* **Asymptotic analysis**:

The main idea of asymptotic analysis is to have a measure of the efficiency of algorithms that don’t depend on machine-specific constants and don’t require algorithms to be implemented and time taken by programs to be compared.

* **Asymptotic notations:**

Asymptotic notations are mathematical tools to represent the time complexity of algorithms for asymptotic analysis.

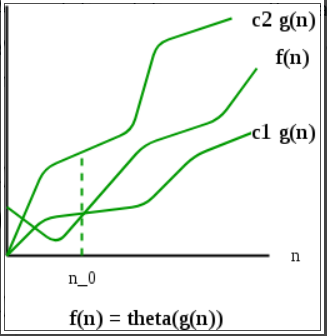
* The following 3 asymptotic notations are mostly used to represent the time complexity of algorithms.

1. **Θ Notation:** The theta notation bounds a function from above and below, so it defines exact asymptotic behavior. **It is used to determine the average case.**

**Θ(g(n)) = {f(n): there exist positive constants c1, c2 and n0 such that**

**n0 <= c1\*g(n) <= f(n) <= c2\*g(n) for all n >= n0}.**

The above definition means, if f(n) is theta of g(n), then the value f(n) is always between c1\*g(n) and c2\*g(n) for large values of n (n >= n0). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0.

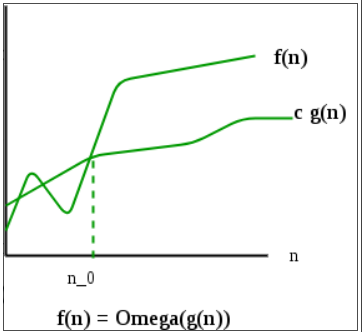


***Note: Theta provides exact bound***

1. **Big O Notation:** The Big O notation defines an upper bound of an algorithm, it bounds a function only from above. **It is used to determine the worst case.**

**O(g(n)) = { f(n): there exist positive constants c and n0 such that**

**n0 <= f(n) <= c\*g(n) for all n >= n0}**

****

***Note: Big O provides exact or upper bound***

1. **Ω Notation:** Ω notation provides an asymptotic lower bound. **It is used to determine the best case.**

**Ω (g(n)) = {f(n): there exist positive constants c and n0 such that**

**n0 <= c\*g(n) <= f(n) for all n >= n0}**

***Note: Omega provides lower bound. Omega notation is the least used notation among all three.***

P.S.:

|  |  |  |
| --- | --- | --- |
| **Scenario** | **Result** | **Description** |
| Best case | Ω (1) |  |
| Average case | Θ (n+1/2) |  |
| Worst case | O (n) | Max time which our code might take |

**Order of time complexity:** O(nn) > O(n!) > O(n3) > O(n2) > O(n.log(n)) > O(n.log(log(n))) > O(n) > O(sqrt(n)) > O(log(n)) > O(1)