Thomas Rafeld, M12808058

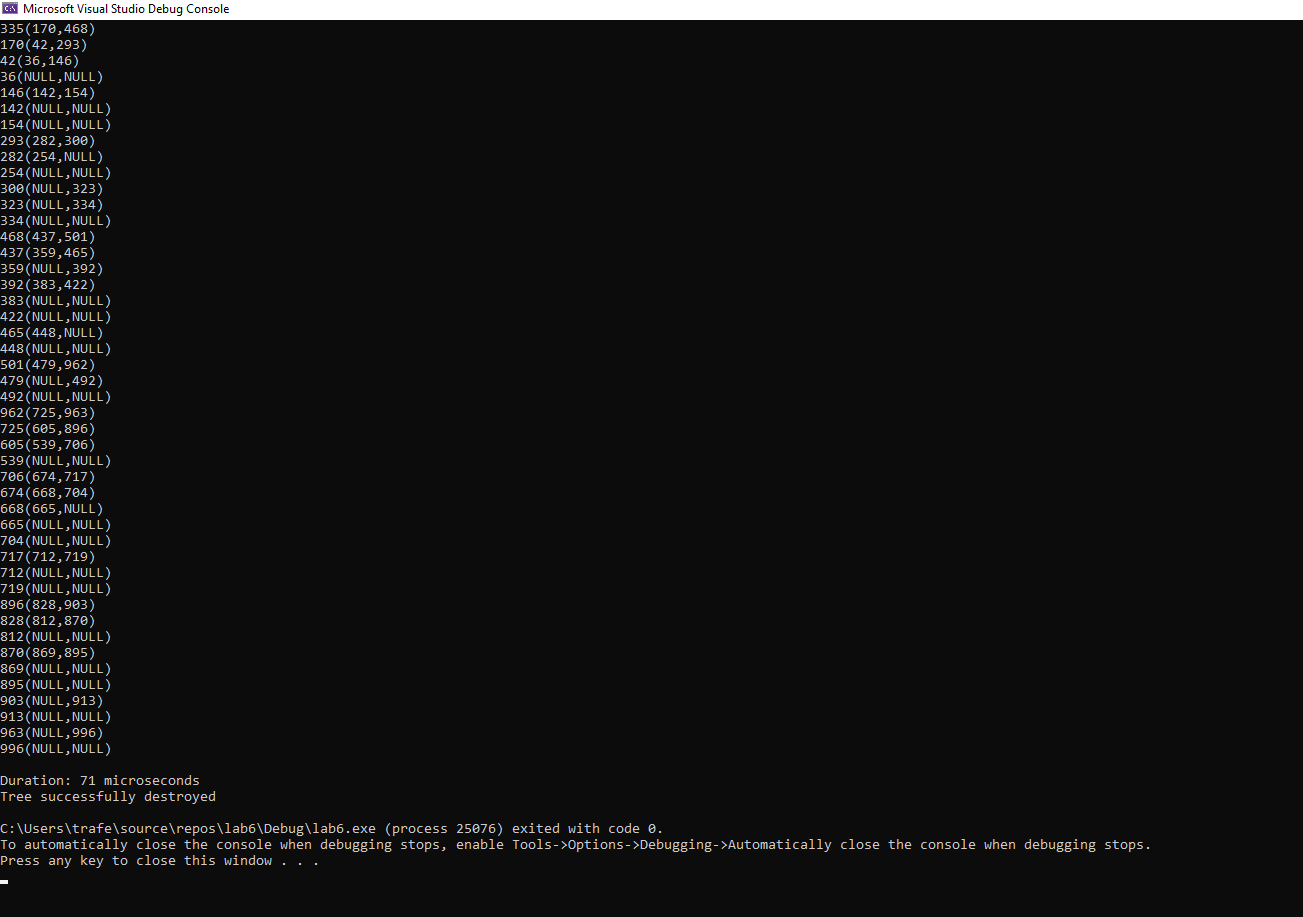
Ryan Winterhalter, M12727389

Lab 6 Report

**Objectives:**

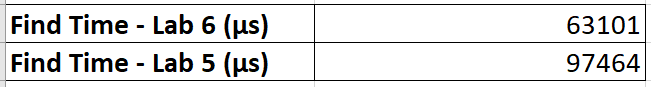
The objectives for this lab were understanding how the shape of the binary tree impacts efficiency as described by big-O notation, understanding how to determine where a balance is necessary, and building balanced Binary Trees. Understanding how the shape impacts efficiency is important to this course because efficiency is an essential part of all data structures and our data structures can take on multiple different shapes. In this course, we have discussed efficiency for each of our structures so understanding how the shape of the tree affects BST efficiency for both inserting and finding items is needed. Understanding a structure's efficiency helps to understand how it works and what structure would be the most useful for your particular application, which is why it is important to understand for a career in engineering; by understanding structures’ efficiencies and what impacts it, an appropriate structure can be chosen for the job being done. Understanding when to balance the tree is important to this class as it is a major part of AVL trees and other self-balancing trees that we have learned about in class. To be able to perform the rotations for the tree, we have to be able to determine whether it needs balancing. In order to be able to complete the lab itself, a basic understanding of this objective is needed. It is important to this class to understand the many structures that will balance themselves. It is also important to a career in engineering if we are working on a project where a balancing tree structure needs to be implemented. To be able to implement this we would have to understand when balancing is needed and where it is needed. The final outcome of building balanced binary trees is needed for this class both for the purposes of completing the lab and the overall objective of understanding data structures. It is not only important for this lab as that is the whole objective of the lab but also to the class overall as being able to build them will help increase our understanding of that data structure. If we were working on a project that needed a balanced binary tree, understanding and being able to build one would be essential to our career in engineering.

**Task 2 Diagram:**

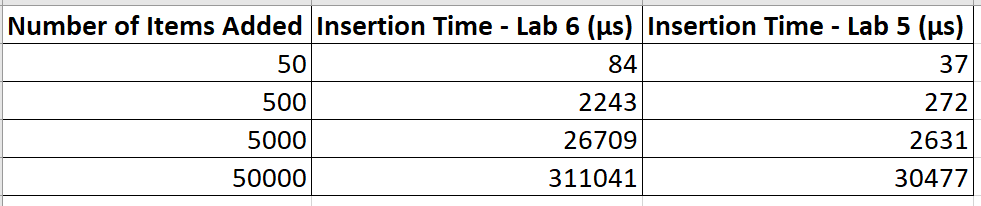
****

**Figure 1**: General structural diagram of our binary tree structure from Task 2.

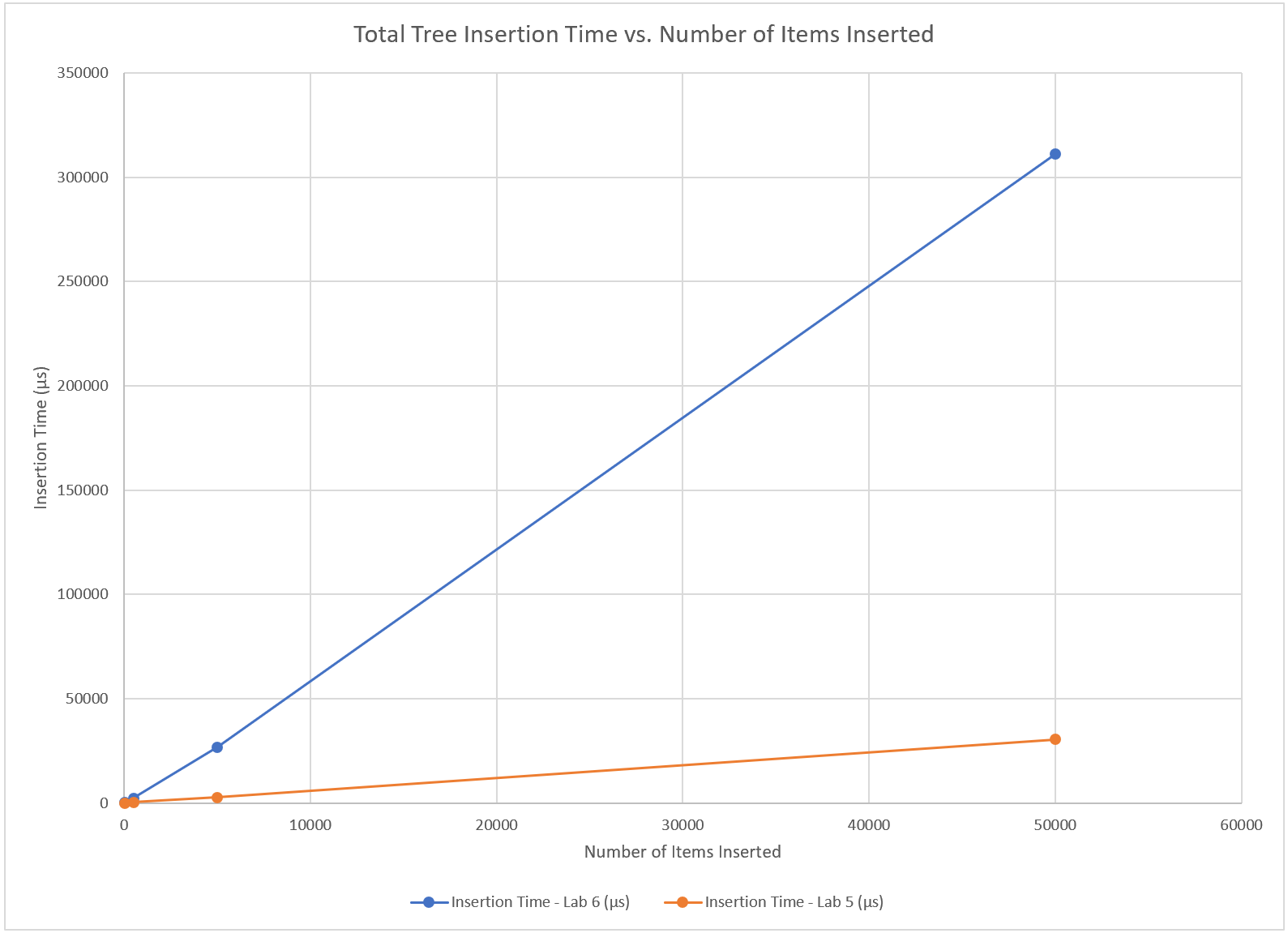
**Task 3 Performance Data:**

****

**Table 1**: Time in microseconds it took to find different items in our binary tree structures (including items that were and weren’t in the tree). Lab 5 is a normal binary tree structure and Lab 6 is a self-balancing binary tree structure.

****

**Table 2**: Time in microseconds it took to insert a certain number of random numbers into our binary tree structures. Lab 5 is a normal binary tree structure and Lab 6 is a self-balancing binary tree structure.

****

**Figure 2**: Graph comparing the insertion times in Table 2 for the normal binary tree structure (Lab 5) and the self-balancing binary tree structure (Lab 6).

The performance data that we gathered for both inserting items into our tree and finding items within our tree follows the trends we expected our data to follow. When inserting items into a self-balancing binary tree, there will be instances where your program will need to balance the tree, which is an expensive process, especially if you are inserting a lot of items into the tree. As a result, we expected our insertion times to be greater for the self-balancing binary tree structure (Lab 6) than for the normal binary tree structure (Lab 5). Additionally, we expected that the difference between insertion times for the two tree structures would increase as the number of items being added to the tree increases. As can be seen from the graph in Figure 2, our insertion times of the self-balancing binary tree structure (Lab 6) were greater than our insertion time of the normal binary tree structure (Lab 5) and that the difference between the two times increased as the number of items being added to the tree structures increased, which makes sense considering that an increase to the number of items being added to the tree requires more rotations to balance the tree.

In regard to finding items in both of our tree structures, we expected that the time it would take to find items in the self-balancing binary tree structure (Lab 6) would be less than the time it would take to find items in the unbalanced binary tree structure. As the self-balancing binary tree structure is balanced, we expect to be able to find items within the tree faster as there are fewer nodes we have to check to find an item (or find that it isn’t in the tree) than our normal, unbalanced binary tree structure. As can be seen from our data in Table 1, the time it took for the self-balancing binary tree structure (Lab 6) to both find items in the tree and attempt to find items that weren’t in the tree was less than the time for our normal, unbalanced binary tree structure (Lab 5).

**Group Contributions:**

The lab was worked on together by both Ryan and Thomas while on a call together in Microsoft Teams. Both worked on adding the rotation code to the header file along with modifying the insert and remove member functions to keep the tree balanced. The code for finding the grandparent and parent for the rotations was written by Ryan. The main file for task 2 was written by Thomas. For the final grade each member of the group should receive 100 percent of the grade as we feel that we both evenly contributed to the lab and worked together for almost the whole time it was being worked on.