**Introduction:**

In this Programming Assignment, I will create and test three different types of binary trees and test their runtimes. The three types of Binary Trees I will be testing are Binary Search Trees, AVL Trees, and Red-Black Trees. I tested these trees with inputs ranging from 10 to 100,000 to see which was the fastest.

**Theoretical Analysis:**

The Binary Search Tree should run on O(n) because that is the time complexity. If we look at the resulting runtimes, we can see that they support our statement. On the other hand, theoretically, the Red-Black Tree and the AVL Tree both have a runtime of log2(n). This is not shown by the data, as the runtimes increase a lot faster than the Binary Search Tree’s. The reason AVL and Red-Black increase faster is because they are more organized, and the rotations make it take more time. Log2(n) is lower, but the rotations also take a time of O(1), which increase the times higher than that of the Binary Search Tree. Red-Black also took slightly more time than the AVL Trees, mainly because the coloring adds operations with a lower time complexity than log(n), which increase the total runtime higher than the AVL Tree. Interestingly, when I inserted random numbers for the AVL and Red-Black Trees, the runtimes got closer to that of the Binary Search Tree.

**Experimental Setup:**

I am currently running this on a computer with 32GB RAM, however, Visual Studio only allocates 2 GB of processing memory for any given project. I am using Visual Studio 2019, which runs C++17 with experimental features of C++20. For my timing mechanism I used the high-resolution clock inbuilt function to accurately time how fast my program ran. I only ran each tree for one trial, but ran them 3 times by inserting increasing, decreasing, and random values.

**Experimental Results:**

This is graphed on a log-log scale, increasing by a factor of ten on both the x and y axes. As clearly indicated by the data, Binary Search Trees have a faster runtime for all insertion techniques. However, the distance heavily slims when the numbers inserted are random because the height of the BST is lower, so the runtime is higher. I assume that if I use much higher numbers, eventually the Binary Search Tree will be much slower than the AVL or Red-Black Trees. Some of my implementation for Red-Black and AVL trees may also be incorrect or maybe my Binary Search Tree is bogged down by a mistake causing high runtimes.