Same Tree

|	
⊙ Difficulty	Easy
≔ Category	Tree
🔗 Question	https://leetcode.com/problems/same-tree/
🔗 Solution	https://youtu.be/vRbbcKXCxOw
⋰ Status	Not started

Question

> 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, and the nodes have the same value.
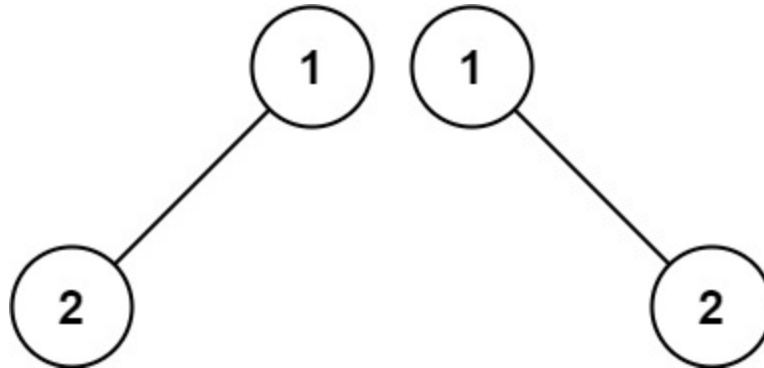
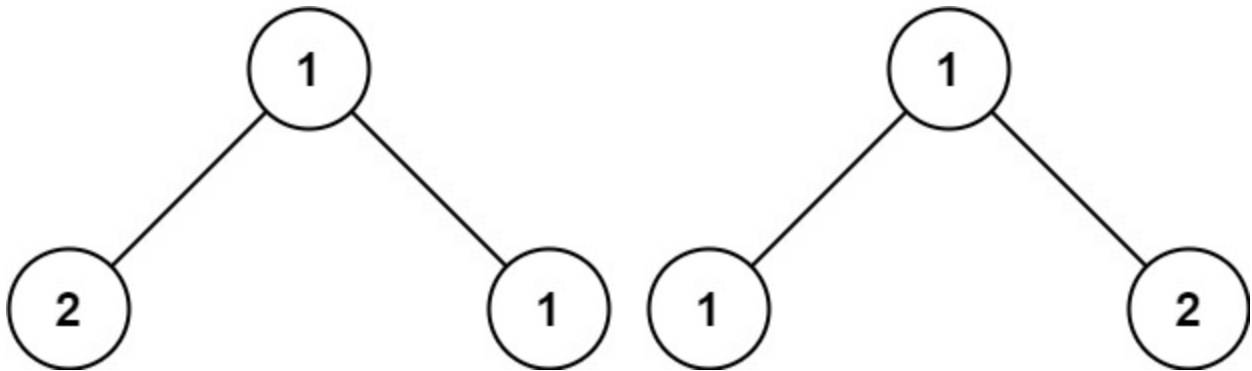## Example

Example 1:
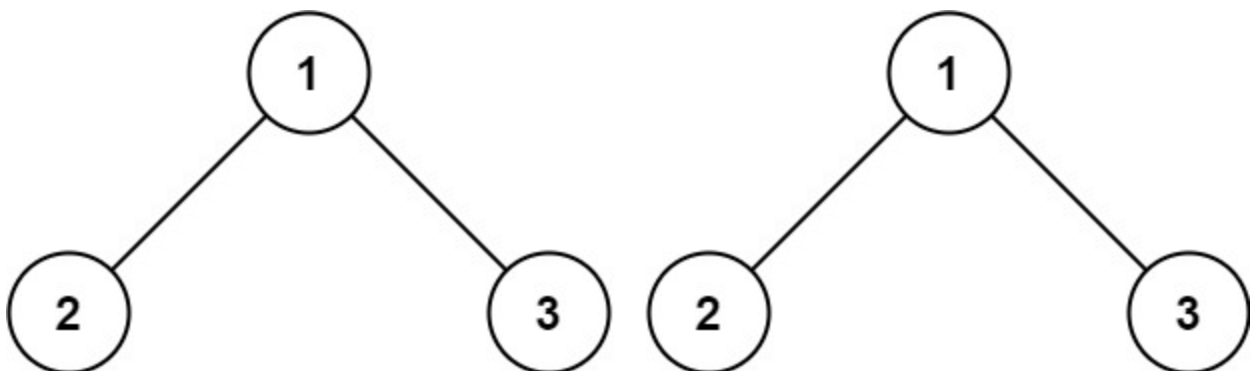

```
Input: p = [1,2,3], q = [1,2,3]
```

```
Output: true
```

Example 2:


```
Input: p = [1,2], q = [1,null,2]
Output: false
```

Example 3:


```
Input: p = [1,2,1], q = [1,1,2]
Output: false
```

Idea

💡 Both node null → same; One of the node null → not same; Node val not equal → not same; Iterate left branch and right branch accordingly

Solution

```python
class Solution:
    def isSameTree(self, p: Optional[TreeNode], q: Optional[TreeNode]) -> bool:

        # If both nodes are None, they are the same (base case).
        if not p and not q:
            return True

        # If either p or q is None or their values are not equal, they are not the same.
        if p and q and p.val == q.val:
            # Recursively check the left and right subtrees.
            return self.isSameTree(p.left, q.left) and self.isSameTree(p.right, q.right)

        else:
            # If any of the above conditions is not met, the trees are not the same.
            return False
```

Explanation

Time Complexity:

- The time complexity of this solution is O(n), where n is the number of nodes in the binary trees. This is because we visit each node once during the recursive traversal.

Space Complexity:

- The space complexity is O(h), where h is the height of the binary trees. In the worst case, if the trees are completely unbalanced (skewed), the space complexity will be O(n), but in balanced trees, it will be O(log(n)).