**Q & A - 02\_Algorithms\_DataStructures**

**Exercise 2: E-commerce Platform Search Function**

**Q1) Explain Big O notation and how it helps in analyzing algorithms.**

**Ans)** Big O notation is a mathematical representation used to describe the time or space complexity of an algorithm in terms of input size (n). It provides a high-level understanding of how the algorithm's performance grows as the size of the input increases.

* **Purpose:** Helps in evaluating the efficiency and scalability of an algorithm.
* **Focus:** Considers the worst-case performance to give a guaranteed upper bound.
* **Example:**
  + Linear search → O(n): Performance increases linearly with input size.
  + Binary search → O(log n): Performance increases logarithmically, making it more efficient for large data sets.

**Q2) Describe the best, average, and worst-case scenarios for search operations.**

**Ans)** Understanding these scenarios helps developers choose the right algorithm depending on expected data and performance needs.

When analyzing search algorithms, it's important to consider three performance cases:

* **Best Case:**  
  The item is found in the first few comparisons.
  + *Example:* In linear search, best case is O(1).
* **Average Case:**  
  The item is located somewhere in the middle, or the search takes a typical amount of time.
  + *Example:* Linear search → O(n/2), still considered O(n).
* **Worst Case:**  
  The item is not in the list or is found at the last position.
  + *Example:* Linear search → O(n), Binary search → O(log n) (with sorted data).

**Exercise 7: Financial Forecasting**

**Q1) Explain the concept of recursion and how it can simplify certain problems.**

**Answer:**

Recursion is a programming technique in which a function calls itself to solve a smaller instance of the original problem. It is particularly useful for problems that have a repetitive or hierarchical structure, such as traversing trees, calculating factorials, or solving mathematical sequences. By reducing a complex task into simpler subproblems, recursion allows for more elegant and readable code. Each recursive function typically has a base case that defines when the recursion should stop, preventing infinite loops or stack overflow errors.

**Key Points:**

* Function calling itself with a smaller input.
* Simplifies problems with repetitive or nested structure.
* Requires a well-defined base case to stop recursion.
* Helps in solving tasks like factorials, Fibonacci series, and tree traversals.