5SENG003W Algorithms
Week 7
Tutorial Exercises: Tree Properties and Balanced Binary Search Trees

These exercises cover: tree traversal, tree properties and AVL trees.

Before you attempt these exercises, make sure you have completed the previous tutorial and understand the solution, we will be re-using and modifying the Binary Tree Java code.

Exercise 1.

Modify the existing `BinarySearchTree.java` class, by defining the constant arrays:

```
final int[]  treeA = { 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55 } ;
final int[]  treeB = { 55, 50, 45, 40, 35, 30, 25, 20, 15, 10, 5 } ;
final int[]  treeC = { 30, 25, 20, 15, 10, 5, 35, 40, 45, 50, 55 } ;
final int[]  treeD = { 30, 15, 45, 10, 40, 20, 50, 5, 35, 25, 55 } ;
```

These will be used for testing purposes, instead of using a randomly generated array of numbers.

(a) Use the test arrays to create and print the corresponding BST using the methods in `BinarySearchTree.java`.

(b) Draw the BST tree represented by each test data array.

Exercise 2.

Check your BST diagrams are correct for your test data arrays by inputting the numbers from each array into David Galles BST visualization tool at:

https://www.cs.usfca.edu/~galles/visualization/BST.html

It is recommended that you use a separate browser tab for each of your trees.

If your diagrams do not match the ones produced in the tool, make sure you understand what your mistake is before proceeding with the following exercises.

It may help to slow the animation right down so that you can follow the steps. You should also take screen shots of your built trees and/or record the building of at least one tree.

Exercise 3.

Extend the BinaryTree class with methods implementing the three traversal algorithms given in week 5:

(a) in-order,
(b) pre-order,
(c) post-order,

where to process a node, we just print out the data (number) stored in the node on a separate line.

Test them on the BSTs created by the test data arrays to check that they traverse the trees in the correct order. Compare the output with your tree.

Exercise 4.

(a) Extend the TreeNode class by adding members **height** and **balance** (both integers).

(b) Based on one of your traversal algorithms, define a method that calculates and stores the height and balance factor of each node in a tree.

(c) Check the results of this method for the trees obtained from the test arrays. To do this, extend one of the traversal functions to print the height and balance factor along with the node's data.

Exercise 5 (Challenge).

Construct the AVL trees for the 4 test trees from **Exercise 5** into David Galles' AVL tree visualization tool at:

https://www.cs.usfca.edu/~galles/visualization/AVLtree.html

It is recommended that you use a separate browser tab for each of the 4 trees.

Then see if you can duplicate the results:

First, implement the re-balancing rotations.

Then, in order for them to work, you also need to figure out after each modification which nodes have become unbalanced. We will only be doing this for insertions for now, using the existing recursive structure of the insertBelow method.

At the end of insertBelow,

- re-compute the node's height and balance factor;
- then, if the node is unbalanced, choose and execute the appropriate rotation.

You may want to add output statements to make sure you know what is going on.