**Week 1: Data Structures and Algorithms**

**Exercise 1: Inventory Management System**

**Understanding the Problem:** Data structures and algorithms are essential for efficiently managing large inventories. They help in quickly adding, removing, and finding items, which keeps the system fast even with many products.

**Suitable Data Structures:**

* **ArrayList:** Good for dynamic resizing and quick access by index, but inserting or deleting items (other than at the end) can be slow.
* **HashMap:** Provides very fast operations for adding, updating, and deleting items on average because of its hashing mechanism.

**Analysis:**

* **HashMap:**
  + **Add:** O(1)
  + **Update:** O(1)
  + **Delete:** O(1)

**Optimization Discussion:**

* I've used HashMap for its speed in looking up and updating items. If maintaining order or frequently iterating is needed, using an ArrayList with proper indexing or a sorted structure might be better.

**Exercise 2: E-commerce Platform Search Function**

**Understanding Asymptotic Notation:**

* **Big O Notation:** Describes the maximum time an algorithm takes as the input size grows, helping compare different algorithms.

**Search Scenarios:**

* **Linear Search:** O(n) for all cases. Simple but slower for large datasets.
* **Binary Search:** O(log n) for sorted arrays. Faster than linear search but requires sorted data.

**Analysis:**

* **Linear Search:** Works on unsorted data but is slow for large datasets.
* **Binary Search:** Fast for sorted data, but requires pre-sorting.

**Suitability Discussion:**

* **Binary Search** is better for large datasets if the data is sorted.
* **Linear Search** is useful for smaller or unsorted datasets.

**Exercise 3: Sorting Customer Orders**

**Understanding Sorting Algorithms:**

* **Bubble Sort:** Simple but slow with O(n^2) time complexity.
* **Insertion Sort:** Better for small or nearly sorted data with O(n^2) time complexity.
* **Quick Sort:** Generally fast with O(n log n) on average, but can be slow in the worst case.
* **Merge Sort:** Consistently fast with O(n log n) in both worst and average cases.

**Analysis:**

* **Bubble Sort:** O(n^2) - slow for large datasets.
* **Quick Sort:** O(n log n) - faster and preferred for large datasets due to its average-case performance.

**Preference Discussion:**

* **Quick Sort** is preferred over Bubble Sort for its better performance on larger datasets.

**Exercise 4: Employee Management System**

**Understanding Array Representation:**

* **Arrays:** Provide fast access with O(1) time complexity but have a fixed size, which limits their flexibility.

**Analysis:**

* **Add:** O(1) if adding at the end; O(n) if inserting in the middle or resizing.
* **Search:** O(n) for unsorted, O(log n) if sorted with binary search.
* **Traverse:** O(n) - visiting each element.
* **Delete:** O(n) - shifting elements.

**Limitations and Use Cases:**

* Arrays are fixed in size and not ideal for frequent insertions or deletions. For more dynamic data, linked lists or dynamic arrays might be better.

**Exercise 5: Task Management System**

**Understanding Linked Lists:**

* **Singly Linked List:** Nodes point to the next node. Good for insertion and deletion but slower for access.
* **Doubly Linked List:** Nodes point to both the next and previous nodes, allowing easier traversal and deletion.

**Analysis:**

* **Add:** O(1) for adding at the start or end; O(n) for specific positions.
* **Search:** O(n) - needs to go through each node.
* **Traverse:** O(n) - visiting each node.
* **Delete:** O(1) if the node is known; O(n) if searching for the node.

**Advantages Discussion:**

* Linked lists are better for data where frequent changes occur. They are more flexible than arrays but have extra overhead due to additional pointers.

**Exercise 6: Library Management System**

**Understanding Search Algorithms:**

* **Linear Search:** Checks each item one by one. Simple but slow for large datasets (O(n)).
* **Binary Search:** Efficient for sorted data, checking the middle and reducing the search space by half (O(log n)).

**Analysis:**

* **Linear Search:** O(n) for all cases.
* **Binary Search:** O(log n) for sorted data.

**Usage Discussion:**

* **Binary Search** is better for large, sorted datasets due to its efficiency.
* **Linear Search** is good for smaller or unsorted datasets.

**Exercise 7: Financial Forecasting**

**Understanding Recursive Algorithms:**

* **Recursion:** A method where a function calls itself to solve smaller parts of the problem, making complex problems simpler.

**Analysis:**

* **Recursion:** Can be slow due to repeated calculations, often leading to high time complexity (e.g., basic Fibonacci sequence).
* **Optimization:** Use techniques like Memoization or Dynamic Programming to save results and reduce time complexity.