Inverses

Today: finding inverses quickly.
Inverses

Today: finding inverses quickly.

Euclid’s Algorithm.
Inverses

Today: finding inverses quickly.

Euclid’s Algorithm.
Runtime.
Today: finding inverses quickly.

Euclid’s Algorithm.
Runtime.
Euclid’s Extended Algorithm.
Does 2 have an inverse mod 8?

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 \( \gcd(x, m) > 1? \)

Yes.

Today: Compute gcd! Compute Inverse modulo m.
Does 2 have an inverse mod 8? No.

Today:
Compute gcd!
Compute Inverse modulo m.
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9?

\[ x \mod m \text{ has an inverse } \iff \gcd(x, m) = 1? \]
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9? Yes.
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9? Yes. 5
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9? Yes. 5
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9? Yes. 5
   \[2(5) = 10 = 1 \mod 9.\]
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?
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.
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
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
\( \gcd(x, m) > 1? \)
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
\[ \gcd(x, m) > 1? \text{ No.} \]
\[ \gcd(x, m) = 1? \]
Does 2 have an inverse mod 8? No.

Does 2 have an inverse mod 9? Yes. $5$

$2(5) = 10 = 1 \pmod{9}$.

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.
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

\[ \gcd(x, m) > 1? \text{ No.} \]

\[ \gcd(x, m) = 1? \text{ Yes.} \]

Today:

Compute \( \gcd \)!
Does 2 have an inverse mod 8? No.
Does 2 have an inverse mod 9? Yes. 5
    \[2(5) = 10 = 1 \pmod{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 \(m\).
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 $m$. 
Divisibility...

**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:**

If $d | x$ and $d | y$ or $x = ℓd$ and $y = kd =⇒ x - y = kd - ℓd = (k - ℓ)d =⇒ d | (x - y)$. 

$d | x$ and $d | y$ or $x = ℓd$ and $y = kd$.
Notation: \( d \mid x \) means “\( d \) divides \( x \)” or \( x = kd \) for some integer \( k \).
Divisibility...

**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 $d | (x - y)$. 
Divisibility...

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

Fact: If \( d \mid x \) and \( d \mid y \) then \( d \mid (x + y) \) and \( d \mid (x - y) \).

Proof: \( d \mid x \) and \( d \mid y \) or
\( x = \ell d \) and \( y = kd \).
Divisibility...

**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

$x = ℓd$ and $y = kd$

$⇒ x − y = kd − ℓd$
Notation: $d \mid x$ means “$d$ divides $x$” or 
$x = kd$ for some integer $k$.

Fact: If $d \mid x$ and $d \mid y$ then $d \mid (x + y)$ and $d \mid (x - y)$.

Proof: $d \mid x$ and $d \mid y$ or 
$x = \ell d$ and $y = kd$

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

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

$⇒ x − y = kd − ℓd = (k − ℓ)d ⇒ d|(x − y)$
More divisibility

**Notation:** $d | x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$. 
More divisibility

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

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

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

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

Proof: 
\[
\text{mod}(x,y) = x - \lfloor x/y \rfloor \cdot y
\]
More divisibility

**Notation:** $d|x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

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

**Proof:**

\[
\text{mod}(x, y) = x - \left\lfloor \frac{x}{y} \right\rfloor \cdot y
\]

\[
= x - s \cdot y \quad \text{for integer } s
\]
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:

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

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

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

**Proof:**

\[
\text{mod}(x,y) = x - \lfloor x/y \rfloor \cdot y
\]

\[
= x - s \cdot y \quad \text{for integer } s
\]

\[
= kd - s \ell d \quad \text{for integers } k, \ell
\]

\[
= (k - s\ell)d
\]
Notation: $d|\!\!\!x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

Lemma 1: If $d|\!\!\!x$ and $d|\!\!\!y$ then $d|\!\!\!y$ and $d|\!\!\!\\text{mod}(x, y)$.

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

Therefore $d|\!\!\!\text{mod}(x, y)$.

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

Proof: 
$x$ and $y$ have the same set of common divisors as $x$ and $\text{mod}(x, y)$ by Lemma. Same common divisors $\Rightarrow$ largest is the same.
More divisibility

**Notation:** $d | x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

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

**Proof:**
\[
\text{mod} \ (x, y) = x - \left\lfloor \frac{x}{y} \right\rfloor \cdot y \\
= x - s \cdot y \quad \text{for integer } s \\
= kd - s\ell d \quad \text{for integers } k, \ell \\
= (k - s\ell) d
\]
Therefore $d | \text{mod} \ (x, y)$. And $d | y$ since it is in condition.

**Lemma 2:**

**Proof...:** Similar.

**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 $\Rightarrow$ largest is the same.
More divisibility

**Notation:** $d | x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

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

**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 $d | \text{mod}(x, y)$. And $d | y$ since it is in condition. \qed
More divisibility

**Notation:** $d | x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

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

**Proof:**
\[
\text{mod}(x, y) = x - \left\lfloor \frac{x}{y} \right\rfloor \cdot y
= x - s \cdot y \quad \text{for integer } s
= kd - s \ell d \quad \text{for integers } k, \ell
= (k - s \ell) d
\]
Therefore $d | \text{mod}(x, y)$. And $d | y$ since it is in condition. \qed

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

**Proof...:** Similar.
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.
More divisibility

**Notation:** $d | x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

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

**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 $d | \text{mod}(x, y)$. And $d | y$ since it is in condition. □

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

**Proof...:** Similar. Try this at home. □
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. \( \square \)

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. \( \square \).

GCD Mod Corollary: \( \gcd(x, y) = \gcd(y, \text{mod} (x, y)) \).
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.
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 \( \Rightarrow \) largest is the same.
More divisibility

**Notation:** $d|x$ means “$d$ divides $x$” or $x = kd$ for some integer $k$.

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

**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 $d| \text{mod}(x,y)$. And $d|y$ since it is in condition.

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

**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))  ***
```
Euclid’s algorithm.

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

\[
\text{gcd} (x, y) \\
\quad \text{if} \ (y = 0) \ \text{then} \\
\quad \quad \text{return} \ x \\
\quad \text{else} \\
\quad \quad \text{return} \ \text{gcd}(y, \mod(x, y)) \quad ***
\]

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

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

\[
gcd \ (x, \ y) \\ \text{if} \ (y = 0) \ \text{then} \\ \quad \text{return} \ x \\ \text{else} \\ \quad \text{return} \ \gcd(y, \mod(x, y)) \quad ***
\]

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

**Proof:** Use Strong Induction.
Euclid’s algorithm.

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

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 \( x \)”
Euclid’s algorithm.

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

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

\( \implies \) “\( x \) is common divisor and clearly largest.”

**Induction Step:** \( \mod(x, y) < y \leq x \) when \( x \geq y \)
Euclid’s algorithm.

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

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

\[
\text{if } (y = 0) \text{ then } \\
\quad \text{return } x \\
\text{else } \\
\quad \text{return } \gcd(y, \mod(x, y)) \quad ***
\]

**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 \( x \)”

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

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

call in line (***). meets conditions plus arguments “smaller”
Euclid’s algorithm.

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

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 \(x\)”
  \(\implies\) “\(x\) 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
Euclid’s algorithm.

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

```
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 \( x \)”

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

**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 \( \gcd(y, \mod(x, y)) \)
Euclid’s algorithm.

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

\[
gcd (x, y) \\
\text{if } (y = 0) \text{ then} \\
\quad \text{return } x \\
\text{else} \\
\quad \text{return } \gcd(y, \mod(x, y)) \\
\]

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

**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.
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 “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...
Excursion: Value and Size.

Before discussing running time of gcd procedure...
What is the value of 1,000,000?
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!
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?
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.
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.
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?
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?

$$n = b(x) \approx \log_2 x$$
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.
Theorem: GCD uses $2n$ “divisions” where $n$ is the number of bits.

Is this good?
**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\}?
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,
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,
**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,
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 $y/2$. 
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!
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 \ldots, check $y/2$.

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

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

\[
gcd(x, y), \ gcd(700, 568), \ gcd(568, 132), \ gcd(132, 40), \ gcd(40, 12), \ gcd(12, 4), \ gcd(4, 0) = 4
\]

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

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

\[ \text{gcd}(700, 568) \]
\[ \text{gcd}(568, 132) \]
\[ \text{gcd}(132, 40) \]
\[ \text{gcd}(40, 12) \]
\[ \text{gcd}(12, 4) \]
\[ \text{gcd}(4, 0) = 4 \]

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

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

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

\[ \text{gcd}(700, 568) \]
Algorithms at work.

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

\[
\begin{align*}
gcd(700, 568) \\
gcd(568, 132)
\end{align*}
\]
Algorithms at work.

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

\[
\begin{align*}
\text{gcd}(700, 568) \\
\text{gcd}(568, 132) \\
\text{gcd}(132, 40)
\end{align*}
\]
Algorithms at work.

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

\[
\begin{align*}
gcd(700, 568) \\
gcd(568, 132) \\
gcd(132, 40) \\
gcd(40, 12)
\end{align*}
\]
Algorithms at work.

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

\[
gcd(700, 568) \\
gcd(568, 132) \\
gcd(132, 40) \\
gcd(40, 12) \\
gcd(12, 4)
\]
Algorithms at work.

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

$$\text{gcd}(700, 568)$$
$$\text{gcd}(568, 132)$$
$$\text{gcd}(132, 40)$$
$$\text{gcd}(40, 12)$$
$$\text{gcd}(12, 4)$$
$$\text{gcd}(4, 0)$$
Algorithms at work.

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

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

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

```
gcd(700, 568)  
gcd(568, 132)  
gcd(132, 40)   
gcd(40, 12)    
gcd(12, 4)     
gcd(4, 0)      
    4
```

Notice: The first argument decreases rapidly.
Try everything
Check 2, check 3, check 4, check 5 . . . , check $y/2$.
“$\text{gcd}(x, y)$” at work.

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

Notice: The first argument decreases rapidly.
At least a factor of 2 in two recursive calls.
Algorithms at work.

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

“gcd($x$, $y$)” at work.

\[
gcd(700, 568) \\
gcd(568, 132) \\
gcd(132, 40) \\
gcd(40, 12) \\
gcd(12, 4) \\
gcd(4, 0) \\
4
\]

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

\[
gcd \ (x, \ y) \\
\text{if} \ (y = 0) \ \text{then} \\
\quad \text{return} \ x \\
\text{else} \\
\quad \text{return} \ gcd(y, \ mod(x, \ y))
\]

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

\[
gcd (x, y) \\
\quad \text{if } (y = 0) \text{ then} \\
\quad \quad \text{return } x \\
\quad \text{else} \\
\quad \quad \text{return } gcd(y, \text{ mod}(x, y))
\]

**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.

\[
gcd (x, y)
\]
\[
ext if (y = 0) then
\]
\[
return x
\]
\[
ext else
\]
\[
return gcd(y, \text{mod}(x, y))
\]

**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.
After \(2\log_2 x = O(n)\) recursive calls, argument \(x\) is 1 bit number.
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.
After $2 \log_2 x = O(n)$ recursive calls, argument $x$ is 1 bit number.
One more recursive call to finish.
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.

After $2\log_2 x = O(n)$ recursive calls, argument $x$ is 1 bit number.
One more recursive call to finish.
1 division per recursive call.
Proof.

```plaintext
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.

After $2 \log_2 x = O(n)$ recursive calls, argument $x$ is 1 bit number.
One more recursive call to finish.
1 division per recursive call.
$O(n)$ divisions. 

Proof.

\[
\text{gcd} (x, y) \\
\quad \text{if } (y = 0) \text{ then} \\
\quad \quad \text{return } x \\
\quad \text{else} \\
\quad \quad \text{return } \text{gcd}(y, \text{mod}(x, y))
\]

**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.
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.

Case 1: \( y \leq x/2 \), first argument is \( y \)
    \( \implies \) true in one recursive call;
Proof.

\[
gcd(x, y) \\
\text{if (} y = 0 \text{) then} \\
\quad \text{return } x \\
\text{else} \\
\quad \text{return } gcd(y, \text{mod}(x, y))
\]

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.

Case 2: Will show “\( y > x/2 \)” \( \implies \) “\( \text{mod}(x, y) \leq x/2. \)”
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.

Case 2: Will show “$y > x/2$” $\implies$ “$mod(x, y) \leq x/2$.”

$mod(x, y)$ is second argument in next recursive call,
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.

Case 2: Will show “$y > x/2$” $\Rightarrow$ “$mod(x, y) \leq x/2$.”

    mod $(x, y)$ is second argument in next recursive call, and becomes the first argument in the next one.
Proof.

\[
gcd (x, y) \\
\quad \text{if } (y = 0) \text{ then} \\
\quad \quad \text{return } x \\
\quad \text{else} \\
\quad \quad \text{return } gcd(y, \mod(x, y))
\]

**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.

Case 2: Will show “\( y > x/2 \)” \( \implies \) “\( \mod(x, y) \leq x/2. \)”

When \( y > x/2 \), then

\[
\left\lfloor \frac{x}{y} \right\rfloor = 1,
\]
Proof.

\[
gcd (x, y) \\
\quad \text{if } (y = 0) \text{ then} \\
\quad \quad \text{return } x \\
\quad \text{else} \\
\quad \quad \text{return } gcd(y, \text{mod}(x, y))
\]

**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.

Case 2: Will show "\( y > x/2 \)" \( \implies \) "\( \text{mod}(x, y) \leq x/2.\)"

When \( y > x/2 \), then

\[
\left\lfloor \frac{x}{y} \right\rfloor = 1,
\]

\[
\text{mod} (x, y) = x - y \left\lfloor \frac{x}{y} \right\rfloor =
\]
Proof.

\[
gcd (x, y) \\
\text{if } (y = 0) \text{ then} \\
\quad \text{return } x \\
\text{else} \\
\quad \text{return } gcd(y, \mod(x, y))
\]

**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.

Case 2: Will show \( y > x/2 \) \( \implies \) "\( \mod(x, y) \leq x/2 \.)."

When \( y > x/2 \), then

\[
\left\lfloor \frac{x}{y} \right\rfloor = 1,
\]

\[
\mod (x, y) = x - y \left\lfloor \frac{x}{y} \right\rfloor = x - y \leq x - x/2
\]
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.

Case 2: Will show “$y > x/2$” $\implies$ “$\text{mod}(x, y) \leq x/2.$”
When $y > x/2$, then

$$\left\lfloor \frac{x}{y} \right\rfloor = 1,$$

$$\text{mod} (x, y) = x - y \left\lfloor \frac{x}{y} \right\rfloor = x - y \leq x - x/2 = x/2$$
Proof.

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

**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.

Case 2: Will show \( y > x/2 \) \( \implies \) \( \text{mod}(x, y) \leq x/2. \)

When \( y > x/2 \), then

\[
\left\lfloor \frac{x}{y} \right\rfloor = 1,
\]

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

We showed how to efficiently tell if there is an inverse.
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))
```
Euclid’s GCD algorithm.

```plaintext
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.
Euclid’s GCD algorithm.

\[
gcd (x, y) \\
\quad \text{if } (y = 0) \text{ then} \\
\quad \quad \text{return } x \\
\quad \text{else} \\
\quad \quad \text{return } gcd(y, \mod(x, y))
\]

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.
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

$$ax + by = \gcd(x, y)$$
Extended GCD

Euclid’s Extended GCD Theorem: For any $x, y$ there are integers $a, b$ such that
\[ ax + by = \gcd(x, y) = d \quad \text{where } d = \gcd(x, y). \]
Extended GCD

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

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

where $d = \text{gcd}(x, y)$.

“Make $d$ out of sum of multiples of $x$ and $y$.”
Euclid’s Extended GCD Theorem: For any $x, y$ there are integers $a, b$ such that
\[ ax + by = \gcd(x, y) = d \quad \text{where} \quad d = \gcd(x, y). \]

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

What is multiplicative inverse of $x$ modulo $m$?
Euclid’s Extended GCD Theorem: For any $x, y$ there are integers $a, b$ such that

$$ax + by = \gcd(x, y) = d$$

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 $\gcd(x, m) = 1$. 

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

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

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

The multiplicative inverse of 12 (mod 35) is 3.
Euclid’s Extended GCD Theorem: For any $x, y$ there are integers $a, b$ such that

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

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

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

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

$$ax + bm = 1$$
Euclid’s Extended GCD Theorem: For any $x, y$ there are integers $a, b$ such that
$$ax + by = \gcd(x, y) = d$$ 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 $\gcd(x, m) = 1$.

$$ax + bm = 1$$

$$ax \equiv 1 - bm \equiv 1 \pmod{m}.$$
Euclid’s Extended GCD Theorem: For any \(x, y\) there are integers \(a, b\) such that \(ax + by = \gcd(x, y) = d\) 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 \(\gcd(a, x) = 1!!\)
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 $\gcd(x, m) = 1$.

$$ax + bm = 1$$
$$ax \equiv 1 \pmod{m}.$$

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

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

$$(3)12 + (-1)35 = 1.$$
**Extended GCD**

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

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

“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 $y = 35$, $\gcd(12, 35) = 1$.

$$ (3)12 + (-1)35 = 1. $$
Euclid’s Extended GCD Theorem: For any \( x, y \) there are integers \( a, b \) such that 
\[
ax + by = \gcd(x, y) = d \quad \text{where} \quad d = \gcd(x, y).
\]

“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 \( b = -1 \).
Extended GCD

**Euclid’s Extended GCD Theorem:** For any \( x, y \) there are integers \( a, b \) such that

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

“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 (mod 35) is 3.
Make $d$ out of $x$ and $y$..?

\[
gcd(35, 12)
\]
Make $d$ out of $x$ and $y$..?

$$\text{gcd}(35, 12)$$
$$\text{gcd}(12, 11) ;; \text{gcd}(12, 35\%12)$$
Make $d$ out of $x$ and $y$..?

\[
\gcd(35, 12) \\
gcd(12, 11) ;; \quad \gcd(12, 35 \mod 12) \\
gcd(11, 1) ;; \quad \gcd(11, 12 \mod 11)
\]
Make $d$ out of $x$ and $y$..?

\[
\begin{align*}
gcd(35, 12) \\
gcd(12, 11) \quad ;; \quad gcd(12, 35 \mod 12) \\
gcd(11, 1) \quad ;; \quad gcd(11, 12 \mod 11) \\
gcd(1, 0) \\
1
\end{align*}
\]
Make $d$ out of $x$ and $y$..?

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

How did gcd get 11 from 35 and 12?

\[
35 - \lfloor \frac{35}{12} \rfloor 12 = 35 - (2 \times 12) = 11
\]

How does gcd get 1 from 12 and 11?

\[
12 - \lfloor \frac{12}{11} \rfloor 11 = 12 - (1 \times 11) = 1
\]

Algorithm finally returns 1.

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

\[
1 = 12 - (1 \times 11) = 12 - (1 \times (35 - (2 \times 12))) = (3 \times 12) + (1 \times 1)
\]

Get 11 from 35 and 12 and plugin.... Simplify.

\[
a = 3 \quad \text{and} \quad b = -1.
\]
Make $d$ out of $x$ and $y$..?

$$\text{gcd}(35,12)$$
$$\text{gcd}(12, 11) ;; \text{gcd}(12, 35 \mod 12)$$
$$\text{gcd}(11, 1) ;; \text{gcd}(11, 12 \mod 11)$$
$$\text{gcd}(1,0)$$
$$1$$

How did gcd get 11 from 35 and 12?
$$35 - \lfloor \frac{35}{12} \rfloor \times 12 = 35 - (2)12 = 11$$
Make $d$ out of $x$ and $y$..?

$\text{gcd}(35,12)$
$\text{gcd}(12, 11) ;; \text{gcd}(12, 35 \text{ mod } 12)$
$\text{gcd}(11, 1) ;; \text{gcd}(11, 12 \text{ mod } 11)$
$\text{gcd}(1,0)$
$1$

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

How does gcd get 1 from 12 and 11?
Make $d$ out of $x$ and $y$...

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

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

How does gcd get 1 from 12 and 11?
\[12 - \lfloor \frac{12}{11} \rfloor \times 11 = 12 - (1)11 = 1\]
Make $d$ out of $x$ and $y$..?

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

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

```plaintext
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?
$35 - \lfloor \frac{35}{12} \rfloor 12 = 35 - (2)12 = 11$

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

Algorithm finally returns 1.

But we want 1 from sum of multiples of 35 and 12?
Get 1 from 12 and 11.
Make $d$ out of $x$ and $y$..?

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

How did \text{gcd} get 11 from 35 and 12?

\[35 - \left\lfloor \frac{35}{12} \right\rfloor 12 = 35 - (2)12 = 11\]

How does \text{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.

\[1 = 12 - (1)11\]
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.
\[
1 = 12 - (1)11 = 12 - (1)(35 - (2)12)
\]

Get 11 from 35 and 12 and plugin....
Make $d$ out of $x$ and $y$..?

\[
\begin{align*}
gcd(35, 12) \\
gcd(12, 11) ;;& \;
gcd(12, 35 \% 12) \\
gcd(11, 1) ;;& \;
gcd(11, 12 \% 11) \\
gcd(1, 0) & \;= 1
\end{align*}
\]

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

\[
\begin{align*}
gcd(35, 12) \\
gcd(12, 11) ;; \quad gcd(12, 35 \% 12) \\
gcd(11, 1) ;; \quad gcd(11, 12 \% 11) \\
gcd(1, 0) \\
1
\end{align*}
\]

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

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

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

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

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

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

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.

\[
\text{ext-gcd}(x,y) \\
\quad \text{if } y = 0 \text{ then return }(x,1,0) \\
\quad \text{else} \\
\quad \quad (d, a, b) := \text{ext-gcd}(y, \text{mod}(x,y)) \\
\quad \quad \text{return } (d, b, a - \text{floor}(x/y) \times b)
\]
Extended GCD Algorithm.

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

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

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)

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

ext-gcd(35,12)
Extended GCD Algorithm.

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

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

Example:

\[
\text{ext-gcd}(35, 12) \\
\text{ext-gcd}(12, 11)
\]
Extended GCD Algorithm.

```plaintext
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)
```

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

Example:

```plaintext
ext-gcd(35, 12)
  ext-gcd(12, 11)
  ext-gcd(11, 1)
```
Extended GCD Algorithm.

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

\[ \text{if } y = 0 \text{ then return } (x, 1, 0) \]

\[ \text{else} \]

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

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

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

Example:

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

```plaintext
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)
```

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

Example: \(a - \lfloor x/y \rfloor \cdot b = \)

```plaintext
ext-gcd(35, 12)
    ext-gcd(12, 11)
        ext-gcd(11, 1)
            ext-gcd(1, 0)
                return (1, 1, 0) ;; 1 = (1)1 + (0) 0
```
Extended GCD Algorithm.

\[
\text{ext-gcd}(x, y)
\]
\[
\text{if } y = 0 \text{ then return } (x, 1, 0)
\]
\[
\text{else }
\]
\[
(d, a, b) := \text{ext-gcd}(y, \text{mod}(x, y))
\]
\[
\text{return } (d, b, a - \text{floor}(x/y) \cdot b)
\]

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

Example:
\[
a - \lfloor x/y \rfloor \cdot b =
\]
\[
1 - \lfloor 11/1 \rfloor \cdot 0 = 1
\]

\[
\text{ext-gcd}(35, 12)
\]
\[
\text{ext-gcd}(12, 11)
\]
\[
\text{ext-gcd}(11, 1)
\]
\[
\text{ext-gcd}(1, 0)
\]
\[
\text{return } (1, 1, 0) ;; 1 = (1)1 + (0)0
\]
\[
\text{return } (1, 0, 1) ;; 1 = (0)11 + (1)1
\]
Extended GCD Algorithm.

```plaintext
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)
```

Claim: Returns \((d, a, b)\): \(d = \gcd(a, b)\) and \(d = ax + by\).
Example: \(a - \lfloor x/y \rfloor \cdot b = 0 - \lfloor 12/11 \rfloor \cdot 1 = -1\)

```plaintext
ext-gcd(35, 12)
  ext-gcd(12, 11)
    ext-gcd(11, 1)
      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
```
Extended GCD Algorithm.

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

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

Example: \(a - \lfloor x/y \rfloor \cdot b = \)

\[
1 - \lfloor 35/12 \rfloor \cdot (-1) = 3
\]

\[
\text{ext-gcd}(35, 12) \\
\quad \text{ext-gcd}(12, 11) \\
\quad \text{ext-gcd}(11, 1) \\
\quad \text{ext-gcd}(1, 0) \\
\quad \quad \text{return } (1, 1, 0) ;; 1 = (1)1 + (0)0 \\
\quad \quad \text{return } (1, 0, 1) ;; 1 = (0)11 + (1)1 \\
\quad \quad \text{return } (1, 1, -1) ;; 1 = (1)12 + (-1)11 \\
\quad \quad \text{return } (1, -1, 3) ;; 1 = (-1)35 + (3)12
\]
Extended GCD Algorithm.

```plaintext
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)

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

ext-gcd(35, 12)
  ext-gcd(12, 11)
    ext-gcd(11, 1)
      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.

```plaintext
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)
```
Extended GCD Algorithm.

```
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)
```

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

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

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

\(^1\)Assume \(d\) is \(gcd(x, y)\) by previous proof.
Correctness.

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

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

\(^1\)Assume \(d\) is \(\gcd(x, y)\) by previous proof.
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
\[d^* = ay + b(\mod (x, y))\]

\(^1\)Assume \(d\) is \(gcd(x, y)\) by previous proof.
Correctness.

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

**Base:** 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: ext-gcd\((y, \mod (x, y))\) returns \((d^*, a, b)\) with \(d^* = ay + b(\mod (x, y))\)

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

\(^1\)Assume \(d\) is \(gcd(x, y)\) by previous proof.
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: $\text{ext-gcd}(y, \ \text{mod}\ (x, y))$ returns $(d^*, a, b)$ with $d^* = ay + b(\ \text{mod}\ (x, y))$

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

$$d = d^* = ay + b(\ \text{mod}\ (x, y))$$

$^1$Assume $d$ is $\text{gcd}(x, y)$ by previous proof.
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: \(\text{ext-gcd}(y, \text{mod}(x, y))\) returns \((d^*, a, b)\) with \(d^* = ay + b(\text{mod}(x, y))\)

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

\[d = d^* = ay + b \cdot (\text{mod}(x, y))\]
\[= ay + b \cdot (x - \left\lfloor \frac{x}{y} \right\rfloor y)\]

\(^1\)Assume \(d\) is \(\text{gcd}(x, y)\) by previous proof.
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: \(\text{ext-gcd}(y, \text{mod}(x, y))\) returns \((d^*, a, b)\) with \(d^* = ay + b(\text{mod}(x, y))\)

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

\[
\begin{align*}
    d &= d^* \\
    &= ay + b \cdot (\text{mod}(x, y)) \\
    &= ay + b \cdot (x - \left\lfloor \frac{x}{y} \right\rfloor y) \\
    &= bx + (a - \left\lfloor \frac{x}{y} \right\rfloor \cdot b)y
\end{align*}
\]

\(^1\)Assume \(d\) is \(gcd(x, y)\) by previous proof.
Correctness.

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

**Base:** `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: `ext-gcd(y, mod(x, y))` returns `(d*, a, b)` with `d* = ay + b( mod(x, y))`

`ext-gcd(x, y)` calls `ext-gcd(y, mod(x, y))` so

\[
\begin{align*}
d &= d^* \\
&= ay + b \cdot ( mod(x, y)) \\
&= ay + b \cdot (x - \lfloor \frac{x}{y} \rfloor y) \\
&= bx + (a - \lfloor \frac{x}{y} \rfloor b) y
\end{align*}
\]

And ext-gcd returns `(d, b, (a − \lfloor \frac{x}{y} \rfloor b))` so theorem holds!

---

\(^1\) Assume `d` is `gcd(x, y)` by previous proof.
Correctness.

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

**Base:** 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: ext-gcd\((y, \mod(x, y))\) returns \((d^*, a, b)\) with 
\[d^* = ay + b(\mod(x, y))\]

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

\[
d = d^* = ay + b \cdot (\mod(x, y))
\]
\[
= ay + b \cdot (x - \left\lfloor \frac{x}{y} \right\rfloor y)
\]
\[
= bx + (a - \left\lfloor \frac{x}{y} \right\rfloor b)y
\]

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

---

\(^1\)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)
The function `ext-gcd(x, y)` is defined recursively as follows:

```plaintext
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: $d = ay + b(x - \lfloor \frac{x}{y} \rfloor \cdot y)$
```
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: $d = ay + b(x - \lfloor \frac{x}{y} \rfloor \cdot y) \implies d = bx - (a - \lfloor \frac{x}{y} \rfloor b)y$

```plaintext
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:** \( d = ay + b(x - \lfloor \frac{x}{y} \rfloor \cdot y) \implies d = bx - (a - \lfloor \frac{x}{y} \rfloor b)y \)

**Returns** \((d, b, (a - \lfloor \frac{x}{y} \rfloor \cdot b))\).
Wrap-up

Conclusion: Can find multiplicative inverses in $O(n)$ time!
Wrap-up

Conclusion: Can find multiplicative inverses in $O(n)$ time!
Very different from elementary school: try 1, try 2, try 3...
Wrap-up

Conclusion: Can find multiplicative inverses in $O(n)$ time!
Very different from elementary school: try 1, try 2, try 3...
$2^{n/2}$

Internet Security.
Public Key Cryptography: 512 digits.
512 divisions vs. $(1000000000000000000000000000000000000000000)$
5 divisions.
Wrap-up

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 1,000,000,000,000,000?
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.
Wrap-up

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 1,000,000,000,000,000? $\leq 80$ divisions.

versus 1,000,000
Wrap-up

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 1,000,000,000,000,000? $\leq 80$ divisions.
versus 1,000,000
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 1,000,000,000,000,000? $\leq 80$ divisions.
   versus 1,000,000
Internet Security.
Wrap-up

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 1,000,000,000,000,000? $\leq 80$
divisions.
  versus 1,000,000

Internet Security.
Public Key Cryptography: 512 digits.
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 $1,000,000,000,000,000$? $\leq 80$ divisions.

versus $1,000,000$

Internet Security.

Public Key Cryptography: 512 digits.

512 divisions vs.
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.
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.

$(100000000000000000000000000000000000000000000000000000000000000)\,5$ divisions.

Next lecture!