Overview

- HW6 due in one week on 11/3. Work on it!
- Thr Q&A: Bring your questions: Relation/Function/Algorithm!
- Last Lecture
  - Algorithm: Explicit instructions of computations
  - Pseudocode to express algorithms
  - Complexity Function by counting # of basic operations.
  - Time Complexity = Growth as a function of # of input items.
- This Lecture: Complete Lectures on Algorithms (Two Lecs)
  - Big-O notation of Growth of Complexity Function
  - Tractable vs Intractable Problems
  - Computability theory
    - Number theory
      - Division, Primes
      - GCD, LCM
    - Euclidean Algorithm for computing GCD
  - Modular arithmetic, Congruence

Review: Analysis of Algorithm

- Algorithm = a finite sequence of precise instructions
- Step 1: Pseudo Code is used to represent an algorithm
- Step 2: Derive Complexity Function $f(n)$ as the number of basic operations executed within loops when given $n$ inputs.
- Step 3: Time Complexity of algorithm $\leftarrow$ Big-O of its complexity function!
  - Idea: Describe the growth of # steps as you increase # of inputs
  - A) Choose a Reference Function $g(n)$ from the list
    - $1 < \log n < n < n \log n < n^2 < n^3 < n^m < 2^n < 10^n < n!$
  - B) Show that $f(n)$ is $O(g(n))$ by finding a pair of C and k such that $f(n) \leq C g(n)$ whenever $n > k$
  - C) Choose the left most function from the list that you can show your $f(n)$ is $O(g(n))$
  - D) Compare algorithms by $g(n)$ of their respective big-O.
• **2-Step Complexity Analysis:** Choose \( g(n) \) and Prove \( O(g) \)!

• “Popular” reference functions \( g(n) \) are:
  
  \[
  1 < \log n < n < \log n < n^2 < n^3 < \ldots < 2^n < 10^n < n!
  \]

  (above are listed from slowest to fastest growth)

• A problem that can be solved with polynomial worst-case complexity is called **tractable**.

• Problems of higher complexity are called **intractable**.

• Problems that no algorithm can solve are called **unsolvable**. (more on this later...)

\[
f(n) \leq CG(n) \text{ for } n \geq k
\]

---

**Useful Rules for Big-O**

• For any **polynomial** \( f(x) = a_n x^n + a_{n-1} x^{n-1} + \ldots + a_0 \), where \( a_n, a_{n-1}, \ldots, a_0 \) are real numbers, \( f(x) \) is \( O(x^n) \).

• If \( f_1(x) \) is \( O(g_1(x)) \) and \( f_2(x) \) is \( O(g_2(x)) \), then
  
  \[
  (f_1 + f_2)(x) \text{ is } O(\max(g_1(x), g_2(x)))
  \]

• If \( f_1(x) \) is \( O(g(x)) \) and \( f_2(x) \) is \( O(g(x)) \), then
  
  \[
  (f_1 + f_2)(x) \text{ is } O(g(x)).
  \]

• If \( f_1(x) \) is \( O(g_1(x)) \) and \( f_2(x) \) is \( O(g_2(x)) \), then
  
  \[
  (f_1 f_2)(x) \text{ is } O(g_1(x) g_2(x)).
  \]
Complexity Function Examples

Function `max_diff(List[1], List[2], ..., List[n])`

```plaintext
m = 0
for i = 1 to n-1
  for j = i + 1 to n
    if |List[i] – List[j]| > m then
      m = |List[i] – List[j]|

// m is the maximum difference between any
// two numbers in the input sequence
```

Comp. Func.: \(n - 1 + n - 2 + n - 3 + \ldots + 1 = (n - 1)n/2 = 0.5n^2 - 0.5n\)

Time complexity is \(O(n^2)\).

Another algorithm solving the same problem:

Function `max_diff(List[1], List[2], ..., List[n])`

```plaintext
min = List[1]
max = List[1]
for i = 2 to n
  if List[i] < min then min = List[i]
  else if List[i] > max then max = List[i]
  m = |max – min|
```

Comp. Func.: \(2(n - 1) = 2n - 2\)

Time complexity is \(O(n)\).
Computability Theory (informal intro.)

- A decision problem is a question with a “yes” or “no” answer, depending on the values of some input parameters. Example: Given two numbers x and y, is $x < y$?
- A decision problem is called decidable (or solvable) if you can construct an algorithm that always returns “yes” or “no” for any input parameters.
- A decision problem is undecidable (or unsolvable) if it is impossible to construct an algorithm that always leads to a “yes” or “no” answer for any input.
- Example: The Halting Problem: For any algorithm A and any input string I, can you tell if $A(I)$ ultimately halt/complete or will it run on forever?
  
i.e. Can you write a program $H(A,I)$ that takes input $(A,I)$ and returns “yes” if A halts on input I and “no” if A runs forever on input I?

---

**Answer is NO: Theorem: The Halting Problem is undecidable**

(Informal) Proof by contradiction: Assume $H(A, I)$ existed (The halting problem is decidable).

Let us define the following algorithm $K(A)$ which uses $H(A,I)$:

**Function K(A)**

```
if (H(A,A) == “no”) // if A(A) runs for ever, return “yes”
  return “yes”
else // if A(A) halts
  for(;;){} // loop forever and never return
```

Note: $K(A)$ halts if and only if $H(A,A)$ returns “no” if and only if $A(A)$ does not halt.

Now, call the algorithm $K$ with input $K$, i.e. $K(K)$

Then, $K(K)$ halts if and only if $H(K,K)$ returns “no” if and only if $K(K)$ does not halt.

This is a contradiction!
Introduction to Number Theory

• Number theory is about integers and their properties.
• We will start by reviewing the basic principles of
  prime numbers
  divisibility,
  greatest common divisors,
  least common multiples, and
  modular arithmetic
and look at some relevant algorithms.
  Euclidian Algorithm

Division

• 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 \( a \mid b \) means that \( a \) divides \( b \).
• We write \( a \nmid b \) when \( a \) does not divide \( b \)
• For integers \( a, b, \) and \( c \) it is true that
  if \( a \mid b \) and \( a \mid c \), then \( a \mid (b + c) \)
  \textbf{Example:} 3 \mid 6 \text{ and } 3 \mid 9, \text{ so } 3 \mid 15.
  \text{ if } a \mid b, \text{ then } a \mid bc \text{ for all integers } c
  \textbf{Example:} 5 \mid 10, \text{ so } 5 \mid 20, 5 \mid 30, 5 \mid 40, \ldots
  \text{ if } a \mid b \text{ and } b \mid c, \text{ then } a \mid c
  \textbf{Example:} 4 \mid 8 \text{ and } 8 \mid 24, \text{ so } 4 \mid 24.
Primes

A positive integer \( p > 1 \) is called prime iff 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.

Examples (where the prime factors are commonly written in order of increasing size):

\[
\begin{align*}
15 & = 3 \times 5 \\
48 & = 2^4 \times 3 \\
17 & = 17 \\
100 & = 2^2 \times 5^2 \\
512 & = 2^9 \\
\end{align*}
\]

The Division Algorithm

Let \( a \) be an integer and \( d \) a positive integer. Then there are unique integers \( q \) and \( r \), with \( 0 \leq r < d \), such that

\[
a = dq + r
\]

In the above equation, \( a \) is called the dividend, \( d \) is called the divisor, \( q \) is called the quotient, and \( r \) is called the remainder.

Example: When we divide 17 by 5, we have \( 17 = 5 \times 3 + 2 \). 17 is the dividend, 5 is the divisor, 3 is the quotient, and 2 is the remainder.

Example: Divide -11 by 3?
Note that the remainder cannot be negative.
We have: \(-11 = 3 \times (-4) + 1\)
**Greatest Common Divisors**

- Let \( a \) and \( b \) be integers, not both zero. The largest positive integer \( d \) such that \( d \mid a \) and \( d \mid b \) 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, 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 \( \gcd(19, 72) = 1 \).

- **How to find \( \gcd(a, b) \)?** Using prime factorizations: Let \( a = p_1^{a_1} p_2^{a_2} \ldots p_n^{a_n}, \ b = p_1^{b_1} p_2^{b_2} \ldots p_n^{b_n} \), where \( p_1 < p_2 < \ldots < p_n \) and \( a_i, b_i \in \mathbb{N} \) for \( 1 \leq i \leq n \) and \( \gcd(a, b) = p_1^{\min(a_1, b_1)} p_2^{\min(a_2, b_2)} \ldots p_n^{\min(a_n, b_n)} \).

- **Example:**
  \[
  a = 60 = 2^2 3^1 5^1 \\
  b = 54 = 2^1 3^3 5^0
  \]

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

**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, \( \gcd(35, 28) = 7 \).

- **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 28 pairwise relatively prime?
  Yes, because \( \gcd(15, 17) = 1, \gcd(15, 28) = 1 \) and \( \gcd(17, 28) = 1 \).
  Are 15, 17, and 27 pairwise relatively prime?
  No, because \( \gcd(15, 27) = 3 \).
Least Common Multiples

- Definition: The \textbf{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 \(\text{lcm}(a, b)\).

- Examples: \(\text{lcm}(3,7)=21, \text{lcm}(4,6)=12, \text{lcm}(5,10)=10\)

- 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 \(a_i, b_i \in \mathbb{N}\) for \(1 \leq i \leq n\)

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

- Example:

\[
\begin{align*}
a &= 60 = 2^2 \cdot 3^1 \cdot 5^1 \\
b &= 54 = 2^1 \cdot 3^3 \cdot 5^0
\end{align*}
\]

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

GCD and LCM

\[
\begin{align*}
a &= 60 = 2^2 \cdot 3^1 \cdot 5^1 \\
b &= 54 = 2^1 \cdot 3^3 \cdot 5^0
\end{align*}
\]

\[
\begin{align*}
\text{gcd}(a, b) &= 2^1 \cdot 3^1 \cdot 5^0 = 6 \\
\text{lcm}(a, b) &= 2^2 \cdot 3^3 \cdot 5^1 = 540
\end{align*}
\]

\textbf{Theorem:} \(a \cdot b = \text{gcd}(a,b) \cdot \text{lcm}(a,b)\)
Application: Program Correctness

Euclidean Algorithm

- By Greek Mathematician Euclid (2300 years ago), one of the oldest known algorithm. Finds the greatest common divisor of 2 nonnegative integers $a$ and $b$

- Example, if we want to find $gcd(287, 91)$, we divide 287 by 91, and get $287 = 91 \times 3 + 14$ \hspace{1cm} \text{// $i=j\times q+r$ in program}

In the next step, if the remainder is not zero, we divide 91 (divisor) by 14 (remainder) and get $91 = 14 \times 6 + 7$

So again we divide 14 by 7 and get to remainder 0! Then we stop $14 = 7 \times 2 + 0$

Therefore, $gcd(287, 91) = 7$.

Euclidean Algorithm:

$GCD($nonnegative integer $a$; nonnegative integer $b$)

//input : $a \geq b$, not both $a$ and $b$ are zero

Local variables: integers $i, j$

$i = a$; $j = b$  \hspace{1cm} // initialize value of $i$ and $j$

while $j \neq 0$ do

compute $i = qj + r$, $0 \leq r < j$

$i = j$

$j = r$
end while

// $i$ now has the value $gcd(a, b)$

return $i$;
end function $GCD$
Property *: \( \forall \) integers \( a, b, q, r \)[
\( (a = bq + r) \rightarrow (\gcd(a, b) = \gcd(b, r)) \)]

Proof by contradiction

Assume that \( a = bq + r \) and \( \gcd(a, b) \neq \gcd(b, r) \)

Let \( a = \gcd(a, b) \cdot c_1 \) and \( b = \gcd(a, b) \cdot c_2 \) for some integers \( c_1 \) and \( c_2 \).

\( r = a - qb = \gcd(a, b) \cdot c_1 - q \cdot \gcd(a, b) \cdot c_2 = \gcd(a, b) \cdot (c_1 - qc_2) \)

So \( \gcd(a, b) \) divides \( r \) (also divides \( a \) and \( b \) by definition of \( \gcd(a, b) \))

Similarly, we can show that \( \gcd(b, r) \) divides \( a \) (also divides \( b \) and \( r \))

As \( \gcd(a, b) \neq \gcd(b, r) \)

Case 1: \( \gcd(a, b) > \gcd(b, r) \), then \( \gcd(b, r) \) cannot be the gcd of \( b \) and \( r \).
(since \( \gcd(a, b) \) is greater value and divides both \( b \) and \( r \))
Contradiction!

Case 2: \( \gcd(a, b) < \gcd(b, r) \) then \( \gcd(a, b) \) cannot be the gcd of \( a \) and \( b \).
(similar argument as case 1), Contradiction!

---

To prove the correctness of this function, we need to prove the following

Let \( i_n \) and \( j_n \): values of \( i \) and \( j \) after the \( n \)th loop in Euclidean Algorithm.

We need to prove \( Q(n): \gcd(i_n, j_n) = \gcd(a, b) \) is true for all \( n \geq 0 \)

Use Induction

- **IB:** \( n = 0 \), \( Q(0): \gcd(i_0, j_0) = \gcd(a, b) \) is true as \( i \) and \( j \) are initialized to be \( a \) and \( b \) before entering the loop.

- **IH:** Assume \( Q(k): \gcd(i_k, j_k) = \gcd(a, b) \)

- **IS:** Show \( Q(k+1): \gcd(i_{k+1}, j_{k+1}) = \gcd(a, b) \)

By the assignment statements within the loop body, we know

it computes \( i_k = a \cdot j_k + r_{k-1} \), \( i_{k+1} = j_k \), and \( j_{k+1} = r_k \)

then \( \gcd(i_{k+1}, j_{k+1}) = \gcd(j_k, r_k) = \gcd(i_k, j_k) \) (by property *)

and \( \gcd(i_k, j_k) = \gcd(a, b) \) (by IH)

Therefore, function \( GCD \) is correct!

---
Modular Arithmetic

- Let \( a \) be an integer and \( m \) be a positive integer. We denote the remainder \( r \) when \( a \) is divided by \( m \) by \( a \mod m \).
- Examples:
  - \( 9 \mod 4 = 1 \)
  - \( 9 \mod 3 = 0 \)
  - \( 9 \mod 10 = 9 \)
  - \( -13 \mod 4 = 3 \)

Congruences

- Let \( a \) and \( b \) be integers and \( m \) be a positive integer. We say that \( a \) is congruent to \( b \) modulo \( m \) if \( m | (a - b) \).
- We use the notation \( a \equiv b \ (\mod m) \) to indicate that \( a \) is congruent to \( b \) modulo \( m \).
- We claim that \( a \equiv b \ (\mod m) \) if and only if \( a \mod m = b \mod m \).

\[ \]
**Theorem:** Let $m$ be a positive integer. Then

$$a \equiv b \pmod{m} \text{ iff } a \equiv b \pmod{m}.$$

**Proof:** Let $a = m \cdot q_1 + r_1$, and $b = m \cdot q_2 + r_2$ (A)

**If part:** show if $a \equiv b \pmod{m}$ then $a \equiv b \pmod{m}$

$a \equiv b \pmod{m}$ implies $r_1 = r_2$, therefore

$A1-A2 \Rightarrow a - b = m(q_1 - q_2)$, thus $a \equiv b \pmod{m}$ by congruence def.

**Only if part:** show if $a \equiv b \pmod{m}$ then $a \equiv b \pmod{m}$

$a \equiv b \pmod{m}$ implies $a - b = m \cdot q$,

$m \cdot q_1 + r_1 - (m \cdot q_2 + r_2) = m \cdot q$ by plugging (A) to the above

this yields $r_1 - r_2 = m(q - q_1 + q_2)$.

Since $0 \leq r_1, r_2 < m$, it must be that $0 \leq |r_1 - r_2| < m$.

The only multiple of $m$ in that range is 0, i.e. $r_1 - r_2 = 0$.

Therefore $r_1 = r_2$, thus $a \equiv b \pmod{m}$.