CS 224 HW 3 - take two weeks. Some of the problems are somewhat vague and open ended. Do what you can.

1. In class we saw that BWT[i] = T[SA[i] - 1], where T is the input string, BWT[i] is the character at position i in the BWT string for T, and SA[i] is the i'th entry in the suffix array for T.

This shows how to compute the BWT from the SA. What about the other direction? If we have the BWT, what else would we need to compute the SA, and in linear time? I don't know the answer.

Answer: Assume that the original string S has \$ at the end, and that the total length of S, including the \$ is n. Recall the list M of the n rotated strings, and the list of pointers LF that were used in the inversion method to recover S from the BWT string. $S(n)$ is \$ and the first string in M starts with \$. So, $SA(1) = n$. Now $LF(1)$ is used to indentify character $n-1$ in S, i.e., $LF(1)$ points to the position in M where the string in M ends with character $S(n-1)$. But the strings in M are sorted lexicographically, which is also the sorted order of the suffixes of S. So the suffix of S that starts in position $n-1$ must be the $LF(1)$'st lexicographically smallest suffix. So, $SA(LF(1)) = n-1$.

In general, $LF(i)$ points to the position in M where the string in M ends with character $S(n-i)$, and $SA(LF(i)) = n-i$. So to fill in SA we just chase the LF pointers as if we were reconstructing S, and use the rule that $SA(LF(i)) = n-i$. Hence this takes linear time.

2. Read the BWT inversion method that I posted on the class website. That method derives the original string from the left to the right, the opposite of what is done in the FM approach. As was done in the FM method, it should be possible to define pointers that can be followed to spell out the original string in forward order. Analogous to what was done in the FM paper, define those pointers and write up a formula for them. Show how those pointers relate to the FM pointers.

Answer: Define F as the first column of M. As before, we can construct F by sorting the characters in the BWT string L. Define l_k is the number of occurances of character $F(k)$ in the first k positions of F. Of course, because of the sorted property of F, all occurances of character $F(k)$ are consecutive in F, so the l_k values are particularly easy to accumulate. Define $l(c,i)$ to be the position of the i'th occurance of character c in L. These values can be accumulated and put into a two-dimensional array in $O(n)$ total time. Finally, define $FL(k) = l(F(k), l_k)$ as a pointer from entry k in F to another entry in F. $F(FL(k))$ is the character the comes after $F(k)$ in the original string S. Suppose we have the pointer I that points to the position in L corresponding to the string S. Then we start with $k = I$ and $i = 1$, and $S(i) = F(k)$. Then increment i, set $k = FL(k)$, and $S(i) = F(k)$. Repeat until $i = n$.

If we don't initially have the pointer I, but instead we have added \$ to S before the BWT was created, then find I by setting it to the position of \$ in L. Then proceed as above, where it was assumed that we know I.

The relationship of FL to LF: Essentially, the FL are just the reverse of the LF pointers. That is, if an LF pointer points from position i to position j, then an FL pointer points from j to i. So, $FL(LF(k)) = k = LF(FL(k))$.