**Experiment-3**

**Student Name: Aditya Soni UID: 22BET10130**

**Branch: BE-IT Section/Group: 22BET-IOT-702-B**

**Semester: 6th Date of Performance: 30/01/2025**

**Subject Name: AP Lab II Subject Code: 22ITT-314 Problem-1**

1. **Aim:** Detect a cycle in a linked list.
2. **Objective:** Given head, the head of a linked list, determine if the linked list has a cycle in it. There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer. Internally, pos is used to denote the index of the node that tail's next pointer is connected to.

# Implementation/Code:

class Solution { public:

bool hasCycle(ListNode\* head) { ListNode\* slow = head; ListNode\* fast = head;

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

fast = fast->next->next; if (slow == fast)

return true;

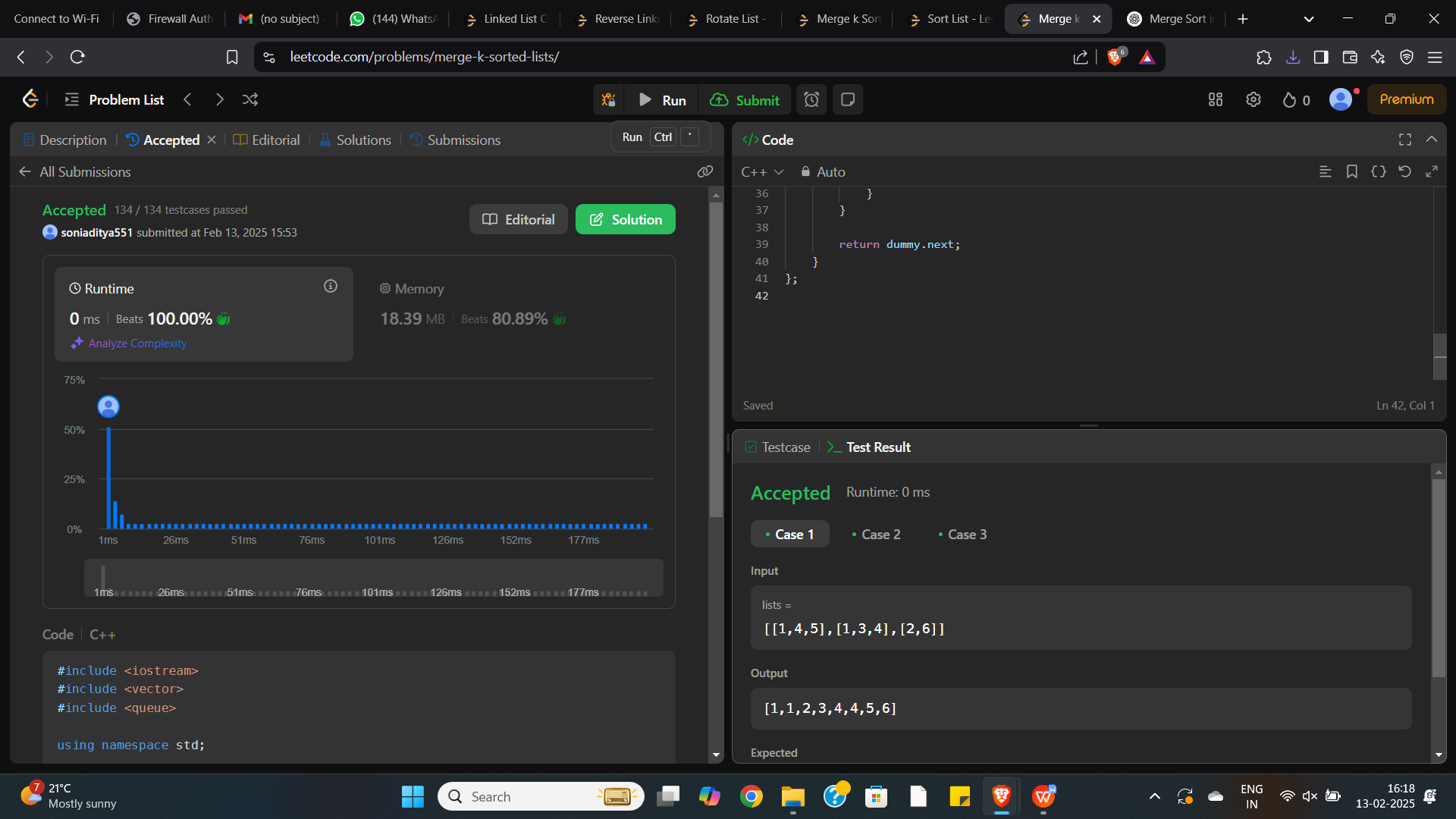
}

return false;

}

};

# Output:

****

**Problem-2**

* 1. **Aim:** Reverse linked list 2
  2. **Objective**: Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return the reversed list.

# Implementation/Code:

class Solution { public:

ListNode\* reverseBetween(ListNode\* head, int left, int right) { if (left == 1)

return reverseN(head, right);

head->next = reverseBetween(head->next, left - 1, right - 1); return head;

}

private:

ListNode\* reverseN(ListNode\* head, int n) { if (n == 1)

return head;

ListNode\* newHead = reverseN(head->next, n - 1); ListNode\* headNext = head->next;

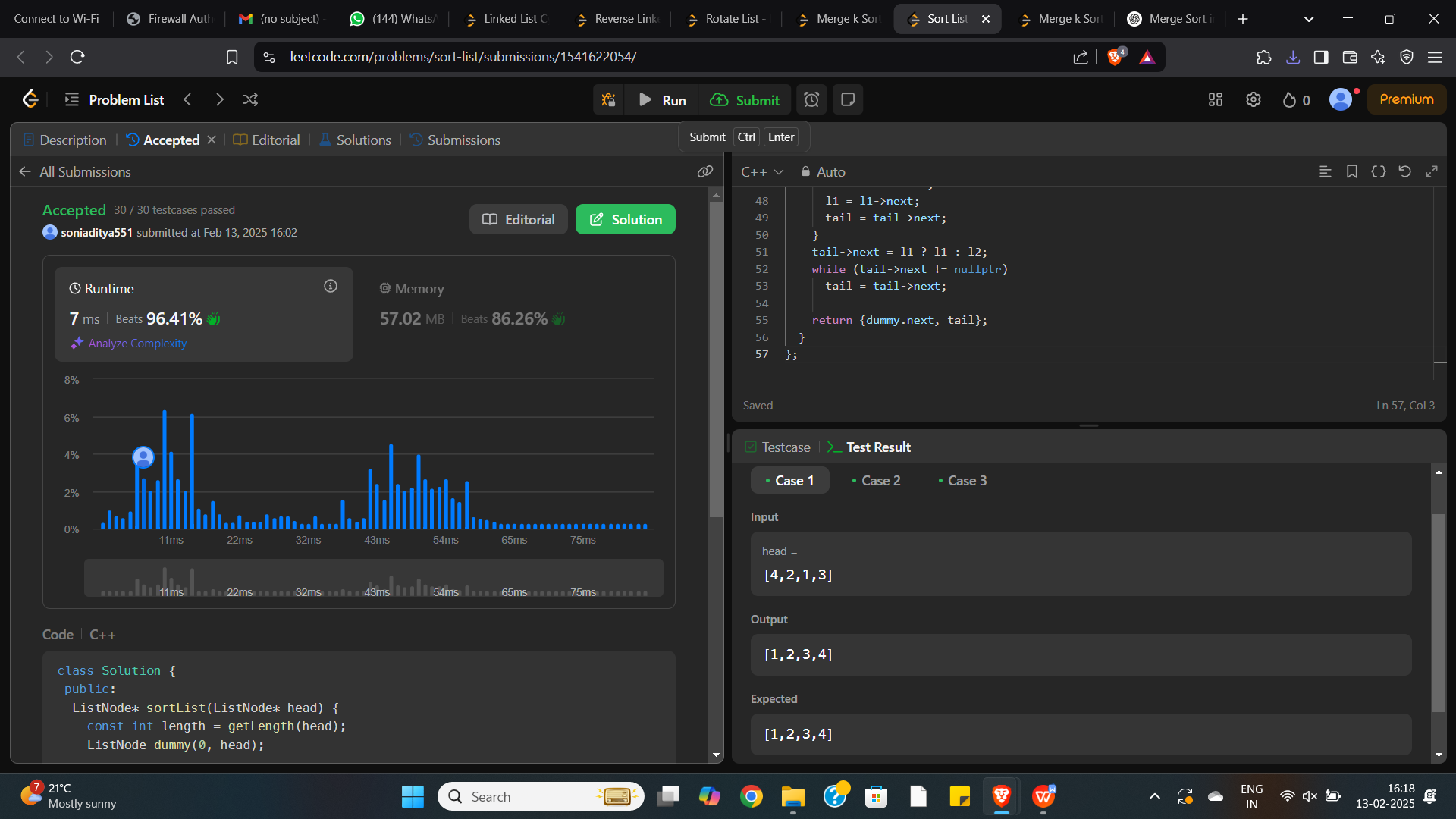
head->next = headNext->next; headNext->next = head;

return newHead;

}

};

# Output:

****

**Problem-3**

1. **Aim:** Rotate a list
2. **Objective:** Given the head of a linked list, rotate the list to the right by k places.

# Code:

class Solution { public:

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

return head;

ListNode\* tail; int length = 1;

for (tail = head; tail->next; tail = tail->next)

++length;

tail->next = head; // Circle the list.

const int t = length - k % length; for (int i = 0; i < t; ++i)

tail = tail->next;

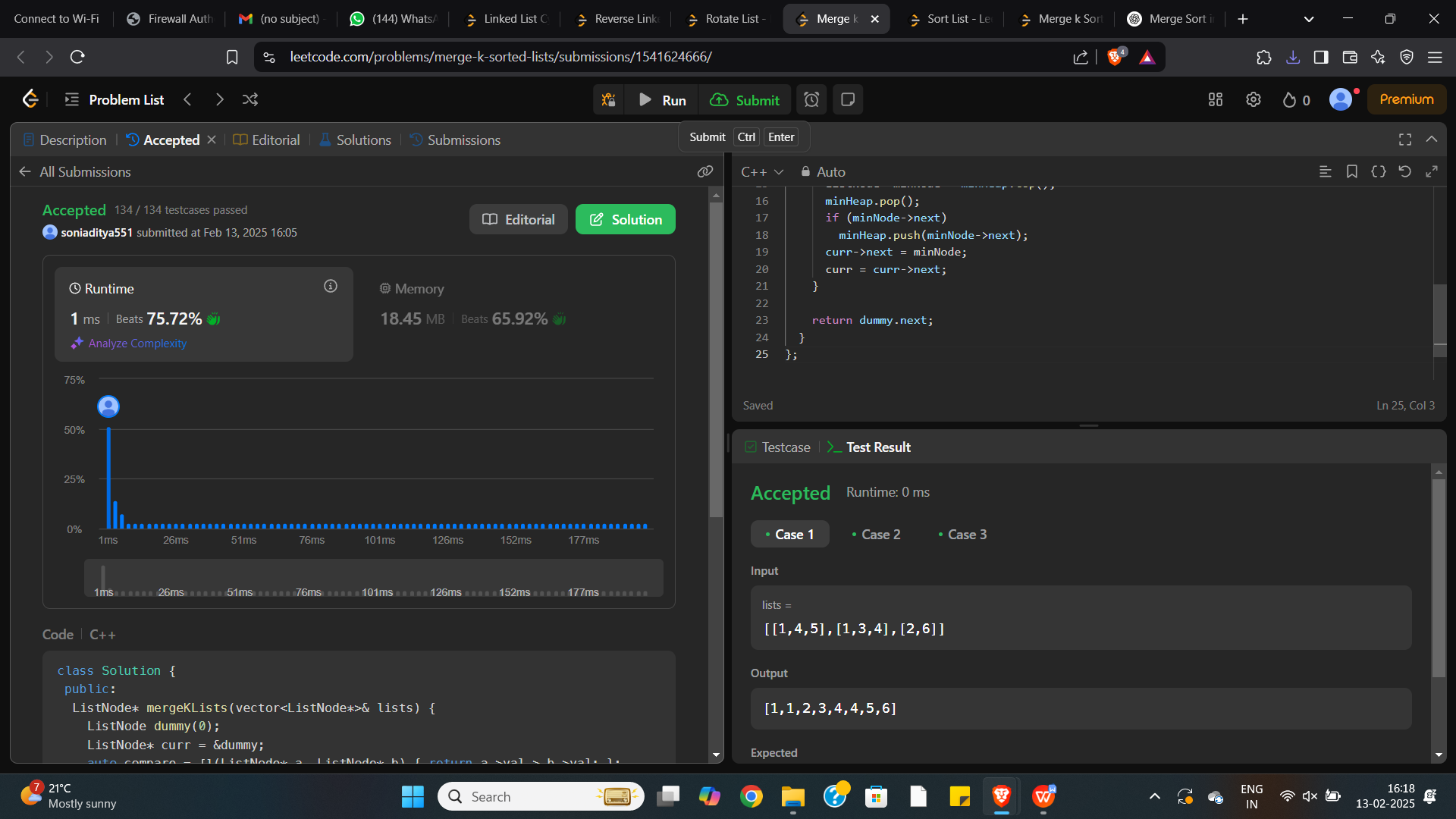
ListNode\* newHead = tail->next; tail->next = nullptr;

return newHead;

}

};

# Output:

****

**Problem-4**

* 1. **Aim:** Merge k sorted lists
  2. **Objective**: You are given an array of k linked-lists lists, each linked-list is sorted in ascending order. Merge all the linked-lists into one sorted linked- list and return it.

# Implementation/Code:

class Solution { public:

ListNode\* mergeKLists(vector<ListNode\*>& lists) { ListNode dummy(0);

ListNode\* curr = &dummy;

auto compare = [](ListNode\* a, ListNode\* b) { return a->val > b->val; }; priority\_queue<ListNode\*, vector<ListNode\*>, decltype(compare)> minHeap(

compare);

for (ListNode\* list : lists) if (list != nullptr) minHeap.push(list);

while (!minHeap.empty()) {

ListNode\* minNode = minHeap.top(); minHeap.pop();

if (minNode->next) minHeap.push(minNode->next); curr->next = minNode;

curr = curr->next;

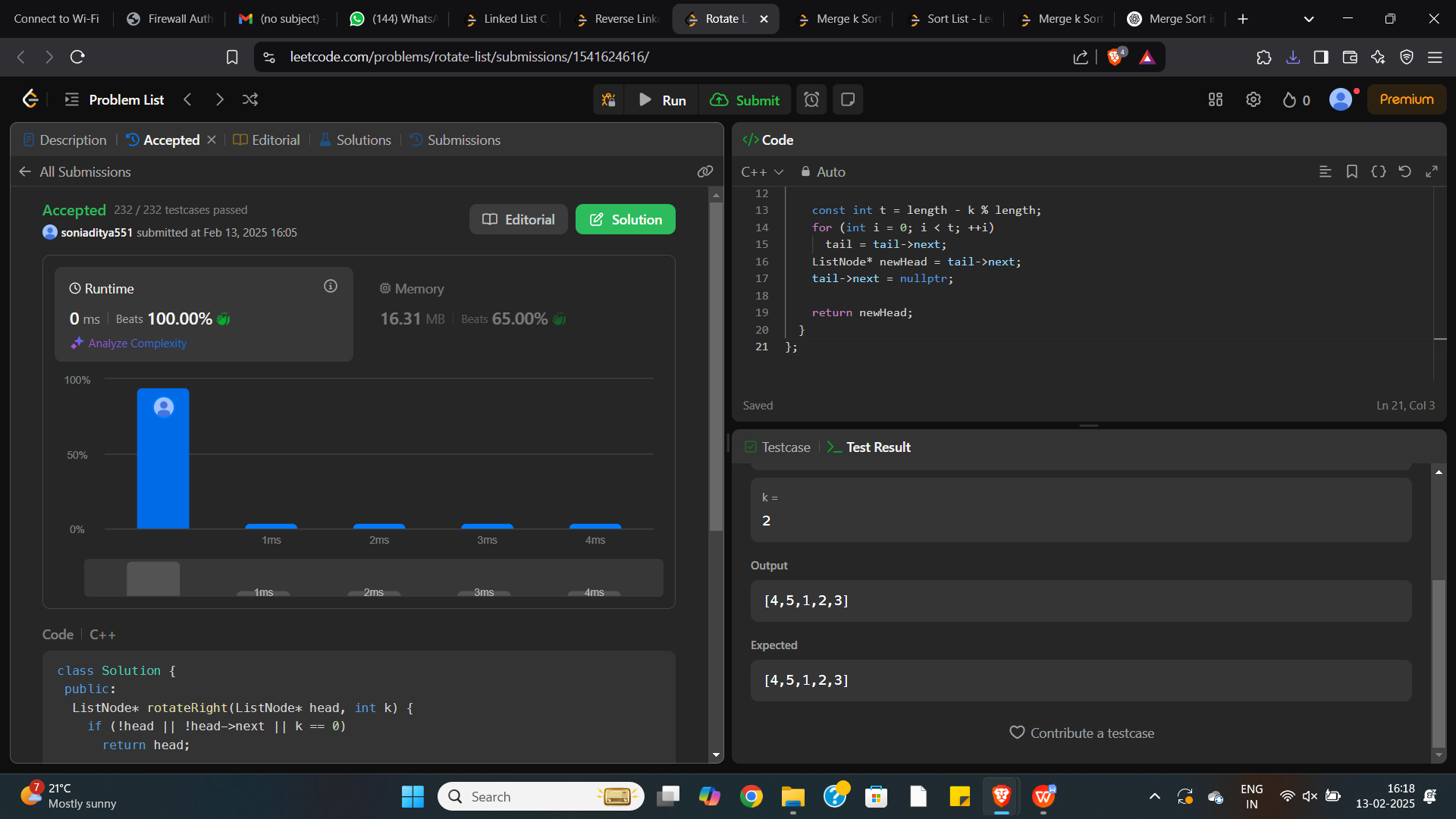
}

return dummy.next;

}

};

# Output:

****

**Problem-5**

1. **Aim:** Sort List
2. **Objective**: Given the head of a linked list, return the list after sorting it in ascending order.

# Implementation/Code:

class Solution { public:

ListNode\* sortList(ListNode\* head) { const int length = getLength(head); ListNode dummy(0, head);

for (int k = 1; k < length; k \*= 2) { ListNode\* curr = dummy.next; ListNode\* tail = &dummy; while (curr != nullptr) { ListNode\* l = curr;

ListNode\* r = split(l, k); curr = split(r, k);

auto [mergedHead, mergedTail] = merge(l, r); tail->next = mergedHead;

tail = mergedTail;

}

}

return dummy.next;

}

private:

int getLength(ListNode\* head) { int length = 0;

for (ListNode\* curr = head; curr; curr = curr->next)

++length; return length;

}

ListNode\* split(ListNode\* head, int k) { while (--k && head)

head = head->next;

ListNode\* rest = head ? head->next : nullptr; if (head != nullptr)

head->next = nullptr; return rest;

}

pair<ListNode\*, ListNode\*> merge(ListNode\* l1, ListNode\* l2) { ListNode dummy(0);

ListNode\* tail = &dummy;

while (l1 && l2) {

if (l1->val > l2->val) swap(l1, l2);

tail->next = l1; l1 = l1->next; tail = tail->next;

}

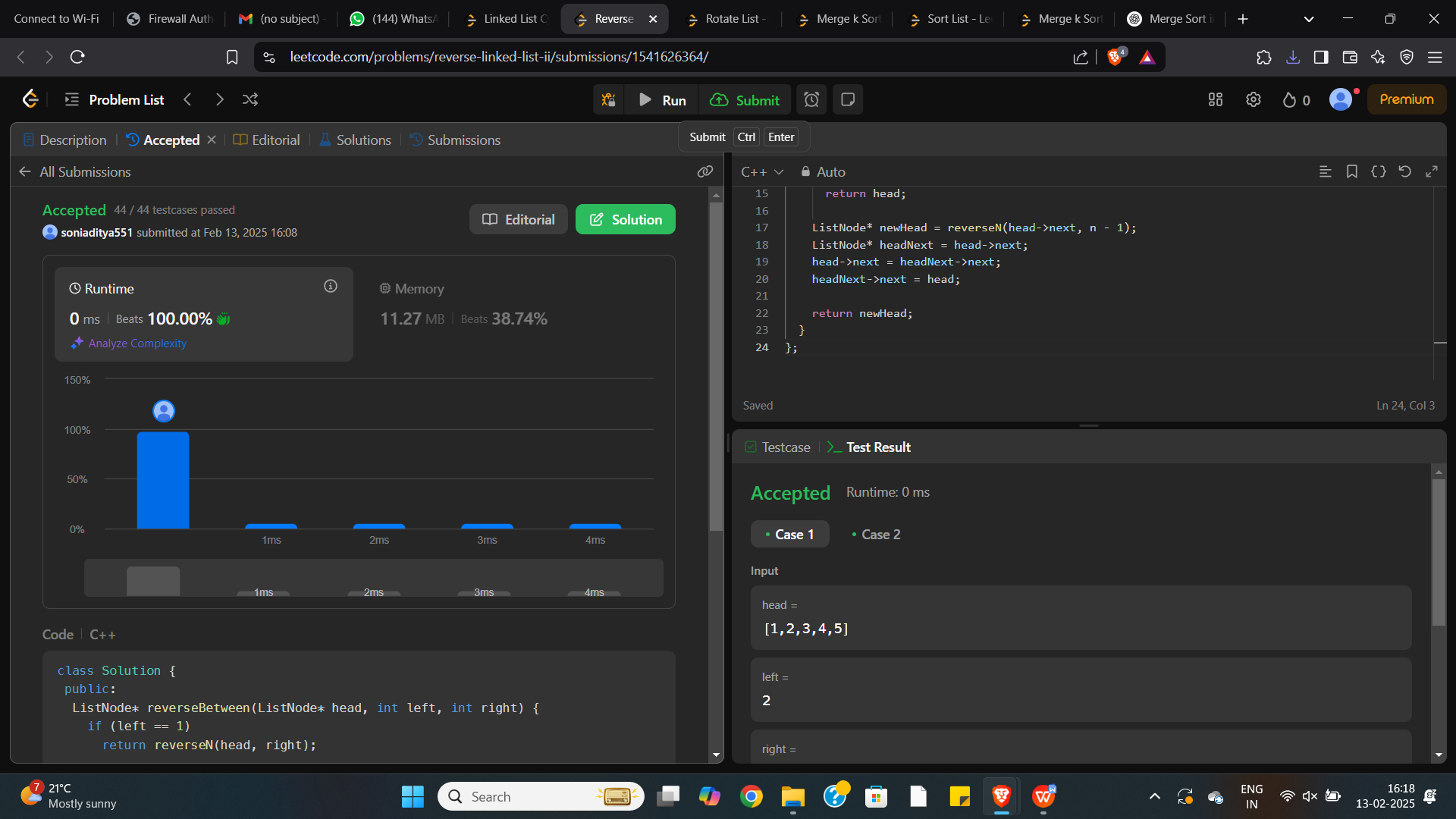
tail->next = l1 ? l1 : l2; while (tail->next != nullptr) tail = tail->next;

return {dummy.next, tail};

}

};

1. **Output:**

****