Asymptotic Notations

Asymptotic analysis of an algorithm refers to defining the mathematical boundation/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

Following are the commonly used asymptotic notations to calculate the running time complexity of an algorithm.

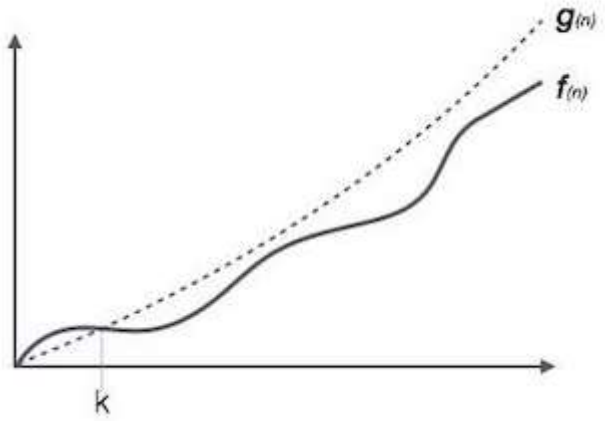
- O Notation
- Ω Notation
- θ Notation

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	–	$2^{O(n)}$

Big Oh Notation, O

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

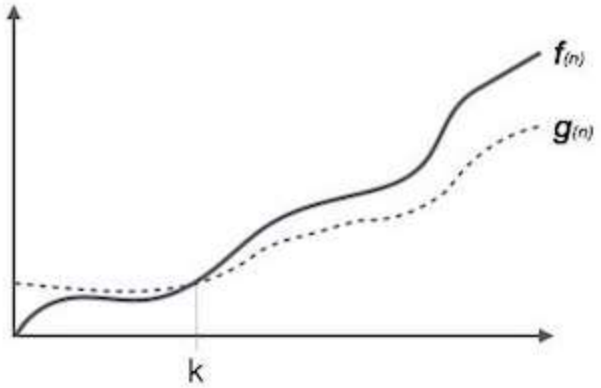


For example, for a function ***f*(n)**

$O(f(n)) = \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } f(n) \leq c.g(n) \text{ for all } n > n_0. \}$

Omega Notation, Ω

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.

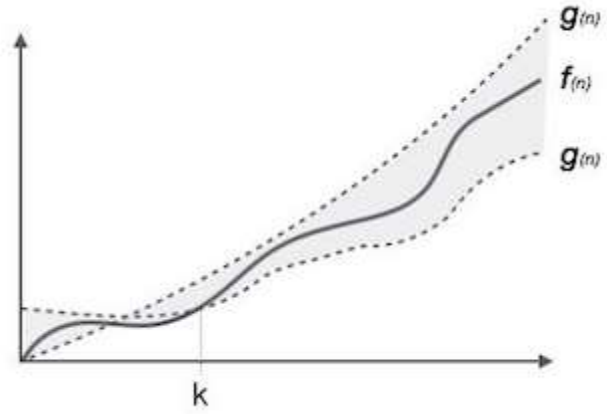


For example, for a function ***f*(n)**

$\Omega(f(n)) \geq \{ g(n) : \text{there exists } c > 0 \text{ and } n_0 \text{ such that } g(n) \leq c.f(n) \text{ for all } n > n_0. \}$

Theta Notation, θ

The notation θ(n) is the formal way to express both the lower bound and the upper bound of an algorithm's running time. It is represented as follows −



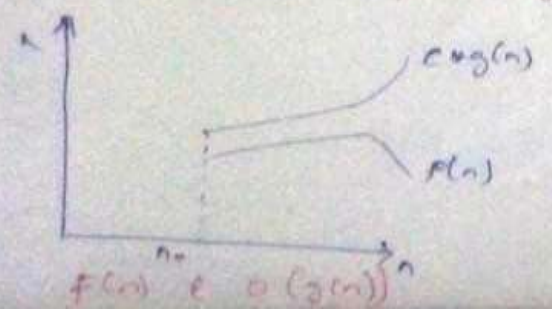
$\theta(f(n)) = \{ g(n)$ if and only if $g(n) = O(f(n))$ and $g(n) = \Omega(f(n))$ for all $n > n_0. \}$

Big oh notation

- 'O' - notation
- It is used to represent the Upper bound of algo's running time
- Longest amount of time taken by the algo to complete

Def

- Let f(n) & g(n) be 2 non negative fn's
- Let n_0 & constant c are 2 integers, and that n_0 denotes some value of input & $n > n_0$
- Similarly 'c' is some const, such that $c > 0$, $f(n) \le c * g(n)$
  Then f(n) is higher of g(n)
- It is also denoted as $f(n) \in O(g(n))$

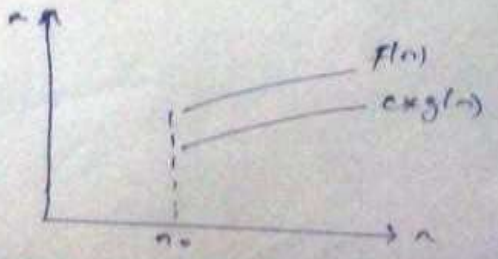


Omega Notation

- Notation → Ω
- It is used to represent the Lower bound of algo's running time
- It denotes, Shortest amount of time taken by algo to complete

Def

- n for f(n) is said to be in Ω(g(n)) if f(n) is bounded below by some +ve const multiple of g(n), that is $f(n) \ge c * g(n)$, for all $n \ge n_0$
- It is denoted as $f(n) \in \Omega(g(n))$



Theta notation

- Θ
- running time is b/w upper & lower bound
- n=2

Def

- Let f(n) & g(n) be 2 non-negative fn's
- There are 2 positive const's, c_1 & c_2, such that $c_1 g(n) \le f(n) \le c_2 g(n)$