Problem 1. (A Universal NTM)  Show that there is a universal nondeterministic Turing machine whose running time when simulating a nondeterministic TM $N$ (encoded by a string $\alpha$) on input $x$, is at most $c_\alpha \cdot \text{Time}_N(x)$ for some constant $c_\alpha$ depending only on the encoding $\alpha$. (Hint: use the “guess and verify” approach to designing efficient nondeterministic algorithms.)

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

Problem 3. (A Tighter Time Hierarchy Theorem)  Prove that for every constant $\varepsilon > 0$, $\text{DTIME}(n \log^\varepsilon n) \not\subseteq \text{DTIME}(n)$. (Hint: use translation. first try to handle the case that $\varepsilon > 1/2$.)

Problem 4. (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 $\text{LP} = \{(\vec{c}, A, \vec{b}, K) : \exists \vec{x} \in \mathbb{Q}^n \text{ s.t. } A\vec{x} \leq \vec{b}, \vec{c} \cdot \vec{x} \geq K\}$. The ellipsoid and interior point algorithms show that $\text{LP} \in \text{P}$; you may use this below.

1. Prove that $\text{LP}$ is $\text{P}$-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 $\text{NP}$-complete.)
2. Show that a language \( L \) has polynomial-sized circuits if and only if there is a sequence of linear programs \( P_n = (A_n, b_n) \) (with no objective function) with \( \text{poly}(n) \) variables and \( \text{poly}(n) \) 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 \( \text{P} \neq \text{NP} \) is to prove a superpolynomial lower bound on the size of linear programs whose feasible solutions project to some \( \text{NP} \) language.

**Problem 5. (Factorizing)** The Factorizing 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 Factorizing. Throughout, you may use the fact that deciding primality is in \( \text{P} \).

1. Show that Factorizing can be cast as an \( \text{NP} \) search problem (in the sense of Problem 3 on Problem Set 0), and hence can be solved in polynomial time if \( \text{P} = \text{NP} \).

2. Show that if Factorizing is \( \text{NP} \)-hard under Cook reductions, then \( \text{NP} = \text{coNP} \).

3. (*) Give an explicit algorithm for Factorizing such that the running time of this algorithm is polynomial if and only if Factorizing 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 Factorizing. Is your algorithm practical?