Summary

Time Complexity: O(n), where n is the number of nodes in the tree. This is because every node is visited once.
Space Complexity: O(n), space needed for the stack in the worst case (unbalanced tree). In the average case it is O(logn).

Problem Pattern

1. In inorder traversal method, the left subtree is visited first, then the root and later the right subtree. We should always remember that every node may represent a subtree itself.
2. Usually we perform inorder traversal using recursion, which is trivial. How will you do this iteratively? Is there some data structure used by recursion implicitly? Yes, stack.
3. So, for an iterative inorder traversal, we can explicitly create our own stack and imitate what the recursive function does.

Problem Approach

1.The in order traversal requires that we print the leftmost node first and the rightmost node at the end. So basically for each node we need to go as far as down and left as possible and then we need to come back up and go right.

2. So we create an empty stack S & initialize current node as root.
3. We push the current node to S and set current = current->left until current is NULL i.e keep going down and left.
4. If current is NULL and stack is not empty then

a) We pop the top item from stack.
b) Store the popped item in the result list, and go to the right of the popped item.
c) We will repeat step 3 again (which is basically what recursion does)

5. How will we know when we are done? If current is NULL and stack is empty.

```
Problem Pseudocode

List <int> inorderTraversal(TreeNode root) {
        res = new List()
        Stack <TreeNode> stack
        TreeNode curr = root
        while (curr != null or stack is not Empty) {
            while (curr != null) {
                stack.push(curr)
                curr = curr.left
            }
            curr = stack.pop
            res.add(curr.val)
            curr = curr.right
        }
        return res
}
```

Alternate Approaches

A. You can solve this recursively

1. Traverse the left subtree, i.e., call Inorder(left-subtree)
2. Visit the root.
3. Traverse the right subtree, i.e., call Inorder(right-subtree)

B. Using Morris Traversal, we can traverse the tree without using stack and recursion. The idea of Morris Traversal is based on Threaded Binary Trees. In this traversal, we first create links to Inorder successor and print the data using these links, and finally revert the changes to restore the original tree. Although the tree is modified through the traversal, it is reverted back to its original shape after the completion. Unlike Stack based traversal, no extra space is required for this traversal.

1. Initialize current as root

2. While current is not NULL

    If the current does not have left child

        a) Print current’s data

        b) Go to the right, i.e., current = current->right

    Else

        a) Make current as the right child of the rightmost node in current's left subtree

        b) Go to this left child, i.e., current = current->left