Subject code:- 21CS42

Subject :- Design & Analysis of Algorithm

Module-01

Introduction

Topic-01

What is Algorithm? It's properties

1. What is Algorithm? What are the properties of Algorithm? Explain with an Example

Ans:-

Algorithm

The Algorithm is defined as a collection of unambiguous Instruction occuring in some specific sequence & such an Algorithm should produce output for given set of Input in finite Amount of time.

Properties of Algorithm

1. Non-Ambiguity

* Each step in an Algorithm should be non-Ambiguous.
* That Means Each Instruction should be clear & precise.

2. Range of Input:-

* The Range of Input should be specified.
*

3. Multiplicity:-

* The same Algorithm can be Represented in several different ways.
* That means we can write in simple English the sequence of Instructions (or) we can write it in the form of pseudo Code

4. Speed :-

* The Algorithm ar written using some specific Ideas.
* But such Algorithm should be Efficient & should produce the output with fast speed.

5. Finiteness :-

* The Algorithm should be finite.
* That means after performing Required Operations it should Terminate

Topic-02

Algorithm specification - Using Natural Language, Using Pseudo code Convention

* Algorithm is a procedure consisting of heading & Body.

* Then in the heading section we should write following things.

// Problem Description:
// Input:
// Output:

* Then Body of an Algorithm is written in which various Programming constructs like if, for, while, & some Assignment statements may be written.

* The compound statements should be enclosed within { and } Brackets.

* Single line comments are written Using // as Beginning of comment.

* The Identifier should Begin by letter & not by digit.

* ~~Assign~~ Using Assignment Operator ← an assignment statement can be given.

Ex. variable ← expression

Example

write an Algorithm to Count the sum of n numbers

```
Algorithm sum(L, n)
// Problem Description:- This Algorithm is for finding The
//sum of given n numbers
//Input : 1 to n numbers
//output : The sum of n numbers
    result ← 0
    for i ← 1 to n do i ← i+1
        result ← result + i
    return result
```

Topic-03

Fundamentals of Algorithmic Problem Solving

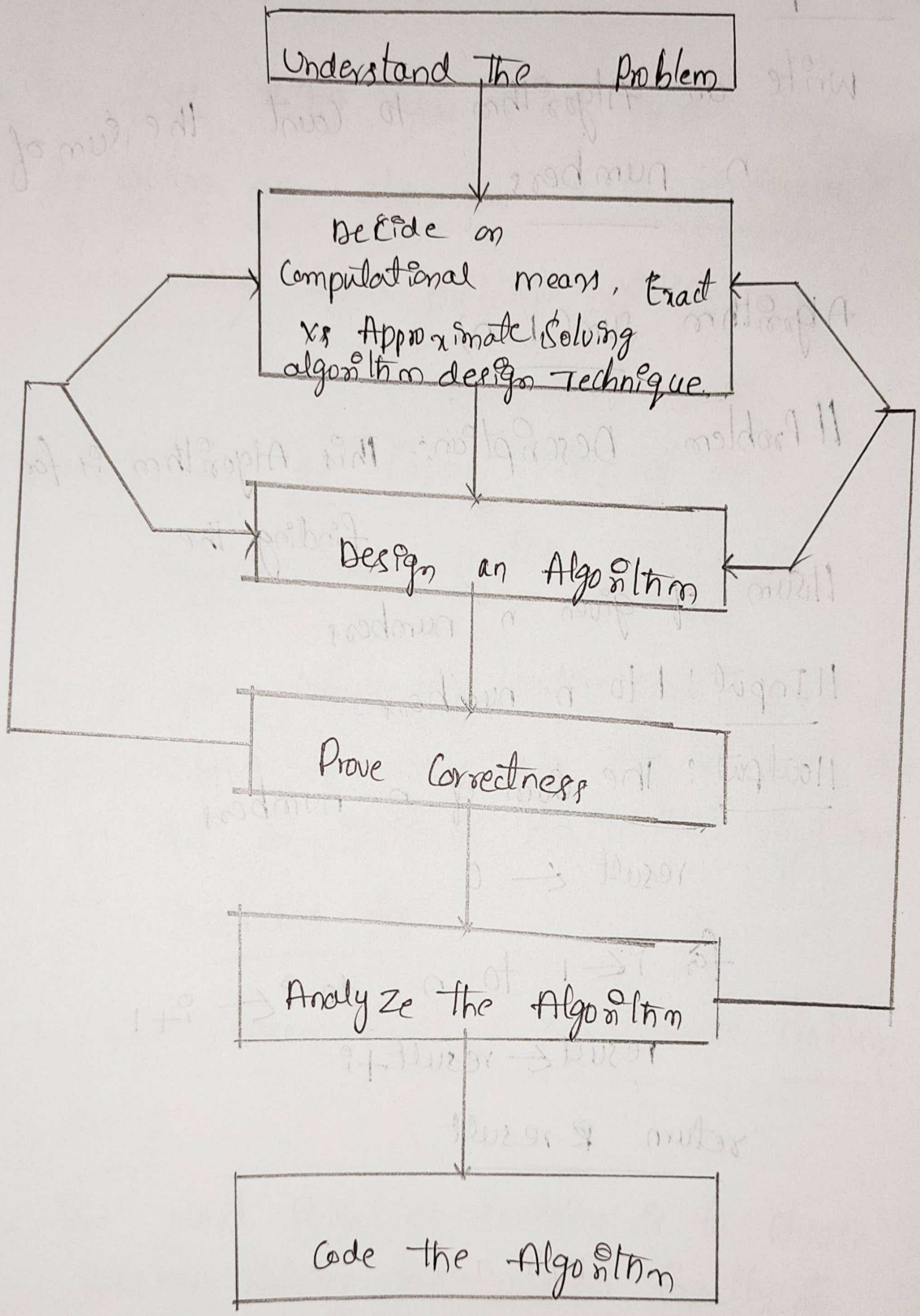


fig:- Algorithm Design & Analysis process

1. Understand the problem

* The first thing you need to do Before designing an algorithm is to understand Completely The problem given.

2. Ascertaining the Capabilities of the Computational Device

* Once you completely understand a problem.
* you Need to Ascertain the capabilities of the computational device the algorithm is indended for.

3. Choosing Between Exact & Approximate Problem Solving

* The Next Principal decision is to choose Between Solving the problem Exactly or Solving it Approximately.

4. Algorithm Design Techniques

* It is a General Approach to Solving Problems Algorithmically that is

applicable to a variety of problems from different areas of computing.

5. Designing an Algorithm & Data Structure

* one should pay close attention to choosing data structures appropriate for the operations performed by the Algorithm.

6. Methods of Specifying An Algorithm

* once you have designed an Algorithm you need to specify it in some fashion.

7. Proving an Algorithm's Correctness

* once an Algorithm has been specified you have to prove it's correctness.

8. Analyzing An Algorithm

* We Usually want our Algorithms to process several qualities.
* After correctness by far the most Important is Efficiency

9. Coding An Algorithm

* Most of Algorithms are designed to be ultimately Implemented as Computer programs.
* Programming an Algorithm presents Both a peril & an opportunity.

Topic-05

Analysis Framework- Time Efficiency, & Space Efficiency

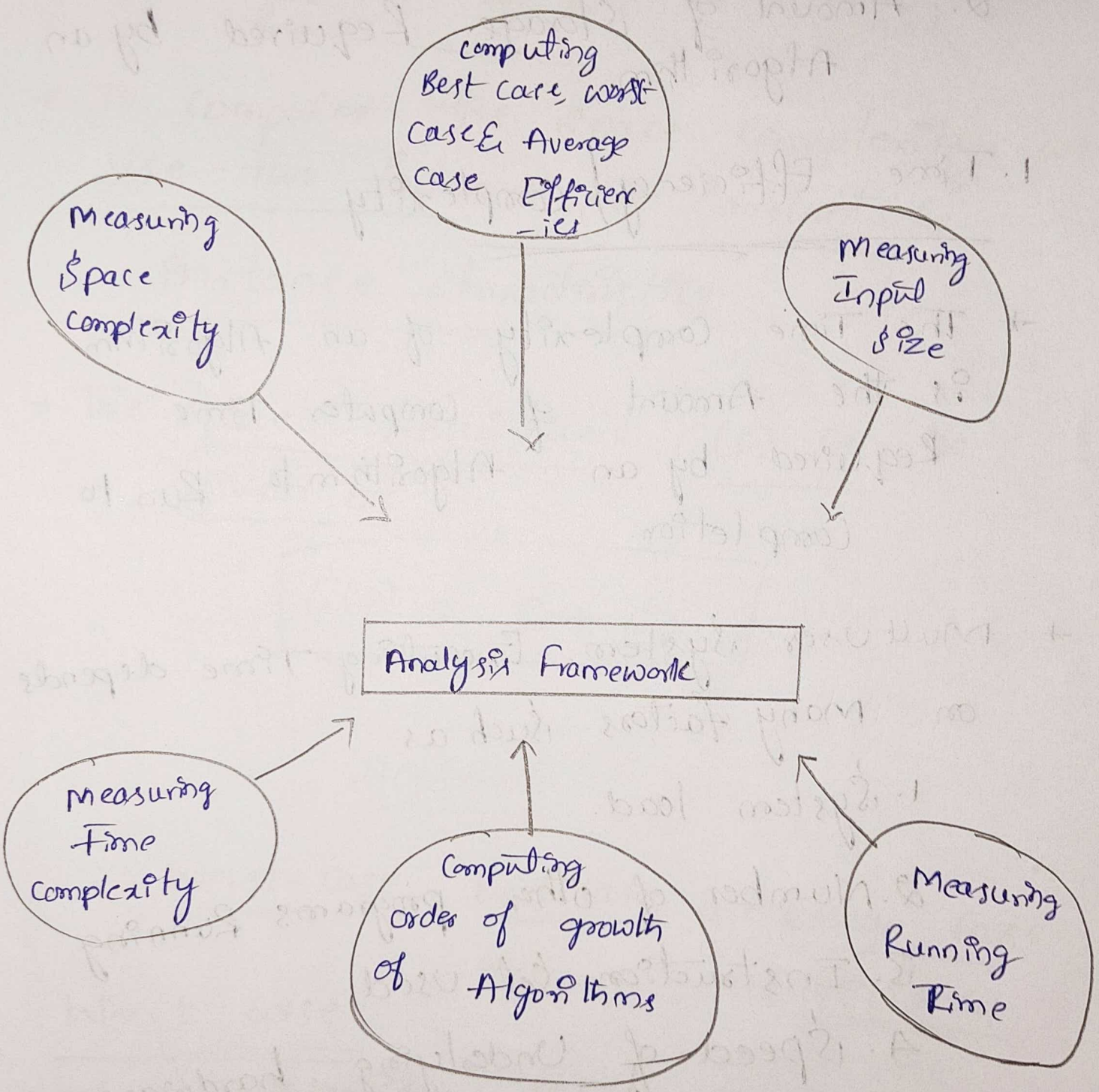


Fig:- Analysis Framework

* We can Measure The performance of an Algorithm by computing Two factors

1. Amount of Time Required by an Algorithm to Execute.
2. Amount of Storage Required by an Algorithm.

1. Time Efficiency / Complexity

* The Time Complexity of an Algorithm is the Amount of Computer Time Required by an Algorithm to Run to Completion.

* Multiuser System, Executing Time depends on many factors such as
  1. System load.
  2. Number of other programs Running
  3. Instruction Set used.
  4. Speed of Underlying hardware.

* The Time Complexity is therefore given in Terms of Frequency Count

2. Space complexity

* The space complexity can be defined as Amount of memory required by an algorithm to run

* To compute the space complexity we use two factors : Constant & instance characteristics

* The space Requirement S(P) can be given as

$$S(P) = C + S_P$$

Topic-06

Worst-case, Best case, & Average case efficiency

Worst case

If an Algorithm Takes maximum Amount of Time to Run to completion for a

specific set of Input then it is called Worst case Time complexity

Ex: while searching an Element by Using Linear Search Method If desired element is placed at the end of the list then we get worst Time complexity

Best case Efficiency

If an Algorithm takes minimum Amount of Time to Run to Completion for a specific set of Input then it is called Best case Time Efficiency

Ex: while searching a particular element by Using Sequential search we get the desired element at first place itself then it is called Best case Time Efficiency

Avarage case Efficiency

- The Time complexity that we get for certain set of Inputs is as a Avarage same.
- Then for corresponding Input such a Time Complexity is called Average case Time complexity

Chapter-02

Performance Analysis

Topic-01

Estimating Space Complexity & Time Complexity of Algorithm

- ~~Both~~ Time & Space Eff

~~Ref~~ Refer ~~Sec~~ Topic-02

Chapter-03

Asymptotic Notations

Topic-01

Big-oh Notation (O)

Big-oh Notation (O)

* The Big-oh Notation is denoted by 'O'.
* It is a Method of Representing the Upper Bound of Algorithm's Running Time.

Definition

Let $f(n)$ and $g(n)$ be two non-negative functions.

Let no and Constant c are two Integers such that no denotes some value of Input &

$n > n_0$.

* Similarly c is some constant such that $c > 0$.

* We can write $f(n) \leq c * g(n)$

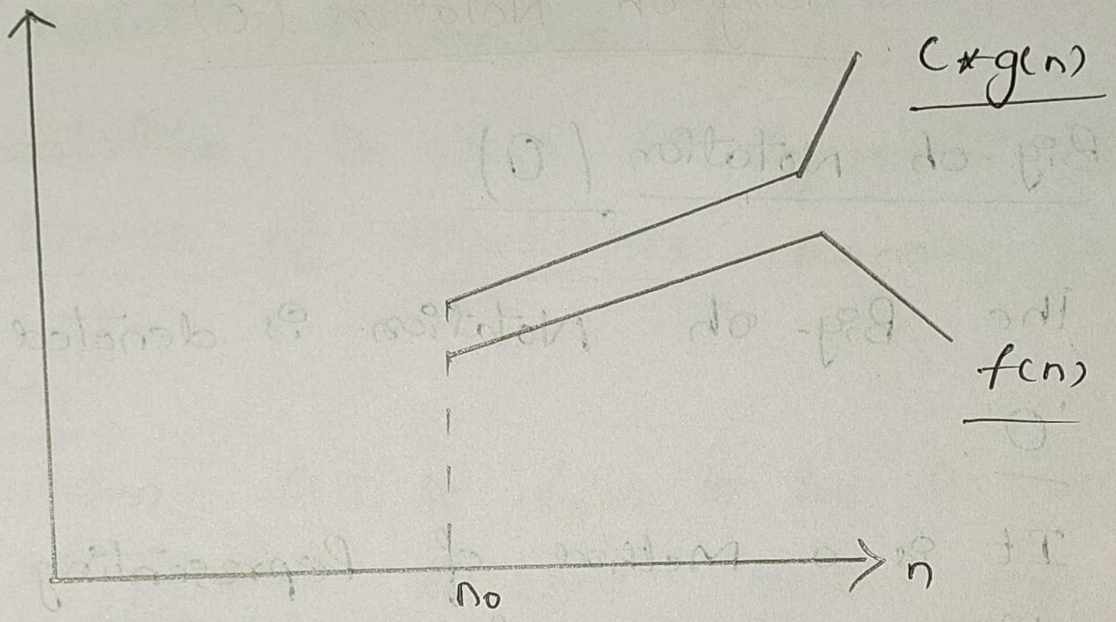


Fig:- Big-oh Notation

* Here f(n) is Big oh of g(n).
* It is also denoted as $f(n) \in O(g(n))$.

Topic-02

Omega Notation (Ω)

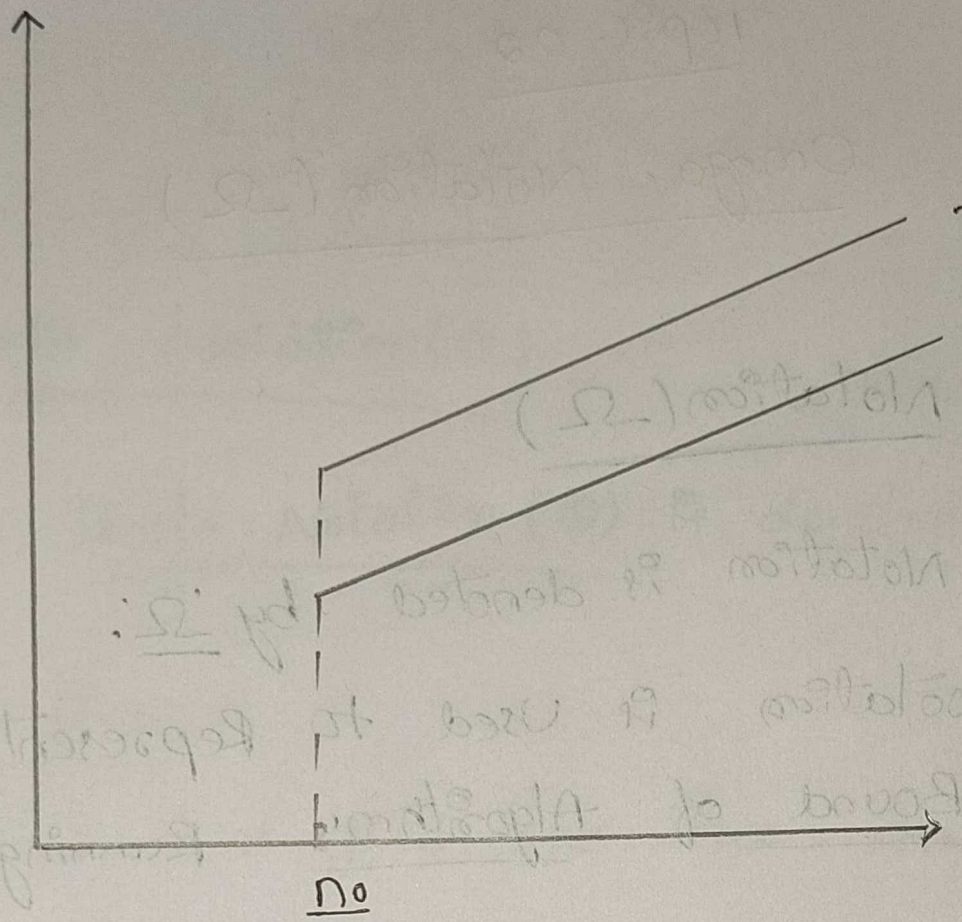
Omega Notation (Ω)

* Omega Notation is denoted by 'Ω'.
* This Notation is used to Represent the lower Bound of Algorithm's Running Time.

Definition

A Function $f(n)$ is said to be in $\Omega(g(n))$ if $f(n)$ is Bounded Below by some positive constant multiple of $g(n)$ such that

$f(n) \geq c * g(n)$ For all $n \geq n_0$



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

Fig: Omega- Notation

* It is denoted as $f(n) \in \Omega(g(n))$
* Above figure illustrates the curve for Ω notation

Topic-03

Theta Notation (Θ)

Theta Notation (Θ)

* The Theta Notation (Θ) is denoted by Θ.

Definition

Let $f(n)$ and $g(n)$ be two non-negative functions.

There are two positive constants namely C_1 & C_2 such that

$$C_2 g(n) \leq f(n) \leq C_1 g(n)$$

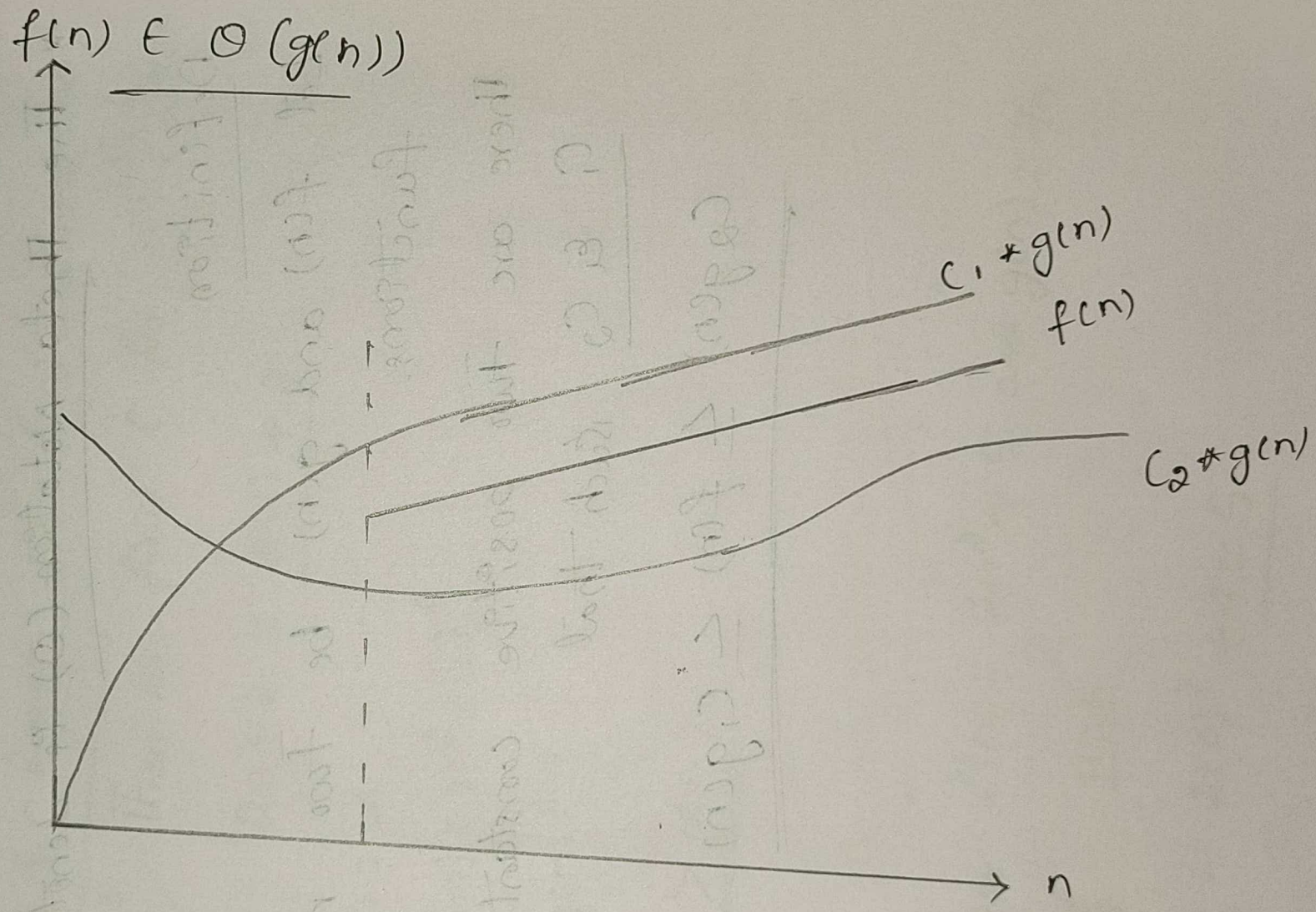


fig:- Theta Notation $f(n) \in \Theta(g(n))$

Example of Big-oh notation (O)

Ex: Consider function $f(n) = 2n+2$ & $g(n) = n^2$

Then we have to find some constant C, so that $f(n) \leq C * g(n)$. As $f(n) = 2n+2$ and $g(n) = n^2$ Then we find C for $n = 1$ Then

$$f(n) = 2n+2 = 2(1)+2$$

$$f(n) = 4$$

and $g(n) = n^2 = (1)^2$

$$g(n) = 1$$

i.e. $f(n) > g(n)$

Example of Omega Notation (Ω)

Ex: Consider $f(n) = 2n^2 + 5$ and $g(n) = 7n$

Then if $n = 0$

$$f(n) = 2(0)^2 + 5 = 5$$

$g(n) = 7(0)$

$= 0$ true

i.e. $f(n) > g(n)$

Example of Θ Notation

Ex). If $f(n) = 2n + 8$ and $g(n) = 5n$

where $n \geq 2$

similarly $f(n) = 2n + 8$

$g(n) = 7n$

i.e $5n < 2n + 8 < 7n$ for $n \geq 2$

Here $C_1 = 5$ and $C_2 = 7$ with $n_0 = 2$.

The Theta (Θ) notation is more precise with Both Big oh and Omega Notation

Topic-04

Basic Efficiency class

Basic Efficiency class

Name of Efficiency class	order of growth	Description	Example
Constant	1	As Input size grows the we get larger Running time	Scanning array Elements.
Logarithm -ic	logn	When we get logarith -mic running time then it is sure that the Algorithm does not consider all its input rather the problem is divided into smaller parts on each iteration	Performing Binary search operation
Linear	n	The Running time of algorithm depends on the Input size n	Performing sequent -ial search operation.
nlogn	nlogn	some instance of input is considered for the list of size n.	Sorting matrix elements

Quadratic	n^2	when the Algorithm has two nested loops then this type of efficiency occurs.	Scanning matrix Elements.
Cubic	n^3	when the Algorithm has three nested loops then this type of Efficiency occurs	Performing matrix multiplication.
Exponential	2^n	when the Algorithm has very faster rate of growth then this type of Efficiency occurs	Generating All subsets of n Elements.
Factorial	$n!$	when an Algorithm is computing all the permutations then this type of Efficiency occurs	Generating all permutations.

Topic-05

Mathematical Analysis of Non-Recursive & Recursive Algorithms with Examples

Mathematical Analysis of Non Recursive Algorithm

General plan for Analyzing the Efficiency of Non-Recursive Algorithms

1. Decide the Input size Based on parameter n.
2. Identify Algorithms' Basic operations
3. Find the whether the Execution of Basic Operation depends Upon the Input sizes n.
   + Determine worst case, Average case, & Base Cases

4. Set up a sum for the number of times the Basic operation is Executed

5. Simplify the sum using standard formula & Rules

Examples: Finding the Element with maximum value in a given array

```
Algorithm Max-Element (A[0...n-1])
// Problem Description: This Algorithm is for finding the
//Maximum value element from the Array
//input : Array A[0...n-1]
// output : Returns the largest Element from Array
    Max_value ← A[0]
    for i ← 1 to n-1 do
    {
```

```
if ( A[i] > Max_value) then
    Max-value ← A[i]
}
return Max_value
```

Mathematical Analysis

Step-1:- The Input size is n i.e. Total number of Elements in array.

Step-2:- The Basic Operation is Comparision in loop for finding larger value

Step-3:- The Comparision is Executed on each Repetition of the loop

Step-4:- Let C(n) be the The Algorithm makes Comparision each Time the loop Executed.

Step-5:- Let Simplify the Sum $C(n) = \sum_{i=1}^{n-1} 1$

Mathematical Analysis of Recursive Algorithms

General plan for Analyzing the Efficiency of Recursive Algorithms

1. Decide the Input size Based on parameter n.
2. Identify Algorithms Basic Operations.
3. check how many times the Basic Operation is Executed.
4. Setup the Recursive Relation with some initial condition & Expressing the Basic operation.
5. Soloing the Recurrence we will Use the forward & Backward substitution Method.

Example: compute factorial of some number n.

The factorial of some number can be obtained by performing Repealed Multiplication for instance: if n=5 then.

step 1: n! = 5

step-2:- 4! * 5

step-3:- 3! * 4 * 5

step 4:- 2! * 3 * 4 * 5

step-5:- 1! * 2 * 3 * 4 * 5

step-6:- 1 * 1 * 2 * 3 * 4 * 5 as 0! = 1

Psudocode

Algorithm factorial (n)

// Problem Description: this Algorithm computes n! Using

// Recursive function

//Input Function: A non negative integer n

//output: Returns the factorial value.

```
if(n==0)
    return 1
else
    return factorial (n-1) * n
```

Mathematical Analysis

Step-1:- The factorial Algorithm works for Input size n.

Step-2:- The Basic operation is computing factorial is multiplication.

Step-3:- The Recursive Function call can be formulated as

$F(n) = f(n-1) * n$ where $n > 0$

Step-4:- In Step-3 the Recurrence Relation is obtained. $m(n) = m(n-1) + 1$

Chapter-03

Brute Force Design Technique

Topic-01:- Selection Sort

Selection Sort

* we start selection sort by scanning the entire given list to find its smallest element & Exchange it with the first element

* putting the smallest Element in its final position in the sorted list.

Algorithm

```
// Sorts a given array by Selection Sort
   (A[0.. - n-1])
// Input: An Array (A[0. - - n-1]) of
```

of orderable Elements.

Inputs:- Array A[0 . . . n-1] sorted in non-decreasing order.

```
for i ← 0 to n-2 do
    min ← i
    for j ← i+1 to n-1 do
        if A[j] < A[min]  min ← j
    swap A[i] and A[min]
```

Example: 89, 45, 68, 90, 29, 34, 17.

Sort the Using Selection Sort

89	45	68	90	29	34	17.
17	45	68	90	29	34	89
17	29	68	90	45	34	89
17	29	34	90	45	68	89
17	29	34	45	90	68	89
17	29	34	45	68	90	89
17	29	34	45	68	89	90

Topic-02

Sequential Search

Sequential Search

Encountered a Brute force Algorithm for the general searching problem it is called Sequential Search.

Algorithm: Sequential Search

// Implement Sequential Search with a search key as a Sequential

// Input: An Array A of n Elements & search key K.

// Output: The Index of the first Element in A[0 - - - n-1] whose value is equal to K or -1 if no such Element is found.

```
A[n] ← K
i ← 0
while A[i] ≠ K do
        i ← i + 1
if i < n return i
else return -1
```

Topic-03

String Matching Algorithm with complexity Analysis

* Given a string of n characters called the Text & a string of m characters ($m \leq n$) called the pattern.
* Find a substring of the Text i that matches the pattern.
* A Brute Force Algorithm for the String-matching problem is quite obvious.

* Align the pattern Against the first m characters of the Text & start matching.

* The corresponding pairs of characters from left to right until either all the m pairs of the character match.

Algorithm : Brute Force String match (T[0--n-1], P[0---m-1])

//Implements Brute-Force string matching

//Input: An Array T[0--n-1] of n characters Representing a text & an Array P[0:--m-1] of m characters Representing a Pattern.

//output: The Index of the first character in the text that starts a matching substring (or) -1 if the search is Unsuccessful

```
for i ← 0 to n − m do
    j ← 0
    while j < m and P[j] = T[i+j] do
        j ← j+1
    if j = m return i
    return −1
```

Example

NOBODY_NOTICED_HIM

```
NOT
  NOT
    NOT
      NOT
        NOT
          NOT
            NOT
              NOT
                NOT
```