Big O notation is a way to describe the efficiency of an algorithm, specifically how its runtime or memory usage changes as the input size grows. It's a way to compare the growth rates of different functions and is primarily used in computer science to analyze algorithm complexity. In essence, it provides an upper bound on how an algorithm's resources scale with the input.

Here's a more detailed explanation:

Asymptotic Analysis:

Big O notation focuses on the asymptotic behavior of an algorithm, meaning how it performs as the input size (n) approaches infinity.

Growth Rate:

It describes how the algorithm's runtime or space requirements increase as the input grows.

Worst-Case Scenario:

Big O typically represents the worst-case scenario for an algorithm's performance, meaning it describes the longest time or space it might require for a given input.

Ignoring Constants:

Big O notation generally ignores constant factors and focuses on the most significant term in the complexity function. For example, O(2n) is considered the same as O(n) because the constant 2 doesn't change the overall growth rate.