# CS590 homework 4 – Binary-Search, and Red-black Trees

The due date for this assignment is Friday, Nov 2nd, at 11.59pm. This assignment is worth 10% of your final grade.

Any sign of collaboration will result in a 0 and being reported to the Graduate Academic Integrity Board. Late submission policy described in the syllabus will be applied.

**(100 points)**

We can use binary-search and red-black trees in order to sort an array of n integers by inserting them into an empty tree, and using a modified INORDER-TREE-WALK algorithm to copy the elements back to the array sorted.

**1.** You are given an implementation of red-black trees. Implement a binary-search tree with the corresponding functionality. You can omit the delete functionality for binary-search and red-black trees, but you have to update the insertion routine of the binary-search and red-black tree to handle duplicate values. The insertion functions do not insert a value if the value is already in the tree.

**2.** Modify the INORDER-TREE-WALK algorithm for binary-search and red-black trees such that it traverses the tree in order to copy its elements back to an array, in a sorted ascending order. The number of elements in the tree might be less than n due to the elimination of key duplicates. The function should therefore return the number n' of elements that were copied into the array (number of tree elements).
**Notes:** The algorithm relies only on the binary-search tree properties which also red-black trees satisfy. Keep in mind that only the first n' elements of your array are afterwards sorted.

**3.** Implement an insertion function that has an array of integers and the array length as arguments. Modify your insertion routine for binary-search and red-black trees such that it counts the following occurrences over the sequence of insertions.

- Counter for the number of duplicates.
- Counter for each of the insertion cases (case 1, case 2, and case 3) (red-black tree only).
- Counter for left rotate and for right rotate (red-black tree only).

You should have 1 counter for binary-search trees and 6 counters for red-black trees altogether.

**4.** Develop a test function for red-black trees such that, given a node of the red-black tree, traverses to each of the accessible leaves and counts the number of black nodes on the path to the leave.
**Notes:** You could use your test function to verify whether or not your red-black tree implementation satisfies red-black property 5.

**5.** Measure the runtime performance of your "Binary-Search Tree Sort" and "Red-Black Tree Sort" for random, sorted, and inverse sorted inputs of size n = 50000; 100000; 250000; 500000; 1000000; 2500000; 5000000. You can use the provided functions *create random, create sorted, create reverse sorted*.
Repeat each test a number of times (usually at least 10 times) and compute the average running time and the average counter values for each combination of input and size n. Report and comment on your results. How do the counters behave and how is the height of the respective trees in comparison to how the running time behaves.

(You might have to adjust the value for n dependent on your computers speed, but allow each test to take up to a couple of minutes. Start with smaller values of n and stop if one instance of the algorithm takes more than 10 min to complete).

Remarks:

- You are not allowed to use code from online resources. Your submission will be tested against that, and will receive a 0, and a report to the Graduate Academic Integrity Board if it is detected.
- No additional libraries are allowed to be used
- Makefiles are provided for both problems to build the code in LinuxLab.
- The programming, and testing will take some time. Start early.
- Feel free to use the provided source code for your implementation. You have to document your code.