



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Student Name: Lovely Sharma

Branch: BE-CSE

Semester: 6th

Subject Name: Advanced Programming II

UID: 22BCS11001

Section/Group: 22BCS_IOT_610_B

Date of Performance: 05/03/25

Subject Code: 22CSP-351

1. Print Linked List:

a) Code:

```
class Solution {  
    // Function to display the elements of a linked list in same line  
    void printList(Node head) {  
        Node cur = head;  
        while(cur!=null){  
            System.out.print(" "+cur.data);  
            cur = cur.next;  
        }  
    }  
}
```

b) Output:

The screenshot shows a coding platform interface. On the left, the 'Output Window' displays 'Compilation Results' for a custom input 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with 1112/1112 test cases passed, 1/8 attempts correct, 12% accuracy, 1/1 points scored, and a time taken of 2.07. On the right, the code editor shows the Java code for the linked list problem, including the Node class and the printList method.

Input:

1 2

Your Output:

1 2

Expected Output:

1 2



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

2. Remove duplicates from a sorted list:

a) Code:

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

        while (current && current->next) {
            if (current->val == current->next->val) {
                // Skip duplicate node
                current->next = current->next->next;
            } else {
                // Move to next node
                current = current->next;
            }
        }

        return head;
    }
};
```

b) Output:

The screenshot displays a coding platform interface with the following components:

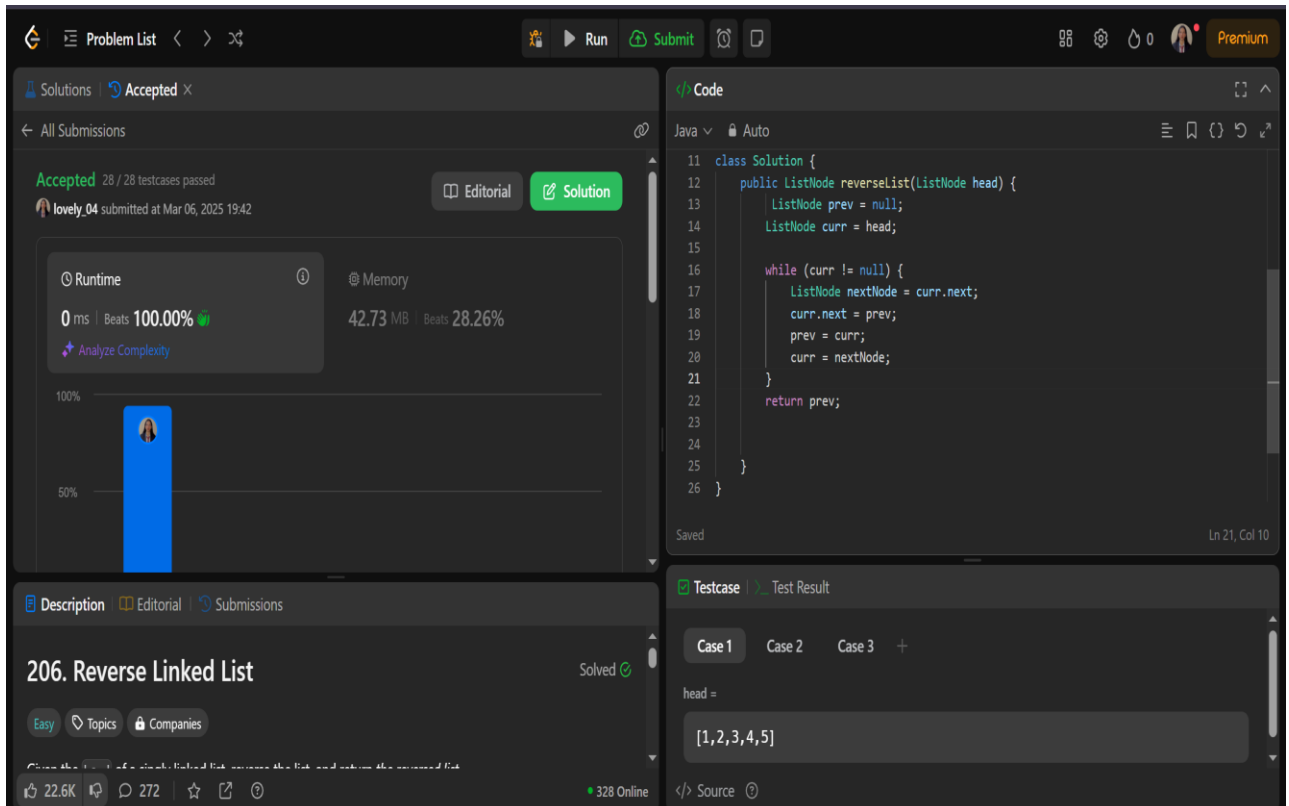
- Problem List:** Shows the current problem and navigation options.
- Solutions:** A tab showing the submission status. It indicates "Accepted" with "168 / 168 testcases passed". The submission was made by "lovely_04" on "Mar 06, 2025 19:40".
- Performance Metrics:**
 - Runtime:** 0 ms, Beats 100.00%.
 - Memory:** 16.21 MB, Beats 35.10%.
- Complexity Analysis:** A bar chart showing the runtime performance relative to other submissions. The user's submission is the fastest, represented by a blue bar at the 0 ms mark.
- Code Editor:** Displays the C++ code for the solution, which matches the code provided in the previous block. The code is saved and the cursor is at line 15, column 1.
- Test Results:** Shows the test cases passed. "Case 1" and "Case 2" are both marked as "Accepted" with a runtime of 0 ms.

3. Reverse a linked list:

a) Code:

```
class Solution {  
    public ListNode reverseList(ListNode head) {  
        ListNode prev = null;  
        ListNode curr = head;  
  
        while (curr != null) {  
            ListNode nextNode = curr.next;  
            curr.next = prev;  
            prev = curr;  
            curr = nextNode;  
        }  
        return prev;  
    }  
}
```

b) Output:



The screenshot displays a coding platform interface for the problem "206. Reverse Linked List". The top navigation bar includes "Problem List", "Run", "Submit", and "Premium" options. The left sidebar shows "Solutions" and "Accepted" tabs, with a submission by "lovely_04" at Mar 06, 2025 19:42. The main content area is divided into three sections: "Runtime" (0 ms, Beats 100.00%), "Memory" (42.73 MB, Beats 28.26%), and a "Description" section. The "Description" section includes the problem title "206. Reverse Linked List", a difficulty level of "Easy", and a brief description: "Given the head of a singly linked list, reverse the list, and return the reversed list." The right sidebar shows the "Code" editor with the Java solution code, and a "Testcase" section with a single test case: "head = [1,2,3,4,5]".

Runtime: 0 ms | Beats 100.00%
Memory: 42.73 MB | Beats 28.26%

206. Reverse Linked List
Easy | Topics | Companies

Given the head of a singly linked list, reverse the list, and return the reversed list.

22.6K | 272 | 328 Online

Testcase: Case 1 | Case 2 | Case 3 | +
head = [1,2,3,4,5]

4. Delete middle node of a list:

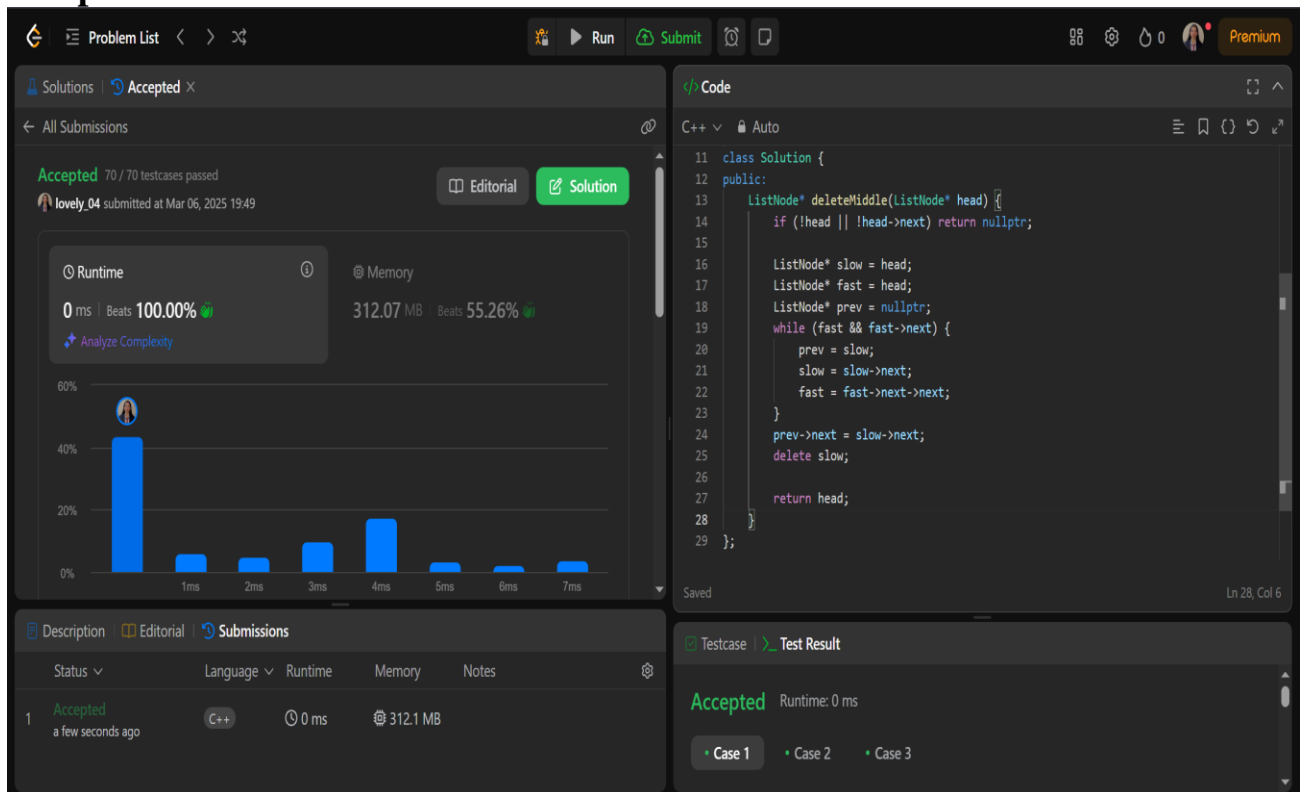
a) Code:

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

        return head;
    }
};
```

b) Output:



5. Merge two sorted linked lists:

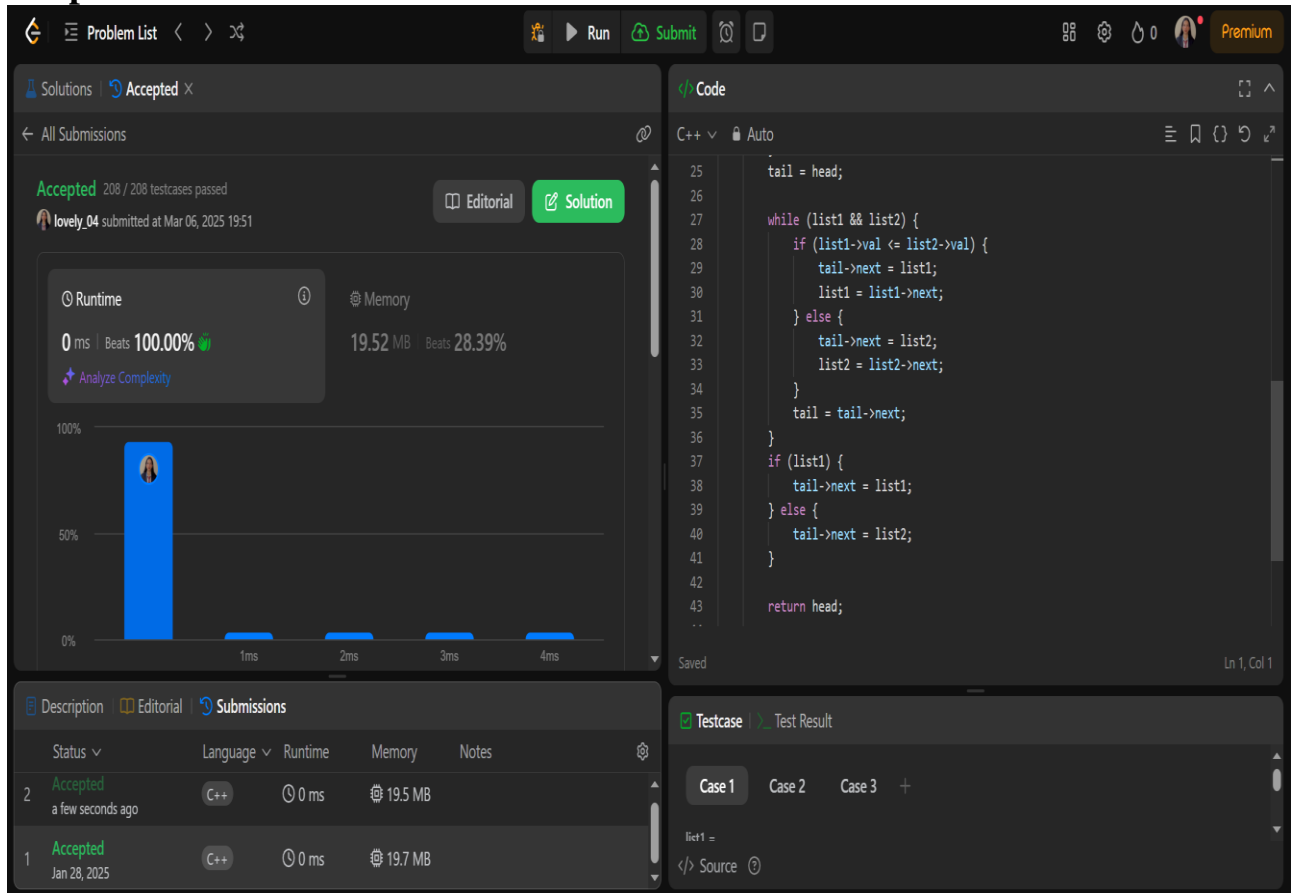
a) Code:

```
class Solution {
public:
    ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
        if (!list1) return list2;
        if (!list2) return list1;
        ListNode* head = nullptr;
        ListNode* tail = nullptr;
        if (list1->val <= list2->val) {
            head = list1;
            list1 = list1->next;
        } else {
            head = list2;
            list2 = list2->next;
        }
        tail = head;

        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;
        } else {
            tail->next = list2;
        }

        return head;
    }
};
```

b) Output:



6. Detect a cycle in a linked list:

a) Code:

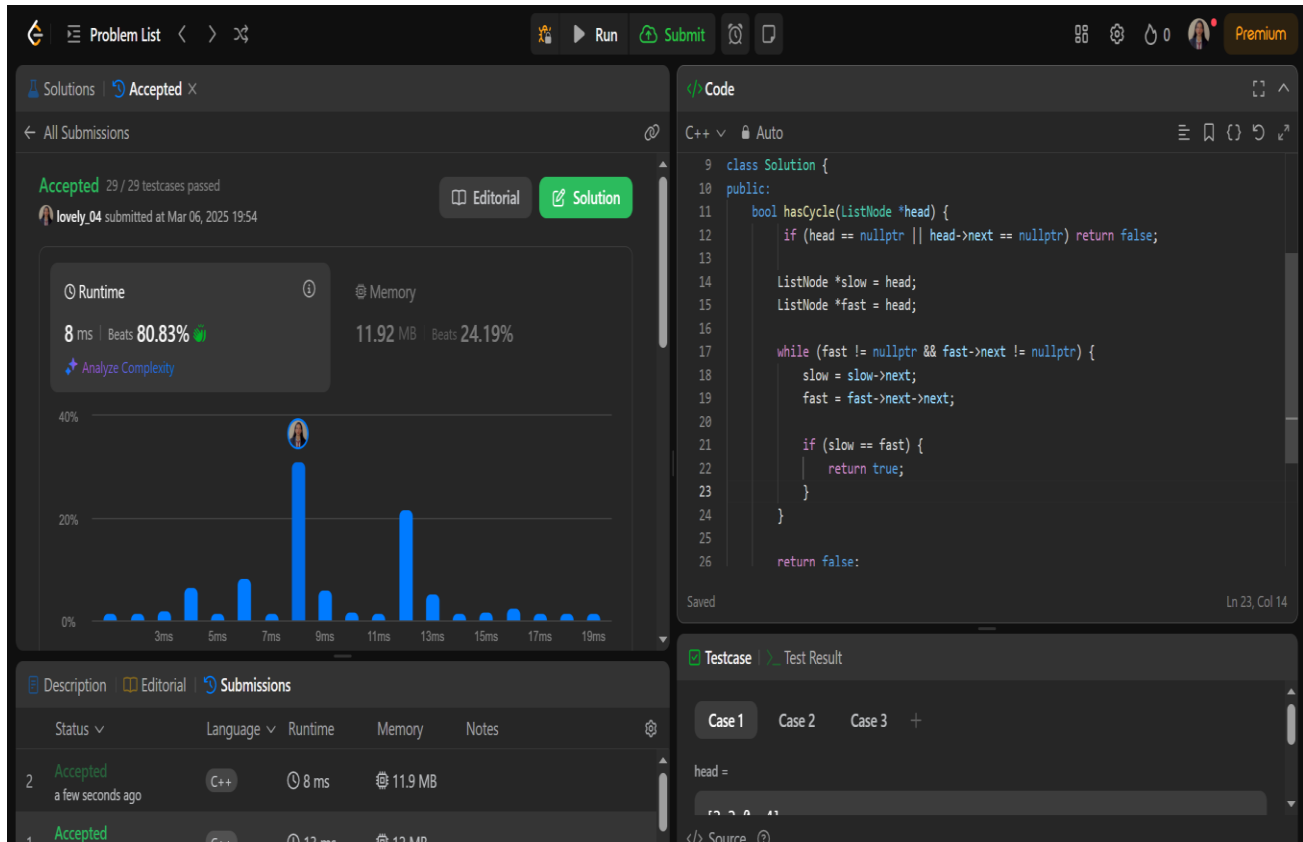
```
class Solution {
public:
    bool hasCycle(ListNode *head) {
        if (head == nullptr || head->next == nullptr) return false;

        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;
    }
};
```

b) Output:

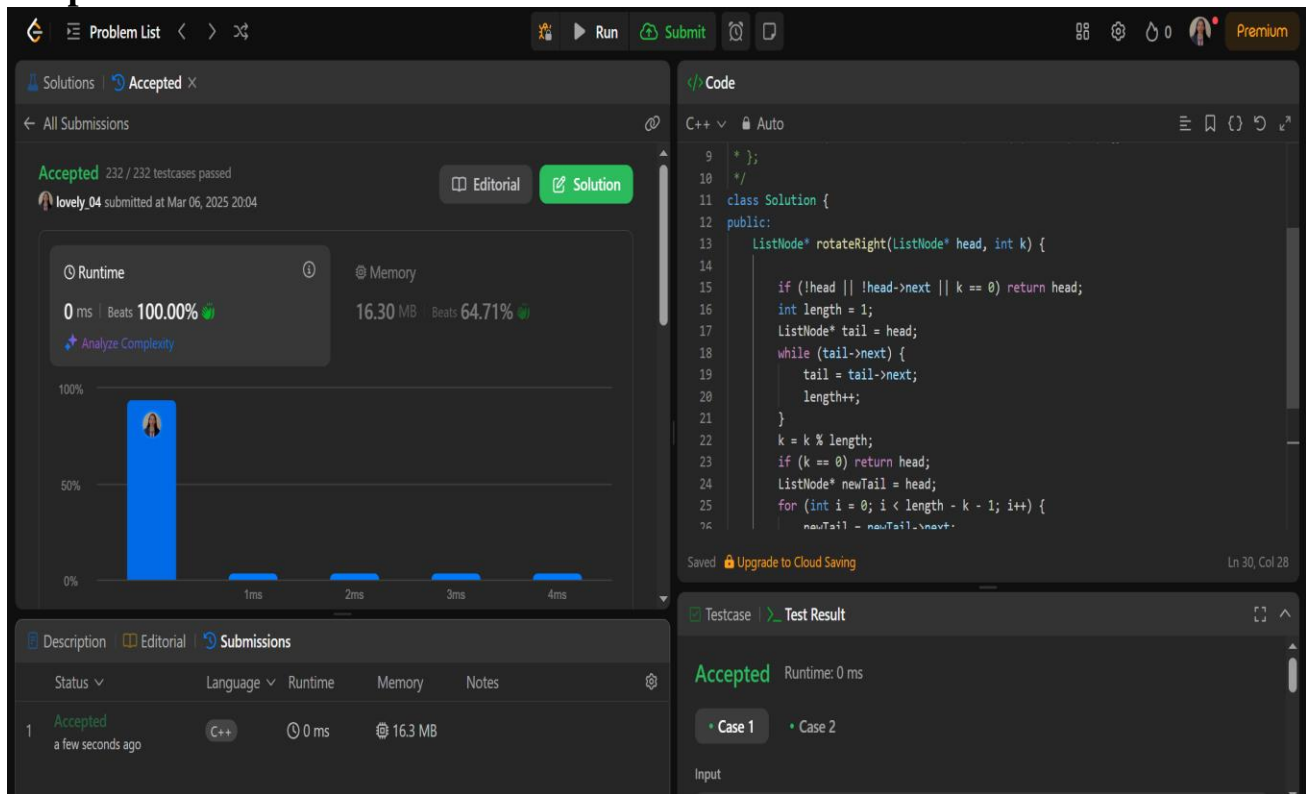


7. Rotate a list:

a) Code:

```
class Solution {
public:
    ListNode* rotateRight(ListNode* head, int k) {
        if (!head || !head->next || k == 0) return head;
        int length = 1;
        ListNode* tail = head;
        while (tail->next) {
            tail = tail->next;
            length++;
        }
        k = k % length;
        if (k == 0) return head;
        ListNode* newTail = head;
        for (int i = 0; i < length - k - 1; i++) {
            newTail = newTail->next;
        }
        ListNode* newHead = newTail->next;
        newTail->next = nullptr;
        tail->next = head;
        return newHead;
    }
};
```

b) Output:



8. Sort List:

a) 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;

```



```

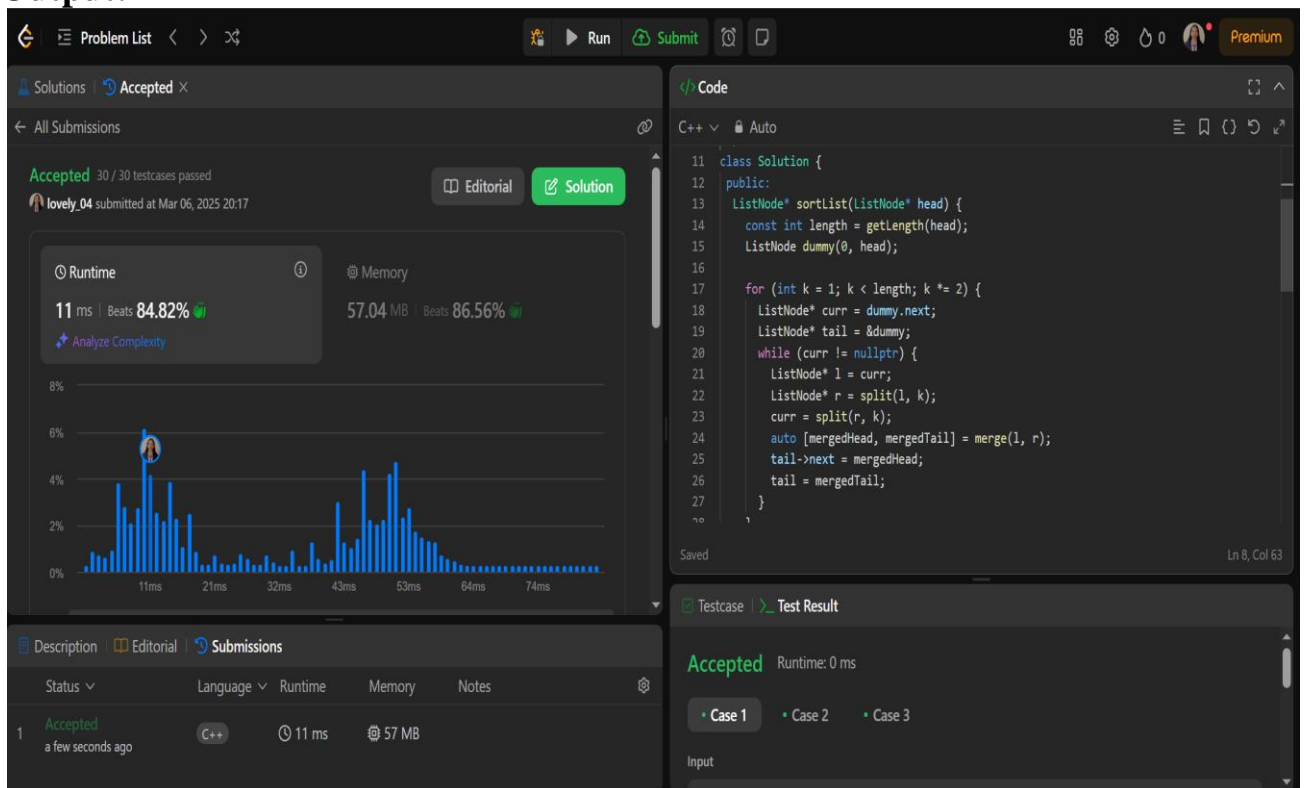
    for (ListNode* curr = head; curr; curr = curr->next)
        ++length;
    return length;
}

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};
}

```

b) Output:



9. Merge k sorted lists:

a) Code:

```
#include <queue>
class Solution {
public:
    struct Compare {
        bool operator()(ListNode* a, ListNode* b) {
            return a->val > b->val; // Min-Heap based on node values
        }
    };

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

        // Step 1: Push the head of each non-empty list into the minHeap
        for (ListNode* list : lists) {
            if (list) minHeap.push(list);
        }

        ListNode dummy(0);
        ListNode* tail = &dummy;

        // Step 2: Process the heap
        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;
    };
};
```



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

b) Output:

The screenshot displays a coding platform interface with the following components:

- Problem List:** Navigation icons for problem list, run, submit, and other tools.
- Solutions:** Tab showing 'Accepted' status for 134 / 134 testcases passed. Submission by 'lovely_04' at Mar 06, 2025 20:26.
- Runtime and Memory:** Runtime is 1 ms (Beats 76.17%) and Memory is 18.48 MB (Beats 66.07%).
- Code Editor:** C++ code for merging k sorted lists using a min-heap.
- Testcase Results:** All testcases are 'Accepted'.

C++ Code:

```
11 #include <queue>
12 class Solution {
13 public:
14     struct Compare {
15         bool operator()(ListNode* a, ListNode* b) {
16             return a->val > b->val;
17         }
18     };
19     ListNode* mergeKLists(vector<ListNode*>& lists) {
20         priority_queue<ListNode*, vector<ListNode*&, Compare> minHeap;
21         for (ListNode* list : lists) {
22             if (list) minHeap.push(list);
23         }
24         ListNode dummy(0);
25         ListNode* tail = &dummy;
26         while (!minHeap.empty()) {
27             ListNode* smallest = minHeap.top();
28             minHeap.pop();
```

Submission Table:

Status	Language	Runtime	Memory	Notes
Accepted	C++	1 ms	18.5 MB	a few seconds ago



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.