**1. Understanding Array Representation:**

Array Representation in Memory

* **Contiguous Block:** Arrays are stored in a single, continuous block of memory. Each element is located next to the previous one.
* **Index-Based Access:** Elements are accessed using indices. The index tells you how far to move from the start of the array to get to the desired element.

Advantages of Arrays

* **Fast Access:** Accessing any element by its index is very fast (constant time, O(1)).
* **Efficient Memory Use:** Arrays use memory efficiently because they don’t require extra space
* **Simple to Use:** Easy to implement and use, with predictable performance.

**2. Time complexity and limitations**

* Add Operation

Time Complexity:

O(1): Adding an element to an ArrayList typically takes constant time, O(1), because ArrayList can grow dynamically and manages capacity internally.

* Search Operation

Time Complexity:

O(n): Searching for an employee involves iterating through the ArrayList to find the employee with the matching ID. In the worst case,we check every element in the ArrayList, resulting in a linear time complexity, O(n), where n is the number of employees.

* Traverse Operation

Time Complexity:

O(n): Traversing the entire ArrayList requires visiting each element once to display it.

* Delete Operation

Time Complexity:

O(n): Similar to the search operation, deleting an employee involves finding the employee by iterating through the ArrayList (O(n)) and then removing it

Limitations of Arrays

* Resizing Costs:
  + Although adding elements to an ArrayList is typically O(1) on average, resizing (when the internal array's capacity is exceeded) involves creating a new, larger array and copying existing elements to it. This resizing operation is O(n), where n is the number of elements in the ArrayList. However, resizing happens infrequently, so the average cost is spread out.
* Insertion/Deletion Costs:
  + O(n) Complexity: Inserting or deleting elements, especially in the middle of the list, requires shifting elements to maintain contiguous storage. This operation has a time complexity of O(n), where n is the number of elements to shift.
* Capacity Management:
  + Unused Space: ArrayList often allocates more space than needed to accommodate future growth, which can lead to unused capacity. This can lead to wasted memory, particularly if the array grows significantly larger than the number of elements it holds.
* Additional Memory for Internal Structures:
  + ArrayList uses an internal array to store its elements, which means that memory is used for both the internal array and any elements stored in it. This can be inefficient if the list is very sparse or if the internal array is resized frequently.
* Index-Based Access:

While accessing elements by index is very efficient (O(1)), other operations like searching for an element (if not by index) or complex queries may not be as efficient and could require iterating through the list.