CS & IT ENGINEERING

Algorithm

Analysis of Algorithms

Lecture No.- 04

By- Aditya sir

Recap of Previous Lecture

Topic Apriori Analysis

step count method

Order of Magnitude

Topic Types of Analysis

B.C, W.C, A.C

Topic Worst-case and Best-Case Behavior

Topics to be Covered

Topic Asymptotic Notations

Topic Big-Oh, Big Omega, Theta Notations

Topic : Asymptotic Notations

input size

T(n) ✓ → Step Count Mtd

✓ ① $4n^2 + 8n + 6$

✓ ② $n^2 + n + 1$

Orders of magnitude

Time complexity (n) → T(n) → ① ②

Polynomial ✓

Exponential

log

Const (n^0)

n → linear

n^2 → Quadratic

n^3 Cubic

a^n ($a > 1$)

ASN → Suitable notation to represent time complexity

Topic : Asymptotic Notations

* Asymptotic Notations (ASN)

→ Mathematical tool to obtain/represent the $\boxed{\text{bounds}}$ of a function

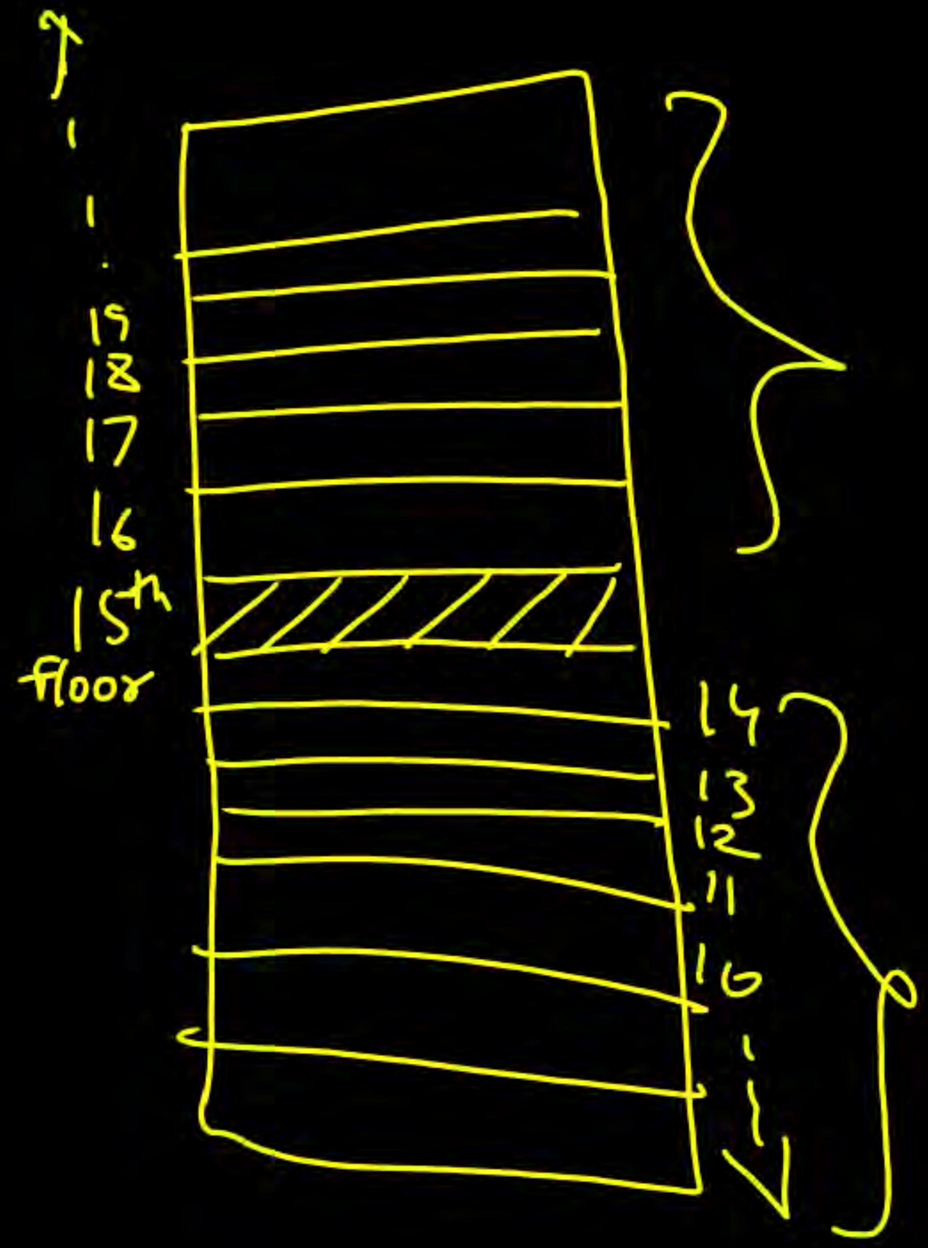
Upper bound
⟨Max⟩

Lower bound
(Min)

Tight bound

$$\text{ASN} \rightarrow \text{UB}$$
$$\text{ASN} \rightarrow \text{LB}$$

Helping us to represent Time and Space Complexity of an Alg. by Functions



ASN (5) $\rightarrow$ $\underbrace{O, \Omega}_{\text{Big}}, \Theta, \underbrace{o, \omega}_{\text{Small}}$

Big

Upper bound $\leftarrow$ Big Oh : O

Lower bound $\leftarrow$ Big Omega : Ω

Tight bound $\leftarrow$ Theta : Θ

Small / little

little Oh : o $\rightarrow$ proper Upper bound

little Omega : ω $\rightarrow$ proper Lower bound

→ let 'f' and 'g' be the functions from the set of integers/real numbers to real numbers

① Big Oh: (O) : Upper bound

f(n) is O(g(n)) iff there exists some constants $c > 0$ & $n_0 > 0$

Such that $f(n) \leq c \cdot g(n)$ whenever $n \geq n_0$

$\boxed{f(n) = O(g(n))}$

or $f(n) \in O(g(n))$

eg1:-

$$1 \le n^2$$

$$1+n \le n^2+n^2$$

$$\boxed{\underbrace{1+n+n^2}_{f(n)} \le \underbrace{3}_{c}\, \underbrace{n^2}_{g(n)}}$$

$$\underbrace{1+n+n^2} \le \underbrace{n^2+n^2+n^2} \longrightarrow \begin{array}{l} L: 1+1+1=3 \\ R: 1+1+1=3 \end{array}$$

$$f(n) \le c \cdot g(n),\ n \ge n_0$$

$$f(n) \le 3n^2,\ \underbrace{n \ge 1}_{n_0}$$

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

$1+n+n^2 = O(n^2)$

Closest upper bound

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

$f(n) = O(n^3)$

$f(n) = O(n^4)$

$\vdots$

$\underbrace{1+n+n^2}_{f} \leq \underbrace{3}_{c} * \underbrace{n^2}_{g}, \quad n \geq \underbrace{1}_{n_0}$

$f(n) \leq c \cdot g(n), \quad n \geq n_0$

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

Imp. whenever we are determining the UB & LB, we should find that function 'g' which is closest to the given function 'f'.

eg1 :

GMaps : 25 mins

P1 : $<$ 1 hr

P2 : $<$ 1 Day

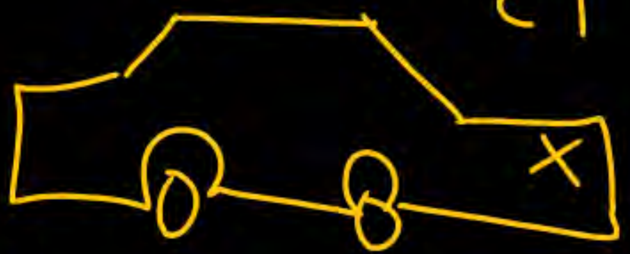
eg2

P1. 15 - 25L

LB UB

Showroom

C1



P2: > 100 Rs

P3: 10 cr

Specification:

1. Automatic
2. 5 Seater
3. Accessories
   - → AC
   - → Music System
   - → etc

dominating

$$f(n) = \underbrace{1+n}+n^2 = O(n^2)$$

$$1+n+n^2 \leq n^2+n^2+n^2$$

$$\boxed{1+n+n^2 \leq 3n^2}$$

Why can we ignore the lower order terms?

Business man :-

Investments

1. Walk → 0 Rs
2. Cycle → 15 K Rs
3. Bike → 1 L Rs
4. Car → 30 L Rs
5. Jet → 100 Cr Rs

→ O(Crs)

Topic : Asymptotic Notations

(2) Big Omega (Ω): Lower bound

→ $f(n)$ is $\Omega(g(n))$ iff there exists some positive constants c & n_0

such that $f(n) \boxed{\geq} c * g(n)$, whenever $n \geq n_0$

Lower bound

$f(n) = 1+n+n^2$

$\Omega(1)$

$\Omega(n)$

$\Omega(n^2)$

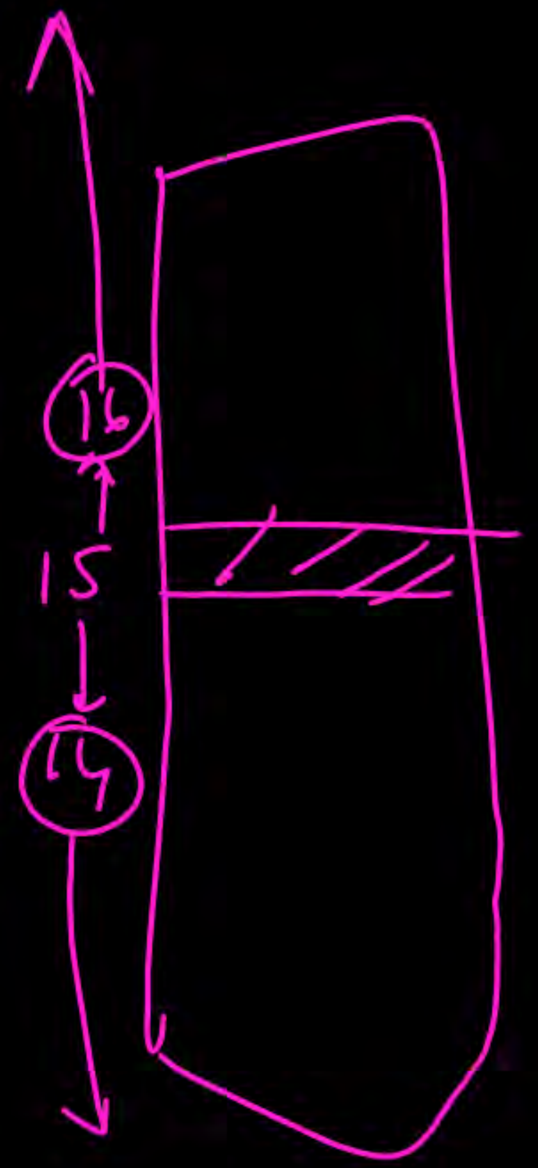
Closest LB

$1+n+n^2 \geq \underset{c}{1} * 1 \;,\; n \geq 1$

$1+n+n^2 \geq \underset{c}{1} * n \;,\; n \geq 1$

$1+n+n^2 \geq 1 * n^2 \;,\; n \geq 1$

$c = 1$



$$\underbrace{\overset{c}{\underline{1}} * n^2 \le 1 + n + n^2}_{} \le \overbrace{\underset{c}{\underline{5}} * n^2}$$

$O(n^2)$

Closest UB

$\Omega(n^2)$

Closest LB

③ Theta (Θ): Tight Bound

→ $f(n)$ is $\Theta(g(n))$ iff

$f(n)$ is $O(g(n))$

AND $f(n)$ is $\Omega(g(n))$

eg:- $1+n+n^2$ → $O(n^2)$ & $\Omega(n^2)$ → $\Theta(n^2)$

$$c1 * g(n) \le f(n) \le c2 * g(n)$$

eg:

$$1 * n^2 \le 1+n+n^2 \le 5 * n^2$$

Topic : Analysis of Algorithms

- Big O notation is a mathematical notation that describes the limiting behavior of a function when the argument tends towards a particular value or infinity.
- Big O is a member of a family of notations invented by Paul Bachmann, Edmund Landau, and others, collectively called Bachmann-Landau notation or asymptotic notation. The letter O was chosen by Bachmann to stand for Ordnung, meaning the order of approximation.
- In computer science, big O notation is used to classify algorithms according to how their run time or space requirements grow as the input size grows. n
- In analytic number theory, big O notation is used to express a bound on the difference between an arithmetical function and a better understood approximation; a famous example of such a difference is the remainder term in the prime number theorem.

Topic : Analysis of Algorithms

- Big O notation is also used in many other fields to provides similar estimates.
- Big O notation characterizes functions according to their growth rates: different functions with the same asymptotic growth rate may be represented using the same O notation. The letter O is used because the growth rate of a function is also referred to as the order of the function. A description of a function in terms of big O notation usually only provides an upper bound on the growth rate of the function.
- Associated with big O notation are several related notations, using the symbols O, Ω, ω and θ, to describe other kinds of bounds on asymptotic growth rates.

Topic : Analysis of Algorithms

Big Oh (O)

Definition: A theoretical measure of the execution of an algorithm, usually the time or memory needed, given the problem size n, which is usually the number of items. Informally, saying some equation $\boxed{f(n) = O(g(n))}$ means it is less than some constant multiple of g(n). The notation is read, "f of n is big oh of g of n".

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

$c > 0,\ n_0 > 0$

Formal Definition: $\boxed{f(n) = O(g(n))}$ means there are $\boxed{\text{positive}}$ constants c and k, such that $(0 < f(n) \leq cg(n)$ for all $n \geq K$. The values of c and k must be $\boxed{\text{fixed}}$ for the function f and must not depend on n.

$n \geq n_0$

Topic : Analysis of Algorithms

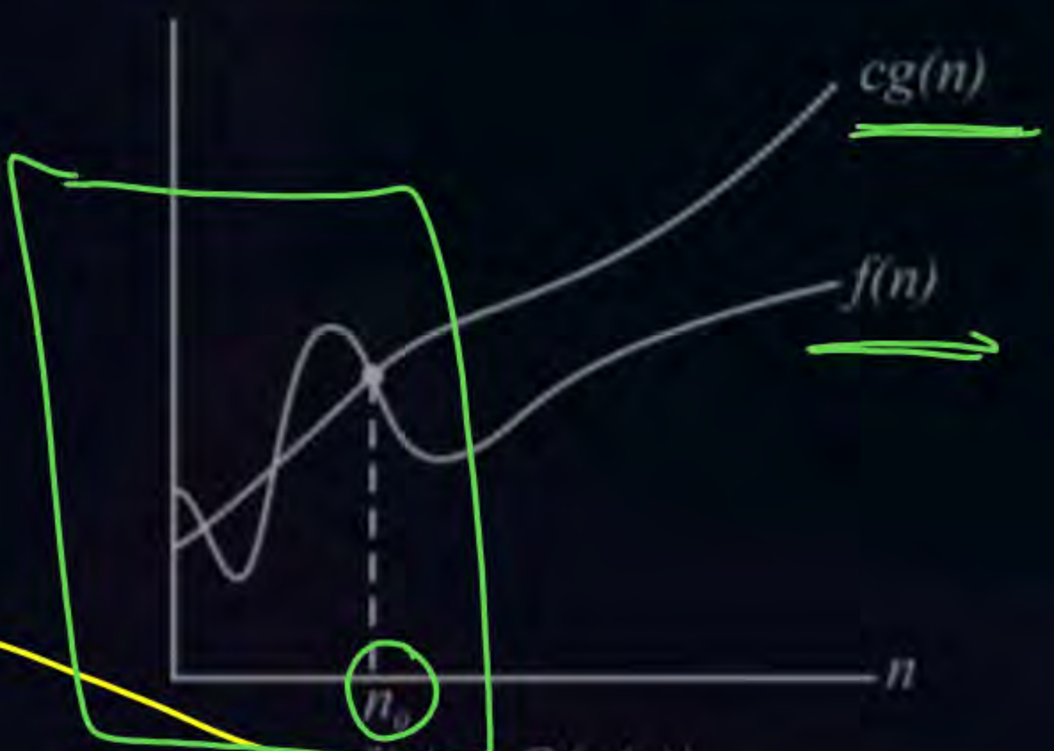
- The formal definitions associated with the Big Notation are as follows:
- · $f(n) = O(g(n))$ means $c.\ g(n)$ is an upper bound on $f(n)$. Thus there exists some constant c such that $f(n)$ is always $\leq c.g(n)$, for large enough n (i.e., $n \geq$ no for some constant n_o).
- · $f(n) = \Omega(g(n))$ means $c.g(n)$ is a lower bound on $f(n)$. Thus there exists some constant c such that $f(n)$ is always $\geq c.\ g(n)$, for all $n \geq n_o$.

Topic : Analysis of Algorithms

$O(g(n)) = \{f(n)$: There exist positive constant c and n_0 such that $0 \le f(n) \le cg(n)$ for all $\boxed{n \ge n_0}\}$.

We write $f(n) = O(g(n)$ to indicate that a function $f(n)$ is a member of the set $O(g(n))$. Note that $f(n) = \theta(g(n))$ implies $f(n) = O(g(n))$. ~~Since θ-notation is a stronger notation that O-notation. Written set-theoretically.~~

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



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

$n^2 \le 2^n$

n	f (n^2)	g (2^n)
1	1	2
2	4	4
3	9	8
4	16	16
5	25	32
6	36	64
7	49	128

Big-Omega Notation (Ω):

Similar to big O notation, big Omega (Ω) function is used in computer science to describe the performance or complexity of an algorithm. If a running time is Ω (f(n)), then for large enough n, the running time is at least k. f(n) for some constant k.

Topic : Analysis of Algorithms

Set

$O(n^3)$

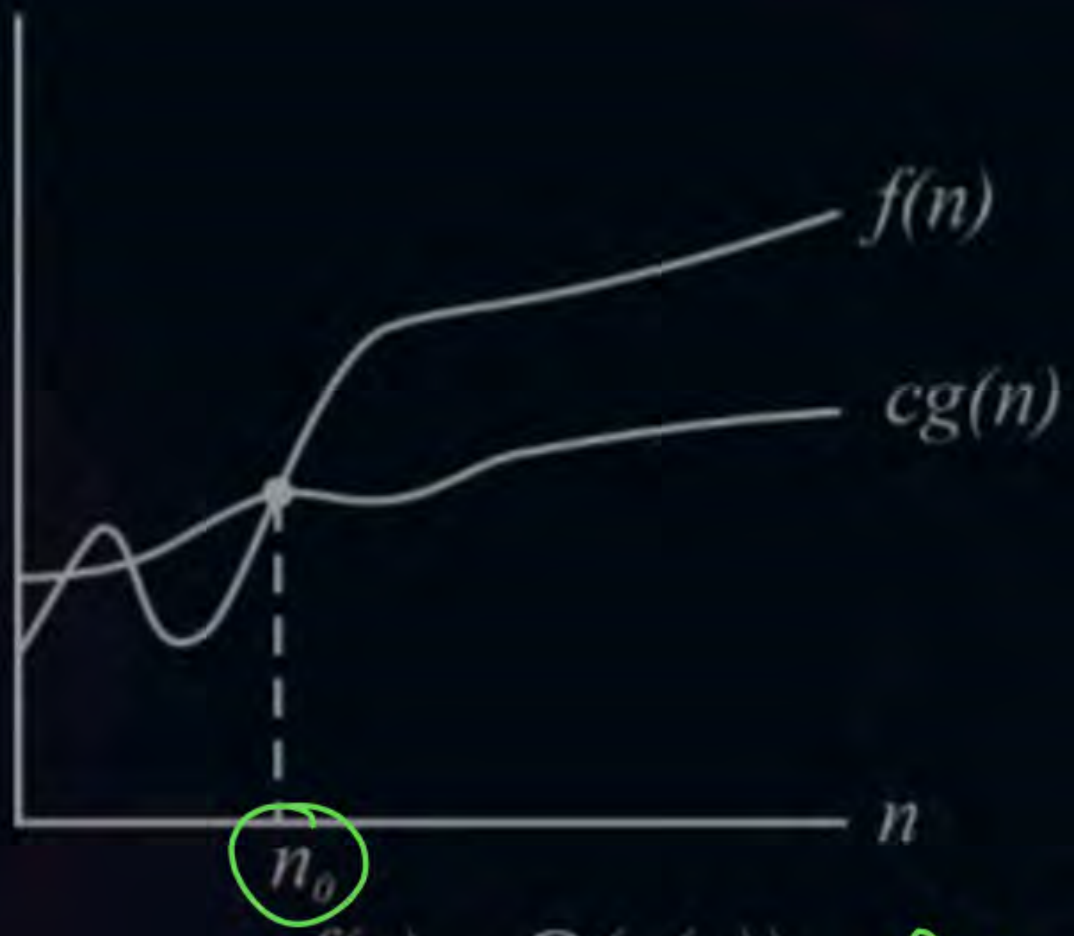
c $\quad n\log n$

n $\quad n^2\log n$

n^2

$\log n$ $\quad \sqrt{n}\log n$

$\sqrt{n}$



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

$n \geq n_0$

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

$n \geq n_0$

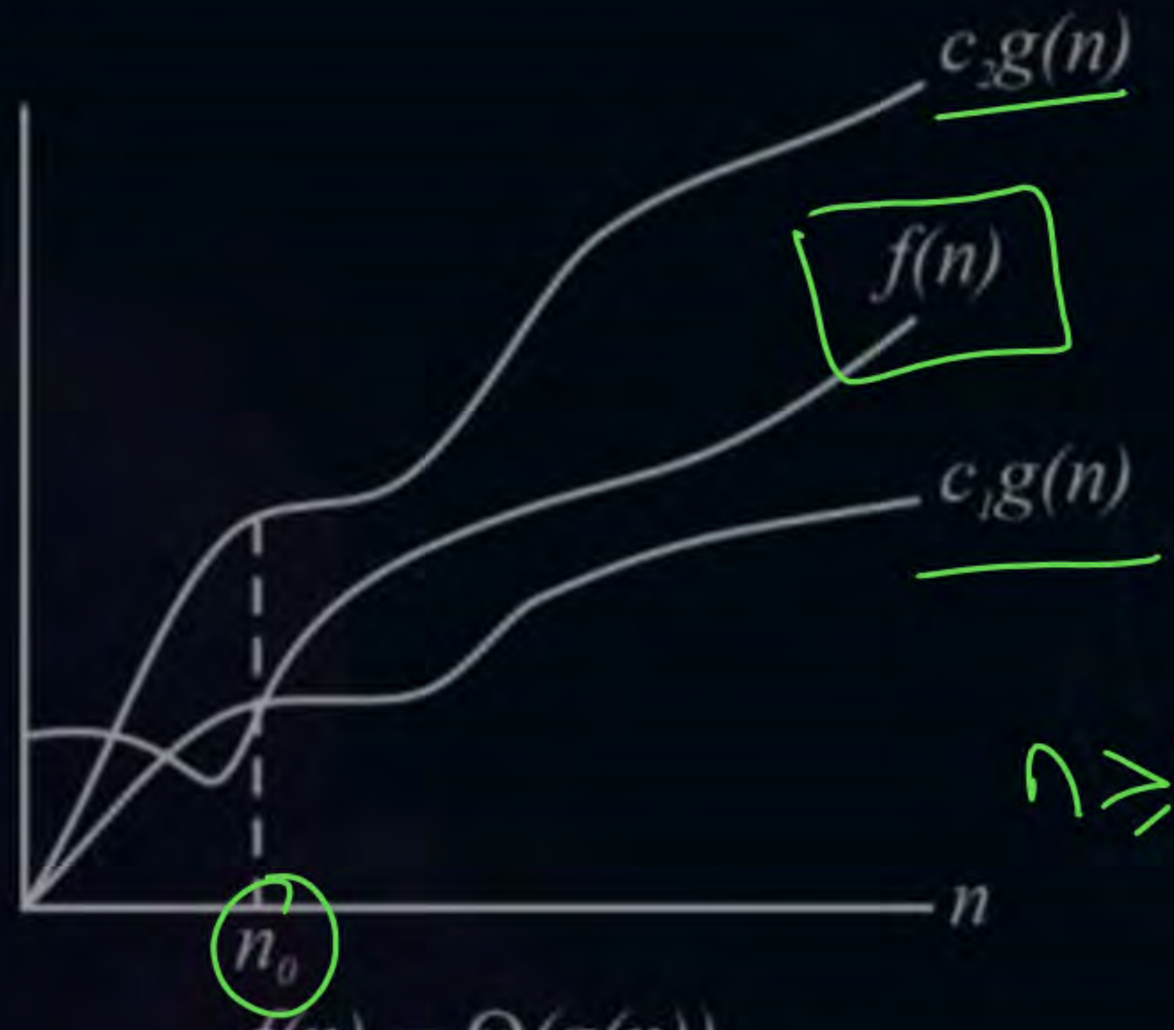
$f(n)$ is $\Omega(g(n))$

Topic : Analysis of Algorithms

- $f(n) = \theta(g(n))$ means $c_1 . g(n)$ is an upper bound on $f(n)$ and $C_2 . g(n)$ is lower bound on $f(n)$, for all $\boxed{n \geq no}$. Thus there exists constant C_1 and C_2 such that $f(n) \leq C1.g(n)$ and $f(n) \geq C_2 . g(n)$. This means that $g(n)$ provides a nice, tight bound on $f(n)$.

$$\boxed{C_2 \, g(n) \leq f(n) \leq C1 . g(n)}$$

Topic : Analysis of Algorithms



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

$f(n) \le c_2\, g(n)$

&

$f(n) \ge c_1 \cdot g(n)$

$n \ge n_0$

Topic : Analysis of Algorithms

Practice Questions:-

① $f(n) = 1 + n + n^2$ → $O(n^2)$, $\Omega(n^2)$ → $\theta(n^2)$

② $f(n) = n$ → $O(n)$, $\Omega(n)$ → $\Theta(n)$

③ $f(n) = 3^{100}$ → $O(1)$, $\Omega(1)$ → $\Theta(1)$

$2^{100} \le 3^{100} * 1$

$2^{100} \ge 1^{100} * 1$

$n + \log n \le 2 \cdot n$

$n + \log n \ge 1 * n$

$\boxed{\log n \le n}$

$n \le n$

$n + \log n \le n + n$

④ $f(n) = n + \log(n)$ → $O(n)$, $\Omega(n)$ → $\Theta(n)$

⑤ $f(n) = \sqrt{n} + \log(n)$ → $O(\ )$, $\Omega(\ )$

$n \le c_1 * n$, $n \ge 1$

$c_1 = 1/2$ $n \le n$ $c_1 = 1$ $c_1 = 5$

$n \ge 1/2\, n$

> 0

$$f(n) = \underbrace{\sqrt{n}} + \underbrace{\log(n)} \;\begin{cases} \to O(\sqrt{n}) \\ \to \Omega(\sqrt{n}) \end{cases} \longrightarrow \Theta(\sqrt{n})$$

$\sqrt{n}$ $\qquad$ $\log(n)$

$n = 64$ $\qquad$ 8 $\qquad$ $\log_2(64) \longrightarrow 6$

$\sqrt{n}$ $\qquad$ $\log(n)$

$n^{1/2}$ $\qquad$ $\log(n)$

$\log(n^{1/2})$ $\qquad$ $\log(\log(n))$

$\frac{1}{2} \times \log(\underline{n}) \geq \log(\underline{\log(n)})$

THANK - YOU