1. My idea:
   1. Make recursive dfs to find all leaf node. Leaf node is node has no left & right child
   2. Check size & value of 2 array
2. World idea:

**Approach 1: Depth First Search**

**Intuition and Algorithm**

Let's find the leaf value sequence for both given trees. Afterwards, we can compare them to see if they are equal or not.

To find the leaf value sequence of a tree, we use a depth first search. Our dfs function writes the node's value if it is a leaf, and then recursively explores each child. This is guaranteed to visit each leaf in left-to-right order, as left-children are fully explored before right-children.

class Solution {

public:

    bool leafSimilar(TreeNode\* root1, TreeNode\* root2) {

        vector<int> leaves1;

        vector<int> leaves2;

        dfs(root1, leaves1);

        dfs(root2, leaves2);

        return leaves1 == leaves2;

    }

    void dfs(TreeNode\* node, vector<int>& leaves) {

        if (node == NULL) return;

        if (node->left == NULL && node->right == NULL)

            leaves.push\_back(node->val);

        dfs(node->left, leaves);

        dfs(node->right, leaves);

    }

};

**Complexity Analysis**

* Time Complexity: O(T1+T2)O(T\_1 + T\_2)*O*(*T*1​+*T*2​), where T1,T2T\_1, T\_2*T*1​,*T*2​ are the lengths of the given trees.
* Space Complexity: O(T1+T2)O(T\_1 + T\_2)*O*(*T*1​+*T*2​), the space used in storing the leaf values.