In the programming (computer) world there are many problem and we may solution regarding to that problem. We have to follow some steps.

Criteria :-

1. Input :- Zero or more quantities are externally supplied.
2. Output :- Atleast one quantity is produced.
3. Definiteness :- Each instruction is clear and unambiguous.
4. Finiteness :- If we trace out the instructions of an algorithm, then for all cases, the algo terminates after a finite number of steps
5. Effectiveness :- Every instruction must be very basic so that it can be carried out in principle by a person using only pencil and paper.

Criteria for evaluating algorithm -

i) Time should be less.
ii) Space should be less.

In order to evaluate time & space complexity we use asymptotic notation.

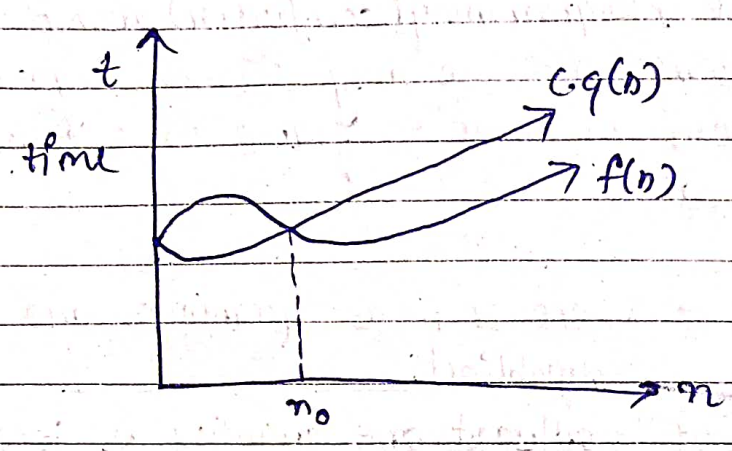
Asymptotic Notation -

1) Best Case
2) Average Case
3) Worst Case

We use asymptotic notation to evaluate the algorithm.

Date :
Page :

1) Big O, upper bound, Worst Case



iff $f(n) \leq c \cdot g(n)$

$n \geq n_0$

$n_0 \geq 1$

$$f(n) = O(g(n))$$

Computer Program :-

Algorithm :-

21/07/23

Asymptotic Notation :-

1) Big O (oh) (upper Bound or worst case)

Eg:- $f(n) = 3n+2$, $g(n) = n$

Soln:- $f(n) = O(g(n))$

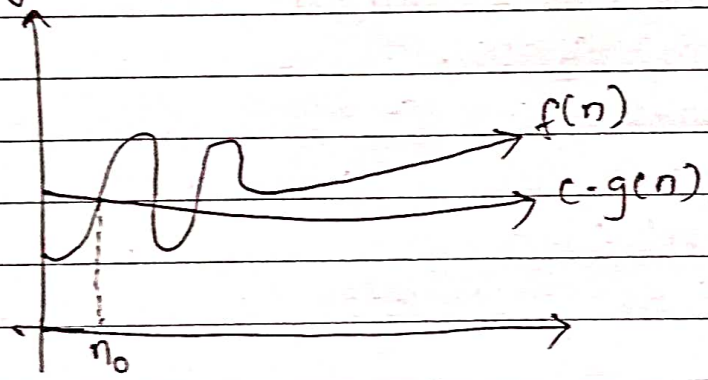
$f(n) \leq c \cdot g(n)$

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

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

$n \geq 2$

2) Big Omega Ω (Best Case, lower bound)



$f(n) \geq c \cdot g(n)$

$n \geq n_0$

$c > 0, n_0 \geq 1$

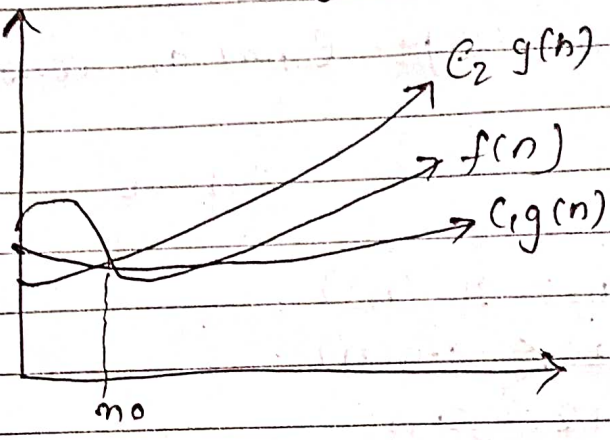
Ques:- $f(n) = 3n+2$, $g(n) = n$

Soln:- $f(n) = \Omega g(n)$

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

$c = 1$

3) Big Theta (Θ) (Average Case)



$f(n) = \Theta(g(n))$

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

c_1 & c_2 are constant > 0

$n \geq n_0$, $n_0 \geq 1$

Ques:- $f(n) = 3n+2$, $g(n) = n$

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

$3n+2 \leq 4n$ $\boxed{c_2 = 2}$

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

$3n+2 \geq 1 \cdot n$

$\boxed{c_1 = 1}$

Eg:-

5	7	1	2	4	10	20	4

1) Best Case $x = 5$

2) Average case $x = 1 + n/2 = n$

3) Worst Case $x = n$

Types of Algorithm :-

Iterative | Recursion

e.g.
```
A()
{ int i;
for (i=1 to n){
    printf("MANIT")
}
```

T.C. → O(n)

```
A()
{ int i,j;
  for (i=1 to n)
     for (i=1 to n)
        print("Hello");
}
```

T.C. → O(n²)

```
A()                    S => 1 3 6 10 — —
{ i=1, S=1             i = 1 2 3 — — k
  while ( S<=n)
     i++;              k(k+1)/2 ≥ n
     S=S+i;            k²+k/2 ≥ n
     printf("Hello");
}
```

T.C. → ~~O(log n)~~ O(√n)

i=2 S=1+2=3

i=3 S=3+3=6

i=4 S=6+4=10

k² ≥ n

k ≥ Ω(√n)

k = O(√n)

```
A()
int i, j, k, n;
for (i=1 to n)
    for (j = 1 to i)
        for (k=1; k<=100; k=k+1)
            printf("Hello");
}
}
}
```

i=1	i=2	i=3	---
j=1	j=2	j=3
K=1x100	K=2x100	K=3x100

$100 + 2\times100 + 3\times100$ ---

$100(1+2+3 \cdots n)$

$\frac{100(n(n+1))}{2} \Rightarrow O(n^2)$

```
A()
{
int i, j, k, n;
for(i=1; i<=n; i++)
{
    for(j=1; j<=i^2; j++)
    {
        for(k=1; k<=n/2; k=k+1)
        {
            printf("Hello");
        }
    }
}
```

$i=1$	$i=2$	$i=3$	$i=n$
$j=1$	$j=4$	$j=9$	$j=n^2$
$k=n/2$	$k=n/2^4$	$k=9\times\frac{n}{2}$	$k=n^2\times\frac{n}{2}$

$$T.C=\frac{n}{2}\left(1+4+9+\cdots n^2\right)$$

$$T.C.=\frac{n}{2}\left(1^2+2^2+3^2+\cdots n^2\right)$$

$$T.C.=\frac{n}{2}\,\frac{n(n+1)(n+2)}{6}$$

$$T.C.=\underline{n^2(n^2-\cdots)}$$

$$T.C=O(n^4)$$

Ques:-
```
A()
{
    for(i=1; i<n; i=i*2)
        printf("Hello");
}
```

$i=1,2,4\cdots n$

$i=1,2^1,2^2,2^3\cdots 2^k$

$2^k=n$

$\log 2^k=\log n$

$k\log 2=\log n \Rightarrow k=(\log_2 n)$

$T.C=O(\log_2 n)$

```
A()
{
    int i,j,k;
    for(i=n/2; i<=n; i++)
        for(j=1; j<=n/2; j++)
            for(k=1; k<=n; k=k*2)
                printf("Hello");
}
```

~~i = n/2~~ ~~i = n/2~~

~~j = n/2~~ ~~j = n/2~~

~~k = 2^n x n/2 x n/2~~ $k = 2^n \times \frac{n}{2} \times \frac{n}{2}$

$T.C = 2^n \times \frac{n}{2} \times \frac{n}{2}$

$T.C = O(n^2 \log_2 n)$

$i = 1$	$i = 2$	--	$i = n$
$j = n/2$	$j = 2 \times \frac{n}{2}$	--	$j = n \times \frac{n}{2}$
$k = 2 \times \frac{n}{2}$	$k = 2^2 \times \frac{n}{2}$	--	$k = 2^n \times n \times \frac{n}{2}$

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

3/

condition:-

Ques:- $T(n) = aT(n/b) + \theta(n^k \log^p n)$

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

1) If $a > b^k$, then $T(n) = \theta(n^{\log_b a})$

2) If $a = b^k$ $\theta(n^{\log_b a} \log^{p+1} n)$
   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 $T(n) = \theta(n^k)$.

Master theorem is used to find time complexity of recursive algorithm.

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

On comp. $a = 3, b = 2, k = 2, p = 0$

$3^k \not< 2^2$

$T(n) = \theta(n^2 \log^0 n)$

$T(n) = \theta(n^2)$

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

Soln:- On comp. $a = 4, b = 2, k = 2, p = 0$

2(a) $\theta(n^{\log_2 4} \log^2 n)$

$\theta(n^2 \log^2 n)$ Ans.

Ques:- $T(n) = T(n/2) + n^2$

Soln:- $a = 1,\ b = 2,\ k = 2,\ p = 0$

$b^k = 4$

$a < b^k$

$T(n) = \theta(n^k \log^p n)$

$T(n) = \theta(n^2 \log^0 n)$

$T(n) = \theta(n^2)$ Ans

Ques:- $T(n) = 2^n T(n/2) + n^2$

Soln $a = 2^n,\ b = 2,\ k = 2,\ p = 0$

$2^n \quad b^k = 4$

~~if n~~ We cannot solve it using master's theorem.

Ques:- $T(n) = 16\,T(n/4) + n$

Soln:- $a = 16,\ b = 4,\ k = 1,\ p = 0$

$16 \quad , \quad 4$

$a > b^k$

$T(n) = \theta(n^{\log_4 16})$

$T(n) = \theta(n^2)$ Ans.

Ques:- $T(n) = 2T(n/2) + n \log n$

Soln $a = 2,\ b = 2,\ k = 1,\ p = 1$

$2 \quad 2$

$a = b^k$

$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$

$T(n) = \Theta(n^{\log_2 2} \log^2 n)$

~~$T(n) = \Theta(n^{\log^2 n})$ Ans.~~

$T(n) = \Theta(n \log^2 n)$

Ques:- $T(n) = 2T(n/2) + \frac{n}{\log n}$

Soln:- $a = 2, \; b = 2, \; k = 1, \; p = -1$

$T(n) = \Theta(n^{\log_b a} \log\log n)$

$T(n) = \Theta(n^{\log_2 2} \log\log n)$

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

Ques:- $T(n) = 2T(n/4) + n^{0.51}$

Soln $a = 2, b = 4, k = 0.51, p = 0$

$a = 2, \; b^k = 4^{0.51}$

~~$a = 2$~~

~~$T(n) = \Theta(n^{\log_b a} \log^{p+1} n)$~~

Case 3:- $\Theta(n^k \log^p n)$

$= \Theta(n^{0.51})$ Ans.

Ques:- $T(n) = 6T(n/3) + n^2 \log n$

Sol $a = 6, \; b = 3, \; k = 2, \; p = 1$

$a < b^k$

$T(n) = \Theta(n^2 \log n)$ Ans.

Ques:- $T(n) = 0.5T(n/2) + \frac{1}{n}$

$a \neq 1$ (That's why master's theorem is not applicable).

Ques:-

Space Complexity :-

The space which is used at execution time known as ~~of~~ Extra space.

The extra space taking up the program known as space complexity.

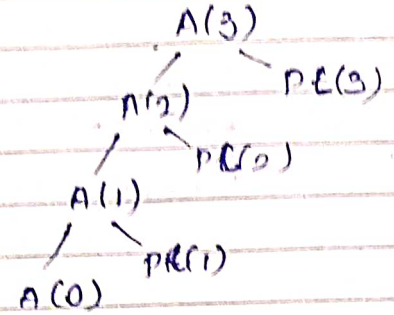
Ans
```
Algo (A,n)                    S.C. → O(n)
{
  int i;
  create B[n];
  for(i=1 to n)
      B[i] = A[i];
}
```

Ques
```
Algo(A,n){
    create B(n,n)
    int i,j;                  S.C -> O(n^2)
    for(i=1 to n)
       for(j to n)
          B[i,j] = A[j];
}
```

Space complexity of Recursive function :-

Ques :-

```
A(n)
{
  if(n>1)
  {
    A(n-1)
    printf(n)
  }
}
```



Total no. of recursive calls = (n+1)

Let say k is space taken by each recursive call

Total space taken by recursive call = (n+1)k .

= O(nk)

= O(n) ~~k~~.

Time complexity = O(n)

$T(n) = k + T(n-1)$ —— (1)

$T(n-1) = k + T(n-2)$ —— (2)

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

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

$T(n-2) = 2k + T(n-2)$ —— (3)

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

$T(1) = nk + T(1)$

$n-k = 0$

$n = k$

Time complexity = nk

$= O(n)$

$T(n-n) + n$

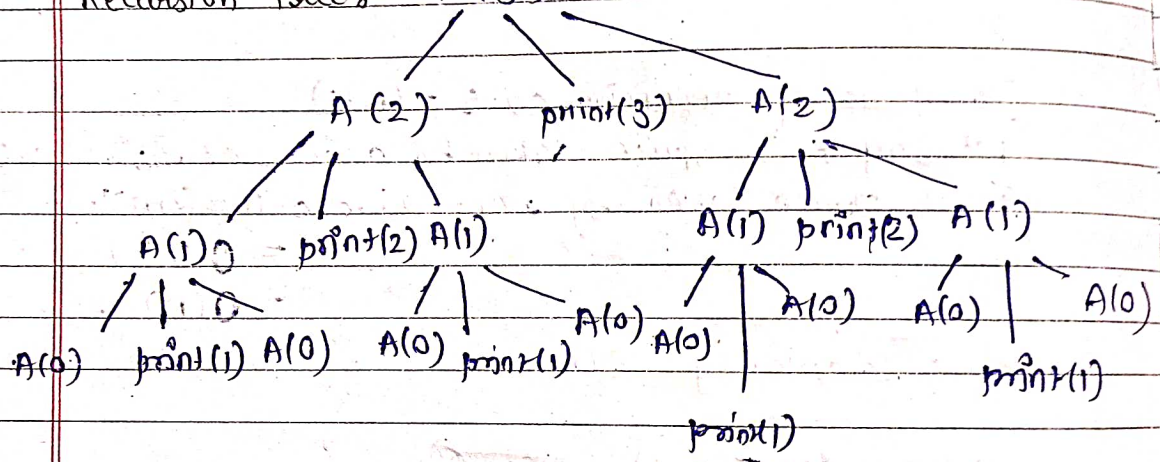
$= T(0) + n$

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

Eg:- A(n) for n=3

```
{
  if(n≥1){
    A(n-1)
    print(n)
    A(n-1)
  }
}
```

Recursion Tree:-



Recursive Algorithm :- Time complexity of recursive algo can found in three ways

1) Back Substitution
2) Recursive Tree method
3) Master's Theorem

```
A(n)
{
  if(n>1)
    return (A(n-1))
}
```

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

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

1) Back Substitution :-

$T(n) = 1 + T(n-1)$ ——— (1)

$n = n-1$

$T(n-1) = 1 + T(n-2)$ ——— (2)

$T(n-2) = 1 + T(n-3)$ ——— (3)

Put (2) in (1)

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

$T(n) = 2 + T(n-2)$ ——— (4)

Put (3) in (4)

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

$= 3 + T(n-3)$

$= k + T(n-k)$

To stop $n-k = 1$

$= k + T(n-k)$

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

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

$= n-1+1$

$T(n) = O(n)$

Ques:- $T(n) = n + T(n-1), n > 1$

$T(n) = 1, \quad n = 1$

Soln:- For $n = n-1$

$T(n-1) = (n-1) + T(n-2)$ ——— (2)

$T(n-2) = (n-2) + T(n-3)$ ——— (3)

Put (2) in (1)

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

$T(n) = 2n - 1 + (n-2)$ ——— (4)

Put (3) in (4)

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

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

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

Terminal Condition

To stop $K = n-1$

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

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

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

$T(n) = O(n^2)$ Ans :

Terminal Condition $n-(k+1) = 1$

$n-k-1 = 1$

$k = n-2$

Put $k = n-2$ in (4)

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

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

$= \frac{n(n+1)}{2} \approx O(n^2)$.

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

$T(n-2) = (n-2)+T(n-3)$

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

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

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

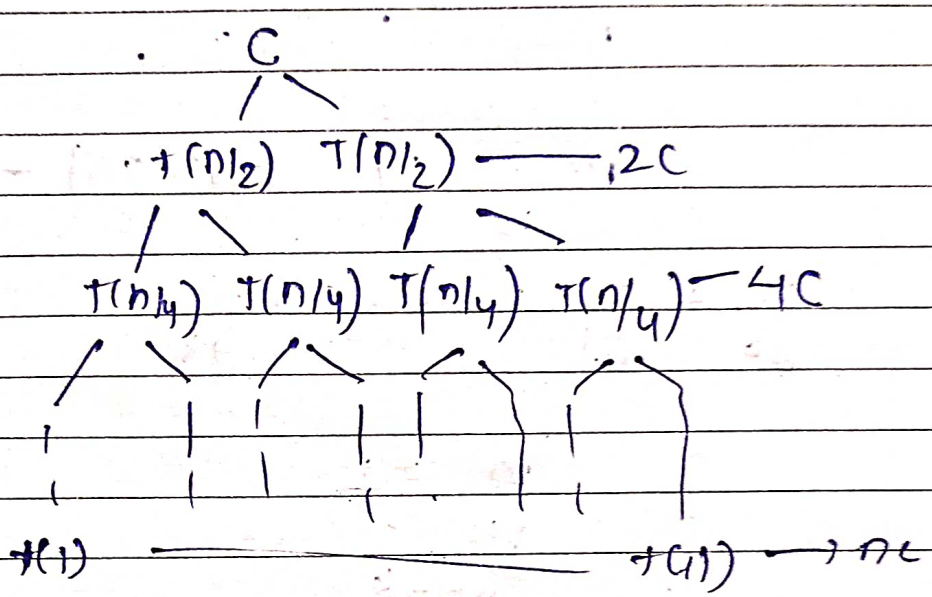
$= n+(n-1)+(n-2)+\cdots+(n-K)+(n-(K+1))$

2) Recursive Tree Method:-

Eg:- $T(n) = 2T\left(\frac{n}{2}\right) + c, \quad n > 1$

$T(n) = c, \quad n = 1$

Sol:



$c + 2c + 4c + 8c - - nc$

$c(1 + 2 + 4 + 8 - - n)$

Substitute $n = 2^k$

$c(2^0 + 2^1 + 2^2 - - - 2^k)$ GP series

$= c\left(\frac{1(2^{k+1} - 1)}{2 - 1}\right)$

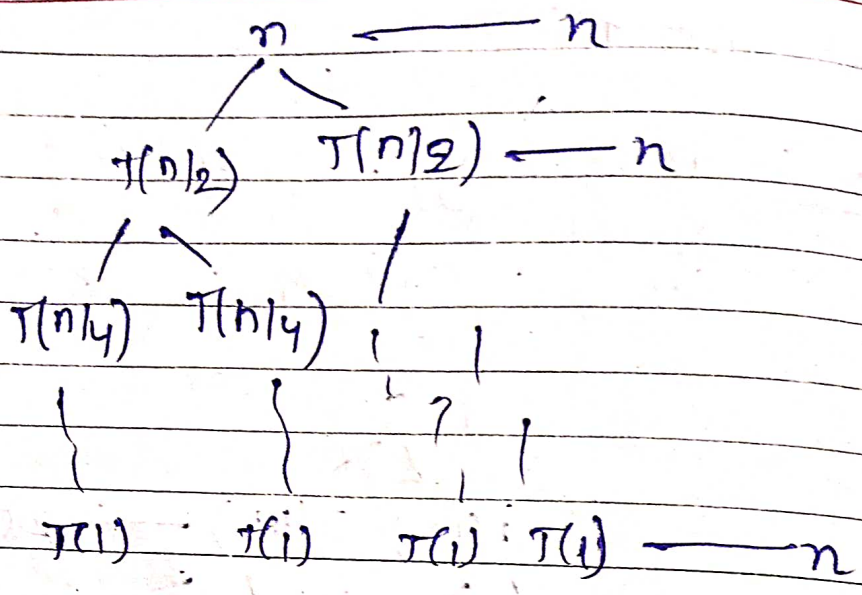
$= c\left(2^k \cdot 2 - 1\right) = c(2n - 1)$

$= O(n)$

Eg $T(n) = 2T(n/2) + n \; ; \; n > 1$

$= 1 \quad ; \; n = 1$

$T(n) =$



$$\frac{n}{2^0} + \frac{n}{2^1} + \cdots + \frac{n}{2^k}$$

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

$$n = 2^k$$

log on both side –

$$k = \log_2 n$$

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

08/08/23

Ques:-1 A(n)

```
2 {
3   if(n>=1)
4   {
5     A(n-1)
6     printf(n)
7     A(n-1)
8   }
9 }
```

A(3) = 15 => $2^{3+1}-1$
A(2) = 7 => $2^{2+1}-1$
A(1) = 3 => $2^{1+1}-1$

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

Total size = n+1
Total size = (n+1)k
Total size = O(nk)
Total size = O(n)

```
#include<stdio.h>
int A(int)
int main()
{
   int i;
   printf("Enter any no");
   scanf("%d",&i);
   A(i);
   return 0;
}

void A(int i){
```

n+1 => because for n=2 n+1 calls are in function stack

It maintains intermediate result in a table.

3	2	1	0

= 3 2 1 4 1 2 3 2 1

~~1 2 3 4 3 2 1~~

for n=4 => 1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

At run time instead of calculating all the value we refer to the table for intermediate results this helps us to ~~do~~ reduce the space

complexity. This is known as dynamic programming.

Ques:- $T(n) = 2T(n-1) + 1 \; ; n \geq 0$

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

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

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

Replace

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

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

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

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

To stop $n-k=0$

put $k=n$

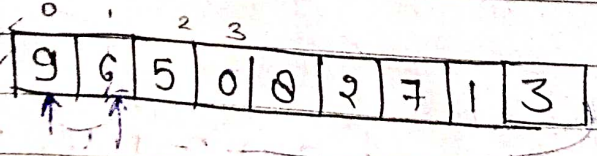
$= 2^n T(0) + 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1$

$= 2^n + 2^{n-1} + 2^{n-2} + \cdots + 2^2 + 2 + 1$

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

$T(n) = O(2^n)$

Insertion Sort:-

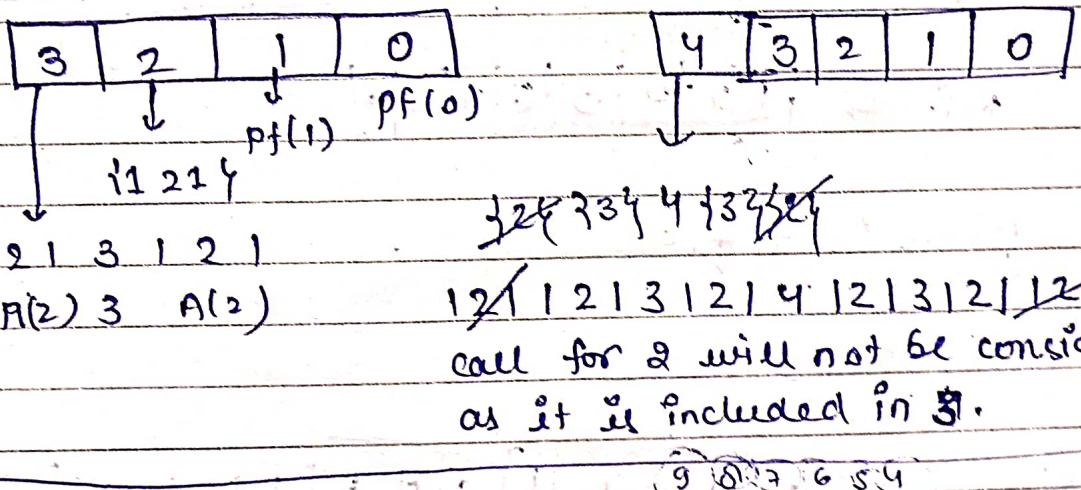


Insertion Sort (A){

for j = 2 to A.length

```
    Key = A[j];
i = j-1;
while( i>0 and A[i]>key)
{
        A[i+1] = A[i]
        i = i-1
}
    A[i+1] = key
}
```

Insertion sort allows sorting by picking one start element and compare it with the immediate next element and place it on left if it is smaller and place on right if it is greater.



Insertion sort T.C. =>

visiting ↑ ↑ swappings

$2 \rightarrow 1+1 = 2 \quad (2\times1)$

$3 \rightarrow 2+2 = 4 \quad (2\times2)$

$4 \rightarrow 3+3 = 6 \quad (2\times3)$

$n \rightarrow (n-1)*(n-1)$

9 8 7 6 5 4

$n = (n-1)$

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

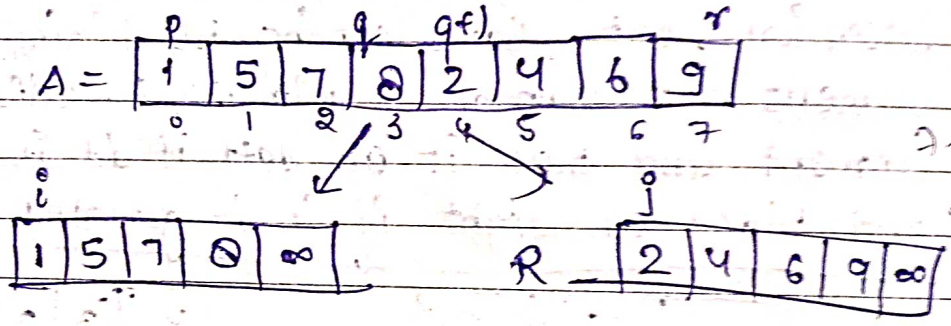
$2(1+2+3+\cdots n-1)$

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

$= \Theta(n^2)$

Divide and Conquer Algo -

Merge Sort :-



Res. ⇒ 1 2 4 5 6 7 8 9

Merge (A, p, q, r)

{

$n_1 = q - p + 1$

$n_2 = r - q$

let $L[1, \ldots n_1+1]$ and $R[1 \text{ to } n_2+1]$ be new array

for (i=1 to n_1)

$L[i] = A[p+i-1]$

for (j=1 to n_2)

$R[j] = A[q+j]$

$L[n_1+1] = \infty$

$R(n_2+1) = \infty$

Mosfankh

Date: Page:

```
i = 1 , j = 1
for ( k = p to r)
{
                                    S.C. = O(n)
    if (L[i] ≤ R[i])                T.C. =
        A[k] = L[i]
        i = i + 1
    else
        A[k] = R[j]
        j = j + 1
    }
}
```

```
Merge_Sort (A, p, r)
{
    if ( p < r){
    {   q = ( p + r / 2 )
        merge-sort (A, p, q) ; → T(n/2)
        merge-sort (A, q+1, r) ; → (n/2)
        Merge (A, p, q, r) ; ⟶ O(n)
    }
}
```

Ques:-

0	1	2	3	4	5	6	7
10	20	30	40	1	5	6	9
			q	q+1			

Soln:- L =

10	20	30	40	∞
0	1	2	3	4

$n_1 = 3 - 0 + 1 = 4$

$4 + 1 = 5$

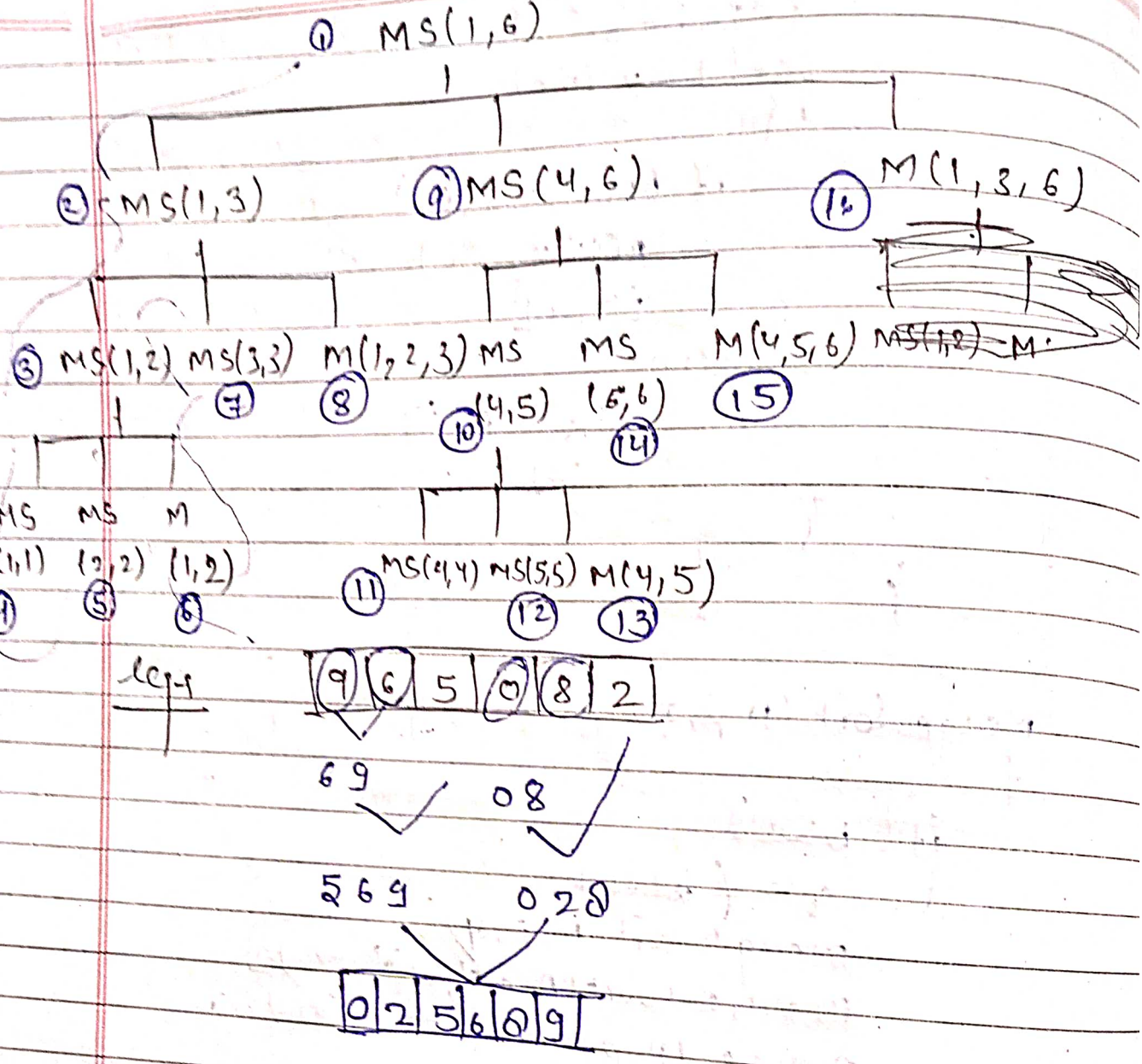
R =

1	5	6	9	∞

$n_2 = 7 - 3 = 4 + 1 = 5$

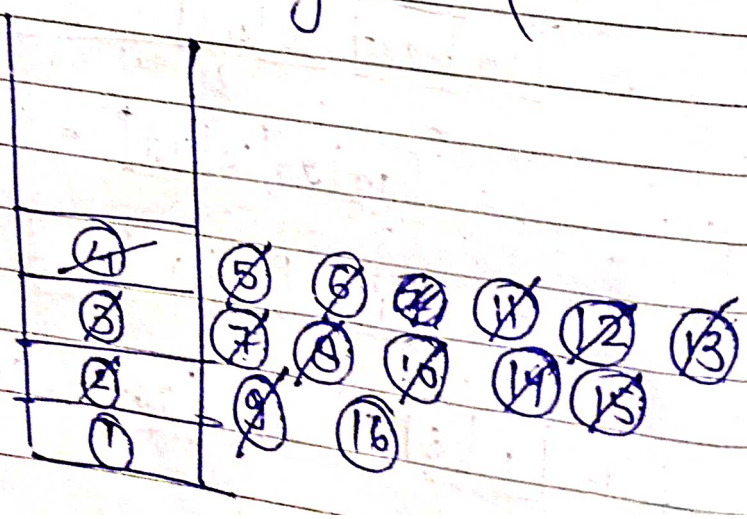
A =

1	5	6	9	10	20	30	40



Time Complexity = $2T(n/2) + O(n)$ (Master's Theorem)
$= O(n \log n)$

Stack Call



Space = depth of tree = $\log n + 1$

$= \log(\log n + 1)k$

$= O(\log n)$

Total space complexity = $O(n) + O(\log n)$

$= O(n)$

Quick Sort :- Good for smaller inputs not for big inputs.

```
Partition(A, p, r) {
        x = A[r] // Declaring last element
        i = p - 1      as pivot
        for(j = p to r-1)
        {
            if( A[j] ≤ x)
            {
                i = i+1
                exchange( A[i] with A[j])
            }
        }
        exchange A[i+1] with A[r]
        return i+1
}

Quick Sort (A, p, r)
{
    if ( p < r)
    {
        q = partition (A, p, r)
        QuickSort (A, p, q-1)
        Quick Sort ( A, q+1, r) { }.
```

Diff b/w merge and quick sort:-

Both works on divide and conquer but in merge sort array divides exactly from the middle and in quick sort array divides from the pos. of pivot element and partition is generally skewed. Only in ideal condn partition will be from middle in quick sort.