
My Notes on Checking if Two Binary Trees are the Same (LeetCode Problem 100)

1. Problem Statement

Given the roots of two binary trees p and q, write a function to check if they are the same or not.

Two binary trees are considered the same if:

- They are structurally identical.
- The values of corresponding nodes are the same.

Example:

Tree 1: Tree 2:

```
  1        1
 / \      / \
2   3    2   3
```

Output: true (Both trees are identical)

---

2. My Thought Process

When I saw this problem, I immediately thought of a recursive approach:

1. If both trees are NULL, return true (both are empty, hence same).
2. If one tree is NULL but the other is not, return false (one is missing nodes).
3. If both trees have a node but values don't match, return false.
4. Recursively check the left and right subtrees.
5. If both left and right subtrees match, return true.

This approach ensures that we check each node one by one, maintaining **O(N) time complexity**, where N is the number of nodes.

---

3. My Code

```
class Solution {
public:
    bool isSameTree(TreeNode* p, TreeNode* q) {
        if (p == NULL && q == NULL) return true;
        if (p == NULL && q != NULL) return false;
        if (p != NULL && q == NULL) return false;
```

```
        if (p->val != q->val) return false;

        bool leftAns = isSameTree(p->left, q->left);
        if (leftAns == false) return false;

        bool rightAns = isSameTree(p->right, q->right);
        if (rightAns == false) return false;

        return true;
    }
};
```

---

4. What I Realized?

- This approach is **simple and efficient**.
- The base cases handle NULL nodes and mismatches correctly.
- The function calls traverse both trees simultaneously, ensuring we compare each corresponding node.
- The time complexity is **O(N)**, which is optimal since we need to check all nodes.

---

5. Conclusion

- This problem is a good example of recursion in trees.
- The approach ensures **efficient comparison** of two binary trees.
- The time complexity **O(N)** makes it scalable.
- Feeling confident about recursion after solving this one! 🚀 🔥