Module-1, Part-1

BCS0012

Design and Analysis of Algorithms

Dr. Gaurav Kumar
Asst. Prof, CEA, GLA

Topic Covered: Algorithm and Its Characteristics, Asymptotic Notations and Its Properties with Examples, Time Complexity Analysis, Performance Measurements of Algorithms, Comparisons of Different Classes

Syllabus

What, Why and How ?

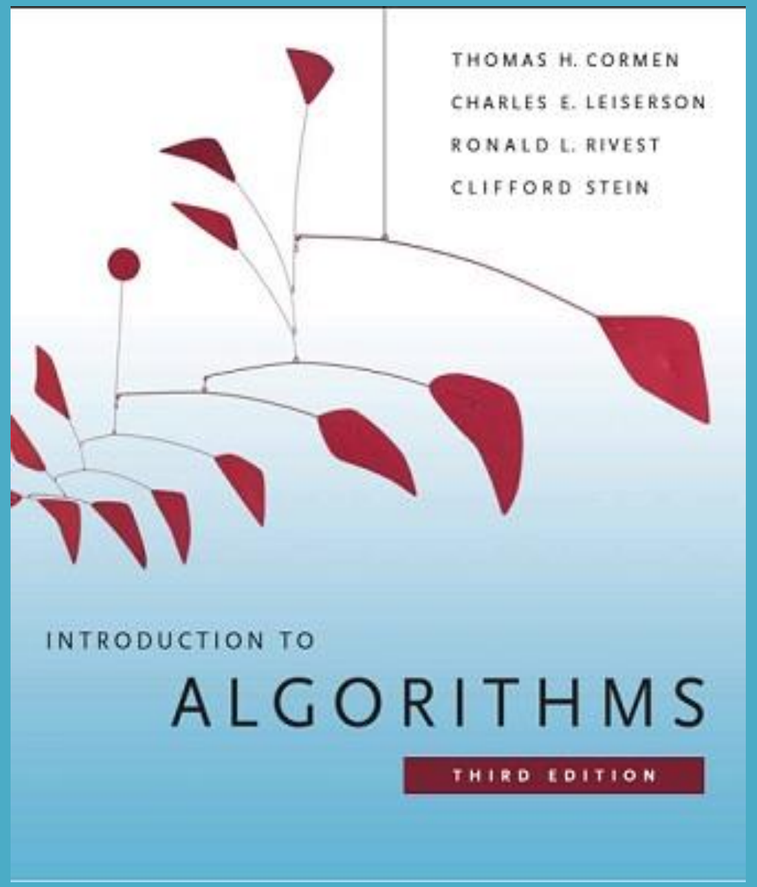
Module 1- Algorithms, Performance Analysis, Recurrence Relations, Sorting Algorithms

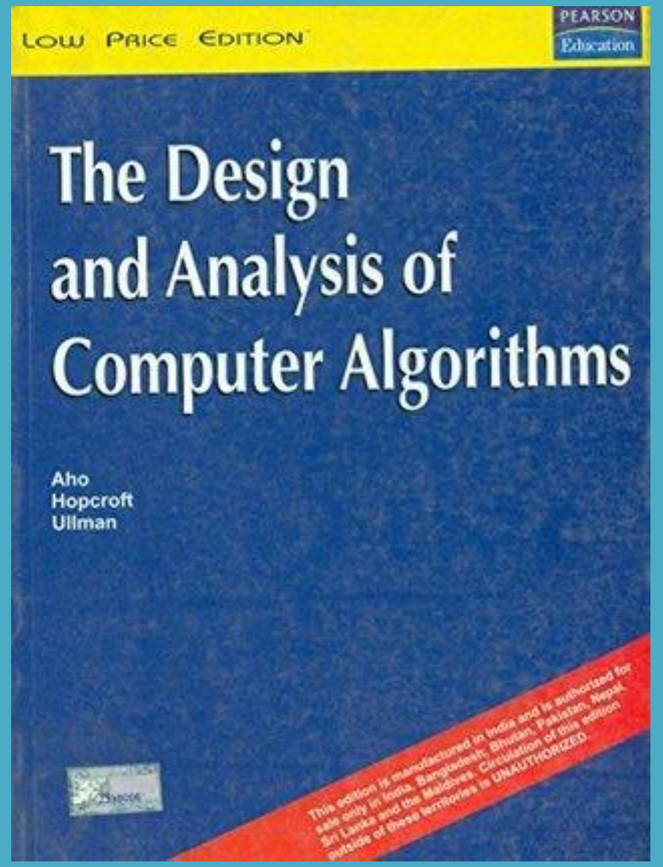
Real Life Applications and Analysis of Different Sorting Algorithms, Recurrence and Tree Data Structure, Searching

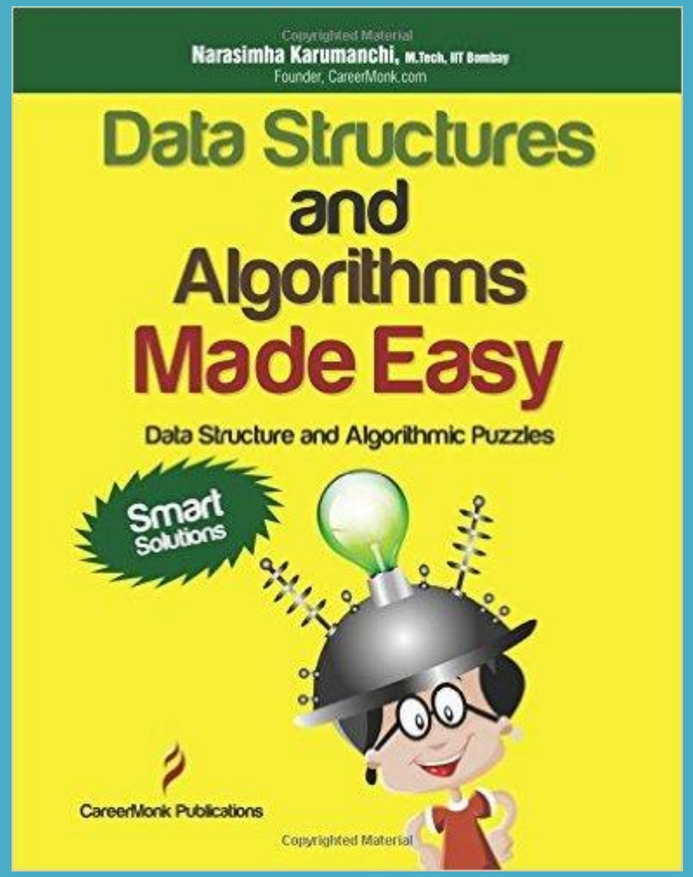
Module 2- Greedy Techniques, Dynamic Programming, Back Tracking , Branch & Bounds, and Advance Data Structures

Understand some of the techniques such as Matrix Multiplications, Spanning Trees and Single Source Shortest Paths, Understand some of the applications of 0-1 Knapsacks, All Pair Shortest Paths, Graphs Coloring, N-Queen Problems , RB Trees

Books for Study





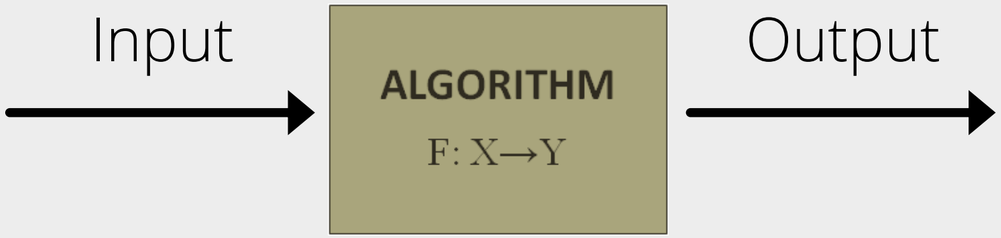


~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Algorithm

For Example

- Making a Delicious Tea



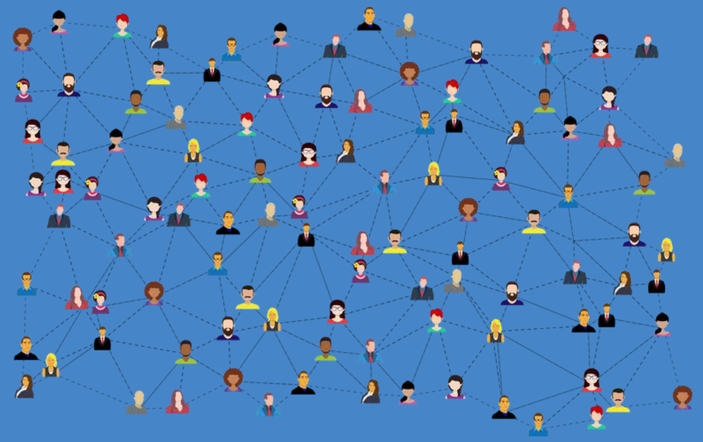
Definition- Step by step procedures to solve a problem

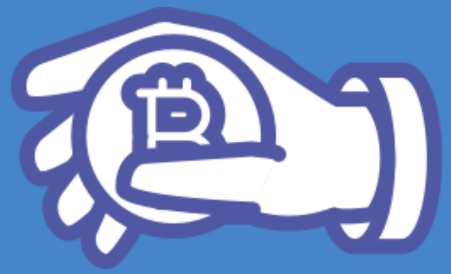
~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

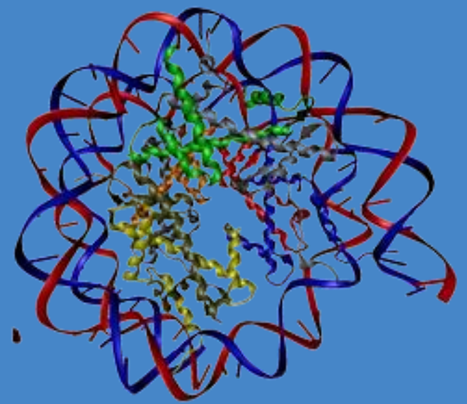
WHY THIS COURSE & ITS BENEFITS ?

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

ALGORITHMS ARE FUNDAMENTAL TO CS

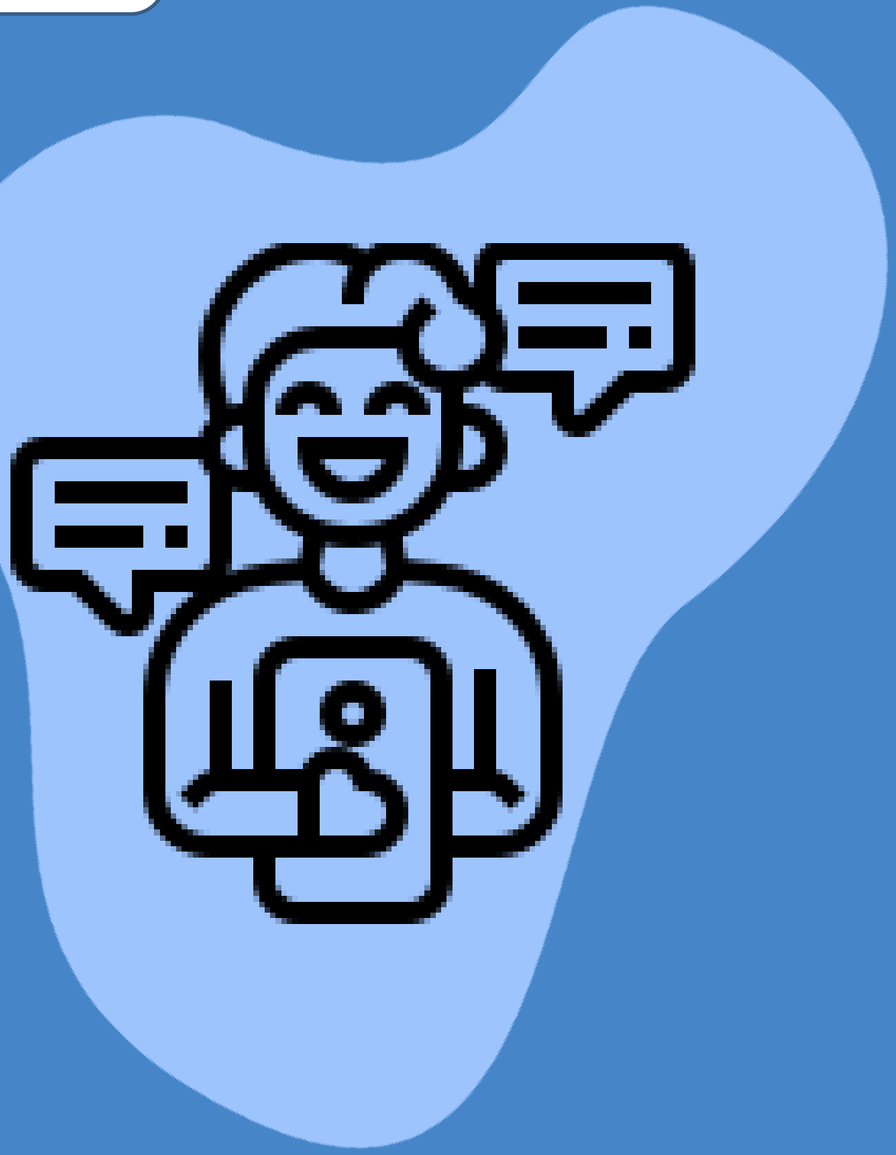






~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Why Algorithms are Useful?

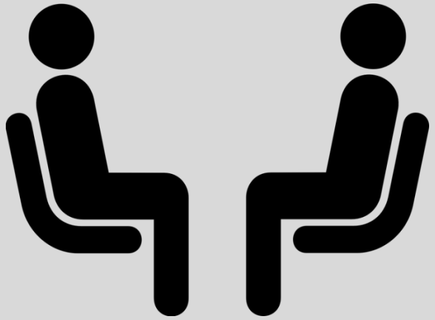


- Once we find an algorithm for solving a problem, we do not need to re-discover it the next time we are faced with that problem.
- All the knowledge required for solving the problem is present in the algorithm.
- For your own use in the future, so that you don’t have spend the time for rethinking it
- Makes it easy when explaining the process to others

ALGORITHMS ARE FUN & USEFUL

Competitive Exam

Interview



~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Algorithm and Its Characteristics

- Step by step procedures to solve a problem
- Each instruction should be cleared and unambiguous
- Zero or more input but at least one output
- Must terminate within a finite amount of time

Difference Between Algorithm and Programme

Algorithm

- An algorithm is a well-defined sequence of steps (written in Natural Language) to solve a given problem
- Design Time
- Domain Knowledge
- Analyze
- Independent from OS and H/W

Programme

- A program is a set of instructions written in Programming Language (computer understandable language) to perform a certain task.
- Program is an implementation of an algorithm
- Knowledge of Programming Language
- Testing
- Dependent on OS and H/W

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Example

5, 8, 2, 32, 12, 9, 11, 3

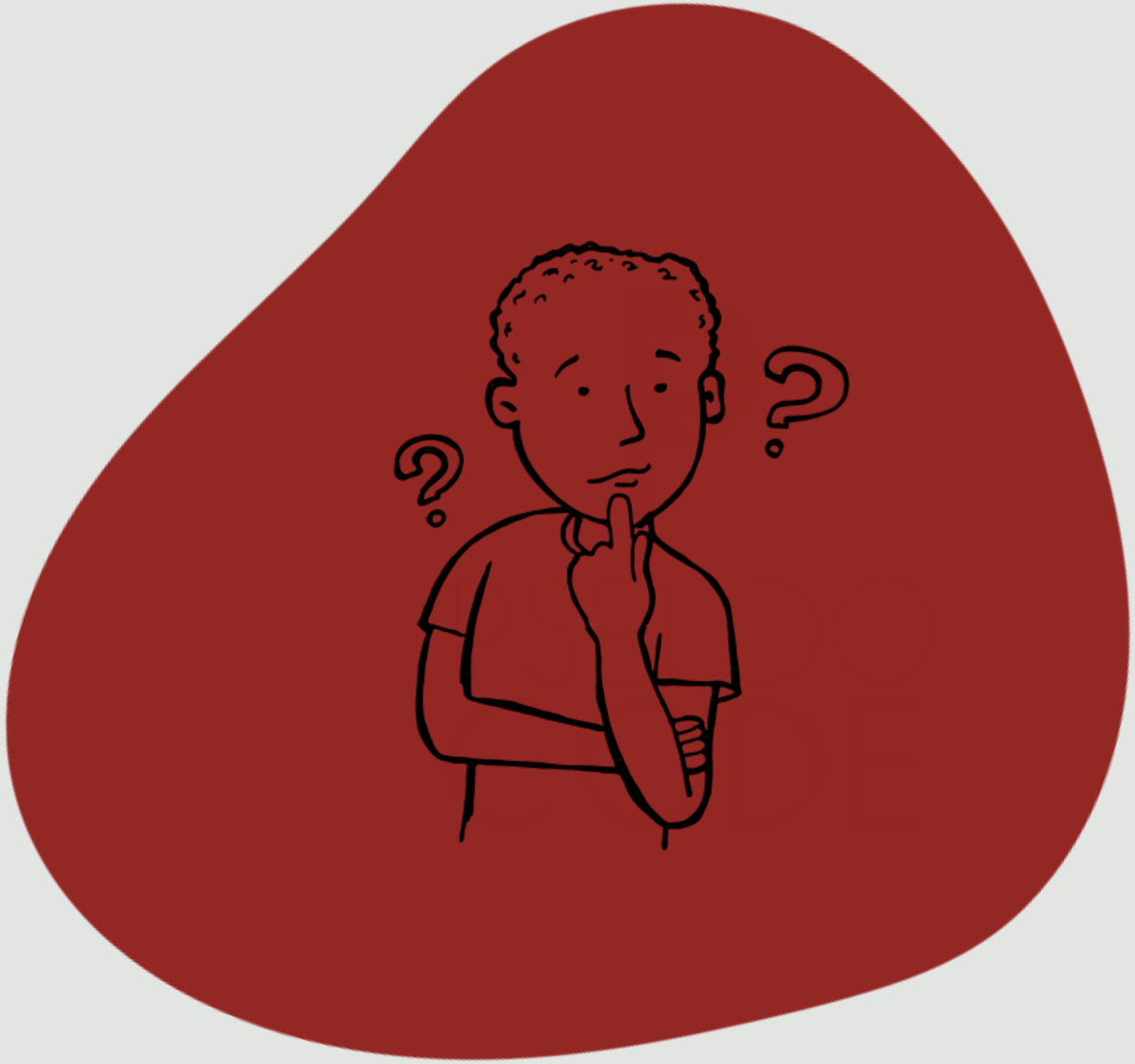
Algorithm of linear search :

- Start from the leftmost element of arr[] and one by one compare x with each element of arr[].
- If x matches with an element, return the index.
- If x doesn’t match with any of elements, return -1.

Program for Linear Search :

```
int search(int arr[], int n, int x)
{
    int i;
    for (i = 0; i < n; i++)
        if (arr[i] == x)
         return i;
    return -1;
}
```

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA



Pseudocode for Linear Search :

```
FUNCTION linearSearch(list, searchTerm):
    FOR index FROM 0 -> length(list):
        IF list[index] == searchTerm THEN
            RETURN index
        ENDIF
        ENDLOOP
            RETURN -1
END FUNCTION
```

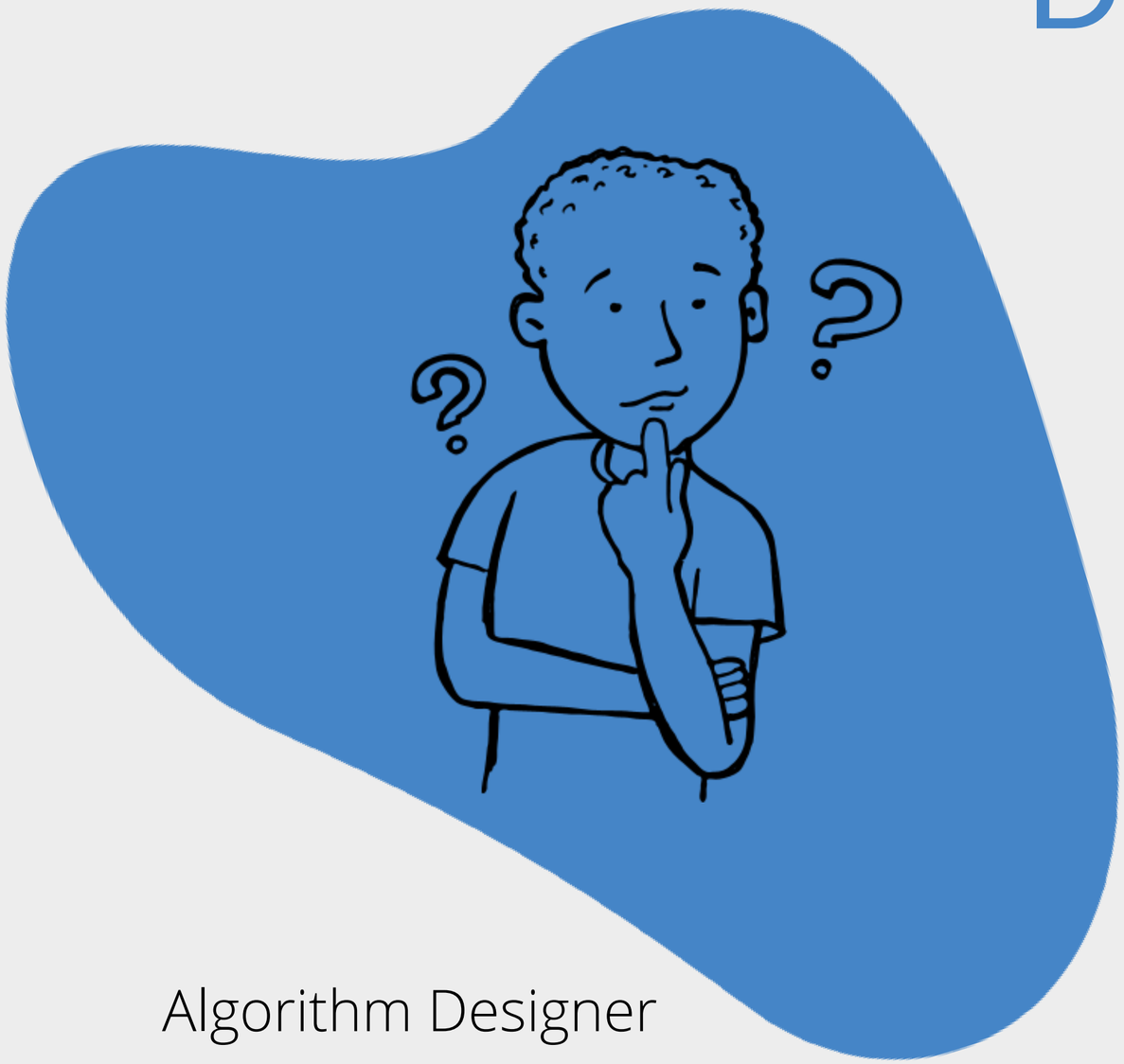
~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Pseudo Code

- It is one of the mathematical notation to represent an algorithm for a program.
- It does not have a specific syntax like any of the programming languages and thus cannot be executed on a computer.
- It is easier to read and understood by programmers who are familiar with different programming languages.
- Pseudocode allows you to include several control structures such as While, If-then-else, Repeat-until, for and case, which is present in many high-level languages.
- Note: Pseudocode is not an actual programming language.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Design and Analysis



Algorithm Designer

Can I do better?

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Analysis of Algorithm

- Given a particular problem of size n. The time required by any algorithm for solving, this problem is denoted by a function such as f(n).

- Time complexity defines the total amount of time an algorithm needs to execute all its key statements and in generating the output. (running time)

- Space: Amount to memory needed by the algorithm for its completion. (Space Complexity)

Good Algorithm

Solve a problem in less amount of time or space complexity or both.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Analysis of Algorithm

Continue...

Generally, we perform the following types of analysis −

For n input size

- **Worst-case –** The maximum amount of time needed by the algorithm to complete a task (worst data or worst input size)
- **Best-case –** The minimum amount of time needed by the algorithm to complete a task (best data or best input data)
- **Average case –** The average amount of time needed by the algorithm to complete a task

Analysis of Algorithm

Continue...

Given two algorithms for a task, how do we find out which one is better?

Task

A1

Algorithm 1

A2

Algorithm 2

Problems

Method-1

Implement both the algorithms and run the two programs on your computer for some inputs and see which one takes less time.

1) It might be possible that for some inputs, first algorithm performs better than the second. And for some inputs second performs better.

Input 1: 3, 2, 7, 1, 8, 9

Input 2: 1, 2,3,7,8, 9

2) It might also be possible that for some inputs, first algorithm perform better on one machine and the second works better on other machine for some other inputs.

Analysis of Algorithm

Continue...

Apriori and Apostiari Analysis

- **Apriori analysis** means analysis is performed prior to running it on a specific system.
  - We determine the time and space complexity of an algorithm by just looking at the algorithm rather than running it on a particular system with a different memory, processor, and compiler. (Theoretical Analysis)

- **Apostiari analysis** of an algorithm means we perform analysis of an algorithm only after running it on a system. It directly depends on the system and changes from system to system. (Empirical Analysis)

Asymptotic Notation

(Algorithm's Growth Rate or Growth of Function)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Asymptotic Notation

(Algorithm's Growth Rate or Growth of Function)

- The main idea of asymptotic analysis is to have a measure of the complexity of algorithms that don’t depend on machine-specific constants
- It doesn’t require algorithms to be implemented and the time taken by programs to be compared.
- It is used for very large value of n (data sets)
- Different types of asymptotic notations are used to represent the complexity of an algorithm.

O − Big Oh (Tightly Upper Bound)

Ω − Big omega (Tightly Lower Bound)

θ − Big theta (both Lower and Upper Bound)

o − Little Oh (Strictly Upper Bound)

ω − Little omega (Strictly Lower Bound)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Example of Asymptotic Analysis

(Best suitable for very large value of n or data sets)

Price: 35 Lakh

Price: 10,000/-

~ 35 Lakh

- The simplest example is a function $f(n) = n^2+3n$, the term 3n becomes insignificant compared to n^2 when n is very large.

- The function "$f(n)$ is said to be asymptotically equivalent to n^2 as $n \to \infty$", and here is written symbolically as $f(n) \sim n^2$.

O – Big Oh Notation

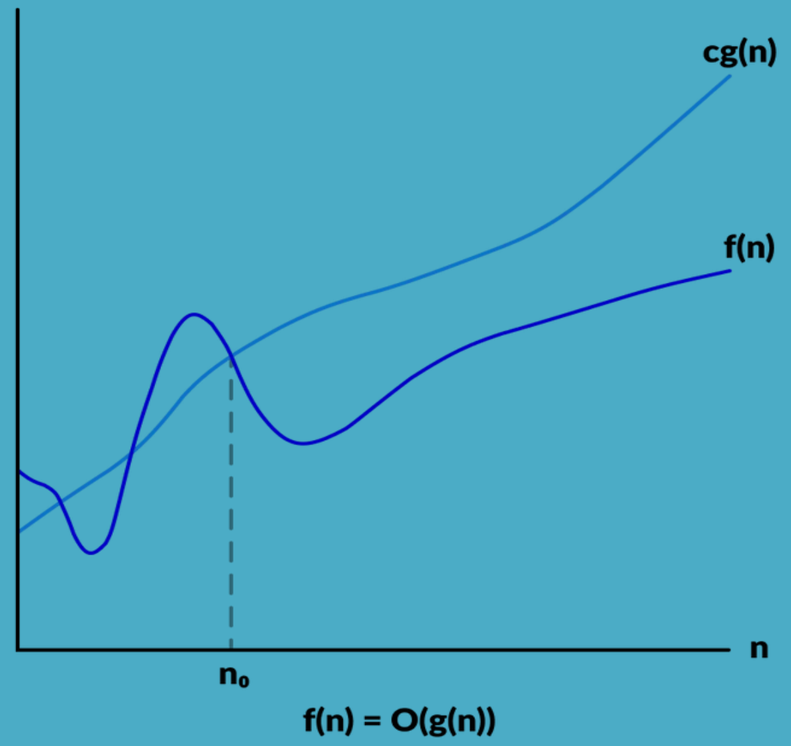
- Tightly Upper Bound of an Algorithm

- Given a particular problem of size n. The time required by any algorithm for solving, this problem is denoted by a function such as f(n).

Lets assume, f(n) and g(n) are two functions

f(n) = O(g(n)) if there exists a positive integer $\mathbf{n_0}$ and a positive constant **c**, such that

$$\mathbf{f(n) \leq c.g(n)\ \forall\ n \geq n_0}$$



Consider the following f(n) and g(n)...

$f(n) = 3n + 2$ and $g(n) = n$

If we want to represent f(n) as O(g(n)) then it must satisfy

$f(n) <= C\ g(n)$ for all values of $C > 0$ and $n >= n_0$,

$f(n) <= C\ g(n) \Rightarrow$ **$3n + 2 <= C\ n$** → **$3n + 2 <= 5n$** for all value of $n > 1$

The above condition is always TRUE for all values of $C = 5$ and $n_0 = 1$.

By using Big - Oh notation

we can represent the time complexity as follows- **$f(n) = 3n + 2 = O(n)$**

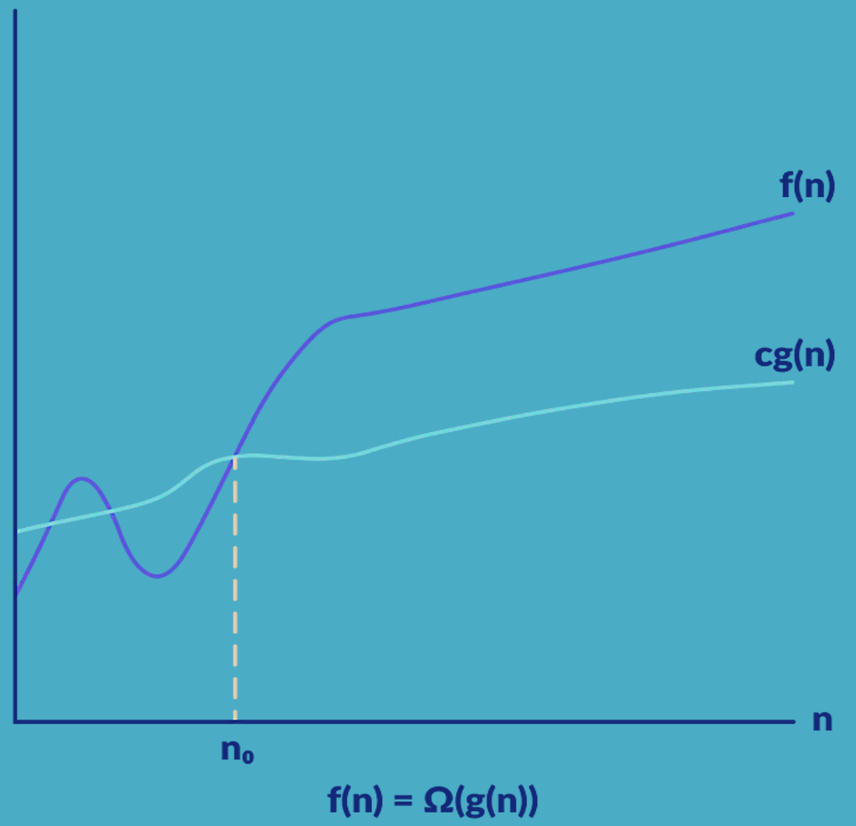
Ω – Big omega Notation

- Tightly Lower bound of the running time of an algorithm.

For a given function f(n) and g(n),

$f(n) \geq c.g(n)$, for $n \geq n_0$; $c > 0$; then

$$f(n) = \Omega\ g(n)$$



Consider the following f(n) and g(n)...

f(n) = 3n + 2 and g(n) = n

If we want to represent f(n) as Ω(g(n)) then it must satisfy f(n) >= C g(n) for

all values of C > 0 and n_0>= 1

⇒ Cn <= 3n + 2 → 4n<= 3n+2 for c=4

The above condition is always TRUE for all values of C = 3 and n >= 1.

By using Big - Omega notation we can represent the time complexity as

follows- f(n)= 3n + 2 = Ω (n)

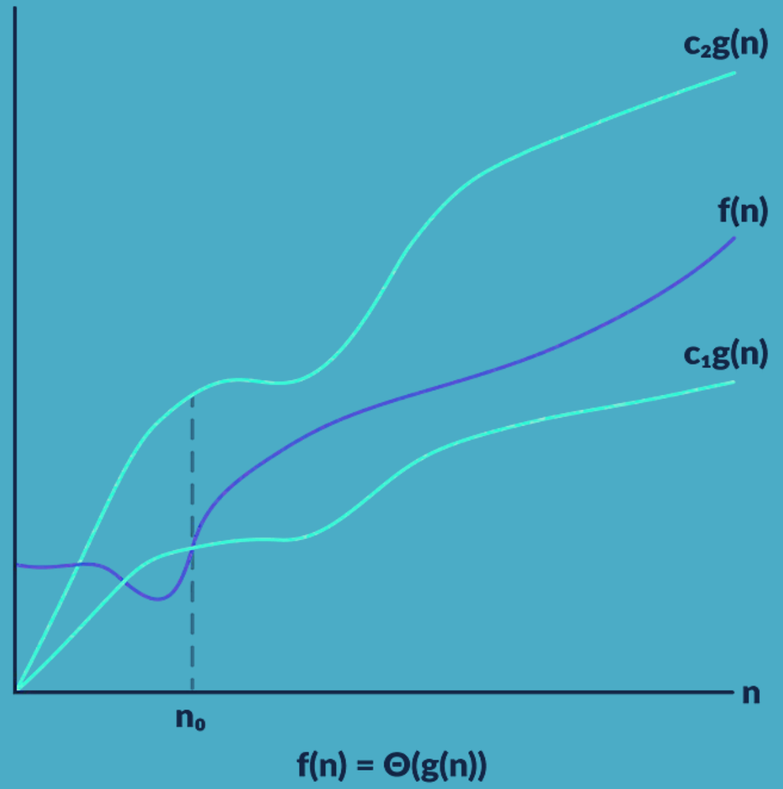
~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

θ – Big theta Notation

- Represents the upper and the lower bound of the running time of an algorithm
- Used for analyzing the average-case complexity of an algorithm.

For a given function f(n) and g(n),

$$c_1 g(n) \leq f(n) \leq c_2 g(n), \text{ for } n \geq n_0 ; c > 0 ; n_0 \geq 1$$



The Theta Notation is more precise than both the big-oh and Omega notation. The function f (n) = θ (g (n)) if g(n) is both an upper and lower bound.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

$f(n) = 3n + 2$ and $g(n) = n$

If we want to represent $f(n)$ as $\Theta(g(n))$ then it must satisfy

$C_1\ g(n) <= f(n) <= C_2\ g(n)$ for all values of $C_1 > 0$, $C_2 > 0$ and $n_0 >= 1$

$C_1\ g(n) <= f(n) <= C_2\ g(n) \Rightarrow \quad C_1\ n <= 3n + 2 <= C_2\ n \rightarrow n <= 3n+2 <= 5n$

Above condition is always TRUE for all values of $C_1 = 1$, $C_2 = 5$ and $n >= 1$.

By using Big - Theta notation we can represent the time complexity as follows...

$$3n + 2 = \Theta(n)$$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

o – Little Oh & ω – Little omega

Little-o Notation (Strictly Upper Bound)

f(n) = o g(n) if, f(n) < c.g(n)

for some constant n_0 and c.

$f(n) = n$, $g(n) = n^2$

f(n) < c.g(n) for c=2, n>=1

f(n)=o(g(n))

Little omega Notation (Strictly Lower Bound)

f(n) = ω g(n) if, f(n) > c.g(n)

for some constant n_0 and c.

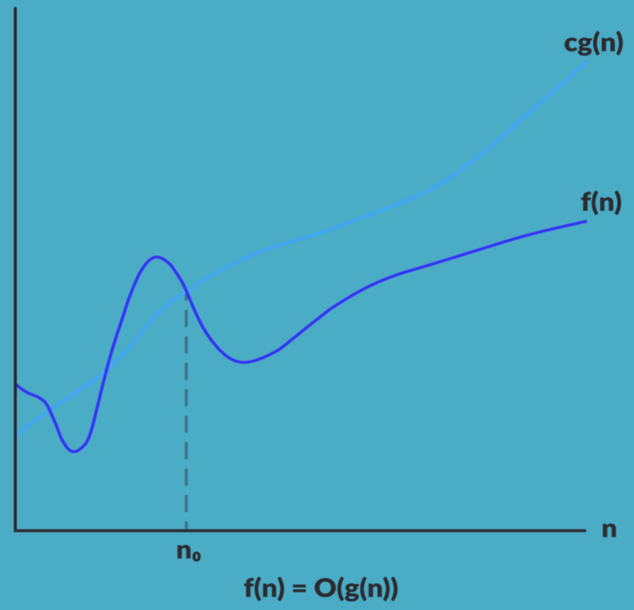
$f(n) = n^2$, $g(n) = n$

f(n) > c.g(n)

f(n)=ω (g(n))

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Example of Asymptotic Notation



O- Big Oh

f(n) = O(g(n)) if there exists a positive integer n_0 and a positive constant c, such that $f(n) \leq c.g(n)\ \forall\ n \geq n_0$

Example-1

As a simple illustrative example, we show that the function

$f(n) = 2n^2 + 5n + 6$

(for larger value of n, we ignore the smaller terms)

For n = 1000, $2n^2$ will be 20,00,000 while 5n + 6 will be 5006 => 20,05,006 ~ 20,00,000

$f(n) = 2n^2 + 5n + 6$ asymptotically equivalent to $\mathbf{O(n^2)}$

For all $n \geq 1$, it is the case that $2n^2 + 5n + 6 \leq 2n^2 + 5n^2 + 6n^2 = \mathbf{13n^2}$

=> f(n)= $2n^2 + 5n + 6 \leq \mathbf{13n^2}$ → $\mathbf{O(n^2)}$

Hence, we can take **c = 13 and n_0 =1**, and the definition is satisfied.

Example of Asymptotic Notation

O- Big Oh

f(n) = O(g(n)) if there exists a positive integer $\mathbf{n_0}$ and a positive constant **c**, such that $\mathbf{f(n) \leq c.g(n)\ \forall\ n \geq n_0}$

Example-2

Prove that $\mathbf{2n^2 = O(n^3)}$

Proof

Assume that $f(n) = 2n^2$, and $g(n) = n^3$, $f(n) = O(g(n))$?

Now we have to find the existence of c and n_0

$f(n) \leq c.g(n) \rightarrow 2n^2 \leq c.n^3 \rightarrow 2 \leq c.n$

if we take, $c = 1$ and $n_0 = 2$ OR $c = 2$ and $n_0 = 1$ then

$2n^2 \leq c.n^3$, Hence $f(n) = O(g(n))$, $c = 1$ and $n_0 = 2$

Example of Asymptotic Notation

O- Big Oh

f(n) = O(g(n)) if there exists a positive integer $\mathbf{n_0}$ and a

positive constant **c**, such that $\mathbf{f(n) \leq c.g(n)\ \forall\ n \geq n_0}$

Example-3

Prove that $\mathbf{1000.n^2 + 1000.n = O(n^2)}$

Proof

Assume that $f(n) = 1000.n^2 + 1000.n$, and $g(n) = n^2$

We have to find existence of c and n_0 such that $0 \leq f(n) \leq c.g(n)\ \forall\ n \geq n_0$

$1000.n^2 + 1000.n \leq c.n^2$

$1000.n^2 + 1000n^2 = 2000.n^2 = c.n^2$

for $c = 2000$, $1000.n^2 + 1000.n \leq 2000.n^2$

this is true for all value of $n \geq 1$, Hence $f(n) = O(g(n))$ for $c = 2000$ and $n_0 = 1$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Example of Asymptotic Notation

O- Big Oh

f(n) = O(g(n)) if there exists a positive integer $\mathbf{n_0}$ and a positive constant **c**, such that $\mathbf{f(n) \leq c.g(n)\ \forall\ n \geq n_0}$

Example-4

Prove that $\mathbf{n^3 = O(n^2)}$

Proof

On contrary we assume that there exist some positive constants c and n_0 such that

$$0 \leq n^3 \leq c.n^2 \qquad \forall\ n \geq n_0$$

$\Rightarrow n^3 \leq c.n^2 \rightarrow n \leq c$

Since c is any fixed number and n is any arbitrary constant, therefore $n \leq c$ is not possible in general.

Hence our supposition is wrong and $n^3 \leq c.n^2$, for $n \geq n_0$ is not true for any combination of c and n_0. Hence, $n^3 = O(n^2)$ does not hold

Example of Asymptotic Analysis

O- Big Oh

f(n) = O(g(n)) if there exists a positive integer $\mathbf{n_0}$ and a positive constant **c**, such that $\mathbf{f(n) \leq c.g(n)\ \forall\ n \geq n_0}$

Example-5

Find an Upper bound for $f(n) = n^4 + 1000n^3 + 5000n$

$n^4 + 1000n^3 + 5000n <= 2*n^4$ for all n>= 10

$F(n) = n^4 + 1000n^3 + 5000n = O(n^4)$ with $c = 2$ and $n_o = 10$

$F(n) = n^4 + 1000n^3 + 5000n >= n^4$ **for all n>= 1 → $F(n) = \Omega(n^4)$ with $C = 1$ and $n_o = 1$**

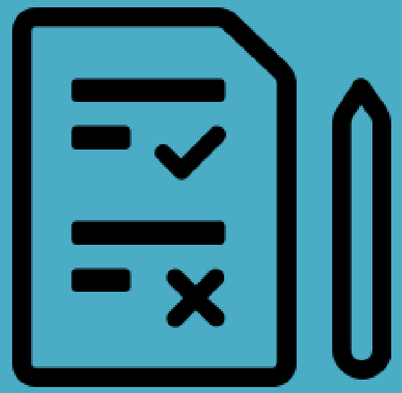
~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Assessment

Question -1

F(n) = 1000 n^2 + 5n^3 + 600000n, What is the Order of Function ?

Answer : f(n)= O(n^3)



Question -2

F(n) = n^5 + 500n^3 + 5000n^2 + 99999999*9999 What is the Order of Function ?

Answer : f(n)= O(n^5)

Question -3

F(n) = n + 500*n + 5000*n + 50,00,000*n What is the Order of Function ?

Answer : f(n)= O(n)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Properties of Asymptotic Notation

1. **Reflexivity**
2. **Symmetry**
3. **Transitivity**
4. **Transpose Symmetry**
5. **General Properties**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Properties of Asymptotic Notation

1. Reflexive Properties

If f(n) is given then **f(n) = O(f(n))**

Example: If $f(n) = n^3 \Rightarrow O(n^3)$

Similarly,

$$f(n) = \Omega(f(n)) \text{ or } f(n) = \Theta(f(n))$$

Properties of Asymptotic Notation

2. Symmetry Properties

(Valid only for Theta Notation)

If $f(n) = \Theta(g(n))$, then $g(n) = \Theta(f(n))$

Example

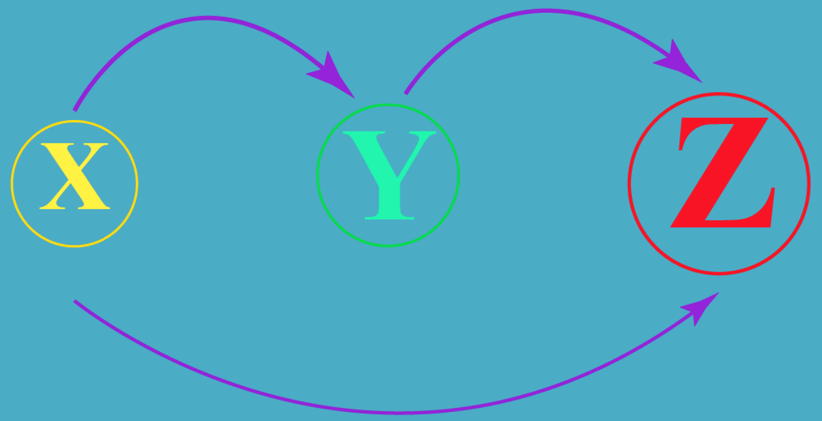
If $f(n) = n^2$ and $g(n) = n^2$

then $f(n) = \Theta(n^2)$ and $g(n) = \Theta(n^2)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Properties of Asymptotic Notation

3. Transitive Properties (for all notations)



f(n) = O(g(n)) and g(n) = O(h(n)) then

⇒ f(n) = O(h(n))

Example

If $f(n) = n$, $g(n) = n^2$ and $h(n) = n^3$

$$n<n^2<n^3$$

⇒ n is $O(n^2)$ and n^2 is $O(n^3)$ then n is $O(n^3)$

Properties of Asymptotic Notation

4. Transpose Symmetry (for Big Oh and Omega notation only)

If f(n) = O(g(n)) then g(n) = Ω(f(n))

Example

If $f(n) = n$ and $g(n) = n^2$

then n is $O(n^2)$ and n^2 is $\Omega(n)$

Properties of Asymptotic Notation

5. General Properties

A. If $f(n) = O(g(n))$ or $\Omega(g(n))$ or $\Theta(g(n))$

Then $a^* f(n) = O(g(n))$ or $\Omega(g(n))$ or $\Theta(g(n))$

***Example*: $f(n) = 2n^2 + 5 = O(n^2)$**

if $a = 5$, → $a^* f(n)$ → $10n^2 + 25 = O(n^2)$

*** Valid for all notations**

Properties of Asymptotic Notation

5. General Properties

B. If f(n) = O(g(n)) and f(n)= Ω (g(n)

Then f(n) = Θ (g(n))

***Example*: f(n) = n^2 and g(n) = n^2**

g(n)<=f(n)<=g(n)

Properties of Asymptotic Notation

5. General Properties

C. If $f(n) = O(g(n))$ and $d(n) = O(e(n))$

Then $f(n) + d(n) = O(\max(g(n), e(n)))$

***Example*: $f(n) = n = O(n)$ and $d(n) = n^2 = O(n^2)$**

$f(n) + d(n) = n + n^2 = O(n^2)$

Properties of Asymptotic Notation

5. General Properties

D. If $f(n) = O(g(n))$ and $d(n) = O(e(n))$

Then $f(n) * d(n) = O((g(n) * e(n)))$

***Example*: $f(n) = n = O(n)$ and $d(n) = n^2 = O(n^2)$**

$f(n) * d(n) = n * n^2 = n^3 = O(n^3)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Practice Examples

Example 1- $F(n) = n^3$ $D(n) = n^4$ $E(n) = n^6$

$T(n) = F(n) + D(n) + E(n)$

$O(T(n)) = ?$ $O(T(n)) = n^6$

Example 2- $F(n) = 3n^2 * n^3 + n * n^2 + 20n^2 * n^2 + 2n^2$

$O(f(n)) = ?$ $O(f(n)) = n^5$

Example 3- $F(n) = 3n^3 + n^{3.5} + 10n^4 + 2n^2$

$O(f(n)) = ?$

$O(f(n)) = n^4$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Practice Examples

Example 4- $F(n) = n^2 + n^4$ $D(n) = n^2$

$T(n) = F(n) * D(n)$

$O(T(n)) = ?$ $O(T(n)) = n^6$

Example 5- $F(n) = 3n^4 + 2n^2$

If $T(n) = 10{,}000 * F(n)$, then $O(T(n)) = ?$ $O(T(n)) = n^4$

GLA UNIVERSITY
MATHURA
Recognised by UGC Under Section 2(f)
Accredited with A Grade by NAAC

Example of Worst-Average-Best Case Analysis of Algorithms

Worst-Average-Best Case

Analysis of Algorithms

- 01 Linear Search

- 02 Binary Search Tree

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

1. Linear Search

8	3	1	2	4	5	6	7

Algorithm of linear search-

Step 1- Start from the leftmost element of arr[] and one by one compare x with each element of arr[].

Step 2- If x matches with an element, return the index.

Step 3- If x doesn't match with any of the elements, return -1.

Key Element = 4 Total Number of Comparision = 5

Key Element = 9 Total Number of Comparision = 8

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Linear Search

8	3	1	2	4	5	6	7

Best Case

Searching the key element present at the first index of a list

Best Case Time Complexity

Key Element = 8 Total Number of Comparision = 1

b(n) = 1 = Constant b(n) = O(1)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Linear Search

8	3	1	2	4	5	6	7

Worst Case

Searching the key element present at last index of a list

Worst Case Time Complexity

Key Element = 7 Total Number of Comparision = 8

for n elements w(n) = n w(n) = O(n)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Linear Search

8	3	1	2	4	5	6	7

Average Case

All possible case time divided by number of cases

Average Case Time Complexity

Very difficult to analyze

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

$A(n) = O(n)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Apply Asymptotic Notations in the Case Analysis

For Liner Search

Worst Case Time w(n) = n (Representing Linear Class)

Can we write

w(n)= O(n)

w(n)= Omega(n)

w(n)= Theta(n)

Yes

For Best Case, Can we write

b(n)= O(1)

b(n)= Omega(1)

b(n)= Theta(1)

There is no standard fixed asymptotic notations to represent the worst-case or best case or average case analysis of algorithms

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

To ease of understanding

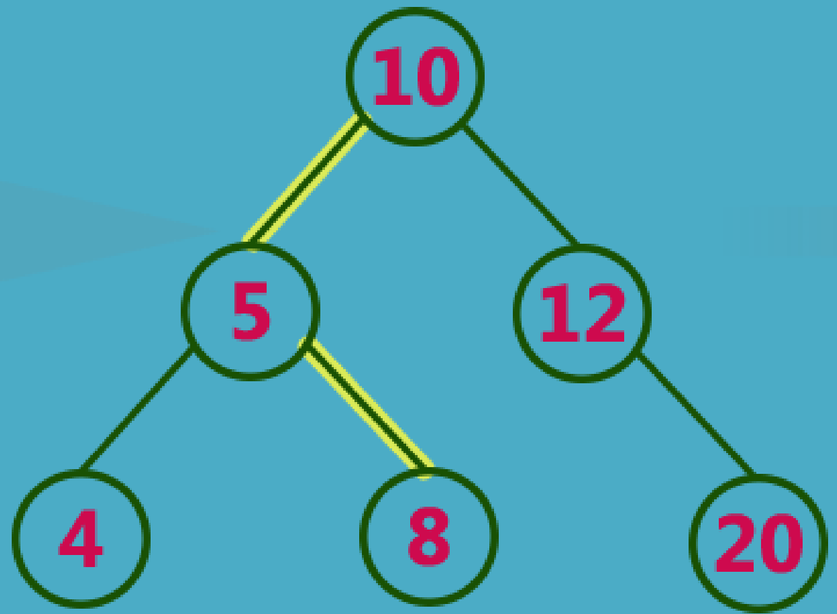
- Worst Case Timing can be represented as Upper Bound but vice versa is not true

In this subject

(depend on the data structure used)

- we will use Big O to represent the Worst-Case Timing
- Big Omega for the Best Case
- Theta for the Average case

2. Binary Search Tree



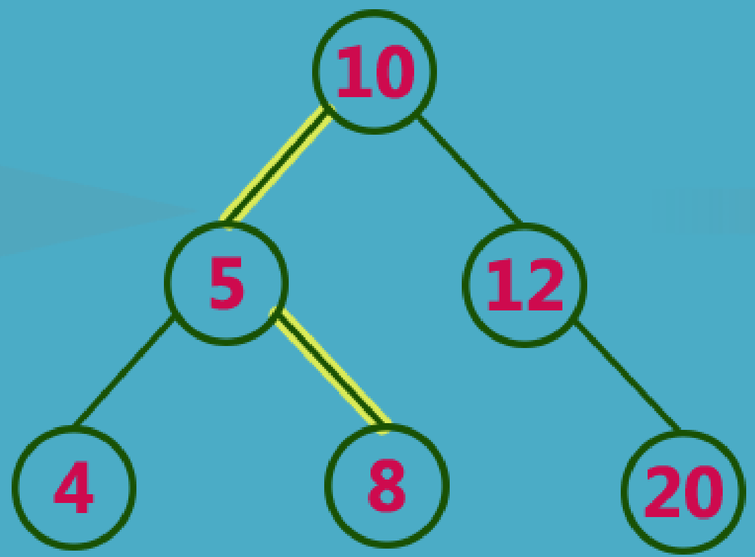
h

Searching a 8 number element in a tree, requires **3** comparisions

For n elements, h= logn

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

2. Binary Search Tree



Best Case

Searching the key element present at the root of the tree

Best Case Time Complexity

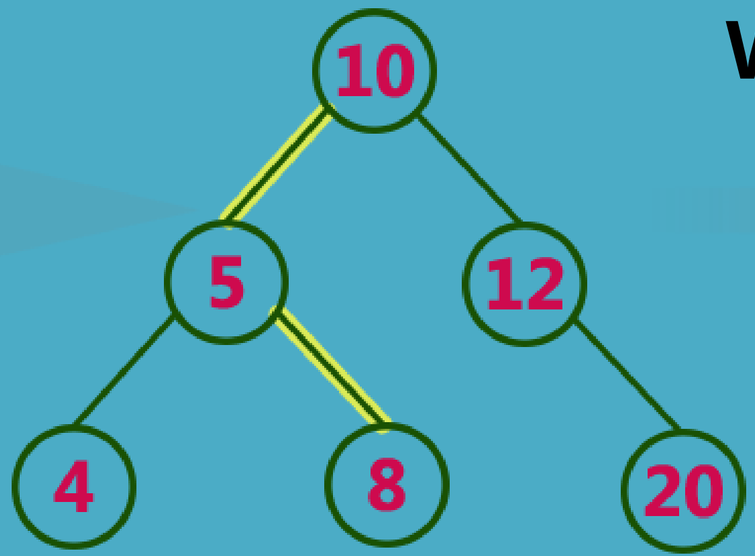
Key Element = 10

Total Number of Comparision = 1

b(n) = 1 = Constant b(n) = O(1)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

2. Binary Search Tree



Worst Case

Searching the key element present at leaf node of a tree

Worst Case Time Complexity

Key Element = 8

Total Number of Comparision = 3

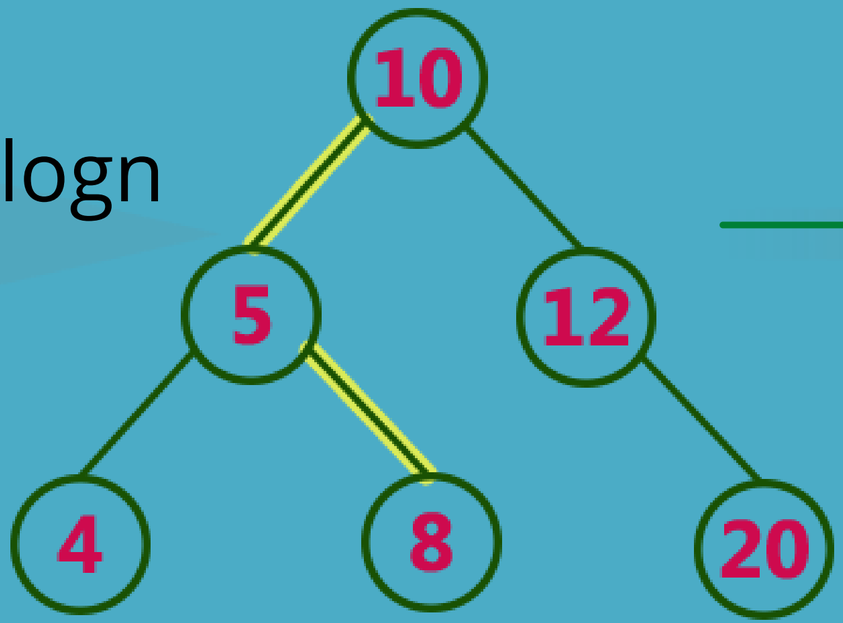
i.e. log 8 =3

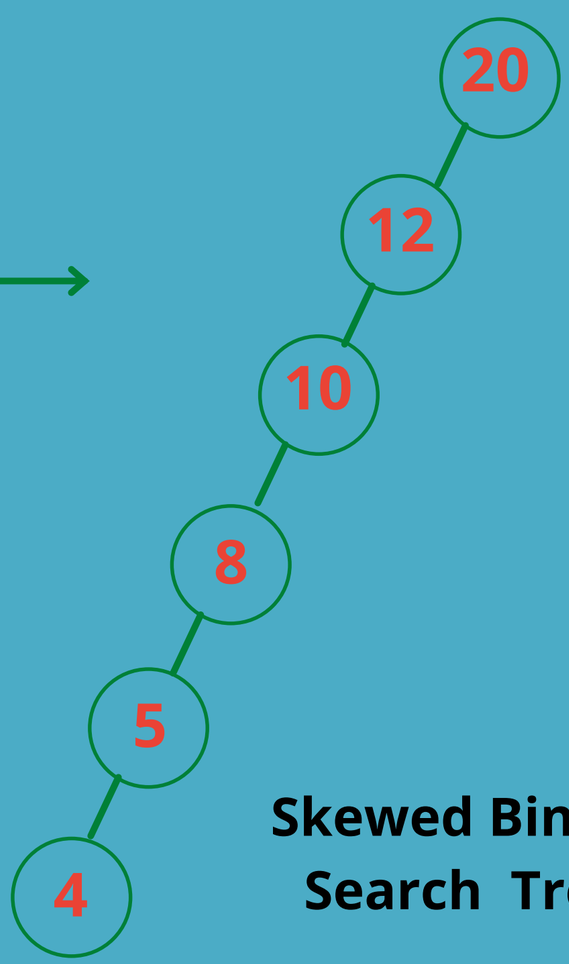
for n elements w(n) = logn w(n) = O(logn)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

2. Binary Search Tree

logn





n

Skewed Binary Search Tree

Worst Case Time Complexity

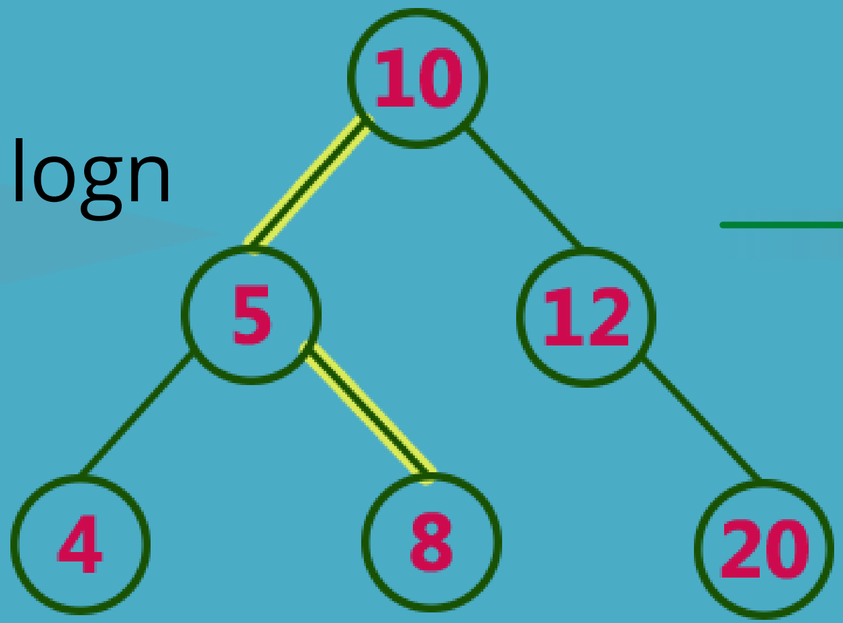
Key Element = 4 Total Number of Comparision = 6, Time = 6 units

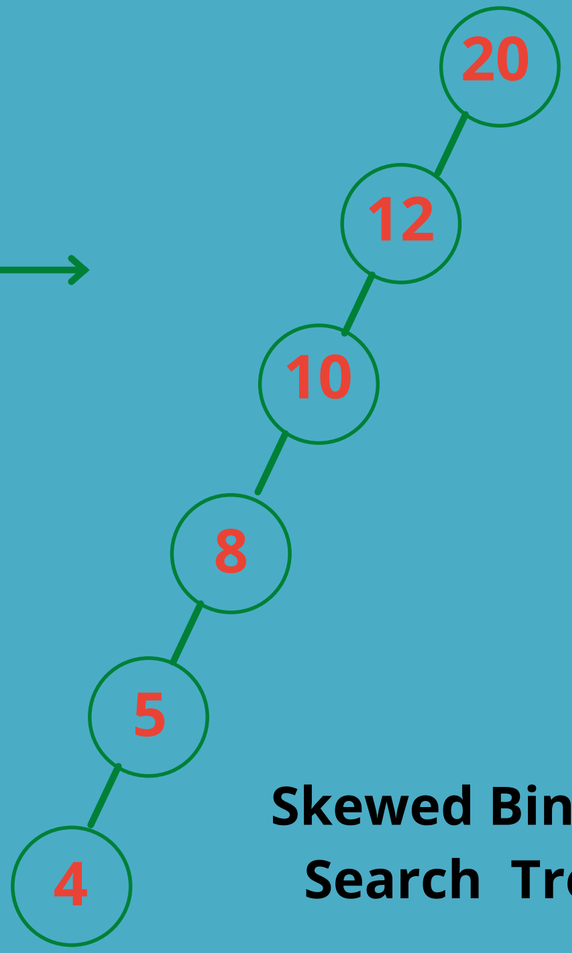
for n elements **w(n) = n** **w(n) = O(n)**

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

2. Binary Search Tree

logn





n

Skewed Binary Search Tree

Worst Case Time Complexity

for n elements BST,

minimum worst case timing w(n) = O(logn) for balanced binary tree

maximum worst case timing w(n) = O(n) for skewed binary tree

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Analysis and Performance Measurements of Algorithms

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Write an Algorithm to Swap two numbers

a=5, b=10

```
temp=a;
a=b;
b=temp;
```

```
Algorithm Swap (a, b)
{
temp=a;
a=b;
b=temp;
}
```

```
Algorithm Swap (a, b)
begin
temp:=a;  or temp<-a;
a:=b; or a<-b;
b:=temp; or b<- temp;
end
```

Note: Any one method discussed above or even different methods can be used to write an algorithm.

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

How to analyze the algorithm

```
Algorithm Swap (a, b)
{
temp=a;   ⟶   1 unit of time
a=b;      ⟶   1 unit of time
b=temp;   ⟶   1 unit of time
}
```

Total= 3 unit of time

*Assume that every statement take one unit of time for execution

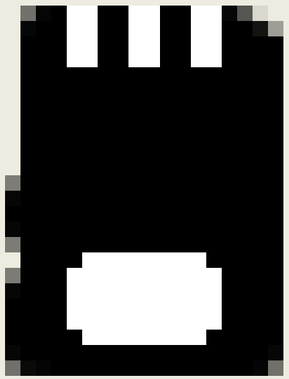
Time Complexity= f(n)= constant = O(1)

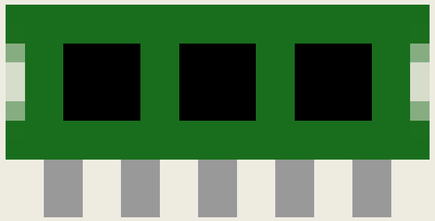
For Example

if a statement is defined as : X= 3*a + 5*b +6 , If we convert into Machine Code, it will have 2 multiplications, 2 addition and one assignment but we will ignore the machine analysis and simple assume that each statement takes 1 unit of time

How to analyze the algorithm

Space Complexity





```
Algorithm Swap (a, b)
{
temp=a;
a=b;
b=temp;
}
```

Total Variable

a ⟶ 1 word

b ⟶ 1 word

temp ⟶ 1 word

Total= 3 words

Space Complexity= s(n)= constant = O(1)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Frequency Count Methods (Loop)

(to analyze the algorithm)

Sum of n number

```
Algorithm Sum (A, n)
{
 S=0;                                  ------>  1
for ( i=0; i<n; i++)                   ------>  n+1
   {                                   i=0  -> 1
                                       i++ -> n
    S= S + A[i];                       i<n -> n+1   ------>  n
    }
return S;                              ------>  1
}
```

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

n=5

A =

6	4	3	8	1
0	1	2	3	4

i=0 ✓
i=1 ✓
i=2 ✓
i=3 ✓
i=4 ✓
i=5 ✗ i<n

condition checked 6 times

i++, i has changed 5 times

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Frequency Count Method (Nested Loop)

(to analyze the algorithm)

Sum of two Matrices of size n*n

```
Algorithm Sum (A, B, n)
{
for ( i=0; i<n; i++)                          ⟶  n+1
  {
   for ( j=0; j<n; j++)                       ⟶  n* (n+1)
    {
     C[i, j ]= A [i, j] + B[i, j];            ⟶  n * n
    }
  }
}
```

f(n) = n+1 + n* (n+1) + n* n = $2n^2 + 2n + 1 = O(n^2)$

Space Complexity

A ----->n * n

B------>n*n

C------>n*n

i------>1

j------>1

n------>1

S(n) = $3n^2+3 = O(n^2)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Frequency Count Method (Nested Loop)

(to analyze the algorithm)

Multiplication of two Matrices of size n*n

```
Algorithm Sum (A, B, n)
{
for ( i=0; i<n; i++)                          ------------>   n+1
  {
    for ( j=0; j<n; j++)                      ------------>   n* (n+1)
     {
        C[i,j]=0;                             ------------>   n* n
        for ( k=0; k<n; k++)                  ------------>   n* n* (n+1)
          {
           C[i, j ]= C[i,j] + A [i, k] * B[k, j];   ---->   n* n* n
          }
     }
  }
}
```

Space Complexity

A ----->n * n

B------>n*n

C------>n*n

i------>1

j------>1

k------>1

n------>1

$S(n) = 3n^2+4 = O(n^2)$

$f(n) = n+1 + n* (n+1) + n* n + n* n*(n+1) + n*n*n$

$= 2n^3 + 3n^2 + 2n+ 1= O(n^3)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Some Examples of For Loop

```
for ( i=0; i<n; i++)
  {
    statement;
  }
```

→ f(n) = O(n)

```
for ( i=n; i>0; i--)
  {
    statement;
  }
```

→ f(n) = O(n)

```
 for ( i=1; i<n; i=i+2)
   {
     statement;
   }
```

→ f(n) = n/2= O(n)

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Some Examples of For Loop

```
for ( i=1; i<n; i=i*2)
   {
     statement;
    }
```

i
1
$1*2=2$
$2*2=2^2$
$2*2*2=2^3$
$2*2*2*2=2^4$
.
.
$2*2*2*2...k = 2^K$

Assume that $2^K>=n$ (stopping condition)

take log both side → $\log_2 (2^K) = \log (n)$

$k = \log_2 n$ → $O(\log_2 n)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Some Examples of For Loop

```
for ( i=n; i>1; i=i/2)
   {
     statement;
    }
```

i

n

$n/2=n/2$

$n/2*2=n/2^2$

$n/2*2*2=n/2^3$

$n/2*2*2*2=n/2^4$.

.

.

$n/2*2*2*2...k = n/2^K$

Assume that $n/2^K=1$ (stopping condition)

$n=2^k$ → take $\log_2$ both side

$\log_2(n) = \log_2(2^k)$ → $k = \log_2 n$ → $O(\log_2 n)$

Some Examples of For Loop

```
p=0;
for ( i=1; p<n; i++)
  {
    p=p+i;
  }
```

i	p
1	0+1
2	1+2
3	1+2+3
4	1+2+3+4
5	1+2+3+4+5
.	.
.	.
k	1+2+3+4+5.....+k

Assume that p=n (stopping condition)

--> p=1+2+3+4+5+...+k

--> k(k+1)/2 = n

$(k^2+k)/2 = n$ ---> $k^2 + k = 2* n$ --> $k^2=n$

k=sqrt (n), k = O(sqrt(n))

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Examples of independent loops

```
for ( i=0; i<n; i++)
{
  statement;
 }
```

→ **n**

```
for ( i=0; i<n; i++)
{
 statement;
 }
```

→ **n**

```
for ( i=0; i<n; i++)
{
 statement;
 }
```

→ **n**

$f(n) = n+n+n$

$=3n$

$f(n)= O(n)$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Commonly used Logarithm and Summations

Logarithms

1. $\log x^y = y\log x$
2. $\log xy = \log x + \log y$
3. $\log^k n = (\log n)^k$
4. $\log \log n = \log (\log n)$
5. $\log(x/y) = \log x - \log y$
6. $a^{\log_b x} = x^{\log_b a}$
7. $\log_b x = \log_a x / \log_a b$

Arithmetic Series

$f(n) = 1+2+3+4+5\ldots+n = n(n+1)/2$

Geometric Series

$f(n) = 1+2+2^2+2^3+\ldots+2^n = 2^{n+1} - 1/n-1$

Harmonic Series

$f(n) = 1 + 1/2 + 1/3 +\ldots..+1/n = \log n$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Type and Comparison of Order of Function of Classes

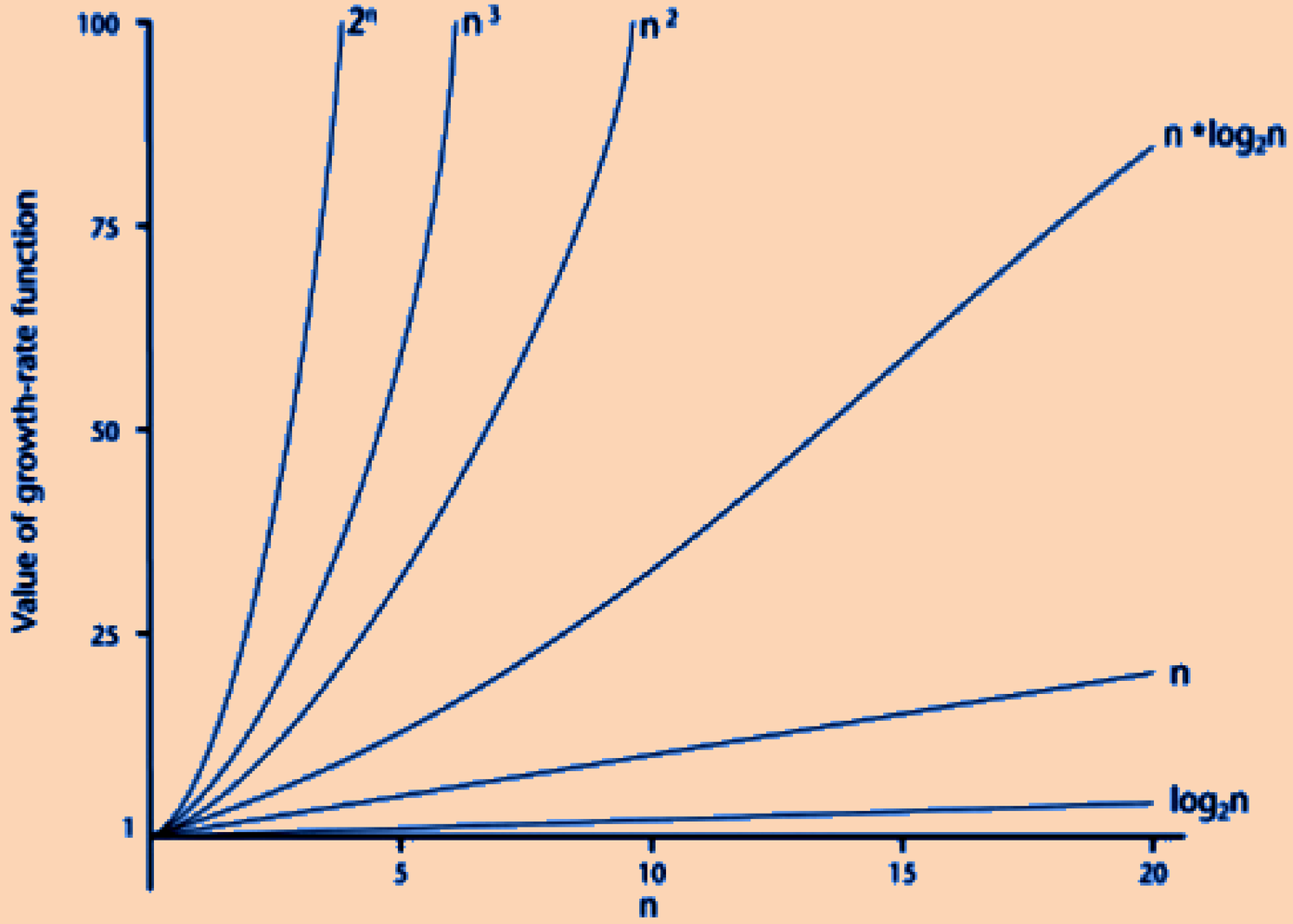
(Valid for larger values of n)

$O(1) < O(\log n) < O(\sqrt{n}) < O(n) < O(n\log n) < O(n^2) < O(n^3) < O(2^n) < O(10^n) < O(n^n)$

	$\log_2 n$	n	n^2	n^3	2^n	n^n
n=1	0	1	1	1	2	1
n=2	1	2	4	8	4	4
n=4	2	4	16	64	16	256
n=8	3	8	64	512	256	1,67,77,216
n=10	3.2	10	100	1000	1024	10^{10}

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Comparison of Order of Growths of Functions



Comparison of Functions

$f(n)= n^2 \log n$ $g(n)= n(\log n)^{10}$

How will you solve the comparison?

1. Substitution Method

	n^2	n^3
n=2	4	8
n=3	9	27
n=4	16	64
n=5	25	125

Comparison of Functions

2. Log Method (Apply Log at both side)

n^2 n^3

Log n^2 **log n^3** Using this formula: $\log x^y = y\log x$

2Log n $<$ **3log n**

Some Logarithms Formula

1. $\log x^y = y\log x$
2. $\log xy = \log x + \log y$
3. $\log^k n = (\log n)^k$
4. $\log \log n = \log (\log n)$
5. $\log(x/y) = \log x - \log y$
6. $a^{\log_b x} = x^{\log_b a}$
5. $\log_b x = \log_a x / \log_a b$
6. if $a^b = n$ then $b = \log_a n$

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Examples of Comparison of Functions

$f(n)= n^2\log n$ $g(n)= n(\log n)^{10}$

Apply Log at both side

$\log (n^2\log n)$ $\log(n(\log n)^{10})$

Using this formula $\log xy = \log x + \log y$

$\log (n^2) + \log(\log n)$ $\log(n)+ \log(\log n)^{10}$

Using this formula: $\log x^y = y\log x$

$2\log n + \log\log n > \log n + 10\log(\log n)$

$f(n)> g(n)$

$n= 2^{1000}$ $\log n = \log 2^{1000} =1000$

$\log(\log n)= \log\log 2^{1000} = \log 1000 \sim= \log 2^{10} =10$

$\log n > \log\log n$ for all value of n

Examples of Comparison of Functions

$f(n) = 3n^{\sqrt{n}}$ $\quad$ $g(n) = 2^{\sqrt{n} * (\log n)}$

Using this $\log x^y = y \log x$

$3n^{\sqrt{n}}$ $\quad$ $2^{\log n^{\sqrt{n}}}$

$3n^{\sqrt{n}}$ $\quad$ $(n^{\sqrt{n}})^{\log 2}$

Using this formula

$a^{\log_b x} = x^{\log_b a}$

$3n^{\sqrt{n}}$ $\quad$ $n^{\sqrt{n}}$

$f(n) > g(n)$

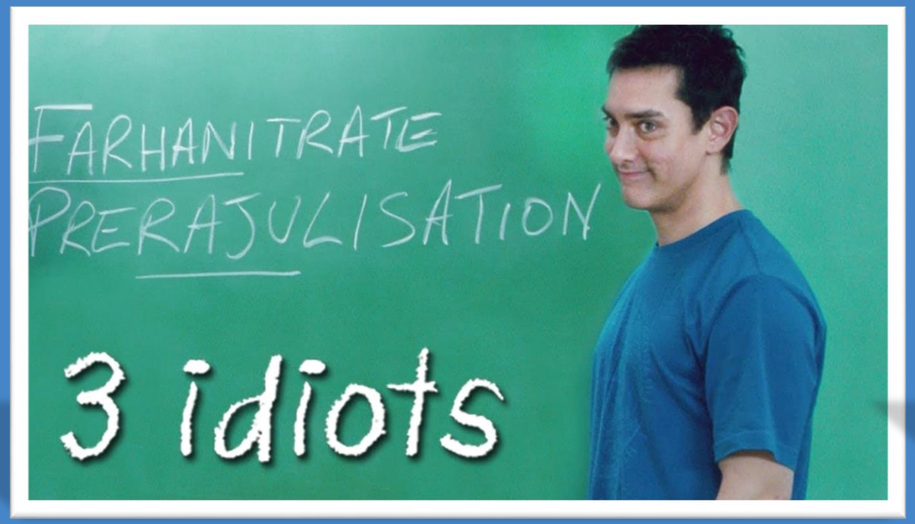
~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Next Module

Designing Techniques of Algorithms, Recurrence Relations, Designing and Analysis of Sorting Algorithms (Insertion Sort, Shell Sort, Quick Sort, Merge Sort, Heap Sort, Liner Sorting (Count Sort)), Red Black Trees, and Searching

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA

Choose Wisely



Happy Learning!

If you have any doubts, or queries , can be discussed in the C-11, Room 310, AB-1.

or share it on WhatsApp 8586968801

if there is any suggestions, please write it on gaurav.kumar@gla.ac.in

~Dr Gaurav Kumar, Asst. Prof, CEA, GLA