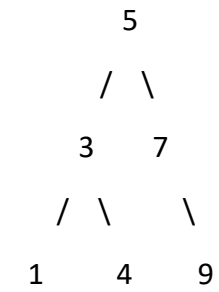
Inorder Traversal of Binary Tree LeetCode

In-order traversal: Left subtree, current node, right subtree

Example:



Output: **[1, 3, 4, 5, 7, 9]**

Approach 1: Perform an in-order traversal of the binary tree using recursion

- Define a recursive **solve** function to traverse the binary tree in the following order:
    - Recursively visit the left subtree.
    - Push the value of the current node.
    - Recursively visit the right subtree.
- In the **inOrderTraversalRecursive** function, call the **solve** function and store the results in a vector.
- **Time Complexity: O(N) as it visits each node exactly once.**
- **Space Complexity: O(N) for the function call stack and the vector.**

Approach 2: Perform an in-order traversal of the binary tree using an iterative approach

- Initialize an empty vector **ans** to store the traversal result and a stack **st** to help traverse the tree iteratively.
- While the current node **currNode** is not null or the stack is not empty:
    - Inside the first while loop, push the current node and move to its left child until **currNode** is null.
    - Process the top node on the stack:
        - Push the value of the current node to **ans**.
        - Move to the right subtree.
- In the **inOrderTraversalIterative** function, return the **ans** vector.

- **Time Complexity: O(N) as it visits each node exactly once.**
- **Space Complexity: O(H), where H is the height of the binary tree. In the worst case, where the tree is skewed, H could be N, making the space complexity O(N). In a balanced tree, it is O(log N).**

Approach 3: Morris Traversal Algorithm to perform an iterative inorder traversal of a binary tree

- Create an empty vector **ans** to store the traversal result.
- Start from the root node as **currNode**.
- While **currNode** is not null:
    - If the current node has no left child, visit it and move to its right child.
    - If the current node has a left child, find its in-order predecessor:
        - Initialize **predecessor** to the left child.
        - Traverse to the rightmost node of the left subtree if not visited already.
        - If the predecessor's right child is not assigned, assign it to the current node, and move to the left child.
        - If the predecessor's right child is already assigned, visit the current node and then move to its right child.
- Return the **ans** vector as the traversal result.
- **Time Complexity: O(N) as it visits each node exactly once.**
- **Space Complexity: O(1) as it doesn't use additional data structures except for the ans vector.**

Conclusion:

- All three approaches successfully perform an in-order traversal of the binary tree and return the results in the same order.
- The recursive, iterative, and Morris traversal methods all yield the expected traversal sequence.
- The Morris traversal approach offers the advantage of a space complexity of O(1), which makes it a memory-efficient option for in-order tree traversal.