**Name : Anjali**

**UID : 22BCS10665  
Batch : 22BCS\_IOT-614(B)**

**Assignment – 3**

**Problem-1 : Print Linked List**  
**Code:**

class Solution {

public:

void printList(Node \*head) {

Node\* temp = head;

while (temp != nullptr) {

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

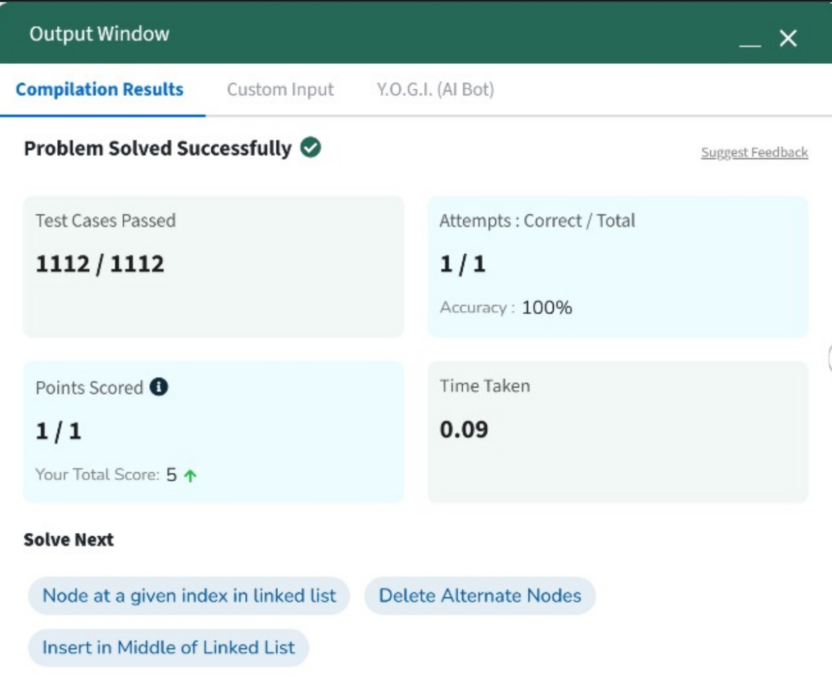
temp = temp->next;

}

}

};

**Output SS:**



**Problem-2 : Remove Duplicates From A Sorted List**  
**Code:**

class Solution {

public:

    ListNode\* deleteDuplicates(ListNode\* head) {

        if (head == nullptr) return head;

        ListNode\* current = head;

        while (current->next != nullptr) {

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

                ListNode\* temp = current->next;

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

                delete temp;

            } else {

                current = current->next;

            }

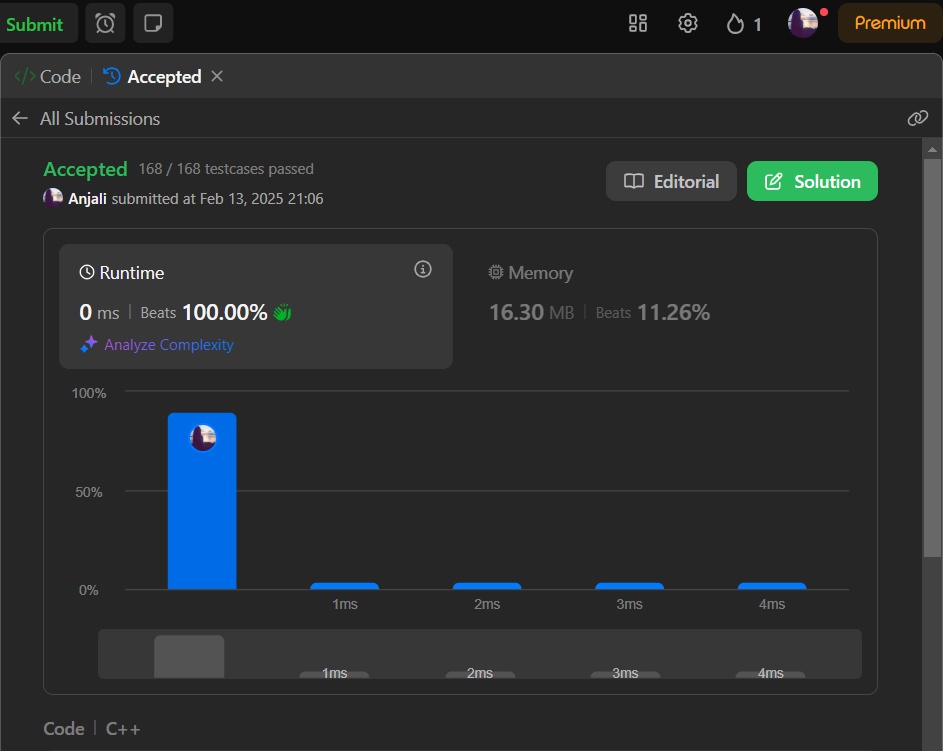
        }

        return head;

    }

};

**Output SS:**



**Problem-3 : Reverse Linked List**  
**Code:**

class Solution {

public:

    ListNode\* reverseList(ListNode\* head) {

        ListNode\* prev = nullptr;

        ListNode\* current = head;

        ListNode\* next = nullptr;

        while (current != nullptr) {

            next = current->next;

            current->next = prev;

            prev = current;

            current = next;

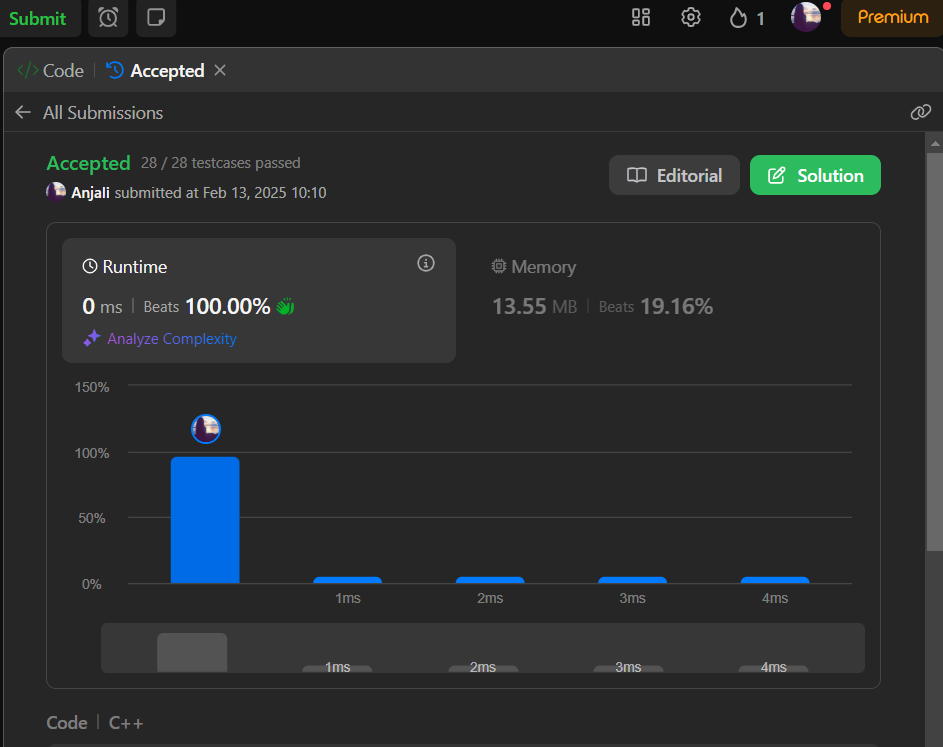
        }

        return prev;

    }

};

**Output SS:**



**Problem-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;

        }

        if (prev) {

            prev->next = slow->next;

        }

        return head;

    }

};

**Output SS:**



**Problem-5 : Print Linked List**  
**Code:**

class Solution {

public:

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

        ListNode dummy(0);

        ListNode\* tail = &dummy;

        while (list1 && list2) {

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

                tail->next = list1;

                list1 = list1->next;

            } else {

                tail->next = list2;

                list2 = list2->next;

            }

            tail = tail->next;

        }

        if (list1) tail->next = list1;

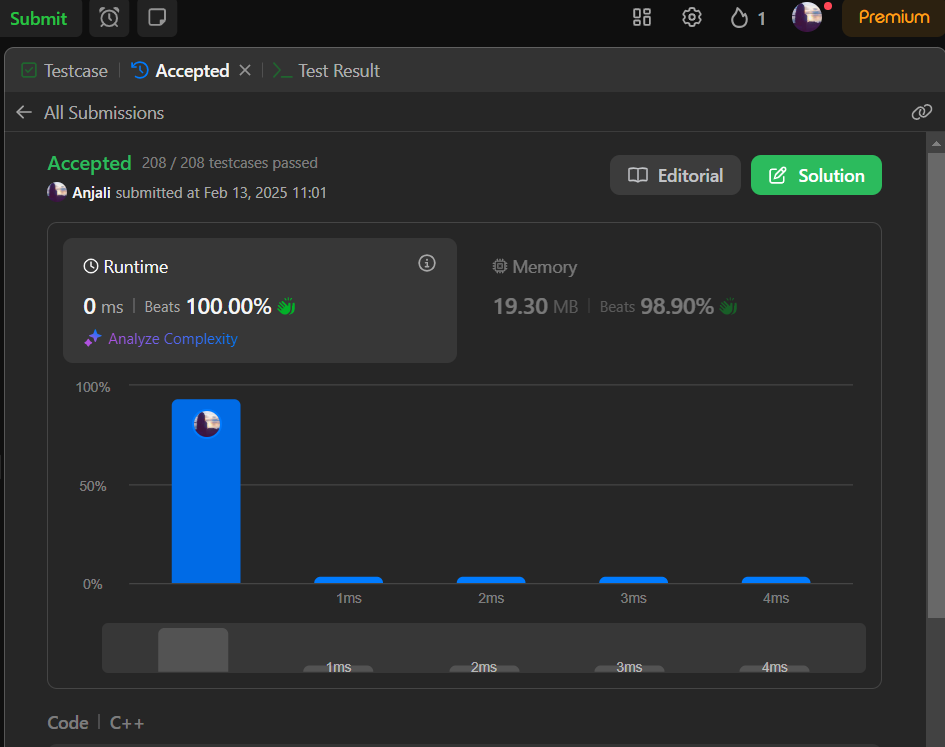
        if (list2) tail->next = list2;

        return dummy.next;

    }

};

**Output SS:**



**Problem-6 : Remove Duplicates From Sorted List II**  
**Code:**

class Solution {

public:

ListNode\* deleteDuplicates(ListNode\* head) {

ListNode\* dummy = new ListNode(0, head);

ListNode\* prev = dummy;

while (head) {

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

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

head = head->next;

}

prev->next = head->next;

} else {

prev = prev->next;

}

head = head->next;

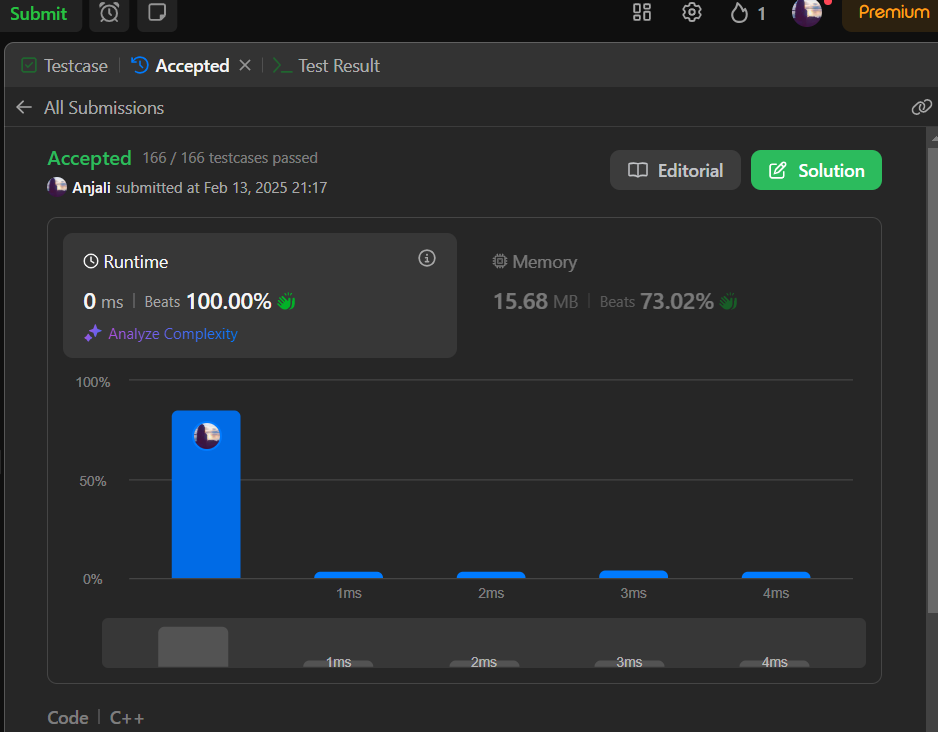
}

return dummy->next;

}

};

**Output SS:**



**Problem-7 : Detect A Cycle In A Linked List**  
**Code:**

class Solution {

public:

bool hasCycle(ListNode \*head) {

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

ListNode\* slow = head;

ListNode\* fast = head;

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

return true;

}

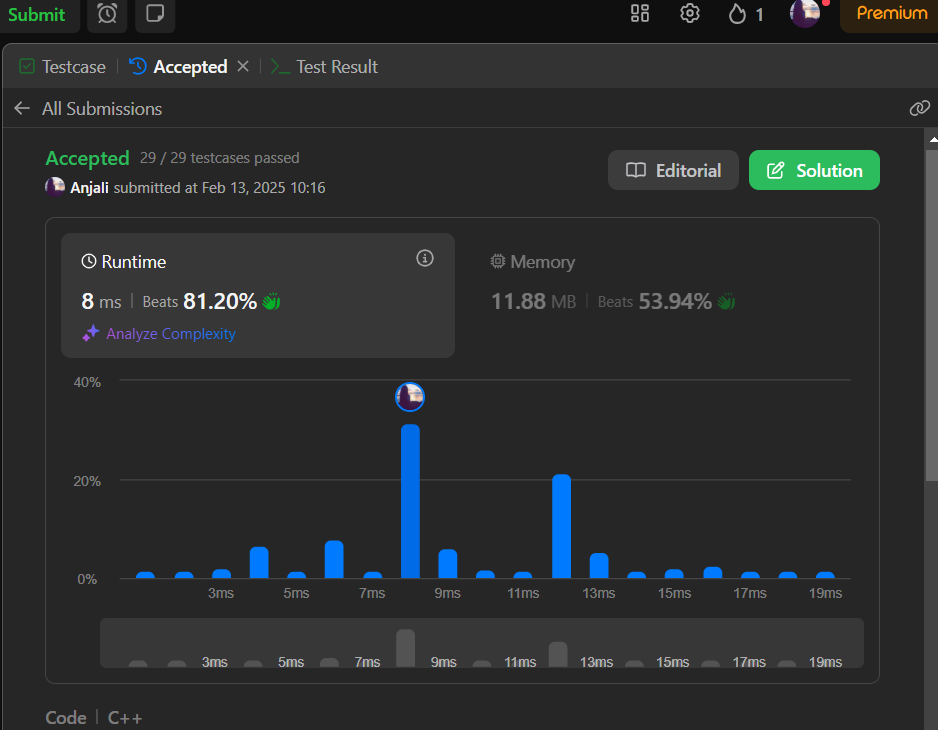
}

return false;

}

};

**Output SS:**



**Problem-8 : Print Linked List**  
**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\* current = prev->next;

ListNode\* next = nullptr;

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

next = current->next;

current->next = next->next;

next->next = prev->next;

prev->next = next;

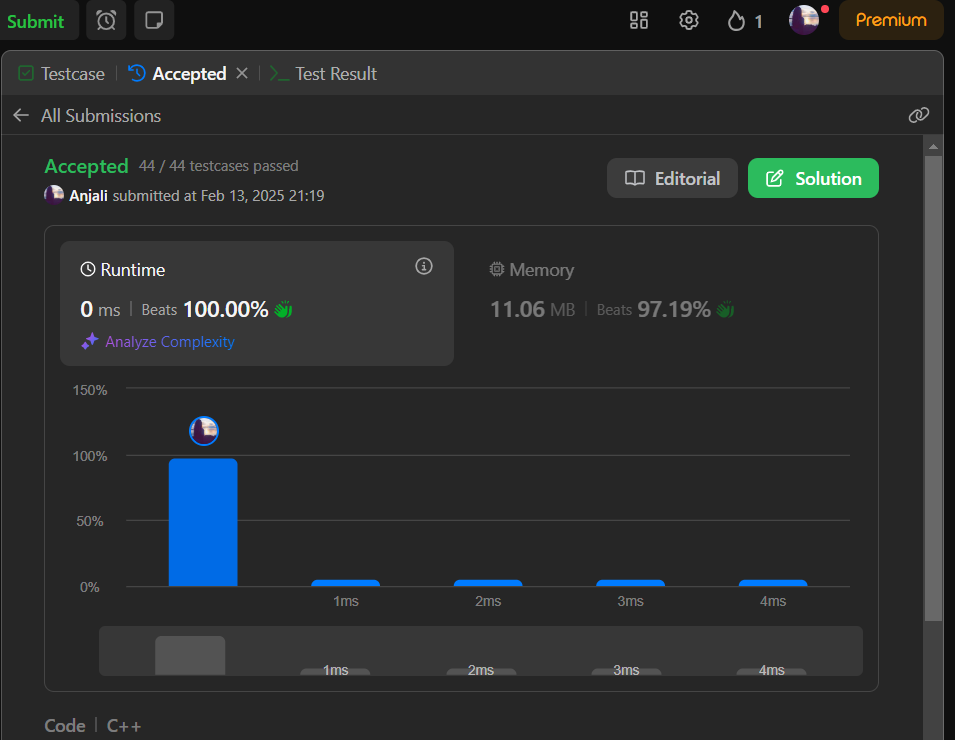
}

return dummy.next;

}

};

**Output SS:**



**Problem-9 : Rotate List**  
**Code:**

class Solution {

public:

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

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

        ListNode\* tail = head;

        int length = 1;

        while (tail->next) {

            tail = tail->next;

            length++;

        }

        k %= length;

        if (k == 0) return head;

        tail->next = head;

        int stepsToNewTail = length - k;

        ListNode\* newTail = head;

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

            newTail = newTail->next;

        }

        ListNode\* newHead = newTail->next;

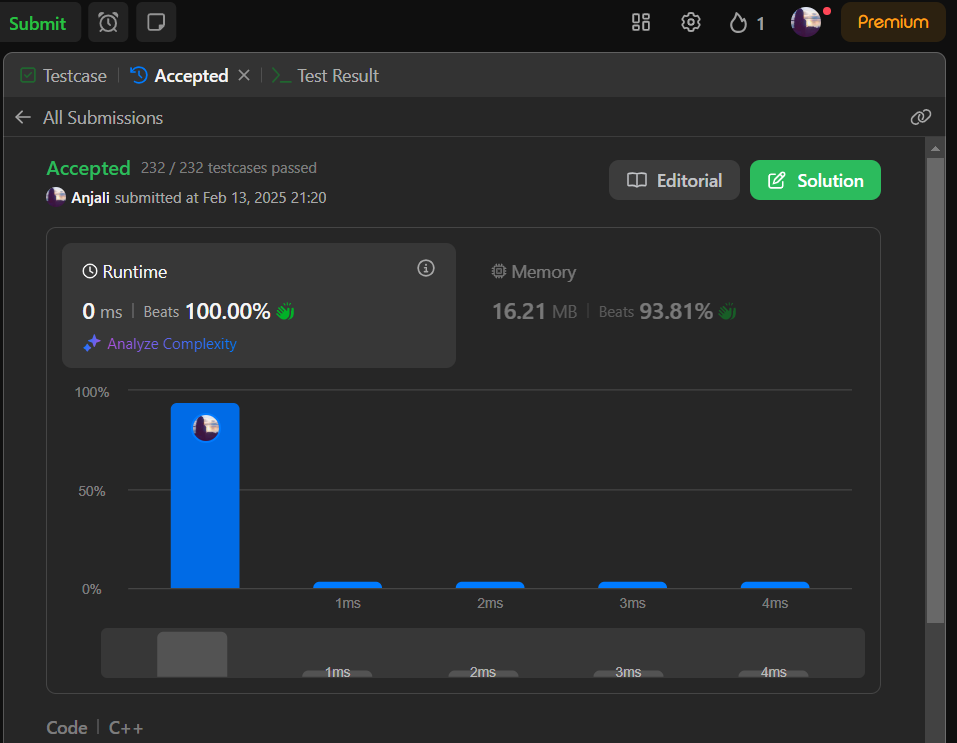
        newTail->next = nullptr;

        return newHead;

    }

};

**Output SS:**



**Problem-10 : Sort List**  
**Code:**

class Solution {

public:

    ListNode\* sortList(ListNode\* head) {

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

        ListNode\* mid = getMiddle(head);

        ListNode\* left = head;

        ListNode\* right = mid->next;

        mid->next = nullptr;

        left = sortList(left);

        right = sortList(right);

        return merge(left, right);

    }

private:

    ListNode\* getMiddle(ListNode\* head) {

        ListNode\* slow = head;

        ListNode\* fast = head->next;

        while (fast && fast->next) {

            slow = slow->next;

            fast = fast->next->next;

        }

        return slow;

    }

    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;

        }

        if (l1) tail->next = l1;

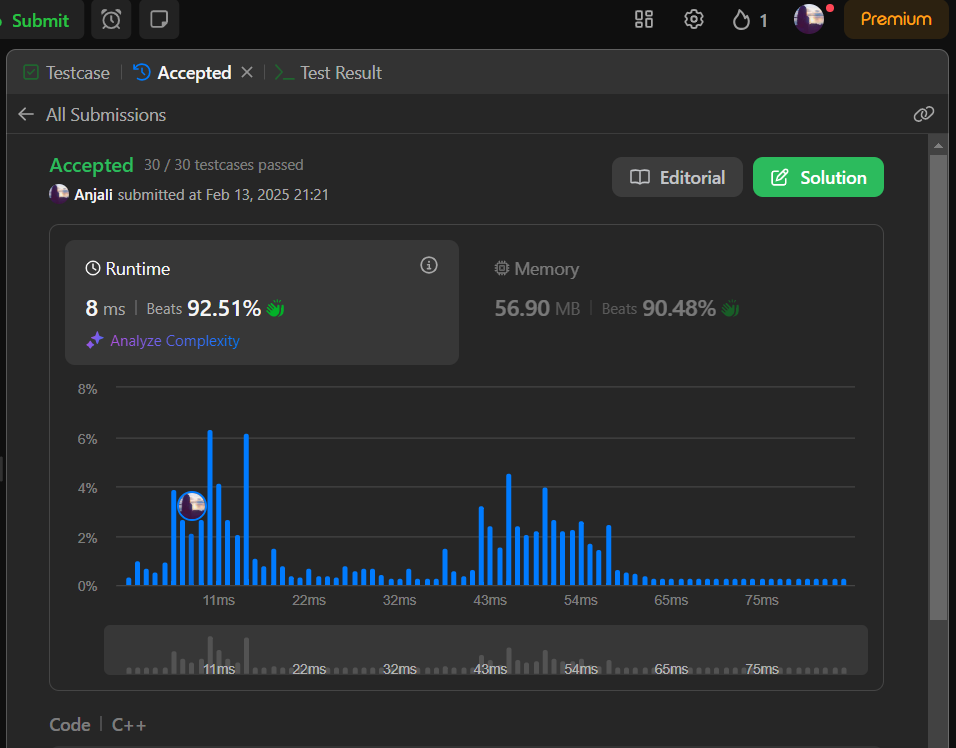
        if (l2) tail->next = l2;

        return dummy.next;

    }

};

**Output SS:**



**Problem-11 : Linked List Cycle II**  
**Code:**

class Solution {

public:

ListNode \*detectCycle(ListNode \*head) {

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

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

while (fast && fast->next) {

slow = slow->next;

fast = fast->next->next;

if (slow == fast) {

ListNode \*entry = head;

while (entry != slow) {

entry = entry->next;

slow = slow->next;

}

return entry;

}

}

return nullptr;

}

};

**Output SS:**

