Inventory Management System

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

Ans. Data structures and algorithms are crucial for efficient inventory management, especially with large inventories, because they provide the means to organize, store, access, and manipulate data in a way that optimizes performance and scalability. Without these tools, managing a large inventory would become slow, inefficient, and prone to errors.

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

Ans. One of the most suitable data structures for this is the hash map, which allows for constant-time (O(1)) average lookup, insertion, and deletion operations. This is especially useful for accessing product details using unique identifiers like product IDs. For maintaining sorted data such as listing products by name, price, or quantity binary search trees (BSTs) are suitable, offering logarithmic time complexity (O(log n)) for essential operations. When prioritizing certain actions, such as identifying products that need restocking, a priority queue or min-heap can be employed to retrieve items with the lowest stock efficiently. For linear storage and sequential operations, arrays or lists serve as simple yet effective structures, although they are less efficient for large datasets.

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

Ans. In our system using HashMap, adding a product has an average time complexity of O(1) and a worst-case of O(n), as it involves direct insertion with the product ID as the key. Updating a product also takes O(1) on average and O(n) in the worst case, since it requires a lookup by key followed by field updates. Deleting a product operates similarly, with an average time complexity of O(1) and a worst-case of O(n), as it removes the entry by its key. Displaying all products has a time complexity of O(n) because it iterates over all values to output product details. The worst-case scenario for a HashMap, which occurs when many hash collisions happen, is rare due to Java’s effective hashing mechanisms.

1. Discuss how you can optimize these operations?

Ans. We can optimise this further by using ConcurrentHashMap for multi-threaded access. We can use caching techniques like LRUCache for frequent access items and even integrate with a database for persistent storage.