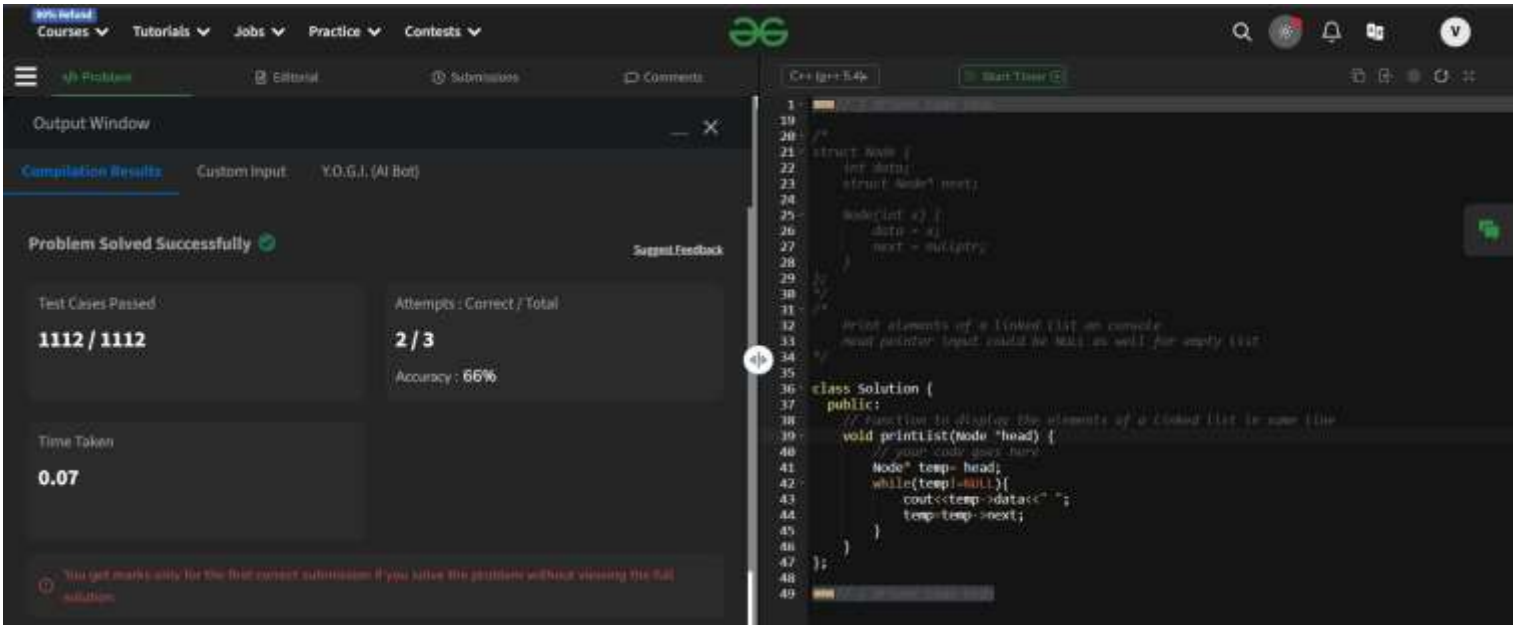
Advanced Programming Assignment 3

Name- Vivek Singh UID-22BCS11457 Section- 607-B

Qs 1. Print Linked List:

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;
        while(temp!=NULL){
            cout<<temp->data<<" ";
            temp=temp->next;
        }
    }
};
```

Submission:

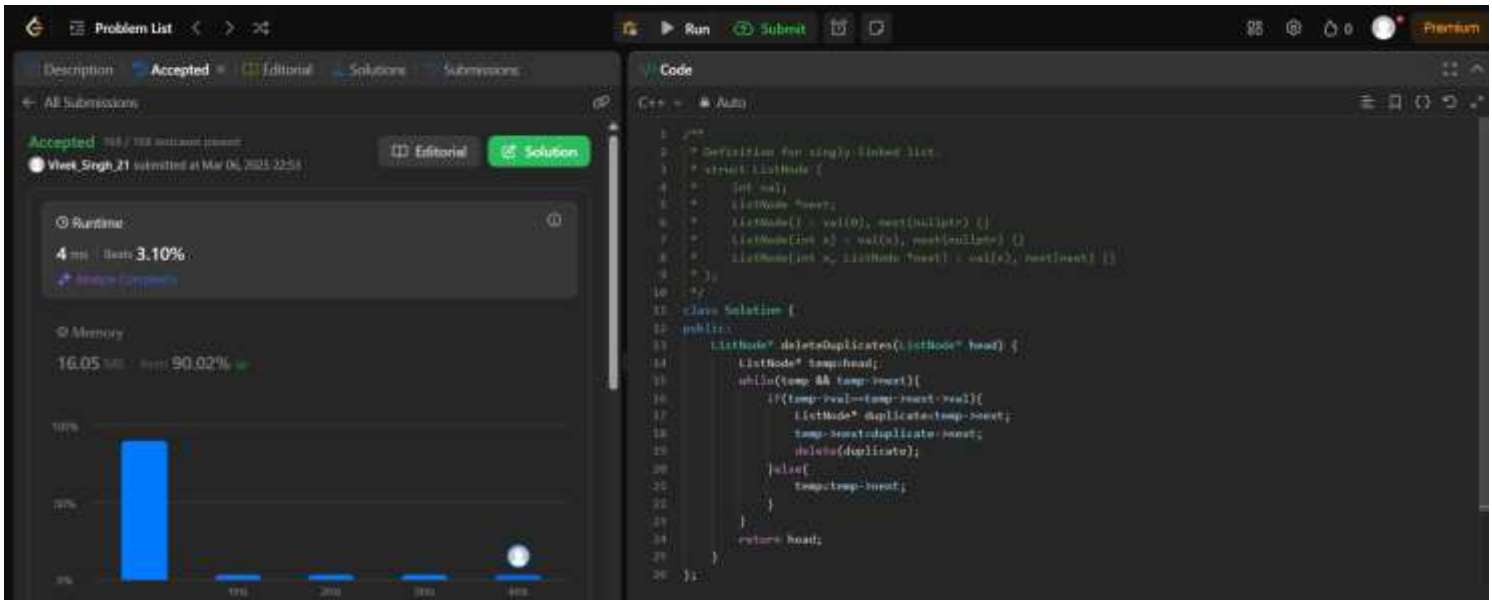


Qs 2. Remove Duplicates from Sorted List:

Code:

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* temp=head;
        while(temp && temp->next){
            if(temp->val==temp->next->val){
                ListNode* duplicate=temp->next;
                temp->next=duplicate->next;
                delete(duplicate);
            }else{
                temp=temp->next;
            }
        }
        return head;
    }};
```

Submission:



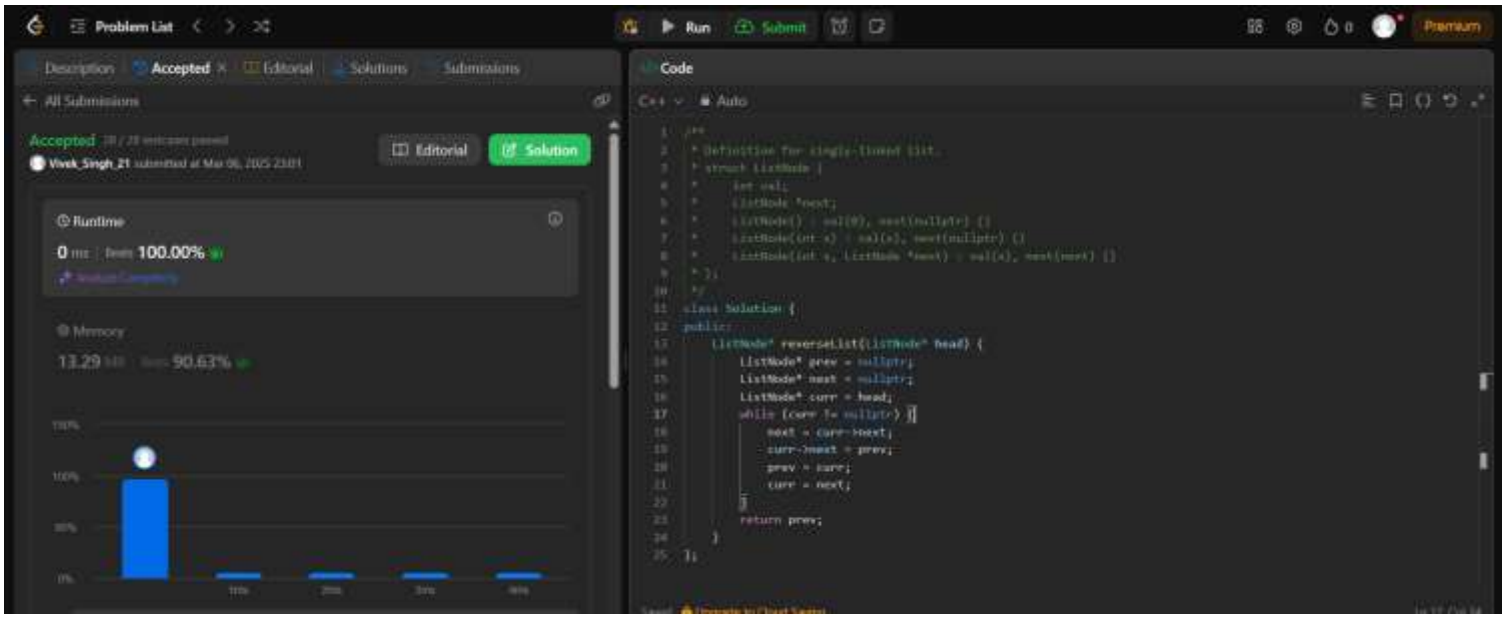
Qs 3. Reverse a linked list:

Code: class Solution {

public:

```
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr;
        ListNode* next = nullptr;
        ListNode* curr = head;
        while (curr != nullptr) {
            next = curr->next;
            curr->next = prev;
            prev = curr;
            curr = next;
        }
        return prev;
    }
};
```

Submission:

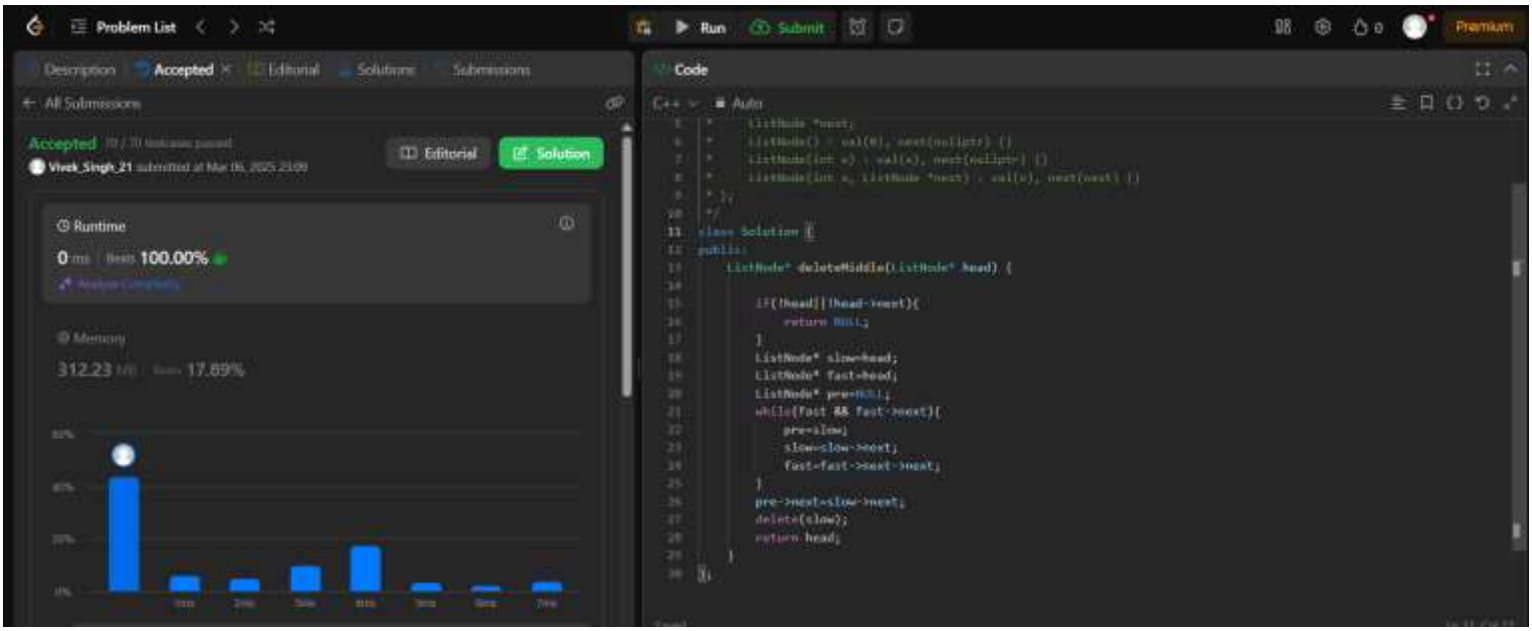


Qs 4. Delete middle node of linked list:

Code: class Solution {

public:

```
    ListNode* deleteMiddle(ListNode* head) {

        if(!head||!head->next){
            return NULL;
        }
        ListNode* slow=head;
        ListNode* fast=head;
        ListNode* pre=NULL;
        while(fast && fast->next){
            pre=slow;
            slow=slow->next;
            fast=fast->next->next;
        }
        pre->next=slow->next;
        delete(slow);
        return head;
    }
};Submission:
```



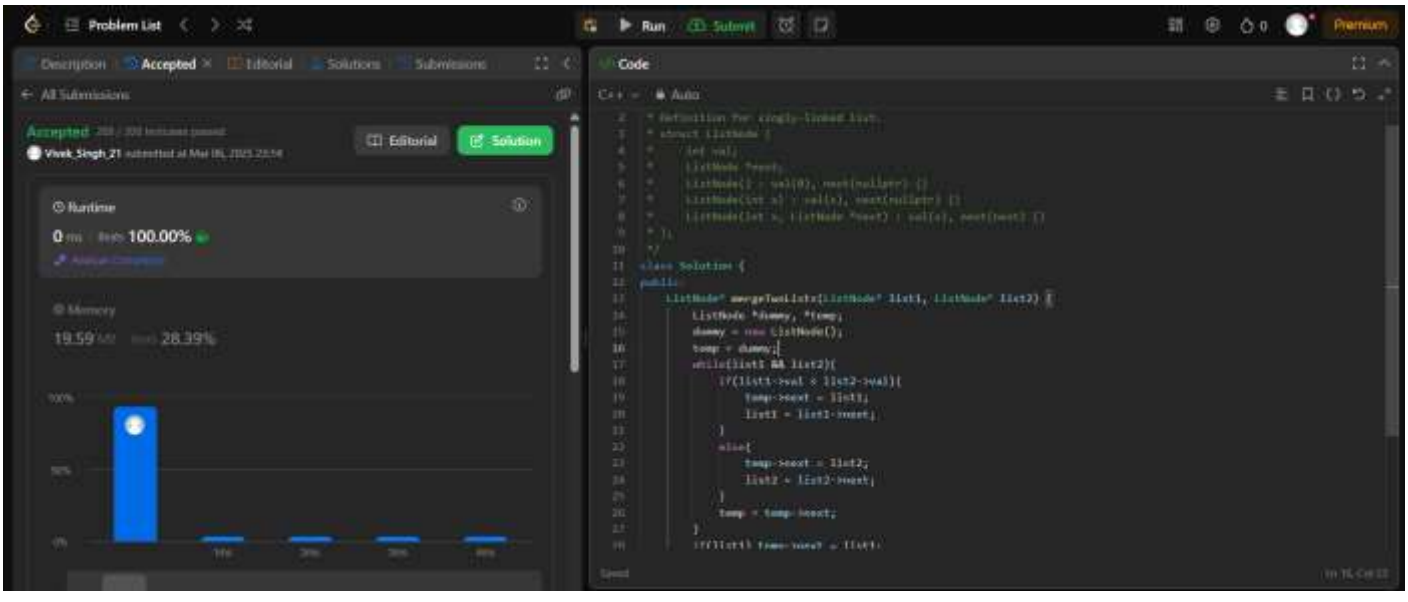
Qs 5. Merge two sorted lists:

Code: class Solution {

public:

```
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        ListNode *dummy, *temp;
        dummy = new ListNode();
        temp = dummy;
        while(list1 && list2){
            if(list1->val < list2->val){
                temp->next = list1;
                list1 = list1->next;
            }
            else{
                temp->next = list2;
                list2 = list2->next;
            }
            temp = temp->next;
        }
        if(list1) temp->next = list1;
        if(list2) temp->next = list2;

        return dummy->next;
    }
};
```

Submission:



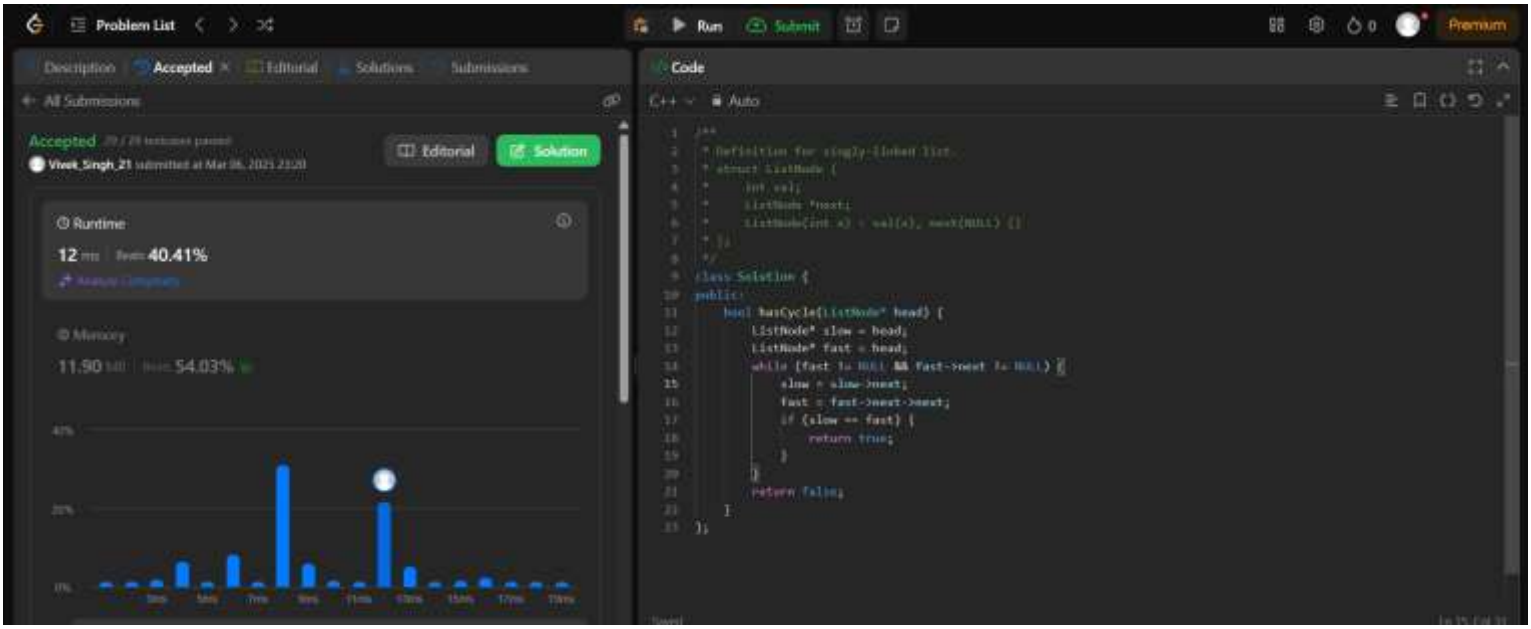
Qs 6. Detect cycle in a linked list:

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;
    }
};
```

Submission:



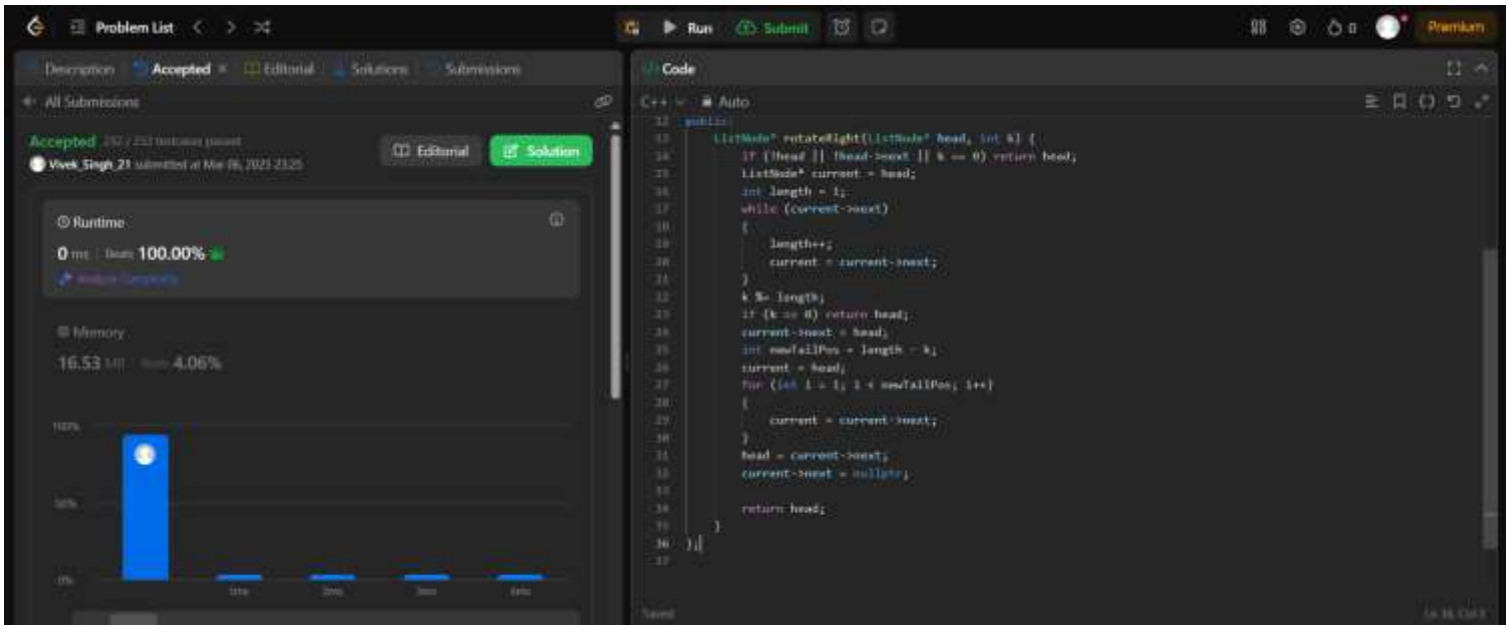
Qs 7. Rotate a list:

Code: class Solution {

public:

```
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        ListNode* current = head;
        int length = 1;
        while (current->next)
        {
            length++;
            current = current->next;
        }
        k %= length;
        if (k == 0) return head;
        current->next = head;
        int newTailPos = length - k;
        current = head;
        for (int i = 1; i < newTailPos; i++)
        {
            current = current->next;
        }
        head = current->next;
        current->next = nullptr;

        return head;
    }
};
```

Submission:



Qs 8. Sort List:

Code: class Solution {

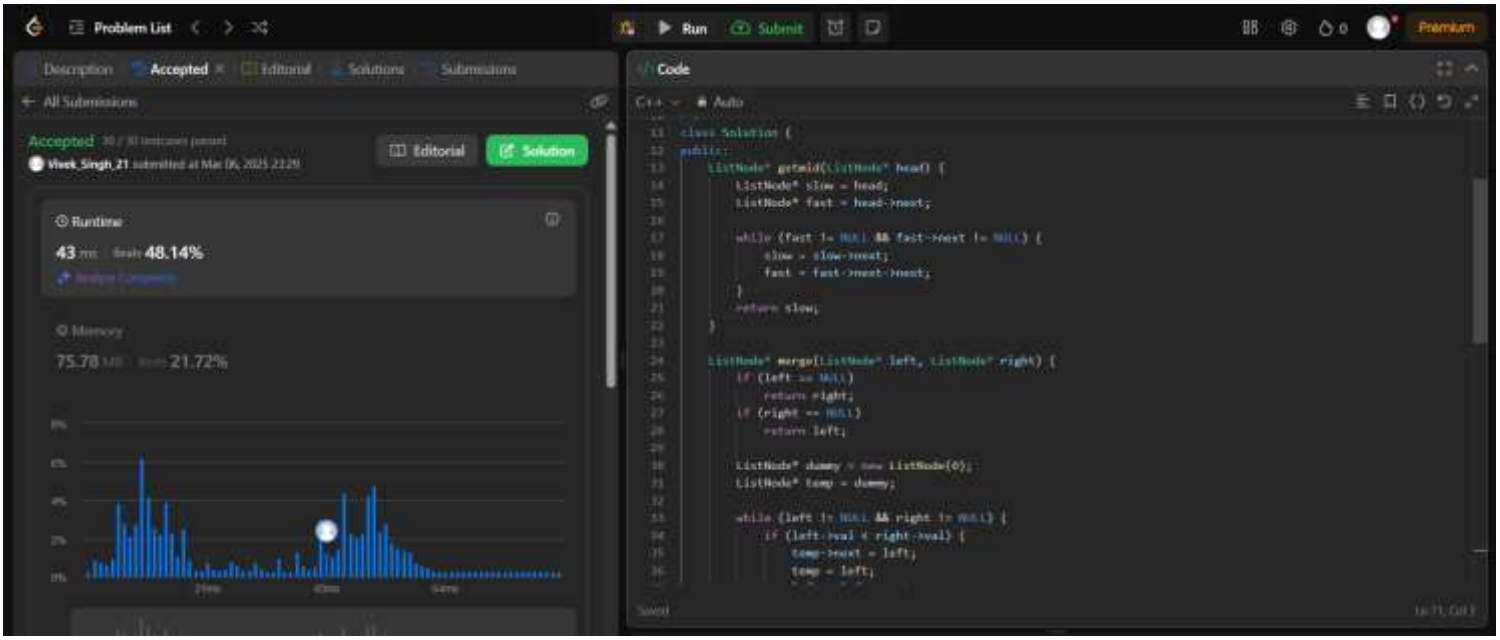
public:

```
    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;
    }

    ListNode* merge(ListNode* left, ListNode* right) {
        if (left == NULL)
            return right;
        if (right == NULL)
```

```
        return left;

    ListNode* dummy = new ListNode(0);
    ListNode* temp = dummy;

    while (left != NULL && right != NULL) {
        if (left->val < right->val) {
            temp->next = left;
            temp = left;
            left = left->next;
        } else {
            temp->next = right;
            temp = right;
            right = right->next;
        }
    }
    while (left != NULL) {
        temp->next = left;
        temp = left;
        left = left->next;
    }
    while (right != NULL) {
        temp->next = right;
        temp = right;
        right = right->next;
    }
    dummy = dummy->next;
    return dummy;
}
```

```
    ListNode* sortList(ListNode* head) {
        if (head == NULL || head->next == NULL) return head;

        ListNode* mid = getmid(head);
        ListNode* left = head;
        ListNode* right = mid->next;
        mid->next = NULL;
        left = sortList(left);
        right = sortList(right);
        ListNode* result = merge(left, right);

        return result;
    }
};
```

Submission:



Qs 9. Merge K sorted list:

Code: class Solution {

public:

```
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<pair<int,ListNode*>, vector<pair<int, ListNode*>> , greater<pair<int,ListNode*>>> pq;
```

```
        for(int i=0;i<lists.size();i++){
            if(lists[i])    pq.push({lists[i]->val , lists[i]});
        }
        ListNode* dummy = new ListNode(-1);
        ListNode* temp = dummy;

        while(!pq.empty()){
            auto it = pq.top();
            if(it.second->next){
                pq.push({it.second->next->val,it.second->next});
            }
            pq.pop();
            temp->next = it.second;
            temp=temp->next;
        }
       return dummy->next;
    }
};
```

Submission: