Lecture

NOTE: FOR FURTHER DETAILS AND MORE COMPREHENSIVE STUDY, PLEASE SEE RECOMMENDED BOOKS OR INTERNET.

Asymptotic Analysis

Asymptotic analysis of an algorithm refers to defining the mathematical foundation/framing of its run-time performance. Using asymptotic analysis, we can very well conclude the best case, average case, and worst case scenario of an algorithm.

Asymptotic analysis is input bound i.e., if there's no input to the algorithm, it is concluded to work in a constant time. Other than the "input" all other factors are considered constant.

Asymptotic analysis refers to computing the running time of any operation in mathematical units of computation. For example, the running time of one operation is computed as ***f(n)*** and may be for another operation it is computed as $g(n^2)$. This means the first operation running time will increase linearly with the increase in ***n*** and the running time of the second operation will increase exponentially when ***n*** increases. Similarly, the running time of both operations will be nearly the same if ***n*** is significantly small.

Usually, the time required by an algorithm falls under three types:

Best Case − Minimum time required for program execution.
Average Case − Average time required for program execution.
Worst Case − Maximum time required for program execution.

Asymptotic Notations

Execution time of an algorithm depends on the instruction set, processor speed, disk I/O speed, etc. Hence, we estimate the efficiency of an algorithm asymptotically.

Time function of an algorithm is represented by ***T(n)***, where ***n*** is the input size.

Different types of asymptotic notations are used to represent the complexity of an algorithm. Following asymptotic notations are used to calculate the running time complexity of an algorithm.

O − Big Oh Notation
Ω − Big omega Notation
θ − Big theta Notation
Other Notations

Big Oh, *O*: Asymptotic Upper Bound

The notation (***n***) is the formal way to express the upper bound of an algorithm's running time. is the most commonly used notation. It measures the worst case time complexity or the longest amount of time an algorithm can possibly take to complete.

A function ***f(n)*** can be represented is the order of ***g(n)*** that is ***O(g(n)),*** if there exists a value of positive integer ***n*** as n_0 and a positive constant ***c*** such that:

$f(n) \leqslant c.g(n)$
$f(n) \leqslant c.g(n)$ *for* $n > n_0$

Hence, function ***g(n)*** is an upper bound for function ***f(n),*** as ***g(n)*** grows faster than ***f(n)***.

Example

Let us consider a given function, $f(n)=4n^3 + 10n^2 + 5n + 1$

$f(n) \leqslant c.g(n)$

$4n^3 + 10n^2 + 5n + 1 \leqslant 19.n^3$ for all the values of $n > 1$

Hence, the complexity of ***f(n)*** can be represented as ***O(g(n))***, i.e. $O(n^3)$

Big Omega, Ω: Asymptotic Lower Bound

The 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 or the best amount of time an algorithm can possibly take to complete.

We say that $f(n) = \Omega(g(n))$ when exists a value of positive integer ***n*** as n_0 and a positive constant ***c*** such that:

$f(n) \geqslant c.g(n)$

$f(n) \geqslant c.g(n)$ *for* $n > n_0$

Hence, function ***g(n)*** is a lower bound for function ***f(n).*** Here ***n*** is a positive integer. It means function ***g*** is a lower bound for function ***f*** ; after a certain value of ***n***, ***f*** will never go below ***g***.

Example

Let us consider a given function, $f(n)=4n^3 + 10n^2 + 5n + 1$

$f(n) \geqslant c.g(n)$

$4n^3 + 10n^2 + 5n + 1 \geqslant 1.n^3$ for all the values of $n > 1$

Hence, the complexity of ***f(n)*** can be represented as ***Ω(g(n))***, i.e. $\Omega(n^3)$

Theta, θ: Asymptotic Average/Tight Bound

The notation ***(n)*** is the formal way to express both the lower bound and the upper bound of an algorithm's running time. Some may refer big theta notation as the average case time complexity; while big theta notation could be almost accurately used to describe the average case, other notations could be used as well.

We say that f(n)=θ(g(n)) when there exists a value of positive integer ***n*** as n_0 and constants c_1 and c_2 such that:

$c_1.g(n) \leqslant f(n) \leqslant c_2.g(n)$ for all sufficiently large value of ***n***.

This means function ***g*** is a tight bound for function ***f***.

Example

Let us consider a given function, $f(n)=4n^3 + 10n^2 + 5n + 1$

$c_1.g(n) \leqslant f(n) \leqslant c_2.g(n)$

$1.n^3 \leqslant 4n^3 + 10n^2 + 5n + 1 \leqslant 19.n^3$ for all the values of $n > 1$

Hence, the complexity of ***f(n)*** can be represented as ***θ(g(n))***, i.e. ***θ(n3)***.

Other Notations:

Little Oh, *o*

The asymptotic upper bound provided by ***O***-notation may or may not be asymptotically tight. The bound $2n^2 = O(n^2)$ is asymptotically tight, but the bound $2n = O(n^2)$ is not.

We use ***o***-notation to denote an upper bound that is not asymptotically tight.

We formally define ***o(g(n))*** (***little-oh*** *of* ***g*** *of* ***n***) as the set ***f(n) = o(g(n))*** for any positive constant $c > 0$ and there exists a value $n_0 > 0$, such that:

$0 \leqslant f(n) \leqslant c.g(n)$.

Intuitively, in the ***o***-notation, the function ***f(n)*** becomes insignificant relative to ***g(n)*** as ***n*** approaches infinity; that is,

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = 0$$

Little Omega, ω

We use ω-notation to denote a lower bound that is not asymptotically tight. Formally, however, we define ω(g(n)) (***little-omega*** of ***g*** of ***n***) as the set $f(n) = \omega(g(n))$ for any positive constant $c > 0$ and there exists a value $n_0 > 0$, such that:

$f(n) \leqslant c.g(n) \leqslant \infty$.

The relation $f(n) = \omega(g(n))$ implies that the following limit exists:

$$\lim_{n\to\infty}\left(\frac{f(n)}{g(n)}\right) = \infty$$

That is, ***f(n)*** becomes arbitrarily large relative to ***g(n)*** as ***n*** approaches infinity.

Following is a list of some common asymptotic notations −

Constant	–	$O(1)$
Logarithmic	–	$O(\log n)$
Linear	–	$O(n)$
n log n	–	$O(n \log n)$
Quadratic	–	$O(n^2)$
Cubic	–	$O(n^3)$
Polynomial	–	$n^{O(1)}$
Exponential	–	$2^{O(n)}$