Summary of Data

Data Set A

A screenshot of a cell phone

Description automatically generated

Linked List:

A screenshot of a cell phone

Description automatically generated

While liked lists can be efficient with small data sets, the time it takes to insert an item increases as the data set increases because it must go through every node before it to get to one that is buried within the linked list.

Binary Search Tree:

A screenshot of a cell phone

Description automatically generated

Binary search tree is the most efficient way to search data set A because of the way the data structure branches out from its root/head and makes it efficient and easy to search and insert. Both search and insert can also be performed recursively making them much faster and less costly.

Hash Table:

A close up of a map

Description automatically generated

Linear Probing:

Data Set A has a lot of collisions, so here we see that while linear probing is very efficient with small data sets, as the set becomes larger then there are more collisions and therefore less efficient to go through all the buckets to get to an open slot. That being said, the data is often unsorted, making searching inefficient too.

Quadratic Probing:

With a small data set quadratic probing is the most efficient because it can easily make big jumps in its has index and therefore easier to insert items.

Chaining:

Since data set A has a lot of collisions, we see that chaining is the least efficient in both insertion and searching for collision prone data sets. When there is a collision with chaining, then heaps of data are going to pile onto the linked list at that site making it a mess of data and inefficient piles of linked lists to get through.

Insert Summary

A screenshot of a cell phone

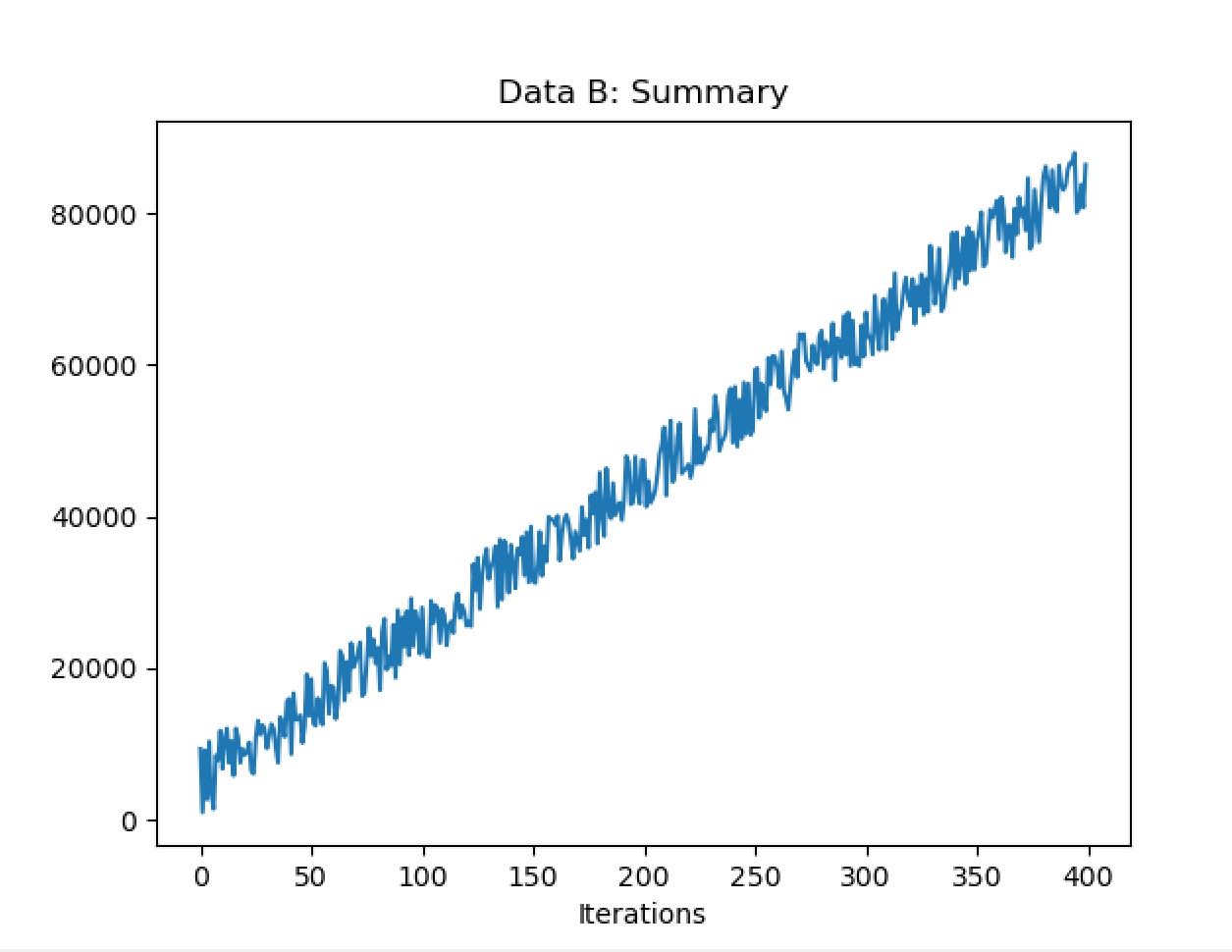
Description automatically generated

Search Summary

A screenshot of a cell phone

Description automatically generated

Data Set B



Linked Lists:

A close up of a map

Description automatically generated

The average insertion time for linked list is very inefficient because the only way to get to a node that is deep in the linked list is by going through every node before that one. Meaning that as the values increase and the insertions increase, more values are buried in the linked list and get increasingly hard to access.

Binary Search Tree:

A screenshot of a cell phone

Description automatically generated

BST is the most efficient for of insertion and searching for data set B because of the way that BST is structured. That is the data structure branches out from the origin in an organized fashion making insertion and search both very efficient and fairly easy to implement recursively.

Hash Table:

A close up of a map

Description automatically generated

Linear Probing:

Linear probing is the most efficient of the Hashing techniques, spiking up to 150 microseconds in this case. Both search and insertion perform similar to that of a linked list, but unorganized.

Quadratic Probing:

Quadratic probing stays at an average of around 150 microseconds and decreases as the hash values increase because when there is a collision the quadratic function can use big jumps to find an available space instead of going through one by one as linear probing does.

Chaining:

Chaining remains at a steady 300-400 microseconds in the higher value inserts fro both insertion and search because both are clumping data into long linked lists in a poorly sorted manner. This therefore takes a lot of time to piece through and find both insertion slots and where something is stored.

Insert Summary

A screenshot of a cell phone

Description automatically generated

Search Summary

A screenshot of a cell phone

Description automatically generated

Conclusion

Considering that the US Postal Service will always need a software that will support large data files and maintain an efficient, low cost insertion and search I recommend a system based in the binary search tree. I say this because of the data laid out above and because of its efficient insertion time and search times.