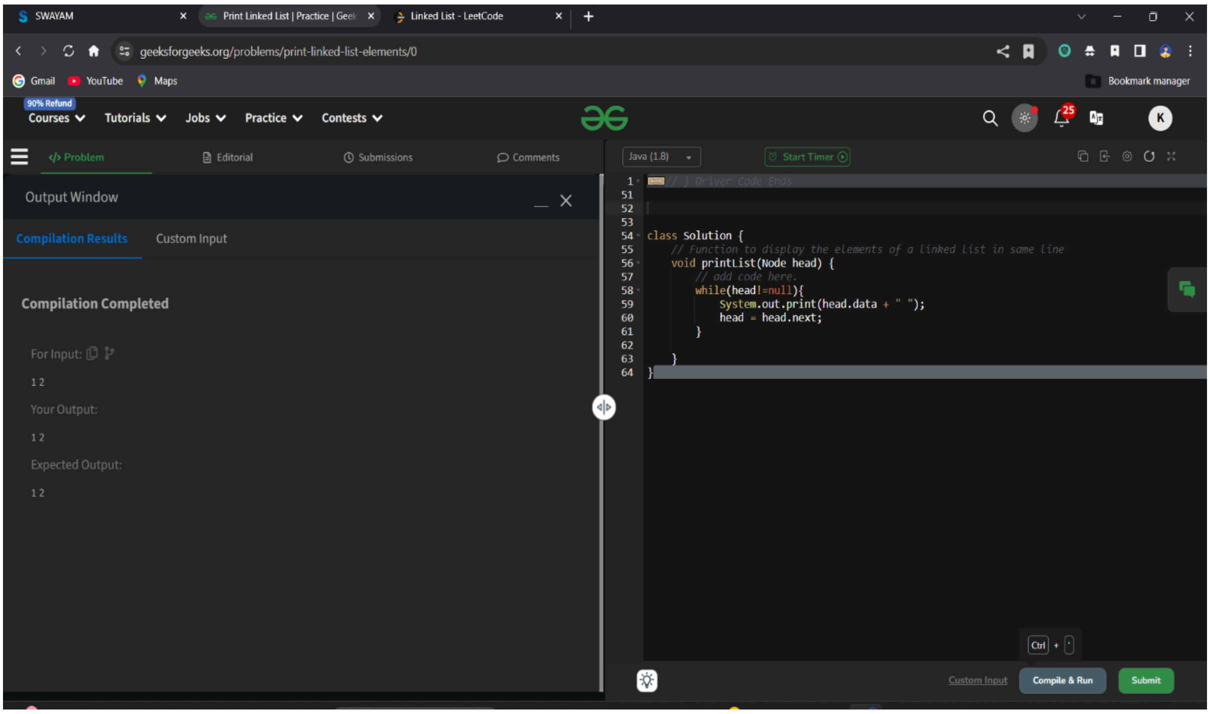
Assignment no.3

Ques 1. Print linked list

```
class Solution {
    // Function to display the elements of a linked list in same line
    void printList(Node head) {
        // add code here.
        while(head!=null){
            System.out.print(head.data + " ");
            head = head.next;
        }
    }
}
```



```
// } Driver Code Ends


class Solution {
    // Function to display the elements of a linked list in same line
    void printList(Node head) {
        // add code here.
        while(head!=null){
            System.out.print(head.data + " ");
            head = head.next;
        }

    }
}
```

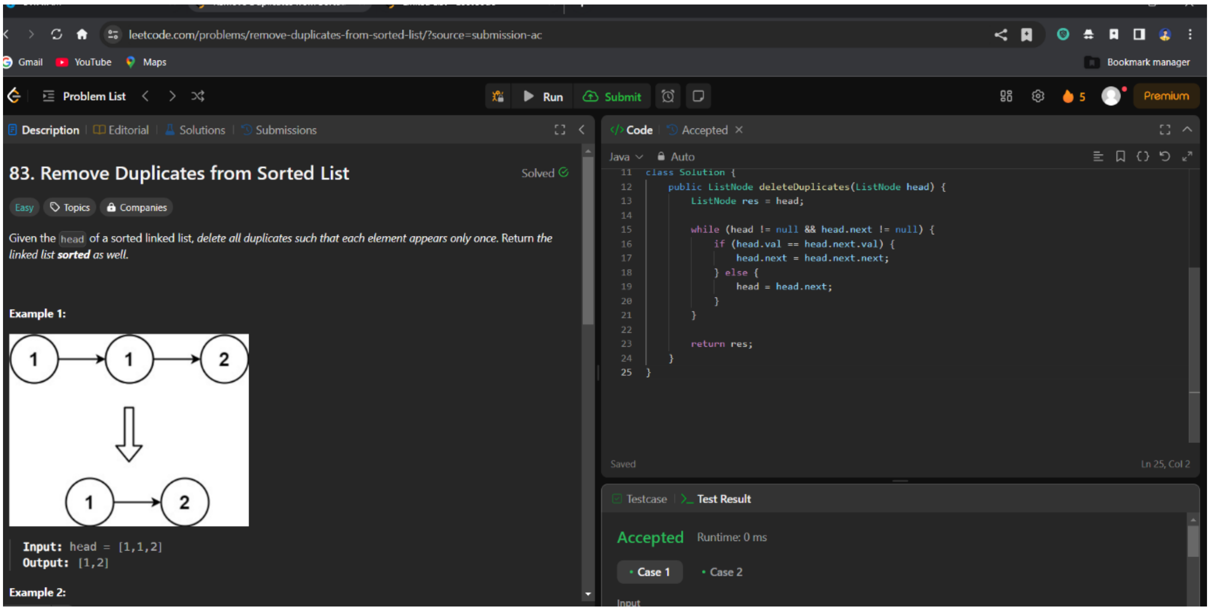
Output Window

Compilation Results | Custom Input

```
Compilation Completed

For Input:
1 2
Your Output:
1 2
Expected Output:
1 2
```

Ques 2. Remove duplicates from a sorted list

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode res = head;

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



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.*

Example 1:

```
Input: head = [1,1,2]
Output: [1,2]
```

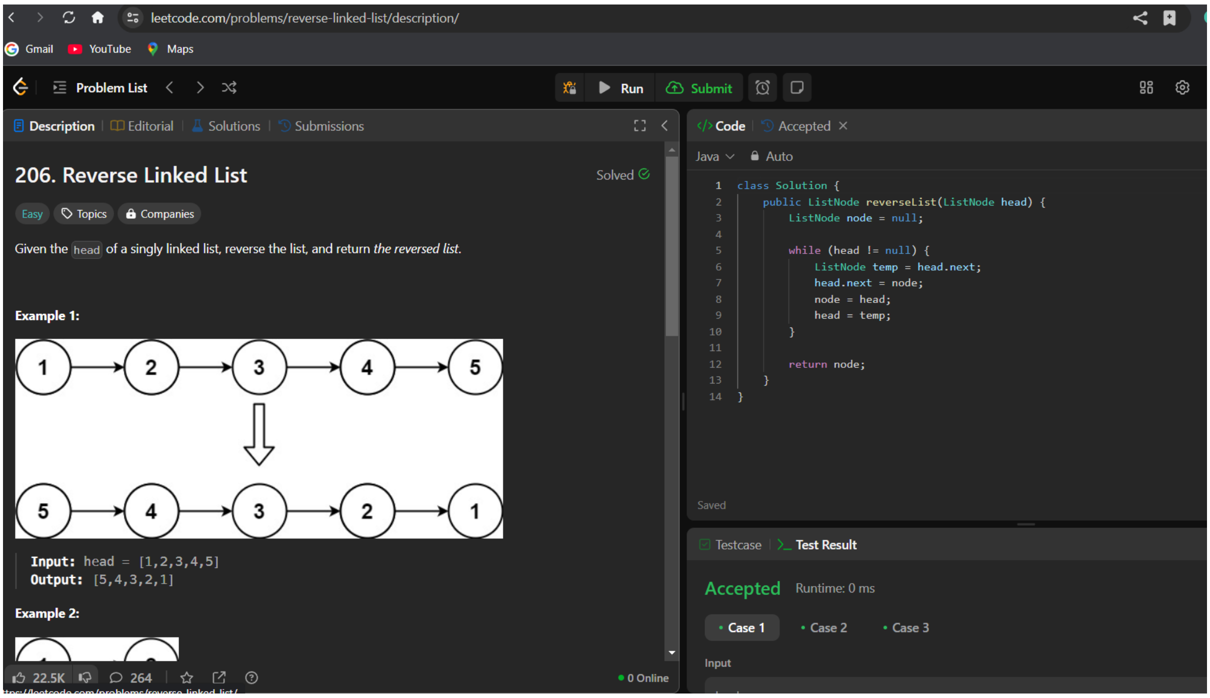
Example 2:

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode res = head;

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

        return res;
    }
}
```

Accepted Runtime: 0 ms

Ques 3. Reverse a linked list] (https://leetcode.com/problems/reverse-linked-list/

```
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;
    }
}
```



206. Reverse Linked List

Solved

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

Example 1:

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

Example 2:

```
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;
    }
}
```

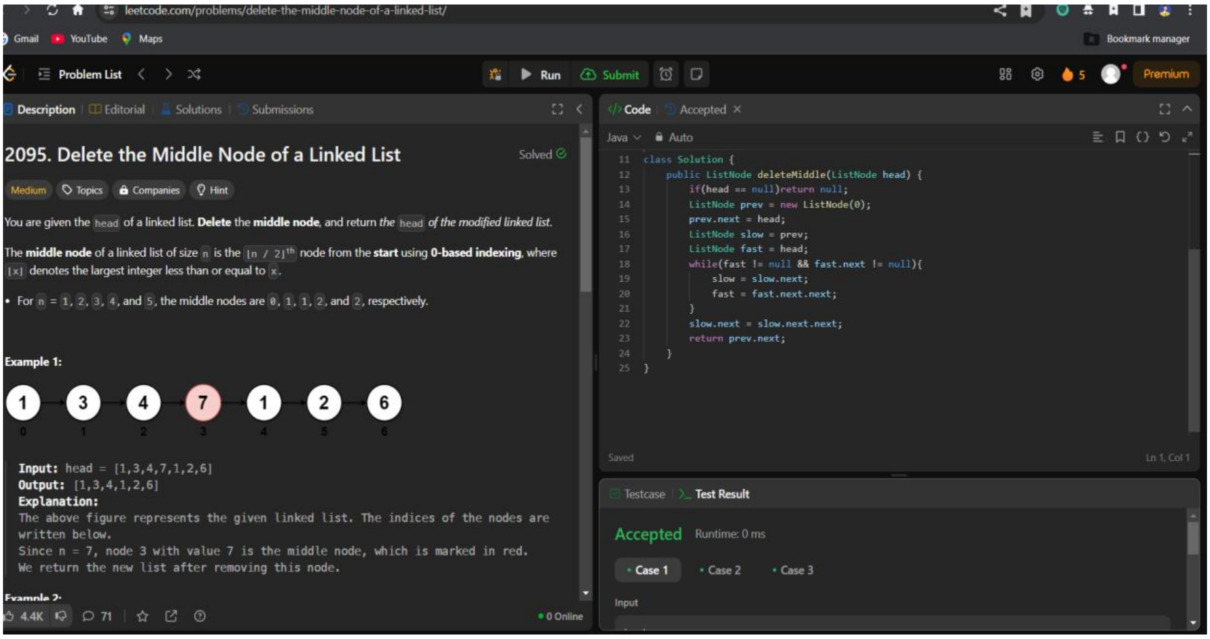
Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

Ques 4. [Delete middle node of a list]

```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if(head == null)return null;
        ListNode prev = new ListNode(0);
        prev.next = head;
        ListNode slow = prev;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        slow.next = slow.next.next;
        return prev.next;
    }
}
```



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 `⌊n / 2⌋th` node from the **start** using **0-based indexing**, where `⌊x⌋` 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.

Example 1:

```
Input: head = [1,3,4,7,1,2,6]
Output: [1,3,4,1,2,6]
Explanation:
The above figure represents the given linked list. The indices of the nodes are
written below.
Since n = 7, node 3 with value 7 is the middle node, which is marked in red.
We return the new list after removing this node.
```

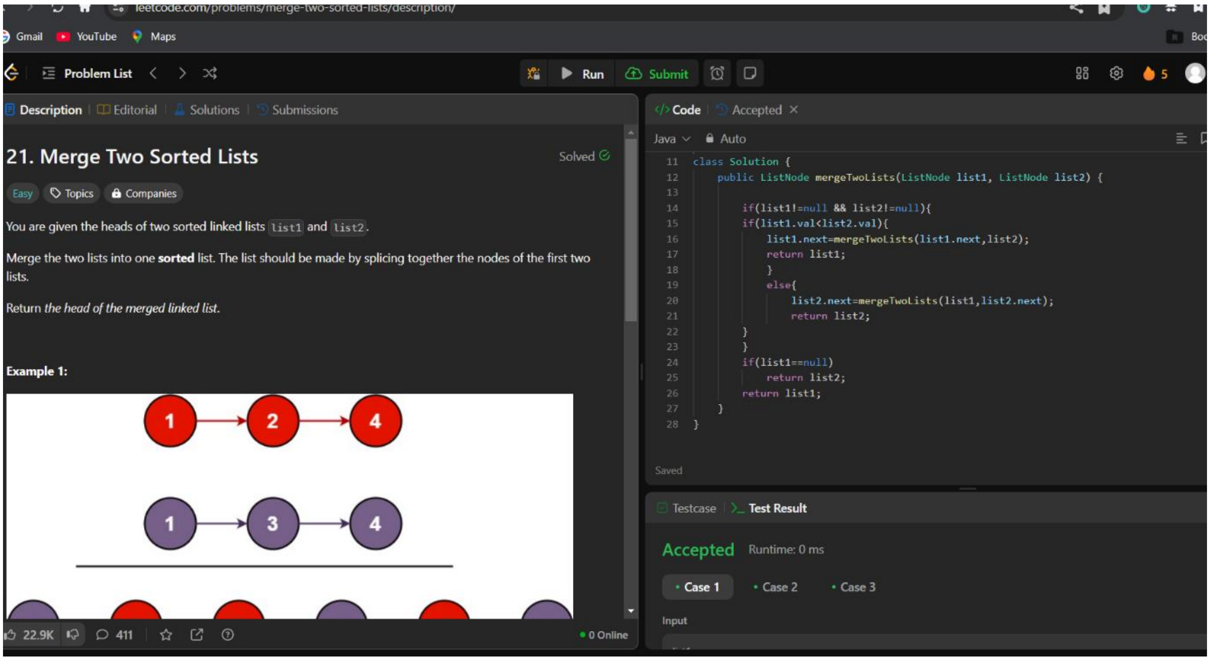
```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        if(head == null)return null;
        ListNode prev = new ListNode(0);
        prev.next = head;
        ListNode slow = prev;
        ListNode fast = head;
        while(fast != null && fast.next != null){
            slow = slow.next;
            fast = fast.next.next;
        }
        slow.next = slow.next.next;
        return prev.next;
    }
}
```

Accepted Runtime: 0 ms

Ques 5. Merge two sorted linked lists

```
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;
  }
}
```



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.*

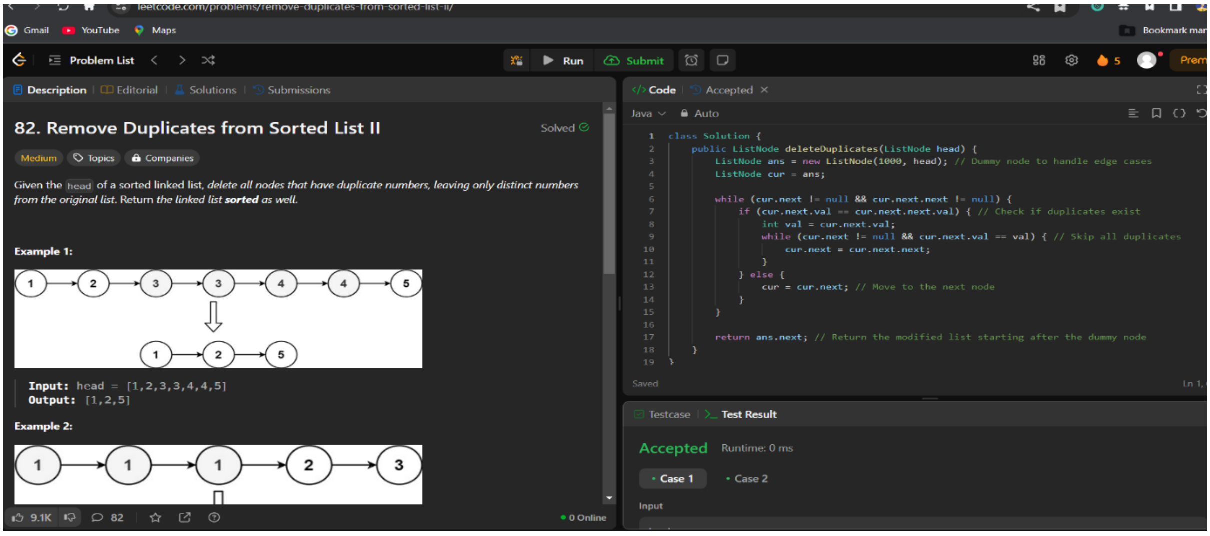
Example 1:

```
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;
    }
}
```

Accepted Runtime: 0 ms

Ques 6. Remove duplicates from sorted lists 2

```
class Solution {
  public ListNode deleteDuplicates(ListNode head) {
        ListNode ans = new ListNode(1000, head); // Dummy node to handle edge cases
        ListNode cur = ans;

        while (cur.next != null && cur.next.next != null) {
            if (cur.next.val == cur.next.next.val) { // Check if duplicates exist
                int val = cur.next.val;
                while (cur.next != null && cur.next.val == val) { // Skip all duplicates
                    cur.next = cur.next.next;
                }
            } else {
                cur = cur.next; // Move to the next node
            }
        }

        return ans.next; // Return the modified list starting after the dummy node
    }
}
```



82. Remove Duplicates from Sorted List II

Solved

Medium · Topics · Companies

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*.

Example 1:

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

Example 2:

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode ans = new ListNode(1000, head); // Dummy node to handle edge cases
        ListNode cur = ans;

        while (cur.next != null && cur.next.next != null) {
            if (cur.next.val == cur.next.next.val) { // Check if duplicates exist
                int val = cur.next.val;
                while (cur.next != null && cur.next.val == val) { // Skip all duplicates
                    cur.next = cur.next.next;
                }
            } else {
                cur = cur.next; // Move to the next node
            }
        }

        return ans.next; // Return the modified list starting after the dummy node
    }
}
```

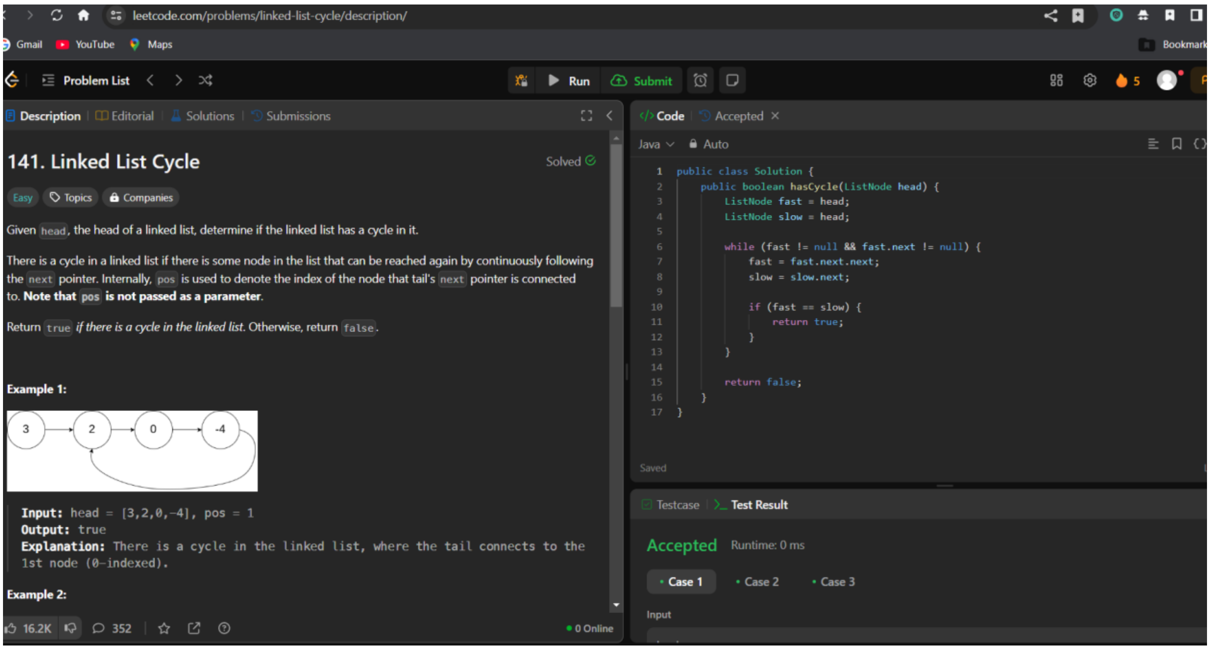
Accepted Runtime: 0 ms

Case 1 · Case 2

Input

Ques 7. [Detect a cycle in a linked list]

```
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode fast = head;
        ListNode slow = head;

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

            if (fast == slow) {
                return true;
            }
        }

        return false;
    }
}
```



141. Linked List Cycle

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

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to. **Note that `pos` is not passed as a parameter.**

Return `true` *if there is a cycle in the linked list*. Otherwise, return `false`.

Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: true
Explanation: There is a cycle in the linked list, where the tail connects to the
1st node (0-indexed).
```

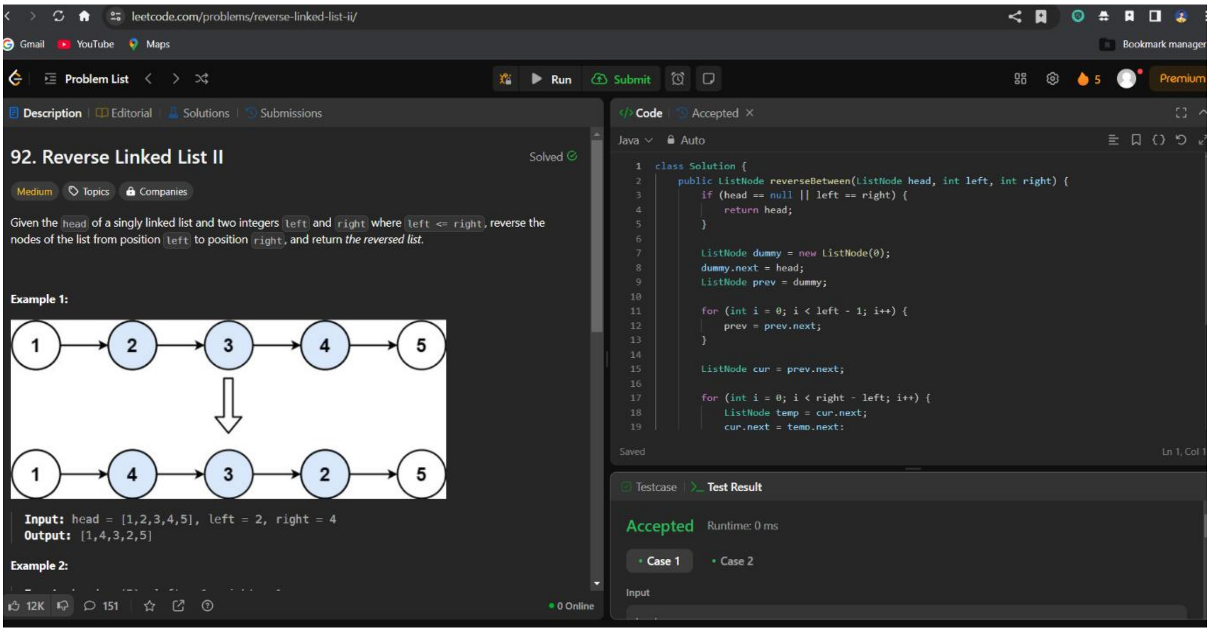
Example 2:

```
public class Solution {
    public boolean hasCycle(ListNode head) {
        ListNode fast = head;
        ListNode slow = head;

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

            if (fast == slow) {
                return true;
            }
        }

        return false;
    }
}
```

Accepted Runtime: 0 ms

Ques 8. Reverse linked list 2

```
class Solution {
    public ListNode reverseBetween(ListNode head, int left, int right) {
        if (head == null || left == right) {
            return head;
        }

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

        for (int i = 0; i < left - 1; i++) {
            prev = prev.next;
        }

        ListNode cur = prev.next;

        for (int i = 0; i < right - left; i++) {
            ListNode temp = cur.next;
            cur.next = temp.next;
            temp.next = prev.next;
            prev.next = temp;
        }
     return dummy.next;
    }
}
```



92. Reverse Linked List II

Solved

Medium · Topics · Companies

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.*

Example 1:

```
Input: head = [1,2,3,4,5], left = 2, right = 4
Output: [1,4,3,2,5]
```

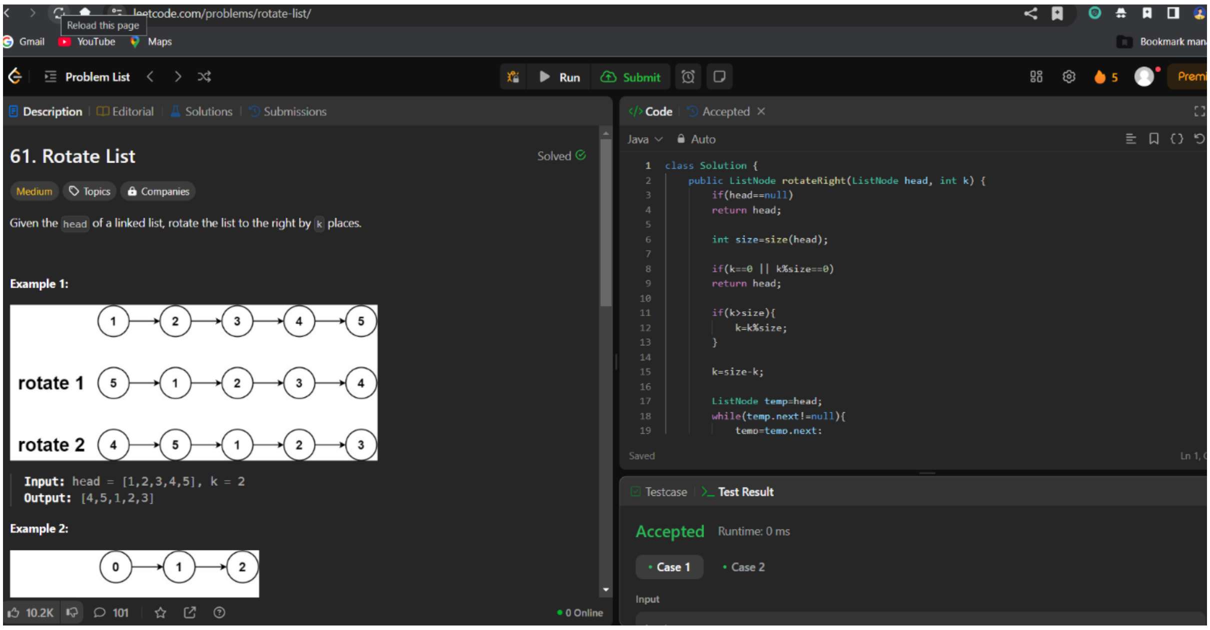
Example 2:

```
class Solution {
    public ListNode reverseBetween(ListNode head, int left, int right) {
        if (head == null || left == right) {
            return head;
        }

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

        for (int i = 0; i < left - 1; i++) {
            prev = prev.next;
        }

        ListNode cur = prev.next;

        for (int i = 0; i < right - left; i++) {
            ListNode temp = cur.next;
            cur.next = temp.next;
```

Accepted Runtime: 0 ms

Ques 9. rotate a list

```
class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if(head==null)
        return head;

        int size=size(head);

        if(k==0 || k%size==0)
        return head;

        if(k>size){
            k=k%size;
        }

        k=size-k;
        ListNode temp=head;
        while(temp.next!=null){
            temp=temp.next;
        }
        ListNode tail=temp;
        temp=head;
        while(k!=1)
        {
            temp=temp.next;
            k--;
        }

        tail.next=head;
        head=temp.next;
        temp.next=null;
        return head;


    }
    public int size(ListNode head){
        int size=0;
        while(head!=null){
            head=head.next;
            size++;
        }return size;
    }
}
```



61. Rotate List

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

Example 1:

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

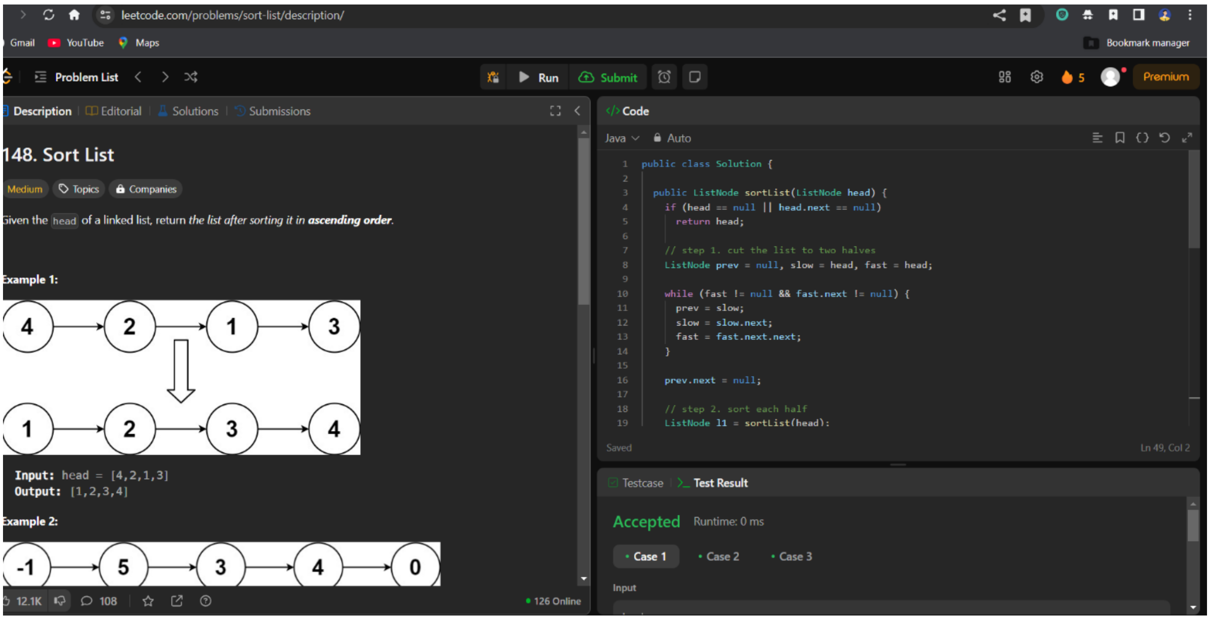
Example 2:

```java
class Solution {
    public ListNode rotateRight(ListNode head, int k) {
        if(head==null)
        return head;

        int size=size(head);

        if(k==0 || k%size==0)
        return head;

        if(k>size){
            k=k%size;
        }

        k=size-k;

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

Accepted Runtime: 0 ms

Ques 10. Sort List

```java
public class Solution {

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

    // step 1. cut the list to two halves
    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;
  }
}
```



148. Sort List

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

Example 1:

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

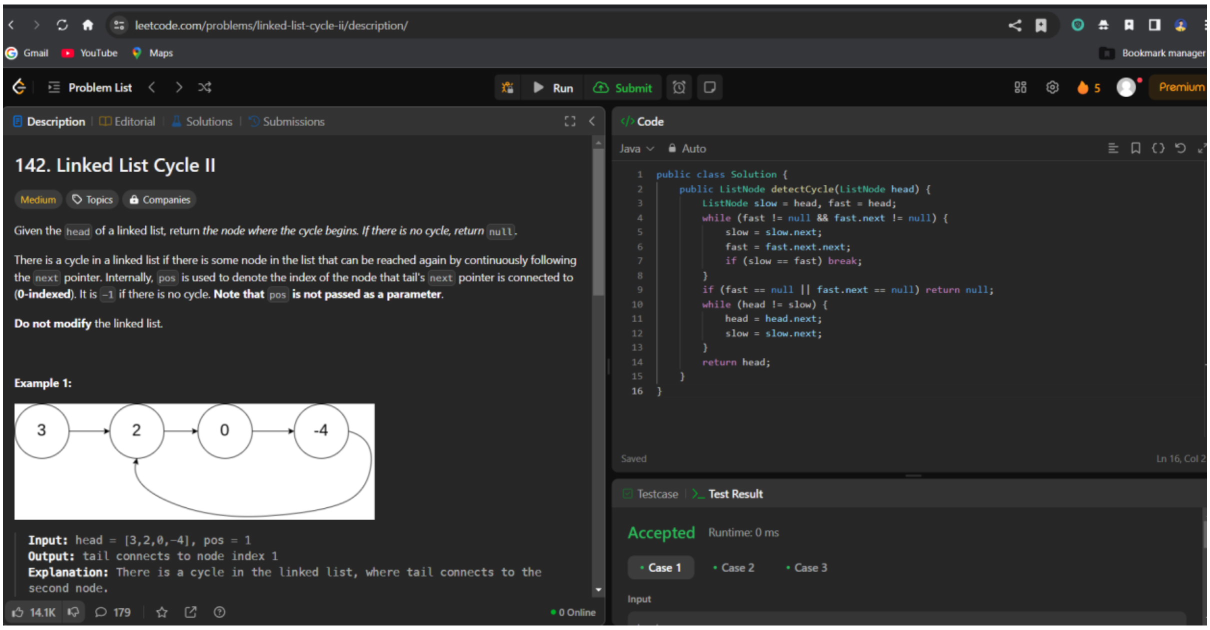
Example 2:

```
public class Solution {

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

    // step 1. cut the list to two halves
    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);
```

Accepted Runtime: 0 ms

Ques 11. Detect a cycle in a linked list 2

```
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;
    }
}
```



142. Linked List Cycle II

Given the `head` of a linked list, return *the node where the cycle begins. If there is no cycle, return* `null`.

There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the `next` pointer. Internally, `pos` is used to denote the index of the node that tail's `next` pointer is connected to (**0-indexed**). It is `-1` if there is no cycle. **Note that** `pos` **is not passed as a parameter**.

Do not modify the linked list.

Example 1:

```
Input: head = [3,2,0,-4], pos = 1
Output: tail connects to node index 1
Explanation: There is a cycle in the linked list, where tail connects to the
second node.
```

```
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;
    }
}
```

Accepted Runtime: 0 ms