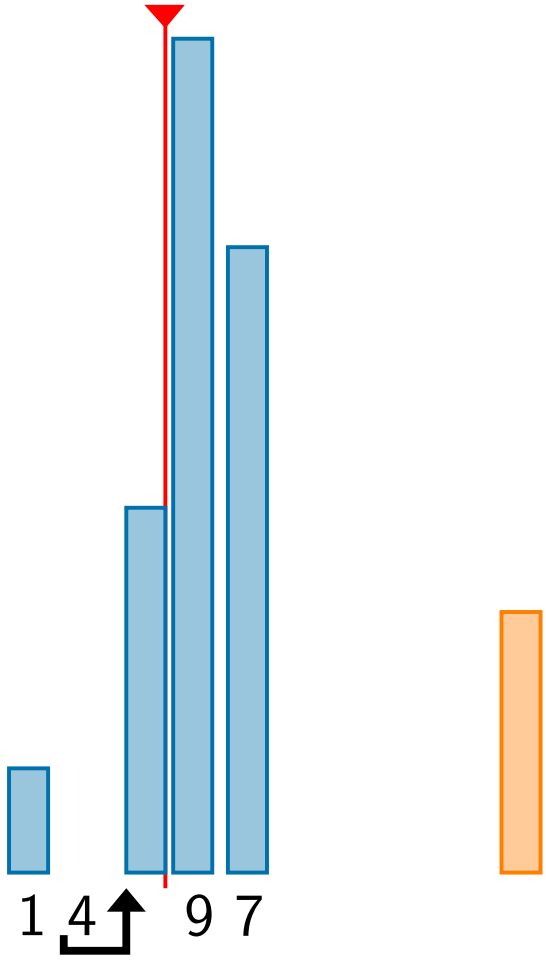
Insertion Sort



$j = 3$: 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];// A[3] = 9
                        i = i − 1;// i = 1
                }
                A[i + 1] = key;
        }
```

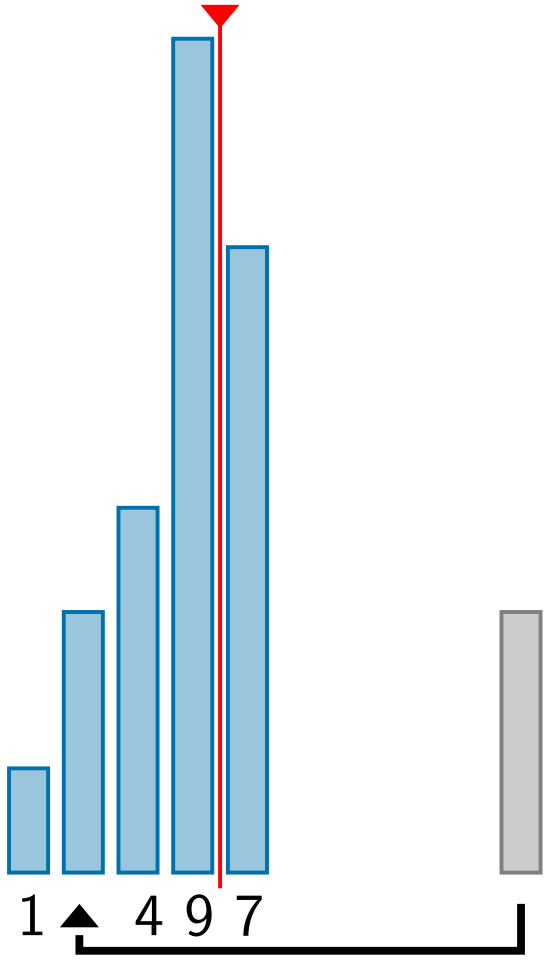
Insertion Sort




```
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];// A[2] = 4
                        i = i − 1;// i = 0
                }
                A[i + 1] = key;
        }
```

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

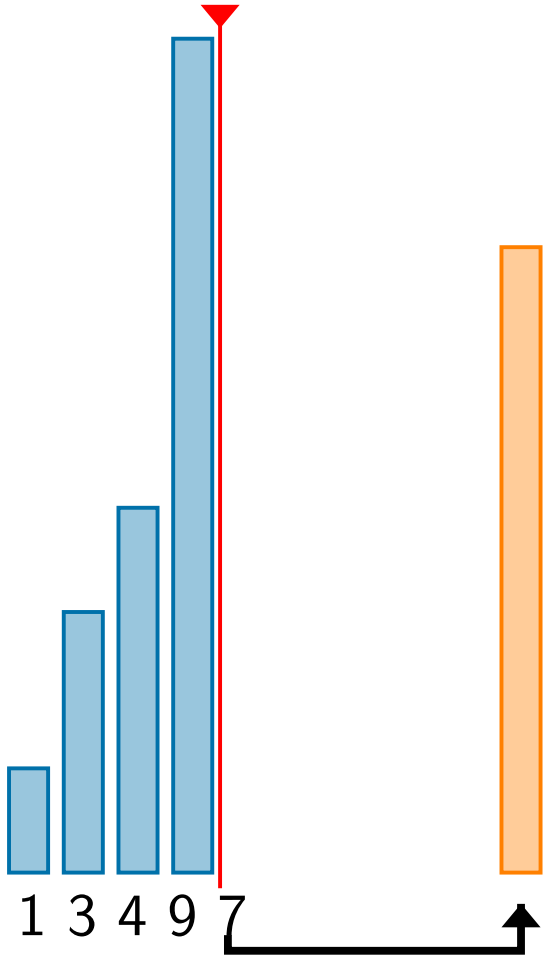
Insertion Sort



$j = 3$: 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; // A[1] = 3
        }
```

Insertion Sort

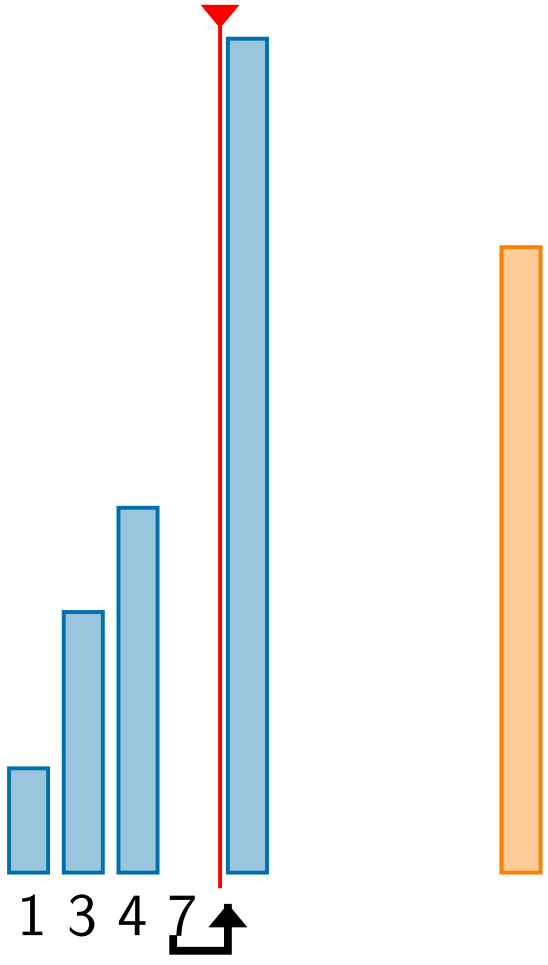


$j = 4$: Anfangszustand

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]; // key = 7
    i = j − 1;// i = 3
    while (i ≥ 0 && A[i] > key) {
        A[i + 1] = A[i];
        i = i − 1;
    }
    A[i + 1] = key;
}
```

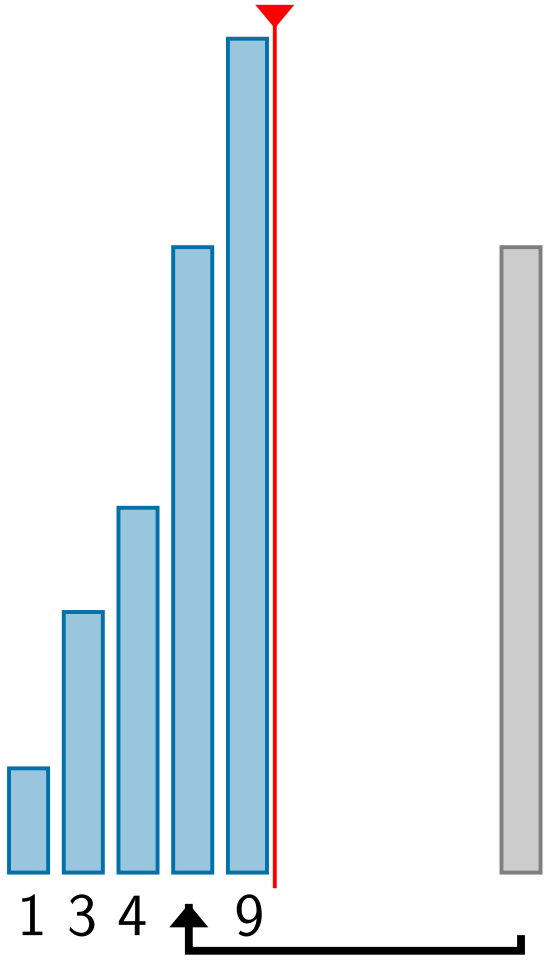
Insertion Sort




```
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];// A[4] = 9
                        i = i − 1;// i = 2
                }
                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; // A[3] = 7
        }
```

Wiederholung: Partielle Korrektheit

Partielle Korrektheit

Ein Programm mit Anweisungen ANW sowie Vorbedingung $\{VOR\}$ und Nachbedingung $\{NACH\}$ heißt partiell korrekt bezüglich VOR, NACH genau dann, wenn

$$\{VOR\}\ \text{ANW}\ \{NACH\}$$

wahr ist.

Das Programm heißt total korrekt bezüglich VOR, NACH genau dann, wenn

- es partiell korrekt ist bezüglich VOR, NACH ist und
- ANW immer dann terminiert, wenn VOR gilt.

Wiederholung: Bausteine von Algorithmen

Wiederholung: Bausteine von Algorithmen

- Elementarer Verarbeitungsschritt (z.B. Zuweisung an Variable)
- Sequenz (elementare Schritte nacheinander)
- Bedingter Verarbeitungsschritt (z.B. if/else)
- Wiederholung (z.B. while-Schleife)

Wiederholung: Korrektheit von Anweisungstypen I

- Elementarer Verarbeitungsschritt α

$$\{VOR\}\ \alpha\ \{NACH\}$$

- Sequenz α; β

$$\{VOR\}\ \alpha;\ \beta\ \{NACH\}$$

wird gezeigt mittels Zwischenbedingung MITTE, also

$$\{VOR\}\ \alpha\ \{MITTE\}$$

und

$$\{MITTE\}\ \beta\ \{NACH\}$$

Wiederholung: Korrektheit von Anweisungstypen II

- Bedingter Verarbeitungsschritt

$$\{VOR\}\ \textbf{if}\ (\mathrm{B})\ \{\ \alpha\ \}\ \text{else}\ \{\ \beta\ \}\ \{NACH\}$$

wird gezeigt mittels

$$\{VOR\} \wedge \{B\}\ \alpha\ \{NACH\}$$

und

$$\{VOR\} \wedge \neg\ \{B\}\ \beta\ \{NACH\}$$

$\rightarrow$ zeige also dass NACH gilt, egal welcher Zweig ausgeführt wird!

Wiederholung: Korrektheit von Anweisungstypen III

- Wiederholung

  $\{VOR\}$
  while (B) { β }
  $\{NACH\}$

  wird gezeigt mittels Schleifeninvariante P:

  1. prüfe, daß VOR $\Rightarrow$ P
     (Vorbedingung garantiert Schleifeninvariante bei Eintritt in Schleife)
  2. prüfe, daß
     $$\{P\} \wedge \{B\}\ \beta\ \{P\}$$
     (wenn Schleife durchlaufen, bleibt Schleifeninvariante wahr)
  3. prüfe, daß $\{\text{P} \wedge \neg \text{B}\} \Rightarrow$ NACH
     (Nachbedingung muß nach Verlassen der Schleife gelten)

Verifikation der Multiplikation durch Addition

Input: Ganze Zahlen a und b
Output: Produkt $c = a \cdot b$
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:

$$\begin{aligned} VOR &= a \in \mathbb{Z} \ \wedge \ b \in \mathbb{Z} \\ NACH &= (c = a \cdot b) \end{aligned}$$

Verifikation der Multiplikation durch Addition

Input: Ganze Zahlen a und b
Output: Produkt $c = a \cdot b$
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;
```

Schleifeninvariante:

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

Verifikation der Multiplikation durch Addition

$\{VOR\}$

```
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;
        }
}
```

$\{NACH\}$

- Überprüfung der einzelnen Programmabschnitte

Verifikation der Multiplikation durch Addition

$\{VOR\}$

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

$c = 0;$

}

$\{NACH\}$

- Zeige wenn $\{VOR\}$ und if-Bedingung gilt folgt $\{NACH\}$

  $a \in \mathbb{Z} \quad \wedge \quad b \in \mathbb{Z} \quad \wedge \quad (a == 0 \quad \vee \quad b == 0) \Rightarrow (c = a \cdot b = 0)$

Verifikation der Multiplikation durch Addition

$\{VOR\}$

...
} **else if** $(a < 0)$ {
$c = -Multiply(-a, b);$
}

$\{NACH\}$

- Falls **Multiply** korrekt gilt $\{NACH\}$
  $\{VOR\} \quad \wedge \quad \neg(a == 0 \quad \vee \quad b == 0) \quad \wedge \quad a < 0$
  $= \{VOR\} \quad \wedge \quad a < 0 \quad \wedge \quad b \neq 0$
  Bei $a < 0$:
  $a = -\|a\| = -1 \cdot \|a\| = -1 \cdot (-a)$
  $c = a \cdot b = (-1 \cdot (-a)) \cdot b$
  Assoziativgesetz:
  $c = -1 \cdot ((-a) \cdot b)$

Verifikation der Multiplikation durch Addition

$\{VOR\}$

```
        ...
        } else {
```

$\{C_0\}$

```
                c = 0;
                i = 1;
```

$\{C_1\}$

```
                while(i ≤ a) {
```

$\{C_2\}$

```
                        c = c + b;
                        i = i + 1;
```

$\{C_3\}$

```
                }
        }
```

$\{NACH\}$

- Vereinfachung der Vorbedingung
  $C_0 = \{VOR\} \quad \wedge \quad \neg(a == 0 \quad \vee \quad b == 0) \quad \wedge \quad \neg(a < 0)$
  $= \{VOR\} \quad \wedge \quad a > 0 \quad \wedge \quad b \neq 0$

Verifikation der Multiplikation durch Addition

$\{VOR\}$

...

$c = 0;$

$i = 1;$

$\{C_1\}$

while$(i \leq a)$ {

$\{C_2\}$

$c = c + b;$

$i = i + 1;$

$\{C_3\}$

}

}

$\{NACH\}$

- Korrektheit des Schleifeneintritts: $C_1 \Rightarrow P$
  $C_1 = (C_0 \quad \wedge \quad c == 0 \quad \wedge \quad i == 1)$
  $P = c + (a - i + 1) \cdot b = 0 + (a - 1 + 1) \cdot b = a \cdot b$

Verifikation der Multiplikation durch Addition

$\{VOR\}$

...

while$(i \leq a)$ {

$\{C_2\}$

$c = c + b;$

$i = i + 1;$

$\{C_3\}$

}

}

$\{NACH\}$

- Prüfung des Schleifenkörpers : $C_2 = (P \quad \wedge \quad i \leq a)$
  $c' = c + b$
  $i' = i + 1$
  $c' + (a - i' + 1) \cdot b = (c + b) + (a - (i + 1) + 1) \cdot b$
  $= c + b + (a - i) \cdot b$
  $= c + (a - i + 1) \cdot b$
  $= a \cdot b$

Verifikation der Multiplikation durch Addition

$\{VOR\}$

...

while$(i \leq a)$ {

$\{C_2\}$

$c = c + b;$

$i = i + 1;$

$\{C_3\}$

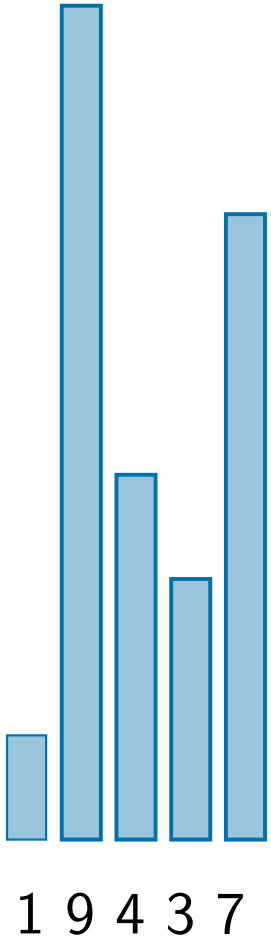
}

}

$\{NACH\}$

- Prüfung der Nachbedingung: $(P \quad \wedge \quad \neg(i \leq a)) \Rightarrow \{NACH\}$
  es gilt $i = a + 1$
  $a \cdot b = c + (a - i + 1) \cdot b$
  $= c + (a - (a + 1) + 1) \cdot b$
  $= c + (a - a - 1 + 1) \cdot b$
  $= c + 0 \cdot b$
  $= c$

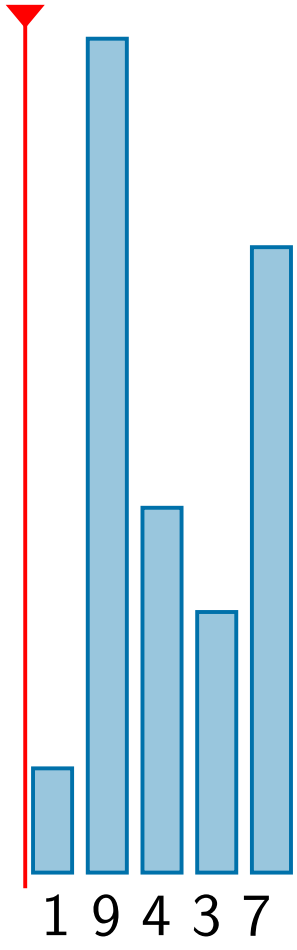
Selection Sort



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 (j ≠ i) {
        Swap(A, i, j);

    }

}
```

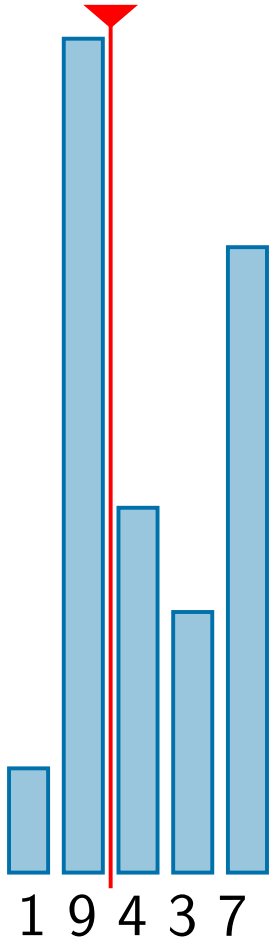
Selection Sort




```
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); // i = 0

                if (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 0$: Index des kleinsten Elements

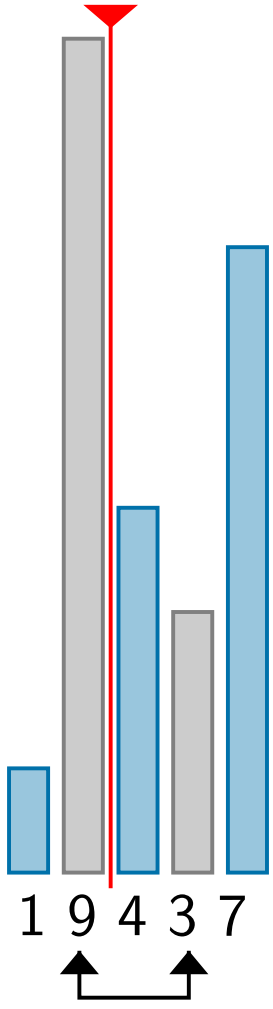
Selection Sort




```
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); // i = 3

                if (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 1$: Anfangszustand

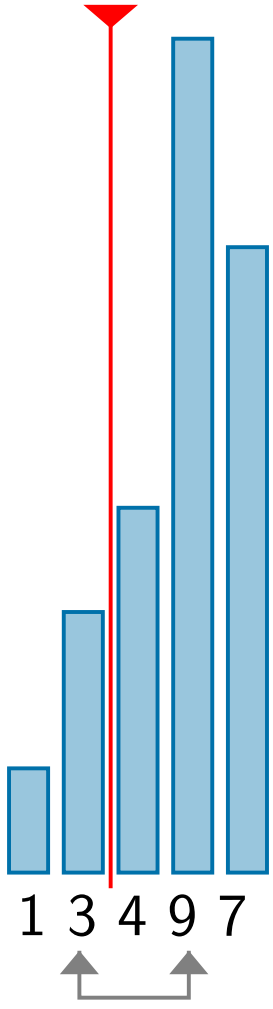
Selection Sort




```
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 (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 1$: Swap 9 und 3

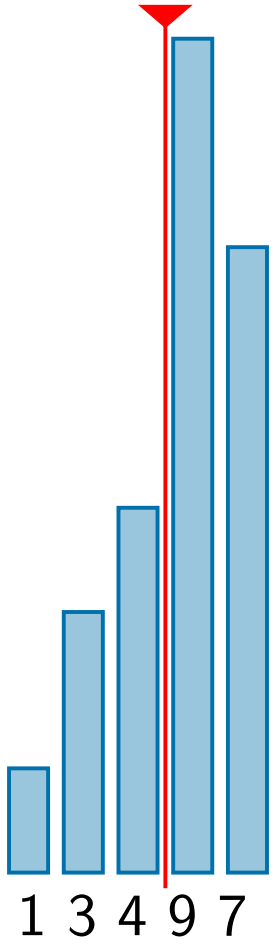
Selection Sort




```
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 (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 1$: Swap 9 und 3

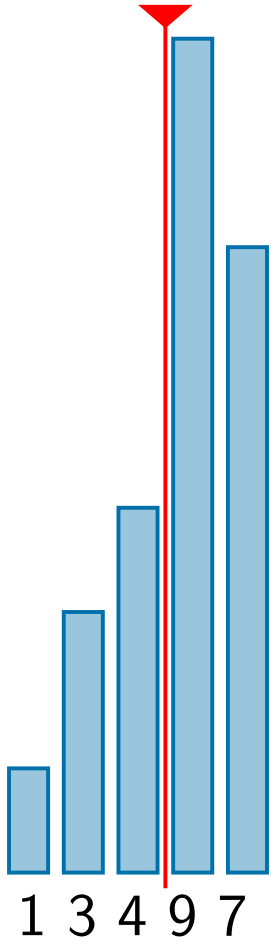
Selection Sort




```
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); // i = 2

                if (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 2$: Index des kleinsten Elements

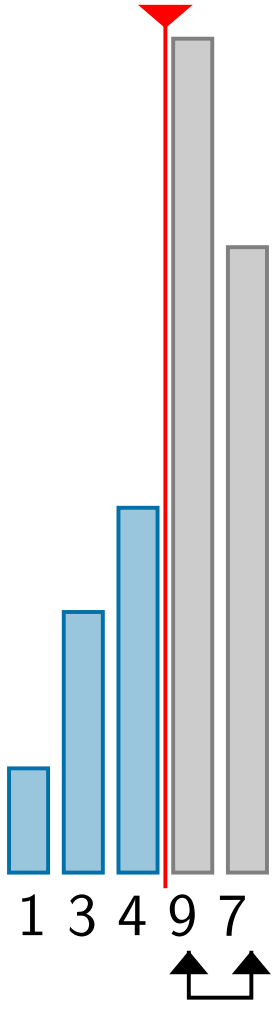
Selection Sort




```
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); // i = 4

                if (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 3$: Anfangszustand

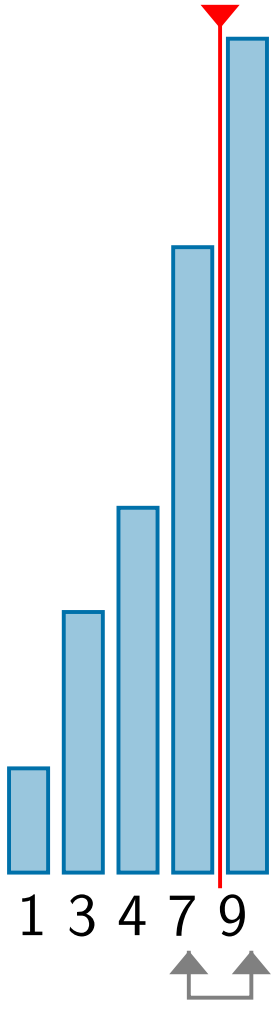
Selection Sort




```
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); // i = 4

                if (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 3$: Swap 9 und 7

Selection Sort




```
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); // i = 4

                if (j ≠ i) {
                        Swap(A, i, j);

                }

        }
```

$j = 3$: Swap 9 und 7

Verifikation von Selection Sort

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);
        }
}
```

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

Verifikation von Selection Sort

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;
        }
}
```

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

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

Verifikation von Selection Sort

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;
```

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

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

Partielle Korrektheit von Selection Sort

$\{VOR\}$

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

$\{NACH\}$

- Ersetzen der For-Schleife in eine äquivalente While-Schleife
- Schleifeninvariante P beschreibt sortierten ($A[0..j-1]$) und unsortierten Teil ($A[j..n-1]$)

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

Partielle Korrektheit von Selection Sort

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

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

$$\begin{aligned} C_1 &= VOR_{sort} \wedge j == 0 \\ &= n \geq 0 \wedge j == 0 \Rightarrow P \end{aligned}$$

Partielle Korrektheit von Selection Sort

$\{VOR\}$

```
        j = 0;
        while (j < n) {
{C_2}
                i = IndexOfMin(A, j);
                if (j ≠ i) {
                        Swap(A, i, j);
                }
                j = j + 1;
        }
```

$\{NACH\}$

- Prüfung des Schleifenkörpers I: Es gelten P und die Schleifenbedingung $j < n$, für $n > 0$ kompatibel mit Vorbedingung von **IndexOfMin**.

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

Partielle Korrektheit von Selection Sort

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

- Prüfung des Schleifenkörpers II: Es gelten C_2 und bei Korrektheit von **IndexOfMin** die Nachbedingungen von **IndexOfMin**.

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

Partielle 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.

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

Partielle Korrektheit von Selection Sort

$\{VOR\}$

$j = 0;$

while $(j < n)$ {

$i = \text{IndexOfMin}(A, j);$

if $(j \neq i)$ {

$\text{Swap}(A, i, j);$

}

$j = j + 1;$

$\{C_5\}$

}

$\{NACH\}$

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

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

Partielle 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

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

Partielle 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: Im Fall $n > 0$:

$$
\begin{aligned}
C_6 \quad = \quad & n > 0 \quad \wedge \quad j \geq n \quad \wedge \quad P = n > 0 \quad \wedge \quad j \geq n \quad \wedge \\
& \text{A enthält permutationen der Originaldaten} \quad \wedge \\
& A[0] \leq A[1] \leq ... \leq A[j-1] \leq A[j..(n-1)] \\
= \quad & n > 0 \quad \wedge \quad j \geq n \quad \wedge \quad ... \quad \wedge \quad A[0] \leq A[1] \leq ... \leq A[n-1] \\
= \quad & C_5 \quad \wedge \quad \neg(j < n) \Rightarrow \{NACH\}
\end{aligned}
$$

Partielle Korrektheit von Selection Sort

$\{VOR\}$

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

$\{NACH\}$

- Prüfung der Nachbedingung:
  Im Fall $n = 0$:
  $C_6 = C_1 \quad \wedge \quad \neg(j < n)$
  $C_6 = n == 0 \quad \wedge \quad j == 0 \quad \wedge \quad P \Rightarrow \{NACH\}$,
  Permutationseigenschaft und Sortierungseigenschaft des leeren Arrays

Partielle Korrektheit von IndexOfMin

$\{VOR\}$

```
i = j;
k = j + 1;
while (k < n) {
        if (A[k] ≤ A[i]) {
                i = k;
        }
        k = k + 1;
}
```

$\{NACH\}$

- Schleifeninvariante P

$$P \quad = A[i] \leq A[j..(k-1)]$$

mit k = Index der gerade untersuchten Stelle

Partielle Korrektheit von IndexOfMin

$\{VOR\}$
$i = j;$
$k = j + 1;$
$\{C_1\}$
while $(k < n)$ {
if $(A[k] \leq A[i])$ {
$i = k;$
}
$k = k + 1;$
}
$\{NACH\}$

- Prüfung des Schleifeneintritts:

$$C_1 \quad = \quad VOR \quad \wedge \quad i == j \quad \wedge \quad k == j + 1$$
$$i == j == k - 1 \Rightarrow A[i] \leq A[i] \Rightarrow P$$

Partielle Korrektheit von IndexOfMin

$\{VOR\}$
$i = j;$
$k = j + 1;$
while $(k < n)$ {
$\{C_2\}$
if $(A[k] \leq A[i])$ {
$i = k;$
}
$k = k + 1;$
}
$\{NACH\}$

- Prüfung des Schleifenkörpers I:
  $C_2 = n > 0 \quad \wedge \quad 0 < k < n \quad \wedge \quad P$
  $= n > 0 \quad \wedge \quad 0 < k < n \quad \wedge \quad A[i] \leq A[j..(k-1)]$

Partielle Korrektheit von IndexOfMin

$\{VOR\}$

$i = j;$
$k = j + 1;$
while $(k < n)$ {
  if $(A[k] \leq A[i])$ {
    $i = k;$
  }
$\{C_3\}$
  $k = k + 1;$
}
$\{NACH\}$

- Prüfung des Schleifenkörpers II:
  Nach der Fallunterscheidung muss gelten:

$$\begin{aligned} C_3 &= n > 0 \quad \wedge \quad \ldots \quad \wedge \quad A[i] \leq A[j..k] \\ &= n > 0 \quad \wedge \quad 0 < k < n \quad \wedge \quad P \quad \wedge \quad A[i] \leq A[k] \end{aligned}$$

Partielle Korrektheit von IndexOfMin

$\{VOR\}$
$i = j;$
$k = j + 1;$
while $(k < n)$ {
if $(A[k] \leq A[i])$ {
$i = k;$
}
$k = k + 1;$
$\{C_4\}$
}
$\{NACH\}$

- Prüfung des Schleifenkörpers III: Verschiebung der Trennstelle, P bleibt gültig

$$C_4 \quad = \quad n > 0 \quad \wedge \quad (0 < k < n \quad \vee \quad k \geq n) \quad \wedge \quad P$$

Partielle Korrektheit von IndexOfMin

$\{VOR\}$

$i = j;$
$k = j + 1;$
while $(k < n)$ {
  if $(A[k] \leq A[i])$ {
    $i = k;$
  }
  $k = k + 1;$
}

$\{NACH\}$

- Prüfung der Nachbedingung:
  Im Fall $n > j + 1$

$$C_5 \quad = \quad C_4 \quad \wedge \quad \neg(k < n) \Rightarrow \{NACH\}$$

  Im Fall $n = j + 1$

$$C_5 \quad = \quad C_1 \quad \wedge \quad \neg(k < n) \Rightarrow \{NACH\}$$

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