145. Binary Tree Postorder Traversal

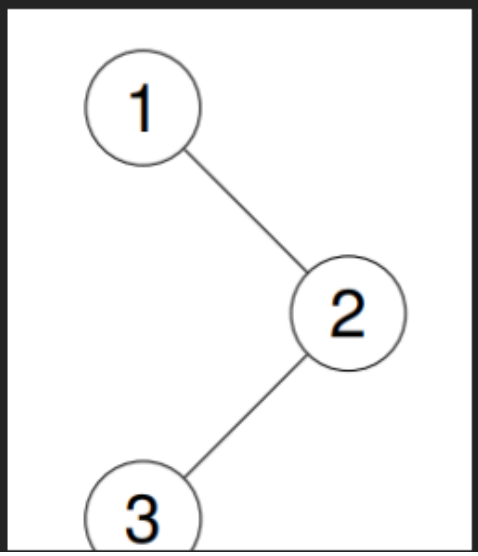
Solved

Easy | Topics | Companies

Given the `root` of a binary tree, return *the postorder traversal of its nodes' values.*

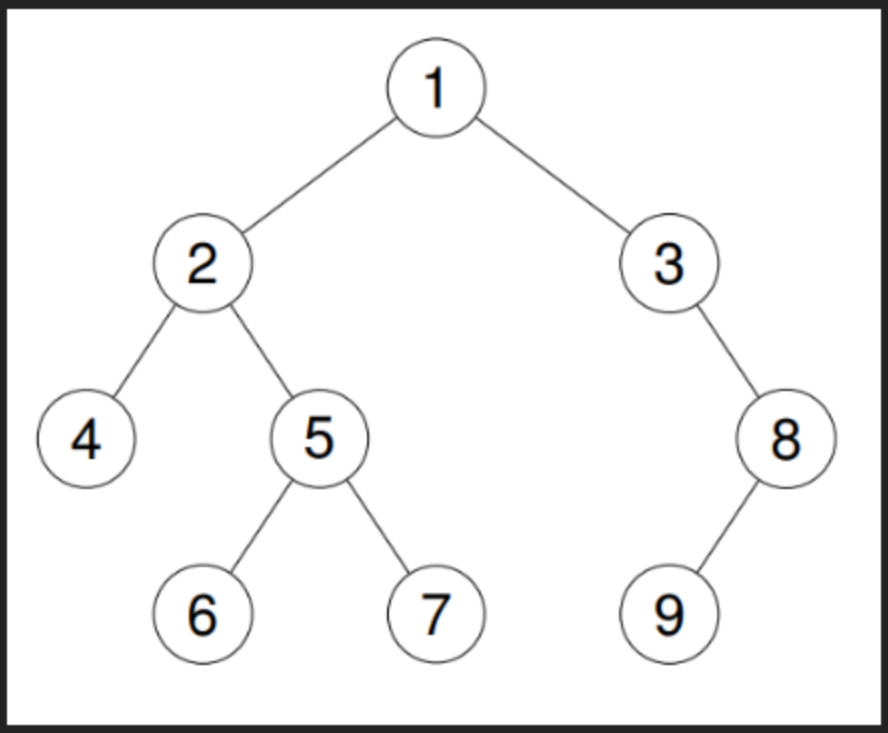
Example 1:

> **Input:** `root = [1,null,2,3]`
>
> **Output:** `[3,2,1]`
>
> **Explanation:**



Example 2:

> **Input:** `root = [1,2,3,4,5,null,8,null,null,6,7,9]`
>
> **Output:** `[4,6,7,5,2,9,8,3,1]`
>
> **Explanation:**



Example 3:

> **Input:** `root = []`
>
> **Output:** `[]`

Example 4:

> **Input:** `root = [1]`
>
> **Output:** `[1]`

Constraints:

- The number of the nodes in the tree is in the range `[0, 100]`.
- `-100 <= Node.val <= 100`

Follow up: Recursive solution is trivial, could you do it iteratively?

Python:

```
# Definition for a binary tree node.
# class TreeNode:
#     def __init__(self, val=0, left=None, right=None):
#         self.val = val
#         self.left = left
#         self.right = right

from typing import Optional, List

class Solution:
    # Recursive solution
    def postorderTraversal_recursive(self, root: Optional[TreeNode]) -> List[int]:
        res = []

        def dfs(node):
            if not node:
                return
            dfs(node.left)
            dfs(node.right)
            res.append(node.val)
```

```
        dfs(root)
        return res

    # Iterative solution
    def postorderTraversal(self, root: Optional[TreeNode]) -> List[int]:
        if not root:
            return []

        stack, result = [root], []

        while stack:
            node = stack.pop()
            result.append(node.val)
            # Push left first, so right is processed first
            if node.left:
                stack.append(node.left)
            if node.right:
                stack.append(node.right)

        # Reverse the result to get Left → Right → Root
        return result[::-1]
```

JavaScript:

```
var postorderTraversal = function(root) {
    let result = [];

    function dfs(node) {
        if (!node) return;
        dfs(node.left);     // visit left
        dfs(node.right);    // visit right
        result.push(node.val); // visit root
    }

    dfs(root);
    return result;
};
```

Java:

```
import java.util.*;

class Solution {
    public List<Integer> postorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
```

```
        helper(root, result);
        return result;
    }

    private void helper(TreeNode node, List<Integer> result) {
        if (node == null) return;

        helper(node.left, result);   // Left
        helper(node.right, result);  // Right
        result.add(node.val);        // Root
    }
}
```