**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Steps:**

1. **Understand the Problem:**
   * Explain why data structures and algorithms are essential in handling large inventories.

* **Efficiency:** Proper data structures and algorithms allow for efficient storage, retrieval, and manipulation of data. This is crucial for operations like searching for a product, updating stock levels, and removing discontinued products.
* **Scalability:** As the inventory grows, the system must handle more data without significant performance degradation. Efficient algorithms ensure that operations remain fast, even with large datasets.
  + Discuss the types of data structures suitable for this problem.
* ArrayList (Dynamic Array): Useful for maintaining a list of products where the size can change dynamically. Provides efficient random access but can be slow for insertions and deletions in the middle.
* HashMap (Dictionary): Ideal for fast lookups, insertions, and deletions based on a unique key (e.g., productId). Provides average O(1) time complexity for these operations.
* LinkedList: Suitable for scenarios where frequent insertions and deletions are required. Provides O(1) complexity for these operations but O(n) for accessing elements by index.
* Binary Search Tree (BST): Useful if you need sorted order access to products. Provides average O(log n) time complexity for insertions, deletions.

1. **Setup:**
   * Create a new project for the inventory management system.
2. **Implementation:**
   * Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.
   * Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).
   * Implement methods to add, update, and delete products from the inventory.
3. **Analysis:**
   * Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.

* Time Complexity Analysis:
* Add Product (addProduct): The put operation of HashMap has an average time complexity of O(1).
* Update Product (updateProduct): The put operation (used for updating) has an average time complexity of O(1). Checking if the product exists (containsKey) also has an average time complexity of O(1).
* Delete Product (deleteProduct): The remove operation of HashMap has an average time complexity of O(1).
* Display Inventory (displayInventory): Iterating over the values in the HashMap has a time complexity of O(n), where n is the number of products.
  + Discuss how you can optimize these operations.
* Use a more efficient data structure if required: If more complex operations are needed (e.g., sorting products), consider using additional data structures such as TreeMap for sorted order or a combination of HashMap and LinkedList for efficient insertions and deletions along with maintaining order.
* Concurrency handling: If the system is to be used in a multi-threaded environment, consider using concurrent data structures like ConcurrentHashMap to avoid synchronization issues.