

# Tutorium Grundlagen: Algorithmen und Datenstrukturen

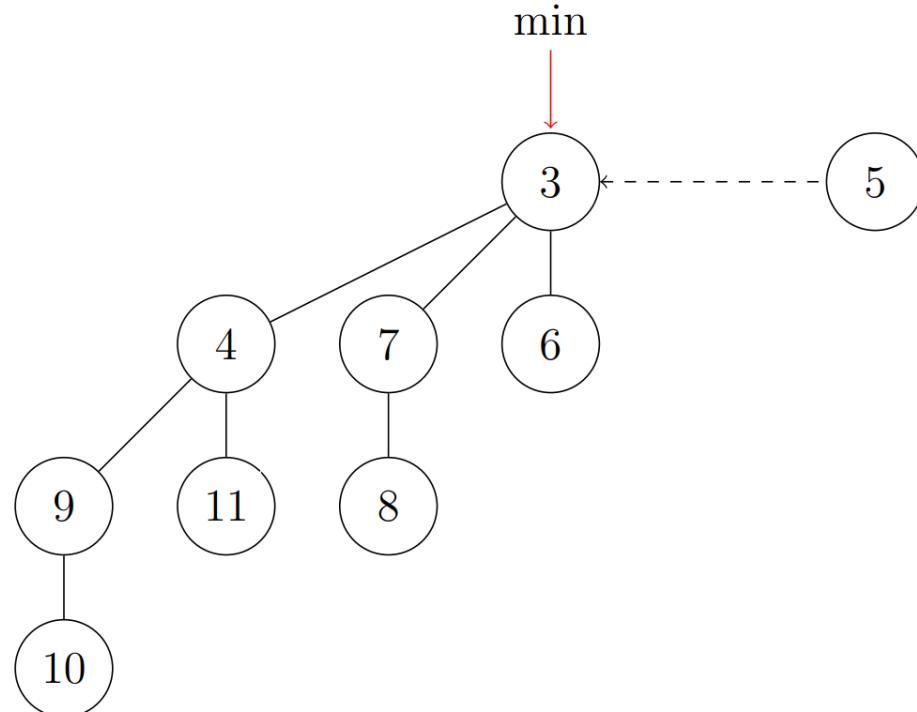
## Übungsblatt Woche 7

# Aufgabe 7.1

# Binomialheaps

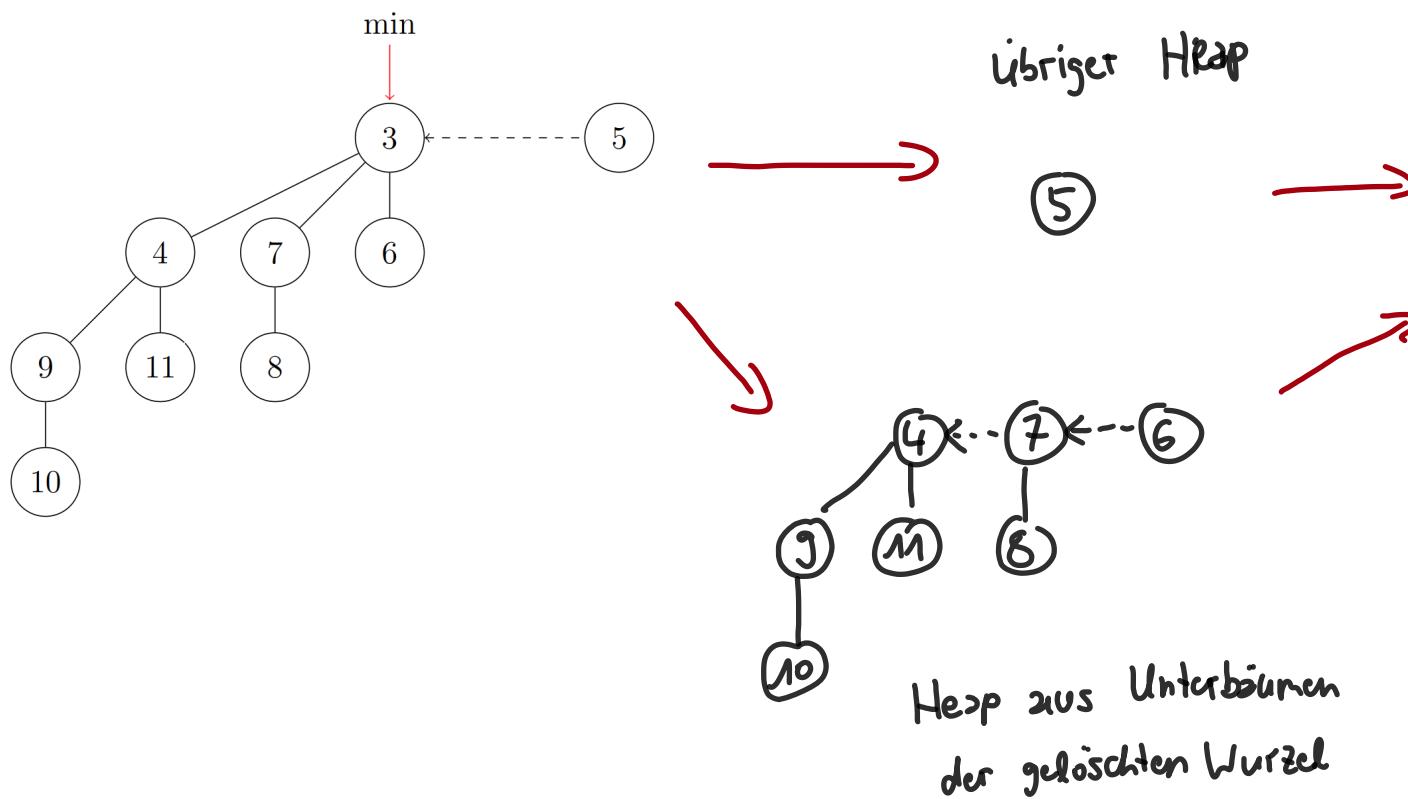
# Aufgabe 7.1

Führen Sie auf dem folgenden Binomial-Heap nacheinander drei deleteMin-Operationen aus. Fügen Sie anschließend die drei entfernten Elemente in der Reihenfolge, in der sie entfernt wurden, wieder hinzu. Zeichnen Sie nach jeder deleteMin- und insert-Operation den entstandenen Binomial-Heap. Sind nach allen Operationen die Werte an derselben Stelle im Heap? Hat der Heap nach allen Operationen dieselbe Struktur? Warum?



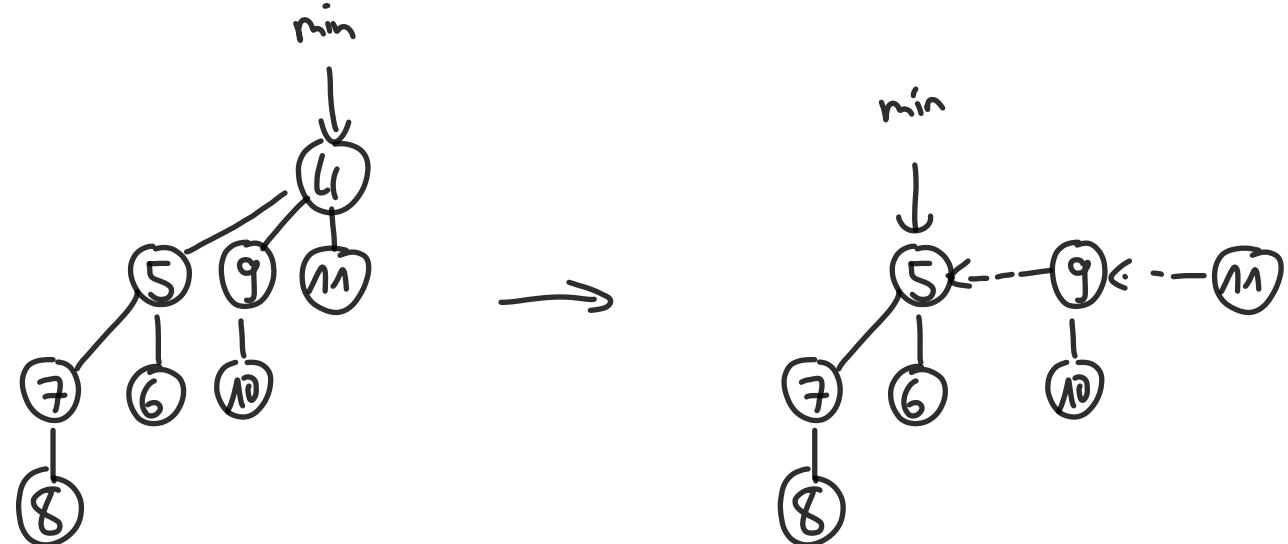
# Aufgabe 7.1

1. *deleteMin()*



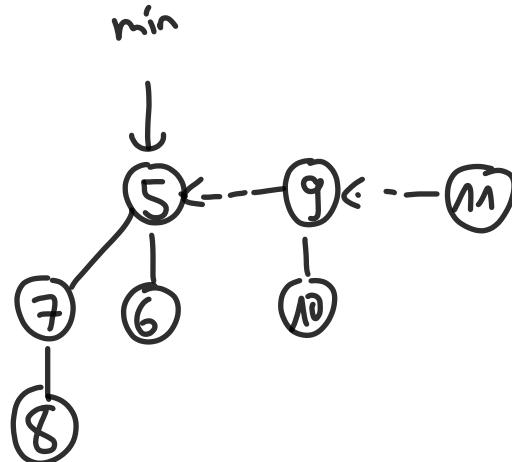
# Aufgabe 7.1

2. *deleteMin()*

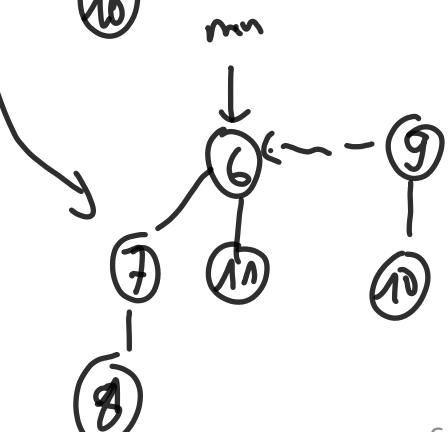
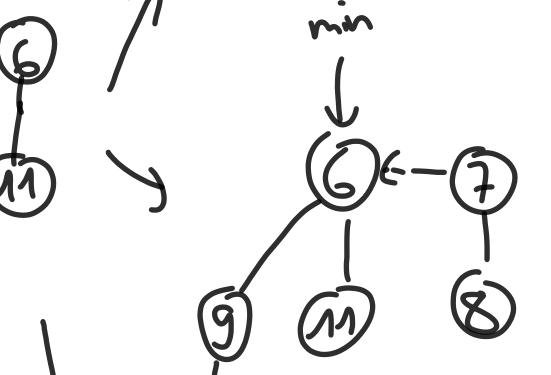
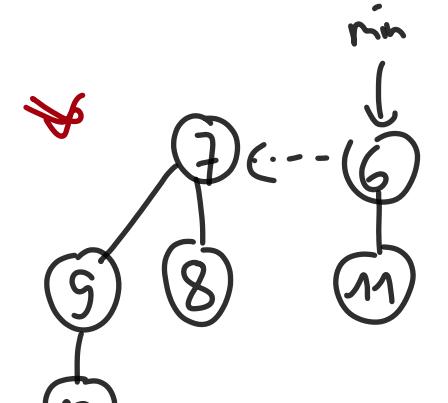
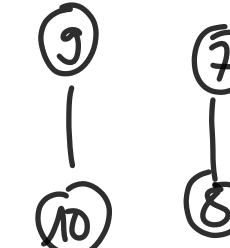


# Aufgabe 7.1

3. `deleteMin()`

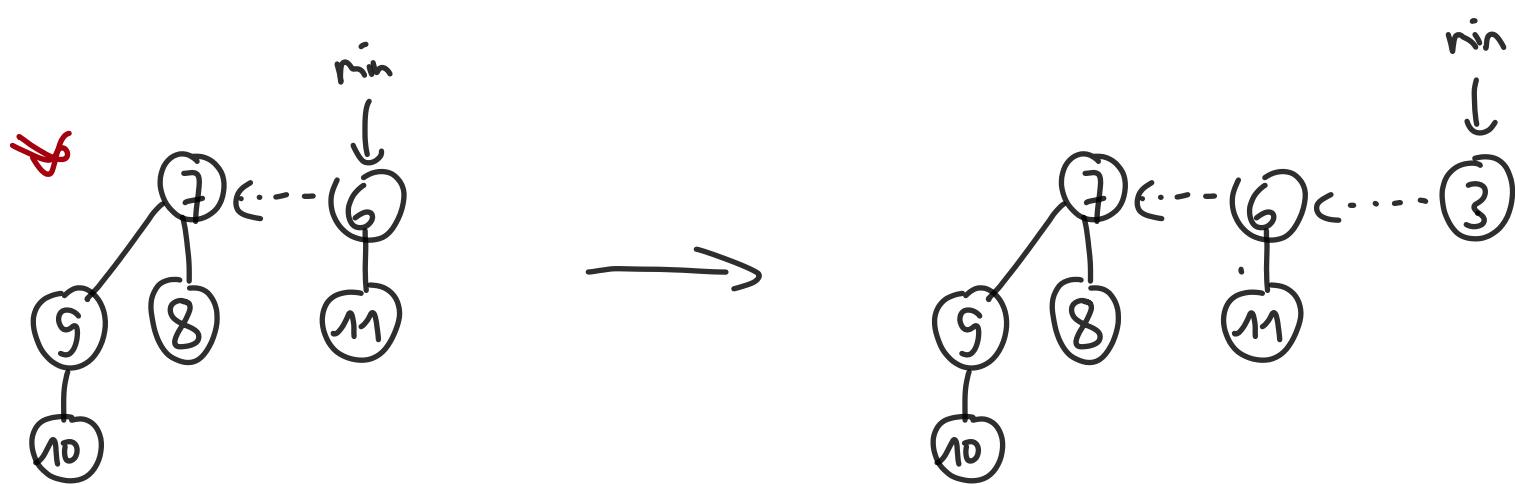


Merge die beiden  
Bäume mit Rang 0



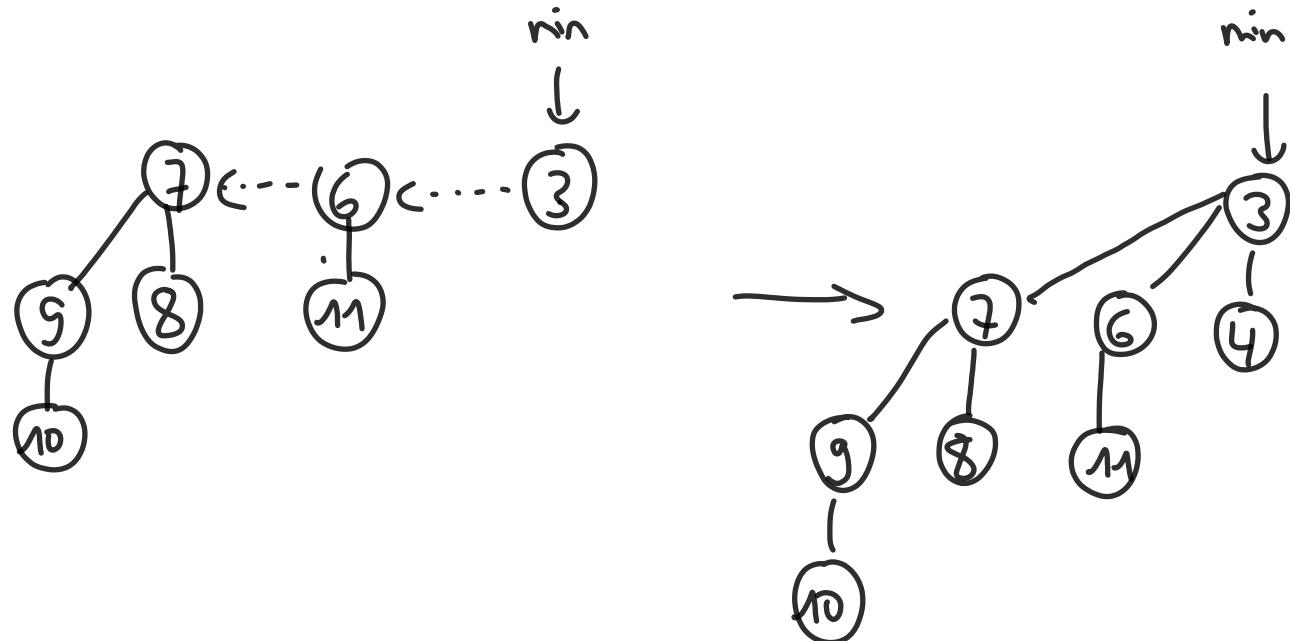
# Aufgabe 7.1

4.  $insert(3)$



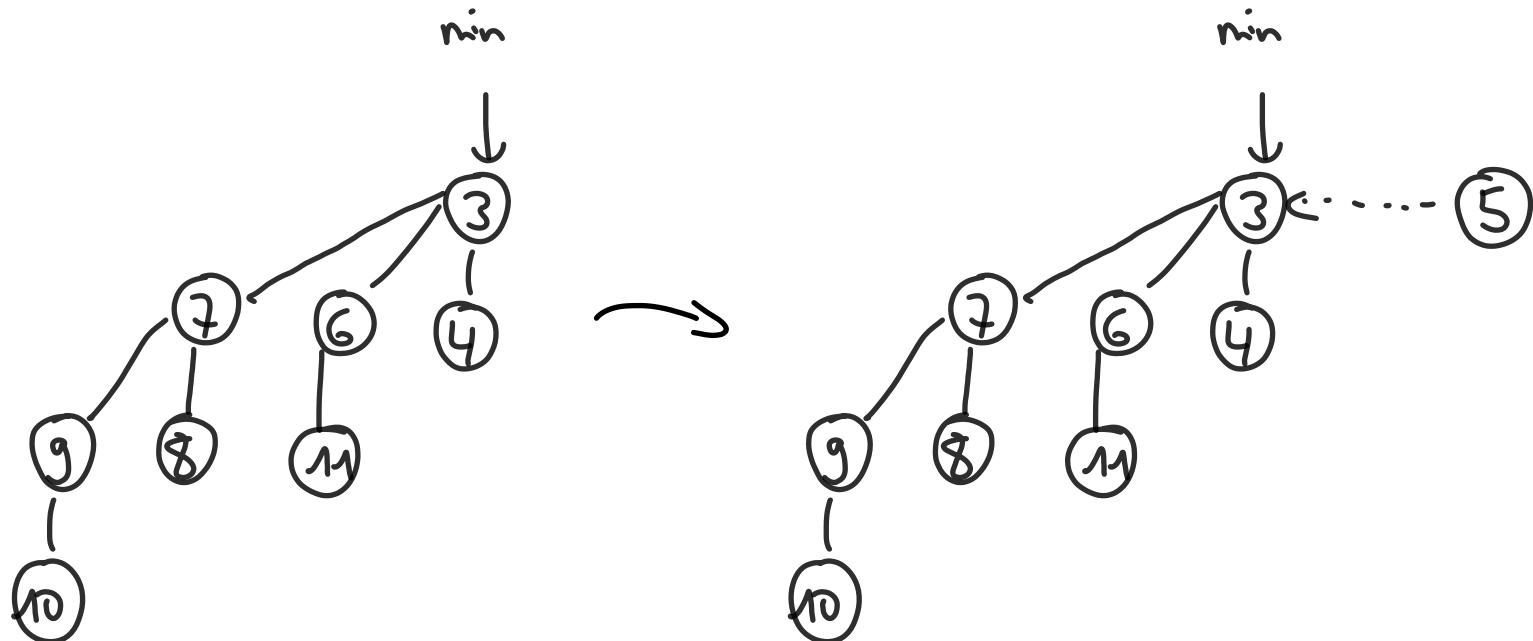
# Aufgabe 7.1

5. *insert(4)*



# Aufgabe 7.1

6.  $\text{insert}(5)$



Elemente an gleicher Stelle?

- Nein, bereits bei Schritt 3 gibt es 3 verschiedene Möglichkeiten
- Beim Einfügen müssen lediglich Elternknoten kleiner als Kindknoten sein

Gleiche Baumstruktur?

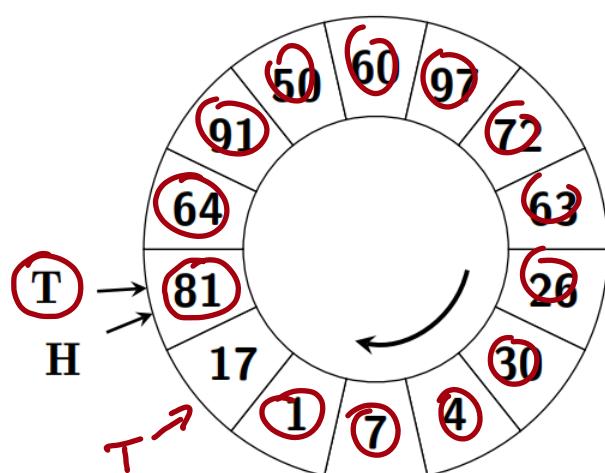
Ja, im Heaps darf jeder Baumring nur einmal vorkommen. Da die Ringe die Knotenzahl des Baumes als Zwischenpunkt angeben und jede Ganzzahl als eindeutige Summe von Zwischenpunkten geschrieben werden kann ist die Heapsstruktur mit  $n$  Knoten eindeutig.

# Aufgabe 7.2

## Rückblick: Circular Queues

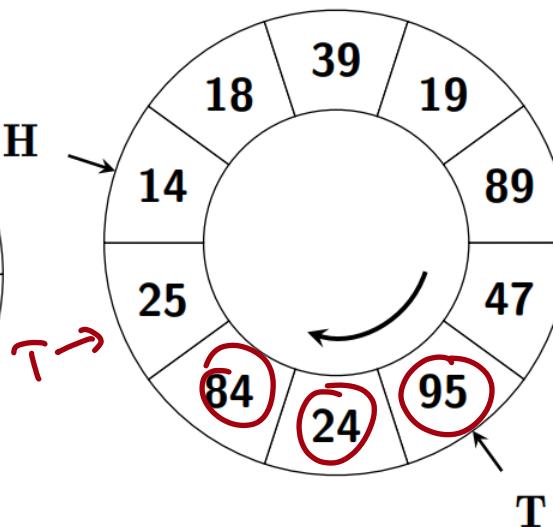
# Aufgabe 7.2 (a)

Gegeben sei ein Zirkulärer Ringspeicher als FIFO-Queue. In folgenden Darstellungen wird der Head mit **H** und der Tail mit **T** markiert. Geben Sie an, wie viele Inserts in folgenden Queues noch möglich sind. Dabei sollen Elemente, die aktuell in der Queue sind, weder überschrieben noch entfernt werden:



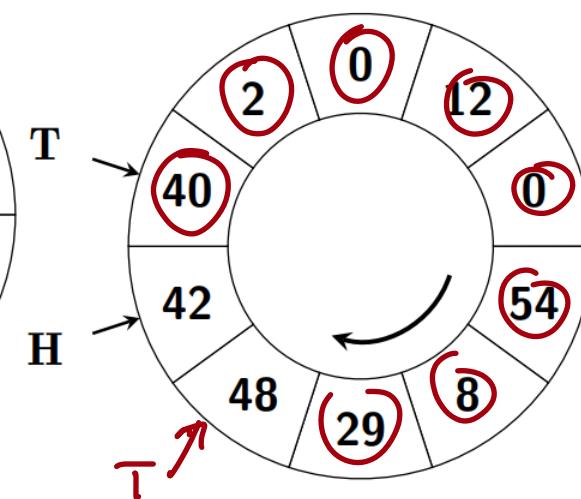
13

Inserts



3

Inserts

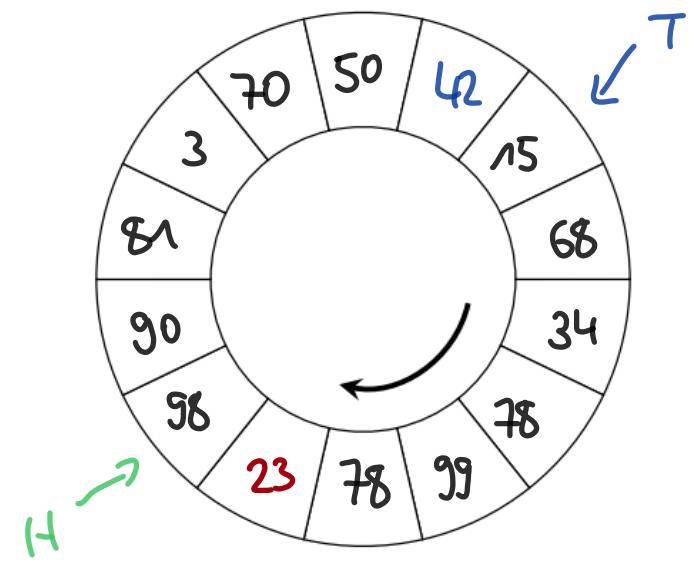
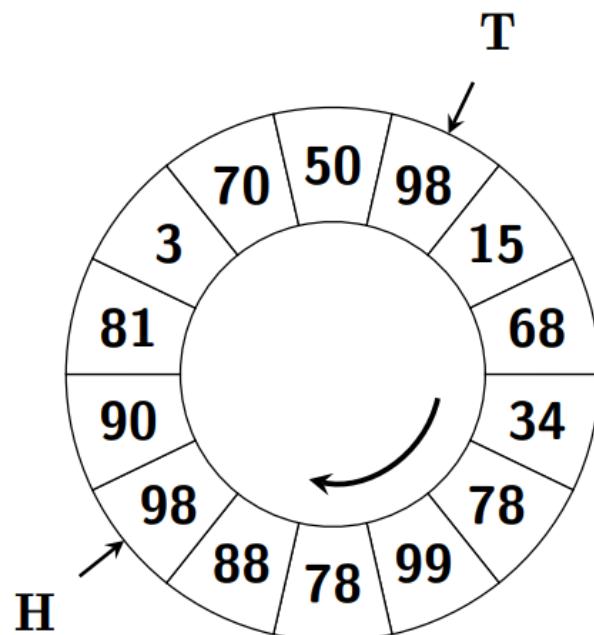


8

Inserts

# Aufgabe 7.2 (b)

Der folgende Ringspeicher wird nun als **Deque** genutzt. Geben Sie den Ringspeicher an, nachdem folgende Operationen ausgeführt wurden: **pushFront(23)**, **pushBack(42)**, **popFront()**



# Aufgabe 7.3

## Rückblick:

# Landausymbole

# Aufgabe 7.3

- a) Gegeben seien die Funktionen  $f, g$  mit  $f(n) = \log_2 n \cdot g(n)$ . Begründen Sie folgende Aussagen oder widerlegen Sie sie mit einem Gegenbeispiel:
- i) Aus  $g = \mathcal{O}(n)$  folgt  $f = \mathcal{O}(\log_2 n \cdot n)$
  - ii) Aus  $g = \sigma(n)$  folgt  $f = \omega(\log_2 n)$
  - iii) Aus  $f = \Theta(n^2)$  folgt  $g = \mathcal{O}(n^2)$
  - iv) Aus  $f \cdot g = \omega(n^4)$  folgt  $f + g = \Omega(n^2)$
- b) Geben Sie eine Funktion  $f : \mathbb{N}_0 \rightarrow \mathbb{R}^+$  an, die  $f = \sigma(\log_2 n)$  und  $\lim_{n \rightarrow \infty} f(n) = \infty$  erfüllt.

# Aufgabe 7.3 (a)

$$f(n) = \log_2(n) \cdot g(n)$$

- i) Aus  $g = O(n)$  folgt  $f = O(\log_2 n \cdot n)$

$$f(n) = \log_2(n) \cdot O(n) = O(\log_2(n) \cdot n) \quad \checkmark$$

- ii) Aus  $g = \sigma(n)$  folgt  $f = \omega(\log_2 n)$

$\times$

$$f(n) = \log_2(n) \cdot o(n)$$

- $g(n) = 1 \in o(n)$

$$\Rightarrow f(n) = \log_2(n) \cdot 1 = \log_2(n) + \omega(\log_2(n))$$

Gegenbeispiel gefunden

# Aufgabe 7.3 (a)

$$f(n) = \log_2(n) \cdot g(n)$$

- iii) Aus  $f = \Theta(n^2)$  folgt  $g = O(n^2)$

$$\Theta(n^2) = \log_2(n) \cdot g(n)$$

$$\frac{\Theta(n^2)}{\log_2(n)} = g(n) \rightarrow g(n) = \Theta\left(\frac{n^2}{\log_2(n)}\right) = O(n^2)$$

$$\frac{n^2}{\log_2(n)} \leq n^2 \quad \forall n \leq 2$$

- iv) Aus  $f \cdot g = \omega(n^4)$  folgt  $f + g = \Omega(n^2)$

$$\log_2(n) \cdot g(n) \cdot g(n) = \omega(n^4)$$

$$g(n)^2 \log_2(n) = \omega(n^4)$$

$$g(n)^2 = \frac{\omega(n^4)}{\log_2(n)}$$

$$g(n) = \sqrt{\frac{\omega(n^4)}{\log_2(n)}}$$

$$g(n) = \omega\left(\sqrt{\frac{n^4}{\log_2(n)}}\right) = \omega\left(\frac{n^2}{\sqrt{\log_2(n)}}\right) = \Omega\left(\frac{n^2}{\sqrt{\log_2(n)}}\right)$$

$$\begin{aligned} \log_2(n) \cdot g(n) + g(n) &= \log_2(n) \sqrt{\frac{n^2}{\log_2(n)}} + \sqrt{\frac{n^2}{\log_2(n)}} \\ &= \Omega\left(\log_2(n) \sqrt{\frac{n^2}{\log_2(n)}}\right) = \Omega\left(\sqrt{\frac{n^2}{\log_2(n)}} n^2\right) \subset \Omega(n^2) \end{aligned}$$

# Aufgabe 7.3 (b)

Geben Sie eine Funktion  $f : \mathbb{N}_0 \rightarrow \mathbb{R}^+$  an, die  $f = \sigma(\log_2 n)$  und  $\lim_{n \rightarrow \infty} f(n) = \infty$  erfüllt.

- $\sqrt{\log_2(n)}$

- $\lim_{n \rightarrow \infty} \left( \frac{\sqrt{\log_2(n)}}{\log_2(n)} \right) = \lim_{n \rightarrow \infty} \left( \frac{1}{\sqrt{\log_2(n)}} \right) = 0 \Rightarrow o(\log_2(n)) \quad \checkmark$

- $\lim_{n \rightarrow \infty} (\sqrt{\log_2(n)}) = \infty \quad \checkmark$

- $\log_2 \log_2(n)$

- $\lim_{n \rightarrow \infty} \left( \frac{\log_2 \log_2(n)}{\log_2(n)} \right) \Leftrightarrow \lim_{n \rightarrow \infty} \left( \frac{\ln \ln(n)}{\ln(n)} \right) = \lim_{n \rightarrow \infty} \left( \frac{\frac{1}{\ln(n)} \cancel{1/n}}{\cancel{1/n}} \right) = \lim_{n \rightarrow \infty} \left( \frac{1}{\ln(n)} \right) = 0$   
 $\Rightarrow o(\log_2(n)) \quad \checkmark$

- $\lim_{n \rightarrow \infty} (\log_2 \log_2(n)) = \infty \quad (\text{streng en Monotonie von } \log_2) \quad \checkmark$

$$\log_2(n) = \frac{\ln(n)}{\ln(2)}$$

$$\log_3(n) = \frac{\ln(n)}{\ln(3)}$$

$$\log_2(n) \cdot \ln(2) = \log_3(n) \cdot \ln(3)$$

$$\log_2(n) = \log_3(n) \cdot \underbrace{\frac{\ln(3)}{\ln(2)}}_{\text{konstant}}$$

$\Rightarrow$  Andere Log-Basis hat selbes asymptotisches Wachstum

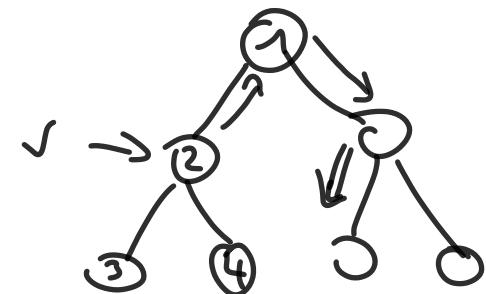
# Aufgabe 7.4

# Baumtraversierung

# Aufgabe 7.4

Ein nichtleerer Binärbaum kann (unter anderem) in PreOrder, InOrder und PostOrder traversiert werden. Diese Traversierungen sind wie folgt rekursiv definiert:

- PreOrder (entspricht dfs-Nummer | linker Teilbaum vor rechtem Teilbaum):
  - a) Besuche die Wurzel.
  - b) Traversiere den linken Teilbaum in PreOrder, falls dieser nicht leer ist.
  - c) Traversiere rechten Teilbaum in PreOrder, falls dieser nicht leer ist.
- InOrder:
  - a) Traversiere den linken Teilbaum in InOrder, falls dieser nicht leer ist.
  - b) Besuche die Wurzel.
  - c) Traversiere den rechten Teilbaum in InOrder, falls dieser nicht leer ist.
- PostOrder (entspricht (dfs-)finish-Nummer):
  - a) Traversiere den linken Teilbaum in PostOrder, falls dieser nicht leer ist.
  - b) Traversiere den rechten Teilbaum in PostOrder, falls dieser nicht leer ist.
  - c) Besuche die Wurzel.



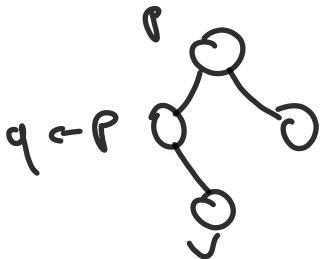
Geben Sie einen Algorithmus an welcher zu einem gegebenen Knoten  $v$  in einem Binärbaum den in PreOrder bzw. PostOrder folgenden Knoten berechnet.

Analysieren Sie die asymptotische Worst-Case-Laufzeit Ihres Pseudocodes.

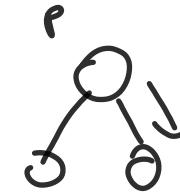
Berechnen Sie die asymp. Laufzeit beim vollständigen Durchlaufen des Baumes mithilfe dieser Operationen.

# Aufgabe 7.4

```
public Node preNext(Node v) {  
    if (hasleftChild(v))  
        return leftChild(v);  
    else if (hasrightChild(v))  
        return rightChild(v);  
    else {  
        Node p = v;  
        Node q;  
        while (!isRoot(p)) {  
            q = p;  
            p = parent(p);  
            if (hasrightChild(p) && rightChild(p) != q)  
                return rightChild(p);  
        }  
        return null;  
    }  
}
```



```
public Node postNext(Node v) {  
    if (!isRoot(v)) {  
        Node p = parent(v);  
        if (hasrightChild(p)) {  
            if (v==rightChild(p))  
                return p;  
            else {  
                Node c = rightChild(p);  
                while (isInternal(c)) {  
                    if (hasleftChild(c))  
                        c = leftChild(c);  
                    else  
                        c = rightChild(c);  
                }  
                return c;  
            }  
        }  
        else return p;  
    }  
    else return null;  
}
```



# Aufgabe 7.4

- Worst-Case-Laufzeit?
- While-Schleife durchläuft in beiden Fällen potentiell den Pfad von Wurzel zu tiefstem Blatt (oder umgekehrt)
- Worst-Case: Degenerierter Baum (Liste)  
↳ Alle Knoten Durchlaufen  $\Rightarrow \mathcal{O}(n)$
- Laufzeit bei vollständiger Traversierung?
  - Besichtige: In beiden Traversierungen wird jede Kante zweimal traversiert.
  - Jeder Knoten hat genau eine Kante zum Elternknoten (außer Wurzel mit 0)  
 $\Rightarrow$  Kantenanzahl beschränkt durch Knotenzahl n  
 $\Rightarrow$  Traversierung aller Kanten in  $\mathcal{O}(n)$

