Save the USPS

Anuragini Sinha
CSCI 2270
Final Project
Prof. Trivedi

The USPS is under strain because of the pandemic the goal of this project is to improve the processing speed of mail and packages by recommending the best search algorithm to find USPS tracking ID. I have considered three algorithms namely, linked list, binary search tree, and hash tables. The USPS is currently using a linked list algorithm to search tracking IDs. I

I have coded these 3 algorithms and will discuss the performance of inserts and searches. We have been provided two datasets A and B. The performance will be reported for both datasets. Finally, we will discuss the reasons for the performance.

Linked List Implementation

The linked list algorithm is being used by the USPS currently. This implementation inserts new data node at the head of the list. Search is performed by traversing through the linked list. This has a time complexity of O(n). All graphs shown have iterations on the x axis, and time on y axis. Time is measured in nanoseconds.

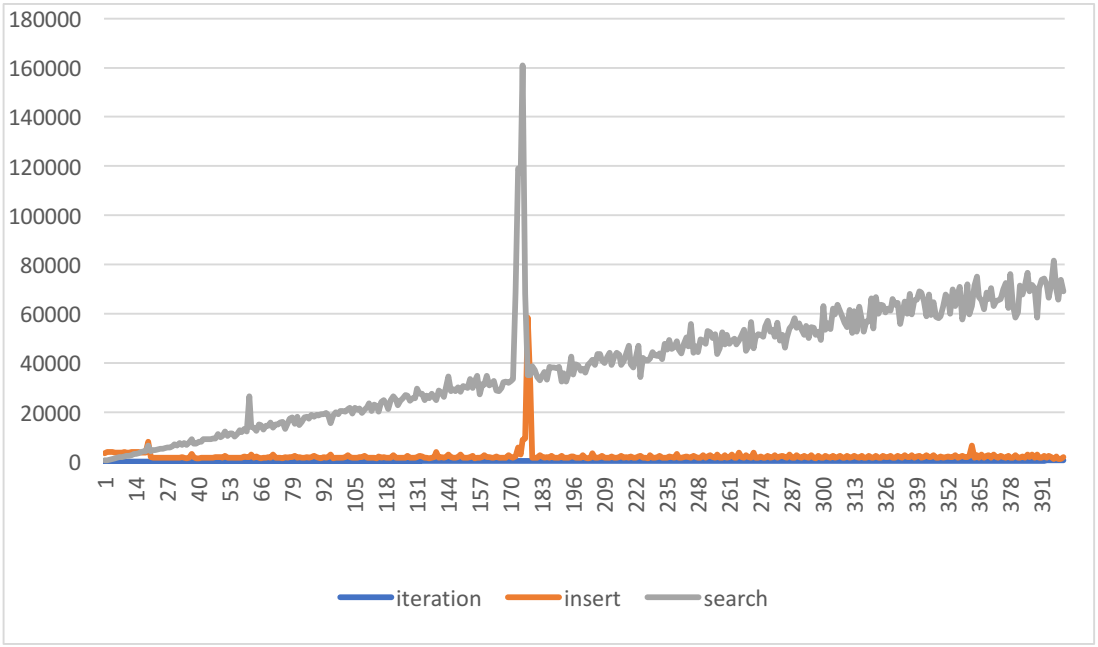

Fig 1: Linked List for Dataset A

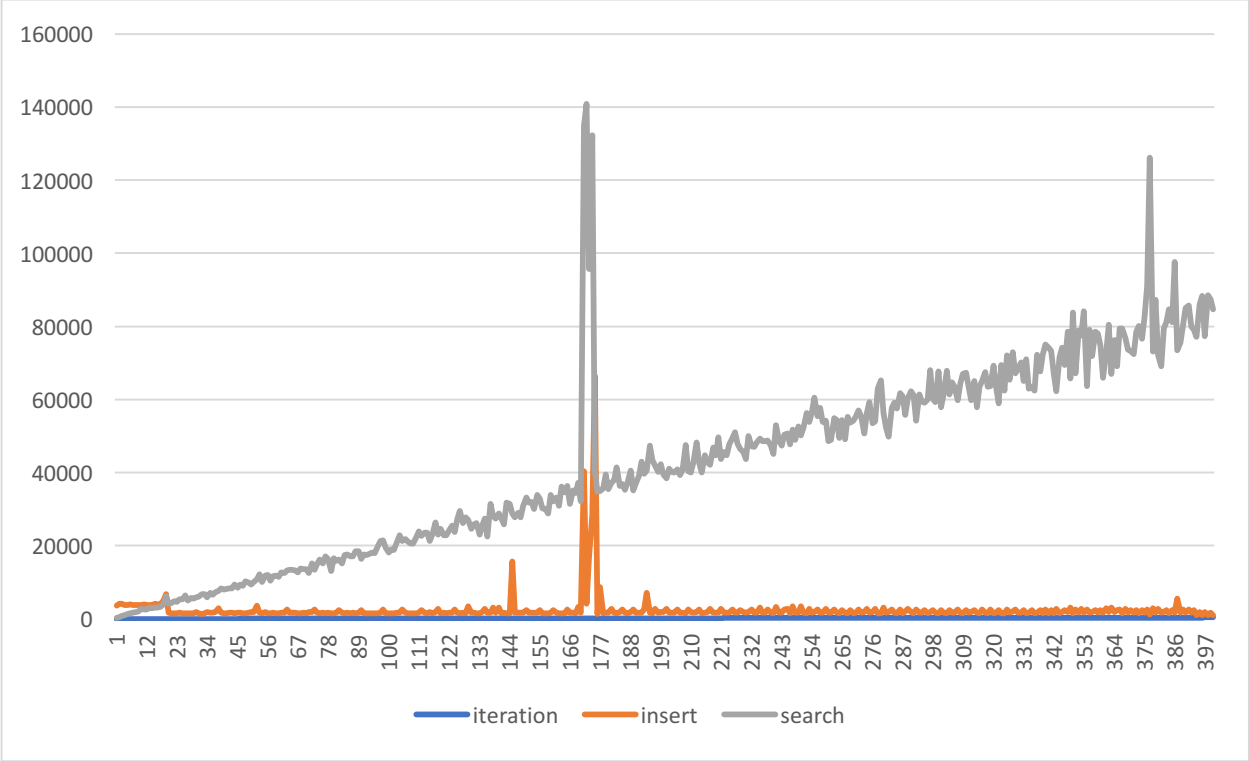

Fig 2: Linked List for Dataset B

We see that the insert time is constant because we are inserting all new nodes at the beginning. This does not require us to traverse the linked list. But the search time increasing linearly suggesting time complexity of O(n). Both dataset A and B behave in a similar manner.

Binary Search Tree

In a binary search tree implementation, a new node is added as a leaf node. The traversal is divided at the root as a result, all nodes do not need to be searched. A well balanced binary tree will roughly divide the data into two equal parts. This provides a time complexity O(logn).

As seen in the figure 3 and 4, the insertion and search shows log(n) time complexity. There are some spikes. This suggests that the tree is not well balanced and there are some data points whose search or insertion requires more time.

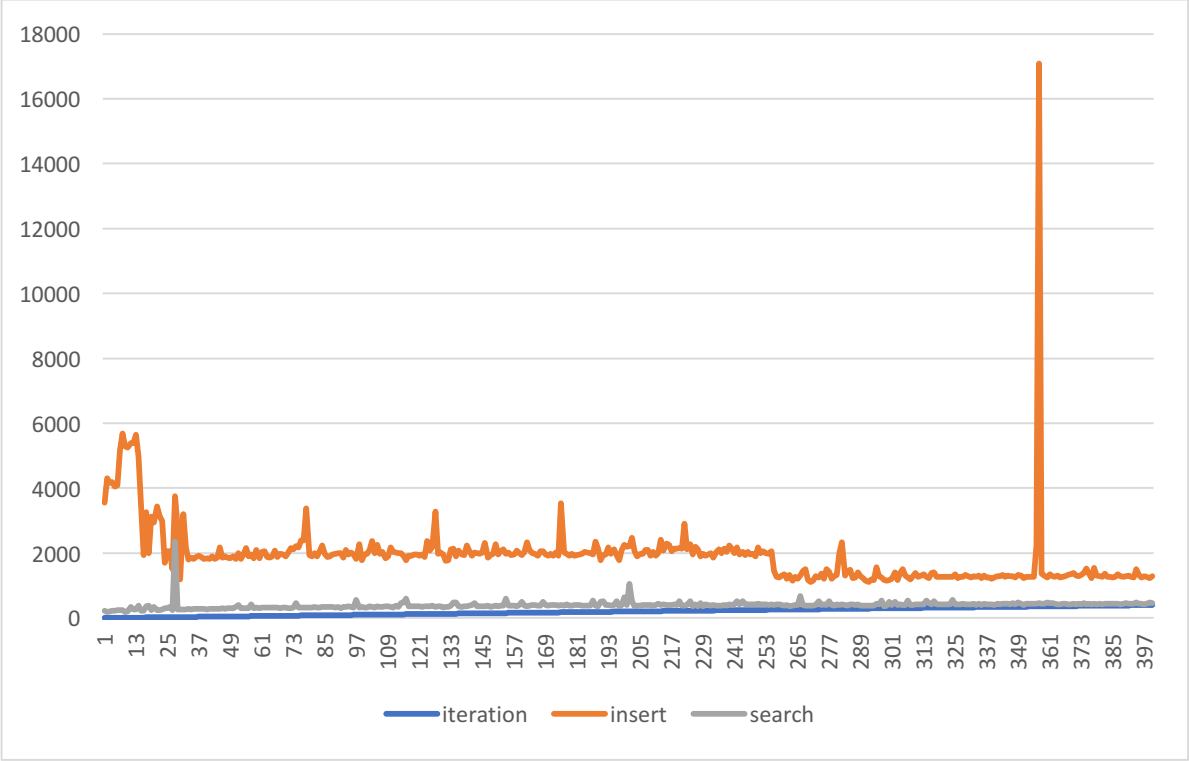

Fig 3: Binary Search Tree for Dataset A

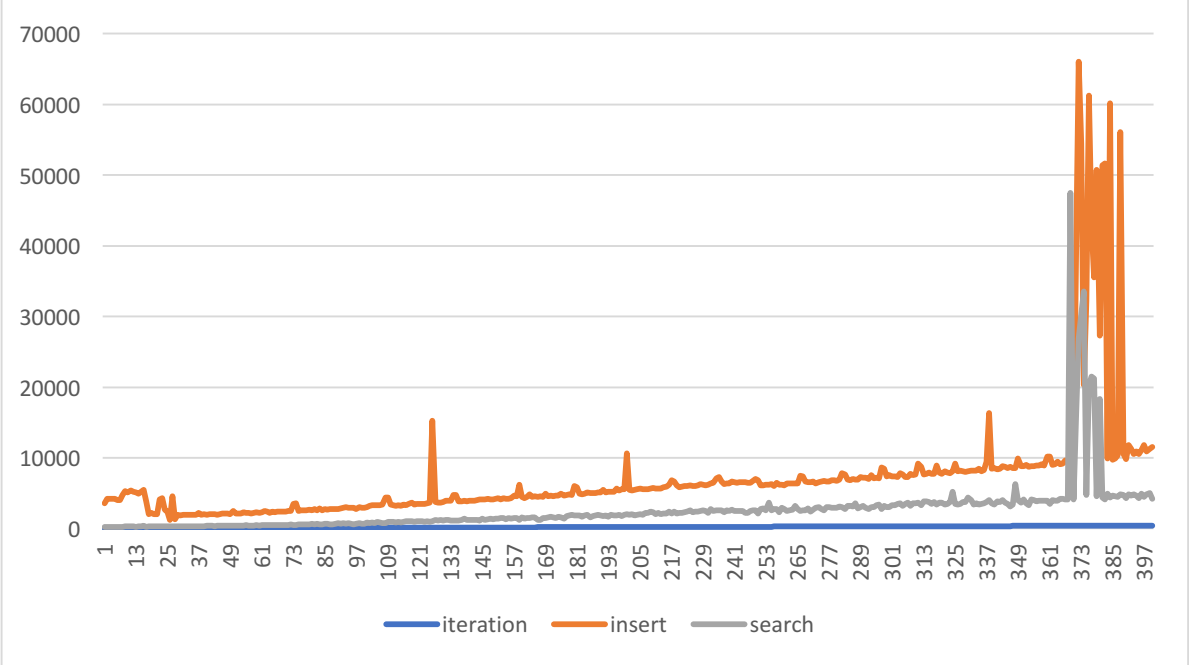

Fig 4: Binary Search Tree for Dataset B

Hashtable Linear Probing

This algorithm implements an array of nodes that have a key and value property. The key is used to access the value. In linear probing, if there is a collision, the next available index in the array is used to store that value. If the last index of the array is reached, it goes back to look at the first index of the array and continues to search from there for an available index.

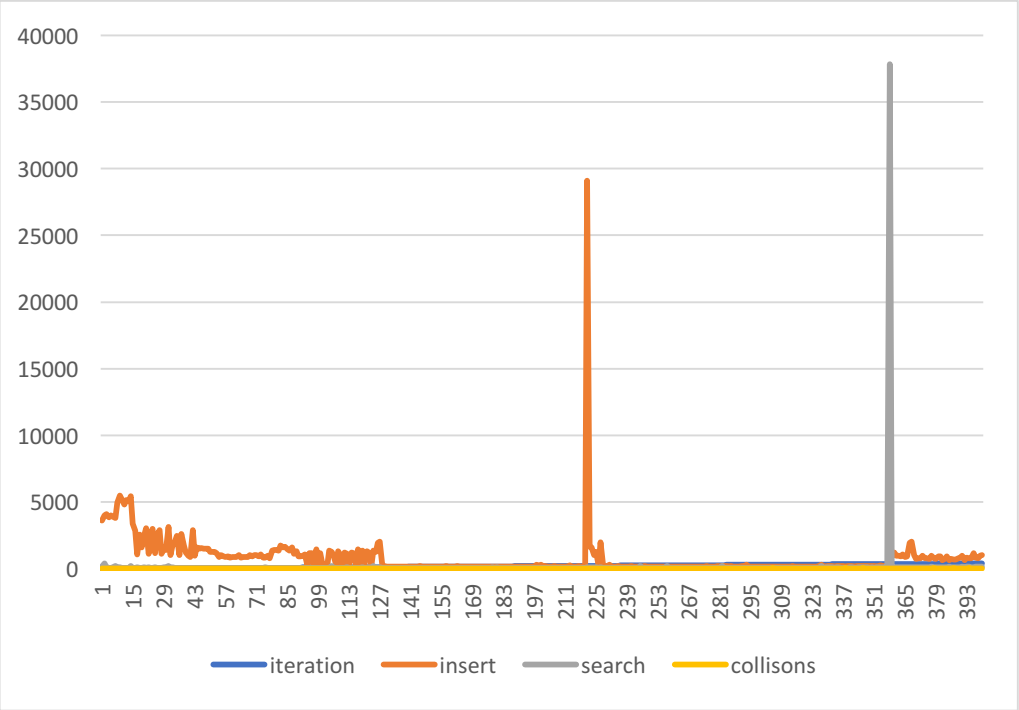

Fig 5: Hashtable Linear Probing for Dataset A

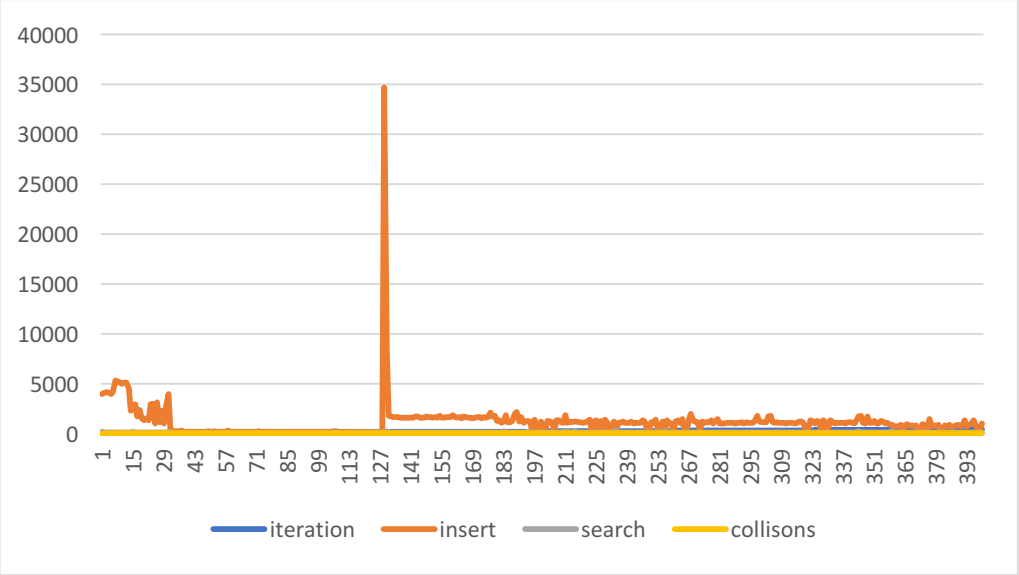

Fig 6: Hashtable Linear Probing for Dataset B

As seen from figure 5 and 6, the hash table insert and search are significantly faster than both linked lists and binary search tree implementations. The time taken is for example, less than 1000 nanoseconds for Hashtable with linear probing, 5000 nanoseconds for binary search tree and more than 50,000 nanoseconds for linked list. The time spikes towards the end because the array is mostly filled up, and there are many collisions requiring them to be resolved to find an empty index location in the array.

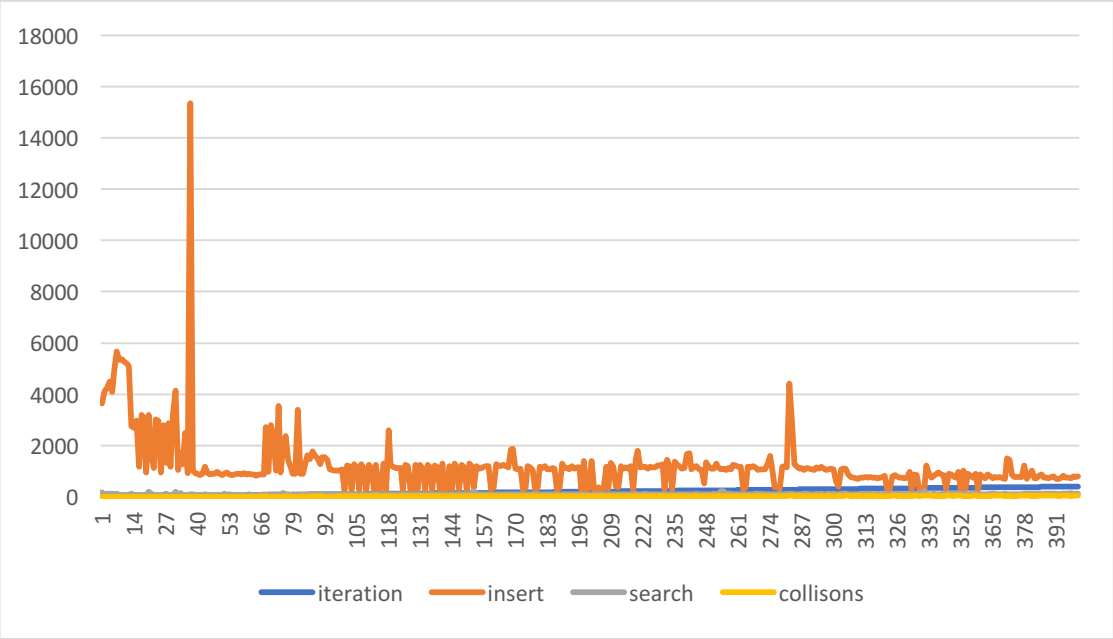

Fig 7: Hashtable Quadratic Probing for Dataset A

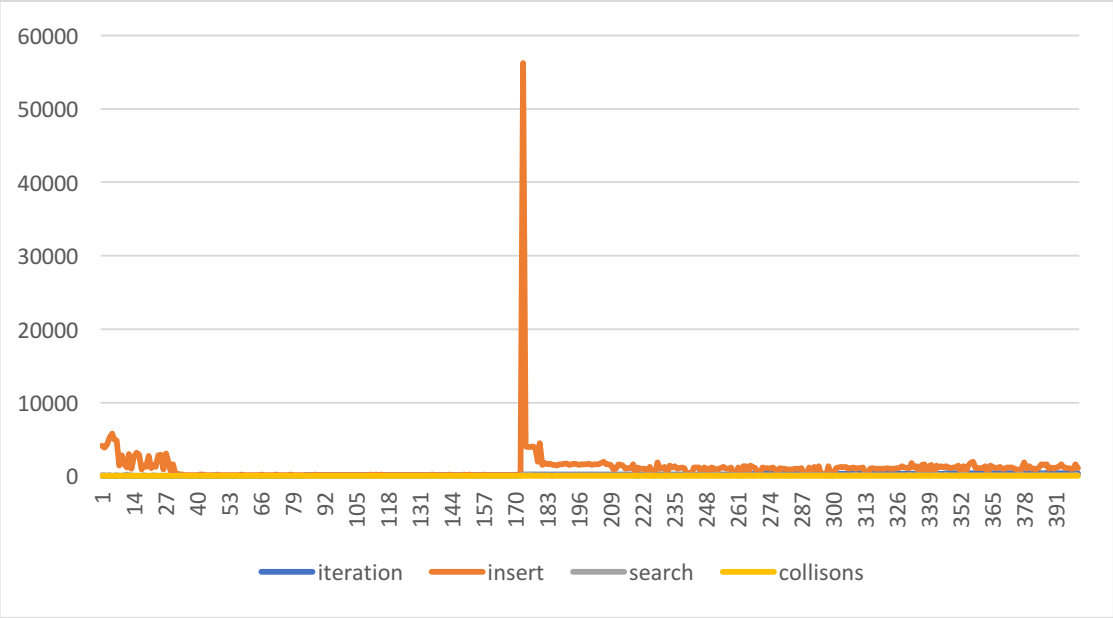

Fig 8: Hashtable Quadratic Probing for Dataset B

Looking at figure 6 and 7, it seems dataset B is well organized, it could be better sorted than dataset A. The quadratic appears to perform better than the linear probing for these datasets. This could be because quadratic takes bigger steps to find available index positions to store values in case of a collision.

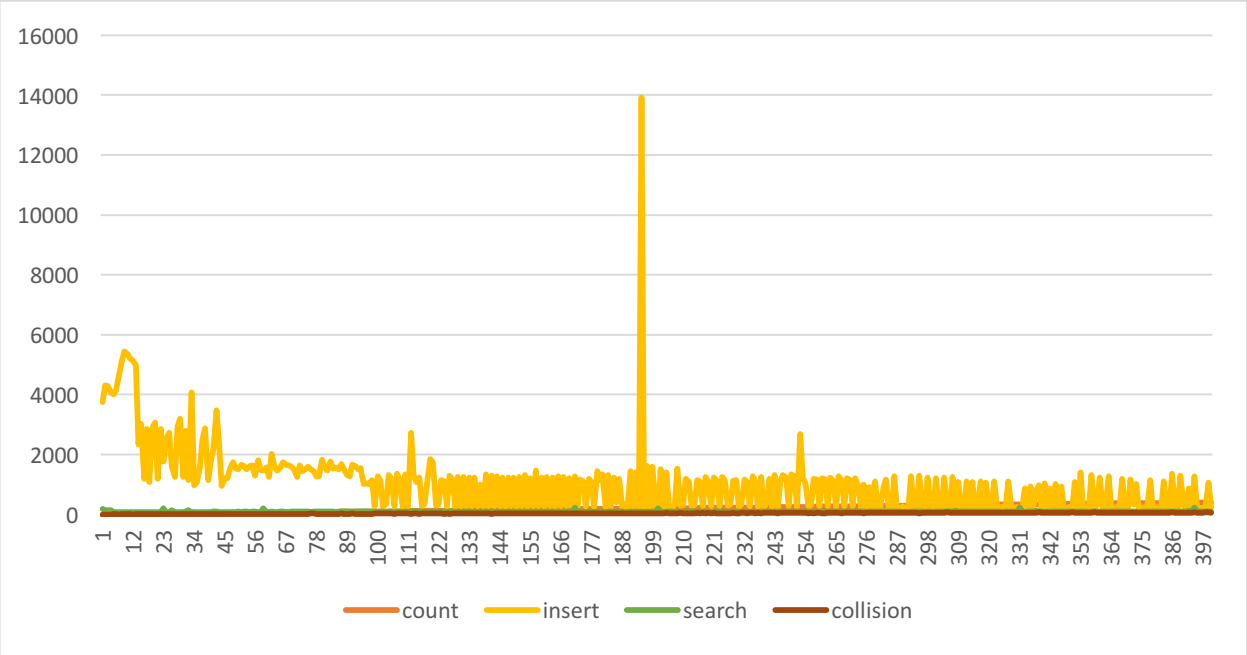

Fig 9: Hashtable Chaining for Dataset A

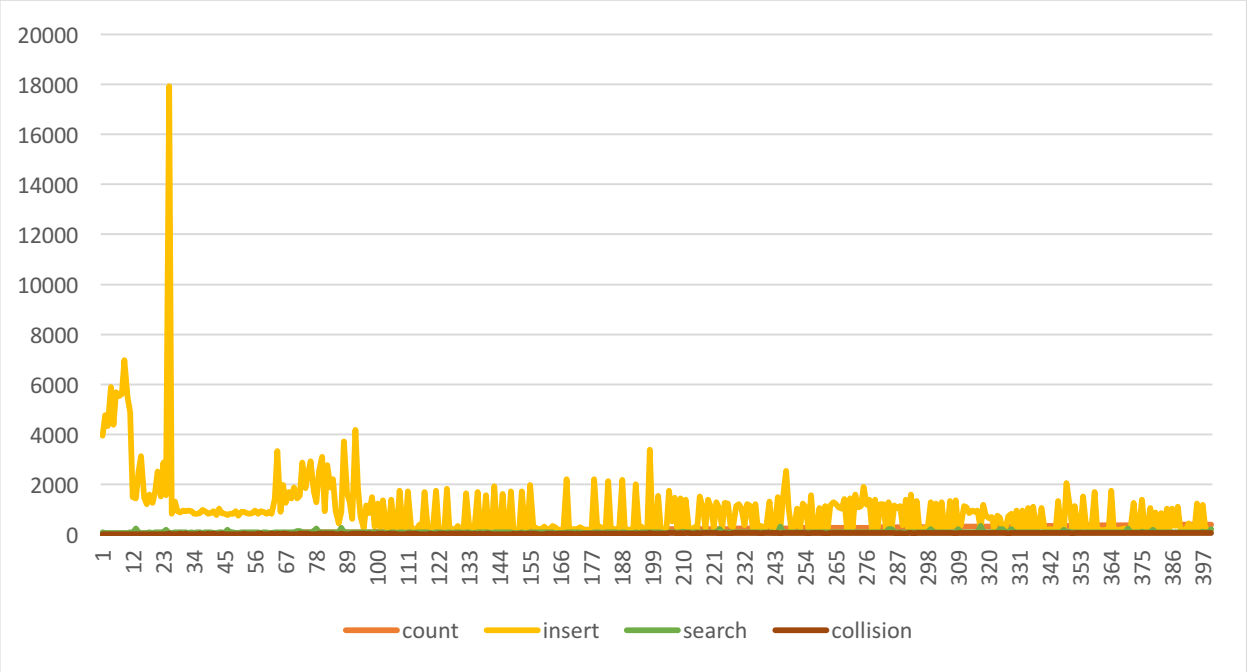

Fig 10: Hashtable Chaining for Dataset B

Hashtable chaining creates a new node at the index where collision if encountered. Initially, these collisions are few but as the number of elements added increase with latter batches (such 300th, 301th etc.) the number of collision increases.

Comparison of Algorithms

BST insert time increases as the tree grows because of the time taken to traverse the large tree to insert new nodes as leaf node. Linked List implementation is slower than Hashtable but faster than BST. This is because we are inserting new node in the linked list at the head position. If we had to traverse the linked list and insert them at tail, this time will be longer.

The following graph in Fig 11 and 12 shows insert time comparison among three search algorithms. It is evident that BST (binary search tree) is the slowest here. The linked list insertion is happening at the head, as a result this is a faster implementation here. However, if we chose to insert a new item at the tail of the linked list, this would have been the slowest implementation. The hashtable insertion time is the fastest.

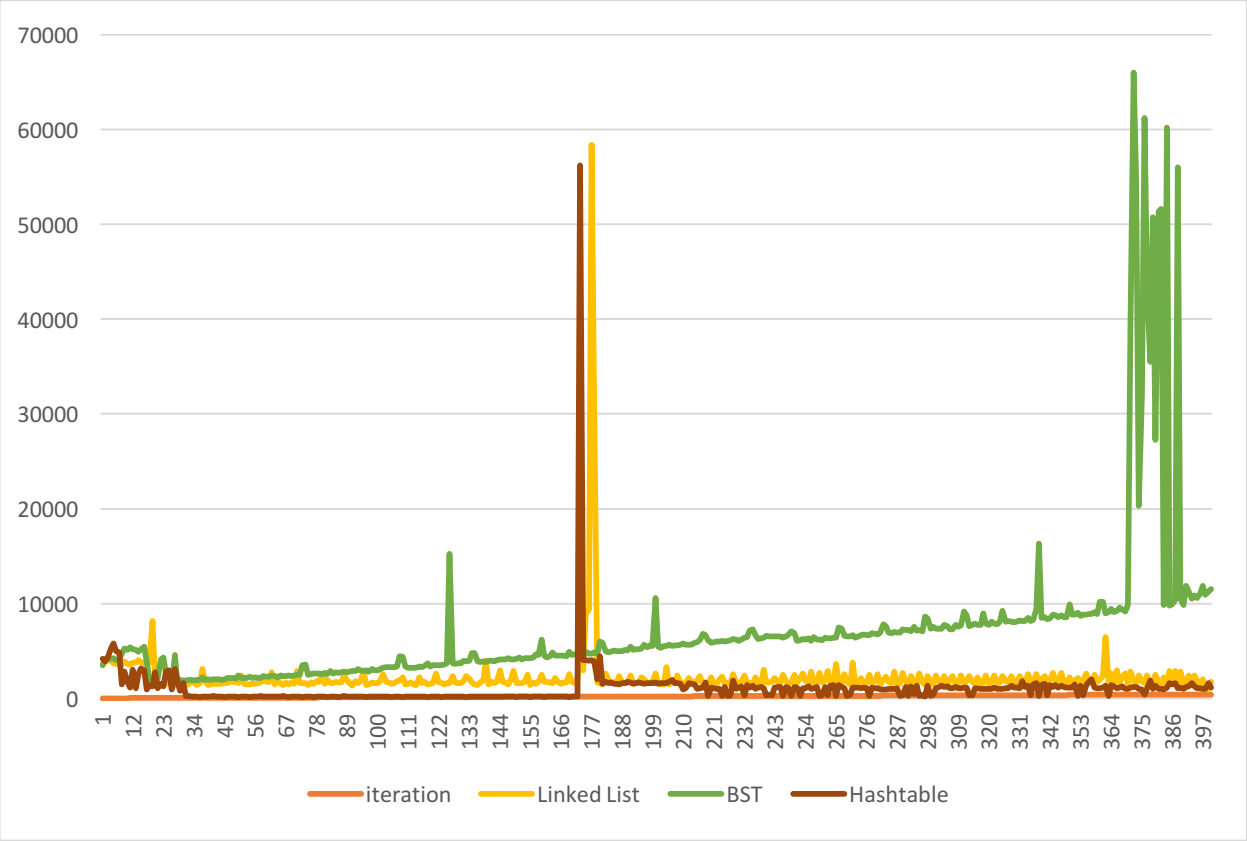

Fig 11: Insert Time Comparison

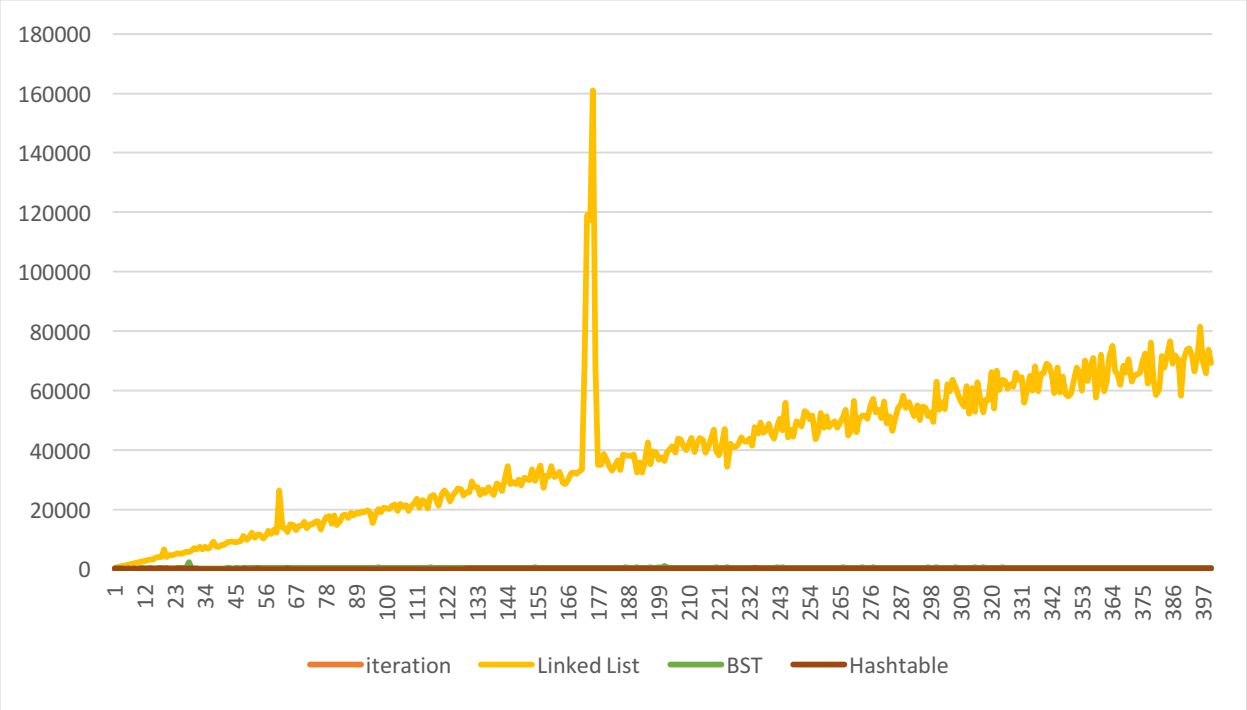

Fig 12: Search Time Comparison

It is clear from figure 11 and 12 that Hashtable insert and search are the fastest for the datasets considered. In any case, Hashtable search has a time complexity of O(1) which means that this the fastest algorithm to search data. This is followed by the binary search tree. It is evident that the linked list algorithm performs the worst in searches. In conclusion, we highly recommend USPS to implement Hashtable with quadratic search to store and search data to improve its operational efficiency.