Experiment- 3

Student Name: Kamal Mehta
UID: 22BET10097
Branch: B.E - IT
Section/Group: 22BET-701/A
Semester: 6th
Date of Performance: 05-02-25
Subject Name: AP Lab -2
Subject Code: 22ITP-351

1. Aim:

Problem 3.1: Remove Duplicates from Sorted List

Problem Statement: To remove duplicate nodes from a sorted singly linked list while maintaining the relative order of elements.

2. Objective:

I. Traverse the linked list and identify duplicate nodes.
II. Traverse the list while reversing the next pointers of each node.
III. Identify the middle node using the fast and slow pointer approach.
IV. Compare nodes from both lists and insert them into the new list in sorted order.
V. Identify consecutive duplicate nodes and remove them entirely.
VI. Locate the left and right positions in the linked list.

3.1. Code 3.1:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
    def deleteDuplicates(self, head: ListNode) -> ListNode:
        current = head
        while current and current.next:
```

```
            if current.val == current.next.val:
                current.next = current.next.next
            else:
                current = current.next
        return head
```

6.1. Output 3.1:

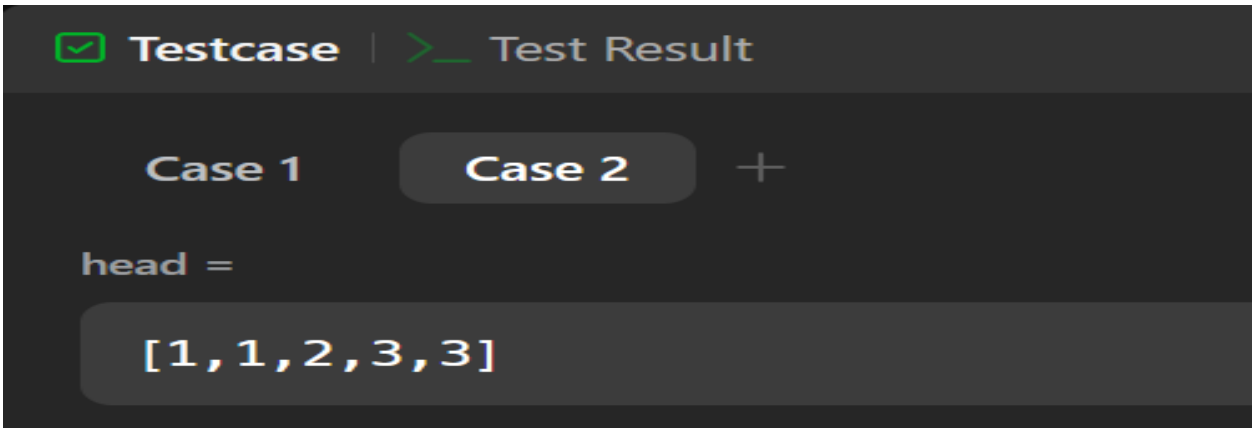

Fig 1: Output for Problem 3.1

Problem 3.2: Reverse Linked List

Problem Statement: To reverse a given singly linked list so that the last node becomes the head.

3.2. Code 3.2:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
    def reverseList(self, head: ListNode) -> ListNode:
        prev = None
        current = head
        while current:
```

```
        next_node = current.next
        current.next = prev
        prev = current
        current = next_node
    return prev
```

6.2. Output 3.2:

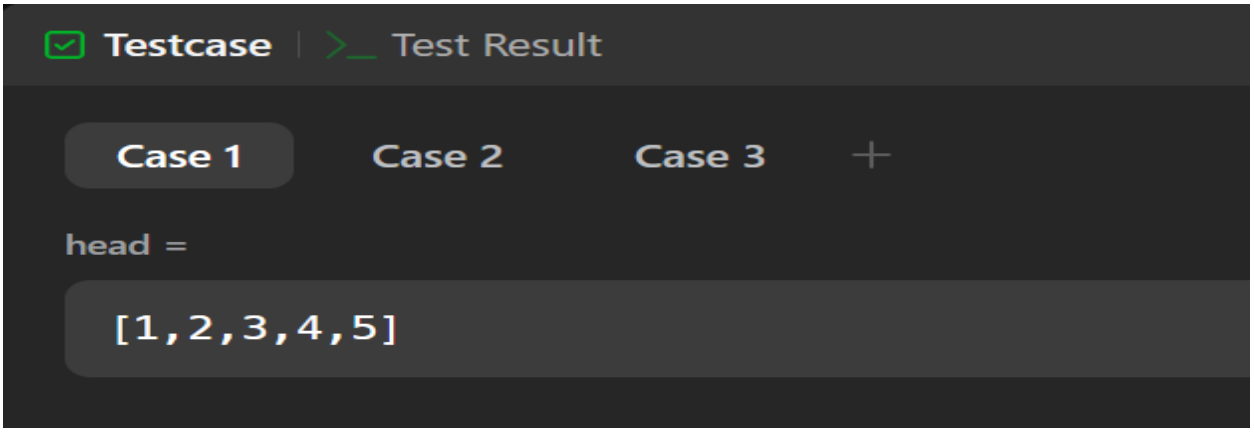

Fig 2: Output for Problem 3.2

Problem 3.3: Delete the Middle Node of a Linked List

Problem Statement: To remove the middle node of a given singly linked list efficiently.

3.3. Code 3.3:

```
class ListNode:
    def __init__(self, val=0, next=None):
        self.val = val
        self.next = next
class Solution:
```

```
def deleteMiddle(self, head: ListNode) -> ListNode:
    if not head or not head.next:
        return None
    slow, fast = head, head
    prev = None
    while fast and fast.next:
        fast = fast.next.next
        prev = slow
        slow = slow.next
    prev.next = slow.next
    return head
    return simplified_path
```

6.3. Output 3.3:

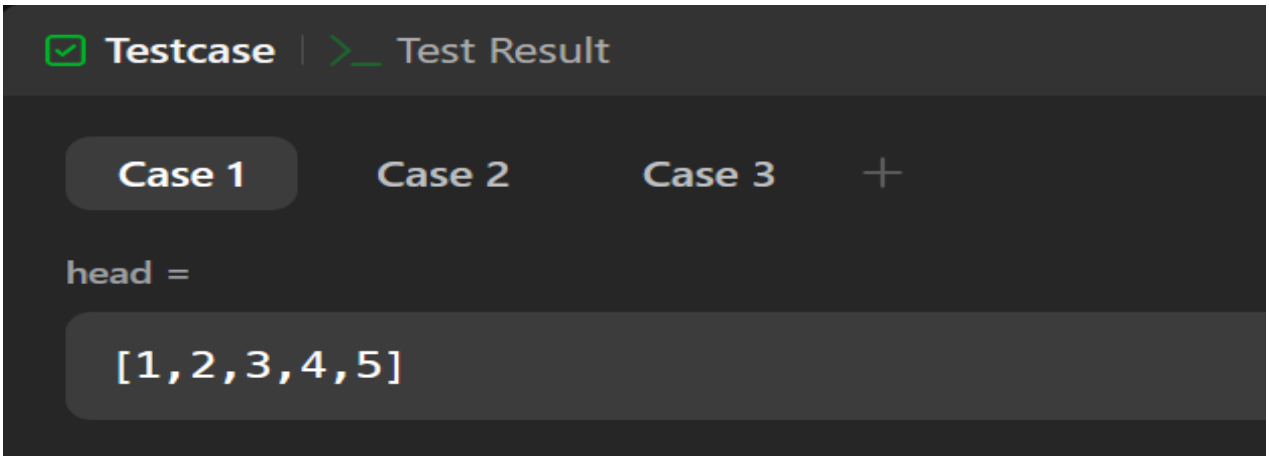

Fig 3: Output for Problem 3.3

Problem 3.4: Merge Two Sorted Lists

Problem Statement: To merge two sorted singly linked lists into a single sorted linked list.

3.4. Code 3.4:

```
class Solution:
    def mergeTwoLists(self, list1: Optional[ListNode], list2: Optional[ListNode]) ->
Optional[ListNode]:
        cur = dummy = ListNode()
        while list1 and list2:
            if list1.val < list2.val:
                cur.next = list1
                list1, cur = list1.next, list1
            else:
                cur.next = list2
                list2, cur = list2.next, list2
        if list1 or list2:
            cur.next = list1 if list1 else list2
        return dummy.next
                self.stack2.append(self.stack1.pop())
```

6.4. Output 3.4:

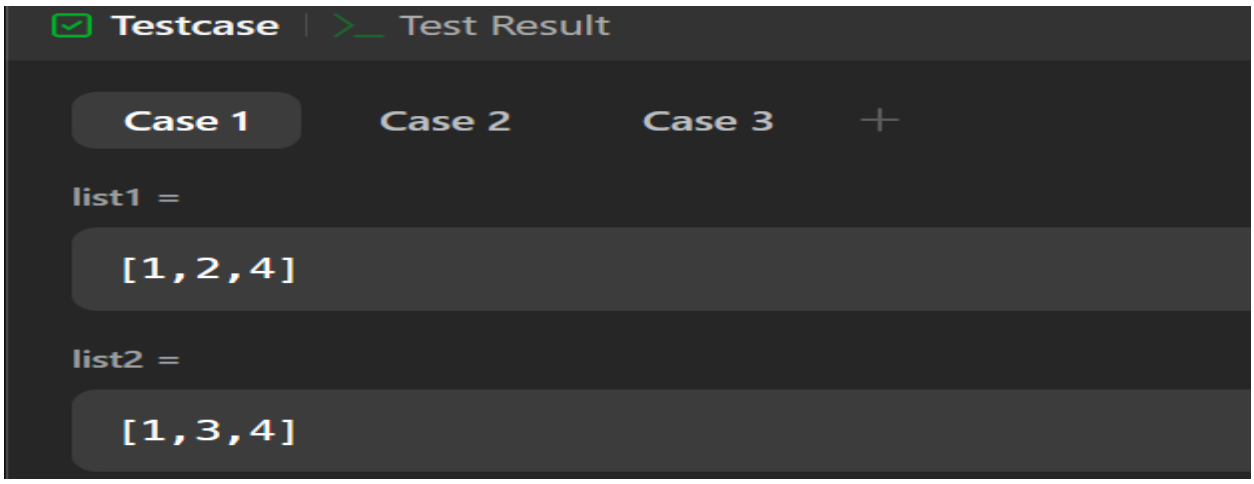

Fig 4: Output for Problem 3.4

Problem 2.5: Remove Duplicates from Sorted List II

Problem Statement: To remove all nodes with duplicate values from a sorted singly linked list, leaving only distinct elements.

3.5. Code 3.5:

```
class Solution:
  def deleteDuplicates(self, head: ListNode) -> ListNode:
    dummy = ListNode(0, head)
    prev = dummy
    while head:
      while head.next and head.val == head.next.val:
        head = head.next
      if prev.next == head:
        prev = prev.next
      else:
        prev.next = head.next
      head = head.next
    return dummy.next
```

6.5. Output 3.5:

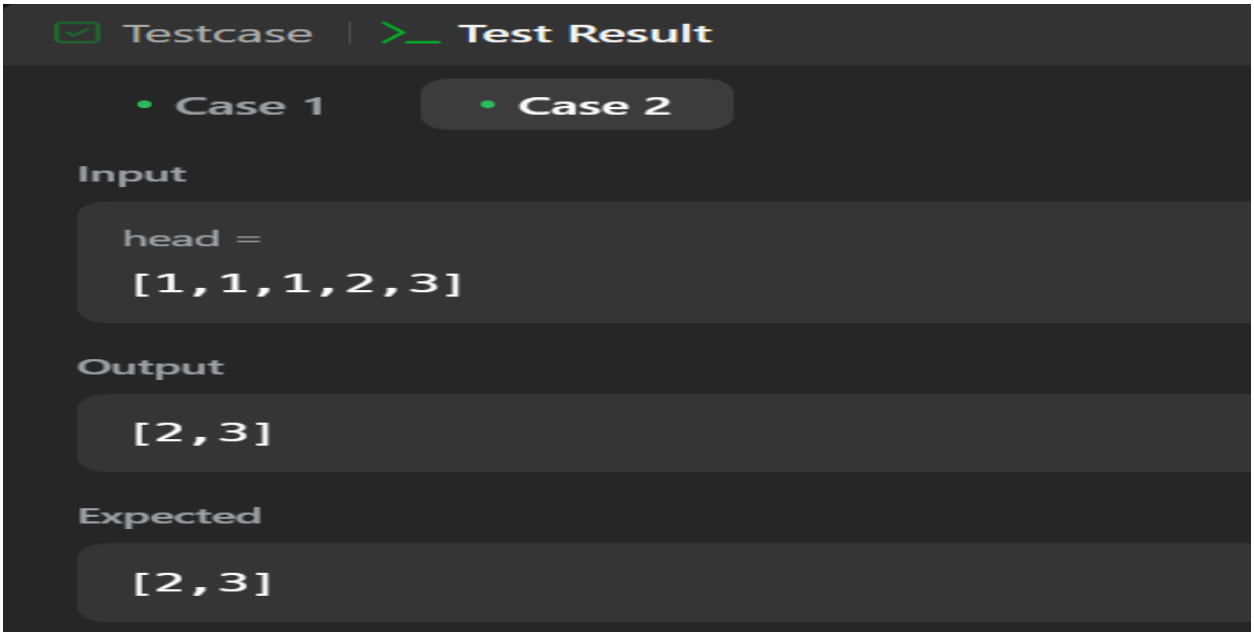

Fig 5: Output for Problem 3.5

Problem 3.6: Reverse Linked List II

Problem Statement: To reverse a specific portion of a singly linked list between given indices **left** and **right**.

3.6. Code 3.6:

```
class Solution:
  def reverseBetween(
      self,
      head: ListNode | None,
      left: int,
      right: int,
  ) -> ListNode | None:
    if left == 1:
      return self.reverseN(head, right)
```

```
    head.next = self.reverseBetween(head.next, left - 1, right - 1)
    return head
  def reverseN(self, head: ListNode | None, n: int) -> ListNode | None:
    if n == 1:
      return head
    newHead = self.reverseN(head.next, n - 1)
    headNext = head.next
    head.next = headNext.next
    headNext.next = head
    return newHead
```

6.6. Output 3.6:

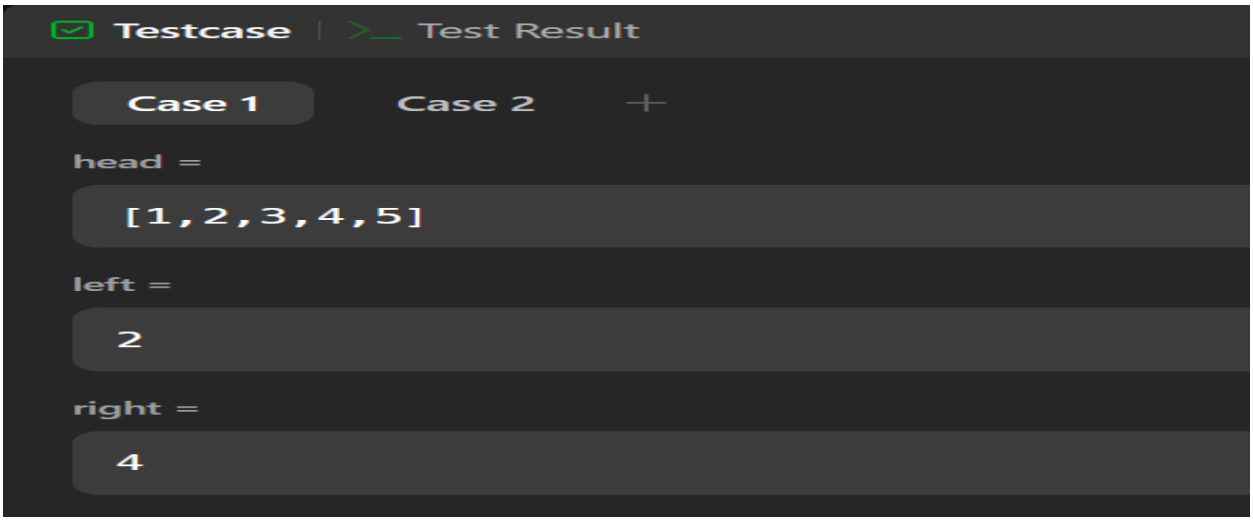

Fig 6: Output for Problem 3.6

5. Learning Outcome:

1. Gained proficiency in inserting, deleting, and modifying nodes in a singly linked list.

2. Learned how to use fast and slow pointers for optimized operations like finding the middle node.
3. Applied efficient strategies for merging sorted linked lists while maintaining order.
4. Developed the ability to handle empty lists, single-node lists, and boundary conditions.
5. Understand when to use recursion (e.g., for reversal problems) and when to prefer iteration for better space efficiency.
6. Gained a deep understanding of in-place reversal of the entire list or a specific sublist.