**Employee Management System**

1. **Understand Array Representation**

**Array Representation in Memory:**

**Definition:** An array is a contiguous block of memory where each element is of the same data type and is accessed using an index. The index provides the offset from the start of the array to the desired element.

**Memory Layout:**  In memory, arrays are stored as a single, continuous block of memory. Each element is accessed via an offset from the base address of the array.

**Advantages:**

**Fast Access:**  Arrays provide constant-time access to elements (O(1)) because the memory address of an element can be computed directly using its index.

**Simplicity:** Arrays are simple to implement and use, making them ideal for scenarios where the size of the data is known and fixed.

**4. Analysis**

**Time Complexity Analysis:**

**Add Operation:**

**Best Case:** O(1) – When there is space in the array.

**Worst Case:** O (n) – When the array needs resizing and copying to a new array.

**Average Case:** O (1) – If resizing is infrequent, amortized constant time.

**Search Operation:**

**Best Case:** O (1) – When the employee is found at the beginning of the array.

**Worst Case:** O (n) – When the employee is at the end or not in the array.

**Average Case:** O (n) – Linear search in an unsorted array.

**Traverse Operation:**

**Time Complexity:** O (n) – All n elements need to be visited.

**Delete Operation:**

**Best Case:** O (n) – When the employee is found at the end.

**Worst Case:** O (n) – When the employee is at the beginning, requires shifting all subsequent elements.

**Average Case:** O (n) – Linear search plus shifting elements.

**Limitations of Arrays:**

* **Fixed Size:** Arrays have a fixed size, which can lead to inefficiency if the number of records grows beyond the initial capacity.
* **Insertion/Deletion Overhead:** Inserting or deleting elements requires shifting elements and can be inefficient, particularly in large arrays.
* **No Built-In Search/Sort:** Unlike more advanced data structures like hash tables or balanced trees, arrays lack built-in support for efficient searching or sorting.

**When to Use Arrays:**

* **Static Data Size:** When the number of elements is known and does not change frequently.
* **Simplicity:** When a simple, contiguous block of memory is sufficient, and operations are primarily read-intensive.
* **Performance:** For scenarios where constant-time access is crucial and resizing or frequent insertions/deletions are minimal.