**Exercise 4: Employee Management System**

**Steps:**

1. **Understand Array Representation:**
   * Explain how arrays are represented in memory and their advantages.

* **Array Representation in Memory:** Arrays are a data structure that stores elements of the same data type in contiguous memory locations. Each element in an array can be accessed using its index, which starts from zero. The memory address of each element can be calculated using the base address (the address of the first element) and the size of each element.
* **Advantages of Arrays:**
* **Direct Access:** Arrays provide constant time access to elements if the index is known.
* **Memory Efficiency:** They have a fixed size and minimal overhead, making them memory efficient.
* **Data Locality:** Elements are stored in contiguous memory locations, which can enhance cache performance.

1. **Analysis:**
   * Analyze the time complexity of each operation (add, search, traverse, delete).
     + **Add Employee:** O(1) — Adding an employee to the end of the array takes constant time.
     + **Search Employee:** O(n) — In the worst case, we might need to traverse the entire array to find an employee.
     + **Delete Employee:** O(n) — In the worst case, after finding the employee, we might need to shift all elements after the deleted employee to the left.
     + **Traverse Employees:** O(n) — We need to visit each employee in the array.
   * Discuss the limitations of arrays and when to use them.

* **Limitations of Arrays:**
  + - * **Fixed Size:** Arrays have a fixed size, which can lead to wasted memory if not all elements are used, or a need for resizing if the array becomes full.
      * **Inefficient Insertion/Deletion:** Insertion or deletion in the middle of the array requires shifting elements, leading to O(n) time complexity.
      * **Limited Flexibility:** Arrays are less flexible compared to other data structures like linked lists or dynamic arrays (like Python's list).
* **When to Use Arrays:**
  + - * When you have a fixed number of elements.
      * When you need fast access to elements using an index.
      * When memory overhead is a concern and you need a compact data structure.