Basic Algorithms in Number Theory

Francesco Pappalardi

Discrete Logs, Modular Square Roots & Euclidean Algorithm.

July 20th 2010

YESTERDAY'S PROBLEMS

1. MULTIPLICATION: for $x, y \in \mathbb{Z}$, find $x \cdot y$
2. EXPONENTIATION: for $x \in G$ (group) and $n \in \mathbb{N}$, find x^n (Complexity of operations in $\mathbb{Z}/m\mathbb{Z}$)
3. GCD: Given $a, b \in \mathbb{N}$ find $\gcd(a, b)$
4. PRIMALITY: Given $n \in \mathbb{N}$ odd, determine if it is prime (Legendre/Jacobi Symbols - Probabilistic Algorithms with probability of error)
5. QUADRATIC NONRESIDUES: given an odd prime p, find a quadratic non residue mod p.
6. POWER TEST: Given $n \in \mathbb{N}$ determine if $n = b^k (\exists k > 1)$
7. FACTORING: Given $n \in \mathbb{N}$, find a proper divisor of n

Contemporary Factoring

Contemporary records in factoring are obtained by the *Number Field Sieve* (NFS) which is an evolution of the *Quadratic Sieve* (QS). These (together with the ECM-factoring) have sub-exponential heuristic complexity.

More precisely let:

$$L_n[a;c] = \exp\left(((c+o(1)(\log n)^a(\log\log n)^{1-a})\right).$$

which is a quantity that oscillates between exponential $(a=1)$ and polynomial $(a=0)$ as a function of $\log n$. Then the complexities are respectively

ECM algorithm with heuristic complexity $L_n[1/2, 1]$ (Lenstra 1987)

NFS algorithm with heuristic complexity $L_n[1/3; 4/3^{3/2}]$ (Pollard)

QS algorithm with heuristic complexity $L_n[1/2, 1]$ (Dickson, Pomerance)

PROBLEM 8. DISCRETE LOGARITHMS:

Given x in a cyclic group $G = \langle g \rangle$, find n such that $x = g^n$.

- to make sense one has to specify how to make the operations in G
- If $G = (\mathbb{Z}/n\mathbb{Z}, +)$ then discrete logs are very easy.
- If $G = ((\mathbb{Z}/n\mathbb{Z})^*, \times)$ then we know that G is cyclic iff $n = 2, 4, p^\alpha, 2 \cdot p^\alpha$ where p is an odd prime. This is a famous theorem of Gauß.
- Already in $(\mathbb{Z}/p\mathbb{Z})^*$ there is no efficient algorithm to compute DL.
- It is already an interesting problem, given p, to compute a primitive root g modulo p (i.e. to determine $g \in (\mathbb{Z}/p\mathbb{Z})^*$ such that $\langle g \rangle = (\mathbb{Z}/p\mathbb{Z})^*$)
- The famous *Artin Conjecture for primitive roots* stated that any g (except $0, \pm 1$ and perfect squares) is a primitive root for a positive proportion of primes
- Known to be true assuming the GRH. It is also known that one out of $2, 3$ and 5 is a primitive root for infinitely many primes.

DISCRETE LOGARITHMS: continues

- Primordial public key cryptography is based on the difficulty of the Discrete Log problem *(Cryptography course from Kalyan Chakraborty)*
- Several algorithms to compute discrete logarithms are known. One for all is the **Shanks Baby Step Giant Step algorithm**.

Input: A group $G = \langle g \rangle$ and $a \in G$

Output: $k \in \mathbb{Z}/|G|\mathbb{Z}$ such that $a = g^k$

1. $M := \lceil \sqrt{|G|} \rceil$
2. For $j = 0, 1, 2, \ldots, M$.
   Compute g^j and store the pair (j, g^j) in a table
3. $A := g^{-M}$, $B := a$
5. For $i = 0, 1, 2, \ldots, M - 1$.
   -1- Check if B is the second component (g^j) of any pair in the table
   -2- If so, return $iM + j$ and halt.
   -3- If not $B = B \cdot A$

DISCRETE LOGARITHMS: continues

- The BSGS algorithm is a generic algorithm.
  It works for every finite cyclic group.
- It is based on the fact that any $x \in \mathbb{Z}/n\mathbb{Z}$ can be written as $x = j + im$ with $m = \lceil \sqrt{n} \rceil$, $0 \leq j < m$ and $0 \leq i < m - 1$
- It is not necessary to know the order of the group G in advance.
  The algorithm still works if an upper bound on the group order is known.
- Usually the BSGS algorithm is used for groups whose order is prime.
- The running time of the algorithm and the space complexity is $O(\sqrt{|G|})$, much better than the $O(|G|)$ running time of the naive brute force
- The algorithm was originally developed by Daniel Shanks.

DISCRETE LOGARITHMS: continues

In some groups Discrete logs are easy. For example if G is a cyclic group and $\#G = 2^m$ then we know that there are subgroups:

$$\langle 1\rangle = G_0 \subset G_1 \subset \cdots \subset G_m = G$$

such that G_i is cyclic and $\#G_i = 2^i$. Furthermore

$$G_i = \left\{ y \in G \text{ such that } y^{2^i} = 1 \right\}.$$

Hence if $G = \langle g\rangle$, for any $a \in G$, either $a^{2^{m-1}} = 1$ or $(ga)^{2^{m-1}} = 1$

From this property we deduce the algorithm:

```
Input:   A group G = ⟨g⟩, |G| = 2^m and a ∈ G
Output:  k ∈ Z/|G|Z such that a = g^k
1.   A := a,  K = 2^m
2.   For j = 1, 2, ..., m.
            If A^(2^(m-j)) ≠ 1,  A := g^(2^(j-1)) · A; K := K − 2^(j-1)
3 Output K
```

Discrete Logarithms: continues

- The above is a special case of the Pohlig-Hellman Algorithm which works when $|G|$ has only small prime divisors
- To avoid this situation one crucial requirement for a DL-resistent group in cryptography is that $\#G$ has a large prime divisor.
- If $p = 2^k + 1$ is a Fermat prime, then DL in $(\mathbb{Z}/p\mathbb{Z})^*$ are easy.
- Classical algorithm for factoring have often analogues for computing discrete logs. A very important one is the *index calculus algorithm*.

PROBLEM 9. Square Roots Modulo a prime:

Given an odd prime p and a quadratic residue a, find x s. t. $x^2 \equiv a \bmod p$

It can be solved efficiently if we are given a quadratic nonresidue $g \in (\mathbb{Z}/p\mathbb{Z})^*$

1. We write $p - 1 = 2^k \cdot q$ and we know that $(\mathbb{Z}/p\mathbb{Z})^*$ has a (cyclic) subgroup G with 2^k elements.
2. Note that $b = g^q$ is a generator of G (in fact if it was $b^{2^j} \equiv 1 \bmod p$ for $j < k$, then $g^{(p-1)/2} \equiv 1 \bmod p$) and that $a^q \in G$
3. Use the last algorithm to compute t such that $a^q = b^t$. Note that t is even since $a^{(p-1)/2} \equiv 1 \bmod p$.
4. Finally set $x = a^{(p-q)/2} b^{t/2}$ and observe that
$$x^2 = a^{(p-q)} b^t = a^p \equiv a \bmod p.$$

The above is not deterministic. However Schoof in 1985 discovered a polynomial time algorithm which is however not efficient.

PROBLEM 10. Modular Square Roots:

Given $n, a \in \mathbb{N}$, find x such that $x^2 \equiv a \bmod n$

If the factorization of n is known, then this problem (efficiently) can be solved in 3 steps:

1. For each prime divisor p of n find x_p such that $x_p^2 \equiv a \bmod p$
2. Use the Hensel's Lemma to lift x_p to y_p where $y_p^2 \equiv a \bmod p^{v_p(n)}$
3. Use the Chinese remainder Theorem to find $x \in \mathbb{Z}/n\mathbb{Z}$ such that $x \equiv y_p \bmod p^{v_p(n)}\ \forall p \mid n$.
4. Finally $x^2 \equiv a \bmod n$.

The last two tools (Hensel's Lemma and Chinese Remainder Theorem) will be covered either later or in Lecture 3.

Modular Square Roots: (continues)

On the opposite direction, suppose that for each $a \in \mathbb{Z}/n\mathbb{Z}$ we can solve $X^2 \equiv a \bmod n$. We want to use this hypothetical algorithm to find a factor of n.

Choose y at random in $\mathbb{Z}/n\mathbb{Z}$ and find x such that $x^2 \equiv y^2 \bmod n$.

Any common divisor of x and y also divides n. So we can assume that x and y are coprime.

If $p > 1$ is a prime factor of n, then p divides $(x+y)(x-y)$. In addition p divides exactly one of the factors $(x+y)$ or $(x-y)$.

If y is random, then any of the primes that divides $x^2 - y^2$ has 50% chances of $x+y$ of $x-y$.

Finally $\gcd(x-y, n)$ is a proper divisor of n.

If the above fails, then try again choosing a different random y. After k choices, the probability that n is not factored is $O(2^{-k})$.

Modular Square Roots: (continues)

The Factoring and Modular square roots are in practice equivalent in difficulty.

The difficulty of solving the analogue problem for e–th roots modulo n

i.e. Given e, C, n**, find** $x \in \mathbb{Z}/n\mathbb{Z}$ **such that** $x^e \equiv C \bmod n$

is the base of the security of RSA *(see K. Chakraborty course)*

PROBLEM 11. Diophantine Equations:

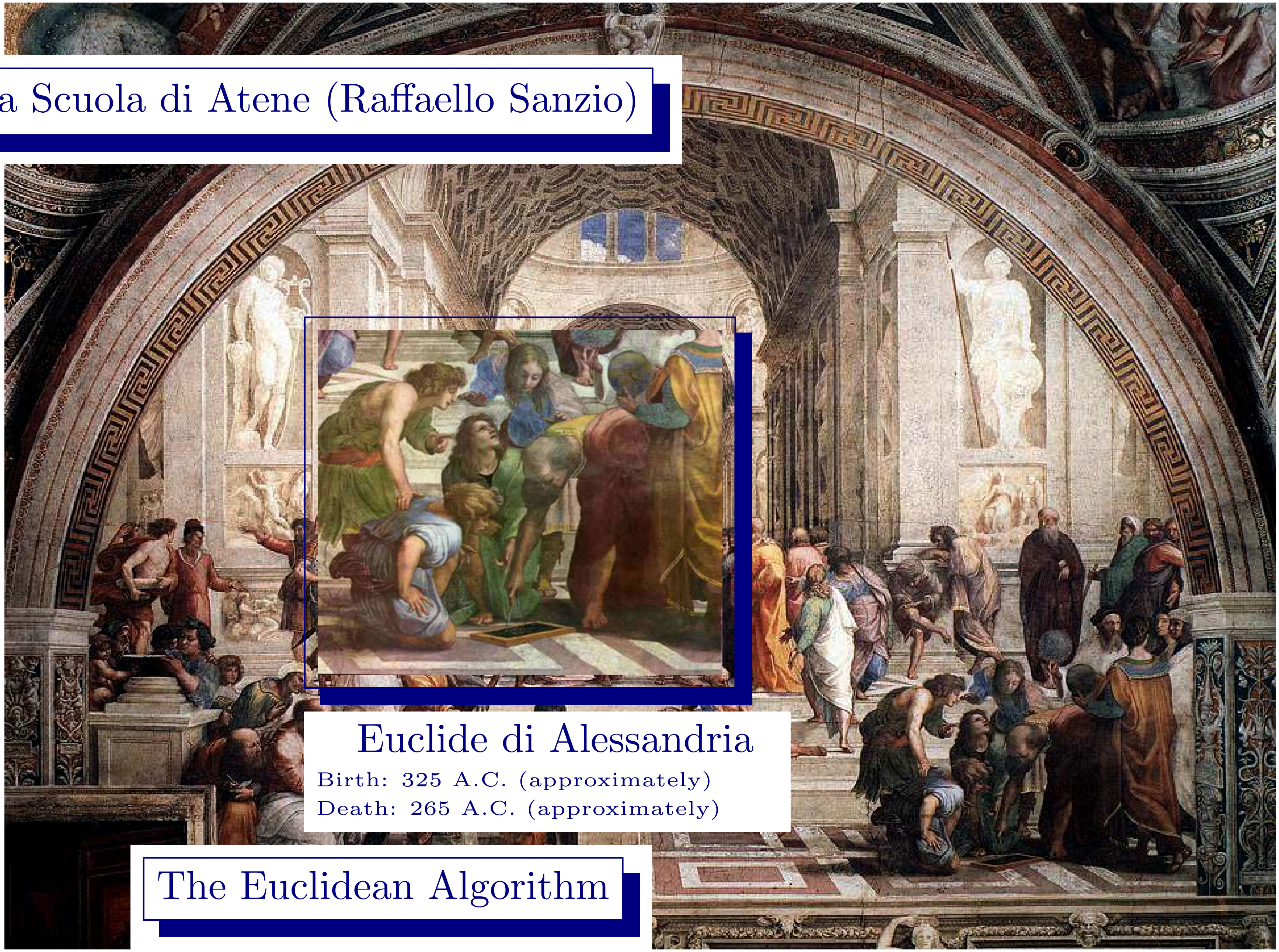
PROBLEM 11. Diophantine Equations: *Given* $f(X_1, \ldots, X_n) \in \mathbb{Z}[X_1, \ldots, X_n]$, *find* $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ *such that* $f(x) = 0$.

For a general f this is an undecidable problem (Matijasevic, Robinson, Davis, Putnam).

Although the problem might be easy for some specific f, there is no algorithm (efficient or otherwise) that takes f as input and always determines whether $f(x) = 0$ has a solution in integers.

Hilbert's tenth problem is the tenth on the list of Hilbert's problems of 1900.

> *Given a Diophantine equation with any number of unknown quantities and with rational integral numerical coefficients: To devise a process according to which it can be determined in a finite number of operations whether the equation is solvable in rational integers.*



La Scuola di Atene (Raffaello Sanzio)

Euclide di Alessandria

Birth: 325 A.C. (approximately)

Death: 265 A.C. (approximately)

The Euclidean Algorithm

Extended Euclidean Algorithm

Let $a, b \in \mathbb{N}$ (not both zero), we will also assume that $a \geq b$. The $\gcd(a, b)$ is greatest common divisor of a and b.

Clearly $\gcd(a, 0) = a$. If the factorization of a and b is known the it is easy to compute $\gcd(a, b)$. In fact

$$\gcd(a, b) = \prod_{p \text{ prime}} p^{\min\{v_p(a), v_p(b)\}}.$$

The p–adic valuation $v_p(n)$ of an integer n is

$$v_p(n) = \max\{\alpha \geq 0 \text{ such that } p^\alpha \text{ divides } n\}$$

so that the product above is indeed finite.

Furthermore

$$\gcd(a, b) = \min\{|xa + yb| > 0 \text{ such that } x, y \in \mathbb{Z}\}.$$

Extended Euclidean Algorithm

From the above identity it follows immediately that $\gcd(a, b)$ exists and that $\gcd(a, b) = xa + by$ for appropriate $x, y \in \mathbb{Z}$. In many applications it is crucial to compute x, y that realize the above identity and they are called the *Bezout coefficients*.

Theorem. *Given $a, b \in \mathbb{N}$, $0 < b \leq a$, then there exists x, y, z such that $z = \gcd(a, b)$ and $z = ax + by$. Furthermore they can be computed with an algorithm (EEA) with bit complexity $O(\log^2 a)$.*

Extended Euclidean Algorithm

It is based on successive divisions:

$$\begin{aligned}
a &= b \cdot q_0 &+ r_1 \\
b &= r_1 \cdot q_1 &+ r_2 \\
r_1 &= r_2 \cdot q_2 &+ r_3 \\
r_2 &= r_3 \cdot q_3 &+ r_4 \\
&\vdots &\vdots \\
r_{k-2} &= r_{k-1} \cdot q_{k-1} &+ r_k \\
r_{k-1} &= r_k \cdot q_k
\end{aligned}$$

Note that

$$\begin{aligned}
a = bq_0 + r_1 \geq bq_0 \geq (r_1 q_1 + r_2) q_0 &\geq r_1 q_1 q_0 \geq \cdots \\
\cdots &\geq r_k q_k q_{k-1} \cdots q_0 \geq q_k q_{k-1} \cdots q_0,
\end{aligned}$$

Extended Euclidean Algorithm

The $j+1$–th division requires time $O(\log r_j \log q_j)$ and using the fact that $\log r_i \leq \log b$, we obtain that the total time for running the EEA is

$$O(\log b \sum_{j=0}^{k} \log q_k) = O(\log b \log(q_0 \cdots q_k)) = O(\log b \log a).$$

A variation of the EEC with the same complexity but other advantages is

Binary gcd-algorithm (J. Stein – 1967)

(a,b)	$=$	`if`	$a<b$	`then`	(b,a)
		`if`	$b=0$	`then`	a
		`if`	$2 \mid a, 2 \mid b$	`then`	$2(a/2, b/2)$
		`if`	$2 \mid a, 2 \nmid b$	`then`	$(a/2, b)$
		`if`	$2 \nmid a, 2 \mid b$	`then`	$(a, b/2)$
				`else`	$((a-b)/2, b)$

Binary GCD Algorithm

$$
\begin{array}{rrcl}
1. & (1547, 560) & = & (1547, 280) \\
2. & (1547, 280) & = & (1547, 140) \\
3. & (1547, 140) & = & (1547, 70) \\
4. & (1547, 70) & = & (1547, 35) \\
5. & (1547, 35) & = & (756, 35) \\
6. & (756, 35) & = & (378, 35) \\
7. & (378, 35) & = & (189, 35) \\
8. & (189, 35) & = & (77, 35) \\
9. & (77, 35) & = & (35, 21) \\
10. & (35, 21) & = & (7, 21) \\
11. & (21, 7) & = & (7, 7) \\
12. & (7, 7) & = & (7, 0) = 7
\end{array}
$$

that can be written in matrix form as:

$$\begin{pmatrix} \alpha_0 & \alpha_1 \\ \beta_0 & \beta_1 \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ 1 & -q_0 \end{pmatrix}, \qquad \begin{pmatrix} \alpha_i \\ \beta_i \end{pmatrix} = \begin{pmatrix} \alpha_{i-2} & \alpha_{i-1} \\ \beta_{i-2} & \beta_{i-1} \end{pmatrix} \begin{pmatrix} 1 \\ -q_{i-1} \end{pmatrix}.$$

Example. $(1547, 560) = 7$

EEC:

$$\begin{aligned}
1547 &= 2 \cdot 560 + 427 \\
560 &= 1 \cdot 427 + 133 \\
427 &= 3 \cdot 133 + 28 \\
133 &= 4 \cdot 28 + 21 \\
28 &= 1 \cdot 21 + 7 && \leftarrow \text{GCD} \\
21 &= 3 \cdot 7
\end{aligned}$$

So that $(q_0, q_1, q_2, q_3, q_4, q_5) = (2, 1, 3, 4, 1, 3)$.

Example: $(1547, 560) = 7$ continues.

$$\begin{cases} \alpha_0 = 0, \alpha_1 = 1 \\ \alpha_i = \alpha_{i-2} - q_{i-1} \cdot \alpha_{i-1} \end{cases} \qquad \begin{cases} \beta_0 = 1, \beta_1 = -q_0 \\ \beta_i = \beta_{i-2} - q_{i-1} \cdot \beta_{i-1} \end{cases}$$

i	q_i	α_i	β_i
0	2	0	1
1	1	1	-2
2	3	-1	3
3	4	4	-11
4	1	-17	47
5	3	21	-58

In fact: $7 = 21 \cdot 1547 - 58 \cdot 560$.

Analysis of EEC on $a, b \in \mathbb{N}$

Assume that $a > b$. We want to show that the number of iterations (i.e. the number of divisions needed) during the EEA is (in the worst case) $O(\log a)$.

Fibonacci Numbers: $F_1 = F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$.

In the very special case when $a = F_n$ and $b = F_{n-1}$ then $r_1 = F_{n-2}$, $r_2 = F_{n-3}, \ldots r_{n-2} = F_1 = 1$ and $r_{n-1} = 0$.
From this we deduce that

1. $\gcd(F_n, F_{n-1}) = 1$
2. The number of divisions required by EEA is $O(n)$.

Proposition. *Let* $\theta = (\sqrt{5}+1)/2$. *Then*

$$F_n = \frac{\theta^n + (1-\theta)^n}{\sqrt{5}}.$$

Hence $\log F_n \sim n\theta$ *(so that* $n = O(\log F_n)$*).*
Proof. By induction. □

Analysis of EEC on $a, b \in \mathbb{N}$

Consequence. *If $a = F_n$ and $b = F_{n-1}$, then EEA requires $O(\log a)$ divisions!*

Proposition. *Assume that $a > b \geq 1$. If the EEA to compute $\gcd(a, b)$ requires k divisions, Then $a \geq F_{k+2}$ and $b \geq F_{k+1}$.*

PROOF. Let us first show that $r_{k-j} \geq F_{j+1}$. Indeed by induction or j:

- $r_k = \gcd(a, b) \geq 1 = F_1$, $r_{k-1} \geq 1 = F_2$
- $r_{k-j} = q_{k-(j-1)} r_{k-(j-1)} + r_{k-(j-2)} \geq F_j + F_{j-1} = F_{j+1}$.

Hence $b = r_0 \geq F_{k+1}$ and $a = q_0 b + r_1 \geq F_{k+1} + F_k = F_{k+2}$. □

Consequence. *The number of divisions $k = O(\log F_{k+2}) = O(\log a) \forall a, b$.*

A more careful analysis (the fact that the size of the integers decreases exponentially) of EEA shows that the bit complexity is $O(\log^2 a)$.

Geometric GCD algorithm (probably the original one)

- To compute (a, b) with $a \geq b > 0$, consider the rectangle with base a and height b.
- Remove from it a square of maximal area obtaining a rectangle of sizes a and $a - b$.
- Reorder them (if needed) and then repeat the process of removing a square.
- Keep on removing squares till it is left a square.
- The edge of the final square is the gcd.

Example. $(1547, 560) = (987, 560) = (427, 560) = (427, 133) = (294, 133) = (161, 133) = (28, 133) = (105, 28) = (77, 28) = (49, 28) = (21, 28) = (21, 7) = (14, 7) = (7, 7) = 7$

Extended GCD algorithm (EEA)

```
Input:   a, b ∈ ℕ,  a > b
Output:  x, y, z where z = gcd(a, b) and z = ax + by
1.   (X, Y, Z) = (1, 0, a)
2.   (x, y, z) = (0, 1, b)
       While Z > 0
           q := ⌊Z/z⌋
           (X, Y, Z) = (x, y, z)
           (x, y, z) = (X − qx, Y − qy, Z − qz)
       Output X, Y, Z
```

To show that it is correct it is enough to check that after one iteration $(X_1, Y_1, Z_1) = (1, -q_0, r_1)$ and after k iterations

$$(X_k, Y_k, Z_k) = (X_{k-2} - q_{k-1}X_{k-1}, Y_{k-2} - q_{k-2}Y_{k-2}, Z_{k-2} - q_{k-1}Z_{k-1}) = (\alpha_k, \beta_k, r_k).$$

The Euler φ–function

A first important application of EEA is to determine the inverses in $\mathbb{Z}/m\mathbb{Z}$

Theorem. *Let $a \in \mathbb{Z}$ and $m \in \mathbb{N}$ with $m > 1$. Then $a \mod m$ is invertible (i.e. $\exists b \in \mathbb{Z}/m\mathbb{Z}$ with $ab \equiv 1 \bmod m$) iff $\gcd(a, m) = 1$. Furthermore the "arithmetic inverse" b can be computed with time $O(\log m^2)$.*

Proof. If $\gcd(a, m) = 1$ then in time $O(\log m^2)$ we can compute $x, y \in \mathbb{Z}$ such that $1 = xa + ym$. Hence $b = x \mod m$ has the required property.
Conversely if $ab \equiv 1 \bmod m$, then $1 = ab + km$ for an appropriate $k \in \mathbb{Z}$. This implies that $\gcd(a, m)$ divides 1 and finally $\gcd(a, m) = 1 \quad \square$.

Corollary. *The set $U(\mathbb{Z}/m\mathbb{Z})$ of invertible elements of $\mathbb{Z}/m\mathbb{Z}$ coincides with*

$$\{a \in \mathbb{N} \ s.t. \ 1 \leq a \leq m, \gcd(a, m) = 1\}.$$

We define the Euler φ function as

$$\varphi(n) = \#U(\mathbb{Z}/m\mathbb{Z}) = \#\{a \in \mathbb{N} \text{ s.t. } 1 \leq a \leq m, \gcd(a, m) = 1\}.$$

The Euler φ–function continues

- $\varphi(1) = 1, \quad \varphi(p) = p - 1, \quad \varphi(p^\alpha) = p^{\alpha-1}(p-1)$
- $\varphi(mn) = \varphi(m)\varphi(n)$ if $\gcd(m,n) = 1$.
  This is a consequence of the Chinese Remainder Theorem
  (we shall meet it later).
- Hence if we can factor $n = p_1^{\alpha_1} \cdots p_r^{\alpha_r}$, then $\varphi(n)$ is easy to compute.
  it is enough to compute $n \prod_{p|n} 1 - 1/p$.
- If we know that $k = \varphi(n)$ and that $n = q \times p$ then we can factor n
  In fact $\{p, q\} = \left\{ \frac{\varphi(n)-n-1 \pm \sqrt{(\varphi(n)-n-1)^2 - 4n}}{2} \right\}$.
- An important **Theorem of Euler:**
  If $a \in U(\mathbb{Z}/m\mathbb{Z})$ *then* $a^{\varphi(n)} \equiv 1 \bmod n$.

The latter is crucial in RSA encryption and decryption *See. K. Chakrabory course*