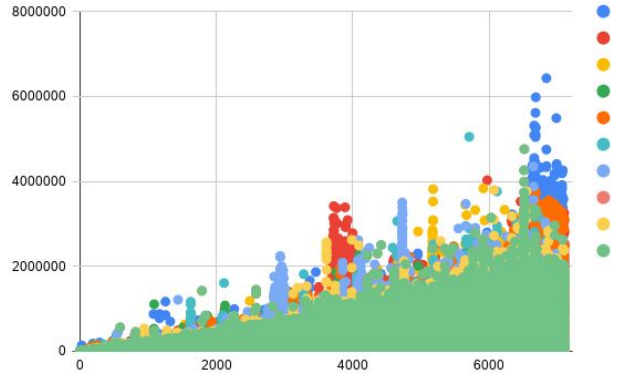
Chris Zhang

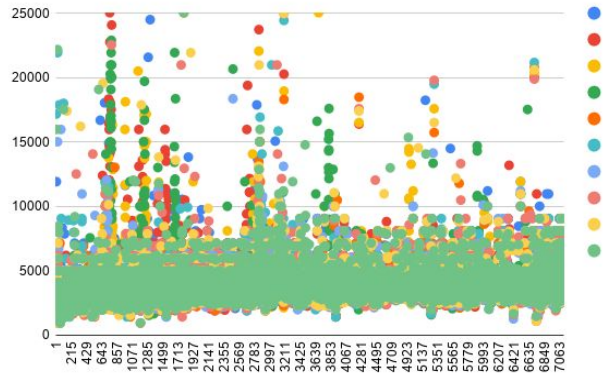
Set Implementations

In Exercise 1, I implemented a set using three different data structures: a linked list, a binary search tree, and a hash table.

The theoretical complexity of adding a word to a linked list is O(n) with respect to the length of the linked list. Since my implementation added words in sorted order, O(n) was an upper bound, but the resulting graph is in fact roughly linear, as it takes more time to traverse the list when the list is longer. The average time to add a word was 602670 ns, and the standard deviation was 461746.

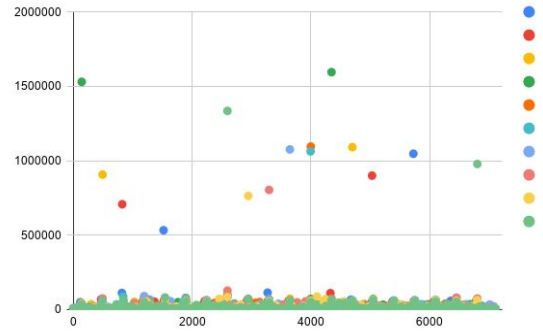


The complexity of adding a word to a binary search tree is O(n) with respect to the number of words added in the worst case (if the tree is completely unbalanced), or about O (log n) on average, assuming a balanced tree. The results of my implementation were also roughly O (log n) on average, with a fair amount of worse-case scenarios. The average time was 3534 ns, and the standard deviation was 10060.



Adding a word to a hash table should be roughly constant, given a hashing function that distributes entries evenly into buckets. My implementation uses a linked list in each bucket, explaining some of the worst-case scenarios, but the results overall remain constant with respect to the number of words added, since the hashing function can immediately approximate

where the new word should go. The average time was 2617 ns, and the standard deviation was 8511.



When searching with a linked list, the best time was 0 ns, the worst time was 4,693,031 ns, and the average was 632,109 ns. When searching a binary tree, the best time was 0 ns, the worst time was 377,178 ns, and the average was 5,673 ns. When searching with a hash table, the best time was 953 ns, the worst time was 87,979 ns, and the average was 3,439. Comparing the three structures, hash tables had the fastest average search time and the lowest worst case time, while linked lists had the slowest average time and worst case time.