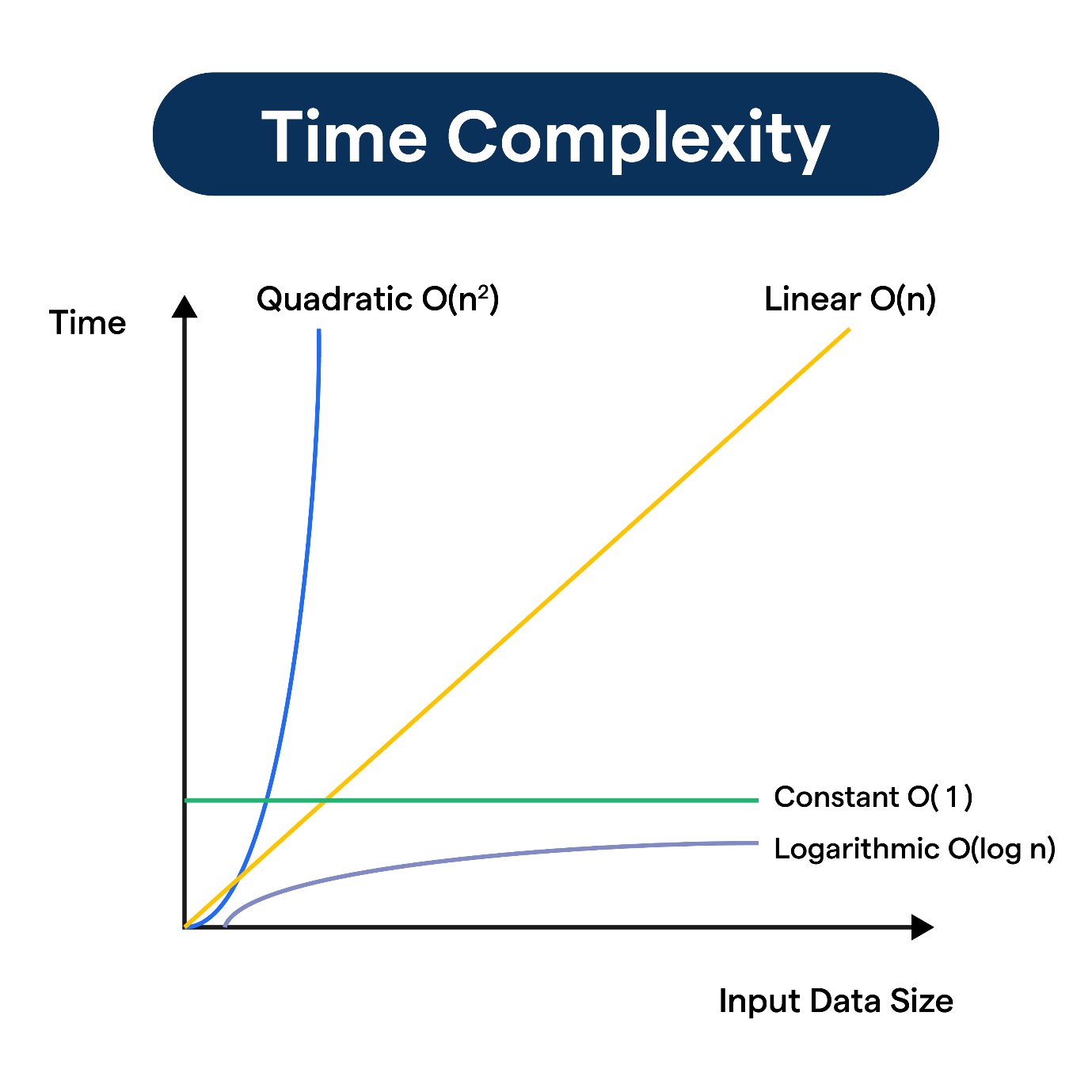
**Time Complexity**



**🖥️ When a single problem has multiple solutions, we need to analyze the algorithms**

* Algorithm analysis helps us to determine the **efficiency** of different solutions **in terms of time** and space required.
* The goal is to identify the most **optimal solution** for a given problem.

**Types of Algorithm Analysis:**

1. **Aposteriori Analysis:**
   * This is the **practical analysis** of an algorithm.
   * Dependent on factors like **compiler optimization, processor speed,** and **data characteristics**.

**Example:**

* **Measuring runtime using tools like a stopwatch**.

1. **Apriori Analysis:**
   * This is the **theoretical analysis** of an algorithm. Focuses on **mathematical estimations**. Independent of system hardware and software.
   * Performed **before implementing** the algorithm.
   * **Evaluates the algorithm based on input size and computational steps without executing it**.

**Example:**

* Count how many times a loop runs.
* Count the number of comparisons or assignments.

**Asymptotic Notations:**

Asymptotic Notations **are mathematical tools used to analyze the performance of algorithms** by understanding **how their efficiency changes as the input size grows**.

1. **Big-O Notation (O):**
   * Describes the **upper bound** of the time complexity.
   * Represents the **worst-case scenario**.
   * Example: O(n), O(n²), O (log n).

Represents the **maximum time** required for an algorithm.

**Example:** A loop that runs **n** times has time complexity **O(n).**

Lightbox

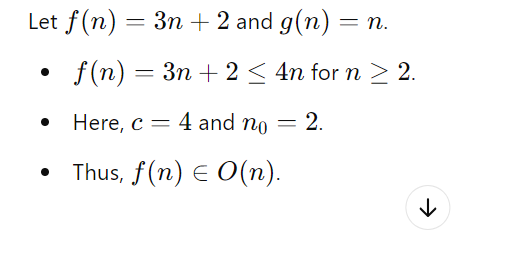
**Mathematical Definition:**

**f(n) = O(g(n))**

If there **exist positive constants** **c** and **n0**

**such that 0 ≤ f(n) ≤ cg(n) for all n ≥ n0**

**Example:**

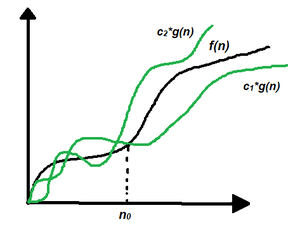


1. **Omega (Ω):**
   * Describes the **lower bound** of the time complexity.
   * Represents the **best-case scenario**.
   * **Example:** Ω(n), Ω(log n).

Represents the **minimum time** required for an algorithm.

Lightbox

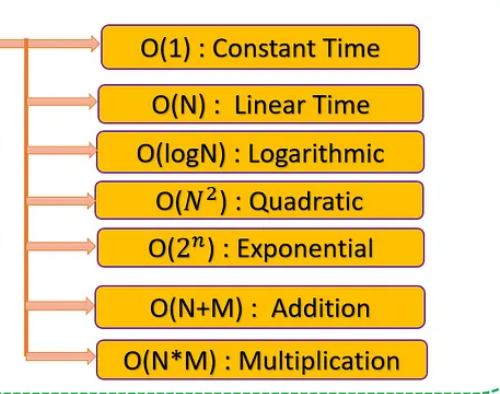
1. **Theta (Θ):**
   * Describes the **tight bound** of the time complexity.
   * Represents the **average-case scenario**.
   * Example: Θ(n), Θ(log n).



Represents the **average-case time** of an algorithm.

**Best, Worst, and Average Cases:**

1. **Best Case:**
   * The scenario where the algorithm performs the **minimum number of operations**.
   * Represented using **Ω notation**.
2. **Worst Case:**
   * The scenario where the algorithm performs the **maximum number of operations**.
   * Represented using **O notation**.
3. **Average Case:**
   * Consider **all possible inputs** the algorithm can handle.
   * Find the **time it takes for the algorithm to complete** for each input.
   * Compute the **average time** across all inputs.
   * Represented using **Θ notation**.



🖥️ **O (1)** – **Constant Time**

* The runtime doesn’t depend on the input size.
* **Example:** Accessing an element in an array.

🖥️ **O(log n)** – **Logarithmic Time**

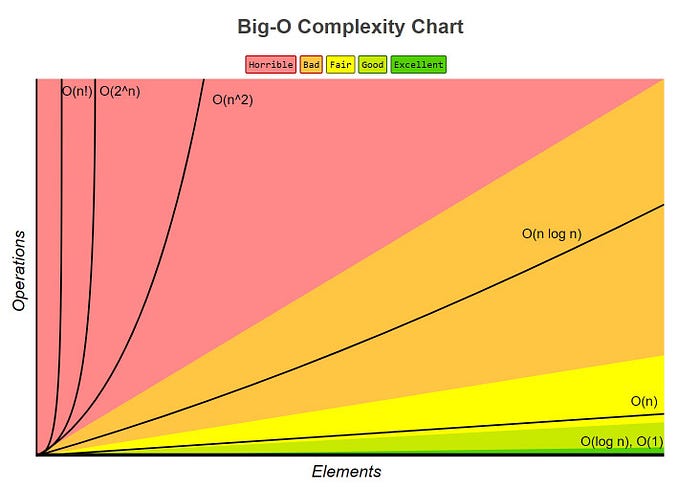
* The runtime grows slowly as input size increases.
* **Example**: Binary search.

🖥️ **O(√n​) –** **Square Root Time**

* Grows faster than **O(log n)**, but slower than **O(n).**
* **Example**:

🖥️ **O(n)** – **Linear Time**

* **Runtime grows directly with input size.**
* **Example:** Traversing an array.



🖥️ **O (n log n)** – **Linearithmic Time**

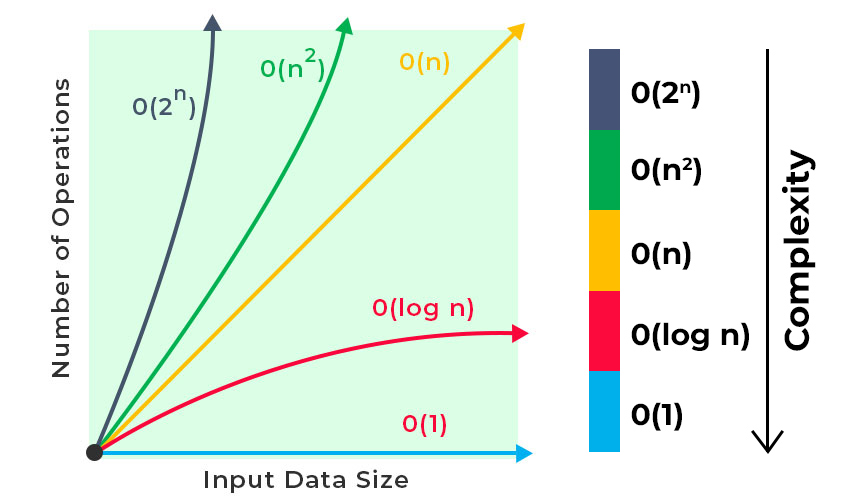
* **Common in divide-and-conquer algorithms** **like Merge Sort or Quick Sort.**

**🖥️ O(n2)** – **Quadratic Time**

* **Runtime grows proportional to the square of the input size (nested loops).**
* **Example:** **Bubble sort** or comparing all pairs in an array.

**🖥️** **O(n3) –** **Cubic Time**

* Runtime grows **proportional to the cube of the input size** (triple nested loops).
* **Example: Matrix multiplication.**



**🖥️ O(2n)** – **Exponential Time**

* **Runtime doubles with each additional input.**
* **Example: Solving the Traveling Salesman Problem using brute force**.

**🖥️** **O(n!) –** **Factorial Time**

* **Runtime grows extremely fast** (all possible permutations).
* **Example:** Solving the N-Queens problem using brute force.

**🖥️ Growth Hierarchy (Smallest to Largest):**

**O(1) < O(logn) < O(n​) < O(n) < O(nlogn) < O(n2) < O(n3) < O(2n) <O(n!)**