Page No.
Date 11 10 2023

DAY 24

* TIME AND SPACE COMPLEXITY FROM ZERO TO ADVANCED

1] Time complexity != Time taken :-

- For example -:

old m/c	new m/c
sum up 10 nos [2 sec]	sum up 10 nos [1 sec]
sum up 10000 nos [200 sec]	sum up 10000 nos [100 sec]

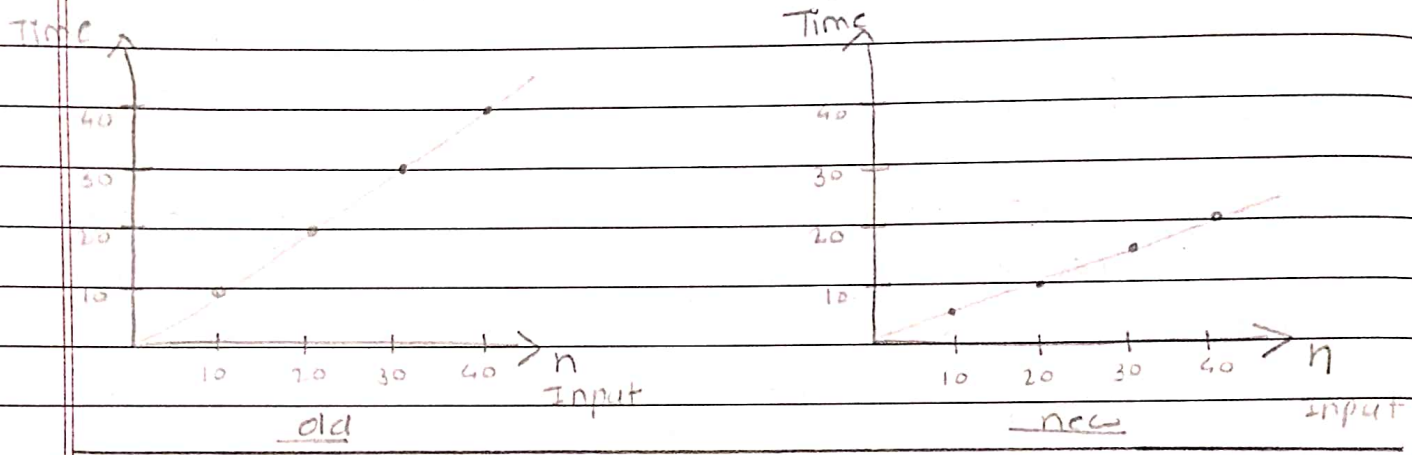
- As shown above, the time taken can depend on the specific machine, + it can depend on the other resources being used by other apps in computer.
- Time taken can vary based on different factors, and hence Time complexity != Time taken

2] Time Complexity:

- It is the total time taken by an algorithm to run as a function of length of the input ('n').
- Time complexity can be measured and can vary depending upon the length of the input n
- It is shown more clearly in the below diagram.

- Example:

- Let's say there are 2 m/c - old and new and there's an algorithm applied in both the machines, the time taken with respect to n is shown below:



- From above graphs we can say,

  Time = n ∴ Time = n/2

- Hence we can say that there's relation between the two algorithms, they're representing the linear function, hence the two machines are running using the same algorithm.

- The time complexity for above algorithm is -

  Time = n & Time = n/2

  hence Time = n & Time = n .... [constants are not considered]

3] Best case, Average case, Worst case:

- While calculating the time complexities, we consider best, avg and worst cases.

- But while developing the algorithm, we always consider the worst case and optimise our solutions.

Worst case (big O) notation: Big (O)	Best case (omega) Ω	Average case (theta) Θ
ex: $O(n)$	ex: $\Omega(n)$	ex: $\Theta(n)$

- **Example:**

calculate time complexity of the code :-

```
for (int i = 1; i <= n; i++) {
    cout << "Chamka";
}
```

① step: 1 + 1 + 1 + 1 + ... n

$1n \Rightarrow \boxed{n}$

$\Rightarrow O(n)$

- **Steps to calculate the Big (O) notation :**

i) Start debugging the code, calculate the steps involved in program

ii) Add all the steps

iii) Ignore the constant terms, if exists any

iv) And if there are powers of N resulting after calculating the sum, then consider the higher order power of N

- **Rules :**

i) Ignore constant terms :

$3n + 1 = \text{Big } O(n)$

ii) Consider the higher power of N :

$N^3 + N^2 + N = O(N^3)$

Page No.
Date

- Why do we follow the above rules?

- To compare the two algorithm shown below, it is confusing to check which one is better.

$3N^3 + 2N^2 + N$	$N^3 + 10^9 N$

- But if we ignore the constant terms and apply rule no. 2, they become

N^3	N^3

- Hence, both the algorithms are almost equal and the constant or lower powers are not ~~gekt~~ going to make a major difference the time complexity, hence we apply the rules for ease of understanding.

4] Examples:

i)

```
cin>> n;
For (i=0; i<10; i++) {
        cout << "Chamka";
}
```

n	steps
10	10
100	10
10000	10

worst
~~Best~~ case $O(10) = O(1)$

best
~~Worst~~ case $\Omega(10) = \Omega(1)$

Avg case $\Theta(10) = \Theta(1)$

- if inner and outer for loops doesn't depend on one other, we can simply write $O(n^2)$ T.C

Page No. | Date

2] `cout << n*(n+1)/2;` ⇒

n	steps
10	1
100	1

best case = $\Omega(1)$

avg case = $\Theta(1)$

worst case = $O(1)$

3]
```
For (i=1; i<=n; i++)      → n   } n*n = n^2
    For (j=1; j<=n; j++)  → n
        cout<<"chamka";
```

calculate steps:

i=1	i=2	i=3	i=4	i=5 ... i=n	
j<=n	j<=n	j<=n	j<=n	j<=n	j<=n
n times printing	n times	n times	n times	n times	n times

$n + n + n + n + n + \dots n$

$= n^2$

Hence, best = $\Omega(n^2)$

worst = $O(n^2)$

avg = $\Theta(n^2)$

4]
```
For (i=1; i<=n; i++)
    For (j=1; j<=i; j++)
        cout<<"chamka";
```

i=1	i=2	i=3 ...	i=n
j<=1	j<=2	j<=3	j<=n
1 times	2 times	3 times	n times

$1 + 2 + 3 + \dots n$ ⇒ sum of all natural nos (1-n)

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

Page No.
Date

$\Rightarrow n * (n+1) / 2$

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

$\Rightarrow \boxed{n^2}$

* best = $\Omega(n^2)$
* worst = $O(n^2)$
* avg = $\Theta(n^2)$

5]
```
for (i=1; i<=n; i++)
    for (j=1; j<=i^2; j++)
        cout<<"chamka";
```

i=1	i=2	i=3	...	i=n
j<=1	j<=4	j<=9		$j <= n^2$
1 times	4 times	9 times		n^2 times

$1^2 + 2^2 + 3^2 + .. n^2$

$\Rightarrow$ sum of square of natural nos

$\Rightarrow n * (n+1) * (2n+1) / 6$

$\Rightarrow 2n^3 + n / 6$

$\Rightarrow \boxed{n^3}$

* best = $\Omega(n^3)$
* worst = $O(n^3)$
* avg = $\Theta(n^3)$

6]
```
for (i=1; i<=n; i++)
    for (j=1; j<=m; j++)
        cout<<"chamka";
```

→ i will be looped n times.
→ j will be looped m times
→ hence complexity is O(n * m)

7] For (i=1; i<=n; i++)
    For (j=1; j<=i²; j++)
        For (k=1; k<=n/2; k++)
            cout<<"chamka";k

i = 1	i = 2	i = 3	---- i = n
j <= i²	j <= 4	j <= 9	j <= n²
k <= n/2	k <= n/2	k <= n/2	k <= n/2
n/2 times printing	4 * (n/2) times printing	9 * (n/2) times	n² * (n/2)

$$\Rightarrow \frac{n^3}{2} + \frac{2^2 n}{2} + \frac{3^2 n}{2} + \frac{n^2 n}{2}$$

$$\Rightarrow \frac{n}{2}\left[1+2^2+3^2+n^2\right] \Rightarrow \frac{n}{2}\left[n*(n+1)*(2n+1)/6\right]$$

$$\Rightarrow \frac{n^4+n^3+n^2+\dots}{12}$$

$$\Rightarrow \boxed{n^4}$$

best = $\Omega(n^4)$
worst = $O(n^4)$
avg = $\Theta(n^4)$

8] For (int i=0; i<n; i++)
    for(j=0;
    if (arr(i) == key)
        return i;
return -1;

- For best case: $\Omega(1)$ | 6 | 5 | 8 | 7 | 1 | key = 6 (↑ at 6)
- For worst case: $O(n)$ (key occurs at last)
  i = 0 to n-1 ⇒ n steps
- For average case: $\Theta(n)$

$$\Rightarrow \frac{1 + 2 + 3 + \dots n}{n} = \frac{n + (n+1)/2}{n}$$

$$= \frac{n+1}{2} = n$$

- 1+2+3 ... n represents that the ele can occur at 1 index or 2 or 3 or upto n, this is the average case so add all the possibilities - steps / n (no. of iterations)

9] For (i=1; i<=n; i*=2)
    cout << "chamka";

i=1 | i=2 | i=4 | i=n
1 times | 1 times | 1 time | 1 times
printing

1 + 1 + 1 + 1 ... + 1 = ~~n~~

- To Find the time complexity in such cases, when

we don't exactly know the no. of steps we're performing, in that case, we can apply some other approach as

printed 3+1 times

$i = 2^0$	$i = 2^1$	$i = 2^2$	$i = 2^3$	$i = 2^K$
1	1	1	1	1

- hence from above example, we can say that the at $i = 2^K$ there will be $K+1$ steps performed

$n = 2^K$ .... $n = 2^K$ after end of loop

$\log n = K \log 2$

$K = \frac{\log n}{\log 2}$

$\boxed{K = \log_2 n}$

- Therefore, as per the formula we've seen the steps performed are $K+1$

i.e. $\log_2 n + 1$

$\Rightarrow \boxed{\log_2 n}$

- The time complexity is $O(\log_2 n)$.
- We can directly, find the TC in some of the cases as:

$i = 1 - 2 - 4 - 8 - 16 - 32 \ldots n$

$\boxed{\log_2 n}$

$i = n \ldots 32 - 16 - 8 - 4 - 2 - 1$

$\boxed{\log_2 n}$

$i = 1 - 3 - 9 - 27 - 81 \ldots n$

$\boxed{\log_3 n}$

$i = 1 - 4 - 16 - 64 \ldots n \Rightarrow \boxed{\log_4 n}$

10)
```
for (i=n/2 ; i<=n ; i++)
    for (j=1; j<=n/2; j++)
        for (k=1 ; k<=n; k++)
            cout << "chamka";
```

- WKT when inner loops don't depend on outer loops then we'll simply multiply the iterations of each loop as :-

1st loop => n/2 times
2nd loop => n/2 times
3rd loop => n times

$\therefore n * (n/2) * (n/2) = \frac{n^3}{4} = \boxed{O(n^3)}$

11)
```
For (i=n/2 ; i<=n ; i++)
    For (j=1; j<=n ;j=2*j)
        For (k=1; k<=n ; k*=2)
            cout << "chamka";
```

- WKT, when outer loops and inner loops don't depend on each other, we can simply multiply their steps to find TC.

1st loop => n/2 ..... n/2 → n
2nd loop => $\log_2 n$ ..... increased by 2
3rd loop => $\log_2 n$ ..... updating by 2

$\therefore \frac{n}{2} * \log_2 n * \log_2 n \Rightarrow \frac{n(\log_2 n)^2}{2}$

$\therefore TC = O(n(\log_2 n)^2)$

{harmonic series means when denominator follows an AP (arithmetic progression)}

Page No.
Date

12]
```
for (j=1; j<=n; j++)
    for (i=1; i<=n; i+=j)
        cout<<"chamka";
```

- Find steps, :

j = 1	j = 2	j = 3 ...	j = n
i = 1 to n	i = 1 to n	i = 1 to n	i = 1 to n
n times	n/2 times	n/3 times	1 (n/n times)

$$n + \frac{n}{2} + \frac{n}{3} + \cdots$$

$$\Rightarrow n + \frac{n}{2} + \frac{n}{3} + \cdots + \frac{n}{n}$$

$$n\left[1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}\right] \quad \text{... \{harmonic series\}}$$

$$O(n * (\log_e n))$$

$$\Rightarrow \boxed{O(n \log_e n)}$$

* Space Complexity :-

- It is the amount of space taken by an algorithm as a function of length of input.
- It doesn't mean that how many GB's TB's or KB's are taken by the algorithm.
- Space complexity is always measured in terms of no.
- input increases, SC increases and vice versa.
- Or sometimes it can be not depend on 'n'.
- There're 2 concepts in SC :
  - i) Auxiliary space complexity ... {most asked}
  - ii) Total space complexity

Page No.
Date

i) Auxiliary space complexity:

- It only considers the amount of space that is allocated by developers in the algorithm and don't consider the input given spaces/ memories

- For example:

given i/p: A[] = |1|2|3|4|5| , size = n

- Hence, SC complexity of above case, if we don't create any extra variables / arrays in the algorithm will be:-

$O(1)$

ii) Total space complexity:

- It considers all the allocated spaces, both input given memories and user created spaces.

- For example:

given i/p: A[] = |1|2|3|4|5| size = n

- Hence, SC of above case, if we don't create any extra memory / arrays in the algorithm will be:- $n + 1$ → user memory

given memory $O(n)$

* Examples on space complexity:

i) - Given: |4|6|2|5|1| (indices 0 1 2 3 4, n)

- Problem: Square all the nos of array

- Algorithm:

|16|36|4|25|1| (n) ← new array

∴ Auxiliary = $O(n)$ Total ∴ $2n = O(n)$

ii) for (i = 1; i <= n ; i++) ⇒ n is given
cout << i

Auxiliary : 1 = O(1) Total : 1 + 1 = O(1)

– We have taken the space as 1 for variable of int n and i because there memory will be same for always unlike array.

* Time complexities [worst to best] :-

O(N!) → worst
O(2^n)
O(N^3)
O(N^2)
O(N log n)
O(N)
O($\sqrt{N}$)
O(log n)
O(1) → best

* Homework Problems :

1) for (int i=1; i<=n*n; i++)
cout << "Coder Army";

Soln: big (O) = $O(n^2)$
big (theta) = $\Theta(n^2)$
big (omega) = $\Omega(n^2)$

Page No.
Date

2] For (int i = 1; i <= n*n ; i = i+2)
cout << "coder army";

Soln: i=1 i=3 i=5 i=7 ... i=n*n

1 1 1 1 ... 1

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

big(Θ) = $\Theta(n^2)$

big(Ω) = $\Omega(n^2)$

3] For (int i=1; i<=n; i++)
For (int j=1; j<=n; j=j+5)
cout << "coder army";

i=1 i=2 i=3 i=n

j=1 to n j=1 to n j=1 to n j=1 to n

print n/5 print n/5 print n/5 print n/5

$\Rightarrow n * \left\{ \frac{n}{5} + \frac{n}{5} + \frac{n}{5} + \dots \right\}$

$\Rightarrow \frac{n^2}{5} + \frac{n^2}{5} + \dots + \frac{n^2}{5}$

$\Rightarrow O\left(\frac{n^2}{5}\right)$ sir's ans $\Rightarrow \boxed{O(n^2)}$

for best case if n given is a multiple of 5

big (omega) = $\Omega(n^2)$

big (theta) = $\Theta(n^2)$

4] For (int i=1; i<=n; i++)
For (int j=i; j<=n; j++)
cout << "coder army";

Page No.	
Date	

i = 1	i = 2	i = 3	i = n
j = 1 to n	j = 2 to n	j = 3 to n	j = n to n
print n times	print n-1 times	print n-2 times	print 1 times

$\Rightarrow n + (n-1) + (n-2) + (n-3) + \dots + 1$

$\Rightarrow n * [n + (n-1) + (n-2) \dots + 1]$ (1st for loop)

$\Rightarrow n^2 + n^2 - 1 + \dots$

$\Rightarrow n^2$

$\Rightarrow$ $O(n^2)$, $\Omega(n^2)$, $\Theta(n^2)$.

5]
```
for (int i=1; i<=n; i++)
    for (int j=1; j<=n; j=j*4)
        cout << "coder army";
```

Soln: For first for loop = n

For second loop = $\log_4 n$

$\therefore$ big(o) = $n * \log_4 n$

$\therefore$ $O(n\log_4 n)$, $\Omega(n\log_4 n)$, $\Theta(n\log_4 n)$

6]
```
for (int i=1; i<=n; i=i*2)
    for (int j=1; j<=i; j=j++)
        cout << "coder army";
```

For first loop = $\log_2 n$

For second loop =>

i = 1	i = 2	i = 4	i = 8	...	i = n
print 1 times	print 2 times	print 4 times	print 8 times		print n times

Page No.
Date

$\Rightarrow 1 + 2 + 3 + 4 + \cdots n$

$\Rightarrow \frac{n * (n+1)}{2}$

hence big (O) $= \log_2 n * \left\{ \frac{n * (n+1)}{2} \right\}$

$= \log_2 n * \frac{n^2 + n}{2}$

	sir's answer
$\Theta(\log_2 n * n^2)$	$O(\log_2 n * n)$
$\Omega(\log_2 n * n^2)$	$\Omega(\log_2 n * n)$
$\Theta(\log_2 n * n^2)$	$\Theta(\log_2 n * n)$

7)
```
For (int i=1; i<=n; i++)
    For (j=1; j<=n; j++)
        For (int k=1; k<=n; k=k*3)
            cout << "Coder army";
```

Soln: For first loop = n

For second loop = n

For third loop = $\log_3 n$ ---- incrementing in multiple of 3

big(O) $= n * n * \log_3 n$

$= n^2 * \log_3 n$

$\therefore$ | $O(n^2 \log_3 n)$ | $\Omega(n^2 \log_3 n)$ | $\Theta(n^2 \log_3 n)$ |

8)
```
For (int i=1; i<=n; i++)
    For (int j=1; j<=n; j++)
        For (int k=1; k<=n; k++)
            cout << "Coder army";
```

Soln: 1st loop = n
2nd loop = n
3rd loop = n

$\therefore$ | $O(n^3)$ | $\Omega(n^3)$ | $\Theta(n^3)$ |