Prepared by,

Chetan Shetty

Assistant Professor

Department of CSE

MSRIT

Bangalore

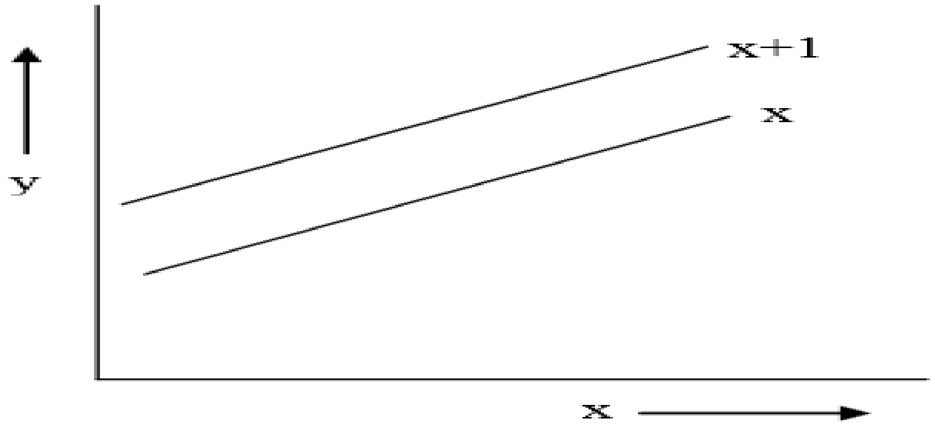
Asymptotic Notation

TIME COMPLEXITY

Total time required for completion of solving a problem is equal to sum of compile time and running time. To execute a program, always it is not mandatory that it must be compiled. Hence running time complexity will be under consideration for an evaluation and finding an algorithm complexity Analysis.

ASYMPTOTE

- An asymptote is a line or curve that a graph approaches but does not intersect.
- An asymptote of a curve is a line in such a way that **distance between curve and line approaches zero** towards large values or infinity.
- **Ex**: x is asymptotic to x+1 and these two lines in the graph will never intersect



- The concept of asymptote will help in understanding the behavior of an algorithm for large value of input.

Bounds

- Bounds will be useful to understand the asymptotic notations.
- **Lower Bound:** A non empty set A and its subset B is given with relation ≤. An element a A is called lower bound of B if a ≤ x ᵾ x Ɛ B (read as if a is less than equal to x for all x belongs to set B).
  - **Example:** A non empty set A and its subset B is given as A={1,2,3,4,5,6} and B={2,3}. The lower bound of B= 1, 2 as 1, 2 in the set A is less than or equal to all element of B.

- **Upper Bound:** An element a A is called upper bound of B if x≤a ᵾ x Ɛ B.
  - For example a non empty set A and its subset B is given as A={1,2,3,4,5,6} and B={2,3}.
  - The upper bound of B= 3,4,5,6 as 3,4,5,6 in the set A is greater than or equal to all element of B.

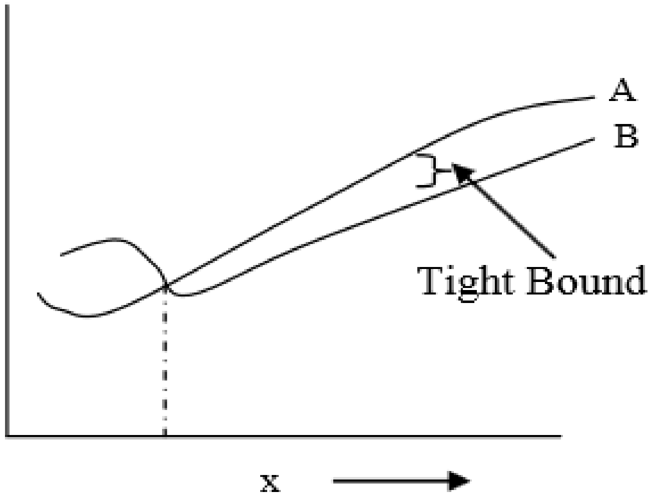


Figure:- 3 Tight Bound

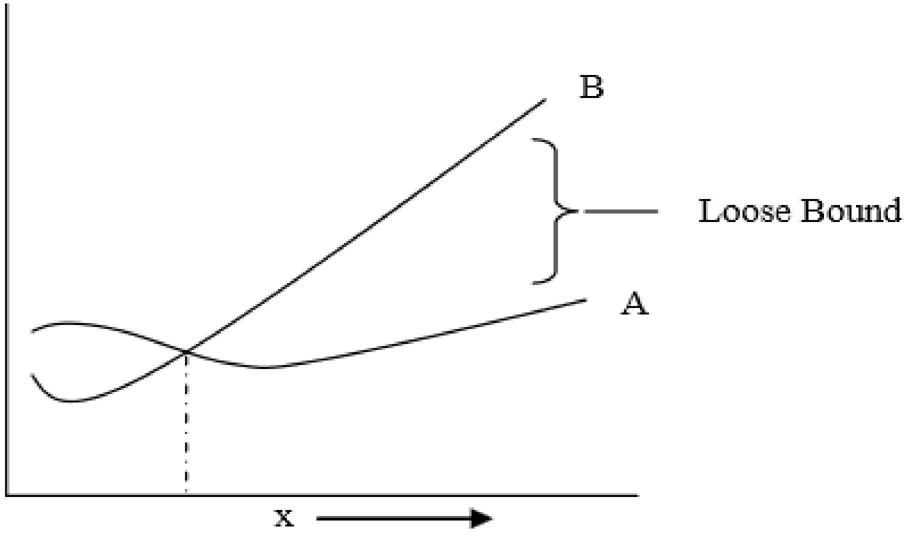
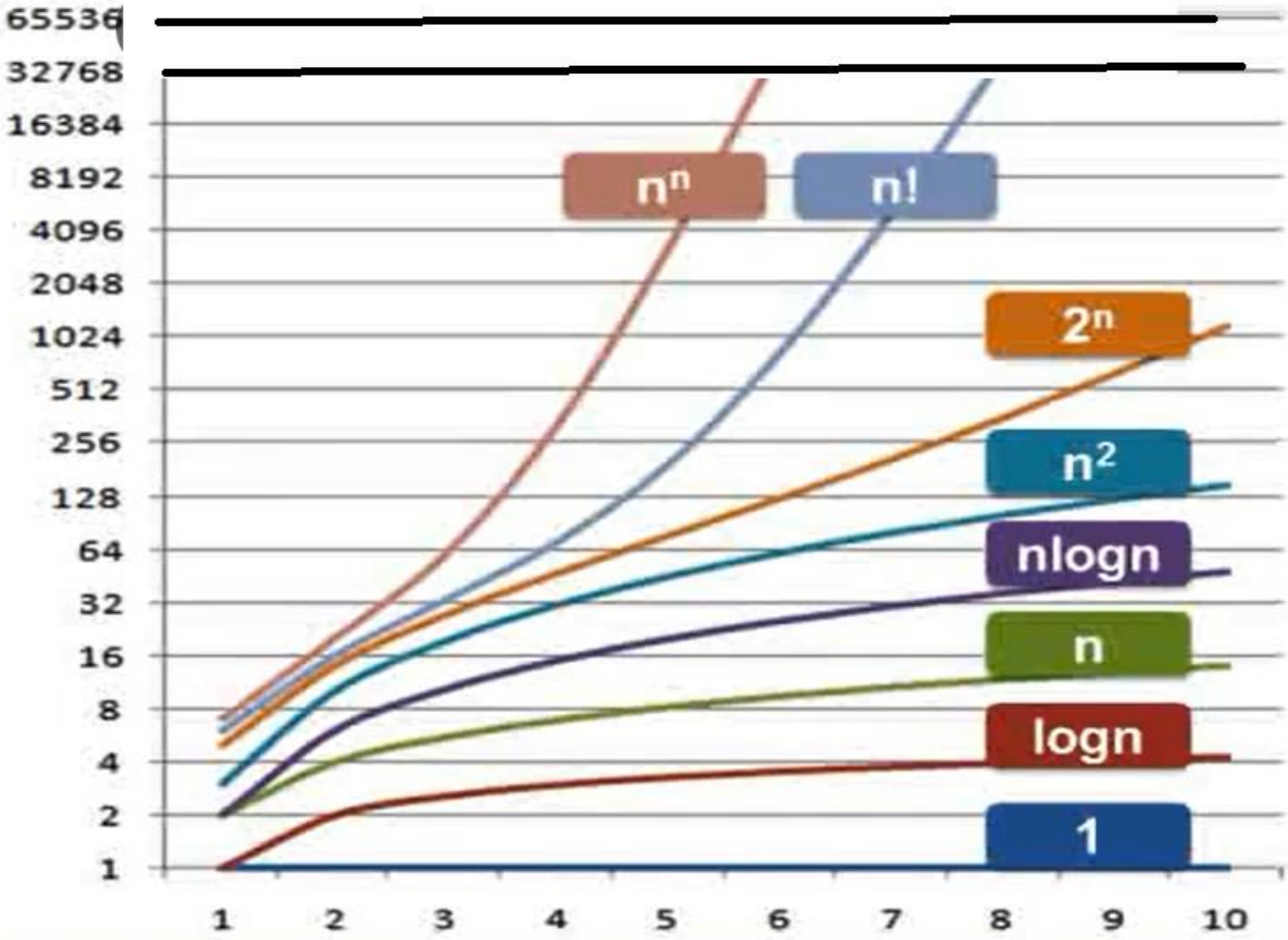


Figure:- 4 Loose Bound

- In figure 3, distance between lines A and B is less as B ≤ A. For large value of x, B will approach to A as it is less than or equal to A.

- In Figure 4, A < B i.e. distance between A and B is large. For example A < B, there will be distance between A and B even for large value of x as it is strictly less than only.

Algorithm with min order of magnitude is best



Big Oh (O)

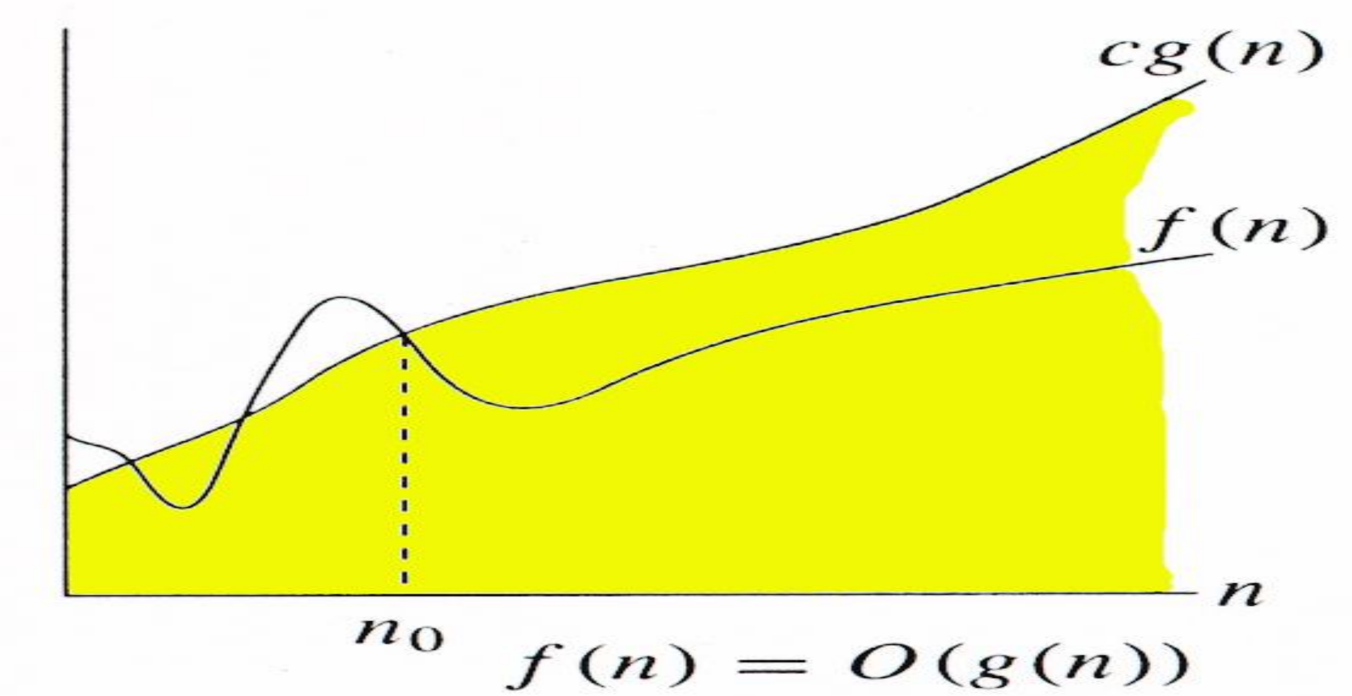
- **Big Oh (O) Notation** This notation provides upper bound for a given function. **O(Big Oh) Notation**: mean `**order at m**ost' i.e bounded above or it will give maximum time required to run the algorithm.

- Let a given function g(n), O(g(n))) is the set of functions f(n) defined as **O(g(n))={f(n):** if there exist positive **constant c** and **n0** such that **0≤f(n) ≤cg(n)** for all n, n>=n0}

- f(n)= O(g(n)) **or** f(n) E O(g(n)), f(n) is bounded above by some positive constant multiple of g(n) for all large values of n.

- $O(g(n)) = \{f(n):$
  ∃ positive constants c and $n_{0,}$
  such that $\forall n \geq n_0,$
  we have $0 \leq f(n) \leq cg(n)$ }

Intuitively: Set of all functions whose *rate of growth* is the same as or lower than that of $g(n)$.



- **$f(n)=3n+2$**

$f(n)=3n+2$ **$g(n) = n$**

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

$f(n) <= c.\ g(n) \ \forall \ c>0 \ \& \ n0 >= 1$

$3n+2 <= c.\ g(n)$

$C=4$

$3(2)+2 <= 4.\ (2)$

$8<=8 \ (T) \ \forall \ c>=4 \ \& \ no=2$

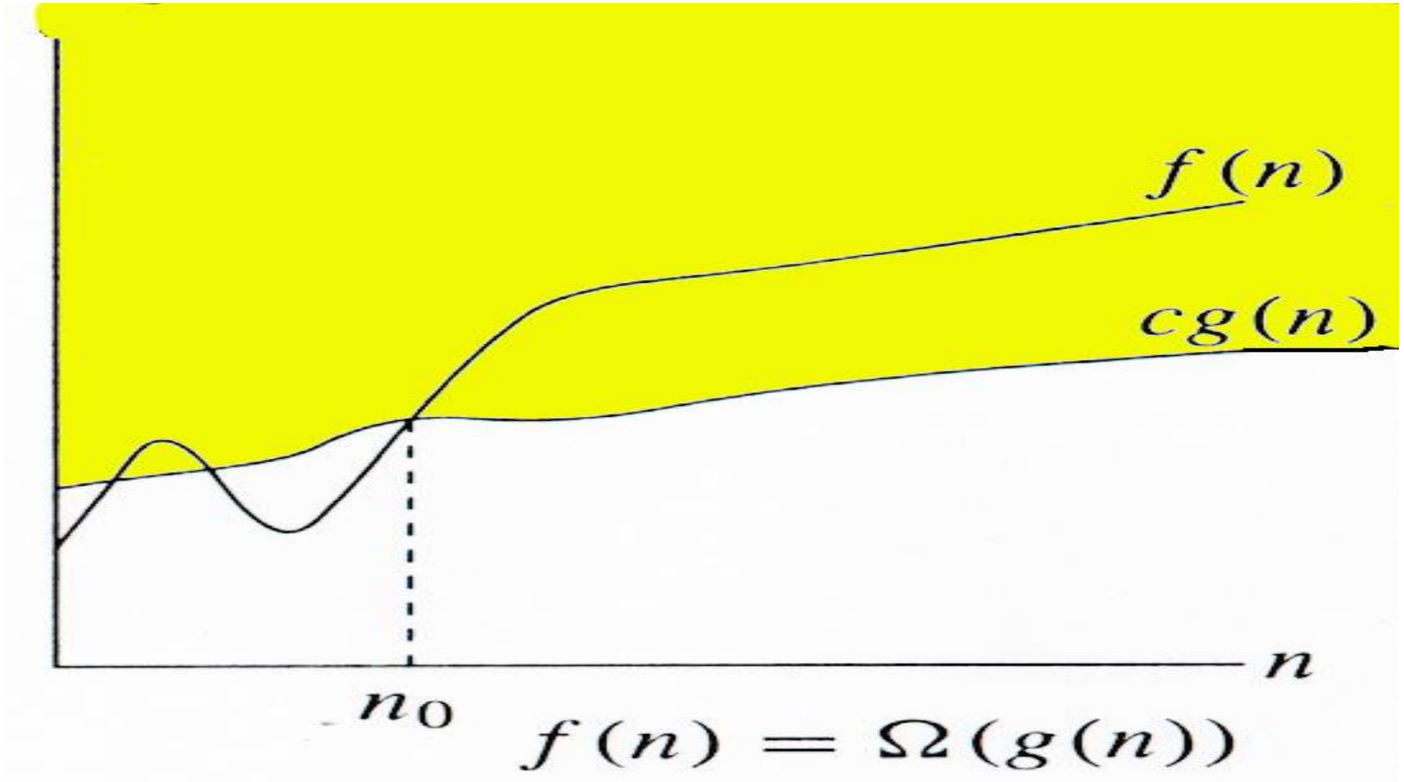
$\boxed{\mathbf{3n+2 <= 4n \ \forall \ no >= 2}}$

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

Big Omega

- This notation provides lower bound for a given function. **(Big Omega):** mean „order at least' i.e minimum time required to execute the algorithm or have lower bound .
- Let a given function g(n). Ώ(g(n))) is the set of functions f(n) defined as
- Ώ(g(n)) ={f(n): if there exist positive constant c and n0 such that 0≤ cg(n) ≤f(n) for all n, n >=n0}

$\Omega(g(n)) = \{f(n)$:
$\exists$ **positive constants** c **and** n_0,
such that $\forall n \geq n_0$,
we have $0 \leq cg(n) \leq f(n)\}$



Intuitively: Set of all functions whose *rate of growth* is the same as or higher than that of *g*(*n*).

Big omega Ω:

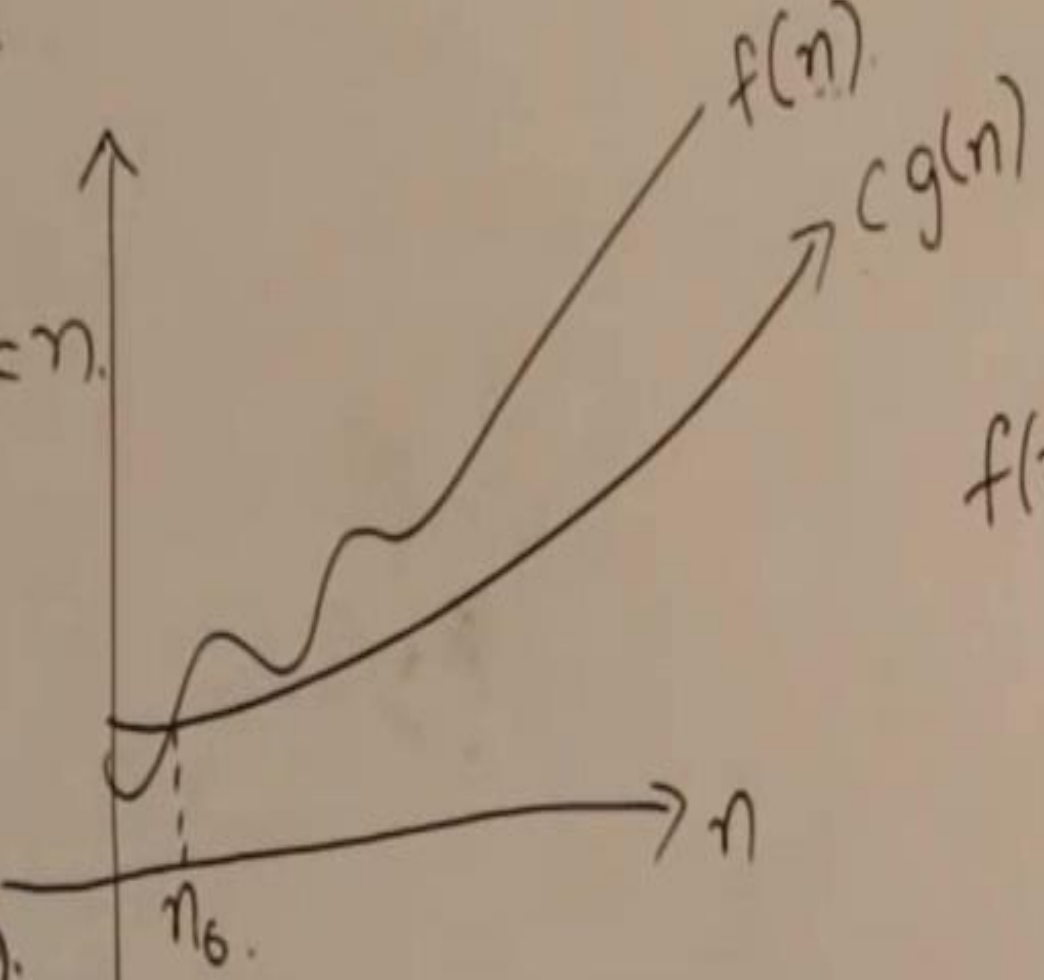
$f(n) = 3n+2 \quad g(n) = n$.

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

$f(n) \geq c\, g(n)$.

$\boxed{3n+2 \geq cn}$ $\quad c = 1$ ✓

$3n+2 = \Omega(n)$. $\quad \underline{n_0 \geq 1}$ ✓



$f(n) \geq c\,g(n)$, $n \geq n_0$

$c > 0$, $n_0 \geq 1$.

Big omega Ω

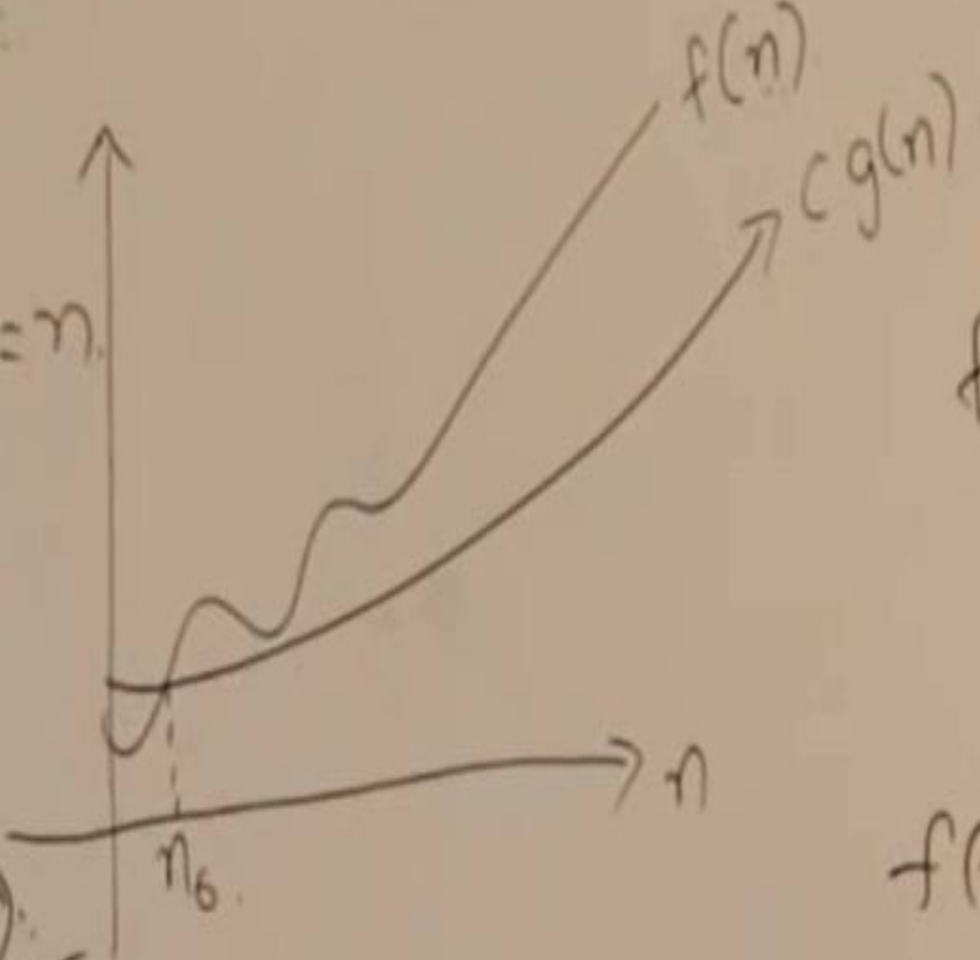
$f(n) = 3n+2 \quad g(n) = n$

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

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

$\boxed{3n+2 \geq cn}$ $c=1$ ✓

$3n+2 = \Omega(n)$ $\underline{n_0 \geq 1}$ ✓



$f(n) = 3n+2 \quad g(n) = n^2$

$3n+2 \stackrel{?}{=} \Omega(n^2)$

$3n+2 \geq cn^2, \; \boxed{n_0}$

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

$\log n$

$(\log\log n)$

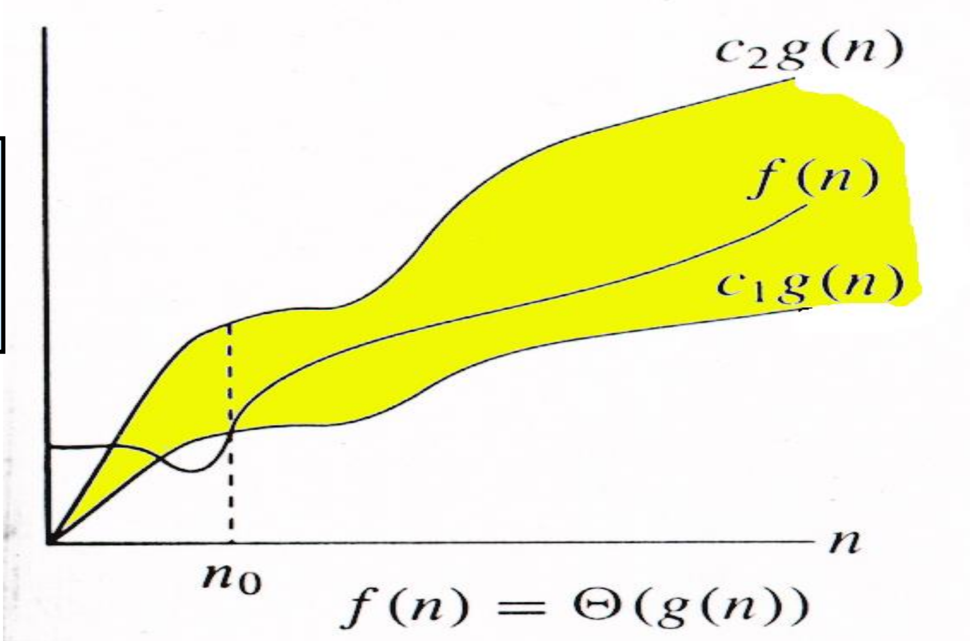
Θ-notation

- It provides both upper and lower bounds for a given function.
- **(Theta) Notation**: means `order exactly'. Order exactly implies a function is bounded above and bounded below both. This notation provides both minimum and maximum value for a function. It further gives that an algorithm will take this much of minimum and maximum time that a function can attain for any input size.
- Let g(n) be given function. f(n) be the set of function defined as

  Θ(g(n)) = {f(n): if there exist positive constant c1,c2 and n0 such that

  0≤c1g(n)≤f(n) ≤c2g(n) for all n, n >= n0}

$\Theta(g(n)) = \{f(n) :$ **$\exists$ positive constants c_1, c_2, and n_0, such that $\forall n \geq n_0$,** $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)\}$



$f(n)$ and $g(n)$ are nonnegative, for large n.

Big theta Θ

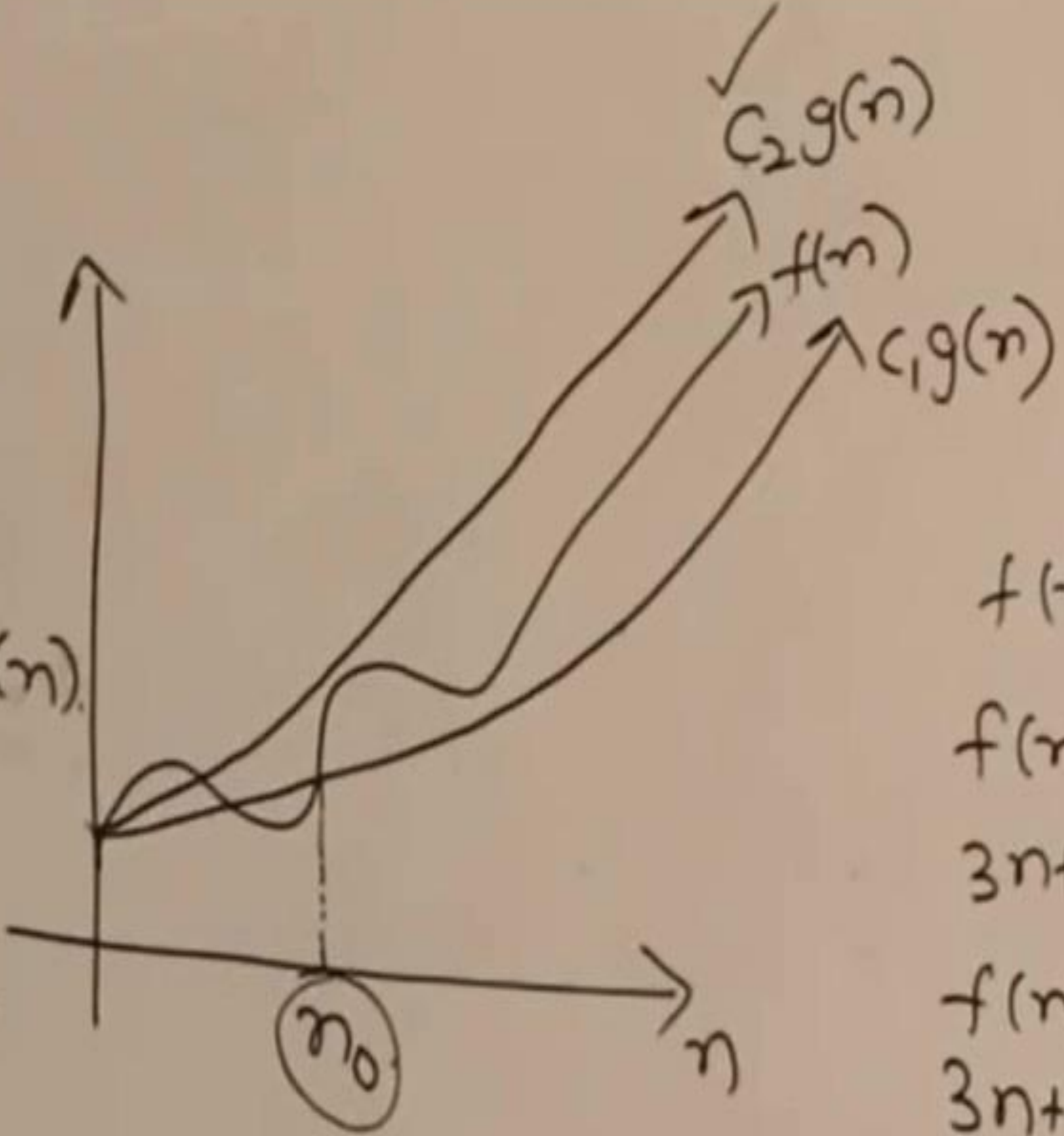
$f(n) = \Theta(g(n))$

$c_1 g(n) \le f(n) \le c_2 g(n)$

$c_1, c_2 > 0$

$n \ge n_0$

$\boxed{n_0 \ge 1}$



$f(n) = 3n+2 \quad g(n) = n$

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

$3n+2 \le 4(n)$, $n_0 \ge 1$

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

$3n+2 \ge n$, $n_0 \ge 1$

- **f(n) = 3n+3 g(n)=n**

f(n)<=c.g(n)

3n+3<= c.g(n)

3n+3<=3n

c=3,4,5.

3n+3<=6n c=6 and n0 =1.

f(n)>=c.g(n)

3n+3>=n c=1 and n0 =1.

n <= 3n+3 <= 6n ᵾ n0 >=1

f(n) = Θ(g(n))