**Exercise 1: Inventory Management System**

**Scenario:**

You are developing an inventory management system for a warehouse. Efficient data storage and retrieval are crucial.

**Steps 1 : Understand the Problem:**

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

Data structures and algorithms are important in managing huge inventories for numerous reasons:

* Efficient Storage and Retrieval: Proper data structures enable rapid access to inventory items, minimizing the time required to locate specific products. This is critical in huge warehouses where inventory can be thousands of things.
* Scalability: As the inventory grows, the system must manage the additional data efficiently. Choosing the correct data structures ensures that the system scales well while maintaining performance.
* Memory Efficiency: Efficient data structures make the best use of memory, which is critical when working with large datasets.
* Data Integrity and Operations: Operations such as adding, updating, and deleting products should be completed efficiently while maintaining data integrity. Suitable algorithms ensure that these procedures are completed accurately and rapidly.
  + **Discuss the types of data structures suitable for this problem.**
  + **ArrayList:** Suitable for maintaining an ordered list of products but not efficient for frequent insertions or deletions in large datasets.
  + **HashMap:** Provides average constant time complexity (O(1)) for insertion, deletion, and lookup operations, making it ideal for managing large inventories where quick access to items is necessary.
  + **LinkedList:** Useful for frequent insertions and deletions but less efficient for lookups.

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

**Time Complexity Analysis:**

* **Add Product (addProduct):** O(1) on average, as inserting into a HashMap is generally constant time.
* **Update Product (updateProduct):** O(1) as it involves a single lookup and update operation.
* **Delete Product (deleteProduct):** O(1) for a single lookup and deletion.
* **Get Product (getProduct):** O(1) for lookup.
  + **Discuss how you can optimize these operations.**
* **Handling Collisions**: HashMap uses a hash function to determine the index for storing key-value pairs. If two keys hash to the same index, a collision occurs. Java's HashMap handles collisions using separate chaining (linked lists) or open addressing (probing).
* **Scaling**: As the number of products increases, the HashMap automatically resizes (doubles its capacity) to maintain efficient performance. This ensures that the load factor (ratio of number of elements to the capacity) stays below a certain threshold (default is 0.75), maintaining O(1) average time complexity for operations.