



b) Wenn die Tiefe  $h=5 \Rightarrow$  max. Anzahl der Elemente  $= 2^6 - 1 = 63$

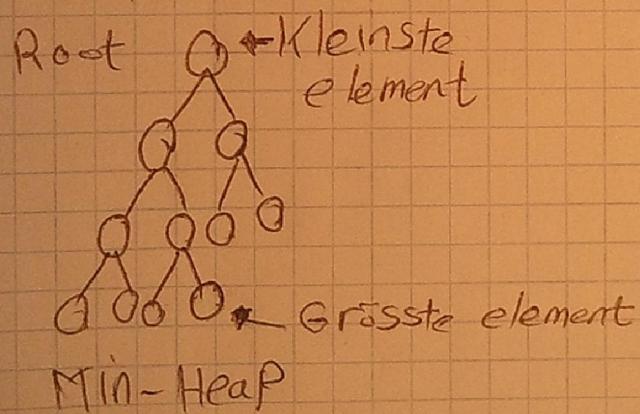
Wenn die Tiefe  $h=6 \Rightarrow$  max. Anzahl der Elemente  $= 2^7 - 1 = 127$

da wir 100 Elementen haben, die letzte Ebene noch nicht vollständig belegt ist

dann  $\boxed{h=6}$ , wie oben in diesem Fall sind die max. Anzahl von Elementen  $= 127$  (wenn es vollständig belegt ist)

also 27 Elemente fehlen in der letzten Ebene.

c) In Min-Heaps ist das grösste Element in der letzten Stelle, das kleinste Element im Root



## Aufgabe 14

# Abbildung eines Heaps auf ein Array, für Parent Funktion

```
int Parent(i){  
    return  $\lfloor (i-1)/2 \rfloor$   
}
```

Binary Heap

```
int Parent(i){  
    return  $\lfloor (i-1)/3 \rfloor$   
}
```

Ternary Heap

```
int Parent(i){  
    return  $\lfloor (i-1)/4 \rfloor$   
}
```

Quaternary Heap

→  $\rightarrow$   
 $\text{int Parent}(i)\{$   
 $\quad \text{return } \lfloor (i-1)/k \rfloor$   
}

K-ary Heap

im Allgemeinen Fall, wobei  
 $k = \text{Anzahl der Kinder}$   
z.B.  $1 \leq i \leq \text{Heapsize}$

# Abbildung eines Heaps auf ein Array, für child Funktion

```
int child(i,j){  
    if (j==1)  
        return 2i+1  
    if (j==2)  
        return 2i+2  
}
```

Binary Heap

```
int child(i,j){  
    if (j==1)  
        return 3i+1  
    if (j==2)  
        return 3i+2  
    if (j==3)  
        return 3i+3  
}
```

Ternary Heap

```
int child(i,j){  
    if (j==1)  
        return 4i+1  
    if (j==2)  
        return 4i+2  
    if (j==3)  
        return 4i+3  
    if (j==4)  
        return 4i+4  
}
```

Quaternary Heap

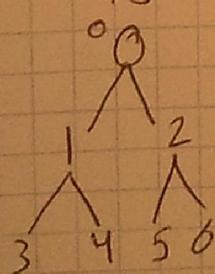
```

====> int child(i,j) {
    if(j==1)
        return iK+1
    if(j==2)
        return iK+2
    if(j==3)
        return iK+3
    :
    if(j==K)
        return iK+K
}

```

K-arry Heap, wobei  $K :=$  Anzahl der Kinder  
 (im Allgemeinen Fall)

allediese Fälle könnte man auch mit modulo Arithmetik  
 darstellen, aber der Code wird Komplexer, jedoch  
 ist es übersichtlich wenn  $K=2$  (im Fall von Binary Heap)



Arithmetik modulo 2

$\text{if}(i/2 == 0) \Rightarrow$  rechts Kind  
 $\text{if}(i/2 == 1) \Rightarrow$  Linkskind

Allgemein Arithmetik modulo(K)  
 wobei  $K :=$  Anzahl der Kinder

# Aufgabe 15)

a) Aufsteigend: Durch Build-Max-Heap(A) wird abhängig von der Eingabegröße ein "unregelmäßiger" Max-Heap erzeugt und es entsteht kein Sonderfall.  
=>  $O(n \cdot \log(n))$

Aufsteigend: Es besteht bereits ein Max-Heap. Der M-H-Build(A) arbeitet umsonst.

In unserem Array befinden sich nun die kleinsten Zahlen auf unterster Ebene.

Der Sort beginnt damit die kleinste Zahl an ~~an der~~ die erste Stelle zu tauschen, um die größte ab zu schneiden.

Dadurch muss der Heapsy der auf die erste Stelle (= kleinste Zahl) angewendet wird mit Sicherheit alle Ebenen durchtauschen damit die kleinste Zahl wieder korrekt im Max-Heap platziert ist.

Da die nächste Zahl, die gebraucht wird eine der beiden kleinen Zahlen sein muss, muss Heapsy(A) wieder alle Ebenen durchtauschen. =>

15) a) fals.

Somit wird im Endeffekt jeder ASt im +  
Somit benötigt Max-Heapify immer kteam  
Ausführungen.

Die Anzahl Ebenen ist gleich  $\lfloor \log_2(n) \rfloor$ .

Damit <sup>hat</sup> schlimmsten Fall

Damit hat Max-Heapify ~~etwa~~ bei verschärften  
ungünstig vorvorherierten Arrays  $O(\log(n))$ .

Weil durch die For-Schleife in Zeilen 2-5  
das Ganze n-mal aufgeworfen wird benötigt ist  
die Gesamt Laufzeit in  $O(n \log(n))$   
immer noch

b)

Heap-Priority:

Wie gezeigt besitzt Heapify (= upheap)  
 $O(\log(n))$  Laufzeit.

Da in insert genau ein mal upheap() aufgerufen wird (+ konstante (add(1))) besitzt upheap  $O(\log(n))$  Laufzeit.

Insertion sort:

Da im schlimmsten Fall zum einsetzen  
eines Elementes alle durchlaufen werden müssen,  
besitzt InsSort  $O(n^2)$