

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;  
        }  
    }  
}
```

Print Linked List

Difficulty: Basic Accuracy: 60.71% Submissions: 142K+ Points: 1

Given a linked list. Print all the elements of the linked list separated by space followed.

Examples:

Input: LinkedList : 1 -> 2

Output Window

Compilation Results Custom Input Y.O.G.I. (AI Bot)

Test Cases Passed: 1112 / 1112

Attempts: Correct / Total: 1 / 1

Accuracy: 100%

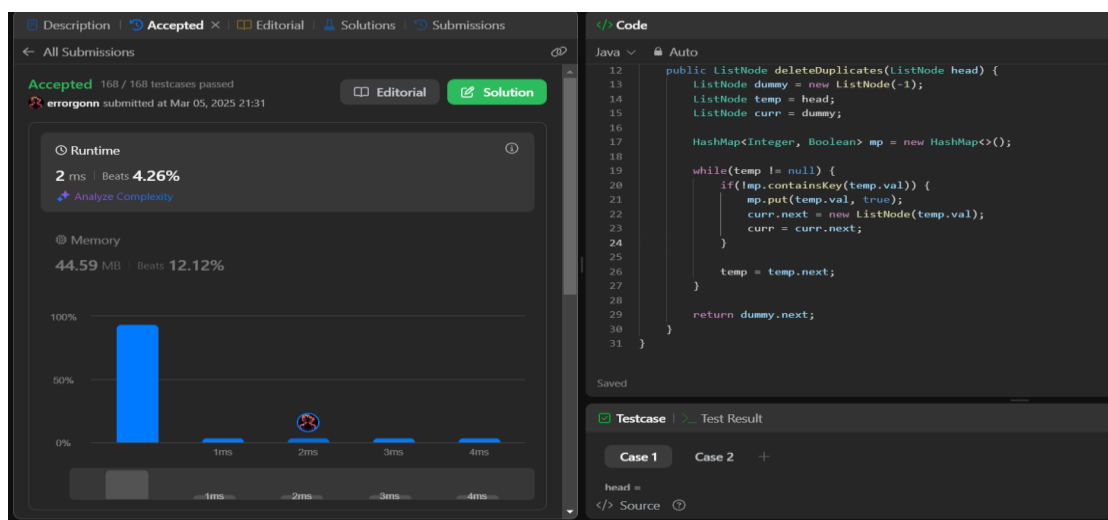
```
1 // Driver Code Ends  
51  
52  
53 /* Node is defined as  
54 class Node {  
55     int data;  
56     Node next;  
57     Node(int x) {  
58         data = x;  
59         next = null;  
60     }  
61 }*/  
62  
63 // Print elements of a linked list on console  
64 // Head pointer input could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

Custom Input Compile & Run Submit

2. Remove duplicates from Sorted List

Code:

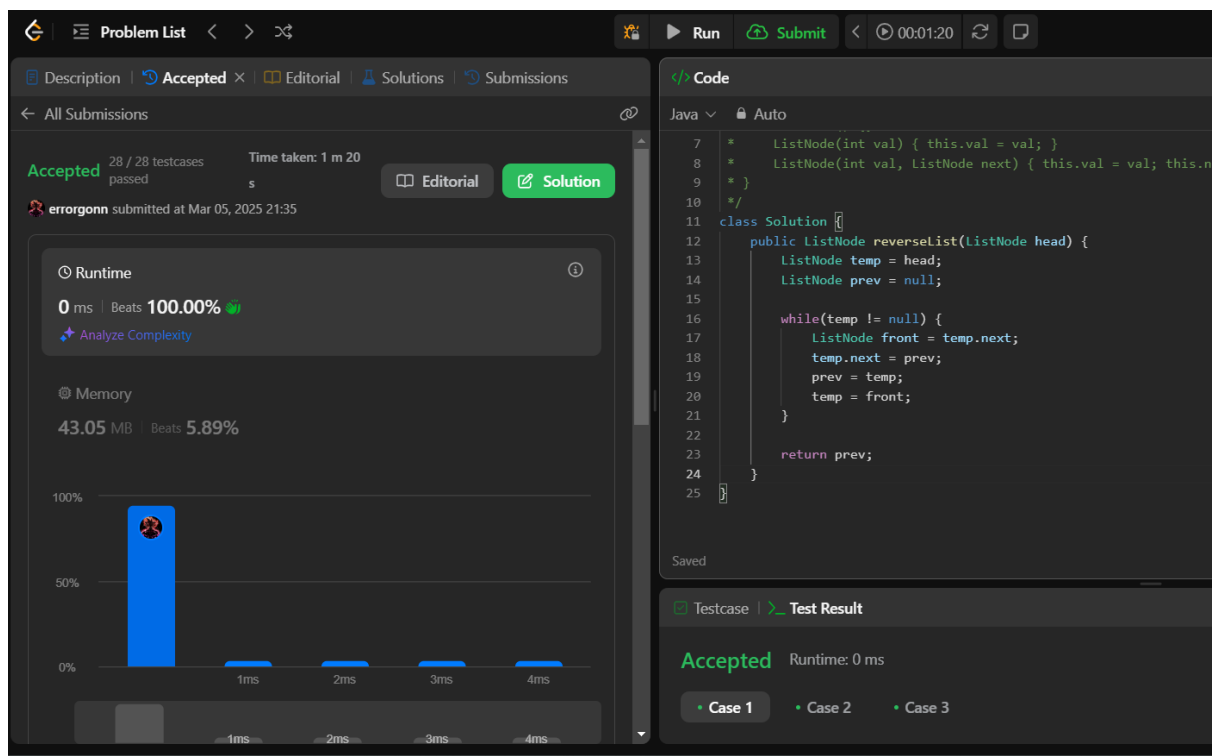
```
class Solution {  
  
    public ListNode deleteDuplicates(ListNode head) {  
  
        ListNode dummy = new ListNode(-1);  
  
        ListNode temp = head;  
  
        ListNode curr = dummy;  
  
        HashMap<Integer, Boolean> mp = new HashMap<>();  
  
        while(temp != null) {  
  
            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;  
    }  
}
```

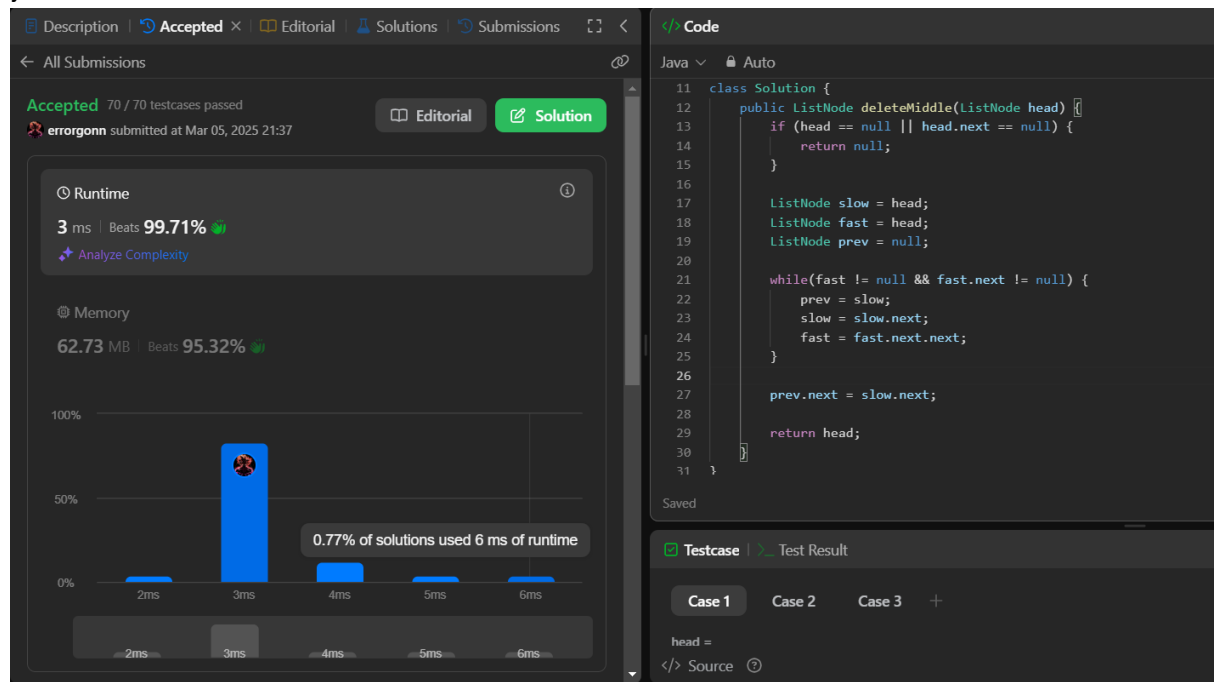


4. Delete middle of a Linked List

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;
    }
}
```

The screenshot displays a code editor interface for a submission. On the left, a sidebar shows the submission status: 'Accepted' with '208 / 208 testcases passed'. Below this, the 'Runtime' section indicates '0 ms' and 'Beats 100.00%'. The 'Memory' section shows '42.52 MB' and 'Beats 57.98%'. On the right, the code editor shows the Java code for the 'mergeTwoLists' method, which is identical to the code provided in the previous block. The code is written in Java and uses a dummy node to merge two sorted linked lists.

```
8 *   ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9 * }
10 */
11 class Solution {
12     public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
13         ListNode dummy = new ListNode(-1);
14         ListNode curr = dummy;
15         ListNode temp1 = list1;
16         ListNode temp2 = list2;
17
18         while(temp1 != null && temp2 != null) {
19             if(temp1.val <= temp2.val) {
20                 curr.next = temp1;
21                 temp1 = temp1.next;
22             } else {
23                 curr.next = temp2;
24                 temp2 = temp2.next;
25             }
26             curr = curr.next;
27         }
```

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;  
    }  
}
```

The screenshot shows a submission page on a coding platform. The top navigation bar includes links for Description, Accepted (29 / 29 testcases passed), Editorial, Solutions, and Submission. Below the navigation bar, there are tabs for Editorial and Solution. The Solution tab is active, showing the code for the 'hasCycle' method. The code is in Java and uses the Floyd's Cycle-Finding algorithm. The left sidebar displays the submission details: 'Accepted' status, '29 / 29 testcases passed', and a submission time of 'Mar 05, 2025 22:03'. Below this, the 'Runtime' section shows '0 ms' and 'Beats 100.00%', with a link to 'Analyze Complexity'. The 'Memory' section shows '44.51 MB' and 'Beats 49.23%'. A bar chart at the bottom of the sidebar shows the user's performance relative to other users, with a blue bar indicating a top performance.

Accepted 29 / 29 testcases passed
error... submitted at Mar 05, 2025 22:03

Runtime
0 ms | Beats 100.00%
[Analyze Complexity](#)

Memory
44.51 MB | Beats 49.23%

100%
50%

Code

```
12 public class Solution {  
13     public boolean hasCycle(ListNode head) {  
14         ListNode slow = head;  
15         ListNode fast = head;  
16  
17         while(fast != null && fast.next != null) {  
18             slow = slow.next;  
19             fast = fast.next.next;  
20  
21             if(slow == fast) {  
22                 return true;  
23             }  
24         }  
25  
26         return false;  
27     }  
28 }
```

Saved

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;
    }
    public 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;
    }
}
```

61. Rotate List Solved

Medium Topics Companies

Given the `head` of a linked list, rotate the list to the right by `k` places.

Example 1:

rotate 1

rotate 2

Input: head = [1,2,3,4,5], k = 2
Output: [4,5,1,2,3]

Example 2:

Testcase Test Result

Case 1 Case 2 +

head =

```
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;
    }
    public 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;
    }
}
```

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 righthhead = middle.next;
    middle.next = null;
    ListNode lefthead = head;

    lefthead = sortList(lefthead);
    righthhead = sortList(righthhead);
    return merge(lefthead, righthhead);
}
}

```

The screenshot shows a submission for a linked list sorting problem. The left sidebar displays the following performance metrics:

- Runtime:** 9 ms, Beats 94.61%
- Memory:** 57.28 MB, Beats 22.93%

The code editor on the right shows the following Java code:

```

51     return slow;
52 }
53
54 public ListNode sortList(ListNode head) {
55     if(head == null || head.next == null) return head;
56
57     ListNode middle = findmiddle(head);
58     ListNode righthhead = middle.next;
59     middle.next = null;
60     ListNode lefthead = head;
61
62     lefthead = sortList(lefthead);
63     righthhead = sortList(righthhead);
64     return merge(lefthead, righthhead);
65 }
66 }

```

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 page for the 'Merge K Sorted List' problem. On the left, the 'Runtime' section shows a time of 2 ms, beating 84.21% of submissions. The 'Memory' section shows a memory usage of 44.30 MB, beating 89.33% of submissions. A bar chart below these metrics shows the distribution of runtime and memory usage across all submissions. On the right, the code editor shows the Java solution. The code defines a `ListNode` class and implements the `mergeKLists` method using a divide-and-conquer approach. The `mergeTwoLists` method is used as a helper to merge two sorted lists. The submission status is 'Accepted' with 134/134 test cases passed. The submission was made on Mar 05, 2025 at 22:28. The bottom of the screen shows the 'Testcase' tab with 'Case 1' selected.