Experiment- 4

Student Name: Sahil **UID: 22BCS10928**

Branch: BE-CSE **Section/Group: KPIT_901(B)**

Semester: 6th **Date of Performance: 21/02/2025**

Subject Name: AP **Subject Code: 22CSH-359**

Problem-1

1. **Aim:** Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists.

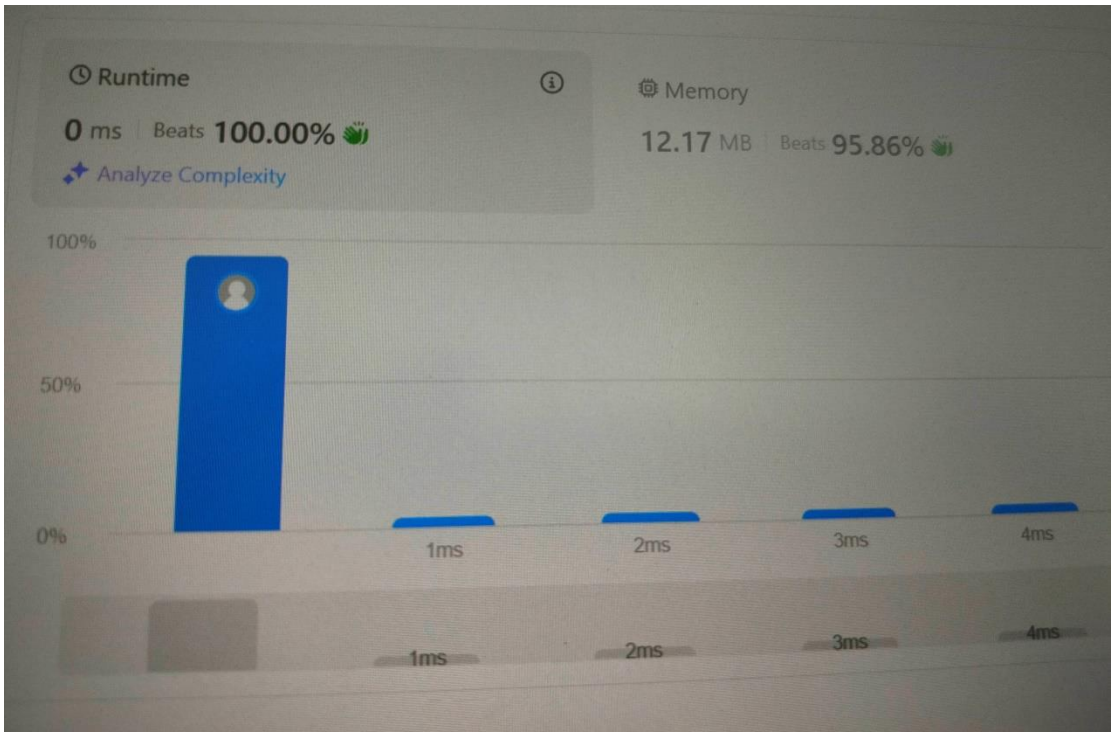
2. **Objective:** You are given the heads of two sorted linked lists list1 and list2. Merge the two lists into one sorted list. The list should be made by splicing together the nodes of the first two lists. Return the head of the merged linked list.

3. **Implementation/Code:**

```
class Solution {
public:
   ListNode* mergeTwoLists(ListNode* list1, ListNode* list2) {
      ListNode dummy;
      ListNode* tail = &dummy;

      while (list1 && list2) {
         if (list1->val < list2->val) {
            tail->next = list1;
            list1 = list1->next;
         } else {
            tail->next = list2;
            list2 = list2->next;
         }
         tail = tail->next;
      }
      tail->next = list1 ? list1 : list2;
      return dummy.next;
   }
};
```

4. Output



5. Learning Outcomes:

- Understanding linked list structure
- Implementing merging of two sorted linked lists
- Using recursion or iteration for efficient merging
- Handling edge cases like empty lists

Problem-2

1. **Aim:** Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

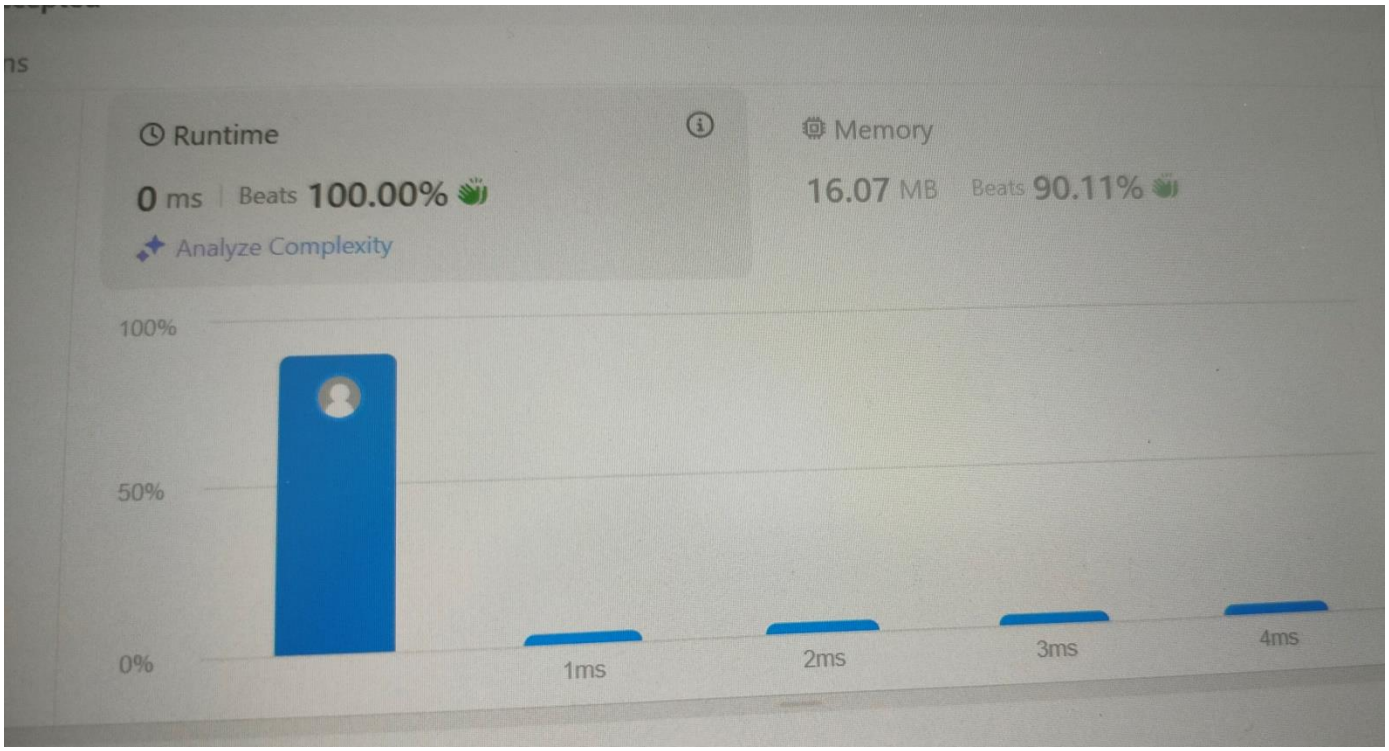
2. **Objective:**

   Given the head of a sorted linked list, delete all duplicates such that each element appears only once. Return the linked list sorted as well.

3. **Implementation/Code:**

```
class Solution {
public:
    ListNode* deleteDuplicates(ListNode* head) {
        ListNode* curr = head;
        while (curr && curr->next) {
            if (curr->val == curr->next->val) {
                ListNode* temp = curr->next;
                curr->next = curr->next->next; // Skip duplicate node
                delete temp; // Free memory
            } else {
                curr = curr->next; // Move forward
            }
        }
        return head;
    }
};
```

4.Output



5. Learning Outcomes:

- Understanding linked list traversal
- Implementing in-place duplicate removal
- Maintaining sorted order while removing duplicates
- Handling edge cases like empty lists or lists with no duplicates