**EX 4**

**Understanding Array Representation**

**Array Representation in Memory**

* **Contiguous Memory Allocation**: Arrays are stored in contiguous memory locations, meaning all elements are stored sequentially in memory. This allows for efficient indexing and access.
* **Indexing**: Elements can be accessed in constant time O(1) using their index because the address of any element can be calculated using the formula: address = base\_address + (index \* element\_size).
* **Advantages**:
  + **Fast Access**: Direct access to elements via indexing.
  + **Cache Friendliness**: Contiguous allocation improves cache performance.
  + **Ease of Iteration**: Simple iteration through elements due to sequential storage.

**Analysis**

**Time Complexity of Operations**

1. **Add Employee**
   * **Time Complexity**: O(1)
   * **Explanation**: Adding an element to the end of the array is a constant-time operation.
2. **Search Employee**
   * **Time Complexity**: O(n)
   * **Explanation**: In the worst case, we may need to search through the entire array to find the employee, leading to linear time complexity.
3. **Traverse Employees**
   * **Time Complexity**: O(n)
   * **Explanation**: Traversing the array involves visiting each element once, resulting in linear time complexity.
4. **Delete Employee**
   * **Time Complexity**: O(n)
   * **Explanation**: Finding the employee to delete takes O(n) time, and shifting elements to fill the gap also takes O(n) time in the worst case.

**Limitations of Arrays**

1. **Fixed Size**: Arrays have a fixed size, meaning they cannot dynamically grow or shrink, which can lead to inefficient use of memory or the need for resizing.
2. **Inefficient Insertions and Deletions**: Adding or removing elements (except at the end) requires shifting elements, leading to O(n)O(n)O(n) time complexity.
3. **Contiguous Memory Allocation**: Requires a large block of contiguous memory, which may not always be available, leading to potential memory allocation issues.
4. **No Direct Support for Advanced Operations**: Operations like dynamic resizing, flexible insertions, and deletions are not directly supported and require additional logic or data structures.

**When to Use Arrays**

* **Fixed-size Collections**: When the number of elements is known in advance and does not change frequently.
* **Fast Access**: When quick access to elements via indices is required.
* **Simple Storage Requirements**: When storage requirements are straightforward without the need for dynamic changes.