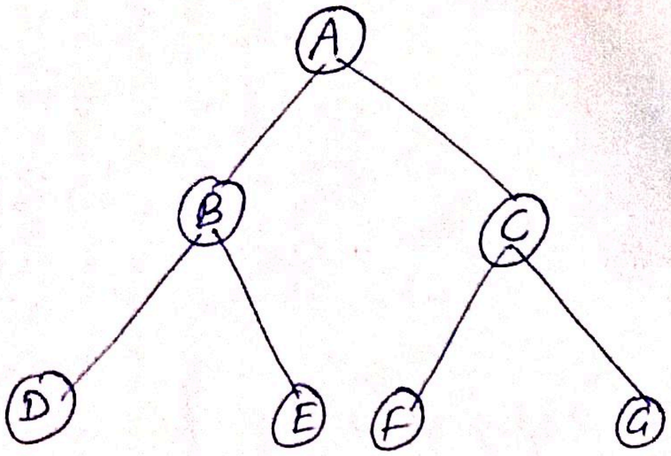
Heap



if a Node is at index — i

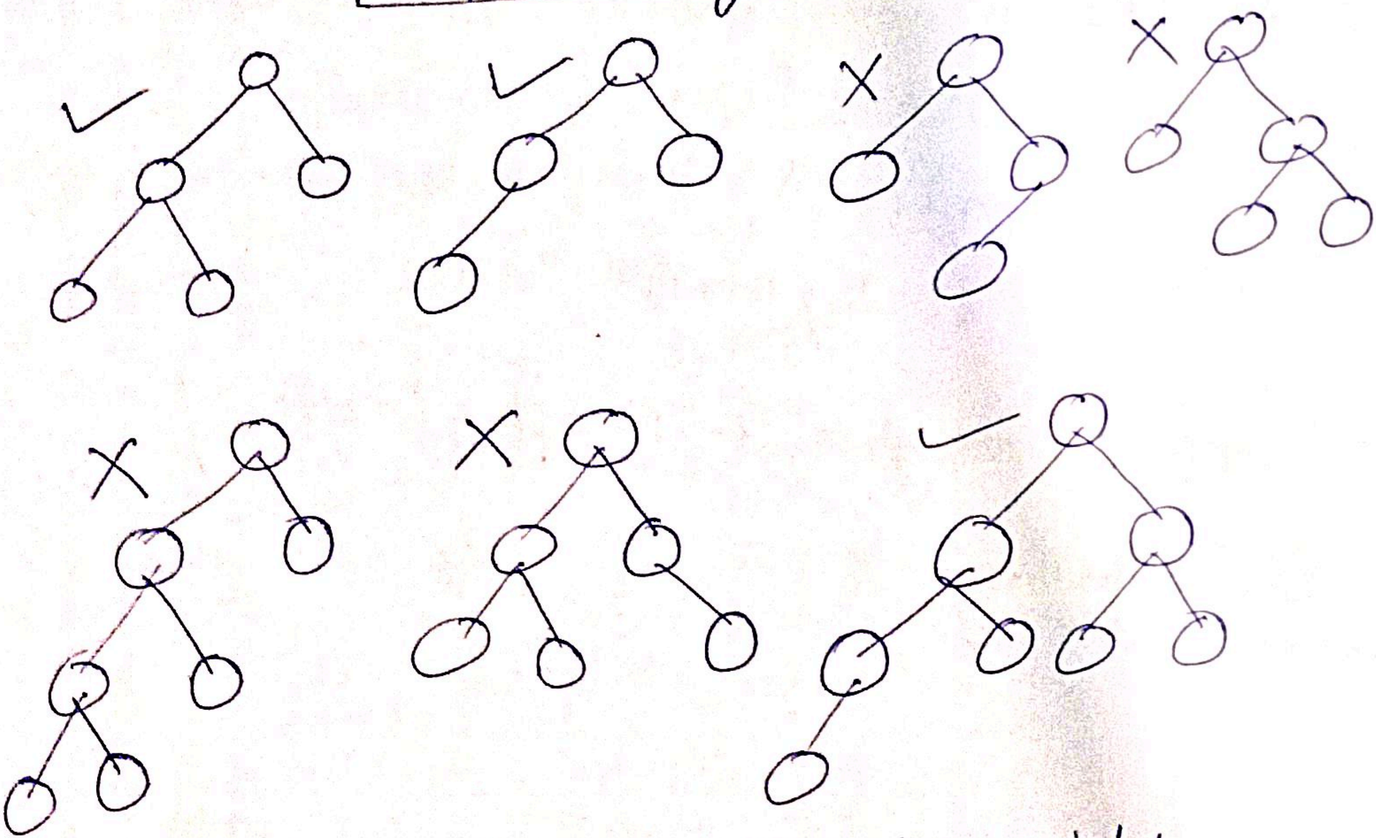
its left child is at — $2 \times i$

its right child is at — $2 \times i + 1$

its parent is at — $\left\lfloor \frac{i}{2} \right\rfloor$

A	B	C	D	E	F	G

Complete Binary Tree



→ Without completing left Do not go right

→ Without completing current level Do not go to Next level.

① If CBT contain K level

$\therefore$ Total node $= 2^K - 1$

② In CBT at K^{th} level

$\therefore$ Total nodes $= 2^{K-1}$

③ In CBT contain nodes leafnode $= \lceil \frac{n}{2} \rceil$

Internal node $= \lfloor \frac{n}{2} \rfloor$

④ In CBT, n-nodes are there & K-level

$n = 2^K - 1$

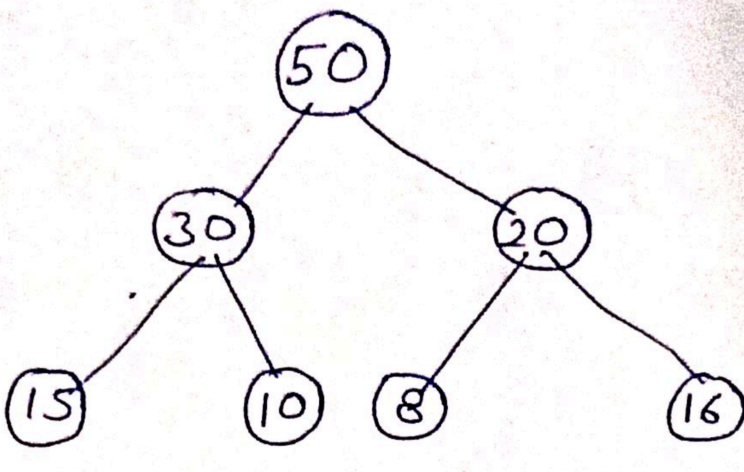
$n+1 = 2^K$

$K = \log_2(n+1)$

⑤ Height = No. of level − 1 $= \boxed{\log_2(n+1) - 1}$

Height ↓ Longest distance between Root to Leaf

Max Heap

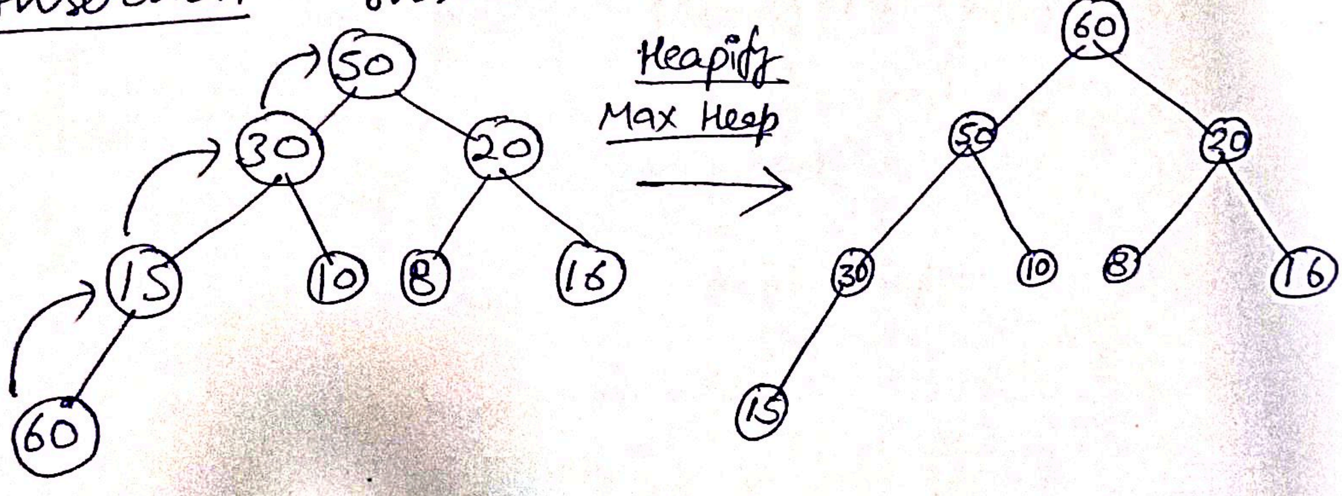


50	30	20	15	10	8	16

Max Heap is a complete Binary tree satisfying the condition that every node is having the element greater than all its descendents.

Insertion

Insert 60



50	30	20	15	10	8	16	60
1	2	3	4	5	6	7	8

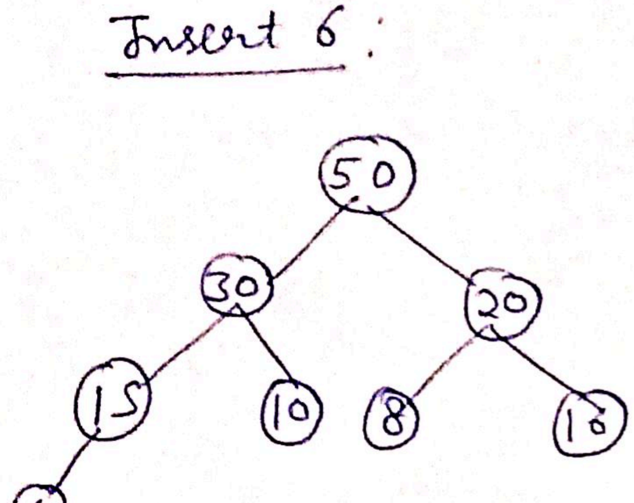
60	50	20	30	10	8	16	15
1	2	3	4	5	6	7	8

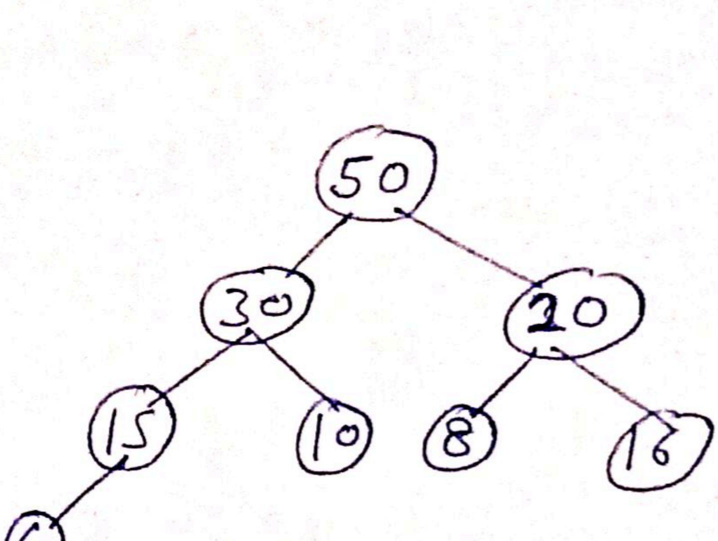
No. of Comparisons = 3

No. of swaps = 3 (Depends on Height)

T.C = O(logn)

Insert 6 :





50	30	20	15	10	8	16	6

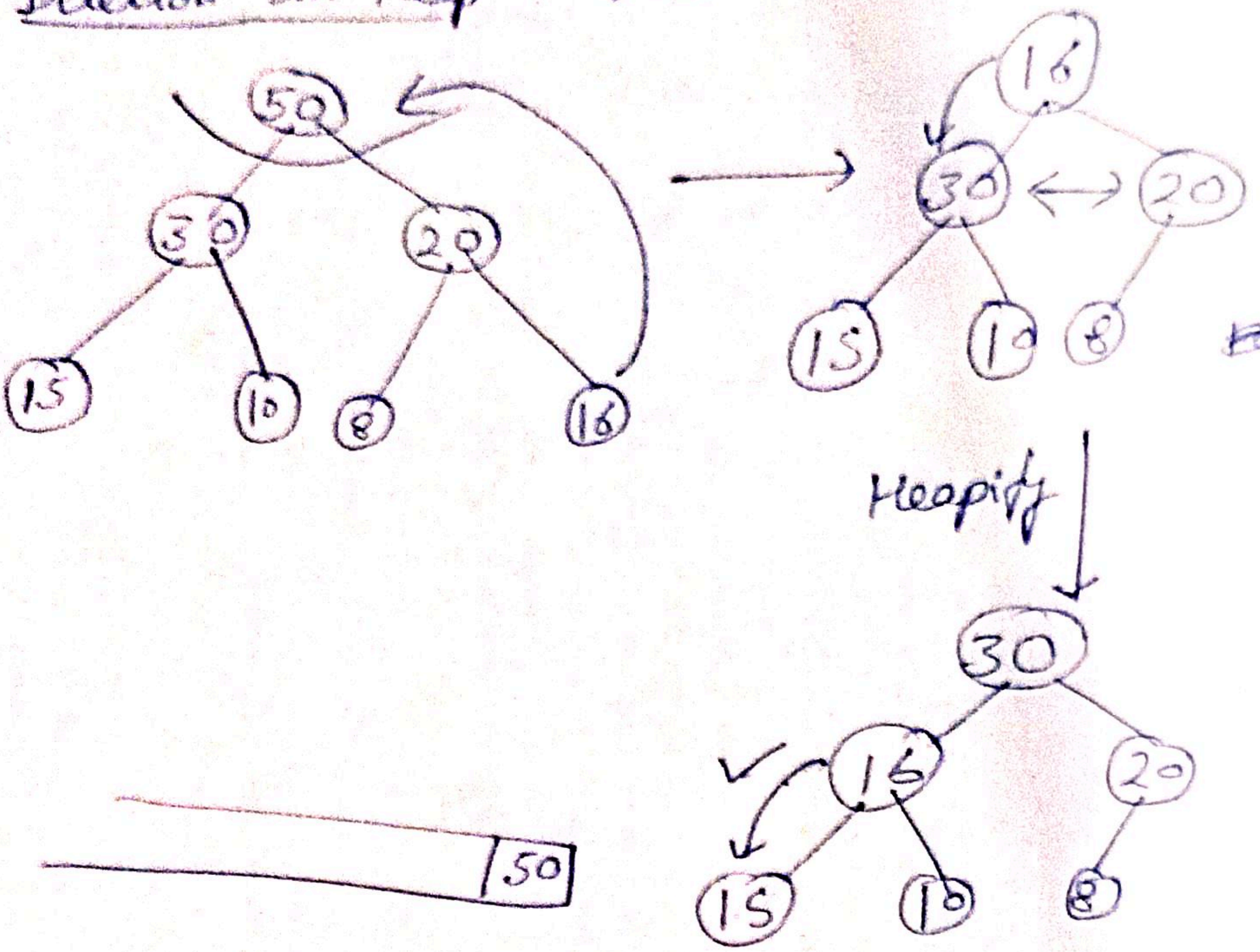
50	30	20	15	10	8	16	6

No. of swaps = 0   T.C. = O(1)

Time taken for inserting one element in a Heap is minimum O(1)

& Maximum O(logn)

Deletion in Heap   Delete 50



Both in insertion & deletion adjustment is done but directions are different.

T.C. = O(logn)

From Max Heap, whenever you delete, you get the largest element from the Heap.

Heap Sort works in two steps

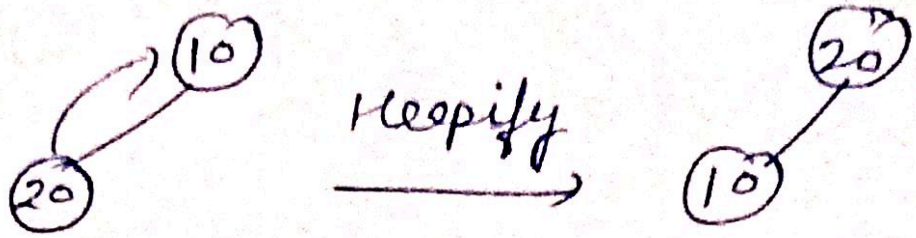
1. Create Heap
2. Delete elements one by one

Create Heap

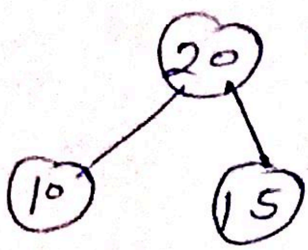
1	2	3	4	5
10	20	15	30	40

Insert 10

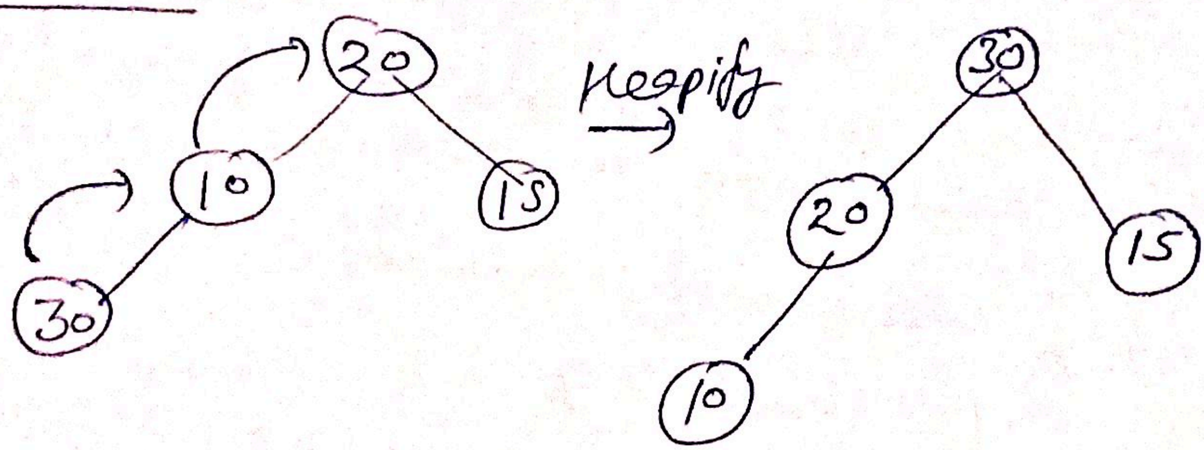
Insert 20



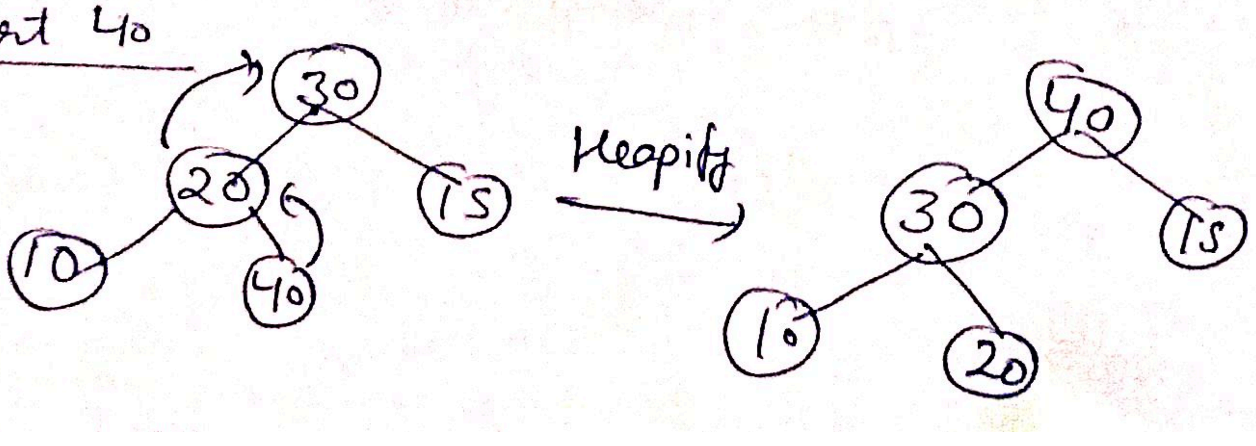
Insert 15



Insert 30



Insert 40

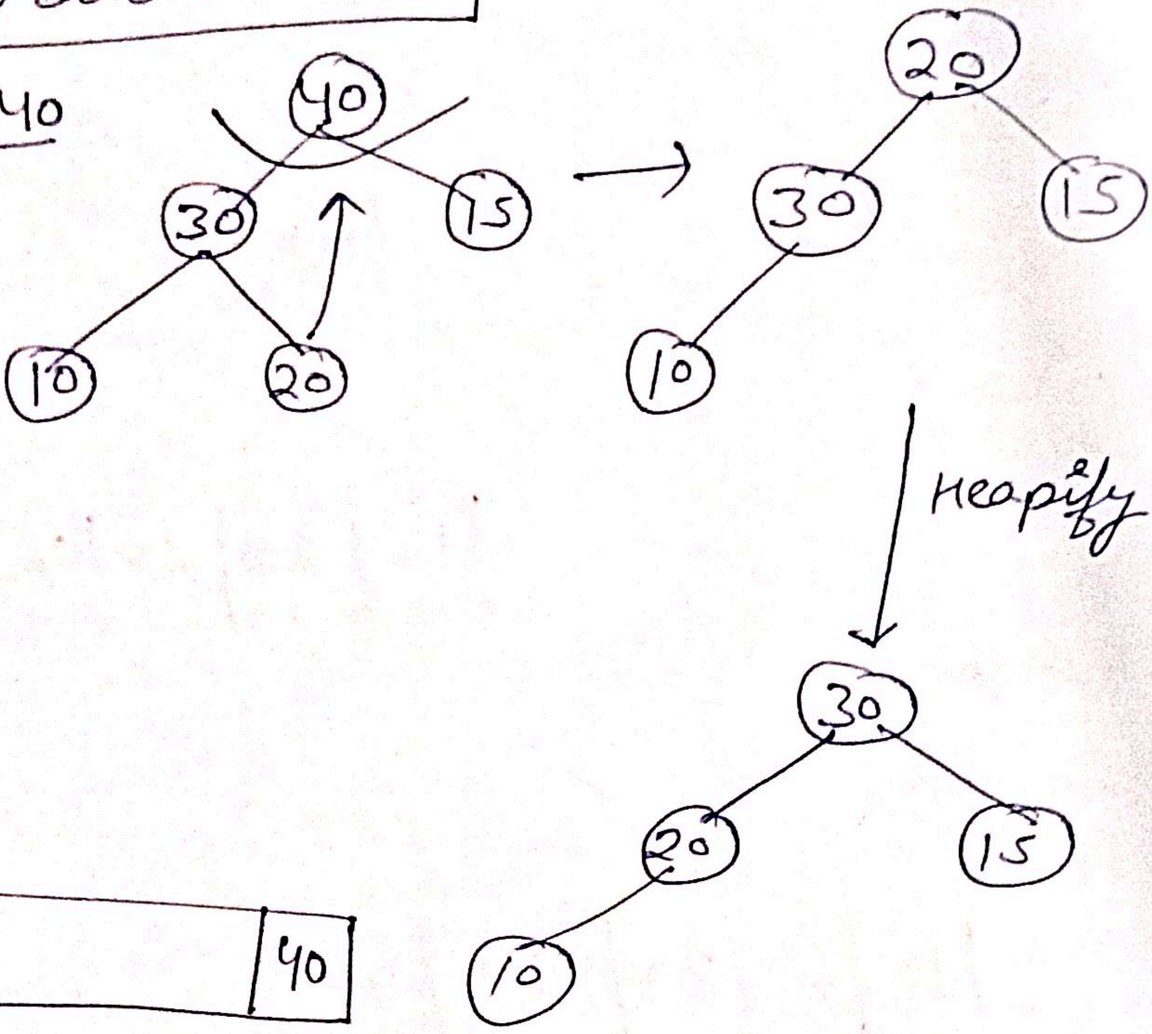


Inserting each element is taking time logn.
Therefore, n elements insertion will take $\boxed{n\log n}$

Till here, First step is completed.
i.e. ① Create Heap

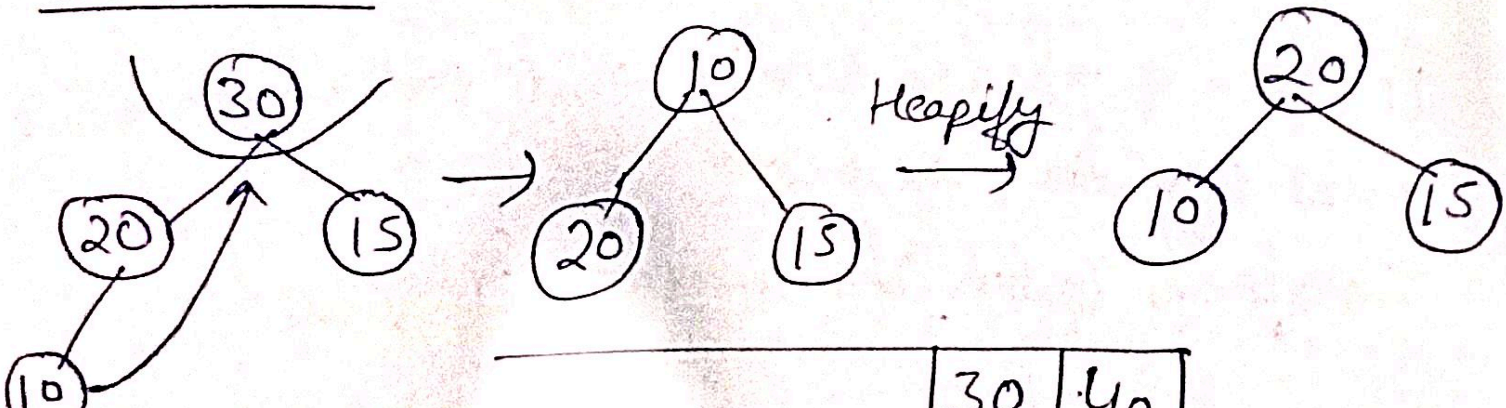
Now, ② Delete elements

Delete 40



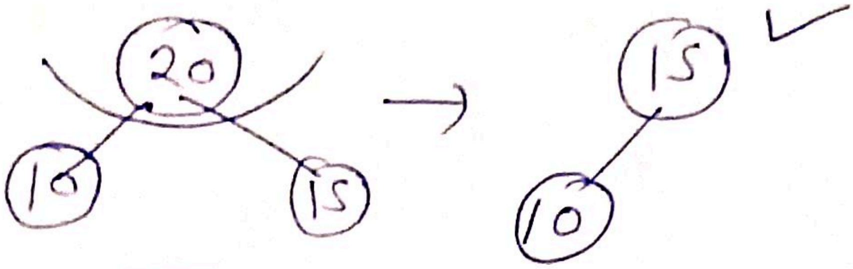
	40

Delete 30



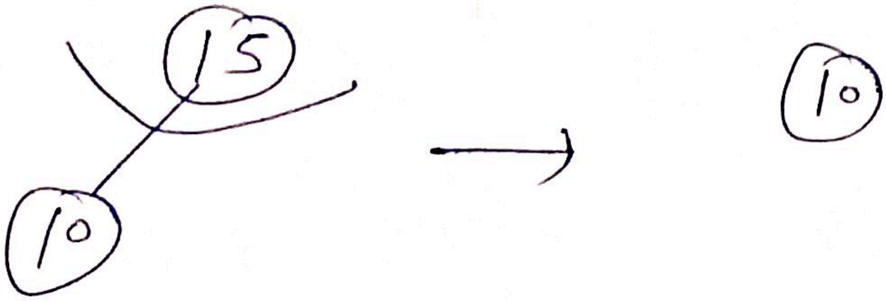
	30	40

Delete 20



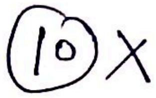
20	30	40

Delete 15



15	20	30	40

Delete 10



10	15	20	30	40

n elements, we are deleting and each element takes logn time to delete

Therefore, Total Time taken $= O(n.\log n)$

Create Heap — $n \log n$

Deletion — $n \log n$

$2 n \log n$

$T.C. = O(n \log n)$

Heapify is a process of creating a Heap.

HeapSort(A)

1. Build-Max-heap(A) — $n \log n$
2. for $i = A.length$ down to 2 ——— n
3. exchange $A[1]$ with $A[i]$ —— n
4. $A.heapsize = A.heapsize - 1$ — n
5. Max-heapify(A, 1) — $n \times \log n$

$O(n \log n)$

Build-Max-heap(A, i)

1. $A.heapsize = A.length$
2. for $i = \lfloor A.length/2 \rfloor$ down to 1 $\Rightarrow \frac{n}{2}$
3. Max-heapify(A, i) $\Rightarrow \frac{n}{2}(\log n)$

Max-heapify(A, i) $\Rightarrow (\log n)$

1. $l = left(i)$
2. $r = Right(i)$
3. if $l \leq A.length$ and $A[l] > A[i]$
4. largest = l
5. else
6. largest = i
7. if $r \leq A.length$ and $A[r] > A[largest]$
8. largest = r
9. if largest $\neq i$
10. exchange $A[i]$ with $A[largest]$
11. Max-heapify(A, largest)

Example

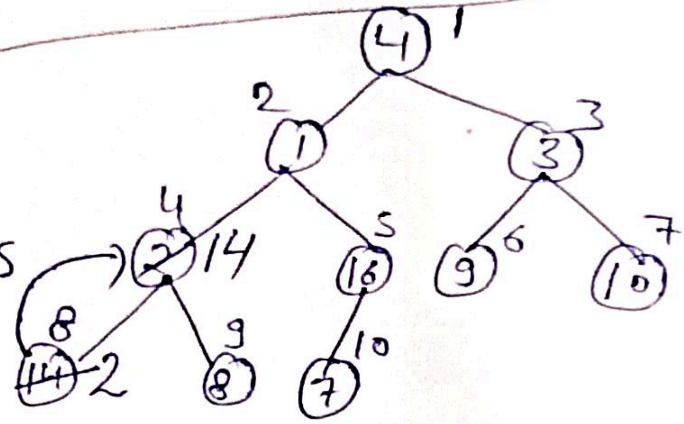
1	2	3	4	5	6	7	8	9	10
4	1	3	2	16	9	10	14	8	7

Build-Max heap (A.i)

1. A. heapsize = 10

2. $\underbrace{\frac{10}{2} = 5 \rightarrow 1}$

Max-heapify (A. i) → 5



$i = 5$

$l = left(5) = 10$

$r = nil$

if $10 \leq A.length$ & $A[10] > A[5]$

$7 > 16$ ✗

else

largest = i = 5

if largest ≠ i ✗

Max-heapify (A, largest)

5

$i = 4$ Max-heapify(A, 4)

$l = left(4) = 8$

$r = right(4) = 9$

if $8 \leq A.length$ & $A[8] > A[4]$

$14 > 2$

largest = 8

if $r \leq A.length$ & $A[r] > A[largest]$

$9 \leq A.$ ✓ & $A[9] > A[8]$

$8 > 14$ ✗

if largest ≠ i | exchange A[4] with A[8]

$8 \neq 4$