**Inventory Management System**

### **Why are Data Structures and Algorithms Essential in Handling Large Inventories?**

Efficient data structures and algorithms are crucial for inventory management because:

* ****Fast Access & Updates:**** Warehouses may have thousands of products. Efficient data structures allow quick search, addition, update, and deletion of products.
* ****Memory Management:**** Proper data structures help in optimal memory usage, preventing wastage and improving performance.
* ****Scalability:**** As inventory grows, operations should remain fast. Good algorithms ensure the system scales well.

### **Suitable Data Structures for Inventory Management**

* ****ArrayList:****Good for small inventories or when order matters, but searching, updating, or deleting by productId is slow (O(n)).
* ****HashMap:****Ideal for large inventories where quick access by productId is needed. Provides average O(1) time for add, update, and delete operations.

**Time Complexity**

**Used Data Structure :** ArrayList

**1. Add Operation:**

public void addProduct(Product product){

products.add(product);

System.out.println("Product Added Successfully");

}

**Time Complexity: O(1)**

Adding an element to the end of an `ArrayList` is generally O(1) because it involves placing the element in the next available slot.

**2. Update Operation:**

public void updateProduct(Product product) {

for(Product prod : products){

if(prod.getProductId()==product.getProductId()){

prod.setProductName(product.getProductName());

prod.setQuantity(product.getQuantity());

prod.setPrice(product.getPrice());

return;

}

}

System.out.println("No product found");

}

**Time Complexity: O(n)**

Searching for a product by its `productId` requires a linear search through the `ArrayList`. In the worst case, this operation has a time complexity of O(n).

**3. Delete Operation:**

public void removeProduct(int productId){

for(Product product : products){

if(product.getProductId() == productId){

products.remove(product);

return;

}

}

System.out.println("No Product Found");

}

**Time Complexity: O(n)**

Finding the product by `productId` requires a linear search, and removing an element involves shifting subsequent elements, which is also O(n) in the worst case.

**Optimizations**

To optimize these operations, we can use a `HashMap` where the key is `productId` and the value is the `Product` object. This will reduce the time complexity of most operations.

**Using HashMap**

**1. Add Operation**

public void addProduct(Product product) {

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

}

**Time Complexity: O(1)**

Inserting an element into a `HashMap` is O(1) on average.

**2. Update Operation**

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

Product product = productsMap.get(productId);

if (product != null) {

product.setProductName(productName);

product.setQuantity(quantity);

product.setPrice(price);

System.out.println("Product with ID : " + productId + " Updated");

} else {

System.out.println("Invalid Product ID : " + productId);

}

}

**Time Complexity: O(1)**

Retrieving an element from a `HashMap` is O(1) on average.

**3. Delete Operation**

public boolean deleteProduct(int productId) {

if (productsMap.remove(productId) != null) {

System.out.println("Product with ID : " + productId + " Removed");

return true;

} else {

System.out.println("Invalid Product ID : " + productId);

return false;

}

}

**Time Complexity: O(1)**

Removing an element from a `HashMap` is O(1) on average.