**Exercise 1: E-commerce Platform Search Function**

**Big O Notation**

Big O notation describes the **upper bound** of an algorithm's time or space complexity in terms of input size n. It helps in analyzing:

* **How scalable** an algorithm is
* Its **performance under load**

Search Scenarios

| **Case** | **Description** | **Example (Linear Search)** |
| --- | --- | --- |

|  |  |  |
| --- | --- | --- |
| **Best** | Target is found at the beginning | O(1) |

|  |  |  |
| --- | --- | --- |
| **Average** | Target is somewhere in the middle | O(n/2) → O(n) |

|  |  |  |
| --- | --- | --- |
| **Worst** | Target is at the end or not present | O(n) |

**Analysis**

**Time Complexity Comparison**

| **Algorithm** | **Best Case** |  |  |  | **Average Case** |  |  | **Worst Case** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Linear Search** | O(1) |  |  |  | O(n) |  |  | O(n) |
| **Binary Search** | O(1) |  |  |  | O(log n) |  |  | O(log n) |

**Exercise 7: Financial Forecasting**

**Recursion**

Recursion is a programming technique where a method **calls itself** to solve a problem by breaking it down into **smaller subproblems**.

**Use of Recursion**

* It simplifies problems that have **repetitive substructures** (e.g., Fibonacci, Tree Traversals).
* In forecasting, it can be used to compute future values based on **past trends or patterns**, e.g., growth compounded over time.

**Setup — Method Signature**

We’ll create a method that calculates **future value (FV)** based on:

* Present Value (PV)
* Growth rate (r)
* Number of years (n)

**Formula:**

FV=PV×(1+r)nFV = PV \times (1 + r)^nFV=PV×(1+r)n

**Analysis**

**Time Complexity**

* **Recursive Depth:** The method calls itself n times.
* **Time Complexity:** O(n)

**Problem: Excessive Computation**

* If the method recomputes overlapping subproblems (like in Fibonacci), it can become slow for large n

**Optimization – Use** Memoization or Iteration

Though not critical here (since only one recursive call per level), in more complex scenarios, **memoization** or converting to an **iterative approach** avoids recomputation.