**Understand Array Representation**

**Explain how Arrays Are Represented in Memory and Their Advantages?**

Memory Representation: Arrays are stored in contiguous memory locations, which means that each element of the array is stored right next to the other. This allows for efficient indexing using simple arithmetic. For example, if an array starts at memory location base\_address, the element at index i is located at base\_address + i \* element\_size. Advantages:

1. Constant-Time Access: Accessing any element by index is O(1), making arrays very efficient for retrieval.
2. Cache Friendliness: Due to their contiguous memory allocation, arrays exhibit good cache locality, leading to faster access times.
3. Predictable Memory Usage: The memory usage of arrays is fixed at creation time, which can be beneficial for memory management in certain applications.

**Analysis**

**Analyze the time complexity of each operation (add, search, traverse, delete)?**

Time Complexity of Each Operation are:

1. Add Employee: O(1) - Adding an employee is a constant-time operation as long as there is space in the array.
2. Search Employee: O(n) - Linear search through the array is required, making the time complexity proportional to the number of employees.
3. Traverse Employees: O(n) - Traversing the array to print each employee takes linear time.
4. Delete Employee: O(n) - Deleting an employee requires shifting elements in the array, resulting in linear time complexity.

**Discuss the limitations of arrays and when to use them?**

Limitations of Arrays and When to Use Them are:

1. Fixed Size: Arrays have a fixed size, which means you must allocate enough space at creation time. This can lead to wasted memory if the array is not fully utilized or the need for resizing operations if the array becomes full.
2. Inefficient Deletion: Deleting an element from an array requires shifting elements, which is inefficient for large arrays.
3. Poor Insertion Performance: Inserting an element at any position other than the end requires shifting elements, which can be inefficient.

Arrays should be used:

1. When Fixed Size Is Known: Arrays are suitable when the number of elements is known in advance and does not change frequently.
2. When Fast Access Is Needed: Arrays are ideal for scenarios where quick access to elements by index is required.
3. When Memory Contiguity Is Beneficial: Arrays are useful when you need memory-efficient storage and good cache performance.