UNIT - 1 - Introduction

Topic : 1

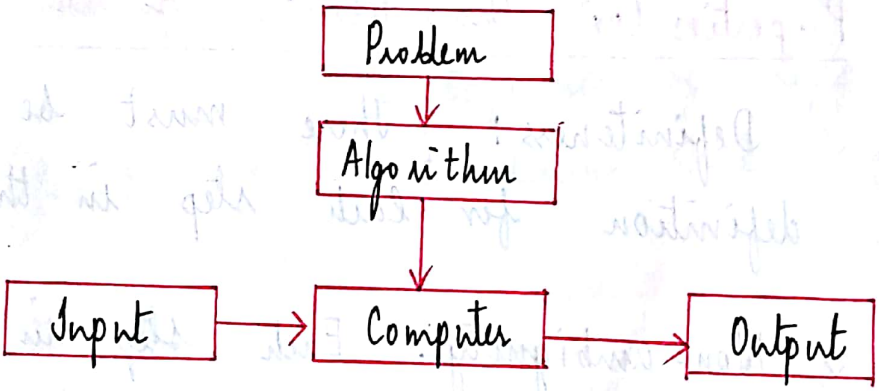
Notion of Algorithm:

UQ (2 mark)

Definition:- ✓ An algorithm is a sequence of unambiguous instructions occuring in some specific sequence and such an algorithm should Produce output for given set of input in a finite amount of time.

✓ An algorithm is a finite step by step procedure to achieve a required result.

✓ An algorithm is a sequence of operations performed on data that have to be organized in data structures.



Explanation:

✔ After understanding the Problem statement we have to create an algorithm for the given Problem.

✓ Then the algorithm is converted into Programming language and given to Computer.

✓ Then the Computer starts to execute.
✓ Then Process of execution requires Input
✓ Finally, result is Produced as Output.
✓ If the given input is invalid, it should raise error message.
✓ If it is Valid, Correct result is Produced as an Output.

✓ Note:-
- The same algorithm can be represented in several different ways.
- Algorithm for the same Problem can be based on different ideas. And same Problem can be solved with different speed.

Properties (or) Characteristics of an Algorithm:

✓ Definiteness: There must be exact definition for each step in the algorithm.

✓ Non-ambiguity: Each step in an algorithm should be non-ambiguous. It means each instruction should be clear and Precise. The instruction of an algorithm should not denote a conflict meaning.

✓ Finiteness: An algorithm must teriminate after a finite number of steps and for each step must be

Executable in finite amount of time.

✓ Input:
An algorithm has zero or more input but only finite number of inputs.

✓ Output:
An algorithm has one or more Output. The requirement of atleast one Output is essential.

✓ Effectiveness:

✓ An algorithm should be effective.

✓ This means, that each of the operation to be Performed in an algorithm must be sufficiently basic that it can, Principle be done exactly and in a finite length of time.

✓ If a sequence of steps is not finite, then it cannot be effective.

How to Write an Algorithm:

Algorithm is basically a sequence of instructions written in simple English language.

Rules for Writing an algorithm:

Rule 1: Algorithm is a Procedure Consisting of heading and body. The heading Contains of Keyword 'Algorithm' and the

name of the algorithm and Parameter list

The Syntax is:

Algorithm name (P1, P2 . . . Pn)

Algorithm — keyword; name — name of the algorithm; Pn — Parameter list

Rule 2: In the heading section, we write the following things:

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

Rule 3: In the body of the algorithm, in which Various Progeamming Constructs like if, for, while or some Assignments may be Written.

Rule 4: The Compound statements should be enclosed within { and } brackets.

Rule 5: Single line Comments are Written using //.

Rule 6: The Identifier should begin by letter and not by digit. It is not necessary to write data types explicitly for Identifiers

Rule 7: Using Assignment operator ← , an Assignment can be given as

Variable ← Expression

Rule 8: The inputting and outputting can be done using read and write.

Example:- Notion of an algorithm:

✓ An algorithm - to Calculate Gcd (Greatest Common divisor) of two numbers.

- A Problem can have more than one solution. There by to find the Gcd of 2 numbers, it has three solutions (or) three methods.

- The three methods to find the Gcd of 2 numbers are:

✓ Euclid's algorithm
✓ Consecutive Integer checking algorithm
✓ Repetitive factor [school method].

Method 1: Finding Gcd using Euclid's algorithm

```
Algorithm Gcd-Euclid (a, b)
// Problem Description : This algorithm
// Computes the Gcd of two numbers
// a and b using Euclid's algorithm
// Input : 2 integers a and b.
// Output : Gcd of a & b.
While ( b ≠ 0) do
{
    c ← a mod b
    a ← b
    b ← c
}
return a.
```

Example:

Consider there are two numbers, 30 and 18. Then we can find GCD as follows:

a	b	Remainder after a/b
30	18	30/18 = 12
18	12	18/12 = 6
12	6	12/6 = 0
6	0	6/0 = 6
		GCD = 6

$\therefore$ gcd(a, 0) = gcd = a

$\therefore$ gcd(b, 0) = b

The above table illustrates the Procedure for finding GCD of 2 no's using Euclidian's algorithm.

step 1: Divide a by b, assign the value of the remainder to 'c'.

step 2: Then assign the value 'b' to 'a' and the value of 'c' to 'b'.

step 3: Repeat step 1 and step 2 until the value of 'b' becomes zero.

step 4: If b = 0, then return the value of 'a'.

step 5: Stop

Method 2: Finding Gcd using Consecutive integer checking algorithm:

Algorithm Gcd_int_check(a, b)
// Problem description: This algorithm computes the Gcd of 2 no's using Consecutive integer checking algorithm.
// Input: 2 integers 'a' and 'b'
// Output: Gcd value of a and b.

```
t ← min(a, b)
While ( t >= 1) do
{
  if (a mod t == 0 and b mod t == 0)
  then
    return t;
  else
    t ← t-1;
}
return 1
```

Example: - Consider a = 15 and b = 10 then,

t = min(a, b) = min(15, 10) = 10.

	a	b	Description
(i)	15 mod 10 = 5 (a mod t)	10 mod 10 = 0 (b mod t)	As a mod t is not given a Zero Value it is not a gcd, ∴ set t = t - 1 = 10 - 1 t = 9
(ii)	15 mod 9 = 6 (a mod t)	10 mod 9 = 1 (b mod t)	As a mod t and b mod t is giving non-Zero, Value, it is not Gcd. ∴ set t = t-1 t = 8

	a	b	Description
(iii)	15 mod 8 = 7 (a mod t)	10 mod 8 = 2 (b mod t)	As a mod t and b mod t is giving non zero values, it is not GCD. ∴ set $t = t-1$ $t = 8-1$ ∴ $t = 7$
(iv)	15 mod 7 = 1	10 mod 7 = 3	As a mod t and b mod t is giving non-zero values, it is not GCD. ∴ set $t = t-1$ ∴ $t = 6$
(v)	15 mod 6 = 3	10 mod 6 = 4	As a mod t and b mod t is giving non-zero values, it is not GCD. ∴ set $t = t-1$ ∴ $t = 5$
(vi)	15 mod 5 = 0 (a mod t)	10 mod 5 = 0 (b mod t)	As a mod t and b mod t are zero values. t = 5 is a gcd value.

∴ gcd (15, 10) = 5

Thus, the Consecutive integers are been checked in this method.

Step 1:- Find min(a, b). Assign the minimum value to 't'.

Step 2:- Then divide 'a' by 't'. If remainder is '0', then divide 'b' by t. If remainder is again '0', then return the value of

't' as a gcd. If any one of the remainder is non-zero, then goto step 3.

Step 3: Decrement the value of 't' by 1 and go to step 2.

Method 3: Using Repetitive factors :-

Example :-

Consider a = 70 and b = 28 are the two numbers, then

2 | 70
7 | 35
5 | 5
1

2 | 28
2 | 14
7 | 7
1

∴ Thus,

70 = (2) x (7) x 5

28 = (2) x 2 x (7)

GCD = 2 x 7 = 14

Algorithm :-

Step 1 :- Find the Prime factors of 'a' and 'b'

Step 2 :- Identify the Common factors from both the Computations made in step 1.

Step 3 :- Compute the Product of the Common factors and return the result as the value of GCD.

Topic No: (2)

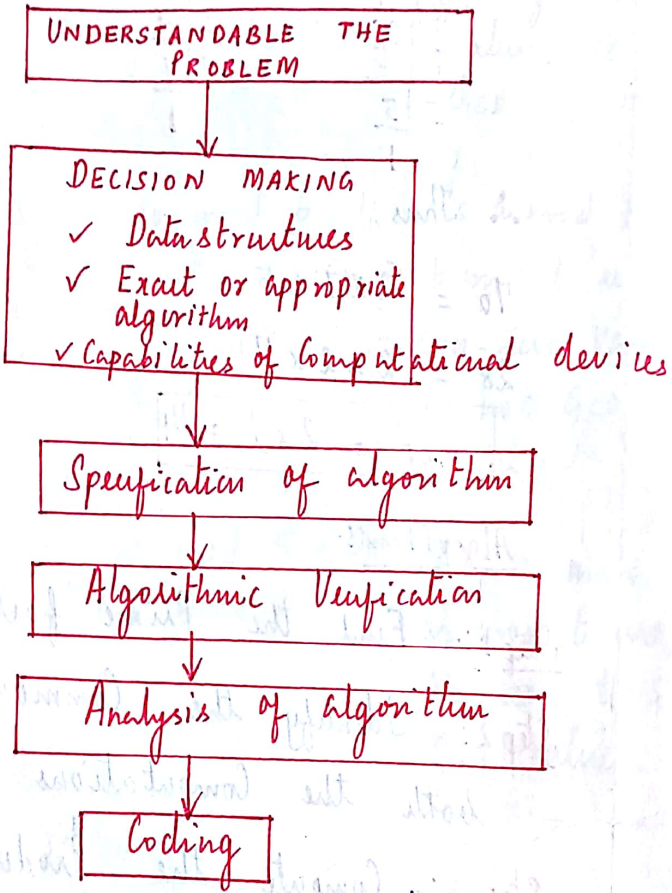
FUNDAMENTALS OF ALGORITHMIC PROBLEM SOLVING:

[OR]

Jun VQ Explain the steps involved in design and Analysis of an algorithm with an neat diagram.

The steps that need to be followed while designing and Analysis of an algorithm are

- ✓ Understanding the Problem.
- ✓ Decision Making
- ✓ Specification of algorithm
- ✓ Algorithmic Verification
- ✓ Analysis of algorithms
- ✓ Implementation / Coding of algorithm.



[Steps in Analysis and design of an algorithm].

1. Understanding the Problem :-

- ✓ Need to Understand the Problem statement Completely.
- ✓ While understanding the Problem statement, read the Problem description

Carefully and ask questions for clarifying the doubts about the Problem.

✓ To solve Problems, there are existing algorithms. If there is no such algorithms, we have to design an algorithm on Our Own.

✓ After Carefully understanding the Problem statement, find out what are the necessary inputs for solving that Problem.

✓ The input to the algorithm is Called, instance of the Problem.

✓ The algorithm should Work Correctly for all Valid inputs.

2. Decision making:

After finding the required input set for the given Problem, we have to analyze the input and need to decide issues such as Which data structures has to be used, Whether to use Exact or approximation algorithm, Capabilities of Computational devices, and to find the algorithmic technique for solving the given Problem.

✓ Capabilities of Computational devices:

- It is necessary to know the Computational Capabilities of devices on which the algorithm will be Running.

- Algorithm is classified into

✓ Sequential algorithm

✓ Parallel algorithm.

- Sequential algorithm runs on the machine in which the instructions are executed one after another.

- Parallel algorithms are run on the machine in which the instructions are executed in parallel.

- There are certain problems which require huge amount of memory or execution time is an impact factor. For solving these problems, it is essential to have proper choice of a computational device which is space and time efficient.

✓ Choice for either exact or appropriate solving method :

- The next important decision is to decide whether the problem is to be solved exactly (or) approximately.

- If the problem needs to be solved correctly, then we need exact algorithm.

- If the problem is complex (or) not getting exact solution, then we need approximation algorithm (Eg: Travelling Sales Person Problem).

✓ Data-Structures:

- The choice of proper data structure is required before designing the actual algorithm.
- The implementation of algorithm is possible with the help of algorithm and data structures.

✓ Algorithmic strategies:

It is a general approach by which many problems can be solved algorithmically.

Algorithm techniques: (or) Design strategies:

- ✓ Brute Force - staright forward with naive approach.
- ✓ Divide and Conquer - Problem is divided into smaller instances
- ✓ Dynamic Programming
- ✓ Greedy Technique - locally optimal decisions are made.
- ✓ Branch and Bound.

③ Specification of Algorithm:

- There are various ways by which we can specify an algorithm.

(i) Using natural language.

(ii) Pseudocode

(iii) Flowchart.

(i) Natural language:-

- It is very simple to specify an algorithm using natural language.

eg:- Algorithm to Perform Addition of two numbers:

Step 1: Read the first number say 'a'.

Step 2: Read the second number say 'b'.

Step 3: Add the two numbers and store the result in 'c'.

Step 4: Display the result.

(ii) Pseudocode:

It is a Combination of natural language and Progeamming Constructs.

eg:- Addition of 2 numbers:

Algorithm Sum (a, b)

// Problem Description: Addition of 2 no's

// Input: 2 integers a, b

// Output: Addition of a & b results in 'c'.

C ← a + b

Write (c).

(iii) Flowchart:

- Geaphical representation of an algorithm is called flowchart.

(4) Algorithmic Verification:-

- Algorithimic Verification means checking Correctness of the algorithm.

- We normally check whether the algorithm gives correct output in finite amount of time for a valid set of input.
- A common method of proving the correctness of the algorithm is by using mathematical Induction.

⑤ Analysis of algorithm:

✓ While analyzing an algorithm, we should consider following factors:

- Time Complexity :- It means the amount of time taken by an algorithm to run. By computing time complexity, we come to know whether the algorithm is slow or fast.

- Space Complexity :- It means the amount of space (memory) taken by an algorithm. By computing space complexity we can analyze whether an algorithm requires more or less space.

⑥ Implementation of algorithm :-

- Implementation of an algorithm is done by progeamming languages like C, C++, Java.

Topic No: ③

Important Problem Types :

There is large number of

Computing Problems and some of them are classified.

They are:

✓ Sorting:-

- Sorting means arranging the elements in increasing or in decreasing order.
- The sorting can be done on numbers, characters, strings or employee records.
- For sorting any record, we need to choose certain piece of information based on which sorting can be done.

eg:- Quicksort, Mergesort.

✓ Searching:-

- Searching is an activity by which we can find out the desired element from the list. The element which is to be searched is called search key.
- There are many searching algorithm such as,
  - ✓ Sequential search (or) linear search
  - ✓ Binary search
  - ✓ Hashing

✓ Graph Problems:

- Graph is a collection of vertices and edges. ($G = \{V, E\}$)

- The graph Problems involve graph traversals, Shortest Path algorithm, and topological sorting.

✓ Numerical Problems :

- Numerical Problems are based on mathematical equations, System of equations, evaluating functions.

- These Problems are solved by approximate algorithms.

Topic no: (4)

Fundamentals of Analysis of Algorithms

✓ Efficiency of an algorithm can be in terms of time (or) Space.

✓ There is a systematic approach that has to be applied for analyzing any given algorithm. This Systematic approach is modelled by a framework called "Analysis Framework".

Analysis Framework

- The efficiency of an algorithm can be divided by measuring the performance of an algorithm.

- We can measure the Performance of an algorithm by

✓ Amount of time required by an algorithm to execute – Time Complexity.

✓ Amount of storage required by an algorithm – Space Complexity.

(i) Space Complexity:

✓ The space complexity can be defined as amount of memory required by an algorithm to run.

✓ To compute space complexities we use two factors:

- ✓ Constant (c)
- ✓ Instance characteristics (S_p)

✓ The space requirement S(P) can be given as

$$S(P) = C + S_p.$$

Where,

- 'C' is a constant (i.e) fixed part and it denotes the space of inputs and outputs. This space is the amount of space taken by the variables and identifiers

- 'S_p' is the space dependent upon instance characteristics.

Example:- Space Complexity.

Algorithm:- To add three numbers:

```
Algorithm Add(a, b, c)
// Problem description: Computes the addition
// of three numbers.
// Input: a, b, c are of floating type
// Output: return (a+b+c)
    return (a+b+c).
```

∴ The space requirement for algorithm is

$S(P) = C$.

(ii) Time - Complexity :

- The time Complexity of an algorithm, is the amount of time required by an algorithm to run the Completion.
- It is difficult to Compute the time Complexity in terms of Physically clocked time.
- Executing time depends on many factors such as
  - ✓ System load
  - ✓ no. of other Programs running
  - ✓ speed of underlying hardware.

∴ The time Complexity is given in terms of 'frequency Count'.

- Frequency Count is a Count denoting no. of times of execution of statement.

Example 1 : Time - Complexity :-

Algorithm :- Sum of 'n' numbers in an Array.

```
Algorithm sum ( n, a[i] )
// Problem description : Calculating sum of
// 'n' numbers in an array.
// Input : Array with list of elements.
// Output: Sum of all elements in an array.
for ( i = 0; i < n; i++)
```

```
{
    Sum = Sum + a[i];
}
```

Analysis :- Time - Complexity [Computation of Frequency Count]

Frequency Count

Statement		
(i)	$i = 0$	– 1
(ii)	$i < n$	– This statement executes for $(n+1)$ times. When Condition is true (i) when $i<n$ is true, the execution happens to be 'n' times and the statement executes one more, when $i<n$ is false
(iii)	$i++$	– 'n' times
(iv)	Sum = Sum + a[i]	– 'n' times.

$\therefore$ Total – $3n + 2$

$\therefore$ Thus, we get frequency count to be $3n+2$. The time Complexity is normally denoted in terms of (O) notation. Here if we neglect the constants, then we get the time complexity to be $O(n)$.

Example : ② :- Matrix Addition :

```
Algorithm MatrixAdd ( A[], B[], c[], n)
// Problem Description : Addition of two matrices.
// Input : Read 2 matrices 'A' and 'B'.
```

// Output : Write the resultant addition 'c'.

```
for (i=0; i<n; i++)
  {
    for (j=0; j<n; j++)
      {
        c[i][j] = A[i][j] + B[i][j]
      }
  }
```

Analysis :- Time Complexity.

	Statement		Frequency Count
(i)	$i = 0$	–	Executes Once : $Fc = 1$
(ii)	$i < n$	–	Executes for $(n+1)$ times.
(iii)	$i++$	–	Executes 'n' times.
(iv)	$j = 0$	–	Executes for $n \times (1) = n$ times (for outer loop; for initialization of 'j')
(v)	$j < n$	–	Executes for $n \times (n+1) = n^2 + n$ (Execution of Outer loop; Execution of inner 'j' loop)
(vi)	$j++$	–	$n \times n = n^2$ times (Execution of outer 'i' loop; Execution of inner 'j' loop)
(vii)	$c[i][j] = A[i][j] + B[i][j]$	–	$n \times n = n^2$ times (Execution of ; Execution of inner 'j' loop.)

∴ Hence the frequency Count is

$1 + (n+1) + n + n + (n^2+n) + n^2 + n^2$

Freq Count ⇒ $\boxed{3n^2 + 4n + 2}$

If the Constants are neglected, then the time Complexity will be $O(n^2)$

(iii) Measuring an Input Size :-

✓ If the input size is longer, the algorithm runs for a longer time. Hence we can Compute the efficiency of an algorithm as a function to which input size is Passed as a Parameter.

✓ Sometimes to implement an algorithm we require prior knowledge of input size.

For eg :- While Performing multiplication of two matrices, we should know Order of these matrices. Then only we can enter the elements of matrices ∴ Prior Knowledge of i/p size is essential.

(iv) Measuring an Running time :-

✓ Already, the time Complexity is measured in terms of a Unit Called frequency Count.

✓ The time in which is measured for analyzing an algorithm is generally

Running time.

- From an Algorithm :-

✓ We first identify the important operation of an algorithm. This operation is called basic operation.

✓ It is not difficult to identify basic operation from an algorithm. The operation which is more time Consuming is a basic operation.

✓ Such basic operation is normally located in inner loop.

Eg:-	Problem Statement	I/p Size	Basic operation
(i)	Performing matrix multiplication	list of n elements with the order n x n	Actual multiplication
(ii)	Computing gcd	Two no's	Division

- Then we Compute total number of time taken by this basic operation. We Can Compute the running time of basic operation by following formula.

$$T(n) \approx C_{op} \cdot C(n)$$

- $T(n)$: Running time of basic operation
- C_{op} : time taken by the basic operation to execute
- $C(n)$: No. of times the operation needs to be executed.

(iv) Order of Growth :-

- Measuring the Performance of an algorithm in relation with the input size n. It is called order of growth.

- For eg: The order of growth for varying input size of n is given below:

n	$\log n$	$n \log n$	n^2	2^n
1	0	0	1	2
2	1	2	4	4
4	2	8	16	16
8	3	24	64	256
16	4	64	256	65,536
32	5	160	1024	4294967296

- From the above table, the logarithmic function is the slowest growing function and the exponential function (2^n) is the fastest one.

Topic No: 5
VA
8m

Asymptotic Notations and it's Properties.

- To choose the best algorithm we need to check efficiency of each algorithm.

- Asymptotic Notations is a shorthand way to represent the time Complexity.

- Using Asymptotic Notations, we can give time complexity as
  - ✓ Latest Possible (or) "Best-Case"
  - ✓ Slowest Possible (or) "Worst-Case"
  - ✓ Average possible (or) "Average Case".

Various Notations used :-

1. Big-oh Notation (O)
2. Omega Notation (Ω)
3. Theta Notation (Θ)

1. Big-oh Notation : (O) :-

- It is denoted by 'O'.
- It is a method of representing the upper bound of algorithm's running time.
- Using big-oh notations, we can give longest amount of time taken by the algorithm to complete.

Defintion :

Let F(n) and g(n) be two non-negative functions. Let 'no' and Constant 'c' are 2 integers, such that 'no' denotes some value of input and $n > no$. Similarly 'c' is a constant such that $c > 0$,

We Write,

$$F(n) \leq C * g(n),$$

then F(n) is big-oh of g(n). It is also denoted as $F(n) \in O(g(n))$.

Example :- Consider the function $F(n) = 2n+2$ and $g(n) = n^2$. 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.

$n=1$:-

$$F(n) = 2n+2$$
$$= 2(1)+2 \quad (n=1)$$
$$\boxed{F(n) = 4}$$

and $g(n) = n^2$
$$= (1)^2$$
$$\boxed{g(n) = 1}$$

(ie) $F(n) > g(n)$

$n=2$:

$$F(n) = 2n+2$$
$$= 2(2)+2$$
$$\boxed{F(n) = 6}$$

and $g(n) = n^2$
$$= (2)^2$$
$$\boxed{g(n) = 4}$$

(ie) $F(n) > g(n)$

$n=3$:

$$F(n) = 2n+2$$
$$= 2(3)+2$$
$$\boxed{F(n) = 8}$$

and $g(n) = (3)^2$
$$\boxed{g(n) = 9}$$

(ie) $F(n) < g(n)$ is True

∴ Here we calculate that for $n > 2$, we obtain $F(n) < g(n)$.

② . Omega Notation : (Ω) :

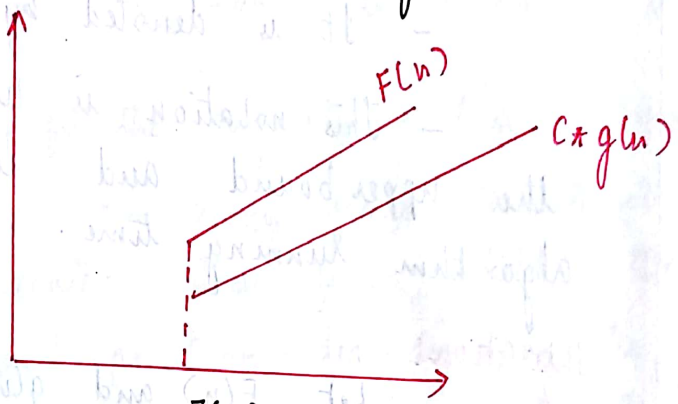
- It is denoted by Ω.
- This notation is used to represent the lower bound of algorithm's running time.

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

$$F(n) \geq C * g(n)$$

- It is denoted by $F(n) \in \Omega(g(n))$.



$F(n) \in \Omega(g(n))$.

Example :

Consider $F(n) = 2n^2 + 5$ and $g(n) = 7n$.

Then if $n = 0$:

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

$$\boxed{F(n) = 5}$$

and $g(n) = 7(0)$

$$\boxed{g(n) = 0}$$

(ie) $F(n) > g(n)$.

if $n = 1$:

$$F(n) = 2(1)^2 + 5$$

$$\boxed{F(n) = 7}$$

and $g(n) = 7(1)$ (ie) $F(n) = g(n)$

$$\boxed{g(n) = 7}$$

if $n = 3$:

$F(n) = 2(3)^2 + 5$

$\boxed{F(n) = 23}$

and $g(n) = 7n$

$= 7(3)$

$\boxed{g(n) = 21}$

(ie) $F(n) > g(n)$

∴ Hence for $n > 3$, we get $F(n) > C * g(n)$

③ Theta Notation: (Θ)

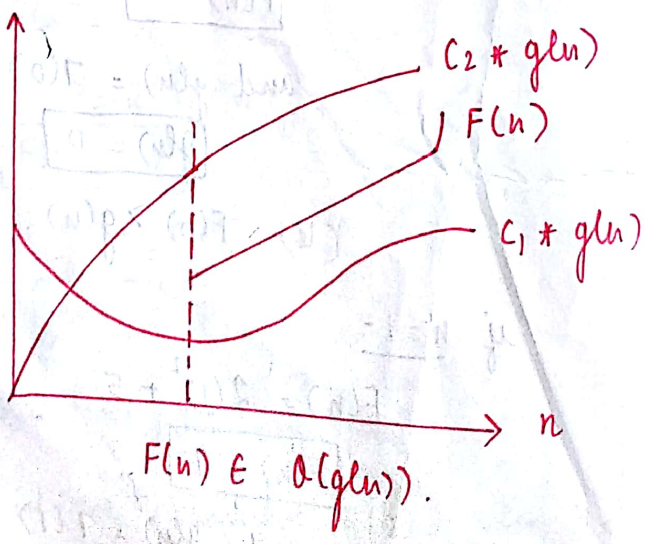
- It is denoted by Θ
- This notation is used to represent the upper bound and lower bound of algorithm running time.

Definition:

Let $F(n)$ and $g(n)$ be two non negative functions. There are two Positive Constants namely c_1 and c_2, such that

$$c_1 * g(n) \leq F(n) \leq c_2 * g(n)$$

- It is denoted as $F(n) \in \Theta(g(n))$



Example:

$f(n) = 2n+8$ and $g(n) = 7n$ (Where $n \geq 2$)

(ie) $5n < 2n+8 \leq 7n$ for $n \geq 2$

Here $c_1 = 5$

$c_2 = 7$

and $\boxed{n_0 = 2}$.

13m

Topic No: (6) **Mathematical Analysis for Recursive Algorithms:**

(X)(X)(X) VIP

- **General Plan / Procedure for Analyzing efficiency of Recursive algorithm:-**

① Decide the input size based on Parameter 'n'.

② Identify the algorithm's basic operation.

③ Check how many times the basic operation is executed. Then find whether the execution of basic operation depends upon the input size 'n'. Determine worst, average and best cases for input of size 'n'.

If the basic operation depends upon worst case, average case and best case, then that has to be analyzed separately.

④ Set up the recurrence relation with some initial condition and expressing the basic operation.

⑤ Solve the recurrence relation using either forward / backward substitution method, and then Correctness of formula can be Proved with the help of mathematical Induction method.

Example 1: - Explain the recursive algorithm for Computing Factorial and analyze its

Problem statement: Computing factorial of a number using recursion.

Algorithm:

Algorithm factorial(n)

// Problem description: This algorithm computes n! using recursive function

// Input: A non-negative integer n.

// output: Returns the factorial value

if (n=0)

return 1

else

return factorial (n-1) * n .

Example:

The factorial of a number can be obtained by performing repeated multiplication.

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: 0! * 1 * 2 * 3 * 4 * 5 (0! = 1)

Step 7: 1 * 1 * 2 * 3 * 4 * 5

Output: n! = 5! = 120

Mathematical Analysis:-

Step 1: The factorial algorithm works for input size 'n'.

Step 2: The basic operation in computing the factorial is multiplication.

Step 3: The recursive function call can be formulated as

$$F(n) = F(n-1) * n \quad \text{Where } n > 0$$

Then the basic operation multiplication is given as M(n). And M(n) is multiplication count to compute factorial (n).

$$M(n) = M(n-1) + 1$$

Where,

M(n-1) → These multiplications are required to compute factorial (n-1)

1 → To multiply factorial (n-1) by 'n'.

Step 4: In step 3, the recurrence relation is obtained.

$$M(n) = M(n-1) + 1$$

Forward Substitution:

n = 1:

$$M(1) = M(0) + 1$$

$$M(1) = 0 + 1 = 1$$

n = 2:

$$M(2) = M(1) + 1$$

$$M(2) = 1 + 1 = 2$$

$n = 3$:

$M(3) = M(2) + 1$

$M(3) = 2 + 1 = 3$

Backward Substitution:-

$M(n) = M(n-1) + 1$ ——— (1)

$M(n-1) = M(n-1-1) + 1$

$M(n-1) = M(n-2) + 1$ ——— (2)

Sub (2) in (1)

$M(n) = [M(n-2) + 1] + 1$

$M(n) = M(n-2) + 2$ ——— (3)

$M(n-2) = M(n-2-1) + 1$

$M(n-2) = M(n-3) + 1$ ——— (4)

Sub (4) in (3)

$M(n) = [M(n-3) + 1] + 1 + 1$

$M(n) = M(n-3) + 3$

By the substitution method, we can establish a formula:

$M(n) = M(n-i) + i$

∴ Thus the time complexity of factorial function is $O(n)$

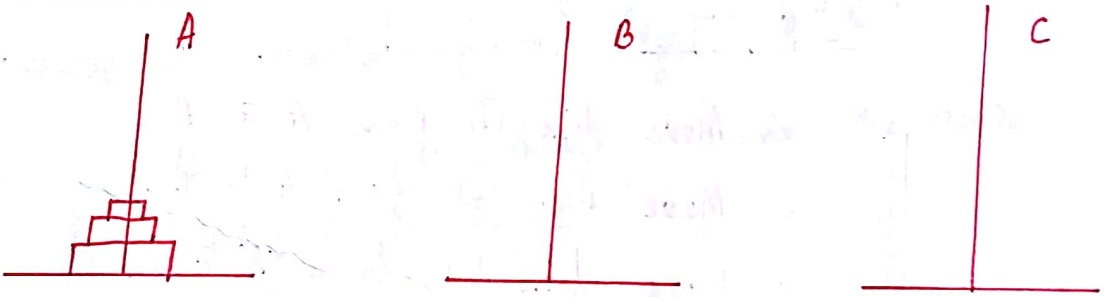
Problem: 2 - Tower of Hanoi:

Algorithm:

Algorithm TOH (A, C, B, n)

```
{
// If only one disk has to be moved.
  If (n=1) then
  {
    Write ("The Peg moved from A to C")
    Return
  }
else
// move top (n-1) disks from A to B using C.
  {
    TOH (n-1, A, B, C);
    // move remaining disk from B to C using A.
    TOH (n-1, B, C, A);
  }
}
```

Problem :-



✓ There are three Pegs named as A, B and C. The three disks of different diameters are Placed on Peg A. The arrangement of the disks is such that every smaller disk is Placed on the larger disk.

✓ The Problem of "Tower of Hanoi" states that move the three disks from Peg 'A' to Peg 'C'.

using Peg 'B' as Auxillary.

Conditions:-

(i) only the top disk on any peg may be moved to any other Peg.

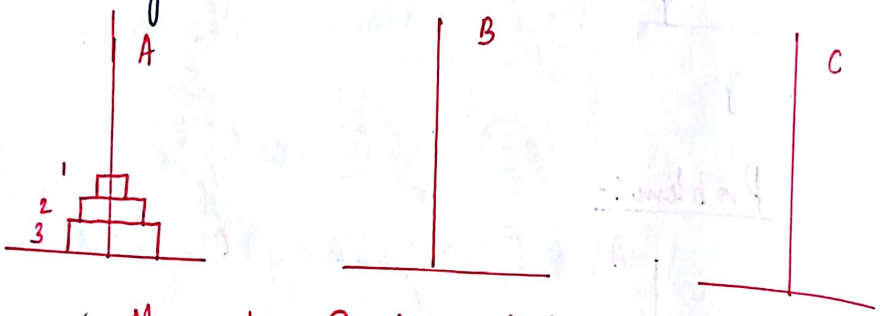
(ii) A larger disk should never rest on the Smaller one.

Solution:

① Move top (n-1) disk from A to B using 'C' as auxiliary.

② Move the remaining disk from A to C

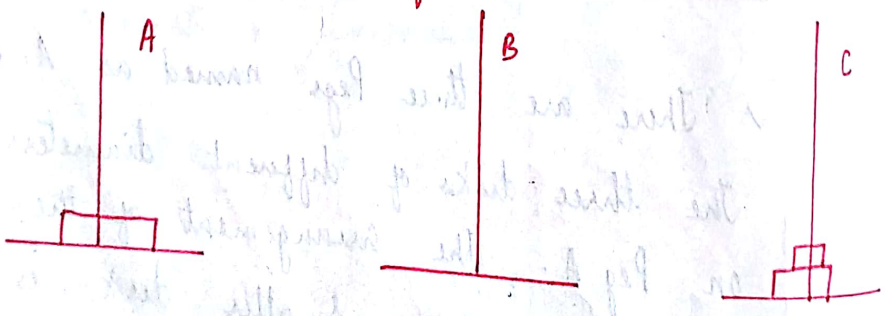
③ Move the (n-1) disks from B to C using A as auxillary.



✓ Move disk ① from A to B

✓ Move disk ② from A to C.

✓ Move disk ① from B to C.

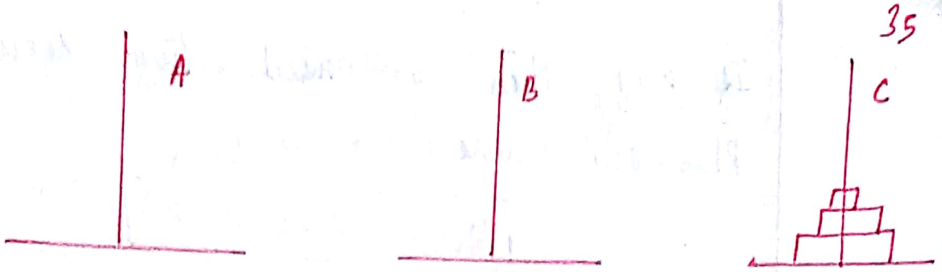


✓ Move disk ③ from A to B

✓ Move disk ① from C to A

✓ Move disk ② from C to B

✓ Move disk ① from A to B



∴ Thus Actually we have moved n-1 disks from Peg A to Peg C.

Mathematical Analysis :-

Step 1 :- The input size is 'n' (i.e) total number of disks.

Step 2 : The basic operation in this Problem is moving disk from One Peg to another. When n > 1, then to move these disks from Peg A to Peg C using Peg B. We first move Recursively (n-1) disks from Peg A to Peg B using Peg C. Then we move the largest disk directly from Peg A to C' and finally move n-1 disks from Peg B to C'.

✓ If n = 1, then simply move the disk from Peg A to Peg C.

Step 3 : The moves of disks are denoted by M(n). M(n) depends on number of disks 'n'

∴ M(1) = 1 ∴ only 1 move is needed

TOH (1, ·, ·, ·,)

If n > 1, then we need two Recursive Calls Plus One move. → To move largest disk from Peg A to C.

$$M(n) = M(n-1) + 1 + M(n-1)$$

To move (n-1) disk from Peg A to B ↗ To move (n-1) disk from Peg B to C ↖

If $n>1$, then we need two recursive calls Plus one more.

$$\therefore \quad \boxed{M(n) = 2M(n-1) + 1}$$

Step 4:-

Solving recurrence equation $M(n) = 2M(n-1)+1$ using substitution methods.

Forward substitution:-

For $n > 1$

(ie) $n=2$ $\quad M(2) = 2M(1) + 1$

$= 2(1) + 1$

$$\boxed{M(2) = 3}$$

$n=3$ $\quad M(3) = 2M(2) + 1$

$= 2(3) + 1$

$$\boxed{M(3) = 7}$$

$n=4$: $\quad M(4) = 2M(3) + 1$

$= 2(7) + 1$

$$\boxed{M(4) = 15}$$

Backward substitution:

$$M(n) = 2M(n-1) + 1 \quad \text{---} \quad (1)$$

$n = n-1$: $\quad M(n-1) = 2M(n-2) + 1 \quad \text{---} \quad (2)$

From (1), sub (2) in (1)

$$M(n) = 2[2M(n-2) + 1] + 1$$

$$\boxed{M(n) = 4M(n-2) + 3} \quad \text{---} \quad (3)$$

∴ This is also written as

$2^2 M(n-2) + 2 + 1$ ——— (4)

$n = n-2$:

$M(n-2) = 2M(n-3) + 1$ ——— (5)

Sub (5) in (3)

$M(n) = 4M(n-2) + 3$

$= 4(2M(n-3)+1) + 3$

$= 8M(n-3) + 7$

∴ This is also written as

$2^3 M(n-3) + 2^2 + 2^1 + 1$

∴ From this, we can establish a formula as $M(n) = 2^i M(n-1) + 2^{i-1} + 2^{i-2} + \ldots 2 + 1$

∴ This can also be written as

$M(n) = 2^i M(n-i) + 2^i - 1$

From this, we can conclude that the tower of hanoi, has the time complexity of $O(2^n)$.

Topic no.(7) **Mathematical Analysis of Non-Recursive algorithms:**

General Plan for Analyzing Non-recursive algorithms

① Decide the input size or Parameter 'n'.

② Identify algorithm's basic operation.

③ Check how many times the basic operation is executed. Then find whether the execution of basic operation depends on the input

size 'n'. Determine the worst, average and best cases for input of size 'n'. If the basic operation depends upon average, worst and best case, then that has to be analyzed separately.

④ Set up a sum for the no. of times the basic operation is executed.

⑤ Simplify the sum using standard formula and rules.

Examples:-

Problem 1: Finding maximum element in an given Array.

Algorithm:-

```
Algorithm Max_element ( A[0...n-1])
// Problem Description: This algorithm is
// for finding max. element in array
// Input: Array A[0...n-1]
// Output: returns the largest elemen

    Max ← A[0]
    for i ← 1 to n-1 do
    {
      If ( A[i] > max) then
        max ← A[i]
    }
    return max;
```

Mathematical Analysis:

Step 1: The input size is 'n' (ie) the total no. 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. As the Comparision made for each value of 'n' there is no need to find best, Worst and Average Case.

Step 4:- Let C(n) be the no. of times, Comparision is executed. The algorithm makes Comparision each time the loop executes. Here for $i = 1$ to $n-1$ times, the Comparision is made. Therefore,

$$C(n) = \sum_{i=1}^{n-1} 1$$

→ One Comparision made for each Value of i.

Step 5: Simplify:-

$$C(n) = \sum_{i=1}^{n-1} 1$$

$$C(n) = n-1 \in O(n)$$

∴ Thus the time Complexity of finding maximum element in an Array is $O(n)$.

Problem: ②:- Obtaining matrix multiplication of a given two matrices:

Algorithm:

Algorithm Matrix_mul (A[0..n-1][0..n-1], B[0..n-1][0..n-1])

// Problem description: This algorithm Performs multiplication of 2 matrices.

// Input: Two matrices A and B.

// Output: 'C' matrix containing multiplication of A and B.

```
for i ← 0 to n-1 do
    for j ← 0 to n-1 do
        C[i,j] ← 0
        for k ← 0 to n-1 do
            C[i,j] = C[i,j] + A[i,K] * B[k,j]
return C;
```

Mathematical Analysis:

Step 1: The input size of above algorithm is simply the order of matrices 'n'.

Step 2: The basic operation is the inner most loop and which is

C[i,j] = C[i,j] + A[i,k] * B[k,j]

In this basic operation, both addition and multiplication are Performed But we will not choose any one of them as basic operation because on each repetition of loop. Each of the two will be executed exactly once. so by counting one automatically. Other will be counted. Here we Consider multiplication as a basic operation.

Step 3: The basic operation depends only upon input size. There are no best, worst and average case.

Step 4: Simplify

The sum can be denoted by M(n).

M(n) = Outermost loop × Inner loop × Innermost loop

= [For loop 'i'] × [For loop j] × [For loop k']

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

$$= \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n$$

$$= \sum_{i=0}^{n-1} n^2$$

$$M(n) = n^3.$$

∴ Thus the time Complexity of matrix multiplication is $O(n^3)$.

Problem ③: Linear Search: [Example for best Case, Worst Case and Average Case].

Searching algorithm - used to search an element sequentially (one by one) from a given array.

Problem:

A:

0	1	2	3	4
7	3	1	2	17

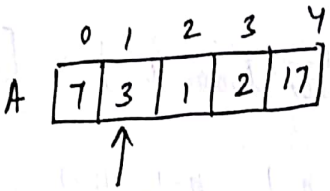
↑

Step 1: Check whether the search element is same as A[0], if it is not same increment the position of an array. Now array points to A[1].

Here x = 17 and 17 ≠ 7

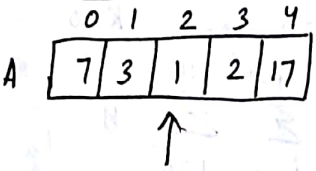
x → search element

Step 2:



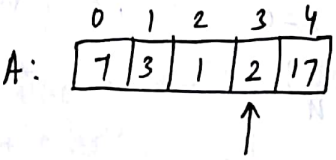
17 ≠ 3

Step 3:



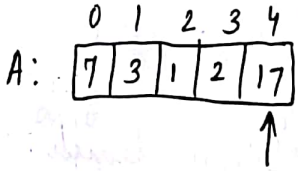
17 ≠ 1

Step 4:



17 ≠ 2

Step 5:



17 = 17

∴ The element is (17) found in 4th Position:

Algorithm:

Algorithm linear search (x, A[0...n-1])

// Problem description: This is an algorithm which searches the element linearly

// Input: Array A[0...n-1], search element x.
// Output: returns the Position of an array element for the Key searched.

```
for i ← 0 to n-1 do
    if (A[i] == x) then
        return i
    endif
end.
```

Mathematical Analysis:

Best Case:

The element which is to be searched, if it searches for a first element, then the time Complexity is O(n).

Average Case:

Let 'p' be the Probability of getting Successful search, 'n' is the total no. of elements in the list.

- The first match of the element will Occur at i^{th} Position Probability of Occuring first match is P/n.

- Let $q = 1 - P$ " Probability of getting Unsuccessful search.

Probability of Successful search (P) + Probability of Unsuccessful search (q)

$$Cavg(n) = [1 \cdot P/n + 2 \cdot P/n + \cdots i \cdot P/n] + n(1-P)$$

$$= P/n\,(1+2+\dots i) + n(1-P)$$

$$= P/n\left[\frac{n(n+1)}{2}\right] + n(1-P)$$

$$C_{avg}(n) = \frac{P(n+1)}{2} + n(1-P)$$

P=0:

$$C_{avg}(n) = \frac{0(n+1)}{2} + n(1-0)$$

$$\boxed{C_{avg}(n) = n}$$

P=2:

$$C_{avg}(n) = \frac{1(n+1)}{2} + n(1-1)$$

$$\boxed{C_{avg}(n) = \frac{(n+1)}{2}}$$

∴ Best Case : $O(1)$

Worst Case : $O(n)$

Average Case: $O(n)$

Worst Case :-

If the element is searched is the last element in the array, then n elements are searched to find the search element ∴ The time complexity is $O(n)$.