Lecture 51

(Heap)

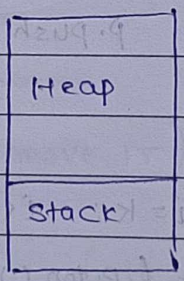
By help of Recursion
↳ gives segmentation fault
space Complexity : O(N)

Stack :
↳ solved and run properly
space Complexity : O(N)

Segmentation fault : We want to access that memory which does not exist.

Recursion works in Stack Memory

Heap ki space
jyada hota hai.



Stack works in heap. On heap, only STL library Command Works.

Heap → Max Heap
↳ Min Heap
} priority Queue.

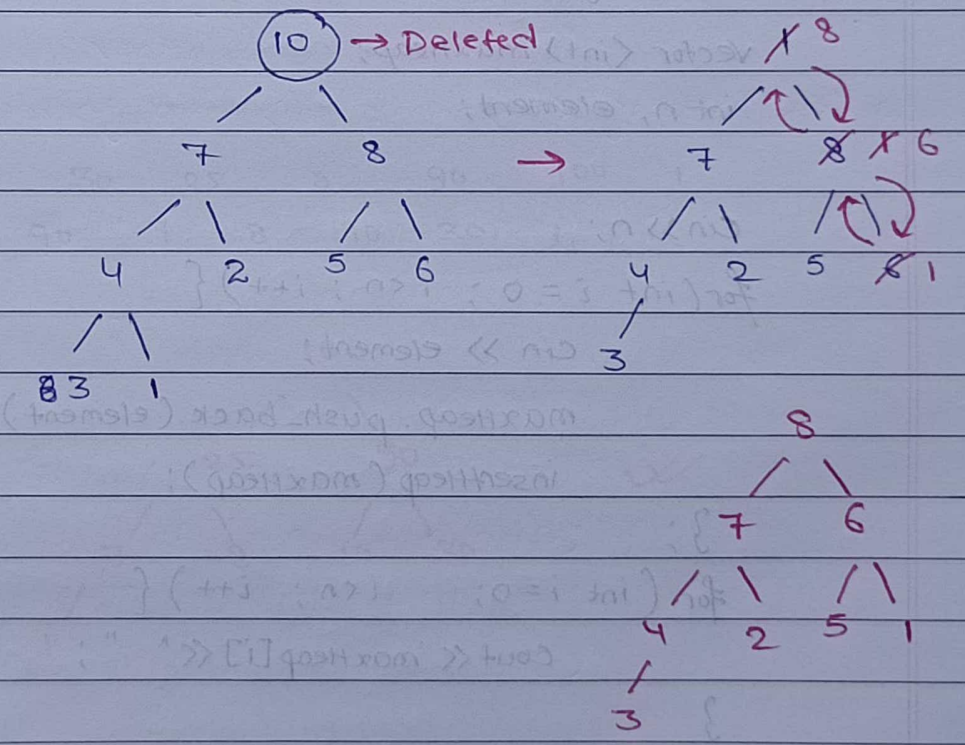
In Max Heap,

parent Node >= child Node

Max Heap & Min Heap can be represented in tree and Array both.

In Heap, creation always starts from top.
In Heap, Deletion always starts from top.

After Deleting the top Node, replace it from the last Node and then swap accordingly.



For creation, we have to use level order traversal in tree.

In Array, we easily access the Nodes and know about the blank space.

parent = (i-1)/2

i = child.

By tree, find the last Node takes time.

But in array, we know about the last space.

Code :

```
#include <iostream>
#include <vector>
using namespace std;

int main() {
    vector <int> maxHeap;
    int n, element;

    cin >> n;
    for(int i = 0; i<n; i++) {
        cin >> element;
        maxHeap.push_back(element);
        insertHeap(maxHeap);
    };
    for (int i=0;  i<n; i++) {
        cout << maxHeap[i] << " ";
    }
return 0;
}
Void insertHeap( vector <int> &maxHeap ) {
    int index = maxHeap.size() -1;
    int parent;
    while ( index > 0) {
        parent = (index -1)/2;
    // parent is small
```

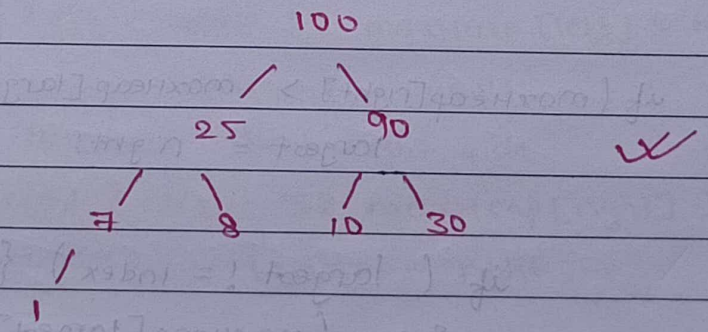
```
if (maxHeap [parent] < maxHeap [index]) {
    swap ( maxHeap [parent], maxHeap [child]);
                                      index
    index = parent;
}
// parent is big
else
    break;
}
}
```

output :

8

10 7 30 25 8 90 100 1

100 25 90 7 8 10 30 1



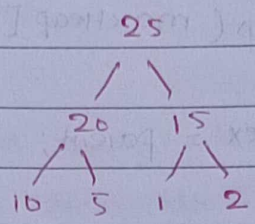
Time Complexity : N log N

N → N element creation

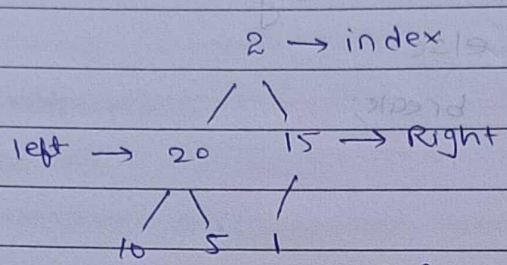
log N → Every element swap to the Max of tree height.

classmate
Date
Page

* Deletion



Delete




```
Heapify (maxheap, largest) {
    largest = index;
    left = 2 * index + 1;
    right = 2 * index + 2;

if (maxHeap[left] > maxHeap[largest])
        largest = left;

if (maxHeap[right] > maxHeap[largest])
        largest = right;

    if ( largest != index) {
        Swap (maxHeap[largest], maxHeap[index]);
        Heapify (maxHeap, largest);
    }
```

```
DeleteHeap(maxHeap);    // function call

Void DeleteHeap(vector<int> &maxHeap){
    // Replace first element with last element
    maxHeap[0] = maxHeap[maxHeap.size()-1];
    // Delete the last element
    maxHeap.pop_back();
    // Correct position pe le kr jao
    Heapify(maxHeap, 0);
}

Void Heapify(vector<int> &maxHeap, int index){
    int largest = index;
    int left = 2*index + 1;
    int right = 2*index + 2;
    int size = maxHeap.size();
    // check left side.
    if(left < size && maxHeap[left] > maxHeap[largest])
        largest = left;
    // check for right side
    if(right < size && maxHeap[right] > maxHeap[largest])
        largest = right;

    // swap.
    if(largest != index){
        swap(maxHeap[largest], maxHeap[index]);
        Heapify(maxHeap, largest);
    }
    return;
}
```

Time Complexity : O(log N)

Space Complexity : O(log N)

→ Max Heap :

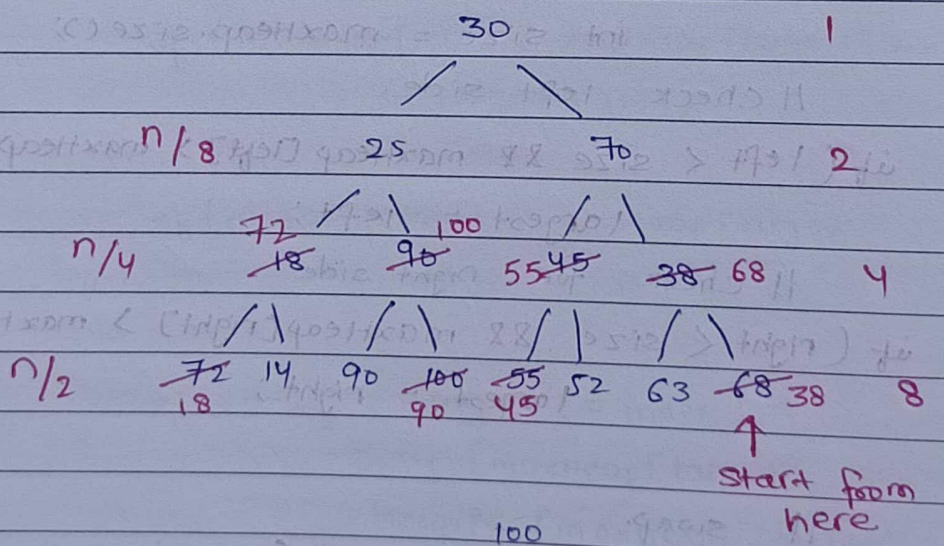
↳ Time Complexity : N log N.

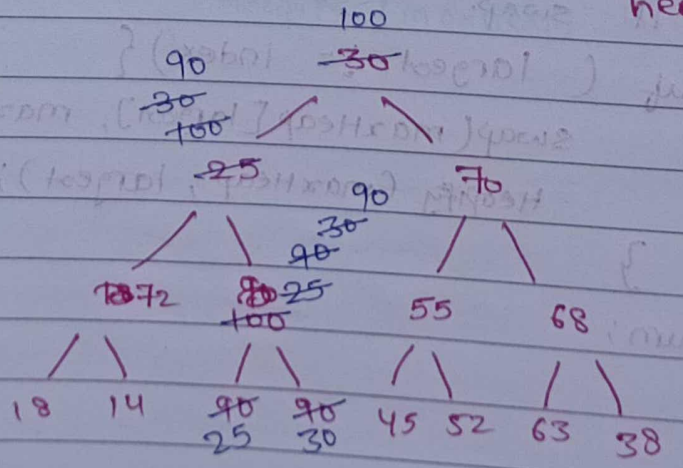
↓

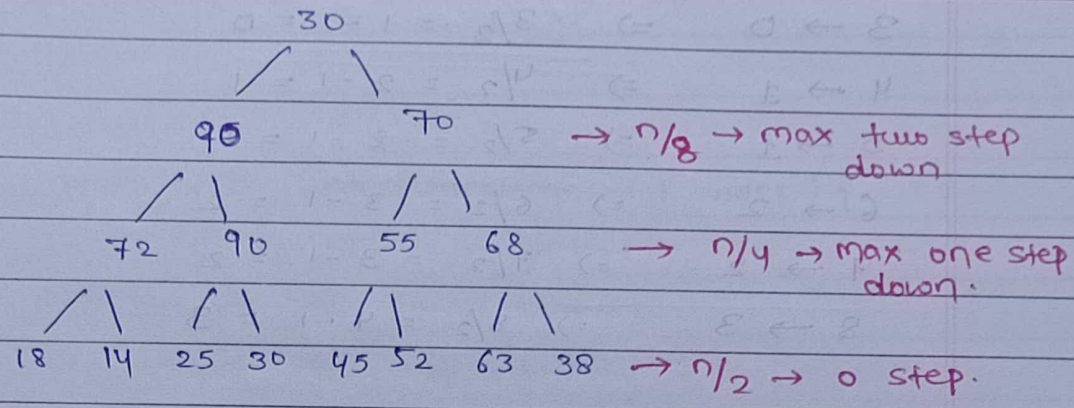
Then we can't use sorting instead of using Max Heap.

But, for Max Heap we can optimize N log N to N.

lets see how :-







$$\left[\frac{n}{4} \times 1 + \frac{n}{8} \times 2 + \frac{n}{16} \times 3 + \frac{n}{32} \times 4 - - - \right]$$

$$= n \left\{ \underbrace{\frac{1}{2^2} + \frac{2}{2^3} + \frac{3}{2^4} + \frac{4}{2^5} + - - -}_{\hookrightarrow \text{sum nearly 1.}} \right\}$$

$$= O(n \times x)$$

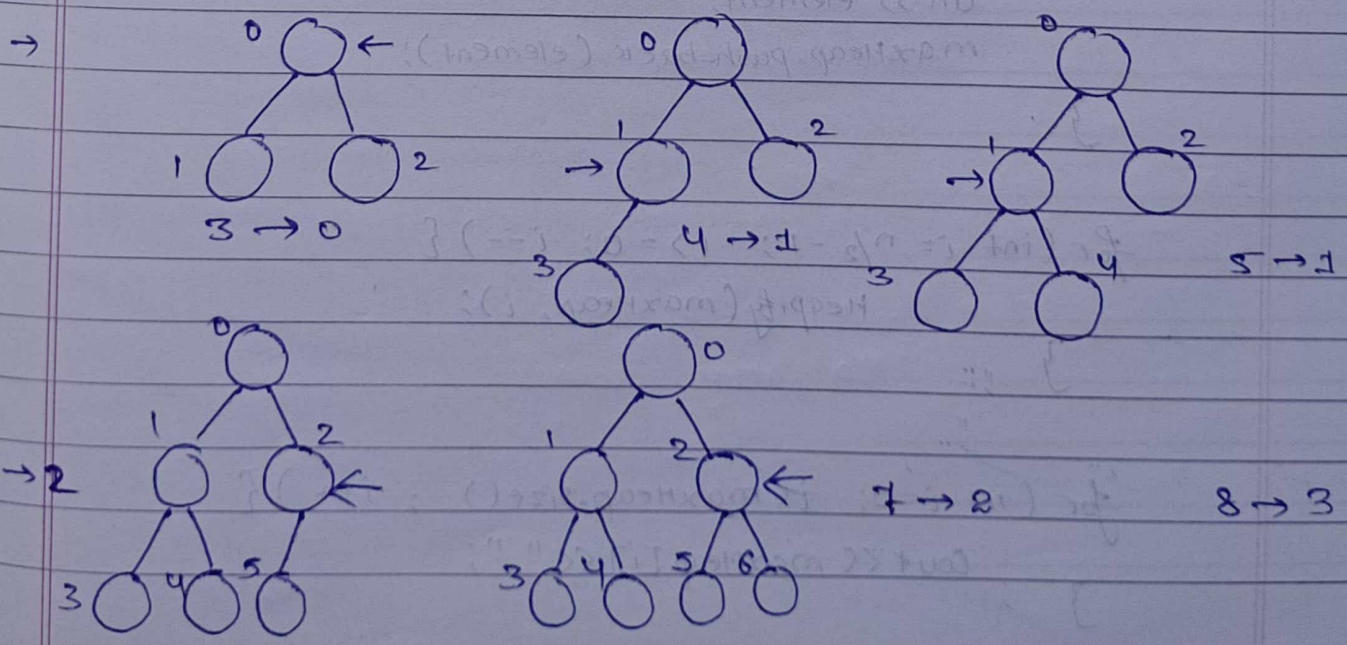
↳ Constant

$$O(n)$$

↳ Time Complexity

Space Complexity : $O(\log n)$.

How to solve : How to find Starting Node.



3 → 0 => 3/2 = 1 - 1 = 0

4 → 1 => 4/2 = 2 - 1 = 1

5 → 1 => 5/2 = 2 - 1 = 1

6 → 2 => 6/2 = 3 - 1 = 2

7 → 2 => 7/2 = 3 - 1 = 2

8 → 3 => 8/2 = 4 - 1 = 3

→
```
for (int i= size/2 -1 ;   i>=0; i--) {
        Heapify (maxHeap, i);
    }
```

* Code:

```
#include <iostream>
using namespace std;

int main () {
    vector <int> maxHeap;
    int n, element;
    cin >> n;
    for (int i=0; i<n; i++) {
        cin >> element;
        maxHeap.push_back (element);
    }

    for (int i= n/2 -1; i>=0; i--) {
        Heapify (maxHeap, i);
    }

    for (int i=0; i< maxHeap.size() ; i++ ) {
        cout << maxHeap [i] << " ";
    }
```

```
    return 0;
}

Void Heapify (vector<int> &maxHeap, int index) {
        int largest = index;
        int left = 2 * index + 1;
        int right = 2 * index + 2;
        int size = maxHeap.size();

    if (left < size && maxHeap[left] > maxHeap[largest])
        largest = left;

    if (right < size && maxHeap[right] > maxHeap[largest])
        largest = right;

    if (largest != index) {
        Swap (maxHeap[largest], maxHeap[index]);
        Heapify (maxHeap, largest);
    }
    return ;
}
```