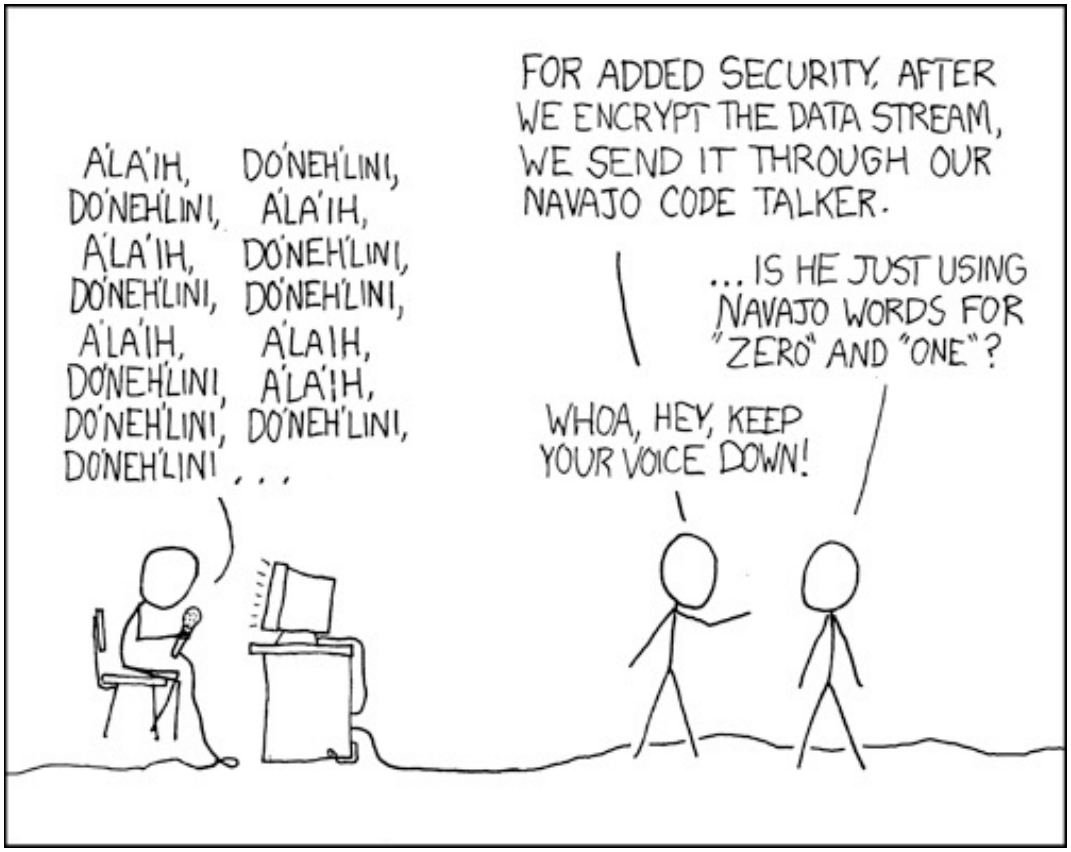
# CSE 311: Foundations of Computing

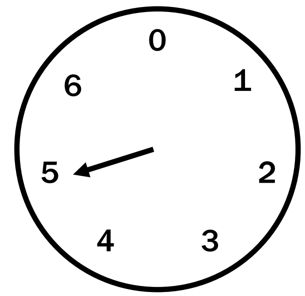
## Lecture 13: Primes, GCD



# Last Time: Modular Arithmetic

**(a + b) mod 7**

**(a × b) mod 7**




**Replace number line with a clock.**
**Taking *m* steps returns to the same place.**

$$x = \underbrace{qm}_{\text{full rotations}} + \overbrace{r}^{\text{where you stop}}$$

$$r = x \bmod m$$

$$q = x \operatorname{div} m$$

**Makes the answers small since** $0 \leq r < m$

**Unclear (so far) that modular arithmetic has the same properties as ordinary arithmetic....**

# Last Time: Modular Arithmetic

**Idea**: Find replacement for "=" that works for modular arithmetic

"=" on ordinary numbers allows us to solve problems, e.g.

- add / subtract numbers from both sides of equations
- substitute "=" values in equations

**Definition: "a is congruent to b modulo m"**

For $a, b, m \in \mathbb{Z}$ with $m > 0$

$$a \equiv_m b \ \leftrightarrow \ m \mid (a - b)$$

Equivalently, $a \equiv_m b$ iff $a = b + km$ for some $k \in \mathbb{Z}$.

# Last Time: Modular Arithmetic

**Definition: "a is congruent to b modulo m"**

For $a, b, m \in \mathbb{Z}$ with $m > 0$

$$a \equiv_m b \leftrightarrow m \mid (a - b)$$

$\boldsymbol{a} \equiv_m \boldsymbol{b}$ if and only if $\boldsymbol{a} \textbf{ mod } \boldsymbol{m} = \boldsymbol{b} \textbf{ mod } \boldsymbol{m}$.

**I.e., $\boldsymbol{a}$ and $\boldsymbol{b}$ are congruent modulo m iff $\boldsymbol{a}$ and $\boldsymbol{b}$ steps stop at the same spot on the "clock" with m numbers**

# Last Time: Modular Arithmetic: Properties

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$ and $\boldsymbol{b} \equiv_m \boldsymbol{c}$, then $\boldsymbol{a} \equiv_m \boldsymbol{c}$

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$ and $\boldsymbol{c} \equiv_m \boldsymbol{d}$, then $\boldsymbol{a} + \boldsymbol{c} \equiv_m \boldsymbol{b} + \boldsymbol{d}$

**Corollary:** If $\boldsymbol{a} \equiv_m \boldsymbol{b}$, then $\boldsymbol{a} + \boldsymbol{c} \equiv_m \boldsymbol{b} + \boldsymbol{c}$

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$ and $\boldsymbol{c} \equiv_m \boldsymbol{d}$, then $\boldsymbol{ac} \equiv_m \boldsymbol{bd}$

**Corollary:** If $\boldsymbol{a} \equiv_m \boldsymbol{b}$, then $\boldsymbol{ac} \equiv_m \boldsymbol{bc}$

# Last Time: Modular Arithmetic: Properties

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$ and $\boldsymbol{b} \equiv_m \boldsymbol{c}$, then $\boldsymbol{a} \equiv_m \boldsymbol{c}$

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$, then $\boldsymbol{a} + \boldsymbol{c} \equiv_m \boldsymbol{b} + \boldsymbol{c}$

If $\boldsymbol{a} \equiv_m \boldsymbol{b}$, then $\boldsymbol{ac} \equiv_m \boldsymbol{bc}$

**"≡" allows us to solve problems in modular arithmetic, e.g.**

- **add / subtract numbers from both sides of equations**
- **chains of "≡" values shows first and last are "≡"**
- **substitute "≡ " values in equations (not *fully* proven yet)**

# Substitution Follows From Other Properties

Given $2y + 3x \equiv_m 25$ and $x \equiv_m 7$,
show that $2y + 21 \equiv_m 25$. (substituting $7$ for $x$)

Start from $x \equiv_m 7$

Multiply both sides $3x \equiv_m 21$

Add to both sides $2\text{y} + 3x \equiv_m 2y + 21$

Combine $\equiv_m$'s $2y + 21 \equiv_m 2\text{y} + 3x \equiv_m 25$

# Basic Applications of mod

- Two's Complement
- Hashing
- Pseudo random number generation

# n-bit Unsigned Integer Representation

- Represent integer $x$ as sum of powers of $2$:

| | | |
|---|---|---|
| 99 | = 64 + 32 + 2 + 1 | = $2^6 + 2^5 + 2^1 + 2^0$ |
| 18 | = 16 + 2 | = $2^4 + 2^1$ |

If $b_{n-1}2^{n-1} + \cdots + b_1 2 + b_0$ with each $b_i \in \{0,1\}$
then binary representation is $b_{n-1} \ldots b_2\ b_1\ b_0$

- For n = 8:

99: 0110 0011
18: 0001 0010

Easy to implement arithmetic $\mathbf{mod}\ \mathbf{2}^n$
... just throw away bits n+1 and up

$2^n \mid 2^{n+k}$ so $b_{n+k}2^{n+k} \equiv_{2^n} 0$
for $k \geq 0$

# n-bit Unsigned Integer Representation

- Largest representable number is $2^n - 1$

$2^n = 100...000$ (n+1 bits)

$2^n - 1 = 011...111$ (n bits)

**Note**: $2^n - 1 = 111...111$

THE WALL STREET JOURNAL.


32 bits
1 = $0.0001
$429,496.7295 max

**Berkshire Hathaway Inc. (BRK-A)**
NYSE - Nasdaq Real Time Price. Currency in USD

**436,401.00** +679.50 (+0.16%)

At close: 4:00PM EDT

# Sign-Magnitude Integer Representation

$n$-bit signed integers

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

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

Any problems with this representation?

# Two's Complement Representation

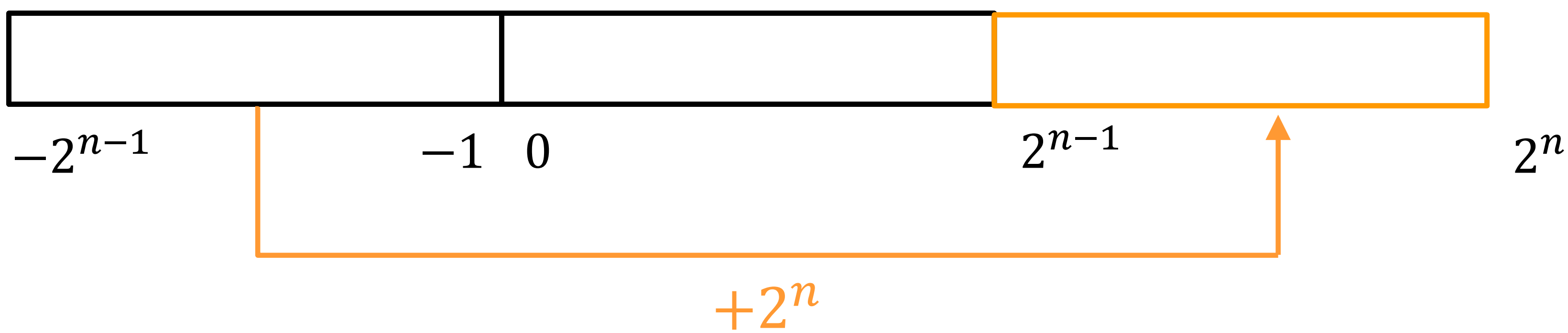
Suppose that $0 \leq x < 2^{n-1}$

$x$ is represented by the binary representation of $x$

Suppose that $-2^{n-1} \leq x < 0$

$x$ is represented by the binary representation of $x + 2^n$

result is in the range $2^{n-1} \leq x < 2^n$




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

# Two's Complement Representation

Suppose that $0 \leq x < 2^{n-1}$

$x$ is represented by the binary representation of $x$

Suppose that $-2^{n-1} \leq x < 0$

$x$ is represented by the binary representation of $x + 2^n$

result is in the range $2^{n-1} \leq x < 2^n$

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

99 = 64 + 32 + 2 + 1

18 = 16 + 2

For n = 8:

99: 0110 0011

-18: 1110 1110 (-18 + 256 = 238)

# Two’s Complement Representation

Suppose that $0 \le x < 2^{n-1}$

$x$ is represented by the binary representation of $x$

Suppose that $-2^{n-1} \le x < 0$

$x$ is represented by the binary representation of $x + 2^n$

result is in the range $2^{n-1} \le x < 2^n$

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

**Key property:** First bit is still the sign bit!

**Key property:** Twos complement representation of any number $\boldsymbol{y}$ is equivalent to $\boldsymbol{y}\ \mathbf{mod}\ \mathbf{2}^{\boldsymbol{n}}$ so arithmetic works $\mathbf{mod}\ \mathbf{2}^{\boldsymbol{n}}$

# Two's Complement Representation

- For $0 < x \leq 2^{n-1}$ , $-x$ is represented by the binary representation of $2^n - x$
  - How do we calculate –x from x?
  - E.g., what happens for "`return -x;`" in Java?

$$2^n - x = (2^n - 1) - \mathrm{x} + 1$$

- 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 $2^n - 1 - x$

    Then add 1 to get $2^n - x$

# Hashing

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

- $\text{hash}(x) = x \bmod p$ for $p$ a prime close to $n$
  - or $\text{hash}(x) = (ax + b) \bmod p$
- Depends on all of the bits of the data
  - helps avoid collisions due to similar values
  - need to manage them if they occur

# Hashing

- $\text{hash}(x) = x \bmod p$ for $p$ a prime close to $n$
- deterministic function with random-ish behavior

- Applications
  - map integer to location in array (hash tables)
  - map user ID or IP address to machine

    requests from the same user / IP address go to the same machine

    requests from different users / IP addresses spread randomly

# 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**

# More Number Theory

## Primes and GCD

# Primality

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

$$p > 1 \wedge \forall \mathrm{x} \in \mathbb{N}\,((x \mid p) \rightarrow ((x = 1) \vee (x = p)))$$

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

$$p > 1 \wedge \exists \mathrm{x} \in \mathbb{N}\,((x \mid p) \wedge (x \neq 1) \wedge (x \neq p))$$

# 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

# Euclid's Theorem

**There are an infinite number of primes.**

Proof by contradiction:

Suppose that there are only a finite number of primes and call the full list $p_1, p_2, \dots, p_n$.

# Euclid's Theorem

**There are an infinite number of primes.**

Proof by contradiction:

Suppose that there are only a finite number of primes and call the full list $p_1, p_2, \ldots, p_n$.

Define the number $P = p_1 \cdot p_2 \cdot p_3 \cdot \cdots \cdot p_n$ and let $Q = P + 1$.

# Euclid's Theorem

There are an infinite number of primes.

Proof by contradiction:

Suppose that there are only a finite number of primes and call the full list $p_1, p_2, \ldots, p_n$.

Define the number $P = p_1 \cdot p_2 \cdot p_3 \cdot \cdots \cdot p_n$ and let $Q = P + 1$. (Note that $Q > 1$.)

Case 1: $Q$ is prime: Then $Q$ is a prime different from all of $p_1, p_2, \ldots, p_n$ since it is bigger than all of them.

Case 2: $Q$ is not prime: Then $Q$ has some prime factor $p$ (which must be in the list). Therefore $p|P$ and $p|Q$ so $p|(Q - P)$ which means that $p|1$.

Both cases are contradictions,
so the assumption is false (proof by cases). ∎

# Famous Algorithmic Problems

- **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]:

1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413

1230186684530117755130494958384962720772853569595334792197322452151726400507263657518745202199786469389956474942774063845925192557326303453731548268507917026122142913461670429214311602221240479274737794080665351419597459856902143413

=

3347807169895689878604416984821269081770479498371376856891243138898288379387800228761471165253174308773781446799948

×

36746043666799590428244633799627952632279158164343087642676032283815739666511279233373417143396810270092798736308917

# Greatest Common Divisor

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) =

$d$ is GCD iff $(d \mid a) \land (d \mid b) \land \forall x \in \mathbb{N}\ (((x \mid a) \land (x \mid b)) \rightarrow (x \leq d))$

# GCD and Factoring

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

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

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

**Factoring is expensive!**

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

# Useful GCD Fact

Let $a$ and $b$ be positive integers.
We have $\gcd(a,b) = \gcd(b, a \bmod b)$

**Proof:**
We will show that every number dividing $a$ and $b$ also divides $b$ and $a \bmod b$.
I.e. $d|a$ and $d|b$ iff $d|b$ and $d|(a \bmod b)$.

Hence, their set of common divisors are the same,
which means that their greatest common divisor is the same.

# Useful GCD Fact

Let *a* and *b* be positive integers.
We have gcd(*a*,*b*) = gcd(*b*, *a* mod *b*)

**Proof:**
By definition of mod, $a = qb + (a \bmod b)$ for some integer $q = a \operatorname{div} b$.

Suppose $d|b$ and $d|(a \bmod b)$.
Then $b = md$ and $(a \bmod b) = nd$ for some integers $m$ and $n$.
Therefore $a = qb + (a \bmod b) = qmd + nd = (qm + n)d$.
So $d|a$.

Suppose $d|a$ and $d|b$.
Then $a = kd$ and $b = jd$ for some integers $k$ and $j$.
Therefore $(a \bmod b) = a - qb = kd - qjd = (k - qj)d$.
So, $d|(a \bmod b)$ also.

Since they have the same common divisors, $\gcd(a, b) = \gcd(b, a \bmod b)$. ∎

# Another simple GCD fact

Let a be a positive integer.
We have gcd($a$,$0$) = $a$.

# Euclid's Algorithm

gcd(a, b) = gcd(b, a mod b)

gcd(a, 0) = a

```
int gcd(int a, int b){ /* Assumes: a >= b, b >= 0 */
    if (b == 0) {
        return a;
    } else {
        return gcd(b, a % b);
    }
}
```

Note: gcd(b, a) = gcd(a, b)

# Euclid's Algorithm

**Repeatedly use $\gcd(a, b) = \gcd(b, a \bmod b)$ to reduce numbers until you get $\gcd(g, 0) = g$.**

gcd(660,126) =

# Euclid's Algorithm

**Repeatedly use $\gcd(a, b) = \gcd(b, a \bmod b)$ to reduce numbers until you get $\gcd(g, 0) = g$.**

gcd(660,126) = gcd(126, 660 mod 126) = gcd(126, 30)
= gcd(30, 126 mod 30) = gcd(30, 6)
= gcd(6, 30 mod 6) = gcd(6, 0)
= 6