

Übung zu
Algorithmen und Datenstrukturen (für ET/IT)
Sommersemester 2015

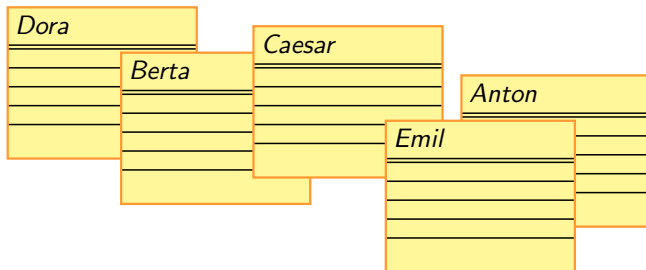
Matthias Wiczorek

Computer-Aided Medical Procedures
Technische Universität München



Sortierung: Beispielproblem

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



Insertion Sort

<i>Dora</i>
<i>Berta</i>
<i>Caesar</i>
<i>Anton</i>
<i>Emil</i>

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

Insertion Sort

<i>Emil</i>

<i>Dora</i>
<i>Berta</i>
<i>Caesar</i>
<i>Anton</i>

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

Insertion Sort

<i>Emil</i>
<i>Anton</i>

<i>Dora</i>
<i>Berta</i>
<i>Caesar</i>

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

Insertion Sort

<i>Emil</i>
<i>Caesar</i>
<i>Anton</i>

<i>Dora</i>
<i>Berta</i>

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

Insertion Sort

<i>Emil</i>
<i>Caesar</i>
<i>Berta</i>
<i>Anton</i>

<i>Dora</i>

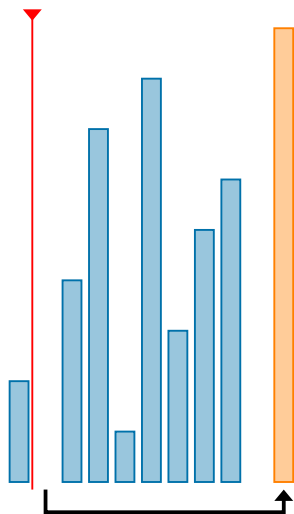
- ▶ Gustav nimmt die Karte von *Dora*, und sortiert sie in den bereits sortierten Stoss ein

Insertion Sort

<i>Emil</i>
<i>Dora</i>
<i>Caesar</i>
<i>Berta</i>
<i>Anton</i>

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

Insertion Sort



$j = 1$: Anfangszustand

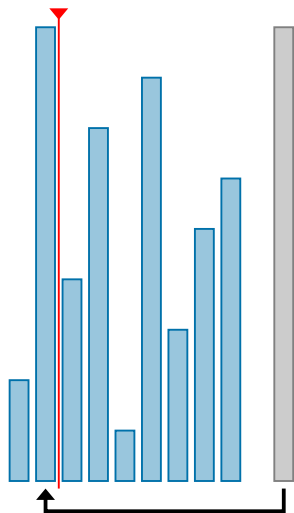
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 1$: Endzustand

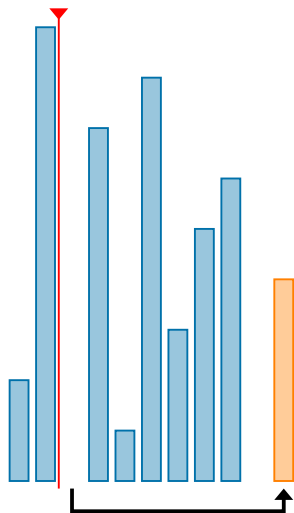
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 2$: Anfangszustand

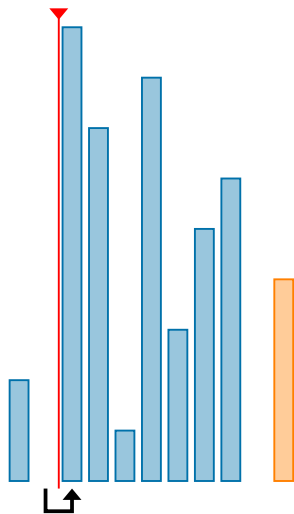
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



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

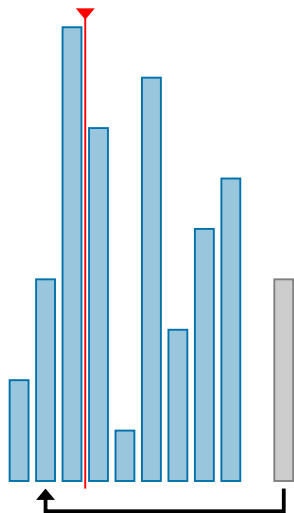
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 2$: Endzustand

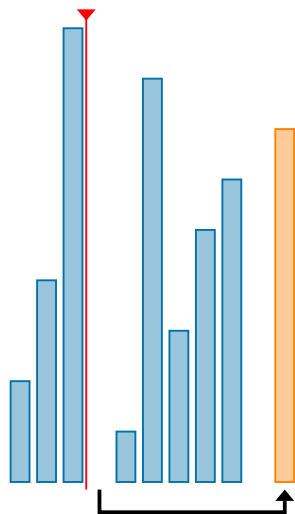
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 3$: Anfangszustand

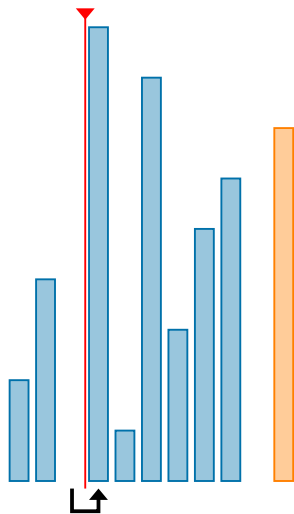
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



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

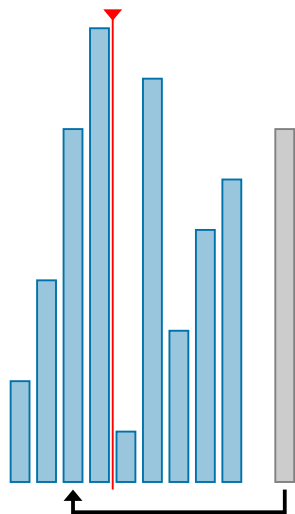
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 3$: Endzustand

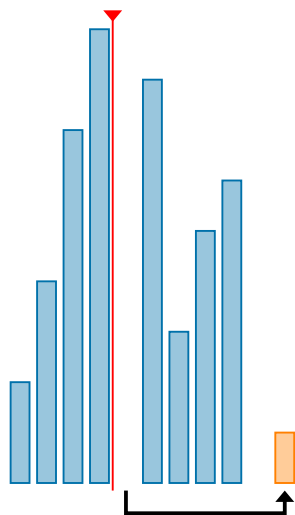
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```


Insertion Sort



$j = 4$: Anfangszustand

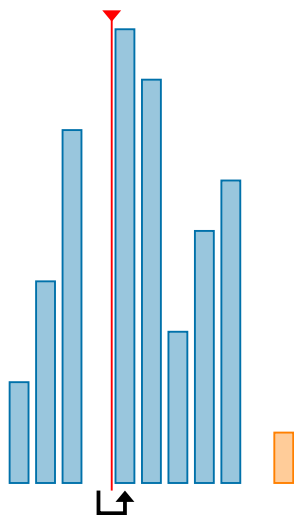
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 4$: Platz für Einfügung

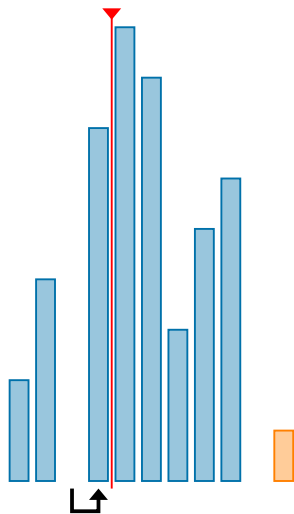
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 4$: Platz für Einfügung

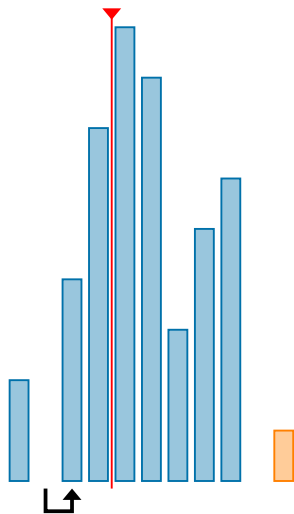
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 4$: Platz für Einfügung

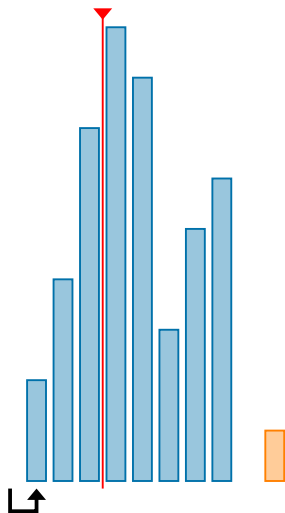
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



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

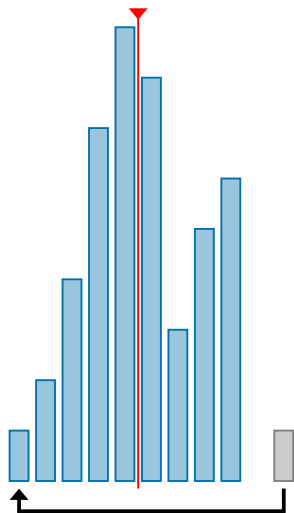
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 4$: Endzustand

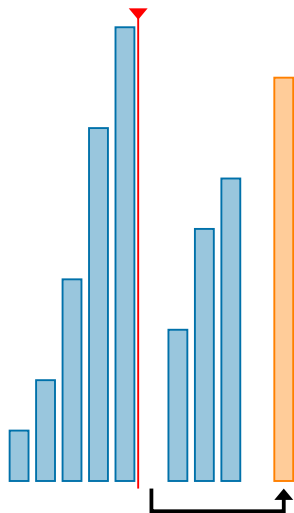
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 5$: Anfangszustand

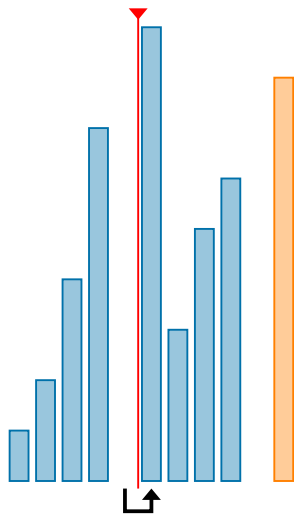
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 5$: Platz für Einfügung – fertig

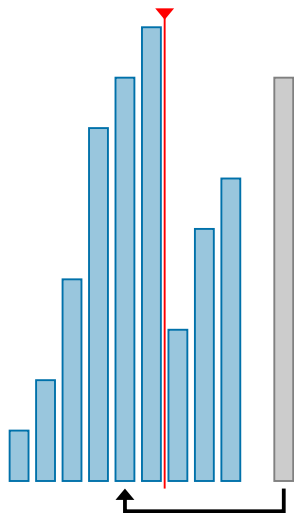
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```


Insertion Sort



$j = 5$: Endzustand

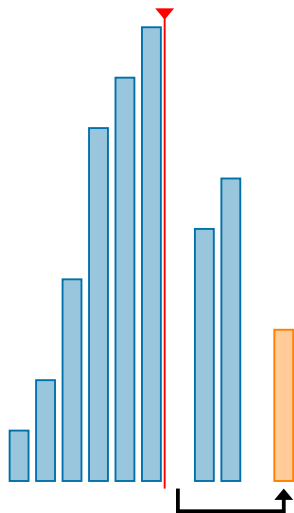
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 6$: Anfangszustand

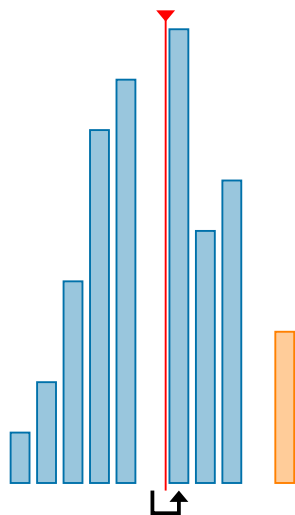
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 6$: Platz für Einfügung

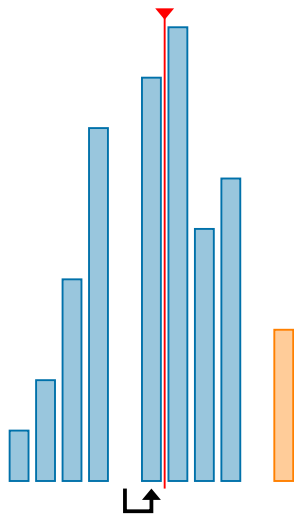
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 6$: Platz für Einfügung

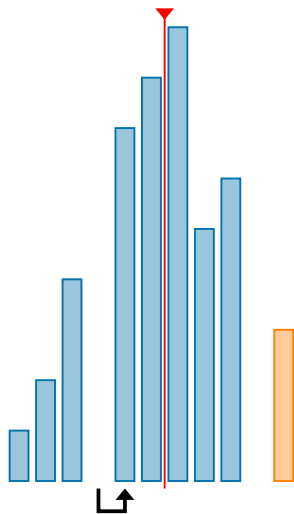
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 6$: Platz für Einfügung

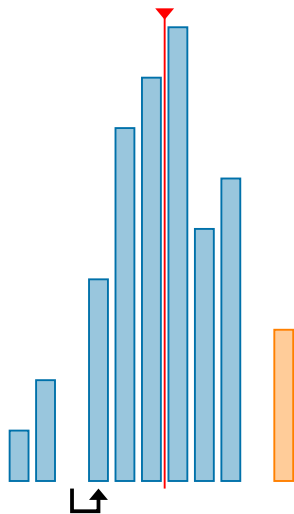
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



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

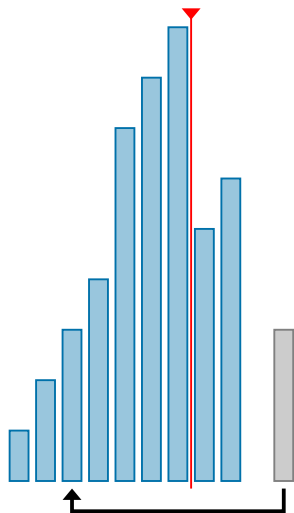
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 6$: Endzustand

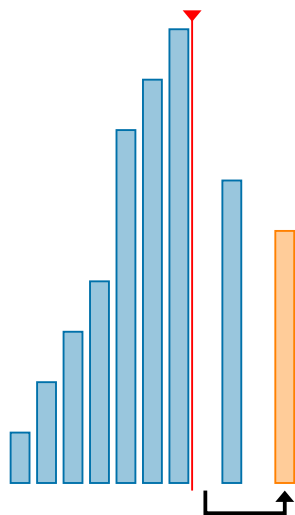
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 7$: Anfangszustand

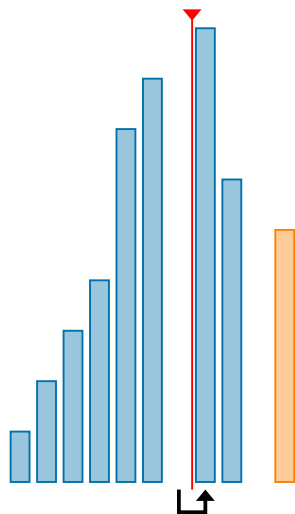
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```


Insertion Sort



$j = 7$: Platz für Einfügung

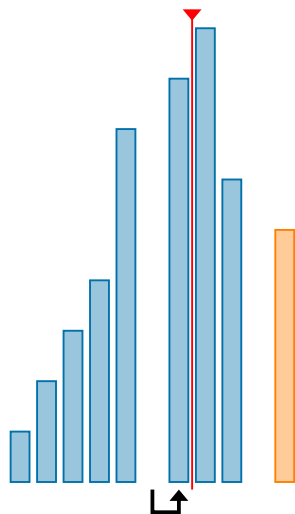
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 7$: Platz für Einfügung

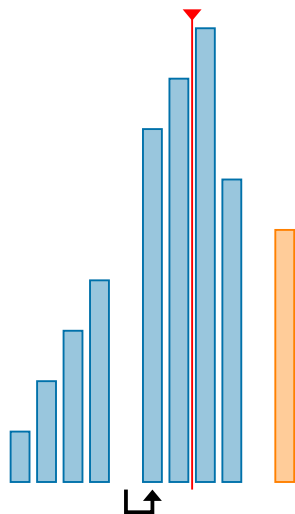
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 7$: Platz für Einfügung – fertig

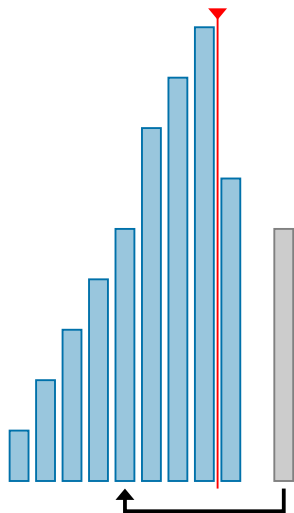
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 7$: Endzustand

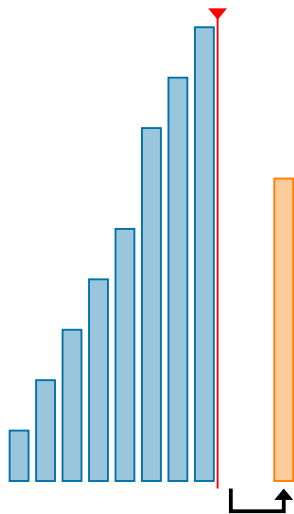
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 8$: Anfangszustand

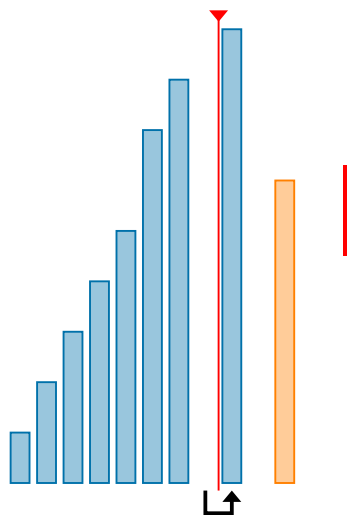
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 8$: Platz für Einfügung

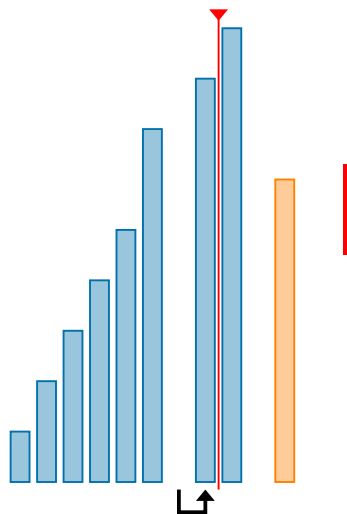
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



$j = 8$: Platz für Einfügung

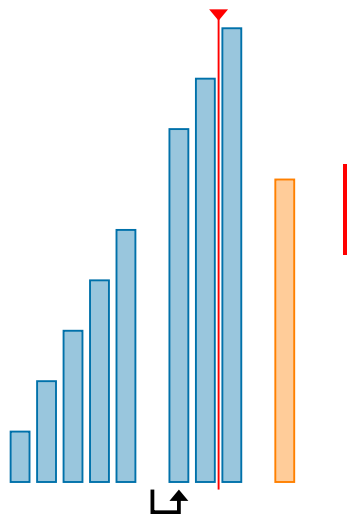
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



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

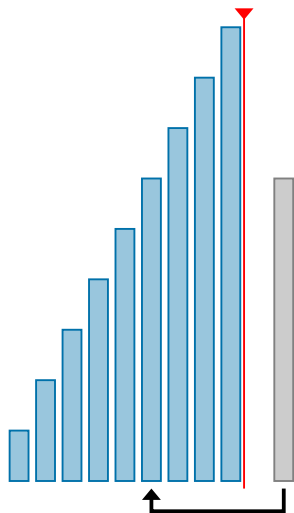
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0 \ \&\& \ A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```


Insertion Sort



$j = 8$: Endzustand

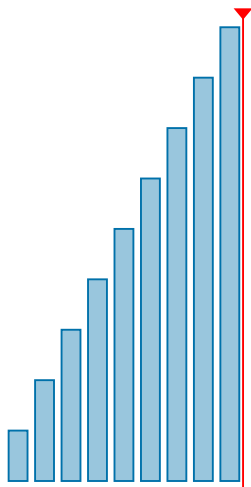
Input: Feld $A[0..n-1]$ von $n \geq 0$ natürlichen Zahlen

Output: Feld A aufsteigend sortiert

InsertionSort(A):

```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Insertion Sort



Endzustand

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

Output: Feld A aufsteigend sortiert

InsertionSort(A):

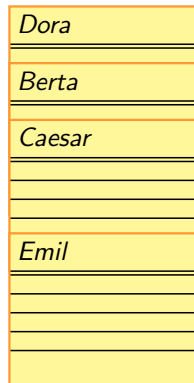
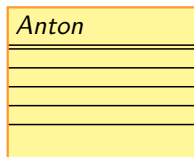
```
for  $j = 1$  to  $n - 1$  {  
     $key = A[j]$ ;  
     $i = j - 1$ ;  
    while ( $i \geq 0$  &&  $A[i] > key$ ) {  
         $A[i + 1] = A[i]$ ;  
         $i = i - 1$ ;  
    }  
     $A[i + 1] = key$ ;  
}
```

Selection Sort

<i>Dora</i>
<i>Berta</i>
<i>Caesar</i>
<i>Anton</i>
<i>Emil</i>

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

Selection Sort



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

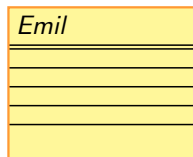
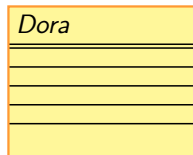
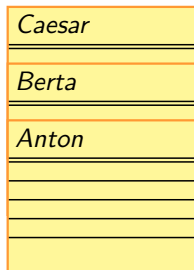
Selection Sort

<i>Berta</i>
<i>Anton</i>

<i>Dora</i>
<i>Caesar</i>
<i>Emil</i>

- ▶ 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

<i>Dora</i>
<i>Caesar</i>
<i>Berta</i>
<i>Anton</i>

<i>Emil</i>

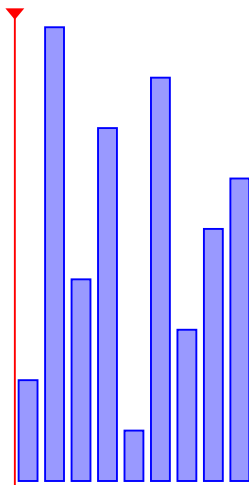
- ▶ Gustav sucht im unsortierten Stapel das „kleinste“ Element und findet *Emil*

Selection Sort

<i>Emil</i>
<i>Dora</i>
<i>Caesar</i>
<i>Berta</i>
<i>Anton</i>

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

Selection Sort

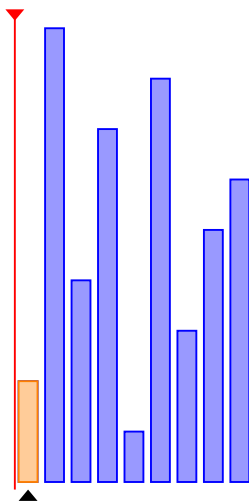


$j = 0$: Anfangszustand

```
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

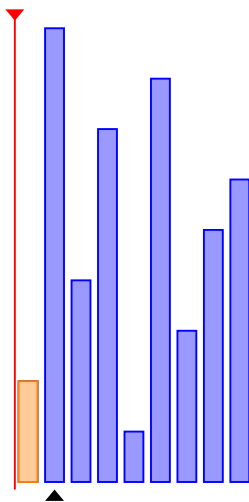


$j = 0$: Suche bei 0

```
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

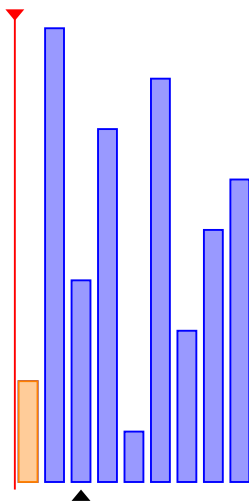


$j = 0$: Suche bei 1

```
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

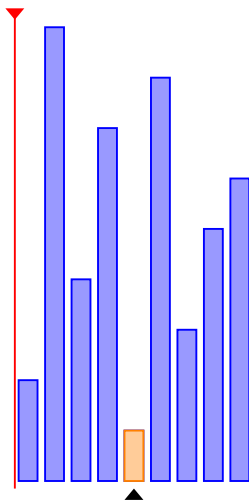


$j = 0$: Suche bei 2

```
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

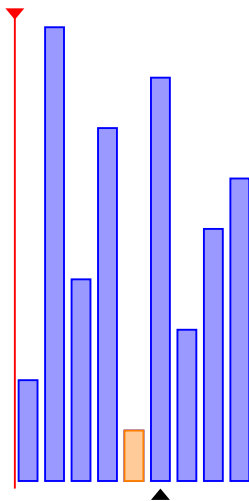


$j = 0$: Suche bei 4

```
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

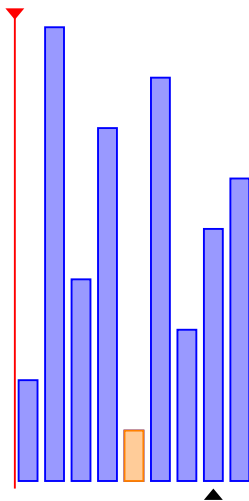


$j = 0$: Suche bei 5

```
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

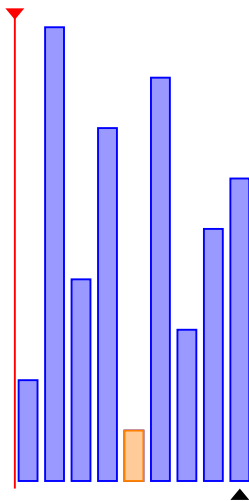


$j = 0$: Suche bei 7

```
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

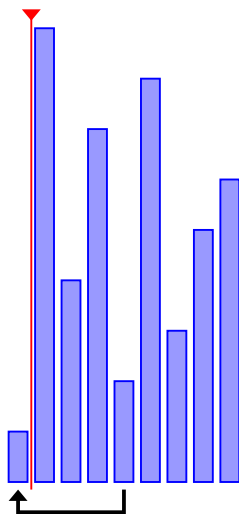


$j = 0$: Suche bei 8

```
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

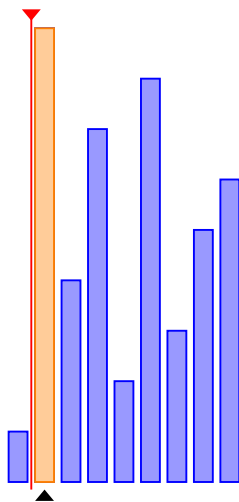


$j = 1$: Anfangszustand

```
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

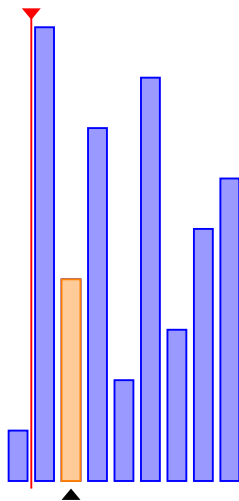


$j = 1$: Suche bei 1

```
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

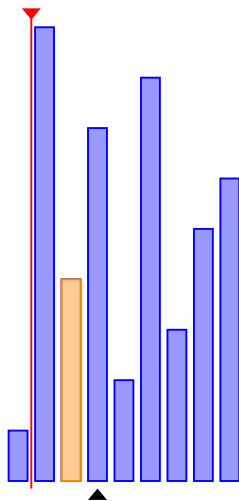


$j = 1$: Suche bei 2

```
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

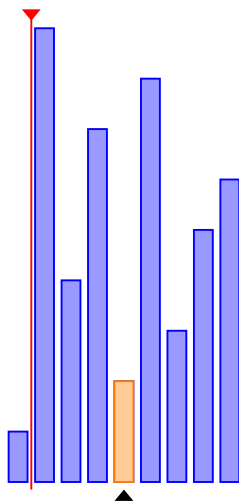


$j = 1$: Suche bei 3

```
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

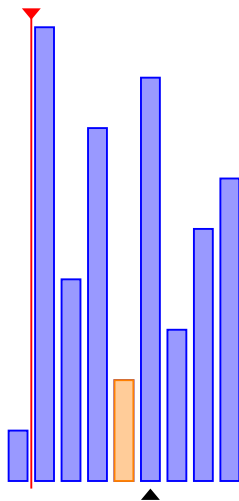


$j = 1$: Suche bei 4

```
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

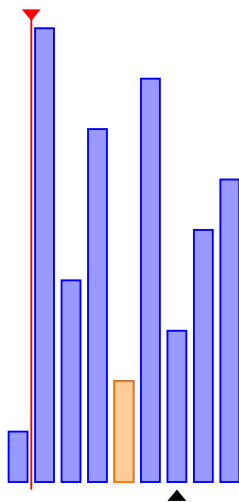


$j = 1$: Suche bei 5

```
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

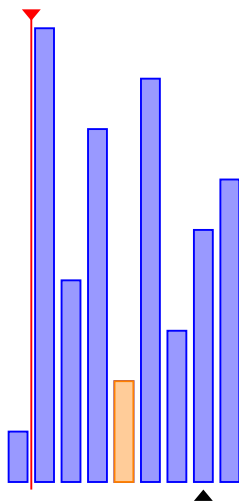


$j = 1$: Suche bei 6

```
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

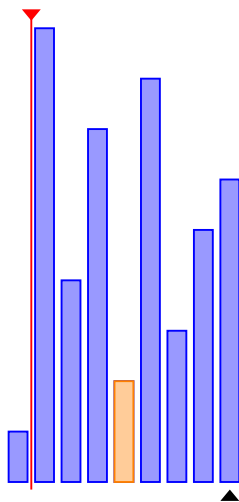


$j = 1$: Suche bei 7

```
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

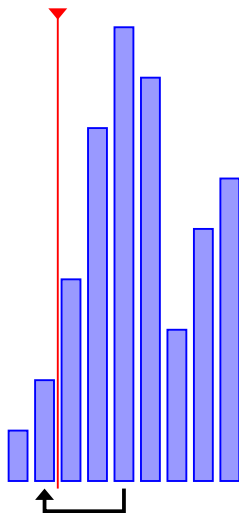


$j = 1$: Suche bei 8

```
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

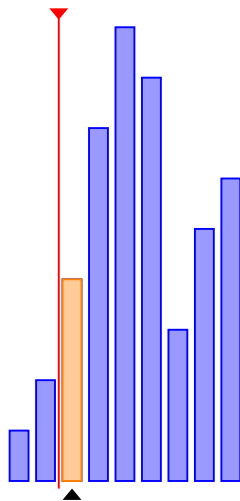


$j = 2$: Anfangszustand

```
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

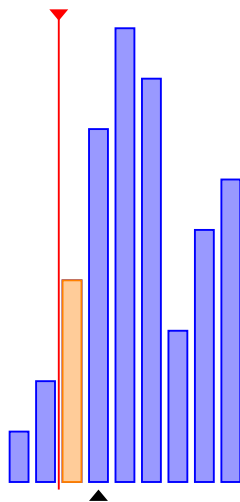


$j = 2$: Suche bei 2

```
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

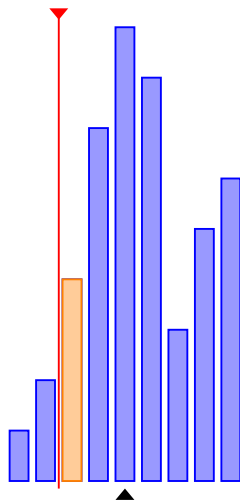


$j = 2$: Suche bei 3

```
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

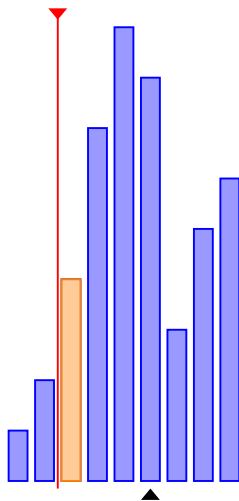


$j = 2$: Suche bei 4

```
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

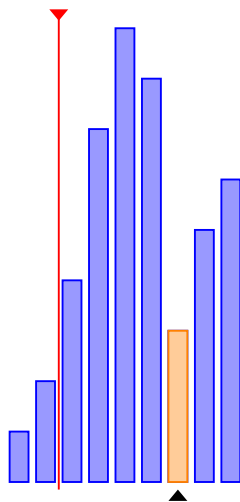


$j = 2$: Suche bei 5

```
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

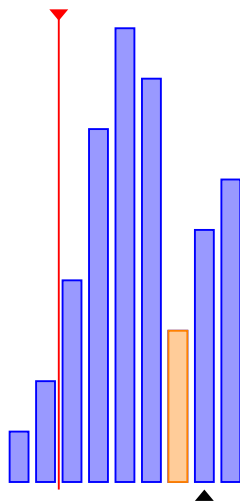


$j = 2$: Suche bei 6

```
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

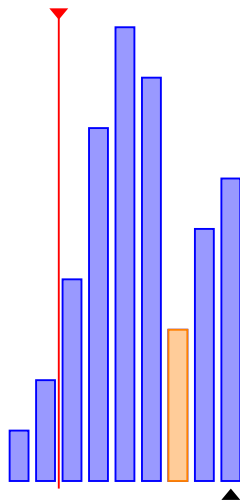


$j = 2$: Suche bei 7

```
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

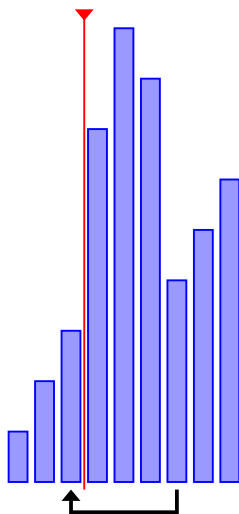


$j = 2$: Suche bei 8

```
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

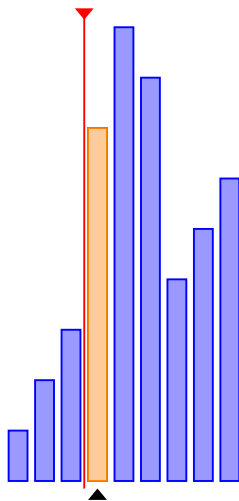


$j = 3$: Anfangszustand

```
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

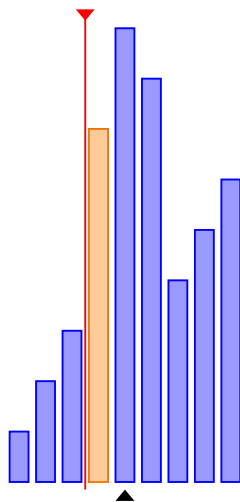


$j = 3$: Suche bei 3

```
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

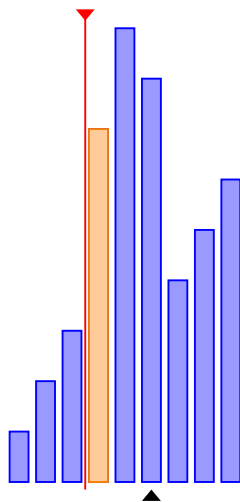


$j = 3$: Suche bei 4

```
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

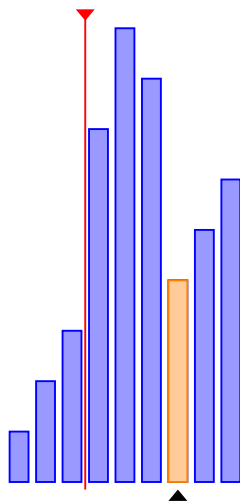


$j = 3$: Suche bei 5

```
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

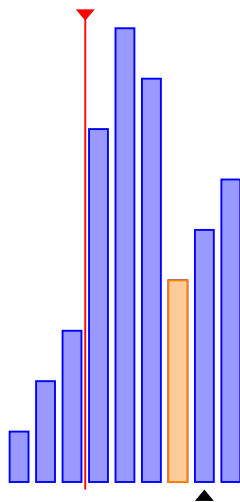


$j = 3$: Suche bei 6

```
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

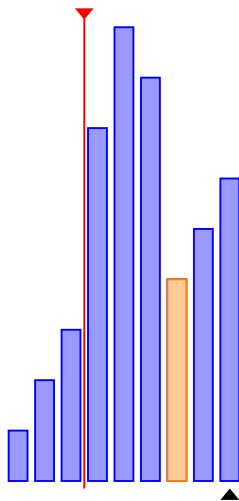


$j = 3$: Suche bei 7

```
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

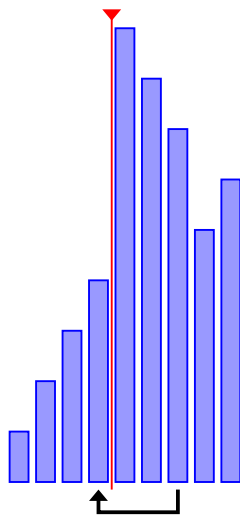


$j = 3$: Suche bei 8

```
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

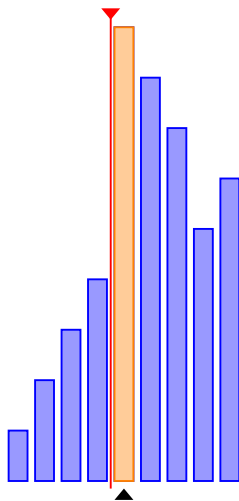


$j = 4$: Anfangszustand

```
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

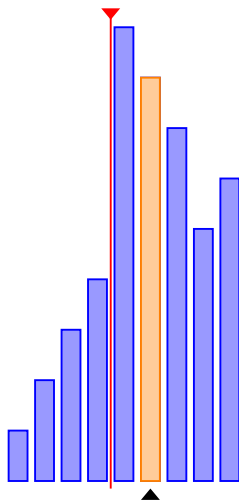


$j = 4$: Suche bei 4

```
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

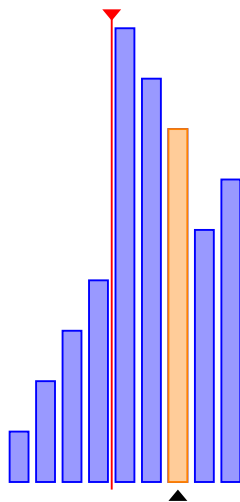


$j = 4$: Suche bei 5

```
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

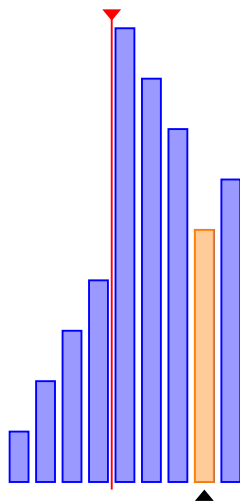


$j = 4$: Suche bei 6

```
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

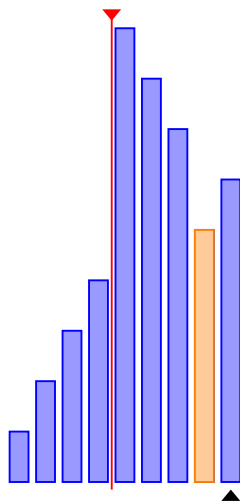


$j = 4$: Suche bei 7

```
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

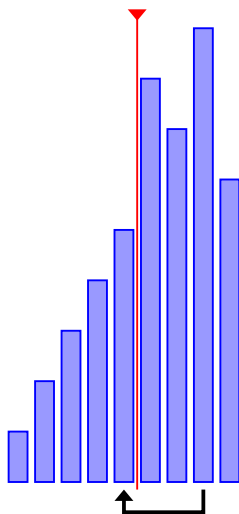


$j = 4$: Suche bei 8

```
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

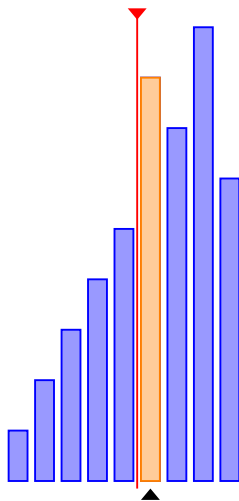


$j = 5$: Anfangszustand

```
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

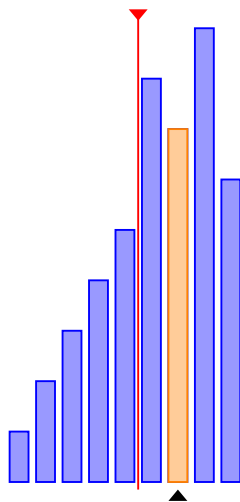


$j = 5$: Suche bei 5

```
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

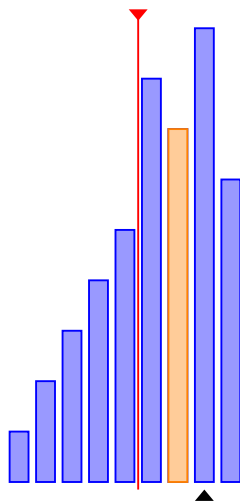


$j = 5$: Suche bei 6

```
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

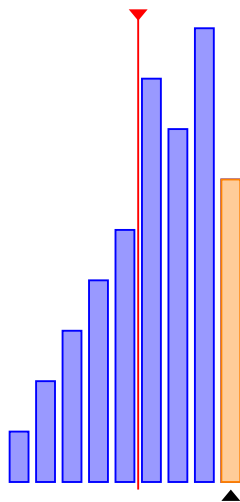


$j = 5$: Suche bei 7

```
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

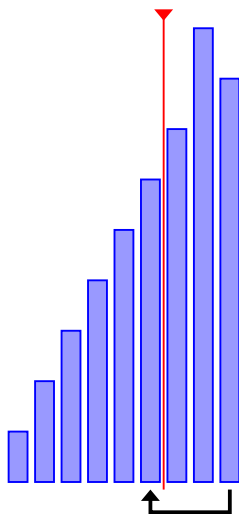


$j = 5$: Suche bei 8

```
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

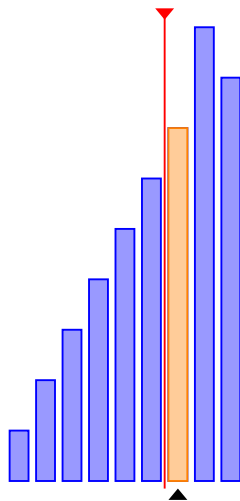


$j = 6$: Anfangszustand

```
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

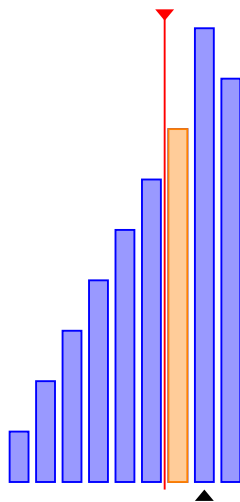


$j = 6$: Suche bei 6

```
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

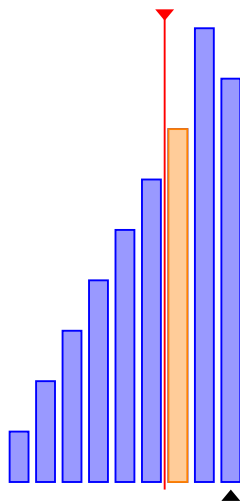


$j = 6$: Suche bei 7

```
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

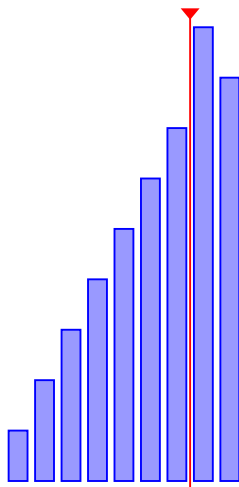


$j = 6$: Suche bei 8

```
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

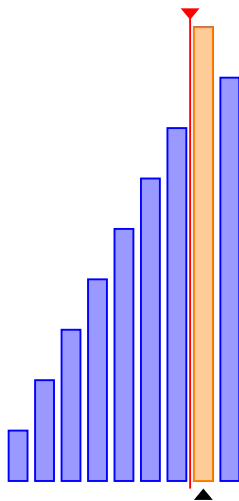


$j = 7$: Anfangszustand

```
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

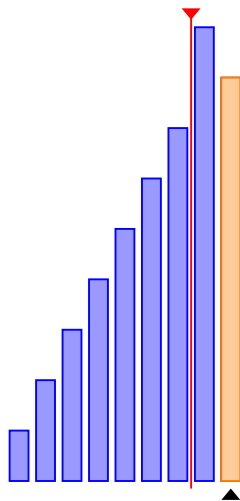


$j = 7$: Suche bei 7

```
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

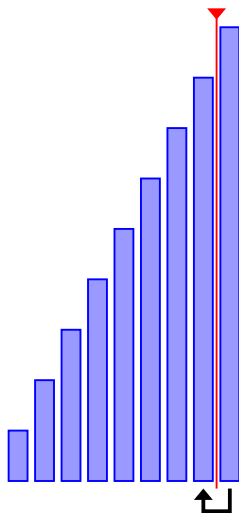


$j = 7$: Suche bei 8

```
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

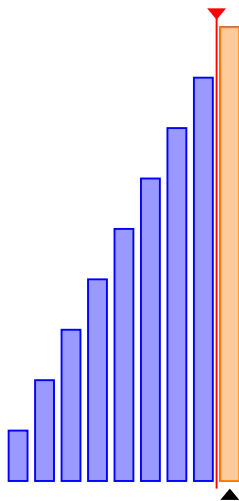


$j = 8$: Anfangszustand

```
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

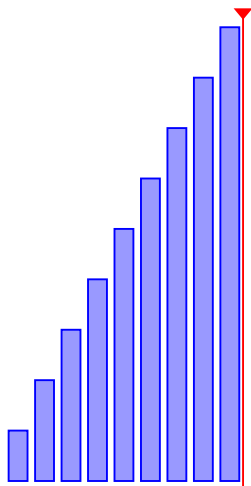


$j = 8$: Suche bei 8

```
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);}
    }
}
```

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  $n - 1$  {  
     $i = \text{IndexOfMin}(A, j)$ ;  
    if ( $j \neq i$ ) {  
        Swap( $A, i, j$ );  
    }  
}
```

$$VOR_{\text{sort}} = n \geq 0$$

$$NACH_{\text{sort}} = A \text{ enthält Permutation der Originaldaten} \wedge \\ A[0] \leq A[1] \leq \dots \leq A[n-1]$$

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 ( $A[k] < A[i]$ ) {  
             $i = k;$   
        }  
    }  
}
```

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

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

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):

$k = A[j];$

$A[j] = A[i];$

$A[i] = k;$

$$VOR_{\text{swap}} = n > 0 \wedge 0 \leq i, j < n \wedge A[i] = a \wedge A[j] = b$$

$$NACH_{\text{swap}} = A[i] = b \wedge A[j] = a \wedge$$

$$A[k] \text{ unverändert } \forall k \in \{0, \dots, n-1\} \setminus \{i, j\}$$

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

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 ($A[j..n-1]$)

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

- ▶ Unterschied zur Verifikation des Insertion Sort!

Korrektheit von Selection Sort

```
{VOR}
  j = 0;
{C1}
  while (j < n) {
    i = IndexOfMin(A, j);
    if (j ≠ i) {
      Swap(A, i, j);
    }
    j = j + 1;
  }
{NACH}
```

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

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

Korrektheit von Selection Sort

```
{VOR}
  j = 0;
  while (j < n) {
{C2}
    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 $j < n$, damit auch die Vorbedingung von **IndexOfMin**.

$$C_2 = 0 \leq j < n \wedge P \wedge \dots$$

Korrektheit von Selection Sort

{VOR}

$j = 0;$

while ($j < n$) {

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

{C₃}

if ($j \neq i$) {

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

}

$j = j + 1;$

}

{NACH}

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

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

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 **Swapen**.

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

Korrektheit von Selection Sort

{VOR}

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

{C₅}

}

{NACH}

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

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

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

$$j \geq n \wedge P \Rightarrow \text{NACH}$$

Korrektheit von Selection Sort

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