**Understand Array Representation**

**Array Representation in Memory**

An array is a collection of elements stored in contiguous memory locations. Each element in the array is of the same data type, and they are accessed using an index. The starting index is usually zero. Arrays provide efficient access to elements because the index directly maps to a memory address, enabling constant time complexity for accessing elements.

**Advantages of Arrays**:

* **Efficient Access**: Direct access to elements using an index with O(1)O(1)O(1) time complexity.
* **Cache-Friendly**: Contiguous memory allocation makes arrays cache-friendly, improving performance.
* **Simplicity**: Simple and easy to understand data structure.

**Analysis**

**Time Complexity of Each Operation**

* **Add Employee**:
  + **Time Complexity**: O(1)O(1)O(1) - Adding an employee is done at the end of the array if there is space.
* **Search Employee**:
  + **Time Complexity**: O(n)O(n)O(n) - Linear search through the array is required to find the employee.
* **Traverse Employees**:
  + **Time Complexity**: O(n)O(n)O(n) - Traversing all elements of the array.
* **Delete Employee**:
  + **Time Complexity**: O(n)O(n)O(n) - Finding the employee is O(n)O(n)O(n), and shifting elements is also O(n)O(n)O(n).

**Limitations of Arrays**

* **Fixed Size**: Arrays have a fixed size, which must be specified at the time of creation. This can lead to wasted memory if the array is too large or insufficient space if the array is too small.
* **Inefficient Deletions and Insertions**: Deleting or inserting elements (other than at the end) requires shifting elements, which is O(n)O(n)O(n) in time complexity.
* **Lack of Flexibility**: Arrays do not support dynamic resizing or flexible data management like linked lists or dynamic arrays (e.g., ArrayList in Java).

**When to Use Arrays**

* **When the Number of Elements is Known and Fixed**: Arrays are suitable when you know the exact number of elements you need to store.
* **When Fast Access is Required**: Arrays provide O(1)O(1)O(1) access time for elements, making them ideal for scenarios where fast read access is critical.
* **Simple and Static Data Structures**: Arrays are suitable for simple and static data structures where the overhead of more complex data structures is unnecessary.