1. Given a set $\mathcal{S}$ of k strings with total length n, devise an $O(n)$-time algorithm to find every string in $\mathcal{S}$ that is a substring of another string in $\mathcal{S}$.

   Solution: (sketch) Let $\mathcal{S} = \{S_1, S_2, \cdots, S_k\}$. The first step is to find the Burrows-Wheeler Transform (BWT) of

   $$T = S_1\$_1 S_2\$_2 \cdots S_k\$_k$$

   where T is the concatenation of all the strings in $\mathcal{S}$, separated by k special dollar sign symbols. We also compute the FM-index information. This takes $O(n)$ time. Then for each S_i, beginning at the end of the string as described in lecture, we pattern match S_i using the BWT. This takes $O(|S_i|)$ time.

   This process will give us the indices in T where S_i occurs. To translate these back into which string(s) S_i actually occurs in, we can at the start keep track of the indices of all the \$ symbols. After we've repeated this for each string in $\mathcal{S}$, we have a runtime $O(n)$ since that is the total length of all the strings.

2. Consider two strings A and B of lengths n and m, respectively. C is an *interleaving* of A and B if a subsequence of C forms A and the remaining characters not in the subsequence form B. For example, if $A = abac$ and $B = bbc$, then a possible interleaving is $abbabcc$. Devise an efficient algorithm to determine whether a string C is an interleaving of A and B.

   Solution: (sketch)

   Let all string indices start from 1. The solution can be implemented using a dynamic programming table DP where $DP[i][j]$ is 1/TRUE if $C[i+j]$ is a valid interleaving of $A[1\cdots i]$ and $B[1\cdots j]$. For all indices $i = 0, 1, \cdots, |A|$ and $j = 0, 1, \cdots, |B|$, the cases are

   - if $i = j = 0$, $DP[i][j] = 1$ (empty string case)
   - if $i = 0$ and $B[j] = C[j]$, $DP[i][j] = DP[i][j-1]$ (A empty case)
   - if $j = 0$ and $A[i] = C[i]$, $DP[i][j] = DP[i-1][j]$ (B empty case)
   - if $A[i] = C[i+j]$ and $B[j] \, ! = C[i+j]$, $DP[i][j] = DP[i-1][j]$ (A matches, B doesn't)
   - if $B[j] = C[i+j]$ and $A[i] \, ! = C[i+j]$, $DP[i][j] = DP[i][j-1]$ (B matches, A doesn't)
   - if $A[i] = B[j] = C[i+j]$, $DP[i][j] = DP[i-1][j]$ or $DP[i][j-1]$ (both match)

For the given example, the table would be:

	“”		b		b		c
“”	1		0		0		0
	↓						
a	1	→	1	→	1		0
	↓		↓				
b	1	→	1		0		0
	↓		↓				
a	0		1	→	1	→	1
					↓		↓
c	0		0		1	→	1

This shows that $DP[m][n] = 1$, so we do in fact have an interleaving. The runtime is $O(mn)$.

3. Let T be a string of length m and let $\mathcal{S}$ be a multiset of n characters. Devise an $O(m)$ time algorithm to find all substrings of T composed of all the characters from $\mathcal{S}$. For example, if $\mathcal{S} = \{a, b, c, c\}$ and $T = acbcagaba$, then *cbca* is a substring of T formed from the characters of $\mathcal{S}$.

   Solution: (sketch) The idea is to create a sliding window of length $|\mathcal{S}|$ and run it across T and see which windows use all the letters from $\mathcal{S}$. You maintain an array of character counts at the end of each window, and update this array incrementally as you slide the string so that you can update the counts in $O(1)$ time for each position in T (the initialization takes $O(|\mathcal{S}|)$ time). For the example above, our sliding window is of length 4. This table shows the character counts for each position in T, beginning after initialization.

T	a	c	b	c	a	g	a	b	a
a				1	1	1	2	2	2
b				1	1	1	0	1	1
c				2	2	1	1	0	0

   The correct character count is $a:1$, $b:1$, $c:2$. This is matched at positions 4 and 5, so *acbc* and *cbca* are the substrings in T composed of all the characters in $\mathcal{S}$.