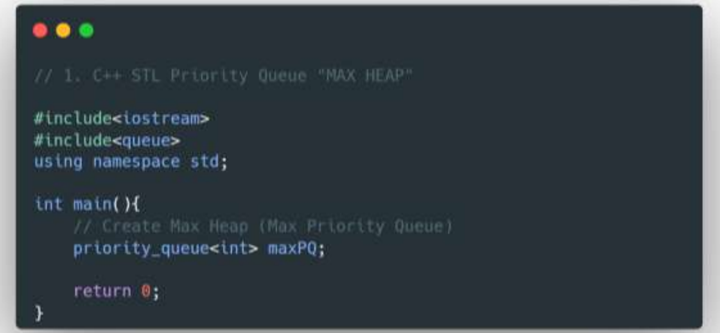
17/12/2023

HEAP CLASS - 2

1. C++ STL Priority Queue (Max Heap)

Create Max Priority Queue --> **Max Heap**



```
// 1. C++ STL Priority Queue "MAX HEAP"

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

int main(){
    // Create Max Heap (Max Priority Queue)
    priority_queue<int> maxPQ;

    return 0;
}
```

Create Max Heap (Max Priority Queue)

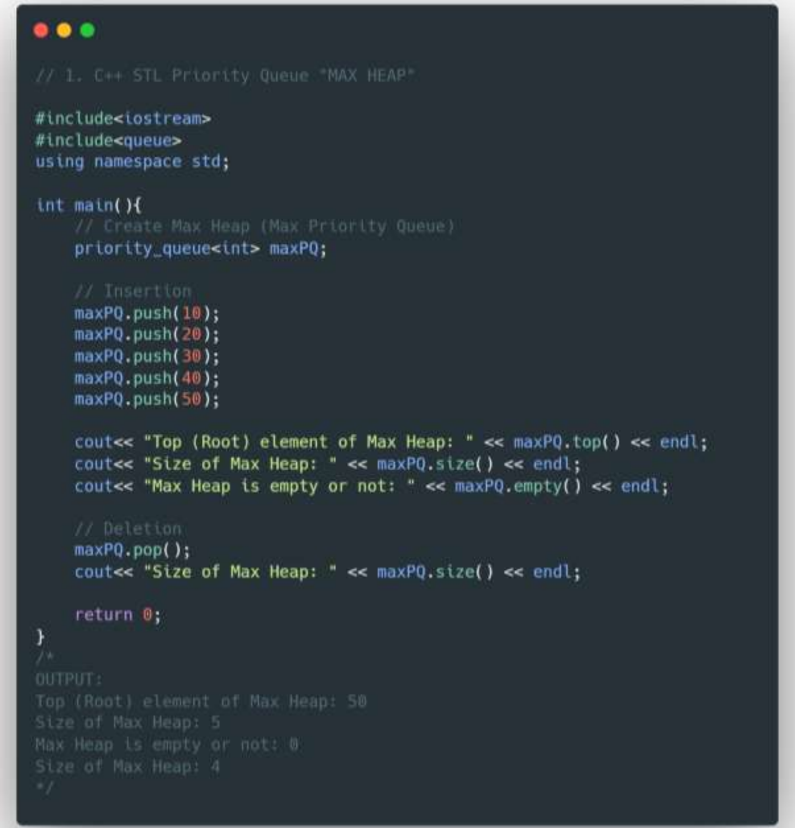
Example:

priority_queue <int> maxPQ;

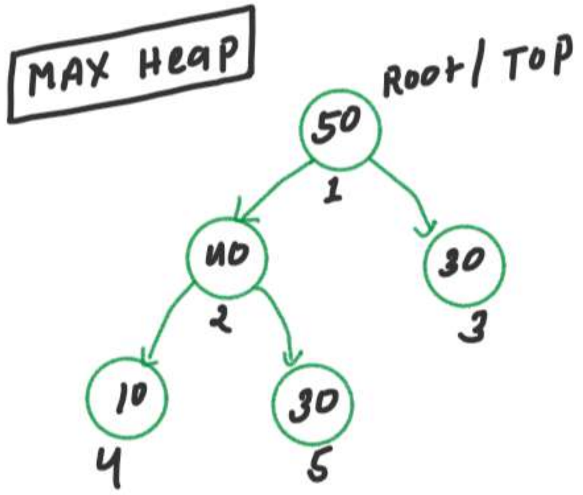
Explanation:

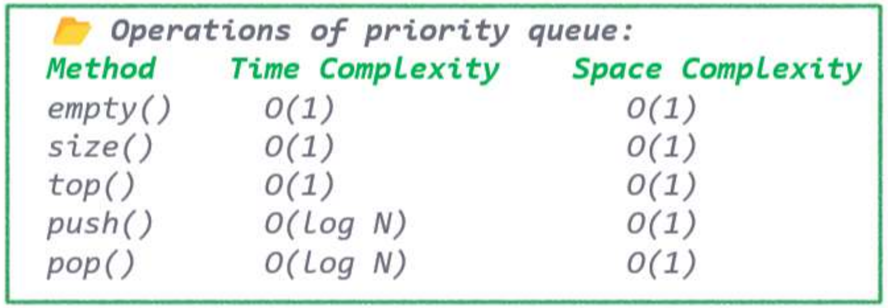
priority_queue< data type> pq_name;

1. Data Type: This is the data type of the elements that will be stored in the priority queue.
In this case, priority queue will store integers



```cpp
// 1. C++ STL Priority Queue "MAX HEAP"

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

int main(){
    // Create Max Heap (Max Priority Queue)
    priority_queue<int> maxPQ;

    // Insertion
    maxPQ.push(10);
    maxPQ.push(20);
    maxPQ.push(30);
    maxPQ.push(40);
    maxPQ.push(50);

    cout<< "Top (Root) element of Max Heap: " << maxPQ.top() << endl;
    cout<< "Size of Max Heap: " << maxPQ.size() << endl;
    cout<< "Max Heap is empty or not: " << maxPQ.empty() << endl;

    // Deletion
    maxPQ.pop();
    cout<< "Size of Max Heap: " << maxPQ.size() << endl;

    return 0;
}
/*
OUTPUT:
Top (Root) element of Max Heap: 50
Size of Max Heap: 5
Max Heap is empty or not: 0
Size of Max Heap: 4
*/
```

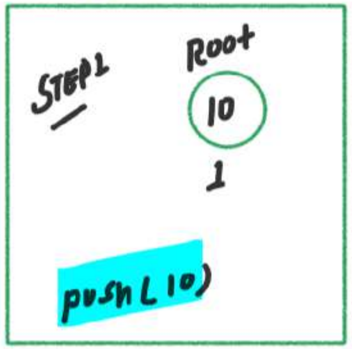


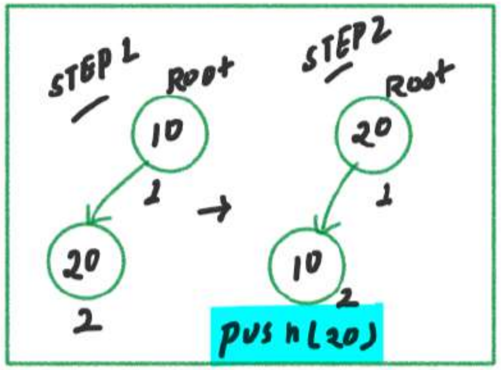


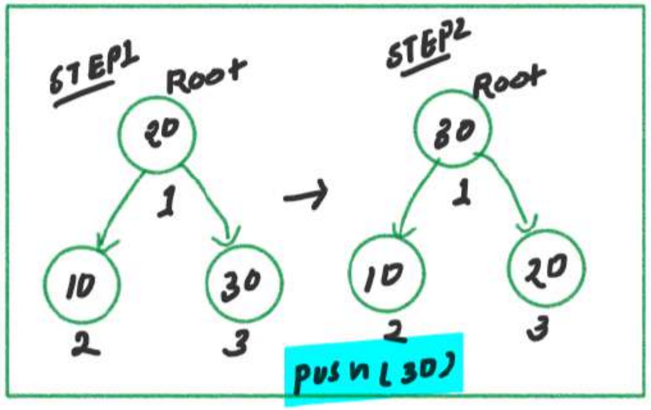
📂 Operations of priority queue:

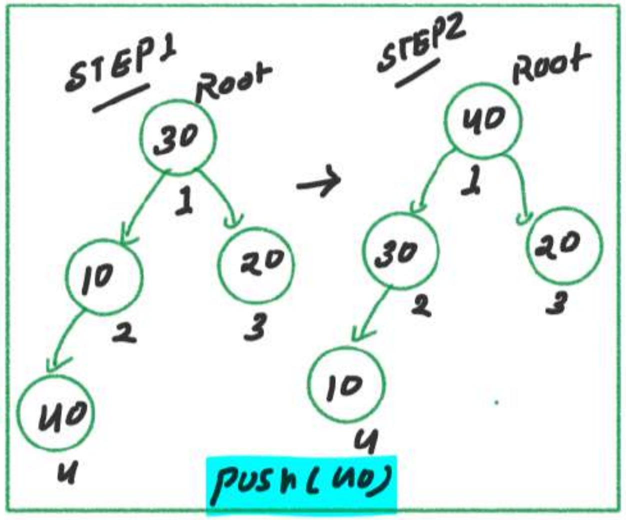
Method	Time Complexity	Space Complexity
empty()	O(1)	O(1)
size()	O(1)	O(1)
top()	O(1)	O(1)
push()	O(Log N)	O(1)
pop()	O(Log N)	O(1)

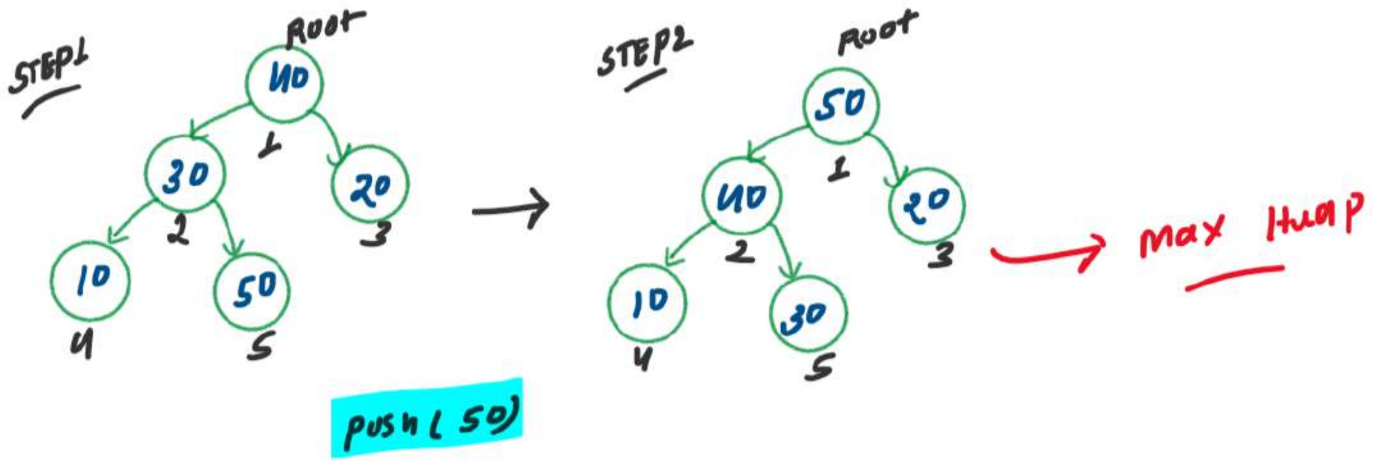
DRY RUN


STEP1
Root
10
1
push(10)


STEP 1
Root
10
1
20
2
STEP 2
Root
20
1
10
2
push(20)

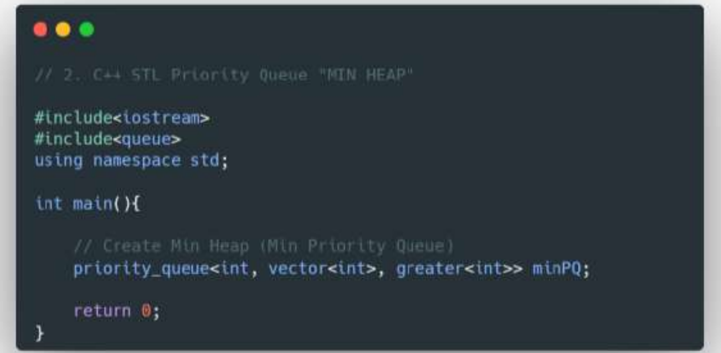

STEP1
Root
20
1
10
2
30
3
STEP2
Root
30
1
10
2
20
3
push(30)


STEP1
Root
30
1
10
2
20
3
40
4
STEP2
Root
40
1
30
2
20
3
10
4
push(40)


STEP1
Root
40
1
30
2
20
3
10
4
50
5
push(50)
STEP2
Root
50
1
40
2
20
3
10
4
30
5
Max Heap

2. C++ STL Priority Queue (Min Heap)

*Create Min Priority Queue --> **Min Heap***



```
// 2. C++ STL Priority Queue "MIN HEAP"

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

int main(){

    // Create Min Heap (Min Priority Queue)
    priority_queue<int, vector<int>, greater<int>> minPQ;

    return 0;
}
```

Create Min Heap (Min Priority Queue)

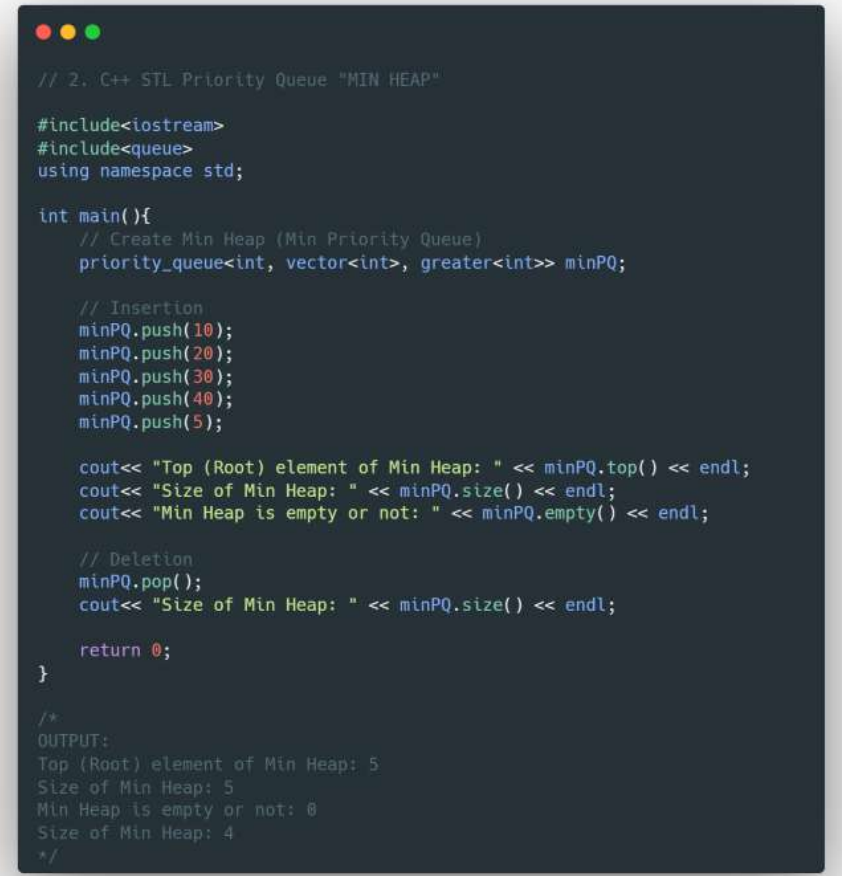
Explanation:

Priority_queue< 1, 2, 3> PQ_name;

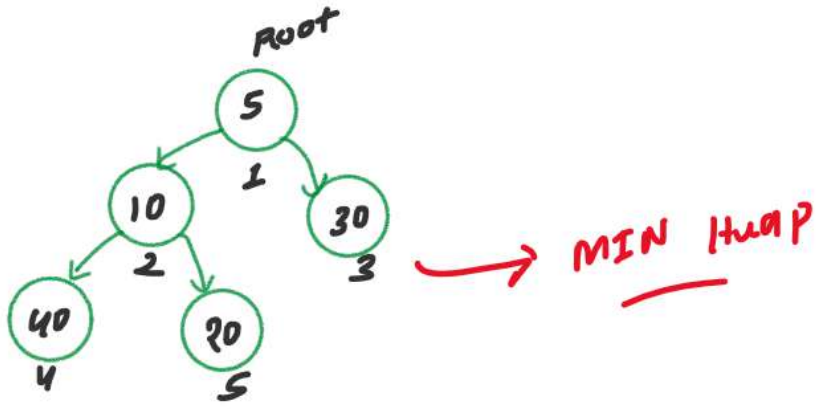
1. Data Type: *This is the data type of the elements that will be stored in the priority queue.*
In this case, priority queue will store integers

2. Container Type: *This is the container type used to store the elements internally.*
In this case, the priority queue is implemented using this vector to store its elements.

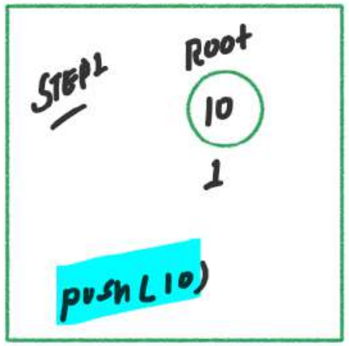
3. Comparator Function: *greater<int> makes the max priority queue act as a min priority queue*

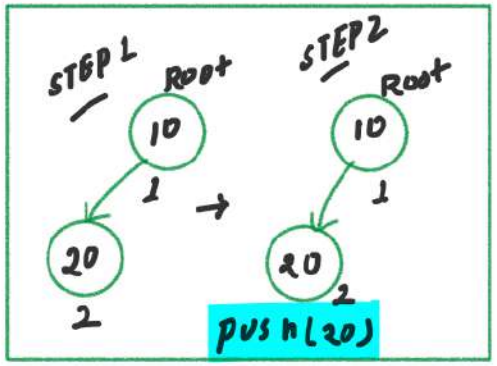


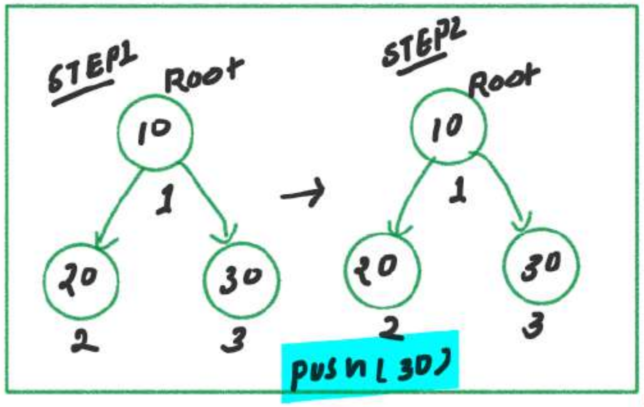
```cpp
// 2. C++ STL Priority Queue "MIN HEAP"

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

int main(){
    // Create Min Heap (Min Priority Queue)
    priority_queue<int, vector<int>, greater<int>> minPQ;

    // Insertion
    minPQ.push(10);
    minPQ.push(20);
    minPQ.push(30);
    minPQ.push(40);
    minPQ.push(5);

    cout<< "Top (Root) element of Min Heap: " << minPQ.top() << endl;
    cout<< "Size of Min Heap: " << minPQ.size() << endl;
    cout<< "Min Heap is empty or not: " << minPQ.empty() << endl;

    // Deletion
    minPQ.pop();
    cout<< "Size of Min Heap: " << minPQ.size() << endl;

    return 0;
}

/*
OUTPUT:
Top (Root) element of Min Heap: 5
Size of Min Heap: 5
Min Heap is empty or not: 0
Size of Min Heap: 4
*/
```

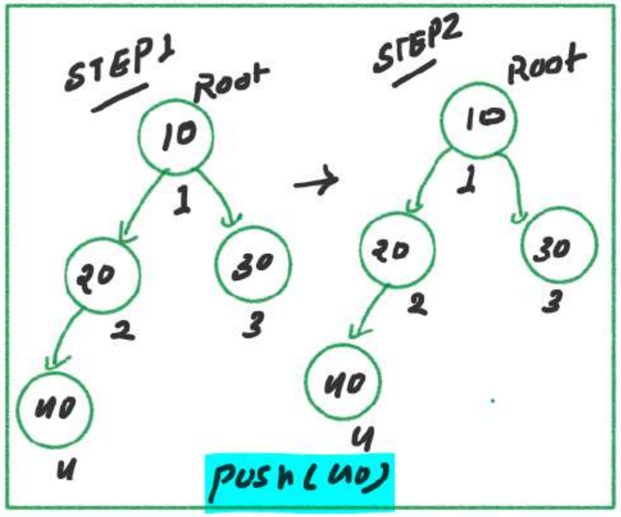


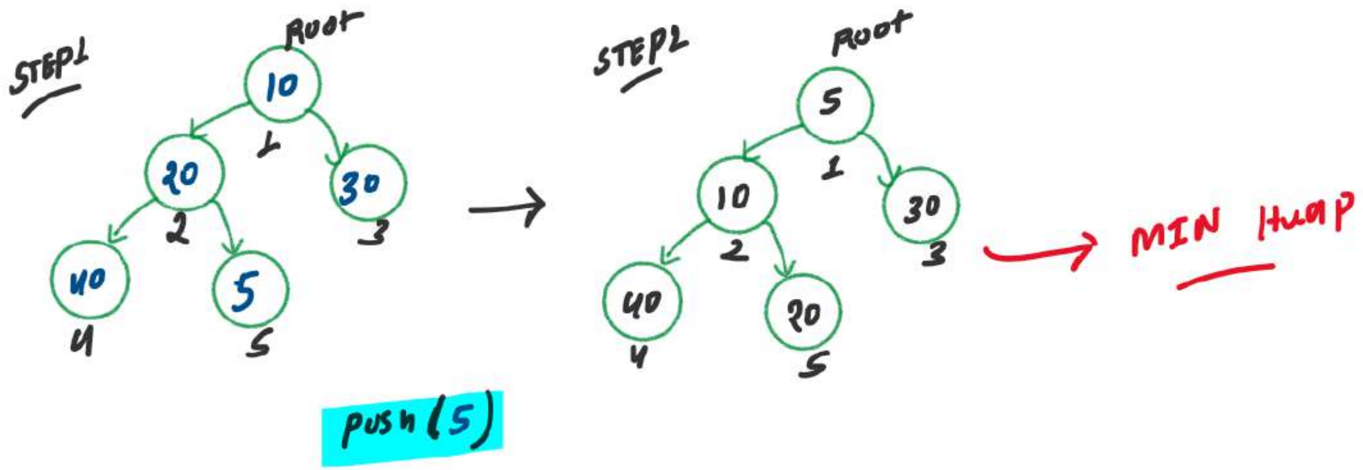
DRY RUN


STEP1
Root
10
1
push(10)


STEP1
Root
10
1
20
2
→
STEP2
Root
10
1
20
2
push(20)


STEP1
Root
10
1
20
2
30
3
→
STEP2
Root
10
1
20
2
30
3
push(30)


STEP1
Root
10
1
20
2
30
3
40
4
→
STEP2
Root
10
1
20
2
30
3
40
4
push(40)


STEP1
Root
10
1
20
2
30
3
40
4
5
5
push(5)
→
STEP2
Root
5
1
10
2
30
3
40
4
20
5
→ MIN Heap

3. Find Kth Smallest Element in an Array using Max Heap (GFG)

EX1

Input K and Array

arr

3	7	4	5	8	6	9
0	1	2	3	4	5	6

↳ output $\boxed{6}$ when K = 4

↳ output $\boxed{5}$ when K = 3

Approach 1

SORT and print arr[K-1]

3	4	5	6	7	8	9
0	1	2	3	4	5	6

T.C. = O(N log N)

Approach 2 Using Min Heap

$\left\{ \begin{array}{l} T.C = O(N) \\ S.C. = O(N) \end{array} \right\}$

$\underbrace{\text{pop I} \quad \text{pop II} \quad \text{pop III}}_{\text{(K-1) delete element}} \quad \underbrace{\text{print VI}}_{\text{print Top from M-Heap}}$

Approach 3 using max heap

T.C. = O(N)

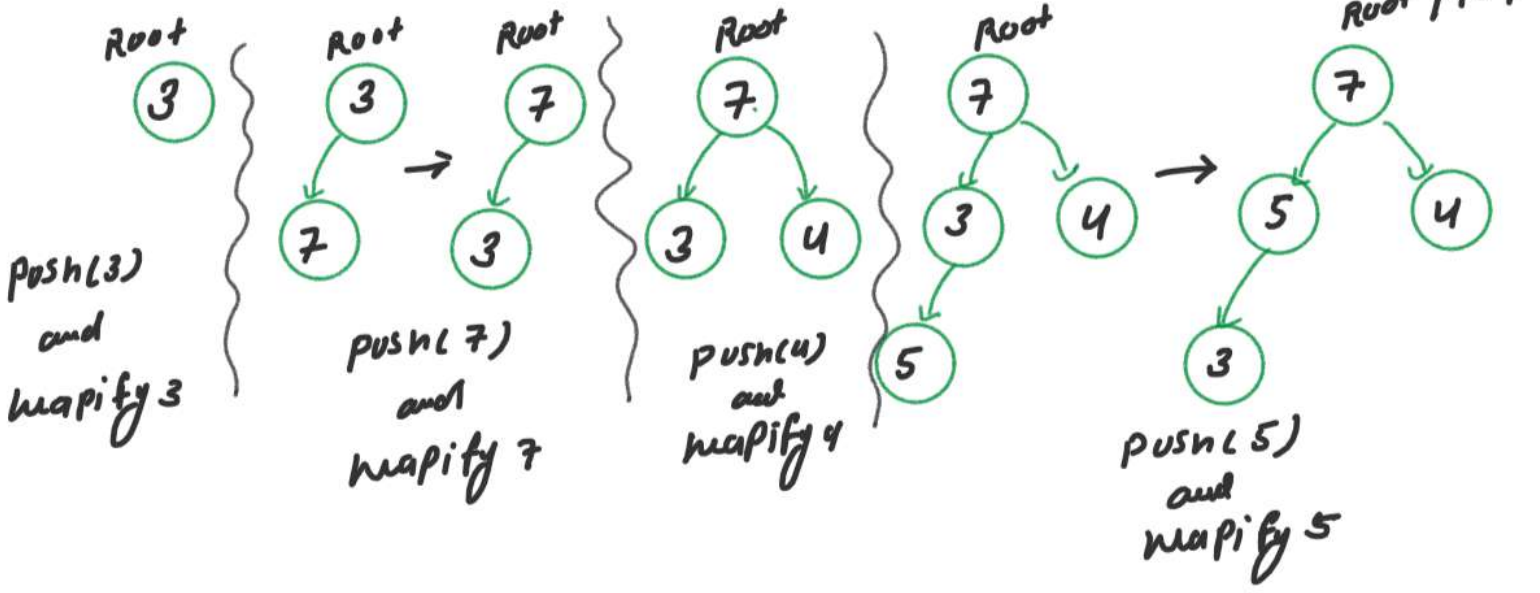
S.C. = O(K)

EX1

Input K = 4

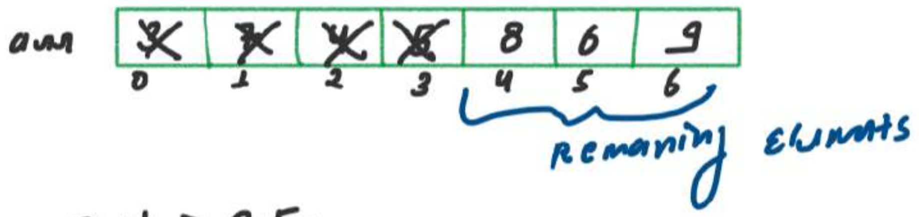
arr	3	7	4	5	8	6	9
	0	1	2	3	4	5	6

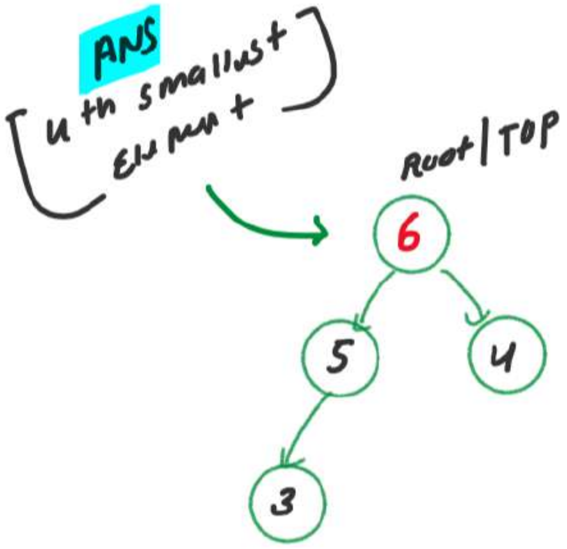
STEP1 K-size ka max heap create karlo



STEP 2 Replace the TOP element with Remaining elements when

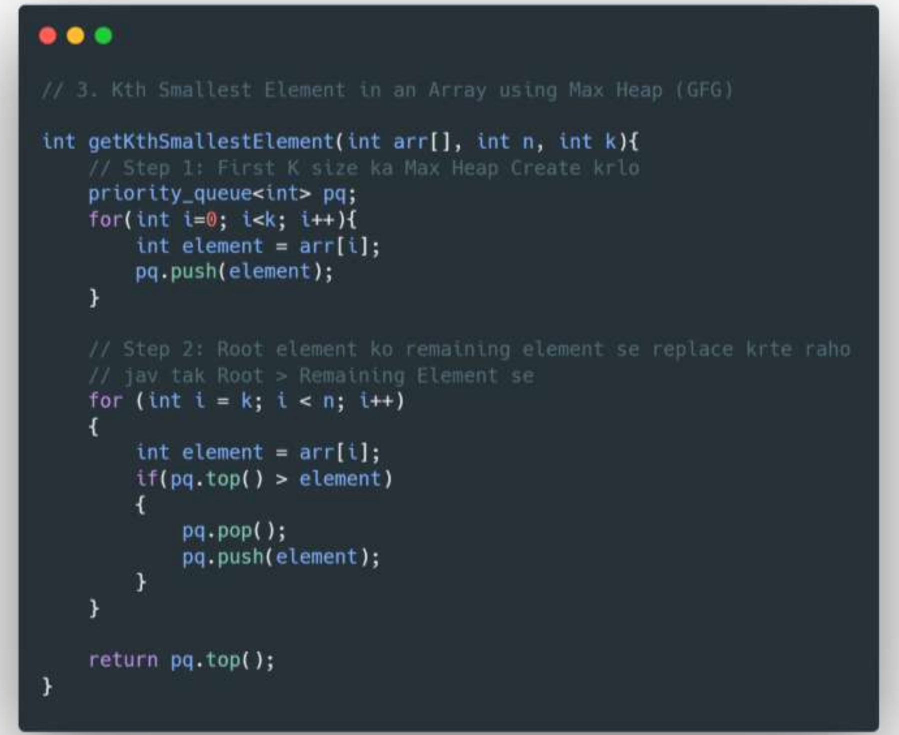
[Root > Remaining elements]





Root > R.E.				
7	>	8	✗	Root = 7
7	>	6	✓	Root = 6
6	>	9	✗	Root = 6

$\boxed{\text{output} = 6}$



```cpp
// 3. Kth Smallest Element in an Array using Max Heap (GFG)

int getKthSmallestElement(int arr[], int n, int k){
    // Step 1: First K size ka Max Heap Create krlo
    priority_queue<int> pq;
    for(int i=0; i<k; i++){
        int element = arr[i];
        pq.push(element);
    }

    // Step 2: Root element ko remaining element se replace krte raho
    // jav tak Root > Remaining Element se
    for (int i = k; i < n; i++)
    {
        int element = arr[i];
        if(pq.top() > element)
        {
            pq.pop();
            pq.push(element);
        }
    }

    return pq.top();
}
```

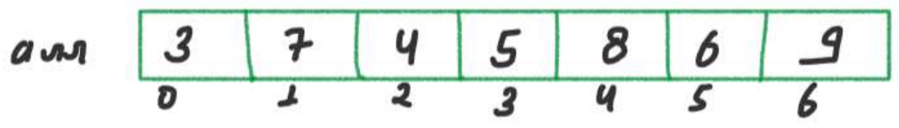
Time Complexity: *O(N),*
Where N is number of elements in array

Space Complexity: *O(K),*
Where K is number of nodes in heap

📂 4. Find Kth Greatest Element in an Array using Min Heap (GFG)

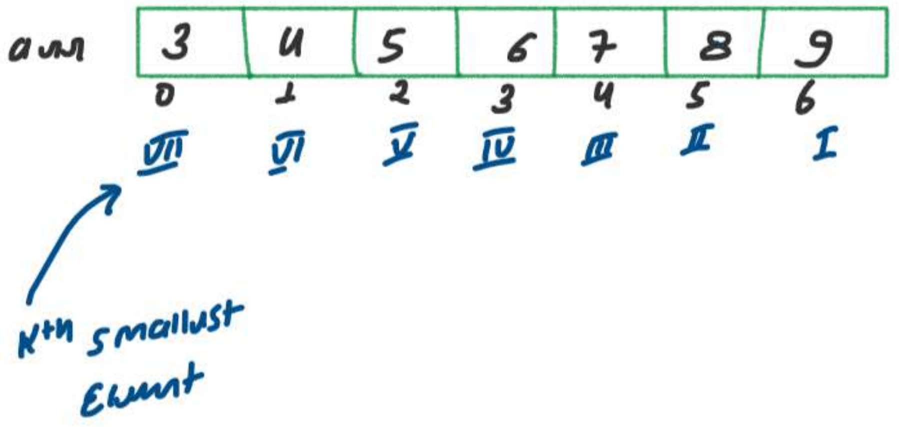
EX1

Input K and Array



↳ output $\boxed{6}$ when $K=4$

↳ output $\boxed{7}$ when $K=3$



DRY RUN

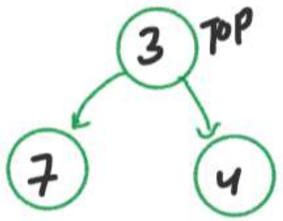
EX1

Input K=3

arr

3	7	4	5	8	6	9
0	1	2	3	4	5	6

STEP1 Create min heap of K-size

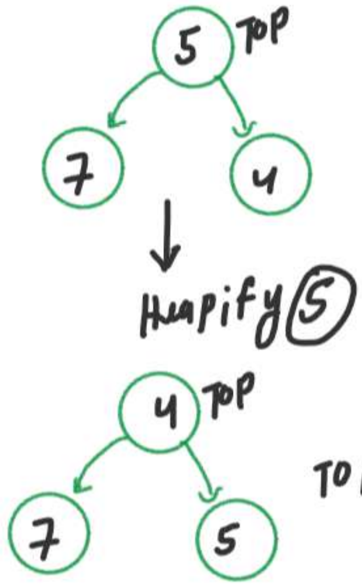


STEP 2 Replace TOP with remaining Elements when they are greater than Root

1

arr

~~3~~	~~7~~	~~4~~	5	8	6	9
0	1	2	3	4	5	6



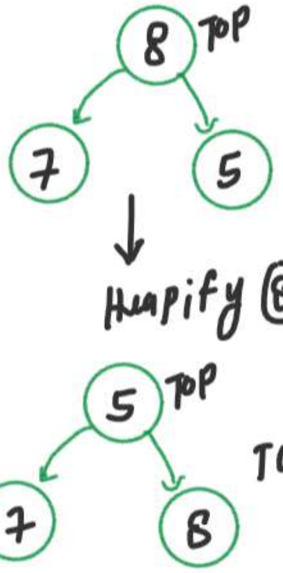
TOP < Elements

3 < 5 ✓ TOP = 5

TOP=4

② arr

0	1	2	3	4	5	6
~~2~~	~~3~~	~~4~~	~~5~~	8	6	9



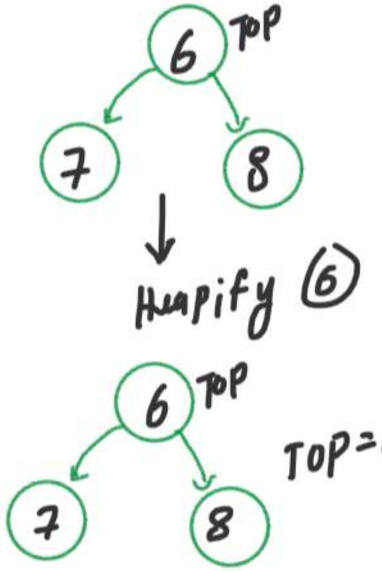
TOP = 5

$\underline{\text{TOP} < \text{Elements}}$

$4 < 8$ ✓ TOP = 8

③ arr

0	1	2	3	4	5	6
~~2~~	~~3~~	~~4~~	~~5~~	~~8~~	6	9

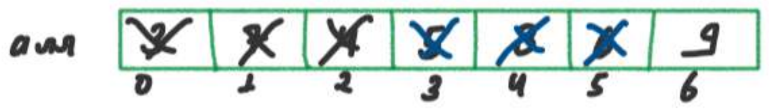


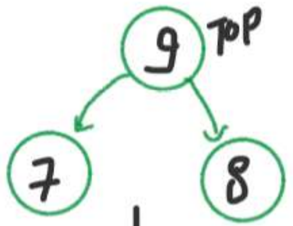
TOP = 6

$\underline{\text{TOP} < \text{Elements}}$

$5 < 6$ ✓ TOP = 6

$\boxed{4}$



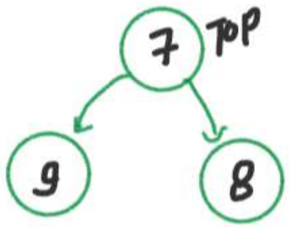


TOP < Elements

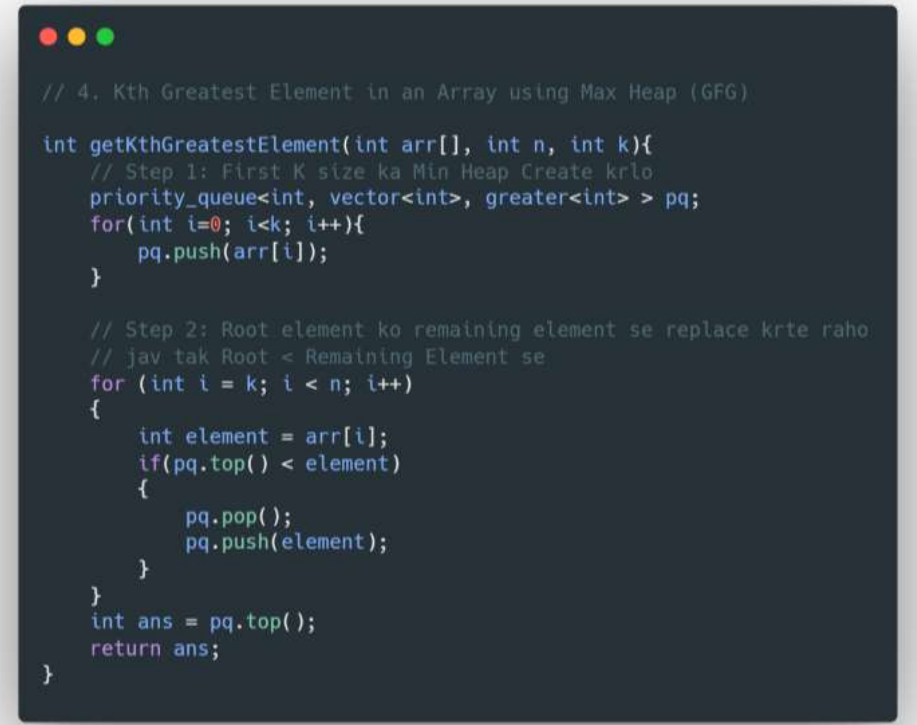
6 < 9 ✓ TOP = 9

↓

Heapify $\boxed{9}$



TOP = 7 $\hookrightarrow$ 3rd greatest Element is TOP = $\boxed{7}$



```cpp
// 4. Kth Greatest Element in an Array using Max Heap (GFG)

int getKthGreatestElement(int arr[], int n, int k){
    // Step 1: First K size ka Min Heap Create krlo
    priority_queue<int, vector<int>, greater<int> > pq;
    for(int i=0; i<k; i++){
        pq.push(arr[i]);
    }

    // Step 2: Root element ko remaining element se replace krte raho
    // jav tak Root < Remaining Element se
    for (int i = k; i < n; i++)
    {
        int element = arr[i];
        if(pq.top() < element)
        {
            pq.pop();
            pq.push(element);
        }
    }
    int ans = pq.top();
    return ans;
}
```

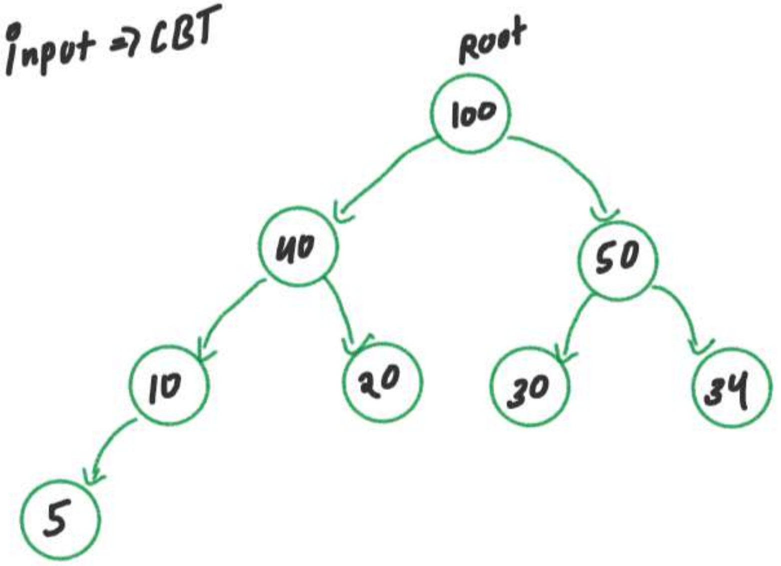
Time Complexity: *O(N),*
Where N is number of elements in array

Space Complexity: *O(K),*
Where K is number of nodes in heap

📂 5. Check if a given CBT is a Max Heap or Not? (GFG)

Ex 1

input => CBT

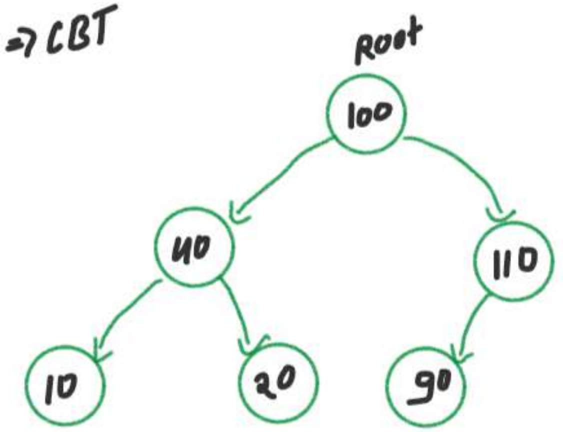


Output TRUE

A MAX Heap

Ex2

input => CBT

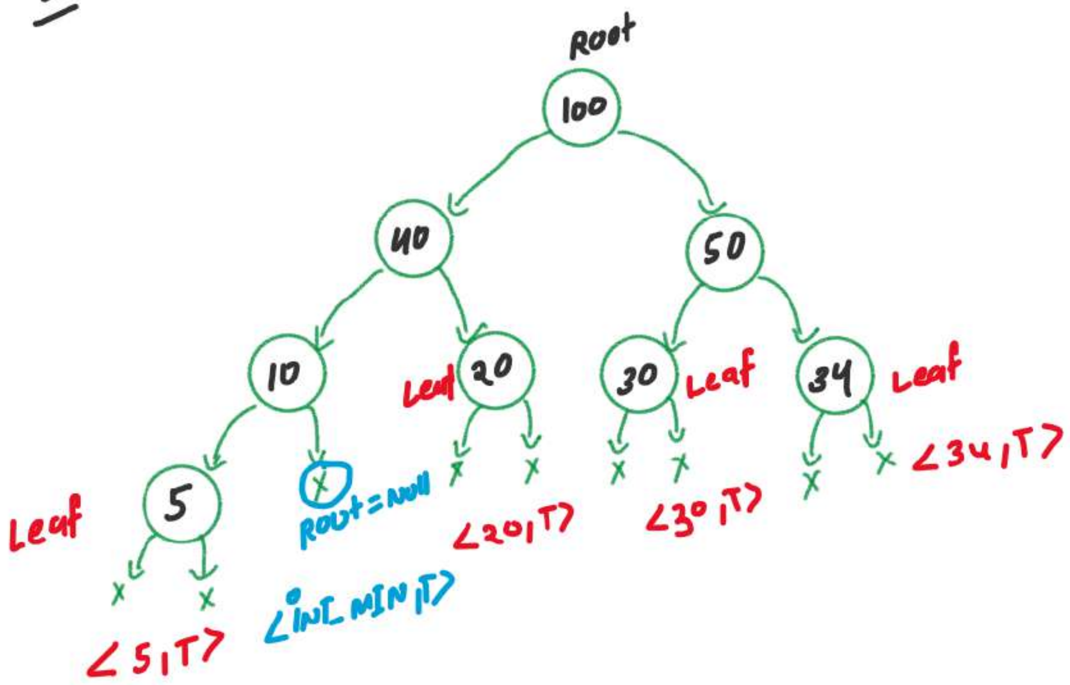


Output False

Not A max Heap

Logic Building

Ex 1



jab HUM Base case par Honge To Two value Return Karenge

< x , y >

Max value of sub Tree

isheap or NOT T/F

Base case 1 Root == Null

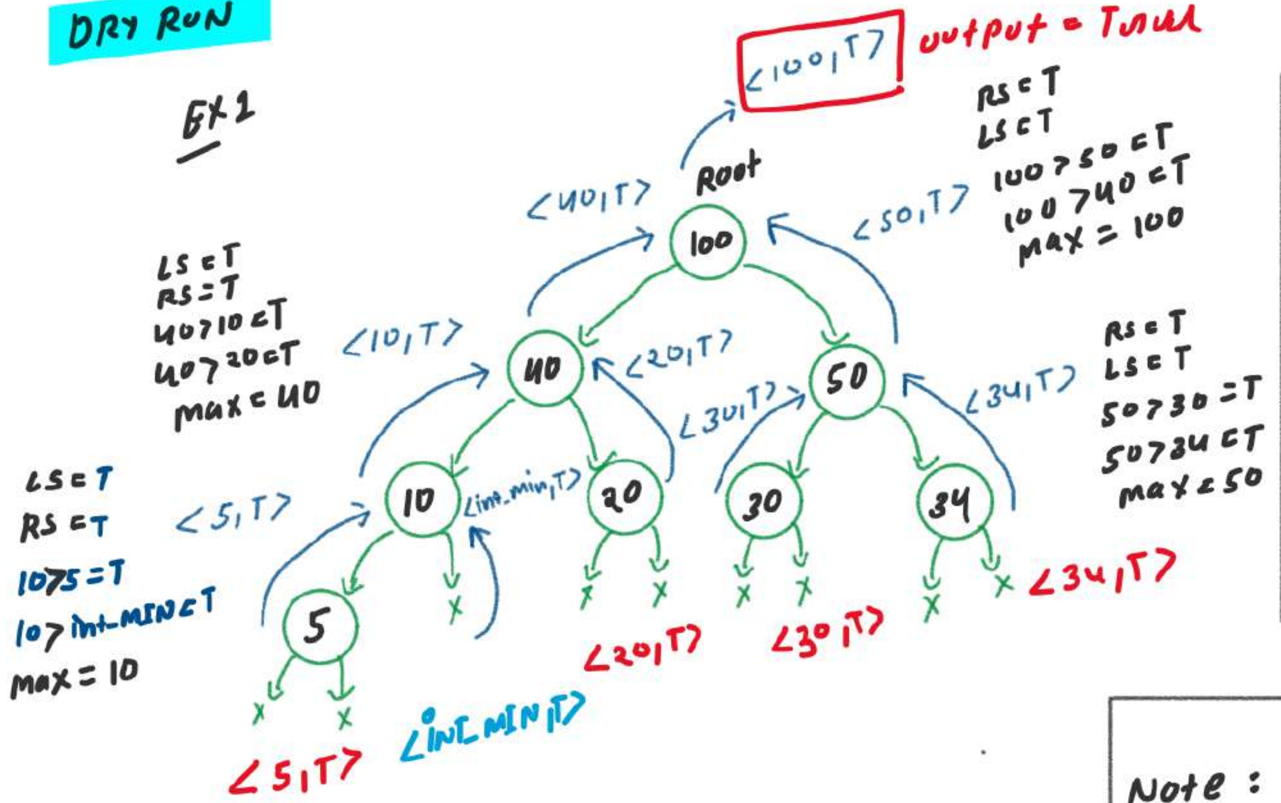
↳ < x , y > Return

Base case 2 Root == leaf Node

↳ < x , y > Return

DRY RUN

Ex 1



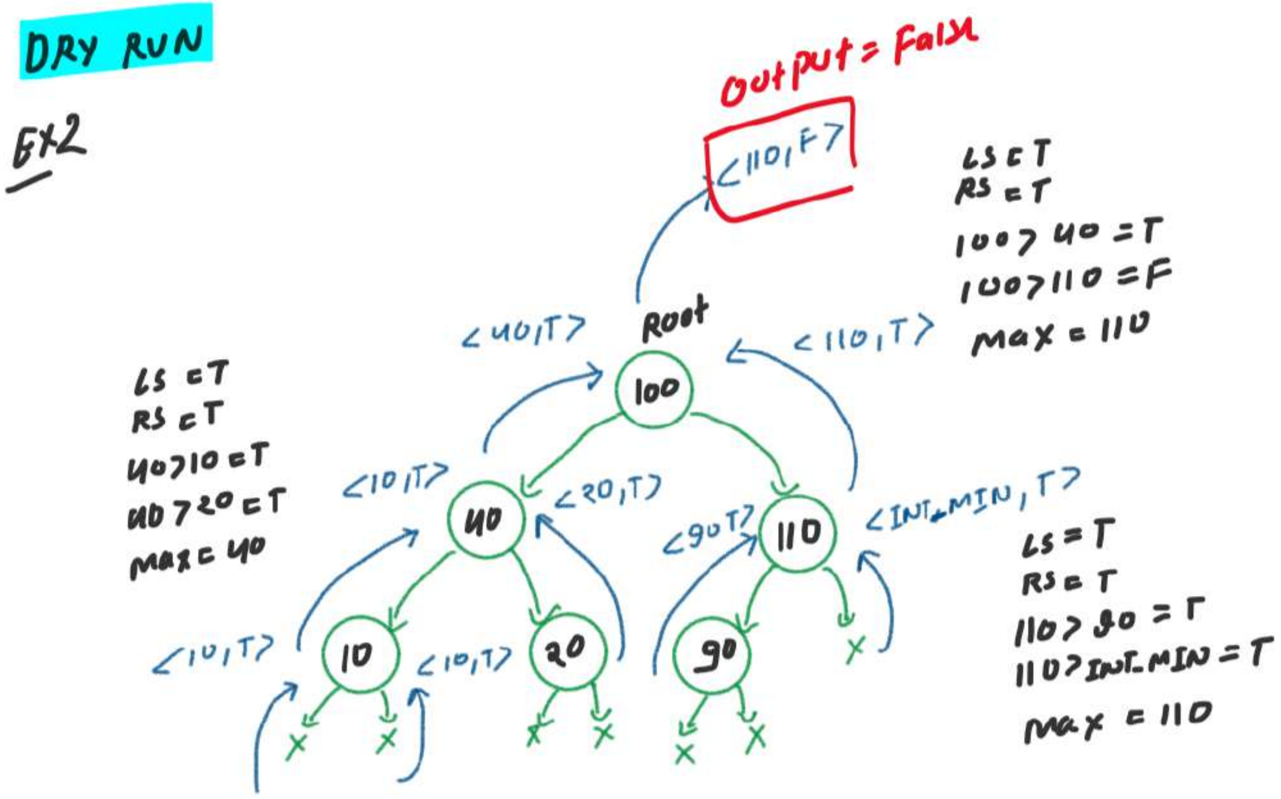
Max min Heap valid kab Hogi

1. Left Sub Tree = True
2. Right sub Tree = True
3. Root → data > Left → data = TRUE
4. Root → data > Right → data = True

Note : if we return only a boolean value even then question will be solved.

DRY RUN

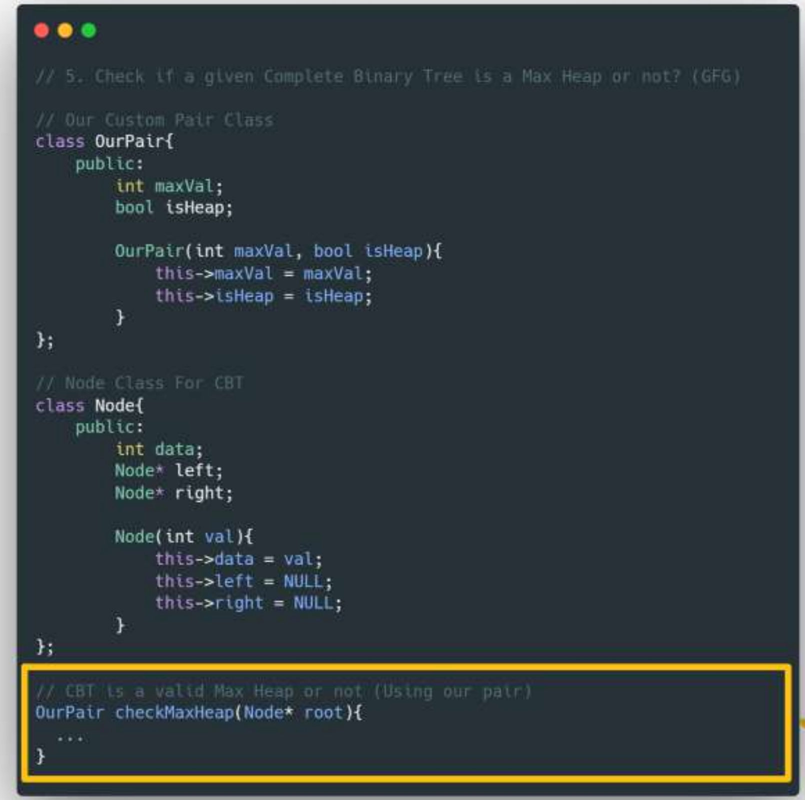
Ex2



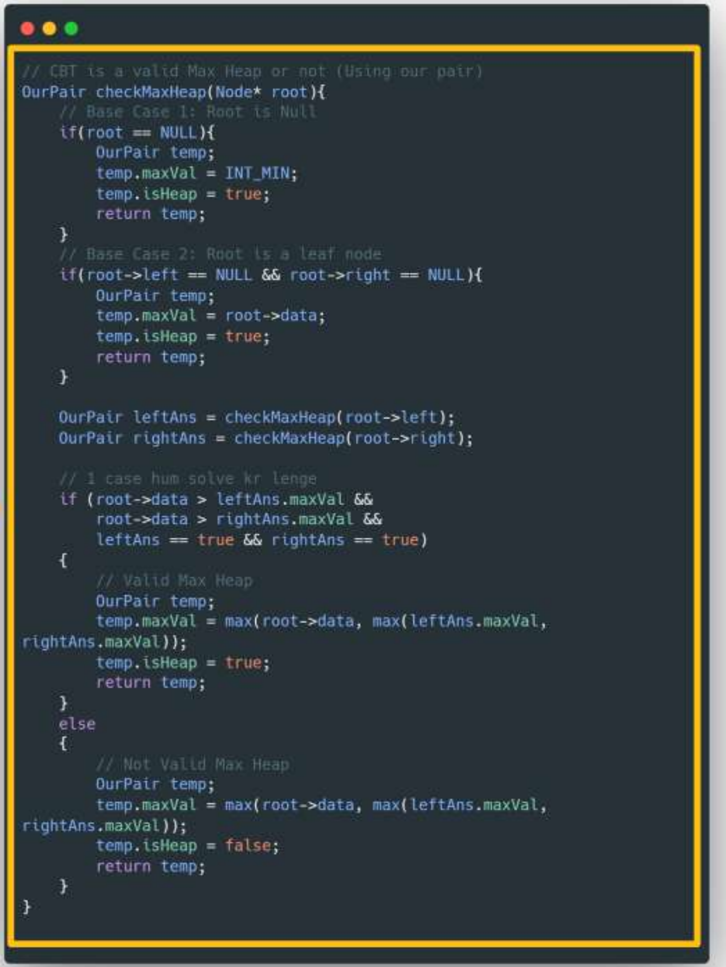
Custom pair class

```
class ourPair {
    public:
        int maxVal;
        bool isHeap;

        ourPair(int a, bool b) {
            maxVal = a;
            isHeap = b;
        }
};
```

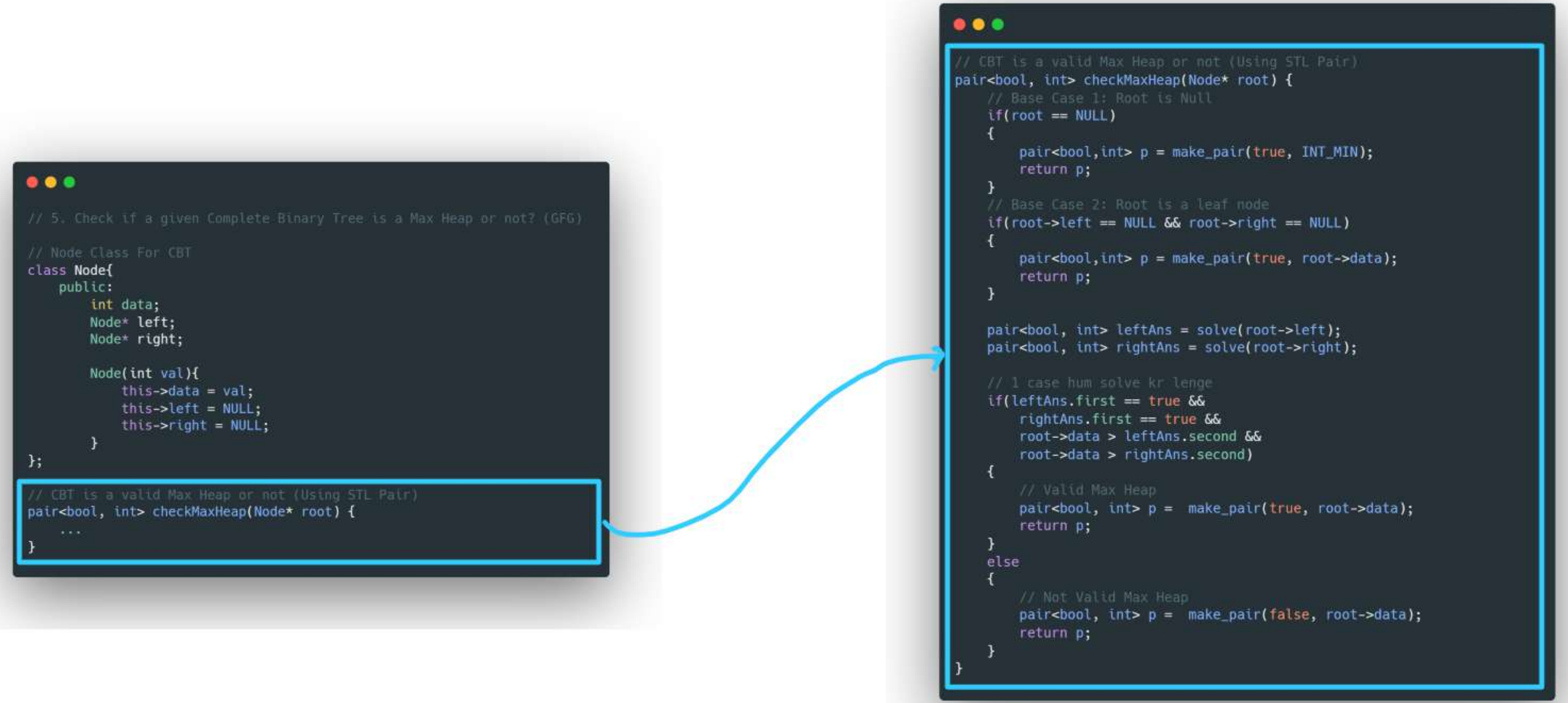
STL pair : pair< X , Y >



```cpp
// 5. Check if a given Complete Binary Tree is a Max Heap or not? (GFG)

// Our Custom Pair Class
class OurPair{
    public:
        int maxVal;
        bool isHeap;

        OurPair(int maxVal, bool isHeap){
            this->maxVal = maxVal;
            this->isHeap = isHeap;
        }
};

// Node Class For CBT
class Node{
    public:
        int data;
        Node* left;
        Node* right;

        Node(int val){
            this->data = val;
            this->left = NULL;
            this->right = NULL;
        }
};
// CBT is a valid Max Heap or not (Using our pair)
OurPair checkMaxHeap(Node* root){
  ...
}
```



```cpp
// CBT is a valid Max Heap or not (Using our pair)
OurPair checkMaxHeap(Node* root){
    // Base Case 1: Root is Null
    if(root == NULL){
        OurPair temp;
        temp.maxVal = INT_MIN;
        temp.isHeap = true;
        return temp;
    }
    // Base Case 2: Root is a leaf node
    if(root->left == NULL && root->right == NULL){
        OurPair temp;
        temp.maxVal = root->data;
        temp.isHeap = true;
        return temp;
    }

    OurPair leftAns = checkMaxHeap(root->left);
    OurPair rightAns = checkMaxHeap(root->right);

    // 1 case hum solve kr lenge
    if (root->data > leftAns.maxVal &&
        root->data > rightAns.maxVal &&
        leftAns == true && rightAns == true)
    {
        // Valid Max Heap
        OurPair temp;
        temp.maxVal = max(root->data, max(leftAns.maxVal,
rightAns.maxVal));
        temp.isHeap = true;
        return temp;
    }
    else
    {
        // Not Valid Max Heap
        OurPair temp;
        temp.maxVal = max(root->data, max(leftAns.maxVal,
rightAns.maxVal));
        temp.isHeap = false;
        return temp;
    }
}
```



```cpp
// 5. Check if a given Complete Binary Tree is a Max Heap or not? (GFG)

// Node Class For CBT
class Node{
    public:
        int data;
        Node* left;
        Node* right;

        Node(int val){
            this->data = val;
            this->left = NULL;
            this->right = NULL;
        }
};

// CBT is a valid Max Heap or not (Using STL Pair)
pair<bool, int> checkMaxHeap(Node* root) {
    ...
}
```

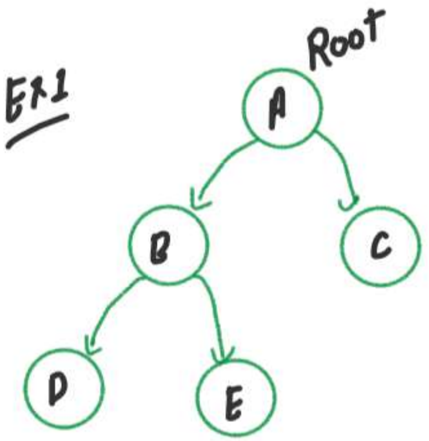
```cpp
// CBT is a valid Max Heap or not (Using STL Pair)
pair<bool, int> checkMaxHeap(Node* root) {
    // Base Case 1: Root is Null
    if(root == NULL)
    {
        pair<bool,int> p = make_pair(true, INT_MIN);
        return p;
    }
    // Base Case 2: Root is a leaf node
    if(root->left == NULL && root->right == NULL)
    {
        pair<bool,int> p = make_pair(true, root->data);
        return p;
    }

    pair<bool, int> leftAns = solve(root->left);
    pair<bool, int> rightAns = solve(root->right);

    // 1 case hum solve kr lenge
    if(leftAns.first == true &&
        rightAns.first == true &&
        root->data > leftAns.second &&
        root->data > rightAns.second)
    {
        // Valid Max Heap
        pair<bool, int> p =  make_pair(true, root->data);
        return p;
    }
    else
    {
        // Not Valid Max Heap
        pair<bool, int> p =  make_pair(false, root->data);
        return p;
    }
}
```

Time complexity: *O(N),*
where N is the number of nodes in the binary tree

Space complexity: *O(H),*
where H is the height of the binary tree.

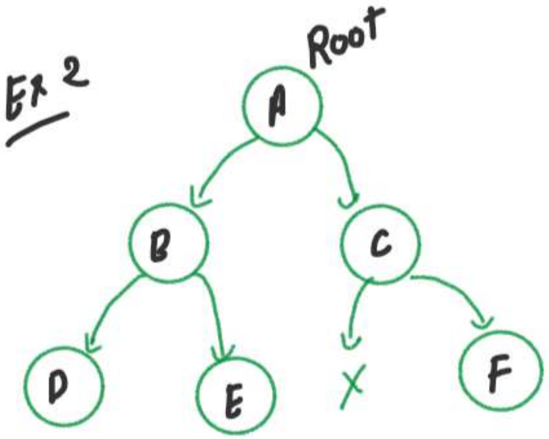
6. Check Whether a Binary Tree is a CBT or not? (Leetcode-958)

Ex 1



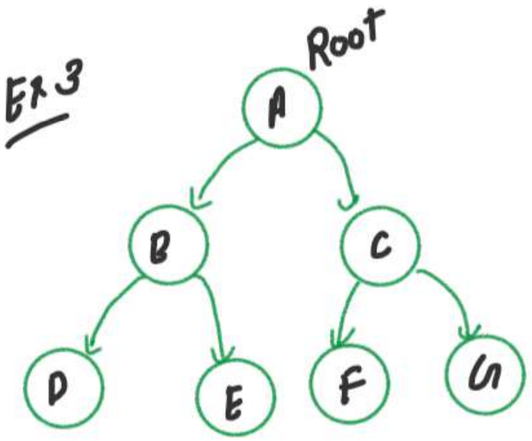
Output TRUE

Ex 2



Output False

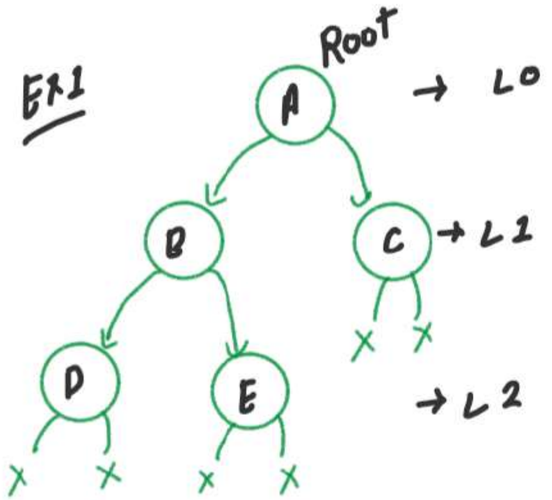
Ex 3



Output TRUE

Logic building

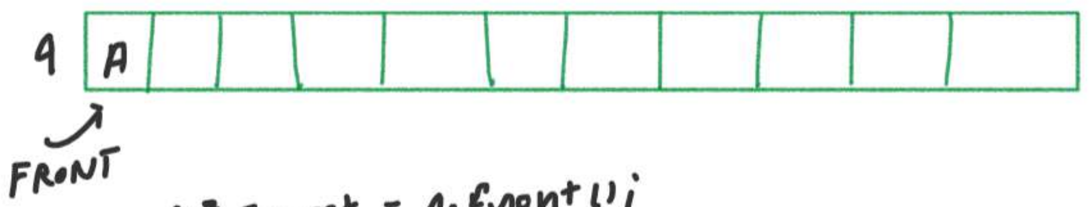
Ex1



Output TRUE

using level order Traversal

initialy ⇒ q.push(Root) and
bool NullFound = False



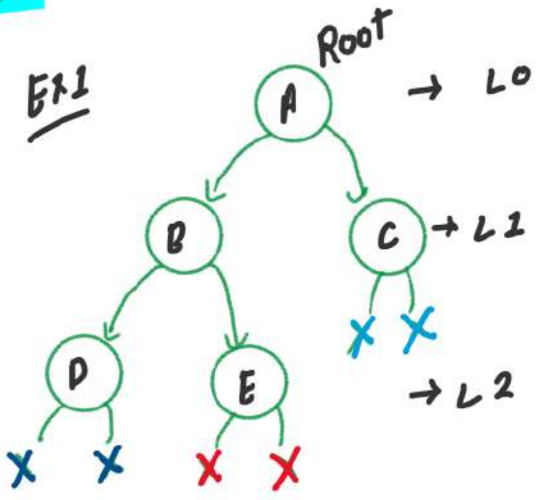
```
Node* Front = q.front();
          q.pop();
if(Front == NULL ){
    NullFond = TRUE
}
```

OR

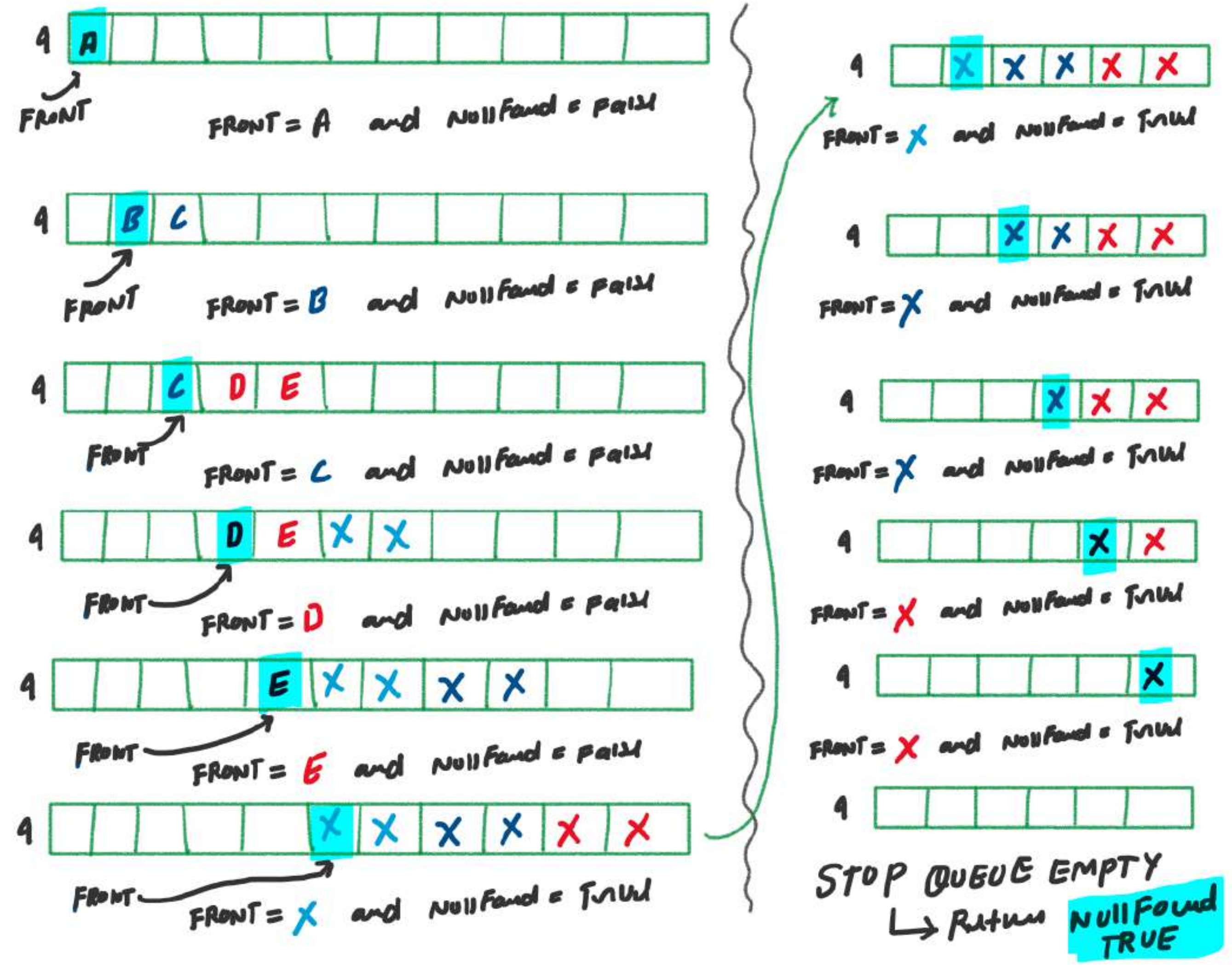
```
if( Front != NULL){
    if( NullFund == True)
        return False;
    q.push( Front→left);
    q.push( Front → right);
}
```

DRY RUN

Ex1



Output TRUE



FRONT = A and NullFound = false

FRONT = B and NullFound = false

FRONT = C and NullFound = false

FRONT = D and NullFound = false

FRONT = E and NullFound = false

FRONT = X and NullFound = True

FRONT = X and NullFound = True

FRONT = X and NullFound = True

FRONT = X and NullFound = True

FRONT = X and NullFound = True

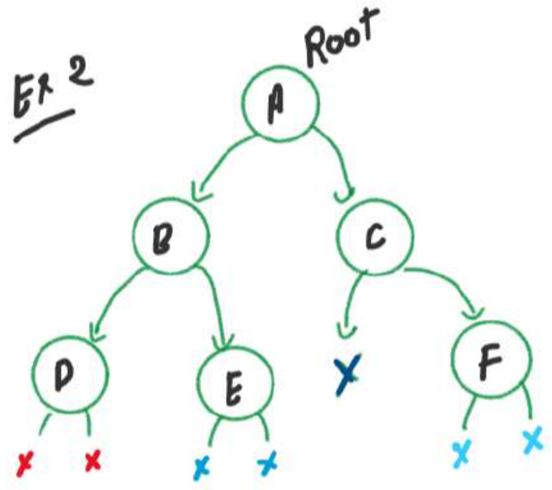
FRONT = X and NullFound = True

STOP QUEUE EMPTY

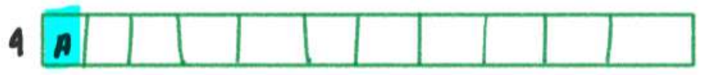
↳ Return **NullFound TRUE**

DRY RUN

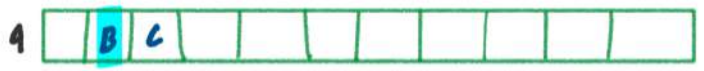
Ex 2



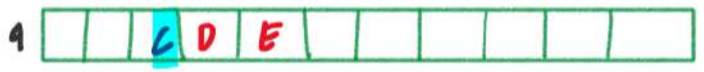
Output False



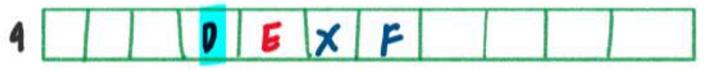
FRONT = A and NullFound = false



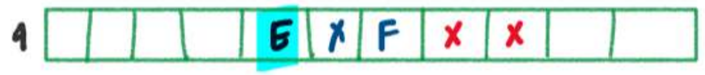
FRONT = B and NullFound = false



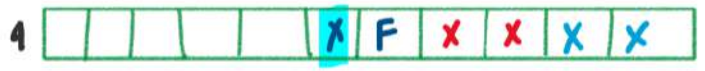
FRONT = C and NullFound = false



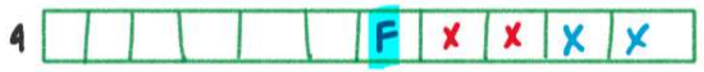
FRONT = D and NullFound = false



FRONT = E and NullFound = false



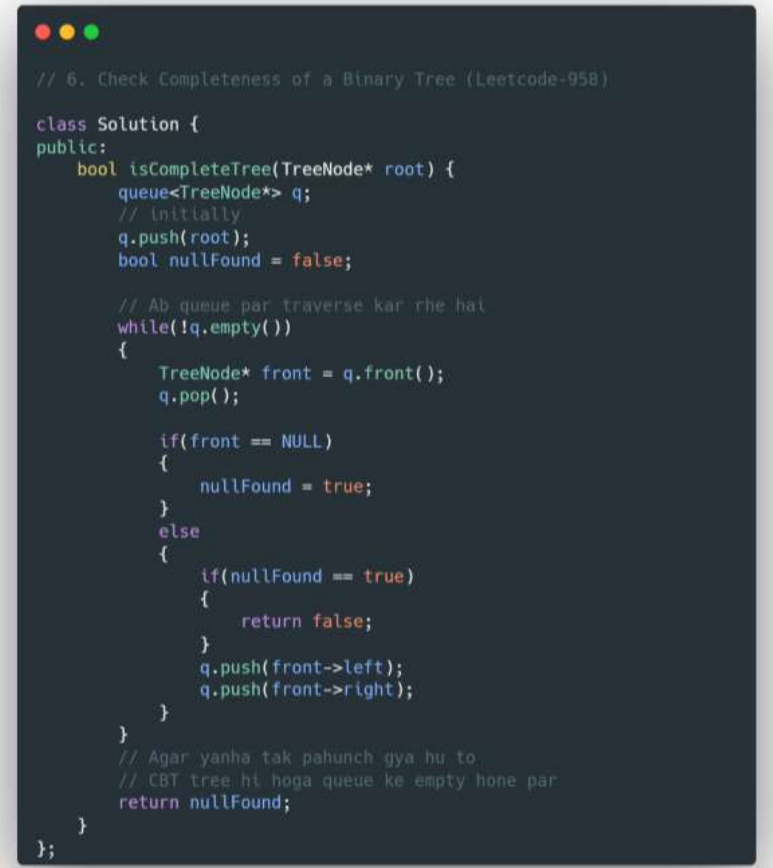
FRONT = X and NullFound = TRUE



FRONT = F and NullFound = TRUE

Stop (FRONT != NULL and NullFound == TRUE)

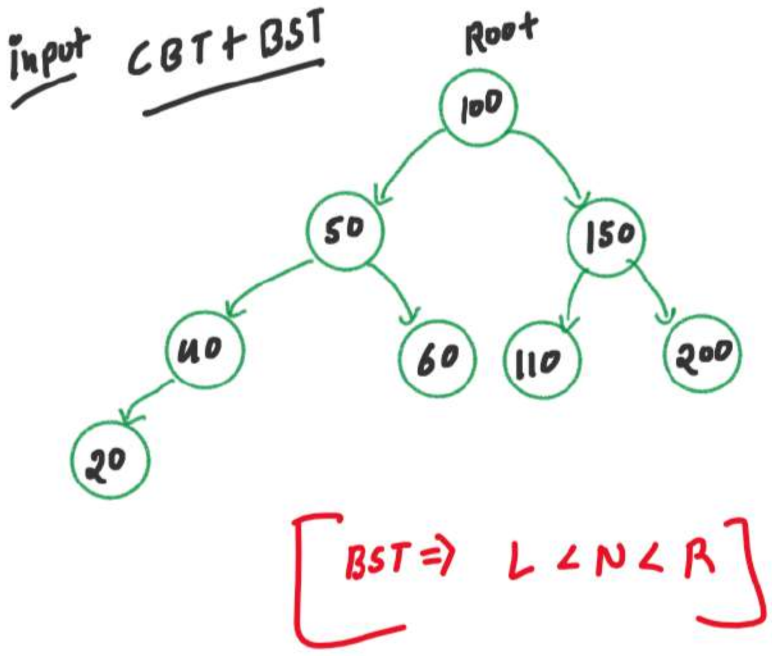
↳ RETURN False

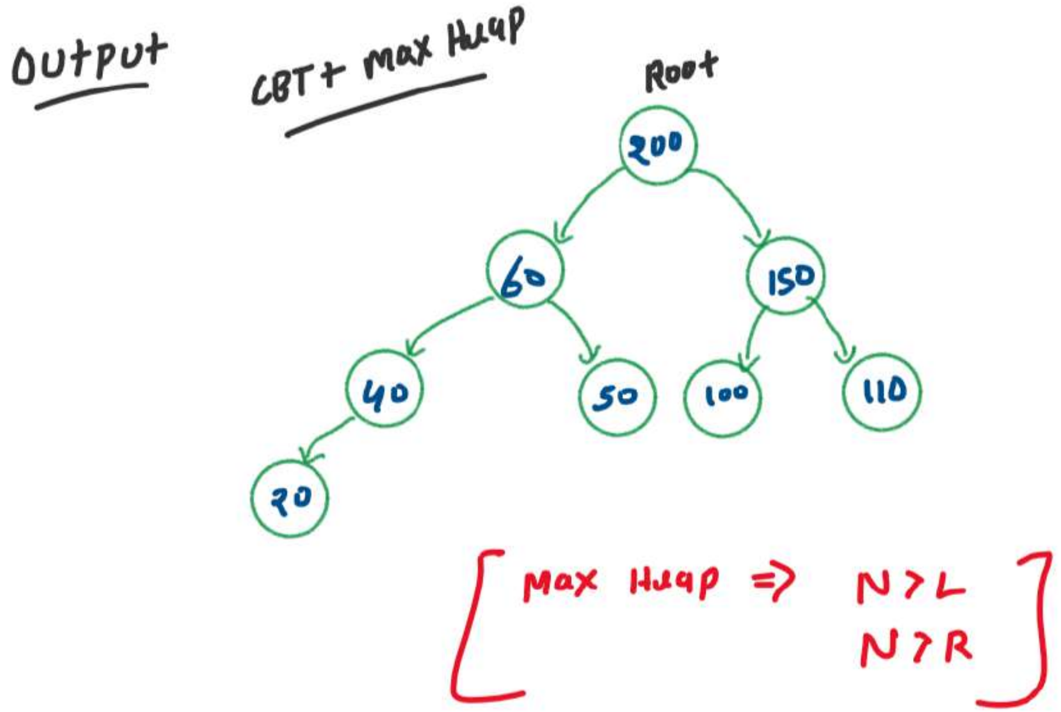


```cpp
// 6. Check Completeness of a Binary Tree (Leetcode-958)

class Solution {
public:
    bool isCompleteTree(TreeNode* root) {
        queue<TreeNode*> q;
        // initially
        q.push(root);
        bool nullFound = false;

        // Ab queue par traverse kar rhe hai
        while(!q.empty())
        {
            TreeNode* front = q.front();
            q.pop();

            if(front == NULL)
            {
                nullFound = true;
            }
            else
            {
                if(nullFound == true)
                {
                    return false;
                }
                q.push(front->left);
                q.push(front->right);
            }
        }
        // Agar yanha tak pahunch gya hu to
        // CBT tree hi hoga queue ke empty hone par
        return nullFound;
    }
};
```

Time Complexity: *O(N)*
Space Complexity: *O(N)*
Where N is number of nodes in binary tree

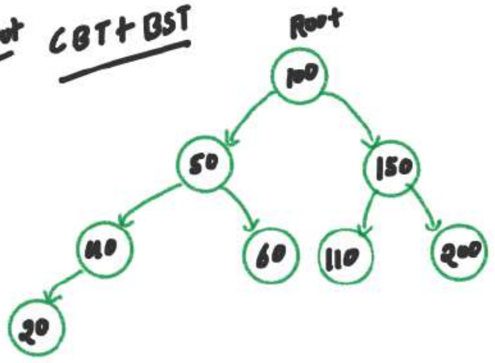
7. Convert Given Combination of CBT and BST into a Valid Max Heap (GFG)





Logic Building

Input CBT+BST



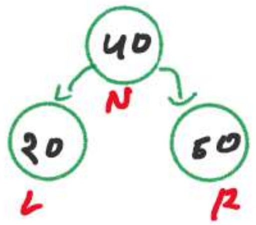
STEP1 STORE INORDER TRAVERSAL $\boxed{LNR}$

array	20	40	50	60	100	110	150	200
	0	1	2	3	4	5	6	7

STEP2 REPLACE Each node→data of CBT + BST TO Build the max Heap using POST-ORDER [LRN]

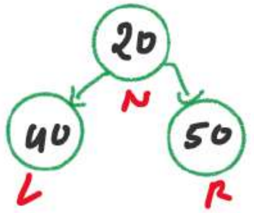
[We have four Traversals to Traverse the BST]

① Inorder ⇒ ✗ LNR



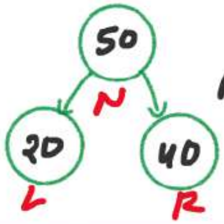
Not max heap ✗

② Pre order ⇒ ✗ NLR



Not max heap ✗

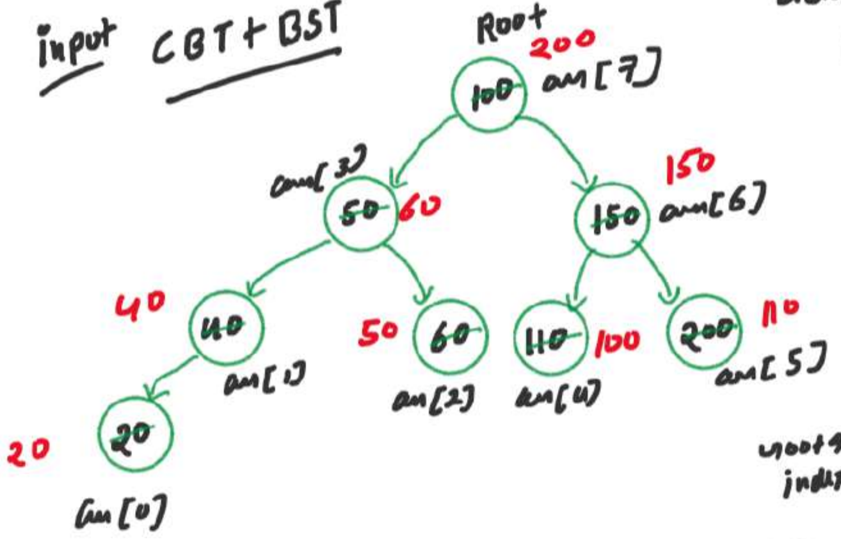
③ post-order ⇒ ✓ LRN



Max Heap ✓

DRY RUN

input CBT + BST

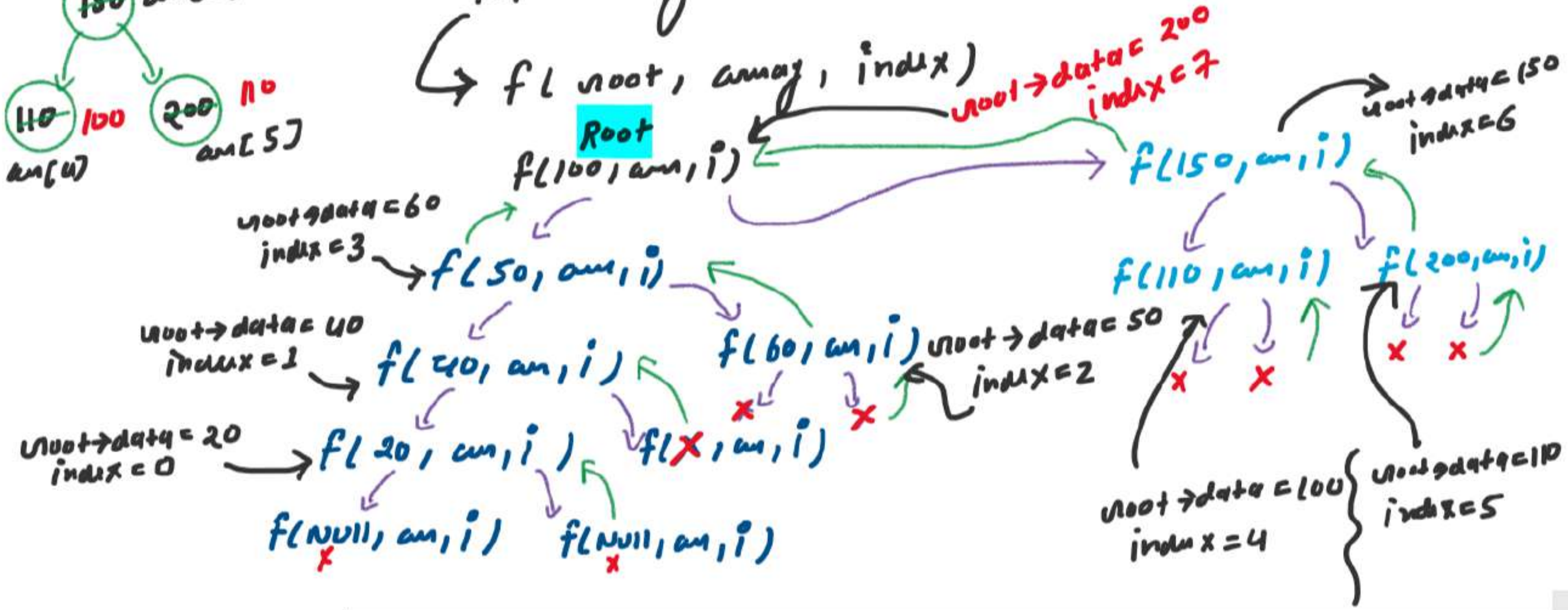


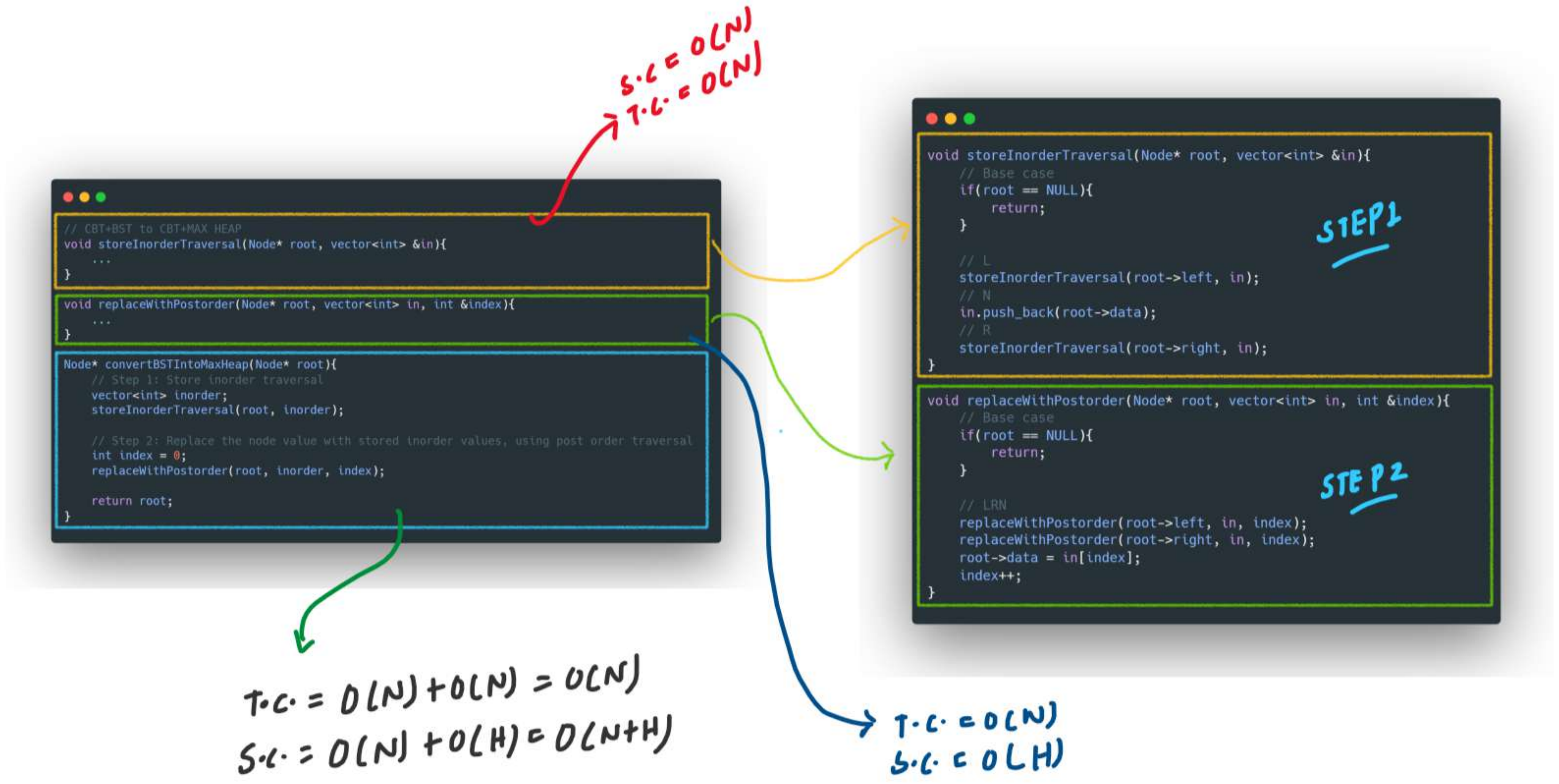
array

20	40	50	60	100	110	150	200
0	1	2	3	4	5	6	7

Replace using Post-Order (LRN)

↳ f(root, array, index)





```cpp
// CBT+BST to CBT+MAX HEAP
void storeInorderTraversal(Node* root, vector<int> &in){
    ...
}
void replaceWithPostorder(Node* root, vector<int> in, int &index){
    ...
}
Node* convertBSTIntoMaxHeap(Node* root){
    // Step 1: Store inorder traversal
    vector<int> inorder;
    storeInorderTraversal(root, inorder);

    // Step 2: Replace the node value with stored inorder values, using post order traversal
    int index = 0;
    replaceWithPostorder(root, inorder, index);

    return root;
}
```

S.C = O(N)
T.C. = O(N)

```cpp
void storeInorderTraversal(Node* root, vector<int> &in){
    // Base case
    if(root == NULL){
        return;
    }

    // L
    storeInorderTraversal(root->left, in);
    // N
    in.push_back(root->data);
    // R
    storeInorderTraversal(root->right, in);
}
```

STEP1

```cpp
void replaceWithPostorder(Node* root, vector<int> in, int &index){
    // Base case
    if(root == NULL){
        return;
    }

    // LRN
    replaceWithPostorder(root->left, in, index);
    replaceWithPostorder(root->right, in, index);
    root->data = in[index];
    index++;
}
```

STEP2

T.C. = O(N) + O(N) = O(N)
S.C. = O(N) + O(H) = O(N+H)

T.C. = O(N)
S.C. = O(H)

Time complexity: *O(N)*
Space complexity: *O(H + N)*

Where N and H are the number of nodes and height of the binary tree respectively.