**Experiment 3**

**Student Name: Akshat Tiwari**  **UID: 22BCS10655**

**Branch: CSE**  **Section/Group: 608-B**

**Semester: 6th Date of Performance:7/3/25**

**Subject Name: Advanced Programming - 2 Subject Code: 22CSH-351**

**Ques 1:**

**Aim:** Print Linked List:

**Code:**

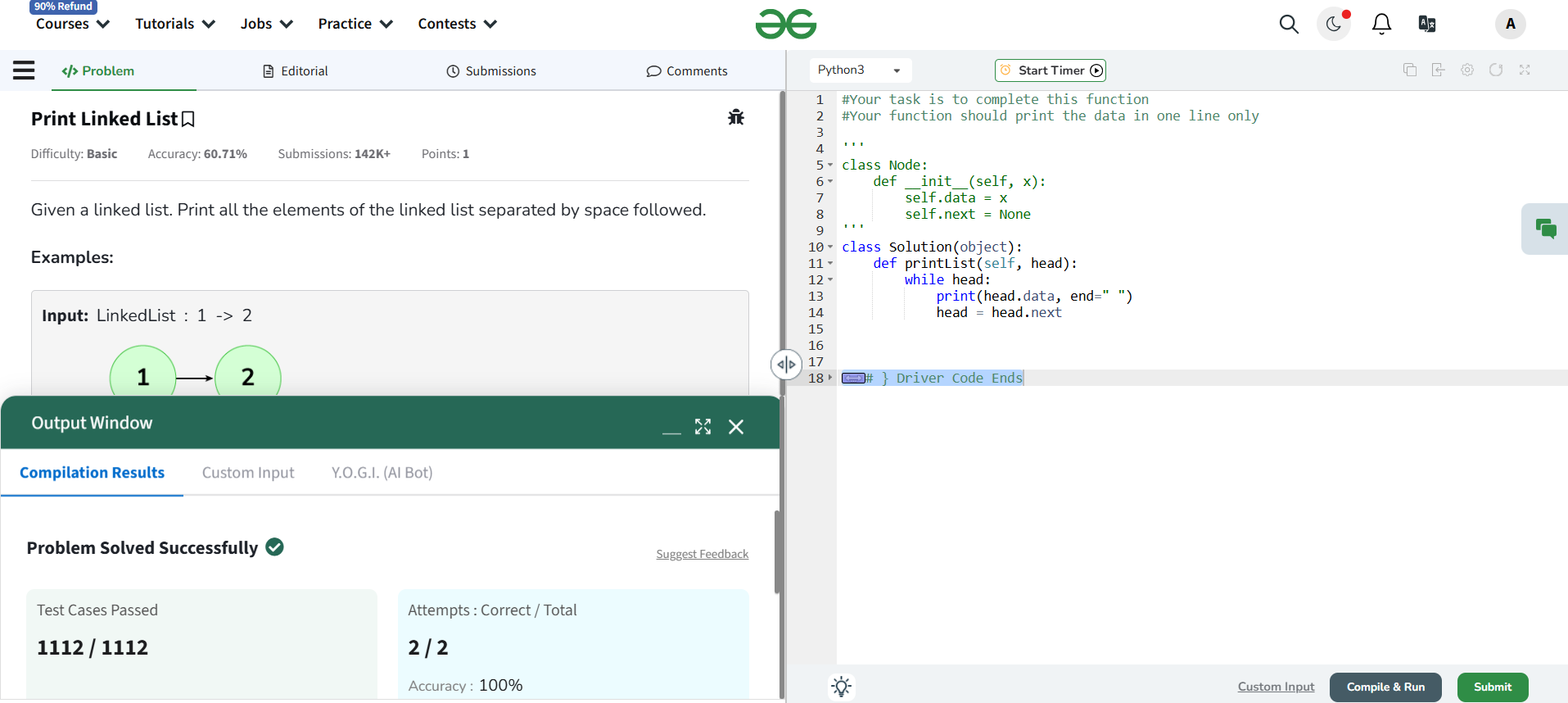
class Solution(object):

def printList(self, head):

while head:

print(head.data, end=" ")

head = head.next

**Submission Screenshot: **

**Ques 2:**

**Aim:** Remove duplicates from a sorted list:

**Code:**

class Solution {

public ListNode deleteDuplicates(ListNode head) {

if(head==null){

return null;

}

ListNode current = head;

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

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

current.next=current.next.next;

}else{

current = current.next;

}

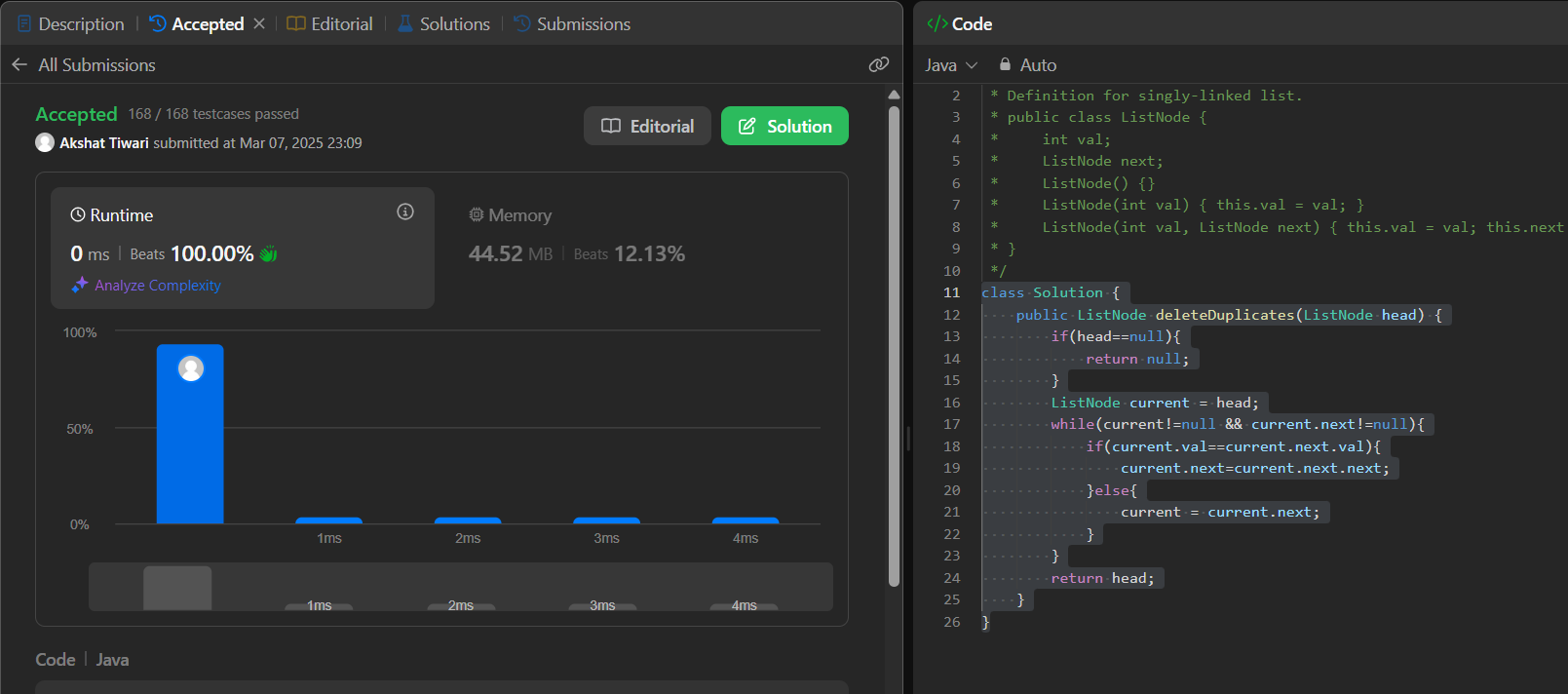
}

return head;

}

}

**Submission Screenshot:**



**Ques 3:**

**Aim:** Reverse a linked list:

**Code:**

class Solution {

public ListNode reverseList(ListNode head) {

ListNode prev = null;

ListNode current = head;

while(current!=null){

ListNode next = current.next;

current.next = prev;

prev = current;

current = next;

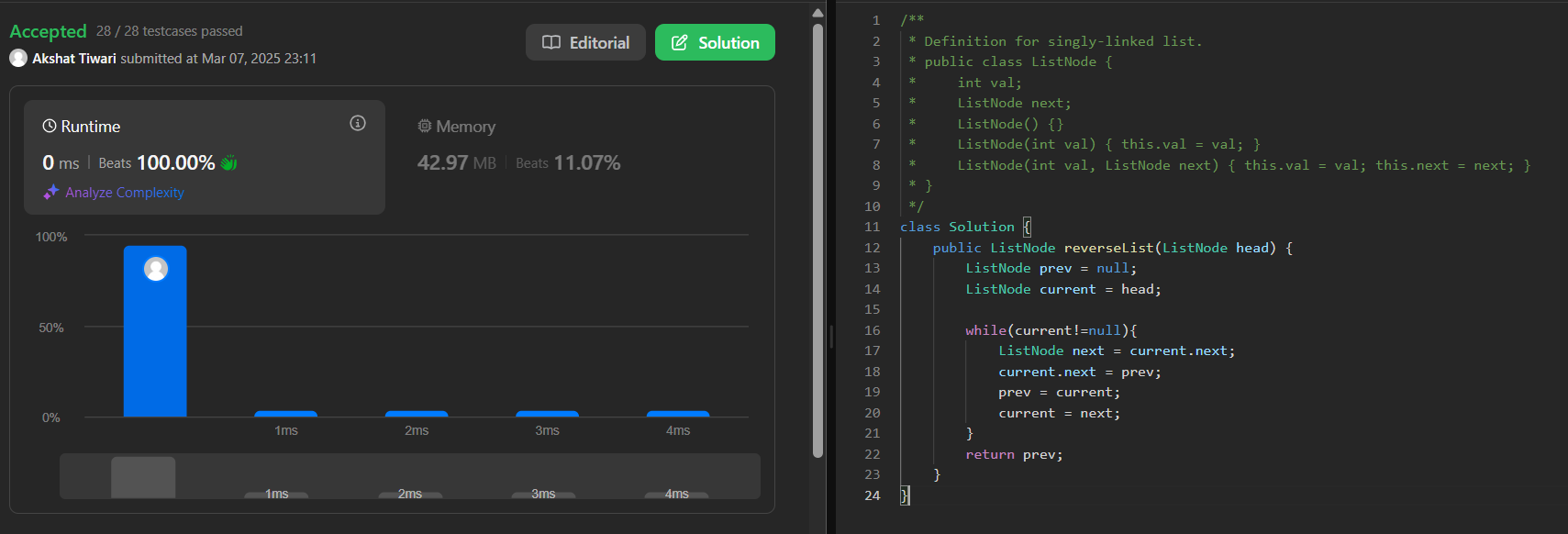
}

return prev;

}

}

**Submission Screenshot:**



**Ques 4:**

**Aim:** Delete middle node of a linked list:

**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;

}

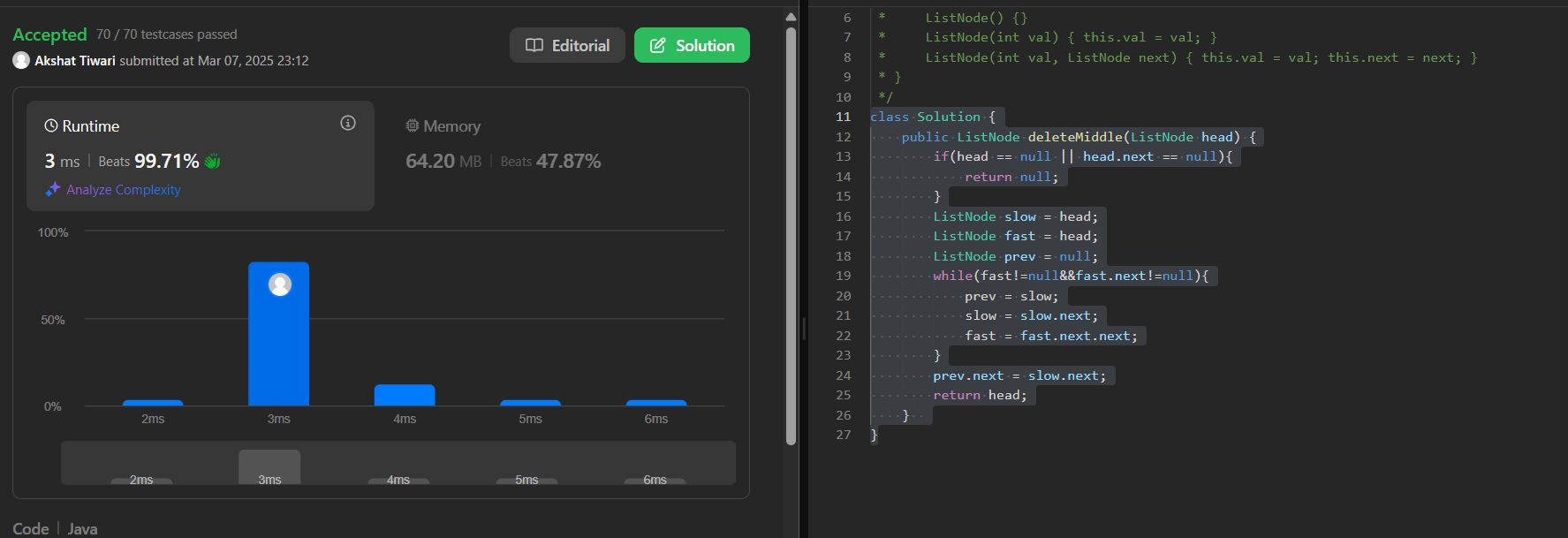
prev.next = slow.next;

return head;

}

}

**Submission Screenshot:**



**Ques 5:**

**Aim:** Merge two sorted linked lists:

**Code:**

class Solution {

public ListNode mergeTwoLists(ListNode list1, ListNode list2) {

ListNode dummy = new ListNode(-1);

ListNode tail = dummy;

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

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

tail.next = list1;

list1 = list1.next;

} else {

tail.next = list2;

list2 = list2.next;

}

tail = tail.next;

}

if (list1 != null) {

tail.next = list1;

} else {

tail.next = list2;

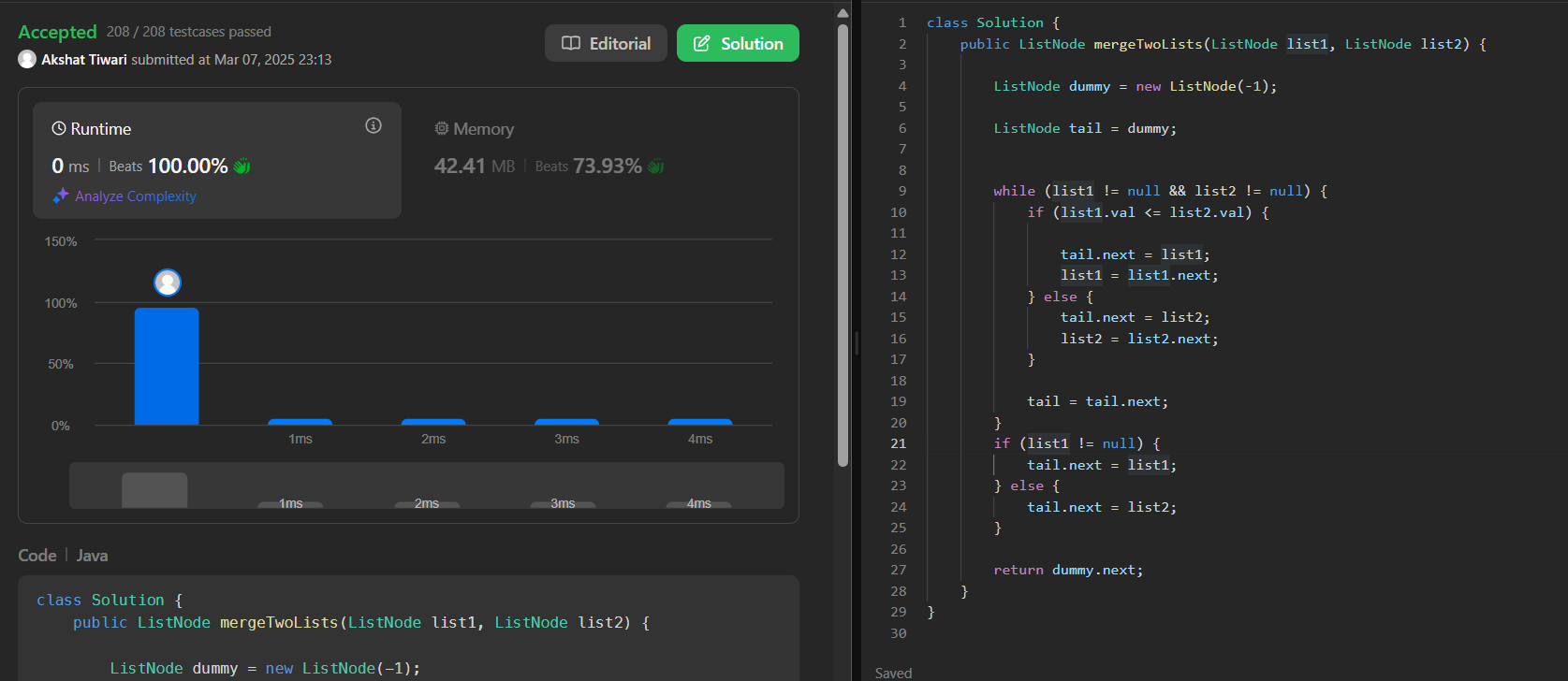
}

return dummy.next;

}

}

**Submission Screenshot:**



**Ques 6:**

**Aim:** Detect a cycle in a linked list:

**Code:**

public class Solution {

public boolean hasCycle(ListNode head) {

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

return false;

}

ListNode slow = head;

ListNode fast = head.next;

while(slow!=fast){

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

return false;

}

slow = slow.next;

fast = fast.next.next;

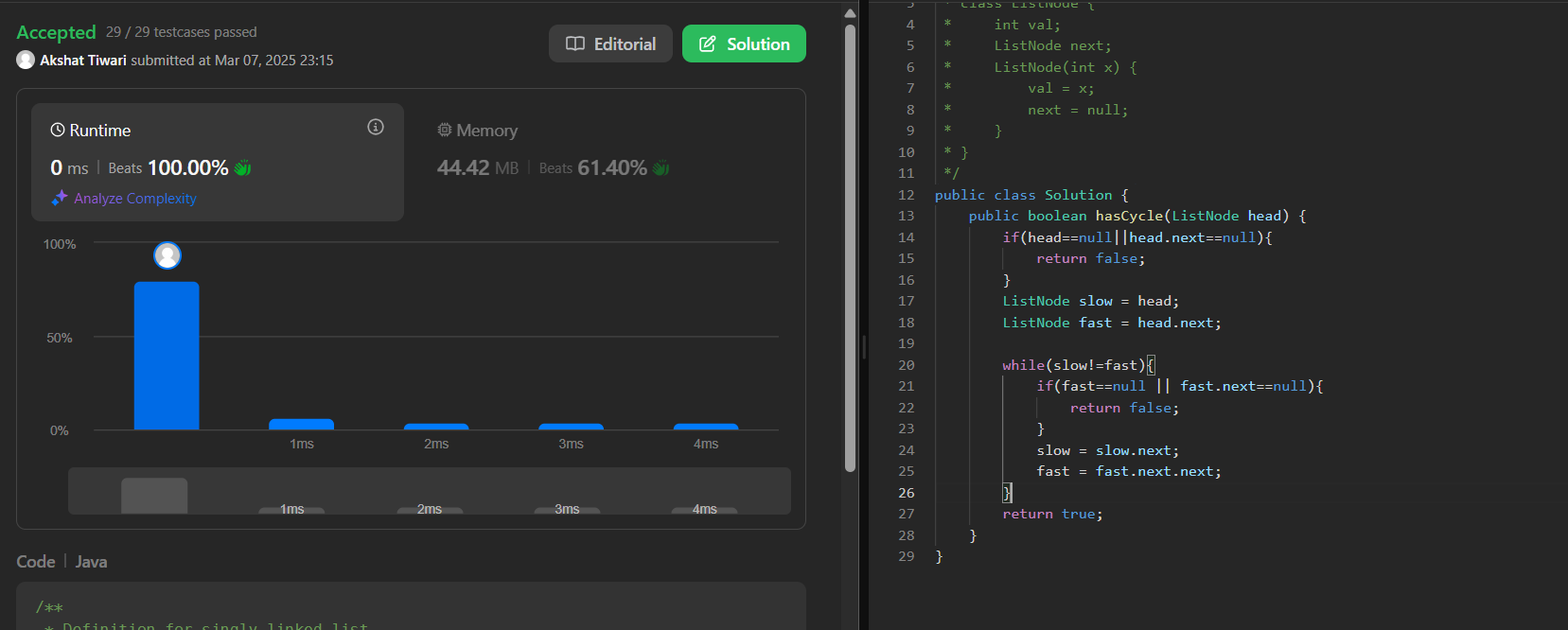
}

return true;

}

}

**Submission Screenshot:**



**Ques 7:**

**Aim:** Rotate a list:

**Code:**

class Solution {

public ListNode rotateRight(ListNode head, int k) {

if(head==null || head.next == null || k==0){

return head;

}

ListNode current = head;

int length = 1;

while(current.next!=null){

current = current.next;

length++;

}

current.next=head;

k = k%length;

int stepToNewHead = length-k;

ListNode newTail = head;

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

newTail = newTail.next;

}

ListNode newHead = newTail.next;

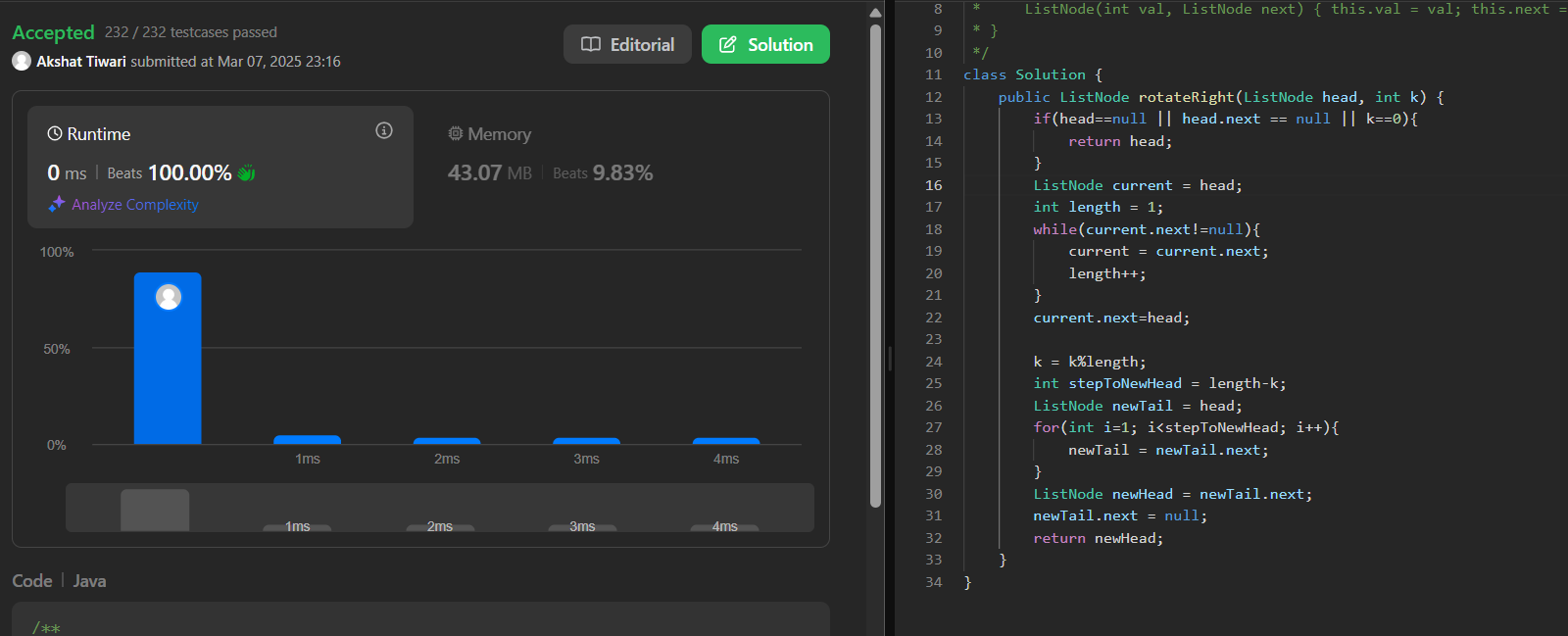
newTail.next = null;

return newHead;

}

}

**Submission Screenshot:**



**Ques 8:**

**Aim:** Sort List:

**Code:**

class Solution {

public ListNode sortList(ListNode head) {

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

return head;

}

ListNode mid = getMid(head);

ListNode right = mid.next;

mid.next = null;

ListNode leftSorted = sortList(head);

ListNode rightSorted = sortList(right);

return merge(leftSorted, rightSorted);

}

private ListNode merge(ListNode l1, ListNode l2) {

ListNode dummy = new ListNode(0);

ListNode current = dummy;

while (l1 != null && l2 != null) {

if (l1.val < l2.val) {

current.next = l1;

l1 = l1.next;

} else {

current.next = l2;

l2 = l2.next;

}

current = current.next;

}

if (l1 != null) {

current.next = l1;

} else if (l2 != null) {

current.next = l2;

}

return dummy.next;

}

private ListNode getMid(ListNode head) {

ListNode slow = head;

ListNode fast = head;

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

slow = slow.next;

fast = fast.next.next;

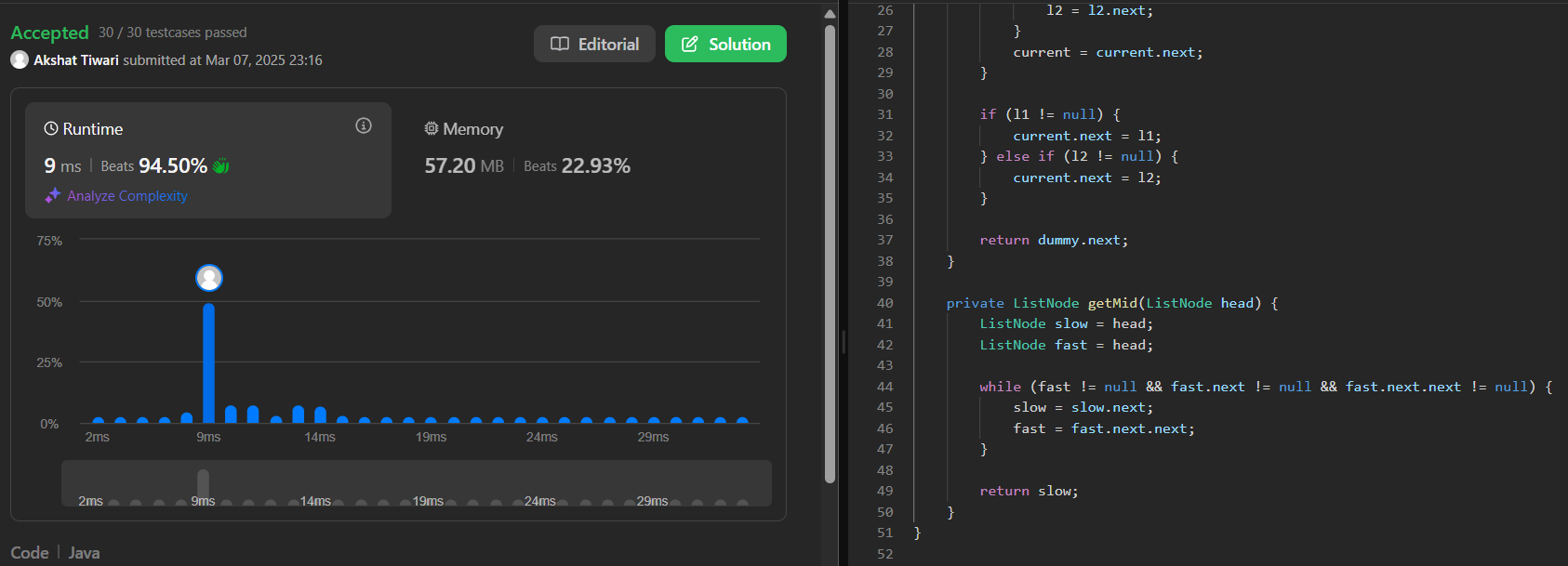
}

return slow;

}

}

**Submission Screenshot:**



**Ques 9:**

**Aim:** Merge k sorted lists:

**Code:**

import java.util.PriorityQueue;

class Solution {

public ListNode mergeKLists(ListNode[] lists) {

PriorityQueue<ListNode> minHeap = new PriorityQueue<>((a, b) -> a.val - b.val);

for (ListNode node : lists) {

if (node != null) {

minHeap.add(node);

}

}

ListNode dummy = new ListNode(0);

ListNode current = dummy;

while (!minHeap.isEmpty()) {

ListNode minNode = minHeap.poll();

current.next = minNode;

current = current.next;

if (minNode.next != null) {

minHeap.add(minNode.next);

}

}

return dummy.next;

}

}

**Submission Screenshot:**

