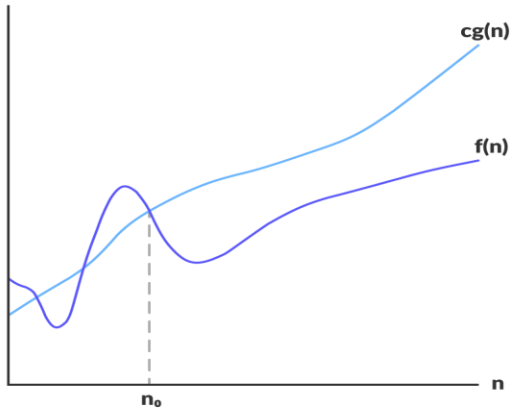
Chapter 9 Growth Functions

- Asymptotic notations are the mathematical notations used to describe the running time of an algorithm when the input tends towards a particular value or a limiting value.
- When it comes to analyzing the complexity of any algorithm in terms of time and space, we can never provide an exact number to define the time required and the space required by the algorithm, instead we express it using some standard notations, also known as Asymptotic Notations.
- Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.
- For example: In bubble sort, when the input array is already sorted, the time taken by the algorithm is linear i.e. the best case. But, when the input array is in reverse condition, the algorithm takes the maximum time (quadratic) to sort the elements i.e. the worst case. When the input array is neither sorted nor in reverse order, than it takes average time.
- Types of Asymptotic Notation
    1. **Big-O Notation (O)** – Big O notation specifically describes worst case scenario.
    2. **Omega Notation (Ω)** – Omega (Ω) notation specifically describes best case scenario.
    3. **Theta Notation (θ)** – This notation represents the average complexity of an algorithm.

Big-O Notation (O):

Big O notation specifically describes scenario. It represents the **upper bound** running time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

$$O(g(n)) = \{ f(n) : \text{there exist positive constants } c \text{ and } n_0 \text{ such that } \mathbf{0} \leq \boldsymbol{f(n)} \leq \boldsymbol{c * g(n)} \text{ for all } n >= n_0 \}$$



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

Example:

$f(n) = 3n + 2$

$g(n) = n$

$can\ we\ write\ f(n) = O(g(n))?$
$Condition\ to\ be\ satisfied\ is\ 0 <= f(n) <= c * g(n)$

$0 <= 3n + 2 <= c * n$

$0 <= 3n + 2 <= 4 * n\ for\ all\ n0 >= 2$

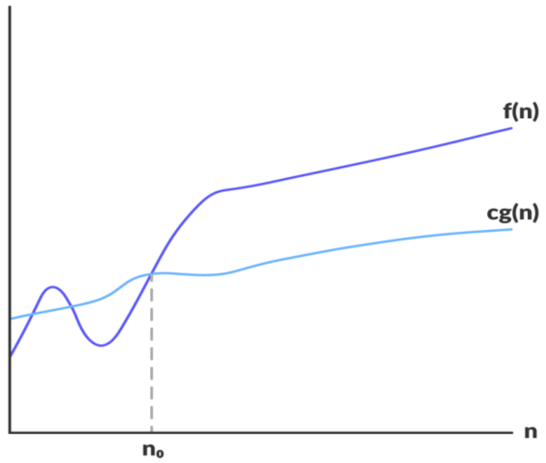
$So, f(n) = O(g(n))$

$3n + 2 = O(n)$

Big-Omega Notation (Ω):

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

$$\Omega(g(n)) = \{ f(n): there\ exist\ positive\ constants\ c\ and\ n_0$$
$$such\ that\ \mathbf{0} \leq \boldsymbol{cg(n)} \leq \boldsymbol{f(n)}$$
$$for\ all\ n \geq n_0 \}$$



f(n) = Ω(g(n))

Example:

$f(n) = 3n + 2$

$g(n) = n$

$can\ we\ write\ f(n) = \Omega\ (g(n))?$

$Solution:$

$Condition\ to\ be\ satisfied\ is\ 0\ <=\ c * g(n)\ <=\ f(n)$

$0\ <=\ c * n\ <=\ 3n + 2$

$0\ <=\ n\ <=\ 3n + 2\ for\ all\ n0 >= 1$

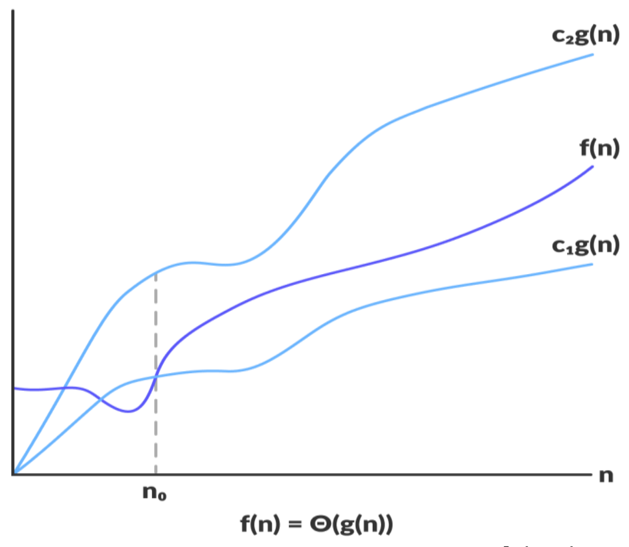
$So, f(n) = \Omega\ (g(n))$

$3n + 2 = \Omega\ (n)$

Theta Notation (θ):

This notation describes both upper bound and lower bound of an algorithm so we can say that it defines **exact asymptotic behavior**. In the real case scenario the algorithm not always run on best and worst cases, the average running time lies between best and worst and can be represented by the theta notation. Thus, it provides the **average case** complexity of an algorithm.

$$\Theta(g(n)) = \{ f(n): there\ exist\ positive\ constants\ c1, c2\ and\ n_0$$
$$such\ that\ \mathbf{0} \leq \mathbf{c1g(n)} \leq \boldsymbol{f(n)} \leq \mathbf{c2g(n)}$$
$$for\ all\ n \geq n_0 \}$$



Chapter 9 Growth Functions

Example:

$f(n) = 3n + 2$

$g(n) = n$

$can\ we\ write\ f(n) = \theta\ (g(n))?$

$Solution:$

$Condition\ to\ be\ satisfied\ is\ c1 * g(n)\ <=\ f(n)\ <=\ c2 * g(n)$

$c1n\ <=\ 3n + 2\ <=\ c2n$

$n\ <=\ 3n + 2\ <=\ 5n\ for\ all\ n0 >= 1$

$So, f(n) =\ \theta\ (g(n))$

$3n + 2 =\ \theta\ (n)$

Little o Notation:

Little o notation is used to describe an upper bound that cannot be tight. In other words, **loose upper bound** of f(n). It is formally defined as:

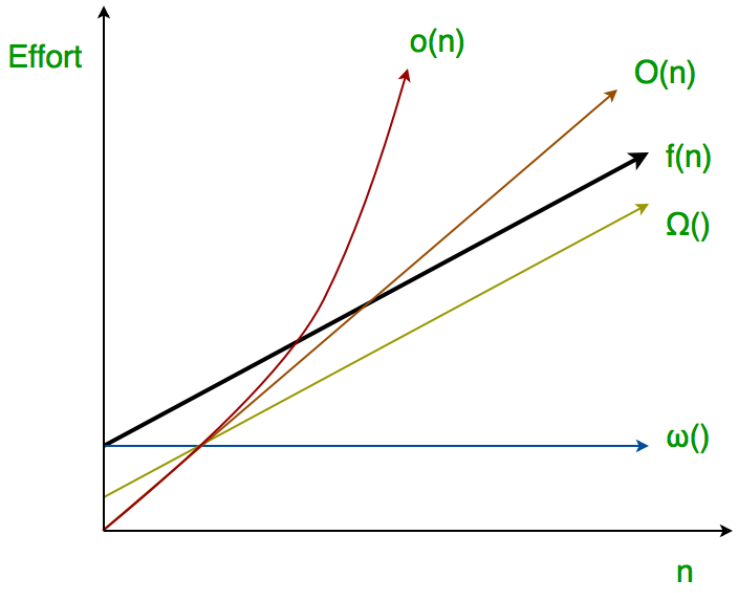
$$O(g(n))\ = \{\, f(n) :\ for\ any\ real\ constants\ c > 0, there\ exists$$
$$an\ integer\ constant\ n_0 \geq 1, such\ that\ \mathbf{0}\ \leq\ \boldsymbol{f}(\boldsymbol{n}) <\ \boldsymbol{c} * \boldsymbol{g}(\boldsymbol{n})$$
$$for\ all\ n\ >=\ n_0\ \}$$

Using mathematical relation, we can say that f(n) = o(g(n)) means, $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$

Little ω notation:

$$\omega(g(n))\ = \{\, f(n): for\ any\ \ real\ constants\ c > 0, there\ exists$$
$$an\ integer\ constant\ \ n_0\ \geq 1, such\ that\ \mathbf{0}\ \leq\ \boldsymbol{c} * \boldsymbol{g}(\boldsymbol{n}) <\ \boldsymbol{f}(\boldsymbol{n})$$
$$for\ all\ n\ \geq\ n_0 \}$$

Note: Omega (ω) is a rough estimate of the order of the growth whereas Big Omega (Ω) may represent exact order of growth.


Effort
o(n)
O(n)
f(n)
Ω()
ω()
n