Heaps

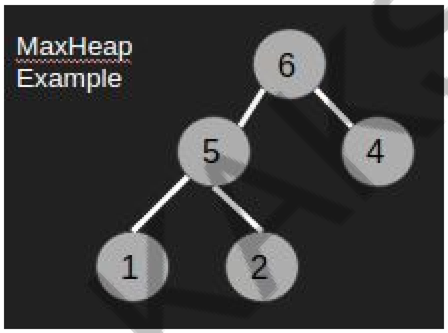
- Heaps are binary tree based data structures.
- Heaps are not necessarily BST (Binary Search Tree)

Heaps are of two types

1. MaxHeap
2. MinHeap

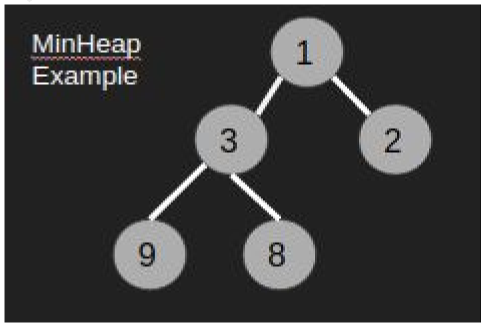
MaxHeap

MaxHeap is a heap in which element present at any node is greater than all the elements present in its subtree.



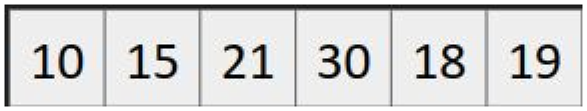
MinHeap

MinHeap is a heap in which element present at any node is smaller than all the elements present in its subtree.



Converting Array into a MaxHeap

Given an array,



10	15	21	30	18	19

Steps

1. Iterate over each element and **insert** it into the MaxHeap.

Now we will see how to insert elements into a MaxHeap

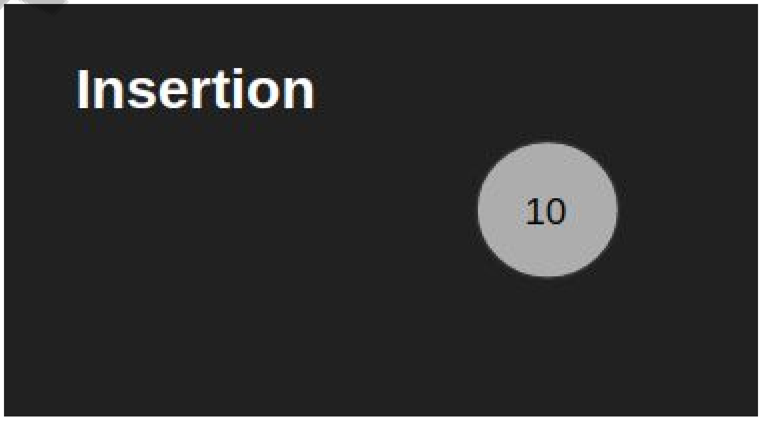
<u>Insertion into a MaxHeap</u>

Initially there were no elements in the MaxHeap, so make a newNode in the MaxHeap. Then for each incoming element, connect it to its position and then heapify.

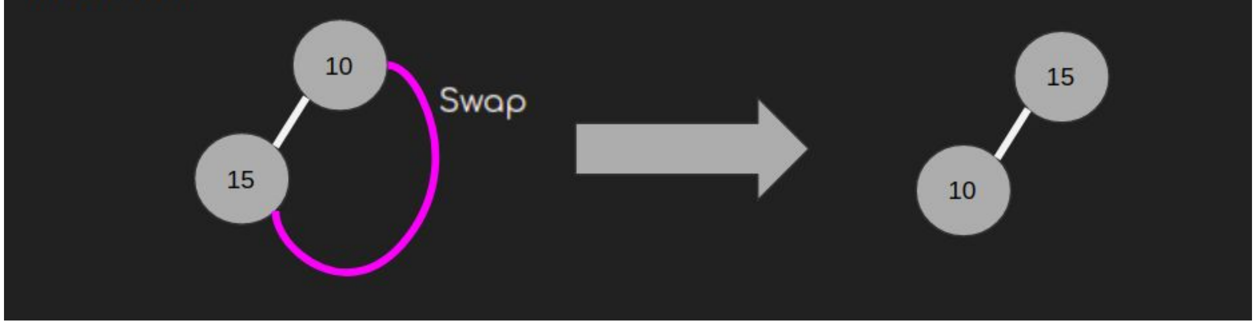
Heapification: Compare the currently inserted element with its parent, there will be 2 cases

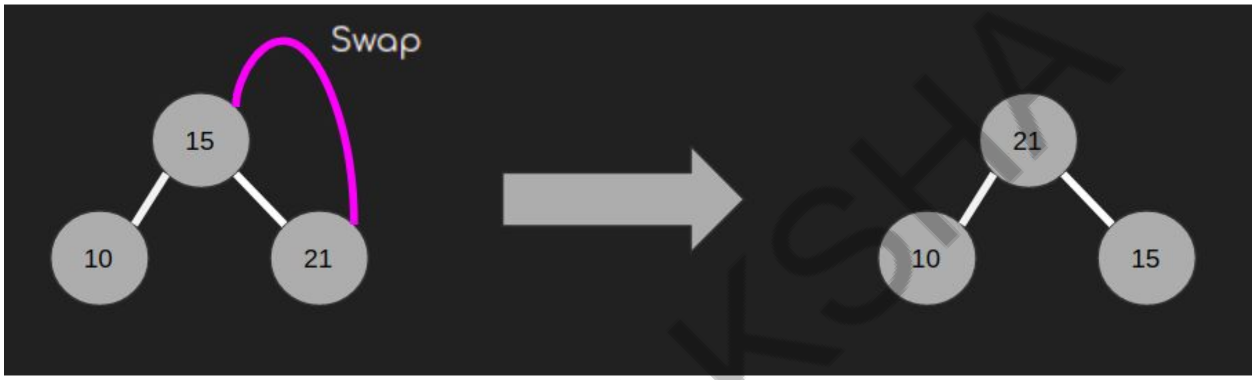
1. Current element > Parent element : Swap the current element and the parent element.
2. Current element <= Parent element: Keep it as it is.

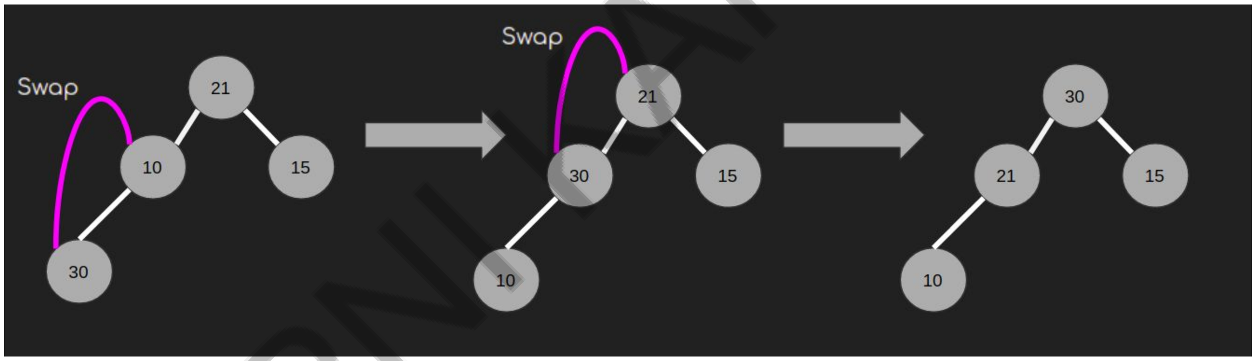
Iterations(insertions)

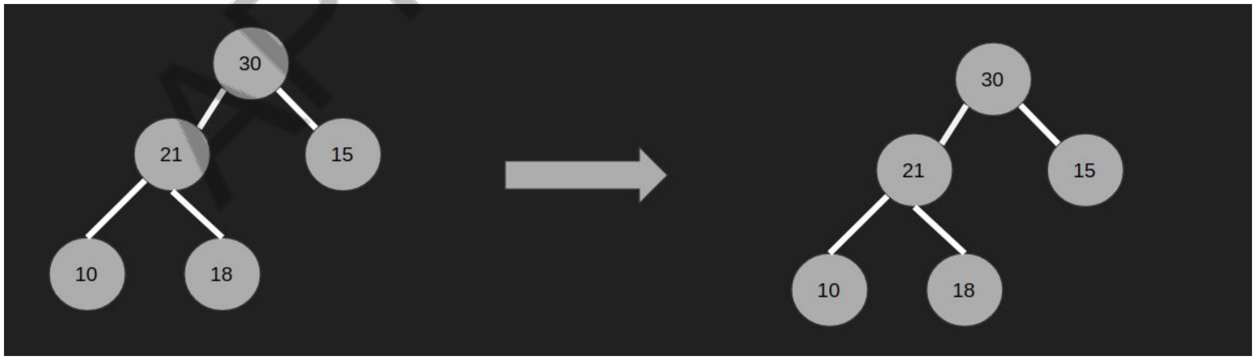


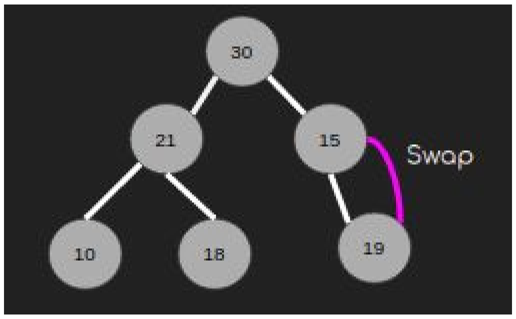
Insertion

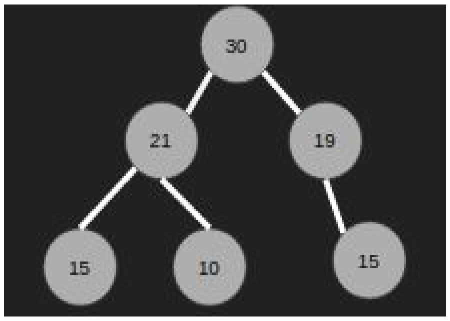

10
15
Swap
15
10


Swap
15
10
21
21
10
15


Swap
21
10
15
30
Swap
21
30
15
10
30
21
15
10

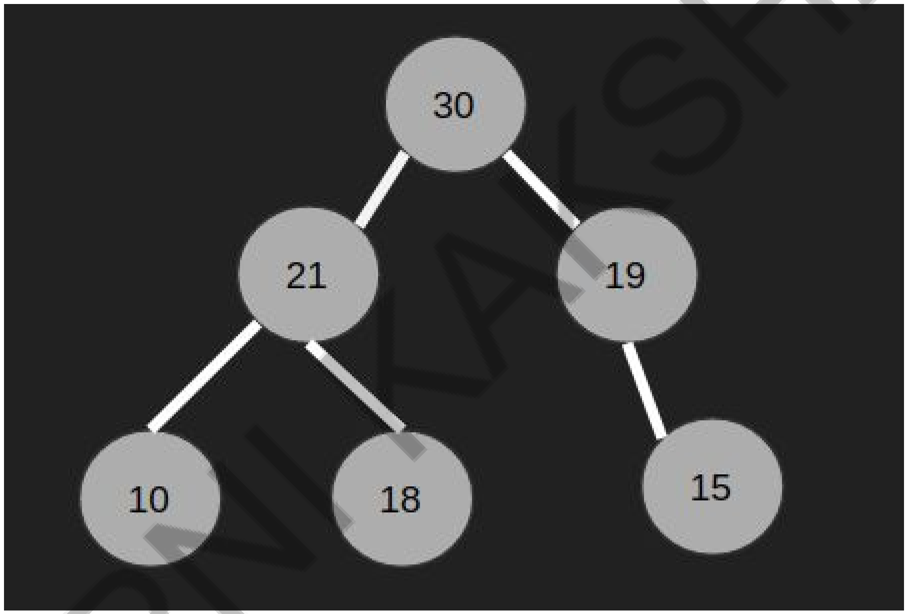

30
21
15
10
18
30
21
15
10
18



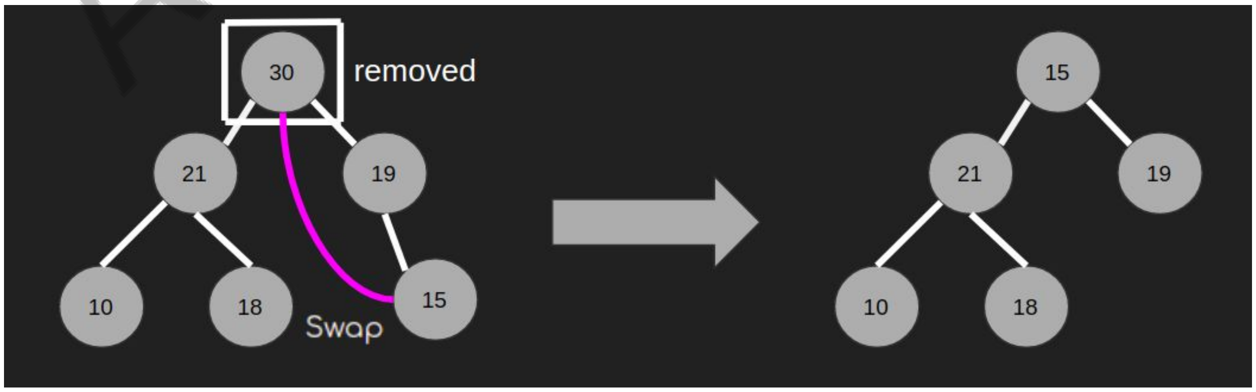


Popping Elements from the MaxHeap

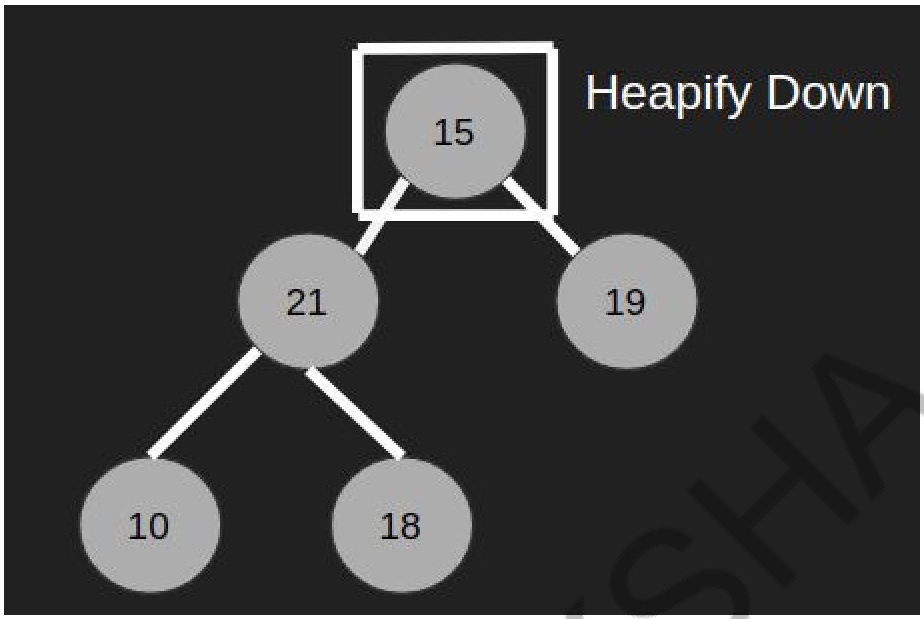
1. Given the MaxHeap



2. Remove the top element and bring the last element to the position of top element.



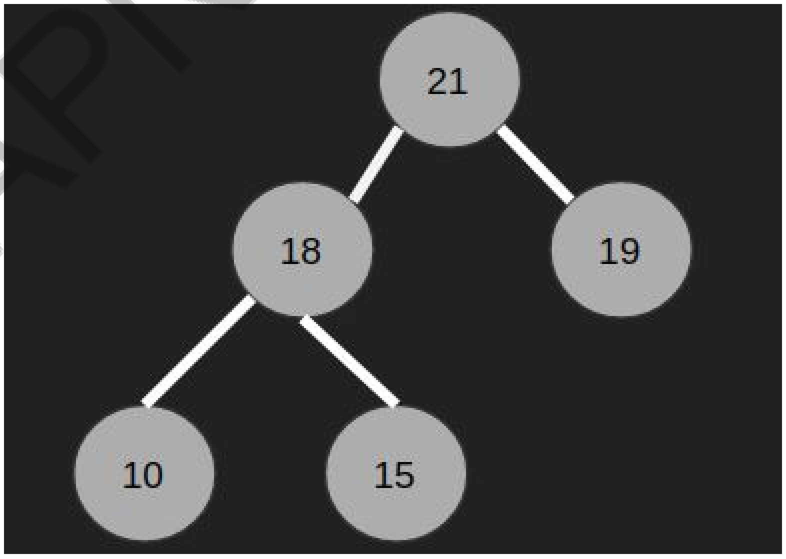
3. Apply heapify down on the top element.



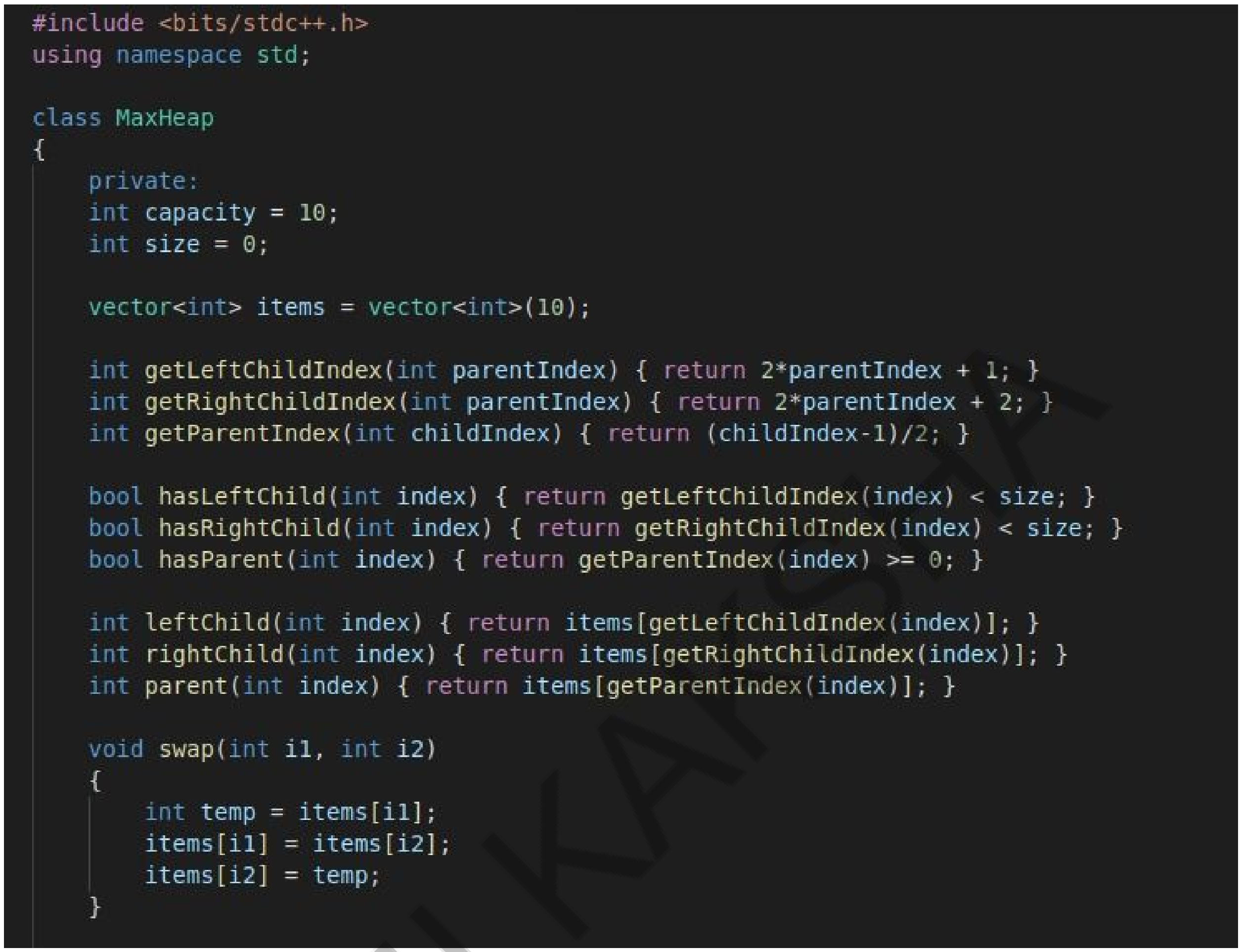
   a. Heapify Down: In case of MaxHeap, compare the current element with the greater child, there will be 2 cases
      i. Current element < Greater child : Swap current element with greater child.
      ii. Current element >= Greater child: Keep the heap as it is.

   Do this step until we reach our (ii) case.

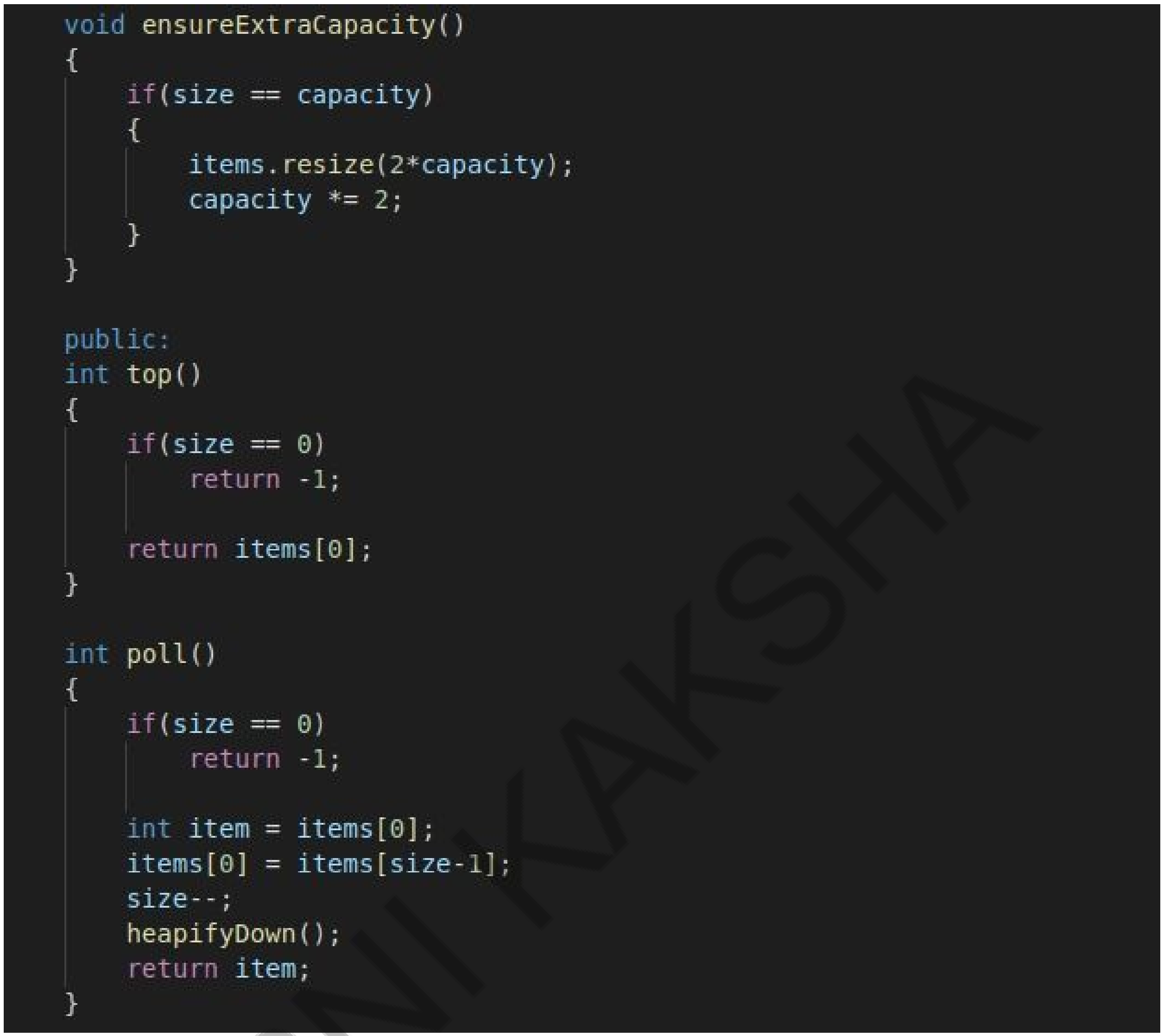
4. After performing step 3, we will get our MaxHeap after popping the element.



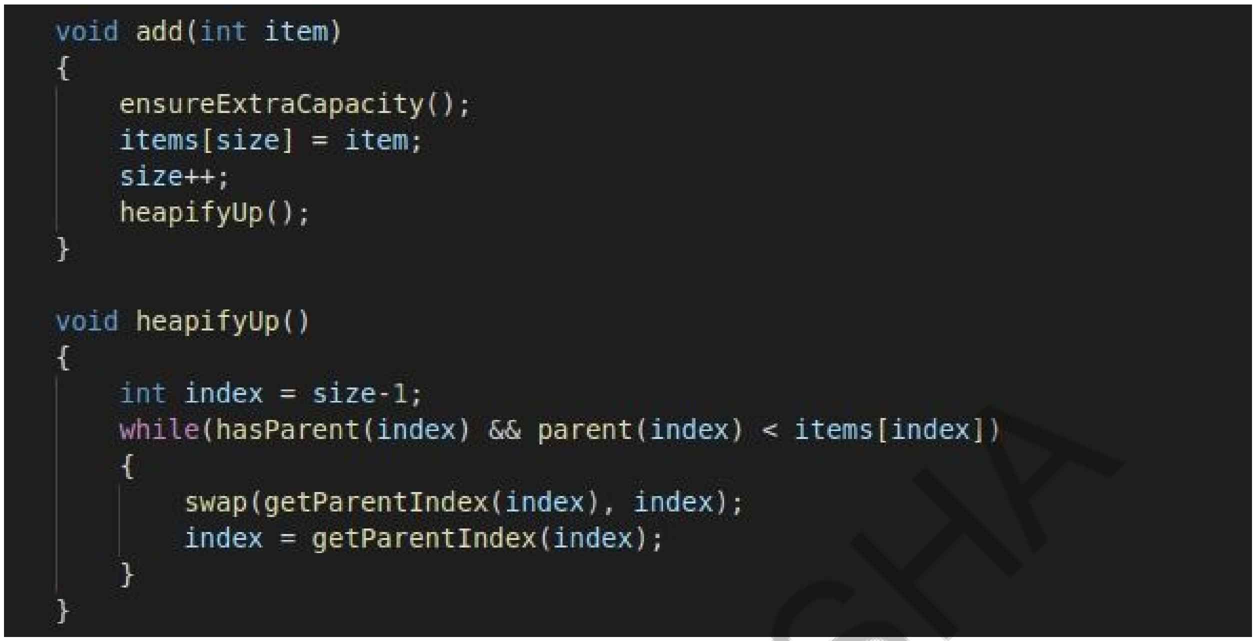
Code



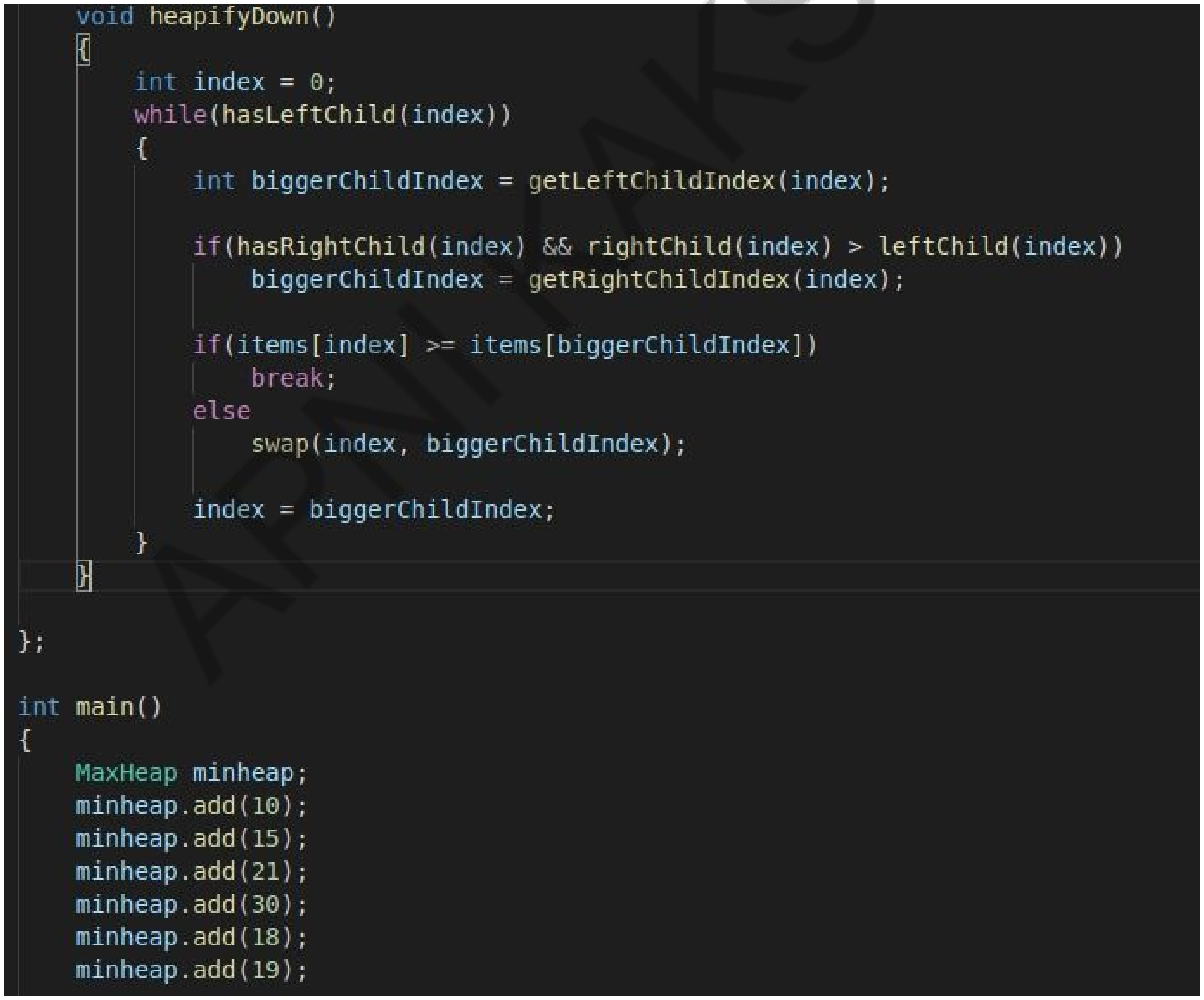
```cpp
#include <bits/stdc++.h>
using namespace std;

class MaxHeap
{
    private:
    int capacity = 10;
    int size = 0;

    vector<int> items = vector<int>(10);

    int getLeftChildIndex(int parentIndex) { return 2*parentIndex + 1; }
    int getRightChildIndex(int parentIndex) { return 2*parentIndex + 2; }
    int getParentIndex(int childIndex) { return (childIndex-1)/2; }

    bool hasLeftChild(int index) { return getLeftChildIndex(index) < size; }
    bool hasRightChild(int index) { return getRightChildIndex(index) < size; }
    bool hasParent(int index) { return getParentIndex(index) >= 0; }

    int leftChild(int index) { return items[getLeftChildIndex(index)]; }
    int rightChild(int index) { return items[getRightChildIndex(index)]; }
    int parent(int index) { return items[getParentIndex(index)]; }

    void swap(int i1, int i2)
    {
        int temp = items[i1];
        items[i1] = items[i2];
        items[i2] = temp;
    }
```



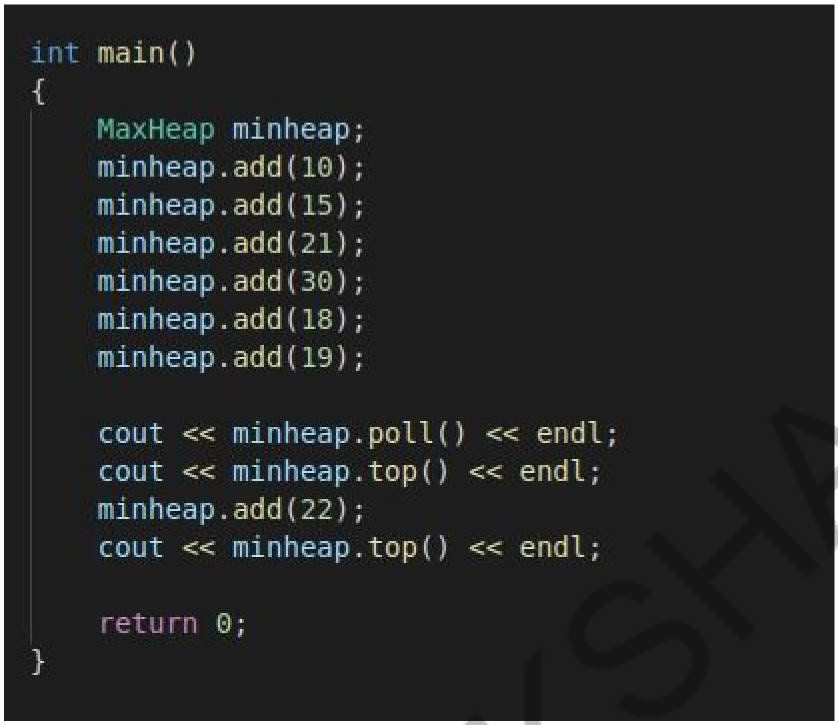
```
void ensureExtraCapacity()
{
    if(size == capacity)
    {
        items.resize(2*capacity);
        capacity *= 2;
    }
}

public:
int top()
{
    if(size == 0)
        return -1;

    return items[0];
}

int poll()
{
    if(size == 0)
        return -1;

    int item = items[0];
    items[0] = items[size-1];
    size--;
    heapifyDown();
    return item;
}
```



```
    void add(int item)
    {
        ensureExtraCapacity();
        items[size] = item;
        size++;
        heapifyUp();
    }

    void heapifyUp()
    {
        int index = size-1;
        while(hasParent(index) && parent(index) < items[index])
        {
            swap(getParentIndex(index), index);
            index = getParentIndex(index);
        }
    }
```



```
    void heapifyDown()
    {
        int index = 0;
        while(hasLeftChild(index))
        {
            int biggerChildIndex = getLeftChildIndex(index);

            if(hasRightChild(index) && rightChild(index) > leftChild(index))
                biggerChildIndex = getRightChildIndex(index);

            if(items[index] >= items[biggerChildIndex])
                break;
            else
                swap(index, biggerChildIndex);

            index = biggerChildIndex;
        }
    }

};

int main()
{
    MaxHeap minheap;
    minheap.add(10);
    minheap.add(15);
    minheap.add(21);
    minheap.add(30);
    minheap.add(18);
    minheap.add(19);
```



```cpp
int main()
{
    MaxHeap minheap;
    minheap.add(10);
    minheap.add(15);
    minheap.add(21);
    minheap.add(30);
    minheap.add(18);
    minheap.add(19);

    cout << minheap.poll() << endl;
    cout << minheap.top() << endl;
    minheap.add(22);
    cout << minheap.top() << endl;

    return 0;
}
```