Unit 8:Asymptotic Notations

Asymptotic notations are languages that allow us to analyze an algorithm's running time by identifying its behavior as the input size of the algorithm increases. This is also known as an algorithm's growth rate. Does the algorithm suddenly become incredibly slow when the input size grows? Does it mostly maintain its quick runtime as the input size increases? Asymptotic notation gives us the ability to answer these questions.

Let us imagine an algorithm as a function f, n as the input size, and f(n) being the running time. So for a given algorithm f, with input size n we get some resultant run time f(n). This results in a graph where the Y axis is the runtime, X axis is the input size, and plot points are the resultants of the amount of time for a given input size.

we can label a function, or algorithm, with an Asymptotic Notation in many different ways. Some examples are, we can describe an algorithm by its best case, worse case, or equivalent case. The most common is to analyze an algorithm by its worst case.

Below are the asymptotic notations used to represent time complexity of an algorithm:

1. **Big-O notation (O)**

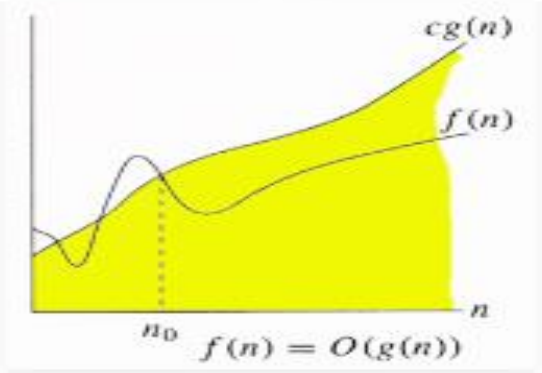
   Big-O, commonly written as **O**, is an Asymptotic Notation for the worst case, or ceiling of growth for a given function. It provides us with an ***asymptotic upper bound*** for the growth rate of runtime of an algorithm.

   If we have a non-negative function g(n) that takes a non-negative argument n then Big-O of g(n) is defined as the set of all the functions f(n) for which there exists some constants c and n_0 such that f(n) is less than or equal to c*g(n), for all n greater than or equal to n_0 .

   O (g (n)) = { f (n) : there exist constants c and n_0 ,

   f (n) ≤ cg (n), for n ≥ n_0

   }



This means that all values of n greater than some threshold n_0 , all of the functions in O(f) have values that are no greater than f. It is the measure of the longest amount of time it could possibly take for the algorithm to complete.

Properties of big-O notation

- Constant factors may be ignored as $5n^2$ and $7n^2$ are both O (n^2).
- Higher powers of n grow faster than lower powers.
- The growth rate of the sum of terms is the growth rate of its fastest growing term. E.g. $5n^3 + 5n^2$ is O (n^3).
- If function f is a polynomial of degree d, then f is O (n^d) .
- Exponential functions grow faster than powers.
- Logarithms grow more slowly than powers.
- The product of the upper bound of the functions gives an upper bound for the product of the functions.
  e.g. f is O (n^2) and g is O(log n), then f*g is O(n^2logn)

Limitations of Big-O notation

- It contains no effort to improve the efficiency of the program.
- It does not tell exhibit the potential of the constants.

  e.g. one algorithm taking $1000n^2$ time and other n^3 time. Big-O for first is O(n^2) and for the later is O(n^3). It implies that first algorithm takes less time than the other which is actually not true for $n < 1000$.

e.g. $f(n) = 7n^2 + 2n + 1$

$g(n) = n^2$

Here, the function $f(n) = O(n^2)$ if we can find the values of constants c and n such that the relation $f(n) \leq c*g(n)$ holds true.

For, $c = 7 + 2 + 1 = 10$ and $n \geq 1$ the above equation holds true.

Note that we can find different values for the constants such that the equation holds true

2. Big-Omega notation (Ω)

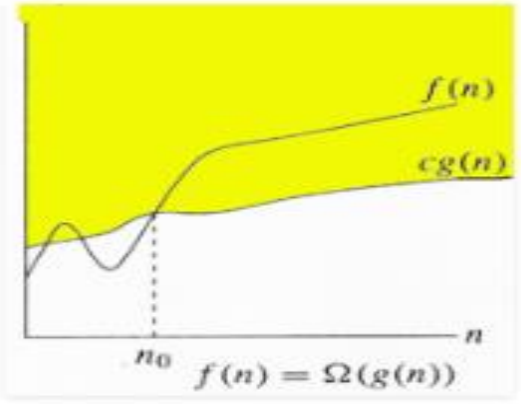
Big-Omega, commonly written as **Ω**, is an Asymptotic Notation for the best case, or a floor growth rate for a given function. It provides us with an ***asymptotic lower bound*** for the growth rate of runtime of an algorithm.

If we have a non-negative function g(n) that takes a non-negative argument n then Big-Omega of g(n) is defined as the set of all the functions f(n) for which there exists some constants c and n_0 such that f(n) is greater than or equal to c*g(n), for all n greater than or equal to n_0 .

$\Omega(g(n)) = \{ f(n)$: there exit constants c and n_0 ,

$c.g(n) \leq f(n)$, for $n \geq n_0$

}



It describes the best that can happen for a given data size.

f (n) grows at least as fast as c*g(n).

e.g. f(n) = $7n^2 + 2n + 8$

$g(n) = n^2$

Here, the function f(n) = $\Omega(n^2)$ if it satisfies the relation c*g (n) ≤ f (n)

We find constants c and n such that the above relation holds true.

So, for any n ≥ 1, the term 2n + 8 is greater than 1. If we choose c =7, for the above relation $7*n^2$ is always less than $7n^2 + 2n + 8$ for n ≥ 1

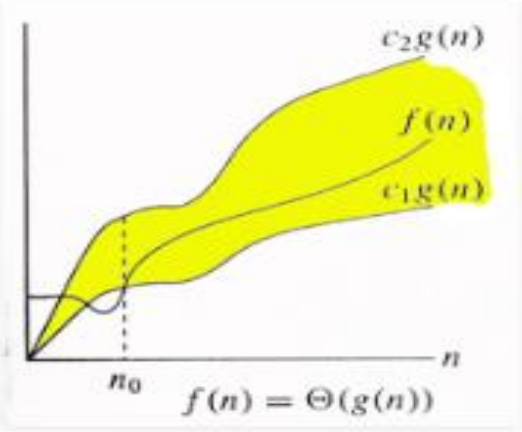
3. Theta notation (Θ)

Theta, commonly written as **Θ**, is an Asymptotic Notation to denote the ***asymptotically tight bound*** on the growth rate of runtime of an algorithm.

If we have a non-negative function g(n) that takes a non-negative argument n then Theta of g(n) is defined as the set of all the functions f(n) for which there exists some constants c1, c2 and n_0 such that f(n) is greater than or equal to c1*g(n) and less than or equal to c2*g(n), for all n greater than or equal to n_0 .

Θ (g (n)) = { f (n) : there exist constants c1, c2 and n_0 ,

c1*g(n) ≤ f (n) ≤ c2*g(n) }



e.g. f(n) = $5n^2 + 2n + 1$

$g(n) = n^2$

we have to show, $c1*g(n) \leq f(n) \leq c2*g(n)$

if we choose values c1 = 5, c2 = 8, n_0 = 1 the above relation holds true so that f(n) = Θ(g(n)).

Note:

If f(n) is O(g(n)), it means that f(n) grows no faster than g(n).

If f(n) is Ω(g(n)), it means that f(n) grows no slower than g(n).

If f(n) is Θ(g(n)), it means f(n) and g(n) grow at the same rate.

Little–o (o)

Let f(n) and g(n) be functions that map positive integers to positive real numbers. We say that f(n) is o(g(n)) (or $f(n) \in o(g(n))$) if for any real constant $c > 0$, there exists an integer constant $n_0 \geq 1$ such that $f(n) < c * g(n)$ for every integer $n \geq n_0$.

Little–Omega (ω)

Let f(n) and g(n) be functions that map positive integers to positive real numbers. We say that f(n) is ω(g(n)) (or $f(n) \in \omega(g(n))$) if for any real constant $c > 0$, there exists an integer constant $n_0 \geq 1$ such that $f(n) > c \cdot g(n)$ for every integer $n \geq n_0$.