# Exercise 1: Inventory Management System

## 1. Understand the Problem

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

In managing a complex logistics operation, where tracking and updating large amounts of data is crucial, the efficiency of data storage and retrieval becomes imperative due to:

* **Scalability:**The system needs to handle growing volumes of data efficiently as the logistics operations expand.
* **Performance:** Operations such as querying inventory levels, updating stock information, and removing outdated records must be optimised for quick responses.
* **Resource Management:** Efficient memory use and processing are crucial to maintain system performance and responsiveness.

**Types of Data Structures Suitable:**

**1. ArrayList (or List in Java):**

Dynamic resizing and easy element access via indexing.

Insertion and deletion operations can be costly as they may require shifting elements.

**2. HashMap (or Dictionary in Python):**

Provides average O(1) time complexity for insertions, deletions, and lookups due to its hashing mechanism.

No inherent ordering of elements; requires handling of potential hash collisions.

**3. Binary Search Tree (BST) or Balanced Trees:**

Maintains sorted order with O(log n) time complexity for search, insertion, and deletion operations.

More complex to implement and manage compared to HashMap.

**4. Priority Queue (if inventory needs management by urgency):**

Efficiently manages items based on priority levels.

Implementation complexity depends on the chosen priority queue algorithm.

**5. LinkedList:**

Allows efficient insertion and deletion operations.

Provides linear time complexity for element access due to sequential traversal.

**2. Setup**

**Create a New Project:**

**1. Initialize Project:**

- Use an IDE such as IntelliJ IDEA, Eclipse, or Visual Studio Code.

- Create a new project directory and set up a version control system (e.g., Git).

**2. Project Structure:**

- Set up directories for source code, test cases, and documentation.

**3. Implementation**

**Constructor:**

* Initialises a new LogisticsSystem instance with an empty inventory using a HashMap.

**addItem(Item item)**

* Adds a new item to the inventory.
* The item is added to the HashMap with its item ID as the key.

**updateItem(String itemId, Item newItem)**

* Updates an existing item in the inventory.
* Replaces the item associated with the given item ID with a new item if the item ID exists in the inventory.

**deleteItem(String itemId)**

* Removes an item from the inventory.
* Deletes the item associated with the specified item ID from the HashMap.

**getItem(String itemId)**

* Retrieves an item from the inventory.
* Returns the item associated with the given item ID from the HashMap.

**4. Analysis**

**Time Complexity:**

**Add Operation:**

- **HashMap:** O(1) average time complexity. Adding a new item involves hashing the key and storing it in the map.

- **ArrayList:** O(1) if adding at the end; O(n) if inserting at a specific position.

**Update Operation:**

**- HashMap:**  O(1) average time complexity. Updating an item involves replacing the value associated with a key.

**- ArrayList:** O(n) due to searching for the item first, then updating it.

**- Delete Operation:**

**- HashMap:**  O(1) average time complexity. Removing an item involves hashing and deleting the key-value pair.

**- ArrayList:** O(n) due to searching for the item, then removing it and shifting elements.

**Optimizations:**

**- HashMap:**  Ensure an effective hash function and manage collisions to sustain O(1) performance.

**- Memory Management:**  Explore space-efficient data structures or data compression techniques if memory usage becomes critical.

- **Concurrency:** Employ concurrent data structures or synchronize access if the system is accessed or modified by multiple threads.