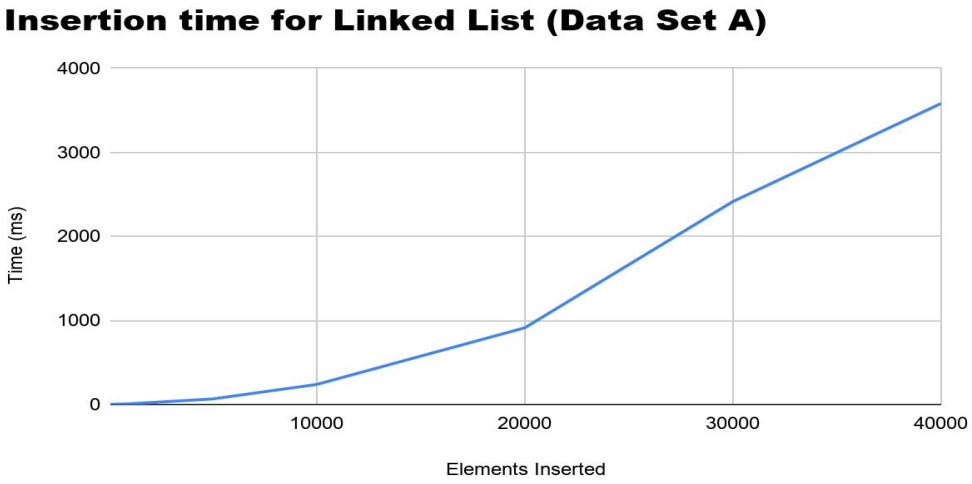
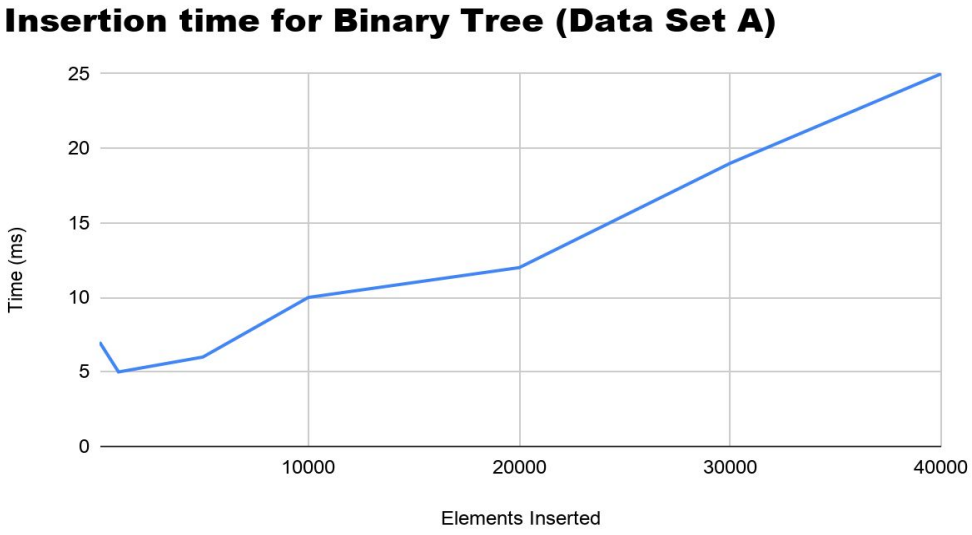
Data Set A Results:

- [Insertion Time]



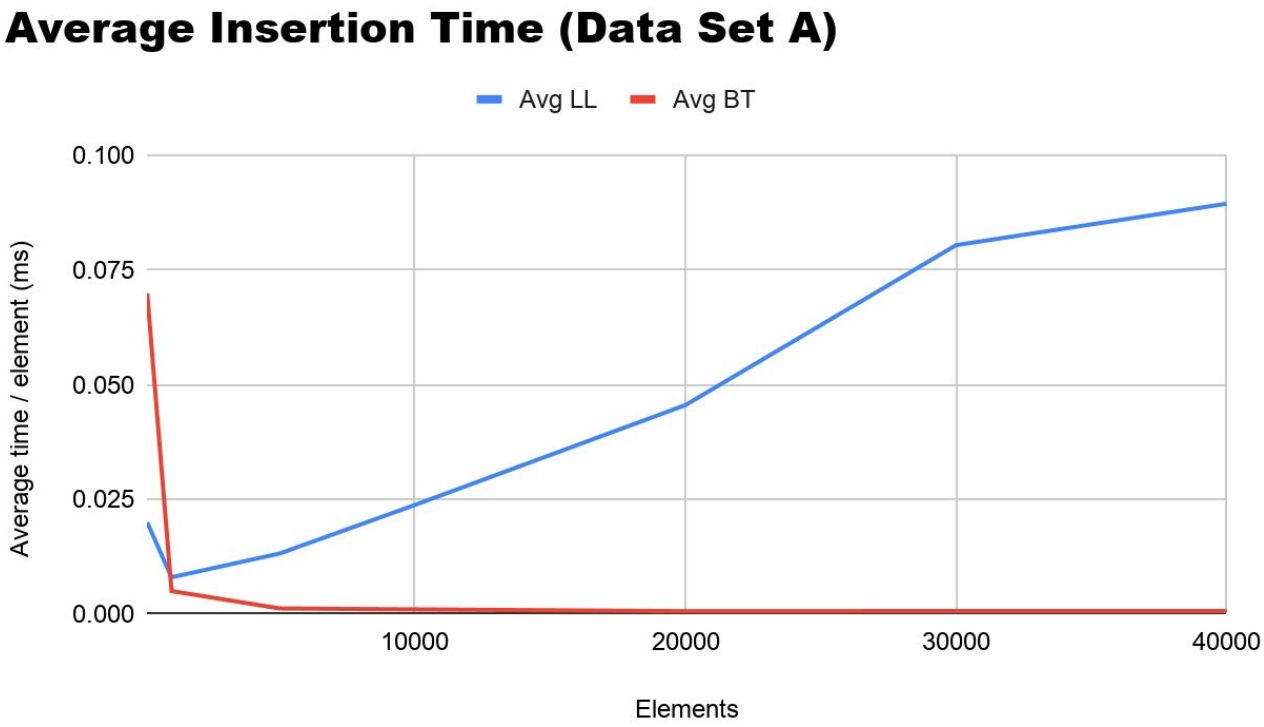


Notes:

- Both graphs show that the run-time increases as the amount of elements increases.
- ***Insertion time was significantly more efficient for Binary Tree when mass inserting elements.***

- Although values were averaged, times may not accurately represent computer run-time especially for a small amount of elements.

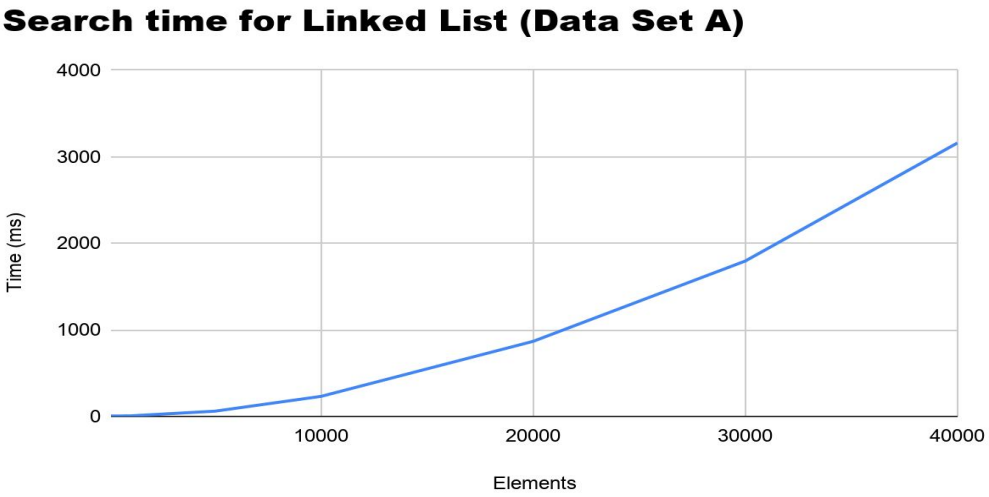
- [Avg Insertion Time]

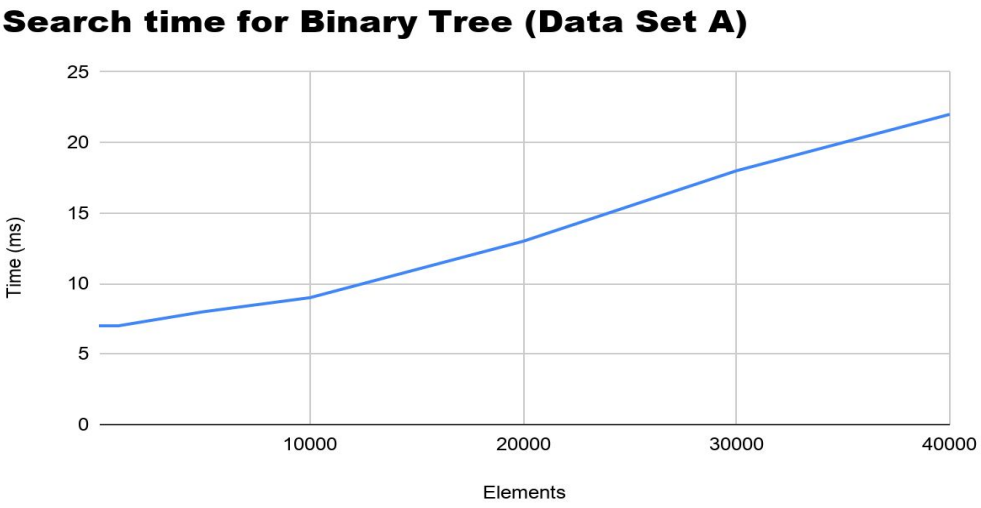


Notes:

- The first takeaway is that the Linked List's average insertion time is increasing whereas the Binary Tree's tends to decrease / stay the same.
- **Binary Tree is more efficient for insertion time / element.**
- The first point for Average BT may be large due to inaccuracy in run-time measurements.

- [Search Time]



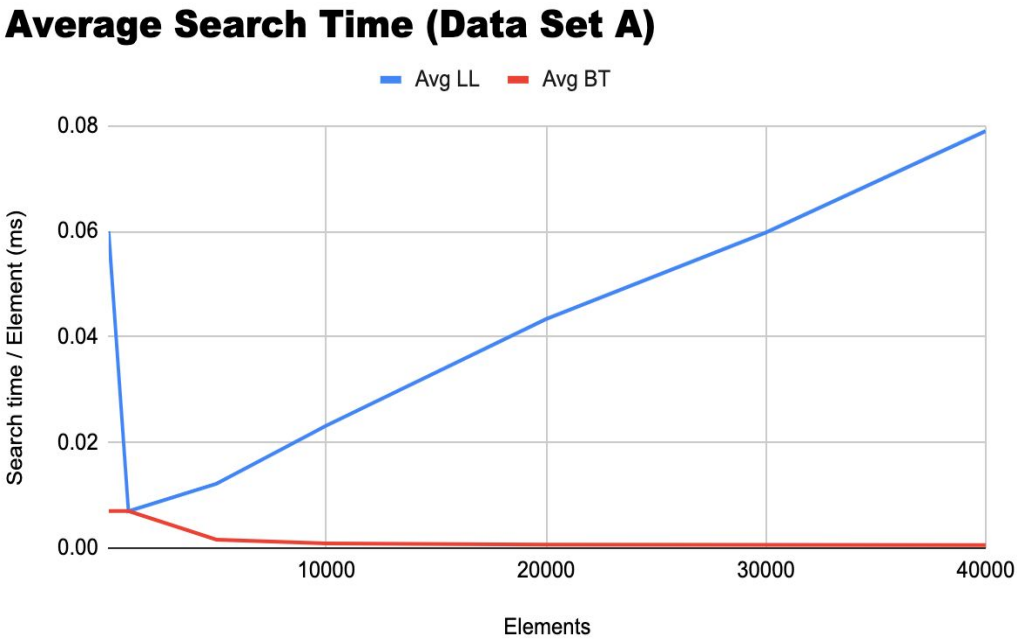


Telly Umada does not exist

Notes

Again, the Binary Tree is ***clearly*** much more efficient when handling mass. Though they both appear relatively linear, the scale in which a Linked List increases per element is thousands of milliseconds more than the Binary Search Tree. This is very likely due to the nature of a Linked List, where the worst-case

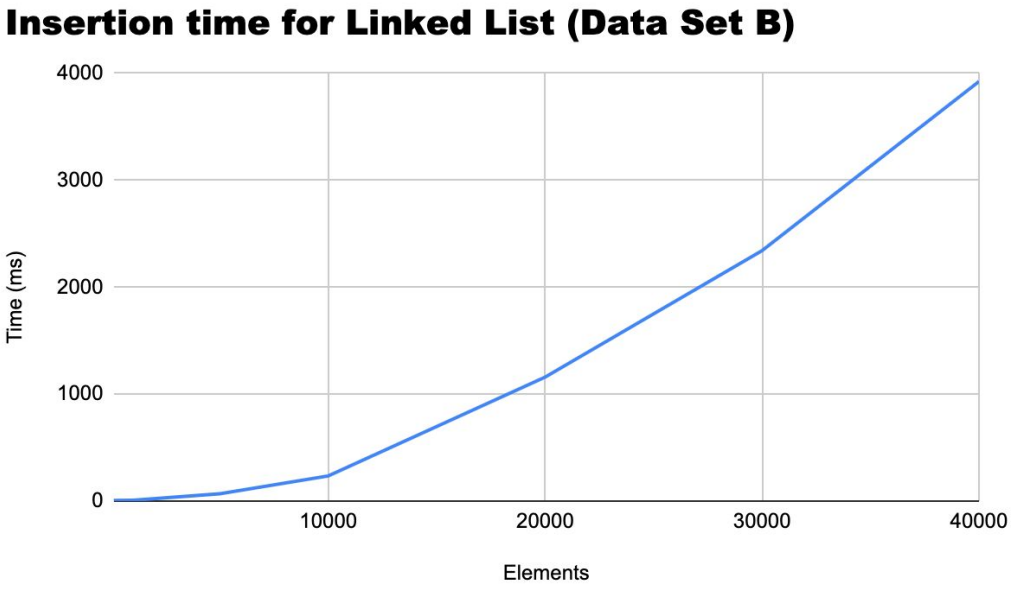
scenario of the search and insertion can take *n* (number of items) comparisons to complete.

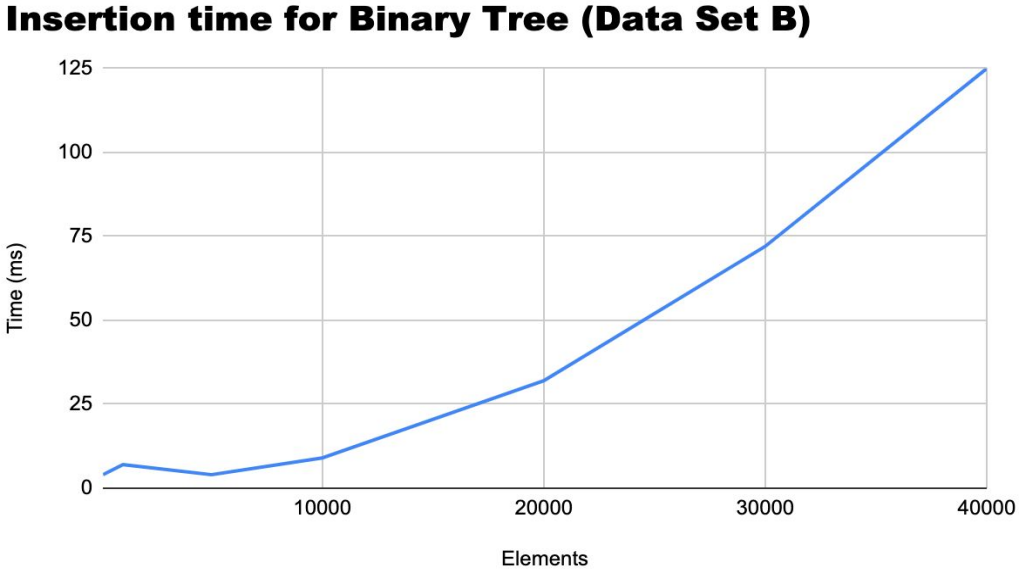


Notes

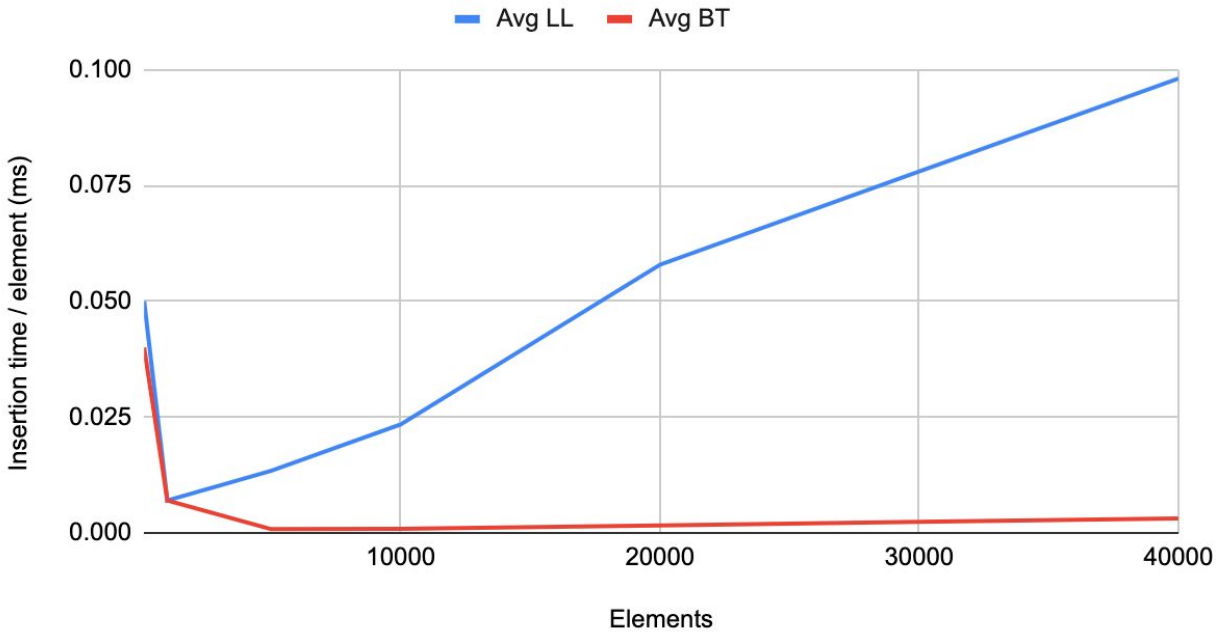
The average time per item more clearly shows a comparison between the linked list and the binary search tree. Evidently, with more and more elements, the linked list takes more time per element, whereas the binary search tree stays relatively the same.

Data Set B Results:

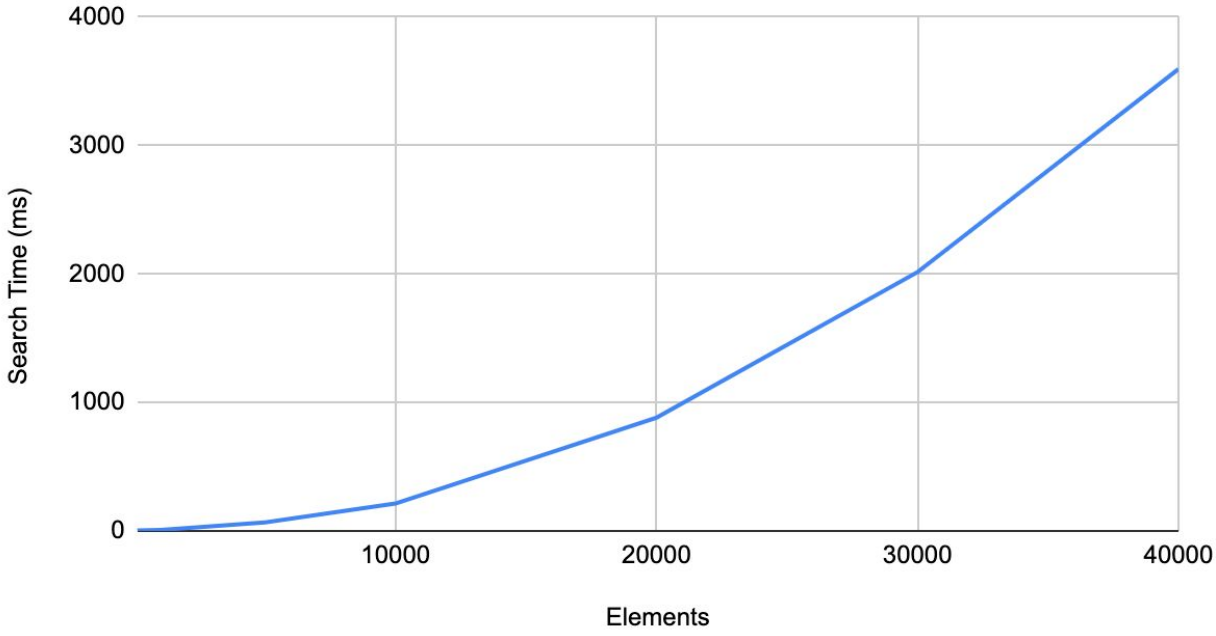


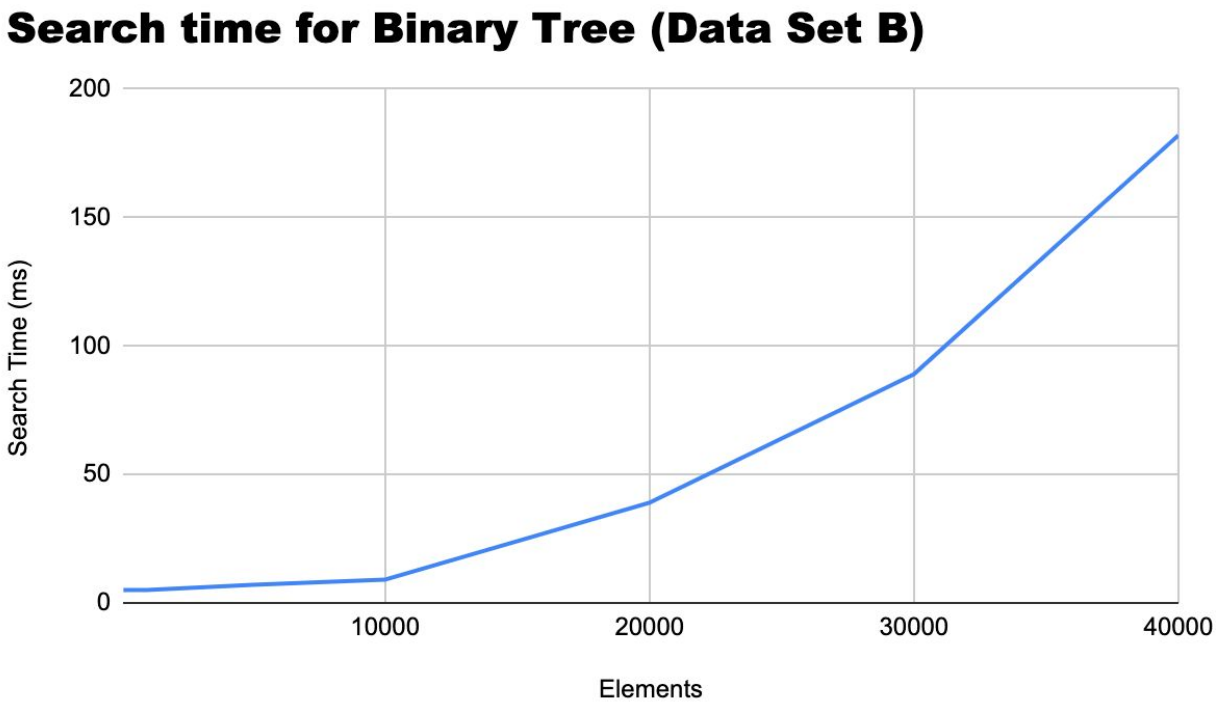


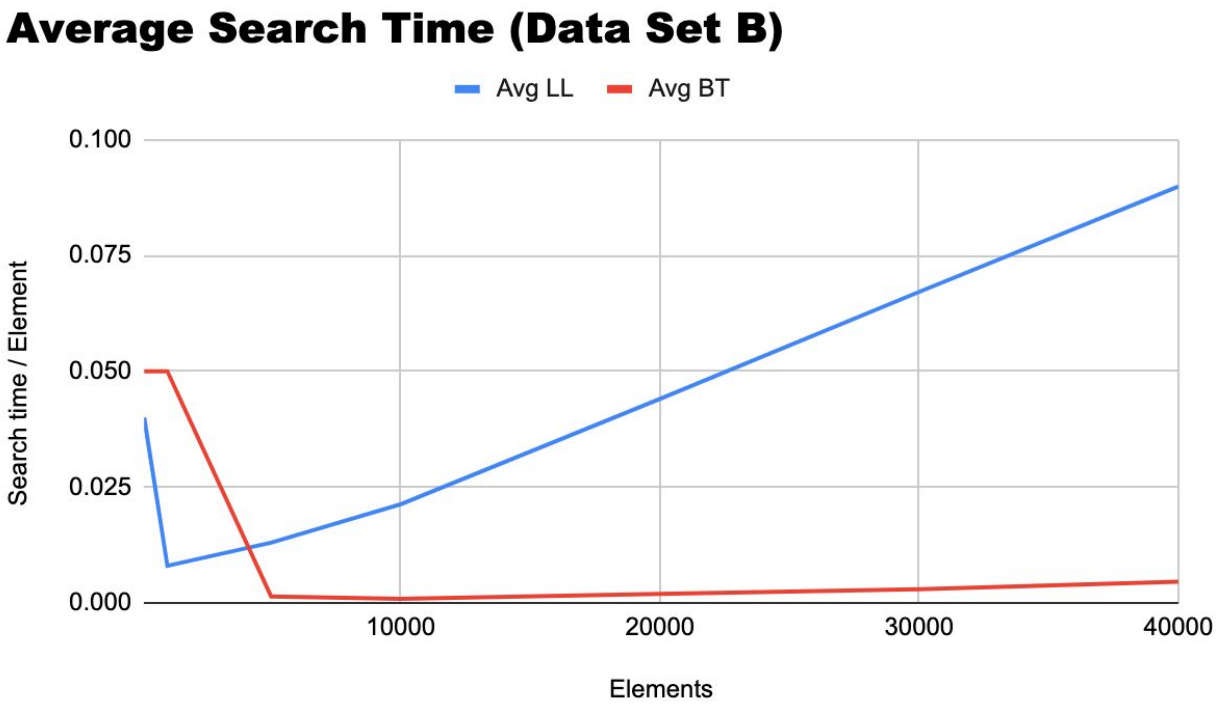
Average Insertion Time (Data Set B)



Search time for Linked List (Data Set B)

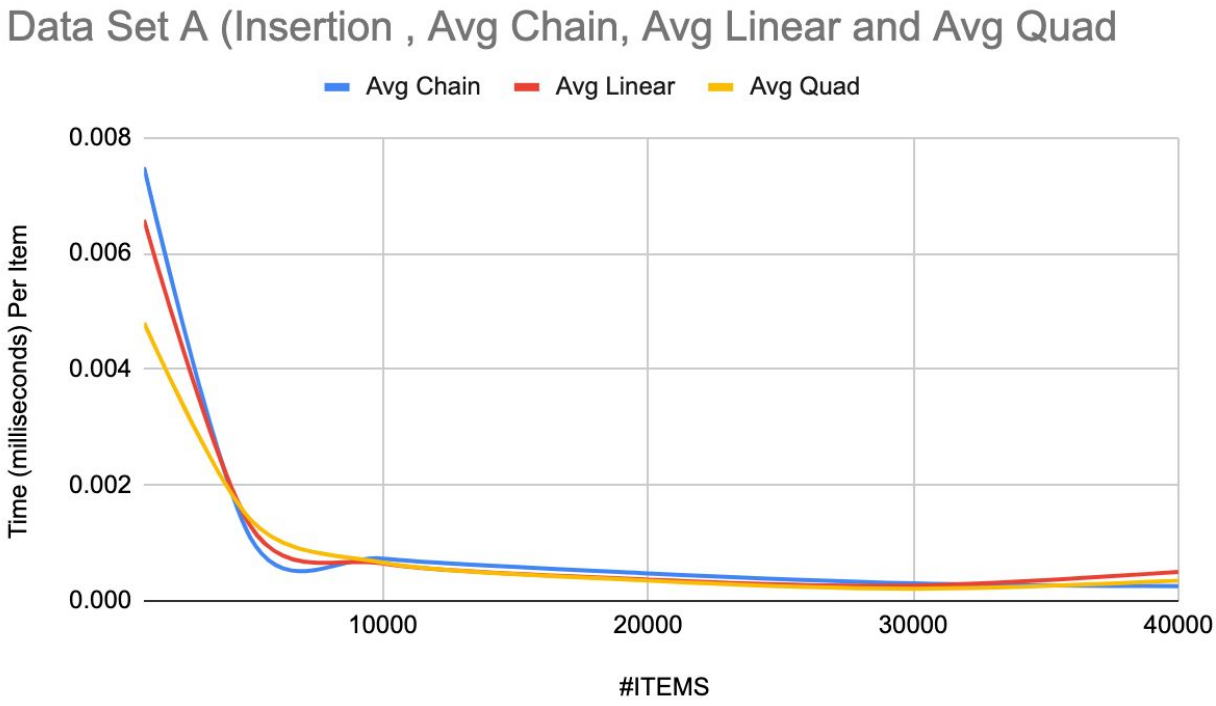


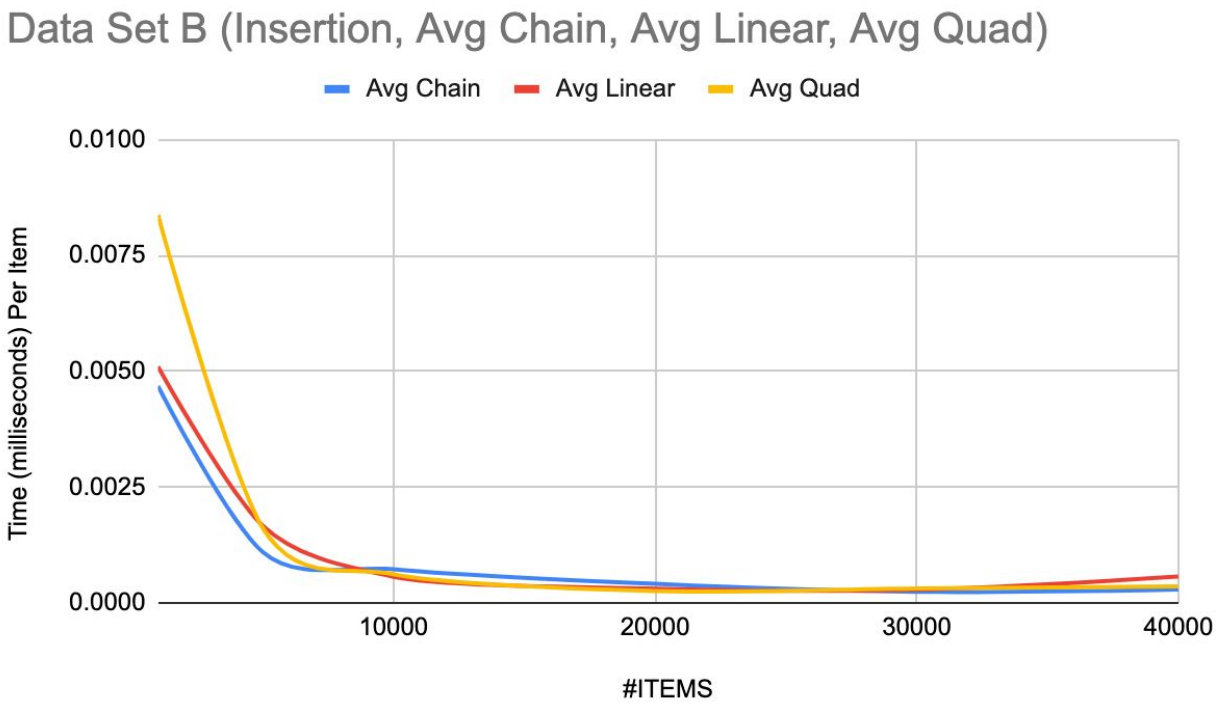




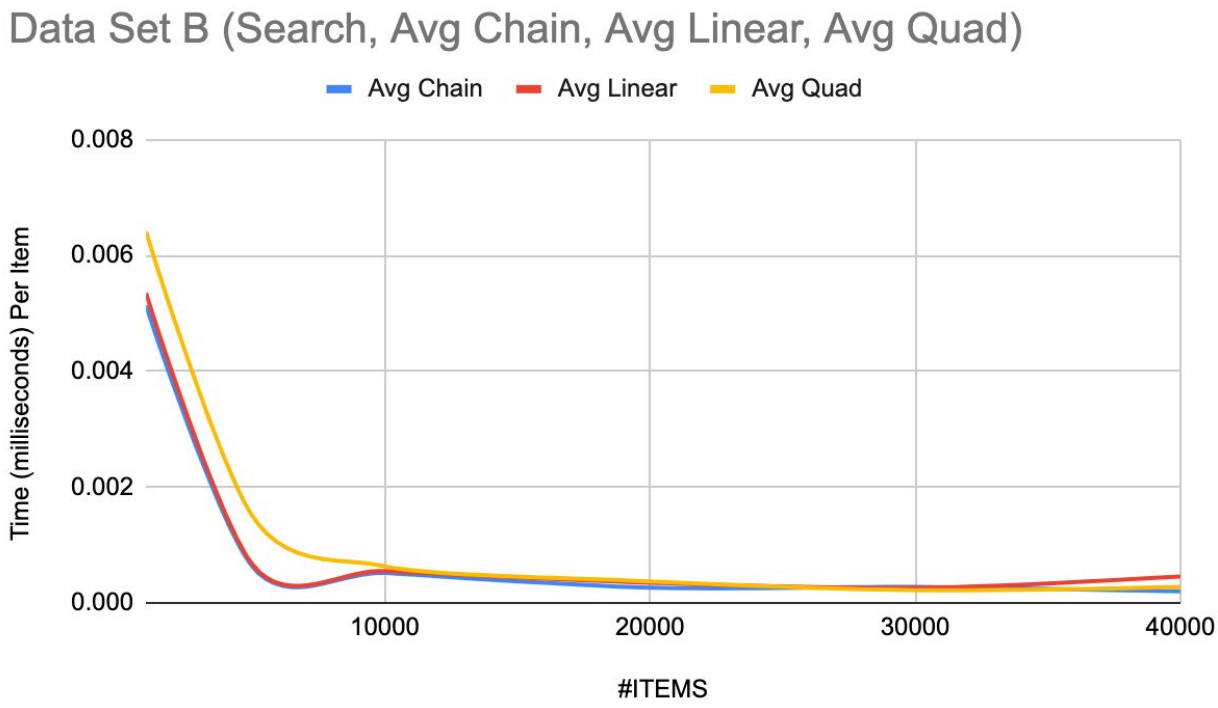
When comparing the two data sets, the relationships remain the same: binary search tree is dually more efficient than the linked list.

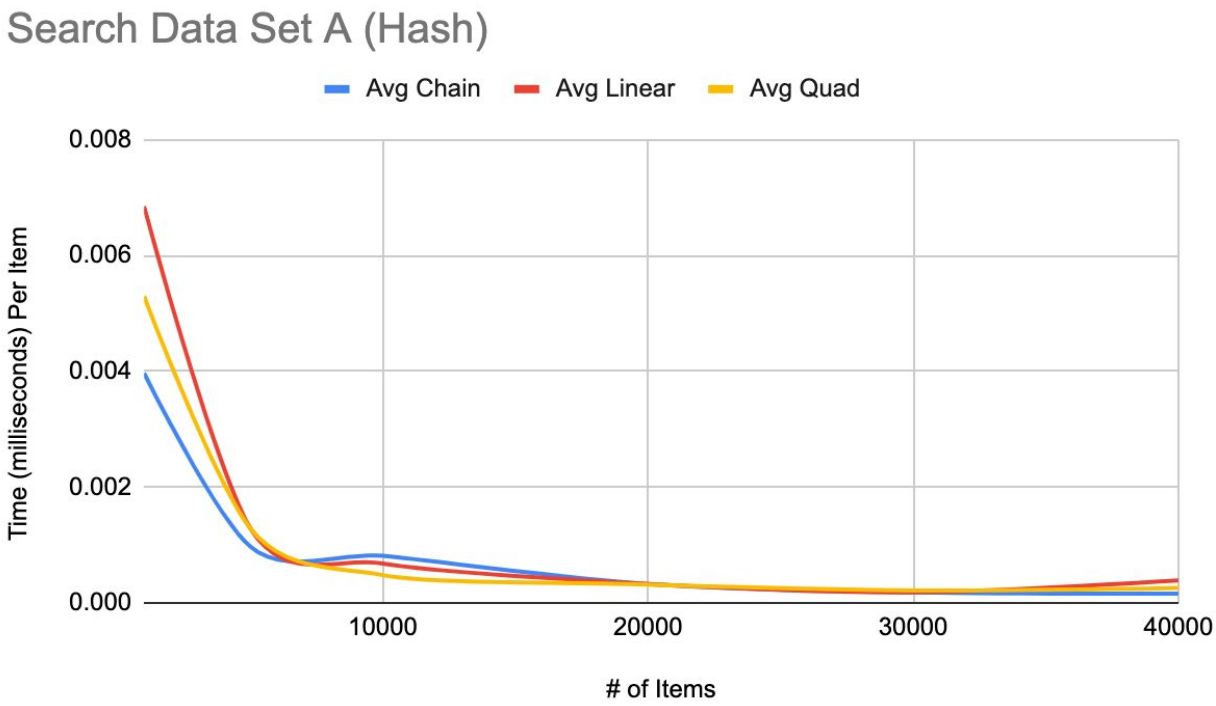
HASH TABLES


Data Set A (Insertion , Avg Chain, Avg Linear and Avg Quad
Avg Chain
Avg Linear
Avg Quad
0.008
0.006
0.004
0.002
0.000
Time (milliseconds) Per Item
10000
20000
30000
40000
#ITEMS

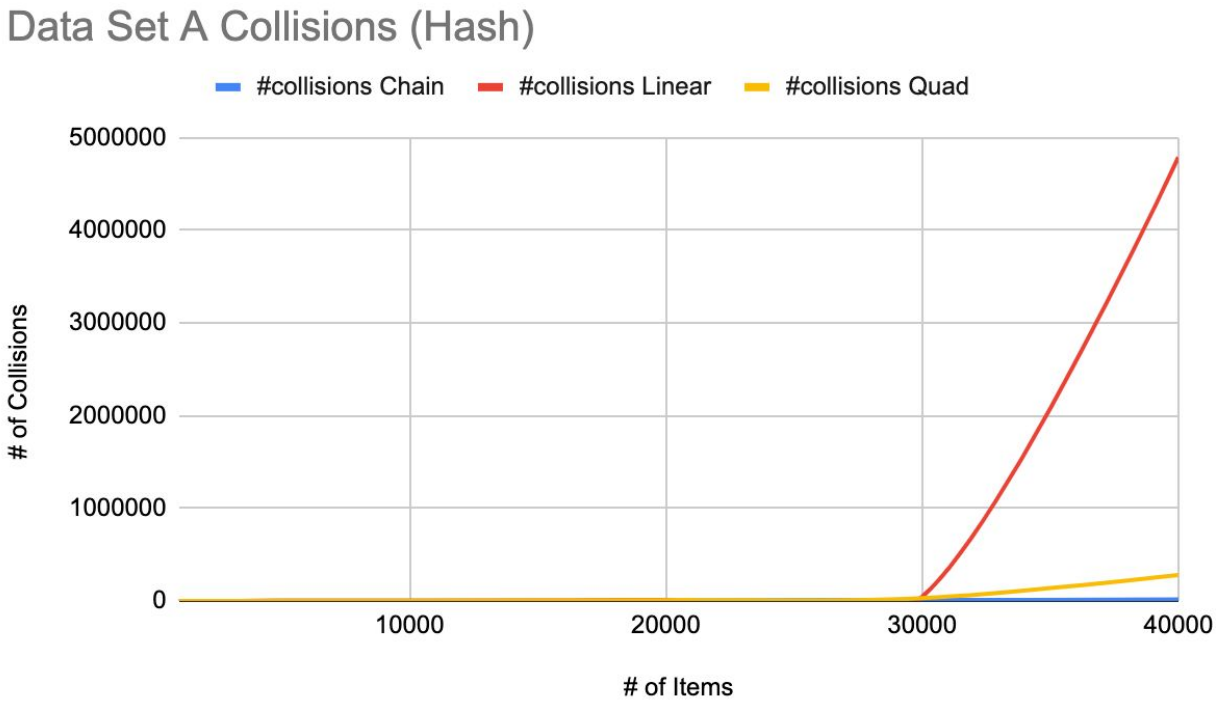

Data Set B (Insertion, Avg Chain, Avg Linear, Avg Quad)
Avg Chain
Avg Linear
Avg Quad
0.0100
0.0075
0.0050
0.0025
0.0000
Time (milliseconds) Per Item
10000
20000
30000
40000
#ITEMS

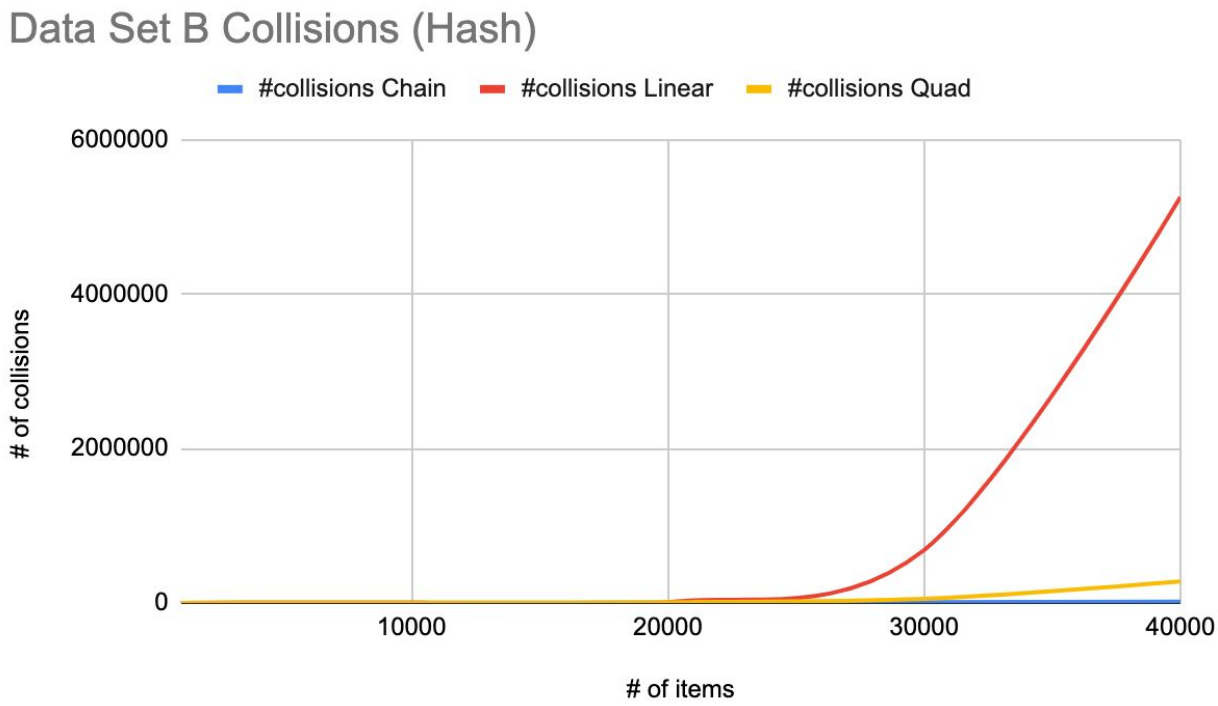
For both data sets, the relationship is very similar. The three variations of the hash table appear to be similar, but in the end, the Chain-collision retains a lower average than linear and quadratic variants do, when dealing with large data sets.





Once again, similar to the insertion, the chain-collision variant very slightly is seen to have lower averages as the data becomes increasingly large. This is likely due to it's comparably low amount of collisions.





In both data sets, we can see that the chain-collision variant has the least amount of collisions, whereas the quadratic begins to grow slightly at the end, and the linear-probing variant spikes heavily. This is likely why the chain-variant seems to win in the long run.

Conclusion:

After completing our data collection, we found that hash tables, in general, work much more efficiently than the standard Linked List or Binary Search Tree. When Inserting / Searching elements in efficient structures, the average time / item tends to flatten out as more items are processed. The Linked List showed a gradual increase in time / item when inserting / searching. Binary search tree out-performed the linked list, however, when comparing with the hash tables, all three hash table variations significantly performed much better than the binary search tree. Among the variations of the hash table, we can see in our figures that the amount of collisions for linear-probing and quadratic-probing begin to jump heavily towards the end of a mass set of data. We hypothesize that this collision

jump is why the chain-variation seems to retain the lowest average time (for insertion and searching), in comparison with the linear and quadratic probing models. We conclude that the hash table, specifically the chaining-variation thereof, is the most efficient data structure of the ones we used to help save the UPS' dire situation.

*When timing small amounts of elements (100) at a time for efficient structures like the BST and Hash Table, there seemed to be an inconclusive set time for how long run-time actually took.