**Exercise 1: Inventory Management System**

**1. Understand the Problem**

**Why data structures and algorithms are essential in handling large inventories:**

Efficient data structures and algorithms are important because they help store, organize, and retrieve data quickly and efficiently. With a large inventory, tasks such as adding, updating, deleting, and searching for items need to be done quickly to maintain optimal performance and user experience Data appropriate systems and algorithms ensure minimal downtime and efficient use of memory.

**Types of data structures suitable for this problem:**

* **ArrayList:** Dynamic array structure that allows fast access and modification.
* **HashMap:** Provides average O(1) time complexity for add, update, and delete operations due to its hashing mechanism.

**2. Setup**

**Create a new project for the inventory management system:**

In your IDE (e.g., IntelliJ IDEA or Eclipse), create a new Java project named InventoryManagementSystem.

**3. Implementation**

**Define the Product class:**

public class Product {

private int productId;

private String productName;

private int quantity;

private double price;

public Product(int productId, String productName, int quantity, double price) {

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

// Getters and Setters

public int getProductId() {

return productId;

}

public void setProductId(int productId) {

this.productId = productId;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

}

**Choose a data structure to store the products:**

Using HashMap to store products with productId as the key for efficient operations.

import java.util.HashMap;

public class Inventory {

private HashMap<Integer, Product> products;

public Inventory() {

this.products = new HashMap<>();

}

public void addProduct(Product product) {

products.put(product.getProductId(), product);

}

public void updateProduct(int productId, Product updatedProduct) {

products.put(productId, updatedProduct);

}

public void deleteProduct(int productId) {

products.remove(productId);

}

public Product getProduct(int productId) {

return products.get(productId);

}

}

**4. Analysis**

**Time complexity of each operation:**

* **Add:** O(1) - Average case
* **Update:** O(1) - Average case
* **Delete:** O(1) - Average case

**Optimizations:**

Using a HashMap is already optimized for average O(1) time complexity for add, update, and delete operations due to its efficient hashing mechanism. However, ensuring a good hash function and managing collisions effectively can further optimize performance.