Find Pair of Given Sum in Binary Search Tree LeetCode

Given the root of a binary search tree and an integer k, return true *if there exist two elements in the BST such that their sum is equal to* k, *or* false *otherwise*.

Example:

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

TargetNode: 32

Output: The Pair exists in BST with target of: 32 [(17, 15), (19, 13)]

Approach 1: Search for a node pair with values summing to 'k' in a binary search tree (Brute Force).

- **Function Purpose**: Find a pair of nodes in a binary search tree (BST) with values summing to a given target 'k'.
- **Explanation**:
    - Recursively check all possible pairs of nodes in the BST.
    - For each pair of nodes, calculate their sum and check if it equals 'k'.
- **Time Complexity: O(N^2) in the worst case, where N is the number of nodes in the BST.**
- **Space Complexity: O(H), where H is the height of the BST.**

Approach 2: Find a node pair with values summing to 'k' in a binary search tree using in-order traversal

- **Function Purpose**: Find a pair of nodes in a binary search tree (BST) with values summing to a given target 'k'.
- **Explanation**:

    - Perform an in-order traversal of the BST using Morris Traversal to store values.
    - Maintain two pointers, 'start' and 'end,' to find the pair.
    - If the sum of values pointed to by 'start' and 'end' equals 'k,' return true.
    - If the sum is less than 'k,' move 'start' to the right; if greater, move 'end' to the left.
- **Time Complexity: O(N), where N is the number of nodes in the BST.**
- **Space Complexity: O(N) to store the in-order traversal result.**

Conclusion:

- **The Optimized In-order Traversal Approach (Approach 2) is better than the Brute Force Approach (Approach 1) as it has a more efficient time complexity of O(N) compared to O(N^2),** making it more suitable for larger trees. It also uses less space for traversal.