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

**Solutions:**

1. **Understanding:**

Arrays are a collection of elements stored in contiguous memory locations. The elements are indexed, and each element can be accessed in constant time using its index, making arrays very efficient for read operations.

**Advantages of Arrays**:

1. **Direct Access**: Accessing an element by its index is very fast (O(1) time complexity).
2. **Efficient Memory Usage**: Arrays use a fixed amount of memory for a given size, making them memory-efficient for static collections of data.
3. **Cache-Friendly**: Due to contiguous memory allocation, arrays make better use of CPU caches, leading to faster access times.
4. **Analysis:**

#### Time Complexity of Each Operation

1. **Add Operation**: O(1) (when adding at the end of the array)
2. **Search Operation**: O(n) (linear search)
3. **Traverse Operation**: O(n)
4. **Delete Operation**: O(n) (finding the element and shifting the subsequent elements)

#### Limitations of Arrays and When to Use Them

**Limitations of Arrays**:

1. **Fixed Size**: Arrays have a fixed size, determined at creation. This makes them inflexible for dynamic data sizes.
2. **Insertion and Deletion**: Inserting or deleting elements can be inefficient, especially in the middle of the array, as it requires shifting elements.
3. **Memory Allocation**: Arrays require a contiguous block of memory, which can be problematic for very large arrays.

**When to Use Arrays**:

1. **Static Collections**: When the number of elements is known and fixed.
2. **Read-Heavy Applications**: When frequent access to elements by index is required.
3. **Memory Efficiency**: When memory overhead should be minimal.