Ivan Rodriguez

Elizabeth Spaulding

CSCI 2270

4-29-20

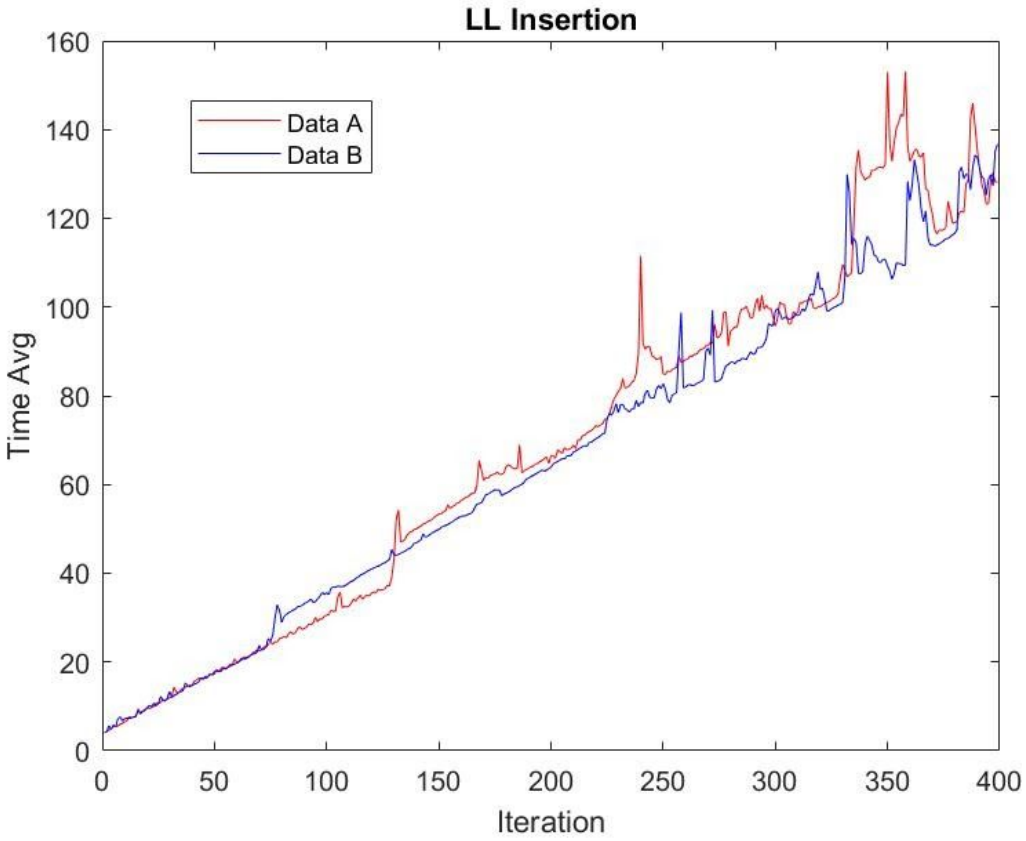
USPS Final Project

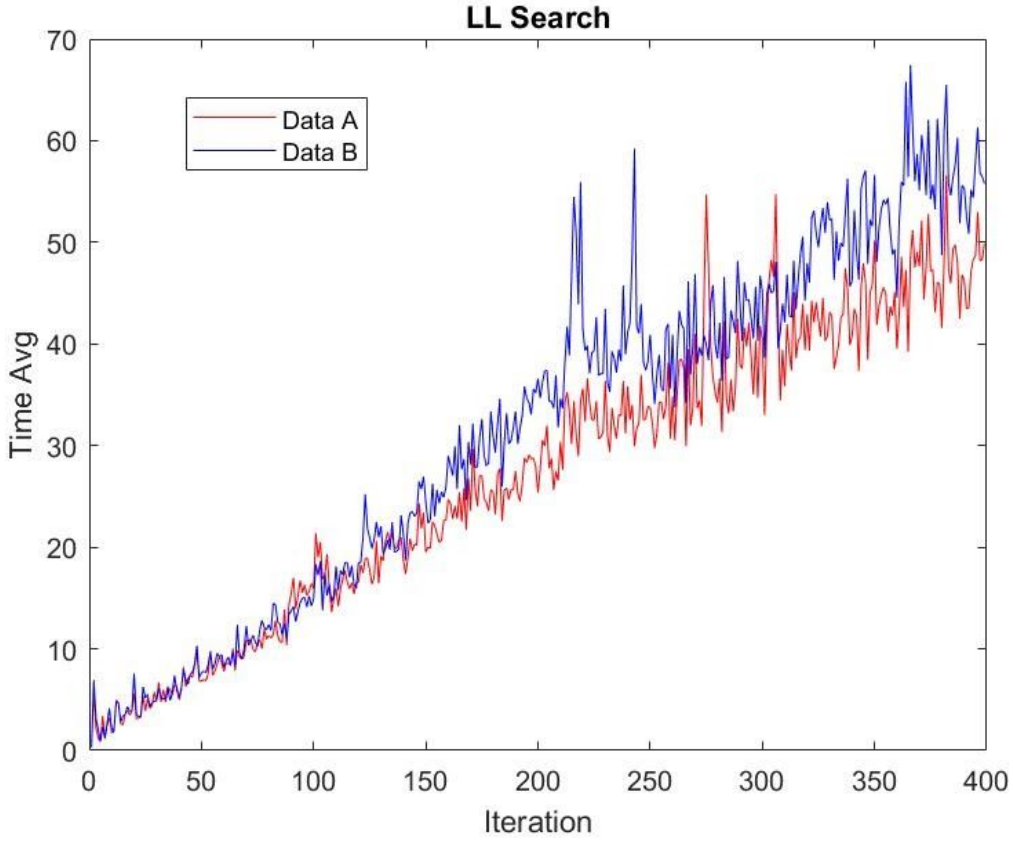
Summary

In the final project for CSCI 2270:Data Structures, we were to implement data we had received (USPS data) from a standard array, into multiple other data structures. Throughout these implementations of structures, we were to test the execution times of certain operations (insert and search). The purpose of this is to compare the execution times between structures and determine which structure performs best for the operation.

Linked List

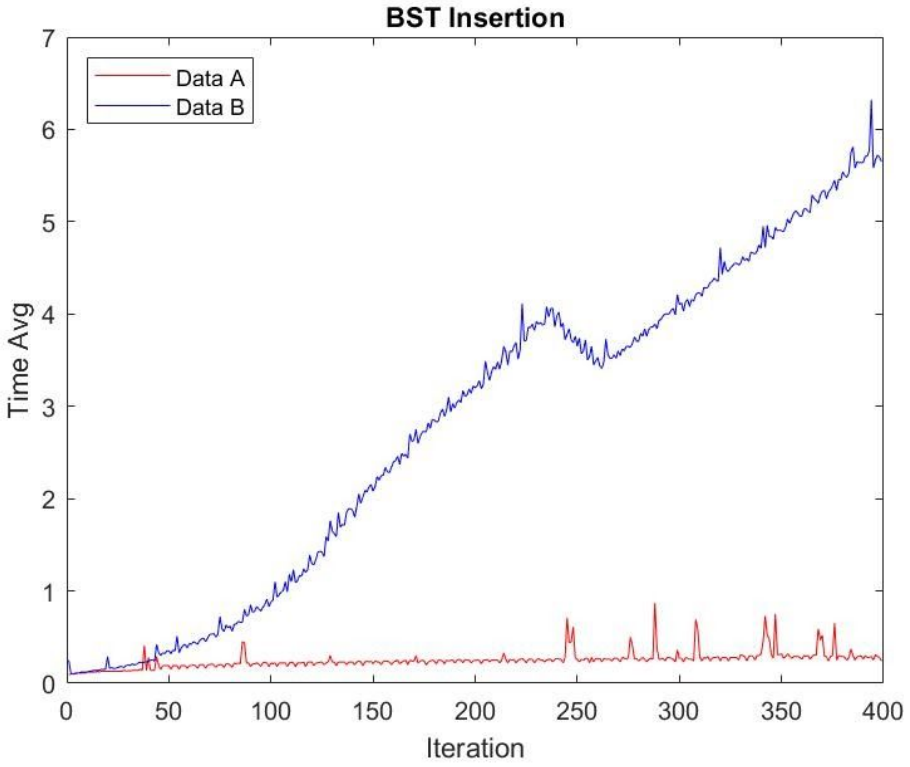
My linked list was not a good implementation in terms of the execution time. Since I was adding nodes to the end of the list, my time complexity was modeled by O(n) for insertion. For search, it has the same time complexity since it has to start from the root until it finds the searched key. On the following page, you can see from my plots that it does indeed follow the linear behaviour that I have described. One way that I thought of afterwards to improve my insertion times was to save a pointer to the last node after every insertion so that it would start there instead of having to iterate the list every time.

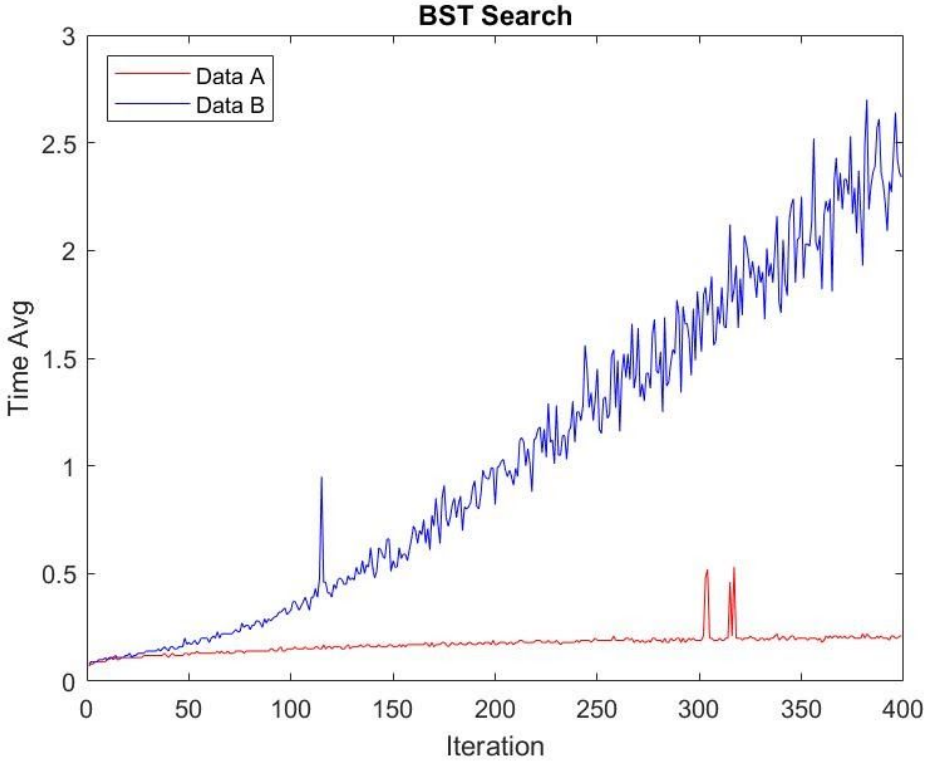

LL Insertion
Data A
Data B
Time Avg
Iteration
0
20
40
60
80
100
120
140
160
0
50
100
150
200
250
300
350
400


LL Search
Data A
Data B
Time Avg
Iteration
0
10
20
30
40
50
60
70
0
50
100
150
200
250
300
350
400

BST

In general, a binary search tree is a good data structure with a time complexity of O(log(n)). This means that after a lot of data points, the complexity kind of plateaus. As seen in the plot below, the figure representing Data A represents logarithmic behavior. It starts off going up with a slightly large slope, but then it flattens out. This would be more visible if I had lowered the bounds of the y axis. For some reason though, dataset B seems to follow a different trend. It's more linear than logarithmic. One of my guesses on this, is maybe that the tree isn't balanced.

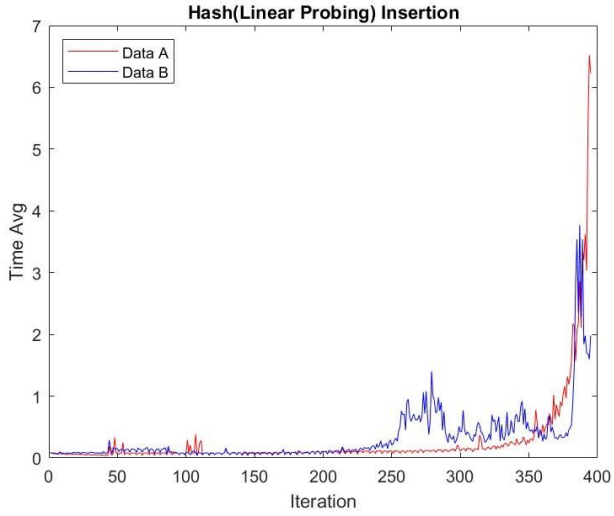

BST Insertion
Data A
Data B
Time Avg
Iteration
0
1
2
3
4
5
6
7
0
50
100
150
200
250
300
350
400

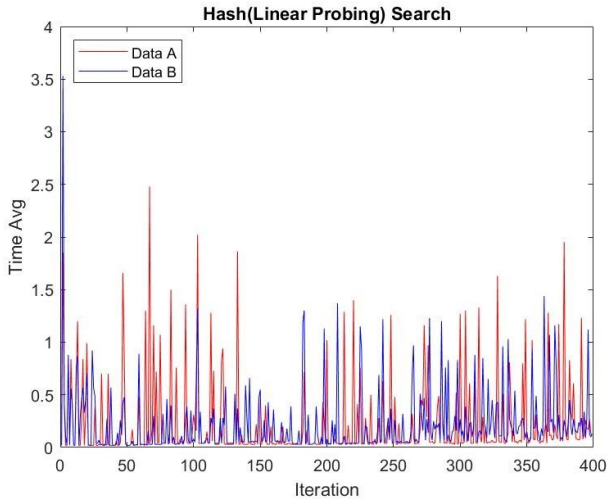

BST Search
Data A
Data B
Time Avg
Iteration
0
0.5
1
1.5
2
2.5
3
0
50
100
150
200
250
300
350
400

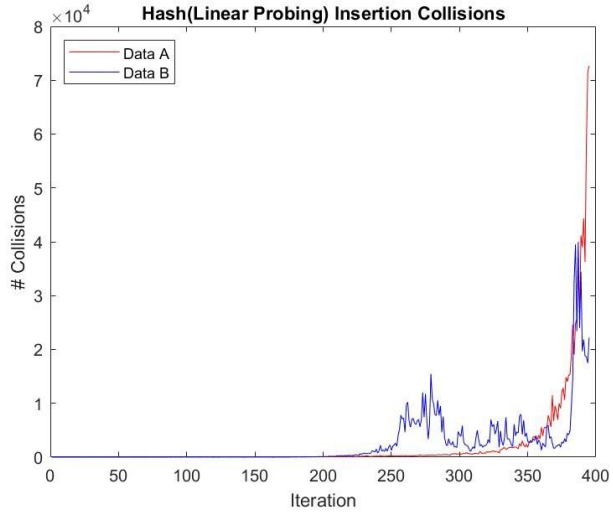
Hashing

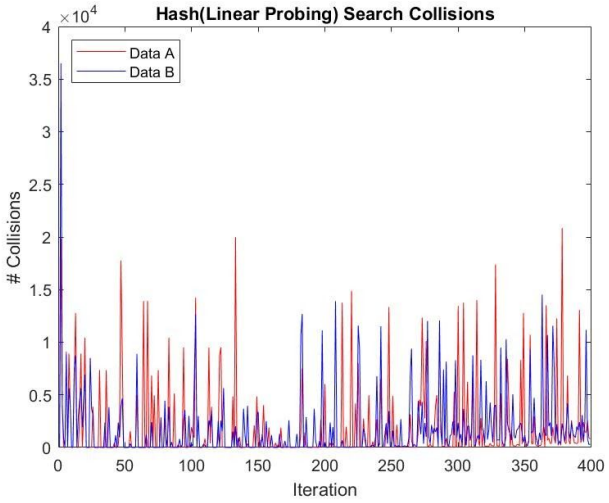
With hashing, I implemented three different methods to deal with collision. These methods were Linear Probing, Quadratic Probing, and Chaining. Before this project, I had never dealt with hashing. Now that I have experience with it, I am aware of how fast the time complexity of this structure is. The structure usually has a complexity of O(1) and it can be seen in a lot of the figures on the following pages. Every now and then, there can be some collisioning which then causes a spike in the time. Overall, it seemed to me that the chaining method was the most effective in showing O(1) behavior.

Linear Probing

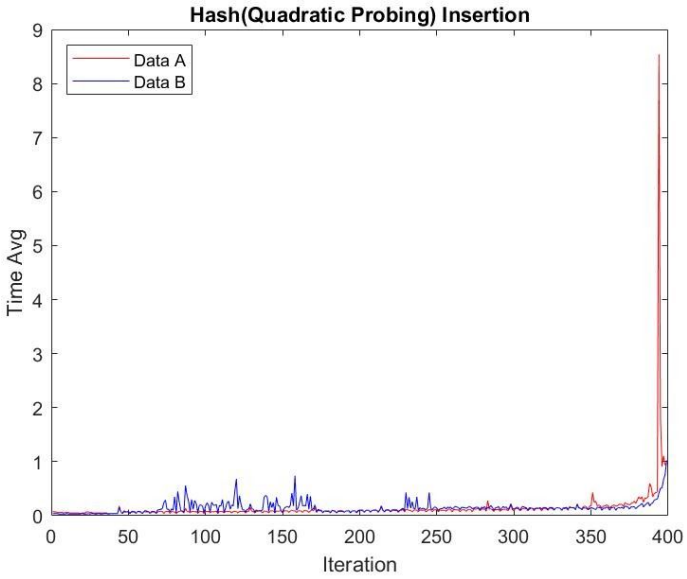


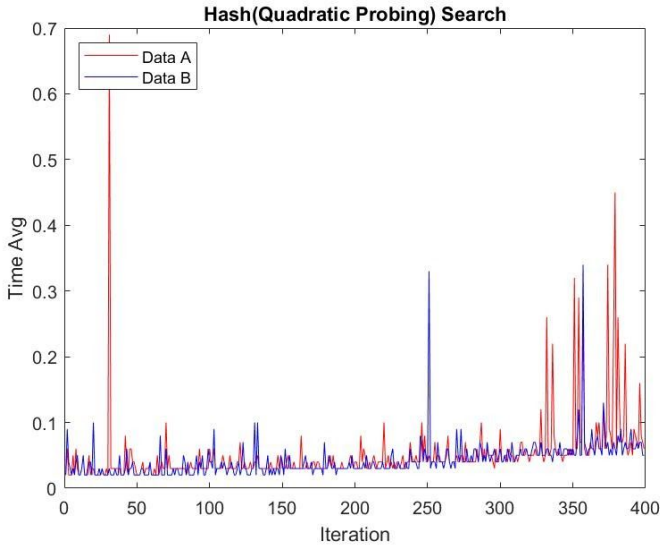


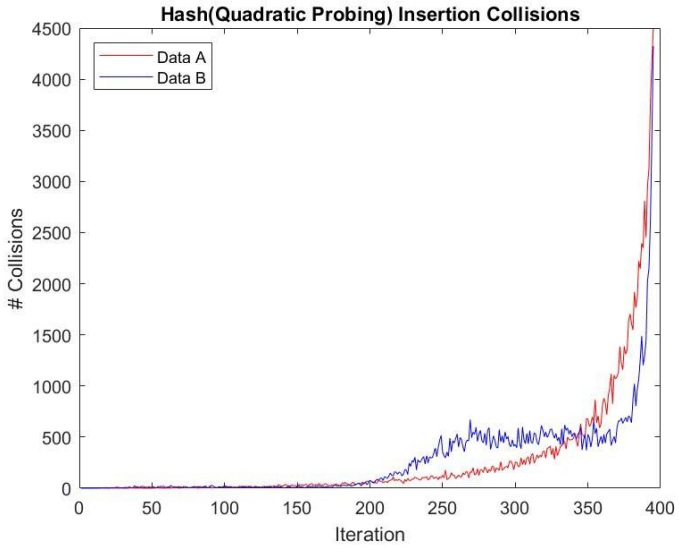


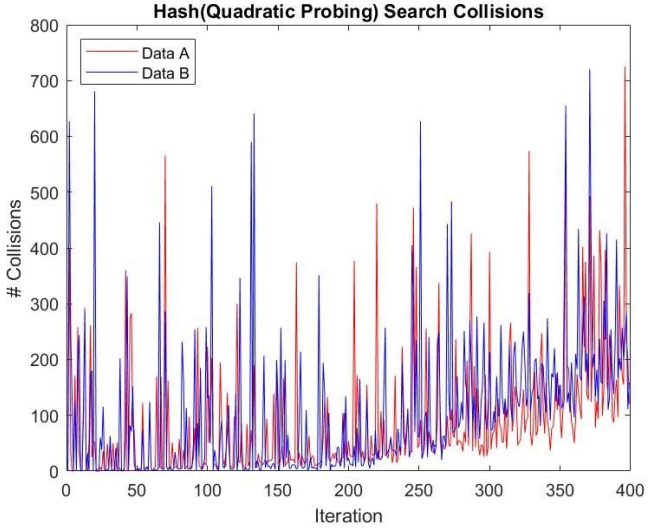


Quadratic Probing

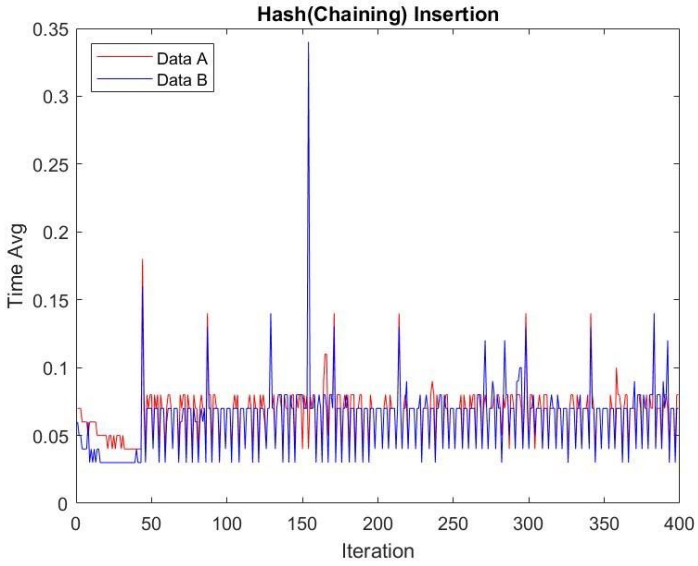


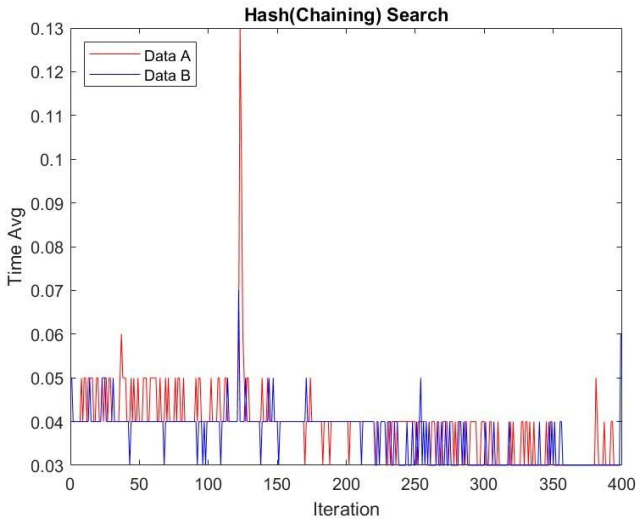


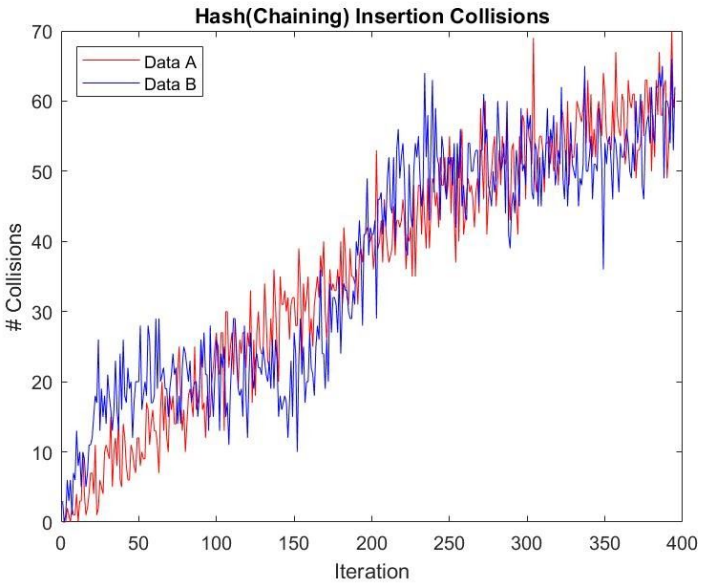


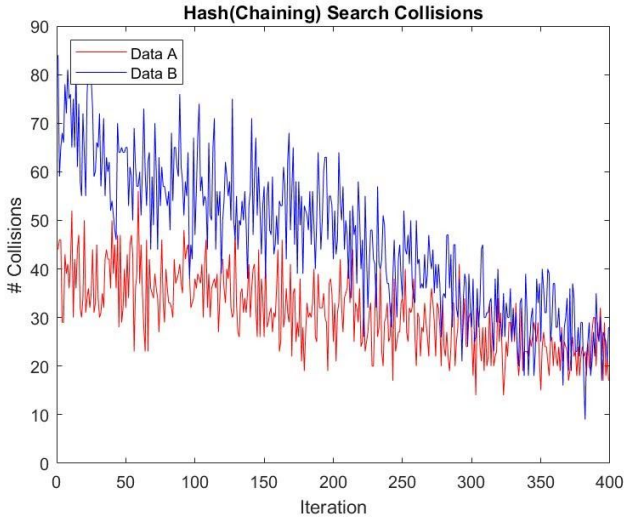


Chaining


Hash(Chaining) Insertion
Data A
Data B
Time Avg
Iteration
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0
50
100
150
200
250
300
350
400


Hash(Chaining) Search
Data A
Data B
Time Avg
Iteration
0.13
0.12
0.11
0.1
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0
50
100
150
200
250
300
350
400


Hash(Chaining) Insertion Collisions
Data A
Data B
Collisions
Iteration
70
60
50
40
30
20
10
0
0
50
100
150
200
250
300
350
400


Hash(Chaining) Search Collisions
Data A
Data B
Collisions
Iteration
90
80
70
60
50
40
30
20
10
0
0
50
100
150
200
250
300
350
400

Conclusion

Below I have plotted all the top time complexities for each structure (LL, BST, Hash Chaining). I had to zoom into the plot just to be able to see the times for Hashing and BST since they're largely smaller than LL. After analyzing them compared to each other, I have determined that Hashing, in my case with chaining, seems to be the best structure for this use case.