**ADVANCED PROGRAMMING-II**

**ASSIGNMENT-05**

**Q1. Implement Queue using Stack:**

**Code:**

class MyQueue {

private:

    stack<int> s1, s2;

public:

    MyQueue() {}

    void push(int x) {

        s1.push(x);

    }

    int pop() {

        if (s2.empty()) {

            while (!s1.empty()) {

                s2.push(s1.top());

                s1.pop();

            }

        }

        int front = s2.top();

        s2.pop();

        return front;

    }

    int peek() {

        if (s2.empty()) {

            while (!s1.empty()) {

                s2.push(s1.top());

                s1.pop();

            }

        }

        return s2.top();

    }

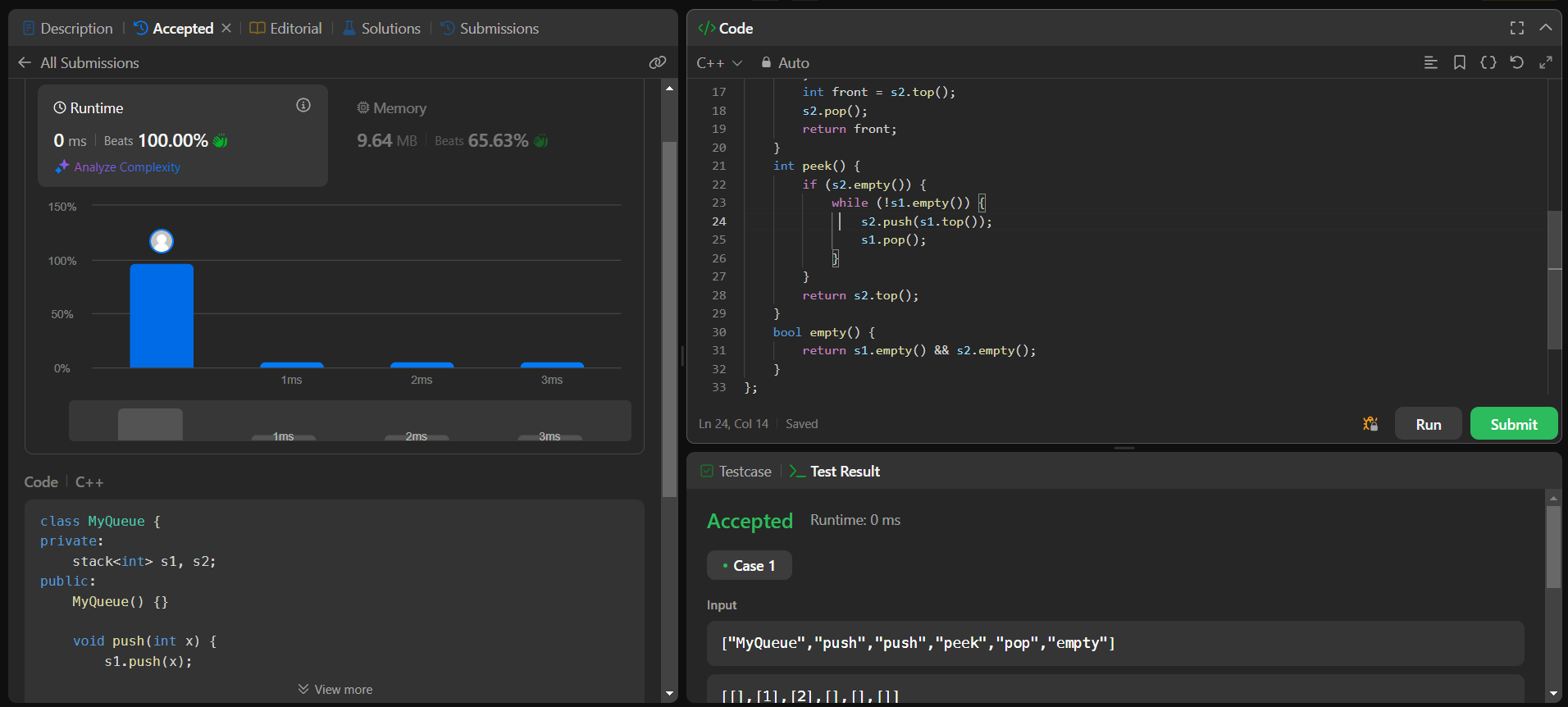
    bool empty() {

        return s1.empty() && s2.empty();

    }

};

**Screenshot:**

****

**Q2.** [**Min Stack**](https://leetcode.com/problems/min-stack/)**:**

**Code:**

class MinStack {

private:

    stack<int> mainStack;

    stack<int> minStack;

public:

    MinStack() {}

    void push(int val) {

        mainStack.push(val);

        if (minStack.empty() || val <= minStack.top()) {

            minStack.push(val);      }    }

    void pop() {

        if (mainStack.top() == minStack.top()) {

            minStack.pop(); }

        mainStack.pop();   }

    int top() {

        return mainStack.top();

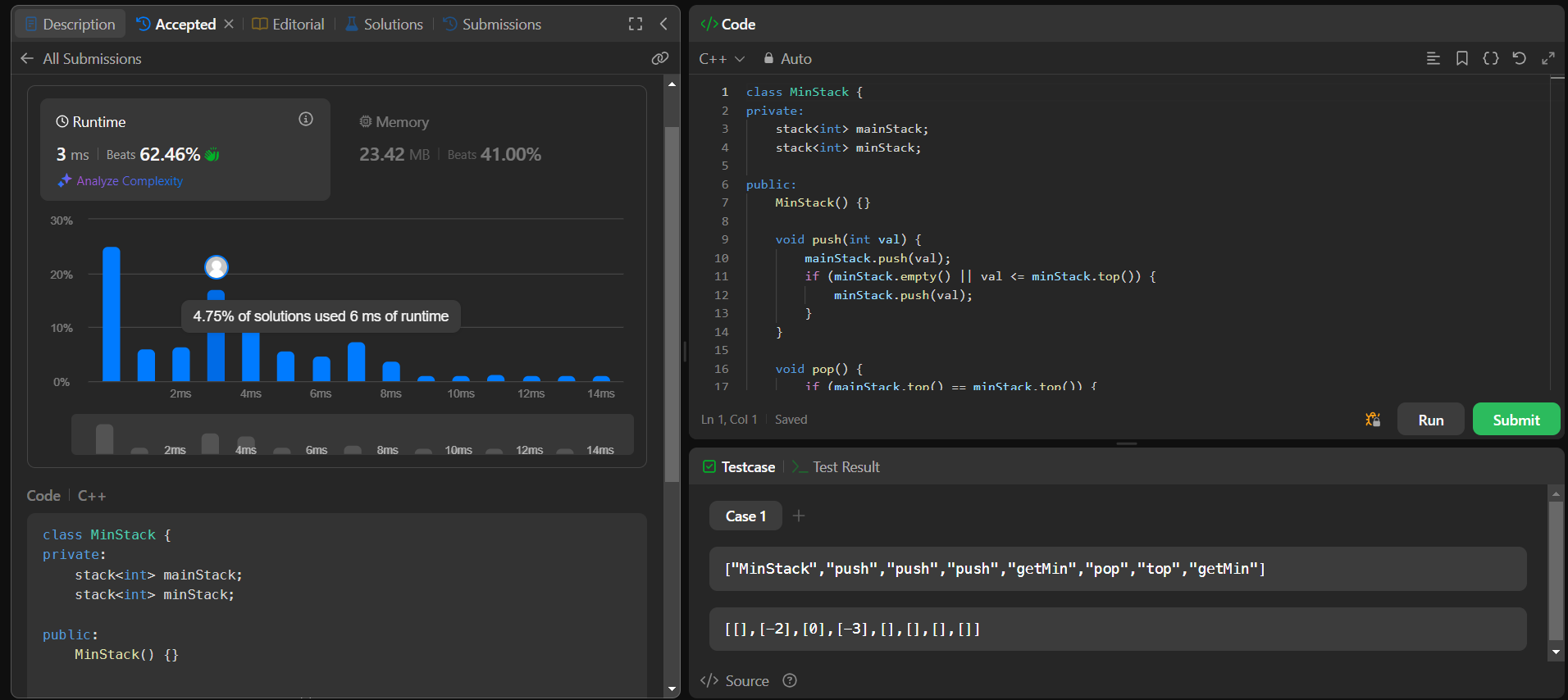
    }

    int getMin() {

        return minStack.top();

    }};

**Screenshot:**



Q3. [**Implement Stack using Queues**](https://leetcode.com/problems/implement-stack-using-queues/):

**Code:**

class MyStack {

private:

    queue<int> q1, q2;

public:

    MyStack() {}

    void push(int x) {

        q2.push(x);

        while (!q1.empty()) {

            q2.push(q1.front());

            q1.pop();

        }

        swap(q1, q2);

    }

    int pop() {

        int topElement = q1.front();

        q1.pop();

        return topElement;

    }

    int top() {

        return q1.front();

    }

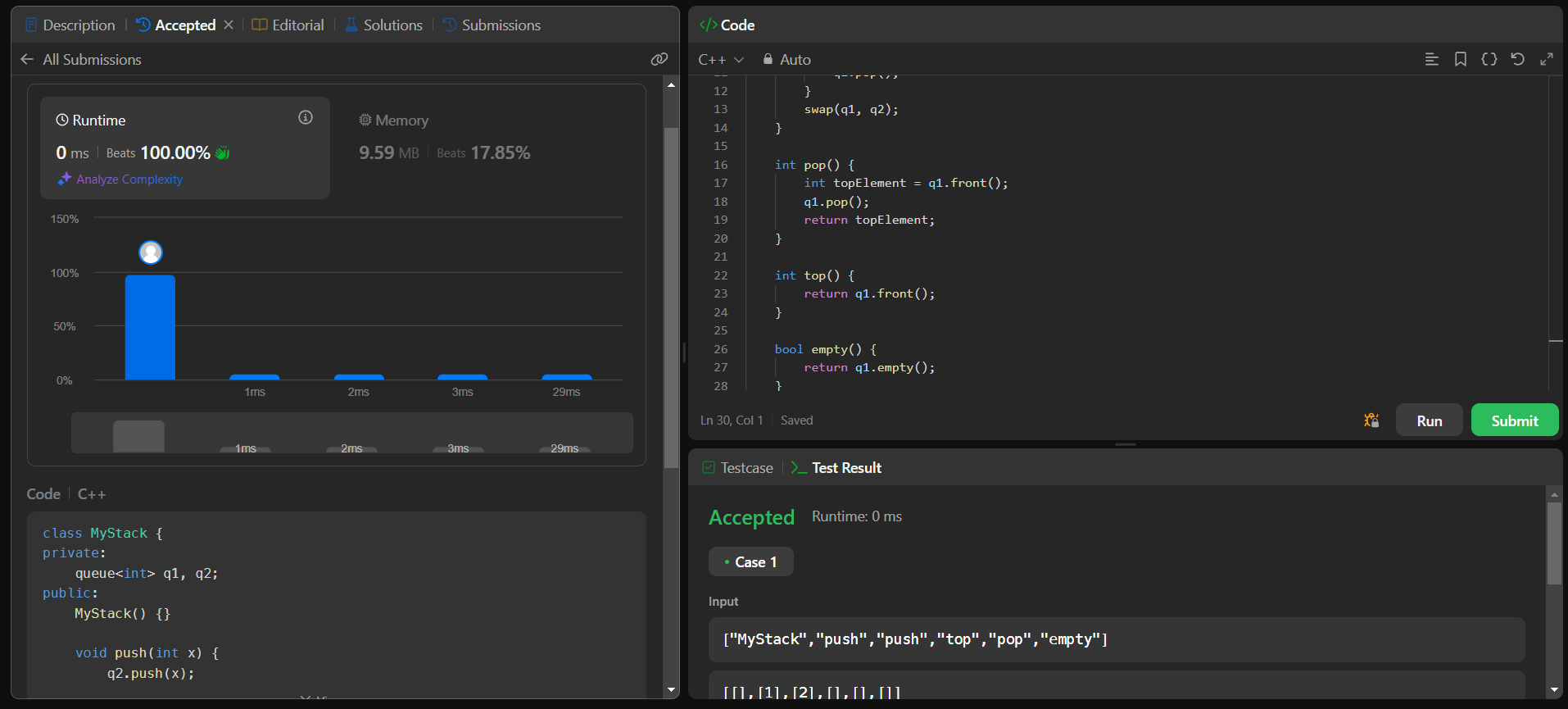
    bool empty() {

        return q1.empty();

    }

};

**Screenshot:**

****

**Q4.** [**Design Circular Queue**](https://leetcode.com/problems/design-circular-queue/)**:**

**Code:**

class MyCircularQueue {

private:

    vector<int> queue;

    int front, rear, size, capacity;

public:

    MyCircularQueue(int k) {

        queue.resize(k);

        capacity = k;

        size = 0;   front = 0;    rear = -1;   }

    bool enQueue(int value) {

        if (isFull()) return false;

        rear = (rear + 1) % capacity;

        queue[rear] = value;

        size++;

        return true;    }

    bool deQueue() {

        if (isEmpty()) return false;

        front = (front + 1) % capacity;

        size--;

        return true;    }

    int Front() {

        return isEmpty() ? -1 : queue[front];    }

    int Rear() {

        return isEmpty() ? -1 : queue[rear];    }

    bool isEmpty() {

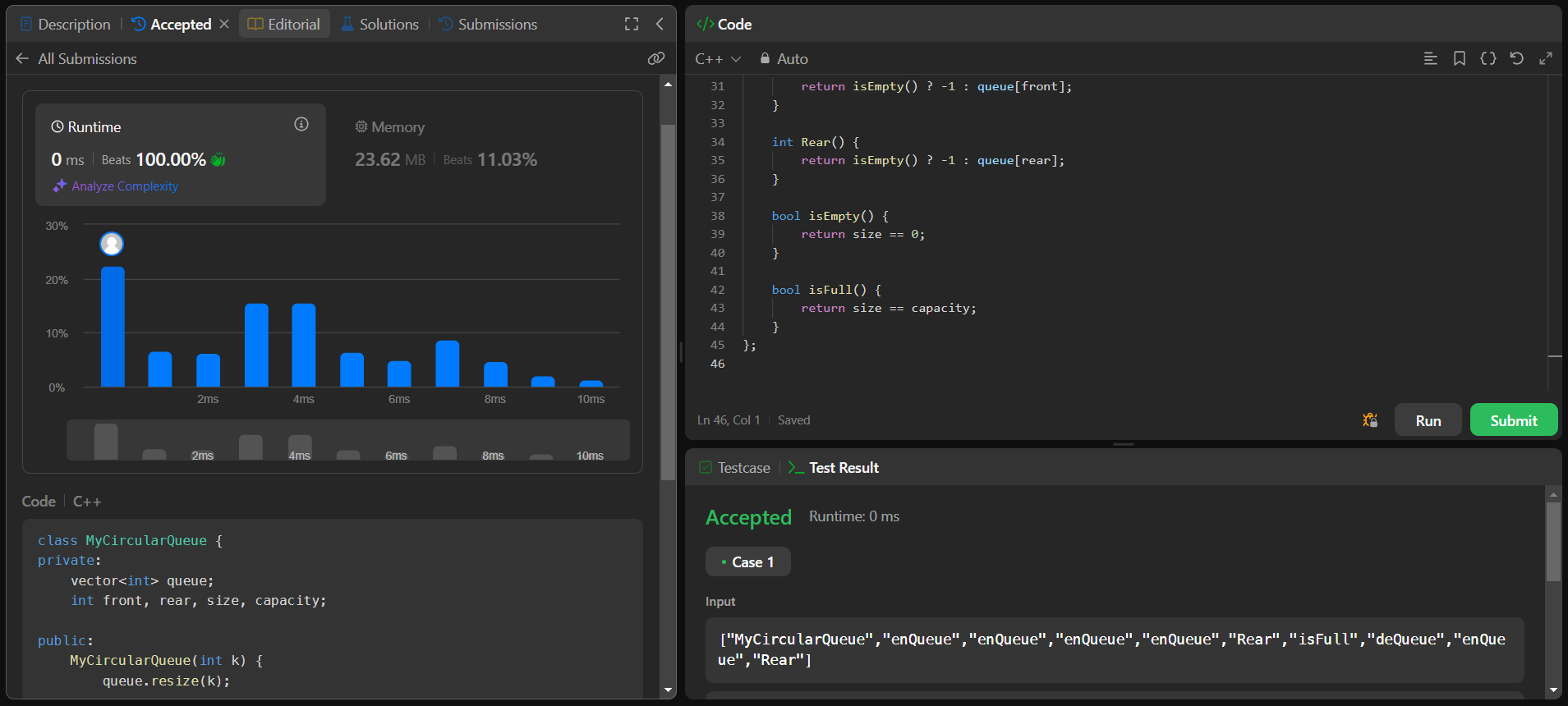
        return size == 0;    }

    bool isFull() {

        return size == capacity;    }

};

**Screenshot:**

****

**Q5.**  [**Kth Largest Element in an Array**](https://leetcode.com/problems/kth-largest-element-in-an-array/)**:**

**Code:**

#include <vector>

#include <queue>

using namespace std;

class Solution {

public:

    int findKthLargest(vector<int>& nums, int k) {

        priority\_queue<int, vector<int>, greater<int>> minHeap;

        for (int num : nums) {

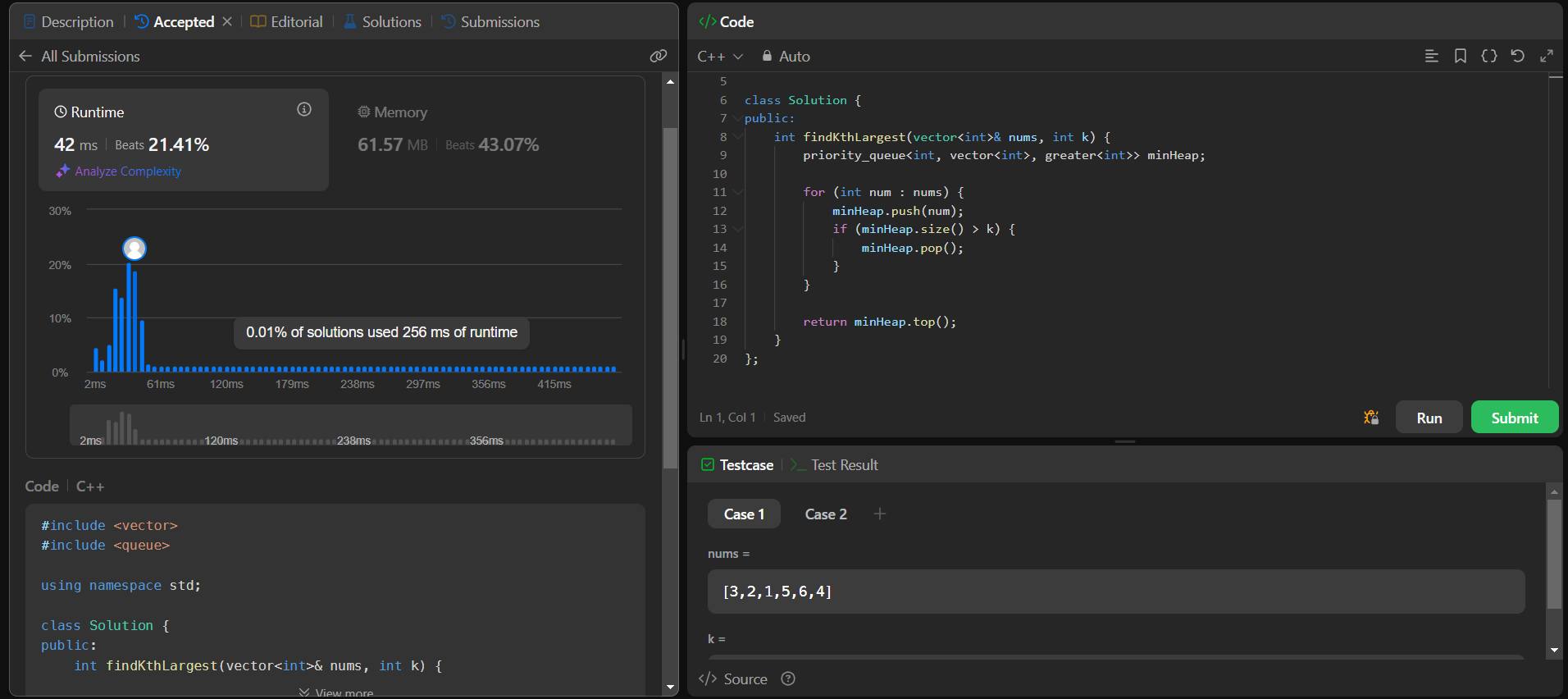
            minHeap.push(num);

            if (minHeap.size() > k) {

                minHeap.pop();     }

        return minHeap.top();    }};

**Screenshot:**

****

**Q6.** [**Design a Stack With Increment Operation**](https://leetcode.com/problems/design-a-stack-with-increment-operation/)**:**

**Code:**

class CustomStack {

private:

    vector<int> stack;

    int maxSize;

public:

    CustomStack(int maxSize) {

        this->maxSize = maxSize;

    }

    void push(int x) {

        if (stack.size() < maxSize) {

            stack.push\_back(x);

        }

    }

    int pop() {

        if (stack.empty()) return -1;

        int topElement = stack.back();

        stack.pop\_back();

        return topElement;

    }

    void increment(int k, int val) {

        int limit = min(k, (int)stack.size());

        for (int i = 0; i < limit; i++) {

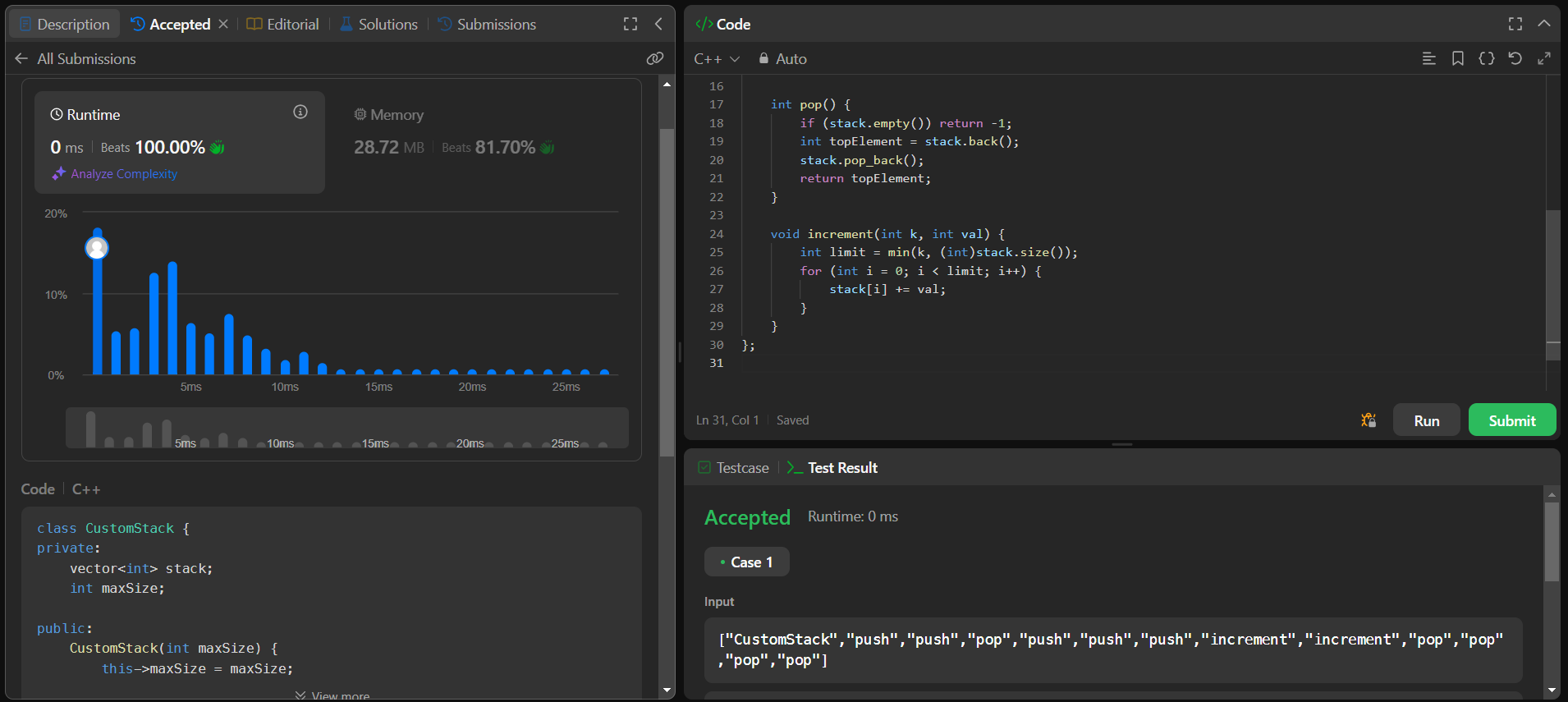
            stack[i] += val;

        }

    }

};

**Screenshot:**



**Q7.** [**Design Front Middle Back Queue**](https://leetcode.com/problems/design-front-middle-back-queue/)**:**

#include <deque>

class FrontMiddleBackQueue {

private:

    std::deque<int> left, right;

    void balance() {

        if (left.size() > right.size()) {

            right.push\_front(left.back());

            left.pop\_back();

        } else if (right.size() > left.size() + 1) {

            left.push\_back(right.front());

            right.pop\_front();

        }    }

public:

    FrontMiddleBackQueue(

    void pushFront(int val) {

        left.push\_front(val);

        balance();

    }

    void pushMiddle(int val) {

        if (left.size() < right.size()) {

            left.push\_back(val);

        } else {

            right.push\_front(val);     }

        balance();  }

    void pushBack(int val) {

        right.push\_back(val);

        balance();    }

    int popFront() {

        if (left.empty() && right.empty()) return -1;

        int val;

        if (!left.empty()) {

            val = left.front();

            left.pop\_front();

        } else {

            val = right.front();

            right.pop\_front();

        }

        balance();

        return val;   }

    int popMiddle() {

        if (left.empty() && right.empty()) return -1;

        int val;

        if (left.size() == right.size()) {

            val = left.back();

            left.pop\_back();

        } else {

            val = right.front();

            right.pop\_front();  }

        balance();

        return val;

    }

    int popBack() {

        if (right.empty()) return -1;

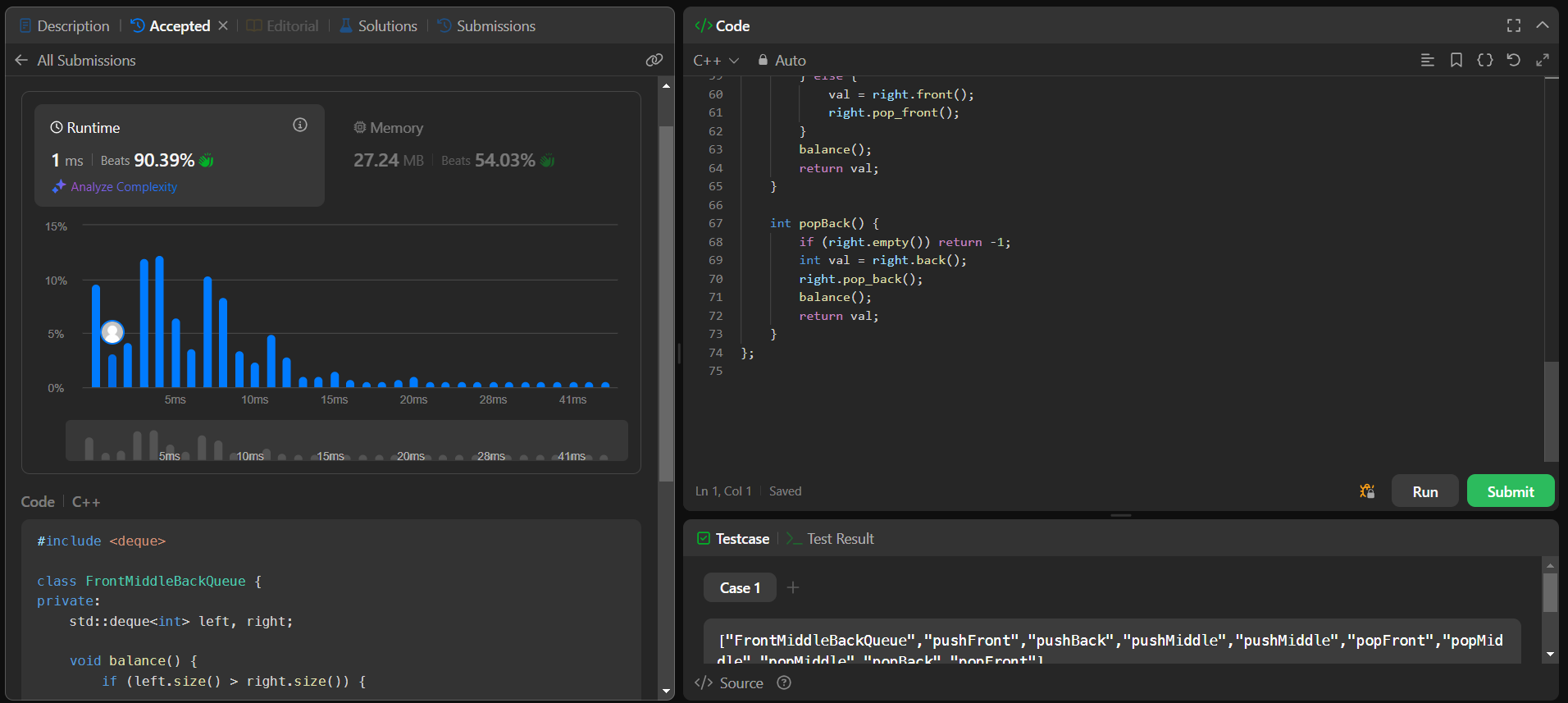
        int val = right.back();

        right.pop\_back();

        balance();

        return val;    }};

**Screenshot:**



**Q8.** [**Merge k Sorted Lists**](https://leetcode.com/problems/merge-k-sorted-lists/)**:**

**Code:**

class Solution {

public:

    struct Compare {

        bool operator()(ListNode\* a, ListNode\* b) {

            return a->val > b->val;

    };

    ListNode\* mergeKLists(vector<ListNode\*>& lists) {

        priority\_queue<ListNode\*, vector<ListNode\*>, Compare> minHeap;

        for (auto list : lists) {

            if (list) minHeap.push(list);

        }

        ListNode\* dummy = new ListNode(-1);

        ListNode\* tail = dummy;

        while (!minHeap.empty()) {

            ListNode\* node = minHeap.top();

            minHeap.pop();

            tail->next = node;

            tail = node;

            if (node->next) {

                minHeap.push(node->next);

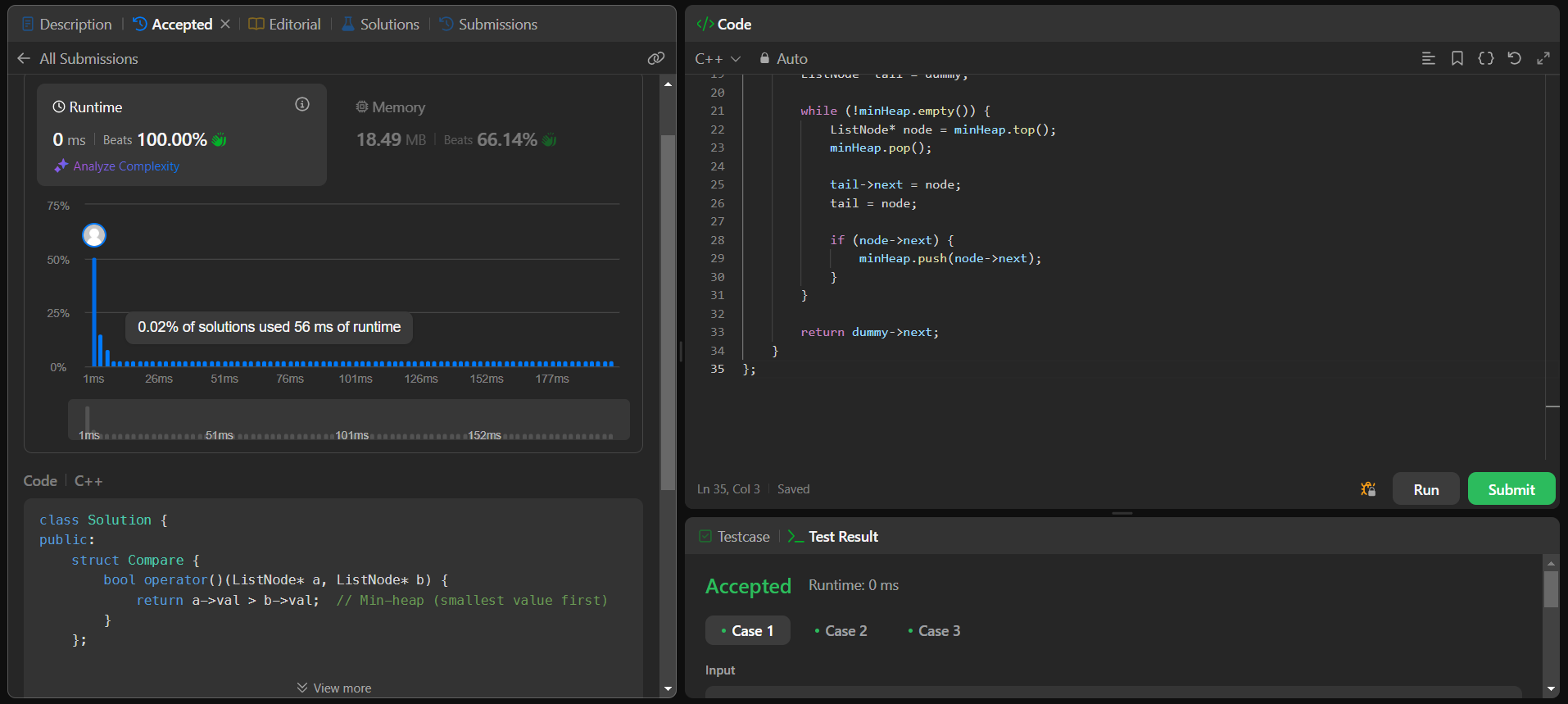
            }

        }

        return dummy->next;

    }

};

**Screenshot:**

**Q9.** [**Convert Sorted List to Binary Search Tree**](https://leetcode.com/problems/convert-sorted-list-to-binary-search-tree/)**:**

**Code:**

class Solution {

public:

    TreeNode\* sortedListToBST(ListNode\* head) {

        if (!head) return nullptr;

        if (!head->next) return new TreeNode(head->val);

        ListNode \*slow = head, \*fast = head, \*prev = nullptr;

        while (fast && fast->next) {

            prev = slow;

            slow = slow->next;

            fast = fast->next->next;

        }

        prev->next = nullptr;

        TreeNode\* root = new TreeNode(slow->val);

        root->left = sortedListToBST(head);

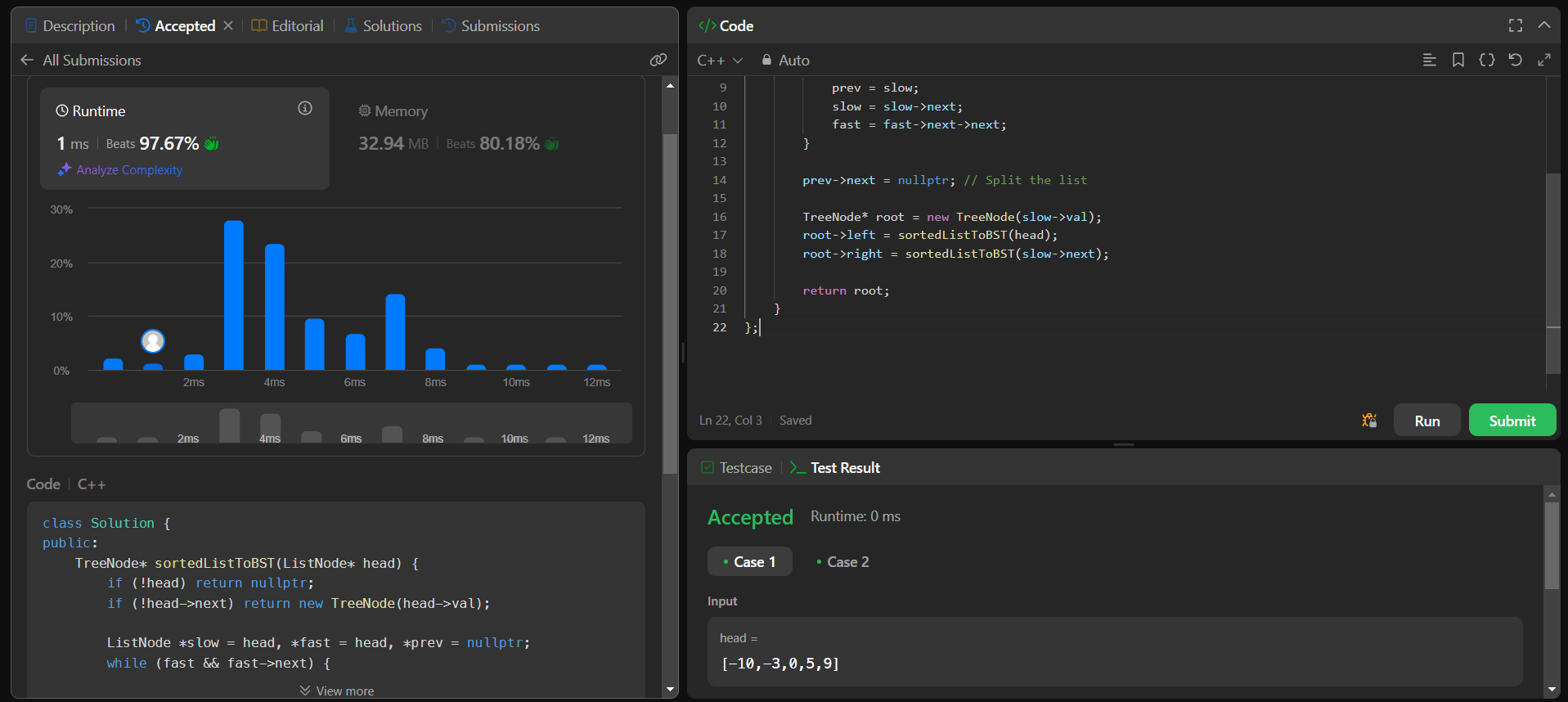
        root->right = sortedListToBST(slow->next);

        return root;

    }

};

**Screenshot:**

****

**Q10.** [**Balance a Binary Search Tree**](https://leetcode.com/problems/balance-a-binary-search-tree/)**:**

**Code:**

class Solution {

public:

    void inorder(TreeNode\* root, std::vector<int>& vals) {

        if (!root) return;

        inorder(root->left, vals);

        vals.push\_back(root->val);

        inorder(root->right, vals);

    }

    TreeNode\* buildBST(std::vector<int>& vals, int left, int right) {

        if (left > right) return nullptr;

        int mid = left + (right - left) / 2;

        TreeNode\* root = new TreeNode(vals[mid]);

        root->left = buildBST(vals, left, mid - 1);

        root->right = buildBST(vals, mid + 1, right);

        return root;  }

    TreeNode\* balanceBST(TreeNode\* root) {

        std::vector<int> vals;

        inorder(root, vals);

        return buildBST(vals, 0, vals.size() - 1);

    }};

**Screenshot:**

