Binary Search Tree pt. 1

A binary search tree (BST) is a simple graph-based data structure that allows for incredibly fast add, remove, and search functionality in the average case. The best way to gain a better understanding of this data structure is to build one of your own!

**Requirements**

You will build a BST that meets the following requirements:

* Create a new project and create your BST class. The class MUST be named BinarySearchTree.
* The tree is composed of nodes (like a Linked List).
* The tree and its nodes are implemented generically to hold any comparable data types.
* The tree has the following functions:
* Add(value) - adds a new value to the tree, following the rules of a BST. Duplicate values go to the right of the parent.
* Contains(value) - returns true if the specified value is in the tree.
* Remove(value) - removes the specified value from the tree if it is present, or does nothing if the value is not present. If the value appears more than once, only the first occurrence of the value is removed. There is no need to return anything.
* Clear( ) - removes all values from the tree.
* Count - returns the number of values in the tree. This must be a property in C# (with a public get and a private/protected set) or a get/set method in Java.
* InOrder( ) - returns an in-order string representation of all the values in the BST
* PreOrder( ) - returns a pre-order string representation of all the values in the BST
* PostOrder() – returns a post-order string representation of all the values in the BST
* All of the “order” functions above return a string in the format of “v1, v2, …, vn”.
* The structure of this string is very important. Make sure you have a comma and space between each value and no trailing comma or spaces.
* Height - returns the height of the BST, where the empty tree is 0, the simple tree is 1.
* ToArray( ) – returns an Array representation of the values using in-order traversal. If you’re working in Java, make sure to take into account Java’s type erasure of generics. It will require a little more work on your part to meet this requirement, but it’s still doable.
* If using java
* Comparable[] result = (Comparable[])Array.newInstance(Comparable.class, count());

You are REQUIRED to follow the signatures as outlined above. This means that you must name the functions and order the parameter lists as I have. With that said, it is expected that you will follow the proper convention for your coding language. The functions are named as they would be in C#, but in Java, each name would begin with a lowercase letter (inOrder, toArray, clear, etc…) Do not deviate from these conventions.

**Rubric**

**Automatic Zero:** Your data structure is not a binary search tree, you deviate from the prescribed function signatures, or your final product is not a library (JAR or DLL).

(50 points) Your BST is implemented correctly and the functions work correctly (points for each passing test)

A Helpful BST Visualizer can be found at <https://www.cs.usfca.edu/~galles/visualization/BST.html>



* **Tree Traversal**
* In-Order (Left, Root, Right)
* 3, 7, 8, 10, 11, 12
* Pre-Order (Root, Left, Right)
* 10, 7, 3, 8, 12, 11
* Post-Order (Left, Right, Root)
* 3, 8, 7, 11, 12, 10