18.783 Elliptic Curves **Spring 2015**
Problem Set #2 **Due: 02/20/2015**

These problems are related to the material covered in Lectures 3-4. I have made every effort to proof-read the problems, but there may well be errors that I have missed. The first person to spot each error will receive 1-3 points of extra credit on their problem set, depending on the severity of the error.

Instructions: Solve **both** Problems 1 and 2 and **one** of Problems 3 and 4. Then complete Problem 5, which is a survey.

Problem 1. Complexity analysis (30 points)

Let F_k denote the kth Fibonacci number, defined by:

$$F_1 := 1, \qquad F_2 := 1, \qquad F_k := F_{k-2} + F_{k-1} \qquad (k \geq 2). \tag{1}$$

Analyze the time complexity of computing F_k modulo an n-bit prime p using each of the algorithms listed below. You may assume that $k > n$. Your answers should be in the form $O(f(k,n))$ and as tight as you can make them. Use $\mathsf{M}(n)$ to denote the complexity of multiplying two n-bit integers.

(a) Compute the integers $F_1, F_2, \ldots, F_k$ via (1), and then reduce F_k mod p.

(b) Same as (a), except perform all computations modulo p (i.e., work in $\mathbb{F}_p$).

(c) Assume $p = \pm 1 \bmod 5$. Compute the roots ϕ and ψ of $x^2 - x - 1$ in $\mathbb{F}_p$ (using a probabilistic root-finding algorithm), and then compute $F_k = \frac{\phi^k - \psi^k}{\phi - \psi}$ in $\mathbb{F}_p$.

(d) Assume $p \neq \pm 1 \bmod 5$. Represent $\mathbb{F}_{p^2}$ as $\mathbb{F}_p[\phi]/(\phi^2 - \phi - 1)$, let $\psi = 1 - \phi$, and then compute $F_k = \frac{\phi^k - \psi^k}{\phi - \psi}$ as an element of $\mathbb{F}_{p^2}$ that actually lies in $\mathbb{F}_p$.

(e) Compute the kth power of $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$ in $\mathrm{GL}_2(\mathbb{F}_p)$ and output its upper right entry.

(f) Assume k is a power of 2. Use the identities

$$F_{2i-1} = F_i^2 + F_{i-1}^2$$
$$F_{2i} = (2F_{i-1} + F_i)F_i$$

to compute (F_1, F_2), (F_3, F_4), (F_7, F_8),$\ldots$ (F_{k-1}, F_k), working modulo p throughout.[1]

(g) Notably absent from the above list is the standard recursive algorithm:

```
def fibonacci(k):
    if k <= 2: return 1
    return fibonacci(k-1) + fibonacci(k-2)
```

What is the time complexity of computing `fibonacci(k)` and reducing it mod p? If instead of returning `1` in the base case you return `GF(p)(1)`, the algorithm will work in $\mathbb{F}_p$ rather than $\mathbb{Z}$. What is its complexity then?

[1] This approach can be generalized to arbitrary k.

Problem 2. Computing rth roots (40 points)

In Lecture 4 we saw how to compute rth roots in a finite field $\mathbb{F}_q$ using a probabilistic root-finding algorithm. In this problem you will implement an entirely different approach for computing rth roots that works in any finite cyclic group G, including the cyclic group $\mathbb{F}_q^*$. In addition to being more general, this method is usually faster than polynomial root-finding for computing rth roots in $\mathbb{F}_q^*$ (but this depends on r and q), and it has the virtue of being deterministic (assuming we are given a generator for G, a probabilistic approach might be used initially to find such a generator).

We assume without loss of generality that r is prime (to compute nth roots, successively compute rth roots for each prime r dividing n, with multiplicity). To simplify the notation, let us write the group G additively; so an *rth root* of $\gamma \in G$ is an element $\rho \in G$ for which $r\rho = \gamma$. We use $|\gamma|$ to denote the *order* of γ, the least positive integer m for which $m\gamma = 0$.

Prove the following statements:

(a) For all $\gamma \in G$ and $n \in \mathbb{Z}$ we have $|n\gamma| = |\gamma| / \gcd(n, |\gamma|)$.

(b) If r does not divide $|G|$, then there is an integer s such that for all $\gamma \in G$ the element $\rho = s\gamma$ is the unique rth root of γ.

(c) If r does divide $|G|$, then the number of rth roots of each $\gamma \in G$ is either 0 or r. In the latter case, the rth roots of γ do not necessarily lie in $\langle\gamma\rangle$ (give an example).

(d) Suppose r divides $|G|$. Let $|G| = ar^k$, where $r \nmid a$. Let $\delta \in G$ be an element of order r^k, let γ be any element of G, and $\alpha = a\gamma$ and $\beta = r^k\gamma$. The following hold:

(i) $\alpha = x\delta$ for some integer $x \in [1, r^k]$.

(ii) If r does not divide x then there is no $\rho \in G$ for which $r\rho = \gamma$.

(iii) If r divides x, and s and t are integers satisfying $sa + tr^{k+1} = 1$, then the element $\rho = s(x/r)\delta + t\beta$ satisfies $r\rho = \gamma$.

The element δ in part (d) is a generator for the r-Sylow subgroup of G. Given a generator σ for G, we can obtain such a δ by computing $\delta = a\sigma$. The integer x is the *discrete logarithm* of α with respect to δ.

Implement the following algorithm for computing an rth root of γ in a cyclic group G of order ar^k, where r is a prime that does not divide a, given $\delta \in G$ of order r^k:

1. If $k = 0$ then compute $s = 1/r \bmod a$ and return $\rho = s\gamma$.
2. Compute $\alpha = a\gamma$ and $\beta = r^k\gamma$.
3. Compute the discrete logarithm of x of α with respect to δ by brute force: check whether $\alpha = x\delta$ for each x from 1 to r^k (this holds for some x, by part (a) of 4).[2]
4. If r does not divide x then return `null`.
5. Compute s and t such that $sa + tr^{k+1} = 1$ using the extended Euclidean algorithm.
6. Return $\rho = s(x/r)\delta + t\beta$.

[2] We will learn much better ways to compute this discrete logarithm later in the course. For the moment, assume r^k is small (for finite fields $\mathbb{F}_q$, this is usually the case, even when q is very large).

The return value `null` is used to indicate that γ does not have any rth roots in G. To compute $s = 1/r \bmod a$ in Sage, use: `s=1/mod(r,a)`. To compute s and t such that $sa + tr^{k+1} = 1$, use: `d,s,t=xgcd(a,r**(k+1))` (here $d = \gcd(a, r^{k+1})$ is 1).

The Python language used by Sage is untyped, so your algorithm can be used to compute rth roots in any cyclic group that Sage knows how to represent; it will automatically perform operations in whatever group the inputs δ and γ happen to lie in. To test your algorithm, you may find it useful to work in the additive group of the ring $\mathbb{Z}/n\mathbb{Z}$, where $n = ar^k$, which you can create in Sage using `Zn=Integers(n)`. You can then use `delta=Zn(a)` to create an element of $\mathbb{Z}/n\mathbb{Z}$ with additive order r^k.

Let E be the elliptic curve $y^2 = x^3 + 31415926x + 21782818$ over $\mathbb{F}_p$ with $p = 2^{255} - 19$. The group $E(\mathbb{F}_p)$ is cyclic, of order $n = 2 \cdot 5 \cdot 11 \cdot m$, where

$$m = 52632767835143725192532265913039958115389053036079993689861929868257541157 3,$$

and the point $P = (x_0 : y_0 : 1)$, where $x_0 = 1$ and

$$y_0 = 13134814009004874435178595602484727393250015841439437156743965100178108553104,$$

is a generator for $E(\mathbb{F}_p)$. Thus for $r = 2, 5, 11$ you can use $\delta = (n/r)P$ as a generator of the r-Sylow subgroup (which in each case has order r).

Let c be the least prime greater than the integer formed by the last four digits of your student ID. Let $Q = (x_1 : y_1 : 1)$, where

$$x_1 = 60444284987523100146759825828591971659407938078156388823988531587461799985 24,$$
$$y_1 = 15213333001572637024476234182397935874270965424629854864096629656428211884435$$

(e) Use your algorithm to find an rth root R of $\gamma = cQ$, for $r = 2, 5, 11$. Note that you can easily check your result by testing whether `r*R==c*Q` holds using Sage (please be sure to do this; your grade on this problem will be depend heavily on whether you compute the points R correctly or not). In your answer you only need to list the point R for each value of r, you don't need to include your code. Be sure to format your answer so that the coordinates of R all fit on the page.

Problem 3. Exponentiation (30 points)

An *addition chain* for a positive integer n is an increasing sequence of integers $(c_0, \ldots, c_m)$ with $c_0 = 1$ and $c_m = n$ such that each entry other than c_0 is the sum of two (not necessarily distinct) preceeding entries. The *length* of an addition chain is the index m of the last entry. When computing a^n with a generic algorithm, the exponents of the powers of a^k computed by the algorithm define an addition chain whose length is the number of multiplications performed. For example, using left-to-right binary exponentiation to compute a^{47} yields the addition chain $(1, 2, 4, 5, 10, 11, 22, 23, 46, 47)$, and right-to-left binary exponentiation yields the addition chain $(1, 2, 3, 4, 7, 8, 15, 16, 32, 47)$, both of which have length 9.

(a) For $n = 715$, determine the addition chains given by: (i) left-to-right binary, (ii) right-to-left binary, (iii) fixed-window, and (iv) sliding-window exponentiation, using a window of size 2 for (iii) and (iv).

(b) Find an addition chain for $n = 715$ that is shorter than any you found in part (a).

(c) Repeat part (a) for the integer N obtained by adding 990,000 to the last 4 digits of your student ID, using a window of size 3 for cases fixed and sliding window cases.

(d) Find the shortest addition chain for N that you can. There is a good chance you can do better than any of the chains you found in part (c).

In groups where inversions are cheap (such as the group of points on an elliptic curve), it can be advantageous to use *signed* binary representations of exponents, where we write the exponent n in the form $n = \sum n_i 2^i$ with $n_i \in \{-1, 0, 1\}$. Such a representation is generally not unique, but there is a unique signed representation with the property that no pair of adjacent digits are both nonzero. This is known as *non-adjacent form* (NAF). The NAF representation of 47, for example, is $10\bar{1}000\bar{1}$, where $\bar{1}$ is used to compactly denote -1.

To construct the NAF representation one begins by writing n in binary with a leading 0, and then successively replaces the least significant block of the form $01\cdots1$ with $10\cdots0\bar{1}$ until there are no adjacent nonzero digits. For example, the computation for 47 proceeds as $0101111, 11000\bar{1}, 10\bar{1}000\bar{1}$, which reduces the number of nonzero digits from 5 to 3. Even though the length is increased by 1, the total cost of exponentiation may be reduced.

An *addition-subtraction* chain extends the definition of an addition chain by allowing $c_k = c_i \pm c_j$. Exponentiation using the NAF representation defines an addition-subtraction chain. For example, using left-to-right binary exponentiation, the NAF representation of 47 yields the chain $(1, 2, 4, 3, 6, 12, 24, 48, 47)$, which is shorter than the addition chain $(1, 2, 4, 5, 10, 11, 22, 23, 46, 47)$ given by standard left-to-right binary exponentiation.

(e) Compute addition-subtraction chains for $n = 715$ and the integer N defined in part (c) using left-to-right binary exponentiation with the NAF representation.

(f) Find the shortest addition-subtraction chains for n and N that you can.

Problem 4. Cornacchia's algorithm (30 points)

Cornacchia's algorithm computes primitive solutions (x, y) to the Diophantine equation

$$x^2 + dy^2 = m, \tag{2}$$

where d and m are positive integers. A *primitive* solution has x and y relatively prime. Typically $m = p$ or $m = 4p$, where p is a prime, but the algorithm works for any m, provided it is given a square root of $-d \bmod m$.

The algorithm is very simple: it uses a partial Euclidean algorithm that terminates as soon as the sequence of remainders r_i drops below the square root of $r_0 = m$.

1. Let $r_0 = m$ and $r_1 = r$, where $r^2 \equiv -d \bmod m$ and $0 \le r \le m/2$.
2. Compute $r_{i+2} = r_i \bmod r_{i+1}$ until $r_k^2 < m$ is reached.
3. If $(m - r_k^2)/d$ is the square of an integer s, return the solution (r_k, s).
   Otherwise, return `null`.

It is clear that if the algorithm returns (r_k, s), then it is a solution to (2). It is not so clear that the algorithm will necessarily find a primitive solution if one exists, but this is true; see the article *On the solution of* $x^2 + dy^2 = m$ by Basilla (MR2062797) for a short and elementary proof.[3] Note that if m is square-free (in particular, if m is prime), then every solution to (2) is primitive, but this is not true if m is not square-free (this fact is pertinent to part 5 below, where $m = 4p$).

(a) Implement this algorithm in Sage. Use `mod(-d,m).is_square()` to test if $-d$ has a square root mod m, and use `int(mod(-d,m).sqrt())` to get a square root.

(b) You may recall Fermat's "Christmas theorem", which states that an odd prime p is the sum of two squares if and only if $p \equiv 1 \bmod 4$. You may also recall that -1 is a square modulo an odd prime p if and only if $p \equiv 1 \bmod 4$.

Let n be the integer corresponding to the last 4 digits of your student ID. For the least prime $p > n \cdot 10^{100}$ congruent to 1 mod 4, write p as the sum of two squares.

(c) Fermat also proved that a prime p can be written in the form $p = x^2 + 3y^2$ if and only if $p \equiv 1 \bmod 3$, which is equivalent to the condition that -3 is a square mod p.

For the least prime $p > n \cdot 10^{100}$ congruent to 1 mod 3, write p in the form $p = x^2 + 3y^2$.

(d) Show that this does not work for $d = 5$ by finding a prime p for which -5 is a square modulo p but p cannot be written in the form $x^2 + 5y^2$. Empirically determine a stronger congruence condition on p that guarantees not only that is -5 a square mod p, but also that p can be written in the form $x^2 + 5y^2$. Then find the least prime $p > n \cdot 10^{100}$ that satisfies your condition and write it in the form $p = x^2 + 5y^2$.

(e) Let E be the elliptic curve $y^2 = x^3 - 35x - 98$. As you found in Problem Set 1, the integer $a_p = p + 1 - \#E(\mathbb{F}_p)$ is zero precisely when -7 is not a square modulo p. When -7 is a square modulo p, the integer a_p satisfies the equation $4p = a_p^2 + 7y^2$, for some integer y. Prove that this equation has no primitive solutions for $p > 2$, and in this case it has a solution if and only if the equation $p = u^2 + 7v^2$ has a solution.

Use your algorithm to find a solution to $p = u^2 + 7v^2$ for the least prime $p > n \cdot 10^{100}$ for which -7 is a square modulo p, and use this to deduce the absolute value of a_p. Determine the sign of a_p, by finding a random point $P \in E(\mathbb{F}_p)$ for which only one of $(p + 1 - a_p)P$ and $(p + 1 + a_p)P$ is zero.

Problem 5. Survey

Complete the following survey by rating each of the problems you attempted on a scale of 1 to 10 according to how interesting you found the problem (1 = "mind-numbing," 10 = "mind-blowing"), and how difficult you found it (1 = "trivial," 10 = "brutal"). Also estimate the amount of time you spent on each problem to the nearest half hour.

[3] N.B., there is an obvious typo in step 3 of the algorithm in Basilla's paper which is corrected above.

	Interest	Difficulty	Time Spent
Problem 1			
Problem 2			
Problem 3			
Problem 4			

Also, please rate each of the following lectures that you attended, according to the quality of the material (1="useless", 10="fascinating"), the quality of the presentation (1="epic fail", 10="perfection"), and the novelty of the material (1="old hat", 10="all new"). For the lecture on 2/10 which was cancelled, base your rating on the lecture notes.

Date	Lecture Topic	Material	Presentation	Novelty
2/10	Integer arithmetic and finite fields		xxxxxxxxxx	
2/12	Finite field arithmetic			

Feel free to record any additional comments you have on the problem sets or lectures.

18.783 Elliptic Curves
Spring 2015