John Leonardo + W Reed

7 March 2018

CSC 130

**Report (Project 2)**

This project took a total of ~12 hours for us. There was only one major obstacle. Due to a bad understanding of how generics and class inheritance works, we had to re-do the entire project after working on it for a while. We had written the Binary Search Tree class incorrectly. The problem was that we used references in Java API docs to write the BST, and we used lecture slides to write the AVL. This turned out to be a problem because both the classes were confused on which methods to call, and what was supposed to happen. After searching for errors for a couple hours, we decided to purge the entire project and start over fresh. The next time around, we wrote the code a lot cleaner, and more organized so that we knew exactly what our thought-process was if we had to go back and check it. This worked out for the better, and we were also a lot more careful with the class inheritance and access properties.

The biggest takeaway from this project (by far) was what finally led us to understand what BigO time complexity really is. I’m not afraid to admit that before this project I still didn’t understand what BigO was. I always thought it was some math formulas for busy work in computer science. I was very wrong. The biggest example of BigO are the performance results of AVL trees vs BST trees. We know from lecture notes that BST operates in a linear fashion, and AVL in a logarithmic fashion; however, we have never witnessed this first-hand until this project. For example, at 100,000 insertions (random integers), BST’s runtime was 36 milliseconds, and AVL’s runtime was 69 seconds. Obviously, with this input size, the binary search tree is crushing the AVL by almost half its runtime. However, scale that way up, and at 100,000,000 data points, the BST’s runtime was 329 seconds, whereas the AVL’s runtime was 245 seconds. This logarithmic growth style of AVL really starts to show with this, and showing its ability to perform efficiently with larger input sizes. This is extremely crucial to production code that has to support large amounts of data. If a company has a messaging platform, and messages take 20 seconds to load, the users will be annoyed and may not even use the service. This was the first assignment in Computer Science at Sac State that finally helped me understand the importance of time complexity.