Overview: Why Analyze Algorithm Efficiency?

Analyzing an algorithm's efficiency helps us predict **how it will perform as input sizes grow**, without needing to run it on every possible dataset. This is crucial for understanding **scalability** and choosing the most appropriate algorithm for a problem.

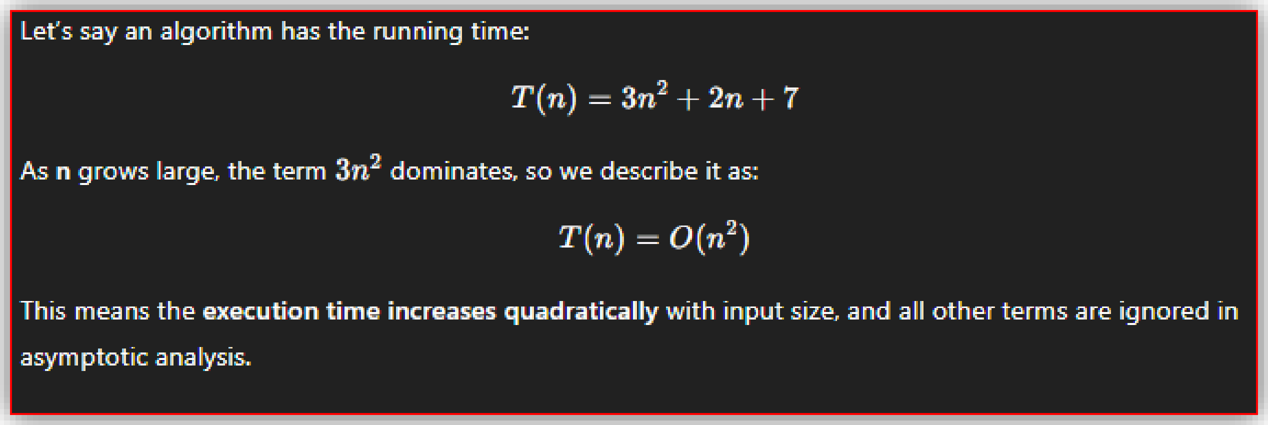
What Is Asymptotic Notation?

Asymptotic notation describes the *growth rate* of an algorithm's running time (or space usage) as the input size **n** increases. It abstracts away constants and lower-order terms, focusing only on the dominant behavior.

The key notations used are:

- **Big-O (O)** – Worst-case upper bound
- **Big-Ω (Ω)** – Best-case lower bound
- **Big-Θ (Θ)** – Tight (exact) bound
- **little-o (o)** – Strictly less than an upper bound
- **little-ω (ω)** – Strictly greater than a lower bound

How It's Used in Algorithm Analysis



Let's say an algorithm has the running time:

$$T(n) = 3n^2 + 2n + 7$$

As **n** grows large, the term $3n^2$ dominates, so we describe it as:

$$T(n) = O(n^2)$$

This means the **execution time increases quadratically** with input size, and all other terms are ignored in asymptotic analysis.

Common Function Classes (From the Book)

In Chapter 4 of the textbook, the authors identify **seven fundamental functions** used frequently in asymptotic analysis:

Function Type	Growth	Example
Constant	O(1)O(1)O(1)	Accessing array index
Logarithmic	O(log⁡n)O(\log n)O(logn)	Binary search
Linear	O(n)O(n)O(n)	Linear search
Log-linear	O(nlog⁡n)O(n \log n)O(nlogn)	Merge sort
Quadratic	O(n2)O(n^2)O(n2)	Bubble sort
Cubic	O(n3)O(n^3)O(n3)	Triple nested loops
Exponential	O(2n)O(2^n)O(2n)	Brute-force subset search

These help categorize and compare algorithms based on their efficiency.

Analytical Techniques

The book introduces several techniques for analyzing algorithms:

- **Loop counting**: Estimate the number of iterations
- **Recurrence relations**: Used for recursive algorithms (e.g., divide-and-conquer)
- **Big-O justification techniques**: Loop invariants, mathematical induction

Practical Implication

- **O(n)** is scalable.
- **O(n^2)** becomes inefficient for large input sizes.
- **O(\log n)** is highly efficient and desirable in search operations.

Understanding asymptotic notation allows developers to **optimize code**, select better algorithms, and predict performance before deployment.

Conclusion:
Asymptotic notation provides a mathematical framework for comparing algorithm efficiency independent of hardware. It's a cornerstone of algorithm design and evaluation, helping identify which solutions are scalable and practical for large datasets.