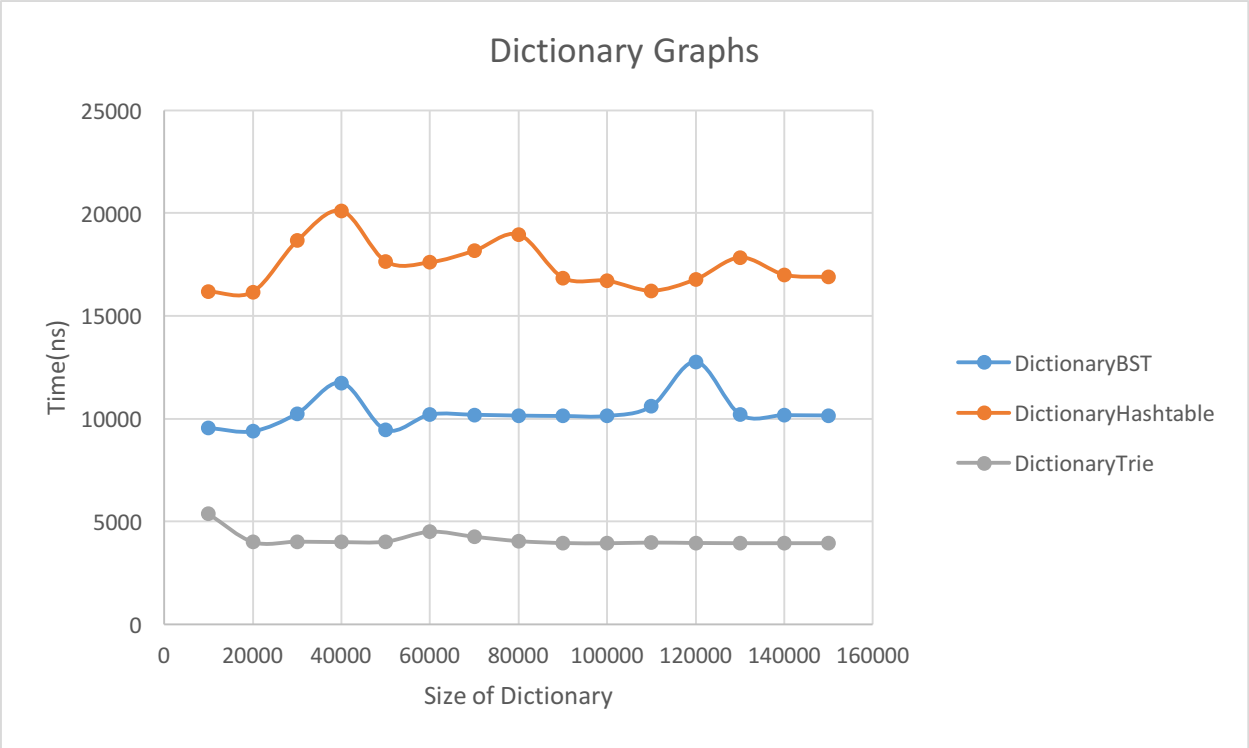
Final Report



Questions

1 As far as consistency with the expected worst-case time to find an element in the dictionary, the Hashtable was the only structure whose result conflicted with its expected run time O(1), while the results from the BST and Trie were close to their respective running times of O(logN) and O(K). Because the longest key is relatively the same size no matter how large the dictionary gets, the running time for the Trie remains roughly within the same range for all dictionary sizes (as seen above by the almost flat plotting of Trie running times). For the BST case, as the size of the dictionary grows, the plotting of run times should be consistent with (logN). In the graph above, we can see the plot slowly increasing with a logN shape. However, we assumed that the Hashtable results are not consistent with O(1) because it's difficult for the Hashtable to handle dictionaries of large sizes; it would have to keep rehashing and adjusting its size according to the amount of keys, which is evident in the unstable plot of the Hashtable results in the graph above).

2 Description of Hashing Functions:

   a. Hash Function 1: Loops through and takes the ASCII value of each individual character of the string and adds all the ASCII values together, resulting in the total ASCII value of the string. Once it has the total ASCII value of the string, mod the value by the table size. (Source: http://research.cs.vt.edu/AVresearch/hashing/strings.php)

b. Hash Function 2: This hash function takes the individual ascii value of each character and adds them together, after shifting the bits of the previous hash value by 6 to the left, and then 16 to the left, and subtracting the old hash value. It does this for every letter in the string, and then adds them together and assigns that as the hash value. (Source: https://www.programmingalgorithms.com/algorithm/sdbm-hash?lang=C%2B%2B)

3 For the test cases, we tested the indexes resulting from the two different hash functions using an all-lowercase string, an all-uppercase string, and one string that included a space, to make sure that each was giving a different index.

a. Hash Function 1: To find the expected hash index using this function, we just calculated the sum of the ASCII values of the characters of some string. We then defined some string and called the function on that string. If the return was the same as the value we calculated, we knew it was correct.
b. Hash Function 2: To find the expected hash index using this function, we wrote out by hand and calculated the bit shifting and addition involved on some string. Once we found the expected index and ran the function calling on the string, we compared the two; if they were the same, we knew the implementation was working as expected.

Hash Function 1- Run 1				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
5	0	10	1	0

Hash Function 1- Run 2				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
10	0	19	1	1
10	1	1	1	1

Hash Function 1- Run 3				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
100	0	182	1	3
100	1	16	1	3
100	2	1	1	3
100	3	1	1	3

Hash Function 1- Run 4				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
1000	0	1786	2	4

1000	1	156	2	4
1000	2	45	2	4
1000	3	11	2	4
1000	4	2	2	4

Hash Function 1- Run 5				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
10000	0	18379	9	19
10000	1	257	9	19
10000	2	215	9	19
10000	3	192	9	19
10000	4	160	9	19
10000	5	159	9	19
10000	6	147	9	19
10000	7	132	9	19
10000	8	96	9	19
10000	9	80	9	19
10000	10	47	9	19
10000	11	43	9	19
10000	12	21	9	19
10000	13	27	9	19
10000	14	14	9	19
10000	15	6	9	19
10000	16	2	9	19
10000	17	2	9	19
10000	18	2	9	19
10000	19	1	9	19

Hash Function 2- Run 1				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
5	0	10	1	0

Hash Function 2- Run 2				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
10	0	19	1	1
10	1	1	1	1

Hash Function 2- Run 3

Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
100	0	184	1	3
100	1	12	1	3
100	2	3	1	3
100	3	1	1	3

Hash Function 2- Run 4				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
1000	0	1805	2	4
1000	1	169	2	4
1000	2	23	2	4
1000	3	2	2	4
1000	4	1	2	4

Hash Function 2- Run 5				
Num_words	#hits	#slots receiving #hits	Average # of Steps	Worst Case Steps
10000	0	18165	2	4
10000	1	1566	2	4
10000	2	235	2	4
10000	3	31	2	4
10000	4	3	2	4

Hash Function 2 works better because there are less significantly less collisions as the Hashtable grows larger, as evident from the data above.