

PRINT LINKED LIST

The screenshot displays a coding platform interface. On the left, the 'Output Window' shows 'Compilation Results' for 'Custom Input' by 'Y.O.G.I. (AI Bot)'. It indicates 'Problem Solved Successfully' with a green checkmark. Performance metrics include: 'Test Cases Passed: 1112 / 1112', 'Attempts: Correct / Total: 1 / 1', 'Accuracy: 100%', 'Points Scored: 1 / 1', and 'Time Taken: 1.7'. A 'Solve Next' section offers links to 'Count Linked List Nodes', 'Delete Alternate Nodes', and 'Insert in Middle of Linked List'. On the right, the code editor shows a Java solution. The code defines a 'Node' class with 'data' and 'next' attributes, and a 'printList' method in the 'Solution' class that traverses the linked list and prints each node's data.

Output Window

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

Problem Solved Successfully ✓ Suggest Feedback

Test Cases Passed
1112 / 1112

Attempts: Correct / Total
1 / 1
Accuracy: 100%

Points Scored ⓘ
1 / 1
Your Total Score: 8 ↑

Time Taken
1.7

Solve Next

Count Linked List Nodes Delete Alternate Nodes Insert in Middle of Linked List

```
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         while(temp != null){
73             System.out.print(temp.data+" ");
74             temp = temp.next;
75         }
76     }
77 }
78
```

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+" ");

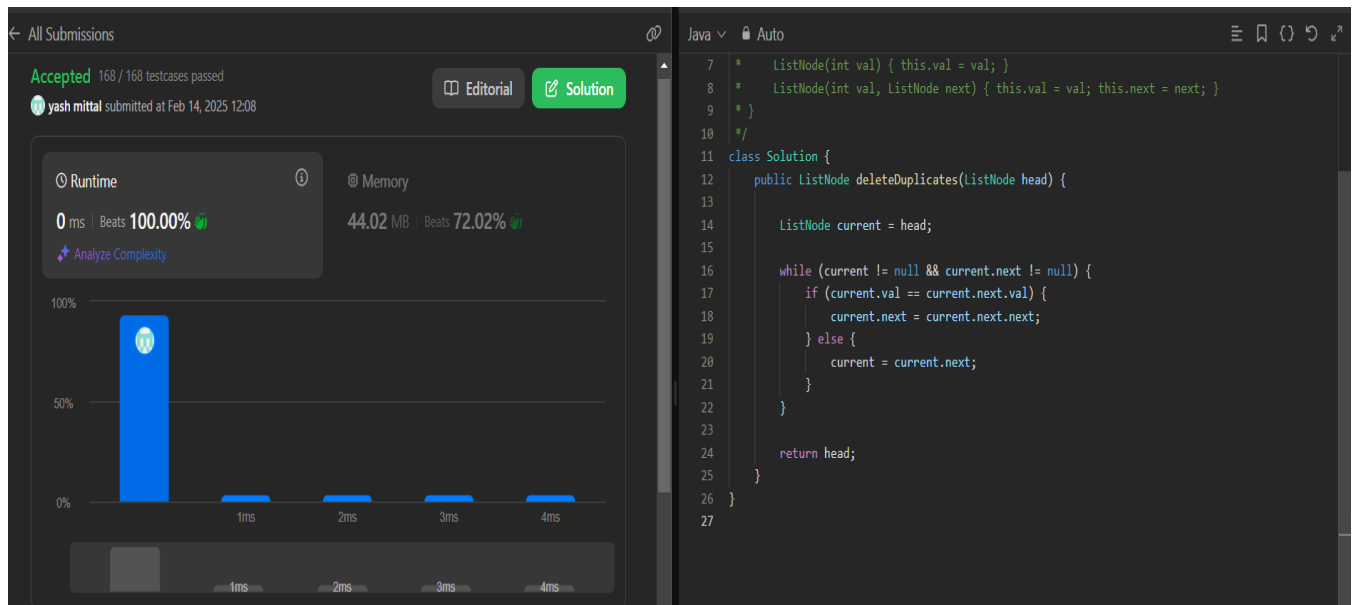
            temp = temp.next;

        }

    }

}
```

REMOVE DUPLICATE ELEMENTS FROM SORTED LIST



CODE:

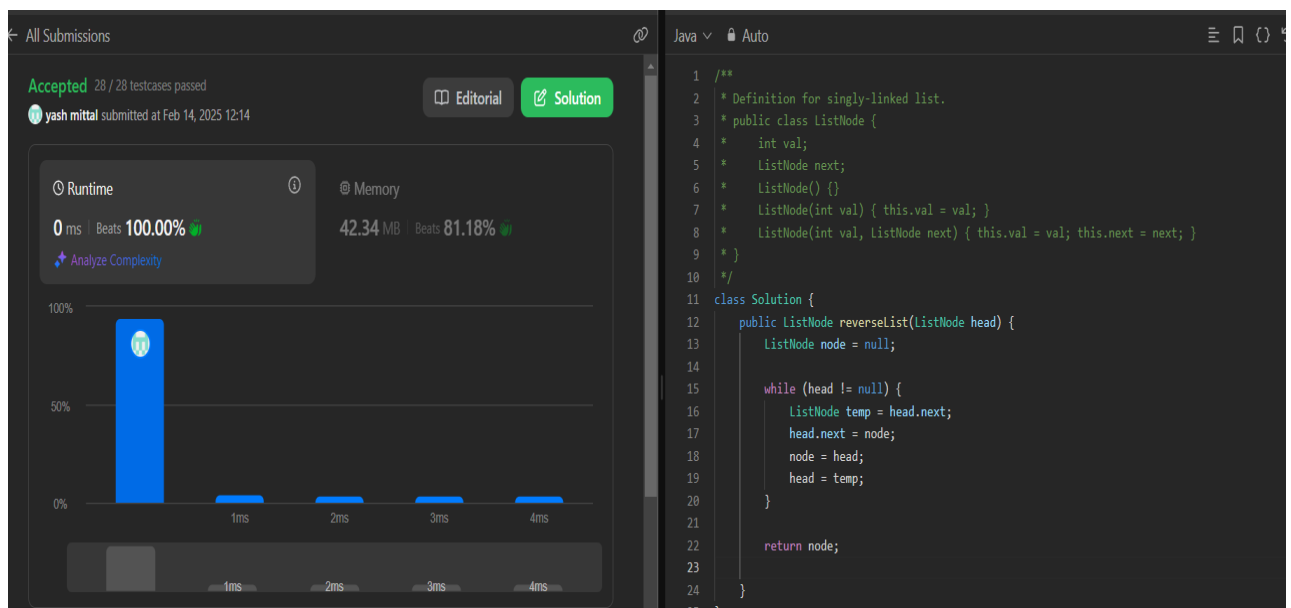
```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {

        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;
    }
}
```

REVERSE LINKED LIST



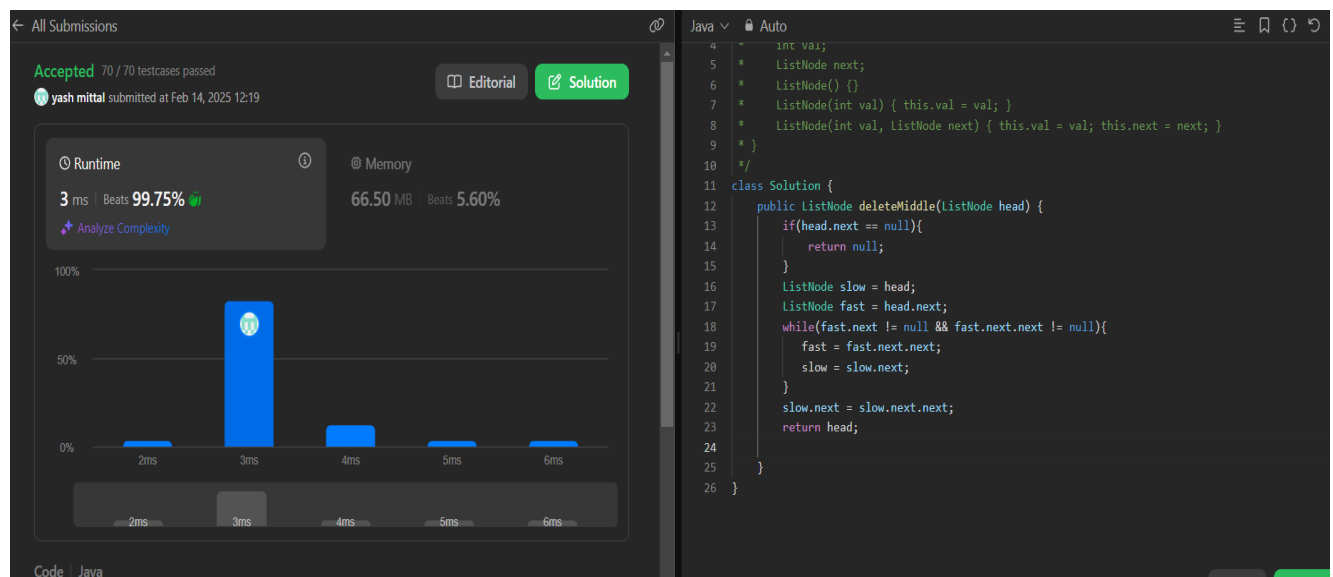
CODE:

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode node = null;

        while (head != null) {
            ListNode temp = head.next;
            head.next = node;
            node = head;
            head = temp;
        }

        return node;
    }
}
```

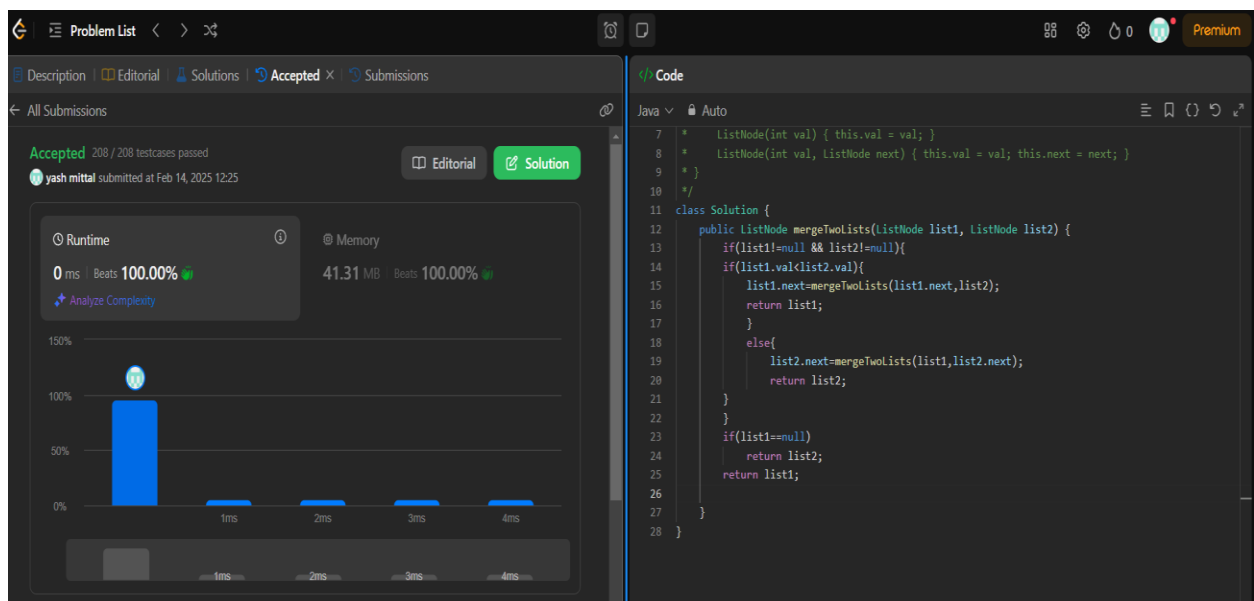
DELETE MIDDLE NODE OF THE LIST



CODE:

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

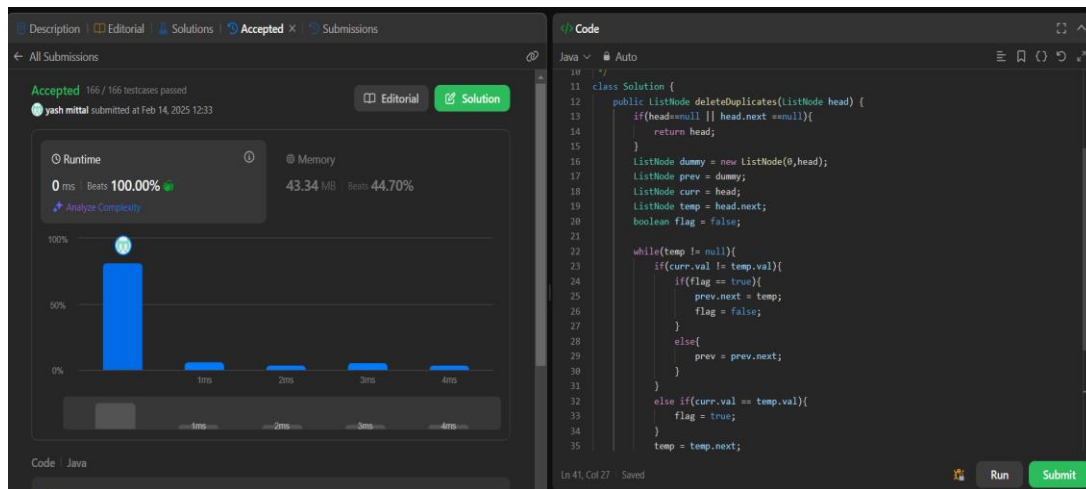
MERGE TWO SORTED LINKED LISTS



CODE:

```
class Solution {  
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {  
        if(list1!=null && list2!=null){  
            if(list1.val<list2.val){  
                list1.next=mergeTwoLists(list1.next,list2);  
                return list1;  
            }  
            else{  
                list2.next=mergeTwoLists(list1,list2.next);  
                return list2;  
            }  
        }  
        if(list1==null)  
            return list2;  
        return list1;  
    }  
}
```

REMOVE DUPLICATES FROM AN ELEMENT 2



CODE:

```
class Solution {

    public ListNode deleteDuplicates(ListNode head) {

        if(head==null || head.next ==null){

            return head;

        }

        ListNode dummy = new ListNode(0,head);

        ListNode prev = dummy;

        ListNode curr = head;

        ListNode temp = head.next;

        boolean flag = false;

        while(temp != null){

            if(curr.val != temp.val){

                if(flag == true){

                    prev.next = temp;

                    flag = false;

                }

                else{

                    prev = prev.next;

                }

            }

            else if(curr.val == temp.val){

                flag = true;

            }

            temp = temp.next;

        }

        return dummy.next;

    }

}
```

```

    }
}
else if(curr.val == temp.val){
    flag = true;
}
temp = temp.next;
curr = curr.next;
}
if(flag == true){
    prev.next = temp;
}
return dummy.next;
}
}

```

ROTATE LIST

The screenshot shows a code editor with a Java solution for rotating a linked list. The left panel displays submission statistics: Accepted, 232/232 testcases passed, Runtime 0 ms (Beats 100.00%), Memory 42.34 MB (Beats 89.19%). The right panel shows the Java code for the rotateRight method.

```

12 public ListNode rotateRight(ListNode head, int k) {
13     if (head == null || head.next == null || k == 0) {
14         return head;
15     }
16
17     int length = 1;
18     ListNode temp = head;
19
20     while (temp.next != null) {
21         temp = temp.next;
22         length++;
23     }
24
25     temp.next = head;
26     k = k % length;
27     k = length - k;
28
29     while (k-- > 0) {
30         temp = temp.next;
31     }
32
33     head = temp.next;
34     temp.next = null;
35
36     return head;
37 }

```

CODE:

```

class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if (head == null || head.next == null || k == 0) {
            return head;

```

```

    }

    int length = 1;

    ListNode temp = head;
    while (temp.next != null) {
        temp = temp.next;
        length++;
    }

    temp.next = head;

    k = k % length;

    k = length - k;

    while (k-- > 0) {
        temp = temp.next;
    }

    head = temp.next;

    temp.next = null;

    return head;

}
}

```

SORT LIST

Problem List < > 🔍

Description Editorial Solutions Accepted Submissions

← All Submissions

Accepted 30 / 30 testcases passed

yash mittal submitted at Feb 14, 2025 14:12

Editorial Solution

Runtime 9 ms | Beats 95.69%

Memory 57.14 MB | Beats 30.13%

Analyze Complexity

75%
50%
25%
0%

2ms 8ms 13ms 18ms 23ms 28ms

Code Java

Ln 56, Col 3 Saved Run Submit

```

11 class Solution {
12     public ListNode sortList(ListNode head) {
13         if (head == null || head.next == null)
14             return head;
15
16         // step 1. cut the list to two halves
17         ListNode prev = null, slow = head, fast = head;
18
19         while (fast != null && fast.next != null) {
20             prev = slow;
21             slow = slow.next;
22             fast = fast.next.next;
23         }
24
25         prev.next = null;
26
27         // step 2. sort each half
28         ListNode l1 = sortList(head);
29         ListNode l2 = sortList(slow);
30
31         // step 3. merge l1 and l2
32         return merge(l1, l2);
33     }
34
35     ListNode merge(ListNode l1, ListNode l2) {

```


CODE:

```
class Solution {  
    public ListNode sortList(ListNode head) {  
        if (head == null || head.next == null)  
            return head;  
  
        ListNode prev = null, slow = head, fast = head;  
  
        while (fast != null && fast.next != null) {  
            prev = slow;  
            slow = slow.next;  
            fast = fast.next.next;  
        }  
  
        prev.next = null;  
  
        // step 2. sort each half  
        ListNode l1 = sortList(head);  
        ListNode l2 = sortList(slow);  
  
        // step 3. merge l1 and l2  
        return merge(l1, l2);  
    }  
  
    ListNode merge(ListNode l1, ListNode l2) {  
        ListNode l = new ListNode(0), p = l;  
  
        while (l1 != null && l2 != null) {  
            if (l1.val < l2.val) {  
                p.next = l1;  
                l1 = l1.next;  
            } else {  
                p.next = l2;  
                l2 = l2.next;  
            }  
            p = p.next;  
        }  
    }  
}
```

```

    }

    if (l1 != null)
        p.next = l1;

    if (l2 != null)
        p.next = l2;

    return l.next;
}
}

```

DETECT THE CYCLE IN LINKED LIST II

The screenshot shows a coding platform interface with a Java solution for the problem "Detect the Cycle in Linked List II". The left sidebar displays submission statistics: "Accepted 18 / 18 testcases passed", "Runtime 0 ms | Beats 100.00%", and "Memory 44.64 MB | Beats 51.34%". A bar chart shows the runtime performance relative to other solutions. The right panel shows the Java code for the "detectCycle" method using Floyd's Cycle-Finding Algorithm.

```

class Solution {
    public ListNode detectCycle(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
            if (slow == fast) break;
        }
        if (fast == null || fast.next == null) return null;
        while (head != slow) {
            head = head.next;
            slow = slow.next;
        }
        return head;
    }
}

```

CODE:

```

public class Solution {

    public ListNode detectCycle(ListNode head) {

        ListNode slow = head, fast = head;

        while (fast != null && fast.next != null) {

            slow = slow.next;

            fast = fast.next.next;

            if (slow == fast) break;

        }

        if (fast == null || fast.next == null) return null;
    }
}

```

```
while (head != slow) {  
    head = head.next;  
    slow = slow.next;  
}  
return head;  
}  
}
```