→ Complexity of Algorithms

* Analysis of algorithm is important task in CS.
* Have some criteria to measure efficiency of algo.
  - space
  - time

* Time is measured by counting no. of key operations
  for ex: In searching and sorting, no. of comparisons
  Time is proportional to no. of key operations.

* Space is measured by counting max. of memory required by algorithm.

* Complexity of an algo 'A' is function f(n) which gives running time / storage space requirement of algorithm in terms of input data size 'n'.

* Storage required is simply in multiple of 'n', so, we use term complexity to refer to running time of the algorithm.

1. Worst Case :- maximum value of f(n) for any input
2. Average Case :- expected value of f(n)
3. Best Case :- minimum possible ~~of~~ value of f(n).

Example :- Algo. to perform linear search.

A linear array DATA with N elements ~~are given~~ and specific ITEM are given. This algo finds location LOC of ITEM in the array DATA or sets LOC = 0 if not found.

1. Set K=1 and LOC=0
2. Repeat 3 & 4 while LOC = 0 and K ≤ N
3. If ITEM = DATA [K] then
4. Set LOC = K ;

   Set K = K+1

   [End of step 2 loop]

5. If LOC = 0 then

   Write : ITEM is not in the DATA

   Else :

   Write : LOC is location of DATA.

6. Exit.

* **Worst Case.** occurs when ITEM is the last element of DATA or is not there at all in array.

$C(n) = n$ — worst case complexity of linear search.

* **Average case** :- ITEM appear in DATA and is equally likely to appear at any position.

No. of comparison can be any of 1, 2, 3, ---, n. and each occur with probability 1/n.

$$C(n) = 1 \cdot \frac{1}{n} + 2 \cdot \frac{1}{n} + 3 \cdot \frac{1}{n} + \cdots + n \cdot \frac{1}{n}$$

$$= \frac{1}{n}(1+2+3+4+\cdots+n)$$

$$= \frac{\cancel{n}(n+1)}{2} \cdot \frac{1}{\cancel{n}} = \frac{n+1}{2}$$

Note: Average case complexity of an algo is usually much more complicated to analyze than worst case. Also, probabilistic distribution we assume for average case may not actually apply in real situation.

* Rate of Growth : Big O Notation

Let M = algorithm n = size of input data
$f(n)$ increases as 'n' increases. Thus, it is rate of increase of $f(n)$ that we want to examine.

Ex:- $\log_2 n$ $n\log_2 n$ n^2 n^3 2^n

n \ $g(n)$	$\log n$	n	$n\log n$	n^2	n^3	2^n
5	3	5	15	25	125	32
10	4	10	40	10^2	10^3	10^3
100	7	100	700	10^4	10^6	10^{30}
1000	10	1000	10000	10^6	10^9	10^{300}

Rate of Growth of Standard functions.

Suppose $f(n)$ and $g(n)$ are functions defined on positive integers with property that $f(n)$ is bounded by some multiple of $g(n)$ for almost all 'n'.

i.e.

There exist a positive integer n_0 and a positive no. 'k' such that for all $n > n_0$ –

$$|f(n)| \le k \cdot |g(n)| \quad \text{i.e. } f(n) = O(g(n))$$

For any polynomial of degree 'm', we get -

$$P(n) = O(n^m)$$

Ex :-

$$8n^3 - 576n^2 + 832n - 248 = O(n^3)$$

Linear Search :	$O(n)$
Binary Search :	$O(\log n)$
Bubble Sort	$O(n^2)$
Merge Sort	$O(n \log n)$.

* **Omega Notation** (Ω).

This used when the function $g(n)$ defines a lower bound for the function $f(n)$.

$f(n) = \Omega\, g(n)$ iff

There exist a positive integer n_0 and a positive no. 'K' such that -

$$f(n) \geq K \cdot g(n). \text{ for all } n > n_0.$$

Ex :- $f(n) = 18n + 9$

$f(n) > 18 \cdot n$ for all 'n'

$f(n) = \Omega(n)$,

Also : $f(n) = 5n + 1$

$f(n) = \Omega(n)$

$f(n) = \Omega(1) \rightarrow$ we never consider latter.

as. $f(n) = \Omega(n)$ represents the largest possible function of 'n' satisfying the definition of Ω.

* Theta Notation

$f(n)$ is bounded from both above and below by function $g(n)$

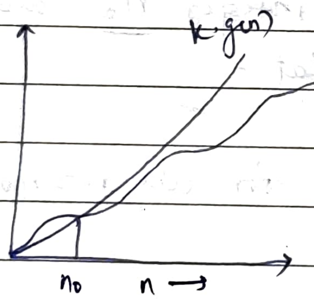
$f(n) = \Theta(g(n))$ iff there exist two +ve constants k_1 & k_2 and a positive integer n_0 such that:

$$k_1\, g(n) \le f(n) \le k_2\, g(n) \quad \text{for all } n \ge n_0$$

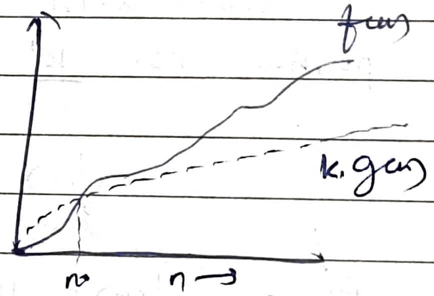
Ex: $f(n) = 18n + 9$

$f(n) \ge 18n$, $f(n) \le 27n$ for $n \ge 1$

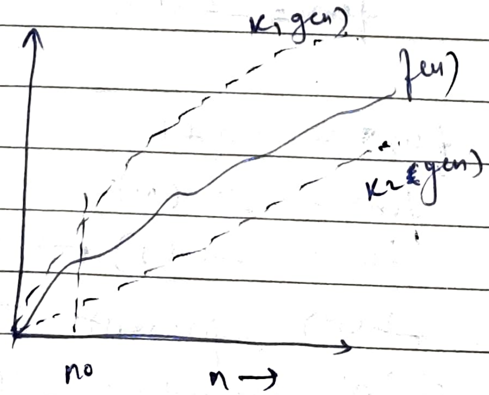
$f(n) = \Theta(n)$.



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



$f(n) = \Omega(g(n))$.



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