I. Asymptotic Notations:

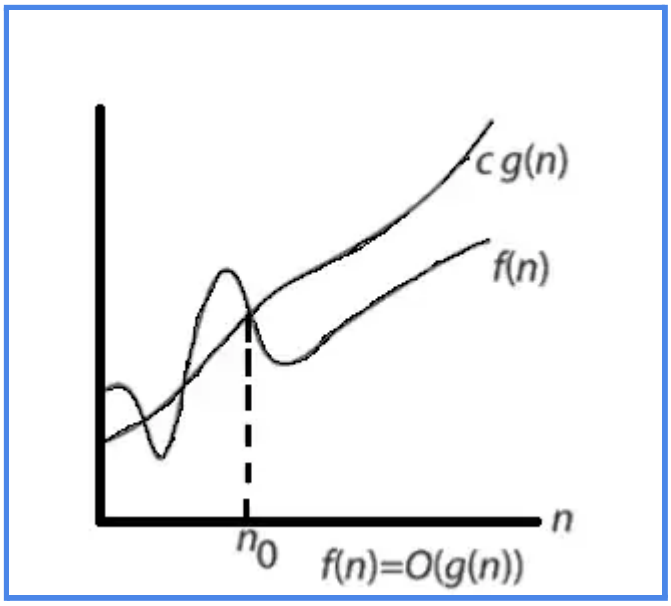
The notations we use to describe the asymptotic (approximate) running time of an algorithm are defined in terms of functions whose domains are the set of natural numbers
N = {0, 1, 2 ... }. The asymptotic notations consist of the following useful notations.

Big Oh (O):

- If we write f(n) = O(g(n)), then there exists a function f(n) such that ∀ n ≥ n0, f(n) ≤ cg (n) with any constant c and a positive integer n0.

OR

- f(n) = O(g(n)) means we can say g(n) is an asymptotic upper bound for f(n).



★ **Example-1:**
Let f(n) = n2 + n + 5. Then f(n) is O(n2), and f(n) is O(n3), but f(n) is not O(n).

★ **Example- 2:**
Let f(n) = 3n . Then f(n) is O(4n) but not f(n) is not O(2n).

O(1) refers to constant time. O(n) indicates linear time; O(nk) (k fixed) refers to polynomial time; O(log n) is called logarithmic time; O(2n) refers to exponential time, etc.

O(1) < O(log n) < O(n) < O(n2) < O(2n)

Omega(Ω):

- If we write f(n) = Ω(g(n)), then there exists a function f(n) such that ∀ n ≥ n0, f(n) ≥ cg(n) with any constant c and a positive integer n0.

OR

- f(n) = Ω(g(n)) means we can say Function g(n) is an asymptotic lower bound for f(n).

★ **Example-1:**

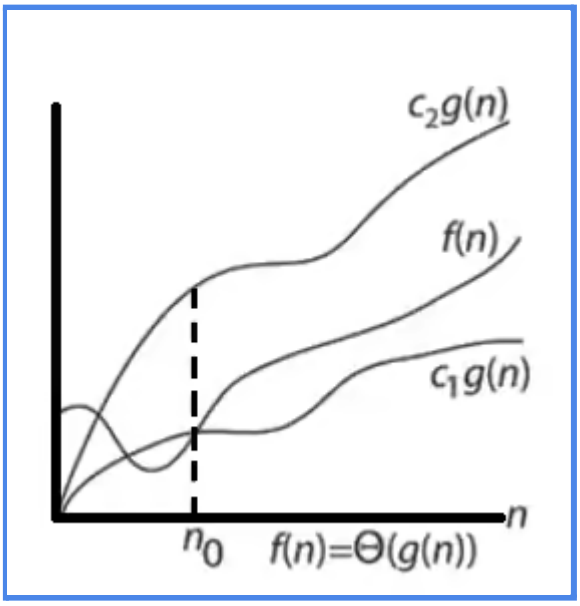
Let f (n) = 2n2 + 4n + 10. Then f (n) is Ω(n2).

Theta(θ):

- If we write f(n) = **θ**(g(n)), then there exists a function f(n) such that ∀ n ≥ n0, c1g(n) ≤ f(n) ≤ c2g(n) with a positive integer n0, any positive constants c1 and c2.

OR

- f(n) = **θ**(g(n)) means we can say Function g(n) is an asymptotically tight bound for f(n).



★ **Example-1:**

Let f (n) = 2n2 + 4n + 10. Then f (n) is **θ**(n2).

Small Oh (o):

- If we write f(n) = o(g(n), then there exists a function such that f(n) < c g(n) with any positive constant c and a positive integer n0.

OR

- f(n) = o(g(n) means we can say Function g(n) is an asymptotically tight upper bound of f(n).

★ **Example-1:**
n1.99 = o(n2)

Small Omega (ω):

- If we write f(n) = ω(g(n)), then these exists a function such that f(n) > cg(n) with any positive constant c and a positive integer n0.

OR

- f(n) = ω(g(n)) means we can say g(n) is asymptotically tight lower bound of f(n).

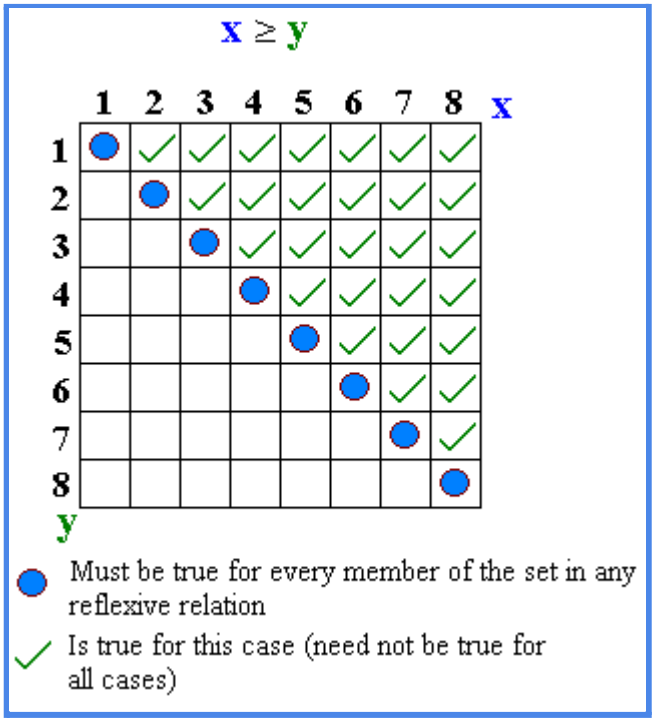
★ **Example-1:**
n2.00001 = ω(n2) and n2 ≠ ω(n2)

II. The relationship between asymptotic notations :

$f(n) \in O(g(n))$ if and only if $g(n) \in \Omega(f(n))$
$f(n) \in o(g(n))$ if and only if $g(n) \in \omega(f(n))$
$f(n) \in \Theta(g(n))$ if and only if $f(n) \in O(g(n))$ and $f(n) \in \Omega(g(n))$
$f(n) \in o(g(n))$ implies $f(n) \in O(g(n))$
$f(n) \in \omega(g(n))$ implies $f(n) \in \Omega(g(n))$
$f(n) \in O(g(n))$ implies $f(n) \notin \omega(g(n))$
$f(n) \in \Omega(g(n))$ implies $f(n) \notin o(g(n))$
$f(n) \sim g(n)$ implies $f(n) \in \Theta(g(n))$
$f(n) \sim g(n)$ is equivalent to $f(n) \in g(n) + o(g(n))$

III. Properties of Asymptotic notations :

1. Reflexive Property:



2. Symmetric Property:

$f(n) \in \Theta(g(n))$ is equivalent to $g(n) \in \Theta(f(n))$

$f(n) \sim g(n)$ is equivalent to $g(n) \sim f(n)$

$f(n) \in o(g(n))$ implies $g(n) \notin o(f(n))$

$f(n) \in \omega(g(n))$ implies $g(n) \notin \omega(f(n))$

3. Transitive Property:

If $f(n) \in O(g(n))$ and $g(n) \in O(h(n))$ then $f(n) \in O(h(n))$

If $f(n) \in \Omega(g(n))$ and $g(n) \in \Omega(h(n))$ then $f(n) \in \Omega(h(n))$

If $f(n) \in \Theta(g(n))$ and $g(n) \in \Theta(h(n))$, then $f(n) \in \Theta(h(n))$

If $f(n) \in o(g(n))$ and $g(n) \in o(h(n))$ then $f(n) \in o(h(n))$

If $f(n) \in \omega(g(n))$ and $g(n) \in \omega(h(n))$ then $f(n) \in \omega(h(n))$

If $f(n) \sim g(n)$ and $g(n) \sim h(n)$ then $f(n) \sim h(n)$

4. Mixed asymptotic transitive Properties:

If $f(n) \in \Theta(g(n))$ and $g(n) \in O(h(n))$, then $f(n) \in O(h(n))$
If $f(n) \in \Theta(g(n))$ and $g(n) \in \Omega(h(n))$, then $f(n) \in \Omega(h(n))$
If $f(n) \in \Theta(g(n))$ and $g(n) \in o(h(n))$, then $f(n) \in o(h(n))$
If $f(n) \in \Theta(g(n))$ and $g(n) \in \omega(h(n))$, then $f(n) \in \omega(h(n))$
If $f(n) \in O(g(n))$ and $g(n) \in o(h(n))$, then $f(n) \in o(h(n))$
If $f(n) \in \Omega(g(n))$ and $g(n) \in \omega(h(n))$, then $f(n) \in \omega(h(n))$

IV. Analysis of Algorithms :

The complexity of an algorithm is a function describing the efficiency of the algorithm in terms of the amount of data the algorithm must process. Usually there are natural units for the domain and range of this function.

- Algorithm can be classified by the amount of time they need to complete compared to their input size.
- The analysis of an algorithm focuses on the complexity of algorithm which depends on time or space.

There are two main complexity measures of the efficiency of an algorithm:

1. Time Complexity:

The time complexity is a function that gives the amount of time required by an algorithm to run to completion.

- ➔ **Worst case time complexity:** It is the function defined by the maximum amount of time needed by an algorithm for an input of size n.
- ➔ **Average case time complexity:** The average-case running time of an algorithm is an estimate of the running time for an "average" input. Computation of average-case running time entails knowing all possible input sequences, the probability distribution of occurrence of these sequences, and the running times for the individual sequences.
- ➔ **Amortized Running Time:** It is the time required to perform a sequence of (related) operations is averaged over all the operations performed. Amortized analysis guarantees the average performance of each operation in the worst case.
- ➔ **Best case time complexity:** It is the minimum amount of time that an algorithm requires for an input of size n.

2. Space Complexity:

The space complexity is a function that gives the amount of space required by an algorithm to run to completion.