Catherine Brennan
Michael Ghattas
April 29, 2020

Final Project Report

Save the USPS!

Introduction

Our team has been requested to assist set up an experiment to identify possible data structures that could improve the current bottleneck the USPS mail tracking software is facing, due to the strain put on its resources by the recent pandemic and the government's continuous failure to lead change at the United States Postal Service. The current algorithm in the software utilizes a Linked-List for inserting and searching all of its shipments, taking an enormous amount of time. The data used for tracking the shipments utilizes integer based tracking IDs. We will perform an analysis on both of these data sets to identify the most ideal data structure based on run-time performance. The data structures we will use in the experiment are Linked-Lists, Binary Search-Trees and Hash-Tables utilizing chaining, linear and quadratic probing.

Method

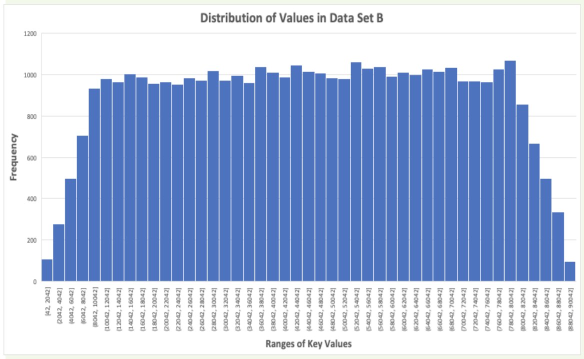
Using proven methods that utilize Big-O notation to assign processing speed for each data structure in inserting and searching for data; we will plan, design and build our own code to test the performance of each of the data-structures on the 2 sets of data provided by the USPS.

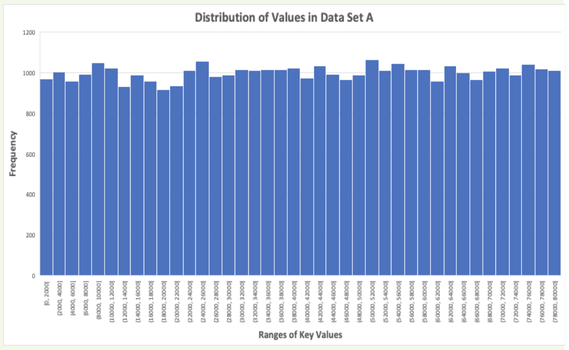
Errors

Our analysis calculations are open to possible errors possibly generated from the lack of a controlled environment in which to perform this experiment. The computer on which this experiment was performed may have been performing other operations/processes in the background, which could account for the random spikes in operation time that can be seen below (as opposed to smoother curves). Additionally, the utilized algorithms could be optimized to run faster and the code can be further improved by a more advanced programmer to provide a cleaner execution.

Test Data

The set of graphs below show the difference in the distribution of key values between data sets A and B. It can be seen that the values of data set A are evenly distributed across their range, while the values of data set B are more clustered toward the center of their range. Summary statistics below provide additional insight into the characteristics of the data sets.





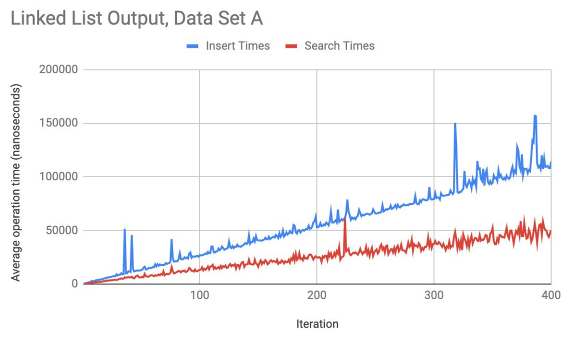
Summary Statistics	**Data Set A**	**Data Set B**
Mean	*40235.872*	*45224.5631*
Standard Error	*115.349172*	*116.173097*
Median	*40304.5*	*45335*
Mode	*54778*	*12137*
Range	*79998*	*89626*
Minimum	*0*	*42*
Maximum	*79998*	*89668*
Sum	*1609434880*	*1808982524*
Count	*40000*	*40000*
Duplicate Count	*8438*	*8196*

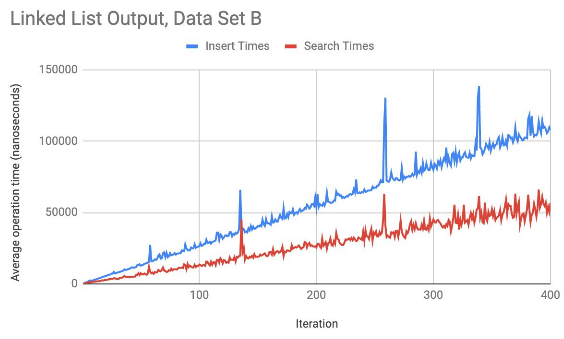
Results:

Our output data and graphs can be found at the following link:

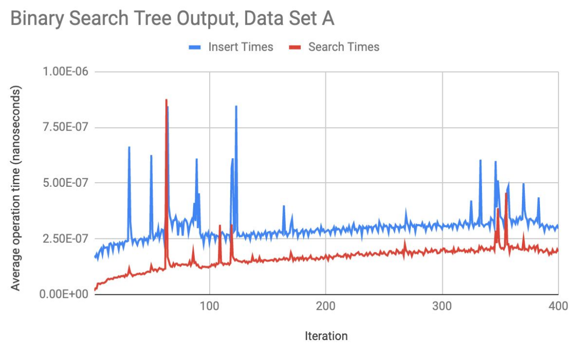
(CSCI 2270 Final Project Output Files)

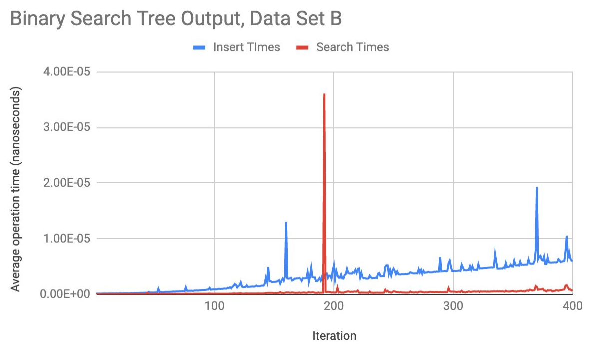
*Linked Lis*t: The linked list graphs show a constant linear increase in insert and search methods, with insert times increasing more quickly than search times.



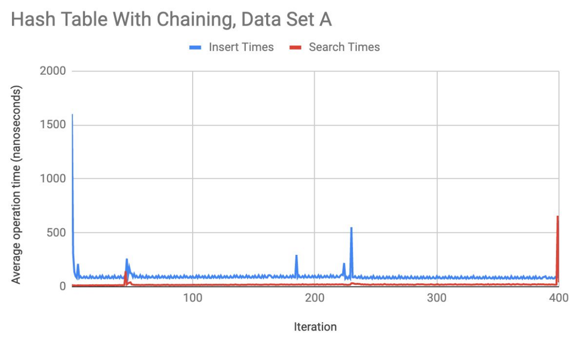


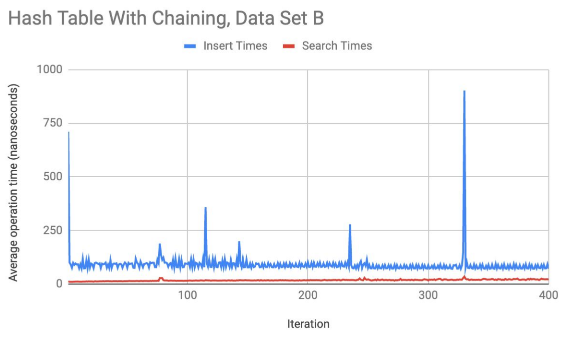
Binary Search Tree: For data set A, both the insert and search times appear to follow a logarithmic curve. For data set B, the insert and search times increase more linearly.

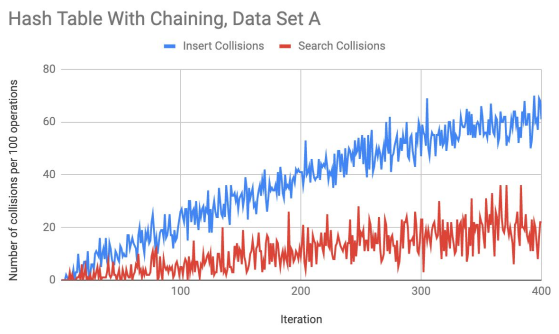


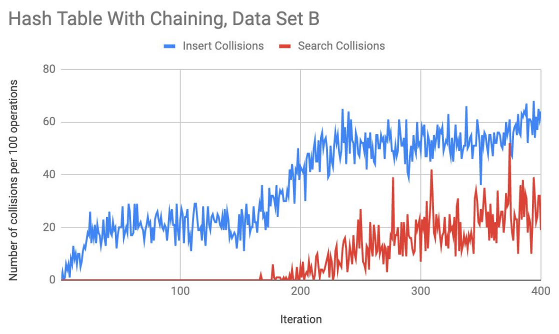


Hash Table (Chaining): Both the insert and search time methods for data set A and B remain relatively constant through most of the iterations, with the exception of the first insert iteration, which is significantly slower than the others. The search times are consistently lower than the insert times. The number of collisions for data set A and B seem to all increase linearly with an increasing number of iterations, with the search collisions consistently lower than the insert collisions. For data set B, the number of search collisions is at or near zero until the ~200th iteration. Furthermore, hashing with chaining seems to be the least efficient hashing method giving us a longer search and insertion time, though the insertion time stays relatively constant through increasing iterations.

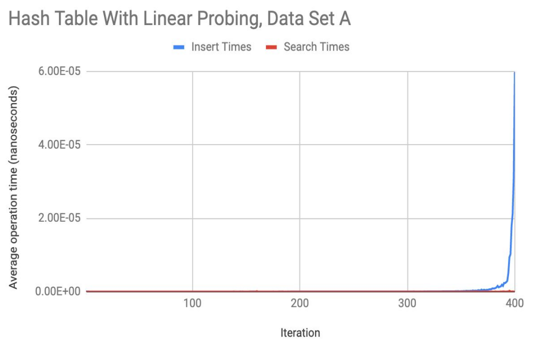


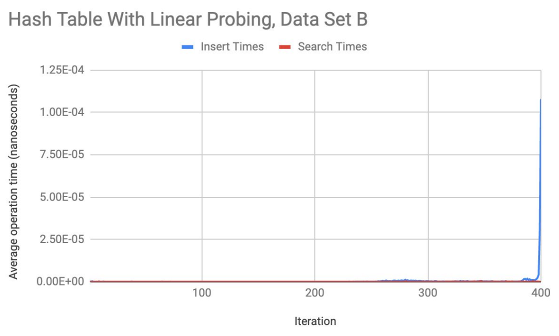


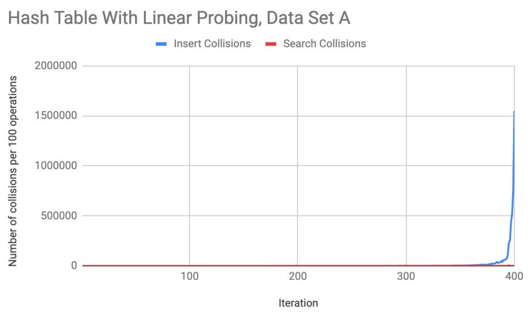


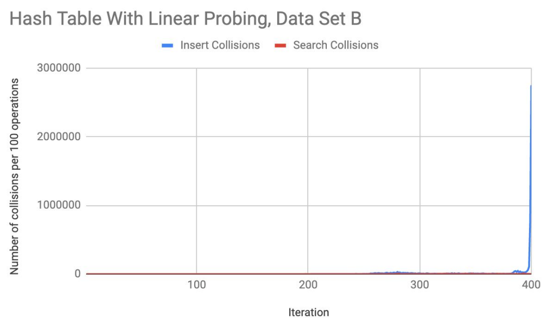


***Hash Table (Linear Probing)*:** Hash tables with linear probing were very efficient at both inserting and searching, until near to the ~350th iteration when insertion times increased exponentially. Collisions are also fairly low, until near the ~350th iteration when insert collisions start to increase exponentially due to a more fuller hash table.

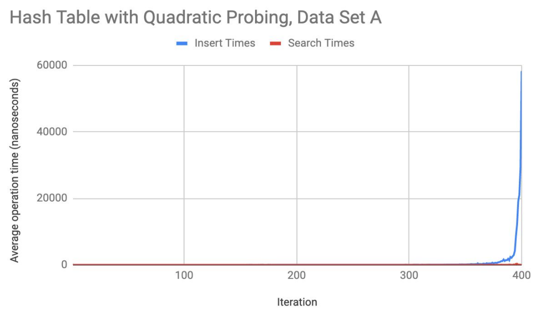


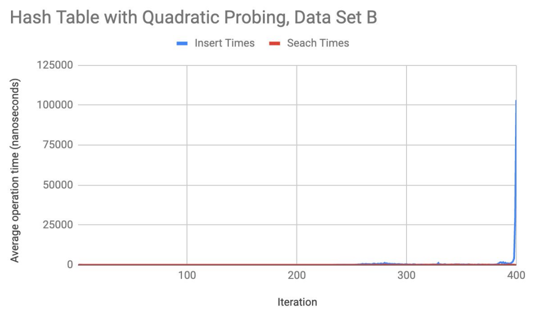


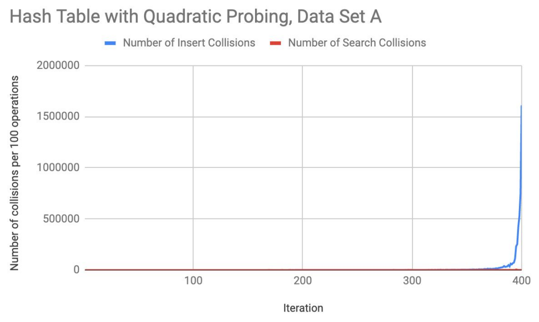


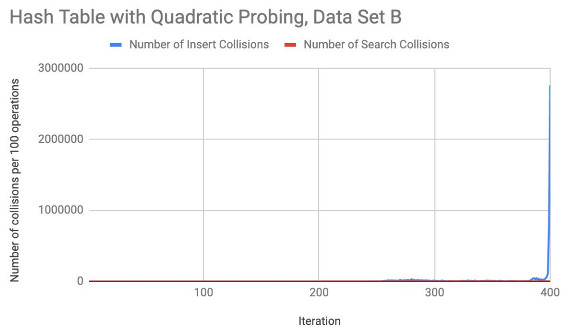


***Hash Table (Quadratic Probing)*:** Hash tables with quadratic probing were fairly efficient at both inserting and searching, although less efficient than linear probing. Near the ~350th iteration the insertion times increased exponentially. Collisions are also fairly low, until near the ~350th iteration when insert collisions start to increase exponentially due to a much more full hash table.

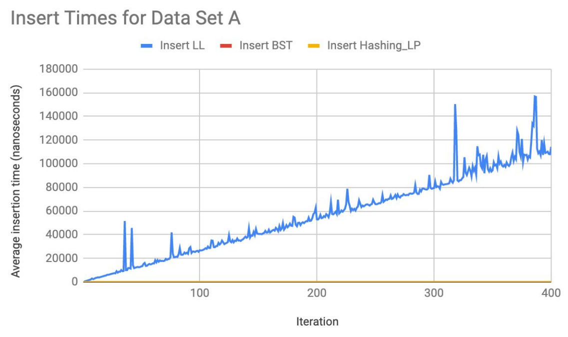


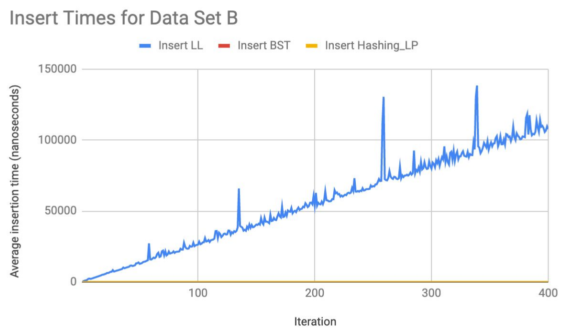




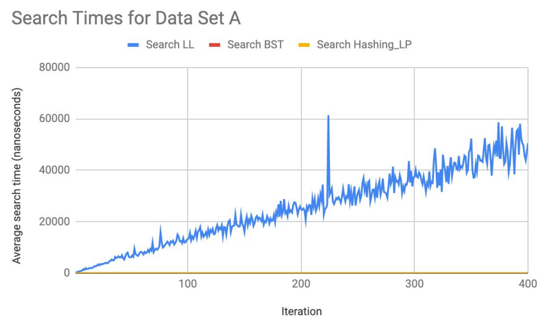


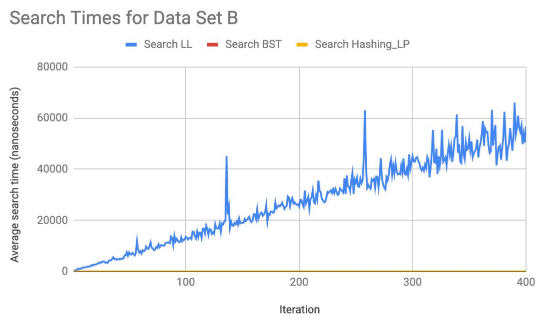
__Summary of Insert Methods__: The Hash Table with linear probing is the most efficient method, up until the ~350th iteration, at which point a BST insertion method is more efficient.





__Summary of Search Methods__: The Hash Table with Linear probing is consistently the most efficient method of searching.





Findings:

- Up to ~350th iteration, hashing with linear probing is the most efficient method of insertion.
- After ~350 iteration, BST insertion time is the most efficient.
- Linear probing is consistently the most efficient method of searching.

Summary:

These results were consistent with our expected projections based on tested methods of the operational efficiency of the tested data structures. Linked list is expected to perform less efficiently than the other tested methods. With insert/search, this is possibly due the need to traverse a list (from the head to index) during every operation. Accordingly, this leads to increasing insert and search times as more nodes are added to the list.
Binary search trees are expected to have a better performance than linked lists. This is due to the insert/search utilized methods, as the number of traversed nodes is equal to the height of the BST and this value is always less than the total number of nodes inserted.
Hash tables in general are known for their efficiency, as long as the hash table is not near it's capacity. The drop in efficiency as the table approaches its capacity is due to the increased collisions while trying to find an empty index.
Hashing with linear probing seems to be the most efficient of the hash table methods.
Hashing with chaining seems to be less efficient, and this might be explained by the time required to create a node and insert into a linked list at each index.
The lower efficiency of quadratic probing was an unexpected result. Though trends in insert times, search times, and number of collisions followed a similar pattern to the linear probing results, it is possible that the algorithm and code syntax can affect the individual operation timing. This may suggest that the hash function took longer time to compute for quadratic probing than linear probing did in our implementation.

Recommendations:

Based on our preliminary results, and as long as the USPS is working with data sets of similar size and comparable key value distribution as our test data, we find hash tables with linear probing to be the most efficient overall method for the USPS tracking system. Although binary search tree insertion operations become more efficient than hash tables with linear probing as the hash table becomes full, a slightly larger hash table relative to the data set size would avoid the lessening efficiency of the hash table implementation.
Please keep in mind that these recommendations are subject to the errors stated above, and should be treated as preliminary findings at best. We recommend further research in a more controlled environment to confirm our results and deepen our understanding of each data structure's operational advantages and limitations.

End.