**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.**

Arrays are a fundamental data structure in many programming languages, including Java. They are represented in memory as a contiguous block of elements, where each element can be accessed directly via its index. The primary advantages of arrays include:

* **Constant-Time Access**: Accessing an element by its index is O(1) because the memory address of any element can be calculated directly.
* **Efficient Use of Space**: Arrays use a fixed amount of space and have minimal overhead.
* **Cache-Friendly**: Since elements are stored in contiguous memory locations, arrays benefit from spatial locality, making them cache-friendly and improving 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).**

**Time Complexity Analysis:**

* **Add**: O(1) - Adding an element to the end of the array is a constant-time operation.
* **Search**: O(n) - In the worst case, you might need to check each element in the array.
* **Traverse**: O(n) - You need to visit each element in the array.
* **Delete**: O(n) - You need to find the element (O(n)) and then shift elements to fill the gap (O(1)).
  + **Discuss the limitations of arrays and when to use them.**

**Limitations of Arrays:**

* **Fixed Size**: Arrays have a fixed size, and resizing them involves creating a new array and copying elements, which is costly.
* **Inefficient Deletion**: Deleting an element involves shifting elements, which can be expensive for large arrays.
* **Inefficient Insertion**: Inserting an element at a specific position requires shifting elements to make space, which is O(n).

**When to Use Arrays:**

* When you need fast access to elements using an index.
* When the number of elements is known in advance and won't change frequently.
* When memory efficiency and cache performance are important.