Write a Program to Merge Two Sorted Linked Lists

Mr. Vikas Kumar
Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Write a Program to Merge Two Sorted Linked Lists**

Lesson Objectives
To merge two sorted linked lists into a single sorted list using pointer manipulation and efficient traversal.
To handle merging conditions when one or both linked lists become empty during the merge process.
To construct two sorted linked lists for merging and initialize pointers to traverse them.
To efficiently merge two sorted linked lists into a single sorted list in O(m + n) time using pointer comparison.

Problem Statement:

Write a program to merge two sorted linked lists into one sorted linked list. The program should maintain the sorted order after merging and efficiently traverse both lists.

1. Create two sorted linked lists
2. Initialize pointers for both lists
3. Compare and attach smaller node
4. Repeat until one list ends
5. Attach remaining nodes to merged list

Concept

- A Linked List is a linear data structure where elements (nodes) are connected using pointers.

Each node consists of:

  - data: stores the element value
  - next: pointer to the next node

- When two linked lists are sorted, we can merge them efficiently by comparing their nodes one by one and arranging them in increasing order — similar to the merge step in Merge Sort.

.

Algorithm/Logic

1. Initialize:

- Create two sorted linked lists — list1 and list2.
- Create a new pointer result (head of merged list).

2. Compare Nodes:

If both list1 and list2 are not empty:

- If list1->data <= list2->data,
    - Set result = list1
    - Move list1 = list1->next
- Else
    - Set result = list2
    - Move list2 = list2->next

Algorithm/Logic

3. Repeat Comparison:
Continue comparing nodes and linking the smaller node to the merged list until one of the lists becomes empty.

4. Append Remaining Nodes:
If any nodes are left in list1 or list2,
attach them directly to the end of the merged list.

5. Return Result:
Return the head pointer of the merged sorted linked list.

Visualization

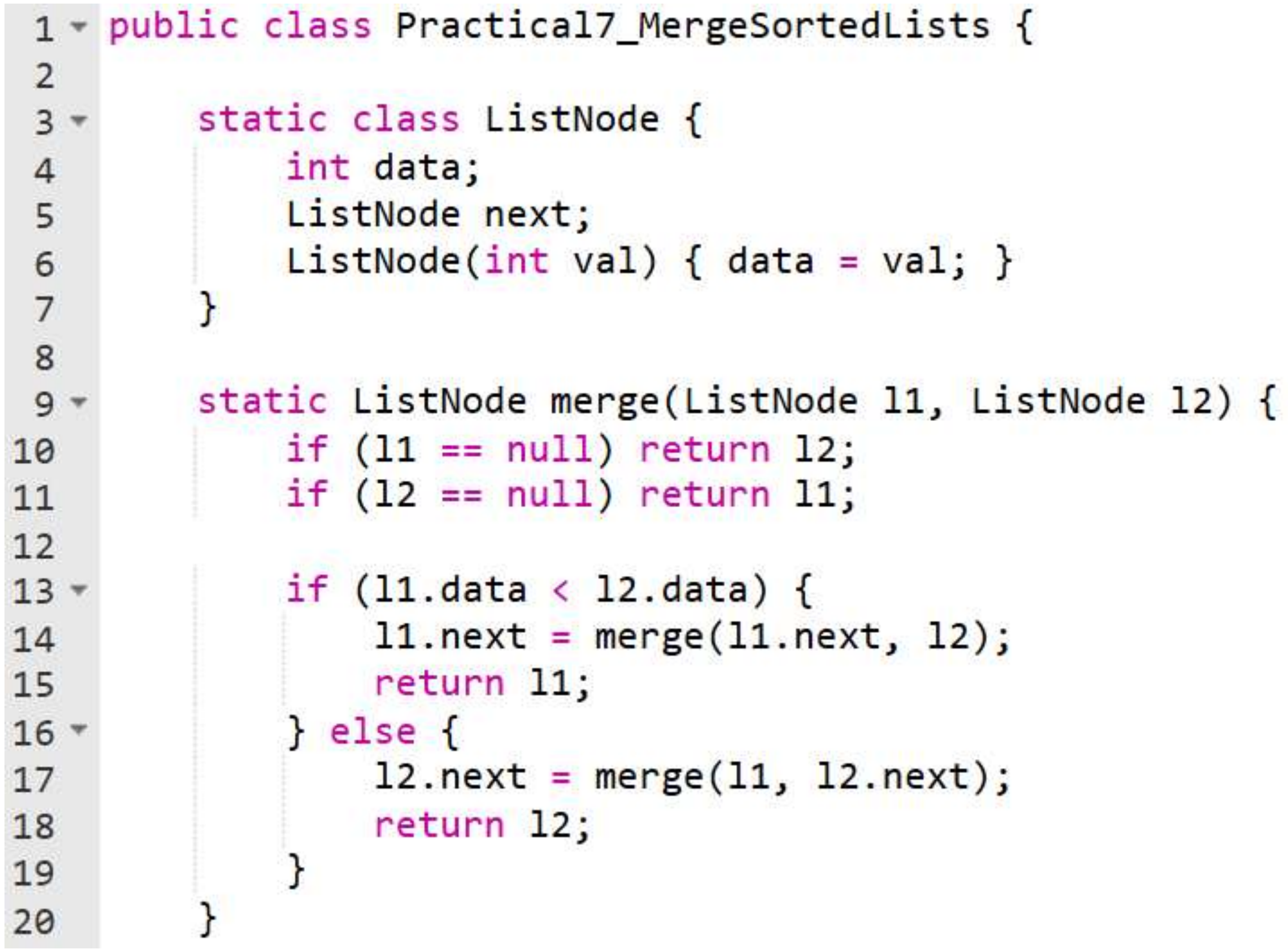
- **Example:**

List 1: 2 → 5 → 8

List 2: 1 → 3 → 7 → 9

Step	**List 1**	**List 2**	**comparison**	**Node Added to Merged List**	**Merged List (current state)**
1	2 → 5 → 8	1 → 3 → 7 → 9	2>1	1	1
2	2 → 5 → 8	3 → 7 → 9	2<3	2	1 → 2
3	5 → 8	3 → 7 → 9	5>3	3	1 → 2 → 3
4	5 → 8	7 → 9	5<7	5	1 → 2 → 3 →5
5	8	7 → 9	8>7	7	1 → 2 → 3 → 5 → 7
6	8	9	8<9	8	1 → 2 → 3 → 5 → 7 → 8
7	--	9	Only one node left	9	1 → 2 → 3 → 5 → 7 → 8 → 9

Code Implementation

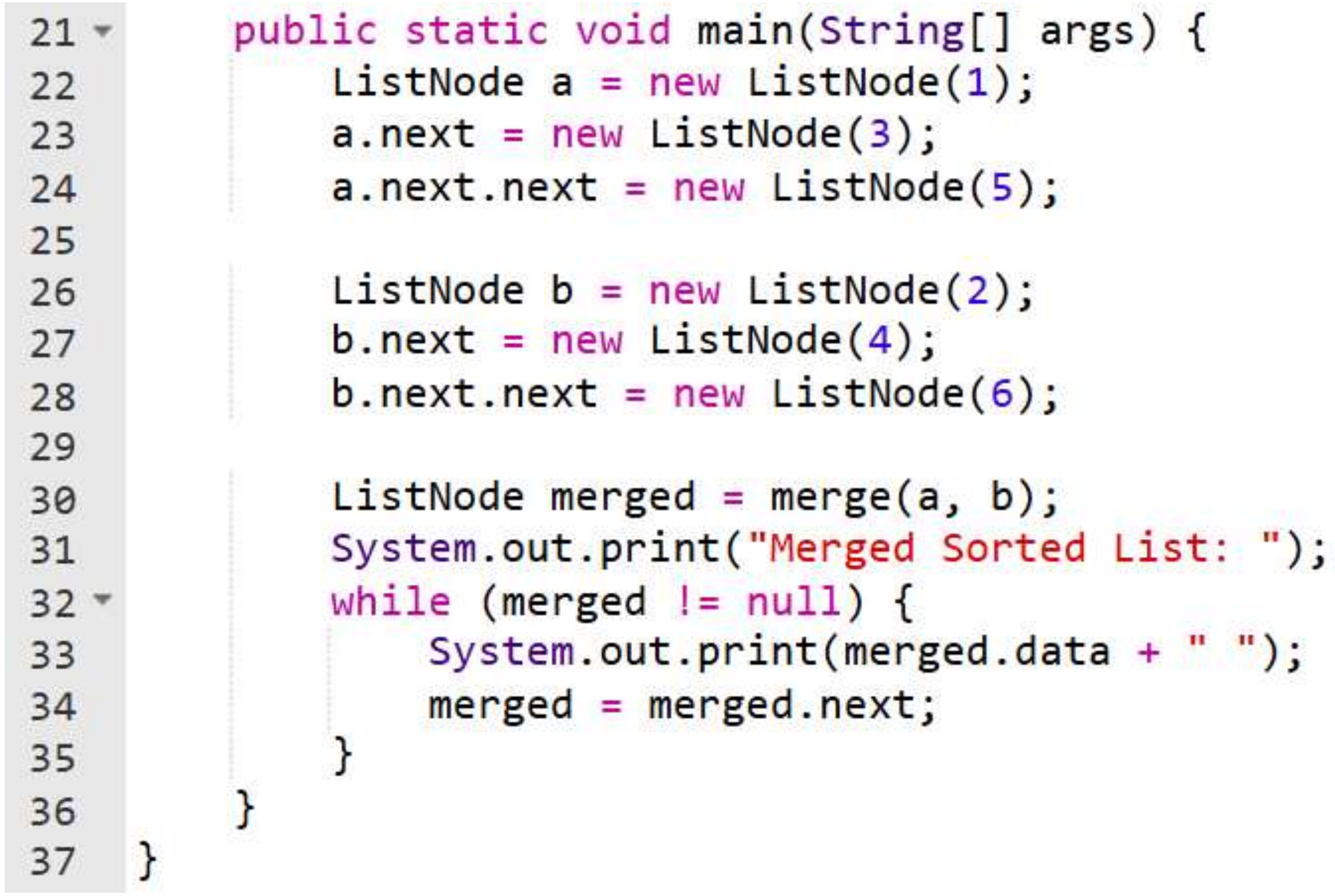


```java
public class Practical7_MergeSortedLists {

    static class ListNode {
        int data;
        ListNode next;
        ListNode(int val) { data = val; }
    }

    static ListNode merge(ListNode l1, ListNode l2) {
        if (l1 == null) return l2;
        if (l2 == null) return l1;

        if (l1.data < l2.data) {
            l1.next = merge(l1.next, l2);
            return l1;
        } else {
            l2.next = merge(l1, l2.next);
            return l2;
        }
    }
```

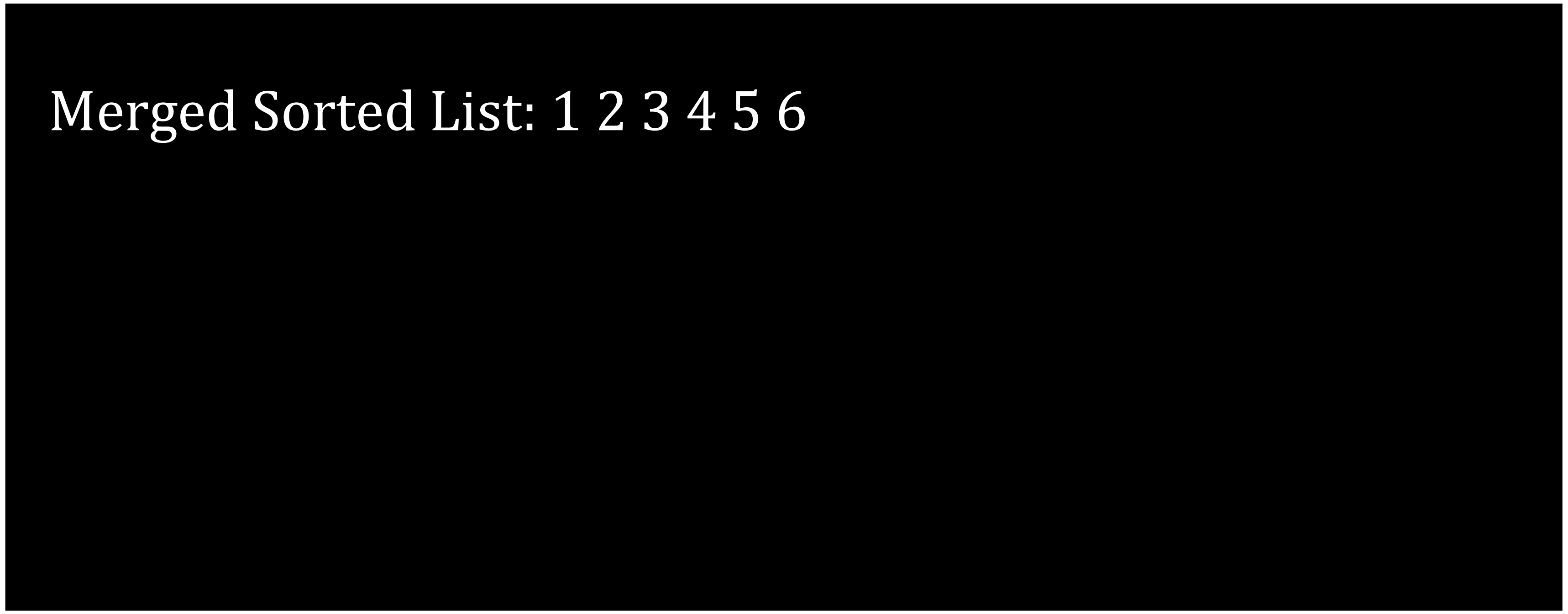


```
    public static void main(String[] args) {
        ListNode a = new ListNode(1);
        a.next = new ListNode(3);
        a.next.next = new ListNode(5);

        ListNode b = new ListNode(2);
        b.next = new ListNode(4);
        b.next.next = new ListNode(6);

        ListNode merged = merge(a, b);
        System.out.print("Merged Sorted List: ");
        while (merged != null) {
            System.out.print(merged.data + " ");
            merged = merged.next;
        }
    }
}
```

Output :



```
Merged Sorted List: 1 2 3 4 5 6
```

Example Walkthrough

Example:

List 1: 1 → 3 → 5 → 7

List 2: 2 → 4 → 6 → 8

Step	**List 1**	**List 2**	**comparison**	**Node Added to Merged List**	**Merged List (current state)**
1	1 → 3 → 5	2 → 4 → 6	1<2	1	1
2	3 → 5	4 → 6	3>2	2	1 → 2
3	3 → 5	4 → 6	3<4	3	1 → 2 → 3
4	5	6	5>4	4	1 → 2 → 3 → 4
5	5	6	5<6	5	1 → 2 → 3 → 4 → 5
6	--	--	Only 6 left	6	1 → 2 → 3 → 4 → 5 → 6

Time & Space Complexity

Operation	Time Complexity	Space Complexity	Why?
Merge Two Sorted Linked Lists (Iterative)	O(m+n)	O(1)	Each node from both lists is compared and merged once.
Merge Two Sorted Linked Lists (Recursive)	O(m+n)	O(m+n)	Recursive calls are made for each node until both lists are merged.

Summary

- Merging two sorted linked lists efficiently combines both lists into a single sorted list using pointer comparison.
  At each step, the smaller node is linked to the result list, ensuring order is maintained.
- This operation runs in O(m + n) time and is fundamental in algorithms like Merge Sort, demonstrating efficient linked list manipulation.

Summary

- It uses pointer manipulation to compare nodes from both lists and attaches the smaller one to the new list, continuing until all nodes are merged.
- This method avoids using extra space and maintains the sorted order efficiently.
  The algorithm runs in linear time O(m + n), making it optimal for large datasets.
- It is widely used in merge sort, data merging applications, and linked list-based systems to maintain ordered data dynamically.

Practice Questions:

1️⃣ **Merge Two Sorted Lists — LeetCode #21**

🔗 https://leetcode.com/problems/merge-two-sorted-lists/

Concept: Recursively or iteratively merge two sorted linked lists.

Why Practice: Directly matches the concept taught in this practical.

Practice Questions:

2 Merge k Sorted Lists — LeetCode #23

🔗 https://leetcode.com/problems/merge-k-sorted-lists/

Concept: Merge multiple sorted lists using a priority queue.

Why Practice: Extends the same concept to handle multiple linked lists efficiently.

Practice Questions:

3 Sort List — LeetCode #148

🔗 https://leetcode.com/problems/sort-list/

Concept: Apply merge sort algorithm on a linked list.

Why Practice: Builds upon merging logic to perform complete list sorting.

Thanks