**1. Understanding Array Representation**

**Array Representation in Memory**

An array is a data structure consisting of a collection of elements, each identified by an index. In memory, arrays are stored as a contiguous block of memory, meaning that elements are placed sequentially in adjacent memory locations. This allows for constant-time access (O(1)) to any element using its index, which is a significant advantage in terms of speed.

**Advantages of Arrays**

* **Efficient Indexing**: Accessing an element by index is O(1), making arrays very efficient for read operations.
* **Predictable Memory Usage**: Arrays use a fixed amount of memory based on their size, which helps in scenarios where memory allocation needs to be predictable.
* **Ease of Iteration**: Due to their contiguous memory layout, iterating through arrays is cache-friendly, which can enhance performance.

**4. Analysis:**

**Time Complexity of Operations**

1. **Add Employee**: O(1)
   * Appending an employee to the end of an array is typically an O(1) operation. However, if the array is full and needs to be resized, the time complexity for that operation becomes O(n).
2. **Search Employee**: O(n)
   * In the worst case, searching requires checking each element, resulting in O(n) time complexity.
3. **Traverse Employees**: O(n)
   * Traversing the array to access each element is an O(n) operation.
4. **Delete Employee**: O(n)
   * Finding the employee to delete requires a search (O(n)), and then deleting involves shifting elements, which also results in O(n) time complexity in the worst case.

**Limitations of Arrays**

1. **Fixed Size**: In static arrays, the size is fixed at the time of creation. This can lead to wasted memory if the array is underutilized or the need for resizing if it is overutilized.
2. **Inefficient Insertions and Deletions**: Insertions and deletions (not at the end) require shifting elements, which is O(n). This can be inefficient for large datasets with frequent modifications.
3. **Memory Contiguity Requirement**: Arrays require a contiguous block of memory, which can be a limitation in systems with fragmented memory.

**When to Use Arrays**

Arrays are best used when:

* The size of the data set is known and relatively stable.
* Fast access by index is required.
* The dataset has frequent read operations and infrequent insertions/deletions.
* Memory predictability is important, and cache locality can be leveraged.

For scenarios with dynamic data sizes or frequent insertions and deletions, more flexible data structures like linked lists or dynamic arrays (like Python's list, which automatically resizes) may be more suitable.