**ADVANCED PROGRAMMING-II**

**ASSIGNMENT-1**

**Q1.** [**Print linked list**](https://www.geeksforgeeks.org/problems/print-linked-list-elements/0)**:**

**CODE:**

class Solution {

public void printList(Node head) {

Node curr = head;

while (curr != null) {

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

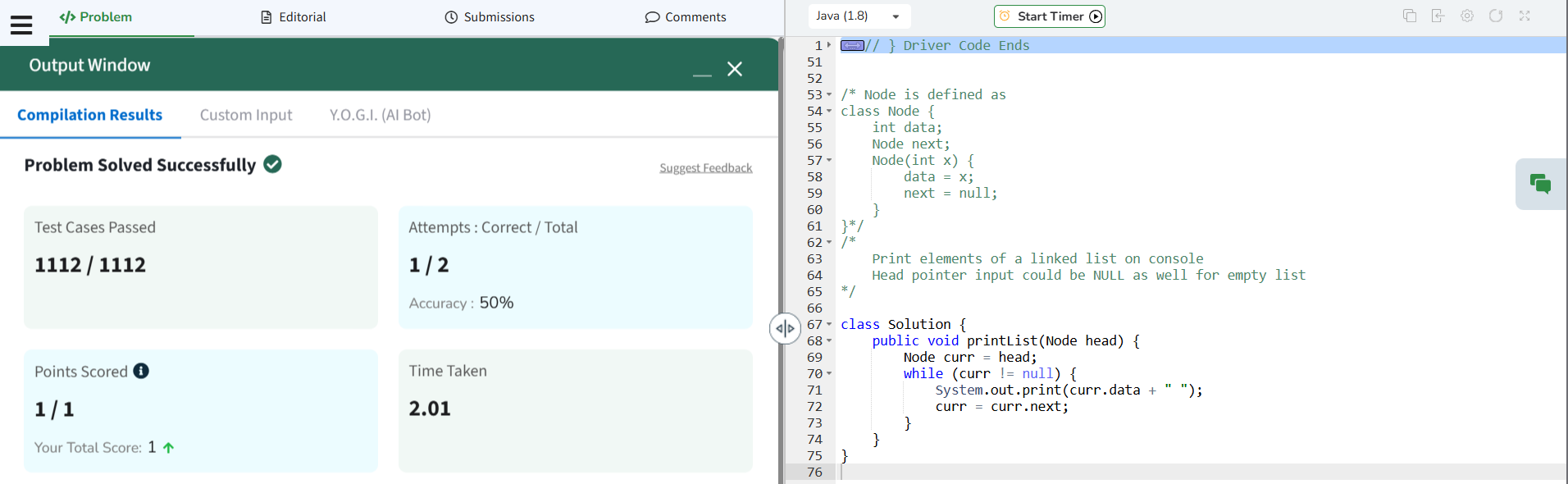
curr = curr.next;

}

}

}

**Screenshot:**

****

**Q2.** [**Remove duplicates from a sorted list**](https://leetcode.com/problems/remove-duplicates-from-sorted-list/description/)**:**

**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;

            }

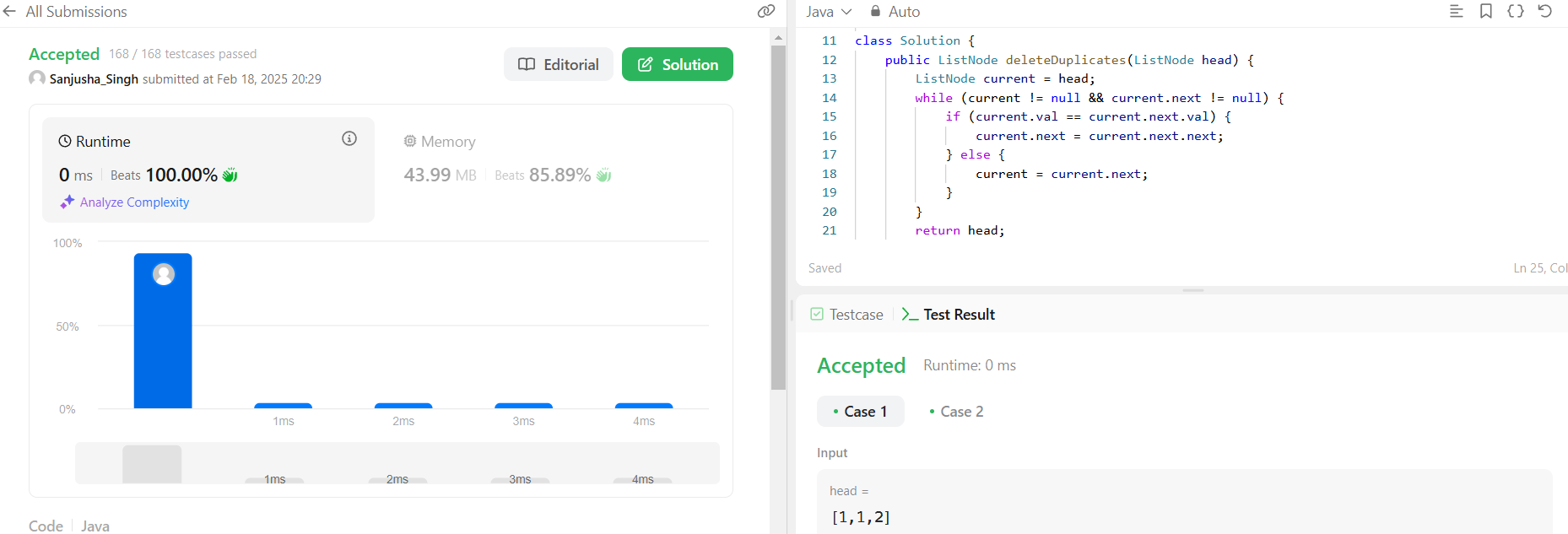
        }

        return head;

    }

}

**SCREENSHOT:**

****

**Q3.Reverse a linked list:**

**CODE:**

class Solution {

    public ListNode reverseList(ListNode head) {

         ListNode curr = head, prev = null, next;

        while (curr != null) {

            next = curr.next;

            curr.next = prev;

            prev = curr;

            curr = next;

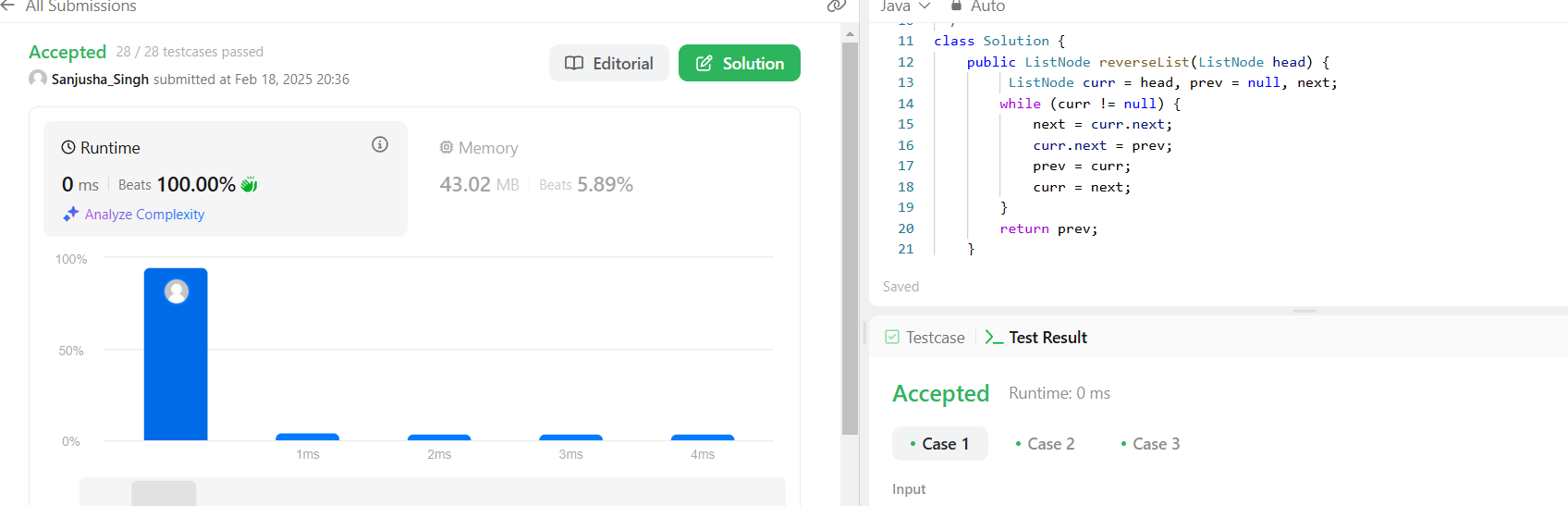
        }

        return prev;

    }

}

**SCREENSHOT:**

****

**Q4.** [**Delete middle node of a list**](https://leetcode.com/problems/delete-the-middle-node-of-a-linked-list/description/)**:**

**CODE:**

class Solution {

    public ListNode deleteMiddle(ListNode head) {

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

            return null;

        }

        ListNode slow = head;

        ListNode fast = head;

        ListNode prev = null;

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

            prev = slow;

            slow = slow.next;

            fast = fast.next.next;

        }

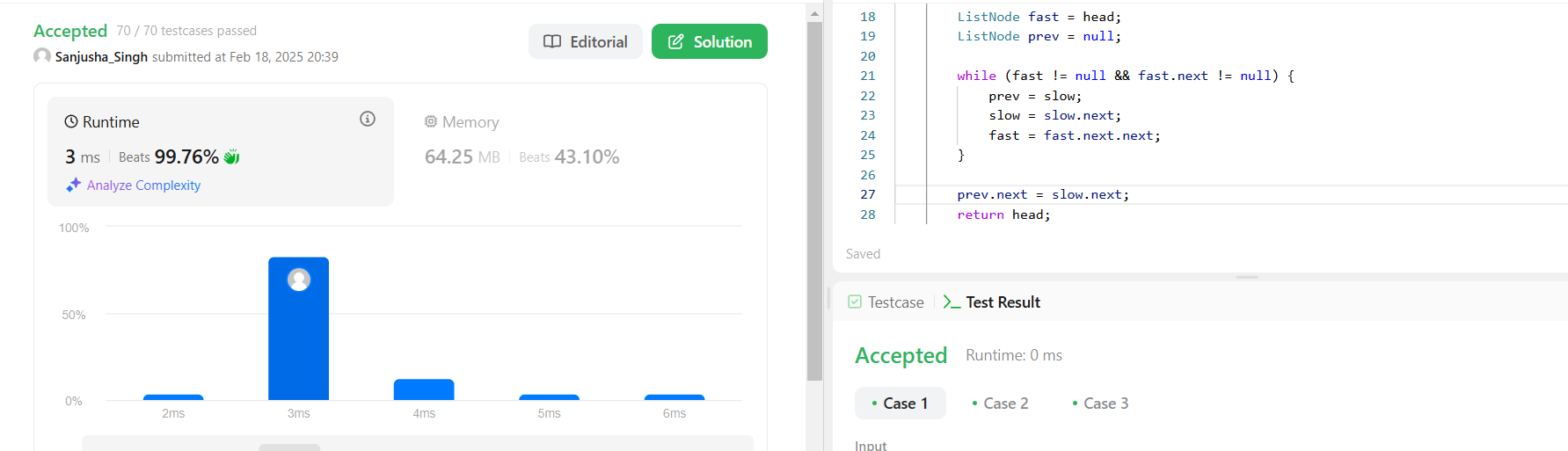
        prev.next = slow.next;

        return head;

    }

}

**SCREENSHOT:**

****

**Q5.** [**Merge two sorted linked lists**](https://leetcode.com/problems/merge-two-sorted-lists/description/)**:**

**CODE:**

class Solution {

    public ListNode mergeTwoLists(ListNode l1, ListNode l2) {

        if (l1 == null) return l2;

        if (l2 == null) return l1;

        if (l1.val < l2.val) {

            l1.next = mergeTwoLists(l1.next, l2);

            return l1;

        } else {

            l2.next = mergeTwoLists(l1, l2.next);

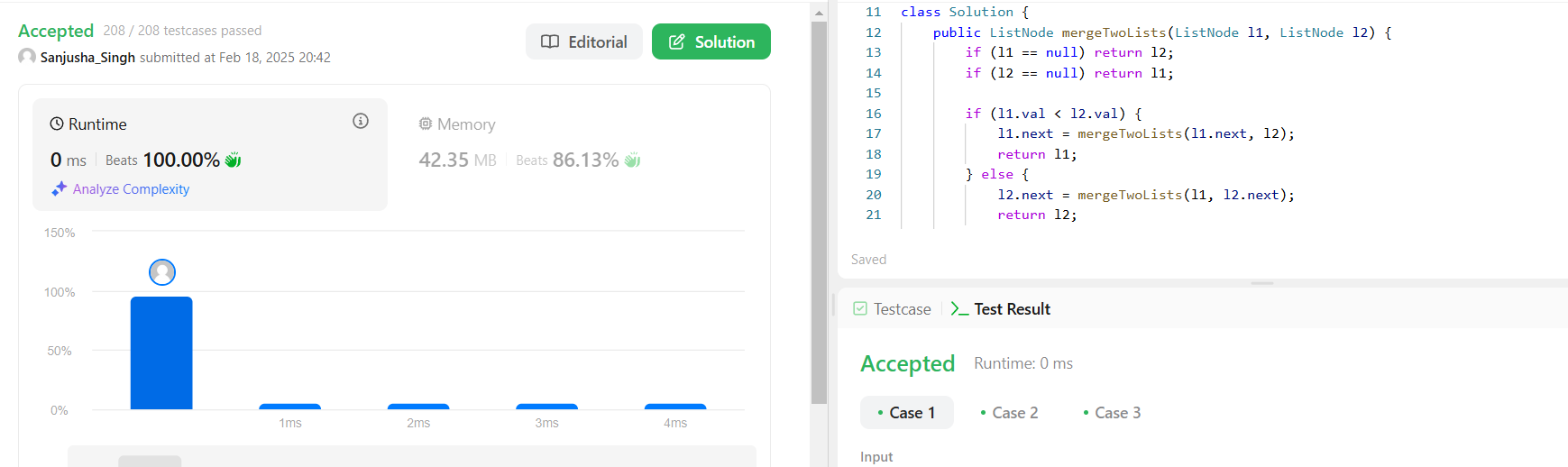
            return l2;

        }

    }

}

**SCREENSHOT:**

****

**Q6.** [**Remove duplicates from sorted lists 2**](https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/)**:**

**CODE:**

class Solution {

    public ListNode deleteDuplicates(ListNode head) {

         ListNode temp = new ListNode(0);

        temp.next = head;

        ListNode prev = temp;

        while (head != null) {

            boolean duplicate = false;

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

                duplicate = true;

                head = head.next; // Skip duplicate nodes

            }

            if (duplicate) {

                prev.next = head.next;

            } else {

                prev = prev.next;

            }

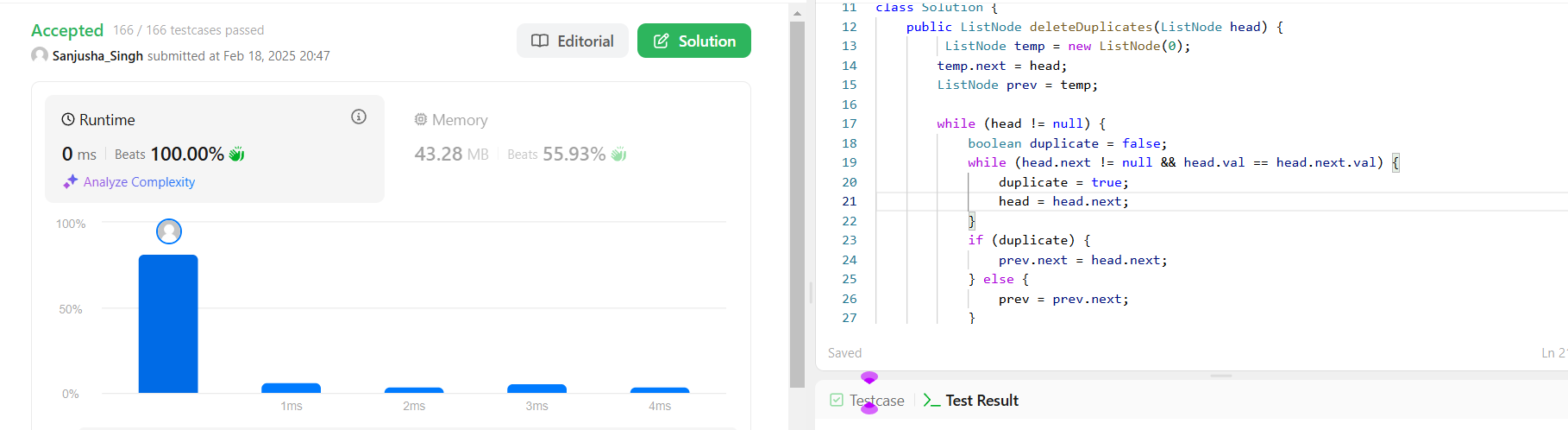
            head = head.next;

        }

        return temp.next;

    }

}

**SCREENSHOT: **

**Q7.** [**Detect a cycle in a linked list**](https://leetcode.com/problems/linked-list-cycle/description/)**:**

**CODE:**

public class Solution {

    public boolean hasCycle(ListNode head) {

          ListNode fast = head, slow = head;

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

            slow = slow.next;

            fast = fast.next.next;

            if (slow == fast) {

                return true;

            }

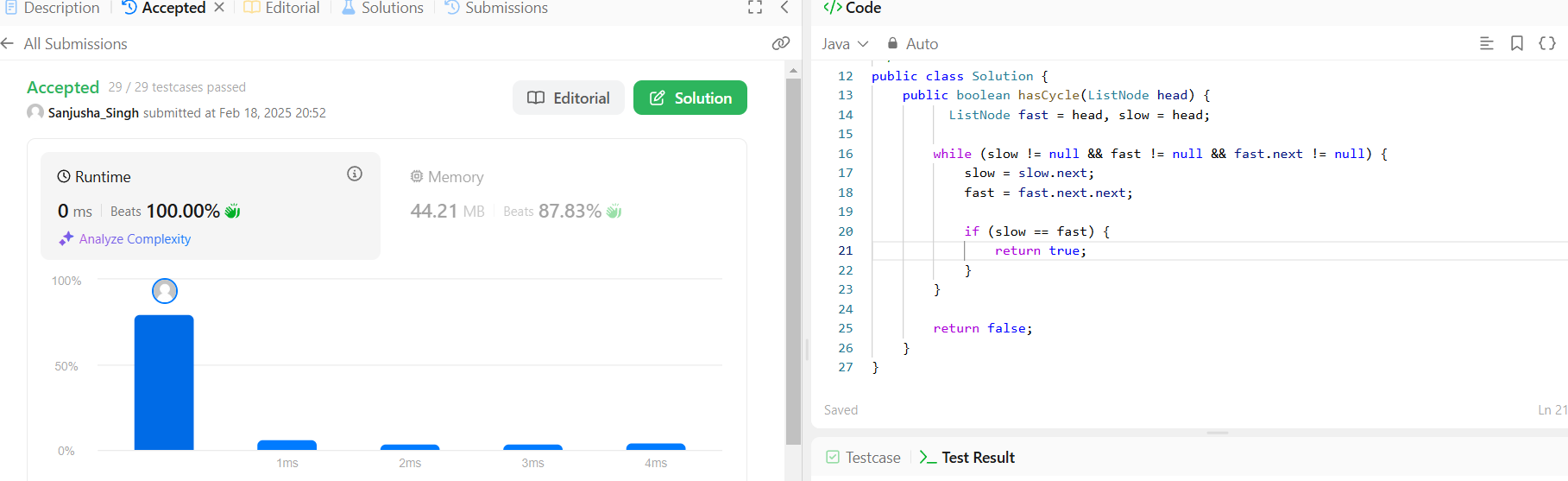
        }

        return false;

    }

}

**SCREENSHOT:**

****

**Q8.** [**Reverse linked list 2**](https://leetcode.com/problems/reverse-linked-list-ii/description/)**:**

**CODE:**

class Solution {

    public ListNode reverseBetween(ListNode head, int left, int right) {

        if (head == null || left == right) return head;

        ListNode temp = new ListNode(0);

        temp.next = head;

        ListNode prev = temp;

        for (int i = 1; i < left; i++) {

            prev = prev.next;

        }

        ListNode curr = prev.next;

        ListNode next = null;

        // Reverse the sublist between left and right

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

            next = curr.next;

            curr.next = next.next;

            next.next = prev.next;

            prev.next = next;

        }

        return temp.next; // Return new head

    }

**SCREENSHOT: **

**Q9.** [**rotate a list**](https://leetcode.com/problems/rotate-list/description/)**:**

**CODE:**

 \*/

class Solution {

    public ListNode rotateRight(ListNode head, int k) {

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

        // Find the length of the list

        int length = 1;

        ListNode tail = head;

        while (tail.next != null) {

            tail = tail.next;

            length++;

        }

        // Reduce k to within the length of the list

        k = k % length;

        if (k == 0) return head;

        tail.next = head;

        // Find the new tail (length - k - 1 steps from head)

        ListNode newTail = head;

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

            newTail = newTail.next;

        }

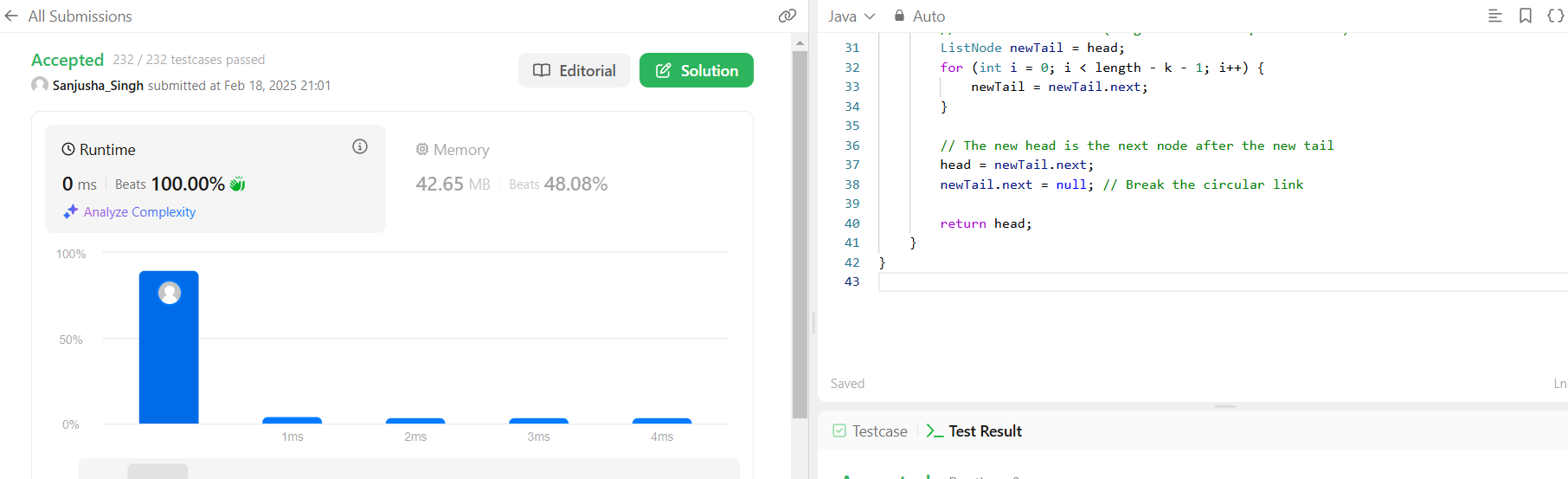
        head = newTail.next;

        newTail.next = null;

        return head;

    }

}

**SCREENSHOT:** 

**Q10.** [**Sort List**](https://leetcode.com/problems/sort-list/description/)**:**

**CODE:**

class Solution {

    public ListNode merge(ListNode left, ListNode right) {

        if (left == null) return right;

        if (right == null) return left;

        ListNode dummy = new ListNode(0);

        ListNode temp = dummy;

        while (left != null && right != null) {

            if (left.val < right.val) {

                temp.next = left;

                left = left.next;

            } else {

                temp.next = right;

                right = right.next;

            }

            temp = temp.next;

        }

        if (left != null) temp.next = left;

        if (right != null) temp.next = right;

        return dummy.next;

    }

    public ListNode sortList(ListNode head) {

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

        ListNode slow = head;

        ListNode fast = head.next;

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

            slow = slow.next;

            fast = fast.next.next;

        }

        ListNode mid = slow.next;

        slow.next = null;

        ListNode left = sortList(head);

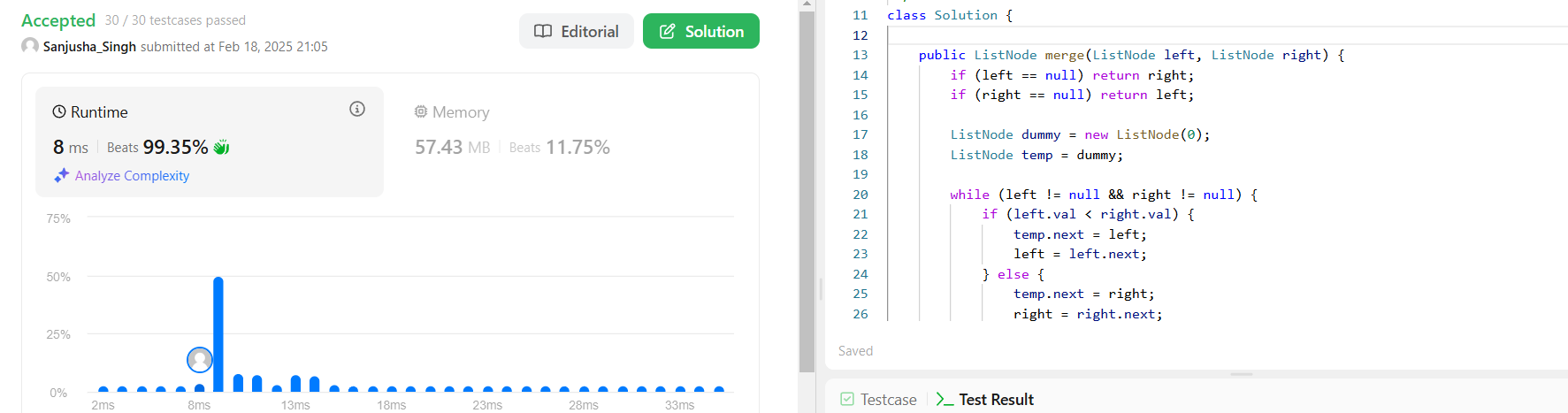
        ListNode right = sortList(mid);

        return merge(left, right);

    }

}

**SCREENSHOT:**

****

**Q11.** [**Detect a cycle in a linked list 2**](https://leetcode.com/problems/linked-list-cycle-ii/description/)**:**

**CODE:**

class Solution {

    public ListNode detectCycle(ListNode head) {

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

        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;

            }

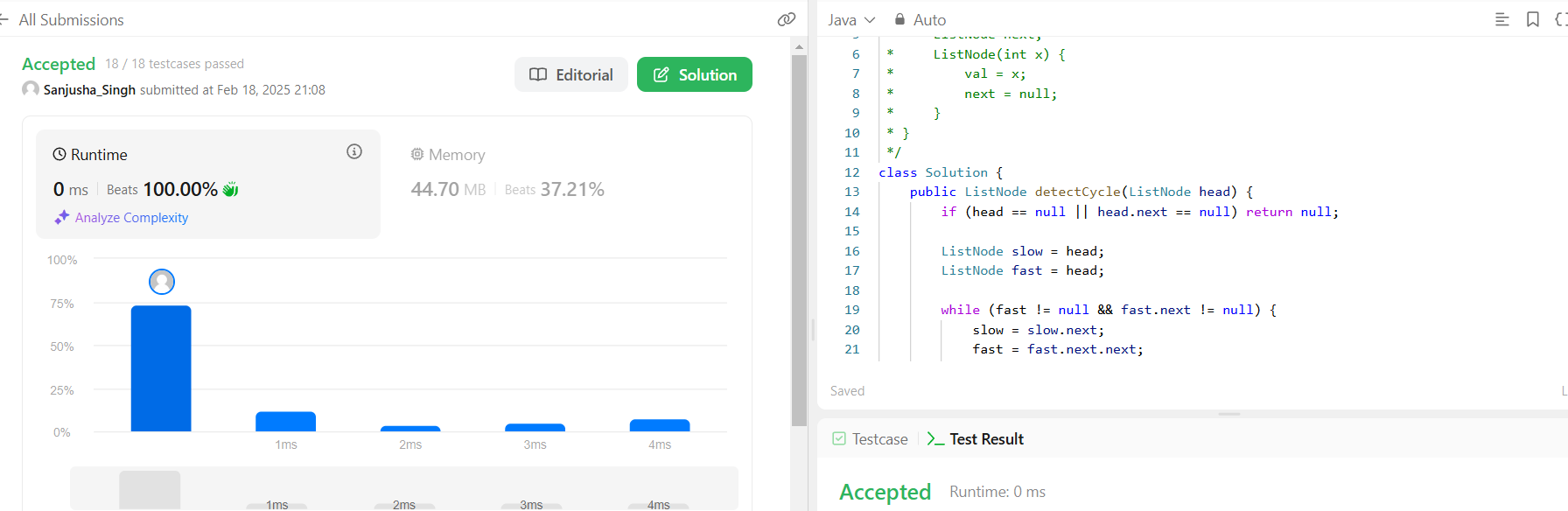
        }

        return null;

    }

}

**SCREENSHOT:**

****