### **Exercise 1: Inventory Management System – Analysis**

**Time Complexity:**

* addProduct: O(1) — HashMap insertion is constant time on average.
* updateProduct: O(1) — Updating a key-value pair in HashMap.
* deleteProduct: O(1) — Removing a key is constant time.
* displayAll: O(n) — Iterates over all values in the map.

**Optimization Discussion:** Using a HashMap<Integer, Product> provides optimal performance for operations based on product ID. If product listings require order preservation, LinkedHashMap could be used. For sorted display, TreeMap is an option but at a cost of O(log n) per operation.

### **Exercise 2: E-commerce Platform Search Function – Analysis**

**Time Complexity:**

* linearSearch: O(n) — Checks every element sequentially.
* binarySearch: O(log n) — Efficient divide-and-conquer method on sorted data.

**Best Use Case Discussion:** Use linearSearch for small, unsorted data or when insertion/deletion is frequent. Use binarySearch when data is static and sorted, as it offers significant performance benefits in larger datasets.

### **Exercise 3: Sorting Customer Orders – Analysis**

**Time Complexity:**

* bubbleSort: O(n^2) — Poor for large data; simple but inefficient.
* quickSort: Average: O(n log n), Worst: O(n^2) — Highly efficient with random pivots or optimized implementations.

**Why Quick Sort is Preferred:** Quick Sort is generally faster in practice than Bubble Sort due to lower constant factors and better cache usage. It is the standard for most real-world sorting.

### **Exercise 4: Employee Management System – Analysis**

**Time Complexity:**

* add: O(1) — Appends at the end.
* search: O(n) — Must scan through the array.
* delete: O(n) — Requires shifting elements.
* traverse: O(n) — Linear iteration.

**Limitations and Considerations:** Fixed-size arrays lack flexibility. Dynamic data structures (e.g., ArrayList) provide resizable capabilities. Linked lists are better if frequent insertions/deletions are needed.

### **Exercise 5: Task Management System – Analysis**

**Time Complexity:**

* addTask: O(1) — Insertion at head.
* search: O(n) — Traverses the list.
* delete: O(n) — May need to scan to find the target.
* traverse: O(n) — Linear scan.

**Advantages over Arrays:** Linked lists allow efficient dynamic memory usage and constant-time insertions/deletions without shifting elements. However, they have higher memory overhead due to pointer storage.

### **Exercise 6: Library Management System – Analysis**

**Time Complexity:**

* linearSearch: O(n)
* binarySearch: O(log n) — On sorted titles.

**Use Case Comparison:** Use linearSearch for small or unsorted data. For large, read-heavy datasets where data is sorted, binarySearch significantly outperforms linear search.

### **Exercise 7: Financial Forecasting – Analysis**

**Time Complexity:**

* Naive Recursion: O(2^n) — Exponential without memoization.
* Memoized Recursion / DP: O(n) — Each value computed once.

**Optimization:** Avoid redundant calculations using memoization or bottom-up dynamic programming. This reduces stack overflow risk and drastically improves performance for large inputs.