**Assignment 8**

Due on **April 21st, at the end of the day**. Please follow the submission instructions in the "notes for all labs" on Moodle.

- Read the "Notes for All Labs" document on Moodle. All assignments must be submitted as specified there. Note that this lab has no written component.
- Your code should be in a file, called your_id_330_lab8.hs, where your_id is your Earlham user ID (dmbarbe16, or whatever); this should contain the Coding Assignments
- Any additional files you use should use a similar naming convention.

Several of these are trickier than they might first appear. You may want to plan a design first.

**Coding Assignment 8.1: (2 Points)**
Create a data structure for a Binary Tree, called `MyBinaryTree`. You may use the implementation from the text/slides. Your tree should be a polymorphic data structure that can hold any type of data. Your null nodes that the leaves have as children should be called `NullNode`. The regular nodes should be called Node. (Naming things as specified will make grading this assignment much faster.) It should derive Show and Eq, at a minimum.

**Coding Assignment 8.2: (2 Points)**
Define a pair of functions, `leftTree` and `rightTree`, that return the left and right children of the root of your `MyBinaryTree`. They should return a NullNode if there is no child.

**Coding Assignment 8.3: (4 Points)**
Define a function `treeElem` that takes two arguments: an `a` and a `MyBinaryTree a`, in that order. It returns true if the `a` is in the tree.
Note that this function (and others like it) only need work if the tree holds a type in the typeclass `Eq`.

**Coding Assignment 8.4: (4 Points)**
Define a pair of functions `treeMax` and `treeMin`. They should find the largest and smallest elements in a `MyBinaryTree`. Your type signature for these functions should specify that they only work if the tree holds a type in the typeclass `Ord`.

**Coding Assignment 8.5a: (4 Points)**
A tree is *reflected* by swapping its left and right subtrees, recursively. Create a function `reflectTree` that does that.

**Coding Assignment 8.5b: (4 Points)**
Create a function `collapseTree` that returns an `[a]`. It should return an in-order traversal of the elements of the tree, as a list.

**Coding Assignment 8.6: (4 Points)**
A *binary search tree* has the property that, for all of its nodes, the left child of a node (if one exists) contains a smaller value than that node, and the right child contains a larger value. Write a function `isBST` that returns True if a given binary tree is a search tree, and False otherwise.

---

**Coding Assignment 8.7: (4 Points)**
Write a function `bstAdd` that takes a value and a tree, in that order. It “adds” the node to the binary tree as though it was a binary search tree, and returns the new tree. (Note that if the given tree is not a binary search tree, this operation will not make it one.) If the given tree is just a NullNode, this function should create a tree with one node. (We will learn a better way to handle making sure a binary search tree stays that way later in the class.)

---

**Coding Assignment 8.8: (6 Points)**
Write a function `listToBST` that takes a list and makes a BST containing its elements. Write a function `treeToBST` that takes an arbitrary tree and makes a BST containing its elements. Hint: Your definition of `treeToBST` can be very short if you use function composition.

---

**Coding Assignment 8.9: (6 Points)**
Write a function `delFromBST` that takes a value and a binary tree, in that order, and deletes the value from the binary tree as though it were a binary search tree. Your function is allowed to fail if the given tree is not a binary search tree. If the value is not found in the tree, your function should just return the same tree it was given.
Hint: This one is a touch trickier than most.
**Note to Future Dave: Cut this one in the future**

---

**Coding Assignment 8.10: (4 Points)**
Write a function `binaryLookup` that takes a value and a binary search tree, in that order, and returns True if the value is in the tree and False otherwise. Your function should be more efficient than the treeElem function you wrote above.

---