# MA/CSSE 473
# Day 06

Euclid's Algorithm

## MA/CSSE 473 Day 06

- **Student Questions**
- Odd Pie Fight
- Euclid's algorithm
- (if there is time) extended Euclid's algorithm

Quick look at review topics in textbook

# REVIEW THREAD

## Another Induction Example

- Pie survivor
  - An odd number of people stand in various positions (2D or 3D) such that no two distances between people are equal.
    - Each person has a pie
    - A whistle blows, and each person simultaneously and accurately throws his/her pie at the nearest neighbor
  - **Claim:** No matter how the people are arranged, at least one person does not get hit by a pie
  - Let P(n) denote the statement: "There is a survivor in every odd pie fight with $2n + 1$ people"
  - Prove by induction that P(n) is true for all $n \geq 1$

Q2

Euclid's Algorithm

Heading toward Primality Testing

# ARITHMETIC THREAD

## Euclid's Algorithm: the problem

- One of the oldest known algorithms (about 2500 years old)
- **The problem:** Find the greatest common divisor (gcd) of two non-negative integers a and b.
- The approach you learned in elementary school:
  - Completely factor each number
  - find common factors (with multiplicity)
  - multiply the common factors together to get the gcd
- Factoring Large numbers is hard!
- Simpler approach is needed

# Euclid's Algorithm: the basis

- Based on the following rule:
  - If x and y are positive integers with $x \geq y$, then gcd(x, y) = gcd(y, x mod y)
- Proof of Euclid's rule:
  - It suffices to show the simpler rule
    gcd(x, y) = gcd(y, x - y)
    since x mod y can be obtained from x and y by repeated subtraction
  - Any integer that divides both x and y must also divide x – y, so gcd(x, y) ≤ gcd(y, x – y)
  - Any integer that divides both y and x - y must also divide x, so gcd(y, x-y) ≤ gcd(y, x)
  - Putting these together: gcd(y, x-y) = gcd(y, x)

# Euclid's Algorithm: the algorithm

```
def euclid(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: gcd(a, b)"""
    if b == 0:
        return a
    return euclid(b, a % b)
```

- Example: euclid(60, 36)
- Does the algorithm work?
- How efficient is it?

# Euclid's Algorithm: the analysis

```python
def euclid(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: gcd(a, b)"""
    if b == 0:
        return a
    return euclid(b, a % b)
```

- Lemma: If **a** ≥ **b**, then **a** % **b** < **a**/2
- Proof
  - If **b** ≤ **a**/2, then **a** % **b** < **b** ≤ **a**/2
  - If **b** > **a**/2, then **a** % **b** = **a** – **b** < **a**/2
- Application
  - After two recursive calls, both **a** and **b** are less than half of what they were, (i.e. reduced by at least 1 bit)
  - Thus if a and b have k bits, at most 2k recursive calls are needed.
  - Each recursive call involves a division, $\Theta(k^2)$
  - Entire algorithm is $\Theta(k^3)$

# gcd and linear combinations

- Lemma: If **d** is a common divisor of **a** and **b**, and **d** = **a**x + **b**y for some integers x and y, then **d** = gcd(**a**,**b**)
- Proof
  - By the first of the two conditions, **d** divides both **a** and **b**. It cannot exceed their greatest common divisor, so **d** ≤ gcd(**a**, **b**)
  - gcd(**a**, **b**) is a common divisor of **a** and **b**, so it must divide **a**x + **b**y = **d**. Thus gcd(**a**, **b**) ≤ **d**
  - Putting these together, gcd(**a**, **b**) = **d**
- If we can supply the x and y as in the lemma, we have found the gcd.
- It turns out that a simple modification of Euclid's algorithm will calculate the x and y.

## Extended Euclid Algorithm

```
def euclidExtended(a, b):
    """ INPUT:  Two integers a and b with a >= b >= 0
        OUTPUT: Integers x, y, d such that d = gcd(a, b)
                and d = ax + by"""
    print ("    ", a, b) # so we can see the process.
    if b == 0:
        return 1, 0, a
    x, y, d =  euclidExtended(b, a % b)
    return y, x - a//b*y, d
```

- Proof that it works
  - First, the number d it produces really is the gcd of a and b. If we ignore the x and y values, and we have the same algorithm as before.

## Example: gcd (33, 14)

- 33 = 2*14 + 5
- 14 = 2 * 5 + 4
- 5 = 1 * 4 + 1
- 4 = 4 * 1 + 0, so gcd(33, 14) = 1.
- **Now work backwards**
- 1 = 5 - 4. Substitute 4 = 14 - 2*5.
- 1 = 5 – (14 - 2*5) = 3*5 - 14. Substitute 5 = 33 - 2*14
- 1 = 3(33 - 2*14) -14 = 3 * 33 – 7 * 14
- Thus x = 3 and y = -7 Done!

## Modular Inverse

- In the real or rational numbers, every non-zero number **a** has an inverse 1/**a**, also written $a^{-1}$
  - x is the inverse of **a** iff **a**x = 1
  - Every non-zero real number has a unique inverse
- **Definition** x is the **multiplicative inverse of a (modulo** N) if **a**x $\equiv$ 1 (mod N)
- We denote this inverse by $\mathbf{a}^{-1}$ (if it exists)
  - Note that 2 has no inverse modulo 6
  - Does 5 have an inverse (modulo 6)?
- **a** has an inverse modulo N if and only if gcd(a, N) = 1
    - i.e. **a** and N are **relatively prime**
- If $\mathbf{a}^{-1}$ exists, it is unique (among 1..N-1)

## Calculate Modular Inverse (if it exists)

- Assume that gcd(a, N) = 1.
- The extended Euclid's algorithm gives us integers x and y such that ax + Ny = 1
- This implies ax $\equiv$ 1 (mod N), so x is the inverse of a
- **Example:** Find $11^{-1}$ mod 25
  - We saw before that -9*11 + 4*25 = 1
  - -9 $\equiv$ 16 (mod 25)
  - So $11^{-1} \equiv 16$ (mod 25)
- Recall that Euclid's algorithm is $\Theta(k^3)$, where k is the number of bits of N.