Many of these problems are *original* or taken from other sources. It is quite possible that I have made some problems which are misworded or otherwise impossible, though I hope not.

1. (*) (4.13 revisited:) Given two binary circular strings \( X = x_0x_1 \ldots x_{n-1} \) and \( Y = y_0y_1 \ldots y_{n-1} \), let \( X \star Y \) be the sequence of integers such that \( X \star Y(i) = \sum_{j=0}^{n-1} x_jy_{i+j} \). Show that there exists a \( O(n \log(n)) \) method to construct the sequence \( X \star Y \). (Hint: you may need a well-known algorithm not mentioned in class.)

Give an algorithm for 4.13 of JP (finding a pattern \( s \) in a text \( T \) with \( \leq k \) mismatches) which runs in \( O(|\Sigma||T| \log |T|) \) time for strings over an alphabet \( \Sigma \).

2. Sketch the Aho-Corasick finite state machine which recognizes the words \( aabb, abba, \) and \( bbab \). Sketch the suffix tree of ‘abracadabra’ including suffix links.

3. Compute the Burrows Wheeler transform of ‘mississippi’

4. Describe an implementation of a suffix tree which requires no more than 20 bytes of space per character and is capable of indexing strings of lengths up to \( 2^{30} - 1 \). If you can’t, just get as close as you can. Describe the space required in terms of valid C-structs or C++-classes\(^1\) for the internal nodes and leaves. If you need to assume that a pointer is 32 bits (I don’t think you do), you may. Feel free to look at publicly available suffix tree codes for inspiration or answers.\(^2\)

5. Give a fast algorithm which will reconstruct a string \( S \in \Sigma^n \) from its Burrows-Wheeler transform \( B \in (\Sigma \cup \{\$\})^{n+1} \). Recall that you can

---

\(^1\) or some other popular language or very precise English

\(^2\) I haven’t seen a 20 byte implementation in the public.
evaluate \( occ(B, i, c) \), the number of occurrences of character \( c \) at all positions less than \( i \), in \( O(\log(n)) \) time.

Let \( \phi \) be the permutation which sorts the suffixes of \( S \), i.e. \( S[\phi(i) \ldots n] < S[\phi(i+1) \ldots n] \). Give a variation on the above algorithm which constructs \( \phi \) from the BWT of \( S \).

(Extra Credit:) Let \( \lambda(i) \) be the length of the longest common prefix of \( S[\phi(i) \ldots n] \) and \( S[\phi(i+1) \ldots n] \). Give a fast algorithm which reconstructs \( \lambda \) from the BWT of \( S \).

6. (*) 9.9 of JP. Recall that a tandem repeat of \( S \) is an occurrence of a substring of \( S \) of the form \( BB \) for some string \( B \). (Hint: there is a divide and conquer approach to this which is \( O(n \log(n)) \) in time.)

7. 9.11 and 9.13 (they are closely related) of JP

8. 9.2 of JP. Random text refers to independent and identically distributed latters with probability \( p_i \) for letter \( i \).

9. \(^3\) Modern filtration methods do not use matching \( k \)-substrings as a basis for filtration but use general \( k \)-subsequences. A \( k \)-subsequence, sometimes called a gapped \( k \)-gram, is a concatenation of letters taken from a set of \( k \) relative positions in \( S \). The relative positions are often specified by a bit-string (called a mask) with exactly \( k \) 1s. For example, for the string \( atttgctcg \), the 4-grams with mask 110101, are \( attc, ttgt, ttcc, tgtg, gcc \).

Given a binary string, \( R \), we say that the mask \( M \) covers \( R \) when \( R \) contains a substring which is 1 wherever the corresponding character in \( M \) is 1.

(a) Let \( R \) be a binary string of length \( m \) containing \((\frac{5}{4} + \epsilon)m \) 1s (where \( \epsilon \) is a positive constant). Prove that for sufficiently large \( m \), all such \( R \) are covered by the mask 11011 (i.e. there must be a substring of length 5, which looks like 11g11, where \( g \) is either 0 or 1). Now show that for any \( m \), there exists a binary string containing at least \( \frac{3}{4}m \) 1s that is not covered by 1111.

\(^3\) My thanks to Bin Ma of U. Western Ontaorio for these problems.
(b) Let $R$ be a randomly generated binary string of length 64, where each position is 1 with probability 0.7. Write a program to generate one million such strings, and check how many of them are covered by the masks 111010010100110111 and 11111111111 respectively.