

Aufgabe 1

Blau: Element an der korrekten Position

Rot: das minimale Element der unsortierten Teilliste

9	33	6	8	7	Vergleiche	Vertauschungen
3	9	66	8	7	4	1
3	6	9	8	77	3	1
3	6	7	88	9	2	1
3	6	7	8	9	1	1
					10	4

- Auch wenn ein Element (wie die 8) bereits an der richtigen Position steht, führt der Algorithmus eine Vertauschung (mit sich selbst) durch. Um das zu verhindern, müsste man vor bei jeder Vertauschung testen, ob eine Vertauschung nötig ist, was am Ende sogar mehr Zeit kosten würde.
- Das letzte Element muss nicht mehr einsortiert werden, da es das grösste Element sein muss. Daher genügt es, die äussere for-Schleife nur bis zum zweitletzten Element laufen zu lassen.

Aufgabe 2

Insertionsort

Blau: relativ sortierte Teilliste

Rot: das nächste einzusortierende Element

9	3	6	8	7	Vergleiche	Verschiebungen
3	9	6	8	7	1	1
3	6	9	8	7	2	1
3	6	8	9	7	2	1
3	6	7	8	9	3	2
					8	13

Aufgabe 3

1 4 3 2
1 4 3 2
1 4 3 2
1 3 4 2
1 3 4 2
1 3 4 2
1 3 2 4
1 2 3 4
1 2 3 4
1 2 3 4
1 2 3 4 _

Aufgabe 4

siehe Theorie

Aufgabe 5

Blau: Element an der korrekten Position

Rot: Vertauschungskandidaten

8	7	5	4	1	Vergleiche	Vertauschungen
7	8	5	4	1	1	1
7	5	8	4	1	1	1
7	5	4	8	1	1	1
7	5	4	1	8	1	1
5	7	4	1	8	1	1
5	4	7	1	8	1	1
5	4	1	7	8	1	1
4	5	1	7	8	1	1
4	1	5	7	8	1	1
1	4	5	7	8	1	1
					10	10

Aufgabe 6

Selectionsort:

(a) die Elemente sind aufsteigend sortiert:

- $6 + 5 + 4 + 3 + 2 + 1 = 21$ Vergleiche
- $1 + 1 + 1 + 1 + 1 + 1 = 6$ Vertauschungen

(b) die Elemente sind absteigend sortiert:

- $6 + 5 + 4 + 3 + 2 + 1 = 21$ Vergleiche
- $1 + 1 + 1 + 1 + 1 + 1 = 6$ Vertauschungen

Aufgabe 7

Insertionsort

(a) die Elemente sind aufsteigend sortiert:

- $1 + 1 + 1 + 1 + 1 + 1 = 6$ Vergleiche
- $0 + 0 + 0 + 0 + 0 + 0 = 0$ Verschiebungen

(b) die Elemente sind absteigend sortiert:

- $1 + 2 + 3 + 4 + 5 + 6 = 21$ Vergleiche
- $1 + 2 + 3 + 4 + 5 + 6 = 21$ Verschiebungen

Aufgabe 8

```
1 def sort(A):
2     n = len(A)
3     for i in range(0, n-1):
4         k = i
5         for j in range(i+1, n):
6             if A[j] < A[k]:
7                 k = j
8         A[k], A[i] = A[i], A[k]
```

Es handelt sich um Selectionsort.

Aufgabe 9

(a)

4	5	1	3	2
4	5	1	3	2
1	4	5	3	2
1	3	4	5	2
1	2	3	4	5

Insertionsort

(b)

4	5	1	3	2
1	5	4	3	2
1	2	4	3	5
1	2	3	4	5
1	2	3	4	5

Selectionsort

Aufgabe 10

	Worst Case	Best Case
Insertionsort	$O(n^2)$	$O(n)$
Gnomesort	$O(n)$	$O(n)$
Bubblesort	$O(n^2)$	$O(n^2)$
Selectionsort	$O(n^2)$	$O(n^2)$

Aufgabe 11

$$C \cdot 1000^2 = 0.1 \text{ s}$$

$$C \cdot 7000^2 = C \cdot (7 \cdot 1000)^2 = 49 \cdot C \cdot 1000^2 = 49 \cdot 0.1 \text{ s} = 4.9 \text{ s}$$

Aufgabe 12

```
1 def isSorted(A):
2     for i in range(0, len(A)-1):
```

```

3         if A[i] > A[i+1]:
4             return False
5     return True

```

Aufgabe 13

```

1 def find_min_pos(A):
2     min_pos = 0
3     for i in range(1, len(A)):
4         if A[i] < A[min_pos]:
5             min_pos = i
6     return min_pos

```

Aufgabe 14

(a)

...	2	8	3	9	4	6	...
...	2	8	3	9	4	6	...
...	2	8	3	9	4	6	...
...	2	3	8	9	4	6	...
...	2	3	8	9	4	6	...
...	2	3	4	9	8	6	...
...	2	3	4	6	8	9	...

(b)

99	100	101	102	103	104	105	106
...
...	2	3	4	6	8	9	...

Der Partitionierungsschritt gibt den Index 103 zurück.

Damit wird den folgenden rekursiven Aufrufen mitgeteilt, das noch die Teillisten mit den Indizes 100–102 und 104–105 sortiert werden müssen.

Aufgabe 15

Der Quicksort-Algorithmus hat die Worst Case-Laufzeit, wenn die zu sortierende Liste bereits auf- oder absteigend sortiert ist.

Aufgabe 16

- *Randomized Quicksort*: Wähle ein zufälliges Element.
- *Median of three*: Wähle den Median des ersten, mittleren und letzten Elements.

Aufgabe 17

8	4	7	1	3	9	5
4	8	7	1	3	9	5
4	1	7	8	3	9	5
4	1	3	8	7	9	5
4	1	3	5	7	9	8
4	1	3				
1	4	3				
1	3	4				
1		4				
				7	9	8
				7	9	8
				7	8	9
				7		9

Aufgabe 18

(a) [8, 12, 27, 3, 16, 29, 17]

sortieren: [3, 8, 12, 16, 17, 27, 29]

Median: 16

(b) [23, 14, 7, 9, 55, 6, 2, 12]

sortieren: [2, 6, 7, 9, 12, 14, 23, 55]

Median: 10.5