M9: Pre Lab

Introduction

This pre-lab is designed to help prepare you to complete the programming component of M9's Lab.

Because the pre-lab section is designed to help you plan your implementation, it is strongly recommended for you to complete this section **before** writing any code. However, it is also natural for your understanding of the implementation requirements and your own strategy to evolve as you complete the code, so feel free to update your answers to the pre-lab section during/after the coding phase of the lab, up to the pre-lab deadline.

Questions

The programming component of this lab consists of extending an unbalanced binary tree to enforce balance using the AVL property. You'll be filling in the method stubs in the file `AVLTree.java`. For each of the methods listed to be filled in (listed below), answer the following questions.

- If the method has a parameter (i.e. an input passed to the method), what does that parameter represent, in your own words? How will the parameter be used by the method? (An example of a parameter is that the method `insert` has one parameter `thing` of type `T`.)
- If the method has a non-`void` return value, what does it return, in your own words?
- If the method has no parameters and/or its return value is void, please state this fact in your answer.

Here are the methods you will be filling in:

1. `private void updateHeight(TreeNode<T> root);`
2. `private int getBalance(TreeNode<T> root);`
3. `private TreeNode<T> rebalance(TreeNode<T> root);`
4. `private TreeNode<T> rightRotate(TreeNode<T> root);`

(I won't ask you to repeat yourself by giving the same info for `leftRotate()`.)
Here is a sample of an expected response for a method called `selectionSort` (not related to this assignment) that has the header

```
public int[] selectionSort(int[] arr);
```

and whose goal is to perform a selection sort on an array of ints.

Method: `public int[] selectionSort(int[] arr);`

- *Parameters*:
  `arr`: a 1-dimensional array of ints that stores the inputs to be sorted. The method will read this input array (multiple times) and write a sorted version of it to a newly allocated output array.
- *Return value*: a new array containing a sorted version of the input.

Once you finish your responses for the methods listed, answer the following additional questions:

5. What is the formula for computing the height of a tree's root node, assuming the heights for its left and right subtrees are known?
6. What should the height be for a newly allocated leaf node? For an empty subtree?
7. Why do the `insert()` and `remove()` methods declare recursive helper functions, rather than themselves being recursive?
8. Here is one line from the `insertHelper()` method:

   ```
   root.setLeft(insertHelper(value, root.left));
   ```

   Explain why the root's child pointer must be updated with the result of the call to `insertHelper()`.