ALGORITHMEN & DATENSTRUKTUREN - ÜBUNGSSTUNDE 5

4.5 a)

Algorithm 2

```
i ← 1
while i ≤ n do
    j ← i
    while 2^j ≤ n do
        f()
        j ← j + 1
    i ← i + 1
```

$$\sum_{i=1}^{n} \sum_{j=i}^{\lfloor \log_2(n) \rfloor} 1 = \sum_{i=1}^{\lfloor \log_2(n) \rfloor} \sum_{j=i}^{\lfloor \log_2(n) \rfloor} 1 = \sum_{i=1}^{\lfloor \log_2(n) \rfloor} (\lfloor \log_2(n) \rfloor - i + 1) = (\lfloor \log_2(n) \rfloor \cdot (\lfloor \log_2(n) \rfloor + 1))/2$$

Exercise 4.4 *Searching for the summit* **(1 point)**.

Suppose we are given an array $A[1 \ldots n]$ with n **unique** integers that satisfies the following property. There exists an integer $k \in [1, n]$, called the *summit index*, such that $A[1 \ldots k]$ is a strictly increasing array and $A[k \ldots n]$ is a strictly decreasing array. We say an array is **valid** is if satisfies the above properties.

(a) Provide an algorithm that find this k with worst-case running time $O(\log n)$. Give the pseudocode and give an argument why its worst-case running time is $O(\log n)$.

Note: Be careful about edge-cases! It could happen that $k = 1$ or $k = n$, and you don't want to peek outside of array bounds without taking due care.

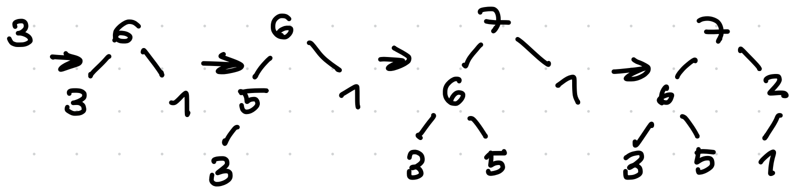
Algorithm 2 Find the summit

```
function FINDSUMMITINDEX(T, i, j)
    m ← ⌊(i + j)/2⌋
    if j = i then
        return i
    if T[m + 1] < T[m] then                     ▷ m is right of the summit (or is the summit)
        return FINDSUMMITINDEX(T, i, m)         ▷ keep searching in the left half
    else                                        ▷ m is strictly left of the summit
        return FINDSUMMITINDEX(T, m + 1, j)     ▷ keep searching in the right half
Input: Valid array T of length n with unique elements
Output: FINDSUMMITINDEX(T, 1, n)
```

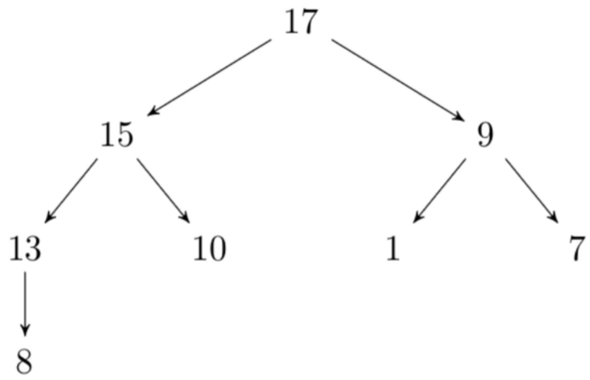
(b) Given an integer x, provide an algorithm with running time $O(\log n)$ that checks if x appears in the (valid) array or not. Describe the algorithm either in words or pseudocode and argue about its worst-case running time.

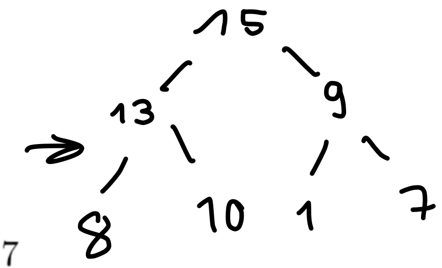
We first search for k and then do binary search on the ascending side. We do a modified version (mirrored) of binary search on the descending side.

i) Draw a Max-Heap that contains the keys $3, 6, 1, 5, 7, 2$.



ii) Draw the Max-Heap obtained from the following Max-Heap by performing the operation `DELETE−MAX` once.





Consider the following recursive function that takes as an input a natural number m that is a power of two (that is, $m = 2^k$ for some nonnegative integer k).

Algorithm 3 $g(m)$

---

if $m > 1$ **then**
  $g(m/2)$
  $g(m/2)$
  $g(m/2)$
  $g(m/2)$
  for $i = 1, \ldots, m^2$ **do**
    $f()$
else
  $f()$

Let $T(m)$ be the number of calls of the function f in $g(m)$.

i) Give a recursive formula for $T(m)$.

$$T(m) = \begin{cases} 4T(\frac{m}{2}) + m^2, & m > 1 \\ 1, & \text{otherwise} \end{cases}$$

ii) Write $T(m)$ in $\mathcal{O}$-notation in terms of m (as tight and simplified as possible).

MT (allenfalls a, b, c angeben) $T(m) \leq O(n^2 \log(n))$