18.783 Elliptic Curves **Spring 2017**
Problem Set #4 **Due: 03/12/2017**

Description

These problems are related to the material covered in Lectures 7-9.

Instructions: Pick any **two** of Problems 1-6 to solve. Then complete Problem 7, which is a short survey. Your solutions are to be written up in latex and submitted as a pdf-file with a filename of the form `SurnamePset4.pdf` via e-mail to `drew@math.mit.edu` by **11:59 p.m.** on the date due.

Collaboration is permitted/encouraged, but you must identify your collaborators, and any references not listed in the course syllabus. The first to spot each non-trivial typo/error in the problem sets or lecture notes will receive 1-5 points of extra credit.

Problem 1. The Hasse invariant (50 points)

Let $E\colon y^2 = f(x)$ be an elliptic curve over $\mathbb{F}_p$, where p is an odd prime and f is a monic cubic. The *Hasse invariant* $H_p(E)$ is the coefficient of x^{p-1} in $f(x)^{(p-1)/2} \in \mathbb{F}_p[x]$.

(a) Prove that $\#E(\mathbb{F}_p) = 1 + \sum_{a\in\mathbb{F}_p} f(a)^{(p-1)/2}$ holds as an identity in $\mathbb{F}_p$.

(b) Prove that for any integer $k \geq 0$ we have

$$\sum_{a\in\mathbb{F}_p} a^k = \begin{cases} -1 & \text{if } k \text{ is a nonzero multiple of } p-1 \\ 0 & \text{otherwise} \end{cases}$$

(c) Use (b) to show that $\#E(\mathbb{F}_p) = 1 - H_p(E)$ holds as an identity in $\mathbb{F}_p$, and that $H_p(E)$ is therefore an element of $\mathbb{F}_p$ that is equal to the trace of Frobenius $\operatorname{tr}\pi_E$ modulo p. Conclude that $H_p(E)$ uniquely determines $\#E(\mathbb{F}_p)$ for all $p > 13$.

(d) Show that for $p = 13$ there exist elliptic curves E_1 and E_2 over $\mathbb{F}_p$ for which we have $H_p(E_1) = H_p(E_2)$ but $\#E_1(\mathbb{F}_p) \neq \#E_2(\mathbb{F}_p)$.

(e) Suppose E is a *Legendre curve* $E_\lambda\colon y^2 = x(x-1)(x-\lambda)$ with $\lambda \in \mathbb{F}_p - \{0,1\}$. Prove that $H_p(E) = (-1)^n S_p(\lambda)$, where $n = (p-1)/2$ and $S_p \in \mathbb{F}_p[x]$ is the polynomial

$$S_p(x) = \sum_{i=0}^{n} \binom{n}{i}^2 x^i.$$

Let us now generalize to the case where $E\colon y^2 = f(x)$ is an elliptic curve over a finite field $\mathbb{F}_q$ of odd characteristic p, with f a monic cubic. For any positive integer r, define $H_{p^r}(E)$ to be the coefficient of x^{p^r-1} in $f(x)^{(p^r-1)/2}$ (for $r = 1$ this generalizes our definition of $H_p(E)$ to elliptic curves defined over any finite extension $\mathbb{F}_p$).

(f) Show that $\#E(\mathbb{F}_q) = 1 - H_q(E)$. Conclude that $H_q(E) \in \mathbb{F}_p \subseteq \mathbb{F}_q$ and that we have $\operatorname{tr}\pi_E \equiv 0 \bmod p$ if and only if $H_q(E) = 0$.

(g) Prove that the identity $H_{p^{r+1}}(E) = H_{p^r}(E)H_p(E)^{p^r}$ holds for all integers $r > 0$. Conclude that $H_q(E) = 0$ if and only if $H_p(E) = 0$.

(h) Show that, up to isomorphism, every elliptic curve E over $\overline{\mathbb{F}}_p$ is a Legendre curve $E_\lambda\colon y^2 = x(x-1)(x-\lambda)$, for some $\lambda \in \overline{\mathbb{F}}_p - \{0,1\}$.

(i) Show that for all $\lambda \in \overline{\mathbb{F}}_p - \{0,1\}$ we have $H_p(E_\lambda) = 0$ if and only if $S_p(\lambda) = 0$. Conclude that (up to isomorphism), among the infinitely many elliptic curves $E/\overline{\mathbb{F}}_p$, only a finitely number have $\operatorname{tr}\pi_E \equiv 0 \bmod p$.

In a later lecture we will show that an elliptic E curve over a finite field of characteristic $p > 0$ is supersingular if and only if $\operatorname{tr}\pi_E \equiv 0 \bmod p$. You have thus proved that there are only finitely many supersingular elliptic curve defined over finite fields of characteristic p (in fact this holds for all fields of characteristic p, not just finite fields).

Problem 2. Computing the L-function of an elliptic curve (50 points)

Let $E\colon y^2 = x^3 + Ax + B$ be an elliptic curve over $\mathbb{Q}$; without loss of generality, we may assume $A, B \in \mathbb{Z}$ with $B \neq 0$. As you may recall from Lecture 1, the L-function of an elliptic curve can defined as an Euler product of the form

$$L(E,s) = \prod_{p|\Delta(E)} (\cdots)^{-1} \prod_{p\nmid\Delta(p)} (1 - a_p p^{-s} + p^{1-2s})^{-1},$$

where $\Delta(E) = -16(4A^3+27B^2)$ is the discriminant of E and $a_p := p+1-\#E_p(\mathbb{F}_p)$ is the trace of Frobenius of the elliptic curve $E_p/\mathbb{F}_p$ obtained by reducing A and B modulo p. Ignoring the finite set of bad primes p that divide $\Delta(E)$ (which are easy to address and will be discussed in a later lecture), computing the L-function of an elliptic curve amounts to computing the sequence of Frobenius traces a_p over good primes p.

Of course we cannot compute all of the infinitely many a_p, but if we compute them for all good primes p up to some bound N, we can approximate $L(E,s)$ or any of its derivatives to high precision (assuming we also know the Euler factors at bad primes). Such computations are critical to testing the Birch and Swinnerton-Dyer conjecture, for example, which relates the order of vanishing of $L(E,s)$ at $s = 1$ to the rank of $E(\mathbb{Q})$.

By applying Schoof's algorithm to each of $E_p/\mathbb{F}_p$ we can compute a_p for all good primes $p \le N$ in time quasi-linear in N (you can use the results of Problem 6 to get a precise estimate). Alternatively, we could use the baby-steps giant-steps algorithm from Problem 4 to obtain a running time that is quasi-linear in $N^{5/4}$; in practice this turns out to be faster than using Schoof's algorithm unless N is extremely large.

Neither of these approaches takes advantage of the fact that we are working with reductions E_p of a *fixed* elliptic curve $E/\mathbb{Q}$. In this problem you will use this fact to develop an algorithm with a complexity is both practically and asymptotically faster than using Schoof's algorithm. It also efficiently generalizes to higher genus curves, which is not true of either Schoof's algorithm or the baby-steps giant-steps approach.

Our basic strategy is to compute the Hasse Invariant $H_p(E_p)$ defined in Problem 1 as the coefficient of x^{p-1} in $f(x)^{(p-1)/2}$, where $f(x) = x^3 + Ax + B$ is the cubic defining our elliptic curve $E\colon y^2 = f(x)$, except now $f \in \mathbb{Z}[x]$ is an integer polynomial. If we iteratively compute $f(x), f(x)^2, f(x)^3, \cdots, f(x)^{\lfloor N/2\rfloor}$ as integer polynomials and for each prime $p = 2n+1$ extract the coefficient of x^{2n} from the polynomial $f(x)^n$ and reduce its value modulo $p = 2n+1$, then we will have determined $a_p \equiv H_p(E_p) \bmod p$ for all primes $p \le N$ where E has good reduction. As shown in part (c) of Problem 1, this will determine the trace of Frobenius $a_p \in \mathbb{Z}$ for all $p > 13$ (and for $p \le 13$ we can just compute $a_p = p + 1 - \#E_p(\mathbb{F}_p)$ using brute force).

(a) Show that at first glance this is a terrible idea by analyzing its complexity as a function of N, assuming the coefficients A and B each have $O(\log N)$ bits. Keep in mind that we are working in $\mathbb{Z}$, so the coefficient sizes in $f(x)^n$ increase with n.

Even using fast multiplication ($\mathsf{M}(n) = O(n \log n \log\log n)$) your answer to (a) will be far from our quasi-linear goal. There are two keys to obtaining an efficient algorithm; the first is to avoid computing all the coefficients of $f(x)^n$.

(b) Let $f = \sum f_i x^i \in \mathbb{Z}[x]$ be a polynomial of degree d, and for each $m \in \mathbb{Z}$ and $n \in \mathbb{Z}_{\geq 0}$ let $f_m^n \in \mathbb{Z}$ denote the coefficient of x^m in $f(x)^n$ (so $f_m^n = 0$ for $m < 0$). Derive the identity

$$m f_0 f_m^n = \sum_{i=1}^{d} ((n+1)i - m) f_i f_{m-i}^n,$$

which expresses the coefficient f_m^n of x^m in $f(x)^n$ as a linear combination of d coefficients $f_{m-1}^n, f_{m-2}^n, \ldots, f_{m-d}^n$ of lower order terms.

(c) For each $m \in \mathbb{Z}$ define the row vector $v_m^n := [f_{m-d+1}^n, f_{m-d+2}^n, \ldots, f_m^n] \in \mathbb{Z}^d$. The linear recurrence in (a) determines integer matrices $M_m^n \in \mathbb{Z}^{d \times d}$, with coefficients depending on $m, n, f_0, \ldots, f_d \in \mathbb{Z}$, that satisfy

$$m f_0 v_m^n = v_{m-1}^n M_m^n$$

for all $m, n \geq 1$. Write down the matrix M_m^n explicitly for the case $d = 3$.

(d) Show that provided $f_0 \neq 0$, for all $m, n \geq 1$ we have

$$v_m^n = \frac{1}{f_0^{m-n} m!} V_0 M_1^n \cdots M_m^n,$$

where $V_0 := [0, \ldots, 0, 1] \in \mathbb{Z}^d$.

(e) Now suppose $p = 2n + 1$ is an odd prime not dividing f_0. For convenience let us specialize to the case $d = 3$ of interest and define

$$M_m := \begin{bmatrix} 0 & 0 & (3-2m)f_3 \\ 2m f_0 & 0 & (2-2m)f_2 \\ 0 & 2m f_0 & (1-2m)f_1 \end{bmatrix}.$$

Notice that the matrices M_m do not depend on n. Prove that if $p \nmid f_0$ is prime and $n = (p-1)/2$ then

$$v_{2n}^n \equiv \frac{-1}{f_0^n} V_0 M_1 \cdots M_{2n} \bmod (2n+1).$$

Now $v_{2n}^n = [f_{2n-2}^n, f_{2n-1}^n, f_{2n}^n]$, so we can use this to compute $H_p(E_p) \equiv f_{2n}^n \bmod p$ for all good primes $p = 2n+1$ that do not divide f_0 using the cubic $f(x) = x^3 + Ax + B$ defining E (so $f_0 = B, f_1 = A, f_2 = 0, f_3 = 1$).

Note that we regard A and B as fixed relative to N, so like the prime $p \leq 13$ we could use brute force (or Schoof's algorithm) to handle primes $p|B$.

Now comes the second key to obtaining a quasi-linear running time, which is to use a recursive strategy for computing the matrix products $M_1 \cdots M_{2n} \bmod (2n+1)$. At first glance this seems hard, since the modulus is changing with n. But we will take a recursive approach that allows for more general moduli. Let $M_1, \ldots, M_N \in \mathbb{Z}^{d \times d}$ be a sequence of integer matrices and let $m_1, \ldots, m_N \in \mathbb{Z}_{\geq 1}$ be a sequence of integer moduli. For $k = 1, \ldots, N$ define the reduced partial products

$$C_k := M_1 \cdots M_{k-1} \bmod m_k$$

(the matrix M_N is never used and could be omitted).

(f) Assume N is a power of 2. Show how to reduce the problem of computing $C_1, \ldots, C_N$ to a problem involving $N/2$ integer matrices and $N/2$ integer moduli where the total number of bits involved is essentially the same (you can view d as a constant).

(g) Analyze the complexity of the recursive algorithm given by (f) under the assumption that the integers appearing in the matrices M_k and the moduli m_k all have bit-sizes bounded by $O(\log N)$. Your bound should be quasi-linear in N. By the prime number theorem there are approximately $N/\log N$ primes $p \leq N$. What is the average running time of your algorithm as a function of $\log p$?

(h) Let $A = 42$ and let B be the least integer greater than or equal to the last 4 digits of your student ID such that $\Delta(E) = 4A^3 + 27B^2 \neq 0$. Using the matrices defined in (e) and the moduli m_k defined by

$$m_k = \begin{cases} k & \text{if } k > 13 \text{ is a prime not dividing } B \text{ or } \Delta(E), \\ 1 & \text{otherwise,} \end{cases}$$

use the recursive algorithm in (f) to compute the Frobenius traces a_p for primes p in the interval $(13, N]$ not dividing B or $\Delta(E)$, where $N = 2^k$ with $k = 10, 11, \ldots, 15$. For each value of N, report the sum of the traces, along with the running time of your algorithm.

Problem 3. The probability of ℓ-torsion (50 points)

Let ℓ be a prime. In this problem you will determine the probability that a random[1] elliptic curve $E/\mathbb{F}_p$ has an $\mathbb{F}_p$-point of order ℓ, where p is either a fixed prime much larger than ℓ, or a prime varying over some large interval. Let $\pi = \pi_E$ be the Frobenius endomorphism of E, and let $\pi_\ell \in \mathrm{GL}_2(\mathbb{F}_\ell)$ denote the matrix corresponding to the action of the Frobenius endomorphism of E on the ℓ-torsion subgroup $E[\ell]$ with respect to some chosen basis (here we have identified $\mathbb{F}_\ell$ with $\mathbb{Z}/\ell\mathbb{Z}$). The matrix π_ℓ is only defined up to conjugacy, since it depends on the choice of basis, but its trace $\operatorname{tr} \pi_\ell = \operatorname{tr} \pi \bmod \ell$ and $\det \pi_\ell = \deg \pi = p \bmod \ell$ are uniquely determined. We will make the heuristic assumption that π_ℓ is uniformly distributed over $\mathrm{GL}_2(\mathbb{F}_\ell)$ as E varies over elliptic curves defined over $\mathbb{F}_p$ and p varies over integers in some large interval (one can prove that the distribution of π_ℓ converges to the uniform distribution on $\mathrm{GL}_2(\mathbb{F}_\ell)$ as $p \to \infty$).

[1]There are several ways to vary the random elliptic curve $E/\mathbb{F}_p$. We will just pick curve coefficients A and B at random and ignore the negligible number of cases where the discriminant is 0.

(a) Determine the probability that $E(\mathbb{F}_p)[\ell] = E[\ell]$, both for a fixed p (in which case the answer will depend on $p \bmod \ell$), and for p varying over some large interval (assume every possible value of $p \bmod \ell$ occurs equally often).

Use your answer to derive a heuristic estimate for the probability that $E(\mathbb{F}_p)$ is cyclic, for large p, by estimating the probability that $E(\mathbb{F}_p)[\ell] \neq E[\ell]$ for all ℓ, assuming that these probabilities are independent.[2] Use Sage to compute the product of these probabilities for primes ℓ bounded by 50, 100, 200, 500, and then given an estimate that you believe is accurate to at least 4 decimal places for all sufficiently large p.

Now test your heuristic estimate using the following Sage script

```
cnt=0
for i in range(0,1000):
    p=random_prime(2^20,2^19); F=GF(p)
    A=F.random_element(); B=F.random_element()
   if EllipticCurve([A,B]).abelian_group().is_cyclic():
        cnt += 1
print cnt/1000.0
```

In the unlikely event that you stumble upon a singular curve, simply rerun the test. Run this script three times (be patient, it may take a few minutes), and compare the results to your estimate.

(b) Show that a necessary and sufficient condition for $E(\mathbb{F}_p)[\ell] \neq \{0\}$ is

$$\operatorname{tr} \pi_\ell \equiv \det \pi_\ell + 1 \pmod{\ell}.$$

(c) Under our heuristic assumption, to determine the probability that $E(\mathbb{F}_p)$ contains a point of order ℓ, we just need to count the matrices π_ℓ in $\mathrm{GL}_2(\mathbb{F}_\ell)$ that satisfy this condition. Your task is to derive a combinatorial formula for this probability as a rational function in ℓ. Do this by summing over the possible values of $\det \pi_\ell$, so that you can also compute the probability for any fixed value of p, which determines $\det \pi_\ell \equiv p \bmod \ell$. For each nonzero value of $d = \det \pi_\ell \in \mathbb{F}_\ell$, you want to count the number of matrices in $\mathrm{GL}_2(\mathbb{F}_\ell)$ that have determinant d and trace $d+1$.

As a warm-up, for $\ell = 3$ use Sage to count the number of matrices $\pi_\ell \in \mathrm{GL}_2(\mathbb{F}_3)$ with trace $d+1$ for $d = 1$ and $d = 2$. You can then compute the probability of ℓ-torsion for a fixed $p \equiv 1 \bmod 3$ or $p \equiv 2 \bmod 3$, and also the average probability for varying p by averaging over the 2 possible values of $d = \det \pi_\ell \equiv p \bmod \ell$.

You can solve this problem with purely elementary methods, but if you know a little representation theory you may find it helpful to consult the character table for $\mathrm{GL}_2(\mathbb{F}_\ell)$ (be sure to list your sources). Assume initially that ℓ is odd, and after obtaining your formula, verify that it also works when $\ell = 2$.

(d) For $\ell = 3, 5, 7$ do the following: Pick two random primes $p_1, p_2 \in [2^{29}, 2^{30}]$, with $p_1 \equiv 1 \bmod \ell$ and $p_2 \not\equiv 1 \bmod \ell$, and for each prime generate 1000 random elliptic curves $E/\mathbb{F}_p$. Count how often $\#E(\mathbb{F}_p)$ is divisible by ℓ, and compare this with the value predicted by the formulas you derived in part (c).

[2] This assumption is false, but the extent to which it is false becomes negligible as $p \to \infty$.

Problem 4. Fast order algorithms (50 points)

Let α be an element of a generic group G, written additively. Let N be a positive integer for which $N\alpha = 0$, and let $p_1^{e_1} \cdots p_r^{e_r}$ be the prime factorization of N. An algorithm that computes the order of α given N and its prime factorization is known as a *fast order algorithm*. It's fast because the knowledge of N and its factorization allows the algorithm to run in polynomial time (polynomial in $n = \log N$); determining the order of α without being given N provably takes exponential time.

The naïve fast order algorithm given in class is rather inefficient. This is irrelevant in the context of computing the order of a point in $\#E(\mathbb{F}_q)$ with the baby-steps giant-steps method; the complexity is dominated by the time to determine N. But this is not the case in every application. In this problem you will analyze two more efficient approaches.

When giving time complexity bounds for generic group algorithms, we simply count group operations, since these are assumed to dominate the computation (so integer arithmetic costs nothing). Space complexity is measured by counting the maximum number of group elements that the algorithm must store simultaneously, but for this problem we will just be concerned with time complexity. You may use the fact that the maximum number of distinct primes dividing an integer N is bounded by $O(n/\log n)$, where $n = \log N$, which follows from the prime number theorem. All your complexity bounds should be specified in terms of n.

(a) The fast order algorithm given in class begins by initializing $m = N$ and then for each prime $p_i | N$ it repeatedly replaces m by m/p_i so long as $p_i | m$ and $(m/p_i)\alpha = 0$. Analyze the time complexity of this algorithm in the worst case, and give separate asymptotic bounds for inputs of the form 2^k and $p_1 \cdots p_k$.

(b) Consider an alternative algorithm that first computes $\alpha_i = (N/p_i^{e_i})\alpha$ for $1 \le i \le r$, and then determines the least $d_i \ge 0$ for which $p_i^{d_i}\alpha_i = 0$ by computing the sequence

$$\alpha_i,\ p_i\alpha_i,\ p_i^2\alpha_i,\ \ldots,\ p_i^{d_i}\alpha_i = 0,$$

where each term is obtained from the previous via a scalar multiplication by p_i. Show that the order of α is $\prod_i p_i^{d_i}$. Analyze the time complexity of this algorithm in the worst case, and give separate asymptotic bounds for inputs of the form 2^k and $p_1 \cdots p_k$.

(c) Consider a third algorithm that uses a recursive divide-and-conquer strategy. In the base case $r = 1$, so $N = p^k$ is a prime power and it computes the sequence $\alpha, p\alpha, p^2\alpha, \ldots, p^d\alpha = 0$ as above and returns p^d. For $r > 1$ it sets $s = \lfloor r/2 \rfloor$ and puts $N = N_1 N_2$ with $N_1 = p_1^{e_1} \cdots p_s^{e_s}$ and $N_2 = p_{s+1}^{e_{s+1}} \cdots p_r^{e_r}$. It then recursively computes $m_1 = |N_1\alpha|$ and $m_2 = |N_2\alpha|$ and outputs $m_1 m_2$.

(i) Prove that this algorithm is correct.

(ii) Analyze the time complexity of this algorithm in the worst case, and give separate asymptotic bounds for inputs of the form 2^k and $p_1 \cdots p_k$.

Problem 5. A Las Vegas algorithm to compute $\#E(\mathbb{F}_p)$ (50 points)

Implement a Las Vegas algorithm to compute $\#E(\mathbb{F}_p)$, as described in class. Use Sage's built-in functions for generating random points on an elliptic curve, for adding points

on an elliptic curve, and for performing scalar multiplication, but write your own code for performing the baby-steps giant-steps search and the fast order computation. When implementing the search, you will want to use a python dictionary to store the baby steps; python will automatically create a hash table to facilitate fast lookups (alternatively you can do a sort and match yourself, just be sure to avoid a linear search).

In the code below, $\mathcal{H}(p) = [p+1-2\sqrt{p}, p+1+2\sqrt{p}]$ denotes the Hasse interval. The following algorithm to compute $\#E(\mathbb{F}_p)$ was given in class:

Input: An elliptic curve $E/\mathbb{F}_p$, where $p > 229$ is prime.
Output: The cardinality of $E(\mathbb{F}_p)$.

1. Find a random non-square element $d \in \mathbb{F}_p$ and use it to compute the equation of a quadratic twist E_1 of $E_0 = E$ over $\mathbb{F}_p$.
2. Set $N_0 = N_1 = 1$ and $i = 0$ (the index i is used to alternate between E_0 and E_1).
3. While neither N_0 nor N_1 has a unique multiple in $\mathcal{H}(p)$:
   (a) Pick a random point P on E_i.
   (b) Use a baby-steps giant-steps search to find a multiple M of $|P|$ in $\mathcal{H}(p)$.
   (c) Compute the prime factorization of M using Sage's `factor` function.
   (d) Compute $m = |P|$ using any of the fast order algorithms from Problem 4.
   (e) Set $N_i = \mathrm{lcm}(m, N_i)$ and set $i = 1 - i$.
4. If N_0 has a unique multiple M in $\mathcal{H}(p)$ then return M, otherwise return $2p+2-M$, where M is the unique multiple of N_1 in $\mathcal{H}(p)$.

(a) By modifying part (b) of step 3, give an alternative method for determining $m = |P|$ that does not require steps (c) and (d).

(b) Let E be the elliptic curve $y^2 = x^3 - 35x - 98$ over $\mathbb{F}_p$ with $p = 4657$. Run your algorithm on $E/\mathbb{F}_p$ and record the values of N_i, M, and m that are obtained as the algorithm progresses.

(c) For $k = 20, 40, 60, 80$, pick a random prime p in the interval $[2^{k-1}, 2^k]$ (using Sage's `random_prime` function with the `lbound` parameter). Record the time it takes for you program to compute $\#E(\mathbb{F}_p)$ for the elliptic curve $y^2 = x^3 + 314159x + 271828$ for each of these primes and list these timings in a table.

The timings will vary depending on your exact implementation and the machine you are running on, but you should be able to see an $O(p^{1/4})$ growth rate for large p; the $k = 20$ and $k = 40$ cases will be too quick too see this, but you should see the times go up by a factor of roughly $2^{20/4} = 32$ as you move from a 40-bit to a 60-bit and then an 80-bit prime. As ball park figures to shoot for, the cases $k = 20, 40$ should both take less than a second, the $k = 60$ case should take a few tens of seconds (under ten if your code is tight); the $k = 80$ case may take several minutes. If you are not seeing $O(p^{1/4})$ growth it likely means that you are inadvertently doing a linear search of the baby steps rather than a table lookup; use a python dictionary to store baby steps and make sure you access it correctly (use "`giant in babys`" not "`giant in babys.keys()`"; the latter will do a linear search). You can use the sage function `cputime()` to time specific sections of your code.

Problem 6. Schoof's algorithm (50 points)

In this problem you will analyze the complexity of Schoof's algorithm, as described in Lecture 9 (Algorithms 9.1 and 9.3) and implemented in this Sage worksheet. In your complexity bounds, use $\mathsf{M}(m)$ to denote the complexity of multiplying two m-bit integers. You may wish to recall that the complexity of multiplying polynomials in $\mathbb{F}_p[x]$ of degree d is $O(\mathsf{M}(d \log p))$, provided that $\log d = O(\log p)$ (in Schoof's algorithm, $\ell = O(\log p)$, so this certainly applies). Under the same assumption, the complexity of inverting a polynomial of degree $O(d)$ modulo a polynomial of degree d is $O(\mathsf{M}(d \log p) \log d)$.

(a) Analyze the time complexity of computing t_ℓ as described in Algorithm 9.3 of the lecture notes and implemented in the `trace_mod` function in the worksheet. Give separate bounds for each of the four non-trivial steps in Algorithm 9.3 as well as overall bounds for the entire algorithm. Express your bounds in terms of ℓ and $n = \log p$, using $\mathsf{M}(m)$ to denote the cost of multiplying two m-bit integers.

(b) Analyze the total time complexity of Schoof's algorithm, as described in Algorithm 9.1 of the lectures notes and implemented in the `Schoof` function of this Sage worksheet, as a function of $n = \log p$. Give your answer in three forms, first using $\mathsf{M}(m)$ to express the cost of multiplying m-bit integers, then after plugging in the naïve bound $\mathsf{M}(m) = O(m^2)$ or the Schönhage-Strassen bound for FFT-based multiplication $\mathsf{M}(m) = O(m \log m \log\log m)$.

(c) In your answer to part (a), you should have found that the time complexity bound for one particular step is strictly worse than any of the other steps of Algorithm 9.3. Explain how to modify Algorithm 9.3 so that this step no longer strictly dominates the asymptotic running time.

(d) Revise your time complexity estimates in part (b) to reflect part (c).

(e) Analyze the space complexity of Schoof's algorithm as a function of n, both before and after your optimization in part (c).

Problem 7. 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 it (1 = "mind-numbing," 10 = "mind-blowing"), and how difficult you found it (1 = "trivial," 10 = "brutal"). Estimate the amount of time you spent on each problem to the nearest half hour.

	Interest	Difficulty	Time Spent
Problem 1			
Problem 2			
Problem 3			
Problem 4			
Problem 5			
Problem 6			

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"), the pace (1="way too slow", 10="way too fast", 5="just right") and the novelty of the material (1="old hat", 10="all new").

Date	Lecture Topic	Material	Presentation	Pace	Novelty
3/5	Point counting				
3/7	Schoof's algorithm				

Please record any additional comments you have on the problem sets or lectures, in particular, ways in which they might be improved.