Exercise 1: Inventory Management System

# 1. UNDERSTAND THE PROBLEM:

# Explain why data structures and algorithms are essential in handling large inventories.

 **Efficiency:** With large inventories, efficient data storage and retrieval become crucial to maintain system performance. Choosing the right data structure helps in optimizing operations like searching for a product, updating stock levels, and removing items.

 **Scalability:** The inventory system should be scalable, meaning it should handle increasing numbers of products without a significant drop in performance. Efficient algorithms ensure that operations remain fast as the inventory grows.

 **Data Integrity:** Proper data structures help maintain the integrity of the inventory data by preventing duplicates, handling concurrent updates, and ensuring that inventory operations are accurate.

# Discuss the types of data structures suitable for this problem.

 **ArrayList:** Useful for maintaining an ordered list of products. However, searching, updating, and deleting can be slow for large inventories as these operations have a time complexity of O(n).

 **HashMap:** Provides fast access, insertion, and deletion operations with an average time complexity of O(1). It’s suitable for scenarios where products can be uniquely identified by a key (e.g., productId).

 **TreeMap:** Maintains an ordered set of keys and offers logarithmic time complexity for insertion, deletion, and access (O(log n)). This can be useful if you need to maintain the order of products based on some criteria (e.g., productId).

# 2. ANALYSIS:

# Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.

 **Add Product:**

* `**HashMap.put()`** method has an average time complexity of O(1).

 **Update Product:**

* Updating a product is effectively the same as adding, so it also has an average time complexity of O(1).

 **Delete Product:**

* `**HashMap.remove()`** method has an average time complexity of O(1).

# Discuss how you can optimize these operations.

 **Efficient Lookups:** Using a **HashMap** is already an optimized choice for fast lookups, insertions, and deletions.

 **Memory Optimization:** If memory usage becomes a concern, you could consider using a more memory-efficient data structure, or compressing data (e.g., storing price as an integer representing cents instead of a double).

 **Concurrency:** If the inventory system needs to handle concurrent access, consider using **ConcurrentHashMap** instead of **HashMap** to ensure thread-safe operations.