Algorithms: Introduction to Algorithmic Analysis - Part 1 Pre Read

Relevel
by Unacademy

Introduction

So far, we have covered fundamental concepts like functions, recursion, iteration, different types of loops, etc., and also written many cool programs around these concepts. But have you ever wondered if your program is optimal in terms of time and space requirements? Is it possible to run your program on the given machine given time and space constraints? How do you compare two algorithms which perform the same task?

Asymptotic analysis of algorithms tries to answer these questions by providing a framework to gauge the efficiency of our algorithms in terms of how much time and space it requires as a function of the input to the program.

Order of Growth of Algorithms

Let's consider two functions:
f(n) = n^2 and g(n) = n^3

From our basic understanding of algebra, we know that g(n) >= f(n). Now consider there are two algorithms, A1 and A2, for which order of growth of time complexity is f(n) and g(n), respectively. Now the question is, which one would you choose to implement, A1 or A2, where n is the input size?

It's immediately obvious to us that A1 is much better than A2 since the time complexity growth of A1 for large values of n will be significantly lower than A2.

Types of Asymptotic Notation

Algorithmic complexity or the asymptotic growth rate has broadly three different types of notations:

1. Big O notation represented by **"O"** for measuring complexity in a worst-case scenario
2. Theta notation represented by **"θ"** for measuring the average complexity of the algorithm
3. Omega notation represented by **"Ω"** for measuring complexity in best case scenario

Teaser for the Upcoming Class

- Time and space complexity calculation in detail