DATE : __/__/____ PAGE NO.: _____

DSA :- Data structure Algorithm

⇒ Data :- Data is raw facts, figures or symbol that represent Information.

⇒ Data structure :- A way to store and organize data.
like : (arrays, linked-list, stack, queues, trees,)

⇒ Algorithm :- step-by-step procedure or logic to solve problems (searching, sharting, dynamic programing)

⇒ Together, DSA helps programmers design efficient, scalable and optimized solutions for real world problem.

⇒ T.C. :- (Time Complexity)

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

Note :- it is not equal to or not the actual execution time of the machine on which the algo. is running on.

Purpose :- many times there are more than one ways to solve a problem with different algo. and we need a way to compare multiple ways.

⇒ It allow us to evaluate and compare algo. based on efficiency, especially for large inputs.

⇒ To measure performance of algorithm, we typically use time & space Complexity Analysis. The idea is to measure measure "order-of-growth" in term of Inputs

DATE : __/__/____ PAGE NO.: _____

1.

Order-of-growth :- It is an approximation of How an algorithm's runtime scales with Input size n.

ex:- $4n^2 + 3n + 100 \rightarrow$ order of growth is n^2

⇒ In simple terms we convert the algo. or code into mathematical equation then we compare them:

- Comparison :-

constant < logarithemic < linear < quadratic < cubic < exponential < polynomical.

⇒ we choose the higher order one's

⇒ mathematical equation

(1) Decrement : eg : recursion

(2) Constant eq :- $y = 5$ (not depent on anything)

(3) logarithmic : $y = \log x + 5$ order of grath = $\log n$

(4) linear fun :- $y = 2x + 5$

(5) Quadritic fn :- $y = 2x^2 + 3x + 5$

(6) cubic :- $y = 2x^3 + 2x^2 + 3$

(7) exponential :- $y = 3^{2x} + 3$

(8) polynomical :- $y = x^{2x} + 2$

Note :- we are chosing the order of growth in for the large x (the quadric term $2x^2$ is dominate in quadritic fun.) in above example

PAGE NO.:

DATE : / /

2. How to Analyze an algo :-

a. Time :- we analyse the algo based on time fun.

b. space :- The second fun. we Analyse is space fun.

⇒ There are other factor like (Network consumption power Consumption and C.P.U. Regester)

⇒ we compare the Algo based on Time and space (memory) most of the time

2.1 Order of analysis :- There are some step to Analyse the Algo.

(a) frequency Count Method :- find the exact count of operation like = $T(n) = 3n^2 + 5n + 2$

(b) Asymptotic Notation :- simplify into Big O, Ω, Θ for easier Comparison like $T(n) = O(n^2)$

3. Frequency Count Method :- (FCM) : It is a mathmethic or mechinal way of finding the exact number of operation an algorithm performs.

eg :-

```
sum(a,b) {                          | Time Complexity
    s=0                             | ---->   1
    for(i=0; i<n; i++){             | ---->   n+1
        s = s + A[i];               | ---->   n
    }                               |
    return s;                       | ---->   1
}                                   |
```

Space Complexity :- A = n, S = 1, n = 1, i = 1

= O(1)

Total :- $f(n) = 2n+3$

T.C. = $O(n)$

DATE : __/__/____ PAGE NO.: _____

3.1 :- Purpose of frequency count method :-
(a) We count how many time each statement or loop run.
(b) To determine the precise running time expression Before simplification
(c) It is the first step to analyzing an Algorithm

4. Asymptotic Notation :- It is a mathematical framework used in computer science and mathematical to describe the growth rate of function based on inputs like Big O, Omega, theata etc.

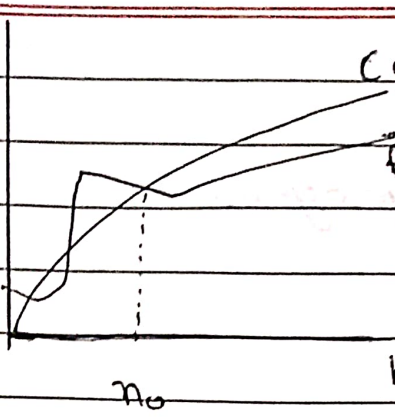
4.1 :- purpose of ~~frequency~~ Asymptotic notation :-
⇒ In simple term like we have Km, centemeter or Kg used for unite of different things and to compare just like that when we use the Time complexity compari we use Asymtotic unite

4.2 Type of Analysis / Asymptotic notation :-

(a) Big-O notation	O-notation	upper-bound
(b) Omega notation	Ω-notation	lower-bound
(c) Theta notation	Θ-notation	Average-bound

(a) Big-O-notation :- It represent the upper bound of the running time of an algo., Therefore it give the worst-case complexity of an algorithm
→ The maximum time required by an algo or the worst case time complexity
→ To return the highest possible output value

DATE : __/__/____ PAGE NO.: _____



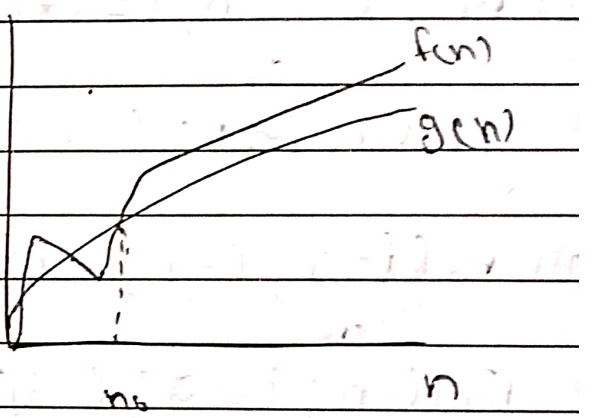
The fun. $f(n) = O(g(n))$ iff and only iff There exist (iff ∃) +ve Constant C and n_0 such that :-

$$f(n) \leq C * g(n) \quad \forall \quad n \geq n_0$$

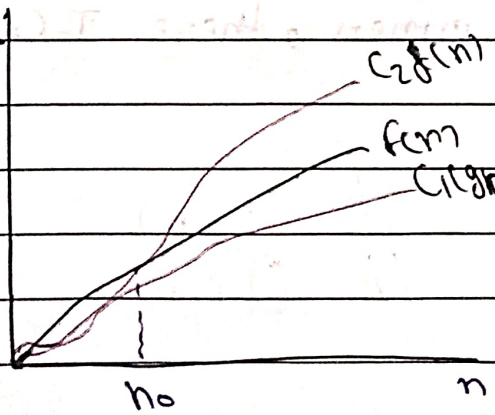
(b) Omega :- it represent the lower bound of the running time of an algo. Thus provide the best case complexity of an algo.

→ The fun. $f(n) = \Omega g(n)$ iff ∃ +ve constant C and n_0 such that

$f(n) \geq C * g(n) \quad \forall \; n \geq n_0$



(c) Theta notation :- It enclose the function from above and below, it represent both upper and lower so it is average



The function $f(n) = \Theta g(n)$ if and only if there exist +ve Constant C_1 and C_2 and n_0 such that

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

DATE : __/__/____ PAGE NO.: ____

Extra notes & tips :-

⇒ we mostly encounter loops in a program :-

⇒ categaries of loops	estimated frq. count	T.C.
(I) for(i=1; i≤n; i=i+c) or for(i=n; i≥1; i=i-c)	= n/c Times	Tc = O(n/c)
II) for(i=1; i≤n; i=i*c) or for(i=n; i≥1; i=i/c)	= $\log_c n$ Times	Tc = $O(\log_c n)$
III) for(i=a; i≤n; i=i^c) or for(i=n; i≥a; i=$\sqrt{i}$)	= $\log_c \log_a n$ Times	Tc = $O(\log_c \log_a n)$

⇒ Dependent & Independent loop :-

1) Independent loop :- loop that do not rely on each orther They can be

(a) sequential :- one loop after esther, there T.C. will be add

ex:-

```
for( i=0; i<n; i++) { }    // O(n)
for( i=0; i<m; i++) { }    // O(m)
```

= Time complexity = O(m+n)

DATE : / /

PAGE NO.:

(b) nested :- one loop inside other but don't rely on each other, there T.C. will be multiply.

ex :-
```
for ( i = 0 ; i < n ; i++ ) {          // O(n)
    for ( j = 0 ; j < m ; j++ ) { }    // O(m)
}
```
- Time complexity = $O(m \cdot n)$

2. Dependent loop :- loops where the inner loop's bound dependent on the outer loop's varable

⇒ This creates growth patterns that aren't fixed you must analyze how the dependency scales.

⇒ mostly we get the A.P and g.p in these kinde of loop and the T.C. deppend on the Inner loop