Exercise 4: Employee Management System

1. Understanding Array Representation

In Java, arrays are contiguous memory blocks that hold elements of the same data type. Each element can be accessed directly via an index, making arrays efficient for read and write operations.

Advantages of Arrays:

- **Fast access**: O(1) time to access any element by index.
- **Memory locality**: Improves cache performance due to contiguous memory.
- **Predictable size**: Useful when the number of elements is fixed or known.

Use Case in Employee Management

In an employee management system, storing employees in an array is beneficial when the number of employees is fixed or doesn't change frequently. It allows efficient traversal and searching by index.

2. Time Complexity Analysis

Adding an Employee

- **Scenario**: Adding a new employee to the next available position in the array.
- **Time Complexity**:
  - **Best/Average Case**: O(1) – if the array is not full.
  - **Worst Case**: O(1) – still constant as insertion is at a known index.

Searching an Employee

- **Scenario**: Searching for an employee by ID or name.

- **Time Complexity**:
    - **Best Case**: O(1) – if the employee is found at the beginning.
    - **Average/Worst Case**: O(n) – need to scan all n elements.

Traversing Employees

- **Scenario**: Printing or processing each employee.
- **Time Complexity**:
    - **Always**: O(n) – must visit every employee in the array.

Deleting an Employee

- **Scenario**: Deleting by ID or index, followed by shifting elements.
- **Time Complexity**:
    - **Best Case**: O(1) – if the employee is at the last index.
    - **Average/Worst Case**: O(n) – shift all elements after the deleted index.

3. Limitations of Arrays and When to Use

Limitations:

- **Fixed size**: Cannot grow dynamically; you must define the size at creation.
- **Inefficient deletion/insertion in the middle**: Requires shifting elements.
- **Wasted space**: If not all positions are used, memory is wasted.

When to Use Arrays:

- The number of employees is known in advance.
- The system requires fast indexed access.

- Insertions and deletions are infrequent.
- Memory overhead must be kept low.