Outline

An efficient algorithm for Euclidean gcd; an efficient algorithm for modular exponentiation. An efficient algorithm for finding inverses in Z/N^*.

- What does it mean for these algorithms to be efficient? Poly(log(N)); the number of bits required to express the number.

Analyze Euclidean Algorithm

Find $\gcd(a,b)$ when $a \geq b$

$a = bq + r_1 \qquad 0 \leq r_1 < b$

$b = r_1 q_1 + r_2 \qquad 0 \leq r_2 < r_1$

$r_1 = r_2 q_2 + r_3 \qquad 0 \leq r_3 < r_2$

$r_2 = r_3 q_3 + r_4 \qquad 0 \leq r_4 < r_3$

$\vdots$

$r_k = 0$

How many steps to reach $r_k = 0$

Observation: $\leq b$ steps of divisions

Running Time is $O(b) = O(2^{\log b})$ exponential w.r.t number of bits of b.

If $r_{i+1} \leq \frac{r_i}{2}$ for all i then running time is $O(\log b)$ linear w.r.t number of bits of b.

Is $r_{i+1} \leq r_i/2$? Not really.

If $r_{i+1} > \frac{r_i}{2}$,

$$r_i = q_{i+1} r_{i+1} + r_{i+2}$$

What could be the value of q_{i+1}?

Can q_{i+1} be 2?

NO. If $q_{i+1} = 2$ then $r_i = 2\underline{r_{i+1}} + r_{i+2}$

$> r_i + r_{i+2}$

not possible.

So $q_{i+1} = 1$

$r_i = r_{i+1} + r_{i+2}$

$\underline{r_{i+2}} = r_i - r_{i+1} < \underline{\frac{r_i}{2}}$.

At most two steps is required to reduce the value r_i to half.

In general, we can prove that

$$r_{i+2} < \frac{r_i}{2} \quad \text{for all values of } i$$

$$r_3 < \frac{r_1}{2} < \frac{b}{2}$$

$$r_5 < \frac{r_3}{2} < \frac{b}{4}$$

$$r_7 < \frac{r_5}{2} < \frac{b}{8}$$

$$r_{2k+1} < \frac{b}{2^k}$$

let $k = \lceil \log_2 b \rceil$

Then $\frac{b}{2^k} < 1$

$$r_{2k+1} < \frac{b}{2^k} < 1$$

So, $r_{2k+1} = 0$

After $2k+1 = 2\lceil \log_2 b \rceil + 1$ steps,

the algorithm terminates.

Running Time of an Algorithm

Analyze an algorithm : running time in terms of # of bits of the input N.

If k bits are required to store input N and the running time of the algorithm is

$O(k^c)$, the algorithm is polynomial

$O(k)$, linear

$O(k^2)$, quadratic

$O(c^k)$ for constant c, exponential

# of bits required to store an input N is $\log_2 N$

Difficulty of a problem

"Easy": Solvable in polynomial time

"Hard": Solvable in exponential time using the most efficient method currently known.

Compute $GCD(a,b)$ $a \geq b$

Euclidean algorithm : $O(\log b)$

Easy problem

Compute $a^{-1} \bmod n$, $\gcd(a,n) = 1$

Extended euclidean algorithm : $O(\log n)$

Easy problem

Modular Exponentiation.

Given g, x, n, compute $g^x \bmod n$.

Trivial successive multiplication

$$g \xrightarrow{g} g^2 \xrightarrow{g} g^3 \xrightarrow{g} g^4, \ldots$$

$O(x)$ modular multiplication

Square and multiply

$O(\log x)$ modular multiplication

Easy

Square and Multiply Approach

Compute $3^{218} \bmod 1000$.

$218 = 2 + 2^3 + 2^4 + 2^6 + 2^7$

$3^{218} = 3^{2+2^3+2^4+2^6+2^7}$

$= 3^2 \cdot 3^{2^3} \cdot 3^{2^4} \cdot 3^{2^6} \cdot 3^{2^7}$

3^{2^i} for $i = 0$ to 7

We compute

i	0	1	2	3	4	5	6	7
$3^{2^i} \pmod{1000}$	3	9	81	561	721	841	281	961

square square

$O(\log x)$ modular multiplication

Square and Multiply Algorithm

Step 1. Compute the binary expansion of A as

$$A = A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r \quad \text{with } A_0, \ldots, A_r \in \{0, 1\},$$

where we may assume that $A_r = 1$.

Step 2. Compute the powers $g^{2^i} \pmod{N}$ for $0 \le i \le r$ by successive squaring,

$$\begin{aligned}
a_0 &\equiv g && \pmod{N} \\
a_1 &\equiv a_0^2 \equiv g^2 && \pmod{N} \\
a_2 &\equiv a_1^2 \equiv g^{2^2} && \pmod{N} \\
a_3 &\equiv a_2^2 \equiv g^{2^3} && \pmod{N} \\
\vdots \quad & \quad \vdots && \quad \vdots \\
a_r &\equiv a_{r-1}^2 \equiv g^{2^r} && \pmod{N}.
\end{aligned}$$

Each term is the square of the previous one, so this requires r multiplications.

Step 3. Compute $g^A \pmod{N}$ using the formula

$$\begin{aligned}
g^A &= g^{A_0 + A_1 \cdot 2 + A_2 \cdot 2^2 + A_3 \cdot 2^3 + \cdots + A_r \cdot 2^r} \\
&= g^{A_0} \cdot (g^2)^{A_1} \cdot (g^{2^2})^{A_2} \cdot (g^{2^3})^{A_3} \cdots (g^{2^r})^{A_r} \\
&\equiv a_0^{A_0} \cdot a_1^{A_1} \cdot a_2^{A_2} \cdot a_3^{A_3} \cdots a_r^{A_r} \pmod{N}. \qquad (1.4)
\end{aligned}$$

Note that the quantities $a_0, a_1, \ldots, a_r$ were computed in Step 2. Thus the product (1.4) can be computed by looking up the values of the a_i's whose exponent A_i is 1 and then multiplying them together. This requires at most another r multiplications.