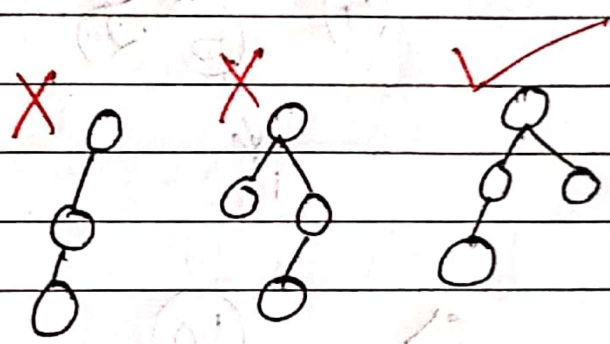
Heap

Heap is a almost complete binary tree.

It must follow two properties.

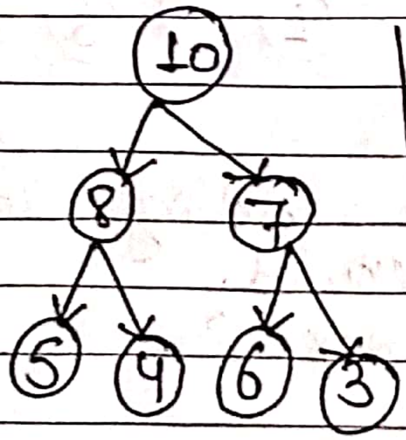
(i) Structural property.
(ii) Ordering property.



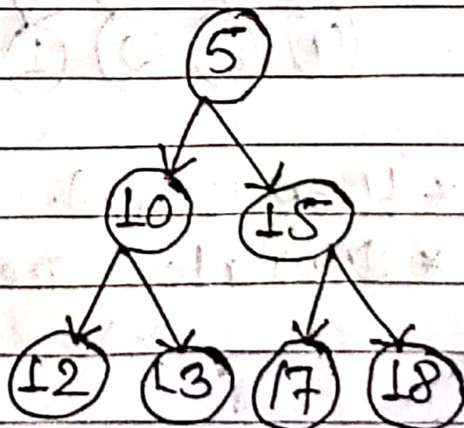
Heap

max. Heap (parent >= child)

min. Heap (parent < child)



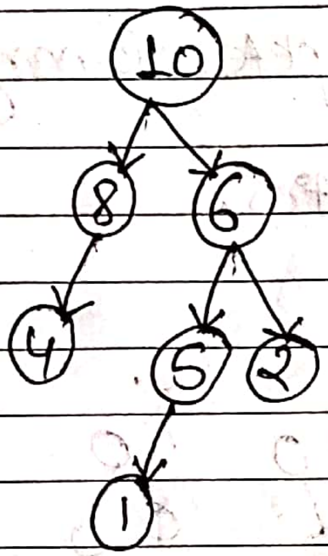
log n



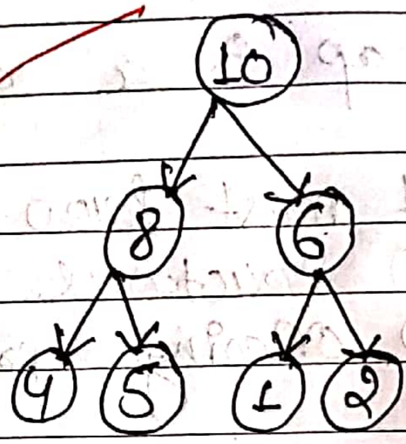
10	8	7	5	4	6	3
1	2	3	4	6	6	7

Which of the following is max. Heap ?

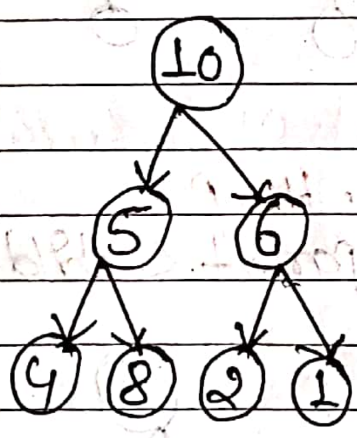
A)



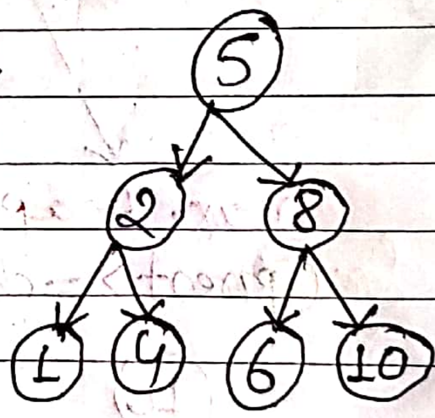
B)



C)



D)



* Heap is not used for Searching purpose.
* Duplicate are also allowed.

node at index i

left child = $2*i$
Right child = $2*i+1$.

Scanned with CamScanner

Heap Tree Construction

Insert key one by one in the given order
$O(n \log n)$

Heapify method
$O(n)$.

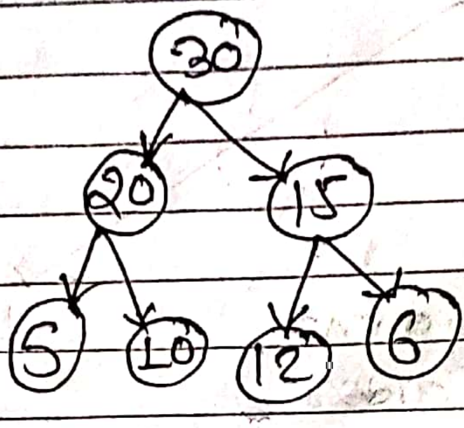
Algo Analysis

1) Insert key one by one
   → To insert a key into empty Heap takes $O(1)$ times.

2) To insert a key into already constructed Heap in worst case $\log(n)$ comparision and $\log(n)$ for swapping.

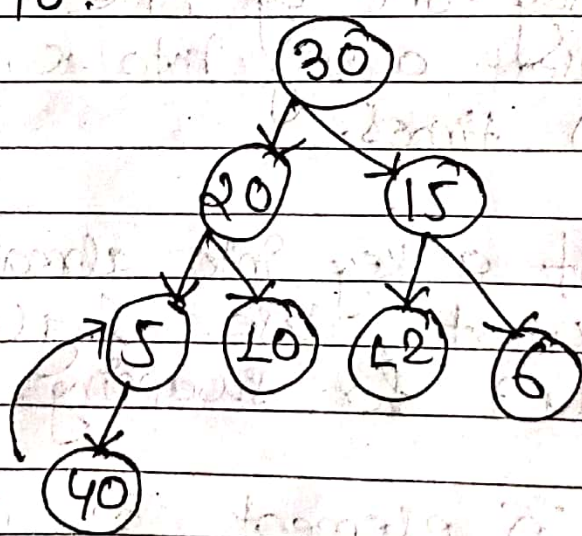
3) Total 'n' elements so, $O(n \log n)$ time.

Insertion In MAX Heap

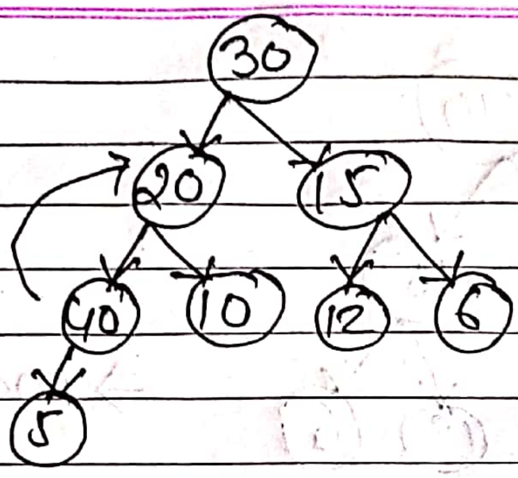


30	20	15	5	10	12	6
1	2	3	4	5	6	7

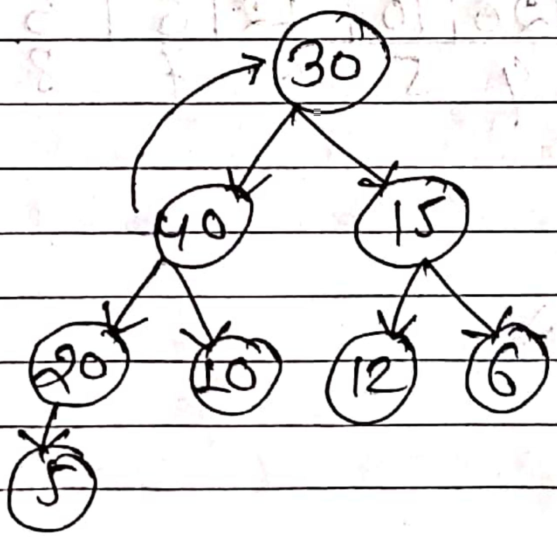
Insert = 40.



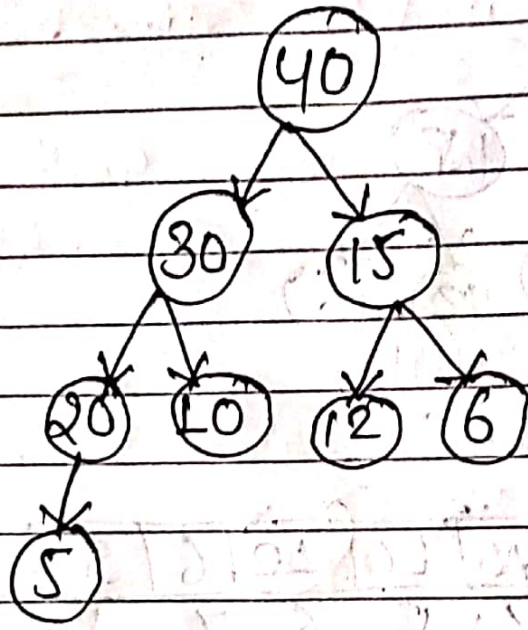
30	20	15	5	10	12	6	40
1	2	3	4	5	6	7	8



30	20	15	40	10	12	6	5
1	2	3	4	5	6	7	8



30	40	15	20	10	12	6	5
1	2	3	4	5	6	7	8



⇒ max. Heap

40	30	15	20	10	12	6	5
1	2	3	4	5	6	7	8

// program to insert an element in Heap

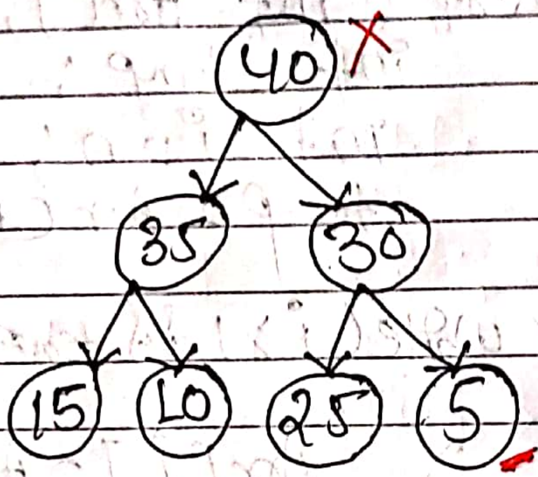
```
void Insert (int Arr[], int n){
        int temp;
        int i = n;
        temp = Arr[n];

    while(i>1 && temp > arr[i/2]){
            Arr[i] = Arr[i/2];
            Arr[i/2] = temp;
            i = i/2;
        }
    }
```

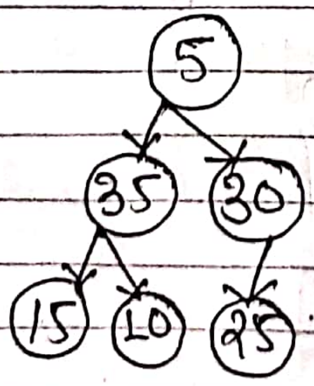
O(logn)

Deletion In Heap

→ In Heap we can only delete from root value.



40	35	30	15	10	25	5
1	2	3	4	5	6	7



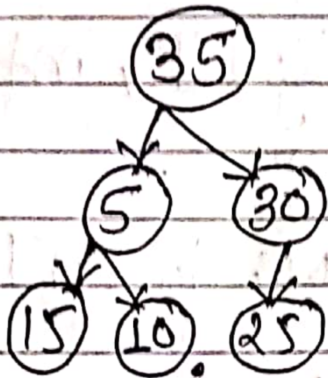
i ↓   ↓ j

5	35	30	15	10	25	40
1	2	3	4	5	6	7

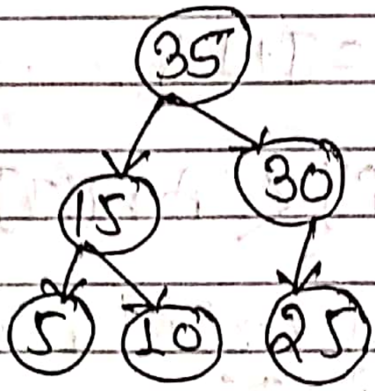
Heap part

left child $= 2 * i$

right child $= 2 * i + 1$



35	5	30	15	10	25	40
1	2	3	4	5	6	7



35	15	30	5	10	25	40
1	2	3	4	5	6	7

Heap part.

// Deletion in Heap

```
void Delete (int Arr[], int n) {
    int i;
    int J;
    swap (Arr[1], Arr[n]);
    i = 1;
    J = 2 * i;
    while ( J < n-1) {
        if (Arr[J+1] > Arr[J]) {
            J = J+1;
        }
        if (Arr[i] < Arr[J]) {
            swap (Arr[i], Arr[J]);
            i = J;
            J = 2 * i;
        }
    }
}
```

Date ____
Page ____

Heap Sort

(i) Create Heap of 'n' elements → $n \log n$

(ii) Delete 'n' elements 1 by 1. → $n \log n$

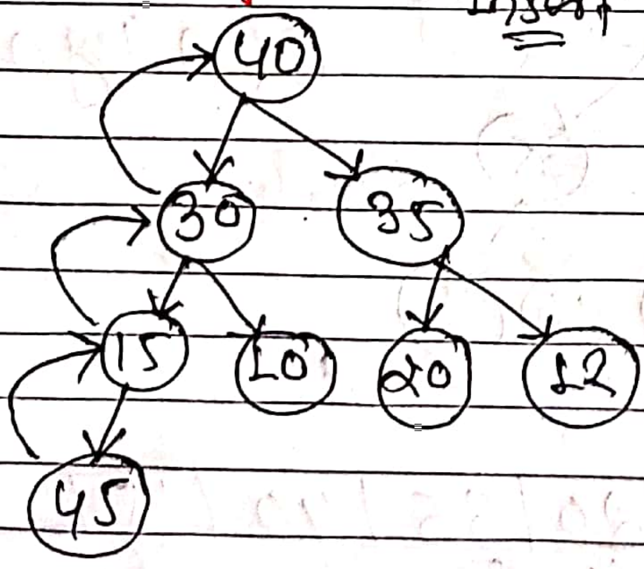
Heap Sort $= n\log n + n\log n$

$= 2n\log n$

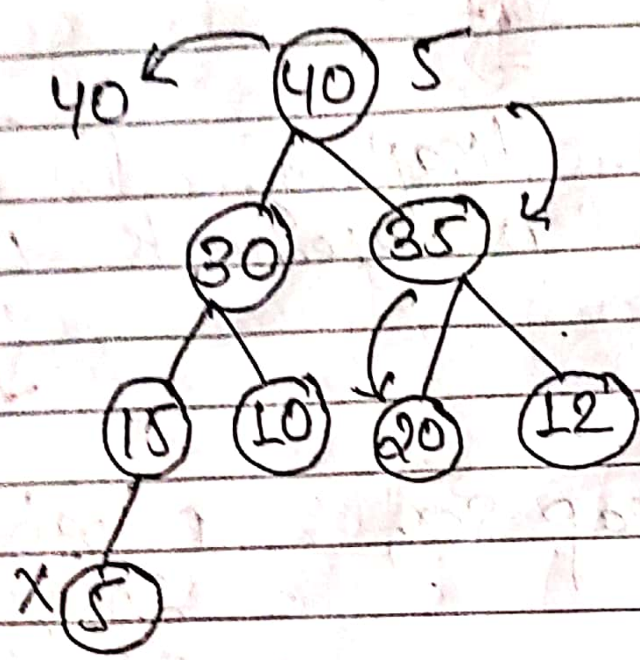
$= O(n\log n)$

Heapify

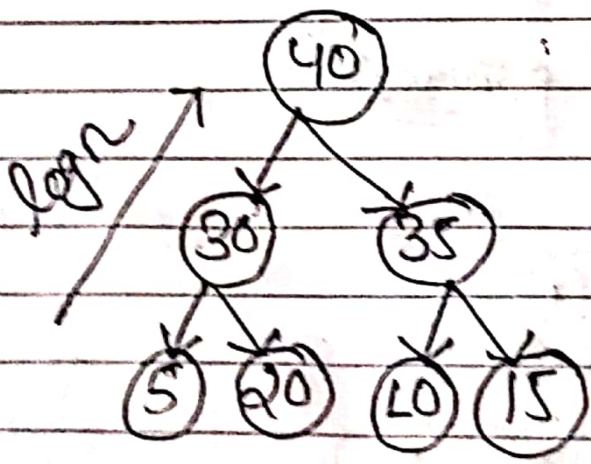
Insert



Delete



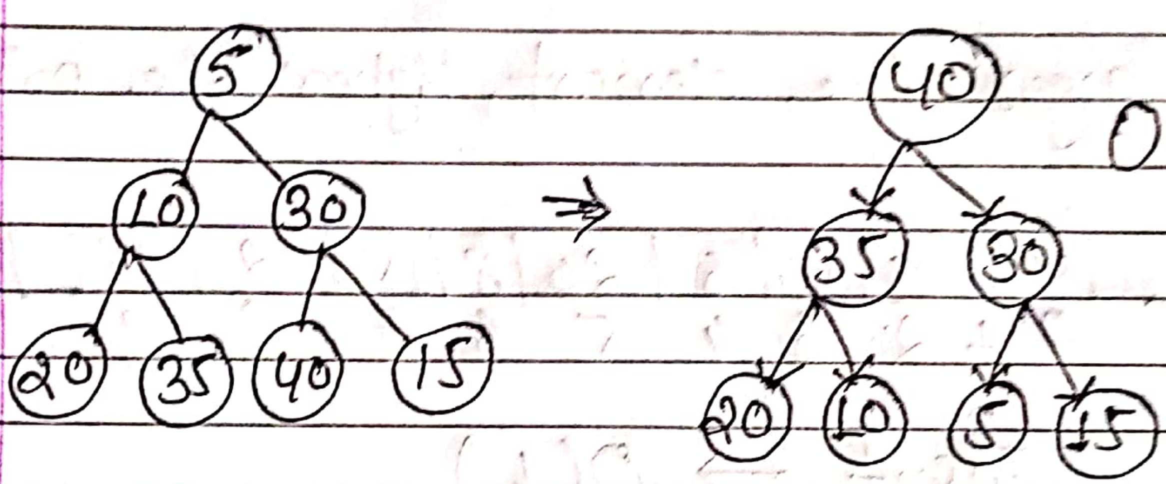
Create Heap (left to right)



$O(n \log n)$

5	10	30	20	35	40	15
1	2	3	4	5	6	7

Heapify (right to left)



$O(n)$

5	10	30	20	35	40	15
1	2	3	4	5	6	7

Binary Heap as priority Queue

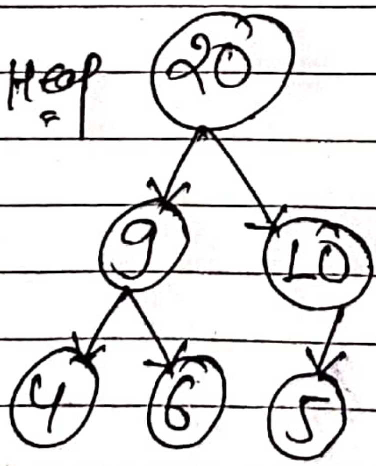
elements → 4, 9, 5, 10, 6, 20, 8, 15, 2, 18.

larger the element, higher the priority.

4	9	5	10	6	20	8	15	2	18	
0	1	2	3	4	5	6	7	8	9	

Insert → $O(1)$
Delete → $n + n$ ← shifting the element
searching $= 2n = O(n)$

max. Heap



Insert → $O(\log n)$
Delete → $O(\log n)$