Assignment 3

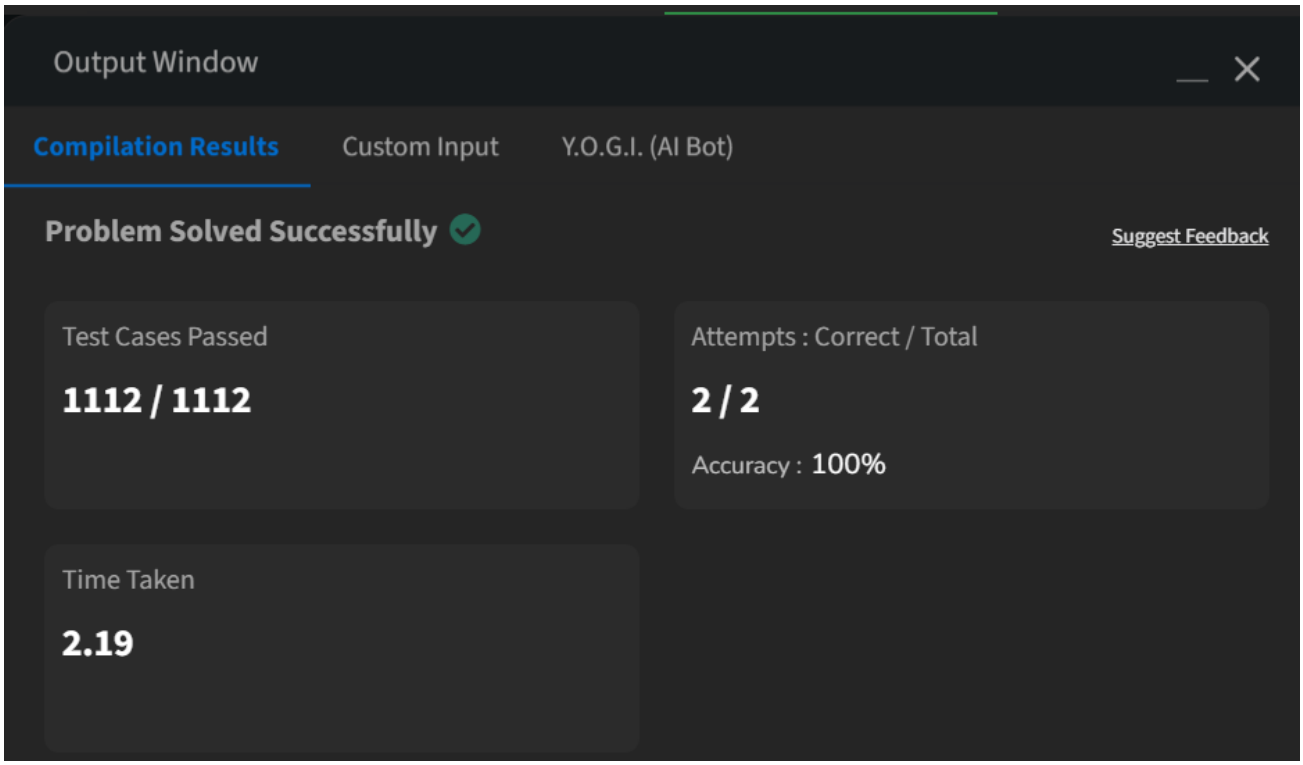
Student Name: Surbhi Priya **UID: 22BCS10268**
Branch: CSE **Section/Group: 22BCS_IOT_612**
Semester: 6th **Date of Performance:14/02/25**
Subject Name: Advance prog. Lab **Subject Code: 22CSP-351**

Q1) Print linked list

- **Code:**

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

- **Screenshot:**

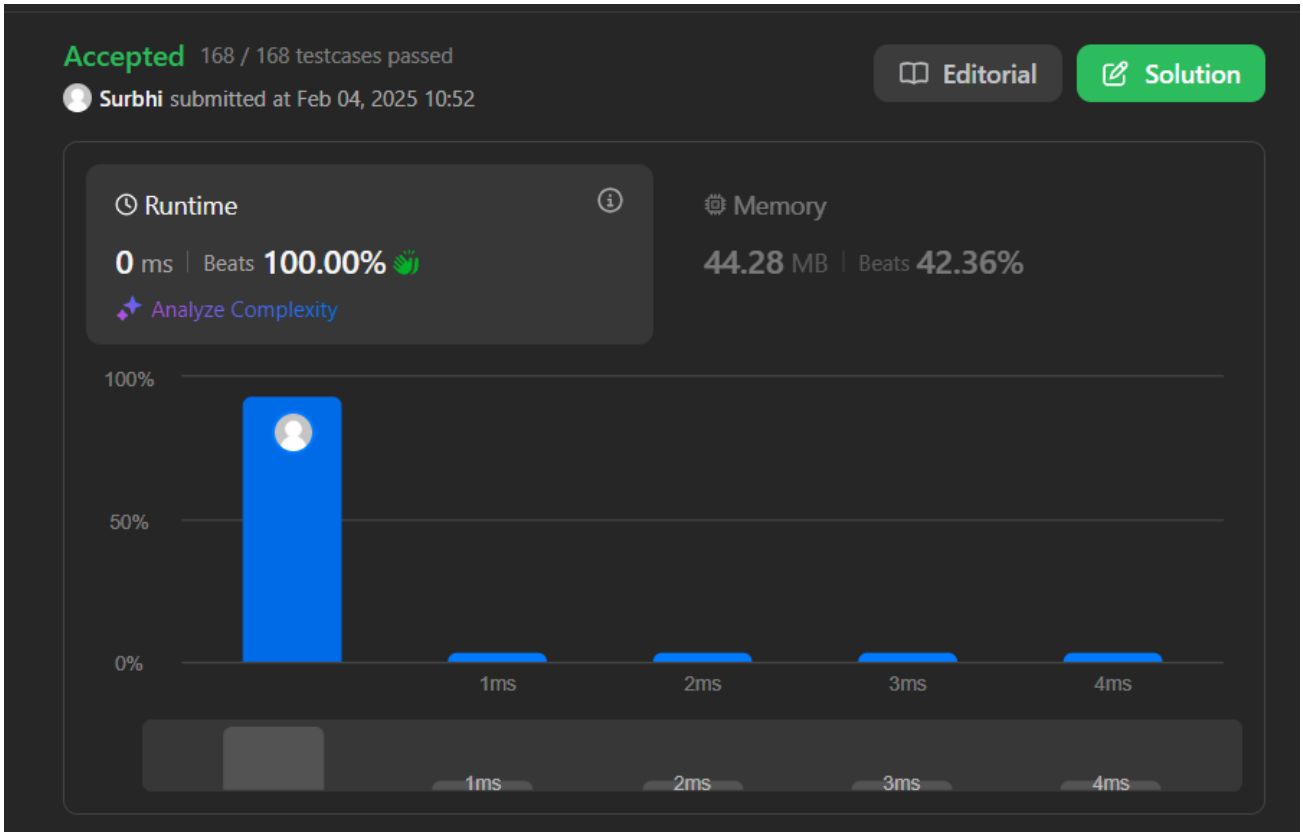


Q2) Remove duplicates from a sorted list

- **Code:**

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if (head == null) return null;

        ListNode hare=head;
        ListNode turtle=head.next;
        while(turtle!=null){
            if(hare.val==turtle.val){
                hare.next=turtle.next;
            }else{
                hare=hare.next;
            }
            turtle=turtle.next;
        }
        return head;
    }
}
```

- **Screenshot:**

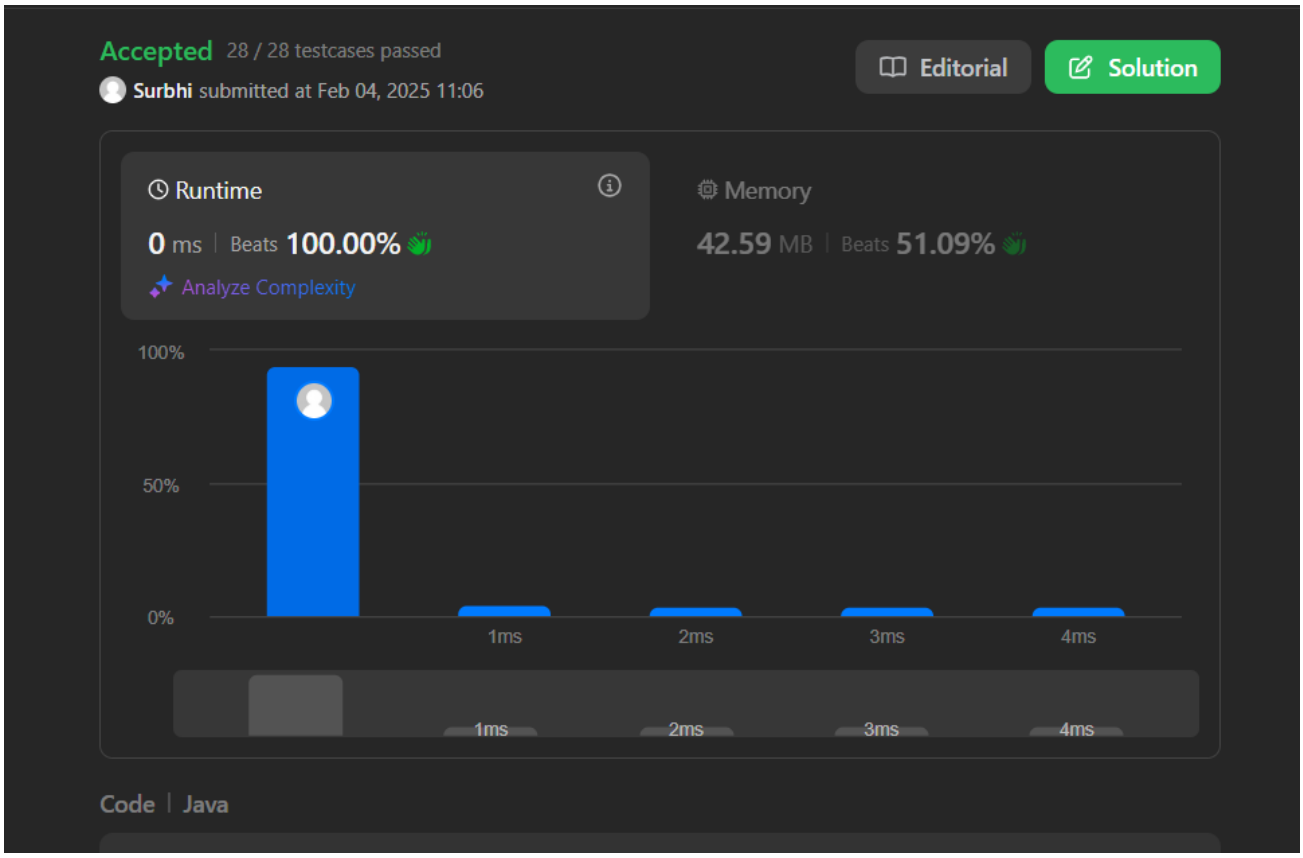


Q3) [Reverse a linked list]

- **Code:**

```
 class Solution{
public ListNode reverseList(ListNode head){
    if(head==null || head.next==null){
        return head;
    }
    ListNode prev=null;
    ListNode curr=head;
    while(curr!=null){
        ListNode nextNode=curr.next;
        curr.next=prev;
        prev=curr;
        curr=nextNode;
    }
    return prev;
}
}
```

- **Screenshot:**

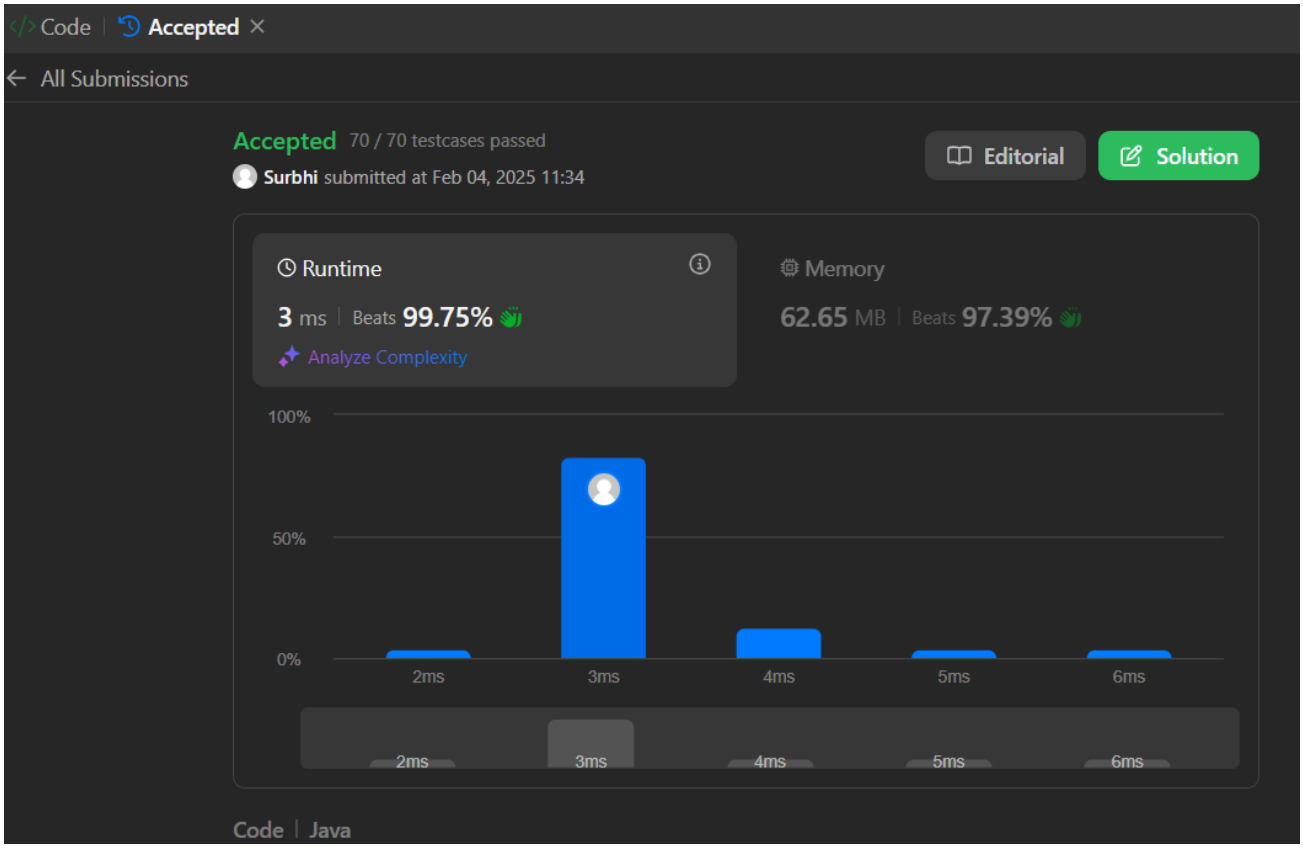


Q4) [Delete middle node of a list]

- **Code:**

```
class Solution {
 public ListNode deleteMiddle(ListNode head) {
    if(head==null || head.next==null){
        return null;
    }
    ListNode prev=null;
    ListNode hare=head,turtle=head;
    while(turtle!=null && turtle.next!=null){
        prev=hare;
        hare=hare.next;
        turtle=turtle.next.next;
    }
    prev.next=hare.next;
    return head;
  }
}
```

- **Screenshot:**



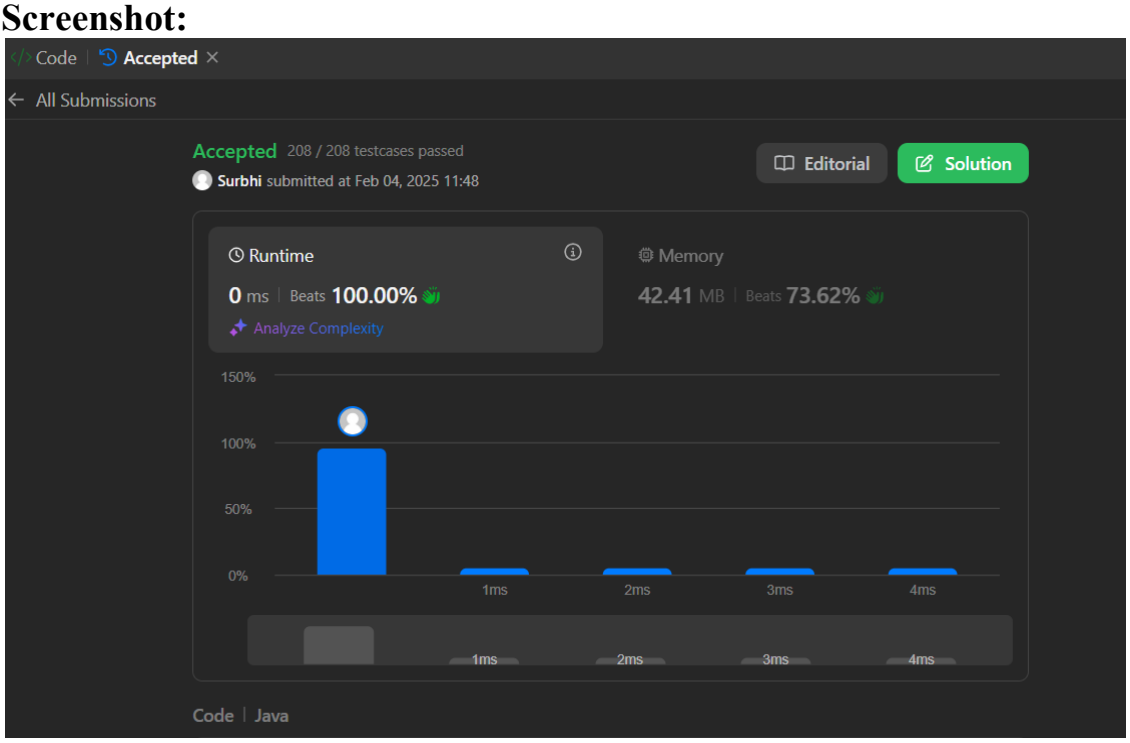
Q5) [Merge two sorted linked lists]

- **Code:**

```
class Solution {
 public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
```

```
        if(list1==null){
            return list2;
        }
        if(list2==null){
            return list1;
        }
        ListNode resNode=new ListNode(Integer.MIN_VALUE);
        ListNode Head=resNode;
        while(list1 !=null && list2!=null){
            if(list1.val <= list2.val){
                resNode.next=list1;
                list1=list1.next;
            }else{
                resNode.next=list2;
                list2=list2.next;
            }
            resNode=resNode.next;
        }
        if(list1==null){
            resNode.next=list2;
        }else if(list2==null){
            resNode.next=list1;
        }
        return Head.next;
    }
}
```

- **Screenshot:**

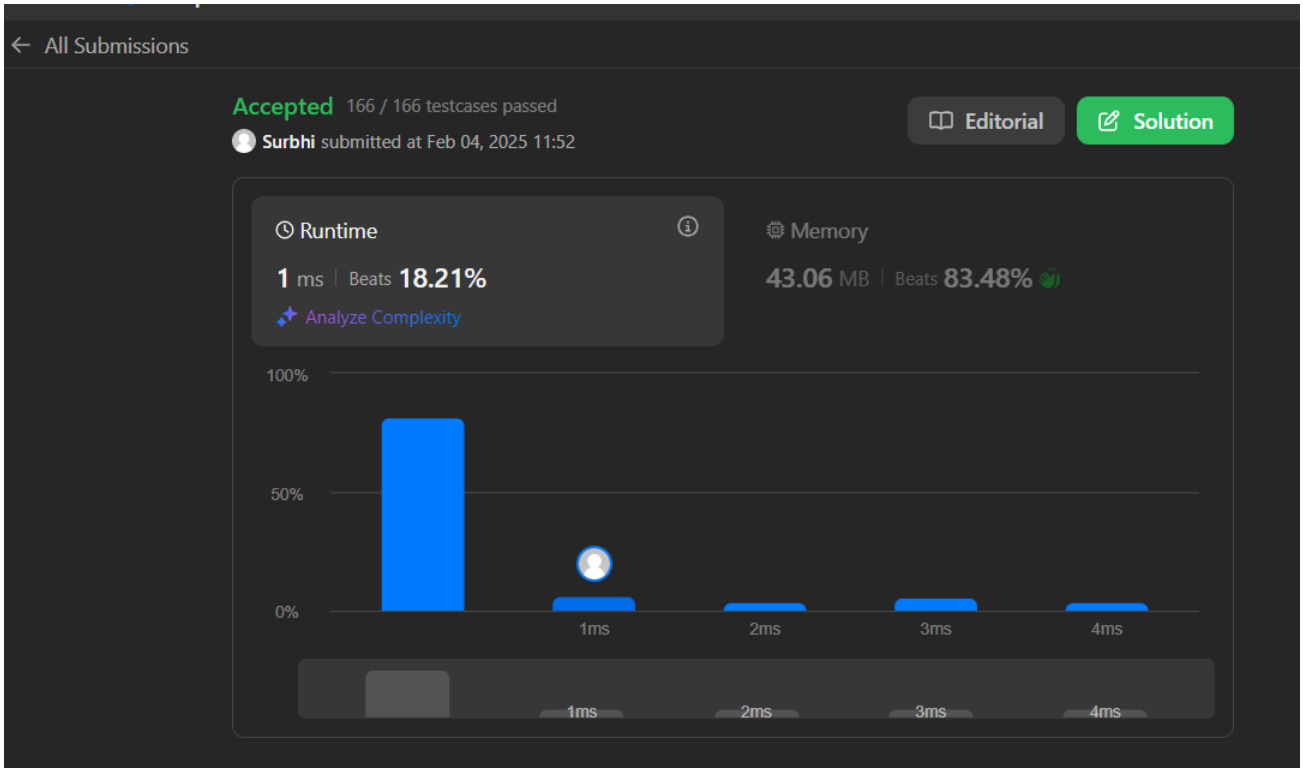


Q6) [Remove duplicates from sorted lists 2]

- **Code:**

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        if(head==null || head.next==null){
            return head;
        }
        ListNode dummy=new ListNode(0,head);
        ListNode prev=dummy;
        while(head!=null){
        if(head.next!=null && head.val==head.next.val){
            while(head.next!=null && head.val==head.next.val){
            head=head.next;
        }
        prev.next=head.next;
        }else{
            prev=prev.next;
        }
        head=head.next;
        }
        return dummy.next;
    }
}
```

- **Screenshot:**



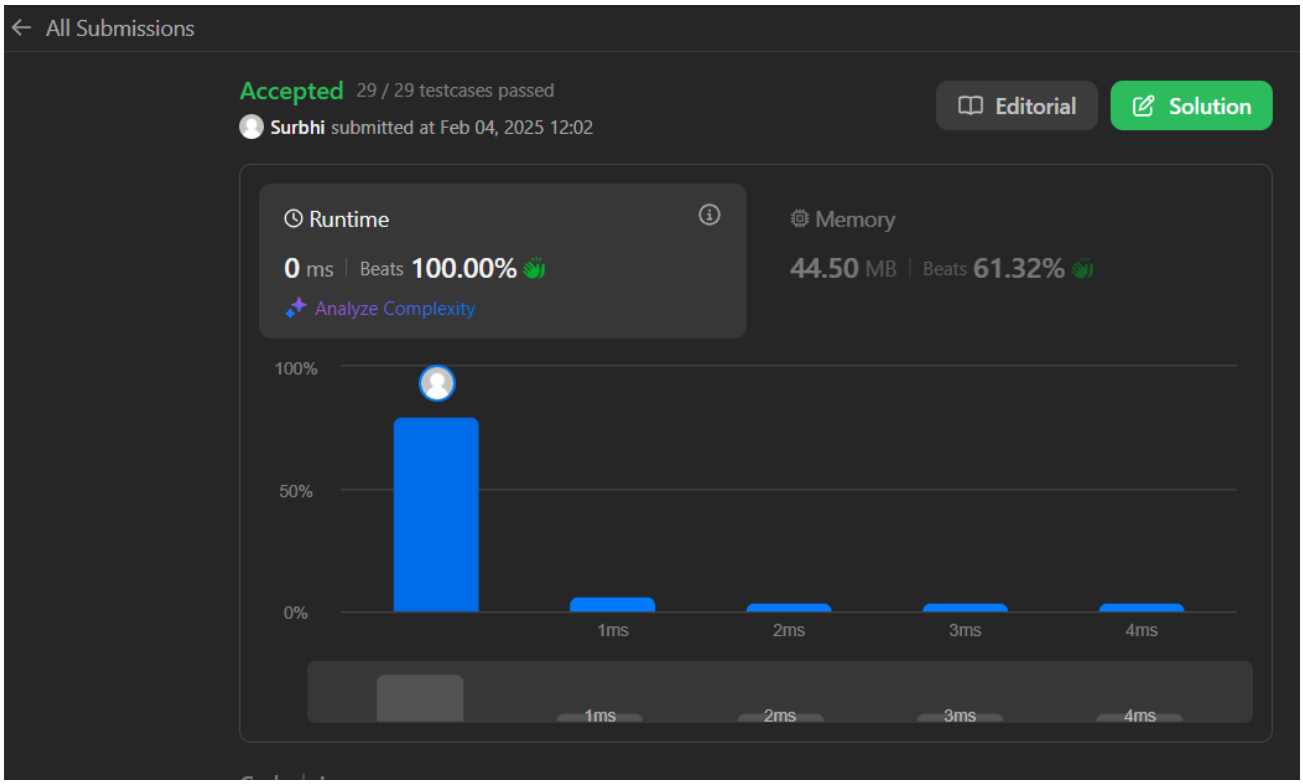
Q7) [Detect a cycle in a linked list]

- **Code:**

```
public class Solution {
    public boolean hasCycle(ListNode head) {
```

```
                ListNode hare=head;
                ListNode turtle=head;
                while(hare!=null && hare.next!=null){
                        hare=hare.next.next;
                        turtle=turtle.next;
                       if(hare==turtle){
                                return true;
                       }
                }
                return false;
        }
    }
```

- **Screenshot:**



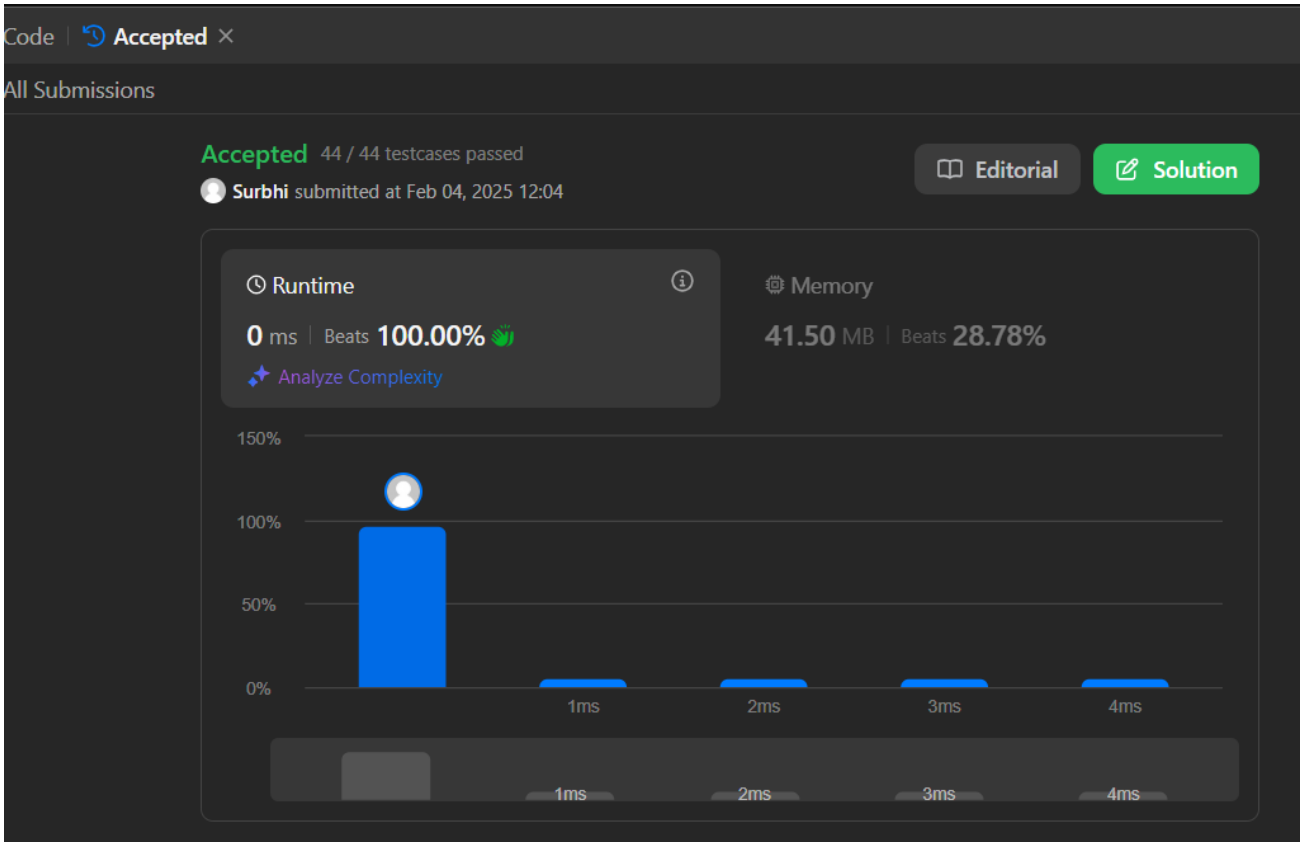
Q8) [Reverse linked list 2]

- **Code:**

```
class Solution {
public ListNode reverseBetween(ListNode head, int left, int right) {
    ListNode dummy=new ListNode(0);
    dummy.next=head;

    ListNode leftPre=dummy;
    ListNode currNode=head;
    for(int i=0;i<left-1;i++){
        leftPre=leftPre.next;
        currNode=currNode.next;
    }
    ListNode subListHead=currNode;
```

```
        ListNode preNode=null;

        for(int i=0;i<=right-left;i++){
            ListNode nextNode=currNode.next;
            currNode.next=preNode;

            preNode=currNode;
            currNode=nextNode;
        }
        leftPre.next=preNode;
        subListHead.next=currNode;
        return dummy.next;
    }
}
```

- Screensot:



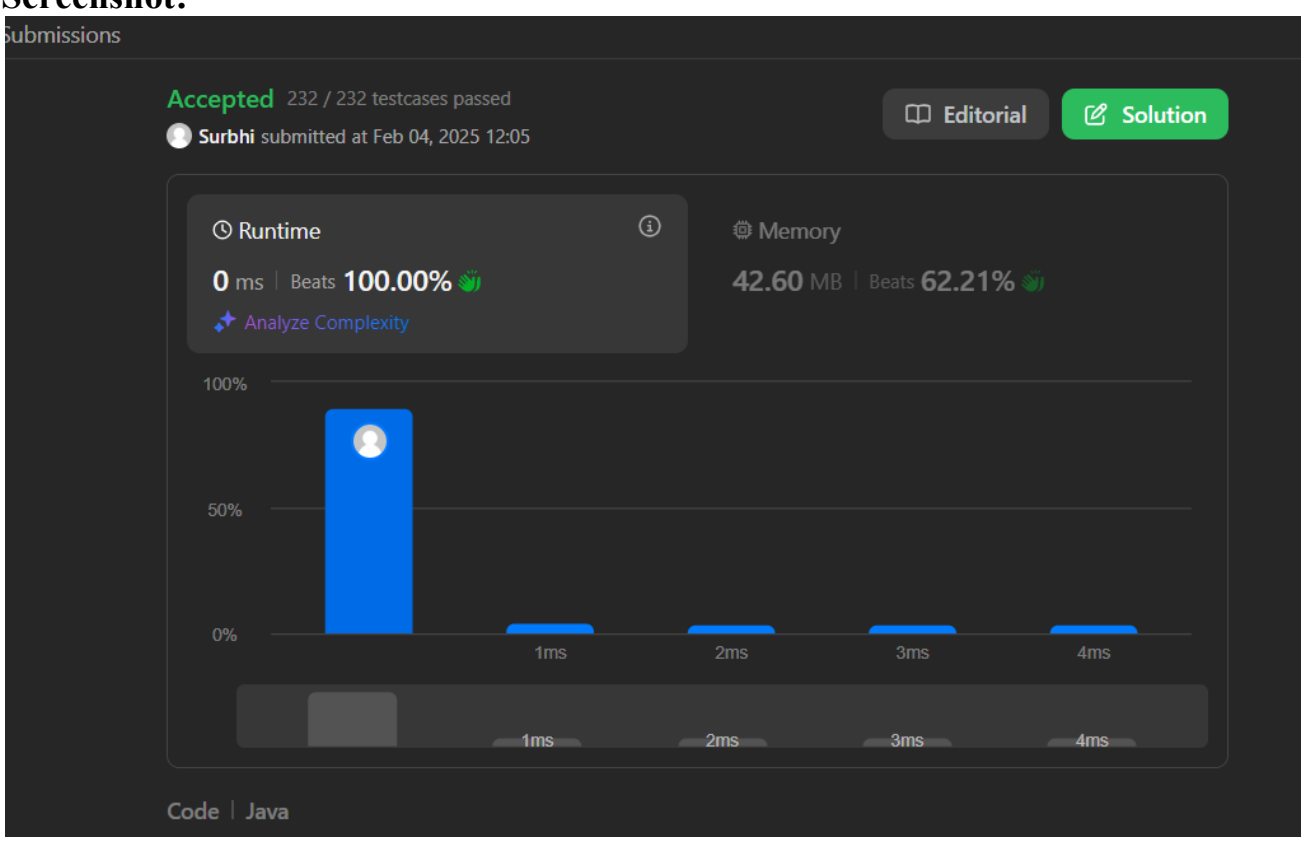
Q9) Rotate a List

- Code:

```
 class Solution {
public 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;
        k=k%length;
        int stepsToNewHead=length-k;

        ListNode newTail=head;
        for (int i = 1; i < stepsToNewHead; i++) {
            newTail = newTail.next;
        }

        // Break the circular list
        ListNode newHead = newTail.next;
        newTail.next = null;

        return newHead;

    }
}
```

- **Screenshot:**



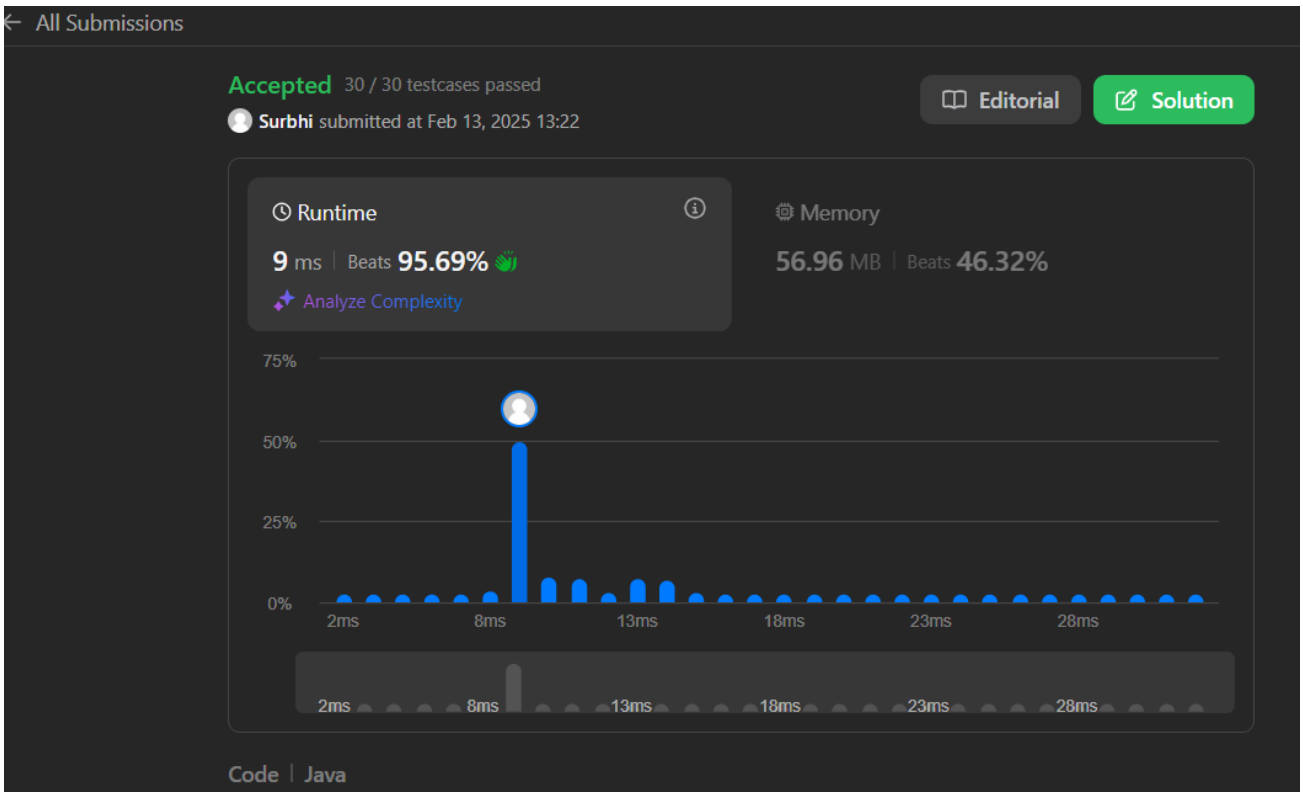
Q10) Sort List

- **Code:**

```
class Solution {
public ListNode sortList(ListNode head) {
    // Base case: If the list is empty or has one node, it's already sorted
```

```
        if (head == null || head.next == null) {
            return head;
        }
        // Split the list into two halves
        ListNode mid = getMiddle(head);
        ListNode rightHalf = mid.next;
        mid.next = null;
        // Recursively sort both halves
        ListNode leftSorted = sortList(head);
        ListNode rightSorted = sortList(rightHalf);

        // Merge the sorted halves
        return merge(leftSorted, rightSorted);
    }
    // Helper function to find the middle of the linked list
    private ListNode getMiddle(ListNode head) {
        ListNode slow = head, fast = head;
        while (fast.next != null && fast.next.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }
        return slow;
    }
    // Helper function to merge two sorted linked lists
    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;
        }
        // Append the remaining nodes from either list
        if (l1 != null) {
            current.next = l1;
        } else {
            current.next = l2;
        }
        return dummy.next;
    }
}
```

- **Screenshot:**

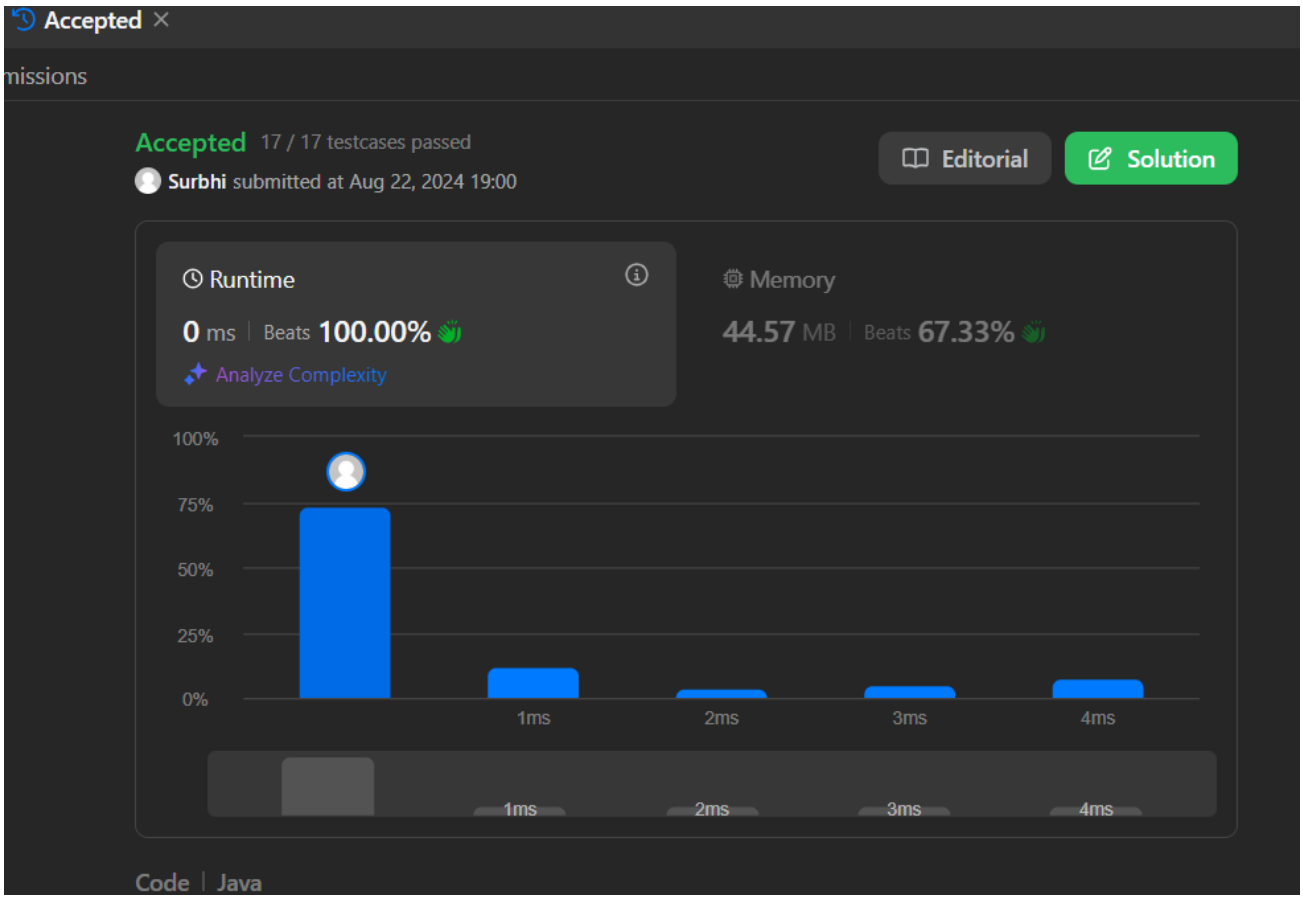


Q11) [Detect a cycle in a linked list 2]

- **Code:**

```
public class Solution {
  public ListNode detectCycle(ListNode head) {
    if (head == null) return null;
    ListNode hare=head;
    ListNode turtle=head;
    while(hare!=null && hare.next!=null){
      //find the entry point of cycle
      hare=hare.next.next;
      turtle=turtle.next;
      if(hare==turtle){
        turtle=head;
        while(turtle!=hare){
            turtle=turtle.next;
            hare=hare.next;
        }
        return hare;
      }
    }
    return null;
  }
}
```

- **Screenshot:**


Accepted
missions
Accepted 17 / 17 testcases passed
Surbhi submitted at Aug 22, 2024 19:00
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
44.57 MB | Beats 67.33%
100%
75%
50%
25%
0%
1ms
2ms
3ms
4ms
Code | Java