Matthew Su, Marcus Chong

CSCI 2270

Final Project Write-Up

4/29/20

*Note: We tried to generate a given data graph (assuming the given data was dataA and dataB), but none of us had excel and google sheets said it was too large. That is the only graph missing.

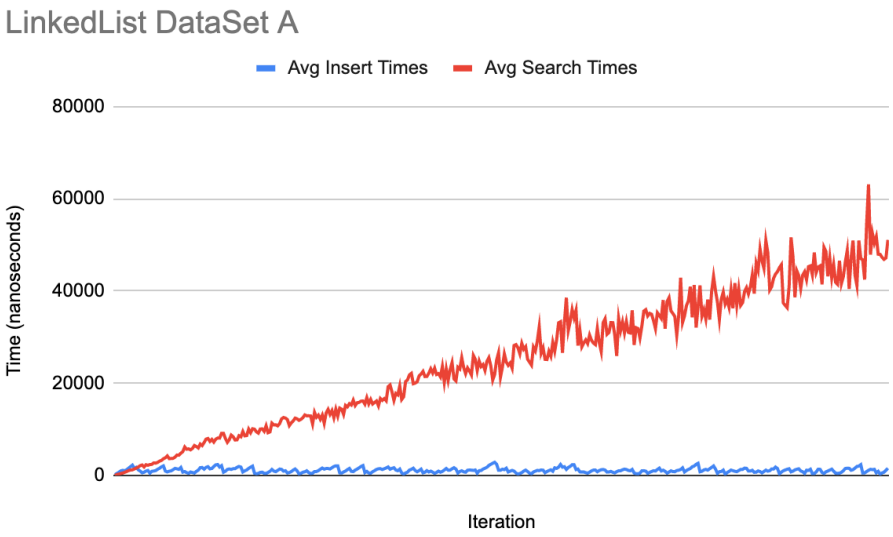
When generating the data needed to plot a graph showing average search and insert times, Marcus and I designed functions within the classes that performed the increments of adding 100 new pieces of data from the dataset and searching 100 times for values within the expected range of random number values that can be searched. This greatly helped keep the actual experiment concise and easy to debug. Overall the results of our project reveal a lot about the real world time to complete that occur with different types of data structures:

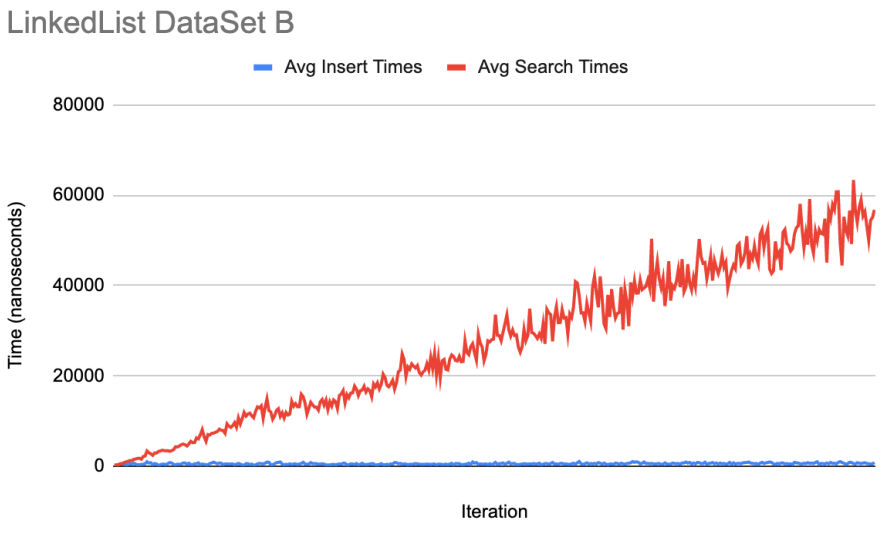
- It is plainly obvious that Linked List is the worst choice for a data structure in terms of processing times and overall efficiency. Between two different data sets, it managed to keep consistency but was consistently the slowest overall in terms of average completion time and overall time to complete. It's simply because of the structure itself; sure traversing the first data point takes no time at all, but having to make your way to the 40000th data point from the first is too efficient for even more than 100 data points let alone hundreds of thousands per hour.
- Binary search trees were a much better improvement upon last place Linked Lists. They also managed to keep consistent average time behavior through both data sets and

were 70-80% more efficient overall. Their overall performance was worlds better, and that's also due to their structure. The insert and search functions perform almost identical traversals in order to reach their destinations, a traversal method that is exponentially more efficient than a linked list. That's why insert and traversal times are similar, and why overall efficiency benefits greatly as a result.

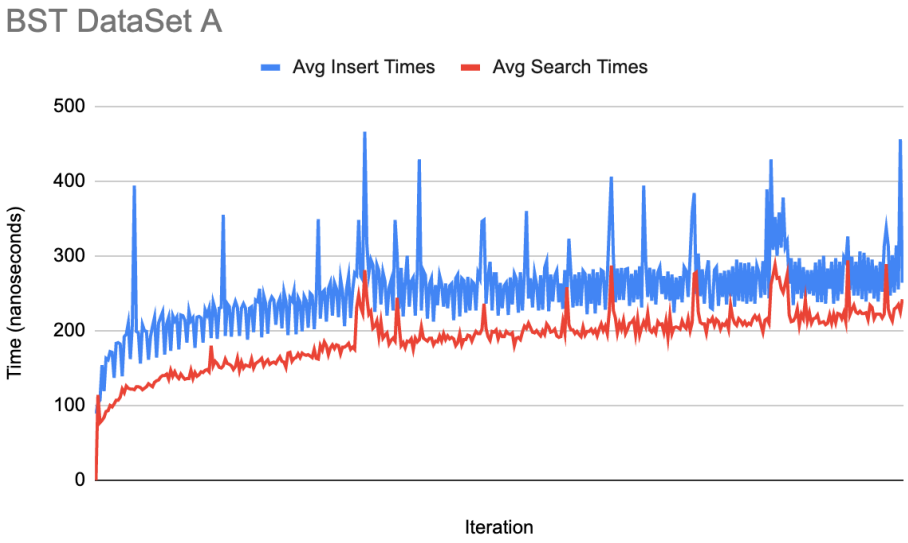
- Hash tables overall took very little time and were comparable to the times for BST insert and search. Outside of some anomalies in the chained link list data, it overall showed sub 250ns average times mostly through the experiment. Out of the three methods of insert/search, our data shows that linear probing was the most consistent and lowest time to complete overall with mostly sub 200nn times, the fastest of all other times. The use of hash functions overall was the major factor in making this data structure the most efficient, because instead of having to do any traversing that the linked list and even the BST have to do, lists can be accessed simply through the use of the table and the hash function position indexes. That makes data insert and search significantly more consistent with the same step consistency.

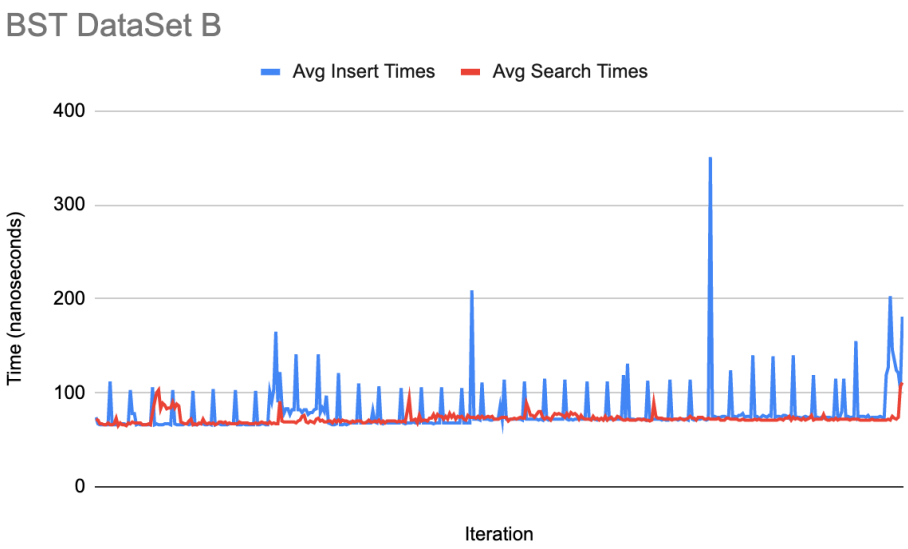
Overall of the three data structures, we recommend the use of the hash table data structure to be the foundation for its mail tracking software. It provides the fastest and most efficient method of traversal, outperforming its competition with marginal to significant gains in all areas.

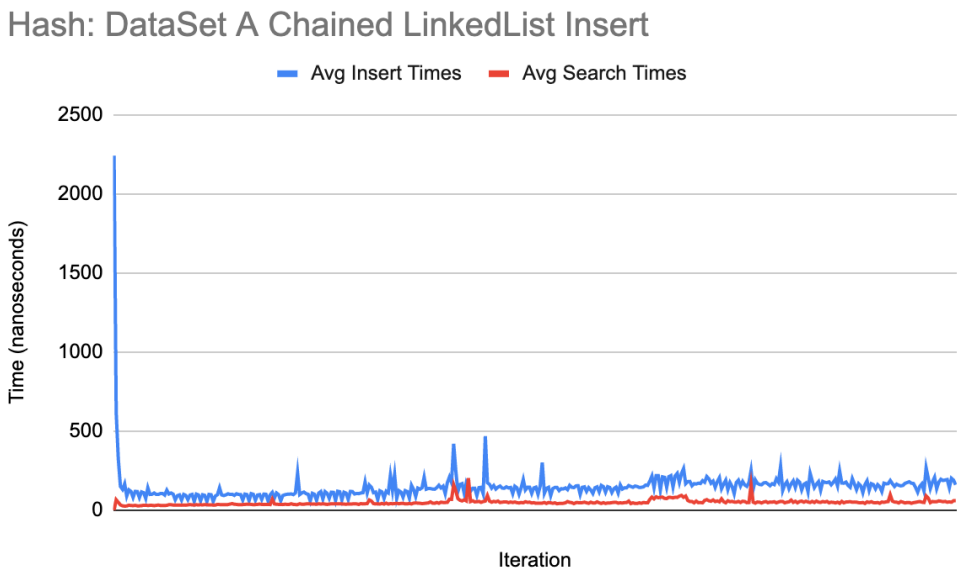

LinkedList DataSet A
Avg Insert Times
Avg Search Times
80000
60000
40000
20000
0
Time (nanoseconds)
Iteration

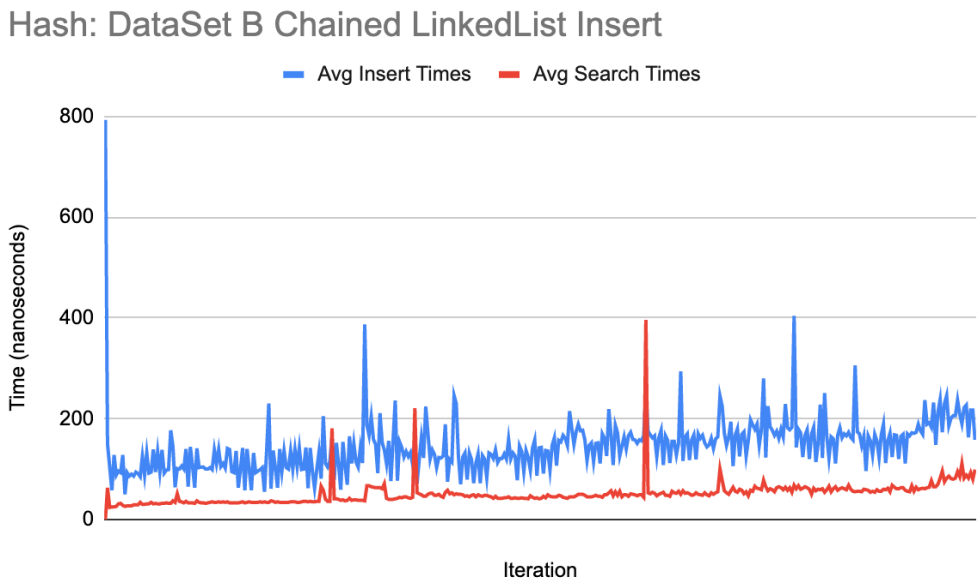

LinkedList DataSet B
Avg Insert Times
Avg Search Times
80000
60000
40000
20000
0
Time (nanoseconds)
Iteration

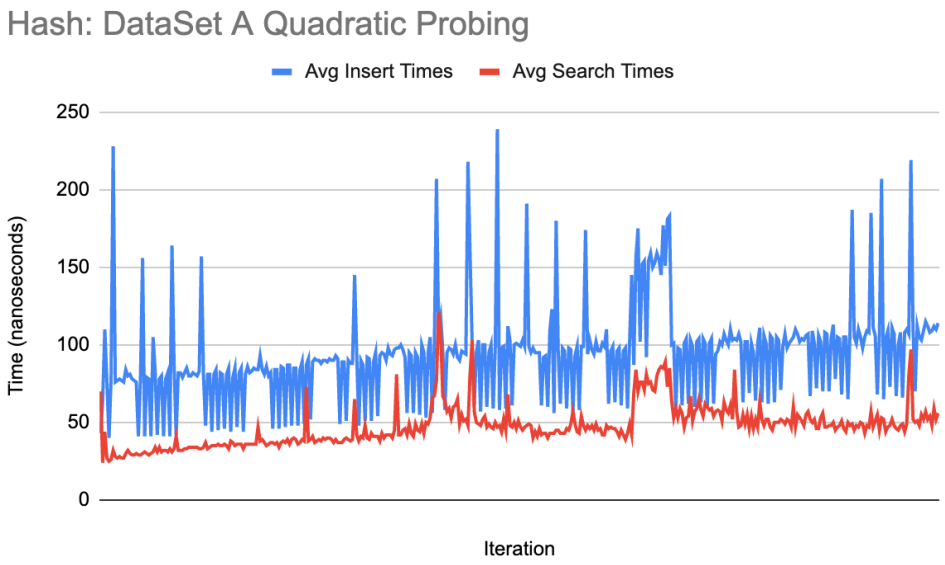
*Note: the graphs were done in nanoseconds, so the difference in average time per insert and search are significantly different, but in reality the difference is fairly negligible. What is important is that the trend remains consistent across two different data sets.

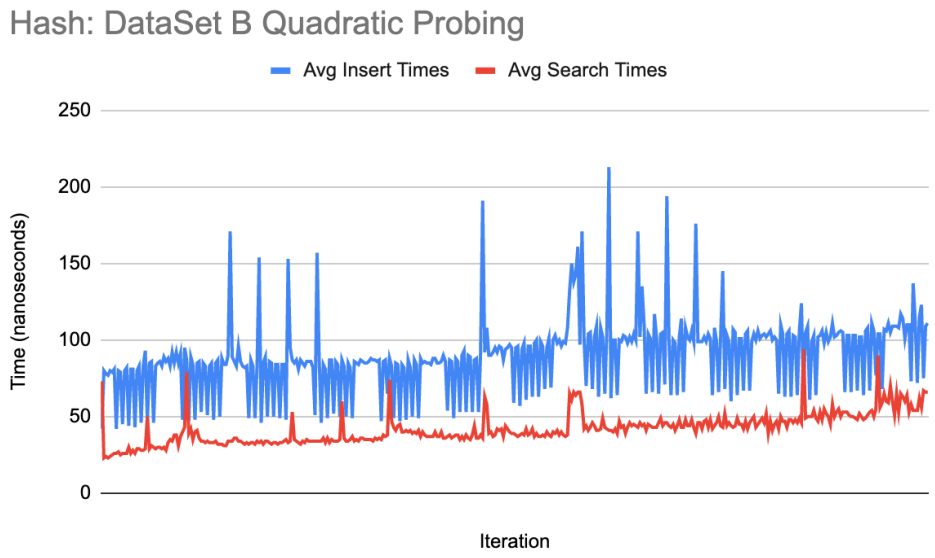


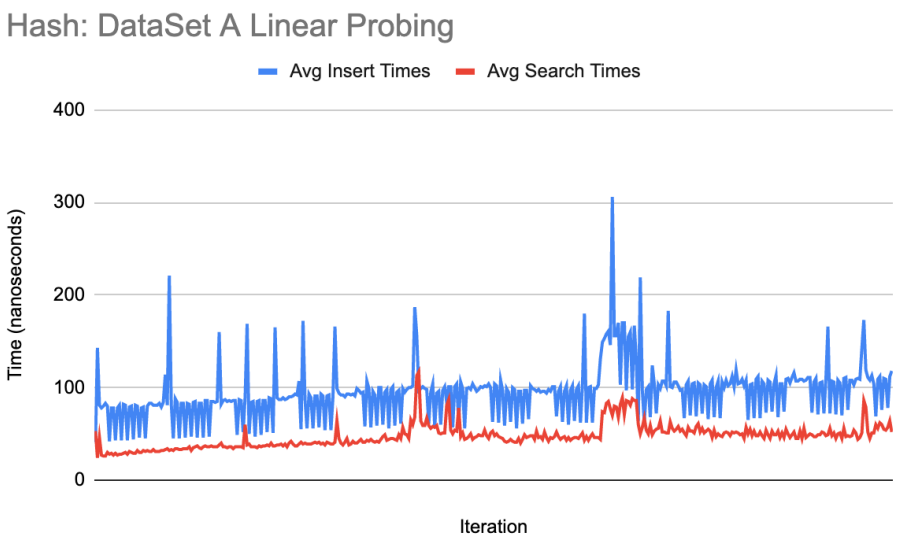


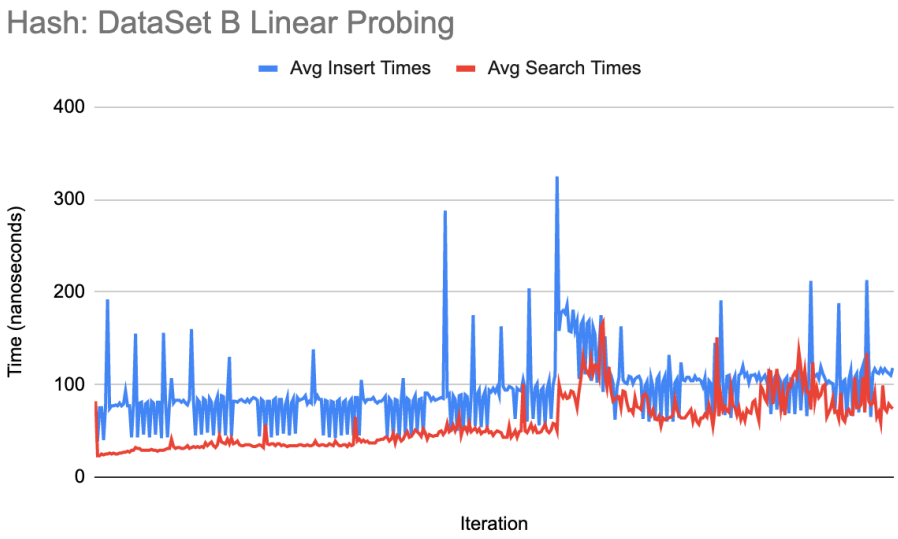

Hash: DataSet A Chained LinkedList Insert
Avg Insert Times
Avg Search Times
2500
2000
1500
1000
500
0
Time (nanoseconds)
Iteration


Hash: DataSet B Chained LinkedList Insert
Avg Insert Times
Avg Search Times
800
600
400
200
0
Time (nanoseconds)
Iteration

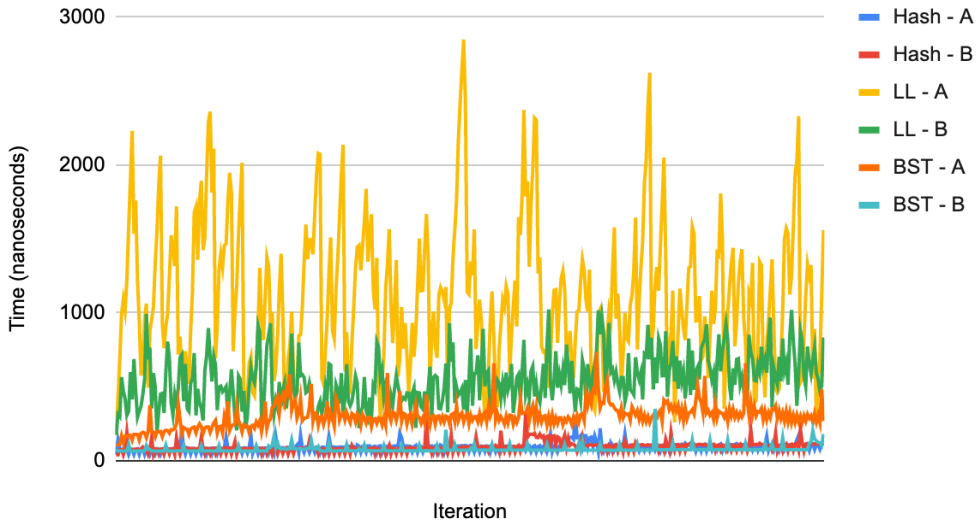

Hash: DataSet A Quadratic Probing
Avg Insert Times
Avg Search Times
250
200
150
100
50
0
Time (nanoseconds)
Iteration


Hash: DataSet B Quadratic Probing
Avg Insert Times
Avg Search Times
250
200
150
100
50
0
Time (nanoseconds)
Iteration


Hash: DataSet A Linear Probing
Avg Insert Times
Avg Search Times
400
300
200
100
0
Time (nanoseconds)
Iteration


Hash: DataSet B Linear Probing
Avg Insert Times
Avg Search Times
400
300
200
100
0
Time (nanoseconds)
Iteration

Summary: Insert



Summary: Search