Understanding Array Representation

In Java, an **array is a contiguous block of memory** where elements are stored at fixed positions. Each element can be accessed directly using an index, which starts from 0. For example, in an array of employees, arr[0] points to the first employee, arr[1] to the second, and so on.

This allows **constant-time access (O(1))** to any element using its index, making arrays highly efficient for **traversal and direct access**.

Since arrays are of **fixed size**, memory is allocated at compile-time (for static arrays) or during runtime (for dynamic arrays).

Their main advantages are **speed, simplicity, and predictability**. However, they lack flexibility in terms of dynamic resizing, which can make insertions or deletions costly in certain cases.

ANALYSIS

* Each operation in the array-based employee management system has different time complexities. **Adding an employee** at the end takes **O(1)** time as we simply insert at the next available index. **Searching for an employee** by ID requires scanning the array, making it **O(n)** in the worst case. **Traversing** the entire array is also **O(n)** as each element must be accessed and printed. **Deleting an employee** is the most costly operation, as we must first find the element (O(n)) and then shift all subsequent elements one position to the left (also O(n)), resulting in **O(n)** time complexity.
* While arrays are simple and fast for access and traversal, their **main limitation** is that they are of **fixed size** and not suitable for frequent insertions or deletions at arbitrary positions. In such cases, **ArrayLists or LinkedLists** offer more flexibility. Arrays are best used when the number of elements is known in advance and operations are mostly read-heavy rather than write-heavy.