**WEEK 1 - ALGORITHMS AND DATA STRUCTUERS**

**EXERCISE 1:** Inventory Management System

**SCENARIO:**

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

**IMPORTENCE OF DATA STRUCTURES & ALGORITHMS**

* To enable **fast lookup** of products using product IDs.
* To **reduce time complexity** in real-time updates and deletions.
* Algorithms help handle **bulk imports, filtering**, and **report generation** effectively.

**SUITABLE DATA STRUCTURES:**

* **HashMap<Integer, Product>** – Ideal for constant-time (O(1)) search, update, and delete operations using productId as the key.
* **ArrayList<Product>** – Good for ordered data, but not efficient for lookup and deletion by ID (O(n)).

### SETUP AND IMPLEMENDATION:

### Product.java

public class Product {

int id;

String name;

int qty;

double price;

public Product(int id, String name, int qty, double price) {

this.id = id;

this.name = name;

this.qty = qty;

this.price = price;

}

public String toString() {

return "ID: " + id + ", Name: " + name + ", Qty: " + qty + ", Price: ₹" + price;

}

}

### Inventory.java

java

CopyEdit

import java.util.\*;

public class Inventory {

HashMap<Integer, Product> items = new HashMap<>();

void add(Product p) {

items.put(p.id, p);

System.out.println("Product added.");

}

void update(int id, int qty, double price) {

Product p = items.get(id);

if (p != null) {

p.qty = qty;

p.price = price;

System.out.println("Product updated.");

}

}

void delete(int id) {

items.remove(id);

System.out.println("Product deleted.");

}

void view() {

for (Product p : items.values()) {

System.out.println(p);

}

}

}

### Main.java

import java.util.;

public class Main {

public static void main(String[] args) {

Inventory inv = new Inventory();

Scanner sc = new Scanner(System.in);

inv.add(new Product(1, "Laptop", 5, 45000));

inv.add(new Product(2, "Mouse", 10, 400));

inv.view();

inv.update(1, 3, 43000);

inv.delete(2);

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

inv.view();

sc.close();

}

}

## OUTPUT:

Product added.

Product added.

ID: 1, Name: Laptop, Qty: 5, Price: ₹45000.0

ID: 2, Name: Mouse, Qty: 10, Price: ₹400.0

Product updated.

Product deleted.

After update and delete:

ID: 1, Name: Laptop, Qty: 3, Price: ₹43000.0

**TIME COMPLEXITY ANALYSIS:**

## In the inventory system, operations like **add, update, delete, and search** are highly optimized using a HashMap, providing constant time complexity ****O(1)**** due to direct key-based access. However, the **view all products** operation takes ****O(n)**** time as it requires iterating over all entries to display the complete inventory.

## OPTIMIZATION

### 1. ****Add Product****

* Already fast using HashMap (O(1)).
* Add a check with containsKey() to prevent duplicate IDs.

### 2. ****Update Product****

* Efficient with direct access (O(1)).
* Validate product ID before updating.
* For multiple updates, use batch processing to reduce function calls.

### 3. ****Delete Product****

* Quick removal with HashMap (O(1)).
* Check if product exists before deleting to improve reliability.

### 4. ****View All Products****

* Takes O(n) time — can't be reduced.
* Add **pagination** for better readability.
* Add optional **sorting** by price or quantity for more meaningful display.