Sarah Ertel
Patrick Greher
Eugen Ljavin

1	2	3	4	Σ

Übungsblatt Nr. 3

(Abgabetermin 10.05.2018)

Aufgabe 1

a)

Algorithm 1: Insertion Sort Algorithmus

```
1  function insertionSort(toSort[ ])
2  for i ← 1; i < toSort.length; i ← i + 1 do
3  |   j ← i;
4  |   while (j > 0) ∧ (toSort[j − 1] > toSort[j]) do
5  |   |   tmp ← toSort[j - 1];
6  |   |   toSort[j-1] ← toSort[j];
7  |   |   toSort[j] ← tmp;
8  |   |   j ← j-1;
9  |   end
10 end
```

Algorithm 2: Minimumsuche + Austausch Algorithmus

```
1  function minimumSwapSort(toSort[ ])
2  for i ← 0; i < toSort.length − 1; i ← i + 1 do
3  |   for j ← i + 1; j < toSort.length; i ← j + 1 do
4  |   |   if toSort[i] > toSort[j] then
5  |   |   |   tmp ← toSort[i];
6  |   |   |   toSort[i] ← toSort[j];
7  |   |   |   toSort[j] ← tmp;
8  |   |   end
9  |   end
10 end
```

b)

c)

	Minimumsuche + Austausch Algorithmus	Insertion Sort
Vertauschungen	0	0
Vergleiche	maximal: $\frac{n^2}{2} - \frac{n}{2}$	$n - 1$

d)

$n \in \mathbb{N}$
$A = \langle n, n+1, n+2, ... \rangle$

Es gibt dann $\frac{n^2}{2} - \frac{n}{2} - (n-1)$ Vergleiche (für beide Algorithmen)

e)

$n \in \mathbb{N}$
$A = \langle n, n-1, n-2, ... \rangle$
Es gibt dann $\frac{n^2}{2} - \frac{n}{2} - (n-1)$ Vertauschungen (für beide Algorithmen)

Aufgabe 2

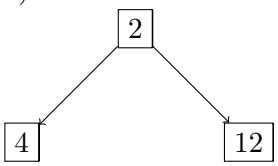
$A = \langle 4, 2, 12, 10, 18, 14, 6, 16, 8 \rangle$

a)

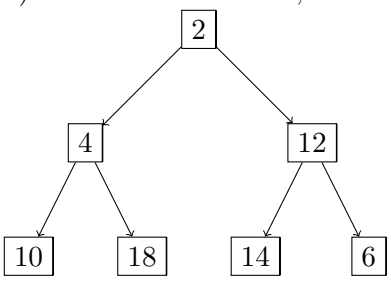
1) Die erste Zahl aus dem Array nehmen und als Wurzel einsetzen

2) Die nächste Zahl aus dem Array als Child a

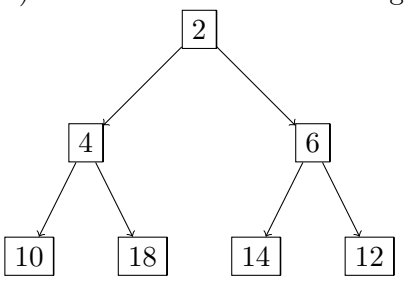
3) 4 und 2 vertauschen, da $2 < 4$



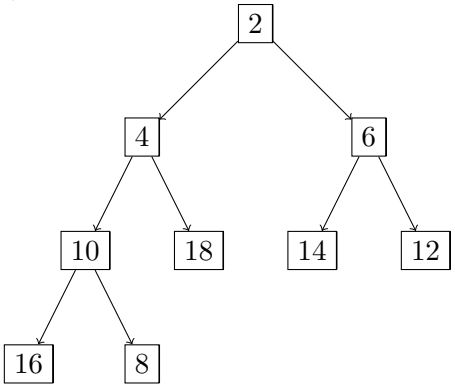
4) nächste Zahl als Child anfügen



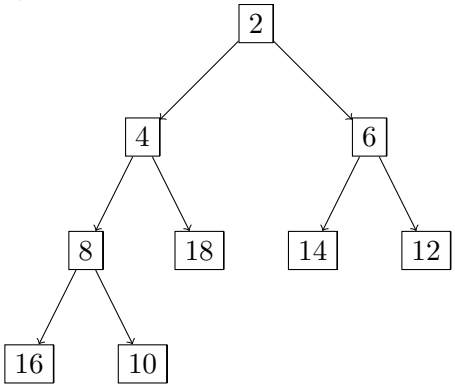
5) Die nächsten vier Elemente werden angefügt (10,18,14 sind größer als die jeweiligen Parent Elemente)



6) Da $6 < 12$ müssen die beiden Elemente vert

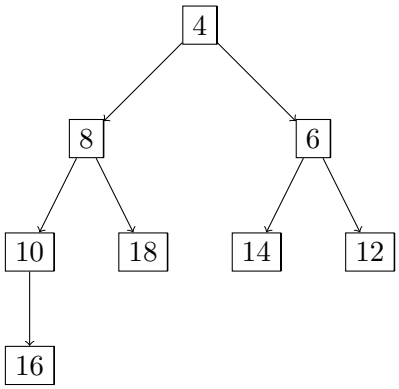


6) Die letzten beiden Elemente werden angefügt

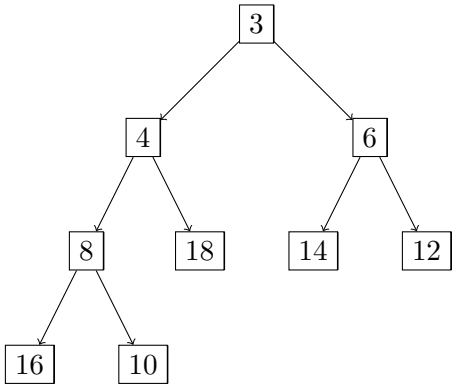


7) 8 und 10 müssen vertrauscht werden

b)



c)



d)

Aufgabe 3

a)

Ein k-Heap kann wie ein Binärer-Heap als Array dargestellt werden. Die Kinderelemente eines Knotens lassen sich hierbei über den Index i^k finden. Ebenso lassen sich die

b)

Die Höhe eines k-Heaps der Größe n beträgt $log_k(n) + 1$

c)

Aufgabe 4

a)

b)

Der Algorithmus ruf sich rekursiv drei mal auf und betrachtet dabei $\frac{2}{3}$ der Arraylänge **n**. Des Weiteren werden zwei Vergleiche durchgeführt, um zu prüfen, ob zwei Elemente

getauscht werden müssen sowie für die Abbruchbedingung. Es ergibt sich daraus die Rekursionsvorschrift $T(n) = 3 \cdot T\left(\frac{2}{3} \cdot n\right) + 1$
Mit Hilfe des Mastertheorems lässt sich folgende Komplexität ermitteln:

$$T(n) = 3 \cdot T\left(\frac{2}{3} \cdot n\right) + 1 \Rightarrow 3 \cdot T\left(\frac{2}{3} \cdot n\right) + \mathcal{O}(1)$$

Nach dem Mastertheorem gilt: $a = 3;\ b = \frac{3}{2};\ f(n) = 1 = \mathcal{O}(n^c)$ mit $c = 0$

Es ist: $c < \log_{\frac{3}{2}} 3$

Nach Fall 1 des Mastertheorems gilt: $T(n) = \mathcal{O}(n^{\log_{\frac{3}{2}} 3}) \approx \mathcal{O}(n^{2,7})$

c)

Sowohl Quck-Sort, als auch Minimumsuche + Austausch sowie Insertion Sort haben im worst case eine Komplexität von $\mathcal{O}(n^2)$.
Setzt man in dieser Teilaufgabe Komplexität mit Effizienz gleich, gilt $\mathcal{O}(n^2) < \mathcal{O}(n^{2,7})$.
Damit ist Zwei-Drittel-Sortieren im worst case **nicht** effizienter als die drei obenstehenden Sortieralgorithmen.