**Exercise 1 : Inventory Management System**

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

In an inventory management system, efficient data storage and retrieval are crucial because:

* Large inventories can involve thousands of products. Efficient data structures ensure that operations on this data remain manageable.
* Efficient algorithms are needed to quickly search for, update, and manage inventory data to meet operational requirements and customer demands.

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

The data structures that are suitable for this problem includes :

* **HashMap:** HashMap provides average-case time complexity of O(1) for insertion, deletion, and access operations. It is suitable for situations where we need to quickly look up products by their unique identifier.
* **ArrayList:** It is useful for maintaining an ordered collection of products. It allows O(1) access time but has O(n) time complexity for insertion and deletion operations if maintaining order is necessary.
* **TreeMap:** A sorted map implementation that provides O(log n) time complexity for insertion, deletion, and access operations. It is suitable if we need to maintain the order of products based on some criteria.

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

* **Add Operation:** In a Hash Map, the time complexity for adding a product is O(1) on average. This is because Hash Map uses hashing to quickly locate where to insert the new entry.
* **Update Operation:** The time complexity for updating a product is O(1) on average, as it involves locating the entry (O(1)) and then replacing it with the updated product (O(1)).
* **Delete Operation:** Deleting a product from a Hash Map also has an average time complexity of O(1), as it involves locating the entry and removing it.

**Discuss how you can optimize these operations.**

* Hash map offers average-case constant time operations, we also need to handle the collisions effectively by using good hashing functions.
* Additionally, if we want the ordered traversal, we can use TreeMap which provides O(log n) and maintains a sorted order.
* For large datasets with frequent insertions and deletions, we can use LinkedHashMap which maintains the insertion order with O(1) complexity.