Introduction to Algorithms

An algorithm is a sequence of unambigous instructions to solve a problem to obtain a output for a legitemate input in a finite amount of Time

An algorithm must be independent of any Programming language.

=> Algorithm to find GCD of two positive integers :-

① GCD(m,n)

// Modulo Division Method

// Input : 2 Positive integers

// Output : GCD of 2 Positive integers

```
While n ≠0
{
    count ← count +1
    rem ← m mod n
    m ← n
    n ← rem
}
GCD ← m
```

② // Subtraction Method

// Input : 2 Positive Integers

// Output : GCD of 2 Positive Integers

```
While m ≠ n
{
    count ← count + 1
    if (m > n)
        m ← m - n
    else
        n ← n - m
}
```

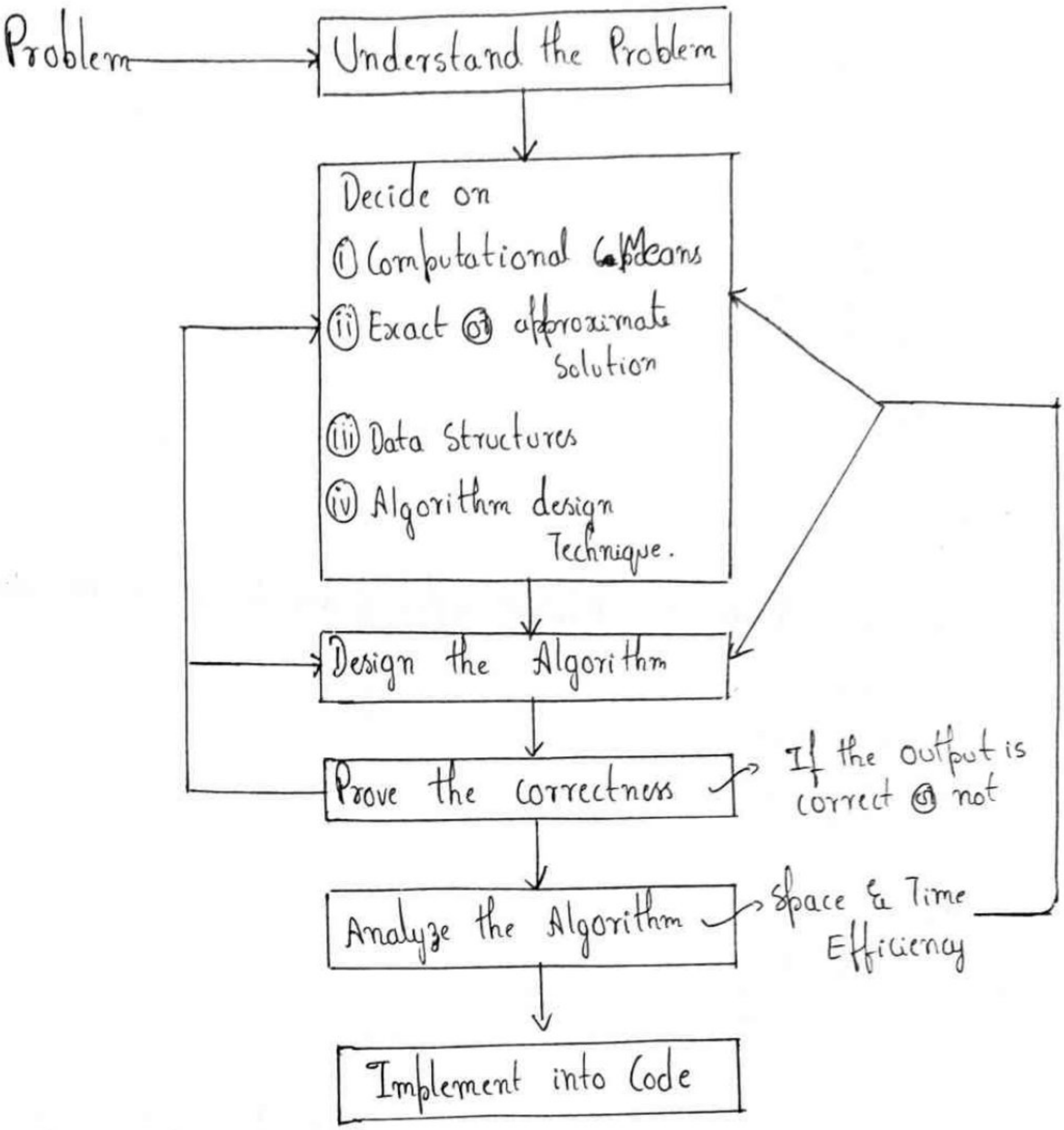
③ // Consecutive Integer Method

// Input : 2 Positive Integer

// Output : GCD of 2 Positive Integers

```
t ← min (m, n)
While t ≠ 1
{
    if (m % t == 0)
        break;
    t ← t - 1
}
gcd ← t
```

-> Steps involved in writing an Algorithm :-



=> Fundamental Problems :-

-> Sorting
-> Searching
-> String
-> Graphs
-> Combinatorial
-> Numerical

① Sorting :-

Stable :- Preserves relative order of any two equal elements in its input

* Bubble
* Binary Search
* Merge Sort

Unstabe :-

* Heap Sort
* Selection Sort
* Quick Sort

Inplace :- An algorithm is said to be Inplace if it does not require extra memory space

Insertion } Inplace
Selection }

Heap } Not Inplace
Merge }

② → Searching :-

It deals with finding a given value called Search Key

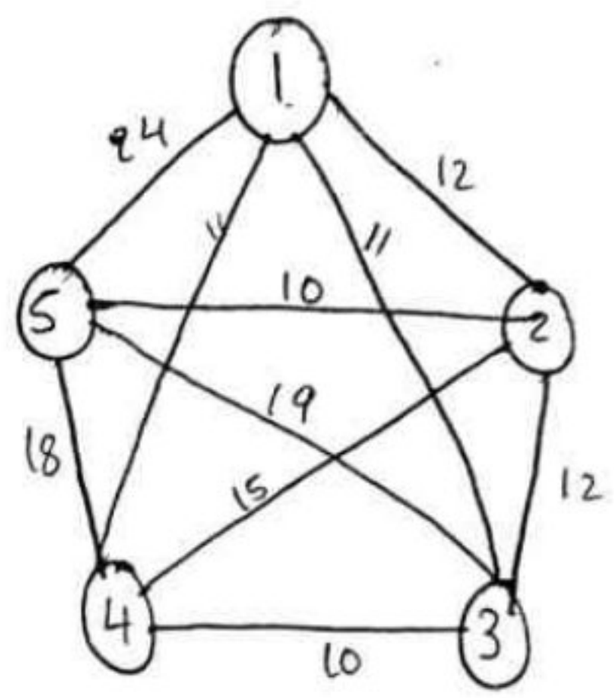
Examples of ~~Linear~~ Searching :-

(i) Linear Search | Sequential Search

(ii) Binary Search (Only for Sorted Arrays).

Graph Problems :-

Ex :- Graph Coloring Problem, Travelling Salesman Problem



In this problem, we need to find the minimum expense for Salesman to travel all 5 cities.

$1 \xrightarrow{12} 2 \xrightarrow{10} 5 \xrightarrow{18} 4 \xrightarrow{10} 3 \xrightarrow{11} 1$

Total = 61 (Minimum)

String Processing :-

Deals with non-numeric Data

(i) String Searching
(ii) String Matching
} Examples.

⇒ Analysis of Time Efficiency :-

①
```
fun(int n)
{
    for (i=1 ; i<=n ; i++)        // ① , n+1 , n
        Printf ("Hello") ;        // → n
}
```

$$T(n) = t \cdot c(n)$$

Here, $c(n) = 1 + n + 1 + n + n = 3n + 2$

②
```
fun (int n)
{
    for (i=1 ; i<=n ; i+=2)       // ① , (n+1)/2 , n/2
        Printf ("Hello");         // → n/2
}
```

$$T(n) = t \cdot c(n)$$

$$c(n) = 1 + \frac{n}{2} + 1 + \frac{n}{2} + \frac{n}{2} = \frac{3}{2}n + 2$$

- For each of following fun determine how many time the function will change if it is incremented by 4.

(i) $T(n) = \log_2 n$

$$\frac{T(4n)}{T(n)} = \frac{\log_2 4n}{\log_2 n} = \frac{\log 4n}{\log n} = \frac{\log 4}{\log n} + \frac{\log n}{\log n} = \frac{2}{\log n} + 1$$

(ii) $T(n) = \sqrt{n}$

$$\frac{T(4n)}{T(n)} = \frac{\sqrt{4n}}{\sqrt{n}} = 2$$

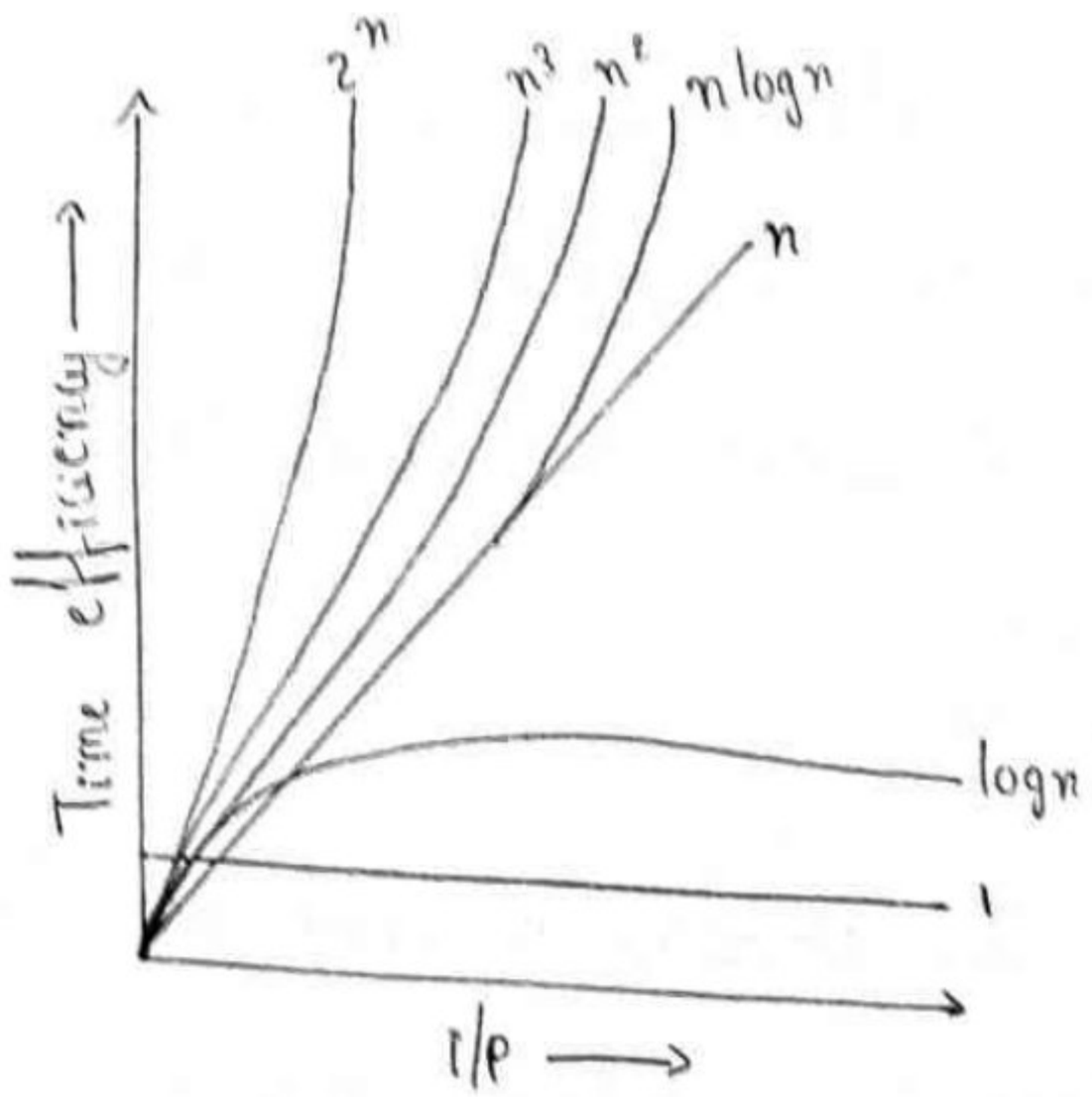
(iii) $T(n) = n$

$$\frac{T(4n)}{T(n)} = \frac{4n}{n} = 4$$

=> Order of growth :-

		Linear		Quadratic	Cubic		Exp	Fact
I/P (n)	log n	n	n log n	n^2	n^3	. . .	2^n	n!
10	3.3	10	3.3 x 10	10^2	10^3	. . . .	2^{10}	10!

=> Analysis of different Algorithm :-



Ex:-

```
SequentialSearch(A[0, ..... n-1], K)
// I/P :- An array A[0 .....n-1] and Search Key K
// O/P :- Returns position of first occurence of Key, else returns -1
                                                    if no match found.
    i ← 0
    while i<n and A[i] ≠ K
        i ← i+1
    if i<n return i
    Else return -1
```

→ Worst Case :- The situation in which an algorithm takes maximum time to complete its execution.

$C_{worst}(n) = n$ [Linear Search]

Basic operation (Comparison)

→ Best Case :- The situation in which an algorithm takes minimum time to complete its execution.

(or)

The minimum work done by an Algorithm

$C_{best}(n) = 1$

→ Average Case :-

Assumptions :-

(i) The probability of successful search = p where $0 \le p \le 1$

(ii) The probability of Key occuring in position 'i' is same for all i's

$$P(i) = \frac{p}{n}$$

$$C_{Average}(n) = \left[1 \cdot p/n + 2 \cdot p/n + \ldots + i \cdot p/n + \ldots n\,p/n\right] + n(1-p)$$

$$= \frac{p}{n}\left(1+2+3+\ldots+n\right) + n(1-p)$$

$$= \frac{p}{n} \times \frac{n(n+1)}{2} + n(1-p)$$

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

When search is successful, $p = 1$

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

When Seach is unsuccessful, $p = 0$

$$C_{Average}(n) = n$$

→ For each of following Algorithm, indicate the following :-

(i) Input Size / Natural Size

(ii) Basic operation

(iii) Wether the basic operation count can be different for input of same size.

(1) $1+2+3+\ldots+n$ (Computing sum of n numbers).

a) Input size = n

b) Basic operation = Addition

c) The operation count will not change

(2) Computing $n!$ $(1\times2\times3\times4\times\ldots\times n)$

a) Input Size = $\log n+1$

b) Basic operation = Multiplication

c) The operation count will change

⇒ Asymptotic Notation & Basic Efficiency Class :-

① O (big oh)

② Ω (big Omega)

③ Θ (big theta)

Consider two non-negative functions $t(n)$ and $g(n)$ over a set of natural numbers.

⇒ Informally, $O(g(n))$ is a set of all function with a same (or) smaller order of growth as $g(n)$ to within constant multiple upto infinity

Ex :- $n \in O(n^2)$

$n^4 \notin O(n^2)$

$100n + 5 \in O(n^2)$

$\frac{n(n-1)}{2} \in O(n^2)$

$0.00001n^3 \notin O(n^2)$

⇒ Informally, $\Omega(g(n))$ is a set of all function with larger (or) same order of growth as $g(n)$ to within constant multiple upto infinity.

$$n^3 \in \Omega(n^2)$$

$$10n + 5 \notin \Omega(n^2)$$

- $\Theta(g(n))$ is a set of all the function whose order of growth is same as that of $g(n)$ to a constant multiple upto infinity.

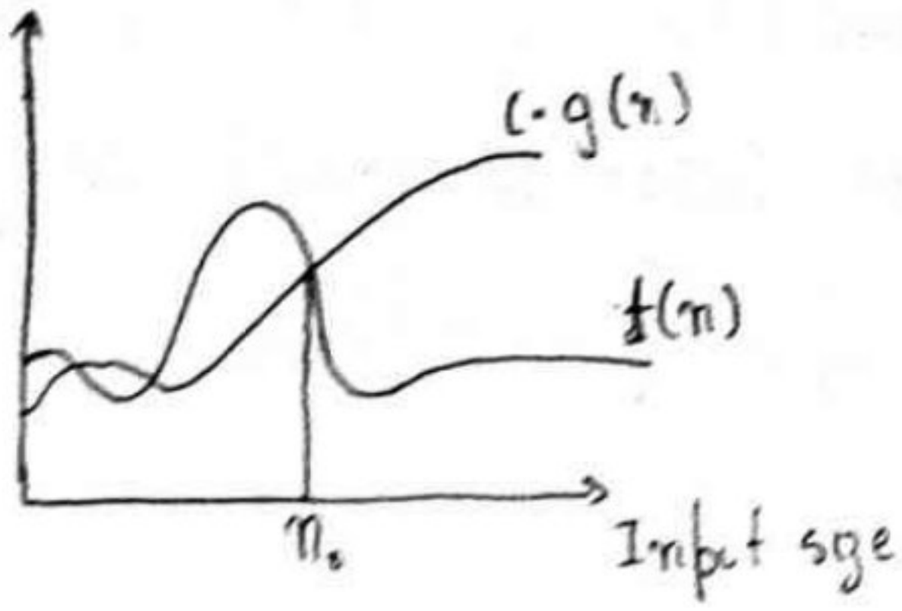
$$5n^2+6 \in \Theta(n^2)$$

⇒ Formal definitions :-

$O(g(n))$:- It is used to give a upper bound ~~to $f(n)$~~ for a function $g(n)$ within a constant multiple.

A function $f(n)$ is said to in $O(g(n))$ denoted as $f(n) \in O(g(n))$ if $f(n)$ is bounded above by some constant multiple of $g(n)$ for all large 'n' i.e there exists some constant 'c', n_0 which is non negative.

$$f(n) \leq c \cdot g(n) \quad \text{for some} \quad \begin{matrix} c > 0 \\ n \geq n_0 \end{matrix}$$



① Consider, $t(n) = 3n+2$ and $g(n) = n$. Prove that $t(n) \in O(g(n))$

To Prove, $t(n) \leq c \cdot g(n)$

$$3n+2 \leq c \cdot n$$

$$\therefore c \geq 4$$

$$\Rightarrow 3n+2 \leq 4n$$

$$n \geq 2$$

n	$t(n)$ $3n+2$	$g(n)$ $c \cdot n$
1	5	4 ✗
2	8	8 ✓
3	11	12 ✓

$$\therefore t(n) \in O(g(n))$$

② Prove that, $3n^2+4n-2 \in O(n^2)$ for which values of c and n_0.

$$3n^2+4n-2 \leq c \cdot n^2$$

Let, $c \geq 4$

$$3n^2+4n-2 \leq 4n^2$$

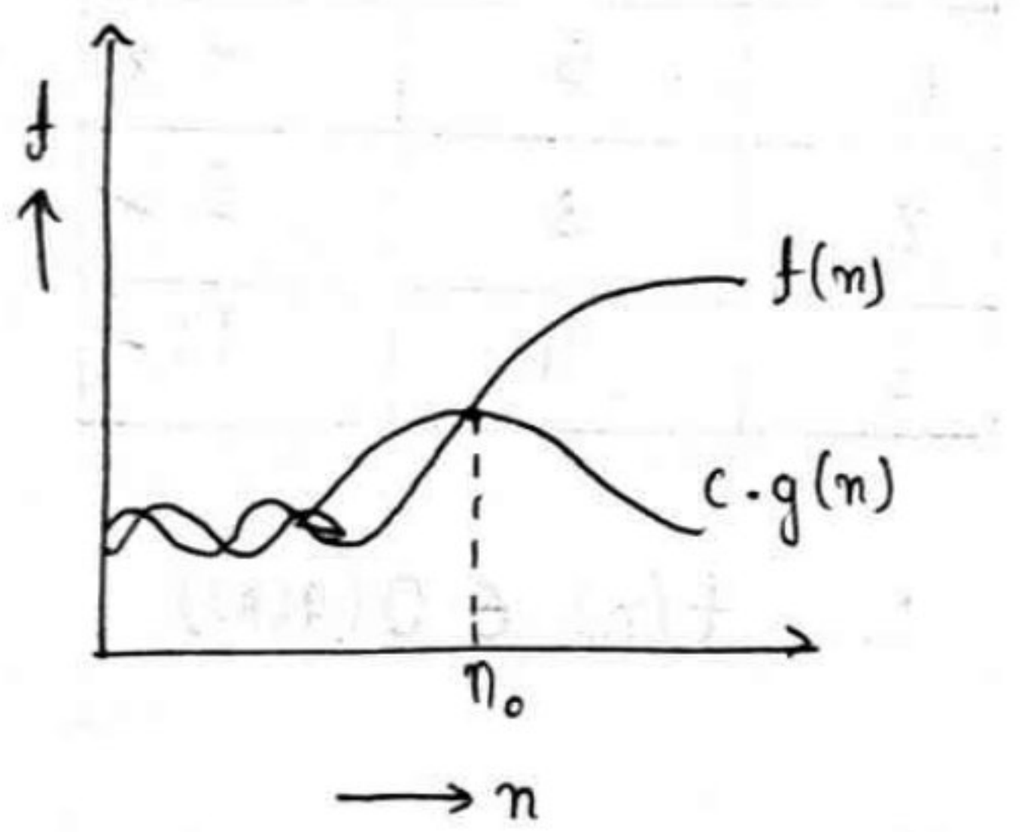
	$t(n)$ $3n^2+4n-2$	$c \cdot g(n)$ $4n^2$
1	5	4
2	18	16
3	37	36
4	62	64 ✓

$$\therefore n_0 = 4$$

=> Omega :-

A function $t(n)$ is said to be in $\Omega(g(n))$, denoted $t(n) \in \Omega(g(n))$ if $t(n)$ is bounded below by some positive constant multiple of $g(n)$ for all large n i.e if there exist some positive constant 'c' and some non-negative integer n_0 such that

$$t(n) \geq c \cdot g(n) \quad \text{for all } n \geq n_0$$



① Consider $t(n) = n^2+n$ and $g(n) = 5n^2$. Prove that $t(n) \in \Omega(g(n))$

To Prove :- $t(n) \geq c \cdot g(n)$

$$n^2+n \geq c \cdot 5n^2$$

$$\Rightarrow c = \frac{1}{5}$$

$$n^2+n \geq n^2$$

$$\therefore n \geq 0$$

$$\Rightarrow n_0 = 0$$

② Consider $f(n) = n^2$, $g(n) = n^2+n$. P T $f(n) \in \Omega(g(n))$

To Prove :- $f(n) \geq c \cdot g(n)$

$$n^2 \geq c(n^2+n)$$

Let, $c = \frac{1}{2}$

$$\Rightarrow n^2 \geq \frac{1}{2}(n^2+n)$$

$$\frac{n^2}{2} + \frac{n^2}{2} \geq \frac{n^2}{2} + \frac{n}{2}$$

$$n^2 \geq n$$

$\therefore n_0 \geq 0$ & $c = \frac{1}{2}$

③ Theta :-

A function $f(n)$ is said to be in $\Theta(g(n))$, denoted $f(n) \in \Theta(g(n))$ if $f(n)$ is bounded both above and below by some positive constant multiple of $g(n)$ for all large n i.e if there exists some positive constants c_1 and c_2 and some non negative integer n_0, such that

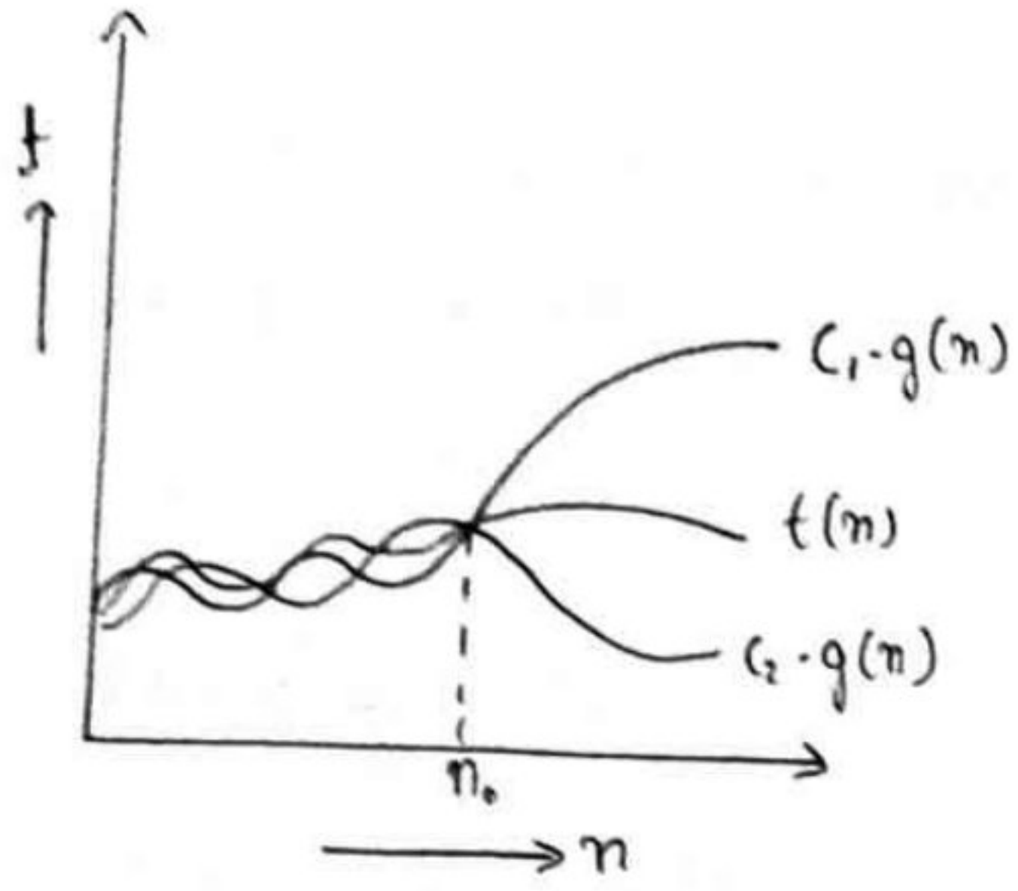
$$c_2 g(n) \leq f(n) \leq c_1 g(n) \text{ for all } n \geq n_0$$

Condition 1 :- $f(n) \in O(g(n))$

$$f(n) \leq c_1 g(n)$$

Condition 2 :- $f(n) \in \Omega(g(n))$

$$f(n) \geq c_2 g(n)$$



① Consider, $t(n) = 3n+2$ and $g(n) = n$, P.T $t(n) \in \Theta(g(n))$

To Prove :- $t(n) \leq C_1 \cdot g(n)$ & $t(n) \geq C_2 \cdot g(n)$

$3n+2 \leq C_1 n$ & $3n+2 \geq C_2 \cdot n$

$C_1 \Rightarrow 4$ $C_2 \Rightarrow 1$

$3n+2 \leq 4n$ $3n+2 \geq 2n$

n	$t(n)$ $3n+2$	$C_1 \cdot g(n)$ $4n$	$C_2 \cdot g(n)$ $2n$
1	5	4	2 ✗
2	8	8	4 ✓
3	11	12	6

$\therefore$ $n_0 = 2$

⇒ Theorem :- If $f_1(n) \in O(g_1(n))$ and $f_2(n) \in O(g_2(n))$

$$f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$$

Ex :- Consider an algorithm which includes both sorting and binary searching.

Let, For Sorting, $g_1(n) = n^2$

For Searching, $g_2(n) = \log n$

$\therefore$ $f_1(n) + f_2(n) \in O(n^2)$

Proof :-

Algebraic Properties used :-

$$\left.\begin{matrix} a_1 > b_1 \\ a_2 > b_2 \end{matrix}\right\} a_1 + a_2 \geq 2 \cdot \text{Max}\{b_1, b_2\} \longrightarrow (i)$$

WKT, By Formal definition of $O(g(n))$

$$f_1(n) \leq C_1 \cdot g_1(n)$$

$$f_2(n) \leq C_2 \cdot g_2(n)$$

$$f_1(n) + f_2(n) \leq C_1 g_1(n) + C_2 \cdot g_2(n)$$

Let, $C_3 = \text{Max}(C_1, C_2)$

Replace C_1 & C_2 by C_3

$$f_1(n) + f_2(n) \leq C_3 \cdot g_1(n) + C_3 \cdot g_2(n)$$

$$\leq C_3\{g_1(n) + g_2(n)\} \longrightarrow (ii)$$

From eqn ① & ②

$$t_1(n) + t_2(n) \leq C \cdot Max\{g_1(n), g_2(n)\}$$

$$t_1(n) + t_2(n) \in O(Max\{g_1(n), g_2(n)\})$$

Where, $C = 2C_3$ and $C_3 = \{Max(C_1, C_2)\}$

and $n_0 = Max\{n_1, n_2\}$

⇒ Using Limits for comparing order of growth :-

$$\lim_{n \to \infty} \frac{t(n)}{g(n)} = \begin{cases} C \end{cases}$$

(i) If $C = 0$, $t(n) < g(n)$ → O

(ii) If $C > 0$, $t(n) = g(n)$ → Θ

(iii) If $C = \infty$, $t(n) > g(n)$ → Ω

① Compare order of growth of $\frac{1}{2}n(n-1)$ and n^2.

$$C = \lim_{n\to\infty} \frac{t(n)}{g(n)}$$

$$= \lim_{n\to\infty} \frac{1}{2}\frac{n^2(n-1)}{n^2}$$

$$= \lim_{n\to\infty} \frac{1}{2}\left[1 - \frac{1}{n}\right]$$

$$= \frac{1}{2}[1]$$

$\therefore C = 1/2$

Since, $C > 1/2$, $t(n) = g(n)$

$\therefore t(n) \in \Theta(g(n))$

i.e $\frac{1}{2}n(n-1) \in \Theta(n^2)$

② Compare order of growth of $t(x) = 4x^3 + 3x^2$ and $g(x) = x^4$

$$C = \lim_{x\to\infty} \frac{t(x)}{g(x)}$$

$$= \lim_{x\to\infty} \frac{4x^3 + 3x^2}{x^4}$$

$$C = \lim_{x \to \infty} \left(\frac{4}{x} + \frac{3}{x^2} \right)$$

$$C = 0$$

∴ Order of growth of $t(x)$ is less than $g(x)$

$$\Rightarrow \quad t(x) \in O(g(x))$$

③ Compare order of growth $5x + 6x^2 = t(x)$ & $g(x) = 3x - 8$

$$C = \lim_{x \to \infty} \frac{t(x)}{g(x)}$$

$$= \lim_{x \to \infty} \frac{5x + 6x^2}{3x - 8}$$

$$C = \infty$$

∴ Order of growth of $t(x)$ is greater than $g(x)$

$$\therefore \quad t(x) \in \Omega(g(x))$$

L - Hopital's Rule :-

If $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{0}{0}$ (or) $\frac{\infty}{\infty}$

then apply

$$\lim_{n\to\infty} \frac{f'(n)}{g'(n)}$$

① Compare order of growth, $f(n) = \log n$, $g(n) = n$.

$$c = \lim_{n\to\infty} \frac{f(n)}{g(n)}$$

$$= \lim_{n\to\infty} \frac{\log n}{n} \quad \left(\frac{\infty}{\infty}\right)$$

$$= \lim_{n\to\infty} \frac{\frac{1}{n}}{1}$$

$$= \lim_{n\to\infty} \frac{1}{n}$$

$$c = 0$$

Since, $c = 0$ $\quad f(n) < g(n)$

$\therefore f(n) \in O(g(n))$

Sterlings Formula :-

For any large number n,

$$n! \approx \sqrt{2\pi n} \cdot \left(\frac{n}{e}\right)^n$$

② Compare order of growth of $n!$ and 2^n

$f(n) = n!$ $\quad$ $g(n) = 2^n$

$$C = \lim_{n \to \infty} \frac{f(n)}{g(n)}$$

$$C = \lim_{n \to \infty} \frac{n!}{2^n}$$

$$= \lim_{n \to \infty} \frac{\sqrt{2n\pi}}{2^n} \cdot \frac{n^n}{e^n}$$

$$= \lim_{n \to \infty} \sqrt{2\pi n} \cdot \left(\frac{n}{2e}\right)^n$$

$C = \infty$

$$\log C = \lim_{n \to \infty} \log(\sqrt{2\pi n}) + n \log\left(\frac{n}{2e}\right)$$

$\log C = \infty + \infty$

$C = \text{Antilog}(\infty)$

Since, $C = \infty$

Order of growth of $n! > 2^n$

$\therefore n! \in \Omega(2^n)$ //

⇒ Mathematical Analysis of Non-Recursive Algorithms:-

(i) Decide on Parameter(s) indicating input size

(ii) Identify the Algorithm's basic operation

(iii) Check wether no of times basic operation executed is based only on size of Input of some other property

(iv) If the count is different input of same size, then investigate the best, worst and average case efficiency seperately.

(v) Set up a sum expressing the no of times the algorithm's basic operation is executed

(vi) Using standard forms & rules of summation, obtain the closed form formula of the algorithm for operation count and establish order of growth.

Algorithm:-

```
large (A[0,.... n-1])
// Determines the value of largest element in given array A
// Input :- An array A[0,... n-1] of n elements
// Output :- The largest element.
```

```
Max ← A[0]
for i←1 to n-1 do
    if A[i] > Max
        Max ← A[i]
return Max
```

Analysis :-

① Input Size is 'n'

② Basic operation is Comparison

③ Investigate on Worst Case

④ $$C(n) = \sum_{i=1}^{n-1} 1$$

⑤ WKT, $$\sum_{i=l}^{u} 1 = u-l+1$$

Here, $u = n-1$ $l = 1$

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

$$= n-1-1+1$$

$$C(n) = n-1$$

$\therefore C(n) \in \Theta(n)$

$\therefore$ It is a linear Algorithm

⇒ Some formulas on Summation:-

① $\sum_{i=l}^{u} 1 = u-l+1 \quad (l \le u)$

② $\sum_{i=1}^{n} i = 1+2+3+\ldots n = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$

③ $\sum_{i=1}^{n} i^2 = 1^2+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{n^3}{3}$

④ $\sum_{i=1}^{n} i^K = 1^K+2^K+\ldots+n^K \approx \frac{1}{K+1} n^{K+1}$

⑤ $\sum_{i=0}^{n} a^i = 1+a+a^2+a^3+\ldots+a^n \approx \frac{a^{n+1}-1}{a-1} \quad (a \ne 1)$

⑥ $\sum_{i=1}^{n} i \cdot 2^i = 1\times2^1+2\times2^2+\ldots+n\times2^n = (n-1)2^{n+1}+2$

⑦ $\sum_{i=1}^{n} \frac{1}{i} = 1+\frac{1}{2}+\frac{1}{3}+\ldots+\frac{1}{n} \approx \ln n + \gamma$ where $\gamma \approx 0.5772$

⑧ $\sum_{i=1}^{n} \log i \approx n \log n$

=> Some Manipulations :-

① $\sum_{i=l}^{u} c a_i = c \sum_{i=l}^{u} a_i$

② $\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$

③ $\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i \quad (l \le m \le u)$

④ $\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$

=> Analyse the efficiency of the algorithm to find the element uniqueness of an array

```
UniqueArray (A[0, . . . . , n-1])
// Determine wether all the elements in a given array are
   Unique @ not
// Input :- An array A[0, . . . n-1] of n elements
// Output :- Returns "True" if all the elements in array
             are distinct and false otherwise
```

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

return True
```

Analysis :-

① Input Size is 'n'

② Basic Operation is Comparison

③ Investigate on Worst Case, Best Case & Average Case

④ Best Case Scenario :- When the first two elements in array are same.

$$C_{best}(n) = 1$$

Worst Case Scenario :- When the last two elements in array are same.

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

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

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

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

$$= n \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-2+1)}{2} - (n-2+1)$$

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

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

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

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

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

$$C_{worst}(n) \approx \frac{1}{2}n^2$$

$$C_{worst}(n) \in \Theta(n^2)$$

∴ This is a Quadratic Algorithm

③ Analyze the efficiency of Algorithm to multiply two matrices A & B of order n x n.

// Input :- Two matrices A & B of order n x n

// Output :- Product of the two matrices.

```
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
```

Analysis :-

(i) Input Size is n x n

(ii) Basic Operation is Multiplication

(iii) Investigate only on Worst Case @ only one case because the count will not change for different input of same size

(iv) $$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{K=0}^{n-1} 1 \quad (n-1+1)$$

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

$$= \sum_{i=0}^{n-1} n \sum_{j=0}^{n-1} 1 \quad (n-1+1)$$

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

$$= n^2 \sum_{i=0}^{n-1} 1 \quad (n-1+1)$$

$$= n^2 \times n$$

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

$$\therefore \quad M(n) \in \theta(n^3)$$

It is a Cubic Algorithm.

④ Find the efficiency class for finding the sum of cubes of first 'n' numbers. algorithm for

// Input :- Number of elements 'n'

// Output :- Sum of cubes of 'n' number

```
sum←0
for i←1 to n do
    sum = sum + i*i*i
return sum
```

Analysis :-

(i) Input Size is 'n'

(ii) Basic operation is Multiplication

(iii) The count will not change based on different input of same size.

$$\therefore M(n) = \sum_{i=1}^{n} 2$$

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

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

$$M(n) = 2n$$

$$\therefore M(n) \in \Theta(n)$$

$\therefore$ The above algorithm is a Linear Algorithm.

⇒ Mathematical Analysis of Recurrsive Algorithm

(i) Decide on Parameter basedon input size

(ii) Identify the basic operation

(iii) Check wether the no of times basic operation executed is based only on Input size (or) some other property, if the count is different for different ~~for~~ input of same size, analyze for all 3 cases seperately.

(iv) Set up a Reccurrence relation with appropriate initial conditions.

(v) Solve the reccurence relation to obtain the operation count and establish order of growth.

(1) Factorial of Number :-

```
Fact(n)
// Input :- An integer (positive) 'n'
// Output :- Returns factorial of a Positive integer.

if  n = 0   return 1
Else return n * Fact(n-1)
```

Analysis :-

① Input Size is n

② Basic Operation is Multiplication

$M(n) = M(n-1) + 1 \qquad M(0) = 0$ (Initial condition)

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

$= M(n-2) + 2$

$= [M(n-3) + 1] + 2$

$= M(n-3) + 3$

$\vdots$

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

Substitute 'i' by 'n' to get initial condition

$\therefore M(n) = M(n-n) + n$

$= M(0) + n$

$\therefore M(n) = n$

$\Rightarrow M(n) \in \Theta(n)$

⇒ Tower of Hanoi :-

Algorithm :-

Move $n-1$ disks from Source to Intermediate

Move n^{th} disk from source to Destination

Move $n-i$ disks from Intermediate to Destination

Analysis :-

① Input Size ⇒ n

② Basic Operation ⇒ Movement of Disks

③ Reccurence Relation :-

$$M(n) = M(n-1) + 1 + M(n-1) \quad , \text{Initial condition } M(1) = 1$$

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

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

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

$$= 2^2 [2M(n-3) + 1] + 2 + 1$$

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

$$\vdots$$

$$= 2^i M(n-i) + 2^{i-1} + 2^{i-2} + 2^{i-2} \ldots + 2^0$$

$$= 2^i M(n-i) + 2^i - 1 \quad \left[\left(\frac{1(2^i-1)}{2-1}\right) = 2^i - 1\right]$$

Put $i = n-1$

$$M(n) = 2^{n-1} \cancel{M(n-(n-1))}^{1} + 2^{n-1} - 1$$

$$= 2 \times 2^{n-1} - 1$$

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

$$\therefore M(n) \in \theta(a^n)$$

$$M(n) \in \theta(2^n)$$

③ Consider the following recursive algorithm for computing the sum of first 'n' cubes.

(i) Set up and solve the recurence relation for no of times algorithm's basic operation is executed.

(ii) Compare this algorithm with non recursive algorithm for computing this function.

Algorithm :-

Sum (n)

// Input :- number of Elements 'n'

// Output :- Sum of cubes of 'n' numbers

if $n = 1$

return 1

Else return Sum $(n-1) + n * n * n$

Analysis :-

① Input Size ⇒ n

② Basic Operation ⇒ Multiplication

③ Recurrence Relation.

$$M(n) = M(n-1) + 2 \qquad M(1) = 0$$

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

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

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

$$= M(n-3) + 6$$

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

Put $i = n-1$

$$M(n) = \cancel{M(n-(n-1))}^{\,0} + 2(n-1)$$

$$= 2(n-1)$$

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

$$\therefore M(n) \in \Theta(n)$$

Q Write an algorithm to find the number of binary computation digits in a numbers representation

Algorithm :- (Iterative)

// Input :- A positive integer 'n'

// Output :- Number of binary digits.

```
C ← 0
While n != 0
    C ← C+1
    n ← n/2
return C
```

Recursive :-

```
Binary (n)
{
    if n = 1
        return 1
    Else return Binary (n/2) + 1
}
```

Analysis :-

① Input Size :- n

② Basic Operation :- Addition

$$A(n) = A(n/2) + 1 \qquad A(1) = 0$$

By smoothness Rule,

$$A(2^k) = A(2^{k-1}) + 1$$

$$= [A(2^{k-2})] + 1 + 1$$

$$= A(2^{k-2}) + 2$$

$$= [A(2^{k-3}) + 1] + 2$$

$$= A(2^{k-4}) + 4$$

$$\vdots$$

$$A(2^k) = A(2^{k-i}) + i$$

Put $i = k$

$$A(2^k) = A(2^{k-k}) + k$$

$$A(2^k) = A(0) + k$$

$$A(2^k) = k$$

$$A(n) = \log_2 n$$

$$A(n) \in \Theta(\log_2 n) //$$

⑤ Find order of growth of given Recurrence Relation.

$$x(n) = x(n/3) + 1 \quad \text{for} \quad n > 1 \qquad x(1) = 1$$

By seamoothness theorem

$$n = 3^K \Rightarrow K = \log_3 n$$

$$x(3^K) = x(3^{K-1}) + 1$$

$$= [x(3^{K-2}) + 1] + 1$$

$$= x(3^{K-2}) + 2$$

$$= [x(3^{K-3}) + 1] + 2$$

$$= [x(3^{K-3}) + 3]$$

$$= \vdots$$

$$x(3^K) = x(3^{K-i}) + 3i$$

Put $i = K$

$$x(3^K) = x(1) + K$$

$$x(3^K) = 1 + K$$

$$x(n) = 1 + \log_3 n$$

$\therefore x(n) \in \Theta(\log_3(n))$ //

⇒ N^th Fibonacci Number :-

// Input :- A positive integer 'n'

// Output :- n^th Fibonnacci number

```
F(n)
{
if  n=0 (or) n=1
    return n;

Else

    return  F(n-1) + F(n-2)
}
```

Recurrence Relation :-

2^nd ordered Homogeneous linear recurrence relation is of form $a \cdot x(n) + b\,x(n-1) + c\,x(n-2) = 0 \longrightarrow (*)$

The above equation has a characteristic quadratic equation

$$ar^2 + br + c = 0$$

Case (i) :- Roots are real & distinct $(d > 0)$

$$x(n) = \alpha r_1^n + \beta r_2^n$$

Case (ii) :- Roots are equal ($d=0$)

$$x(n) = \alpha r^n + \beta n r^n$$

Case (iii) :- Complex roots ($d<0$)

$$x(n) = \gamma^n [\alpha \cos n\theta + \beta \sin n\theta]$$

$$\gamma = \sqrt{\mu^2+v^2} \qquad \theta = \arctan \frac{v}{\mu}$$

The recurrence relation is,

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

$$F(n) - F(n-1) - F(n-2) = 0$$

The characteristic equation is

$$r^2 - r - 1 = 0$$

$$r_{1,2} = \frac{-(-1) \pm \sqrt{(-1)^2 - 4(1)(-1)}}{2}$$

$$r_{1,2} = \frac{1 \pm \sqrt{5}}{2}$$

∴ Roots are real and distinct

$$r_1 = \frac{1+\sqrt{5}}{2} \qquad r_2 = \frac{1-\sqrt{5}}{2}$$

$$\therefore \quad F(n) = \alpha\left(\frac{1+\sqrt{5}}{2}\right)^n + \beta\left(\frac{1+\sqrt{5}}{2}\right)^n \rightarrow (1)$$

Put, $n = 0$ in (1)

$$F(0) = \alpha + \beta$$

$$\therefore \alpha + \beta = 0 \rightarrow (i)$$

Put, $n = 1$ in (1)

$$F(1) = \alpha\left(\frac{1+\sqrt{5}}{2}\right) + \beta\left(\frac{1-\sqrt{5}}{2}\right)$$

$$1 = \frac{\alpha(1+\sqrt{5}) + \beta(1-\sqrt{5})}{2}$$

$$\alpha(1+\sqrt{5}) + \beta(1-\sqrt{5}) = 2 \rightarrow (ii)$$

Multiply (i) with $-(1+\sqrt{5})$

$$-\alpha(1+\sqrt{5}) - (1+\sqrt{5})\beta = 0$$

$$\alpha(1+\sqrt{5}) + (1-\sqrt{5})\beta = 2$$

$$(-1-\sqrt{5}+1-\sqrt{5})\beta = 2$$

$$-2\sqrt{5}\,\beta = 2$$

$$\beta = \frac{-1}{\sqrt{5}}$$

$$\Rightarrow \quad \alpha = \frac{1}{\sqrt{5}}$$

$$\therefore \quad F(n) = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n$$

$$= \frac{1}{\sqrt{5}}\left(\phi^n - \hat{\phi}^n\right)$$

where, $\phi = 1.61803$

$$\hat{\phi} = \frac{-1}{\phi} = -0.61803$$

Since, $\hat{\phi}$ is very less as $n \to \infty$

it can be negleted

$$F(n) = \frac{1}{\sqrt{5}}\phi^n$$

where, $\phi = \frac{1+\sqrt{5}}{2}$

Mathematical Analysis :-

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

Initial condition
$A(0)=0 \quad A(1)=0$

$$A(n) - A(n-1) - A(n-2) = 1$$

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

$$B(n) - B(n-1) - B(n-2) = 0$$

where $B(n) = A(n)+1$

$\Rightarrow B(0)=1$ & $B(1)=1$

$$B(n) = F(n+1)$$

$$A(n)+1 = F(n+1)$$

$$A(n) = F(n+1) - 1$$

$$\therefore A(n) = \frac{1}{\sqrt{5}} \phi^{n+1} - 1$$

$$\Rightarrow A(n) \in \theta(\phi^n)$$

The recursive method is not efficient because the same function is called again and again although it is computed already.

Ex:- $F(2)$ is called while computing $F(3)$, $F(4)$ etc.

Best Algorithm for Fibonacci Number :-

$f_1 \leftarrow 0$

$f_2 \leftarrow 1$

for $i \leftarrow 2$ to n

$f_3 \leftarrow f_1 + f_2$

$f_1 \leftarrow f_2$

$f_2 \leftarrow f_3$

return f_3

The above algorithm is a Linear Algorithm.