* Analysis of Algorithm.

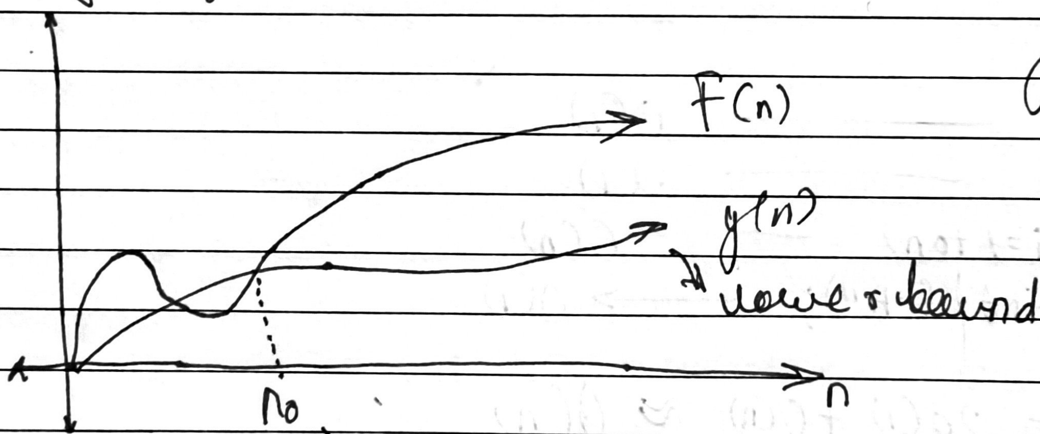
i) Big Oh - O notation.

Where C is constant
$\boxed{C > 0}$

$Cg(n) \longrightarrow$ upper bound function.

$f(n) \longrightarrow$ Running time of an Algorithm.

ii) Big Omga Ω (Best case T.C) $O(n^2)$
$\uparrow$
$g(n)$



(It valid from this)

iii) Theta $\Theta \longrightarrow$ Average Case.
$\downarrow$
Best Case & Worst Case are same.

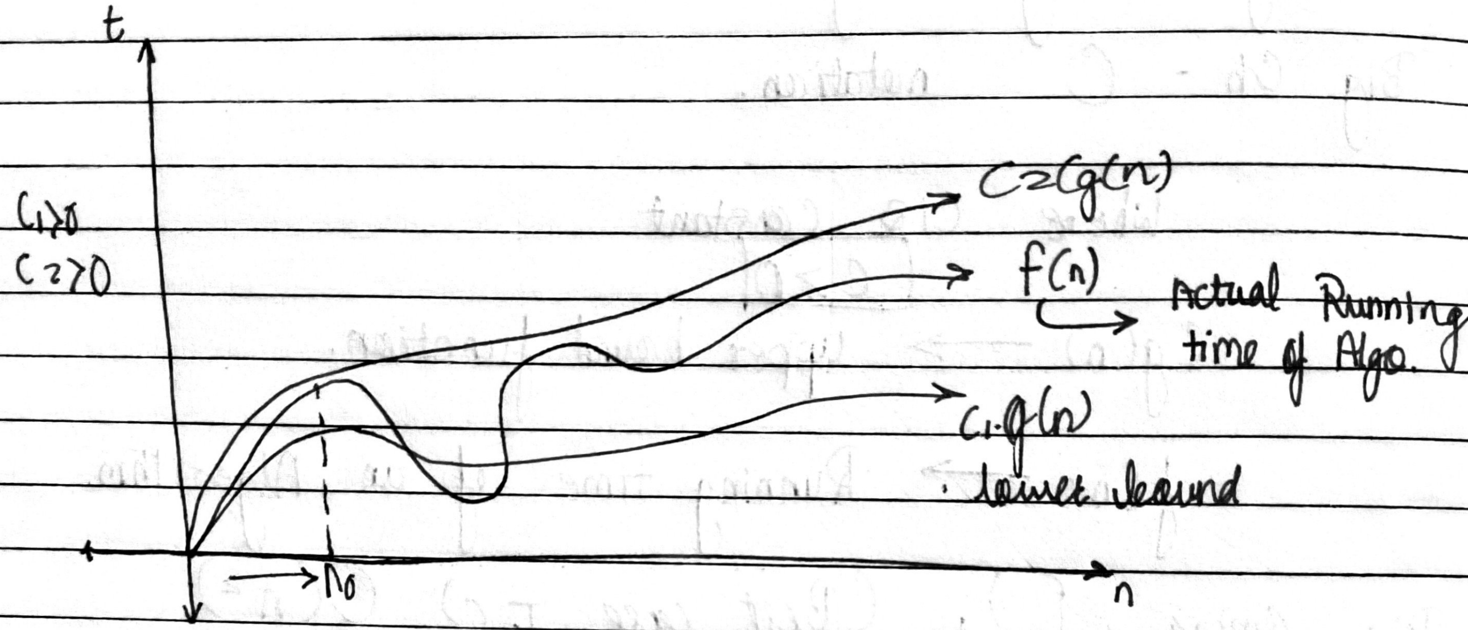
$\Theta(n^2)$

$O(n^2)$ $\Omega(n^2)$

$O(k \cdot n) \approx O(n)$

Page No.

Date / /



ex. A()

{ int i ——— $O(1)$

int i; ——— $O(1)$

for(i=1 to n) ——→ $O(n)$

printf("A"); ——→ $O(1)$

$T.C = 2O(1) + O(n) \approx O(n)$

ex. 2) A()

{ int i,j; ——→ $O(1)$

for (i=1 to n) ——→ $O(n)$

for (j=1 to n) ——→ $O(n)$

printf ("A") ——→ $O(1)$ ←—— $n \times n$

$O(n^2)$

$T.C = O(1) + O(n^2)$

$= O(n^2)$

Page No.

Date / /

ex: 3) A()

```
{ i=1                          → O(1)
  For (i = 1, i*i <= n ; i++)
        print ("A")            → O(√n)
}
```

i = 1

i = 2

⋮

$i^2 <= n$

$i <= \sqrt{n}$

$\sqrt{n}$ times

$\boxed{T.C = O(\sqrt{n})}$

ex. 4)

```
    A()
  { int i;
    int j;
n   for (i = 1; i <= n; i++)          → "n" times
    {
n     for (j = 1; j <= i; j++)        → "i" times
      {
n       for (k = 1; k <= 100, k++)    → "100" times
        {
          print ("A")
    }   }   }
```

un rolling the loop

$i=1$ $j = 1$ time $k = 100$ times	$i=2$ $j = 2$ time $k = 200$ times	-------	$i = n$ $j = n$ times $k = n \times 100$ times

$$T.C = 100 + 2(100) + 3(100) + \cdots n(100)$$

$$T.C = 100 \cdot n(n+1)/2$$

$$T.C = 50n^2 + 50n$$

$$TC \approx n^2 + n$$

Note: $f(n) = n^k + n^{k-1} + \cdots$

$$= O(n^k)$$

Que. A()

```
{ for (i=1; i<n; i = i * 2)
  { print ("A.A")
  } }
```

$\rightarrow$ $i = 1, 2, 4, 8, 16, \cdots$ n terminate pt.

$2^0, 2^1, 2^2, 2^3, \cdots$ 2^k Loop ~~terminate~~

$i = n \rightarrow$ ter. pt.

$$n = 2^k$$

$$\log_2 n = \log_2 2^k.$$

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

$$k = \log n$$

$$\boxed{\approx O(\log_2 n)}$$

Page No.______

Date ___/___/___

Que. A()

{

while (n>1)

{ n = n/2

} }

Assume $n \geq 2$

$n=100$	$n=50$	$n=25$
$A=\frac{100}{2}$	$A=\frac{50}{2}$	$A=\frac{25}{2}$
$\boxed{A=50}$	$\boxed{A=25}$	$\boxed{A=12.5}$

no. of ile slops = 4

for (n = 100^k ; n > 1; n = n/2)

  for (n = 1 ; n <= k; n = n×2)

2) Recursive Functions :

A(n) ⟶ T(n) → Time f^n

{ if() —— O(1) ⟶ O(1)

{ yetuch A(n/2) + A(n/2) ⟶ 2T(n/2)

}

$\boxed{T.C = T(n) = O(1) + 2T\left(\frac{n}{2}\right)}$ Rec. eqn:

·) A(n) ⟶ T(n) → Time f^n.

{ if(n>1) ⟶ O(1)

  return (A(n-1)) ⟶ T(n-1)

T.C = T(n) = O(1) + T(n-1) : n > 1

// R. car.

Base case (B.C)

Ti(n) = O(1) $\boxed{T(n) = 1}$; n = 1

// B.C.

Page No.____
Date __/__/__

* Back - Substitution Method :-

$$\boxed{T(n) = T(n-1) + 1} \quad ; n > 1 \quad \text{--- } ①$$
$$T(n) = 1 \quad ; n = 1$$

- $\boxed{\text{For : } T(n-1) = T(n-2) + 1}$ — ②

$(n-1-1)$

- For $T(n-2) = T(n-3) + 1$ — ③

Eq^n ① becomes

in eqn① put ~~T(n)=~~

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

sub①- $T(n) = T(n-2) + 2$

sub⓪ $T(n) = [T(n-3) + 1] + 2$

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

⋮

After Substitution 'K' times.

$$\boxed{T(n) = T(n-k) + k}$$

T(1)

$n-k = 1$ ⟶ Ter. Pt. /B.C

$n-k = 1$ ⟹ $\boxed{k = n-1}$

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

Que) eqn:-

$$\boxed{T(n) = T(n-1) + n} \text{ ——— (1)} \quad ; n > 1$$

$$T(n) = 1 \quad ; n = 1.$$

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

$= T(n-2) + (n-1)$ ——— (2)

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

$= T(n-3) + (n-2)$ ——— (3)

eqn (1) becomes (after back-Substitution Method) (2)

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

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

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

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

After 'k' substitution:-

$$T(n) = T(n-k) + kn - k.$$

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

terminates when $T(n-k)$ becomes $T(1)$

and $n-k$ becomes 1

$$n - k = 1$$

$$\boxed{k = n - 1}$$

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

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

$$\boxed{T(n) = O(n^2)}$$

Page No.

Date / /

$$T(n) = 2T\left(\frac{n}{2}\right) + n \quad ; n > 1$$
$$T(n) = 1 \quad ; n = 1$$

$$T\left(\frac{n}{2}\right) = 2T\left(\frac{n}{2^2}\right) + \frac{n}{2} \quad \text{—— (2)}$$

$$T\left(\frac{n}{2^2}\right) = 2T\left(\frac{n}{2^2}\right) + \frac{n}{2^2} \quad \text{—— (3)}$$

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

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

$$= 2^2\left[2T\left(\frac{n}{2^3}\right) + \frac{n}{2^2}\right] + n + n$$

$$T(n) = 2^3 T\left(\frac{n}{2^3}\right) + n + n + n$$

After 'k' Substitutions

$$\boxed{T(n) = 2^k \left(T\left(\frac{n}{2^k}\right) + k \cdot n\right.}$$

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

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

$$\boxed{T(n) = O(n \log n)}$$

Page No.

Date / /

★ Master's Theorem :

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + \Theta\left(n^k \log^p n\right)$$

$a \geq 1$, $b > 1$, $k \geq 0$ and P is real no)

case's

i) if $a > b^k$ then $T(n) = \Theta(n^{\log_b a})$

2) if $a = b^k$

a) if $P > -1$, then $T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

b) if $P = -1$ then $T(n) = \Theta(n^{\log_b a} \log \log n)$

c) if $P < -1$, then $T(n) = \Theta(n^{\log_b a})$

3) if $a < b^k$

a) if $P \geq 0$, then $T(n) = \Theta(n^k \log^p n)$

b) if $P < 0$, then $T(n) = O(n^k)$

Page No.____

Date __/__/__

☆ linear Search:

```
LSearch (Arr, N, X)
{ ... i = 0, found = 0;
  while (i < N)
  {
      if Arr[i] = = X
      {
          print(i)
          found = 1;
          i++
      }
      if (found == 0)
          print("Not found") }
```

→

Arr	2	4	8	5	0	-1	2	9	5	10
	0	1	2	3	4	5	6	7	8	9

B.C = No. of Steps $i = N-1$

B.C = Ω (N)

W.C = No. of Steps. $= N-1$

$= O(N)$

Average case $= O(N)$

Page No.____
Date __/__/__

* Linear Search on Sorted Array:

```
Lsearch(Arr /N /X)
{  i = 0, found = 0,
   while (i<N && A[i] <K)
   { if Arr[i] = X
     {
        print (i)
        found = 1;
        i++ }
     if (found == 0)
        print "Not found" }
```

```
i=0
{
 i=0, found = 0
 { while (i = N
     if (A(i) <N)
     else Arr(i) = N.
 }
```

B.C = No. of Steps = 1
T.C = $\Omega(1)$
W.C = No. of Steps = N - 1
T.C = O(N) [No. of Steps <= (N-1)]
→ for sorted array.

* Binary Search:

```
recurs. b.search (Arr /N/ X/ lo /hi)
{
   mid, retval;
   if (low > high)
      retval = -1
   else
   {  mid = [(lo+hi)/2]
      if (A[mid] == X
      { retval = mid
      } }
```

```
elseif (A[mid] > x)
{ retual = rec-b.search(Arr N/ X/ CO, mid-1)
}
else
{ retival = rec-bsearch (Arr, N1, X, mid+1/n)
}
return retrivial.
}
```

★ Re-cursive Binary Search. (A[ ], low, high, X)

```
{
    int retval, mid;
    mid = (low + high) / 2;
O(1)   if (A[mid] == X)
            retual = mid;
O(1)   else if (A[mid] < X) // low = mid+1, high = same.
       retual = recursive binary search(A[], mid+1, high, X)

T(n/2) else if (A[mid] > X) // low = same, high mid-1.
       retual = recursive binary search(A, low, mid-1, X).

       else if (low > high) // crossover of array endpoint
        retual = -1

       return retval;
```

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

ex. Arr

10	20	30	40	45	50	55
0	1	2	3	4	5	6

rbs(A, 0, 6, 450) $\Rightarrow$

Time complexity $T(n)$

T.C $\Rightarrow$

$$T(n) = T\left(\frac{n}{2}\right) + O(1) \quad \text{— (A)}$$

low = 4, high = 6, mid = 5	return = 5.
low = 0, high = 6, mid = 3.	return → rbs (A, 4, 6, 450)

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

$$T(n) = \left[T\left(\frac{n}{4}\right) + 1\right] + 1 \quad \text{— Ist substitution.}$$

$$T(n) = T\left(\frac{n}{4}\right) + 2 \quad \text{— B.} = T\left(\frac{n}{2^2}\right) + 2$$

$$T(n) = \left[T\left(\frac{n}{8}\right) + 1\right] + 2 \quad \text{— 2nd Substitution}$$

$$T(n) = T(n/8) + 3 \quad \text{— C.}$$

$\vdots$

$$T(n) = T(n/2^k) + k \quad \text{— after } k^{th} \text{ substitution.}$$

Base case $\Rightarrow T(n) = 1 \; ; n = 1.$

If $T\left(\frac{n}{2^k}\right) = T(1) \Rightarrow \frac{n}{2^k} = 1.$

$$n = 2^k \Rightarrow \log_2 n = \log_2 2^k$$

$$K = \log_2 n. \Rightarrow B \Rightarrow T(n) = 1 + \log n.$$

$$T(n) = O(\log n).$$

★ Tertiary Search.

```
{
    int retual, mid1, mid2;
    mid1= (low+high)/3 ;
    mid2=2(low + high)/3;
     if (A[mid1] == x)
          retual= mid1;
    elseif (A[mid2] == x)
          retual= mid 2;
    elseif (A[mid1] > x)
          retual = rbs(A, low, mid1-1, x);
    elseif( A[mid1] < x && A[mid2] > x)
          retual =rbs(A, mid1+1, mid2-1, x);
    elseif( A[mid2] < x)
          retual =rbs(A, mid2+1, high, x);
    else if( low > high)
          retual= -1;

    return retval.
```

★. Trees.

1} Tree is a collection of vertices,(V), set of Edges (E).

2} Tree is a connected graph.

3} Their are no cycles, it is acyclic.

4} When we have a tree where every node has K or less than k children.
Then it is called k-array tree.

Page No.

Date / /

5. Maximum number of children it can have is 2 childrens, but leaf node have 0 children.
6. Root node doesn't have any incoming edge.
7. Leaf node doesn't have any outgoing edge.
8. If maximum level in the binary tree is k then total number of nodes that can be present equals to $2^{k+1} - 1$.
9. Height is a largest level possible in the given tree.

★ Recurssive definition of a tree
Tree is a root node + left sub tree + Right sub tree, where LST & RST will be the trees by themselves.

★ Implementation of tree.

```
Struct tree node
{
  int data
  tree nod * Lptr * rptr
 } TN ;
```

Que. Write a function to count no. of nodes in binary trees.

→ $N(T) = 1 + N(LST) + N(RST)$

```
NT (root) {
 if ( root -> lptr == NULL, root -> rptr == NULL)
      "No.of node = 1".
 else
      " no.of node = 1+ NT(root -> lptr) + NT(root -> rptr).
```