Arrays

Course: Algorithms, Data Structures, and Programming

LECTURE GOALS

Understand the concept

Of an array as a data structure.

Know the organization

Of arrays in memory.

Analyze the complexity

Of array operations.

Understand the differences

Between one-dimensional and multi-dimensional arrays.

Introduction and Motivation

Problem Statement

How to store grades for 100 students?

- Inefficient to use separate variables.
- Impossible to process in a loop.
- Difficulty in scaling.

Solution — Arrays

An array as a container for storing homogeneous data:

- Access by index in O(1).
- Possibility of processing in loops.

Array Data Structure

An array is a fixed-size data structure that stores elements of the same type in a contiguous memory region.

Fixed Size

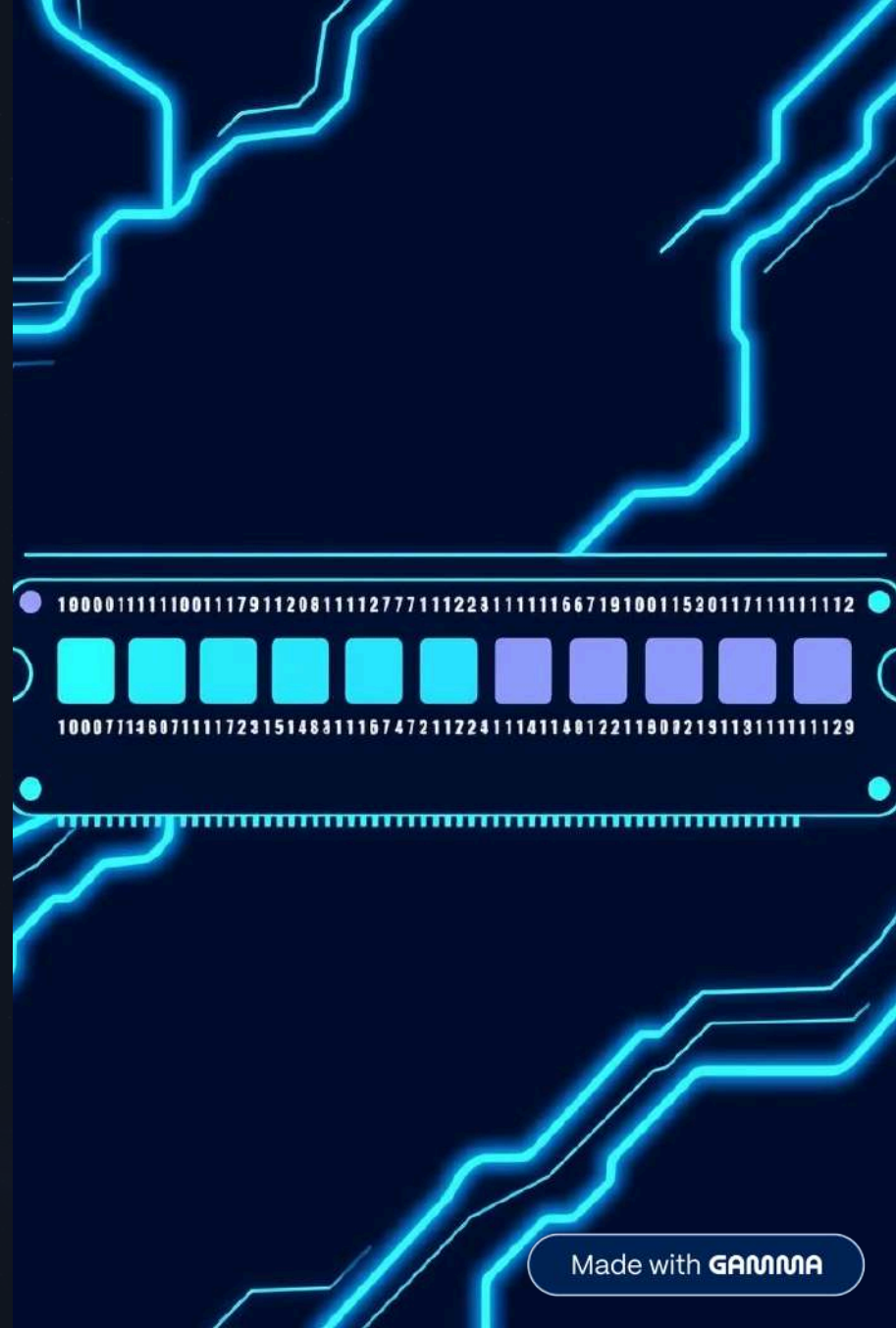
Defined at creation, cannot be changed.

Homogeneous Elements

All elements have the same data type.

Sequential Placement

Elements are located one after another in memory.



Memory Allocation and Access

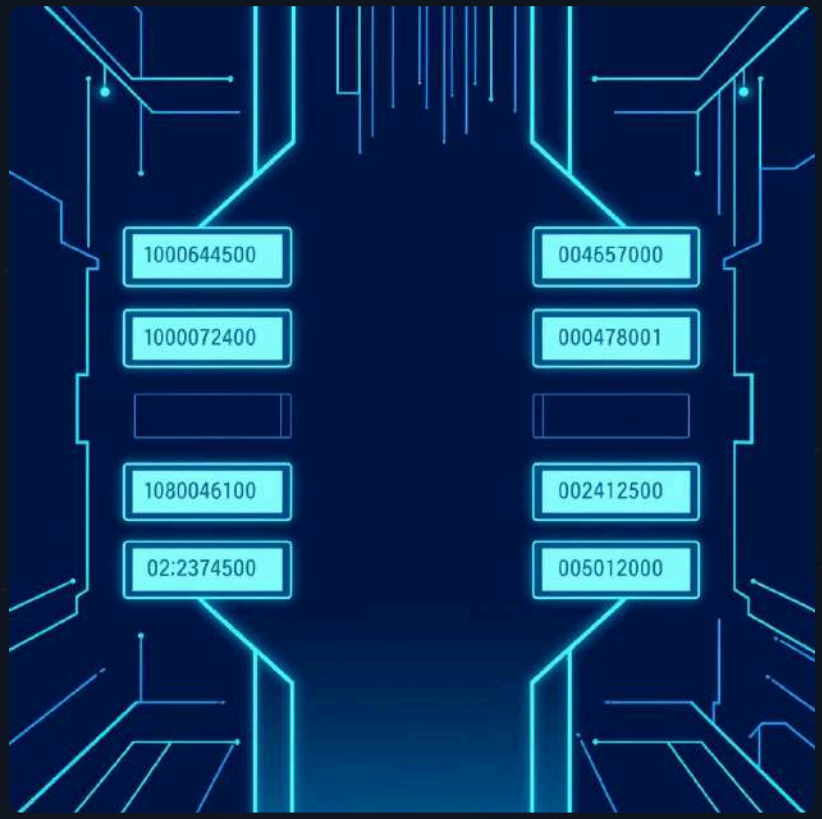
Memory Allocation Scheme

```
Array int arr[5] = {10, 20, 30, 40, 50};
Memory:
Address: 1000 1004 1008 1012 1016
+-----+
Value:| 10 | 20 | 30 | 40 | 50 |
+-----+
Index: 0 1 2 3 4
```

Element Address Calculation

Address arr[i] = BaseAddress + i × sizeof(data_type)

Example: arr[3] = 1000 + 3 × 4 = 1012



Time Complexity Analysis

The time complexity of an algorithm describes how the algorithm's execution time grows with the increase in input size. This is critically important for understanding the scalability and performance of programs.

Types of Analysis

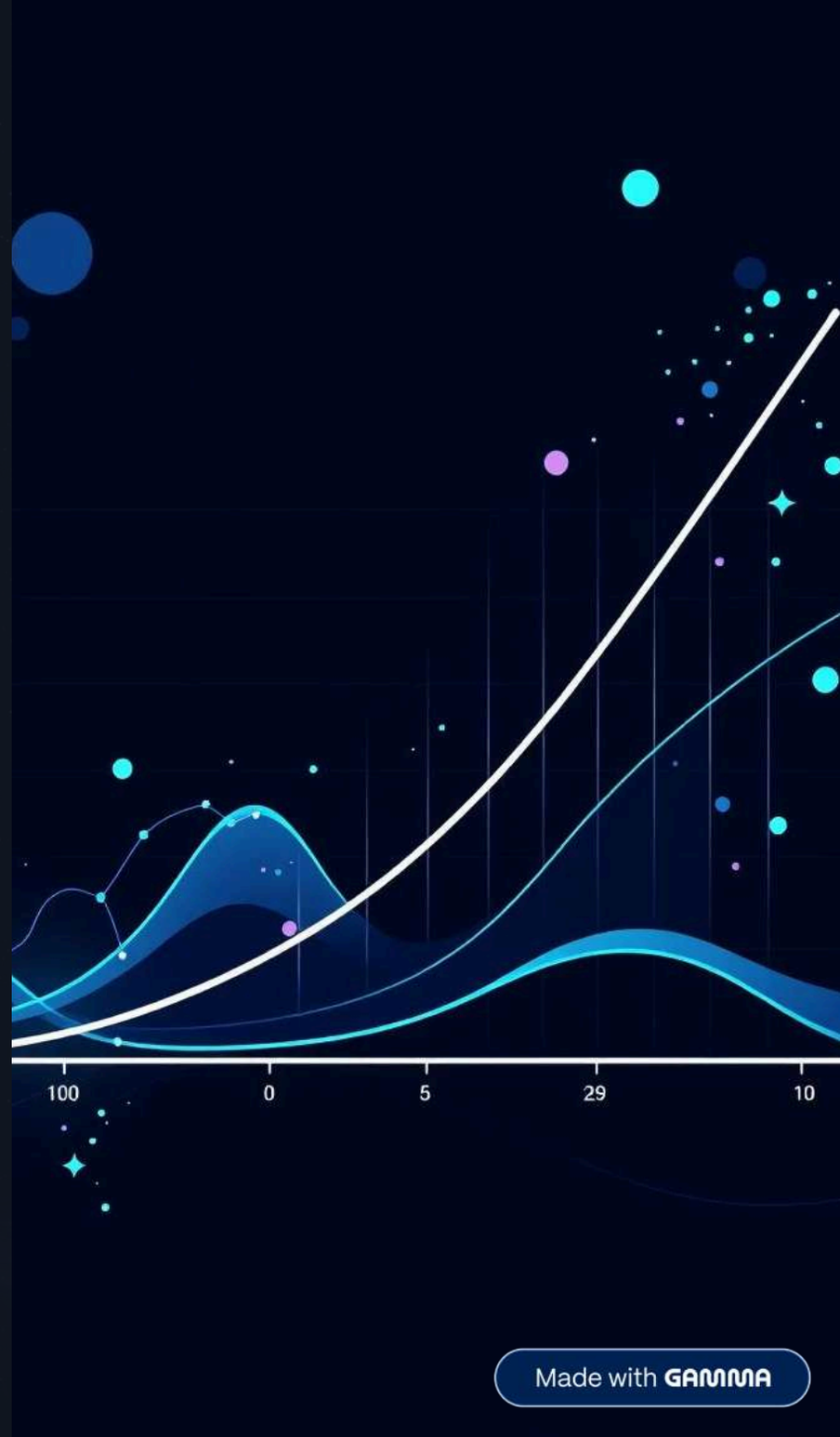
There are best, average, and worst-case scenarios for execution. We are most often interested in the worst-case, as it guarantees an upper bound on performance.

Why O-Notation?

Big O notation (O-notation) allows us to abstract away from specific implementation details and hardware, focusing on how an algorithm scales with increasing input data (N).

What is O-Notation?

It describes the upper bound of the growth of execution time, showing how quickly an algorithm becomes slower as the input data size (N) increases. For example, O(N) for linear growth, O(N^2) for quadratic.



One-dimensional Arrays: Operations and Complexity

Declaration and Initialization in C++

```
// Объявление
int arr[10]; // массив из 10 целых чисел

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

Accessing Elements

```
arr[0] = 100; // запись
int x = arr[2]; // чтение
```

Core Operations and Their Complexity

Operation	Complexity	Explanation
Index Access	O(1)	Direct addressing
Element Search	O(n)	Requires iteration
Insert at End	O(1)	If space is available
Insert in Middle	O(n)	Shifting elements
Deletion	O(n)	Shifting elements

Two-Dimensional Arrays (Matrices)

Concept and Declaration

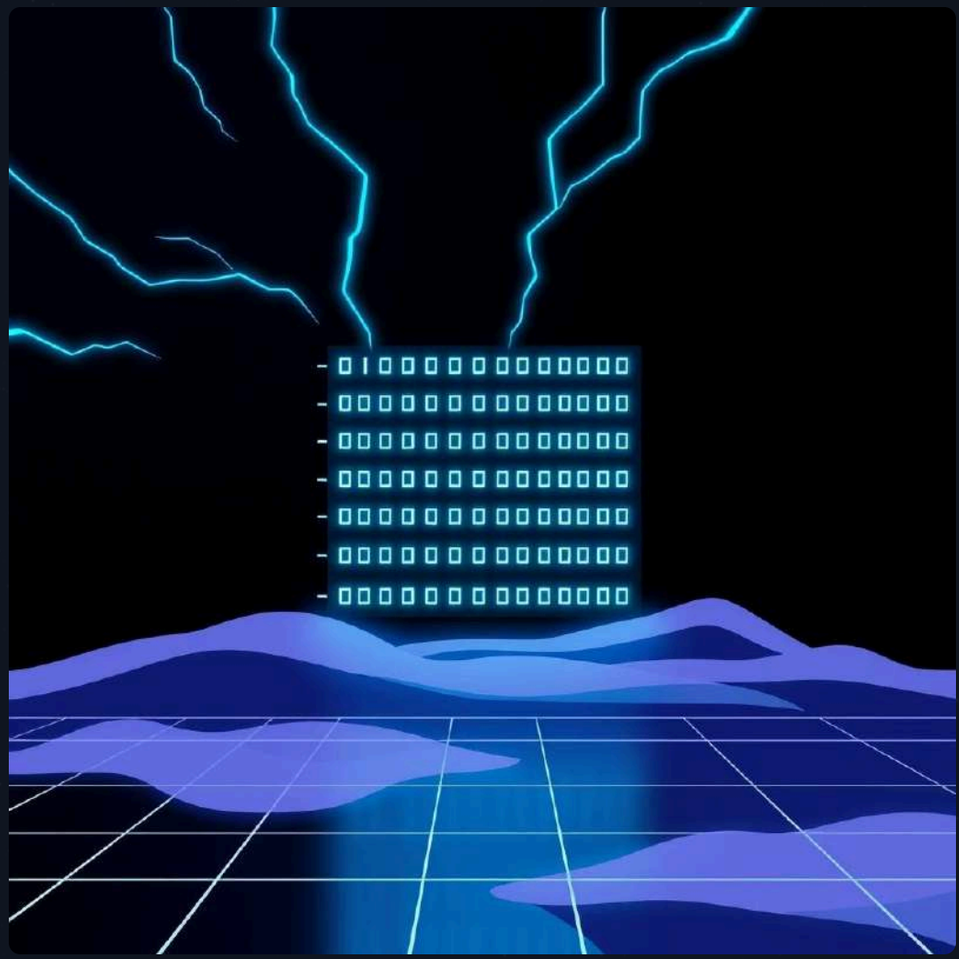
Array of arrays (matrix), a table with rows and columns.

```
int matrix[3][4]; // matrix 3x4
```

Memory Allocation (Row-major order)

Logical Representation:
[0] [1] [2] [3]
[0] 1 2 3 4
[1] 5 6 7 8
[2] 9 10 11 12

Physical Arrangement:
[1][2][3][4][5][6][7][8][9][10][11][12]
↑ ↑ ↑
row 0 row 1 row 2



Formula for Element Address matrix[i][j]

Address = BaseAddress + (i × number_of_columns + j) × sizeof(type)

Example: matrix[2][1] = 2000 + (2 × 4 + 1) × 4 = 2036

Time Complexity Analysis

1

Finding Maximum in an Array

```
int findMax(int arr[], int n) {
    int max = arr[0];
    for(int i = 1; i < n; i++)
        if(arr[i] > max) max = arr[i];
    return max;
} // Overall Complexity: O(n)
```

2

Linear Search for an Element

```
int linearSearch(int arr[], int n, int key) {
    for(int i = 0; i < n; i++)
        if(arr[i] == key) return i;
    return -1;
} // Complexity: O(n)
```

3

Matrix Processing

```
int sumMatrix(int mat[][M], int N, int M) {
 int sum = 0;
 for(int i = 0; i < N; i++)
 for(int j = 0; j < M; j++)
 sum += mat[i][j];
 return sum;
} // Overall Complexity: O(N × M)
```

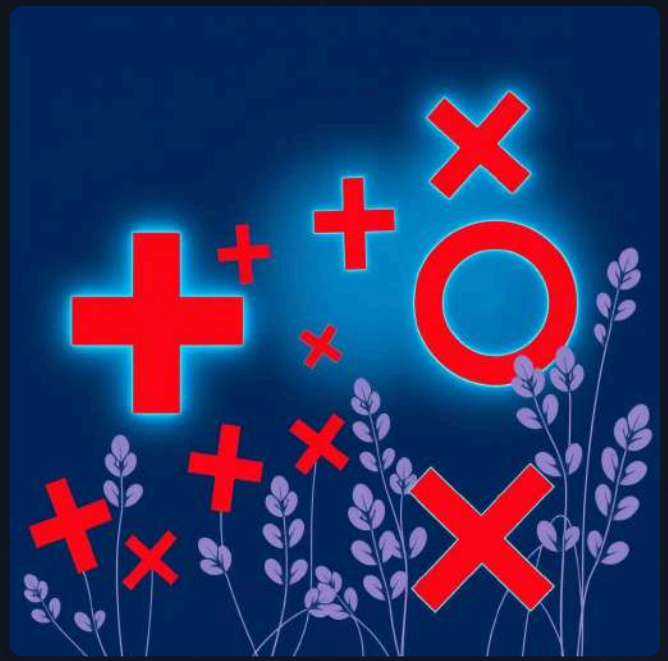
Practical Aspects and Application

Advantages of Arrays

- Fast access by index O(1).
- Efficient memory usage.
- Good data locality (cache-friendly).
- Simplicity of implementation.

Disadvantages of Arrays

- Fixed size.
- Inefficient insertion/deletion O(n).
- Possibility of out-of-bounds access.
- Unused memory with partial filling.



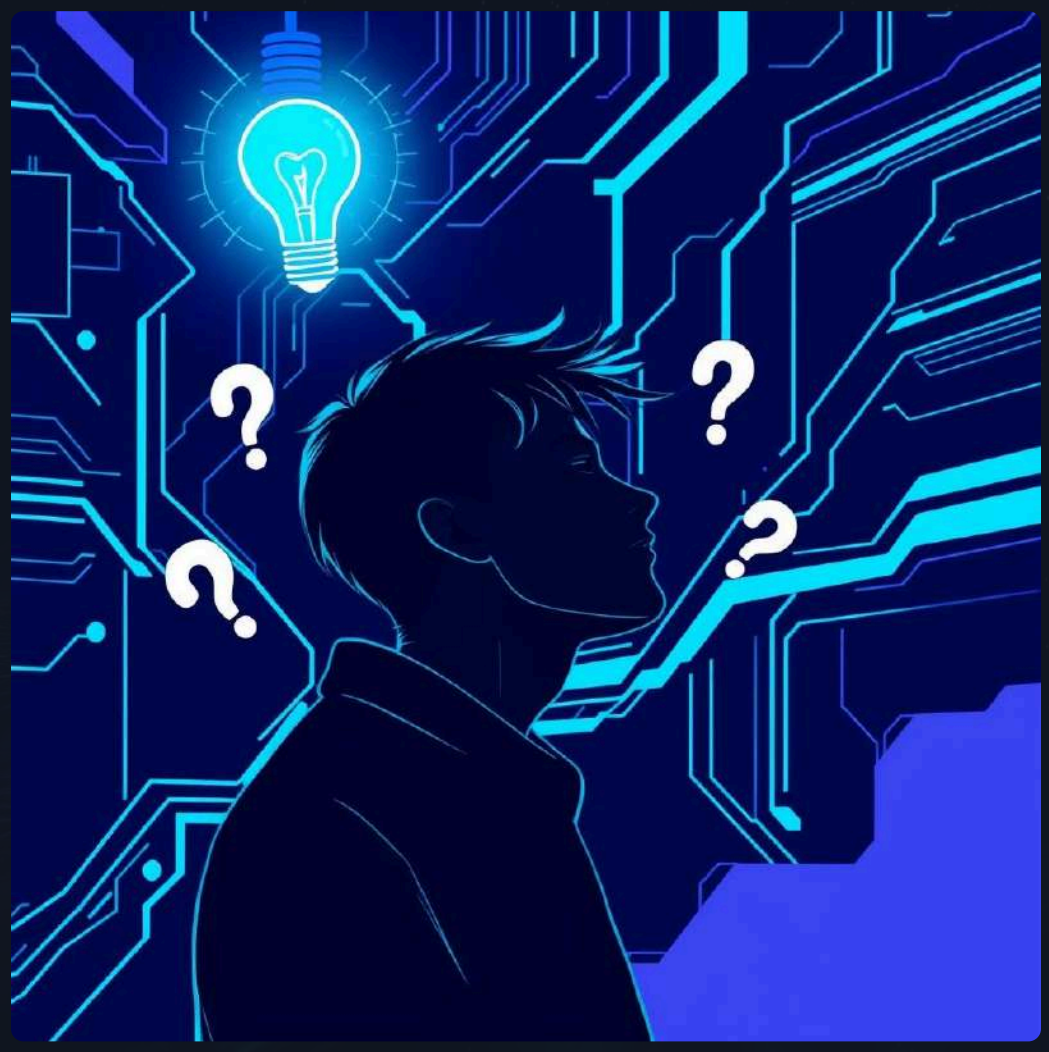
When to use Arrays

- Maximum data size is known.
- Frequent access by index.
- Rare insertions/deletions.
- Maximum performance is required.

Review Questions and Homework

Review Questions

- What is an array and what are its characteristics?
- How to calculate the address of an array element?
- Time complexity of accessing an element?
- Difference between a one-dimensional and a two-dimensional array?
- Why is traversing a matrix row by row more efficient?



Homework

Student grade processing program:

- Store grades of N students for M subjects.
- Calculate the average score for each student.
- Find the best student and the most difficult subject.
- Display statistics in a table format.