**Rules:** As usual, please cite all sources that you may use. Articles not of your authorship will not be graded.

**Ryan’s Factorization Homework.** This homework will familiarize you with some simple randomized factoring methods, culminating in an algorithm that runs in $O(N^{1/4} \cdot \text{poly}(\log N))$ steps. Note this is a quadratic speedup over the obvious algorithm that tries all possible factors. Each problem builds upon the previous one, so it’s probably best to do them in order. **Start Early!**

For simplicity, let $N = pq$ where $p$ and $q$ are prime in the following problems.

1. **Take a Wild Guess.** The first method requires roughly $\sqrt{N}$ GCD computations in the worst case, so in that sense it is no better than trial division. But (we hope) it is still interesting to think about.
   
   (a) Prove that a random $r \in \mathbb{Z}_N^+$ has probability at least $1 - (p+q-1)/N$ of being relatively prime to $N$.
   
   (b) Let $e$ be the base of the natural logarithm.
   
   Prove that for all integers $x > 1$, $(1 - 1/x)^{x-1} \geq 1/e \geq (1 - 1/x)^x$.
   
   (Note: of course it suffices to prove the inequality for all real $x > 1$, if that’s easier.)
   
   (c) Prove that there is a fixed constant $c > 0$ such that the following algorithm factors $N$ in $O(\text{poly}(\log N) \cdot N/(p+q))$ time, with probability at least $c$:
   
   Repeat until a factor is found:
   
   Choose $r \in \mathbb{Z}_N^+$ uniformly at random.
   
   If $\text{GCD}(r,N) \neq 1$ then return $r$ as a non-trivial factor of $N$.

2. **Applying the Birthday Paradox.** The second method is still not asymptotically better than trial division, but it will bring us a little closer to a method that is better.
   
   (a) The birthday paradox says that if you have 23 “randomly chosen” people in a room, then the probability that two in the room have the same birthday is actually quite high, at least $1/2$. (This assumes that each person has a birthday selected uniformly at random over all 365 days in the year.)
   
   Prove that there is a constant $c > 0$ such that, for $r_1, \ldots, r_{\sqrt{N}}$ chosen from $\mathbb{Z}_n^+$ uniformly at random, the probability that there exists $i \neq j$ such that $r_i = r_j$ is at least $c$.
   
   (b) Let $x_1, \ldots, x_{N^{1/4}}$ be uniformly randomly chosen integers from $[1, N]$.
   
   Prove: The probability that there exists $i \neq j$ such that $|x_i - x_j|$ has $p$ as a factor is at least some universal constant $c$.
   
   *Hint:* Use modular arithmetic.
   
   (c) Propose another simple randomized algorithm for factoring $N$ that runs in roughly $N^{1/2}$ steps, and say intuitively why it works. (Your algorithm and analysis should rely crucially on 2b.)
3. **Saving Some Time.** Our final algorithm will run in roughly $N^{1/4}$ steps. To achieve it, we need to take an unexpected turn.

(a) Let $G = (V, E)$ be a directed graph whose nodes have outdegree at most 1. Let $s$ be a node in $G$. You have two node-pointers $p_1$ and $p_2$; both pointers initially point to node $s$. The only access to $G$ available is that you are allowed to move one of the two pointers to the successor of that pointer’s current node (if the successor exists – if there is no successor then such a move would return an error).

Under this kind of graph access, give an algorithm to determine if $s$ has a path to a node in a cycle. That is, you want to detect the following pattern:

![Diagram]

Your algorithm should run in linear time ($O(|V| + |E|)$) with constant additional workspace. That is, all that one really keeps track of are the node-pointers, with perhaps $O(1)$ additional information.

(b) Suppose you are given access to a function $f : \mathbb{Z}^+_N \rightarrow \mathbb{Z}^+_N$ that was chosen uniformly at random over all such functions (i.e. a random oracle).

Prove that, if we choose $x_1 \in \mathbb{Z}^+_N$ at random, and set $x_k = f(x_{k-1})$ for $k = 2, \ldots, N^{1/4}$, then there is a non-zero constant probability (over the choice of $f$ and $x_1$) that $i \neq j$ exist such that $x_i \equiv x_j \pmod{p}$.

(c) Suppose $p$ is given. Let $f$, $x_1$, and $x_k$’s be chosen as in the previous problem. Give an $O(N^{1/4} \cdot \text{poly}(\log N))$ time algorithm that finds $x_i$ and $x_j$ such that $i \neq j$ and $x_i \equiv x_j \pmod{p}$, with probability at least some $c > 0$.

*Hint: Define a graph $G$ where each vertex is a congruence class modulo $p$. (So, the vertices are $\{0, \ldots, p-1\}$.) Put directed edges $(u, f(u))$ in $G$. Pick a random vertex $(x_1 \mod p)$ in $G$...*

(d) You may assume access to a random oracle in this problem. Propose an algorithm that factors $N$ in $O(N^{1/4} \cdot \text{poly}(\log N))$ steps, with probability at least some $c > 0$.

(Note in this case, $p$ is NOT given!)

For extra credit, prove the correctness of your algorithm.