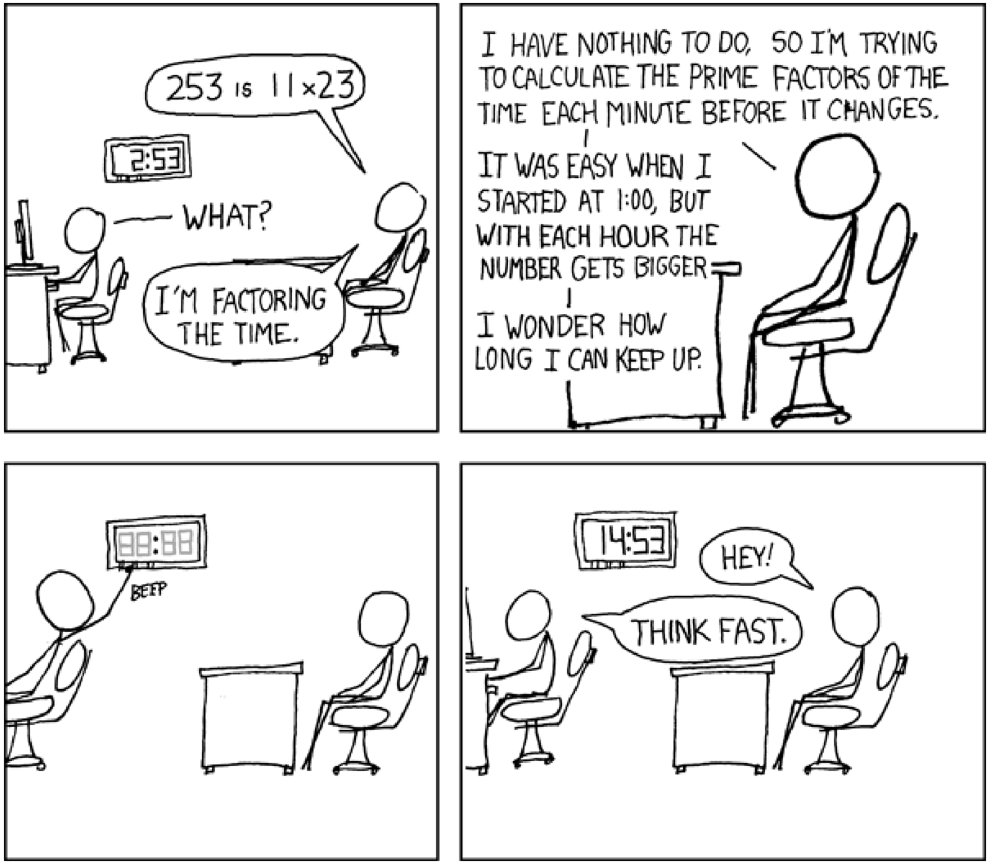
Spring 2015
Lecture 12: Primes, GCD, applications



$\rightarrow a \bmod b \equiv a \pmod b$

casting out 3s

$a = b \Rightarrow 0 \cdot a = 0 \cdot b$

Theorem: A positive integer n is divisible by 3 if and only if the sum of its decimal digits is divisible by 3.

$261 \qquad 2+6+1=9$

> 387

$10 \equiv 1 \pmod 3$

$3 \mid n \iff n \equiv 0 \pmod 3$

(desire)

$n = (d_k d_{k-1} \cdots d_1 d_0)_{10} \qquad \equiv d_k + d_{k-1} + \cdots + d_1 + d_0 \pmod 3$

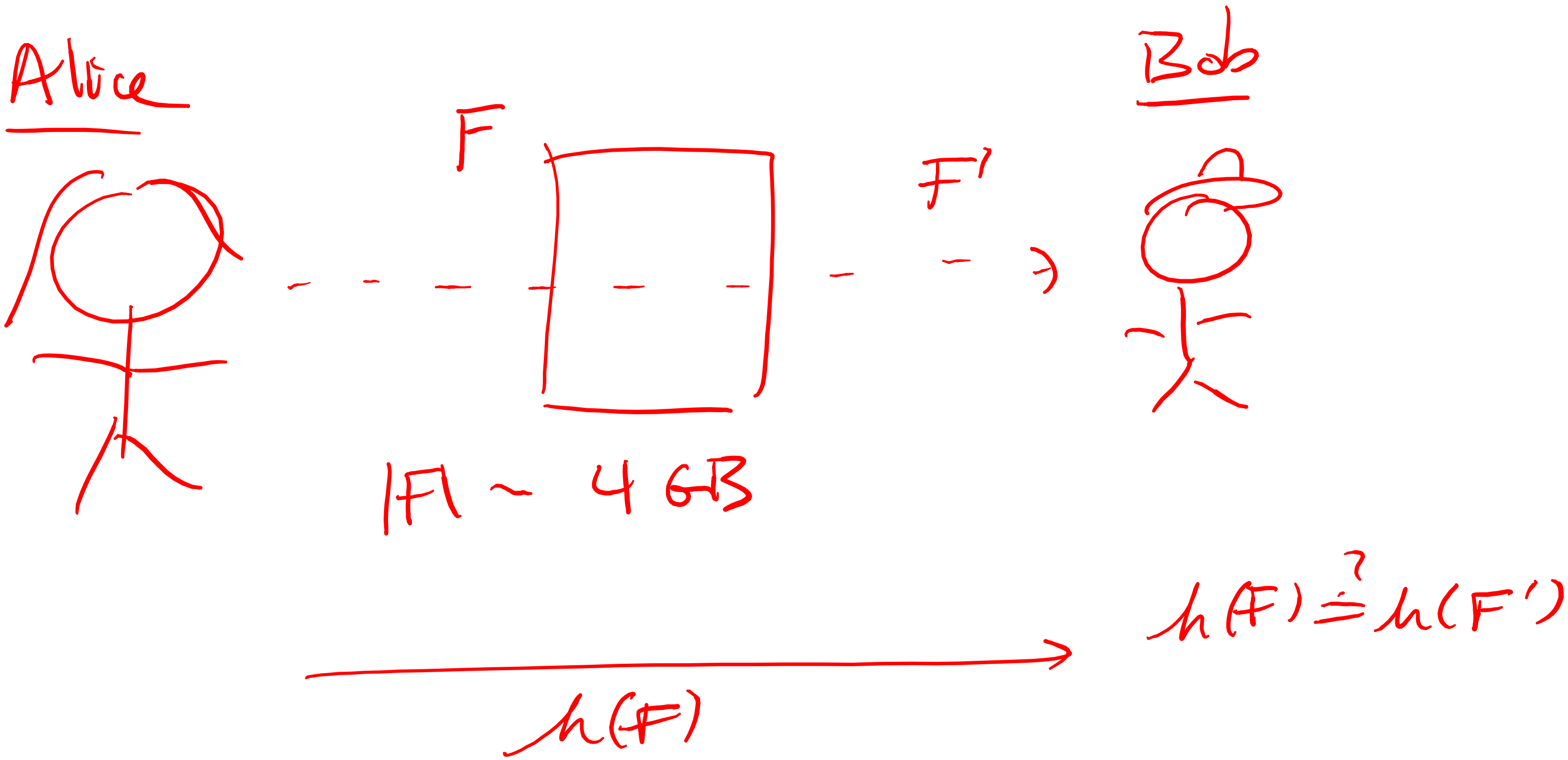
$n = d_k \cdot 10^k + d_{k-1} \cdot 10^{k-1} + \cdots + d_1 \cdot 10 + d_0$

$\equiv d_k (1)^k + d_{k-1} \cdot (1)^{k-1} + \cdots + d_1 \cdot (1) + d_0 \pmod 3$

$\equiv d_k + d_{k-1} + \cdots + d_1 + d_0 \pmod 3 \qquad \square.$

$a \equiv b \pmod m$

leaked game of thrones



Alice chooses a random prime p

and computes $\underbrace{F \bmod p}_{h(F)} \in \{0, 1, \ldots, p-1\}$

$|F| = 4GB$, hash size 64 bits

$P[h(F) = h(F') \text{ but } F \neq F'] \leq 0.000000006$.

$$h(F) = h(F')$$

$F \bmod p = F' \bmod p \iff F \equiv F' \pmod{p}$

$\iff p \mid F - F'.$

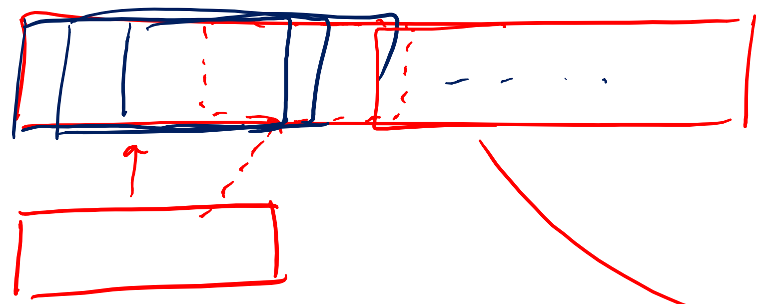
$\underline{p \mid 0 \quad \text{always true}}$

$$0 = p \cdot 0$$

$\underbrace{F - F'}_{D \neq 0}$ is n-bit number

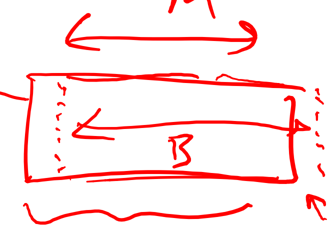
Upper bound # distinct prime factors of D? $\leq n$

pattern matching

$X =$  n-bit

$Y =$

m-bit



m

B

b

$X = 10110\ 10\ 010\ 110\ 101$

$Y = 010$

$h(A) = A \bmod p$

$\boxed{B = 2(A - A_0) + b}$

$B \bmod p = (2(A \bmod p - A_0) + b) \bmod p$

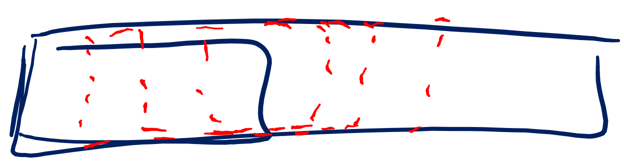
(64 bit)

$\approx m \cdot n$ comparisons

$F(Y) = Y \bmod p$

$\approx \log_2 m$ bits

$x^{(1)}$

$X =$ 

$F(x^{(1)}) = x^{(1)} \bmod p$

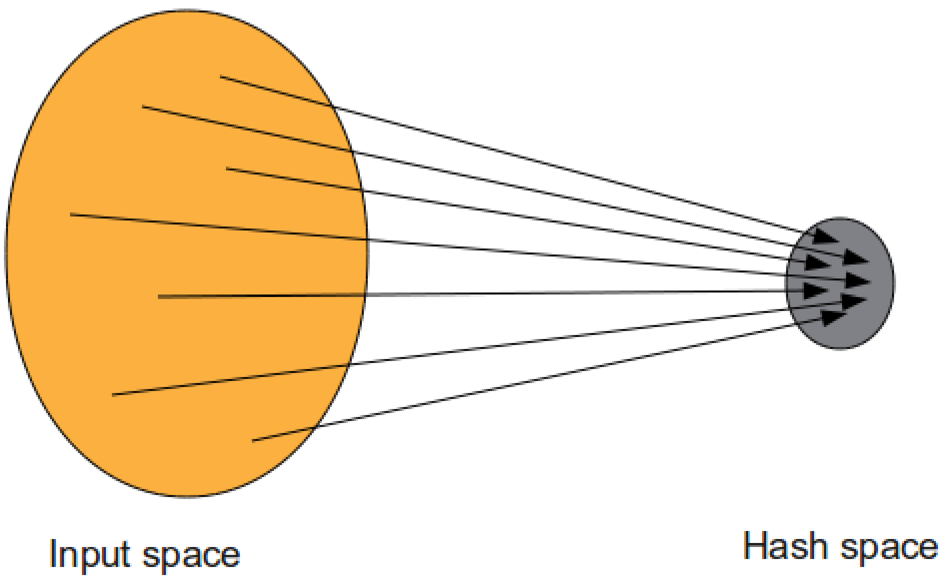
$\boxed{2^m \bmod p}$

basic applications of mod

- Hashing
- Pseudo random number generation
- Simple cipher

Scenario:

Map a small number of data values from a large domain $\{0, 1, \ldots, M-1\}$ into a small set of locations $\{0,1, \ldots, n-1\}$ so one can quickly check if some value is present.



Scenario:

Map a small number of data values from a large domain $\{0, 1, \ldots, M - 1\}$ into a small set of locations $\{0,1, \ldots, n - 1\}$ so one can quickly check if some value is present

- $\mathrm{hash}(x) = x \bmod p$ for p a prime close to n
  - or $\mathrm{hash}(x) = (ax + b) \bmod p$

- Depends on all of the bits of the data
  - helps avoid collisions due to similar values
  - need to manage them if they occur

pseudo-random number generation

Linear Congruential method:

$$x_{n+1} = (a\, x_n + c) \bmod m$$

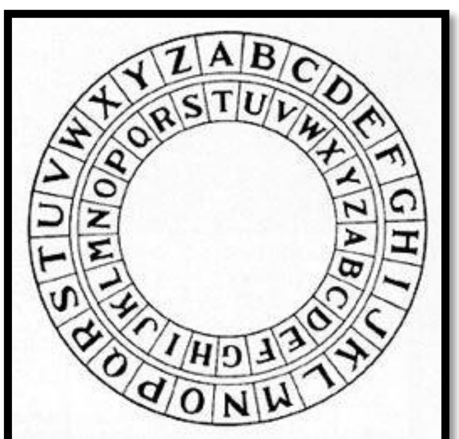
Choose random x_0, a, c, m and produce a long sequence of x_n's

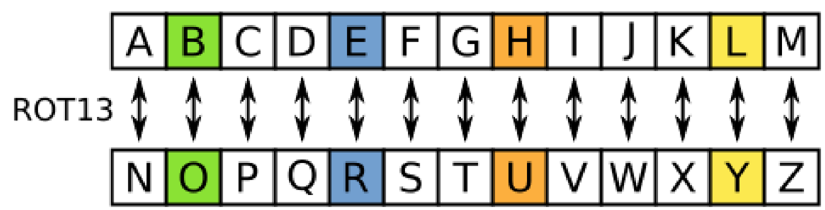
[good for some applications, really bad for many others]

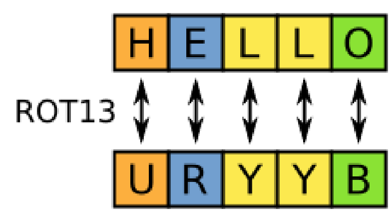
simple ciphers

- Caesar cipher, A = 1, B = 2, . . .
  – HELLO WORLD
- Shift cipher
  – $f(\mathrm{p}) = (\mathrm{p} + \mathrm{k}) \bmod 26$
  – $f^{-1}(\mathrm{p}) = (\mathrm{p} - \mathrm{k}) \bmod 26$
- More general
  – $f(\mathrm{p}) = (\mathrm{ap} + \mathrm{b}) \bmod 26$

```
/**//*/};)/**/main(/*//**/tang    ,gnat/**//*/,ABBA~,0-0(avnz;)0-0,tang,raeN
,ABBA(niam&&)))2-]--tang-[kri     - =raeN(&&0<)/*clerk*/,noon,raeN){(!tang&&
noon!=-1&&(gnat&2)&&((raeN&&(    getchar(noon+0)))||(1-raeN&&(trgpune(noon
)))))||tang&&znva(/*//**/tang   ,tang,tang/**|**//*/((||)))0(enupgrt=raeN
(&&tang!(||)))0(rahcteg=raeN(  &&1==tang((&&1-^)gnat=raeN(;;;)tang,gnat
,ABBA,0(avnz;)gnat:46+]552&)191+gnat([kri?0>]652%)191+gnat([kri=gnat
(&&)1-^gnat(&&)1& ABBA(!;)raeN,tang,gnat,ABBA(avnz&&0>ABBA{)raeN
,/**/);}znva(/*//**/tang,gnat,ABBA/**//*/(niam;}1-,78-,611-,321
-,321-,001-,64-,43-,801-,001-,301-,321-,511-,53-,54,44,34,24
,14,04,93,83,73,63,53,43,33,85,75,65,55,45,35,25,15,05,94,84
,74,64,0,0,0,0,0,0,/**/){ABBA='N'==65;(ABBA&&(gnat=trgpune
(0)))||(!ABBA&&(gnat=getchar(0-0)));(--tang&1)&&(gnat='n'<=
gnat&&gnat<='z'||'a'<=gnat&&gnat<='m'||'N'<=gnat&&gnat<='Z'
||'A'<=gnat&&gnat<='M'?(((gnat&/*//**/31/**//*/,21,11,01,9,8
,7,6,5,4,3,2,1,62,52,42,/**/)+12)%26)+(gnat&/*//**/32/**//*/,
22,12,02,91,81,71,61,51,41{=]652[kri};)/*pry*/)+65:gnat);main
(/*//**\**/tang^tang/**//*/,/*     */,~/*//*-*/tang,gnat,ABBA-
0/**//*/(niam&&ABBA||))))tang(      rahcteg&&1-1=<enrA(||))tang(
enupgrt&&1==enrA((&&)2&gnat(&&      )1-^tang(&&ABBA!(;)85- =tang
(&&)'a\'=gnat(&&)1-==gnat(&&)4       ==ABBA(&&tang!;))))0(enupgrt=
 gnat(&&)tang!((||)))0(rahcteg       =gnat(&&tang((&&ABBA;;)1-'A'=!
'Z'=tang(&&ABBA{)enrA/***/);gnat       ^-1&&znva(tang+1,gnat,1+gnat);
 main(ABBA&2/*//*\\**/,tang,gnat       ,ABBA/**//*/(avnz/**/);}/*//**/
```







modular exponentiation mod 7

x	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

a	a^1	a^2	a^3	a^4	a^5	a^6
1						
2						
3						
4						
5						
6						

$2^m \bmod p$

modular exponentiation mod 7

x	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

a	a^1	a^2	a^3	a^4	a^5	a^6
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4						
5						
6						

modular exponentiation mod 7

x	1	2	3	4	5	6
1	1	2	3	4	5	6
2	2	4	6	1	3	5
3	3	6	2	5	1	4
4	4	1	5	2	6	3
5	5	3	1	6	4	2
6	6	5	4	3	2	1

a	a^1	a^2	a^3	a^4	a^5	a^6
1	1	1	1	1	1	1
2	2	4	1	2	4	1
3	3	2	6	4	5	1
4	4	2	1	4	2	1
5	5	4	6	2	3	1
6	6	1	6	1	6	1

- Compute 78365^{81453}

$\log_2(81453)$

- Compute $\underbrace{78365}_{a}{}^{81453} \bmod \underbrace{104729}_{m}$

- Output is small
  - need to keep intermediate results small

$(((((a \cdot a) \bmod m) a) \bmod m) a) \bmod m$

81,453 times

repeated squaring – small and fast

Since $a \bmod m \equiv a \pmod{m}$ for any a

we have $a^2 \bmod m = (a \bmod m)^2 \bmod m$

and $a^4 \bmod m = (a^2 \bmod m)^2 \bmod m$

and $a^8 \bmod m = (a^4 \bmod m)^2 \bmod m$

and $a^{16} \bmod m = (a^8 \bmod m)^2 \bmod m$

and $a^{32} \bmod m = (a^{16} \bmod m)^2 \bmod m$

Can compute $a^k \bmod m$ for $k = 2^i$ in only i steps

fast exponentiaion

```
public static long FastModExp(long base, long exponent, long modulus) {
       long result = 1;
       base = base % modulus;

       while (exponent > 0) {
           if ((exponent % 2) == 1) {
               result = (result * base) % modulus;
               exponent -= 1;
           }
           /* Note that exponent is definitely divisible by 2 here. */
           exponent /= 2;
           base = (base * base) % modulus;
           /* The last iteration of the loop will always be exponent = 1 */
           /* so, result will always be correct. */
       }
       return result;
   }
```

$b^e \bmod m = (b^2)^{e/2} \bmod m$, when e is even)

$b^e \bmod m = (b*(b^{e-1} \bmod m) \bmod m)) \bmod m$

$78365^{81453} \bmod M$

$= ((78365 \bmod M) * (78365^{81452} \bmod M)) \bmod M$

$= (78365 * ((78365^2 \bmod M)^{81452/2} \bmod M)) \bmod M$

$= (78365 * ((78852)^{40726} \bmod M)) \bmod M$

$= (78365 * ((78852^2 \bmod M)^{20363} \bmod M)) \bmod M$

$= (78365 * (86632^{20363} \bmod M)) \bmod M$

$= (78365 * ((86632 \bmod M) * (86632^{20362} \bmod M)) \bmod M$

$= \ldots$

$= 45235$

Another way:

$$81453 = 2^{16} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^{9} + 2^{5} + 2^{3} + 2^{2} + 2^{0}$$

$$a^{81453} = a^{2^{16}} \cdot a^{2^{13}} \cdot a^{2^{12}} \cdot a^{2^{11}} \cdot a^{2^{10}} \cdot a^{2^{9}} \cdot a^{2^{5}} \cdot a^{2^{3}} \cdot a^{2^{2}} \cdot a^{2^{0}}$$

$$\begin{aligned}
&a^{81453} \bmod m = \\
&(\ldots(((((a^{2^{16}} \bmod m \cdot \\
&\quad a^{2^{13}} \bmod m\ ) \bmod m \cdot \\
&\quad a^{2^{12}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{11}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{10}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{9}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{5}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{3}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{2}} \bmod m) \bmod m \cdot \\
&\quad a^{2^{0}} \bmod m) \bmod m
\end{aligned}$$

The fast exponentiation algorithm computes $a^n \bmod m$ using $O(\log n)$ multiplications $\bmod m$

An integer *p* greater than 1 is called *prime* if the only positive factors of *p* are 1 and *p*.

A positive integer that is greater than 1 and is not prime is called *composite*.

fundamental theorem of arithmetic

Every positive integer greater than 1 has a unique prime factorization

48	=	2 • 2 • 2 • 2 • 3
591	=	3 • 197
45,523	=	45,523
321,950	=	2 • 5 • 5 • 47 • 137
1,234,567,890	=	2 • 3 • 3 • 5 • 3,607 • 3,803

If n is composite, it has a factor of size at most $\sqrt{n}$.

There are an infinite number of primes.

Proof by contradiction:

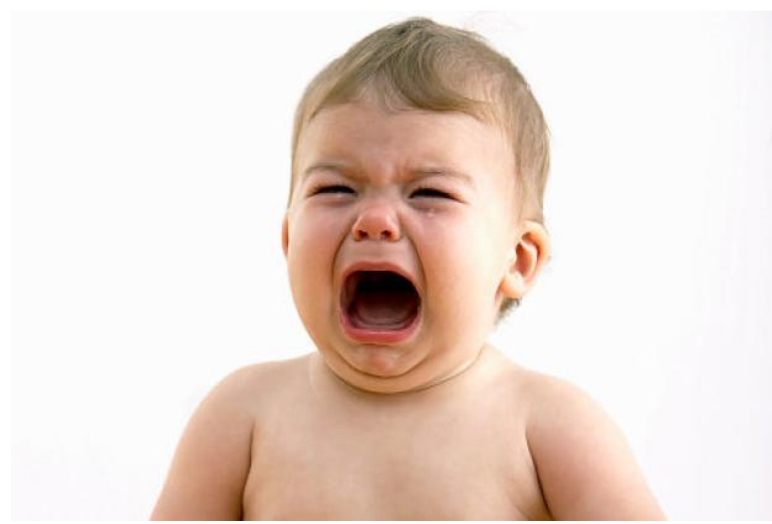
Suppose that there are only a finite number of primes:
$p_1, p_2, \ldots, p_n$

- Primality Testing
  - Given an integer n, determine if n is prime
- Factoring
  - Given an integer n, determine the prime factorization of n

factoring

Factor the following 232 digit number [RSA768]:

12301866845301177551304949583849627207
72853569595334792197322452151726400507
26365751874520219978646938995647494277
40638459251925573263034537315482685079
17026122142913461670429214311602221240
47927473779408066535141959745985690214
3413



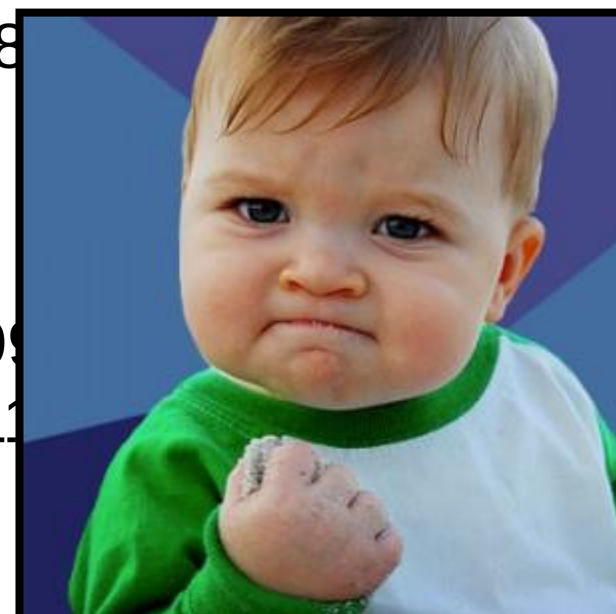
1230186684530117755130494958384962720772853569595334
7921973224521517264005072636575187452021997864693899
5647494277406384592519255732630345373154826850791702
6122142913461670429214311602221240479274737794080665
351419597459856902143413

=

3347807169895689878604416984821269081770479498 37
1376856891243138898288379387 17
430877378144679994 89

×

3674604366679959042824463379 643
4308764267603228381573966651 968
102700927987363089 17



GCD(a, b):

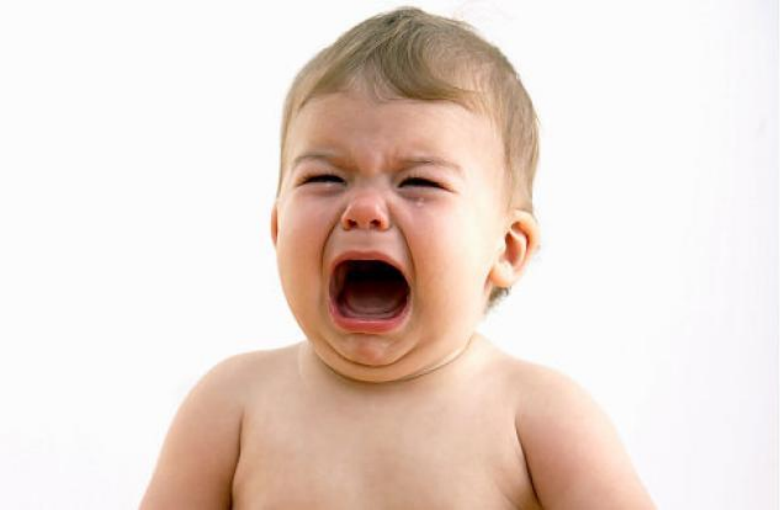
Largest integer d such that $d \mid a$ and $d \mid b$

- GCD(100, 125) =
- GCD(17, 49) =
- GCD(11, 66) =
- GCD(13, 0) =
- GCD(180, 252) =

$a = 2^3 \bullet 3 \bullet 5^2 \bullet 7 \bullet 11 = 46{,}200$

$b = 2 \bullet 3^2 \bullet 5^3 \bullet 7 \bullet 13 = 204{,}750$

$GCD(a, b) = 2^{\min(3,1)} \bullet 3^{\min(1,2)} \bullet 5^{\min(2,3)} \bullet 7^{\min(1,1)} \bullet 11^{\min(1,0)} \bullet 13^{\min(0,1)}$



Factoring is expensive!

Can we compute GCD(a,b) without factoring?

useful GCD fact

If a and b are positive integers, then

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

Proof:

By definition $a = (a \text{ div } b) \cdot b + (a \bmod b)$

If $d \mid a$ and $d \mid b$ then $d \mid (a \bmod b)$.

If $d \mid b$ and $d \mid (a \bmod b)$ then $d \mid a$.

euclid's algorithm

Repeatedly use the GCD fact to reduce numbers
until you get $\mathrm{GCD}(x, 0) = x$.

GCD(660,126)

GCD(x, y) = GCD(y, x mod y)

```
int GCD(int a, int b){ /* a >= b, b > 0 */
    int tmp;
    while (b > 0) {
        tmp = a % b;
        a = b;
        b = tmp;
    }
    return a;
}
```

Example: GCD(660, 126)