ALGORITHMS (SET OF INSTRUCTIONS)
EFFICIENCY,ASYMPTOTIC NOTATIONS

Introduction/Course Description

- Algorithim
  - Efficiency Of Algorithm
  - Asymptotic Notations
  - Time Space Trade off

Algorithms

Algorithms

It is a well-defined set of instructions used to solve a particular problem.

Example:

Write an algorithm for finding the location of the largest element of an array Data.

Largest-Item (Data, N, Loc)

1. set k:=1, Loc:=1 and Max:=Data[1]
2. while k<=N repeat steps 3, 4
3. If Max < Data[k] then Set Loc:=k and Max:=Data[k]
4. Set k:=k+1
5. write: Max and Loc
6. exit

Complexity of Algorithms

•The complexity of an algorithm M is the function f(n) which gives the running time and/or storage space requirement of the algorithm in terms of the size n of the input data.

• Two types of complexity

1. Time Complexity : It quantifies the amount of **time** taken by an algorithm to run as a function of the length of the string representing the input

2. Space Complexity: Total space taken by the algorithm with respect to the input size

0-Notation

O-notation

- A function f(n)=O(g(n)) if there are positive constants c and n_0 such that

  f(n)<=c.g(n) for all n>= n_0.

- When f(n)=O(g(n)), it is guaranteed that f(n) grows at a rate no faster than g(n).

  So g(n) is an upper bound on f(n).

Example:

(a) f(n) = 3n+2

Here f(n) <= 5n for n>=1

So, f(n) = O(n).

(b) f(n) = $3n^2$-2

Here f(n) < $3n^2$ for n>=1

So, f(n) = O(n^2).

Some rules related to asymptotic notation

Rule-1

If $f_a(n) = O(g_a(n))$ and $f_b(n) = O((g_b(n))$ then

(a) $f_a(n)+f_b(n) = \max(O(g_a(n)),O(g_b(n))$

(b) $f_a(n) * f_b(n) = O(g_a(n) * g_b(n))$

Rule-2

If f(n) is a polynomial of degree k, then $f(n) = \Theta(n^k)$.

Rule-3

$\text{Log}^k n = O(n)$ for any constant

Typical Growth Rates

Function	Name
c	Constant
logn	Logarithmic
$\log^2 n$	Log-squared
n	Linear
nlogn	
n^2	Quadratic
n^3	Cubic
2^n	Exponential