**JAVA-FSE ALOGIRTHMS\_DATA STRUCTURE**

**Exercise 1: Inventory Management System**

**Step 1: Understand the Problem**

Why are Data Structures and Algorithms Important?

* Warehouses deal with thousands of products — efficient search, update, and deletion are crucial.
* The right data structure ensures fast operations, better memory usage, and easier scalability.

**Suitable Data Structures:**

| **Data Structure** | **Use Case** | **Pros** | **Cons** |
| --- | --- | --- | --- |
| ArrayList | Simple list, suitable for small inventory | Easy to implement | Slow for search/update |
| HashMap | Fast access using product ID as key | O(1) add/update/delete | Slightly more memory usage |
| TreeMap | Sorted access | Maintains sorted keys | O(log n) operations |

**Step 2: Setup**

Create a Java project named InventoryManagementSystem.

**Step 3: Implementation**

It has been done successfully.

**Step 4: Time Complexity Analysis**

| **Operation** | **HashMap Complexity** | **Explanation** |
| --- | --- | --- |
| Add Product | O(1) | Direct insertion using put() |
| Update Product | O(1) | Lookup and replace using get() |
| Delete Product | O(1) | Removal using remove() |
| List All | O(n) | Traverse all entries once |

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

**1. Understanding Asymptotic Notation:**

Big O notation describes the performance or complexity of an algorithm.

- Best case: Optimal condition (e.g., first match in linear search).

- Average case: Expected performance over typical inputs.

- Worst case: Most time-consuming scenario (e.g., item not found).

**2. Setup:**

Product class includes productId, productName, and category.

**3. Implementation:**

- Linear Search: Iterates through unsorted array.

- Binary Search: Requires sorted array of products by name or ID.

**4. Analysis:**

- Linear Search: O(n)

- Binary Search: O(log n)

Binary search is preferable when data is sorted and large in size.

**Exercise 3: Sorting Customer Orders**

**1. Understanding Sorting Algorithms:**

- Bubble Sort: Simple but slow (O(n²)).

- Quick Sort: Efficient with average complexity O(n log n), faster in most cases.

**2. Setup:**

An Order class is created with orderId, customerName, and totalPrice.

**3. Implementation:**

Two sorting functions are implemented:

- bubbleSort(Order[] orders)

- quickSort(Order[] orders, int low, int high)

**4. Analysis:**

Quick Sort is generally preferred over Bubble Sort for better performance, especially with larger data sets.

**Exercise 4: Employee Management System**

**1. Understand Array Representation**

How Arrays Work in Memory

* Arrays are stored in contiguous memory locations.
* Direct indexing (array[i]) gives O(1) access.
* They are fixed in size, so you need to define capacity in advance.

**Advantages of Arrays**

* Fast random access (O(1)).
* Simple to use and efficient for **small, fixed-size collections**.

**2, 3 & 4: Java Code with All Operations**

| **Operation** | **Time Complexity** | **Reason** |
| --- | --- | --- |
| Add Employee | O(1) | Add at the end |
| Search | O(n) | Linear scan |
| Traverse | O(n) | Visit all |
| Delete | O(n) | Find and shift elements |

**Exercise 5: Task Management System**

**1. Understanding Linked Lists:**

Linked lists allow dynamic memory allocation and ease of insertion and deletion compared to arrays. Types:

- Singly Linked List: Nodes point to the next node.

- Doubly Linked List: Nodes have both previous and next references.

**2. Setup**:

The Task class contains taskId, taskName, and status.

**3. Implementation:**

A singly linked list is used with methods to:

- Add tasks at the beginning or end

- Search by taskId

- Traverse all tasks

- Delete a task

**4. Analysis:**

- Add/Delete/Search: O(n) (O(1) if done at head)

Linked lists are better suited for dynamic lists with frequent changes.

**Exercise 6: Library Management System**

**1. Understanding Search Algorithms:**

- Linear Search: Suitable for small or unsorted datasets.

- Binary Search: Efficient for large, sorted data.

**2. Setup:**

A Book class is created with bookId, title, and author.

**3. Implementation:**

- Linear search loops through the array.

- Binary search applies divide-and-conquer on a sorted list.

**4. Analysis:**

- Linear: O(n)

- Binary: O(log n)

Binary search is more efficient, but only applicable to sorted data.

**Exercise 7: Financial Forecasting**

**1. Understanding Recursive Algorithms:**

Recursion solves problems by breaking them into smaller subproblems. It helps express logic in a concise way but can be inefficient without optimization.

**2. Setup:**

A recursive method is defined to calculate future value using:

futureValue = presentValue \* (1 + rate)^years

**3. Implementation:**

A recursive method calculates value over time.

**4. Analysis:**

- Time complexity: O(n)

- Optimizable using memoization or iteration to avoid stack overflow.