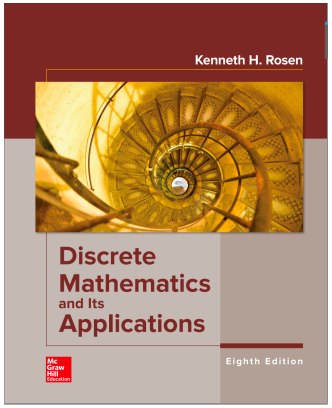
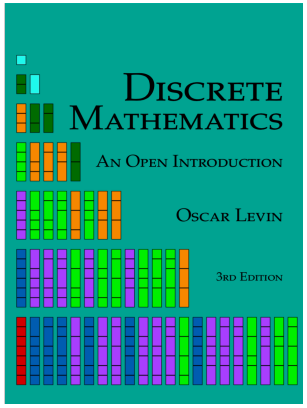
Vietnam National University of HCMC
International University
School of Computer Science and Engineering





Number Theory

Assoc. Prof. Dr. Nguyen Van Sinh

nvsinh@hcmiu.edu.vn

Outline

- Integers, Division and Primes
- Sequence and Summation
- Algorithms and Complexity

Refer: Chapter 4 in the textbook

Part 1

Integers, Division and Primes

Introduction:

Historically, *number theory* has been a beautiful area of study in pure mathematics. However, in modern times, number theory is very important in the area of security and its application in Computer Science.

Algorithms heavily depend on modular arithmetic, and our ability to deal with large integers.

We need appropriate techniques to deal with such algorithms.

Divisors

Define: let a, b and c be integers such that

$$a = b \cdot c$$

Then b and c are said to **divide** (or are **factors**) of a, while a is said to be a multiple of b (as well as of c). The pipe symbol "|" denotes "divides" so the situation is summarized by:

$$b|a \wedge c|a$$

In C++/Java, "b|a" is implemented as "a % b ==0"

Examples

77 | 7: false bigger number can't divide smaller positive number

7 | 77: true because $77 = 7 \cdot 11$

24 | 24: true because $24 = 24 \cdot 1$

1 | 2: true, 1 divides everything.

2 | 1: false.

0 | 24: false, only 0 is divisible by 0

24 | 0: true, 0 is divisible by every number ($0 = 24 \cdot 0$)

Divisor Theorem

- Theorem: Let a, b and c be integers. Then:
  1. $a|b \wedge a|c \rightarrow a|(b+c)$
  2. $a|b \rightarrow a|bc$
  3. $a|b \wedge b|c \rightarrow a|c$
- Proof: see 3-36, textBook
- Example:
  1. $17|34 \wedge 17|170 \rightarrow 17|204$
  2. $17|34 \rightarrow 17|340$ //2.(340=34x10)
  3. $6|12 \wedge 12|144 \rightarrow 6|144$

Prime Numbers

Define: A positive integer n > 1 is called Prime if it is only divisible by 1 and itself. A positive integer n > 1 is not Prime is called composite.

Question: Which of the following are primes?

- *0, 1, 2, …., 20. (from 1 to 20)*
- *Think about finding all primes from 1 to n?*

A theorem

- Theorem: Every positive integer n > 1 can be written uniquely as a prime or as a product of two or more primes where the prime factors are written in order of non decreasing size:

- Example: The prime factorizations of 100, 641, 999, 1024:

$100 = 2.2.5.5 = 2^2 . 5^2$

$641 = 641$

$999 = 3.3.3.37 = 3^3 . 37$

$1024 = 2.2.2.2.2.2.2.2.2.2 = 2^{10}$

- Question: Express each of the following number as a product of primes: 22, 200, 12, 17 ?

Testing Prime Numbers

Prime numbers are very important in encryption schemes. Essential to be able to verify if a number is prime or not. It turns out that this is quite a difficult problem. First try:

```
boolean isPrime(integer n)
   if ( n < 2 ) return false
   for(i = 2 to n -1)
      if ( i |n )          // "divides"! not disjunction
         return false
   return true
```

Application of Prime Numbers

RSA Algorithm Example:
https://en.wikipedia.org/wiki/RSA_(cryptosystem)

- Choose two distinct prime numbers: p = 3 and q = 11
- Compute n = p * q = 3 * 11 = 33
- Compute φ(n) = (p - 1) * (q - 1) = 2 * 10 = 20
- Choose any e: 1 < e < φ(n), e and n are coprime. Let e = 7
- Compute a value for d such that (d * e) % φ(n) = 1. One solution is d = 3 [(3 * 7) % 20 = 1]
- Public key is (e, n) => (7, 33)
- Private key is (d, n) => (3, 33)
- The encryption of $m = 2$ is $c = 2^7 \% 33 = 29$
- The decryption of $c = 29$ is $m = 29^3 \% 33 = 2$

Testing Prime Numbers in C++

```
int i, N, count=0;
//input N
for(i=1; i<=N; i++)
    {
        if(N % i == 0)
        {
            count++;
        }
    }
    if(count==2)
    {
        cout<<“Prime number \n";
    }
    else
    {
        cout<<“Not a prime number \n";
    }
```

Result:

Input: N=5
Prime number

Input: N= 8
Not a prime number

Time Complexity

The previous algorithm has a time complexity O(n) (assuming that a|b can be tested in O(1) time). For an 8-digit decimal number, it is thus $O(10^8)$. This is terrible. Can we do better?

Yes! Try only smaller prime numbers as divisors.

In fact, only need to try to divide by prime numbers no larger than $\sqrt{n}$ as we'll see next:

Primality testing theorem

LEMMA: If n is a composite, then its smallest prime factor is $\leq \sqrt{n}$

Proof (by contradiction). Suppose the smallest prime factor p is greater than $\sqrt{n}$

Then n = p.q where q > p and p > $\sqrt{n}$

This is a contradiction, since the right hand side > n.

(see page ? in the textbook for detail)

A Fundamental Theorem

THM: Any number $n \geq 2$ is expressible as as a unique product of 1 or more prime numbers.

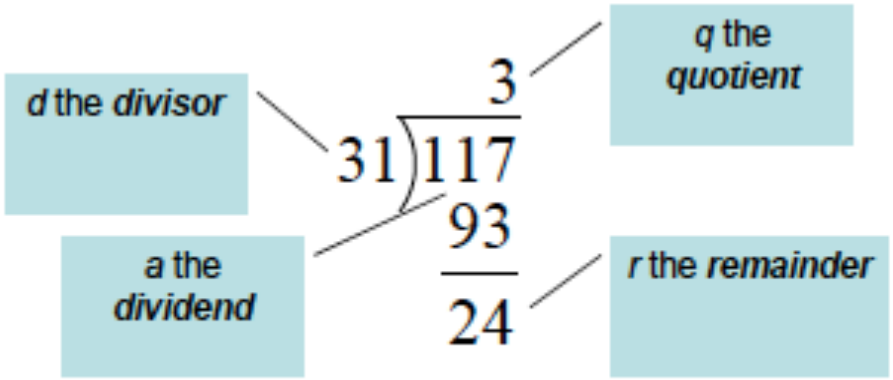
Note: prime numbers are considered to be "products" of 1 prime.

We'll need induction and some more number theory tools to prove this.

Q: Express each of the following number as a product of primes: 22, 100, 12, 17

Division

Remember long division?



$117 = 31 \cdot 3 + 24$

$a = dq + r$

Division

THM: Let a be an integer, and d be a positive integer. There are unique integers q, r with $r \in \{0,1,2,\ldots,d\text{-}1\}$ satisfying

$$a = dq + r$$

The proof is a simple application of long-division. The theorem is called the ***division algorithm*** though really, it's long division that's the algorithm, not the theorem.

Greatest Common Divisor

DEF Let a,b be integers, not both zero. The ***greatest common divisor*** of a and b (or $\gcd(a,b)$) is the biggest number d which divides both a and b.

Equivalently: $\gcd(a,b)$ is smallest number which divisibly by any x dividing both a and b.

DEF: a and b are said to be ***relatively prime*** if $\gcd(a,b) = 1$, so no prime common divisors.

Greatest Common Divisor

Q: Find the following gcd's:

1. gcd(11,77)
2. gcd(33,77)
3. gcd(24,36)
4. gcd(24,25)

Q: Compute gcd (36, 54, 81)

Euclid's gcd Algorithm

```
procedure gcd (a, b)
x:= a; y := b
while y ≠ 0
        begin
                r:= x mod y
                x:= y
                y:= r
        end

The gcd of (a, b) is x.
```

Let a = 12, b= 21

gcd (21, 12)
= gcd (12, 9)
= gcd (9, 3)

Since 9 mod 3 = 0
The gcd is 3

The mod Function

A: Compute

1. 113 **mod** 24:

$$\begin{array}{r} 4 \\ 24\overline{)113} \\ \underline{96} \\ 17 \end{array}$$

2. -29 **mod** 7

(mod) Congruence

DEF: Let a, a' be integers and b be a positive integer. We say that a is congruent to a' modulo b (denoted by $a \equiv a' \pmod{b}$) iff $b \mid (a - a')$.

Equivalently: $a \bmod b = a' \bmod b$

Q: Which of the following are true?

1. $3 \equiv 3 \pmod{17}$
2. $3 \equiv -3 \pmod{17}$
3. $172 \equiv 177 \pmod{5}$
4. $-13 \equiv 13 \pmod{26}$

(mod) Congruence

- $a \bmod b \equiv a \pmod{b}$
- Suppose $a \equiv a' \pmod{b}$ and $c \equiv c' \pmod{b}$
  Then:
  - $a+c \equiv (a'+c') \pmod{b}$
  - $ac \equiv a'c' \pmod{b}$
  - $a^k \equiv a'^k \pmod{b}$

Harder examples

Q: Compute the following.

1. 307^{1001} **mod** 102
2. $(-45 \cdot 77)$ **mod** 17
3. $\left(\sum_{i=4}^{23} 10^i\right) \bmod 11$

Harder examples

A: Use the previous identities to help simplify:

1. Using multiplication rules, before multiplying (or exponentiating) can reduce modulo 102:

307^{1001} **mod** 102 $\equiv 307^{1001} \pmod{102}$

$\equiv 1^{1001} \pmod{102} \equiv 1 \pmod{102}$. Therefore, 307^{1001} **mod** 102 = 1.

A: Use the previous identities to help simplify:

2. Repeatedly reduce after each multiplication:

$(-45 \cdot 77)$ **mod** 17 $\equiv (-45 \cdot 77) \pmod{17}$

$\equiv (6 \cdot 9) \pmod{17} \equiv 54 \pmod{17} \equiv 3 \pmod{17}$.

Therefore $(-45 \cdot 77)$ **mod** 17 = 3.

Therefore: (-45.77) mod 17 ≡ (6.9) mod 17

Because (-45 + 77 = 32; 32 mod 17 = 15); similarly (6+9 = 15; 15 mod 17 = 15)

Congruence & Application

A linear congruence is of the form

$$ax \equiv b \pmod{m}$$

Where a, b, m are integers, and x is a variable.
To solve it, find all integers that satisfy this congruence

- Application of Congruence:
  - Hashing function
  - Cryptography
  - .. See 4.4 (textbook), page 290

Classwork!

What are the greatest common divisors of these pairs of integers?

a) $2^2 \cdot 3^3 \cdot 5^5$, $2^5 \cdot 3^3 \cdot 5^2$

b) $2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 13$, $2^{11} \cdot 3^9 \cdot 11 \cdot 17^{14}$

c) 17, 17^{17}

d) $2^2 \cdot 7$, $5^3 \cdot 13$

e) 0, 5

f) $2 \cdot 3 \cdot 5 \cdot 7$, $2 \cdot 3 \cdot 5 \cdot 7$

- What is Least Common Multiple (LCM)?
- Write a program in C/C++ to find GCD(of two and three integer numbers)
- Write a program in C/C++ to find LCM(of two and three integer numbers)

Part 2

Sequence and Summation

Sequence

A sequence is an ordered list of elements.

- A sequence is often given as
  - $a_1, a_2, \ldots, a_n, \ldots$
  - a_n is a term in the sequence.
- A sequence is actually a function *f* from a subset of **Z** to a set S
  - Usually from the positive or non-negative integers
  - a_n is the image of *n*: $f(n) = a_n$

Examples of Sequence

- $a_n = 3n$
  - The terms in the sequence are $a_1, a_2, a_3, \ldots$
  - The sequence $\{a_n\}$ is $\{ 3, 6, 9, 12, \ldots \}$

- $b_n = 2^n$
  - The terms in the sequence are $b_1, b_2, b_3, \ldots$
  - The sequence $\{b_n\}$ is $\{ 2, 4, 8, 16, 32, \ldots \}$

Examples of Sequence

- The difference is in how they grow
- Arithmetic sequences increase by a constant *amount*
  - $a_n = 3n$
  - The sequence is { 3, 6, 9, 12, … }
  - Each number is 3 more than the last
  - Of the form: $f(x) = dx + a$
- Geometric sequences increase by a constant *factor*
  - $b_n = 2^n$
  - The sequence is { 2, 4, 8, 16, 32, … }
  - Each number is twice the previous
  - Of the form: $f(x) = ar^x$

Examples of Sequence

- Not all sequences are arithmetic or geometric sequences.
- An example is Fibonacci sequence

  - $F_n = F_{n-1} + F_{n-2}$, where the first two terms are 1
    - Alternative, $F(n) = F(n\text{-}1) + F(n\text{-}2)$
  - Each term is the sum of the previous two terms
  - Sequence: { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … }
  - This is the Fibonacci sequence

Examples of Sequence

- Sequences can be neither geometric or arithmetic
  - $F_n = F_{n-1} + F_{n-2}$, where the first two terms are 1
    - Alternative, $F(n) = F(n\text{-}1) + F(n\text{-}2)$
  - Each term is the sum of the previous two terms
  - Sequence: { 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, … }
  - This is the Fibonacci sequence
  - Full formula: $F(n) = \dfrac{\left(1+\sqrt{5}\right)^n - \left(1-\sqrt{5}\right)^n}{\sqrt{5}\cdot 2^n}$

More on Fibonacci Sequence

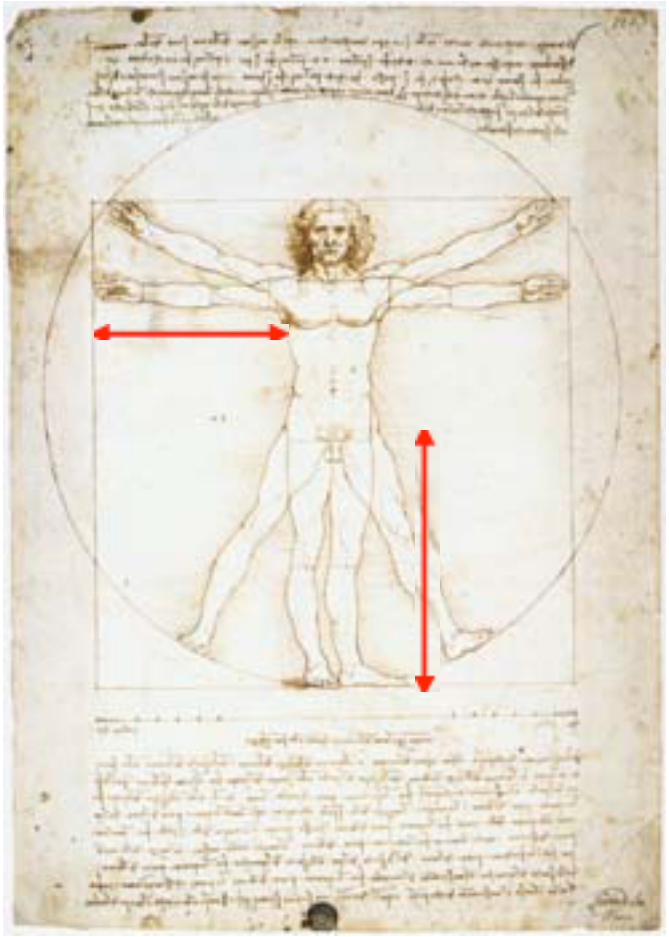
- As the terms increase, the ratio between successive terms approaches 1.618

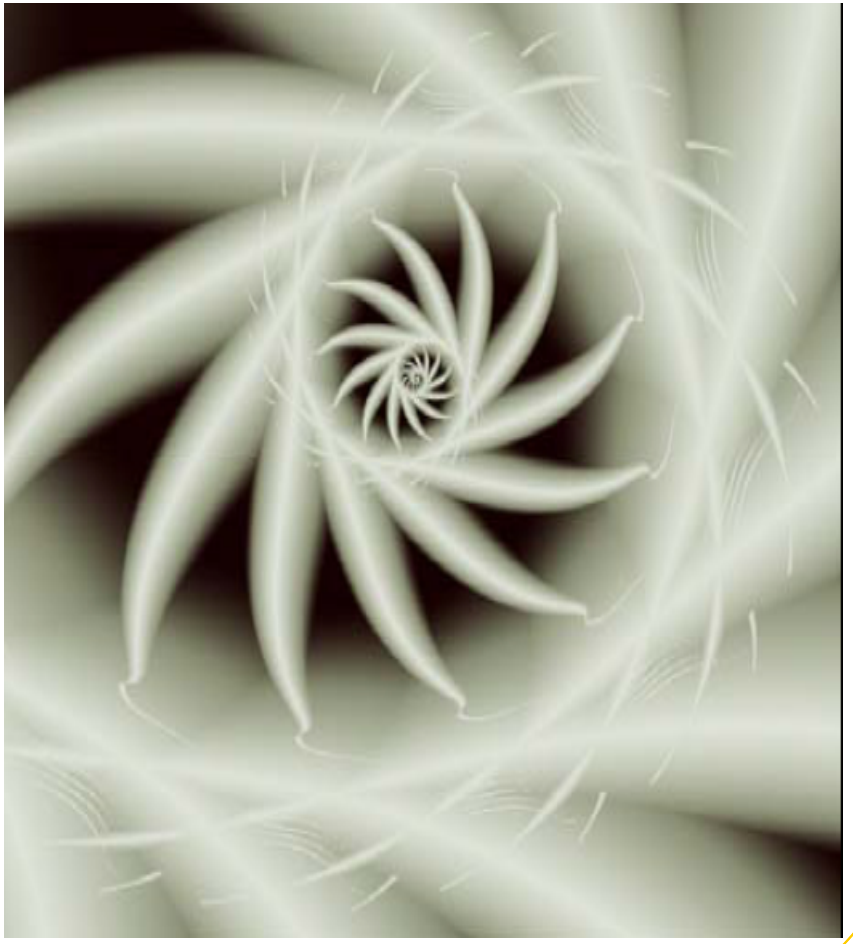
$$\lim_{n\to\infty}\frac{F(n+1)}{F(n)}=\phi=\frac{\sqrt{5}+1}{2}=1.618933989$$

- This is called the "golden ratio"
  - Ratio of human leg length to arm length
  - Ratio of successive layers in a conch shell

- Reference: http://en.wikipedia.org/wiki/Golden_ratio

Examples of Golden Ratio





Sequence Formula

a) 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, …
   - The sequence alternates 1's and 0's, increasing the number of 1's and 0's each time

b) 1, 2, 2, 3, 4, 4, 5, 6, 6, 7, 8, 8, …
   - This sequence increases by one, but repeats all even numbers once

c) 1, 0, 2, 0, 4, 0, 8, 0, 16, 0, …
   - The non-0 numbers are a geometric sequence (2^n) interspersed with zeros

d) 3, 6, 12, 24, 48, 96, 192, …
   - Each term is twice the previous: geometric progression
   - $a_n = 3*2^{n-1}$

Sequence Formula

e) 15, 8, 1, -6, -13, -20, -27, …
- Each term is 7 less than the previous term
- $a_n = 22 - 7n$

f) 3, 5, 8, 12, 17, 23, 30, 38, 47, …
- The difference between successive terms increases by on each time
- $a_1 = 3$, $a_n = a_{n-1} + n$
- $a_n = n(n+1)/2 + 2$

g) 2, 16, 54, 128, 250, 432, 686, …
- Each term is twice the cube of n
- $a_n = 2*n^3$

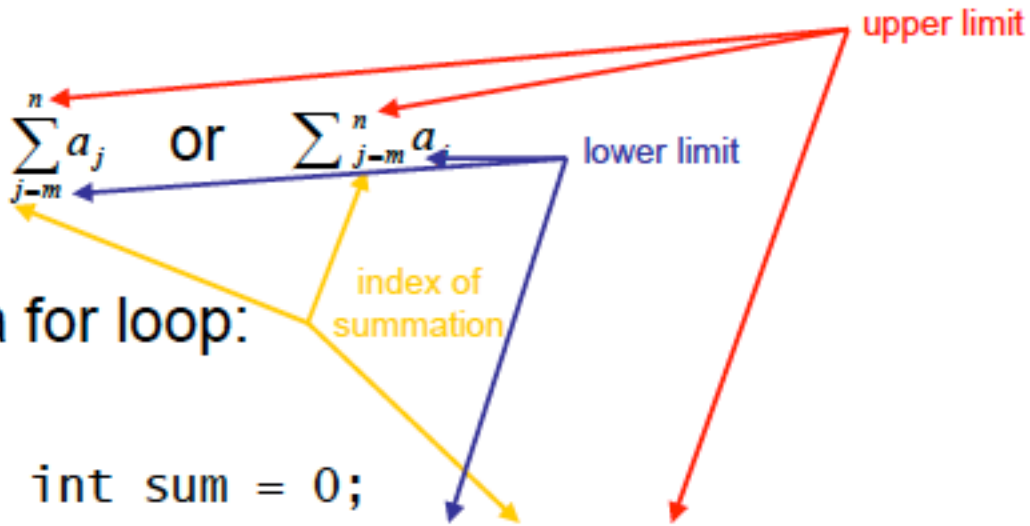
h) 2, 3, 7, 25, 121, 721, 5041, 40321
- Each successive term is about n times the previous
- $a_n = n! + 1$

Some useful sequences

- n^2 = 1, 4, 9, 16, 25, 36, …
- n^3 = 1, 8, 27, 64, 125, 216, …
- n^4 = 1, 16, 81, 256, 625, 1296, …
- 2^n = 2, 4, 8, 16, 32, 64, …
- 3^n = 3, 9, 27, 81, 243, 729, …
- $n!$ = 1, 2, 6, 24, 120, 720, …

Summation

- A summation:



- is like a for loop:

```
int sum = 0;
for ( int j = m; j <= n; j++ )
      sum += a(j);
```

Evaluating sequences

$\sum_{k=1}^{5}(k+1)$

- $2 + 3 + 4 + 5 + 6 = 20$

$\sum_{j=0}^{4}(-2)^j$

- $(-2)^0 + (-2)^1 + (-2)^2 + (-2)^3 + (-2)^4 = 11$

$\sum_{i=1}^{10}3$

- $3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 = 30$

$\sum_{j=0}^{8}\left(2^{j+1}-2^j\right)$

- $(2^1-2^0) + (2^2-2^1) + (2^3-2^2) + \ldots (2^{10}-2^9) = 511$
  - Note that each term (except the first and last) is cancelled by another term

Arithmetic Series

Consider an arithmetic series $a_1, a_2, a_3, \ldots, a_n$. If the common difference $(a_{i+1} - a_1) = d$, then we can compute the k^{th} term a_k as follows:

$$a_2 = a_1 + d$$

$$a_3 = a_2 + d = a_1 + 2d$$

$$a_4 = a_3 + d = a_1 + 3d$$

$$a_k = a_1 + (k-1).d$$

Evaluating sequences

- Let S = { 1, 3, 5, 7 }

- What is $\Sigma_{j \in S} j$
  - $1 + 3 + 5 + 7 = 16$
- What is $\Sigma_{j \in S} j^2$
  - $1^2 + 3^2 + 5^2 + 7^2 = 84$
- What is $\Sigma_{j \in S} (1/j)$
  - $1/1 + 1/3 + 1/5 + 1/7 = 176/105$
- What is $\Sigma_{j \in S} 1$
  - $1 + 1 + 1 + 1 = 4$

Sum of arithmetic series

Given n numbers, a_1, a_2, …, a_n with common difference d, i.e. $a_{i+1} - a_i = d$.

What is a simple closed form expression of the sum?

$$S_n = \sum_{i=1}^{n} a_i$$

$$S_n = a_1 + (a_1 + d) + (a_1 + 2d) + \ldots\ldots + (a_1 + (n-2)d) + (a_1 + (n-1)d)$$

$$S_n = a_n + (a_n - d) + (a_n - 2d) + \cdots + (a_n - (n-2)d) + (a_n - (n-1)d)$$

Adding the equations together gives:

$$2S_n = n(a_1 + a_n)$$

Rearranging and remembering that $a_n = a_1 + (n-1)d$, we get:

$$S_n = \frac{n(a_1 + a_n)}{2} = \frac{n[2a_1 + (n-1)d]}{2}$$

Solve this

$1 + 2 + 3 + \ldots + n = ?$

[Answer: $n.(n+1) / 2$]

why?

Calculate $1^2 + 2^2 + 3^2 + 4^2 + \ldots + n^2$

[Answer $n.(n+1).(2n+1) / 6$]

Can you evaluate this?

$$\sum_{i=1}^{n} \frac{1}{k(k+1)}$$

Here is the trick. Note that

$$\frac{1}{k(k+1)} = \frac{1}{k} - \frac{1}{k+1}$$

Does it help?

Double Summation

- Like a nested for loop

$$\sum_{i=1}^{4}\sum_{j=1}^{3} ij$$

- Is equivalent to:

```
int sum = 0;
for ( int i = 1; i <= 4; i++ )
     for ( int j = 1; j <= 3; j++ )
          sum += i*j;
```

Useful summation formulae

TABLE 2 **Some Useful Summation Formulae.**

Sum	*Closed Form*
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1} - a}{r - 1}, r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x \rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty}, kx^{k-1}, \lvert x \rvert < 1$	$\frac{1}{(1-x)^2}$

Products

$$\prod_{i=1}^{n} a_i := a_1 \cdot a_2 \cdots a_n$$

$$\prod_{k=1}^{5} k^2$$

$$\prod_{k=1}^{n} \frac{k}{k+1}$$

$$\prod_{k=1}^{n} 2^k$$

Dealing with Products

Factorial defines a product:

$$n! = 1 \cdot 2 \cdot 3 \cdot n = \prod_{i=1}^{n} i$$

How to estimate n!?

Turn product into a sum taking logs:

$$\ln(n!) = \ln(1 \cdot 2 \cdot 3 \cdots (n-1) \cdot n)$$

$$= \ln 1 + \ln 2 + \cdots + \ln(n-1) + \ln(n)$$

$$= \sum_{i=1}^{n} \ln(i)$$

Example: solve the summation in C++

List n number of Fibonacci from 0:

```
//declare and assign the initial values for variables
int n, c, first = 0, second = 1, next;
   cout << "Input the number of Fibonacci series you want: " << endl;
   cin >> n;
   for ( c = 0 ; c < n ; c++ )
   {
      if ( c <= 1 )
         next = c;
      else
      {
         next = first + second;
         first = second;
         second = next;
      }
      cout << next << "\t";
   }
```

Part 3

Algorithms and Complexity

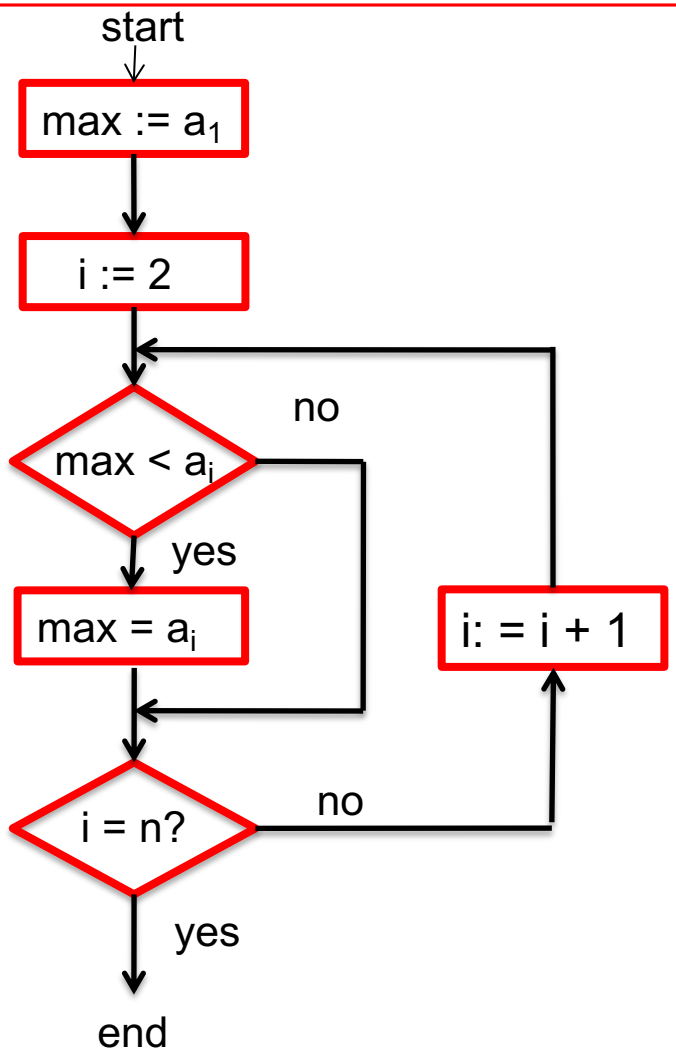
What is an algorithm

A finite set (or sequence) of precise instructions for performing a computation.

Example: Maxima finding

procedure *max* (*a1, a2, …, an*: integers)
max := *a1*
for i :=*2* to *n*
    if *max* < *a1* then *max* := a_i
return *max* {the largest element}

Flowchart for maxima finding



Given n elements, can you count the total number of operations?

Time complexity of algorithms

Measures the largest number of basic operations required to execute an algorithm.

Example: Maxima finding

procedure *max* (*a1, a2, …, an*: integers)
max := *a1*
1 operation
for i :=*2* to *n*
n-1 times
if *max* < *a1* then *max* := a_i
2 operations
return *max* {the largest element}

The total number of operations is 2n-1

Time complexity of algorithms

Example of linear search (Search x in a list $a_1\ a_2\ a_3 \ldots a_n$)

k := 1

(1 operation)

while k ≤ n do

{if x = a_k then *found* else k: = k+1}
(2n operations)

search failed

The maximum number of operations is 2n+1. If we are lucky, then search can end even in a single step.

Sorting algorithm

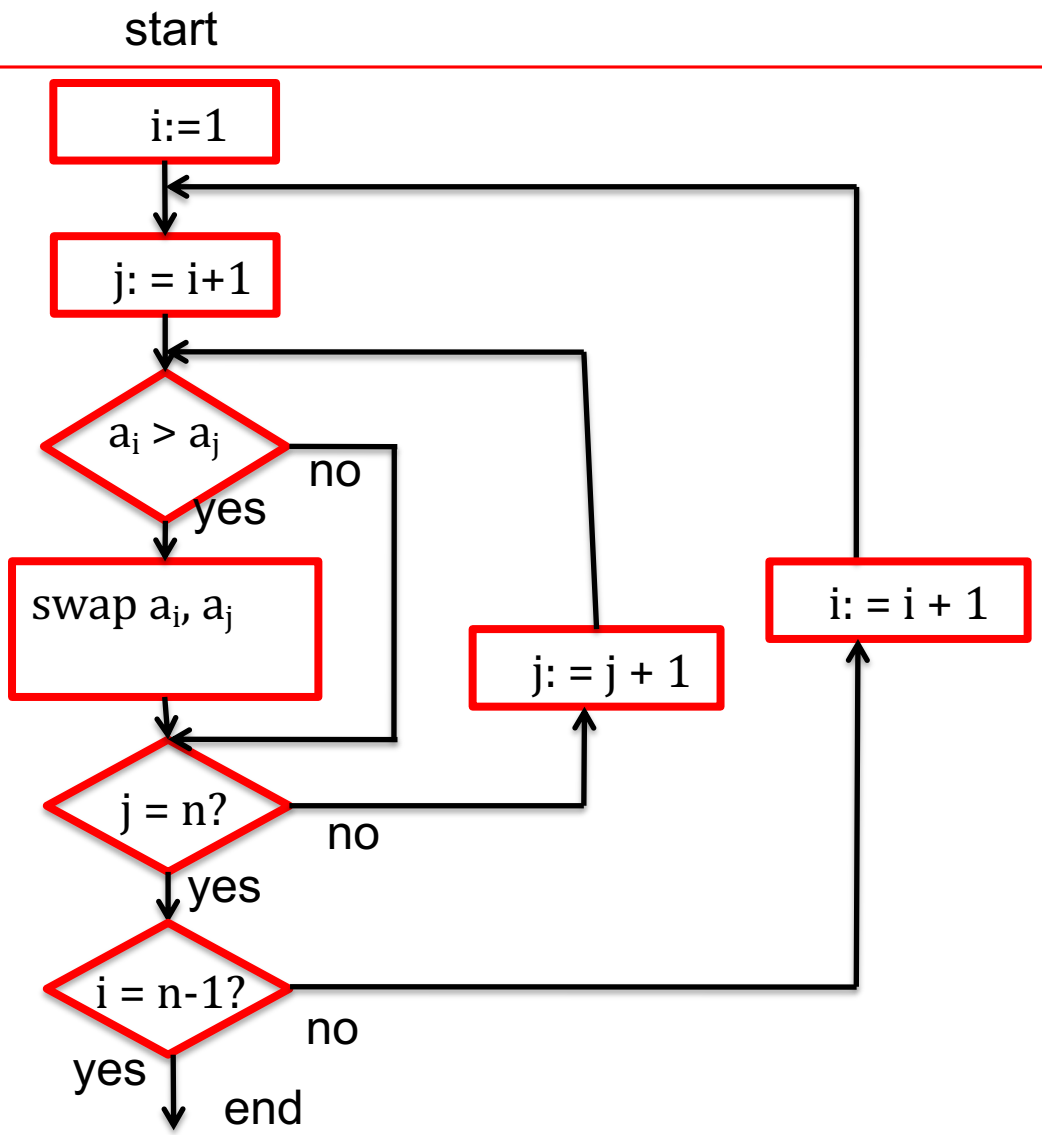
Sort a list $a_1\, a_2\, a_3 \ldots a_n$ in the ascending order

```
for i:= 1 to n-1
    for j:= i+1 to n

        if a_i > a_j then swap (a_i , a_j)
```

How many basic operations will you need here?

Example of a sorting algorithm



Given n elements, can you count the total number of operations?

Bubble Sort

Procedure bubblesort

{sort n integers $a_1, a_2, \ldots, a_n$ in ascending order}

for i:= 1 to n-1

for j:= 1 to n-i

if $a_j > a_{j+1}$ then swap (a_j, a_{j+1})

3	2	4	1	5		n-1 operations
2	3	1	4	5	(first pass)	n-2 operations
2	1	3	4	5	(second pass)	n-3perations
1	2	3	4	5	(third pass)	…
1	2	3	4	5	(fourth pass)	1

Bubble Sort

3	2	4	1	5		n-1 operations
2	3	1	4	5	(first pass)	n-2 operations
2	1	3	4	5	(second pass)	n-3 operations
1	2	3	4	5	(third pass)	…
1	2	3	4	5	(fourth pass)	1

The worst case time complexity is

$$(n-1) + (n-2) + (n-3) + \ldots + 2 + 1$$
$$= n(n-1)/2$$

The Big-O notation

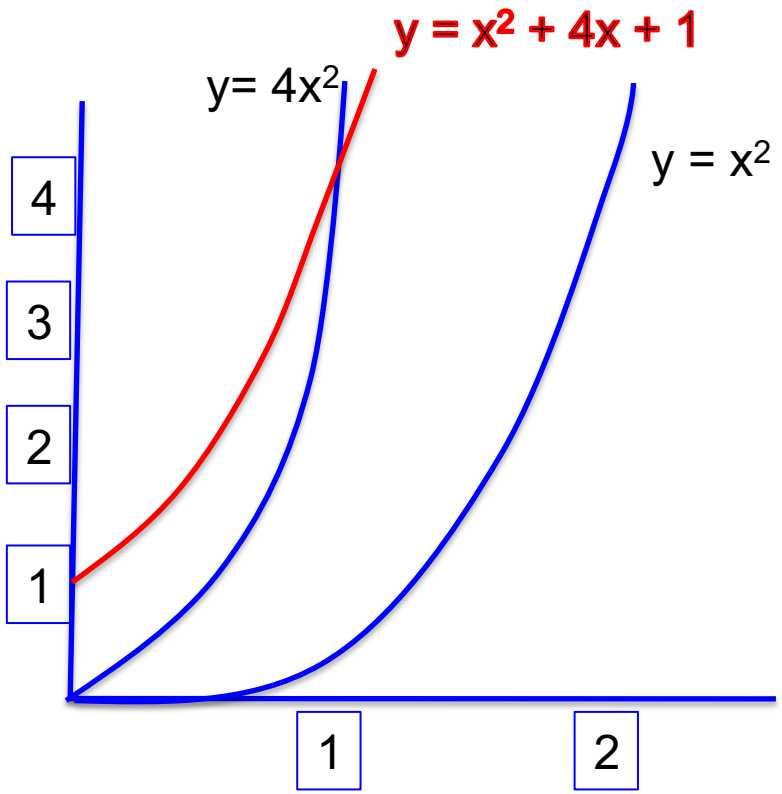
It is a measure of the growth of functions and often used to measure the complexity of algorithms.

DEF. Let f and g be functions from the set of integers (or real numbers) to the set of real numbers. Then f is O(g(x)) if there are constants **C** and **k**, such that

$$|f(x)| \leq C|g(x)| \qquad \text{for all } x > k$$

Intuitively, f(x) grows "slower than" some multiple of g(x) as x grows without bound. Thus O(g(x)) defines an upper bound of f(x).

The Big-O notation



$x^2 + 4x + 1 = O(x^2)$

Since = $4x^2 > x^2 + 4x + 1$

whenever $x > 1$, $4x^2$ defines an upper bound of the growth of $x^2 + 4x + 1$

Defines an upper bound of the growth of functions

The Big-Ω (omega) notation

DEF. Let f and g be functions from the set of integers (or real numbers) to the set of real numbers. Then f is Ω(g(x)) if there are constants C and k, such that

$$|f(x)| \geq C|g(x)| \quad \text{for all } x > k$$

Example. $7x^2 + 9x + 4$ is $\Omega(x^2)$, since $7x^2 + 9x + 4 \geq 1.\ x^2$ for all x

Thus Ω defines the lower bound of the growth of a function

Question. Is $7x^2 + 9x + 4$ $\Omega(x)$?

The Big-Theta (Θ) notation

DEF. Let f and g be functions from the set of integers (or real numbers) to the set of real numbers. Then f is Θ(g(x)) if there are constants C_1 and C_2 a positive real number k, such that

$C1.|g(x)| \leq |f(x)| \leq C2.|g(x)|$ for all $x > k$

Example. $7x^2 + 9x + 4$ is $\Theta(x^2)$,

since 1. $x^2 \leq 7x^2 + 9x + 4 \leq 8. x^2$ for all $x > 10$

Average case performance

EXAMPLE. Compute the average case complexity of the ***linear search*** algorithm.

a_1 a_2 a_3 a_4 a_5 ..... a_n

(Search for x from this list)

If x is the 1st element then it takes 3 steps

If x is the 2nd element then it takes 5 steps

If x is the ith element then it takes (2i + 1) steps

So, the average number of steps = 1/n (3+5+7+…+2n+1) = ?

Classification of complexity

Complexity	Terminology
$\Theta(1)$	Constant complexity
$\Theta(\log n)$	Logarithmic complexity
$\Theta(n)$	Linear complexity
$\Theta(n^c)$	Polynomial complexity
$\Theta(b^n)$ $(b>1)$	Exponential complexity
$\Theta(n!)$	Factorial complexity

We also use such terms when Θ is replaced by O (big-O)

Greedy Algorithms

In optimization problems, algorithms that use the best choice at each step are called greedy algorithms.

Example. Devise an algorithm for making change for n cents using quarters, dimes, nickels, and pennies using the least number of total coins?

Greedy Change-making Algorithm

Let $c_1, c_2, \ldots, c_r$ be the denomination of the coins, and $c_i > c_{i+1}$

for i:= 1 to r
  while $n \geq c_i$
    begin
      add a coin of value c_i to the change
      $n := n - c_i$
    end

Question. Is this optimal? Does it use the least number of coins?

Let the coins be 1, 5, 10, 25 cents. For making 38 cents, you will use

1 quarter
1 dime
3 cents

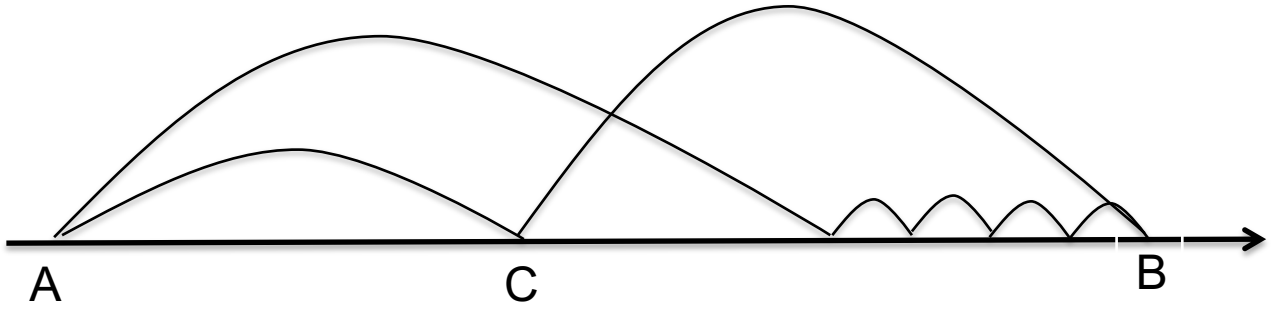
The total count is 5, and it is optimum.

Greedy Change-making Algorithm

But if you don't use a nickel, and you make a change for 30 cents using the same algorithm, the you will use 1 quarter and 5 cents (total 6 coins). But the optimum is 3 coins (use 3 dimes!)

So, greedy algorithms produce results, but the results may be sub-optimal.

Greedy Routing Algorithm



If you need to reach point B from point A in the fewest number of hops, Then which route will you take? If the knowledge is local, then you are Tempted to use a greedy algorithm, and reach B in 5 hops, although It is possible to reach B in only two hops.

Other classification of problems

- Problems that have polynomial worst-case complexity are called tractable. Otherwise they are called intractable.
- Problems for which no solution exists are known as unsolvable problems (like the halting problems). Otherwise they are called solvable.
- Many solvable problems are believed to have the property that no polynomial time solution exists for them, but a solution, if known, *can be checked in polynomial time*. These belong to the class NP (as opposed to the class of tractable problems that belong to class P)

The Halting Problems

The **Halting problem** asks the question.

Given a program and an input to the program, determine if the program will eventually stop when it is given that input.

Take a trial solution

- Run the program with the given input. If the program stops, we know the program stops.
- But if the program doesn't stop in a reasonable amount of time, then we cannot conclude that it won't stop. Maybe we didn't wait long enough!

Not decidable in general!