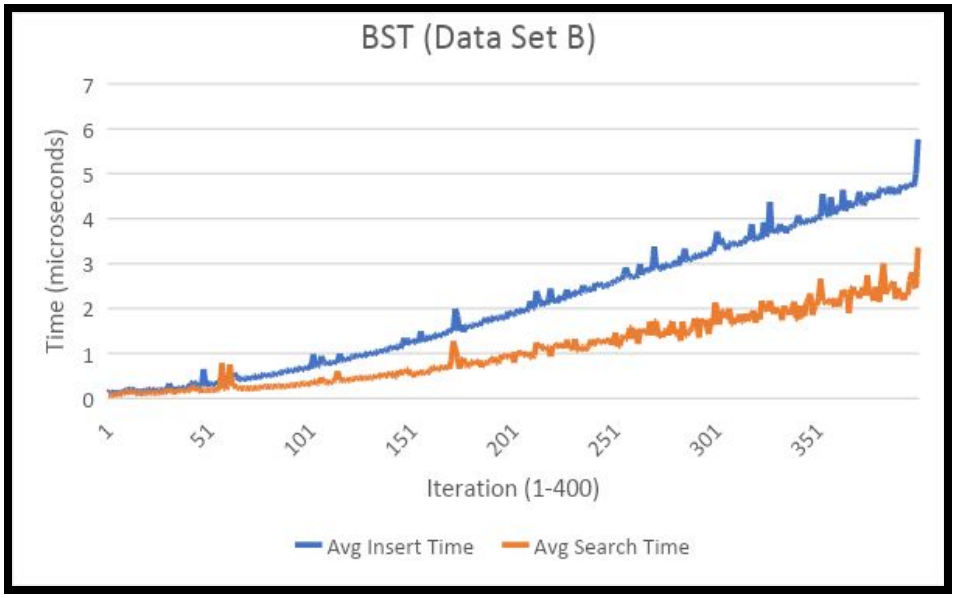
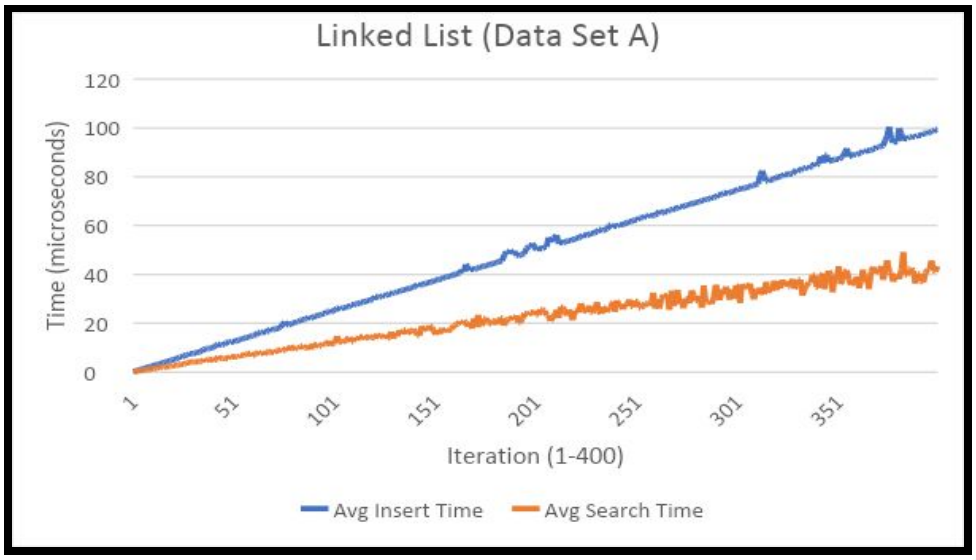
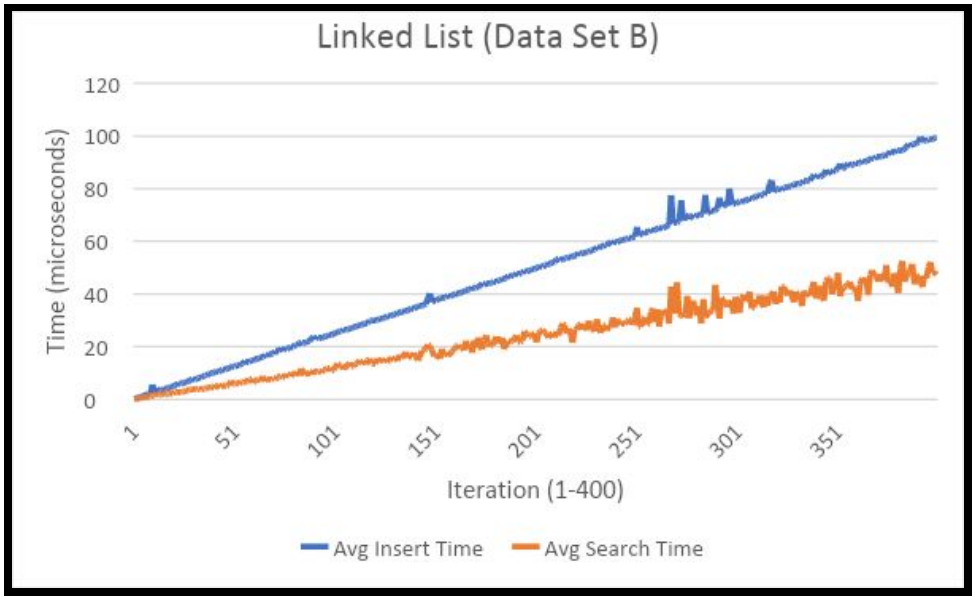
Chirag Telang
Matt King
CSCI 2270
29 April 2020

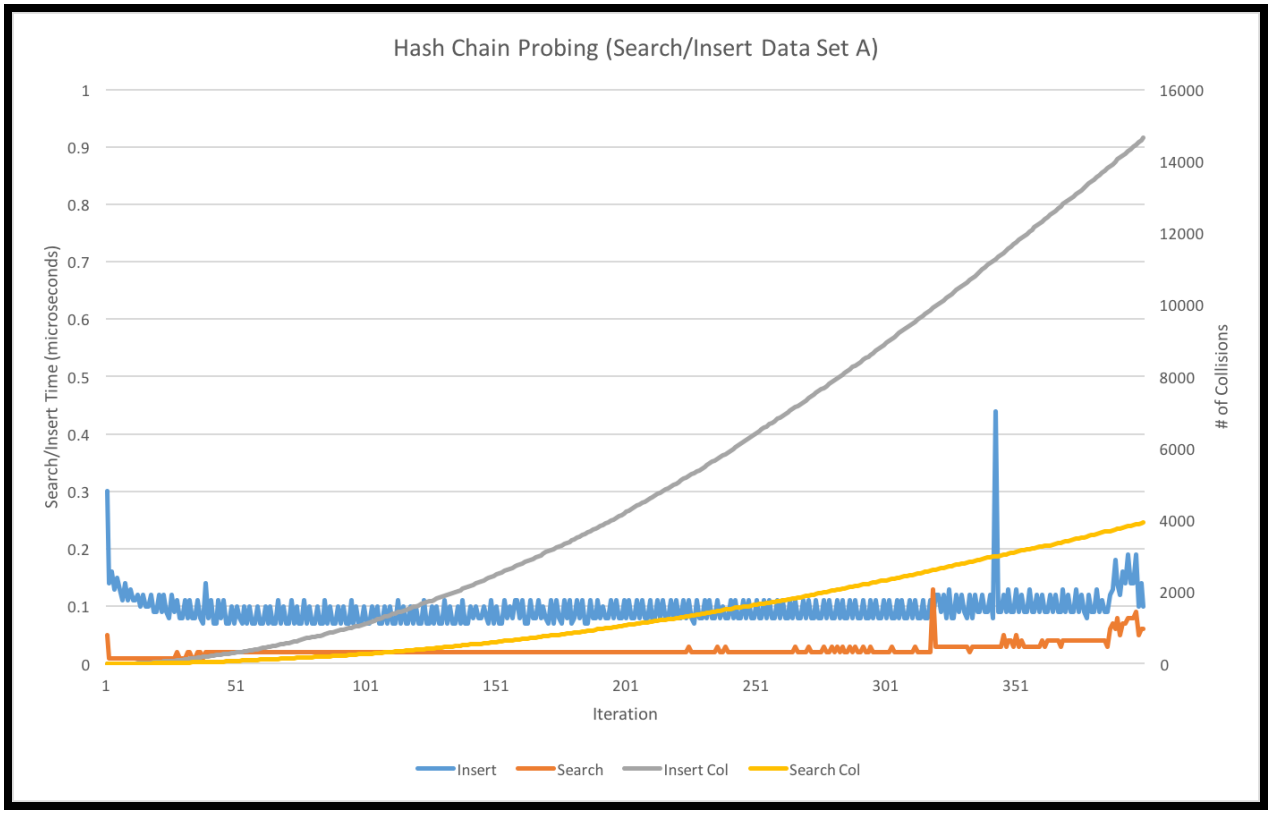
CSCI-2270 Final Project Report (Saving the USPS)

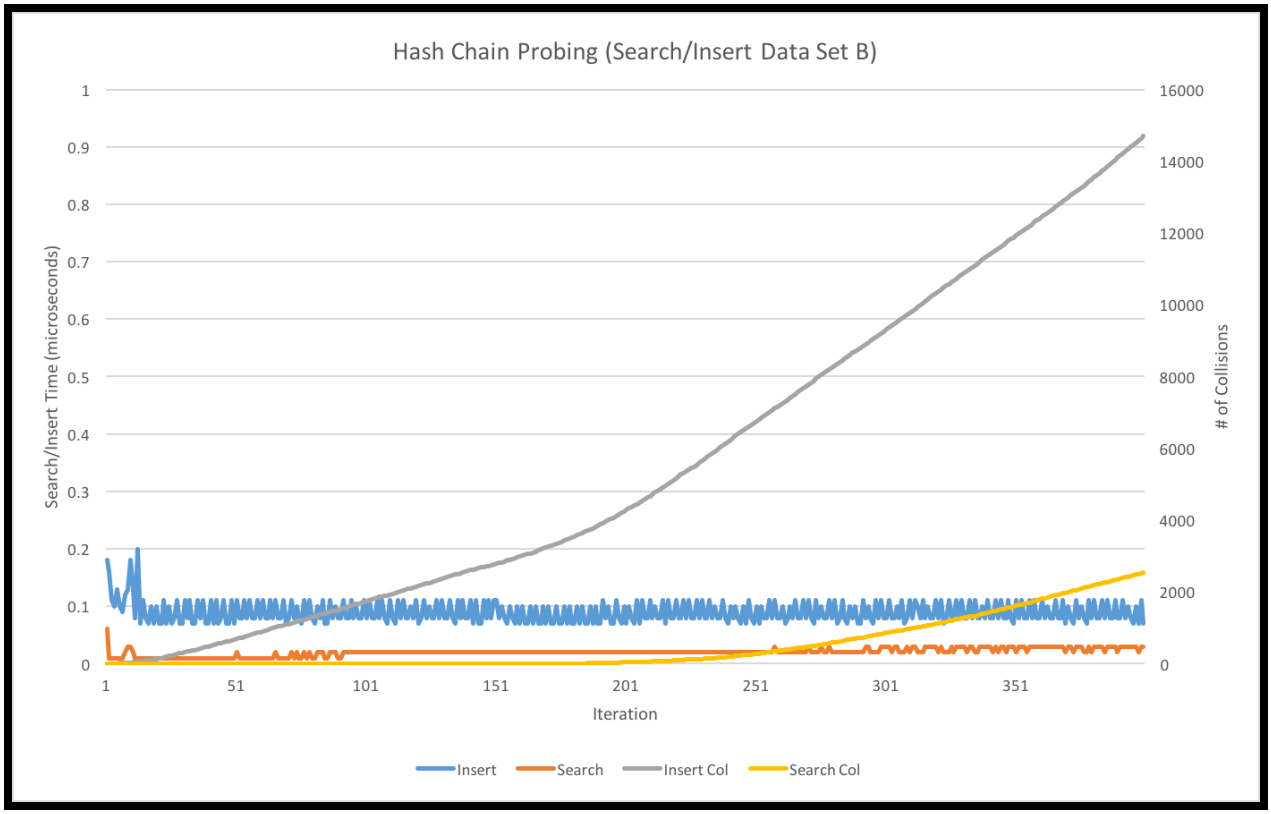
To determine the optimal data structure for the USPS to use, we calculated and compared the average insertion and search time for linked lists, binary search trees, and hash tables. To calculate the average time, we used the C++ Chrono library to measure the time to insert 100 elements. This number was then divided by 100 and added to an array for future reference. This was repeated 400 times, and each time 100 new elements were added to the data structure. This allows us to see the performance of search and inserting into the data structure while its size grows.

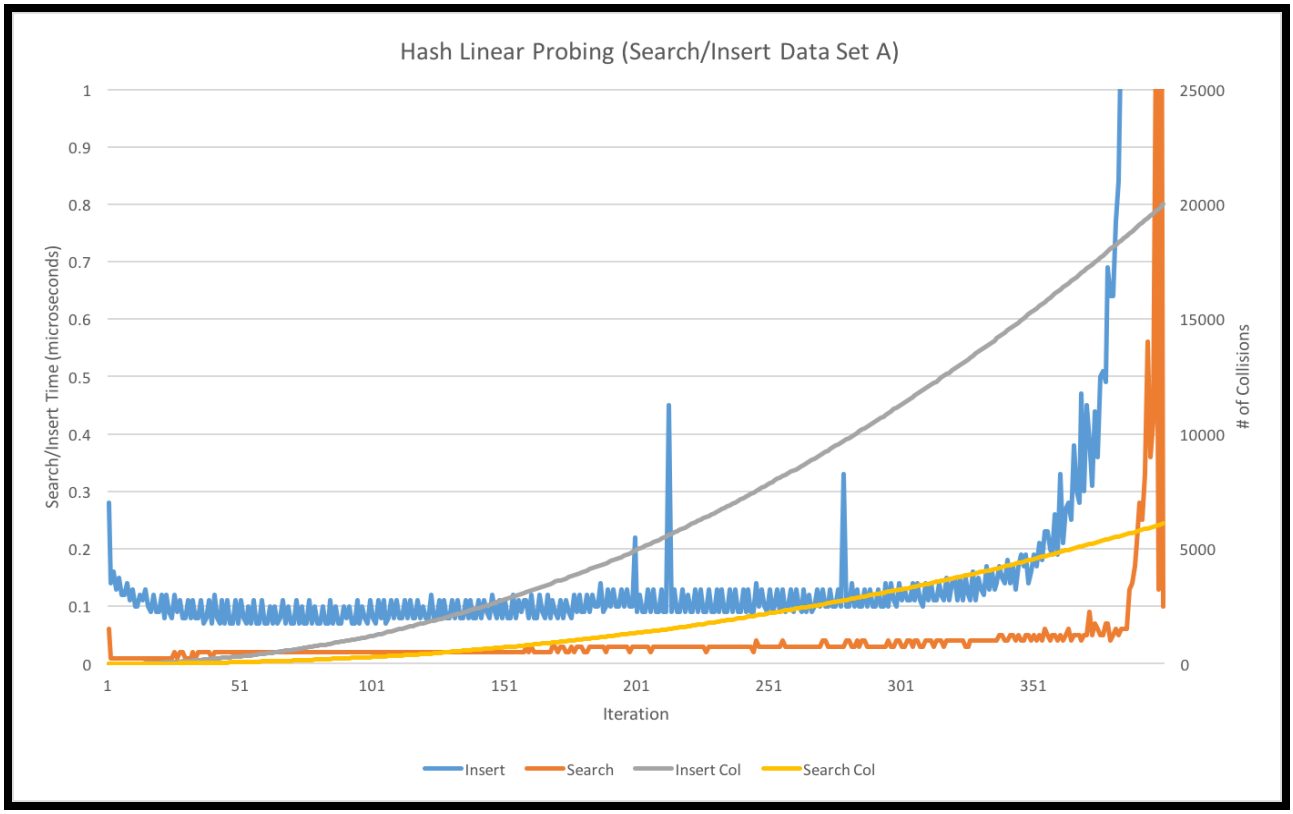


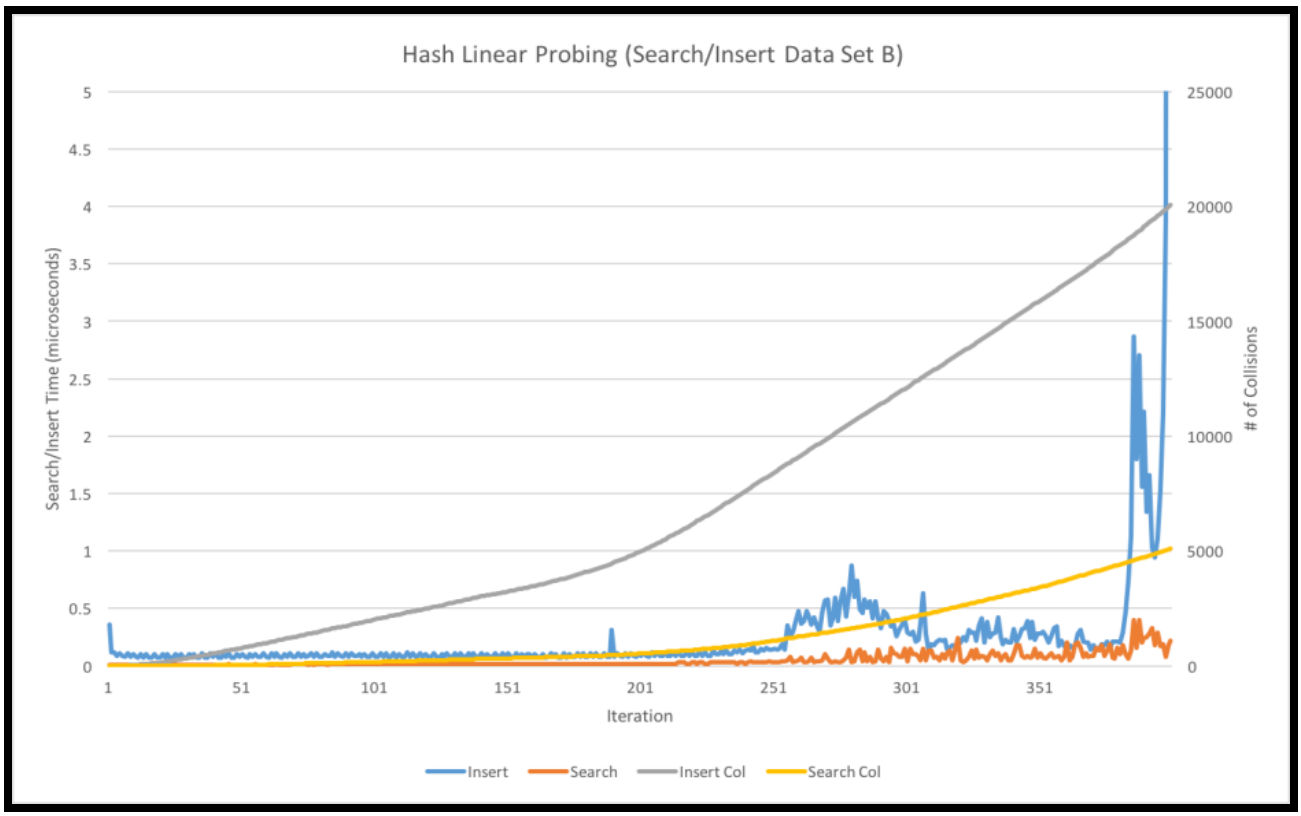

Linked List (Data Set A)
Time (microseconds)
120
100
80
60
40
20
0
1
51
101
151
201
251
301
351
Iteration (1-400)
Avg Insert Time
Avg Search Time

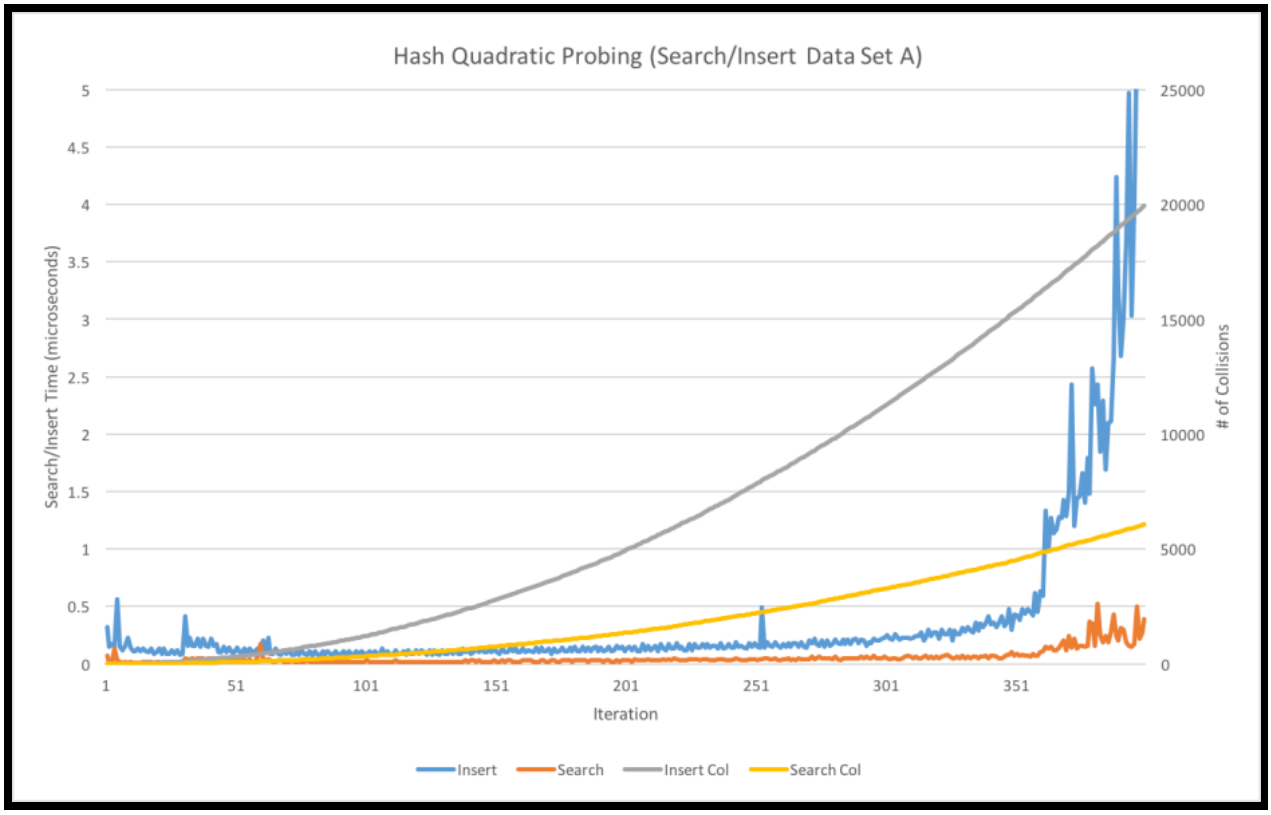

Linked List (Data Set B)
Time (microseconds)
120
100
80
60
40
20
0
1
51
101
151
201
251
301
351
Iteration (1-400)
Avg Insert Time
Avg Search Time

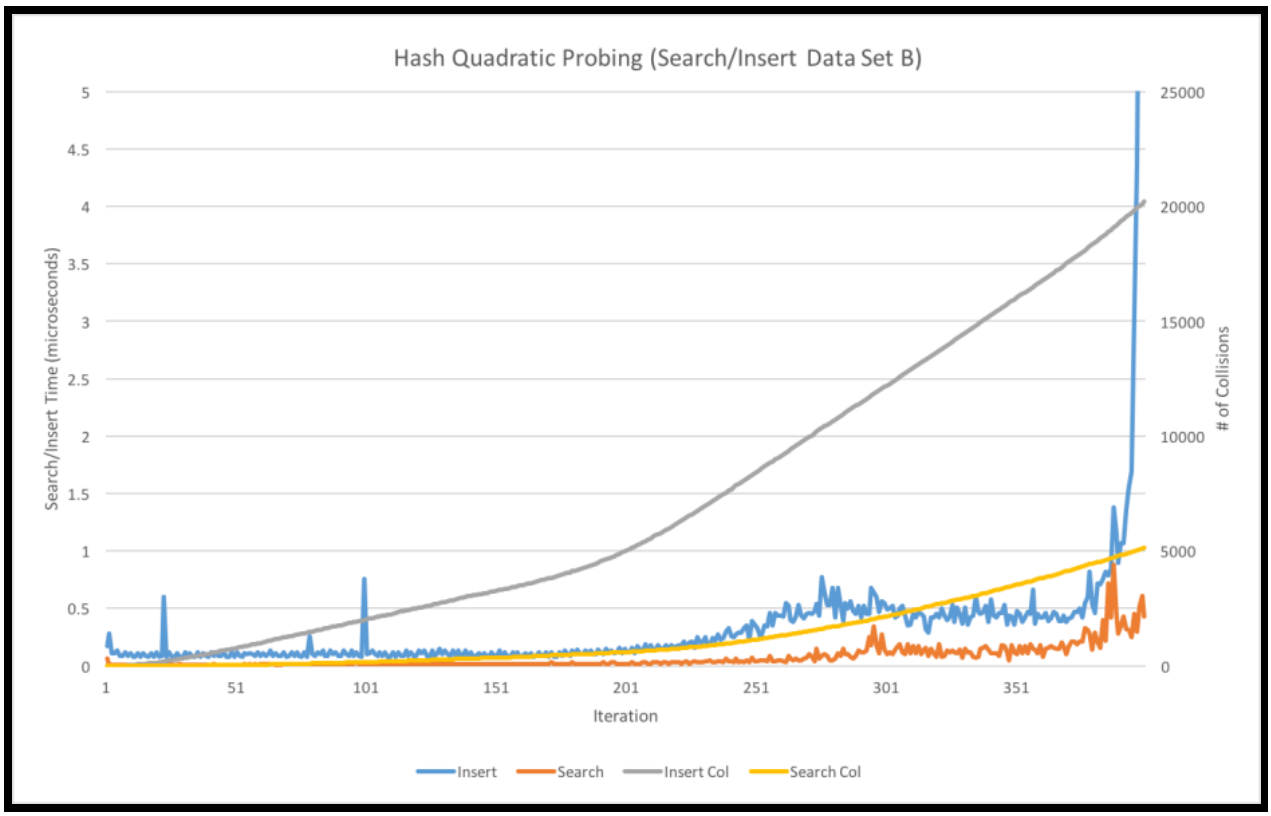

Hash Chain Probing (Search/Insert Data Set A)
Search/Insert Time (microseconds)
# of Collisions
Iteration
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
16000
14000
12000
10000
8000
6000
4000
2000
0
1
51
101
151
201
251
301
351
Insert
Search
Insert Col
Search Col

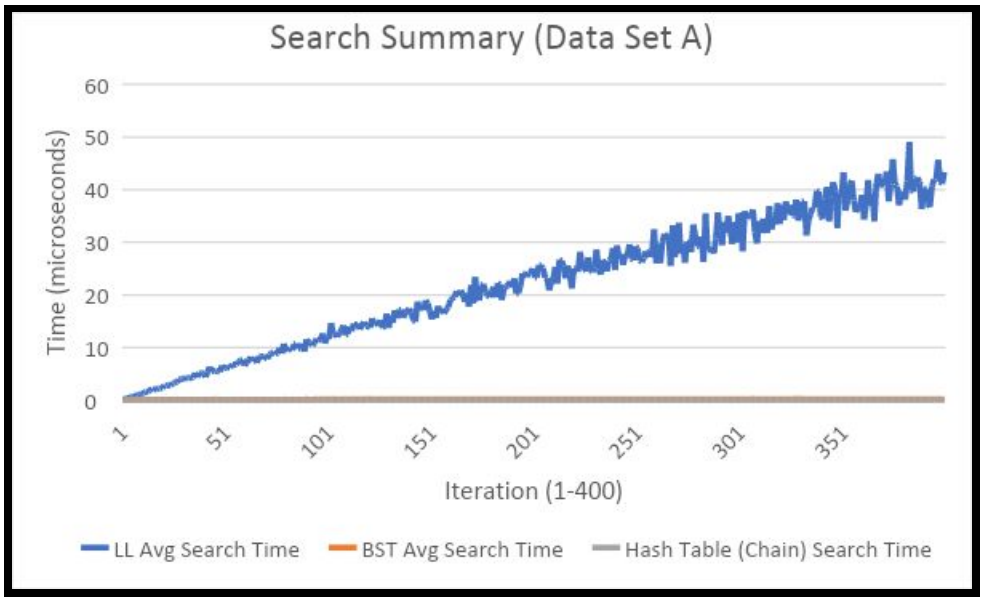

Hash Chain Probing (Search/Insert Data Set B)
Search/Insert Time (microseconds)
# of Collisions
Iteration
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
16000
14000
12000
10000
8000
6000
4000
2000
0
1
51
101
151
201
251
301
351
Insert
Search
Insert Col
Search Col

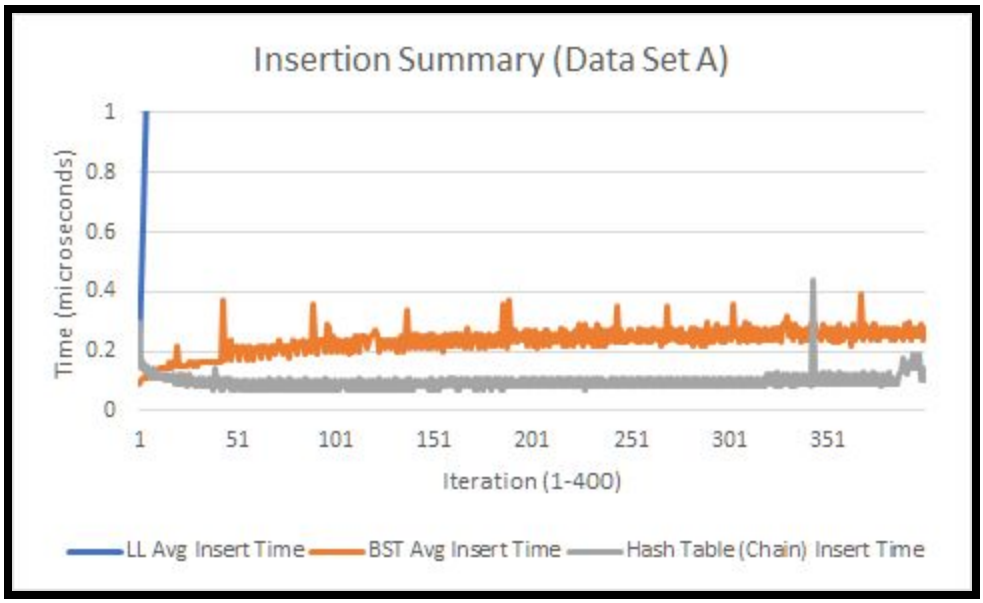

Hash Linear Probing (Search/Insert Data Set A)
Search/Insert Time (microseconds)
# of Collisions
Iteration
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
25000
20000
15000
10000
5000
0
1
51
101
151
201
251
301
351
Insert
Search
Insert Col
Search Col

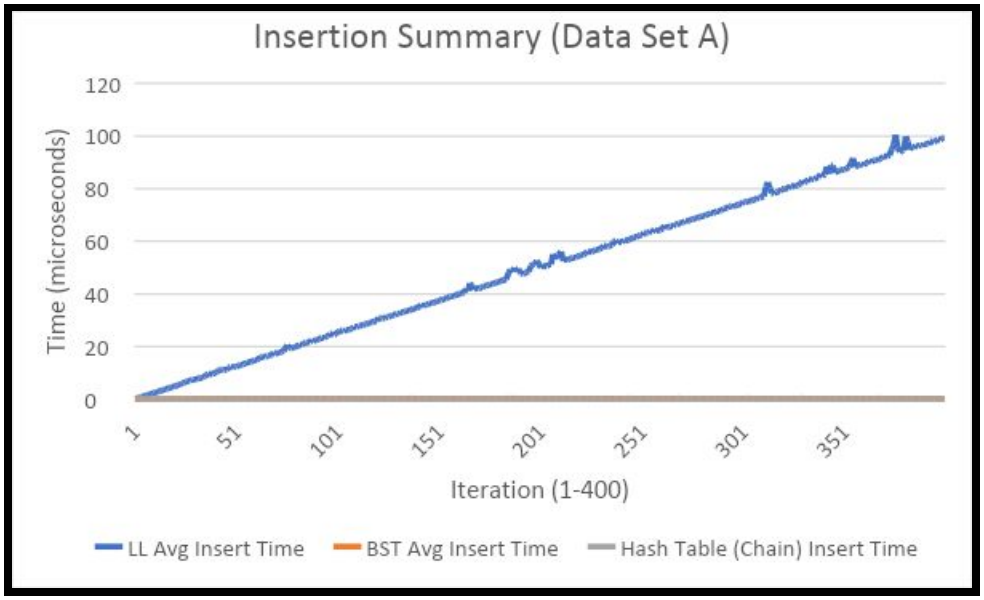

Hash Linear Probing (Search/Insert Data Set B)
Search/Insert Time (microseconds)
# of Collisions
Iteration
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
25000
20000
15000
10000
5000
0
1
51
101
151
201
251
301
351
Insert
Search
Insert Col
Search Col

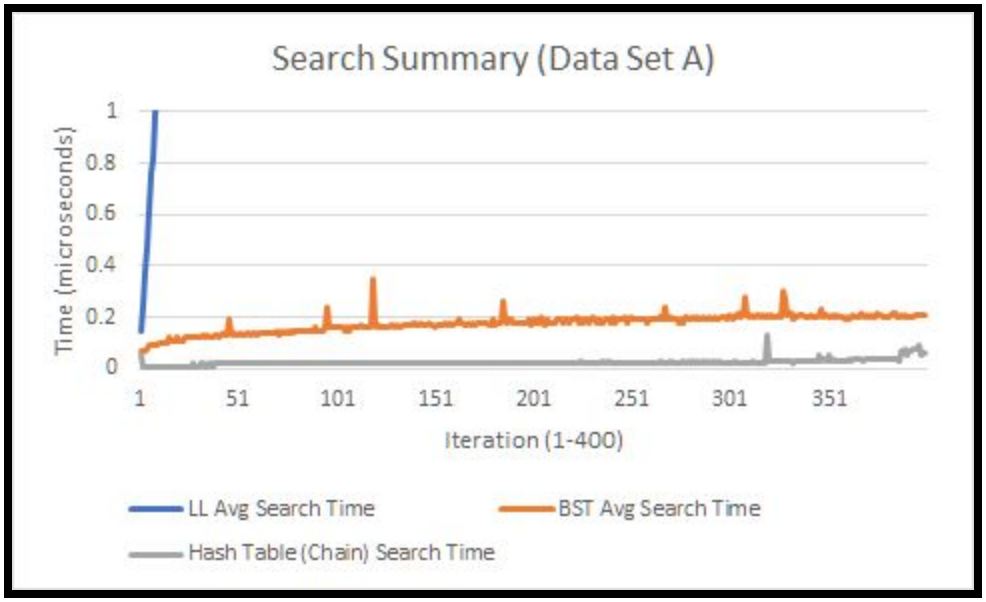

Hash Quadratic Probing (Search/Insert Data Set A)
Search/Insert Time (microseconds)
# of Collisions
Iteration
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
5000
10000
15000
20000
25000
1
51
101
151
201
251
301
351
Insert
Search
Insert Col
Search Col


Hash Quadratic Probing (Search/Insert Data Set B)
Search/Insert Time (microseconds)
# of Collisions
Iteration
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
0
5000
10000
15000
20000
25000
1
51
101
151
201
251
301
351
Insert
Search
Insert Col
Search Col


Search Summary (Data Set A)
Time (microseconds)
60
50
40
30
20
10
0
1
51
101
151
201
251
301
351
Iteration (1-400)
LL Avg Search Time
BST Avg Search Time
Hash Table (Chain) Search Time


Insertion Summary (Data Set A)
Time (microseconds)
1
0.8
0.6
0.4
0.2
0
1
51
101
151
201
251
301
351
Iteration (1-400)
LL Avg Insert Time
BST Avg Insert Time
Hash Table (Chain) Insert Time


Insertion Summary (Data Set A)
Time (microseconds)
120
100
80
60
40
20
0
1
51
101
151
201
251
301
351
Iteration (1-400)
LL Avg Insert Time
BST Avg Insert Time
Hash Table (Chain) Insert Time


Search Summary (Data Set A)
Time (microseconds)
1
0.8
0.6
0.4
0.2
0
1
51
101
151
201
251
301
351
Iteration (1-400)
LL Avg Search Time
BST Avg Search Time
Hash Table (Chain) Search Time

For the linked list, our implementation did not store the last node in the linked list. For this reason, insertion into the linked list has time complexity of O(n) because every node must be traversed to reach the last node. Searching in a linked list is also O(n) because the list must be traversed blindly until the right key is found. This is not optimal time complexity. Binary search trees have search and insertion time of O(logn) because we can make comparisons between the current node and the sought node to determine whether to go to the left or right child. This only holds if the tree is balanced.

Binary search trees can have a worse case search and insertion time of O(n) because a very unbalanced binary search tree resembles a linked list. This could explain why the time complexity of the binary search tree looks like O(logn) for set A, but O(n) for set B. Set B could be less random, leading to an unbalanced binary search tree. The hash table performed well for both data sets. However, toward the end, the time to search or insert an element using linear or quadratic probing sometimes spiked. This could be due to most of the table being full, so the collision resolution must probe for a while. In the chaining implementation of the hash table, the time spike did not appear at the end. This could be because the chaining method does not need to probe; it just appends to the linked list.

In summary, the USPS should implement a hash table with chaining to resolve collisions. Ideally, a hash table has insertion and search times of O(1). This is better than BSTs and linked lists. However, with collisions, probing can make search and insertion times longer. With this size of data, it seems like using chaining for collision resolution led to the best time complexity.