PostOrder Traversal of Binary Tree LeetCode

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

Example:

Example:

```
     5
    / \
   3    7
  /  \    \
11   1     6
```

Output: **[11, 1, 3, 6, 7, 5]**

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

- In this approach, we use a recursive function **solve** to perform post-order traversal.
- The function first recursively visits the left subtree, then the right subtree, and finally pushes the value of the current node.
- The **postOrderTraversalRecursively** function initializes an empty vector **ans**, calls **solve**, and returns the **ans** vector as the result.

Time Complexity: O(N), where N is the number of nodes in the binary tree. We visit each node once.

Space Complexity: O(N) in the worst case due to the function call stack.

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

- In this approach, we use an iterative algorithm to perform post-order traversal.
- We utilize a stack to simulate the recursive process.
- We maintain two pointers, **currNode** to track the current node and **lastVisited** to remember the last visited node.
- We traverse the tree until **currNode** becomes null and use a loop to handle the stack operations.
- We push nodes onto the stack while moving to the left child. When we encounter a node with a right child, we check if it has already been visited. If not, we move to its right child; otherwise, we process the current node and pop it from the stack.

- We store the results in the **ans** vector.

Time Complexity: O(N), where N is the number of nodes in the binary tree. We visit each node once.

Space Complexity: O(H), where H is the height of the binary tree. In the worst case, when the tree is skewed, the stack can have at most H nodes.