**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 contiguous blocks of memory where elements of the same type are stored sequentially. The memory address of each element is calculated based on the starting address and the index of the element. This allows for:

* **Constant Time Access**: O(1) time complexity for accessing any element by its index.
* **Memory Efficiency**: Minimal overhead because the elements are stored sequentially.
* **Predictable Performance**: Fixed size means the memory allocation is static, making performance predictable.

**Advantages of Arrays**

1. **Fast Access**: Direct access to elements using their index.
2. **Memory Efficiency**: Lower memory overhead compared to dynamic data structures.
3. **Predictable Iteration**: Easy to iterate through elements using loops.
4. **Setup:**

A class Employee is created with attributes like employeeId, name, position, and salary. The program name is EmployeeManagementSystem.

1. **Implementation:**

The implementation code is provided in the wordpad.

1. **Analysis:**
2. **Analyze the time complexity of each operation (add, search, traverse, delete).**

Time Complexity of Each Operation

* Add Operation:
  + Time Complexity: O(1) - Adding an element at the end of the array is a constant time operation, assuming there is space available.
* Search Operation:
  + Time Complexity: O(n) - In the worst case, you might need to search through all elements.
* Traverse Operation:
  + Time Complexity: O(n) - Traversing through all elements in the array is a linear time operation.
* Delete Operation:
  + Time Complexity: O(n) - In the worst case, you might need to shift all elements after the deleted element.

1. **Discuss the limitations of arrays and when to use them.**

**Limitations of Arrays**

**Fixed Size**: Once an array is created, its size cannot be changed. This can lead to either wasted space or the need to create a new larger array and copy the elements.

**Insertion and Deletion**: Inserting or deleting elements requires shifting other elements, which can be inefficient for large arrays.

**Contiguous Memory Allocation**: Arrays require a block of contiguous memory, which might not be available, leading to allocation failures for large arrays.

**When to Use Arrays**

**When Size is Known**: Arrays are suitable when the number of elements is known and unlikely to change.

**Fast Access Required**: Arrays provide constant-time access to elements using their index.

**Simple Data Structures**: Arrays are straightforward to implement and understand, making them ideal for simple data structures.

For more dynamic scenarios where the number of elements can vary, other data structures like ArrayLists, LinkedLists, or more advanced collections might be more suitable.