**Exercise 4: Employee Management System**

**Array Representation in Memory**

**Memory Representation:**

* **Contiguous Allocation:** Arrays are stored in contiguous memory locations, allowing direct access to elements via their indices.
* **Indexing:** Elements are accessed using the formula: Address(A[i])=Base Address+(i×Element Size)

**Advantages:**

1. **Direct Access:** O(1) time complexity for element access.
2. **Cache-Friendly:** Benefits from spatial locality, improving cache performance.
3. **Ease of Iteration:** Elements stored consecutively simplify iteration.
4. **Predictable Memory Usage:** Fixed size upon creation.
5. **Efficient Memory Use:** No overhead for pointers or structures.

**Time Complexity of Array Operations**

1. **Add (Insertion):**
   * **At the End:** O(1) if there's space.
   * **At a Specific Index:** O(n) because elements need to be shifted.
2. **Search:**
   * **Unsorted Array:** O(n) for linear search.
   * **Sorted Array:** O(log n) for binary search.
3. **Traverse:**
   * **Time Complexity:** O(n) as each element is accessed once.
4. **Delete:**
   * **At a Specific Index:** O(n) because elements need to be shifted.
   * **At the End:** O(1).

**Limitations of Arrays**

1. **Fixed Size:**
   * Arrays have a fixed size defined at creation, making them inflexible for dynamic datasets.
2. **Inefficient Insertion/Deletion:**
   * Inserting or deleting elements, especially not at the end, is inefficient (O(n)) due to shifting elements.
3. **Contiguous Memory Requirement:**
   * Requires contiguous memory, which may be difficult to allocate for large arrays.

**When to Use Arrays**

1. **When Predictable Memory Usage is Required:**
   * Arrays provide predictable and efficient memory allocation.
2. **For Fast Access and Retrieval:**
   * Ideal for scenarios requiring fast access to elements via indices (O(1) time complexity).
3. **When the Data Size is Fixed:**
   * Suitable for datasets with a known and fixed size.