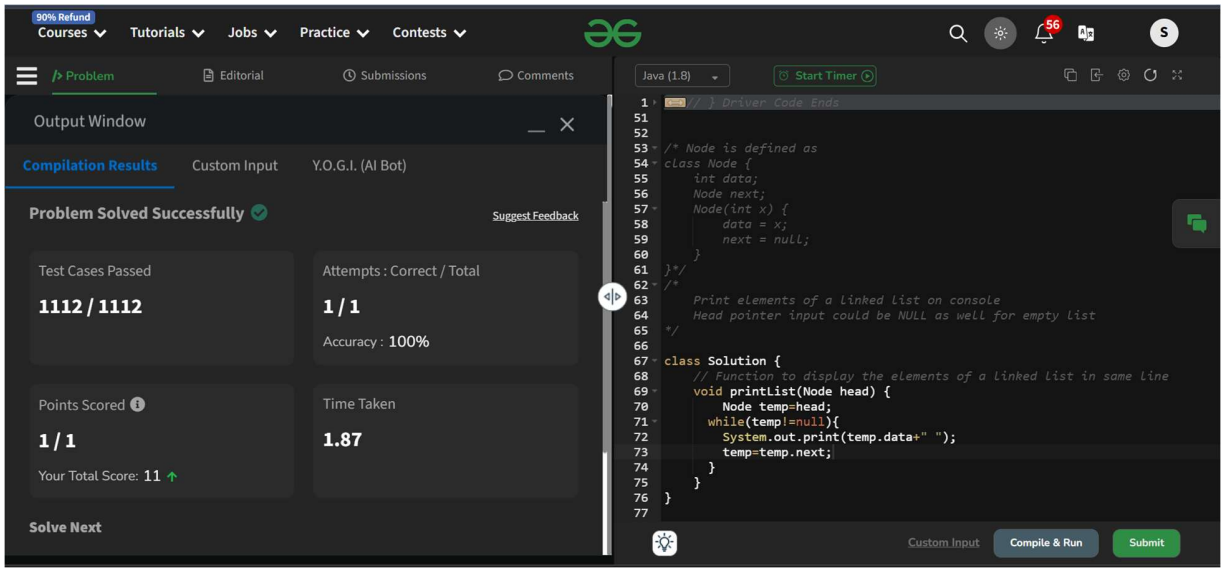
SNEHASIS MEDDA **22BCS16163** **DAY 2**

Print Linked List

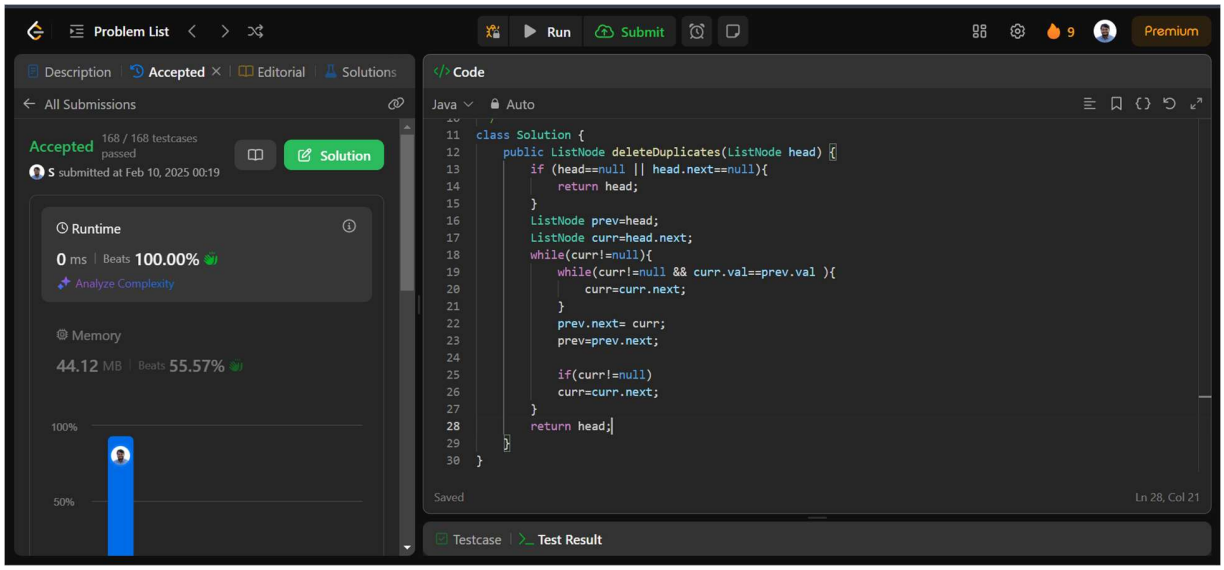
Given a linked list. Print all the elements of the linked list separated by space followed.




```
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

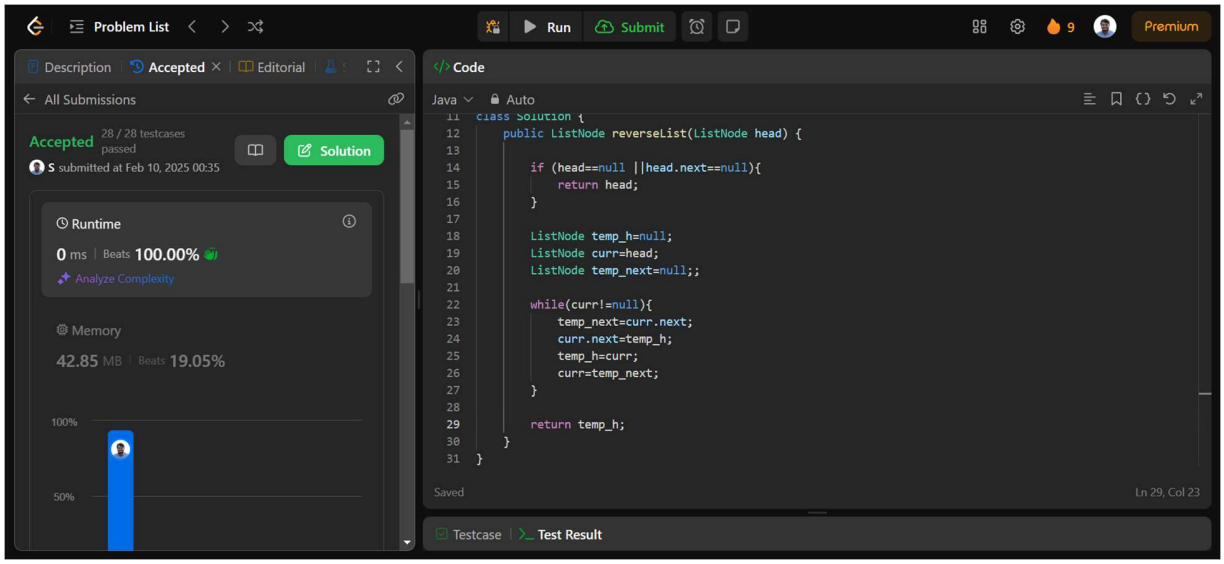
Given the head of a sorted linked list, *delete all duplicates such that each element appears only once*. Return *the linked list* ***sorted*** *as well*.




```
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*.



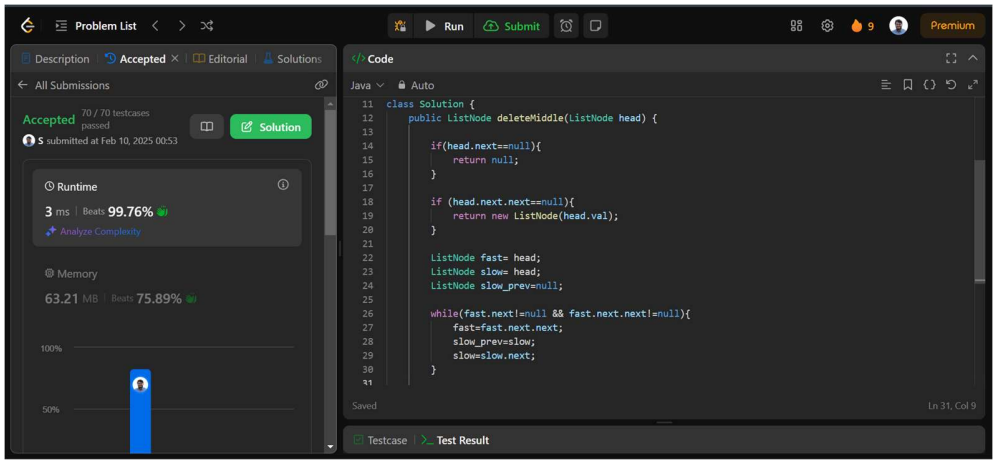

```
class Solution {
    public ListNode reverseList(ListNode head) {
        if (head==null ||head.next==null){
            return head;
        }
        ListNode temp_h=null;
        ListNode curr=head;
        ListNode temp_next=null;;
        while(curr!=null){
            temp_next=curr.next;
            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 $\lfloor n / 2 \rfloor^{th}$ 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.



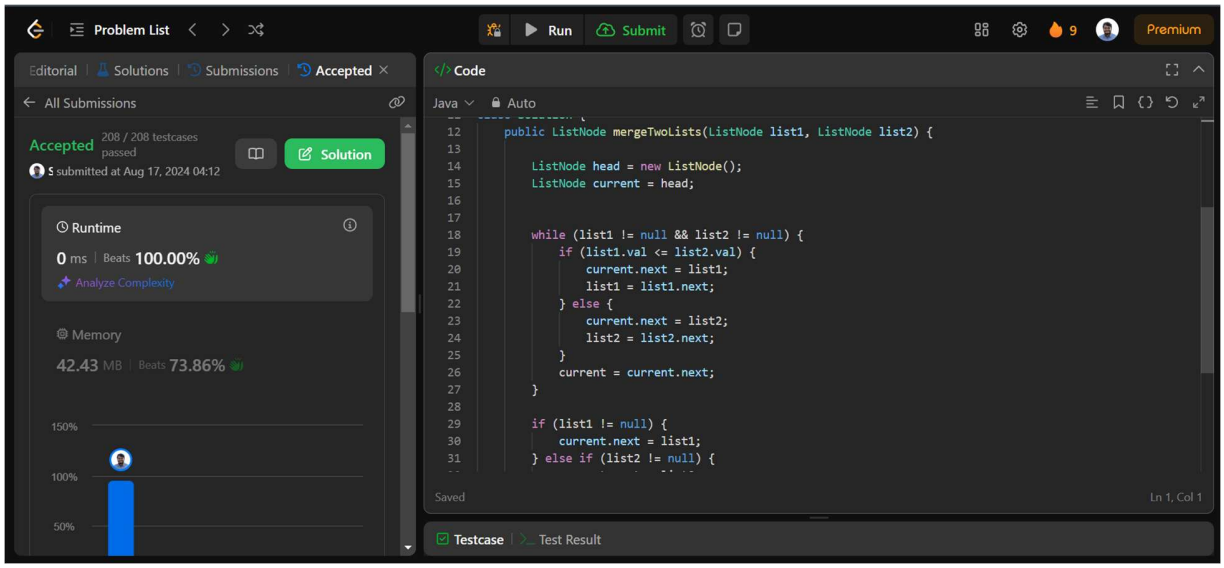

```
class Solution {
  public ListNode deleteMiddle(ListNode head) {
    if(head.next==null)    return null;
    if (head.next.next==null)   return new ListNode(head.val);
    ListNode fast= head;
    ListNode slow= head;
    ListNode slow_prev=null;
    while(fast.next!=null && fast.next.next!=null){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

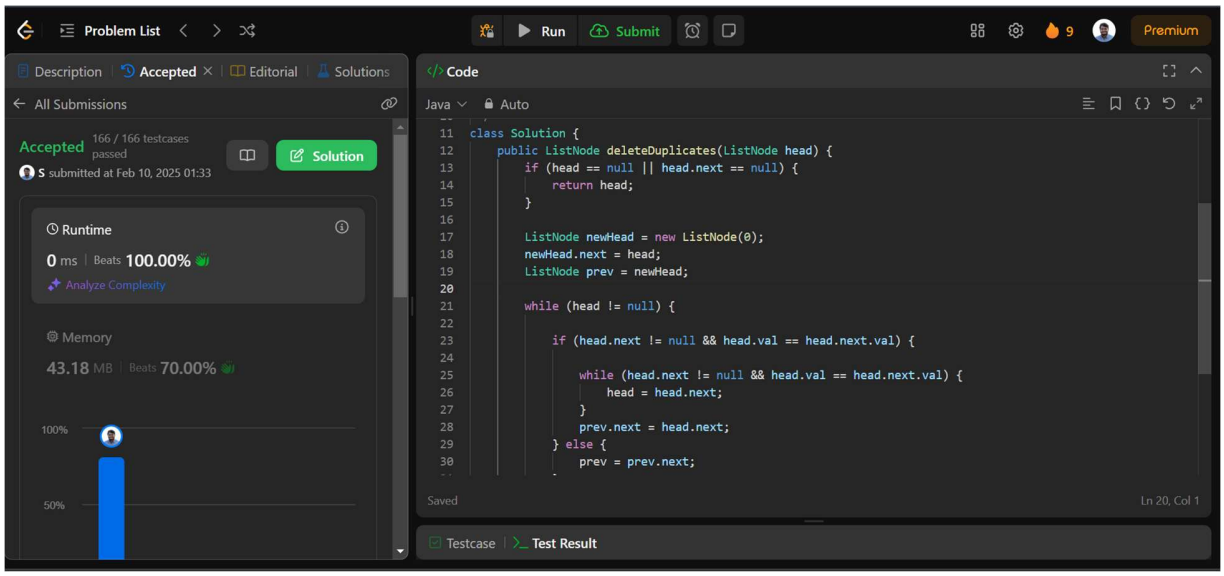
Return *the head of the merged linked list*.




```
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

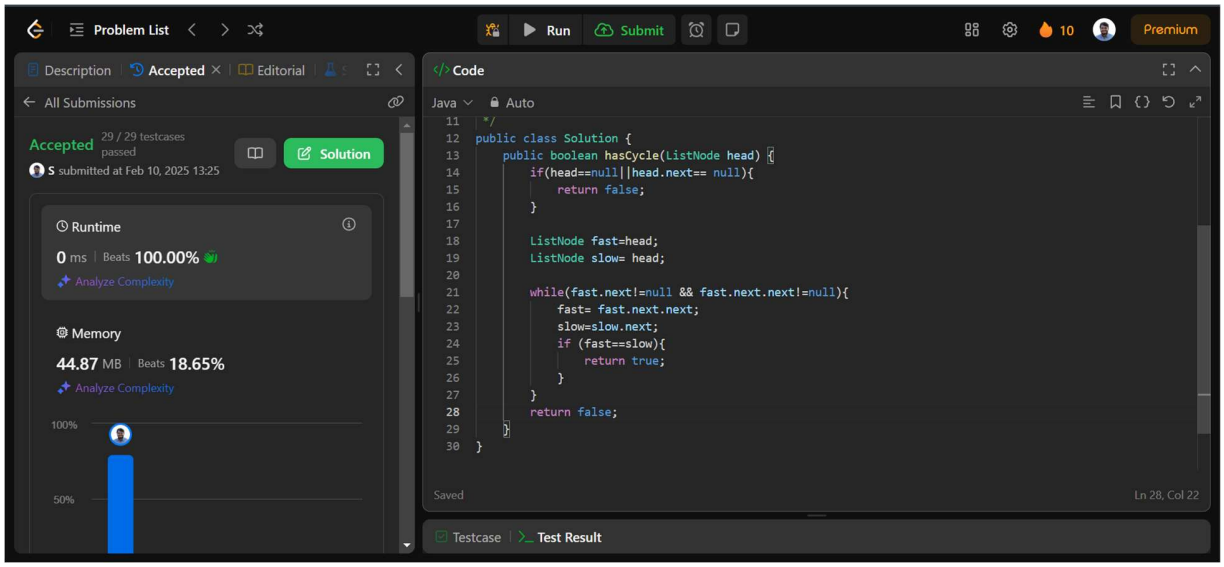
Given the head of a sorted linked list, *delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list*. Return *the linked list* ***sorted*** *as well*.




```
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

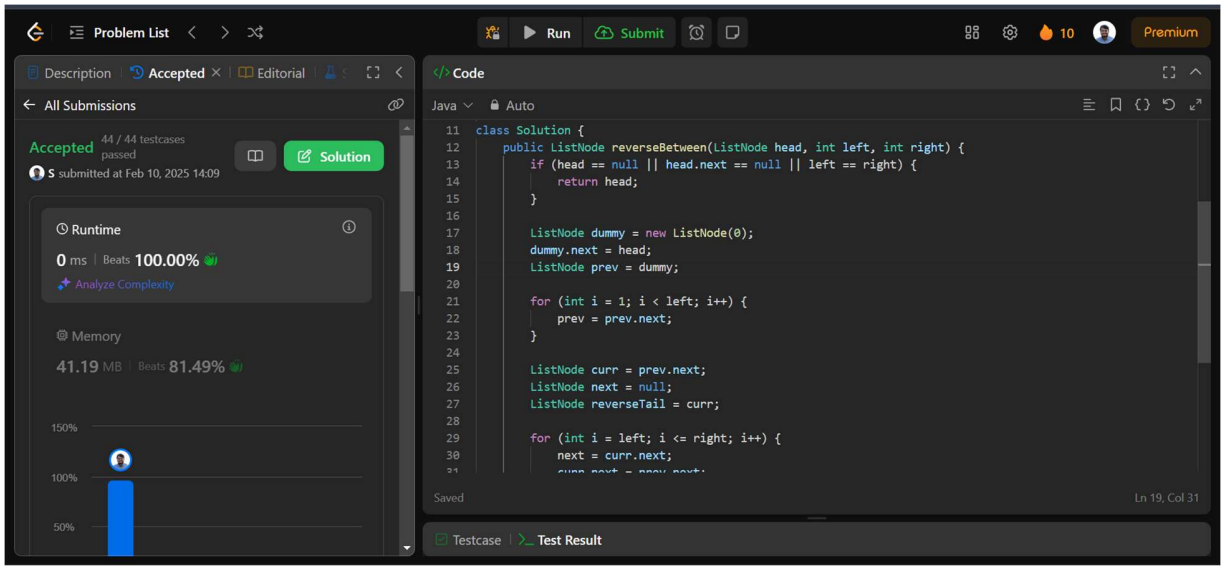
Given head, the head of a linked list, determine if the linked list has a cycle in it.




```
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

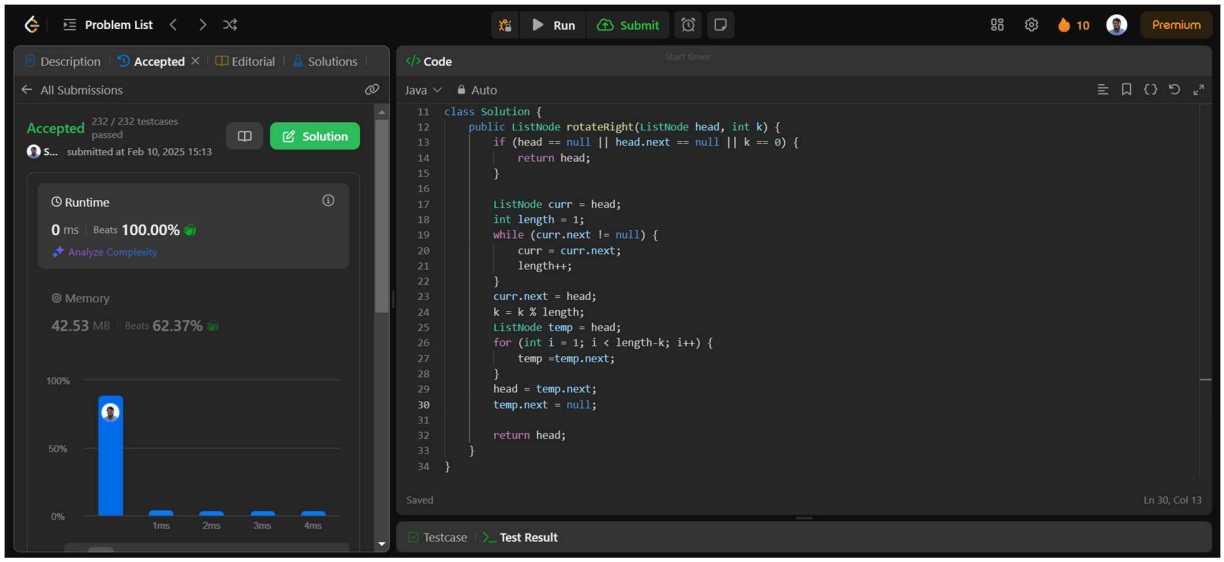
Given the head of a singly linked list and two integers left and right where left <= right, reverse the nodes of the list from position left to position right, and return *the reversed list*.




```
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

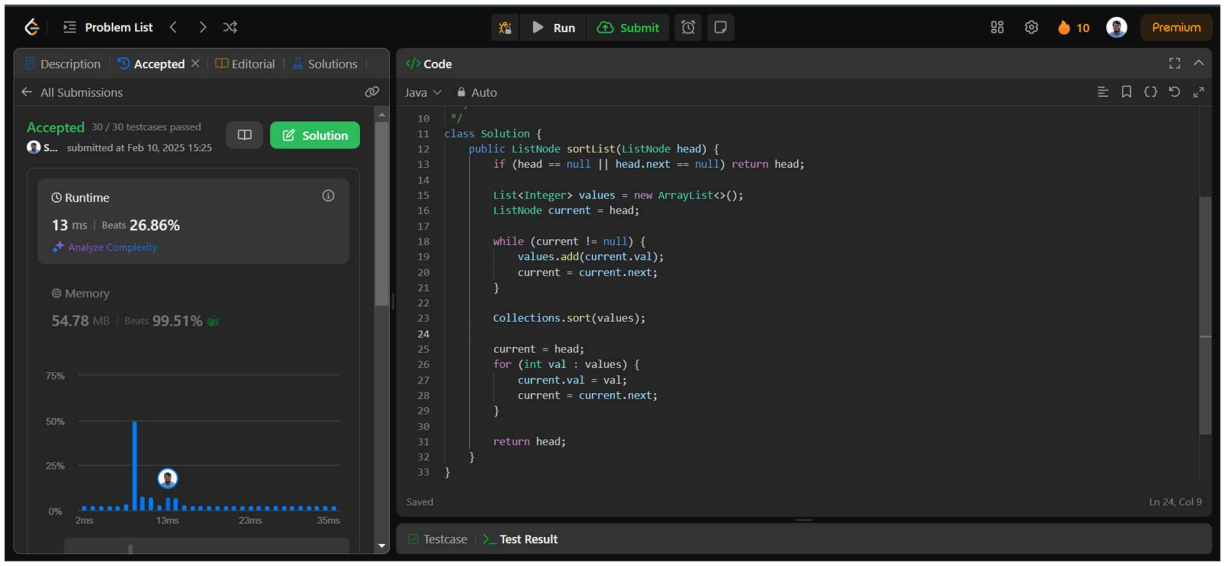
Given the head of a linked list, rotate the list to the right by k places.




```
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;
    }
}
```

148. Sort List

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

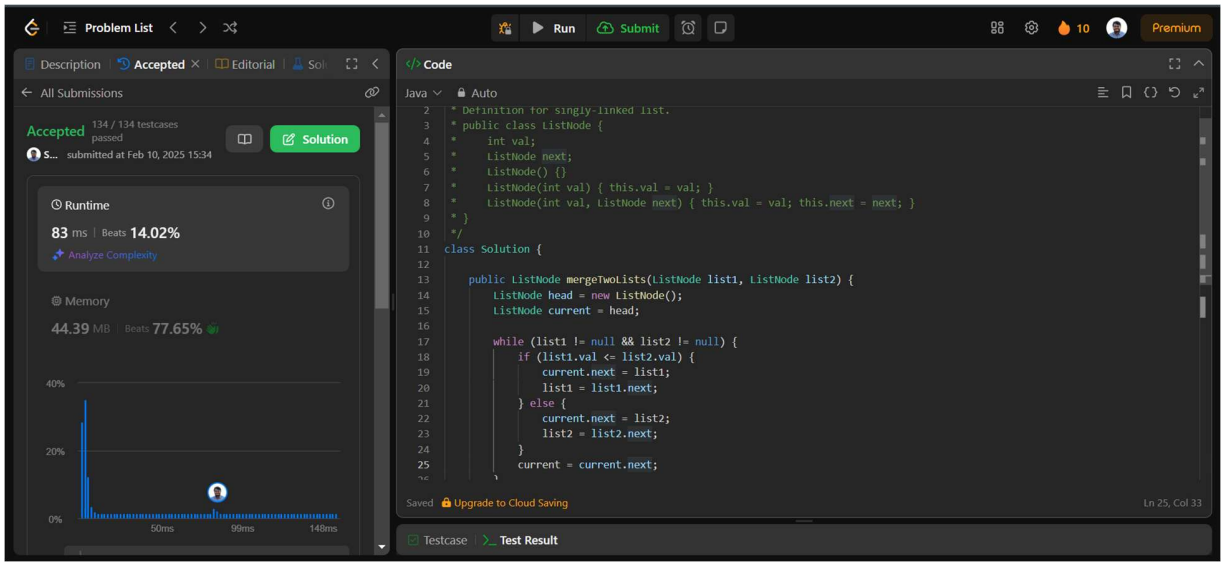



```
class Solution {
   public ListNode sortList(ListNode head) {
      if (head == null || head.next == null) return head;

      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




```
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;
   }
}
```