**DATA STRUCTURES AND ALGORITHMS:**

**Exercise 1:Inventory Management:**

**➤ Why are Data Structures and Algorithms important?**

When we deal with large inventories in a warehouse, there can be hundreds or thousands of products. To handle this efficiently, we need proper data structures that help in storing, searching, and updating data quickly. Algorithms also help to reduce the time taken for these operations. Without them, the system might become slow and unresponsive, especially as the number of products grows.

**➤ Suitable Data Structures**

Some data structures suitable for this problem are:

* **ArrayList** – It's easy to use and good for storing products in a list format, especially when order matters.
* **HashMap** – It's best when we want to access products quickly using a unique key like productId.
* **LinkedList or TreeMap** – These can be used if we want to maintain insertion order or sorted order of products.

For my implementation, I preferred using **ArrayList**, as it's simple and beginner-friendly, and I used loops to go through the list for operations like update and delete.

**2. Setup**

I created a new Java project named InventoryManagementSystem. Inside it, I made three Java files:

1. Product.java – to represent each product.
2. Inventory.java – to manage the list of products and their operations.
3. Main.java – to run the program and test the functionality.

3. **Implementation:**

**Product.java**

public class Product {

    int id;

    String name;

    int quantity;

    double price;

    Product(int id, String name, int quantity, double price) {

        this.id = id;

        this.name = name;

        this.quantity = quantity;

        this.price = price;

    }

    void printDetails() {

        System.out.println("Product ID: " + id);

        System.out.println("Product Name: " + name);

        System.out.println("Quantity: " + quantity);

        System.out.println("Price: " + price);

        System.out.println();

    }

}

**Inventory.java**

import java.util.ArrayList;

public class Inventory {

    ArrayList<Product> products = new ArrayList<>();

    // Add product

    void addProduct(Product p) {

        products.add(p);

    }

    // Update product

    void updateProduct(int id, int quantity, double price) {

        for (int i = 0; i < products.size(); i++) {

            if (products.get(i).id == id) {

                products.get(i).quantity = quantity;

                products.get(i).price = price;

            }

        }

    }

    // Delete product

    void deleteProduct(int id) {

        for (int i = 0; i < products.size(); i++) {

            if (products.get(i).id == id) {

                products.remove(i);

                break;

            }

        }

    }

    void showAll() {

        for (int i = 0; i < products.size(); i++) {

            products.get(i).printDetails();

        }

    }

}

**Main.java**

public class Main {

    public static void main(String[] args) {

        Inventory inventory = new Inventory();

        Product p1 = new Product(1, "Pen", 100, 5.0);

        Product p2 = new Product(2, "Notebook", 50, 30.0);

        inventory.addProduct(p1);

        inventory.addProduct(p2);

        System.out.println("All Products:");

        inventory.showAll();

        inventory.updateProduct(1, 120, 4.5);

        inventory.deleteProduct(2);

        System.out.println("After update and delete:");

        inventory.showAll();

    }

}

**OUTPUT:**

A screenshot of a computer

AI-generated content may be incorrect.

**4. Analysis**

**➤ Time Complexity (Using ArrayList)**

| **Operation** | **Time Complexity** | **Reason** |
| --- | --- | --- |
| **Add** | O(1) | When we add a product at the end of the list, it takes constant time. |
| **Update** | O(n) | We have to go through the list to find the product with the given ID. |
| **Delete** | O(n) | Similar to update, we have to search for the item before removing it. |
| **Display** | O(n) | We loop through all products one by one to show them. |

**➤ How to Optimize These Operations**

* For small inventories, using ArrayList is fine and easy to manage.
* But for larger inventories (hundreds or thousands of items), it is better to use a HashMap.

**Why?**  
Because with a HashMap, we can access, update, or delete a product using its ID in constant time — O(1) on average. That’s much faster than searching through the list every time.

So if performance becomes a concern in the future, switching from ArrayList to HashMap<Integer, Product> would be a better choice, especially when product IDs are unique and frequently used for lookup.