**Exercise 4: Employee Management System**

**Scenario**:

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

Step 1: Understand Array Representation

Arrays in Memory

* Concept: Arrays are contiguous blocks of memory where each element is of the same type and can be accessed using an index. The memory address of an element can be calculated using its index and the size of the element type.
* Advantages:
  + Constant Time Access: Accessing any element in an array is O(1) because the memory address can be directly calculated.
  + Memory Efficiency: Arrays do not have the overhead of additional references like linked lists, making them more memory efficient for simple data structures.
  + Cache Performance: Due to their contiguous memory layout, arrays benefit from spatial locality and are cache-friendly.

Step 2: Setup

Create a Class Employee: Define an Employee class with attributes employeeId, name, position, and salary.

Step 3: Implementation

Use an Array to Store Employee Records

Define Management class that uses an array to store and manage employees.

Step 4: Analysis

Time Complexity

* Add Employee: O(1)- Adding an employee at the end of the array takes constant time if there is space available.
* Search Employee: O(n) - In the worst case, we might need to check every element in the array.
* Delete Employee: O(n) - In the worst case, we might need to shift all elements after the deleted element.
* Traverse Employees: O(n) - We need to visit each element in the array once.

Limitations of Arrays and When to Use Them

* Fixed Size: Arrays have a fixed size, meaning once the array is full, you cannot add more elements without creating a new larger array and copying elements over. This can be inefficient for dynamic datasets.
* Insertion and Deletion: Inserting or deleting elements (other than at the end) requires shifting elements, which can be costly (O(n)O(n)O(n) time complexity).
* Memory Allocation: Arrays require contiguous memory allocation, which can be a limitation in systems with fragmented memory.

When to Use Arrays:

* Static Data: When the size of the dataset is known and doesn't change frequently.
* Performance-Critical Applications: When fast access to elements is required, as arrays provide O(1) access time.
* Simple Data Structures: When the dataset is simple and doesn't require frequent insertions or deletions.