1. Inventory Management System

• Why data structures and algorithms are essential in handling large inventories

- **Efficiency**: Efficient data storage and retrieval are crucial for managing large inventories. The right data structures and algorithms ensure that operations like adding, updating, and deleting products are performed quickly, even as the inventory grows.

- **Scalability**: As the inventory size increases, the performance of operations must remain consistent. Data structures and algorithms designed for efficiency and scalability help maintain performance.

- **Memory Management**: Proper data structures help in managing memory efficiently, ensuring that the system can handle large amounts of data without running into memory issues.

- **Search and Retrieval**: Efficient algorithms enable quick searching and retrieval of product information, which is essential for operations like checking stock levels or processing orders.

Types of Data Structures suitable for this problem

ArrayList:

- **Advantages**: Easy to use, dynamic resizing, good for scenarios where read operations are more frequent than writes.
- **Disadvantages**: Poor performance for add/delete operations in the middle of the list due to shifting elements.

HashMap:

- **Advantages**: Provides average O(1) time complexity for add, update, and delete operations. Excellent for scenarios where quick lookup by product ID is needed.
- **Disadvantages**: Requires more memory due to hashing, less efficient for ordered data retrieval.

LinkedList:

- **Advantages**: Good for frequent insertions and deletions, especially in the middle of the list.
- **Disadvantages**: Poor performance for search operations (O(n) time complexity).

Analysis of the code performance

1. Add Operation:

- **ArrayList**: O(1) when adding to the end, O(n) when adding at a specific position due to shifting.
- **HashMap**: O(1) on average, O(n) in the worst case due to collisions.
- **LinkedList**: O(1) when adding to the beginning or end, O(n) when adding in the middle.

2. Update Operation:

- **ArrayList**: O(n) for searching, followed by O(1) for updating.
- **HashMap**: O(1) on average for direct access and update.
- **LinkedList**: O(n) for searching, followed by O(1) for updating

3. Delete Operation:

- **ArrayList**: O(n) due to searching and shifting elements.
- **HashMap**: O(1) on average, O(n) in the worst case.
- **LinkedList**: O(n) for searching, followed by O(1) for deletion.

Optimisation

1. **Use of Hashing**: For quick lookups, updates, and deletions, a HashMap is highly efficient due to its average O(1) time complexity.

2. **Combining Structures**: For different parts of the inventory system, use a combination of data structures. For instance, use a HashMap for fast lookups and a LinkedList for maintaining order of insertion.

3. **Lazy Deletion**: In some cases, lazy deletion (marking items as deleted without actually removing them) can improve performance by deferring the actual deletion to a less critical time.