TIME COMPLEXITY ANALYSIS

1) **Experimental Analysis**

= Here we actually execute 2 or more codes and compare time and storage.

↓

We compare using relative time ~~not~~ since time is MACHINE DEPENDENT.

LIMITATIONS :-

1) In order to compare we need to write all possible soln.
2) General soln as a function of input size is not available (soln ⇒ Time consumed)
3) One input (given) can be good for one Algorithm and bad for other Algorithm. This does not gives us the right Sorting comparison.

Eg: Input : 1 2 3 4

	Iterations used
Insertion	1
Selection	3

Comparison

Input size Vs No. of operations i.e. Complexity (No units)

Note = ∴ We generally do analysis using worst size or worst input.

= The time complexities for 2 different inputs but of same size can vary. i.e. dependent on nature of input

= We take only significant figures of input size i.e. n (powers of n) higher powers will be kept all other figures (powers of n)

GOOD WRITE

Sum of n-natural numbers $= \frac{n(n+1)}{2}$

DATE: __/__/____
PAGE _____

and other small constants can be ignored.

Eg: Bubble Sort

Complexity $T(A, n)$ = Time taken in comparisons and exchanging 2 nos.

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

representation

∴ Input sizes in real life is quite big

Eg: $n = 10^{10}$.

$\therefore T(A, n) = \frac{(10^{10})^2 - 10^{10}}{2} \approx 10^{20}$.

⇒ Ifi

$T(A, n) \leq C \cdot f(n)$ → constant

then we can say that Algorithm is

$O(f(n))$

Eg: Bubble sort :-

$\left(\frac{n^2-n}{2}\right)k \leq n^2 \cdot k$

Then, $T(A, n) = O(n^2)$

Bcoz (Bubble Sort)

= As size of input (n) tends our actual time function $\left(\frac{n^2-n}{2}\right)$ approaches big-oh function.

$f(n) = (n^2)$

= This time $T(A, n)$ we calculate is the worst case time of any algorithm since we calculate max. no. of operations if performed.

GOOD WRITE

= Complexity is means not the loops (their no.s) but the total no. of times we access a particular element or going at a index while scanning

= It can happen that for the same value of 'n', time taken is different that will depend upon the values of input i.e. input set of values.

Time Complexities

Time Complexity for a given value of 'n' can range from BestCase to WORST case time and hence we take $O(f(n))$ s.t. $T(A,n) \leq c\, f(n)\ \forall\ n > n_0$

- (1) Worst Case : Max Time
- (2) BEST CASE : Min Time

($T(A,n)$: worst case timing func.)

⇒ Note: Improve Algorithm for Bubble sort for the case of Best case time.

Time Complexities for :-

(1) selection sort (3) Insertion Sort

(2) Bubble sort (4) Linear Search

(4) Linear Search
- BestCase : At index '1'.
- Worst case : when at last index or when next found in Array

(1) Selection Sort

(a) Worst Case

= when the Array is in order (reverse) we want to sort.

⇒ 1st → $(n-1)k_c + k_s$

2nd → $(n-2)k_c + k_s$

3rd → $(n-3)k_c + k_s$

$(n-1)$ → $k_c + k_s$

GOOD WRITE

$T(A,n) = \frac{(n-1)(n)}{2} k_c + (n-1) k_s$

DATE: __/__/____
PAGE _____

$T(A,n) = \frac{n^2-3n+2}{2} (k_c) + (n-1) k_s$

for, $k_c \approx k_s$

$T(ss,n) = \frac{n^2-3n+2}{2} (k) + (n-1) k$

$T(ss,n) = n^2$

(b) **Best Case**

⇒ When, the Array is already sorted then after 1st comparison only

= When the $T(A,n)$ is of $O(1)$, this means that constant time is taken by algorithm. This means only same constant no. of operations are done each time independent of input size. For two Algorithms having x and y operations performed both | For so operational Algorithm | are $O(1)$ not $O(20)$

Eg: For n=1;
For n=2;
For n=3;

$T(A,n) = O(1)$
not $O(20)$
∴ we even neglect constants.

RECURRENCE RELATIONS ANALYSIS

Recurrence Relation

(1) BINARY SEARCH

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

Add All Equation

GOOD WRITE

$T(n) = k + k + k + k + - - - + k$ (x times)

$\Downarrow$

x times, where $\boxed{x = \log_2 n}$ as we can write,

$n/2^0 \leftarrow T(n) =$ - -

$n/2^1 \leftarrow T(n/2) =$ - -

$n/2^2 \leftarrow T(n/4) =$ - -

:

that will depend upon n $\quad n/2^{(x)} \leftarrow T(\frac{n}{x}) =$ - -

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

$n = 2^x$

$\boxed{x = \log_2 n}$ ②

$\therefore \boxed{T(n) = k \cdot \log_2 n}$

Best case scenario.

$\Downarrow$

$O(1)$ = when the element to be found is at middle

Note: Remember that whenever there is something like the input is divided by the factor of '2' in Algorithm Analysis then there will always be a factor of 'log$_2$' in the output function of time complexity.

Also, $\boxed{\log_2 n << n}$

= Above 'k' is the time taken to compute middle and compare with middle the searching element.

GOOD WRITE

Bubble sort $T(\ ) <$ Selection sort $T(\ ) <$ Insertion sort $T(\ )$

DATE: / /

PAGE

MERGE SORT

⇒ Recurrence Relations

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

- $\left(\frac{n}{2} + \frac{n}{2}\right)$: Copying the elements
- $T\left(\frac{n}{2}\right) + T\left(\frac{n}{2}\right)$: 2 recursive calls
- $\left(\frac{n}{2} + \frac{n}{2}\right)$: Merging the 2 sorted Arrays

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

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

$$T(n/4) = 2\left(\frac{n}{4}\right) + 2T\left(\frac{n}{8}\right)$$

$$\vdots$$

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

$2^{x+1} = n$

$x+1 = \log n$

$x = \log n - 1$

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

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

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

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

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

no. of nodes at level $= 2^{\text{level}-1}$

$2^{(\log_2 n + 1) - 1} = 2^{\text{level}-1}$

Factorial :-

Time taken to check for base case

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

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

GOOD WRITE

$T(1) = k$

$T(n) = (n-1)k + k \quad (n>1)$

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

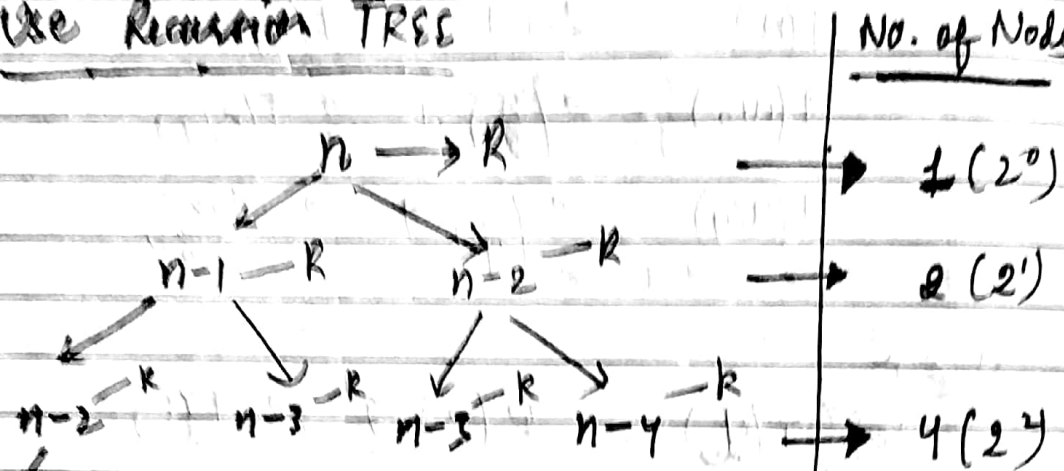
$\boxed{T(n) = O(n^2)}$

Fibonacci [when there are more than one calls]

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

↳ Non-Homogeneous, $\theta = 2$, $D = 1$, Relation

Use Recursion TREE



= Fibonacci Recursion tree will never be a COMPLETE tree. For a root left tree will be always of greater height than right subtree.

Not all nodes present here

some of the branches of tree will finish before but we will assume all nodes are present.

$\therefore T(n) = 2^0 + 2^1 + 2^2 + \cdots + 2^{n-1} + 2^{n-1} T(1)$ $\left(n \text{ to } 2^{\log_2 n} \text{ i.e. } T(1)\right)$

$= \underline{\text{Exponential Complexity}}$ [Total No. of levels − 1]

= WORST CASE in case of trees is when the TREE IS COMPLETE TREE (Binary Tree). ∴ max no of steps comparisons would be there then only.

find Complexity

Eg: 1. Given, m & n;

```
for(i=0; i<=n-1; i++)
{
    for(j=i+1; j<=m; j++)
    {
        Constant no. of operations.
    }
}
```

→ The Complexity Answer depends upon the RELATIONSHIP of m & n

And the condition of II loop can turn false before Outer loop gets false. then also constant work will be done.

{ i.e. m=n, m<n, m>n } so as to think of neglection.

→ ~~Initialisation~~ + condition check

When, $m <= n$; $\Rightarrow$ ~~n^2+n-m (Ans)~~ $\frac{m(m+1)}{2}$ (Ans)

When, $n < m$; $\Rightarrow$ $\frac{2mn-n^2-n}{2}$ (Ans)

Eg: 2. :- Change m by n in 'I' Eg:

Eg: 3.

```
for(i=0; i<=n-1; i++)          n+2
{
    for(; i<=n; i++)           n+1
    {
        Constant no. of operations.
    }
}
```

GOOD WRITE

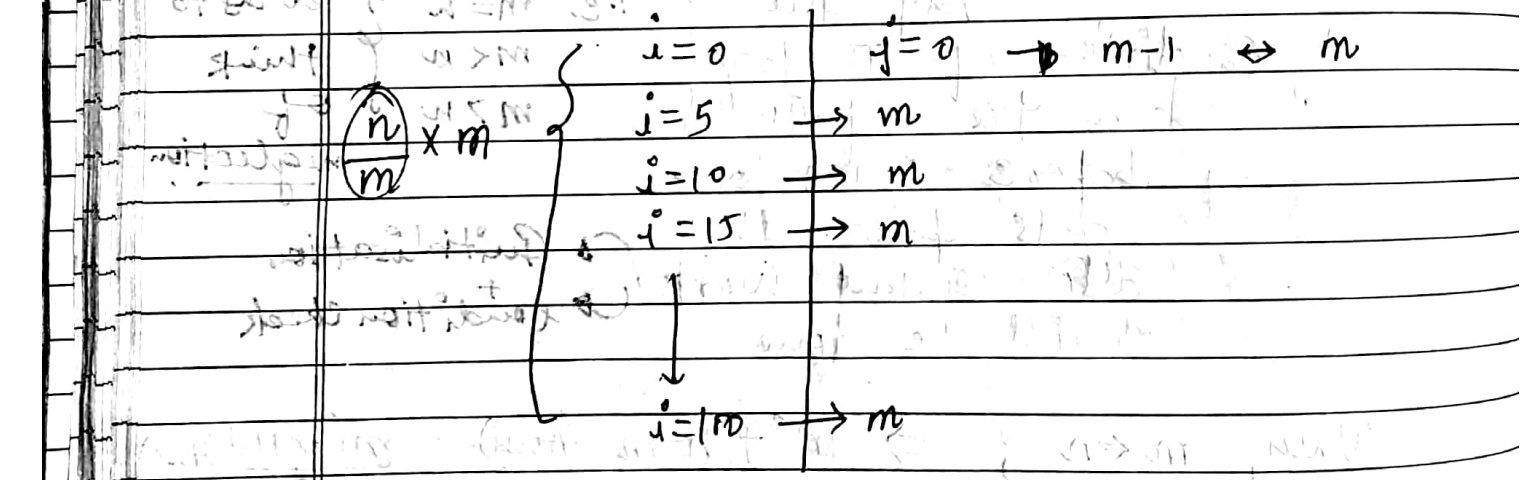
Eg:

```
for(i=0; i<=n;)
{
  for(j=0; j<=m-1; j++)
  {
  }
  i = i+j;
}
```

Case I : $(m < n)$

Case II : $(m = n)$ [Ans = 2m]

Case III : $m > n$ [Ans = m]



Ans : $m \times \left[\frac{n}{m}\right]$ → Greatest Integer Function

Space Complexity

= "Except Input space" required or Extra space" required apart from Input.

= Maximum space required at any point of time.

Then, Time Complexity = $T_1 + T_2 + T_3$

Space Complexity = $Max(S_1, S_2, S_3)$

= This difference is due to Reutilization property of GOOD WRITE space.

S(A, n) ⇝ Bubble
→ Insertion
→ Binary search (Iterative version)
→ Linear
→ Selection

Constant Space required : only some of the variables are made once and input is ignored

Hence, O(1).

Array

"Iterative" function's space Complexity can be CONSTANT hence O(1).

but,

that of "Recursive" functions can never be O(1).

Eg: Recursion - factorial :-

Combined they require 'k' space.

Eg: n=5,

fact(5) | n=5

fact(4)

fact(3)

fact(2)

fact(1)

fact(0) (Base case)

Space will also be required to store the calling functions queues. i.e. while returning - where to return will also be saved.

Deallocation of Space

∴ S(A, n) = nk not (n+1)k.

= Maximum amount of space required will be when base case is achieved Hence this will be the space Complexity.

but that no. of calls depends upon generally on INPUT size

In Iterative versions the extra space required can be reused each time in SCOPE & GOOD WRITE there of variables. But in Recursive Definitions old variables cannot be reused again ∴ for each call we need to Extra more space.

Q = Calculate space complexity for Worst Case of BINARY SEARCH (RECURSIVE).

DATE: __/__/__
PAGE

B.S.(a, n) — size

Only Address is passed No new Array is formed

main → B.S.(a, n)

n/2^0 ⇐ for each call n/2^1 ⇐ a, n ⇐ { start, end, mid } this much space is required

n/2^2 ⇐

n/2^x ⇐

n	→ k
n/2	→ k
n/4	→ k
⋮	
(1)	→ k

∴ No. of levels = x+1

⇒ $2^x = n$ or $\frac{n}{2^x} = 1$

$x \log_2 2 = \log_2 n$

$\boxed{x = \log_2 n}$

∴ Space required = $k \cdot \log_2 n + k$

or $S(A,n) \in O(\log_2 n)$ ← simplified way of writing

= Above in term 'log n' base is 2 ∵ recursive calls are '2' for each Definition. This could be $\log_m n$ if no. of recursive call are 'm' for each definitions. ∴ • $\frac{n}{2^x} = 1 \Rightarrow \boxed{x = \log_2 n}$

GOOD WRITE!

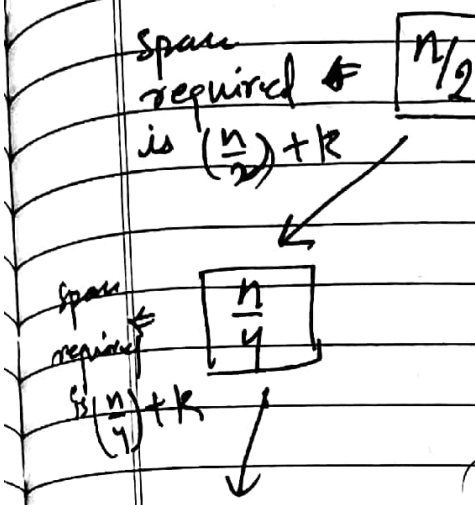
Q = Also Calculate for MERGE SORT.

DATE: __/__/____

PAGE _____

[Answer = n]

n ⇒ Requires 'n' amount of space + 'K' space to save function call. → This is to store the Array make the new Arrays.



Recurrence Relation :-

$S(n) = \underbrace{n}_{\frac{n}{2}+\frac{n}{2}} + S(n/2) + S(n/2)$

Creating New SubArrays

⇒ The Above is Wrong ∵ for Acc to traversal Recursive schema the space for Right Subtree will be the reutilized space used before by $\frac{n}{2^x}$ left Subtree

$$\therefore \quad k\log_2 n + \left[\frac{n}{2} + \frac{n}{2} + \frac{n}{4} + \cdots + 1\right]$$

$2^0 \simeq 2n$

$= k\log n + 2n$

$\therefore = O(n)$ [∵ $n > \log n$]

= In another Approach where no new Array is created we generally use low, mid, high but then also Space Complexity will be O(n) since we will require then also an array of size 'n' to merge the arrays.

No. of terms ⇒ $x = \log_2 n$

$$S_x = \frac{n\left(1-\left(\frac{1}{2}\right)^x\right)}{\left(1-\frac{1}{2}\right)} = 2n\left(1-\left(\frac{1}{2}\right)^{\log_2 n}\right)$$

GOOD WRITE

$$S_x = 2n\left(1-\left(\frac{1}{2}\right)^{\log_2 n}\right)$$

DATE: __/__/__
PAGE

$$S_x = 2n\left(1-2^{-\log_2 n}\right)$$

$$S_x = 2n\left[1-\left(2^{\log_2(n)}\right)^{-1}\right]$$

$A = 2^{\log_2(n)}$

$$S_x = 2n\left(1-(n)^{-1}\right)$$

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

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

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

$$\boxed{S_x \approx 2n}$$

Q = Also compute space complexity for 'fibonacci Recursive problem'!