**Exercise 1: Inventory Management System**

1. Explain why data structures and algorithms are essential in handling large inventories.

* **Efficient Storage**: Data structures help in organizing and storing inventory data efficiently. This ensures that space is utilized optimally.
* **Fast Retrieval**: Algorithms enable quick search, retrieval, and update operations, which are crucial for real-time inventory management.
* **Scalability**: Proper data structures and algorithms ensure that the system can handle a growing number of products without significant performance degradation.
* **Complex Operations**: Advanced algorithms can be used for inventory forecasting, restocking alerts, and optimization, which are important for warehouse management.

1. Discuss the types of data structures suitable for this problem.

* **ArrayList**: Good for storing a dynamic list of products where order matters and random access is needed. It provides average O(1) time complexity for accessing elements but O(n) for insertions and deletions in the middle.
* **HashMap**: Ideal for situations where fast lookups, insertions, and deletions are needed. It offers average O(1) time complexity for these operations. It maps product IDs to product details.
* **LinkedList**: Useful if frequent insertions and deletions are required, but less efficient for random access as it provides O(n) time complexity for accessing elements.
* **TreeMap**: Useful if you need a sorted map of products. It provides O(log n) time complexity for insertions, deletions, and lookups.

1. Define a class Product with attributes like **productId**, **productName**, **quantity**, and **price**.

public class Product {

private int productId;

private String productName;

private int quantity;

private double price;

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

this.productId = id;

this.productName = productName;

this.quantity = quantity;

this.price = price;

}

public void setProductId(int productId) {

this.productId = productId;

}

public int getProductId() {

return productId;

}

public int getQuantity() {

return quantity;

}

public void setQuantity(int quantity) {

this.quantity = quantity;

}

public double getPrice() {

return price;

}

public void setPrice(double price) {

this.price = price;

}

public String getProductName() {

return productName;

}

public void setProductName(String productName) {

this.productName = productName;

}

}

1. Choose an appropriate data structure to store the products (e.g., ArrayList, HashMap).

Implement methods to add, update, and delete products from the inventory.

import java.util.HashMap;

import java.util.Map;

public class Inventory {

HashMap<String,Product> products;

Inventory(){

products = new HashMap<>();

}

void addProduct(Product product){

this.products.put(product.getProductName().toLowerCase() , product);

}

void removeProduct(Product product){

this.products.remove(product.getProductName().toLowerCase());

}

void updateProduct(Product product,double price){

product.setPrice(price);

}

void displayProducts(){

System.out.println("Number of Products in Inventory : " + products.size());

System.out.printf("%-20s %-20s %s\n","Product Name","Quantity","Price");

for(Map.Entry<String,Product> product : products.entrySet()){

Product currentProduct = product.getValue();

System.out.printf("%-20s %-20s %s\n",currentProduct.getProductName(),currentProduct.getQuantity(),currentProduct.getPrice());

}

}

}

**Main Method**

import java.util.HashMap;

public class Main {

public static void main(String[] args) {

Inventory inventory = new Inventory();

Product rice = new Product(100,"Rice",1,75.0);

Product salt = new Product(101,"Salt",1,24.0);

Product sugar = new Product(102,"Sugar",1,40.0);

inventory.addProduct(rice);

inventory.addProduct(salt);

inventory.addProduct(sugar);

System.out.println("Before Deleting a Product");

inventory.displayProducts();

inventory.removeProduct(salt);

System.out.println("After deleting a Product");

inventory.displayProducts();

inventory.updateProduct(sugar , 38.0);

System.out.println("After updating a Product");

inventory.displayProducts();

}

}

1. Analyze the time complexity of each operation (add, update, delete) in your chosen data structure.

**Time Complexity Analysis:**

* **Add Product**: The time complexity is O(1) on average since HashMap allows for constant-time insertion.
* **Update Product**: The time complexity is O(1) on average since updating an existing entry in a HashMap is also a constant-time operation.
* **Delete Product**: The time complexity is O(1) on average because deleting an entry from a HashMap is a constant-time operation.
* **Display Products**: The time complexity is O(n), where n is the number of products in the inventory, as we need to iterate over all entries to display them.

1. Discuss how you can optimize these operations.

**Optimization:**

* **Efficient Use of Memory**: Ensure that the HashMap capacity is managed efficiently to avoid excessive memory usage and rehashing.
* **Concurrency**: If the inventory system is used in a multi-threaded environment, use Concurrent HashMap instead of HashMap to handle concurrent updates and accesses.