

# Assignment 3

## 1. Print Linked List

Code:

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

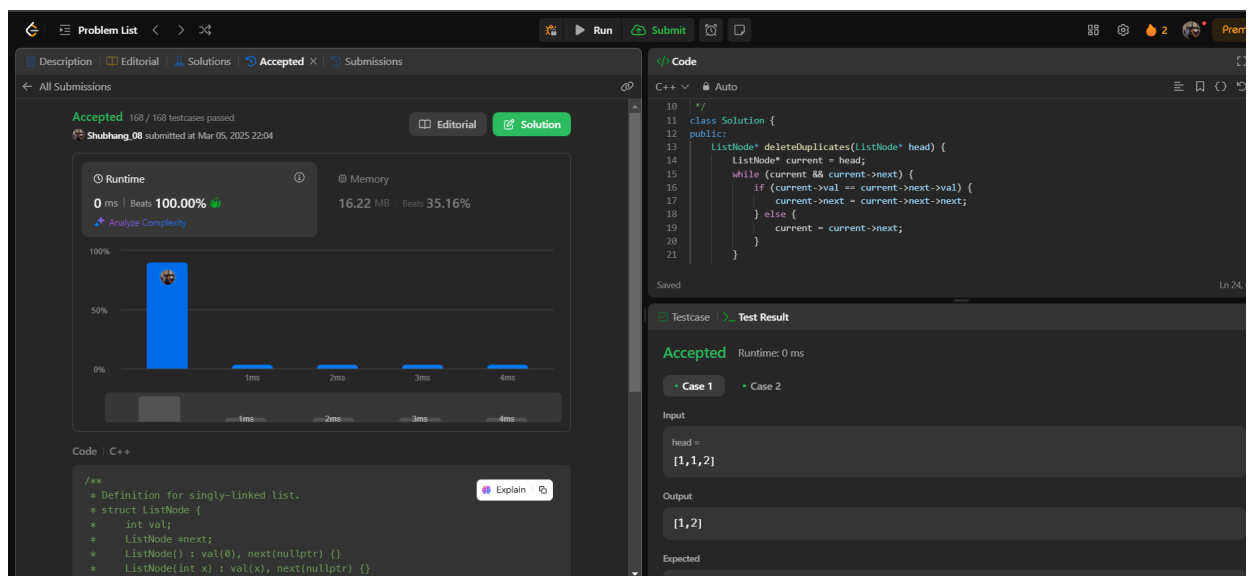
The screenshot displays a coding platform interface with the following components:

- Navigation Bar:** Includes links for Courses, Tutorials, Jobs, Practice, and Contests.
- Problem Header:** Shows the problem title, a '50% Refund' badge, and navigation tabs for Problem, Editorial, Submissions, and Comments.
- Output Window:** A green header bar with a close button.
- Compilation Results:** A section indicating 'Problem Solved Successfully' with a green checkmark and a 'Suggest Feedback' link.
- Performance Metrics:**
  - Test Cases Passed:** 1112 / 1112
  - Attempts:** Correct / Total: 1 / 1
  - Accuracy:** 100%
  - Points Scored:** 1 / 1
  - Time Taken:** 0.09
  - Your Total Score:** 8 (with an upward arrow)
- Solve Next:** A section with buttons for 'Count Linked List Nodes', 'Delete Alternate Nodes', and 'Insert in Middle of Linked List'.
- Code Editor:** Displays C++ code for the 'Print Linked List' problem. The code includes a struct for a linked list node and a class Solution with a printList method. The method iterates through the linked list and prints each node's data on the same line.
- Footer:** Includes a 'Custom Input' field and buttons for 'Compile & Run' and 'Submit'.

## 2. Remove duplicates from Sorted List

Code:

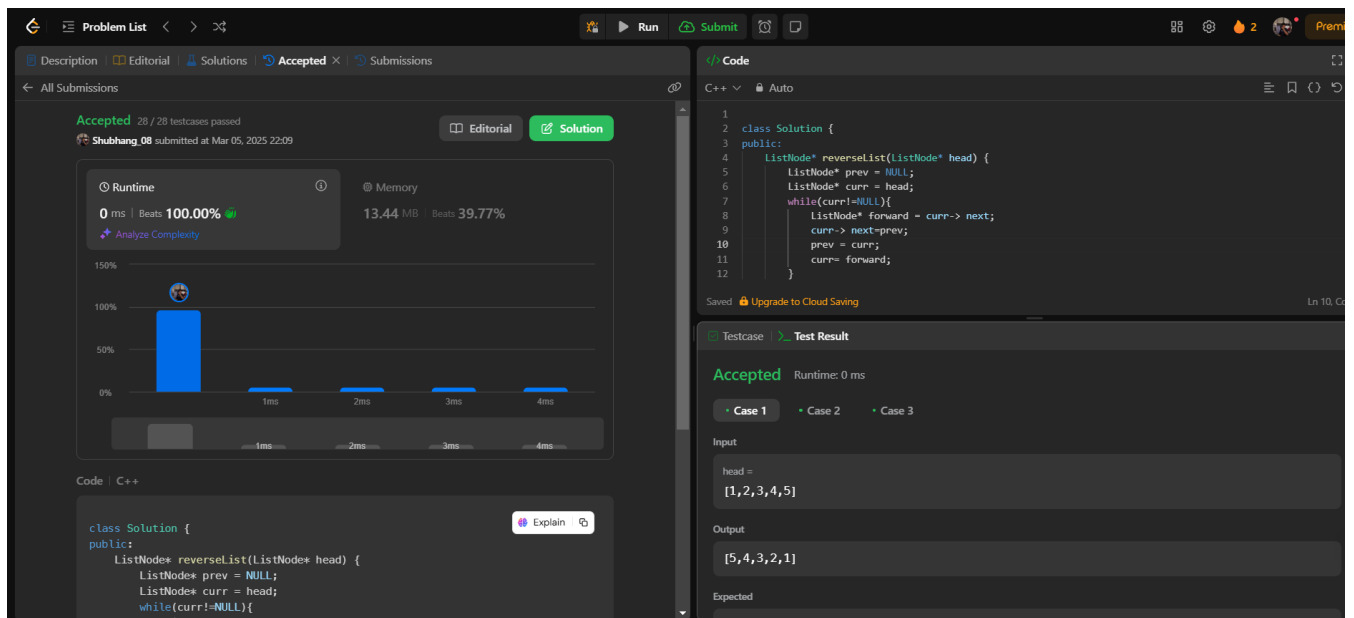
```
class Solution {  
    public ListNode deleteDuplicates(ListNode head) {  
  
        ListNode dummy = new ListNode(-1);  
  
        ListNode temp = head;  
  
        ListNode curr = dummy;  
  
        HashMap<Integer, Boolean> mp = new HashMap<>();  
  
        while(temp != null) {  
  
            if(!mp.containsKey(temp.val)) {  
  
                mp.put(temp.val, true);  
  
                curr.next = new ListNode(temp.val);  
  
                curr = curr.next;  
  
            }  
  
            temp = temp.next;  
  
        }  
  
        return dummy.next;  
  
    }  
}
```



### 3. Reverse a Linked List

Code:

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

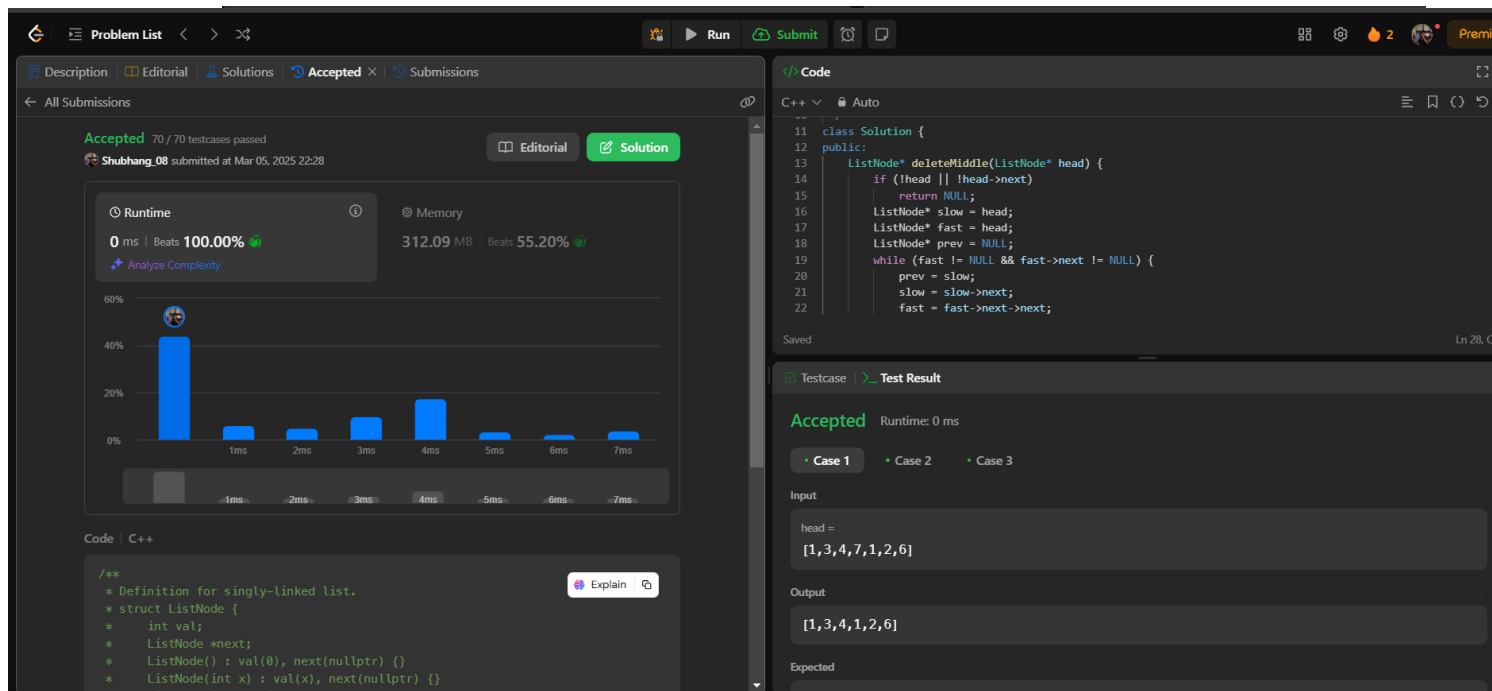


#### 4. Delete middle of a Linked List

Code:

```
class Solution {
public:
    ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) {
            return null;
        }
        ListNode slow = head;
        ListNode fast = head;
        ListNode prev = null;

        while(fast != null && fast.next != null) {
            prev = slow;
            slow = slow.next;
            fast = fast.next.next;
        }
        prev.next = slow.next;
        return head;
    }
}
```

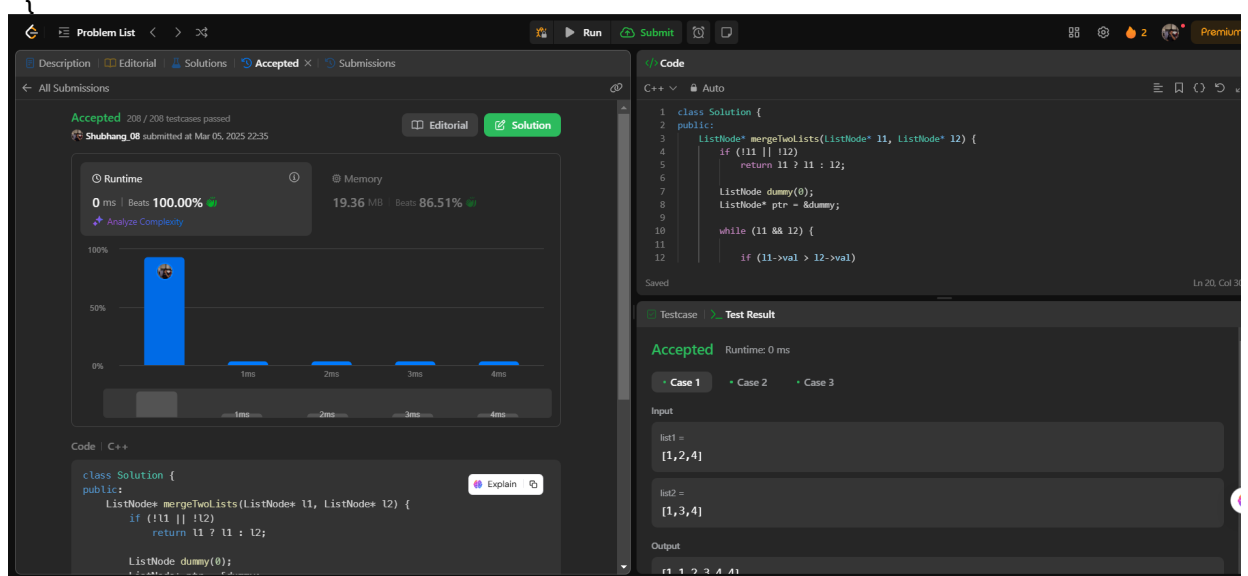


## 5. Merge Two sorted Linked List

Code:

```
class Solution {
public:
    ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1);
        ListNode curr = dummy;
        ListNode temp1 = list1;
        ListNode temp2 = list2;

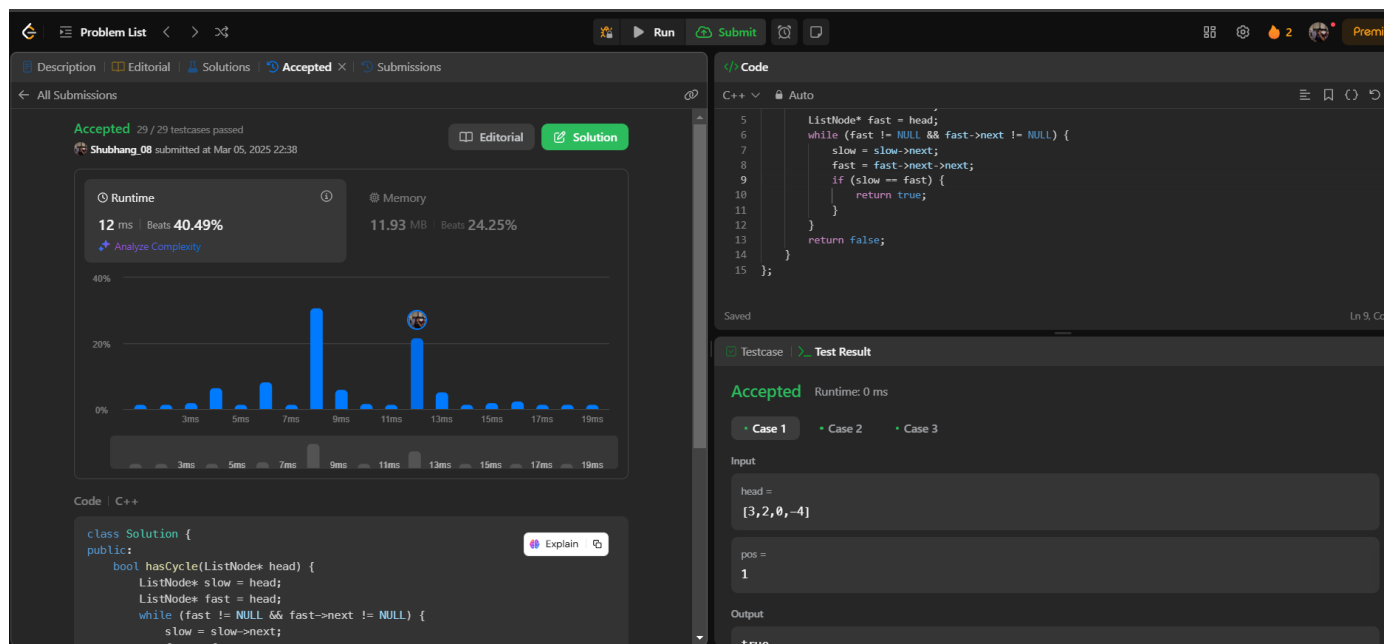
        while(temp1 != null && temp2 != null) {
            if(temp1.val <= temp2.val) {
                curr.next = temp1;
                temp1 = temp1.next;
            } else {
                curr.next = temp2;
                temp2 = temp2.next;
            }
            curr = curr.next;
        }
        if(temp1 != null) {
            curr.next = temp1;
        } else {
            curr.next = temp2;
        }
        return dummy.next;
    }
}
```



## 6. Detect a cycle in a Linked List

Code:

```
public class Solution {  
    public boolean hasCycle(ListNode head) {  
        ListNode slow = head;  
        ListNode fast = head;  
        while(fast != null && fast.next != null) {  
            slow = slow.next;  
            fast = fast.next.next;  
            if(slow == fast) {  
                return true;  
            }  
        }  
        return false;  
    }  
}
```



## 7. Rotate a Linked List

Code:

```
class Solution {
public:
    ListNode findNthNode(ListNode head, int k) {
        ListNode temp = head;
        int cnt = 1;
        while(temp != null) {
            if(cnt == k) return temp;
            cnt++;
            temp = temp.next;
        }
        return temp;
    }

    ListNode rotateRight(ListNode head, int k) {
        if(head == null || k == 0) return head;
        ListNode tail = head;
        int len = 1;
        while(tail.next != null) {
            tail = tail.next;
            len++;
        }
        if(k % len == 0) return head;
        k = k % len;
        tail.next = head;
        ListNode newLastNode = findNthNode(head, len - k);
        head = newLastNode.next;
        newLastNode.next = null;
        return head;
    }
}
```

The screenshot displays a coding platform interface with a dark theme. The top navigation bar includes 'Problem List', 'Run', 'Submit', and 'Premium' (with a 2-hour timer). The main content area is divided into three sections:

- Accepted:** Shows '232 / 232 testcases passed' and 'Shubhang\_08 submitted at Mar 05, 2025 22:41'. It includes a performance graph with a blue bar at 100% runtime and 16.34 MB memory. Below the graph is a table with columns for runtime (0ms) and memory (16.34 MB).
- Code:** Displays the C++ solution for the 'rotateRight' function. The code is as follows:

```
1 class Solution {
2 public:
3     ListNode* rotateRight(ListNode* head, int k) {
4         if(head==NULL || head->next==NULL) return head;
5         int n=0;
6         ListNode* temp=head;
7         ListNode* tail=NULL;
8         while(temp!=NULL){
9             if(temp->next==NULL) tail=temp;
10            temp=temp->next;
11            n++;
12        }
13    }
14 }
```
- Testcase:** Shows 'Case 1' with input 'head = [1,2,3,4,5]' and 'k = 2'.

The bottom of the interface shows the C++ code editor with the same solution code.

## 8. Sort List

Code:

```
class Solution {
    public ListNode merge(ListNode first, ListNode second) {
        ListNode t1 = first;
        ListNode t2 = second;
        ListNode dNode = new ListNode(-1);
        ListNode temp = dNode;
        while(t1 != null && t2 != null) {
            if(t1.val > t2.val) {
                temp.next = t2;
                temp = t2;
                t2 = t2.next;
            } else {
                temp.next = t1;
                temp = t1;
                t1 = t1.next;
            }
        }

        if(t1 != null) {
            temp.next = t1;
        } else {
            temp.next = t2;
        }

        return dNode.next;
    }

    public ListNode findmiddle(ListNode head) {
        ListNode slow = head;
        ListNode fast = head.next;

        if(head == null || head.next == null) return head;

        while(fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        return slow;
    }
}
```



```
public ListNode sortList(ListNode head) {
    if(head == null || head.next == null) return head;
```

```
    ListNode middle = findmiddle(head);
    ListNode righthand = middle.next;
    middle.next = null;
    ListNode lefthead = head;
```

```
    lefthead = sortList(lefthead);
    righthand = sortList(righthand);
    return merge(lefthead, righthand);
```

```
}
}
```

**Accepted** 30 / 30 testcases passed  
 Shubhang\_08 submitted at Mar 05, 2025 22:43

**Runtime** 54 ms | Beats 21.08%  
**Memory** 75.70 MB | Beats 35.90%

**Code** C++

```
1 class Solution {
2 public:
3     ListNode *middle(ListNode *head)
4     {
5         ListNode * slow=head;
6         ListNode * fast=head->next;
7         while(fast!=NULL&&fast->next!=NULL)
8         {
9             slow=slow->next;
10            fast=fast->next->next;
11        }
12    }
```

**Testcase** Case 1 Case 2 Case 3 +

head =

[4, 2, 1, 3]

**Code** C++

```
class Solution {
public:
    ListNode *middle(ListNode *head)
    {
        ListNode * slow=head;
        ListNode * fast=head->next;
        while(fast!=NULL&&fast->next!=NULL)
        {
```

## 9. Merge K Sorted List

Code:

```
import java.util.List;
```

```
class Solution {
```

```
    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {
```

```
        if (l1 == null) return l2;
```

```
        if (l2 == null) return l1;
```

```
        if (l1.val < l2.val) {
```

```
            l1.next = mergeTwoLists(l1.next, l2);
```

```
            return l1;
```

```
        } else {
```

```
            l2.next = mergeTwoLists(l1, l2.next);
```

```
            return l2;
```

```
        }
```

```
    }
```

```
    public ListNode mergeKLists(ListNode[] lists) {
```

```
        if (lists.length == 0) return null;
```

```
        return divideAndConquer(lists, 0, lists.length - 1);
```

```
    }
```

```
    private ListNode divideAndConquer(ListNode[] lists, int left, int right) {
```

```
        if (left == right) return lists[left];
```

```
        int mid = left + (right - left) / 2;
```

```
        ListNode l1 = divideAndConquer(lists, left, mid);
```

```
        ListNode l2 = divideAndConquer(lists, mid + 1, right);
```

```
        return mergeTwoLists(l1, l2);
```

```
    }
```

```
}
```

The screenshot displays the LeetCode submission interface for the 'Merge K Sorted List' problem. The top section shows the problem description and a code editor with a C++ solution. The solution is accepted, with a runtime of 3 ms and memory usage of 23.98 MB. The test result panel shows the input lists = [[1,4,5], [1,3,4], [2,6]] and the output [1,1,2,3,4,4,5,6].

**Runtime:** 3 ms | Beats 64.83%  
**Memory:** 23.98 MB | Beats 5.82%

**Code:**

```
class Solution {
public:
    ListNode* merge(ListNode *left, ListNode *right) {
        ListNode *dummy = new ListNode(-1);
        ListNode *temp = dummy;
        while (left != nullptr && right != nullptr) {
            if (left->val < right->val) {
                temp->next = left;
                temp = temp->next;
                left = left->next;
            } else {
                temp->next = right;
                temp = temp->next;
                right = right->next;
            }
        }
        if (left != nullptr) temp->next = left;
        if (right != nullptr) temp->next = right;
        return dummy->next;
    }

    ListNode* mergeKLists(ListNode* lists) {
        if (lists.empty()) return nullptr;
        return divideAndConquer(lists, 0, lists.size() - 1);
    }

    private:
    ListNode* divideAndConquer(ListNode* lists, int left, int right) {
        if (left == right) return lists[left];
        int mid = left + (right - left) / 2;
        ListNode* l1 = divideAndConquer(lists, left, mid);
        ListNode* l2 = divideAndConquer(lists, mid + 1, right);
        return merge(l1, l2);
    }
};
```

**Testcase 1: Accepted** Runtime: 0 ms

**Case 1:**

Input: lists = [[1,4,5], [1,3,4], [2,6]]

Output: [1,1,2,3,4,4,5,6]

Expected: [1,1,2,3,4,4,5,6]