So: to make an autocomplete function, I need to have the dictionary with keys (basically just copy from hw6?) and then do some checks of those keys for the input...

“Implement the autocomplete solution in Java. The input is a set of words (no need to read from file). Then given a search text (here assuming it's only a single or partial word), the output is a set of words from the input word set that starts with the given search text. You can only use Java natative data types and java.util classes such as map, stack, queue, etc.”

So: the set is an array of Strings. No need for dictionary.

First: turn the set into a Char[][] (O(1) for each word, O(n) for all words, additive)

Second: check the first letter of the input (it will also be a Char[] at this point), set the portions of the Char[][] that do not match to null, continue with second,third, etc. until either input has no more or everything is null. (O(n) for total, decreasing to

O(logn)? because it does not need to check anything that is null for an average case, multiplicative so n^2 based on search for worst case, best case O(n) because everything will be null) {Future me: It’s actually O(nm), because there are 2 bases for O. It’s like O(n^2), but not quite.}

Third: Return the char arrays that survive, and turn it into a String[].

Fourth: Output the String[] as a String.

Wow, that’s it?

It’s gotta be harder than it looks, then.

{Future me: Nope.}

Given a set of words ["hello", "high", "seattle", "seatac", "see", "hollow", "how"]

* when you input "h", it'll produce ["hello", "high", "hollow", "how"]
* when you input "se", it'll produce ["seattle", "see"]
* when you input "sea", it'll produce ["seattle", "seatac"]
* when you input "ho", it'll produce ["how"]
* when you input "xyz", it'll produce []

Input: h

Step 1: [[h,e,l,l,o],[h,i,g,h],[s,e,a,t,t,l,e],[s,e,a,t,a,c],[s,e,e],[h,o,l,l,o,w],[h,o,w]]

Step 2: check each based on input:

[[h,e,l,l,o],[h,i,g,h],[s,e,a,t,t,l,e],[s,e,a,t,a,c],[s,e,e],[h,o,l,l,o,w],[h,o,w]] becomes

[[h,e,l,l,o],[h,i,g,h],null,null,null,[h,o,l,l,o,w],[h,o,w]]

Step 3: Turn it back into a String[] (["hello", "high", "hollow", "how"]) (O(n), additive) I will specifically use String.valueOf(Char[]) because it removes the excess.

{It is impossible for me, at the moment, to do this faster, simply because O(n) is the fastest way I can think of to manipulate an array. When doing it based on m, O(nm) is the fastest possible way. If there is one faster, though, I will gladly implement that instead.}