

13  
.3

# Postorder Traversal

EASY

## ■ What You Will Learn

- ✓ Postorder: Left → Right → Root — root comes LAST
- ✓ Why Postorder is perfect for deletion and bottom-up problems
- ✓ The two-stack iterative trick
- ✓ Real-world analogy: computing directory sizes

## ■ Intuition First

Postorder means root comes POST (after) everything else. Process all children before the parent. Think of it like cleaning up a meal: you clear plates from the children first, then finally clear the parent's plate (the table).

Perfect for: computing subtree values, deleting a tree (delete children before parent), evaluating expression trees (compute operands before operator).

## ■ The Pattern — Left → Right → Root

- Process left subtree completely
- Process right subtree completely
- Visit root last
- Use case: delete tree, evaluate expressions, compute subtree sums

## ■ Visual

Sample tree

```
1
 / \
2   3
 / \
4   5
```

Postorder: 4 → 5 → 2 → 3 → 1

(leaves come first, root comes very last)

## ■ Dry Run

|               |                                                                |
|---------------|----------------------------------------------------------------|
| <b>Step 1</b> | Go left to 2, left to 4. Visit 4 (leaf). result=[4]            |
| <b>Step 2</b> | Back to 2. Go right to 5. Visit 5 (leaf). result=[4,5]         |
| <b>Step 3</b> | Back to 2. Both children done. Visit 2. result=[4,5,2]         |
| <b>Step 4</b> | Back to 1. Go right to 3. 3 is leaf. Visit 3. result=[4,5,2,3] |
| <b>Step 5</b> | Back to 1. Both children done. Visit 1. result=[4,5,2,3,1] ✓   |

## ■ Java Code — Recursive

```
public static List<Integer> postorderTraversal(TreeNode root) {
    List<Integer> result = new ArrayList<>();
    postorder(root, result);
    return result;
}

private static void postorder(TreeNode node, List<Integer> result) {
    if (node == null) return;
    postorder(node.left, result); // L – process left subtree
    postorder(node.right, result); // R – process right subtree
    result.add(node.val); // N – visit root LAST
}
```

## ■ Java Code — Iterative (Two Stack Trick)

Insight: Postorder (L→R→Root) is the REVERSE of (Root→R→L). So: do a modified preorder into stack2, then pop stack2!

```
public static List<Integer> postorderIterative(TreeNode root) {
    List<Integer> result = new ArrayList<>();
    if (root == null) return result;
    Stack<TreeNode> stack1 = new Stack<>();
    Stack<TreeNode> stack2 = new Stack<>();
    stack1.push(root);
    while (!stack1.isEmpty()) {
        TreeNode node = stack1.pop();
        stack2.push(node); // Don't add to result yet
        if (node.left != null) stack1.push(node.left);
        if (node.right != null) stack1.push(node.right);
    }
    // Pop stack2 to get Postorder
    while (!stack2.isEmpty()) {
        result.add(stack2.pop().val);
    }
    return result;
}
```

```
}
```

■ ■ Complexity

| Case      | Time | Space | What it means                                       |
|-----------|------|-------|-----------------------------------------------------|
| All Cases | O(n) | O(n)  | Two stacks, but each holds at most O(n) nodes total |

■ The Two-Stack Insight

- Postorder ( $L \rightarrow R \rightarrow N$ ) is the mirror of ( $N \rightarrow R \rightarrow L$ )
- ( $N \rightarrow R \rightarrow L$ ) is just a modified preorder: visit root, go right, go left
- Do that modified preorder and push results onto stack2
- Popping stack2 reverses it  $\rightarrow$  gives correct  $L \rightarrow R \rightarrow N$  postorder

■ ■ Common Mistakes

- Deleting a node before its children in a tree deletion algorithm — always postorder!
- Single-stack iterative postorder is tricky — two-stack is much cleaner
- In expression trees: evaluating operator before operands gives wrong result

■ Pro Tips

- Any 'bottom-up' tree problem likely needs postorder thinking
- Subtree sum, subtree size, tree height — all computed postorder
- Expression tree evaluation: leaf = operand, internal = operator  $\rightarrow$  postorder gives correct result

■ Visual Thinking

Imagine a company org chart. Postorder = every employee submits their report before their manager.

Data flows bottom-up: leaves  $\rightarrow$  branches  $\rightarrow$  root.

Animation: nodes fade from leaves upward. Root is the last to glow.

Expression tree: imagine bubbles carrying values floating UP from leaves to operators.

■ Practice Problems

[Easy] Postorder of tree [1,null,2,3] — trace manually

[Med] Sum of all left leaves using postorder

[Med] Delete a binary tree safely (postorder is why!)

[Hard] Evaluate an arithmetic expression tree using postorder

### ■ Quick Revision Box

- Postorder = Left → Right → Root (root LAST)
- Recursive: left, right, then visit
- Iterative: two-stack trick (reverse of modified preorder)
- $O(n)$  time,  $O(n)$  space
- Bottom-up problems: height, size, deletion, expression eval