**1. Understand Array Representation**

**Array Representation in Memory**

* **Contiguous Memory Allocation**: Arrays are stored in contiguous memory locations. This means that once an array is allocated, the memory for all its elements is allocated together.
* **Index-Based Access**: Arrays provide O(1) time complexity for accessing elements by index because it directly computes the memory address using the base address and index.
* **Fixed Size**: Arrays have a fixed size. Once declared, their size cannot be changed without creating a new array.
* **Advantages**:
  + **Efficient Index-Based Access**: Quick access to elements using indices.
  + **Simple Structure**: Easy to implement and understand.
  + **Cache-Friendly**: Because elements are contiguous in memory, arrays benefit from spatial locality, which can enhance performance due to better cache utilization.

**4. Analysis**

**Time Complexity of Each Operation**

* **Add Operation**:
  + **Time Complexity**: O(1) for adding an employee to the end of the array.
  + **Limitation**: Adding an employee requires that there be available space in the array. If the array is full, resizing (if necessary) is not handled by the above code.
* **Search Operation**:
  + **Time Complexity**: O(n), where n is the number of employees. This is because, in the worst case, you may need to scan through all elements.
* **Traverse Operation**:
  + **Time Complexity**: O(n), where n is the number of employees. Each element is accessed once.
* **Delete Operation**:
  + **Time Complexity**: O(n), where n is the number of employees. This is because you may need to scan through the array to find the employee and then shift elements to fill the gap.

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

* **Fixed Size**: Arrays have a fixed size, which means you need to know the maximum number of employees ahead of time. If the number of employees exceeds this size, you'll need to create a new larger array and copy the old data.
* **Inefficient Deletion**: Deleting an employee requires shifting all subsequent elements, which can be inefficient.
* **Limited Flexibility**: Arrays do not provide dynamic resizing or easy insertion/deletion capabilities.

**When to Use Arrays**:

* **When the maximum size of the data is known** and does not change frequently.
* **When memory usage is a concern** and the overhead of dynamic data structures is unacceptable.
* **When the dataset is relatively small**, and operations such as search or delete are not performance-critical.

**Alternatives**:

* **ArrayList**: For dynamic resizing and more flexible operations, consider using ArrayList, which handles resizing and provides easier methods for insertion and deletion.
* **LinkedList**: For more efficient insertions and deletions, especially if these operations are frequent, LinkedList provides better performance for such operations compared to arrays.

In summary, while arrays are straightforward and efficient for fixed-size datasets and simple operations, their limitations make other data structures like ArrayList or LinkedList more suitable for dynamic and frequently modified datasets.