

Algorithmen und Datenstrukturen

3. Übungsserie

Florian Ludewig (185722)

25. Mai 2020

Aufgabe 1

a)

STOOGE-SORT funktioniert für Felder mit 0, 1 oder 2 Elementen:

- **0 Elemente**
Das Feld ist bereits sortiert.
- **1 Element**
Das Feld ist bereits sortiert.
- **2 Elemente**
Sei $A=[x,y]$, dann ist $i=0$ und $j=1$. Getauscht wird, wenn $y < x$, wodurch das Feld in jedem Fall korrekt sortiert ist.
Außerdem ist $l = j - i + 1 = 1 - 0 + 1 = 2$ und $2 \not\geq 2$ wodurch der Algorithmus terminiert.

Angenommen STOOGE-SORT funktioniert für alle Felder der Länge $n - 1$ oder kleiner.

Betrachten eines Feldes A der Länge n . Der Algorithmus arbeitet drei Schritte ab:

1. STOOGE-SORT($i, j-k$)
Hier wird STOOGE-SORT rekursiv mit $i = 0$ und $j = (n - 1) - \lfloor \frac{n}{3} \rfloor$ aufgerufen. Weil wir davon ausgehen, dass der Algorithmus für Felder mit kleineren Längen funktioniert, sind nach diesem rekursiven Aufruf alle Zahlen im ersten Drittel des Feldes $[0, \dots, \lfloor \frac{n}{3} \rfloor]$ kleiner als im zweiten Drittel $[\lceil \frac{n}{3} \rceil, \dots, n - \lfloor \frac{n}{3} \rfloor]$.
2. STOOGE-SORT($i+k, j$)
Nun wird STOOGE-SORT rekursiv mit $i = \lfloor \frac{n}{3} \rfloor$ und $j = n - 1$ aufgerufen. Danach sind mit Sicherheit alle Zahlen im zweiten Drittel des Feldes $[\lfloor \frac{n}{3} \rfloor, \dots, n - \lfloor \frac{n}{3} \rfloor]$ kleiner als die Zahlen im letzten Drittel des Feldes $[n - \lceil \frac{n}{3} \rceil, \dots, n - 1]$.

Demnach befinden sich an diesem Punkt die größten Zahlen im letzten Drittel des Feldes.

3. **STOOGE-SORT**(*i*, *j-k*)

Zum Schluss wird **STOOGE-SORT** nochmal rekursiv mit $i = 0$ und $j = (n - 1) - \lfloor \frac{n}{3} \rfloor$ aufgerufen, wodurch das Feld in den ersten beiden Dritteln sortiert wird. Weil die größten Zahlen sich bereits im letzten Drittel befinden ist das Feld nach diesem Schritt vollständig sortiert.

b)

STOOGE-SORT ist ein rekursiver Algorithmus und seine Laufzeit kann durch folgendes rekursives Schema beschrieben werden:

$$T(0) = 1, T(1) = 1, T(2) = 1$$

$$T(n) = 3 \cdot T\left(\frac{2}{3}n\right) + O(1) = 3 \cdot T\left(\frac{n}{1.5}\right) + O(1)$$

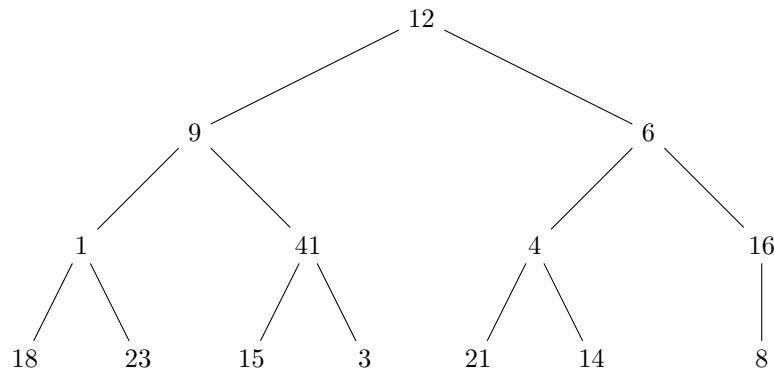
Weil der Algorithmus sich selbst **drei** mal aufruft und jedes mal **zwei Drittel** des Feldes sortiert. Mithilfe des ersten Falles des Mastertheorems lässt sich dann die Laufzeit bestimmen:

$$T(n) \in \Theta(n^{\log_b a}) = \Theta(n^{\log_{\frac{3}{2}} 3}) \approx \Theta(n^{2.7})$$

Aufgabe 2

a)

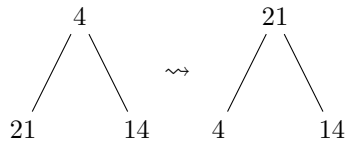
$A = (12, 9, 6, 1, 41, 4, 16, 18, 23, 15, 3, 21, 14, 8)$ als Heap:



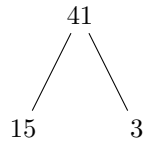
- **HEAPIFY(A, 7)**



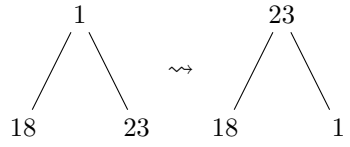
- **HEAPIFY(A, 6)**



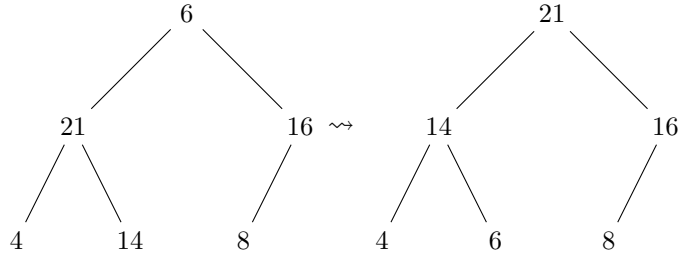
- **HEAPIFY(A, 5)**



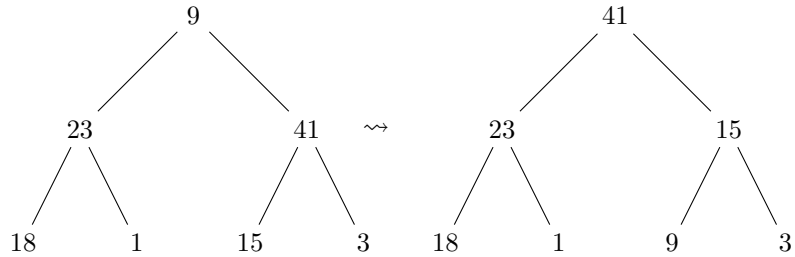
- **HEAPIFY(A, 4)**



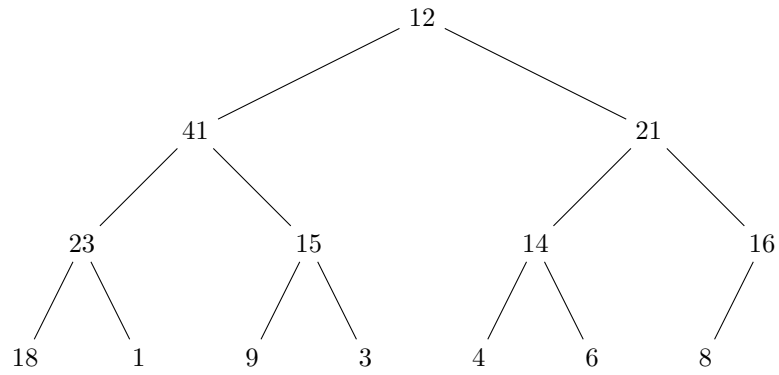
- **HEAPIFY(A, 3)**



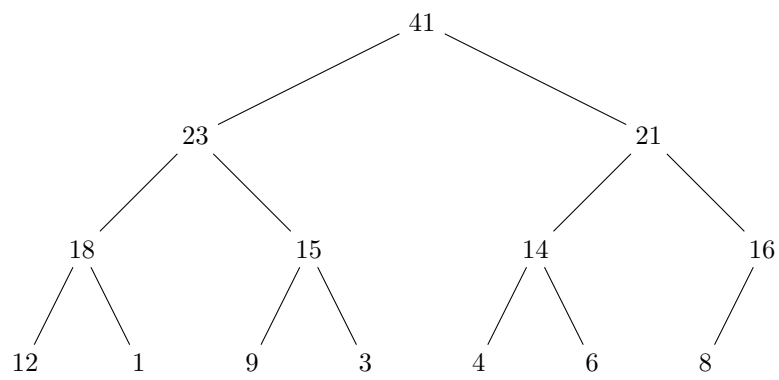
- **HEAPIFY(A, 2)**



- **HEAPIFY(A, 1)**

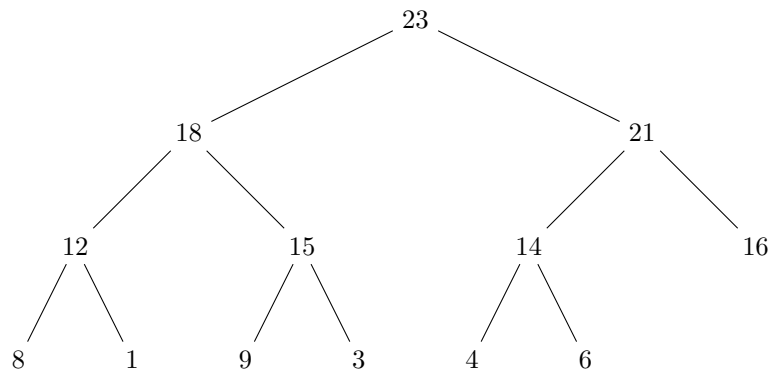


⋮

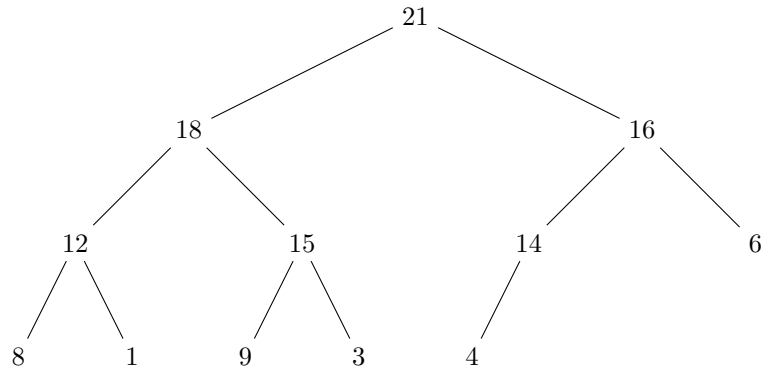


b)

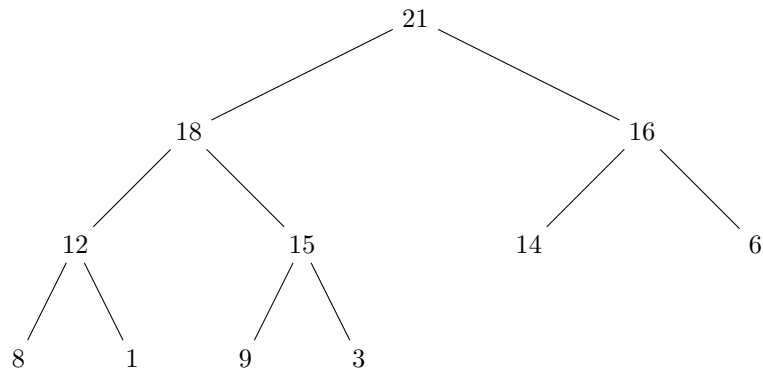
- **Nach 1x Heap-Extract-Max(A)**



- Nach 2x Heap-Extract-Max(A)



- Nach 3x Heap-Extract-Max(A)



Aufgabe 3

a)

- HeapDelete(A, i)
 - $A[i] \leftarrow A[\text{heapsize}[A]]$
 - $\text{heapsize}[A] \leftarrow \text{heapsize}[A] - 1$
 - MaxHeapify(A, i)

MaxHeapify hat eine Laufzeit von $O(\log n)$. Somit hat dieser HeapDelete Algorithmus ebenfalls eine Laufzeit von $O(\log n)$.

b)

```
• FindMin(A,i)
  min ← A[⌊ $\frac{n}{2}$ ⌋ + 1]
  for i ← ⌊ $\frac{n}{2}$ ⌋ + 2 to n do
    if A[i] < min then
      min ← A[i]
    end if
  end for
  return min
```

FindMin braucht $n - \lfloor \frac{n}{2} \rfloor + 1$ Schritte und hat somit eine Laufzeit von $O(n)$.

Aufgabe 4

a)

$$T(n) = 5 \cdot T\left(\frac{n}{2}\right) + n\sqrt{n}$$

b)

$$T(n) = 16 \cdot T\left(\frac{n}{8}\right) + n^{\frac{4}{3}}$$

c)

$$T(n) = 5 \cdot T\left(\frac{n}{7}\right) + n \log n$$

d)

$$T(n) = 2 \cdot T\left(\frac{n}{4}\right) + 3n^{\frac{1}{4}} + \log n$$

e)

$$T(n) = 16 \cdot T\left(\frac{n}{4}\right) + n^2 \log n$$