**EXERCISE 4: Employee Management System**

**Understanding Array Representation**

**Array Representation in Memory:**

* **Contiguous Memory Allocation:** Arrays are stored in contiguous memory locations, meaning each element is stored in a consecutive block of memory. This allows for efficient indexing and quick access to any element using its index.
* **Indexing:** The elements of an array can be accessed using their indices, with the first element at index 0. The memory address of an element can be calculated as: Address of A[i]=Base Address+(i×Size of each element)\text{Address of } A[i] = \text{Base Address} + (i \times \text{Size of each element})Address of A[i]=Base Address+(i×Size of each element)
* **Fixed Size:** Arrays have a fixed size, which is defined at the time of creation. This means that the size cannot be changed once the array is created.

**Advantages of Arrays:**

* **Direct Access:** Allows for O(1) time complexity for accessing elements.
* **Memory Efficiency:** Contiguous memory allocation can be more memory efficient and cache-friendly.
* **Ease of Use:** Simple to declare, initialize, and use.

**Analysis**

**Time Complexity:**

* **Add:**
* **Time Complexity:** O(1) when there is space in the array.
* **Explanation:** Adding an employee to the array is done at the end, which takes constant time.
* **Search:**
* **Time Complexity:** O(n)
* **Explanation:** In the worst case, we might have to search through the entire array to find an employee.
* **Traverse:**
* **Time Complexity:** O(n)
* **Explanation:** Traversing the array involves visiting each element once.
* **Delete:**
* **Time Complexity:** O(n)
* **Explanation:** In the worst case, we might have to search through the entire array to find the employee to delete. After finding, deleting takes constant time but we then replace the deleted element with the last element and reduce the size, which takes O(1).

**Limitations of Arrays:**

* **Fixed Size:** Once the array is created, its size cannot be changed. This can lead to wasted space if the array is too large, or the need to create a new array and copy elements if it is too small.
* **Inefficient Deletion and Insertion:** Deleting or inserting elements (not at the end) requires shifting elements, which can be inefficient for large arrays.
* **Linear Search:** Searching for an element requires O(n) time in an unsorted array, which can be slow for large datasets.