**7\_Inventory Management System**

**a. Why are data structures and algorithms important for handling large inventories?**

* When we are managing a large inventory, thousands or even millions of products, things can get out of hand quickly if we are not using the right tools. This is where data structures and algorithms come in.
* They help us organize and access product data efficiently, so operations like adding, searching, updating, or deleting products do not become slow. For instance, if we need to look up a product by its ID, we do not want the system scanning through every item one by one — that would be too slow. Ideally, we want operations to happen instantly (O (1)) or in a few steps (O (log n)).
* Good data structures ensure our system stays fast and responsive, even as our inventory grows. Without them, performance drops, user experience suffers, and maintaining the system becomes a challenge.

**b. What data structures are suitable for this kind of system?**

* **Array List** - Great for keeping items in order, but searching by product ID takes time, not ideal for large inventories.
* **HashMap** - This is usually the best choice. It lets us quickly find, add, or remove products based on their unique ID in constant time (on average).
* **Tree Map** - Useful if we need products sorted by ID, though operations take a bit more time (logarithmic).
* **LinkedList** - Not a good fit here. It is slow for searching and accessing elements at random.

Considering we want fast lookup by productid (unique), **HashMap** is the best choice.

**c. Time Complexity of operations with HashMap -**

|  |  |  |
| --- | --- | --- |
| **Operation** | **Average Time Complexity** | **Reason** |
| Add | O(1) | HashMap inserts in constant average time. |
| Update | O(1) | Lookup + update in constant time. |
| Delete | O(1) | Removal by key in constant time. |
| Search | O(1) | Lookup by key is constant time on average. |

**How to optimize these operations -**

* Use **HashMap** for constant-time key-based access.
* Ensure good hashCode implementations for keys (here, productId as String works well).
* For concurrency, use ConcurrentHashMap to allow thread-safe operations.
* For persistence, consider integrating a database and indexing on productId.