18.783 Elliptic Curves **Spring 2017**
Problem Set #5 **Due: 03/19/2015**

Description

These problems are related to the material covered in Lectures 10-11.

Instructions: Either pick **two** of Problems 1-3 to solve, or solve **both** Problems 4 and 5. Then complete Problem 6, 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 `SurnamePset5.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. Rubik's pie (50 points)

The baby-steps giant-steps method and Pollard rho algorithm are often referred to as *birthday paradox* algorithms (or $\sqrt{N}$*-algorithms*). They both work by searching for collisions among N alternatives in a way that ensures (or at least makes it very likely) that a collision will be found after $O(\sqrt{N})$ steps, rather than the $O(N)$ steps required by a brute force linear search. Another class of algorithm that falls into this category are *bidirectional search* (or *meet-in-the-middle*) algorithms. In this problem you will use a bidirectional search algorithm to construct an optimal solver for the Rubik's pie puzzle that you received in class.

The Rubik's pie consists of 18 pieces: 8 corner pieces, 8 edge pieces, and 2 center pieces. Every piece of the puzzle can be uniquely identified by specifying its shape (corner, edge, center), and the colors on it; for example, there is exactly one white-blue corner piece. The puzzle has 6 faces. The two circular faces, which we will call the *front* and *back* faces, each contain 9 pieces: 4 corners, 4 edges, and a center piece. The four side faces, which we will call the *up* ,*down*, *right*, and *left* faces, each contain 6 pieces: 4 corners and 2 edges. Note that each side face intersects each circular face in 3 pieces (2 corners and an edge), and adjacent side faces intersect in 2 corner pieces.

There are seven permissible *moves*, all of which are performed while looking directly at the front face (this determines the meaning of "clockwise" – you are assumed to be looking at the front face while turning the back face).

1. A clockwise quarter-turn of the back face (b), which moves the back pieces on the right face to the down face.

2. A counter-clockwise quarter-turn of the back face (f), which moves the back pieces on the right face to the up face.

3. A half-turn of the back face (B).

4. Half-turns of any of the four side faces (u, d, r, l).

All moves keep the center front piece in a fixed orientation and each has a unique inverse. We do not include rotations of the front face; up to orientation, these are equivalent to a

rotation of the back face (the counter-clockwise quarter-turn of the back face is labeled f because it is equivalent to a clockwise quarter-turn of the front face).

We define a *solved* puzzle to be one in which the front face is white and the colors of the stickers running clockwise along the side of the front layer starting from the blue corner are blue, blue, orange, orange, red, red, green, green, and the colors of the stickers along the side of the back layer match those in the front. Figure 1 shows a solved puzzle.[1]

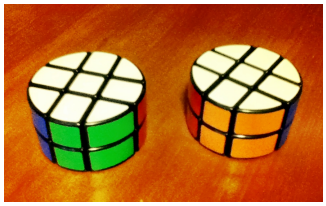

Figure 1. Two views of a solved puzzle.

A *configuration* of the puzzle specifies the location of each edge and corner piece in a fixed orientation with the white center piece in front; for example, a particular configuration might have the yellow-red edge is in the down-back position. There are four possible configurations of a solved puzzle. We say that a solved puzzle is in the *standard configuration* if the white-blue corner is in the up-left-front position (which also means that the yellow-red edge is in the down-back position).

Below are four views of a puzzle that had the move sequence **`frBul`** applied to a solved puzzle in the standard configuration. The view on the far left has the same orientation as the standard configuration. You may wish to verify this with your puzzle.

Figure 2. Views of a solved puzzle in the standard configuration after applying **`frBul`**.

Your task is to implement an *optimal solver* (also known as "God's algorithm") for the Rubik's pie. This is an algorithm that, given any starting configuration, outputs a shortest sequence of moves that leads to a solved configuration (not necessarily the standard one).[2]

Consider the graph $G = (V, E)$ whose vertex set V consists of all possible configurations of the puzzle and whose edges (v_1, v_2) are labeled with one of the seven permissible moves $m \in \{u, d, r, l, b, B, f\}$, where applying the move m to configuration v_1 yields the configuration v_2. This is a bidirected graph in which each vertex has degree 7; the move labeling the edge (v_1, v_2) is the inverse of the move labeling the edge (v_2, v_1).

Given two vertices s and t in this graph, we wish to find a (not necessarily unique) shortest path from s to t. The edge labels on this path give us a sequence of moves

[1] Unfortunately the manufacturer did not deliver all of the puzzles in a solved state, according to our definition (the order of the colors may vary). You can deal with this either by taking your puzzle apart (rotate an side face 45 degrees and gently force a corner out) and reassembling it, or by using your solver to put the puzzle in a solved state before answering the questions that follow.

[2] Unlike a Rubik's cube, with the Rubik's Pie, every starting configuration can be solved; this means that if your puzzle falls apart it does not matter how you put it back together.

$w = m_1 m_2 \cdots m_k$ that will take the puzzle from configuration s to configuration t in k moves, where k is the distance from s to t. Reversing the path and inverting each move yields a sequence $w^{-1} = m_k^{-1} m_{k-1}^{-1} \cdots m_1^{-1}$ that takes the puzzle from t to s.

To find such a path you will use a *bidirectional search*. Let $N(s, r)$ denote the *r-neighborhood* of s, the set of vertices v whose distance from s is at most r. For each vertex in $v \in N(s, r)$ we include a path w from s to v of length r (it does not matter which path is chosen) that we store together with v, so we view $N(s, r)$ as a set of pairs (v, w) where the v's are all distinct, and we index this set by v (in Python this can be conveniently implemented using a dictionary).

The bidirectional search algorithm works by alternately expanding neighborhoods of s and t until they intersect. An outline of the algorithm is given below. We use ϵ to denote the empty path.

1. Set $N(s, 0) = \{(s, \epsilon)\}$ and $N(t, 0) = \{(t, \epsilon)\}$, and set $r_s = r_t = 0$.
2. Repeat until $N(s, r_s)$ and $N(t, r_t)$ contain a common vertex v:
   a. If $r_s = r_t$, compute $N(s, r_s + 1)$ by extending $N(s, r_s)$ and then increment r_s; Otherwise, compute $N(t, r_t + 1)$ by extending $N(t, r_t)$ and then increment r_t.
3. Output the path $w_1 w_2^{-1}$, where $(v, w_1) \in N(s, r_s)$ and $(v, w_2) \in N(t, r_t)$.

Note that when extending a neighborhood you may encounter the same vertex multiple times, but you should only keep one pair (v, w) for each v (alternatively you could keep them all and compute every shortest path from s to t). Once you have implemented and tested your algorithm, use it to answer the following questions:

(a) Find an optimal solution to the configuration obtained by applying the sequence

`blurdbrBrdflblfrBrBrBdrbub`

to the standard puzzle. Below are four views of a standard puzzle after applying this move sequence. The leftmost has the same orientation as the standard configuration.

Figure 3. Standard puzzle after applying **`blurdbrBrdflblfrBrBrBdrbub`**.

Your solution should be a shortest move sequence that, when applied to the puzzle pictured on the left, yields a solved puzzle. Equivalently, it should be a shortest inverse of the move sequence above. Also record how long it took your algorithm to find a solution.

(b) Next, generate a "random" sequence of moves m using the Python code snippet

```
m = ''.join(['fbBudlr'[d] for d in (N^10).digits(7)])
```

where N is the first four digits of your student ID. Find a shortest inverse to m.

(c) By linearly extrapolating from the time it took your program to solve part 1, give a rough estimate (to within an order of magnitude) of the time it would take your program to find an optimal solution to the puzzle in Figure 3 if you had instead used a breadth-first search rather than a bidirectional search (i.e. just expand a neighborhood of s until it contains t). Assume that memory is not a limiting factor.

Problem 2. A Las Vegas algorithm to compute $E(\mathbb{F}_p)$. (50 points)

Let $E/\mathbb{F}_p$ be an elliptic curve over a finite field $\mathbb{F}_p$ of prime order p. In this problem you will use the extended discrete logarithm to design (but need not implement) a Las Vegas algorithm to determine the structure of $E(\mathbb{F}_p)$ as a sum of two cyclic groups

$$E(\mathbb{F}_p) \simeq \mathbb{Z}/N_1\mathbb{Z} \oplus \mathbb{Z}/N_2\mathbb{Z},$$

with $N_1|N_2$. We will assume that the group order N has already been computed, either by Schoof's algorithm or by the Las Vegas algorithm from Problem Set 3.

Our strategy is to determine the structure of the ℓ-Sylow subgroups of $E(\mathbb{F}_p)$ for each prime ℓ dividing N. Recall that an ℓ-Sylow subgroup is a maximal ℓ-group (a group in which the order of every element is a power of ℓ), and in an abelian group, there is a just one ℓ-Sylow subgroup and it contains every element whose order is a power of ℓ. If ℓ divides N but ℓ^2 does not, then the ℓ-Sylow subgroup is obviously isomorphic to $\mathbb{Z}/\ell\mathbb{Z}$, so we only need to consider primes whose square divides N. Furthermore, even if ℓ^2 does divide N, unless ℓ divides $p-1$, the ℓ-Sylow subgroup will still be cyclic:

(a) Prove that if the ℓ-Sylow subgroup of $E(\mathbb{F}_p)$ is not cyclic then $p \equiv 1 \bmod \ell$.

This yields the following high-level algorithm to compute N_1 and N_2, given N.

1. Compute the prime factorization N.
2. Set $N_1 = 1$ and $N_2 = 1$, and for each maximal prime power ℓ^e dividing N:
   (a) If $e = 1$ or ℓ does not divide $p-1$, then set $N_2 = \ell^e N_2$ and continue.
   (b) Otherwise, compute the structure $\mathbb{Z}/\ell^{e_1}\mathbb{Z} \oplus \mathbb{Z}/\ell^{e_2}\mathbb{Z}$ of the ℓ-Sylow subgroup of $E(\mathbb{F}_p)$ as described below, with $e_1 \le e_2$, and set $N_1 = \ell^{e_1} N_2$ and $N_2 = \ell^{e_2} N_2$.
3. Output N_1 and N_2

All we need now is an algorithm to compute the ℓ-Sylow subgroup G_ℓ of $E(\mathbb{F}_p)$, given the orders ℓ^e and N of G_ℓ and $E(\mathbb{F}_p)$, respectively. Our strategy is to first pick two random points $P_1, P_2 \in G_\ell$, by generating random points in $E(\mathbb{F}_p)$ and multiplying them by N/ℓ^e. We hope that these points generate G_ℓ. Next, we reduce them to what we hope is a *basis* for G_ℓ, that is, points Q_1 and Q_2 such that $G_\ell \simeq \langle Q_1\rangle \oplus \langle Q_2\rangle$. We then have $G_\ell \simeq \mathbb{Z}/\ell^{e_1}\mathbb{Z} \oplus \mathbb{Z}/\ell^{e_2}\mathbb{Z}$ where $\ell^{e_1} = |Q_1|$, $\ell^{e_2} = |Q_2|$. Note that we can quickly compute the order of any element of G_ℓ, since it must be a power of ℓ. Provided that we know the points Q_1 and Q_2 are *independent* (meaning that $\langle Q_1, Q_2\rangle \simeq \langle Q_1\rangle \oplus \langle Q_2\rangle$), in order to verify that we actually have computed a basis for G_ℓ and not some proper subgroup, we just need to check that $e_1 + e_2 = e$. If this does not hold, we try again with two new random points P_1 and P_2; eventually we must succeed.

Your job is to flesh out this strategy and analyze the resulting algorithm. We first recall the definition of the extended discrete logarithm given in Lecture 9.

Definition. For elements α and β of a finite group G, the *extended discrete logarithm* of β with respect to α, denoted $\mathrm{DL}^*(\alpha,\beta)$, is the pair of positive integers (x,y) with $\alpha^x = \beta^y$, where y is minimal subject to $\beta^y \in \langle\alpha\rangle$, and $x = \log_\alpha \beta^y$; in additive notation, $x\alpha = y\beta$, with y minimal subject to $y\beta \in \langle\alpha\rangle$ and $x > 0$ minimal.

(b) Prove each of the following statements for a finite abelian ℓ-group G containing elements α and β.

(i) If G has ℓ-rank at most 2 and α and β are random elements uniformly distributed over the elements of G, then the probability that $G = \langle\alpha,\beta\rangle$ is at least $3/8$.

(ii) If $(x,y) = \mathrm{DL}^*(\alpha,\beta)$ then y is a power of ℓ.

(iii) For $(x,y) = \mathrm{DL}^*(\alpha,\beta)$ the following are equivalent:

- $x = |\alpha|$ and $y = |\beta|$;
- $\langle\alpha,\beta\rangle$ has order $|\alpha|\cdot|\beta|$.
- α and β are independent;

(iv) If $|\alpha| \geq |\beta|$ and $(x,y) = \mathrm{DL}^*(\alpha,\beta)$ then $y|x$ and $\gamma = \beta - (x/y)\alpha$ and α are independent.

The key fact is (iv), which tells us that we should order the P_i so that $|P_1| \leq |P_2|$ and then let $Q_1 = P_1 - (x/y)P_2$ and $Q_2 = P_2$, where $(x,y) = \mathrm{DL}^*(P_2,P_1)$. If we then compute $\ell^{e_1} = |Q_1|$ and $\ell^{e_2} = |Q_2|$, it follows from (iii) that $G_\ell = \langle Q_1, Q_2\rangle$ if and only if $e_1 + e_2 = e$. Fact (i) tells that we expect this to occur within less than 3 iterations, on average. By (ii), we can compute $(x,y) = \mathrm{DL}^*(P_2,P_1)$, by attempting to compute $x = \log_{P_2} \ell^i P_1$ for $i = 0,1,2,\ldots$ until we succeed, at which point we have $y = \ell^i$.[3]

To compute $\log_{P_2} \ell^i P_1$, we use the prime-power case of the Pohlig-Hellman algorithm described in Lecture 10 to reduce the problem to a discrete logarithm computation in a group of prime order ℓ for which we use the baby-steps giant-steps method.

(c) Prove that in a cyclic group of prime-power order $N = \ell^e$ the complexity of the Pohlig-Hellman algorithm is

$$O(e\log\ell\log e + e\sqrt{\ell})$$

group operations. Use this to bound the bit-complexity of computing $\log_{P_2} \ell^i P_1$ in the ℓ-Sylow subgroup of $E(\mathbb{F}_p)$ with order ℓ^e.

(d) Write down a high-level description (not a program) of an algorithm to compute the structure of the ℓ-Sylow subgroup G_ℓ of $E(\mathbb{F}_p)$ in the form $\mathbb{Z}/\ell^{e_1}\mathbb{Z} \oplus \mathbb{Z}/\ell^{e_2}\mathbb{Z}$, given $N = \#E(\mathbb{F}_p)$ and $\ell^e = \#G_\ell$, and analyze its expected time complexity as a function of ℓ, $n = \log p$, and e.

(e) Analyze the total expected time complexity of the algorithm to compute the structure of $E(\mathbb{F}_p)$ in the form $\mathbb{Z}/N_1\mathbb{Z} \oplus \mathbb{Z}/N_2\mathbb{Z}$, given $N = \#E(\mathbb{F}_p)$ (hint: first figure out what the worst case is, then analyze that). You can assume that we have a Las Vegas algorithm that factors N in subexponential time, meaning it is faster than N^ϵ for any $\epsilon > 0$.

[3] There are much better ways to do this (a binary search, for example), but using them won't improve the worst-case complexity of the overall algorithm.

Problem 3. Student's choice (50 points)

Solve one of the four (out of six) problems from Problem Set 4 that you did not turn in.

Problem 4. 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. 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 5. 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-Erdős 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 $\sum_{p \le x} \frac{1}{p} = \log\log x + O(1)$.[4]

(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.

(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 6. 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 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			
Problem 5			

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

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
3/13	The discrete logarithm problem				
3/15	Index calculus				

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.