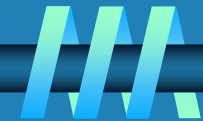


Fundamentals of the Analysis of Algorithm Efficiency

Analysis of algorithms



∞ Issues:

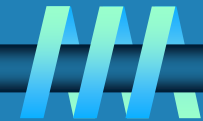
- **correctness**
- **time efficiency**
- **space efficiency**
- **optimality**

∞ Approaches:

- **theoretical analysis**
- **empirical analysis**

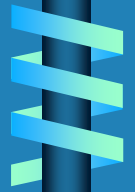
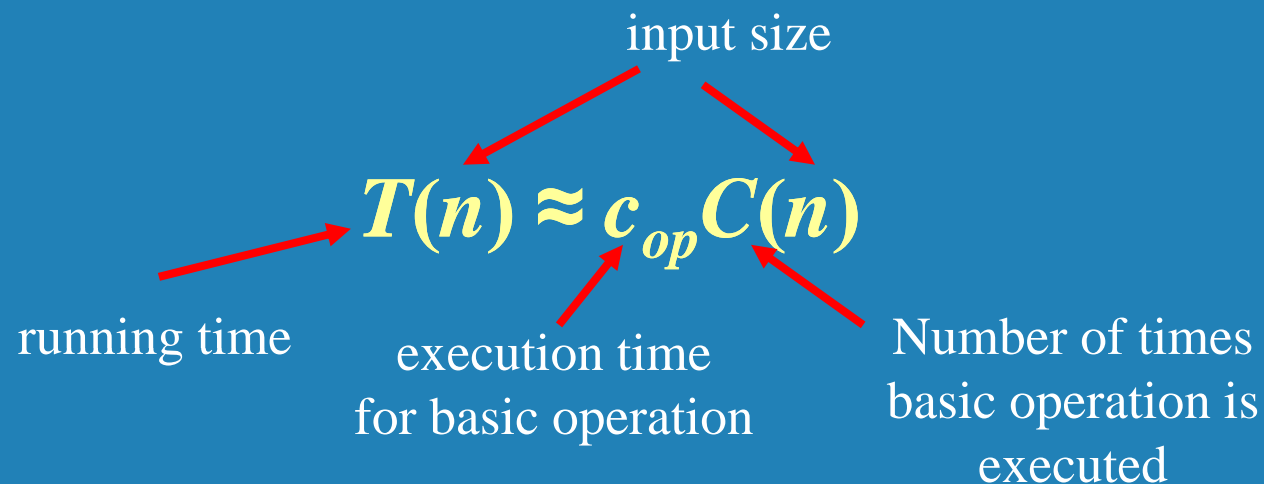


Theoretical analysis of time efficiency



Time efficiency is analyzed by determining the number of repetitions of the basic operation as a function of input size

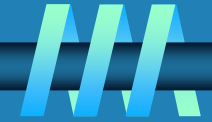
∞ Basic operation: the operation that contributes most towards the running time of the algorithm



Input size and basic operation examples

<i>Problem</i>	<i>Input size measure</i>	<i>Basic operation</i>
Searching for key in a list of n items	Number of list's items, i.e. n	Key comparison
Multiplication of two matrices	Matrix dimensions or total number of elements	Multiplication of two numbers
Checking primality of a given integer n	n 's size = number of digits (in binary representation)	Division
Typical graph problem	#vertices and/or edges	Visiting a vertex or traversing an edge

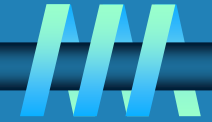
Empirical analysis of time efficiency



- ∞ **Select a specific (typical) sample of inputs**
- ∞ **Use physical unit of time (e.g., milliseconds)**
or
Count actual number of basic operation's executions
- ∞ **Analyze the empirical data**

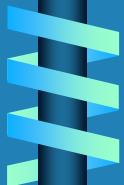


Best-case, average-case, worst-case

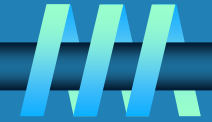


For some algorithms efficiency depends on form of input:

- ∞ **Worst case:** $C_{\text{worst}}(n)$ – maximum over inputs of size n
- ∞ **Best case:** $C_{\text{best}}(n)$ – minimum over inputs of size n
- ∞ **Average case:** $C_{\text{avg}}(n)$ – “average” over inputs of size n
 - Number of times the basic operation will be executed on typical input
 - NOT the average of worst and best case
 - Expected number of basic operations considered as a random variable under some assumption about the probability distribution of all possible inputs



Example: Sequential search



ALGORITHM *SequentialSearch*($A[0..n - 1]$, K)

//Searches for a given value in a given array by sequential search

//Input: An array $A[0..n - 1]$ and a search key K

//Output: The index of the first element of A that matches K

// or -1 if there are no matching elements

$i \leftarrow 0$

while $i < n$ **and** $A[i] \neq K$ **do**

$i \leftarrow i + 1$

if $i < n$ **return** i

else return -1

Ω **Worst case**

Ω **Best case**

Ω **Average case**



Types of formulas for basic operation's count

∞ Exact formula

e.g., $C(n) = n(n-1)/2$

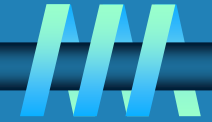
∞ Formula indicating order of growth with specific multiplicative constant

e.g., $C(n) \approx 0.5 n^2$

∞ Formula indicating order of growth with unknown multiplicative constant

e.g., $C(n) \approx cn^2$

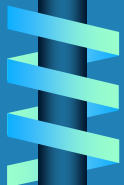
Order of growth



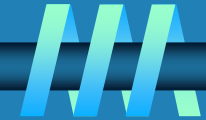
⌚ **Most important: Order of growth within a constant multiple as $n \rightarrow \infty$**

⌚ **Example:**

- **How much faster will algorithm run on computer that is twice as fast?**
- **How much longer does it take to solve problem of double input size?**



Values of some important functions as $n \rightarrow \infty$

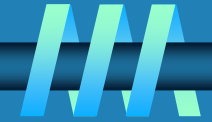


n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Table 2.1 Values (some approximate) of several functions important for analysis of algorithms

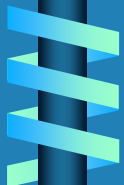


Asymptotic order of growth



A way of comparing functions that ignores constant factors and small input sizes

- $\mathcal{O}(g(n))$: class of functions $f(n)$ that grow no faster than $g(n)$
- $\Theta(g(n))$: class of functions $f(n)$ that grow at same rate as $g(n)$
- $\Omega(g(n))$: class of functions $f(n)$ that grow at least as fast as $g(n)$



Big-oh

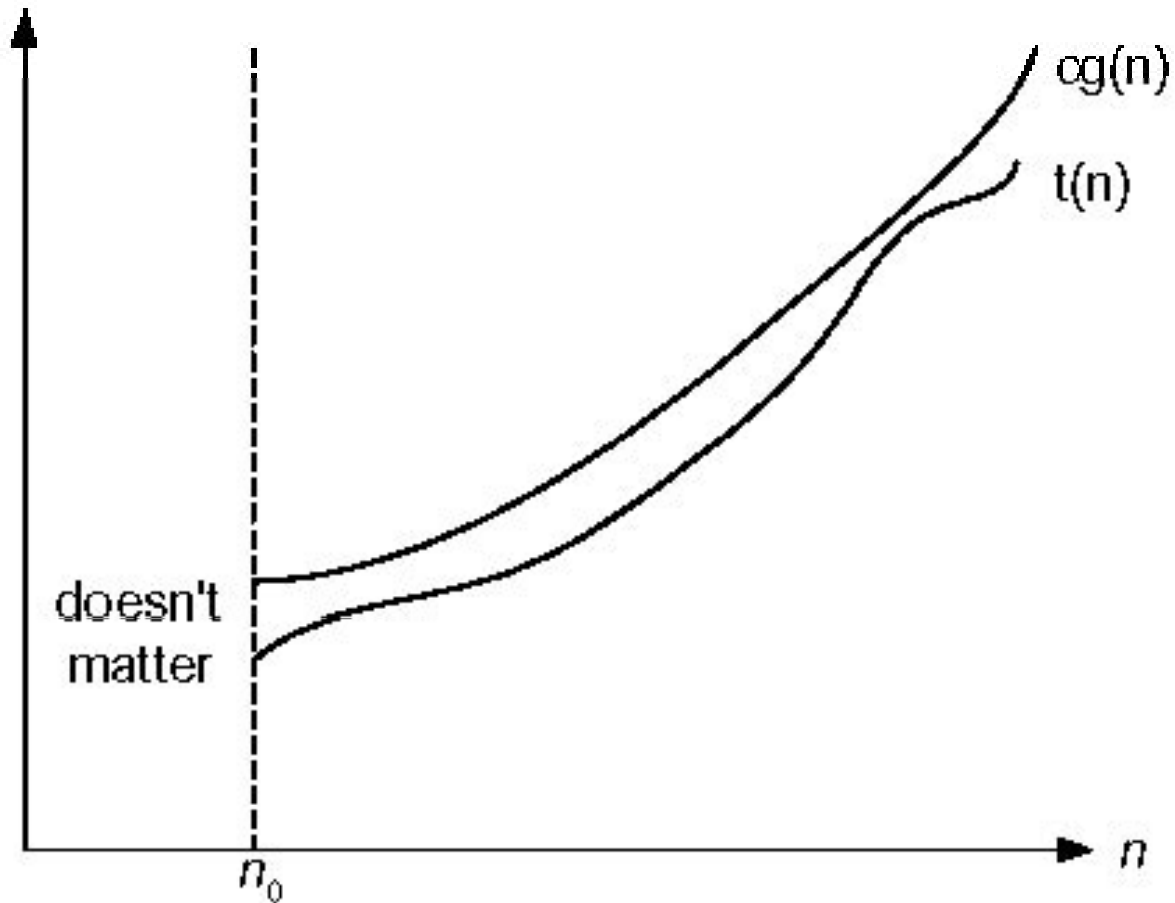
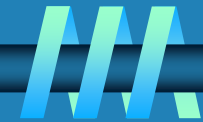


Figure 2.1 Big-oh notation: $t(n) \in O(g(n))$



Big-omega

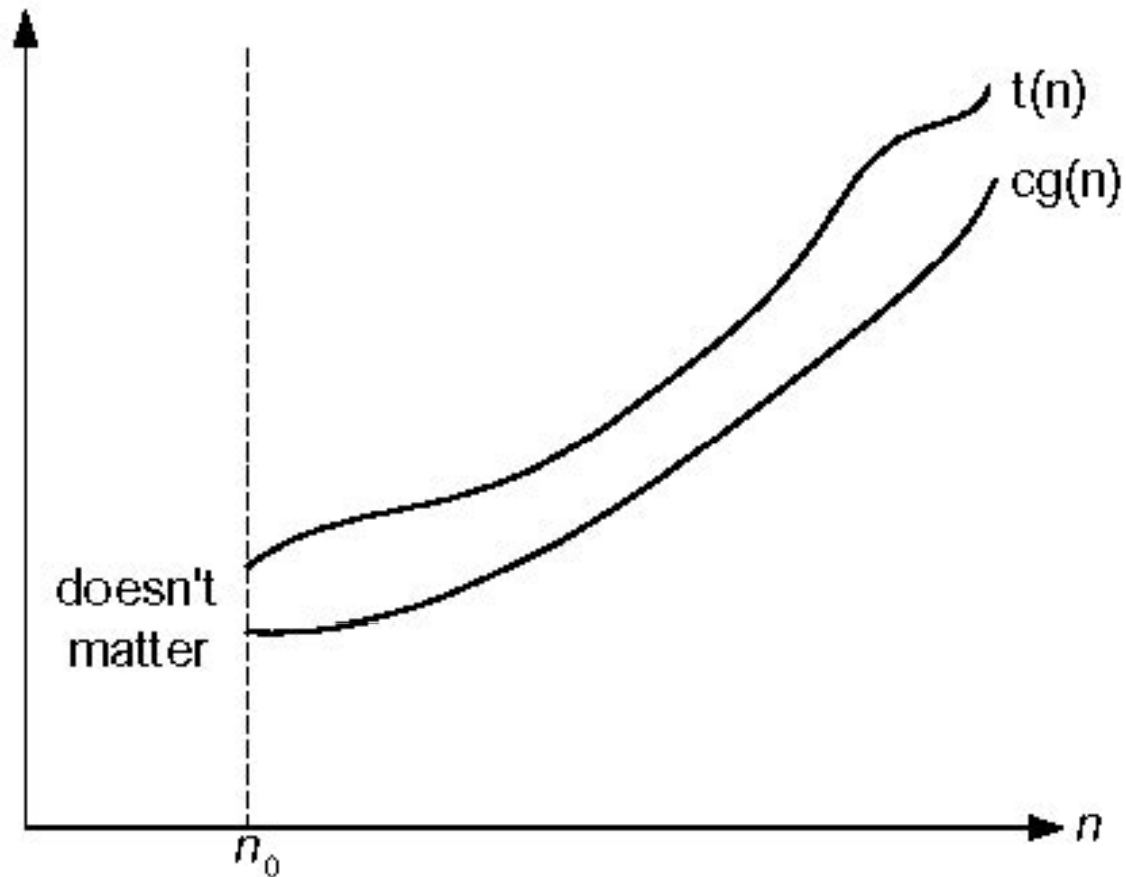
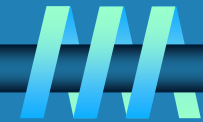
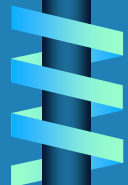


Fig. 2.2 Big-omega notation: $t(n) \in \Omega(g(n))$



Big-theta

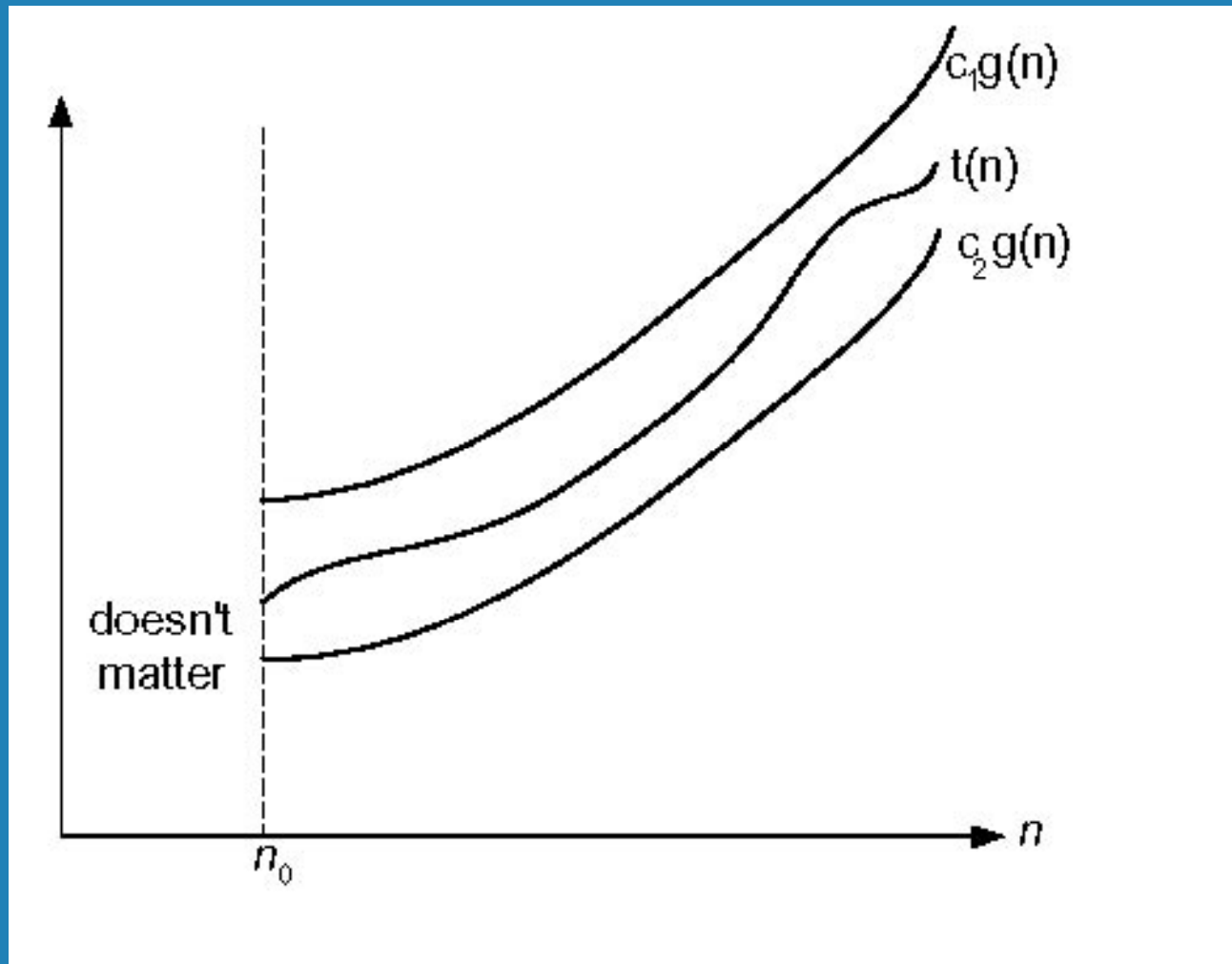
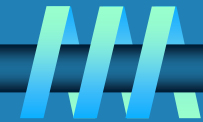
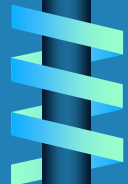
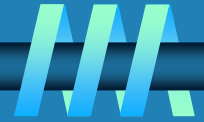


Figure 2.3 Big-theta notation: $t(n) \in \Theta(g(n))$



Establishing order of growth using the definition



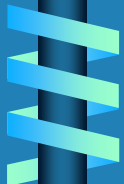
Definition: $f(n)$ is in $O(g(n))$ if order of growth of $f(n) \leq$ order of growth of $g(n)$ (within constant multiple), i.e., there exist positive constant c and non-negative integer n_0 such that

$$f(n) \leq c g(n) \text{ for every } n \geq n_0$$

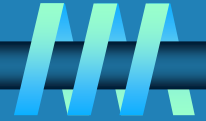
Examples:

⌚ $10n$ is $O(n^2)$

⌚ $5n+20$ is $O(n)$



Some properties of asymptotic order of growth



$$\Omega f(n) \in O(f(n))$$

$$\Omega f(n) \in O(g(n)) \text{ iff } g(n) \in \Omega(f(n))$$

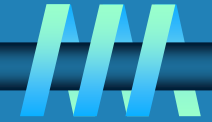
$$\Omega \text{ If } f(n) \in O(g(n)) \text{ and } g(n) \in O(h(n)), \text{ then } f(n) \in O(h(n))$$

Note similarity with $a \leq b$

$$\Omega \text{ If } f_1(n) \in O(g_1(n)) \text{ and } f_2(n) \in O(g_2(n)), \text{ then}$$
$$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$$



Establishing order of growth using limits



$$\lim_{n \rightarrow \infty} T(n)/g(n) = \begin{cases} 0 & \text{order of growth of } T(n) < \text{order of growth of } g(n) \\ c > 0 & \text{order of growth of } T(n) = \text{order of growth of } g(n) \\ \infty & \text{order of growth of } T(n) > \text{order of growth of } g(n) \end{cases}$$

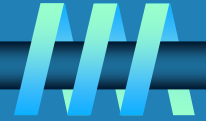
Examples:

• $10n$ vs. n^2

• $n(n+1)/2$ vs. n^2



L'Hôpital's rule and Stirling's formula



L'Hôpital's rule: If $\lim_{n \rightarrow \infty} f(n) = \lim_{n \rightarrow \infty} g(n) = \infty$ and the derivatives f' , g' exist, then

$$\lim_{n \rightarrow \infty} \frac{f(n)}{g(n)} = \lim_{n \rightarrow \infty} \frac{f'(n)}{g'(n)}$$

Example: $\log n$ vs. n

Example: 2^n vs. $n!$

Stirling's formula: $n! \approx (2\pi n)^{1/2} (n/e)^n$



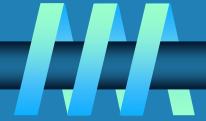
Orders of growth of some important functions

- ⌚ All logarithmic functions $\log_a n$ belong to the same class $\Theta(\log n)$ no matter what the logarithm's base $a > 1$ is
- ⌚ All polynomials of the same degree k belong to the same class: $a_k n^k + a_{k-1} n^{k-1} + \dots + a_0 \in \Theta(n^k)$
- ⌚ Exponential functions a^n have different orders of growth for different a 's
- ⌚ order $\log n < \text{order } n^\alpha \ (\alpha > 0) < \text{order } a^n < \text{order } n! < \text{order } n^n$

Basic asymptotic efficiency classes

1	constant
$\log n$	logarithmic
n	linear
$n \log n$	n-log-n
n^2	quadratic
n^3	cubic
2^n	exponential
$n!$	factorial

Time efficiency of nonrecursive algorithms

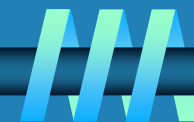


General Plan for Analysis

- ❧ Decide on parameter n indicating input size
- ❧ Identify algorithm's basic operation
- ❧ Determine worst, average, and best cases for input of size n
- ❧ Set up a sum for the number of times the basic operation is executed
- ❧ Simplify the sum using standard formulas and rules (see Appendix A)



Useful summation formulas and rules



$$\sum_{l \leq i \leq u} 1 = 1 + 1 + \dots + 1 = u - l + 1$$

In particular, $\sum_{1 \leq i \leq n} 1 = n - 1 + 1 = n \in \Theta(n)$

$$\sum_{1 \leq i \leq n} i = 1 + 2 + \dots + n = n(n+1)/2 \approx n^2/2 \in \Theta(n^2)$$

$$\sum_{1 \leq i \leq n} i^2 = 1^2 + 2^2 + \dots + n^2 = n(n+1)(2n+1)/6 \approx n^3/3 \in \Theta(n^3)$$

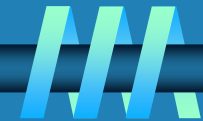
$$\sum_{0 \leq i \leq n} a^i = 1 + a + \dots + a^n = (a^{n+1} - 1)/(a - 1) \text{ for any } a \neq 1$$

In particular, $\sum_{0 \leq i \leq n} 2^i = 2^0 + 2^1 + \dots + 2^n = 2^{n+1} - 1 \in \Theta(2^n)$

$$\sum (a_i \pm b_i) = \sum a_i \pm \sum b_i \quad \sum c a_i = c \sum a_i \quad \sum_{l \leq i \leq u} a_i = \sum_{l \leq i \leq m} a_i + \sum_{m+1 \leq i \leq u} a_i$$



Example 1: Maximum element



ALGORITHM *MaxElement*($A[0..n - 1]$)

//Determines the value of the largest element in a given array

//Input: An array $A[0..n - 1]$ of real numbers

//Output: The value of the largest element in A

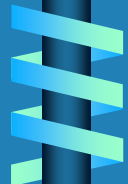
$maxval \leftarrow A[0]$

for $i \leftarrow 1$ **to** $n - 1$ **do**

if $A[i] > maxval$

$maxval \leftarrow A[i]$

return $maxval$



Example 2: Element uniqueness problem

ALGORITHM *UniqueElements*($A[0..n - 1]$)

//Determines whether all the elements in a given array are distinct

//Input: An array $A[0..n - 1]$

//Output: Returns “true” if all the elements in A are distinct

// and “false” otherwise

for $i \leftarrow 0$ **to** $n - 2$ **do**

for $j \leftarrow i + 1$ **to** $n - 1$ **do**

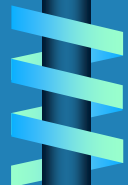
if $A[i] = A[j]$ **return false**

return true

Example 3: Matrix multiplication



```
ALGORITHM MatrixMultiplication( $A[0..n - 1, 0..n - 1]$ ,  $B[0..n - 1, 0..n - 1]$ )  
  //Multiplies two  $n$ -by- $n$  matrices by the definition-based algorithm  
  //Input: Two  $n$ -by- $n$  matrices  $A$  and  $B$   
  //Output: Matrix  $C = AB$   
  for  $i \leftarrow 0$  to  $n - 1$  do  
    for  $j \leftarrow 0$  to  $n - 1$  do  
       $C[i, j] \leftarrow 0.0$   
      for  $k \leftarrow 0$  to  $n - 1$  do  
         $C[i, j] \leftarrow C[i, j] + A[i, k] * B[k, j]$   
  return  $C$ 
```



Example 4: Counting binary digits

ALGORITHM *Binary*(n)

//Input: A positive decimal integer n

//Output: The number of binary digits in n 's binary representation

$count \leftarrow 1$

while $n > 1$ **do**

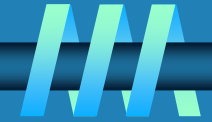
$count \leftarrow count + 1$

$n \leftarrow \lfloor n/2 \rfloor$

return $count$

It cannot be investigated the way the previous examples are.

Summarization



- ⌚ **Algorithm efficiency: time efficiency and space efficiency**
- ⌚ **How to measure the time efficiency**
- ⌚ **worst-case, best-case, and average-case**
- ⌚ **Asymptotic orders of growth: O , Θ and Ω**
- ⌚ **Constant, log, linear, $n \log n$, quadratic, cubic and exponential**
- ⌚ **Analyze the time complexity of non-recurrence relation**

