LeetCode 94 — Binary Tree Inorder Traversal

Link: 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)

You are given the root of a binary tree.

You must return the **inorder traversal** of the tree.

Inorder traversal rule:

➡ **Left → Node → Right**

3. Examples (Input → Output)

Example 1

Input:

```
  1
   \
    2
   /
  3
```

Output: [1,3,2]

Explanation:

Left subtree of 1 is empty → 1 → Go to right → left of 2 is 3 → 3 → 2

Example 2

Input: root = []

Output: []

Example 3

Input:

```
  1
 /
2
```

Output: [2,1]

4. Constraints

- Number of nodes ≤ 10^4

- Node values may be duplicates
- Traversal must follow **Left → Node → Right**
- Stack space = recursion depth O(h)

5. Core Concept (Pattern / Topic)

⭐ **Tree Traversal — DFS (Depth First Search)**

Variants: Preorder, Inorder, Postorder

6. Thought Process (Step-by-Step Explanation)

Brute Force?

There is no brute force; traversal logic is fixed.

Approach 1 — Recursion (Simple & Clean)

Use call stack:

1. Recurse left
2. Add current node
3. Recurse right

Approach 2 — Iterative Using Stack

Simulate recursion manually using a stack.

Steps:

- Push left nodes
- When no left, pop → add to result
- Move to right subtree

This avoids recursion limit.

Approach 3 — Morris Traversal (Advanced, O(1) Space)

Thread the tree temporarily.

Used for interviews with space constraint requirement.

7. Visual / Intuition Diagram (ASCII)

Tree:

```
   1
  / \
 2   3
```

Inorder Path:

Left → Node → Right

2 → 1 → 3

Iterative Process:

stack: push 1 → push 2

pop 2 → result=[2]

pop 1 → go right → push 3

pop 3 → result=[2,1,3]

8. Pseudocode (Language Independent)

```
function inorder(root):
    stack = []
    result = []
    curr = root

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

        curr = pop from stack
        add curr.val to result
        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

Iterative / Recursive:

- **Time:** O(n)
- **Space:** O(h)
    - h = tree height (O(n) worst, O(log n) if balanced)

Morris Traversal:

- **Time:** O(n)
- **Space: O(1)**

11. Common Mistakes / Edge Cases

❌ Forgetting to traverse right subtree

❌ Wrong order (Node → Left → Right → incorrect)

❌ Not checking curr or stack in while loop

❌ Infinite loop in iterative approach

❌ Not handling root = None

Edge cases:

- Empty tree → []
- Single node
- Only left chain
- Only right chain

12. Detailed Dry Run (Step-by-Step)

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	go right(null)	[1]	[3,2]
pop 1	go right(2 done)	[]	[3,2,1]

Final result: **[1,3,2]**

13. Common Use Cases (Real-Life / Interview)

- Validating BST property (inorder gives sorted array)
- Serializing / printing hierarchical structures
- Syntax tree evaluation
- Querying ordered dataset

14. Common Traps (Important!)

⚠ Wrong traversal order

⚠ Pushing right before left

⚠ Forgetting to move curr to curr.right

⚠ Using recursive solution when depth may exceed limits

15. Builds To (Related Problems)

- LC 144 (Preorder Traversal)
- LC 145 (Postorder Traversal)
- LC 230 (Kth Smallest in BST)
- LC 98 (Validate BST)

16. Alternate Approaches + Comparison

Method	Time	Space	Notes
Recursion	O(n)	O(h)	Simple
Iterative Stack	O(n)	O(h)	Most common
Morris	O(n)	O(1)	Advanced, interview favorite

17. Why This Solution Works (Short Intuition)

Inorder traversal simply processes nodes in the natural left→root→right order of a binary tree.
The stack solution simulates the call stack perfectly.

18. Variations / Follow-Up Questions

- Perform inorder **without recursion**? (Iterative)
- Perform inorder in **O(1) space**? (Morris)
- Use inorder to check if a tree is a BST
- Build BST from inorder + preorder