Computer Algorithm

Algorithm — Program

In- Depends on programming language

- with reasonal
- language

No Software
↑ Well defined steps

Scientist :-
Mohammad ibn Mussa Al Khuwarizmi
9th century
5 steps to solve any problem

Defination of algorithm :-

Well defined computational steps of sequance of steps use 0 or more than zero input and it should produce at least one output as solution to problem within a finite amount of time.

→ If it dosen't produce a single output Then it is not algorithm

Properties of algorithm :-

1. Input :- 0 or more than zero inputs supplied extrnally.

2. Output :- Produce atleast 1 output

3. Definiteness :- unambiguous & clear, simple every step - shouldn't create ambiguity.

4. Finiteness :- Every step's processing should terminate in finite time - otherwise we will not

5. Effectiveness :- Every step should be effective basic which is understandable to other users.

u1 -> Partial algorithm

u2 — work → Produce Complete soln

Specifications of algorithm -

Pseudocode conventions
- Rules to write algorithm

1] Comments :- Description of steps
" // "

2] Blocks :- { } (Body of functions) — conditional looping

A - To indicate same type of statements
main() - Standard fn

3] Identifier -
Name given to variable
used to store value
1. dosen't start with digit, special character.
2. No white space.
3. Should not use keywords.
4. Underscore are allowed.
5. Length : 32 bits are significant (MSB)

4] Assignment Statement -
Ex. int a = 15;
b += 1
Not an ex. a == d;
compiler flow - top. bottom left. right

5] Boolean Value -
2 boolean values (0/1)
bool - keyword is used.

6] Array -
To access elements of multi-dimensional array [ ] is used.

7] Conditional statements.
if, else, else-if
switch, ternary operator.

Ex. 1) if < condition > then < statement >

2) if < condition 1 > then < statement 1 >
else < stmt 2 >

3) Looping statements -
do - while , for , do

Ex. while < condition > then < stmt 1 >

Repeat
{
——
——
——
}
until < condition >;

at least one output

Ex. i = 1;
for (i < 10)
{
i++;
}

— feasible —
initialization , inc/dec are optional
Condition part - compulsory.

for var 2 = var 1 to var n step step do
{
< var 1 >;
< var 2 >;
}

9] Input / Output :

Read / Write.

10] Header and Body :

header -> Algorithm Name_of_algo (parameter)

Body -> {

}

* Recursive Algorithm

function call itself

In case of algorithm - direct recursive

```
A1
{
    ---
    A1
}
```

- Indirect recursive

```
A1
{
    ---
    B1
}
B1
{
    ---
    A1
}
```

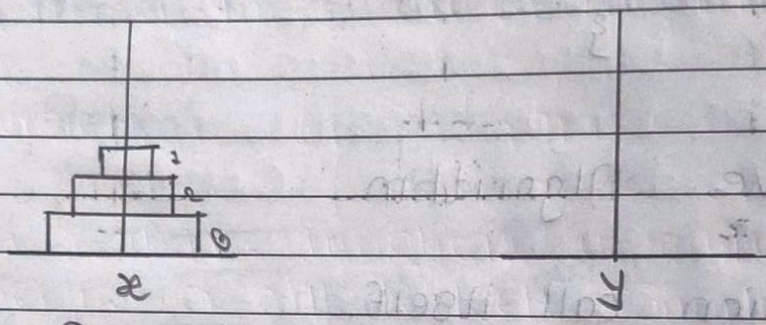
Real life example of recursion :-

Indirect :-

Precipitation
Evaporation
Condensation

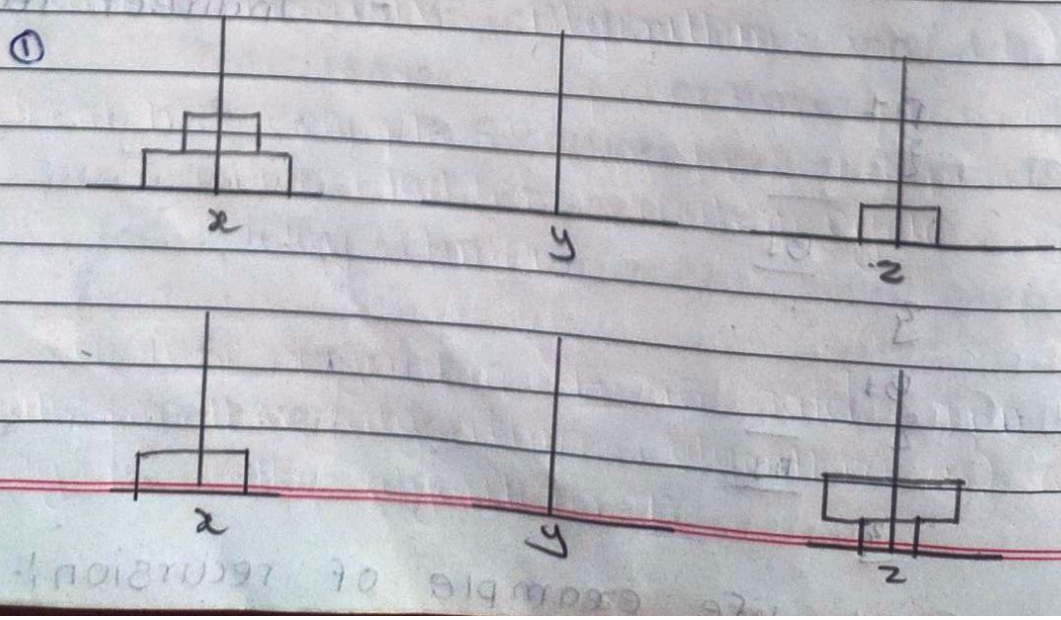
Tower of Hom Honai :-

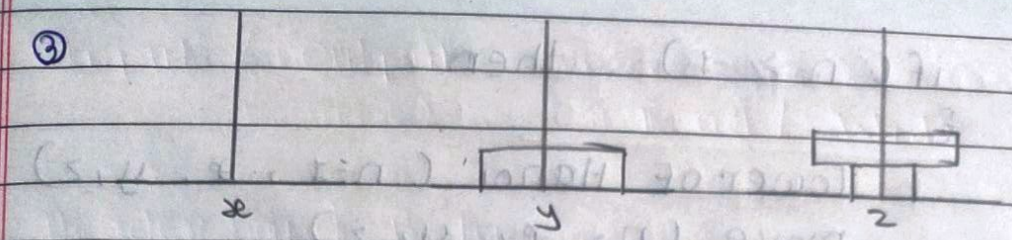
No condition :-

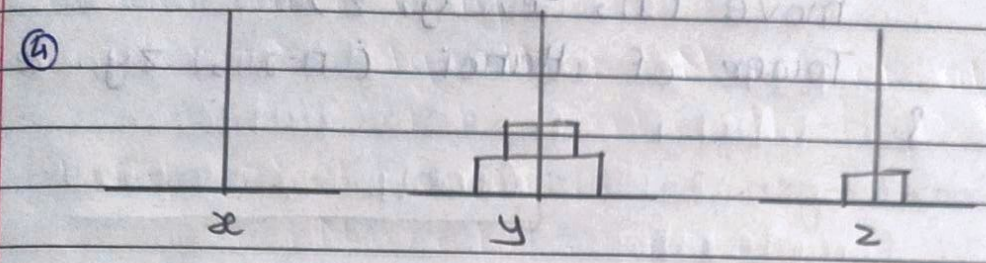


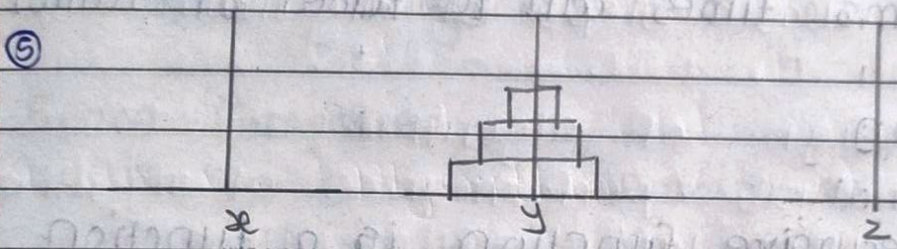
Source Destination.

1. 1 → z
2. 2 → y → z
3. 3 → y
4. 1 → y
5. 1 → y

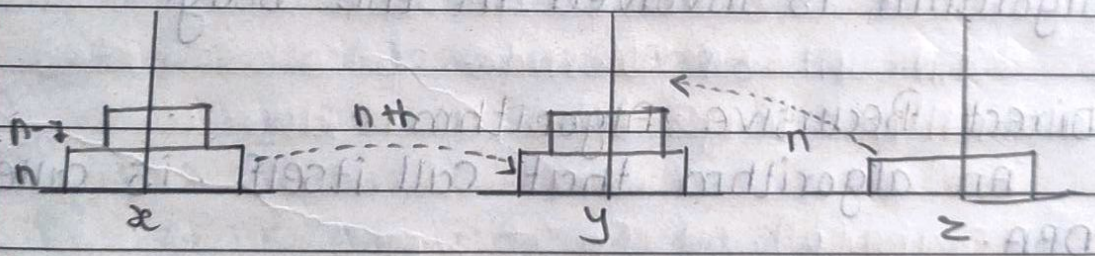








With Constraints



① Move n-1 x → z
② Move n x → y
③ Move n-1 x → y

Algorithm of tower of Hanoi-

↓ No. of discs.

Algo Tower of Hanoi (n, A, B, C)
x y z

{

```
if( n >= 1)  then
{
    Tower of Hanoi ( n-1 , x, y, z)
    move (n, x, y, z)
    Tower of Hanoi (n-1 , zy, x)
}
}
```

8 items times call to tower of Hanoi.

* Recursion

A recursive function is a function that is define or called itself. Similarly an algorithm is said to be recursive if the same algorithm is involved in the body.

Direct Recursive Algorithm-
An algorithm that call itself. is called as DRA.

Indirect Recursive Algorithm -

Real life example -
Onion, ladder, water cycle.
delay timers.

Consider Algorithm A is said IRA if it calls another algorithm (algorithm B) which agains calls algorithm A.

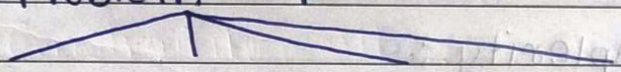
Examples of Recursive -

factorial
fibonacci
Tower of Hanoi.

Performance analysis of algorithm -

Need of analysis :-

Space Complexity :- Amount of storage required to produce result.



Soln 1 | Soln 2 | Soln 3 | Soln 4
t1 s1 | t2 s2 | t3 s3 | t4 s4

within less time & space - higher performance

$t_2 < t_1 < t_3 < t_4$
$s_2 < s_4 < s_3 < s_1$

All algo gives soln but performance is imp →

Algo 1 (2*k) Algo 2 (k+k)

Consider k = 6 6+6 = 12
2*6 = 12

```
  0  1  1  0   (6)
  0  0  1  0   (2)
  0  0  0  0
  0  1  1  0
0 0 0 0 0 0
0 0 0 0 0 0
```

n x n (4x4 = 16 unit time)

digits.

```
  0   1   1   0   (6)
  0   0   1   0   (2)
 ---------------
  1   1   0   0
```

n+n = 2n (4 unit time)

n

(16) t1 > 4 (t2)

Note - Basic steps of algorithm

imp
1. Design of algorithm
2. Analysis / performance of algorithm
3. Implementation of algorithm.

Space Complexity :

Amount of storage space required by algorithm to produce a result.

Space(p) = Cp + Sp

↑ Constant or static var.

↑ Dynamic var present in program (run time / depend on another int a = x[i])

p

var
int a = 6

Program to perform addition of two variables and result stored in third variable.

```
main ()
{
    int x, y, z;
    printf (" Enter x, y ");
    scanf (" %d %d ", &x, &y);
    z = x+y
    printf (" Result of add = %d ", z);
}
```

∴ Space (P) = Cp + Sp
= 3 units + 0 units
= 3 × 4 + 0 × 4
= 12 units

Algorithm :-

Array size, of array

```
Algo Sum (x, n)
{
    int total = 0
    for i ← 1 to n step step do
    {
        total = total + x[i]
                        ↑ Dynamic variable
    }
}
```

Space Complexity -

Cp + Sp
3 + n (units)

x, n, total - fixed variable

Time Complexity -

for (i=0; i<n; i++) also same for
for(i=n; i<0; i--)

{
    Stmt; // execute n time
}

Stmt execute n time
& loop execute n+1 because where also check one false condition.

Time complexity -

(n+1) +n = 2n +1 = O(n)

Asyncroti

* Asymptotic Notations -

Used to represent time complexity of algorithm.

IMP 1) $1 \leq \log n \leq \sqrt{n} \leq n \leq n\log n \leq n^2 \leq n^3 \ldots < 2^n < 3^n \ldots < n^n$

There are 3 types:-

1. Bigoh (O)

The function $f(n) = O(g(n))$ iff $\exists$ the constant c & n_0 such than $f(n) \leq c * g(n)$ for all n. $n > n_0$

formula of Bigoh

2. (Ω) Omega

The function $f(n) = \Omega(g(n))$ iff $\exists$ (there exists the constant c & n_0) such than $f(n) \geq c * g(n)$ for all n. $n > n_0$

1) Example (Bigoh)

1) $f(n) = 2n + 3$

$2n+3 \leq 5n$

$n = 1$

$2+3 \leq 5 \quad O(n)$

2) $f(n) = 2n+3$

$2n+3 \leq 2n+3n$

3) $f(n) = 2n+3$

$2n+3 \leq 2n^2+3n^2 \quad O(n^2)$

2) Example (Ω) omega.

1) $f(n) \geq \Omega\ g(n)$

$f(n) = 2n+3$

$2n+3 \geq 1 * n \quad \Omega(n)$

$c * g(n)$

2) $f(n) = 2n+3$

$2n+3 \geq \log n$

$= \Omega(\log n)$

3) $2n+3 \gg n^2$ not allowed in omega.

$f(n) = 2n+3$

$2n+3 = \sqrt{n}$

$\Omega(\sqrt{n})$

3. Θ (Theta) Notation :-

The function $f(n) = \Theta(g(n))$ iff $\exists$ the constant c & ~~n_0~~ n_0

such than $\boxed{c_1\, g(n) \leq f(n) \leq c_2\, g(n)}$ all n
$n > n_0$

Example:-

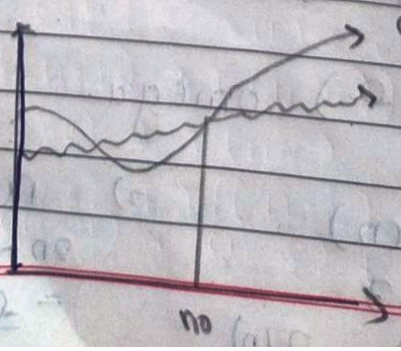
1) $f(n) = 2n+3$

$\sqrt{n} \leq 2n+3 \leq 10n$

$\log n \leq 2n+3 \leq 2n^2 + 3n^2$

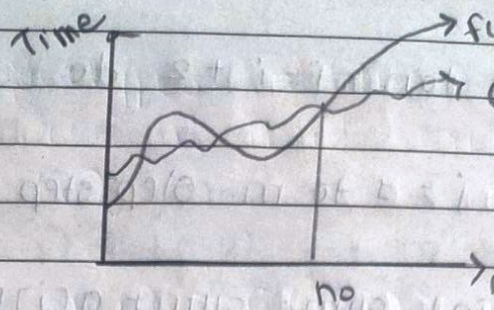
$\underline{\Theta(n)}$ - time complexity

Graph :-

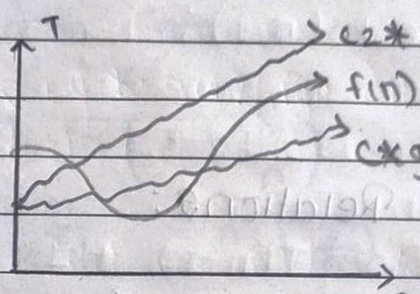


① Bigoh

② Omega (Ω)



③ Theta (Θ)



Algorithm of Sum (a[], n, m)

```
{
    for i = 1 to n step step do                    (n+1)
    {
        for j = 1 to m step step do                n×(m+1)
        {
            int sum = sum + a[i][j];               } n×(m)
        }
    }
    return (sum);
}
```

(n)

$= n + 1 + n \times (m+1) + n \times (m) + 1$

$= 2nm + 2n + 2$

$n + 1 + n(m+1) + n(m) + 1$

$= n + n(m) + n(m+1) + 1 + 1$

$= n + nm + nm + n + 1 + 1$

$= 2n + 2nm + 2$

$= 2nm + 2n + 2$

$= O(n)$

```
for i = 1 to n  i = i + 2  do
{
    for j = 1 to m  step step do
    {
        int sum = sum + a[i][j]
    }
}
```

n/2

Recurence Relations:

1. Substitude method } syllabus
2. Recurance Tree method.
3. Master theorm

1. Sustitude method

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \quad \text{Base condition} \\ n * T(n-1) & \text{— eqn ①} \end{cases}$$

$T(n-1) = (n-1) * T((n-1)-1))$

$T(n-1) = (n-1) * T(n-2)$ — eqn ②

$T(n-2) = (n-2) * T((n-2)-1)$

$T(n-2) = (n-2) * T(n-3)$ — eqn ③

Substitute eqn ② into eqn ①

$= n * ((n-1) * T(n-2))$

$= n * (n-1) * T(n-2)$

$= n * (n-1) * T((n-2) * T(n-3))$

$= n * (n-1) * (n-2) * T(n-3) \ldots n-1$

$= n*(n-1)*(n-2)*(n-3)\cdots T(n(n-1))$

$= n*(n-1)*(n-2)\cdots T(1)$

$= n*(n-1)*(n-2)\cdots 1$

$= n*(n-1)*(n-2)\cdots 2*1$

$*(n-3) \quad 3*$

Take n common from above eqn

$$n\left(1-\frac{1}{n}\right)*n\left(1-\frac{2}{n}\right)*\cdots n\left(\frac{2}{n}\right)*n\left(\frac{1}{n}\right)$$

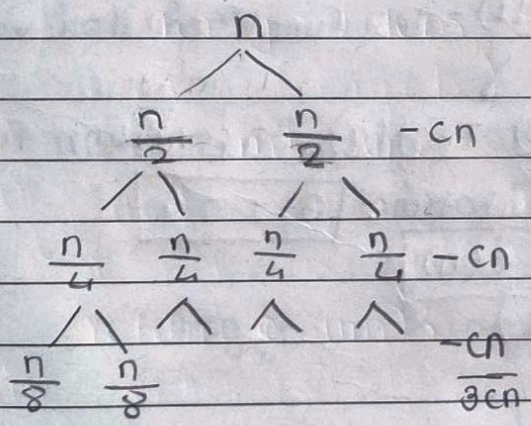
$n\left(1-\frac{3}{n}\right)*\cdots n\left(\frac{3}{n}\right)$

$= n*n\cdots*n = n^n$

$= O(n^n)$

Recrance Relation (using recursion tree)

$T(n) = 2T(n/2) + cn$

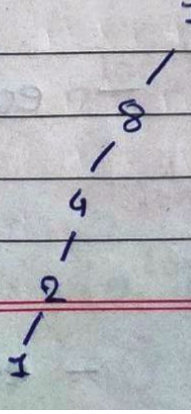


constant variable so keep it is idle

$= \log_2 n * cn$

$= n\log_2 n = O(n\log n)$

Suppose $n = 16$



hight = 4

8.
```
void Test ( int n)
{
   if ( n > 0)
   {
      printf ("%d", n);
      Test (n-1);
   }
}
```

T(n) Iteration-

0
0
1
T(n-1)
0
0

-> $T(n) = T(n-1) + 1$ n = 3

Test (3) Test funtⁿ call for val 3

pf (3) Test (2) - Test functⁿ call for val 2

pf(2) Test(1) - Test functⁿ call for val 1

pf(1) Test(0) - condition false

checking condition at time 3+1 = 4

for n n+1 = O(n)

$$T(n) = \begin{cases} 1 & \text{if } n = 1 \\ 2T(n/2) + n & \text{otherwise} \end{cases} \quad \text{— eqⁿ (1)}$$

Replace n/2 by n/2 eqⁿ (1)

$$T(n/2) = 2T\left(\frac{n/2}{2}\right) + (n/2)$$

$$= 2T(n/4) + (n/2) \quad \text{— eqⁿ (2)}$$

Replace n by n/4 eqⁿ (1)

$$T(n/4) = 2T\left(\frac{n/4}{2}\right) + (n/4)$$

$$= 2T\left(\frac{n}{4} \times \frac{1}{2}\right) + (n/4)$$

$$= 2T(n/8) + (n/4) \quad \text{— eqⁿ (3)}$$

Page No.
Date

Replace eqn ② into eqn ①

$= 2(2T(n/4) + n/2) + n$

$= 4T(n/4) + n + n$

$= 4T(n/4) + 2n$ — ④

Now substitute $T(n/4)$ in eqn ④

$= 4(2T(n/8) + (n/4) + n$

$= 8T(n/8) + n + 2n$

$= 8T(n/8) + 3n$

$= 2^3 T(n/2^3) + 3n$

$= 2^4 T(n/2^4) + 4n$

$\vdots$

$= 2^k T(n/2^k) + kn$ ---- ⑤

eliminate this term.

$\therefore$ Let $\boxed{\frac{n}{2^k} = 1}$

$n = 2^k$ take log on both side.

$\log_2 n = \log_2 2^k$

$= \log_2 n \cdot k \log_2 2$

$= \boxed{\log_2 n = k}$

- put value of k in eqn ⑤

$= 2^{\log_2 n} T\left(\frac{n}{2^{\log_2 n}}\right) + n\log_2 n$

$= 2^{\log n} T(1) + n\log n$

$= 2^{\log n} * 1 + \underline{n\log n}$ - max term

$= O(n\log n)$ Time complexity.

Q. $T(n) = \begin{cases} 1 & n = 0 \\ T(n-1) + 1 & n > 0 \end{cases}$

Ans $T(n-1) + 1$ — (1)

Replace $n-1$ by n eqn (1)

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

$T(n-2) + 1$ — (2)

~~$T(n-1)$~~

Replace $n-2$ by n eqn (1)

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

$= T(n-3) + 1$ — (3)

$= T(n-2)$

$T(n-2) = T(n-2)$

Substitute $T(n-2)$ eq (2) in (1)

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

$= T(n-3) + 1) + 1) + 1)$

$= T(n-3) + 3$

$= T(n-4) + 4$

$= T(n-5) + 5$

$\vdots$

$= T(n-k) + k$

$=$ let $n - k = 0$ $\quad n = k$

~~take log on both~~

$= T(n-n) + n$

$= T(0) + n$

$= O(n)$ Time complexity

Q.

```
1. Algorithm add (a, b, c, m, n)                    - 0
2. {                                                - 0
3.     for (i = 1 to m) step step do                - m+1
4.     {                                            - 0
5.         for (j = 1 to n) step step do            - m(n+1)
6.         {
7.             c[i][j] = a[i][j] + b[i][j];         - mn
8.         }
9.     }
10. }
```

Step count.

Ans:-

$= 0 + 0 + m+1 + m(n+1) + nm$

$= m+1 + m(n+1) + mn$

$= m+1 + mn + m + mn$

$= 2mn + 2m + 1$ - Total time complexity.

$= \boxed{O(n)}$