**Exercise 2: Implementing an E-commerce Platform Search Feature**

**Understanding Algorithm Efficiency**

**Big O Notation:**

* **Definition:** Big O Notation describes the upper bound of an algorithm's runtime or space requirements as a function of input size, providing insight into its worst-case efficiency.
* **Purpose:** It allows comparison between algorithms by focusing on their performance growth relative to input size.

**Performance Scenarios:**

* **Best Case:**
  + **Linear Search:** O(1) if the target element is the first item.
  + **Binary Search:** O(1) if the target element is exactly at the midpoint.
* **Average Case:**
  + **Linear Search:** O(n) on average, requiring examination of about half of the elements.
  + **Binary Search:** O(log n) due to halving the search space with each comparison.
* **Worst Case:**
  + **Linear Search:** O(n) if the target is the last item or not present.
  + **Binary Search:** O(log n), provided the data is sorted, reducing search space exponentially.

**Setting Up the Project**

**Defining the Product Class:**

1. **Class Product:**
   * **Purpose:** Represents a product with attributes such as product ID, name, and category.
   * **Methods:**
     + **Constructor:** Initializes a Product instance with specified attributes.
     + **Getters and Setters:** Methods to access and modify product attributes.

**Implementing Search Algorithms**

**Linear Search:**

* **Method linearSearch(Product[] products, String searchTerm)**
  + **Purpose:** Searches through an array of Product objects to find one with a matching name.
  + **Functionality:** Iterates through the array, comparing each product's name with the search term. If a match is found, the corresponding product is returned; otherwise, null is returned.

**Binary Search (requires sorted array by product name):**

* **Method binarySearch(Product[] products, String searchTerm)**
  + **Purpose:** Efficiently searches for a Product in a sorted array based on product names.
  + **Functionality:** Uses a binary search approach by repeatedly dividing the search space into halves. It compares the middle product's name with the search term and adjusts the search range accordingly. If a match is found, the product is returned; if not, null is returned.
* **Method sortProductsByName(Product[] products)**
  + **Purpose:** Prepares the array for binary search by sorting products alphabetically by their names.
  + **Functionality:** Uses a sorting algorithm to arrange products in ascending order based on their names, ensuring that binary search can be applied effectively.

**Performance Analysis**

**Comparing Time Complexity:**

* **Linear Search:**
  + **Time Complexity:** O(n)
  + **Space Complexity:** O(1)
  + **Advantages:** Simple implementation, no need for sorted data.
  + **Disadvantages:** Inefficient for large datasets as it may require scanning every element.
* **Binary Search:**
  + **Time Complexity:** O(log n)
  + **Space Complexity:** O(1)
  + **Advantages:** Efficient for large datasets, as it reduces search time significantly.
  + **Disadvantages:** Requires the dataset to be sorted, which adds preprocessing time with a complexity of O(n log n).

**Determining Suitability for an E-commerce Platform:**

* **Binary Search:** Generally preferable for large datasets where the data is sorted, as it provides faster search times.
* **Linear Search:** Suitable for smaller or unsorted datasets but becomes less efficient with increasing size.