# Exercise 1: Inventory Management System

### 1. Understand the Problem

When managing a large inventory, it's important to store product information in a way that makes it easy and fast to add new products, update existing ones, or remove products that are no longer available. Choosing the right data structure helps us quickly find products by their ID without searching through everything one by one. For example, using a dictionary (or HashMap) lets us access products directly using their unique IDs.

**Suitable Data Structures:**

- HashMap (Dictionary)
- ArrayList (Dynamic Array)
- Database Systems

### 2&3. Setup and Implementation:

```
import java.util.HashMap;

class Product {
    String productId, productName;
    int quantity;
    double price;

    public Product(String id, String name, int qty, double price) {
        this.productId = id;
        this.productName = name;
        this.quantity = qty;
        this.price = price;
    }

    @Override
    public String toString() {
        return productId + ": " + productName + ", Qty: " + quantity + ", Price: " + price;
    }
}

class InventoryManagementSystem {
    private HashMap<String, Product> inventory = new HashMap<>();

    public void addProduct(Product p) {
        inventory.put(p.productId, p);
        System.out.println("Product added: " + p);
    }

    public void updateProduct(String id, int qty, double price) {
        Product p = inventory.get(id);
        if (p != null) {
            p.quantity = qty;
            p.price = price;
            System.out.println("Product updated: " + p);
        } else {
```

```
36              System.out.println("Product not found.");
37          }
38      }
39
40      public void deleteProduct(String id) {
41          if (inventory.remove(id) != null) {
42              System.out.println("Product with ID " + id + " deleted.");
43          } else {
44              System.out.println("Product not found.");
45          }
46      }
47
48      public void displayInventory() {
49          System.out.println("Current Inventory:");
50          for (Product p : inventory.values()) {
51              System.out.println(p);
52          }
53      }
54  }
55
56  public class Main {
57      public static void main(String[] args) {
58          InventoryManagementSystem ims = new InventoryManagementSystem();
59          ims.addProduct(new Product("P001", "Laptop", 10, 75000));
60          ims.addProduct(new Product("P002", "Smartphone", 50, 20000));
61          ims.displayInventory();
62          ims.updateProduct("P001", 12, 74000);
63          ims.deleteProduct("P002");
64          ims.displayInventory();
65      }
66  }
```

**4. Analysis :**

The average time complexity for add, update, and delete operations in a C# Dictionary is O(1),meaning these operations are very fast because the dictionary uses hashing to access elements directly; however, in rare cases like resizing or many hash collisions, the complexity can degrade to O(n)

**Ways to Optimize These Operations:**

- Use Efficient Hash Functions
- Handle Collisions Properly
- Pre-size the Dictionary
- Avoid Frequent Resizing
- Use Immutable Keys