CSCI-2270 Final Project Report

In order to solve the problem presented by the United States Postal Service's numerous tracking IDs during the COVID-19 pandemic, we have implemented three potential data structures to reduce searching and inserting time. A linked list, binary search tree, and hash tables with three different types of collision resolution (chaining, quadratic probing, and linear probing) were implemented in order to determine time complexity for search and insert. The figures below depict the summary of inserts and searches for both data set A and data set B. The x-axis represents iterations of 400 deltas each representing the average time it takes to insert or search for 100 records. A high resolution clock from the built in C++ Chrono library was implemented to keep track of time.

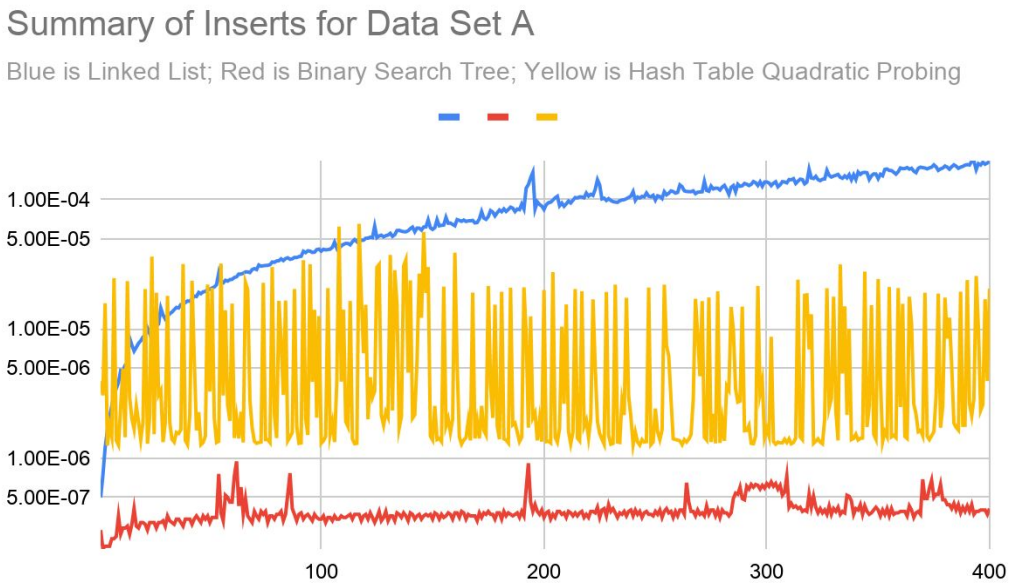

Figure 1: Summary of Inserts for Data Set A on all three data structures. A logarithmic scale was implemented to make visualization of results clearer.

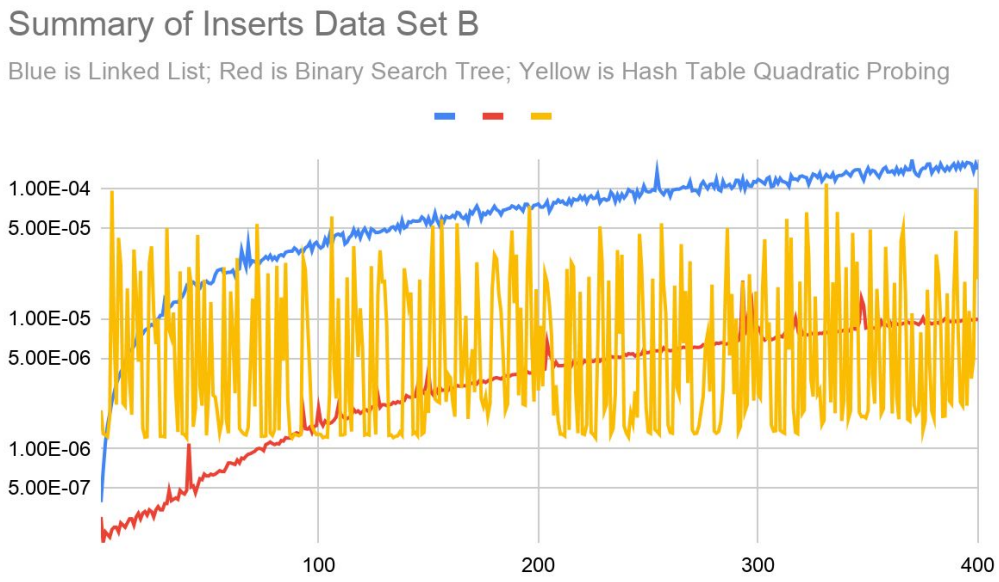

Figure 2: Summary of Inserts for Data Set B on all three data structures. A logarithmic scale was implemented to make visualization of results clearer.

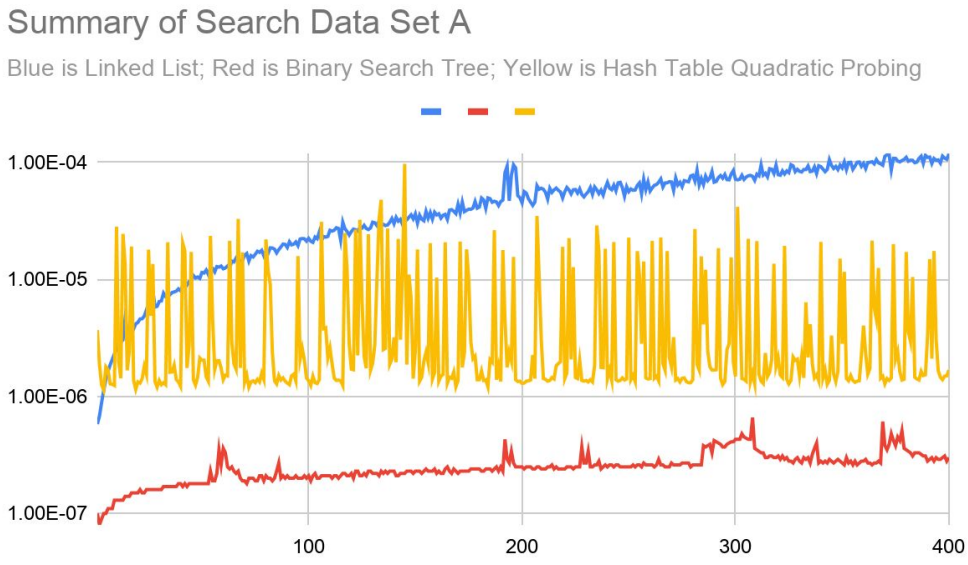

Figure 3: Summary of Searches for Data Set A on all three data structures. A logarithmic scale was implemented to make visualization of results clearer.

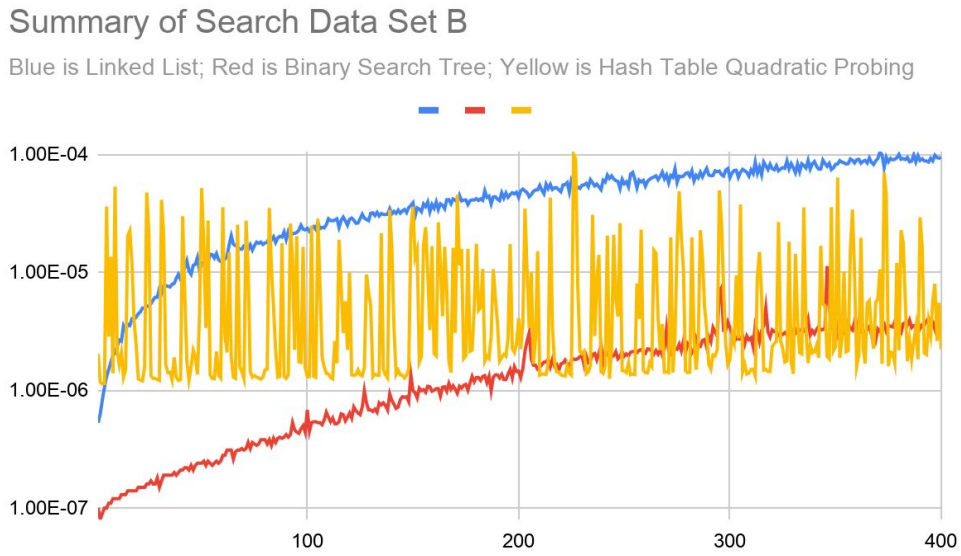

Figure 4: Summary of Searches for Data Set B on all three data structures. A logarithmic scale was implemented to make visualization of results clearer.

Following the implementation of three different data structures to accommodate the tracking ID data files of the USPS, we have found that the binary search tree insert is the most efficient with a big O notation of O(logN) where N is the number of records in the data file. The most effective search function was provided by the Binary Search Tree with an O(logN) performance. Although the hash table quadratic probing insert, search, and delete times complexity in theory should be O(1), we noted a large number of collisions which dragged the insert performance to O(N). It is evident that the figures above show that the BST yielded faster insert and search times than the linked list and hash table. As a result, the best way to insert and search for records in the USPS tracking application is a binary search tree with a performance of O(logN).

A binary search tree can also support displaying keys in ascending order by using an inorder traversal. A hash table on the other hand makes it harder to ascertain all keys in an ascending order since the division hashing function can hash two adjacent values completely differently based on a modulus function. Other benefits that the BST holds over hash tables include ordered statistics and fast range queries, as well as easier implementations. We believe that the BST's performance was better than the hash table in this project due to the fact that the hash table requires more expensive operations such as re-hashing with the various collision resolution methods. A tree does not need to handle collision issues and the data is sorted in order. In cases where there is a lot of data to analyze, an ordered list may be more beneficial to search or insert in.

A linked list data structure would not be the best choice to use for the USPS to use because while inserting may be an O(1) operation, searching to the end of the list and inserting at the end costs O(N) complexity while search at worst case is O(N). Furthermore, while there are numerous advantages with hash tables, such as better cache behavior and less memory reads, it was the binary search tree that proved to be the more efficient data structure with inserting and searching data.