**HEAP**

**Binary Heap**: It is a complete binary tree,i.e., all levels are completely filled except last level and last level is filled from left to right.

Left(i)=2i+1

Right(i)=2i+2

Parent(i)=floor(i-1)/2

With above properties we can represent it as array.

Last Internal node in complete binary tree =Parent of last node= parent of n-1 =(n-2)/2

Number of nodes at a height h in complete binary tree ceil of (n/2^h)

**Two types**:

* **Min Heap**: Highest priority is assigned lowest value. Every node has a value smaller than its descendants in this complete binary tree.
* **Max Heap**: Highest priority is assigned highest value.

**Applications**:

* Heap sort
* Implementation of priority queue.
* Used in Dijsktra shortest path algo, Prim’s minimum Spanning tree prob, Huffman coding prob

Heap Sort: After building heap, it is same as selection sort where we find the highest element and place it at last.

Use Max heap for ascending order and Min Heap for descending order.

Steps:

* Build heap : Start from last internal node till root, and heapify all subtrees

void buildHeap(int arr[], int n) {

int index = (n - 2) / 2;

for (int i = index; i >= 0; i--) {

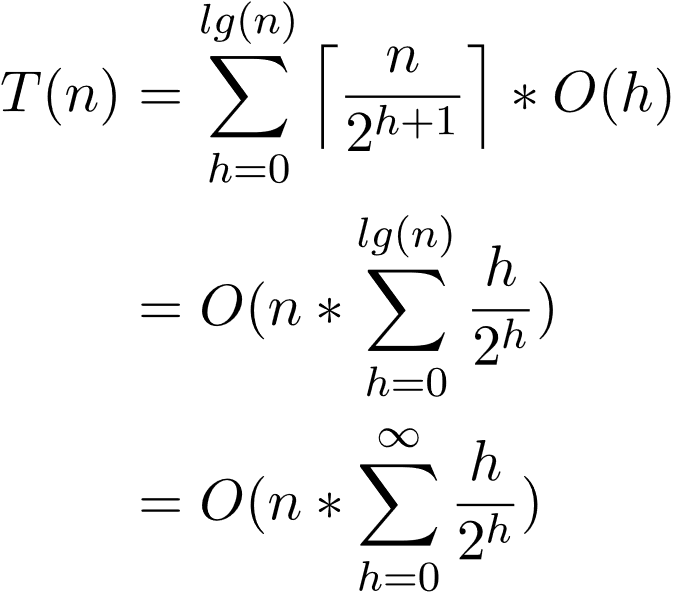
heapify(arr, n, i);

}

}

TimeComplexity of build heap :O(n) It might look nlogn but its not because time complexity of heapify is O(h ) and each subtree is of different height which is smaller than height of tree.

Explanation : Summation of number of nodes on each height \* height



= O(n){ ½ + ¼ +1/8+1/16 ….}

Infinite Geometric Progression =1/(1-x)=1/(1-1/2) =2

=O(2n)

=O(n)

* Swap root with last element and reduce heap size Call heapify on root.

In Java ,Priority queue is by default min heap , if we want max heap , we can change like :

new PriorityQueue(Collections.reverseOrder());

for custom order , pass comparator in constructor