**CSCI 2270: Data Structures Final Project**

**Nathan Straub and Pourna Sengupta**

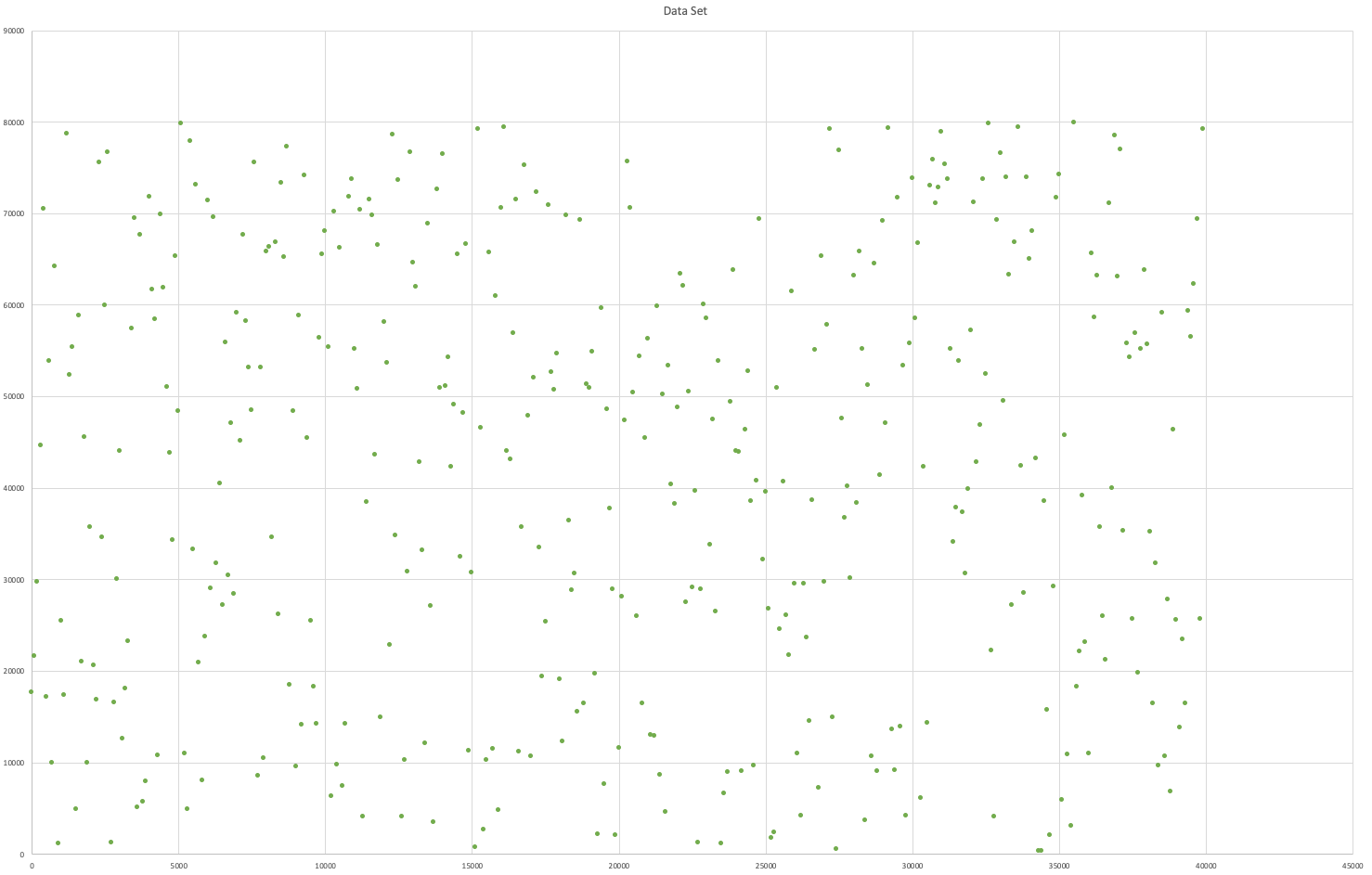
**April 28, 2020**

Of the three structures, Linked List, Binary Search Tree, and Hash Table, it was found that the Hash Table (Quadratic) performed the fastest overall. No matter if the data is ordered or scattered, the Quadratic Hash Table is able to insert the first hundred data points within 100 nanoseconds of the same speed as the last hundred data points. Even better, the Quadratic Hash Table can produce essentially the same runtime to search for 100 different data points out of 100, then it can for 100 different data points out of 40,000.

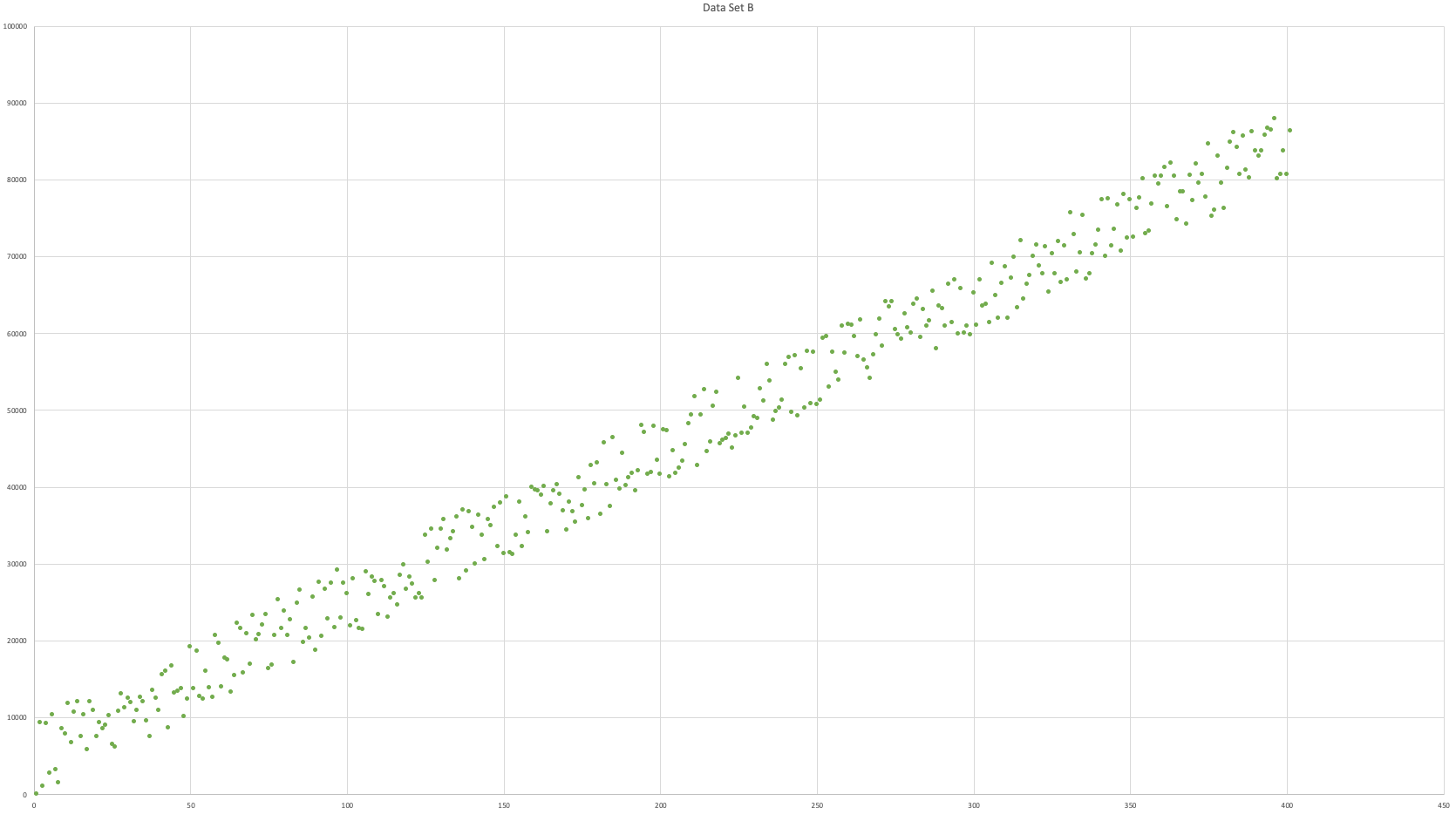
While other structures like the Binary Search Tree have similar runtimes for insertion and search functions, the BST doesn’t do particularly well with data that is ordered. This makes sense because if the data falls within a couple of digits of one another (and the pattern is continued as the indices get larger), the tree will have to be recursively called more and more times both to the right and left in order to find either where the key is. Another structure that has similar insertion and search runtime is the Chain Hash Table. Although the Chain Hash Table has better insertion time for ordered data (on average), the Chain Hash Table’s biggest flaw is insertion for data sets that are scattered. This makes sense since scattered data can be both big and small in the size of the number at any particular indices, which means that there’s more of a chance that multiple of the numbers share the same bucket. Anything to do with a Linked List takes significantly more time because you have to traverse the entire list before inserting the node at the end and you must check for each node with no sense of direction or order (just down the long pointed list).

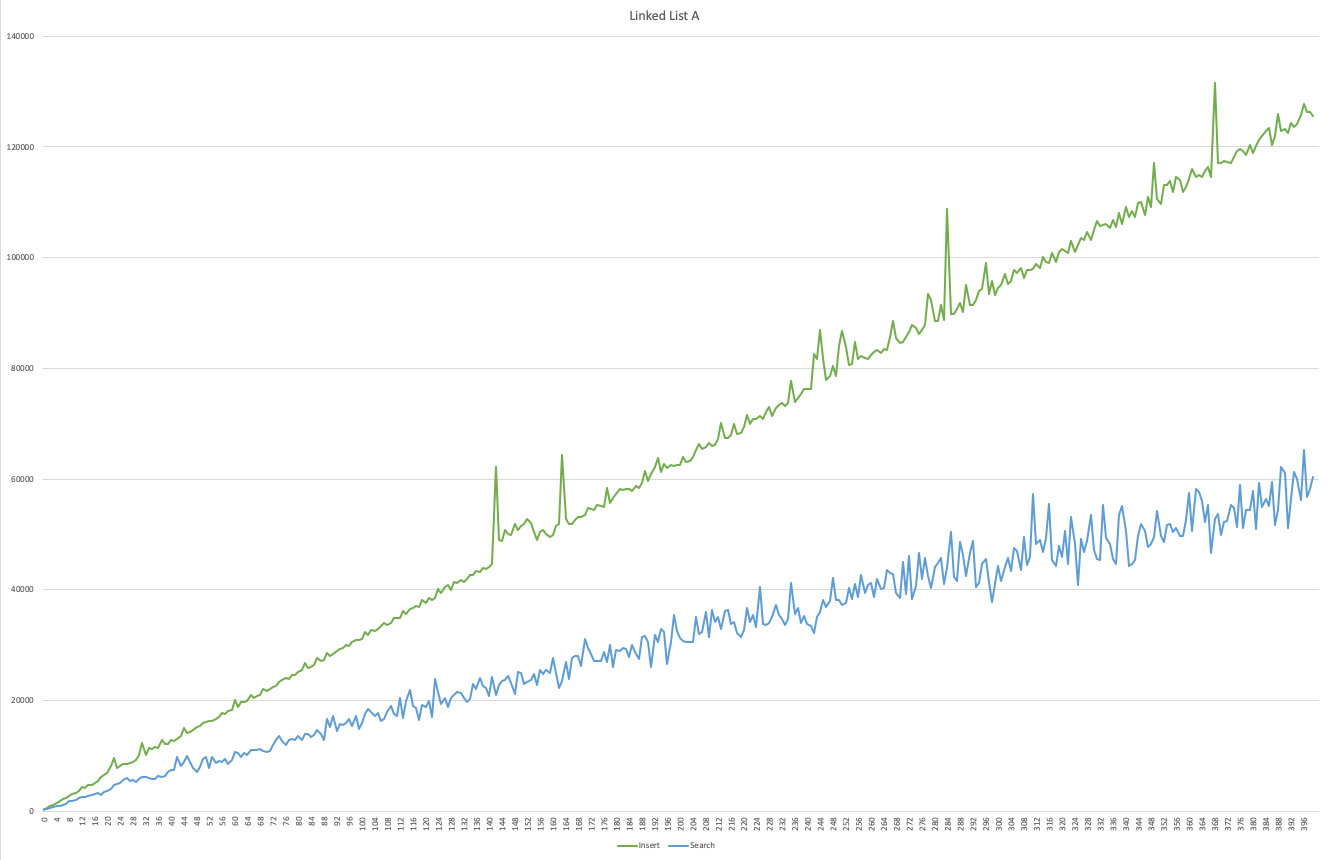
As more and more data points were added to each of the three structures listed above, the runtime it took to search for any random 100 points in the structure took significantly more time than before when there were fewer points. Along with this, in all cases except Hash Tables, the structures took longer to insert the first 100 points of data rather than search for any random hundred points. This seemed to hold true for the BST and Linked List as more and more data points were inserted into each structure. The only reason the Hash Tables were able to have better than or equal to search runtime as compared to insertion is because of how the structure is set up itself. Since each index is specific to the value of the data, this is able to pinpoint the location of the value you are searching for and is then able to locate the value by either linked list or bucket hopping (linear or quadratic).

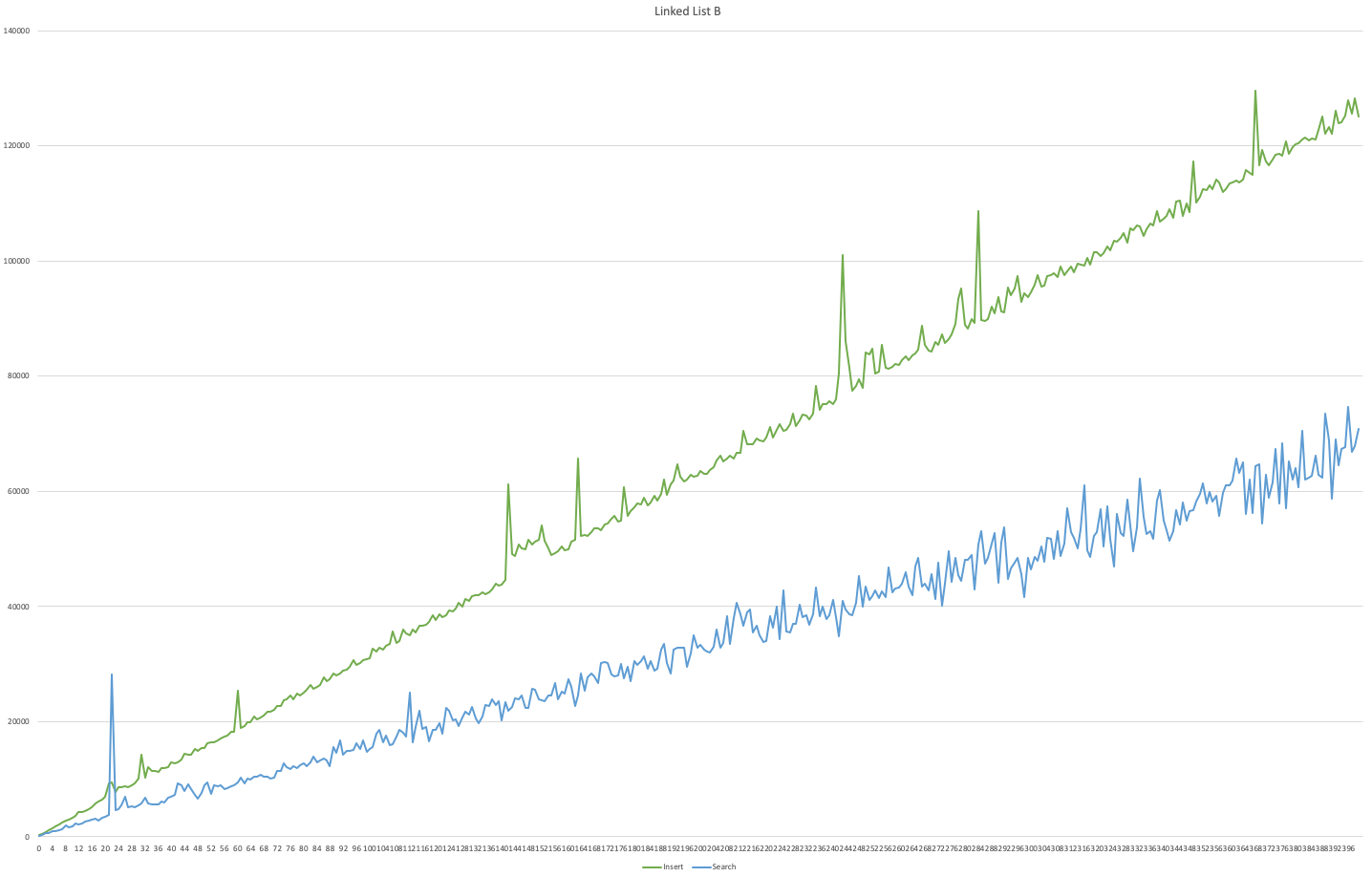
The Binary Search Tree ran faster than the Linked List, making the Linked List the slowest structure out of the three. The run times for indices 1-400 were measured and averaged. The data was then plotted in the graphs below.

**Data Set A (Scattered Data)**

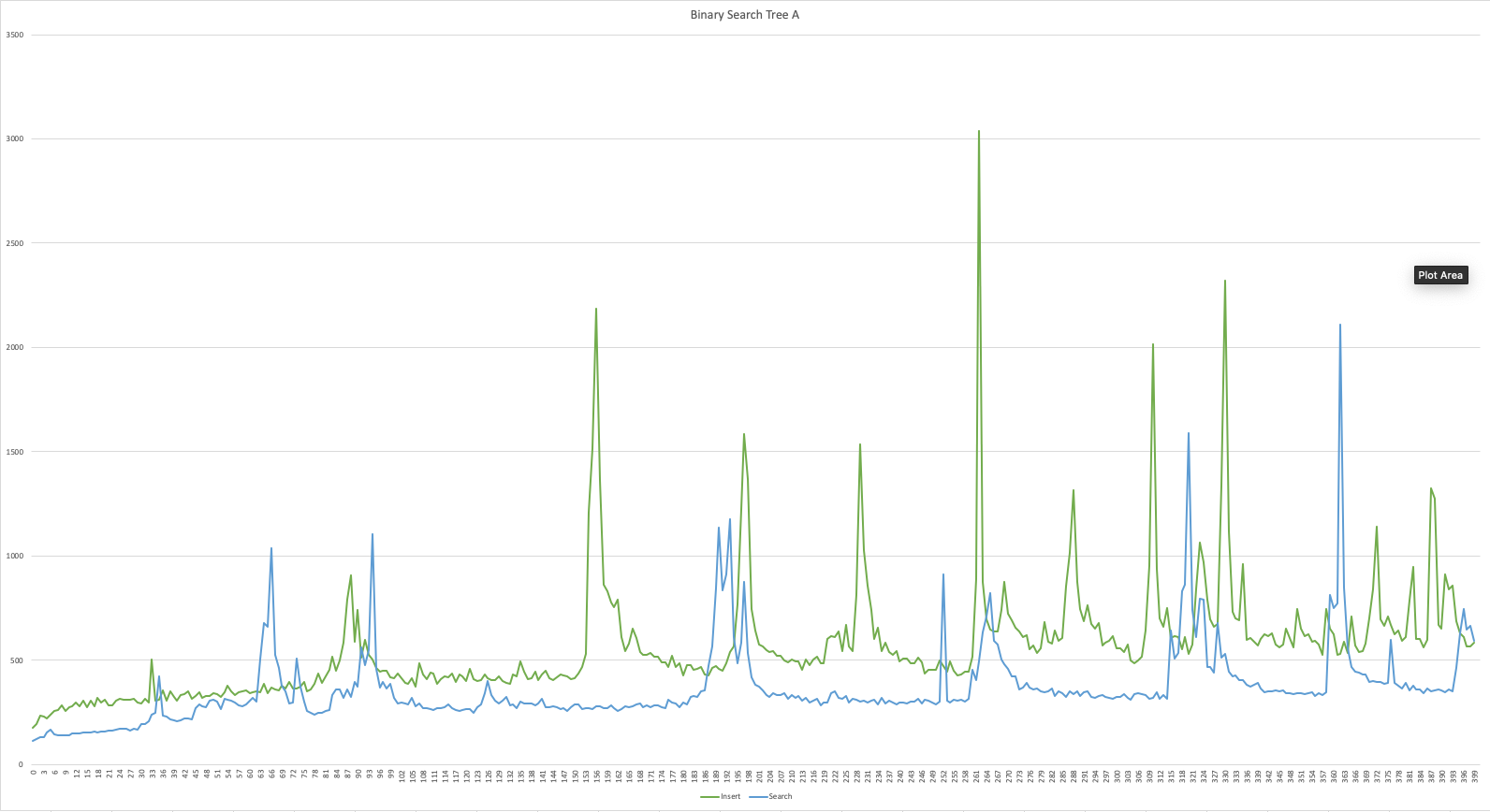
**Data Set B (Ordered Data)**

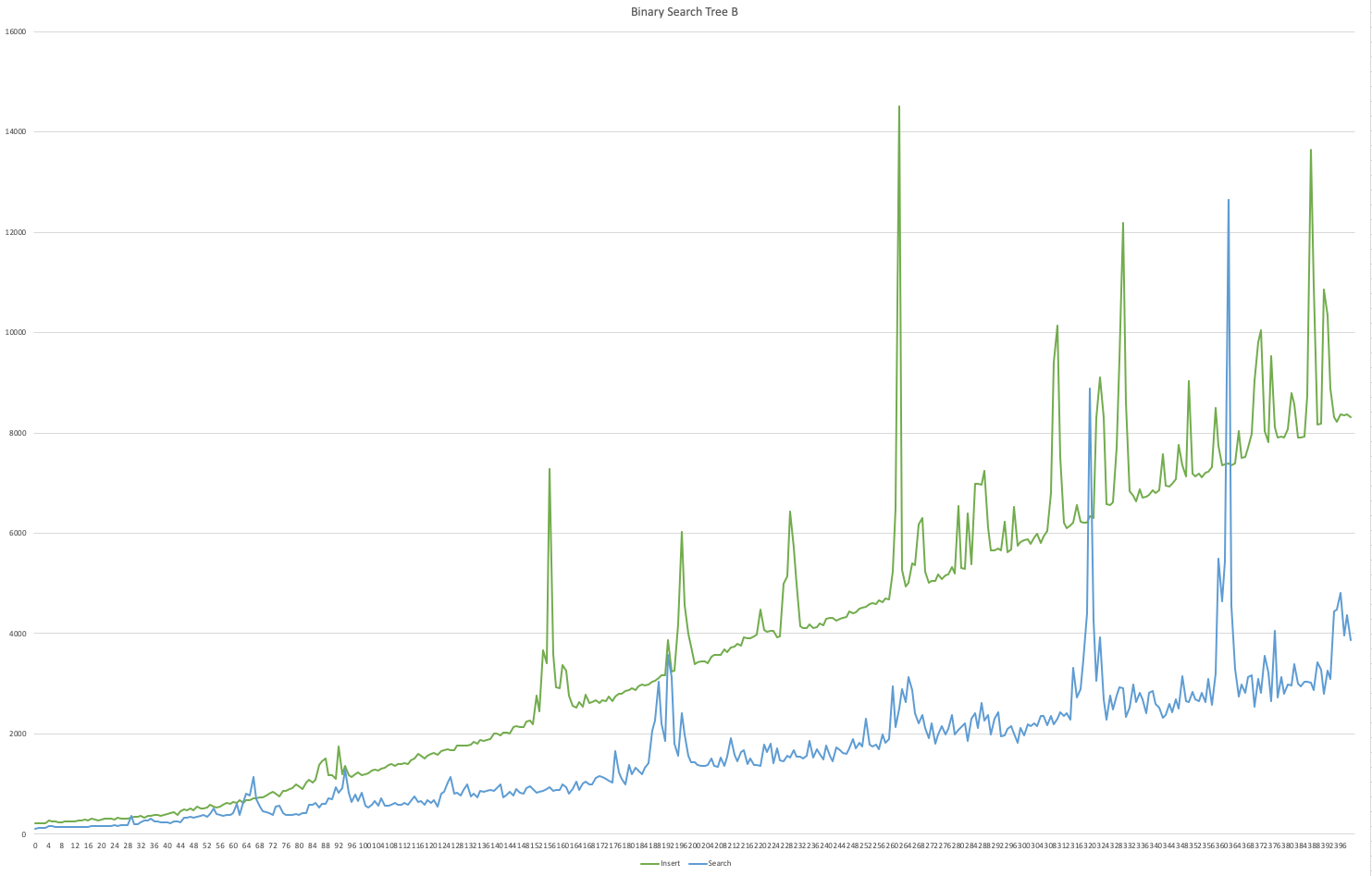
****

**Linked List A (Scattered Data)**

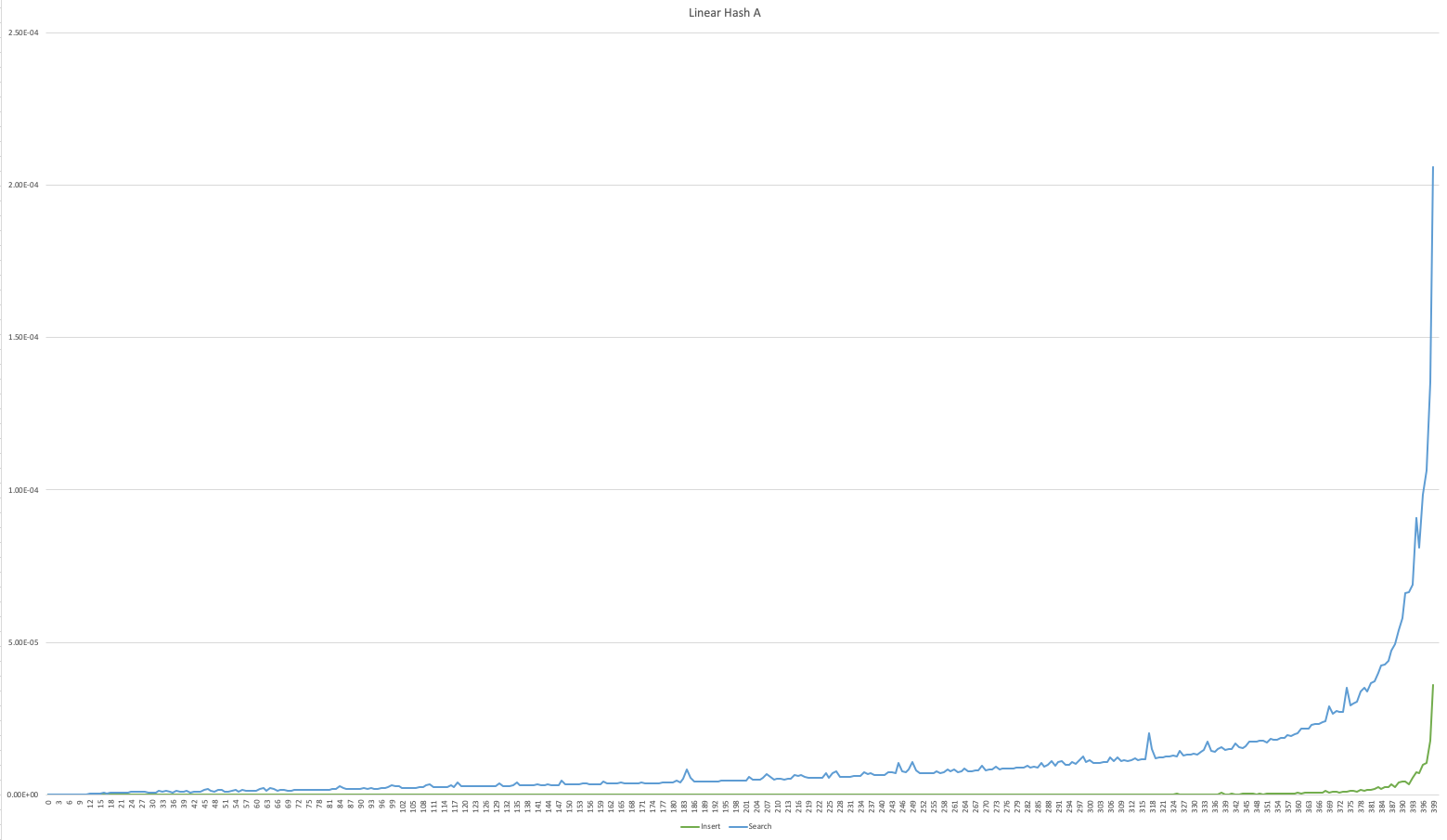
**Linked List B (Ordered Data)**

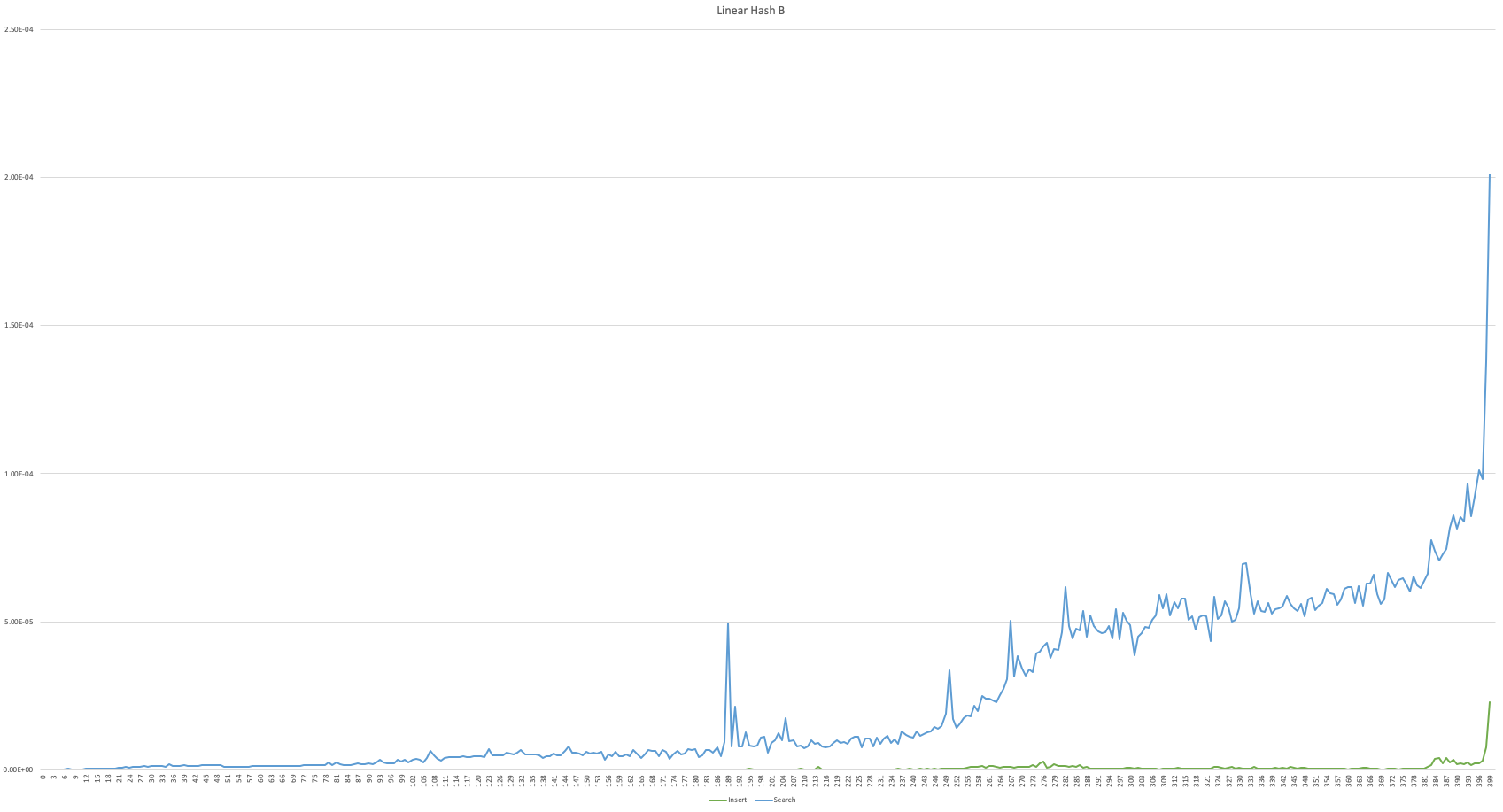
**Binary Search Tree A (Scattered Data)**

****

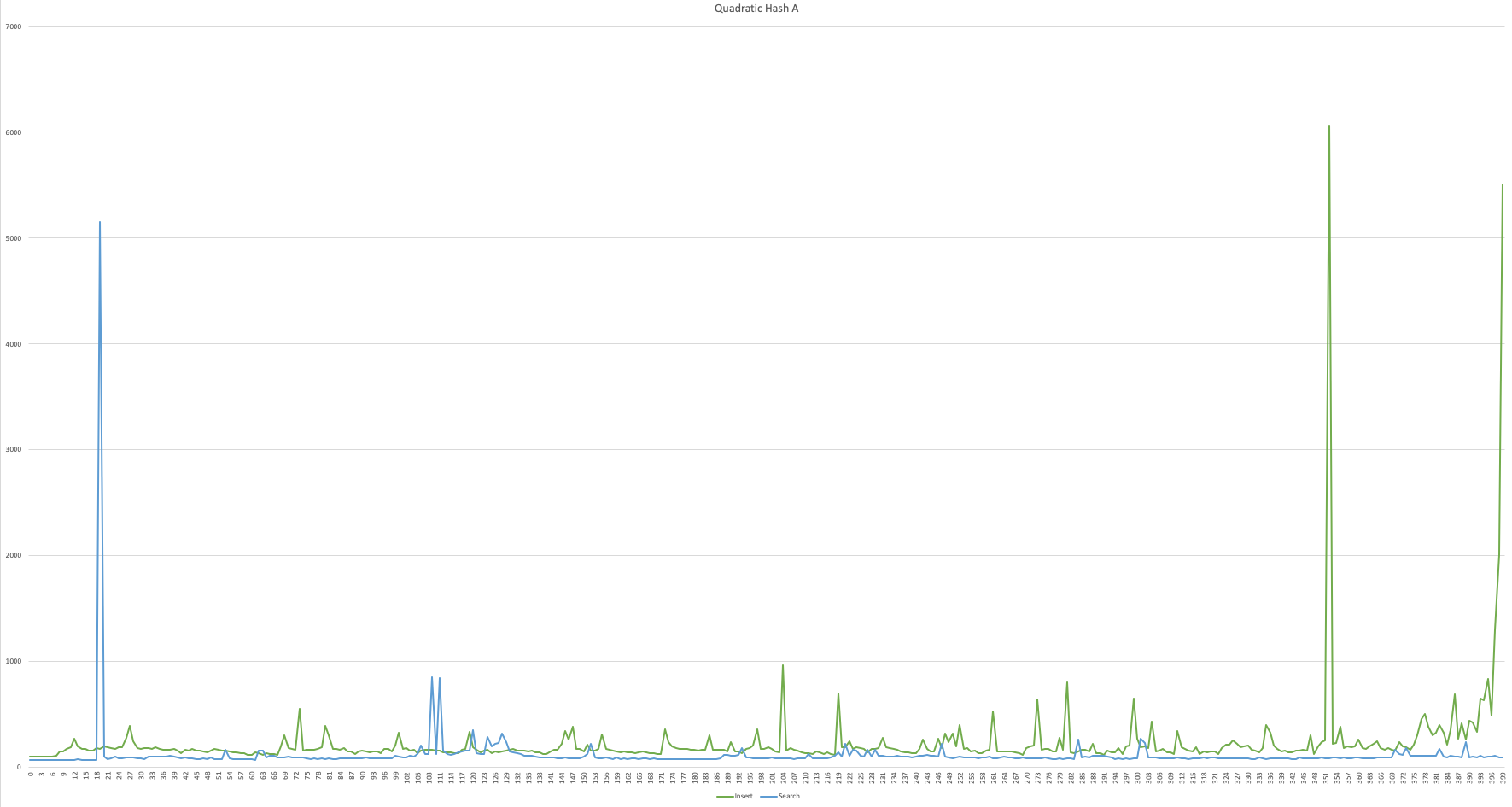
**Binary Search Tree B **

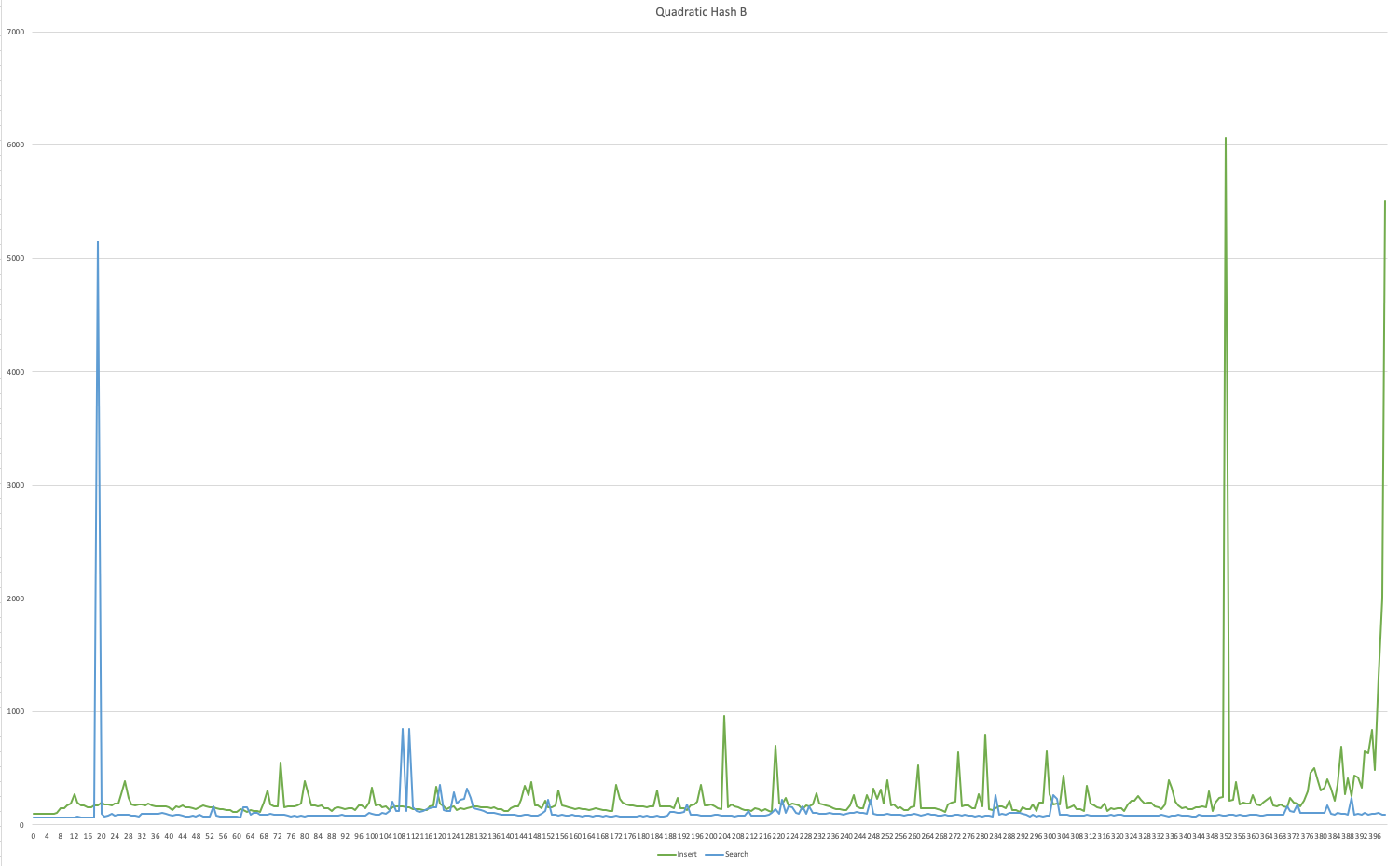
**Linear Hash Table A (Scattered Data)**

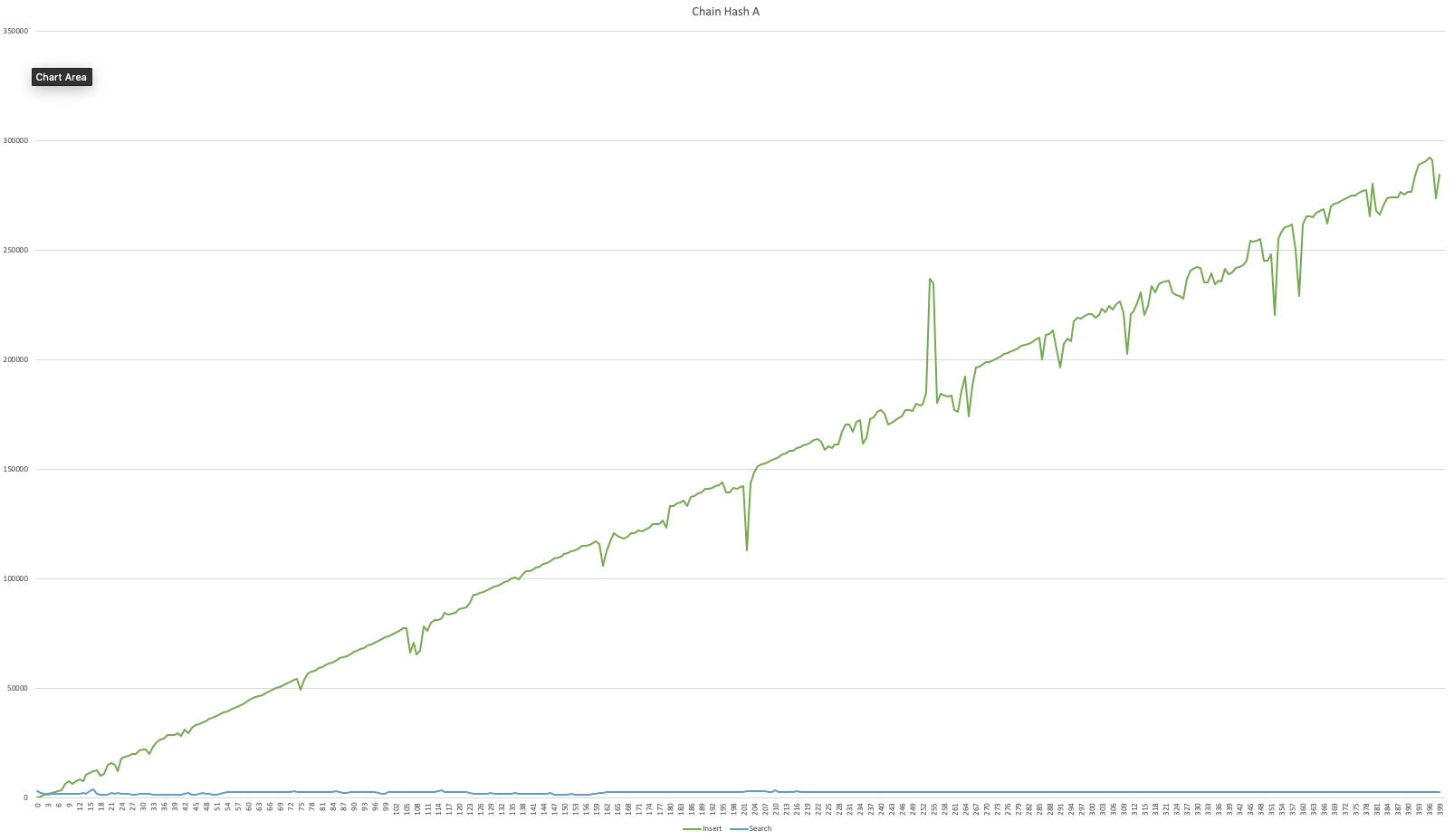
****

**Linear Hash Table B (Ordered Data)**

**Quadratic Hash Table A (Scattered Data)**

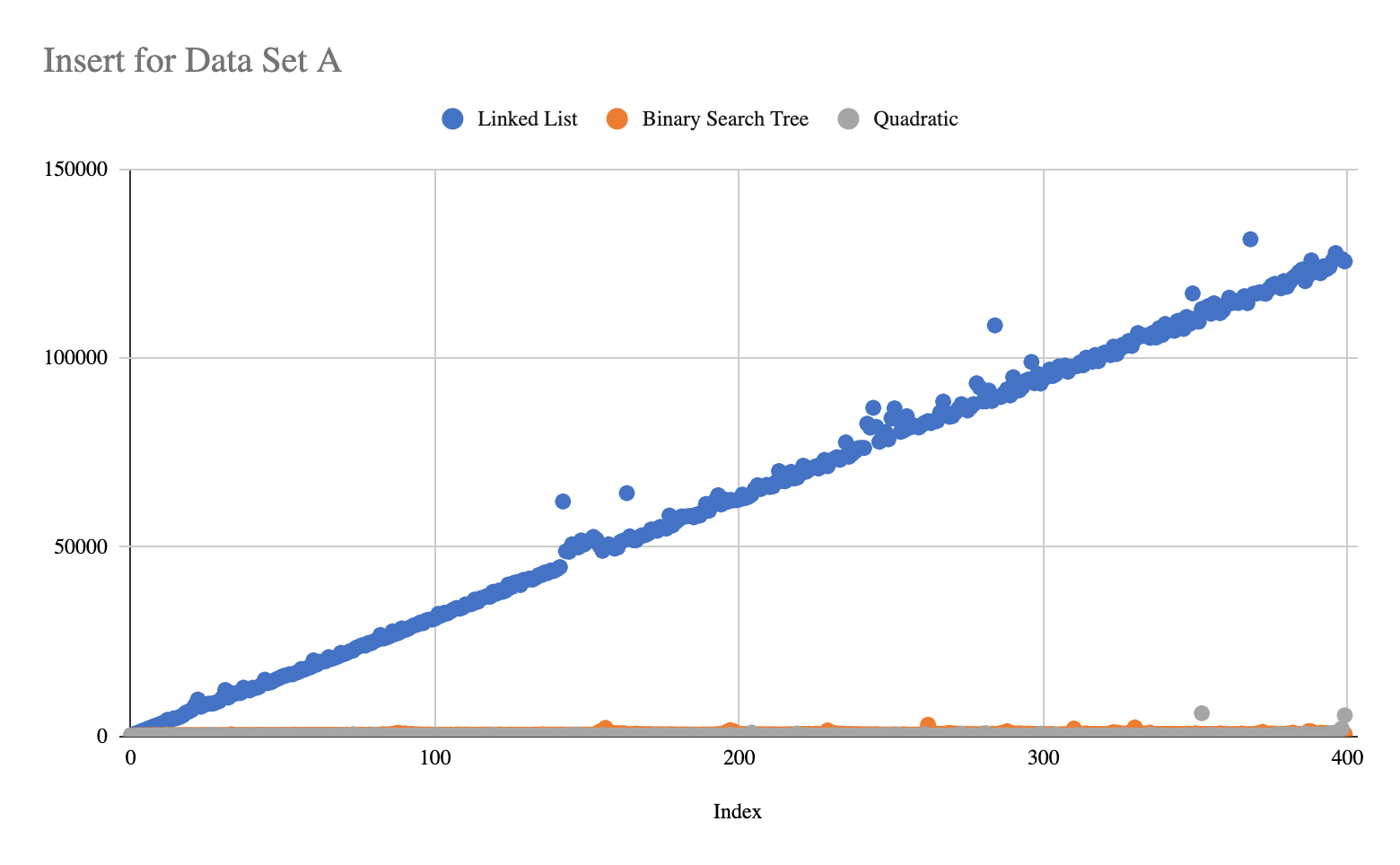
****

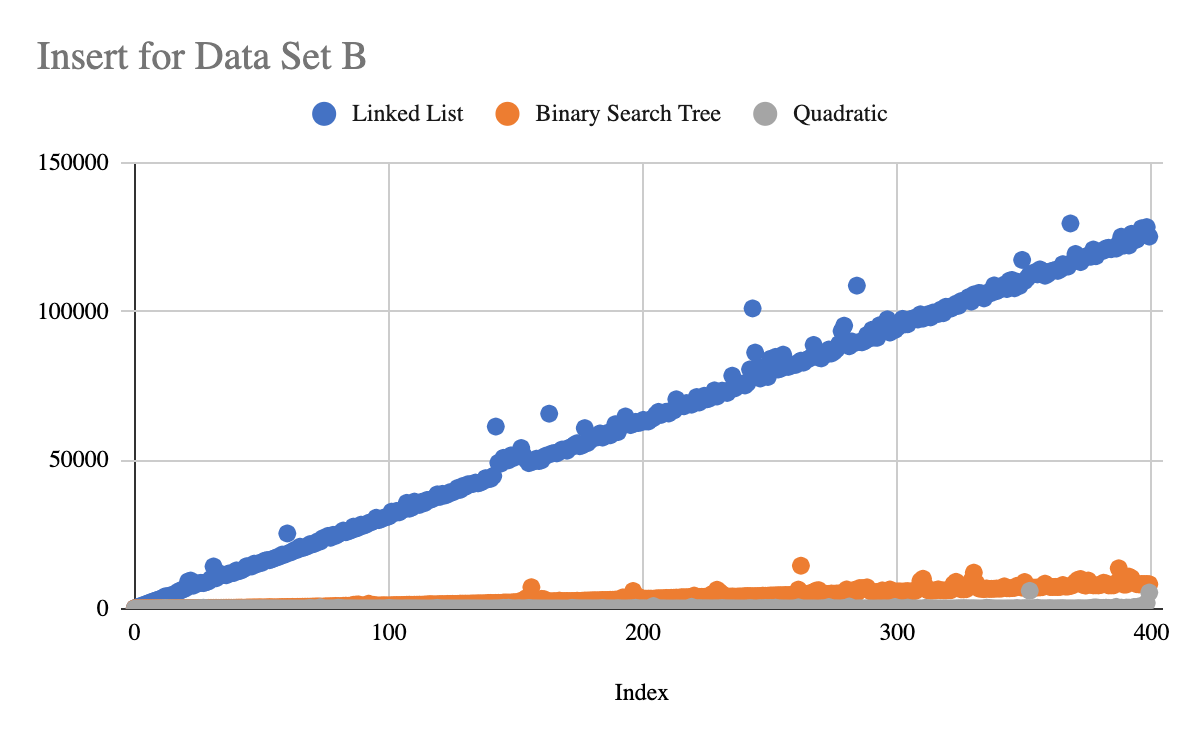
**Quadratic Hash Table B (Ordered Data)**

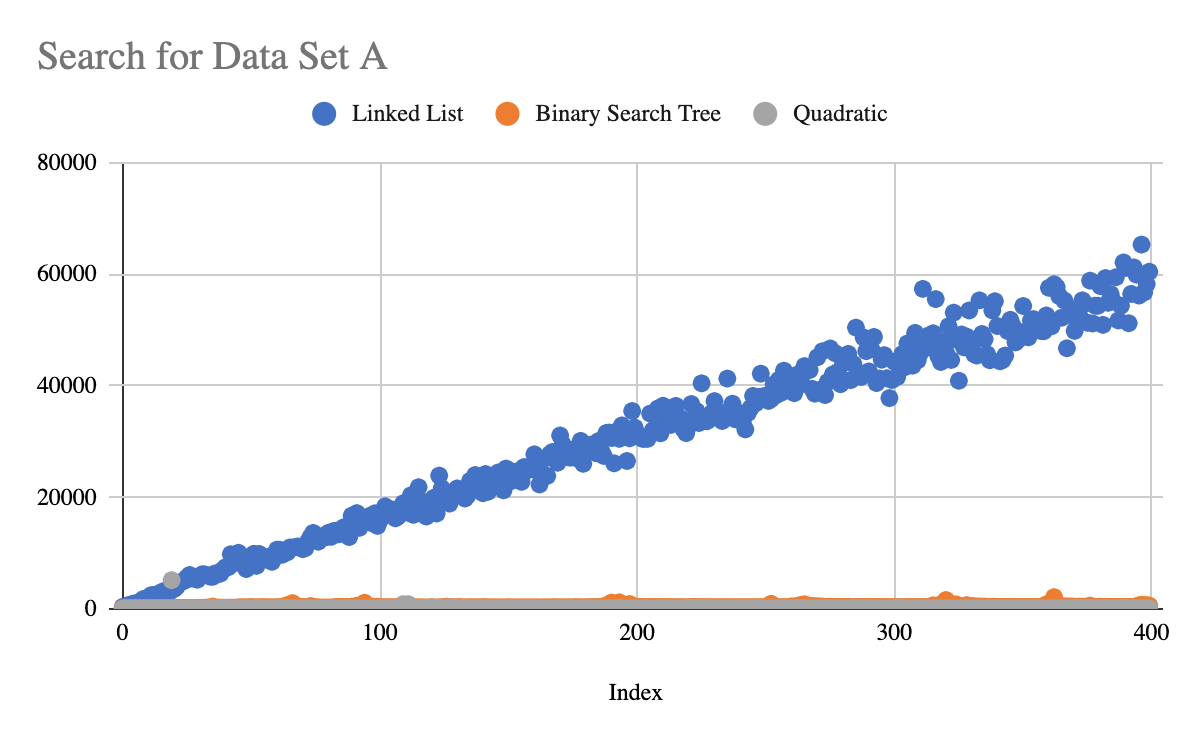
**Chain Hash Table A (Scattered Data)**

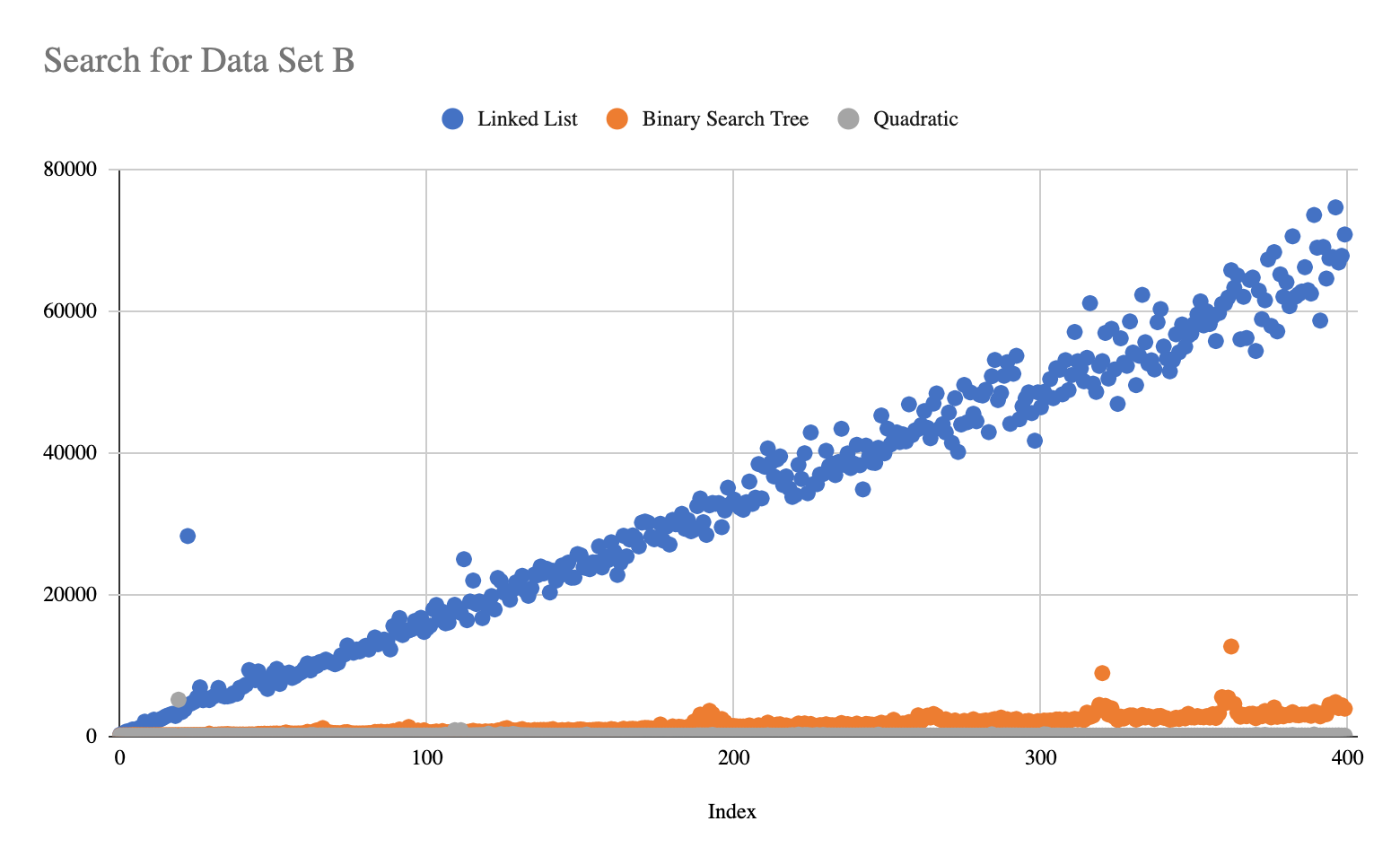
**Chain Hash Table B (Ordered Data)**

****

****

****

****

****