AP Assignment

Student Name: Shreya Singh **UID:** 22BCS12423
Branch: BE-CSE **Section/Group:** 606-B
Semester: 6th **Date :** 06/03/25
Subject Name: AP Lab-2 **Subject Code:** 22CSP-351

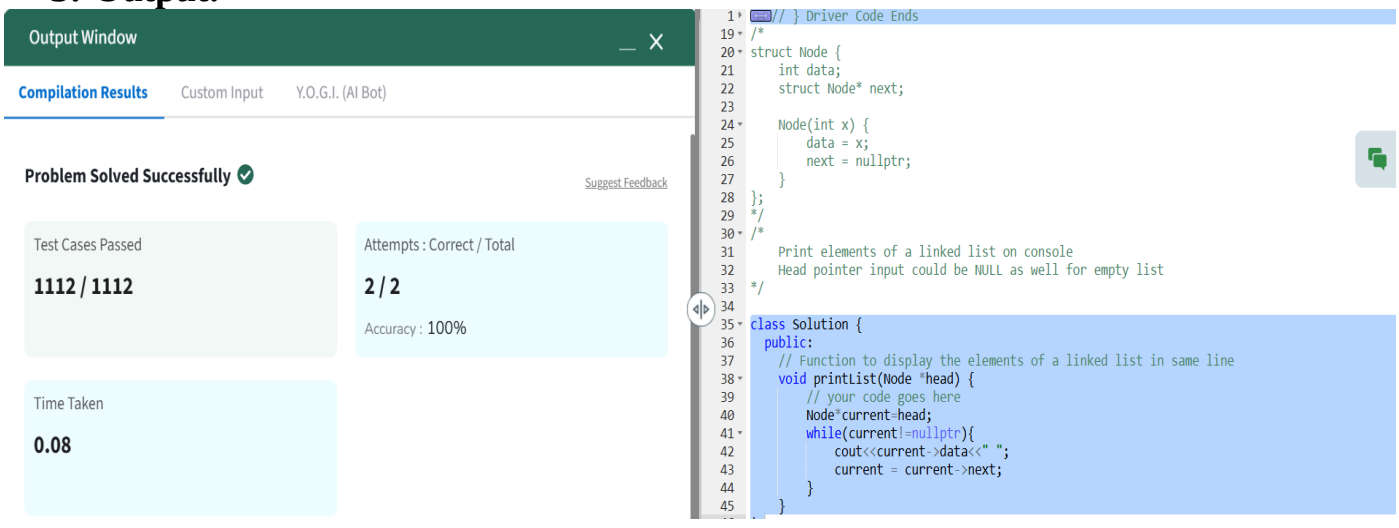
Problem 1

1. Aim: Given a linked list. Print all the elements of the linked list separated by space followed.

2. Implementation/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*current=head;
        while(current!=nullptr){
            cout<<current->data<<" ";
            current = current->next;
        }
    }}
```

3. Output:



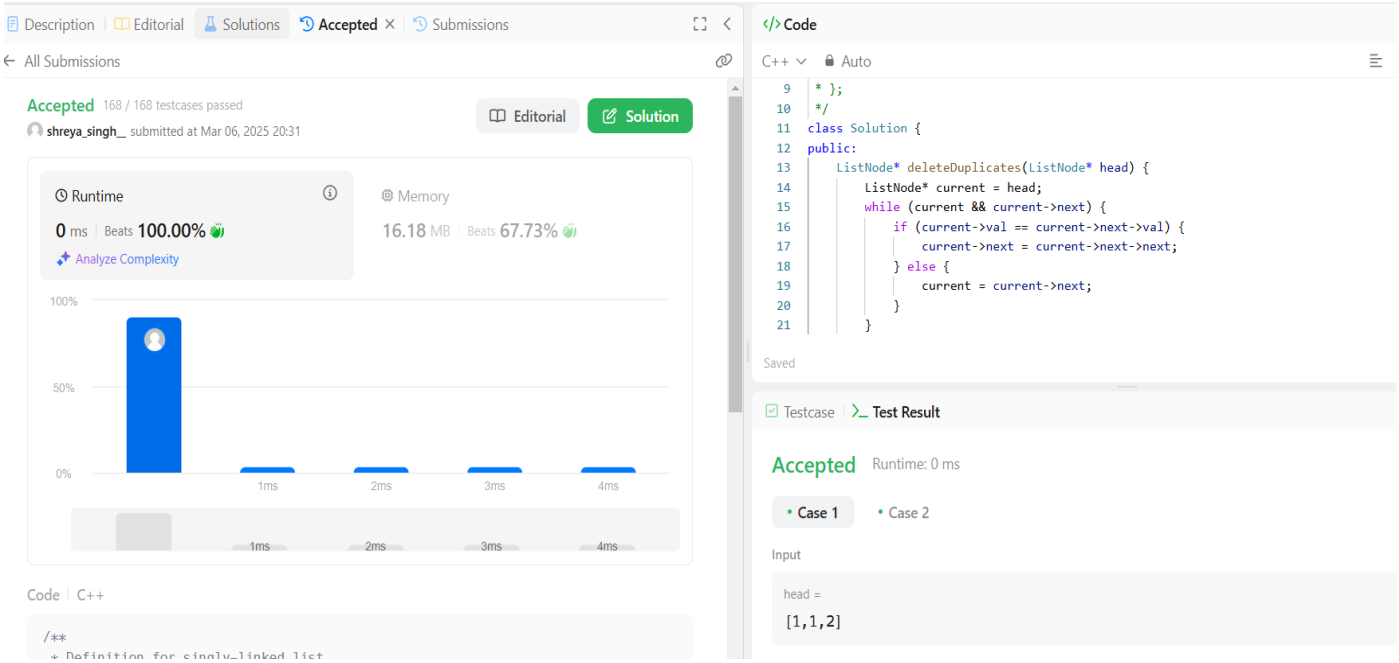
Problem 2

1. Aim: Remove duplicates from a sorted list

2. Implementation/Code:

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

3. Output:



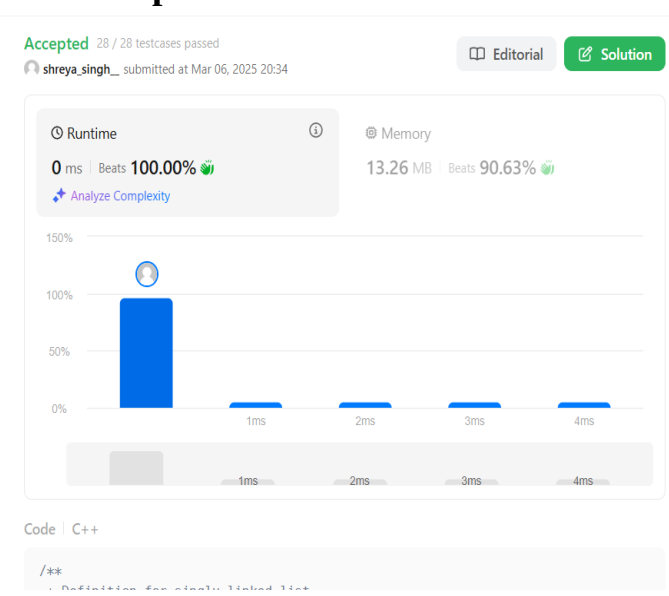
Problem 3

1. **Aim:** Reverse a linked list.

2. **Implementation/Code:**

```
class Solution {
public:
   ListNode* reverseList(ListNode* head) {
      ListNode *nextNode, *prevNode = NULL;
      while (head) {
         nextNode = head->next;
         head->next = prevNode;
         prevNode = head;
         head = nextNode;
      }
      return prevNode;
   }
};
```

3. **Output:**



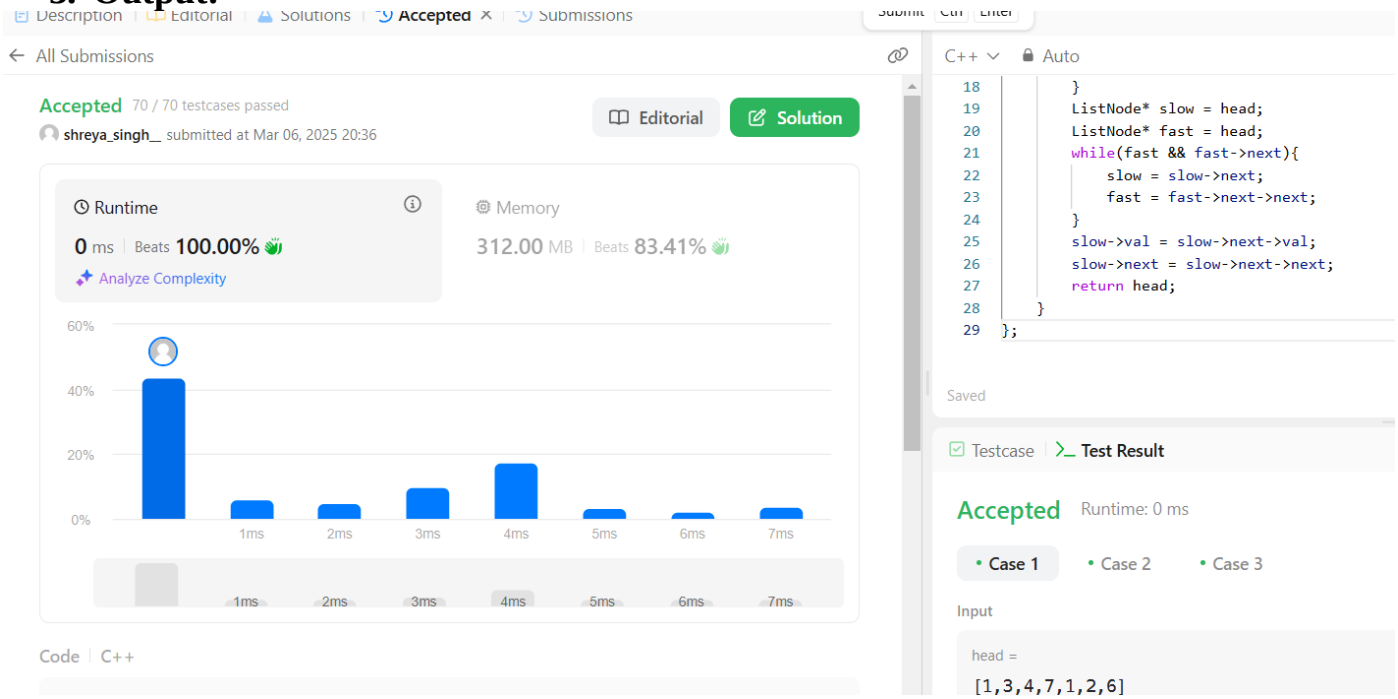
Problem 4

1. **Aim:** Delete middle node of a list

2. **Implementation/Code:**

```
class Solution {
public:
   ListNode* deleteMiddle(ListNode* head) {
      if(!head->next) return NULL;
      if(!head->next->next){
         head->next = NULL;
         return head;
      }
      ListNode* slow = head;
      ListNode* fast = head;
      while(fast && fast->next){
         slow = slow->next;
         fast = fast->next->next;
      }
      slow->val = slow->next->val;
      slow->next = slow->next->next;
      return head;
   }
};
```

3. **Output:**



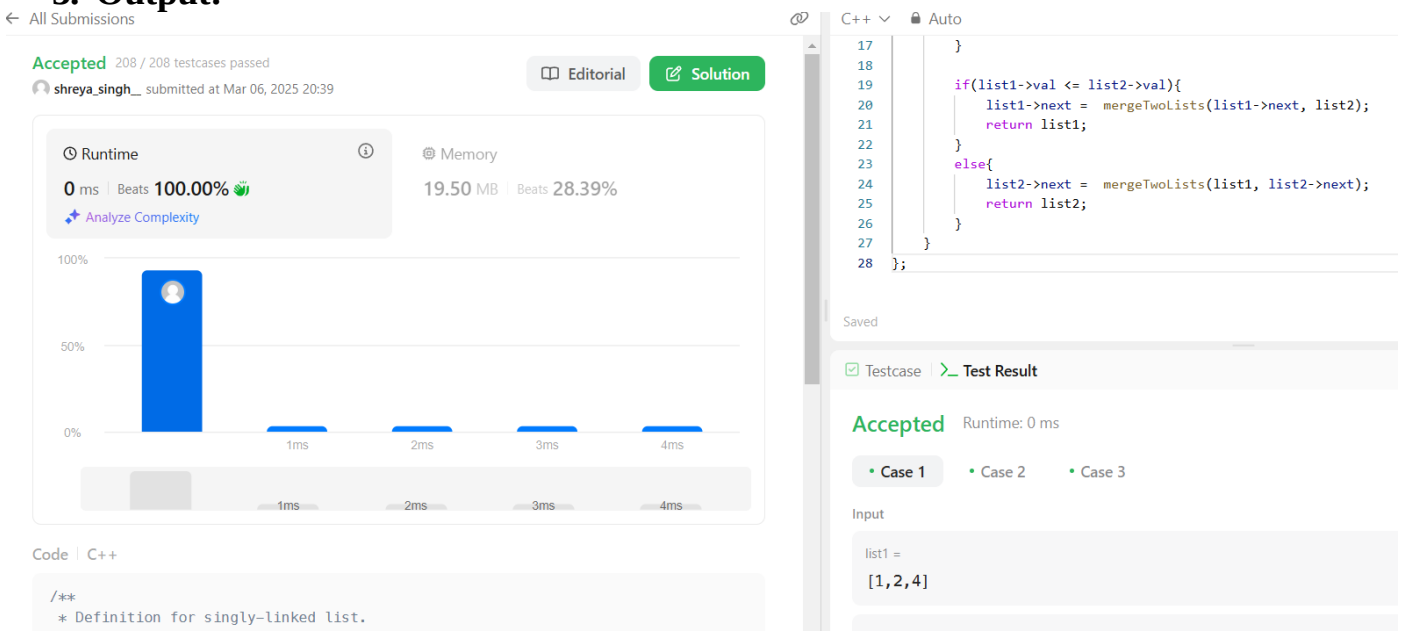
Problem 5

1. **Aim:** Merge two sorted linked lists

2. **Implementation/Code:**

```
class Solution {
public:
   ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
      if(list1 == NULL || list2 == NULL){
         return list1 == NULL ? list2 : list1;
      }

      if(list1->val <= list2->val){
         list1->next =  mergeTwoLists(list1->next, list2);
         return list1;
      }
      else{
         list2->next =  mergeTwoLists(list1, list2->next);
         return list2;
      }
   }
};
```

3. **Output:**



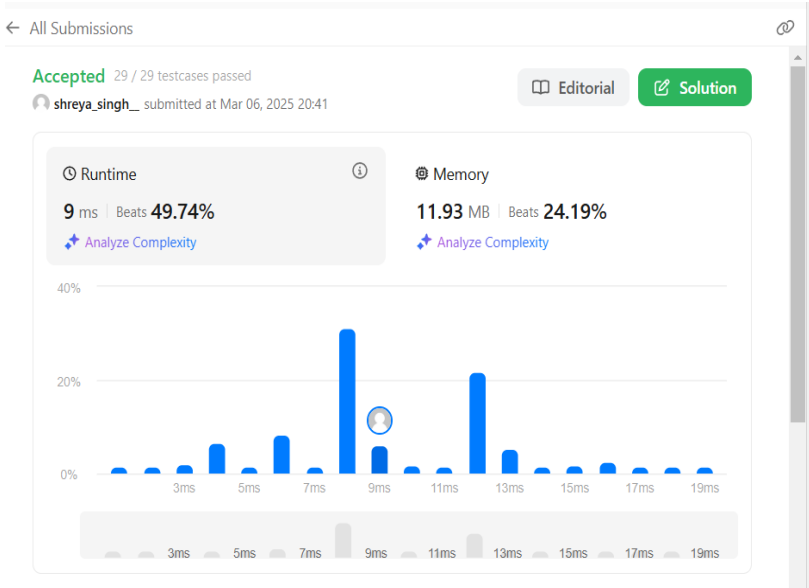
Problem 6

1. **Aim:** Detect a cycle in a linked list.

2. **Implementation/Code:**

```
class Solution {
public:
   bool hasCycle(ListNode* head) {
      if (head == NULL || head->next == NULL) {
         return false;
      }
      ListNode* slow = head;
      ListNode* fast = head->next;
      while (fast != slow) {
         if (fast->next == NULL || fast->next->next == NULL) {
            return false;
         }
         slow = slow->next;
         fast = fast->next->next;
      }
      return true;
   }
};
```

3. **Output:**



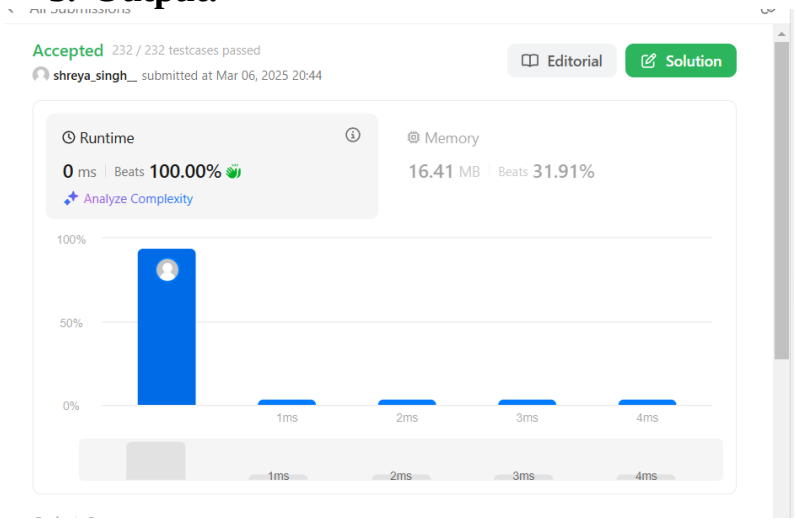
Problem 7

1. **Aim:** Rotate a list.

2. **Implementation/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 stepsToNewHead = length - k;
        ListNode* newTail = tail;

        while (stepsToNewHead--) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next;
        newTail->next = nullptr;
        return newHead;
    }
};
```

3. **Output:**



Problem 8

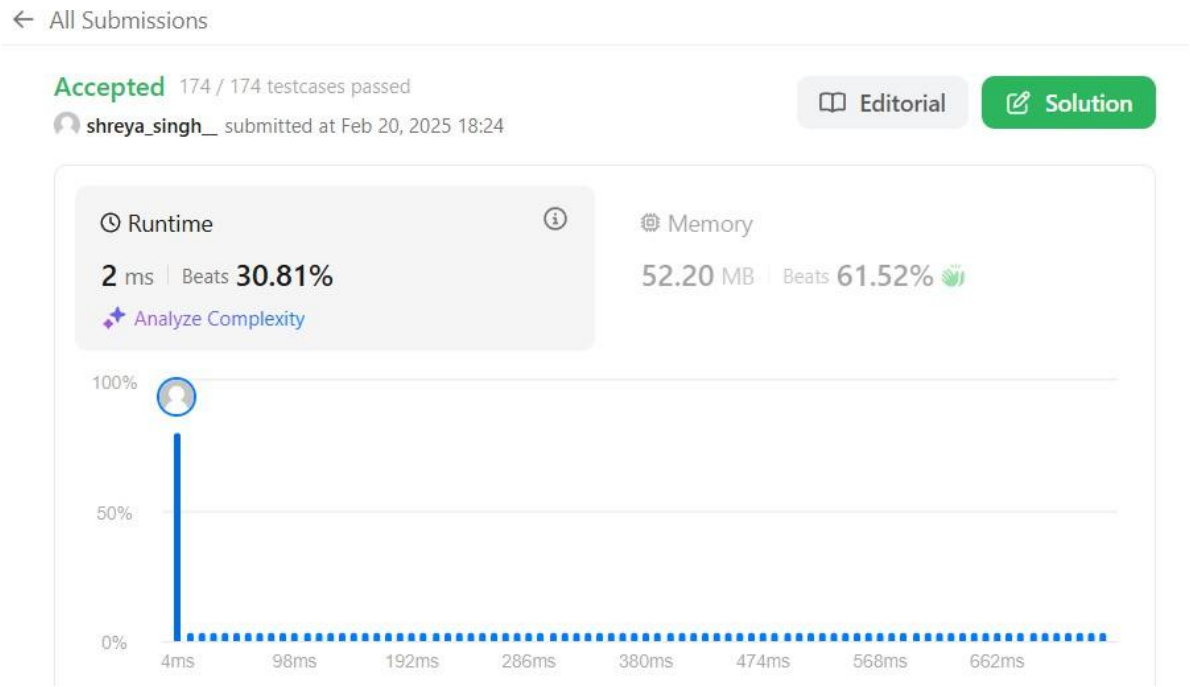
1. **Aim:** Sort List.

2. **Implementation/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) {
        // using merge sort

        // base case
        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;
    }
};
```

3. Output:



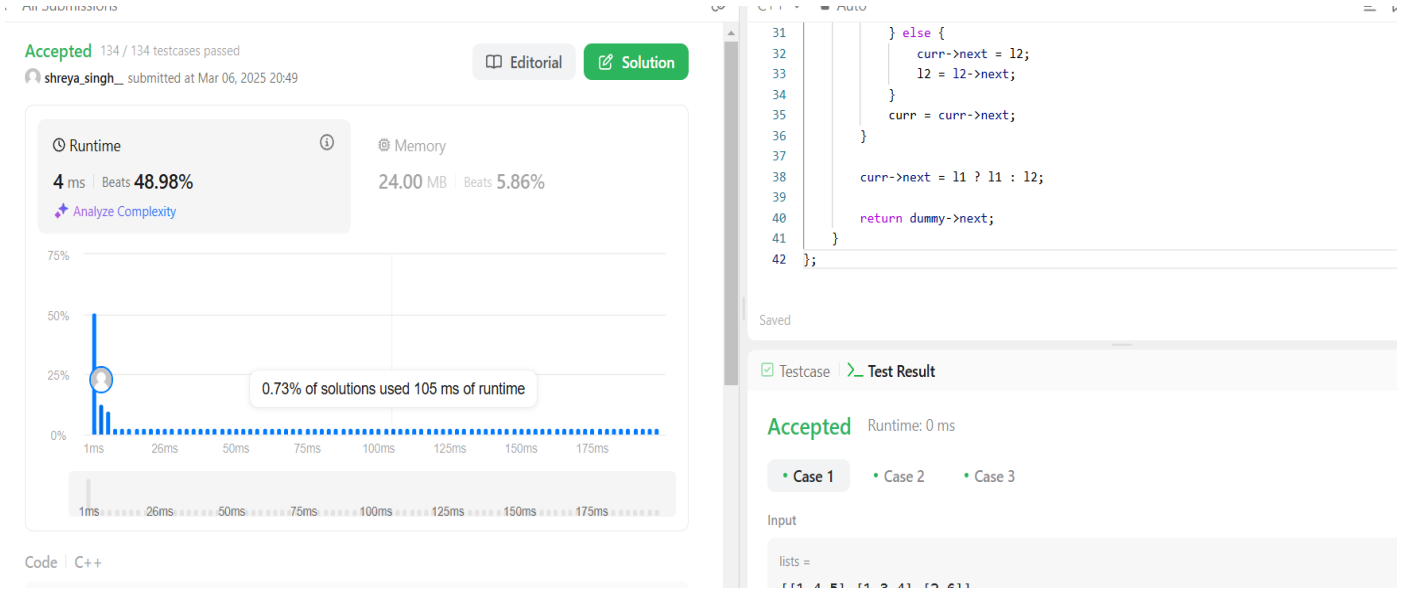
Problem 9

1. **Aim:** Merge k sorted lists.

2. **Implementation/Code:**

```
class Solution {
public:
    ListNode* mergeKLists(vector<ListNode*>& lists) {
        if (lists.empty()) {
            return nullptr;
        }
        return mergeKListsHelper(lists, 0, lists.size() - 1);
    }

    ListNode* mergeKListsHelper(vector<ListNode*>& lists, int start, int end) {
        if (start == end) {
            return lists[start];
        }
        if (start + 1 == end) {
            return merge(lists[start], lists[end]);
        }
        int mid = start + (end - start) / 2;
        ListNode* left = mergeKListsHelper(lists, start, mid);
        ListNode* right = mergeKListsHelper(lists, mid + 1, end);
        return merge(left, right);
    }

    ListNode* merge(ListNode* l1, ListNode* l2) {
        ListNode* dummy = new ListNode(0);
        ListNode* curr = dummy;

        while (l1 && l2) {
            if (l1->val < l2->val) {
                curr->next = l1;
                l1 = l1->next;
            } else {
                curr->next = l2;
                l2 = l2->next;
            }
            curr = curr->next;
        }

        curr->next = l1 ? l1 : l2;

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

3. Output:


Accepted 134 / 134 testcases passed
shreya_singh_ submitted at Mar 06, 2025 20:49
Editorial
Solution
Runtime
4 ms | Beats 48.98%
Analyze Complexity
Memory
24.00 MB | Beats 5.86%
75%
50%
25%
0%
0.73% of solutions used 105 ms of runtime
1ms 26ms 50ms 75ms 100ms 125ms 150ms 175ms
Code | C++
} else {
curr->next = l2;
l2 = l2->next;
curr = curr->next;
curr->next = l1 ? l1 : l2;
return dummy->next;
};
Saved
Testcase | Test Result
Accepted Runtime: 0 ms
Case 1
Case 2
Case 3
Input
lists =