Übung zu
Algorithmen und Datenstrukturen (für ET/IT)

Wintersemester 2012/13

Jakob Vogel

Computer-Aided Medical Procedures
Technische Universität München

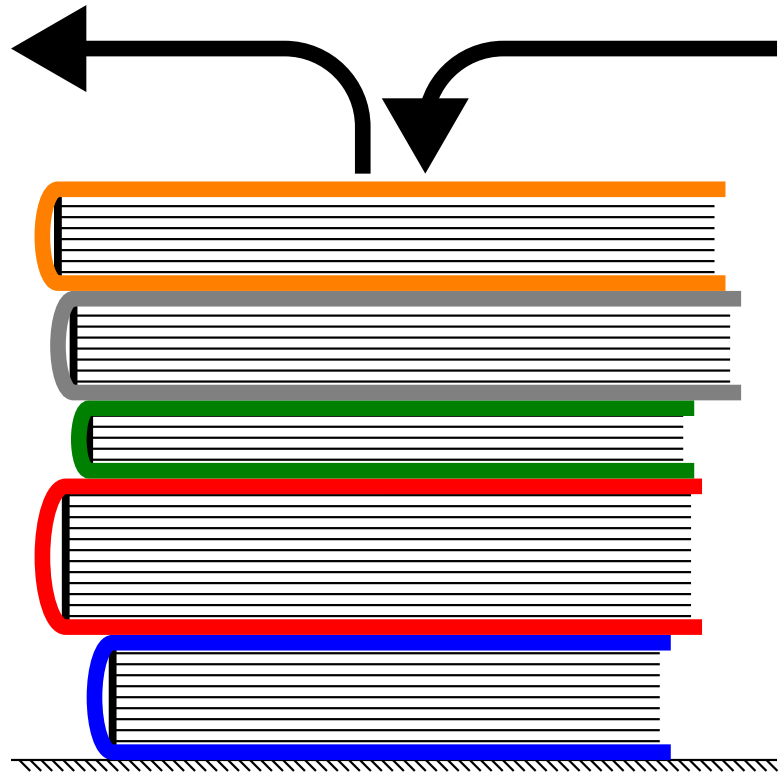
Abstrakte Datentypen

- Menge möglicher Operationen definiert das *Interface*
- Deren Auswirkungen sind wohldefiniert durch *Constraints*

- Die eigentliche Implementation ist *nicht* eindeutig definiert!

Stack

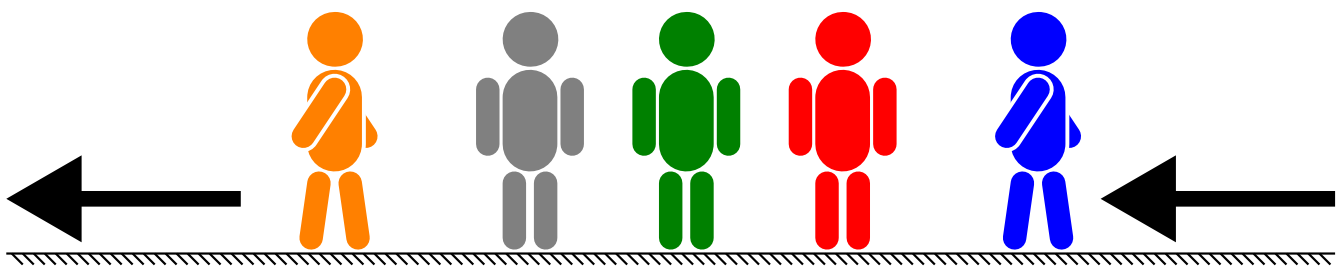
- Abstrakt definiert
- Interface-Funktionen
  - `PUSH`
  - `POP`
  - ...
- Constraint
  - LI-FO (Last In, First Out)
  - „Bücherstapel“
- Mögl. Implementation:
  - Array mit Schreibindex
  - Doppelt verkettete Liste
  - ...



Queue

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

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




Code

Unit Tests

- Korrektheitsbeweis ist oft nicht praktikabel.
- Code wird oft als **Black Box** betrachtet.
- Separater Test, ob bei bekanntem Input das tatsächliche dem erwarteten Output entspricht...
- Design der Testfälle kritisch!
- Später mehr...


Code

Was ist der `std::iterator`?

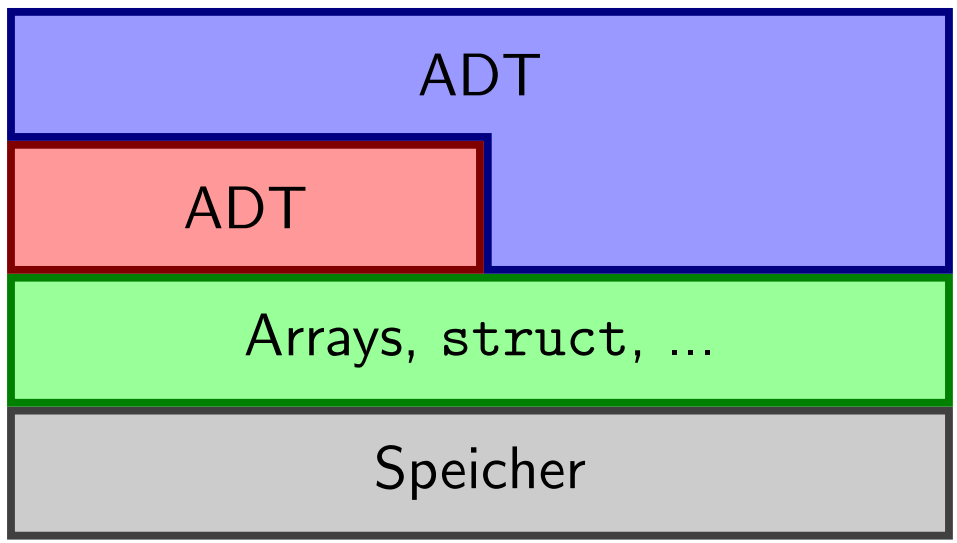
- Abstrakter Datentyp zum Durchlaufen von Datenstrukturen (Teil der Standardbibliothek)
- Interface von `std::iterator it` u.a.
  - Weiterschaltung über `it++`
  - Wert über `*it`
  - ...
- Verfügbar etwa in `std::vector` oder `std::list` via `begin()`, aber natürlich jeweils unterschiedlich implementiert!
- Bei einer verketteten Liste existiert etwa intern ein Zeiger `p` auf einen Container:
  - `it++` bewirkt `p = p->next;`
  - `*it` ergibt `p->data`
  - ...

Grundsätzliche Organisation von Daten im Speicher

- Einzelne Werte und Objekte
- Felder (*Arrays*) von unmittelbar hintereinander liegenden Werten oder Objekten
  - Speicherposition der Nachbarn berechenbar
  - Beispiel: Bei einem `int`-Array liegt `array[i+5]` genau `5*sizeof(int)` Bytes hinter `array[i]`!
  - Die Größe ist fix und wird gewählt zur Laufzeit (Heap) oder zur Compile-Zeit (Call Stack).
- „Zufällig“ verteilte Objekte, die auf ihre Nachbarn mittels Zeigern verweisen
  - Speicherposition der Nachbarn durch Verfolgen der Zeiger
  - Dynamische Größe und komplexere Topologie der Strukturen
  - Die Verwaltung ist aufwändiger!

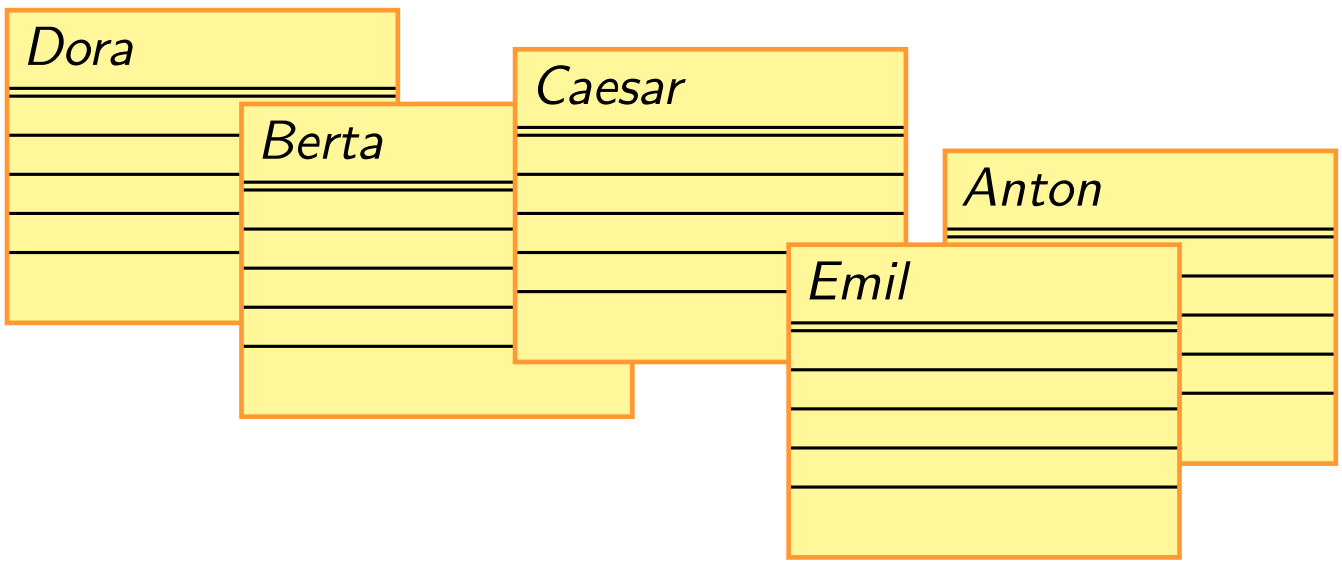
Implementation von Abstrakten Datentypen

- Verwendung der eben genannten Grundstrukturen
- Verwendung anderer Abstrakter Datentypen

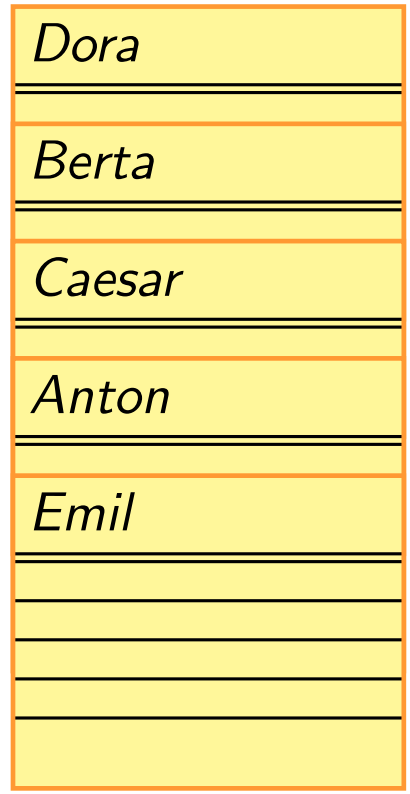


Sortierung

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

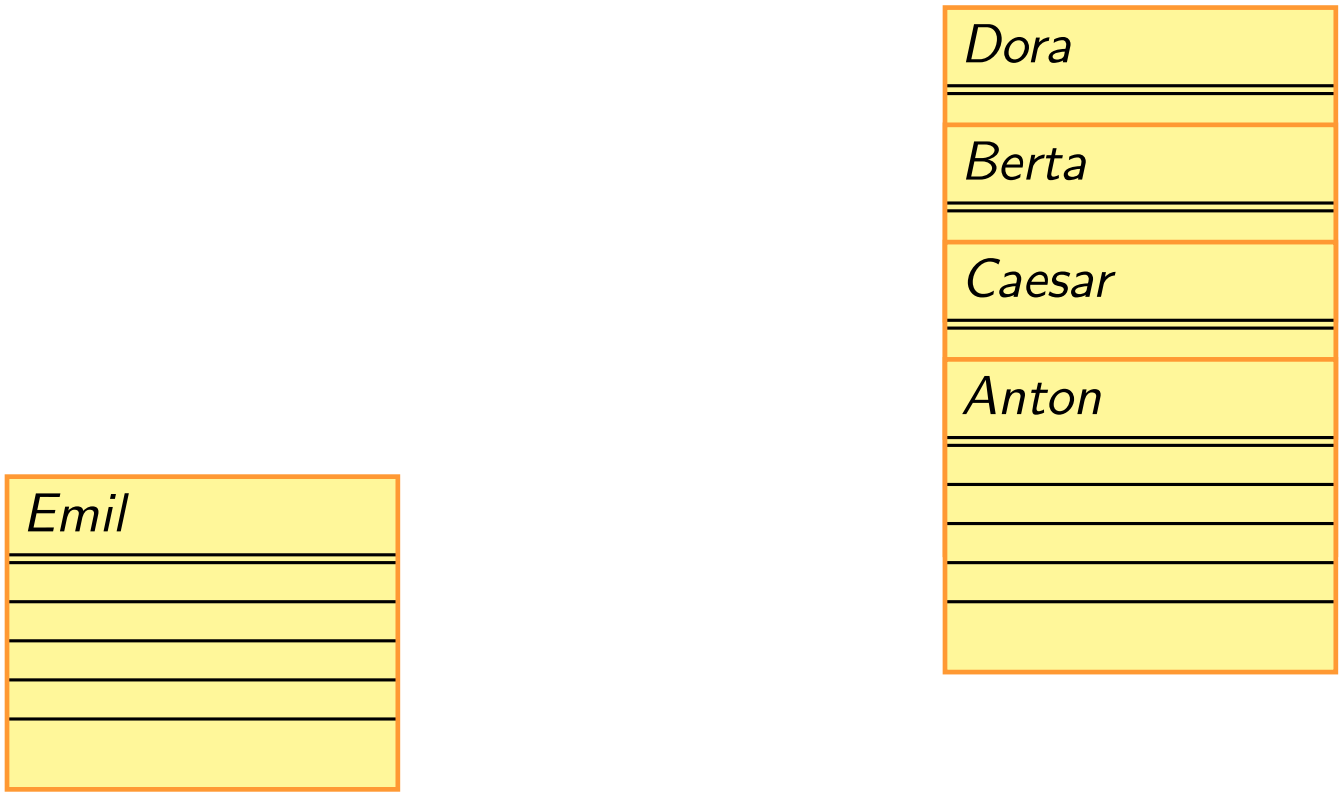


Insertion Sort



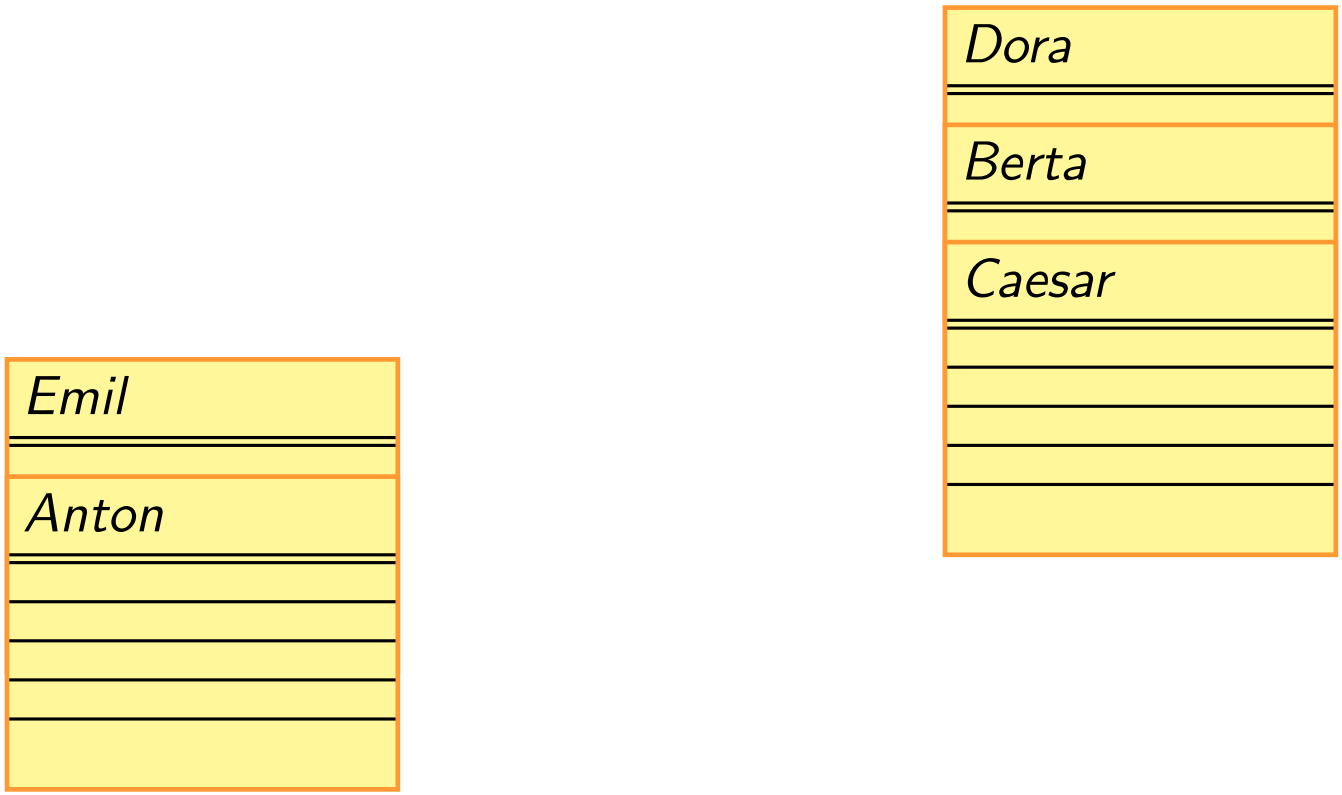
- Gustav nimmt die oberste Karte von *Emil*, und eröffnet damit den *sortierten* Stoss

Insertion Sort



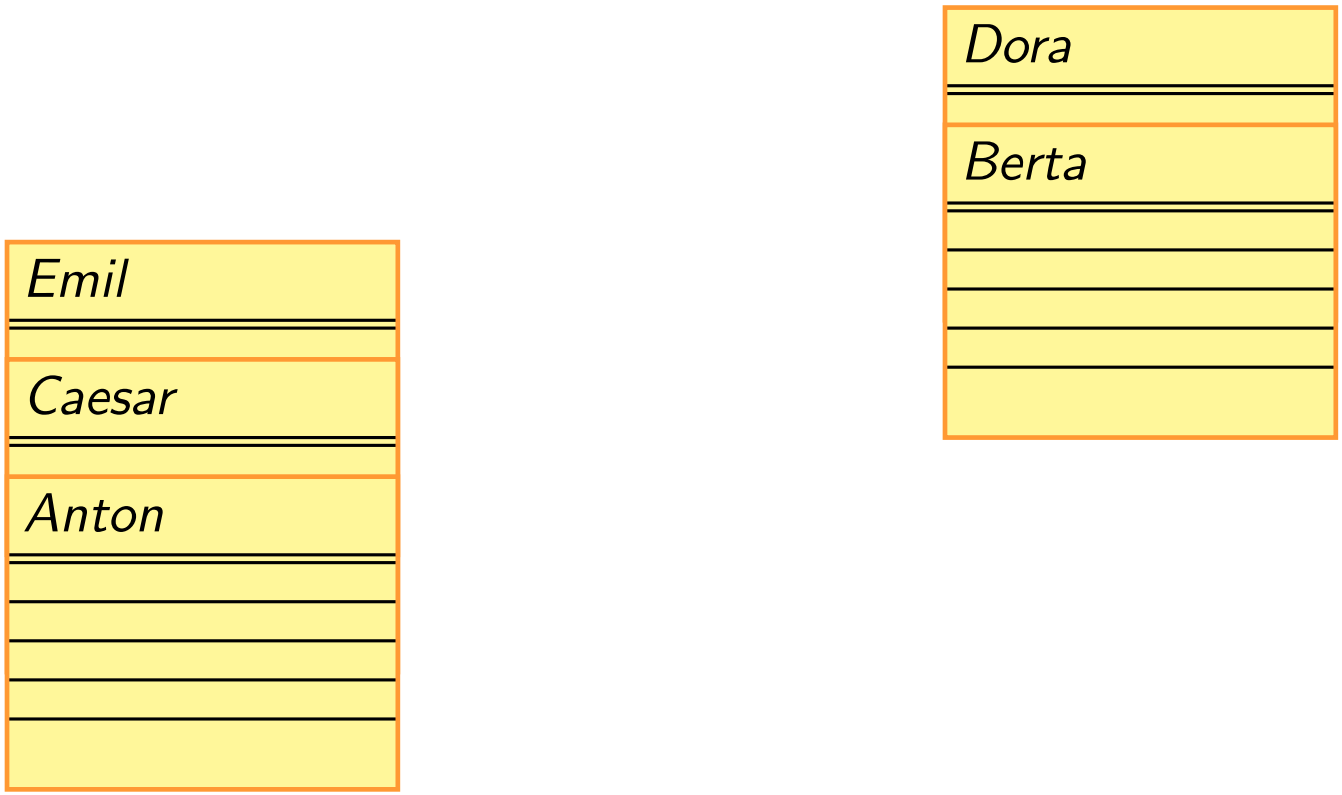
- Gustav nimmt die Karte von *Anton*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort



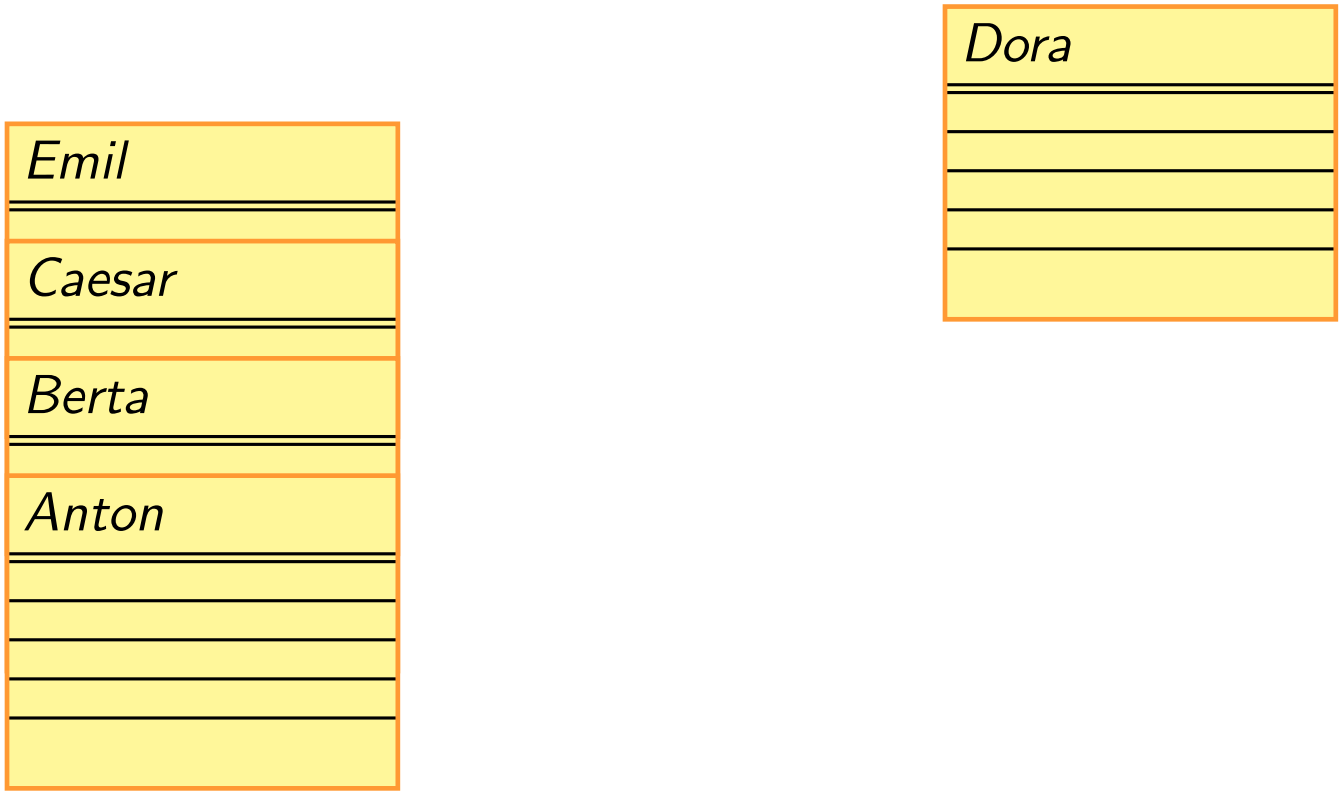
- Gustav nimmt die Karte von *Caesar*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort



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

Insertion Sort

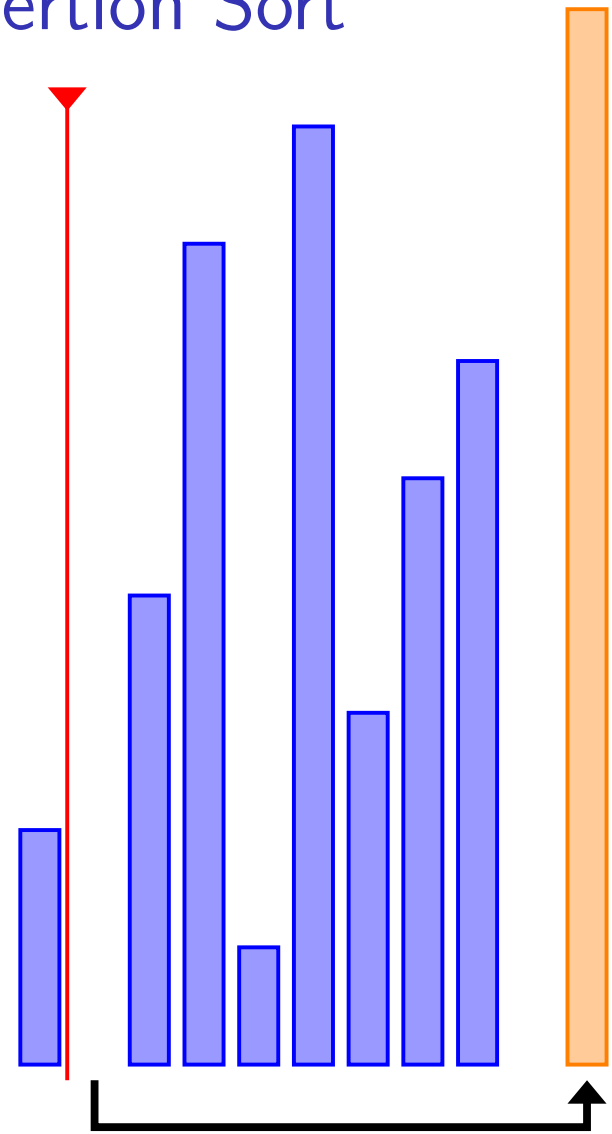


- Gustav nimmt die Karte von *Dora*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort

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

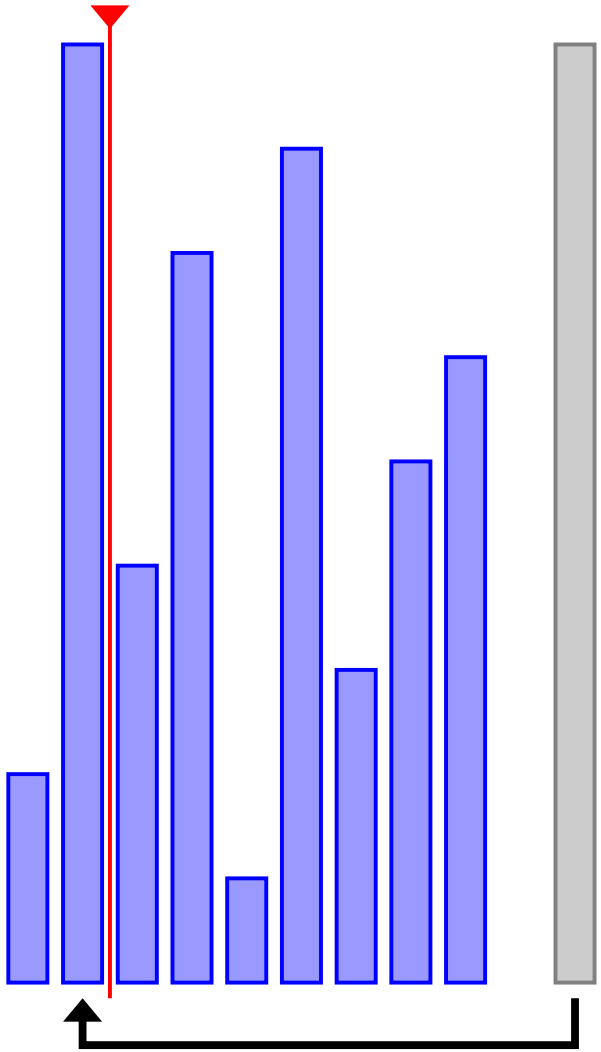
Insertion Sort



$j = 1$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

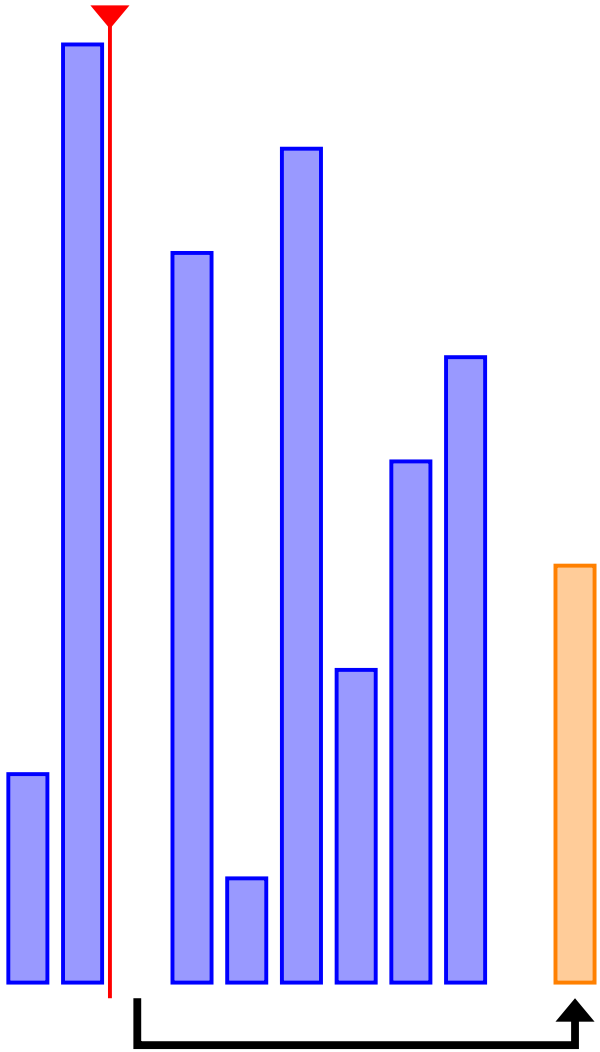
Insertion Sort



$j = 1$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

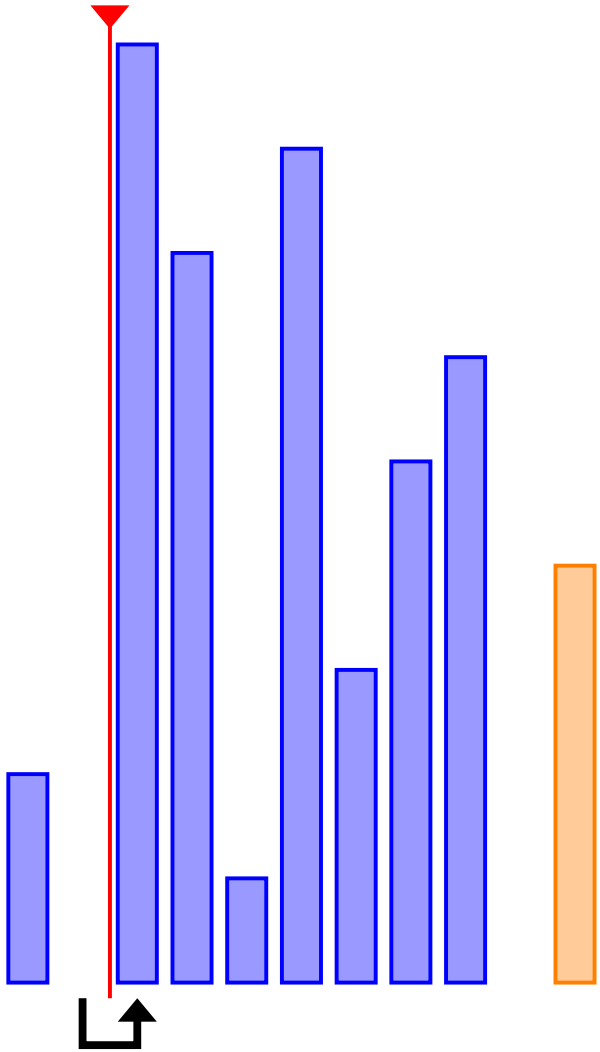
Insertion Sort



$j = 2$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

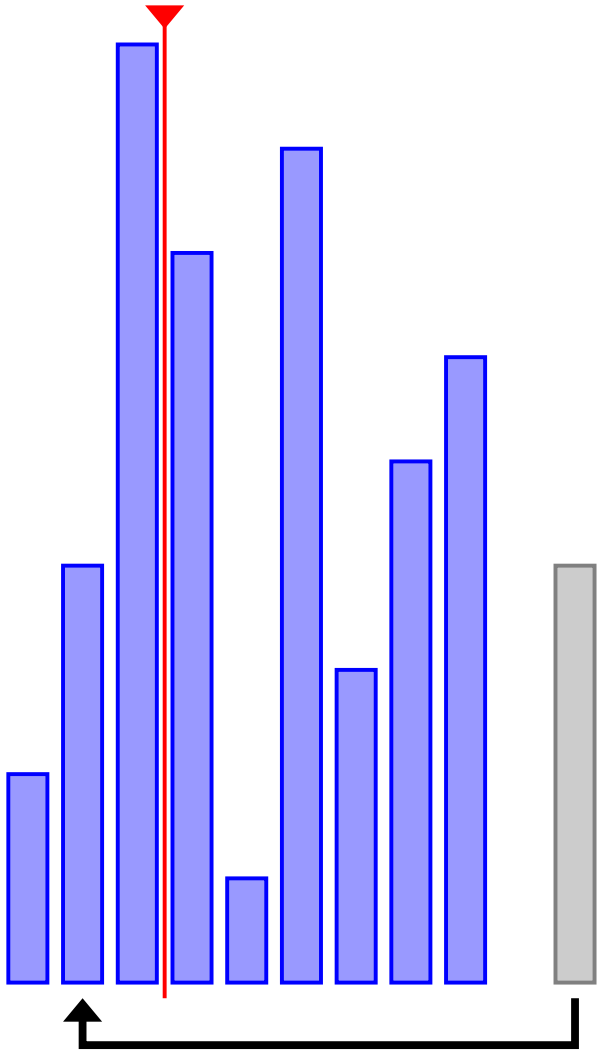
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

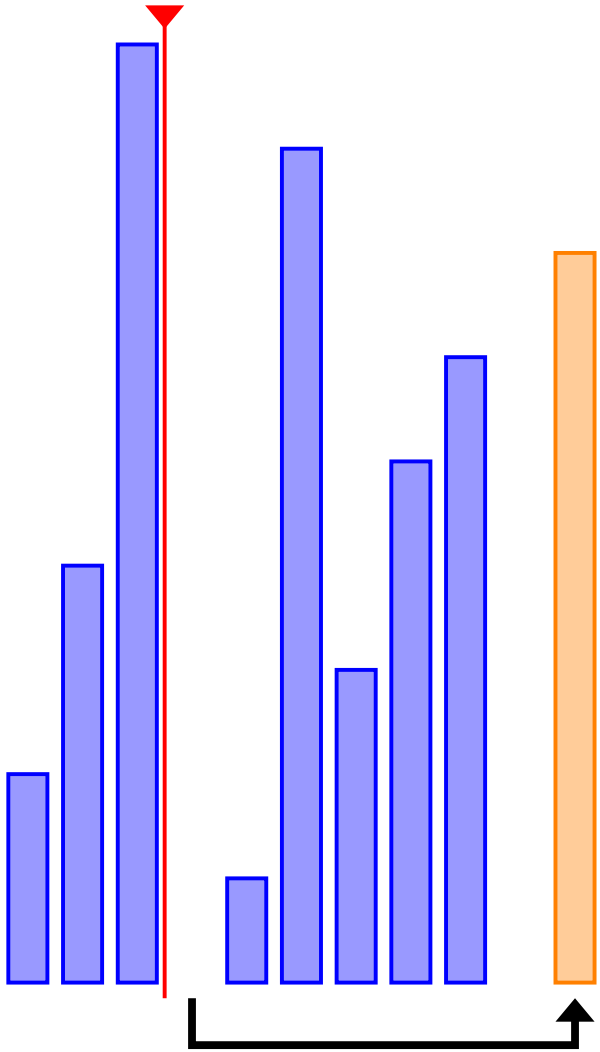
Insertion Sort



$j = 2$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

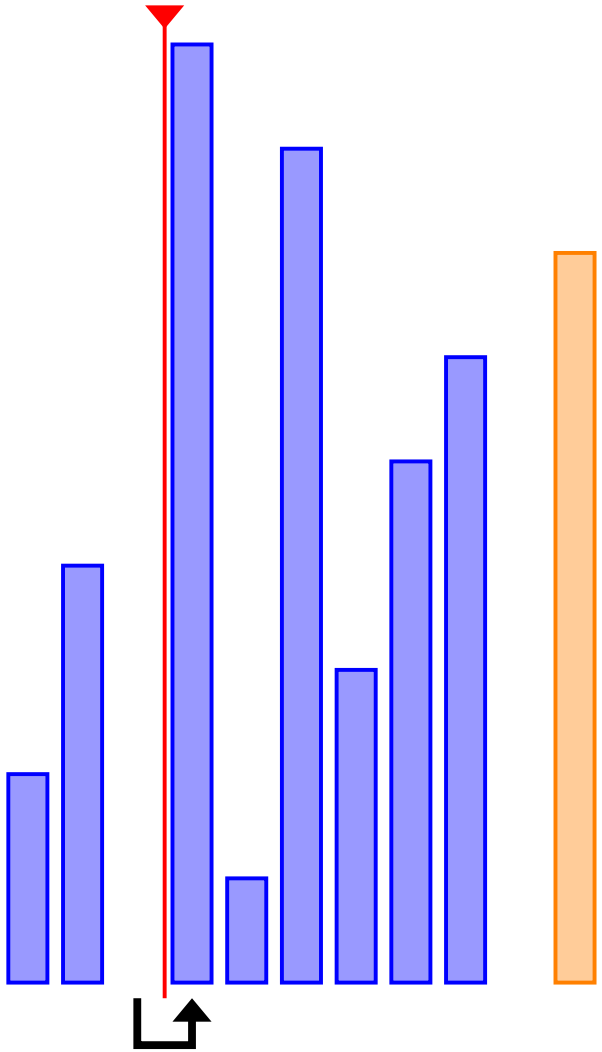
Insertion Sort



$j = 3$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

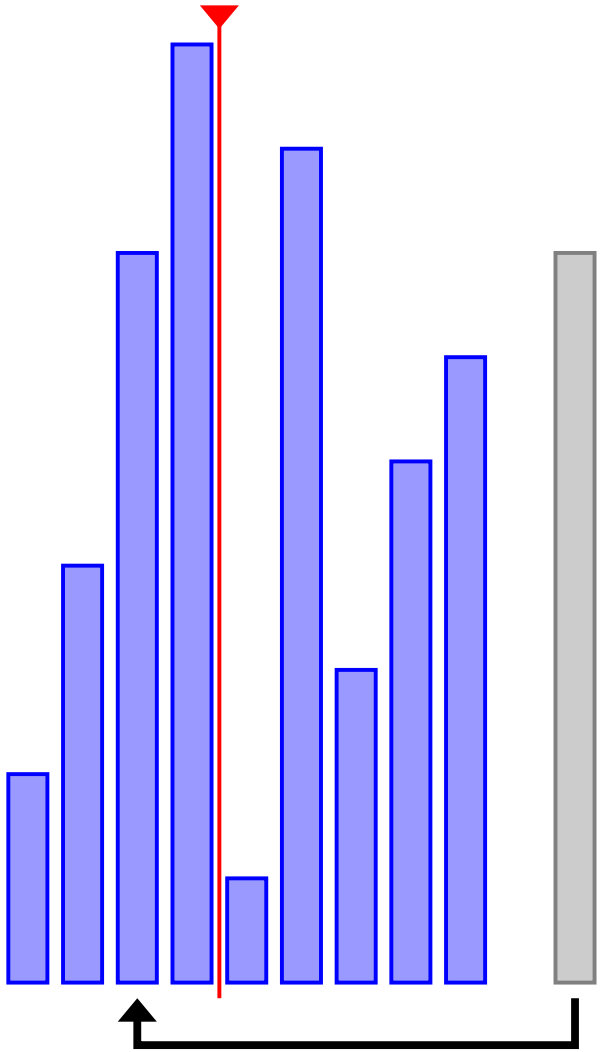
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

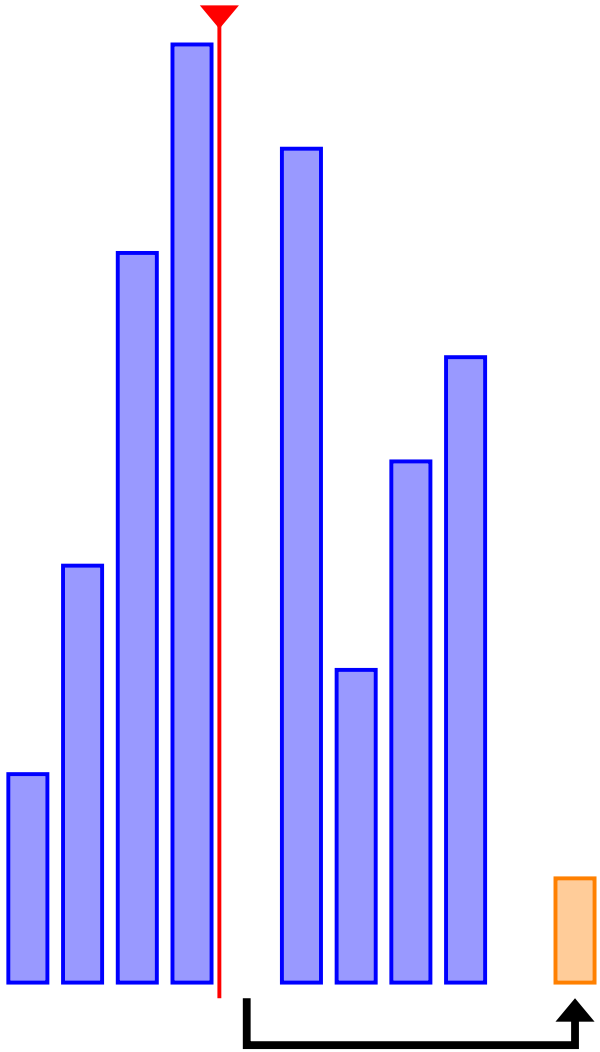
Insertion Sort



$j = 3$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

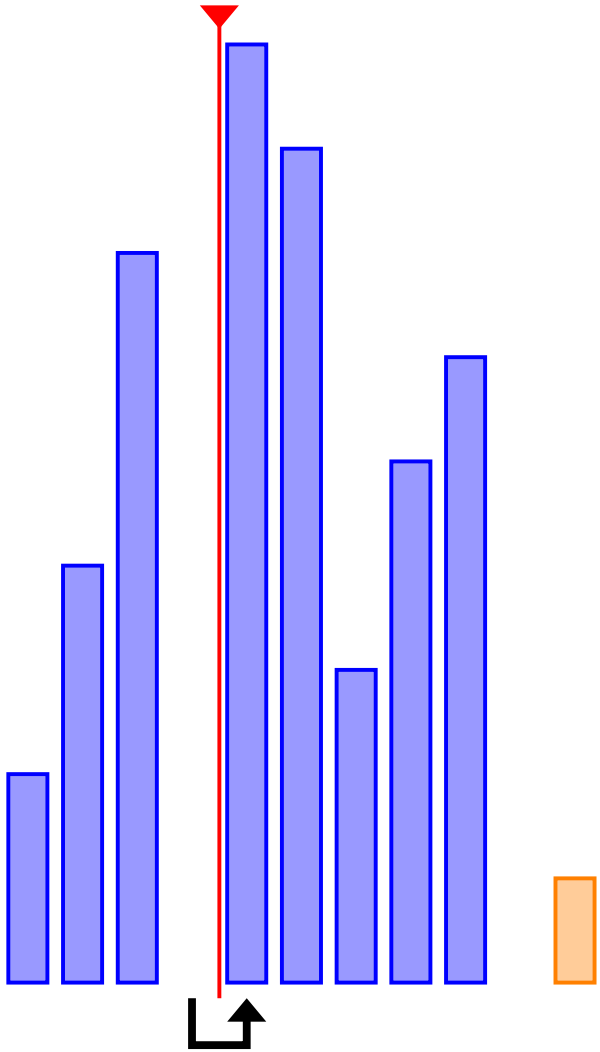
Insertion Sort



$j = 4$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

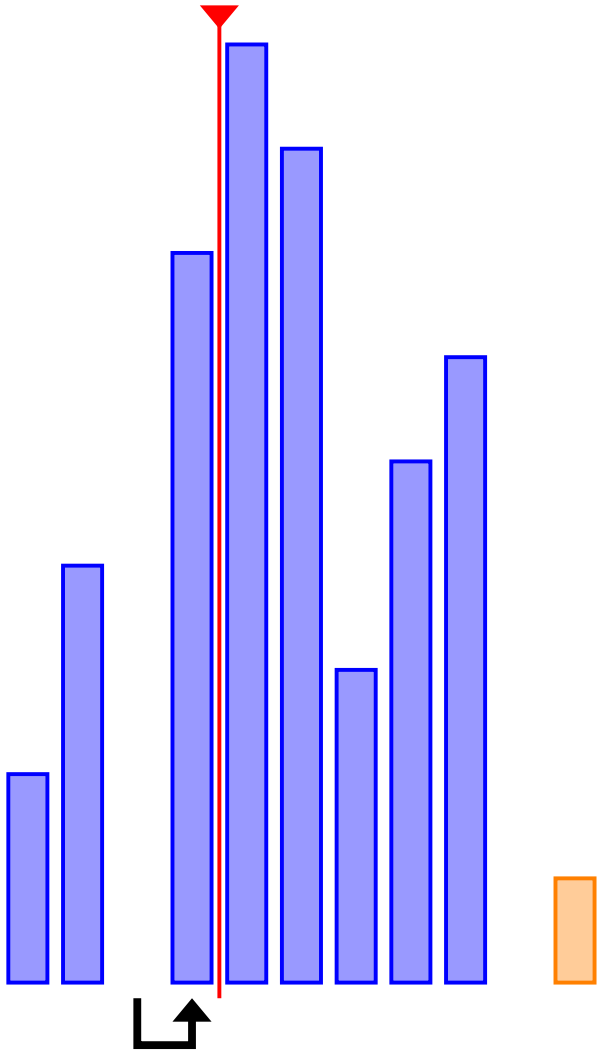
Insertion Sort



$j = 4$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

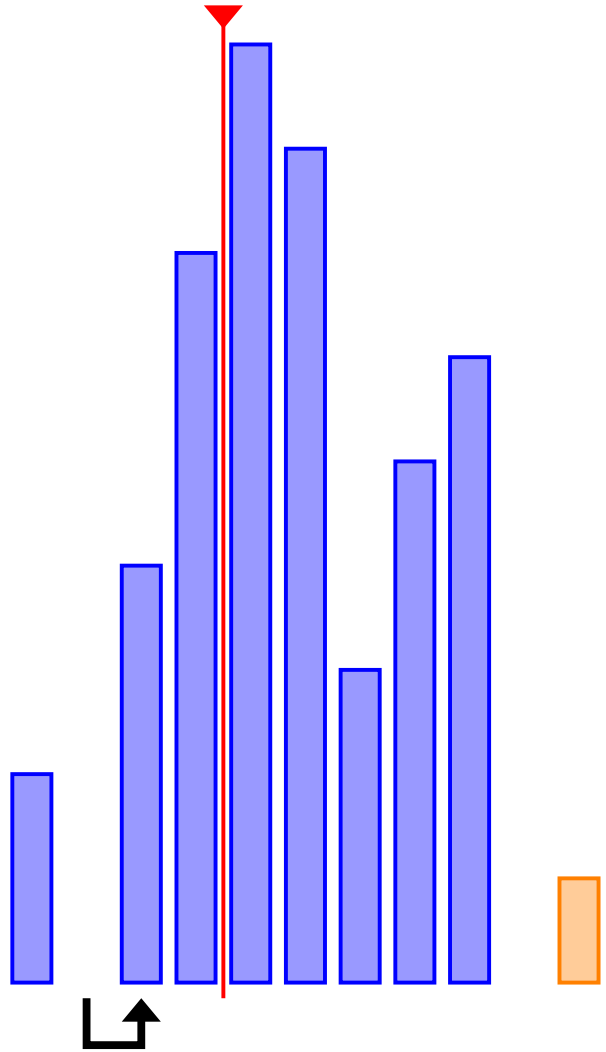
Insertion Sort



$j = 4$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

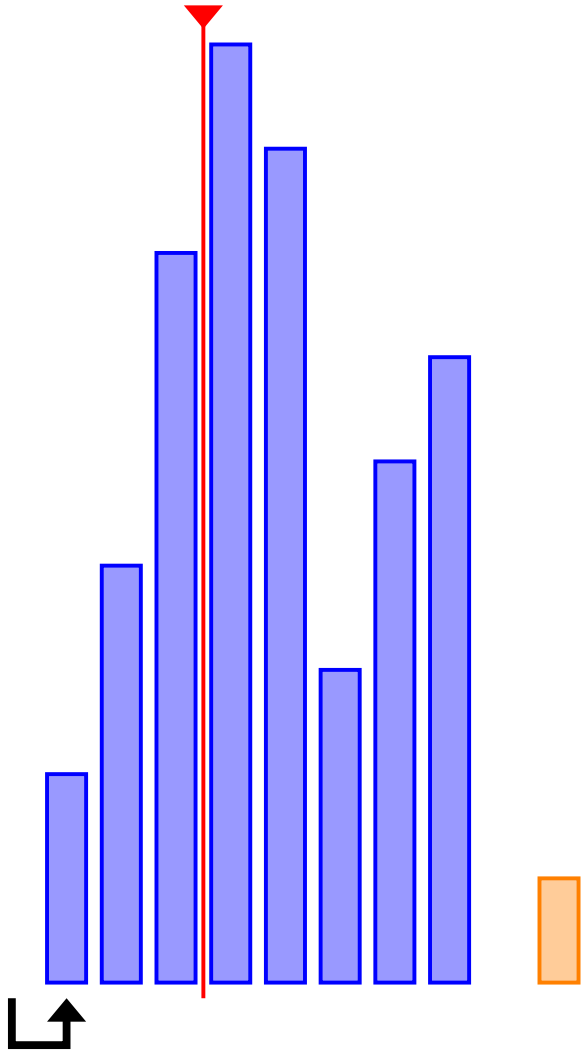
Insertion Sort



$j = 4$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

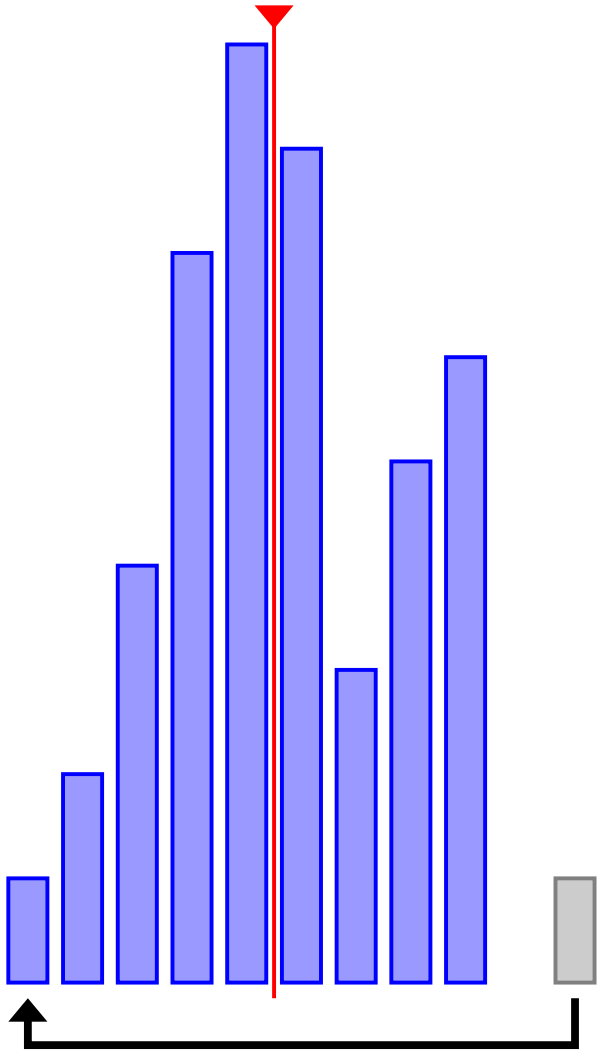
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

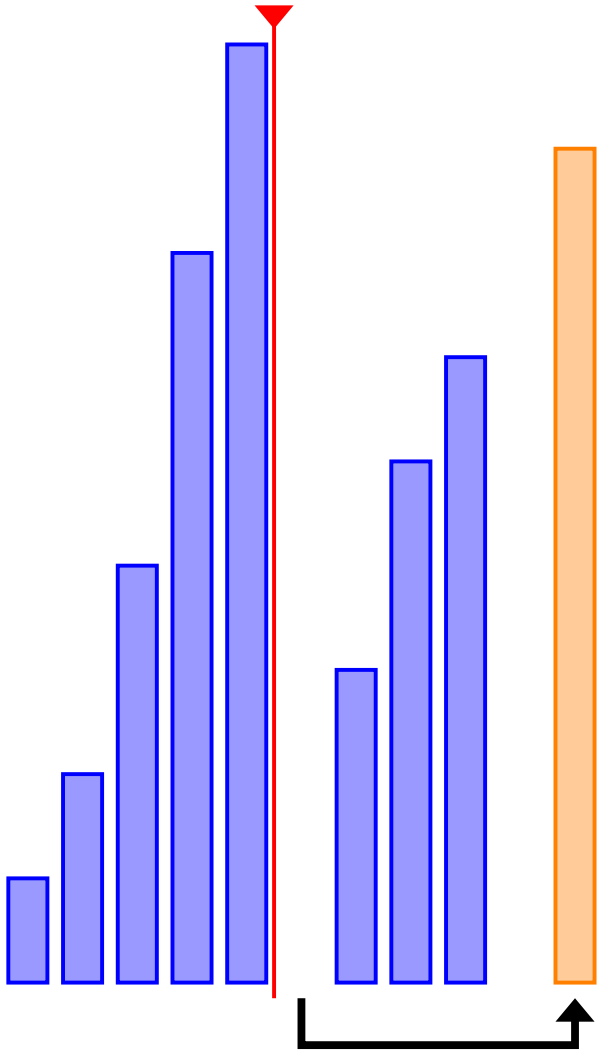
Insertion Sort



$j = 4$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

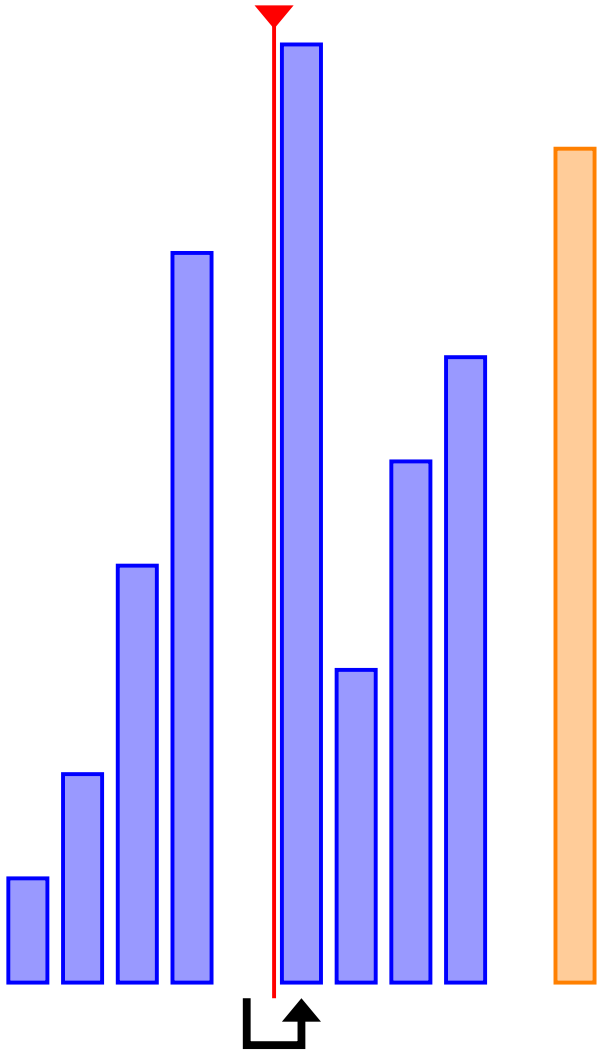
Insertion Sort



$j = 5$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

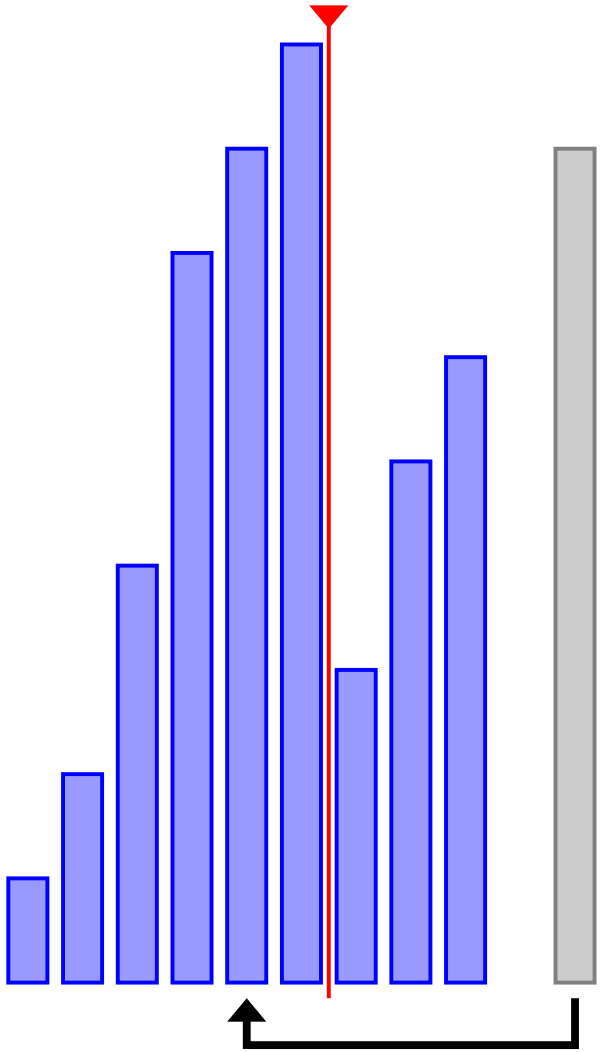
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

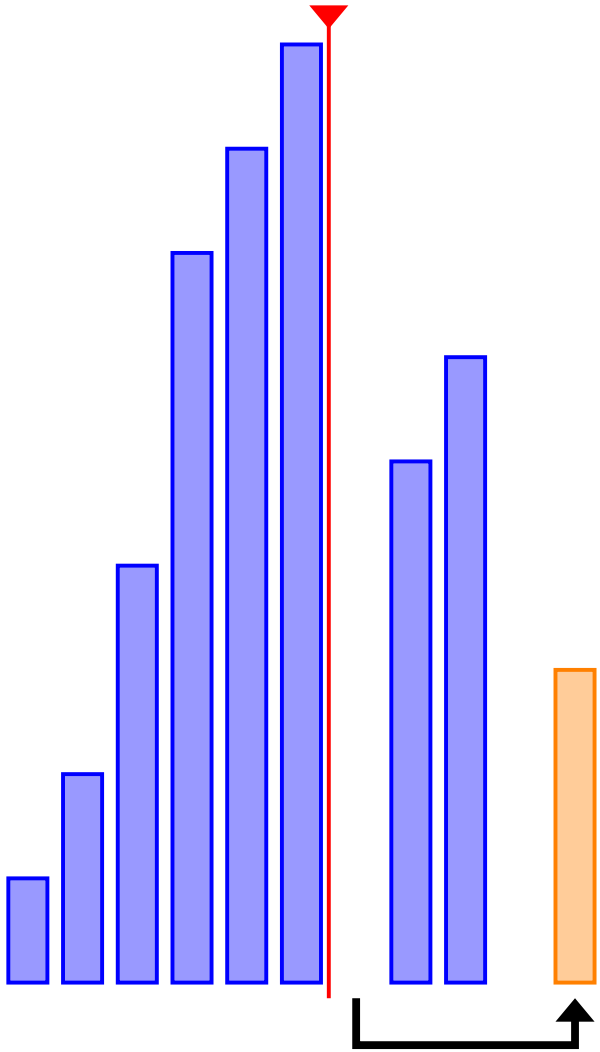
Insertion Sort



$j = 5$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

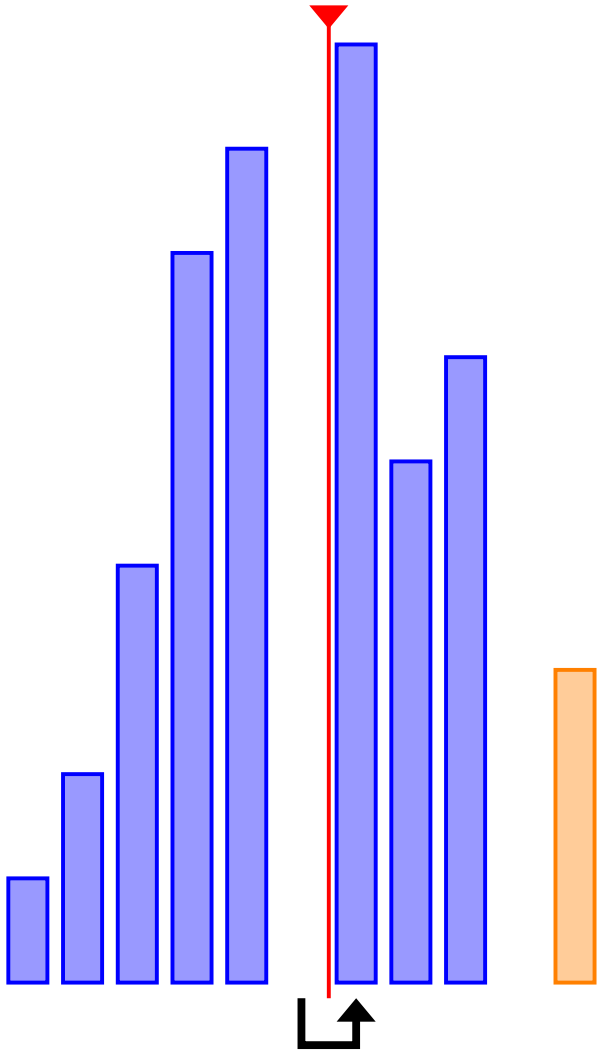
Insertion Sort



$j = 6$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

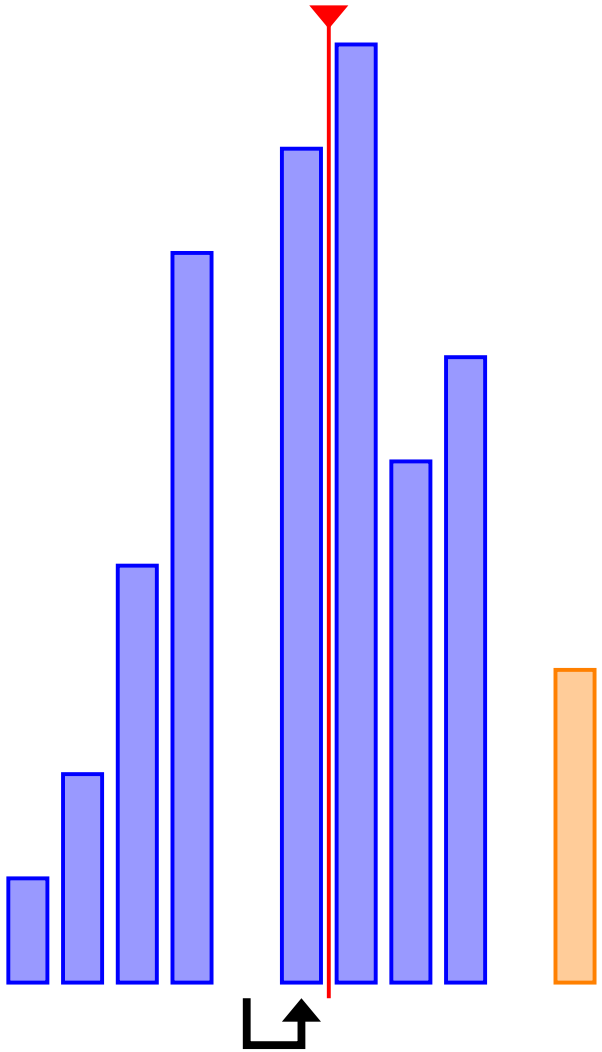
Insertion Sort



$j = 6$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

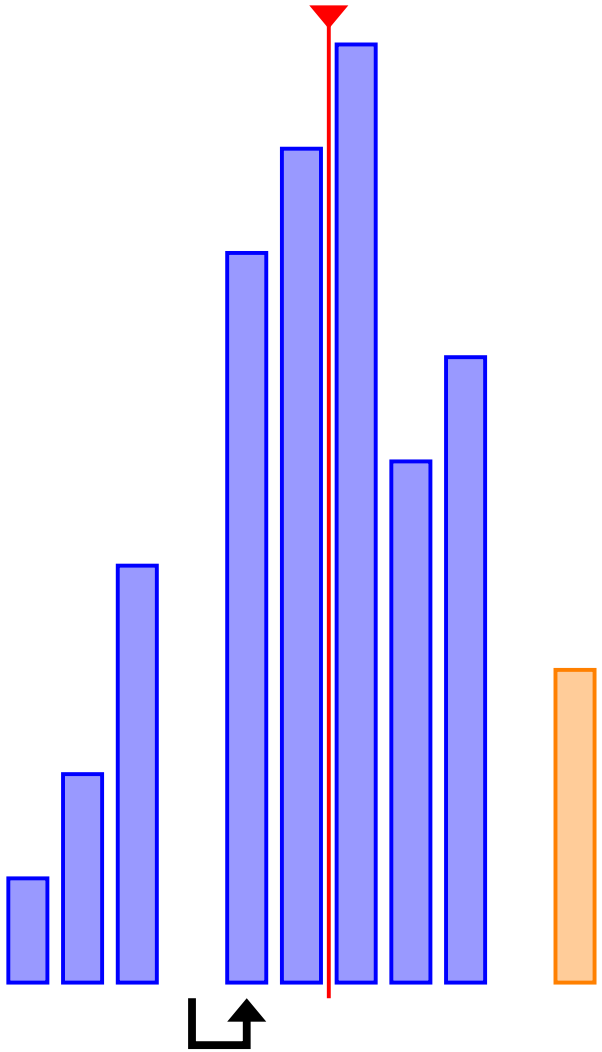
Insertion Sort



$j = 6$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

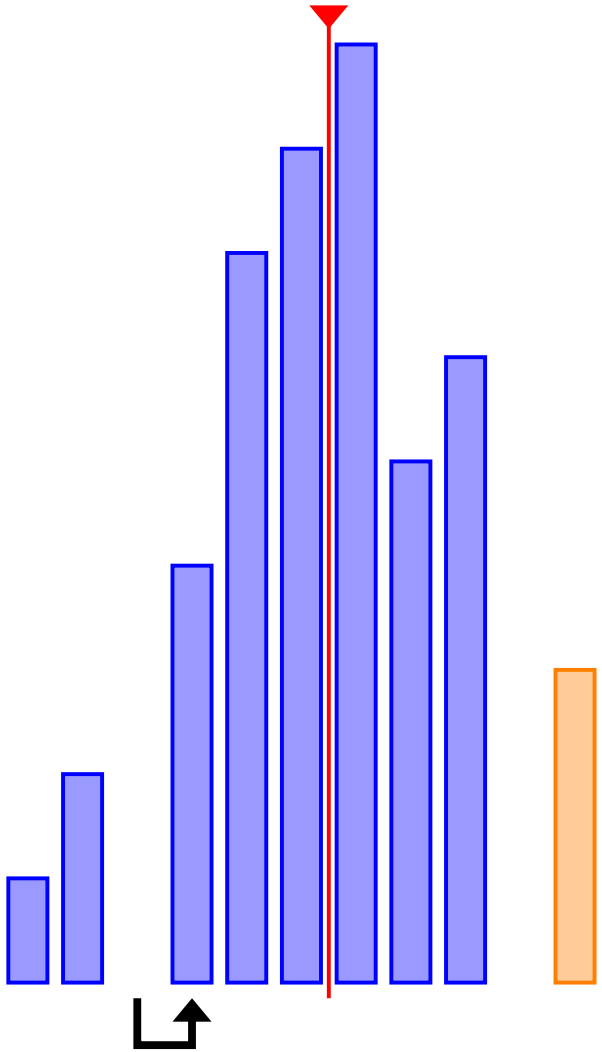
Insertion Sort



$j = 6$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

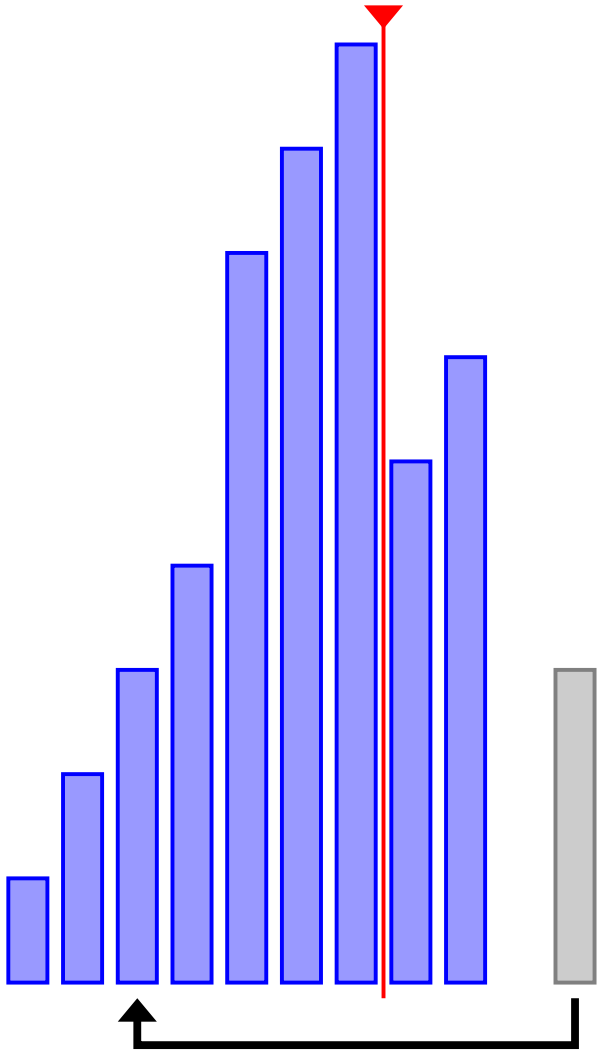
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

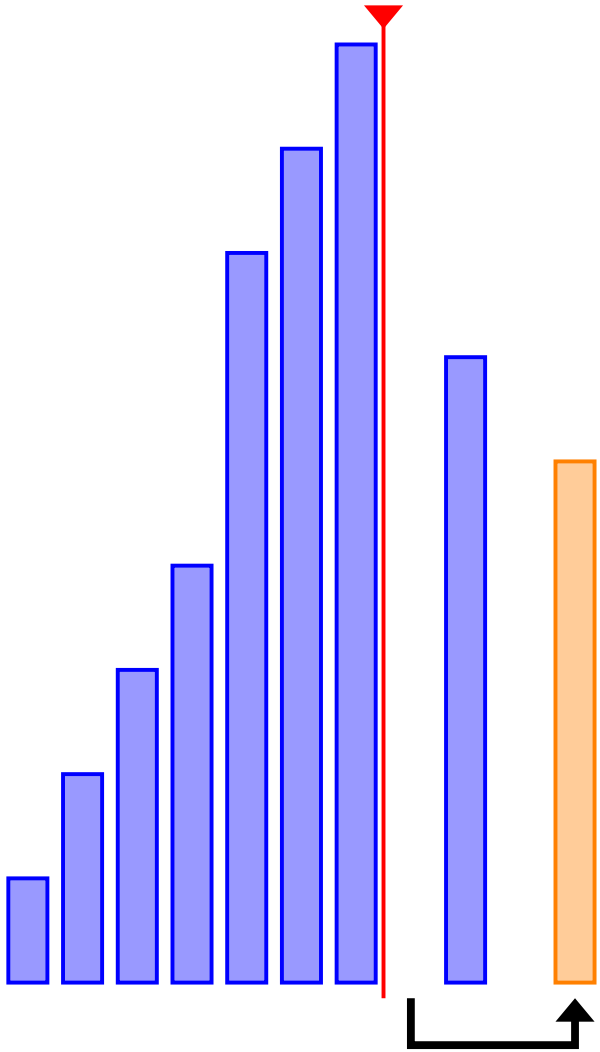
Insertion Sort



$j = 6$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

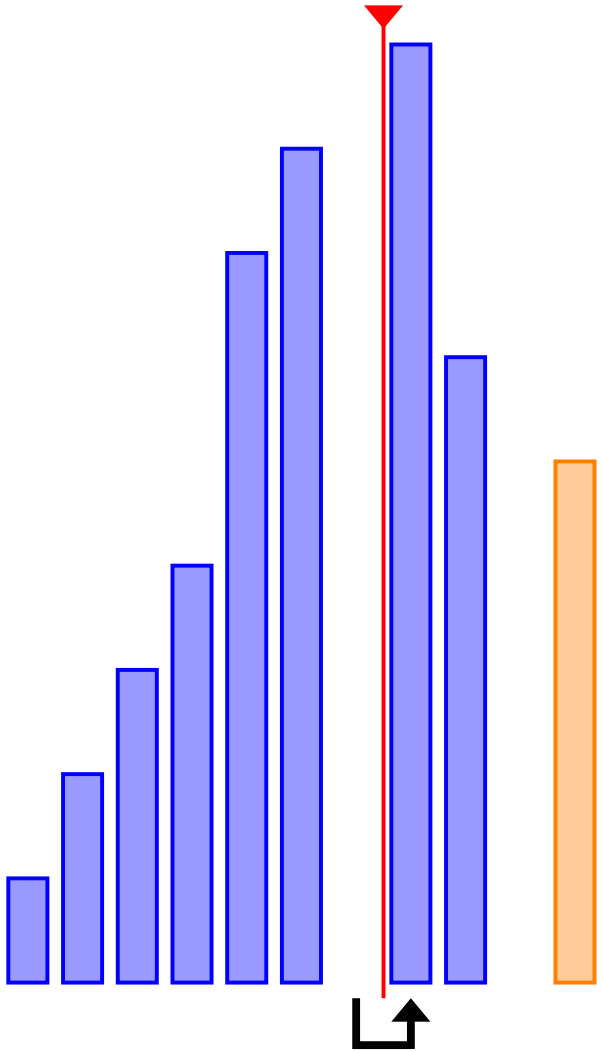
Insertion Sort



$j = 7$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

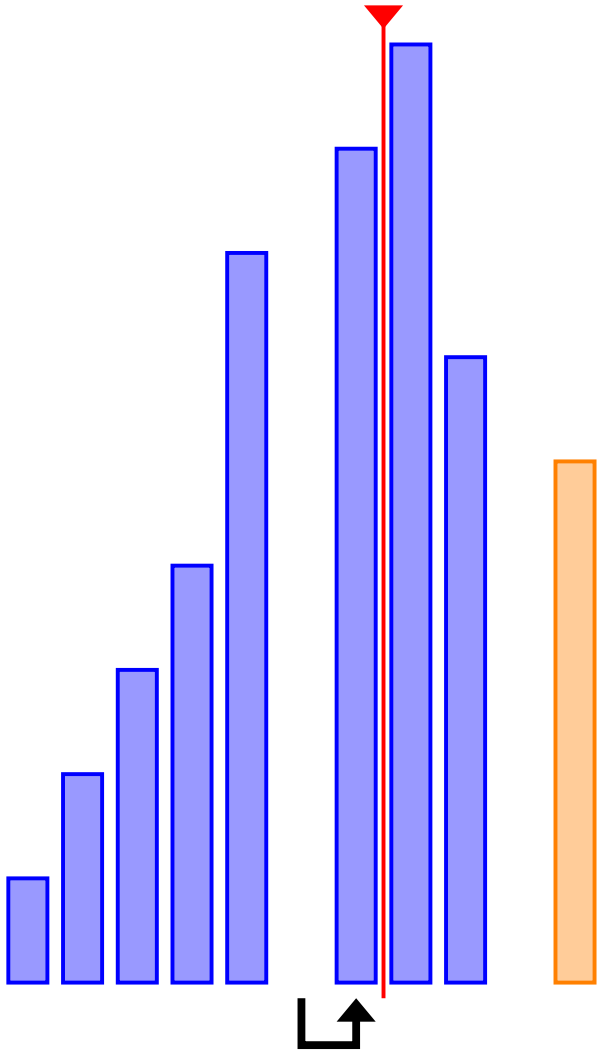
Insertion Sort



$j = 7$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

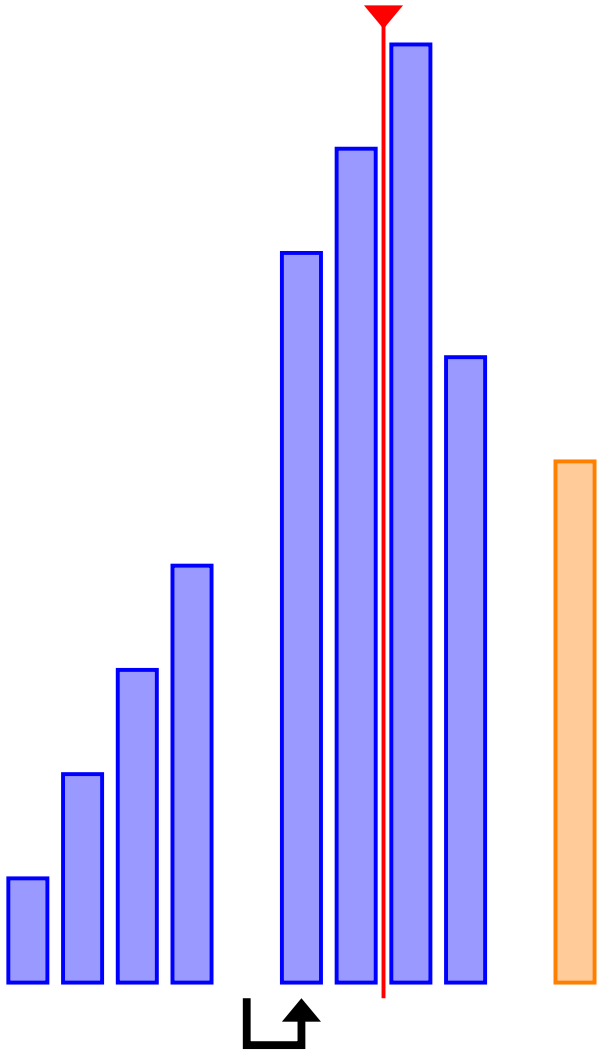
Insertion Sort



$j = 7$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

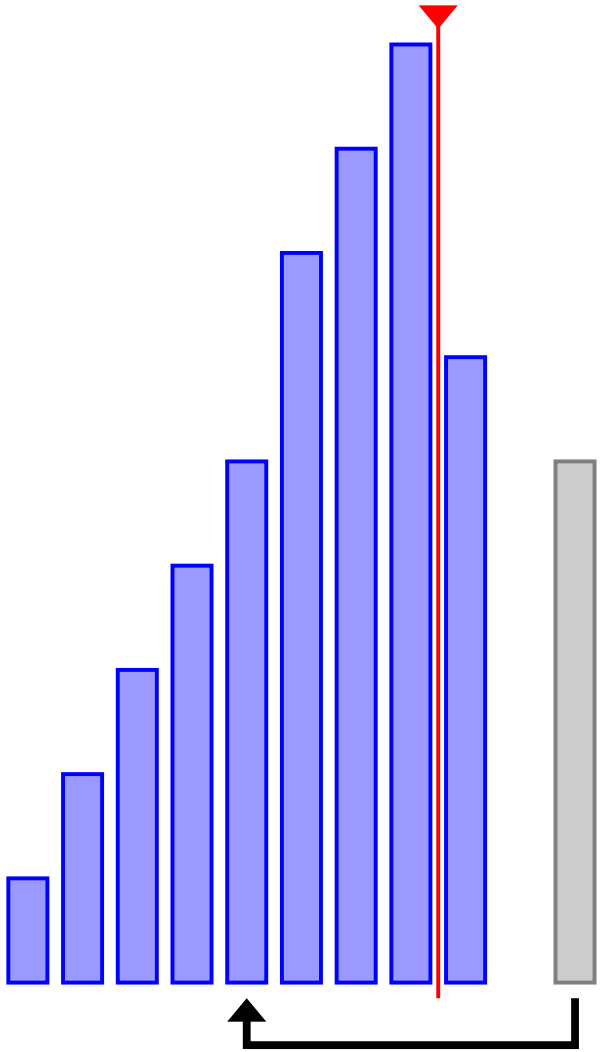
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

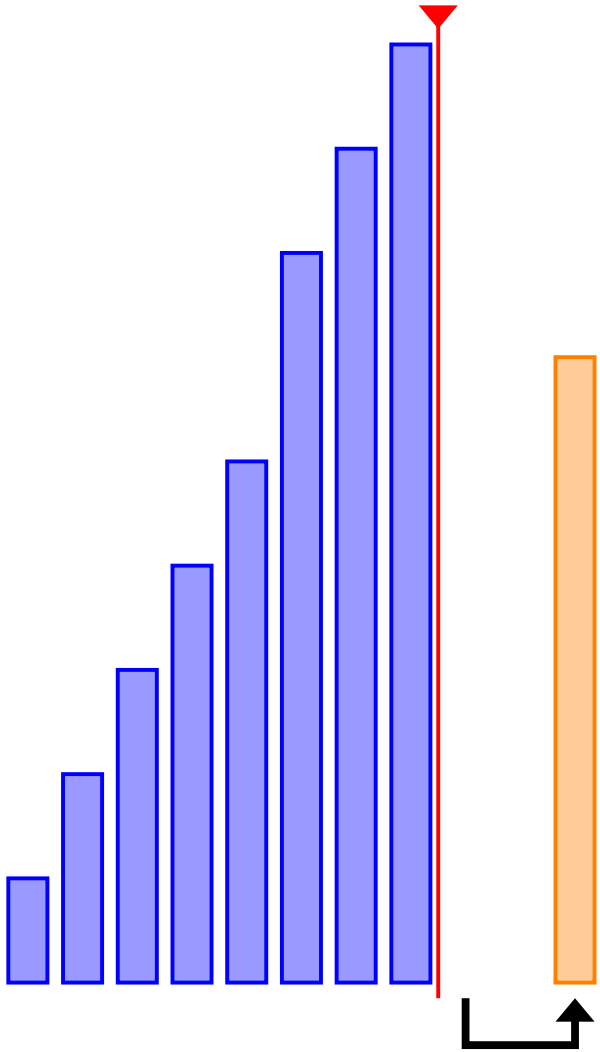
Insertion Sort



$j = 7$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

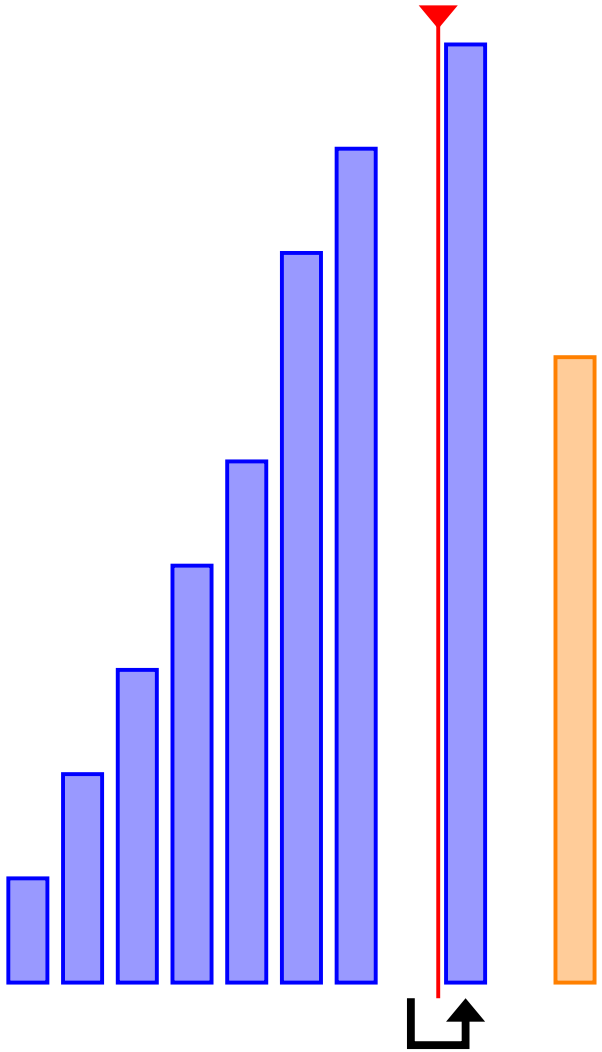
Insertion Sort



$j = 8$: Anfangszustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

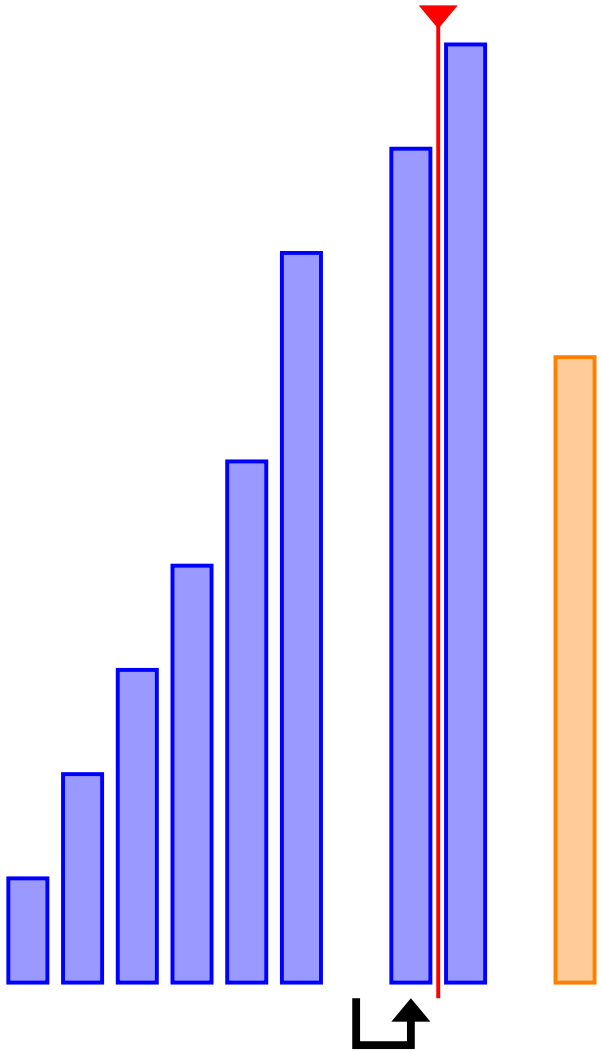
Insertion Sort



$j = 8$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

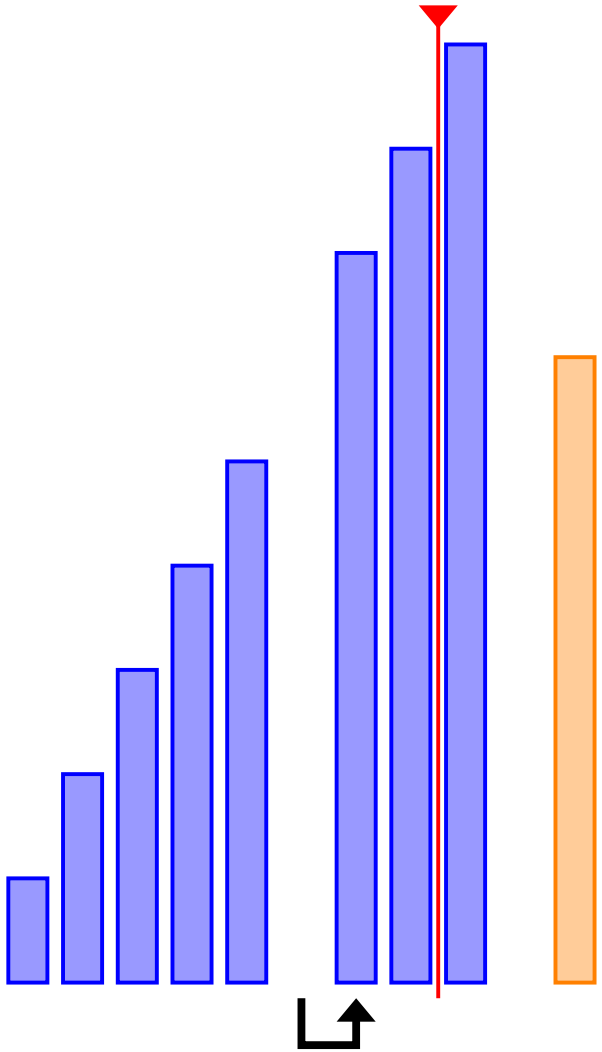
Insertion Sort



$j = 8$: Platz für Einfügung

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

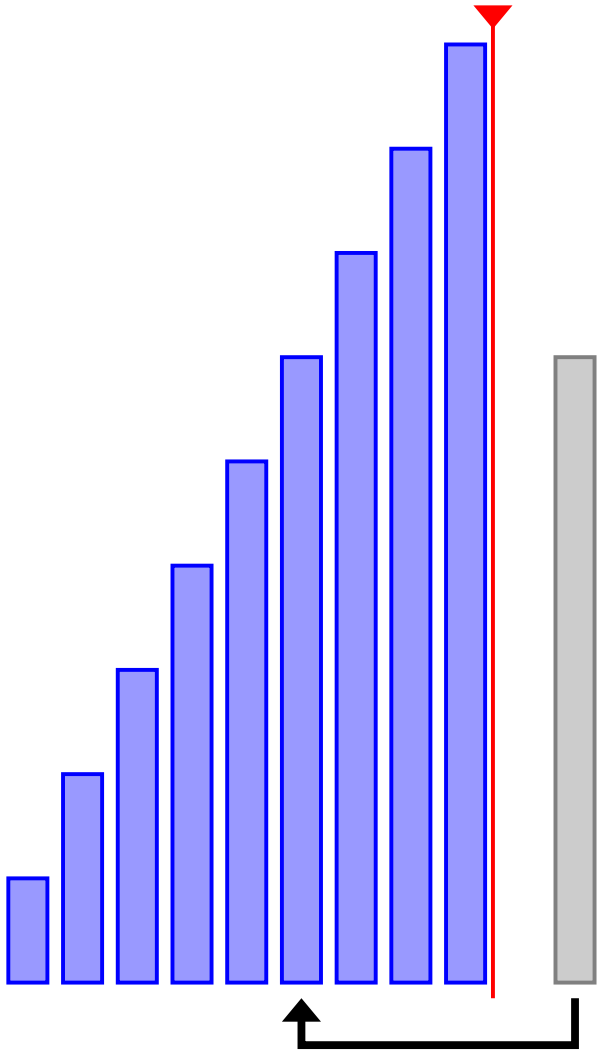
Insertion Sort



```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int 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**

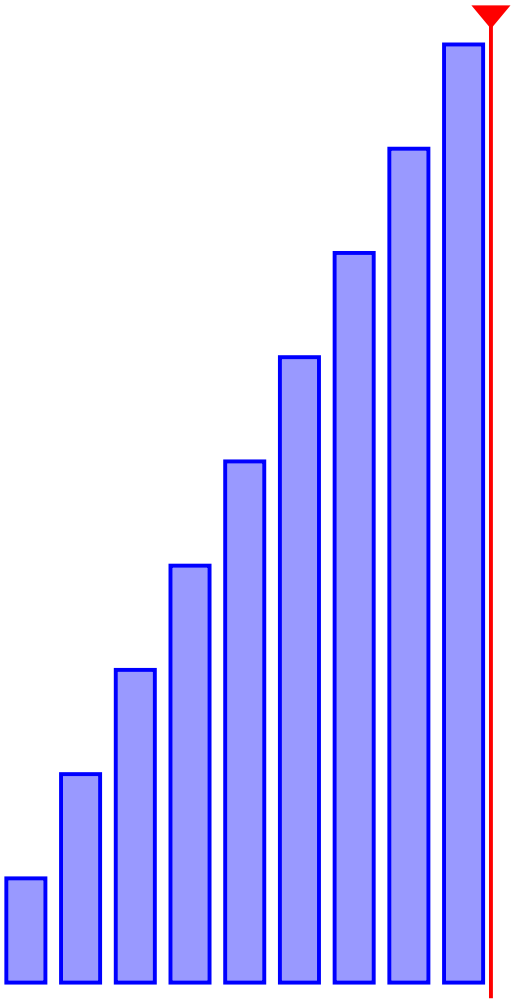
Insertion Sort



$j = 8$: Endzustand

```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

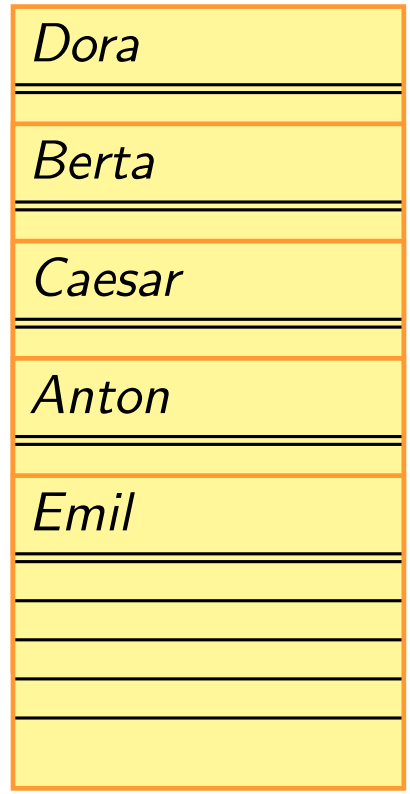
Insertion Sort



Endzustand

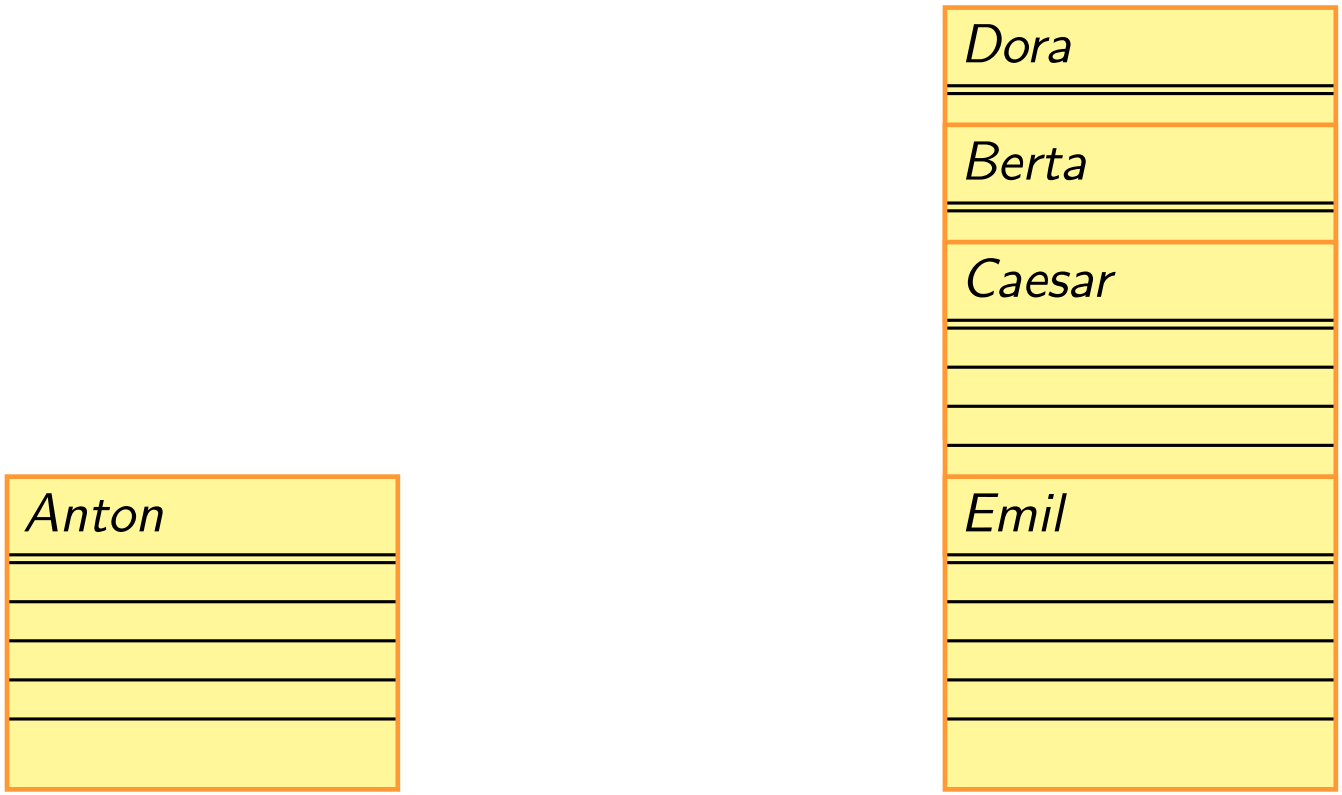
```
void insertion_sort(int* A, int n)
{
        for (int j = 1; j < n; ++j)
        {
                int key = A[j];
                int i = j - 1;
                while (i >= 0 && A[i] > key)
                {
                        A[i + 1] = A[i];
                        i = i - 1;
                }
                A[i+1] = key;
        }
}
```

Selection Sort



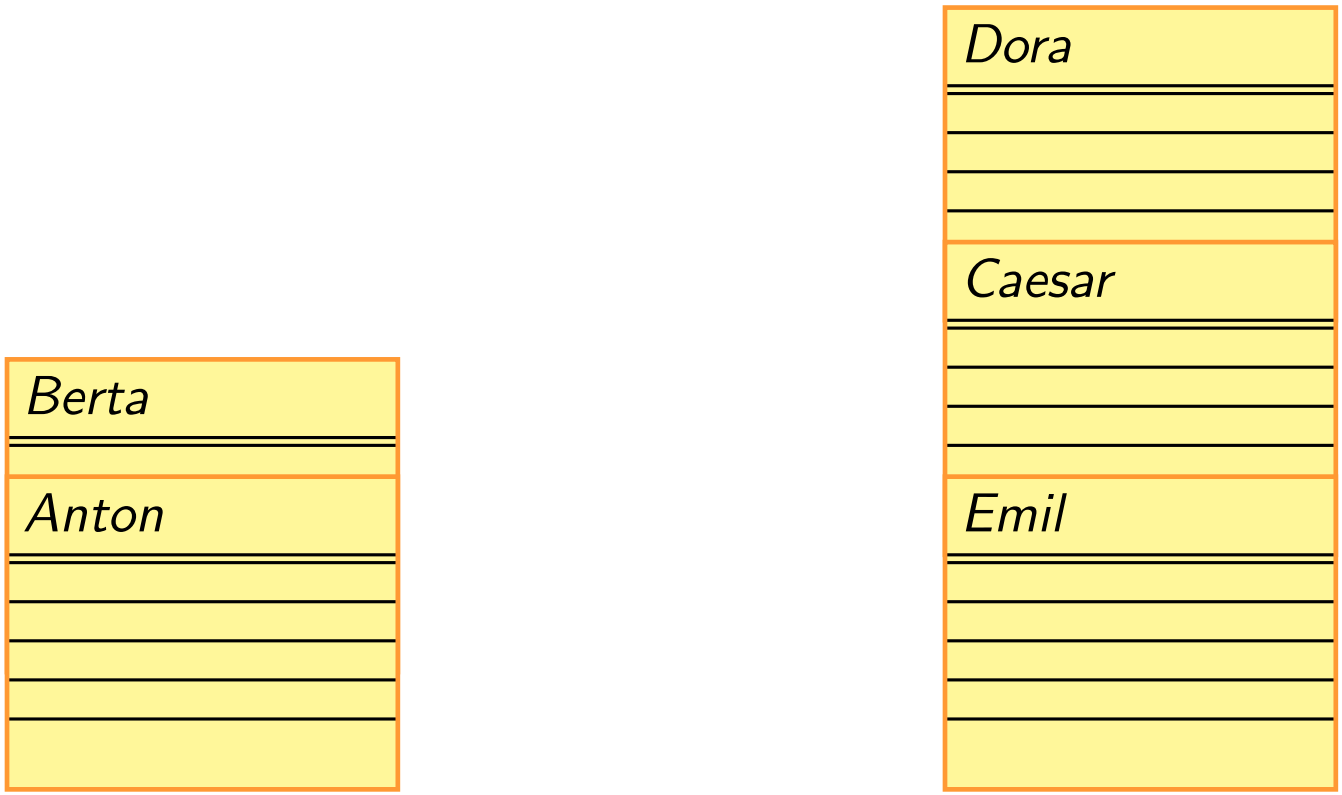
- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Anton*

Selection Sort



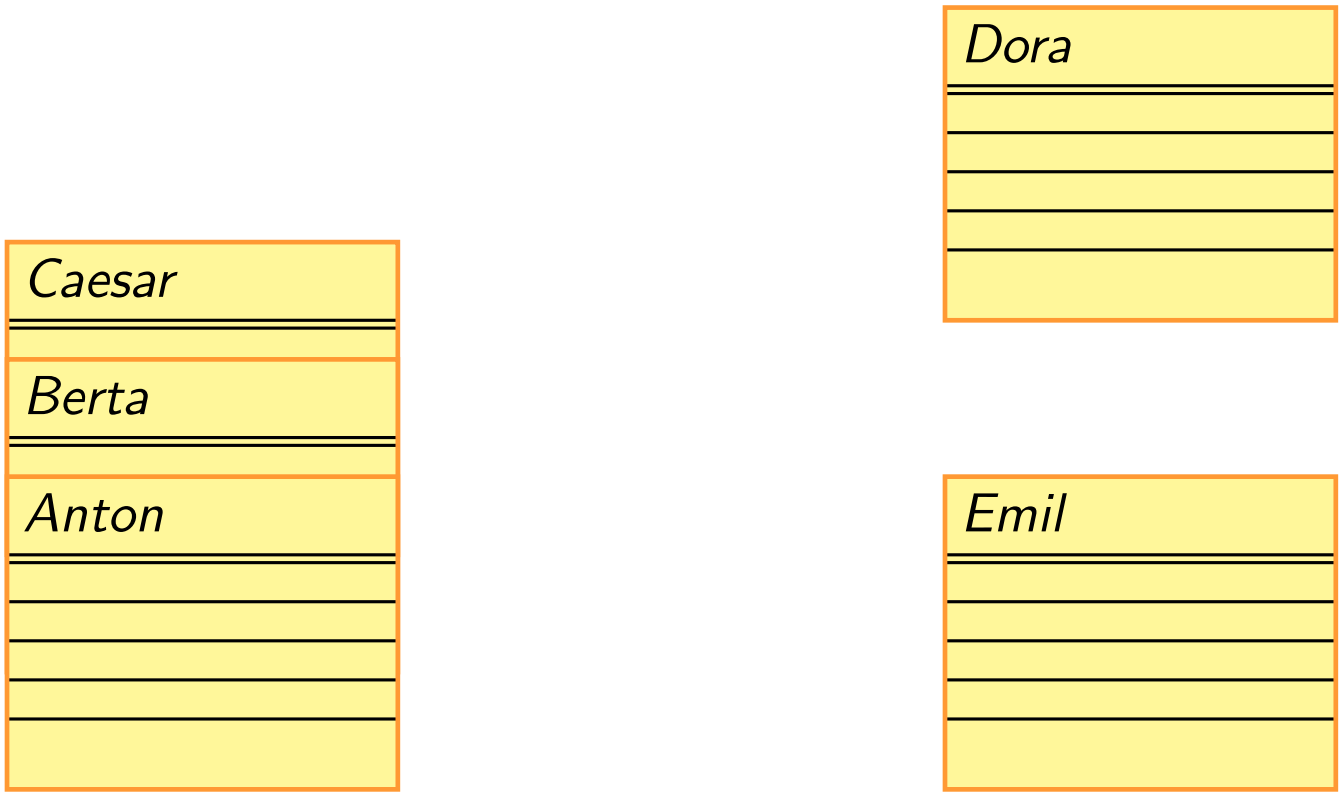
- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Berta*

Selection Sort



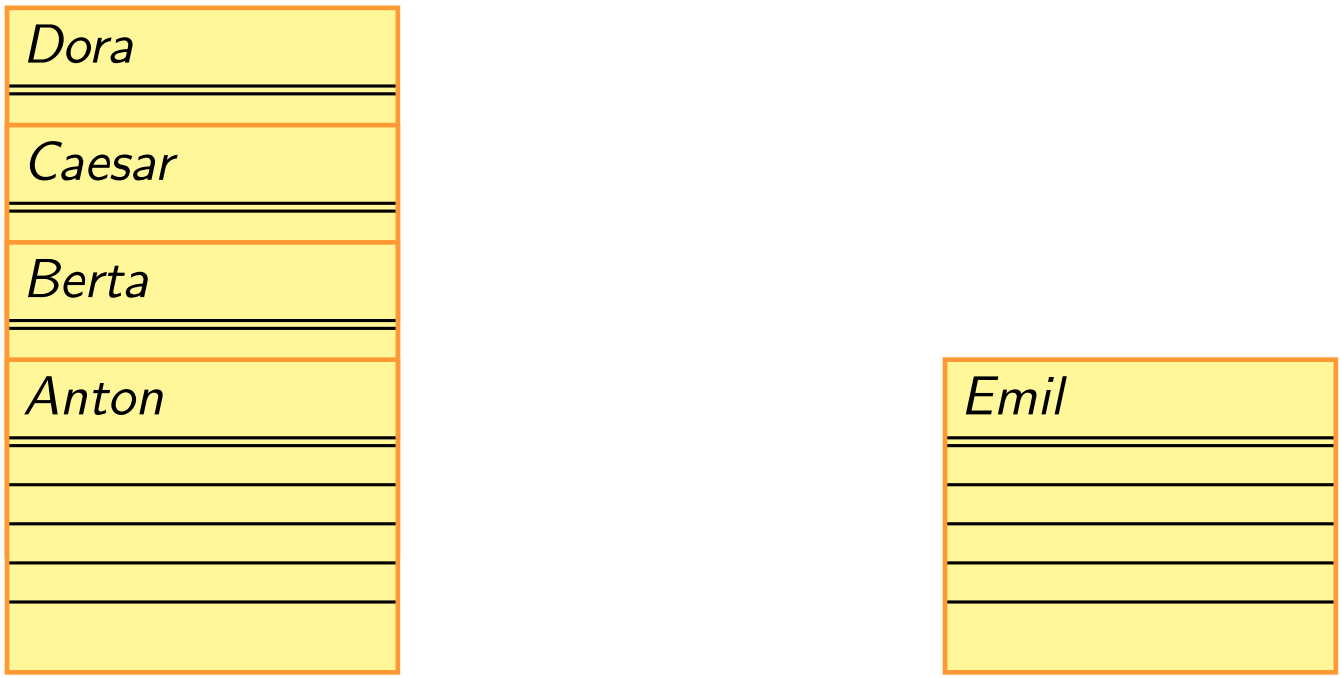
- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Caesar*

Selection Sort



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

Selection Sort

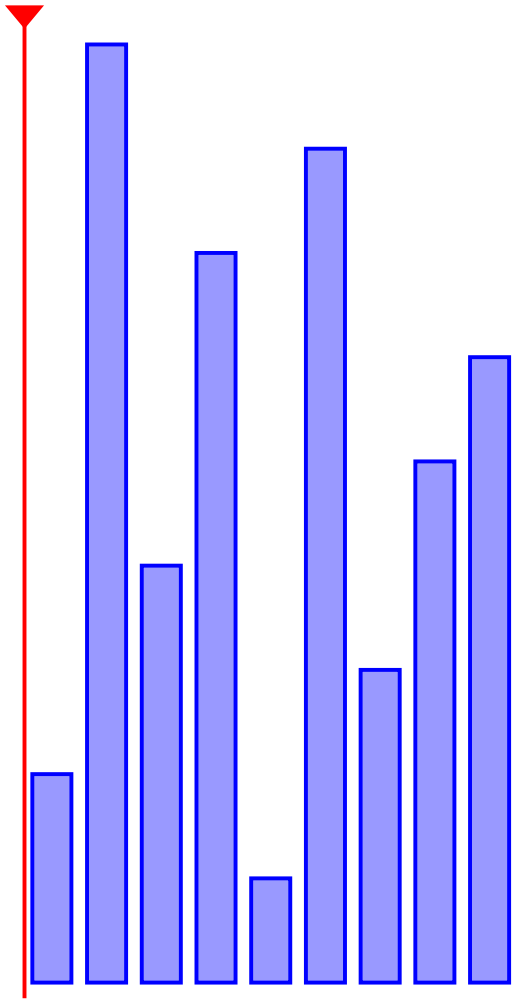


- Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Emil*

Selection Sort

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

Selection Sort

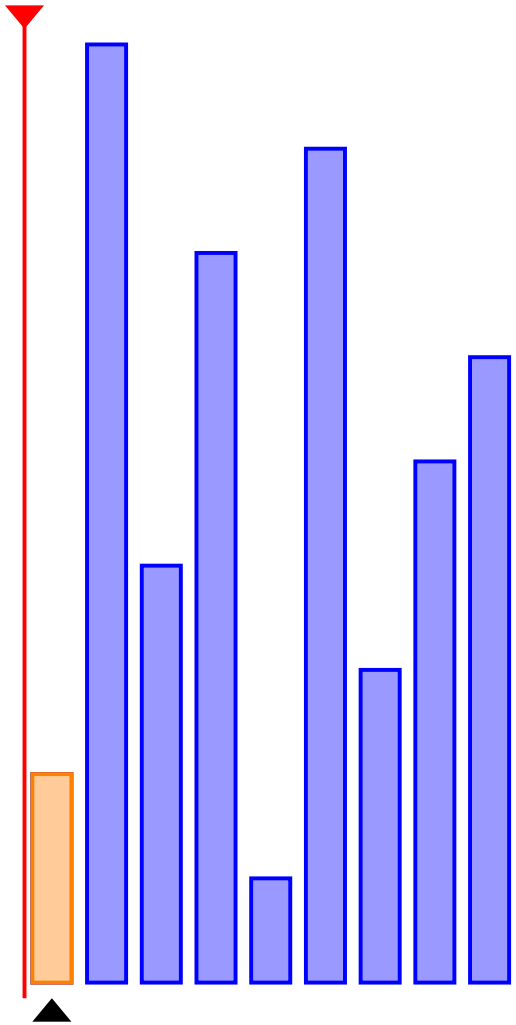


$j = 0$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

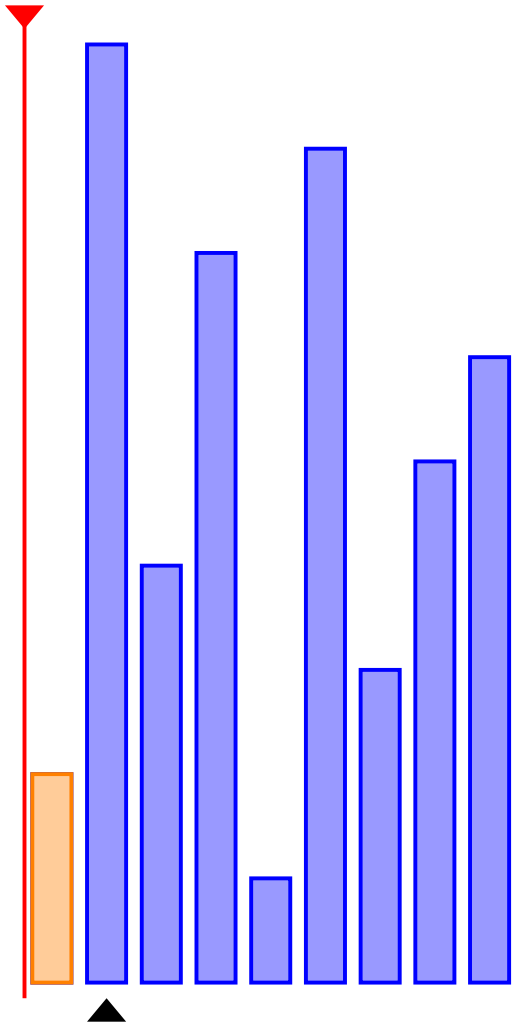


$j = 0$: Suche bei 0

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

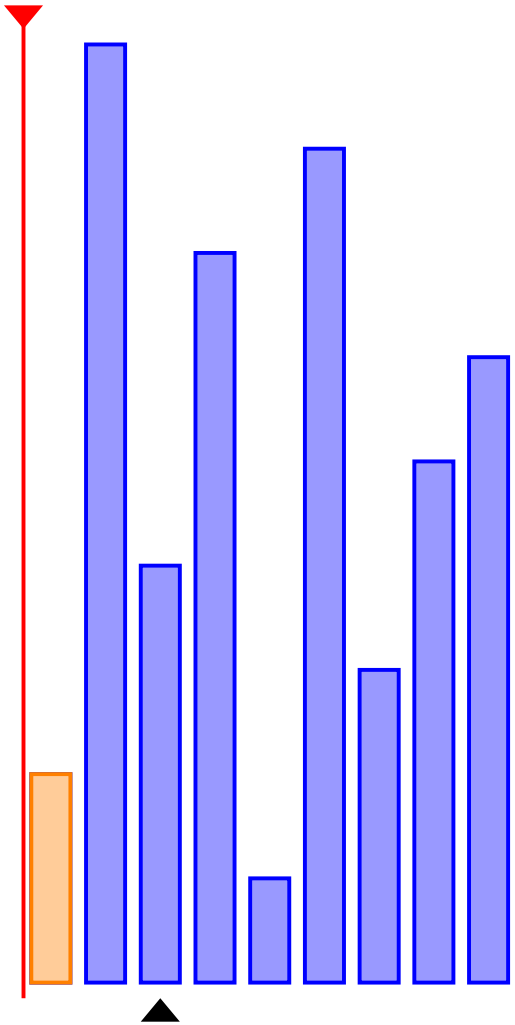


$j = 0$: Suche bei 1

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

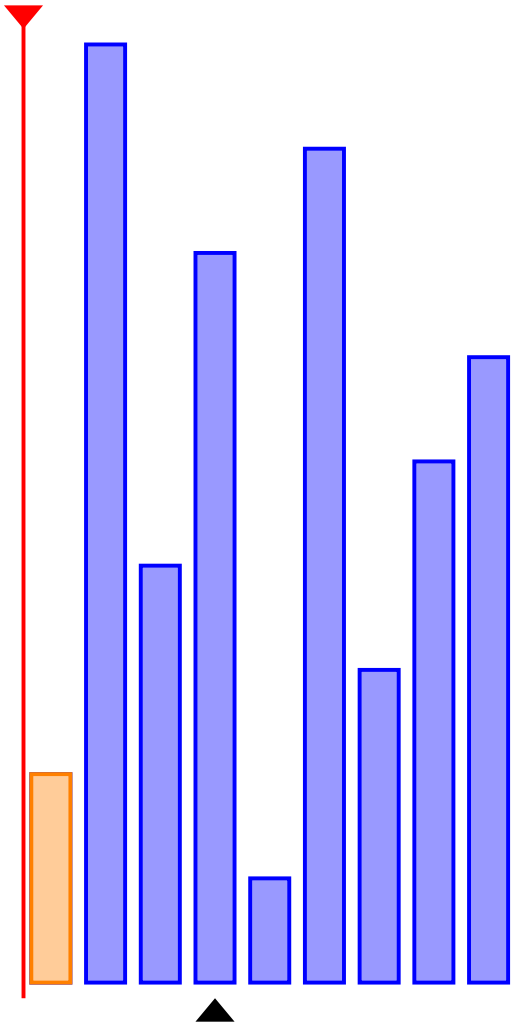


$j = 0$: Suche bei 2

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

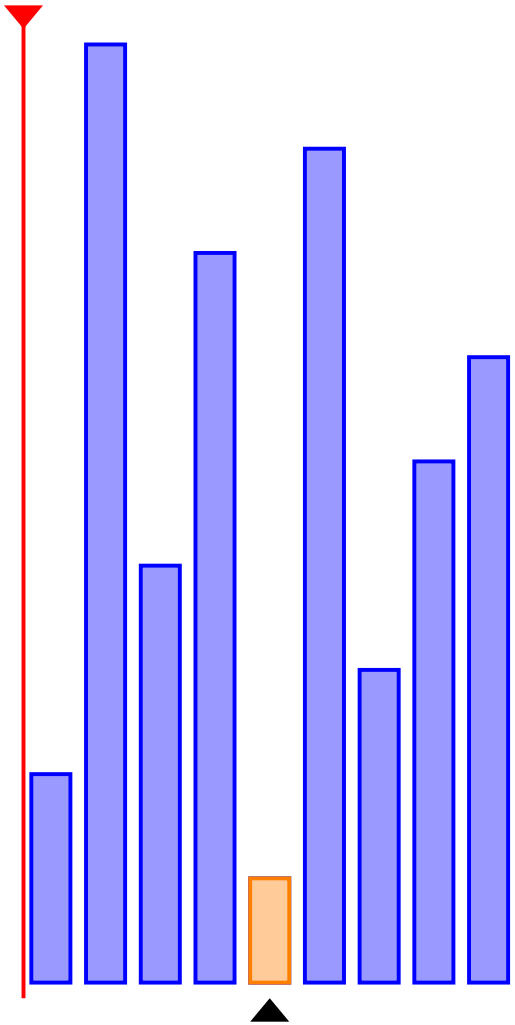


$j = 0$: Suche bei 3

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

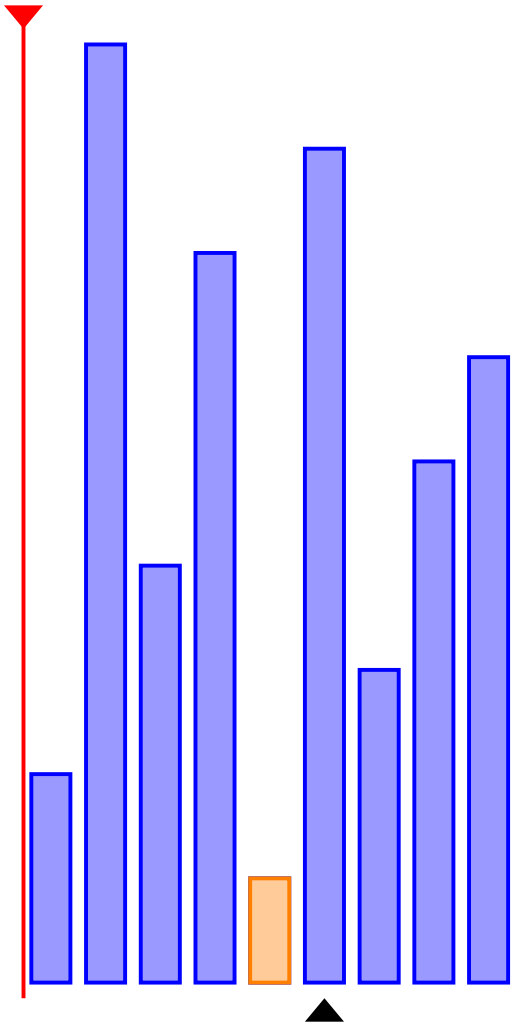


$j = 0$: Suche bei 4

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

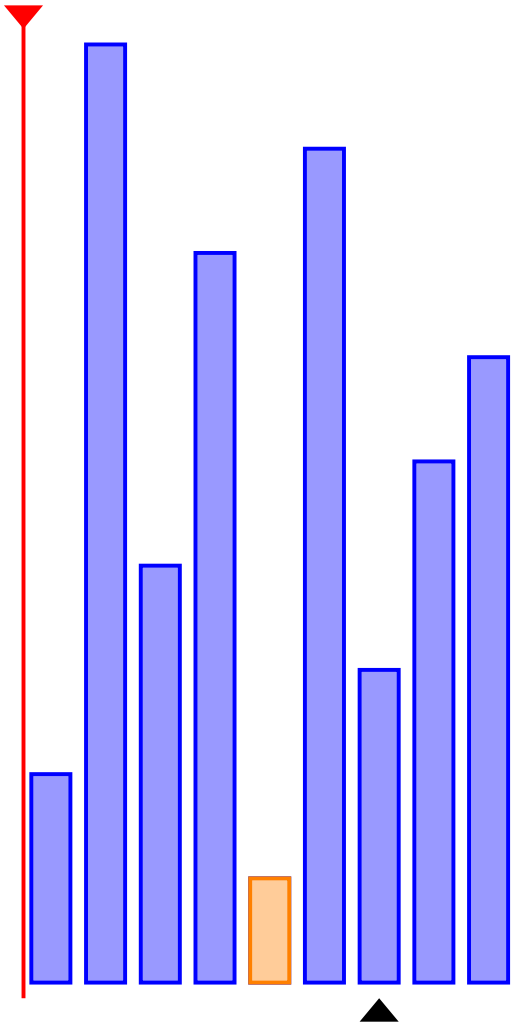


$j = 0$: Suche bei 5

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

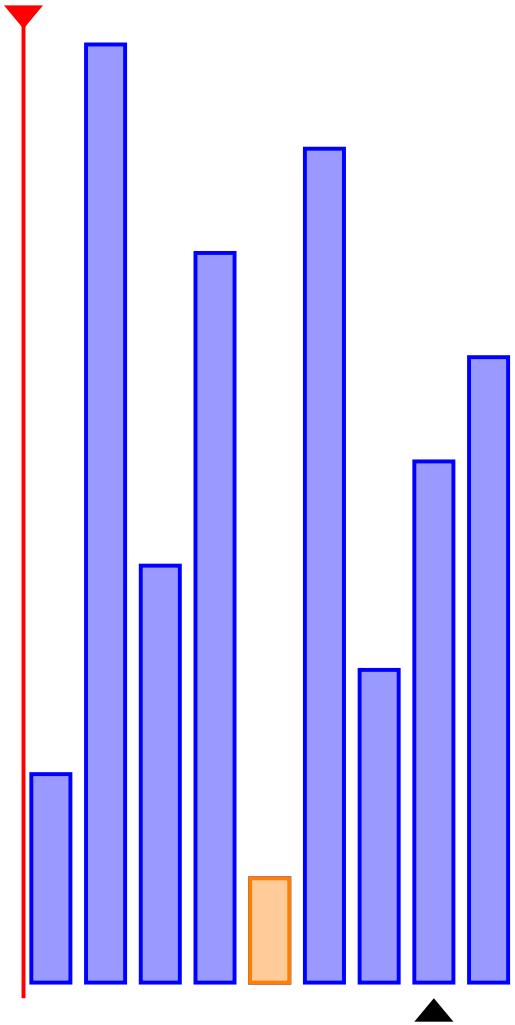


$j = 0$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

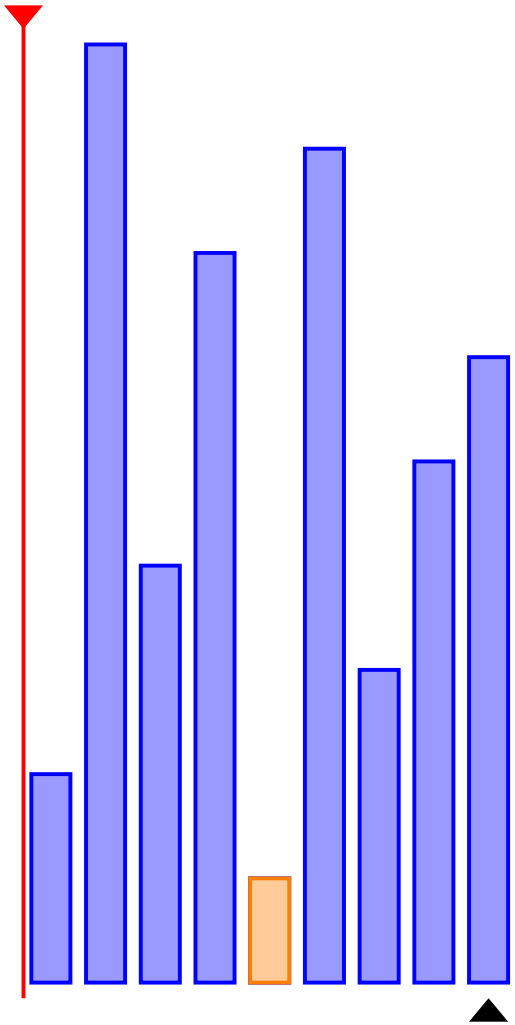


$j = 0$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

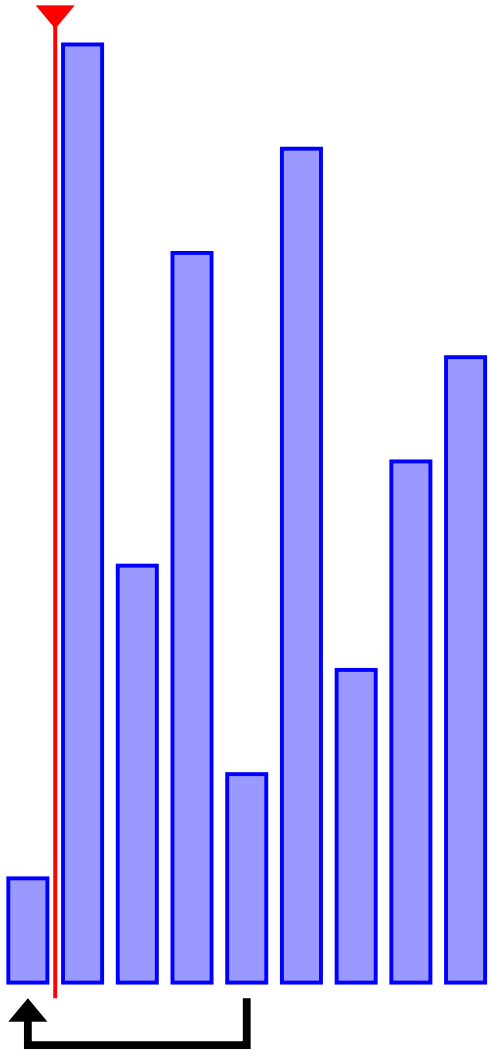


$j = 0$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

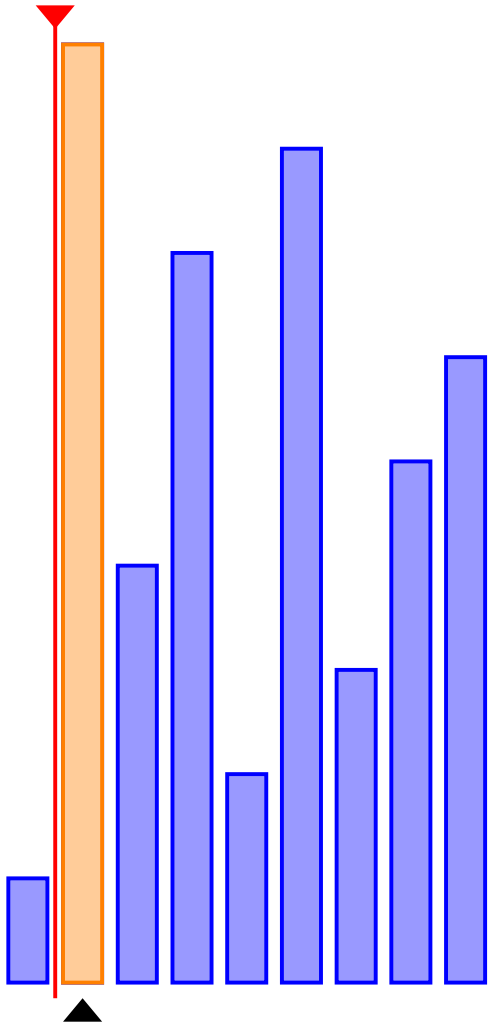


$j = 1$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

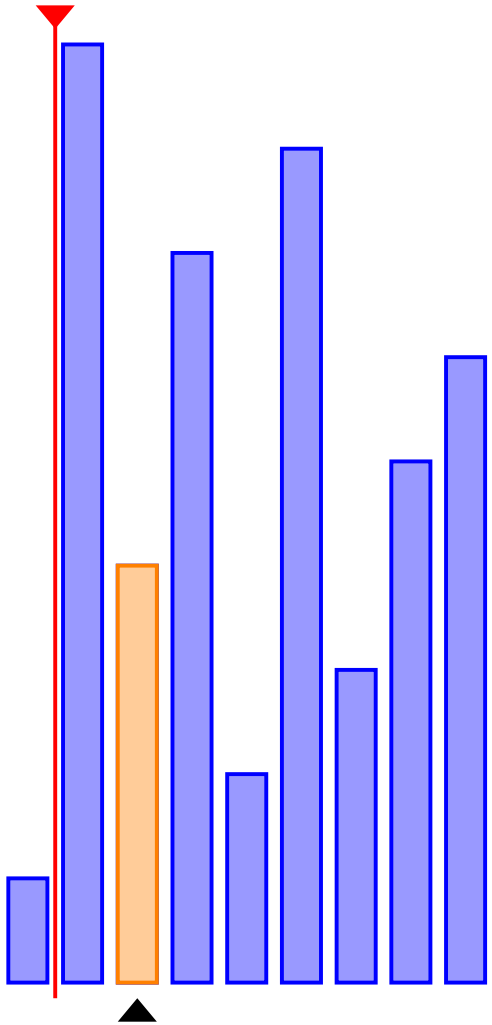


$j = 1$: Suche bei 1

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

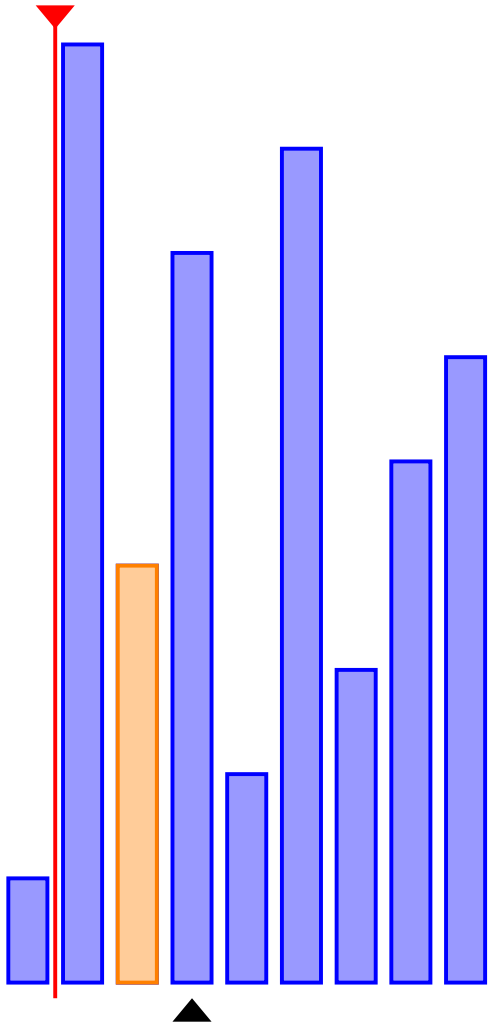


$j = 1$: Suche bei 2

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

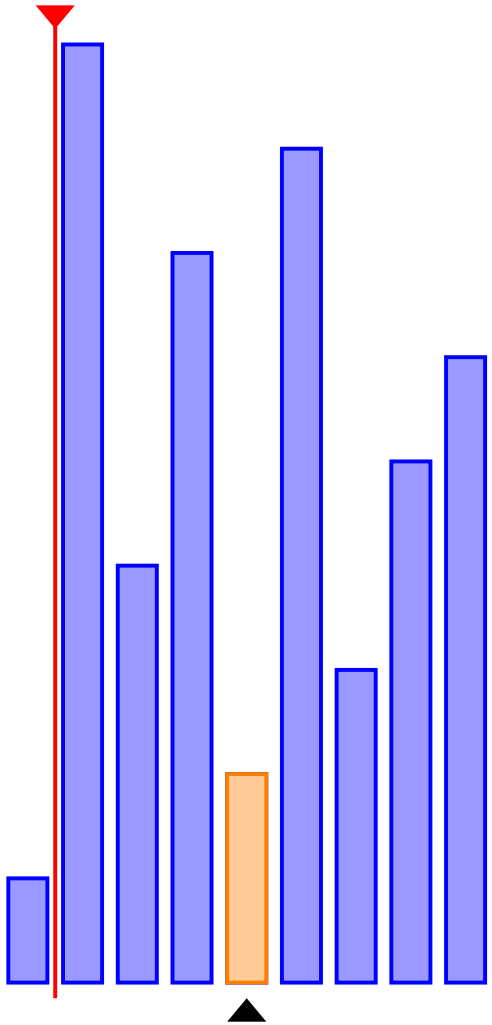


$j = 1$: Suche bei 3

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

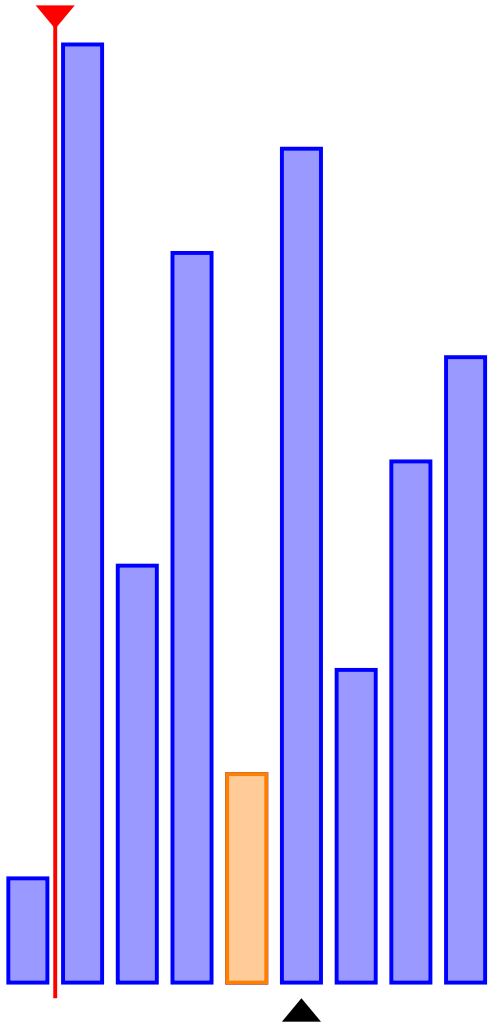


$j = 1$: Suche bei 4

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

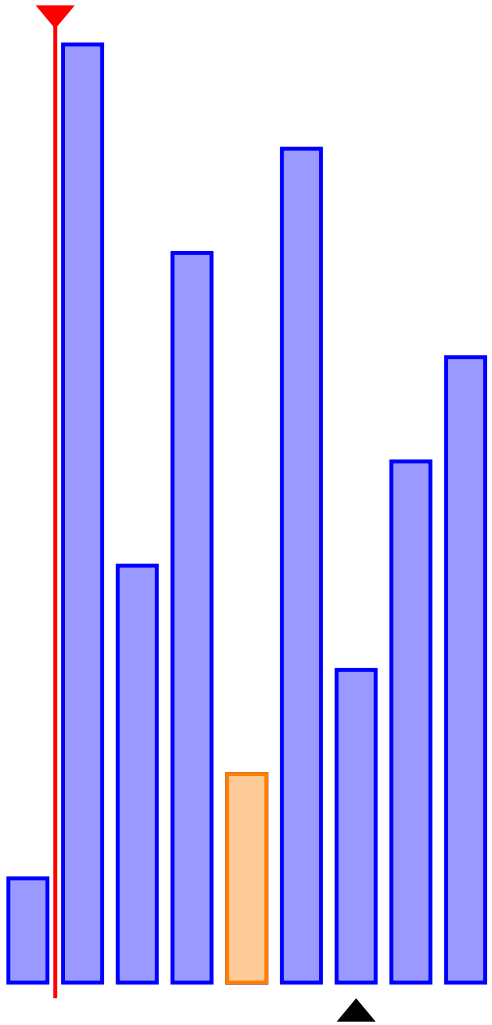


$j = 1$: Suche bei 5

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

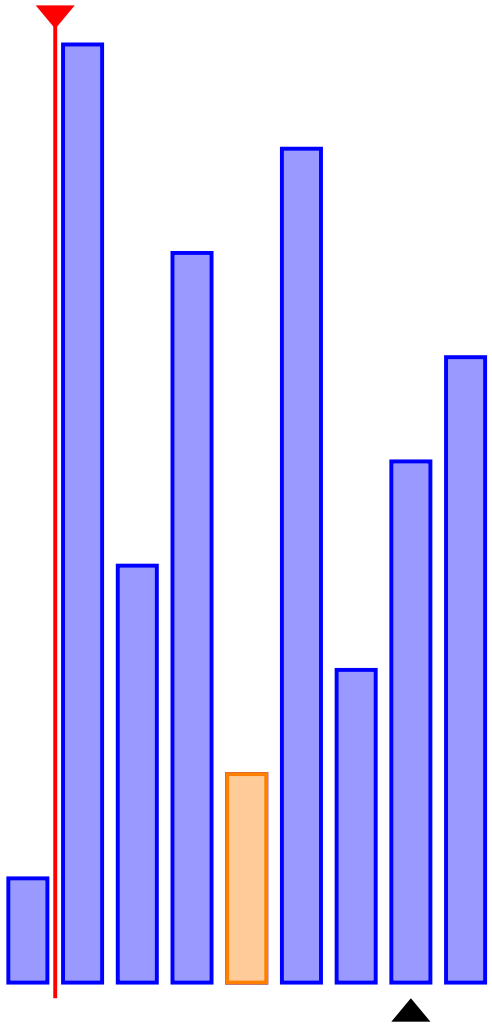


$j = 1$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

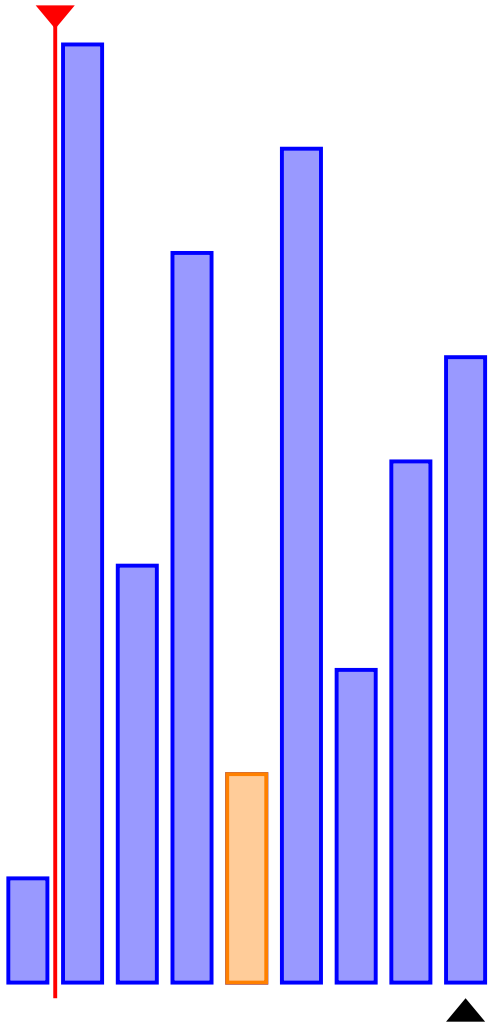


$j = 1$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

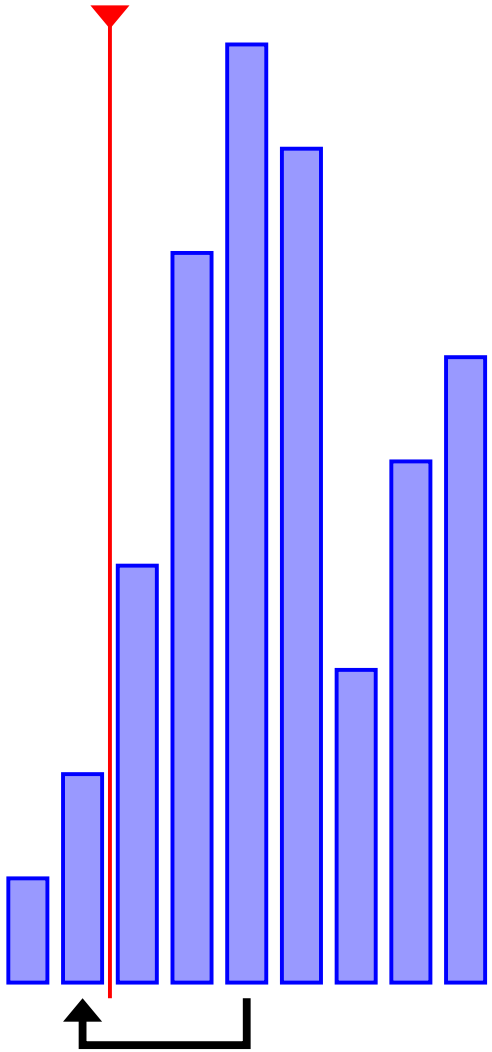


$j = 1$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

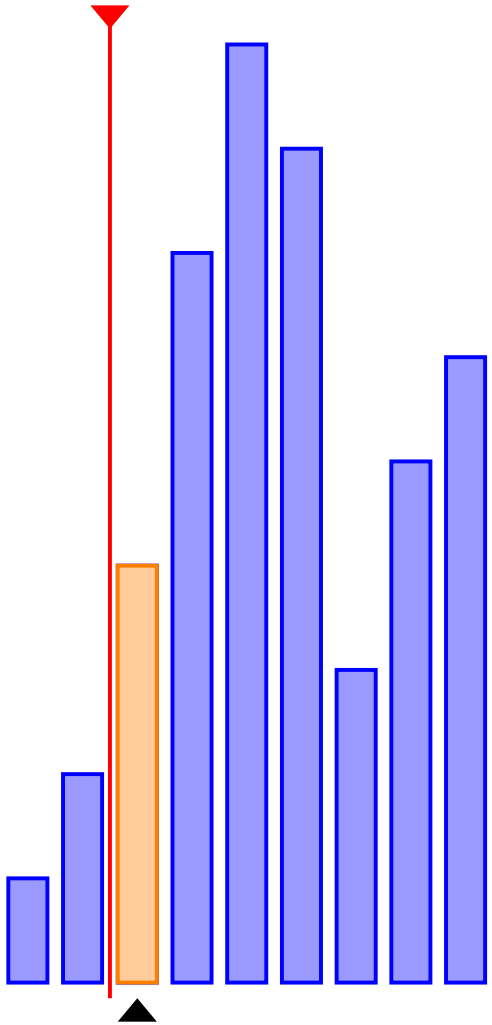


$j = 2$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

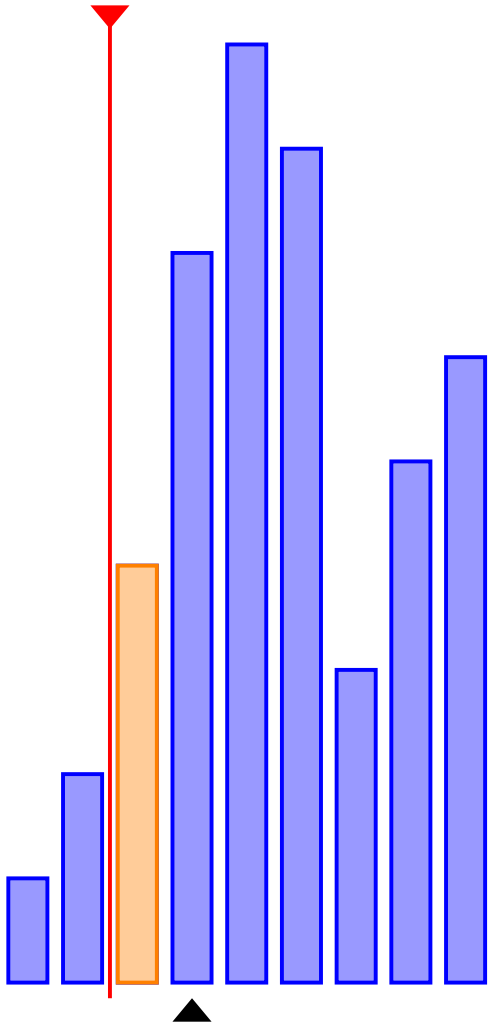


$j = 2$: Suche bei 2

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

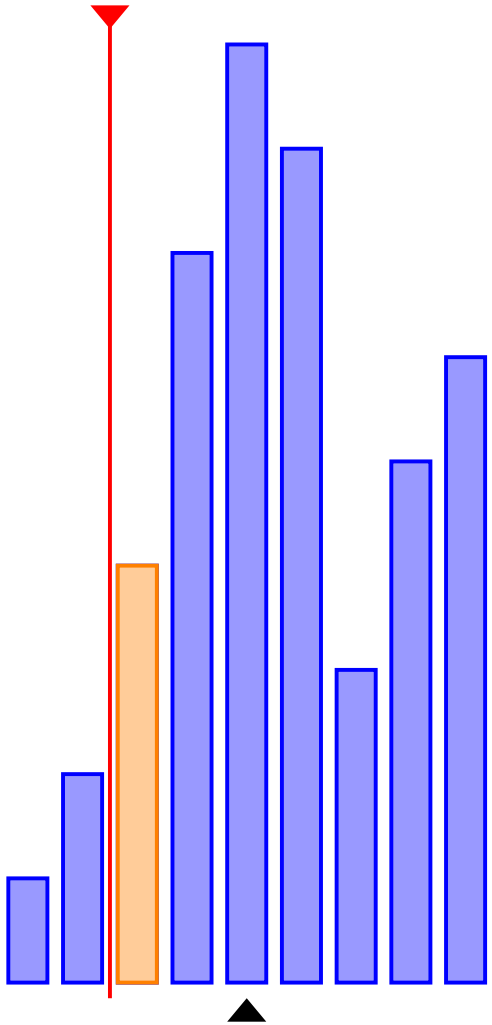


$j = 2$: Suche bei 3

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

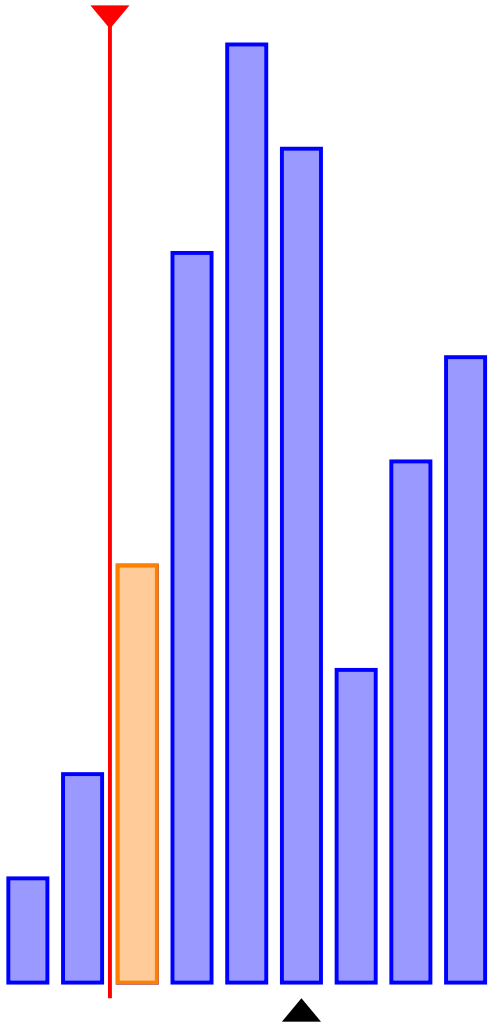


$j = 2$: Suche bei 4

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

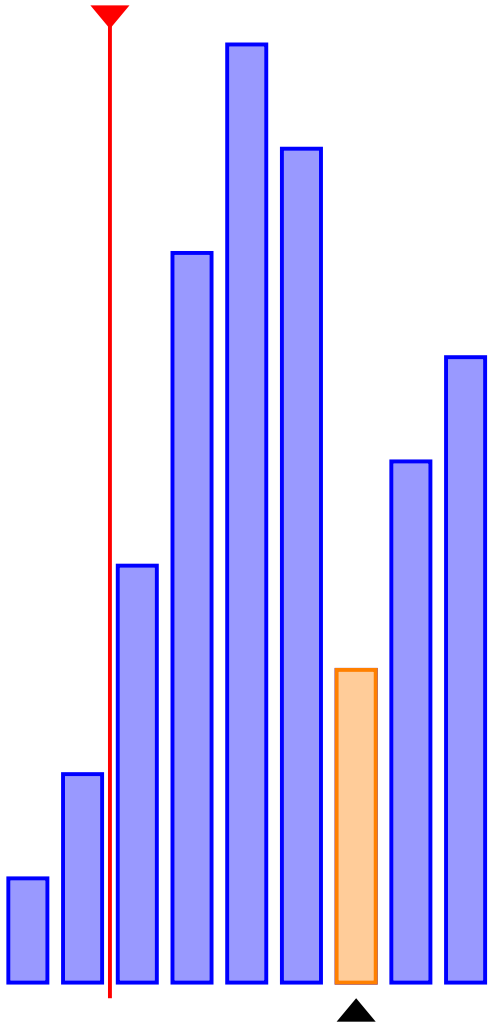


$j = 2$: Suche bei 5

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

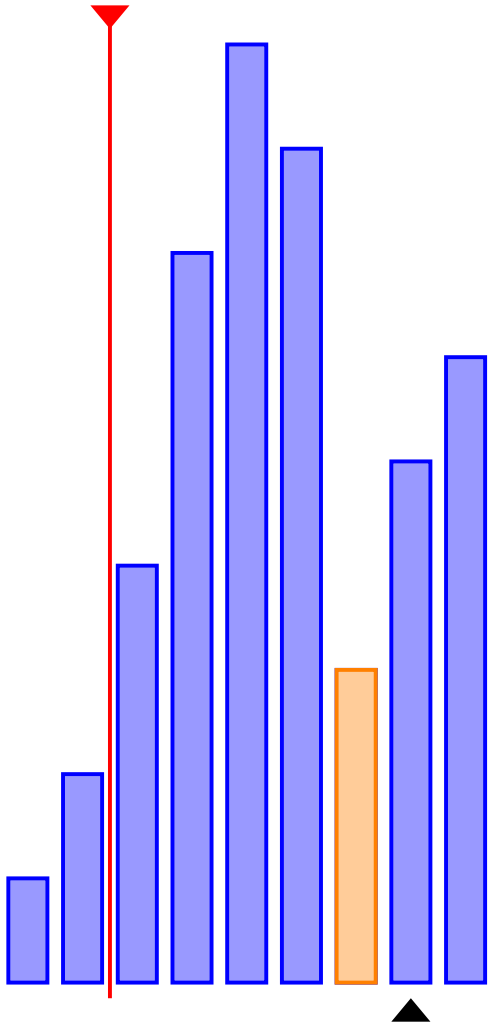


$j = 2$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

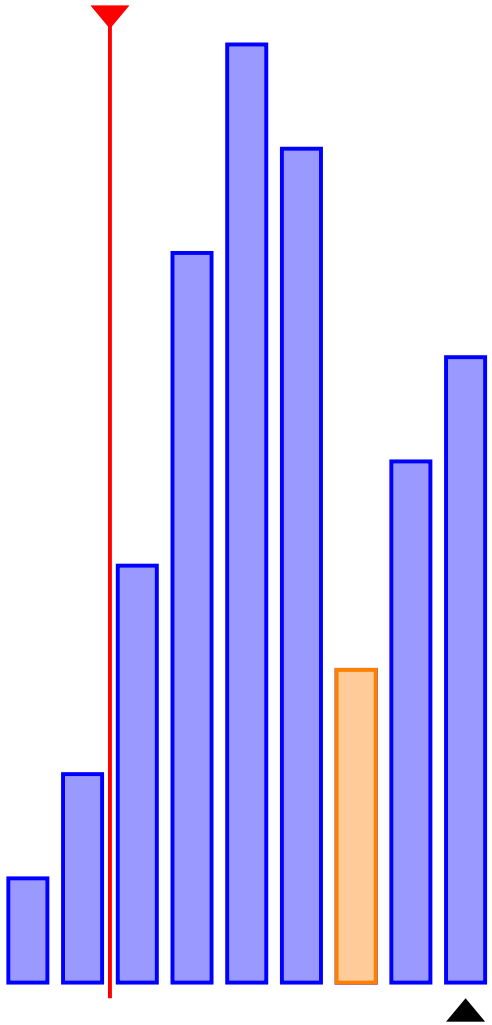


$j = 2$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

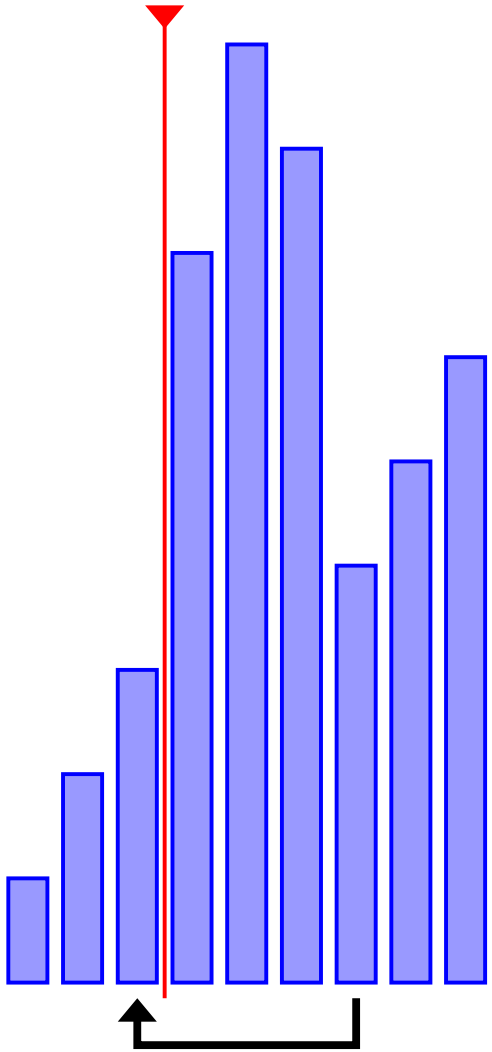


$j = 2$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

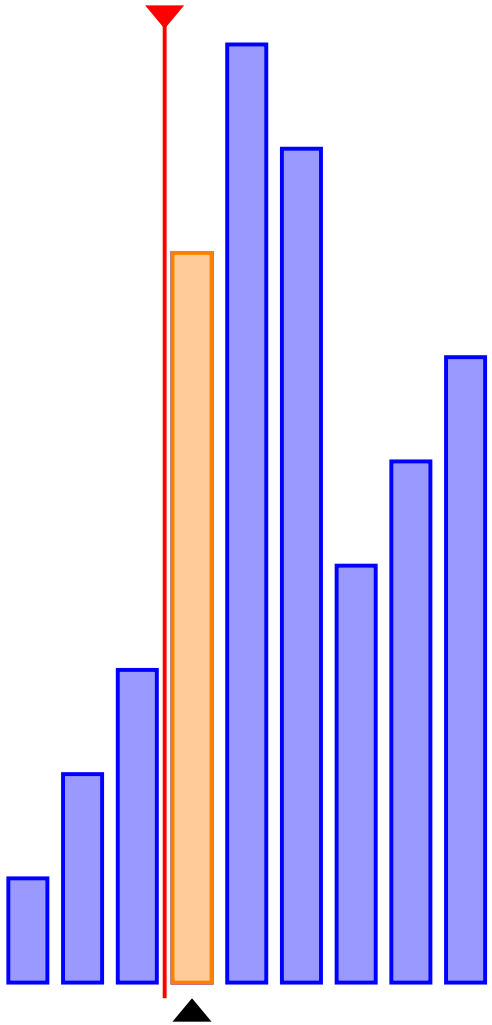


$j = 3$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

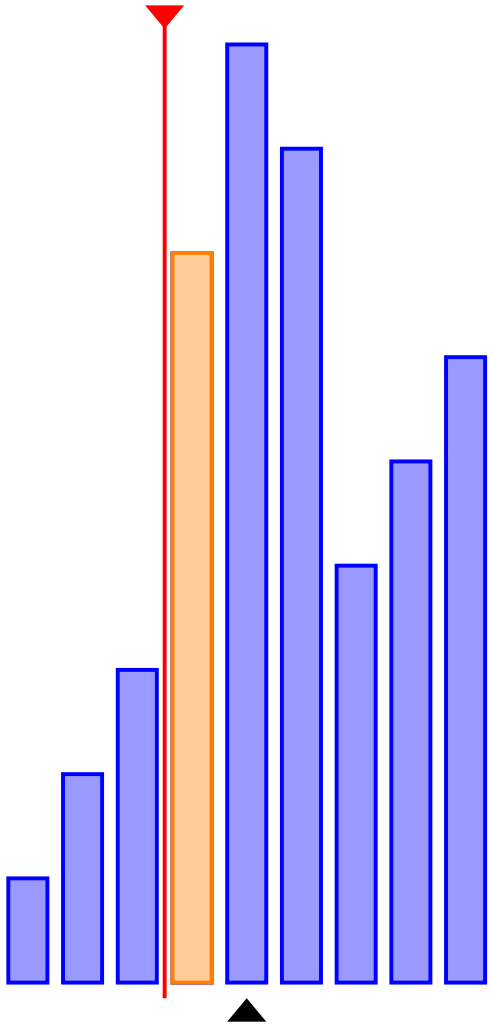


$j = 3$: Suche bei 3

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

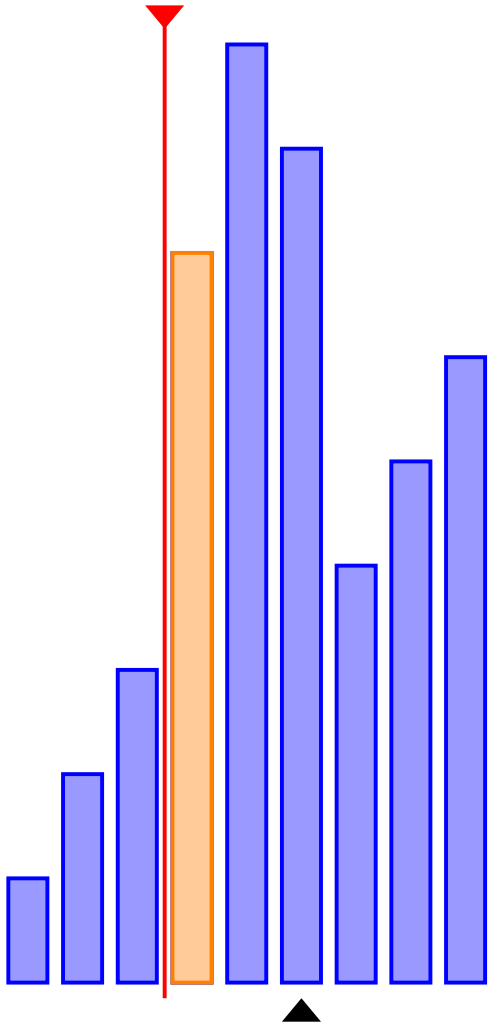


$j = 3$: Suche bei 4

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

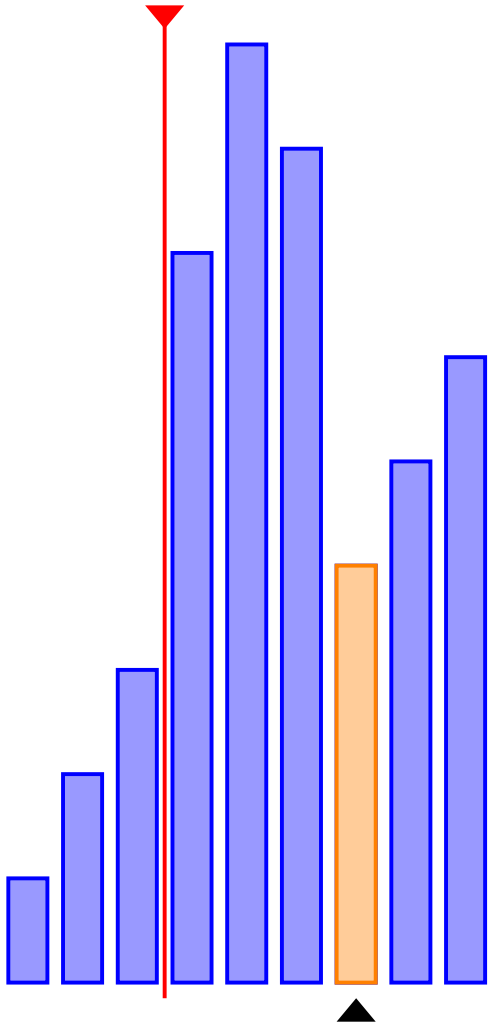


$j = 3$: Suche bei 5

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

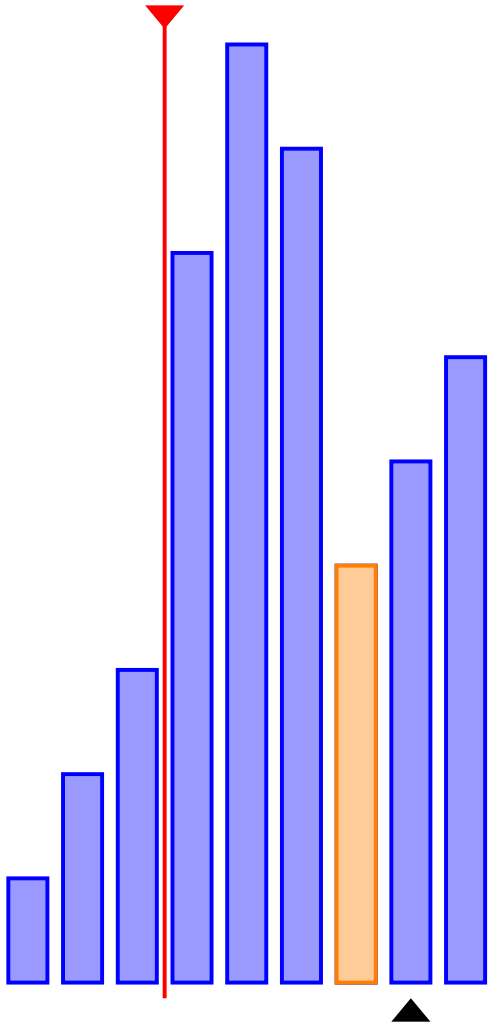


$j = 3$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

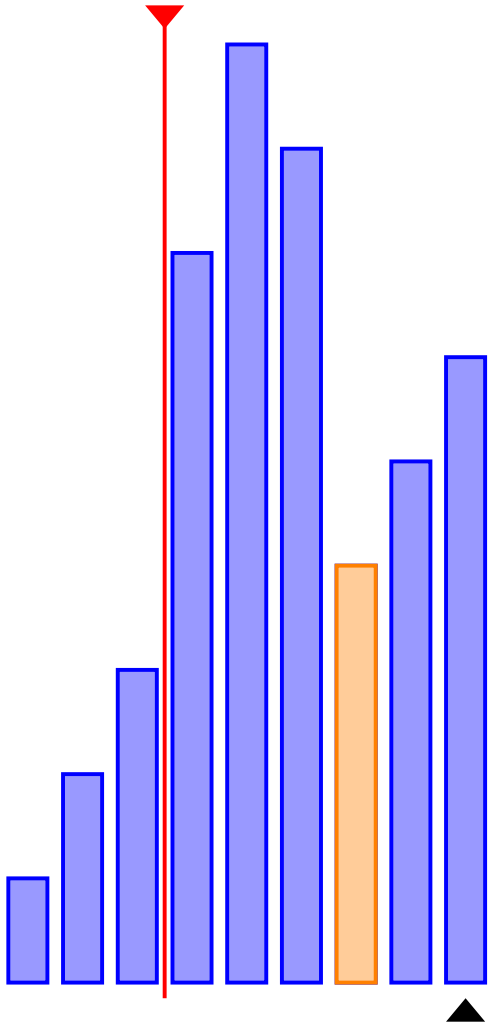


$j = 3$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

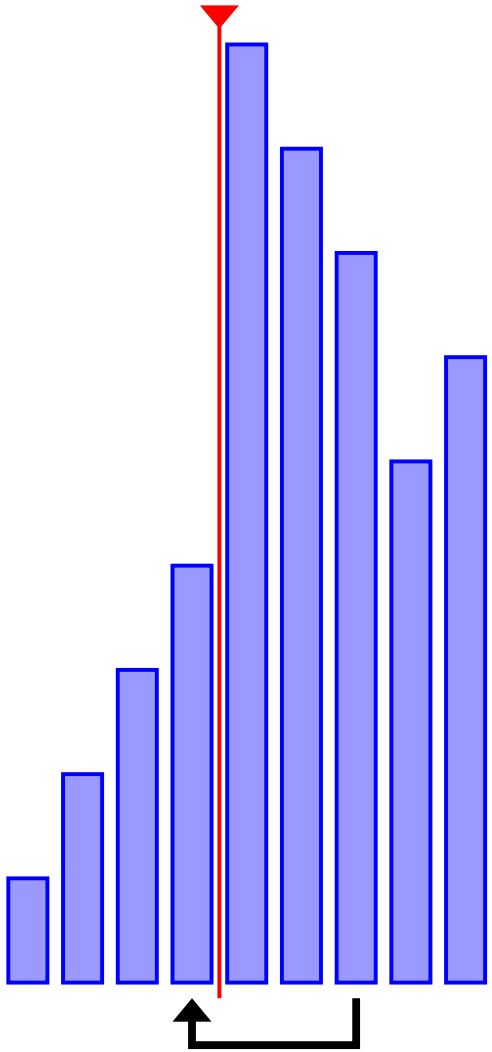


$j = 3$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

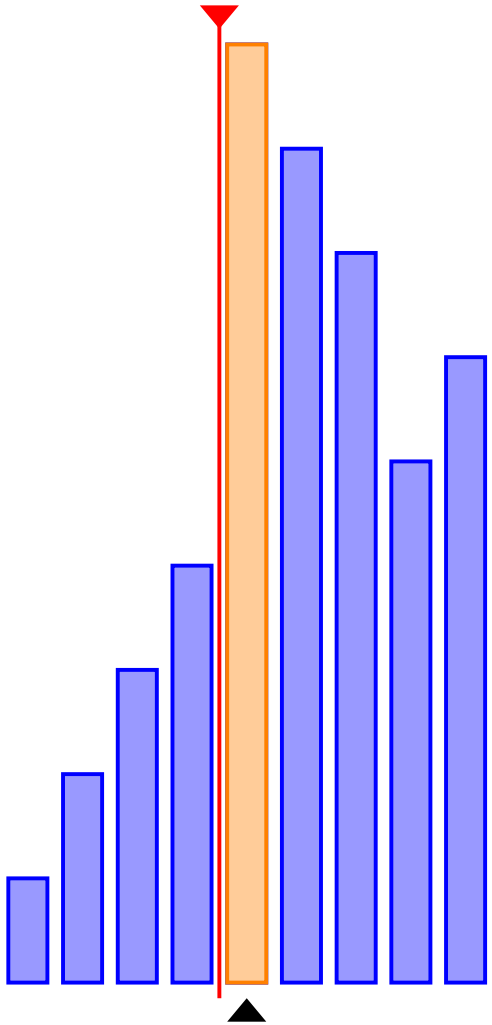


$j = 4$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

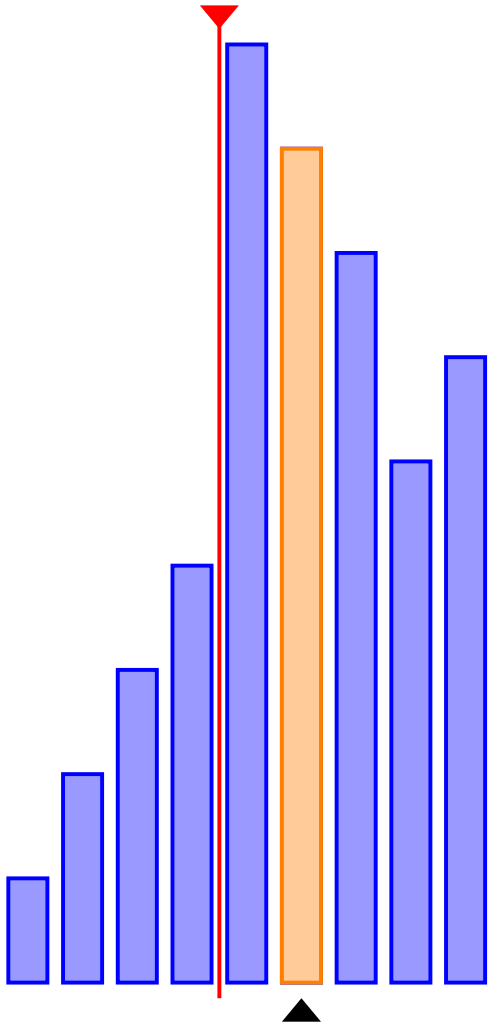


$j = 4$: Suche bei 4

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

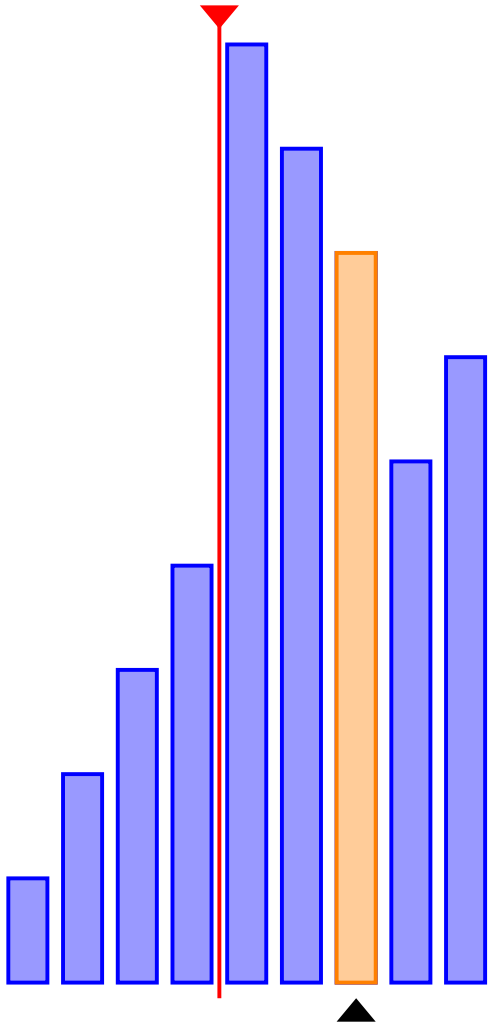


$j = 4$: Suche bei 5

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

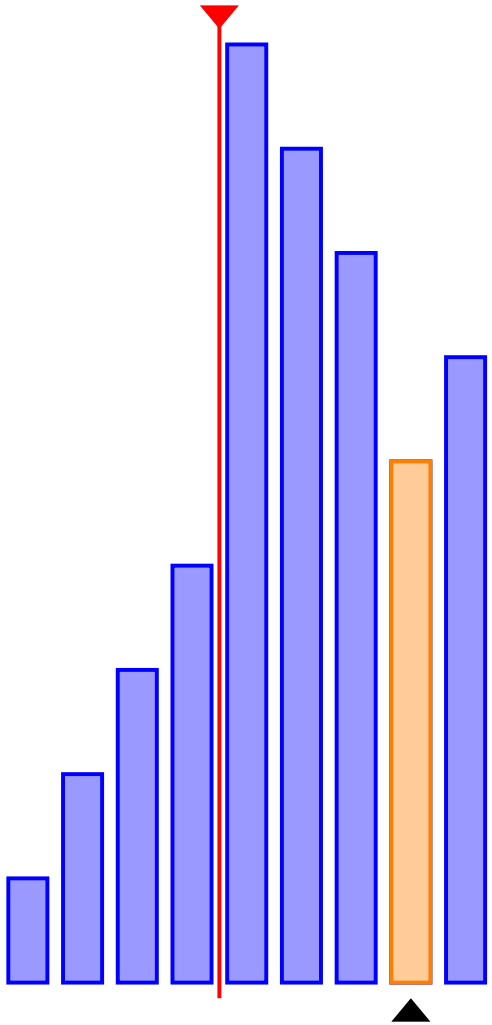


$j = 4$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

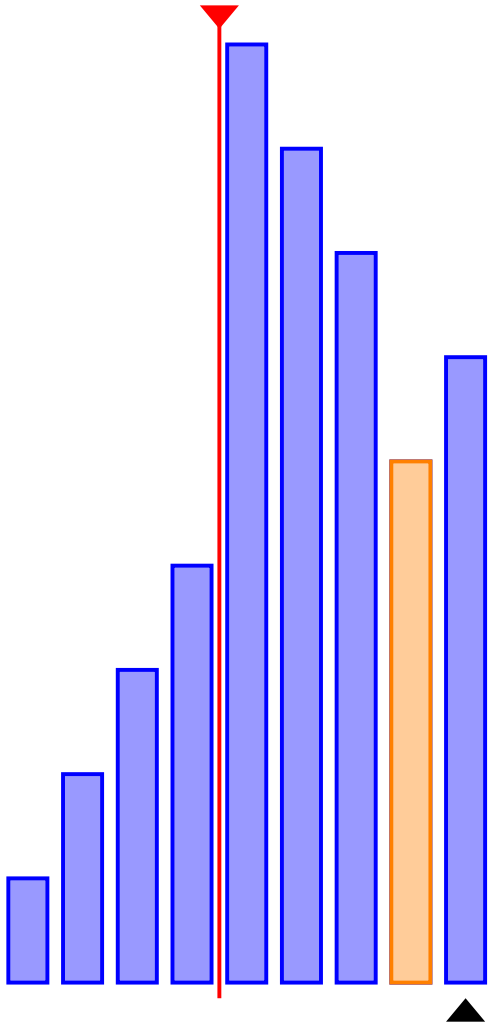


$j = 4$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

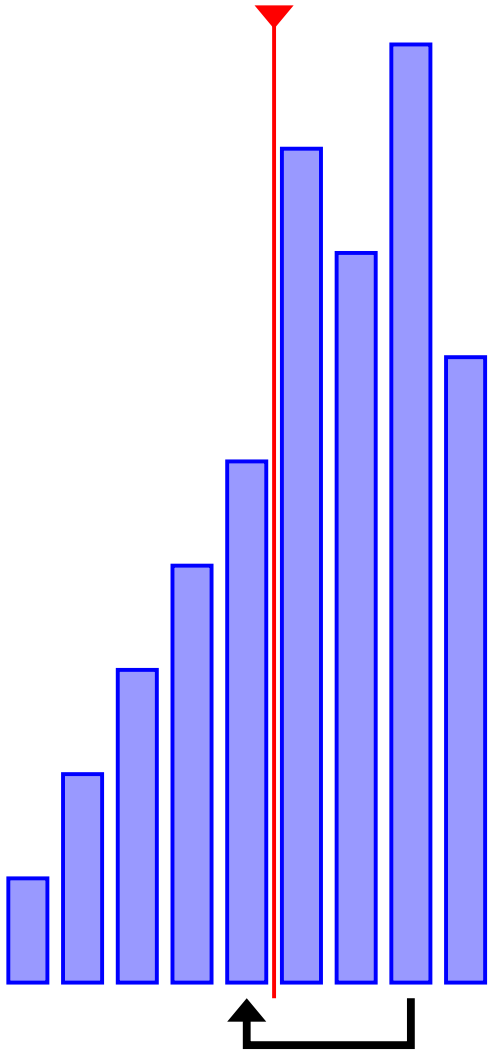


$j = 4$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

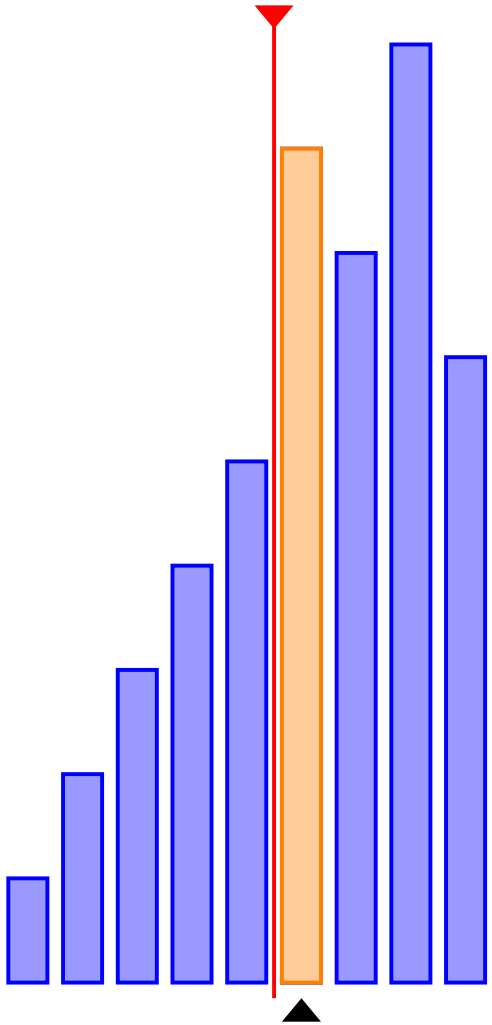


$j = 5$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

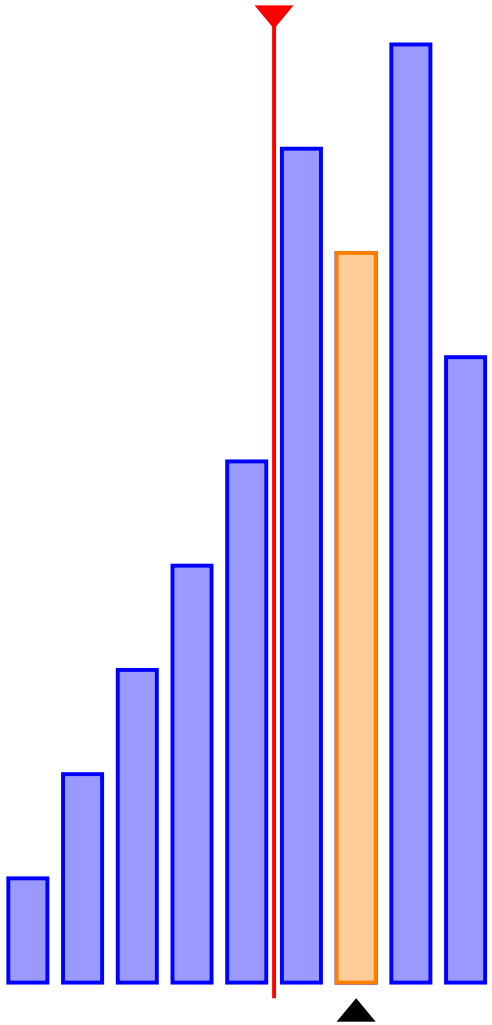


$j = 5$: Suche bei 5

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

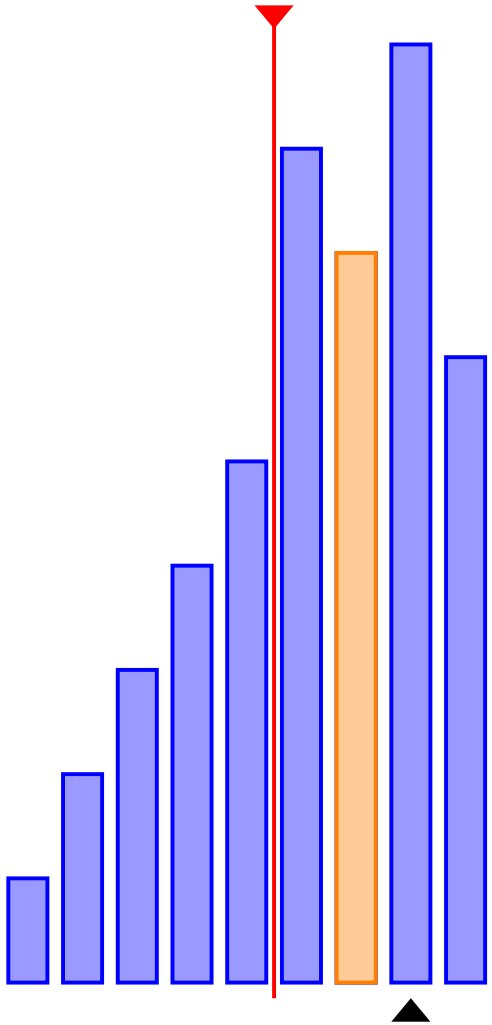


$j = 5$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

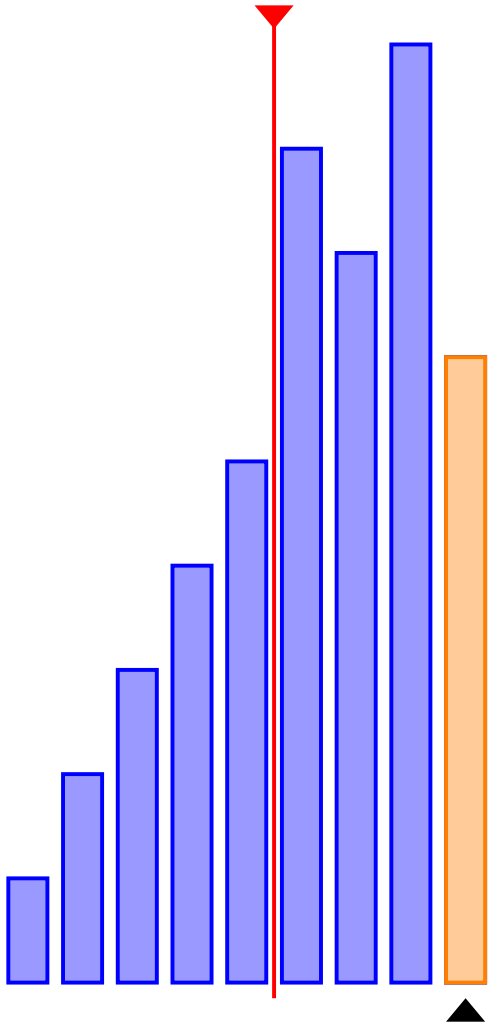


$j = 5$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

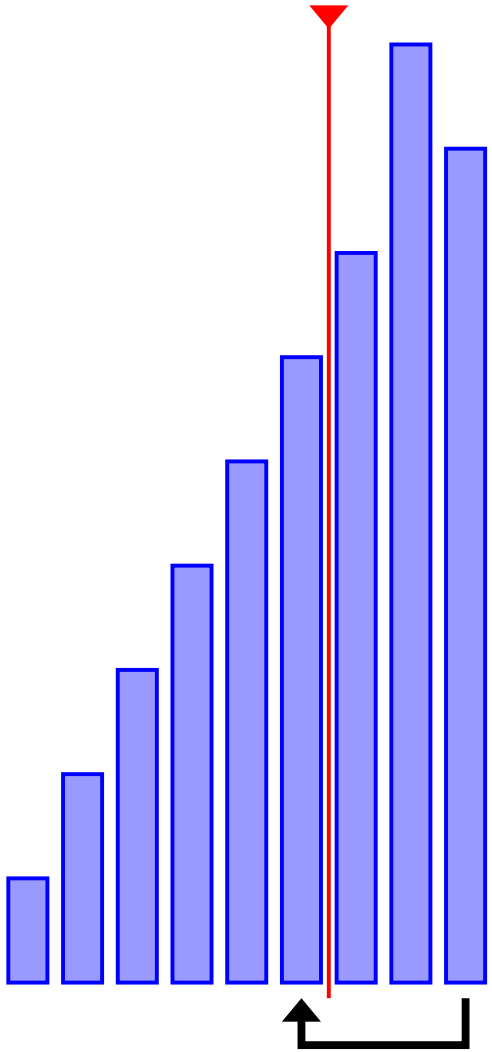


$j = 5$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

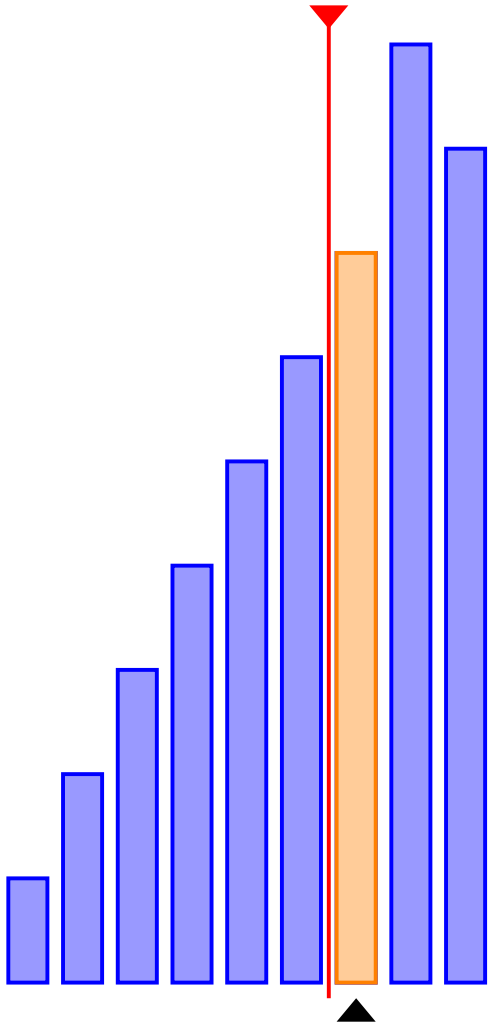


$j = 6$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

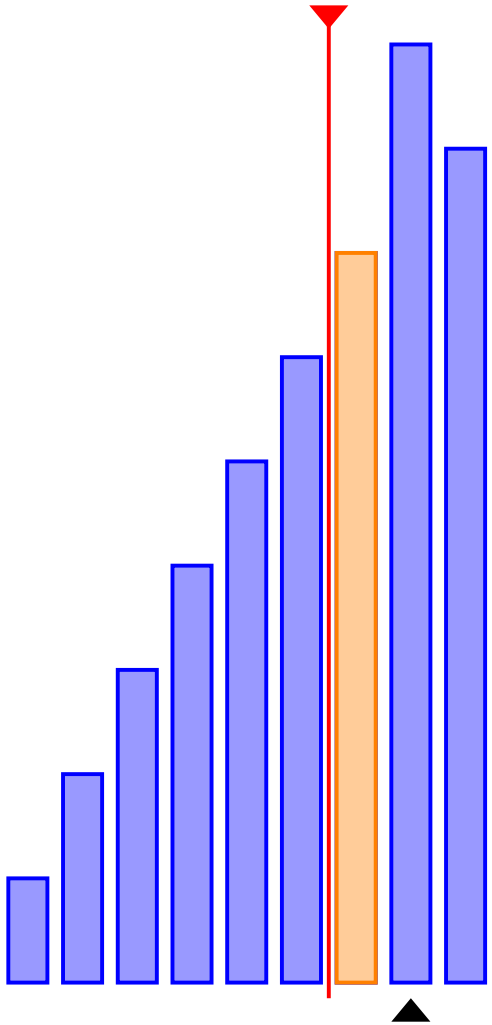


$j = 6$: Suche bei 6

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

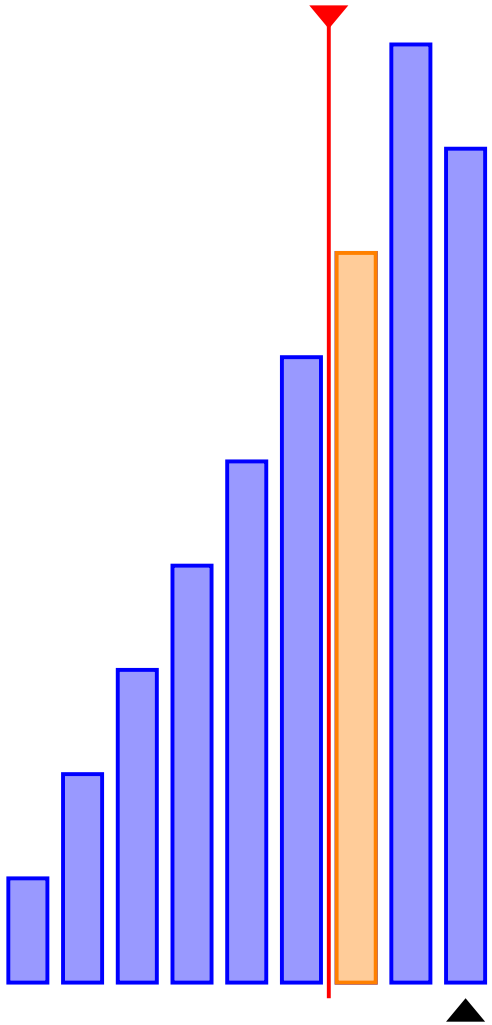


$j = 6$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

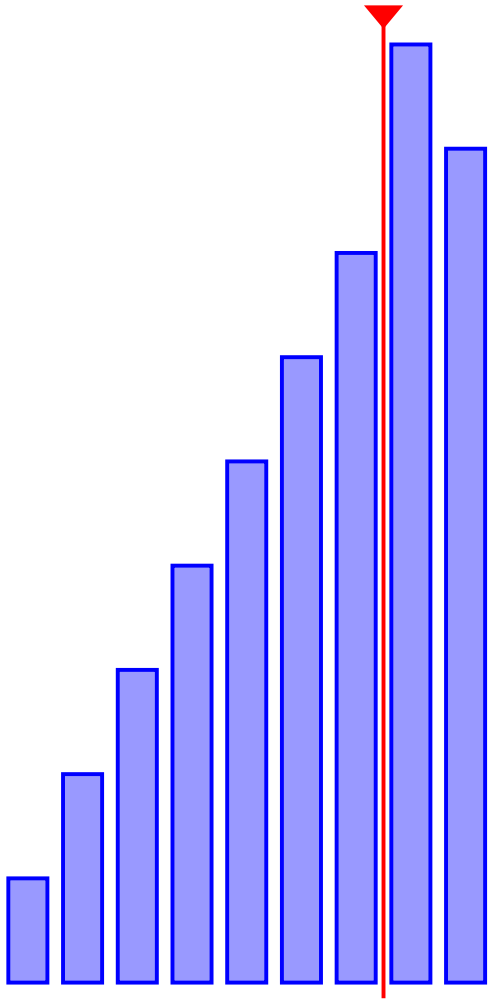


$j = 6$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

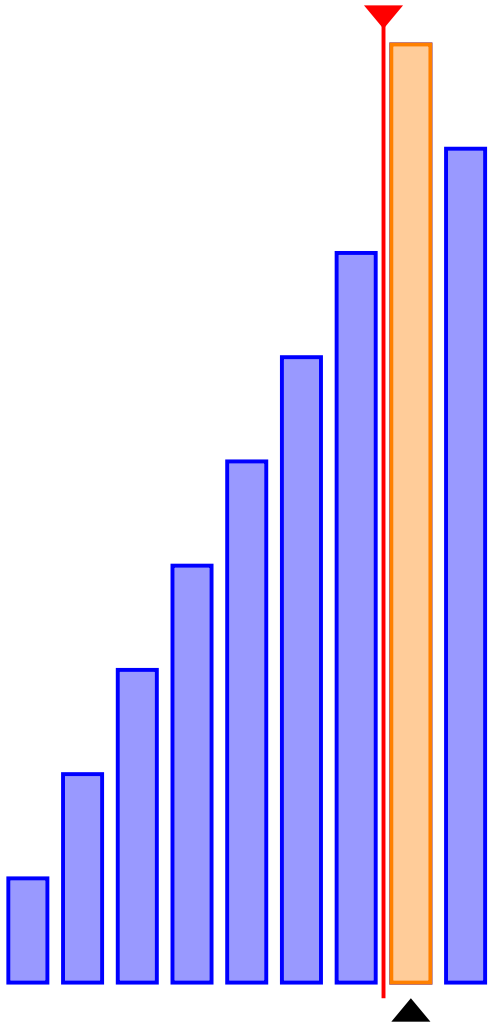


$j = 7$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

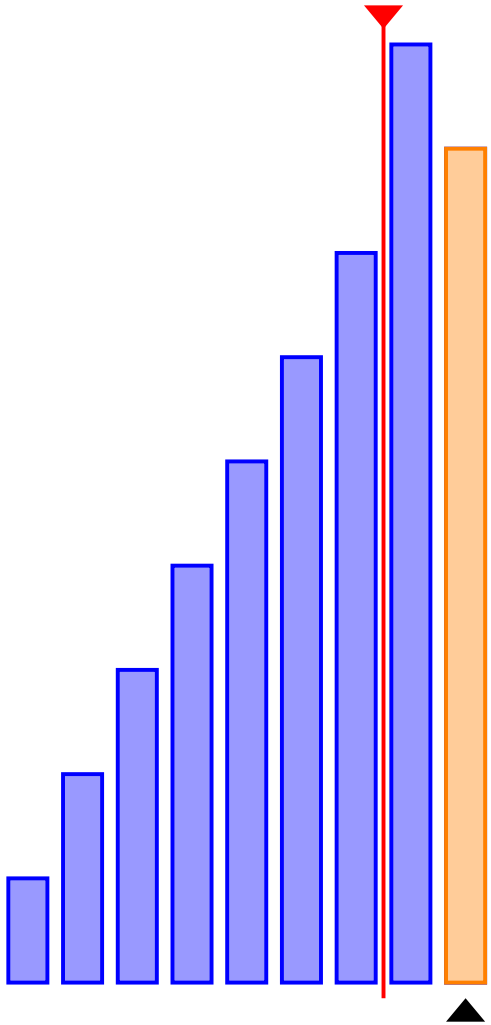


$j = 7$: Suche bei 7

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

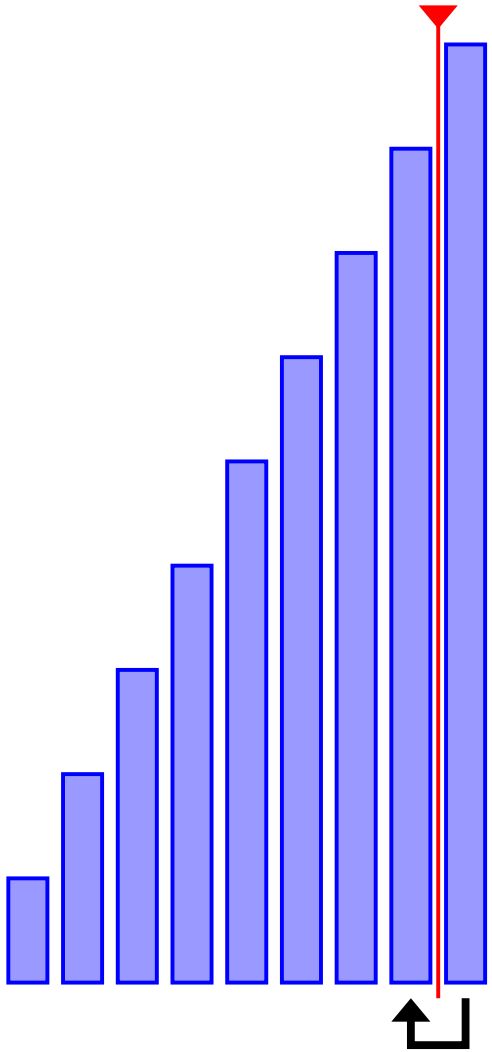


$j = 7$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

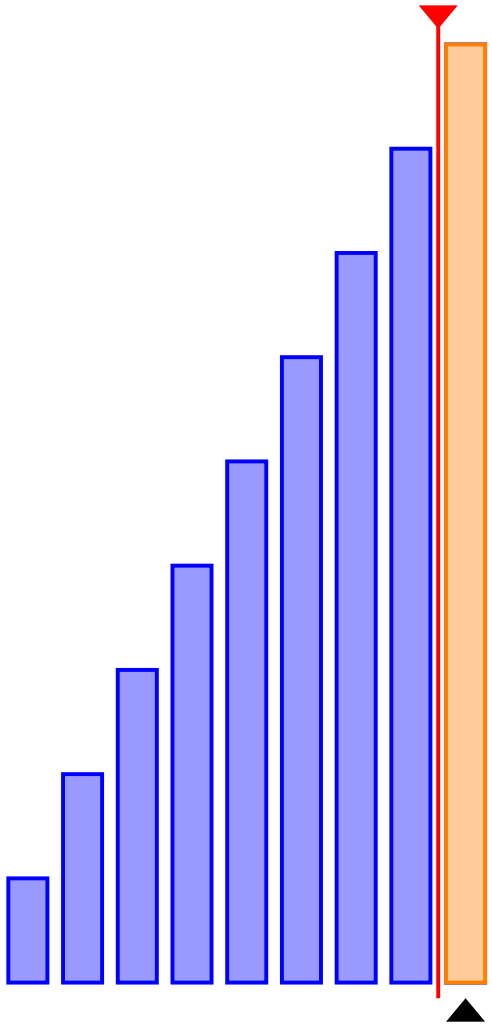


$j = 8$: Anfangszustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort

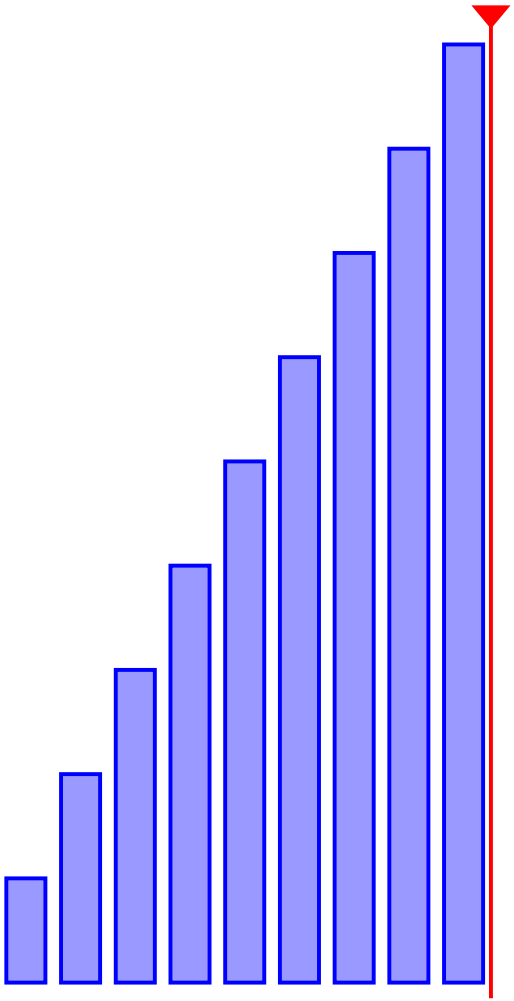


$j = 8$: Suche bei 8

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

Selection Sort



Endzustand

```
int index_of_min(int* A, int n, int j)
{
        int i = j; int Ai = A[j];
        for (int k = j+1; k < n; ++k)
        {
                if (A[k] < Ai)
                        {i = k; Ai = A[k];}
        }
        return i;
}

void selection_sort(int* A, int n)
{
        for (int j = 0; j < n; ++j)
        {
                int i = index_of_min(A, n, j);
                if (i != j)
                        {swap(A, i, j);}
        }
}
```

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: $\{P \wedge C\}\ \beta\ \{P\}$
  - $P \wedge \neg C \Rightarrow B$

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 $A[k...(n-1)]$
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 = IndexOfMin(A, j);
                if (j ≠ i) {
                        Swap(A, i, j);
                }
                j = j + 1;
        }
{NACH}
```

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

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

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 $(j < n)$ {
$i = \text{IndexOfMin}(A, j);$
if $(j \neq i)$ {
$\text{Swap}(A, i, j);$
}
$\{C_4\}$
$j = j + 1;$
}
$\{NACH\}$

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

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

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!