Asymptotic Notations

Understanding Algorithm Efficiency

By Farid ALVI

Introduction

- Algorithms are used to solve problems, but not all algorithms are equally efficient.

- We need to investigate efficiencies of these algorithms to utilize them constructively.

- One way is to describe their efficiency mathematically – using Asymptotic notations.

- Asymptotic notations are mathematical measures of time and space complexity.

Why Asymptotic Analysis?

- Efficiency of algorithms don't depend on machine-specific constants and don't require algorithms to be implemented and time taken by programs to be compared.

- These notations provide a concise way to express the behavior of an algorithm's time or space complexity as the input size approaches infinity.

- By using asymptotic notations, such as Big O, Big Omega, and Big Theta, we can categorize algorithms based on their worst-case, best-case, or average-case time or space complexities, providing valuable insights into their efficiency.

- The main focus is to know, how does performance scales with input size (n)?

Key Notations

- Big O (O) – Upper Bound

- Omega (Ω) – Lower Bound

- Theta (Θ) – Tight Bound

- Little o (o) – Strictly less growth

- Little omega (ω) – Strictly greater growth

Big-O Notation (O-notation):

- *Big-O notation represents the upper bound of the running time of an algorithm. Therefore, it gives the worst-case complexity of an algorithm.*

- *It is the most widely used notation for Asymptotic analysis that specifies the upper bound of a function i.e. The maximum time required by an algorithm or the worst-case time complexity.*

- *It returns the highest possible output value(big-O) for a given input - Big-O(Worst Case)*

- *It is defined as the condition that allows an algorithm to complete statement execution in the longest amount of time possible.*

Mathematical Representation of Big-O Notation:

- *$O(g(n)) = f(n)$: { there exist positive constants c and $n0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n0$ }*

For example,

- Consider the case of Insertion Sort. It takes linear time in the best case and quadratic time in the worst case. We can safely say that the time complexity of the Insertion sort is $O(n^2)$.

The Big-O notation is useful when we only have an upper bound on the time complexity of an algorithm. Many times we easily find an upper bound by simply looking at the algorithm.

Worst-case upper bound Example:

$$3n^2 + 5n + 2 = O(n^2)$$

c g(n)

f(n)

n

n_0
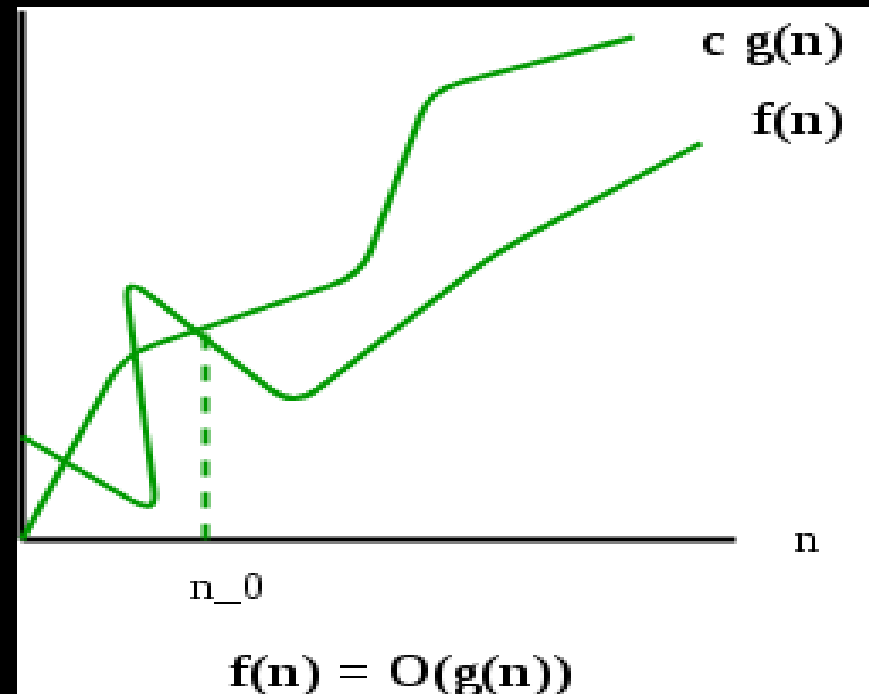
$$f(n) = O(g(n))$$

Note: $O(n^2)$ also covers linear time.

Big O Notation

- Definition: $f(n) = O(g(n))$ if $\exists$ constants c, n_0

  such that $f(n) \leq c \cdot g(n)$ for $n \geq n_0$

- Meaning: Worst-case upper bound.

- Example: $3n^2 + 5n + 2 = O(n^2)$

Omega Notation (Ω-Notation):

- *Omega notation represents the lower bound of the running time of an algorithm. Thus, it provides the best case complexity of an algorithm.*

- *The execution time serves as a lower bound on the algorithm's time complexity.*

- *It is defined as the condition that allows an algorithm to complete statement execution in the shortest amount of time.*

- *Let g and f be the function from the set of natural numbers to itself. The function f is said to be Ω(g), if there is a constant c > 0 and a natural number n0 such that c*g(n) ≤ f(n) for all n ≥ n0*

Mathematical Representation of Omega notation :

- *Ω(g(n)) = f(n) { there exist positive constants c and n0 such that 0 ≤ cg(n) ≤ f(n) for all n ≥ n0 }*

CC
Group

Example

- Let us consider the same Insertion sort example here. The time complexity of Insertion Sort can be written as $\Omega(n)$, but it is not very useful information about insertion sort, as we are generally interested in worst-case and sometimes in the average case.

If we use Θ notation to represent the time complexity of Insertion sort, we have to use two statements for best and worst cases:

- The worst-case time complexity of Insertion Sort is $\Theta(n2)$.
- The best case time complexity of Insertion Sort is $\Theta(n)$.

f(n)

c g(n)

n

n_0

$$f(n) = Omega(g(n))$$
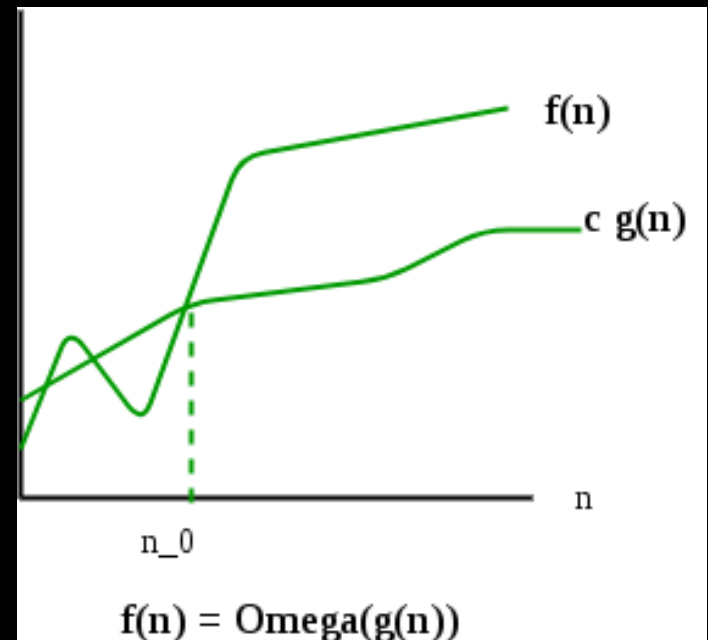
Note: O(n2) also covers linear time.

Omega Notation

- Definition: $f(n) = \Omega(g(n))$ if $\exists$ constants c, n_0 such that $f(n) \geq c \cdot g(n)$ for $n \geq n_0$

- Meaning: Best-case lower bound.

- Example: $3n^2 + 5n + 2 = \Omega(n^2)$

Theta Notation (Θ-Notation):

- *Theta notation encloses the function from above and below. Since it represents the upper and the lower bound of the running time of an algorithm, it is used for analyzing the **average-case** complexity of an algorithm.*

- *Theta (Average Case) You add the running times for each possible input combination and take the average in the average case.*

Mathematical Representation of Theta notation:

- *$\Theta(g(n)) = f(n)$: { there exist positive constants $c1$, $c2$ and $n0$ such that $0 \le c1 * g(n) \le f(n) \le c2 * g(n)$ for all $n \ge n0$}*

- ***Note:*** *$\Theta(g)$ is a set*

Example

- The above mathematical expression can be described as if f(n) is theta of g(n), then the value f(n) is always between c1 * g(n) and c2 * g(n) for large values of n (n ≥ n0). The definition of theta also requires that f(n) must be non-negative for values of n greater than n0.

- **The execution time serves as both a lower and upper bound on the algorithm's time complexity.**

- **It exist as both, most, and least boundaries for a given input value.**

- A simple way to get the Theta notation of an expression is to drop low-order terms and ignore leading constants.

For example, Consider the expression

$$3n^3 + 6n^2 + 6000 = \Theta(n^3),$$

the dropping lower order terms is always fine because there will always be a number(n) after which $\Theta(n^3)$ has higher values than $\Theta(n^2)$ irrespective of the constants involved.

Theta Notation

- Definition: f(n) = Θ(g(n)) if f(n) = O(g(n)) and f(n) = Ω(g(n))

- Meaning: Tight bound (upper & lower).

- Example: $3n^2 + 5n + 2 = \Theta(n^2)$

Little o and Little ω

- Little o: f(n) = o(g(n)) → grows strictly slower than g(n).

- Example: n = o(n²)

- Little ω: f(n) = ω(g(n)) → grows strictly faster than g(n).

- Example: n² = ω(n)

Summary

- Big O → Upper bound

- Ω → Lower bound

- Θ → Tight bound

- o / ω → Strict comparisons

- Asymptotic analysis = foundation of algorithm efficiency.

- Linear Search: O(n)

- Binary Search: O(log n)

- Merge Sort: O(n log n)

- Bubble Sort: O(n²)

- Traveling Salesman (brute force): O(n!)

References

- Introduction to Algorithms by Cormen, Leiserson, Rivest, & Stein (CLRS)

- Algorithm Design by Kleinberg & Tardos