1. **Explain how arrays are represented in memory and their advantages.**

An array is a collection of elements stored in contiguous memory locations. Each element in the array is of the same data type and can be accessed using an index.

* **Memory Layout**: Arrays are stored in a single, contiguous block of memory. The index of an array element determines its location in this block. For example, the element at index i can be accessed using the formula:

Address of element=Base address+(i×Size of element)

* **Advantages**:
  + **Direct Access**: Arrays allow constant-time access to elements using indices (O(1) time complexity for access).
  + **Simplicity**: Arrays are straightforward to implement and use.
  + **Memory Efficiency**: They have low overhead since they store elements in contiguous memory.

**Analyze the time complexity of each operation (add, search, traverse, delete).**

**Time Complexity**

1. **Adding Employees**:
   * **Average Case**: O(1) – Adding an employee if there's space.
   * **Worst Case**: O(n) – When resizing is required.
2. **Searching for Employees**:
   * **Average Case**: O(n) – Linear search for a specific employeeId.
   * **Worst Case**: O(n) – Linear search, irrespective of data distribution.
3. **Traversing Employees**:
   * **Time Complexity**: O(n) – Visit each employee in the array.
4. **Deleting Employees**:
   * **Average Case**: O(n) – Linear search followed by shifting elements.
   * **Worst Case**: O(n) – As above, involves searching and shifting.

**Discuss the limitations of arrays and when to use them.**

**Limitations of Array**

* **Fixed Size**: Arrays have a fixed size once created, requiring resizing operations which can be costly.
* **Inefficient Insertions/Deletions**: Adding or removing elements involves shifting elements, which is inefficient compared to other data structures like linked lists.
* **Limited Flexibility**: Arrays are not suitable for scenarios requiring frequent size adjustments.

**When to Use Arrays**

* **Static Data**: When the number of elements is known and does not change frequently.
* **Simple Use Cases**: For simple applications where performance impact of resizing and shifting is negligible.
* **Direct Access Needs**: When constant-time access to elements via indices is needed.

For dynamic and large-scale data management, alternative data structures like ArrayList or other collections from the Java Collections Framework may be more appropriate.