ALGORITHMS & DATA STRUCTURE

October 25th

PLAN FOR TODAY

- Bonus Exercises
- Recap on Sorting

EXERCISE 4.2

(a) $T(1) = 1$, $T(n) = 4T\left(\frac{n}{2}\right) + 2n$, $T(n) = 3n^2 - 2n \; (*)$

Ind. über $k = \log_2 n$, $n = 2^k$

$\boxed{k=0}$ $T(2^0) = T(1) = 1$ mit $(*)$ $T(2^0) = 3\cdot(2^0)^2 - 2\cdot 2^0$
$= 3 - 2 = 1$

$\boxed{IH}$ $(*)$ gilt für ein (beliebiges) $k \in \mathbb{N}$

IS $k \to k+1$

$$T(2^{k+1}) = 4T(2^k) + 2\cdot 2^{k+1}$$

$$\overset{IH}{=} 4(3\cdot 2^{2k} - 2^{k+1}) + 2^{k+2}$$

$$= 3\cdot 2^{2k+2} - 4\cdot 2^{k+1} + 2\cdot 2^{k+1}$$

$$= 3(2^{k+1})^2 - 2\cdot 2^{k+1}$$

$\square$

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 $n \in \mathbb{N}$,*

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

Merge Sort

$T(n) = 2T(\frac{n}{2}) + c \cdot n$

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

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

$a = 2$

$b = 1$

$b = \log_2 a$

$O(n \log n)$

If the function T is increasing, then the condition $n = 2^k$ can be dropped. If (1) holds with "$=$", then we may replace O with Θ in the conclusion.

Algorithm 2 $g(n)$

```
if n > 1 then
    for i = 1, ..., 3n/2 do
        f()
    g(n/2)
else
    f()
    f()
    f()
    f()
    f()
```

$$T(1) = 5$$

$$T(n) = T\left(\frac{n}{2}\right) + \frac{3}{2} n$$

$$T(n) = 3n + 2 \quad (*)$$

$$k = \log_2 n, \quad n = 2^k$$

k = 0 $\quad T(1) = 5, \quad (*) \quad T(1) = 3 \cdot 1 + 2 = 5$

IH $\quad \ldots$

IS $\quad k \to k+1 \quad T(2^{k+1}) = T(2^k) + \frac{3}{2} \cdot 2^{k+1}$

$$\overset{IH}{=} 3 \cdot 2^k + 2 + 3 \cdot 2^k$$

$$= 3 \cdot 2^{k+1} + 2 \quad \square$$

Algorithm 3 $g(n)$

```
if n > 1 then
    for i = 1, ..., 4 do
        g(n/2)
    for i = 1, ..., n/2 do
        for j = 1, ..., 7n do
            f()
else
    for i = 1, ..., 4 do
        f()
```

$T(1) = 4$

$T(n) = 4\,T\left(\frac{n}{2}\right) + \frac{7}{2} n^2$

$T(n) = \frac{7}{2} n^2 \log n + 4n^2 \quad (*)$

$u = \log_2 n, \quad n = 2^u$

$u = 0$: $T(1) = 4, \quad (*)\; \frac{7}{2} \cdot 1 \log_2 1 + 4 \cdot 1^2 = 4$

IH ...

IS

$$T(2^{u+1}) = 4\,T(2^u) + \frac{7}{2} \cdot (2^{u+1})^2$$

$$\overset{IH}{=} 4 \cdot \left[\frac{7}{2} \cdot 2^{2u} \cdot K + 4 \cdot 2^{2u}\right] + \frac{7}{2} \cdot 2^{2u+2}$$

$$= \frac{7}{2} \cdot \left[2^{(u+1)}\right]^2 \cdot u + 4 \cdot 2^{2u+2} + \frac{7}{2} \cdot (2^{u+1})^2$$

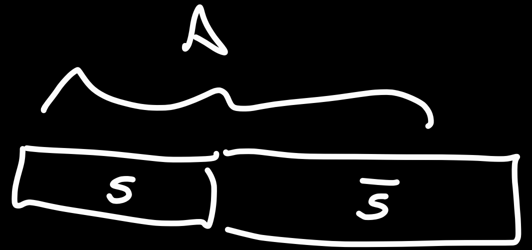
$$= \frac{7}{2} (2^{u+1})^2 (u+1) + 4 \cdot (2^{u+1})^2 \quad \square$$

EXERCISE 4.4

Algorithm 4 EXCHANGESORT(A)

for $1 \leq i \leq n$ **do**
    for $i + 1 \leq j \leq n$ **do**
        if $A[j] < A[i]$ **then**
            $T \leftarrow A[j]$
            $A[j] \leftarrow A[i]$
            $A[i] \leftarrow T$
return A

$\underline{INV(i)}$: die kleinste i Elemente von A sind in die ersten i Positionen, aufsteigend sortiert.

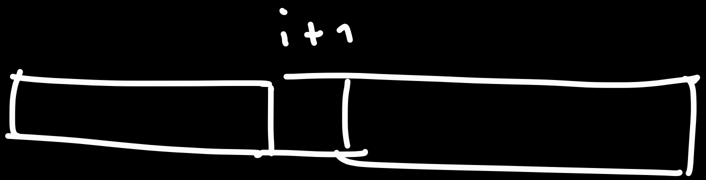


$\forall s \in S, s' \in \bar{S} \quad s \leq s'$

S sortiert ist

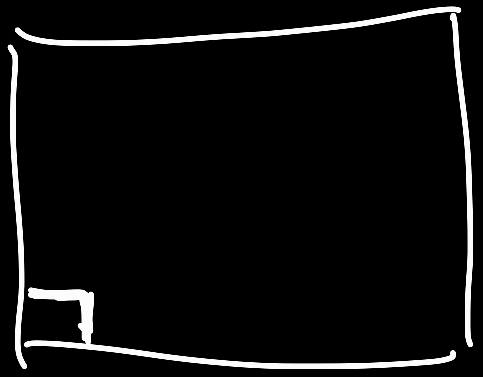
$INV(1)$

$INV(i) \Rightarrow INV(i+1)$



$INV(n) \Rightarrow A$ sortiert ist

EXERCISE 4.5



<u>Naive</u> $O(n \log n)$

ALGO(b, r, c)

if $r = 0$ ODER $c = n$ "nicht gefunden"

if $b = A[r][c]$

if $b > A[r][c]$ return ALGO$(b, r-1, c)$

if $b < A[r][c]$ return ALGO$(b, r, c+1)$

$$T(1) = 1$$

$$T(2n) = T(2n-1) + c$$

$$
\begin{aligned}
T(2n) &= T(2n-1) + c \\
&= T(2n-2) + 2\tilde{c} \\
&= \dots \\
&= T(2n-n) = n \cdot c'
\end{aligned}
$$

$$\Rightarrow T(n) = 2n \cdot c'' \in O(n)$$

SORTING, PART II

WHICH OF THE FOLLOWING ARE TRUE?

- There is no sorting algorithm with worst case complexity better than O(n log n) FALSCH!
- The lower bound for sorting in the general case is $\Omega(n \log n)$ WAHR
- There are comparison based sorting algorithms with complexity $\Theta(n \log n)$ WAHR, MergeSort, HeapSort
- The complexity of QuickSort is O(n log n) FALSCH

WHICH OF THE FOLLOWING ARE TRUE?

- There is no sorting algorithm with worst case complexity better than O(n log n)
- The lower bound for sorting in the general case is $\Omega(n \log n)$
- There are comparison based sorting algorithms with complexity $\Theta(n \log n)$
- The complexity of QuickSort is O(n log n)

LOWER BOUND $\Omega(n \log n)$

BUCKET SORT

RADIX SORT

$\Theta(n)$

- This holds for comparison based sorting, aka the general case.

- For some special cases, faster algorithms exist.

$A = A_1 \dots A_n \quad , \quad A_i \in \{0, 1\} \quad \forall i \in \{1 \dots n\}$

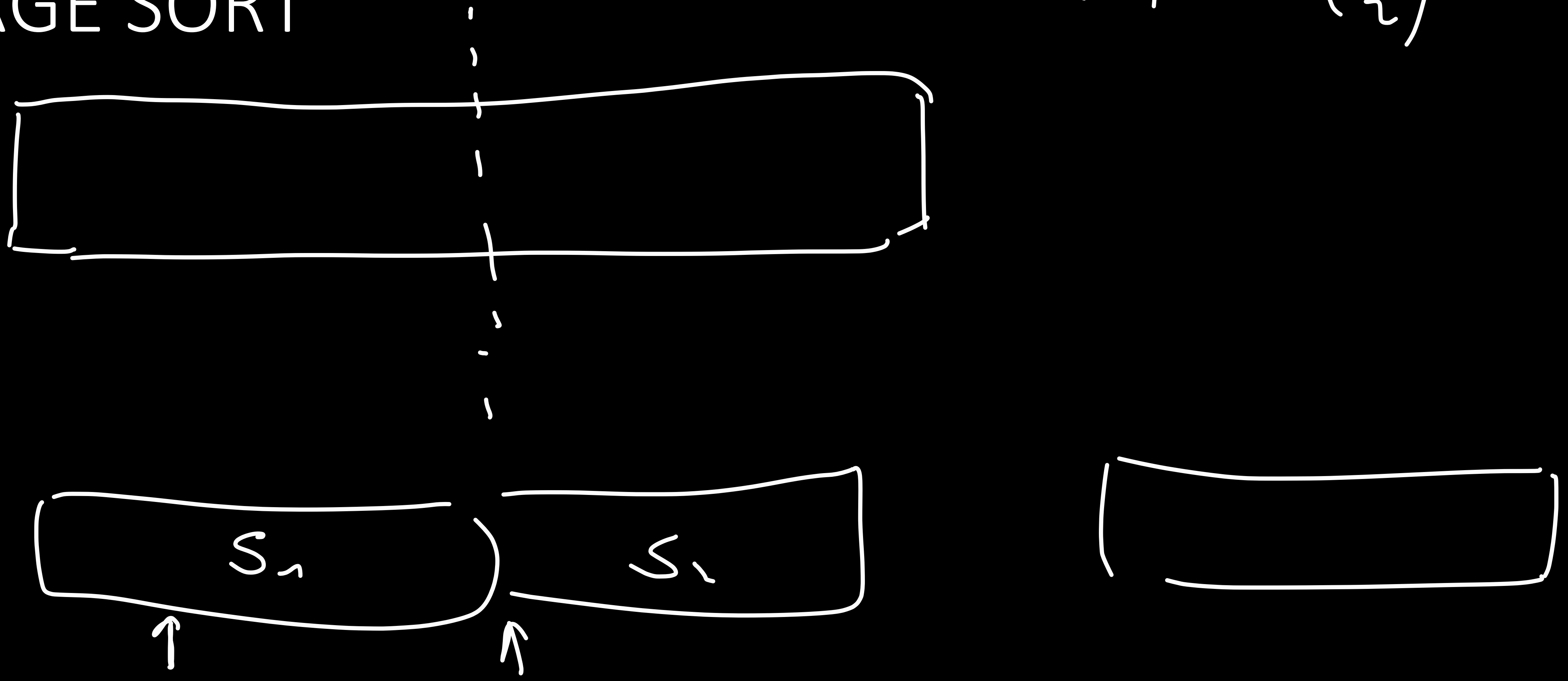
```
counter ← 0
for i = 1 ... n
    counter += A_i
```

COUNTING SORT

return $(n - \text{counter})$ 0s $\oplus$ (counter) 1s

MERGE SORT

$$T(n) = 2T\left(\frac{n}{2}\right) + cn$$



DISCLAIMER: ALGORITHMS VS DATA STRUCTURES (Array, Liste, Min/Max Heap)

Binary Search on Array

$T(1) = c$

$T(n) = T\left(\frac{n}{2}\right) + c$

$O(\log n)$

ALGO(n, h)

Base Case...

VERGLEICHE

u und $h(\frac{n}{2})$

....

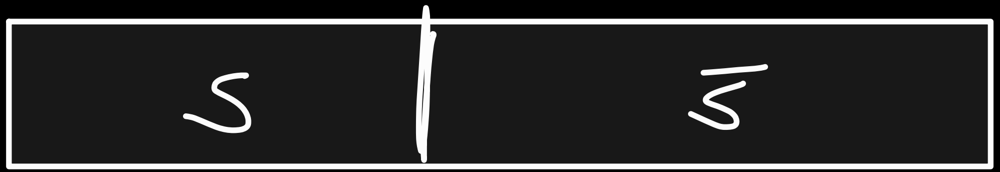
Binary Search on List

$T(1) = c$

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

$O(n)$

HEAP SORT



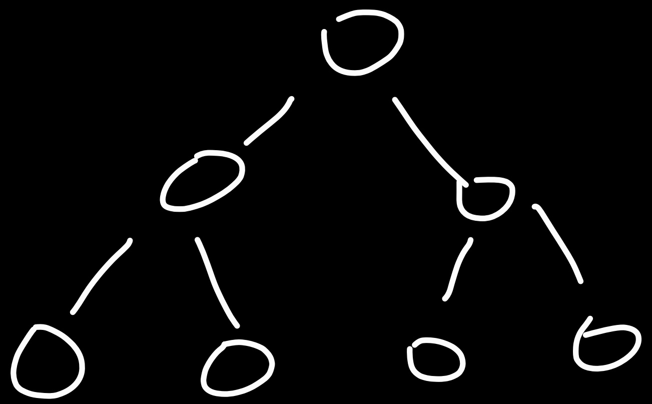
Für $i = 1 \dots n$

Finde i-te kleinste Element $\Rightarrow$ $O(\log n)$ Heap Sort

$O(n)$ selection sort

vertausch es, so dass der i-te kleinste Element in die richtige Position ist

---



- Logaritmische Höhe
- Gegeben einen Knoten, alle "descendants" sind $\geq$.

Finde den Minimum $O(1)$

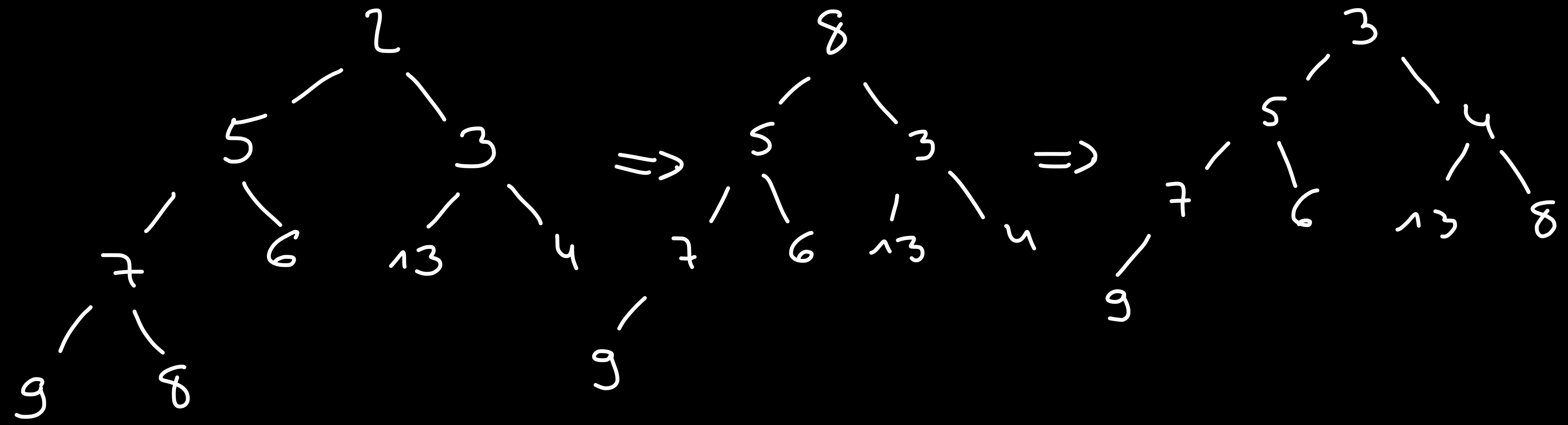
Entferne das Minimum $O(\log n)$

Gegeben n Elemente, baue ein Heap $O(n \log n)$

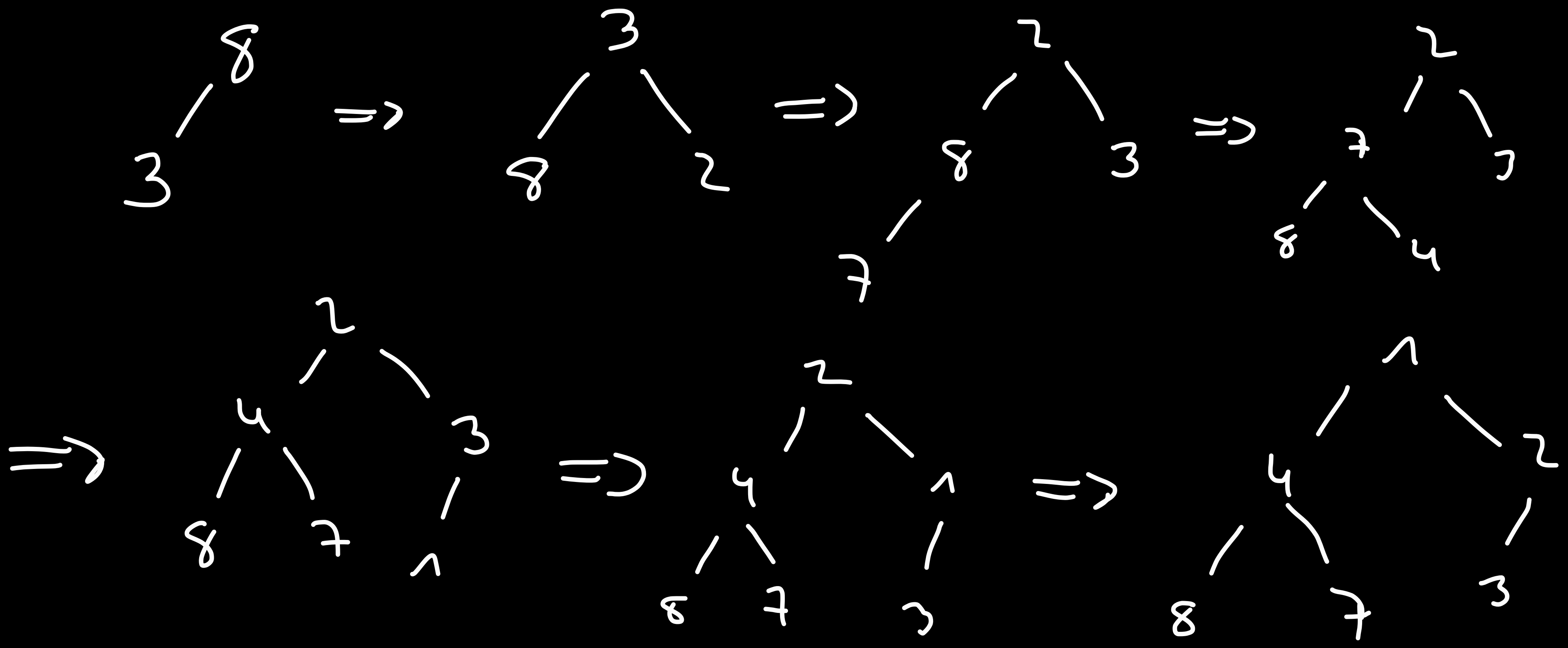
Das folgende Array enthält die Elemente eines in üblicher Form gespeicherten Min-Heaps. Entfernen Sie das minimale Element aus dem Heap, stellen Sie die Heap-Bedingung wieder her, und geben Sie das resultierende Array an.

2	5	3	7	6	13	4	9	8
1	2	3	4	5	6	7	8	9

3	5	4	7	6	13	8	9
1	2	3	4	5	6	7	8



Min-Heap: Draw the Min-Heap that is obtained when inserting into an empty heap the keys 8, 3, 2, 7, 4, 1 in this order.



QUICK SORT

(+) In place

(−) $O(n^2)$ in worst-case

↳ (+) "unwahrscheinlich"

a) Gegeben sei das folgende Array, das nach einem Quicksort-Aufteilungsschritt entstanden ist. Welcher Schlüssel wurde als Pivotelement verwendet? Markieren Sie *alle* möglichen Kandidaten.

✓	✗	✗	✓	✗	✗	✓	✗	✗
1	4	3	5	7	6	8	10	9
1	2	3	4	5	6	7	8	9

Execute the pivot step of the sorting algorithm Quicksort on the given array (in-situ, i.e., without an auxiliary array). Use the rightmost element of the array as the pivot element.

25	12	83	2	58	68	19	34	47	99	37	56	41

25	12	37	2	34	19	41	68	58	47	99	83	56

TRADE-OFF SEARCHING/ SORTING

LINEAR SEARCH

BINARY SEARCH

$O(n \cdot k)$

$O(n \log n + k \cdot \log n)$

$k = \Theta(\log n)$