Chapter 3
The Fundamentals: Algorithms The Integers

Objectives

- Algorithms
- The Growth of Functions
- Complexity of Algorithms
- The Integers and Division
- Primes and Greatest Common Divisors
- Integers and Algorithms

3.1- Algorithms

An algorithm is a finite set of precise instructions for performing a computation or for solving a problem.

Specifying an algorithm: natural language/ pseudocode

Properties of an algorithm

- Input
- Output
- Definiteness
- Correctness
- Effectiveness
- Generality

Finding the Maximum Element in a Finite Sequence

```
**Procedure** max ($a_1, a_2, a_3, \ldots, a_n$: integers)
    max:=$a_1$
    **for** i:=2 **to** n
        **if** max < $a_i$ **then** max:= $a_i$
{max is the largest element}
```

The Linear Search

```
Procedure linear search ($x$: integer, $a_1, a_2, \ldots, a_n$ :distinct
    integers)
    i:=1
    while $i \le n$ and $x \ne a_i$
            $i := i+1$
    if $i \le n$ then location:= i
    else location:=0
{location is the subscript of the term that equals x, or is 0 if
    x is not found}
```

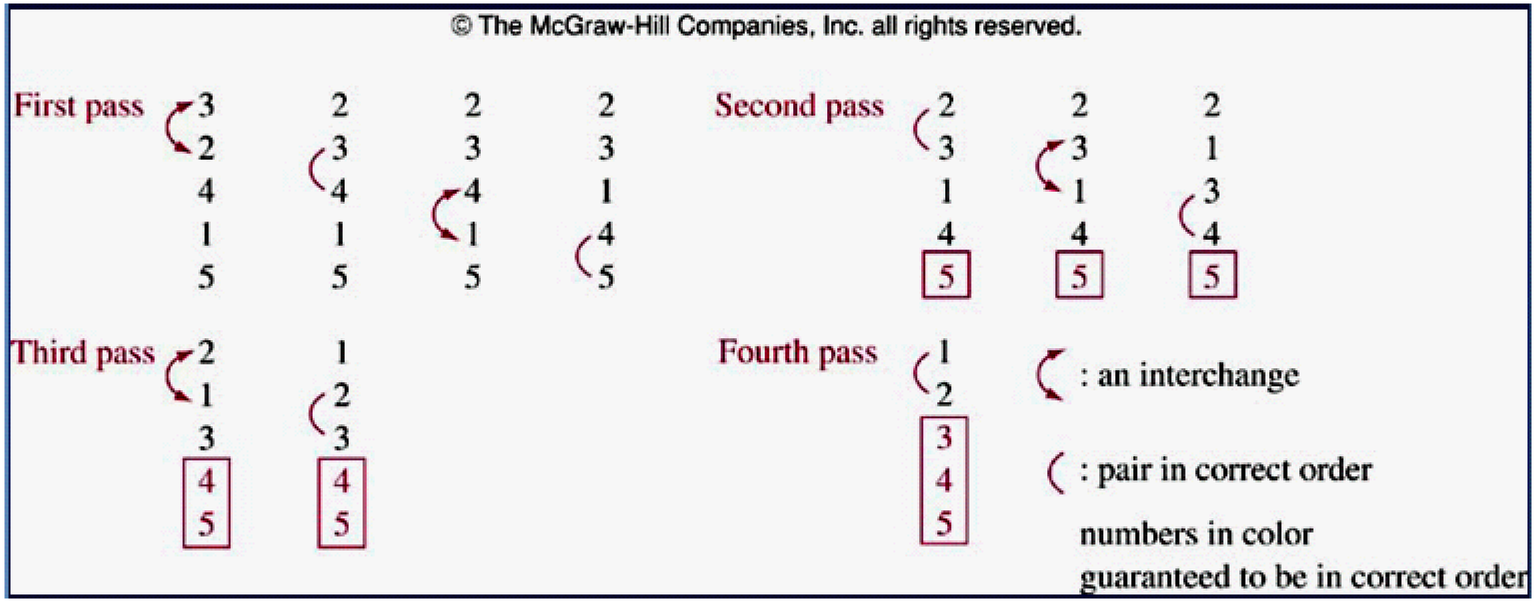
The Binary Search

```
procedure binary search ( x:integer, $a_1, a_2, \ldots, a_n$ : increasing
    integers)
    i:=1 { i is left endpoint of search interval}
    j:=n { j is right endpoint of search interval}
    while i<j
       begin
          m:= $\lfloor (i+j)/2 \rfloor$
          if  $x > a_m$ then  i:=m+1
          else  j:= m
       end
       if  $x = a_i$ then location := i
     else location:= 0
{location is the subscript of the term that equals x, or is 0 if x is not
    found}
```

Sorting

- Putting elements into a list in which the elements are in increasing order.
- There are some sorting algorithms
- Bubble sort
- Insertion sort
- Selection sort (exercise p. 178)
- Binary insertion sort (exercise p. 179)
- Shaker sort (exercise p.259)
- Merge sort and quick sort (section 4.4)
- Tournament sort (10.2)

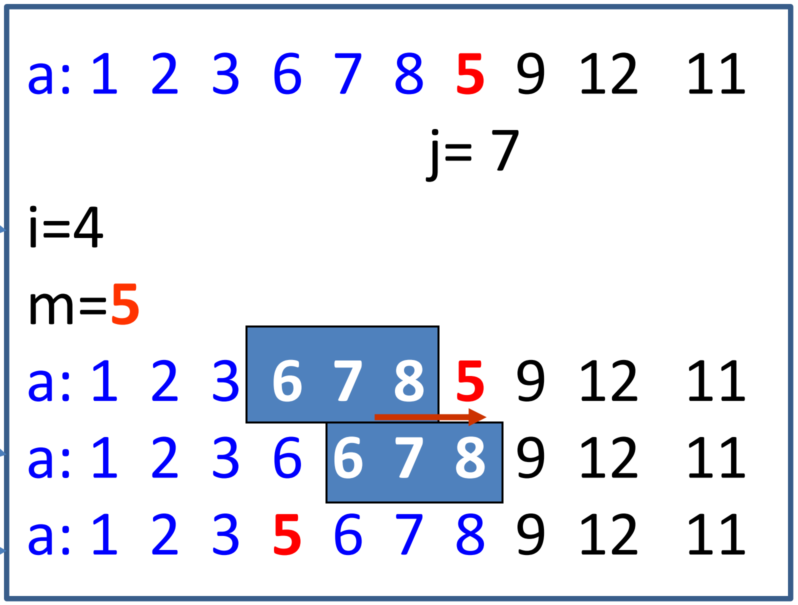
Bubble Sort


© The McGraw-Hill Companies, Inc. all rights reserved.
First pass
Second pass
Third pass
Fourth pass
: an interchange
: pair in correct order
numbers in color guaranteed to be in correct order

```
procedure buble sort ($a_1, a_2, \ldots, a_n$ :real numbers with $n \geq 2$)
for i:= 1 to n-1
    for j:=1 to n- i
        if $a_j > a_{j+1}$ then interchange $a_j$ and $a_{j+1}$
{$a_1, a_2, \ldots, a_n$ are sorted}
```

Insertion Sort

```
procedure insertion sort ($a_1,a_2,\ldots,a_n$ :real numbers with $n \geq 2$)
for j:= 2 to n { j: position of the examined element }
  begin
     { finding out the right position of $a_j$ }
     i:=1
     while $a_j > a_i$   i:= i+1
     m:=$a_j$ { save $a_j$ }
     { moving j-i elements backward }
     for k:=0 to j-i-1 $a_{j-k}$:=$a_{j-k-1}$
     {move $a_j$ to the position i}
     $a_i$:= m
  end
{$a_1,a_2,\ldots,a_n$ are sorted}
```



It is usually not the most efficient

Greedy Algorithm

- They are usually used to solve optimization problems: Finding out a solution to the given problem that either minimizes or maximizes the value of some parameter.
- Selecting the best choice at each step, instead of considering all sequences of steps that may lead to an optimal solution.
- Some problems:
  - Finding a route between two cities with smallest total mileage (number of miles that a person passed).
  - Determining a way to encode messages using the fewest bits possible.
  - Finding a set of fiber links between network nodes using the least amount of fiber.

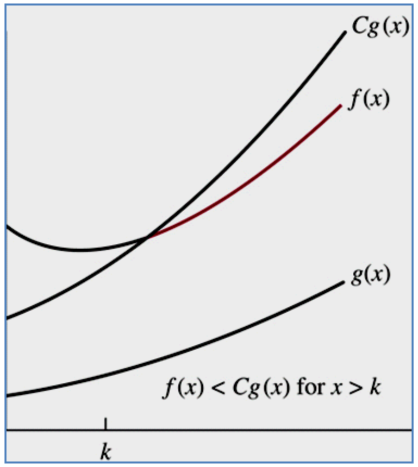
3.2- The Growth of Functions

- The complexity of an algorithm that acts on a sequence depends on the number of elements of sequence.
- The growth of a function is an approach that help selecting the right algorithm to solve a problem among some of them.
- Big-O notation is a mathematical representation of the growth of a function.

3.2.1-Big-O Notation

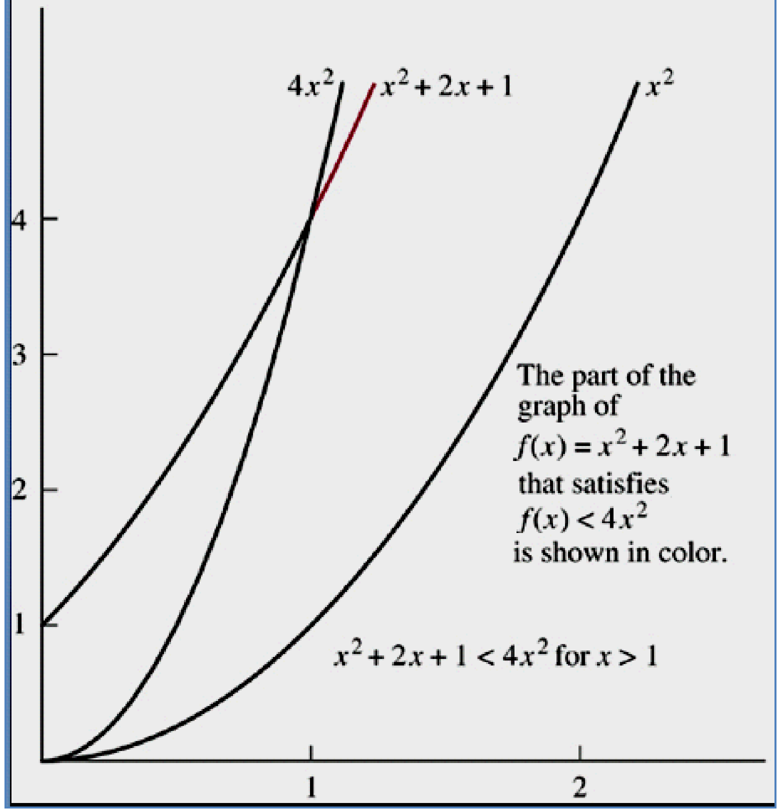
Definition:

Let *f* and g be functions from the set of integers or the set of real numbers to the set of real numbers. We say that f(x) is O(g(x)) if there are constants C and k such that $|f(x)| \le C|g(x)|$ whenever $x>k$



Example*:* **Show that $f(x)=x^2 + 2x +1$ is $O(x^2)$**

- Examine with $x>1 \Rightarrow x^2 > x$
- $\Rightarrow f(x)=x^2 + 2x +1 < x^2 + 2x^2 + x^2$
- $\Rightarrow f(x) < 4x^2$
- $\Rightarrow$ Let $g(x)= x^2$
- $\Rightarrow C=4, k=1, |f(x)| \le C|g(x)|$
- $\Rightarrow f(x)$ is $O(x^2)$



Big-O: Theorem 1

Let $f(x)=a_nx^n + a_{n-1}x^{n-1}+\ldots+a_1x+a_0$, where $a_0,a_1,\ldots,a_n$ are real number, then $f(x)$ is $O(x^n)$

If $x>1$

$|f(x)| = | a_nx^n + a_{n-1}x^{n-1}+\ldots+a_1x+a_0|$

$\leq | a_nx^n |+|a_{n-1}x^{n-1}|+\ldots+|a_1x|+|a_0|$ { triangle inequality }

$\leq x^n (| a_n |+|a_{n-1}x^{n-1}/x^n|+\ldots+|a_1x/x^n|+|a_0/x^n|)$

$\leq x^n (| a_n |+|a_{n-1}/x|+\ldots+|a_1/x^{n-1}|+|a_0/x^n|)$

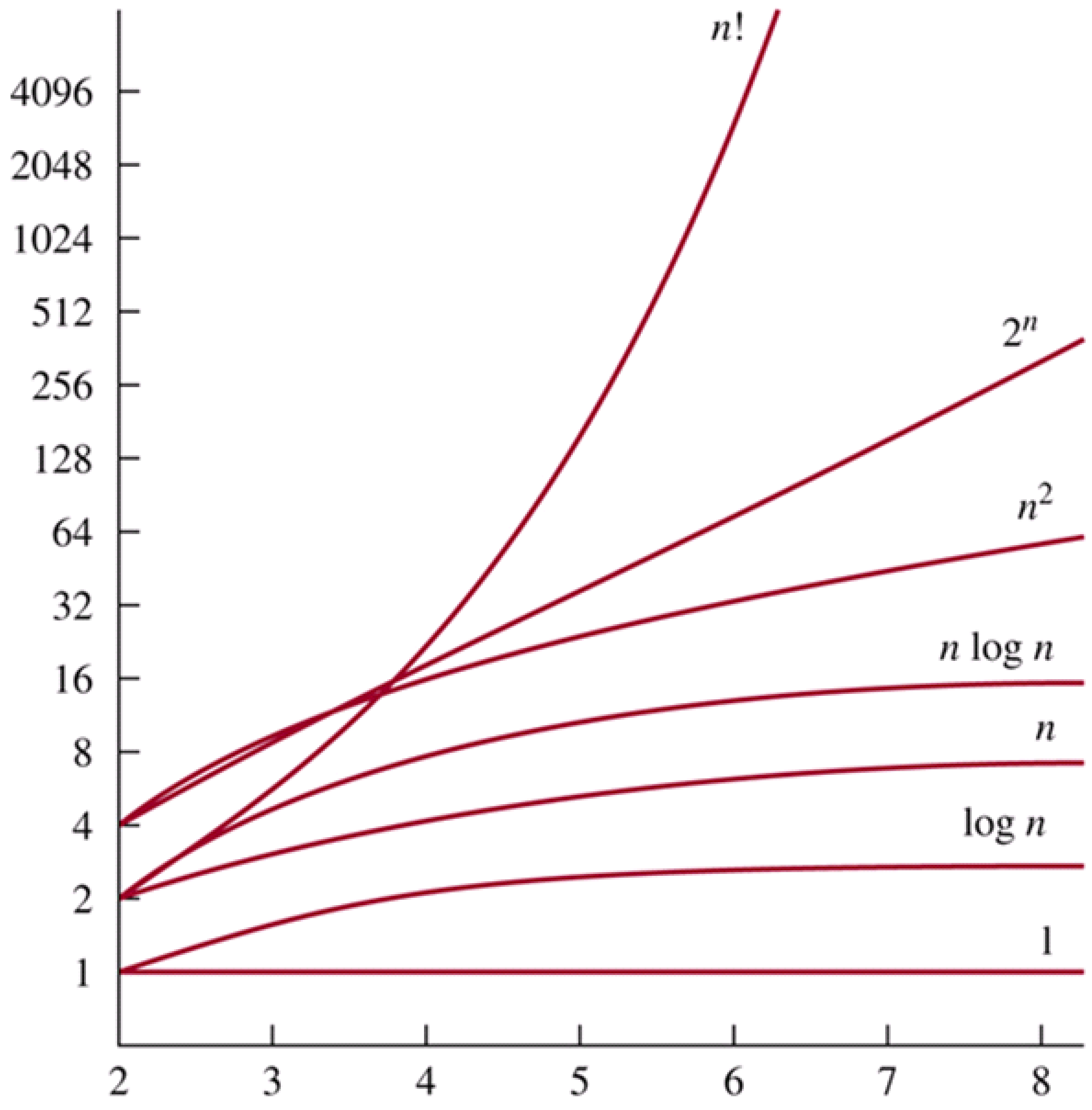
$\leq x^n (| a_n |+|a_{n-1}|+\ldots+|a_1|+|a_0|)$

Let $C= | a_n |+|a_{n-1}|+\ldots+|a_1|+|a_0|$

$|f(x)| \leq Cx^n$

➔ **$f(x) = O(x^n)$**

The Growth of Combinations of Functions



Big-O : Theorems

Theorem 2:

$f_1(x)=O(g_1(x)) \wedge f_2(x)=O(g_2(x))$

$\rightarrow (f_1+f_2)(x) = O(\max(|g_1(x)|,|g_2(x)|))$

Theorem 3:

$f_1(x)=O(g_1(x)) \wedge f_2(x)=O(g_2(x))$

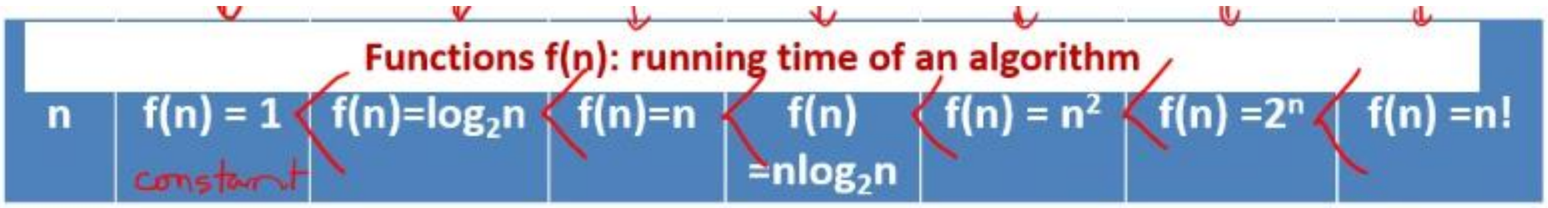
$\rightarrow (f_1 f_2)(x) = O(g_1 g_2(x)))$

Corollary 1:

$f_1(x)=O(g\,(x)) \wedge f_2(x)=O(g\,(x)) \rightarrow (f_1+f_2)(x) = O(g(x))$

3.2.2- Big-Omega and Big-Theta Notation

- Big-O does not provide the lower bound for the size of f(x)
- Big-Ω, Big- θ were introduced by Donald Knuth in the 1970s
- Big-Ω provides the lower bound for the size of f(x)
- Big- θ provides the upper bound and lower bound on the size of f(x)



Functions f(n): running time of an algorithm							
n	f(n) = 1	< f(n)=$\log_2 n$	< f(n)=n	< f(n) =$n\log_2 n$	< f(n) = n^2	< f(n) =2^n	< f(n) =n!
	constant						

Big-Omega and Big-Theta Notation

- Definitions

  $\exists c>0, k\ x \geq k \wedge |f(x)| \geq C|(g(x)| \rightarrow |f(x)| = \Omega(g(x))$

  $f(x)= O(g(x)) \wedge f(x)= \Omega(g(x)) \rightarrow f(x) = \theta(g(x))$

  If $f(x) = \theta(g(x))$ then $f(x)$ is of order $g(x)$

Show that $f(x)=1+2+\ldots+n$ is $\theta(n^2)$
Examining $x>0$
$f(x)=1+2+\ldots+n = n(n+1)/2 = (n^2 +n) /2$
$f(x) \leq (2n^2)/2$
$f(x) \leq n^2$
➔Let $c_1=1/2$, $c_2=1$, $g(x)= n^2$
➔ $c_1g(x) \leq f(x) \leq c_2g(x)$
➔ $f(x) = \theta\ (n^2)$ with $x>0$

Big-Omega and Big-Theta Notation

- Theorem 4

  Let $f(x)=a_n x^n + a_{n-1} x^{n-1}+\ldots+a_1 x+a_0$, where $a_0,a_1,\ldots,a_n$ are real number, then $f(x)$ is of order x^n

3.3- Complexity of Algorithms

- Computational complexity = Time complexity + space complexity.
- Time complexity can be expressed in terms of the number of operations used by the algorithm.
  - – Average-case complexity
  - – Worst-case complexity
- Space complexity will not be considered.

Demo 1

Describe the time complexity of the algorithm for finding the largest element in a set:

```
Procedure max ( $a_1, a_2, \ldots, a_n$ : integers)
  max:= $a_1$
  for i:=2 to n
    If  max < $a_i$ then max:= $a_i$
```

Time Complexity is θ(n)

i	Number of comparisons
2	2
3	2
…	2
n	2
n+1	1, max< a_i is omitted

2(n-1) +1 = 2n-1 comparisions

Demo 2

Describe the average-case time complexity of the linear-search algorithm :

```
Procedure linear search (x: integer, $a_1,a_2,\dots,a_n$ :distinct integers)
    i:=1
    while (i $\le$ n and x $\neq a_i$) i:=i+1
    if i $\le$ n then location:= i
    else location:=0
```

i	Number of comparisons done
1	2
2	4
…	
n	2n
n+1	1, $x \neq a_i$ is omitted

Avg-Complexity= [(2+4+6+…+2n)]/n +1 +1
= [2(1+2+3+…+n) /n+2
= [2n(n+1)/2]/n + 2
=[n(n+1)]/n + 2
= n+1 + 2 = n+3
= θ(n)

See demonstrations about the worst-case complexity: Examples 5,6 pages 195, 196

Understanding the Complexity of Algorithms

Complexity	Terminology	Problem class
Θ(1)	Constant complexity	Tractable (dễ), class P
Θ(log n)	Logarithmic complexity	Class P
Θ(n)	Linear complexity	Class P
Θ(n logn)	n log n complexity	Class P
Θ(n^b) , b⍰1, integer	Polynominal complexity	Intractable, class NP
Θ(b^n), b>1	Exponential complexity	
Θ(n!)	Factorial complexity	

NP : Non-deterministic Polynomial time

3.4- The Integers and Division

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

When a divides b, we say that:

a is a factor of b

b is a multiple of a

Notation: $a|b$: a divides b $a \nmid b$: a does not divide b

Example:

$3 \nmid 7$ because $7/3$ is not an integer

$3|12$ because $12/3=4$, remainder=0

Corollary 1:

$a|b \wedge a|c \rightarrow a|(mb+nc)$, m,n are integers

The Division Algorithm

Theorem 2: 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 $a=dq+r$

d: divisor , r : remainder, q: quotient (thương số)

Note: r can not be negative

Definition: $a=dq+r$

a: dividend	d: divisor
q: quotient	r : remainder,

q= a div d r = a mod d

Example:

101 is divided by 11 :101 = 11.9 + 2 ➔ q=9, r=2

-11 is divided by 3 : 3(-4)+ 1 ➔ q=-4, r=1

No OK : -11 is divided by 3 : 3(-3)-2 ➔ q=-3, **r = -2**

Modular Arithmetic

Definition: a, b: integers, m: positive integer.

a is called ***congruent*** to b modulo m if m|a-b

Notations:

$a \equiv b \pmod{m}$, a is congruent to b modulo m

$a \not\equiv b \pmod{m}$, a is not congruent to b mod m

Examples:

15 is congruent to 6 modulo 3 because 3|15-6

15 is **not** congruent to 7 modulo 3 because $3 \nmid 15-7$

Modular Arithmetic

Theorem 3

a,b: integers, m: positive integer

$a \equiv b \pmod{m} \leftrightarrow a \bmod m = b \bmod m$

Proof

(1) $a \equiv b \pmod{m} \rightarrow a \bmod m = b \bmod m$

$a \equiv b \pmod{m} \rightarrow m \mid a-b \rightarrow a-b= km \rightarrow a=b + km$

$\rightarrow a \bmod m = (b + km) \bmod m$

$\rightarrow a \bmod m = b \bmod m$ { km mod m = 0 }

(2) $a \bmod m = b \bmod m \rightarrow a \equiv b \pmod{m}$

$a = k1m + c \wedge b=k2m + c \rightarrow a-b = (k1-k2)\, m$ { suppose a>b)

$= km$ { k= k1-k2}

$\rightarrow m\mid a-b \rightarrow a= b \pmod{m}$

Modular Arithmetic…

Theorem 4

a,b: integers, m: positive integer

a and b are congruent modulo m if and only if there is an integer k such that a = b + km

Proof

(1) $a \equiv b \pmod{m} \rightarrow a = b + km$

$a \equiv b \pmod{m} \rightarrow m \mid a-b \rightarrow a-b = km$ { from definition of division}

(2) $a = b + km \rightarrow a \equiv b \pmod{m}$

$a = b + km \rightarrow a-b=km \rightarrow m \mid a-b \rightarrow a \equiv b \pmod{m}$

Modular Arithmetic…

Theorem 5

m: positive integer

$a \equiv b \pmod{m} \wedge c \equiv d \pmod{m} \rightarrow$

$a+c \equiv b+d \pmod{m} \wedge ac \equiv bd \pmod{m}$

Proof : See page 204

Corollary 2:

m : positive integer, a,b : integers

$(a+b) \bmod m = ((a \bmod m) + (b \bmod m)) \bmod m$

$ab \bmod m = ((a \bmod m)(b \bmod m)) \bmod m$

Proof: page 205

Applications of Congruences

Hashing Function: H(k) = k mod m

Using in searching data tin memory.

k: data searched, m : memory block

Examples:

H(064212848) mod 111= 14

H(037149212) mod 111= 65

Collision: $H(k_1) = H(k_2)$. For example, H(107405723) = 14

Applications of Congruences

Pseudo-random Numbers $x_{n+1}=(ax_n+c) \bmod m$

a: multiplier, c: increment, x_0: seed

with $2 \leq a<m$, $0 \leq c<m$, $0 \leq x_0<m$

Examples:

$m=9 \rightarrow$ random numbers: 0..8

$a= 7$, $c=4$, $x_0=3$

Result: Page 207

Applications of Congruences

Cryptography: letter 1 $\rightarrow$ letter 2

Examples: shift cipher with k f(p) =(p+k) mod 26

→ f^{-1}(p)=(p-k) mod 26

Sender: (encoding)

Message: "ABC" , k=3

Using f(p) = (p+3) mod 26 // 26 characters

ABC ➔ 0 1 2 → 3 4 5 ➔ DEF

Receiver: (decoding)

DEF ➔ 3 4 5

Using f^{-1}(p) = (p-3) mod 26

3 4 5 ➔ 0 1 2 ➔ ABC

3.5- Primes and Greatest Common Divisors

Definition 1:

A positive integer p greater than 1 is called ***prime*** if the only positive factors are 1 and p

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

Examples:

Primes: 2 3 5 7 11

Composites: 4 6 8 9

Finding very large primes: tests for supercomputers

Primes…

Theorem 1- The fundamental theorem of arithmetic:

Every positive integer greater than 1 can be written uniquely as a prime or as the product of two or more primes where the prime factors are written in order of nondecreasing size

Examples:

Primes: 37

Composite: 100 = 2.2.5.5 = $2^2 5^2$

999 = 3.3.3.37 = $3^3 37$

Primes…

Converting a number to prime factors

Examples: 7007

Try it to 2,3,5 : 7007 can not divided by 2,3,5

7007 : 7

1001 : 7

143: 11

13: 13

0

→ $7007 = 7^2.11.13$

Primes…

Theorem 2: If n is a composite, then n has a prime **divisor** less than or equal to $\sqrt{n}$

Proof:

n is a composite ➔ n = ab in which a or b is a prime

If $(a > \sqrt{n} \wedge b > \sqrt{n}) \rightarrow ab > n$ ➔ false

➔ Prime divisor of n less than or equal to $\sqrt{n}$

Primes…

Theorem 3: There are infinite many primes

Proof: page 212

Theorem 4: The prime number theorem:

The ratio of the number of primes not exceeding x and x/ln x approaches 1 and grows with bound (ln x: natural logarithm of x)

See page 213

Example:

x= 10^{1000} → The odds that an integer near 10^{1000} is prime are approximately 1/ln 10^{1000} ~ 1/2300

Conjectures and Open Problems About Primes

See pages: 214, 215

- 3x + 1 conjecture
- Twin prime conjecture: there are infinitely many twin primes

Greatest Common Divisors and Least Common Multiples

Definition 2:

Let a, 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.

Notation: gcd(a,b)

Example: gcd(24,36)=?

Divisors of 24: 2 3 4 6 8 12 = $2^3 3^1$

Divisors of 36: 2 3 4 6 9 12 18 = $2^2 3^2$

gcd(24,36)=12 = $2^2 3^1$ // Get factors having minimum power

Greatest Common Divisors and Least Common Multiples

Definition 3:

The integers a, b are *relatively prime* if their greatest common divisor is 1

Example:

gcd(3,7)=1 ➔ 3,7 are relatively prime

gcd (17,22)=1 ➔ 17,22 are relatively prime

gcd(17,34) = 17 ➔ 17, 34 are **not** relatively prime

Greatest Common Divisors and Least Common Multiples

Definition 4:

The integers $a_1,a_2,a_3,\ldots,a_n$ are *pairwise relatively prime* if $\gcd(a_i,a_j)=1$ whenever $1 \le i<j \le n$

Example:

7 10 11 17 23 are pairwise relatively prime

7 10 11 16 24 are **not** pairwise relatively prime

➔ Adjacent number of every composite in sequence must be a prime.

Greatest Common Divisors and Least Common Multiples

Definition 5:

The Least common multiple of the positive integer a and b is the smallest integer that is divisible by both a and b

Notation: lcm(a,b)

Example:

lcm(12,36) = 36   lcm(7,11) = 77

lcm $(2^3 3^5 7^2, 2^4 3^3) = 2^4 3^5 7^2$

$2^3 3^5 7^2, 2^4 3^3 7^0 \rightarrow 2^4 3^5 7^2$ // get maximum power

Greatest Common Divisors and Least Common Multiples

Theorem 5:

Let a, b be positive integers then

ab= gcd(a,b). lcm(a,b)

Example: gcd(8, 12) = 4 lcm(8,12)=24 ➔ 8.12 = 4.24

Proof: Based on analyzing a, b to prime factors to get gcd(a,b) and lcm(a,b)

➔ ab=gcd(a,b). lcm(a,b)

3.6- Integers and Algorithms

- Representations of Integers
- Algorithms for Integer Operations
- Modular Exponentiation
- Euclid Algorithm

Fpt University

Representations of Integers

Theorem 1:

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, $a_0, a_1, a_2, \ldots, a_k$ are nonnegative integers less than b and $a_k \neq 0$

Proof: page 219

Example: $(245)_8 = 2.8^2 + 4.8+5 = 165$

Common Bases Expansions: Binary, Octal, Decimal, Hexadecimal

Finding expansion of an integer: Pages 219, 220, 221

Algorithm 1: Constructing Base b Expansions

```
Procedure base b expansion ( n: positive integer)
    q:=n
    k:=0
    while q $\neq$ 0
    begin
        $a_k$ :=q mod b
        q := $\lfloor q/b \rfloor$
        k := k +1
    end { The base b expansion of n is $(a_{k-1}a_{k-2}\ldots a_1a_0)$}
```

Algorithms for Integer Operations

Algorithm 2: Addition integers in binary format

Algorithm 3: Multiplying integers in binary format

Algorithm 4: Computing div and mod integers

Algorithm 5: Modular Exponentiation

Algorithm 2: Adding of Integers

Complexity: O (n)

$$\begin{array}{rccccc}
 & 1 & 1 & 1 & 0 & 0 \\
 & & 1 & 1 & 1 & 0 \text{ (a)} \\
+ & & 1 & 0 & 1 & 1 \text{ (b)} \\
\hline
 & 1 & 1 & 0 & 0 & 1 \text{ (s)}
\end{array}$$

```
procedure add ( a,b: integers)
{  a: $(a_{n-1}a_{n-2}\ldots a_1a_0)_2$          b: $(b_{n-1}b_{n-2}\ldots b_1b_0)_2$ }
c:=0
for  j:=0  to n-1
Begin
    d:= $\lfloor(a_j+b_j+c)/2\rfloor$   // next carry of next step
    $s_j= a_j+b_j+c - 2d$    // result bit
    c:=d              // updating new carry to next step
End
$s_n= c$ // rightmost bit of result
{ The binary of expansion of the sum is $(s_ns_{n-1}\ldots s_1s_0)$}
```

Algorithm 3: Multiplying Integers

Complexity: O (n^2)

```
      1 1 0 (a)
X     1 0 1 (b)
      1 1 0
+   0 0 0 0
  1 1 0 0 0
  1 1 1 1 0 (p)
```

```
procedure multiply ( a,b: integer)
{ a: $(a_{n-1}a_{n-2}\ldots a_1a_0)_2$      b: $(b_{n-1}b_{n-2}\ldots b_1b_0)_2$ }
for j:= 0 to n-1
begin
    if $b_j$ =1 then $c_j$ := a shifted j places
end
{ $c_0, c_1, \ldots, c_{n-1}$  are the partial products }
p := 0
for j:= 0 to n-1
    p:=p+$c_j$
{p is the value of ab}
```

Complexity: O (n)

Algorithm 4: Computing div and mod

```
procedure division ( a: integer; d: positive integer)
 q:=0
 r:= |a|
while r ≥ d  {quotient= number of times  of successive subtractions}
   begin
      r:= r-d
      q := q+1
   end
If a<0 and r>0  then  {updating remainder when a<0}
   begin
      r:= d-r
     q := -(q+1)
   end
{ q = a div d is the quotient, r=a mod d is the remainder}
```

Algorithm 5: Modular Exponentiation

```
{ $b^n$ mod m = ? . Ex: $3^{644}$ mod 645 = 36 }
procedure mod_ex ( b: integer, n=$(a_{k-1}a_{k-2}\ldots a_1a_0)_2$, m: positive integer)
x:=1
power := b mod m
for i:=0 to k-1
begin
    if $a_i$=1 then x:= (x.power) mod m
    power := (power.power) mod m
end
{ x equals $b^n$ mod m }
```

Corollary 2: ab mod m = ((a mod m)(b mod m)) mod m

b^n mod m = result of successive factor_i mod m

The Euclidean Algorithm

Lemma: Proof: page 228

Let a= bq+r, where a, b, q, r are integers. Then gcd(a,b) = gcd(b,r)

Example: 287 = 91.3 + 14 ➔gcd(287,91) =gcd(91,14)= 7

```
procedure  gcd(a,b: positive integer)
 x:=a
 y:=b
 while y $\neq$ 0
  begin
    r := x mod y
    x:=y
    y:= r
   end  {gcd(a,b) is x}
```

Summary

- Algorithms
- The Growth of Functions
- Complexity of Algorithms
- The Integers and Division
- Primes and Greatest Common Divisors
- Integers and Algorithms

Thanks