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

Arrays are represented in memory as a **contiguous block of memory locations**. Each element in the array is stored at a fixed distance (based on the data type size) from the previous one. This allows for **constant time O(1) access** to any element using its index, as the memory address of any element can be directly calculated using the base address and the index.

**Advantages of arrays include**:

* **Fast access** to elements using indexing.
* **Ease of implementation** due to their simple structure.
* **Efficient use of memory** for fixed-size collections.

However, arrays have a fixed size once declared, and inserting or deleting elements in the middle requires shifting elements, which can be inefficient.

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

**Time Complexity Analysis (Using Arrays)**  
**Add**: Best case is O(1) if adding at the end. Worst case is O(n) if inserting in the middle, as it requires shifting elements.  
**Search**: O(n), since linear search is needed unless the array is sorted and binary search is applied.  
**Traverse**: O(n), because each element must be visited once.  
**Delete**: O(n), as elements must be shifted after deletion to maintain the order.

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

### ****Limitations of Arrays and When to Use Them****

Arrays are of fixed size, which means the size must be known in advance and cannot be changed at runtime. They also require contiguous memory, which can be a problem for large data. Inserting or deleting elements is costly because shifting is required.

However, arrays are ideal when:

* The number of elements is fixed or known beforehand.
* Fast access to elements using index is required.
* Memory overhead must be kept low.