**Inventory Management System - Analysis**

* **Time Complexity Analysis :**

1. **Add Product**
   * **Operation**: Adding a product to the inventory.
   * **Time Complexity**: O(1) (average case)
   * **Explanation**: In a HashMap, the put method has an average time complexity of O(1) because it computes the hash code of the key and places the value in the appropriate bucket. However, in the worst-case scenario, where there are many hash collisions, the time complexity could degrade to O(n), where n is the number of entries in the map.
2. **Update Product**
   * **Operation**: Updating an existing product in the inventory.
   * **Time Complexity**: O(1) (average case)
   * **Explanation**: Updating a product in a HashMap is similar to adding a product. The put method will overwrite the existing value if the key already exists, and this operation has an average time complexity of O(1). As with adding a product, the worst-case scenario time complexity is O(n).
3. **Delete Product**
   * **Operation**: Deleting a product from the inventory.
   * **Time Complexity**: O(1) (average case)
   * **Explanation**: Deleting a product in a HashMap uses the remove method, which also has an average time complexity of O(1) as it relies on finding the bucket for the key and removing the entry. Again, the worst-case scenario is O(n) due to potential hash collisions.
4. **Get Product**
   * **Operation**: Retrieving a product from the inventory.
   * **Time Complexity**: O(1) (average case)
   * **Explanation**: Retrieving a product uses the get method of the HashMap, which has an average time complexity of O(1) because it quickly finds the bucket associated with the key. The worst-case scenario, due to hash collisions, is O(n).
5. **Display Inventory**
   * **Operation**: Displaying all products in the inventory.
   * **Time Complexity**: O(n)
   * **Explanation**: Displaying all products involves iterating over the values in the HashMap, which requires O(n) time, where n is the number of products in the inventory.

* **Optimization Strategies :**

1. **Reduce Hash Collisions**
   * Use a good hash function to evenly distribute keys.
2. **Monitor Load Factor and Rehash**
   * Resize the hash table when necessary to maintain O(1) operations.
3. **Batch Processing**
   * Implement batch updates and deletions to reduce overhead.
4. **Cache Frequently Accessed Products**
   * Improve read performance by caching commonly used products.