**EMPLOYEE MANAGEMENT SYSTEM**

Arrays are represented in memory as contiguous blocks of memory locations, where each location holds a single element of the array. This contiguous allocation allows for efficient access and manipulation of elements, as the memory address of each element can be calculated using the base address and index. The advantages of arrays include fast access and modification of elements, cache efficiency, and minimal memory overhead. Additionally, arrays allow for simple implementation of algorithms and data structures, making them a fundamental data structure in programming. Overall, arrays provide a compact and efficient way to store and manipulate collections of data.

Arrays have limitations, including fixed size, homogeneous elements, and inefficient insertion/deletion. They waste memory if not fully utilized. However, arrays excel in performance-critical code, cache efficiency, and simple data structures like vectors or matrices. Use arrays when data size is fixed and speed is crucial. For dynamic data or varied element types, consider alternative data structures like linked lists, vectors, or dictionaries, which offer greater flexibility and efficiency in those scenarios.

**Time Complexity of Operations:**

* **Add Employee:**
  + Best case: O(1) (if the array is not full)
  + Worst case: O(1) (still O(1), as we add the employee at the end)
* **Search Employee:**
  + Best case: O(1) (if the employee is the first element)
  + Worst case: O(n) (if the employee is not found or is the last element)
* **Traverse Employees:**
  + O(n) (where n is the number of employees)
* **Delete Employee:**
  + Best case: O(1) (if the employee is the last element)
  + Worst case: O(n) (if the employee is the first element, requiring shifting of all subsequent elements)

**Limitations of Arrays:**

* **Fixed Size:** Once an array is declared, its size cannot be changed. This can lead to wasted memory or the need to resize the array, which is a costly operation.
* **Inefficient Insertions and Deletions:** Inserting or deleting elements in the middle of the array requires shifting elements, leading to O(n) time complexity.
* **Memory Contiguity:** Arrays require contiguous blocks of memory, which may be problematic for very large arrays.

**When to Use Arrays:**

* Use arrays when you have a known, fixed number of elements and require efficient random access. They are also suitable for simple data structures like stacks and queues, where elements are added or removed from only one end.