Write a program to verify and validate mirrored trees.

Mr. Vikas Kumar
Assistant Professor
Industry Embedded Program

Lesson Plan

Subject/Course	**Competitive Coding**
Lesson Title	**Verify and Validate if Two Binary Trees are Mirror of Each Other**

Lesson Objectives
Understand the concept of mirror trees and structural symmetry in binary trees.
Learn how to compare two binary trees node-by-node recursively.
Implement an efficient algorithm to verify if two trees are mirrors of each other.
Analyze time and space complexity for recursive tree comparison.

Problem Statement:

Given two binary trees, determine whether they are **mirror images** of each other or not.

Return true if both trees are mirror images; otherwise, return false.

Concept

- Two binary trees are said to be **mirror images** of each other if:
- Their **root nodes** have the same value, and
- The **left subtree** of one tree is a **mirror** of the **right subtree** of the other (and vice versa).

- This property helps in understanding **tree symmetry**, which is widely used in **graphics, structural analysis, and data reflection problems**.

Algorithm/Logic

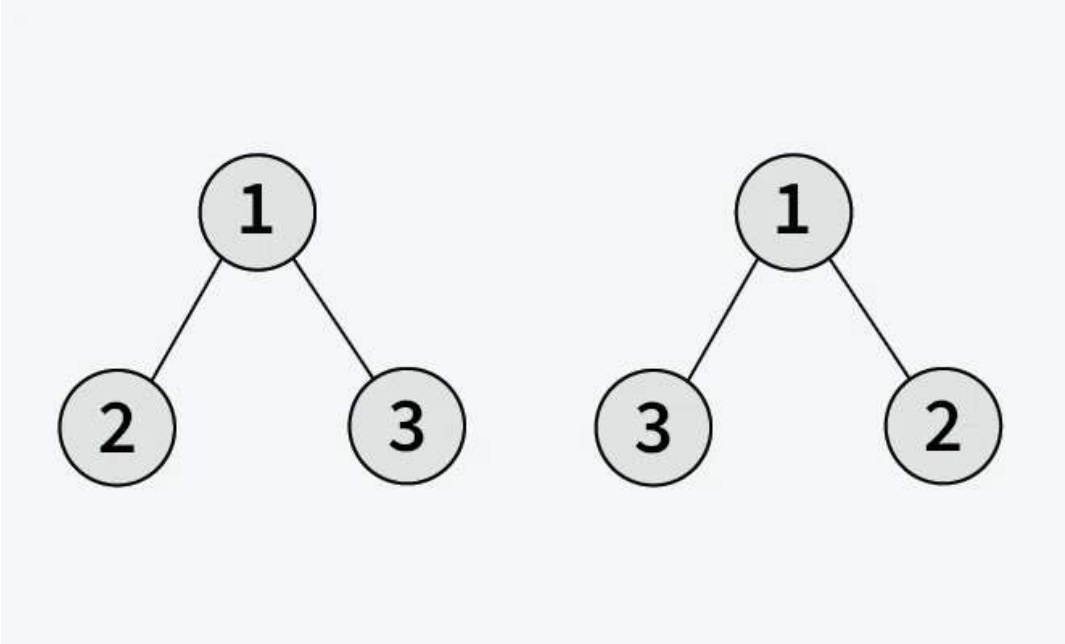
1. If both trees are empty → return **true**.
2. If one is empty and the other isn't → return **false**.
3. If the root values are different → return **false.**
4. Recursively check:

   Left subtree of first tree ↔ Right subtree of second tree

   Right subtree of first tree ↔ Left subtree of second tree

5. Return true only if both checks are true.

Visualization



Output: True
Explanation: *Both trees are mirror images of each other, so output is True*

Code Implementation

```
public class MirrorTrees {

    static class Node {
        int data;
        Node left, right;
        Node(int val) { data = val; }
    }

    static boolean areMirror(Node t1, Node t2) {
        if (t1 == null && t2 == null) return true;
        if (t1 == null || t2 == null) return false;
        return (t1.data == t2.data) &&
                areMirror(t1.left, t2.right) &&
                areMirror(t1.right, t2.left);
    }
```

```
    public static void main(String[] args) {
        Node t1 = new Node(1);
        t1.left = new Node(2);
        t1.right = new Node(3);

        Node t2 = new Node(1);
        t2.left = new Node(3);
        t2.right = new Node(2);

        if (areMirror(t1, t2))
            System.out.println("The two trees are mirror images.");
        else
            System.out.println("The two trees are NOT mirror images.");
    }
}
```

Output :

```
The two trees are mirror images.
```

Time & Space Complexity

Operation	Time Complexity	Space Complexity	Explanation
Mirror Check	O(n)	O(h)	Every node is compared once; recursion depth = height of tree (h).

Summary

- Two trees are **mirrors** if their structures and node values are **symmetrically opposite**.
- Recursive comparison ensures efficient and clean checking.
- **Time Complexity:** O(n)
- **Space Complexity:** O(h)
- Widely applied in **UI rendering, data reflection, and symmetric tree validation problems**

Practice Questions:

◆ **Symmetric Tree— LeetCode #101**

🔗 *https://leetcode.com/problems/symmetric-tree/*

Concept:

Check whether a binary tree is **symmetric (mirror of itself)** using recursive or iterative methods.

Why Practice:

- Strengthens understanding of **recursive tree comparison**.
- Builds logic for **mirror structure detection** and **balanced tree problems**.

Thanks