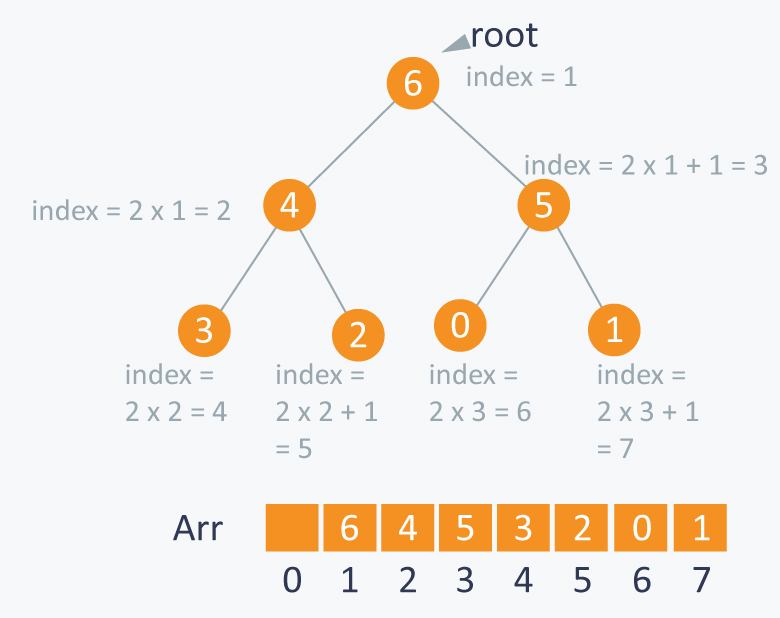
**Heap**

It’s a tree based datastructure. All nodes of the tree are in specific order. commonly-used heap type are Binary heap(only 2 child nodes).



if we are storing one element at index i in array , then its parent will be stored at index i/2. In above item 4 is stored in i(index 2) then its parent will be in i/2(index 1). So from root its left child can be accessed by i\*2 and right will be accessed by i\*2 + 1. Same continues for each parent to fetch its child

Why Heap?

Heaps allow for efficient insertion and deletion operations with an average time complexity of O(log n). This is because heaps maintain a balanced structure by rearranging the elements after each insertion or deletion.

When you need to frequently insert, delete, or query the minimum or maximum element from a dataset, a heap is an excellent choice.

Heaps are ideal for priority queuing applications where elements have different priorities and need to be processed accordingly.

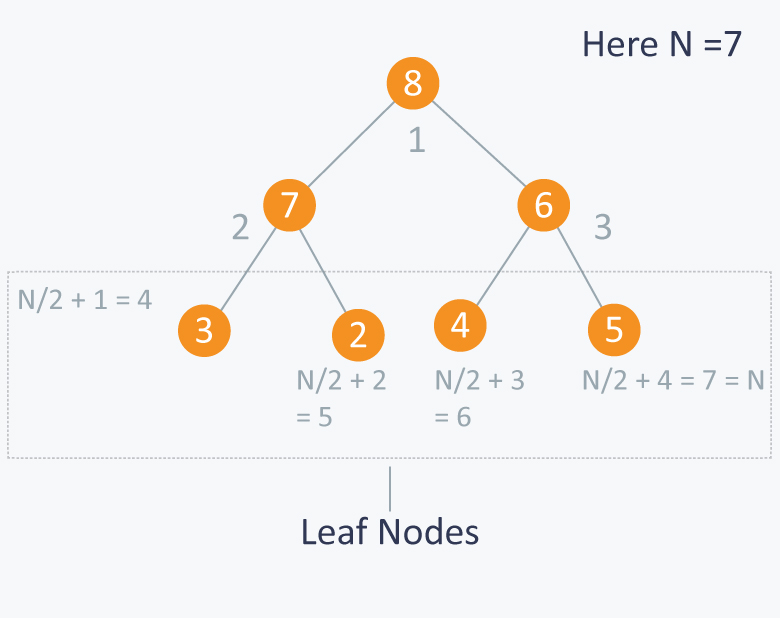
There 2 main implementation of present for tree data structure:

* **TreeSet**: Elements are sorted based on their natural order (i.e., alphabetical or numerical order).
* **PriorityQueue**: Elements are sorted based on their priority or value, which is typically a custom ordering defined by the application.

**Types of heap:**

1. Min heap: Parent node value will be less than child nodes value
2. Max heap: Parent node Value will be higher than child nodes value

**Max heap:**



Above is example for Max heap. In this we can also see an interesting property on leaf nodes. leaves indexed by **N/2+1, N/2+2, N/2+3, N/2+4,…**

Using this formula we can use bottom up approach to sort the tree

So In below image will start from N/2 = 3

Note: n/2 -1 need to be used if you start storing the element form index 0. Above example its from index 1

Sorting a value will be done by heapify method. But for heaping we will giving index which start from N/2 as per bottom up approach

What Heapify Method do?

Input param-> index i of parent

Step 1: finds the Childs by 2\*i (left) and 2\*i +1 (right)

Step 2: max = max(arr[I], arr[left], arr[right])

Step 3: if(arr[i] < max) swap(arr[i], max) and make recursive call with input as index of the max calculated above (either left child or right child). This make sure child present below will also be sorted since value are swapped

Step 3: else (arr[i] is already highest) close the loop

What sort method will do as per bottom up approach?

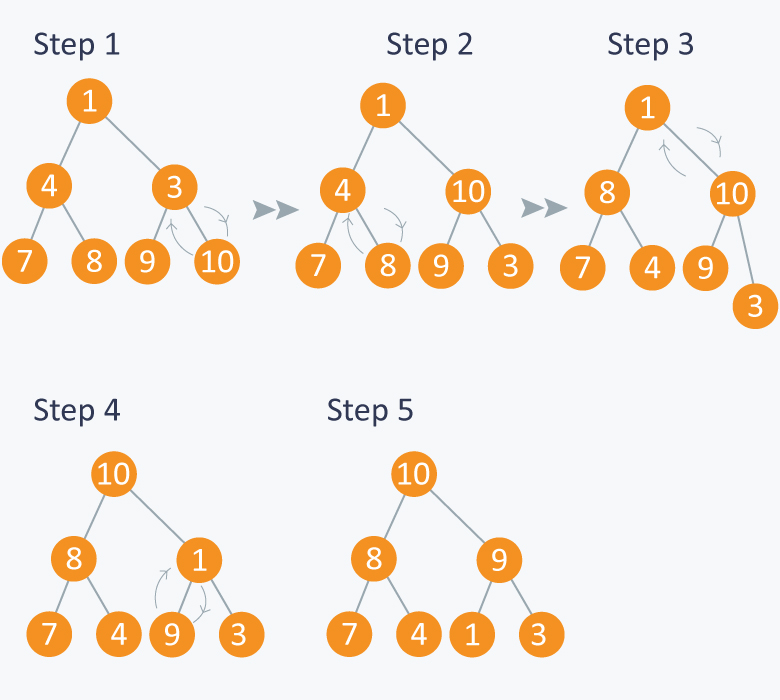
void sort(int Arr[ ])

{

for( int i = N/2 ; i >= 1 ; i--)

heapify (Arr, i);

}



**Decoding step by step**

1. For is declared as for(int i = N/2 ; i >= 1 ; i-- ) First iteration is N/2 = 7/2 = 3. Heaping method check for index 3. Find right child is larger than root and left, so swaps the value(3 with 10). Heapify method ends
2. Next iteration is 2. Heaping method check for index 2 Find right child is larger than root and left, so swaps the value(4 with 8). Heapify method ends
3. Next iteration is 1. Heaping method check for index 1 Find right child is larger than root and left, so swaps the value(1 with 10). Makes a recursive call with Childs index because child is not sorted properly
4. Heapify method check for index 3. Find left child is larger than parent and right, so swaps the value(1 with 9).
5. Gets sorted array

**Min heap:**

Same as Max heap but heapify method will swap in such way that parent have less value than child