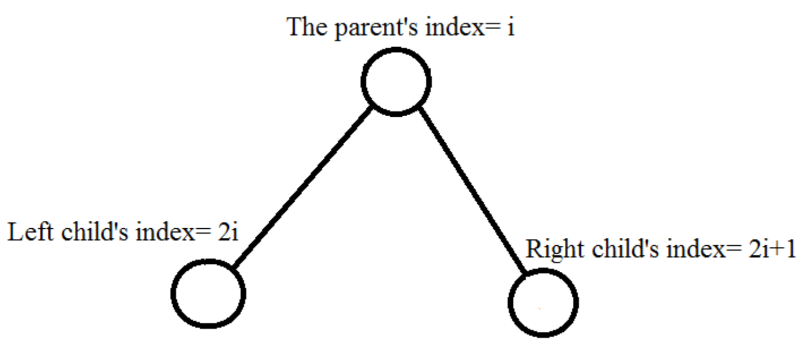
Binary heap

Is a complete binary tree, and:

1. All the nodes in all levels except possibly the last level have two children (It means that all the levels are fully filled, except possibly the last one)
2. We ALWAYS add nodes from left to right to the heap.
3. The height of the binary heap is $\lfloor \log n \rfloor$ (Why?)
4. Below figure:



Max-heap

The value of each internal node is **greater than or equal to** the values of its left and right child.

Insertion in max-heap O(logn)

1. Create a new leaf to the heap with the inserted value assigned to it. (NOTE: We ALWAYS add nodes from left to right.)
2. If the new node.value is **smaller** than its parent, **stop**.
3. If not, swap the element with its parent.
4. Go back to step 2.

Deleting the root O(logn)

1. Swap the **root** with **the last element** of the heap. *O(1)*
2. Delete the old root (now is located as the last element/leaf in the heap). *O(1)*
3. Apply Max-Heapify algorithm for the new root. *O(logn)*

Max-Heapify O(logn)

1. If the value of node **i** is greater than the value of its children, **stop**.
2. If not, find the maximum value between children.
3. Swap node **i** with the child with max value
4. Go back to step 1, and update **i** to the index of that child with max value (either **2i** or **2i+1**).

Max-Heapify (*A*, *i*)

max_index=i

left_index= 2i
right_index=2i + 1

if *left_index* ≤ *size(A)* **and** *A*[*left_index*] > A[*max_index*] // if *i has left child* and left child's value is greater than i value
    max_index= left_index
if *right_index* ≤ size(*A)* **and** *A*[*right_index*] > *A*[*max_index*] // if *i has right child* and left child's value is greater than i value
    max_index= right_index
if *max_index* ≠ *i*
    swap A[max_index] ⟷ A[i]
    Max-Heapify(A, max_index)

Building a max-heap

Two solutions:

1. Successive insertions. Running time: **O(nlogn) because** ➔ $\sum_{i=1}^{n} \log i = O(n \log n)$
2. Follow the below steps: Running time: **O(n)**
    a. Adding the elements into a binary tree
    b. Apply the max-heapify function starting from the last element (has index **n**)in the tree and move upward (until you reach index **1**)

Heapsort ϴ(nlogn)

1. Build a max-heap.
2. Remove the root (largest element of the max-heap) and insert it to the last position in the array.
3. Repeat step 2 until the max-heap becomes empty.