Today: finding inverses quickly.

Euclid’s Algorithm.
Runtime.
Euclid’s Extended Algorithm.
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9? Yes. $5$
$2(5) = 10 = 1 \mod 9.$
Does 6 have an inverse mod 9? No.

$x$ has an inverse modulo $m$ if and only if
$gcd(x, m) > 1$? No.
$gcd(x, m) = 1$? Yes.

Today:
Compute gcd!
Compute Inverse modulo $m$. 
**Notation:** $d | x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

**Fact:** If $d | x$ and $d | y$ then $d | (x + y)$ and $d | (x - y)$.

**Proof:** $d | x$ and $d | y$ or

$x = \ell d$ and $y = kd$

$\implies x - y = kd - \ell d = (k - \ell)d \implies d | (x - y)$

□
More divisibility

**Notation:** \( d \mid x \) means “\( d \) divides \( x \)” or \( x = kd \) for some integer \( k \).

**Lemma 1:** If \( d \mid x \) and \( d \mid y \) then \( d \mid y \) and \( d \mid \text{mod} \ (x, y) \).

**Proof:**
\[
\text{mod} \ (x, y) = x - \lfloor x/y \rfloor \cdot y \\
= x - s \cdot y \quad \text{for integer } s \\
= kd - s \ell d \quad \text{for integers } k, \ell \\
= (k - s \ell) d
\]

Therefore \( d \mid \text{mod} \ (x, y) \). And \( d \mid y \) since it is in condition.

**Lemma 2:** If \( d \mid y \) and \( d \mid \text{mod} \ (x, y) \) then \( d \mid y \) and \( d \mid x \).

**Proof:** Similar. Try this at home.

**GCD Mod Corollary:** \( \gcd(x, y) = \gcd(y, \text{mod} \ (x, y)) \).

**Proof:** \( x \) and \( y \) have **same** set of common divisors as \( x \) and \( \text{mod} \ (x, y) \) by Lemma.

Same common divisors \( \implies \) largest is the same.
Euclid’s algorithm.

**GCD Mod Corollary:** \( \gcd(x, y) = \gcd(y, \mod(x, y)) \).

```plaintext
gcd (x, y)
  if (y = 0) then
    return x
  else
    return gcd(y, mod(x, y)) ***
```

**Theorem:** Euclid’s algorithm computes the greatest common divisor of \( x \) and \( y \) if \( x \geq y \).

**Proof:** Use Strong Induction.

**Base Case:** \( y = 0 \), “\( x \) divides \( y \) and \( x \)”

\[ \implies \text{“\( x \) is common divisor and clearly largest.”} \]

**Induction Step:** \( \mod(x, y) < y \leq x \) when \( x \geq y \)

call in line (***) meets conditions plus arguments “smaller”
and by strong induction hypothesis computes \( \gcd(y, \mod(x, y)) \)
which is \( \gcd(x, y) \) by GCD Mod Corollary.
Excursion: Value and Size.

Before discussing running time of gcd procedure...
What is the value of 1,000,000? one million or 1,000,000!
What is the “size” of 1,000,000? Number of digits: 7.
Number of bits: 21.
For a number $x$, what is its size in bits?

$$n = b(x) \approx \log_2 x$$
GCD procedure is fast.

**Theorem:** GCD uses $2n$ “divisions” where $n$ is the number of bits. Is this good? Better than trying all numbers in $\{2, \ldots y/2\}$?

Check 2, check 3, check 4, check 5 . . . , check $y/2$.

$2^{n-1}$ divisions! Exponential dependence on size!

101 bit number. $2^{100} \approx 10^{30} = \text{“million, trillion, trillion” divisions}!$

$2n$ is much faster! .. roughly 200 divisions.
Algorithms at work.

Trying everything
Check 2, check 3, check 4, check 5 ... , check \( y/2 \).
“gcd(x, y)” at work.

\[
\begin{align*}
gcd(700, 568) \\
gcd(568, 132) \\
gcd(132, 40) \\
gcd(40, 12) \\
gcd(12, 4) \\
gcd(4, 0) \\
4
\end{align*}
\]

Notice: The first argument decreases rapidly.
At least a factor of 2 in two recursive calls.
(The second is less than the first.)
Proof.

gcd \( (x, y) \)
  if \( (y = 0) \) then
    return \( x \)
  else
    return gcd(\( y, \) mod(\( x, y) \))

Theorem: GCD uses \( O(n) \) ”divisions” where \( n \) is the number of bits.

Proof:

Fact:
First arg decreases by at least factor of two in two recursive calls.

Proof of Fact: Recall that first argument decreases every call. After \( 2 \log_2 x \) recursive calls, argument \( x \) is a bit number. One more recursive call to finish. “mod(\( x, y) \leq x/2.”

Case 1: \( y = x/2 \), first argument is \( y \) division per recursive call. mod(\( x, y) \) is second argument in next recursive call, and becomes the first argument in the next one.

\[
\text{mod}(x, y) = x - y \left\lfloor \frac{x}{y} \right\rfloor = x - y \leq x - x/2 = x/2
\]
Finding an inverse?

We showed how to efficiently tell if there is an inverse. Extend Euclid’s algo to find inverse.
Euclid’s GCD algorithm.

gcd (x, y)
  if (y = 0) then
    return x
  else
    return gcd(y, mod(x, y))

Computes the gcd(x, y) in $O(n)$ divisions.
For $x$ and $m$, if gcd($x, m$) = 1 then $x$ has an inverse modulo $m$. 
Multiplicative Inverse.

GCD algorithm used to tell if there is a multiplicative inverse. How do we find a multiplicative inverse?
Extended GCD

Euclid’s Extended GCD Theorem: For any $x, y$ there are integers $a, b$ such that

$$ax + by = \gcd(x, y) = d \quad \text{where } d = \gcd(x, y).$$

“Make $d$ out of sum of multiples of $x$ and $y$.”

What is multiplicative inverse of $x$ modulo $m$?

By extended GCD theorem, when $\gcd(x, m) = 1$.

$$ax + bm = 1$$

$$ax \equiv 1 - bm \equiv 1 \pmod{m}.$$ 

So $a$ multiplicative inverse of $x$ if $\gcd(a, x) = 1$!!

Example: For $x = 12$ and $y = 35$, $\gcd(12, 35) = 1$.

$$(3)12 + (-1)35 = 1.$$ 

$a = 3$ and $b = -1$.

The multiplicative inverse of $12 \pmod{35}$ is $3$. 

Make $d$ out of $x$ and $y$..?

\[
gcd(35, 12) \\
gcd(12, 11) ;; \ gcd(12, 35 \% 12) \\
gcd(11, 1) ;; \ gcd(11, 12 \% 11) \\
gcd(1, 0) \\
1
\]

How did gcd get 11 from 35 and 12?
\[
35 - \left\lfloor \frac{35}{12} \right\rfloor 12 = 35 - (2)12 = 11
\]

How does gcd get 1 from 12 and 11?
\[
12 - \left\lfloor \frac{12}{11} \right\rfloor 11 = 12 - (1)11 = 1
\]

Algorithm finally returns 1.

But we want 1 from sum of multiples of 35 and 12?

Get 1 from 12 and 11.
\[
1 = 12 - (1)11 = 12 - (1)(35 - (2)12) = (3)12 + (-1)35
\]

Get 11 from 35 and 12 and plugin.... Simplify. $a = 3$ and $b = -1$. 
Extended GCD Algorithm.

\[
\text{ext-gcd}(x, y)
\]

if \( y = 0 \) then return \((x, 1, 0)\)
else

\[
(d, a, b) := \text{ext-gcd}(y, \text{mod}(x, y))
\]

return \((d, b, a - \text{floor}(x/y) \times b)\)

Claim: Returns \((d, a, b)\): \(d = \gcd(a, b)\) and \(d = ax + by\).

Example: \(a - \lfloor x/y \rfloor \cdot b = 1 - \lfloor 11/1 \rfloor \cdot 0 = 10 - \lfloor 12/11 \rfloor \cdot 1 = -11 - \lfloor 35/12 \rfloor \cdot (-1) = 3\)

\[
\text{ext-gcd}(35, 12)
\]
\[
\text{ext-gcd}(12, 11)
\]
\[
\text{ext-gcd}(11, 1)
\]
\[
\text{ext-gcd}(1, 0)
\]

return \((1, 1, 0)\) ;; \(1 = (1)1 + (0)0\)

return \((1, 0, 1)\) ;; \(1 = (0)11 + (1)1\)

return \((1, 1, -1)\) ;; \(1 = (1)12 + (-1)11\)

return \((1, -1, 3)\) ;; \(1 = (-1)35 + (3)12\)
Extended GCD Algorithm.

\[
\text{ext-gcd}(x, y)
\]

\[
\begin{align*}
\text{if } y &= 0 \text{ then return }(x, 1, 0) \\
\text{else} \\
&\quad (d, a, b) := \text{ext-gcd}(y, \text{mod}(x, y)) \\
&\quad \text{return } (d, b, a - \text{floor}(x/y) \times b)
\end{align*}
\]

**Theorem:** Returns \((d, a, b)\), where \(d = \gcd(a, b)\) and

\[
d = ax + by.
\]
Correctness.

**Proof:** Strong Induction.\(^1\)

**Base:** \(\text{ext-gcd}(x, 0)\) returns \((d = x, 1, 0)\) with \(x = (1)x + (0)y\).

**Induction Step:** Returns \((d, A, B)\) with \(d = Ax + By\)

Ind hyp: \(\text{ext-gcd}(y, \mod(x, y))\) returns \((d^*, a, b)\) with 
\[d^* = ay + b(\mod(x, y))\]

\(\text{ext-gcd}(x, y)\) calls \(\text{ext-gcd}(y, \mod(x, y))\) so

\[
d = d^* = ay + b \cdot (\mod(x, y))
\]

\[
= ay + b \cdot (x - \left\lfloor \frac{x}{y} \right\rfloor y)
\]

\[
= bx + (a - \left\lfloor \frac{x}{y} \right\rfloor \cdot b)y
\]

And \(\text{ext-gcd}\) returns \((d, b, (a - \left\lfloor \frac{x}{y} \right\rfloor \cdot b))\) so theorem holds! \(\square\)

\(^1\)Assume \(d\) is \(gcd(x, y)\) by previous proof.
ext-gcd(x, y)
    if y = 0 then return(x, 1, 0)
    else
        (d, a, b) := ext-gcd(y, mod(x, y))
        return (d, b, a - floor(x/y) * b)

Recursively: \[ d = ay + b(x - \left\lfloor \frac{x}{y} \right\rfloor \cdot y) \implies d = bx - (a - \left\lfloor \frac{x}{y} \right\rfloor b)y \]

Returns \((d, b, (a - \left\lfloor \frac{x}{y} \right\rfloor \cdot b))\).
Conclusion: Can find multiplicative inverses in \(O(n)\) time!

Very different from elementary school: try 1, try 2, try 3...

\[2^{n/2}\]

Inverse of 500,000,357 modulo 1,000,000,000,000? \(\leq 80\) divisions.

versus 1,000,000

Internet Security.
Public Key Cryptography: 512 digits.

512 divisions vs.

\[(100000000000000000000000000000000000000000)^5\] divisions.

Next lecture!