



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

Assignment 3

Student Name: Lakshyaraj Singh
Branch: B.E CSE
Semester: 6th
Subject Name: Advanced Programming

UID: 22BCS10687
Section/Group: 22BCS-610/B
Date of Performance: 27/02.25
Subject Code: 22CSP-351

1. Aim: Print Linked List

2. Code:

```
class Solution {  
    void printList(Node head) {  
        Node temp = head;  
        while (temp != null) {  
            System.out.print(temp.data + " ");  
            temp = temp.next;  
        }  
        System.out.println();  
    }  
}
```

3. Output:

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

- Problem Status:** Problem Solved Successfully
- Test Cases Passed:** 1112 / 1112
- Attempts:** 1 / 5
- Accuracy:** 20%
- Points Scored:** 1 / 1
- Time Taken:** 2.01
- Your Total Score:** 11
- Solve Next:** Count Linked List Nodes, Delete Alternate Nodes, Insert in Middle of Linked List
- Code Editor:** Java (1.8) with the following code:

```
1 // Driver Code ends  
51  
52  
53 /* Node is defined as  
54 class Node {  
55     int data;  
56     Node next;  
57     Node(int x) {  
58         data = x;  
59         next = null;  
60     }  
61 }*/  
62  
63 Print elements of a linked list on console  
64 Head pointer input could be NULL as well for empty list  
65 */  
66  
67 class Solution {  
68     void printList(Node head) {  
69         Node temp = head;  
70         while(temp != null){  
71             System.out.print(temp.data+" ");  
72             temp = temp.next;  
73         }  
74     }  
75 }  
76  
77  
78  
79
```

1. Aim: Remove duplicates from a sorted list

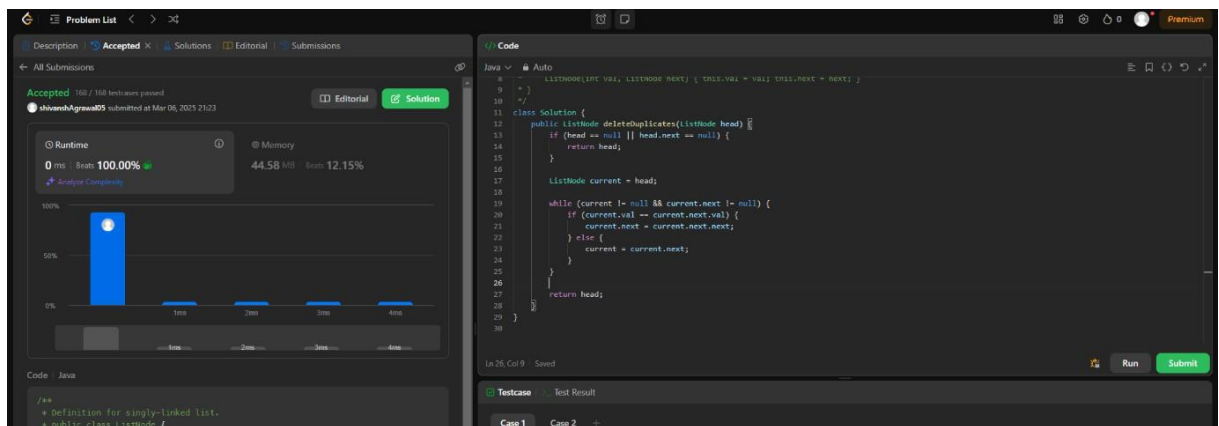
2. Code:

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode current = head;

        while (current != null && current.next != null) {
            if (current.val == current.next.val) {
                current.next = current.next.next;
            } else {
                current = current.next;
            }
        }

        return head;
    }
}
```

3. Output:



1. Aim: Reverse a linked list

2. Code:

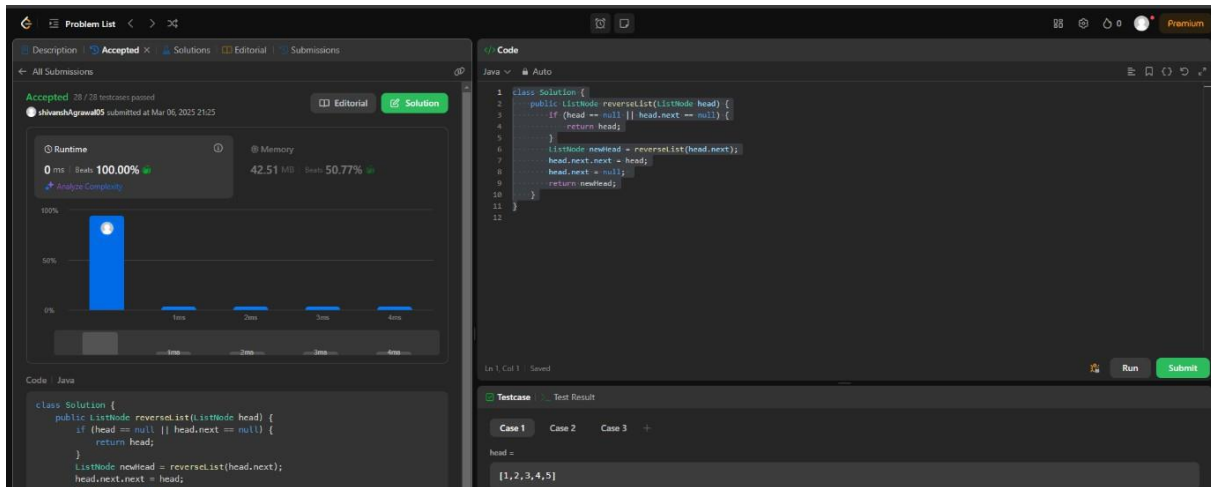
```
class Solution {
    public ListNode reverseList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode newHead = reverseList(head.next);
        head.next.next = head;
    }
}
```

```

        head.next = null;
        return newHead;
    }
}

```

3. Output:



1. Aim: Delete middle node of a list

2. Code:

```

class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if (head == null || head.next == null) return null;

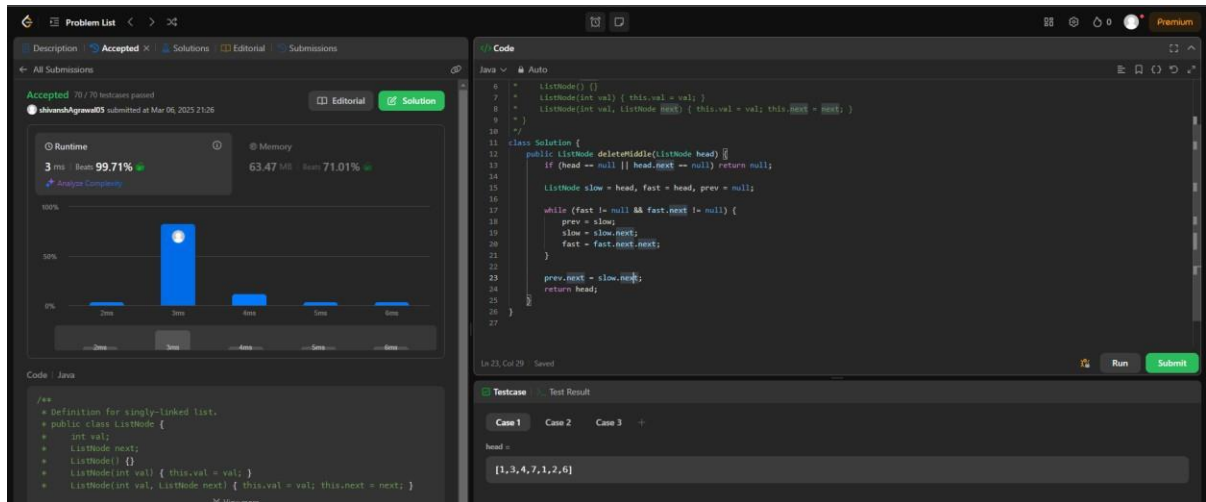
        ListNode slow = head, fast = head, prev = null;

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

        prev.next = slow.next;
        return head;
    }
}

```

3. Output:



1. Aim: Merge two sorted linked lists:

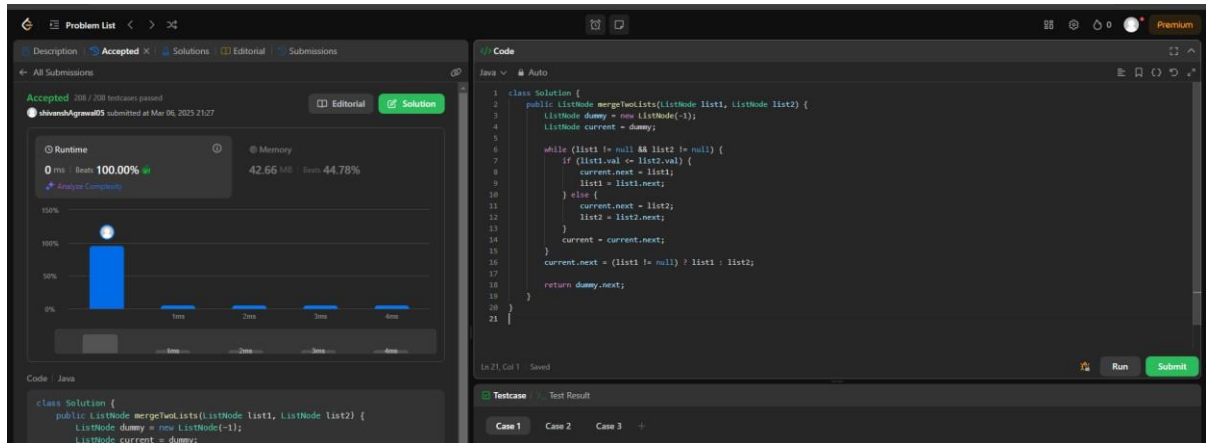
2. Code:

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode(-1);
        ListNode current = dummy;

        while (list1 != null && list2 != null) {
            if (list1.val <= list2.val) {
                current.next = list1;
                list1 = list1.next;
            } else {
                current.next = list2;
                list2 = list2.next;
            }
            current = current.next;
        }
        current.next = (list1 != null) ? list1 : list2;

        return dummy.next;
    }
}
```

3. Output:



1. Aim: Detect a cycle in a linked list

2. Code:

```

public class Solution {
    public boolean hasCycle(ListNode head) {
        if (head == null || head.next == null) return false;
    
```

```

        ListNode slow = head;
        ListNode fast = head;
    
```

```

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

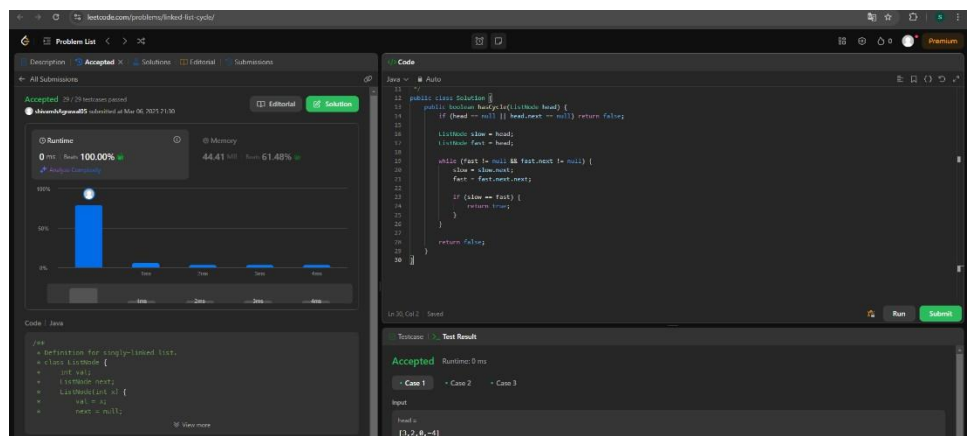
```

        if (slow == fast) {
            return true;
        }
    
```

```

        return false;
    }
}
    
```

3. Output:



1. Aim: Rotate a list

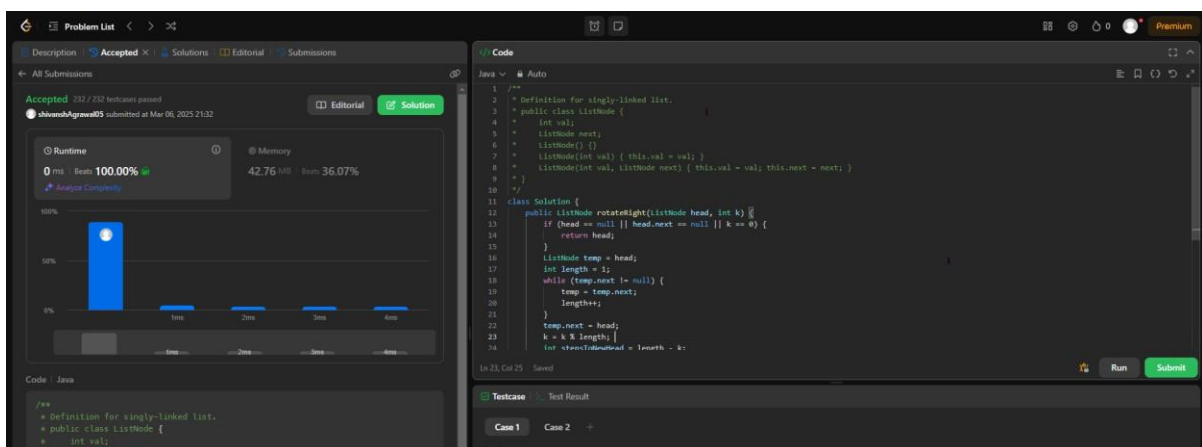
2. Code:

```
class Solution {
public ListNode rotateRight(ListNode head, int k) {
    if (head == null || head.next == null || k == 0) {
        return head;
    }
    ListNode temp = head;
    int length = 1;
    while (temp.next != null) {
        temp = temp.next;
        length++;
    }
    temp.next = head;
    k = k % length;
    int stepsToNewHead = length - k;
    ListNode newTail = head;

    for (int i = 1; i < stepsToNewHead; i++) {
        newTail = newTail.next;
    }
    head = newTail.next;
    newTail.next = null;

    return head;
}
}
```

3. Output:



1. Aim: Sort List

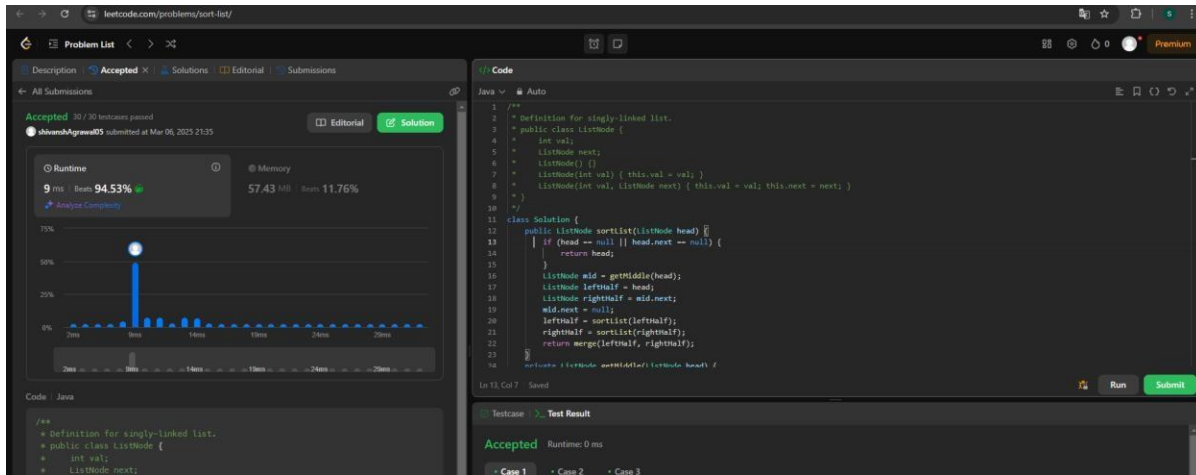
2. Code:

```
class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) {
            return head;
        }
        ListNode mid = getMiddle(head);
        ListNode leftHalf = head;
        ListNode rightHalf = mid.next;
        mid.next = null;
        leftHalf = sortList(leftHalf);
        rightHalf = sortList(rightHalf);
        return merge(leftHalf, rightHalf);
    }
    private ListNode getMiddle(ListNode head) {
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
    private ListNode merge(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                tail.next = l1;
                l1 = l1.next;
            } else {
                tail.next = l2;
                l2 = l2.next;
            }
            tail = tail.next;
        }
        if (l1 != null) tail.next = l1;
        if (l2 != null) tail.next = l2;

        return dummy.next;
    }
}
```

3. Output:



1. Aim: Merge k sorted list

2. Code:

```
import java.util.PriorityQueue;
```

```

class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) {
            return null;
        }
        PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) -> a.val - b.val);
        for (ListNode list : lists) {
            if (list != null) {
                minHeap.add(list);
            }
        }
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;
        while (!minHeap.isEmpty()) {
            ListNode smallest = minHeap.poll();
            tail.next = smallest;
            tail = tail.next;

            if (smallest.next != null) {
                minHeap.add(smallest.next);
            }
        }
    }
}

```




DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

return dummy.next;

```
}  
}
```

3. Output:

