PreOrder Traversal of Binary Tree LeetCode

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

Example:

```
          5
        /   \
      3       7
    /   \       \
  1       4       9
```

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

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

- Define a recursive solve function to traverse the binary tree in the following order:
    - Push the value of the current node.
    - Recursively visit the left subtree.
    - Recursively visit the right subtree.
- In the preOrderTraversalRecursively 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 pre-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 value of the current node.
        - If the current node has a right child, push it onto the stack.
        - Move to the left child.
    - In the outer while loop, pop a node from the stack to process its right subtree.

- In the **preOrderTraversalIteratively** 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 Preorder 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, visit the current node, push the value to **ans**, and assign it to the predecessor's right child. Finally, move to the left child.
        - If the predecessor's right child is already assigned, reset it to nullptr and move to the right child of the current node.
- 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 a pre-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), making it a memory-efficient option for pre-order tree traversal.