Ap-Assignment 3

**Name:** Harmandeep Singh **UID:**22BCS14975 **Leetcode Profile:** <https://leetcode.com/u/Harman001/>

**Q1.Print Linkedlist:**

**a.Code:**

class Solution { public:

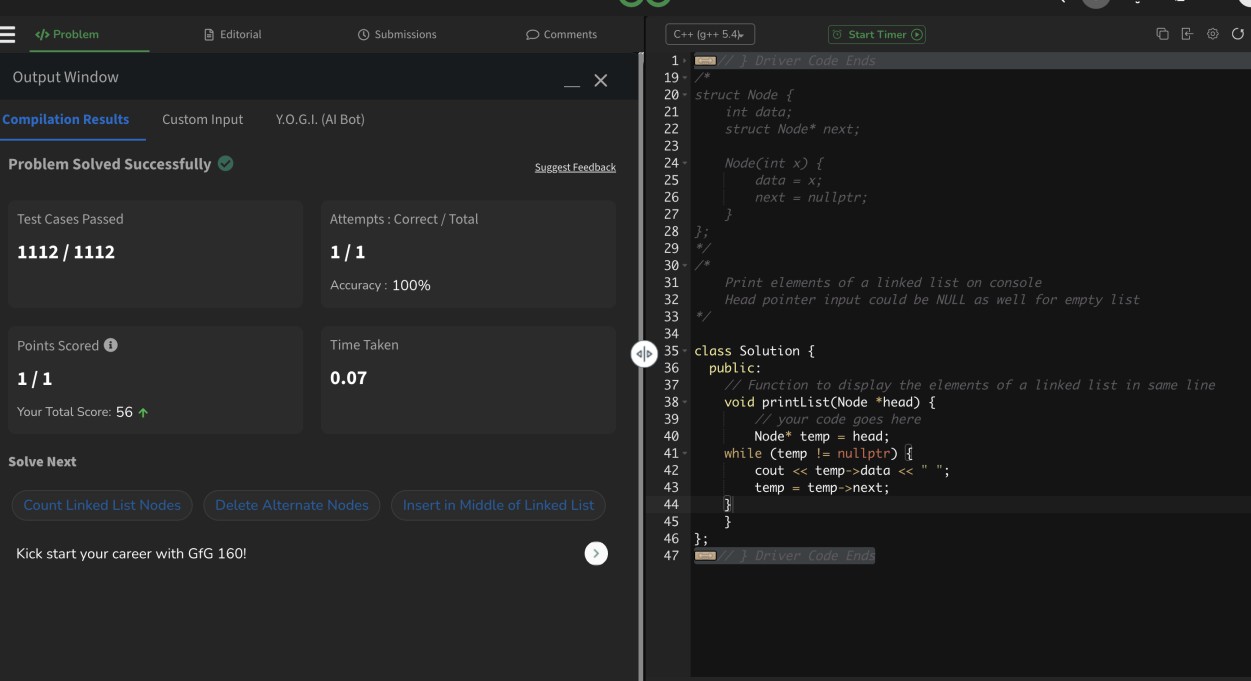
// Function to display the elements of a linked list in same line void printList(Node \*head) {

// your code goes here Node\* temp = head;

while (temp != nullptr) { cout << temp->data << " "; temp = temp->next;

}

}

};

**Q2.Remove Duplicates from Sorted list**

1. **class Solution { public:**

**ListNode\* deleteDuplicates(ListNode\* head) { ListNode\* current = head;**

**while (current != nullptr ss current->next != nullptr) { if (current->val == current->next->val) {**

**ListNode\* temp = current->next; current->next = current->next->next; delete temp;**

**} else {**

**current = current->next;**

**}**

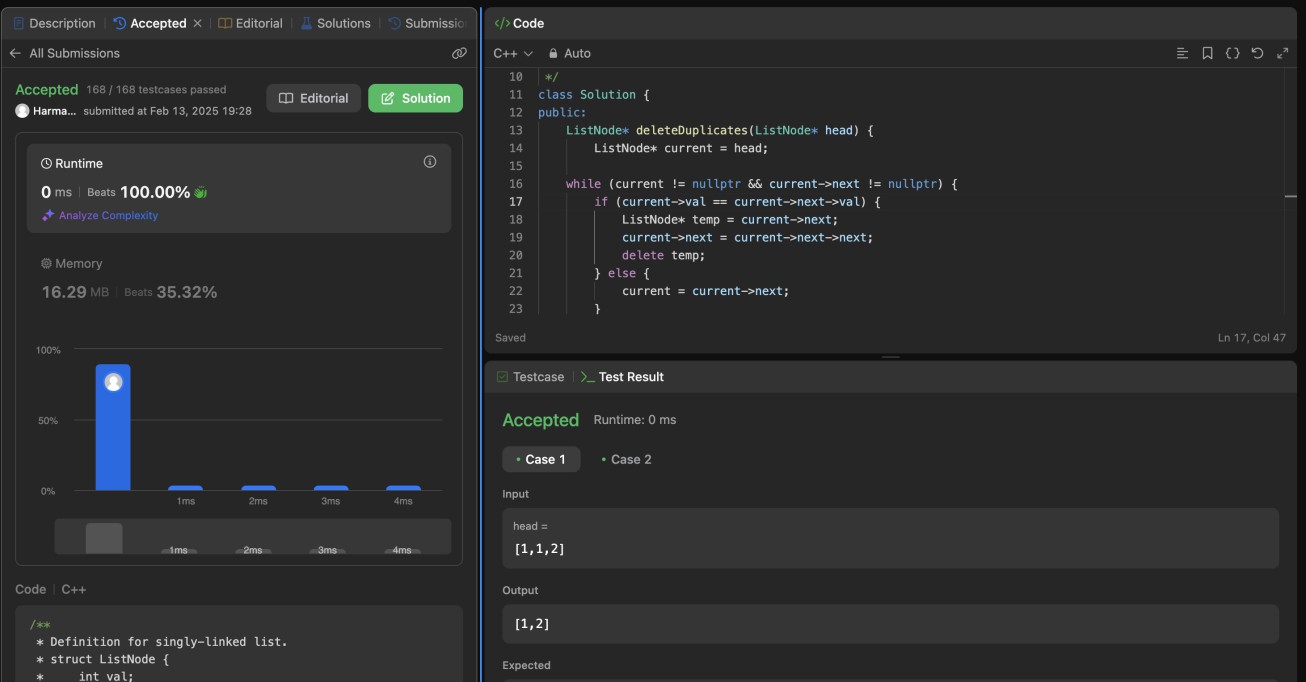
**}**

**return head;**

**}**

**};**

1. **Output**

****

**Q3.Reverse Linkedlist**

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;

}

ListNode\* createLinkedList(const vector<int>& nums) { if (nums.empty()) return nullptr;

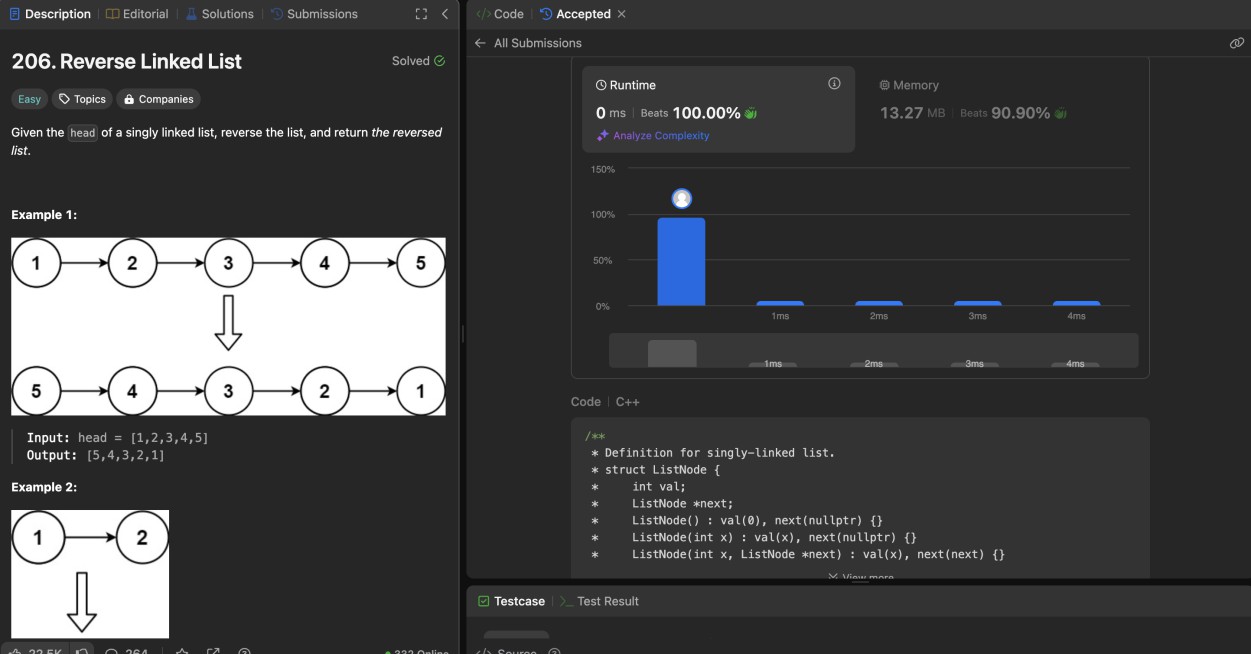
ListNode\* head = new ListNode(nums[0]); ListNode\* current = head;

for (size\_t i = 1; i < nums.size(); i++) { current->next = new ListNode(nums[i]); current = current->next;

}

return head;

}



**Q4.Delete Middle Node**

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;

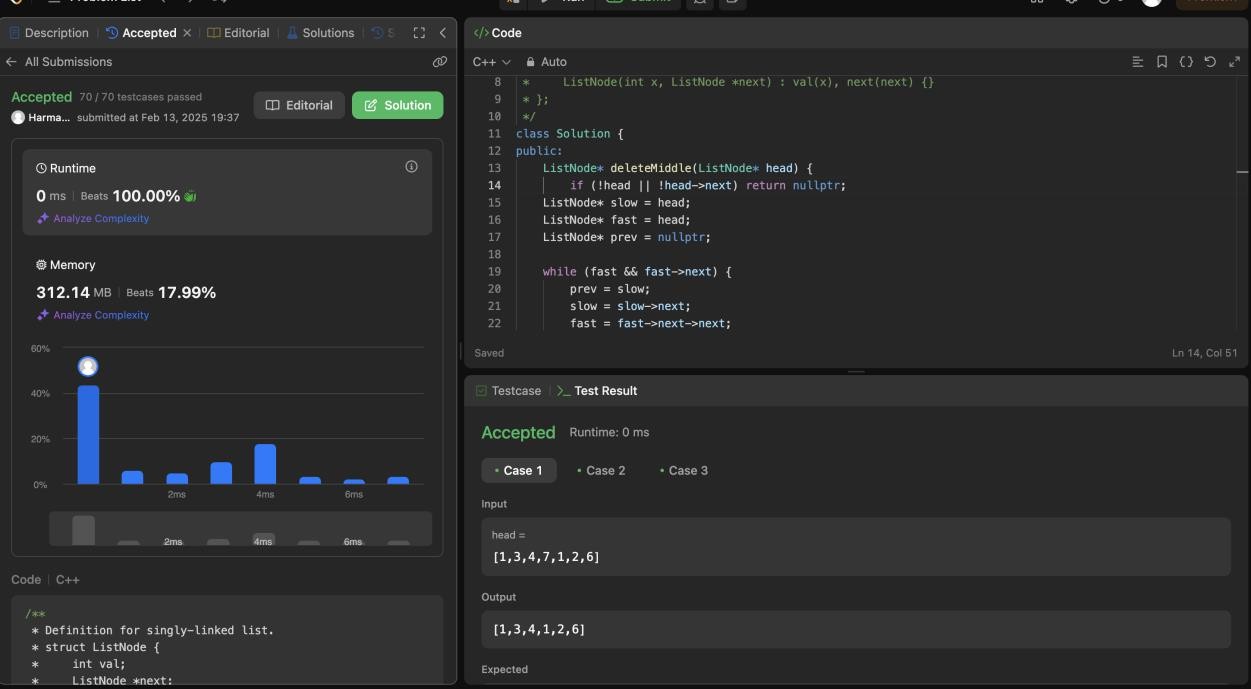
}

prev->next = slow->next; delete slow;

return head;

}

};



**Q5.Merge two Sorted linkedlist**

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;

}

tail->next = list1 ? list1 : list2;

return dummy.next;

}

void printList(ListNode\* head) { while (head) {

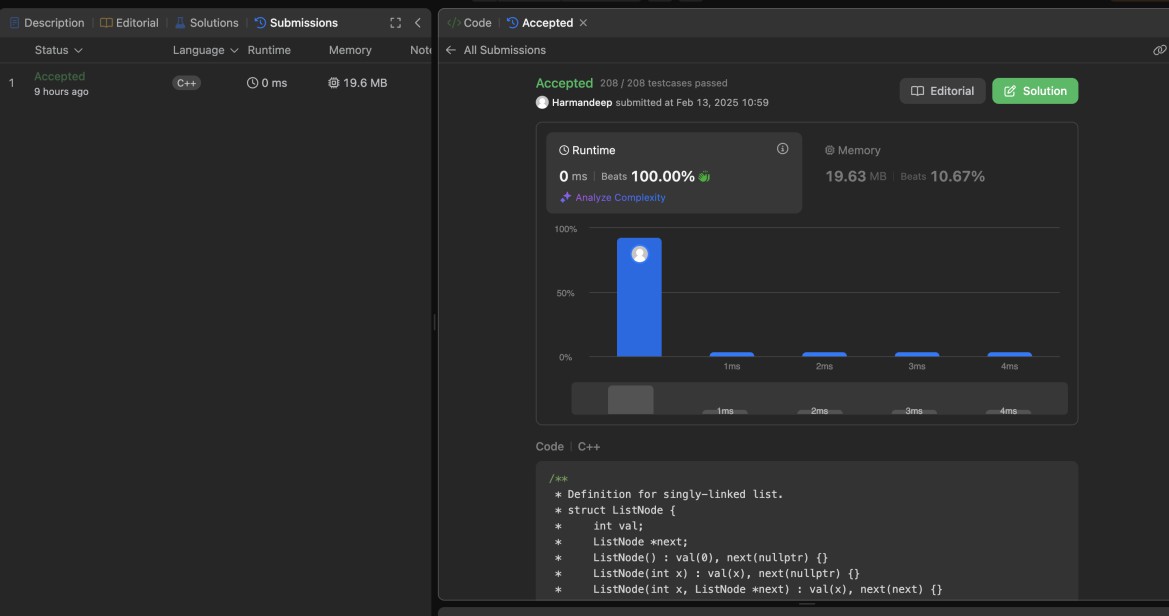
std::cout << head->val << " -> "; head = head->next;

}

std::cout << "NULL" << std::endl;

}

};



**Q6.Remove duplicates from sorted list2**

class Solution { public:

ListNode\* deleteDuplicates(ListNode\* head) { if (!head) return nullptr;

ListNode\* dummy = new ListNode(0); dummy->next = 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;

}

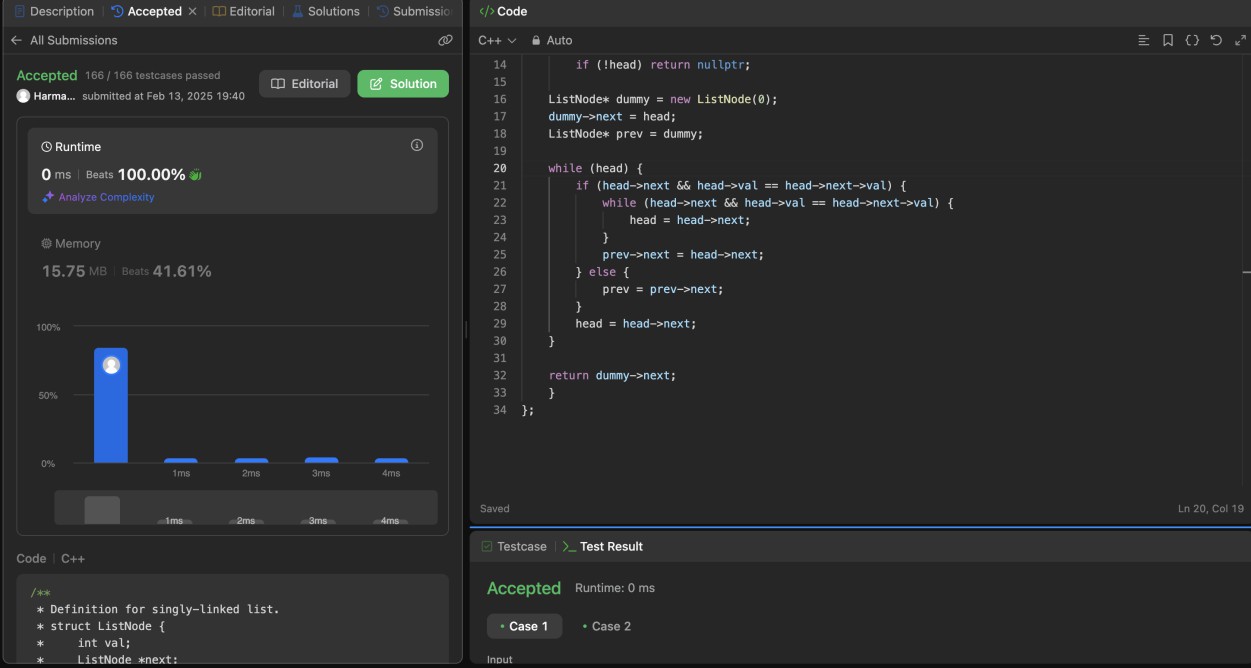
head = head->next;

}

return dummy->next;

}

};



**Q7.Detect Cycle in Linkedlist**

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;

}

ListNode\* createCycleList(const vector<int>& nums, int pos) { if (nums.empty()) return nullptr;

ListNode\* head = new ListNode(nums[0]); ListNode\* current = head;

ListNode\* cycleNode = nullptr;

for (size\_t i = 1; i < nums.size(); i++) { current->next = new ListNode(nums[i]); current = current->next;

if (static\_cast<int>(i) == pos) { cycleNode = current;

}

}

if (pos >= 0) {

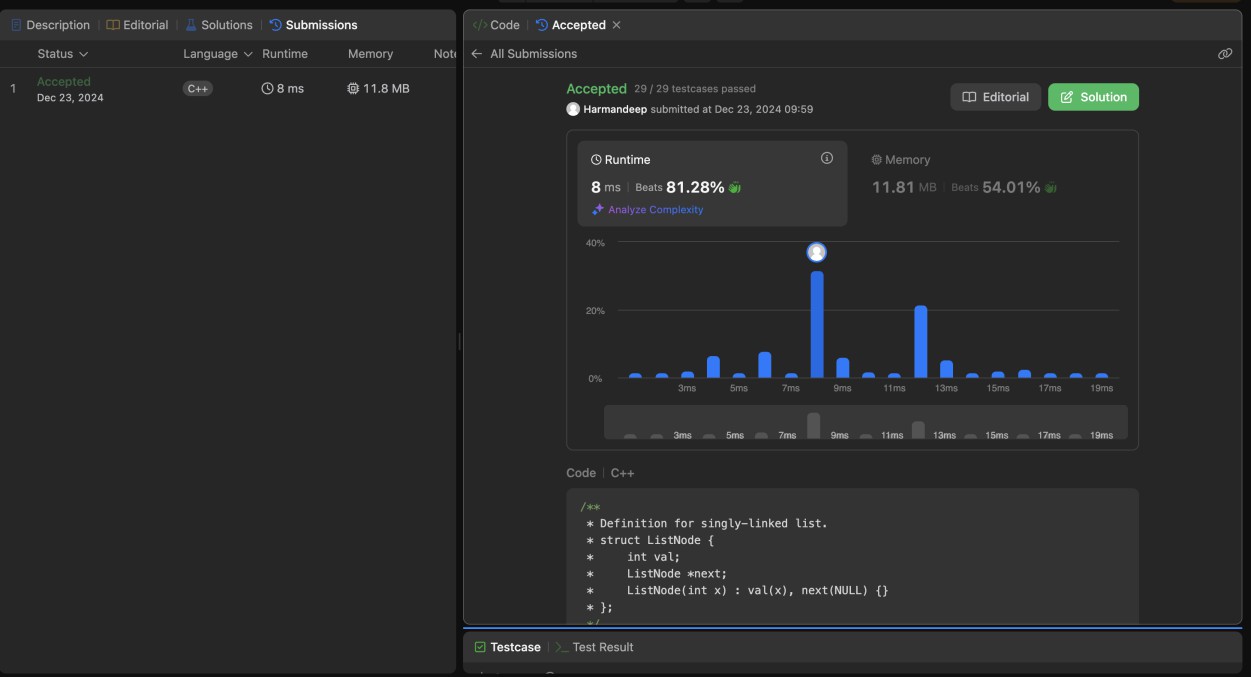
current->next = cycleNode;

}

return head;

}

};



**Q8.Reverse LinkedList 2**

class Solution { 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 = 1; i < left; i++) { prev = prev->next;

}

ListNode\* curr = prev->next; ListNode\* next = nullptr;

ListNode\* prevReversed = nullptr; for (int i = 0; i <= right - left; i++) {

next = curr->next;

curr->next = prevReversed; prevReversed = curr;

curr = next;

}

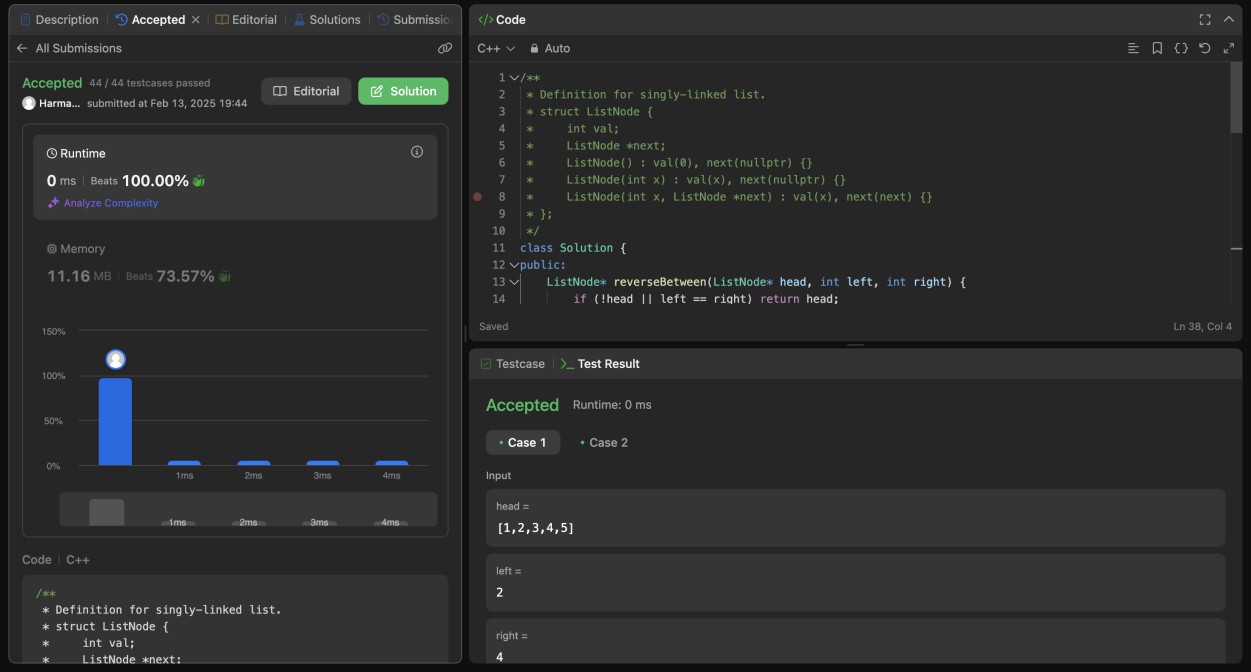
prev->next->next = curr;

prev->next = prevReversed;

return dummy->next;

}

};



**QG.Rotate a list**

class Solution { public:

ListNode\* rotateRight(ListNode\* head, int k) { if (!head || !head->next || k == 0) return head;

int n = 1;

ListNode\* tail = head; while (tail->next) {

tail = tail->next; n++;

}

k = k % n;

if (k == 0) return head;

ListNode\* newTail = head;

for (int i = 0; i < n - k - 1; i++) { newTail = newTail->next;

}

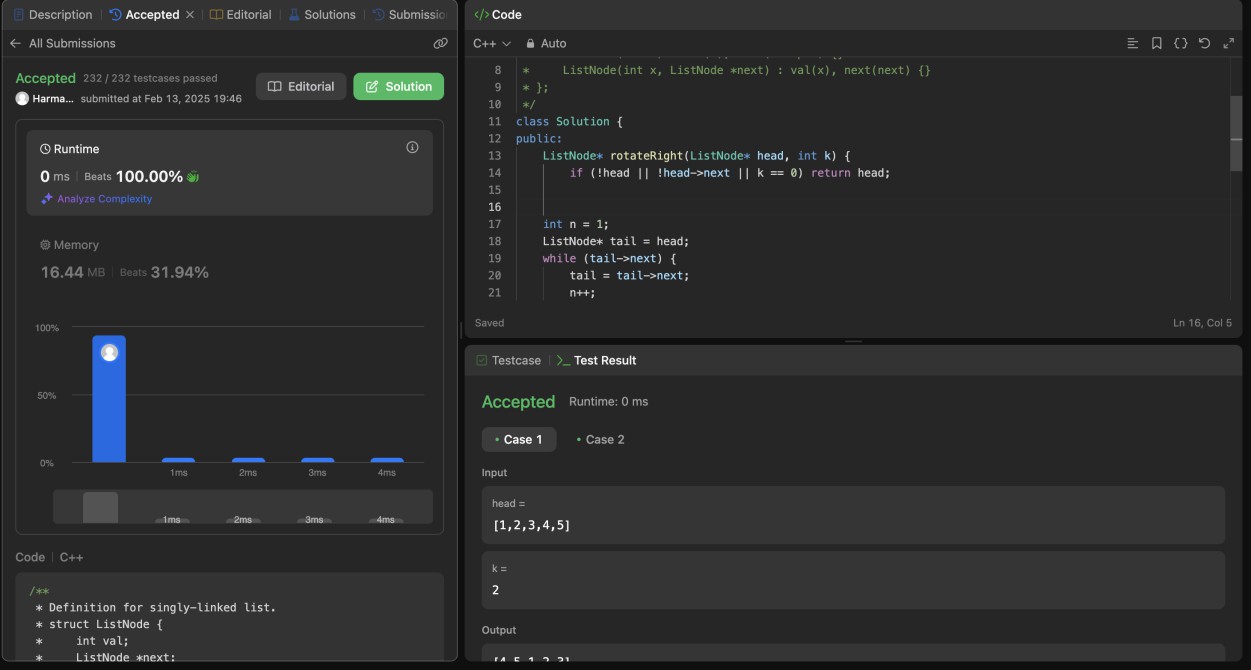
ListNode\* newHead = newTail->next; newTail->next = nullptr;

tail->next = head;

return newHead;

}

};



**Q10.Detect a cycle in Linkedlist 2**

class Solution { public:

ListNode \*detectCycle(ListNode \*head) { if (!head || !head->next) return nullptr;

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;

slow = head;

while (slow != fast) { slow = slow->next; fast = fast->next;

}

return slow;

}

void createCycle(ListNode\* head, int pos) { if (pos == -1) return;

ListNode\* cycleNode = head;

ListNode\* tail = head;

int index = 0;

while (tail->next) {

if (index == pos) cycleNode = tail; tail = tail->next;

index++;

}

tail->next = cycleNode;

}

};

