```
CS 6301.002.  Implementation of advanced data structures and algorithms
Spring 2016
Short Project 4 (Trees)
Wed, Mar 9, 2016

Ver 1.0: Initial description (Mar 9, 9:00 AM).

Due: 1:00 PM, Thu, Mar 17.
```

Solve at least one problem from the following list. First solution will be graded out of 10. Each additional problem will be considered for 1 extra point.

a. Level order

Write the following method for the BST class:

```
Comparable[] levelOrderArray() { ... }
// Return an array with the elements using a level order traversal of the tree
```

b. Balanced binary search tree from a sorted array

Write another constructor for the BST class:

```
// Build a balanced BST using elements of sorted array.
BST(T[] arr) { ... }

// Optional challenge problem: Do the same for a sorted list,
// without using extra space to store the elements temporarily.
// Build a balanced BST using elements of a sorted list.
BST(List<T> lst) { ... }
```

c. Modifying remove in BST class

The BST class implements remove by moving the minimum element in the right subtree to replace the removed node (2-child case). It could have been written to replace the removed node by the maximum element in the left subtree. Rewrite remove so that it alternates between these two possibilities.

d. Correcting poorly written code on trees

Consider Tree.java (http://www.utdallas.edu/~rbk/teach/2016s/java/code/Tree.java). It calculates some function at every node of a tree that depends on the depth and height of that node, and then outputs the min. Find out why the code takes so long for larger values of n and correct it so that it runs fast, even for n=1,000,000. You can add fields to any of the classes, as needed. Score of -∞ will be given for solutions that use hashing.

e. Implement add() in Red-Black trees

Create an RBTree class for Red-Black trees. You can make it an inherited class of BST or take the BST code and modify it as needed. Implement the add function of Red-Black trees.

f. AVL tree verification

Given an AVL tree, write a function to verify whether it is a valid AVL tree. The tree may violate AVL tree conditions in any of the following ways: ordering condition, null value stored as element, balance condition.

```
boolean verifyAVLTree() { ... }
// A member function in the AVLTree class.
```