**Exercise 4: Employee Management System**

**1. Understand Array Representation**

**Array Representation in Memory:**

* **Contiguous Memory Allocation:** Arrays are stored in contiguous memory locations, which means all elements are placed next to each other in memory. This allows for constant-time access to elements using an index.
* **Fixed Size:** Arrays have a fixed size defined at the time of declaration, which cannot be changed during runtime. This is both an advantage (predictable memory usage) and a limitation (lack of flexibility).
* **Indexing:** Arrays use zero-based indexing, meaning the first element is at index 0, the second at index 1, and so on. Accessing elements by index is an O(1) operation.

**Advantages of Arrays:**

* **Fast Access:** Elements can be accessed in constant time, O(1), due to direct indexing.
* **Efficient Memory Usage:** Since arrays are stored in contiguous memory, there is no overhead of additional memory pointers, leading to efficient memory usage.

**Analysis**

**Time Complexity Analysis:**

* **Add Employee:** The add operation checks if there is space in the array (O(1)), then adds the employee at the end of the array (O(1)). Thus, the time complexity is O(1).
* **Search Employee:** The search operation iterates through the array to find the employee with the given ID. In the worst case, it may need to check all employees, resulting in a time complexity of O(n), where n is the number of employees.
* **Traverse Employees:** Traversing the array involves iterating through all elements and printing them. This has a time complexity of O(n).
* **Delete Employee:** The delete operation involves searching for the employee (O(n)) and then shifting elements to fill the gap left by the deleted employee (O(n)). Thus, the time complexity is O(n).

**Limitations of Arrays:**

* **Fixed Size:** Arrays have a fixed size, which means their capacity cannot be changed once they are created. This limits their flexibility in dynamic scenarios.
* **Inefficient Deletion:** Deleting an element from an array requires shifting elements, which can be inefficient (O(n)).
* **Inefficient Insertion:** Inserting an element at a specific position requires shifting elements, which can also be inefficient (O(n)).
* **Wasted Space:** If the array is not fully utilized, it results in wasted memory space.

**When to Use Arrays:**

* When the number of elements is known and fixed.
* When fast access to elements by index is required.
* When memory overhead needs to be minimized.

In dynamic scenarios where the number of elements can change frequently, data structures like linked lists, dynamic arrays (ArrayList in Java), or other more flexible structures might be more appropriate.