

13
.2

Preorder Traversal

EASY

■ What You Will Learn

- ✓ Preorder: Root → Left → Right — root comes first
- ✓ Recursive and iterative implementations
- ✓ Why Preorder is used to clone/serialize trees
- ✓ Comparison with Inorder and Postorder

■ Intuition First

Preorder means the root comes PRE (before) everything else. You visit the node first, then dive into its left subtree, then its right subtree. Think of it like exploring a maze — you mark where you ARE before deciding where to go next.

Preorder is perfect for copying or printing a tree because you capture the root before its children — which is exactly the order you'd need to reconstruct it.

■ The Pattern — Root → Left → Right

- Visit root FIRST (mark your current position)
- Then recurse into left subtree fully
- Then recurse into right subtree fully
- Use case: clone a tree, serialize a tree, print directory structure

■ Visual

Sample tree

```
1
 / \
2   3
 / \ \
4  5 6
```

Preorder: 1 → 2 → 4 → 5 → 3 → 6

Inorder: 4 → 2 → 5 → 1 → 3 → 6

Postorder: 4 → 5 → 2 → 6 → 3 → 1

■ Dry Run

Step 1	Visit 1 (root). result=[1]. Go left to 2.
Step 2	Visit 2. result=[1,2]. Go left to 4.
Step 3	Visit 4. result=[1,2,4]. No left/right. Backtrack to 2.
Step 4	Go right to 5. Visit 5. result=[1,2,4,5]. Backtrack to 1.
Step 5	Go right to 3. Visit 3. result=[1,2,4,5,3]. Go right to 6.
Step 6	Visit 6. result=[1,2,4,5,3,6]. Done ✓

■ Java Code — Recursive

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

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

■ Java Code — Iterative (Stack)

Iterative trick: push RIGHT first, then LEFT — because Stack is LIFO, left gets processed first.

```
public static List<Integer> preorderIterative(TreeNode root) {
    List<Integer> result = new ArrayList<>();
    if (root == null) return result;
    Stack<TreeNode> stack = new Stack<>();
    stack.push(root);
    while (!stack.isEmpty()) {
        TreeNode node = stack.pop();
        result.add(node.val); // Visit immediately on pop
        // Push right FIRST so left is processed first (LIFO)
        if (node.right != null) stack.push(node.right);
        if (node.left != null) stack.push(node.left);
    }
    return result;
}
```

■ Complexity

Case	Time	Space	What it means
All Cases	$O(n)$	$O(h)$	Every node visited once. Stack holds at most $O(h)$ nodes (h = height)

■ Preorder = Serialization

- If you serialize a tree using preorder, you can reconstruct it perfectly
- Root always comes first → you know what to create before its children
- This is how XML/JSON tree structures are written — parent before children
- BST preorder uniquely identifies the tree (no duplicates needed)

■ Common Mistakes

- Iterative: pushing LEFT first instead of RIGHT — reverses the output
- Confusing with Inorder — the only difference is WHEN you visit the root
- Forgetting null checks before pushing children to stack

■ Pro Tips

- Preorder + Inorder together uniquely reconstruct any binary tree
- Directory listing (`ls -R`) uses preorder — show folder before contents
- Preorder iterative is the simplest iterative tree traversal

■ Visual Thinking

Picture an explorer entering a forest. Preorder = wave a flag the moment you step into any clearing.

You wave flag at root → dive deep left → wave left nodes → backtrack → dive right → wave right nodes.

Animation: node glows the moment it's 'entered' — before going to children.

Compare with a camera that takes a photo (inorder = takes photo as you leave the left subtree).

■ Practice Problems

[Easy] Preorder of a single-node tree [10]?

[Med] Reconstruct binary tree from Preorder + Inorder arrays

[Med] Path sum — use preorder DFS to explore all root-to-leaf paths

[Hard] Serialize a binary tree using preorder (encode + decode)

■ Quick Revision Box

- Preorder = Root → Left → Right
- Recursive: visit node, recurse left, recurse right
- Iterative: stack, push right then left (LIFO order)
- $O(n)$ time, $O(n)$ space
- Used for cloning, serialization, directory printing
- Preorder + Inorder = unique tree reconstruction