**Exercise 4: Employee Management System**

* **Explain how arrays are represented in memory and their advantages**

**:-Representation of Arrays in Memory**

**Arrays** are a fundamental data structure in programming, and they are represented in memory as a contiguous block of memory locations. Each element in the array is stored at a specific index, and the memory address of each element can be calculated using the base address of the array and the size of each element.

**Memory Representation:**

1. **Contiguous Memory Allocation**: All elements of the array are stored in consecutive memory locations. This allows for efficient access and manipulation of the elements.
2. **Index Calculation**: The address of an element at index i in an array can be calculated using the formula: [ \text{Address}(A[i]) = \text{Base Address}(A) + i \times \text{Size of each element} ] where Base Address(A) is the starting address of the array, and Size of each element is the size of each array element in bytes.

**Example**:-

int[] A = {10, 20, 30, 40, 50};

Advantages of Arrays

1. Direct Access: Arrays provide direct access to elements using their index. This allows for constant time complexity (O(1)) for accessing any element, making arrays very efficient for read operations.
2. Memory Efficiency: Arrays are memory-efficient because they store elements in contiguous memory locations. This reduces the overhead associated with memory allocation and deallocation.
3. Cache Friendliness: Due to their contiguous memory allocation, arrays are cache-friendly. This means that accessing elements in an array is faster because of better cache utilization.
4. Ease of Use: Arrays are simple to declare and use. They provide a straightforward way to store and access multiple elements of the same type.
5. Fixed Size: The size of an array is fixed at the time of its creation. This can be an advantage when the number of elements is known in advance, as it allows for efficient memory allocation.
6. Support for Multiple Dimensions: Arrays can be multi-dimensional, allowing for the representation of more complex data structures like matrices and tables.
   * **Analyze the time complexity of each operation (add, search, traverse, delete).**
   * **Discuss the limitations of arrays and when to use them**.

**Time Complexity Analysis:**

* **addEmployee()**: O(1) - constant time, because we simply add the employee to the end of the array.
* **searchEmployee()**: O(n) - linear time, because we iterate through the array to find the employee.
* **traverseEmployees()**: O(n) - linear time, because we iterate through the array to display all employees.
* **deleteEmployee()**: O(n) - linear time, because we iterate through the array to find the employee and then shift the remaining employees.