Exercise 1: Inventory Management System

Understand the Problem:

- Data structures and algorithms are vital in handling large inventories efficiently.Proper structures improve lookup, insertion, and deletion performance.

- Suitable data structures: ArrayList (for small datasets), HashMap (for fast access using productId), TreeMap (if sorted order is needed).

Implementation:

Defining a Product class and use HashMap to store product entries.

Implementing methods to add, update, delete, and display products.

```
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 String getProductName() { return productName; }
    public int getQuantity() { return quantity; }
    public double getPrice() { return price; }

    public void setProductName(String productName) { this.productName = productName; }
    public void setQuantity(int quantity) { this.quantity = quantity; }
```

```java
    public void setPrice(double price) { this.price = price; }

    @Override
    public String toString() {
        return "Product[ID=" + productId + ", Name=" + productName +
               ", Qty=" + quantity + ", Price=" + price + "]";
    }
}



import java.util.HashMap;
import java.util.Map;

public class InventoryManager {
    private Map<Integer, Product> inventory = new HashMap<>();

    public void addProduct(Product product) {
        inventory.put(product.getProductId(), product);
    }

    public void updateProduct(int productId, String name, int qty, double price) {
        Product p = inventory.get(productId);
        if (p != null) {
            p.setProductName(name);
            p.setQuantity(qty);
            p.setPrice(price);
        }
    }

    public void deleteProduct(int productId) {
        inventory.remove(productId);
    }

    public void displayInventory() {
        for (Product p : inventory.values()) {
            System.out.println(p);
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        InventoryManager manager = new InventoryManager();

        Product p1 = new Product(101, "Mouse", 50, 299.99);
        Product p2 = new Product(102, "Keyboard", 30, 599.99);

        manager.addProduct(p1);
        manager.addProduct(p2);
        manager.displayInventory();

        manager.updateProduct(101, "Wireless Mouse", 45, 349.99);
        manager.displayInventory();

        manager.deleteProduct(102);
        manager.displayInventory();
    }
}
```

4. Analysis:

- Using HashMap, the average time complexity for add, update, and delete is O(1).

- Displaying inventory takes O(n) as it iterates over all products.

Optimization:

- Ensure unique productIds to avoid collisions.

- For concurrency, use ConcurrentHashMap.

- For persistent storage, integrate with a file or database.