Instructions: Problem 1 is to be handed in by next Friday (as always, theoretical part on Moodle, SAGE exercises via CoCalc).

1. This exercise is about finding/extracting square roots in $\mathbb{F}_p^*$. We fix an odd prime number p, and an element $a \in \mathbb{F}_p^*$.

   (a) How can you check in polynomial time that a is a square or not in $\mathbb{F}_p^*$?

   (b) From now on, assume that a is a square in $\mathbb{F}_p^*$. Prove that if $p \equiv 3 \pmod 4$, then $a^{(p+1)/4}$ is a square root of a.

   (c) Show that the following algorithm terminates, and returns a square root of a. What is the probability that a random element r is not a square mod p ? Implement this algorithm in SAGE.

Algorithm 1 Tonelli-Shanks

Require: A prime p and a square $a \in \mathbb{F}_p^*$.
Ensure: A square root of a in $\mathbb{F}_p^*$.

Write $p - 1 = 2^S Q$ with Q odd
Choose a random element r which is not a square mod p
Set $y := r^Q$ and compute the inverse a' of $a \pmod p$
Set $w := a^{(Q+1)/2}$ and $j = 1$
while $j > 0$ **do**
  Find the smallest $i \geq 0$ such that $(w^2 a')^{2^i} = 1$ and set $j := i$
  if $j > 0$ **then**
    $w := wy^{2^{S-i-1}}$
  end if
end while
return w

   (d) Assume that we have an algorithm that solves square roots in $\mathbb{F}_p^*$ for every prime p, as above. Prove that, for a given square-free integer n, a (probabilistic) algorithm to find square roots modulo n is equivalent to a (probabilistic) algorithm to factor n.

2. Let S denote a set with n elements. For any $x_0 \in S$ and any bijection $f : S \to S$, we consider the iterates $x_{j+1} = f(x_j)$ of f for $j = 1, 2, \ldots$. Let k denote the first index such that $f(x_k) = f(x_j)$ for some $j < k$.

   (a) Show that k is at most n and takes every value between 1 and n with equal probability.

   (b) Show that if one averages over all pairs (f, x_0) where f is a bijection and $x_0 \in S$, the average value of k is $(n + 1)/2$.

   (c) What does this imply about using linear functions $f(x) = ax + b$ in Pollard's ρ method?

3. (a) In SAGE, implement a function `PollardRho(g,h,p)` that solves the discrete logarithm problem $g^x \equiv h \mod p$ in $\mathbb{F}_p^*$ for a generator g using Pollard's rho method. *(Hint: you may use SAGE's xgcd function in your code.)*

   (b) Now test your code for $p = 80783447$ by finding a generator of $\mathbb{F}_p^*$ and running through forty random discrete log computations and checking their correctness.

4. (a) In SAGE, implement a function `IndexCalc(g,h,p)` that solves the discrete logarithm problem $g^x \equiv h \mod p$ in $\mathbb{F}_p^*$ for a generator g using the index calculus method. *(Hint: to simplify the linear algebra* $\mod p-1$ *part, you may assume* $p-1$ *is square-free. You can also treat the prime* $2|(p-1)$ *separately: it suffices to compute Legendre symbols of factor base primes.)*

   (b) Test your code as in the previous exercise for $p = 80783447$ and compare the time used by both algorithms. You can also play around with larger primes (can you do $p = 4294967291$?) and compare.