WORKSHEET 2

Student Name: Sachin **UID:** 22BCS15411

Branch: BE-CSE **Section/Group:** 22BCS_NTPP-602-A

Semester: 6th **Date of Performance:** **20**/01/2025

Subject Name: AP LAB - II **Subject Code:** 22CSP-351

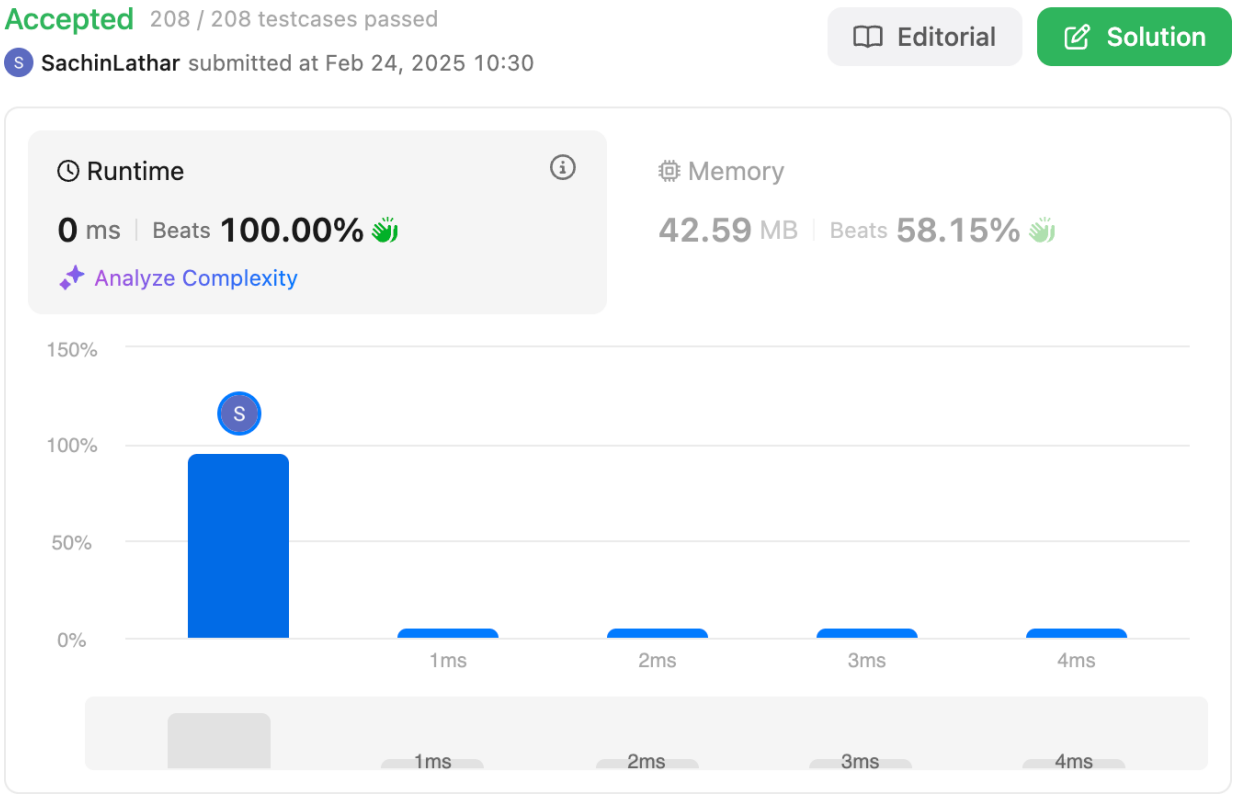
1. **Aim:** **Merge Two Sorted Lists** You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

2. **Source Code:**

```
class Solution {
    public ListNode mergeTwoLists(ListNode list1, ListNode list2)
        ListNode dummy = new ListNode(-1);
        ListNode current = dummy;
        while (list1 != null && list2 != null) {
            if (list1.val < list2.val) {
                current.next = list1;
                list1 = list1.next;
            } else {
                current.next = list2;
                list2 = list2.next;
            }
            current = current.next;
        }
        if (list1 != null) {
            current.next = list1;
        } else {
            current.next = list2;
        }

        // Return the merged list starting from the node after the dummy
        return dummy.next;
    }
}
```

3. Screenshots of outputs:



2.

Aim: Remove Duplicates from Sorted List II Given the head of a sorted linked list, delete all nodes that have duplicate numbers, leaving only distinct numbers from the original list. Return the linked list sorted as well.

Source Code:

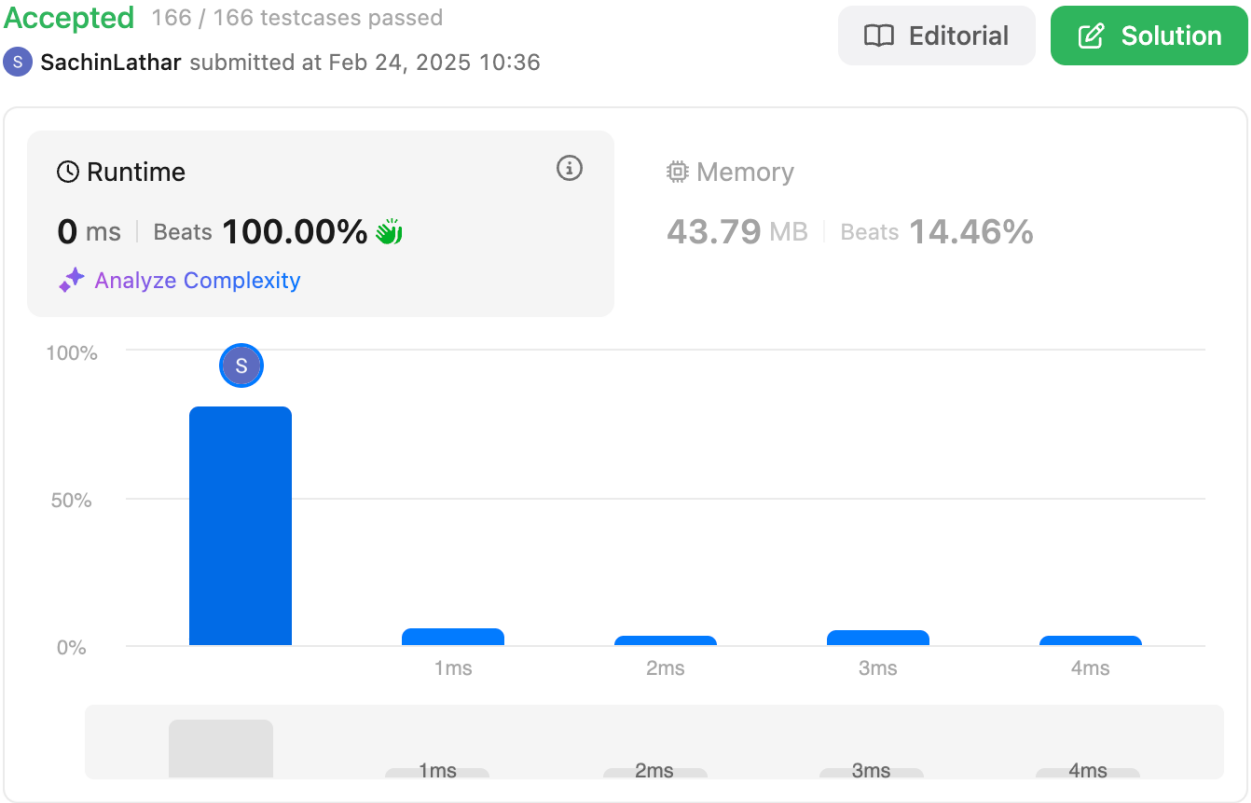
```
class Solution {
    public ListNode deleteDuplicates(ListNode head) {
        // Create a dummy node to handle edge cases such as the head being a
duplicate
        ListNode dummy = new ListNode(-1);
        dummy.next = head;
        ListNode prev = dummy;

        while (head != null) {
            // If the current node is a duplicate, skip all nodes with the
same value
            if (head.next != null && head.val == head.next.val) {
                // Skip all nodes with the same value
```

```
                while (head.next != null && head.val == head.next.val) {
                    head = head.next;
                }
                // Connect prev to the next distinct node
                prev.next = head.next;
            } else {
                // If no duplicate, just move prev to the current node
                prev = prev.next;
            }
            // Move head to the next node
            head = head.next;
        }

        // Return the merged list starting from the node after dummy
        return dummy.next;
    }
}
```

Screenshots of outputs:



3.

Aim: All O'one Data Structure Design a data structure to store the strings' count with the ability to return the strings with minimum and maximum counts.

Source Code:

```
import java.util.*;

class AllOne {
    private Map<String, Integer> keyCountMap;
    private Map<Integer, Set<String>> countKeyMap;
    private int minCount;
    private int maxCount;

    public AllOne() {
        keyCountMap = new HashMap<>();
        countKeyMap = new HashMap<>();
        minCount = 0; // Initialize to 0, as counts will never be less
        maxCount = 0; // Initialize to 0
    }

    public void inc(String key) {
        int count = keyCountMap.getOrDefault(key, 0);
        int newCount = count + 1;

        keyCountMap.put(key, newCount);

        countKeyMap.putIfAbsent(newCount, new HashSet<>());
        countKeyMap.get(newCount).add(key);

        if (count > 0) {
            countKeyMap.get(count).remove(key);
            if (countKeyMap.get(count).isEmpty()) {
                countKeyMap.remove(count);
                if (count == minCount) {
                    minCount = newCount; // Correct min update
                }
            }
        } else {
            minCount = 1; // Correctly set minCount for the first time
        }
```

```
        maxCount = Math.max(maxCount, newCount); // Simplified max update
    }

    public void dec(String key) {
        int count = keyCountMap.get(key);
        int newCount = count - 1;

        keyCountMap.put(key, newCount);

        if (newCount > 0) {
            countKeyMap.putIfAbsent(newCount, new HashSet<>());
            countKeyMap.get(newCount).add(key);
        }

        countKeyMap.get(count).remove(key);
        if (countKeyMap.get(count).isEmpty()) {
            countKeyMap.remove(count);
            if (count == minCount) {
                minCount = findNextMin(); // Find the next valid min
            }
            if (count == maxCount) {
                maxCount = findNextMax(); // Find the next valid max
            }
        }

        if (newCount == 0) {
            keyCountMap.remove(key);
        }
    }

    private int findNextMin() {
        for (int i = minCount + 1; ; i++) {
            if (countKeyMap.containsKey(i)) {
                return i;
            }
        }
    }

    private int findNextMax() {
        for (int i = maxCount - 1; i >= 1; i--) {
            if (countKeyMap.containsKey(i)) {
                return i;
```

```
            }
        }
        return 0; // Return 0 if no max exists
    }


    public String getMaxKey() {
        return (maxCount == 0 || countKeyMap.get(maxCount).isEmpty()) ? "" :
countKeyMap.get(maxCount).iterator().next();
    }

    public String getMinKey() {
        return (minCount == 0 || countKeyMap.get(minCount).isEmpty()) ? "" :
countKeyMap.get(minCount).iterator().next();
    }
}
```

4. Screenshots of outputs: