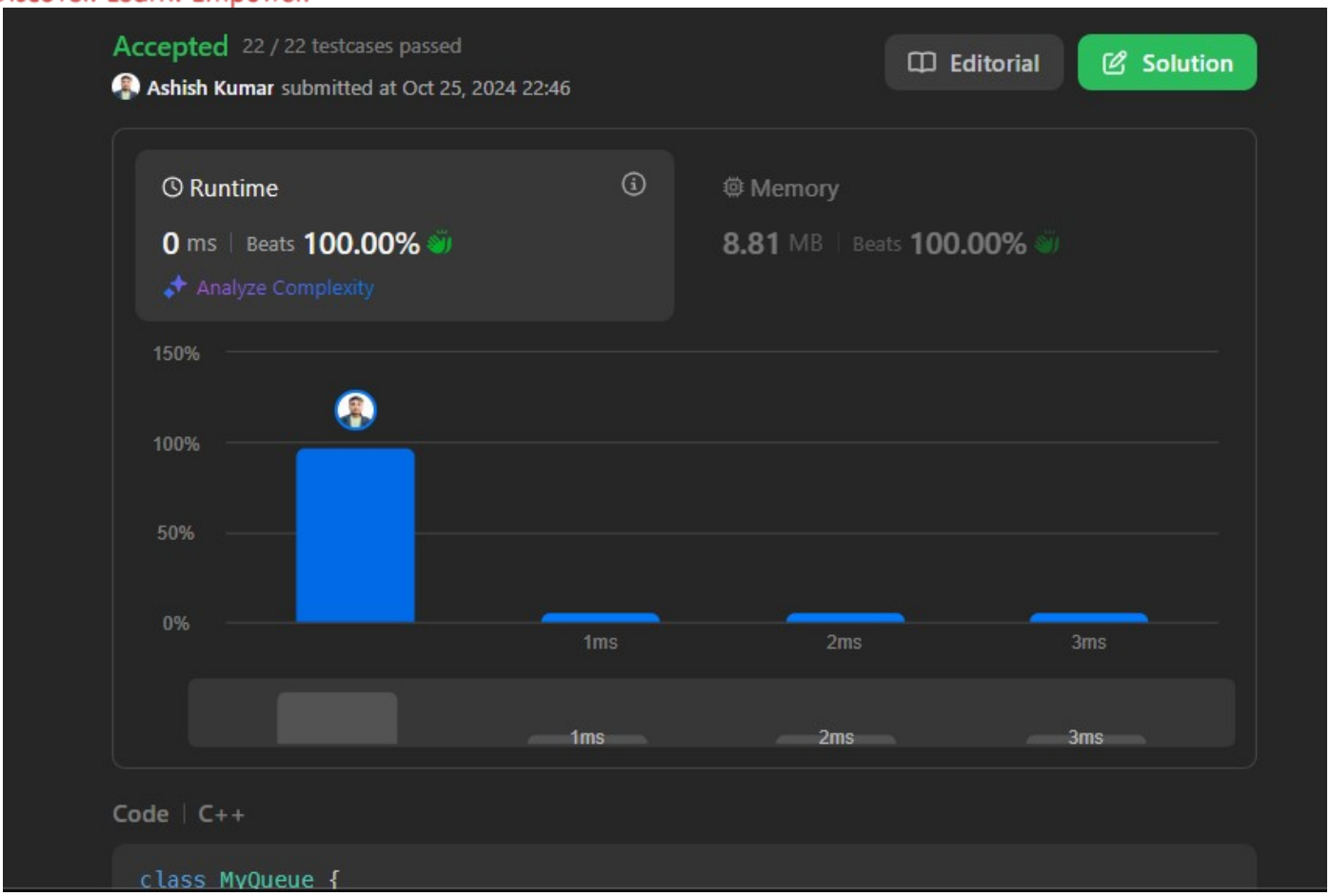
Experiment 6

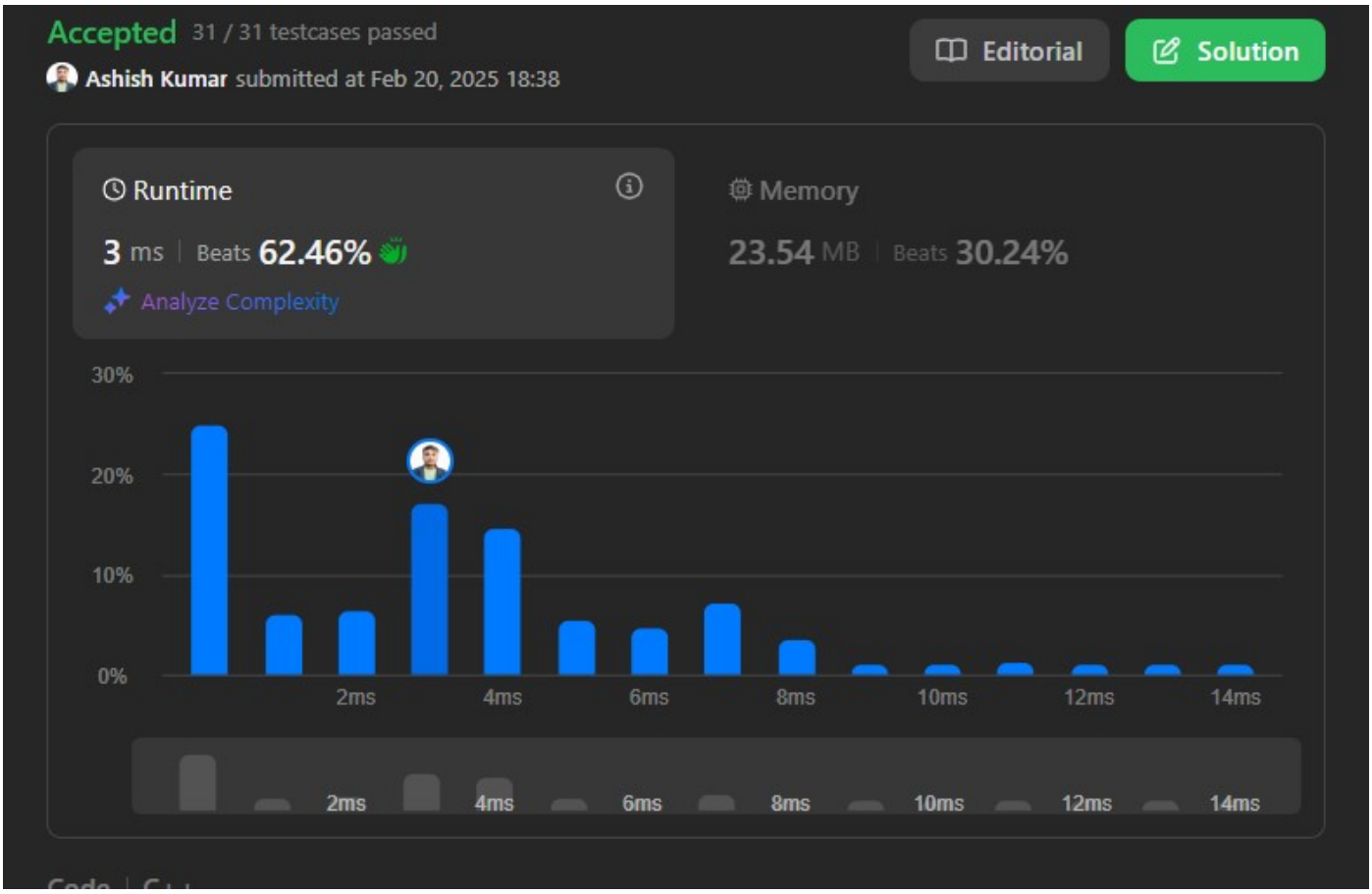
Student Name: Ashish Kumar **UID:22bcs11958**
Branch: CSE **Section/Group:614(B)**
Semester: 6 **Date of Performance:12/3/25**
Subject Name: AP Lab **Subject Code: 22CSP-351**

```
class MyQueue {
public:
    MyQueue() {

    }
    stack<int>input,output;
    void push(int x) {
        input.push(x);
    }

    int pop() {
    int val=peek();
    output.pop();
    return val;
    }
    int peek() {
        if(output.empty())
        {
            while(!input.empty())
            {
                output.push(input.top());
                input.pop();
            }
        }
        return output.top();
    }

    bool empty() {
        return input.empty() && output.empty();
    }
};

/**
 * Your MyQueue object will be instantiated and called as such:
 * MyQueue* obj = new MyQueue();
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->peek();
 * bool param_4 = obj->empty();
 */
```



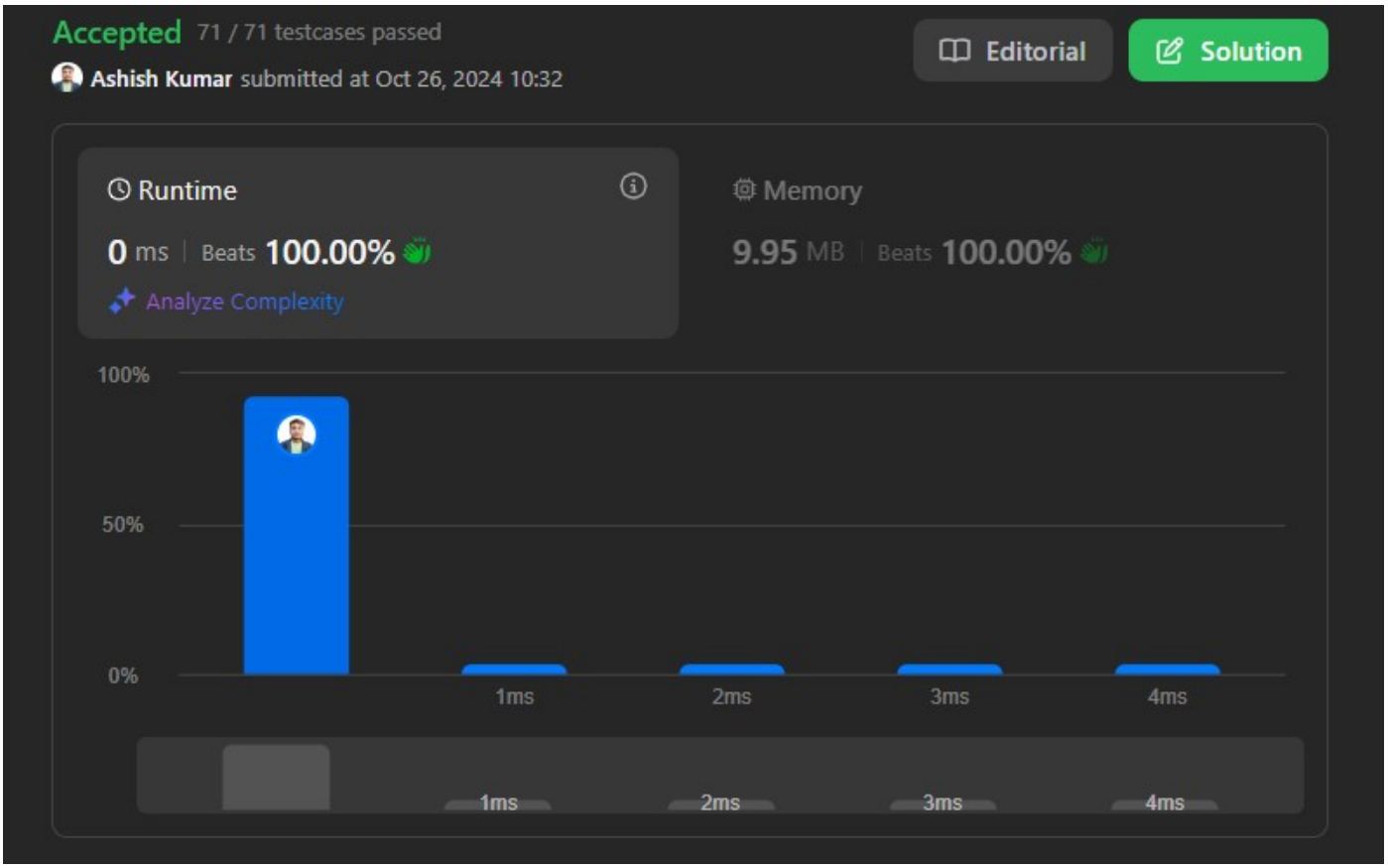

```cpp
class MinStack {
public:
   stack<int>st,st2;
    MinStack() {

    }

    void push(int val) {
       if(st2.empty() || st2.top()>=val)
       {
        st2.push(val);
       }
       st.push(val);
    }

    void pop() {
        if(st.top()==st2.top())
        {
            st2.pop();
        }
        st.pop();
    }
```

```
    int top() {
        return st.top();
    }

    int getMin() {
        return st2.top();
    }
};

/**
 * Your MinStack object will be instantiated and called as such:
 * MinStack* obj = new MinStack();
 * obj->push(val);
 * obj->pop();
 * int param_3 = obj->top();
 * int param_4 = obj->getMin();
 */
```



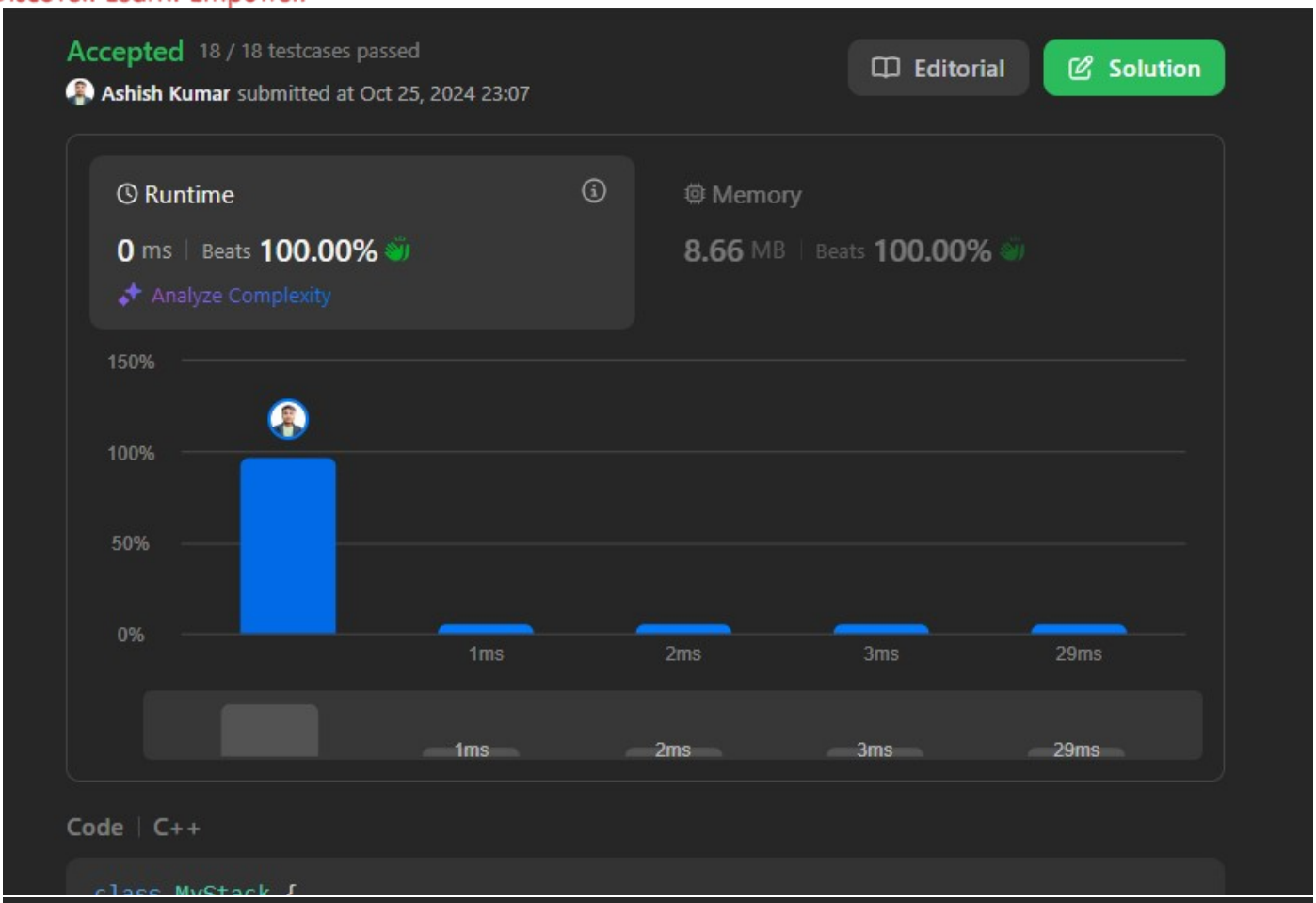

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
```

```
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    void inordrTempVal(TreeNode*node,vector<int>&ans)
    {
     if(node==NULL)
     {
        return;
     }
     inordrTempVal(node->left,ans);
     ans.push_back(node->val);
     inordrTempVal(node->right,ans);
    }
    vector<int> inorderTraversal(TreeNode* root) {
       vector<int>ans;
       inordrTempVal(root,ans);
       return ans;
    }
};
```




```
class MyStack {
```

```
public:
    MyStack() {

    }
    queue<int>q1,q2;
    void push(int x) {
        while(!q1.empty())
        {
            q2.push(q1.front());
            q1.pop();
        }
        q1.push(x);
        while(!q2.empty())
        {
            q1.push(q2.front());
            q2.pop();
        }

    }

    int pop() {
        int val=q1.front();
        q1.pop();
        return val;
    }

    int top() {
        return q1.front();
    }

    bool empty() {
        return q1.empty();
    }
};

/**
 * Your MyStack object will be instantiated and called as such:
 * MyStack* obj = new MyStack();
 * obj->push(x);
 * int param_2 = obj->pop();
 * int param_3 = obj->top();
 * bool param_4 = obj->empty();
 */
```




```cpp
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    vector<vector<int>> levelOrder(TreeNode* root) {
        vector<vector<int>>ans;
        queue<TreeNode*>qu;
        if(root==NULL)
        {
            return ans;
        }
        qu.push(root);
        while (!qu.empty()) {
            int n = qu.size();
            vector<int> level;
```

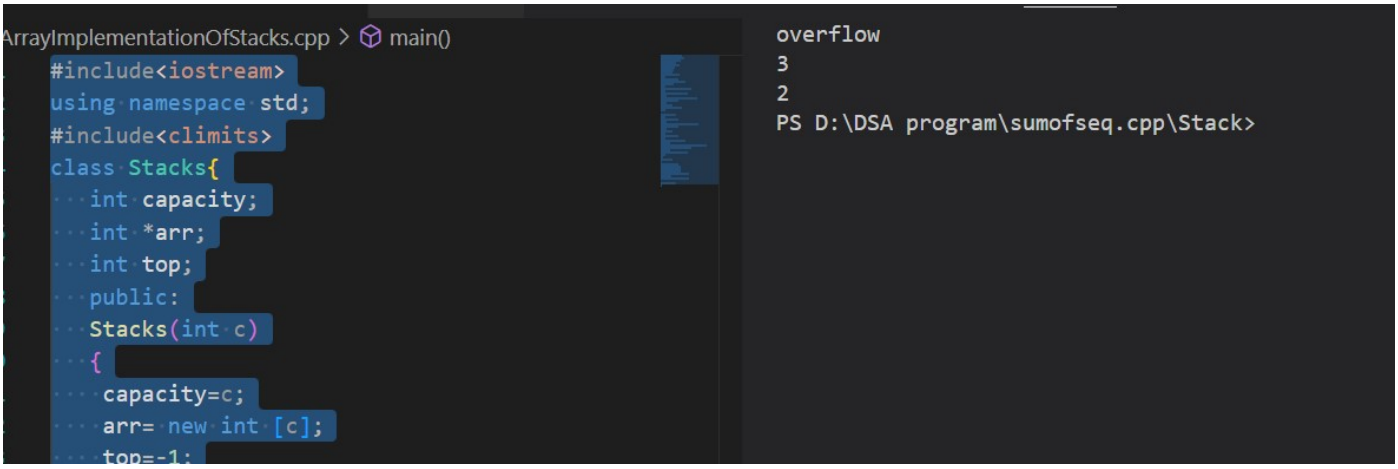
```
            for (int i = 0; i < n; i++) {
                TreeNode* currnode = qu.front();
                qu.pop();
                level.push_back(currnode->val);

                if (currnode->left != nullptr) {
                    qu.push(currnode->left);
                }
                if (currnode->right != nullptr) {
                    qu.push(currnode->right);
                }
            }
            ans.push_back(level);
        }
        return ans;
    }
};
```


```
#include<iostream>
using namespace std;
#include<climits>
class Stacks{
```

```
    int capacity;
    int *arr;
    int top;
    public:
    Stacks(int c)
    {
     capacity=c;
     arr= new int [c];
     top=-1;
    }
    void push_val(int data)
    {
     if(top==capacity-1)
     {
         cout<<"overflow"<<endl;
         return;
     }
     top++;
     arr[top]=data;
    }

    void pop()
    {
     if(top==-1)
     {
         cout<<"underflow"<<endl;
         return;
     }
     top--;
    }
    int getTop()
    {
        if (top == -1)
        {
            cout << "underflow" << endl;
            return INT_MIN;
        }
        return arr[top];
    }
};
int main()
{
  Stacks st(2);
  st.push_val(2);
  st.push_val(3);
  st.push_val(4);
cout<<st.getTop()<<endl;
st.pop();
```

```
cout<<st.getTop()<<endl;
```

```
    return 0;
}
```




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

class Queue {
    int front;
    int back;
    vector<int> v;
```

```
public:
    Queue() {
        front = -1;
        back = -1;
    }
```

```
    void Enqueue(int val) {
        v.push_back(val);
        back++;
        if (back == 0) front = 0;
    }
```

```
    void Dequeue() {

        if (front == back) {
            front = -1;
            back = -1;
            v.clear();
        } else {
            front++;
        }
```

```
    }

    int getfront() {
        if (front == -1) {
            cout << "Queue is empty." << endl;
            return -1;
        }
        return v[front];
    }

    bool empty() {
        return front == -1;
    }
};

int main() {
    Queue qu;
    qu.Enqueue(5);
    qu.Enqueue(6);
    qu.Enqueue(8);
    qu.Dequeue();
    qu.Enqueue(9);

    while (!qu.empty()) {
        cout << qu.getfront() << endl;
        qu.Dequeue();
    }
    return 0;
}
```


```
#include<iostream>
#include<algorithm>
using namespace std;
int const N=1000;
void InsertionMinHeap(int Heap[], int val, int &size)
{
    size++;
    Heap[size] = val;
```

```
    int curr = size;
    while (curr / 2 > 0 && Heap[curr / 2] > Heap[curr])
    {
        swap(Heap[curr / 2], Heap[curr]);
        curr = curr / 2;
    }
}
int main()
{
    int Heap[N] = {-1, 10, 20, 30, 40, 50};  // Initializing array with -1 at index 0
    int size = 5;
    InsertionMinHeap(Heap, 5, size);
    for (int i = 1; i <= size; i++)
    {
        cout << Heap[i] << " ";
    }
    return 0;
}
```

InsertionInMinHeap.cpp > main()

```
#include<iostream>
#include<algorithm>
using namespace std;
int const N=1000;
void InsertionMinHeap(int Heap[], int val, int &size)
```

```
5 20 10 40 50 30
PS D:\DSA program\sumofseq.cpp\Heaps>
```

```
#include<iostream>
#include<algorithm>
using namespace std;
int const N=1000;
void InsertionMaxHeap(int Heap[], int val, int &size)
{
    size++;
    Heap[size] = val;
    int curr = size;
    while (curr / 2 > 0 && Heap[curr / 2] < Heap[curr])
    {
        swap(Heap[curr / 2], Heap[curr]);
        curr = curr / 2;
    }
}
int main()
{
    int Heap[N] = {-1, 10, 20, 30, 40, 50};
    int size = 5;
    InsertionMaxHeap(Heap, 100, size);
    for (int i = 1; i <= size; i++)
    {
        cout << Heap[i] << " ";
```

```
    }
    return 0;
}
```

InsertionInMaxHeap.cpp > main()

```
#include<iostream>
#include<algorithm>
using namespace std;
int const N=1000;
void InsertionMaxHeap(int Heap[], int val, int &size)
{
    size++;
    Heap[size] = val;
```

```
100 20 10 40 50 30
PS D:\DSA program\sumofseq.cpp\Heaps>
```

```
#include<bits/stdc++.h>
using namespace std;
struct Myhash{
    int *arr;
    int cap,size;
    Myhash(int c)
    {
        cap=c;
        size =0;
        arr = new int[cap];
        for(int i=0;i<cap;i++)
        {
            arr[i]=-1;
        }
    }
    int hash(int key)
    {
      return key%cap;
    }
    void Insert(int key) {
        int i = hash(key);

        while (arr[i] != -1) {
            i = (i + 1) % cap;
        }
```

```
        arr[i] = key;
    }
    bool Search(int key)
    {
        int h=hash(key);
        int i=h;
        while(arr[i]!=-1)
        {
            if(arr[i]==key)
            return true;
```

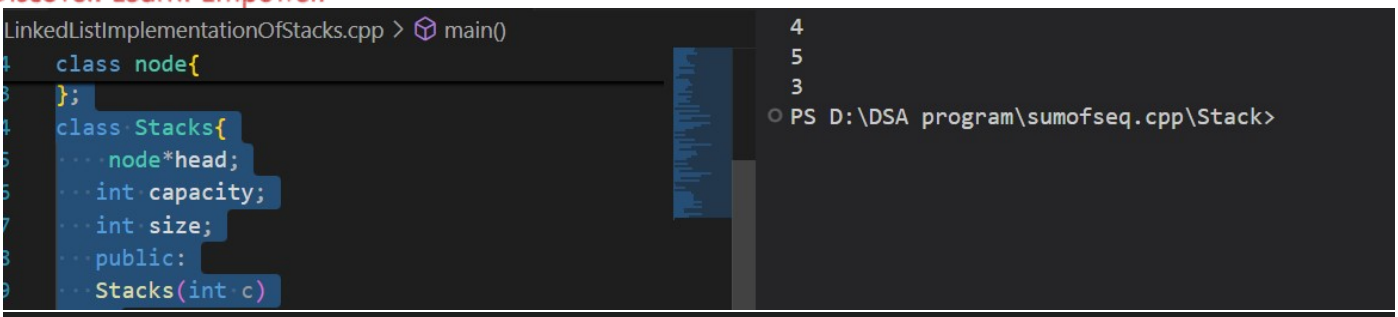
```
            i=(i+1)%cap;
            if(i==h)
            return false;
        }
    return false;
    }
    bool Remove(int key)
    {
       int h=hash(key);
        int i=h;
        while(arr[i]!=-1)
        {
            if(arr[i]==key)
            arr[i]=-2;
            return true;
            i=(i+1)%cap;
            if(i==h)
            return false;
        }
    return false;
    }
    void display()
    {
        for(int i=0;i<cap;i++)
        {
            if(arr[i]!=-1)
            {
            cout<<arr[i]<<" ";
            }
            else
            {
            cout<<"empty";
            }
        }
    }
    ~Myhash() {
        delete[] arr;
    }
};
int main()
{
  Myhash M(7);
  M.Insert(49);
  M.Insert(50);
  M.Insert(63);
  M.Insert(64);
  M.Insert(69);
  M.Insert(68);
```

```
  M.display();
  if(M.Search(56))
    {
        cout<<"56 found";
    }
    else{
        cout<<" 56 not found";
    }
```

```
    M.Remove(68);
    return 0;
}
```


```
#include<iostream>
#include<climits>
using namespace std;
class node{
    public:
    int val;
    node*next;
    node(int data)
    {
        val=data;
        next=NULL;
    }
};
class Stacks{
    node*head;
   int capacity;
   int size;
   public:
   Stacks(int c)
   {
    capacity=c;
    size=0;
    head=NULL;
   }
void push_val(int val)
{
    if(size==capacity)
```

```
    {
        cout<<"overflow"<<endl;
        return;
    }
   node*new_node=new node(val);
   new_node->next=head;
   head=new_node;
   size++;
}
void pop()
{
    if(head==NULL)
    {
        cout<<"underflow"<<endl;
        return;
    }
    node*temp=head;
    head=head->next;
    free(temp);
    size--;
}
int getTop()
{
  if(head==NULL)
    {
        cout<<"underflow"<<endl;
        return INT_MIN;
    }
    return head->val;
}
};
int main()
{
  Stacks st(5);
  st.push_val(2);
  st.push_val(3);
  st.push_val(4);
cout<<st.getTop()<<endl;
  st.push_val(5);
cout<<st.getTop()<<endl;
st.pop();
st.pop();
cout<<st.getTop()<<endl;
    return 0;
}
```

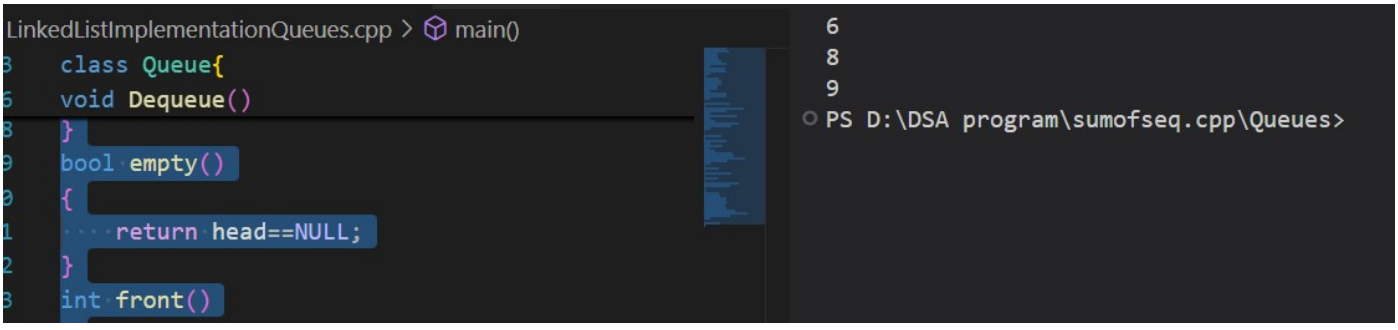



```cpp
#include<iostream>
using namespace std;
class node{
    public:
    int val;
    node*next;
    node(int data)
    {
        val=data;
        next=NULL;
    }
};
class Queue{
    node*head;
    node*tail;
    public:
    Queue()
    {
        head=NULL;
        tail=NULL;
    }

void Enqueue(int val)
{
    node*new_node=new node(val);
    if(head==NULL)
    {
        head=tail=new_node;
    }
    else{
        tail->next=new_node;
        tail=new_node;
    }
}
```

```cpp
void Dequeue()
{
    if(head==NULL)
    {
        return;
```

```
    }
    else{
        node*temp=head;
        head=head->next;
        if(head==NULL) tail=NULL;
        free(temp);
    }
}
bool empty()
{
    return head==NULL;
}
int front()
{
    if(head==NULL)return -1;
    return head->val;
}
};
int main() {
    Queue qu;
    qu.Enqueue(5);
    qu.Enqueue(6);
    qu.Enqueue(8);
    qu.Dequeue();
    qu.Enqueue(9);
    while (!qu.empty()) {
        cout << qu.front() << endl;
        qu.Dequeue();
    }
```

```
    return 0;
}
```




```
#include<iostream>
using namespace std;
class node{
    public:
    int val;
```

```
    node*left;
    node*right;
    node(int data)
    {
        val=data;
        left=right=NULL;
    }
};
class BST{
    public:
    node*root;
    BST()
    {
        root=NULL;
    }
};
bool SearchElement(node*&root,int sear)
{
    if(root==NULL)
    {
        return false;
    }
    if(root->val==sear)
    {
        return true;
    }
    if(root->val>sear)
    {
        return SearchElement(root->left,sear);
    }
    if(root->val<sear)
    {
        return SearchElement(root->right,sear);
    }
}
void InsertionElement(node*&root,int val)
{
    node*newNode=new node(val);
    if(root==NULL)
    {
        root=newNode;
        return;
    }
    node*curr=root;
    while(true)
    {
        if(curr->val>val)
        {
```

```
            if(curr->left==NULL)
            {
              curr->left=newNode;
              return;
            }
            curr=curr->left;
          }
          else{
              if(curr->right==NULL)
              {
                  curr->right=newNode;
                  return;
              }
              curr=curr->right;
          }
      }
}
void inorderTra(node*rootnode)
{
    if(rootnode==NULL)
    {
        return;
    }
    inorderTra(rootnode->left);
    cout<<rootnode->val<<" ";
    inorderTra(rootnode->right);
}
int main()
{
    BST bst1;
    InsertionElement(bst1.root,3);
    InsertionElement(bst1.root,1);
    InsertionElement(bst1.root,4);

    inorderTra(bst1.root);
    cout << endl;
    cout << (SearchElement(bst1.root, 3) ? "Element found" : "Element not found") << endl;
    return 0;
}
```

SearchInBST.cpp > main()

```
#include<iostream>
using namespace std;
class node{
    public:
```

```
1 3 4
Element found
PS D:\DSA program\sumofseq.cpp\Tree>
```