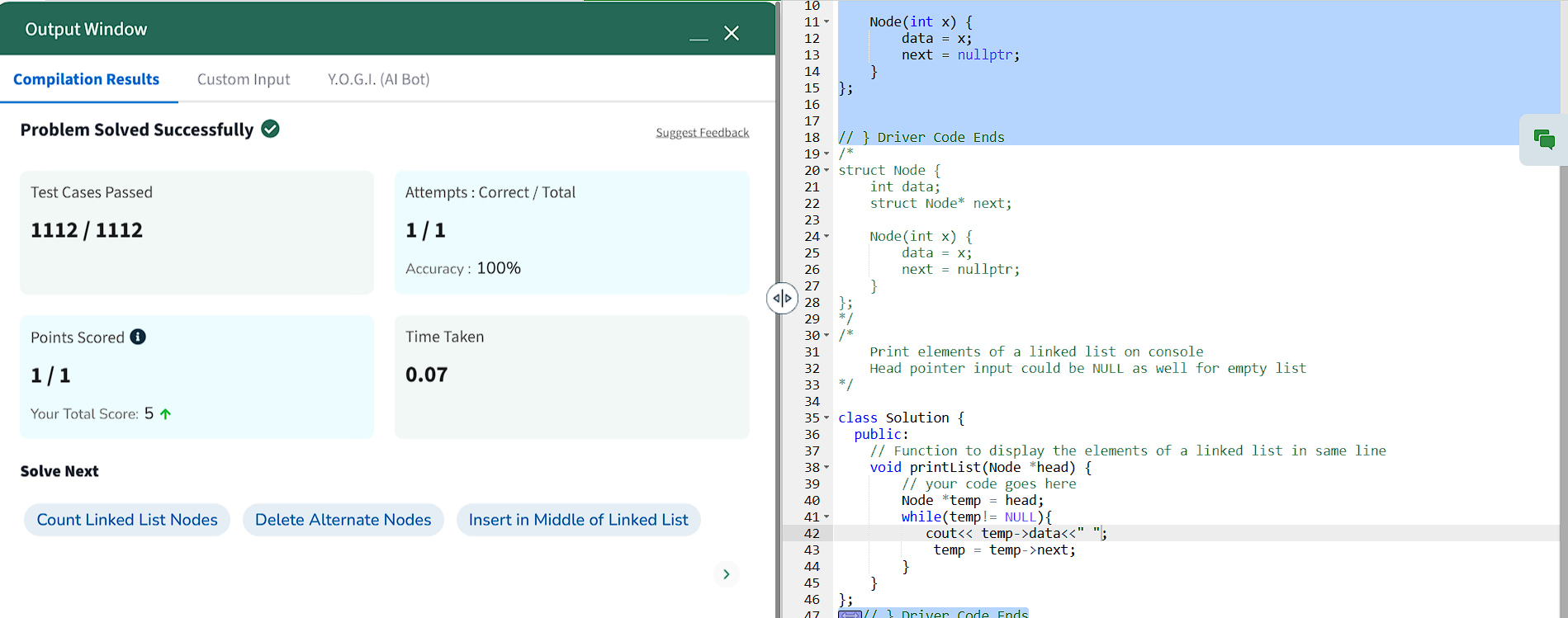
**Aim:** Given a linked list. Print all the elements of the linked list separated by space followed.

**Code:**

class Solution {  
  public:  
    // Function to display the elements of a linked list in same line  
    void printList(Node \*head) {  
        // your code goes here  
        Node\* temp=head; // temp ptr pointing to head  
        while(temp!=NULL){  //traverse till the last element  
            cout<<temp->data<<" "; //print each element  
            temp=temp->next; // increment temp ptr  
        }  
    }  
};

**Output:**



**Aim:** [Remove Duplicates from Sorted List](https://leetcode.com/problems/remove-duplicates-from-sorted-list/)

**Code:**

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* current = head;

while (current && current->next) {

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

current->next = current->next->next;

} else {

current = current->next;

}

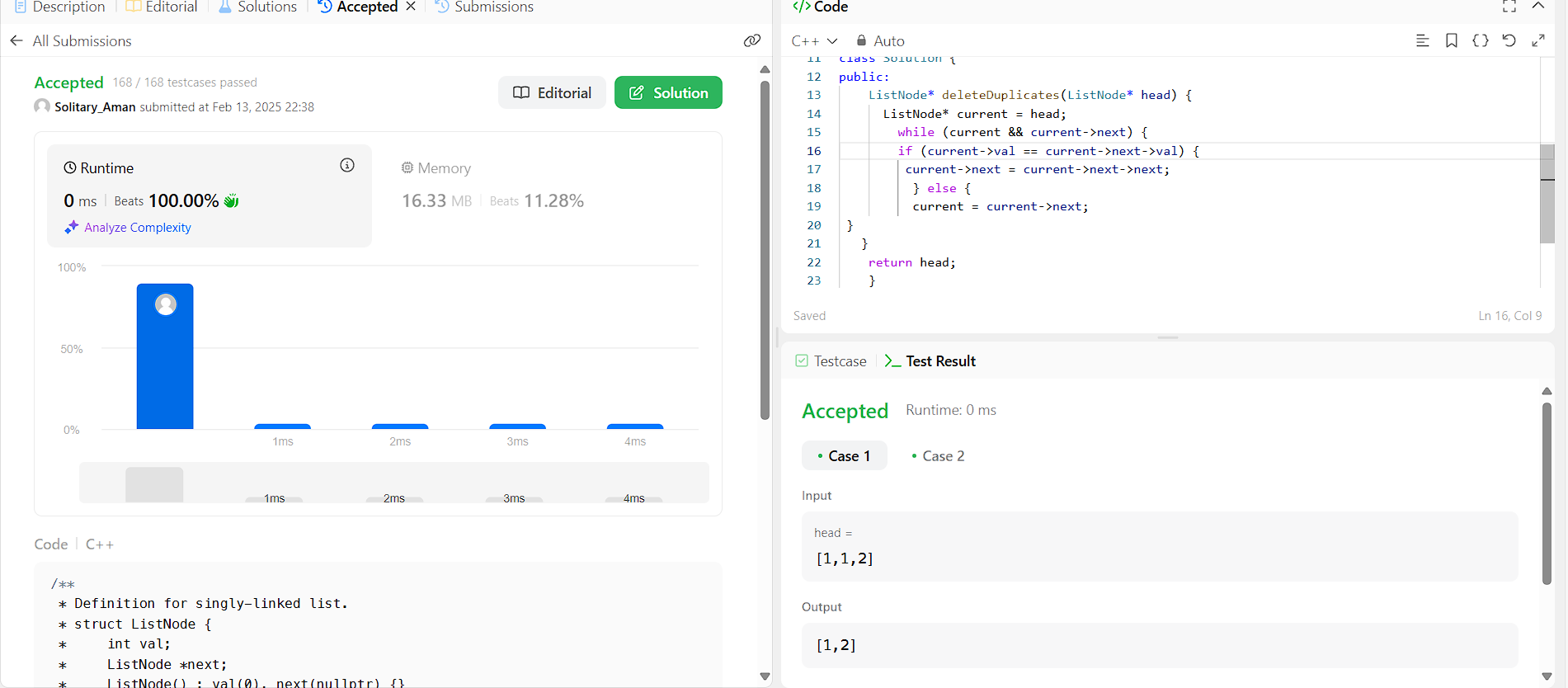
}

return head;

}

};

**Output:**



**Aim:** [Reverse Linked List](https://leetcode.com/problems/reverse-linked-list/)

**Code:**

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode\* prev = nullptr;

ListNode\* current = head;

while (current) {

ListNode\* nextNode = current->next;

current->next = prev;

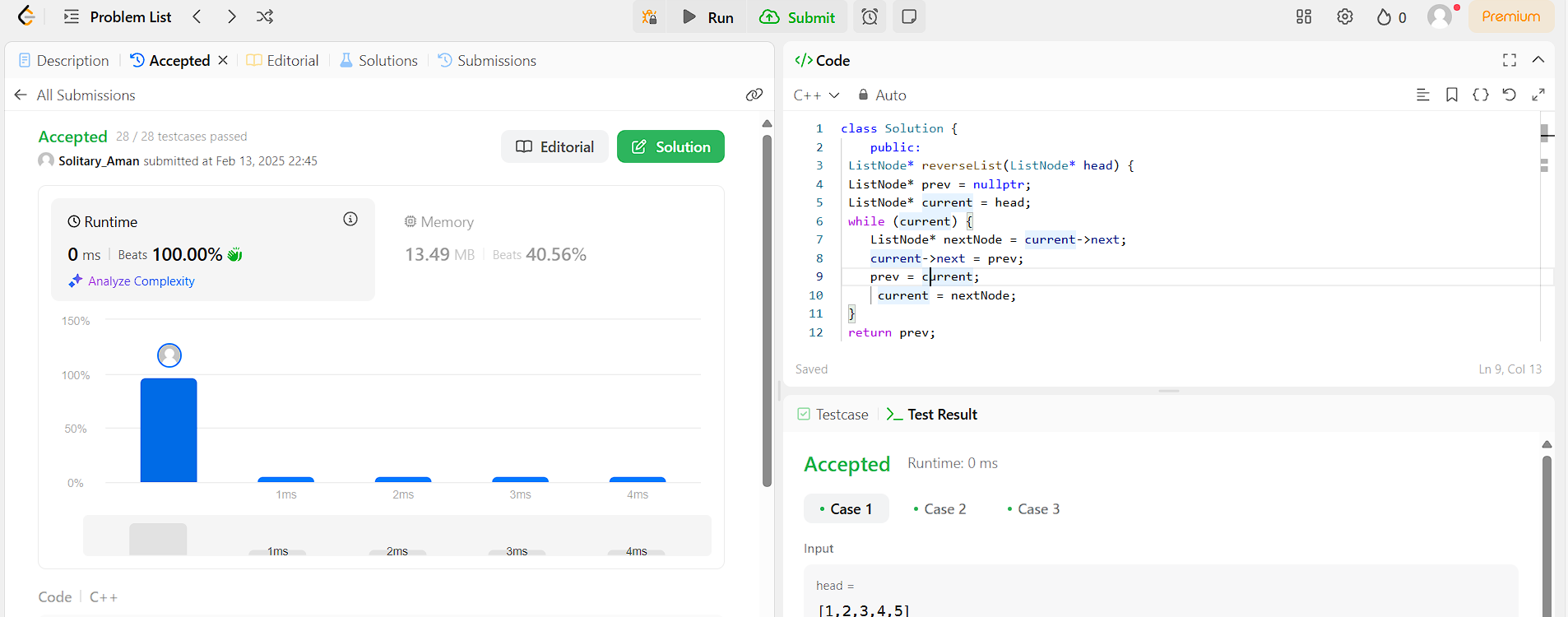
prev = current;

current = nextNode;

}

return prev;

**Output:**

****

**Aim:** [Delete the Middle Node of a Linked List](https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/)

**Code:**

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

return NULL;

}

ListNode\*slow=head;

ListNode\*fast=head;

ListNode\*prev=NULL;

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

prev=slow;

slow=slow->next;

fast=fast->next->next;

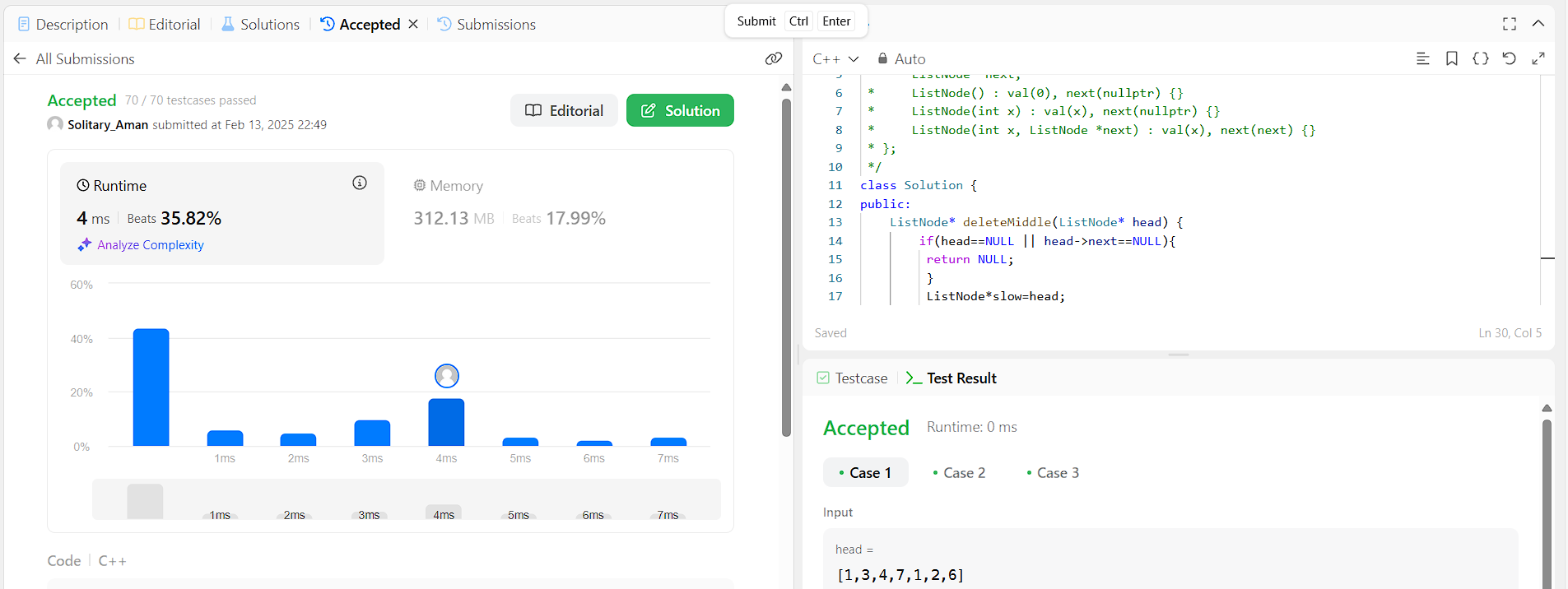
}

prev->next=slow->next;

delete slow;

return head;

**Output:**



**Aim:** [Merge Two Sorted Lists](https://leetcode.com/problems/merge-two-sorted-lists/)

**Code:**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;

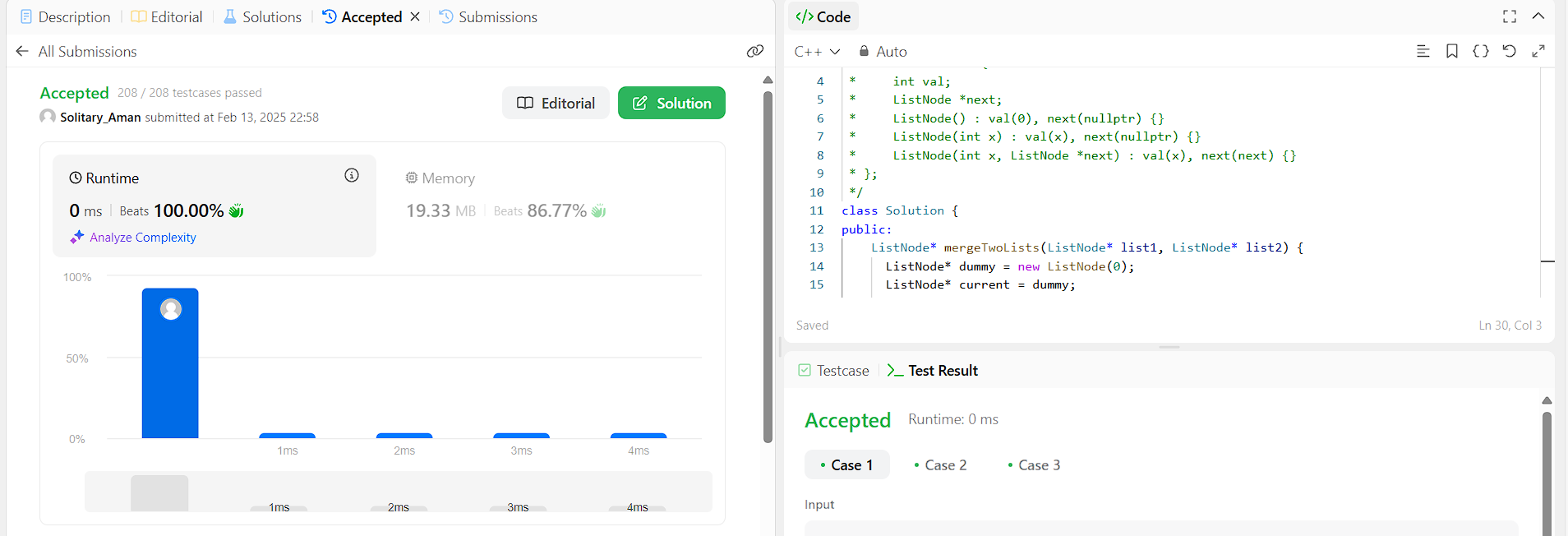
}

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

return dummy->next;

}

**Output:**

****

**Aim:** [Remove Duplicates from Sorted List II](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/)

**Code:**

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode dummy(0, head);

ListNode\* prev = &dummy;

while (head) {

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

head = head->next;

if (prev->next == head)

prev = prev->next;

else

prev->next = head->next;

head = head->next;

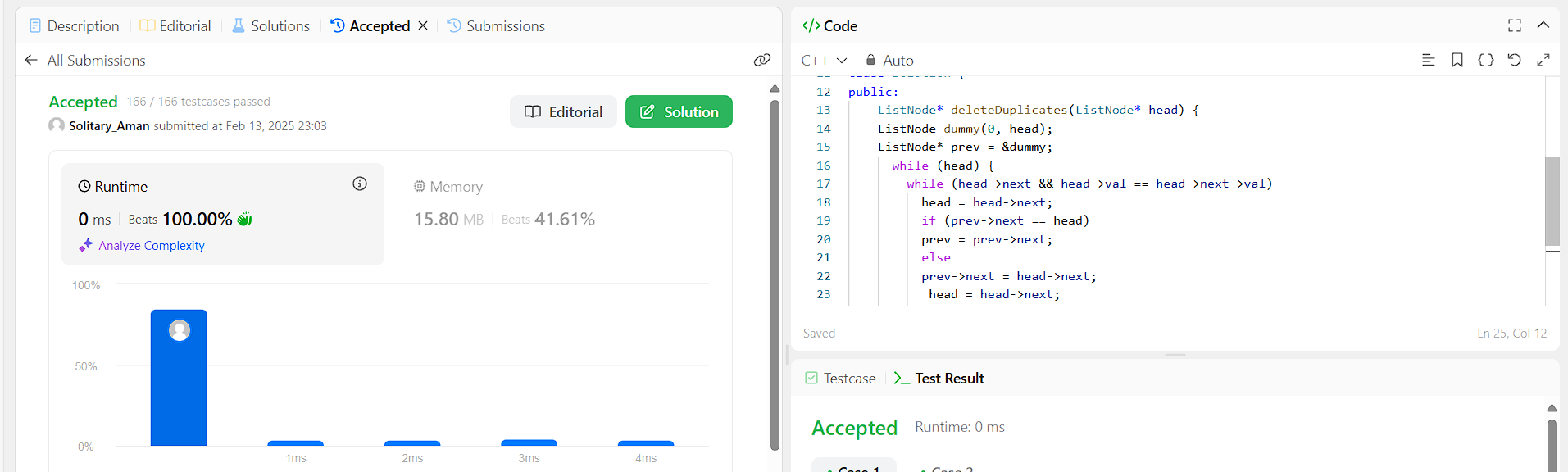
}

return dummy.next;

}

};

**Output:**

****

**Aim:** [[Linked List Cycle](https://leetcode.com/problems/linked-list-cycle/)](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/)

**Code:**

bool hasCycle(ListNode \*head) {

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

while(fast&& fast->next){

slow= slow->next;

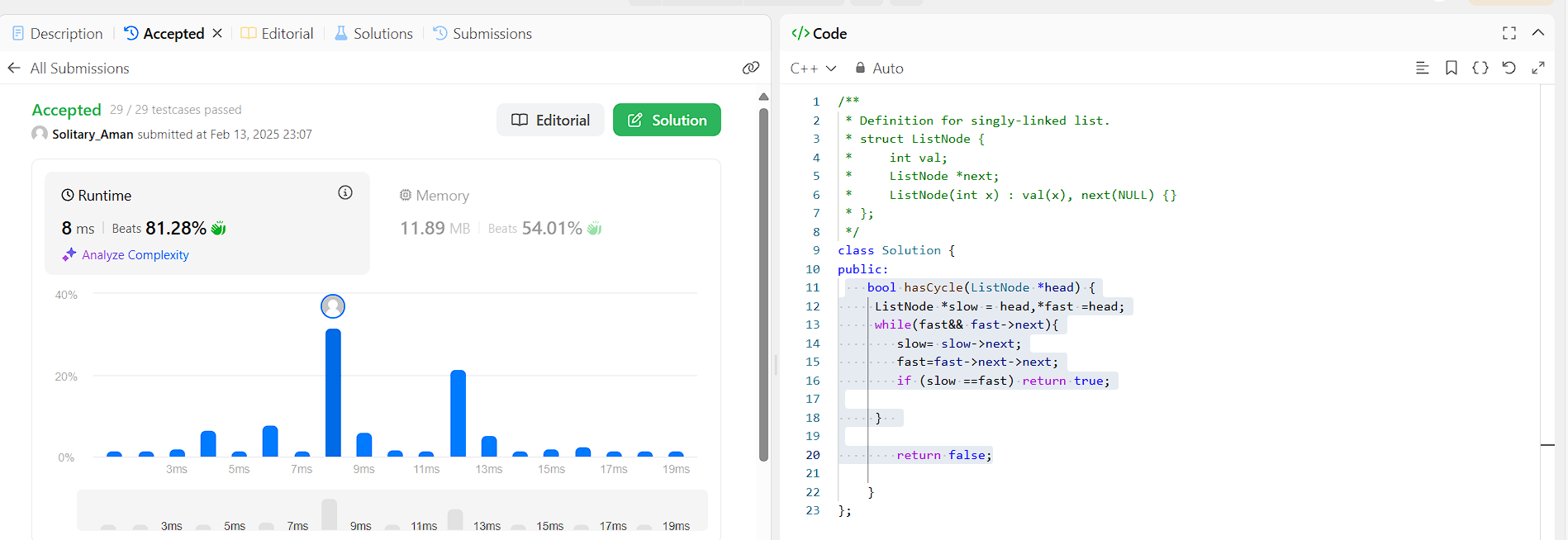
fast=fast->next->next;

if (slow ==fast) return true;

}

return false;

**Output:**

****

**Aim :** [Reverse Linked List II](https://leetcode.com/problems/reverse-linked-list-ii/)

**Code:**

public:

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

if (!head || left == right) {

return head;

}

ListNode\* dummy = new ListNode(0);

dummy->next = head;

ListNode\* prev = dummy;

for (int i = 0; i < left - 1; i++) {

prev = prev->next;

}

ListNode\* cur = prev->next;

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

ListNode\* temp = cur->next;

cur->next = temp->next;

temp->next = prev->next;

prev->next = temp;

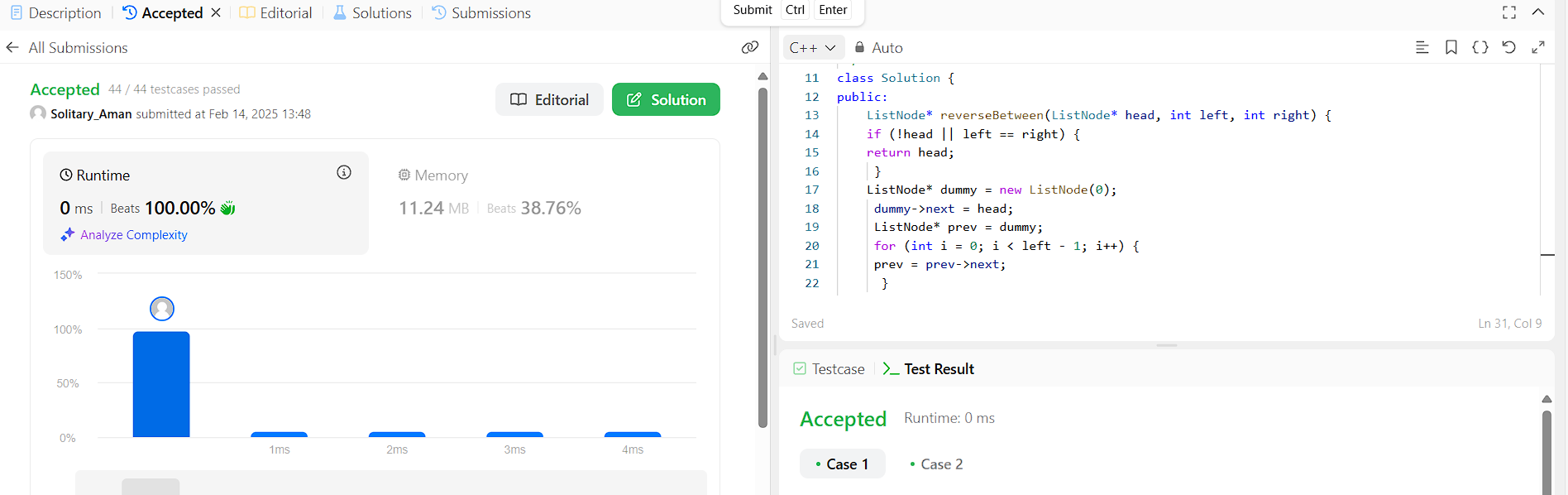
}

return dummy->next;

}

};

**Output:**



**Aim :** [Rotate List](https://leetcode.com/problems/rotate-list/)

**Code:**

if (!head) return nullptr;

int n = 1;

ListNode\* temp = head;

while (temp->next) {

n++;

temp = temp->next;

}

k %= n;

temp->next = head;

for (int i = 0; i < n - k; i++) temp = temp->next;

ListNode\* new\_head = temp->next;

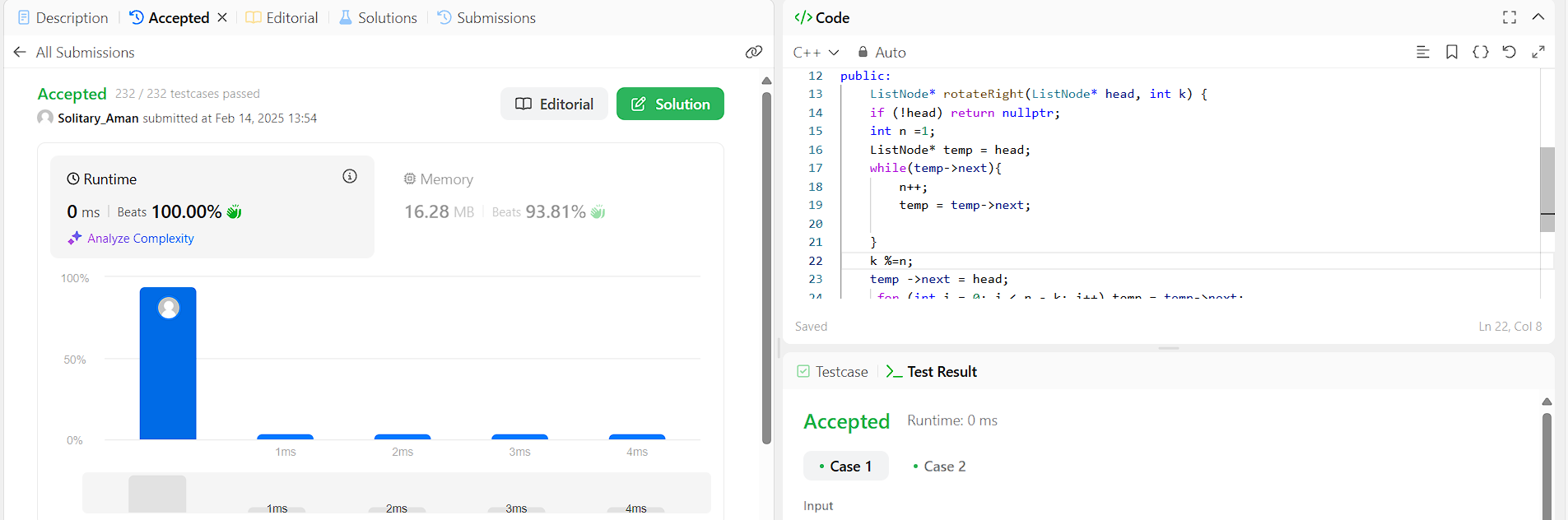
temp->next = nullptr;

return new\_head;

}

};

**Output:**



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

**Code:**

public:

ListNode\* getmid(ListNode\* head) {

ListNode\* slow = head;

ListNode\* fast = head->next;

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

slow = slow->next;

fast = fast->next->next;

}

return slow;

… right = sortList(right);

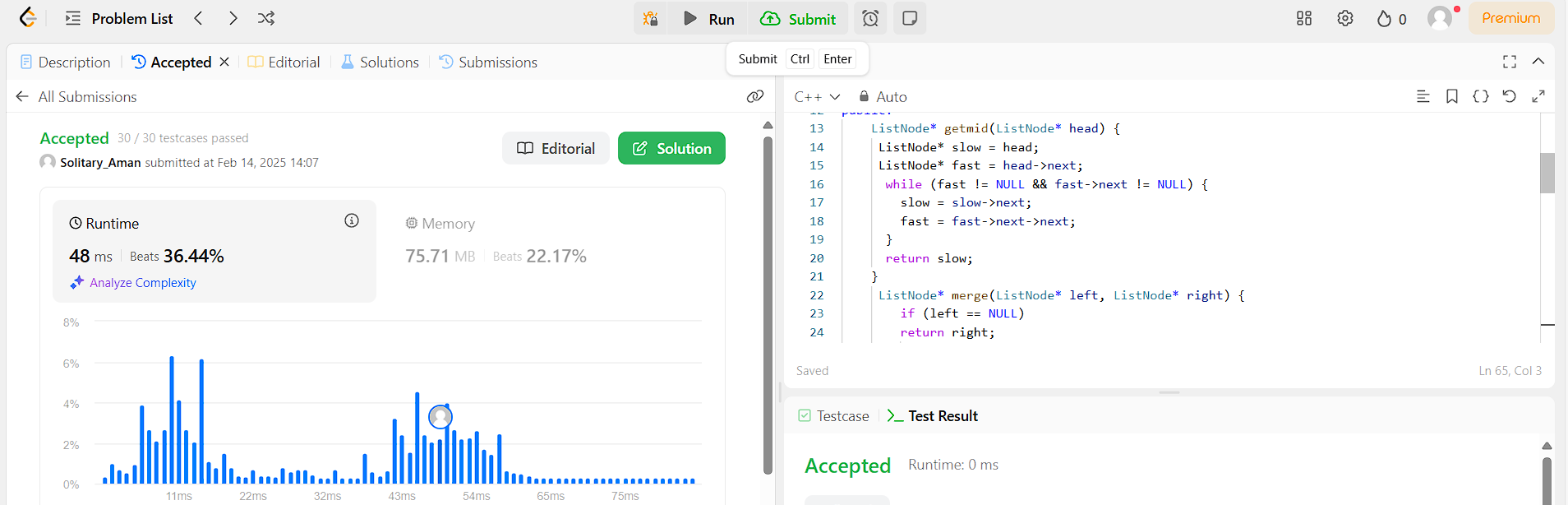
ListNode\* result = merge(left, right);

return result;

}

};

**Output:**



**Aim :** [Linked List Cycle II](https://leetcode.com/problems/linked-list-cycle-ii/)

**Code:**

public:

ListNode \*detectCycle(ListNode \*head) {

ListNode\* slow = head;

ListNode\* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) break;

}

if (!fast || !fast->next) return nullptr;

fast = head;

while (fast != slow) {

fast = fast->next;

slow = slow->next;

}

return slow;

}

};

**Output:**

