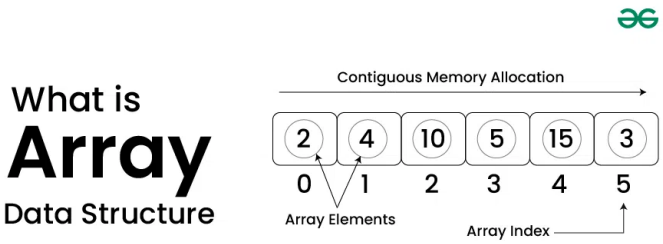
ARRAY

1. **Definition** - Array is linear DS, where elements are arranged sequentially. It is a collection of elements of the same data type stored at **contiguous memory locations**.

contiguous memory locations.



2. **Memory representation of Array**
    a. In an array, all the elements are stored in contiguous memory locations. So, if we initialize an array, the elements will be allocated sequentially in memory. This allows for efficient access and manipulation of elements.

3. **Fixed arrays** and **dynamic arrays**

Fixed Array

- **Size**: Size is set during initialization and cannot be changed.
- **Memory Allocation**: Allocated in contiguous memory locations.
- **Access**: Fast access using an index (O(1)).
- **Resizing**: Not possible to resize; once full, a new array must be created.
- **Efficiency**: Space-efficient since memory is pre-allocated.
- **Example**: Arrays in C (`int arr[10];`).

Dynamic Array

- **Size**: Size can grow or shrink dynamically as needed.
- **Memory Allocation**: Allocates memory in blocks and expands when necessary.
- **Access**: Also supports fast access using an index (O(1)).

- **Resizing**: Automatically resizes by copying old elements to a new larger/smaller block of memory.
- **Efficiency**: Less space-efficient due to potential overallocation when resizing.
- **Example**: Lists in Python (`arr = [1, 2, 3]`).

Example in Python (dynamic array):

```
arr = [1, 2, 3]  # Dynamic array (list)
arr.append(4)    # Array grows dynamically
```

4. **Applications:**
   a. **Data Storage**: Arrays store data like lists of user IDs, scores, or sensor readings.
   b. **Image Processing**: Images are stored as 2D arrays of pixel values.
   c. **Hash Tables**: Arrays are used in implementing hash tables for quick lookup.
   d. **Scheduling Algorithms**: Arrays store jobs or tasks in operating system schedulers.

5. **Subarrays, subsequence, and subsets:**
   a. **Subarray**: Contiguous. n * (n + 1)
   b. **Subsets:** Any combination of elements (order irrelevant). 2^n.
   c. **Subsequence**: Maintains order, but elements may be skipped. 2^n.

```
# Generating subarrays
arr = [1, 2, 3, 4, 5]
res = []
for i in range(len(arr)):
    for j in range(i + 1, len(arr)+1):
        res.append(arr[i:j])
print(res)
```

6. **Operations:**

   Key operations in arrays include:

   1. **Traversal**: Access each element one by one.
      - Example: Looping through the array.
      - Time Complexity: O(n)
   2. **Accessing Elements**: Retrieve an element using its index.
      - Example: `arr[2]` to access the third element.

- Time Complexity: O(1)

3. **Insertion**: Add an element at a specific position.
   - Example: Inserting at the end or at an index.
   - Time Complexity:
     - End: O(1)
     - Middle/Beginning: O(n) (requires shifting elements)
4. **Deletion**: Remove an element from a specific position.
   - Example: Removing an element at index `i`.
   - Time Complexity: O(n) (requires shifting elements)
5. **Searching**: Find the index of a specific element.
   - Example: Linear search or binary search.
   - Time Complexity:
     - Linear search: O(n)
     - Binary search (sorted array): O(log n)
6. **Updating**: Modify an element at a specific index.
   - Example: `arr[1] = 10`
   - Time Complexity: O(1)
7. **Rotation**: Rotating the array by k times left or right.

7. **Advantages:**
   a. **Fast Access**: Access elements in O(1) time using index.
      - Example: `arr[2]` gives the third element instantly.
   b. **Memory Efficient**: Stored contiguously, saving memory.
      - Example: An array like `arr = [1, 2, 3]` takes less memory than linked lists.
8. **Disadvantages:**
   a. **Fixed Size**: Cannot resize after creation (in static arrays).
      - Example: `arr = [1, 2, 3]` can't grow beyond 3 elements.
   b. **Costly Insert/Delete**: Shifting elements takes O(n) time.
      - Example: Deleting the first element of `[1, 2, 3]` shifts `2` and `3`, taking more time.