**Exercise 5: Task Management System**

**1. Understand Array Representation**

**Array Representation in Memory**

**Arrays** are a collection of elements, all of the same type, stored in contiguous memory locations. This contiguous memory allocation allows for efficient data access and manipulation.

* **Memory Layout**:
  + Arrays are allocated in a continuous block of memory.
  + Each element in the array occupies a fixed amount of memory, allowing constant time access to any element using its index.
* **Indexing**:
  + The index of an array starts from 0.
* **Advantages**:
  + - **Fast Access**: Direct access to elements using index provides O(1) time complexity for access and modification.
    - **Efficient Storage**: Arrays provide compact and efficient storage for data of the same type.
    - **Ease of Use**: Simple syntax and direct access make arrays easy to use for many applications.

**2. Analysis: Time Complexity and Limitations**

**Time Complexity of Operations**

1. **Add Employee**:
   * **Time Complexity**: O(1) on average.
   * **Reason**: Adding an employee involves placing the new employee at the end of the array, which requires constant time as long as there's space.
2. **Search Employee**:
   * **Time Complexity**: O(n).
   * **Reason**: In the worst case, we have to check each employee until we find the desired one, leading to linear time complexity.
3. **Traverse Employees**:
   * **Time Complexity**: O(n).
   * **Reason**: We need to iterate through all the employees to print them, which takes linear time in relation to the number of employees.
4. **Delete Employee**:
   * **Time Complexity**: O(n).
   * **Reason**: After finding the employee, the remaining employees need to be shifted to fill the gap, leading to a linear time complexity.

**Limitations of Arrays**

1. **Fixed Size**:
   * Once an array is created, its size cannot be changed. This can lead to situations where the array is either full, preventing new additions, or underutilized, wasting memory space.
2. **Inefficient Deletions and Insertions**:
   * Deleting or inserting an element in the middle of the array requires shifting the other elements, resulting in a time complexity of O(n). This can be inefficient for large datasets.
3. **Memory Waste**:
   * If the array is initialized with a size larger than needed, it can lead to wasted memory space. Conversely, if it's too small, you might need to create a new larger array and copy the data over, which is costly.

**When to Use Arrays**

* **When the Number of Elements is Known**:
  + Arrays are ideal when you have a fixed number of elements and do not expect the need to add more elements than the array can hold.
* **When Memory Overhead Needs to Be Minimal**:
  + Arrays provide minimal memory overhead compared to other data structures like linked lists or hash tables.
* **When Fast Access is Required**:
  + Arrays allow O(1) access time for reading and writing elements by index, which is suitable when quick access to elements is essential.