**Arrays** are a fundamental data structure used to store collections of items. In memory, arrays are represented as contiguous blocks of memory. Each element in an array is of the same type, and the position of each element is determined by an index.

**Advantages of Arrays:**

* **Fast Access:** Elements can be accessed in constant time O(1)because the memory address of each element can be calculated directly from the base address and the index.
* **Memory Efficiency:** Arrays have minimal overhead and are space-efficient since there is no additional data structure needed to keep track of elements.
* **Simplicity:** Arrays are straightforward to implement and understand, making them a good choice for simple use cases.

**Time Complexity:**

* **Add:** O(1) if there's space in the array. However, if you need to resize the array, it would involve additional operations. In this basic implementation, we assume that resizing isn't required.
* **Search:** O(n) where n is the number of employees. You have to potentially check each element in the array.
* **Traverse:** O(n) where n is the number of employees. You visit each employee exactly once.
* **Delete:** O(n) where n is the number of employees. You need to find the employee and then shift the remaining elements.

**Limitations of Arrays:**

* **Fixed Size:** Once an array is created, its size is fixed. If the number of employees exceeds the array's capacity, you can't add more employees without resizing the array.
* **Inefficient Deletions:** Deleting an element requires shifting the remaining elements, which can be costly in terms of time.
* **Search Operations:** Searching through an array takes linear time, which can be inefficient for large datasets.

**When to Use Arrays:**

Arrays are suitable when:

* The number of elements is known in advance and is fixed.
* Fast access by index is needed.
* The dataset is relatively small, and operations like search, delete, and add are not frequent or critical performance-wise.