**EXERCISE 4**

**Employee Management System**

**1. Understanding Array Representation**

**Array Representation in Memory**

Arrays are data structures that store elements of the same type in contiguous memory locations. Each element in the array can be accessed directly using its index. The key features of arrays are:

* **Contiguous Memory Allocation**: Arrays allocate memory in a single block, which leads to efficient indexing and retrieval of elements.
* **Direct Access**: Array elements can be accessed in constant time O(1) using their indices.
* **Fixed Size**: The size of an array is defined at the time of its creation and cannot be changed.

**Advantages of Arrays**:

* **Efficient Access**: Direct indexing provides quick access to any element.
* **Low Overhead**: Arrays have minimal overhead compared to other data structures like linked lists or hash tables.

**2. Setup**

Define a class to represent an employee with attributes such as employeeId, name, position, and salary. This class will be used to create and manage employee records.

**3. Implementation**

**Using an Array to Manage Employee Records**

* **Add Employees**: To add employees, you insert them into the array at the next available position. If the array is full, additional space needs to be allocated or handled accordingly.
* **Search Employees**: To find an employee, you iterate through the array and check each element until the desired employee is found.
* **Traverse Employees**: To display all employees, iterate through the array and print each employee's details.
* **Delete Employees**: To remove an employee, locate the employee, shift subsequent elements left to fill the gap, and update the size of the array.

**4. Analysis**

**Time Complexity**

* **Add Operation**:
  + **Best Case**: O(1) – If there is available space in the array.
  + **Worst Case**: O(1) – Since adding an element involves inserting it at the next available index..
* **Search Operation**:
  + **Best Case**: O(1) – If the employee is located at the first index.
  + **Worst Case**: O(n) – If the employee is located at the end of the array or not present.
* **Traverse Operation**:
  + **Best Case**: O(n) – Traversing the entire array to display all employees.
  + **Worst Case**: O(n) – Similar to the average case as traversal requires checking each element.
* **Delete Operation**:
  + **Best Case**: O(n) – If the employee is at the end of the array, only a single step is needed to delete.
  + **Worst Case**: O(n) – Similar to the average case since deletion involves shifting elements.

**Limitations of Arrays**

* **Fixed Size**: Arrays have a fixed size which means that resizing is not straightforward and can lead to inefficient use of space.
* **Insertion and Deletion Complexity**: Adding or removing elements involves shifting elements, which can be inefficient.
* **Wasted Space**: If the array is initially allocated with extra capacity, it can result in wasted memory if the space is not utilized.

**Use of Arrays**

* **Predictable Size**: Use arrays when the size of the data is known and remains constant.
* **Efficient Indexing**: When you need fast access to elements using indices and the data does not change frequently.
* **Simple Data Handling**: For simple data storage where complex operations are not required.

Arrays are beneficial for their simplicity and efficiency in accessing elements but are less suitable for dynamic data structures where frequent resizing and modifications are needed.