# **Analysis:**

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

In the ‘**InventoryManagement**’ class, we use a ‘**HashMap**’ to store products. The HashMap provides efficient operations for insertion, update, and deletion. Here is the analysis of the time complexity for each operation:

Time Complexity Analysis

* Add Product
  + ‘containsKey’: O(1) on average because hash table lookups are O(1).
  + ‘Put’: O(1) on average for hash table insertions.
  + Overall Complexity: O(1) on average.
* Update Product
  + ‘containsKey’: O(1) on average.
  + ‘Put’: O(1) on average.
  + Overall Complexity: O(1) on average.
* Delete Product
  + ‘containsKey’: O(1) on average.
  + ‘Remove’: O(1) on average.
  + Overall Complexity: O(1) on average.

Using a ‘**HashMap**’, all operations (add, update, delete) have an average time complexity of O(1). This efficiency makes ‘**HashMap**’ a suitable choice for managing large inventories where quick access and modifications are necessary.

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

To optimize the HashMap operations in the inventory management system, consider the following strategies:

Optimizations

1. Initial Capacity and Load Factor:
   * Set an initial capacity close to the expected number of entries to reduce costly resizes.
   * Adjust the load factor to balance time complexity and memory usage.
2. Efficient Hashing:
   * Ensure the ‘**hashCode’** method for ‘**productId**’ provides a well-distributed hash to minimize collisions.
3. Concurrency Handling:
   * Use ‘**ConcurrentHashMap’** for thread-safe operations in a multi-threaded environment
4. Batch Operations:

* Implement batch add or update operations to improve efficiency