**Step 1: Understand the Problem**

**Why Data Structures and Algorithms Are Essential:**

Data structures and algorithms are critical for efficiently managing and accessing data. In an inventory management system, you'll need to:

* **Store Products Efficiently**: Handle a potentially large number of products and ensure that operations like search, update, and deletion are quick.
* **Optimize Operations**: Ensure that operations like adding, updating, and deleting products are efficient, particularly as the inventory grows.
* **Ensure Scalability**: Choose data structures that allow the system to scale effectively with increasing amounts of data.

**Suitable Data Structures:**

1. **ArrayList**:
   * **Pros**: Provides fast access to elements by index, suitable for smaller inventories or when order matters.
   * **Cons**: Adding or removing elements can be costly (O(n)) as it might require shifting elements.
2. **HashMap**:
   * **Pros**: Provides average-case O(1) time complexity for add, update, and delete operations due to hashing.
   * **Cons**: Requires extra memory for hashing and may suffer from collisions.
3. **TreeMap**:
   * **Pros**: Provides sorted order and O(log n) time complexity for add, update, and delete operations.
   * **Cons**: Slower than HashMap for basic operations but useful if you need sorted data.

**Chosen Data Structure**: For most inventory systems, a HashMap is suitable due to its efficiency in handling add, update, and delete operations with average-case constant time complexity.

**Step 4: Analysis**

**Time Complexity Analysis:**

1. **Add Operation** (addProduct):
   * **HashMap**: Average-case O(1) time complexity due to hashing.
   * **ArrayList**: O(n) due to the need to check for duplicates and possible shifting.
2. **Update Operation** (updateProduct):
   * **HashMap**: Average-case O(1) time complexity for both checking existence and updating the value.
   * **ArrayList**: O(n) for searching and updating the element.
3. **Delete Operation** (deleteProduct):
   * **HashMap**: Average-case O(1) time complexity for both checking existence and removing the element.
   * **ArrayList**: O(n) for searching and removing the element.

**Optimizations:**

1. **HashMap**: Use for efficient add, update, and delete operations with constant time complexity.
2. **Handling Collisions**: If using HashMap, ensure good hash function implementation to minimize collisions.
3. **Memory Considerations**: Be aware of memory usage with HashMap and adjust load factor if necessary to balance between time complexity and memory usage.