Search In BST LeetCode

You are given the root of a binary search tree (BST) and an integer val.

Find the node in the BST that the node's value equals val and return the subtree rooted with that node. If such a node does not exist, return null.

Example:

```
              10
           /     \
         7        17
       /        /    \
      5        15      19
   /          /
 1          13
```

SearchNode = 13
Output: The Node 13 is present in BST.

Approach 1: Recursive function to search for a value in the BST

1. **Recursive Function:** In this approach, a recursive function **searchBSTRecursive** is used to search for a given value within the BST.
2. **Base Cases:** The function starts with the root of the tree. It checks if the current node is **nullptr** (indicating an empty tree) or if the current node's value matches the target value. There are two base cases:
    - If the tree is empty (**root** is **nullptr**), it returns **nullptr** as the value is not found.
    - If the current node's value matches the target value, it returns the current node.
3. **Recursion:** If the target value is not found in the current node, the function makes a recursive call on the left subtree if the target value is smaller than the current node's value, or on the right subtree if the target value is greater.
4. **Time Complexity: The time complexity of this recursive approach is O(H), where H is the height of the BST. In the worst case, when the tree is skewed (a degenerate tree), H can be N (the number of nodes in the tree).**
5. **Space Complexity: The space complexity is also O(H), as the function's call stack can grow to a depth of H in the worst case.**

Approach 2: Iterative function to search for a value in the BST

1. **Iterative Function:** In this approach, an iterative function **searchBSTIterative** is used to search for a given value within the BST.
2. **Initialization:** It starts with the root of the tree and initializes a current node (**currNode**) to the root.
3. **Iterative Loop:** It enters a while loop that continues until **currNode** becomes **nullptr**. Within the loop, it checks if the current node's value matches the target value. There are two possibilities:
    - If the current node's value matches the target value, it returns the current node, indicating a successful search.
    - If the target value is smaller than the current node's value, it updates **currNode** to the left child. If it's greater, it updates **currNode** to the right child.
4. **Time Complexity: The time complexity of this iterative approach is also O(H), where H is the height of the BST. It traverses the tree vertically, making comparisons at each level.**
5. **Space Complexity: The space complexity is O(1) because it uses a constant amount of additional space, regardless of the height of the tree.**

Conclusion:

- Both approaches are effective for searching in a BST.
- The recursive approach is simple to understand and implement but consumes memory proportional to the tree's height.
- The iterative approach is more memory-efficient, using a constant amount of space. It is recommended if memory efficiency is a priority.
- Both approaches have the same time complexity for the search operation, making them equally efficient in terms of time.