**DAY 1**

**Student Name:** Prabhnoor **UID:** 22BCS15639

**Branch:** BE-CSE **Section/Group:** 22BCS\_FL\_IOT-601/B

**Semester:** 6TH  **Date of Performance:** 03/02/2025

**Subject Name:** Advanced Programming Lab - 2

**Subject Code:** 22CSH-352

**LINKED LIST**

**Problem 1.** [Print linked list](https://www.geeksforgeeks.org/problems/print-linked-list-elements/0)

**Code:**

class Solution {

public:

void printList(Node \*head) {

if(!head) return;

Node\* temp=head;

while(temp!=NULL){

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

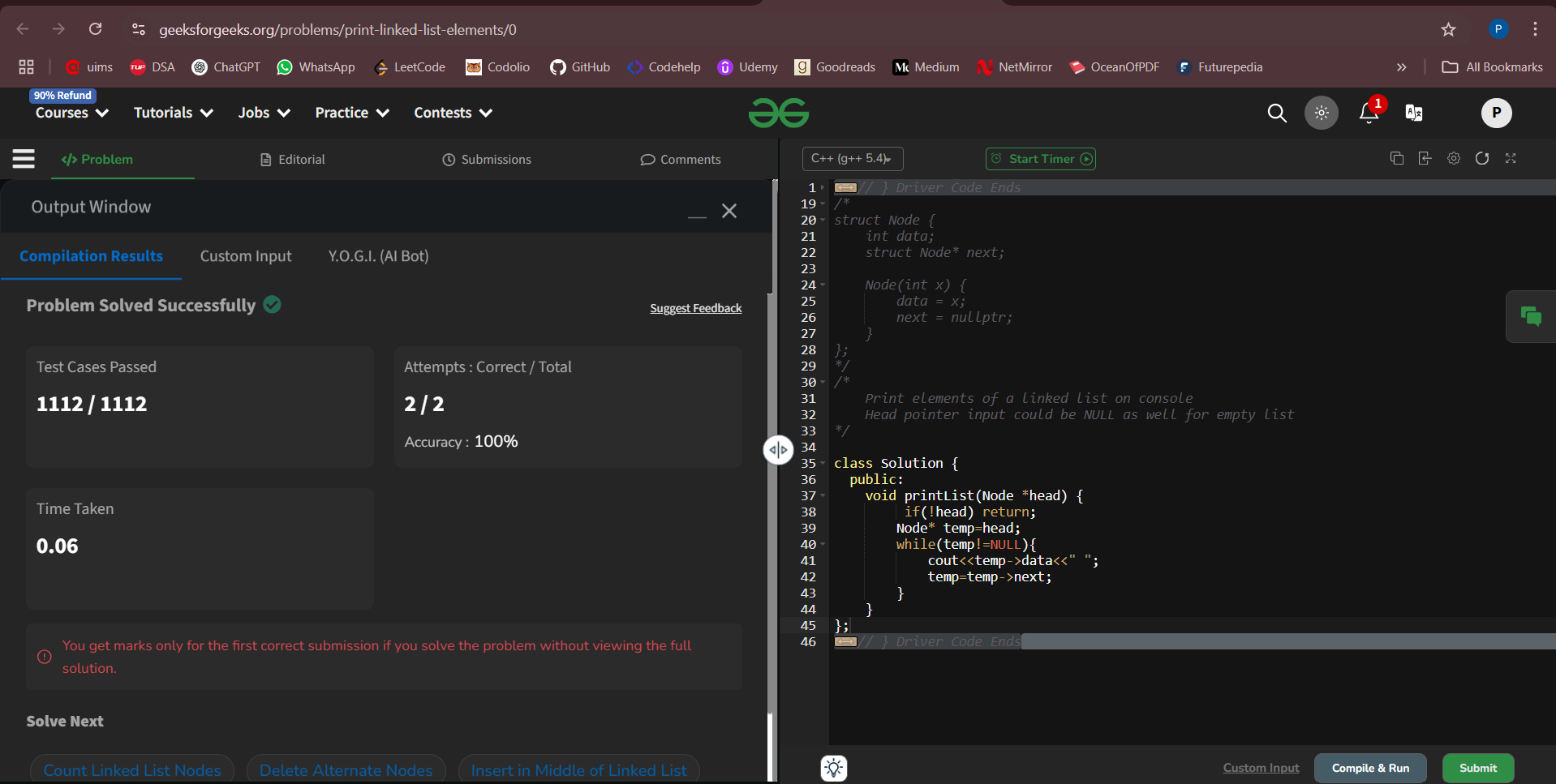
temp=temp->next;

}

}

};

**Acceptance Screenshot:**

****

**Problem 2.** [Remove duplicates from a sorted list.](https://leetcode.com/problems/remove-duplicates-from-sorted-list)

**Code:**

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

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

ListNode\* temp = head;

while (temp && temp->next) {

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

ListNode\* del = temp->next;

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

delete del;

} else {

temp = temp->next;

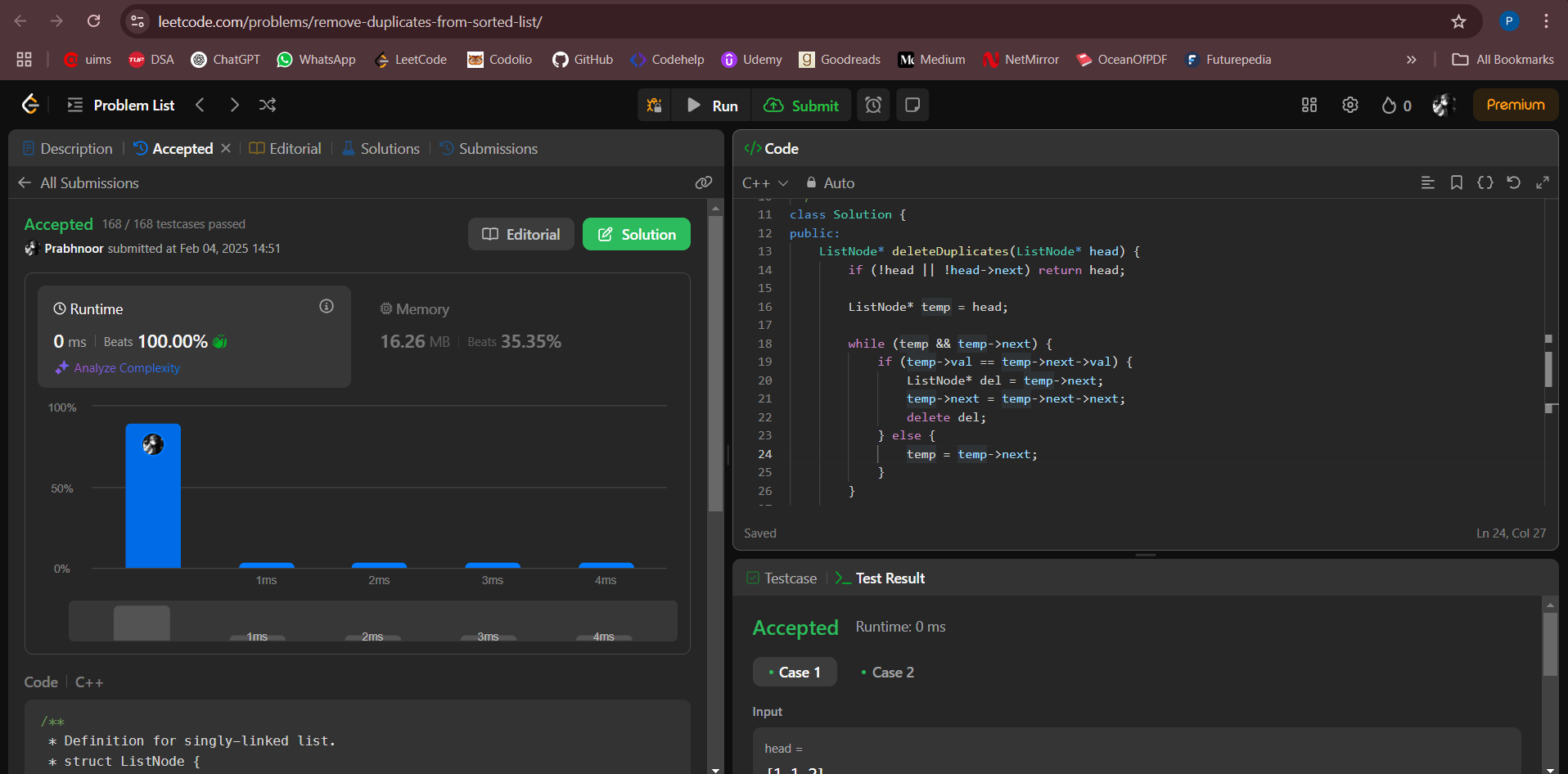
}

}

return head;

}};

**Acceptance Screenshot:**

****

**Problem 3.** [Reverse a linked list.](https://leetcode.com/problems/reverse-linked-list/)

**Code:**

class Solution {

public:

ListNode\* reverseList(ListNode\* head) {

ListNode\* temp=head;

ListNode\* prev=NULL;

ListNode\* front=head;

while(temp!=NULL){

front=temp->next;

temp->next=prev;

prev=temp;

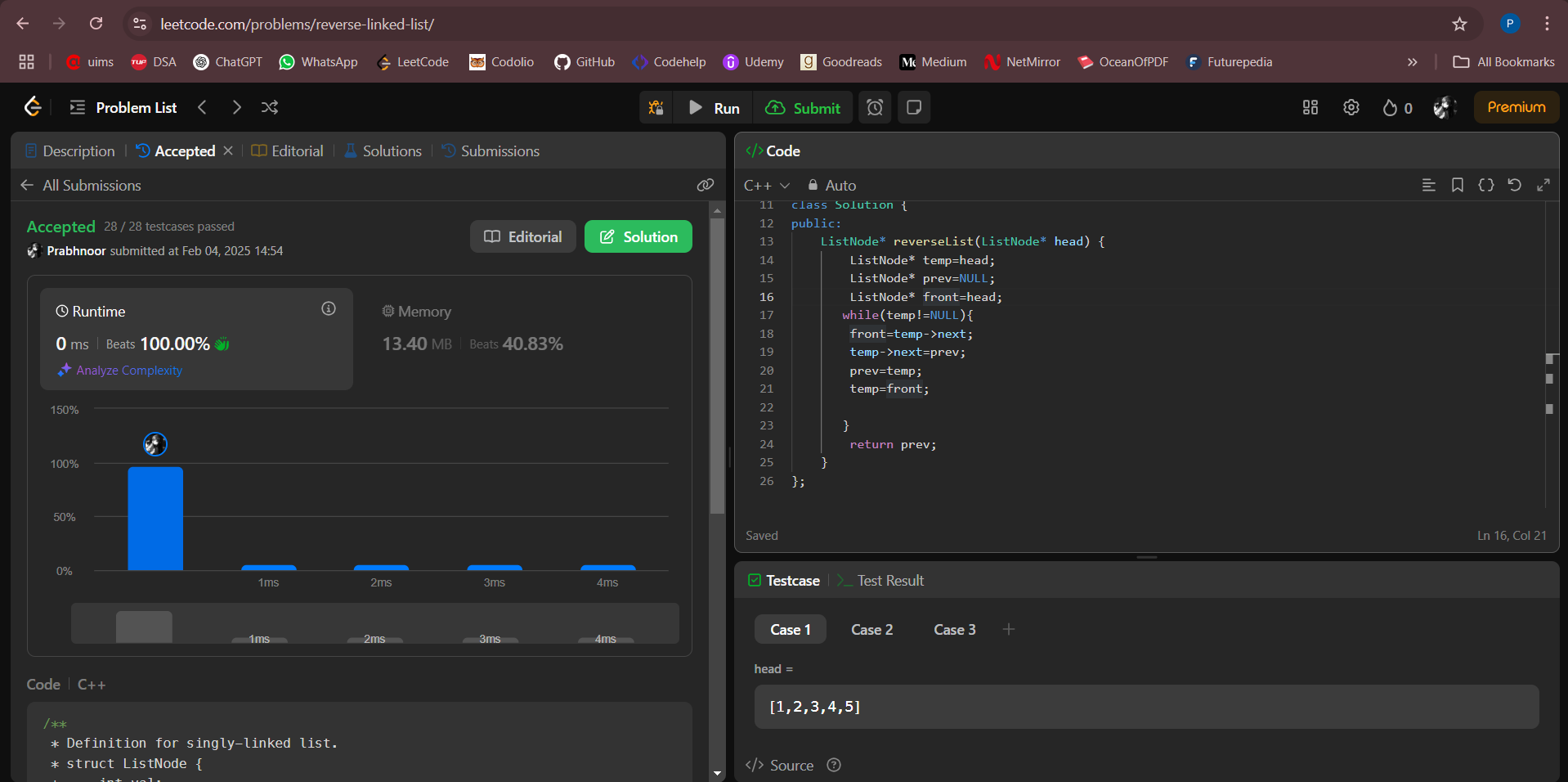
temp=front;

}

return prev;

}};

**Acceptance Screenshot:**



**Problem 4.** [Delete middle node of a list.](https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list)

**Code:**

class Solution {

public:

ListNode\* deleteMiddle(ListNode\* head) {

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

return NULL;

}

ListNode\* start=head;

ListNode\* end=head;

ListNode\* temp;

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

temp=start;

start=start->next;

end=end->next->next;

}

ListNode\* delNode=start;

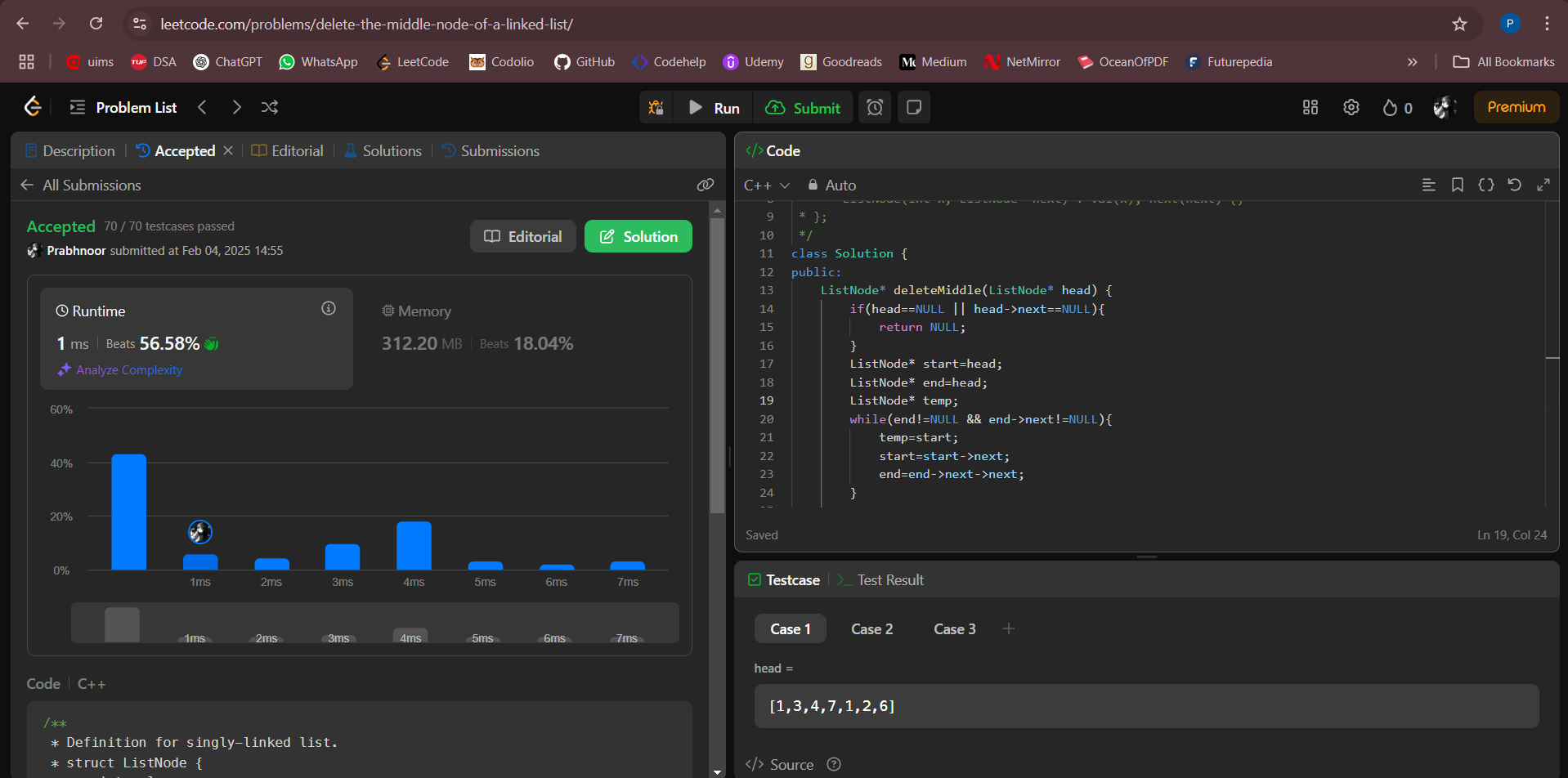
temp->next=temp->next->next;

delete delNode;

return head;

}};

**Acceptance Screenshot:**

****

**Problem 5.** [Merge two sorted linked list.](https://leetcode.com/problems/merge-two-sorted-lists)

**Code:**

class Solution {

public:

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

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

ListNode\* temp=dummy;

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

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

temp->next=list1;

temp=list1;

list1=list1->next;

}

else{

temp->next=list2;

temp=list2;

list2=list2->next;

}

}

if(list1!=NULL){

temp->next=list1;

}

if(list2!=NULL){

temp->next=list2;

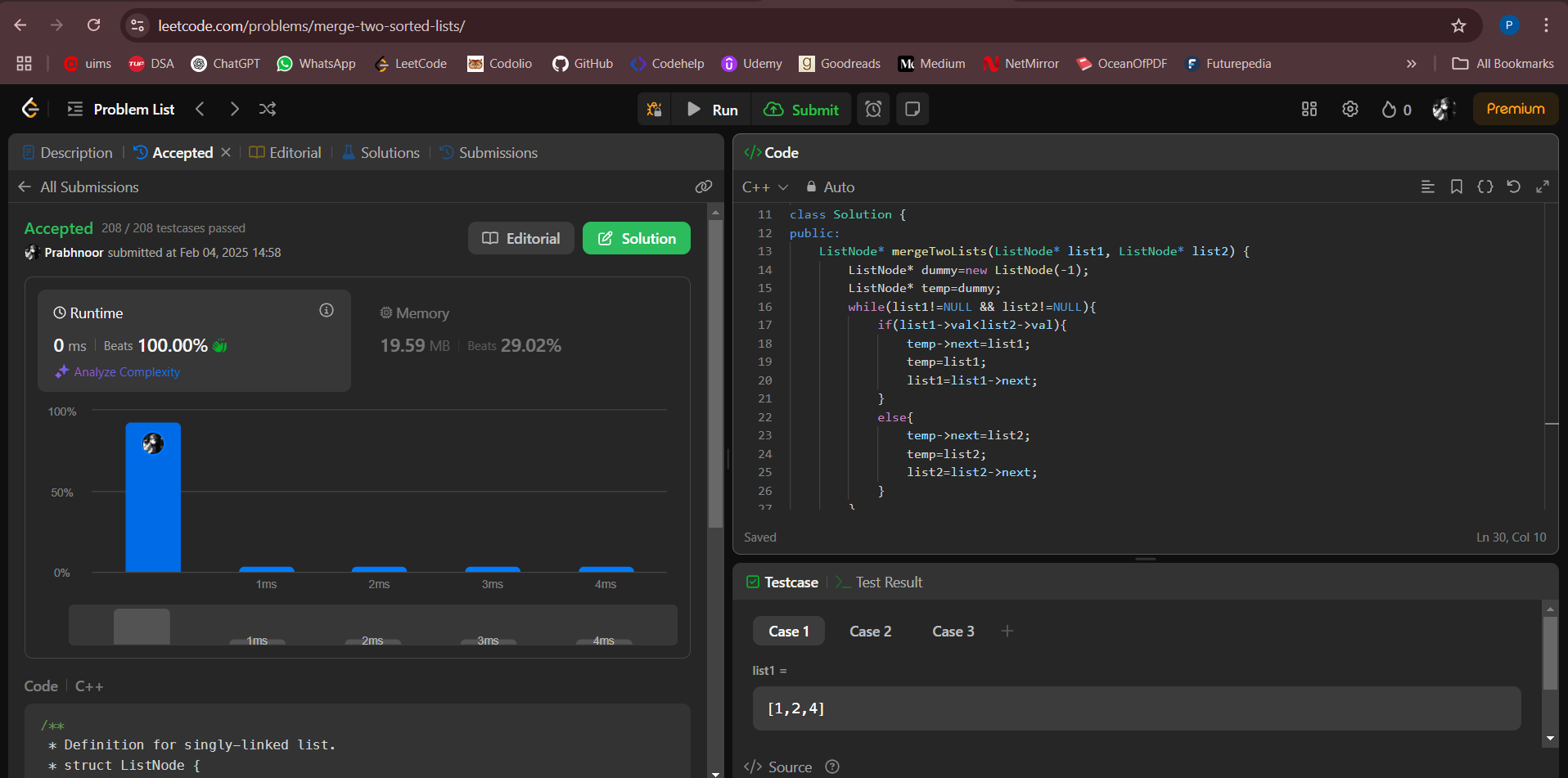
}

return dummy->next;

}

};

**Acceptance Screenshot:**

****

**Problem 6.** [Remove duplicates from sorted list 2.](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii)

**Code:**

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

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

        ListNode\* dummy = new ListNode(0);

        dummy->next = head;

        ListNode\* prev = dummy;

        while (head) {

            bool duplicate = false;

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

                head = head->next;

                duplicate = true;

            }

            if (duplicate) {

                prev->next = head->next;

            } else {

                prev = prev->next;

            }

            head = head->next;

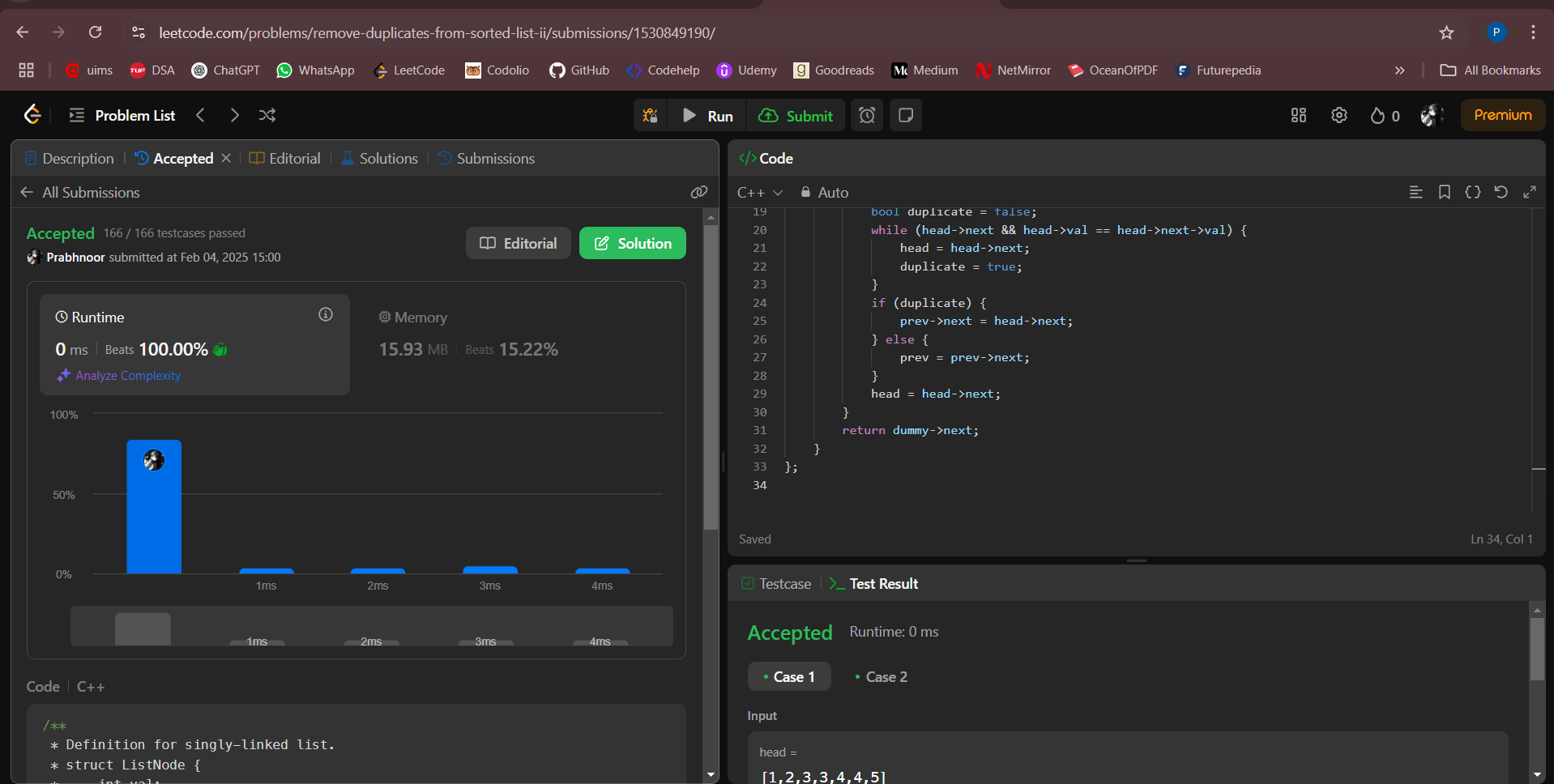
        }

        return dummy->next;

    }

};

**Acceptance Screenshot:**

****

**Problem 7.** [Detect a cycle in a linked list.](https://leetcode.com/problems/linked-list-cycle)

**Code:**

class Solution {

public:

bool hasCycle(ListNode \*head) {

ListNode\* slow=head;

ListNode\* fast=head;

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

slow=slow->next;

fast=fast->next->next;

if(slow==fast){

return true;

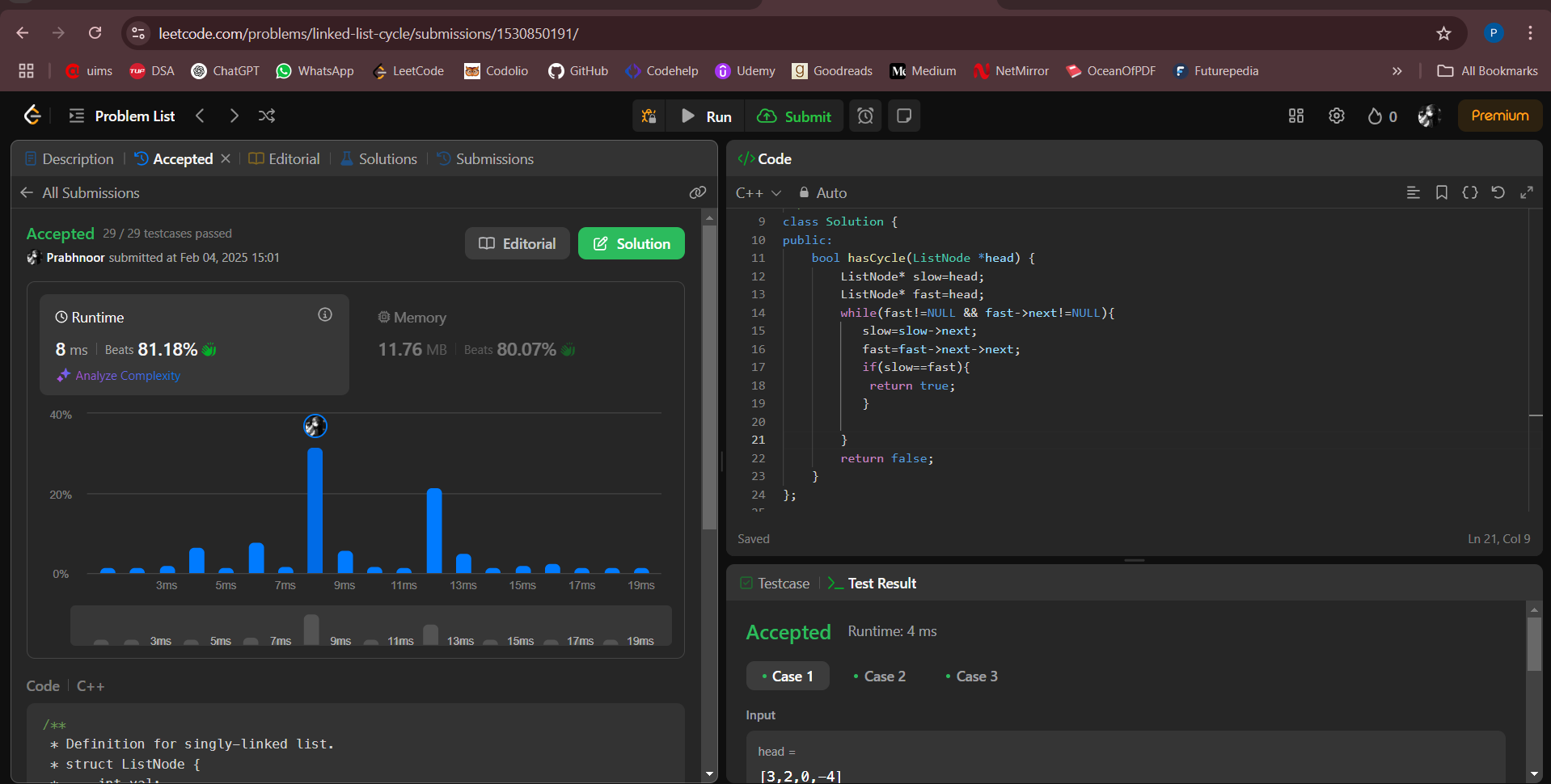
} }

return false;

}

};

**Acceptance Screenshot:**

****

**Problem 8.** [Reverse linked list 2.](https://leetcode.com/problems/reverse-linked-list-ii)

**Code:**

class Solution {

public:

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

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

        ListNode dummy(0);

        dummy.next = head;

        ListNode\* prev = &dummy;

        for (int i = 1; i < left; ++i) prev = prev->next;

        ListNode\* curr = prev->next;

        ListNode\* nextNode = nullptr;

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

            nextNode = curr->next;

            curr->next = nextNode->next;

            nextNode->next = prev->next;

            prev->next = nextNode;

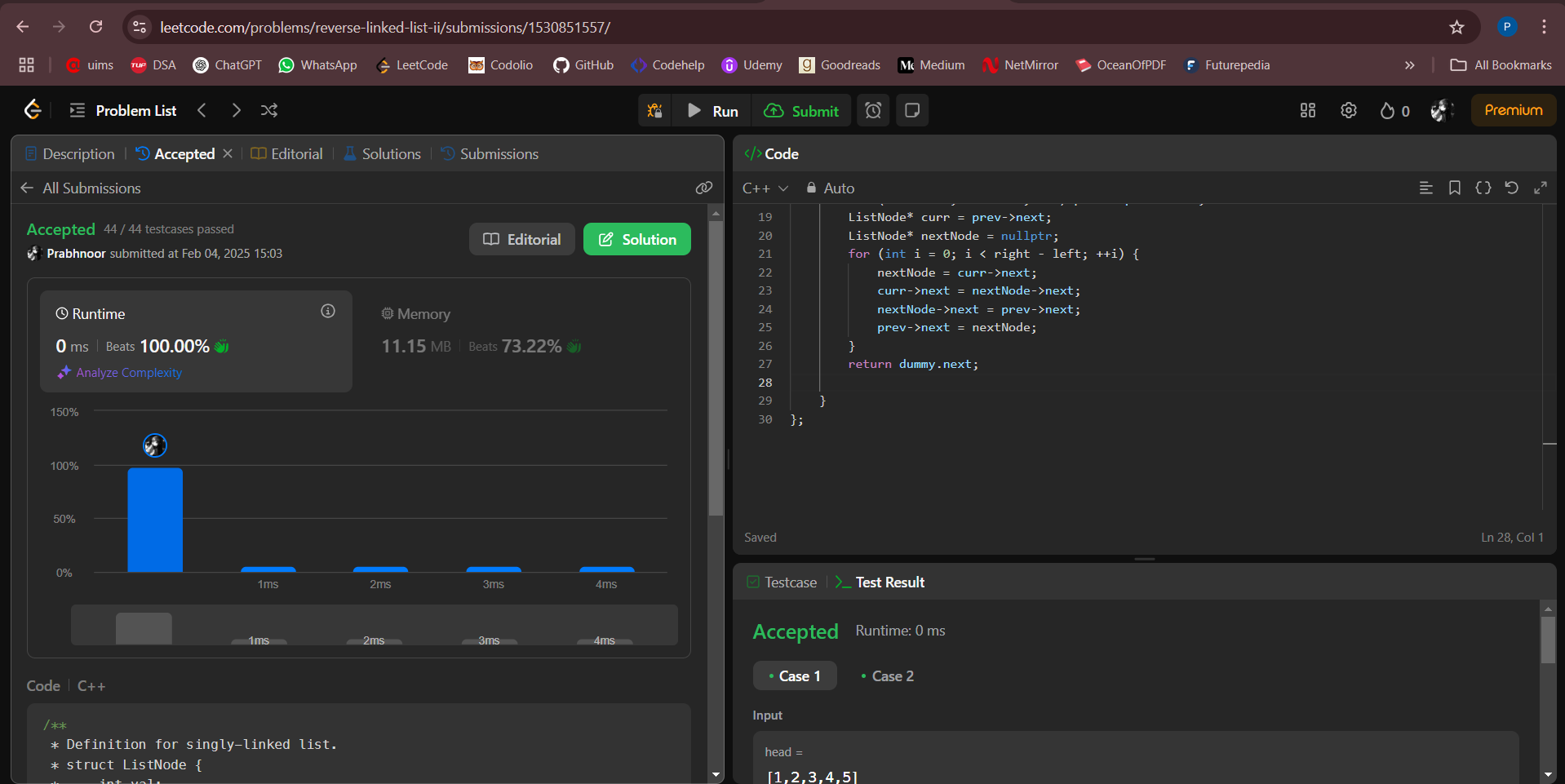
        }

        return dummy.next;

    }

};

**Acceptance Screenshot:**



**Problem 9.**  [Rotate a list.](https://leetcode.com/problems/rotate-list)

**Code:**

class Solution {

public:

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

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

ListNode\* temp=head;

int size=0;

while(temp!=NULL){

size++;

temp=temp->next;

}

if(k%size==0) return head;

k=k%size;

ListNode\* front=head;

temp=head;

while(temp->next!=NULL){

temp=temp->next;

}

temp->next=front;

int rotateList=size-k;

while(temp!=NULL && rotateList>0){

temp=temp->next;

rotateList--;

}

if(temp->next!=NULL) head=temp->next;

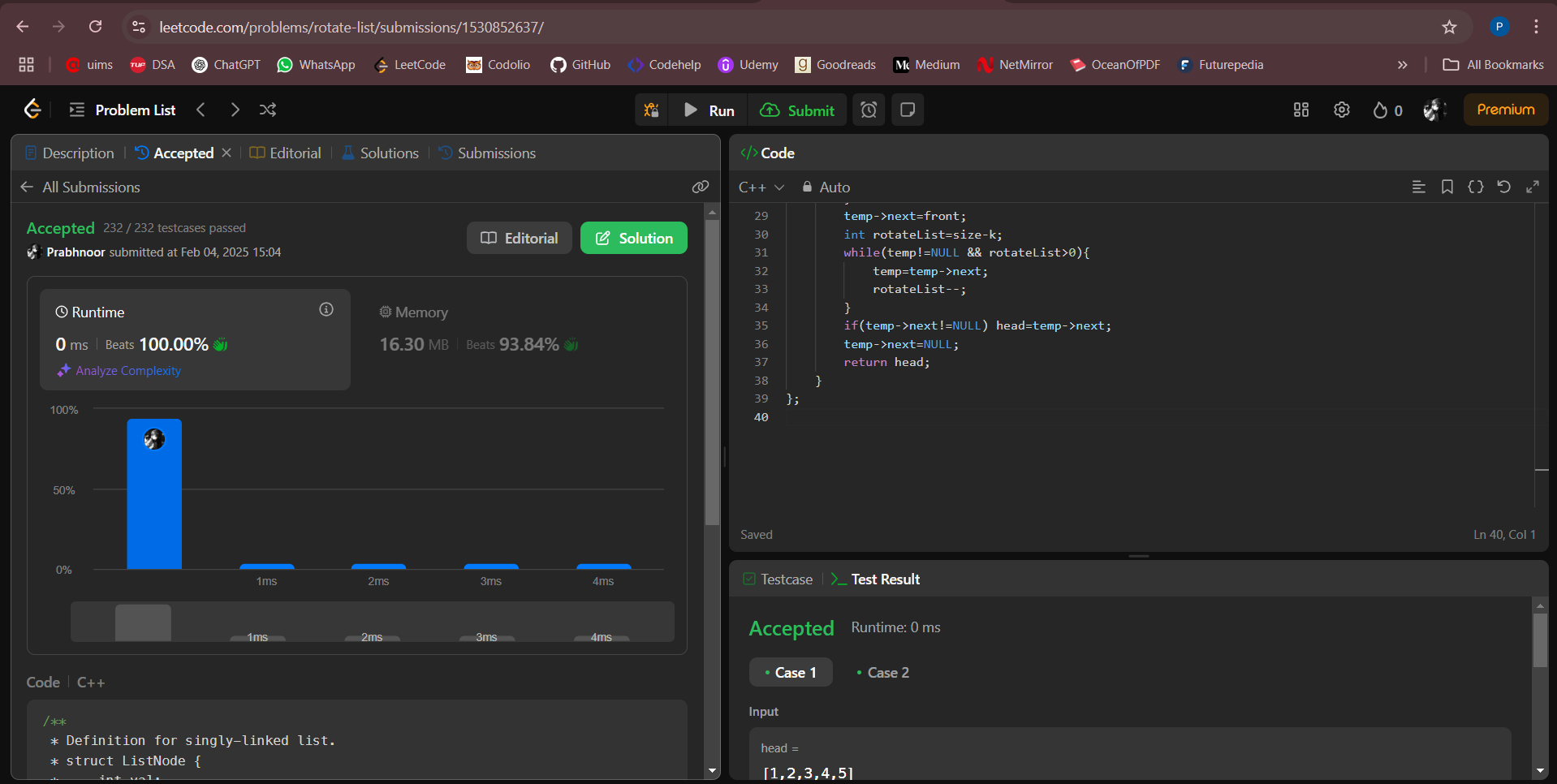
temp->next=NULL;

return head;

}

};

**Acceptance Screenshot:**

****

**Problem 10.**  [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 (ListNode\* list : lists) {

            if (list) minHeap.push(list);

        }

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

        ListNode\* current = dummy;

        while (!minHeap.empty()) {

            ListNode\* smallest = minHeap.top();

            minHeap.pop();

            current->next = smallest;

            current = current->next;

            if (smallest->next) {

                minHeap.push(smallest->next);

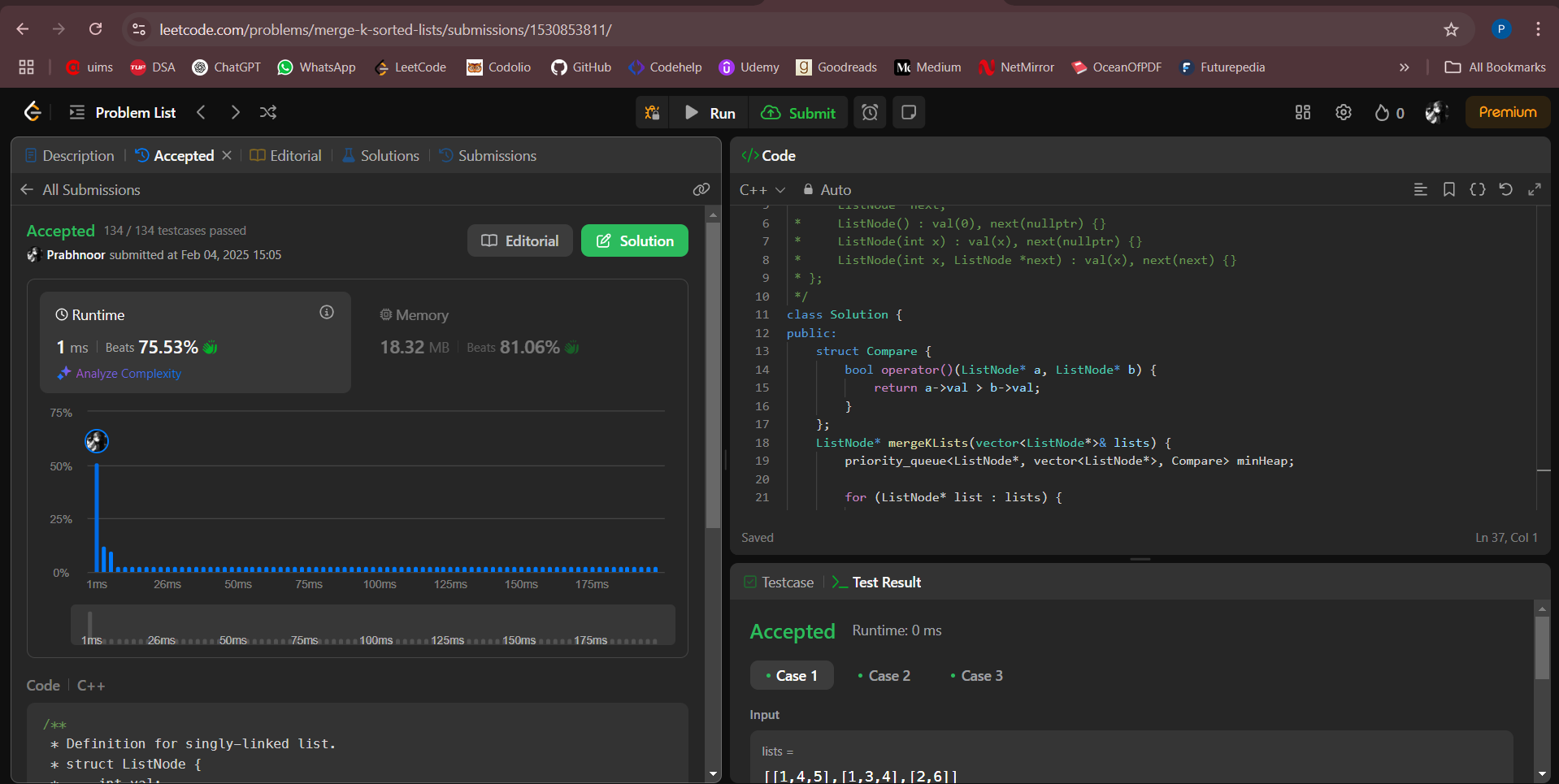
            }        }

        return dummy->next;

    }

};

**Acceptance Screenshot:**



**Problem 11.** [Sort List.](https://leetcode.com/problems/sort-list/description/)

**Code:**

class Solution {

public:

ListNode\* sortList(ListNode\* head) {

vector<int> s;

ListNode\* temp =head;

while(temp){

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

temp=temp->next;

}

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

int i =0;

temp = head;

while(temp){

temp->val=s[i];

temp=temp->next;

i++;

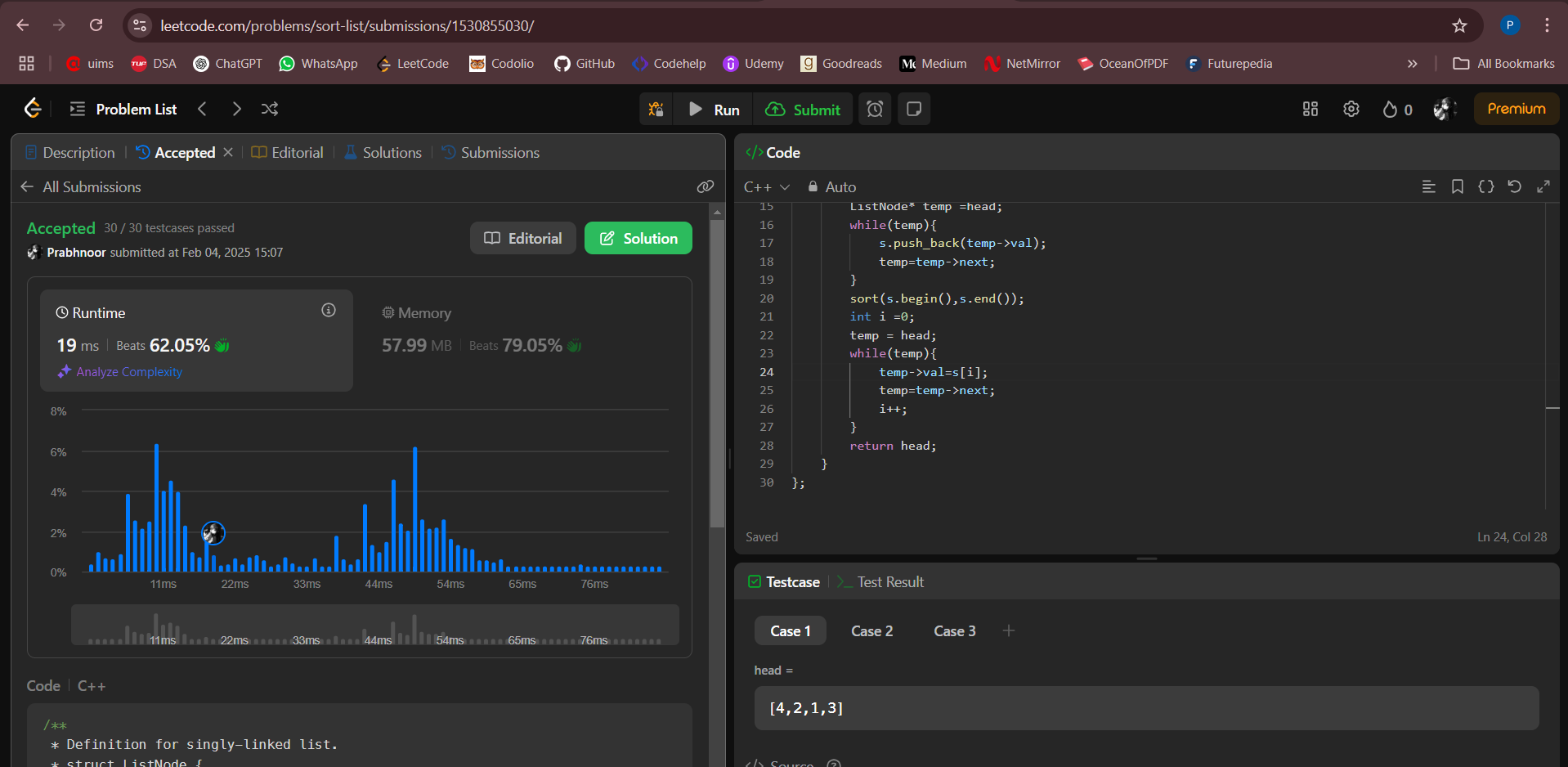
}

return head;

}

};

**Acceptance Screenshot:**

****