

# Aufgabe (Schreibtischtest, Algorithmenanalyse)

Führen Sie einen Schreibtischtest für den Algorithmus **Positionsort** für das folgende Eingabe-Array durch. Geben Sie nach jedem Durchlauf der **for**-Schleife mit Laufvariablen  $i$  das Array  $B$  an. Gehen sie davon aus das  $B$  mit den Werten 0 initialisiert ist.

- $S = [5, 2, 7, 5, 5, 7]$   
Ordnung: natürliche Ordnung auf den natürlichen Zahlen

Erläutern Sie den Algorithmus.

---

## Positionsort( $S$ )

---

```
1:  $n := |S|;$ 
2:  $B :=$  neues Array der Länge  $n$ 
3: for  $i = 1 .. n$  do
4:    $pos := 1;$ 
5:   for  $j = 1 .. i - 1$  do
6:     if  $S[j] \leq S[i]$  then
7:        $pos := pos + 1;$ 
8:     end if
9:   end for
10:  for  $j = i + 1 .. n$  do
11:    if  $S[j] < S[i]$  then
12:       $pos := pos + 1;$ 
13:    end if
14:  end for
15:   $B[pos] := S[i];$ 
16: end for
17: return  $B;$ 
```

---

# Aufgabe (Schreibtischttest, lexikographische Ordnung)

Führen Sie einen Schreibtischttest für den Algorithmus **Selectionsort** aus der VL für die folgenden Eingabe-Arrays durch. Geben Sie das Array  $S$  nach jedem Durchlauf der Zeilen 5-7 an.

- $S = [bca, abe, bdd, aae, ecd, bce]$   
Ordnung: lexikographische Ordnung  
auf  $\Sigma = \{a, b, c, d, e\}$  mit  
 $a < b < c < d < e$

---

## Selectionsort( $S$ )

---

```
1:  $n := |S|;$ 
2: for  $i = 1 .. n - 1$  do
3:   for  $j = i + 1 .. n$  do
4:     if  $S[i] > S[j]$  then
5:        $tmp := S[i];$ 
6:        $S[j] := S[i];$ 
7:        $S[i] := tmp;$ 
8:     end if
9:   end for
10: end for
```

---

**Lexikographische Ordnung** für Zeichenketten aus  $\Sigma^m$  (wobei  $\Sigma$  geordnetes Alphabet):  
Es gilt  $a_1 \dots a_m < b_1 \dots b_m$  g.d.w. ein  $i$  mit  $1 \leq i \leq m$  existiert, so dass  $a_i < b_i$  und  $a_j = b_j$  für alle  $j < i$ .

# Aufgabe (Stabilität)

In dieser Aufgabe sortieren wir Arrays mit Einträgen des abstrakten Datentyps **Element**. Ein Element  $e$  hat einen **Schlüssel**  $e.\text{key}$  aus einer Menge  $\mathbb{K}$  und einen **Wert**  $e.\text{val}$  aus einer Menge  $\mathbb{V}$ .

Elemente werden anhand ihrer Schlüssel sortiert. Dazu ist auf der Menge  $\mathbb{K}$  der Schlüssel eine lineare Ordnung  $\leq$  definiert. Für beliebige Elemente  $e_1$  und  $e_2$  schreiben wir

$e_1 \leq e_2$  genau dann, wenn  $e_1.\text{key} \leq e_2.\text{key}$ .

Ein Sortierverfahren heißt **stabil**, wenn Elemente mit gleichen Schlüsseln nach der Sortierung in der gleichen Reihenfolge aufeinander folgen wie vor der Sortierung.

# Aufgabe (Stabilität)

In dieser Aufgabe sortieren wir Arrays mit Einträgen des abstrakten Datentyps **Element**. Ein Element  $e$  hat einen **Schlüssel**  $e.\text{key}$  aus einer Menge  $\mathbb{K}$  und einen **Wert**  $e.\text{val}$  aus einer Menge  $\mathbb{V}$ .

Elemente werden anhand ihrer Schlüssel sortiert. Dazu ist auf der Menge  $\mathbb{K}$  der Schlüssel eine lineare Ordnung  $\leq$  definiert. Für beliebige Elemente  $e_1$  und  $e_2$  schreiben wir

$e_1 \leq e_2$  genau dann, wenn  $e_1.\text{key} \leq e_2.\text{key}$ .

Ein Sortierverfahren heißt **stabil**, wenn Elemente mit gleichen Schlüsseln nach der Sortierung in der gleichen Reihenfolge aufeinander folgen wie vor der Sortierung.

Entscheiden Sie, ob **Selectionsort** stabil ist. Falls **Selectionsort** nicht stabil ist, geben Sie eine (möglichst kleine) Instanz als Gegenbeispiel an; andernfalls begründen Sie, weshalb **Selectionsort** stabil ist.

---

## Selectionsort( $S$ )

---

```
1:  $n := |S|;$ 
2: for  $i = 1 .. n - 1$  do
3:   for  $j = i + 1 .. n$  do
4:     if  $S[i] > S[j]$  then
5:        $\text{tmp} := S[i];$ 
6:        $S[j] := S[i];$ 
7:        $S[i] := \text{tmp};$ 
8:     end if
9:   end for
10: end for
```

---

# Aufgabe (Stabilität)

In dieser Aufgabe sortieren wir Arrays mit Einträgen des abstrakten Datentyps **Element**. Ein Element  $e$  hat einen **Schlüssel**  $e.\text{key}$  aus einer Menge  $\mathbb{K}$  und einen **Wert**  $e.\text{val}$  aus einer Menge  $\mathbb{V}$ .

Elemente werden anhand ihrer Schlüssel sortiert. Dazu ist auf der Menge  $\mathbb{K}$  der Schlüssel eine lineare Ordnung  $\leq$  definiert. Für beliebige Elemente  $e_1$  und  $e_2$  schreiben wir

$e_1 \leq e_2$  genau dann, wenn  $e_1.\text{key} \leq e_2.\text{key}$ .

Ein Sortierverfahren heißt **stabil**, wenn Elemente mit gleichen Schlüsseln nach der Sortierung in der gleichen Reihenfolge aufeinander folgen wie vor der Sortierung.

Entscheiden Sie, ob **Positionsort** stabil ist. Falls **Positionsort** nicht stabil ist, geben Sie eine (möglichst kleine) Instanz als Gegenbeispiel an; andernfalls begründen Sie, weshalb **Positionsort** stabil ist.

---

## Positionsort( $S$ )

---

```
1:  $n := |S|$ ;  
2:  $B :=$  neues Array der Länge  $n$   
3: for  $i = 1 .. n$  do  
4:    $\text{pos} := 1$ ;  
5:   for  $j = 1 .. i - 1$  do  
6:     if  $S[j] \leq S[i]$  then  
7:        $\text{pos} := \text{pos} + 1$ ;  
8:     end if  
9:   end for  
10:  for  $j = i + 1 .. n$  do  
11:    if  $S[j] < S[i]$  then  
12:       $\text{pos} := \text{pos} + 1$ ;  
13:    end if  
14:  end for  
15:   $B[\text{pos}] := S[i]$ ;  
16: end for  
17: return  $B$ ;
```

---

# Allgemeine Sortierverfahren

Ein **allgemeines Sortierverfahren** ist ein Sortierverfahren, welches nur Wissen über die Anordnung der zu sortierenden Elemente erlangen kann, indem es Elemente vergleicht. Ansonsten können keine weiteren Eigenschaften der Elemente vorausgesetzt bzw. ausgenutzt werden.

Welche der aus der Vorlesung bekannten Sortierverfahren sind allgemeine Sortierverfahren?

# Aufgabe (Sortierung spezieller Arrays)

Beweisen oder widerlegen Sie: Es gibt ein allgemeines Sortierverfahren, welches ein Array von  $n$  beliebigen (in konstanter Zeit vergleichbaren) Elementen im Worst Case in Laufzeit  $\mathcal{O}(n)$  sortiert, falls...

- a) ... in dem Array das Anfangsstück der Länge  $\lfloor \frac{3}{4}n \rfloor$  bereits vorsortiert ist.

*Hinweise:*

- *Für den Fall, dass es ein solches allgemeines Sortierverfahren gibt, können Sie als Beweis die Idee eines konkreten Verfahrens beschreiben.*
- *Falls kein solches allgemeines Sortierverfahren existiert, können Sie die in der Vorlesung gezeigte untere Schranke für allgemeine Sortierverfahren nutzen.*

# Aufgabe (Sortierung spezieller Arrays)

Beweisen oder widerlegen Sie: Es gibt ein allgemeines Sortierverfahren, welches ein Array von  $n$  beliebigen (in konstanter Zeit vergleichbaren) Elementen im Worst Case in Laufzeit  $\mathcal{O}(n)$  sortiert, falls...

- a) ... in dem Array das Anfangsstück der Länge  $\lfloor \frac{3}{4}n \rfloor$  bereits vorsortiert ist.
- b) ... in dem Array nur Zahlen aus  $\{1, \dots, n\}$  enthalten sind.

*Hinweise:*

- *Für den Fall, dass es ein solches allgemeines Sortierverfahren gibt, können Sie als Beweis die Idee eines konkreten Verfahrens beschreiben.*
- *Falls kein solches allgemeines Sortierverfahren existiert, können Sie die in der Vorlesung gezeigte untere Schranke für allgemeine Sortierverfahren nutzen.*



# Entscheidungsbaum für **Selectionsort**

Wie sieht der Entscheidungsbaum für Selectionsort mit 3 Elementen aus?

---

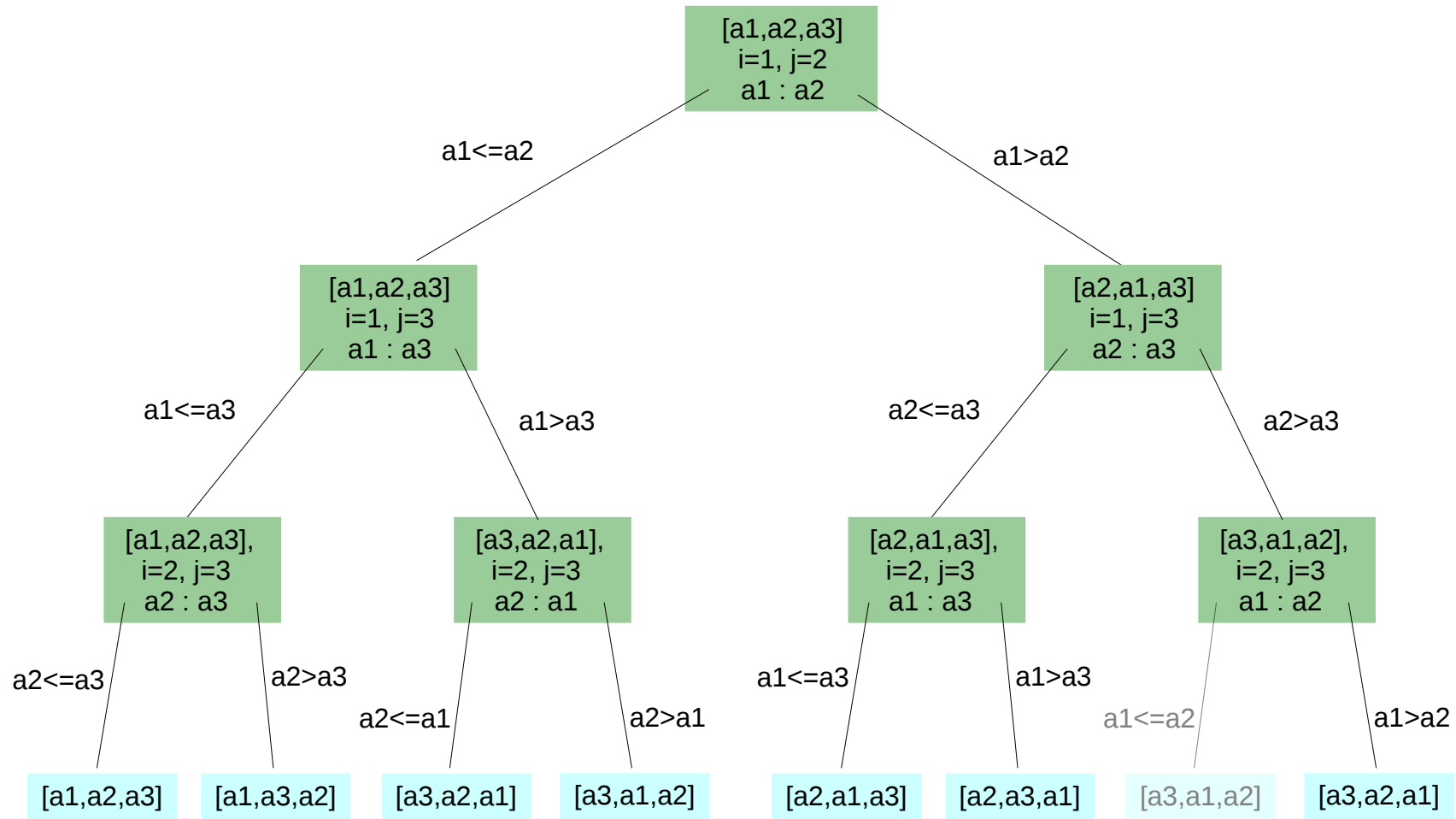
## **Selectionsort( $S$ )**

---

```
1:  $n := |S|;$   
2: for  $i = 1 .. n - 1$  do  
3:   for  $j = i + 1 .. n$  do  
4:     if  $S[i] > S[j]$  then  
5:        $tmp := S[i];$   
6:        $S[j] := S[i];$   
7:        $S[i] := tmp;$   
8:     end if  
9:   end for  
10: end for
```

---

# Entscheidungsbaum für Selectionsort



# Entscheidungsbaum für Insertionsort

Wie sieht der Entscheidungsbaum für Insertionsort mit 3 Elementen aus?

---

## Insertionsort( $S$ )

---

```
1:  $n := |S|;$   
2: for  $i = 2 .. n$  do  
3:    $j := i;$   
4:    $k := S[j];$   
5:   while  $(S[j - 1] > k)$  and  $(j > 1)$  do  
6:      $S[j] := S[j - 1];$   
7:      $j := j - 1;$   
8:   end while  
9:    $S[j] := k;$   
10: end for
```

---

# Entscheidungsbaum für Insertionsort

