Algorithms

Growth of Function:

- The growth of functions refers to how the resource requirements of an algorithm (usually time or space) change as the input size increases.
- **Key Concepts**

1. **Input Size (n)**: The input size, usually denoted by n, is the primary factor that influences the growth of an algorithm's running time or space requirements.
2. **Number of Operations**: The total number of operations an algorithm performs about the input size. This is expressed as a function f(n), which represents the growth of the resource usage.
3. **Asymptotic Analysis**: When we discuss the growth of functions, we're concerned with their behavior as the input size n approaches infinity (i.e., as n becomes very large).

Asymptotic Notation for Growth of Functions

- **Big-O**: Describes the worst-case upper bound on the growth of a function. It is the most common way to express how an algorithm behaves as input size increases.
- **Big-Omega (Ω)**: Describes the lower bound or best-case scenario for an algorithm's performance.
- **Big-Theta (Θ)**: Provides a tight bound, meaning the function grows exactly at this rate, both upper and lower bound.

Definition of Big-O

Big-O notation provides an **upper bound** on the growth rate of a function, meaning it describes the worst-case scenario in terms of time complexity or space complexity.

Formally, for a given function $f(n)$, we say that:

$$f(n) = O(g(n))$$

If and only if there are positive constants c and n_0 such that for all $n \geq n_0$:

$$f(n) \leq c \cdot g(n)$$

Where:

- $f(n)$ is the function representing the number of operations or space required by the algorithm.
- $g(n)$ is the "comparison" function that represents the asymptotic growth (e.g., n, n^2, $\log n$).
- c is a constant multiplier.
- n_0 is the point from which the Big-O approximation becomes valid.

This notation tells us that beyond a certain point n_0, the function $f(n)$ grows at most as fast as some constant multiple of $g(n)$.