ASSIGNMENT: 2

Q1.

```
find smallest (arr[])
    min = arr[0]
    for i=0 to arr.length-1
        if (arr[i] < min)
            min = arr[i]
    return min
```

Time complexity : O(logn)

Q2. A = <28, 52, 17, 35, 24, 48, 11, 20, 17, 30>

Steps :

I	17, 28, [52], 17	(17<52)
II	17, 28, [52], 35	(35<52)
III	17, 28, 35, [52] 24	(24<52)
IV	17, 24, 28, 35, [52] 48	(48<52)
V	17, 24, 28, 35, 48, [52], 11	(11<52)
VI	11, 17, 24, 28, 35, 48, [52], 20	(20<52)
VII	11, 17, 20, 24, 28, 35, 48, [52], 17	(17<52)
VIII	11, 17, 17, 20, 24, 28, 35, 48, [52], 30	(30<52)
IX	11, 17, 17, 20, 24, 28, 30, 35, 48 [52]	

Sort^{d} Array : = <11, 17, 17, 20, 24, 28, 30, 35, 48, 52>

Worst case :-

The worst case is when the array elements are in descending order

worst case : $O(n^2)$

Best case :-

The best case is when the array is already sorted

best case : $O(n)$

Avg. Case :-

average case : $O(n^2)$.

3). Insertion sort runs in $8n^2$ steps

Heap sort runs in $64 n \log n$ steps

for a certain value of n, Insertion sort must outperform heap sort

∴ Insertion must top less number of step to execute then heap sort

∴ $8n^2 < 64 n \log n$

$n^2 < 8 n \log n$

$n < 8 \log n$

∴ for $n > 1$, Insertion sort outperform heap sort.

4.) i) node i is present in 19 th position

$$\left[\frac{19}{12}\right]$$

$[9.5]$

$= 9$

∴ The parent node of node i is present in 9th position.

and i is on the right side of parent node

$\because \frac{19}{2} = 9.5$

$\therefore$ node i is right child.

ii) height $= \lceil \log n \rceil$

$= \lceil \log 19 \rceil$

$= 4.24$

height $= \underline{\underline{4}}$

iii) no. of leaves $= \lceil n/2 \rceil$

$= \frac{100}{2} = 50$

$\because$ leaves have no child

$\therefore$ There are 50 nodes with zero children.

$\because$ There are even no. of leaves there are only 1 node with 1 child,

$\therefore$ left 49 nodes all have 2 children.

4) The left subtree has 63 nodes

and the right subtree has 36 nodes

5). i) no. of leaves $\lceil n/2 \rceil = 100/2$

$= \boxed{50}$

$\therefore$ There are 100 nodes and 50 are leaves.

ii) Total no of leaves = 50

iii) height of heap = $\lceil \log n \rceil$

$= \log 100$

$= 6.64$

$= \underline{6}$

iv) There are 8 nodes at height 3 of the heap.

6). For a max heap, The K^{th} largest item / element must be K^{th} position in heap.

height of heap $= \log_2 n$

height of K^{th} largest element $= \log_2 K$

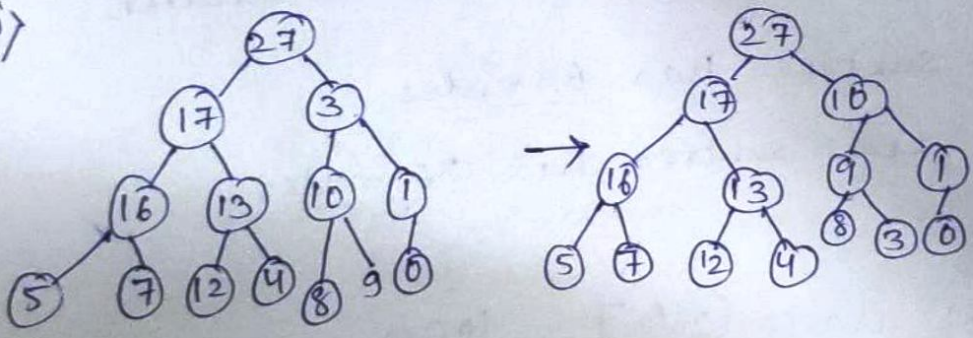
height of 35^{th} largest element $= \log_2 35$

$= 5.12$

$= \underline{5}$

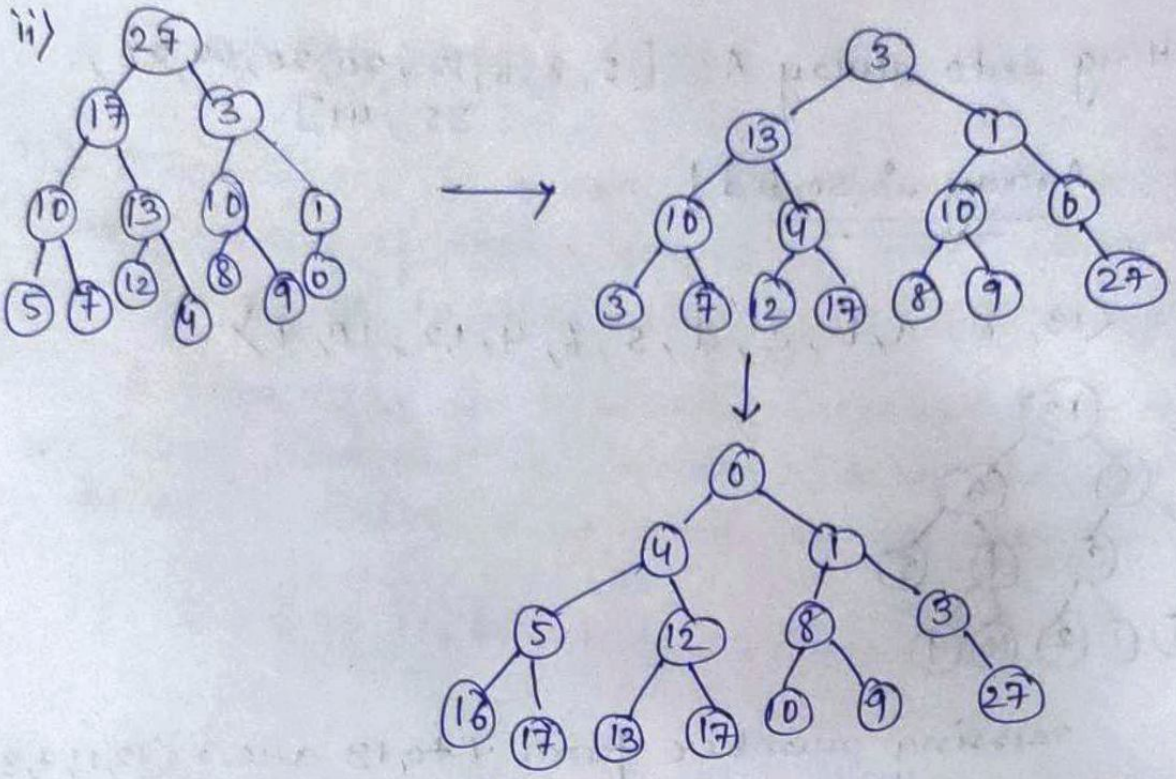
7) A = ⟨27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0⟩

i)



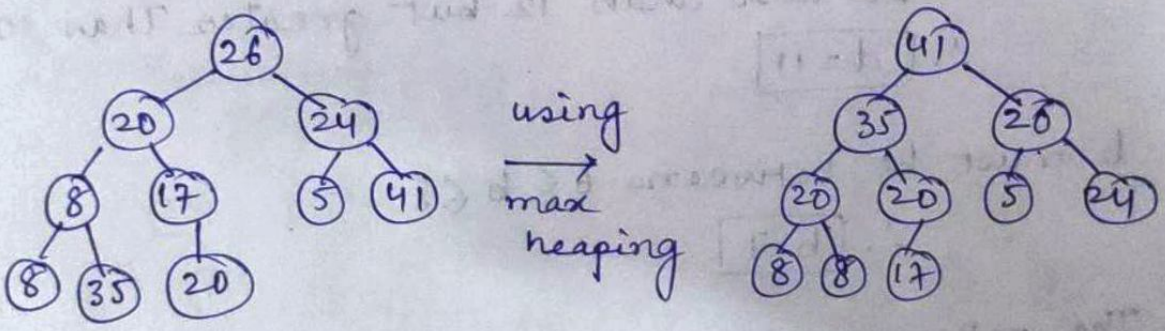
No. of comparison : 2

ii)

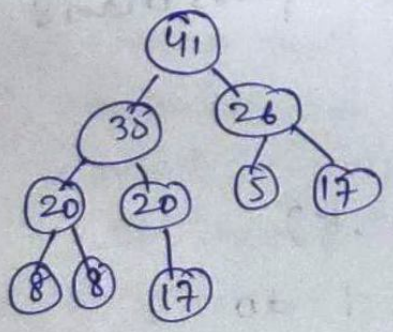


No: of comparison : 11

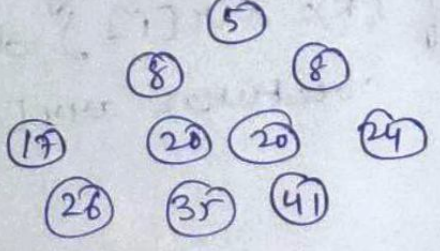
8). A = <26, 20, 24, 8, 17, 5, 41, 8, 35, 20>



putting into array A = [41, 35, 26, 20, 20, 5, 24, 8, 8, 17]



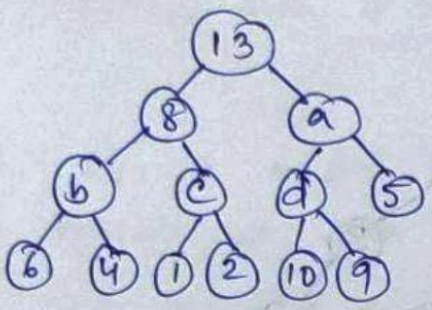
comparing A[1] & A[n] and removing larger elements from heap.



putting into array A = [5, 8, 8, 17, 20, 20, 24, 26, 35, 41]

Array is sorted!

(9). A = <13, 8, a, b, c, d, 5, 6, 4, 1, 2, 10, 9>



missing number from 1 to 13 are = <12, 11, 7, 3>

→ 'i' must be lesser than 13 but larger than 10

∴ a = 12

d must be less than 12 but greater than 10

∴ d = 11

b must be between $6 \leq b \leq 8$

∴ b = 7

The only missing number left is 3.

which is greater than 2 & less than 8

∴ c = 3

10). 1) Print less or Equal (arr[], k)

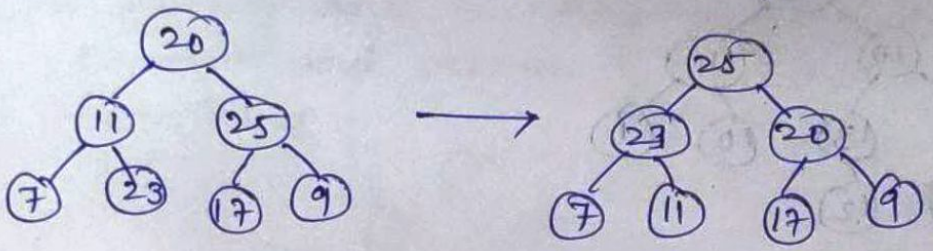
```
{
    for i=1 to arr.length-1 do
    {
        if (k)= arr[i] ) do
            return arr[i];
    }
}
```

ii) Time Complexity : O(n)

11.) A max heap is a completely binary tree in which the value of each internal node is greater than or equal to the value of children of nodes.

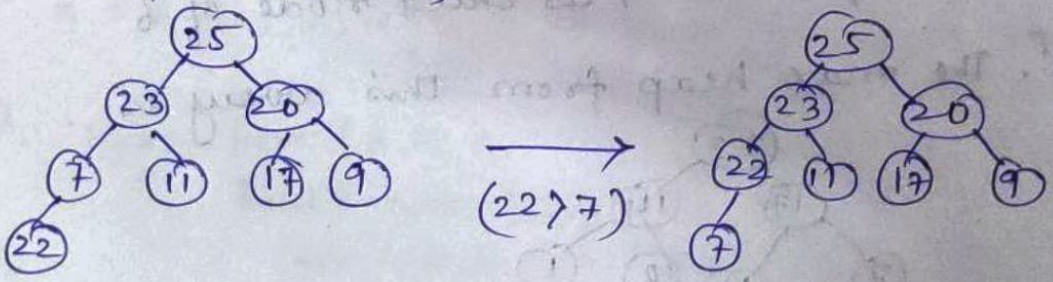
A max heap is typically represent as an array the root element is in A[0] and the last leaf is A[i] where i is total no. of elements.

i) A = <20, 11, 25, 7, 23, 17, 9>



max heap

ii) heap Insert (22)



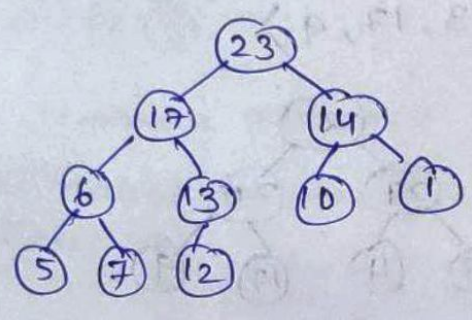
12.) Heap sort is not a stable sort. It does not restrict some order of equal elements in the sorted array.

It reverse the original order of array to implement as heap and it is not time efficient.

13). checkMaxheap(int arr[], int n)

```
for (c=1) to (c<n)
    p = (c-1)/2
    if [arr[p] < arr[c]]
    {
        return false;
    }
return true;
```

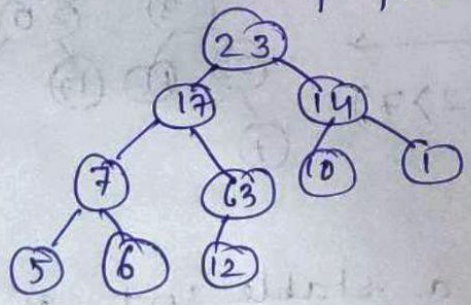
A = <23, 17, 14, 6, 13, 10, 1, 5, 7, 12>



∴ The given array does not represent maxheap

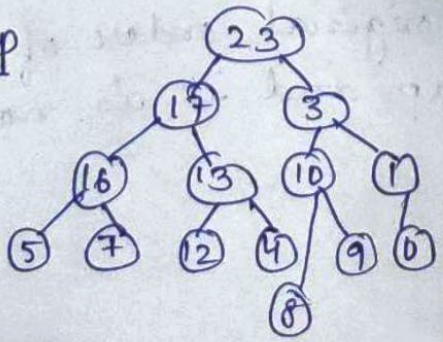
∵ 7 > 6 but 7 is child node of 6

∴ The max heap from this array is .

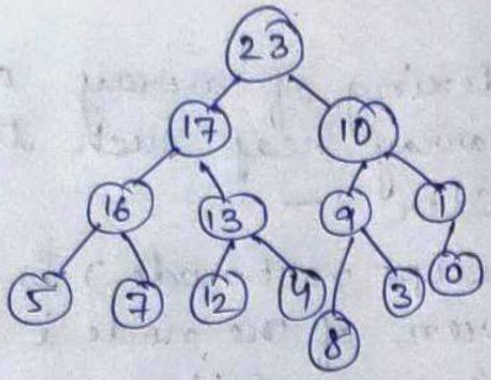


14). A = <23, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0>

Max heap



Step I

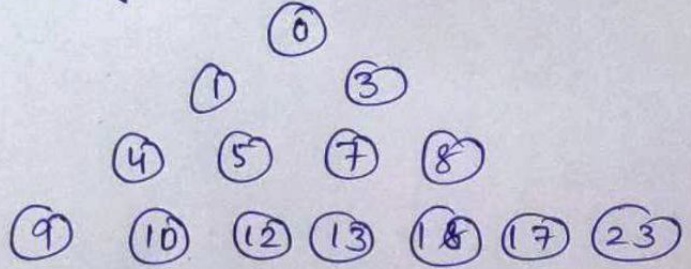


heap sort :-

putting max heap into array.

<23, 17, 10, 16, 13, 9, 1, 5, 7, 12, 4, 8, 3, 0>

comparing and Exchanging A[1] and A[n] element and removing laregest sorted element from heap.



A = <0, 1, 3, 4, 5, 7, 8, 9, 10, 12, 13, 16, 17, 23

15). Heap-Sort (A, n)

build max-heap (A)

for i = A.length do upto 2

exchange A[1] ↔ A[i]

heap-size [A] = heap-size[A] - 1;

max-heapify (A, 1)

Heap sort is not a stable sorting algorithm

∴ It does not matter if array is in ascending order or descending order.

It will take the value /time both of them to compute.

∴ Best case = Worst case = O(n log n)

16). i) Assuming 0 based indexing of array or array represent a k-array heap such that for any node we consider :—

Parent of node i (except root node) is located at index $(i-1)/k$. children of the node i are at indices $(ki)+1$, $(ki)+2$, ... $(ki)+k$.