**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. Problem Understanding**

**Importance of Data Structures and Algorithms in Inventory Management**

* **Efficiency:** Large inventories require efficient data structures for fast operations like searching, adding, updating, and deleting products. Poor choices can slow down the system as the inventory grows**.**
* **Scalability:** The ability of algorithms to handle increasing data size is crucial to maintaining system performance.
* **Memory Management:** Optimal data structures prevent memory issues and ensure the system can manage large datasets effectively.

**Suitable Data Structures**

* **ArrayList**: Ideal for storing and iterating through products, especially when quick index-based access is needed.
* **HashMap:** Provides O(1) average time complexity for search, update, and delete operations, making it perfect for operations based on unique keys like productId.
* **LinkedList:** Useful for frequent insertions and deletions, though less efficient for random access.

**2. Setup**

**Project Creation for Inventory Management**

* **Structure:** Create a Java project with a main class for user interaction and another for managing products and inventory.

**3. Implementation:**

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Scanner;

class Product {

int productId;

String productName;

int quantity;

double price;

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

this.productId = productId;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

@Override

public String toString() {

return "Product ID: " + productId + ", Name: " + productName + ", Quantity: " + quantity + ", Price: $" + price;

}

}

class Inventory {

private HashMap<Integer, Product> productMap;

public Inventory() {

this.productMap = new HashMap<>();

}

// Add a new product

public void addProduct(Product product) {

if (productMap.containsKey(product.productId)) {

System.out.println("Product ID already exists. Use updateProduct to modify it.");

} else {

productMap.put(product.productId, product);

System.out.println("Product added: " + product);

}

}

// Update an existing product

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

if (productMap.containsKey(productId)) {

Product product = productMap.get(productId);

product.productName = productName;

product.quantity = quantity;

product.price = price;

System.out.println("Product updated: " + product);

} else {

System.out.println("Product ID not found. Use addProduct to add a new product.");

}

}

// Delete a product

public void deleteProduct(int productId) {

if (productMap.containsKey(productId)) {

Product removedProduct = productMap.remove(productId);

System.out.println("Product deleted: " + removedProduct);

} else {

System.out.println("Product ID not found.");

}

}

// Display all products

public void displayAllProducts() {

if (productMap.isEmpty()) {

System.out.println("No products in the inventory.");

} else {

for (Product product : productMap.values()) {

System.out.println(product);

}

}

}

}

public class InventoryManagementSystem {

public static void main(String[] args) {

Inventory inventory = new Inventory();

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nInventory Management System");

System.out.println("1. Add Product");

System.out.println("2. Update Product");

System.out.println("3. Delete Product");

System.out.println("4. Display All Products");

System.out.println("5. Exit");

System.out.print("Choose an option: ");

int choice = scanner.nextInt();

switch (choice) {

case 1:

System.out.print("Enter Product ID: ");

int id = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter Product Name: ");

String name = scanner.nextLine();

System.out.print("Enter Quantity: ");

int quantity = scanner.nextInt();

System.out.print("Enter Price: ");

double price = scanner.nextDouble();

inventory.addProduct(new Product(id, name, quantity, price));

break;

case 2:

System.out.print("Enter Product ID to update: ");

int updateId = scanner.nextInt();

scanner.nextLine(); // Consume newline

System.out.print("Enter New Product Name: ");

String newName = scanner.nextLine();

System.out.print("Enter New Quantity: ");

int newQuantity = scanner.nextInt();

System.out.print("Enter New Price: ");

double newPrice = scanner.nextDouble();

inventory.updateProduct(updateId, newName, newQuantity, newPrice);

break;

case 3:

System.out.print("Enter Product ID to delete: ");

int deleteId = scanner.nextInt();

inventory.deleteProduct(deleteId);

break;

case 4:

inventory.displayAllProducts();

break;

case 5:

System.out.println("Exiting...");

scanner.close();

System.exit(0);

break;

default:

System.out.println("Invalid option. Please try again.");

}

}

}

}

**4. Analysis**

**Time Complexity of Operations**

* **Add Product (addProduct)**
  + **Time Complexity:** O(1) on average due to efficient HashMap insertions.
  + **Optimization:** Ensure minimal hash collisions with well-chosen hash functions.
* **Update Product (updateProduct)**
  + **Time Complexity**: O(1) on average, as it involves a quick lookup and update in the HashMap.
  + **Optimization:** Focus on minimizing hash collisions to maintain optimal performance.
* **Delete Product (deleteProduct)**
  + **Time Complexity:** O(1) on average, requiring just a lookup and deletion.
  + **Optimization:** Maintain effective hash functions for efficient deletions.
* **Display All Products (displayAllProducts)**
  + **Time Complexity:** O(n), where n is the number of products, due to iteration through the HashMap.
  + **Optimization:** Reduce the number of operations inside the loop to enhance efficiency.

**Optimizing Operations**

* **Hashing Strategy:** Ensure well-distributed hash functions to minimize collisions and maintain O(1) operations.
* **Concurrency:** Use ConcurrentHashMap for safe access in multi-threaded environments**.**
* **Batch Processing:** Implement batch methods for bulk updates or deletions to reduce overhead.