Problem Set #8

You should try to solve these problems by yourself. I recommend that you start early and get help in office hours if needed. If you find it helpful to discuss problems with other students, go for it. **You do not need to turn these problems in. The goal is to be ready for the in class quiz that will cover the same or similar problems.**

Problem 1: Divide and Conquer

Suppose you are given a sorted sequence of *distinct* integers $\{a_1, a_2, \ldots a_n\}$. Give an $O(\log n)$ algorithm to determine whether there exists an index i such that $a_i = i$. For example, in $\{-10, -3, 3, 5, 7\}$, $a_3 = 3$; there is no such i in $\{2, 3, 4, 5, 6, 7\}$. Write the recurrence for your algorithm and show that its recurrence solves to $O(\log n)$ (e.g., using the Master Method, a recursion tree, or the substitution method).

Problem 2: Circular Shift Divide and Conquer

Suppose you are given an array A of n sorted numbers that has been *circularly shifted* to the right by k positions. For example $\{35, 42, 5, 15, 27, 29\}$ is a sorted array that has been circularly shifted $k = 2$ positions, while $\{27, 29, 35, 42, 4, 15\}$ has been shifted $k = 4$ positions. Give an $O(\log n)$ algorithm to find the largest number in A. You may assume the elements of A are distinct. Write the recurrence for your algorithm and show that its recurrence solves to $O(\log n)$ (e.g., using the Master Method, a recursion tree, or an inductive proof).

Problem 3: Dynamic Programming

Consider the all pairs shortest path problems with nodes $V = \{1,2,...,n\}$. Let A[i, j, k] be the length of the shortest path from node i to node j using a subset of nodes $\{1,2,...,k\}$. Write a dynamic programming equation to find all-pairs shortest path. Also, develop high level pseudocode for filling a 2-D array that would represent all-pairs shortest paths for nodes $V = \{1,2,...,n\}$. What is the running time of your algorithm?

Problem 4: Dynamic Programming and Subsequences

You're consulting for a group of people (who would prefer not to be mentioned here by name) whose jobs consist of monitoring and analyzing electronic signals coming from ships in coastal Atlantic waters. They want a fast algorithm for a basic primitive that arises frequently: "untangling" a superposition of known signals. Specifically, they're picturing a situation in which each of two ships is emitting a short sequence of 0s and 1s over and over, and they want to make sure that the signal they're hearing is simply an *interleaving* of these two emissions, with nothing extra added in.

This describes the whole problem; we can make it a little more explicit as follows. Given a string x consisting of 0s and 1s, we write x^k to denote k copies of x concatenated together. We say that a string x' is a *repetition* of x if it is a prefix of x^k for some number k. So $x' = 10110110110$ is a repetition of $x = 101$.

We say that a string s is an *interleaving* of x and y if its symbols can be partitioned into two (not necessarily contiguous) subsequences s' and s'', so that s' is a repetition of x and s'' is a repetition

of y. (So each symbol in s must belong to exactly one of s' or s''.) For example, if $x = 101$ and $y = 00$, the $s = 100010101$ is an interleaving of x and y since characters 1, 2, 5, 7, 8, 9 form 101101 (a repetition of x), and the remaining characters 3, 4, 6 form 000 (a repetition of y).

In terms of our application, x and y are the repeating sequences from the two ships, and s is the signal we're listening to. We want to make sure that s "unravels" into simple repetitions of x and y. Given an efficient algorithm that takes strings s, x, and y and decides if s is an interleaving of x and y.