

Algorithmen, Pseudocode, Sortieren (Teil 2)

Manfred Hauswirth | Open Distributed Systems | Einführung in die Programmierung, WS 23/24

Rückblick

- VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
- VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort
- VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort**
- VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortiervverfahren
- VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue
- VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen
- VL 6 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C
- VL 7 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort
- VL 8 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise
- VL 9 „Prioritätenslangen/Halden/Heaps“: Heap Sort, Binärer Heap, Heap Operationen
- VL 10 „Fortgeschrittene Sortiervverfahren“: Quick Sort, Radix Sort
- VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung
- VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem
- VL 13 „Q & A“: Offene Vorlesung/Wiederholung

Weitere Sortieralgorithmen

- Sortieren ist ein sehr intensiv untersuchtes Problem
- Es gibt eine große Zahl von Algorithmen mit jeweils verschiedenen Varianten
- Generell ordnet man Sortierverfahren in zwei Gruppen:
 - 1) Vergleichende Sortierverfahren
 - A. Einfache Sortierverfahren
 - B. Fortgeschrittene Sortierverfahren
 - 2) Nicht vergleichende Sortierverfahren

- Einfache vergleichende Sortierverfahren
 - Sortieren durch Einfügen (insertion sort)
 - Sortieren durch Auswählen (selection sort)
 - Sortieren durch Vertauschen (bubble sort)
- Fortgeschrittene vergleichende Sortierverfahren
 - Sortieren durch Gruppieren (quick sort)
 - Sortieren durch Mischen (merge sort)
- Nicht vergleichende Sortierverfahren
 - Sortieren durch Zählen (count sort)
 - Sortieren durch Fachverteilen (radix sort)

Problem: Sortieren

- Eingabe: Folge von n Zahlen (a_1, \dots, a_n)
- Ausgabe: Permutation (a'_1, \dots, a'_n) von (a_1, \dots, a_n) , so dass
$$a'_1 \leq a'_2 \leq \dots \leq a'_n$$

Beispiel:

- Eingabe: 15, 7, 3, 18, 8, 4
- Ausgabe: 3, 4, 7, 8, 15, 18

Wiederholung: Insertion Sort

Insertion Sort

InsertionSort(Array A)

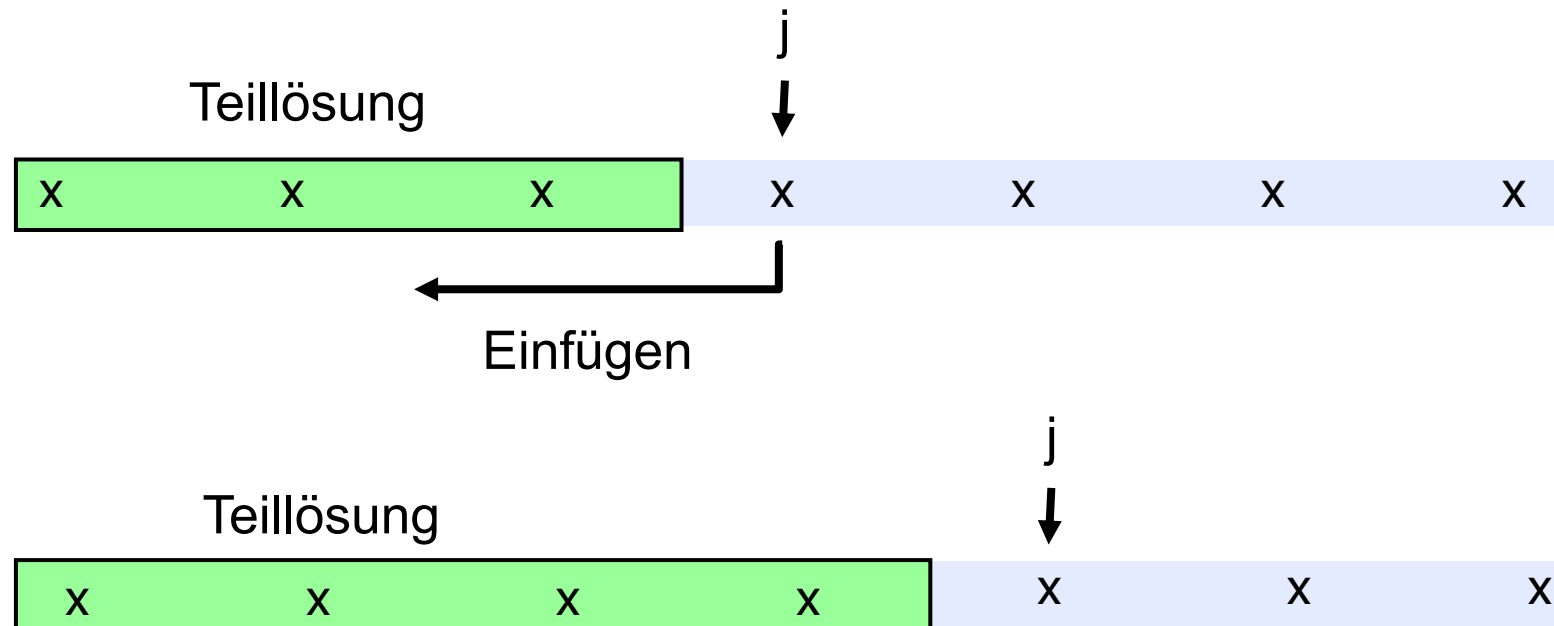
```
1. for j  $\leftarrow$  2 to length(A) do
2.     key  $\leftarrow$  A[j]
3.     i  $\leftarrow$  j-1
4.     while i>0 and A[i]>key do
5.         A[i+1]  $\leftarrow$  A[i]
6.         i  $\leftarrow$  i-1
7.     A[i+1]  $\leftarrow$  key
```

Idee Insertion Sort:

- Die ersten $j-1$ Elemente sind sortiert (zu Beginn $j=2$)
- Innerhalb eines Schleifendurchlaufs wird das j -te Element in die sortierte Folge eingefügt
- Am Ende ist die gesamte Folge sortiert

Sortieren durch Einfügen (Insertion Sort)

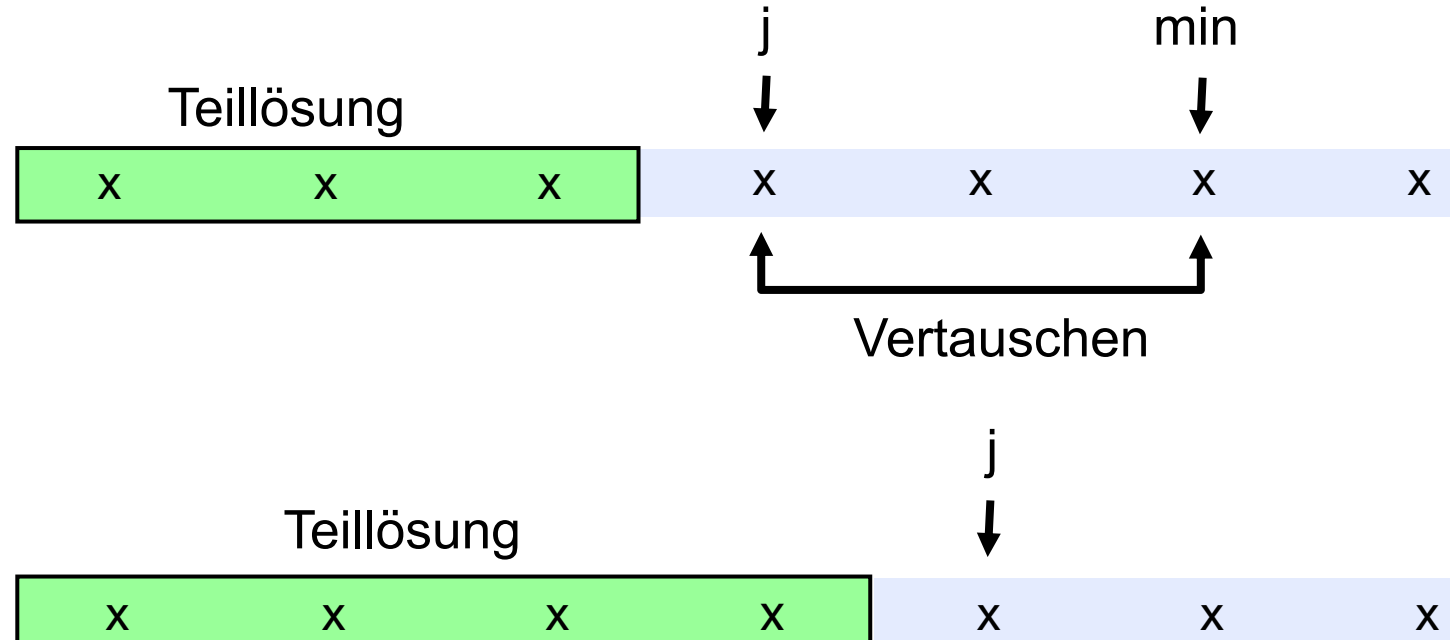
- Arbeitsweise



Selection Sort

Sortieren durch Auswählen (Selection Sort)

- Minimum der verbleibenden unsortierten Restfolge wird direkt ausgewählt und mit dem aktuellen Element vertauscht



Selection Sort

SelectionSort(Array A)

```
1. for j  $\leftarrow$  1 to length(A) - 1 do  
2.     min  $\leftarrow$  j  
3.     for i  $\leftarrow$  j + 1 to length(A) do  
4.         if A[i] < A[min] then min  $\leftarrow$  i  
5.     h  $\leftarrow$  A[min]  
6.     A[min]  $\leftarrow$  A[j]  
7.     A[j]  $\leftarrow$  h
```

Idee Selection Sort:

- Die ersten $j-1$ Elemente sind sortiert (zu Beginn $j=1$)
- Innerhalb eines Schleifendurchlaufs wird das j -kleinste Element (entspricht dem kleinsten aus dem Rest) an die sortierte Folge „angehängt“
- Am Ende ist die gesamte Folge sortiert

Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

Idee Selection Sort:

- *Die ersten $j-1$ Elemente sind sortiert (zu Beginn $j=1$)*
- *Innerhalb eines Schleifendurchlaufs wird das j -kleinste Element (entspricht dem kleinsten aus dem Rest) an die sortierte Folge „angehängt“*
- *Am Ende ist die gesamte Folge sortiert*

Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel

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

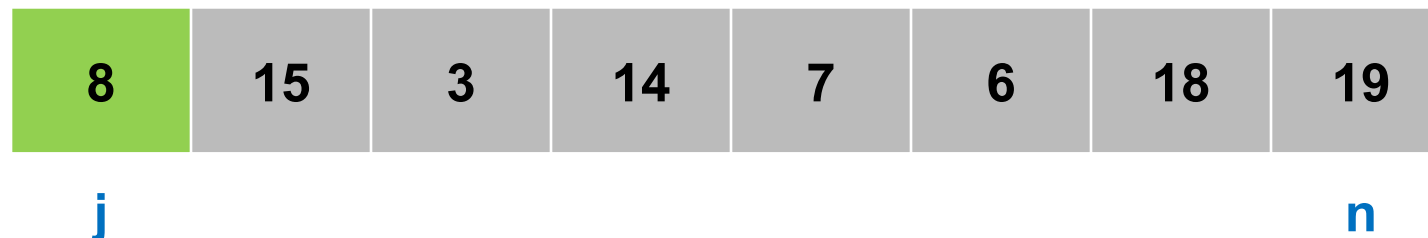
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



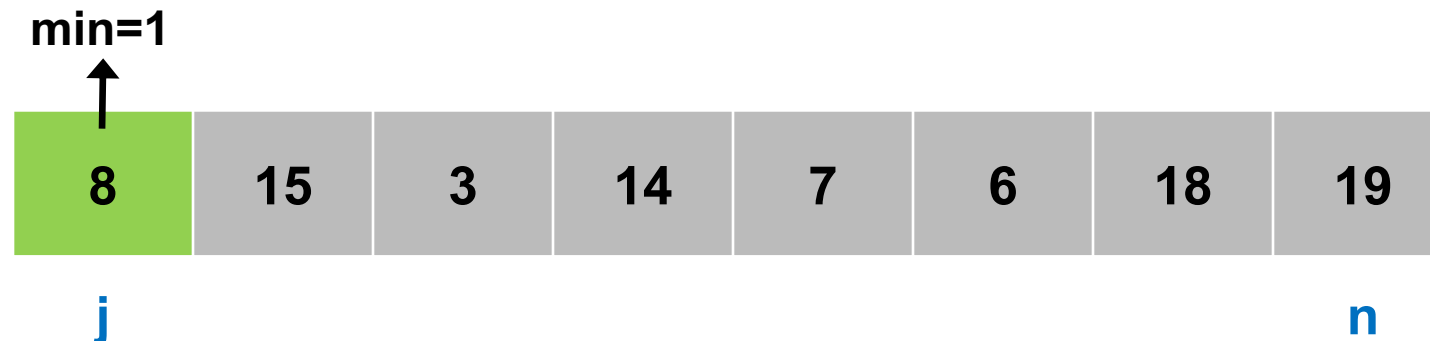
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



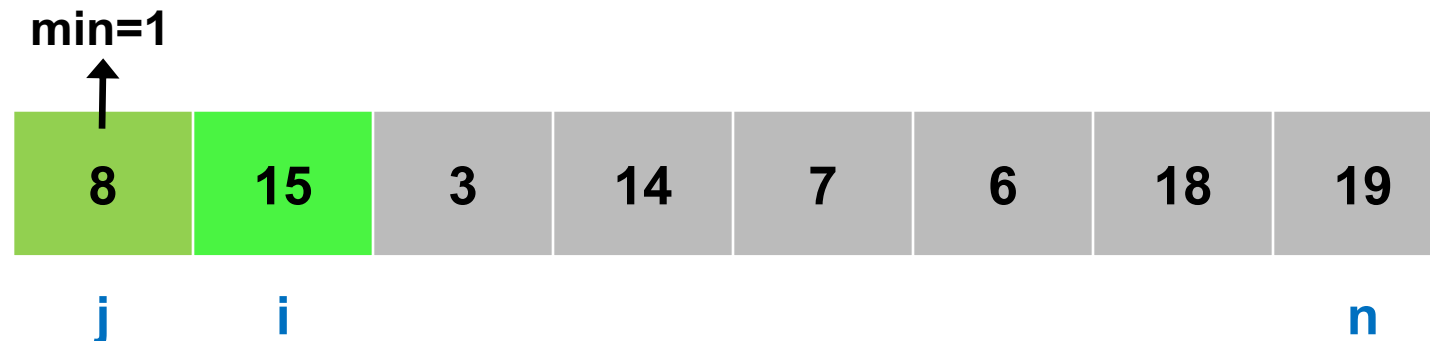
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



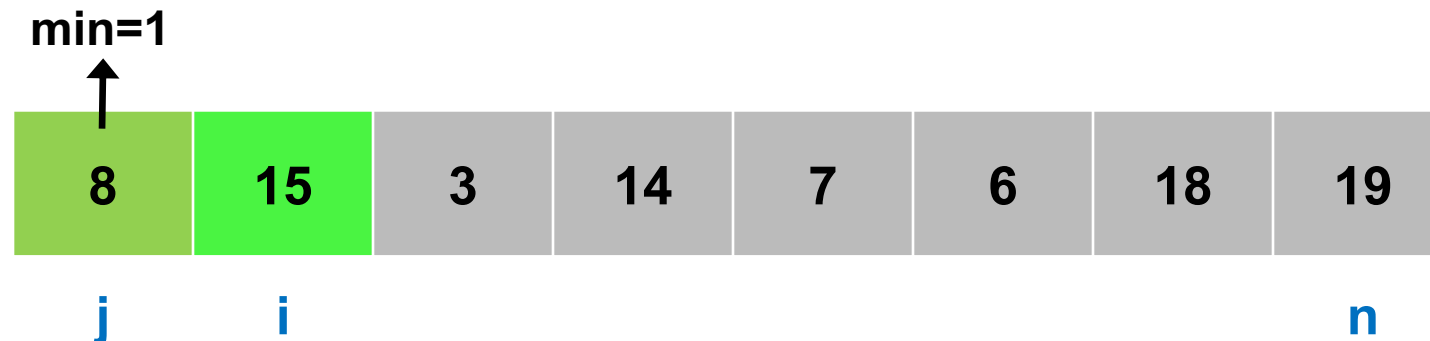
Selection Sort – mit swap

SelectionSort(Array A)

```
1. for j ← 1 to length(A) - 1 do  
2.   min ← j  
3.   for i ← j + 1 to length(A) do  
4.     if A[i] < A[min] then min ← i  
5.   swap(A, min, j)
```

- Eingabegröße n
- length(A) = n

Beispiel



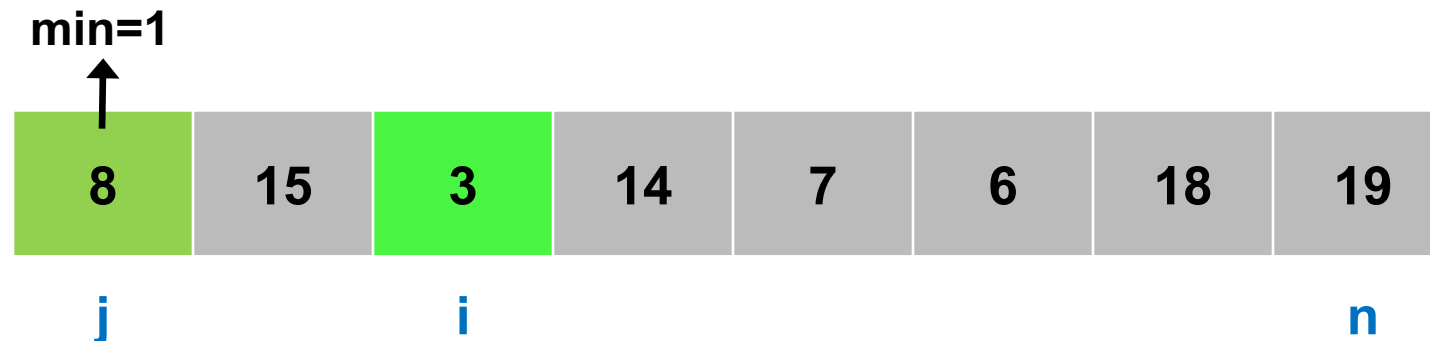
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



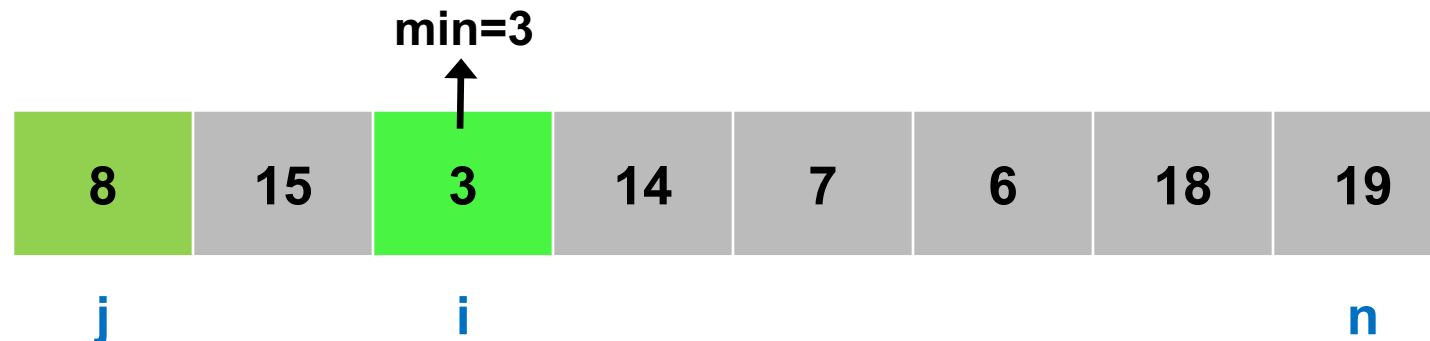
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

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

2. $\text{min} \leftarrow j$

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

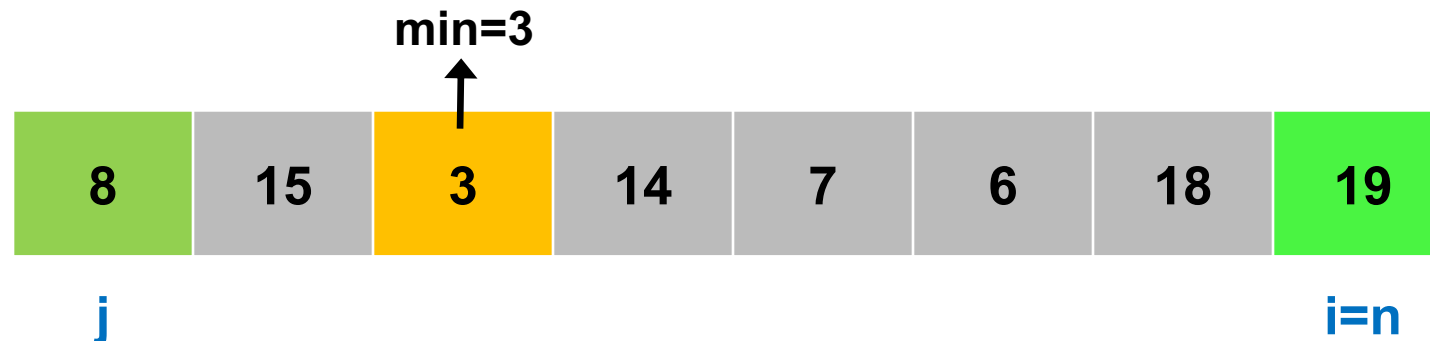
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$

5. $\text{swap}(A, \text{min}, j)$

➤ Eingabegröße n

➤ $\text{length}(A) = n$

Beispiel



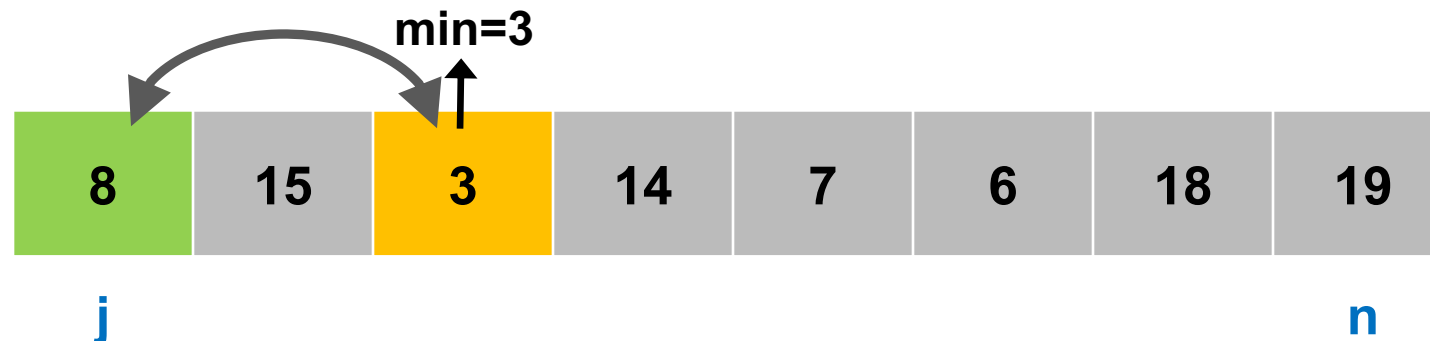
Selection Sort – mit swap

SelectionSort(Array A)

```
1. for j  $\leftarrow$  1 to length(A) - 1 do  
2.   min  $\leftarrow$  j  
3.   for i  $\leftarrow$  j + 1 to length(A) do  
4.     if A[i] < A[min] then min  $\leftarrow$  i  
5.   swap(A, min, j)
```

- Eingabegröße n
- length(A) = n

Beispiel



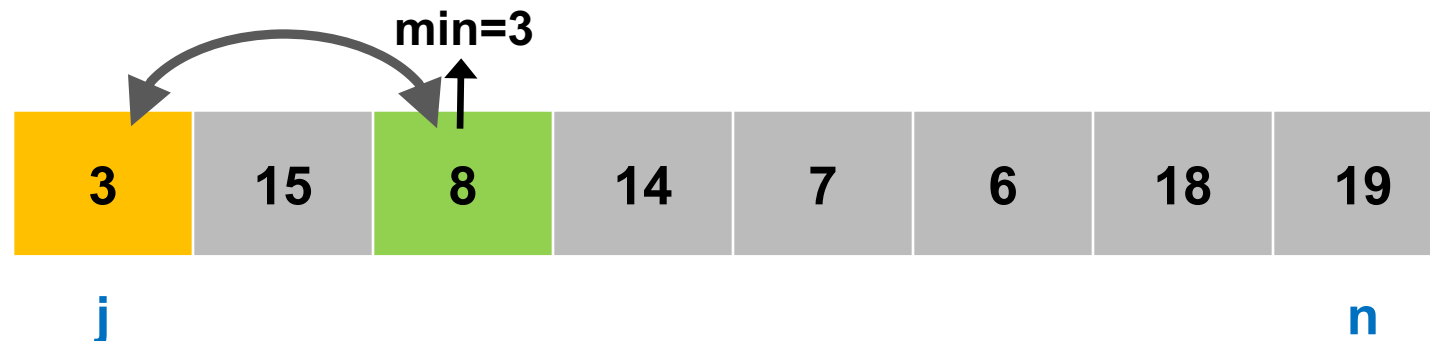
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



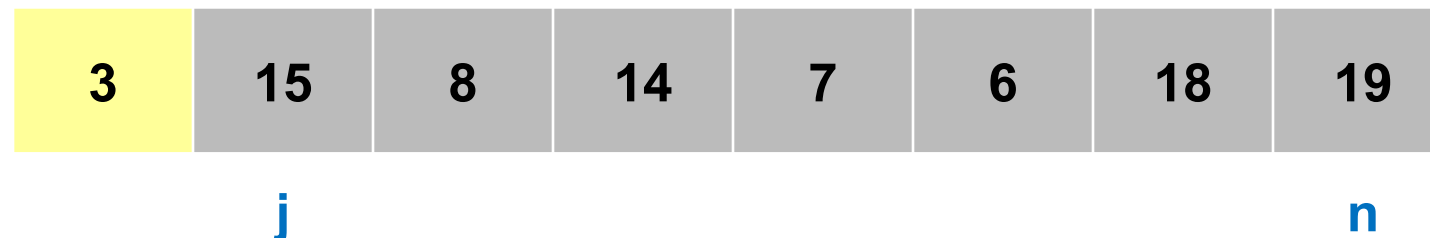
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



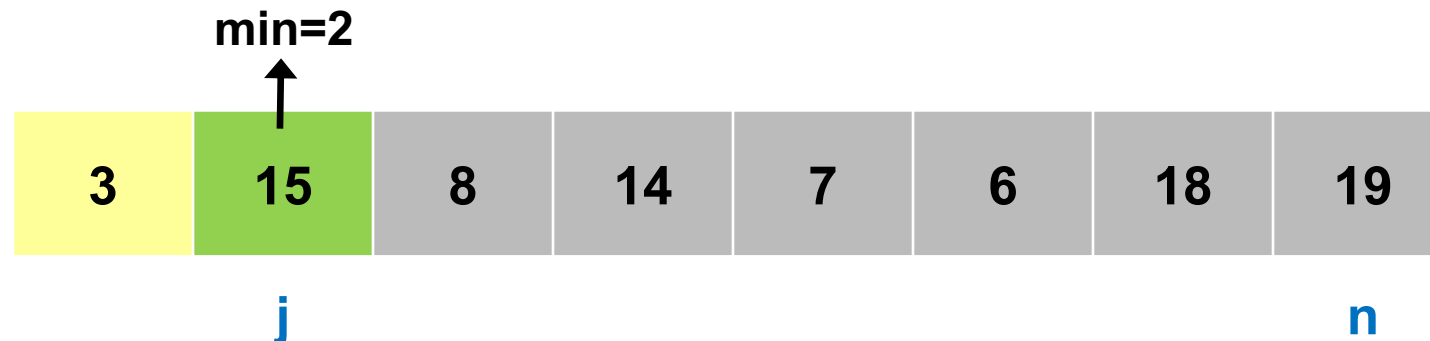
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

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

2. $\text{min} \leftarrow j$

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

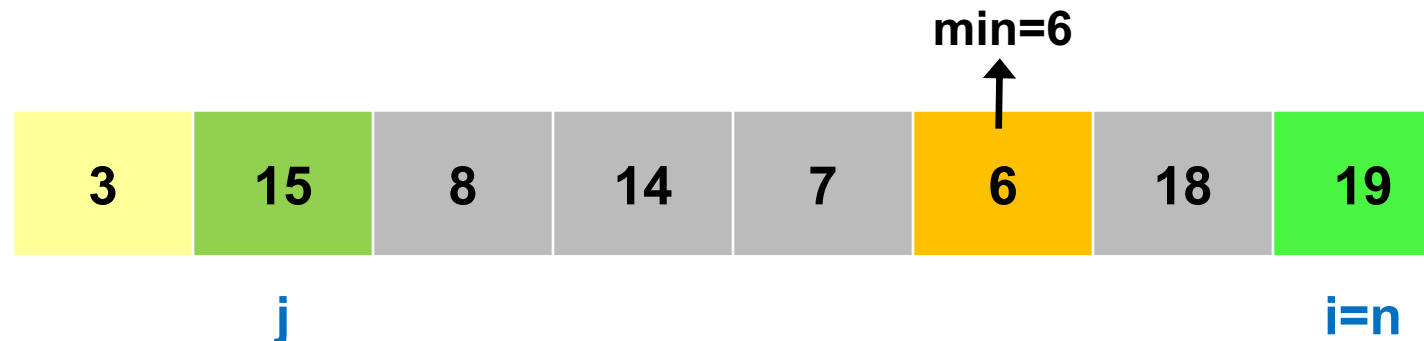
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$

5. $\text{swap}(A, \text{min}, j)$

➤ Eingabegröße n

➤ $\text{length}(A) = n$

Beispiel



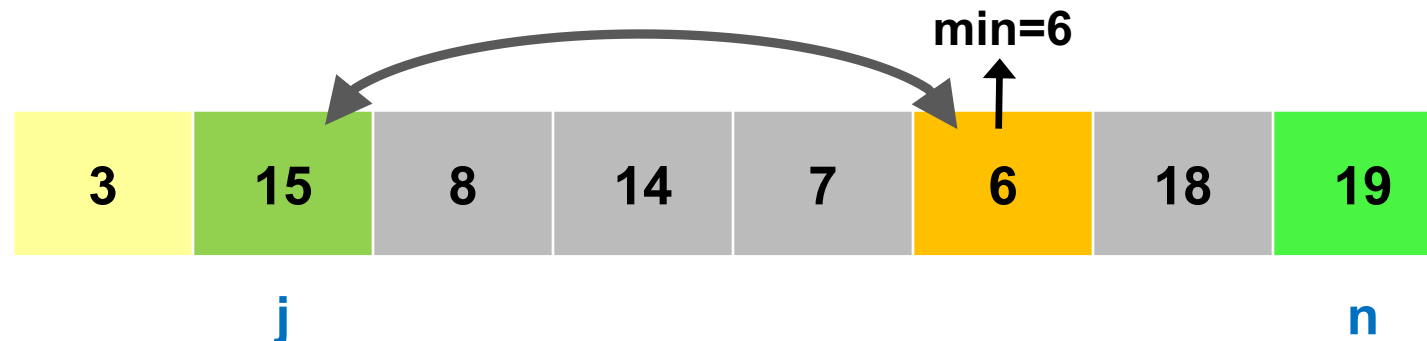
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



Selection Sort – mit swap

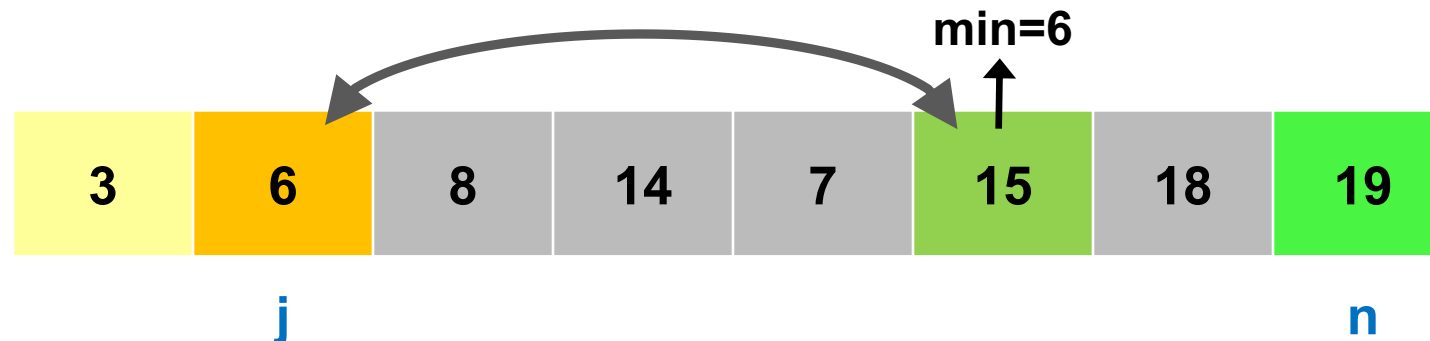
SelectionSort(Array A)

```
1. for j  $\leftarrow$  1 to length(A) - 1 do  
2.   min  $\leftarrow$  j  
3.   for i  $\leftarrow$  j + 1 to length(A) do  
4.     if A[i] < A[min] then min  $\leftarrow$  i  
5.   swap(A, min, j)
```

➤ Eingabegröße n

➤ length(A) = n

Beispiel



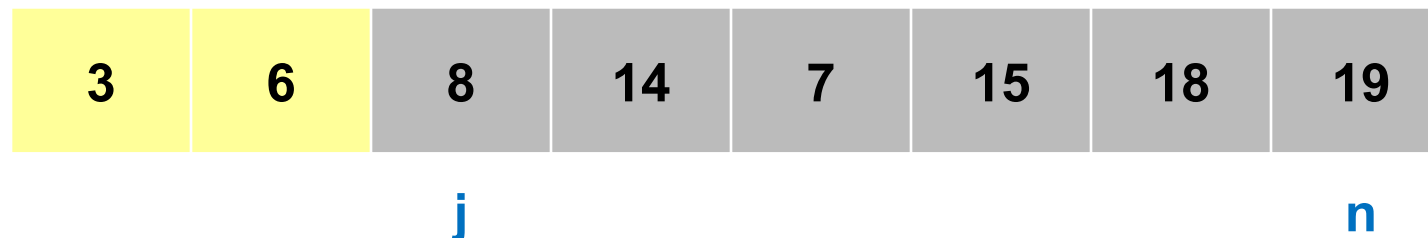
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



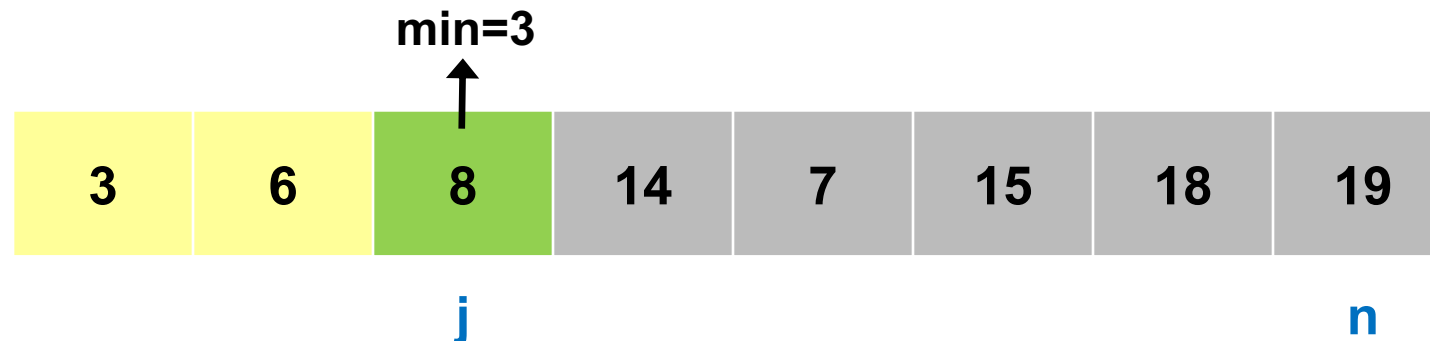
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

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

2. $\text{min} \leftarrow j$

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

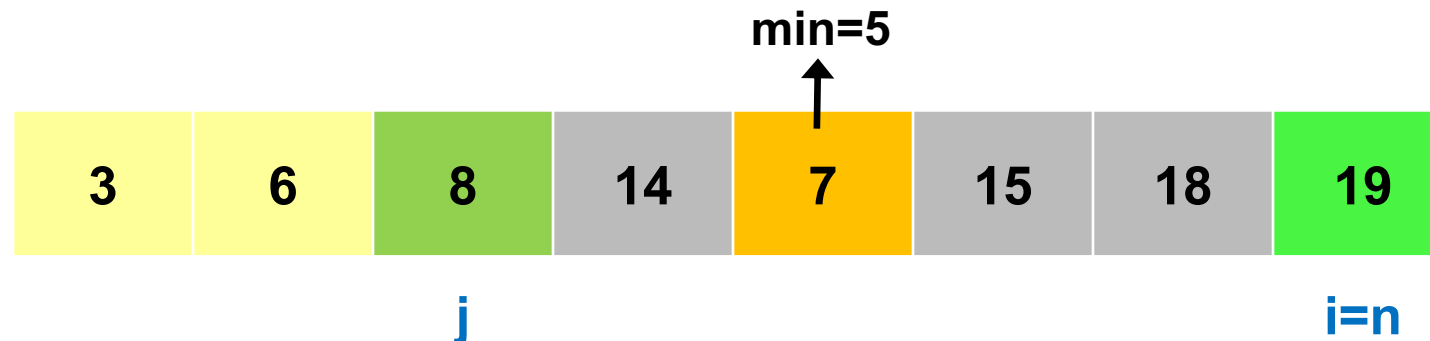
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$

5. $\text{swap}(A, \text{min}, j)$

➤ Eingabegröße n

➤ $\text{length}(A) = n$

Beispiel



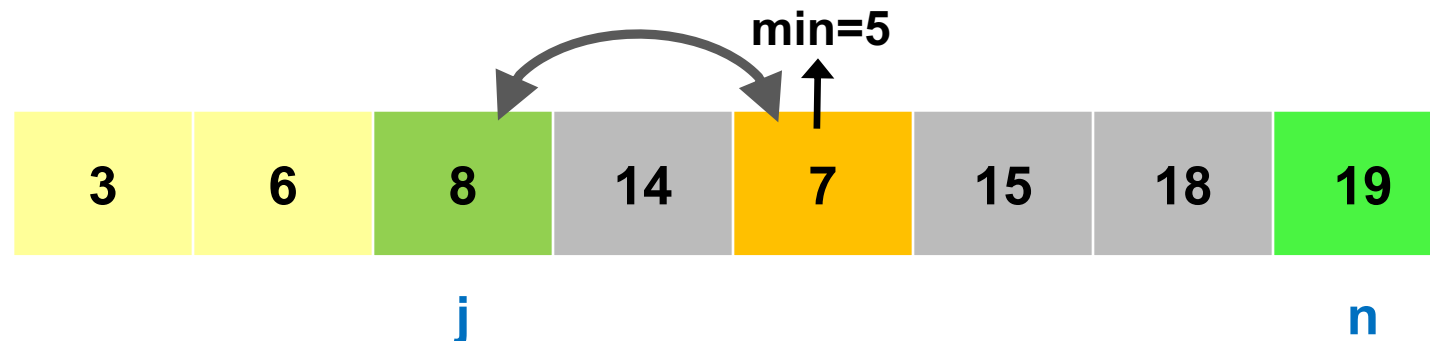
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



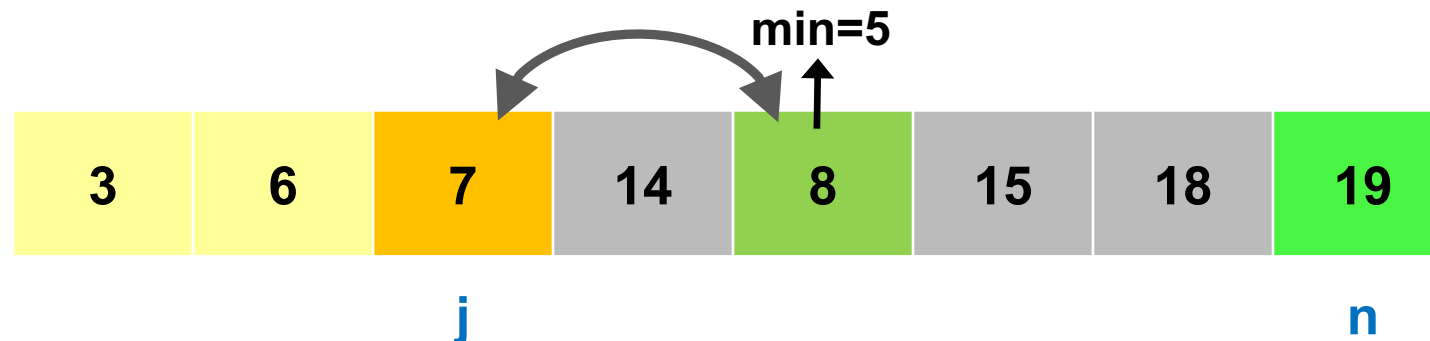
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



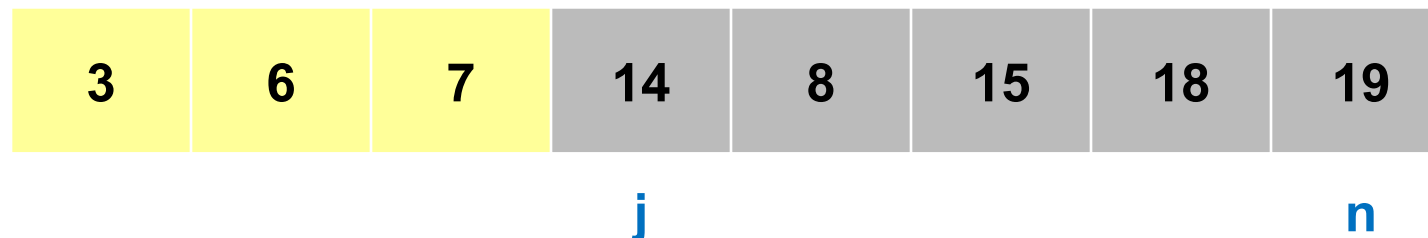
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$

Beispiel



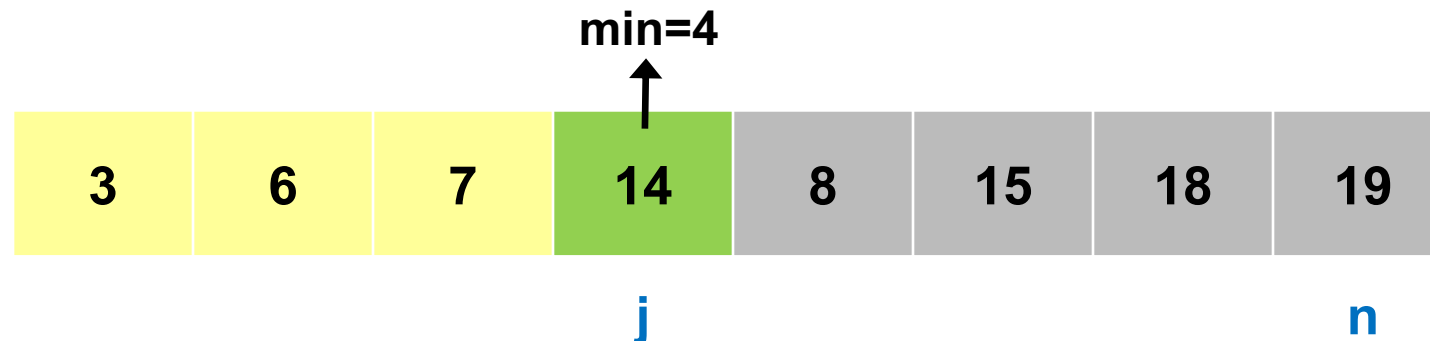
Selection Sort – mit swap

SelectionSort(Array A)

1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. $\text{swap}(A, \text{min}, j)$

- Eingabegröße n
- $\text{length}(A) = n$
- Finde das kleinste Element
- In $A[j+1 \dots n]$
- Tausche das aktuelle Element mit „dem kleinsten“

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

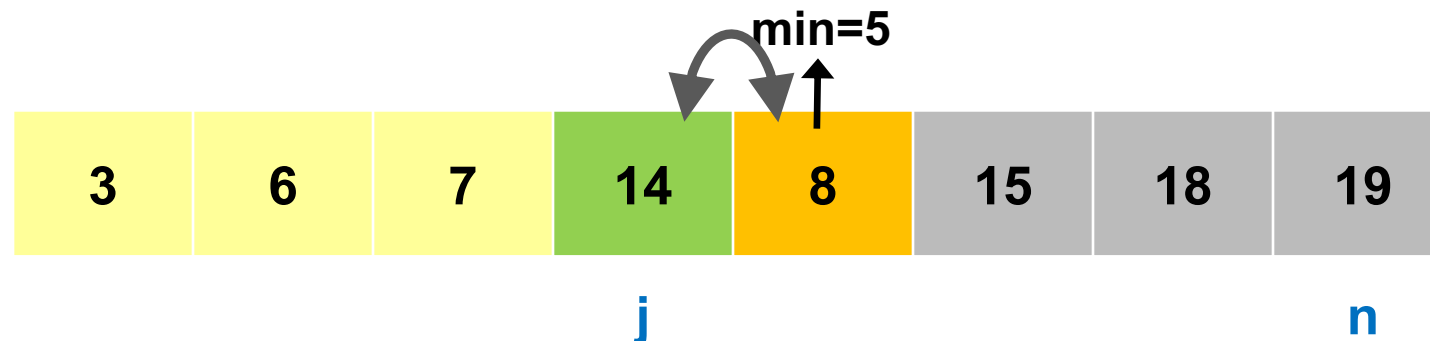
1. **for** $j \leftarrow 1$ **to** $\text{length}(A) - 1$ **do**
2. $\text{min} \leftarrow j$
3. **for** $i \leftarrow j + 1$ **to** $\text{length}(A)$ **do**
4. **if** $A[i] < A[\text{min}]$ **then** $\text{min} \leftarrow i$
5. swap(A, min, j)

- Eingabegröße n
- $\text{length}(A) = n$

- Finde das kleinste Element
- In $A[j+1 \dots n]$

- Tausche das aktuelle Element mit „dem kleinsten“

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

```
1. for j ← 1 to length(A) - 1 do  
2.   min ← j  
3.   for i ← j + 1 to length(A) do  
4.     if A[i] < A[min] then min ← i  
5.   swap(A, min, j)
```

➤ Eingabegröße n

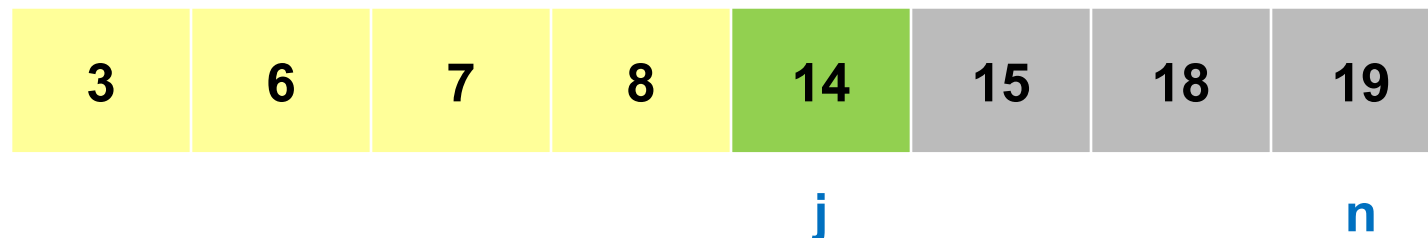
➤ length(A) = n

➤ Finde das kleinste Element

➤ In A[j+1...n]

➤ Tausche das aktuelle Element mit „dem kleinsten“

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

```

1. for j ← 1 to length(A) - 1 do
2.   min ← j
3.   for i ← j + 1 to length(A) do
4.     if A[i] < A[min] then min ← i
5.   swap(A, min, j)
  
```

➤ Eingabegröße n

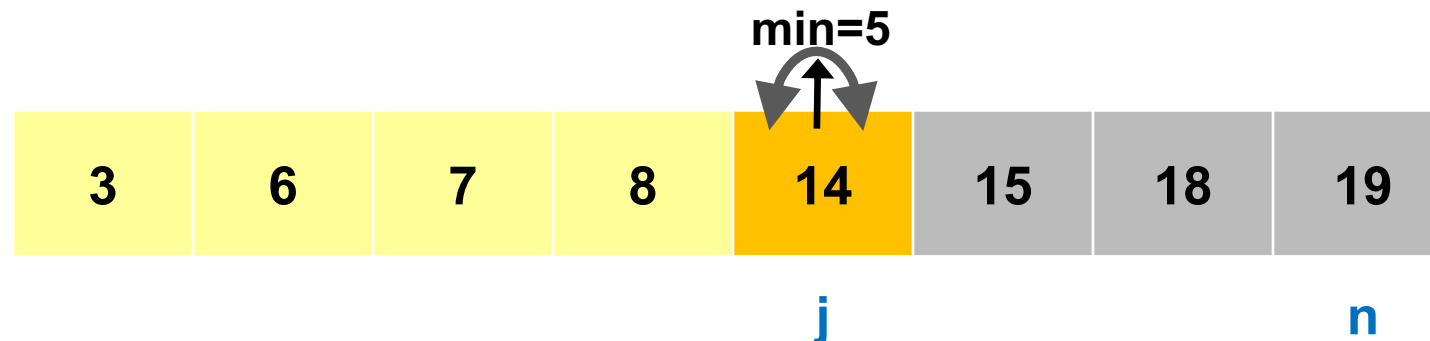
➤ length(A) = n

➤ Finde das kleinste Element

➤ In A[j+1...n]

➤ Tausche das aktuelle Element mit „dem kleinsten“

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

```
1. for j ← 1 to length(A) - 1 do  
2.   min ← j  
3.   for i ← j + 1 to length(A) do  
4.     if A[i] < A[min] then min ← i  
5.   swap(A, min, j)
```

➤ Eingabegröße n

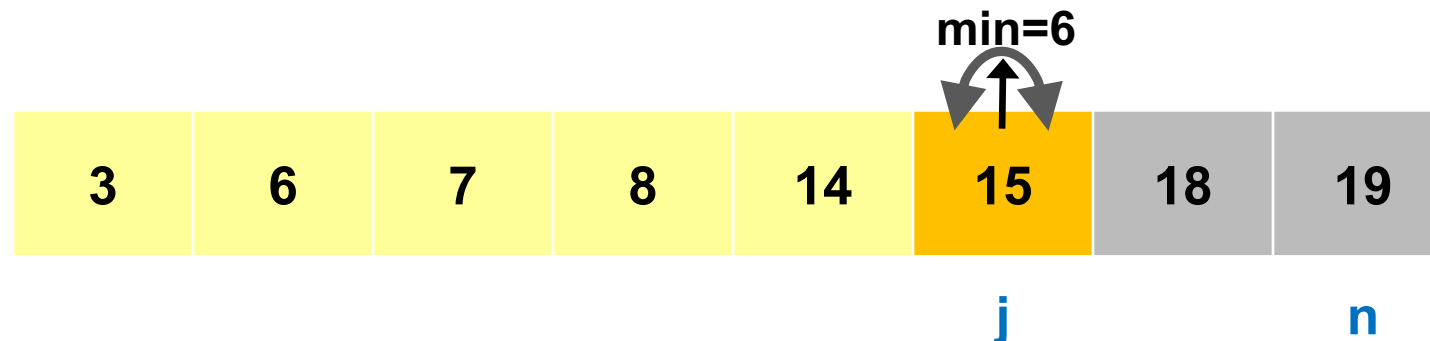
➤ length(A) = n

➤ Finde das kleinste Element

➤ In A[j+1...n]

➤ Tausche das aktuelle Element mit „dem kleinsten“

Beispiel



Selection Sort – mit swap

SelectionSort(Array A)

```
1. for j ← 1 to length(A) - 1 do  
2.   min ← j  
3.   for i ← j + 1 to length(A) do  
4.     if A[i] < A[min] then min ← i  
5.   swap(A, min, j)
```

➤ Eingabegröße n

➤ length(A) = n

➤ Finde das kleinste Element

➤ In A[j+1...n]

➤ Tausche das aktuelle Element mit „dem kleinsten“

Beispiel

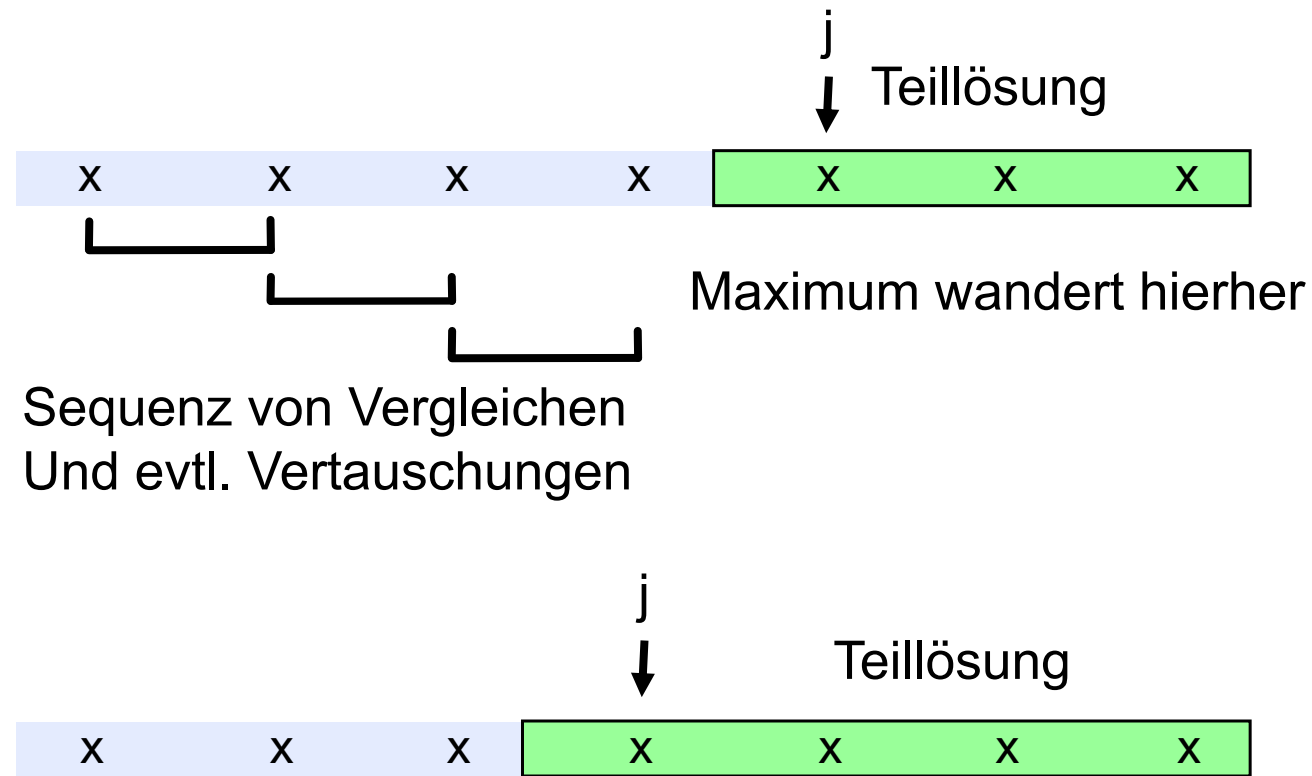


j=n

Bubble Sort

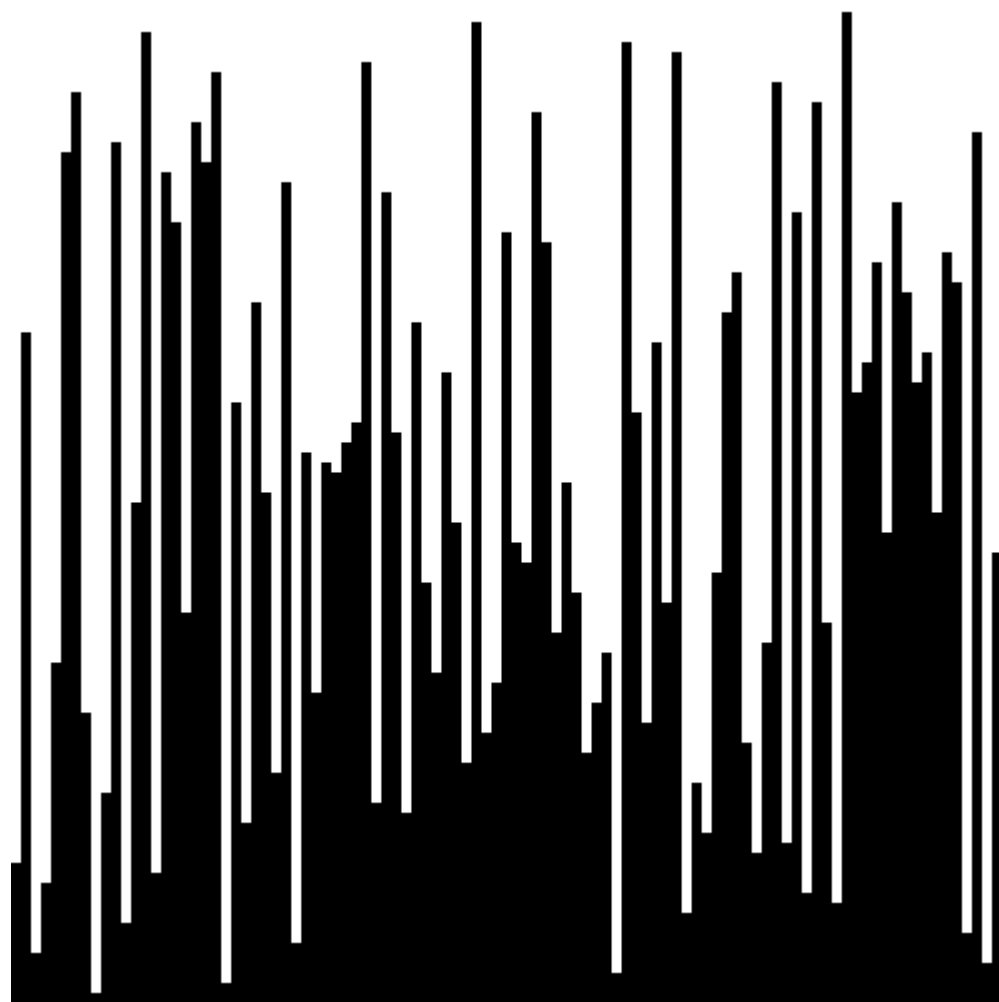
Bubble Sort

- Arbeitsweise:



Kann auch umgekehrt arbeiten, so dass die Minima nach links wandern.

Bubble Sort



Bubble Sort

BubbleSort(Array A)

1. **for** $j \leftarrow \text{length}(A) - 1$ **downto** 1 **do**
2. **for** $i \leftarrow 1$ **to** j **do**
3. **if** $A[i] > A[i+1]$ **then** swap($A, i, i+1$)

Idee Bubble Sort:

- *Die letzten Elemente von j bis n sind sortiert (zu Beginn $j = n-1$)*
- *Die größten Elemente steigen auf (bubblen), wie Luftblasen, die zu ihrer richtigen Position aufsteigen*
- *Am Ende ist die gesamte Folge sortiert*

- Alle drei Verfahren finden die Lösung durch schrittweises Sortieren mittels Vergleichen.
- Dabei verkleinern sie in jedem Schritt das Restproblem um eins.
- D.h., der Teil des Arrays der unsortiert ist verkleinert sich mit jedem Durchlauf der äußeren Schleife um 1.

- Einfache vergleichende Sortierverfahren
 - Sortieren durch Einfügen (insertion sort)
 - Sortieren durch Auswählen (selection sort)
 - Sortieren durch Vertauschen (bubble sort)
- Fortgeschrittene vergleichende Sortierverfahren
 - Sortieren durch Gruppieren (quick sort)
 - Sortieren durch Mischen (merge sort)
- **Nicht vergleichende Sortierverfahren**
 - **Sortieren durch Zählen (count sort)**
 - Sortieren durch Fachverteilen (radix sort)

Schnelle, digitale Sortiervverfahren

- Unter gewissen Einschränkungen des Wertebereichs können die Werte dazu verwendet werden, den endgültigen Platz direkt anzusteuern.
 - Sortieren durch Zählen (count sort)
 - Sortieren durch Fachverteilen (radix sort)
- Diese Verfahren sind jedoch nicht immer sinnvoll einsetzbar, z.B. wenn
 - Das Sortieren stabil sein soll, d.h. positionstreu
 - Der Wertebereich zu groß ist

Exkurs: Stabilität von Sortierverfahren

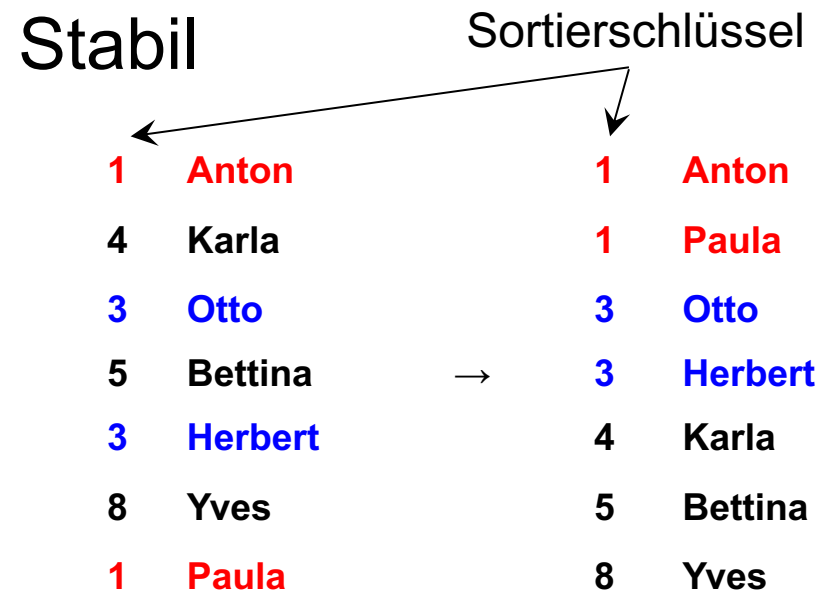
Definition „Stabiles Sortierverfahren“

Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, bewahrt.

Exkurs: Stabilität von Sortierverfahren

Definition „Stabiles Sortierverfahren“

Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, bewahrt.



Exkurs: Stabilität von Sortierv Verfahren

Definition „Stabiles Sortierv Verfahren“

Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, bewahrt.

Stabil

Sortierschlüssel	
1	Anton
4	Karla
3	Otto
5	Bettina
3	Herbert
8	Yves
1	Paula

→

1	Anton
1	Paula
3	Otto
3	Herbert
4	Karla
5	Bettina
8	Yves

Instabil

Sortierschlüssel	
1	Anton
4	Karla
3	Otto
5	Bettina
3	Herbert
8	Yves
1	Paula

→

1	Paula
1	Anton
3	Otto
3	Herbert
4	Karla
5	Bettina
8	Yves

Exkurs: Stabilität von Sortierv Verfahren

Definition „Stabiles Sortierv Verfahren“

Sortieralgorithmus, der die Reihenfolge der Datensätze, deren Sortierschlüssel gleich sind, bewahrt.

Stabil

Sortierschlüssel	
1 Anton	1 Anton
4 Karla	1 Paula
3 Otto	3 Otto
5 Bettina	3 Herbert
3 Herbert	4 Karla
8 Yves	5 Bettina
1 Paula	8 Yves

Instabil

Sortierschlüssel	
1 Anton	1 Paula
4 Karla	1 Anton
3 Otto	3 Otto
5 Bettina	3 Herbert
3 Herbert	4 Karla
8 Yves	5 Bettina
1 Paula	8 Yves

Sortieren durch Zählen (Count Sort)

- Annahme:
 - Die Werte stammen aus einem kleinen Wertebereich, d.h. sie liegen so dicht, dass sie zum Indizieren eines Arrays verwendet werden können.
 - Es ist wahrscheinlich, dass Werte mehrfach auftreten.
- Idee:
 - Die Häufigkeit jedes Elements wird ermittelt und daraus wird die endgültige Lage im Zielbehälter berechnet (streuendes Umspeichern).
 - Zum Schluss kann die Folge in den ursprünglichen Behälter zurückkopiert werden.

Count Sort – Beispiel

Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7

Count Sort – Beispiel

Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7
Ausgabe							
Index	1	2	3	4	5	6	7

Count Sort – Beispiel

Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7
Ausgabe							
Index	1	2	3	4	5	6	7

Zwischenspeicher

Wert	Anzahl

Count Sort – Beispiel

Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7
Ausgabe							
Index	1	2	3	4	5	6	7

Zwischenspeicher

Wert	Anzahl
1	
2	
3	
4	
5	

Count Sort – Beispiel

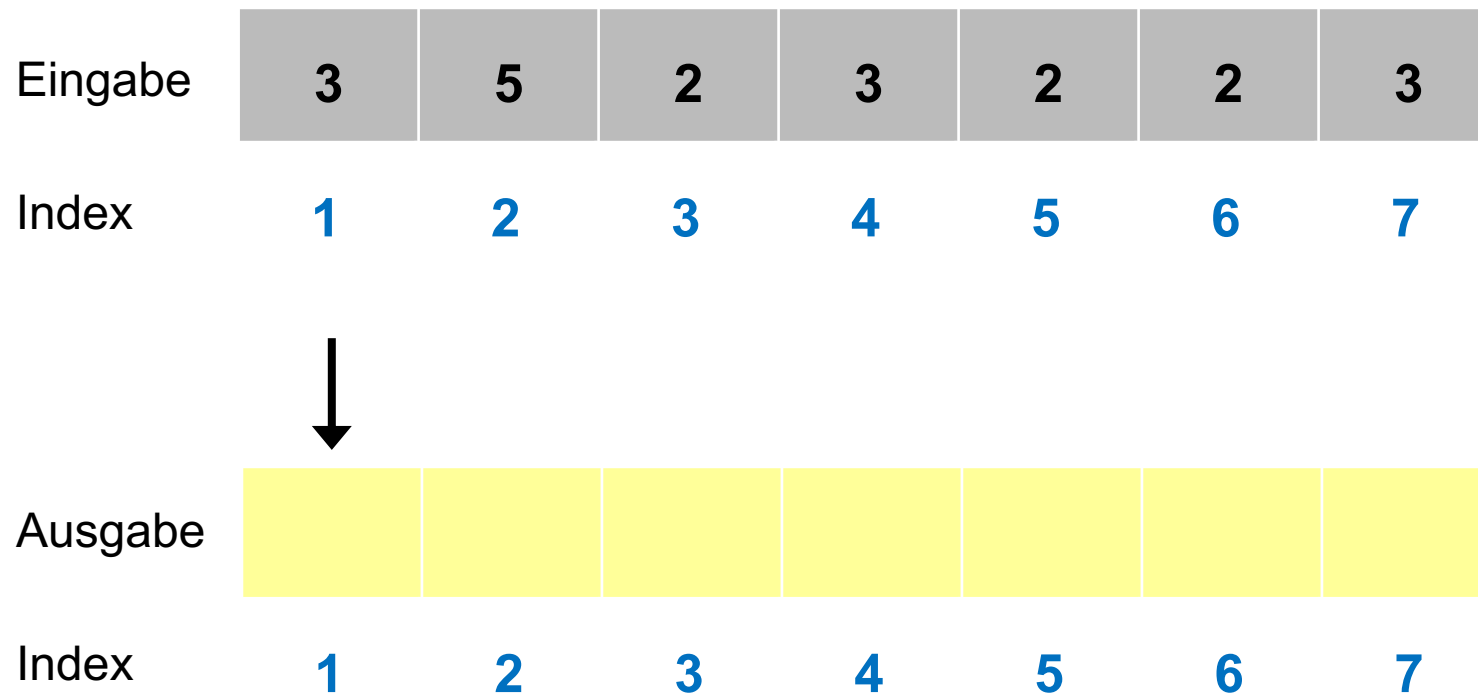
Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7
Ausgabe							
Index	1	2	3	4	5	6	7

Zwischenspeicher

Wert	Anzahl
1	0
2	3
3	3
4	0
5	1

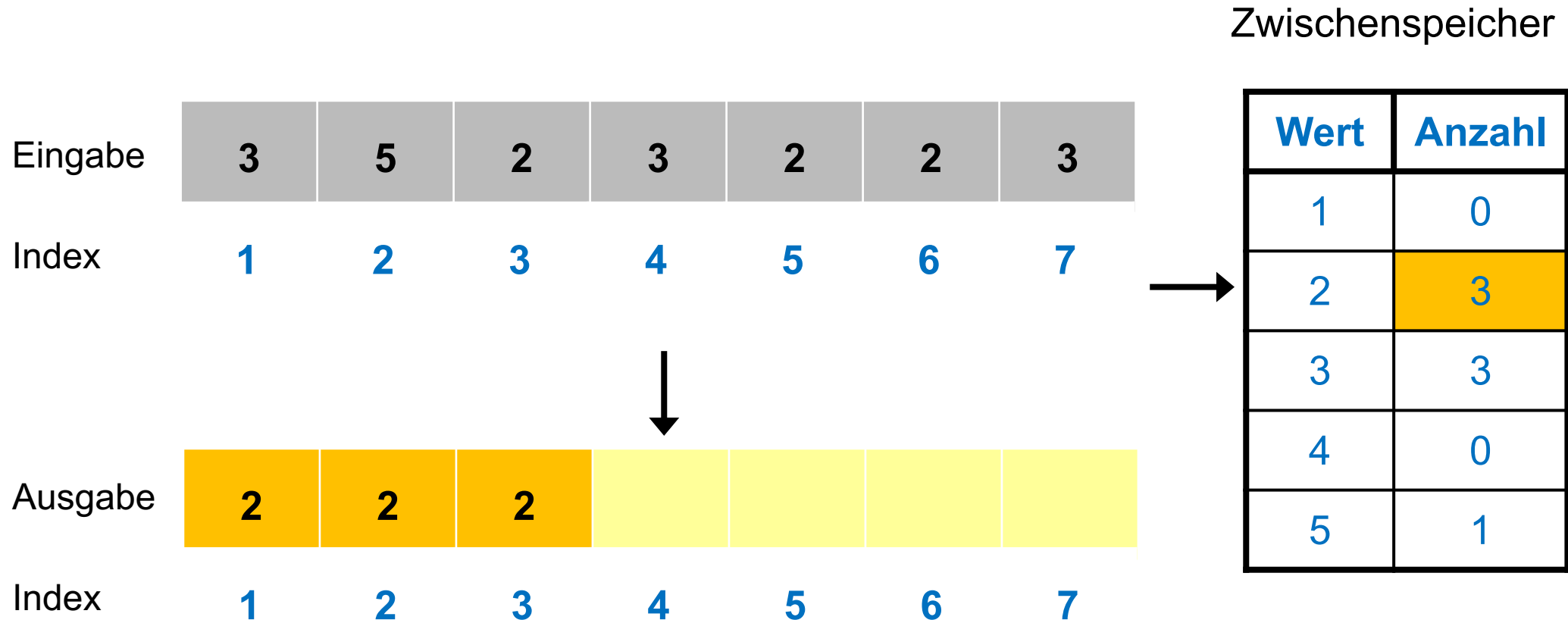
Count Sort – Beispiel

Zwischenspeicher



Wert	Anzahl
1	0
2	3
3	3
4	0
5	1

Count Sort – Beispiel



Count Sort – Beispiel

Zwischenspeicher

Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7
Ausgabe	2	2	2	3	3	3	
Index	1	2	3	4	5	6	7



Wert	Anzahl
1	0
2	3
3	3
4	0
5	1

Count Sort – Beispiel

Zwischenspeicher

Eingabe	3	5	2	3	2	2	3
Index	1	2	3	4	5	6	7
Ausgabe	2	2	2	3	3	3	
Index	1	2	3	4	5	6	7

Wert	Anzahl
1	0
2	3
3	3
4	0
5	1

Count Sort – Beispiel

Zwischenspeicher

Eingabe	3	5	2	3	2	2	3
---------	---	---	---	---	---	---	---

Index	1	2	3	4	5	6	7
-------	---	---	---	---	---	---	---

Ausgabe	2	2	2	3	3	3	5
---------	---	---	---	---	---	---	---

Index	1	2	3	4	5	6	7
-------	---	---	---	---	---	---	---

Wert	Anzahl
1	0
2	3
3	3
4	0
5	1

Count Sort – Beispiel

Ausgabe	2	2	2	3	3	3	5
Index	1	2	3	4	5	6	7

Count Sort

CountSort(Array A_in, Array A_out)

1. C ist Hilfsarray mit 0 initialisiert

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

3. C[A_in[j]] \leftarrow C[A_in[j]] + 1

4. k \leftarrow 1

5. **for** j \leftarrow 1 **to** length(C) **do**

6. **for** i \leftarrow 1 **to** C[j] **do**

7. A_out[k] \leftarrow j

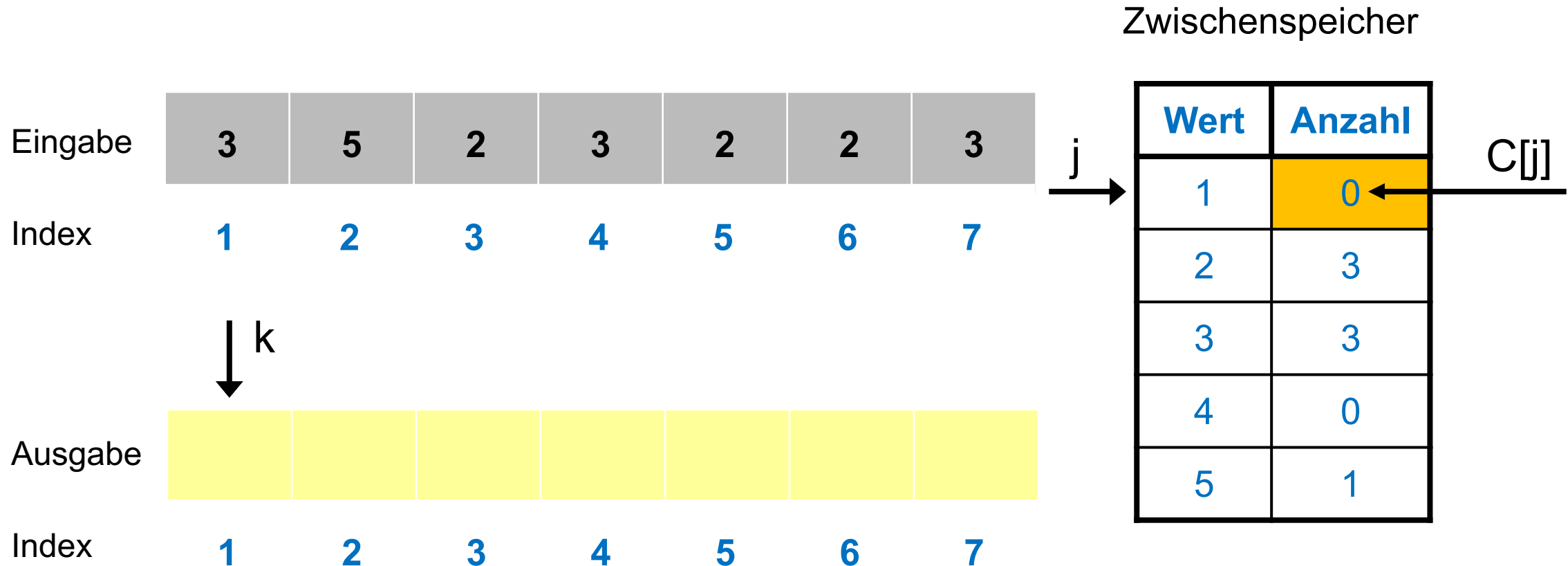
8. k \leftarrow k + 1

- Annahmen:
- Eingabegröße n
- length(A_in) = length(A_out) = n
- **Wertebereich von A_in: 1 – m**
- **length(C) = m**

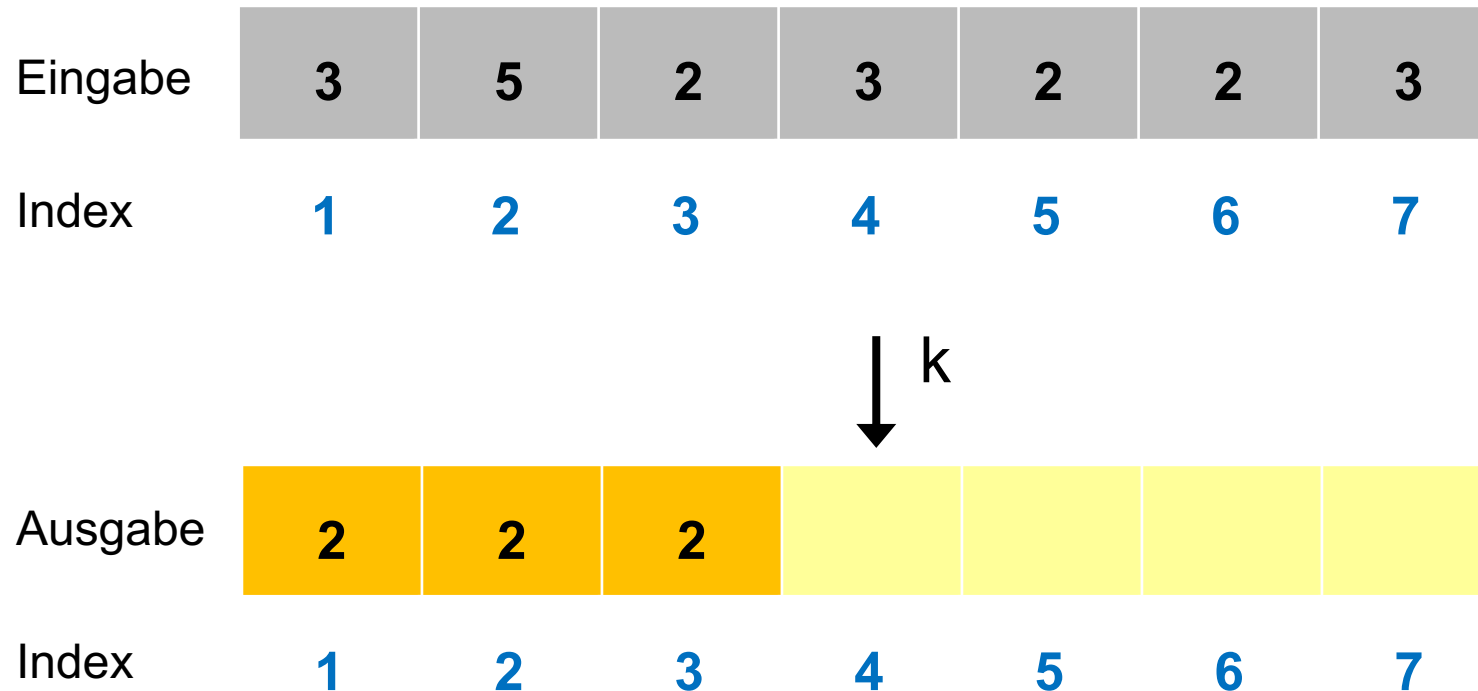
- Zähle, wie häufig jedes Element vorkommt

- Füge jedes Element der Reihe nach entsprechend seiner Häufigkeit in das Array hinein.

Count Sort – Beispiel



Count Sort – Beispiel



Zwischenspeicher

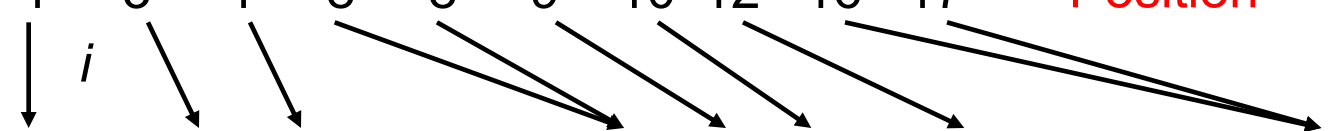
Wert	Anzahl
1	0
2	3
3	3
4	0
5	1

Diagram illustrating the intermediate storage (Zwischenspeicher) used in Count Sort. The table shows the frequency of each value. The value 2 has a frequency of 3, and the value 3 has a frequency of 3. The value 5 has a frequency of 1. The value 4 has a frequency of 0. The value 1 has a frequency of 0. The table is labeled C[j] and has an arrow pointing to the cell containing the value 3 in the 'Anzahl' column.

Count Sort – 2. Beispiel

j:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A_in:	3	1	3	3	5	7	2	8	9	7	6	3	8	1	8	8

j:	1	2	3	4	5	6	7	8	9	10	Werte Zähler Position					
C:	2	1	4	0	1	1	2	4	1	0						
	1	3	4	8	8	9	10	12	16	17						
A_out:	1	1	2	3	3	3	3	5	6	7	7	8	8	8	8	9
k:	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16



Count Sort

CountSort(Array A_in, A_out)

1. C ist Hilfsarray mit 0 initialisiert
2. **for** j \leftarrow 1 **to** length(A_in) **do**
3. C[A_in[j]] \leftarrow C[A_in[j]] + 1
4. k \leftarrow 1
5. **for** j \leftarrow 1 **to** length(C) **do**
6. **for** i \leftarrow 1 **to** C[j] **do**
7. A_out[k] \leftarrow j
8. k \leftarrow k + 1

Count Sort

CountSort(Array A_in, A_out)

1. C ist Hilfsarray mit 0 initialisiert

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

3. C[A_in[j]] \leftarrow C[A_in[j]] + 1

4. k \leftarrow 1

5. **for** j \leftarrow 1 **to** length(C) **do**

6. **for** i \leftarrow 1 **to** C[j] **do**

7. A_out[k] \leftarrow j

8. k \leftarrow k + 1

Initialisierung des
Hilfsarrays



Count Sort

CountSort(Array A_in, A_out)

1. C ist Hilfsarray mit 0 initialisiert

Initialisierung des
Hilfsarrays

2. **for** j \leftarrow 1 **to** length(A_in) **do**
3. C[A_in[j]] \leftarrow C[A_in[j]] + 1

Berechnung der
Häufigkeiten

4. k \leftarrow 1

5. **for** j \leftarrow 1 **to** length(C) **do**

6. **for** i \leftarrow 1 **to** C[j] **do**

7. A_out[k] \leftarrow j

8. k \leftarrow k + 1

Count Sort

CountSort(Array A_in, A_out)

1. C ist Hilfsarray mit 0 initialisiert

Initialisierung des
Hilfsarrays

2. **for** j \leftarrow 1 **to** length(A_in) **do**
3. C[A_in[j]] \leftarrow C[A_in[j]] + 1

Berechnung der
Häufigkeiten

4. k \leftarrow 1
5. **for** j \leftarrow 1 **to** length(C) **do**
6. **for** i \leftarrow 1 **to** C[j] **do**
7. A_out[k] \leftarrow j
8. k \leftarrow k + 1

Schreiben des
sortierten Arrays

Ausblick

- VL 0 „Organisation und Inhalt“: Ablauf der Vorlesung, Termine
- VL 1 „Algorithmen, Pseudocode, Sortieren I“: Insertion Sort
- VL 2 „Algorithmen, Pseudocode, Sortieren II“: Selection Sort, Bubble Sort, Count Sort**
- VL 3 „Laufzeit und Speicherplatz“: Laufzeitanalyse der vorgestellten Sortiervverfahren
- VL 4 „Einfache Datenstrukturen“: Arrays, verkettete Listen, Structs in C, Stack, Queue
- VL 5 „Bäume“: Binärbäume, Baumtraversierung, Laufzeitanalyse Baumoperationen
- VL 6 „Dateien in C“: Dateien, Dateisysteme, Verzeichnisse, Dateiverwaltung mit C
- VL 7 „Teile und Herrsche I“: Einführung der algorithmischen Methode, Merge Sort
- VL 8 „Korrektheitsbeweise“: Rechnermodel, Beispielbeweise
- VL 9 „Prioritätenslangen/Halden/Heaps“: Heap Sort, Binärer Heap, Heap Operationen
- VL 10 „Fortgeschrittene Sortiervverfahren“: Quick Sort, Radix Sort
- VL 11 „AVL Bäume“: Definition, Baumoperationen, Traversierung
- VL 12 „Teile und Herrsche II“: Generalisierung des algorithmischen Prinzips, Mastertheorem
- VL 13 „Q & A“: Offene Vorlesung/Wiederholung