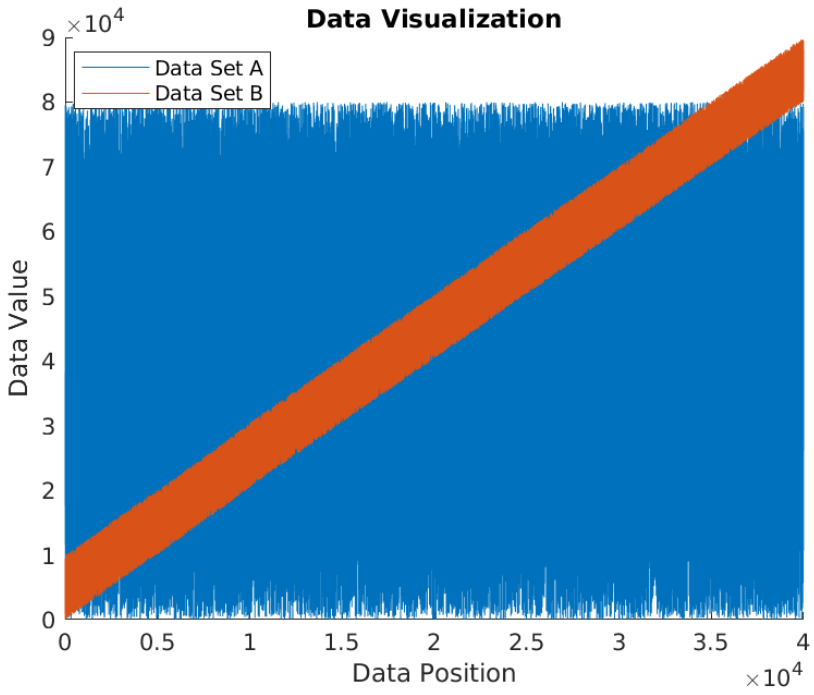
Helping Streamline Data For USPS

Luke Morrissey

April 2019

1 Introduction

USPS has given us two sets of data to test on. Before beginning it will be useful to visualize the two data sets.

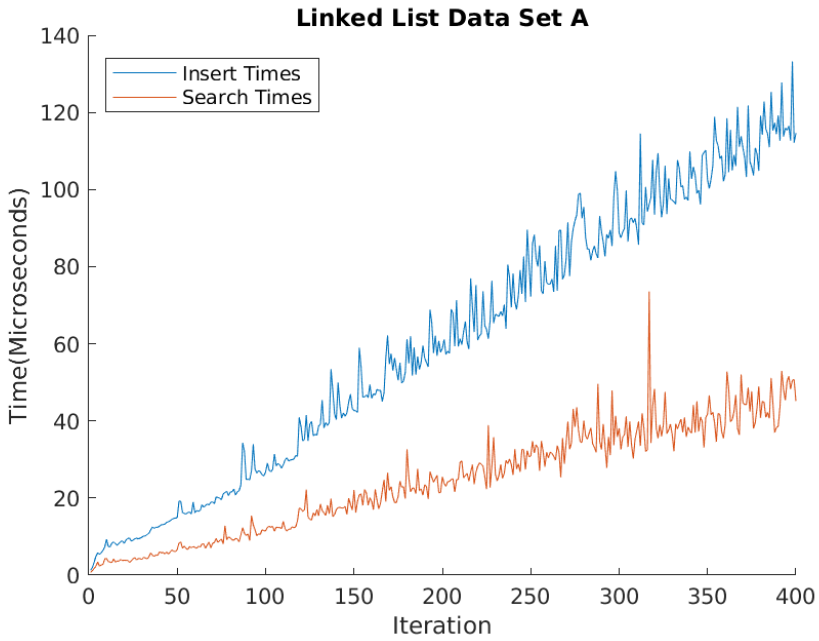


As you can see from the representation data set A appears to be random while data set B increases with the position of the data. This is important to keep in mind as we interpret the results from our different data structures.

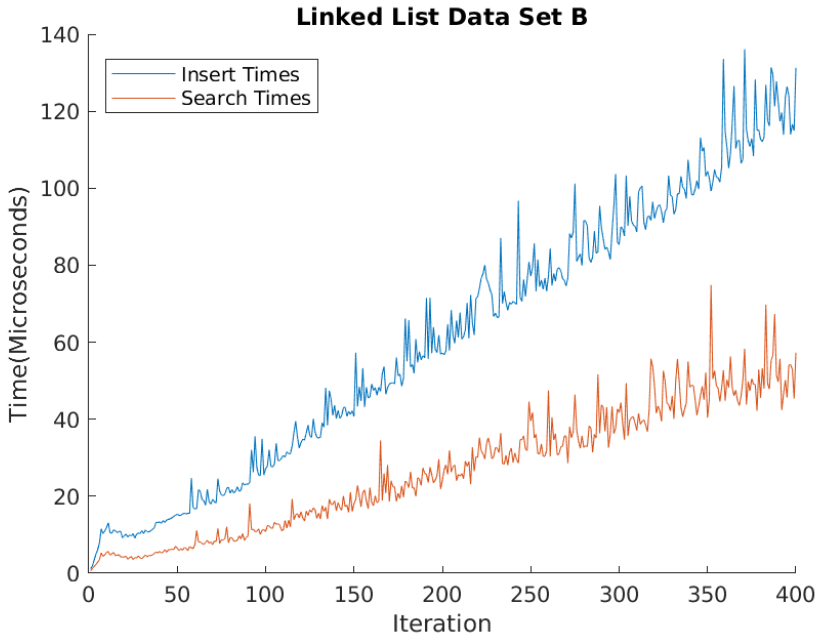
2 Data Structures

2.1 Linked List

First let's begin by looking at the current data structure implemented by the USPS, the linked list. First here is the data from data set A.



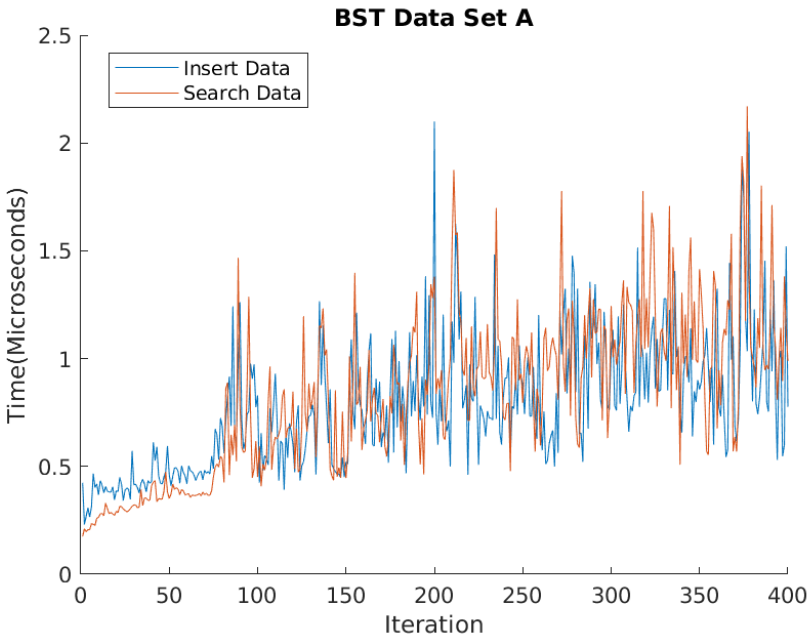
And similarly for data set B.



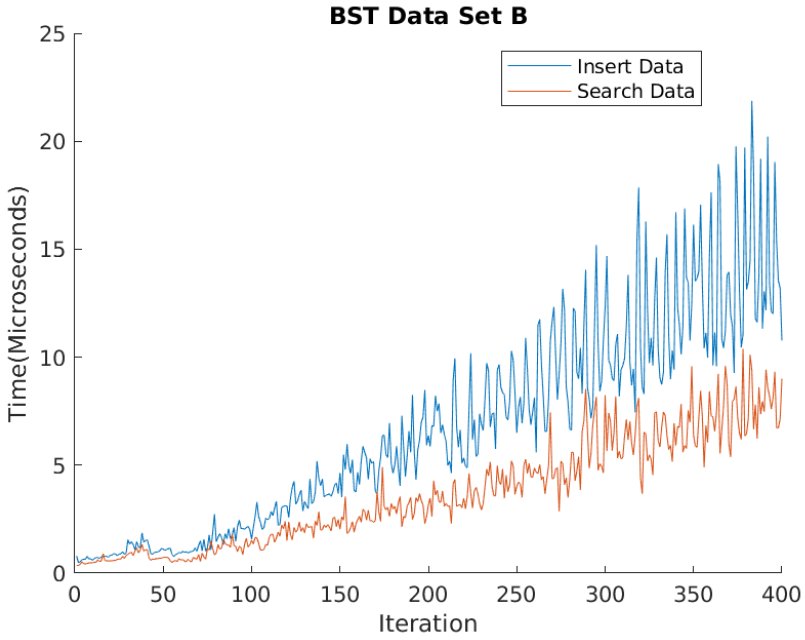
As you can see from the graphs, the insert and search times increase linearly as elements are added to the linked list which makes sense because it is O(n).

2.2 Binary Search Tree

Let us now take a look at a binary search tree. First here is the graph of insert and search times for data set A.



And similarly for data set B.

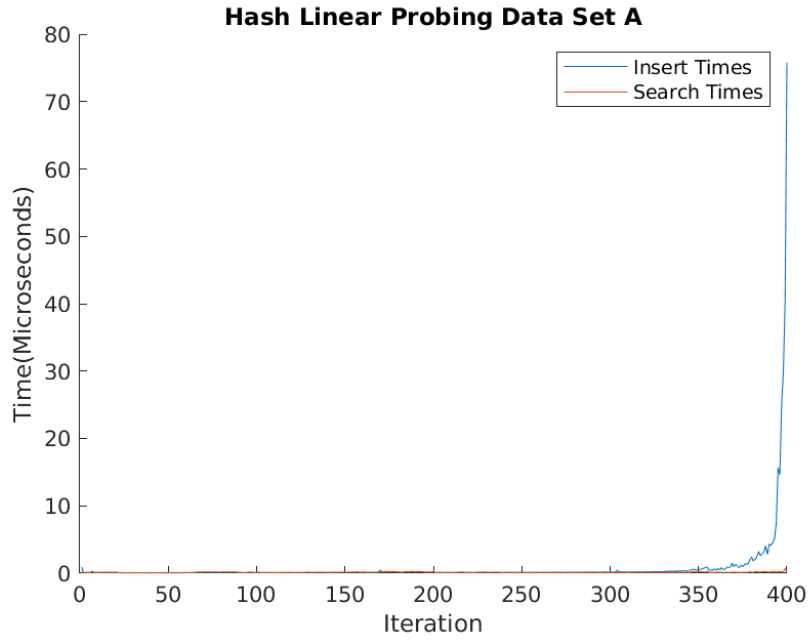


These graphs are a bit more interesting, data set A takes a lot less time to insert and search than data set B. When looking back at the data this makes sense. A random data set will create a more balanced tree and an increasing data set will create a right heavy tree making it more efficient. It is hard to tell from data set A but it is plausible that it could be O(log(n)) from the graph.

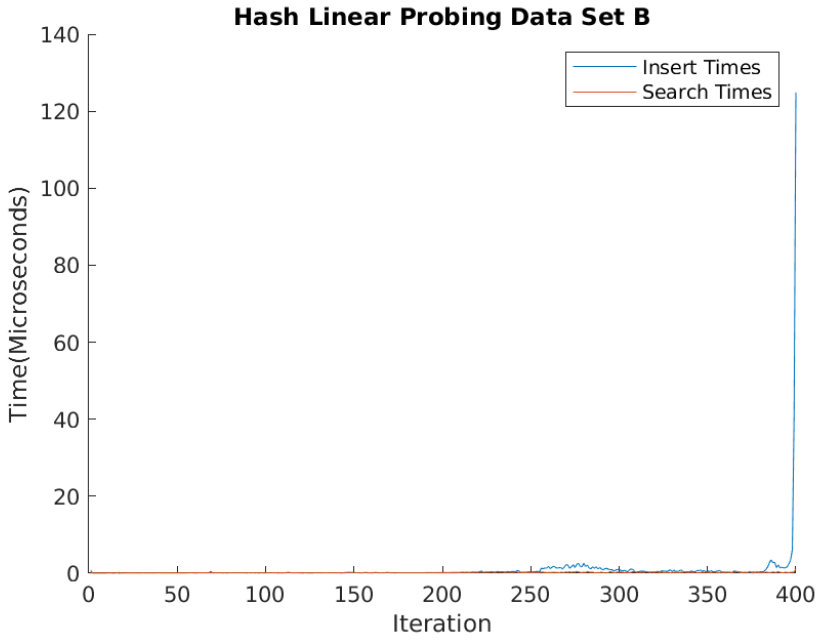
2.3 Hash Table

2.3.1 Linear Probing

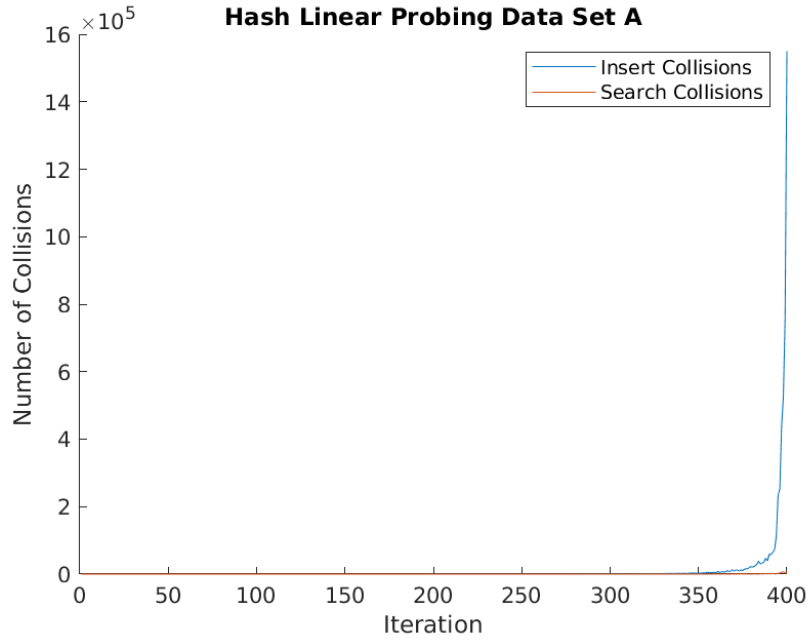
Our next data structure we are looking at is a hash table and for the first implementation I used linear probing to resolve collisions. Here are the results for data set A.



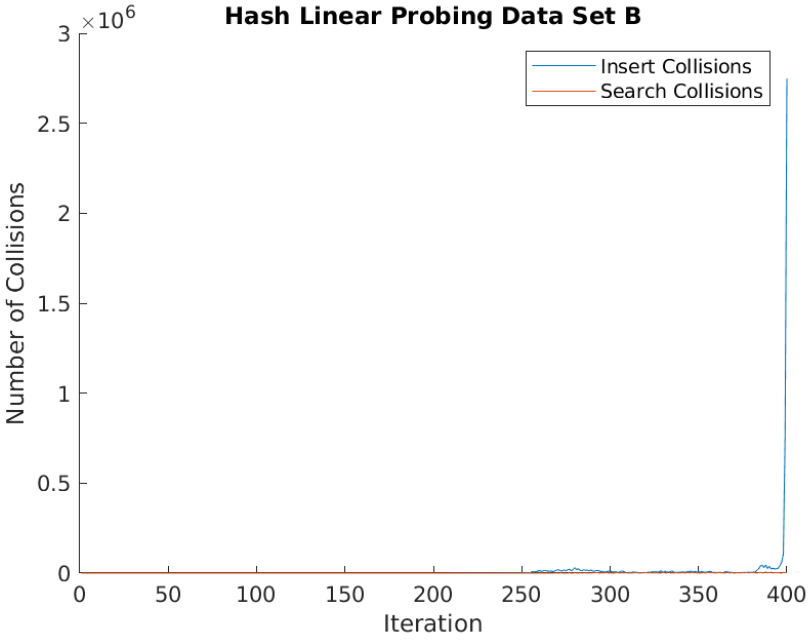
And similarly for data set B.



We also recorded the amount of collisions per 100 inserts and searches and will include that data below.



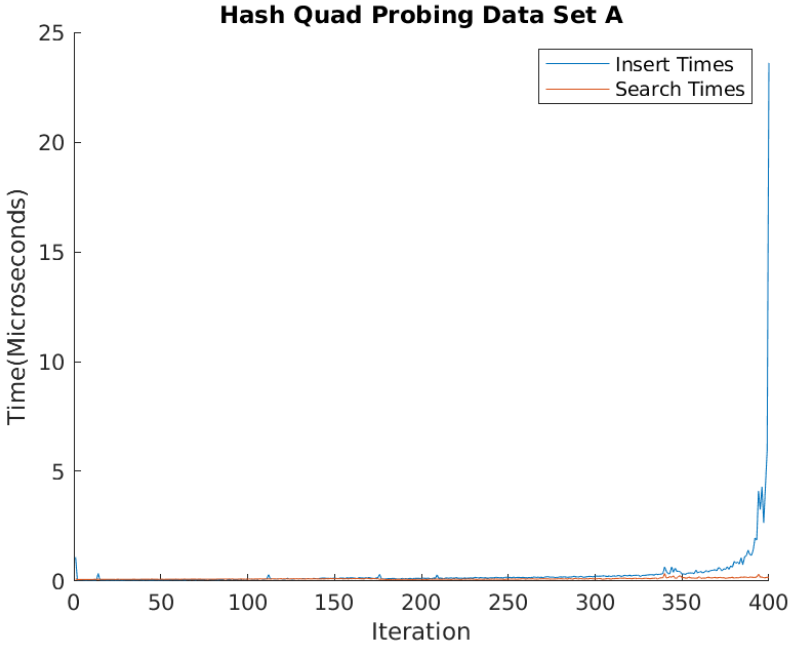
And similarly for data set B.



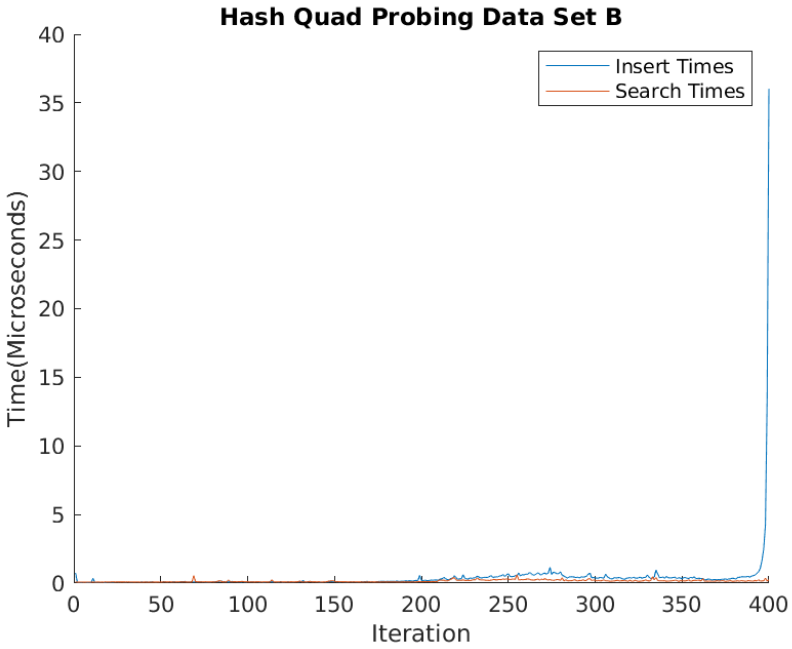
This data clearly shows the strengths and weaknesses of the hash table. While the hash table is relatively empty the inserts and searches are very quick. This is because as seen in the other graphs there are very few collisions when the table is empty. As more data is put in the hash table there are more and more collisions. At the end the time and collisions spike tremendously. This is because the table size was 40009 and there were 40000 elements. This means at the end trying to find an open spot took a very long time with a lot of collisions.

2.3.2 Quadratic Probing

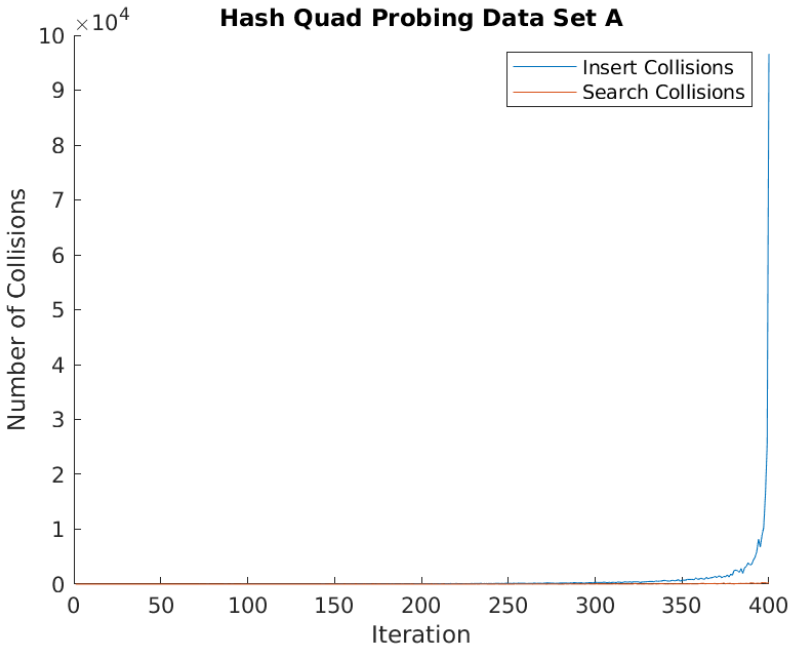
In this implementation we used quadratic probing to resolve collisions. Lets see what this data looked like.



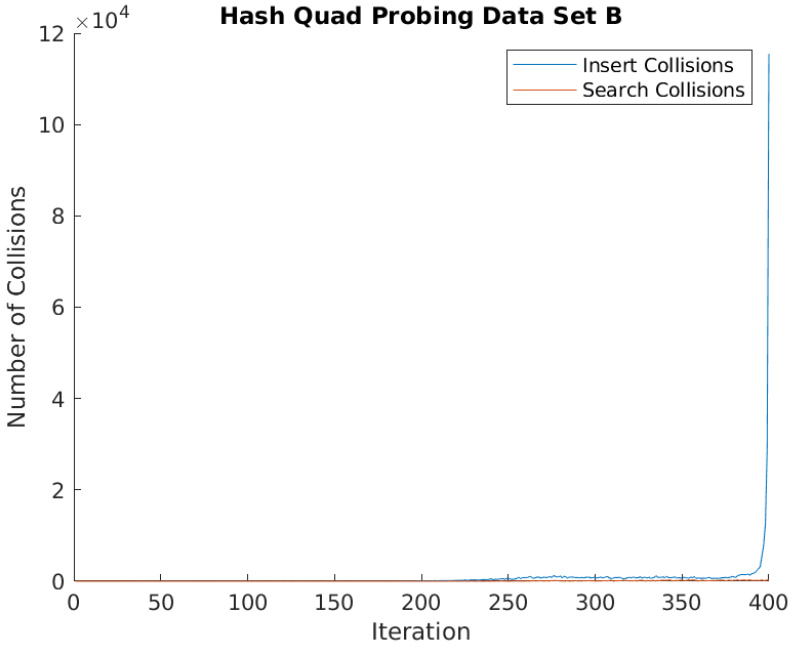
And similarly for data set B.



We also recorded the amount of collisions per 100 inserts and searches and will include that data below.



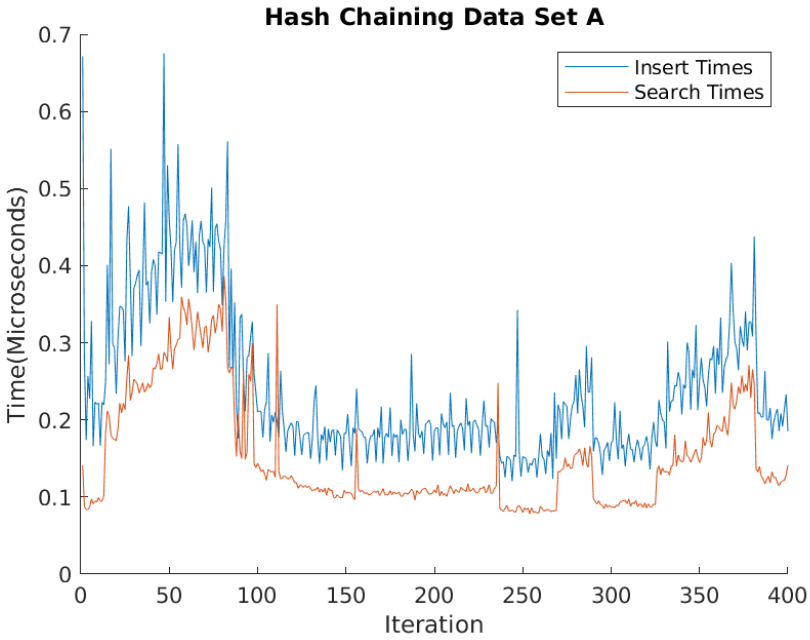
And similarly for data set B.



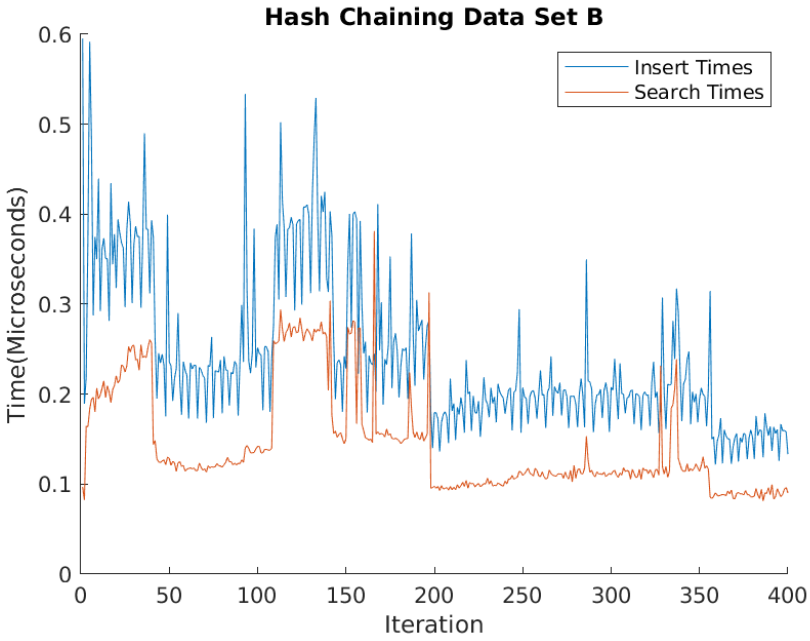
This is very similar to linear probing given that it spikes dramatically at the end but is about 3 times faster because the quadratic probing resolves collisions quicker. I would guess this is because with linear probing there is a lot more clustering.

2.3.3 Chaining

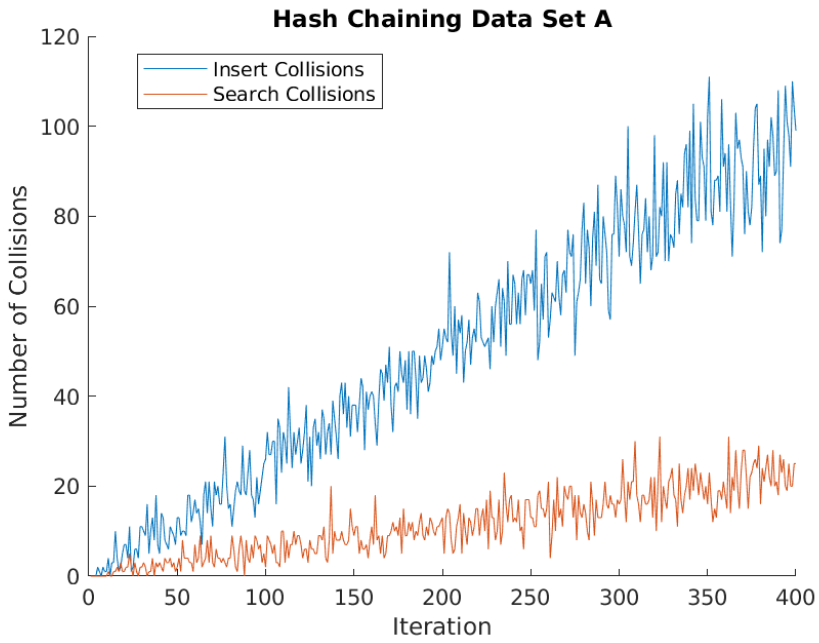
For this implementation of the hash table each hash value is essentially a linked list so when there is a collision, instead of searching for an empty slot in the table you add it to the end of the linked list of that value. Here is that data.



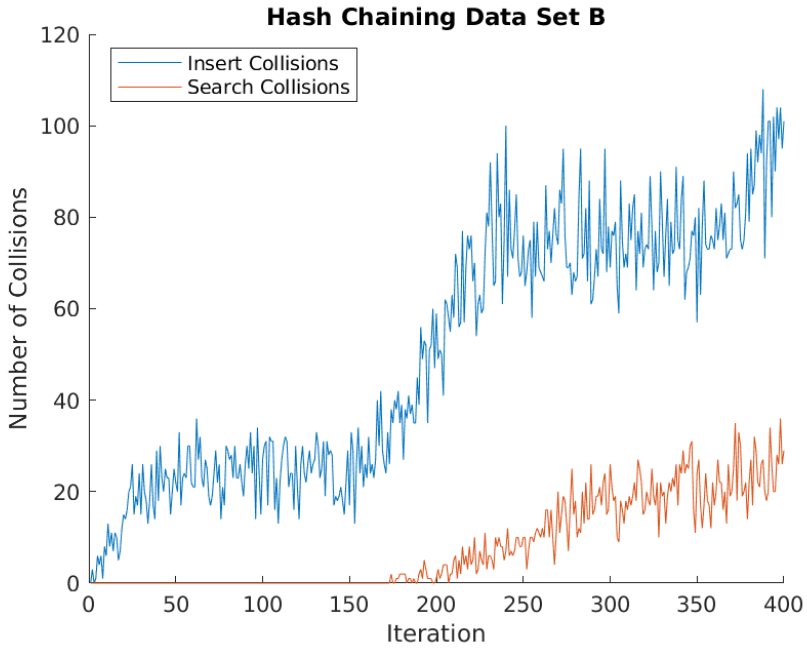
And similarly for data set B.



We also recorded the amount of collisions per 100 inserts and searches and will include that data below.



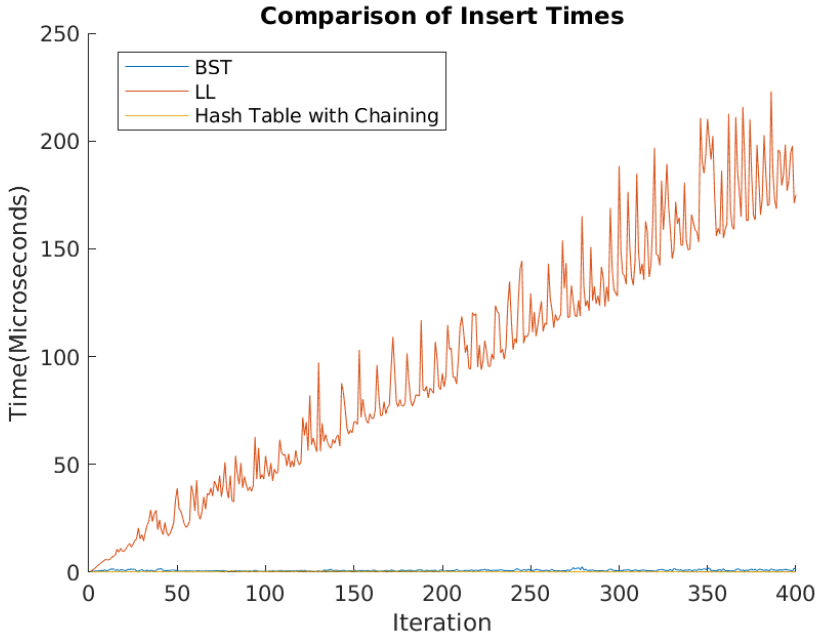
And similarly for data set B.



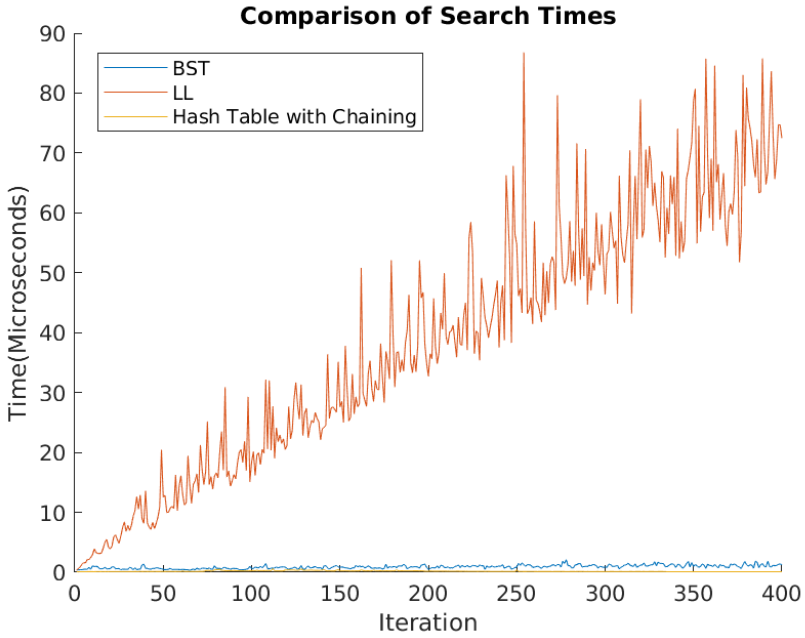
This performs very well and with some error close to that O(1) level that optimal hash tables perform at. This can be seen by the fact that insert collisions never go over 120 collisions per 100 inserts. This is minuscule compared to the collisions in the probing methods.

3 Conclusion

Finally lets take a look at how the methods compare. We will look at data set A and only include the best hashing method, the chaining one. First here is inserts.



And similarly for searches.



As seen in the graph, the linked list is far and away the worst. This is why I strongly encourage USPS to switch away from that data structure. Even using data set A, which is random and therefore optimal for BST, the hashing with chaining is quicker. Because of this I recommend that USPS switch to using a hash table with chaining as there collision resolution.