Khelsey Gozum A12070231
Eliott Ham A13186685

Final Report

For Hashtable, our empirical results were not consistent with the analytical running time expectation of O(1). In the given graph, our results has shown a graph that's fairly linear, so it's running time is close to that of O(N). This may be due to our implementation of unordered_set. For our BST, our empirical results were consistent with the analytical running time. The graph shows one that is similar to logn. Similarly, with our TST, our empirical results were consistent with the analytical running time of O(K). Our graph is fairly linear.

The hash functions used in benchhash were both taken from stackoverflow. The first hash function, hash_1, works by iterating through the string and taking the integer value of a character plus a prime, 31, times the previous hash value. It then returns the summed hash value modded by the table size. The second hash function, hash_2, uses prime numbers to help with collisions. It takes the xor'd value of the previous hash value multiplied by a prime constant, A, with the product of the integer value of a given character and another prime constant, B. It then returns the final hash value modded by the table size.

To verify the correctness of each of these functions I calculated the expected value by hand and compared them with the functions by running them in benchhash.cpp. I have commented out these tests so that the statistical data of each function prints as required by the assignment handout. For hash_1, I ran the strings "cat", "dogs", and "hash" by hand and got the values 2, 3, and 2 respectively. When comparing them against the actual function output, I found them to be correct. I ran the same strings with hash_2 and received the values 3, 2, and 1 respectively. These also came back consistent with the function output.

When running the functions with 10000 words taken from shuffled_freq_dict.txt, we received the following results:

Results of hash_1 (table size of 20000)

#hits	#slots receiving that # hits
0	12194
1	5965
2	1537
3	260
4	40
5	3
6	1

The average number of steps in a successful search for hash_1 would be 1.2602.
The worst case steps that would be needed to find a word is 6.

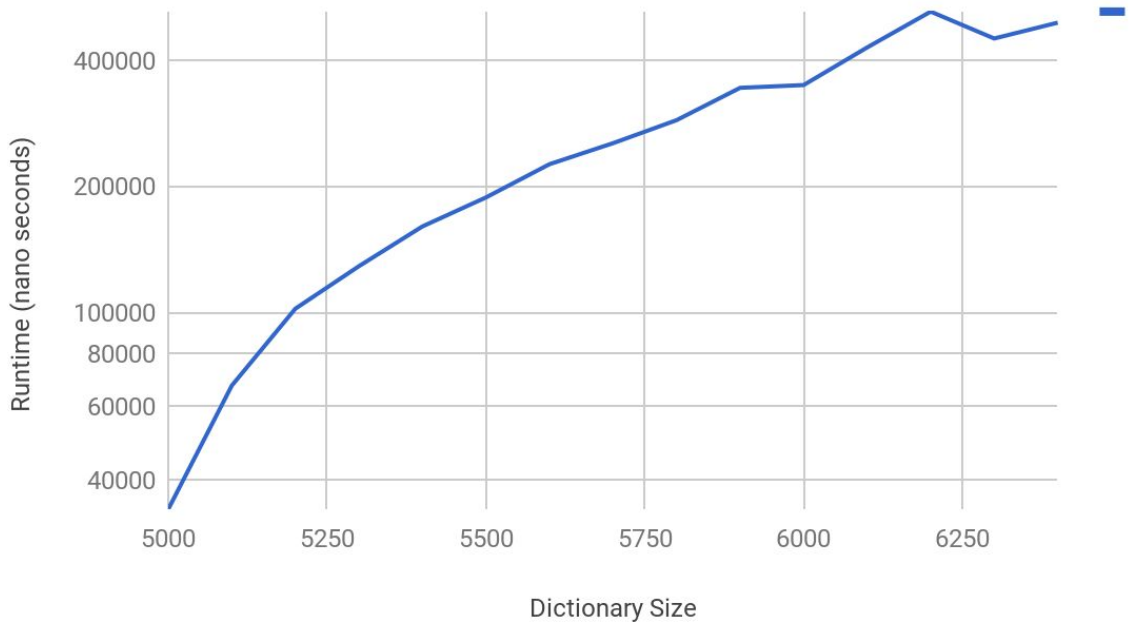
Results of hash_2 (table size of 20000)

#hits	#slots receiving that #hits
0	12140
1	6056
2	1504
3	268
4	28
5	4

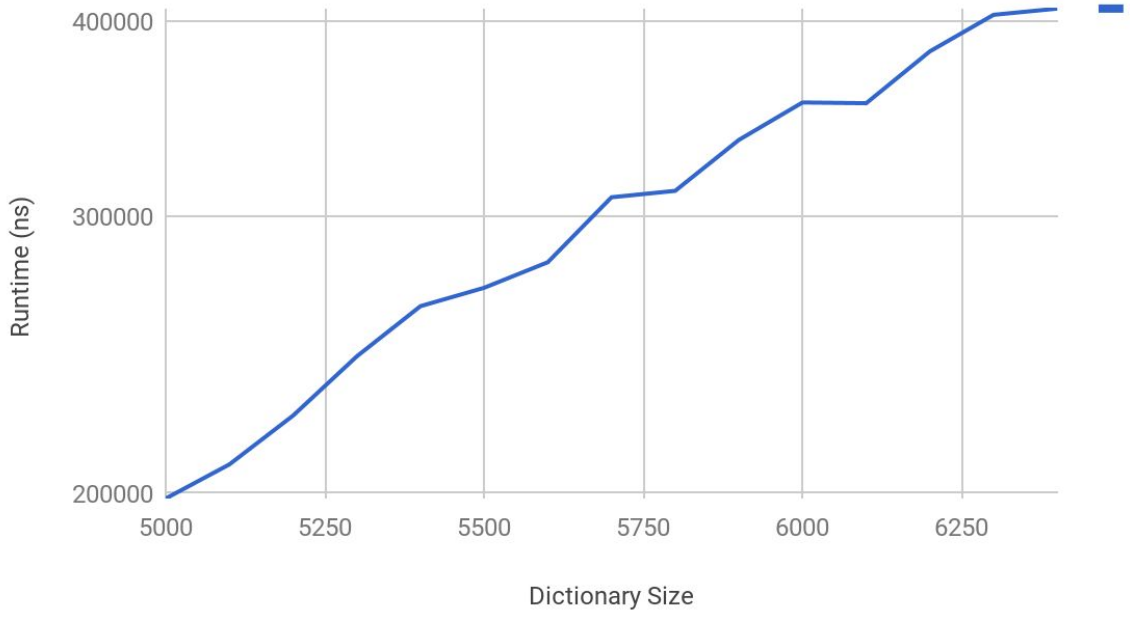
The average number of steps in a successful search for hash_2 would be 1.2516.
The worse case steps that would be needed to find a word is 5.

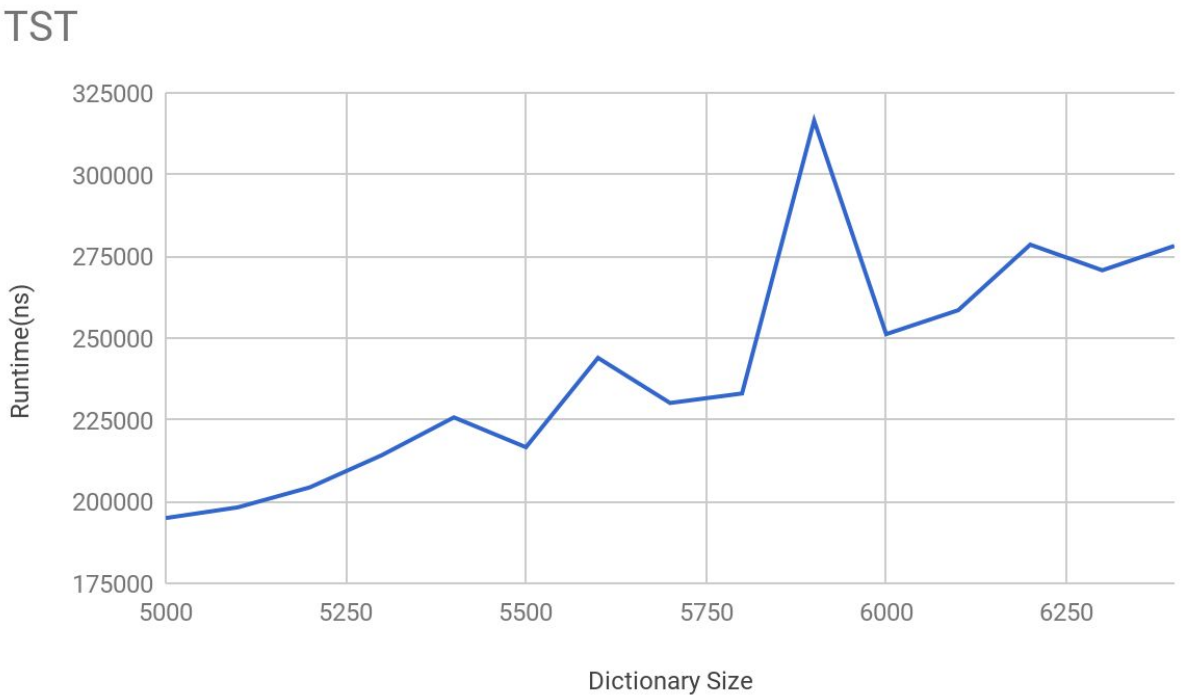
From the data we can see that hash_2 had both a lower average number of steps and lower worst case steps to show that is the more efficient function. This result matched my expectation due to the fact that it is more complex than the first function, however it did not exceed the first function by as much as I had initially thought it would have.

BST



HashTable




TST
325000
300000
275000
250000
225000
200000
175000
Runtime(ns)
5000
5250
5500
5750
6000
6250
Dictionary Size