An Algorithm is essentially a sequence of steps taken to solve a problem. In order to analyze algorithms and differentiate between good and bad algorithms, we need to take several things into account.

* Time Complexity:

The “time” the algorithm takes to return an output.

* Space Complexity:

The “Space and memory” resources required to run the algorithm.

* Performance:

Whether the algorithm returns an ‘exact’ or ‘approximate’ solution.

“Time and Space” complexity is most commonly known as “Computational Complexity”.

We count the total number of “atomic operations” performed by the algorithm. Each atomic operation takes a unit time. The number of operations depends on the size of input, and so the time complexity is a function of the input size.

Asymptotic Growth of the function f: This measures how fast the output f(n) grows as the input ‘n’ grows .

The notation f = O(g) is read as “f is big-Oh of g”. This is the formal definition of Big-Oh.

Essentially, There is a constant c such that when f(n) and cg(n) are graphed, the graph of cg(n) will remain higher than f(n), as n gets large. That is,

f(n) ≤ cg(n), n ≥ no

The following are the rules for Asymptotic Growth:

Let f, g, and h be functions from R+ to R+:

1. If f = O(h) and g = O(h), then f + g = O(h).
2. If f = O(g) and c is a constant greater than 0, then cf = O(g).
3. If f = O(g) and g = O(h), then f = O(h).

Here is an example for Time Complexity with a specific algorithm:

**x = Fixed Number**

**For y=1 till n**

**Compare x and A[y]**

**End**

Here the total number of operations is (n), and this is this algorithm’s Time complexity. This is the Linear Search algorithm.