**Exercise 4: Employee Management System**

**Scenario:**

You are developing an employee management system for a company. Efficiently managing employee records is crucial.

**Steps:**

1. **Understand Array Representation:**
   * Explain how arrays are represented in memory and their advantages.

**Array Representation in Memory**:

* **Contiguous Memory Allocation**: Arrays are stored in consecutive memory locations, with each element placed next to the previous one.
* **Indexing**: Arrays provide direct access to elements using their index, starting from 0.
* **Fixed Size**: Arrays have a fixed size determined at creation and cannot be resized at runtime.

**Advantages of Arrays**:

* **Fast Access**: Accessing an element by its index takes constant time, O(1).
* **Memory Efficiency**: Arrays have low overhead and require no additional storage for pointers or metadata.
* **Predictability**: The predictable memory layout can lead to better cache performance.

1. **Setup:**
   * Create a class Employee with attributes like **employeeId**, **name**, **position**, and **salary**.
2. **Implementation:**
   * Use an array to store employee records.
   * Implement methods to **add**, **search**, **traverse**, and **delete** employees in the array.
3. **Analysis:**
   * Analyze the time complexity of each operation (add, search, traverse, delete).
   * Discuss the limitations of arrays and when to use them.

**Time Complexity of Each Operation**:

* **Add Employee**: O(1) - Adding is constant time if space is available.
* **Search Employee**: O(n) - Requires a linear scan through the array.
* **Traverse Employees**: O(n) - Involves visiting each element.
* **Delete Employee**: O(n) - Finding the employee is O(n), and shifting elements is O(n).

**Limitations of Arrays and When to Use Them**:

* **Fixed Size**: Arrays cannot be resized after creation, which can lead to wasted space or insufficient capacity.
* **Insertion and Deletion**: These operations can be inefficient (O(n)), especially if done at the beginning or middle of the array due to the need for shifting elements.
* **When to Use**: Arrays are best for datasets with a known, stable size, requiring fast access by index, and where memory efficiency is important.