Asymptotic Notation:- Asymptotic Notation of an algorithm is a mathematical representation of its complexity.

Note:- When we want to represent the complexity of an algorithm, we use only the most significant terms in the complexity of that algo. and ignore least significant terms.

Eg. Algo 1: $5n^2+2n+1$ } time complexities of
Algo 2: $10n^2+8n+3$ } two algo.

When we analyze an algo., we consider the time complexity for larger values of input data (i.e. n value).

$2n+1$ has least significance than $5n^2$
$8n+3$ " " " " $10n^2$

So we can ignore the least significant terms to represent overall time required by an algo.

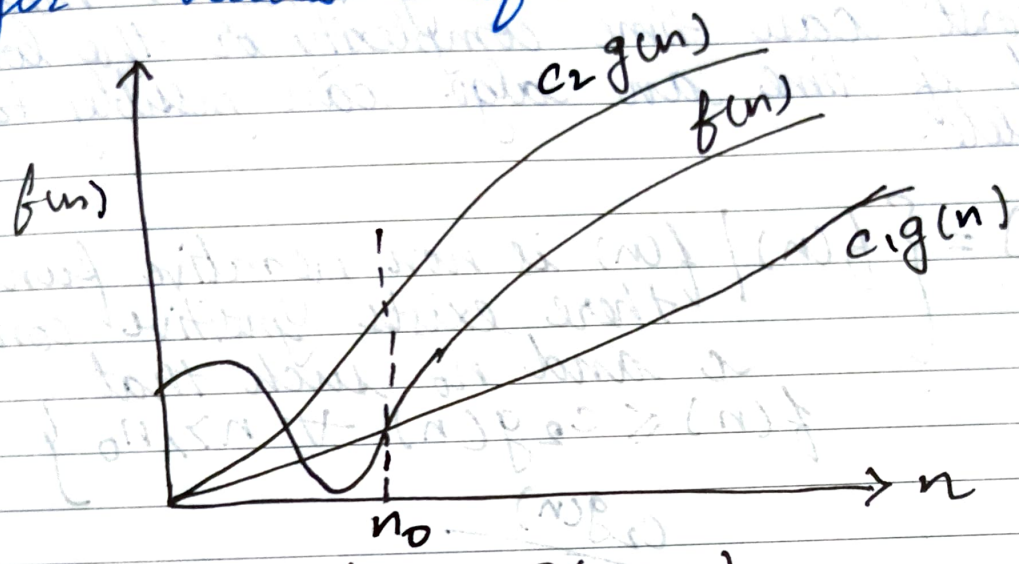
Theta Notation (θ) - It is the formal way to express both the lower bound and upper bound of an algo's running time. (Define the average bound)

f, g - non negative functions.

$\theta(g(n)) = \{ f(n) \mid f$ is non-negative function such that $\exists$ two constants c_1, c_2, n_0 such that

$c_1 g(n) \leq f(n) \leq c_2 g(n) \quad \forall\ n \geq n_0 \}$.

If $f(n)$ is theta of $g(n)$, then the value $f(n)$ is always b/w $c_1 g(n)$ and $c_2 g(n)$ for larger values of n ($n \geq n_0$).



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

→ **Properties –**

1) $f(n) \in \theta(g_1)$, $h(n) \in \theta(g_2)$

$f(n) + h(n) \in \theta(g_1 + g_2)$

2) If $g = g_1 = g_2$

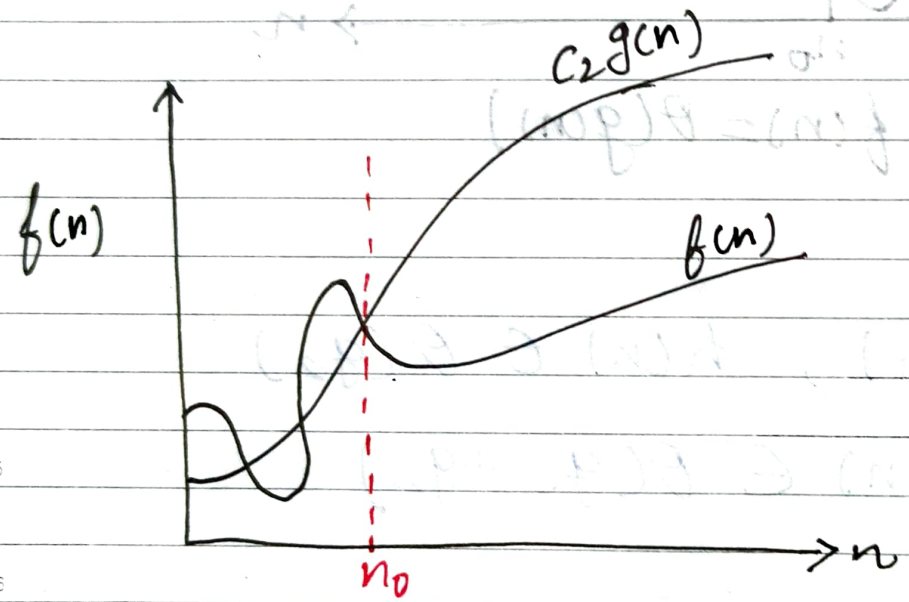
$f(n) + h(n) \in \theta(g)$

Note :–

Suppose $f \in \theta(g)$
Common style $f = \theta(g) \Rightarrow f(n) = \theta(g(n))$
we can not write $\theta(g) = f$ x

2> **Big Oh Notation (O)** – It is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity or the longest amount of time an algo. can possibly take to complete.

$$O(g(n)) = \{ f(n) \mid f(n) \text{ is non negative function there exists positive constants } c \text{ and } n_0 \text{ such that } f(n) \leq c_2 g(n) \ \forall \ n \geq n_0 \}$$



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

3> **Omega Notation (Ω)** – It is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity or the best amount of time an algorithm can possibly take to complete.

$\Omega(g(n)) = \{ f(n) \mid f(n)$ is non negative function $\exists$ positive constants c_1 and n_0 such that $c_1 g(n) \leq f(n) \ \forall \ n \geq n_0 \}$

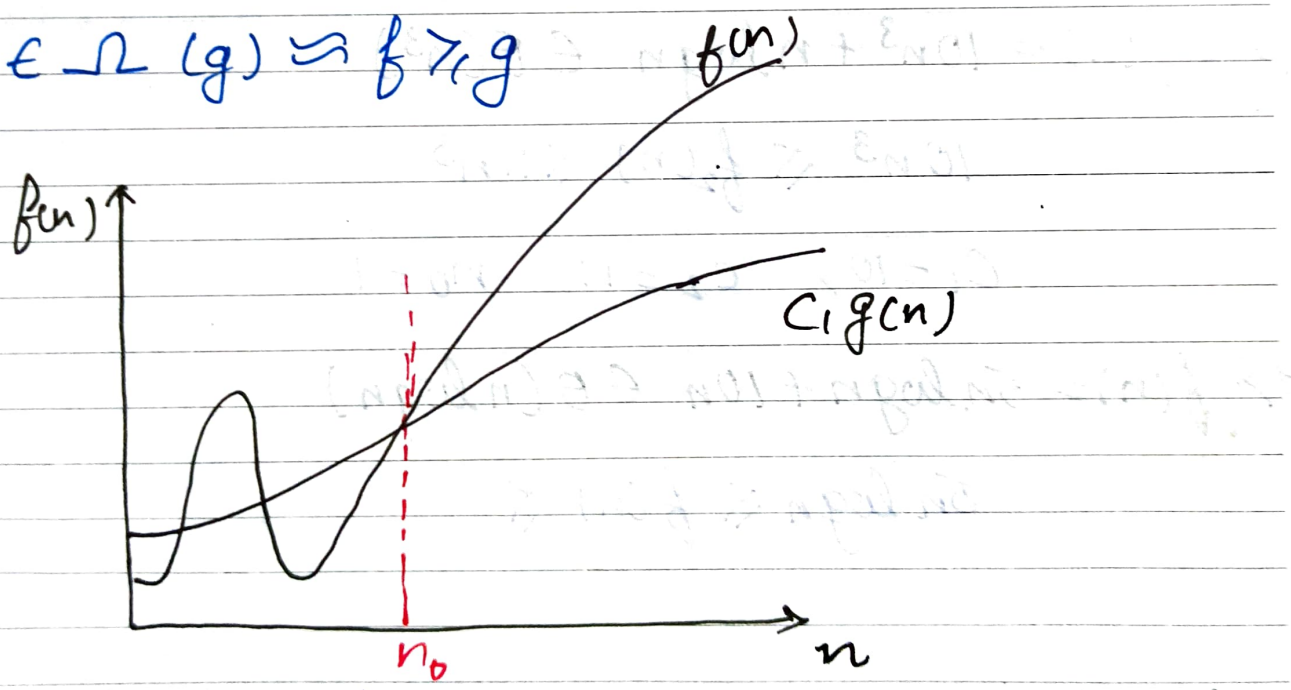
→ Properties of all:-

1) $f \in \Theta(g) \Rightarrow f \approx g$

2) $f \in O(g) \Rightarrow f \leq g$

3) $f \in \Omega(g) \Rightarrow f \geq g$

4) $f(n) = \Theta(g(n)) \Leftrightarrow f = O(g(n))$ and $f = \Omega(g(n))$



Transitivity → $f(n) = \Theta(g(n))$ and $g(n) = \Theta(h(n))$
$\Rightarrow f(n) = \Theta(h(n))$
Same for O & Ω

Reflexive → $f(n) = \Theta(f(n))$ same for O & Ω

Symmetry → $f(n) = \Theta(g(n))$ iff $g(n) = \Theta(f(n))$

⇒ Examples-

1) $f_1(n) = 10n^3 + 5n^2 + 17 \in \theta(n^3)$

→ $c_1 g(n) \leq f(n) \leq c_2 g(n)$

$10n^3 \leq f_1(n) \leq (10+5+17)n^3 = 32n^3$

$c_1 = 10, \quad c_2 = 32$

$c_1 n^3 \leq f_1(n) \leq c_2(n) \quad \forall \; n \geq 1 = n_0$

2) $f_2(n) = 10n^3 + n\log n \in \theta(n^3)$

$10n^3 \leq f_2(n) \leq 11n^3$

$c_1 = 10, \quad c_2 = 11 \quad n_0 = 1$

3) $f(n) = 5n\log n + 10n \in \theta(n\log n)$

$5n\log n \leq f(n) \leq$

24 Sunday

4) $f(n) = 2 + 1/n = \theta(1)$ → (class of constant and minor variables.

$2 \leq 2 + 1/n = f(n) \leq 3$

$c_1 = 2 \quad c_2 = 3 \quad n_0 = 1$

5) $T(n) \leq 15n^3 + 7n^2 + 35 \leq 57n^3$

$T(n) = O(n^3)$ —— (I)

$T(n) \geq 2n^3 + 37 \geq 2n^3$

$T(n) = \Omega(n^3)$ —— (II)

from (I) & (II) $T(n) = \Theta(n^3)$

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

6) $f(n) = 3n^2 \in O(n^2)$

7) $f(n) = 10n^3 + 5n + 7 \in O(n^3)$

8) $f(n) = 10n^3 + 5n + 7 \in O(n^4)$

1) $f(n) = n$ $g(n) = \log n^2$ $f(n) = \Omega(g(n))$

2) $f(n) = 10$ $g(n) = \log 10$ $f(n) = \Theta(g(n))$

3) $f(n) = 2^n$ $g(n) = 3^n$ $f(n) = O(g(n))$

4) $f(n) = 2^n$ $g(n) = n^2$ $f(n) = \Omega(g(n))$

$\log 1 = 0$

$\log 0 =$ not a number

$\Omega = c_1 g_n \leq f(n)$

$\log n^2 \leq n$

$2 \log n \leq n$

$0 \leq 1$

$c_1 g_n \leq f(n) \leq c_2 g(n)$

$\log 10 \leq 10 \leq \log 10$

$1 \leq 10 \leq$

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