CSCI 2270

Evaluation/Testing Method(s)

- The following tests/experiments were conducted on the selected data-structures in an effort to accurately evaluate the stats in each given scenario(s). Hence the "most" optimal one can be found and then proposed for adoption/implementation in the currently laden USPS® backend system[0].

	Practical Feature Test #1› File(s)	Practical Feature Test #2› File(s)	Practical Feature Test #1› File(s)	Practical Feature Test #1› File(s)
Linked-List	Insertion› DataSetA[1]	Search/Locate› DataSetA	Insertion› DataSetB[2]	Search/Locate› DataSetB
Binary Search-Tree	Insertion› DataSetA[1]	Search/Locate› DataSetA	Insertion› DataSetB[2]	Search/Locate› DataSetB
Hash Table - Chaining	Insertion› DataSetA[1]	Search/Locate› DataSetA	Insertion› DataSetB[2]	Search/Locate› DataSetB
Hash Table Open Address Linear Probing	Insertion› DataSetA[1]	Search/Locate› DataSetA	Insertion› DataSetB[2]	Search/Locate› DataSetB
Hash Table Open Address Quadratic Probing	Insertion› DataSetA[1]	Search/Locate› DataSetA	Insertion› DataSetB[2]	Search/Locate› DataSetB

Generalizations Taken to Improve Efficiency and Accuracy

- We utilized **nanoseconds** to measure elapsed time. This allowed us to get a better picture of the data's trend because we were able to see each data set's run time. We kept this consistent to allow for better data comparisons.

- We **filtered out duplicate data** in the insert function of all data structures. This was done in in order to make the search function more efficient. The time we spend searching in the insert function will improve the search function times. The searching done for the insert function was included in the runtime for the function as a whole to gather the most accurate data.

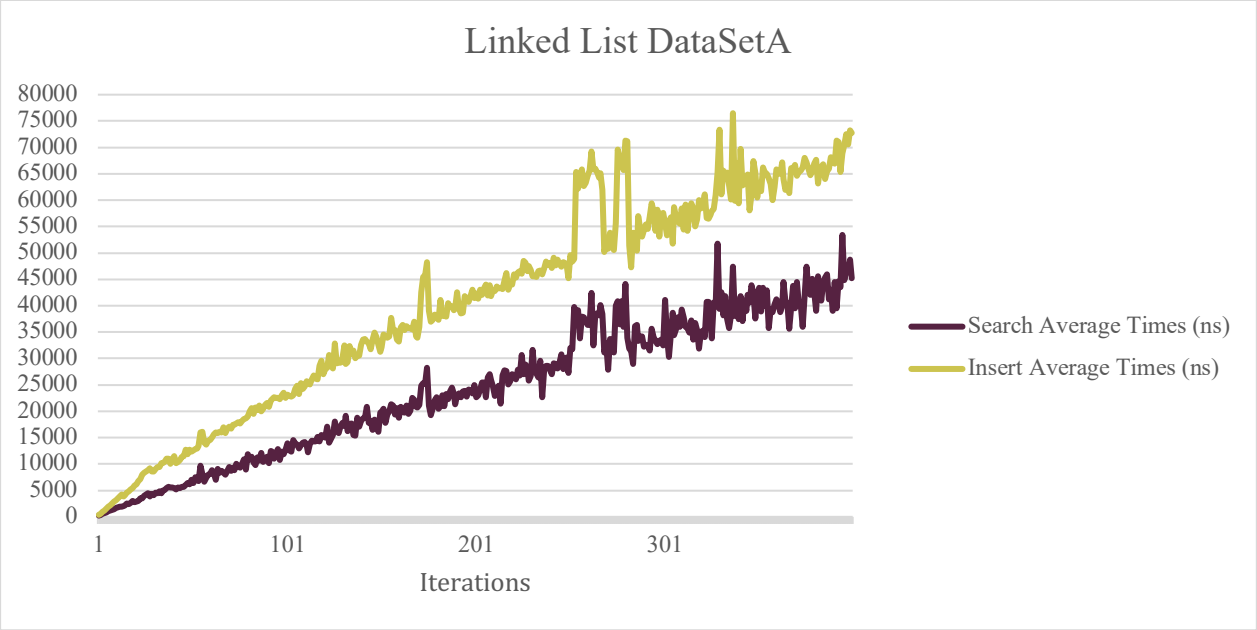
- Linked Lists: **All values inserted were inserted at the head**. This was done so that the insert function would not have to traverse a Linked List in order to find the specified location to insert.

- Binary Search Trees: We did not have to worry about BST duplicates and how to handle those because we filtered out duplicate data points at insert. **Seeded "srand()"**in order to get the most out of the random numbers function instead of the same random numbers on repeat.

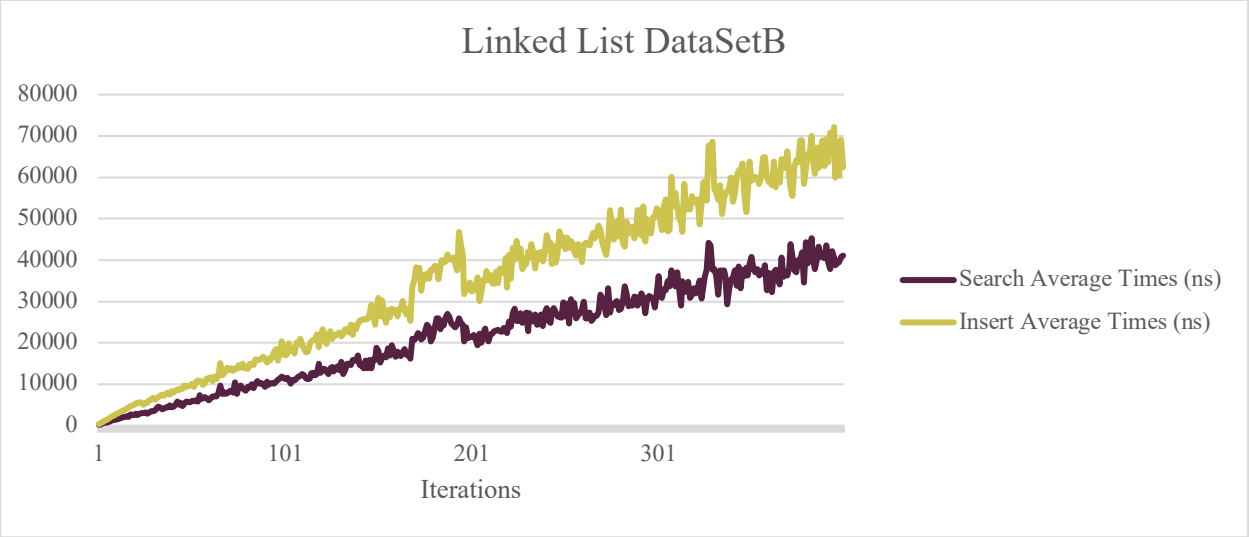
- Hash Tables: **Each collision was marked each time a value was inserted and searched (as long as conditions occurred for one to occur)**. Standard/basic implementation of Hash Tables, similar to currently existing basic/common implementations, in an effort to ensure maximum accuracy of information provided to the client (with regards to industry availabilities).

1 **DataSetA provided for testing by University of Colorado Boulder, Computer Science Department**
2 **DataSetB provided for testing by University of Colorado Boulder, Computer Science Department**
***** **(ns) – refers to Nano Seconds**

Linked List DataSetA



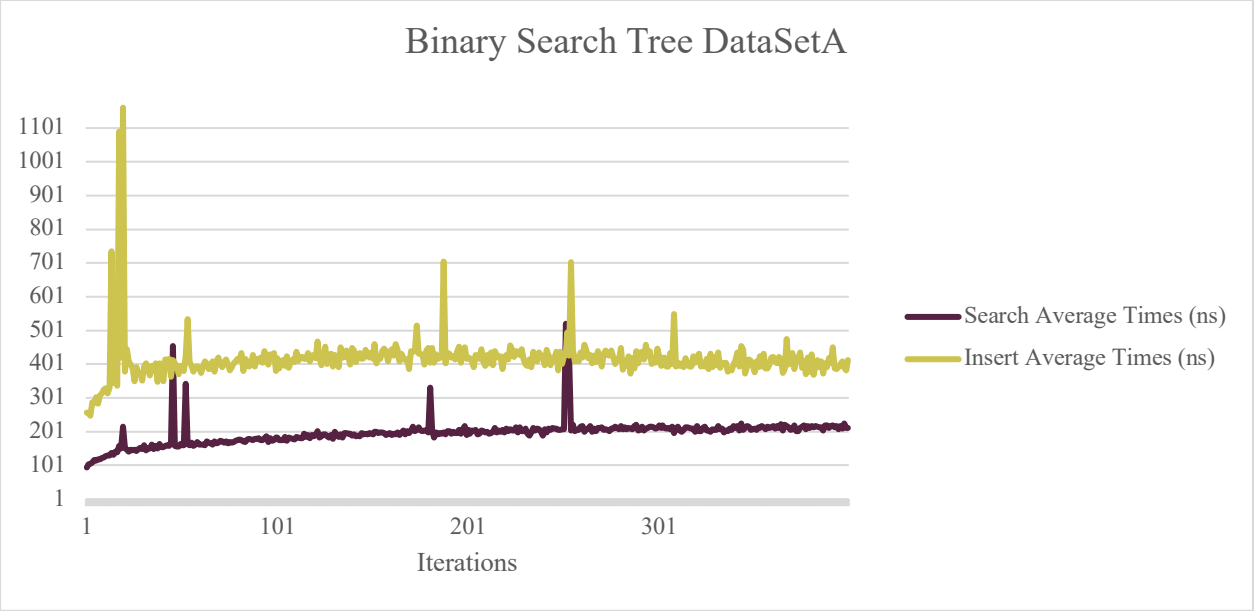
Linked List DataSetB



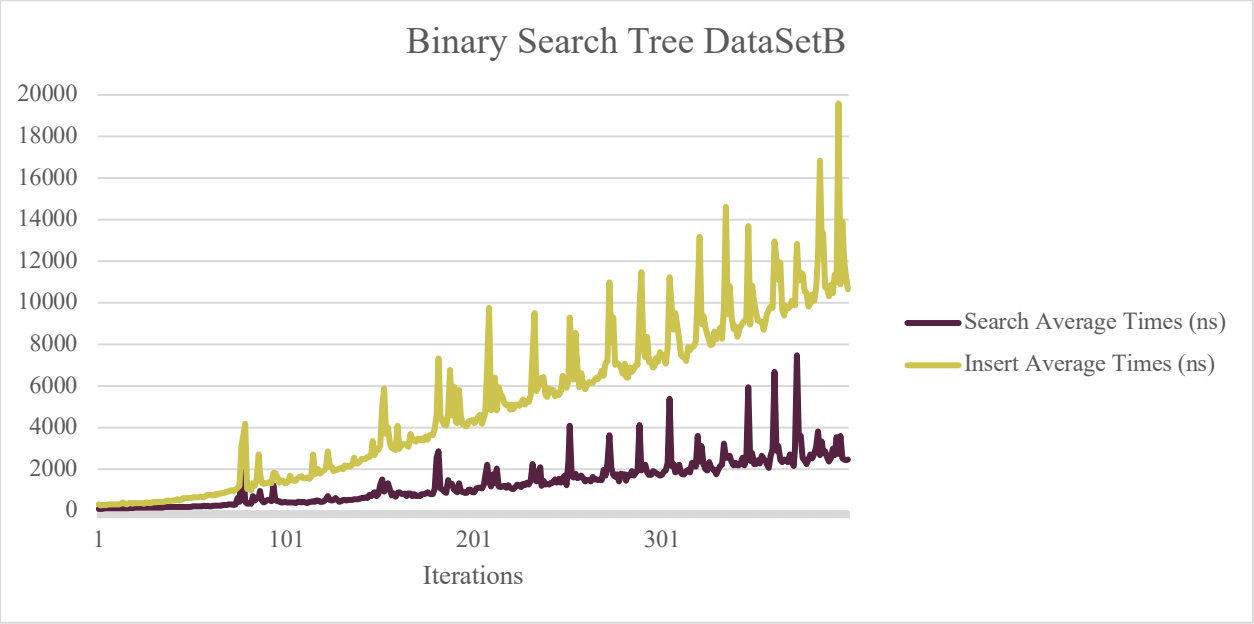
Linked List Analysis

- The data is linear. This is attributed to the BIG O for Linked Lists. Search times are smaller because we avoided duplicates making the list smaller.
- BIG O Insert: (average) O(1) / (worst) O(n)
- BIG O Search: O(n)

Binary Search Tree DataSetA



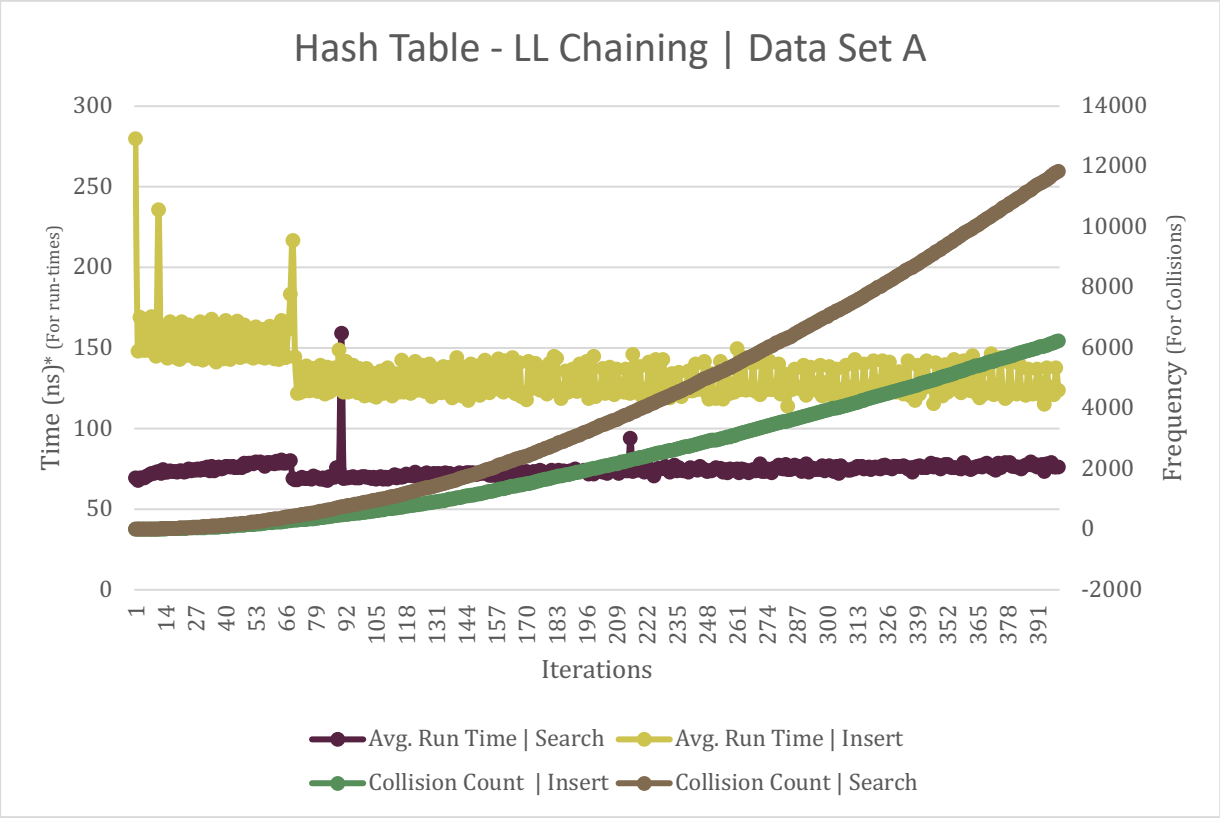
Binary Search Tree DataSetB



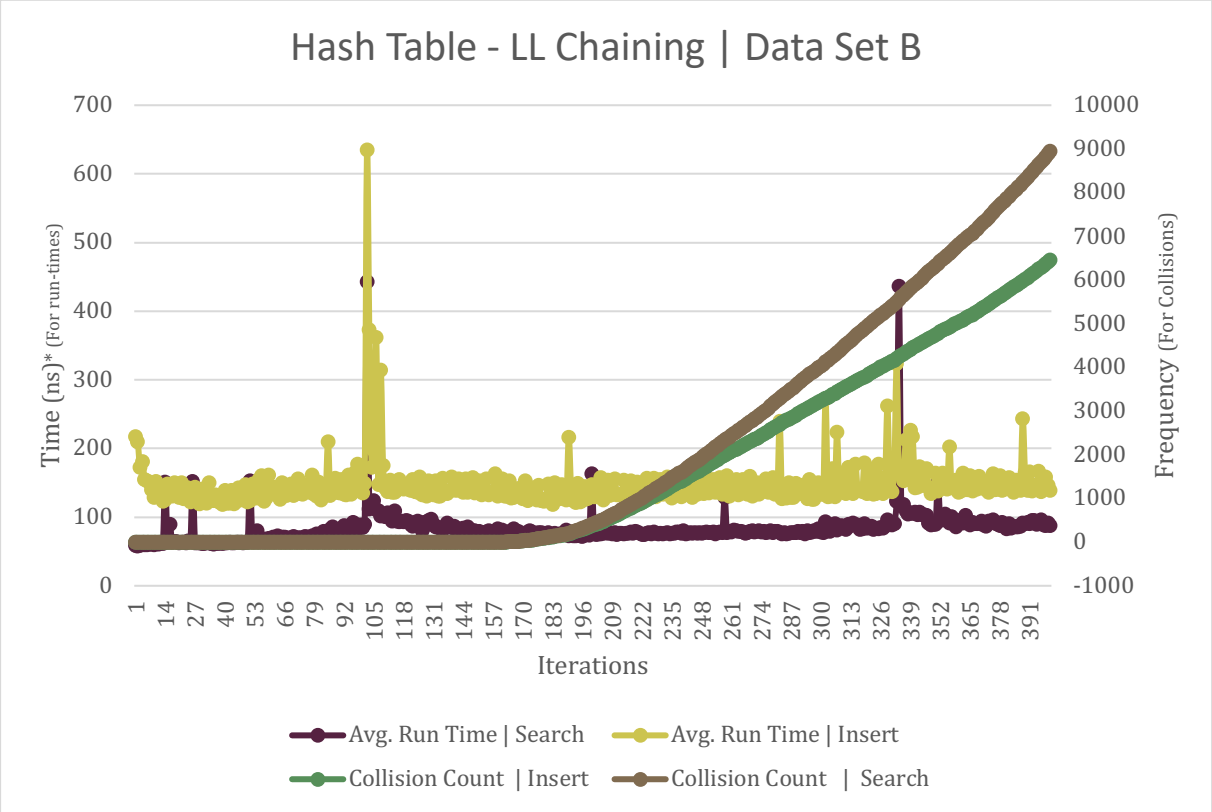
Binary Search Trees Analysis

- The data is logarithmic in data set A. This is attributed to the BIG O for Binary Search Trees.
- The data is fairly linear in data set B. This is attributed to the unbalanced nature of this data sets tree. The unbalanced tree causes the variation to the insert and search runtimes.
- BIG O Insert: (average) O(log(n)) / (worst) O(n)
- BIG O Search: (average) O(log(n)) / (worst) O(n)

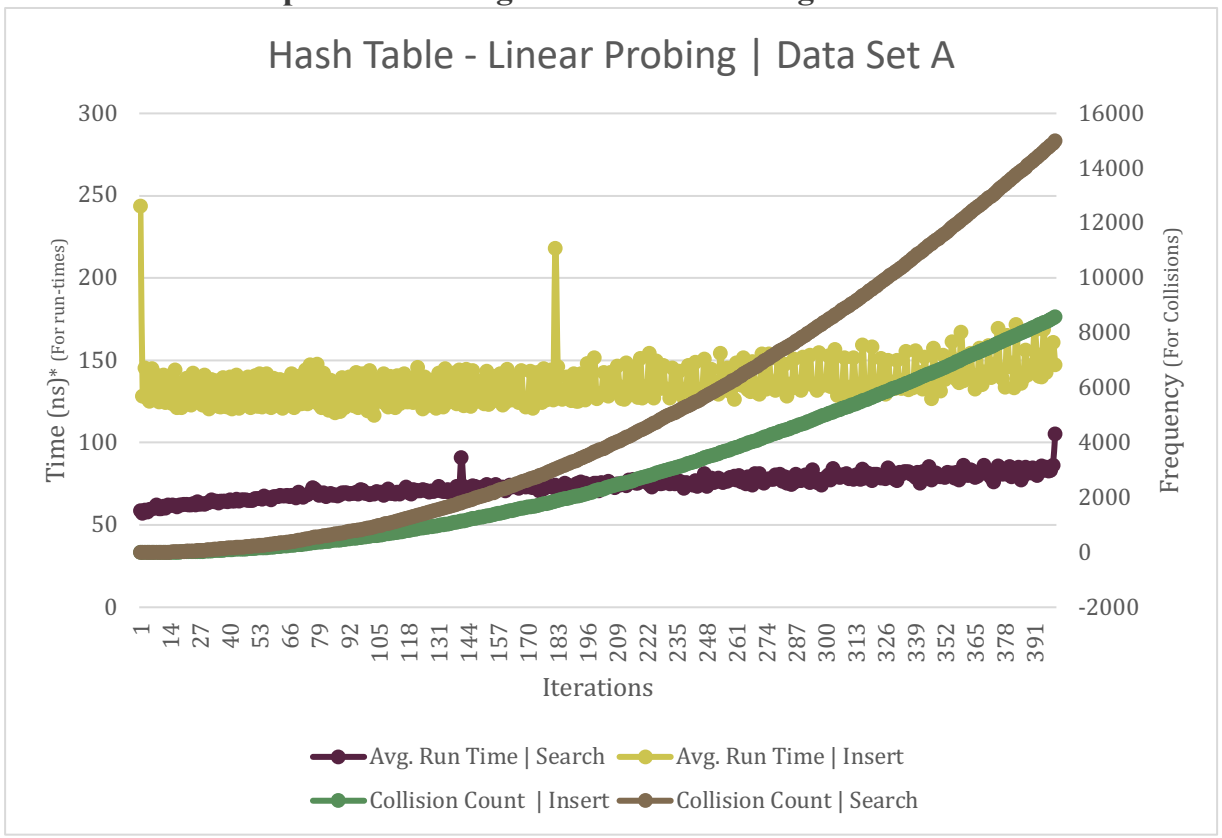
Hash Table with Chaining DataSetA



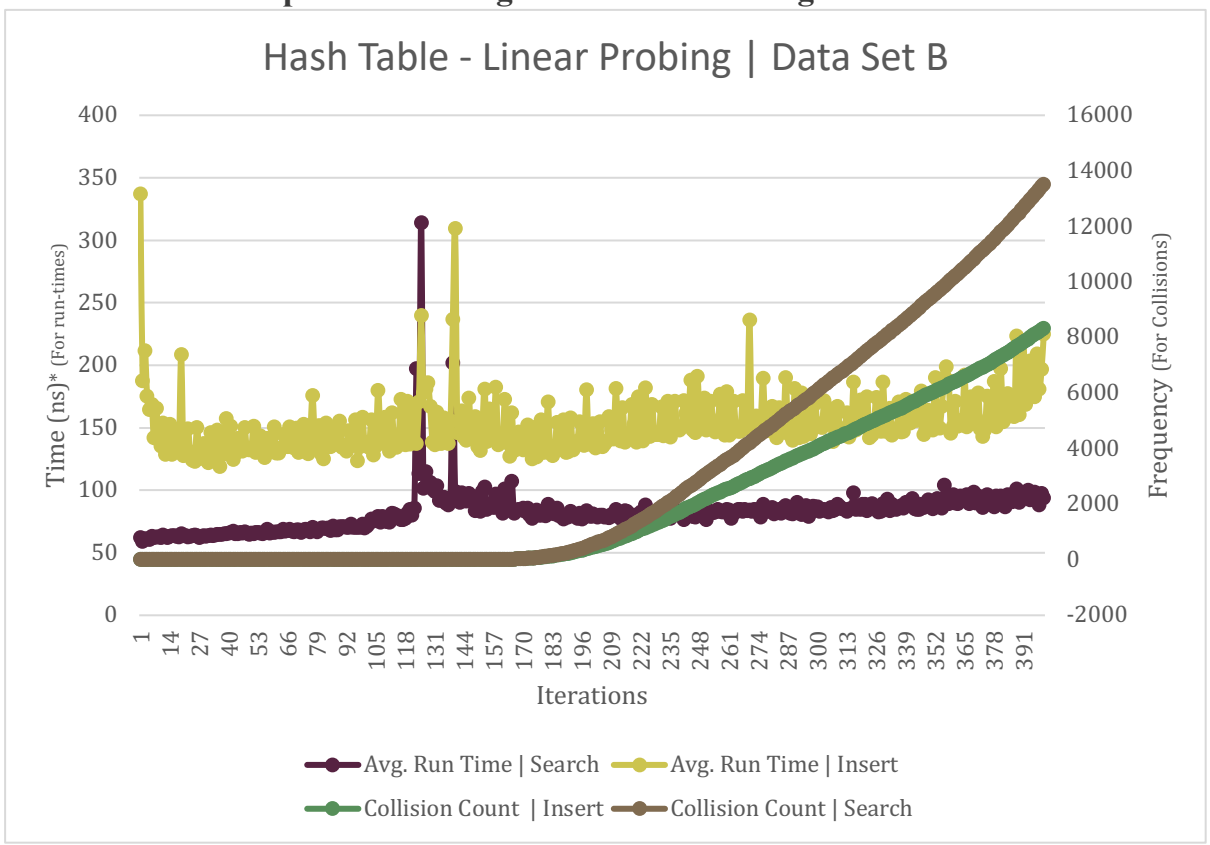
Hash Table with Chaining DataSetB



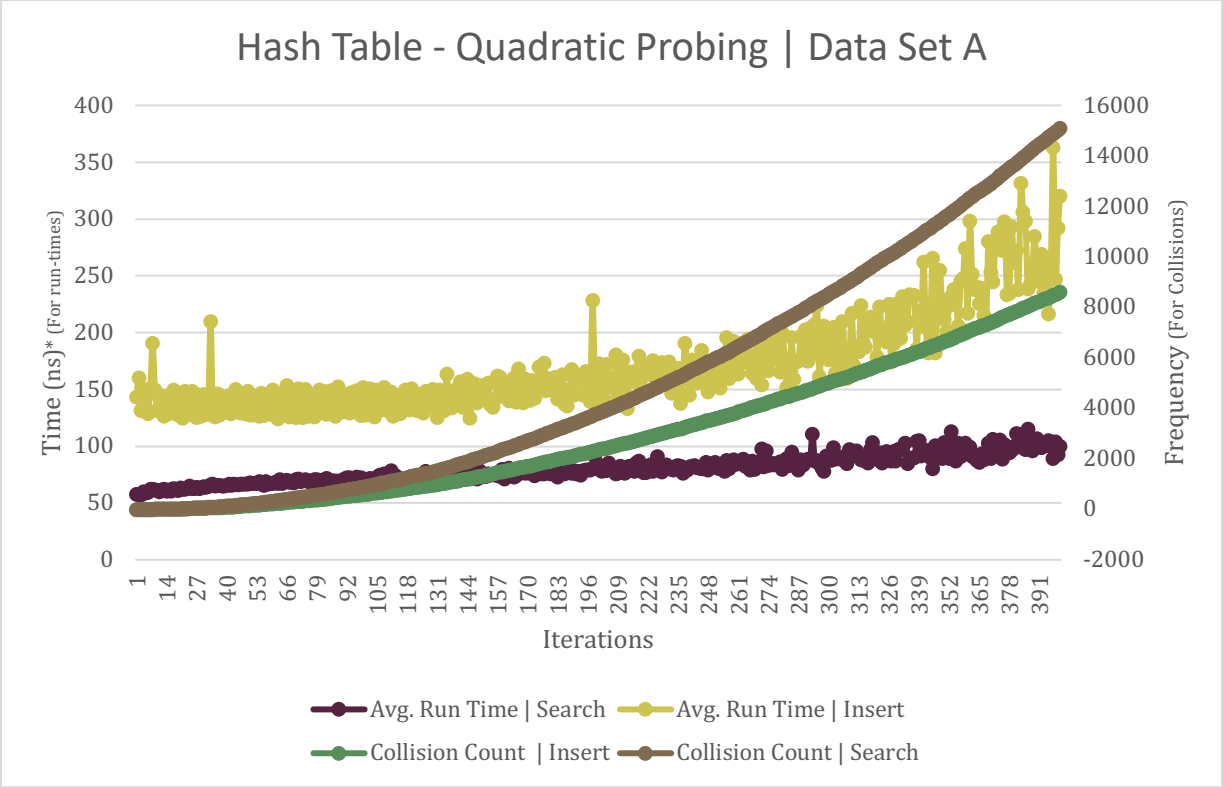
Hash Table with Open Addressing and Linear Probing DataSetA



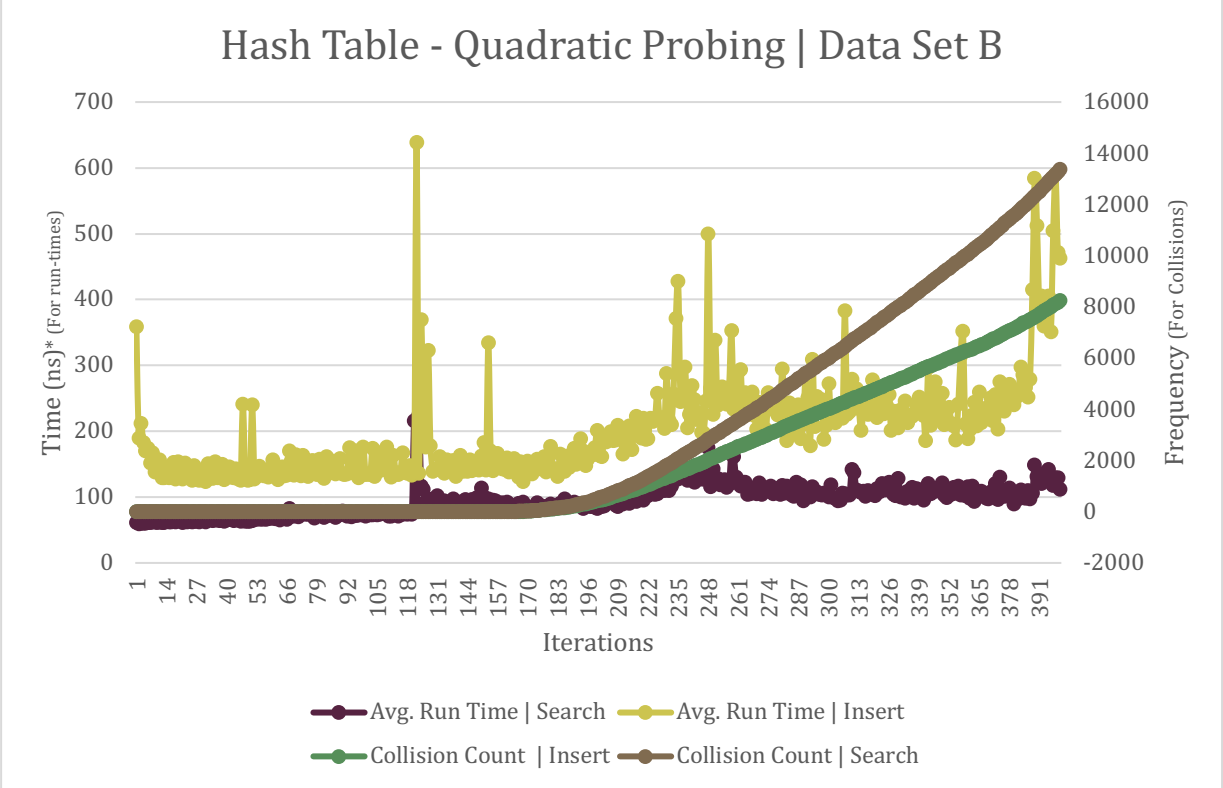
Hash Table with Open Addressing and Linear Probing DataSetB



Hash Table with Open Addressing and Quadratic Probing DataSetA



Hash Table with Open Addressing and Quadratic Probing DataSetB

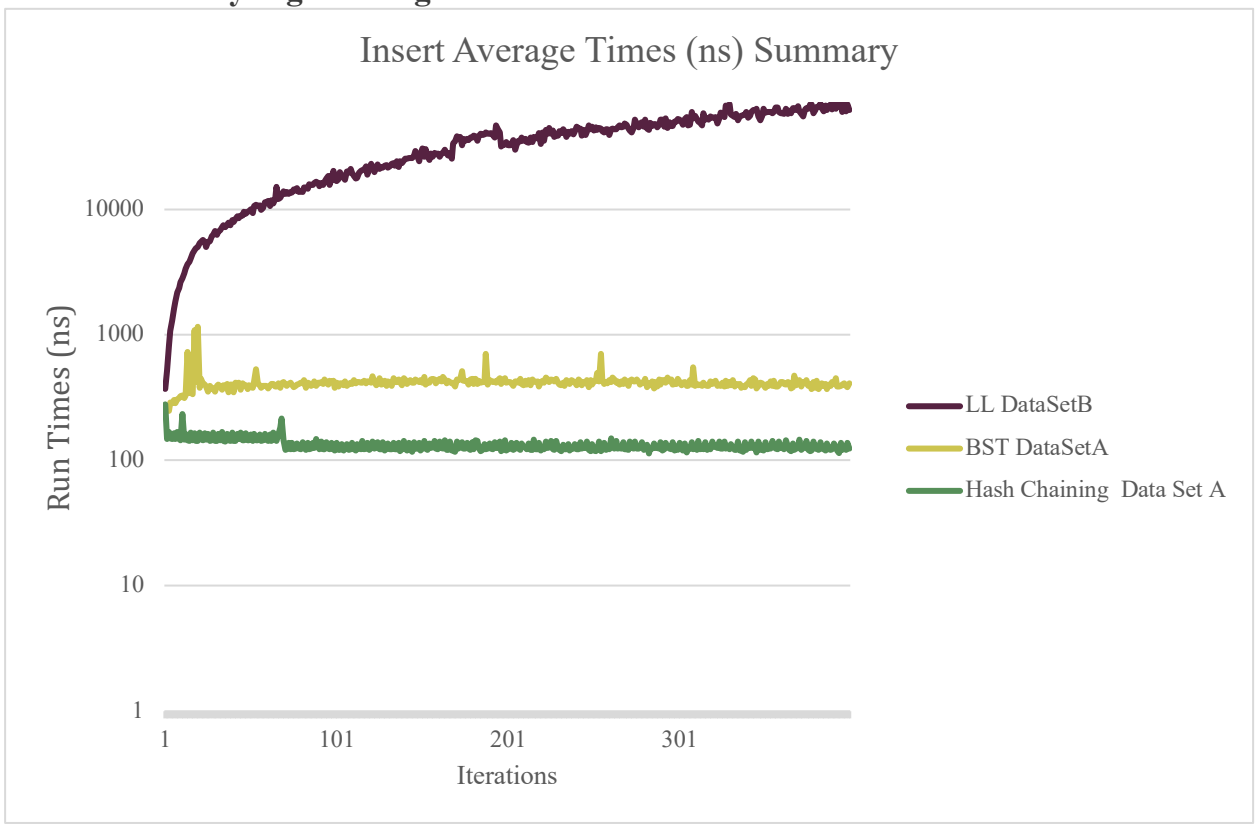


Hash Table Analysis

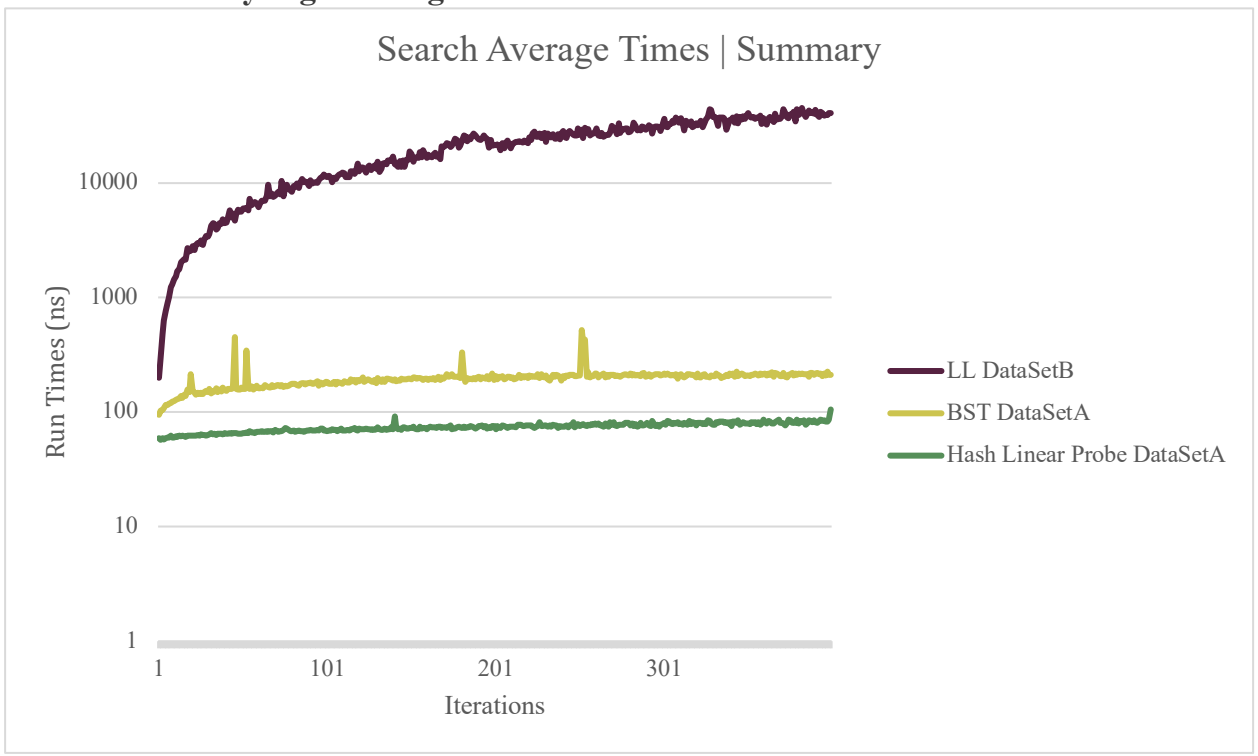
- The data is scattered/unidentifiable (in terms of trend) in these results (exception: logarithmic for quadratic - data set A). Majority of them seem linear but more so with a “slope” of 1 (consistent) when looked at from the big picture. This is attributed to the BIG O for Hash Tables. The clear winner here is slightly hard to determine due to the similarity between conflict resolution via Chaining or Linear Probing.
    - (More on this in the final summary)
- BIG O Insert: (average) O(1) / (worst) O(n)
- BIG O Search: (average) O(1) / (worst) O(n)

This area was purposely left blank, please proceed to the next page.

Insert Summary Figure: Logarithmic Axis Scale



Search Summary Figure: Logarithmic Axis Scale



Final Analyzation/Recommendation

After much testing, and analyzation, the following recommendation is made. Due to the high linear and logarithmic run-times seen in data structures like BST and Linked Lists (along with consistent irregularities and severe lack of consistency), and Hash Tables (although varying) provided as close of a (O(1)) relatively level/consistent and low overall run-time(in comparison to the other options). Fittingly, despite the "collisions"(when two separate pieces of information technically belong in the same spot/index) involved with the expected (and more often achieved) low (O(1)) time of Hash Tables, they prove to the be the most efficient option for USPS to currently implement. Based on the testing and analysis conducted in this study, it is recommended that the client implement a Hash Table (w/Linked List Chaining Collision Resolution), in order to get the fastest "speed(s)" possible. Furthermore, Linked List based Chaining with the Hash Table is widely utilized throughout various industries, allowing for easily available support/troubleshooting, and previous market test trials.