Instructions  This exam consists of two problems worth a total of 100 points. Do short-answer Problem (1). From long-answer Problems (2) and (3), choose one. In other words, do a total of two problems. If you do more than two, only two will be graded.

When asked to design an algorithm, be sure to:

(i) describe your algorithm using prose and pictures,
(ii) argue that your algorithm is correct, and
(iii) analyze its running time.

Pseudocode is not required, but may be useful for analyzing the running time.

You have 75 minutes. You are allowed one sheet of notes (two sides of one page) that must be turned in with the exam. Other than this sheet, the exam is closed notes, closed book, and closed computer. Good luck!
Two different data structures—suffix arrays and the Ferragina-Manzini index—are both useful for computations that involve exact string matching, and each has their advantages and disadvantages.

Give short answers to the following questions. Be brief and precise. (Hint: Consider factors such as space, time, and ease of implementation.)

(a) (10 points) List two advantages of the Ferragina-Manzini index (FMI) over suffix arrays (SA). Be as precise as possible.

(b) (10 points) List two advantages of full suffix arrays (SA) over the Ferragina-Manzini index with a compressed suffix array (FMI). Be as precise as possible.
(2) **Position identifiers** (80 points) Suppose we have a string $X$ of length $m$, and a set of strings $\mathcal{Y} = \{Y^{(1)}, Y^{(2)}, \ldots, Y^{(k)}\}$ of total length $n$. A position identifier for position $i$ in $X$ is the shortest substring of $X$ starting at position $i$ that does not occur as a substring of any string in $\mathcal{Y}$.

Design an algorithm that, given string $X$ and set $\mathcal{Y}$, finds the position identifier for every position in $X$ (at which such an identifier exists) in $O(n + m \log(m + n))$ total time, using a suffix array. Be sure to argue your algorithm is correct.

(Hint: Make use of finding longest common prefixes of suffixes.)

(Note: It is actually possible to achieve $O(n + m)$ total time, which is optimal, using a suffix array, without using $O(1)$ time interval-minimum queries.)

(3) **String compression** (80 points) At a high level, standard string compression algorithms, such as the so-called Lempel-Ziv encoding, do the following. Given a string $S[1:n]$ to compress, they scan $S$ from left to right. When they are processing position $i$ in $S$ during the scan, they find the longest prefix of $S[i:n]$ that occurs in the already-processed portion of the string $S[1:i-1]$. Suppose this longest prefix has length $\ell > 0$. They then skip past this matched prefix, and repeat, now processing position $i+\ell$. (If $\ell = 0$, they advance to position $i+1$.) The procedure terminates when the scan has processed all of $S$.

Design an algorithm that implements the above processing for a string $S$ of length $n$ so it runs in $O(n \log n)$ time, using the Ferragina-Manzini index with a full suffix array. Be sure to argue your algorithm is correct.

(Hint: Try constructing the Ferragina-Manzini index and the suffix array for the reverse of string $S$. On the reverse of $S$, the above processing proceeds from right to left.)

(Note: It is possible to achieve $O(n)$ time, which is optimal, using $O(1)$ time interval-minimum queries.)
(This page is blank.)
(This page is blank.)
(This page is blank.)
(This page is blank.)