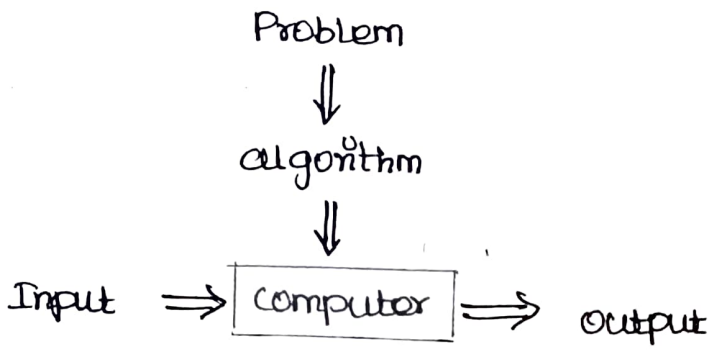
UNIT- I

Algorithm:

An algorithm is a sequence of unambiguous instruction for solving a problem.

Notation of algorithm:



Properties of an Algorithm:

1. The instruction should be clear unambiguous.
2. The same algorithm can be represented in several ways.
3. Several algorithms for solving the same problem may exist.
4. The range of input for an algorithm has to be specified carefully.

EX: 1: Greatest Common Divisor

$gcd(m, n) = gcd(n, m \bmod n)$

$gcd(60, 24) = gcd(24, 60 \bmod 24)$

$= gcd(24, 12)$

$= gcd(12, 24 \bmod 12)$

$= gcd(12, 0) = 12$ //

- gcd(m, n) : Algorithm

Step 1: If n=0, return m as the answer

otherwise,

Step 2: Divide m by n and assign the remainder to r.

Step 3: Assign n to m and r to n;

goto step 1.

- Euclid (m, n) : Pseudocode

// compute gcd(m, n)

// Input : Two non-negative number m, n

// output : greatest common divisor

while $n \neq 0$ do

$r \leftarrow m \bmod n$

$m \leftarrow n$

$n \leftarrow r$

return m.

Ex: 2 : Middle - school procedure for gcd

Step 1 : Find the prime factors of m

Step 2 : Find the prime factors of n

Step 3 : Compute the product of all common factors

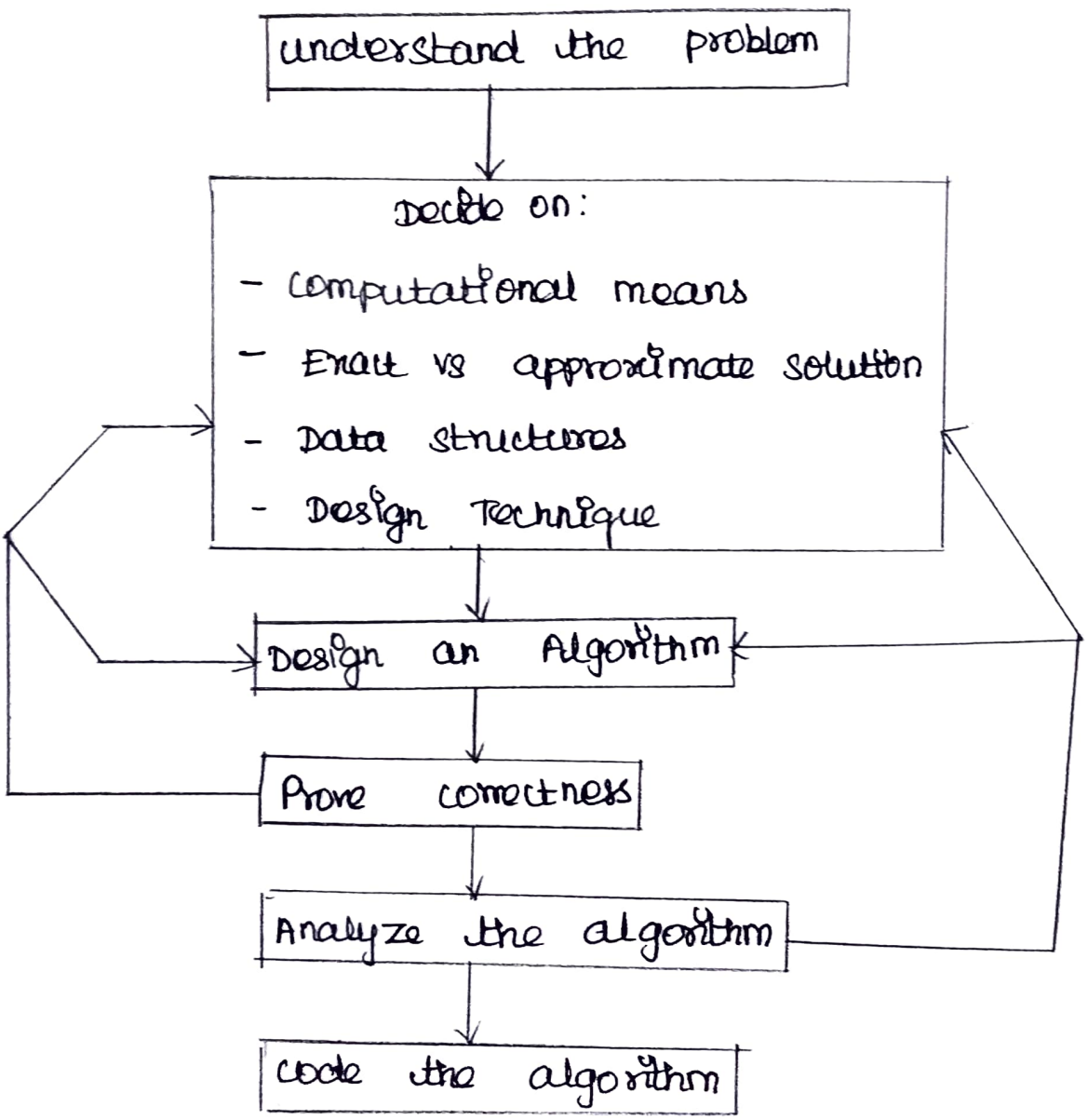
EX: gcd(60, 24)

$60 = 2 \times 2 \times 3 \times 5$

$24 = 2 \times 2 \times 2 \times 3$

gcd(60, 24) = 2 x 2 x 3 = 12 //.

Design, Analysis and coding of Algorithm:



Understand the problem:

* Read the problems carefully and ask

questions if you have any doubts.

* Do few examples by hand and think about special cases.

— Decide on computational device

(i) Random Access Machine (RAM)

- Instructions are executed one after another.
- One operation at a time

This is known as sequential Algorithm

(ii) Parallel Algorithm:

- The algorithm able to execute operations concurrently.

— Exact vs Approximate Problem solving

Exact: The algorithm provide the exact answer for all instances.

Approximate: The algorithm cannot solve the problem exactly for some instances.

Ex: - Extracting square root
- Solving non-linear equations.

— Appropriate Data structures:

Program = Algorithm + Data structure

Need to select appropriate Data structure

EX: queue / stack / Linked List / Tree.

Design Technique:

* Algorithm design Technique is a general approach to solve the problems

EX:
- Divide and conquer
- Dynamic programming
- Brute force.

— Specifying an Algorithm:

(i) Natural language
(ii) Pseudo code
(iii) flow chart

* Natural language:
  - difficult to give unambiguous statement

* pseudo code:
  - A mixture of Natural language and Programming language like construct.

* flow chart:
  - A graphical representation of algorithm.
  - Easy for understanding the flow of an algorithm.

— proving an Algorithms correctness:

* prove that the algorithm yields a required result for every input in a finite time.

— Analyzing an Algorithm:

- The algorithms are analyzed based on factors.

(i) Time efficiency
(ii) Space efficiency
(iii) Simplicity

— coding an Algorithm:

- Efficient Implementation (coding) is required for speed up.
- The modern compiler provide such a technique for efficient implementation.

IMPORTANT PROBLEM TYPES:

(1) Sorting
(2) Searching
(3) String processing
(4) Graph problems
    - Travelling salesman problem
    - graph coloring.
(5) combinatorial problems

- permutation, combination, subset

(b) Geometric problem:

- closest pair
- convex - hull

Analysis of Algorithm Efficiency:

Time efficiency: Indicates how fast the algorithm works

Space efficiency: how much additional memory required by the algorithm.

— Measuring the Input Size:

- The algorithms are run longer on larger Input

Ex: • Sorting larger Array.
• Multiplying two large matrices

$b = [\log_2 n] + 1$. // measuring the size of inputs.

Measuring Running Time:

- The running time is varied from computer to computer for some Algorithm.

∴ the no of basic steps in the Algorithm execution is the best measure.

Ex: * In sorting algorithm, the no of comparison is the basic operation.

* In Matrix multiplication, no of multiplication

is the basic operation.

let, Cop - Execution time of basic operation

C(n) → No of time the basic operation is executed

then, the running time,

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

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

Now, the input size is doubled, then how much longer?

$$\frac{T(2n)}{T(n)} \approx \frac{C_{op}\, C(2n)}{C_{op}\, C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2} n^2} = 4 \text{ //}$$

Note: * The constants are CPn called

* The Cop also cancelled.

* The order of growth only based on the input size.

— Orders of growth: Ex:

n	$\log_2 n$	$2n$	$2n \log_2 n$	$2n^2$	$2n^3$	2^n	$2n!$
10	3.3	10	3.3×10^1	10^2	10^3	10^3	3.6×10^6
10^2	6.6	10^2	6.6×10^2	10^4	10^6	1.3×10^{30}	9.3×10^{157}
⋮							

Worst case, Best-case and Average case Efficiency:

WORST CASE: The algorithm runs the longest among all possible input of the size.

$C_{worst}(n)$ = The inputs yields the largest value of basic operation count.

Ex: Sequential Search (A[0 .... n-1], K)

// Search for a given value.

// Input: An Array, consist of 'n' element

// Output: The array index of the element

```
i ← 0
while i < n and A[i] ≠ K do
    i ← i+1
If i < n return i
                    ↳ [return the index of the Key element]
else
return -1.
```

here,

$C_{worst}(n) = n$ // The searching element at the end of the array.

Best case: The algorithm runs the fastest among all possible input of size

$C_{best}(n) = 1$ // The searched element is the first element of the array.

Average - case:

To analyze the algorithm average - case efficiency we must make some assumptions,

(a) the probability of successful search is equal to p.

(b) the probability of the first match occurring in the i^{th} position of the list is same for every i.

Probability ↙ Successful search ↘

$$C_{avg}(n) = \left[1 \cdot p/n + 2 \cdot p/n + \cdots + i \cdot p/n + \cdots + n \cdot p/n\right]$$

unsuccessful $+ n(1-p)$ → if element is not true

$$= p/n\left[1+2+3+\cdots+n\right] + n(1-p)$$

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

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

If the search is successful,

i.e, $p=1$ and, $\boxed{C_{avg}(n) = (n+1)/2 \Rightarrow \Theta\frac{(n+1)}{2}}$

If the search is not successful,

$P=0$ then

$$C_{avg}(n) = n \Rightarrow \Theta(n) \text{ //.}$$

Amortized efficiency:

It applies when a sequence of operations performed on the same data structure.

Ex: Multiple Search on the same Array.

ASYMPTOTIC NOTATIONS:

To compare and rank the efficiency of different algorithm, there three different notations are used.

(i) O - big oh

(ii) Ω - big omega

(iii) Θ - big theta

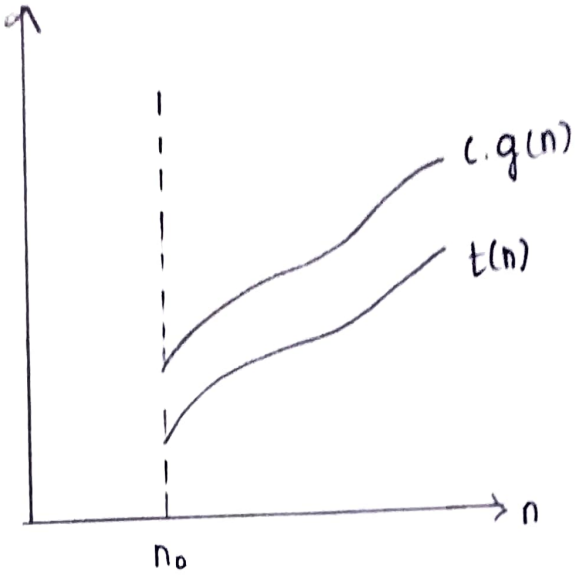
Big oh : O: A function $t(n)$ is said to be in $O(g(n))$, denoted $t(n) \in O(g(n))$.

$t(n)$ is bounded above by some constant multiple of $g(n)$ for all large n.

i.e) $t(n) \leq c \cdot g(n)$ for all $n \geq n_0$

c - positive constant

n_0 - non-negative integer.



Ex:1: $100n + 5 \in O(n^2)$

It is in the form

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

here, $t(n) = 100n + 5$

$g(n) = n^2$

need to prove $t(n) \leq c.g(n)$ for some n and c

$100n + 5 \leq c.n^2$

$c = 101$
$n_0 = 3$

when $c = 101$

i) $n = 0$: $5 \leq 0$

ii) $n = 1$: $105 \leq 101$

iii) $n = 2$: $205 \leq 404$

iv) $n = 3$: $305 \leq 909$

v) $n = 4$: $405 \leq 1616$

vi) $n = 5$: $505 \leq 2525$

vii) $n = 6$: $605 \leq 3636$

viii) $n = 7$: $705 \leq 4949$

Ex:2: $12n^2 + 8 \in O(n)$

It is in the form

$t(n) \leq c.g(n)$

Here, $t(n) = 18n^2+8$

$g(n) = n$

$$18n^2+8 \leq c.n$$

→ never be true

EX:3: $4n^2-3n \in O(n^2)$.

It is in the form

$t(n) \leq c.g(n)$

Here, $t(n) = 4n^2-3n$

$g(n) = n^2$

$4n^2-3n \leq c.n^2$

$\therefore$ $c = 4$, $n_0 = 0$

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

When $c=4$

i) $n=0$ $\quad 0 \leq 0$

ii) $n=1$ $\quad 1 \leq 4$

iii) $n=2$ $\quad 10 \leq 16$

iv) $n=3$ $\quad 27 \leq 36$

EX:4: $2n+8 \in O(n)$

It is in the form

$t(n) \leq c.g(n)$

Here, $t(n) = 2n+8$

$g(n) = n$

$2n+8 \leq c(n)$

$c=3$ $\quad n_0 = 8$

$2n+8 \leq 3(n)$

When $c=3$

i) $n=8$ $\quad 24 \leq 24$

ii) $n=9$ $\quad 26 \leq 27$

iii) $n=10$ $\quad 28 \leq 30$

iv) $n=11$ $\quad 30 \leq 33$

EX: 5: $n^2 + 5n = O(n^3)$

It is in the form

$$t(n) \leq c \cdot g(n)$$

Here, $t(n) = n^2 + 5n$

$$g(n) = n^3$$

$$n^2 + 5n \leq c \cdot n^3$$

$\boxed{c = 2}$ $\boxed{n_0 = 2}$

$$n^2 + 5n \leq 2 \cdot n^3$$

when $c = 2$

(i) $n = 0$: $0 \leq 0$

(ii) $n = 1$: $6 \leq 2$

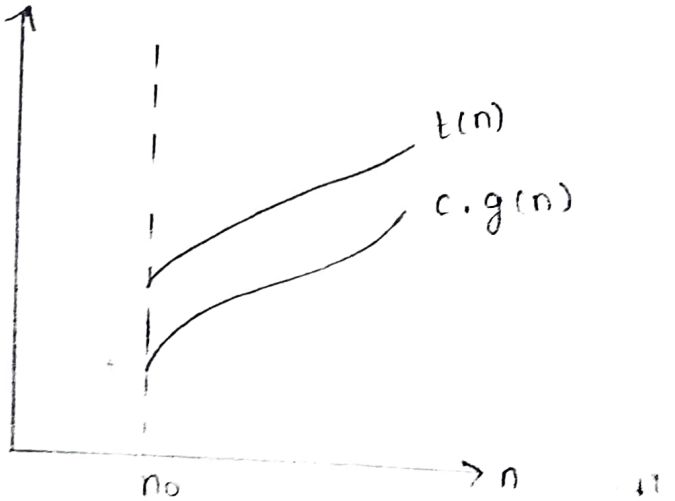
(iii) $n = 2$: $14 \leq 16$

(iv) $n = 3$: $24 \leq 54$

Big Omega (Ω):

* 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.

ie, $t(n) \geq c \cdot g(n)$ for all $n \geq n_0$



Ex:1: $n^3 \in \Omega(n^2)$

Here, $t(n) = n^3$

$g(n) = n^2$

need to prove,

$t(n) \geq c.g(n)$

$n^3 \geq c.n^2 \Rightarrow n^3 \geq n^2$

$c=1$
$n_0=0$

When $c=1$

i) $n=0$ $0 \geq 0$

ii) $n=1$ $1 \geq 1$

iii) $n=2$ $8 \geq 4$

iv) $n=3$ $27 \geq 9$

Ex:2: $5n^2 \in \Omega(n^2)$

Here, $t(n) = 5n^2$

$g(n) = n^2$

need to prove,

$t(n) \geq c.g(n)$

$5n^2 \geq c.n^2 \Rightarrow 5n^2 \geq 4n^2$

$\therefore$

$c=4$
$n_0=0$

When $c=4$

i) $n=0$ $0 \geq 0$

ii) $n=1$ $5 \geq 4$

iii) $n=2$ $80 \geq 16$

iv) $n=3$ $45 \geq 36$

Ex:3: $n^3+n \in \Omega(n^3)$

Here, $t(n) = n^3+n$

$g(n) = n^3$

need to prove,

$t(n) \geq c.g(n)$

$n^3+n \geq c.n^3 \Rightarrow n^3+n \geq n^3$

$c=1$
$n_0=0$

When $c=1$

i) $n=0$ $0 \geq 0$

ii) $n=1$ $2 \geq 1$

iii) $n=2$ $10 \geq 4$

iv) $n=3$ $30 \geq 27$

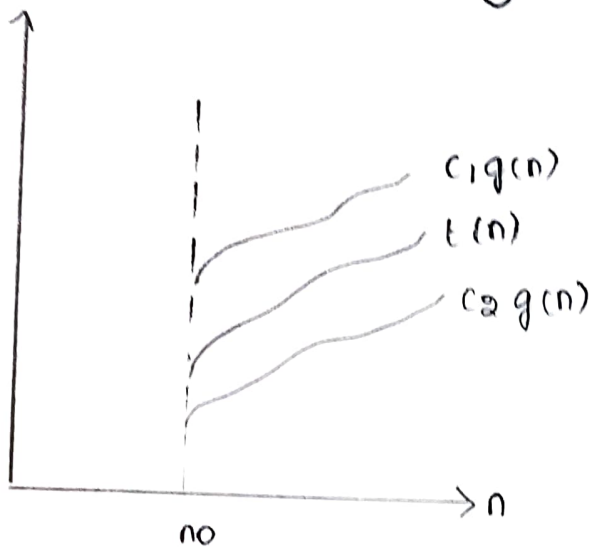
Big - theta (Θ):

A function $t(n)$ is said to be in $\Theta\, g(n)$, denoted $t(n) \in \Theta(g(n))$.

If $t(n)$ is bounded both above and below by some positive constant multiples of $g(n)$ for all large

i.e) $c_2\, g(n) \leq t(n) \leq c_1\, g(n)$ for all $n \geq n_0$

(Ω over $c_2\, g(n) \leq t(n)$; O over $t(n) \leq c_1\, g(n)$)



Ex1: $\frac{1}{2}\, n(n-1) \in \Theta(n^2)$

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

$g(n) = n^2$

$\frac{1}{2}\, n^2 - \frac{1}{2}\, n \geq c_1 (n^2)$

Proof for upper bound (O):

$\frac{1}{2}\, n(n-1) = \frac{1}{2}\, n^2 - \frac{1}{2}\, n \leq n^2$ for $n \geq 0$

$\frac{1}{2}\, n^2 - \frac{1}{2}\, n \leq c_2 (n^2)$

$c_2 = 1/2$
$n_0 = 0$

Proof for lower bound (Ω):

$$\frac{1}{2} n(n-1) = 1/2\, n^2 - 1/2\, n \geq 1/4\, n^2$$

$$1/2\, n^2 - 1/2\, n \geq c_1(n^2)$$

$c_1 = 1/4$
$n_0 = 2$

$\Rightarrow c_1 \cdot g(n) \leq t(n) \leq c_2 \cdot g(n)$

$$\frac{1}{4} n^2 \leq 1/2\, n^2 - 1/2\, n \leq 1/2 \cdot n^2$$

$$n_0 = \max(0, 2)$$

$\therefore$ We can select

$$c_1 = 1/4 \;,\; c_2 = 1/2 \;,\; n_0 = 2 \quad //.$$

EX:2: $6n^2 - 8n \in \Theta(n^2)$

Here, $t(n) = 6n^2 - 8n$

$g(n) = n^2$

need to prove,

$$c_1 g(n) \leq t(n) \leq c_2 g(n)$$

Big Oh (O):

$$6n^2 - 8n \leq c_2 * n^2$$

$\therefore$

$c_2 = 6$
$n_0 = 0$

$\therefore 6n^2 - 8n \leq 6n^2$

When $c_2 = 6$

i) $n=0$: $0 \leq 0$

(ii) $n=1$: $-2 \leq 6$

(iii) $n=2$: $8 \leq 24$

iv) $n=3$: $30 \leq 54$

Big omega (Ω):

$$c_1 n^2 \leq 6n^2 - 8n$$

$$6n^2 - 8n \geq c_1 (n^2)$$

$c_1 = 1$
$n_0 = 2$

$$\Rightarrow \quad n^2 \leq 6n^2 - 8n \leq 6n^2$$

$$n_0 = \text{Max}(0, 2)$$

We consider,

$$c_1 = 1, \quad c_2 = 6, \quad n_0 = 2$$

When $c_1 = 1$

(i) $n = 0$: $0 \geq 0$

(ii) $n = 1$: $-2 \geq 1$

(iii) $n = 2$: $8 \geq 4$

(iv) $n = 3$: $30 \geq 9$

Mathematical Analysis of Non-recursive Algorithms

General plan:

1. Decide on a parameter indicating an input size.
2. Identify the algorithms basic operation.
3. Find the number of times the basic operation is executed.
4. Set up a sum expressing the number of times the basic operation is executed.
5. Using standard formulas find the closed form for the count and get order of growth.

Ex 1: Maximum Element in an array

Algorithm: Max Element (A[0 . . . . n-1])

// Determine the Max element

// Input: An array A

// Output: The maximum element

```
maxval ← A[0]
for i←1 to n-1 do
    If A[i] > maxval
        maxval ← A[i]
return Maxval.
```

Analysis:

1. The Input size is n
2. The basic operation is If A[i] > maxval

∴ No of times $C(n) = \sum_{i=1}^{n-1} (1)$

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

$= n-1$

$\in \Theta(n)$

Ex 2: Matrix Multiplication

Algorithm: Matrix Multiplication

Input: An array A, B of n element

Output: $C = A * B$

```
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.
```

Analysis:

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

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

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

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

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

$$= n^3$$

$$\in \Theta(n^3)$$

EX: 3: Unique Element problem (or) Distinct Element

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

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

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

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

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

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

Note:

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

$$= \sum_{i=0}^{n-2} (n-1) - \left[\frac{(n-2)(n-1)}{2}\right]$$

$$= (n-1)[n-2-0+1] - \left[\frac{(n-2)(n-1)}{2}\right]$$

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

$$= (n-1)^2 - \left[\frac{(n-2)(n-1)}{2}\right] \quad [\text{Taking LCM}]$$

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

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

$$= \frac{n-1[2n-2-n+2]}{2}$$

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

$$= \frac{n2}{2} - \frac{n}{2}$$

$\in \Theta(n2)$

∴ The algorithm needs to compare all $n(n-1)/2$ times.

EX:4: No of binary digits

Algorithm: Binary(n)

// Input: A positive decimal, Integer n

// output: No of binary digits.

count ← 1

while n > 1 do

count ← count + 1

n ← n/2

return count

8 4 2 1

8 → 1 0 0 0 ⇒ 4 digits

8 > 1 4 > 1 2 > 1 1 > 1 ↓ false

c ← 2 c ← 3 c ← 4

n = 4 n ← 2 n ← 1

32 > 1 16 > 1

c ← 2 c ← 3

n = 16 n = 8

⇒ No of true cases, $\log_2 n$

⇒ No of false case 1

No of comparison = $\log_2 n + 1$

$\log_2 2^3 = 3 \log_2 2$

$= 3$

$\log_2 n + 1 \leftarrow$ false case

EX:5: Insertion sort

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

for $i \leftarrow 1$ to $n-1$ do

$v \leftarrow A[i]$

$j \leftarrow i-1$

while $j \geq 0$ and $A[j] > v$ do

$A[j+1] \leftarrow A[j]$

$j \leftarrow j-1$

$A[j+1] \leftarrow v$

Analysis:

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

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

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

$$\in \Theta(n^2)$$

Note:

$1+2+3+\cdots+n-1$

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

Mathematical Analysis of Recursive Algorithm:

General plan for analysing Recursive Algorithm:

1. Decide an parameter indicating input size
2. Identify the basic operation.
3. Find no of time the basic operation is executed.

4. Set up a recurrence relation

5. Solve the recurrence relation

Recursive Algorithm:

A algorithm call itself with different Parameter value to solve the problem.

EX:1: Factorial(n):

// compute n! recursively

// Input: A non-negative Integer n

// output: The value of n!

If n=0 return 1 ⟹ Initial condition

else

return Factorial(n-1) * n ⟹ basic operation

Analysis:

1. Input Size = n
2. The basic operation Multiplication
3. No of times, M(n)
4) $M(n) = M(n-1) + 1$ for $n > 0$.

   ↓ (n-1) multiplication ; ↳ nth value multiplication

5) TO solve the recurrence relation we need Initial condition.

Eg: 3! = 2! * 3

= 1! * 2 * 3

= F(0) * 1 * 2 * 3

f(3) ⇒ f(2) * 3

f(2) ⇒ f(1) * 2 * 3

f(1) → f(0) * 1 * 2 * 3

= 1 * 1 * 2 * 3

Here, If n=0 return 1

i.e) No multiplication.

∴ n = 0,

M(n) = 0

Note:

M(0) = 0

↑ ↑

The calls stop when n = 0 — no multiplications when n = 0

~~Initial condition~~ : M(0) = 0

Backward Substitution Method:

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

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

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

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

= M(n-3) + 3

⋮

M(n) = M(n-i) + i // Have to bring Initial condition

M(n) = M(n-n) + n

= M(0) + n

= 0 + n

M(n) = n

∈ Θ(n)

Forward Substitution:

M(n) = M(n-1) + 1 for n > 0

Initial condition: $M(0) = 0$

forward substitution,

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

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

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

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

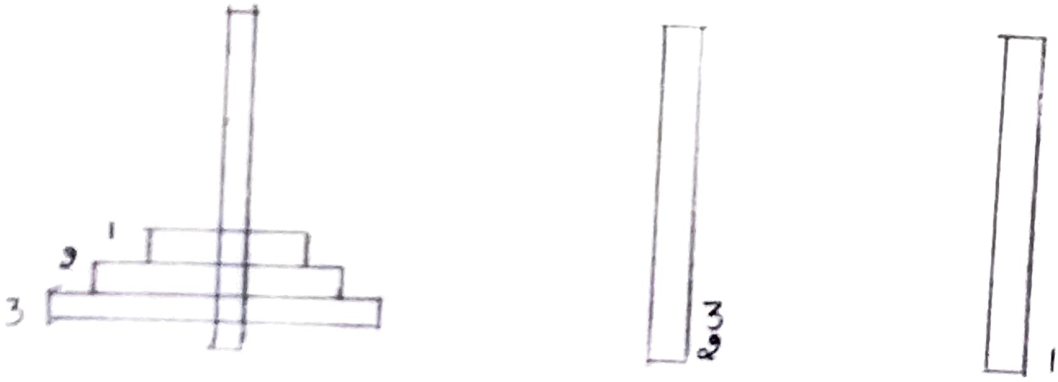
$$M(3) = M(3-1) + 1$$

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

$$\boxed{M(n) = n}$$

$$\in \theta(n)$$

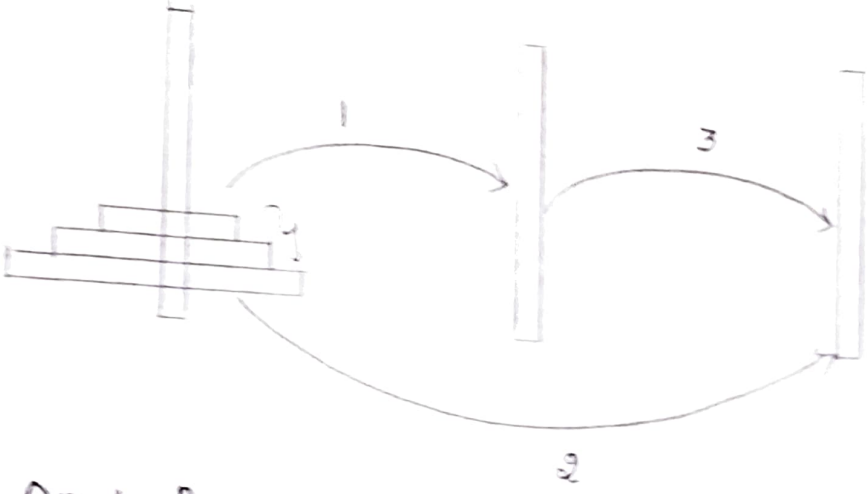
Ex: 2: Tower of Hanoi problem



* n disks of different sizes and three pegs

* Initially all disks are in one pegs largest at the bottom and smallest on the top

* The goal is move all the disks to the third peg using second one as auxiliary



Analysis:

→ No of move $M(n)$

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

$M(1) = 1 \Rightarrow$ Initial condition

Backward Substitution:

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

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

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

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

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

[$1+2+2^2+2^3+\ldots$

$1+2+4+8+16+\ldots$

$= 2^n - 1$]

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

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

$M(n) = \Theta(2^n)$

EX: 3: Number of Binary digit

// Input: A positive Integer

// Output: The no of binary digit

```
BinRec(n)
If n=1 return 1
else
return BinRec([n/2]) + 1
```

Analysis:

Recurrence equation:

$$A(n) = A([n/2]) + 1 \quad \text{for } n > 1$$

$$A(1) = 0 \Rightarrow \text{Initial condition}$$

Solution:

Assume, $\boxed{n = 2^k} \Rightarrow$ Smoothness Rule

$$A(2^k) = A\left[\frac{2^k}{2}\right] + 1$$

$$= A(2^{k-1}) + 1 \quad \text{for } k > 0$$

$$A(2^k) = A(2^{k-1}) + 1 \quad [\text{Sub: } A(2^{k-1}) = A(2^{k-2}) + 1]$$

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

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

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

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

$\vdots$

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

$= A(2^0) + k$

$= A(1) + k$

$= k$

Convert k term to n term,

$A(n) = \log_2 n$

$A(n) \in \Theta(\log n)$

$n = 2^k$

$\therefore k = \log_2 n$

EX: 4: Solve the following

$x(n) = x(n-1) + 5$ for $n > 1$,

$x(1) = 0$

$x(n) = x(n-1) + 5$ [Sub: $x(n-1) = x(n-2) + 5$]

$x(n-1) = [x(n-2) + 5] + 5$

$x(n-1) = x(n-2) + 2 \times 5$ [Sub: $x(n-2) = x(n-3) + 5$]

$x(n-2) = [x(n-3) + 5] + 2 \times 5$

$= x(n-3) + 3 * 5$

$x(n-i) = x(n-i-1) + (i-1) \times 5$

$= x(n-(n-1)) + (n-1) \times n$

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

$= 0 + n(n-1)$

$\boxed{x(n) = n(n-1)}$ //

EX:5: Solve: $x(n) = x(n/3) + 1$ for $n > 1$

$x(1) = 1$

let, $n = 3^k$

$x(3^k) = x(3^k/3) + 1$

$= x(3^{k-1}) + 1$ [sub: $x(3^{k-1}) = x(3^{k-2}) + 1$]

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

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

$= [x(3^{k-3}) + 1] + 2$ [sub: $x(3^{k-2}) = x(3^{k-3}) + 1$]

$= x(3^{k-3}) + 3$

$\vdots$

$= x(3^{k-k}) + k$

$= x(3^0) + k$

$= 1 + k$

$\boxed{x(n) = 1 + \log_3 n}$

Note: $n = 3^k$

$k = \log_3 n$

Ex:6: Solve $x(n) = 3x(n-1)$ for $n>1$; $x(1)=4$

$x(n) = 3x(n-1)$

$= 3[3x(n-2)]$ [Sub: $x(n-1) = 3x(n-2)$]

$= 3^2 x(n-2)$

$= 3^2[3x(n-3)]$ [Sub: $x(n-2) = 3x(n-3)$]

$= 3^3 \cdot x(n-3)$

$\vdots$

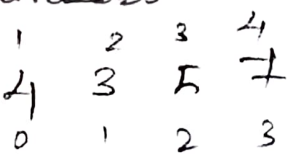
$= 3^{n-1} \cdot x(n-(n-1))$

$= 3^{n-1} \cdot x(1)$

$\boxed{x(n) = 4 \cdot 3^{n-1}}$ $\therefore$ $x(n) \in (3^n)$

EX:7: consider the following recursive algorithm for computing the sum of n given numbers

```
Add(a, n)
// Input: positive integer n, array a
// Output: The sum of n given numbers
If n<0 return 0
else
return Add(a, n-1) + a[n]
```



(9, 3)+ +

(a, 2) + 5 + 7

(a) Setup and solve recursive Algorithm.

(b) Compare with the non-recursive Algorithm

(a) Recurrence equation:

$A(n) = A(n-1) + 1, \quad n > 1, \quad A(0) = 0$

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

$= A(n-2) + 2$ [Sub: $A(n-2) = A(n-3)+1$]

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

$= A(n-3) + 3$

$\vdots$

$= A(n-n) + n$

$= A(0) + n$

$= n$

$\boxed{A(n) = n}$

(b) Non-recursive Algorithm:

```
Add (a, n)
i ← 0
while i < n do
    S = S + a[i]
    i++;
```

Analysis:

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

$$\boxed{A(n) = n}$$

EX: 8: Algorithm for computing Fibonacci Numbers

Algorithm : F(n)

// Input : A non-negative integer n

// output : n^{th} Fibonacci number

If $n \leq 1$ return n

else

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

Recurrence equation:

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

$A(0) = 0$, $A(1) = 0$

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

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

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

$B(0) = 1$, $B(1) = 1$

characteristic equation,

$r^2 - r - 1 = 0$

roots are different,

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

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

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

and

$$\boxed{A(n) = \frac{1}{\sqrt{5}} \theta^n}$$

//.