Quiz

$5e^{2\ln(n)} = \Theta(e^{10\ln(n)})$

- True
- False

$5e^{2\ln(n)} = 5e^{\ln(n^2)} = 5n^2$

$e^{10\ln(n)} = e^{\ln(n^{10})} = n^{10}$

Suppose you have some algorithm A and let an increasing function $T(n)$ be its runtime on an input of size n. Assume you know that $T(n) \leq T(n-1) + 100n$ and that $T(1) = 10$. Then $T(n) \leq O(n^2)$.

- True
- False

schon gesehen bei QuickSort

$$\begin{aligned} T(n) &\leq T(n-1) + 100n \\ &\leq T(n-2) + 100n + 100(n-1) \\ &\vdots \\ &\leq T(1) + 100 \cdot \sum_{i=1}^{n} i \\ &= 10 + 100 \cdot \frac{n(n+1)}{2} \\ &\leq O(n^2) \end{aligned}$$

Every comparison-based sorting algorithm needs to perform at least $\Omega(n\log(n))$ swaps.

- True
- False

0 swaps if array is already sorted for Bubble Sort

Suppose we apply *insertion sort* to the array $A_n = [n, n-1, n-2, \ldots, 3, 2, 1]$.

(E.g., $A_5 = [5, 4, 3, 2, 1]$)

Let $s(n)$ be the number of **swap** operations that are performed before the array is fully sorted (in ascending order).

Select one:

a. $s(n) \geq \Omega(n^2)$

b. $s(n) \leq O(n)$

Iteration j braucht $j-1$ swaps

$$\sum_{j=1}^{n} j-1 = \sum_{i=0}^{n-1} i = \left(\sum_{i=0}^{n} i\right) - n \geq \Omega(n^2)$$

Which of the following sorting algorithms have the invariant that: after j steps, the first j elements are sorted?

(A step refers to one iteration of the outer for loop for each sorting algorithm).

Select one or more:

- a. Merge sort
- b. Bubble sort
- c. Insertion sort
- d. Selection sort

Asymptotically (in Θ-notation), *merge sort*, *heap sort*, and deterministic *quick sort* all have the same worst-case runtime.

$\Theta(n \log n)$ $\Theta(n^2)$

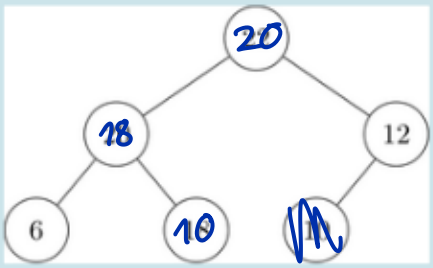
- True
- False

Suppose all keys in a max-heap are unique. Then the node with the minimum key is always the left-most leaf.

- True
- False

Counterexample

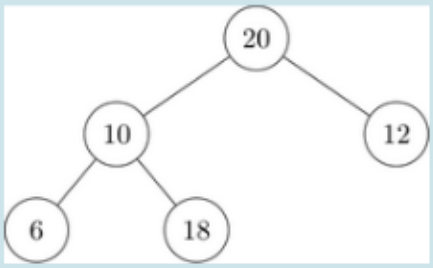
Consider the following max-heap,



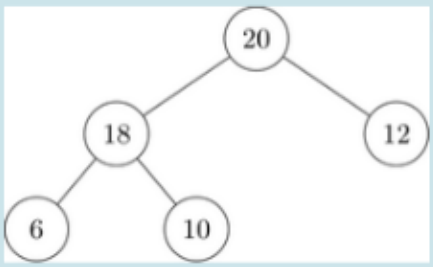
After executing one **`ExtractMax`** operation, what is the correct max-heap?

Select one:

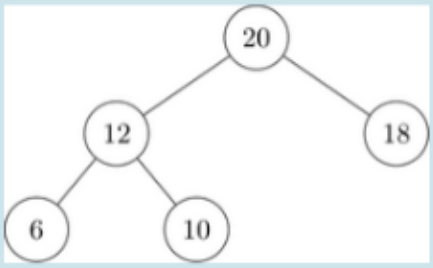
a.



b.



c.



Master Theorem

Theorem 1 (Master theorem). *Let $a, C > 0$ and $b \geq 0$ be constants and $T : \mathbb{N} \to \mathbb{R}^+$ a function such that for all even $n \in \mathbb{N}$,*

$$T(n) \leq aT(n/2) + Cn^b. \tag{1}$$

Then for all $n = 2^k$, $k \in \mathbb{N}$, the following statements hold

(i) If $b > \log_2 a$, $T(n) \leq O(n^b)$.

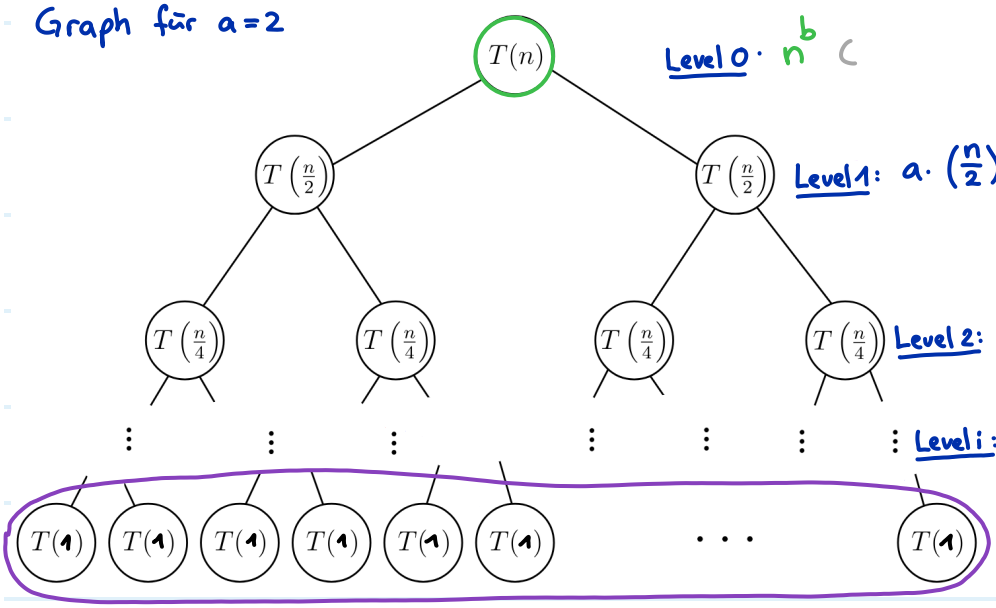
(ii) If $b = \log_2 a$, $T(n) \leq O(n^{\log_2 a} \cdot \log n)$.

(iii) If $b < \log_2 a$, $T(n) \leq O(n^{\log_2 a})$.

If the function T is increasing, then the condition $n = 2^k$ can be dropped.

$T(n) \leq \underbrace{a \cdot T(\frac{n}{2})}_{\text{Rekursion}} + \underbrace{Cn^b}_{\text{Arbeit für den Knoten selbst}}$

Graph für $a = 2$



Level 0: $n^b \cdot C$

Level 1: $a \cdot \left(\frac{n}{2}\right)^b \cdot C$

Level 2: $a^2 \cdot \left(\frac{n}{2^2}\right)^b \cdot C$

Level i: $a^i \left(\frac{n}{2^i}\right)^b \cdot C$

Level $\log_2 n$: $a^{\log_2 n} \cdot \underbrace{\left(\frac{n}{2^{\log_2 n}}\right)^b}_{=1} C = n^{\log_2 a} C$

Total: $\sum_{i=0}^{\log_2 n} a^i \left(\frac{n}{2^i}\right)^b = \sum_{i=0}^{\log_2 n} a^i \frac{n^b}{2^{ib}} = n^b \sum_{i=0}^{\log_2 n} \frac{a^i}{2^{ib}} = n^b \sum_{i=0}^{\log_2 n} \left(\frac{a}{2^b}\right)^i$

Potenzgesetz — Distributivgesetz — Potenzgesetz

Fall $b = \log_2 a$: Alle Level haben gleich viel Arbeit

$\Rightarrow O(n^b \cdot \log n)$ Höhe des Baumes = Anzahl Level = $\log_2 n$

Fall $b > \log_2 a$: Oberstes Level dominiert und alle anderen werden vernachlässigt

$\Rightarrow O(n^b)$

Fall $b < \log_2 a$: Unterstes Level dominiert und alle anderen werden vernachlässigt

$\Rightarrow O(n^{\log_2 a})$

Fall $a = 2^b \Leftrightarrow \log_2 a = b$: $n^b \sum_{i=0}^{\log_2 n} \left(\frac{a}{2^b}\right)^i = n^b \sum_{i=0}^{\log_2 n} 1^i \leq O(n^b \cdot \log n)$

Fall $a < 2^b \Leftrightarrow \log_2(a) < b$: $n^b \sum_{i=0}^{\log_2 n} \left(\frac{a}{2^b}\right)^i \leq n^b \cdot \underbrace{\frac{1}{1 - \frac{a}{2^b}}}_{\text{Konstante}} \leq O(n^b)$

↪ Geometrische Reihe $\sum_{i=0}^{\infty} r^i = \frac{1}{1-r}$ für $r < 1$

Fall $a > 2^b \Leftrightarrow \log_2(a) > b$:

Geometrische Reihe $\sum_{i=0}^{k} r^i = \frac{r^{k+1} - 1}{r - 1}$ für $r > 1$

$$n^b \sum_{i=0}^{\log_2 n} \left(\frac{a}{2^b}\right)^i = n^b \cdot \frac{\left(\frac{a}{2^b}\right)^{\log_2 n + 1} - 1}{\frac{a}{2^b} - 1}$$

↳ Konstanten weglassen

$$\leq O\left(n^b \cdot \left(\frac{a}{2^b}\right)^{\log_2 n}\right)$$

↳ Potenzgesetz

$$\leq O\left(n^b \cdot \frac{a^{\log_2 n}}{2^{b \cdot \log_2 n}}\right)$$

↳ Logarithmusgesetz

$$\leq O\left(n^b \frac{a^{\log_2 n}}{2^{\log_2(n^b)}}\right)$$

$$\leq O\left(n^b \frac{a^{\log_2 n}}{n^b}\right)$$

$$\leq O\left(a^{\log_2 n}\right)$$

↳ Logarithmusgesetz $x^{\log y} = y^{\log x}$

$$\leq O\left(n^{\log_2 a}\right)$$

Exercise 1

Let $T(1) = 3, T(n) = 2T(n/2) + n^2$ for $n > 1$. Using the master theorem, show that

$$T(n) \leq O(n^2).$$

$a = 2$ und $b = 2$

$\log_2 a = \log_2 2 = 1 < 2 = b$. Wir wenden Fall (i) an: $T(n) \leq O(n^2)$

Algorithm 3

(b)
```
function A(n)
    i ← 0
    while i < n^2 do
        j ← n
        while j > 0 do
            f()
            f()
            j ← j − 1
        i ← i + 1
    k ← ⌊n/2⌋
    for l = 0 ... 3 do
        if k > 0 then
            A(k)
            A(k)
```

$$\#\text{calls} = \sum_{i=0}^{n^2-1} \sum_{j=1}^{n} 2 + \sum_{\ell=0}^{3} 2 \cdot A\left(\lfloor \tfrac{n}{2} \rfloor\right)$$

$$= \sum_{i=0}^{n^2-1} 2n + 8 \cdot A\left(\lfloor \tfrac{n}{2} \rfloor\right)$$

$$= 2n^3 + 8 \cdot A\left(\lfloor \tfrac{n}{2} \rfloor\right)$$

Mit $a = 8, b = 3, c = 2$ und $\log_2 a = \log_2 8 = 3 = b$ wenden wir Fall (ii) des Master Theorems an und erhalten $\Theta(n^3 \log n)$ Funktionsaufrufe

Algorithm 5

```
function C(n)
    if n ≤ 1 then
        return 0
    end if
    j ← 1
    while √j ≤ n^3 do
        f()
        j ← j + 1
    end while
    k ← ⌊n/2⌋
    C(k)
    C(k)
    C(k)
end function
```

$$\#\text{calls} = \sum_{j=1}^{n^6} 1 + 3C\left(\lfloor \tfrac{n}{2} \rfloor\right)$$

$$= 3C\left(\lfloor \tfrac{n}{2} \rfloor\right) + n^6$$

$$\overset{\text{Master Th.}}{=} \Theta(n^6)$$

$3 < 2^6$

$\Rightarrow \log_2(3) < 6$

Sortieren

Entscheidungsbaum:

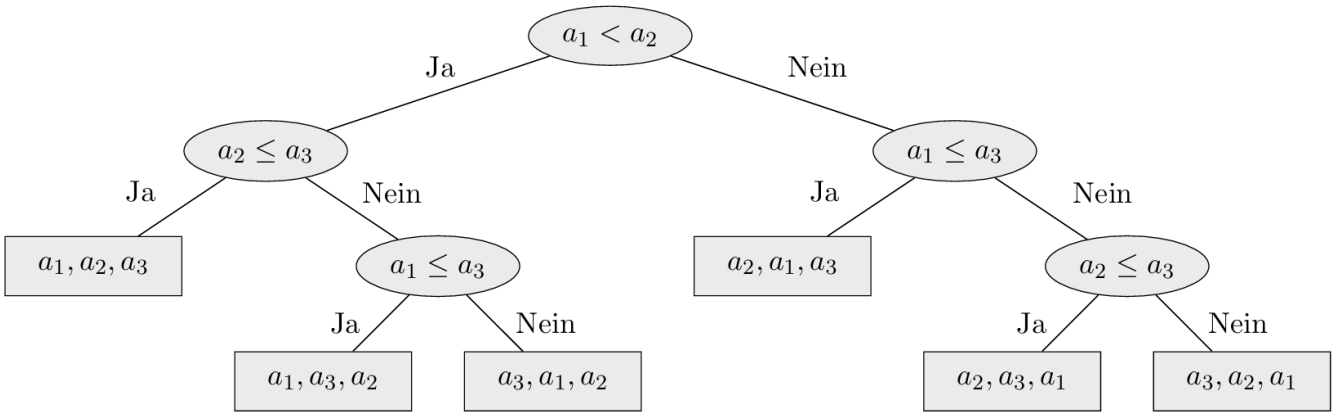


Abb. 2.5 Beispiel Entscheidungsbaum mit Array $[a_1, a_2, a_3]$

Wie viele Blätter muss der Baum haben? $n! = n\,(n-1)\cdot(n-2)\cdot\ldots\cdot 2\cdot 1$

Höhe h hat 2^h Knoten

$$2^h \geq n!$$

$$\Rightarrow 2^h \geq \left(\frac{n}{2}\right)^{\frac{n}{2}}$$

$$\Rightarrow h \geq \log_2\left(\left(\frac{n}{2}\right)^{\frac{n}{2}}\right)$$

$$\Rightarrow h \geq \frac{n}{2} \cdot \log_2\left(\frac{n}{2}\right)$$

$$\Rightarrow h \geq \Omega(n \log n)$$

Alternativ: 1 Vergleich heisst wir können 2 Permutationen unterscheiden

2 Vergleiche heisst wir können 4 Permutationen unterscheiden

i Vergleiche heisst wir können 2^i Permutationen unterscheiden

wir müssen n! Permutationen unterscheiden $\Rightarrow$ wir brauchen $\log_2(n!)$ Vergleiche

$\geq \Omega(n \log n)$

QUICKSORT$(A[1..n], l, r)$

```
1  if l < r then
2      k ← Aufteilen(A, l, r)        ▷ Teile A[l..r] in zwei Gruppen auf
3      Quicksort(A, l, k − 1)        ▷ Sortiere linke Gruppe
4      Quicksort(A, k + 1, r)        ▷ Sortiere rechte Gruppe
```

AUFTEILEN$(A[1..n], l, r)$

```
1  i ← l                             ▷ Index für die linke Gruppe
2  j ← r − 1                         ▷ Index für die rechte Gruppe
3  p ← A[r]                          ▷ Pivotelement
4  repeat
5      while i < r and A[i] ≤ p do
6          i ← i + 1                 ▷ Suche nächstes Element für rechte Gruppe
7      while j > l and A[j] > p do
8          j ← j − 1                 ▷ Suche nächstes Element für linke Gruppe
9      if i < j then
10         Vertausche A[i] und A[j]  ▷ Vertausche Elemente
11 until i ≥ j                       ▷ Schleife endet, wenn sich i und j „treffen“
12 Vertausche A[i] und A[r]          ▷ Pivotelement an die richtige Stelle tauschen
13 return i                          ▷ Gib Position des Pivotelements zurück
```

Bsp [3 7 8 9 4 2 11 1 5] (i, j, Pivot)

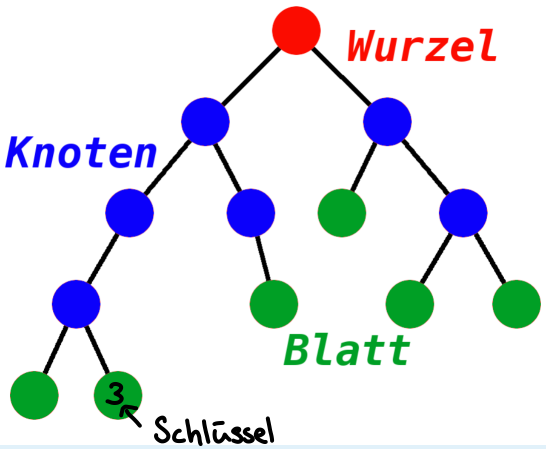
Best case: $T(n) \leq \underbrace{2 \cdot T(\frac{n}{2})}_{\text{Rekursion}} + \underbrace{c\,n}_{\text{Aufteilen}}$ $\overset{\text{Master Theorem}}{\Rightarrow}$ $T(n) \leq O(n \log n)$

Worst case $T(n) \leq T(n-1) + c\,n \leq T(n-2) + c\,(n + (n-1)) \leq T(n-3) + c\,(n + (n-1) + (n-2))$

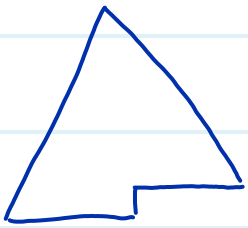
$\overset{\text{Teleskopieren}}{=}$ $T(n) \leq O(n^2)$

Randomisiert: $\mathbb{E}[T(n)] \leq O(n \log n)$

Max Heap



1. *Vollständiger Binärbaum*: Jeder Knoten hat genau zwei Kinder, ausser eventuell in den beiden untersten Levels des Baums. Das unterste Level ist von links nach rechts aufgefüllt. Somit haben Knoten im untersten Level keine Kinder, und im zweituntersten Level können Knoten mit 0, 1, oder 2 Kindern auftreten.



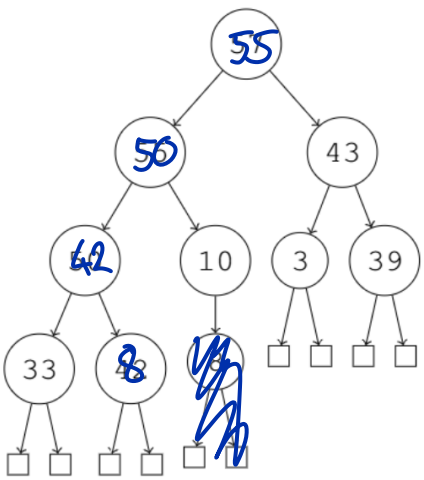
2. *Heap-Bedingung*: Der Schlüssel jedes Knotens ist grösser oder gleich den Schlüsseln seiner Kinder.

ExtractMax: 1.) Wurzel entfernen und letztes Blatt zur Wurzel tauschen.

2.) Wurzel versickern lassen, d.h. so lange mit dem grössten Kind tauschen bis die Heap-Bedingung wieder erfüllt ist.

i) Extract the maximum of the following **max-heap** and do all necessary operations to restore the heap condition (it suffices to only draw the final tree).

2 min an Prüfung
1/60 Punkte



Einfügen: 1.) neuen Knoten an nächste freie Stelle

2.) solange mit Eltern tauschen bis die Heap-Bedingung erfüllt ist

Laufzeit $O(\log n)$

HEAPSORT$(A[1..n])$

---

1 $H \leftarrow$ Heapify(A) } $O(n)$

2 **for** $i \leftarrow n, n-1, \ldots, 1$ **do** } $O(n \log n)$

3 $\quad A[i] \leftarrow$ ExtractMax(H) $O(\log n)$

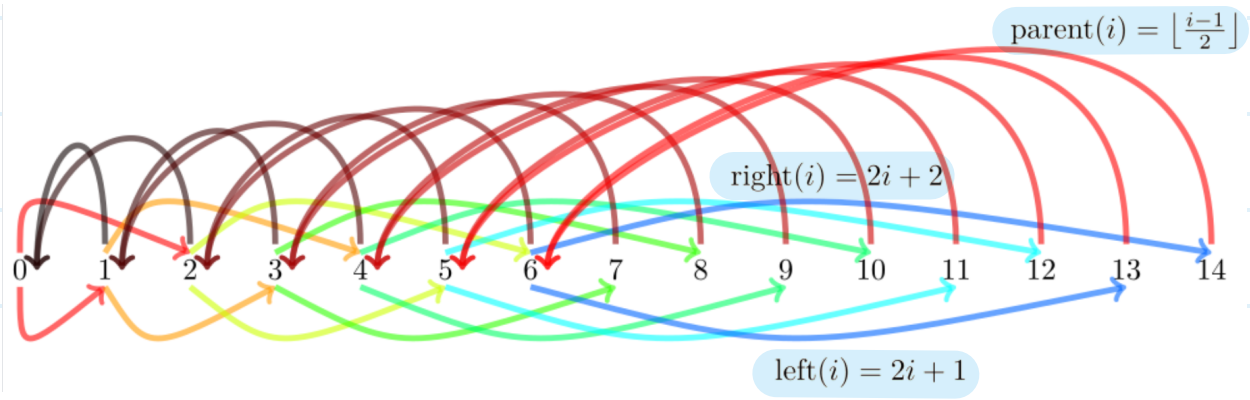
▷ *Wandle Array in Heap um.*

▷ *Entferne Elemente aus Heap*

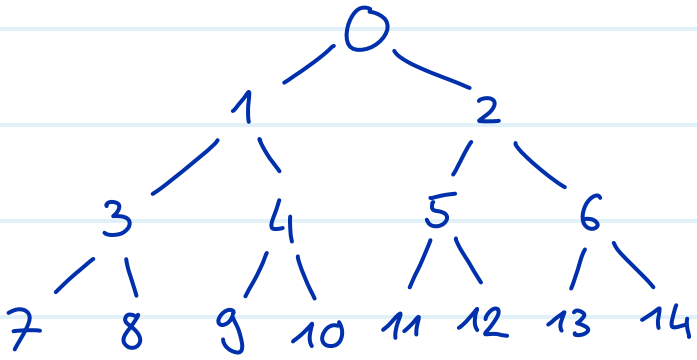
Heapify: 1.) Fülle Elemente des Arrays in einen Vollständigen Binärbaum

2) Versichere alle Knoten in absteigender Reihenfolge der Levels

Laufzeit $O(n)$



Achtung: In Vorlesung 2i und 2i+1



In Place Algorithmus

$A = [a_1 \quad a_2 \quad a_3 \quad .. \quad a_n]$

Heapify

tausche Wurzel und letztes Blatt

$[a_n \quad a_2 \quad a_3 \quad .. \quad a_1]$

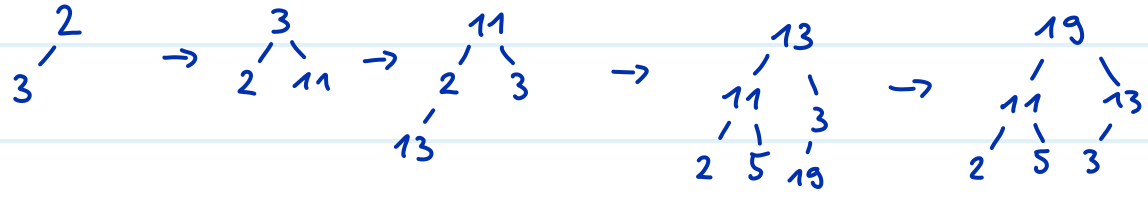
versickern

$[a_2 \quad a_n \quad a_3 \quad .. \quad | \quad a_1]$

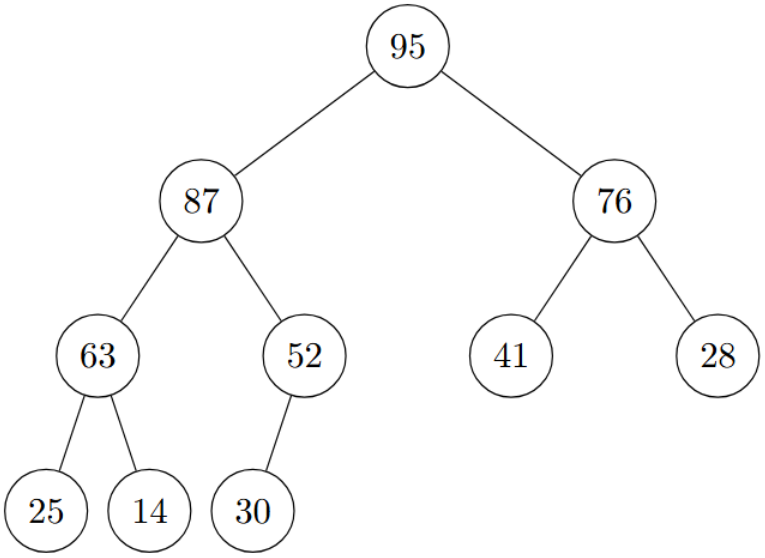
Heap — grössten Elemente sortiert

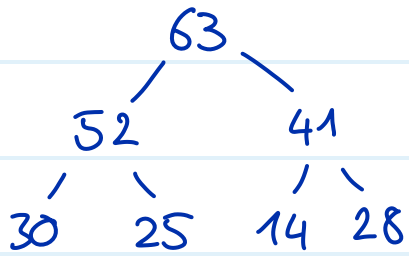
Nachteil von Heapsort: schlechte Lokalität

iii) Draw the **max-heap** obtained from inserting the keys 2, 3, 11, 13, 5, 19 in this order into an empty heap. It suffices to only draw the final heap. If there are several solutions, it is enough to draw only one of them.



Draw the Heap after ExtractMax is called three times





BUCKET-SORT($A[1..n]$) ▷ *alle Einträge in* $\{1,..,n\}$

```
1  B[1..n] ← [0, 0, ..., 0]
2  for j ← 1, 2, ..., n do
3      B[A[j]] ← B[A[j]] + 1        ▷ Zähle in B[i], wie oft i vorkommt
4  k ← 1
5  for i ← 1, 2, ..., n do
6      A[k, ..., k + B[i] − 1] ← [i, i, .., i]   ▷ Schreibe B[i]-mal den Wert i in A
7      k ← k + B[i]                 ▷ A ist bis Position k+B[i]−1 gefüllt.
```

- Beispiel für nicht vergleichsbasierten Sortieralgorithmus
- Laufzeit $\Theta(n)$
- Bsp. A = [3 4 4 6 5 1 2]

  B = [1 1 1 2 1 1]

  Ausgabe = [1 2 3 4 4 5 6]