**Name –** Sidhant

**UID –** 22BCS15880

**Section –** 601-‘B

**Experiment –** 2

1. **Print linked list:**

**Problem Link :** https://www.geeksforgeeks.org/problems/print-linked-list-elements/0

**Solution :**

class Solution {

void printList(Node head) {

Node temp = head;

while(temp != null) {

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

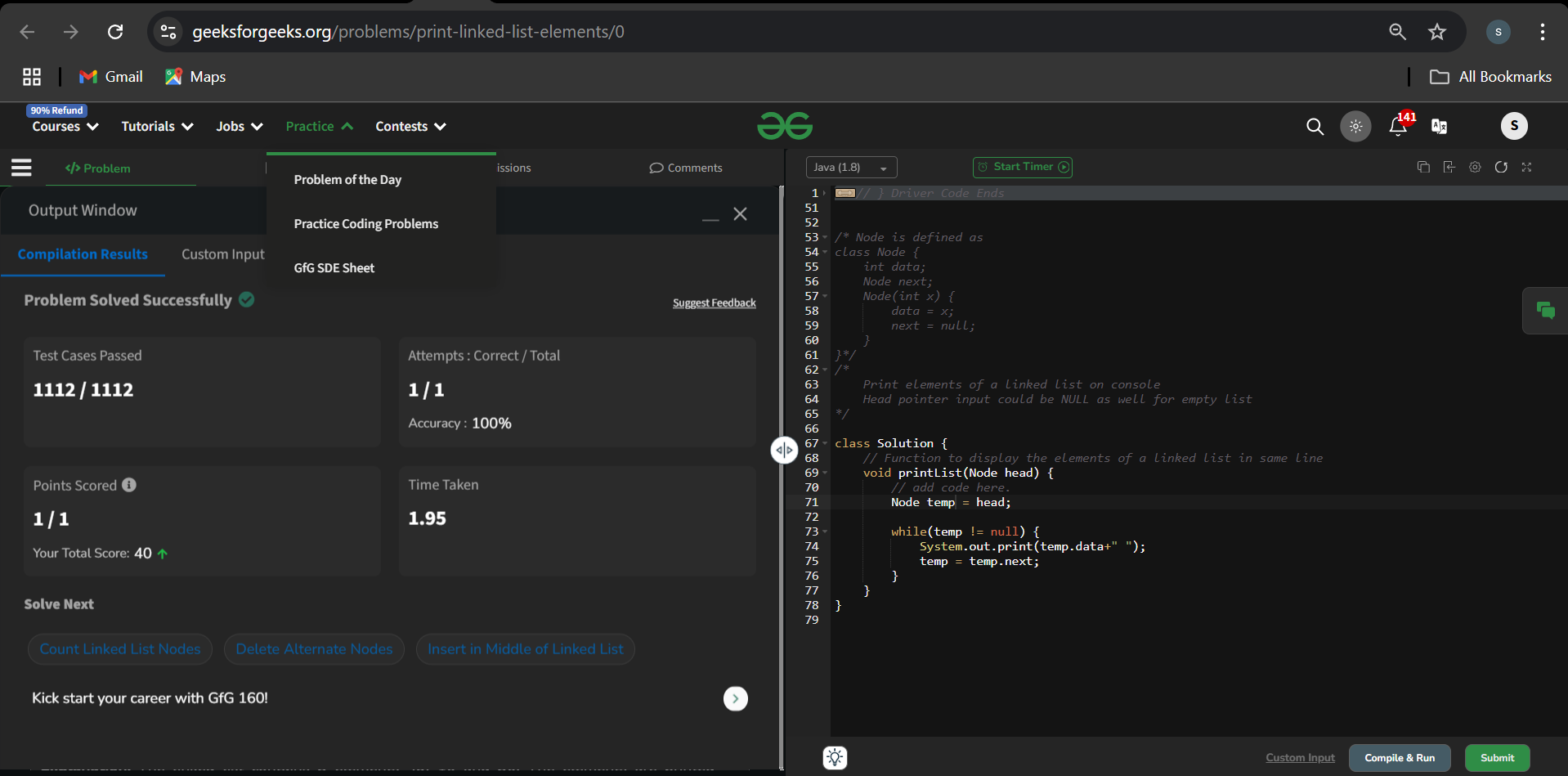
temp = temp.next;

}

}

}

**Output:**



1. **Remove duplicates from a sorted list:**

**Problem Link :** https://leetcode.com/problems/remove-duplicates-from-sorted-list/

**Solution:**

class Solution {

public ListNode deleteDuplicates(ListNode head) {

ListNode node = head;

if(node==null) return node;

while(node.next!=null){

if(node.val==node.next.val){

node.next=node.next.next;

}

else{

node = node.next;

}

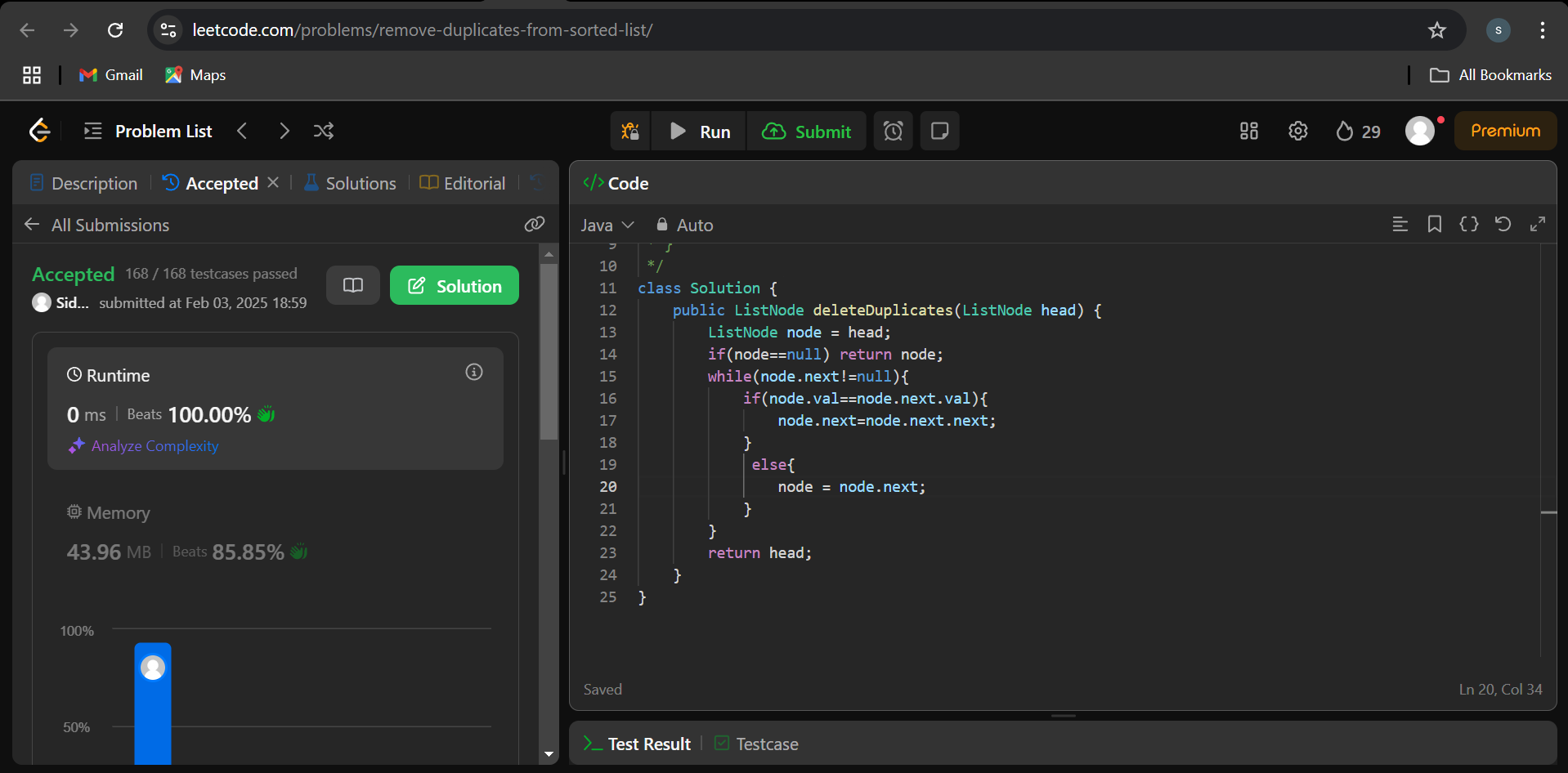
}

return head;

}

}

**Output:**



1. **Reverse a linked list:**

**Problem Link :** https://leetcode.com/problems/reverse-linked-list/

**Solution:**

class Solution {

public ListNode reverseList(ListNode head) {

ListNode prev = null;

ListNode curr = head;

ListNode next;

while(curr != null) {

next = curr.next;

curr.next = prev;

prev = curr;

curr = next;

}

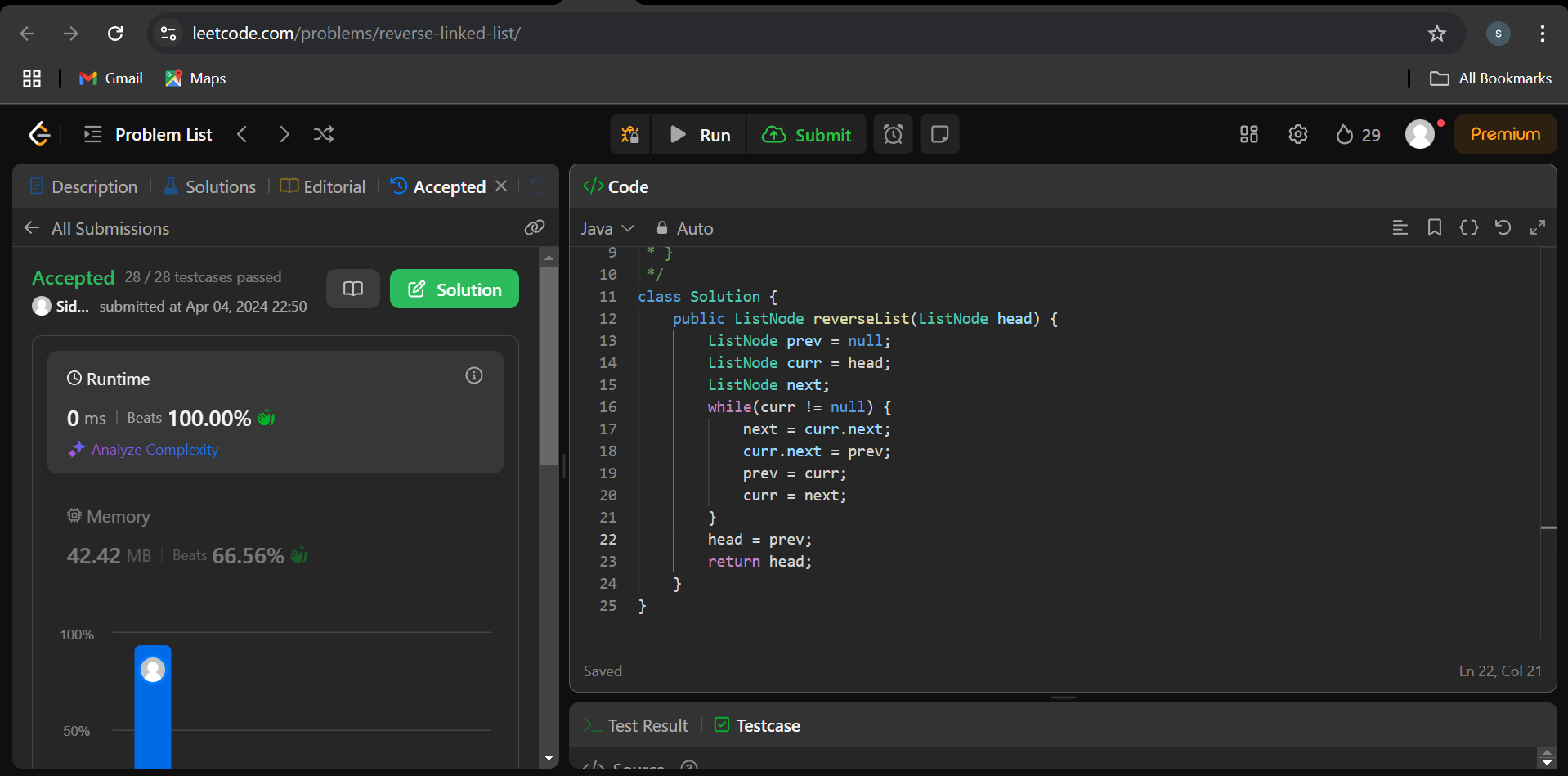
head = prev;

return head;

}

}

**Output:**



1. **Delete middle node of a list:**

**Problem Link :** https://leetcode.com/problems/reverse-linked-list/

**Solution:**

class Solution {

public ListNode getMid(ListNode head) {

ListNode slow = head;

ListNode fast = head;

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

slow = slow.next;

fast = fast.next.next;

}

return slow;

}

public ListNode deleteMiddle(ListNode head) {

ListNode midNumber = getMid(head);

// Node temp1 = slow;

ListNode temp = head;

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

return null;

}

while(temp.next != midNumber) {

temp = temp.next;

}

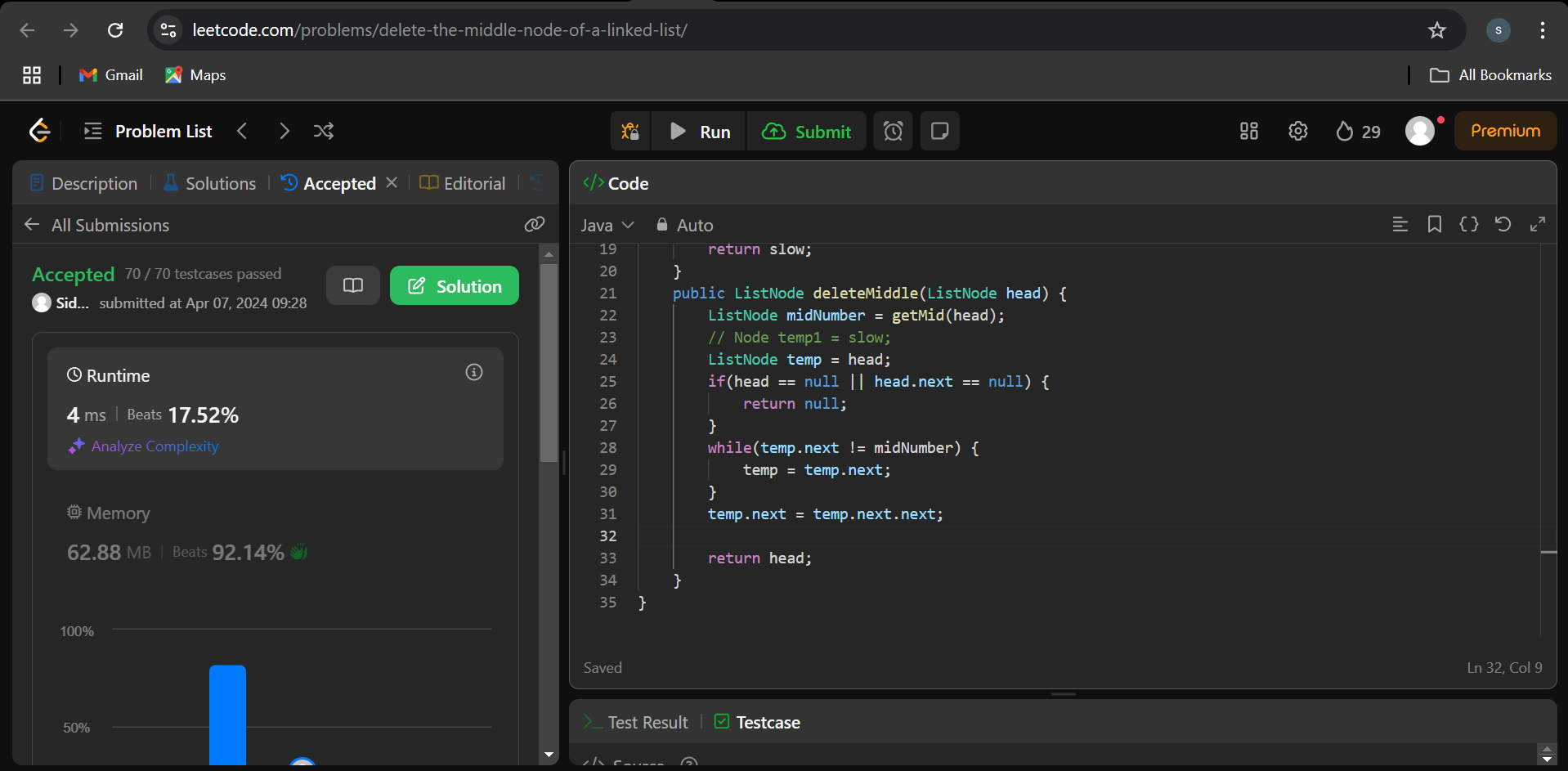
temp.next = temp.next.next;

return head;

}

}

**Output:**



1. **Merge two sorted linked lists:**

**Problem Link :** <https://leetcode.com/problems/merge-two-sorted-lists/description/>

**Solution:**

class Solution {

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

ListNode mergedLL = new ListNode(-1);

ListNode temp = mergedLL;

while(list1 != null && list2 != null) {

if(list1.val <= list2.val) {

temp.next = list1;

list1 = list1.next;

temp = temp.next;

} else {

temp.next = list2;

list2 = list2.next;

temp = temp.next;

}

}

while(list1 != null) {

temp.next = list1;

list1 = list1.next;

temp = temp.next;

}

while(list2 != null) {

temp.next = list2;

list2 = list2.next;

temp = temp.next;

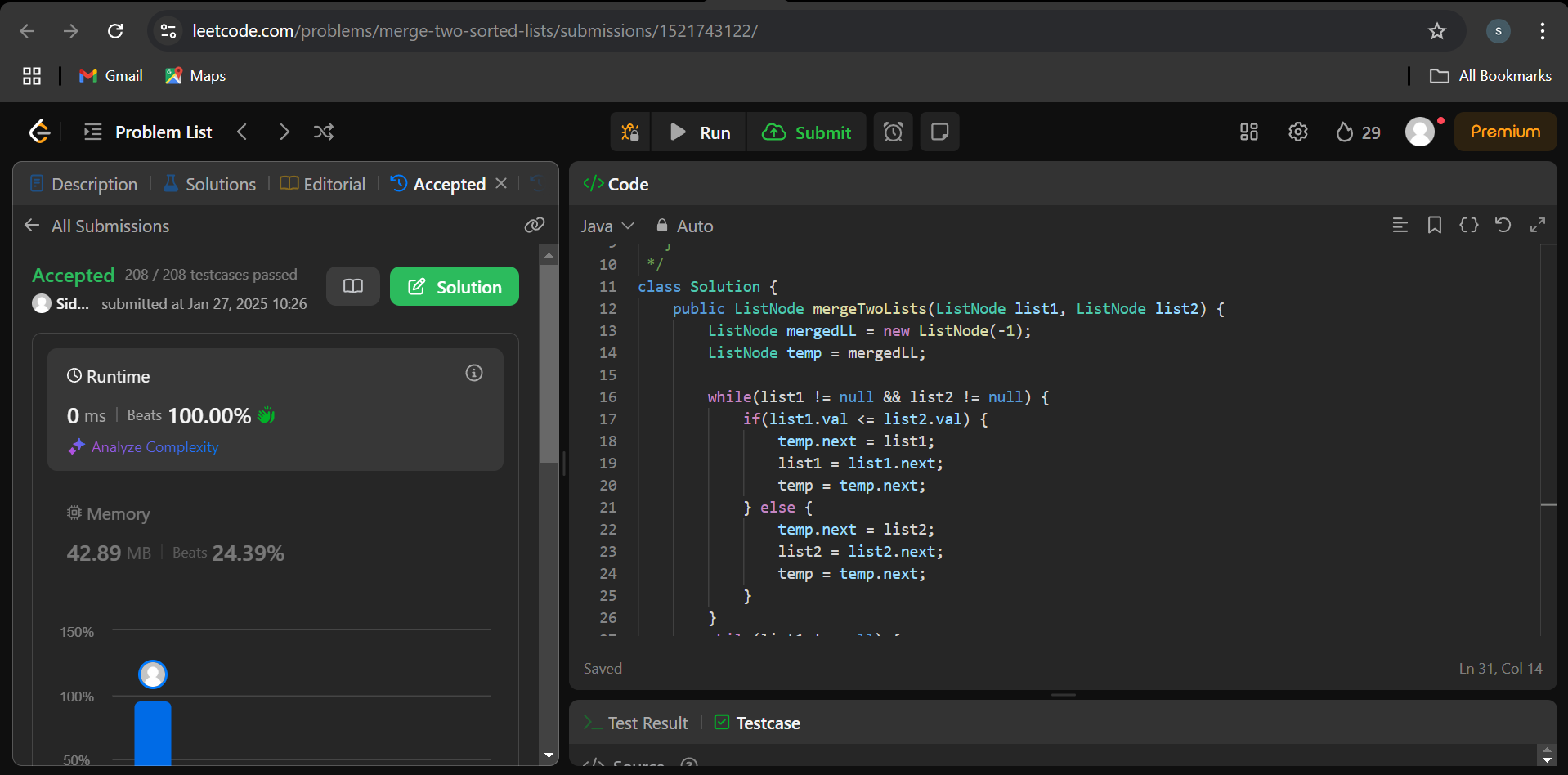
}

return mergedLL.next;

}

}

**Output:**

****

1. **Remove duplicates from sorted lists 2:**

**Problem Link :** https://leetcode.com/problems/remove-duplicates-from-sorted-list-ii/description/

**Solution:**

class Solution {

public ListNode deleteDuplicates(ListNode head) {

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

ListNode res = new ListNode(0,head);

ListNode prev = res;

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

if(head.next.val == head.val){

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

head = head.next;

}

prev.next = head.next;

}else{

prev = prev.next;

}

head = head.next;

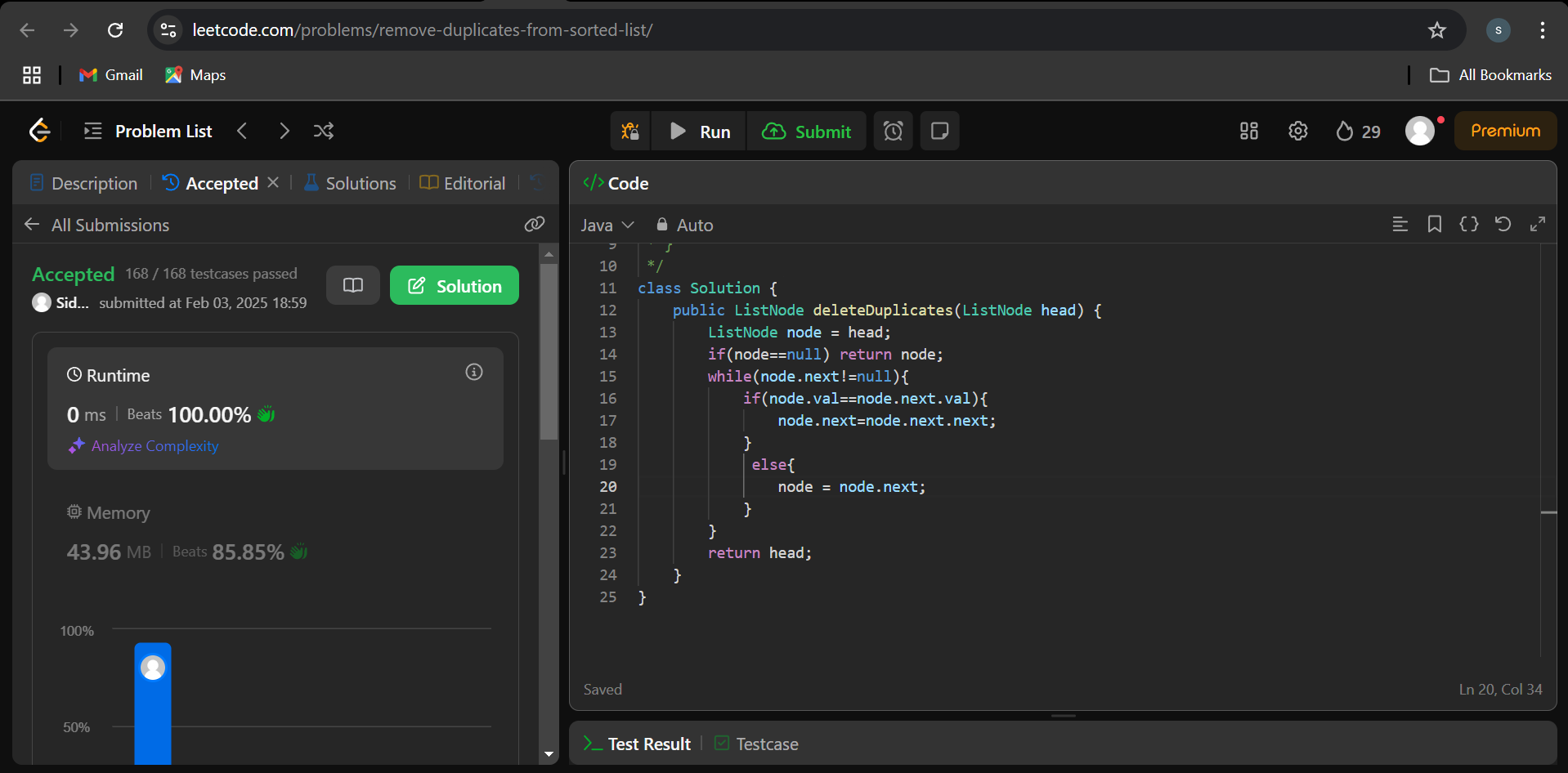
}

return res.next;

}

}

**Output:**



1. **Detect a cycle in a linked list:**

**Problem Link :** <https://leetcode.com/problems/linked-list-cycle/description/>

**Solution:**

public class Solution {

public boolean hasCycle(ListNode head) {

ListNode slow = head;

ListNode fast = head;

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

slow = slow.next;

fast = fast.next.next;

if(slow == fast) {

return true;

}

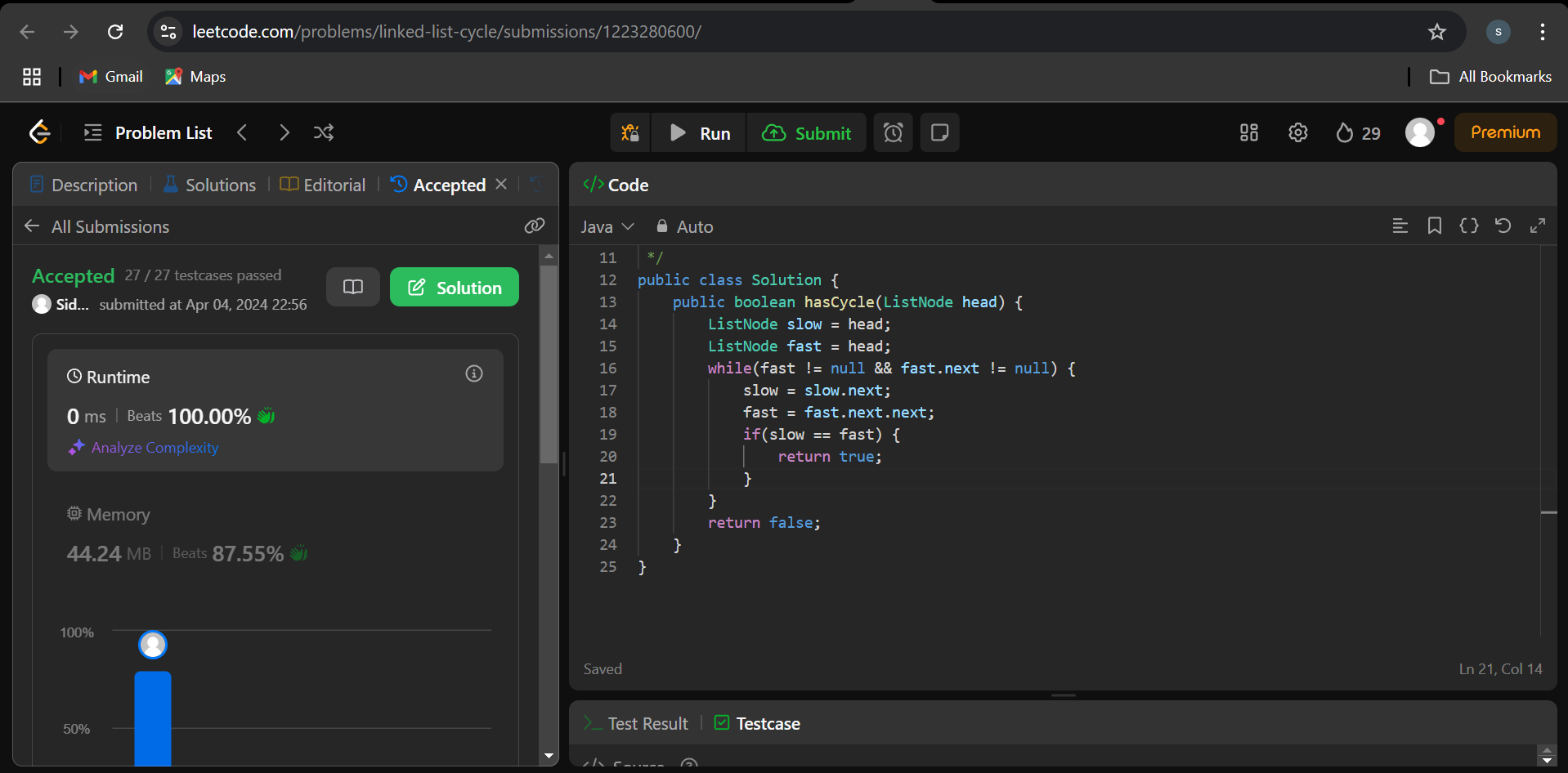
}

return false;

}

}

**Output:**

****

1. **Reverse linked list 2:**

**Problem Link:** <https://leetcode.com/problems/reverse-linked-list-ii/>

**Solution:**

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 curr = prev.next;

ListNode next = curr.next;

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

curr.next = next.next;

next.next = prev.next;

prev.next = next;

next = curr.next;

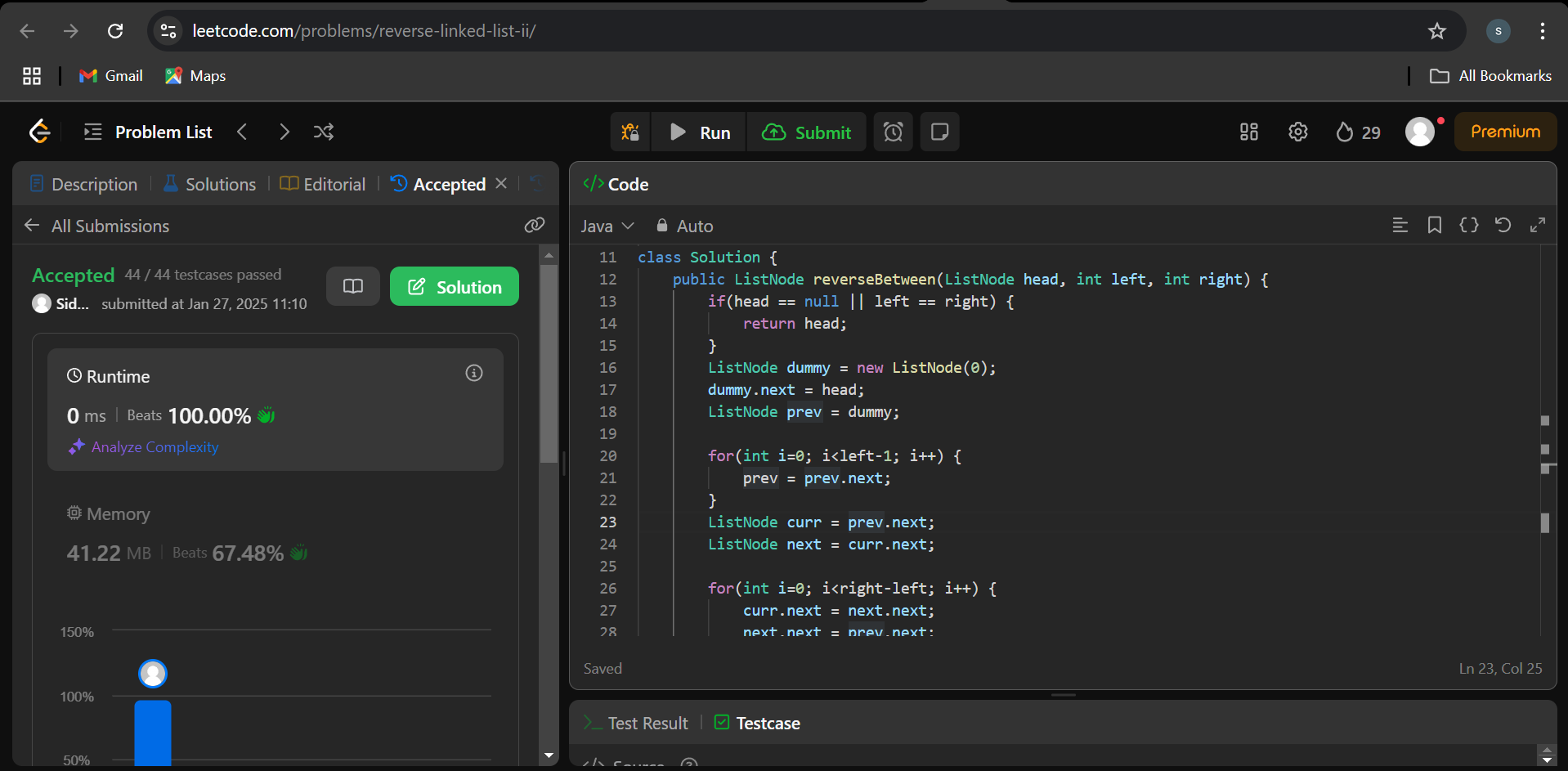
}

return dummy.next;

}

}

**Output:**

****

1. **Rotate a list:**

**Problem Link:** https://leetcode.com/problems/rotate-list/description/

**Solution:**

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;

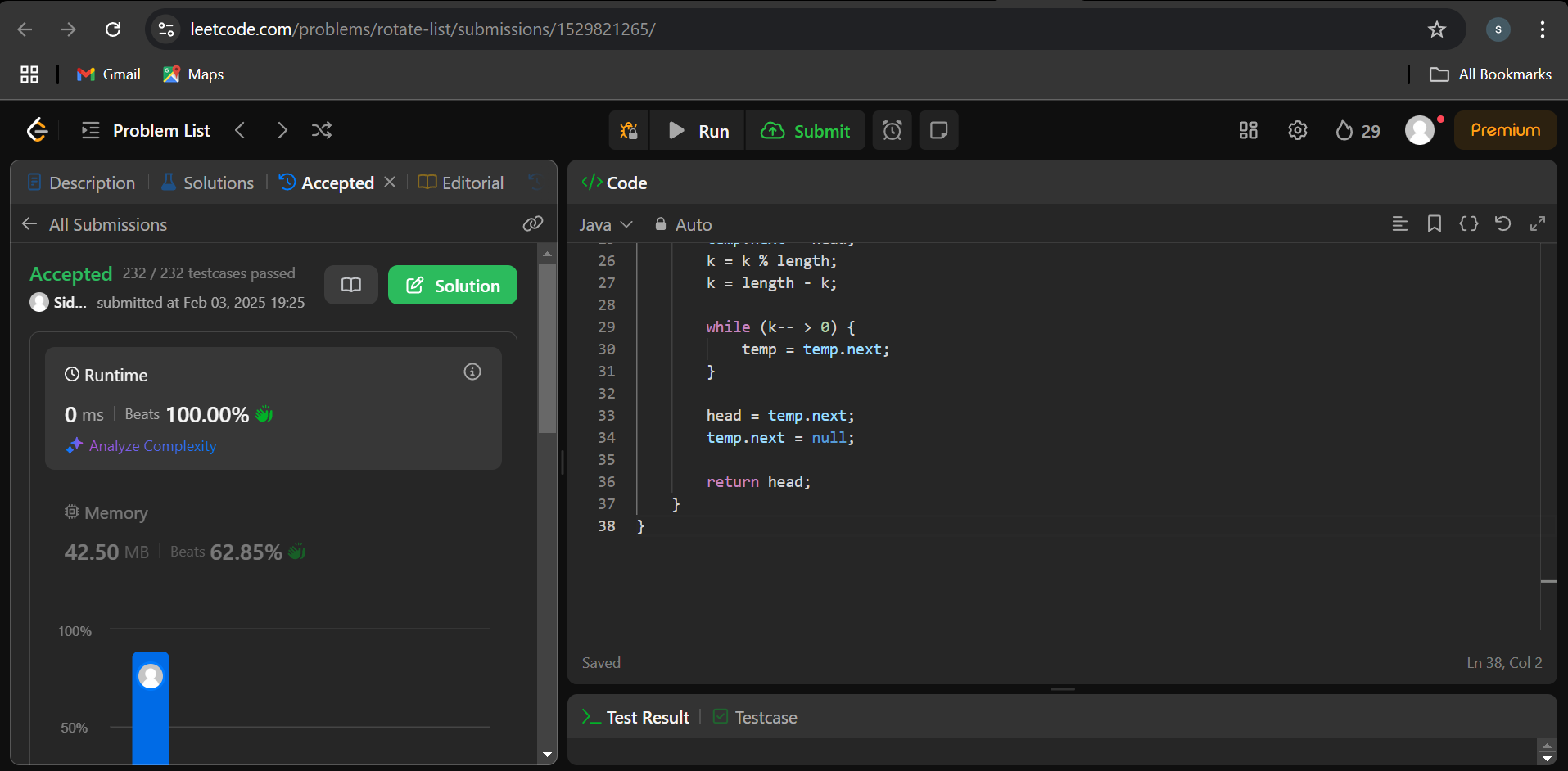
temp.next = null;

return head;

}

}

**Output:**

****

1. **Merge k sorted lists:**

**Problem Link:** <https://leetcode.com/problems/merge-k-sorted-lists/description/>

**Solution:**

class Solution {

private ListNode mergeTwoLists(ListNode l1, ListNode l2) {

ListNode dummy = new ListNode(0);

ListNode prev = dummy;

ListNode p1 = l1, p2 = l2;

while(p1 != null && p2 != null) {

if(p1.val < p2.val) {

prev.next = p1;

p1 = p1.next;

} else {

prev.next = p2;

p2 = p2.next;

}

prev = prev.next;

}

prev.next = (p1 != null) ? p1 : p2;

return dummy.next;

}

public ListNode mergeKLists(ListNode[] lists) {

if(lists.length == 0) return null;

while(lists.length > 1) {

int mergedSize = (lists.length + 1) / 2;

ListNode[] merged = new ListNode[mergedSize];

for(int i=0; i<mergedSize; i++) {

int idx1 = i \* 2;

int idx2 = i \* 2 + 1;

ListNode l1 = lists[idx1];

ListNode l2 = (idx2 < lists.length) ? lists[idx2] : null;

merged[i] = mergeTwoLists(l1, l2);

}

lists = merged;

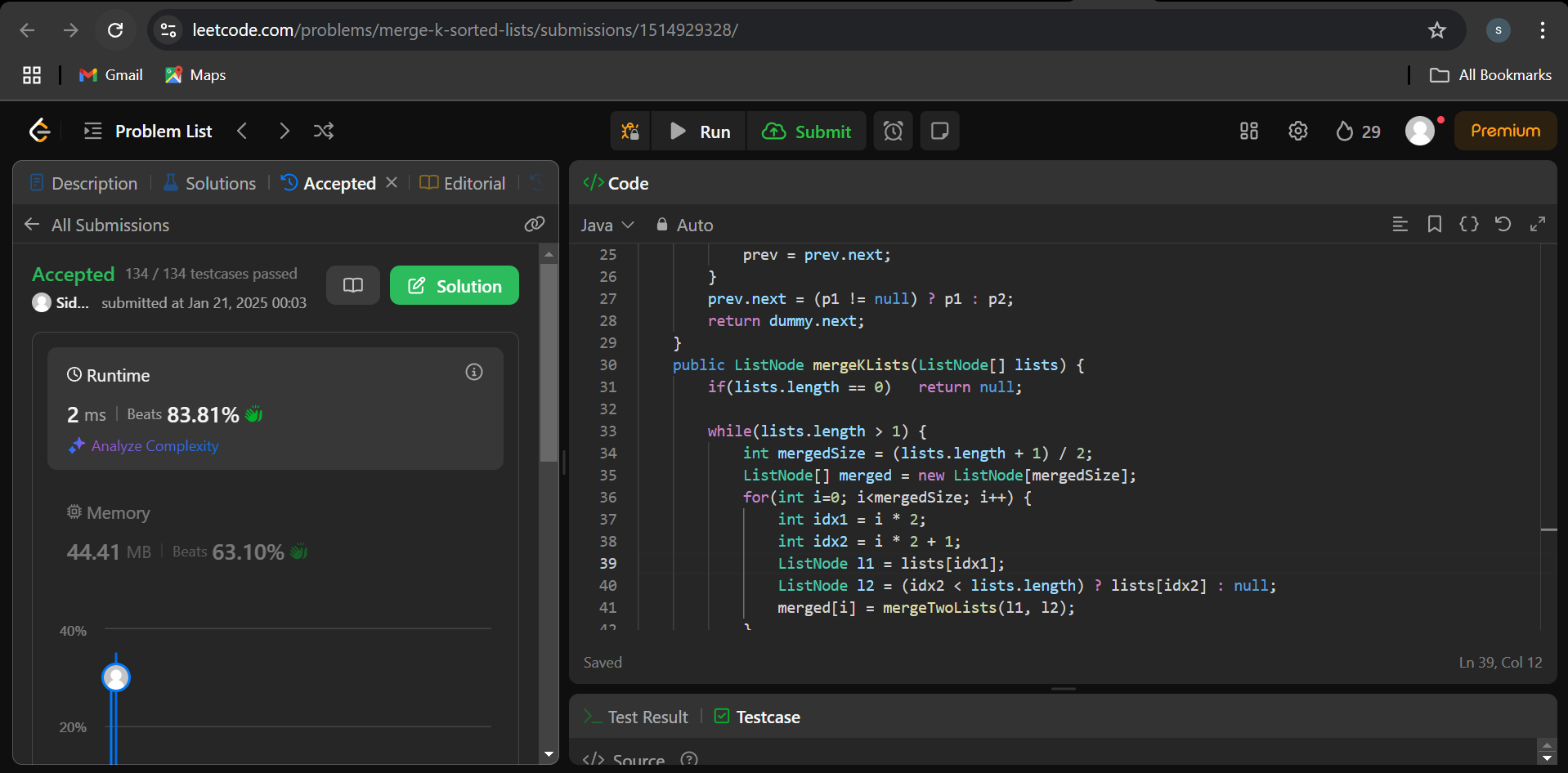
}

return lists[0];

}

}

**Output:**



1. **Sort List:**

**Problem Link:** <https://leetcode.com/problems/sort-list/description/>

**Solution:**

class Solution {

public 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;

}

public ListNode sortList(ListNode head) {

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

return head;

// Step 1: Divide the list into 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);

return merge(l1, l2);

}

}

**Output:**

