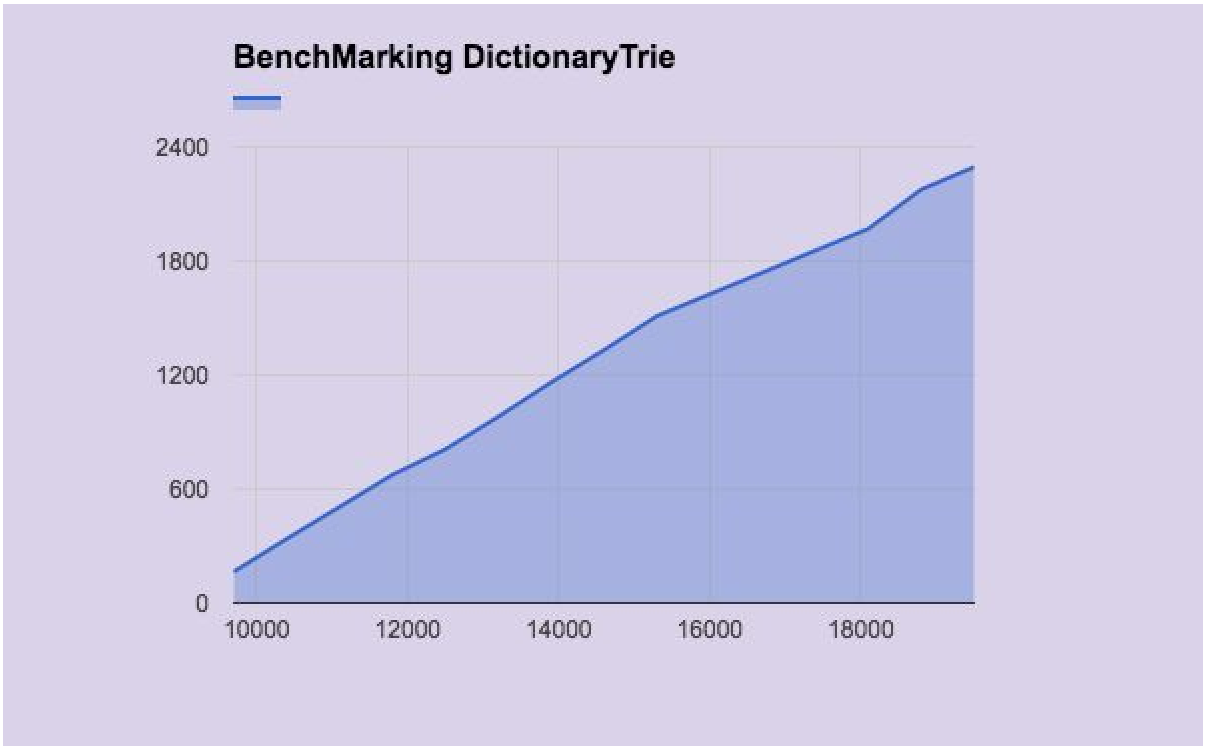
Names: Ramiz Alhaddad, Sanshit Sagar (A12776094, A92073034)
PA2
Benchmarking Dictionaries and Hashtables pdf

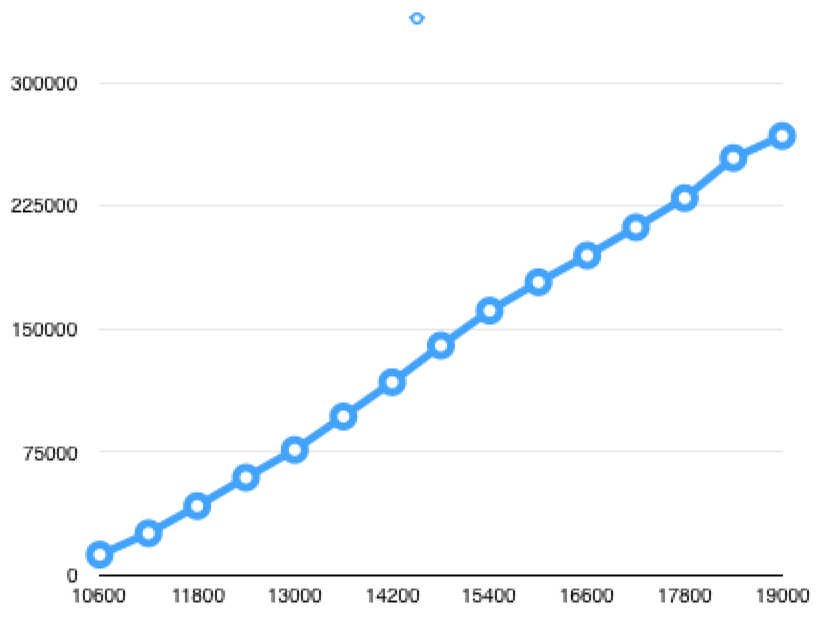
Benchmark Dictionaries

For all the graphs below, the y axis is size of the dictionary, and the x axis is the runtime. Through analysing these graphs, as shown in the write up, we can analyse the runtime efficiency of the implementations.



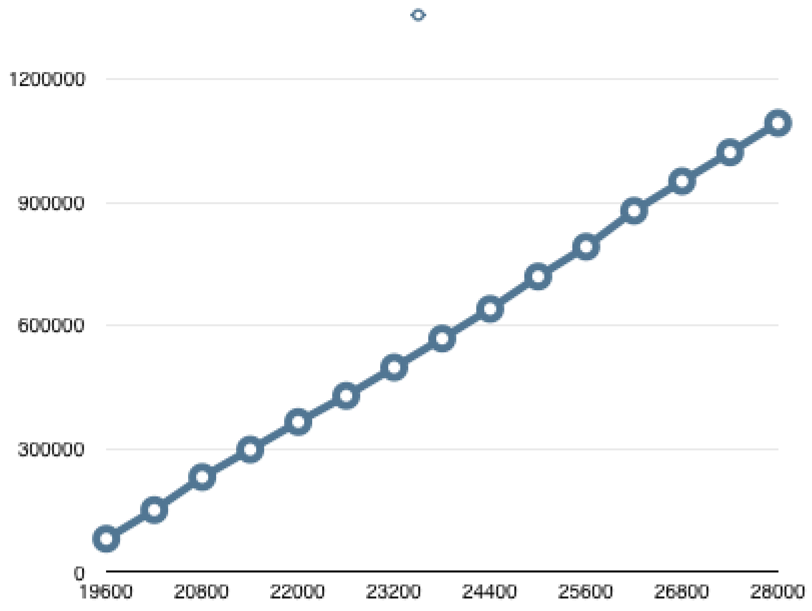
For the dictionaryTrie, the expected time to find an element in the graph would be O(k). The graph is a little more steep than what is expected. There could be a number of reasons for these results such as the way the code is implemented and the compiler and other reasons. However, the scale of the graph can be adjusted to get a sense of a graph that looks closer to O(k). Slope = 1800-1200)/(1700-1150) = roughly 0.21. The graph looks linear, and has a seemingly almost constant slope such that 0<slope <1. It can therefore be said that this graph makes sense.

Names: Ramiz Alhaddad, Sanshit Sagar (A12776094, A92073034)
PA2
Benchmarking Dictionaries and Hashtables pdf



Benchmarking DictionaryHashTable.
As for the results of the Dictionary HashTable graph above, the result should reflect the runtime of O(1). The graph is a bit more steep that what is supposed to be which maybe the case of factors such as the way the instructions layout in the code, as well as the fact that the wrapper that we use to implement the c++ standard library's hash table, that might have been inefficient. Furthermore, given that we didn't change their implementation all that much, it is likely that this inefficiency could have resulted from inefficiency in our benchDict program.

Names: Ramiz Alhaddad, Sanshit Sagar (A12776094, A92073034)
PA2
Benchmarking Dictionaries and Hashtables pdf



BenchMarking DictionaryBST.
We calculated the slope for the graph and it is roughly 28000/110000 = 0.25. Also, the slope looks roughly constant for 19600 < x < 28000. However, a BST should have a finding efficiency of 0(log(n)). Firstly, as in the previous example, the scale of the graph would affect the apparent appearance of the graph a lot. Secondly, in terms of growth rate, logn and n functions are very close to each other relatively speaking, when compared to functions such as 2^n, n! And n^n. So, it is possible that when actually running the implementation in real life, for any finite data set, distinguishing between n and logn growth rates would be hard, and would not tell us much. Lastly, an argument can be made, even if it is a little optimistic, that the graph curves towards the x axis a little in the high x values, and this resembles the logn function’s growth rate.

Names: Ramiz Alhaddad, Sanshit Sagar (A12776094, A92073034)
PA2
Benchmarking Dictionaries and Hashtables pdf

4. *Research and compare the performance of different hash functions for strings*

(a) We chose two hash functions, the first of which is a very simple hash function that we found all over the internet. One specific example of it being implemented can be seen here: http://www.ida.liu.se/opendsa/OpenDSA/Books/TDDI16F16/html/HashFuncExamp.html. The second hash function can be seen implemented similarly here: http://stackoverflow.com/questions/8317508/hash-function-for-a-string .The first function simply adds up the individual value of the individual characters' ASCII value. This is fine, but not ideal since the function outputs the same value for two palindromes. The second function simply multiples the previous hash by a prime number, in this case 131, and then adds the ascii value for the character. The second function can differentiate between palindromes, and since this solves the problem and should therefore, theoretically be better.

(b) We verified the correctness of our hash functions, by simulating the collisions in a vector very similar to what we did when printing the simulated table in bench hash. Only difference when testing the hash functions was, that we gave the functions our own 3 words, and then checked if after inserting all the words, the indices of the vector all had the correct number of collisions. We did this for both the hash functions, to verify that they were correct.

(c) Sample Output:

```
[ralhadda@acs-cseb240-11]:ralhadda:1102$ ./benchhash freq1.txt 10000
Printing the statistics for hashFunction0 with hash table size 20000
#hits   #slots receiving the #hits
0       18261
1       363
2       209
3       153
4       148
5       130
6       108
7       103
8       78
9       88
10      72
11      61
12      41
13      44
14      44
15      31
16      17
17      17
18      6
19      14
20      3
21      4
22      1
23      2
24      1
25      0
```

Names: Ramiz Alhaddad, Sanshit Sagar (A12776094, A92073034)
PA2
Benchmarking Dictionaries and Hashtables pdf

```
The average number of steps that would be needed to find a word is 0.498725
The worst case steps that would be needed to find a word is 25
Printing the statistics for hashFunction1 with hash table size 20000
#hits    #slots receiving the #hits
0      12182
1      5965
2      1561
3      257
4      33
The average number of steps that would be needed to find a word is 0.49955
The worst case steps that would be needed to find a word is 4
```

```
[ralhadda@acs-cseb240-11]:ralhadda:1104$ ./benchhash freq2.txt 1000
Printing the statistics for hashFunction0 with hash table size 2000
#hits    #slots receiving the #hits
0    1402
1    366
2    130
3    54
4    36
5    7
6    3
7    1
The average number of steps that would be needed to find a word is 0.496248
The worst case steps that would be needed to find a word is 7
Printing the statistics for hashFunction1 with hash table size 2000
#hits    #slots receiving the #hits
0    1211
1    607
2    158
3    20
4    3
The average number of steps that would be needed to find a word is 0.497749
The worst case steps that would be needed to find a word is 4
```

```
[ralhadda@acs-cseb240-11]:ralhadda:1105$ ./benchhash freq3.txt 50
Printing the statistics for hashFunction0 with hash table size 100
#hits    #slots receiving the #hits
0    64
1    25
2    9
3    1
The average number of steps that would be needed to find a word is 0.464646
The worst case steps that would be needed to find a word is 3
Printing the statistics for hashFunction1 with hash table size 100
#hits    #slots receiving the #hits
0    57
1    37
```

Names: Ramiz Alhaddad, Sanshit Sagar (A12776094, A92073034)
PA2
Benchmarking Dictionaries and Hashtables pdf

```
2	5
The average number of steps that would be needed to find a word is 0.474747
The worst case steps that would be needed to find a word is 2
```

(d)The second hash function for reasons explained in the second paragraph, should theoretically be better, and as can be seen by the output is about the same for average steps taken, and better for the worst case run time. This makes practical sense, since the function isn't really catered to being more efficient than the simple hash. Instead, it makes sure that the values are usually more spread out than the simple hash. This is because it uses a prime number, and can differentiate between palindromes, as well as two words having the same letters, but in a different order. This would manifest itself, in the worst case run time(the third print statement for bench hash) being lower for the second hash function than for the first - which it is. Therefore, the results for benchmarking these two hash functions, make sense.