**Exercise 1: Inventory Management System**

**Scenario:**

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

**Steps:**

***Understand the Problem:***

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

***Ans:*** Efficient data structures and algorithms are vital for handling large inventories because they allow for quick and efficient storage, retrieval, and manipulation of data. With a large number of products, operations like adding, updating, and deleting inventory items need to be performed efficiently to ensure the system remains responsive and scalable.

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

***Ans:*** ArrayList: Good for maintaining a list of products where random access is required. However, adding or removing elements can be inefficient because it may require shifting elements.

HashMap: Excellent for scenarios requiring fast access to elements based on a key (e.g., productId). Provides average O(1) time complexity for add, update, and delete operations.

LinkedList: Useful for scenarios where frequent additions and deletions occur. However, accessing elements is slower (O(n) time complexity).

***Setup:***

Create a new project for the inventory management system.

Code: ProductInventory.java

***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.

Code: ProductInventory.java

***Analysis:***

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

***Ans:* Add Product (addProduct)**: Average Case: O(1) because adding a key-value pair to a HashMap is typically O(1).

**Update Product (updateProduct)**: Average Case: O(1) because accessing an element by key

and updating it is typically O(1).

**Delete Product (deleteProduct)**:Average Case: O(1) because removing a key-value pair from a HashMap is typically O(1).

Discuss how you can optimise these operations.

***Ans:*** We can optimise these operations by ensuring the HashMap has an appropriate initial capacity to minimise rehashing operations.

We can also implement additional indexing or caching mechanisms that can further optimise retrieval times for specific queries.