Special Topics on Algorithms
Fall 2023
Number-theoretic problems: Exponentiation, Fibonacci numbers and GCD

Vangelis Markakis

Exponentiation

- **Exponentiation:**

I: Two positive integers a,n

Q: Find a^n

- Main operation in many cryptographic protocols (e.g., RSA)
- Very important to be able to compute this fast

Apply the definition

```
Exp1(a,n);
//a, n positive integers
p:= 1;

for i:=1 to n do p:=p*a;
return p;
```

Complexity: O(n)

Suppose a ≤ n (or that a is of the same magnitude as n)

$|I| = \Theta(\log n) \Rightarrow n = \Theta(2^{|I|})$, $O(n)$ is $O(2^{|I|}) = O(\exp(I))$ NOT POLYNOMIAL !

$N(I) = n$, $O(n)$ is $O(\text{poly}(N(I)))$ PSEUDO-POLYNOMIAL !

Is there a polynomial algorithm for EXP ?

Exponentiation

Repeated Squaring

k is O(logn)

Consider n in *binary*, n = $b_k b_{k-1} \ldots b_2 b_1 b_0$, e.g. 29 = 11101 => 29 = 16+8+4+1

$$a^{29} = a^{16} \cdot a^8 \cdot a^4 \cdot a^1$$

Idea: Compute sequentially the powers a, a^2, a^4, a^8,...

and keep track which ones are needed

```
Exp2(a,n)
p=1;
z=a;
for i=0 to k do
      { if b_i=1 then p=p·z;
        z=z^2 ; }
Return p;
```

Time: O(k) = **O(logn)** !

O(poly|I|) !

Exponentiation

Or equivalently:

```
Exp3(a,n)
p=1;
z=a;
while n>0 do {
    if n is odd then p=p·z;
    z=z²;
    n=⌊n/2⌋;    }
Return p;
```

Time: **O(logn)**

29	1	lsb
14	0	
7	1	
3	1	
1	1	msb
0		

Exponentiation – Even more...

- Or yet another implementation
- Based on the recurrence relation:

$$\alpha^n = \begin{cases} \left(\alpha^{\frac{n}{2}}\right)^2 & ,\ \text{n even} \\ \alpha\left(a^{\left\lfloor \frac{n}{2} \right\rfloor}\right)^2 & ,\ \text{n odd} \end{cases}$$

```
Exp4(a,n)
if n=0 then return 1;
z=Exp4(a,⌊n/2⌋);
if n is even then
return z²
else  return  a·z²
```

Complexity: T(n) = T(n/2) + O(1)

Solving the recurrence ⇒ O(logn)

Fibonacci Numbers

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89...

$$\text{Definition}: F_n = F_{n-1} + F_{n-2}, \quad F_0 = 0, \quad F_1 = 1$$

Problem **Fibonacci:**
I: a natural number n∈N
Q: Find F_n

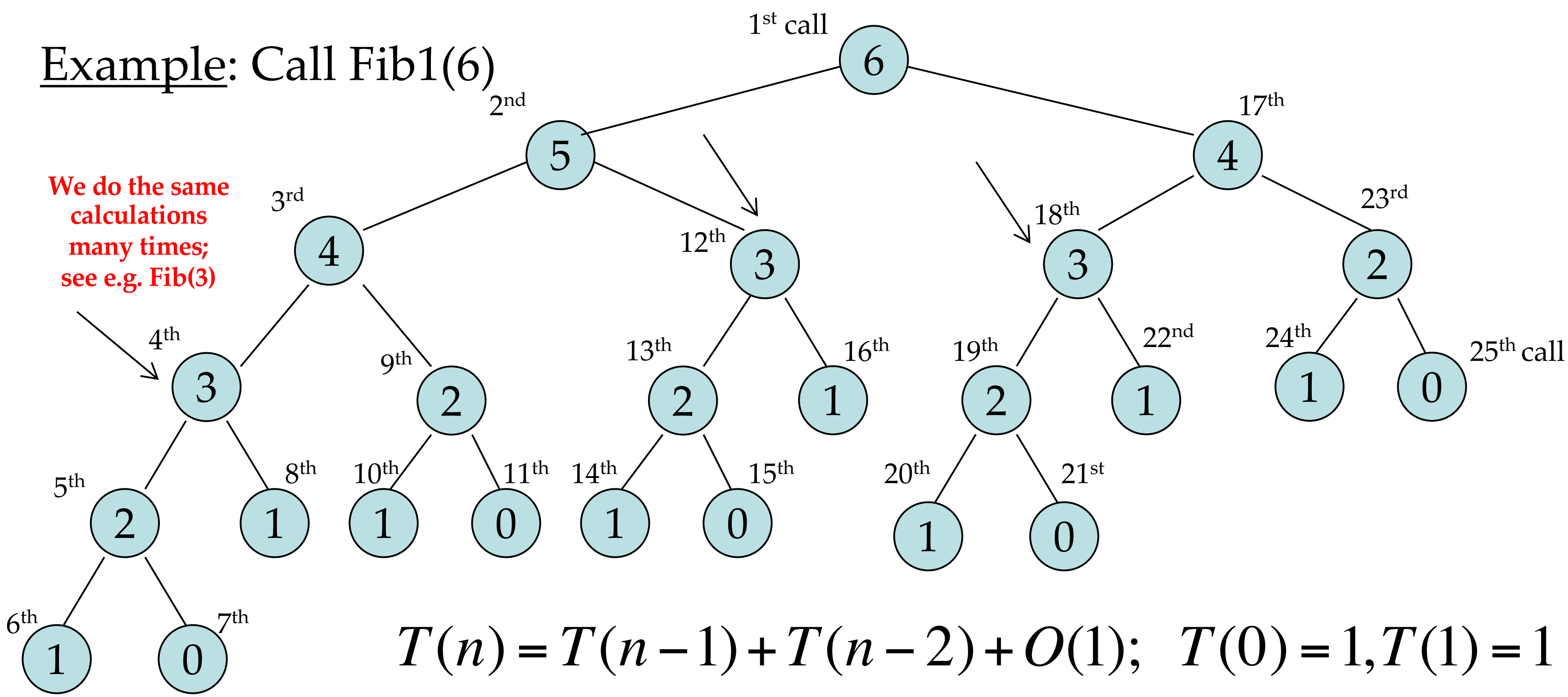
Direct Implementation of Recurrence

```
Fib1(n)
if n<2 then return n
else return Fib1(n-1) + Fib1(n-2)
```

Complexity of Fib1(n): T(0) = T(1) = 1,
T(n) = T(n-1) + T(n-2) + O(1)

Fibonacci Numbers

Example: Call Fib1(6)



$$T(n) = T(n-1) + T(n-2) + O(1); \quad T(0) = 1, T(1) = 1$$

$$T(n) = \Omega(2^{n/2}): \quad \text{Tree full to depth } n/2$$

$$\Omega(2^{n/2}) = \Omega\left(\left(\sqrt{2}\right)^n\right) = \Omega\left(1.41^n\right)$$

Fibonacci Numbers / Dynamic Programming

```
Fib2(n)
f[0]=0; f[1]=1;
  for i=2 to n do
       f[i] = f[i-1] + f[i-2];
Return f[n]
```

Time: $\Theta(n)$

Space: $\Theta(n)$

Big improvement over Fib 1

But: *NOT* O(poly(|I|)),

recall |I|= O(logn)

Save Space: No need for an array

```
Fib3(n);
if n<2 then return n
a=0; b=1;
for i=2 to n do
       { f=b+a;     a=b;
b=f; }
Return f;}
```

Time: still $\Theta(n)$, *NOT* O(poly(|I|))

Space: $\Theta(1)$ (we only use 3 variables)

Fibonacci Numbers / Closed Form Formula

- Relation to the golden ratio:

$$F_n = \frac{\phi^n}{\sqrt{5}} - \frac{\hat{\phi}^n}{\sqrt{5}}, \quad \text{where } \phi = \frac{1+\sqrt{5}}{2} = 1.618 \quad \text{(golden ratio)}$$

$$\text{and } \hat{\phi} = \frac{1-\sqrt{5}}{2} = -0.618$$

$$\text{(roots of } x^2 - x - 1 = 0, \quad \hat{\phi} = 1 - \phi = -\frac{1}{\phi}, \quad \phi^2 = \phi + 1)$$

- To simplify a bit, let ε be:

$$\varepsilon = \left|\frac{\hat{\varphi}^n}{\sqrt{5}}\right| < \frac{1}{2}, \forall n \geq 0 \qquad \left(\left.\begin{matrix} |\hat{\varphi}| < 1 \Rightarrow |\hat{\varphi}|^n < 1 \Rightarrow |\hat{\varphi}^n| < 1 \\ 1/\sqrt{5} < 1/2 \end{matrix}\right\} \Rightarrow \left|\frac{\hat{\varphi}^n}{\sqrt{5}}\right| < \frac{1}{2} \right)$$

Fibonacci Numbers / Closed Form Formula

$$F_n = \frac{\phi^n}{\sqrt{5}} - \frac{\hat{\phi}^n}{\sqrt{5}} \Rightarrow \left\{ \begin{array}{ll} F_n = \frac{\phi^n}{\sqrt{5}} + \varepsilon, & n \text{ odd} \\ F_n = \frac{\phi^n}{\sqrt{5}} - \varepsilon, & n \text{ even} \end{array} \right\} \Rightarrow F_n = round\left(\frac{\phi^n}{\sqrt{5}}\right)$$

or $$F_n = \left\lfloor \frac{\phi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

F_n is $\boldsymbol{\Theta(\varphi^n)}$

Consequences:

- Better lower bound for Fib1:
  - $T(n) = T(n-1) + T(n-2) + O(1) \geq F_n$
  - $T(n) = \boldsymbol{\Omega(\varphi^n)}$ that is $\boldsymbol{\Omega(1.6^n)}$
- We can calculate F_n by using the Exponentiation algorithm, **Exp2(φ,n)**

  Complexity: **O(logn)**

← ***But we don't like real (irrational) numbers!***

Fibonacci Numbers / Exponentiation

- We can work only with integer arithmetic
- Use the Exponentiation algorithm again, but to an array this time!

Matrix representation:

$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n = A^n, \quad \text{that is} \quad (-1)^n = F_{n+1}F_{n-1} - F_n^2$$

(Cassini's identity)

Prove this by induction

- Hence, just need to compute A^n
- Use the exponentiation algorithm
  - Exactly as before but replacing number multiplication by matrix multiplication (multiplications of 2 x 2 matrices)
- All intermediate results in the run of the algorithm are integer numbers
- Complexity: O(logn)

Number theory - Divisibility

- Divisibility
  - $d \mid a$: d divides a (d is a divisor of a)
  - Hence, $a = kd$ for some integer k
    - Every integer divides 0
    - If $a > 0$ and $d \mid a$, then $|d| \leq |a|$
  - Every integer a has as trivial divisors 1 and a itself
  - The non-trivial divisors of a are called factors
    - Factors of 20 : 2, 4, 5, and 10

Number theory - Divisibility

- Simple facts:
  - $a|b \Rightarrow a|bc$ for every integer c
  - $a|b \Rightarrow |a| \leq |b|$ or $b = 0$
  - $a|b \wedge b|c \Rightarrow a|c$
  - $a|b \wedge a|c \Rightarrow a|(b + c)$ and $a|(b - c)$
  - $a|b \wedge a|c \Rightarrow a|(bx + cy)$ for all integers x, y
  - $a|b \wedge b|a \Rightarrow |a| = |b|$

Number theory - Divisibility

- Division theorem:
  - **For every pair of integers *a, b* with b≠0, there are unique integers *q* and *r* such that**

$$a = qb + r \quad (0 \le r < |b|)$$

  - q = ***quotient*** = $\left\lfloor \frac{a}{b} \right\rfloor$
  - $r = a \bmod b$ = ***remainder***
- Proof:
  - Existence: either by induction or by looking into the smallest positive integer in the sequence

    ….., a-3b, a-2b, a-b, a, a+b, a+2b, a+3b,…
  - Uniqueness: by contradiction

Number theory - Divisibility

- Common divisors
  - If *d* l a, and *d* l b, then *d* is a ***common divisor of*** *a* and *b*
    - e.g., the divisors of 30 are 1, 2, 3, 5, 6, 10, 15, 30
    - divisors of 24: 1, 2, 3, 4, 6, 8, 12, 24
    - Common divisors of 24 and 30: 1, 2, 3, and 6
    - 1 is a common divisor for any 2 integers
  - Every common divisor of a and b is at most min (|a|, |b|)

Greatest Common Divisor (GCD)

- Greatest common divisor
  - gcd(a,b): The biggest among the common divisors (sometimes also written as (a, b)).
  - If $a \neq 0$, and $b \neq 0$, then gcd(a, b) is an integer between 1 and min(|a|, |b|)
  - Convention:
    - gcd(0, 0) = 0
  - Simple properties:
    - gcd(a,b) = gcd(b,a)
    - gcd(a,b) = gcd(|a|, |b|)
    - gcd(a,0) = |a|
    - gcd(a, ak) = |a| for every $k \in Z$

Greatest Common Divisor (GCD)

GCD

I: a, b ∈ ℕ

Q: Find gcd(a,b)

A simple algorithm:

```
GCD (a,b)
while a≠b do
      if a>b then a=a-b
      else b=b-a
return a
```

Greatest Common Divisor (GCD)

Correctness of GCD(a,b)

Claim 1: if $a > b$ then $\gcd(a,b) = \gcd(a-b,b)$

Proof:

Let $g = \gcd(a,b)$, and $g' = \gcd(a-b,b)$

Then, $a=gx$ and $b=gy$ for some $x, y \Rightarrow g \mid a-b \Rightarrow g' \geq g$

Also, $a-b = g'z$ and $b=g'w$ for some $z, w \Rightarrow a = g'(z+w) \Rightarrow g' \mid a \Rightarrow g \geq g'$. Hence $g = g'$

Complexity of GCD(a,b)

Worst case ($a=1$ or $b=1$): Complexity **O(w)**, with $w=\max\{a,b\}$

$|\mathbf{I}| = O(\log a + \log b) = \mathbf{O(\log w)}$

O(w) is not O(poly|I|)!

Euclid's Algorithm

Around 300 B.C., Euclid's elements, Book 7

```
EUCLID (a,b)
if b=0 then return a
else return EUCLID(b, a mod b)
```

Correctness of EUCLID (a,b)

Claim 2: if a > b then gcd(a,b) = gcd(b, a mod b)

Proof: Apply repeatedly Claim 1

Example

a	b
55	34
34	21
21	13
13	8
8	5
5	3
3	2
2	1
①	0

Euclid's Algorithm

More examples:

- a = 1742, b = 494
- $1742 = 3 \cdot 494 + 260$
- $494 = 1 \cdot 260 + 234$
- $260 = 1 \cdot 234 + 26$
- $234 = 9 \cdot 26$
- gcd(1742, 494) = 26

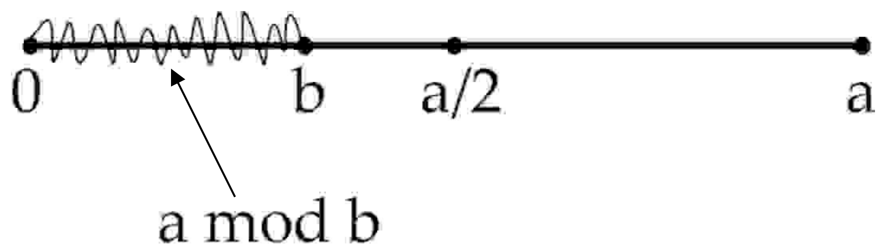
- a = 132, b = 35
- $132 = 3 \cdot 35 + 27$
- $35 = 1 \cdot 27 + 8$
- $27 = 3 \cdot 8 + 3$
- $8 = 2 \cdot 3 + 2$
- $3 = 1 \cdot 2 + 1$
- $2 = 2 \cdot 1$
- gcd(132, 35) = 1

"We might call it the granddaddy of all algorithms because it is the oldest nontrivial algorithm that has survived to the present day", (D. Knuth)

Euclid Algorithm

Complexity of EUCLID(a,b)

- One of a and b is at least halved at every call
- Both a and b are at least halved after any two recursive calls

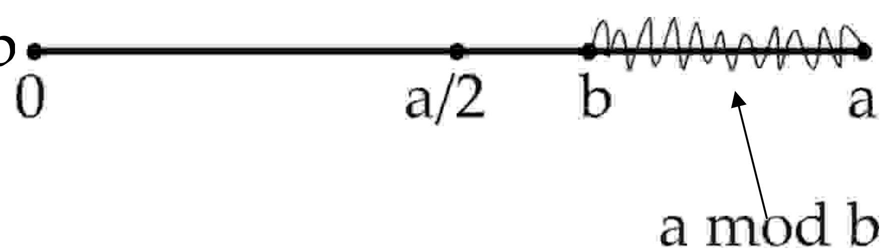


Claim 3: if a≥b then a mod b<a/2

Proof

Case 1: b ≤ a/2, then a mod b < b <a/2

Case 2: b > a/2 then a mod b = a-b



Time complexity:

At most k = loga + logb calls, that is **O(loga+logb)**

How many Euclid calls for Fibonacci Numbers?

Tight example on the complexity

	EUCLID(F_{k+1},F_k) (=EUCLID(F_k, F_{k+1} mod F_k))
k-1	EUCLID(F_k,F_{k-1})
k-2	EUCLID(F_{k-1},F_{k-2})
.	....
.	
..	....
.	
2	EUCLID(F_3,F_2) (=EUCLID(2,1))
1	EUCLID(1,0) =1

F_{k+1} mod F_k=
(F_k+F_{k-1}) mod F_k=
F_k mod F_k + F_{k-1} mod F_k=
0+F_{k-1} mod F_k = F_{k-1}

=k-1 recursive calls

Complexity: O(logF_{k+1}+logF_k)

EUCLID and Fibonacci numbers

If Euclid needs k calls, can we extract more information about a and b?

$b=0 \Rightarrow k=0$ calls
$a=b \Rightarrow k=1$ calls

Lemma: For $a>b>0$, if EUCLID(a,b) performs $k \geq 1$ recursive calls, then $a \geq F_{k+2}$ and $b \geq F_{k+1}$

Proof: By induction on k

Induction base: for k=1 call:

$b > 0 \Rightarrow b \geq 1 = F_2 \Rightarrow$ $b \geq F_2$

$a > b \Rightarrow a \geq 2 = F_3 \Rightarrow$ $a \geq F_3$

Inductive hypothesis: suppose true for k-1 calls:

$a \geq F_{k+1}, \; b \geq F_k$

EUCLID and Fibonacci numbers

Inductive step: suppose the algorithm needed k calls

- $k > 0 \Rightarrow b > 0 \Rightarrow$ EUCLID(a,b) calls EUCLID(b, a mod b)
- $b = a'$, a mod $b = b'$: EUCLID(a', b') performs k-1 calls
- By hypothesis

$a' \geq F_{k+1} \Rightarrow b \geq F_{k+1}$ and $b' \geq F_k \Rightarrow a \bmod b \geq F_k$

Also, $a > b$ and by the division theorem

$\Rightarrow a \geq b + (a \bmod b)$

$\Rightarrow a \geq b + F_k \geq F_{k+1} + F_k = F_{k+2} \Rightarrow a \geq F_{k+2}$

Corollary: Lame's Theorem

For $k \geq 1$, if $a > b > 0$, and $b < F_{k+1}$

EUCLID(a,b) performs at most k-1 recursive calls

EUCLID and Fibonacci numbers

$$\text{k calls} \Rightarrow$$

$$b \geq \frac{\phi^{k+1}}{\sqrt{5}} \Rightarrow$$

$$\phi^{k+1} \leq b\sqrt{5} \Rightarrow$$

$$k+1 \leq \log_{\phi}(b\sqrt{5}) = \log_{\phi} b + \log_{\phi} \sqrt{5} = \log_{\phi} b + 1.672 \Rightarrow$$

$$k \leq \log_{\phi} b + 0.672 \Rightarrow$$

$$k \text{ is } \mathrm{O}(\log \mathrm{b})$$

Extended Euclid's Algorithm

- Let a, b be "large" integers
- It is useful to understand further how gcd(a, b) looks like
- If someone claims that gcd(a, b) = d, how can we check this?
- It is not enough to check if $d|a$ and $d|b$!
  - (this would show that d is a divisor of a and b, but not necessarily the greatest)

Extended Euclid's Algorithm

Claim 3: If d | a, d | b and d = xa+yb, x,y ∈ **Z**, then gcd(a,b) = d

Proof:

$$d \mid a \text{ and } d \mid b \Rightarrow \left\{ \begin{array}{l} \gcd(a,b) \geq d \\ \gcd(a,b) \mid xa+yb = d \Rightarrow \gcd(a,b) \leq d \end{array} \right\}$$

Even further:
Claim 4: gcd(a, b) is the smallest positive integer from the set {ax +by : x, y є Z} of the linear combinations of *a* and *b*

Useful in certain applications (e.g., cryptosystems) to compute these coefficients

Extended Euclid's Algorithm –Correctness

<u>Example:</u> gcd(13,4) = 1, since 13*1 + 4*(-3) = 1

Existence of integer coefficients x, y for every pair of integers a, b, a>b:

Proof by strong induction on b:

Base: For b=0, we have that gcd(a,0) = a = a*x + 0*y, which holds for x=1 and every integer y

By induction hypothesis, assume that it holds for any integer <b: let gcd(b, a mod b) = bx' + (a mod b)y'

Induction Step:

$$\text{Then } \gcd(a,b) = \gcd(b, a \bmod b) = bx' + (a \bmod b)y'$$

$$= bx' + \left(a - \left\lfloor \frac{a}{b} \right\rfloor b\right) y' = ay' + b\left(x' - \left\lfloor \frac{a}{b} \right\rfloor y'\right)$$

Hence, x = y' and y = x'- a/b y'

Extended Euclid's Algorithm - Examples

One way to think at it is to run Euclid backwards:

- a = 1742, b = 494
- $1742 = 3 \cdot 494 + 260$
- $494 = 1 \cdot 260 + 234$
- $260 = 1 \cdot 234 + 26$
- $234 = 9 \cdot 26$
- (1742, 494) = 26

- $26 = 260 - 234$
  $= 260 - (494 - 260)$
  $= 2 \cdot 260 - 494$
  $= 2 \cdot (1742 - 3 \cdot 494) - 494$
  $= 2 \cdot 1742 - 7 \cdot 494$

- a = 132, b = 35
- $132 = 3 \cdot 35 + 27$
- $35 = 1 \cdot 27 + 8$
- $27 = 3 \cdot 8 + 3$
- $8 = 2 \cdot 3 + 2$
- $3 = 1 \cdot 2 + 1$
- $2 = 2 \cdot 1$
- (132, 35) = 1

- $1 = 3 - 2$
  $= 3 - (8 - 2 \cdot 3)$
  $= 3 \cdot 3 - 8$
  $= 3 \cdot (27 - 3 \cdot 8) - 8$
  $= 3 \cdot 27 - 10 \cdot 8$
  $= 3 \cdot 27 - 10 \cdot (35 - 27)$
  $= 13 \cdot 27 - 10 \cdot 35$
  $= 13 \cdot (132 - 3 \cdot 35) - 10 \cdot 35$
  $= 13 \cdot 132 - 49 \cdot 35$

Extended Euclid's Algorithm

```
ExtEUCLID(a,b)
Input: a,b ∈ℕ ; a ≥ b ≥ 0;
Output: x,y,d ∈ℤ : gcd(a,b)=d=ax+by
if b=0 then return (1,0,a)
else (x',y',d)=ExtEUCLID(b, a mod b);
```

return $(y', x' - \left\lfloor \frac{a}{b} \right\rfloor y', d)$

Correctness: follows by the existence proof
Complexity: O(logb) as EUCLID(a,b)

Extended Euclid's Algorithm

Example

$$y' \qquad x' - \left\lfloor \frac{a}{b} \right\rfloor y'$$

a	b	⌊a/b⌋	x	y	d
99	78	1	-11	14	3
78	21	3	3	-11	3
21	15	1	-2	3	3
15	6	2	1	-2	3
6	3	2	0	1	3
3	0	-	1	0	3

$ax + by = d$

$99(-11) + 78*14 =$

$-1089+1092=$ <u>3</u>