Announcements

- Readings
  - Today:
    - 3.4 (5th Edition: 2.4)
  - Monday and Wednesday:
    - 3.5, 3.6, 3.7 (5th Edition: 2.5, 2.6)

Number Theory (and applications to computing)

- Branch of Mathematics with direct relevance to computing
- Many significant applications
  - Cryptography
  - Hashing
  - Security
- Important tool set

Modular Arithmetic

- Arithmetic over a finite domain
- In computing, almost all computations are over a finite domain

What are the values computed?

```csharp
public void Test1() {
    byte x = 250;
    byte y = 20;
    byte z = (byte) (x + y);
    Console.WriteLine(z);
}

class Program {
    static void Main(string[] args) {
        Test1();
    }
}
```

Arithmetic mod 7

- $a +_7 b = (a + b) \mod 7$
- $a \times_7 b = (a \times b) \mod 7$

```
+ 0 1 2 3 4 5 6
0 0 1 2 3 4 5 6
1 1 2 3 4 5 6 0
2 2 3 4 5 6 0 1
3 3 4 5 6 0 1 2
4 4 5 6 0 1 2 3
5 5 6 0 1 2 3 4
6 6 0 1 2 3 4 5
```
### Group Theory

- A group \( G=(S, \cdot) \) is a set \( S \) with a binary operator \( \cdot \) that is "well behaved":
  - Closed under \( \cdot \)
  - Associative: \( a \cdot (b \cdot c) = (a \cdot b) \cdot c \)
  - Has an identity
  - Each element has an inverse
- A group is commutative if the \( \cdot \) operator also satisfies \( a \cdot b = b \cdot a \)

### Groups, mod 7

- \( \{0,1,2,3,4,5,6\} \) is a group under \(+_7\)
- \( \{1,2,3,4,5,6\} \) is a group under \( \times_7 \)

### Multiplicative Inverses

- Euclid’s theorem: if \( x \) and \( y \) are relatively prime, then there exists integers \( s, t \), such that:
  \[
  sx + ty = 1
  \]
- Prove \( a \in \{1, 2, 3, 4, 5, 6\} \) has a multiplicative inverse under \( \times_7 \)

### Generalizations

- \( (\{0,\ldots,n-1\}, +_n) \) forms a group for all positive integers \( n \)
- \( (\{1,\ldots,n-1\}, \times_n) \) is a group if and only if \( n \) is prime

### Basic applications

- Hashing: store keys in a large domain \( 0\ldots M-1 \) in a much smaller domain \( 0\ldots n-1 \)

### Hashing

- Map values from a large domain, \( 0\ldots M-1 \) in a much smaller domain, \( 0\ldots n-1 \)
  - Index lookup
  - Test for equality
  - Hash(x) = x mod p
  - Often want the hash function to depend on all of the bits of the data
    - Collision management
Pseudo Random number generation
- Linear Congruential method
  \[ x_{n+1} = (ax_n + c) \mod m \]

Data Permutations
- Caesar cipher, \( a = 1, b = 2, \ldots \)
  - HELLO WORLD
- Shift cipher
  - \( f(x) = (x + k) \mod n \)
  - \( f^{-1}(x) = (x - k) \mod n \)
- Affine cipher
  - \( f(x) = (ax + b) \mod n \)
  - \( f^{-1}(x) = (a^{-1}(x-b)) \mod n \)

Modular Exponentiation

<table>
<thead>
<tr>
<th>x</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>1</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>6</td>
<td>2</td>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>2</td>
<td>6</td>
<td>3</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>3</td>
<td>1</td>
<td>6</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>5</td>
<td>4</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Fermat’s Little Theorem
- If \( p \) is prime, \( 0 < a \leq p-1, \ a^{p-1} \equiv 1 \pmod{p} \)
- Group theory
  - Index of \( x \), smallest \( i > 0 \) such that \( x^i = 1 \)
  - The index of \( x \) divides the order of the group

Exponentiation
- Compute \( 78365^{81453} \)
- Compute \( 78365^{81453} \mod 104729 \)

Fast exponentiation
```csharp
int FastExp(int x, int n){
    long v = (long)x;
    int m = 1;
    for (int i = 1; i <= n; i++){
        v = (v * v) % modulus;
        m = m + m;
        Console.WriteLine("i : "+ i + ", m : "+ m + ", v : "+ v );
    }
    return (int)v;
}
```
**Program Trace**

\[
\begin{align*}
&i: 1, m: 2, v: 52915 \\
&i: 2, m: 4, v: 95592 \\
&i: 3, m: 8, v: 70252 \\
&i: 4, m: 16, v: 26992 \\
&i: 5, m: 32, v: 74970 \\
&i: 6, m: 64, v: 71358 \\
&i: 7, m: 128, v: 20594 \\
&i: 8, m: 256, v: 10143 \\
&i: 9, m: 512, v: 61355 \\
&i: 10, m: 1024, v: 68404 \\
&i: 11, m: 2048, v: 4207 \\
&i: 12, m: 4096, v: 75698 \\
&i: 13, m: 8192, v: 56154 \\
&i: 14, m: 16384, v: 83314 \\
&i: 15, m: 32768, v: 99519 \\
&i: 16, m: 65536, v: 29057
\end{align*}
\]

**Fast exponentiation algorithm**

- What if the exponent is not a power of two?

\[
81453 = 2^{16} + 2^{13} + 2^{12} + 2^{11} + 2^{10} + 2^9 + 2^8 + 2^3 + 2^0
\]

The fast multiplication algorithm computes \(a^n \mod p\) in time \(O(\log n)\)

**Big number arithmetic**

- Computer Arithmetic 32 bit (or 64 bit, or 128 bit)
- Arbitrary precision arithmetic
  - Store number in arrays or linked lists
- Runtimes for standard algorithms for \(n\) digit numbers
  - Addition:
  - Multiplication:

**Discrete Log Problem**

- Given integers \(a, b\) in \([1, \ldots, p-1]\), find \(k\) such that \(a^k \mod p = b\)

**Primality**

- An integer \(p\) is prime if its only divisors are 1 and \(p\)
- An integer that is greater than 1, and not prime is called composite
- Fundamental theorem of arithmetic:
  - Every positive integer greater than one has a unique prime factorization

**Factorization**

- If \(n\) is composite, it has a factor of size at most \(\sqrt{n}\)
Euclid’s theorem

- There are an infinite number of primes.
- Proof by contradiction:
  - Suppose there are a finite number of primes: \( p_1, p_2, \ldots, p_n \)

Distribution of Primes

- If you pick a random number \( n \) in the range \([x, 2x]\), what is the chance that \( n \) is prime?

Famous Algorithmic Problems

- Primality Testing:
  - Given an integer \( n \), determine if \( n \) is prime
- Factoring
  - Given an integer \( n \), determine the prime factorization of \( n \)

Showing a number is NOT prime

- Trial division by small primes
- Fermat’s little theorem
  - \( a^{p-1} \mod p = 1 \) if \( p \) is prime
- Miller’s Test
  - if \( p \) is prime, the only square roots of one are 1 and -1
  - if \( p \) is composite other numbers can be the square root of one
  - repeated squaring used to find a non-trivial square root of one from a starting value \( b \)

Primality Testing

- Is the following 200 digit number prime:

```
409240841606028179761235258752542092850986220134
039205234540055280352862154399159482687571883797847351
18621136102969490409809013306665025656080656092539012888
01902035441804876167944219033
```

- Conduct Miller’s test for a random \( b \)
  - If \( p \) is prime, it always passes the test
  - If \( p \) is not prime, it fails with probability \( \frac{3}{4} \)
- Primality testing
  - Choose 100 random \( b \)'s and perform Miller’s test on each
  - If any say false, answer “Composite”
  - If all say true, answer “Prime”

Probabilistic Primality Testing
Greatest Common Divisor

- **GCD(a, b):** Largest integer d such that d|a and d|b
- **GCD(100, 125) =**
- **GCD(17, 49) =**
- **GCD(11, 66) =**

Euclid’s Algorithm

- **GCD(x, y) = GCD(y, x mod y)**

```c
int GCD(int a, int b){   /* a >= b,   b > 0 */
    int tmp;
    int x = a;
    int y = b;
    while (y > 0){
        tmp = x % y;
        x = y;
        y = tmp;
    }
    return x;
}
```

Extended Euclid’s Algorithm

- If GCD(x, y) = g, there exist integers s, t, such sx + ty = g;
- The values x, y in Euclid’s algorithm are linear sums of a, b.
  – A little book keeping can be used to keep track of the constants

Chinese Remainder Theorem

- Find an x in $[0 \ldots 11484]$ such that
  - $x \mod 11 = 9$
  - $x \mod 29 = 7$
  - $x \mod 36 = 14$

  **Simple version:**
  Suppose: p, q prime
  - $x \equiv a \pmod{p}$
  - $x \equiv b \pmod{q}$
  What is $x \pmod{pq}$?

  - p, q prime, $x \mod p = a$, $x \mod q = b$
  - Choose s, t such that sp + tq = 1
  - Let $f(a, b) = (atq + bsp) \mod pq$
  - $f(a, b) \mod p = a$; $f(a, b) \mod q = b$
  - f is 1 to 1 between $[0..p-1] \times [0..q-1]$ and $[0..pq - 1]$
  - Corollary:
    - $x \mod p = a; x \mod q = a$, then $x \mod pq = a$

Cryptography

ALICE  BOB
Perfect encryption

- Alice and Bob have a shared n-bit secret S
- To send an n-bit message M, Alice sends $M \oplus S$ to Bob
- Bob receives the message N, to decode, Bob computes $N \oplus S$

Public Key Cryptography

- How can Alice send a secret message to Bob if Bob cannot send a secret key to Alice?

RSA

- Rivest – Shamir – Adelman
- $n = pq$. p, q are large primes
- Choose $e$ relatively prime to $(p-1)(q-1)$
- Find $d$, $k$ such that $de + k(p-1)(q-1) = 1$ by Euclid’s Algorithm
- Publish $e$ as the encryption key, $d$ is kept private as the decryption key

Message protocol

- Bob
  - Precompute $p$, $q$, $n$, $e$, $d$
  - Publish $e$, $n$
- Alice
  - Read $e$, $n$ from Bob’s public site
  - To send message $M$, compute $C = M^e \mod n$
  - Send $C$ to Bob
- Bob
  - Compute $C^d$ to decode message $M$

Decryption

- $de = 1 + k(p-1)(q-1)$
- $C^d \equiv (M^e)^d \equiv M^{de} \equiv M^1 \cdot k(p-1)(q-1) \pmod{n}$
- $C^d \equiv M \cdot (M^{p-1})^k \equiv M \pmod{p}$
- $C^d \equiv M \cdot (M^{q-1})^k \equiv M \pmod{q}$
- Hence $C^d \equiv M \pmod{pq}$

Practical Cryptography

- Alice
  - Here is my public key
  - I want to talk to you, here is my private key
- Bob
  - Okay, here is my private key
  - Yadda, yadda, yadda