The following solutions for Homework 2 are sketches, meant to convey the ideas behind the solutions to the problems, but may not exhaustively cover all details of a complete solution.

(1) **(Longest suffix-prefix overlaps)** Suppose we have a collection of \( k \) strings \( S = \{S^{(1)}, S^{(2)}, \ldots, S^{(k)}\} \). For an ordered pair of strings \((A, B)\), their **suffix-prefix overlap** is the longest exact match between a suffix of \( A \) and a prefix of \( B \). For each string \( S^{(i)} \) in \( S \), we would like to know the longest suffix-prefix overlap \((S^{(i)}, S^{(j)})\) that \( S^{(i)} \) has over all other strings \( S^{(j)} \), and similarly the longest overlap \((S^{(j)}, S^{(i)})\) that \( S^{(i)} \) has over all other strings \( S^{(j)} \). Note that for \( k \) strings, this is \( 2k \) pieces of information: two longest overlaps for each string \( S^{(i)} \).

Given a collection \( S \) of \( k \) strings of total length \( n \), design an algorithm that determines these longest suffix-prefix overlaps for all \( k \) strings in \( O(n \log n) \) time. Argue why your algorithm is correct.

**Solution sketch** Form the concatenated string

\[
S^* := S^{(1)} ∈ S^{(2)} ∈ \cdots ∈ S^{(k)},
\]

where \( ∈ \) is a symbol not in the alphabet of the input strings. Construct the suffix array \( A \) for \( S^* \), the inverse \( B \) of the suffix array, and the height array \( H \).

Consider a given string \( S^{(i)} \), for which we wish to compute its longest suffix that overlaps with another string \( S^{(j)} \). For each position \( p \) within \( S^{(i)} \) in \( S^* \), find the location of position \( p \) in \( A \) (using the inverse array \( B \)). Suppose \( p \) occurs at index \( i \) in \( A \). Let arrays \( L[1 : n] \) and \( R[1 : n] \) be defined as follows:

- \( L[i] \) is the index in \( A \) of the closest start position of some \( S^{(j)} \) that occurs to the left of index \( i \), and similarly
- \( R[i] \) is the index of the closest start position of some \( S^{(j)} \) that occurs to the right of \( i \).

Take the minimum of the height array \( H \) over the two intervals \([L[i], i) \) and \([i, R[i))\), which gives the longest common prefix length between this suffix of \( S^{(i)} \) and a prefix of any string \( S^{(j)} \) falling to the left or right in \( A \), and hold onto the larger of these two minima. If the larger minimum has length at least \( |S^{(i)}| - p + 1 \), then this longest common prefix match extends all the way to the end of string \( S^{(i)} \), and hence is a match with a suffix of \( S^{(i)} \). Record the position \( p \) in \( S^{(i)} \) which gave the longest such length.

In the above process, if index \( L[i] \) happens to hold the start position of string \( S^{(i)} \), then instead use index \( L[L[i]] \). Similarly if \( R[i] \) corresponds to \( S^{(i)} \), then use \( R[R[i]] \). This is to guarantee that we are finding the longest overlap with a string \( S^{(j)} \) that is not \( S^{(i)} \).

This procedure finds the longest suffix-prefix overlap for every string \( S^{(i)} \). To find the longest prefix-suffix overlap for every \( S^{(i)} \), simply reverse all the input strings and repeat the above procedure.

In the above, arrays \( L \) and \( R \) can be precomputed in a preprocessing step by respectively doing a right-to-left and left-to-right scan of the suffix array \( A \).

To analyze the running time, precomputing arrays \( A, B, H, L, \) and \( R \) takes total time \( O(n) \), where \( n \) is the length of \( S^* \). Computing the two minima of \( H \) for every
position $p$ takes total time $O(n \log n)$ using an efficient data structure for interval-minimum queries. So the entire algorithm takes $O(n \log n)$ time.

(2) (Minimum cover) Given strings $A$ and $B$, a minimum cover of $A$ by $B$ is a decomposition $A = w_1w_2 \cdots w_k$ where each $w_i$ is a substring of $B$ and $k$ is minimum.

Design an algorithm that computes a minimum cover (if one exists) of strings $A$ and $B$ of lengths $m$ and $n$ in $O((m + n) \log(m + n))$ time. Argue why your algorithm is correct.

**Algorithm** Given input strings $X[1 : m]$ and $Y[1 : n]$, start with $j := 1$, $i := 0$, and while $j \leq m$, do the following:

(a) Find the longest prefix $x$ of $X[j : m]$ that is a substring of $Y$.
(b) Set $i := 1$ and $w_i := x$.
(c) Advance $j := |w_i|$.

Set $k := i$, and output $w_1, w_2, \ldots, w_k$.

**Finding longest prefixes** We answer the longest prefix queries in Step (a) as follows.

Form string $S := X♯Y$, where ♯ is a character not in the alphabet of $X$ and $Y$, and construct the suffix array $A$ and height array $H$ for $S$.

Suppose suffix $X_j$ occurs at position $\tilde{k}$ in $S$ and index $k$ in $A$. Identify the closest indices $\ell < k$ to the left and $r > k$ to the right that correspond to positions in $Y$. By the same reasoning as in the Longest Common Substring Problem,

$$h := \max \{ \text{lcp}(S_A[\ell], S_A[k]), \text{lcp}(S_A[r], S_A[k]) \},$$

is the length of the longest substring that is a prefix of $X_j$ and is common to $Y$. We can obtain these two lcp values from

$$\min_{\ell \leq i < k} \{ H[i] \}, \min_{k < i \leq r} \{ H[i] \}.$$

For the preprocessing, we compute arrays $A$ and $H$, the inverse suffix array $B$ for mapping $\tilde{k}$ to $k$, and two arrays $L$ and $R$ such that $L[k] = \ell$ and $R[k] = r$. Arrays $L$ and $R$ can each be filled in by a linear scan of $A$ in opposite directions. This preprocessing takes $\Theta(m + n)$ time.

To answer the query for $X_j$, we obtain $k, \ell, r$ using $B, L, R$, and then find $h$ by computing two interval minima of $H$. Since the intervals $[\ell, k]$ and $[k, r]$ are effectively fixed for all $k$, we can precompute and tabulate these minima for all queries when filling in $L$ and $R$ without increasing the preprocessing time, after which we can lookup the value of $h$ from an additional array in $O(1)$ time for any $X_j$. This answers the longest prefix query for $X_j$ in $O(1)$ total time.

**Analysis of cover algorithm** Using this solution for finding the longest prefix of $X_j$ in Step (a) of the prior procedure for Minimum Cover, it finds the decomposition $w_1w_2 \cdots w_k$ in $O(m + n)$ preprocessing time and $\Theta(k)$ time for the $k$ iterations of the while loop, for a total of $\Theta(m + n)$ time.

**Correctness of cover algorithm** We prove correctness of the prior greedy procedure for Minimum Cover using the following lemma.
Lemma 1  Suppose $w_1 w_2 \cdots w_i$ is a prefix of an optimal cover for strings $X, Y$. Let $\bar{w}_{i+1}$ be the next substring chosen by the greedy procedure. Then $w_1 w_2 \cdots w_i \bar{w}_{i+1}$ is also a prefix of an optimal cover for $X, Y$.

Proof  Let $w_1 \cdots w_i w_{i+1}^* \cdots w_k^*$ be an optimal cover for $X, Y$. Suppose $w_{i+1}^* \neq \bar{w}_{i+1}$. Since the greedy procedure always chooses a longest prefix, $|w_{i+1}^*| < |\bar{w}_{i+1}|$.

Let $w_j^*$ be the substring where $\bar{w}_{i+1}$ ends in the optimal cover, and $\bar{w}_j$ be the remaining suffix of $w_j^*$ following $\bar{w}_{i+1}$. Then

$$w_1 \cdots w_i \bar{w}_{i+1} \bar{w}_j w_{j+1}^* \cdots w_k^*$$

is a cover of $X$ by $Y$ that uses no more substrings, so it is an optimal cover as well. Moreover, it has prefix $w_1 \cdots w_i \bar{w}_{i+1}$. \qed

Theorem 1  The greedy procedure for Minimum Cover is correct.

Proof  Immediate from the lemma using induction on $k$. \qed