## Understand the Problem

### Why data structures and algorithms are essential in handling large inventories:

In a warehouse inventory system, we deal with a lot of product details like ID, name, quantity, and price. When the number of products increases, managing them efficiently becomes very important.  
  
Data structures help us to store the data in a proper format so that we can easily access, update, or delete it whenever needed. Without using a suitable data structure, it will take a long time to search for a product or change its details. For example, if the data is not organized properly, we might have to check each product one by one.  
  
Algorithms help us in performing these operations in a better and faster way. They define how the data should be processed. Together, data structures and algorithms save time, reduce complexity, and make the system fast and smooth, especially when the data is huge.

### Types of data structures suitable for this problem:

**- ArrayList:**  
 Simple and easy to use. Suitable if we just want to display all products or store them in order. But not efficient for searching by ID, because we have to go through each product one by one.  
  
**- HashMap:**  
 This is very useful when each product has a unique ID. We can use the product ID as the key and the product object as the value. Searching, updating, and deleting can be done in constant time on average. So, it is best suited for fast operations.

**- LinkedList:**  
 Useful when we need to frequently insert or remove elements in the middle. But searching is slow, so not the best for this case.  
  
**- TreeMap:**  
 If we want to maintain the products in a sorted way (for example, sorted by ID), TreeMap is a good option. But it takes slightly more time than HashMap.

## Analysis

### Time Complexity:

- Add Product:  
 Using HashMap, adding a product using .put() takes O(1) average time.  
  
- Update Product:  
 Since we can access the product directly by using its ID (key), updating the product details also takes O(1) time on average.  
  
- Delete Product:  
 Removing an item from the HashMap using .remove() also takes O(1) time on average.

### How to Optimize These Operations:

* Even though HashMap is already fast, we can make more efficient by:  
  - Avoid using duplicate keys..  
  - Handle null values properly to avoid errors.  
  - Keep the load factor of the HashMap balanced