18.783 Elliptic Curves **Spring 2015**
Problem Set #6 **Due: 03/20/2015**

Description

These problems are related to the material covered in Lectures 11-12. As usual, the first person to spot each non-trivial typo/error will receive one point of extra credit.

Instructions: Solve Problem 1 and **one** of Problems 2-4.

I suggest reading through all the problems 2-4 before picking the one you will solve. In terms of the theory/computation trade-off, Problem 2 is about the second stage of ECM and involves mostly theory and very little coding (you are asked to optimize a function using Newton's method). Problem 3 asks you to construct an elliptic curve primality proof which you can write a program to do if you wish, but you might find it more expedient to just do this "by hand" with the assistance of Sage, in which case no real coding is involved; it does ask for a complexity analysis and the last part of the problem will require a bit of thought and perhaps some creativity. Problem 4 asks you to construct a special form of a primality proof for a large prime (over 1000 bits). This will involve some code — the algorithm is not complicated and should be easy to implement, but your code will need to be reasonably efficient; on the other hand this problem will probably have the shortest write up.

Problem 1. Subexponential bounds (20 points)

This short problem is meant to familiarize you with subexponential complexity bounds. You do not need to show your work, but be sure to think through your answers carefully, not all of them are immediately obvious. Recall that our subexponential complexity bounds have the form

$$L_N[\alpha, c] := \exp\Big(\big(c + o(1)\big)(\log N)^{\alpha}(\log\log N)^{1-\alpha}\Big),$$

where $0 \le \alpha \le 1$ and $c > 0$. The notation $o(1)$ denotes any function $\epsilon(N)$ whose absolute value converges to 0 as $N \to \infty$. Thus $L_N[\alpha, c]$ should really viewed as a set of functions. A function $f(N)$ belongs to the set $L_N[\alpha, c]$ if and only if

$$\lim_{N\to\infty} \frac{\log f(N)}{(\log N)^{\alpha}(\log\log N)^{1-\alpha}} = c.$$

To get a sense of how these bounds grow with N, and how to compare consider the following table, in which the $o(1)$ term is assumed to be 0 and $n = \log_2 N$.

n	n^5	$L_N[1/4, 1]$	$L_N[1/2, 1]$	$L_N[1/2, \sqrt{2}]$	$n^2 L_N[1/2, \sqrt{2}]$	$N^{1/4}$.
64	1.1×10^9	1.1×10^3	4.3×10^5	9.3×10^7	3.8×10^{11}	6.6×10^4
128	3.4×10^{10}	1.3×10^4	4.6×10^8	1.8×10^{12}	2.9×10^{16}	4.3×10^9
256	1.1×10^{12}	2.8×10^5	1.5×10^{13}	4.2×10^{18}	2.7×10^{23}	1.8×10^{19}
512	3.5×10^{13}	1.3×10^7	6.7×10^{19}	1.1×10^{28}	2.9×10^{33}	3.4×10^{38}
1024	1.1×10^{15}	1.6×10^9	4.4×10^{29}	8.4×10^{41}	8.8×10^{47}	1.2×10^{77}
2048	3.6×10^{16}	6.1×10^{11}	1.2×10^{44}	2.2×10^{62}	9.2×10^{68}	1.3×10^{154}

(a) Simplify the following expressions, in which $0 < \alpha, \beta < 1$ and $c, d > 0$, and $p(x)$ denotes a polynomial of degree $k > 0$. Interpret sums and products of complexity bounds (sets of functions) in the obvious way, e.g. $S + T$ is the set of all functions $s + t$ with $s \in S$ and $t \in T$. Write your answer in the form $L_N[\gamma, e]$, where γ and e may depend on α, β, c, d, k.

(i) $L_N[\alpha, c] + L_N[\beta, d]$

(ii) $L_N[\alpha, c]L_N[\beta, d]$

(iii) $L_N[\alpha, c]p(\log N)$

(iv) $p(L_N[\alpha, c])$

(v) $L_{p(N)}[\alpha, c]$

(vi) $L_{L_N[\alpha,c]}[\beta, d]$

(b) For each of the following pairs of complexity bounds $A(N)$ and $B(N)$ representing sets of functions A and B, indicate which of the following holds: (a) $A \subsetneq B$, (b) $B \subsetneq A$, (c) $A = B$, (d) $A \cap B = \emptyset$, or (e) none of the above. Assume $0 < \alpha, \beta < 1$.

(i) $L_N[\alpha, c]$ and $O(L_N[\alpha, c])$.

(ii) $L_N[\alpha, c]$ and $L_N[\beta, d]$ with $\alpha > \beta$.

(iii) $L_N[\alpha, c]$ and $L_N[\alpha, d]$ with $c > d$.

(iv) $L_N[\alpha, c]$ and $O\Big(\exp\big(c(\log N)^\alpha\big)\Big)$.

Problem 2. ECM second stage (80 points)

The elliptic curve factorization method (ECM) can be extended to incorporate a *second stage* that substantially improves its practical performance. In this problem you will analyze the benefit of this second stage, and, as a side benefit, derive a generic algorithm to compute the order of a group element using $o(\sqrt{N})$ group operations.

Given an integer N to be factored, a bound M on the largest prime divisor of N one hopes to find, and a smoothness bound $B_1 = L_M[1/2, 1/\sqrt{2}]$, ECM generates random elliptic curves $E/\mathbb{Q}$ with a known point P of infinite order and computes the scalar multiple $mP = (x_m : y_m : z_m)$, working with projective coordinates reduced modulo N. The integer $m = \prod \ell_i^{e_i}$ is a product of prime powers with $\ell_i^{e_i} \le (\sqrt{M} + 1)^2 \le \ell_i^{e_i+1}$, ranging over all primes $\ell_i \le B_1$. The goal is to find a curve for which $\gcd(z_m, N)$ is non-trivial (we actually check $\gcd(z_{m_i}, N)$ for the partial products $m_i = \prod \ell_i^{e_i}$ as we go).

But suppose that, as often happens, $\gcd(z_m, N) = 1$. Let us assume that N has a prime factor $p \le M$ at which E has good reduction, and let E_p denote the reduction of E modulo p. We know that $\#E_p(\mathbb{F}_p)$ is not B_1-smooth, meaning that it has a prime factor $q > B_1$, but suppose that there is just one such q. Then the reduction of the point $Q = mP$ must have order q as an element of $E_p(\mathbb{F}_p)$. Provided q is not too large, say, $q \le B_2$ for some bound $B_2 \approx B_1^2$, then we can try to "compute" the order of mP in $E_p(\mathbb{F}_p)$ using a baby-steps giant-steps search up to the bound B_2. This is not as simple as it sounds: we don't know p so we must work modulo N while checking for collisions modulo p, but there is an efficient algorithm for detecting collisions [3, §3]. The details of this algorithm do not concern us here, we simply want to consider the potential speedup we might gain from such a *second stage*.

If the prime factors of an integer n are all smaller than y, and all but one of them is smaller than z, then n is said to be *semismooth* with respect to y and z. The function $\psi(x, y, z)$ counts the number of such integers $n \le x$. We are interested in the quantity $\frac{1}{M}\psi(M, B_2, B_1)$. Under the heuristic assumption that the orders of random elliptic

curves over a finite field are about as likely to be semismooth as integers of similar size, this is the probability that our algorithm will be able to find an integer n for which $nP \equiv 0 \bmod q$, either in the first or second stage (we aren't guaranteed to succeed if this happens, we also need $nP \not\equiv 0 \bmod N$, but this is very likely to be true).

Let $B_1 = M^{1/u}$. We saw in class that, under our heuristic assumption, the expected running time of ECM with just a single stage is proportional to

$$M^{1/u}(\psi(M, M^{1/u})/M)^{-1}\mathsf{M}(\log N). \tag{1}$$

Using the Canfield-Erdő s-Pomerance bound $\psi(x, x^{1/u})/x = u^{-u+o(u)}$, we found that we should pick $u = \sqrt{2\log M/\log\log M}$ and obtained the bound $L[1/2,\sqrt{2}]\mathsf{M}(\log N)$. But this is a very rough approximation and we ignored several factors logarithmic in M along the way (these are hidden in the $o(1)$ term in the subexponential notation).

We can get a much more precise estimate by using the Dickman function $\rho(u)$ to approximate $\psi(x, x^{1/u})/x$. The Dickman function $\rho(u)$ is defined via the differential delay equation

$$\rho'(u) = -\rho(u-1)/u,$$

with $\rho(u) = 1$ for $0 \le u \le 1$. Asymptotically $\rho(u) = \psi(x, x^{1/u})/x + o(1)$, and in practice $\rho(u)$ is very close to $\frac{1}{x}\psi(x, x^{1/u})$ for x and u in the range we are interested in. Sage has a built-in function `dickman_rho(u)` that computes a good numerical approximation to $\rho(u)$. See [2, §1] if you want to know more about $\rho(u)$ and its relation to $\psi(x, y)$.

To minimize (1) it suffices to thus suffices to minimize

$$M^{1/u}/\rho(u). \tag{2}$$

(a) Using Newton's method, write a simple function in Sage that approximates (to at least 3 decimal places) the value of u that minimizes (2).

For the sake of simplicity, let us suppose that $B_2 = B_1^2 = M^{2/u}$ and that the second stage has a running time approximately equal to that of the first. Then the expected running time of ECM with a BSGS second stage is heuristically proportional to

$$2M^{1/u}(M/\psi(M, M^{2/u}, M^{1/u})) \cdot \mathsf{M}(\log N), \tag{3}$$

with the same constant of proportionality as in our single stage analysis. In fact, we should optimally spend asymptotically slightly *less* time on the second stage than the first; this would allow us to save the factor of 2 in (3). You will prove below that this can actually be achieved using $B_2 = B_1^2$ if we modify the baby-steps giant-steps search appropriately.

Analogous to $\rho(u)$, Bach and Peralta [1] define the semismooth probability function

$$G(a, b) = \lim_{x\to\infty} \frac{1}{x}\psi(x, x^b, x^a)$$

(note the order of x^a and x^b). The function $G(a, b)$ can be numerically approximated using the Dickman function in terms of the function $F(\alpha) = \rho(1/\alpha)$ as

$$G(\alpha, \beta) = F(\alpha) + \int_\alpha^\beta F\left(\frac{\alpha}{1-t}\right)\frac{dt}{t}.$$

By numerically approximating $G(a,b)$ we can determine a suitable choice of u to minimize the quantity

$$M^{1/u}/G(1/u, 2/u). \tag{4}$$

This calculation is a bit time consuming, so a table of optimal u values for $M = 2^k$ with $k = 10, 20, \ldots, 200$ has been prepared for you and can be found in this Sage worksheet, which also implements a function `G(a,b)` that approximates $G(\alpha, \beta)$ using $\rho(u)$.

(b) Use the algorithm you implemented in (a) to generate a similar table of optimal u values that minimize (2). Then, for $k = 20, 40, 60, \ldots, 200$ compute $M^{1/u_1}/\rho(u_1)$ and $M^{1/u_2}/G(1/u_2, 2/u_2)$, with $M = 2^k$ and u_1 chosen to minimize the first quantity and u_2 chosen to minimize the second. List these values and their ratio in a table.

The ratios express the speedup we might hope to gain by using a second stage. You should find that the speedup is clearly increasing with k, implying that it is asymptotically better than a constant factor. Nevertheless, the second stage does not improve the subexponential complexity bound, which ignores even polynomial factors of $\log M$.

(c) Prove that the heuristic expected running time of ECM with a second stage is still $L_M[1/2, \sqrt{2}]\mathsf{M}(\log N)$, the same as with just one stage. Based on the data in your table from part (b), estimate what the asymptotic speedup is as a function of $\log M$.

Let $Q = mP$ be the point obtained after an unsuccessful first stage. When using baby-steps giant-steps to implement the second stage we can take advantage of the fact that, for any prime divisor $p \le M$ of N, in the group $E(\mathbb{F}_p)$ the reduction of the point Q cannot have order divisible by any prime $p_i \le B_1$. Indeed, the second stage will succeed only in the case where Q has prime order $q \in (B_1, B_2]$ in $E(\mathbb{F}_p)$.

This means that our baby-steps giant-steps search only needs to check $O(B_2/\log B_2)$ distinct multiples of Q, those corresponding to prime values. In principle, this could potentially be achieved with just $\sqrt{B_2/\log B_2}$ group operations, but it is not obvious how to do this. At a minimum, we can certainly avoid checking multiples of small primes $2, 3, 5, \ldots, \ell$ whose product t is substantially less than $\sqrt{B_2}$, for the sake of concreteness, let's say $t \approx B_2^{1/4}$. We should then compute baby steps of the form iQ with $\gcd(i,t) = 1$ for all $1 \le i \le r$ for some multiple r of t, followed by giant steps of the form jrQ for $1 \le j \le s$, where $rs \ge B_2$.

(d) Explain how to choose r and s so that the number of baby steps and giant steps are approximately equal, and give a tight asymptotic bound on the total number of steps in terms of B_2. You may use the Prime Number Theorem and standard facts it implies, such as $\sum_{p \le x} \log p \sim x$ and $\sum_{p \le x} \frac{1}{p} = \log\log x + O(1)$.[1]

(e) Now forget about ECM. Using your answer to part (d), describe a generic algorithm to compute the order of an element $\alpha \in G$ given an integer $N > |\alpha|$ that uses $o(\sqrt{N})$ group operations (the order of α may be prime or composite).

(f) Modify the algorithm in part (e) to not require N as an input, so that it computes $|\alpha|$ using $o(\sqrt{|\alpha|})$ group operations and give an asymptotic bound on the number group operations it uses.

[1] The second fact doesn't require the Prime Number Theorem, it was proved earlier by Mertens.

(g) Computing $|\alpha|$ is equivalent to computing the discrete logarithm of the identity with respect to α. Explain why your algorithm does not contradict Shoup's $\Omega(\sqrt{p})$ generic lower bound for the discrete logarithm problem even when $|\alpha| = p$ is prime.

It is worth noting that you have just disproved what was once a standard assumption, namely, that the worst-case complexity of computing $|\alpha|$ is $\Omega(\sqrt{|\alpha|})$ group operations.

Problem 3. ECPP (80 points)

Let us define an *elliptic curve primality proof* (ECPP) for p as a sequence of *certificates* $C_1, C_2, \ldots, C_k$, where each certificate C_i is of the form $(p_i, A_i, B_i, x_i, y_i, p_{i+1})$ with $p_1 = p$ and $p_{k+1} < (\log p)^4$. In each certificate C_i, the primes p_i and p_{i+1} satisfy

$$(\sqrt[4]{p_i} + 1)^2 < p_{i+1} < (\sqrt{p_i} + 1)^2/2, \tag{5}$$

and $P_i = (x_i, y_i)$ is a point of order p_{i+1} on $E_i\colon y^2 = x^3 + A_i x + B_i$ over $\mathbb{F}_{p_i}$.

(a) Let p be the least prime greater than $2^{128} \cdot N + 3^{64}$, where N is the first four digits of your student ID (use the `next_prime` function in Sage to compute p). Construct a short elliptic curve primality proof for p; this means each prime p_{i+1} should be close to the lower bound in (5) (you should not need more than 6 or 7 certificates). Note: the Goldwasser-Kilian algorithm typically will **not** produce a proof this short, it will have p_{i+1} closer to the upper bound in (5), so you will need to do something slightly different.

(b) Give an algorithm for verifying an elliptic curve primality proof and analyze its complexity. Express your answer solely in terms of $n = \log p$ and assume the worst-case (so the proof might not be as short as the one you generated in (a)).

(c) Analyze the asymptotic complexity of constructing an elliptic curve primality proof using the Goldwasser-Kilian algorithm given in class, under the heuristic assumption that the orders of random elliptic curves over $\mathbb{F}_p$ have factorizations comparable to random integers in the interval $[p, 2p]$. Assume that trial division and the Miller-Rabin test are used for attempted factorizations. Use an $O(n^5 \log\log n)$ complexity bound for point-counting via Schoof's algorithm.

(d) Now suppose that you want to construct an elliptic curve primality proof that can always be verified in $O(n\mathsf{M}(n))$ time, where $n = \log p$. Under the heuristic assumption above, give a probabilistic algorithm for constructing such a proof whose expected running time is bounded by $L_p[\alpha, c]$, using the smallest value of α that you can (hint: you can make $\alpha < 1/2$). Your answer should include a high-level description of the algorithm and a (heuristically proven) bound on its complexity.

Problem 4. Pomerance proofs (80 points)

A *Pomerance proof* is a special form of an elliptic curve primality proof that involves just a single certificate (p, A, x_0, k) and uses a Montgomery curve $By^2 = x^3 + Ax^2 + x$ over $\mathbb{F}_p$ on which there is a point (x_0, y_0) of point of order $2^k > (\sqrt[4]{p} + 1)^2 \geq 2^{k-1}$. Note that neither the y-coordinate nor B is needed to verify the certificate (no matter what $x_0^3 + Ax_0^2 + x_0$ is, there exists a nonzero B and a y_0 that will work and the verifier does

not need to know what they are), but the verifier should check that $\gcd(A^2-4,p)=1$ to ensure that the curve is not singular.

Every prime p has a Pomerance proof, but for a general prime p no efficient algorithm is known for finding one. In this problem you will develop a very efficient algorithm to construct a Pomerance proof for primes of a special form.

Let us first convince ourselves that every sufficiently large prime has a Pomerance proof. To do this we note the following theorem, which we will prove later in the course.

Theorem 1. *Let p be a prime. For every integer N in the Hasse interval*

$$\mathcal{H}(p) = [p+1-2\sqrt{p},\ p+1+2\sqrt{p}]$$

there exists an elliptic curve $E/\mathbb{F}_p$ for which $E(\mathbb{F}_p)$ is a cyclic group of order N.

(a) Using the theorem above, prove that every prime $p > 31$ has a Pomerance proof.

Now let E be the elliptic curve $y^2 = x^3 + 8$ over $\mathbb{F}_p$.

(b) Using the formula $\#E(\mathbb{F}_p) = p+1+\sum_{x\in\mathbb{F}_p}\left(\frac{x^3+8}{p}\right)$, prove that for every odd prime $p \equiv 2 \bmod 3$ we have $\#E(\mathbb{F}_p) = p+1$.

(c) Prove that for any prime $p \equiv 11 \bmod 12$ the curve $E/\mathbb{F}_p$ can be put in Montgomery form $By^2 = x^3 + Ax^2 + x$. Give a deterministic algorithm that computes A and B in time $O(n\mathsf{M}(n))$, where $n = \log p$.

(d) Give a probabilistic algorithm to construct a Pomerance proof for primes of the form $p = 3\cdot 2^m c - 1$, where c is odd and $2^m > (\sqrt[4]{p}+1)^2$, and analyze its complexity. Be sure to address the fact that the algorithm you gave in part (c) assumes that p is prime, but now it must also handle composite values of p.

(e) Implement your algorithm and use it to construct a Pomerance proof for a prime of the form $p = 2^k \cdot 3^m - 1$ that is greater than 2^{1000}. Be sure to format you answer so that all of the digits in the certificate you construct fit on the page. To speed things up, you may wish to do some trial division by small primes to eliminate obviously composite values of p before attempting to construct a primality proof.

(f) As noted above, no efficient algorithm is known for constructing a Pomerance proof. On the other hand, there certainly *is* an algorithm; for example, one could simply enumerate all the possible certificates (clearly a finite set) and attempt to verify them. But you can certainly do better than this. Give the most efficient algorithm you can come up with for constructing a Pomerance proof for a given prime $p > 31$ and bound its complexity. Your algorithm need not be deterministic, and you should feel free to assume any heuristics you believe are reasonable.

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 hard you found the problem (1 = "trivial," 10 = "brutal"). Also 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			

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
3/12	Index calculus, factoring integers				
3/17	Elliptic curve primality proving				

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

References

[1] E. Bach and R. Peralta, *Asymptotic semismoothness probabilities*, Mathematics of Computation **65** (1998) 1701–1715.

[2] A. Granville, *Smooth numbers, computational number theory and beyond*, in Algorithmic Number Theory: Lattices, Number Fields, Curves and Cryptography (MSRI Workshop), MSRI Publications **44** (2008), 267–324.

[3] P. Zimmermann and B. Dodson, *20 years of ECM*, Algorithmic Number Theory 7th International Symposium (ANTS VII), LNCS 4076 (2006), 525–542.