**ALGORITHM DATASTRUCTURES**

**Exercise 1: Inventory Management System**

**Scenario:**

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

**Steps:**

1. **Understand the Problem:**
   * Explain why data structures and algorithms are essential in handling large inventories.
   * Discuss the types of data structures suitable for this problem.
2. **Setup:**
   * Create a new project for the inventory management system.
3. **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.
4. **Analysis:**
   * Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.
   * Discuss how you can optimize these operations.

**NEED OF DATASTRUCTURE:**

**Efficient Data Storage & Organization:** Helps store and manage product details systematically for easy access and modification.

**Fast Retrieval & Updates:** Structures like HashMap allow constant-time (O(1)) access using product IDs, ensuring quick lookups and edits.

**Scalability:** Suitable data structures handle large inventories efficiently without performance drop as data grows.

**Data Integrity:** Ensures uniqueness (e.g., no duplicate product IDs) and maintains consistent inventory records.

**DATASTRUCTURE USED:**

**Hashmap**

* The key is the productId (a unique identifier for each product).
* The value is the Product object containing name, quantity, and price.

**PROGRAM:**

import java.util.\*;

public class InventoryManagementSystem {

public static void main(String[] args) {

Inventory inventory = new Inventory();

inventory.addProduct(new Product("P101", "Keyboard", 50, 899.99));

inventory.addProduct(new Product("P102", "Mouse", 100, 499.49));

inventory.addProduct(new Product("P103", "Monitor", 30, 12499.00));

inventory.updateProduct("P102", 80, 475.00);

inventory.deleteProduct("P103");

inventory.printInventory();

}

}

class Product {

private String productId;

private String productName;

private int quantity;

private double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public String getProductId() {

return productId;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public void setPrice(double price) {

this.price = price;

}

public String toString() {

return "ID: " + productId + ", Name: " + productName + ", Qty: " + quantity + ", Price: ₹" + price;

}

}

class Inventory {

private Map<String, Product> products = new HashMap<>();

public void addProduct(Product product) {

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

}

public void updateProduct(String productId, int quantity, double price) {

Product product = products.get(productId);

if (product != null) {

product.setQuantity(quantity);

product.setPrice(price);

}

}

public void deleteProduct(String productId) {

products.remove(productId);

}

public void printInventory() {

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

System.out.println(p);

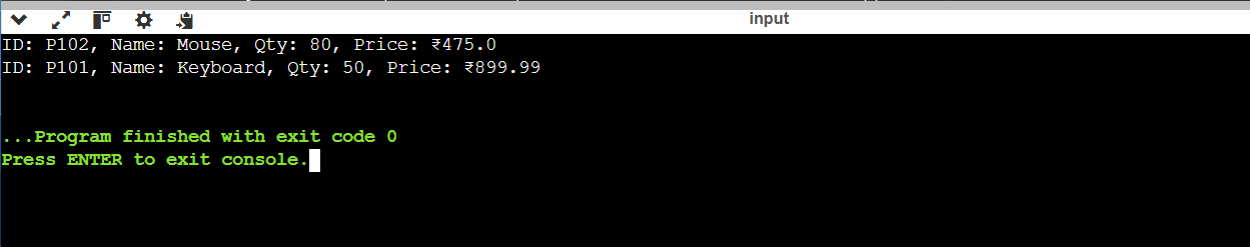
}

}

}

**PROGRAM:**

**OUTPUT:**

****

**ANALYSIS:**

**Time Complexity**

* Add Product: O(1) — Direct insertion using product ID as the key.
* Update Product: O(1) — Quick lookup and update by key.
* Delete Product: O(1) — Instant removal using key.
* Search Product: O(1) — Fast retrieval with key-based access.
* Print All Products: O(n) — Iterates over all n products.

**Space Complexity**

* Overall Space: O(n) — Stores n product objects in the HashMap.
* Each product includes attributes: productId, productName, quantity, and price.
* The HashMap itself consumes extra memory for storing key-value mappings and internal buckets.

**WAYS TO OPTIMIZE**

Use HashMap for fast lookups; TreeMap for sorted data.

Minimize memory by reusing objects and using primitives.

Batch process add/update/delete operations.

Apply caching or lazy loading for large inventories.