**ASSINGMENT 3 AP LAB**

**NAME- NIKHIL UID-22BCS11558**

**Q1 PRINT LINKED LIST:-**

CODE:-

class Solution {

public:

void printList(Node \*head) {

while (head != nullptr) {

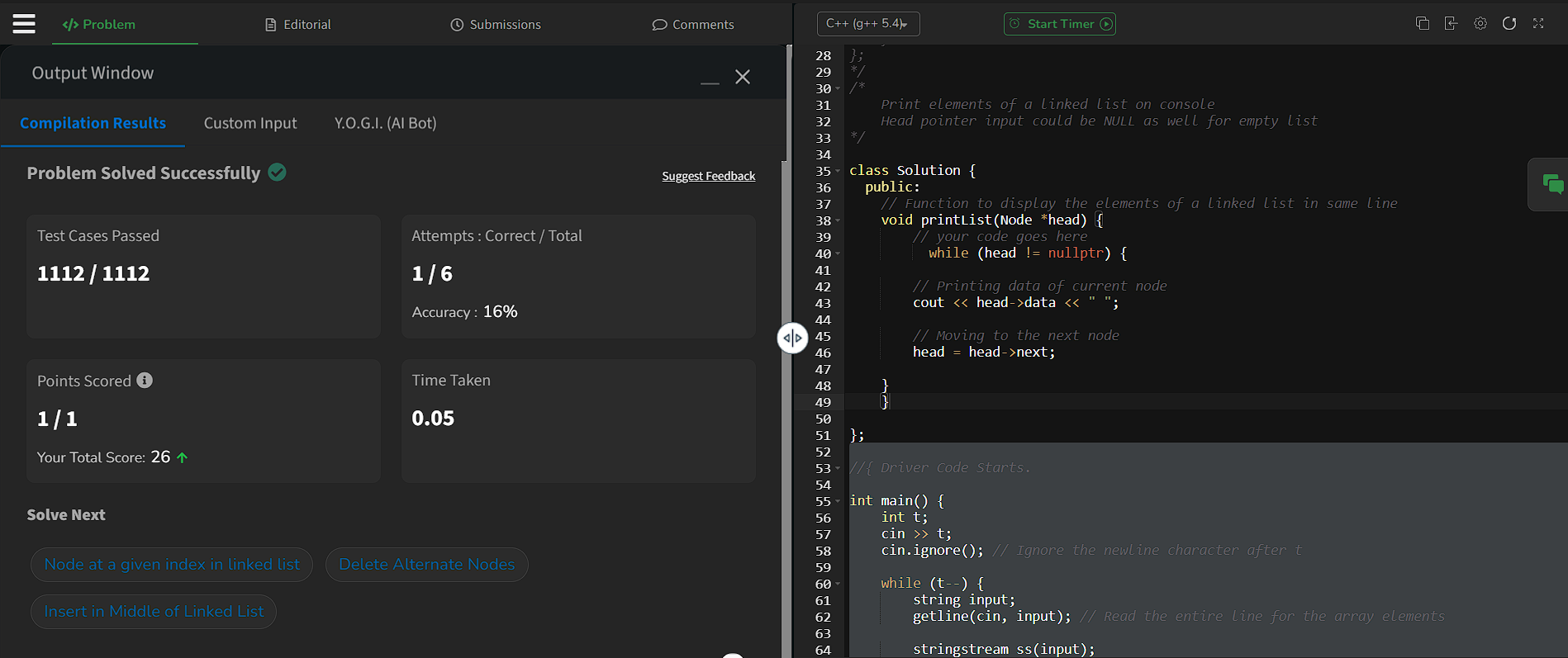
cout << head->data << " ";

head = head->next;

}

}

SCRENSHOT:-



**Q2 .[Remove duplicates from a sorted list](https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/):-**

CODE:-

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

            if (!head) return head; // Handle empty list

        ListNode\* current = head;

        while (current && current->next) {

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

                ListNode\* duplicate = current->next;

                current->next = duplicate->next;  // Skip duplicate node

                delete duplicate;  // Free memory

            } else {

                current = current->next;  // Move to the next distinct node

            }

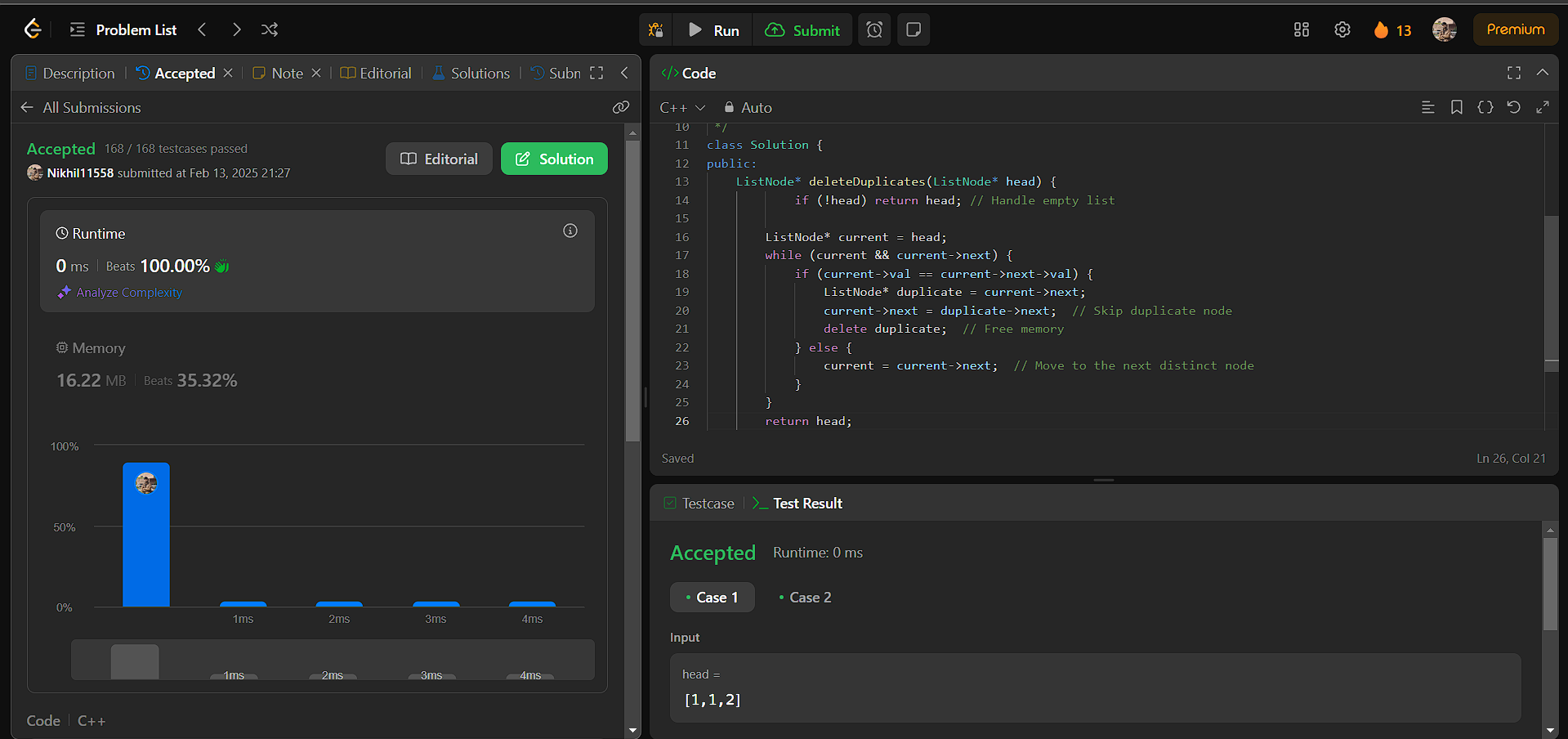
        }

        return head;

    }

};

SCREENSHOT:-



**Q3.  Reverse a linked list:-**

CODE:-

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = nullptr;  // Previous node

        ListNode\* current = head;  // Current node

        while (current) {

            ListNode\* next = current->next;  // Save next node

            current->next = prev;  // Reverse the link

            prev = current;  // Move prev forward

            current = next;  // Move current forward

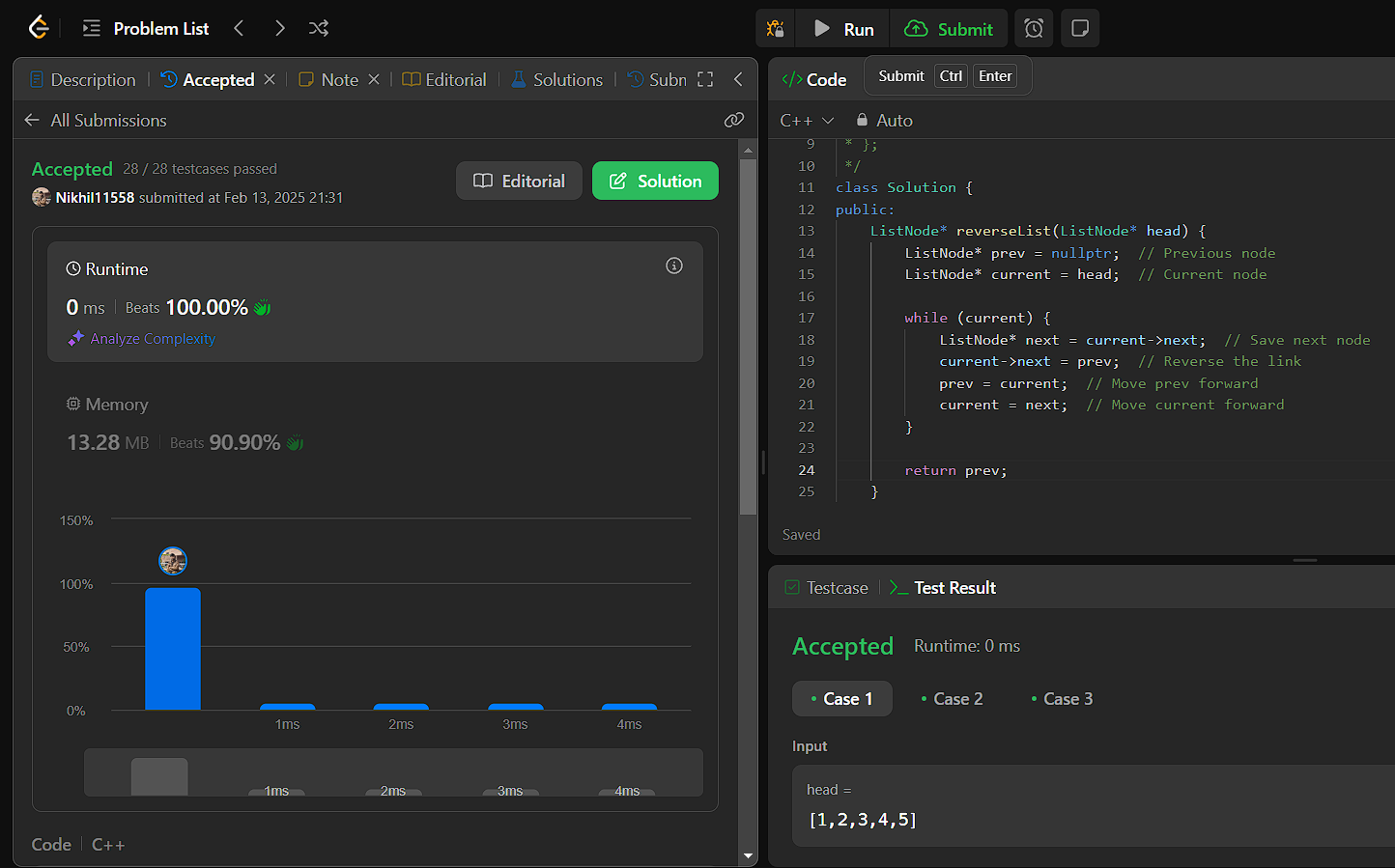
        }

        return prev;

    }

};

SCREENSHOT:-



**Q4 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;

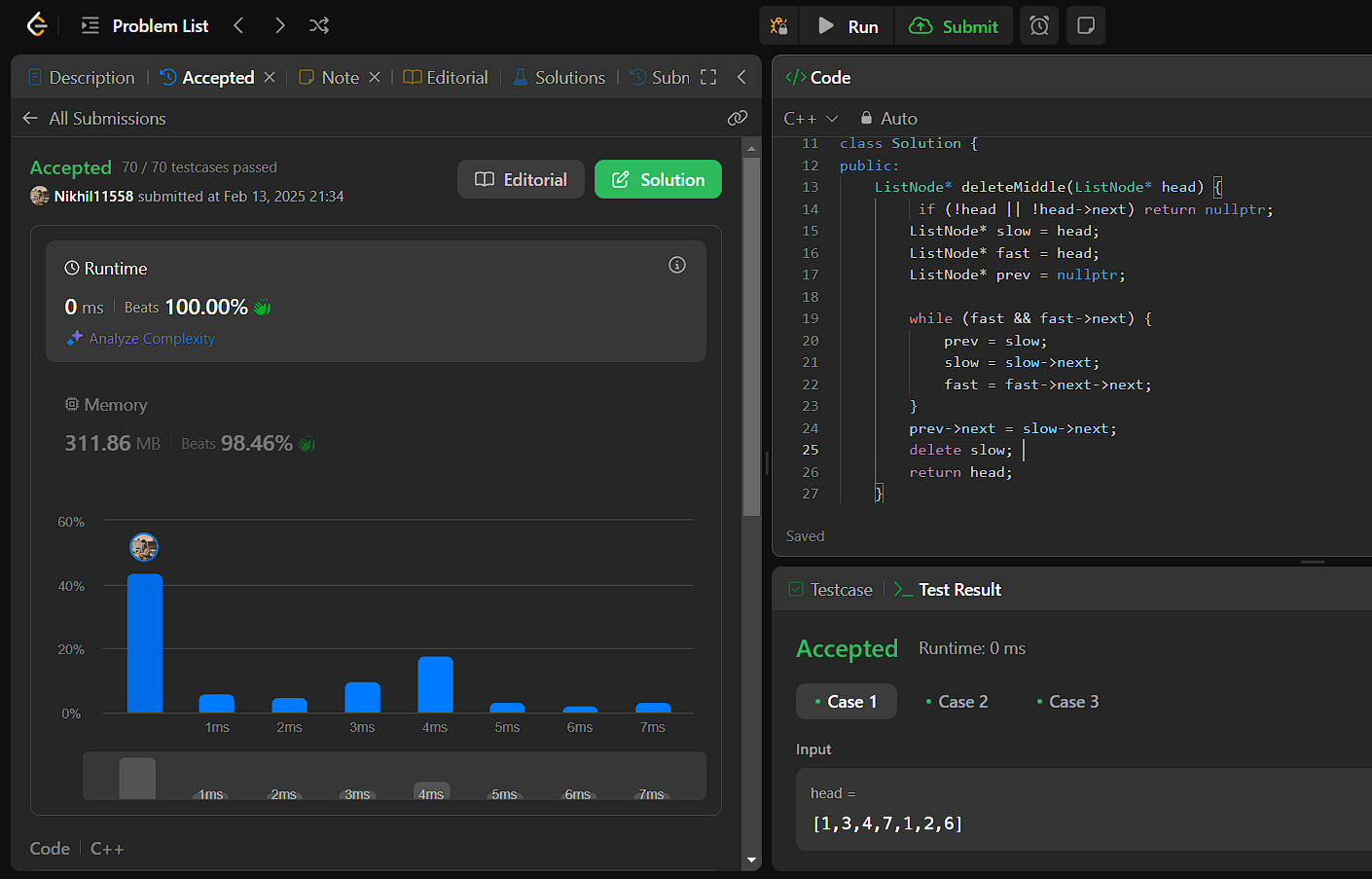
        delete slow;

        return head;

    }

};

SCREENSHOT:-



**Q5 [Merge two sorted linked lists](https://leetcode.com/problems/merge-two-sorted-lists/description/):-**

CODE:-

class Solution {

public:

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

        ListNode dummy(0);

        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;

        }

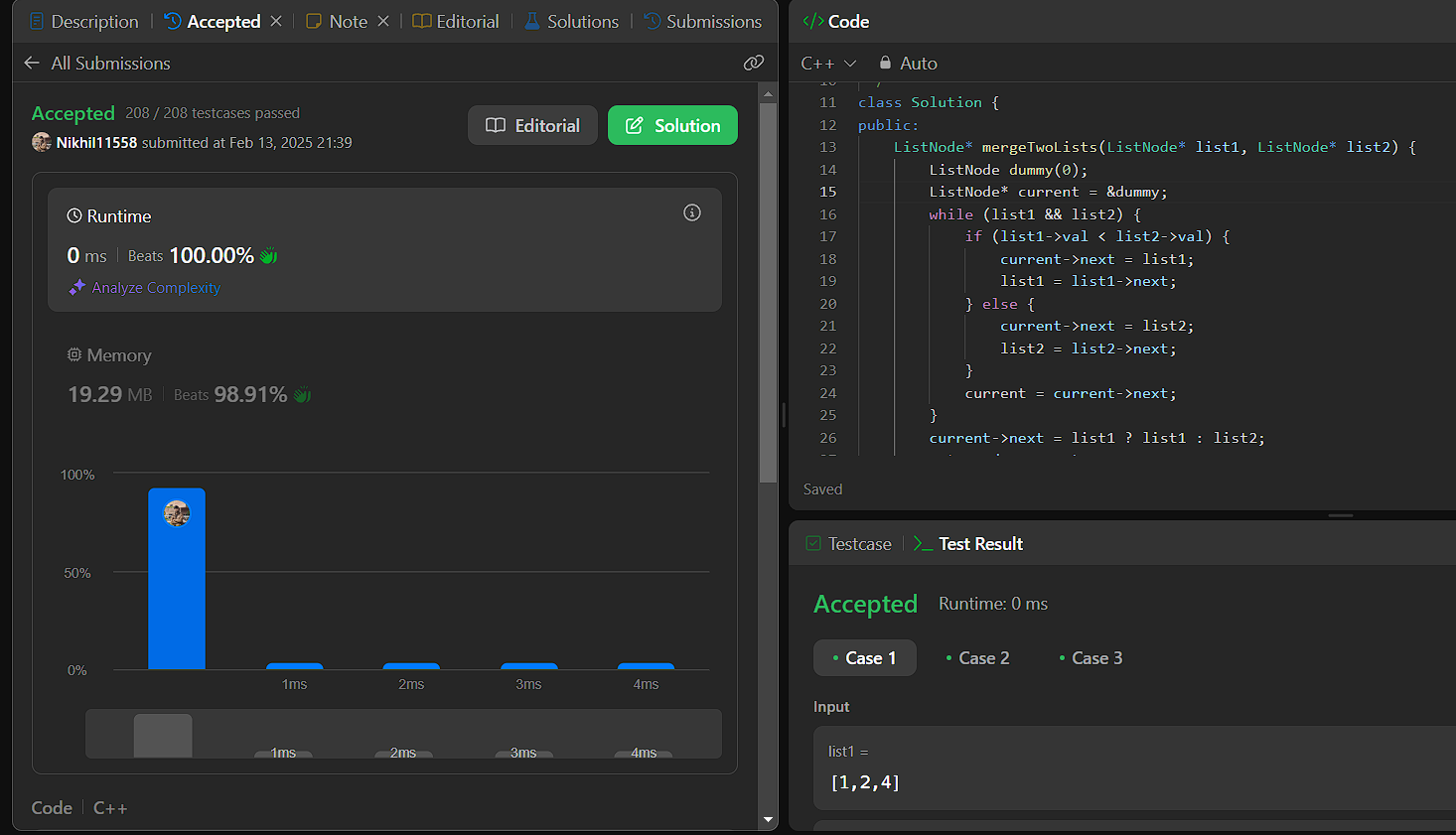
        current->next = list1 ? list1 : list2;

        return dummy.next;

    }

};

SCREENSHOT:-



**Q6 [Remove duplicates from sorted lists 2](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/):-**

CODE:-

ListNode\* deleteDuplicates(ListNode\* head) {

         ListNode dummy(0, head);  // Dummy node before head

        ListNode\* prev = &dummy;  // Pointer to track the node before duplicates

        while (head) {

            bool isDuplicate = false;

            // Skip all duplicate nodes

            while (head->next && head->val == head->next->val) {

                head = head->next;  // Move forward

                isDuplicate = true;

            }

            if (isDuplicate) {

                prev->next = head->next;  // Remove all duplicates

            } else {

                prev = prev->next;  // Move prev forward

            }

            head = head->next;  // Move head forward

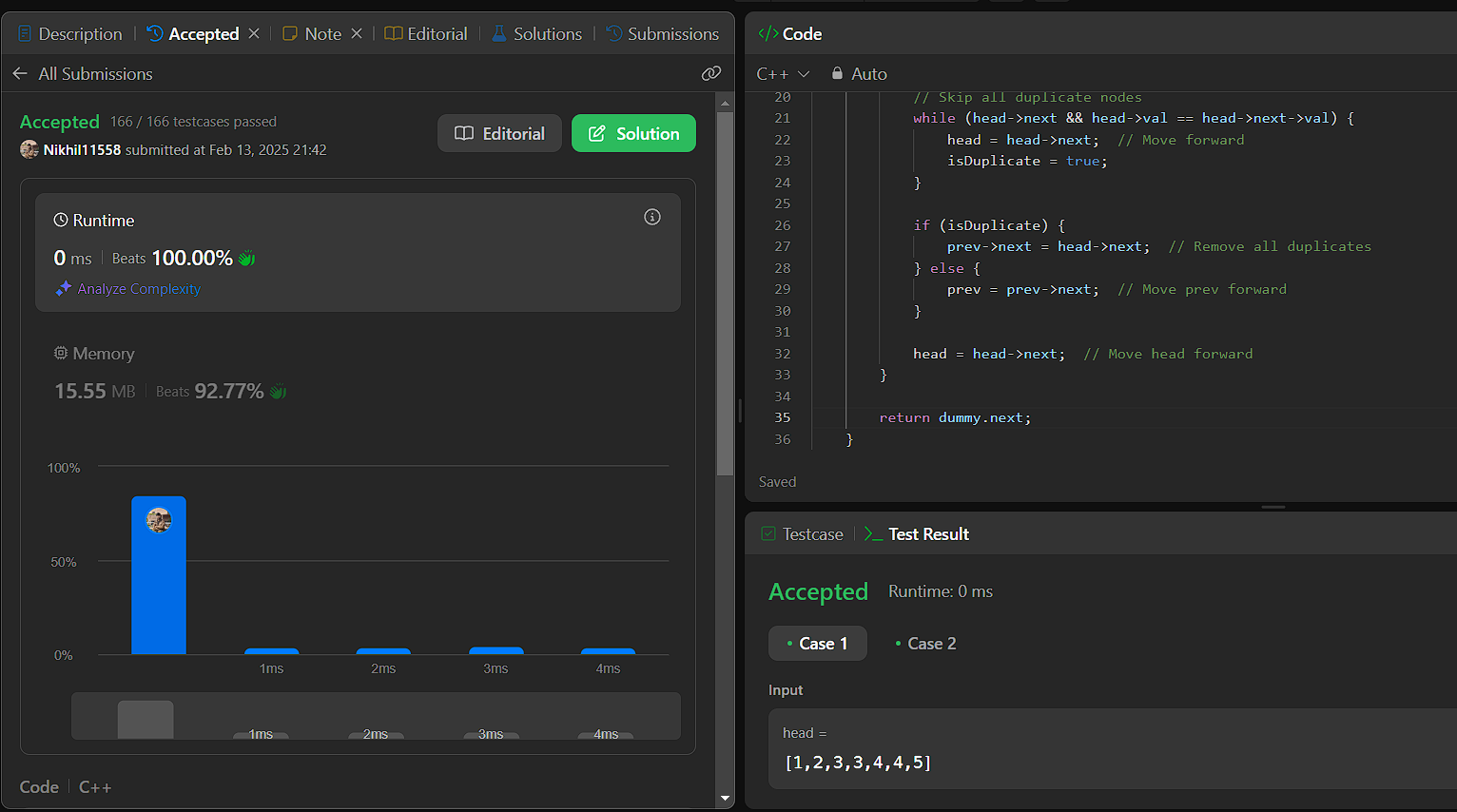
        }

        return dummy.next;

    }

};

SCREENSHOT:-



**Q7 [Detect a cycle in a linked list](https://leetcode.com/problems/linked-list-cycle/description/):-**

CODE:-

class Solution {

public:

    bool hasCycle(ListNode \*head) {

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

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

            if (slow == fast) return true;

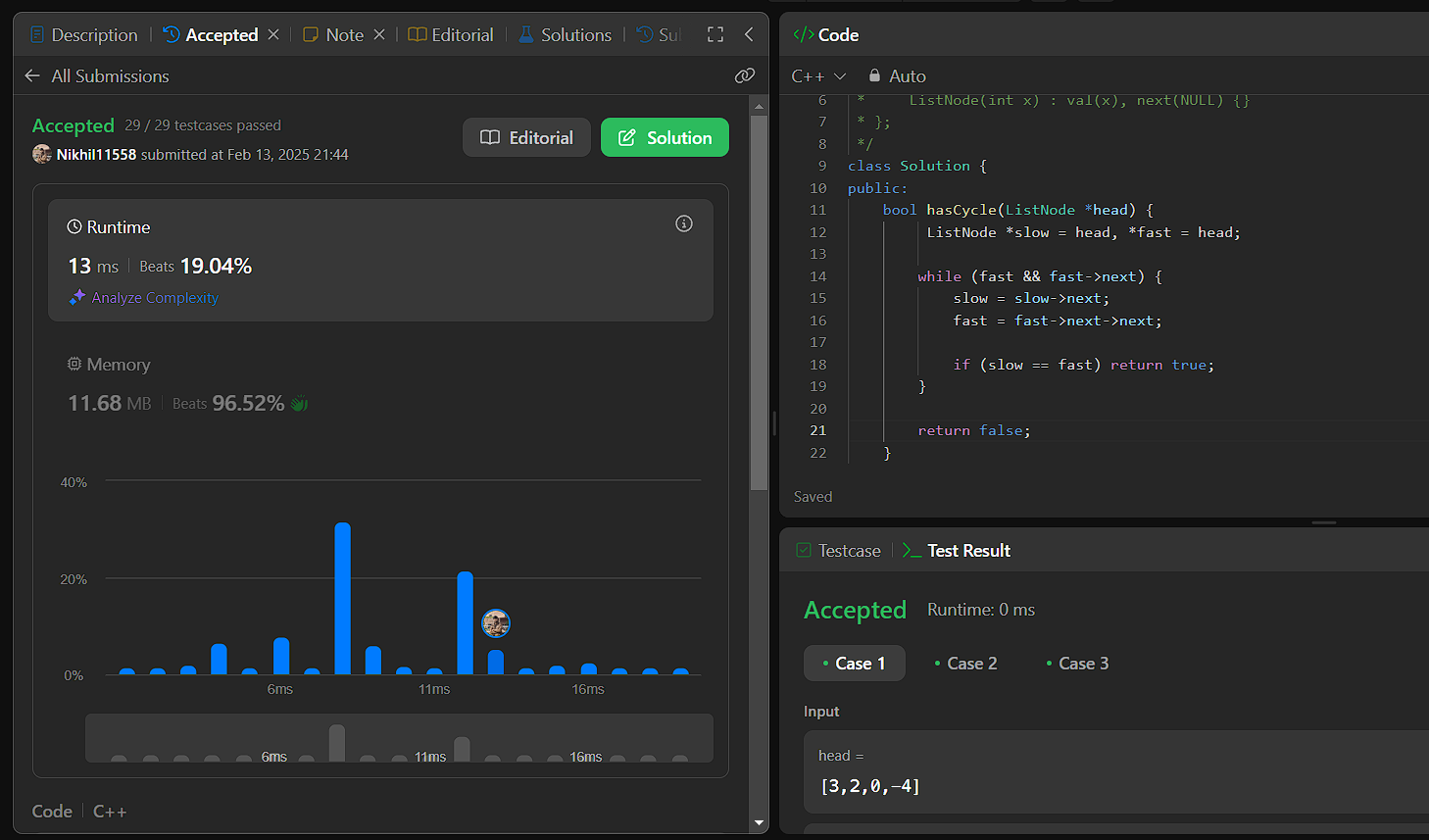
        }

        return false;

    }

};

SCREENSHOT:-



**Q8 [Reverse linked list 2](https://leetcode.com/problems/reverse-linked-list-ii/description/):-**

CODE:-

class Solution {

public:

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

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

        ListNode dummy(0, head);

        ListNode\* prevLeft = &dummy;

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

            prevLeft = prevLeft->next;

        }

        ListNode\* curr = prevLeft->next;

        ListNode\* prev = nullptr;

        ListNode\* next = nullptr;

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

            next = curr->next;

            curr->next = prev;

            prev = curr;

            curr = next;

        }

        prevLeft->next->next = curr;

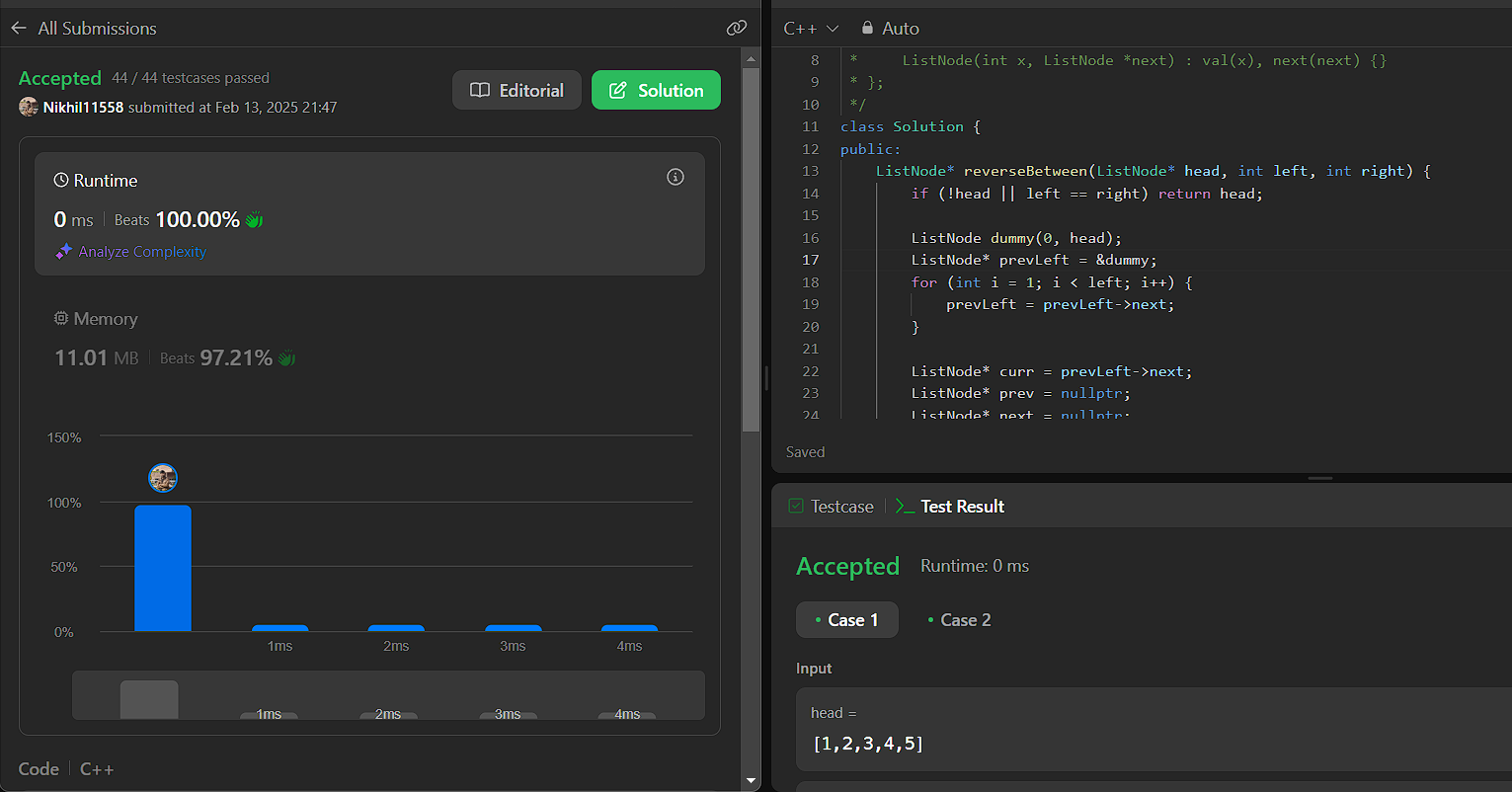
        prevLeft->next = prev;

        return dummy.next;

    }

};

SCREENSHOT:-



**Q9 [rotate a list](https://leetcode.com/problems/rotate-list/description/):-**

CODE:-

class Solution {

public:

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

        if (!head || !head->next || k == 0) return head;

        ListNode\* temp = head;

        int length = 1;

        while (temp->next) {

            temp = temp->next;

            length++;

        }

        temp->next = head;

        k = k % length;

        int newTailPos = length - k;

        ListNode\* newTail = head;

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

            newTail = newTail->next;

        }

        ListNode\* newHead = newTail->next;

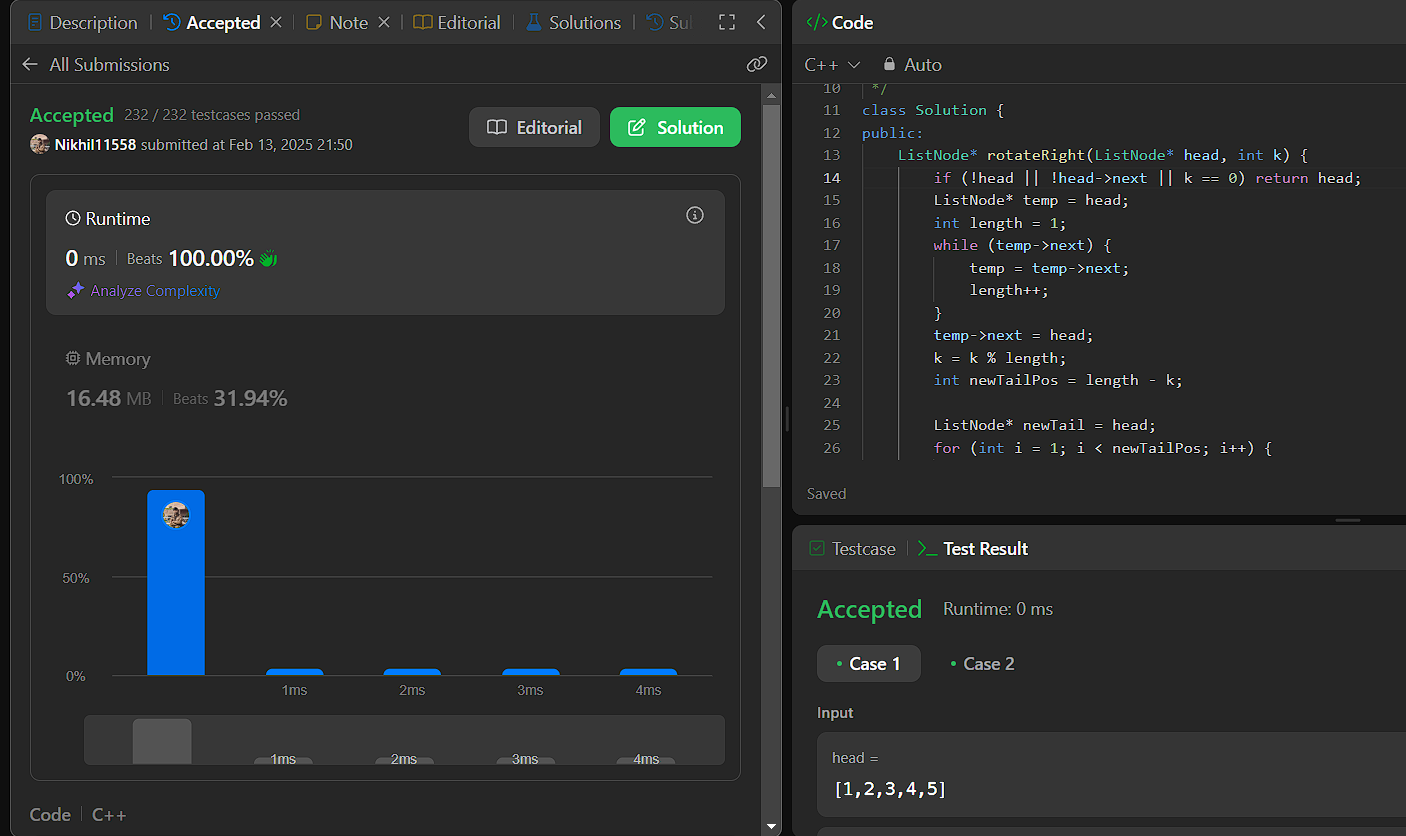
        newTail->next = nullptr;

        return newHead;

    }

};

SCREENSHOT:-



**Q10 [Sort List](https://leetcode.com/problems/sort-list/description/):-**

CODE:-

class Solution {

public:

  ListNode\* merge(ListNode\* left, ListNode\* right) {

        ListNode dummy;

        ListNode\* tail = &dummy;

        while (left && right) {

            if (left->val < right->val) {

                tail->next = left;

                left = left->next;

            } else {

                tail->next = right;

                right = right->next;

            }

            tail = tail->next;

        }

        tail->next = left ? left : right;

        return dummy.next;

    }

    ListNode\* sortList(ListNode\* head) {

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

        ListNode\* slow = head, \*fast = head->next;

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

        }

        ListNode\* mid = slow->next;

        slow->next = nullptr;

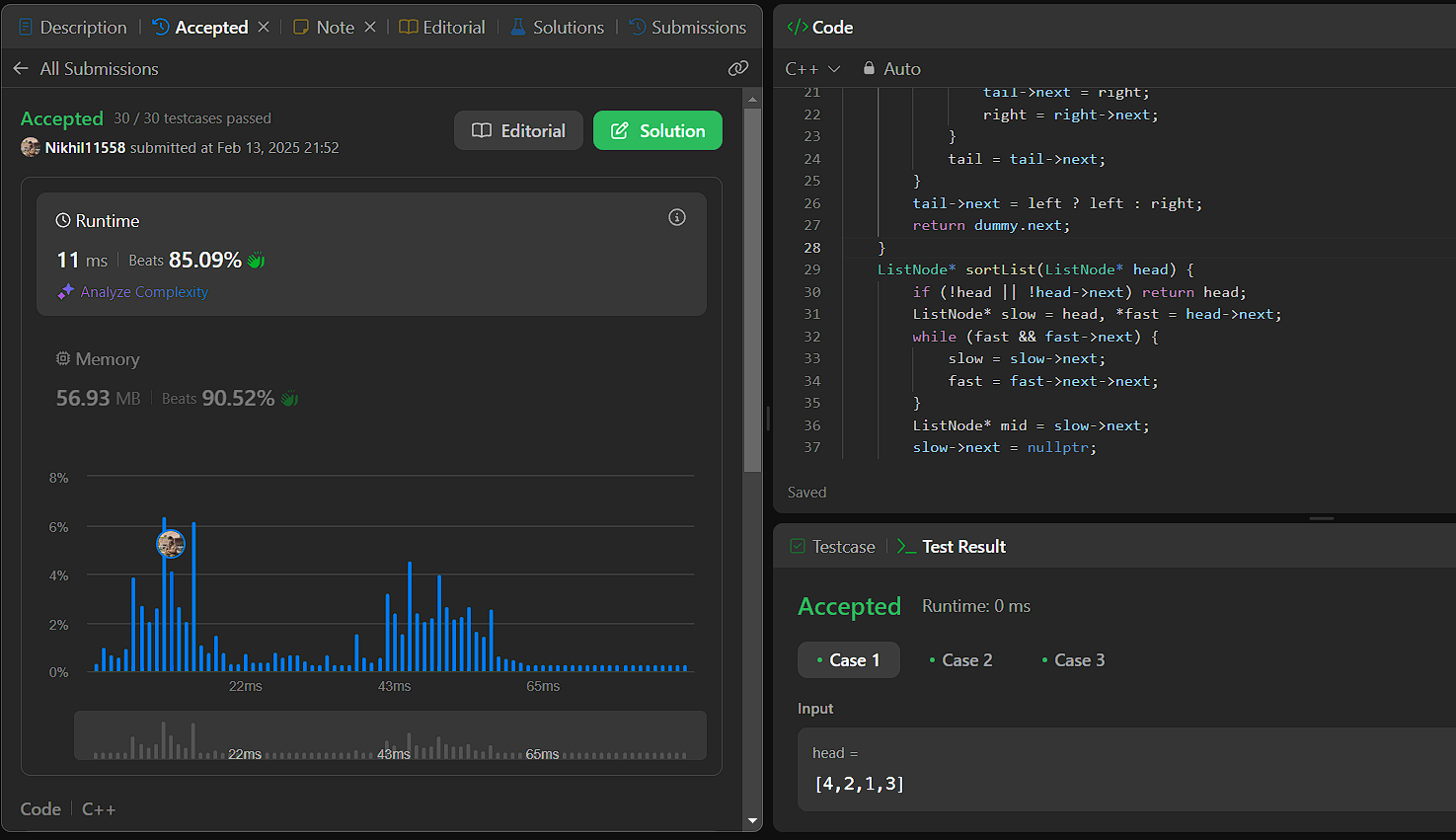
        ListNode\* left = sortList(head);

        ListNode\* right = sortList(mid);

        return merge(left, right);

    }

};

SCREENSHOT:-

**Q11 [Detect a cycle in a linked list 2](https://leetcode.com/problems/linked-list-cycle-ii/description/):-**

CODE:-

class Solution {

public:

    ListNode \*detectCycle(ListNode \*head) {

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

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

            if (slow == fast) {

                slow = head;

                while (slow != fast) {

                    slow = slow->next;

                    fast = fast->next;

                }

                return slow;

            }

        }

        return nullptr;

    }

};

SCREENSHOT:-

