DIPLOMA WALLAH

(Your **One Stop Hub** For Diploma Resources)

DATA STRUCTURES

WITH PYTHON

Complete Notes Based on Full Syllabus

- Diploma Engineering

4th Semester

www.diplomawallah.in

Notes prepared by **Sangam**

Unit-02

Algorithm Analysis :-

Algorithm analysis in data structure is the process of evaluating the efficiency of an algorithm, primarily focusing on its performance with varying input sizes and resources usage.

Why Algorithm Analysis is important

1. Efficiency measurement (Time & space)
   - Every algorithm uses
     - Time (how fast it runs)
     - Space (how much memory it uses)

   Analysis helps us find which algorithm is faster or uses less memory.

2. predicting performance without running code
   - you don't always need to execute the code to estimate its performance.
   - just analyze the algorithm and predict how it will behave.

3. Avoiding Bad Solution
- Helps you avoid inefficient solutions that seem correct but are very slow or memory consuming.

4. Choosing Best Algorithm
- multiple algorithm may solve the same problem.
- Analysis helps choose the best one depending on input size, hardware, or performance goals.

5. Scalability check
- Asymptotic analysis like Big O notation tells us how the algorithm behaves when input increase.

Time and Space Complexity

Many times there are more than one ways to solve problem with different algorithms and we need a way to compare multiple ways.
Also, there are situations where we would like to know how much time and resources an algorithm might take when implemented.

- To measure performance of algorithms, we typically use time and space complexity analysis.

Time Complexity

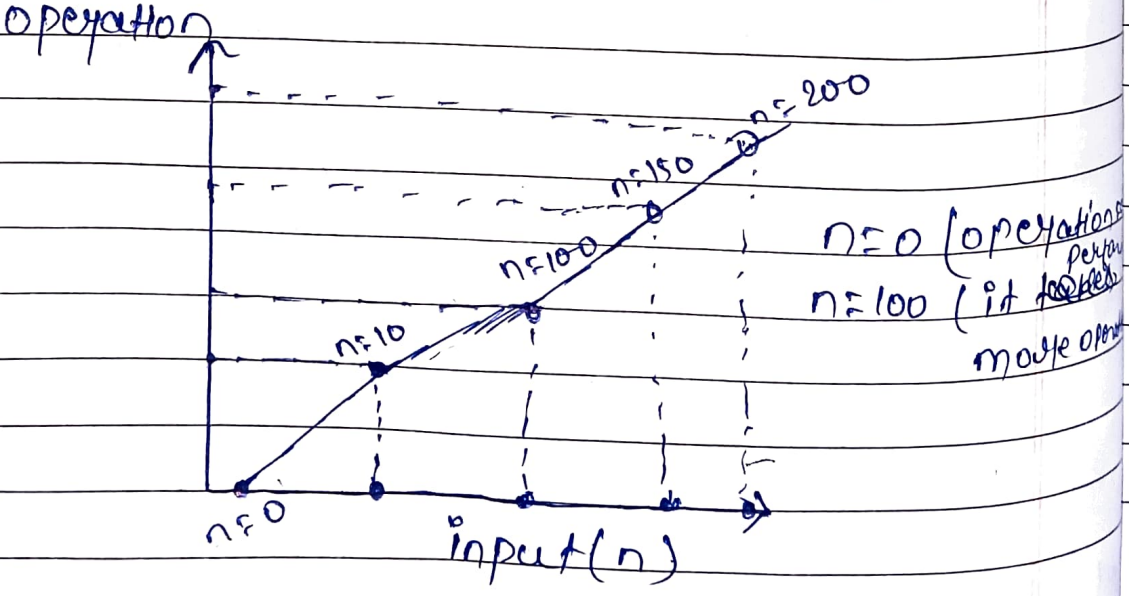
Time complexity != Time taken to run an algorithm.

(window - take 2 min
mac - " 1 min) — different time taken on same algorithm.

Time complexity is a behaviour.
It can be any mathematical function.

example:-

```
for(i=0; i<n; i++){
    print(arr[i])
}
```



As increasing the number of n the operation is performed is increases.

* Time Complexity always measure the ~~worst~~ Cases.

O → Big O

* Constant value are not taken.

Example:- (To solving Time Complexity).

$O(5n^3 + 2n^2 + 52)$

(i) Remove constant

$O(n^3 + n^2)$

(ii) worst value filter (Big value)
i.e. n^2 is less than always of n^3. So,

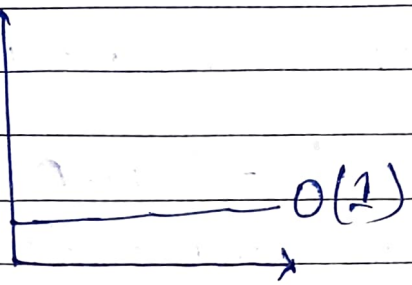
$O(n^3)$

↳ Big O notation.

Sangam

Common Complexity

i) O(1) → constant



It mostly used when we want to find the array index value. by performing the single operation.

Ex- arr[index]

O(1)

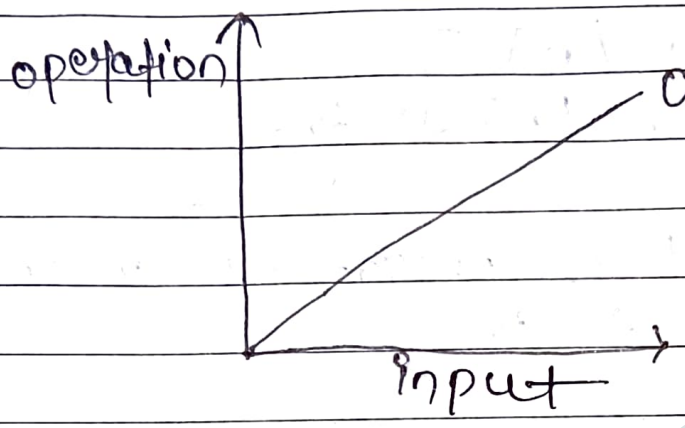
for 'func(arr[])

printf('Hi')

↳ it does not depend on function.

ii) O(n) - Linear TC

In this full array is search one by one.



When we have multiple time complexity in single code we exclude small operation and ~~prog~~ taken.

Ex. print O(1) ✗

loop O(n) ✓

By complexity.

large Time Complexity ✗ Bad Complexity

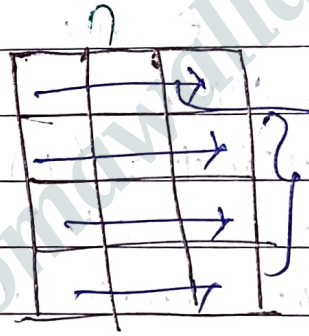
less time complexity is best.

iii) $O(n^2)$ Quadratic TC
mostly used in nested loop.

```
for (n)
  for (n)   ] nesting
```

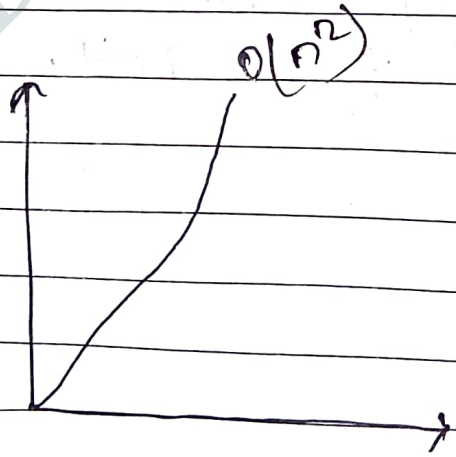
operates two times operations means it take (n^2) times.

Ex- matrix



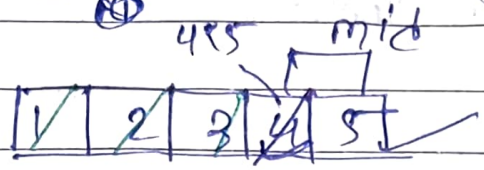
operation will perform

$n \times n$
$= n^2$



iv) O(log n) Better than Linear TC.

Generally used in Binary Search algorithm.



Binary Search from mid

↳ in every iteration we decrease the input size by half.

① n = 100

↳ n = 50

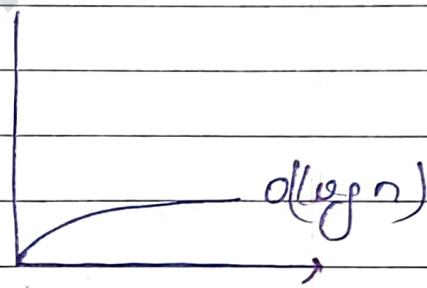
↳ n = 25

↳ n = 12

↳ n = 6

↳ n = 3

↳ n = 1



Lets let array 1 and we have to increase maximum to 100.

Start from.

$1 \times 2 \times 2 \times 2 \times 2 \times 2$

↳ let x time multiple

$1 \times 2^x = 100$

↳ final result

In terms of math

$x = \log_2 100$

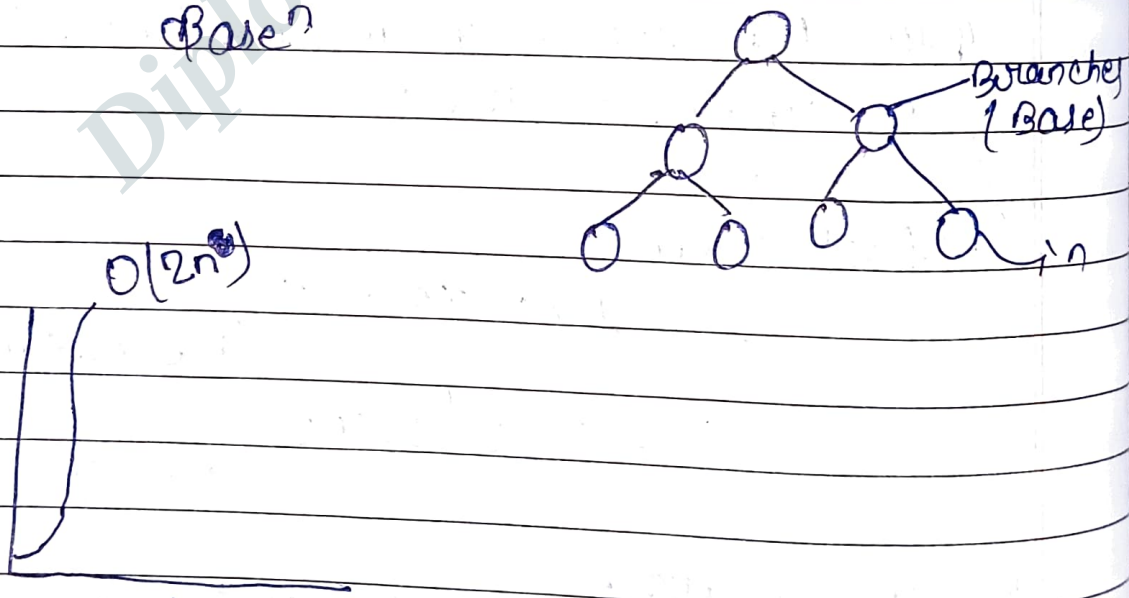
no. of operation. → $x = \log n$

$O(\log n) \gg O(n)$ $O(n \log n) \gg O(n*n)$

↳better

v) $O(n \log n)$ → Sorting
mostly used in Sorting.

vi) $O(2^n)$ → Exponential Recursion.

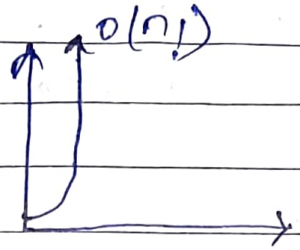


* Bad time complexity.
thats why we use dynamic programming for better time complexity.

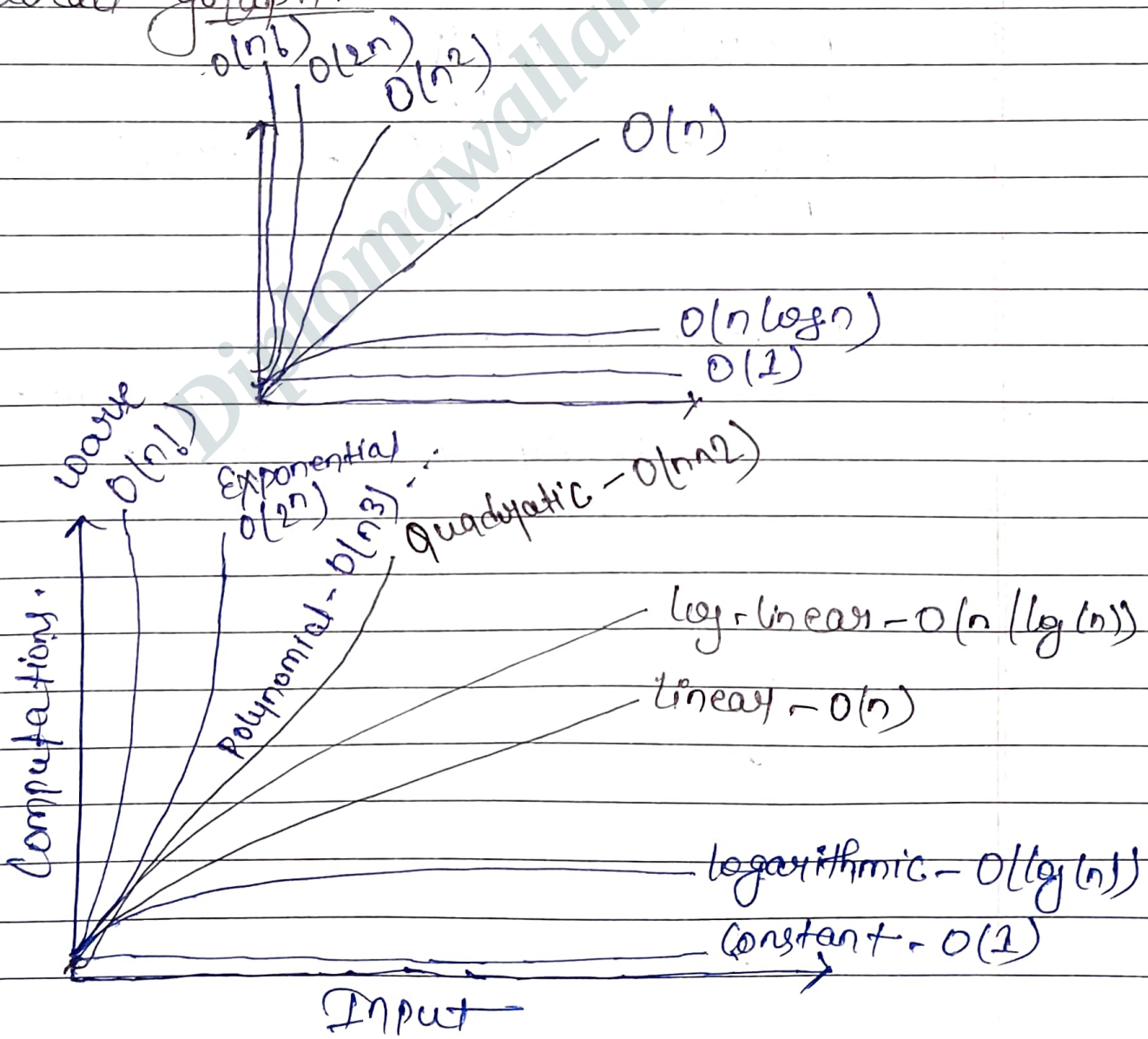
Dynamic programming TC is used for better TC.

vii) O(n!) (permutation)

worst time complexity.



overall graph.



* problem Solving

1 Second ≈ 10^8 operation.
in 1 Second 10^8 operation perform.

$n > 10^8$	$O(\log n), O(1)$
$n \leq 10^8$	$O(n)$
$n \leq 10^6$	$O(n \log n)$
$n \leq 10^4$	$O(n^2)$
$n \leq 500$	$O(n^3)$
$n \leq 25$	$O(2^n)$ – Brute force Recursion
$n \leq 12$	$O(n!)$

operation decreases = $\frac{1}{\text{Time Complexity}}$

inversly proportional.

* Space Complexity * O(n)

1) Amount of Space taken by an algorithm as function of input size (n).

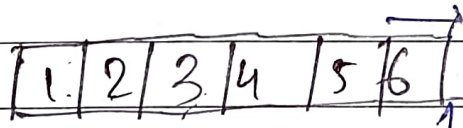
Space can be taken by two ways :-

written code → input (arr, String ...)

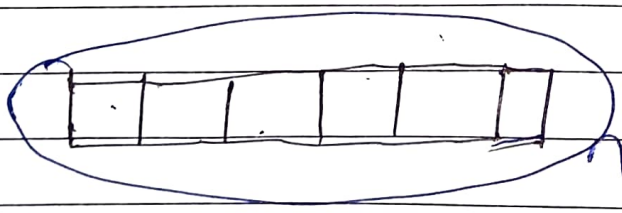
→ auxillary
↳ extra space which is other than input.
(eg - linear search which other than input)

* Input are not considered in Space complexity. The extra space taken by code is count as Space complexity.

(1) eg.



inp - arr[]
out - Square arr[]



extra (auxilliary Space)

input ↑ ∝ auxilliary size ↑
that's why Space complexity is. O(n)

(11)

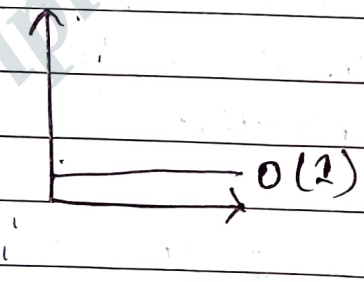
inp - arr[ ]
out - sum

```
int sum = 0
        ↳ constant
for(int i = 0; i < n ; i++)
        sum = sum + arr[i]
```

The size of sum will be constant one memory only take size not dynamically change.
So, the space complexity will be constant.

O(k)
↳ constant

O(1)



Problem

Question

Time complexity for prime no. $O(n^2)$ - Worst

Time Complexity

* Space complexity
O(1)

```
public class Main
{
    psvm (String args[])
    {
O(1) * Constant ←   int n=29;
O(1)            ←   boolean isprime = true;
O(n^2)          ←   for(int i=2; i*i<=n; i++)
                    {
O(1)            ←       if(n%i==0){
O(1)            ←           Sopln("non prime");
                            isprime = false;
                            break;
                        }
                    }
O(1)                if(isprime && n>1)
                    { Sopln("prime no."); }
                    else if(n!=1)
                    {
                        Sopln("non prime");
                    }
    }
}
```

Defination

Time Complexity:-

The time complexity of an algorithm qualifies the amount of time taken by an algorithm to run as a function of the length of the input.

Simply time complexity means how much time an algorithm takes to run, based on the size of the input:-

- It doesn't depend on your computer speed.
- It tells you how the steps increases as the input grows.
- It assume worst-case, average case and best-case scenarios depending on context.

Example:-

```
public Class Add
{
    Psvm {
    int a=10;
    Pnt b=20;
    Pnt sum= a+b;          ---- operation 1
    SOPln(sum);                 O(1)
}                               ↳ Time & Space
}
```

Space Complexity :—

The space complexity of an algorithm quantifies the total amount of memory used by the algorithm as a function of the input size n.

This include both the temporary memory used during execution and the memory needed to store the final result.

It is parallel concept to time complexity.

Auxiliary Space :— It is the extra space (allocated in RAM) or temporary space used by an algorithm.

- The memory needed for input is not counted in auxiliary space. Only the extra memory used to run the algorithm (like recursion stack, temporary array, variables that count).

O(1) — constant
O(n) → linear
O(n²) — quadratic (nested loop)
O(log n) → Binary search
O(n log n) → Sorting
O(2ⁿ) — exponential
O(n!) — worst ever.

Asymptomatic Notations

It is the mathematical tool which used to analyse the performance of algorithms by understanding how their efficiency changes as the input size grows.

- Notations provides a concise way to express the behaviour of an algorithm time and Space Complexity as the input size approaches infinity.

There are mainly three asymptomatic Notation:-

(i)

1. Big O Notation (O-notation) worst case (upper bound)
2. Omega Notation (Ω-notation) lower bound (Best)
3. Theta Notation (Θ-notation) Average case
   ↳ Exact time
   tight bound

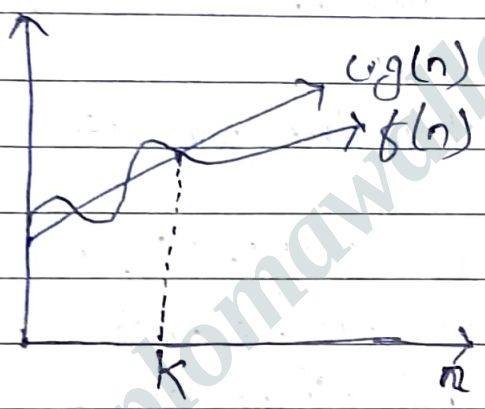
It focus on:-

(i) Compare algorithm

(ii) mathematical tool that estimates time based on input size.

(iii) focus on how many basic operation the program performs.

(1) Big O notation:-



$f(n) = Og(n)$

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

$c > 0$ → constant

$n \geq K$

$K \geq 0$

→ worst case

→ upper bound (Atmost)

{Atleast ? value ?}

let say: $f(n) = 2n^2 + n$

$f(n) = O(\ )$

$f(n) =$ order of ( )

what

$2n^2 + n \leq c \cdot g(n^2)$

↳ it should be bigger that's why we use big O square.

↳ upper bound which should be closest to left side $2n^2$ ↳ that's why we choose n^2.

if $c = 2$

$2n^2 + n \leq 2 \cdot n^2$ ✗

if $c = 3$

$2n^2 + n \leq 3 \cdot n^2$ ✓

$n \leq 3n^2 - 2n^2$

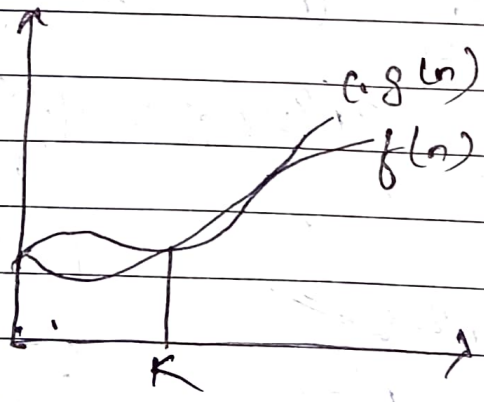
$n \leq n^2$

which is

$\boxed{n \geq 1}$

(condition valid for $c = 3$)

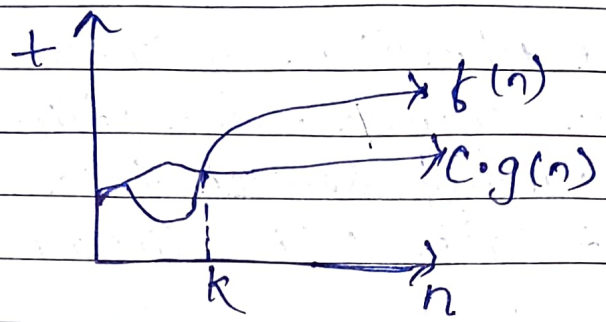
- It is a way to express the upper bound of an algorithm time or space complexity.
- Describes the asymptotic behaviour (order of growth time or space in term of input size) of a function, not its exact value.
- It provides an upper limit on the time taken by an algorithm in terms of the size of the input.
  We mainly consider the worst case scenario.
- It's denoted as O(f(n)), where f(n) is a function that represent the number of operation (steps) that an algorithm performs to solve a problem of size n.



Importance

1. measure Algorithm efficiency
2. Analyse Worse-case performance.
3. Ex - Binary Search take O(log(n)) in worst case. while linear Search in O(n).
4. Guides Better Code Design.
5. Predicts Scalability for large inputs.
6.

(11) Big-omega Notation.



→ Best case
→ Lower Bound (At least)
↳ we have to take greatest lower bound

$f(n) = \Omega g(n)$
$f(n) \geq C \cdot g(n)$

just opposite of big O notation.

let say $f(n) = 2n^2 + n$

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

$2n^2 + n \geq C \cdot n^2$ ← greatest lower value.

if we take $C = 2$ then.

$2n^2 + n \not\geq 2 \cdot n^2$ ↵

greater than R.H.S

So, all value of $n \geq 0$

Big omega Notation is a way to express the asymptotic lower bound of an algorithm's time complexity.

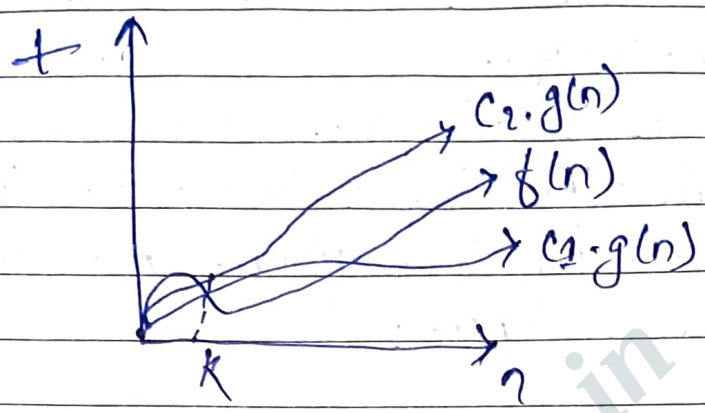
* It is always best-case situation of algorithm.
* It provide the lower limit on the time taken by algorithm in terms of the size of input.

* It is denoted as $\Omega(f(n))$, where $f(n)$ is a function that represent the number of operation (steps) that algorithm performs to solve a problem of size n.

* In simple word, Big-omega Ω when we want to represent that the algorithm will take at least a certain amount of time or space.

* Steps to determine:
  - → Break the program into smaller segment: Break the algorithm into smaller segment such that each segment has certain runtime complexity.
  - → Find the complexity of each segment.
  - → Add the complexity to all segment.
  - → Remove all the constant.

(iii) Theta Notation:— (Θ)



→ Average case
→ Exact time.

$$C_1 \cdot g(n) \leq f(n) \leq C_2 \cdot g(n)$$

$$2n^2 \leq 2n^2+n \leq 3n^2$$

let take example of copy notes:—

→ Big(O) is worst case means we find the content details in last page.
→ Ω-notation is best case means we find the content in first page which is easy.
→ Θ-notation → in this neither the topic in first or now in last it is in middle i.e average.

unit-02 completed
Pangam