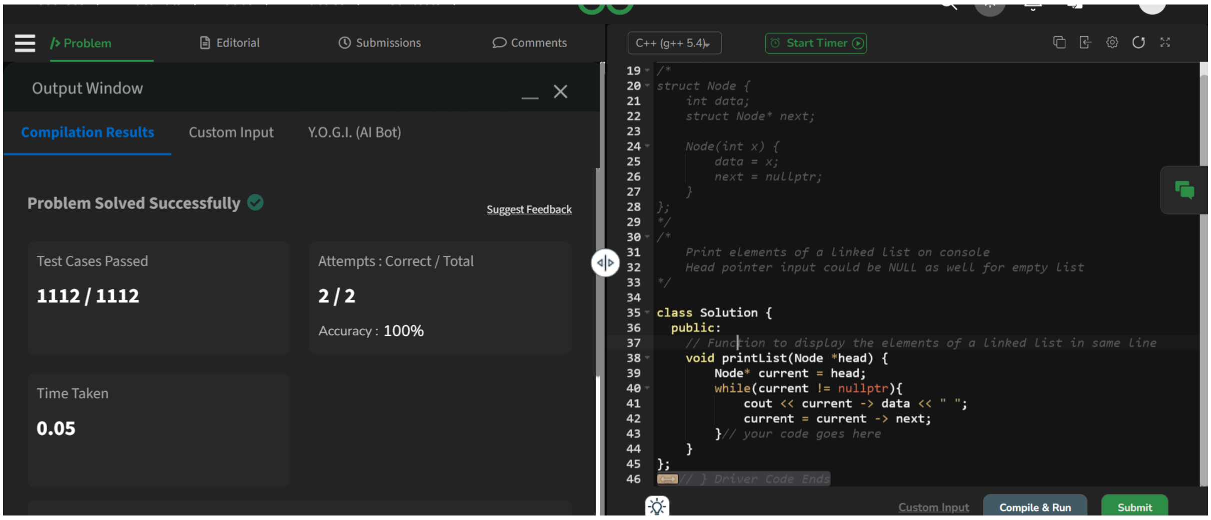
NAME : Arya Singh **UID :** 22BCS10203 **SECTION :** 607-B

ASSIGNMENT 3

1. Print Linked List:

```
class Solution {
  public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        Node* current = head;
        while(current != nullptr){
            cout << current -> data << " ";
            current = current -> next;
        }// your code goes here
    }
};
```



Problem Solved Successfully

Test Cases Passed
1112 / 1112

Attempts : Correct / Total
2 / 2
Accuracy : 100%

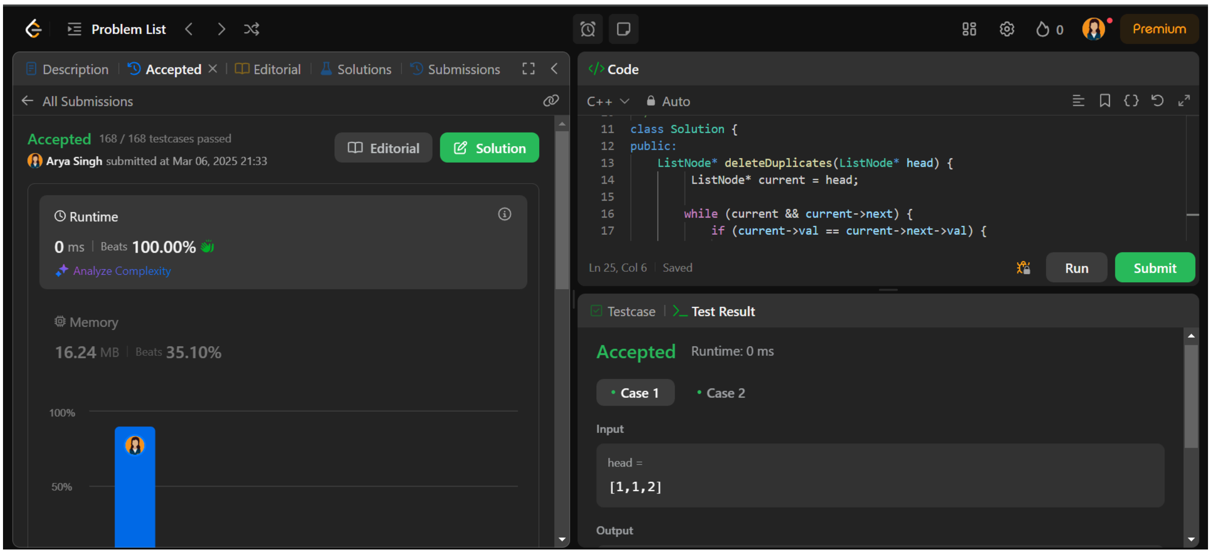
Time Taken
0.05

```
/*
struct Node {
    int data;
    struct Node* next;

    Node(int x) {
        data = x;
        next = nullptr;
    }
};
*/
/*
    Print elements of a linked list on console
    Head pointer input could be NULL as well for empty list
*/

class Solution {
  public:
    // Function to display the elements of a linked list in same line
    void printList(Node *head) {
        Node* current = head;
        while(current != nullptr){
            cout << current -> data << " ";
            current = current -> next;
        }// your code goes here
    }
};
// } Driver Code Ends
```

2. Remove duplicates from a sorted list:

```
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; // Skip duplicate node
            } else {
```

```
            current = current->next; // Move to the next node
        }
    }

    return head;
  }
};
```



Accepted 168 / 168 testcases passed

Arya Singh submitted at Mar 06, 2025 21:33

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

16.24 MB | Beats 35.10%

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* current = head;

        while (current && current->next) {
            if (current->val == current->next->val) {
```

Ln 25, Col 6 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2

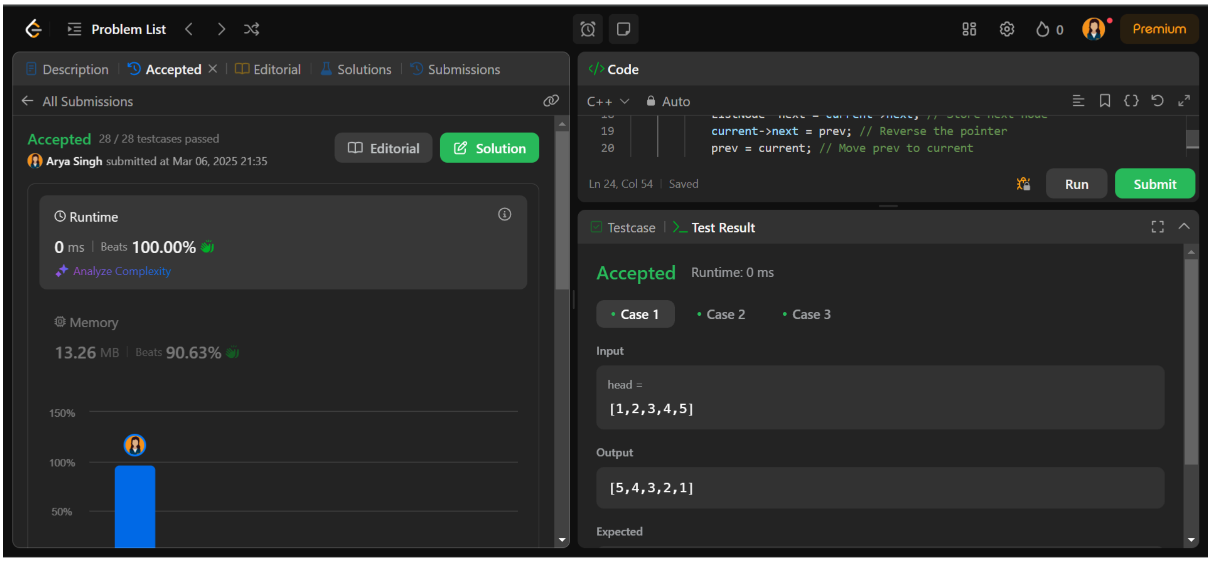
Input

head =
[1,1,2]

Output

3. Reverse a linked list:

```
ListNode* prev = nullptr;
    ListNode* current = head;

    while (current) {
        ListNode* next = current->next; // Store next node
        current->next = prev; // Reverse the pointer
        prev = current; // Move prev to current
        current = next; // Move current to next node
    }

    return prev; // New head of the reversed list
```



Accepted 28 / 28 testcases passed
Arya Singh submitted at Mar 06, 2025 21:35

Runtime
0 ms | Beats 100.00%

Memory
13.26 MB | Beats 90.63%

```
        current->next = prev; // Reverse the pointer
        prev = current; // Move prev to current
```

Ln 24, Col 54 | Saved

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

head =

```
[1,2,3,4,5]
```

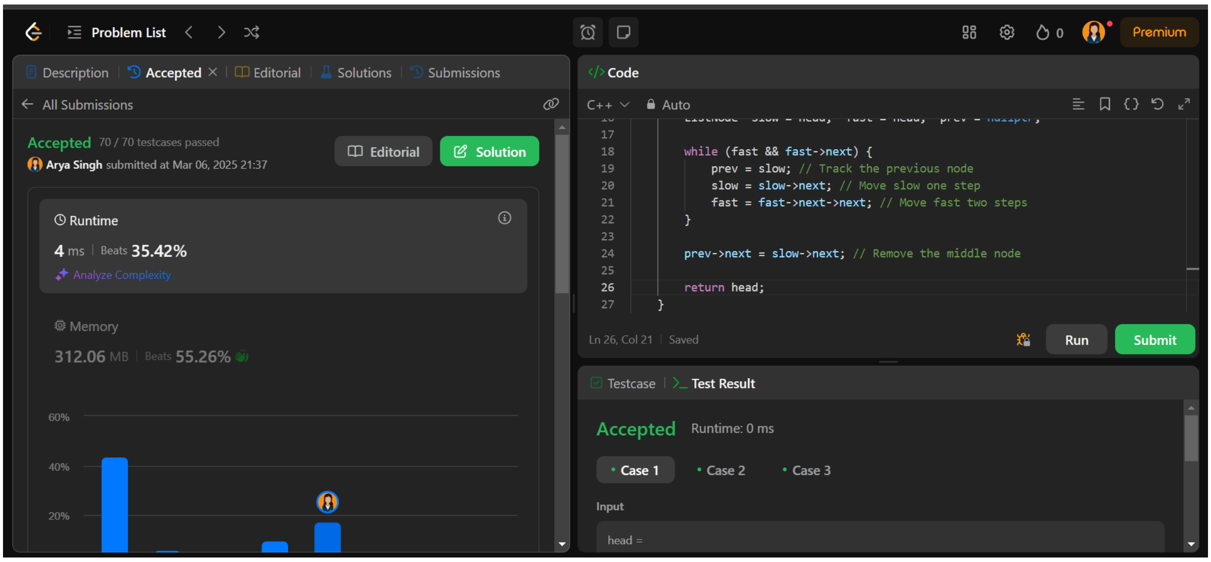
Output

```
[5,4,3,2,1]
```

Expected

4. Delete middle node of a list:

```
(!head || !head->next) return nullptr; // Edge case: 0 or 1 node

    ListNode* slow = head, *fast = head, *prev = nullptr;

    while (fast && fast->next) {
        prev = slow; // Track the previous node
        slow = slow->next; // Move slow one step
        fast = fast->next->next; // Move fast two steps
    }

    prev->next = slow->next; // Remove the middle node

    return head;
```



Accepted 70 / 70 testcases passed
Arya Singh submitted at Mar 06, 2025 21:37

Runtime
4 ms | Beats 35.42%

Memory
312.06 MB | Beats 55.26%

```

    while (fast && fast->next) {
        prev = slow; // Track the previous node
        slow = slow->next; // Move slow one step
        fast = fast->next->next; // Move fast two steps
    }

    prev->next = slow->next; // Remove the middle node

    return head;
}
```

Ln 26, Col 21 | Saved

Accepted Runtime: 0 ms

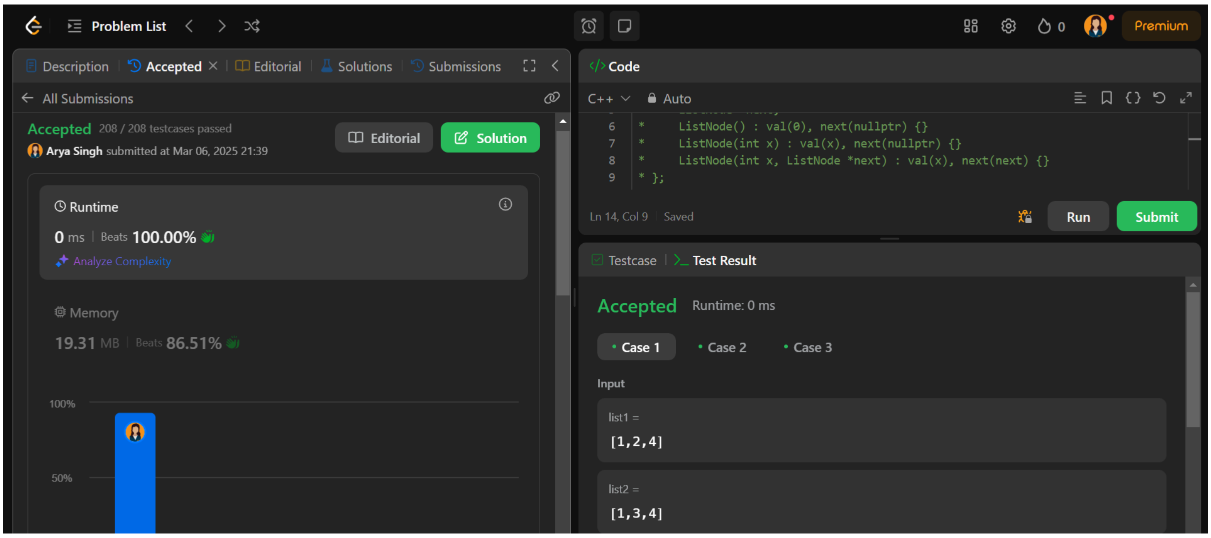
Case 1 Case 2 Case 3

Input

head =

5. Merge two sorted linked lists:

```
ListNode dummy(0); // Dummy node to simplify code
ListNode* current = &dummy;

  while (list1 && list2) {
      if (list1->val <= list2->val) {
          current->next = list1;
          list1 = list1->next;
      } else {
          current->next = list2;
          list2 = list2->next;
      }
      current = current->next;
  }

  // Attach remaining nodes from either list
  if (list1) current->next = list1;
  if (list2) current->next = list2;

  return dummy.next; // Return merged list (excluding dummy node)
```



Accepted 208 / 208 testcases passed
Arya Singh submitted at Mar 06, 2025 21:39

0 ms | Beats 100.00%

19.31 MB | Beats 86.51%

```
*     ListNode() : val(0), next(nullptr) {}
*     ListNode(int x) : val(x), next(nullptr) {}
*     ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
```

Ln 14, Col 9 | Saved

Accepted Runtime: 0 ms

Input

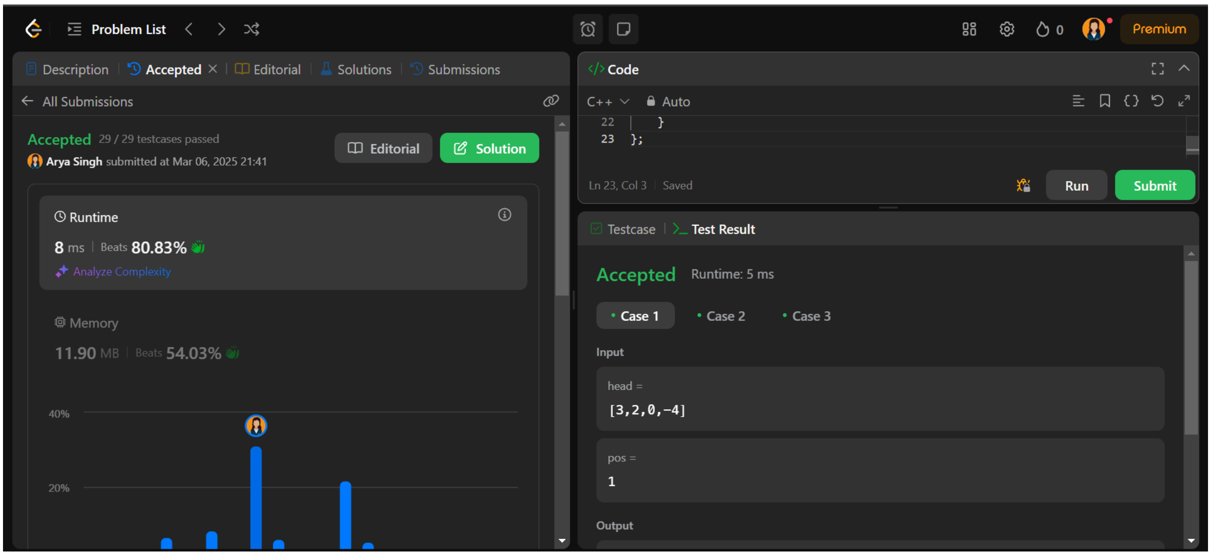
list1 =
[1,2,4]

list2 =
[1,3,4]

6. Detect a cycle in a linked list

```
class Solution {
public:
    bool hasCycle(ListNode *head) {
         ListNode *slow = head, *fast = head;

        while (fast && fast->next) {
            slow = slow->next;          // Move one step
            fast = fast->next->next;   // Move two steps
```

```
      if (slow == fast) return true; // Cycle detected
    }

    return false;
  }
};
```



Accepted 29 / 29 testcases passed

Arya Singh submitted at Mar 06, 2025 21:41

Runtime

8 ms | Beats 80.83%

Memory

11.90 MB | Beats 54.03%

```
    }
};
```

Ln 23, Col 3 | Saved

Accepted Runtime: 5 ms

Input

head =
[3,2,0,-4]

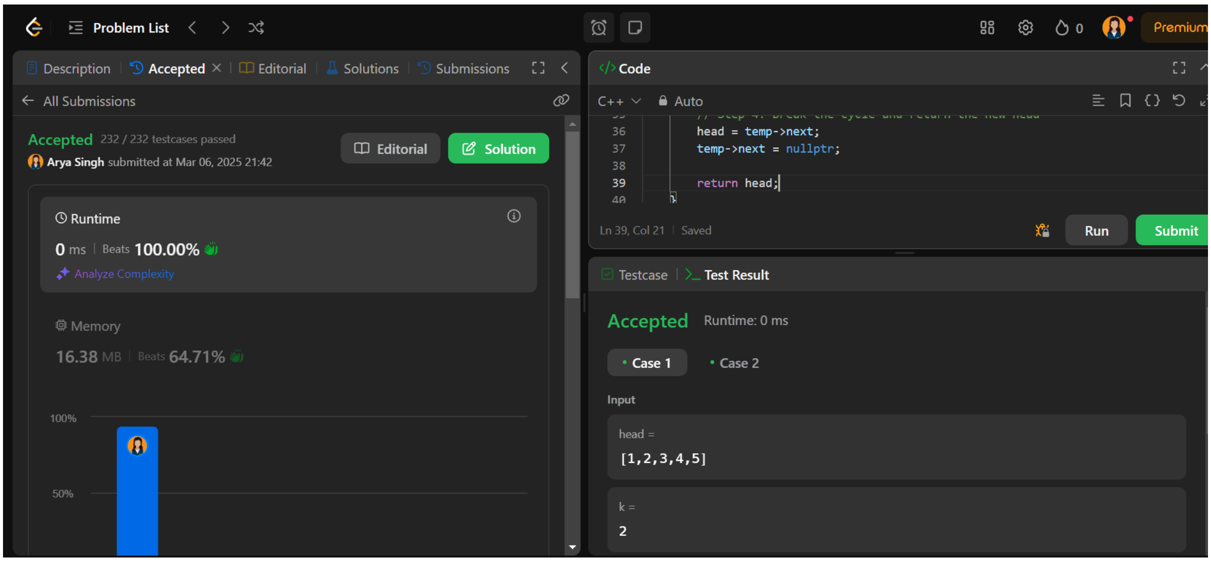
pos =
1

Output

7. Rotate a list:

```
class Solution {
public:
  ListNode* rotateRight(ListNode* head, int k) {
    if (!head || !head->next || k == 0) return head; // Edge cases

    // Step 1: Find the length of the list
    ListNode* temp = head;
    int length = 1;
    while (temp->next) {
      temp = temp->next;
      length++;
    }

    // Step 2: Compute the effective rotations
    k = k % length;
    if (k == 0) return head; // No rotation needed

    // Step 3: Find the new tail (length - k - 1) and new head
    temp->next = head; // Make it circular
    temp = head;
    for (int i = 0; i < length - k - 1; i++) {
      temp = temp->next;
```

```
        }

        // Step 4: Break the cycle and return the new head
        head = temp->next;
        temp->next = nullptr;

        return head;
    }
};
```



Accepted 232 / 232 testcases passed
Arya Singh submitted at Mar 06, 2025 21:42

Runtime
0 ms | Beats 100.00%
Analyze Complexity

Memory
16.38 MB | Beats 64.71%

```
        // Step 4: Break the cycle and return the new head
        head = temp->next;
        temp->next = nullptr;

        return head;
    }
```

Ln 39, Col 21 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2

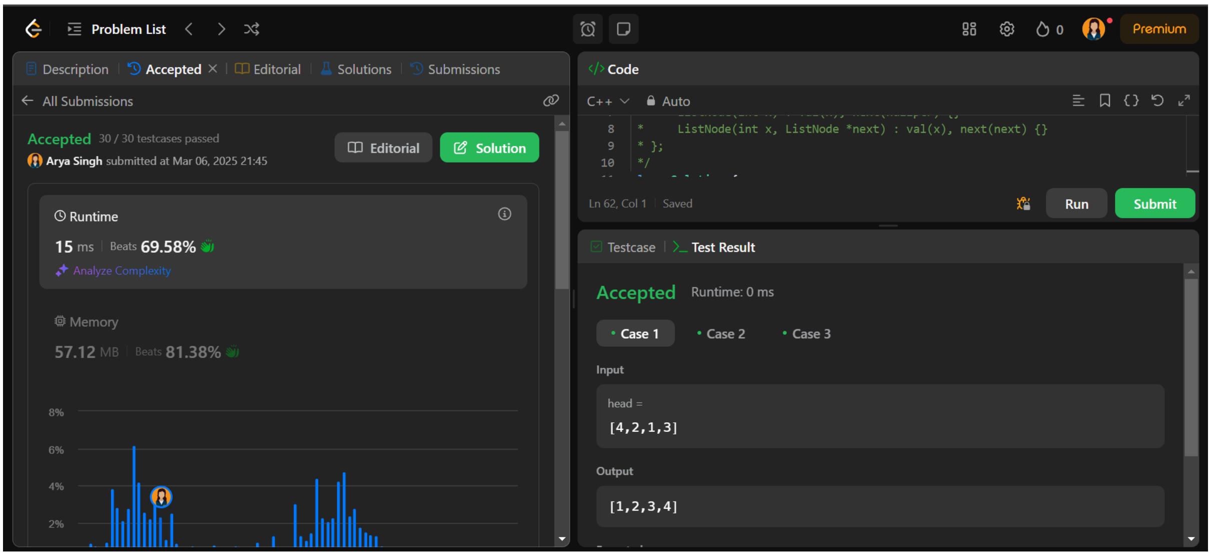
Input

head =
[1,2,3,4,5]

k =
2

8. Sort List:

```
class Solution {
public:
    // Function to find the middle of the linked list
    ListNode* getMid(ListNode* head) {
        ListNode* slow = head;
        ListNode* fast = head->next; // `fast` starts at `head->next` to split properly

        while (fast && fast->next) {
            slow = slow->next;
            fast = fast->next->next;
        }
        return slow;
    }

    // Function to merge two sorted linked lists
    ListNode* merge(ListNode* list1, ListNode* list2) {
        ListNode dummy(0); // Dummy node for merged list
        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;
        }

        // Attach remaining nodes from either list
        tail->next = list1 ? list1 : list2;
        return dummy.next;
    }

    ListNode* sortList(ListNode* head) {
        if (!head || !head->next) return head; // Base case

        // Step 1: Split the list into two halves
        ListNode* mid = getMid(head);
        ListNode* right = mid->next;
        mid->next = nullptr; // Break the list

        // Step 2: Recursively sort both halves
        ListNode* leftSorted = sortList(head);
        ListNode* rightSorted = sortList(right);

        // Step 3: Merge the sorted halves
        return merge(leftSorted, rightSorted);
    }
};
```



Accepted 30 / 30 testcases passed

Arya Singh submitted at Mar 06, 2025 21:45

Runtime

15 ms | Beats 69.58%

Memory

57.12 MB | Beats 81.38%

```
*     ListNode(int x, ListNode *next) : val(x), next(next) {}
* };
*/
```

Ln 62, Col 1 | Saved

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

head =

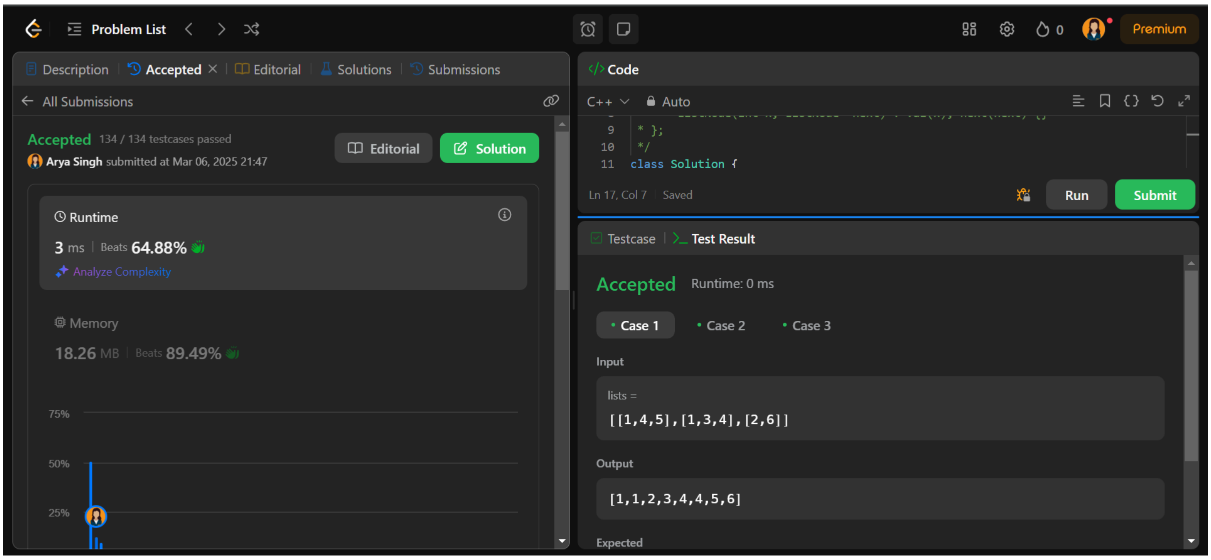
```
[4,2,1,3]
```

Output

```
[1,2,3,4]
```

9. Merge k sorted lists:

```
#include <queue>

class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val; // Min-Heap: smaller values have higher priority
        }
    };

    ListNode* mergeKLists(vector<ListNode*>& lists) {
        priority_queue<ListNode*, vector<ListNode*>, Compare> minHeap;

        // Push the head of each list into the min-heap
        for (ListNode* list : lists) {
            if (list) minHeap.push(list);
        }

        ListNode dummy(0); // Dummy node to simplify merging
        ListNode* tail = &dummy;

        while (!minHeap.empty()) {
            ListNode* smallest = minHeap.top();
            minHeap.pop();

            tail->next = smallest;
            tail = tail->next;

            if (smallest->next) minHeap.push(smallest->next);
        }

        return dummy.next; // Return merged sorted list
    }
};
```



Accepted 134 / 134 testcases passed

Arya Singh submitted at Mar 06, 2025 21:47

Runtime

3 ms | Beats 64.88%

Memory

18.26 MB | Beats 89.49%

```
 * };
 */
class Solution {
```

Ln 17, Col 7 | Saved

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

lists =

```
[[1,4,5],[1,3,4],[2,6]]
```

Output

```
[1,1,2,3,4,4,5,6]
```

Expected