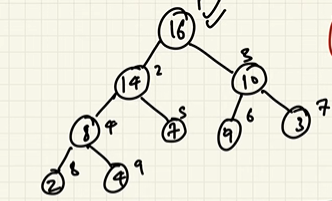
Heaps

Heap is a data structure which allows us to get greatest or smallest values in O(1) time . they store there element as a array which are interpreted as a tree .

Array representations : here we consider array index start from 1  
  
  
as Tree we interpret it like this



1. This tree is as complete binary tree means each level is completely full except for leaf nodes which are also fill from left to right
2. Every node is >= its children in term of max heap its reverse in min heap
3. One more thing the first element in array is always max and rest elements can be in any order our priority is always to keep min or max item as first element rest does not matter

Important formulas :

1. Root of the tree : i=0 or 1// depends on array indexing
2. Parent of node (i)= i/2 // every node parent is at index i/2 :

(example : node index i=2 then its parent will be at 2/2=1 similarly node index = 5 then its parent will be at 5/2= 2 in index we always do integer divisions)

1. Left Child : root index \*2// this is in 1 base indexing we add +1 in zero base indexing
2. Right Child : root index \*2+1// this is in 1 base indexing we add +1 in zero base indexing

(example : root index i=2 then its left child will be at 2\*2=4 similarly right child will be at 2\*2+1 = 5)

1. The height of the heap tree will be log(n) cause it’s a complete binary tree
2. Insertion in Min Heap

* Add at end: Place the new element at the end of the array/list (heap’s last position).
* Up-heap (heapify up):
  + Compare the new element with its parent.
  + If it is smaller than parent → swap.
  + Repeat until either root is reached or parent is smaller.
* Stop once the heap property (parent ≤ children) is restored.

Time Complexity: O(log n) (height of heap).

1. Removal (Extract Min)

* Take root: The minimum element is always at index 0. Store it (to return later).
* Replace root: Move the last element of the heap to the root position.
* Remove last: Delete the last element from the array/list.
* Down-heap (heapify down):
  + Compare the root with its smallest child.
  + If the root is larger → swap with the smaller child.
  + Repeat until leaf is reached or heap property is satisfied.
* Return removed root.

📌 Time Complexity: O(log n) (height of heap).