**Array Representation:**

Arrays are contiguous blocks of memory. Each element in the array is stored in a sequential memory location, and the address of the first element is used as a reference. The position of each element is determined by its index, allowing direct access to any element using its index.

* **Advantages:**
  + Fast Access: Arrays provide O(1) time complexity for accessing elements by index because you can compute the memory address directly.
  + Simple Data Structure: Arrays are straightforward to implement and use, with minimal overhead.
  + Efficient for Fixed-size Collections: Ideal for scenarios where the size of the data collection is known and does not change frequently.

1. **Time Complexity Analysis:**

* **Add Employee:** O(1)
* **Search Employee by ID:** O(n)
* **Traverse Employees:** O(n)
* **Delete Employee by ID:** O(n)

1. **Limitations of Arrays:**

* **Fixed Size:** Arrays have a fixed size that must be defined at initialization. Resizing an array involves creating a new array and copying elements, which can be inefficient.
* **Inefficient Insertions/Deletions:** Adding or removing elements requires shifting other elements, which can be time-consuming and inefficient for large arrays.
* **Overhead:** Arrays may waste space if the allocated size is much larger than the number of actual elements, or require resizing if the size is underestimated.

**When to Use Arrays:**

* **Fixed Size Collections:** Use arrays when the number of elements is known and does not change frequently.
* **Simple Data Structures:** Use arrays for simple, non-dynamic data structures where operations are straightforward and performance requirements are manageable.