**BIG-O NOTATION**

Big O notation is a way to express the upper bound of an algorithm's runtime in terms of its input size. It is a mathematical notation used to describe the growth rate of an algorithm's time complexity, in terms of how the size of the input grows. The term "Big O" refers to the order of magnitude of the time complexity.

In computer science, algorithms are analyzed in terms of their time and space complexity. The time complexity of an algorithm refers to the amount of time it takes to execute, while the space complexity refers to the amount of memory it requires. Big O notation is used to describe the time complexity of an algorithm.

Big O notation is expressed as "O(f(n))", where "f(n)" is a function that represents the growth rate of the algorithm. It means that the time complexity of the algorithm is proportional to the growth rate of "f(n)".

Here are some common Big O notations and their meanings:

* **O(1)**: Constant time complexity. The algorithm's runtime is independent of the input size. An example of an O(1) algorithm is accessing an element in an array or a hash table.
* **O(log n)**: Logarithmic time complexity. The algorithm's runtime increases logarithmically with the input size. An example of an O(log n) algorithm is binary search in a sorted array.
* **O(n)**: Linear time complexity. The algorithm's runtime increases linearly with the input size. An example of an O(n) algorithm is iterating through an array.
* **O(n log n)**: Linearithmic time complexity. The algorithm's runtime increases at a rate of n log n with the input size. An example of an O(n log n) algorithm is merge sort.
* **O(n^2)**: Quadratic time complexity. The algorithm's runtime increases at a rate of n^2 with the input size. An example of an O(n^2) algorithm is nested loops iterating through a 2D array.
* **O(2^n)**: Exponential time complexity. The algorithm's runtime increases at a rate of 2^n with the input size. An example of an O(2^n) algorithm is generating all possible subsets of a set.

Here's an example to illustrate the concept of Big O notation:

Suppose we have an algorithm that finds the maximum value in an unsorted array of integers. One way to do this is to iterate through the entire array and compare each element to the maximum value seen so far. The time complexity of this algorithm is O(n), where "n" is the size of the array. The worst-case scenario is when the maximum value is at the end of the array, and the algorithm has to iterate through all elements.

Another way to find the maximum value is to first sort the array and then return the last element. The time complexity of this algorithm is O(n log n), where "n" is the size of the array. The worst-case scenario is when the sorting algorithm takes the most time.

Big O notation is a powerful tool for analyzing and comparing algorithms. It allows us to understand how algorithms perform as the input size grows, and it helps us choose the best algorithm for a given problem.