In Class Exercise

Problem 1:
We can build a heap out of an arbitrary array using successive calls to `Heapify-down`, starting at element $\lfloor \text{length}[H]/2 \rfloor$ and going down to 1. If each call to `Heapify-down` takes $O(\log n)$ time and we have $O(n/2)$ such calls, we can build a heap in $O(n \log n)$ time. Prove that this process is actually faster than $O(n \log n)$ (i.e., provide a *tighter* bound on the process's running time).

In the solutions, we will show that the bound for building a heap out of an array is actually $O(n)$.

Solution

Remark 1. The *height of a node* in a tree is the number of edges on the longest simple downward path from the node to a leaf, and the *height of a tree* is the height of its root.

The cost of `Heapify-down` depends on the height of the node on which it is called. Therefore, to calculate the bound for this problem, we need to know the following:

- Cost of `Heapify-down` for a node at height h—O(h) (at most h swaps)
- How many nodes they are at height h or at a level?
- How many total levels of nodes are there on which `Heapify-down` will be called?

Let's start with the last questions first: In order to answer this question, we need do know the height of the n-element heap. To be exact, the height of an n element heap is $\lfloor \log n \rfloor$, or in other words $O(\log n)$ (because it's nearly complete binary tree.)

Remark 2. The number of leaves in a complete binary tree is $\lceil \frac{n}{2} \rceil$

Theorem 1. *In an n element heap, there are $\lfloor \frac{n}{2^{h+1}} \rfloor$ nodes at height h.*

Proof. Proof by induction.

Base Case $h = 0$ The leaves of a binary tree are at height $h = 0$. So, the base case is true from Remark 2.

Inductive Hypothesis Suppose the claim is true for height $h-1$ i.e., there are $\lfloor \frac{n'}{2^{h+1}} \rfloor$ at height $h-1$ for an n' element heap.

Inductive Step Let N_h be the number of nodes at height h in an n-node tree T.

Consider T' formed by removing the leaves of T. T' has $n' = n - \lceil n/2 \rceil = \lfloor n/2 \rfloor$ nodes.

Nodes at height h in T are at height $h-1$ in T' (because T' is missing the bottom level of T).

Let N'_{h-1} denote the number of nodes at height $h-1$ in T'.

$N_h = N'_{h-1} = \lceil n'/2^h \rceil = \lceil \lfloor n/2 \rfloor / 2^h \rceil \leq \lceil (n/2)/2^h \rceil = \lceil n/2^{h+1} \rceil$.

□

Now, to calculate the bound on building a heap, we start at node $\lfloor \text{length}[H]/2 \rfloor$. This node will be at $h = 1$, there will be $\lfloor \frac{n}{2^2} \rfloor$, such nodes and the cost of `Heapify-down` will be 1. Similarly, at $h = 2$, there will be $\lfloor \frac{n}{2^3} \rfloor$, such nodes and the cost of `Heapify-down` will be 2. In general, we can write

$$\begin{aligned} & \textstyle\sum_{h=0}^{\lfloor \log n \rfloor} \lceil \frac{n}{2^{h+1}} \rceil O(h) \\ = \quad & O(n \textstyle\sum_{h=1}^{\lfloor \log n \rfloor} \frac{h}{2^h}) = O(n) \end{aligned}$$

Because,

$$\sum_{h=0}^{\infty} \frac{h}{2^h} = \frac{1/2}{(1-1/2)^2} = 2$$

(Adding the $h = 0$, does not change anything since the cost at that height is 0.)