Inventory management system

**Step 1: Understand the Problem**

Why Data Structures and Algorithms are Essential in Handling Large Inventories:

Data structures and algorithms play a pivotal role in handling large inventories due to several reasons:

**Efficiency**: Proper data structures ensure efficient storage and retrieval of inventory data. This is crucial in a warehouse where quick access to inventory information can impact operational efficiency.

**Scalability**: As the size of the inventory grows, the performance of data retrieval, addition, and modification should remain optimal. Efficient algorithms and data structures allow the system to scale without significant degradation in performance.

**Complexity Management:** Complex inventory operations, such as searching for products, sorting inventory, and managing stock levels, can be simplified with the right data structures and algorithms.

**Data Integrity**: Proper data structures ensure the integrity and consistency of the inventory data, which is essential for accurate inventory management

**STEP 2:Types of Data Structures Suitable for This Problem:**

1. **ArrayList**: Useful for storing products in a sequential manner. It allows for quick access and iteration but has limitations in terms of insertion and deletion times.
2. **HashMap**: Ideal for quick lookups, insertions, and deletions based on keys (e.g., productId). This data structure provides average O(1) time complexity for these operations, making it highly efficient for inventory management.
3. **LinkedList**: Can be used if frequent insertions and deletions are needed, but access time is O(n), which can be a drawback for large inventories.
4. **TreeMap**: Offers sorted storage of products with O(log n) complexity for insertions, deletions, and lookups. Useful if order-based operations are frequently needed

**Step 4: Analysis**

**Time Complexity Analysis:**

1. **Add Product**:
   * Time Complexity: O(1)
   * Explanation: Adding a product involves inserting a key-value pair into the HashMap, which on average takes constant time.
2. **Update Product**:
   * Time Complexity: O(1)
   * Explanation: Updating a product involves checking for the existence of the key and then updating the value, both of which take constant time on average.
3. **Delete Product**:
   * Time Complexity: O(1)
   * Explanation: Deleting a product involves removing a key-value pair from the HashMap, which on average takes constant time.