**Employee Management System**

**Understand Array Representation**

* **Explain how arrays are represented in memory and their advantages.**
* **Representation in Memory:**
* **Contiguous Memory:** Elements are stored one after another in a single block of memory.
* **Fixed Size:** Arrays have a predetermined size, which is set when they are created.
* **Indexed Access:** Elements can be accessed directly using their index, calculated based on the starting memory address and the size of each element.
* **Advantages:**
* **Efficient Access:** Retrieving elements by index is fast because the position of each element can be directly calculated.
* **Spatial Locality:** Since elements are stored next to each other, this improves caching and memory locality, leading to faster access times.
* **Ease of Implementation:** Arrays are straightforward to implement and use in programming languages, making them a fundamental data structure.
* **Optimized for Iteration:** Iterating through an array is efficient, especially in languages with good support for loop optimizations.

**Analysis**

* **Analyse the time complexity of each operation (add, search, traverse, delete).**
* **Adding an Employee:**
* **Worst Case:** due to potential resizing, which involves copying all elements to a new, larger array.
* **Average Case (without resizing):**
* **Searching for an Employee by ID:**
* **Time Complexity:** as it may require scanning the entire array to locate the employee.
* **Traversing and Displaying All Employees:**
* **Time Complexity:** since each employee in the array needs to be visited exactly once.
* **Deleting an Employee by ID:**
* **Worst Case:** which includes finding the employee and shifting subsequent elements to fill the gap.
* **Average Case:** , primarily due to the shifting of elements.
* **Discuss the limitations of arrays and when to use them.**
* **Limitations of Arrays**
* **Fixed Size:**
* Once an array is created, its size cannot be changed. This can lead to either wasted memory if the array is too large or insufficient capacity if the array is too small.
* **Inefficient Insertions and Deletions:**
* Inserting or deleting elements (except at the end) requires shifting elements, which can be time-consuming ( complexity).
* **Contiguous Memory Requirement:**
* Arrays require a block of contiguous memory, which can be difficult to allocate if the array is large, leading to potential fragmentation issues.
* **When to use Arrays**
* **Fixed Size Collections:**
* When the number of elements is known beforehand and does not change, arrays are ideal due to their efficient indexed access.
* **Fast Access by Index:**
* When quick access to elements by their index is required, arrays are optimal due to their access time.
* **Memory Efficiency:**
* Arrays are more memory-efficient compared to other data structures like linked lists, especially when the overhead of additional pointers in linked lists is a concern.