Asymptotic Notation

- Asymptotic Notations used for calculating the running time complexity.
- For example:-

(i) The running time of any operation is computed as ***f*(n)**

This means the operation running time will increase linearly with the increase in **n**.

(ii) If running time is $f(2^n)$. This means the running time operation will increase exponentially when **n** increases.

Similarly, the running time of both operations will be nearly the same if **n** is significantly small.

Types of Data Structure Asymptotic Notation

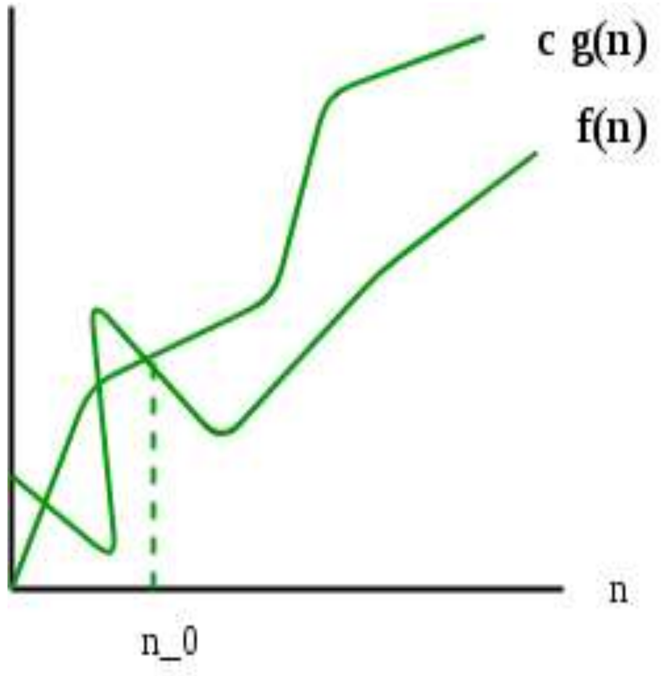
(1) O Notation:- **O(n)** is the formal way to express the upper bound of an algorithm's running time. It measures the worst case time complexity.

(2) Ω Notation:- **Ω(n)** is the formal way to express the lower bound of an algorithm's running time. It measures the best case time complexity.

(3) θ Notation:- **θ(n)** is the formal way to express both the lower bound and the upper bound of an algorithm's running time

O Notation

- Upper bound of an algorithm's running time.
- O(g(n)) = { f(n): there exist positive constants c and n_0 such that 0 <= f(n) <= c*g(n) for all n >= n_0 }



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

Example :- f(n) =2n+3

2n+3 <= 10n or 2n+3 <= 7n

But we will consider (2n+3n) instead of struggling to get value of cg(n)

2n+3 <= 2n+3n

So, 2n+3 <= 5n , n>=1

f(n) c g(n)

wkt f(n) = O(g(n))

= O(n)

Assume $2n+3 <= 2n^2 + 3n^2$

So, $2n+3 <= 5n^2$

f(n) c g(n)

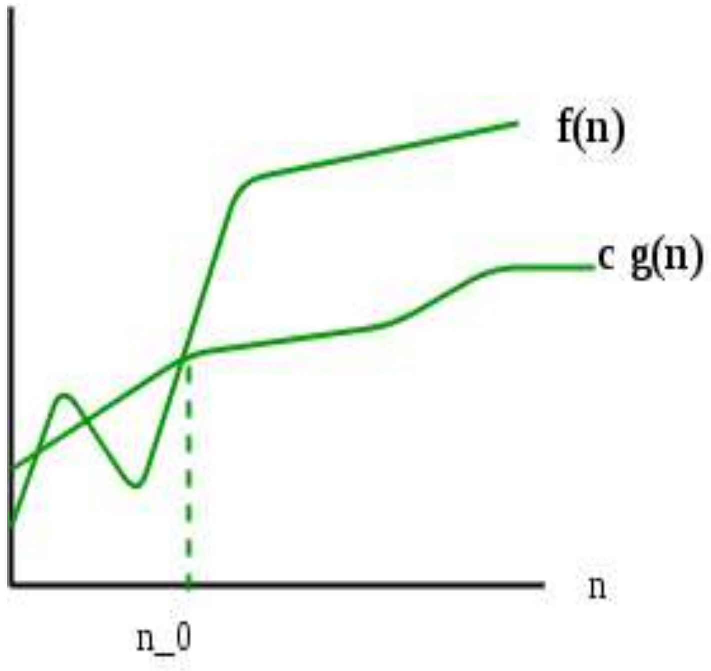
$f(n) = O(n^2)$

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \ldots < 2^n < 3^n < n^n$

Ω Notation

- Lower bound of an algorithm's running time.
- Ω (g(n)) = {f(n): there exist positive constants c and n0

  such that 0 <= c*g(n) <= f(n) for all n >= n0}



f(n) = Omega(g(n))

Example f(n) =2n+3

2n+3 >= 2n

f(n) c g(n)

wkt f(n) > = Ω (g(n))

So, f(n) = Ω (n)

If we consider log n as g(n) then,

2n+3 >= 1*logn

f(n) c g(n)

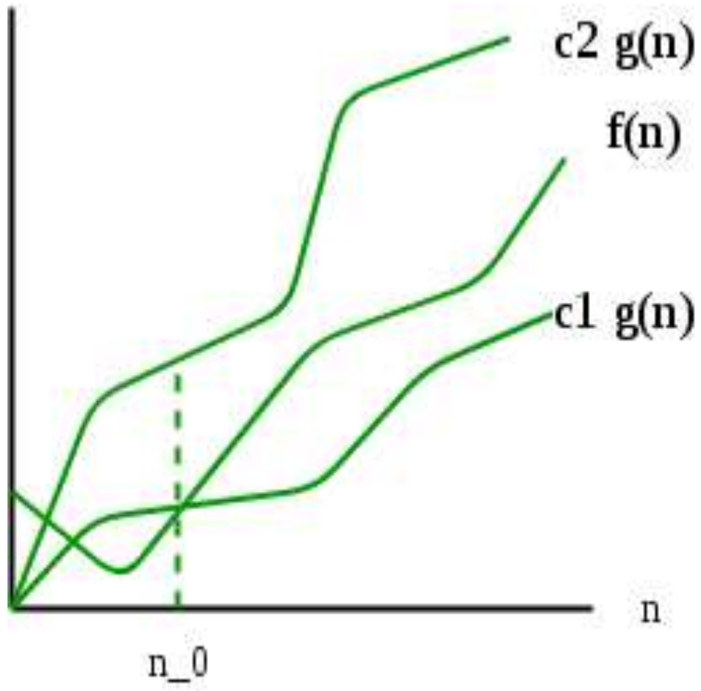
So, f(n) = Ω (log n)

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \ldots < 2^n < 3^n < n^n$

θ Notation

- Lower bound and Upper bound of an algorithm's running time
- **Θ(g(n)) =** {f(n): there exist positive constants c1, c2 and n0

  such that 0 <= c1*g(n) <= f(n) <= c2*g(n) for all n >= n0}



f(n) = theta(g(n))

Example f(n) =2n+3

c1 g(n) <= f(n) <= c2 g(n)

1 n <= 2n+3 <= 5 n

So, f(n) = Θ (g(n)) = *Θ (n)*

Now assume g(n)= n^2

then, $1n^2 <= 2n+3 <= 5n^2$

or

log n <= 2n+3 <= log n

But in this example it is not possible

$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < \ldots < 2^n < 3^n < n^n$