**Code Reflection for Binary Search Tree**

By: Darrell Walker

The purpose of this code is to implement a Binary Search Tree (BST) to efficiently manage bids data loaded from a CSV file. The BST allows for efficient insertion, deletion, and search operations, ensuring that these operations can be performed in logarithmic time on average. The primary techniques used include recursive methods for inserting new nodes, deleting existing nodes, and traversing the tree in various orders (in-order, pre-order, and post-order). Each node in the tree contains a bid, and the nodes are organized such that for any given node, all nodes in its left subtree have smaller bid IDs, and all nodes in its right subtree have larger bid IDs.

One significant challenge encountered during the implementation was ensuring that the tree structure remains correct after deletions, especially when the node to be deleted has two children. This requires finding the in-order successor (the smallest node in the right subtree) to replace the deleted node while maintaining the BST properties. To address this, careful handling of node pointers and recursive calls was necessary to correctly adjust the tree structure.

The code is structured to be modular and reusable, with each function performing a single, well-defined task. For example, the addNode method handles the recursive insertion of nodes, while the removeNode method manages node deletion. Additionally, traversal methods like inOrderTraversal, preOrderTraversal, and postOrderTraversal are implemented to allow for different ways of visiting all nodes in the tree. The code is thoroughly commented to explain the purpose and functionality of each section, enhancing readability and maintainability.

**Pseudocode**

The following pseudocode outlines the core functionality of the Binary Search Tree implementation:

1. **Class BinarySearchTree**:
   * **Method init**:
     + Initialize root to None.
   * **Method insert(bid)**:
     + If root is None, set root to a new Node with the bid.
     + Else, call addNode with root and bid.
   * **Method remove(bidId)**:
     + Call removeNode with root and bidId.
   * **Method search(bidId)**:
     + Return the result of searchNode with root and bidId.
   * **Method inOrder()**:
     + Call inOrderTraversal with root.
   * **Method preOrder()**:
     + Call preOrderTraversal with root.
   * **Method postOrder()**:
     + Call postOrderTraversal with root.
2. **Method addNode(node, bid)**:
   * If bid.bidId < node.bid.bidId:
     + If node.left is None, set node.left to a new Node with bid.
     + Else, call addNode with node.left and bid.
   * Else:
     + If node.right is None, set node.right to a new Node with bid.
     + Else, call addNode with node.right and bid.
3. **Method removeNode(node, bidId)**:
   * If node is None, return node.
   * If bidId < node.bid.bidId, set node.left to the result of removeNode with node.left and bidId.
   * Else if bidId > node.bid.bidId, set node.right to the result of removeNode with node.right and bidId.
   * Else (node to be deleted found):
     + If node.left is None, set node to node.right.
     + Else if node.right is None, set node to node.left.
     + Else:
       - Set temp to the result of findMin(node.right).
       - Set node.bid to temp.bid.
       - Set node.right to the result of removeNode with node.right and temp.bid.bidId.
4. **Method searchNode(node, bidId)**:
   * If node is None or node.bid.bidId equals bidId, return node.
   * Else if bidId < node.bid.bidId, return searchNode with node.left and bidId.
   * Else, return searchNode with node.right and bidId.
5. **Method findMin(node)**:
   * While node.left is not None, set node to node.left.
   * Return node.
6. **Method inOrderTraversal(node)**:
   * If node is not None:
     + Call inOrderTraversal with node.left.
     + Print node.bid.
     + Call inOrderTraversal with node.right.
7. **Method preOrderTraversal(node)**:
   * If node is not None:
     + Print node.bid.
     + Call preOrderTraversal with node.left.
     + Call preOrderTraversal with node.right.
8. **Method postOrderTraversal(node)**:
   * If node is not None:
     + Call postOrderTraversal with node.left.
     + Call postOrderTraversal with node.right.
     + Print node.bid.
9. **Method destroyNode(node)**:
   * If node is not None:
     + Call destroyNode with node.left.
     + Call destroyNode with node.right.
     + Delete node.