Step 1: Understand Array Representation

Array Representation in Memory:

Arrays are a collection of elements stored in contiguous memory locations.

The address of the first element (base address) and the element's size are used to calculate the address of any element.

Advantages:

Direct Access: Arrays allow direct access to elements using their index, providing O(1) access time.

Memory Efficiency: They are memory efficient since they store elements in contiguous blocks.

Step 2: Setup

Employee Class:

Create an Employee class with attributes employeeId, name, position, and salary.

Step 3: Implementation

Using Arrays for Employee Records:

Arrays are used to store employee records.

Implement methods to add, search, traverse, and delete employees in the array.

Step 4: Analysis

Time Complexity:

Add Employee:

Best Case: O(1) - If there's space in the array, adding an employee is an O(1) operation.

Worst Case: O(n) - If the array is full and needs resizing (not covered here), copying elements to a new array takes O(n) time.

Search Employee:

Best, Average, and Worst Case: O(n) - Searching an unsorted array requires checking each element.

Traverse Employees:

Best, Average, and Worst Case: O(n) - Traversing the array to print all employees requires visiting each element once.

Delete Employee:

Best Case: O(1) - If the employee to be deleted is the last one.

Worst Case: O(n) - If the employee to be deleted is the first one, all subsequent elements need to be shifted.

Limitations of Arrays:

Fixed Size: Arrays have a fixed size; adding more elements than the allocated space requires creating a new, larger array and copying existing elements.

Inefficient Deletion/Inserting: Deleting or inserting elements (other than at the end) requires shifting elements, which is O(n).

Contiguous Memory: Arrays require contiguous memory allocation, which can be a limitation in systems with fragmented memory.

Example Analysis Using Code

Adding Employees:

Employees are added sequentially to the array until it is full.

If the array is full, a message is printed indicating that no more employees can be added.

Searching for an Employee:

Searches through the array for an employee with the given employeeId.

Returns the employee if found, otherwise returns null.

Traversing Employees:

Prints all employees in the array.

Traverses the array from start to end, printing each employee.

Deleting an Employee:

Searches for the employee with the given employeeId.

If found, shifts all subsequent employees one position to the left to fill the gap.

Decreases the size of the array and removes the last reference.

Conclusion:

Arrays provide efficient direct access and are memory efficient, but their fixed size and inefficiency in inserting and deleting elements limit their use for dynamic datasets.

Alternative Data Structures: For dynamic datasets with frequent insertions and deletions, data structures like ArrayLists, LinkedLists, or HashMaps might be more suitable.