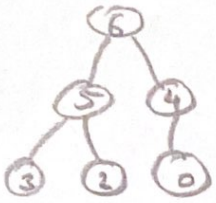
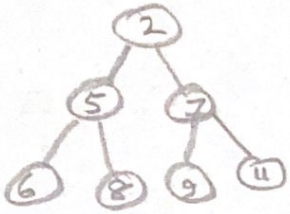
HEAP TREE

- It is a complete binary tree



Max heap tree

A[parent] ≥ A[child]

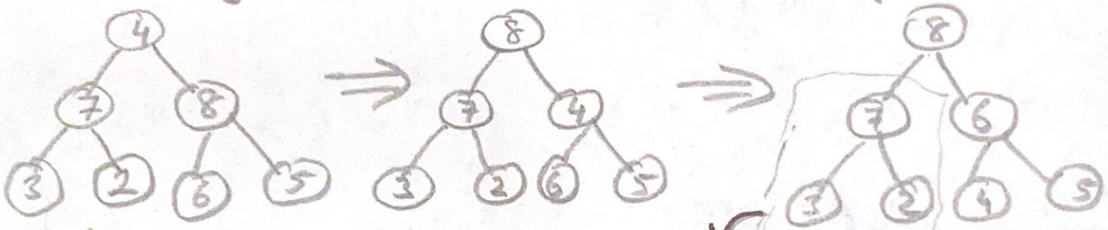


Min heap tree

A[parent] ≤ A[child]

- Left child may be greater than right child

Heapify: If the heap tree is wrong, switch the parent with max child (for max heap). And if we do this as recursive "switched side subtree" eventually will be okay for heap order.



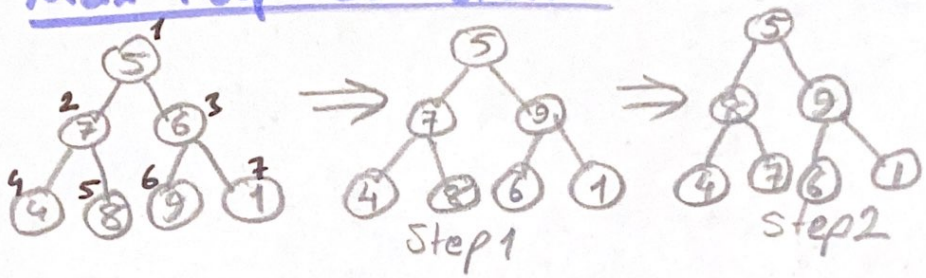
if there was a mistake here, we couldn't fix it only with heapify. Because it looks only switched subtree

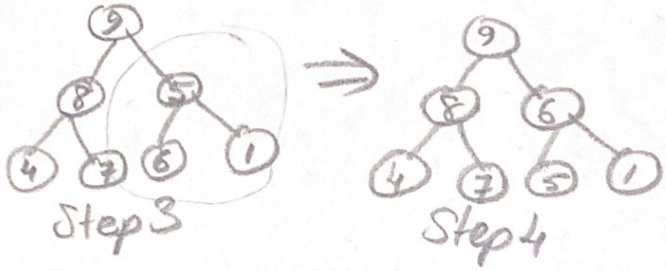
Complexity of heapify → O(log N) !!

- If we wanted to control whether it is heap tree or not, it would be enough N/2 to 1 index since more than N/2 are leaf nodes.

- To find the min element in max-heap tree, we should check all leafs. It means N/2 to N

Max Heap Conversion



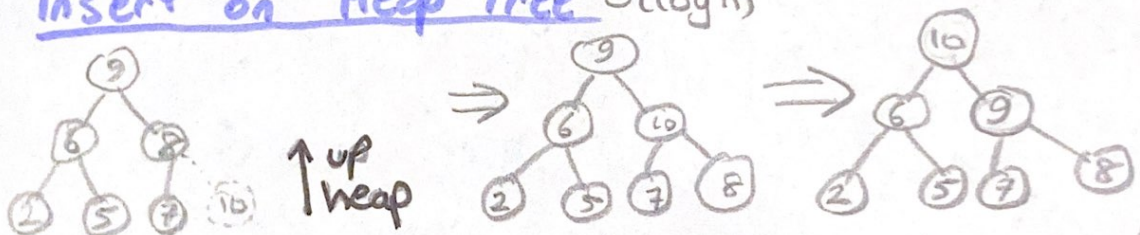


N=7 so we start at 3 (last parent) to heapify

```
void build_max_heap (int array[], int N){
    int i;
    for( i=N/2 ; i<=1 ; i--){
        max_heapify(array, i, N);
    }
}
```

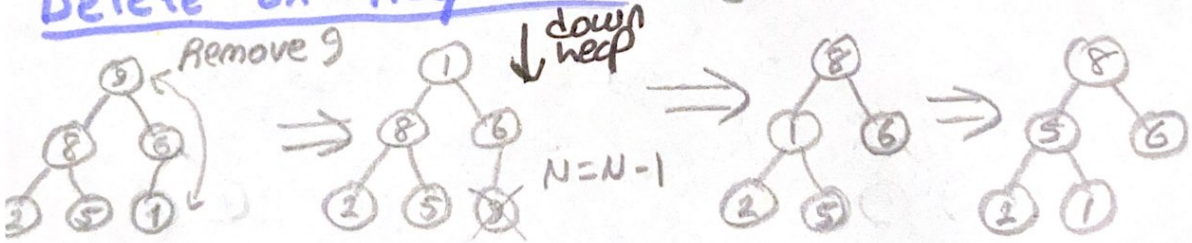
complexity of build_max_heap → O(n)

Insert on Heap Tree O(log n)



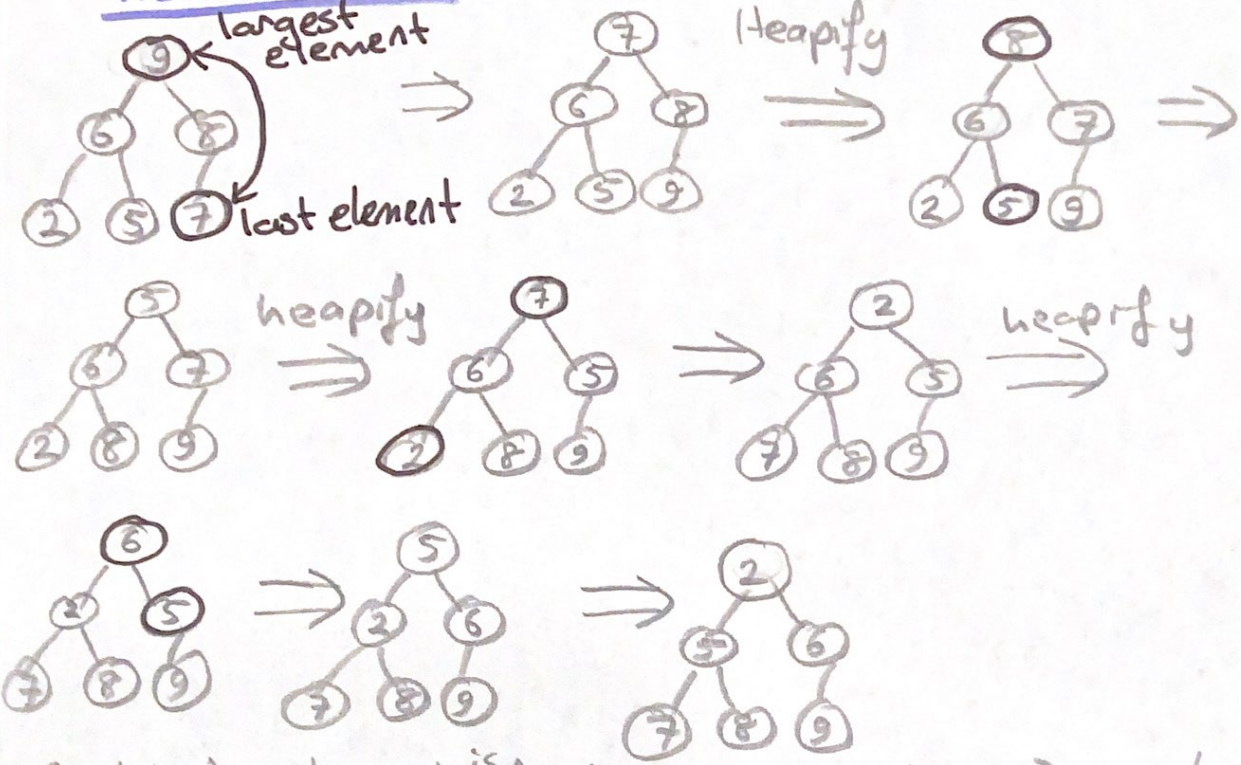
we add the value to the end and heapify...
bottom→up : we only check whether it is greater then parent

Delete on Heap Tree O(log n)



top→down : we check both child and max one is switched with parent
we switched last element with it and we reduce the size by one

HEAP SORT $O(N \log N)$



- Switched element is firstly N (last element) and every step it decreases. We always switch it with largest element

---

```
void maxHeapify(int array[], int i, int N){
    left = 2*i ; // since index starts with 1
    right = 2*i + 1;
    if(left ≤ N && array[left] > array[i])
        largest = left;
    else
        largest = i;
    if(right ≤ N && array[right] > array[largest])
        largest = right;
    if(largest != i){
        swap(array[i], array[largest]);
        maxHeapify(array, largest, N);
    }
}
```