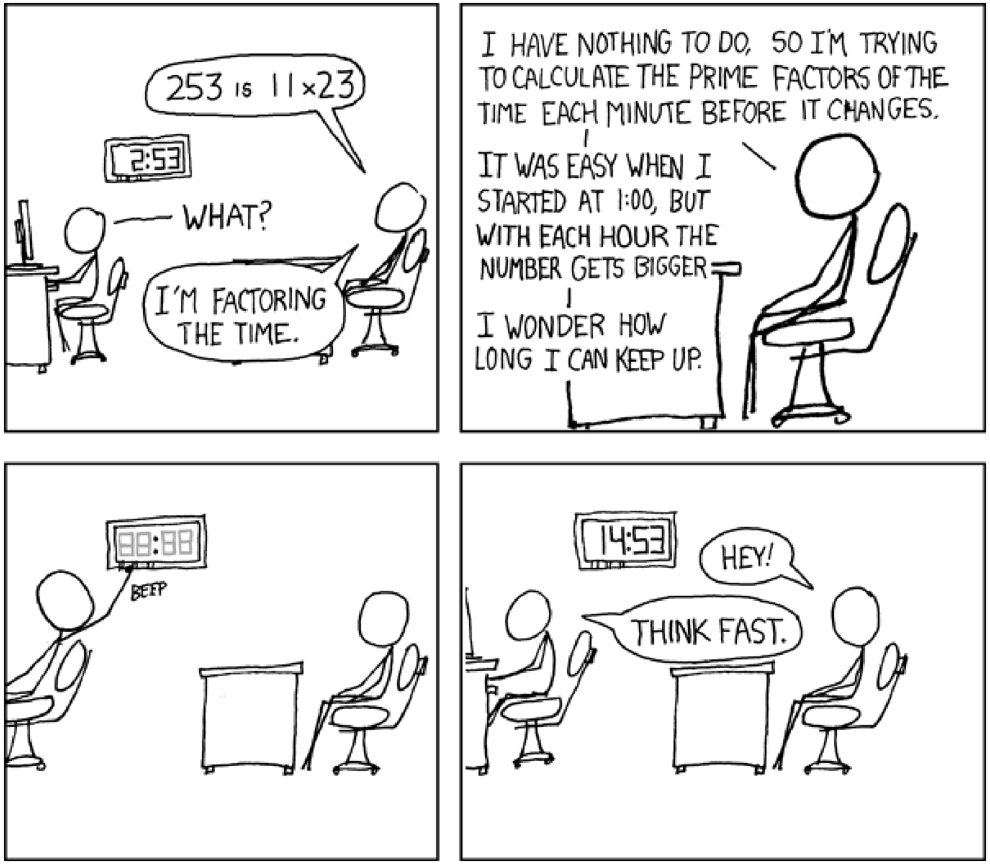
Fall 2015

Lecture 12: Primes, GCD, applications



$a \equiv b \mod m$
$c \equiv d \mod m$
$\Rightarrow ac \equiv bd \mod m$

Let n be an integer.
Prove that $n^2 \equiv 0 \pmod 4$ or $n^2 \equiv 1 \pmod 4$

$n \equiv 0 \quad 4 \Rightarrow n^2 \equiv 0^2 \mod 4$
$n \equiv 1 \quad 4 \Rightarrow n^2 \equiv 1^2 \mod 4$
$n \equiv 2 \quad 4 \Rightarrow n^2 \equiv 2^2 \equiv 0 \mod 4$
$n \equiv 3 \quad 4 \Rightarrow n^2 \equiv 9 \equiv 1 \mod 4$

Case 1 (n is even):

Suppose $n \equiv 0 \pmod 2$.
Then, $n = 2k$ for some integer k.
So, $n^2 = (2k)^2 = 4k^2$.
So, by definition of congruence, $n^2 \equiv 0 \pmod 4$.

Case 2 (n is odd):

Suppose $n \equiv 1 \pmod 2$.
Then, $n = 2k + 1$ for some integer k.
So, $n^2 = (2k + 1)^2 = 4k^2 + 4k + 1 = 4(k^2 + k) + 1$.
So, by definition of congruence, $n^2 \equiv 1 \pmod 4$.

n-bit unsigned integer representation

- Represent integer x as sum of powers of 2:
  If $x = \sum_{i=0}^{n-1} b_i 2^i$ where each $b_i \in \{0,1\}$
  then representation is $b_{n-1} \cdots b_2\, b_1\, b_0$

  99 = 64 + 32 + 2 + 1
  18 = 16 + 2

  01100011 = 99
  000 10010 = 18

- For n = 8:

  99: 0110 0011
  18: 0001 0010

sign-magnitude integer representation

n-bit signed integers

Suppose $-2^{n-1} < x < 2^{n-1}$

First bit as the sign, n-1 bits for the value

99 = 64 + 32 + 2 + 1

18 = 16 + 2

For n = 8:

99: 0110 0011

-18: 1001 0010

0110 0011
1001 0010

1111 0101

$-117 \neq 99 + (-18)$

Any problems with this representation?

two's complement representation

n-bit signed integers, first bit will still be the sign bit

$-1 \equiv m-1 \mod m$
$a \equiv a+m \mod m$

Suppose $0 \leq x < 2^{n-1}$,
  x is represented by the binary representation of x
Suppose $0 \leq x \leq 2^{n-1}$,
  $-x$ is represented by the binary representation of $2^n - x$

> **Key property:** Two's complement representation of any number y is equivalent to y mod 2^n so arithmetic works mod 2^n

$n=8$

99 = 64 + 32 + 2 + 1
18 = 16 + 2

$-x \equiv 2^n - x \mod 2^n$

-18 $\quad 256 - 18 = 238$
$= 128 + 64 + 32 + 8 + 4 + 2$

```
 11101110
 01100011
 --------
 01010001  = 81
```

For n = 8:
 99: 0110 0011
 -18: 1110 1110

sign-magnitude vs. two's complement

-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1111	1110	1101	1100	1011	1010	1001	0000	0001	0010	0011	0100	0101	0110	0111

Sign-Magnitude

-8	-7	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6	7
1000	1001	1010	1011	1100	1101	1110	1111	0000	0001	0010	0011	0100	0101	0110	0111

Two's complement

two's complement representation

- For $0 < x \leq 2^{n-1}$, $-x$ is represented by the binary representation of $2^n - x$

- To compute this: Flip the bits of x then add 1:
  - All 1's string is $2^n - 1$, so

    Flip the bits of x $\equiv$ replace x by $2^n - 1 - x$

$$
\begin{array}{r}
00010010 = 18 \\
11101101 \\
1 \\
\hline
11101110 \quad -18
\end{array}
$$

$$
\begin{array}{rl}
2^n - 1 & = 11 \cdots\cdots 11 \\
- & 0 x_{n-2} \cdots\cdots x_2 x_1 x_0 = x \\
\hline
(2^n - 1 - x) & = 1 \bar{x}_{n-2} \cdots\cdots \bar{x}_2 \bar{x}_1 \bar{x}_0 \\
 & +1 \\
\hline
 & 2^n - x
\end{array}
$$

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

$372 \qquad 3+7+2=12$

$372 = 3 \cdot 124$

$n = d_k d_{k-1} \cdots d_0$

$3 \mid n \iff 3 \mid d_k + d_{k-1} + \cdots + d_0$

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

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

$$\equiv d_k + d_{k-1} + \cdots + d_0 \mod 3$$

$$n \bmod 3 = d_k + \cdots + d_0 \mod 3.$$

$10 \equiv 1 \mod 3$

$10^2 \equiv 1^2 \mod 3$

$10^k \equiv 1^k \mod 3$

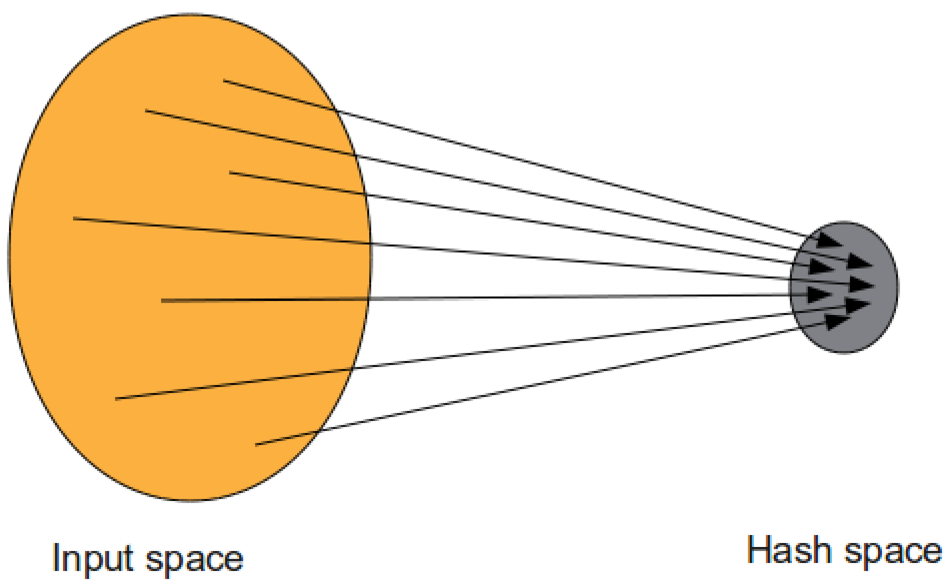
$d_k \cdot 10^k \equiv d_k \cdot 1^k$

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 $\mathrm{hash}(x) = (ax + b) \bmod p$

$x = 2^{46}$

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

$a[2^{46} \bmod p] = x$

$2^{46} \equiv 2^{46} + p \mod p$

pseudo-random number generation

Linear Congruential method:

x_0

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

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

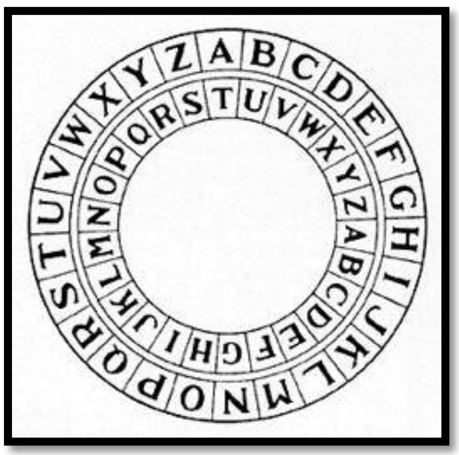
Adv : fast

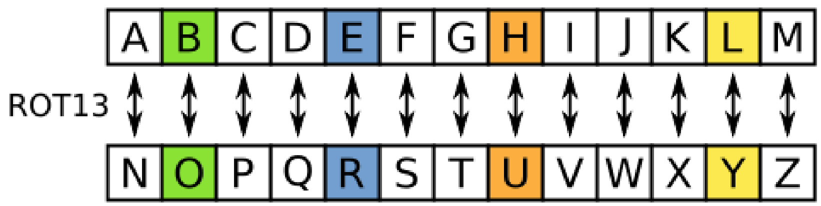
Dis : ~~A~~ No too random.

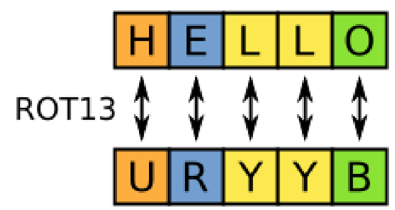
[good for some applications, really bad for many others]

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

```
/**//*/};)/**/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						

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}

  1787240000000000

  b

- Compute $78365^{81453} \bmod 104729$

  a m

- Output is small
  - need to keep intermediate results small

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

81453 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)

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

Let M = 104729 | program trace

$78365^{81453} \bmod M$

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

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

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

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

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

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

$= \ldots$

$= 45235$

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.

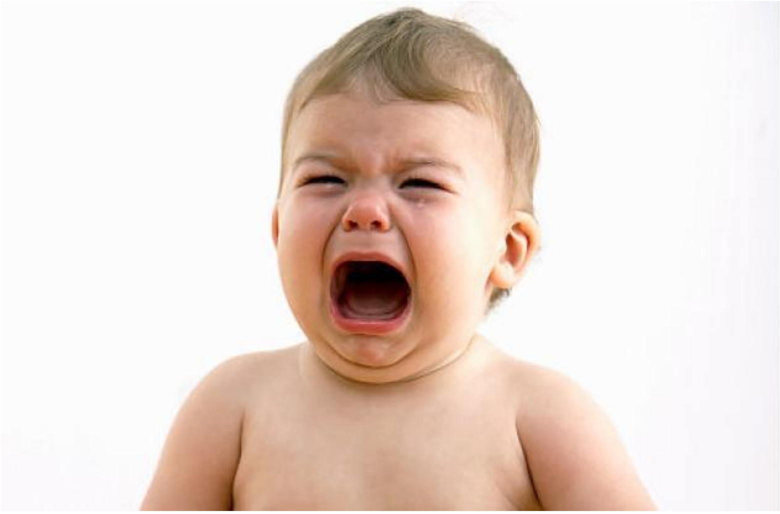
Proof by contradiction:

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

Factor the following 232 digit number [RSA768]:

1230186684530117755130494958384962720772
8535695953347921973224521517264005072636
5751874520219978646938995647494277406384
5925192557326303453731548268507917026122
1429134616704292143116022212404792747377
94080665351419597459856902143413



1230186684530117755130494958384962720772853569595334 79
2197322452151726400507263657518745202199786469389956 47
4942774063845925192557326303453731548268507917026122 14
2913461670429214311602221240479274737794080665351419 59
745985690214 3413

=

334780716989568987860441698482126908177047949837
1376856891243138898288379387 … 17
430877378144679999489

×

3674604366679959042824463379 … 643
43087642676032283815739666511 … 968
10270092798736308917

GCD(a, b):

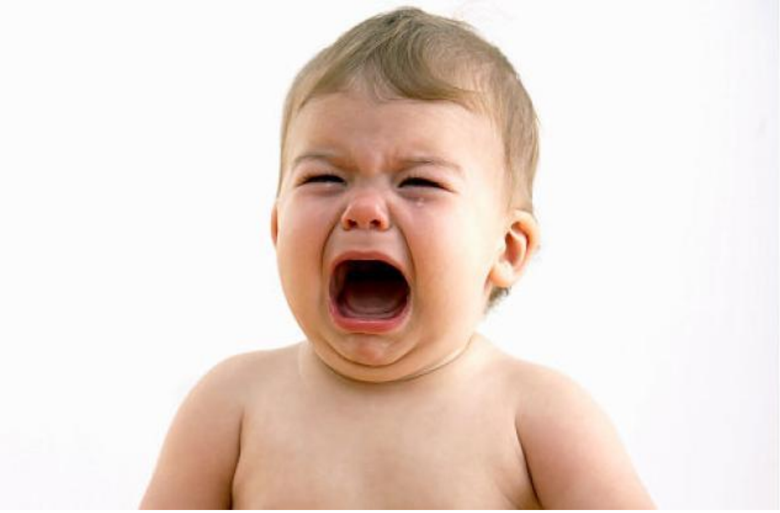
Largest integer d such that $d \mid a$ and $d \mid b$

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

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

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

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



Factoring is expensive!

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

If a and b are positive integers, then

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

Proof:

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

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)