**Arrays**

**How Arrays are Represented in Memory**

* Memory Allocation: Arrays are stored in contiguous memory locations. This means each element is placed directly next to the previous element in memory.
* Indexing: Arrays allow for constant-time (O(1)) access to elements using an index, which makes them very efficient for read operations.

**Advantages:**

*Fast Access*: Due to their contiguous memory storage, arrays allow for O(1) time complexity for accessing elements by index.

*Memory Efficiency*: Arrays have low memory overhead compared to other data structures because they do not require additional pointers or metadata.

**Analysis**

**Time Complexity of Each Operation**

* **Add Employee:**
  + Average Case: O(1) (if there's space in the array)
  + Worst Case: O(n) (if resizing the array is necessary)
* **Search Employee:**
  + Time Complexity: O(n) (linear search through the array)
* **Traverse Employees:**
  + Time Complexity: O(n) (visiting each element)
* **Delete Employee:**
  + Time Complexity: O(n) (searching for the employee + shifting elements)

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

* *Fixed Size*: Arrays have a fixed size, which means they need to be resized if more elements are added than the initial capacity.
* *Costly Insertions/Deletions*: Inserting or deleting elements in the middle of an array requires shifting elements, which can be costly in terms of time complexity (O(n)).
* **When to Use:**
  + Small Data Sets: When the number of elements is small or known in advance.
  + Frequent Access: When you need fast access to elements by index.
  + Static Data: When the data is relatively static and does not change frequently.