

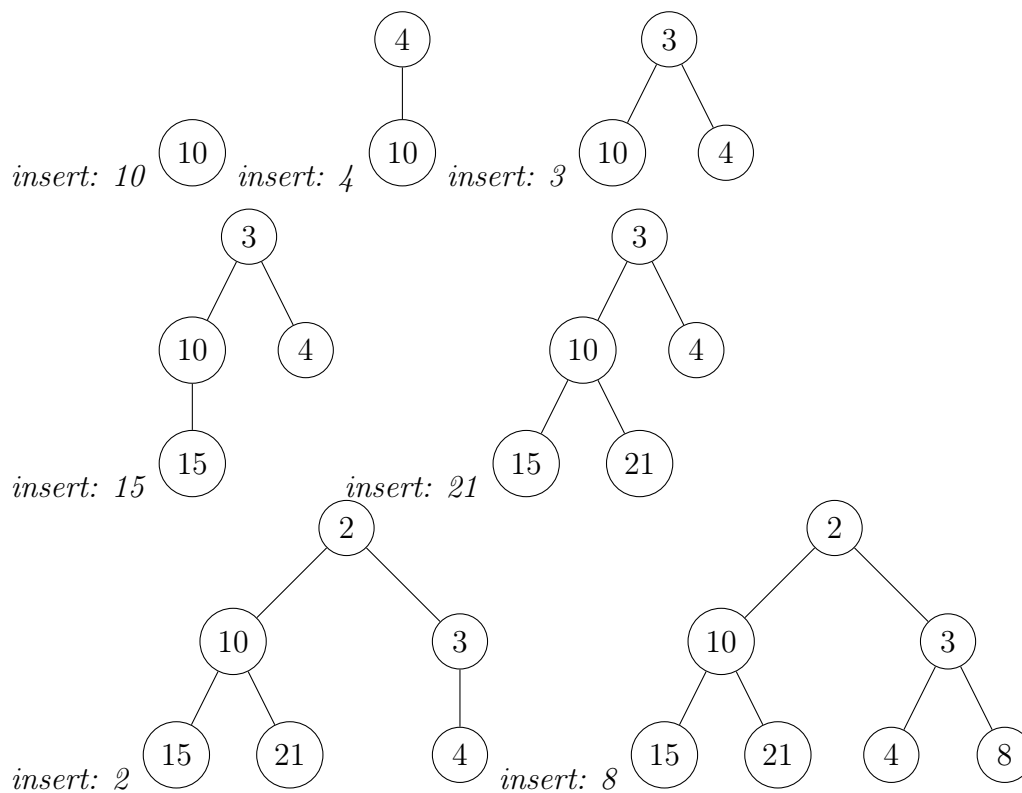
# Hausaufgabe 3

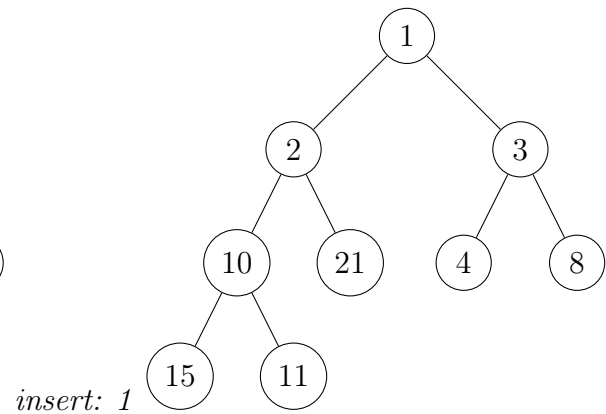
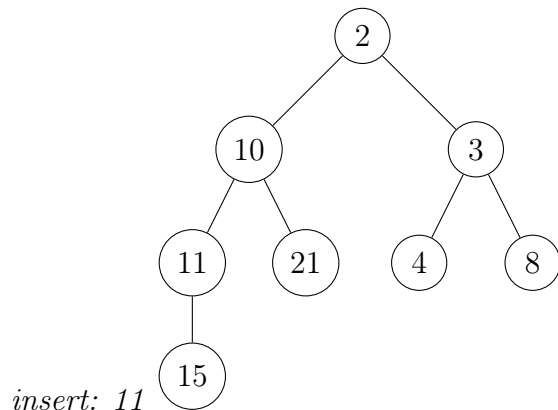
Aaron Sastry

3 mei 2022

## Aufgabe 3.2 Heaps

- a) Fügen Sie die Werte 10, 4, 3, 15, 21, 2, 8, 11 und 1 in einen anfangs leeren Heap ein. Stellen Sie nach jeder Einfüge-Operation den Heap als Baum dar und geben Sie das Array an, welches dem fertigen Heap entspricht.



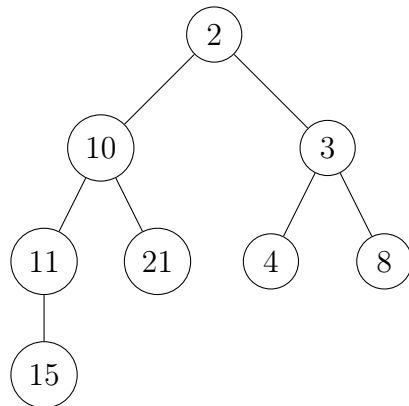
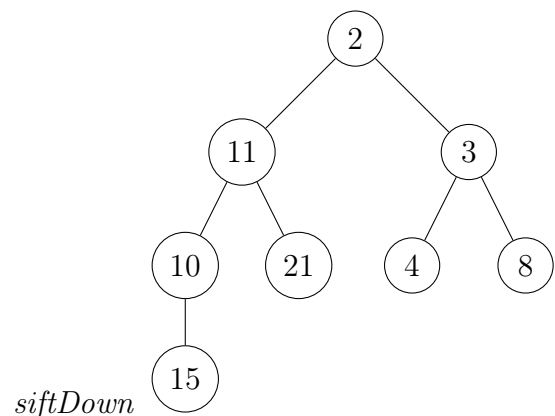
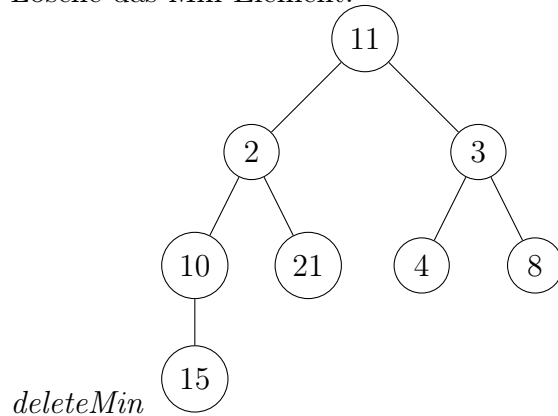


Das array zu diesem heap sieht wie folgt aus:

$heap = [1, 2, 3, 10, 21, 4, 8, 15, 11]$

- b) Führen Sie auf dem soeben gebauten Heap zwei deleteMin-Operationen durch und geben Sie jeweils den resultierenden Heap in Baumdarstellung und als Array an.

Lösche das Min-Element:



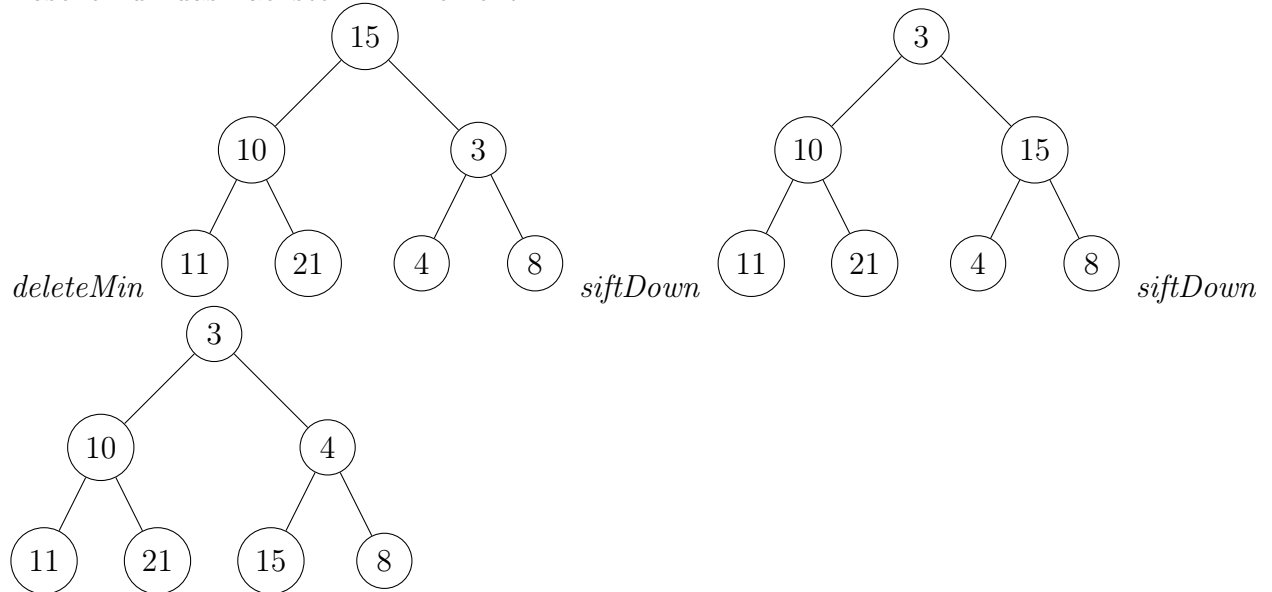
Min-Element gelöscht

output: *deleted element is 1*

Neues Min-Element ist: 2

der heap sieht nun wie folgt aus:  $[2, 10, 3, 11, 21, 4, 8, 15]$

Lösche nun das nächste Min-Element:



Min-Element gelöscht

output: *deleted element is 2*

Neues Min-Element ist: 3

der heap sieht nun wie folgt aus:  $[3, 10, 4, 11, 21, 15, 8]$

- c) Beschreiben Sie einen Algorithmus, der  $k$  sortierte Listen mit Gesamtlänge  $n$  in  $O(n \log k)$  Zeit zu einer sortierten Liste zusammenfügt. Benutzen Sie dabei einen Heap. Begründen Sie kurz, dass Ihr Algorithmus die Laufzeitschranke einhält.

```
sortListwithHeap(A: Array of k lists)
```

```
    sei B ein leerer heap
```

```
    sei C ein leeres array
```

```
    sei D ein leeres array
```

```
    for i ← 1 to k do
```

```
        /*tupel aus der value des minElements
        der list und der listenNummer */
        C.append([A[i][1], k])
```

Remove(A[i][1])

B. BuildMinHeap(C)

```
for i ← 1 to n do
    item = B.minElement
    listnumber = item[2]
    deleteMin()
    D.append(item[1])

    if A[listnumber] nicht leer
        B.insert((A[listnumber][1], listnumber))
        Remove(A[listnumber][1])
```

Dieser Algorithmus hält die Schranke ein, da deleteMin() und insert() laut VL in  $O(\log(n))$  sind, und dies wird  $n$  mal getan  $\rightarrow O(n \log(n))$

- d) Gegeben sei die folgende alternative Prozedur zum Erstellen eines binären Heaps für ein unsortiertes Array A[1..n]:

```
buildHeapInsert(A : Array):
    for i ← 1 to n do
        insert(A[i])
```

Geben Sie ein Beispiel für eine Eingabe an, sodass buildHeapInsert eine schlechtere Laufzeit für das Aufbauen des Heaps hat als  $O(n)$ . Was ist die worst-case Laufzeit von buildHeapInsert? Begründen Sie!

**Antwort:**

Im schlechtesten Falle ist für jedes Element  $i$ , dass eingefügt wird das folgende  $i+1$  Element, welches dannach eingefügt wird kleiner als  $i$ , sodass jedes Element immer per siftUp bis auf die Min-Position hoch gegeben werden muss.

Somit würde Insert des  $i$ -ten Elements in  $\lfloor \log(i) \rfloor$  passieren. Und der gesamte Aufbau wäre in:

$$O(\sum_{i=1}^n \log(i)) \iff O(\log(n!))$$

Sobald nun  $\log(i) > 1$ , wird auch die Summe  $\sum_{i=1}^n \log(i)$  bald größer als die Summe  $\sum_{i=1}^n 1 \implies O(\log(n!)) > O(n)$

$$\begin{aligned} &\text{Stirling's approximation} \\ &\implies O(\log(n!)) = O(n \log(n)) \end{aligned}$$