

## \* Analysis of Algorithm.

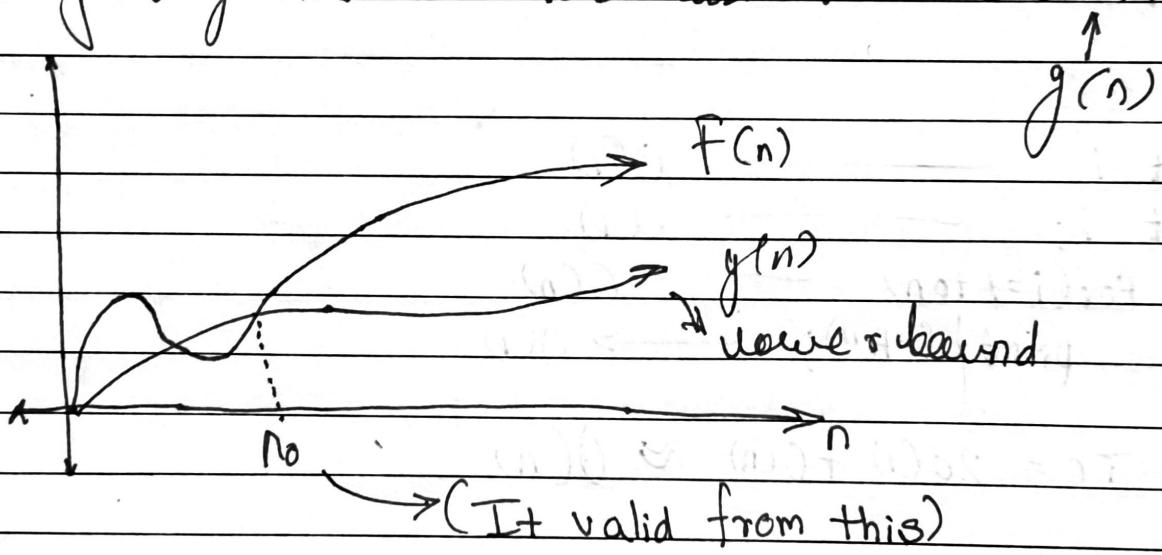
i) Big Oh - O notation.

where  $C$  is constant  
 $(C > 0)$

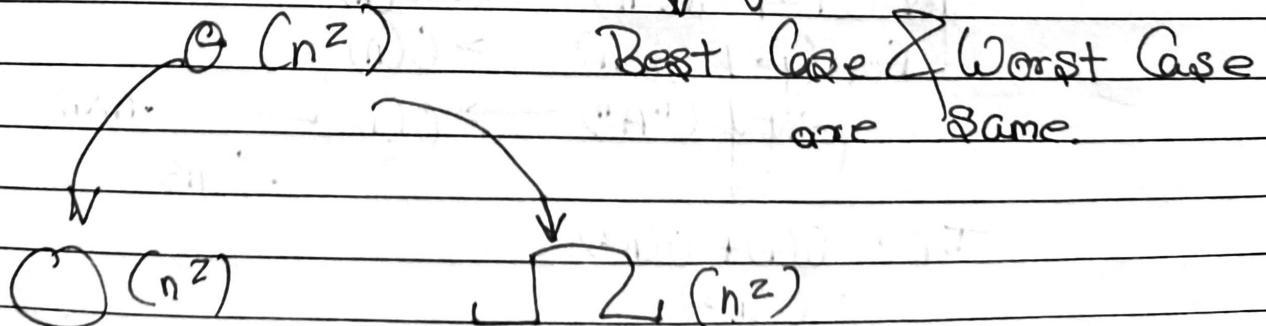
$Cg(n) \rightarrow$  upper bound function.

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

ii) Big Omega  $\Omega$ , (Best case T.C)  $O(n^2)$



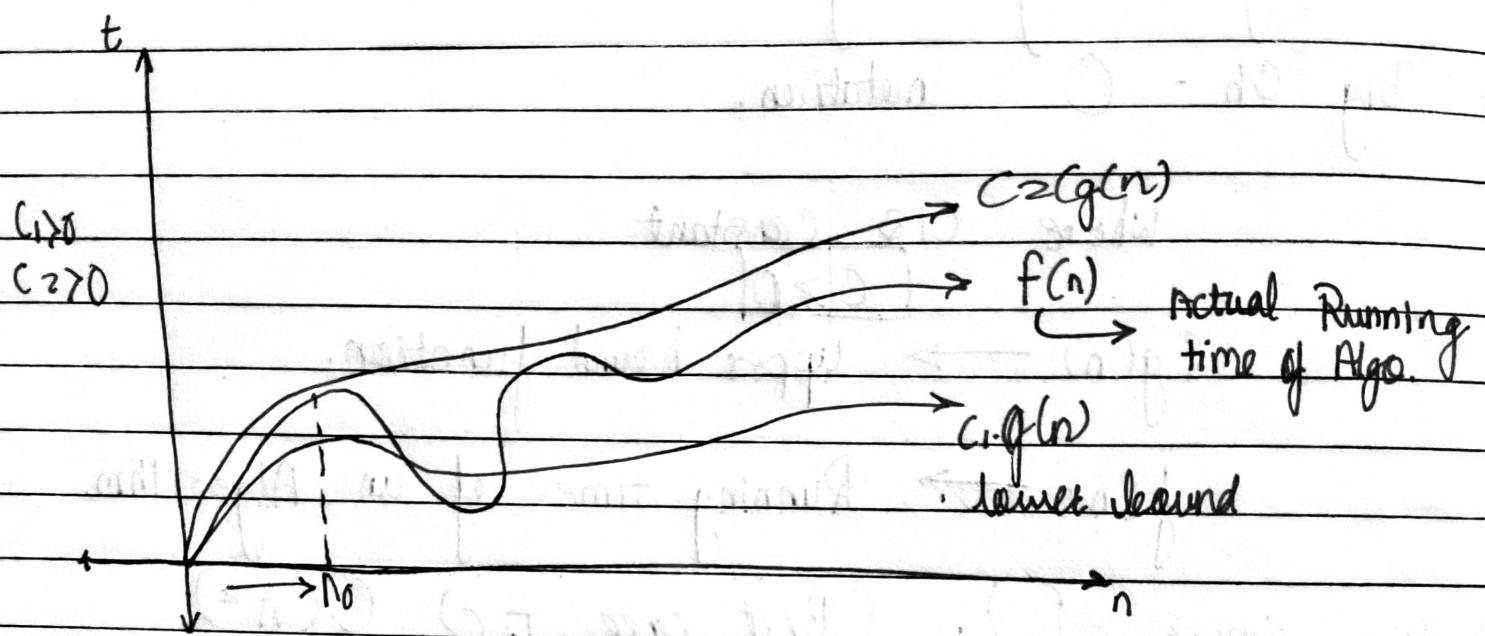
iii) Theta  $\Theta \rightarrow$  Average Case.



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

Page No. \_\_\_\_\_

Date / /



Ex. 1) 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) ← nxn
  
```

$$\begin{aligned}
 T.C &= O(1) + O(n^2) \\
 &= O(n^2)
 \end{aligned}$$

Ex: 3) A()

{  $i = 1 \longrightarrow O(1)$ For ( $i = 1, i * i \leq n; i++$ )print("A")  $\longrightarrow O(\sqrt{n})$ 

{

 $i = 1$   
 $i = 2$  $\vdots$  $i^2 \leq n$  $i \leq \sqrt{n}$  $\sqrt{n}$  timesT.C =  $O(\sqrt{n})$ 

Ex: 4) A()

{ int i;

int j;

for ( $i = 1; i \leq n; i++$ ) }  $\rightarrow "n"$  times{ for ( $j = 1; j \leq i; j++$ ) }  $\rightarrow "i"$  timesfor ( $k = 1; k \leq 100; k++$ ) }  $\rightarrow "100"$  times

↓ print("A")

{ } { }

unrolling the loop

$i=1$	$i=2$	$\dots$	$i=n$
$j = 1 \text{ time}$	$j = 2 \text{ times}$	$\dots$	$j = n \text{ times}$
$k = 100 \text{ times}$	$k = 200 \text{ times}$	$\dots$	$k = n \times 100 \text{ times}$

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

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

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

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

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

Ques A()

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

$\rightarrow i = 1, 2, 4, 8, 16, \dots$   $\Rightarrow$  terminate pt.  
 $2^0, 2^1, 2^2, 2^3, \dots$  loop terminates

$$n = 2^k$$

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

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

$$k = \log n$$

$$\approx O(\log n)$$

Ques.  $A()$

$\left\{ \begin{array}{l} \text{while } (n > 1) \\ \quad \left\{ \begin{array}{l} n = n/2 \\ \quad \end{array} \right. \\ \end{array} \right.$

Assume  $n \geq 2$

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

no. of iterations = 4

for ( $n = 100$ ;  $n > 1$ ;  $n = n/2$ )

for ( $n = 1$ ;  $n \leq k$ ;  $n = n \times 2$ )

## 2) Recursive Functions:

$$\left\{ \begin{array}{l} A(n) \\ \quad \text{if } () \rightarrow O(1) \\ \quad \sum \text{ yet each } A\left(\frac{n}{2}\right) + A\left(\frac{n}{2}\right) \end{array} \right. \rightarrow T(n) \rightarrow \text{Time } f^n$$

$$T.C = T(n) = O(1) + 2T\left(\frac{n}{2}\right) \quad \text{Ref. eqn.}$$

$$\left\{ \begin{array}{l} A(n) \\ \quad \text{if } (n > 1) \\ \quad \text{return } (A(n-1)) \end{array} \right. \rightarrow T(n) \rightarrow \text{Time } f^n$$

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

|| R. c. r.

Base case (B.C.)

$$T(n) = O(1) \quad [T(n) = 1] ; n = 1$$

|| B.C.

\* Back - Substitution Method :-

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

$$T(n) = I \quad ; n = I$$

• For :  $T(n-1) = T(n-2) + I$  --- (2)

$(n-1-1)$

• For  $T(n-2) = T(n-3) + I$  --- (3)

Eg.  $n$  (1) becomes  
in eqn (1) put  $T(n) =$

$$\text{sub (1)} - T(n) = [T(n-2) + I] + I$$

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

$$\text{sub (2)} \quad T(n) = [T(n-3) + I] + 2$$

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

After Substitution 'k' times.

∴  $T(n) = T(n-k) + k$

$$n-k = I \rightarrow \text{Ter. Pt. (B)}$$

$$n-k = I \Rightarrow k = n-I$$

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

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

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

Ques > Eg :-

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

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

$$\begin{aligned} \text{For } T(n-1) &= T(n-1-1) + (n-1) \\ &= T(n-2) + (n-1) \end{aligned} \quad (2)$$

$$\begin{aligned} \text{For } T(n-2) &= T(n-2-1) + n \\ &= T(n-3) + (n-2) \end{aligned} \quad (3)$$

Eg (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:-

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

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

dominates when  $T(n-k)$  becomes  $T(1)$   
and  $n-k$  becomes  $I$

$$n-k = I$$

$$k = n - I$$

$$T(n) = T(0) + (n-1)(n-1)$$

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

$$\boxed{T(0) = O(n^2)}$$

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

$$T(n) = 1 ; n = 1$$

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

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

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

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

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

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

After 'k' Substitutions

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

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

$$n = 2^k$$

$$k = \log n$$

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

$$= n^{\log 2} + n \log n$$

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

\* Master's Theorem :

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

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

case's

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

ii) 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  $T(n) = \Theta(n \log_a n)$

iii) if  $a < b^k$

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

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

## ★ 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	18	5	10	-1	2	9	15	10
	0	1	2	3	4	5	6	7	8	9

B.C = No. of Steps      i. = N - 1

B.C = O(N)

W.C = No. of Steps. = N - 1  
= O(N)

Average Case = O(N)

## \* Linear Search on Sorted Array :

```

LSearch(ARR/N/X) {
    { i = 0, found = 0,
      while (i < N && A[i] <= X) } { i = 0, found = 0
        { if (ARR[i] == X) } { while (i = N)
          print(i) } { if (found == 0) } { else ARR(i) = N.
          found = 1;
          i++ } { print "Not Found" } }
    }
}

```

$B.C = \text{No. of Steps} = 1$

$T.C = O(1)$

$W.C = \text{No. of Steps} = N - 1$

$T.C = O(N)$  [No. of Steps  $\leq (N-1)$ ]  $\rightarrow$  for sorted array.

## \* Binary Search:

recurs. b.search (ARR/N/X/A[ci])

```

    { mid, result;
      if (low > high)
        result = -1
      else
    }

```

```

    { mid = [(low + high) / 2]
      if (A[mid] == X)
        result = mid
    }

```

else if ( $A[\text{mid}] > x$ )

{  $\text{return} = \text{rec\_b\_search}(A, N/x, 0, \text{mid}-1)$

}

else

{  $\text{return} = \text{rec\_b\_search}(A, N, x, \text{mid}+1)$

}

return  $\text{return}$ .

}

\* Re-cursive Binary Search. ( $A[], \text{low}, \text{high}, x$ )

{

int  $\text{return}, \text{mid};$

$\text{mid} = (\text{low} + \text{high}) / 2;$

O(1)

if ( $A[\text{mid}] == x$ )

$\text{return} = \text{mid};$

O(1) Use if ( $A[\text{mid}] < x$ ) //  $\text{low} = \text{mid} + 1$ ,  $\text{high} = \text{same}$ .

$\text{return} = \text{recursive binary search}(A, \text{mid} + 1, \text{high}, x)$

T( $\frac{n}{2}$ ) Use if ( $A[\text{mid}] > x$ ) //  $\text{low} = \text{same}$ ,  $\text{high} = \text{mid} - 1$ .  
 $\text{return} = \text{recursive binary search}(A, \text{low}, \text{mid} - 1, x).$

else if ( $\text{low} \geq \text{high}$ ) // crossover of array endpoints  
 $\text{return} = -1$

return  $\text{return};$

Ex. Arr [ 10 | 20 | 30 | 40 | 45 | 50 | 55 ]  
 0 1 2 3 4 5 6

rbS( A, 0, 6, 45 ) >

Time complexity T(n)

T.C.  $\Rightarrow$

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

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

low = 4  
high = 6  
mid = 5.

low = 0  
high = 6  
mid = 3.

$$T(n) = \left\lceil T\left(\frac{n}{4}\right) + 1 \right\rceil + 1 \quad \text{--- 1st substitution.}$$

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

$$T(n) = \left\lceil T\left(\frac{n}{8}\right) + 3 \right\rceil \quad \text{--- 2nd substitution}$$

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

:

$$T(n) = T\left(\frac{n}{2^k}\right) + k \quad \text{--- after } k^{\text{th}} \text{ substitution.}$$

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

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

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

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

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

## \* Tertiary Search.

```
int retual, mid1, mid2;  
mid1 = (low + high) / 2; // 3 ;  
mid2 = 2 * (low + high) / 3 ;  
if (A[mid1] == x)  
    retual = mid1;  
elseif (A[mid2] == x)  
    retual = mid2;  
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);  
elseif (low > high)  
    retual = -1;
```

return retual.