**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.

Solution:

Array Representation in Memory

* Memory Layout: Arrays are stored in contiguous memory locations. This means that each element is placed in a sequential block of memory, which allows for efficient access and iteration.
* Advantages:
  + Index-Based Access: Elements can be accessed in constant time O(1)O(1)O(1) using their index.
  + Efficient Iteration: Since elements are contiguous, iterating through the array is very efficient.
  + Memory Efficiency: Arrays have a fixed size, and their memory is allocated once, which avoids the overhead of dynamic resizing.

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.

Solution:

Setup and Implementation part is in attached java file.

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

Solution:

Time Complexity of Operations

1. Add Employee: O(1) (assuming there is space in the array; otherwise, it's O(n) if resizing is needed).
2. Search Employee by ID: O(n) (linear search through the array).
3. Traverse Employees: O(n) (iterating through the array).
4. Delete Employee by ID: O(n) (linear search plus shifting elements).
   * Discuss the limitations of arrays and when to use them.

Solution:

Limitations of Arrays

* Fixed Size: Arrays have a fixed size, so if you need to add more employees than initially allocated, you will need to resize or use a more flexible data structure.
* Inefficient Insertions/Deletions: Adding or deleting elements, especially in the middle of the array, requires shifting elements, which can be inefficient.

When to Use Arrays

* When the number of elements is known and fixed: Arrays are useful when the size of the dataset is known beforehand and does not change often.
* When constant-time access is required: Arrays provide efficient access to elements via indices.