**COGNIZANT DIGITAL NURTURE – 3.0**

**JAVA FSE**

**WEEK – 1 EXERCISES**

**DATA STRUCTURES AND ALGORITHMS**

**Exercise 1: Inventory Management System**

**Step 1: Understand the Problem:**

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

For the following reasons, managing huge inventories requires the use of data structures and algorithms:  
  
**Efficiency:** Managing huge inventories requires effective data storage, retrieval, and manipulation, all of which are made possible by appropriate data structures.  
**Scalability:** As inventory grows, scalable algorithms make sure that operations stay quick and responsive.  
**Maintainability:** Selecting appropriate data structures facilitates system expansion and maintenance.

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

Suitable data structures for an inventory management system include the following ones:  
**ArrayList:** Offers dynamic array functionality at a slower rate for insertions and deletions (O(n)) but with quick access speeds (O(1) for get operations).  
**HashMap:** Provides insertion, deletion, and lookup operations with average-case constant time complexity (O(1)), which makes it extremely effective for handling key-value pairs in which the product ID serves as the key.  
**LinkedList:** Access times are slower (O(n)) but useful for quick insertions and deletions (O(1) if the position is known).

**Step 2: Setup**

***Refer Program Files***

**Step 3: Implementation**

***Refer Program Files***

**Step 4: Analysis:**

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

* **Time Complexity:**  
  **Add Product:** The average-case time complexity of adding a product to a hash map is O (1).  
  **Update Product:** The lookup and insertion processes required to update a product have an O (1) average-case time complexity.  
  **Delete Product:** The average-case time complexity of removing a product from a hash map is O (1).

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

* **Optimization:**

**Load Factor Management:** To reduce rehashing, modify the load factor and hash map's starting capacity.  
**Batch Operations:** Batching updates can cut down on the number of rehashing operations required if several updates need to be made.  
**Concurrency:** To handle concurrent access in a multi-threaded system, think about utilising ConcurrentHashMap.