

Name: **KARANDEEP SINGH**

UID: **22BCS16869**

Subject: **Advance Programming Lab – II**

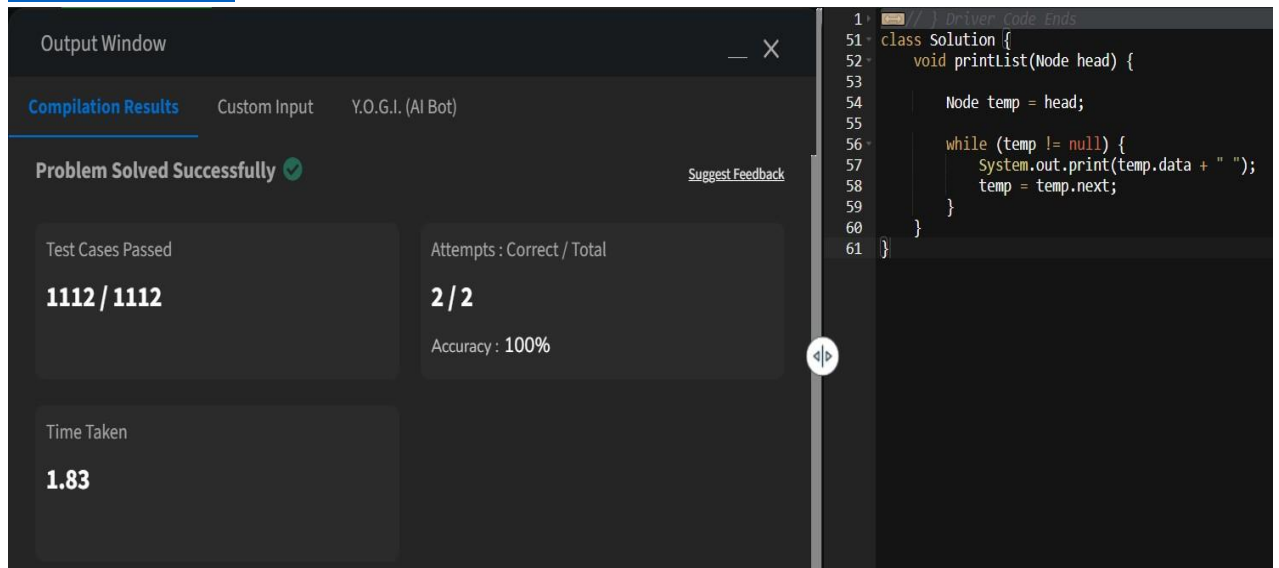
Section: **22BCS-IoT_626 [B]**

Subject Code: **22CSP-351**

Date of Submission: **14th Feb**

Assignment – 3

1. Print linked list



The screenshot shows a coding interface with the following details:

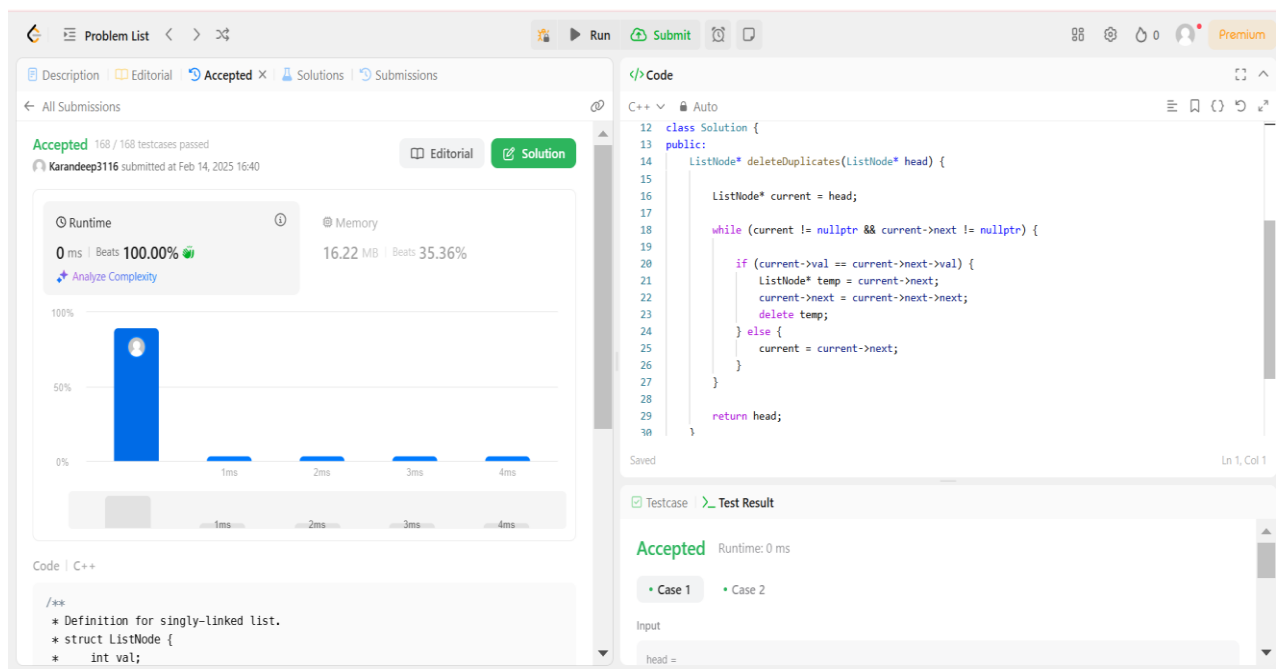
- Output Window:**
 - Compilation Results: Custom Input, Y.O.G.I. (AI Bot)
 - Problem Solved Successfully (Green Checkmark)
 - Test Cases Passed: **1112 / 1112**
 - Attempts: Correct / Total: **2 / 2**
 - Accuracy: **100%**
 - Time Taken: **1.83**
- Code Editor:**

```

1 // Driver Code Ends
51 class Solution {
52     void printList(Node head) {
53
54         Node temp = head;
55
56         while (temp != null) {
57             System.out.print(temp.data + " ");
58             temp = temp.next;
59         }
60     }
61 }

```

2. Remove duplicates from a sorted list



The screenshot shows a coding interface with the following details:

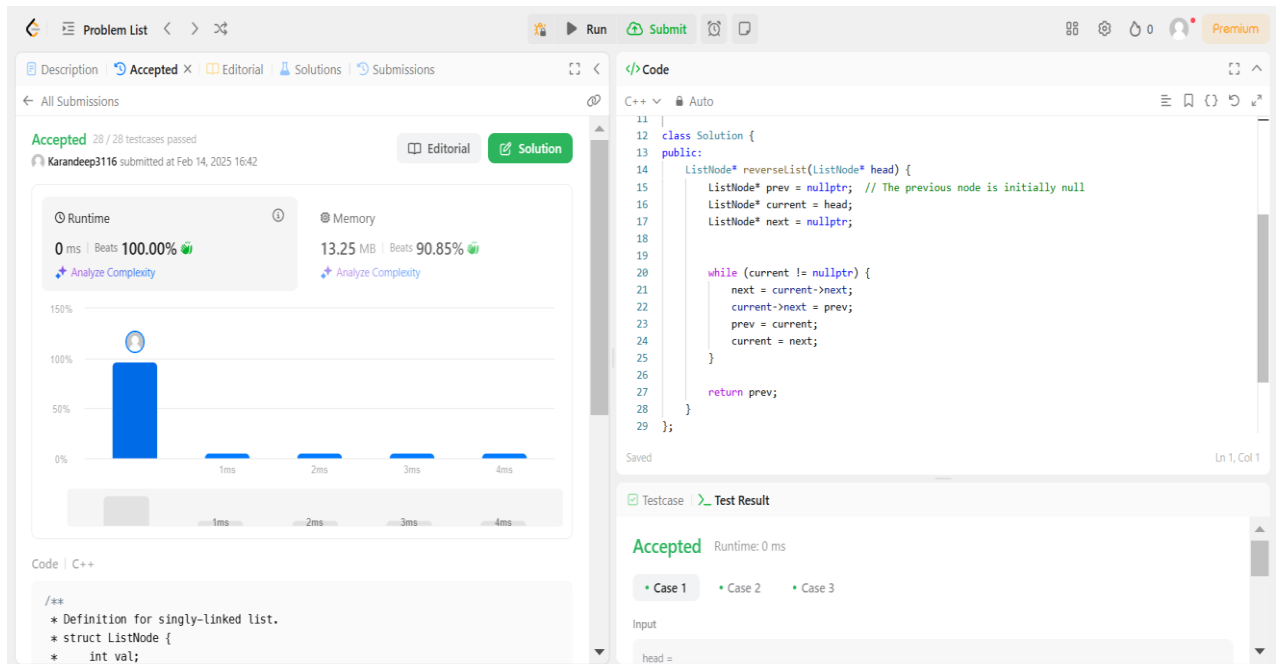
- Problem List:**
 - Description | Editorial | Accepted (X) | Solutions | Submissions
 - All Submissions
 - Accepted: 168 / 168 testcases passed
 - Karandeep3116 submitted at Feb 14, 2025 16:40
 - Editorial | Solution
- Runtime & Memory:**
 - Runtime: 0 ms | Beats 100.00%
 - Memory: 16.22 MB | Beats 35.36%
 - Analyze Complexity
- Code Editor:**

```

12 class Solution {
13 public:
14     ListNode* deleteDuplicates(ListNode* head) {
15
16         ListNode* current = head;
17
18         while (current != nullptr && current->next != nullptr) {
19
20             if (current->val == current->next->val) {
21                 ListNode* temp = current->next;
22                 current->next = current->next->next;
23                 delete temp;
24             } else {
25                 current = current->next;
26             }
27         }
28
29         return head;
30     }

```
- Testcase:**
 - Accepted Runtime: 0 ms
 - Case 1 | Case 2
 - Input: head =

3. Reverse a Linked List



The screenshot displays a coding interface for the problem "Reverse a Linked List". The left panel shows the submission status as "Accepted" with 28/28 testcases passed. The runtime is 0 ms (100.00% beats) and memory is 13.25 MB (90.85% beats). A bar chart shows the runtime distribution. The right panel shows the C++ code for reversing a linked list.

```

class Solution {
public:
    ListNode* reverseList(ListNode* head) {
        ListNode* prev = nullptr; // The previous node is initially null
        ListNode* current = head;
        ListNode* next = nullptr;

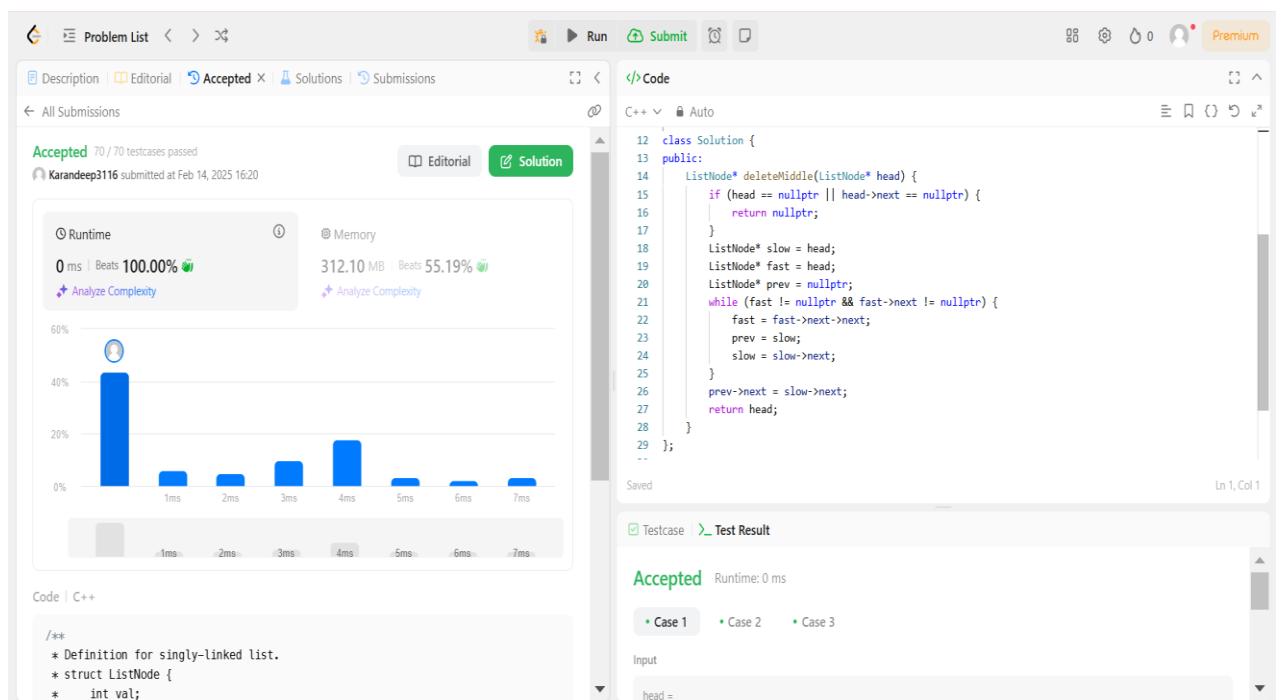
        while (current != nullptr) {
            next = current->next;
            current->next = prev;
            prev = current;
            current = next;
        }

        return prev;
    }
};

```

The code defines a `ListNode` structure and a `reverseList` function that iteratively reverses the linked list by updating the `next` pointer of each node to point to the previous node.

4. Delete middle node of a list



The screenshot displays a coding interface for the problem "Delete middle node of a list". The left panel shows the submission status as "Accepted" with 70/70 testcases passed. The runtime is 0 ms (100.00% beats) and memory is 312.10 MB (55.19% beats). A bar chart shows the runtime distribution. The right panel shows the C++ code for deleting the middle node of a linked list.

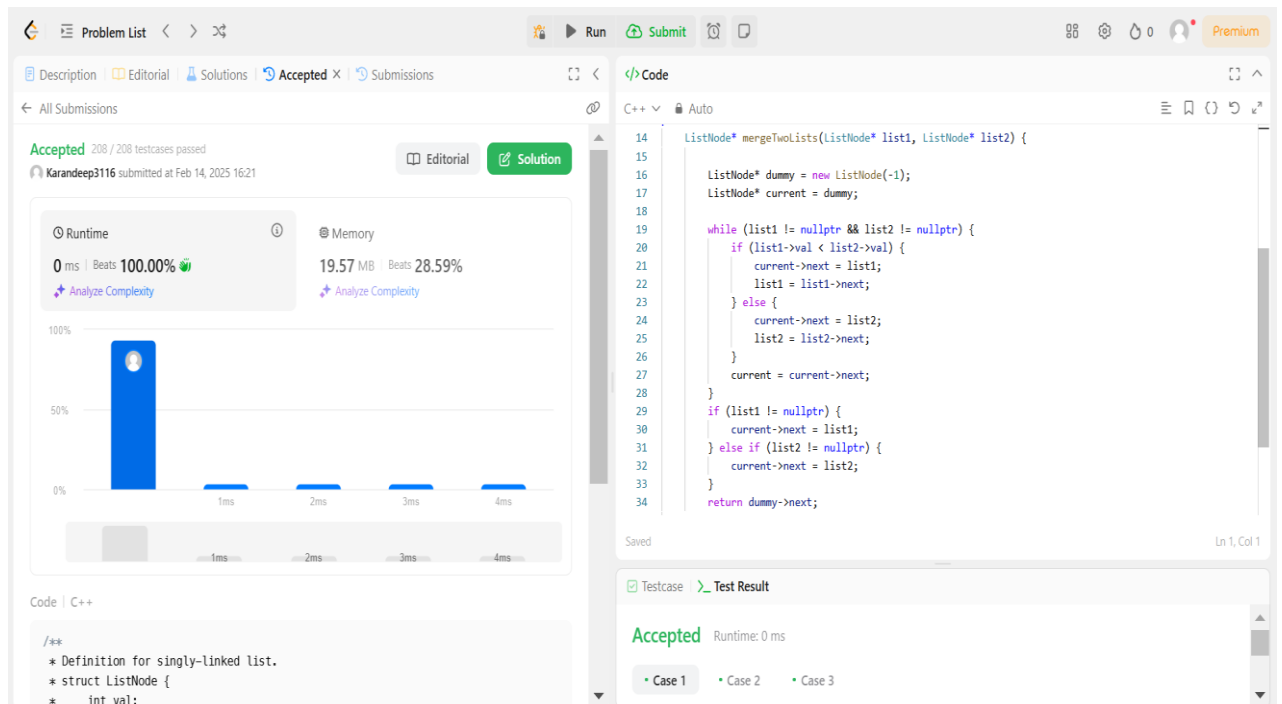
```

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

```

The code defines a `deleteMiddle` function that uses a slow and fast pointer technique to find the middle node of the linked list. Once the middle node is identified, it is removed by updating the `next` pointer of the previous node to skip the middle node.

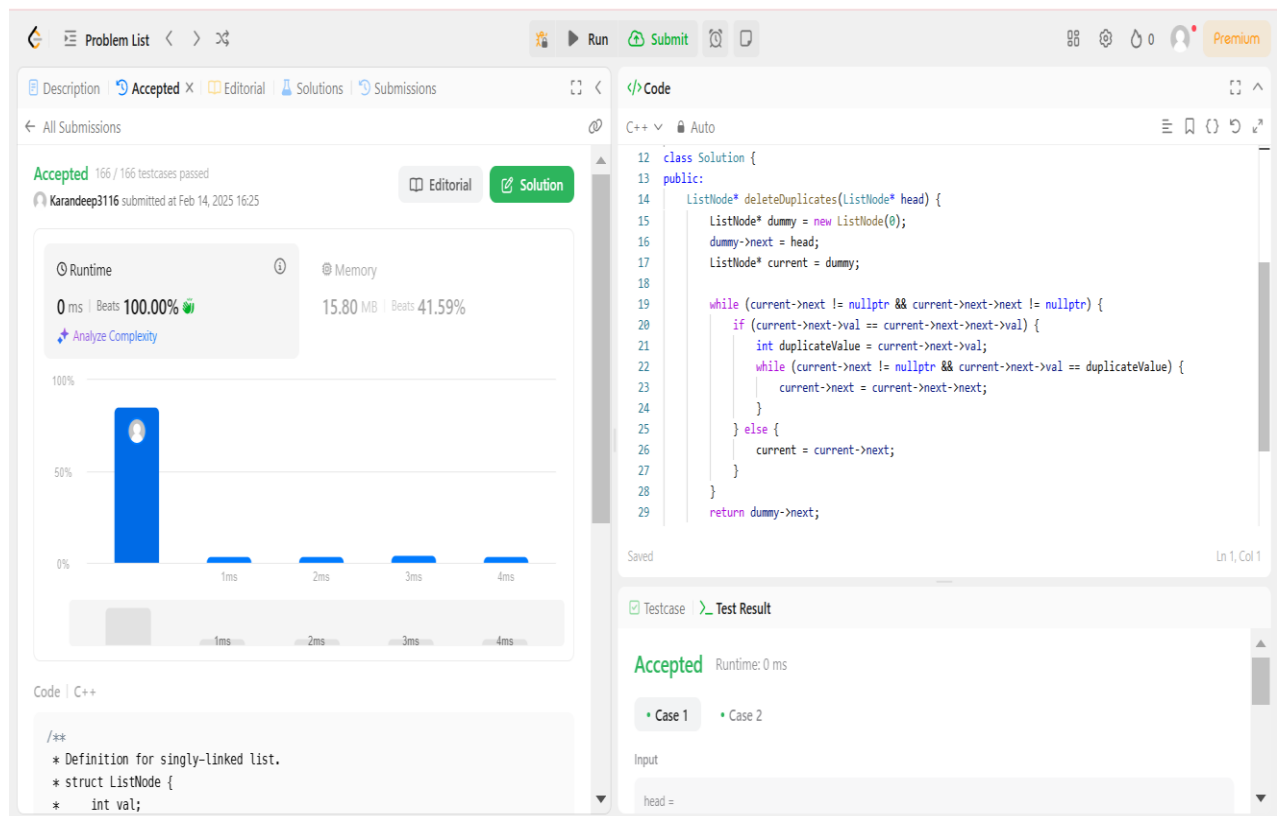
5. Merge two sorted linked lists



The screenshot shows a C++ code editor with the following components:

- Problem List:** Shows the problem "Merge two sorted linked lists" with a status of "Accepted" and 208/208 testcases passed. The submission was made by Karandeep3116 on Feb 14, 2025 at 16:21.
- Runtime and Memory:** Runtime is 0 ms (Beats 100.00%) and Memory is 19.57 MB (Beats 28.59%).
- Code:** The code defines a singly-linked list structure and a function to merge two sorted linked lists. The function uses a dummy node and a while loop to merge the two lists.
- Testcase:** Shows the test result for Case 1, which is "Accepted" with a runtime of 0 ms.

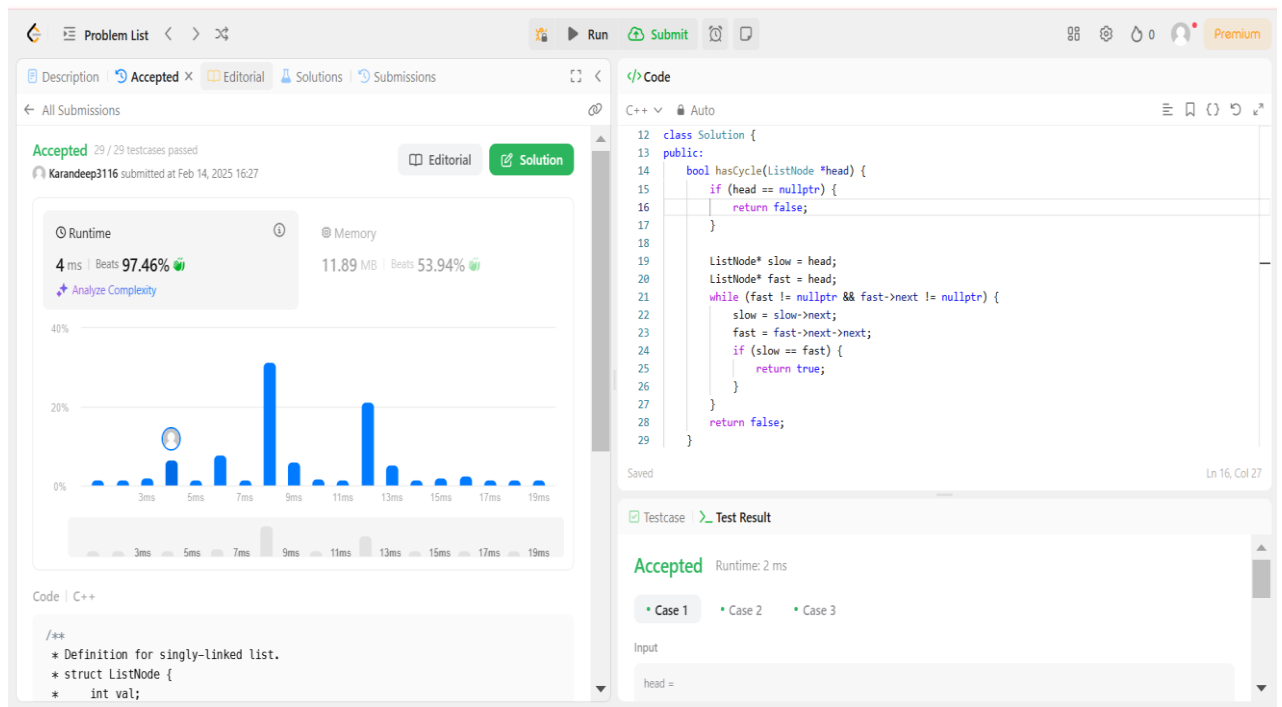
6. Remove duplicates from sorted lists 2



The screenshot shows a C++ code editor with the following components:

- Problem List:** Shows the problem "Remove duplicates from sorted lists 2" with a status of "Accepted" and 166/166 testcases passed. The submission was made by Karandeep3116 on Feb 14, 2025 at 16:25.
- Runtime and Memory:** Runtime is 0 ms (Beats 100.00%) and Memory is 15.80 MB (Beats 41.59%).
- Code:** The code defines a singly-linked list structure and a function to remove duplicates from a sorted linked list. The function uses a dummy node and a while loop to traverse the list and remove duplicates.
- Testcase:** Shows the test result for Case 1, which is "Accepted" with a runtime of 0 ms.

7. Detect a cycle in a linked list



Accepted 29 / 29 testcases passed
Karandeep3116 submitted at Feb 14, 2025 16:27

Runtime 4 ms | Beats 97.46%
Memory 11.89 MB | Beats 53.94%

Code C++

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;

```

```

12 class Solution {
13 public:
14     bool hasCycle(ListNode *head) {
15         if (head == nullptr) {
16             return false;
17         }
18
19         ListNode* slow = head;
20         ListNode* fast = head;
21         while (fast != nullptr && fast->next != nullptr) {
22             slow = slow->next;
23             fast = fast->next->next;
24             if (slow == fast) {
25                 return true;
26             }
27         }
28         return false;
29     }

```

Testcase **Test Result**

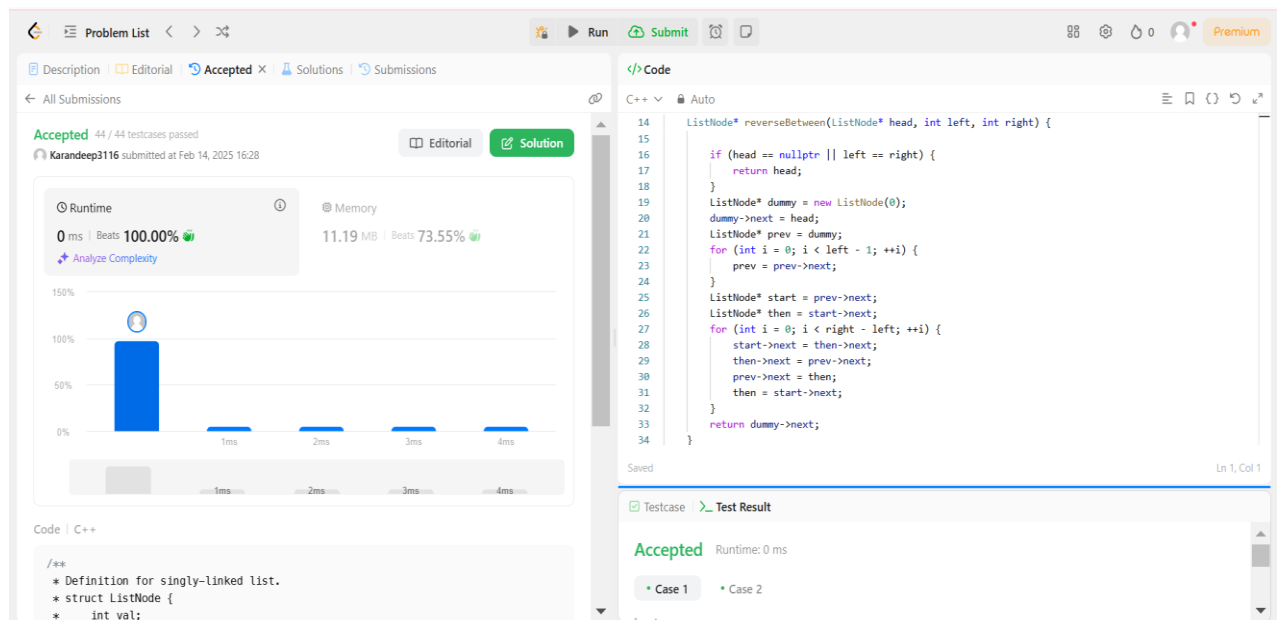
Accepted Runtime: 2 ms

Case 1 Case 2 Case 3

Input

head =

8. Reverse linked list 2



Accepted 44 / 44 testcases passed
Karandeep3116 submitted at Feb 14, 2025 16:28

Runtime 0 ms | Beats 100.00%
Memory 11.19 MB | Beats 73.55%

Code C++

```

/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;

```

```

14     ListNode* reverseBetween(ListNode* head, int left, int right) {
15
16         if (head == nullptr || left == right) {
17             return head;
18         }
19         ListNode* dummy = new ListNode(0);
20         dummy->next = head;
21         ListNode* prev = dummy;
22         for (int i = 0; i < left - 1; ++i) {
23             prev = prev->next;
24         }
25         ListNode* start = prev->next;
26         ListNode* then = start->next;
27         for (int i = 0; i < right - left; ++i) {
28             start->next = then->next;
29             then->next = prev->next;
30             prev->next = then;
31             then = start->next;
32         }
33         return dummy->next;
34     }

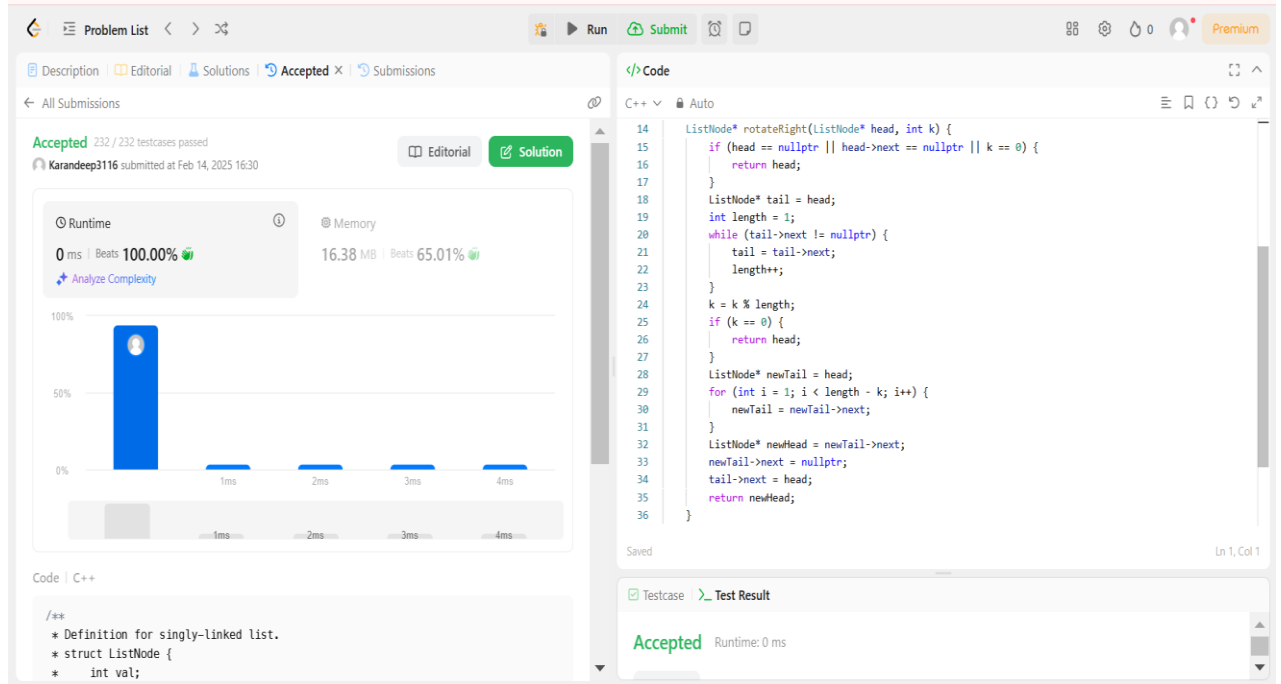
```

Testcase **Test Result**

Accepted Runtime: 0 ms

Case 1 Case 2

9. [Rotate a list](#)



Accepted 232 / 232 testcases passed
Karandeep3116 submitted at Feb 14, 2025 16:30

Runtime 0 ms | Beats 100.00%
Memory 16.38 MB | Beats 65.01%

Code | C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;

```

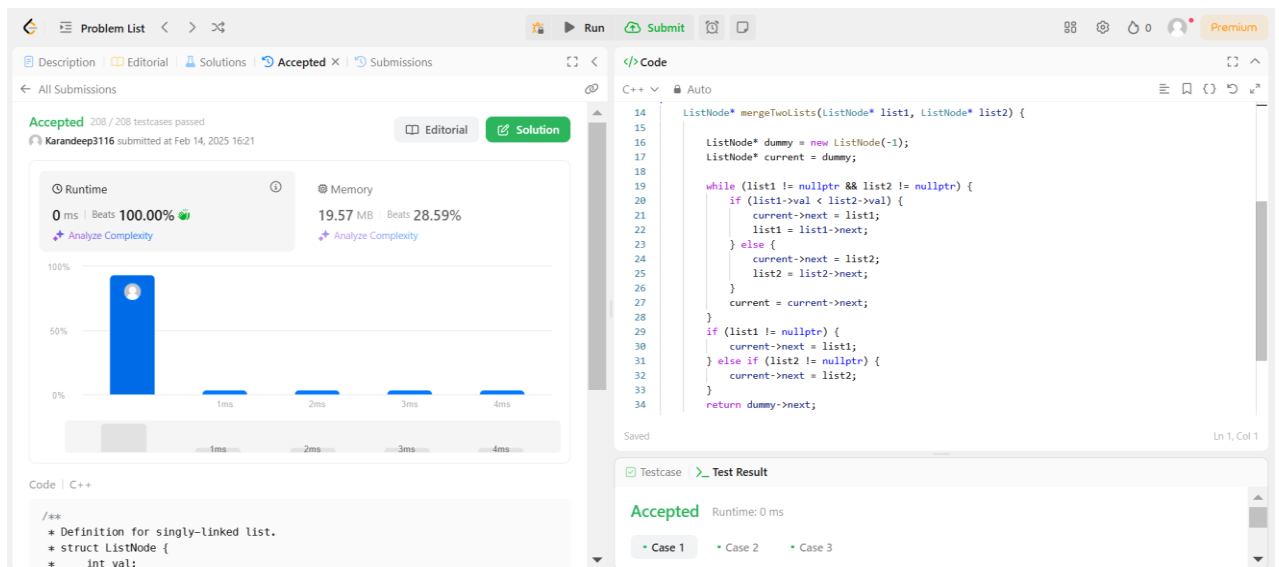
```

14  ListNode* rotateRight(ListNode* head, int k) {
15      if (head == nullptr || head->next == nullptr || k == 0) {
16          return head;
17      }
18      ListNode* tail = head;
19      int length = 1;
20      while (tail->next != nullptr) {
21          tail = tail->next;
22          length++;
23      }
24      k = k % length;
25      if (k == 0) {
26          return head;
27      }
28      ListNode* newTail = head;
29      for (int i = 1; i < length - k; i++) {
30          newTail = newTail->next;
31      }
32      ListNode* newHead = newTail->next;
33      newTail->next = nullptr;
34      tail->next = head;
35      return newHead;
36  }

```

Testcase **Test Result**
Accepted Runtime: 0 ms

10. [Sort List](#)



Accepted 208 / 208 testcases passed
Karandeep3116 submitted at Feb 14, 2025 16:21

Runtime 0 ms | Beats 100.00%
Memory 19.57 MB | Beats 28.59%

Code | C++

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;

```

```

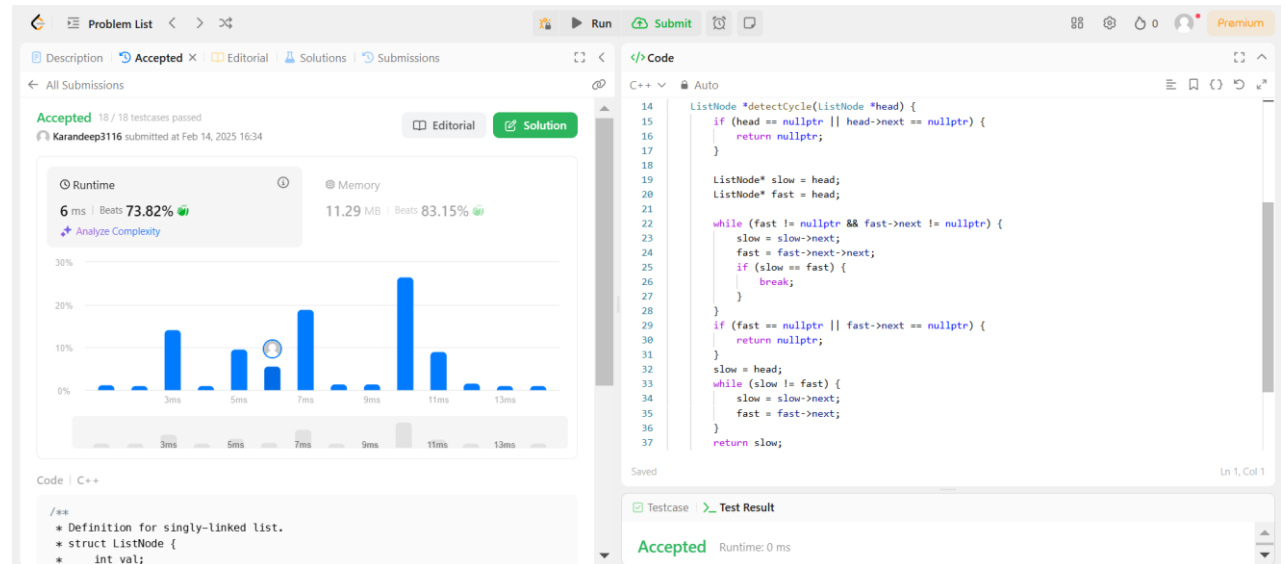
14  ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
15      ListNode* dummy = new ListNode(-1);
16      ListNode* current = dummy;
17
18      while (list1 != nullptr && list2 != nullptr) {
19          if (list1->val < list2->val) {
20              current->next = list1;
21              list1 = list1->next;
22          } else {
23              current->next = list2;
24              list2 = list2->next;
25          }
26          current = current->next;
27      }
28      if (list1 != nullptr) {
29          current->next = list1;
30      } else if (list2 != nullptr) {
31          current->next = list2;
32      }
33      return dummy->next;
34  }

```

Testcase **Test Result**
Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

11. [Detect a cycle in a linked list 2](#)



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

- Problem List:** Shows the current problem, "Detect a cycle in a linked list 2", with tabs for Description, Accepted (18 / 18 testcases passed), Editorial, Solutions, and Submissions.
- Submission Details:** The submission by Karandeep3116 is marked as "Accepted" and was submitted on Feb 14, 2025, at 16:34.
- Performance Metrics:**
 - Runtime:** 6 ms, Beats 73.82%.
 - Memory:** 11.29 MB, Beats 83.15%.
- Bar Chart:** A bar chart showing the distribution of runtime and memory usage across various test cases. The x-axis represents runtime in milliseconds (3ms, 5ms, 7ms, 9ms, 11ms, 13ms), and the y-axis represents the percentage of test cases (0%, 10%, 20%, 30%).
- Code Editor:** Displays the C++ code for the solution, which implements Floyd's Cycle-Finding Algorithm (tortoise and hare). The code is as follows:

```
14 ListNode *detectCycle(ListNode *head) {
15     if (head == nullptr || head->next == nullptr) {
16         return nullptr;
17     }
18
19     ListNode* slow = head;
20     ListNode* fast = head;
21
22     while (fast != nullptr && fast->next != nullptr) {
23         slow = slow->next;
24         fast = fast->next->next;
25         if (slow == fast) {
26             break;
27         }
28     }
29     if (fast == nullptr || fast->next == nullptr) {
30         return nullptr;
31     }
32     slow = head;
33     while (slow != fast) {
34         slow = slow->next;
35         fast = fast->next;
36     }
37     return slow;
}
```
- Testcase Results:** The "Test Result" tab shows the submission is "Accepted" with a runtime of 0 ms.