CS 221: Computational Complexity **Prof. Salil Vadhan**

Problem Set 1

Assigned: Thu. Feb. 4, 2010 Due: Thu. Feb. 18, 2010 (5 PM sharp)

- You must *type* your solutions. LaTeX, Microsoft Word, and plain ascii are all acceptable. Submit your solutions *via email* to `cs221-hw@seas.harvard.edu`. If you use LaTeX, please submit both the compiled file (`.ps`) and the source (`.tex`). Please name your files `PS1-yourlastname.*`.
- Strive for clarity and conciseness in your solutions, emphasizing the main ideas over low-level details. Do not despair if you cannot solve all the problems! Difficult problems are included to stimulate your thinking and for your enjoyment, not to overwork you. *'ed problems are extra credit.

Problem 1. (Poly-time vs. Linear Space) Prove $\mathbf{P} \neq \mathbf{SPACE}(n)$. (Hint: use translation.)

Problem 2. (An Average-Case Time Hierarchy) Let $f, g : \mathbb{N} \to \mathbb{N}$ be such that $f(n) \log f(n) = o(g(n))$ and g is time-constructible. Show that there is a language $L \in \mathbf{DTIME}(g(n))$ with the property that for every TM M running in time $f(n)$, there is a constant $\varepsilon_M > 0$ such that for all sufficiently large n, M errs in deciding L on at least an ε_M fraction of inputs of length n.

Problem 3. (Linear Programming) A *linear program* consists of a collection of variables $x_1, \ldots, x_n$, a linear objective function $\sum_i c_i x_i$ (specified by the vector $\vec{c} \in \mathbb{Q}^n$), and a collection of constraints each of which is a linear inequality $\sum_i a_i x_i \leq b$ (specified by $\vec{a} \in \mathbb{Q}^n$ and $b \in \mathbb{Q}$). To *solve* a linear program is to find a vector $\vec{x} \in \mathbb{Q}^n$ maximizing the objective function subject to the given constraints. In vector notation, we maximize $\vec{c} \cdot \vec{x}$ subject to $A\vec{x} \leq \vec{b}$, where A is the matrix whose rows are the constraint vectors $\vec{a}$ and the inequality is componentwise.

The decisional version of this problem is $\mathrm{LP} = \{(\vec{c}, A, \vec{b}, K) : \exists \vec{x} \in \mathbb{Q}^n \text{ s.t. } A\vec{x} \leq \vec{b}, \vec{c} \cdot \vec{x} \geq K\}$. The ellipsoid and interior point algorithms show that $\mathrm{LP} \in \mathbf{P}$; you may use this below.

1. Prove that LP is **P**-complete under logspace mapping reductions. (Remark: Integer Programming, the variant of Linear Programming where all numbers in the problem are integers and we solve for integer solutions, is actually **NP**-complete.)
2. Show that a language L has polynomial-sized circuits if and only if there is a sequence of linear programs $P_n = (A_n, \vec{b_n})$ (with no objective function) with $\mathrm{poly}(n)$ variables and $\mathrm{poly}(n)$ constraints and entries from $\{-1, 0, 1\}$ such that for every input $w \in \{0,1\}^n$, $w \in L$ if and only if P_n has a feasible solution $\vec{x}$ whose first n coordinates equal w. Thus another approach to proving $\mathbf{P} \neq \mathbf{NP}$ is to prove a superpolynomial lower bound on the size of linear programs whose feasible solutions project to some **NP** language.

Problem 4. (FACTORING) The FACTORING problem is: given a number n, find its prime factorization. There is no polynomial-time algorithm known for this problem; indeed, much of public-key cryptography relies on its presumed hardness. In this problem, you will explore the complexity of FACTORING. Throughout, you may use the fact that deciding primality is in **P**.

1. Show that FACTORING can be cast as an **NP** search problem (in the sense of Problem 3 on Problem Set 0), and hence can be solved in polynomial time if $\mathbf{P} = \mathbf{NP}$.
2. Show that if FACTORING is **NP**-hard under Cook reductions, then $\mathbf{NP} = \mathbf{co\text{-}NP}$.
3. (*) Give an explicit algorithm for FACTORING such that the running time of this algorithm is polynomial if and only if FACTORING can be solved in polynomial time. (Hint: think diagonalization.) Compare the asymptotic running time of your algorithm to the running time of the fastest possible algorithm for FACTORING. Is your algorithm practical?

Problem 5. (BOOLEAN MATRIX MULTIPLICATION) Given two $n \times n$ matrices A, B with Boolean entries, their *Boolean product* $A \cdot B$ is the $n \times n$ matrix C such that

$$C_{ij} = \bigvee_{k=1}^{n} (a_{ik} \wedge b_{kj})$$

1. Give a logspace algorithm to compute $A \cdot B$ given A and B.
2. Given an $n \times n$ matrix A and $k \in \mathbb{N}$, describe an $O((\log n)(\log k))$-space algorithm to compute A^k, the k'th Boolean power of A. (Hint: first consider k that is a power of 2)
3. Give another proof of Savitch's Theorem using Item 2.