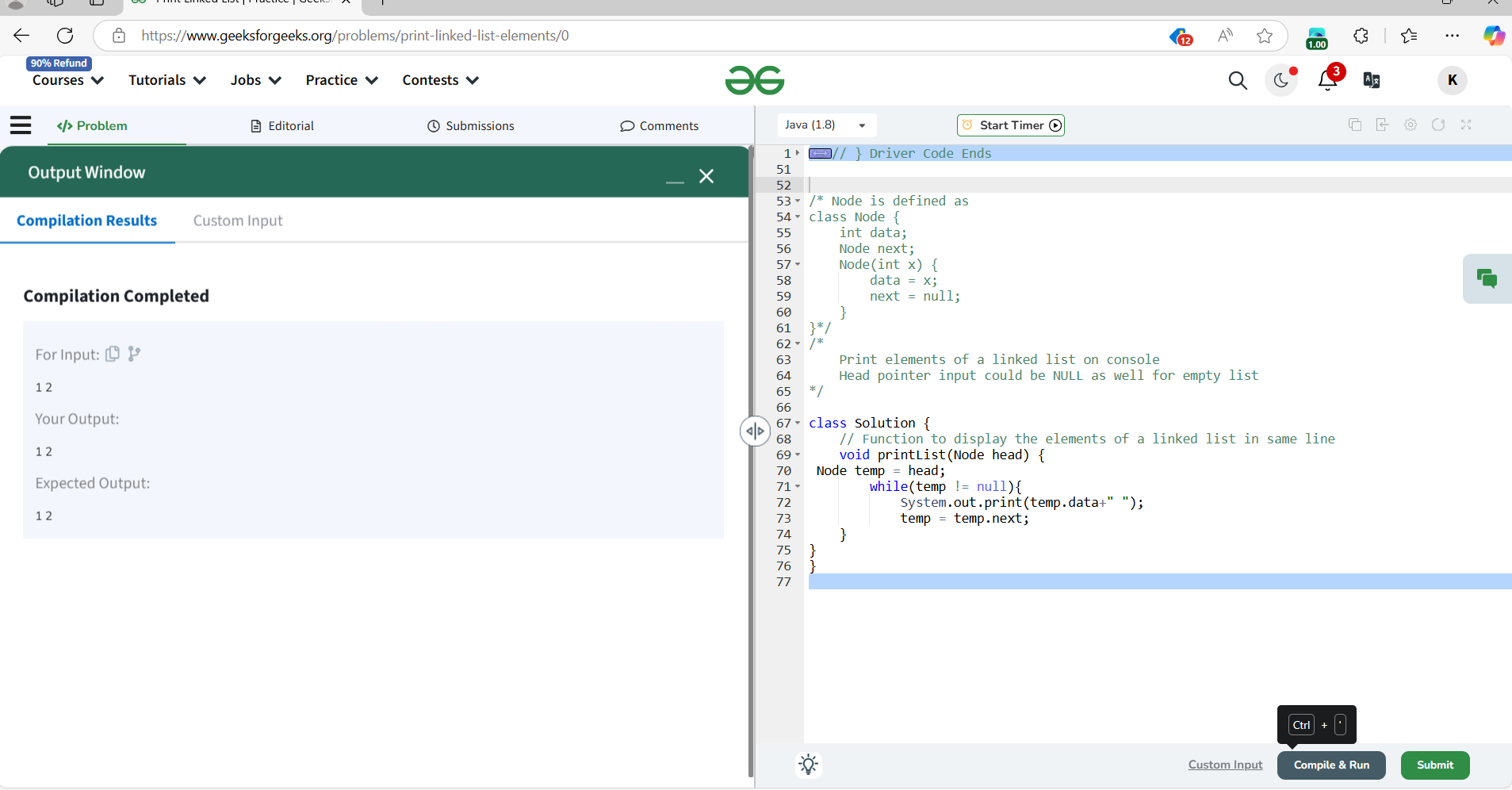
1. **Print linked list**



**Solution:-**

class Solution {

// Function to display the elements of a linked list in same line

void printList(Node head) {

Node temp = head;

while(temp != null){

System.out.print(temp.data+" ");

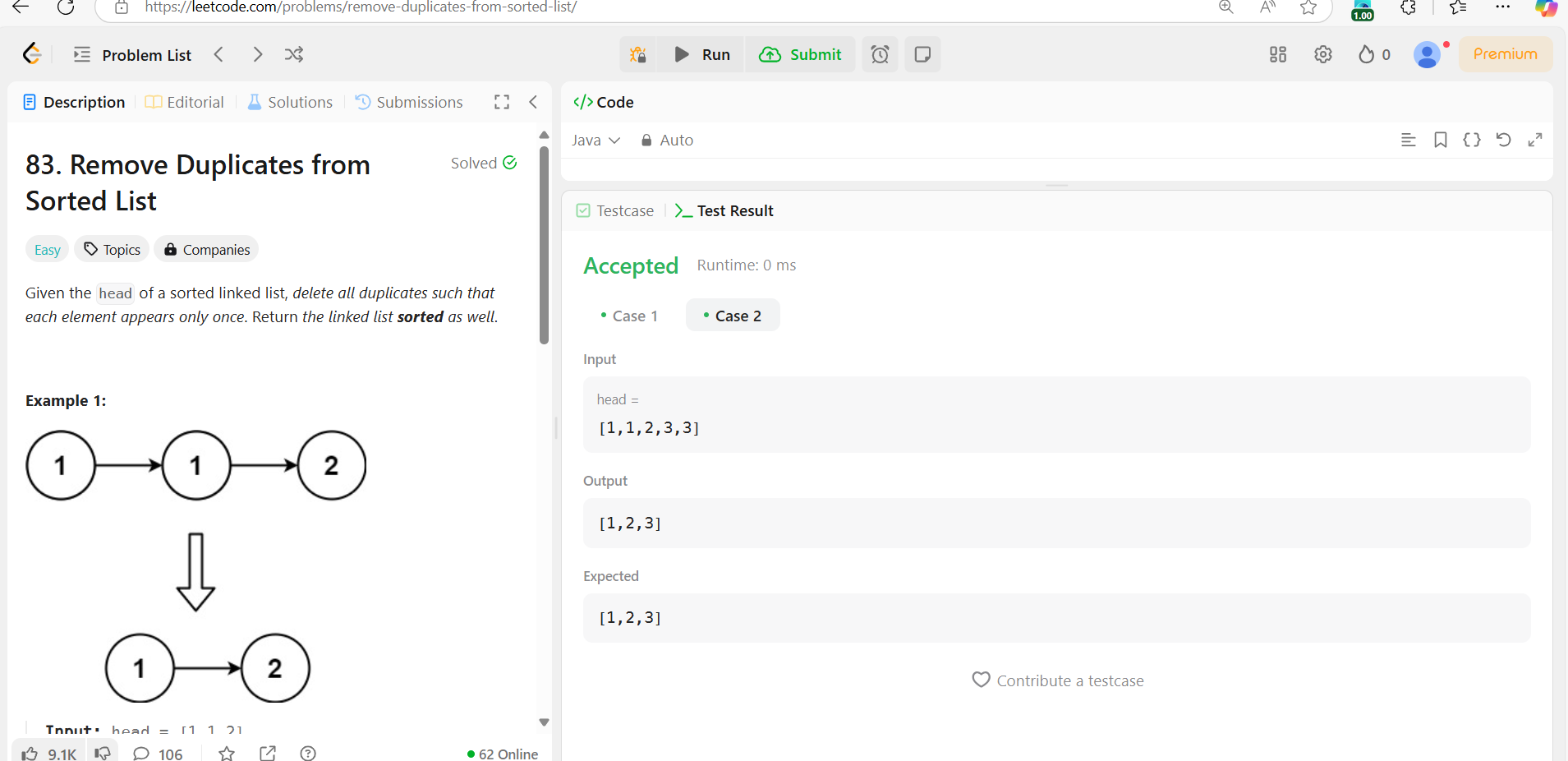
temp = temp.next;

}

}

}

**83.**[**Remove duplicates from a sorted list**](https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/)

****

**Solution:-**

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; // Skip the duplicate

            } else {

                current = current.next; // Move to the next node

            }

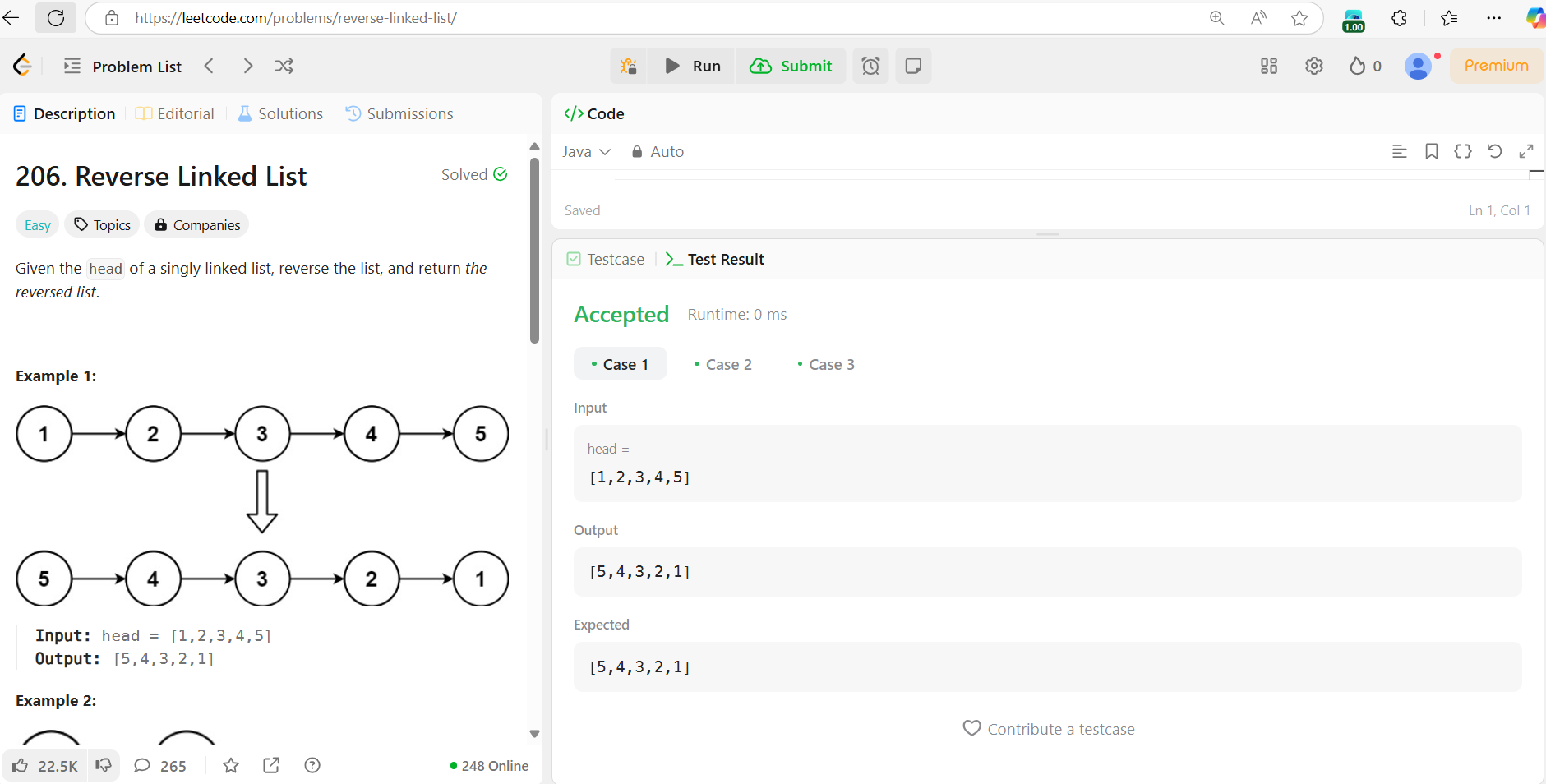
        }

        return head;

    }

}

**206.[ Reverse a linked list] (**[**https://leetcode.com/problems/reverse-linked-list/**](https://leetcode.com/problems/reverse-linked-list/)**)**

****

**Solution:-**

class Solution {

    public ListNode reverseList(ListNode head) {

        ListNode prev = null;

        ListNode next = null;

        ListNode curr = head;

        while(curr != null)

        {

            next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

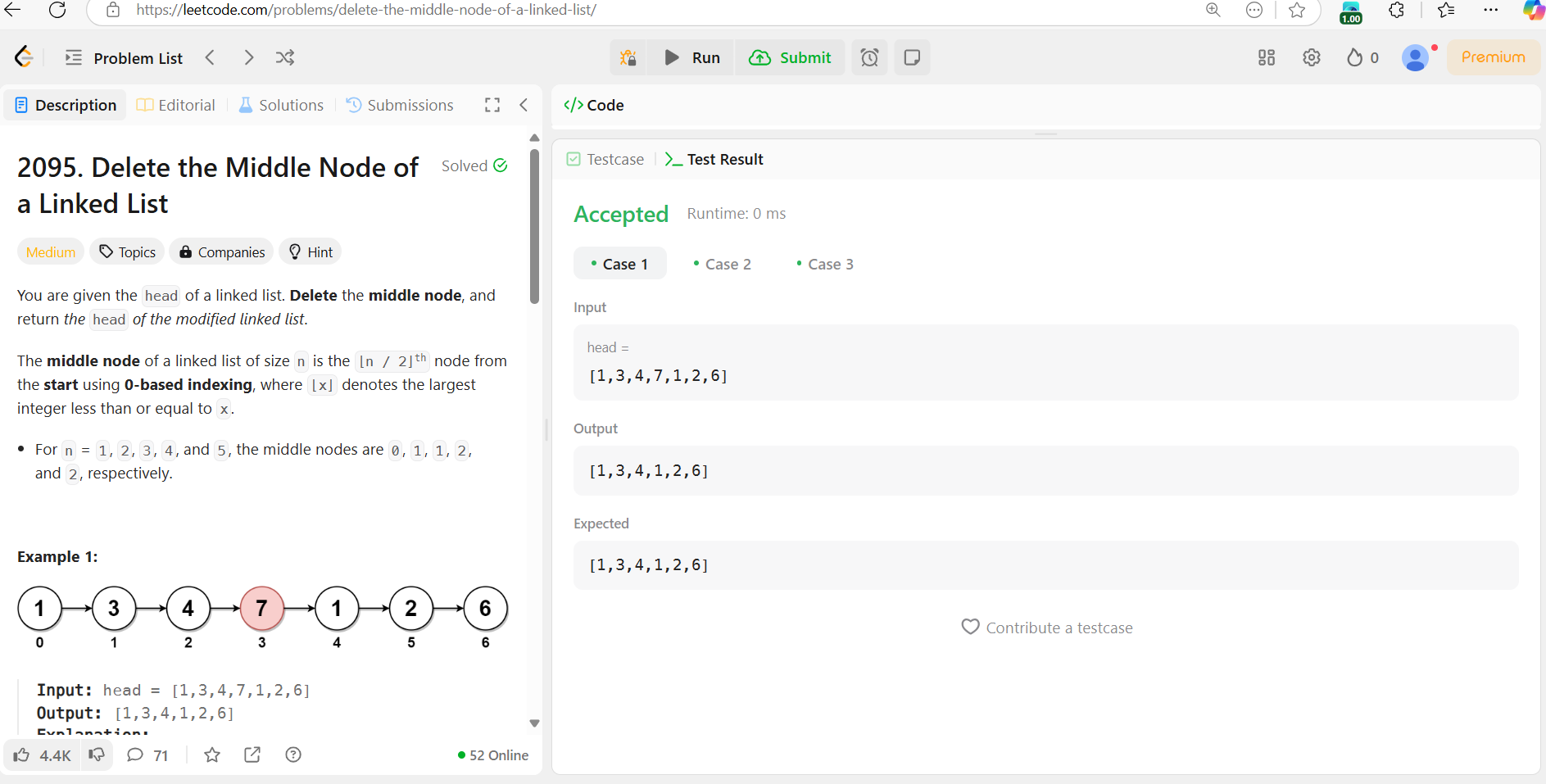
        }

        return prev;

    }

}

**2095.**[**Delete middle node of a list**](https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/)

****

**Solution:-**

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;

        }

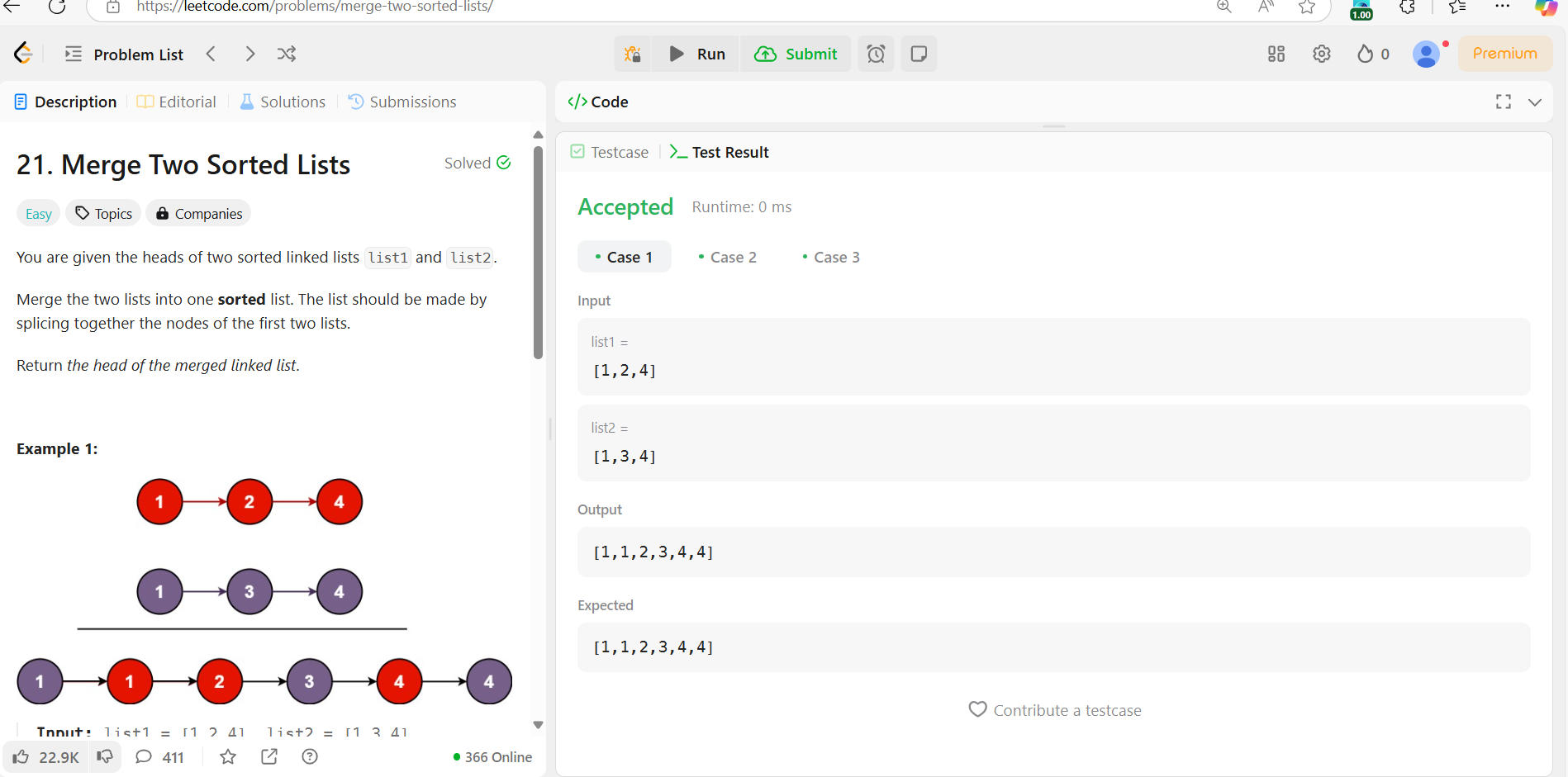
        slow.next = slow.next.next;

        return prev.next;

    }

}

**21.**[**Merge two sorted linked lists**](https://leetcode.com/problems/merge-two-sorted-lists/description/)

****

**Solution:-**

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)

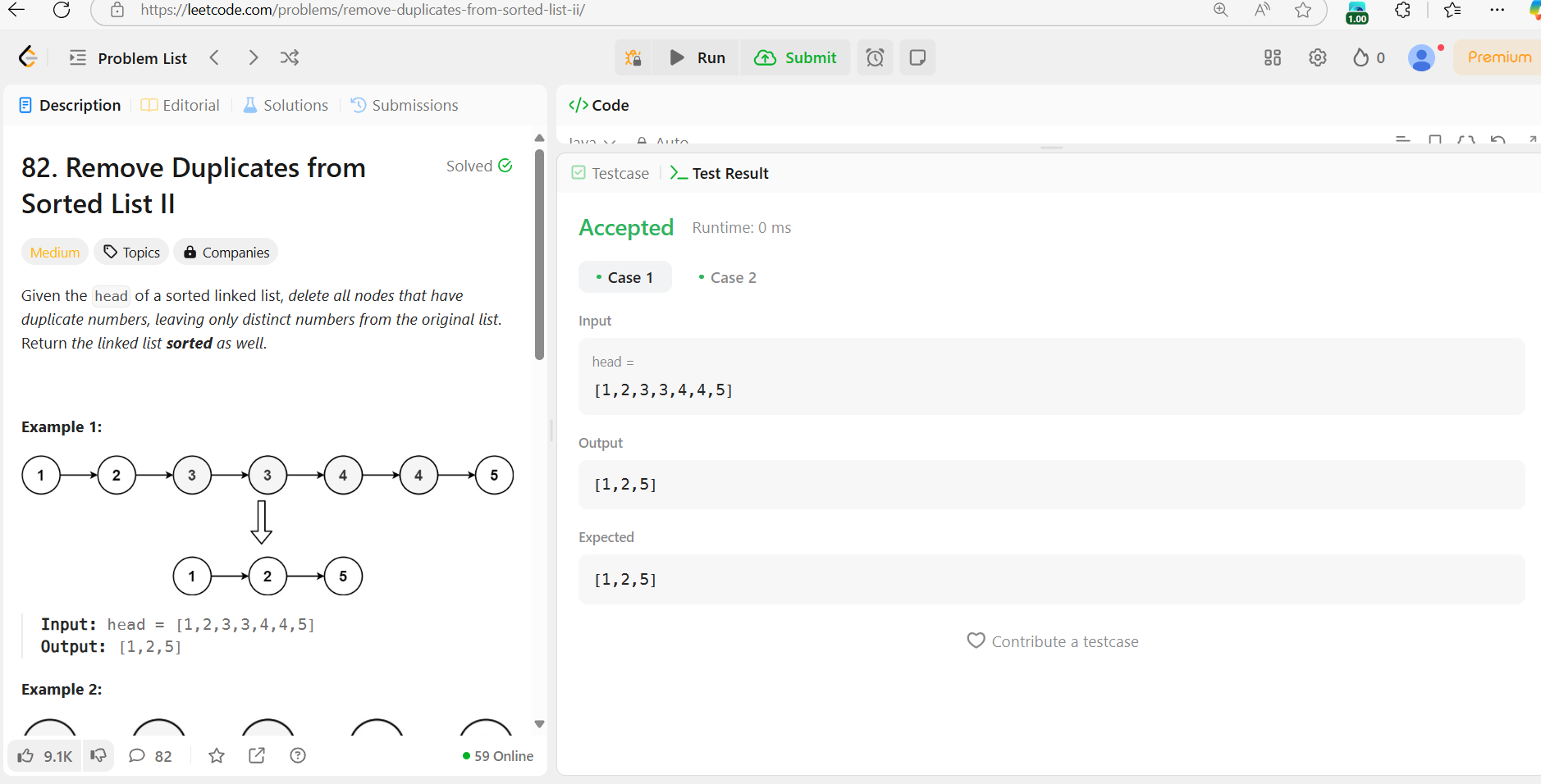
            return list2;

        return list1;

    }

}

**82.**[**Remove duplicates from sorted lists 2**](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/)

****

**Solution:-**

class Solution {

    public ListNode deleteDuplicates(ListNode head) {

        ListNode ans = new ListNode(0);

        ListNode current = ans;

        ListNode temp = head;

        while (temp != null) {

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

                int value = temp.val;

                while (temp != null && temp.val == value) {

                    temp = temp.next;

                }

            } else {

                current.next = temp;

                current = temp;

                temp = temp.next;

            }

        }

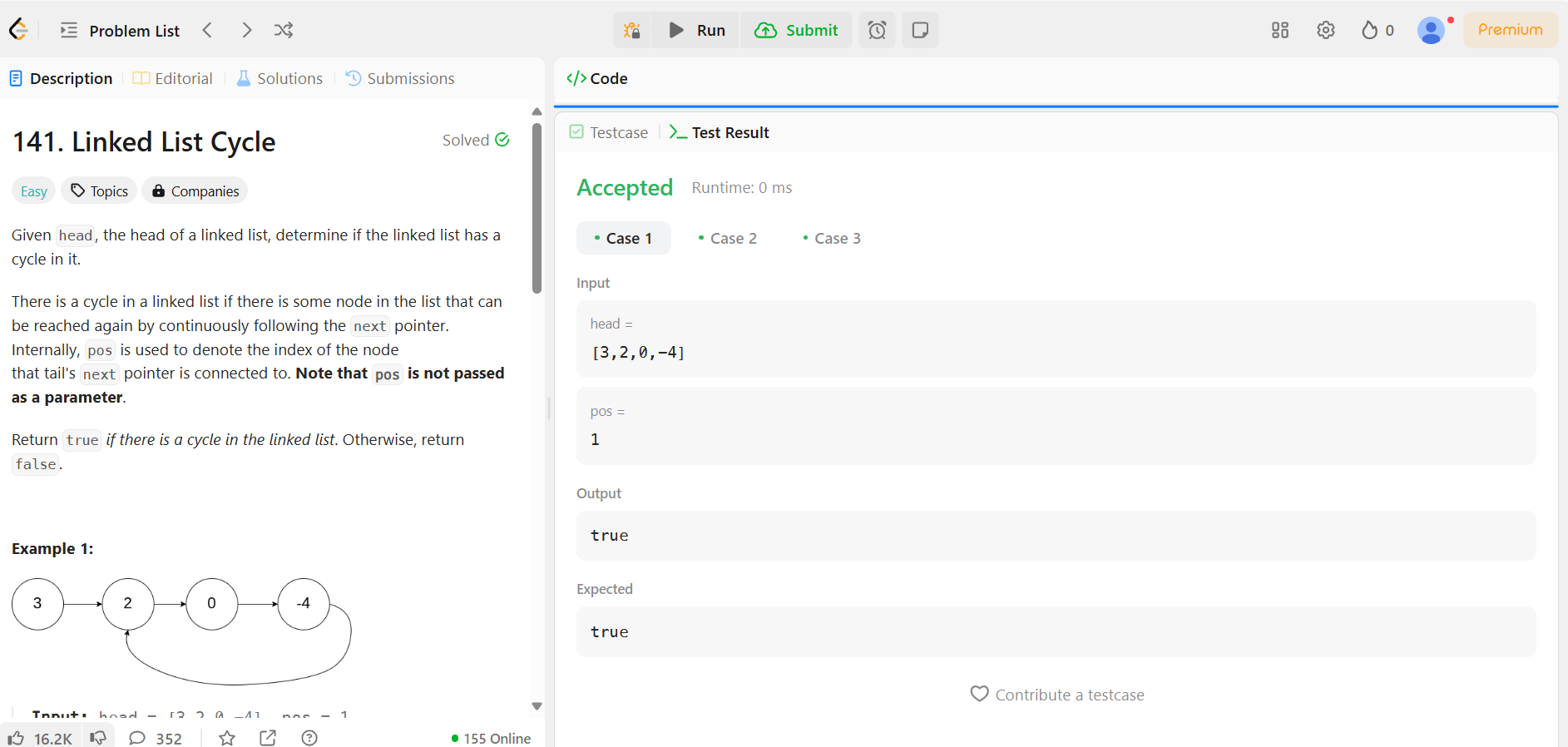
        current.next = null;

        return ans.next;

    }

}

**141.**[**Detect a cycle in a linked list**](https://leetcode.com/problems/linked-list-cycle/description/)

****

**Solution:-**

public class Solution {

    public static boolean hasCycle(ListNode head) {

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

        ListNode slow = head, fast = head;

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

            slow = slow.next;        // Move 1 step

            fast = fast.next.next;   // Move 2 steps

            if (slow == fast) return true;  // Cycle detected

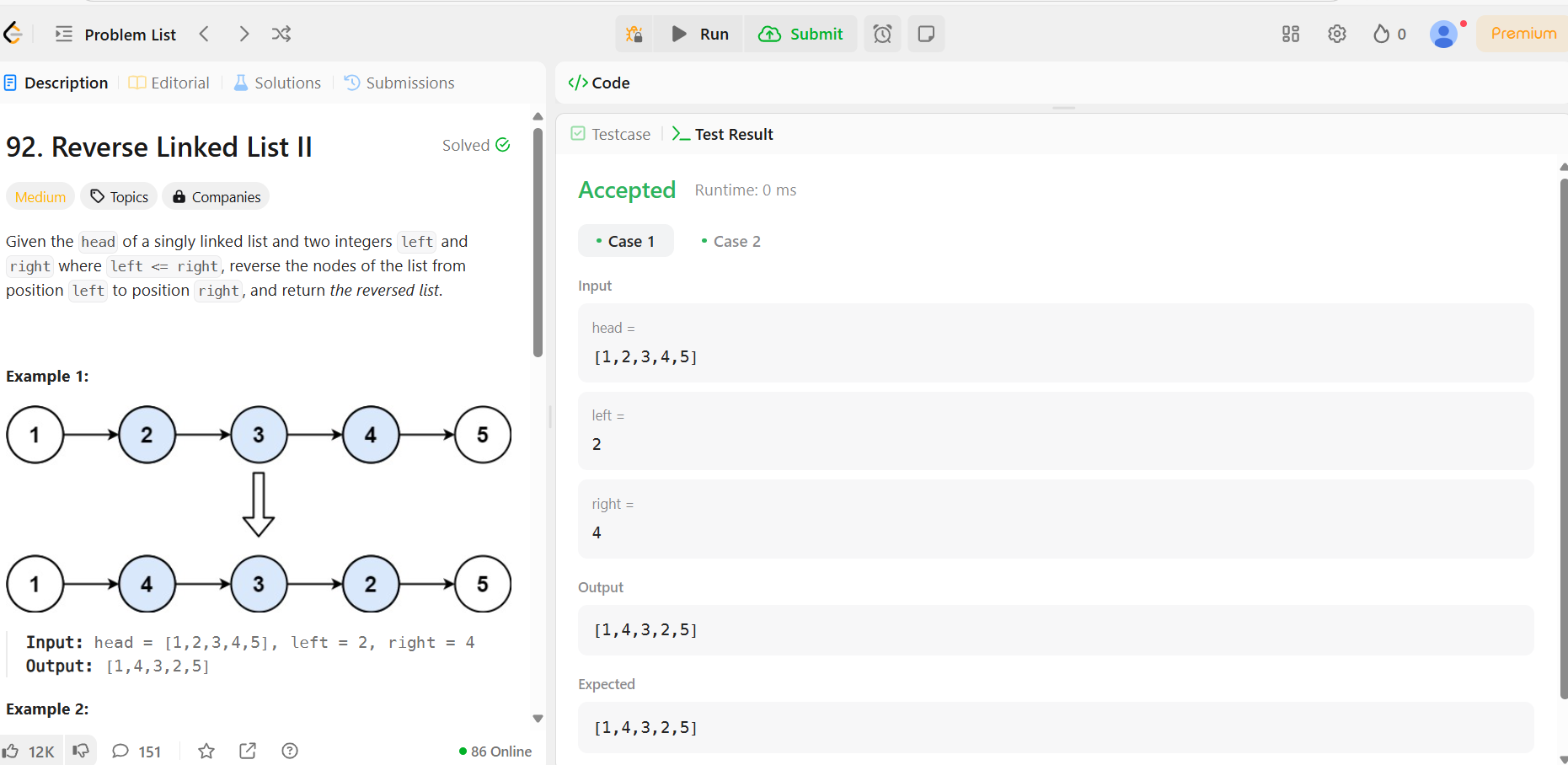
        }

        return false;  // No cycle found

    }

}

**92.**[**Reverse linked list 2**](https://leetcode.com/problems/reverse-linked-list-ii/description/)

****

**Solution:-**

class Solution {

    public static 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 start = prev.next;

        ListNode then = start.next;

        for (int i = 0; i < right - left; i++) {

            start.next = then.next;

            then.next = prev.next;

            prev.next = then;

            then = start.next;

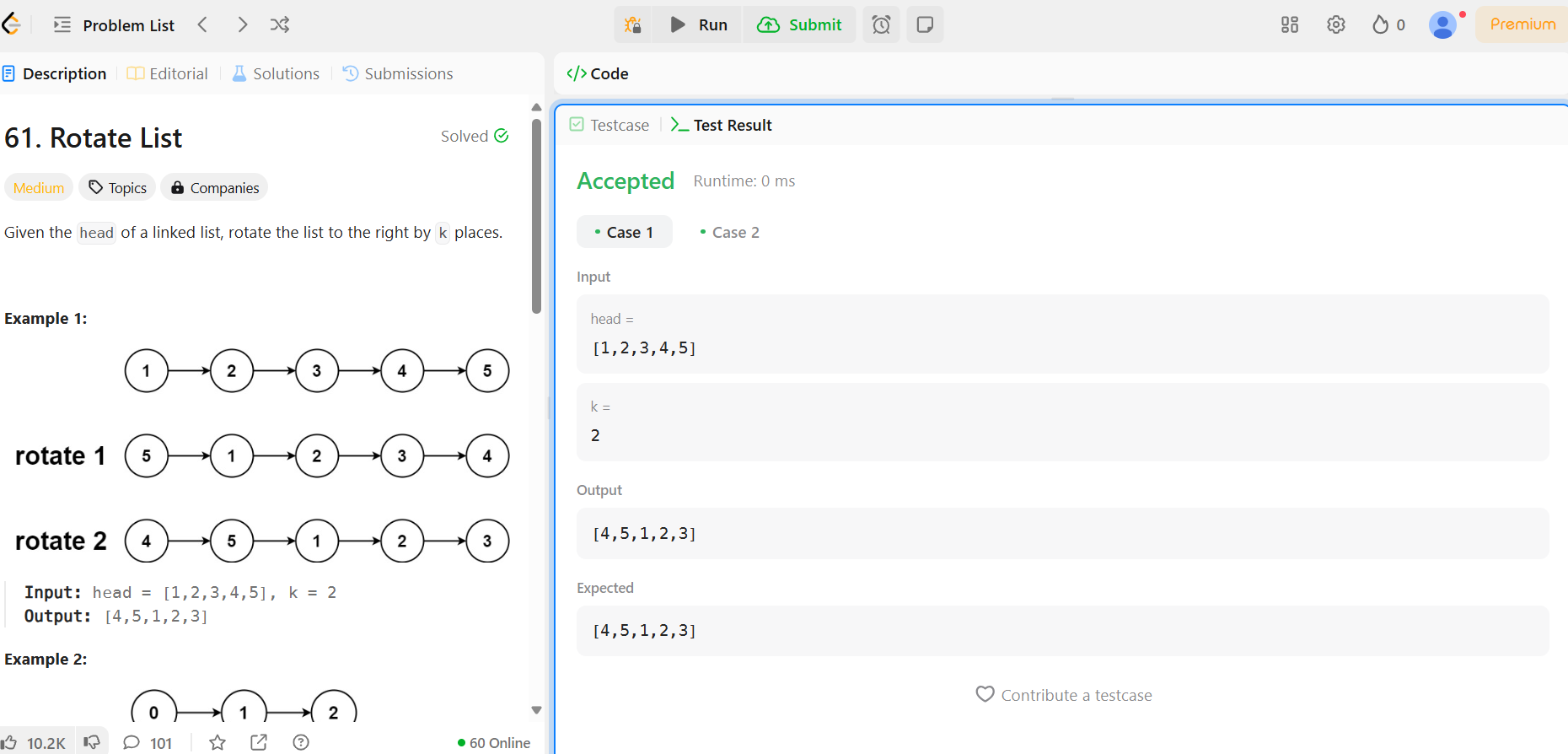
        }

        return dummy.next;

    }

}

**61.**[**rotate a list**](https://leetcode.com/problems/rotate-list/description/)



**Solution:-**

class Solution {

        public static ListNode rotateRight(ListNode head, int k) {

        if (head == null || head.next == null || k == 0) return head;

        ListNode temp = head;

        int length = 1;

        while (temp.next != null) {

            temp = temp.next;

            length++;

        }

        temp.next = head; // Make circular list

        k = k % length;

        ListNode newTail = head;

        for (int i = 0; i < length - k - 1; i++) {

            newTail = newTail.next;

        }

        ListNode newHead = newTail.next;

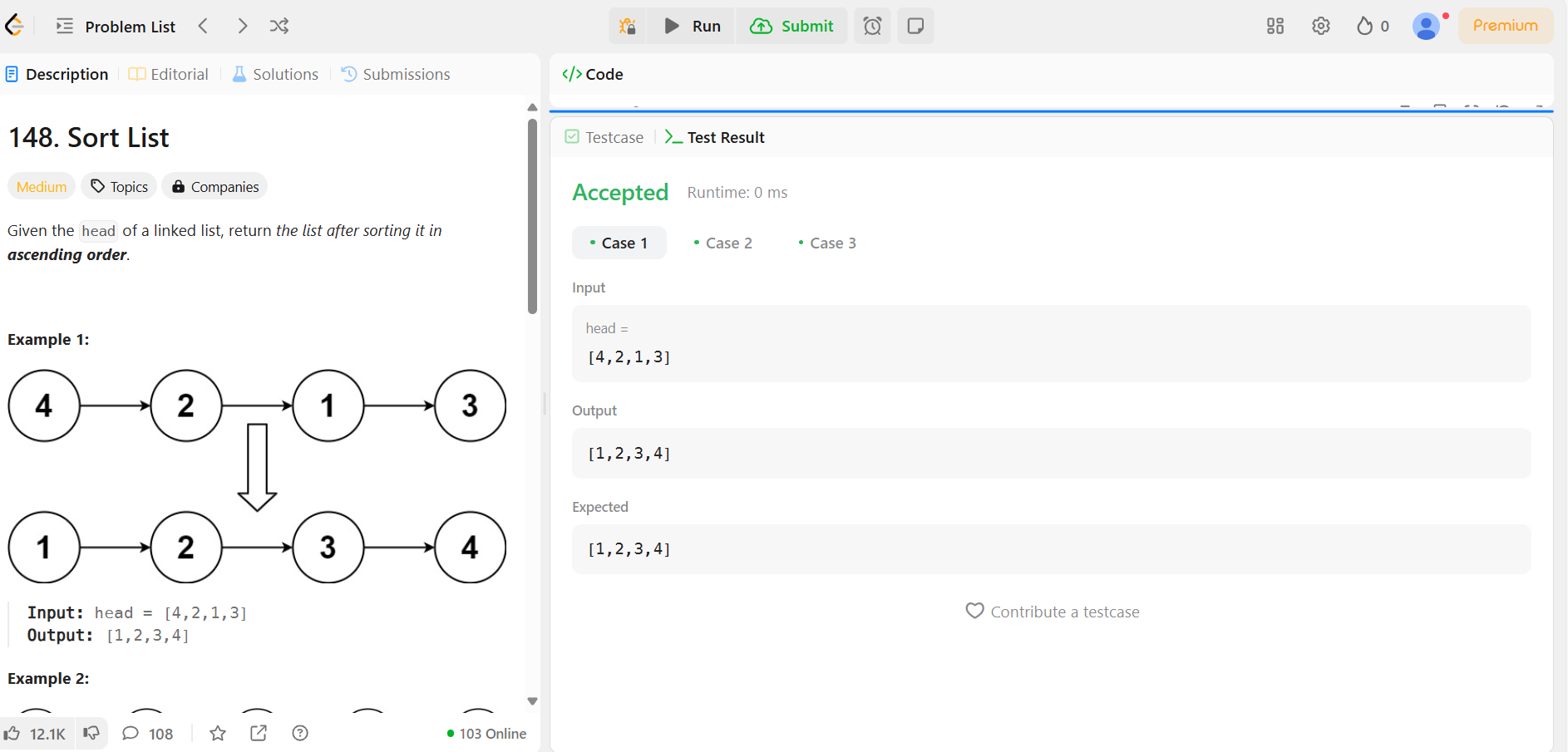
        newTail.next = null;

        return newHead;

    }

}

**148.**[**Sort List**](https://leetcode.com/problems/sort-list/description/)



**Solution:-**

public 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;

    ListNode l1 = sortList(head);

    ListNode l2 = sortList(slow);

    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;

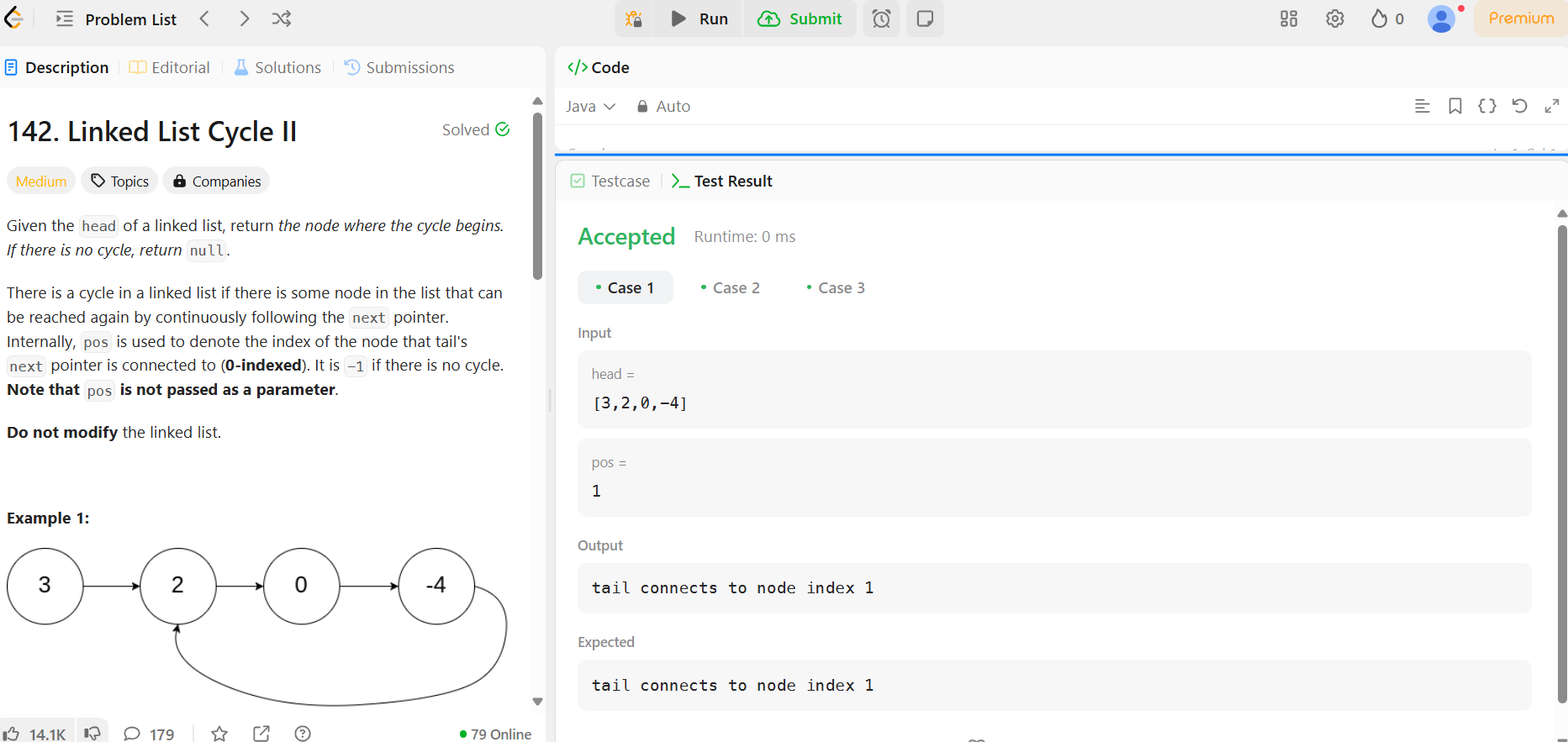
    if (l2 != null)

      p.next = l2;

    return l.next;

  } }

**142.**[**Detect a cycle in a linked list 2**](https://leetcode.com/problems/linked-list-cycle-ii/description/)

****

**Solution:-**

public class Solution {

  public ListNode detectCycle(ListNode head) {

    ListNode slow = head;

    ListNode fast = head;

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

      slow = slow.next;

      fast = fast.next.next;

      if (slow == fast) {

        slow = head;

        while (slow != fast) {

          slow = slow.next;

          fast = fast.next;

        }

        return slow;

      }

    }

    return null;

  }

}