CSC 202 Data Structures Lab 3:

Objective: In this lab activity, you will implement and explore more advanced operations and concepts related to Binary Search Trees (BST).

Prerequisites:

- Basic understanding of Binary Search Trees and their basic operations (insertion, deletion, search, and traversal).

Tasks:

1. Implement the **isBST()** method to check if the given binary tree is a valid Binary Search Tree.
2. Implement the **convertToSortedArray()** method to convert the BST into a sorted array.
3. Implement the **lowestCommonAncestor()** method to find the lowest common ancestor of two nodes in the BST. Lowest common ancestor is the ancestor you find while going down in the depth of a tree or the levels down from root node.
4. Implement the **deleteTree()** method to delete the entire BST. Delete nodes one by one from leaf nodes to root.

Guidelines:

- Create a class named **TreeNode** to represent each node in the BST. The class should have attributes for the node's value, left child, and right child.
- Create a class named **BST** to represent the Binary Search Tree. Implement all the required methods within this class.

Instructions:

1. Implement the **TreeNode** class.
2. Implement the **BST** class with the methods listed above.
3. Test each method with various test cases to ensure their correctness.
4. For the **isBST()** method, create both valid and invalid Binary Search Trees to verify the results.
5. For the **lowestCommonAncestor()** method, create test cases with different scenarios (nodes at different levels, one node being an ancestor of the other, etc.).
6. Ensure that your implementation handles edge cases appropriately.

Note: You can use my code from my canvas page under the tab course material → bst_insertion_search_deletion.txt as a starter code.