**1. Inventory Management with HashMap**

* **Time Complexity:** Adding, updating, and deleting products in a HashMap typically takes constant time on average, thanks to its efficient hashing mechanism.
* **Optimizations:** To maintain high performance, consider factors like initial HashMap size, load factor, and effective collision handling. Caching frequently accessed products can further boost retrieval speed.

**2. Linear vs. Binary Search**

* **Linear Search:** Examines each item sequentially. While simple, it's less efficient for large datasets.
* **Binary Search:** Significantly faster for large, sorted datasets due to its logarithmic time complexity. However, it requires the data to be sorted beforehand.
* **E-commerce Recommendation:** Binary search is often preferred for large product catalogs due to its speed, assuming products are sorted. Linear search might be suitable for smaller catalogs or unsorted data.

**3. Bubble Sort vs. Quick Sort**

* **Bubble Sort:** Inefficient for large datasets due to its quadratic time complexity in most cases.
* **Quick Sort:** Generally faster with an average time complexity of n log n. It's a preferred choice for its efficiency and in-place sorting nature.
* **Performance:** Quick Sort's divide-and-conquer approach often outperforms Bubble Sort in practical scenarios.

**4. Array Performance**

* **Time Complexity:** Adding, searching, traversing, and deleting elements in an array can vary based on factors like array size and element position. Generally, these operations have average time complexities of O(n).
* **Performance Considerations:** Resizing arrays can be costly due to data copying. Additionally, inserting or deleting elements often requires shifting subsequent elements.

**5. Linked List Performance**

* **Time Complexity:** Similar to arrays, linked list operations like adding, searching, traversing, and deleting typically have average time complexities of O(n).
* **Advantages:** Linked lists excel at dynamic resizing and efficient insertions/deletions compared to arrays.
* **Disadvantages:** Linked lists have higher memory overhead per element and lack direct access to elements.

**6. Linear vs. Binary Search Revisited**

* **Linear Search:** Suitable for small datasets or unsorted data where simplicity is prioritized over speed.
* **Binary Search:** Ideal for large, sorted datasets to achieve faster search times.

**7. Recursive Algorithm Analysis**

* **Time Complexity:** Recursive algorithms often have time complexities directly tied to the number of recursive calls. For instance, calculating Fibonacci numbers recursively results in exponential time complexity due to overlapping subproblems.
* **Space Complexity:** Recursive functions consume stack space for each function call, leading to space complexity proportional to the maximum recursion depth.
* **Optimization:** Memorization can significantly improve performance by storing intermediate results and avoiding redundant calculations.