Assignment 3

Name- Rahul Gupta

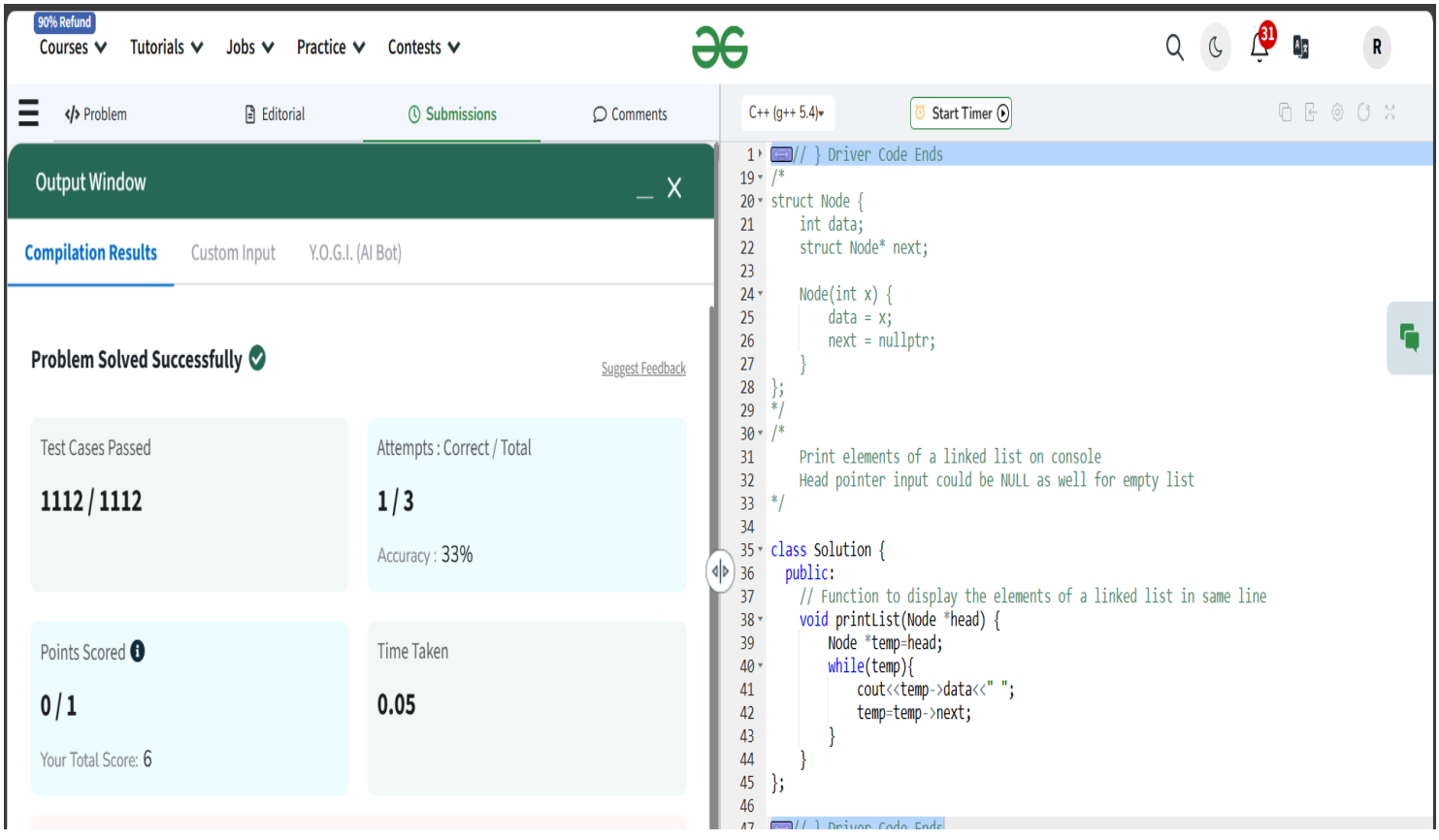
UID- 22BCS10469

Class- 609-B

Question 1) Print Linked List

Code)

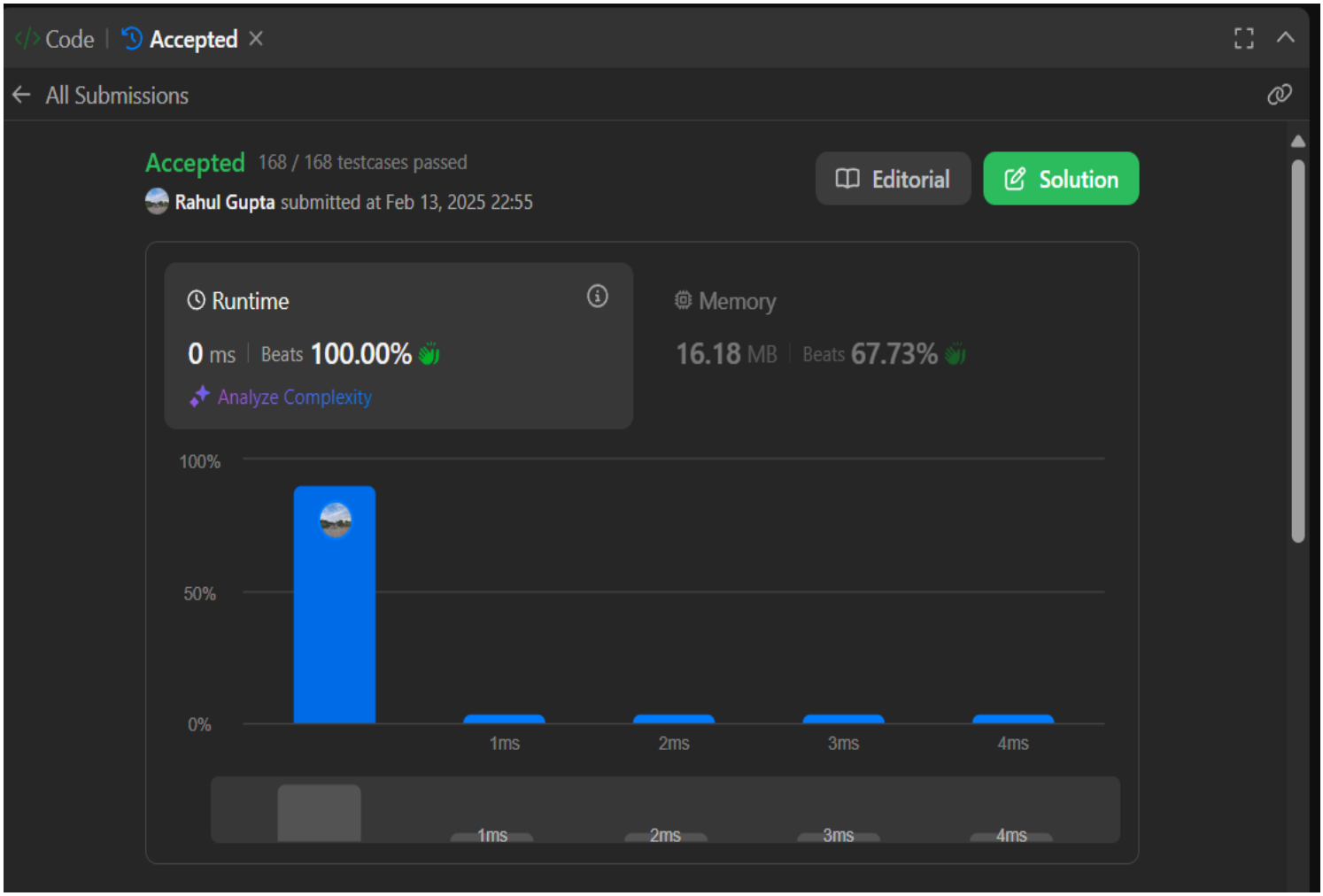
```
class Solution {
  public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        Node *temp=head;
        while(temp){
            cout<<temp->data<<" ";
            temp=temp->next; } } };
```



Question 2) [Remove Duplicates from Sorted List]

Code)

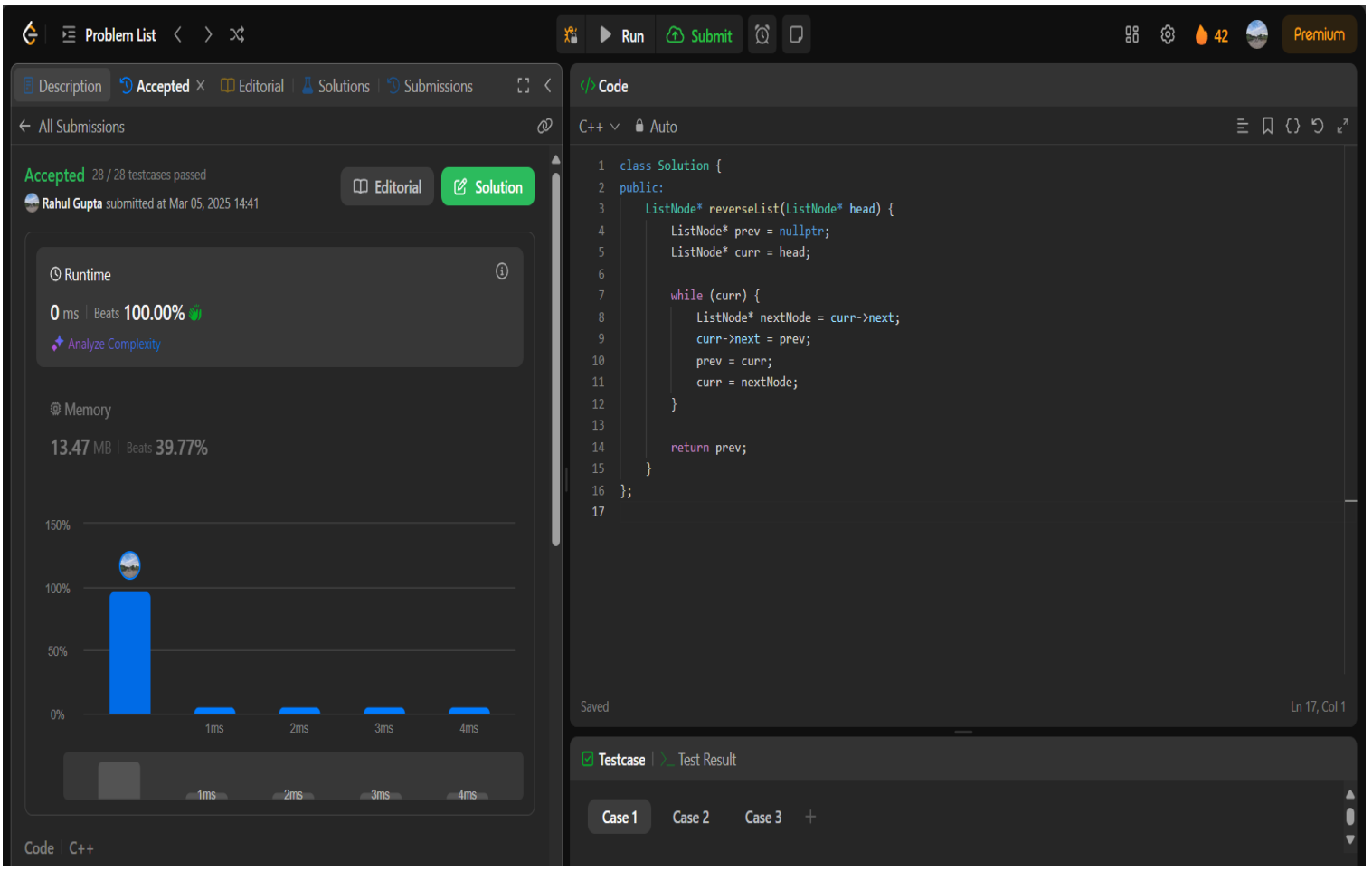
```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;
        while (current && current->next) {
            if (current->val == current->next->val) {
                current->next = current->next->next;  // Skip duplicate node
            } else {
                current = current->next;  // Move to the next node
            }
        }
         return head; } };
```



Question 3) Reverse Linked List

Code)

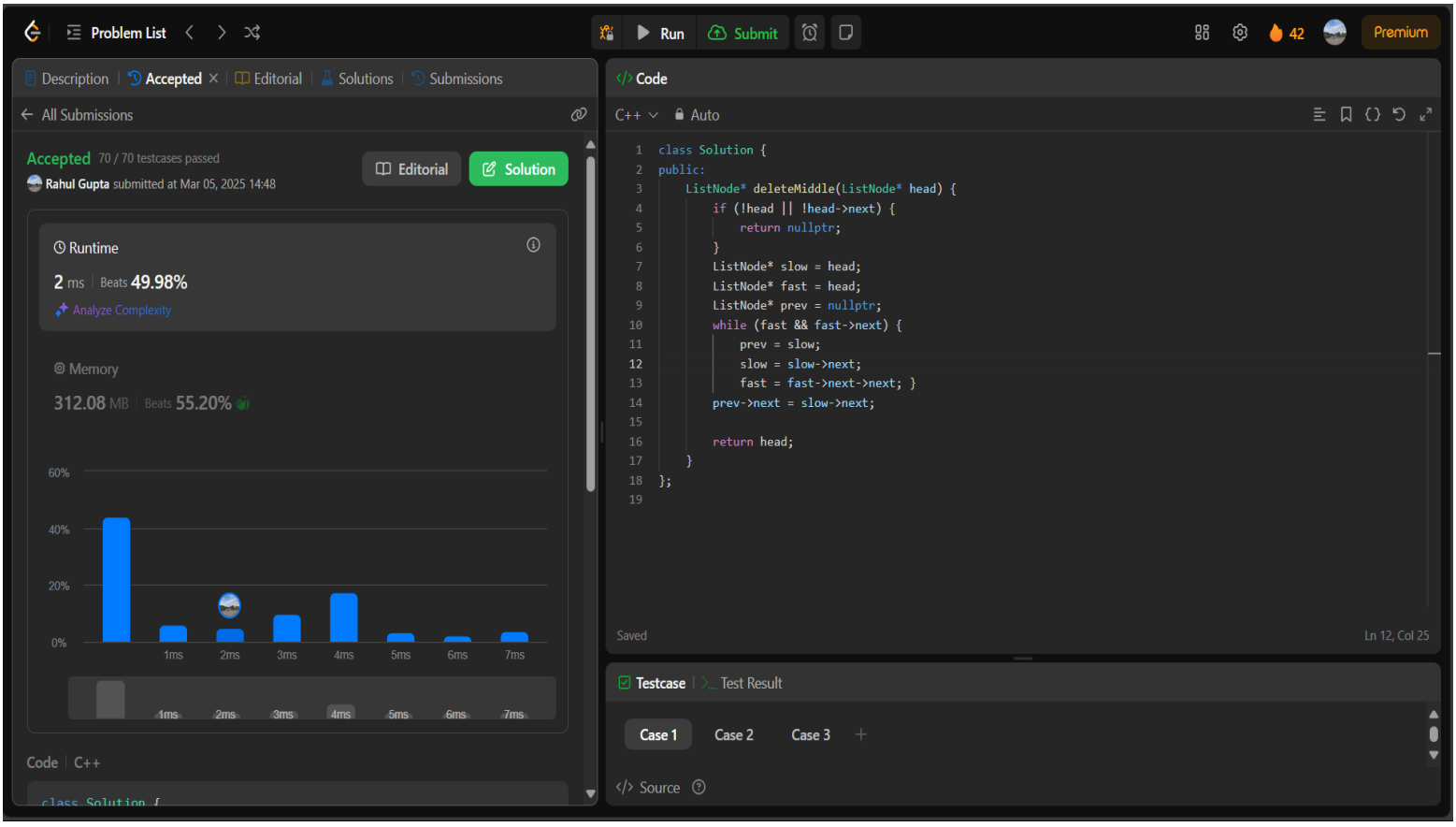
```
class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* curr = head;

        while (curr) {
            ListNode* nextNode = curr->next;
            curr->next = prev;
            prev = curr;
            curr = nextNode;
        }
        return prev;   } };
```



Question 4) Delete the Middle Node of a Linked List

Code)

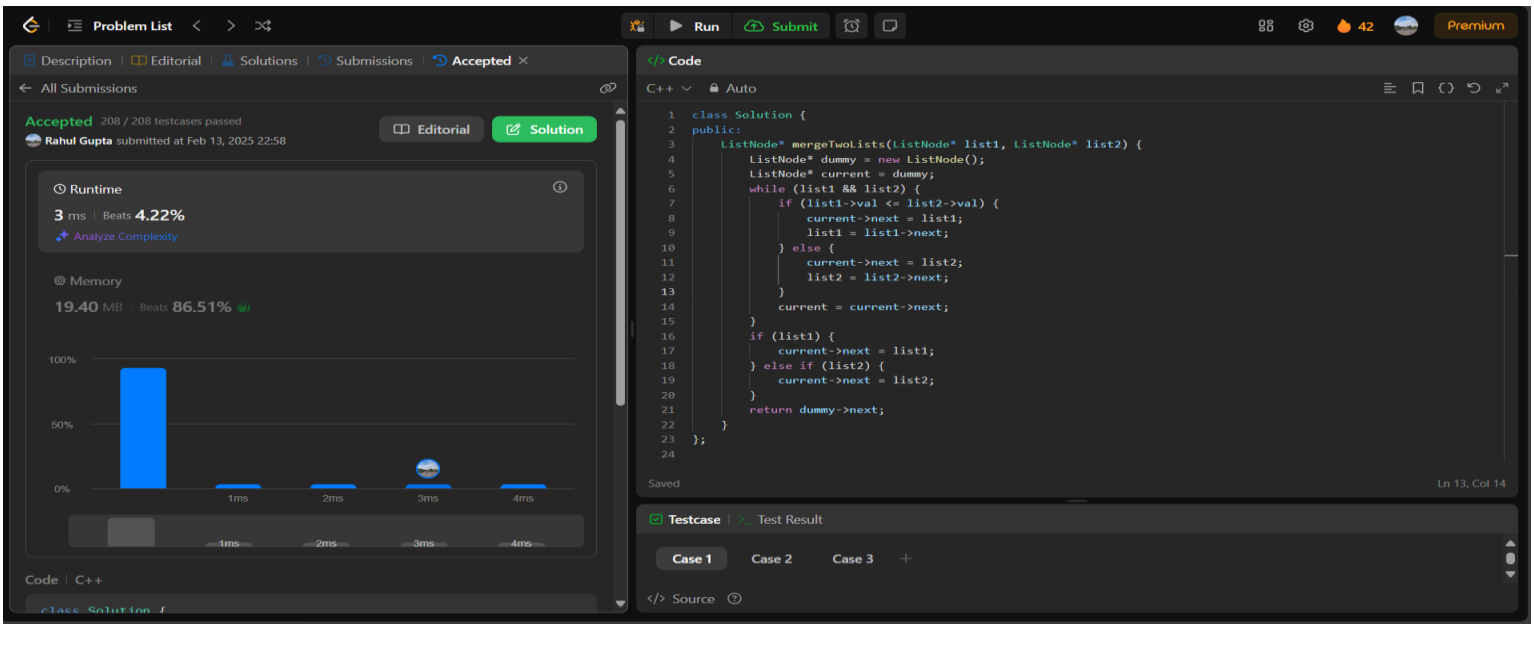
```
class Solution {
public:
    ListNode* deleteMiddle(ListNode* head) {
        if (!head || !head->next) {
            return nullptr;  }
        ListNode* slow = head;
        ListNode* fast = head;
        ListNode* prev = nullptr;
        while (fast && fast->next) {
            prev = slow;
            slow = slow->next;
            fast = fast->next->next; }
        prev->next = slow->next;
        return head; }};
```



Question 5) [Merge Two Sorted Lists]

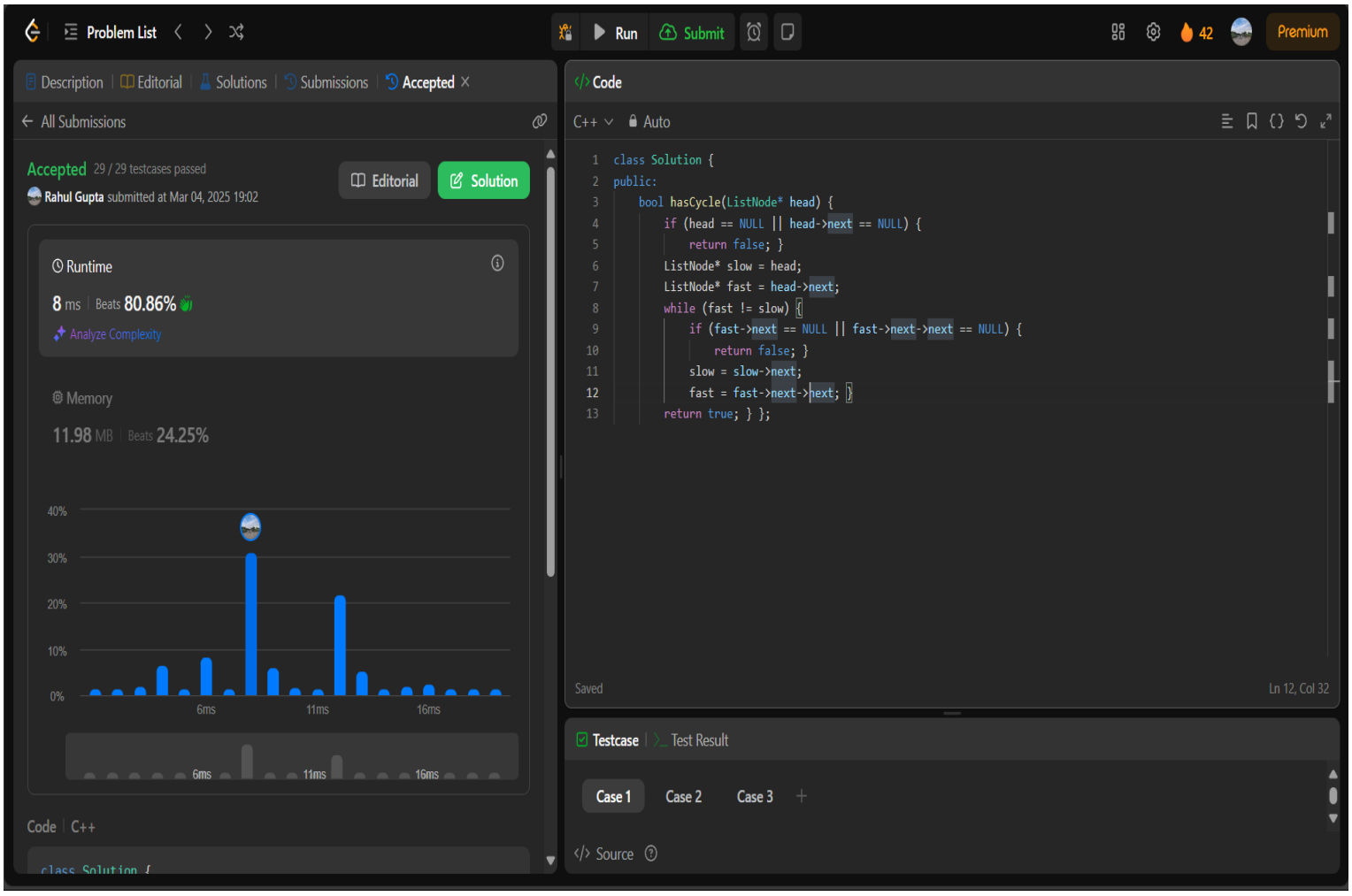
Code)

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode* dummy = new ListNode();
        ListNode* current = dummy;
        while (list1 && list2) {
            if (list1->val <= list2->val) {
                current->next = list1;
                list1 = list1->next;
            } else {
                current->next = list2;
                list2 = list2->next;  }
            current = current->next;  }
        if (list1) {
            current->next = list1;
        } else if (list2) {
            current->next = list2; }
        return dummy->next; } };
```



Question 6) Linked List Cycle

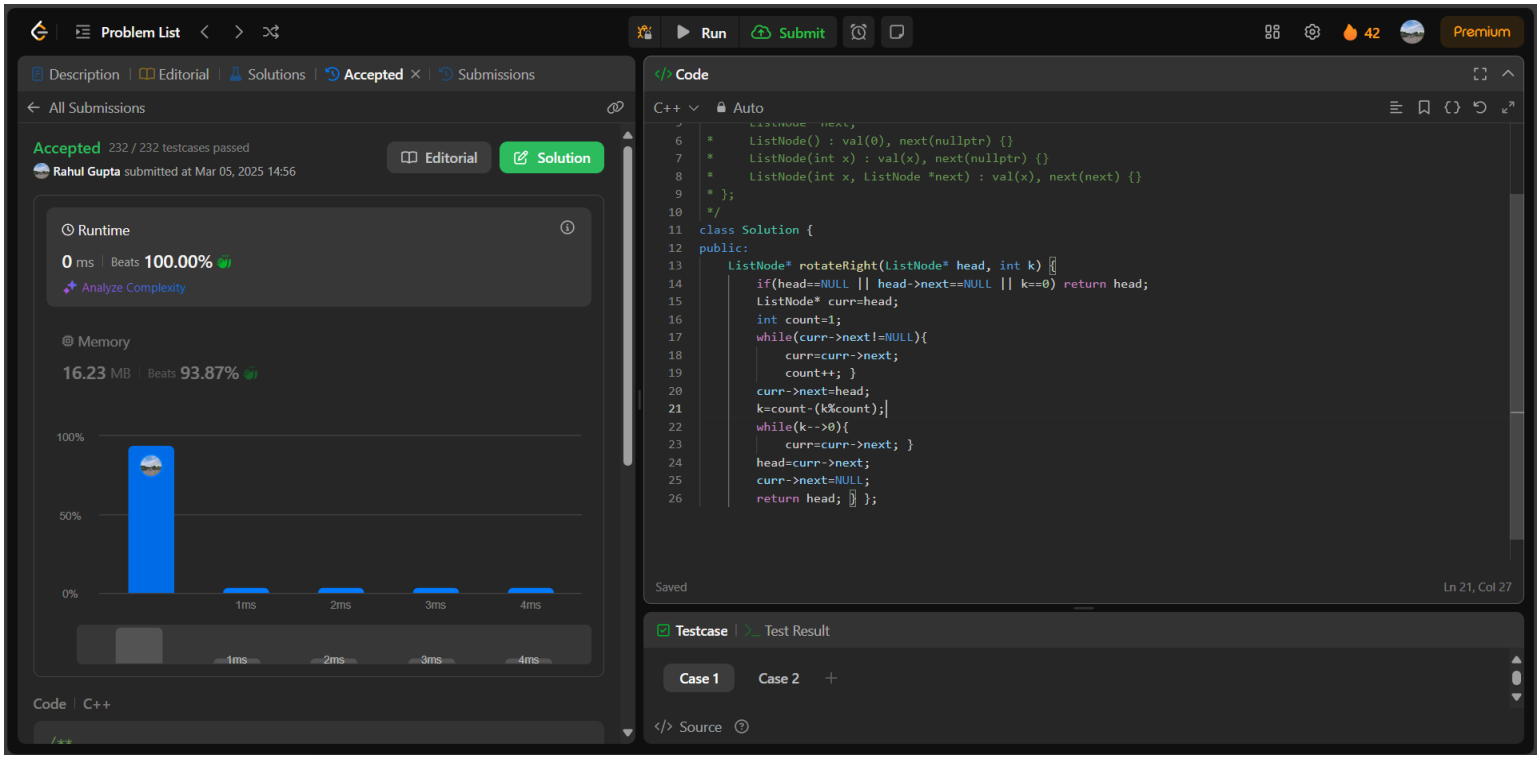
```
class Solution {
public:
    bool hasCycle(ListNode* head) {
        if (head == NULL || head->next == NULL) {
            return false; }
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast != slow) {
            if (fast->next == NULL || fast->next->next == NULL) {
                return false; }
            slow = slow->next;
            fast = fast->next->next; }
        return true; } };
```



Question 7) Rotate List

Code)

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if(head==NULL || head->next==NULL || k==0) return head;
        ListNode* curr=head;
        int count=1;
        while(curr->next!=NULL){
            curr=curr->next;
            count++; }
        curr->next=head;
        k=count-(k%count);
        while(k-->0){
            curr=curr->next; }
        head=curr->next;
        curr->next=NULL;
        return head; } };
```

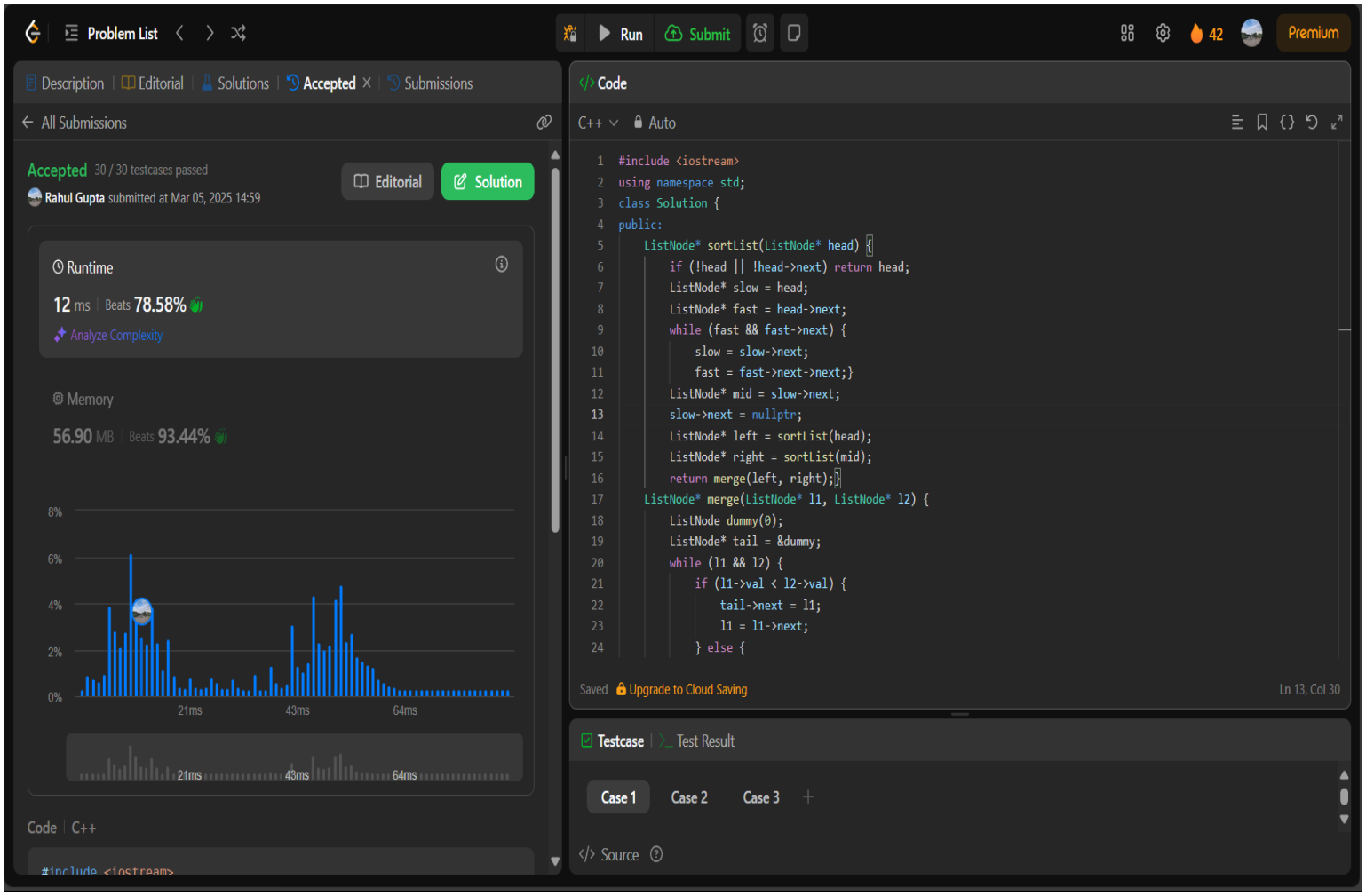


Question 8) Sort List

Code)

```
#include <iostream>
using namespace std;
class Solution {
public:
    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head;
        ListNode* slow = head;
        ListNode* fast = head->next;
        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;}
        ListNode* mid = slow->next;
        slow->next = nullptr;
        ListNode* left = sortList(head);
        ListNode* right = sortList(mid);
        return merge(left, right);}
    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode dummy(0);
        ListNode* tail = &dummy;
        while (l1 && l2) {
            if (l1->val < l2->val) {
                tail->next = l1;
                l1 = l1->next;
            } else {
```

```
        tail->next = l2;
        l2 = l2->next;}
    tail = tail->next;}
tail->next = l1 ? l1 : l2;
 return dummy.next; } };
```



Question 9) Merge k Sorted Lists

Code)

```
#include <vector>
using namespace std;
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* l1, ListNode* l2) {
        if (!l1) return l2;
        if (!l2) return l1;

        if (l1->val < l2->val) {
            l1->next = mergeTwoLists(l1->next, l2);
            return l1;
        } else {
            l2->next = mergeTwoLists(l1, l2->next);
            return l2;
        }
    }

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) return nullptr;
        return divideAndConquer(lists, 0, lists.size() - 1);
    }

    ListNode* divideAndConquer(vector<ListNode*>& lists, int left, int right) {
        if (left == right) return lists[left];
```

```
        int mid = left + (right - left) / 2;
        ListNode* l1 = divideAndConquer(lists, left, mid);
        ListNode* l2 = divideAndConquer(lists, mid + 1, right);
        return mergeTwoLists(l1, l2);
    }
};
```