AP-Worksheet-1

Name-Aditya Sharma Section-604-B

Uid-22BCS15081

Problem – 26

Code-

class Solution {

public:

    int removeDuplicates(vector<int>& nums) {

        int n= nums.size();

        int i=0;

       for(int j=1;j<n;j++){

        if(nums[j]!=nums[i]){

            i++;

            nums[i]=nums[j];

        }

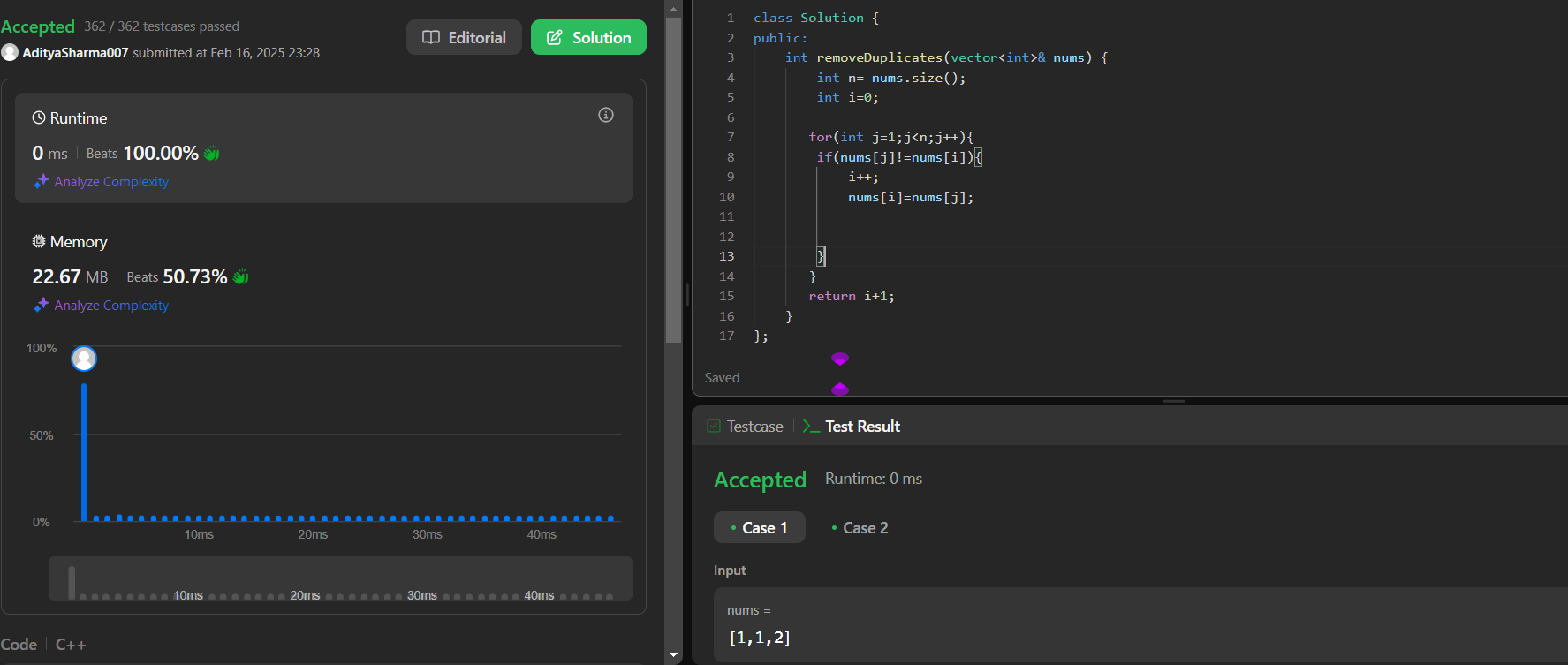
       }

       return i+1;

    }

};

Output-



Problem- Insertion Sort

Code-

int n = arr.size();

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

int j=i;

while(j>0 && arr[j-1]>arr[j]){

swap(arr[j-1],arr[j]);

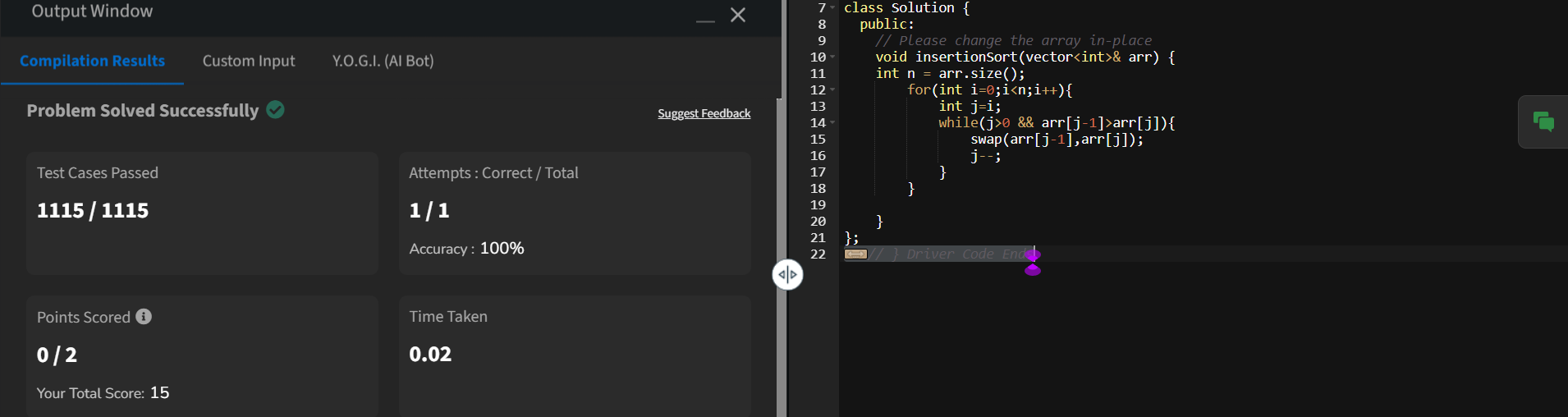
j--;

}

}

}

Output-



Problem- 217

Code-

class Solution {

public:

    bool containsDuplicate(vector<int>& nums) {

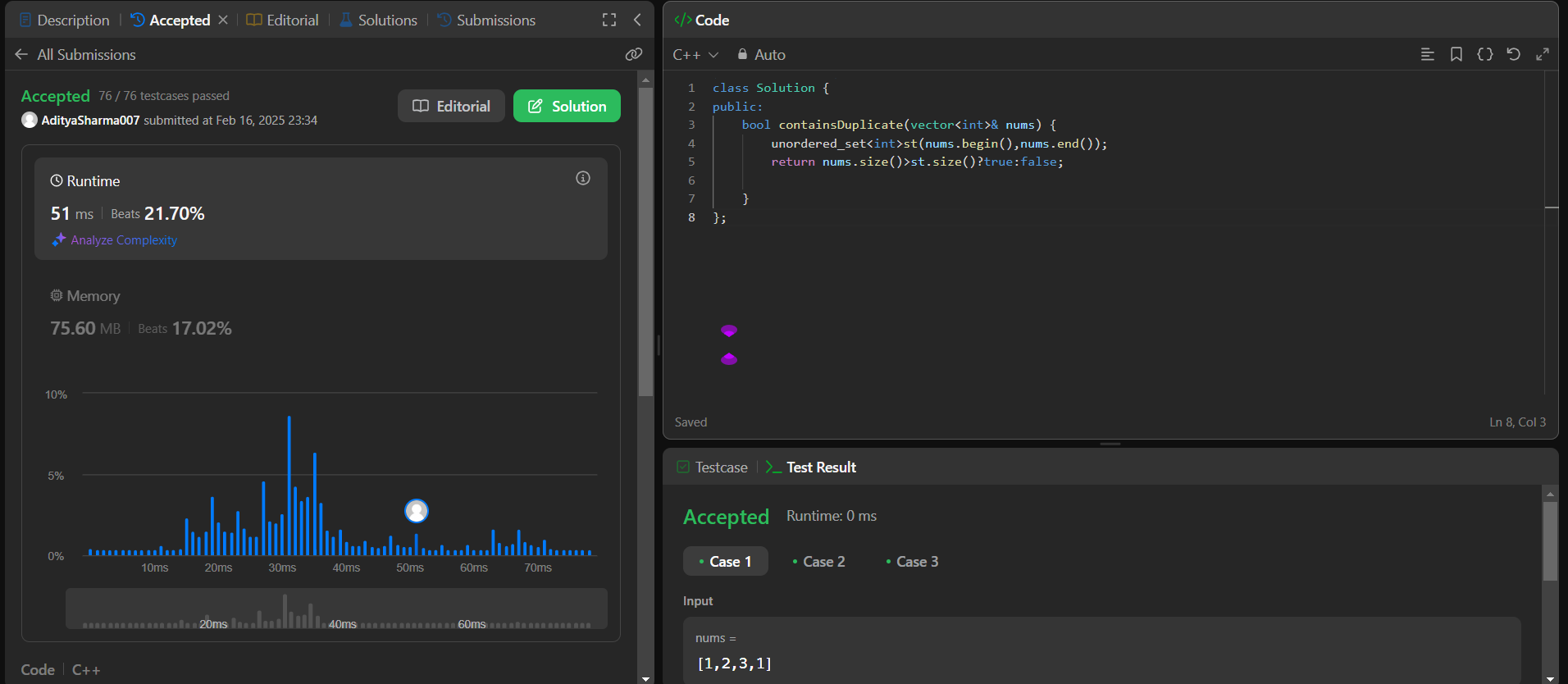
        unordered\_set<int>st(nums.begin(),nums.end());

        return nums.size()>st.size()?true:false;

    }

};

Output-



Problem- 1

Code-

class Solution {

public:

    vector<int> twoSum(vector<int>& nums, int target) {

        unordered\_map<int,int>mpp;

        int n=nums.size();

        for(int i=0;i<n;i++){

            int num=nums[i];

            int needed=target-num;

            if(mpp.find(needed)!=mpp.end()){

                return {mpp[needed],i};

                }

                mpp[num]=i;

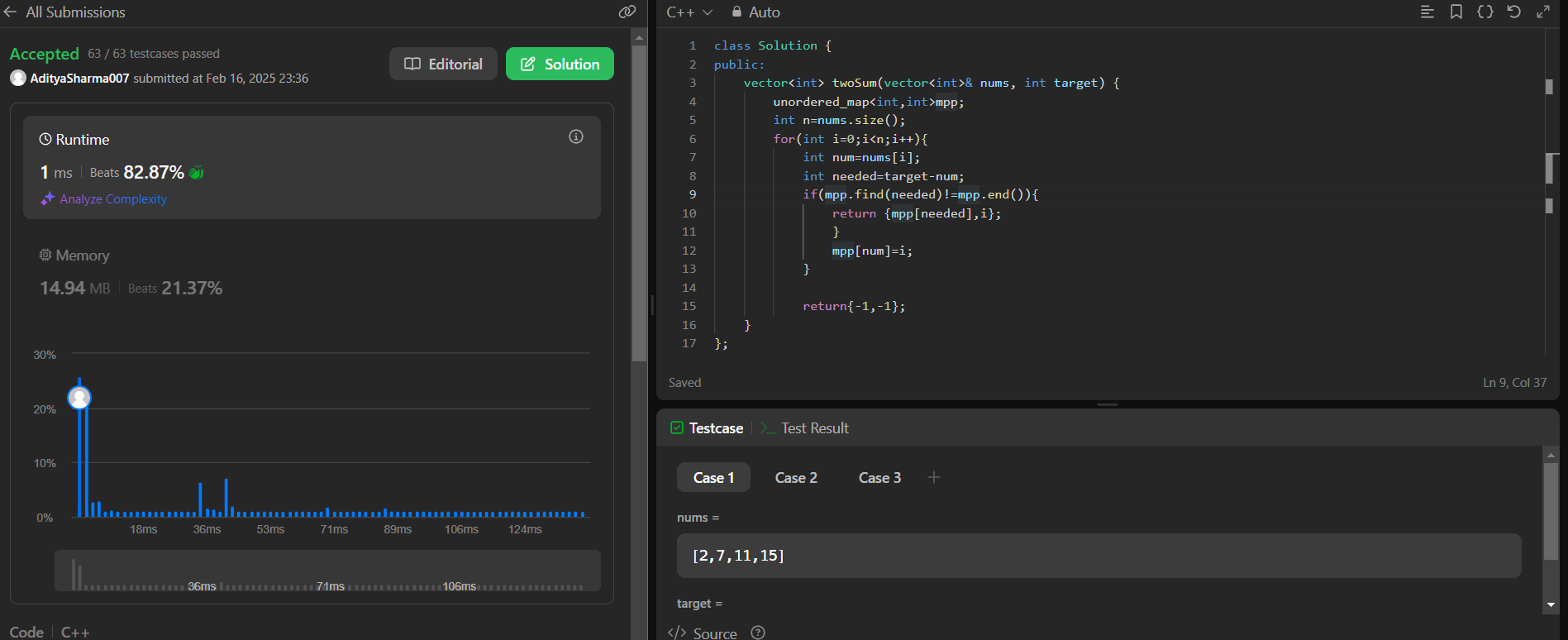
            }

            return{-1,-1};

    }

};

Output-



Probem- 55

Code-

class Solution {

public:

    bool canJump(vector<int>& nums) {

        int n = nums.size();

        int i = 0;

        if(n==1){

            return true;

        }

        while (i < n) {

            if (i + nums[i]>= n - 1) {

                return true;

            }

            if (nums[i] == 0) {

                return false;

            }

            i = i + nums[i];

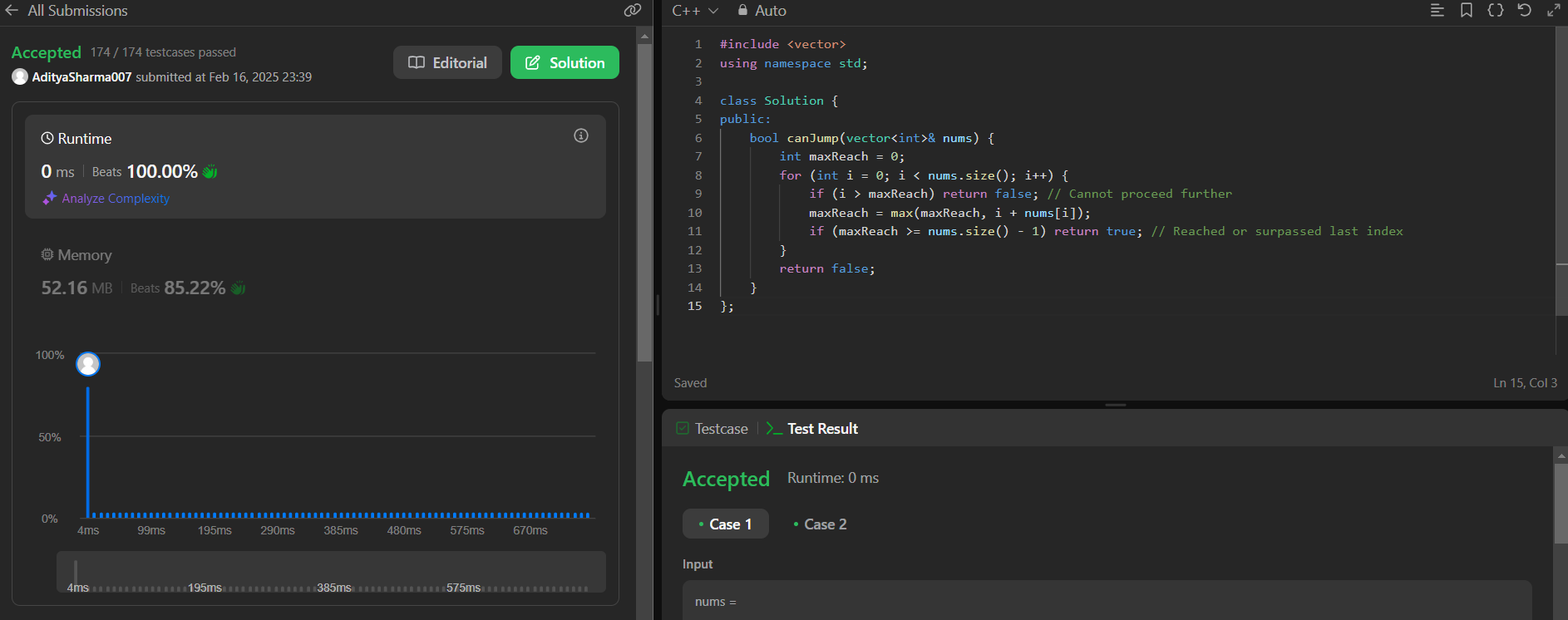
        }

        return false;

    }

};

Output-



Problem- 169

Code-

class Solution {

public:

    int majorityElement(vector<int>& nums) {

        int count=0;

        int n=nums.size();

        int el;

        for(int i=0;i<n;i++){

            if(count==0){

                el=nums[i];

            }

            if(nums[i]==el)count++;

            else count--;

        }

        int count1;

        for(int i=0;i<n;i++){

            if(nums[i]==el){

                count1++;

            }

        }

        if(count1>n/2){

            return el;

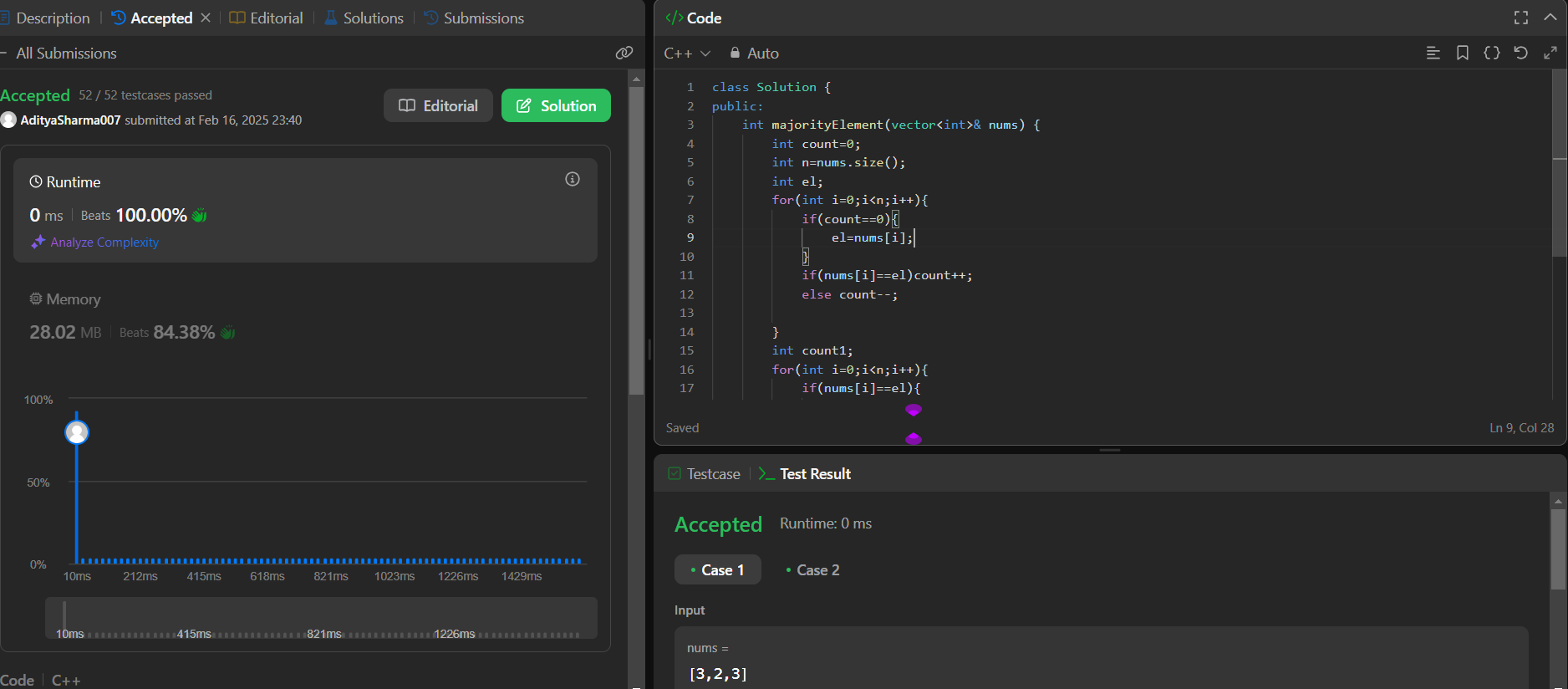
        }

        return -1;

    }

};

Output-



Problem-125

Code-

class Solution {

public:

    bool isPalindrome(string s) {

       int start=0;

       int end=s.size()-1;

       while(start<=end){

           if(!isalnum(s[start])){start++; continue;}

           if(!isalnum(s[end])){end--;continue;}

           if(tolower(s[start])!=tolower(s[end]))return false;

           else{

               start++;

               end--;

           }

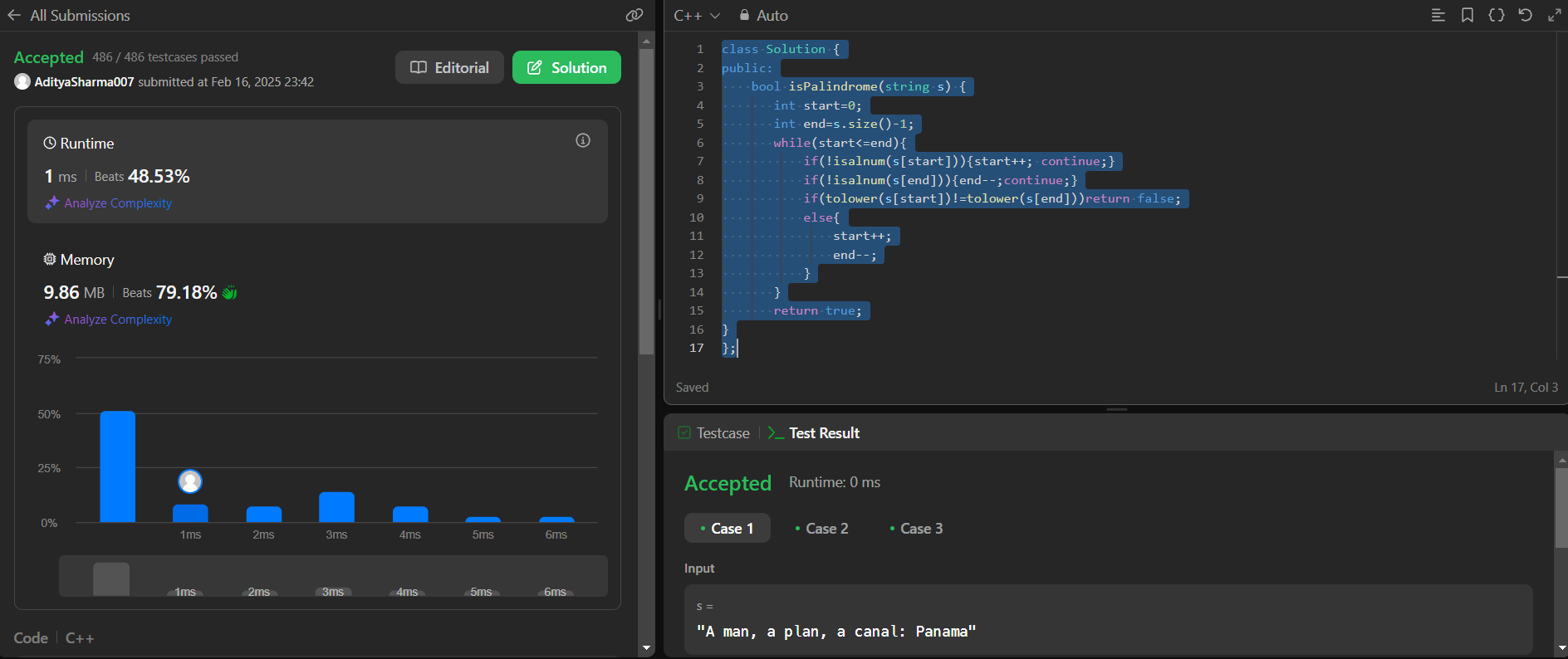
       }

       return true;

}

};

Output-



Problem-45

Code-

class Solution {

public:

    int jump(vector<int>& nums) {

        int jumps = 0, currentEnd = 0, farthest = 0;

        for (int i = 0; i < nums.size() - 1; i++) {

            farthest = max(farthest, i + nums[i]);

            if (i == currentEnd) {

                jumps++;

                currentEnd = farthest;

            }

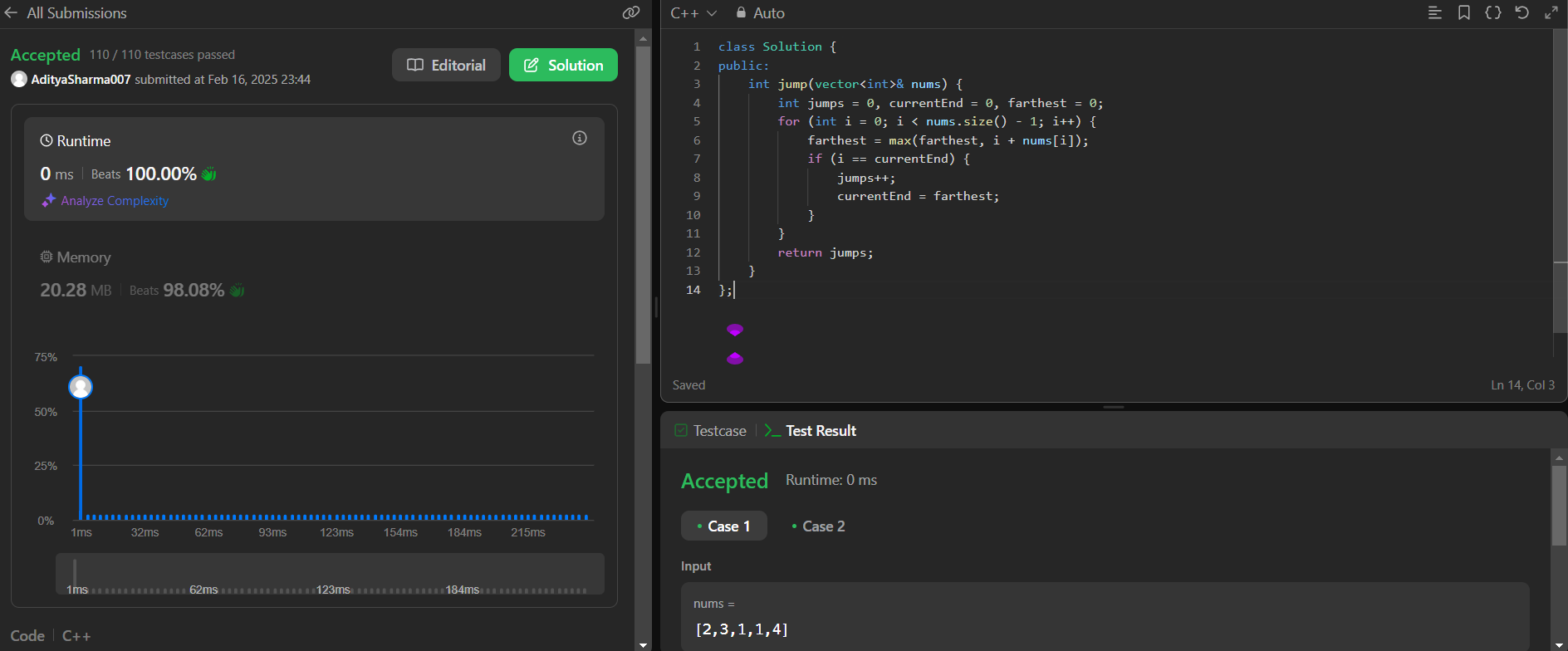
        }

        return jumps;

    }

};

Output-



Problem-15

Code-

class Solution {

public:

    vector<vector<int>> threeSum(vector<int>& nums) {

        int target = 0;

        sort(nums.begin(), nums.end());

        set<vector<int>> s;

        vector<vector<int>> output;

        for (int i = 0; i < nums.size(); i++){

            int j = i + 1;

            int k = nums.size() - 1;

            while (j < k) {

                int sum = nums[i] + nums[j] + nums[k];

                if (sum == target) {

                    s.insert({nums[i], nums[j], nums[k]});

                    j++;

                    k--;

                } else if (sum < target) {

                    j++;

                } else {

                    k--;

                }

            }

        }

        for(auto triplets : s)

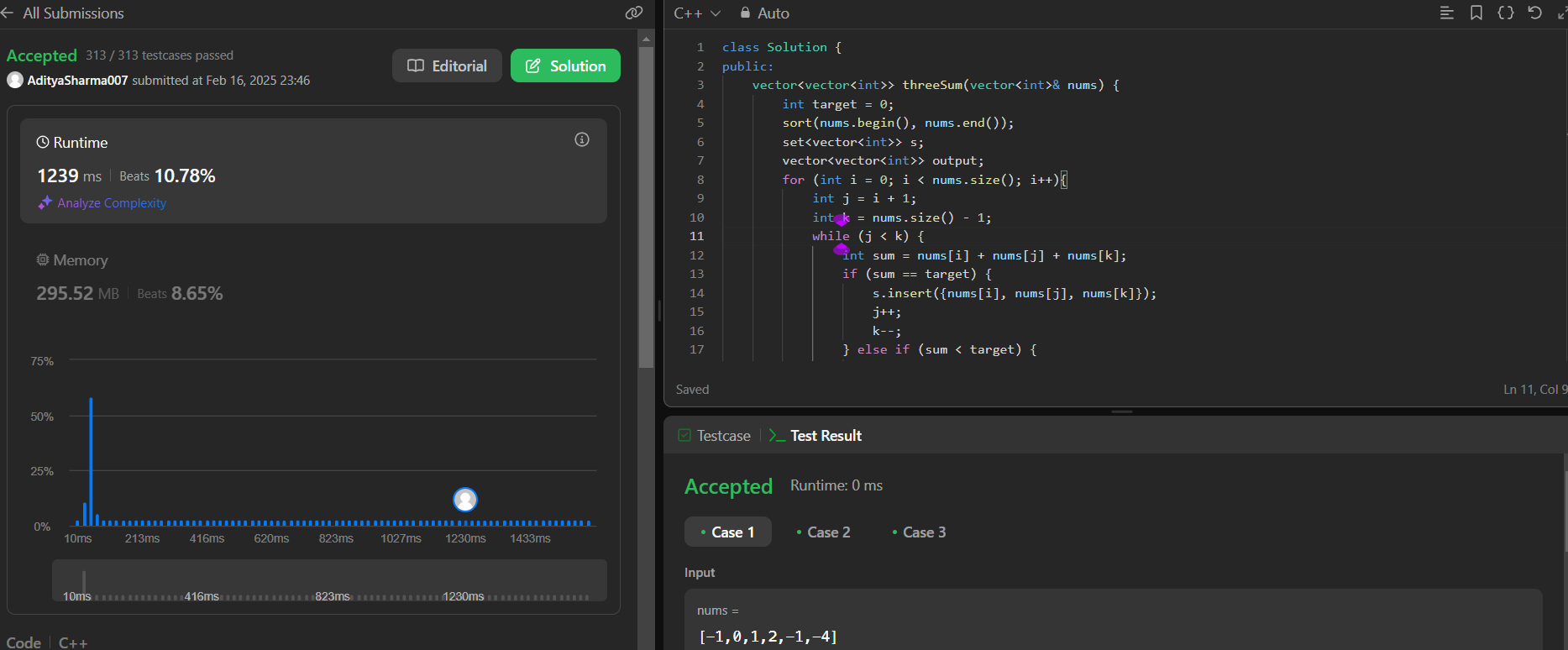
            output.push\_back(triplets);

        return output;

    }

};

Output-



Problem-73

Code-

class Solution {

public:

    void setZeroes(vector<vector<int>>& matrix) {

        int m=matrix.size();

        int n=matrix[0].size();

        vector<int>row(m,0);

        vector<int>col(n,0);

        for(int i=0;i<m;i++){

            for(int j=0;j<n;j++){

                if(matrix[i][j]==0){

                    row[i]=-10;

                    col[j]=-10;

                }

            }

        }

         for(int i=0;i<m;i++){

            for(int j=0;j<n;j++){

                if(row[i]==-10 || col[j]==-10){

                    matrix[i][j]=0;

                }

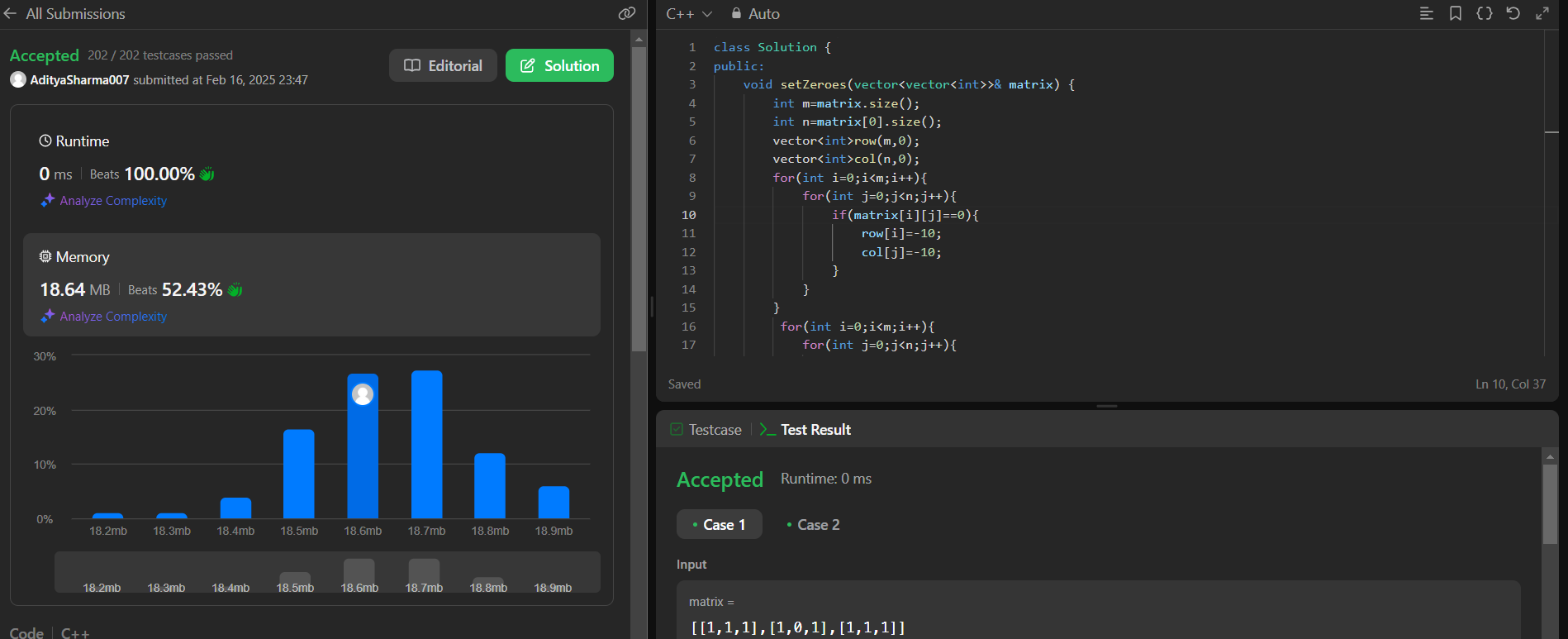
            }

        }

    }

};

Output-



Problem-3

Code-

class Solution {

public:

int lengthOfLongestSubstring(string s) {

int n = s.length();

int maxLength = 0;

unordered\_set<char> charSet;

int left = 0;

for (int right = 0; right < n; right++) {

if (charSet.count(s[right]) == 0) {

charSet.insert(s[right]);

maxLength = max(maxLength, right - left + 1);

} else {

while (charSet.count(s[right])) {

charSet.erase(s[left]);

left++;

}

charSet.insert(s[right]);

}

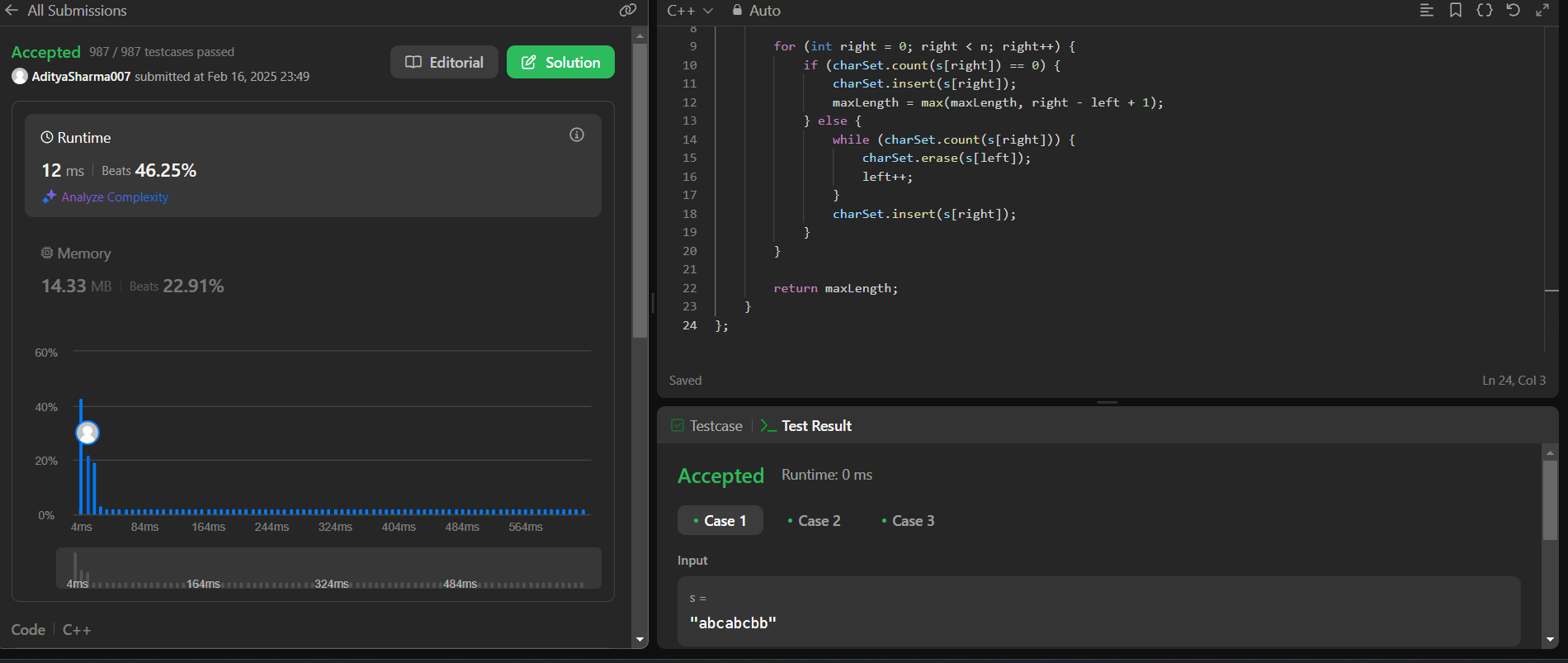
}

return maxLength;

}

};

Output-



Problem-287

Code-

class Solution {

public:

int findDuplicate(std::vector<int>& nums) {

int left = 1;

int right = nums.size() - 1;

while (left < right) {

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

int count = 0;

// Count the numbers less than or equal to mid

for (int num : nums) {

if (num <= mid) {

count++;

}

}

// If count is greater than mid, the duplicate lies in the left half

if (count > mid) {

right = mid;

} else { // Otherwise, it lies in the right half

left = mid + 1;

}

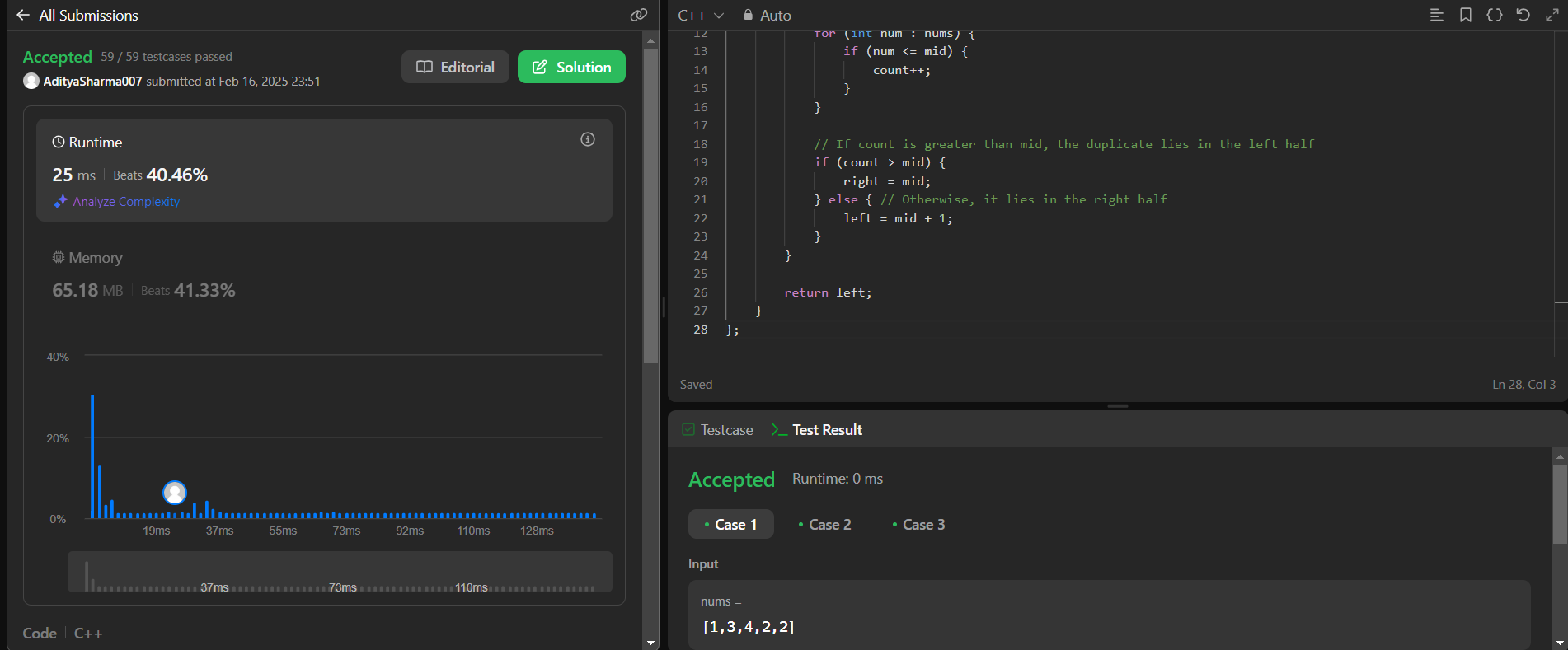
}

return left;

}

};

Output-



Problem- Print linklist

Code-

class Solution {

public:

void printList(Node \*head) {

Node\* temp=head;

while(temp!=NULL){

cout<<temp->data<<" ";

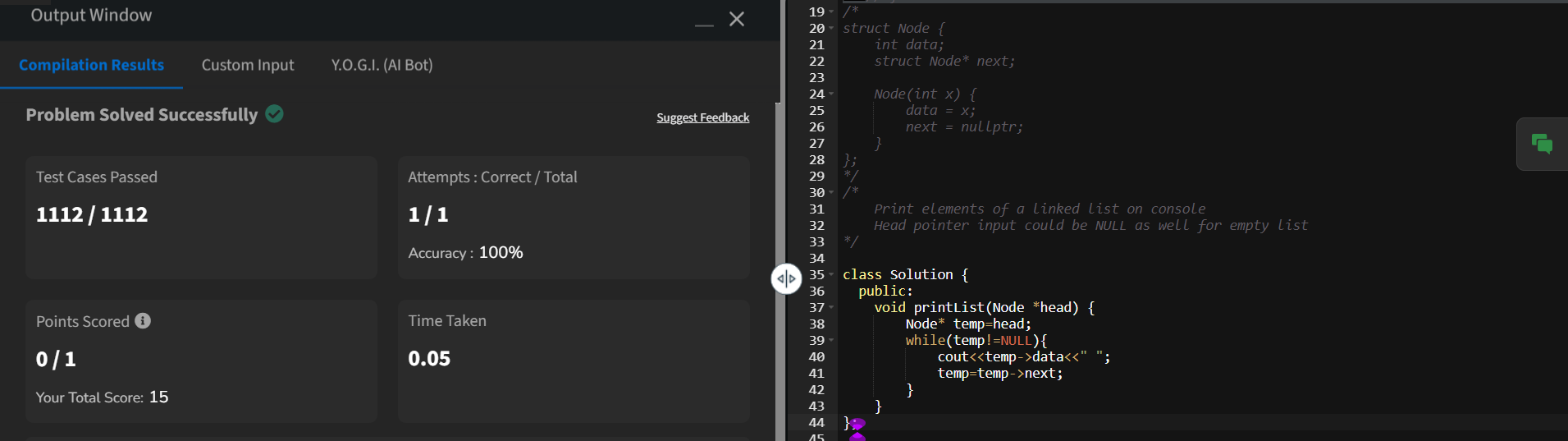
temp=temp->next;

}

}

};

Output-



Problem- 83

Code-

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

//edge case

if(head == NULL || head->next == NULL){

return head;

}

ListNode\* temp = head;

while(temp && temp->next){

if(temp->val == temp->next->val){

temp->next = temp->next->next;

}else{

temp = temp->next;

}

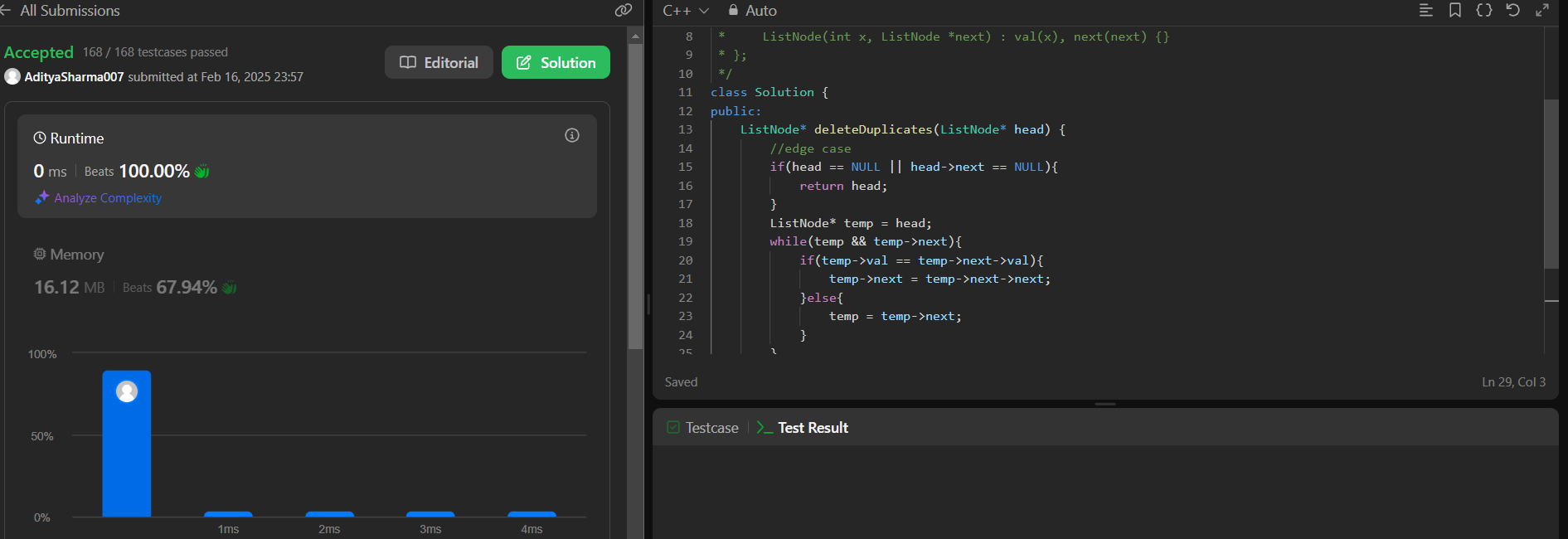
}

return head;

}

};

Output-



Problem- 206

Code-

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        // Initialize pointers

        ListNode\* prev = nullptr; // Previous node starts as NULL

        ListNode\* next = nullptr; // Next node

        ListNode\* curr = head;    // Current node starts at the head

        // Traverse the list

        while (curr != nullptr) {

            // Save the next node

            next = curr->next;

            // Reverse the link

            curr->next = prev;

            // Move pointers forward

            prev = curr; // Move prev to the current node

            curr = next; // Move curr to the next node

        }

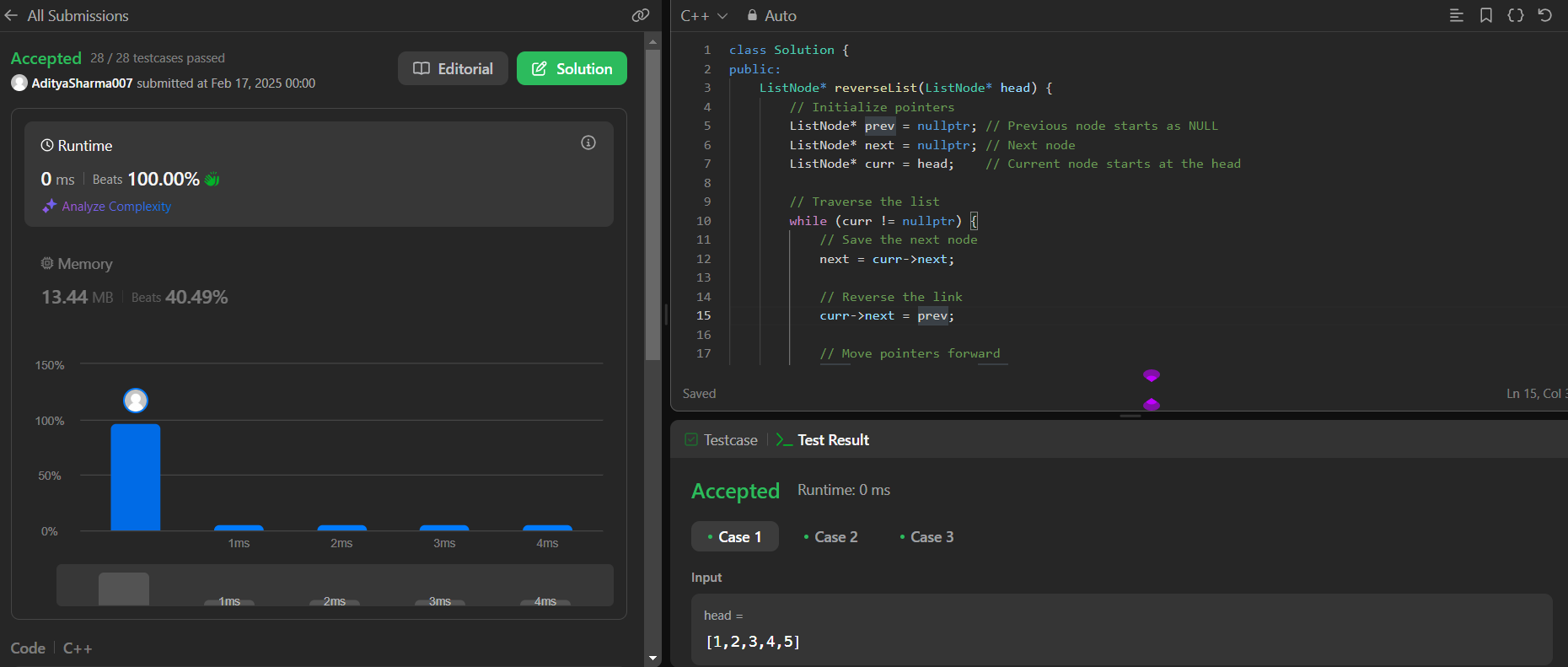
        // prev is now the new head of the reversed list

        return prev;

    }

};

Output-



Problem- 2095

Code-

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* deleteMiddle(ListNode\* head) {

if (!head || !head->next)

return NULL;

ListNode\* ans = new ListNode(0);

ans->next = head;

ListNode\* slow = ans;

ListNode\* fast = head;

while (fast != NULL && fast->next != NULL) {

slow = slow->next;

fast = fast->next->next;

}

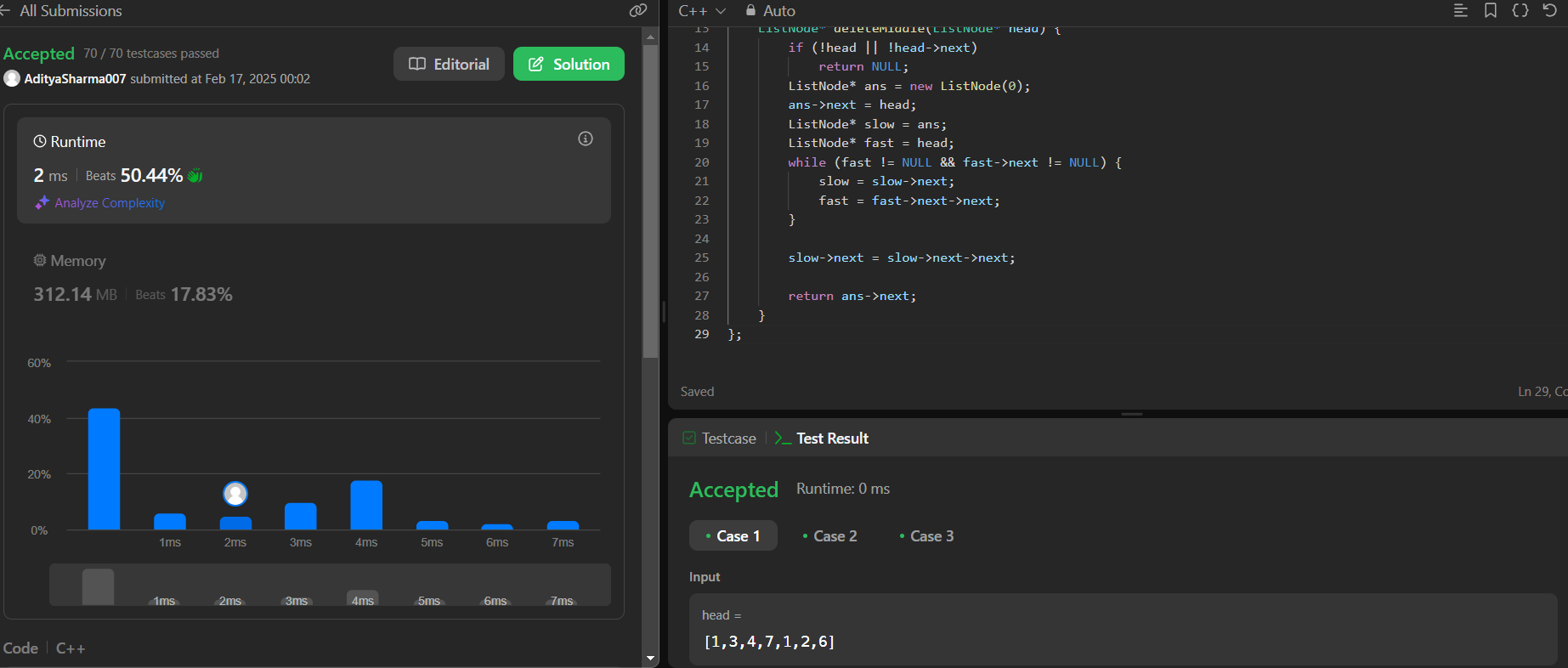
slow->next = slow->next->next;

return ans->next;

}

};

Output-



Problem- 21

Code-

#include <iostream>

using namespace std;

class Solution {

public:

ListNode\* mergeTwoLists(ListNode\* list1, ListNode\* list2) {

ListNode\* dummy = new ListNode(0);

ListNode\* current = dummy;

while (list1 != nullptr && list2 != nullptr) {

if (list1->val < list2->val) {

current->next = list1;

list1 = list1->next;

} else {

current->next = list2;

list2 = list2->next;

}

current = current->next;

}

// Attach remaining nodes

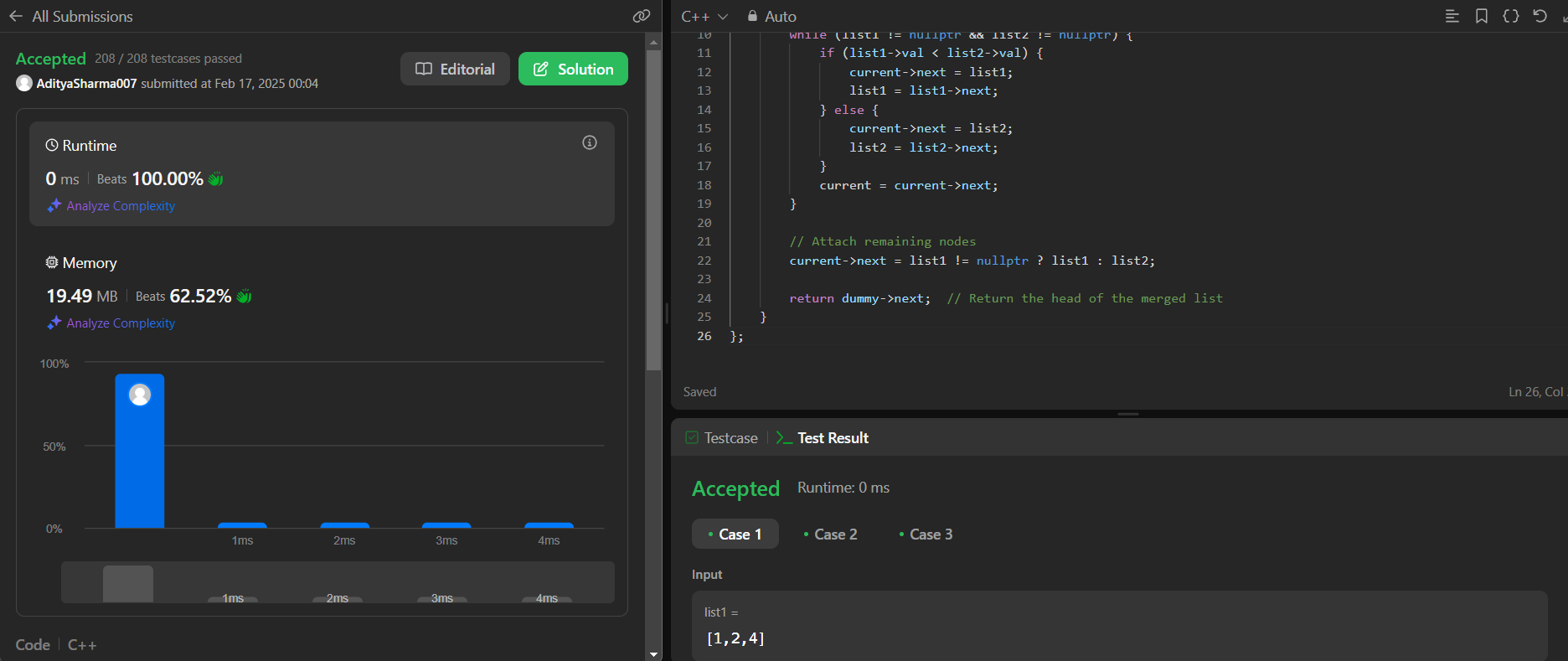
current->next = list1 != nullptr ? list1 : list2;

return dummy->next; // Return the head of the merged list

}

};

Output-



Problem-82

Code-

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

        // Edge case

        if (head == NULL || head->next == NULL) {

            return head;

        }

        ListNode\* dummy = new ListNode(0, head);

        ListNode\* prev = dummy;

        ListNode\* temp = head;

        while (temp) {

            if (temp->next != NULL && temp->next->val == temp->val) {

                while (temp->next != NULL && temp->next->val == temp->val)

                    temp = temp->next;

                     prev->next = temp->next;

            }

        else {

            prev = prev->next;

        }

            temp = temp->next;

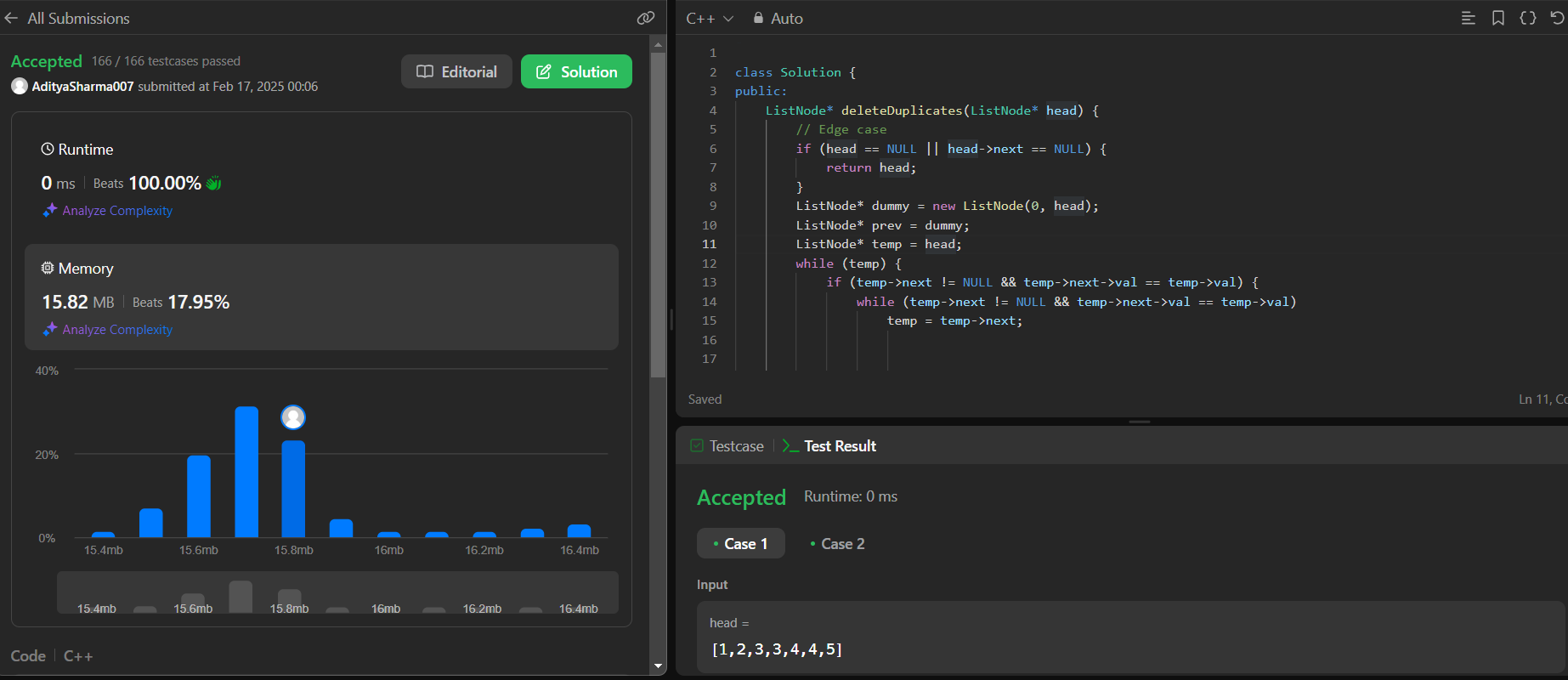
        }

        return dummy->next;

    }

};

Output-



Problem- 141

Code-

class Solution {

public:

bool hasCycle(ListNode\* head) {

if (head == NULL || head->next == NULL) {

return false;

}

ListNode\* slow = head;

ListNode\* fast = head->next;

while (fast != slow) {

if (fast->next == NULL || fast->next->next == NULL) {

return false;

}

slow = slow->next;

fast = fast->next->next;

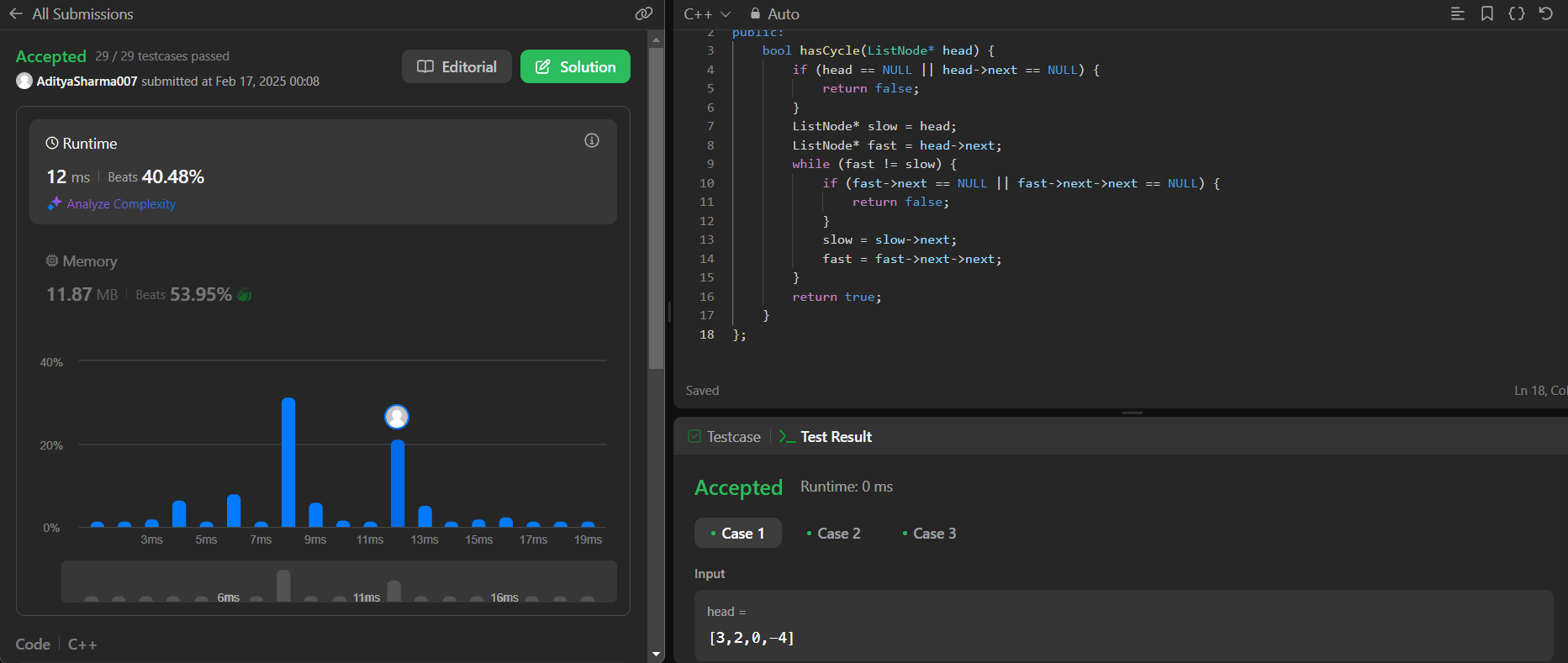
}

return true;

}

};

Output-



Problem- 92

Code-

class Solution {

public:

void reverse( ListNode\*& root,ListNode\*curr,ListNode \*pre){

if (!curr) return;

if (!curr->next) {

curr->next = pre;

root = curr;

return;

}

ListNode\* next = curr->next;

curr->next = pre;

reverse(root, next, curr);

}

ListNode\* reverseBetween(ListNode\* head, int left, int right) {

if (!head || left == right) return head;

ListNode dummy(0);

dummy.next = head;

ListNode\* pre = &dummy;

for (int i = 1; i < left; ++i) {

pre = pre->next;

}

ListNode\* leftNode = pre->next;

ListNode\* rightNode = leftNode;

for (int i = 0; i < right - left; ++i) {

rightNode = rightNode->next;

}

ListNode\* nextNode = rightNode->next;

rightNode->next = nullptr;

ListNode\* root = nullptr;

reverse(root, leftNode, nullptr);

pre->next = root;

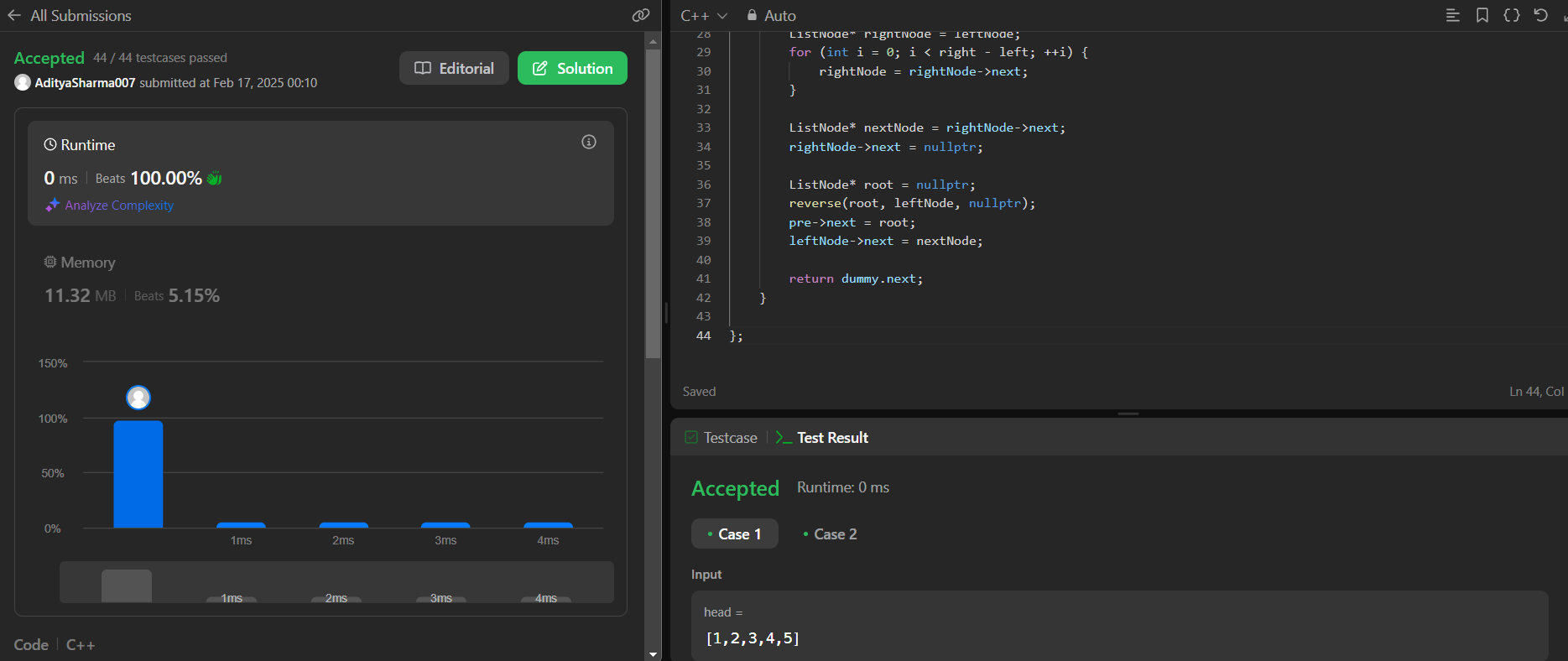
leftNode->next = nextNode;

return dummy.next;

}

};

Output-



Problem-61

Code-

class Solution {

public:

ListNode\* rotateRight(ListNode\* head, int k) {

if(!head || !head->next) return head;

int count = 0;

ListNode\* temp = head;

while (temp) {

count++;

temp = temp->next;

}

k = k % count;

if(k==0) return head;

temp = head;

for (int i = 1; i < count - k; i++) {

temp = temp->next;

}

ListNode\* pre = temp->next;

temp->next = NULL;

ListNode\* prevv = pre;

while (prevv->next) {

prevv = prevv->next;

}

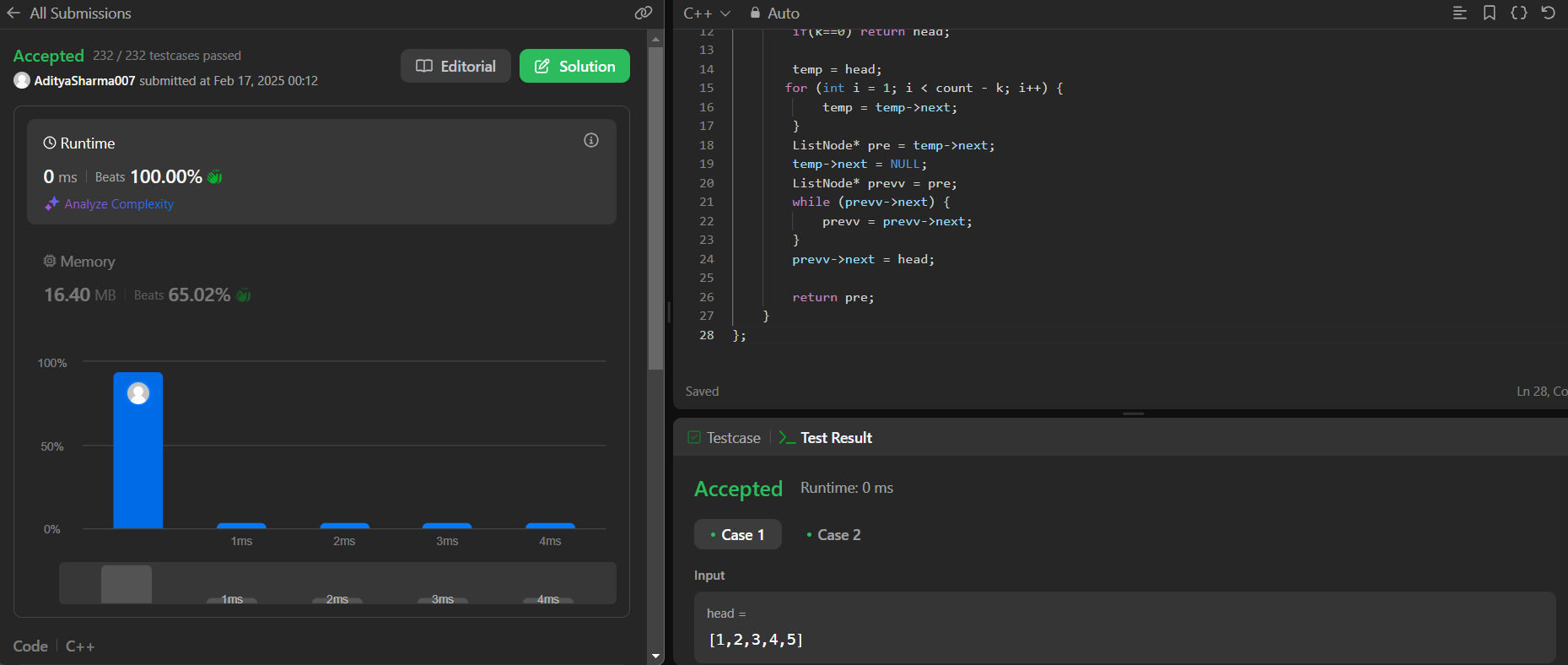
prevv->next = head;

return pre;

}

};

Output-



Problem-23

Code-

class Solution {

public:

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

if (lists.empty()) {

return nullptr;

}

return mergeKListsHelper(lists, 0, lists.size() - 1);

}

ListNode\* mergeKListsHelper(vector<ListNode\*>& lists, int start, int end) {

if (start == end) {

return lists[start];

}

if (start + 1 == end) {

return merge(lists[start], lists[end]);

}

int mid = start + (end - start) / 2;

ListNode\* left = mergeKListsHelper(lists, start, mid);

ListNode\* right = mergeKListsHelper(lists, mid + 1, end);

return merge(left, right);

}

ListNode\* merge(ListNode\* l1, ListNode\* l2) {

ListNode\* dummy = new ListNode(0);

ListNode\* curr = dummy;

while (l1 && l2) {

if (l1->val < l2->val) {

curr->next = l1;

l1 = l1->next;

} else {

curr->next = l2;

l2 = l2->next;

}

curr = curr->next;

}

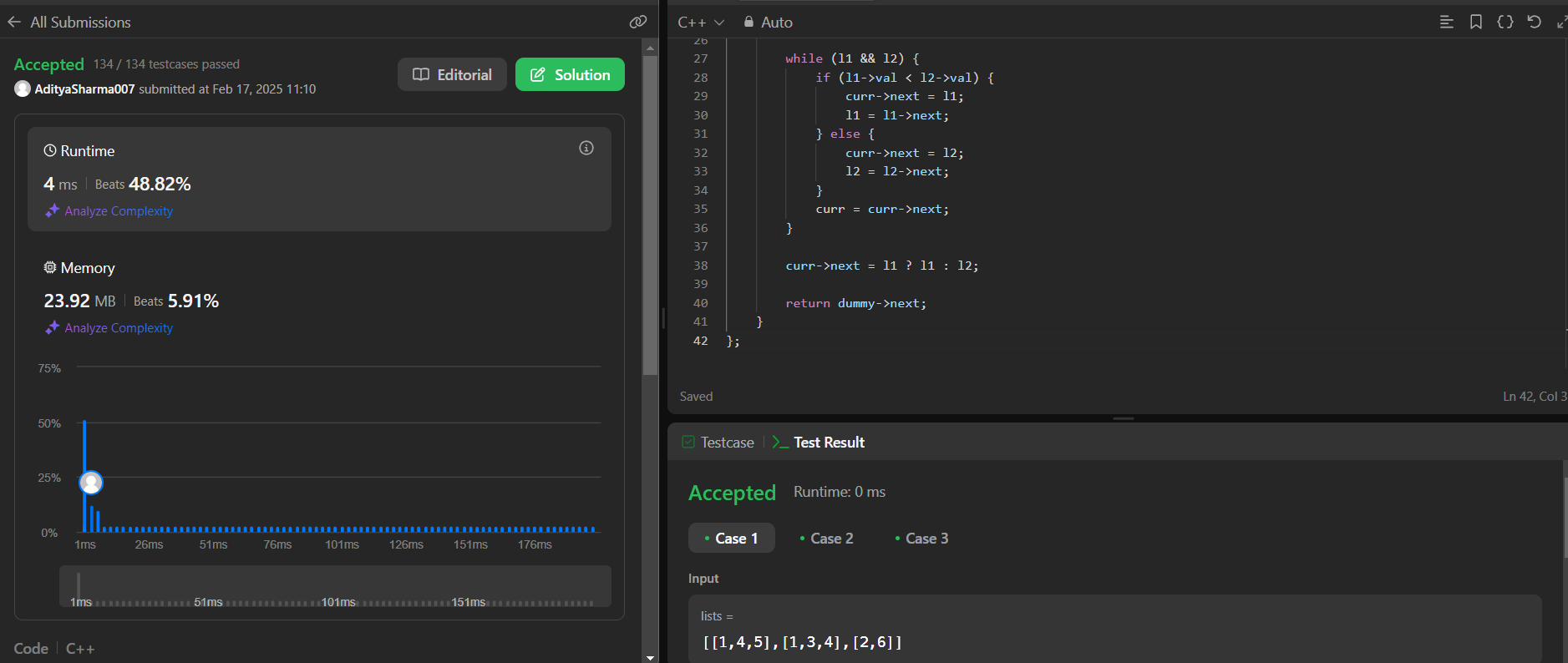
curr->next = l1 ? l1 : l2;

return dummy->next;

}

};

Output-



Problem- 148

Code-

/\*\*

\* Definition for singly-linked list.

\* struct ListNode {

\* int val;

\* ListNode \*next;

\* ListNode() : val(0), next(nullptr) {}

\* ListNode(int x) : val(x), next(nullptr) {}

\* ListNode(int x, ListNode \*next) : val(x), next(next) {}

\* };

\*/

class Solution {

public:

ListNode\* sortList(ListNode\* head) {

vector<int> arr;

ListNode\* temp = head;

// Extract values from the linked list and store them in the vector

while(temp != nullptr){

arr.push\_back(temp->val);

temp = temp->next;

}

// Sort the vector

sort(arr.begin(), arr.end());

// Update the linked list with sorted values

temp = head;

for(int i = 0; temp != nullptr; i++){

temp->val = arr[i];

temp = temp->next;

}

return head;

}

};

Output-

