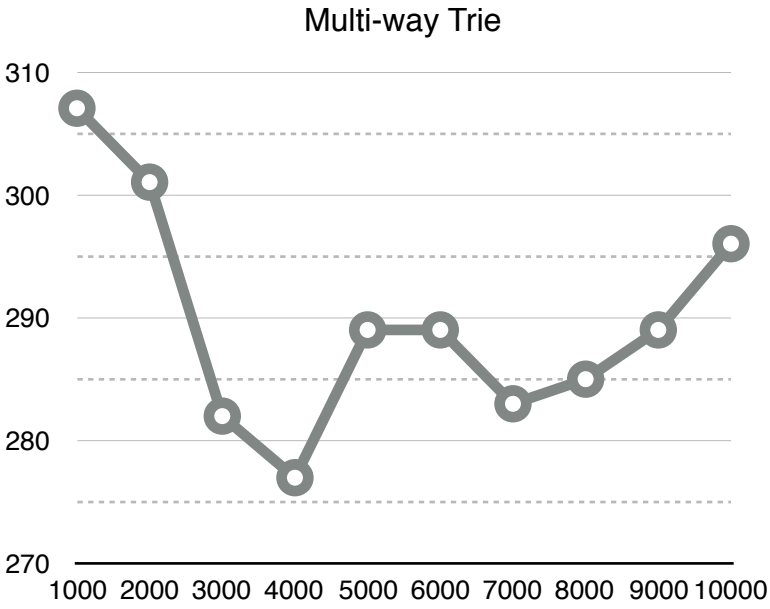
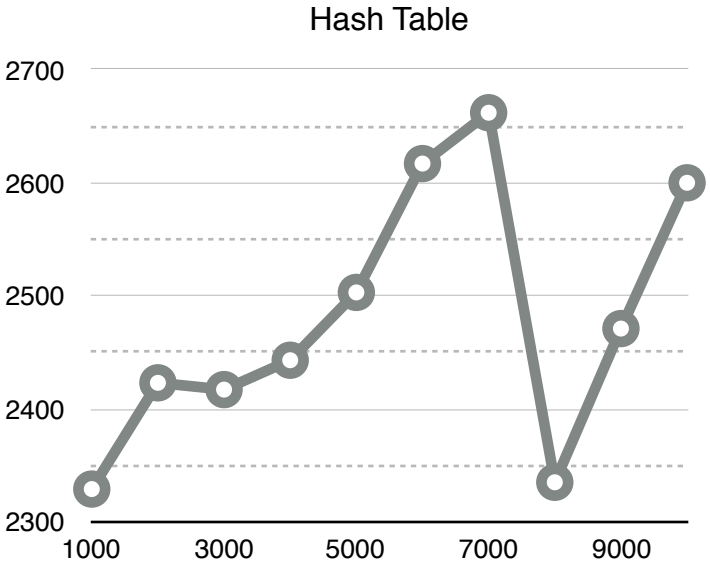
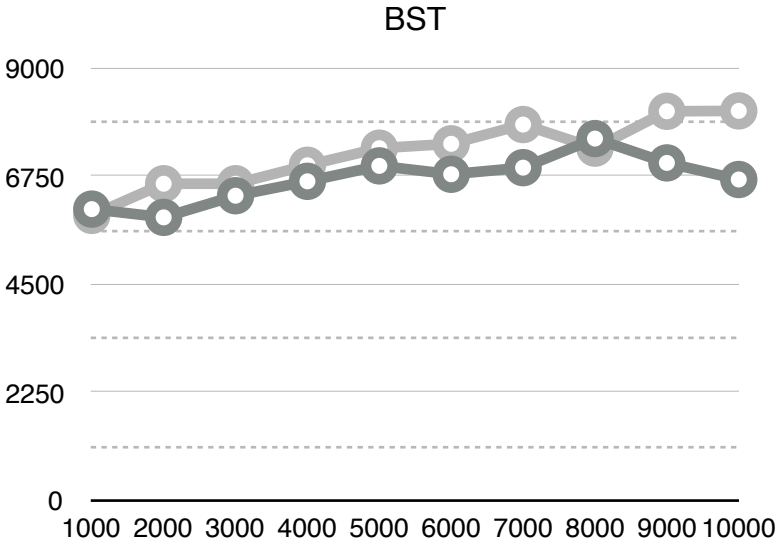
Name: Madeline Hsia
Login: cs100wew







1. What running time function do you expect to see for the find method in each of your three dictionary classes as a function of N (number of words), C (number of unique characters in the dictionary) and/or D (word length), and why?

Expected worst case for the three data structures:
Trie: O(D), D is the length of each keyword.
BST: O(N) N is number of words in the tree
Hash table: O(1) because the C++ hash table is implemented to always give average time, however worst case is still O(N), N is the number of words in the tree

2. Are your results consistent with your expectations?

Trie ran as expected, with the runtime not correlating with the dictionary size, but with the length of each keyword, and also being the fastest of the three data structures, because the word size D is much smaller than dictionary size. In the graph, the first finds seem slower than the others, but in the other data I got I can see that in general there is no correlation between the iterations, and has a pretty constant average time of around 300 nanosecs.

BST, instead of giving log(N), is giving more of a linear result as shown in the graph. But a larger dictionary size is taking longer than smaller dictionary size shown in the positive correlation, which is expected. In the graph, I included two sets of data to show that while it seems linear, the time taken is still kind of random, which might be because of memory usage (tested during peak hours…program already running for a long time, size of tree is large).

Hash table is not giving worst case performance, which was expected. The results of hashtable is more random as shown in the graph, possibly due to collision resolutions, other processes, and memory usages.

3. Explain the algorithm that you used to implement the predictCompletions method in your dictionaryTrie class. What running time function do you expect to see for the predictCompletions as a function of N (number of words), C (number of unique characters in the dictionary) and/or D (word length), and why? This step also requires mathematical analysis. Your function may depend on any combination of N, C, and D, or just one of them.

N: dictionary size
P: prefix size
W: num words with prefix

predictCompletions code:
* initialize objects null check O(1)
* for length of prefix, find prefix by traversing to desired index O(P)
* traverseTrie (recursive function) starting at end of prefix O(W)
* loop numCompletions time and put words on vector O(1)

traverseTrie code:
* initialize objects & null check O(1)
* for each index O(1)
  * get char O(1)
  * push char O(1)
  * traverseTrie O(W)
  * remove char O(1)

predictCompletion's running time will not be correlated to N (number of total words) because dictionary size is not related to my algorithm, since it takes O(P) time to get to the node of prefix, then it searches through the subtree which would require number of words with the prefix. In total, it would take O(P) + O(W), which is O(P+W) times.