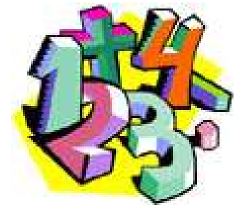

CSCE 222
Discrete Structures
Number Theory

Dr. Tim McGuire

Grateful acknowledgement to Professor Bart Selman, Cornell University, and Prof. Johnnie Baker, Kent State, for some of the material upon which these notes are adapted.

Based on Chapter 4 of Rosen
Discrete Mathematics and its Applications

Introduction to Number Theory

- Number theory is about **integers** and their properties.
- We will start with the basic principles of
- **divisibility,**
- **greatest common divisors,**
- **least common multiples, and**
- **modular arithmetic**
- and look at some relevant algorithms.

Division (§ 4.1.2)

If a and b are integers with $a \neq 0$, we say that a **divides** b if there is an integer c so that $b = ac$.

When a divides b we say that a is a **factor** of b and that b is a **multiple** of a.

The notation $\mathbf{a \mid b}$ means that a divides b.

We write $\mathbf{a \nmid b}$ when a does not divide b

Divisibility Theorems

For integers a, b, and c it is true that

if $a \mid b$ and $a \mid c$, then $a \mid (b + c)$
Example: $3 \mid 6$ and $3 \mid 9$, so $3 \mid 15$.

if $a \mid b$, then $a \mid bc$ for all integers c
Example: $5 \mid 10$, so $5 \mid 20$, $5 \mid 30$, $5 \mid 40$, …

if $a \mid b$ and $b \mid c$, then $a \mid c$
Example: $4 \mid 8$ and $8 \mid 24$, so $4 \mid 24$.

Primes (§ 4.3)

A positive integer p greater than 1 is called ***prime*** if the only positive factors of p are 1 and p.
Note: 1 is not a prime

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

The fundamental theorem of arithmetic:

Every positive integer can be written **uniquely** as the **product of primes**, where the prime factors are written in order of increasing size.

Also called the unique-prime-factorization theorem

Primes

- **Examples**:

15 =	$3 \cdot 5$
48 =	$2 \cdot 2 \cdot 2 \cdot 2 \cdot 3 = 2^4 \cdot 3$
17 =	17
100 =	$2 \cdot 2 \cdot 5 \cdot 5 = 2^2 \cdot 5^2$
512 =	$2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 \cdot 2 = 2^9$
515 =	$5 \cdot 103$
28 =	$2 \cdot 2 \cdot 7$

Primes

If *n* is a composite integer, then *n* has a prime divisor less than or equal $\sqrt{n}$.

This is easy to see: if n is a composite integer, it must have at least two prime divisors. Let the largest two be p_1 and p_2. Then $p_1 \cdot p_2 <= n$.

p_1 and p_2 cannot both be greater than $\sqrt{n}$, because then $p_1 \cdot p_2 > n$.

If the smaller number of p_1 and p_2 is not a prime itself, then it can be broken up into prime factors that are smaller than itself but ≥ 2.

The Division Algorithm (§ 4.1.3)

Let a be an integer and d a **positive** integer.
Then there are unique integers q and r, with $0 \le r < d$, such that
$a = dq + r$.

In the above equation,

- d is called the *divisor*,
- a is called the *dividend*,
- q is called the *quotient*, and
- r is called the *remainder*.

The Division Algorithm

Example:

When we divide 17 by 5, we have

$17 = 5 \cdot 3 + 2.$

17 is the dividend,
5 is the divisor,
3 is the quotient, and
2 is the remainder.

In C and Java

```
17 / 5 = 3
17 % 5 = 2
```

The Division Algorithm

Another example:

What happens when we divide -11 by 3 ?

Note that the remainder cannot be negative.

$-11 = 3\cdot(-4) + 1.$

-11 is the dividend,
3 is the divisor,
-4 is called the quotient, and
1 is called the remainder.

Note: This differs from C and Java

```
-11 / 3 = -3
-11 % 3 = -2
```

Greatest Common Divisors (§ 4.3.6)

Let a and b be integers, not both zero.
The largest integer d such that d | a and d | b is called the **greatest common divisor** of a and b.

The greatest common divisor of a and b is denoted by gcd(a, b).

Example 1: What is gcd(48, 72) ?

The positive common divisors of 48 and 72 are 1, 2, 3, 4, 6, 8, 12, 16, and 24, so gcd(48, 72) = 24.

Example 2: What is gcd(19, 72) ?

The only positive common divisor of 19 and 72 is 1, so gcd(19, 72) = 1.

Greatest Common Divisors

Using prime factorizations:

$a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$, $b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$, where $p_1 < p_2 < \ldots < p_n$ and $a_i, b_i \in \mathbf{N}$ for $1 \leq i \leq n$

$\gcd(a, b) = p_1^{\min(a_1, b_1)} p_2^{\min(a_2, b_2)} \ldots p_n^{\min(a_n, b_n)}$

Example:

$a = 60 = \quad 2^2\ 3^1\ 5^1$

$b = 54 = \quad 2^1\ 3^3\ 5^0$

$\gcd(a, b) = \quad 2^1\ 3^1\ 5^0 = 6$

Relatively Prime Integers

Definition:

Two integers a and b are **relatively prime** if $\gcd(a, b) = 1$.

Examples:

Are 15 and 28 relatively prime?

Yes, $\gcd(15, 28) = 1$.

Are 55 and 28 relatively prime?

Yes, $\gcd(55, 28) = 1$.

Are 35 and 28 relatively prime?

No, $\gcd(35, 28) = 7$.

Relatively Prime Integers

Definition:

The integers $a_1, a_2, \ldots, a_n$ are **pairwise relatively prime** if $\gcd(a_i, a_j) = 1$ whenever $1 \leq i < j \leq n$.

Examples:

Are 15, 17, and 27 pairwise relatively prime?

No, because gcd(15, 27) = 3.

Are 15, 17, and 28 pairwise relatively prime?

Yes, because gcd(15, 17) = 1, gcd(15, 28) = 1 and gcd(17, 28) = 1.

Least Common Multiples

Definition:

The **least common multiple** of the positive integers *a* and *b* is the smallest positive integer that is divisible by both *a* and *b*.

We denote the least common multiple of *a* and b by lcm(*a, b*).

Examples:

lcm(3, 7) = 21

lcm(4, 6) = 12

lcm(5, 10) = 10

Least Common Multiples

Using prime factorizations:

$a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}$, $b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n}$, where $p_1 < p_2 < \ldots < p_n$ and $a_i, b_i \in \mathbf{N}$ for $1 \le i \le n$

$\text{lcm}(a, b) = p_1^{\max(a_1, b_1)} p_2^{\max(a_2, b_2)} \ldots p_n^{\max(a_n, b_n)}$

Example:

$a = 60 = 2^2\, 3^1\, 5^1$

$b = 54 = 2^1\, 3^3\, 5^0$

$\text{lcm}(a, b) = 2^2\, 3^3\, 5^1 = 4 \cdot 27 \cdot 5 = 540$

GCD and LCM

$a = 60 = 2^2\, 3^1\, 5^1$

$b = 54 = 2^1\, 3^3\, 5^0$

$\gcd(a, b) = 2^1\, 3^1\, 5^0 = 6$

$\text{lcm}(a, b) = 2^2\, 3^3\, 5^1 = 540$

Theorem: $a \cdot b = \gcd(a,b) \cdot \text{lcm}(a,b)$

Modular Arithmetic (§ 4.1.4)

Let a be an integer and m be a positive integer.
We denote by ***a* mod *m*** the remainder when a is divided by m.

Examples:

9 mod 4 = 1

9 mod 3 = 0

9 mod 10 = 9

-13 mod 4 = 3

Congruences (§ 4.4)

Let a and b be integers and m be a positive integer. We say that **a is congruent to b modulo m** if m divides a – b.

We use the notation **a ≡ b (mod m)** to indicate that a is congruent to b modulo m.

In other words:
a ≡ b (mod m) if and only if **a mod m = b mod m.**

Congruences

Examples:

Is it true that $46 \equiv 68 \pmod{11}$?

Yes, because 11 | (46 – 68).

Is it true that $46 \equiv 68 \pmod{22}$?

Yes, because 22 | (46 – 68).

For which integers z is it true that $z \equiv 12 \pmod{10}$?

It is true for any z∈{…,-28, -18, -8, 2, 12, 22, 32, …}

Theorem: Let *m* be a positive integer. The integers *a* and *b* are congruent modulo *m* if and only if there is an integer *k* such that $a = b + km$.

The Euclidean Algorithm (§ 4.3.7)

The **Euclidean Algorithm** finds the **greatest common divisor** of two integers a and b.

For example, if we want to find gcd(287, 91), we **divide** 287 by 91:

$287 = 91 \cdot 3 + 14$

⇨ $287 - 91 \cdot 3 = 14$

⇨ $287 + 91 \cdot (-3) = 14$

We know that for integers a, b and c,

if a | b, then a | bc for all integers c.

Therefore, any divisor of 91 is also a divisor of $91 \cdot (-3)$.

The Euclidean Algorithm

$287 + 91 \cdot (-3) = 14$

We also know that for integers a, b and c,
if a | b and a | c, then a | (b + c).

Therefore, any divisor of 287 and 91 must also be a divisor of $287 + 91 \cdot (-3)$, which is 14.

Consequently, the greatest common divisor of **287 and 91** must be the same as the greatest common divisor of **14 and 91**:

gcd(287, 91) = gcd(14, 91).

The Euclidean Algorithm

In the next step, we divide 91 by 14:

$91 = 14 \cdot 6 + 7$

This means that gcd(14, 91) = gcd(14, 7).

So we divide 14 by 7:

$14 = 7 \cdot 2 + 0$

We find that 7 | 14, and thus gcd(14, 7) = 7.

Therefore, gcd(287, 91) = 7.

The Euclidean Algorithm

In **pseudocode**, the algorithm can be implemented as follows:

procedure gcd(a, b: positive integers)
x := a
y := b
while y ≠ 0 **do**
    r := x **mod** y
    x := y
    y := r
endwhile **{x is gcd(a, b)}**

Representations of Integers (§ 4.2.2)

Let b be a positive integer greater than 1.
Then if n is a positive integer, it can be expressed **uniquely** in the form:

$$n = a_k b^k + a_{k-1} b^{k-1} + \ldots + a_1 b + a_0,$$

where k is a nonnegative integer, $a_0, a_1, \ldots, a_k$ are nonnegative integers less than b, and $a_k \neq 0$.

Example for b=10:

$859 = 8 \cdot 10^2 + 5 \cdot 10^1 + 9 \cdot 10^0$

Representations of Integers

Example for b=2 (binary expansion):

$(10110)_2 = 1\cdot2^4 + 1\cdot2^2 + 1\cdot2^1 = (22)_{10}$

Example for b=16 (hexadecimal expansion):

(we use letters A to F to indicate numbers 10 to 15)

$(3A0F)_{16} = 3\cdot16^3 + 10\cdot16^2 + 15\cdot16^0 = (14863)_{10}$

Representations of Integers

How can we construct the base b expansion of an integer n?

First, divide n by b to obtain a quotient q_0 and remainder a_0, that is,

$n = bq_0 + a_0$, where $0 \le a_0 < b$.

The remainder a_0 is the rightmost digit in the base b expansion of n.

Next, divide q_0 by b to obtain:

$q_0 = bq_1 + a_1$, where $0 \le a_1 < b$.

a_1 is the second digit from the right in the base b expansion of n. Continue this process until you obtain a quotient equal to zero.

Representations of Integers

Example:

What is the base 8 expansion of $(12345)_{10}$?

First, divide 12345 by 8:

$12345 = 8 \cdot 1543 + 1$

$1543 = 8 \cdot 192 + 7$

$192 = 8 \cdot 24 + 0$

$24 = 8 \cdot 3 + 0$

$3 = 8 \cdot 0 + 3$

The result is: $(12345)_{10} = (30071)_8$

Representations of Integers

- Exercises
  - Convert 3456_{10} to binary (base-2)
  - Convert 640_{10} to hexadecimal (base-16)
  - Convert 25_{10} to octal (base-8)

Conversions Between Hex and Octal

- When converting from one binary compatible base to another, it is easiest to go through binary as an intermediate step
- Hexadecimal to octal (go through binary)

  $\mathtt{3F74_{16} = 0011\ 1111\ 0111\ 0100}$

  $\mathtt{= 0\ 011\ 111\ 101\ 110\ 100 = 037564_{8}}$

Exercise:

- Convert the following ***octal*** numbers to ***hexadecimal***:
  - 12, 5655, 2550276, 76545336, 3726755

Representations of Integers

procedure base_b_expansion(n, b: positive integers)

q := n
k := 0
while $q \neq 0$ **do**
  a_k := q mod b
  $q := \lfloor q/b \rfloor$
  k := k + 1
endwhile
{the base b expansion of n is $(a_{k-1} \ldots a_1 a_0)_b$ }

Addition of Integers

How do we (humans) add two integers?

Example:

```
  1 11     carry
    7583
 +  4932
 -------
   12515
```

Binary expansions:

$$\begin{array}{r} 1\ 1 \text{ carry} \\ (1011)_2 \\ +\ (1010)_2 \\ \hline (\ 10101\ )_2 \end{array}$$

base 16

```
  5B39
 +7AF4
  D62D
```

base 2

- easier than hex because the addition table is so small

```
 100101111
+000110110
 101100101
```

Addition of Integers

Let $a = (a_{n-1}a_{n-2}\ldots a_1a_0)_2$, $b = (b_{n-1}b_{n-2}\ldots b_1b_0)_2$.

How can we **algorithmically** add these two binary numbers?

First, add their rightmost bits:

$a_0 + b_0 = c_0 \cdot 2 + s_0$,

where s_0 is the **rightmost bit** in the binary expansion of a + b, and c_0 is the **carry**.

Then, add the next pair of bits and the carry:

$a_1 + b_1 + c_0 = c_1 \cdot 2 + s_1$,

where s_1 is the **next bit** in the binary expansion of a + b, and c_1 is the carry.

Addition of Integers

Continue this process until you obtain c_{n-1}.

The leading bit of the sum is $s_n = c_{n-1}$.

The result is:

$a + b = (s_ns_{n-1}\ldots s_1s_0)_2$

Addition of Integers

Example:

Add $a = (1110)_2$ and $b = (1011)_2$.

$a_0 + b_0 = 0 + 1 = 0 \cdot 2 + 1$, so that $c_0 = 0$ and $s_0 = 1$.

$a_1 + b_1 + c_0 = 1 + 1 + 0 = 1 \cdot 2 + 0$, so $c_1 = 1$ and $s_1 = 0$.

$a_2 + b_2 + c_1 = 1 + 0 + 1 = 1 \cdot 2 + 0$, so $c_2 = 1$ and $s_2 = 0$.

$a_3 + b_3 + c_2 = 1 + 1 + 1 = 1 \cdot 2 + 1$, so $c_3 = 1$ and $s_3 = 1$.

$s_4 = c_3 = 1$.

Therefore, $s = a + b = (11001)_2$.

Addition of Integers

procedure add(a, b: positive integers)
c := 0
for j := 0 **to** n-1 **do**
    $d := \lfloor (a_j + b_j + c)/2 \rfloor$
    $s_j := a_j + b_j + c - 2d$
    c := d
endwhile
$s_n := c$

- **{the binary expansion of the sum is $(s_n s_{n-1} \ldots s_1 s_0)_2$}**

Two's Complement Arithmetic

- So far, the numbers we have discussed have had no size restriction on them
- However, since computer arithmetic is performed in registers, this will restrict the size of the numbers
  - On some machines, this is 8 bits, others 16 bits, and others it is 32 or even 64 bits
  - To keep it simple, we'll use 4 bits at first

The Fabulous Four-bit Machine (FFM)

- The possible numbers it can hold are:
  `0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111,`
  `1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111`
- In order to identify particular bits in a byte or word, we use the following terms:
  - lsb -- least significant bit -- rightmost bit position
    - always numbered as bit 0
  - msb -- most significant bit -- leftmost bit position
    - on FFM it is bit 3

Unsigned Integers

- An unsigned integer is one which is never negative (addresses of memory locations, ASCII codes, counters, etc.)
- On the FFM, then, we can represent numbers from 0 to 15 (0000b to 1111b)
  - On the Intel 8086 the range of integers is 0 to 2^{16}-1 (0 to 65535)
- If the lsb is 0, the number is even; if it is 1, the number is odd

41

Signed Integers

- How do we represent negative numbers?
- We have three possible methods:
  - Sign and magnitude
  - One's complement
  - Two's complement

42

Sign and Magnitude

- The **msb** of the number represents the sign of the number
- 0 means positive, 1 means negative
- On FFM
  - **`0000`** to **`0111`** represent 0 - 7
  - **`1001`** to **`1111`** represent -1 to -7
  - **`1000`** is not used (negative **`0`**)
- Easy to understand, but doesn't work well in computer circuits (too many special cases)

One's Complement

- The ***one's complement*** of an integer is obtained by complementing each bit
  - The one's complement of 5 (**`0101b`**) is **`1010b`** = -5
  - The one's complement of 0 (**`0000b`**) is **`1111b`** = -0 (*here we go again*)
- - 5 + 5 = **`0101`** + **`1010`** = **`1111`** = **`-0`**
- The negative 0 problem can be solved by using ***two's complement***

Two's Complement

- To get the two's complement of an integer, just add 1 to its one's complement
- The two's complement of 5 (**`0101`**) is **`1010+1`** = **`1011`**
- When we add 5 and -5 we get

```
  0101
  1011
 -----
 10000
```

- Because the FFM can only hold 4 bits, the 1 carried out from the msb is lost and the 4-bit result is 0

45

Complementing the Complements

- It should be obvious that taking the 1's complement of a number twice will give the original number (-(-5) = 5)
- If the 2's complement is to be useful, it must have the same property
- 5 = **`0101`**, -5 = **`1011`**, -(-5) = **`0100+1`** = **`0101`**

46

Moving past 4 bits

- Everything true of the FFM is still true for 8, 16, 32, or even 64 bit machines
- Example: Show how the base-10 integer -97 would be represented in (a) 8 bits and (b) in 16 bits, expressing the answer in hex
- (a)
  - `97 = 6x16+1 = 61h = 0110 0001b`
  - `-97 = 1001 1110 + 1 = 1001 1111b = 9Fh`
- (b)
  - `97 = 0000 0000 0110 0001b`
  - `-97 = 1111 1111 1001 1111b = FF9Fh`

47

Decimal Interpretation

- We have seen how signed and unsigned decimal integers may be represented in the computer
- The reverse problem is how to interpret the contents as a signed or unsigned integer
  - Unsigned is relatively straightforward -- just do a hex to decimal conversion
  - Signed is more difficult -- if the msb is 0, the number is positive, and the conversion is the same as unsigned
  - If the msb is 1, the number is negative -- to find its value, takes the two's complement, convert to decimal, and prefix a minus sign

48

Signed and Unsigned Interpretations in 16 bits

Hex	Unsigned decimal	Signed decimal
0000	0	0
0001	1	1
0002	2	2
. . .	. . .	. . .
0009	9	9
000A	10	10
. . .	. . .	. . .
7FFE	32766	32766
7FFF	32767	32767
8000	32768	-32768
8001	32769	-32767
. . .	. . .	. . .
FFFE	65534	-2
FFFF	65535	-1

49

Example

- Suppose the 8086 **AX** register contains FE0Ch
  - The unsigned decimal interpretation is:
    - 65036
  - To find the signed interpretation:
    - FE0Ch = `1111 1110 0000 1100`
    - 1's complement = `0000 0001 1111 0011`

```
                     +1
---------------------------
0000 0001 1111 0100
= 01F4h = 500
```

Thus, **AX** contains -500

50

Character Representation

- Not all data are treated as numbers
- However they must be coded as binary numbers in order to be processed
- ASCII (American Standard Code for Information Interchange) is the standard encoding scheme used to represent characters in binary format on personal computers

ASCII Code

- 7-bit encoding => 128 characters can be represented
  - codes 0-31 and 127 are control characters (nonprinting characters)
  - control characters used on PC's are: LF, CR, BS, Bell, HT, FF

Applications of Congruences (§ 4.5)

1. Hashing Functions (§ 4.5.1)
2. Pseudorandom Numbers (§ 4.5.2)
3. Cryptography (Caesar Cipher) (§ 4.6.2)

OTHER APPLICATIONS OF NUMBER THEORY IN COMPUTER SCIENCE

1. Hashing Functions

Assignment of memory location to a student record

$$h(k) = k \textbf{ mod } m$$

Key: social security #

# of available memory location

Example: h (064212848) = 064212848 mod 111 = 14 when m = 111

2. Pseudorandom Numbers

- Needed for computer simulation
- Linear congruential method : $x_{n+1} = (ax_n + c) \textbf{ mod } m$
- Put them between 0 and 1 as: $y_n = x_n/m$

3. Cryptography (Caesar Cipher)

a) Encryption:

- Making messages secrets by shifting each letter three letters forward in the alphabet

  B → E X → A

  A = 0
  B =1
  …
  Z=25

- Mathematical expression:

$$f(p) = (p + 3) \ \mathbf{mod}\ 26 \qquad 0 \le p \le 25$$

- Example: What is the secret message produced from the message "Meet you in the park"

Solution:

1. Replace letters with numbers:
   meet = 12 4 4 19
   you = 24 14 20
   in = 8 1 3
   the = 19 7 4
   park = 15 0 17 10
2. Replace each of these numbers p by $f(p) = (p + 3)$ **mod** 26
   meet = 15 7 7 22
   you = 1 17 23
   in = 11 16
   the = 22 10 7
   park = 18 3 20 13
3. Translate back into letters: "PHHW BRX LQ WKH SDUN"

b) Decryption (Deciphering)

$f(p) = (p + k)$ **mod** 26 (shift cipher)
$\Rightarrow f^{-1}(p) = (p - k)$ **mod** 26

Caesar's method and shift cipher are very vulnerable and thus have low level of security (reason frequency of occurrence of letters in the message)
$\Rightarrow$ Replace letters with blocks of letters.

This topic is optional and we may omit it in the interest of time.

PUBLIC KEY CRYPTOGRAPHY

About this module section…

- This is just an introduction to Public Key Cryptography and RSA Encryption
- Much of the underlying theory we will not be able to get to
  - It's beyond the scope of this course
- Much of why this all works won't be taught
  - It's just an introduction to how it works

61

Private key cryptography

- The function and/or key to encrypt/decrypt is a secret
  - (Hopefully) only known to the sender and recipient
- The same key encrypts and decrypts
- How do you get the key to the recipient?

62

Public key cryptography

- Everybody has a key that encrypts and a *separate* key that decrypts
  - They are not interchangable!
- The encryption key is made public
- The decryption key is kept private

Public key cryptography goals

- Key generation should be relatively easy
- Encryption should be easy
- Decryption should be easy
  - With the right key!
- Cracking should be *very* hard

Is that number prime?

- Use the Fermat primality test
- Given:
  - n: the number to test for primality
  - k: the number of times to test (the certainty)
- The algorithm is:

```
repeat k times:
   pick a randomly in the range [1, n–1]
   if a^(n–1) mod n ≠ 1 then return composite
return probably prime
```

65

Is that number prime?

- The algorithm is:

```
repeat k times:
   pick a randomly in the range [1, n–1]
   if a^(n–1) mod n ≠ 1 then return composite
return probably prime
```

- Let $n = 105$
  - Iteration 1: $a = 92$: $92^{104} \bmod 105 = 1$
  - Iteration 2: $a = 84$: $84^{104} \bmod 105 = 21$
  - Therefore, 105 is composite

66

Is that number prime?

- The algorithm is:

```
repeat k times:
   pick a randomly in the range [1, n-1]
   if a^(n-1) mod n ≠ 1 then return composite
return probably prime
```

- Let $n = 101$
  - Iteration 1: $a = 55$: $55^{100} \bmod 100 = 1$
  - Iteration 2: $a = 60$: $60^{100} \bmod 100 = 1$
  - Iteration 3: $a = 14$: $14^{100} \bmod 100 = 1$
  - Iteration 4: $a = 73$: $73^{100} \bmod 100 = 1$
  - At this point, 101 has a $(½)^4 = 1/16$ chance of still being composite

67

More on the Fermat primality test

- Each iteration halves the probability that the number is a composite
  - Probability = $(½)^k$
  - If $k = 100$, probability it's a composite is $(½)^{100}$ = 1 in 1.2×10^{30} that the number is composite
    - Greater chance of having a hardware error!
  - Thus, $k = 100$ is a good value
- However, this is not certain!
  - There are known numbers that are composite but will always report prime by this test
- Source: **http://en.wikipedia.org/wiki/Fermat_primality_test**

68

RSA

- Stands for the inventors: Ron Rivest, Adi Shamir and Len Adleman

- Three parts:
  - Key generation
  - Encrypting a message
  - Decrypting a message

69

Key generation steps

1. Choose two *random* large prime numbers $p \neq q$, and $n = p*q$

2. Choose an integer $1 < e < n$ which is relatively prime to $(p-1)(q-1)$

3. Compute d such that $d * e \equiv 1 \pmod{(p-1)(q-1)}$
   - Rephrased: d*e mod $(p-1)(q-1) = 1$

4. Destroy all records of p and q

70

Key generation, step 1

- Choose two *random* large prime numbers $p \neq q$
  - In reality, 2048 bit numbers are recommended
    - That's ≈ 617 digits
  - From previous slide: chance of a random odd 2048 bit number being prime is about 1/710
    - We can compute if a number is prime relatively quickly via the Fermat primality test
- We choose $p = 107$ and $q = 97$
- Compute $n = p*q$
  - $n = 10379$

71

Key generation, step 1

- Java code to find a big prime number:

```
BigInteger prime = new BigInteger
    (numBits, certainty, random);
```

The number of bits of the prime

Certainty that the number is a prime

The random number generator

72

Key generation, step 1

- Java code to find a big prime number:

```
import java.math.*;
import java.util.*;

class BigPrime {

   static int numDigits = 617;
   static int certainty = 100;

   static final double LOG_2 = Math.log(10)/Math.log(2);
   static int numBits = (int) (numDigits * LOG_2);

   public static void main (String args[]) {
        Random random = new Random();
        BigInteger prime = new BigInteger (numBits, certainty,
                                           random);
        System.out.println (prime);
   }
}
```


Key generation, step 1

- How long does this take?
  - These tests done on a 2.6GHz Core i7 8^{th} generation (i.e., fast)
  - Average of 100 trials (certainty = 100)

  - 200 digits (664 bits): about 0.1 seconds
  - 617 digits (2048 bits): about 3.75 seconds
  - 1234 digits (4096 bits): about 15 seconds

Key generation, step 1

- Practical considerations
  - p and q should not be too close together
  - (p-1) and (q-1) should not have small prime factors
  - Use a good random number generator

75

Key generation, step 2

- Choose an integer $1 < e < n$ which is relatively prime to (p-1)(q-1)
- There are algorithms to do this efficiently
  - We aren't going over them in this course
- Easy way to do this: make e be a prime number
  - It only has to be relatively prime to (p-1)(q-1), but can be fully prime

76

Key generation, step 2

- Recall that p = 107 and q = 97
  - (p-1)(q-1) = 106*96 = 10176 = 2^6*3*53
- We choose e = 85
  - 85 = 5*17
  - gcd (85, 10176) = 1
  - Thus, 85 and 10176 are relatively prime

77

Key generation, step 3

- Compute d such that:

$$d * e \equiv 1 \pmod{(p-1)(q-1)}$$

  - Rephrased: d*e mod (p-1)(q-1) = 1
- There are algorithms to do this efficiently
  - We aren't going over them in this course
- We choose d = 4669
  - 4669*85 mod 10176 = 1

78

Key generation, step 3

- Java code to find d:

```
import java.math.*;

class FindD {
  public static void main (String args[]) {

        BigInteger pq = new BigInteger("10176");
        BigInteger e = new BigInteger ("85");

        System.out.println (e.modInverse(pq));
   }
}
```

- Result: 4669

79

Key generation, step 4

- Destroy all records of *p* and *q*

- If we know *p* and *q*, then we can compute the private encryption key from the public decryption key

$$d * e \equiv 1 \pmod{(p-1)(q-1)}$$

80

The keys

- We have $n = p*q = 10379$, $e = 85$, and $d = 4669$
- The public key is $(n,e) = (10379, 85)$
- The private key is $(n,d) = (10379, 4669)$
- Thus, n is not private
  - Only d is private
- In reality, d and e are 600 (or so) digit numbers
  - Thus n is a 1200 (or so) digit number

81

Encrypting messages

- To encode a message:
  1. Encode the message m into a number
  2. Split the number into smaller numbers $m < n$
  3. Use the formula $c = m^e \bmod n$
     - c is the ciphertext, and m is the message
- Java code to do the last step:
  - m.modPow (e, n)
  - Where the object m is the BigInteger to encrypt

82

Encrypting messages example

1. Encode the message into a number
   - String is "Gig 'em!"
   - ***Modified*** ASCII codes:
     - 41 75 73 02 09 71 79 03
2. Split the number into numbers < n
   - 4175 7302 0971 7903

I modified the ASCII codes for this example by subtracting 30 from each value
This is just so this example in the slides can use smaller values for p and q

83

Encrypting messages example

1. Split the number into numbers < n
   - 4175 7302 0971 7903
2. Use the formula $c = m^e \bmod n$
   - $4175^{85} \bmod 10379 = 5428$
   - $7302^{85} \bmod 10379 = 10173$
   - $0971^{85} \bmod 10379 = 8246$
   - $7903^{85} \bmod 10379 = 5466$

- Encrypted message:
  - 5428 10173 8246 5466

84

Decrypting messages

1. Use the formula $m = c^d \bmod n$ on each number
2. Split the number into individual ASCII character numbers
3. Decode the message into a string

85

Decrypting messages example

- Encrypted message:
  - 5428 10173 8246 5466

1. Use the formula $m = c^d \bmod n$ on each number
   - $5428^{4669} \bmod 10379 = 4175$
   - $10173^{4669} \bmod 10379 = 7302$
   - $8246^{4669} \bmod 10379 = 971$
   - $5466^{4669} \bmod 10379 = 7903$
2. Split the numbers into individual characters
   - 41 75 73 02 09 71 79 03
3. Decode the message into a string
   - Unmodified ASCII codes: (by adding 30 to each one)
     - 71 105 103 32 39 101 109 33
   - Retrieved String is "Gig 'em!"

86