Analysis of Build-Max-Heap (Kashif Murtaza PUCIT)

Courtesy: CLRS 3rd Edition, Chapter 6

Height of a Node in a Tree: Height of any node in a rooted tree is the longest *simple* path (in terms of number of edges) from that node to any leaf. There is a unique simple path between any two nodes in a tree. A simple path is a path in which no edge is revisited. All leaves have height zero. The root node has maximum height out of all the nodes of a tree. The height of root node (the maximum height out of all the nodes) is also called the height of the tree denoted by "h". If h = $\Theta(\lg n)$, then the tree is called *balanced*.

Complete Binary Tree: A binary tree is called *complete* if it is full (all parents have exactly two children) and all the leaves are at the same level (the last level).

Binary Heap Tree: A binary tree is called heap-tree (or tree-heap) if it is complete till the second last level, and the last level is filled from left to right. The last level may not be completely filled, but the order of filling the last level is strictly "left to right". Heap-tree is always balanced.

Binary Max-Heap: If every parent node in a binary heap-tree is larger or equal to it's children (Max-Heap property), the heap-tree is called Binary-Max-Heap. If you replace the world "larger" with "smaller" in the above definition, you get Binary-Min-Heap.

Theorem-1: There are exactly $ceil(\frac{n}{2^{k+1}})$ number nodes of height k in a complete binary tree of nodes n.

Theorem-2: There are at-most $ceil(\frac{n}{2^{k+1}})$ number nodes of height k in a binary Heap-tree of nodes n. (The only difference is the last level by the way).

Question-1: Let say a particular parent node "x" has height "k" in a heap-tree. If "x" violates the max-heap-property than how many swaps do we need in the worst case to adjust the max-heap-property for node "x"?

Answer-1: The algorithm we studied in the class builds the max-heap in a bottom up fashion. With respect to that algorithm, the total number of swaps needed, in the worst case, to adjust the max-heap property at node "x" is the height of the node "x", which is "k".

Question-2: How many nodes are there, at max, of height "k" in the Binary-Max-Heap of nodes n?

Answer-2: $\frac{n}{2^{k+1}}$ (theorem 2 [ignoring ceil function, asymptotically does not matter])

Question-3: For a binary heap-tree of size n, how many steps are required to adjust the max-heap-property in a bottom up fashion for all the parent nodes in the worst case? (What is the running time of Build-Max-Heap?)

Answer-3: Well, adjusting all the nodes of height "k" will take $k * \frac{n}{2^{k+1}}$ steps in the worst case (because there are $\frac{n}{2^{k+1}}$ nodes of height "k" and each may take "k" steps). Adjusting for all the heights, we have total running time (in the worst case) as follows:

$$T(n) = \sum_{k=0}^{\lg n} k * \frac{n}{2^{k+1}}$$

$$\Rightarrow T(n) = \frac{n}{2} \sum_{k=0}^{\lg n} k(\frac{1}{2})^k$$; Taking n/2 common

$$\Rightarrow T(n) \leq \frac{n}{2} \sum_{k=0}^{\infty} k(\frac{1}{2})^k$$; Even if we take infinite such terms, all terms are positive

$\Rightarrow T(n) = O(n)$; Because $\sum_{k=0}^{\infty} k(\frac{1}{2})^k = \Theta(1)$, it is derivative of decreasing geometric series with common ratio ½.