1 Heaps: Homework 1

1.1 Exercise 4

Problem
Show that, with the array representation, the leaves of a binary heap containing n nodes are indexed by $\lfloor \frac{n}{2} \rfloor + 1, \lfloor \frac{n}{2} \rfloor + 2, ..., n$.

Solution
Since the elements of the heap are presented sequentially, starting from the root, then proceeding by the first level of children, followed by the second and so on, it is sufficient to prove that $\lfloor \frac{n}{2} \rfloor + 1$ is the first leaf and n the last leaf for the conclusion to follow.

Let's focus on the second task: by starting our indexing from 1, then, since we suppose to have n nodes, the last node must be a leaf. If it wasn't, then it would mean that it would be possible for the node n to have a child having index $2n$, absurd.

With the same reasoning, the node in $\lfloor \frac{n}{2} \rfloor$ has a child in $2 \cdot \lfloor \frac{n}{2} \rfloor = n$, while the node in $\lfloor \frac{n}{2} \rfloor + 1$ cannot, since its left child would have index $2 \cdot \left(\lfloor \frac{n}{2} \rfloor + 1 \right) = n + 2$, absurd.

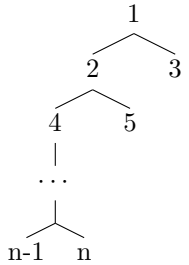
Therefore we prove that $\lfloor \frac{n}{2} \rfloor + 1$ is the first leaf.

Q.E.D

1.2 Exercise 5

Let's make an example of the worst case scenario on a min heap, with nodes having values from 1 to n, and suppose we remove the minimum. It has already been shown that the height of the heap (i.e. the number of nodes in a path from the root to a leaf) is equivalent to $O(log\ n)$.

Then consider the following graph



Once we remove 1, n goes to the top: heapify will have to push it down to the last level by recursively being applied to every left subtree starting from the root and ending on the leaf on the last level, since n is greater than every other element of the tree, hence the complexity of the algorithm in the worst case is equivalent to the height of the heap, $O(log\ n)$, Q.E.D.

1.3 Exercise 6

Let's prove this by induction.

Base case: Let's consider the root. As proven before, the root of the tree has $h = log_2 n$. Then $\lceil n/2^{h+1} \rceil = \lceil n/2^{log_2 n+1} \rceil = \lceil n/(2n) \rceil = \lceil 1/2 \rceil = 1$

Induction step: Let's suppose our assumption is true for the i_{th} level and prove it for the $(i-1)_{th}$ level. At the i_{th} level we will then have $\lceil n/2^{i+1} \rceil$. At the i_{th} level we will have a maximum of $2 \cdot \lceil n/2^{i+1} \rceil$ children (the equality is not necessarily satisfied only if the $(i+1)_{th}$ level is the last level, $h = 0$). This is exactly the maximum number of nodes on the $(i+1)_{th}$ level, i.e. $2 \cdot \lceil n/2^{i+1} \rceil = \lceil n/2^{(i-1)+1} \rceil$, Q.E.D

2 Homework 2

2.1 Exercise 2

Consider the next algorithm:

```
def Ex2 (A)
        D <- build (A)
        while not is_empty (D)
                extract_min (D)
        endwhile
enddef
```

where A is an array. Compute the time-complexity of the algorithm when:

1. build, is_empty $\in \Theta(1)$, extract_min $\in \Theta(|D|)$;
2. build $\in \Theta(|A|)$, is_empty $\in \Theta(1)$, extract_min $\in O(logn)$;

Solution

1. In the first case, build has complexity $\Theta(1)$ and it is outside a loop so we can asymptotically ignore it with respect to the second part of the algorithm. The while loop checks a condition in time $\Theta(1)$ each time it runs, and a single execution of extract_min takes time $\Theta(|D|)$ so this is where we have to focus. How many times does the while loop get performed? Based on a sensible interpretation of the effect of the routines, the data structure (probably the heap) loses an element (its minimum), at each iteration of the while loop, up until it's completely empty and the function ends. Therefore the complexity can be summarized as $\sum_{i=1}^{|D|} \Theta(i) \approx \Theta(|D|^2)$
2. (Here we will be supposing n = —D—) In the second scenario instead we need to take into consideration also the coroutine build. Our complexity

now will be

$$\Theta(|A| + \sum_{i=0}^{|D|} log(i)) =$$

$$\Theta(|A| + log(\prod_{i=0}^{|D|} i)) =$$

$$\Theta(|A| + log(|D|!)) \approx \Theta(|A| + n\ log(n))$$

where the last passage was obtained by the usage of Stirling Approximation