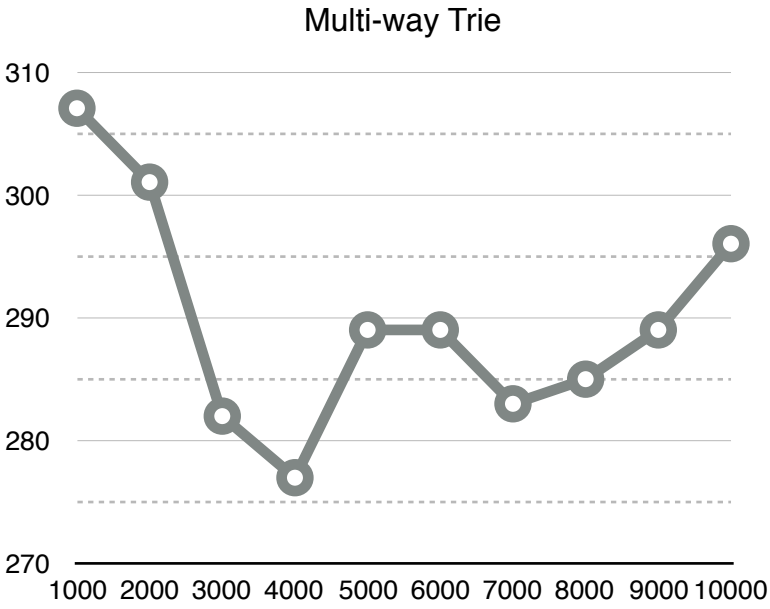
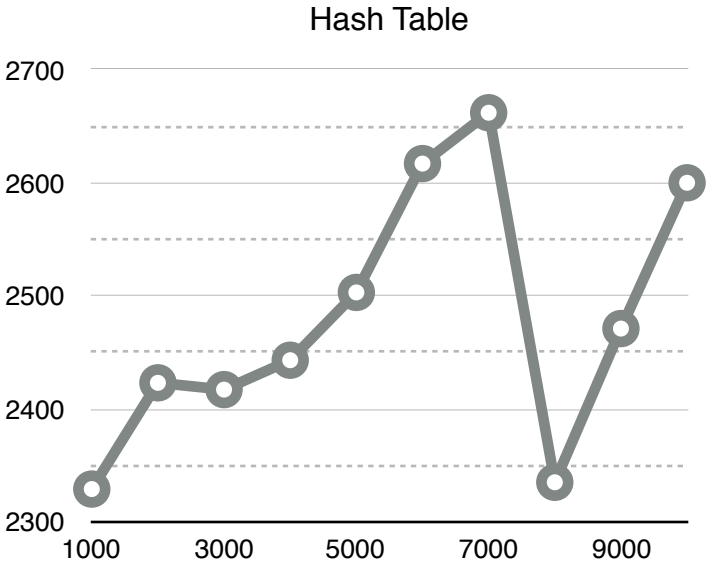
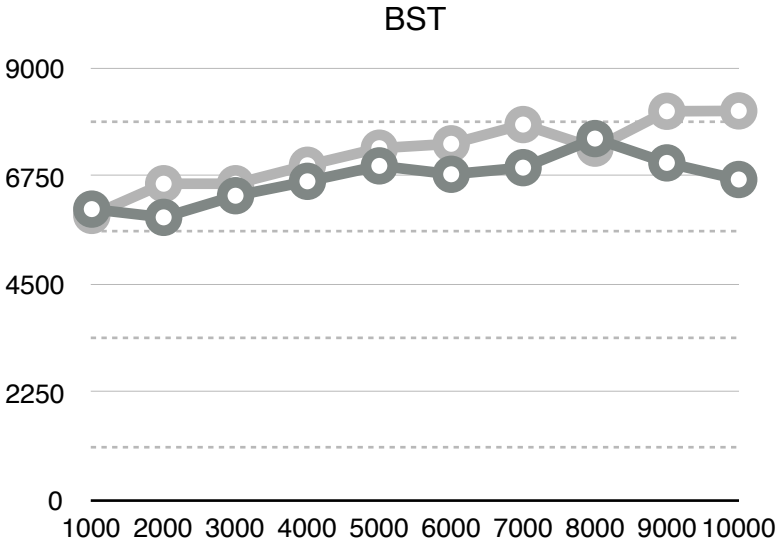
Name: Madeline Hsia
Login: cs100wew







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? This step requires mathematical analysis. Your function may depend on any combination of N, C, and D, or just one of them.

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? If yes, justify how by making reference to your graphs. If not, explain why not and also explain why you think you did not get the results you expected.

Trie ran as expected, with the runtime not correlated with the dictionary size.
BST, instead of giving log(N) every time, is giving more of a linear result, but a larger dictionary size is taking longer than smaller dictionary size, which is expected.
Hash table is not giving worst case performance, which was expected.

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.

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(Length of prefix) time to get to the node of prefix, then it searches through the subtree which would require number of words that's after the prefix.

N: dictionary size
P: prefix size
W: word size

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)
* if isWord make pair and push O(1)
* for each index O(1)
  * get char O(1)
  * push char O(W)
  * traverseTrie O(W)
  * remove char O(1)