**CSC 364 Assignment # 4**

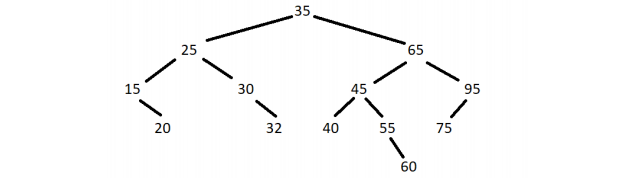
**Total Points: 40**

**BST Reconstruction**

Consider a Binary Search Tree that is generated by inserting the following integers in the given

sequence.

Insert into BST: 35, 25, 65, 30, 15, 20, 95, 45, 40, 55, 32, 60, 75



**In-order traversal of the above tree:** 15, 20, 25, 30, 32, 35, 40, 45, 55, 60, 65, 75, 90

**Pre-order traversal of the above tree:** 35, 25, 15, 20, 30, 32, 65, 45, 40, 55, 60, 95, 75

**Post-order traversal of the above tree:** 20, 15, 32, 30, 25, 40, 60, 55, 45, 75, 95, 65, 35

When we save a BST into a file, we can consider any of the above traversals to save the BST.

However, to reconstruct the BST into the exact same tree, we either need to save the BST in

pre-order or post-order.

Below is the algorithm to reconstruct the BST with a given pre-order traversal.

1. The first element in the current input array is the root.

2. Loop from the second element till the end.

3. Stop when the element is greater than the root.

4. Split the array into two sub-arrays, which implies the elements in the left and right

sub-trees.

5. Recursively start from step 1 to reconstruct the left and right subtrees from the two

sub-arrays.

**Example:**

Pre-Order traversal input: 35, 25, 15, 20, 30, 32, 65, 45, 40, 55, 60, 95, 75

Root: 35

Left-sub-array: 25, 15, 20, 30, 32

Right-sub-array: 65, 45, 40, 55, 60, 95, 75

Recursively repeat for left and right sub-arrays.

The opposite logic can be applied to a post-order traversal to reconstruct the BST. Below is the

algorithm to reconstruct the BST with a given post-order traversal.

1. The last element N in the current input array is the root.

2. Loop from beginning till N-1.

3. Stop when the element is greater than the root.

4. Split the array into two sub-arrays, which implies the elements in the left and right

sub-trees.

5. Recursively start from step 1 to reconstruct the left and right subtrees from the two

sub-arrays.

**Example:**

Post-Order traversal input: 20, 15, 32, 30, 25, 40, 60, 55, 45, 75, 95, 65, 35

Root: 35

Left-sub-array: 20, 15, 32, 30, 25

Right-sub-array: 40, 60, 55, 45, 75, 95, 65

Recursively repeat for left and right sub-arrays.

**Tasks:**

1. Implement the following methods in the **BST** class

**a. isEqualTo(BST<?> otherTree)**

*// This method should be called to check if a BST is equal to another BST.*

*Returns Boolean true/false.*

**b. isEqualTo(TreeNode<?> root1, TreeNode<?> root2)**

*// This is a private helper method to implement the recursive checking of*

*the two BSTs*

*// Recursively check both the trees by traversing both the trees at the same*

*time. The type of traversal is flexible, as long as all the nodes are visited.*

*Return true if recursion is complete. Return false if any of the elements is*

*a mismatch during the traversal.*

2. Implement the following methods in the **BSTReconstructor** class

**a. preOrderReconstructor(List<Integer> inputArray)**

*// This method will take an inputArray of pre-order traversal items and*

*create the original BST, and save reconstructed tree in the*

*preOrderReconstructedBST variable.*

**b. postOrderReconstructor(List<Integer> inputArray)**

*// This method will take an inputArray of post-order traversal items and*

*create the original BST, and save the reconstructed tree in the*

*postOrderReconstructedBST variable.*

**Execution:** Run the main() method in the BSTReconstructor class to validate your

implementation.

**Submission:** Submit the BST.java and BSTReconstructor.java files.