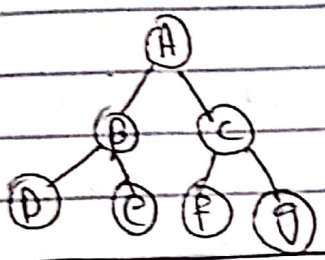
Heap

1- Array Representation of BT
2- Complet Binary Tree
3 - Heap
4 → Insert and Delete
5 → Heap Sort
6 → Heapify
7 → Priority Queue

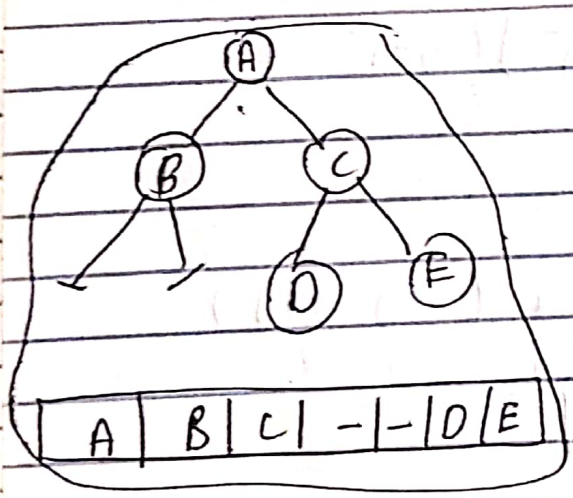
(1)

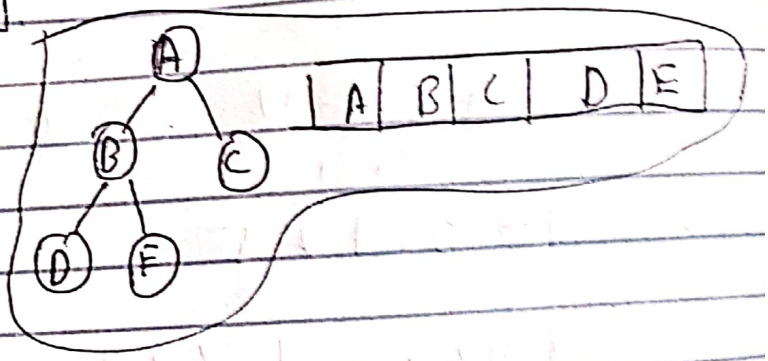


If Node at Index → i
Left child → 2 x i
Right child → 2i + 1
its parent is → $\lfloor \frac{i}{2} \rfloor$

T:

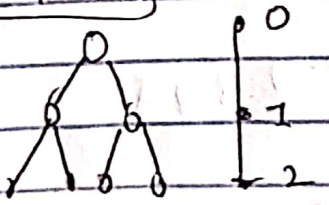
1	2	3	4	5	6	7
A	B	C	D	E	F	G





Full Binary Tree

Complet



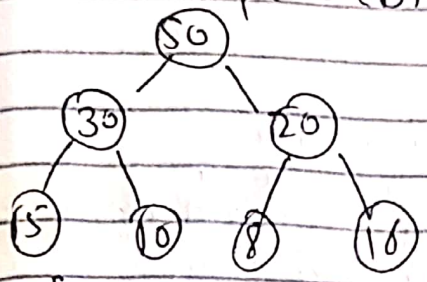
Full Binary Tree are h=2 Full fill with nodes when we add more node then height will increase....

Number of Total Node on Full Binary Tree = $2^{h+1} - 1$

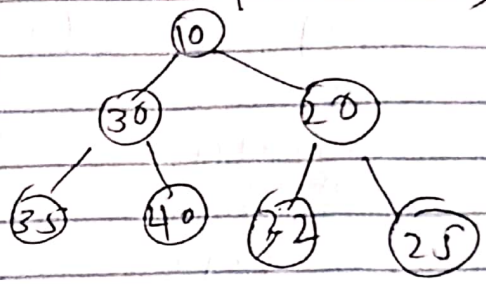
Note: In Array Representation
↳ If any element is missing inbetween the element Array then it's not Full Binary Tree

Complet Binary Tree is a full Binary Tree up to Height H minus -1

Max Heap (also CBT)



Min Heap (Also CBT)



H = [50, 30, 20 / 15, 10, 8, 16]

How Insertion

let we insert (60)

First

H = [50, 30, 20, 15, 10, 8, 16 | 60]

60 > 50 → left side

60 > 30 → left side

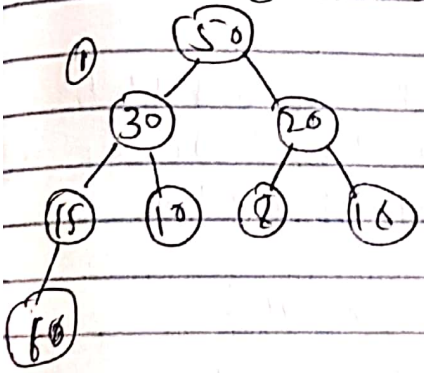
60 > 15 → left side

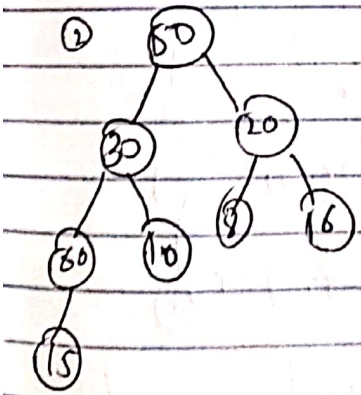
then going on operation with parent of 60

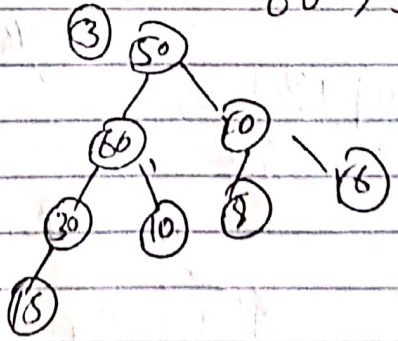
60 > 15 → go parent

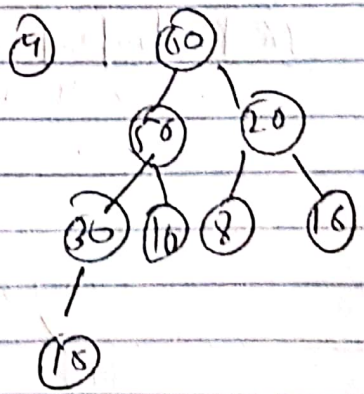
60 > 30 → go parent

60 > 50 → go parent









Now its time to Analysis

↳ Depend height

height = O(log n)

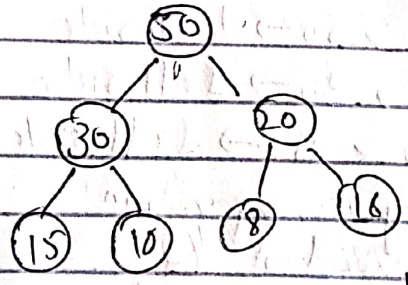
O(1) to O(log n)

↓

Minimum time if no swap occurs for example Insert 6

↳ Maximum time when maximum swap occurs there Number of swap equal to the height of Heap which is log(n) which take time log(n)

Deletion

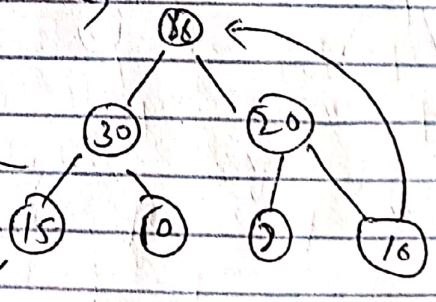


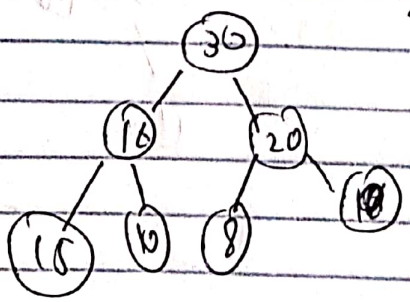
Delete 50

↳ Actual Always we deleted the root element because of main function

50	30	20	15	10	8	16
1	2	3	4	5	6	7

16	30	20	15	10	8	16
1	2	3	4	5	6	7





→ deleted root so still have BST property

Now its time to Analysis

Swap → depend to number node → Height

swap → ~~Upward~~ downward

In Insertion the swap was moving upward----

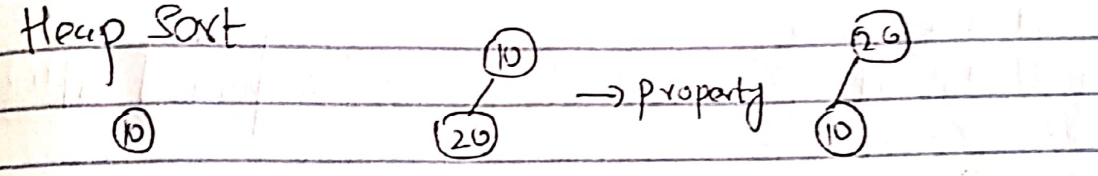
Note

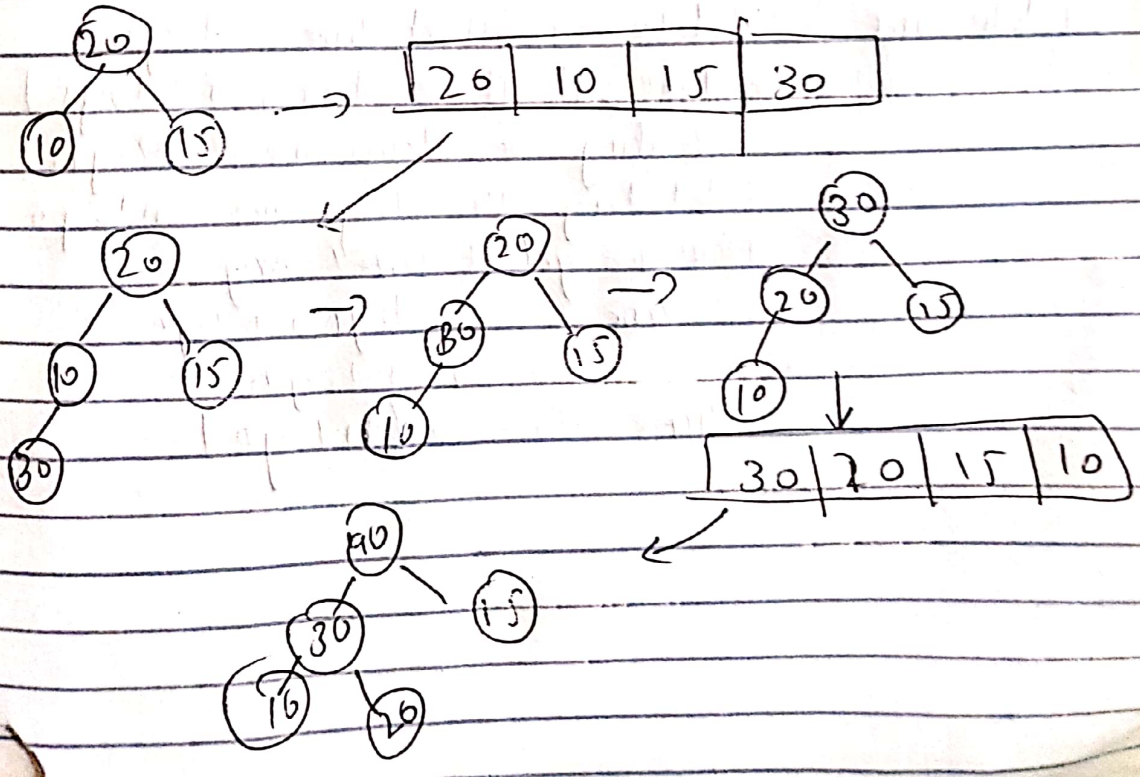
When you delete root then yeha set their positing but his Actual position save the deleted value when all the BST delete then you will get Sorted ~~Array~~ but its not ~~BST~~

"Idea of Heap sort"

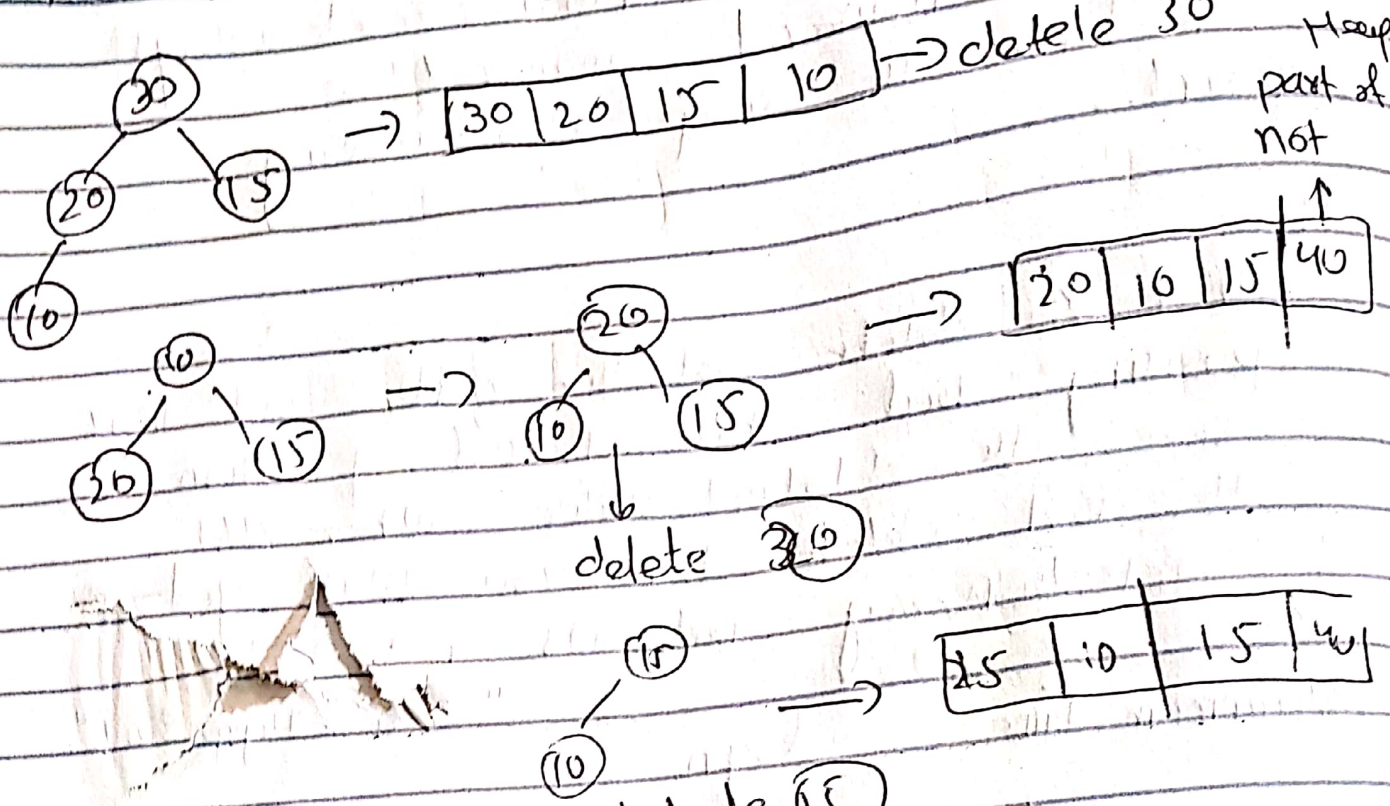
Maximum Time $\log n$

Heap Sort





Now start deleting



Have some mistake do it when you again do it

Now this become we get sorted Array

Delete time - delete one Node time = log n
= delete N Node time = n(log n)
= Creating the Heap time = n log n
deleting the Heap time = n log n
Now we get it sorted Array
time = n log n + n log n
= 2 n log n
time = O(n log n)

Heapify → Heap property

- Max Heap

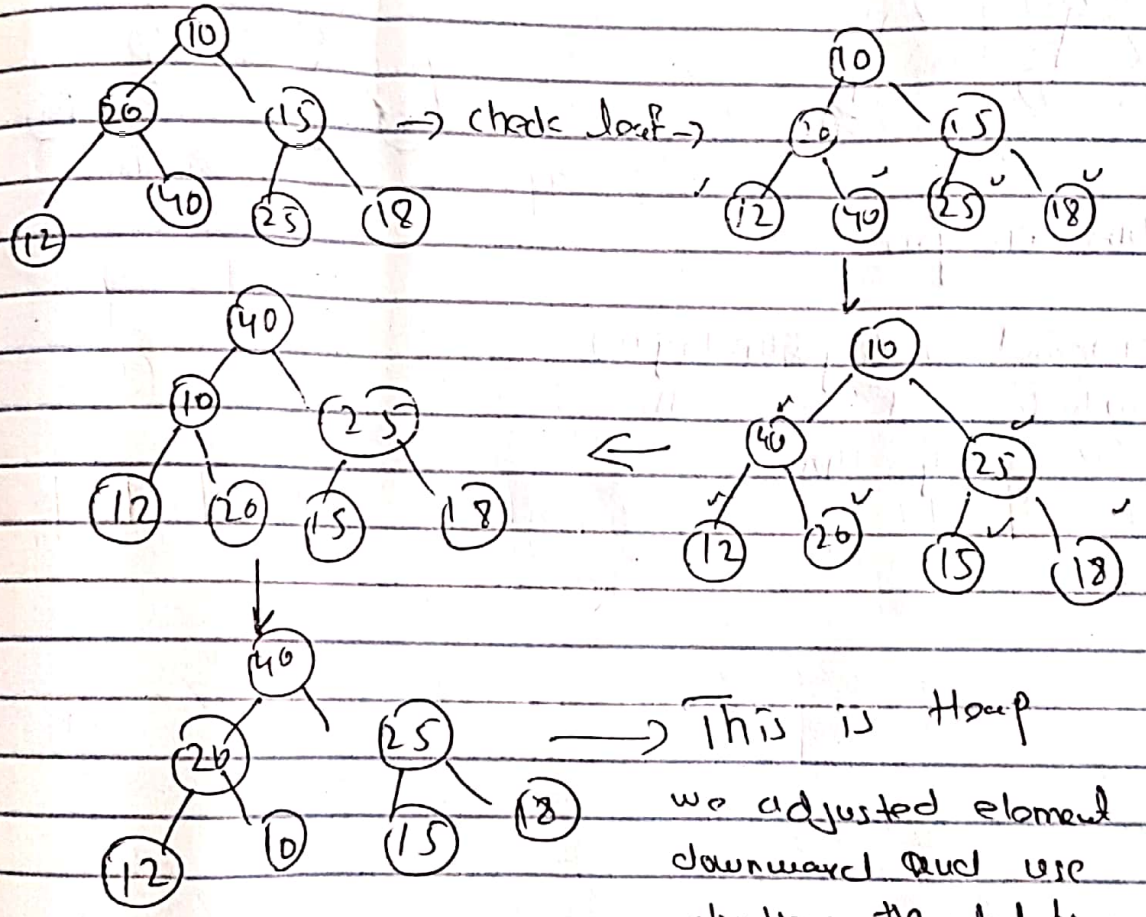
For any Given Node:

If P is parent of Node C:

then key(the value) of P is greater than or equal to key of C

Creating Heap = upward → from leaf

delete Heap = downward → from root



→ This is Heap

we adjusted element downward and use strategy to deleting the Node.

Time taken Heapify = $O(n)$

Create and deleting = $n(\log n)$

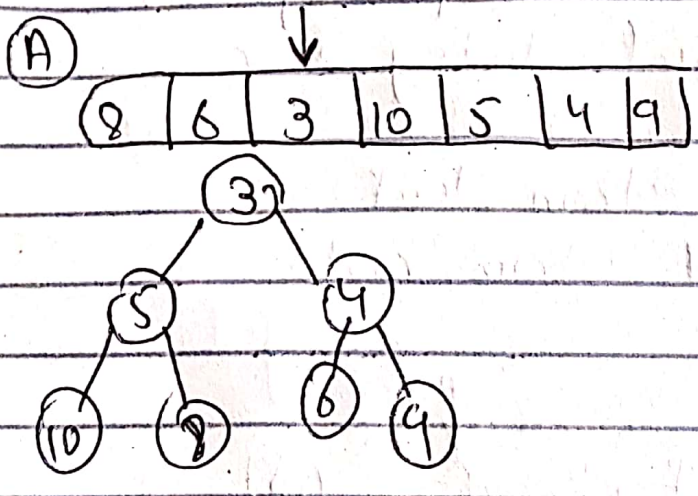
$O(n) \not< (n \log n)$

↳ more useful

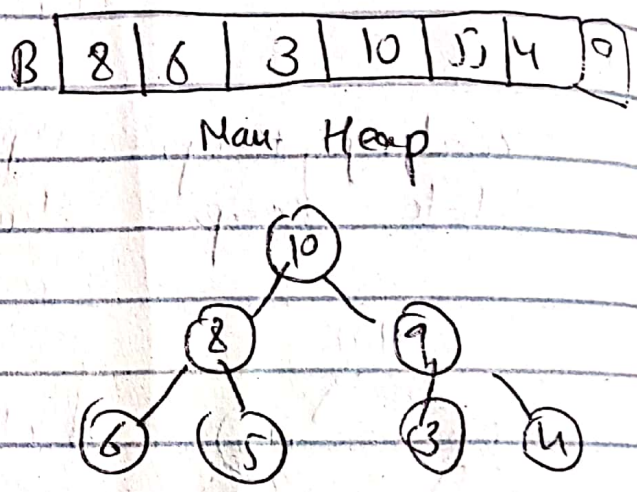
Priority Queue

Heap Priority Queue

Small Number High priority



large Number High Priority



Time Analysis

delete and creating $O(n \log n)$
which best for the
Priority Queue