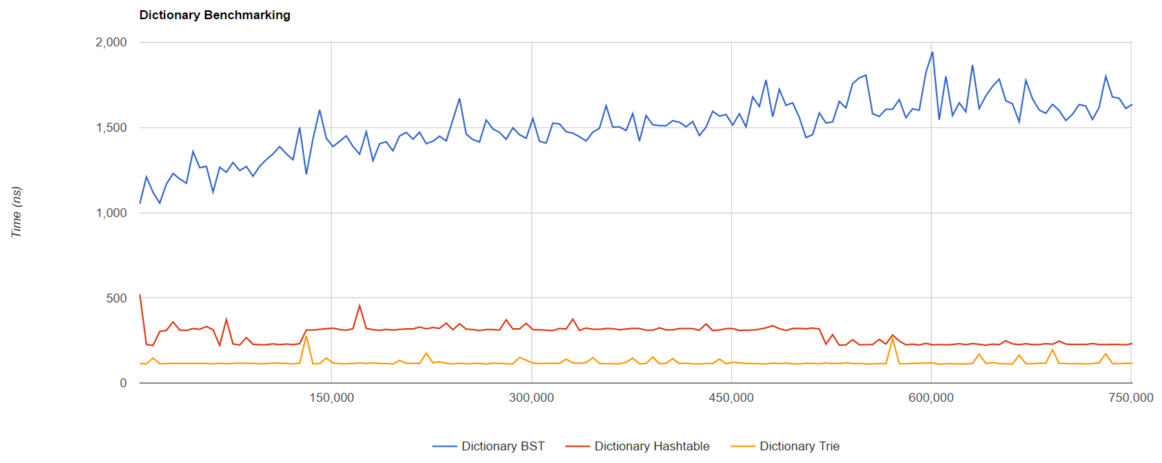

1. BST: O(logn) The tree is built based on the number of nodes which is based on the number of words. A binary tree on average case runs O(logn). The length of the words or unique character does not matter in the construction of the nodes or the tree.
   HashTable: O(1) The hash table search is based on calculating a hash value, gaining O(1) time at the expense of space. The hash function might depend on the length of the word. So the hashing function might be O(D).
   Trie: O(D) The trie I implemented contains an array of pointers in each node from A to Z. The number of times I need to cycle through each letter (traverse down the tree) is based on the length of the word.
2. Yes
   a. The BST has a general log curve which is consistent with the property of BST.
   b. The Hash table obviously has a O(1) runtime. The line is unsurprisingly flat
   c. Trie depends on the length of the word to traverse down the level. Since between each search, the length doesn't change, the time is constant.
3. I optimized Breadth First Search by including the Max Frequency of a given subtree in each node. The Breadth First Search visits the subtree with the highest potential first. The retrieved data is sorted by a max heap based on frequency. These data are popped into the vector until the data is less than the Max of the next subtree. This process then repeats for the next subtree. The autocomplete terminates when the vector has enough strings as user defined or all the possible completions have been exhausted. Breadth First Search is based on the number of nodes needing to be visited. Since the nodes are representations of letters, the process is a linear function of D (word length), O(D).