In-Order BST Traversal (3 methods)
Cpt S 321 Homework Assignment
Washington State University

Submission Instructions: (see syllabus)

Assignment Instructions:

Read ALL the instructions *carefully* before you write any code.

Create a console application that builds and displays contents of a binary search tree. You may use integers as the data values in the nodes (consider using BitInteger instead if you want). Build the tree with 20 to 30 random values, each between 0 and 100. Implement three different algorithms to display the contents of the tree in order and demo each of them after the tree is constructed:

1. In-order traversal using 2 recursive calls. This should be trivial and is mainly included in this assignment to help you verify accuracy of parts 2 and 3.

2. In-order traversal without using recursion and instead using a single Stack<T> data structure. You should only need 1 stack to do this and cannot modify tree contents in any way during the process.

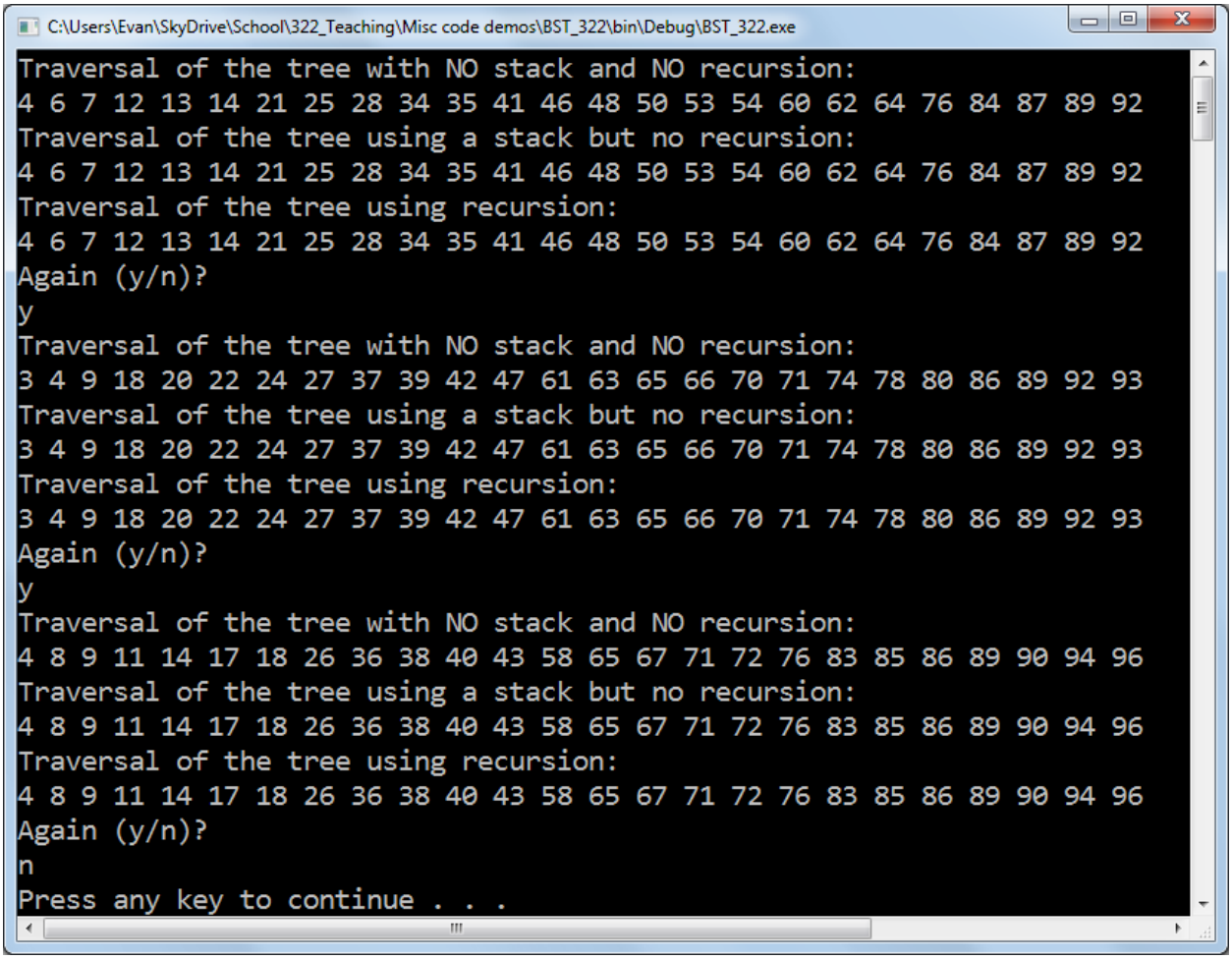
3. In-order traversal without using recursion and without using a stack. This method must have constant space complexity. You should never need more than a couple node references to implement this method. This method can temporarily alter the tree, and in fact that's required to make it work, but it must have it back to its original form after the traversal is complete.

- Do this without removing nodes entirely from the tree and then adding them back. You must leave all nodes in their original starting place in the tree at all times.
- Think about what aspects of tree you can alter/utilize if you can't remove items from the tree. These requirements should narrow it down to the one thing that you *are* able to do in order to implement the traversal algorithm.

All methods must run in worst case O(n) time!!!!

Display the results from part 3 first, then part 2 then part 1. This is to ensure that the tree contents are preserved by the operations in parts 2 and 3. Do NOT rebuild the tree between traversals. Execute all 3 in sequence.

Provide the option for the user to have another tree generated and traversed after the 3 traversal outcomes are displayed. See the demo image below as an example of this, where 3 different trees were built and each had 3 different traversal methods execute on it.



The grader (TA or instructor depending on the semester) will examine your code to ensure that each method is obeying the requirements. Any recursive calls in parts 2 or 3 will render those implementations worth 0 points. Incorrect output produced by any algorithm also renders it worth 0 points.