**Exercise 1: Inventory Management System**

**Step 1: Understand the Problem**

**Importance of Data Structures and Algorithms in Handling Large Inventories**

Data structures and algorithms are essential in handling large inventories because they:

1. **Optimize Performance:** Efficient data structures ensure that operations such as adding, updating, and deleting products are performed quickly, which is crucial for large inventories.
2. **Improve Scalability:** Proper algorithms allow the system to handle growing data without significant performance degradation.
3. **Enhance Data Retrieval:** Effective data structures enable fast retrieval of information, which is important for inventory tracking and management.
4. **Reduce Memory Usage:** Efficient data structures can minimize memory consumption, which is important for large-scale systems.

**Suitable Data Structures for Inventory Management**

1. **ArrayList:** Provides dynamic array capabilities with efficient index-based access. Suitable for situations where frequent random access is needed.
2. **HashMap:** Offers average O(1) time complexity for insertion, deletion, and lookup operations. Suitable for scenarios where fast access to products based on a unique identifier (e.g., productId) is required.
3. **TreeMap:** Maintains a sorted order of elements and provides log(n) time complexity for insertion, deletion, and lookup operations. Suitable if the inventory needs to be kept in a sorted order.
4. **LinkedList:** Useful for operations that involve frequent insertions and deletions, especially when the location is unknown beforehand.

**Time Complexity of Operations**

1. **Add Product:**
   * Time Complexity: O(1) on average.
   * Explanation: Inserting a product into the HashMap involves computing the hash and placing the product in the appropriate bucket.
2. **Update Product:**
   * Time Complexity: O(1) on average.
   * Explanation: Updating a product involves retrieving the product by key and updating its attributes.
3. **Delete Product:**
   * Time Complexity: O(1) on average.
   * Explanation: Deleting a product involves removing the product by key from the HashMap.

**Optimizing Operations**

1. **Minimize Collisions:**
   * Ensure a good hash function to minimize collisions in the HashMap, which keeps the average time complexity close to O(1).
2. **Efficient Rehashing:**
   * Implement efficient rehashing strategies to handle load factors, ensuring the HashMap remains performant as the number of entries grows.
3. **Thread Safety:**
   * If concurrent access is required, use concurrent data structures like ConcurrentHashMap to avoid issues with synchronization.