Analysis of Algorithms	**Problem Set 4**
CS 6820, Fall 2020	**Due Tuesday, November 24, 6:00pm**

In solving the following problems you are allowed to use, without proof, any facts from the lectures or assigned readings in the Fall 2020 version of CS 6820. Hand in your solutions electronically using CMS. Scanned PDF files are allowed, if you prefer to hand-write your solutions. Each group is responsible for submitting just one solution set.

You may omit either problem 1 or problem 2, but not both. Note, however, that problem 1 is only worth half as many points as problem 2, so you stand to earn a higher score if you omit problem 1. If you turn in solutions to both problems, we will ignore your problem 1 solution and grade the problem 2 solution. A perfect score on this problem set is 30 points, but it's possible to get 35 points by solving problem 2 and earning the 5 points of extra credit on problem 3.

(1) *(5 points)* The *LP containment problem* is the following decision problem. We are given matrices A_0, A_1 and vectors b_0, b_1, and we must decide if the relation

$$\{x \mid A_0 x \preceq b_0\} \quad \subseteq \quad \{x \mid A_1 x \preceq b_1\}$$

holds. You may assume that A_0 and b_0 have the same number of rows, that A_1 and b_1 also have the same number of rows, and that A_0 and A_1 have the same number of columns. (All three of these assumptions are needed, in order for the notation in the problem definition to even make sense.) You may also assume that the entries of A_0, A_1, b_0, b_1 are integers.

Give a polynomial-time algorithm to solve the LP containment problem. You can assume that your algorithm has access to a subroutine that solves linear programs of size N in time $t(N)$, where $t(N)$ is polynomial in N. When analyzing the running time of your own algorithm, you may express it in terms of $t(N)$. In other words, don't worry about the precise running time of the subroutine to solve linear programs, just worry about how many times your algorithm calls the subroutine, and how much time it spends on other operations.

(2) *(10 points)* Let us call an execution of the simplex algorithm "greedy" if, whenever it is choosing the entering and leaving variables in a pivot operation, it selects one of the choices that makes the value of the objective function as large as possible at the end of the pivot (i.e., when the entering variable has been added to the basis and the leaving variable has been removed from it). Construct a family of linear programs LP_n for $n = 1, 2, \ldots$, such that the following properties hold.

1. LP_n has at least n variables.
2. The number of constraints in LP_n is bounded by a polynomial function of n.
3. There exists a greedy execution of the simplex algorithm, applied to LP_n, that performs at least 2^n pivots.

For the third property, you are allowed to assume that the greedy execution of the simplex algorithm chooses an arbitrary initial basis (e.g., the worst possible choice).

Hint: Take a Klee-Minty cube and "shave off" small slivers of it.

(3) *(10 points)* In the MINIMUM k-FOLD SET COVER problem, one is given positive integers k, n, m, a collection of sets $S_1, \ldots, S_n \subseteq [m]$, and corresponding weights $w(1), \ldots, w(n) \in \mathbb{N}$. One aims to choose a finite sequence of indices $i_1, i_2, \ldots, i_t \in [n]$ such that each element $j \in [m]$ belongs to at least k sets in the sequence $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$, and the combined weight $\sum_{s=1}^{t} w(i_s)$ is as small as possible. Note that a set may occur two or more times in the sequence $S_{i_1}, S_{i_2}, \ldots, S_{i_t}$.

Design a randomized algorithm with *expected* running time bounded by $\text{poly}(k, n, m)$, satisfying the following approximation guarantee: when $k \geq \log_2(m)$, the algorithm always outputs a k-fold set cover whose combined weight is at most four times the optimum.

If your algorithm's approximation factor is a constant larger than 4, you will only lose 2 points. If your algorithm is deterministic you will receive 5 points of extra credit.

(4) *(10 points)* In this exercise you will use the Chernoff bound to analyze algorithms for biased coin detection. Define *stream of coin tosses with bias p* to be an infinite sequence of independent random variables $X_1, X_2, X_3, \ldots$, each taking values in $\{0, 1\}$, such that $\mathbb{E}[X_i] = p$ for all i.

(4a) *(5 points)* Consider the problem of detecting whether the bias parameter, p, is greater or less than $\frac{1}{2}$, given that $\left|p - \frac{1}{2}\right|$ is bounded away from zero by a specified amount, $\frac{1}{k}$. One simple algorithm for solving this problem is to fix a number of samples s (depending on k), and take a majority vote of the first s samples. In other words, if $X_1 + \cdots + X_s \geq \frac{s}{2}$ then one guesses that $p > \frac{1}{2}$ and if $X_1 + \cdots + X_s < \frac{s}{2}$ then one guesses that $p < \frac{1}{2}$. Find an explicit constant $C < \infty$ such that whenever $k \geq 2$ and $\left|p - \frac{1}{2}\right| \geq \frac{1}{k}$, the majority-vote algorithm with $s \geq Ck^2$ samples outputs the correct answer with probability at least 0.95.

(4b) *(5 points)* Now suppose the biased coin detection algorithm is not given any lower bound on $\left|p - \frac{1}{2}\right|$. Design an algorithm that sequentially observes a stream of coin tosses $X_1, X_2, \ldots$ and may decide to stop at any time $s \geq 1$ and declare either $p > \frac{1}{2}$ or $p < \frac{1}{2}$. The algorithm is not required to stop — i.e., it may choose to run forever — but it must satisfy the following properties.

1. The decision to stop at time s depends only on the first s coin-flips. In other words, if $X_1, X_2, \ldots$ and $X'_1, X'_2, \ldots$ are two sequences that satisfy $X_i = X'_i$ for $1 \leq i \leq s$, and if the algorithm stops at time s when observing the first sequence, then it must stop at time s when observing the second sequence as well.
2. If $p > \frac{1}{2}$, then with probability at least 0.95 the algorithm eventually stops and declares $p > \frac{1}{2}$.
3. If $p < \frac{1}{2}$, then with probability at least 0.95 the algorithm eventually stops and declares $p < \frac{1}{2}$.
4. If $p = \frac{1}{2}$, then with probability at least 0.95 the algorithm never stops.

When $\left|p - \frac{1}{2}\right| \geq \frac{1}{k}$, your algorithm's expected stopping time should be $O(k^2 \log\log k)$. Partial credit will be awarded for solutions whose expected stopping time is polynomial in k, but asymptotically greater than $O(k^2 \log\log k)$.