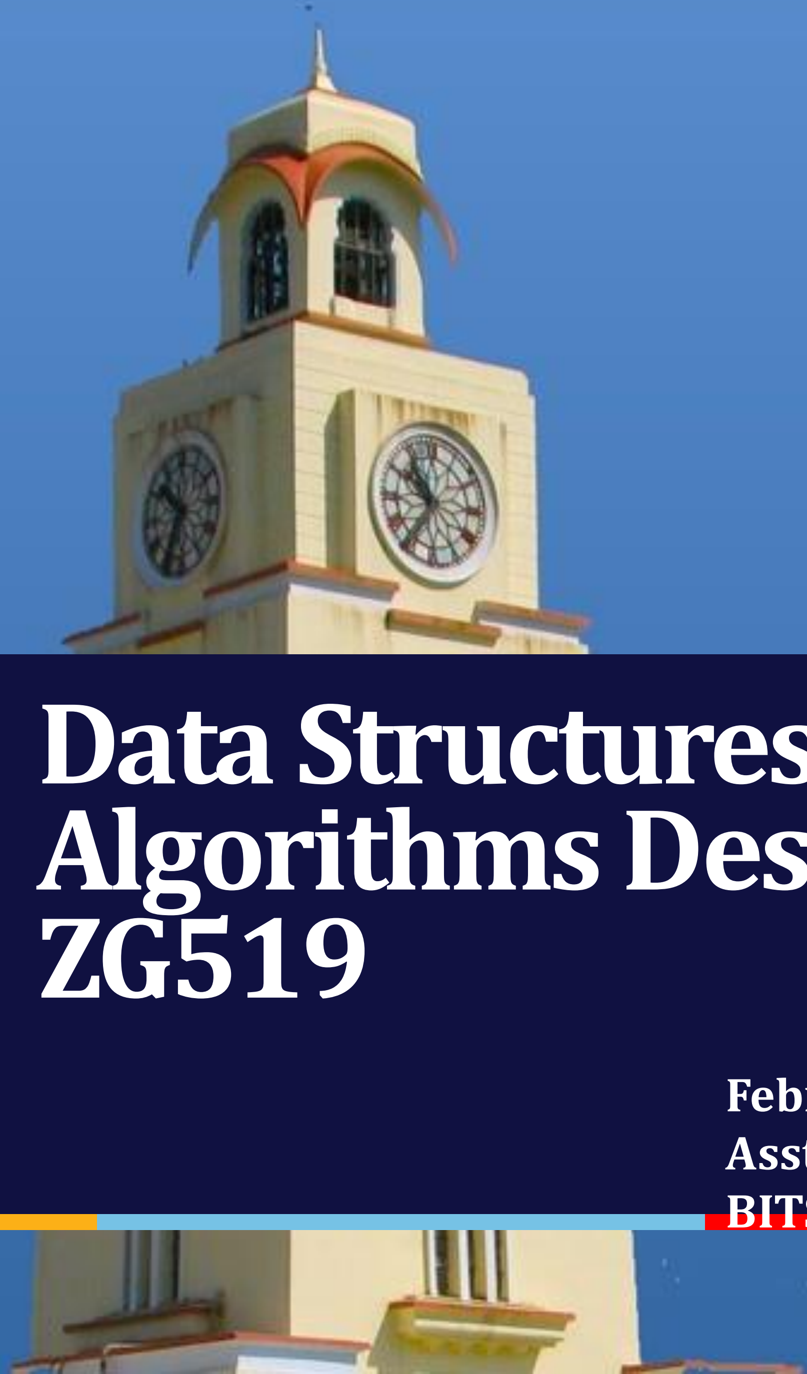


Data Structures and Algorithms Design ZG519

BITS Pilani
Hyderabad Campus

Febin.A.Vahab
Asst.Professor(Offcampus)
BITS Pilani,Bangalore

Course Objectives

No	Objective
CO1	Introduce mathematical and experimental techniques to analyze algorithms
CO2	Introduce linear and non-linear data structures and best practices to choose appropriate data structure for a given application
CO3	Teach various dictionary data structures (Lists, Trees, Heaps, Bloom filters) with illustrations on possible representation, various operations and their efficiency
CO4	Exposes students to various sorting and searching techniques
CO5	Discuss in detail various algorithm design approaches (Greedy method, divide and conquer, dynamic programming and map reduce) with appropriate examples, methods to make correct design choice and the efficiency concerns
CO6	Introduce complexity classes , notion of NP-Completeness, ways of classifying problem into appropriate complexity class
CO7	Introduce reduction method to prove a problem's complexity class.

TEXT BOOKS

No	Author(s), Title, Edition, Publishing House
T1	Algorithms Design: Foundations, Analysis and Internet Examples Michael T. Goodrich, Roberto Tamassia, 2006, Wiley (Students Edition).
R1	Data Structures, Algorithms and Applications in Java, Sartaj Sahni, Second Ed, 2005, Universities Press
R2	Introduction to Algorithms, TH Cormen, CE Leiserson, RL Rivest, C Stein, Third Ed,2009, PHI

Topics

- **Analysing Algorithms**
- Elementary Data Structures
- Non-Linear Data Structures
- Dictionaries
  - Ordered/Unordered Dictionaries
  - Hash Tables
- Binary Search Trees
- Algorithm Design Techniques
  - Greedy Method
  - Divide and Conquer
  - Dynamic Programming
  - Graph Algorithms
- Complexity Classes

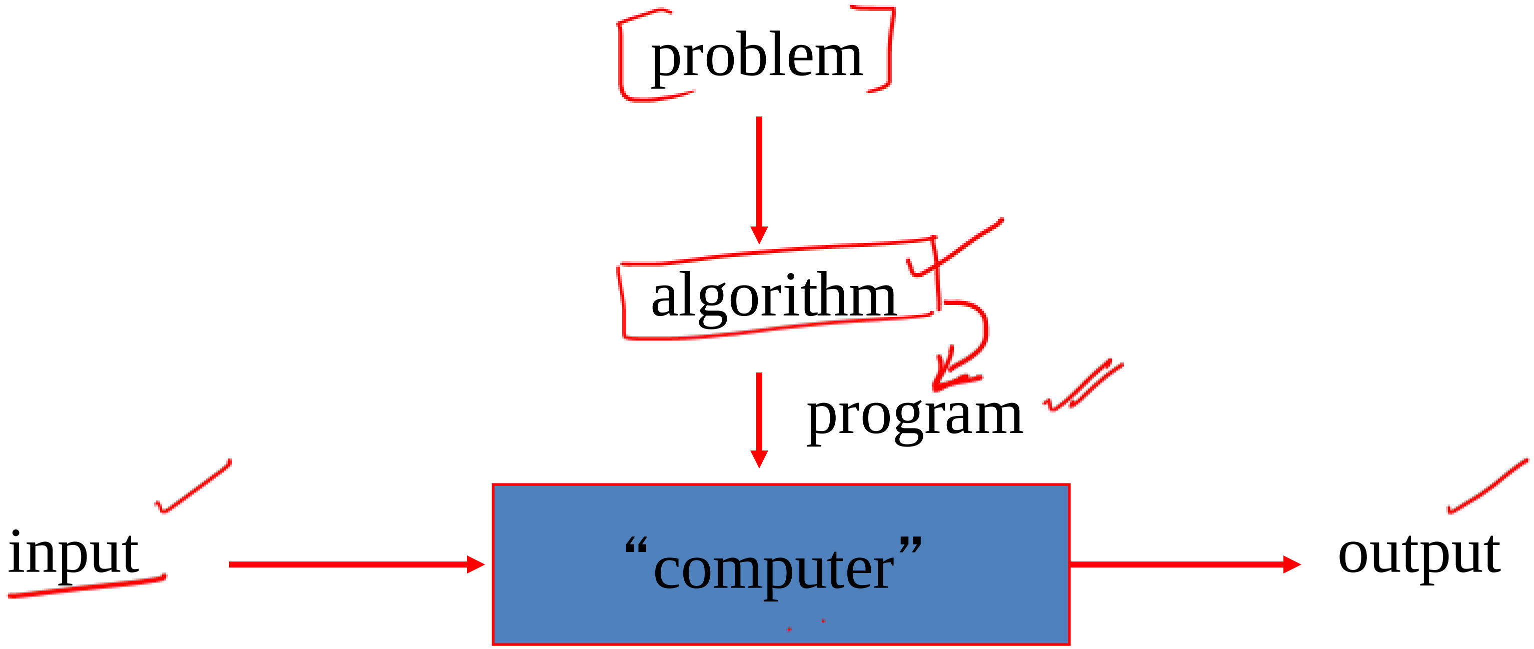
SESSION 1 -PLAN

Contact Sessions(#)	List of Topic Title	Text/Ref Book/external resource
1	Algorithms and it's Specification, Experimental Analysis, Analytical model-Random Access Machine Model, Counting Primitive Operations, Basic Operation method, Analyzing non recursive algorithms, Order of growth	T1: 1.1, 1.2

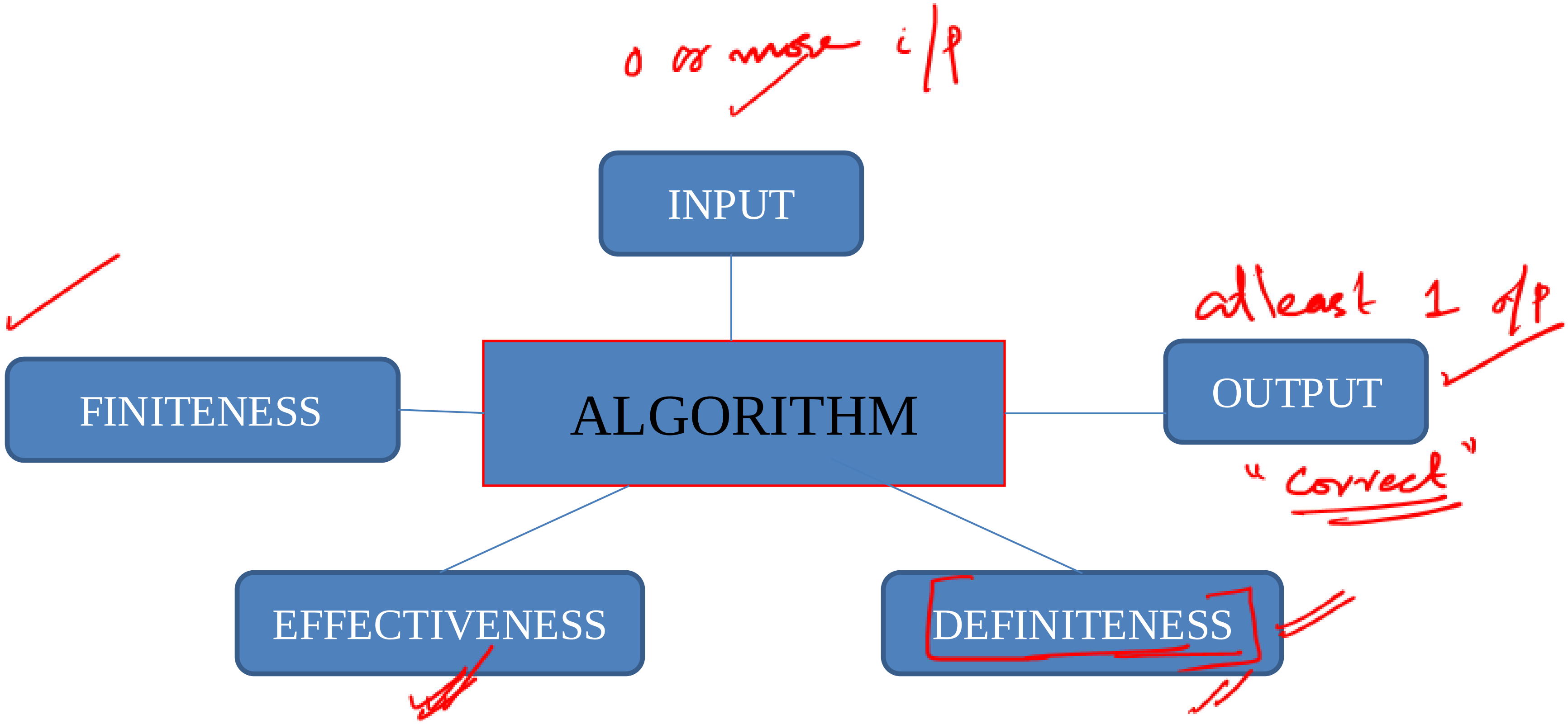
Algorithm

- An algorithm is a **finite sequence of unambiguous, step by step instructions** followed to accomplish a given task.

Notion of algorithm



Properties of Algorithms



Properties of Algorithms

- Finiteness:
  - An algorithm must terminate after finite number of steps.
- Definiteness:
  - The steps of the algorithm must be precisely defined. Each instruction must be clear and unambiguous.
- Effectiveness:
  - The operations of the algorithm must be basic enough to be put down on pencil and paper.
- Input-Output:
  - The algorithm must have certain(zero or more) initial and precise inputs, and outputs that may be generated both at its intermediate and final steps.

EUCLID'S ALGORITHM

ALGORITHM *Euclid(m, n)*

//Computes gcd*(m, n)* by Euclid's algorithm

//Input: Two nonnegative, not-both-zero integers *m* and *n*

//Output: Greatest common divisor of *m* and *n*

Step 1 If *n* = 0, return the value of *m* as the answer and stop; otherwise,

proceed to Step 2.

Step 2 Divide *m* by *n* and assign the value of the remainder to *r*.

Step 3 Assign the value of *n* to *m* and the value of *r* to *n*. Go to Step 1.

m	n	r m mod n
6	10	6
10	6	4
6	4	2
4	2	0
2	0	

gcd(6, 10) = 2

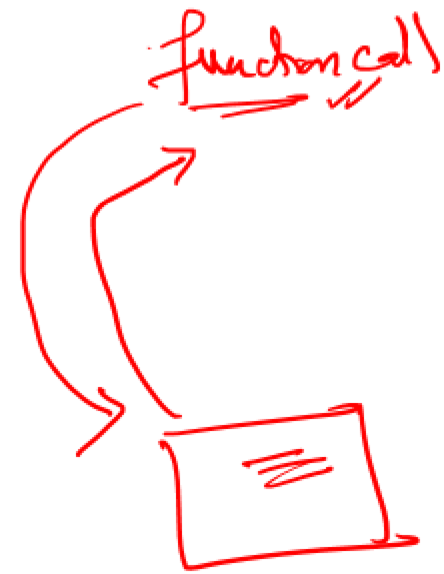
Analysis of algorithms

- Design the most efficient algorithm.

- SPACE
- TIME

EFFICENCY

- SPACE →
  - Program space
  - Data space
  - Stack space



Analysis of algorithms

- TIME EFFICIENCY → how fast ?
  - Speed of computer
  - Choice of programming language
  - Compiler used
  - ***Choice of algorithm***
  - ***Number(size) of inputs***

→ Speed of comp
→ Choice of
→ pgmg lang
compiler

$\boxed{n}$

↳ $n \propto$ size of the data

$\boxed{\text{Time efficiency} \rightarrow n}$

Analysis of algorithms

- <u>Experimental Analysis</u>
  - Write a program implementing the algorithm
  - Execute the program on various test inputs and record the actual time spent.
  - Use system calls (Ex: System.currentTimeMillis()) to get an accurate measure of the actual running time.

→ representative

Analysis of algorithms

innovate achieve lead

- <u>Limitations of experimental studies</u>
  - Implementation is a must.
  - Execution is possible on limited set of inputs.
  - Inputs must be representatives of real time scenarios
  - If we need to compare two algorithms ,we need to use the same environment (like hardware, software etc)

Analysis of algorithms

- Analytical Model
  - High level description of the algorithm.
  - Takes into account all possible inputs.
  - Allows one to evaluate the efficiency of any algorithm in a way that is independent of the hardware and the software environment

Pseudo-code

- A mixture of natural language and high level programming concepts that describes the main ideas behind a generic implementation of a data structure and algorithms.
- Find maximum element in an array :

Algorithm *arrayMax(A, n)*

Input: An array *A storing n>=1 integers*

Output: The maximum element in *A*

```
currentMax ←A[0]
for i ← 1 to n - 1 do
if currentMax <A[i] then
    currentMax ← A[i]
return currentMax
```

Pseudo-code

- Expressions
  - Use standard mathematical symbols to describe numeric and Boolean expressions.
  - Uses ← for assignment.(= in Java)
  - Use = for the equality relationship.(== in Java)
- Method declaration
  - Algorithm name(param1,param2…)
- Method returns: **return** value
- Control flow
  - **if** … **then** … [**else** …]
  - **while** … **do** …
  - **repeat** … **until** …
  - **for** … **do** …
  - Indentation replaces braces

Primitive Operations

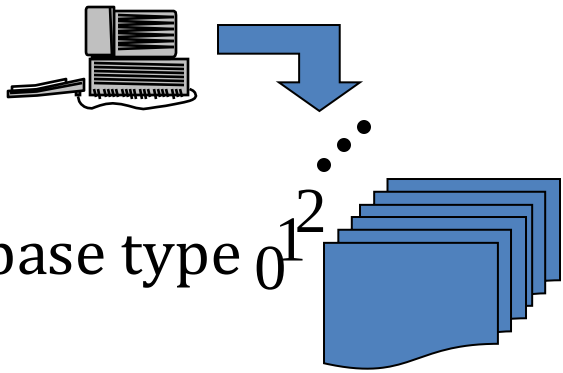
Primitive operations are basic computations performed by an algorithm

- Assigning a value to a variable
- Calling a method
- Performing arithmetic operation
- Indexing into an array
- Following an object reference
- Returning from a method
- Comparing two numbers

*b = &a

The Random Access Machine (RAM) Model

- **NOT** Random access memory
- A CPU connected to a bank of memory cells.
- Each memory cell stores a word-Value of a base type
  - Number
  - Character string
  - An address etc



- Random Access: Ability of CPU to access an arbitrary memory cell with one *primitive operation*
- The CPU can perform any primitive operation in a *constant number of steps* which do not depend on the size of the input.

The Random Access Machine (RAM) Model

- Approach of simply counting primitive operations.
- Each primitive operation corresponds to constant time instruction.
- *A bound on the number of primitive operations* → *running time of that algorithm.*

Counting Primitive Operations

increment i

- Focus on each step of the algorithm and count the primitive operation it takes
- Consider the arrayMax Algorithm.

8
7
5
3
2

$i \leftarrow 1$
$i < n$

Algorithm *arrayMax(A, n)*

currentMax ← *A[0]* ← 2
for *i* ← *1 to n - 1 do* ← 1 + n
if *currentMax* < *A[i] then* ← 2
currentMax ← *A[i]* ← 2
$i \leftarrow i+1$ ← 2
return *currentMax* ← 1

n - 1

6 4

- Assigning a value to a variable
- Calling a method
- Performing arithmetic operation
- Indexing into an array
- Following an object reference
- Returning from a method
- Comparing two numbers

Counting Primitive Operations

Statement	# of primitive operations
currentMax ←A[0] Indexing into array and assignment	2
for *i ← 1 to n - 1 do*	
• i initialised to 1	1
i<n is verified , comparing 2 numbers(1 po) The comparison is performed n times(at the end of each iteration of the loop)	$\boxed{n}$
if currentMax <A[i] Comparison and Indexing	2
currentMax ← A[i] Assignment and indexing	2
Counter incremented $i \leftarrow i+1$ Summing and assignment	2
Either 4 or 6 primitive operations, each iteration	

Counting Primitive Operations

Statement	# of primitive operations
Body of the loop	n-1
Total	
• If condition satisfied	6(n-1)
• If condition not satisfied	4(n-1)
return *currentMax*	1

The number of primitive operations executed by the algorithm

<u>Atleast </u>(Best case): 2+1+n+4(n-1)+1=5n

<u>Atmost </u>(worst case):2+1+n+6(n-1)+1=7n-2

Basic operation Method

- Identify the basic operation
- Basic operation???
  - Operation that contributes most towards the running time of an algorithm.
  - Statement that executes maximum number of times.

St 1 ← 1ms

St 2 ← 2ms

St 3 ← 1ms

St 4 ← 5ms

Basic operation Method

- Identify the basic operation
- Obtain the total number of times that operation is executed.

General Plan for Non recursive algorithms

- Decide on parameter n indicating input size
- Identify algorithm's basic operation
- Check whether the number of times the basic operation is executed depends only on the input size n. If it also depends on the type of input, investigate worst, average, and best case efficiency separately.
- Set up summation reflecting the number of times the algorithm's basic operation is executed.
- Simplify summation using standard formulas

Basic operation Method

ArrayMax

Algorithm arrayMax(A, n)

currentMax ←A[0]

for i ← 1 to n - 1 do

if currentMax <A[i] then

currentMax ← A[i]

return currentMax

Basic operation Method

- **Input Size**: n
- **Basic Operation:** Comparison in the for loop
- Depends on worst case or best case? No, has to go through the entire array
- $T(n)$ = number of comparisons
- $T(n) = \sum_{i=1}^{n-1} 1 = n\text{-}1$

$n-1-1+1$

$T(n)$

$T(n) = n-1$

Asymptotic

Basic operation Method

Matrix multiplication

3 x 4 5 x 3

3x4 4x5

3 x 5

ALGORITHM *MatrixMultiplication*$(A[0..n-1, 0..n-1], B[0..n-1, 0..n-1])$

//Multiplies two square matrices of order n by the definition-based algorithm
//Input: Two $n \times n$ 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**
      $C[i, j] \leftarrow C[i, j] + \boxed{A[i, k] * B[k, j]}$
return C

Basic operation Method

Matrix multiplication

- **Input Size**: n
- **Basic Operation** :Multiplication
- Depends on worst case or best case? No, has to go through the entire array
- $M(n)$ = number of comparisons

Basic operation Method

Matrix multiplication

n^3

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1$$

$2(n-1) - 0 + 1$

$M(n) = n$

$2n$

$$M(n) = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1}\sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1}\sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

$n\,[n-1-0+1]$

$n^2\,(n-1-0+1) = n^3$

$M(n) = n^3$

$8 \Rightarrow 64$

$2 = \rightarrow 4$

Basic operation Method

- 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 ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] = A[j] return false
    return true
```

Basic operation Method

- Element uniqueness problem
- **Input's size:**n, the number of elements in thearray.
- **Algorithm's basic operation:**The comparison of two elements
- The number of element comparisons depends not only on n but also on whether there are equal elements in the array and, if there are, which array positions they occupy.
- We will limit our investigation to the worst case only

Basic operation Method

8 3 2 4 5 5

- **Worst Case of this problem:** The worst case input is an array for which the number of element comparisons is the largest among all arrays of size n
- **Two kinds of worst-case inputs**:
- The algorithm does not exit the loop prematurely - arrays with no equal elements
- The last two elements are the only pair of equal elements

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \in \quad (n^2).$$

Order of growth

- How the value of 'n' affects the time complexity of the algorithm?

OR

- How time complexity grows wrt 'n'?

1

$M(n) = n^3$ Cubic

$f(n) = n^2$ quadratic

$T(n) = \boxed{n-1}$ Linear

$\rightarrow 1$

$5 \propto 5$

$10 \propto c \times 10$

Order of growth

- The running time of an algorithm can be

CONSTANT	1
LOGARITHMIC	Log N
LINEAR	N
N Log N	-
QUADRATIC	N^2
CUBIC	N^3
EXPONENTIAL	2^N
FACTORIAL	N!

2

4

8

64

Order of growth Exercise

Asymptotic

- Compare the order of growth
  - n(n+1) and 200n ^{2} --- Same
  - 100n ^{2} and 0.01 n ^{3}
  - Log n and ln n — Same
  - 2 ^n-1 and 2 ^n — Same

<u>Refer 1.3.2 in text book for Logarithms and Exponents</u>

Order of growth

Consider a program with time complexity $O(n^2)$.

- For the input of size n, it takes 5 seconds.
- If the input size is doubled (2n). –
- **then it takes 20 seconds**.

$(2n)^2$

Consider a program with time complexity O(n).

- For the input of size n, it takes 5 seconds.
- If the input size is doubled (2n). –
- **then it takes 10 seconds.**

Consider a program with time complexity $O(n^3)$.

- For the input of size n, it takes 5 seconds.
- If the input size is doubled (2n) . –
- **then it takes 40 seconds**.

Order of growth

	constant	*logarithmic*	*linear*	*N-log-N*	*quadratic*	*cubic*	*exponential*
n	**O(1)**	**O(log *n*)**	**O(*n*)**	**O(*n* log *n*)**	**O(n^2)**	**O(n^3)**	**O(2^n)**
1	1	1	1	1	1	1	2
2	1	1	2	2	4	8	4
4	1	2	4	8	16	64	16
8	1	3	8	24	64	512	256
16	1	4	16	64	256	4,096	65536
32	1	5	32	160	1,024	32,768	4,294,967,296
64	1	6	64	384	4,069	262,144	1.84 x 10^{19}

Exercise

- Which kind of growth best characterizes each of these functions?
- $(3/2)^n$
- $3n$
- 1
- $(3/2)n$
- $2n^3$
- 2^n
- $3n^2$
- 1000

Exercise-Solution

- Which kind of growth best characterizes each of these functions?

	Constant	Linear	Polynomial	Exponential
$(3/2)^n$				✓
3n		✓		
1	✓			
(3/2)n		✓		
$2n^3$			✓	
2^n				✓
$3n^2$			✓	
1000	✓			

Examples

What is the order of growth of the below function?

```
int fun1(int n)
{
  int count = 0;
  for (int i = 0; i < n; i++)
    for (int j = i; j > 0; j--)
      count = count + 1;
  return count;
}
```

Ans:O(n^2)

THANK YOU!