Arrays

1. Introduction to Arrays

- **Definition:**
  An array is a collection of elements of the **same data type**, stored at **contiguous memory locations**, and accessed using an **index.**

- **Indexing:**
  - → Index starts from **0.**

Example:

```
int arr[5] = {10, 20, 30, 40, 50};
// arr[0] = 10, arr[1] = 20, arr[2] = 30 …
```

2. Features of Arrays

- Stores multiple values under one name.
- Fixed size (declared once).
- Fast access (O(1)) using index.
- Elements stored in contiguous memory.

3. Declaring and Initializing Arrays

Declaration

```
int arr[5];   // uninitialized array of size 5
```

Initialization (Static)

```
int arr[5] = {10, 20, 30, 40, 50};
int arr[] = {1, 2, 3, 4};   // compiler decides size
```

Partial Initialization

```
int arr[5] = {10, 20};
// arr[0]=10, arr[1]=20, arr[2]=0, arr[3]=0, arr[4]=0
```

4. Taking Input from User

```
#include <iostream>
using namespace std;

int main() {
    int n;
    cout << "Enter size of array: ";
    cin >> n;

    int arr[n];
    cout << "Enter elements: ";
    for(int i=0; i<n; i++) {
        cin >> arr[i];
    }

    cout << "Array elements: ";
    for(int i=0; i<n; i++) {
```

```
        cout << arr[i] << " ";
    }
    return 0;
}
```

5. Traversal of Array

- Visiting each element once using loop.
- **Time Complexity:** O(n).

```
for(int i=0; i<n; i++) {
    cout << arr[i] << " ";
}
```

6. Array Operations

(A) Insertion

At end:

```
arr[n] = value;
```

-
- At index: shift elements → costly operation **O(n).**

(B) Deletion

- Remove element at index and shift left.

(C) Updation

```
arr[2] = 100; // updates 3rd element
```

(D) Searching

- **Linear Search** → O(n).
- **Binary Search** (only for sorted arrays) → O(log n).

7. Memory Representation of Array

- Arrays are stored sequentially in memory.

Formula:

```
Address(arr[i]) = Base address + (i * size_of_each_element)
```

-
- **Example:**
  If base address = 1000, each int = 4 bytes →
  - arr[0] = 1000
  - arr[1] = 1004
  - arr[2] = 1008 …

8. Types of Arrays

(A) One-Dimensional Array (1D Array)

Stores elements in a single row.

```
int arr[5] = {1, 2, 3, 4, 5};
```

(B) Two-Dimensional Array (2D Array or Matrix)

Stores elements in rows and columns.

```
int matrix[2][3] = {{1,2,3}, {4,5,6}};
```

(C) Multi-Dimensional Array

Arrays with more than two dimensions (rare).

```
int arr[3][3][3];
```

9. Advantages of Arrays

Random access (O(1)).
Easy to use and implement.
Best for storing fixed-size data.

10. Disadvantages of Arrays

Fixed size (cannot grow/shrink).
Insertion/Deletion is costly (O(n)).
Wastage of memory if array is not full.

11. Complexity Analysis

Operation	Time Complexity
Traversal	O(n)
Insertion (at end)	O(1)
Insertion (at position)	O(n)
Deletion	O(n)
Searching (Linear)	O(n)
Searching (Binary, Sorted)	O(log n)