LeetCode 94 — Binary Tree Inorder Traversal

🔗 https://leetcode.com/problems/binary-tree-inorder-traversal/

1. Problem Title & Link

Title: LeetCode 94: Binary Tree Inorder Traversal
Link: https://leetcode.com/problems/binary-tree-inorder-traversal/

2. Problem Statement (Short Summary)

Given the root of a binary tree, return its **inorder traversal**.

Traversal order:

➡ Left → Node → Right

3. Examples (Input → Output)

Example 1

Tree:

```
 1
  \
   2
  /
 3
```

Output:

[1,3,2]

Example 2

Input: root = []

Output: []

Example 3

Input:

```
  1
 /
2
```

Output: [2,1]

4. Constraints

- 0 ≤ number of nodes ≤ 10^4
- Node values can repeat

- Must follow strict inorder L → N → R
- Recursive depth = height of tree

5. Core Concept (Pattern / Topic)

⭐ **DFS → Binary Tree Traversal (Inorder)**

6. Thought Process (Step-by-Step)

✨ **Approach 1: Recursive (easy & clean)**

- Visit left subtree
- Visit node
- Visit right subtree

✨ **Approach 2: Iterative with Stack (most common in interviews)**

We manually simulate recursion:

- Push all left nodes
- Pop → record value
- Move to right child

✨ **Approach 3: Morris Traversal (advanced, O(1) space)**

Make temporary threaded links (no stack, no recursion).

7. Visual / Intuition Diagram

Example Tree:

```
   1
 /  \
 2    3
```

Inorder Path:

Left → Node → Right

2 → 1 → 3

8. Pseudocode

```
result = []
stack = []
curr = root

while curr != null or stack not empty:
    while curr != null:
        stack.push(curr)
        curr = curr.left

    curr = stack.pop()
    result.append(curr.val)
    curr = curr.right

return result
```

9. Code Implementation

✅ Python

```
class Solution:
    def inorderTraversal(self, root):
        res, stack = [], []
        curr = root

        while curr or stack:
            while curr:
                stack.append(curr)
                curr = curr.left

            curr = stack.pop()
            res.append(curr.val)
            curr = curr.right

        return res
```

✅ Java

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        Stack<TreeNode> stack = new Stack<>();
        TreeNode curr = root;

        while (curr != null || !stack.isEmpty()) {
            while (curr != null) {
                stack.push(curr);
                curr = curr.left;
            }
```

```
                curr = stack.pop();
                res.add(curr.val);
                curr = curr.right;
            }

            return res;
        }
}
```

10. Time & Space Complexity

⏱ **Time: O(n)**

Each node visited exactly once.

🧠 **Space:**

- Recursion: O(h)
- Iteration using stack: O(h)
- Morris: **O(1)**

(h = tree height)

11. Common Mistakes / Edge Cases

❌ Returning preorder by mistake

❌ Forgetting to push left chain

❌ Not using while curr or stack

❌ Infinite loop: forgetting curr = curr.right

Edge cases:
✔ Empty tree
✔ Single node
✔ Left-only chain
✔ Right-only chain

12. Detailed Dry Run

Tree:

```
 1
  \
   2
  /
```

3

Step	curr	stack	result
push 1	1→2	[1]	[]
push 2	2→3	[1,2]	[]
push 3	3→null	[1,2,3]	[]
pop 3	null	[1,2]	[3]
pop 2	right=null	[1]	[3,2]
pop 1	right visited	[]	[3,2,1]

Final: [1,3,2]

13. Common Use Cases

- BST inorder → sorted output
- Syntax tree traversal
- Building expression evaluators
- Validating BST correctness

14. Common Traps

⚠ Pushing right child too early

⚠ Wrong order of appending

⚠ Stack pop before finishing left chain

⚠ Using recursion when depth might exceed limit

15. Builds To (Related Problems)

- LC 144 — Preorder
- LC 145 — Postorder
- LC 98 — Validate BST
- LC 230 — Kth Smallest in BST

16. Alternate Approaches + Comparison

Approach	Time	Space	Notes
Recursion	O(n)	O(h)	Cleanest
Stack	O(n)	O(h)	Most used in interviews
Morris	O(n)	O(1)	Advanced

17. Why This Solution Works

Inorder traversal always processes the left subtree, then the root, then the right subtree.
The stack replicates this exact recursive behavior.

18. Variations / Follow-Up

- Do inorder **without recursion**
- Do inorder with **O(1) space** (Morris)
- Inorder traversal of BST gives **sorted list**
- Find kth smallest using inorder