Practical-7

Aim : Write a Program to Merge two linked lists(sorted).

Example Linked Lists

- First List (l1): 1 -> 2 -> 4
- Second List (l2): 1 -> 3 -> 4

Merging Process

1. Initialization: Start with a dummy node and a pointer (l3) pointing to it.
2. Comparisons:
    - Compare the current nodes of l1 and l2.
    - First Comparison: Both are 1. Take from l2. Merged: 1.
    - Second Comparison: l1 is 1, l2 is 3. Take from l1. Merged: 1 -> 1.
    - Third Comparison: l1 is 2, l2 is 3. Take from l1. Merged: 1 -> 1 -> 2.
    - Fourth Comparison: l1 is 4, l2 is 3. Take from l2. Merged: 1 -> 1 -> 2 -> 3.
    - Fifth Comparison: Both are 4. Take from l2. Merged: 1 -> 1 -> 2 -> 3 -> 4.
3. Link Remaining Nodes: l2 is exhausted, link remaining 4 from l1.

Final Merged List

- Result: 1 -> 1 -> 2 -> 3 -> 4 -> 4

Algorithm:-

1. Initialization:
    - Create a dummy node (dummy) to serve as the starting point of the merged linked list. This helps simplify the merging process.

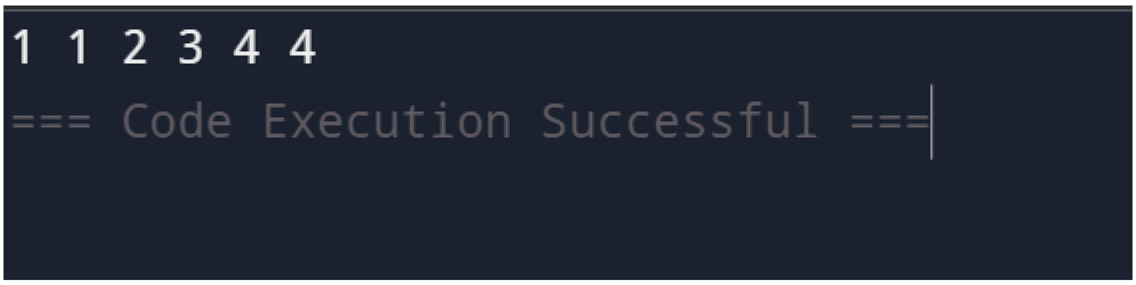
- Initialize a pointer (l3) that points to the dummy node. This pointer will be used to build the merged list.

2. Traverse Both Lists:
    - While both linked lists (l1 and l2) are not null:
        - Compare the data of the current nodes of l1 and l2.
        - If the data in l1 is less than the data in l2:
            - Set l3.next to point to the current node of l1.
            - Move l1 to its next node.
        - Otherwise:
            - Set l3.next to point to the current node of l2.
            - Move l2 to its next node.
        - Move the l3 pointer to its next node (the last node added to the merged list).
3. Link Remaining Nodes:
    - After exiting the loop, one of the lists may still have remaining nodes:
        - If l1 is not null, set l3.next to point to l1 (link the remaining nodes of l1).
        - If l2 is not null, set l3.next to point to l2 (link the remaining nodes of l2).
4. Return the Merged List:
    - Return dummy.next, which points to the head of the merged linked list (skipping the dummy node).

Program:-

```
import java.util.*;


class Node {
    int data;
    Node next;


    Node(int data) {
        this.data = data;
    }
}


public class Main {
    public static Node mergeTwoLists(Node l1, Node l2) {
        Node dummy = new Node(0); // Create a dummy node to simplify the
merging process
        Node l3 = dummy; // This will be the tail of the merged list


        // Traverse both lists and merge them in sorted order
        while (l1 != null && l2 != null) {
            if (l1.data < l2.data) {
                l3.next = l1; // Link the smaller node to the merged list
                l1 = l1.next; // Move to the next node in l1
            } else {
                l3.next = l2; // Link the smaller node to the merged list
                l2 = l2.next; // Move to the next node in l2
            }
            l3 = l3.next; // Move the tail pointer forward
        }

        // If one of the lists is not exhausted, link the remaining nodes
        if (l1 == null) {
            l3.next = l2; // If l1 is exhausted, link the rest of l2
        } else {
            l3.next = l1; // If l2 is exhausted, link the rest of l1
        }

        return dummy.next; // Return the merged list, which starts from the next of
dummy
    }

    public static void main(String[] args) {
        // Create first sorted linked list: 1 -> 2 -> 4
        Node l1 = new Node(1);
        l1.next = new Node(2);
        l1.next.next = new Node(4);

        // Create second sorted linked list: 1 -> 3 -> 4
        Node l2 = new Node(1);
        l2.next = new Node(3);
        l2.next.next = new Node(4);

        // Merge the two lists
        Node mergedList = mergeTwoLists(l1, l2);
        // Print the merged list
        while (mergedList != null) {
            System.out.print(mergedList.data + " ");
            mergedList = mergedList.next;
        }
    }
}
```

Output:-



```
1 1 2 3 4 4
=== Code Execution Successful ===
```