Theory Recap Week 5

Algorithms and Data Structures

Georg Hasebe — 23.10.2023

What we have learned so far

Big O Notation

- Differences between Θ, O and Ω
- Limit definitions, Set definitions
- L'Hôspital Rule, Logarithm rules, Limit rules, $e^{\ln x}$ trick…
- Interpretation of notation: "grows slower/faster/equal"
- Runtime analysis

Induction

- (Starke) Induktion
- Base, Hypotheses, Step…
- Not every exercise has to be solved using induction…

Prove that $n^2 + n$ is even for all $n \in \mathbb{N}$.

Solution

Notice that $n^2 + n = n(n+1)$. Either n is even or if n is odd, $n + 1$ is even. Any product of integers that contains an even integer is even.

Another example:

(c) For any $k \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$, prove the following two inequalities

$$\sum_{i=1}^{2^k} \frac{1}{i} \leq k + 1$$

and

$$\sum_{i=1}^{2^k} \frac{1}{i} \geq \frac{k+1}{2}.$$

Design Algorithms

- Runtime, Correctness, Description/Pseudocode
- Formal proof of correctness by induction using invariants
- Naive algorithms, smart algorithms, very smart algorithms
- Lower bound of runtimes?

Maximum Subarray Sum

- Naive, Divide and Conquer, Inductive algorithm
- Lower bound for time complexity

Search Algorithms

- Binary Search
- Linear Search
- Lower bound for searching

Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort

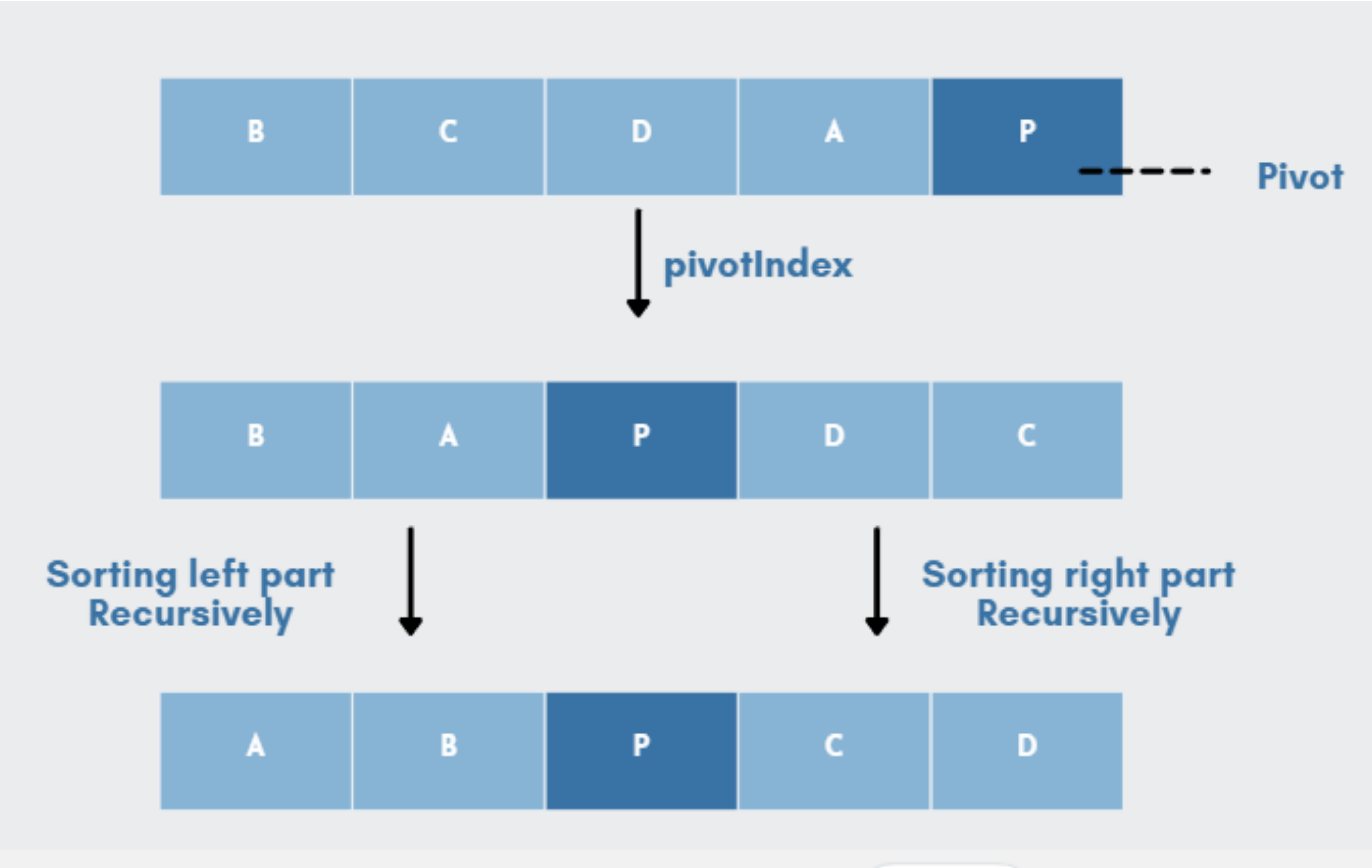
Today's topics

- Sort algorithms
- Specifically HeapSort and QuickSort
- Lower bound for sorting
- Data Structures
- Specifically Array, List, Linked List

	Comparisons	Swaps	Space Complexity
Bubble Sort	$O(n^2)$	$O(n^2)$	$O(1)$
Selection Sort	$O(n^2)$	$O(n)$	$O(1)$
Insertion Sort	$O(n \log n)$	$O(n^2)$	$O(1)$
Merge Sort	$O(n \log n)$	$O(n \log n)$	$O(n)$

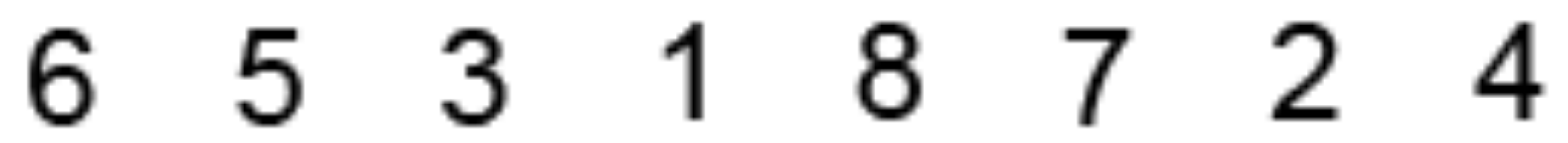
QuickSort

- choose pivot index p
- find correct position for p in the array
- put all elements $\leq p$ to the left of p and the others ($>$) to the right
- sort recursively left and right part (with respect to p)



https://towardsdatascience.com/an-overview-of-quicksort-algorithm-b9144e314a72

QuickSort GIF



https://en.wikipedia.org/wiki/Quicksort#/media/File:Quicksort-example.gif

QuickSort Pseudocode

```
QUICKSORT(A, l, r)

if l < r then                              ▷ Array enthält mehr als einen Schlüssel
    k = PARTITION(A, l, r)                 ▷ Führe Aufteilung durch
    QUICKSORT(A, l, k − 1)                 ▷ Sortiere linke Teilfolge rekursiv
    QUICKSORT(A, k + 1, r)                 ▷ Sortiere rechte Teilfolge rekursiv
```

```
PARTITION(A, l, r)

i = l
j = r − 1
p = A[r]
repeat
    while i < r and A[i] < p do i = i + 1
    while j > l and A[j] > p do j = j − 1
    if i < j then Vertausche A[i] und A[j]
until i ≥ j
Tausche A[i] und A[r]
return i
```

QuickSort Pseudocode

5 **while** $i < r$ **and** $A[i] < p$ **do** $i = i + 1$
6 **while** $j > l$ **and** $A[j] > p$ **do** $j = j - 1$

after these loops we have $A[j] < p < A[i]$, thus we swap.

5 **while** $i < r$ **and** $A[i] < p$ **do** $i = i + 1$
6 **while** $j > l$ **and** $A[j] > p$ **do** $j = j - 1$
7 **if** $i < j$ **then** Vertausche $A[i]$ und $A[j]$

if we have $i \geq j$ after lines 5-7, we have $A[i] > p$ (nothing swapped in line 7)

5 **while** $i < r$ **and** $A[i] < p$ **do** $i = i + 1$
6 **while** $j > l$ **and** $A[j] > p$ **do** $j = j - 1$
7 **if** $i < j$ **then** Vertausche $A[i]$ und $A[j]$
8 **until** $i \geq j$
9 Tausche $A[i]$ und $A[r]$

QuickSort Java Implementation

```java
public static void quickSort(int[] A, int l, int r) {
    if (l < r) {
        int k = partition(A, l, r);

        quickSort(A, l, k - 1);
        quickSort(A, k + 1, r);
    }
}
```

Find it on my Github:

https://github.com/ghasebe/java-algorithms

```java
public static int partition(int[] A, int l, int r) {
    int p = A[r];
    int i = l - 1;

    for (int j = l; j <= r - 1; ++j) {
        if (A[j] < p) {
            i++;
            if (i != j) {
                int tmp = A[i];
                A[i] = A[j];
                A[j] = tmp;
            }
        }
    }

    i++;
    int tmp = A[i];
    A[i] = A[r];
    A[r] = tmp;

    return i;
}
```

QuickSort Runtime

The partition function compares $O(r-l)$ keys. Runtime is therefore mainly dependant on the pivot element p.

Best case:

$$T(n) = 2T(n/2) + c \cdot n, T(1) = 0 \qquad \in O(n \log n)$$

Worst case:

$$T'(n) = T'(n-1) + c \cdot n, T(1) = 0, \qquad \in O(n^2)$$

HeapSort Motivation

- Can we improve SelectionSort?
- What if the maximum was readily available each time we need it?
- Can we achieve this using another data structure?

What is a Heap?

MAX-HEAP

Heaps Ein Array $A = (A[1], \ldots, A[n])$ mit n Schlüsseln heisst *Max-Heap*, wenn alle Positionen $k \in \{1, \ldots, n\}$ die *Heap-Eigenschaft*

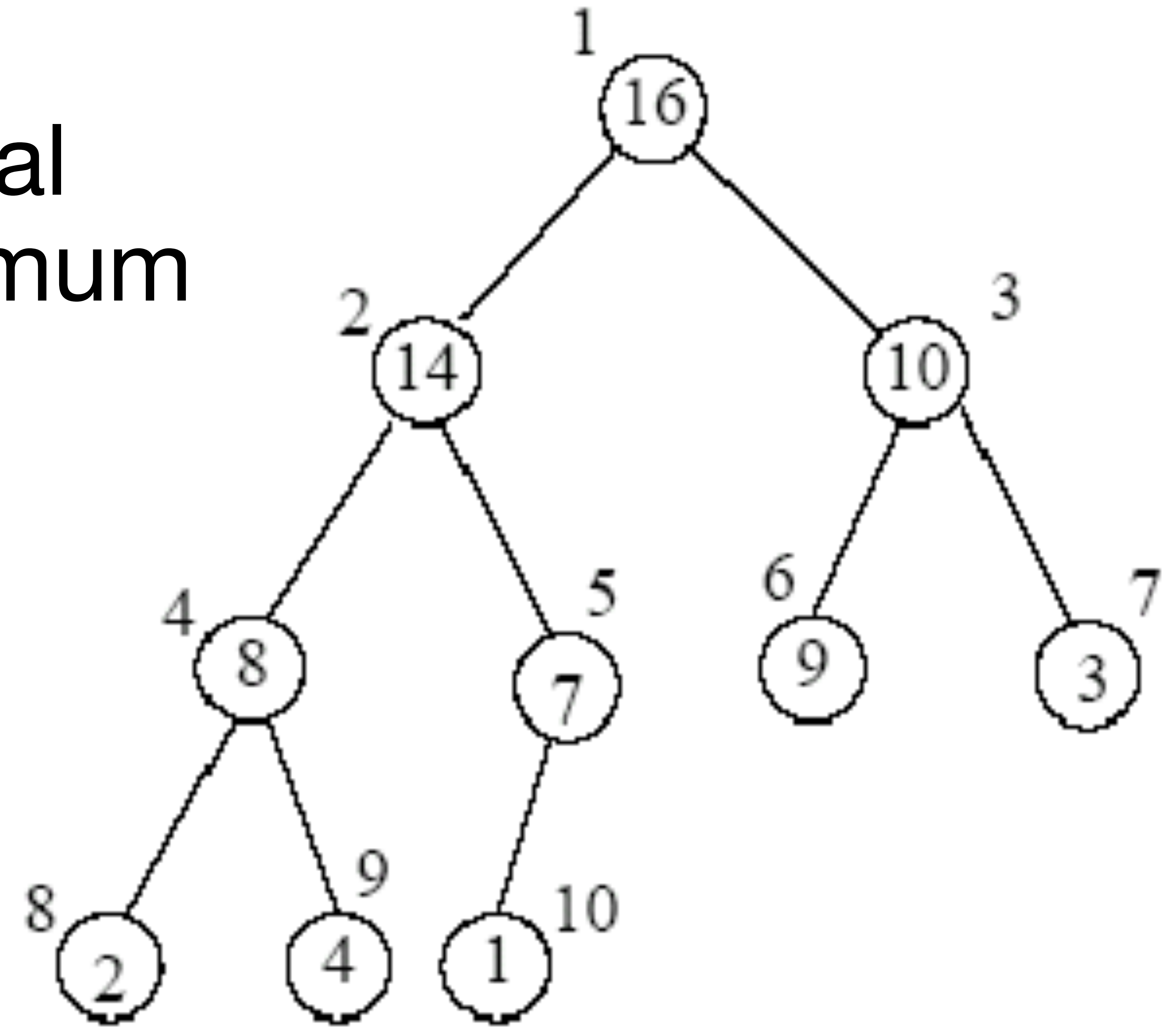
HEAP-EIGENSCHAFT

$$\big((2k \leq n) \Rightarrow (A[k] \geq A[2k])\big) \text{ und } \big((2k+1 \leq n) \Rightarrow (A[k] \geq A[2k+1])\big) \tag{38}$$

🤨

What is a Heap?

child is at most smaller equal
than parent => A[1] is maximum



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

http://www.cse.hut.fi/en/research/SVG/TRAKLA2/tutorials/heap_tutorial/taulukkona.html

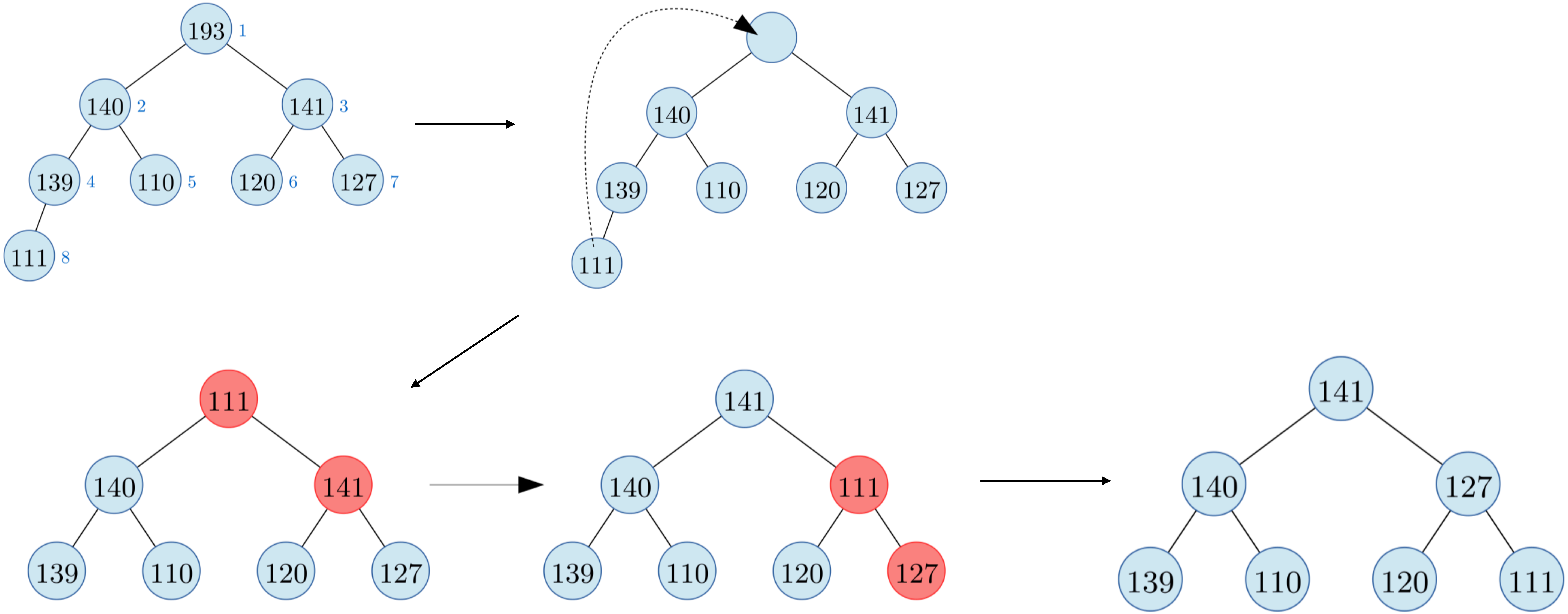
HeapSort

- generate Heap from array
- take max and put it into the correct position (swap)
- repair heap

HeapSort Visualisation

https://www.cs.usfca.edu/~galles/visualization/HeapSort.html

HeapSort



193 1
140 2
141 3
139 4
110 5
120 6
127 7
111 8
140
141
139
110
120
127
111
111
140
141
139
110
120
127
141
140
111
139
110
120
127
141
140
127
139
110
120
111

Pseudocode

```
HEAPSORT(A)

for i ← ⌊n/2⌋ downto 1 do
    RESTORE-HEAP-CONDITION(A, i, n)
for m ← n downto 2 do
    Vertausche A[1] und A[m]
    RESTORE-HEAP-CONDITION(A, 1, m − 1)
```

```
RESTORE-HEAP-CONDITION(A, i, m)

while 2 · i ≤ m do                          ▷ A[i] hat linken Nachfolger
    j ← 2 · i                               ▷ A[j] ist linker Nachfolger
    if j + 1 ≤ m then                       ▷ A[i] hat rechten Nachfolger
        if A[j] < A[j + 1] then j ← j + 1   ▷ A[j] ist grösserer Nachfolger
    if A[i] ≥ A[j] then STOP                ▷ Heap-Bedingung erfüllt
    Vertausche A[i] und A[j]                ▷ Reparatur
    i ← j                                   ▷ Weiter mit Nachfolger
```

Pseudocode + Runtime Analysis

```
HEAPSORT(A)

for i ← ⌊n/2⌋ downto 1 do
      RESTORE-HEAP-CONDITION(A, i, n)
for m ← n downto 2 do
      Vertausche A[1] und A[m]
      RESTORE-HEAP-CONDITION(A, 1, m − 1)
```

Lines 1–2: $O(n \log n)$ we can even reduce it further to $O(n)$

Lines 3–5: $O(n \log n)$

```
RESTORE-HEAP-CONDITION(A, i, m)

while 2 · i ≤ m do                         ▷ A[i] hat linken Nachfolger
      j ← 2 · i                            ▷ A[j] ist linker Nachfolger
      if j + 1 ≤ m then                    ▷ A[i] hat rechten Nachfolger
            if A[j] < A[j + 1] then j ← j + 1   ▷ A[j] ist grösserer Nachfolger
      if A[i] ≥ A[j] then STOP             ▷ Heap-Bedingung erfüllt
      Vertausche A[i] und A[j]             ▷ Reparatur
      i ← j                                ▷ Weiter mit Nachfolger
```

HeapSort Java Implementation

```java
public static void heapSort(int[] A, int n) {
    for (int i = n / 2 - 1; i >= 0; --i) {
        heapify(A, i, n);
    }

    for (int i = n - 1; i >= 0; --i) {
        int tmp = A[i];
        A[i] = A[0];
        A[0] = tmp;

        heapify(A, 0, i);
    }
}
```

```java
public static void heapify(int[] A, int i, int n) {
    while (2 * i + 1 < n) {
        int j = 2 * i + 1;

        if (j + 1 < n && A[j + 1] > A[j]) {
            j++;
        }

        if (A[i] > A[j]) {
            return;
        }

        int tmp = A[i];
        A[i] = A[j];
        A[j] = tmp;

        i = j;
    }
}
```

Find it on my Github:

https://github.com/ghasebe/java-algorithms