The Division Theorem

- **Theorem**  Let $n$ be a fixed integer $\geq 2$. For any $z \in \mathbb{Z}$ we can find unique integers $q, r$ such that

$$z = qn + r \text{ where } 0 \leq r \leq n - 1.$$

- $q$ is called the *quotient* and $r$ the *remainder* modulo $m$.

- Another way to put the Division Theorem is that $q$ is the largest integer such that $qn \leq z$, and $r = z - qn$. 
Examples

• $17 = 3 \cdot 5 + 2; \ q = 3, r = 2.$

• $-39 = (-8) \cdot 5 + 1; \ q = -8, r = 1.$

• If $z = qn + r$ we put $z \mod n = r$ and $z \div n = q.$

• Thus $17 \mod 5 = 2, \ -39 \mod 5 = 1, \ -39 \div 5 = -8.$
Characterizing congruence mod $n$

- **Theorem** For any integers $x$ and $y$, $x \equiv y \pmod{n}$ if and only if $x \mod{n} = y \mod{n}$.

- We prove the $\Rightarrow$ direction.

- Assume that $x \equiv y \pmod{n}$. By definition this means that $x - y = kn$ for some $k \in \mathbb{Z}$. Use the Division Theorem twice to write

  $$x = q_1n + r_1$$
  $$y = q_2n + r_2$$

  where we may as well suppose $r_1 \geq r_2$; otherwise just interchange the role of $x$ and $y$. Therefore, by subtraction,

  $$x - y = (q_1 - q_2)n + (r_1 - r_2)$$

  where $0 \leq r_1 - r_2 < n$.

  But

  $$x - y = kn = kn + 0$$

  so by uniqueness in the Division Theorem, $r_1 - r_2 = 0$, or $r_1 = r_2$ as we wanted.

- The $\Leftarrow$ proof is left to you!
More on congruences modulo $n$

- **Proposition**  If $a \equiv b$ and $c \equiv d \pmod{n}$ then
  
  (1) $a + c \equiv b + d \pmod{n}$ and (2) $ac \equiv bd \pmod{n}$.

- **Proof:** (1) Assume the hypotheses. Write

  $a - b = kn$ for some $k$
  $c - d = ln$ for some $l$

  Then by adding these equations

  $$(a + c) - (b + d) = (k + l)n$$

  which is conclusion (1).

- For (2), we use a trick. Using $a - b = kn$ we multiply both sides by $c$, getting $ac - bc = ckn$. Likewise we multiply the equation $c - d = ln$ by $b$, getting $bc - bd = bkn$. Adding the two derived equations gives us $ac - bd = (ck + bl)n$, which gives us (2).
**Arithmetic modulo** $n$

- Let $\mathbb{Z}_n = \{0, 1, \ldots, n - 1\}$. For $a, b \in \mathbb{Z}_n$ define

  $a \oplus_n b = (a + b) \mod n$

  and

  $a \otimes b = ab \mod n$.

- For example, when $n = 3$

  $\begin{array}{c|c|c|c} \oplus & 0 & 1 & 2 \\ \hline 0 & 0 & 1 & 2 \\ 1 & 1 & 2 & 0 \\ 2 & 2 & 0 & 1 \end{array}$

  $\begin{array}{c|c|c|c} \otimes & 0 & 1 & 2 \\ \hline 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 \\ 2 & 2 & 0 & 1 \end{array}$
Application of modular ideas: Hashing

• Suppose every student has a 10-digit student id number, but there are only 35,000 student records you wish to store in a fixed amount of array space, say in an array with 50,000 lines.

• If you in fact had $10^{10}$ lines in the array, you could use the student id number itself as an index into it. But you don’t.

• In this case, when actual memory is limited, you can perform a function, called a hashing function on the student id numbers to come up with a new index into the limited array. So, you think of the id number as an integer $m$, and hash it using the function

$$h(m) = m \mod 50,000.$$  

This is an easy number to compute. And even though the function $h$ is not one-to-one, we almost never get a collision $h(m) = h(p)$ for $m \neq p$. If we do, there are tricks to store the superfluous index.

• This is what we do when we put your grades on the web using the last 4 digits of your id.
A second application: Pseudo-random numbers

- You can get a computer to produce a really random-looking sequence of numbers. This is useful when you want to simulate a real-life experiment on the machine.

- Such a sequence is called a pseudo-random sequence, and the procedure that produces it is called a pseudo-random number generator.

- Simple pseudo-random number generators can be given using modular arithmetic. We choose a large modulus, often related to word size in memory, like $2^{31} - 1$. Then we choose an integer seed $a_0$, using it as the base case for an inductive definition
  \[ a_{n+1} = (16,807 \cdot a_n) \mod (2^{31} - 1). \]
  (The number 16,807 is carefully chosen here.)

- There are many generators of the form
  \[ a_{n+1} = (b \cdot a_n + c) \mod m \]
  They are called linear congruential generators.
Towards a cryptography application: Fast greatest common divisors

- We now begin studying the number theory we need to understand a basic method for encrypting Internet messages.

- This involves several concepts and algorithms. We begin by studying a very old algorithm, due to Euclid, for finding greatest common divisors.

- You can theoretically do this by factoring the two numbers and taking minimum exponents. But factoring huge numbers is an extremely time-consuming process, and nobody knows how to do it in a way that can be implemented at all.

- Euclid’s gcd algorithm is both simple and fast!
Euclid’s GCD algorithm

function gcd(m:N^[+]; n:N); %(gcd(m, 0) = m)
{
    a := m;
    b := n;
    while b != 0 do % gcd(a, b) = gcd(m, n)
        {r := a mod b;
         a := b;
         b := r;}
    gcd(m, n) := a
}

Example: gcd(91, 287).

\[
\begin{array}{ccccccc}
  a & 91 & 287 & 91 & 14 & 7 \\
  b & 287 & 91 & 14 & 7 & 0 \\
  r & ? & 91 & 14 & 7 & 0 \\
\end{array}
\]

The gcd is \( a = 7 \).
Why does this work?

• Rewrite the program a little more compactly as

```plaintext
function gcd(m: \mathbb{N}^+; n: \mathbb{N});
{
    (a, b) := (m, n);
    while b != 0 do % gcd(a, b) = gcd(m, n)
        (a,b) := (b, a mod b);
    gcd(m,n) := a
}
```

• Lemma  For any \(x, y\):

\[
gcd(x, y) = gcd(y, x \mod y).
\]

• This means that the statement in the comment at the head of the while-loop is always true no matter how many times around the loop you go. So when you come out of the loop, \(a = gcd(m, n)\).

• We prove the lemma on the next slide.
Proving the lemma

*Proof:* We show that the set $lb(x, y) = lb(y, x \mod y)$, where $lb(x, y)$ is the set of common divisors of $x$ and $y$, i.e., lower bounds of $x$ and $y$ in the $|$ ordering. It follows that the two numbers $x$ and $y$ have the same greatest common divisor as $y$ and $x \mod y$.

To show $lb(x, y) \subseteq lb(y, x \mod y)$ let $k \in lb(x, y)$. Then $k \mid x$ and $k \mid y$. By the Division Theorem, $x = yq + r$, so that $r = x - yq$. Since $k \mid y$, $k \mid yq$. But $k \mid x$ so that $k \mid x - yq = r = x \mod y$. Thus $k \in lb(y, x \mod y)$.

Conversely, let $k \mid y$ and $k \mid r = x - yq$. Then $(x - yq) = ck$ and $y = dk$ for some $c$ and $d$. Therefore

$$x = ck + yq = ck + dkq = k(c + dq)$$

so that $x$ is a multiple of $k$, or $k \in lb(x, y)$ as desired. □
How fast is Euclid’s algorithm?

• It really only depends on the number $n$ we give it, because in the very first time through the loop, it computes a remainder modulo $n$.

• We’ll measure the time it takes, using the number of times the loop is executed as our measure of “time”. (We’re really deriving an $O$-estimate.)

• We’ll see that Euclid’s algorithm is exponentially faster than simple factoring using, say, factor trees as in grade school.

• The running time is (amazingly) intimately related to the Fibonacci numbers

$$f_0 = 0, f_1 = 1, f_2 = 1, f_3 = 2, \ldots, f_{k+1} = f_k + f_{k-1}.$$
Fibonacci and Euclid – first encounter.

**Proposition**  For any $n \geq 1$, Euclid’s algorithm takes $n-1$ trips through the loop to compute $\gcd(f_n, f_{n-1})$.

**Proof:** By induction on $n$. First let’s review the algorithm:

```plaintext
function gcd(m:N^+; n:N);
{
(a, b) := (m, n);
while b != 0 do % \(\gcd(a, b) = \gcd(m, n)\)
(a,b) := (b, a mod b);
gcd(m,n) := a
}
```

* Basis: $n = 1$. To compute $\gcd(f_1, f_0) = \gcd(1, 0) = 1$ we go $0 = n-1$ times through.

* Induction step: Assume that we go through the loop $k-1$ times to compute $\gcd(f_k, f_{k-1})$. To compute $\gcd(f_{k+1}, f_k)$ we compute $\gcd(f_k, f_{k+1} \mod f_k)$. But

$$f_{k+1} = f_k + f_{k-1} = 1 \cdot f_k + f_{k-1},$$

so by the Division Theorem, $f_{k+1} \mod f_k = f_{k-1}$. By induction hypothesis it takes $k-1$ times through the loop for this, then one more trip for a total of $k$ as we wanted.

\[\square\]
Theorem (Lamé). For any $k \geq 1$, if Euclid’s algorithm takes $k$ trips to compute $\gcd(m, n)$, where $m \geq n$, then $n \geq f_{k+1}$.

Proof. By strong induction on $k$.

Basis: $k = 1$. If we went through the loop once then certainly $n \geq 1 = f_2$. And when $k = 2$ we went through the loop twice, so $n > 1$, and thus $n \geq 2 = f_3$.

Induction step: Assume for all integers $\leq k$ that if we go through the loop $k$ times, then $n \geq f_{k+1}$. We must prove the same statement with $k$ replaced by $k + 1$. Suppose that it takes $k + 1$ trips to compute $\gcd(m, n)$. Write out the first two trips

\[
\begin{align*}
\gcd(m, n) &= \gcd(n, m \mod n) \\
&= \gcd(m \mod n, n \mod (m \mod n))
\end{align*}
\]
Continuing the proof

By induction hypothesis

\[ m \mod n \geq f_k \]

and

\[ n \mod (m \mod n) \geq f_{k-1}. \]

We can simplify this using the Division Theorem: \( m = q_1 n + r_1 \) and \( n = q_2 r_1 + r_2 \), where \( r_1 = m \mod n \) and

\[ r_2 = n \mod r_1 = n \mod (m \mod n). \]

Note that \( r_1 = m \mod n < n \) so that \( q_2 > 0 \). By IH, \( r_1 \geq f_k \) and \( r_2 \geq f_{k-1} \). Therefore

\[
\begin{align*}
    n &= q_2 r_2 + r_1 \\
    &\geq r_2 + r_1 \quad \text{(because } q_2 > 0) \\
    &\geq f_{k-1} + f_k \quad \text{by the two induction hypotheses} \\
    &= f_{k+1} \quad \text{by the inductive definition of Fibonacci.}
\end{align*}
\]
Towards an $O$-estimate for Euclid

- Lamé’s theorem restated: For any $k, n, m$ such that $k \geq 1$ and $m \geq n$, if it takes Euclid $k$ steps to compute $\gcd(m, n)$ then $n \geq f_{k+1}$. (A “step” is an iteration of the loop.)

- This is logically equivalent to saying that if $n < f_{k+1}$, then it takes at most $k - 1$ steps to compute $\gcd(m, n)$.

- In our first example of strong induction proofs, we showed

  $$(\forall k \geq 2)(f_k > \alpha^{k-2})$$

  where $\alpha = (1 + \sqrt{5})/2$.

- So if $n \leq \alpha^{k-1}$, then $n < f_{k+1}$ and so it takes at most $k - 1$ steps to compute $\gcd(m, n)$. 
Finishing the $O$-estimate for Euclid

- We may restate the conclusion on the last slide as saying that for any $k$, $m$, and $n$, if $n \leq \alpha^k$, then it takes at most $k$ steps to compute $gcd(m, n)$.

- We know

$$ n \leq \alpha^k \iff \log_{\alpha} n \leq k. $$

- Let $k = \lceil \log_{\alpha} n \rceil$, so that $\log_{\alpha} n < k < \log_{\alpha} n + 1$.

- Then it takes at most $\log_{\alpha} n + 1$ steps to compute $gcd(m, n)$.

- Since $\log_{\alpha} n = \log_{10} n \cdot \log_{\alpha} 10$, this gives us an $O(\log_{10} n)$ algorithm for the gcd. This is proportional to the number of decimal digits in $n$.

- Compare this with the time it takes to factor an 800-digit number.