Explain how arrays are represented in memory and their advantages.

* Contiguous Memory Allocation: Arrays are stored in contiguous memory locations, meaning each element is placed next to the other in memory. This allows for constant-time access to any element using its index.
* Indexing: Elements in an array can be accessed directly using their index, which makes array operations fast and efficient. For example, the element at index i in an array arr can be accessed with arr[i].

Advantages of Arrays:

* Direct indexing allows O(1) time complexity for accessing elements.
* Arrays have a fixed size, and the memory allocation is done at the time of array creation, ensuring no overhead for additional memory management.
* Since arrays are stored contiguously, cache locality is better, leading to faster access times due to fewer cache misses.

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

Time Complexity of Operations:

* Add Operation: O(1) - Adding an element to the end of the array takes constant time if there is available capacity.
* Search Operation: O(n) - In the worst case, we might have to search through the entire array to find an element.
* Traverse Operation: O(n) - We need to visit each element in the array.
* Delete Operation: O(n) - In the worst case, we need to shift all elements after the deleted element.

Discuss the limitations of arrays and when to use them.

These are array limitatation:

* Arrays have a fixed size, which means the capacity must be defined at the time of creation. If the array is full, no more elements can be added without resizing the array, which is an expensive operation.
* Inefficient Insertions/Deletions: Inserting or deleting elements, especially in the middle of the array, requires shifting elements, which takes O(n) time.
* Arrays are not dynamic and cannot easily grow or shrink in size. This can lead to wasted space or overflow.

Use of arrays:

* Static Data Size: When the number of elements is known in advance and does not change frequently.
* Fast Access: When fast access to elements by index is required.
* Memory Efficiency: When memory allocation needs to be efficient and predictable.