**Exercise 4: Employee Management System**

**Scenario:**

You are developing an employee management system for a company. Efficiently managing employee records is crucial.

1. **Understand Array Representation:**

**How Arrays are Represented in Memory**

**Memory Representation:**

1. **Contiguous Memory Allocation:** Arrays are stored in contiguous memory locations. This means that each element of the array is placed next to its neighbouring element in memory.
2. **Index-Based Access:** Each element in the array can be accessed using its index, starting from 0 up to the size of the array minus one.
3. **Fixed Size:** The size of an array is fixed upon creation. This means that the number of elements it can hold is determined at the time of its declaration and cannot be changed dynamically.

**Advantages:**

1. **Fast Access:** Due to contiguous memory allocation and index-based access, retrieving an element from an array using its index is extremely fast (O(1) time complexity).
2. **Efficient Memory Usage:** Arrays use a compact representation, which minimizes the overhead associated with memory allocation.
3. **Analysis:**

**Time Complexity of Each Operation**

1. **Add Operation:**
   * **Scenario:** Adding an element to the end of the array (assuming dynamic resizing is handled, like in ArrayLists).
   * **Time Complexity:** O(1) for simple append, but O(n) if resizing of the array is needed.
2. **Search Operation:**
   * **Scenario:** Searching for an element in the array.
   * **Time Complexity:** O(n) in the worst case, as it may require scanning through all elements.
3. **Traverse Operation:**
   * **Scenario:** Traversing all elements of the array.
   * **Time Complexity:** O(n), where n is the number of elements in the array.
4. **Delete Operation:**
   * **Scenario:** Deleting an element from the array.
   * **Time Complexity:** O(n) in the worst case, as it may require shifting elements to fill the gap left by the deleted element.

**Limitations of Arrays and When to Use Them**

**Limitations:**

1. **Fixed Size:** Once an array is created, its size cannot be changed. This makes arrays less flexible for scenarios where the number of elements can vary.
2. **Costly Insertion/Deletion:** Inserting or deleting an element (other than at the end) can be expensive because it may require shifting elements to maintain contiguity.
3. **Memory Waste:** If the size of the array is overestimated, it can lead to wasted memory. Conversely, underestimating the size requires creating a new larger array and copying elements.

**When to Use Arrays:**

1. **Static Data:** When the number of elements is known and does not change frequently.
2. **Fast Access:** When constant-time access to elements using an index is crucial.
3. **Memory Efficiency:** When a compact memory representation is needed, and the data structure's size is known in advance.