Zoe Stewart and Zoe Cruse
CSCI 2270 Final Project
29 April 2020

Throughout the process of this final project, we were able to gain a better understanding of time complexities for a range of different data structures. This allowed us to recognize that data structures can be implemented in various ways that are more efficient, depending on the data that is being evaluated. In this project the data structures tested were a linked list, binary search tree, and a hash table. For the hash table we tested three different methods of collision resolution: chaining with a linked list and linear and quadratic probing through open addressing.

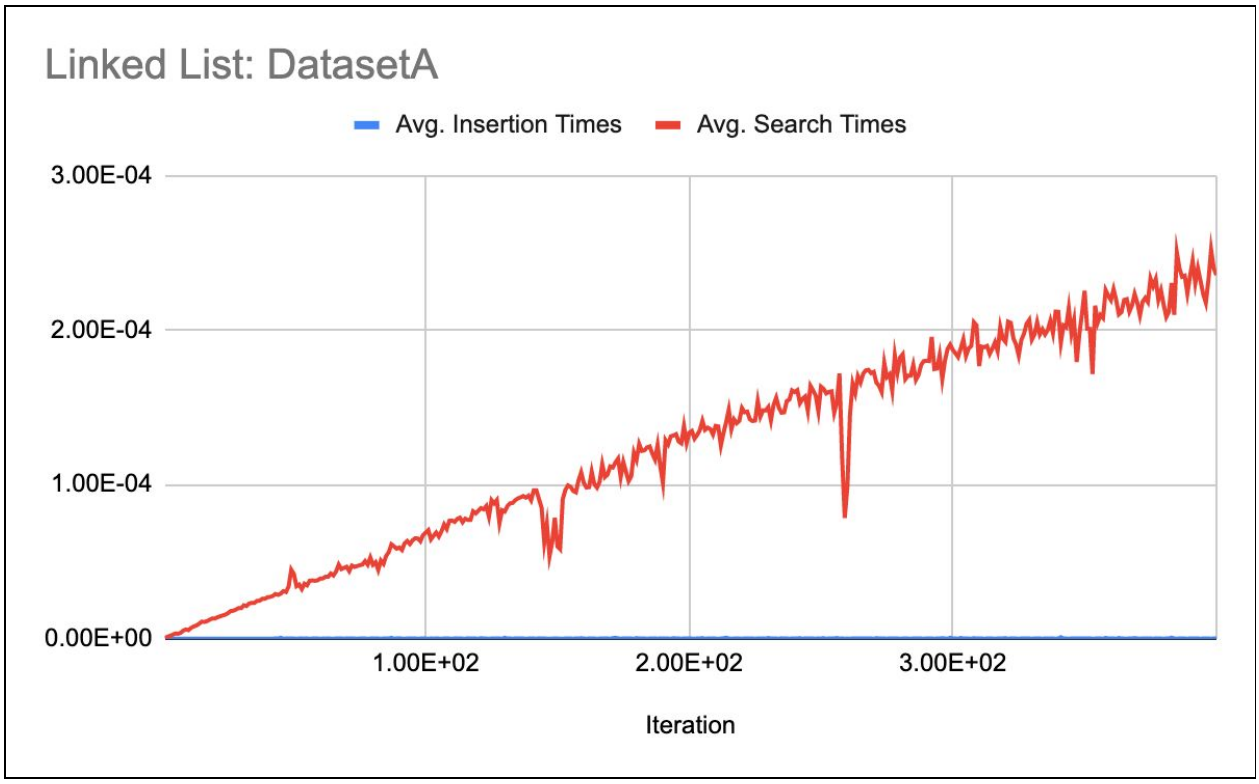
After running through the two datasets we found that the open address hash table with quadratic probing was the most efficient structure because it was able to resolve collisions and insert and search with the lowest times. In the assignment we were given data of a determined size to evaluate, which meant that quadratic probing would be most effective. This type of probing allows us to skip regions of the hash table with possible clusters and more efficiently insert and search for data.

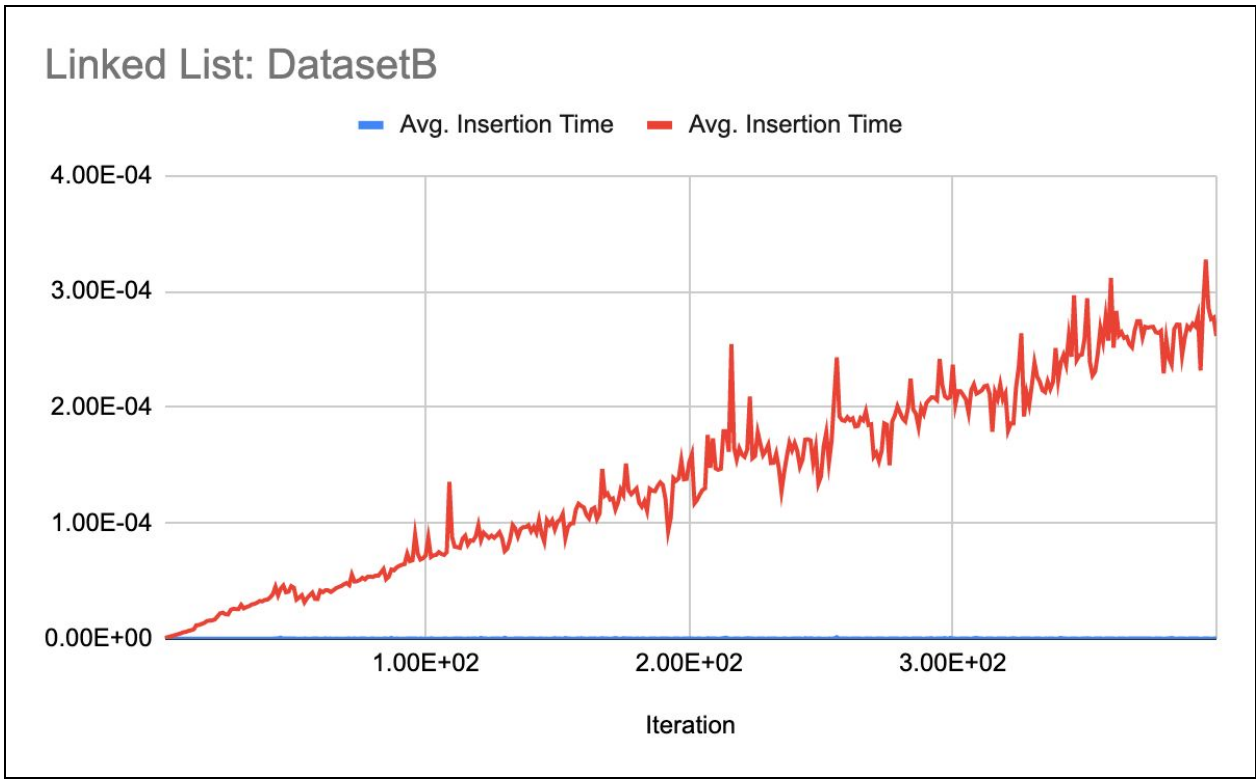
Overall, we found that hash tables were the most effective structure for our use in this situation. However, if we needed to sort the data, a binary search tree or linked list may be a much more powerful resource. For inserting especially, the hash table with chaining showed to be the most efficient at inserting the packing applications with a specifically sized dataset. Since we don't know how frequently the keys may need to be inserted or deleted, this method would be helpful if the amount of data to be inserted was unknown - since the table cannot "run out of room" in the same way other hash tables could. Although a table with open addressing would perform a better cache performance as everything is stored within the same table, this format allows for more flexibility which might be necessary in various given situations.

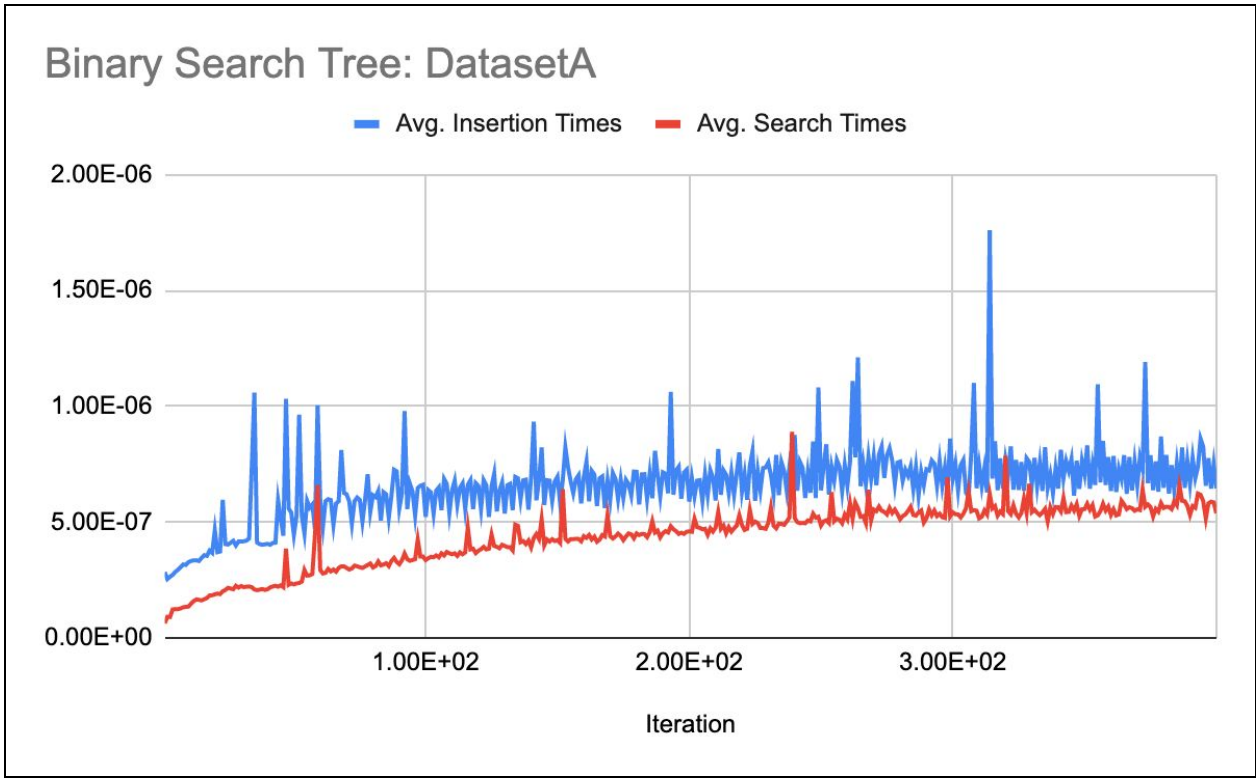
Although those were our results from the data structures we specifically tested, a form of double hashing -where we would apply a second hash function where each collision occurs- would probably be more efficient than all the rest. The logic behind this structure would be that the

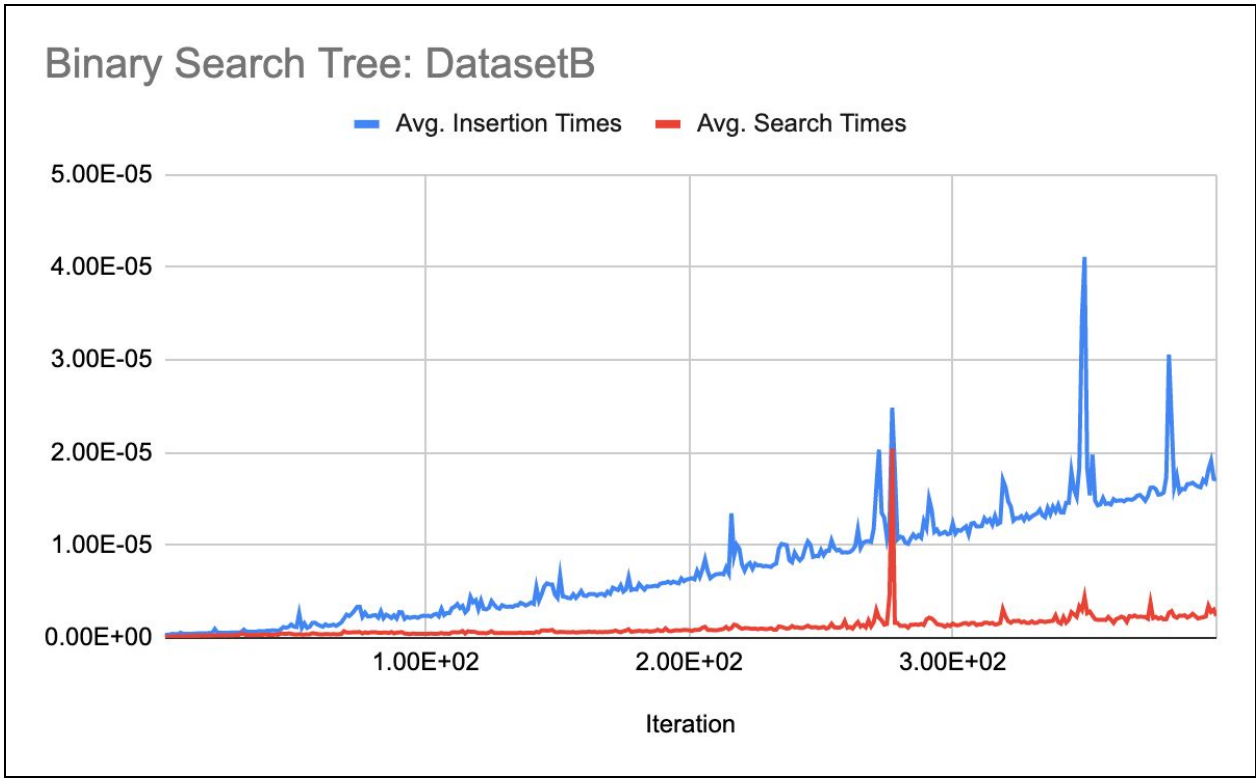
second hash function would be the number of positions until the final insert. The probing would be computed as fi(x)= (f(x)+if'(x))%z (where f(x) is the original function, f'(x) is the following function, z is the table size, and i is the number of collisions generated). Although we chose quadratic probing as the most efficient form, it also introduces the issue of secondary clustering that would be completely resolved with double hashing, causing double hashing to be the most efficient structure. Unfortunately, we did not have the time to implement this in code since we ran out of time, we could only explain the logistics and give the computation that would be implemented into the code (replacing other probing in the same overall hash function).

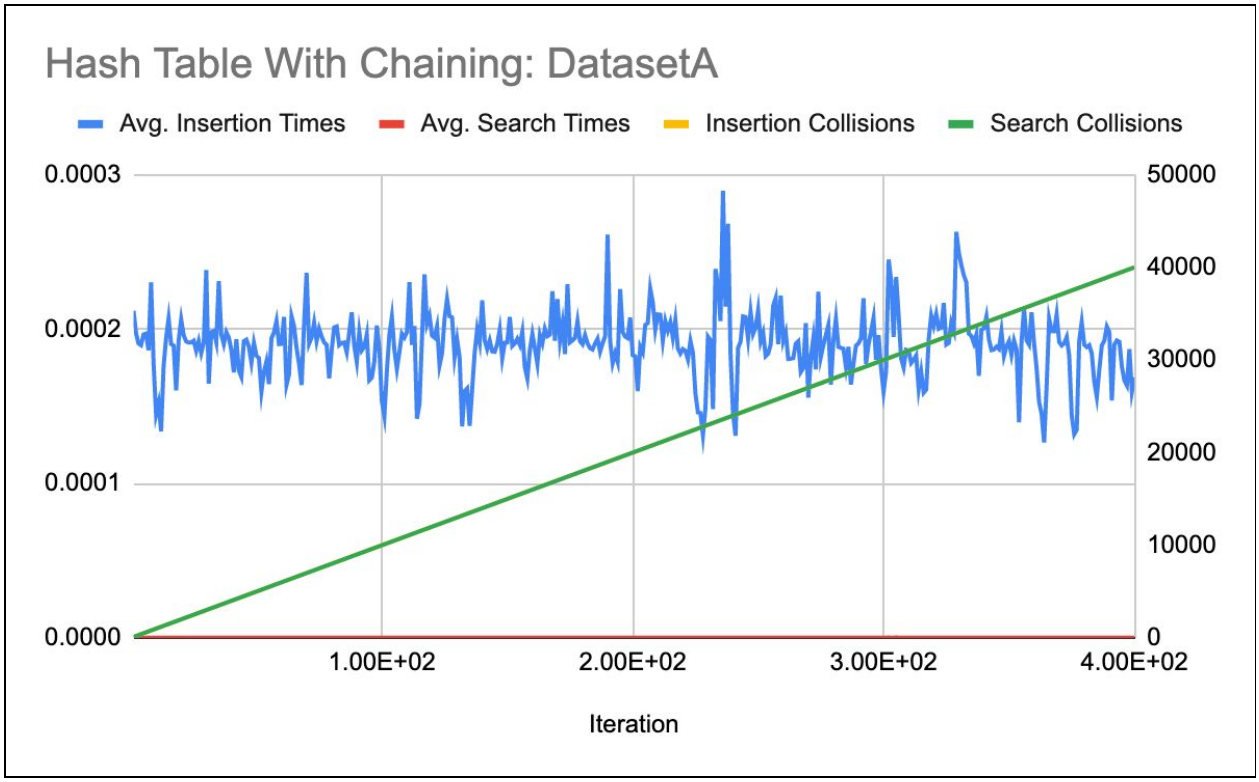
Graphical Data Collected:

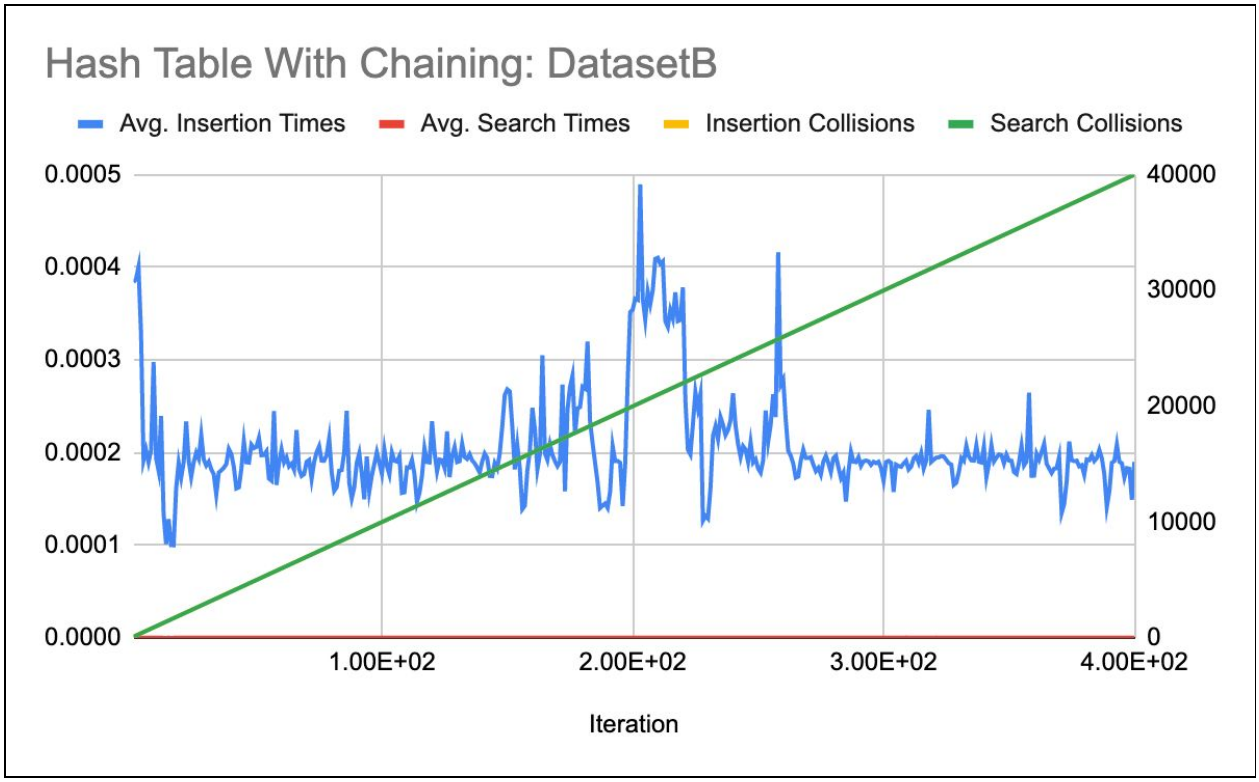


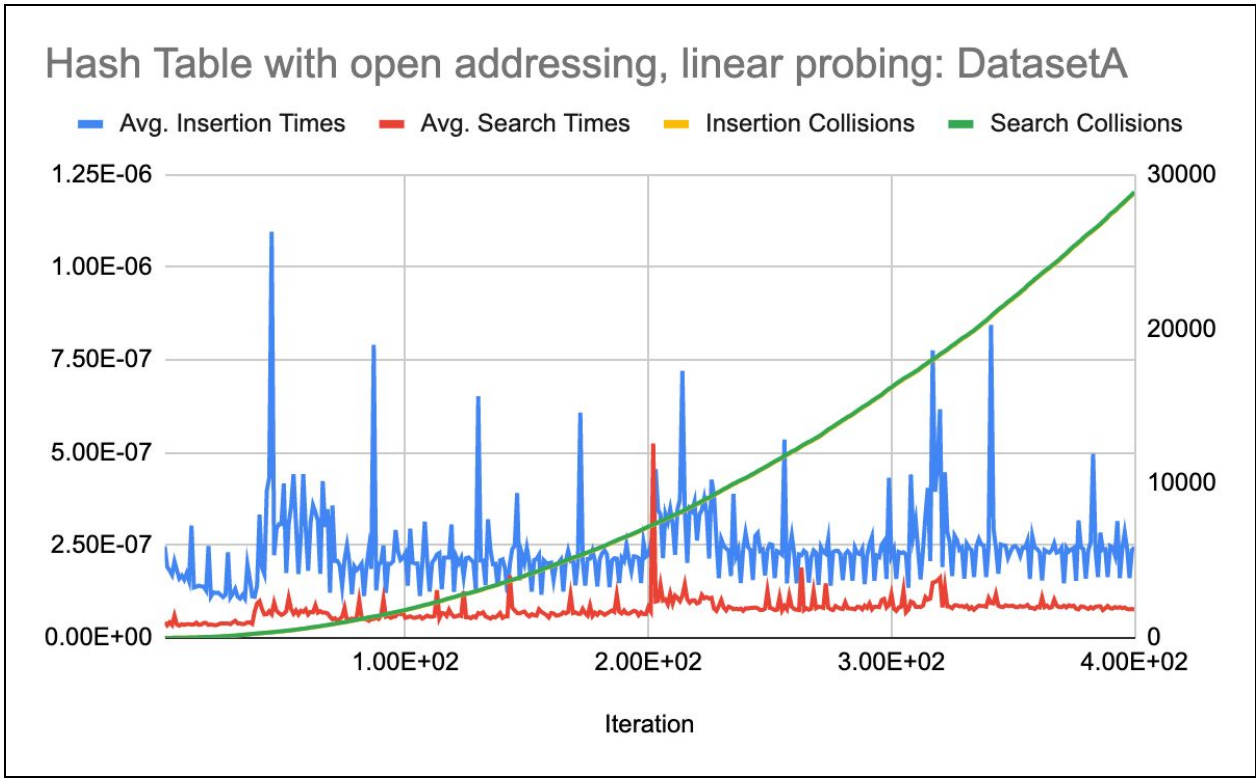


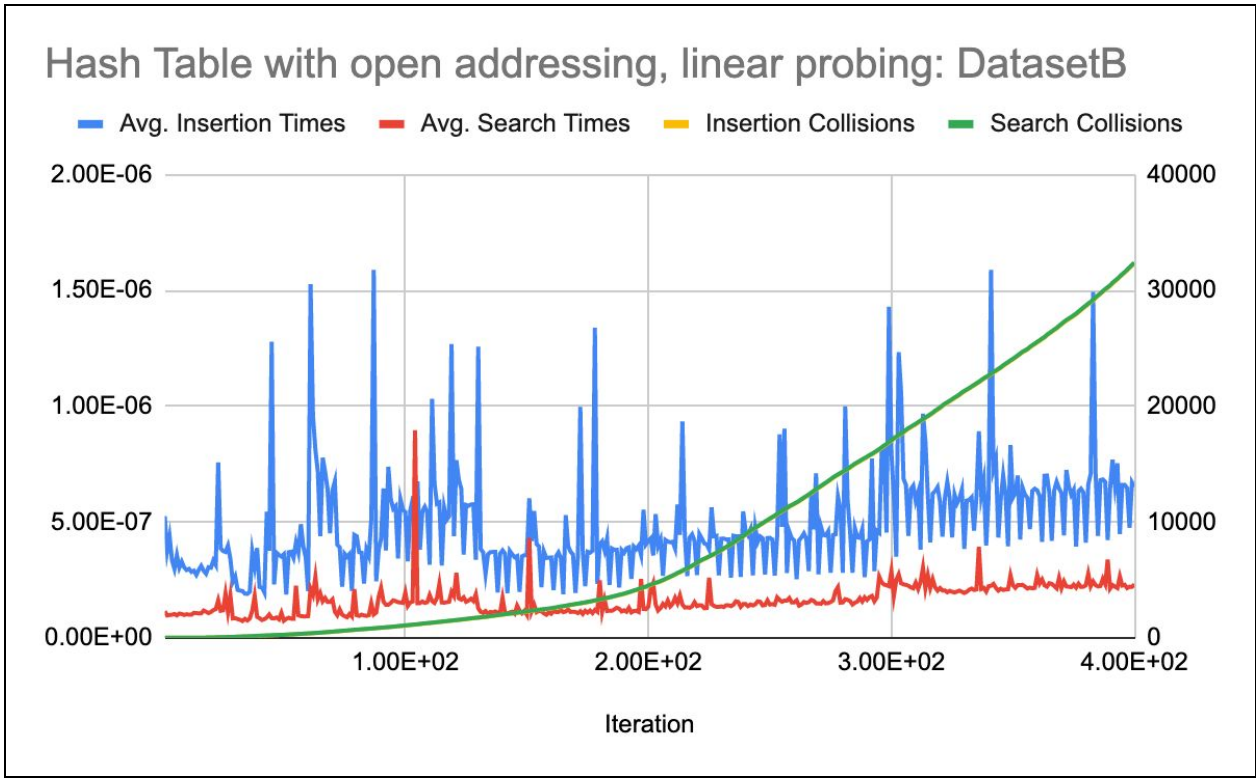

Binary Search Tree: DatasetA
Avg. Insertion Times
Avg. Search Times
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
1.00E+02
2.00E+02
3.00E+02
Iteration

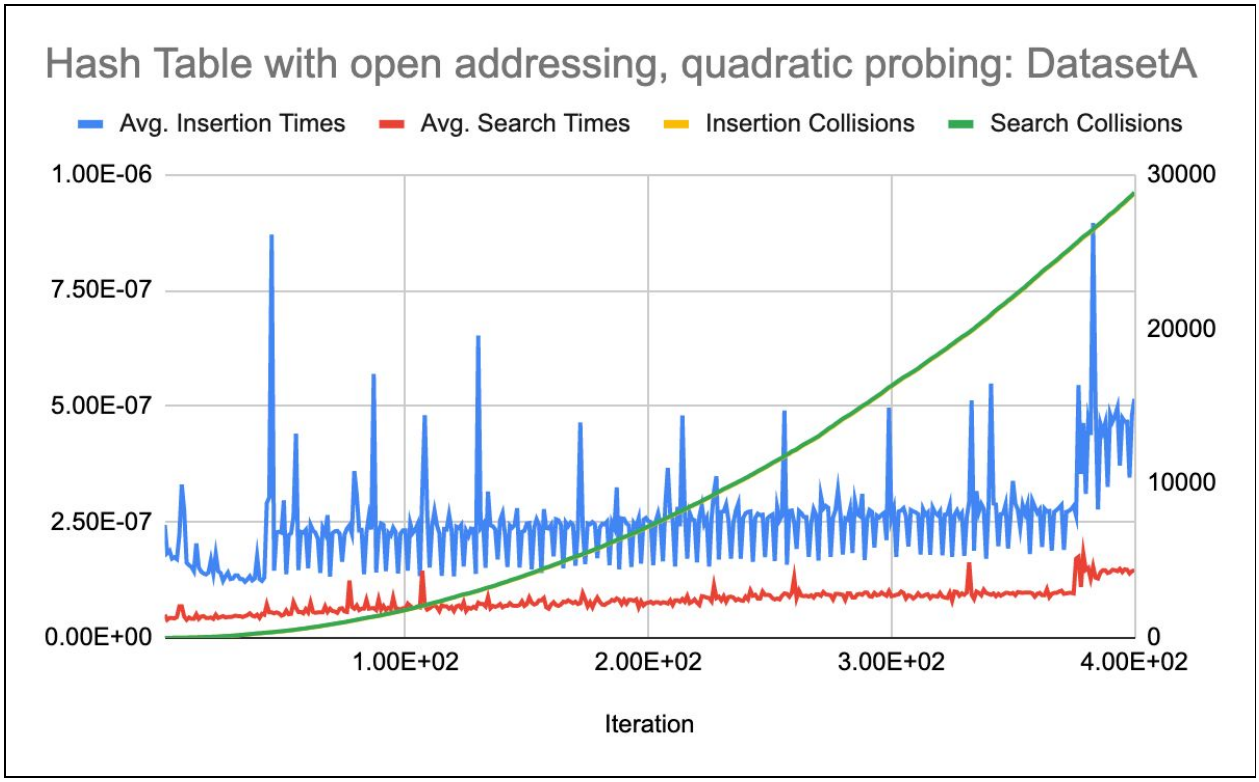

Binary Search Tree: DatasetB
Avg. Insertion Times
Avg. Search Times
5.00E-05
4.00E-05
3.00E-05
2.00E-05
1.00E-05
0.00E+00
1.00E+02
2.00E+02
3.00E+02
Iteration

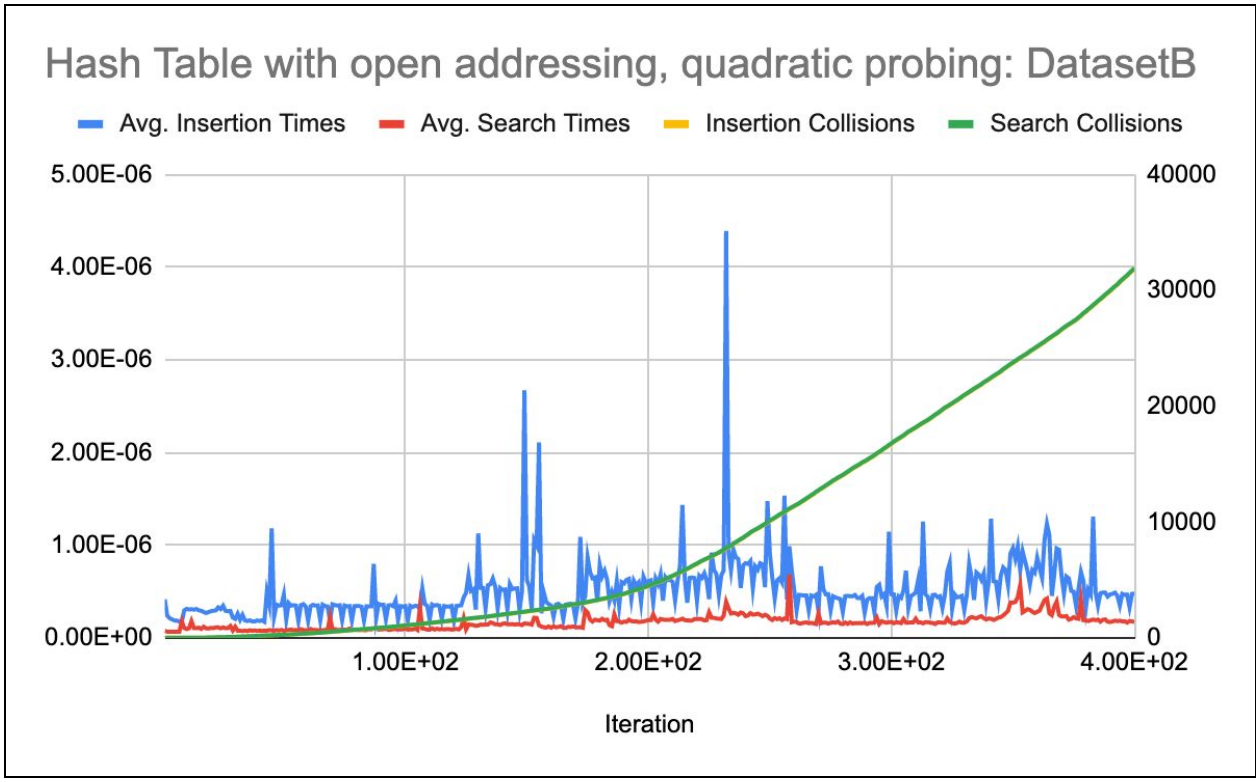

Hash Table With Chaining: DatasetA
Avg. Insertion Times
Avg. Search Times
Insertion Collisions
Search Collisions
0.0003
0.0002
0.0001
0.0000
50000
40000
30000
20000
10000
0
1.00E+02
2.00E+02
3.00E+02
4.00E+02
Iteration

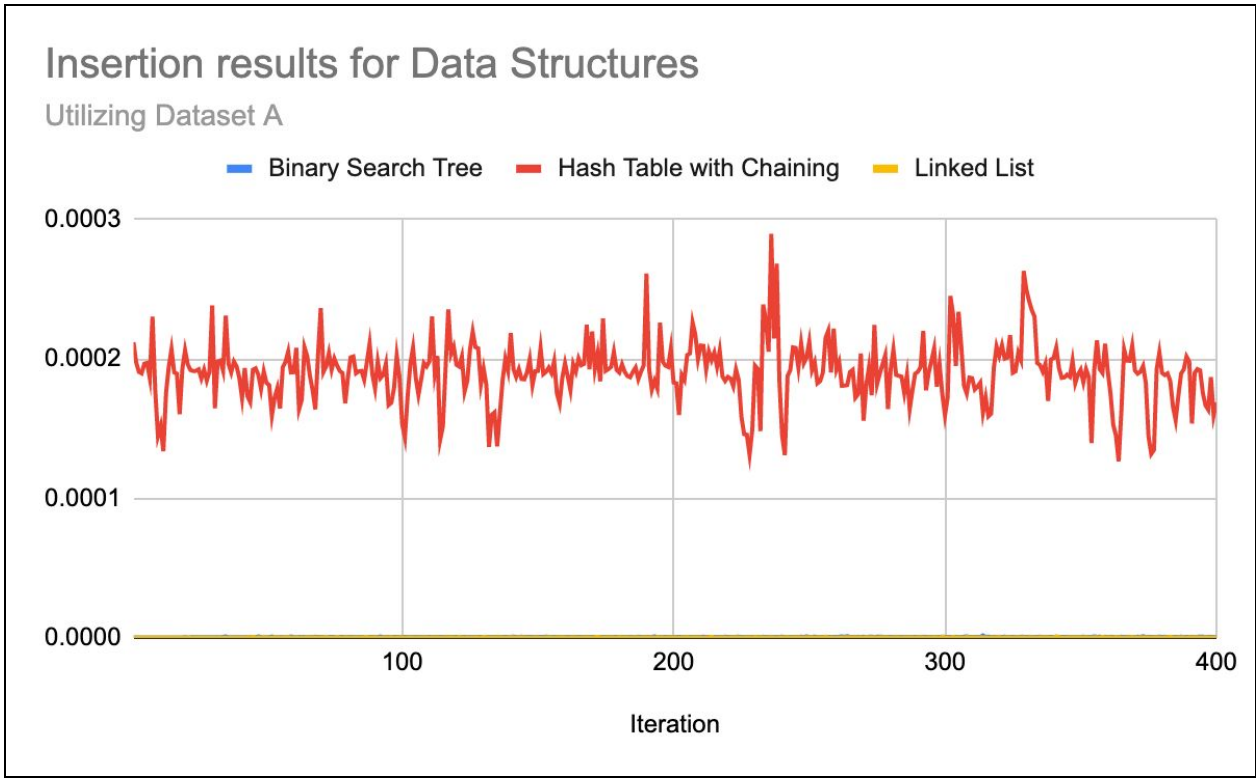

Hash Table With Chaining: DatasetB
Avg. Insertion Times
Avg. Search Times
Insertion Collisions
Search Collisions
0.0005
0.0004
0.0003
0.0002
0.0001
0.0000
40000
30000
20000
10000
0
1.00E+02
2.00E+02
3.00E+02
4.00E+02
Iteration

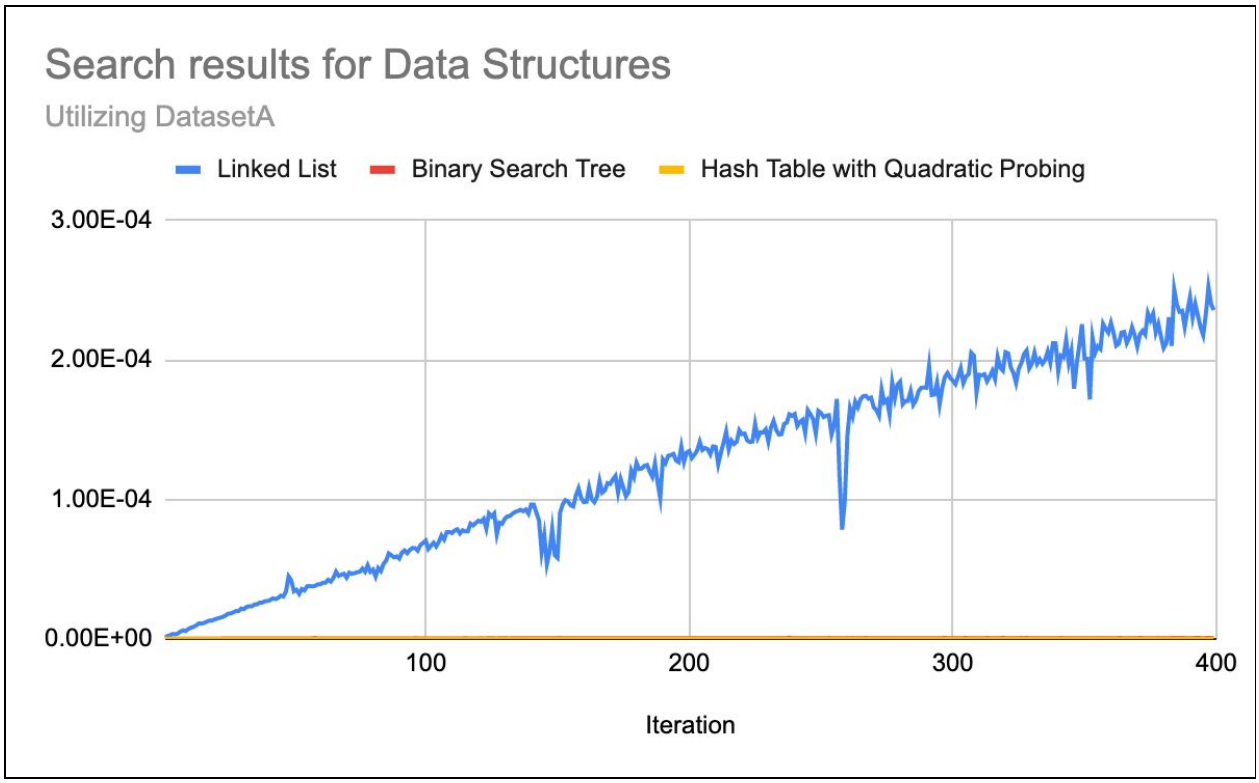

Hash Table with open addressing, linear probing: DatasetA
Avg. Insertion Times
Avg. Search Times
Insertion Collisions
Search Collisions
1.25E-06
1.00E-06
7.50E-07
5.00E-07
2.50E-07
0.00E+00
30000
20000
10000
0
1.00E+02
2.00E+02
3.00E+02
4.00E+02
Iteration


Hash Table with open addressing, linear probing: DatasetB
Avg. Insertion Times
Avg. Search Times
Insertion Collisions
Search Collisions
2.00E-06
1.50E-06
1.00E-06
5.00E-07
0.00E+00
40000
30000
20000
10000
0
1.00E+02
2.00E+02
3.00E+02
4.00E+02
Iteration


Hash Table with open addressing, quadratic probing: DatasetA
Avg. Insertion Times
Avg. Search Times
Insertion Collisions
Search Collisions
1.00E-06
7.50E-07
5.00E-07
2.50E-07
0.00E+00
30000
20000
10000
0
1.00E+02
2.00E+02
3.00E+02
4.00E+02
Iteration


Hash Table with open addressing, quadratic probing: DatasetB
Avg. Insertion Times
Avg. Search Times
Insertion Collisions
Search Collisions
5.00E-06
4.00E-06
3.00E-06
2.00E-06
1.00E-06
0.00E+00
40000
30000
20000
10000
0
1.00E+02
2.00E+02
3.00E+02
4.00E+02
Iteration


Insertion results for Data Structures
Utilizing Dataset A
Binary Search Tree
Hash Table with Chaining
Linked List
0.0003
0.0002
0.0001
0.0000
100
200
300
400
Iteration


Search results for Data Structures
Utilizing DatasetA
Linked List
Binary Search Tree
Hash Table with Quadratic Probing
3.00E-04
2.00E-04
1.00E-04
0.00E+00
100
200
300
400
Iteration