Asymptotic Notation

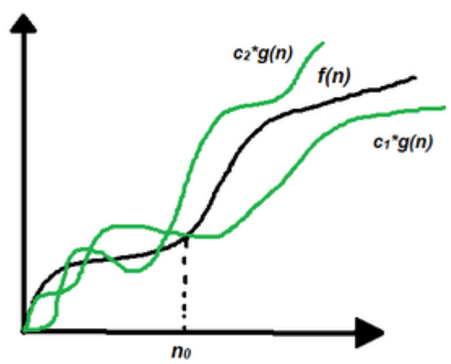
The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don't depend on machine-specific constants and don't require algorithms to be implemented and time taken by programs to be compared. Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

> ***There are mainly three asymptotic notations:***
>
> 1. ***Big-O Notation (O-notation)***
> 2. ***Omega Notation (Ω-notation)***
> 3. ***Theta Notation (Θ-notation)***

1. Theta Notation (Θ-Notation):

> *Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the **average-case** complexity of an algorithm.*

Let g and f be the function from the set of natural numbers to itself. The function f is said to be Θ(g), if there are constants c1, c2 > 0 and a natural number n0 such that c1* g(n) ≤ f(n) ≤ c2 * g(n) for all n ≥ n0



Mathematical Representation of Theta notation:

> *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}*

> ***Note:*** *Θ(g) is a set*

The above expression can be described as if f(n) is theta of g(n), then the value f(n) is always between c1 * g(n) and c2 * g(n) for large values of n ($n \geq n0$). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0.

The execution time serves as both a lower and upper bound on the algorithm's time complexity.

It exist as both, most, and least boundaries for a given input value.

A simple way to get the Theta notation of an expression is to drop low-order terms and ignore leading constants. For example**,** Consider the expression $\mathbf{3n^3 + 6n^2 + 6000 = \Theta(n^3)}$, the dropping lower order terms is always fine because there will always be a number(n) after which $\Theta(n^3)$ has higher values than $\Theta(n^2)$ irrespective of the constants involved. For a given function g(n), we denote Θ(g(n)) is following set of functions. **Examples :**

> *{ 100 , log (2000) , 10^4 } belongs to* ***Θ(1)***
> *{ (n/4) , (2n+3) , (n/100 + log(n)) } belongs to* ***Θ(n)***
> *{ (n^2+n) , (2n^2) , (n^2+log(n))} belongs to* $\boldsymbol{\Theta(n^2)}$
>
> ***Note: Θ provides exact bounds.***

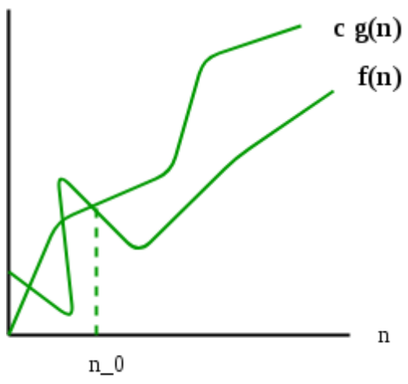
2. Big-O Notation (O-notation):

> *Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.*

If f(n) describes the running time of an algorithm, f(n) is O(g(n)) if there exist a positive constant C and n0 such that, $0 \leq f(n) \leq cg(n)$ for all $n \geq n0$

It returns the highest possible output value (big-O)for a given input.

The execution time serves as an upper bound on the algorithm's time complexity.



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

For example**,** Consider the case of Insertion sort. It takes linear time in the best case and quadratic time in the worst case. We can safely say that the time complexity of the Insertion sort is $O(n^2)$.
Note: $O(n^2)$ also covers linear time.

If we use Θ notation to represent the time complexity of Insertion sort, we have to use two statements for best and worst cases:

- The worst-case time complexity of Insertion Sort is $\Theta(n^2)$.
- The best case time complexity of Insertion Sort is $\Theta(n)$.

The Big-O notation is useful when we only have an upper bound on the time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

Examples :

> *{ 100 , log (2000) , 10^4 } belongs to* ***O(1)***
> ***U*** *{ (n/4) , (2n+3) , (n/100 + log(n)) } belongs to* ***O(n)***
> ***U*** *{ (n^2+n) , (2n^2) , (n^2+log(n))} belongs to* ***O(n^2)***
>
> ***Note:*** *Here,* ***U represents union****, we can write it in these manner because* ***O provides exact or upper bounds .***

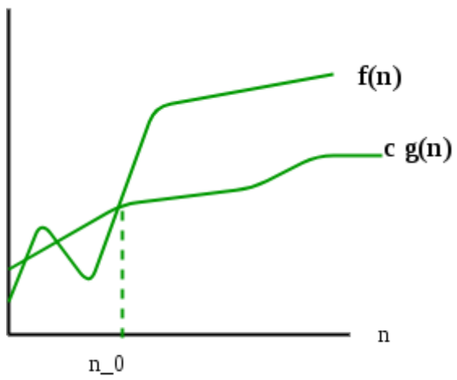
3. Omega Notation (Ω-Notation):

> *Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.*

The execution time serves as a both lower bound on the algorithm's time complexity.

It is defined as the condition that allows an algorithm to complete statement execution in the shortest amount of time.

Let g and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that c*g(n) ≤ f(n) for all n ≥ n0



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

Mathematical Representation of Omega notation :

> *Ω(g(n)) = { f(n): there exist positive constants c and n0 such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0 }*

Let us consider the same Insertion sort example here. The time complexity of Insertion Sort can be written as Ω(n), but it is not very useful information about insertion sort, as we are generally interested in worst-case and sometimes in the average case.

Examples :

> *{ (n^2+n) , (2n^2) , (n^2+log(n))} belongs to* ***Ω(n^2)***
> ***U*** *{ (n/4) , (2n+3) , (n/100 + log(n)) } belongs to* ***Ω(n)***
> ***U*** *{ 100 , log (2000) , 10^4 } belongs to* ***Ω(1)***
>
> ***Note:*** *Here,* ***U represents union,*** *we can write it in these manner because* ***Ω provides exact or lower bounds.***