Chapter 9 Growth Functions

The function that derives the running time of an algorithm and its memory space requirements for a given set of input is referred as algorithm complexity.

Asymptotic notation is the most simple and easiest way of describing the running time of an algorithm.

Asymptotic notation describes the time complexity in terms of three common measures:

- Best case
- Worst case
- Average case

The three most common asymptotic notations are:

- Big Oh Notation
- Omega Notation
- Theta Notation

Algorithm Performance:

The performance of an algorithm is obtained by totaling the number of occurrences of each operation when running the algorithm.

The performance of algorithm is evaluated as a function of the input size ‘n” and is said to be considered as modulo a multiplicative constant.

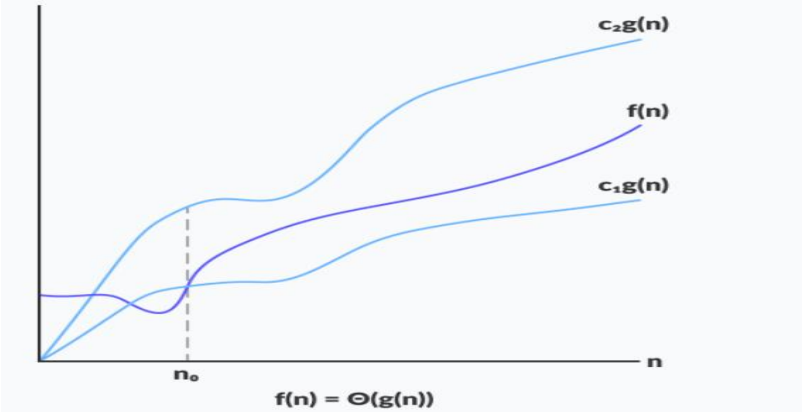
The following notations are commonly used in performance analysis and used to characterize the complexity of an algorithm:

Theta Notation (Θ-notation) -Tightly Bound

Big-O Notation (O-notation) - Upper Bound

Omega Notation (Ω-notation) - Lower Bound

Theta Notation (Θ –Notation):



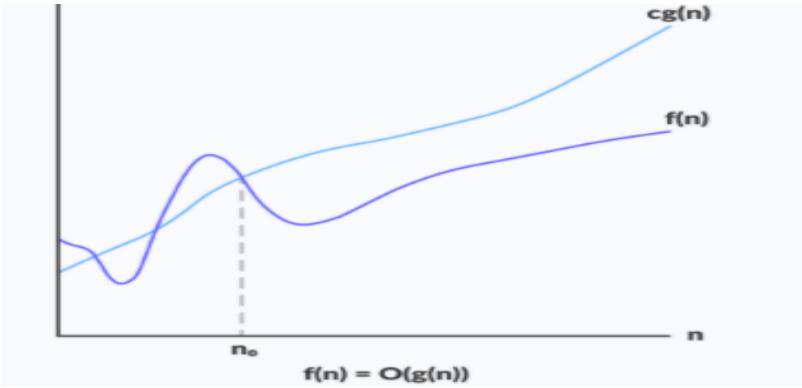
Theta notation bounds a function within constant factors.

f(n)= Θ(g(n) if there exists positive constants n_0, C_1 and C_2 such that to the right of n_0, the value of f(n) always lies between C_1g(n) and C_2g(n).

Big-O Notation (O-notation)

This notation gives an upper bound for function to within constant factor.

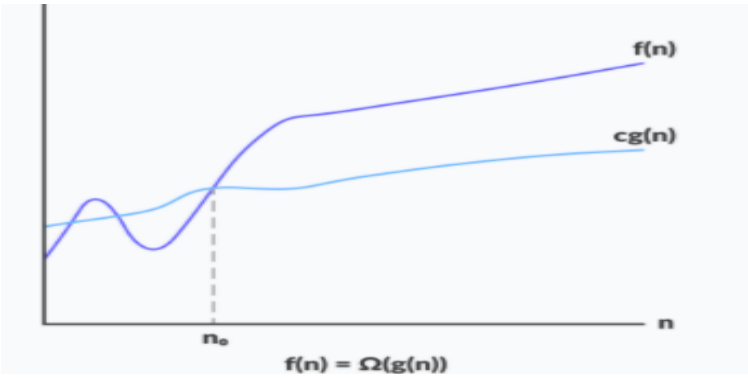
f(n) = O(g(n) if there exists positive constants n_0 and C such that to the right of n_0, the value of f(n) always lies on or below Cg(n).



Omega Notation (Ω-notation)

This notation gives the lower bound for a function to within constant factor.

F(n)= Ω g(n) if there exists positive constants n_0 and C such that to the right of n_0 the value of f(n) always lie on or above Cg(n).



Big O Notation Examples

Big O notations are used to clarify running time function. If f(n) is O(g(n), then informally, f(n) is within a constant factor of g(n).

Definition: Let f(n) and g(n) be two functions on positive integers. We say f(n) is O(g(n)) if there exist two positive constants C and K such that f(n) <= C g(n) for all n >= K.

Example 1

f(n) = 10n+5 and g(n)= n

Solution:

Here, f(n) = O(g(n)) = O(n)

To show f(n) is O(g(n)), we must show constants C and k such that $f(n) \leq Cg(n)$ for all $n \geq k$.

Or, $10n+5 \leq Cn$ for all $n \geq k$.

We are allowed to choose C and k to be integer, we want as long as they are positive.

They can be as big as we want but they cannot be the function of n.

Let C=15, then we need to show, $10n+5 \leq 15n$

Solving for n,

$10n+5 \leq 15n$

Or, $5 \leq 5n$

Or, $1 \leq n$

So, f(n)= 10+5 <= 15.g(n) for all $n \geq 1$.

Hence, we have shown f(n) is O(g(n)).

Example 2

$f(n) = 3n^2 + 4n + 1$. Show f(n) is $O(n^2)$.

Solution:

We know, $4n \leq 4n^2$ for all $n \geq 1$.

and $1 \leq n^2$ for all $n \geq 1$.

$3n^2 + 4n + 1 \leq 3n^2 + 4n^2 + n^2$ for all $n \geq 1$.

$\leq 8n^2$ for all $n \geq 1$.

Hence, $f(n) \leq 8n^2$ for all $n \geq 1$.

Therefore, f(n) is $O(n^2)$. [C = 8, K = 1]