Project 3: Balanced Binary Search Tree

Instructions: In our labs we have used unbalanced Binary Search Trees. In this project:

1. Pick two balanced binary search tree implementations (AVL, Red-Black, ScapeGoat, 2-3, AA, Splay, Treap, etc) and implement our interface.
2. Next research what situations each perform better or worse in. Namely:
   (a) Tell me the situation one exhibits better performance.
   (b) Tell me why you believe that is the case.
3. Then generate test cases show one performing better than the other. ***Generate a performance graph.***

You may add functions to our Binary Tree implementation!

Internet: You may make full use of the internet for this project, except for the generation of the performance graphs.

Warning: I have been informed that online versions of Red-Black trees are faulty!

```
#ifndef BINARY_TREE_H
#define BINARY_TREE_H

#include <string>

template<class T>
class BinaryTreeNode {
    public:
        BinaryTreeNode<T> () {
        }
};

template<class T>
class BinaryTree {
    private:
        /* You fill in private member data. */

    public:

        /* Creates an empty binary tree. */
        BinaryTree();

        /* Does a deep copy of the tree. */
        BinaryTree(const BinaryTree<T> &tree);

        /* Add a given value to the Binary Tree.
```

```
         * Must maintain ordering!
         */
        void put(const T &val);

        /* Returns the height of the binary tree. */
        int getHeight();

        /* Returns true if an item exists in the Binary Tree */
        bool contains(const T &val) const;

        /* Removes a specific val from the Binary Tree.
         * Returns true if the value exists (and was removed.)
         * Otherwise, returns false.
         */
        bool remove(const T &val);

        /* Returns a string representation of the binary Tree in order. */
        std::string inorderString();

        /* Returns a string representation of the binary Tree in order. */
        std::string preorderString();

        /* Returns a string representation of the binary Tree in order. */
        std::string postorderString();

        /* Always free memory. */
        ~BinaryTree();
};

/* Since BinaryTree is templated, we include the .cpp.
 * Templated classes are not implemented until utilized (or explicitly
 * declared.)
 */
#include "binarytree.cpp"

#endif
```

Additional Resources:
`http://opendatastructures.org/versions/edition-0.1c/ods-cpp.pdf`

Analysis Report:
You shall describe one situation were each balanced tree algorithm is better than the other (2 in total.) These metrics may be insertion time, retrieval time, deletion time, memory usage or any other functional test. As with most metrics a line graph (with size of tree on the x axis/time on y axis) shall depict the difference clearly.

Write some test cases:

Create some test cases, using cxxtestgen, that you believe would cover all aspects of your code.

STL:
You may utilize anything from the STL that does not implement a balanced Binary Tree. (Example: Do not use map)

Memory Management:
Now that we are using new, we must ensure that there is a corresponding delete to free the memory. Ensure there are no memory leaks in your code! Please run Valgrind on your tests to ensure no memory leaks!

How to turn in:
Turn in via GitHub. Ensure the file(s) are in your directory and then:

- $ git add <files>
- $ git commit
- $ git push

Due Date: April 1, 2019 2359 (no joke.)

Teamwork: No teamwork, your work must be your own. Please cite what resource you used for your balanced binary tree and research. (Yes, you may use the internet!)