EXPERIMENT

Student Name: Dhruv **UID:**22BET10129

Branch: BE -IT **Section/Group:**22BET_IOT-703(A)

Semester: 6th **Subject Code:** 22ITP-351

PROBLEM-1

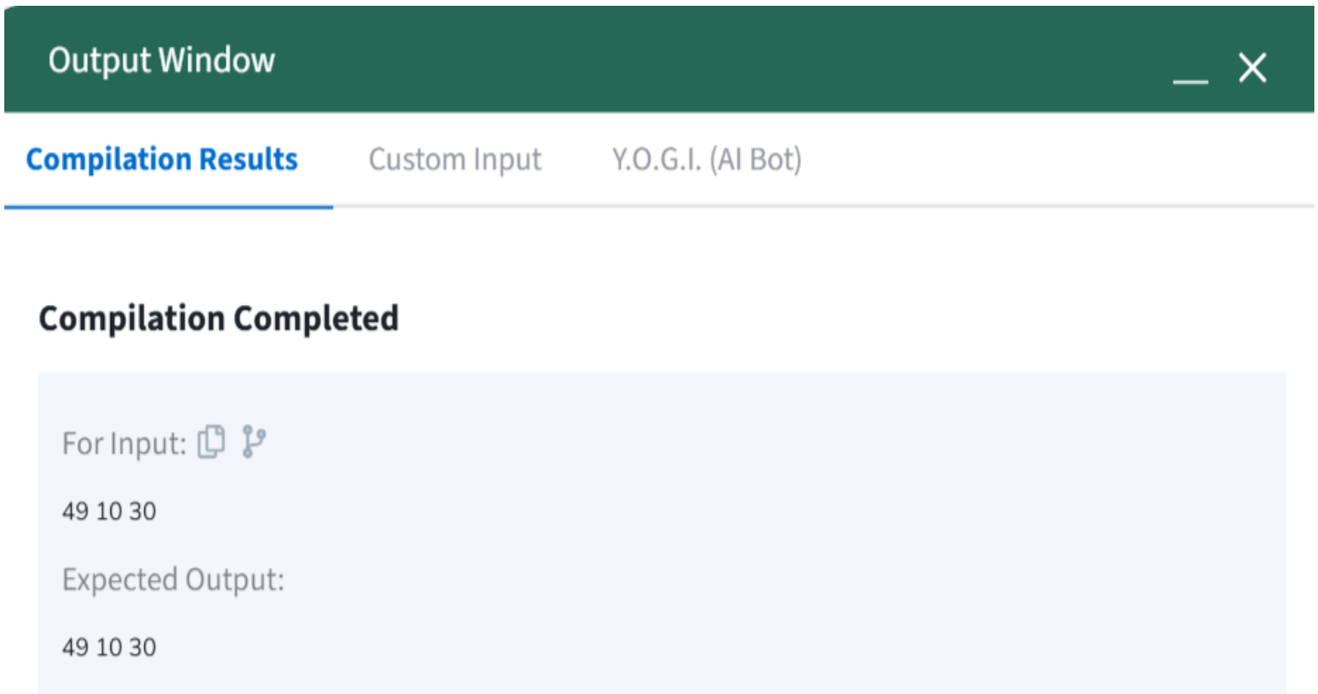
AIM:-

Print linked list

CODE:-

```
class Node {
    int data;
    Node next;
    Node(int data) {
        this.data = data;
        this.next = null;
    }
}

class Solution {
    void printList(Node head) {
        Node temp = head;
        while (temp != null) {
            System.out.print(temp.data + " ");
            temp = temp.next;
        }
        System.out.println();
    }
}

public class Main {
    public static void main(String[] args) {
        Node head = new Node(49);
        head.next = new Node(10);
        head.next.next = new Node(30);

        Solution sol = new Solution();
        sol.printList(head);
    }
}
```

OUTPUT:-



PROBLEM-2

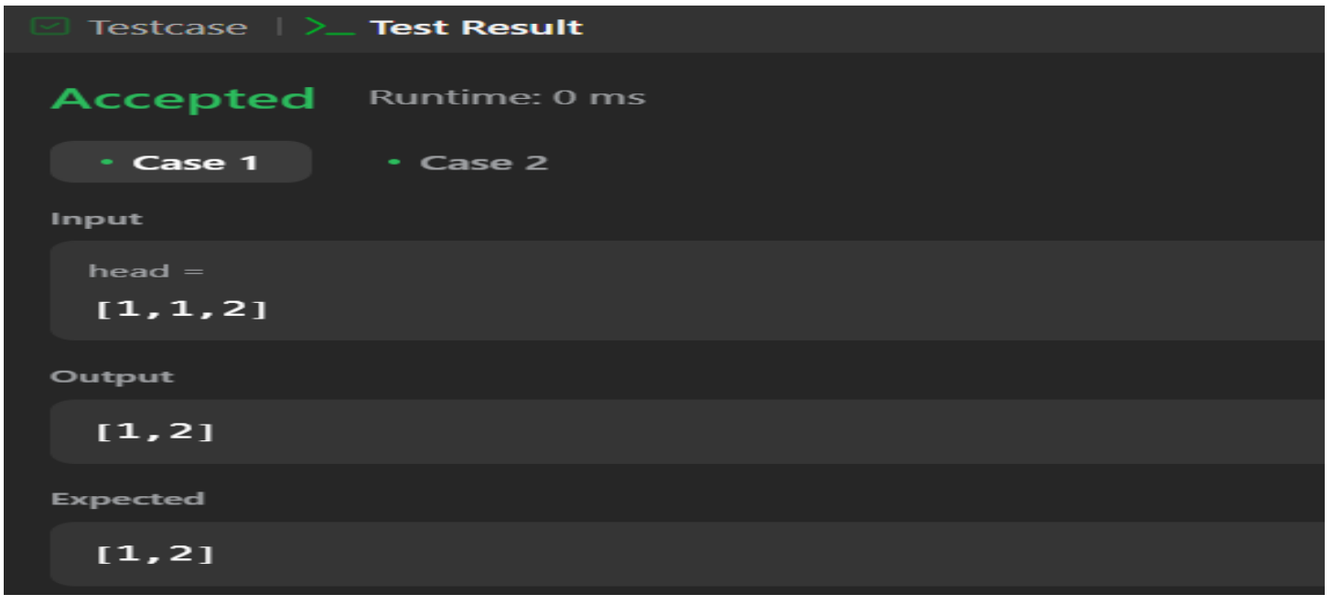
AIM:-

Remove duplicates from a sorted list

CODE:-

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode res = head;

        while (head != null && head.next != null) {
            if (head.val == head.next.val) {
                head.next = head.next.next;
            } else {
                head = head.next;
            }
        }

        return res;
    }
}
```

OUTPUT:-



PROBLEM-3

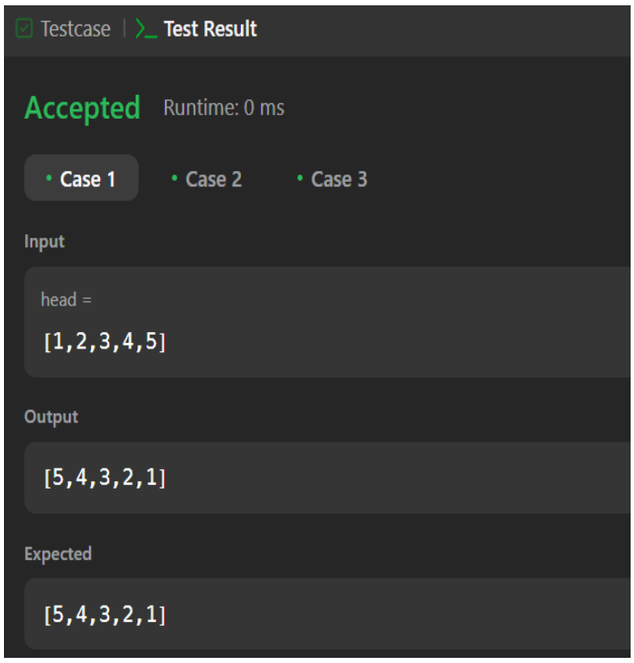
AIM:-

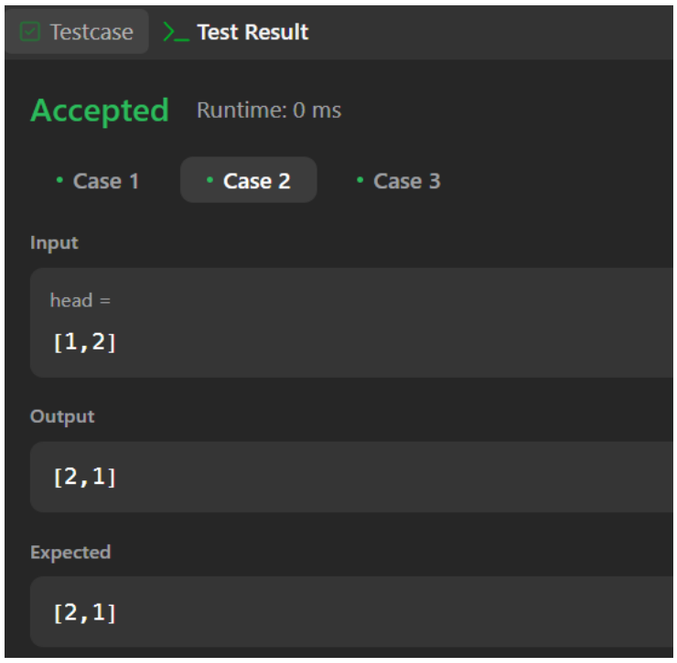
Reverse a linked list

CODE:-

```
class Solution {
    public ListNode reverseList(ListNode head) {
        ListNode node = null;

        while (head != null) {
            ListNode temp = head.next;
            head.next = node;
            node = head;
            head = temp;
        }

        return node;
    }
}
```

OUTPUT:-





PROBLEM-4

AIM:-

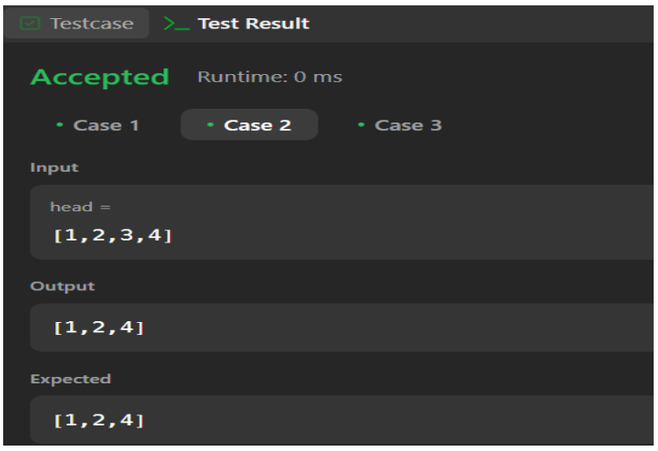
Delete middle node of a list

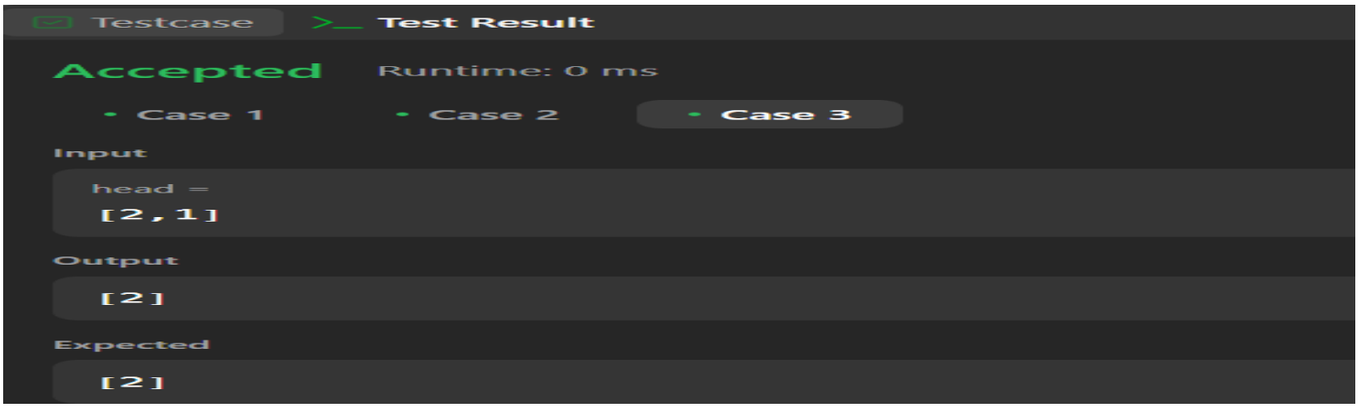
CODE:-

```
class Solution {
    public ListNode deleteMiddle(ListNode head) {
        ListNode counter = head;
        int count = 0;
        while(counter != null){
            counter = counter.next;
            count++;
        }
        ListNode curr = head;
        if(count == 1){
            return curr.next;
        }
        int middle = (count/2) - 1;
```

```
        count = 0;
        while(count != middle){
            curr = curr.next;
            count++;
        }
        curr.next = curr.next.next;
        return head;
    }
}
```

OUTPUT:-





PROBLEM-5

AIM:-

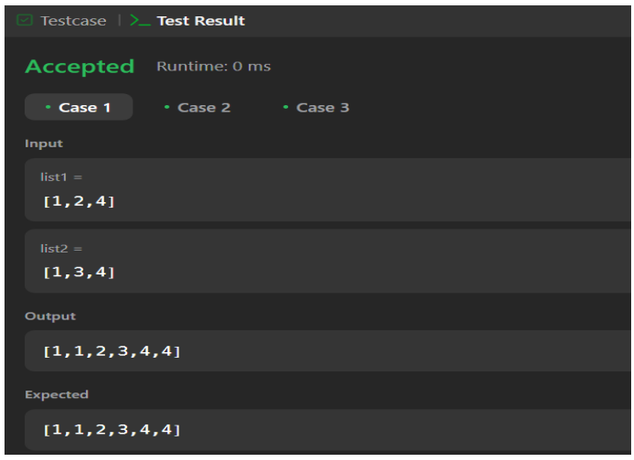
Merge two sorted linked lists

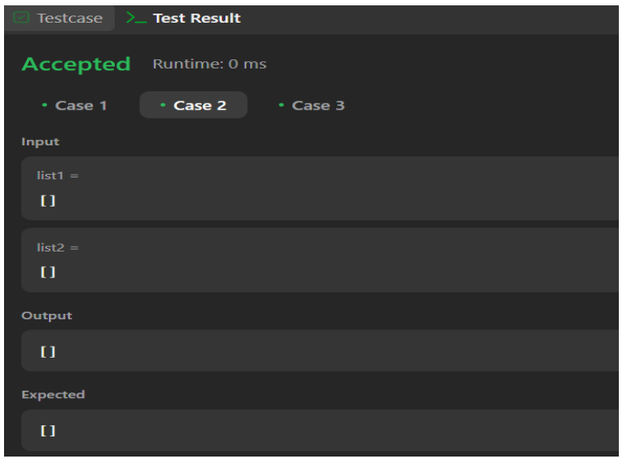
CODE:-

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2) {
        ListNode dummy = new ListNode();
        ListNode cur = dummy;

        while (list1 != null && list2 != null) {
            if (list1.val > list2.val) {
                cur.next = list2;
                list2 = list2.next;
            } else {
                cur.next = list1;
                list1 = list1.next;
            }
```

```
            cur = cur.next;
        }

        cur.next = (list1 != null) ? list1 : list2;

        return dummy.next;
    }
}
```

OUTPUT:-





PROBLEM-6

AIM:-

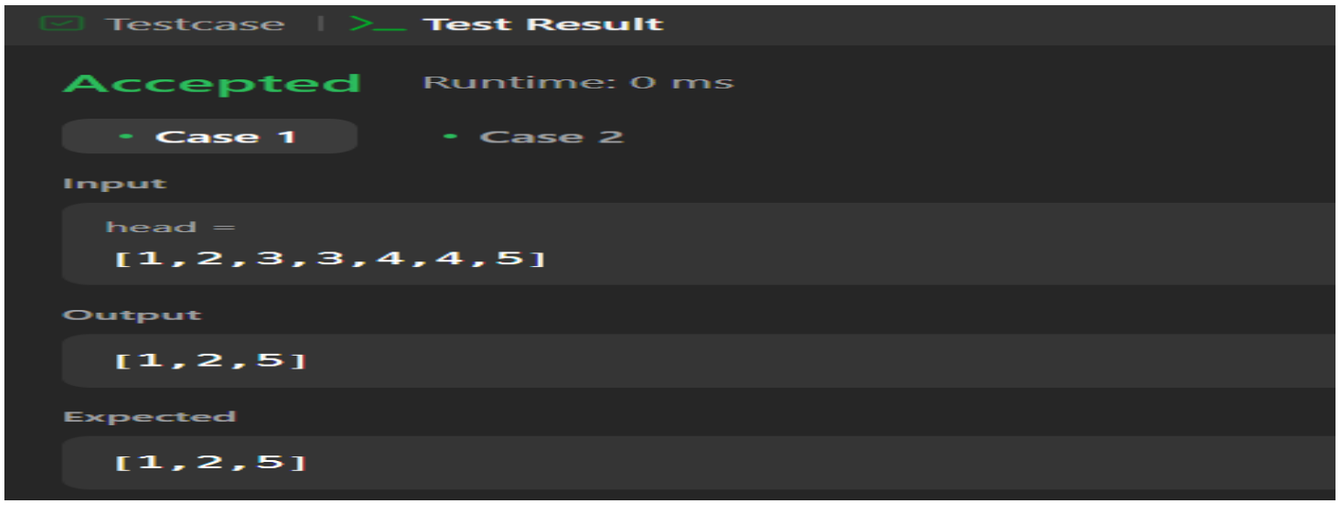
Remove duplicates from sorted lists 2

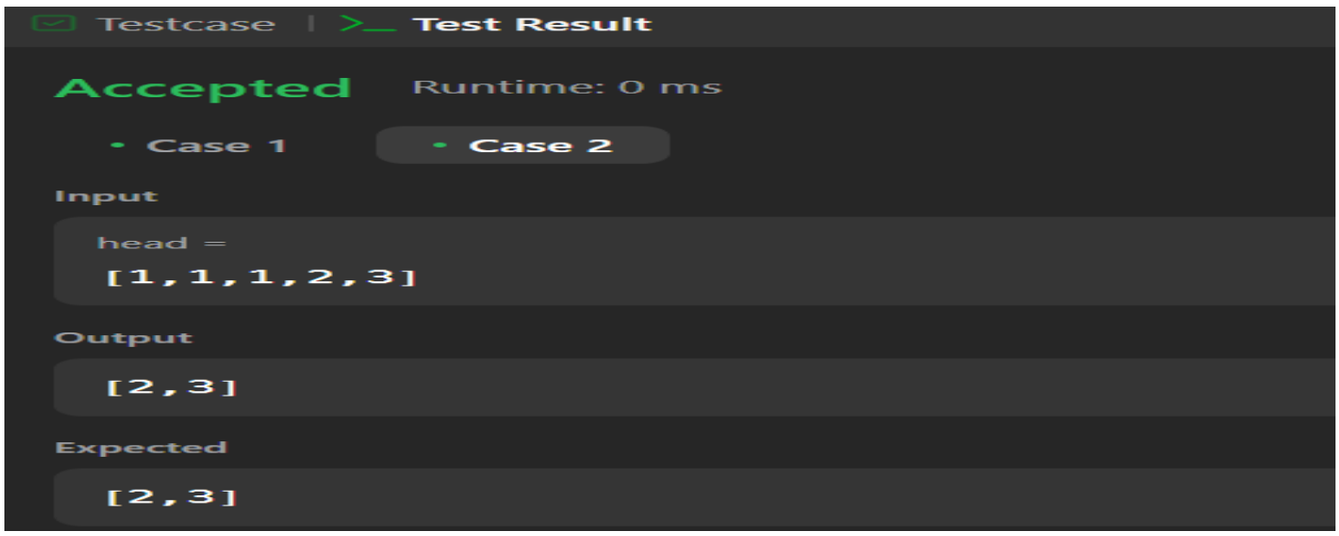
CODE:-

```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        ListNode ans = new ListNode(1000, head);
        ListNode cur = ans;

        while (cur.next != null && cur.next.next != null) {
            if (cur.next.val == cur.next.next.val) {
                int val = cur.next.val;
                while (cur.next != null && cur.next.val == val) {
                    cur.next = cur.next.next;
                }
            } else {
                cur = cur.next;
```

```
            }
        }

        return ans.next;
    }
}
```

OUTPUT:-





PROBLEM-7

AIM:-

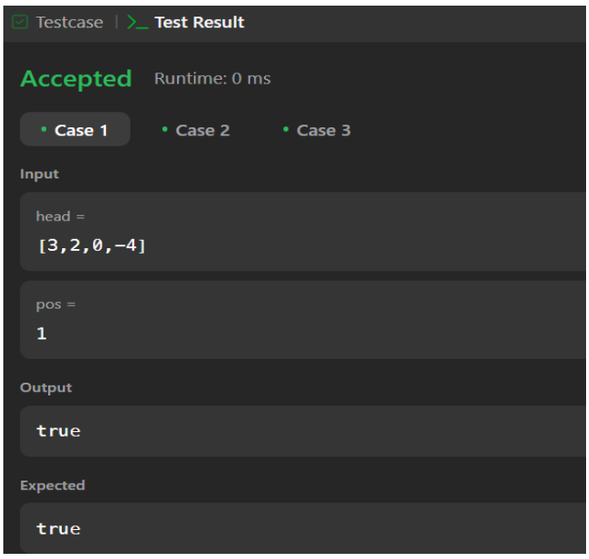
Detect a cycle in a linked list

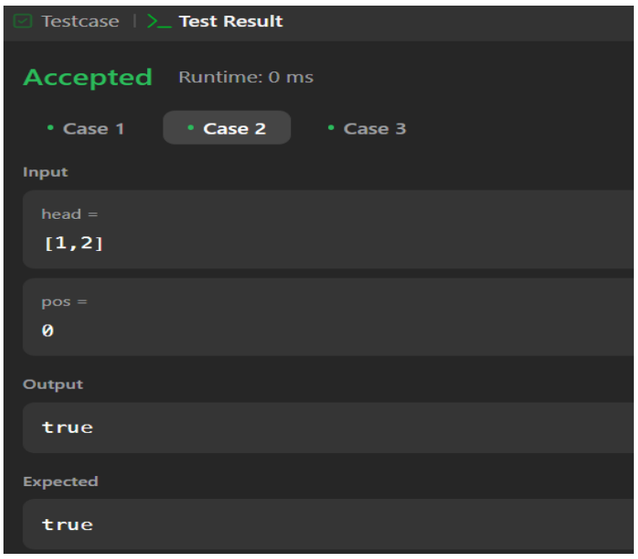
CODE:-

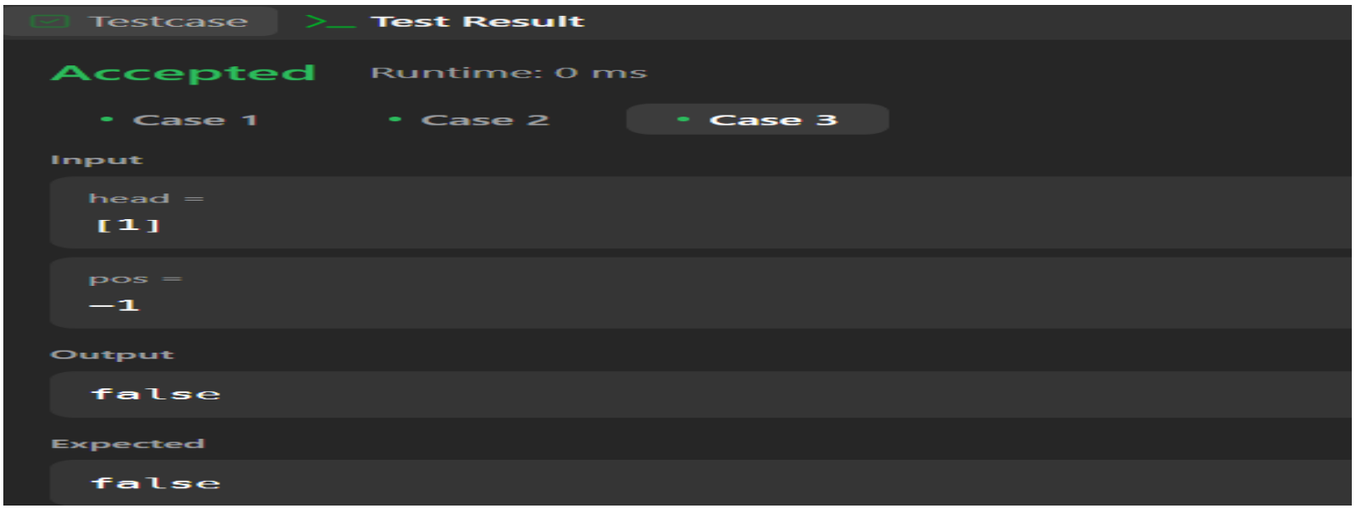
```
public class Solution {
 public boolean hasCycle(ListNode head) {
    ListNode fast = head;
    ListNode slow = head;

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

        if (fast == slow) {
            return true;
        }
    }
```

```
            return false;
        }
    }
```

OUTPUT:-







PROBLEM-8

AIM:-

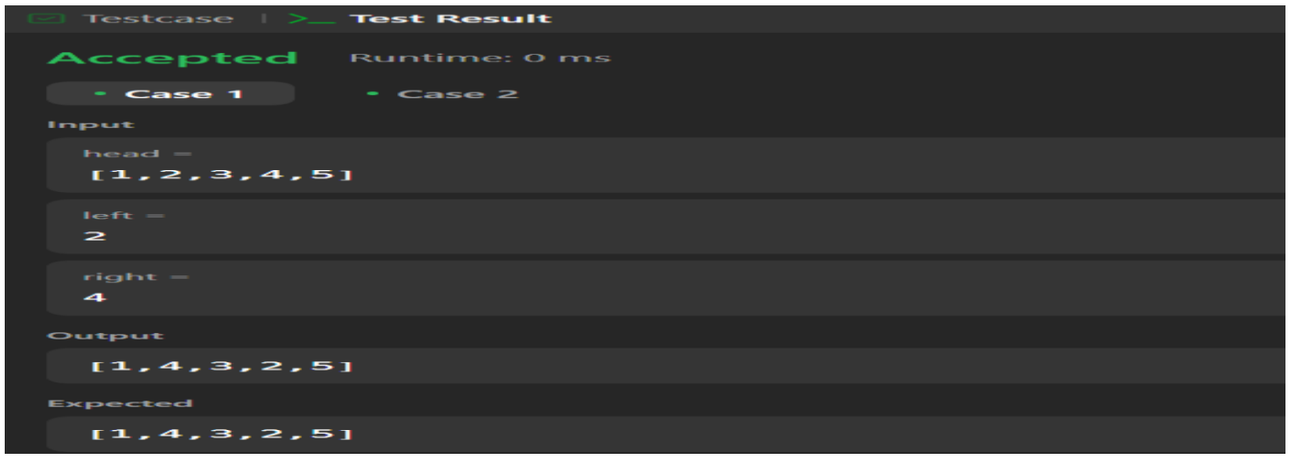
Reverse linked list 2

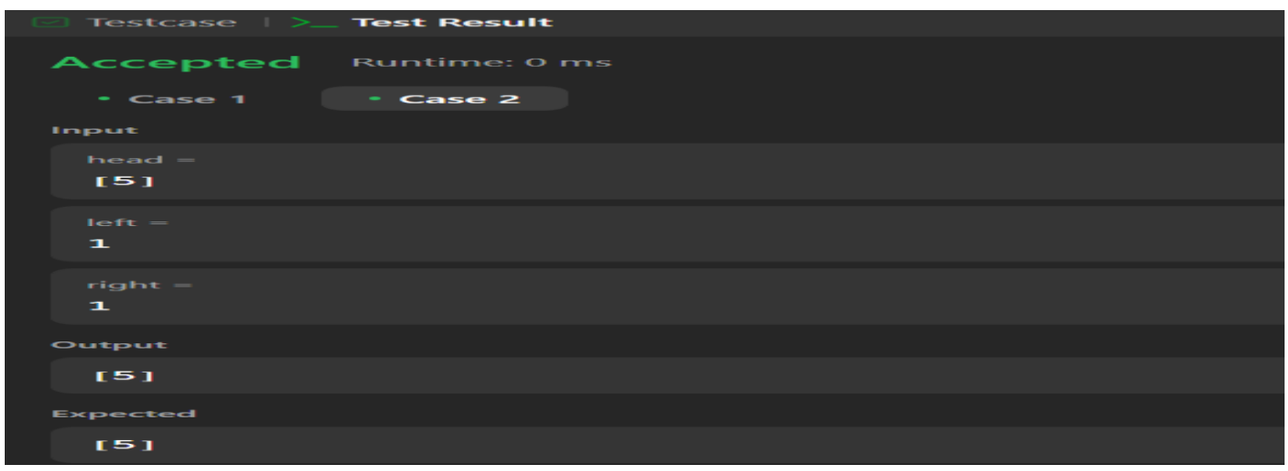
CODE:-

```
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 cur = prev.next;
        for (int i = 0; i < right - left; i++) {
```

```
            ListNode temp = cur.next;
            cur.next = temp.next;
            temp.next = prev.next;
            prev.next = temp;
        }
        return dummy.next;
    }
}
```

OUTPUT:-





PROBLEM-9

AIM:-

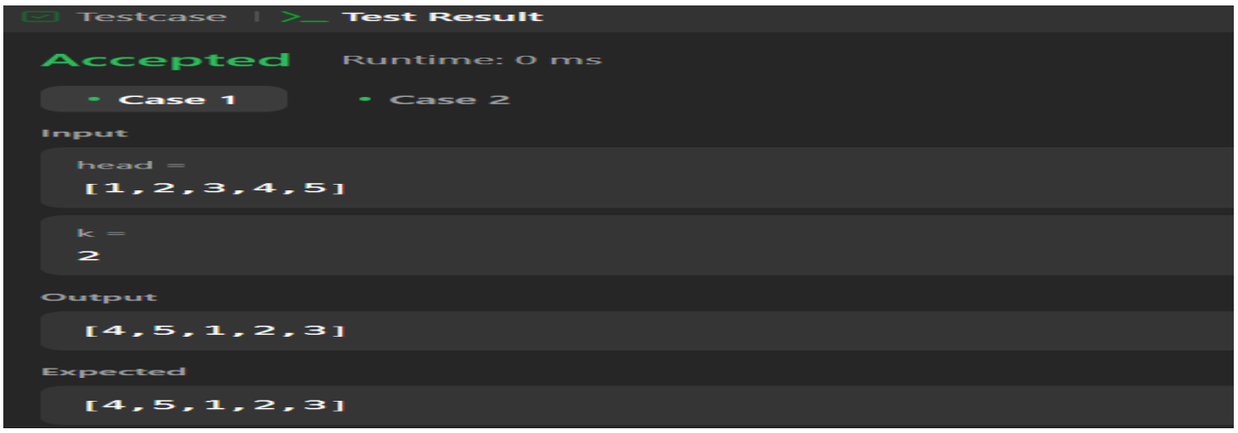
Rotate a list

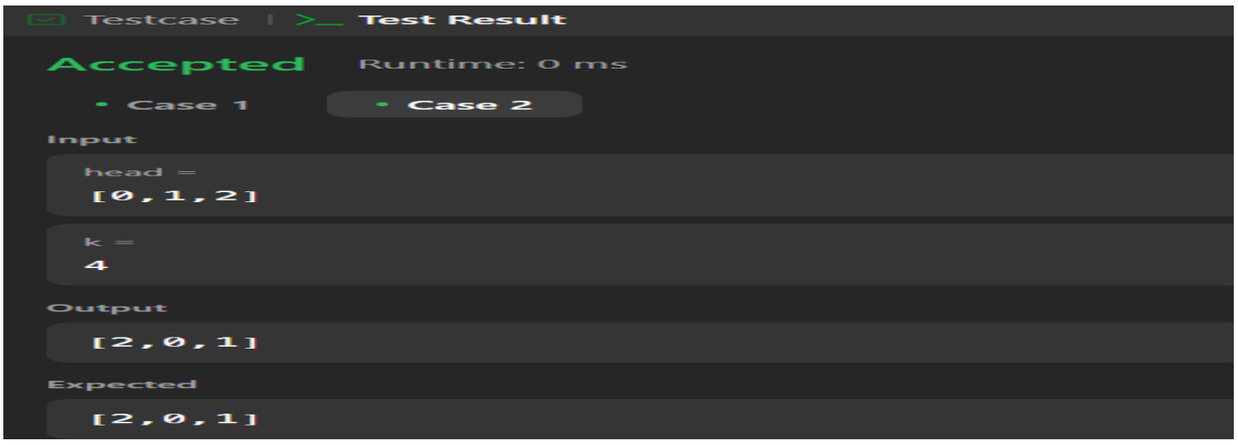
CODE:-

```
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:-





PROBLEM-10

AIM:-

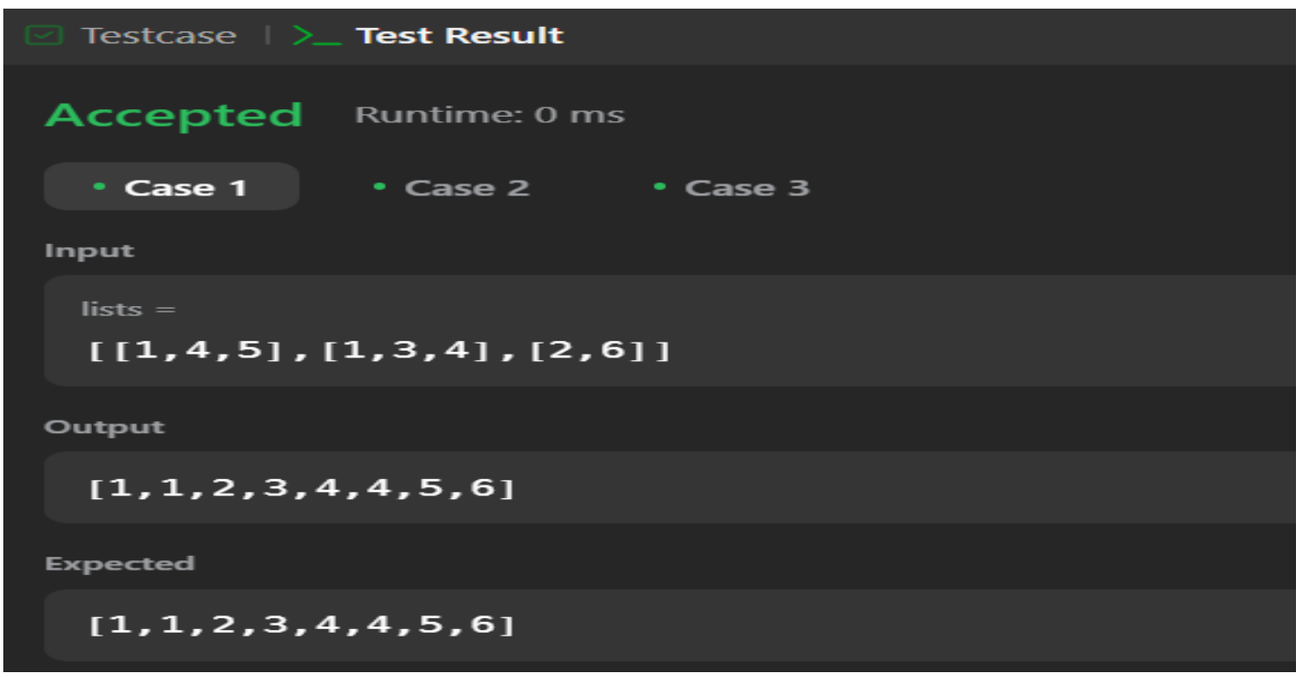
Merge k sorted lists

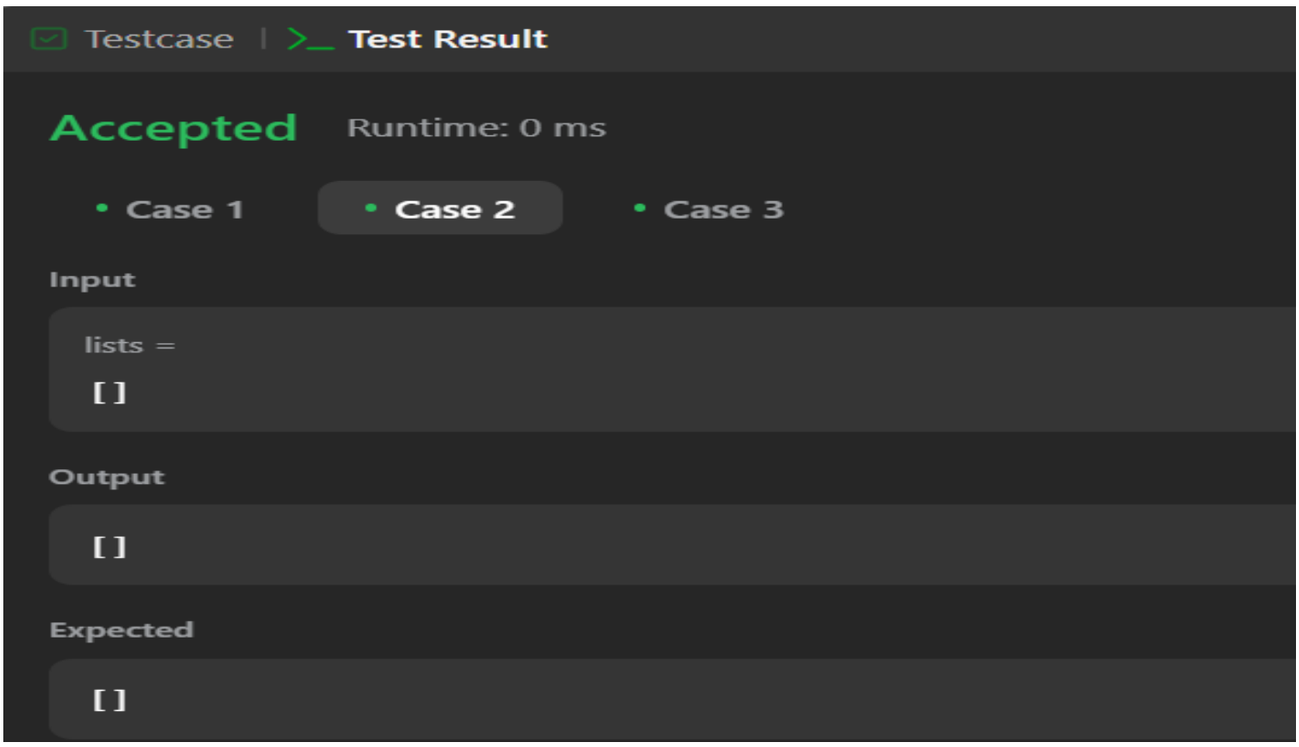
CODE:-

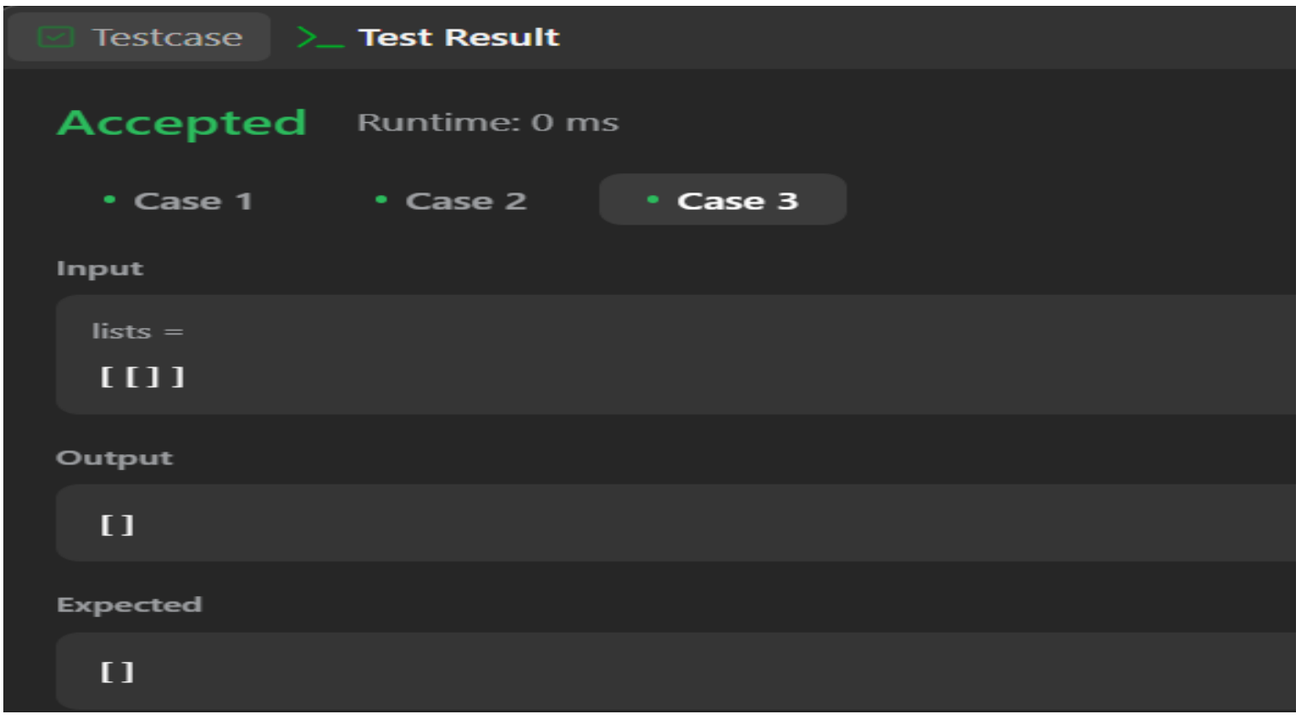
```
class Solution {
    public ListNode mergeKLists(ListNode[] lists) {
        if (lists == null || lists.length == 0) {
            return null;
        }
        return mergeKListsHelper(lists, 0, lists.length - 1);
    }

    private ListNode mergeKListsHelper(ListNode[] lists, int start, int end) {
        if (start == end) {
            return lists[start];
        }
        if (start + 1 == end) {
```

```
            return merge(lists[start], lists[end]);
        }
        int mid = start + (end - start) / 2;
        ListNode left = mergeKListsHelper(lists, start, mid);
        ListNode right = mergeKListsHelper(lists, mid + 1, end);
        return merge(left, right);
    }

    private ListNode merge(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode curr = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                curr.next = l1;
                l1 = l1.next;
            } else {
                curr.next = l2;
                l2 = l2.next;
            }
            curr = curr.next;
        }
        curr.next = (l1 != null) ? l1 : l2;
        return dummy.next;
    }
}
```

OUTPUT:-







PROBLEM-11

AIM:-

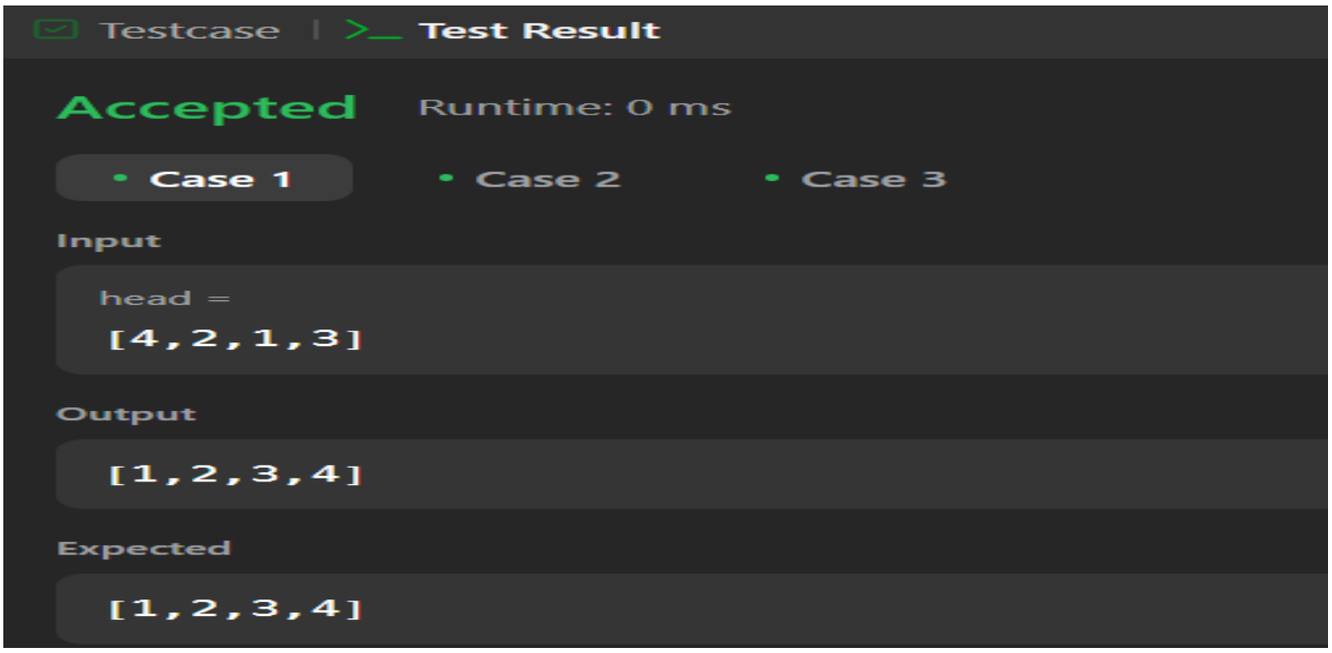
Sort List

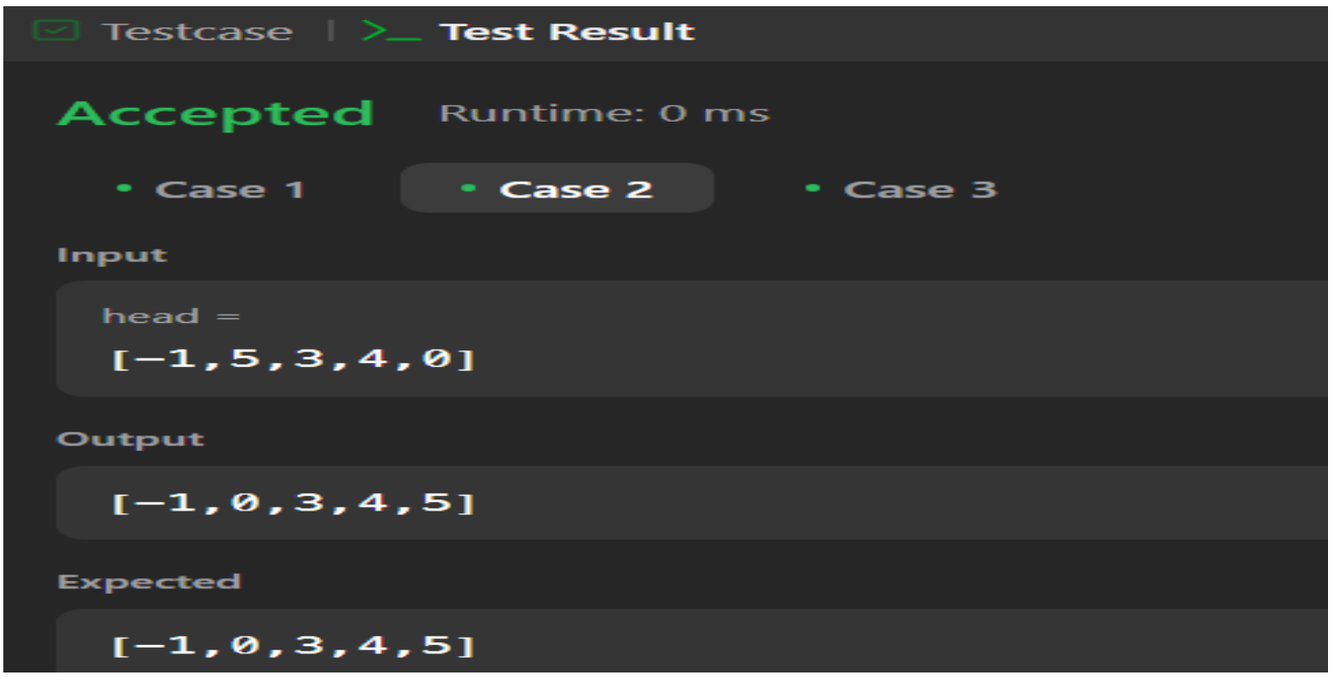
CODE:-

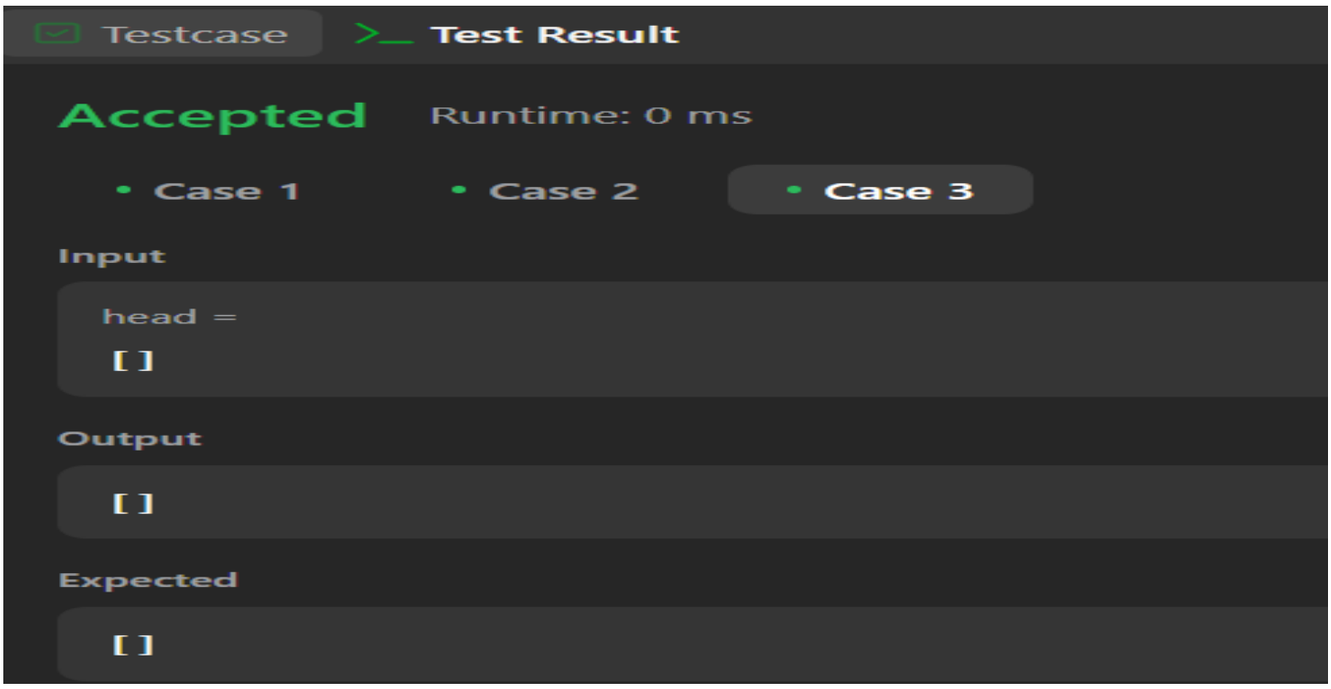
```
class Solution {
    public ListNode sortList(ListNode head) {
        if (head == null || head.next == null) return head;
        ListNode slow = head, fast = head.next;
        while (fast != null && fast.next != null) {
            slow = slow.next;
            fast = fast.next.next;
        }

        ListNode mid = slow.next;
        slow.next = null;
```

```
        ListNode left = sortList(head);
        ListNode right = sortList(mid);
        return merge(left, right);
    }
    private ListNode merge(ListNode l1, ListNode l2) {
        ListNode dummy = new ListNode(0);
        ListNode tail = dummy;

        while (l1 != null && l2 != null) {
            if (l1.val < l2.val) {
                tail.next = l1;
                l1 = l1.next;
            } else {
                tail.next = l2;
                l2 = l2.next;
            }
            tail = tail.next;
        }

        tail.next = (l1 != null) ? l1 : l2;
        return dummy.next;
    }
}
```

OUTPUT:-




Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
head =
[-1,5,3,4,0]
Output
[-1,0,3,4,5]
Expected
[-1,0,3,4,5]


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
head =
[]
Output
[]
Expected
[]