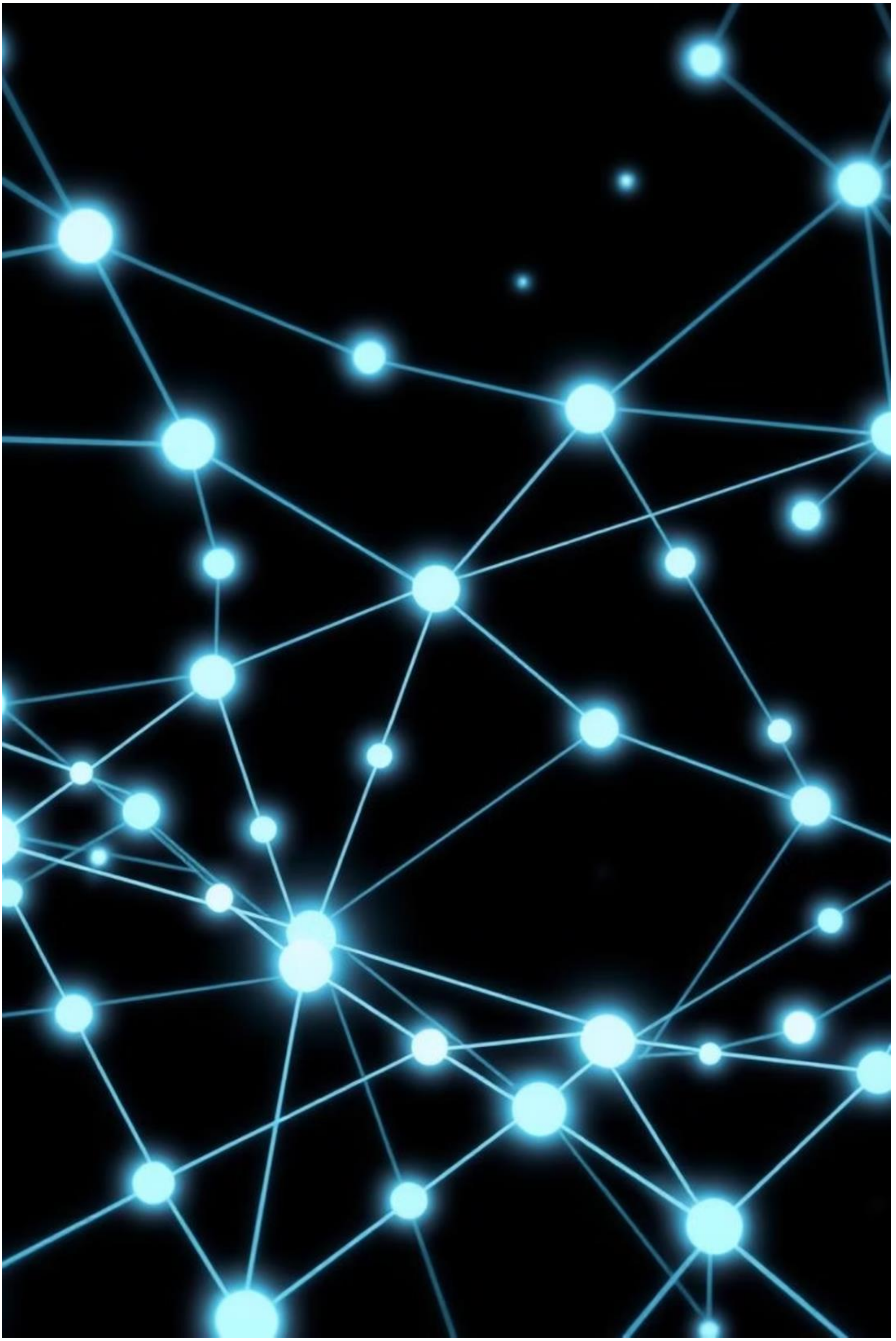


Data Structures

Lecture 3: Array

Instructor:
Md Samsuddoha
Assistant Professor
Dept of CSE, BU

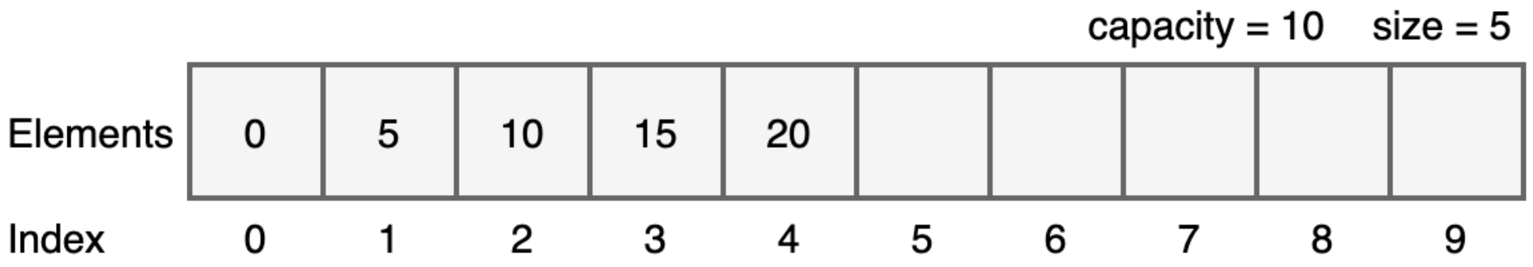
Contents

Arrays: The Contiguous Memory Workhorse

- **Fixed-Size, Contiguous Storage**
  - Arrays store elements in a fixed-size, contiguous block of memory.
  - This adjacency enables direct access to any element using its index in constant time (O(1)).
- **Efficiency Trade-offs**
  - While reads are lightning fast, insertions or deletions in the middle are costly (O(n)) as elements need to be shifted.
  - They are useful for lookup tables, image processing, and serve as the foundation for implementing other complex structures.

Array

- A linear data structure that stores elements **in contiguous memory locations**.
- Each element is identified by an **index** (starting from 0).
- All elements in an array are of the **same data type**.
- Almost all the programs use the Array in some places.
- Supported operations are Insertion, Deletion, Search, and Access by index.



Types of Array

- One Dimensional (Linear Structure)
- 2D Array (Table/Matrix form)
- Multi-dimensional (More than two dimensions)

1D Array (Basic)

- Declaring Array
- Assigning Value with index
- Assigning Value manually
- Assigning Value using Loop
- Print value using Loop
- Find Size of an Array
- Find the sum of all numbers from a given list.
- Find highest or lowest number from a number list.

Operations

Operation	Short Description	Time Complexity	Space Complexity
Accessing	Access the element by its index. A single lookup O(1) is sufficient to access an element from the specific index.	O(1)	O(1)
Searching	Search the element in the Array. A sequential search is required in order to find the element from an unsorted Array.	O(n)	O(1)
Insertion	Insert the new element into the Array. This requires shifting the elements to one position right to accommodate the new element. So, in the worst case, all the elements need to be shifted to insert the element at index 0.	O(n)	O(1)
Deletion	Delete the element by its index. This requires shifting elements to one position left to replace the deleted element. So, in the worst case, all the elements need to be shifted to delete the element at index 0.	O(n)	O(1)

Array Exercises

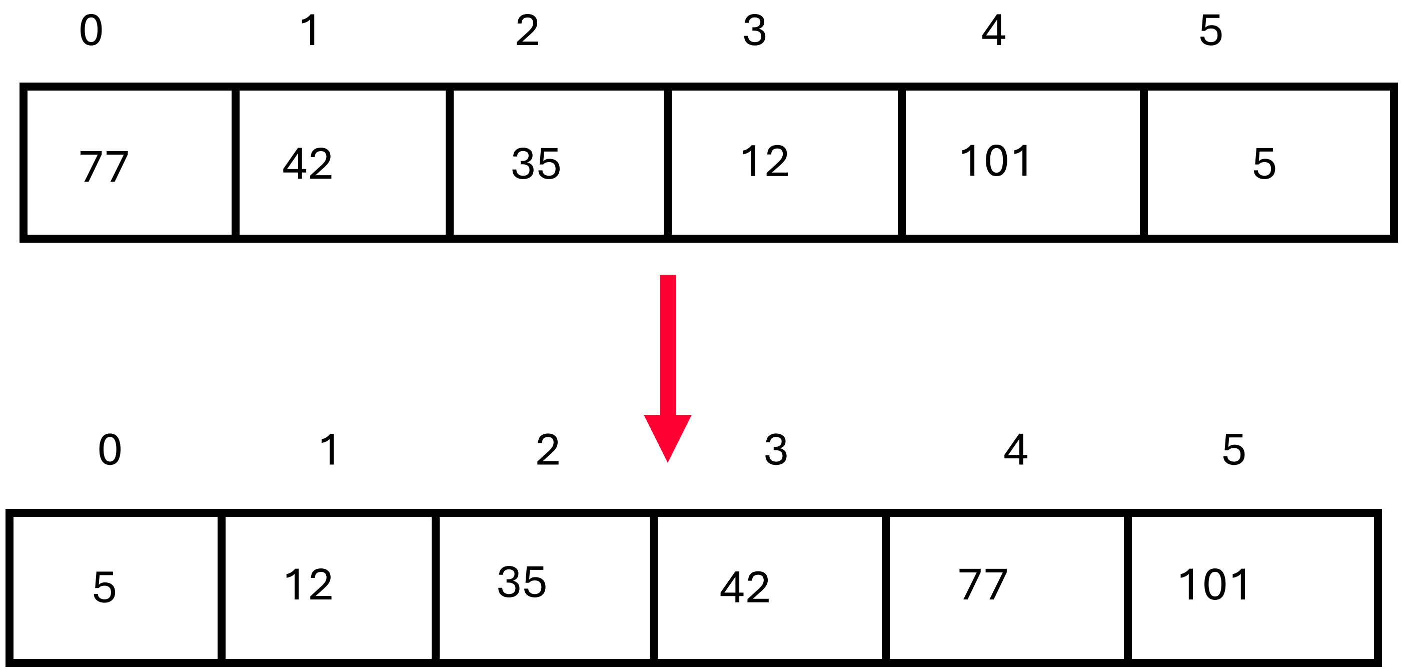
- Calculate the sum and average of elements in an array of 10 integers.
- Write a program to find the largest and smallest number in an array.
- Count how many even and odd numbers are there in an array.
- Take an array and a number from the user, and check if that number exists in the array.
- Add a new element at the end of an array.
- Insert an element at a specific position in an array.
- Remove an element from an array by its index.
- Remove an element from an array by its value.
- Sort an array in ascending or descending order.
- Remove duplicate elements from an array.
- Find the second largest number in the array.
- Write a program to merge two arrays into one.
- Write a program to reverse the elements of an array.
- Find how many times each element appears in an array.

Searching

- Search array for a key value
- Linear search
    - Compare each element of array with key value
    - Useful for small and unsorted arrays
    - Time Complexity is O(n) & Space Complexity is: O(1), No extra space is used.
- Binary search
    - Can only be used on sorted arrays
    - Compares middle element with key
        - If equal, match found
        - If key < middle, repeat search through the first half of the array
        - If key > middle, repeat search through the last half of the array
    - Time Complexity : O(log n)
    - Space Complexity:
        - Iterative Version: O(1)
        - Recursive Version: O(log n) → due to recursion call stack

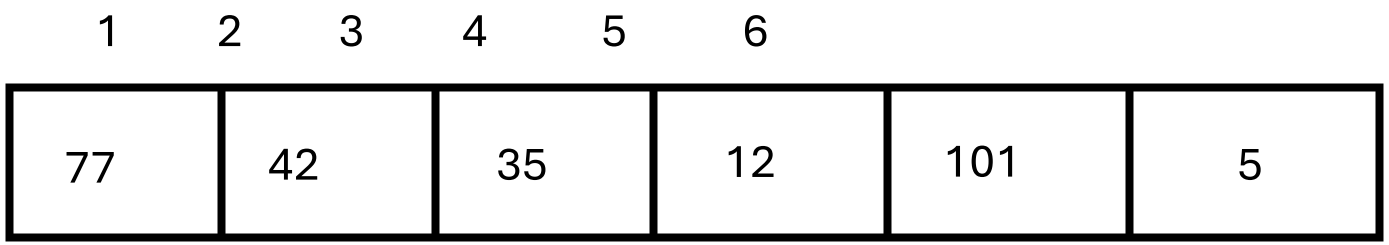
Sorting

- **Sorting takes an unordered collection and makes it an ordered one.**



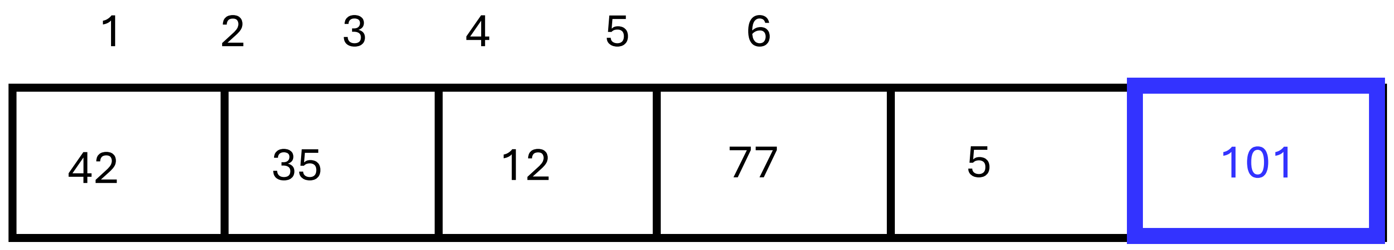
Bubble Sort: "Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



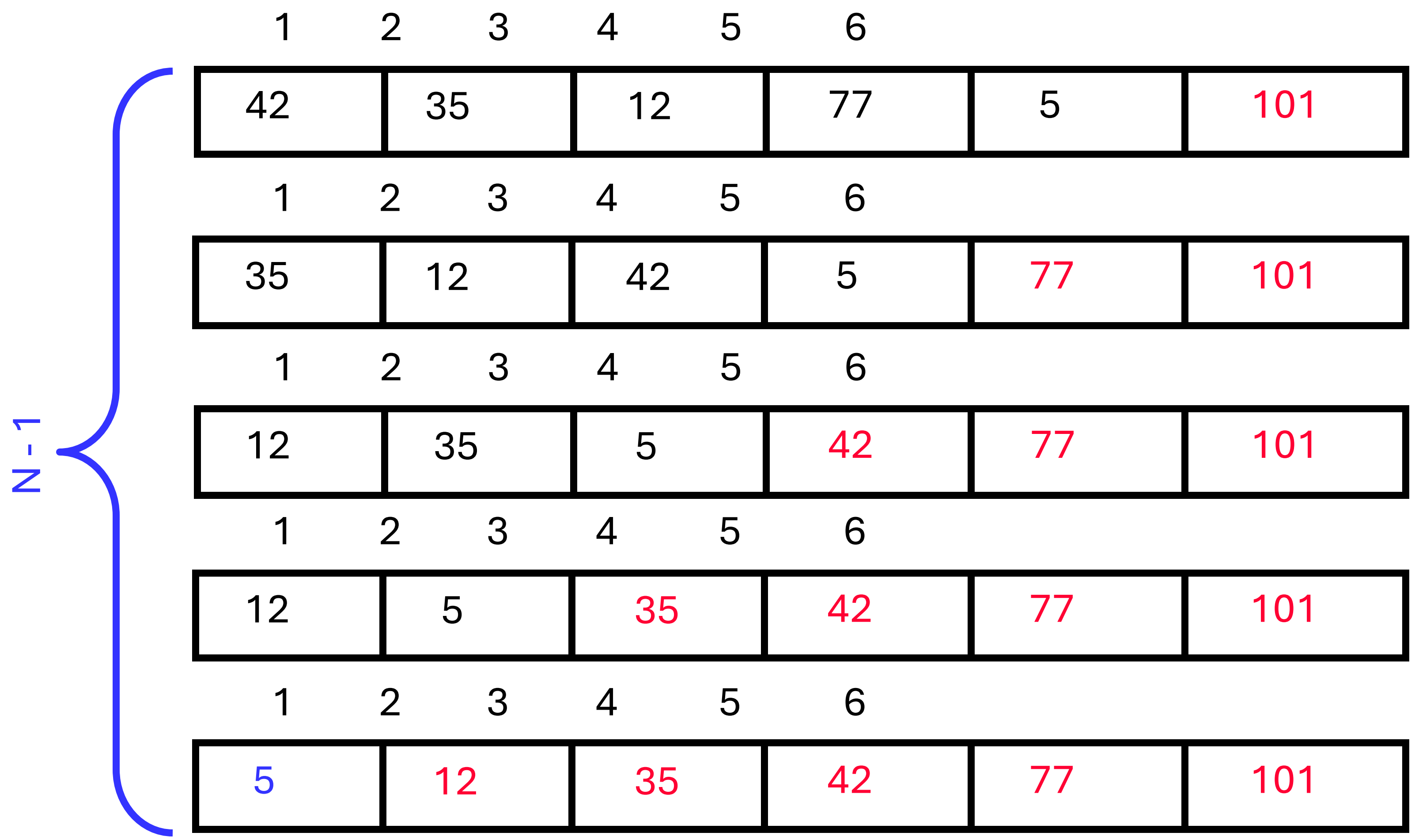
"Bubbling Up" the Largest Element

- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **"Bubble" the largest value to the end using pair-wise comparisons and swapping**

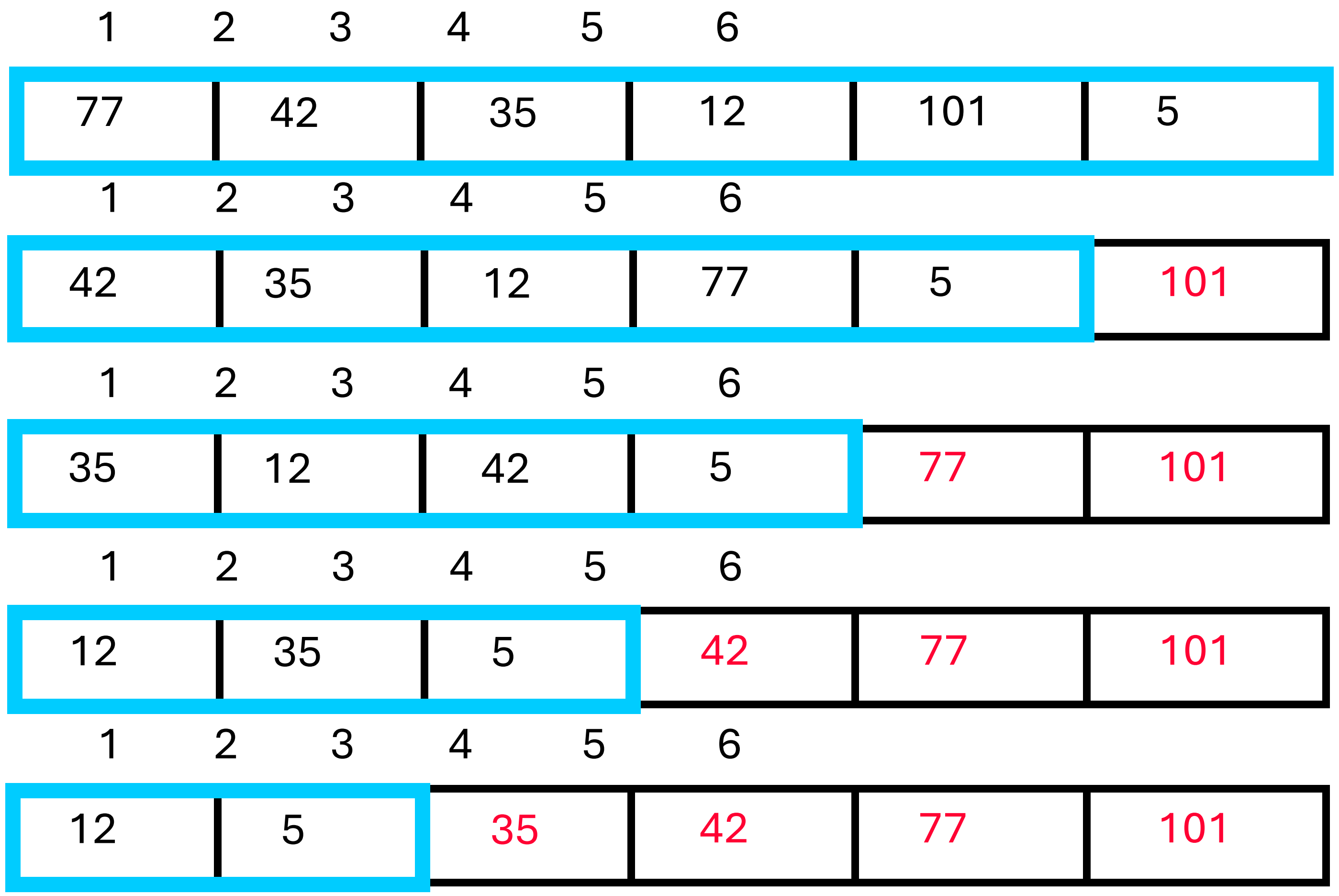


Largest value correctly placed

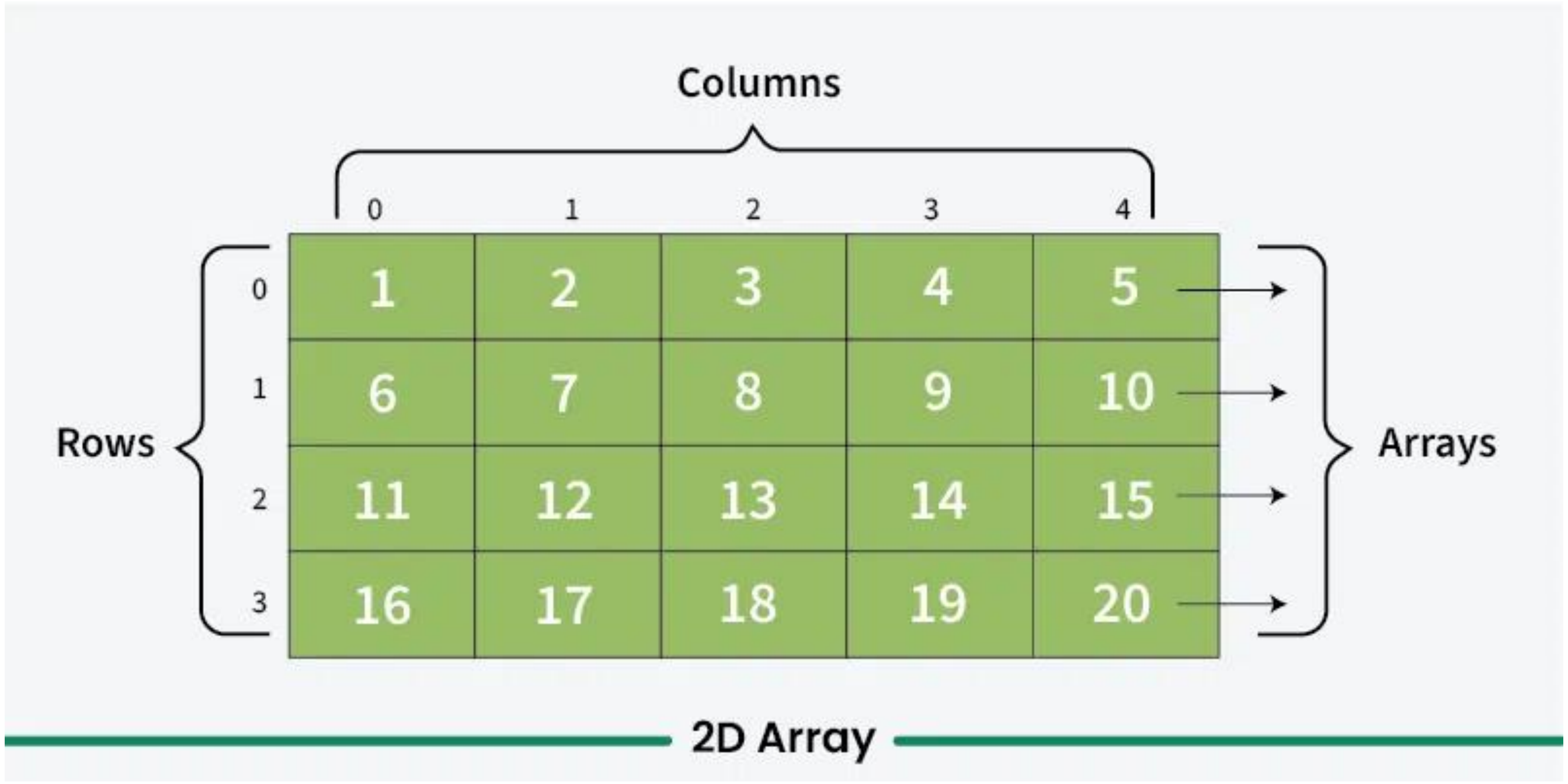
“Bubbling” All the Elements



Reducing the Number of Comparisons



2D Array


Columns
0 1 2 3 4
Rows
0 1 2 3
1 2 3 4 5
6 7 8 9 10
11 12 13 14 15
16 17 18 19 20
Arrays
2D Array

2D Array

- **Concepts**
  - A 2-dimensional array can be visualized as a grid or a table, with data organized in rows and columns. It's an array where each element is itself an array.
- **Declaration**

```
int matrix[3][4]; //Declares a 3x4 integer matrix
```

- **Operations**
  - Accessing elements: matrix[row][column]
  - Initializing: Assigning values to specific cells or using nested loops.
  - Traversing: Iterating through rows and columns using nested loops.
- **Examples**
  - Used in image processing (pixel data), game boards (tic-tac-toe), and spreadsheet applications.

Operations

- Addition
- Multiplication
- Subtraction
- Transpose
- Comparison

Exercises

- Determine whether a given matrix is an identity matrix or not.
- A matrix is symmetric if it equals its transpose. Write a program to check this.
- Find the sum of the main diagonal and the secondary (anti) diagonal of a square matrix.
- Search for a specific value in a matrix. Print its position if found.
- Write a program to compute the sum of upper and lower triangles of a square matrix.

References

- **<u>Chapter 4:</u>**
  - **Data Structures using C** by E. Balagurusamy

Thank You