Autocomplete Design:

My solution for this problem works in the following manner:

We begin with an initialize() function. This function creates a sequence (fixed array) of hash maps, with the ith position in this sequence corresponding to the (i+1)th letter of each word (i.e. at index 0, we contain a hash map whose keys are all the first letters of each word). initialize() maps each letter to all words that contain this letter as their (i+1)th letter (i.e. at index 1 of our hash map array, the key “e” in the hash map at that position is mapped to all words whose second letter is “e”). We store all of these words in a hash set (since order of results doesn’t matter and we want an ADT with a very fast remove() function based on containment). Overall, initialize() is slow. If we have words to process and the length of the longest word we must process is , initialize will take to complete.

After initializing, the search() function proves to be much faster. We start with a hash set that will contain our results and initialize it by using the first letter of our search term as the “key” to retrieve the hash set corresponding with all words starting with that letter (i.e. if my search term is “green”, I take “g”, and use it as a key to retrieve a hash set of all letters starting with g). Then, we iterate over each letter of our search term taking the set intersection (the retainAll() function) of our hash set and the hash set representing all words containing letter of our search term (i.e suppose our search term is “se” and initial hash set is {seattle, seat, singer, song}. In iteration 2, we take “e” and use it as a key to the set of all words with “e” as their second letter: {retrieve, seat, seattle, pen}. We remove() all elements in the first set that aren’t contained in the second set, giving us the hash set {seattle, seat}). Our time complexity for search() is much nicer. Suppose it takes steps to retrieve a hash set and the length of our search term is . If it takes steps to complete a set intersection, total search() time should be , which translates to constant () time.  
I chose to mainly rely on different implementations of the Dictionary ADT because time complexity during search() is the value we most want to minimize, and the hash map implementation of the Dictionary ADT allows for retrieval of a set of matching terms for each letter of our search term.   
  
Results of searches:  
“h” -> {“how”, “high”, “hollow”, “hello”}  
“se” -> {“see”, “seattle”, “seatac”}  
“sea” -> {“seattle”, “seatac”}  
“ho” -> {“how”, “hollow”}  
“xyz” -> {}