My solution used the Trie tree you mentioned in class as it was the cleanest solution, for example when you enter in s it traverses to s from the tree (going from the beginning empty node to the first s as all you entered was s) and then recursively saves all the branches off of that to a linked list of strings to then be given back for use. The way it generates the linked list is to go down one branch until it hits a node whose boolean value tells it that it is the end of a word, during the passthrough it saves the values to a linked list resulting in the completed words. Given that it only passes through the trie from the end letter of the entry down that branch and just adds the values to a linked list, I would imagine that its performance complexity would be N, I don't think it would be n when adding to the Trie but when pulling out i do think it would be N. for example the output of the test cases included with my code give this :

Hello,hero,high,hollow,how,hat ( for the input of h )

-----

Seattle,seatac,see ( for the input of se )

-----

Seattle,seatac ( for the input of sea )

-----

Hollow,how ( for the input of ho )

-----

( for the input of xyz )

-----

Cow,chain,cheese,cheif,castle ( for the input of c )

-----

Chain,cheese,cheif ( for the input of ch )

-----

Book ( for the input of bo )

-----

( for the input of robotic )

-----

Seattle,seatac,see,spider ( for the input of s )

-----

As far as i know it is the fastest way i can come up with, there may be a better way to input and remove but N removal complexity is pretty good. The people i have spoken with have gotten similar results and the one guy who got better refuses to share his secrets with me until after the assignment. So in the future I should be, if he has calculated it correctly, be able to do it better.