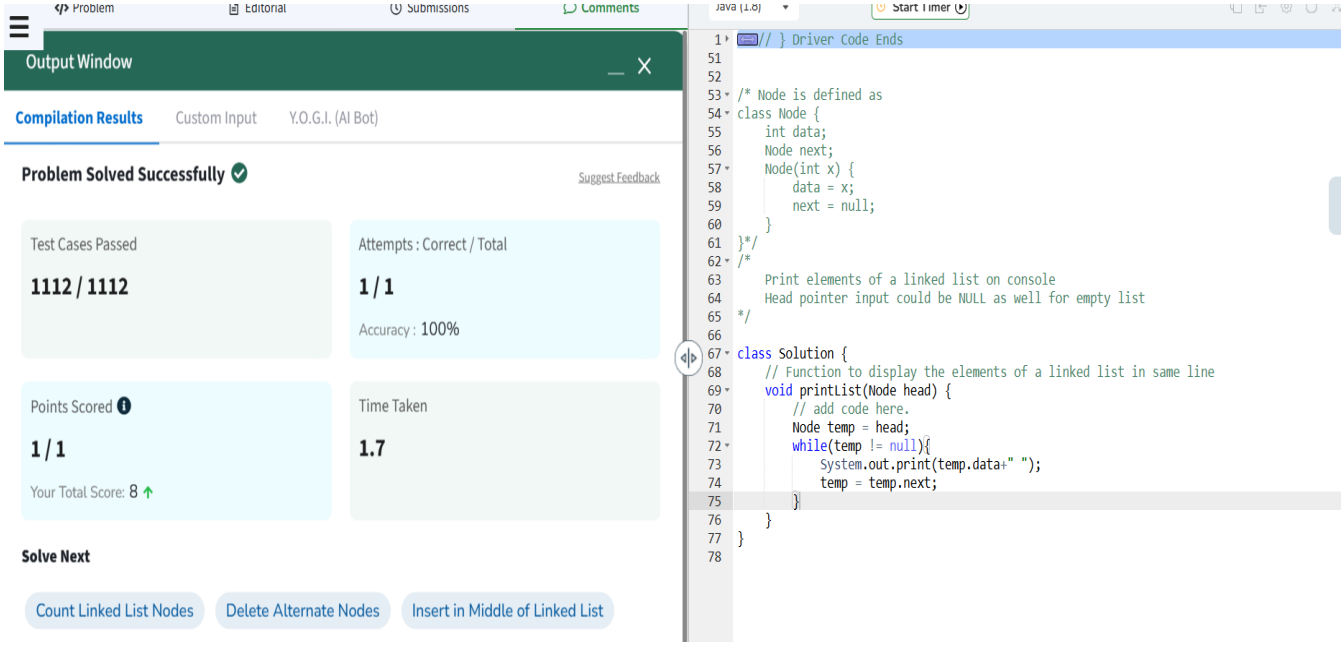
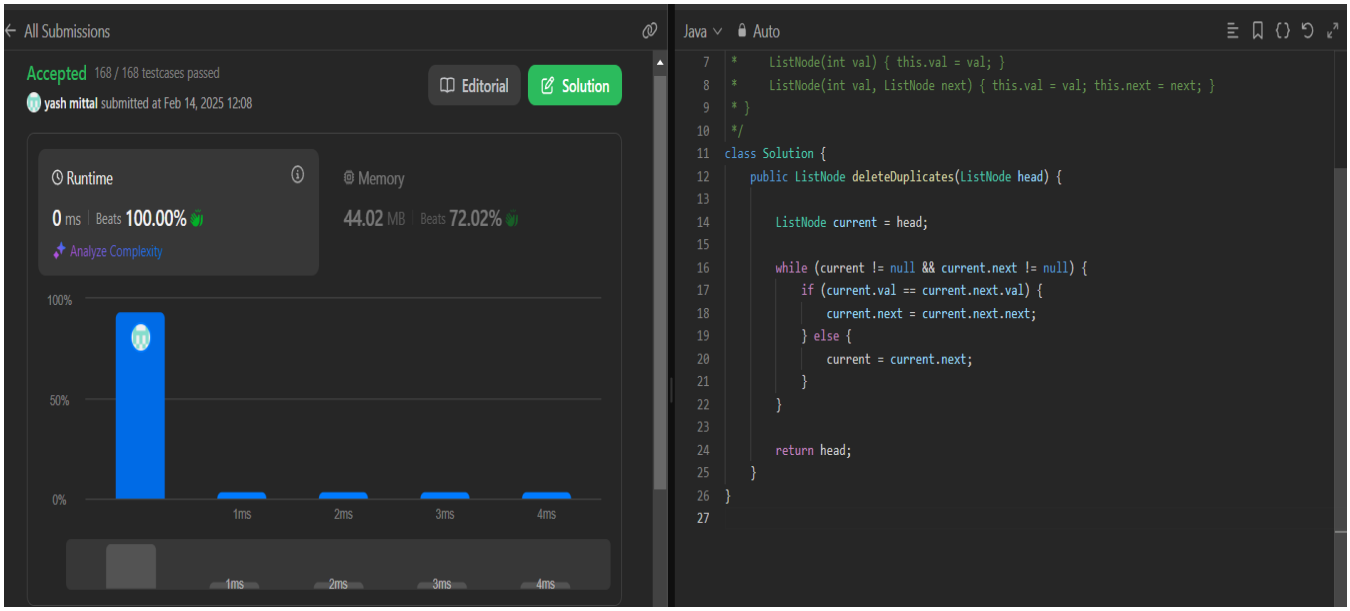
PRINT LINKED LIST



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;
      }
   }
}
```

REMOVE DUPLICATE ELEMENTS FROM SORTED LIST



CODE:

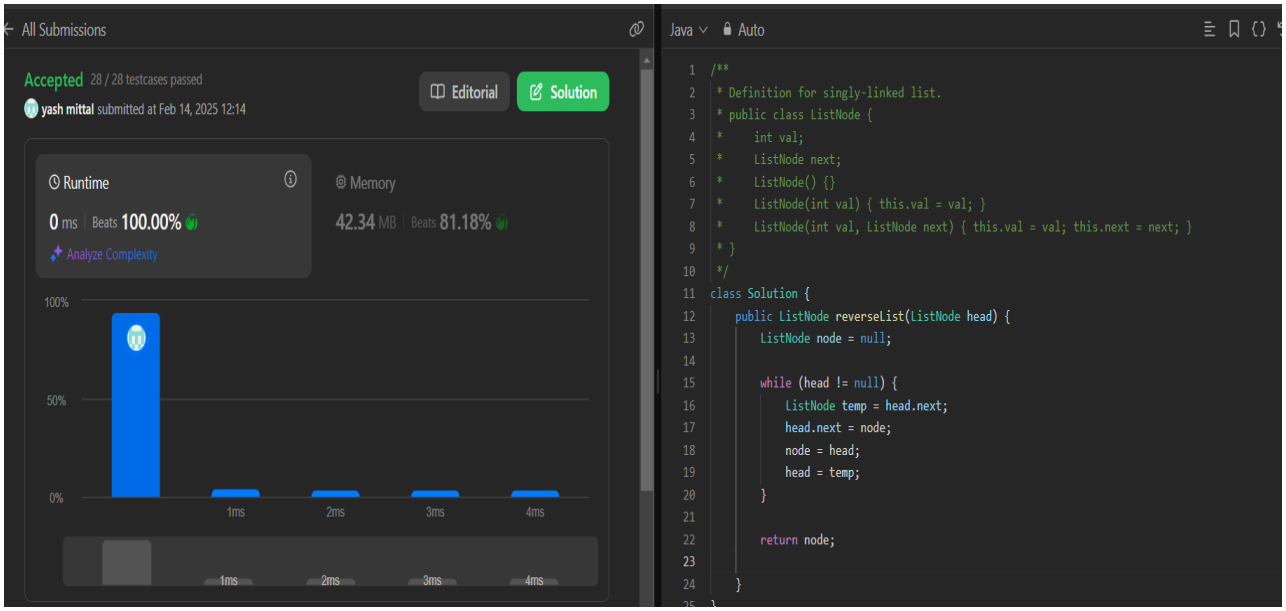
```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {

        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:

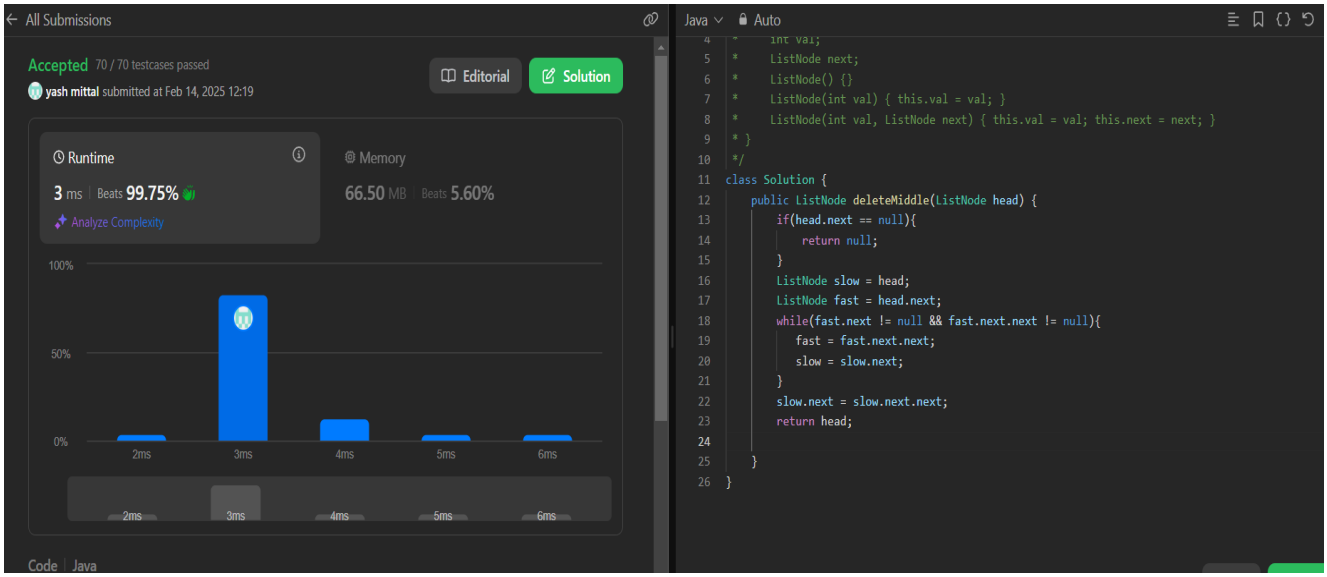
```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode node = null;

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

        return node;

    }
}
```

DELETE MIDDLE NODE OF THE LIST

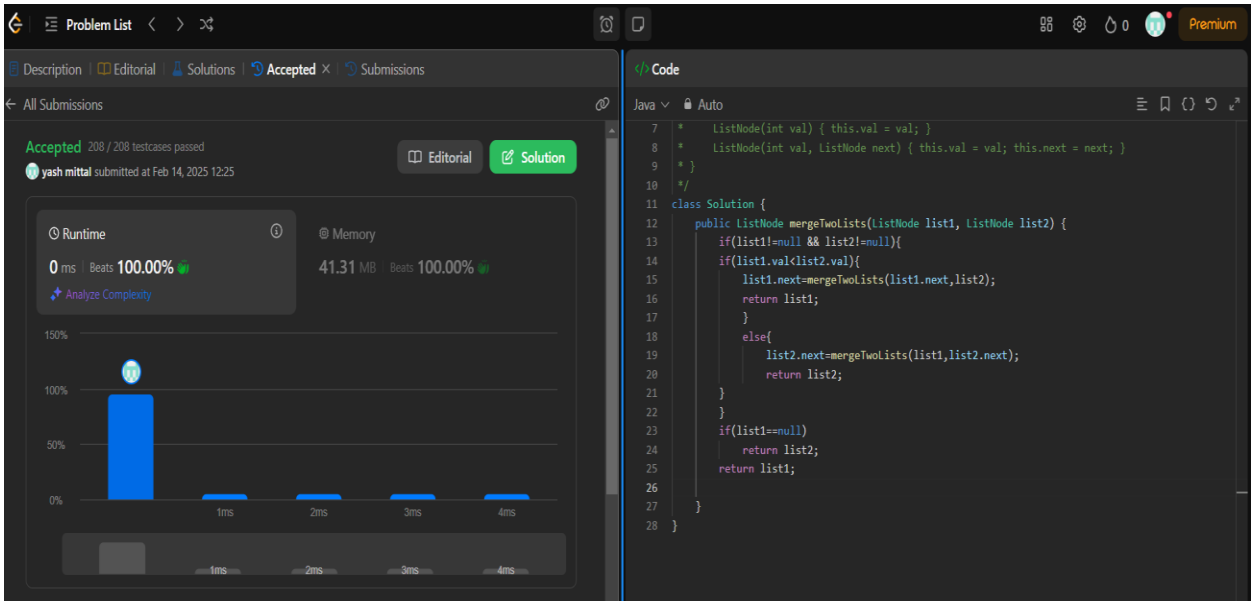


CODE:

```
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;

    }
}
```

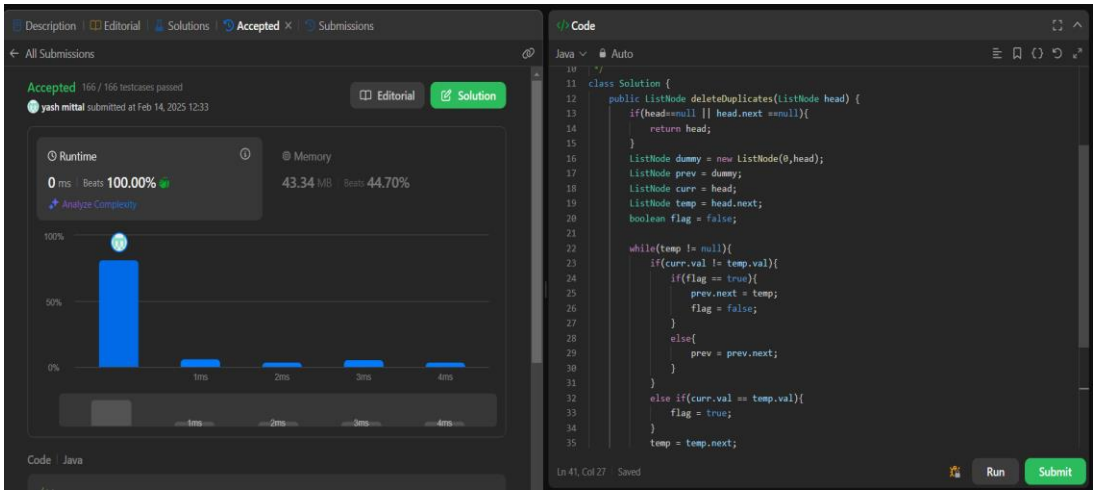
MERGE TWO SORTED LINKED LISTS



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



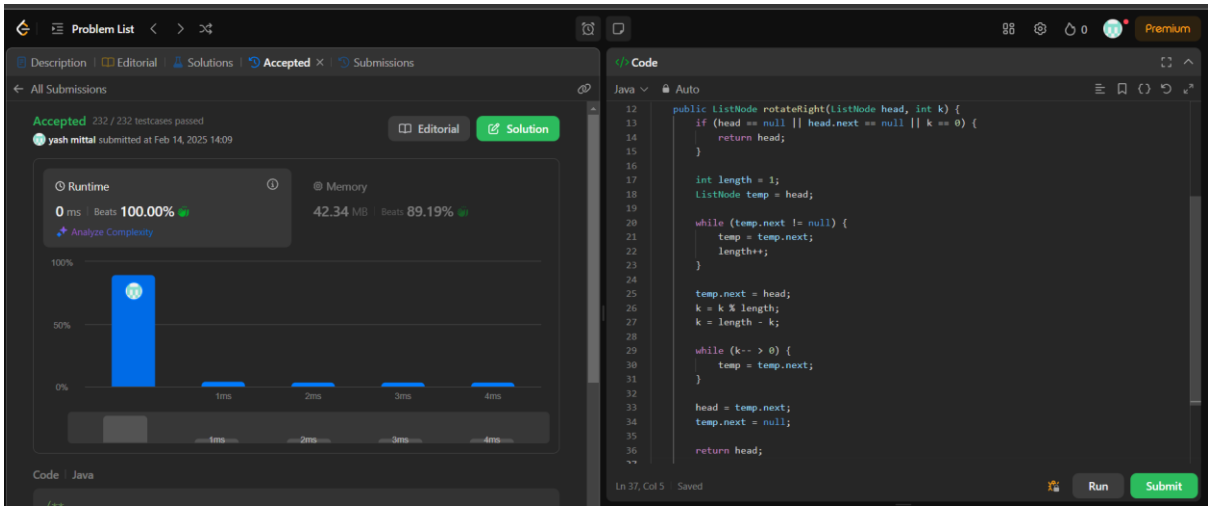
CODE:

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

ROTATE LIST

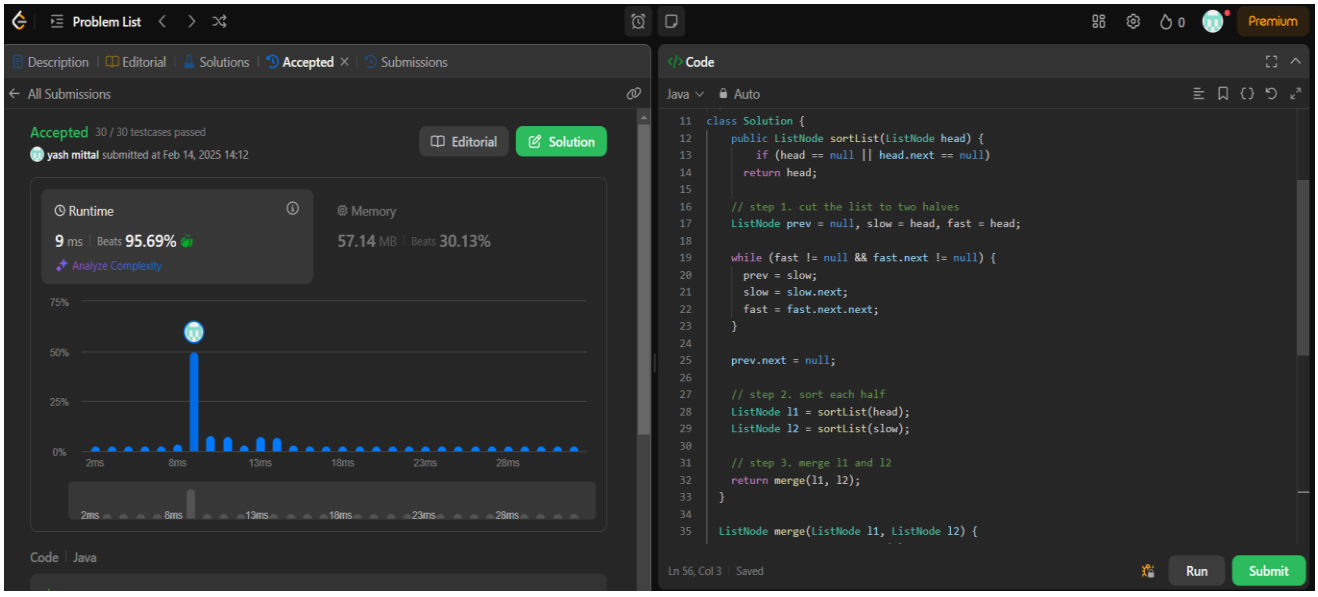


CODE:

```
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

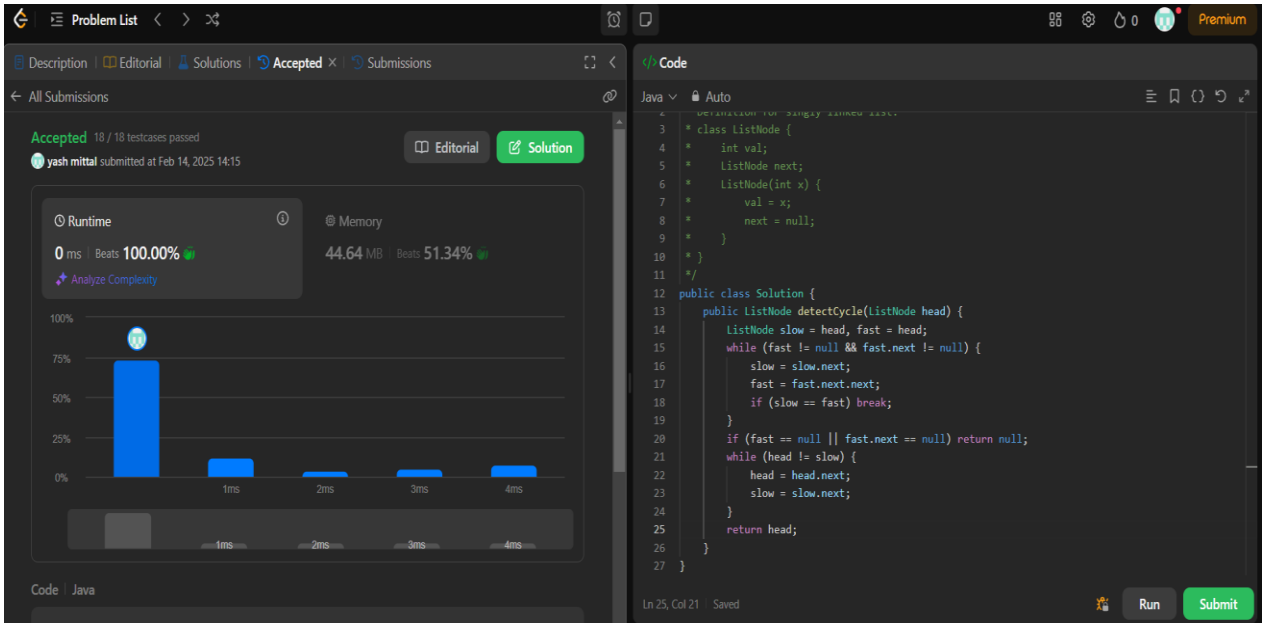


CODE:

```
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



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;
    }
}
```