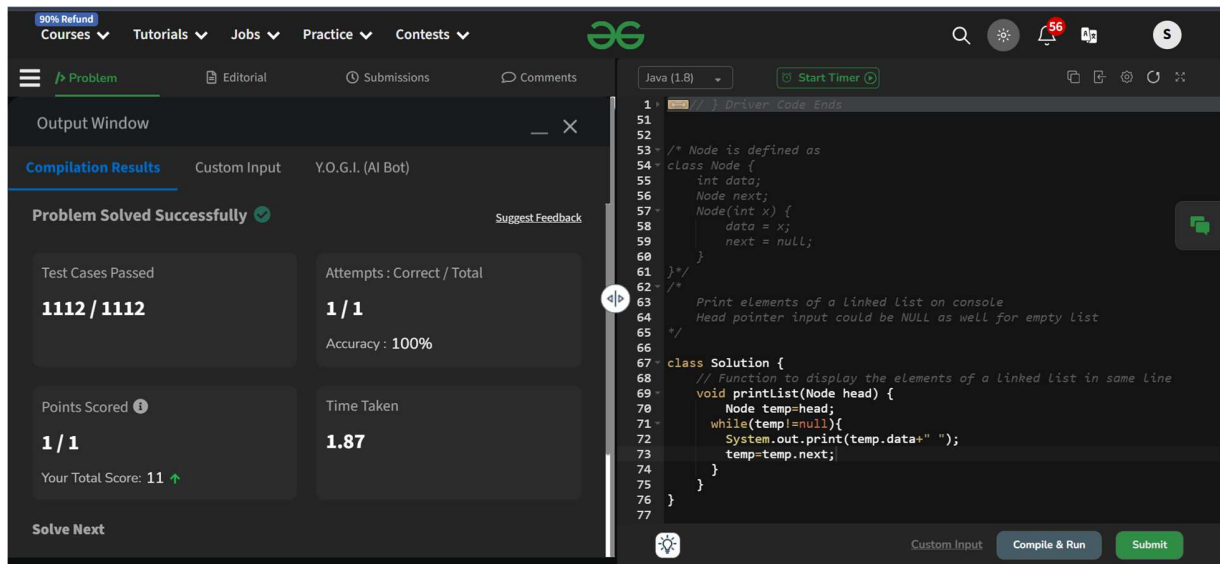


[Print Linked List](#)

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

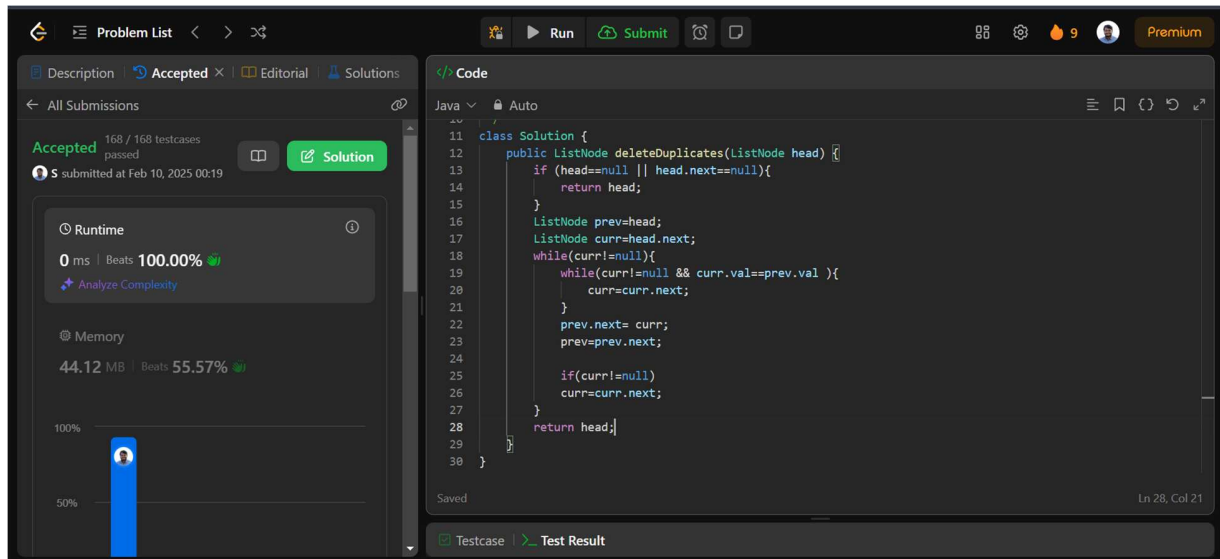


```
1  // Driver Code Ends
51
52
53  /* Node is defined as
54  class Node {
55      int data;
56      Node next;
57      Node(int x) {
58          data = x;
59          next = null;
60      }
61  }
62  */
63  /*
64  Print elements of a Linked list on console
65  Head pointer input could be NULL as well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

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

83. Remove Duplicates from Sorted List

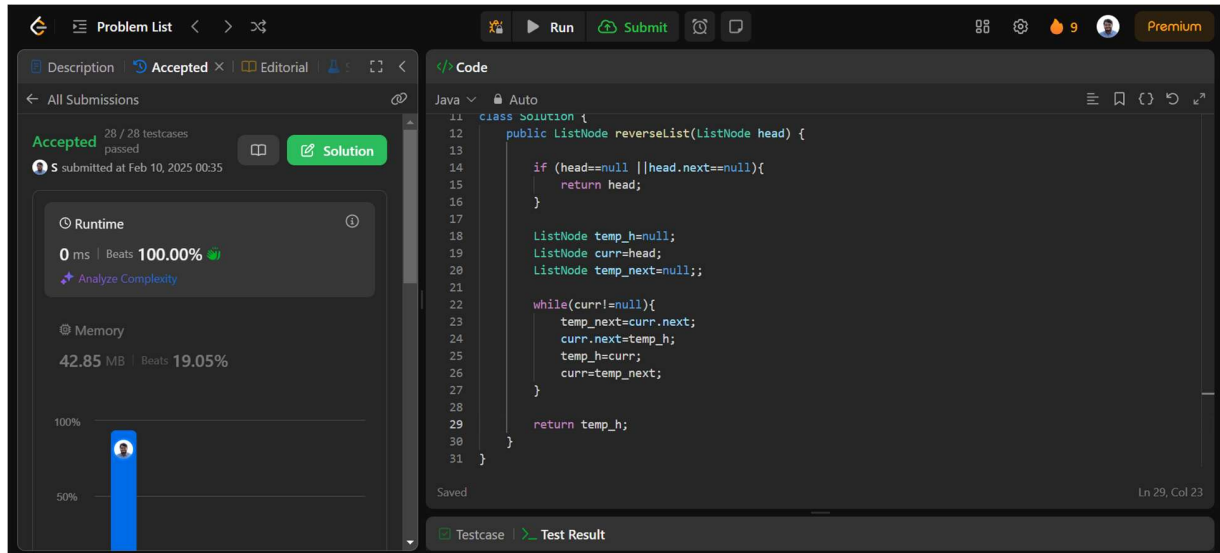
Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list **sorted** as well.



```
class Solution {  
    public ListNode deleteDuplicates(ListNode head) {  
        if (head==null || head.next==null) return head;  
        ListNode prev=head;  
        ListNode curr=head.next;  
        while(curr!=null){  
            while(curr!=null && curr.val==prev.val) curr=curr.next;  
            prev.next= curr;  
            prev=prev.next;  
            if(curr!=null) curr=curr.next;  
        }  
        return head;  
    }  
}
```

206. Reverse Linked List

Given the head of a singly linked list, reverse the list, and return *the reversed list*.



```
11 class Solution {
12     public ListNode reverseList(ListNode head) {
13
14         if (head==null || head.next==null){
15             return head;
16         }
17
18         ListNode temp_h=null;
19         ListNode curr=head;
20         ListNode temp_next=null;;
21
22         while(curr!=null){
23             temp_next=curr.next;
24             curr.next=temp_h;
25             temp_h=curr;
26             curr=temp_next;
27         }
28
29         return temp_h;
30     }
31 }
```

```
class Solution {

    public ListNode reverseList(ListNode head) {

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

            return head;

        }

        ListNode temp_h=null;

        ListNode curr=head;

        ListNode temp_next=null;;

        while(curr!=null){

            temp_next=curr.next;

            curr.next=temp_h;

            temp_h=curr;

            curr=temp_next;

        }

        return temp_h;

    }

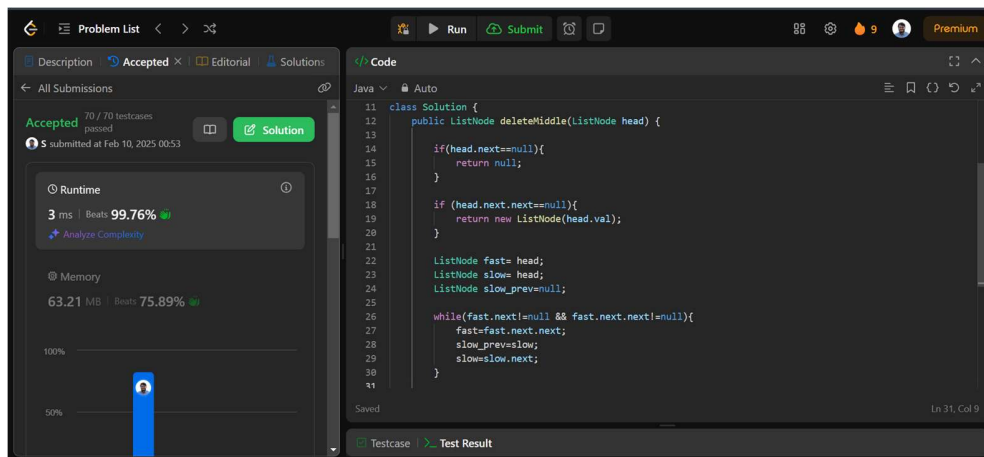
}
```

2095. Delete the Middle Node of a Linked List

You are given the head of a linked list. **Delete** the **middle node**, and return *the head of the modified linked list*.

The **middle node** of a linked list of size n is the $\lfloor n / 2 \rfloor^{\text{th}}$ node from the **start** using **0-based indexing**, where $\lfloor x \rfloor$ denotes the largest integer less than or equal to x .

- For $n = 1, 2, 3, 4$, and 5 , the middle nodes are $0, 1, 1, 2$, and 2 , respectively.



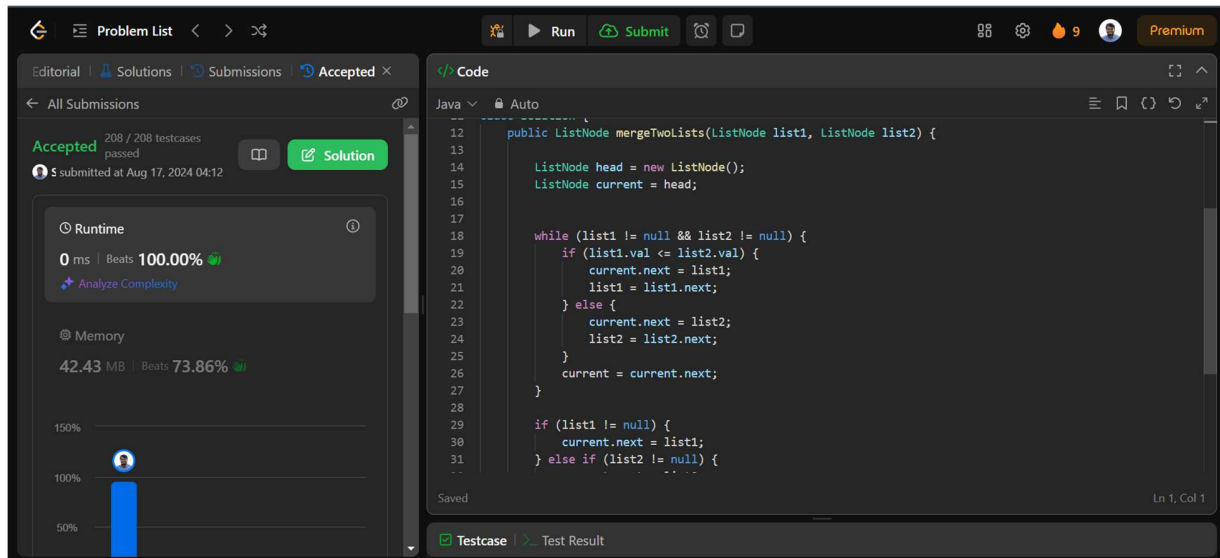
```
class Solution {  
    public ListNode deleteMiddle(ListNode head) {  
        if(head.next==null) return null;  
        if (head.next.next==null) return new ListNode(head.val);  
        ListNode fast= head;  
        ListNode slow= head;  
        ListNode slow_prev=null;  
        while(fast.next!=null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

21. Merge Two Sorted Lists

You are given the heads of two sorted linked lists list1 and list2.

Merge the two lists into one **sorted** list. The list should be made by splicing together the nodes of the first two lists.

Return *the head of the merged linked list*.



The screenshot shows a LeetCode editor interface. On the left, the 'Accepted' status is confirmed with '208 / 208 testcases passed' and a submission time of 'Aug 17, 2024 04:12'. The runtime is '0 ms' (Beats 100.00%) and memory is '42.43 MB' (Beats 73.86%). The main code editor on the right displays the following Java code:

```
12 public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
13
14     ListNode head = new ListNode();
15     ListNode current = head;
16
17
18     while (list1 != null && list2 != null) {
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
29     if (list1 != null) {
30         current.next = list1;
31     } else if (list2 != null) {
32         current.next = list2;
33     }
34 }
```

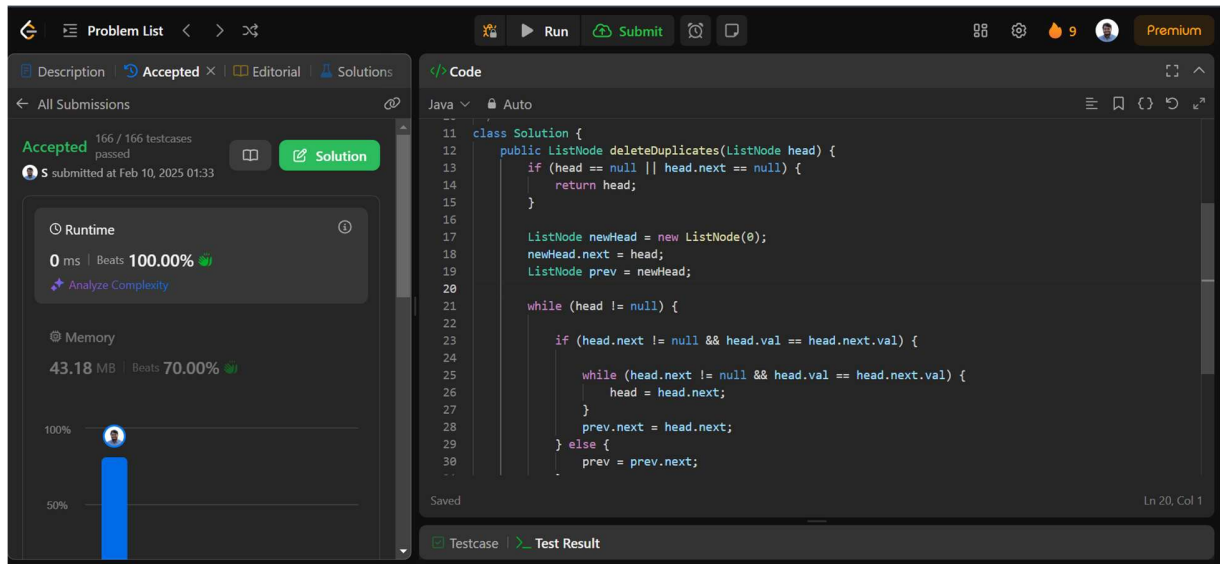
```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode head = new ListNode();
        ListNode current = head;

        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;
        }

        if (list1 != null) current.next = list1;
        else if (list2 != null) current.next = list2;
        return head.next;
    }
}
```

82. Remove Duplicates from Sorted List II

Given the head of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list*. Return the linked list **sorted** as well.



The screenshot shows the LeetCode submission page for the problem '82. Remove Duplicates from Sorted List II'. The left sidebar displays the submission status: 'Accepted' with 166/166 testcases passed, submitted on Feb 10, 2025 at 01:33. It also shows performance metrics: Runtime is 0 ms (Beats 100.00%) and Memory is 43.18 MB (Beats 70.00%). The main area shows the Java code for the solution, which uses a linked list structure to remove duplicates. The code is as follows:

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

        ListNode newHead = new ListNode(0);
        newHead.next = head;
        ListNode prev = newHead;

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

        return newHead.next;
    }
}
```

```
class Solution {

    public ListNode deleteDuplicates(ListNode head) {

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

        ListNode newHead = new ListNode(0);

        newHead.next = head;

        ListNode prev = newHead;

        while (head != null) {

            if (head.next != null && head.val == head.next.val) {

                while (head.next != null && head.val == head.next.val) head = head.next;

                prev.next = head.next;

            }

            else prev = prev.next;

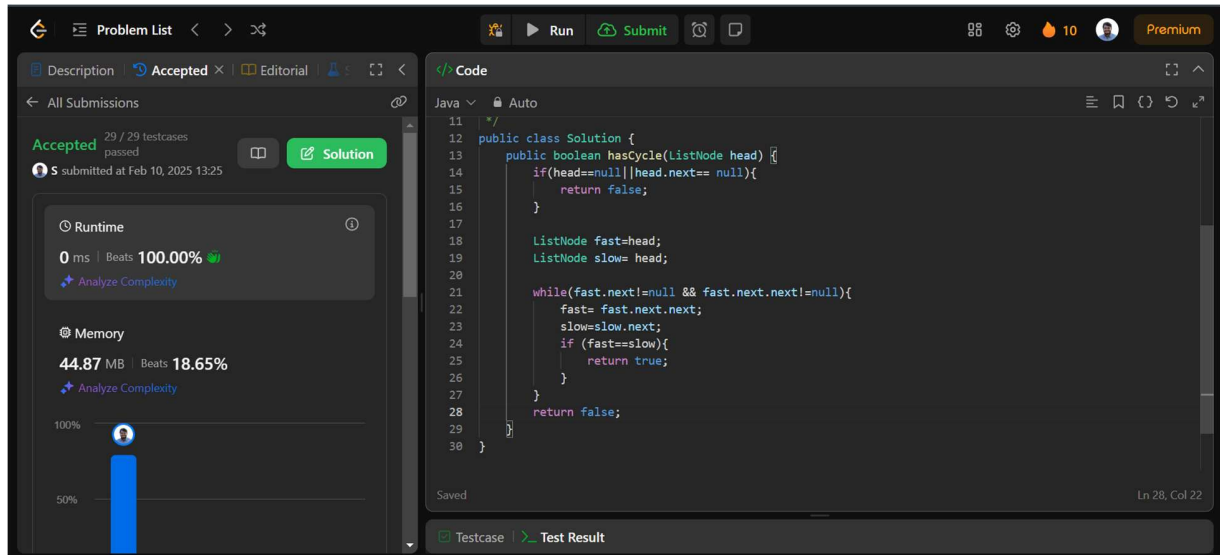
            head = head.next;

        }

        return newHead.next;
    }
}
```

141. Linked List Cycle

Given head, the head of a linked list, determine if the linked list has a cycle in it.

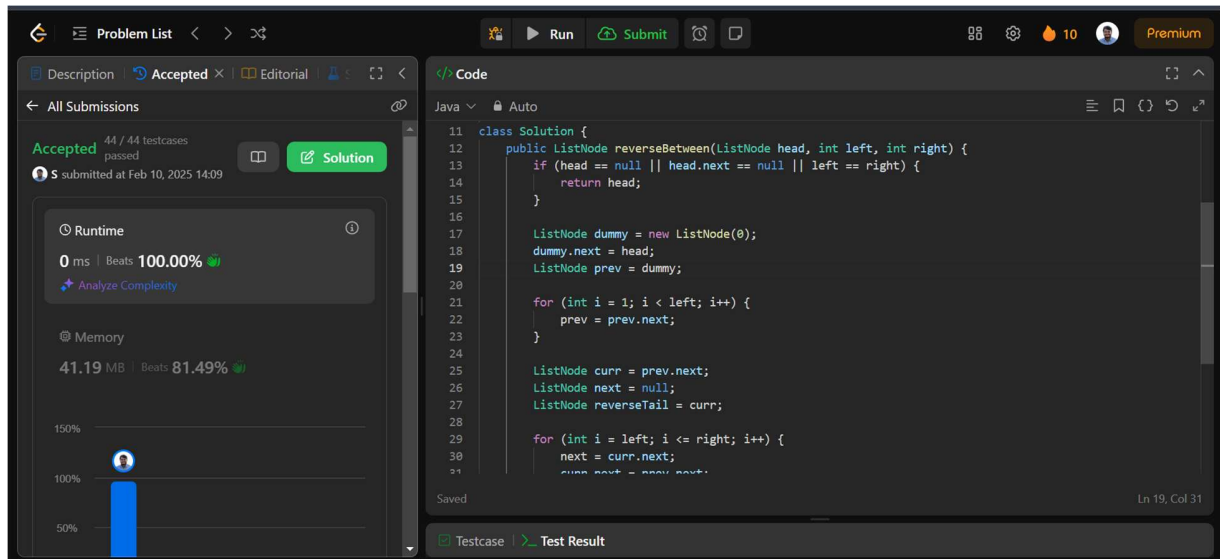


```
11  /*
12  public class Solution {
13      public boolean hasCycle(ListNode head) {
14          if(head==null || head.next== null){
15              return false;
16          }
17
18          ListNode fast=head;
19          ListNode slow= head;
20
21          while(fast.next!=null && fast.next.next!=null){
22              fast= fast.next.next;
23              slow=slow.next;
24              if (fast==slow){
25                  return true;
26              }
27          }
28          return false;
29      }
30  }
```

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

92. Reverse Linked List II

Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return *the reversed list*.



The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with 'All Submissions' and a 'Runtime' section showing '0 ms | Beats 100.00%'. The main area displays the code for the 'reverseBetween' method in a 'Solution' class. The code is in Java and uses a dummy node and two loops to reverse the list between the 'left' and 'right' positions. The bottom status bar indicates 'Ln 19, Col 31'.

```
11 class Solution {
12     public ListNode reverseBetween(ListNode head, int left, int right) {
13         if (head == null || head.next == null || left == right) {
14             return head;
15         }
16
17         ListNode dummy = new ListNode(0);
18         dummy.next = head;
19         ListNode prev = dummy;
20
21         for (int i = 1; i < left; i++) {
22             prev = prev.next;
23         }
24
25         ListNode curr = prev.next;
26         ListNode next = null;
27         ListNode reverseTail = curr;
28
29         for (int i = left; i <= right; i++) {
30             next = curr.next;
31             curr.next = prev.next;
```

```
class Solution {
    public ListNode reverseBetween(ListNode head, int left, int right) {
        if (head == null || head.next == null || left == right) {
            return head;
        }
        ListNode dummy = new ListNode(0);
        dummy.next = head;
        ListNode prev = dummy;
        for (int i = 1; i < left; i++) {
            prev = prev.next;
        }

        ListNode curr = prev.next;
        ListNode next = null;
        ListNode reverseTail = curr;
        for (int i = left; i <= right; i++) {
            next = curr.next;
            curr.next = prev.next;
            prev.next = curr;
            curr = next;
        }
        reverseTail.next = curr;
        return dummy.next;
    }
}
```


61. Rotate List

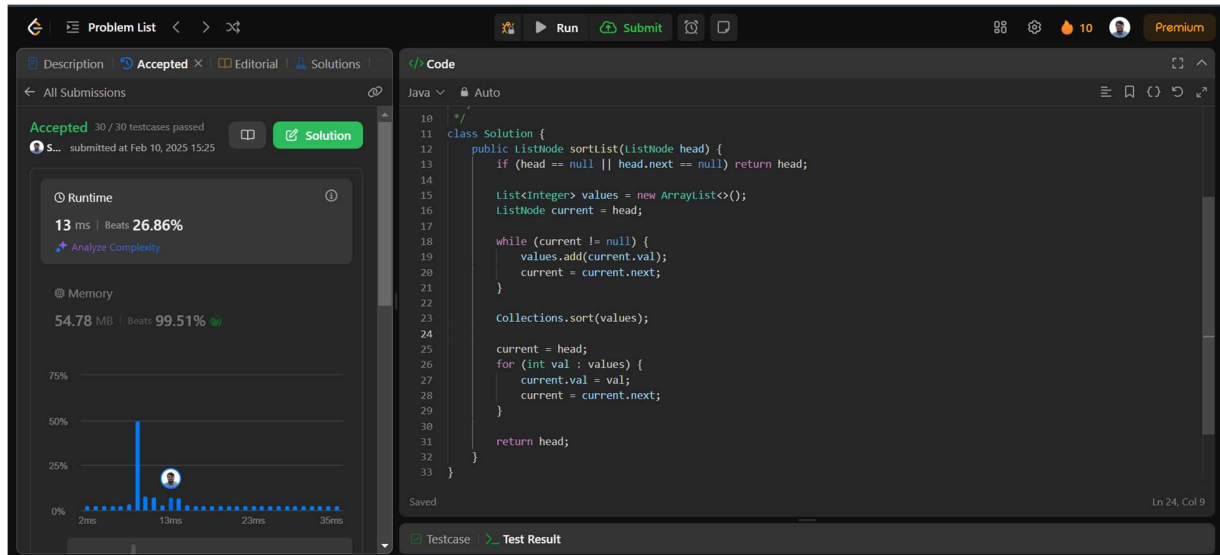
Given the head of a linked list, rotate the list to the right by k places.

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

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

148. Sort List

Given the head of a linked list, return *the list after sorting it in ascending order*.



```
10 //
11 class Solution {
12     public ListNode sortList(ListNode head) {
13         if (head == null || head.next == null) return head;
14
15         List<Integer> values = new ArrayList<>();
16         ListNode current = head;
17
18         while (current != null) {
19             values.add(current.val);
20             current = current.next;
21         }
22
23         Collections.sort(values);
24
25         current = head;
26         for (int val : values) {
27             current.val = val;
28             current = current.next;
29         }
30
31         return head;
32     }
33 }
```

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

        List<Integer> values = new ArrayList<>();
        ListNode current = head;

        while (current != null) {
            values.add(current.val);
            current = current.next;
        }

        Collections.sort(values);

        current = head;
        for (int val : values) {
            current.val = val;
            current = current.next;
        }

        return head;
    }
}
```

23. Merge k Sorted Lists

You are given an array of k linked-lists lists, each linked-list is sorted in ascending order.

Merge all the linked-lists into one sorted linked-list and return it

The screenshot shows a LeetCode submission interface. On the left, the 'Runtime' tab is active, showing a runtime of 83 ms and a memory usage of 44.39 MB, both of which are better than 14.02% and 77.65% of other submissions respectively. The 'Code' tab on the right displays the Java code for the solution. The code defines a ListNode class and a Solution class with two methods: mergeTwoLists and mergeKLists. The mergeTwoLists method uses a while loop to compare the values of two lists and merge them into a new list. The mergeKLists method uses a recursive approach to merge the lists.

```
2  * Definition for singly-linked list.
3  * public class ListNode {
4  *     int val;
5  *     ListNode next;
6  *     ListNode() {}
7  *     ListNode(int val) { this.val = val; }
8  *     ListNode(int val, ListNode next) { this.val = val; this.next = next; }
9  * }
10 */
11 class Solution {
12
13     public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
14         ListNode head = new ListNode();
15         ListNode current = head;
16
17         while (list1 != null && list2 != null) {
18             if (list1.val <= list2.val) {
19                 current.next = list1;
20                 list1 = list1.next;
21             } else {
22                 current.next = list2;
23                 list2 = list2.next;
24             }
25             current = current.next;
26         }
27
28         if (list1 != null) current.next = list1;
29         else if (list2 != null) current.next = list2;
30         return head.next;
31     }
32
33     public ListNode mergeKLists(ListNode[] lists) {
34         if (lists.length == 0) return null;
35         if (lists.length == 1) return lists[0];
36         ListNode list1 = lists[0];
37         for (int i = 1; i < lists.length; i++) {
38             list1 = mergeTwoLists(list1, lists[i]);
39         }
40         return list1;
41     }
42 }
```

```
class Solution {
```

```
    public ListNode mergeTwoLists(ListNode
list1, ListNode list2) {
```

```
        ListNode head = new ListNode();
```

```
        ListNode current = head;
```

```
        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;
```

```
        }
```

```
        if (list1 != null) current.next = list1;
```

```
        else if (list2 != null) current.next = list2;
```

```
        return head.next;}
```

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

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

```
        if (lists.length == 1) return lists[0];
```

```
        ListNode list1 = lists[0];
```

```
        for (int i = 1; i < lists.length; i++) {
```

```
            list1 = mergeTwoLists(list1, lists[i]);
```

```
        }
```

```
        return list1;
```

```
    }
```

```
}
```