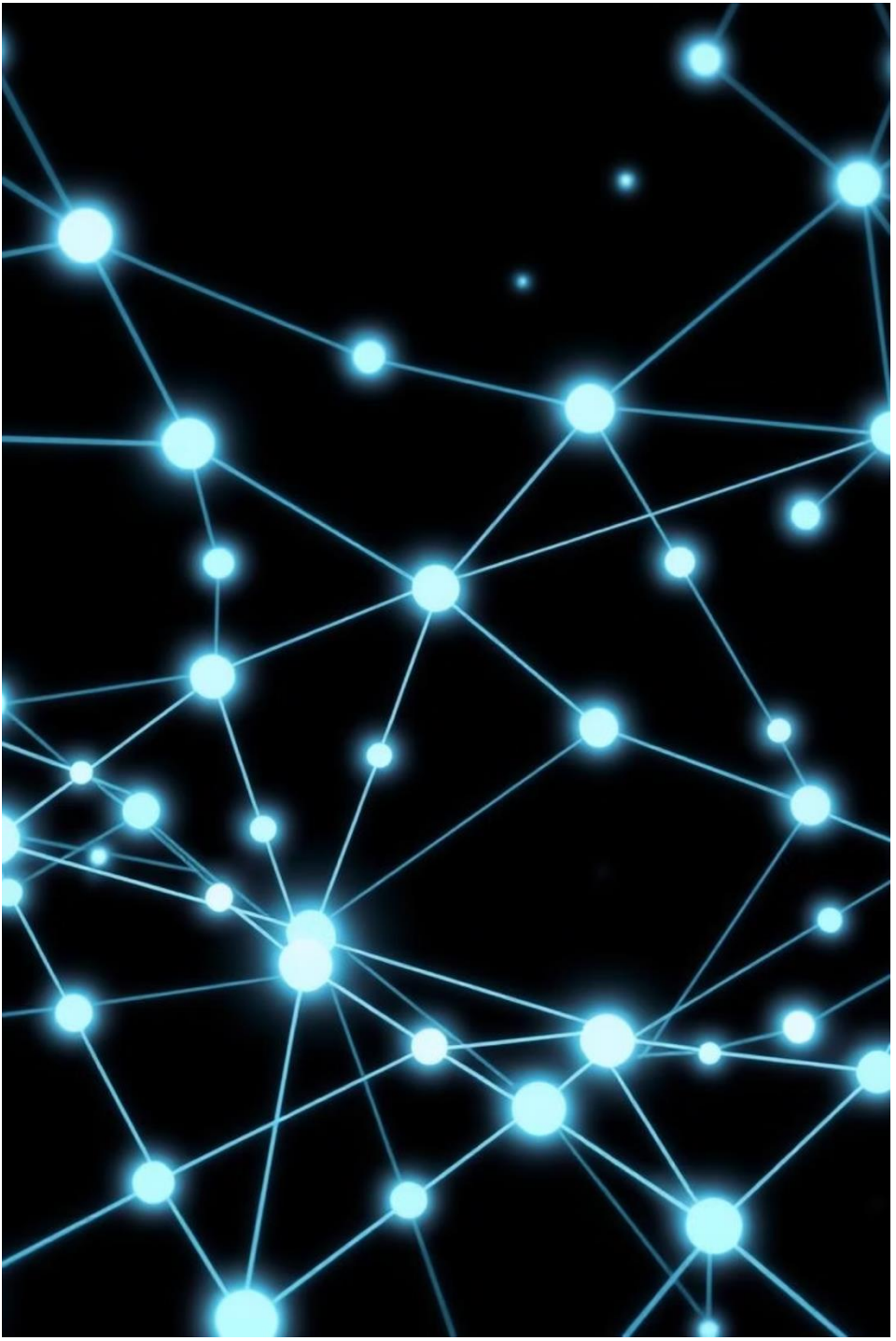


Data Structures

Lecture 8: Searching & Sorting

Instructor:
Md Samsuddoha
Assistant Professor
Dept of CSE, BU

Contents

- Searching (Linear & Binary)
- Sorting (Bubble sort)
- Complexity Analysis
- Pseudocode
- Exercises
- Homework

Searching in Data Structure

- Searching in data structures involves finding the ***location of a specific element or value*** within a collection.
- It is a fundamental operation that can greatly influence data handling efficiency in applications like ***databases and search engines.***
- ***Searching Algorithms***
  - Linear Search
  - Binary Search
  - Ternary Search
  - Jump Search
  - Interpolation Search
  - Fibonacci Search
  - Exponential Search

Linear & Binary Search

- 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

Binary Search

A=[5,7,8,10,13,15,20,25,37,45,50]

- N=11
- LeftIndex=0
- RightIndex=N-1
- ***MidIndex=(LeftIndex+RightIndex)/2***
- Repeat While LeftIndex<=RightIndex
  - If(SearchValue==A[MidIndex]
    - Return SearchValue
  - Else if(SeachValue>A[MidIndex])
    - LeftIndex=MidIndex+1
  - Else
    - RightIndex=MidIndex-1

- Find: 45
- Iterations Table

LeftIndex	RightIndex	MidIndex
0	10	5
6	10	8
9	10	9

- 45 found in 9 index
- Complexity: O (log n)

Binary Search (Pseudocode)

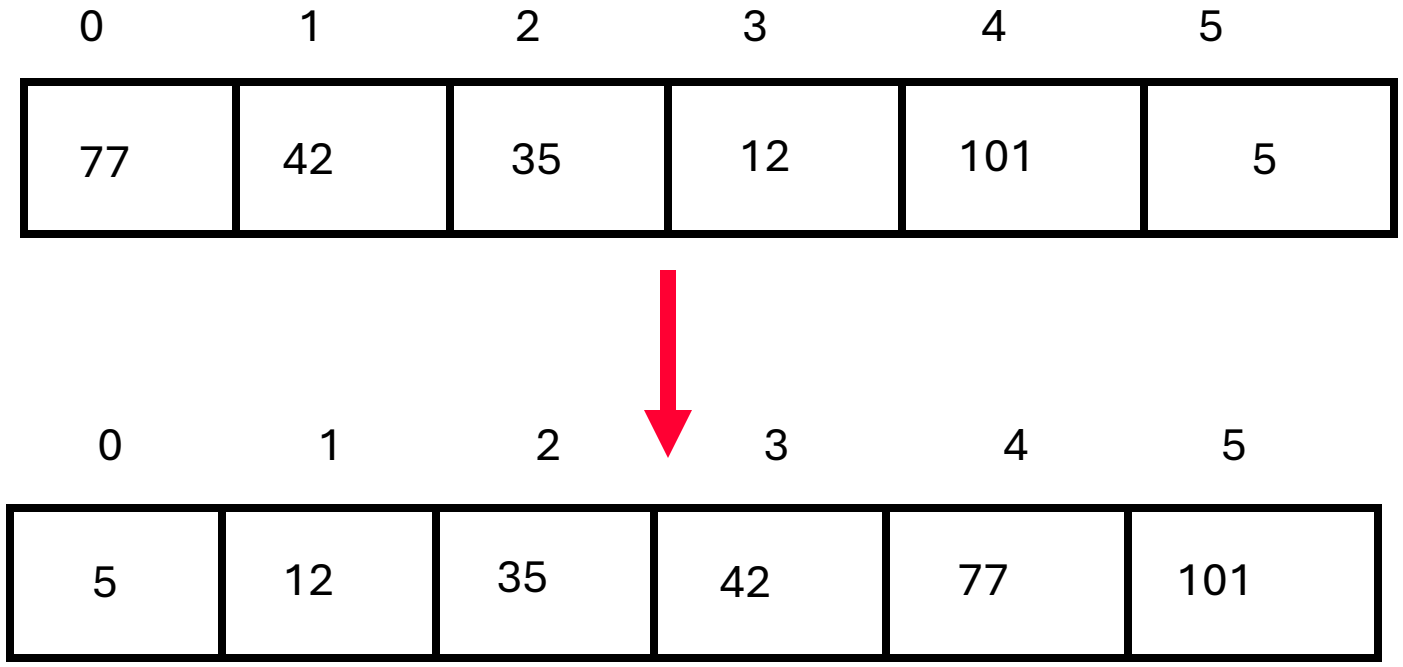
```
BinarySearch(A, n, x):
    low ← 0
    high ← n - 1
    while low ≤ high do
        mid ← (low + high) / 2
        if A[mid] = x then
            return mid
        else if A[mid] < x then
            low ← mid + 1
        else
            high ← mid - 1
    return -1
```

Sorting

- Sorting in data structures is the process of ***arranging a collection of elements*** in a specific order.
- This order can be numerical (ascending or descending), alphabetical, chronological, or based on any other defined criterion.
- The primary goal of sorting is to organize data in a way that facilitates more efficient ***searching, retrieval, and manipulation***.

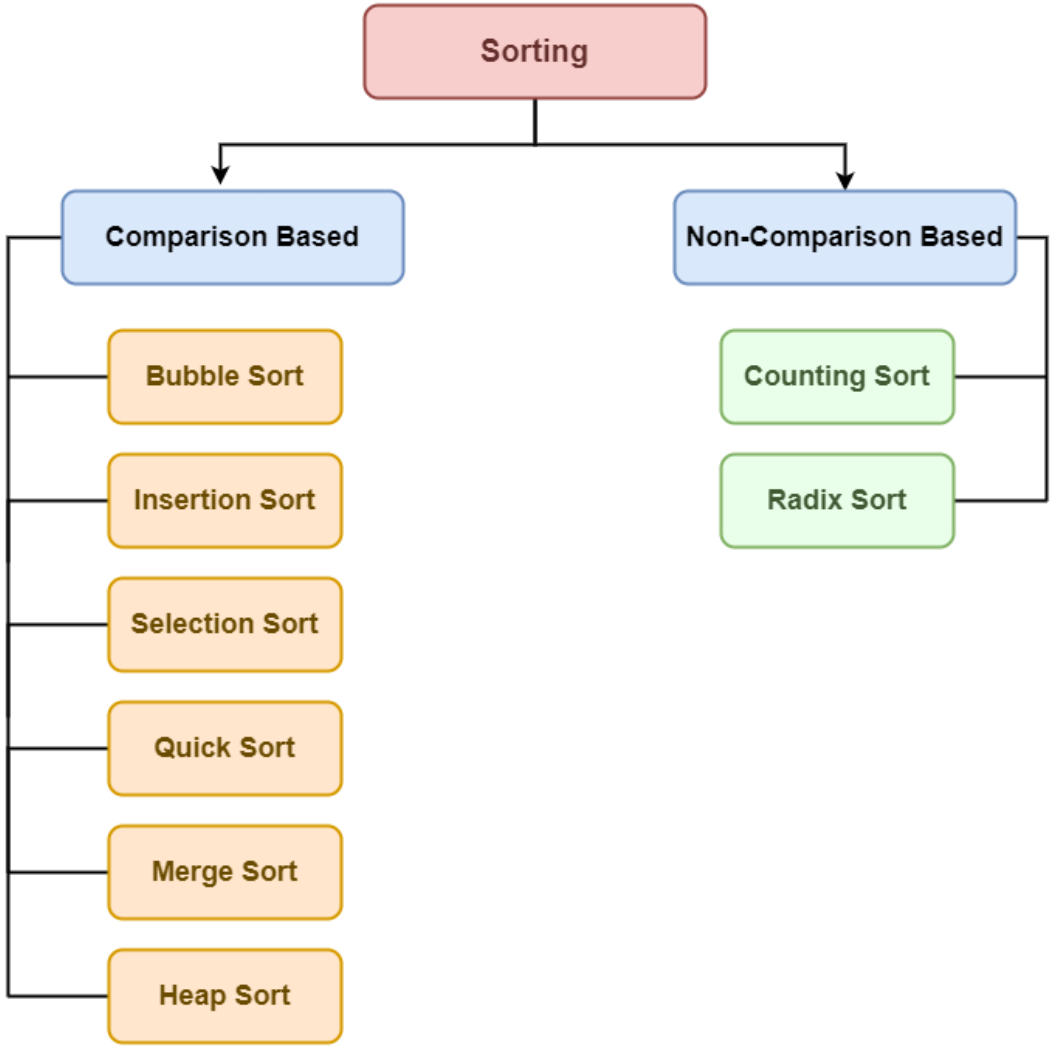
Sorting

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



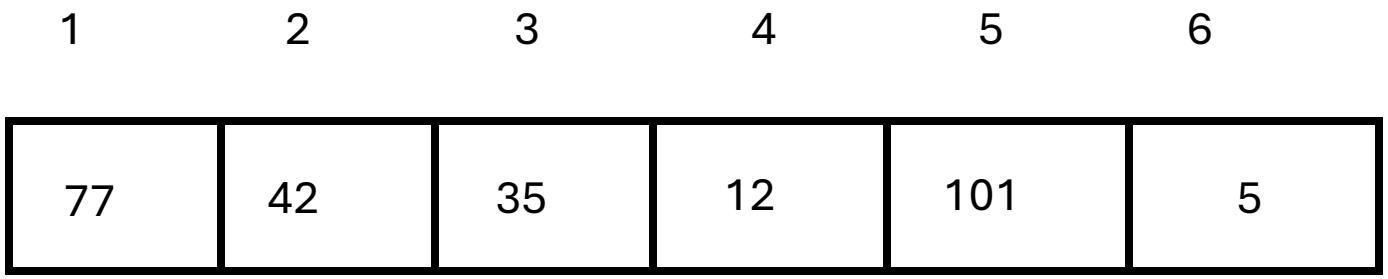
Sorting Algorithms

- Bubble Sort
- Selection Sort
- Insertion Sort
- Merge Sort
- Quick Sort
- Heap Sort
- *Counting Sort*
- *Radix Sort*
- *Bucket Sort*



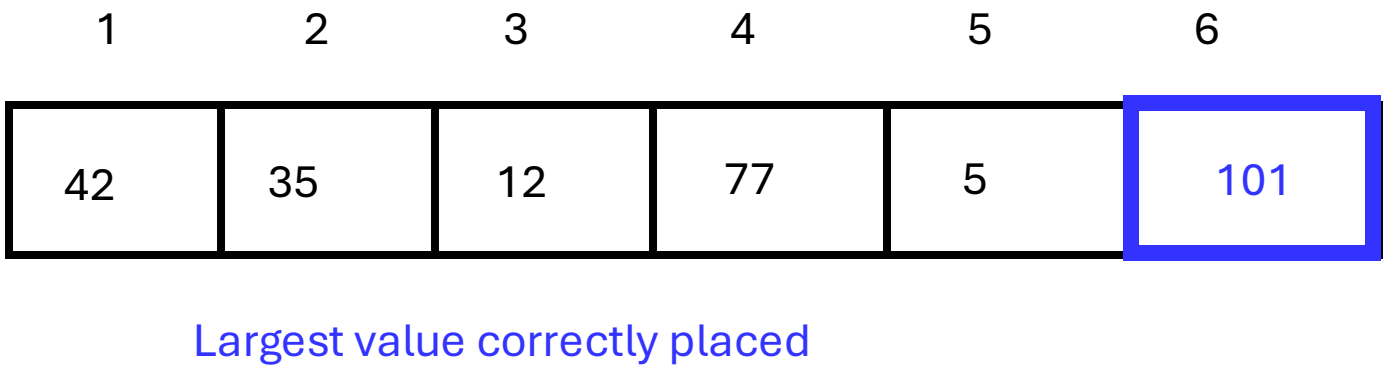
Bubble Sort: "Bubbling Up" the Largest Element

- **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**

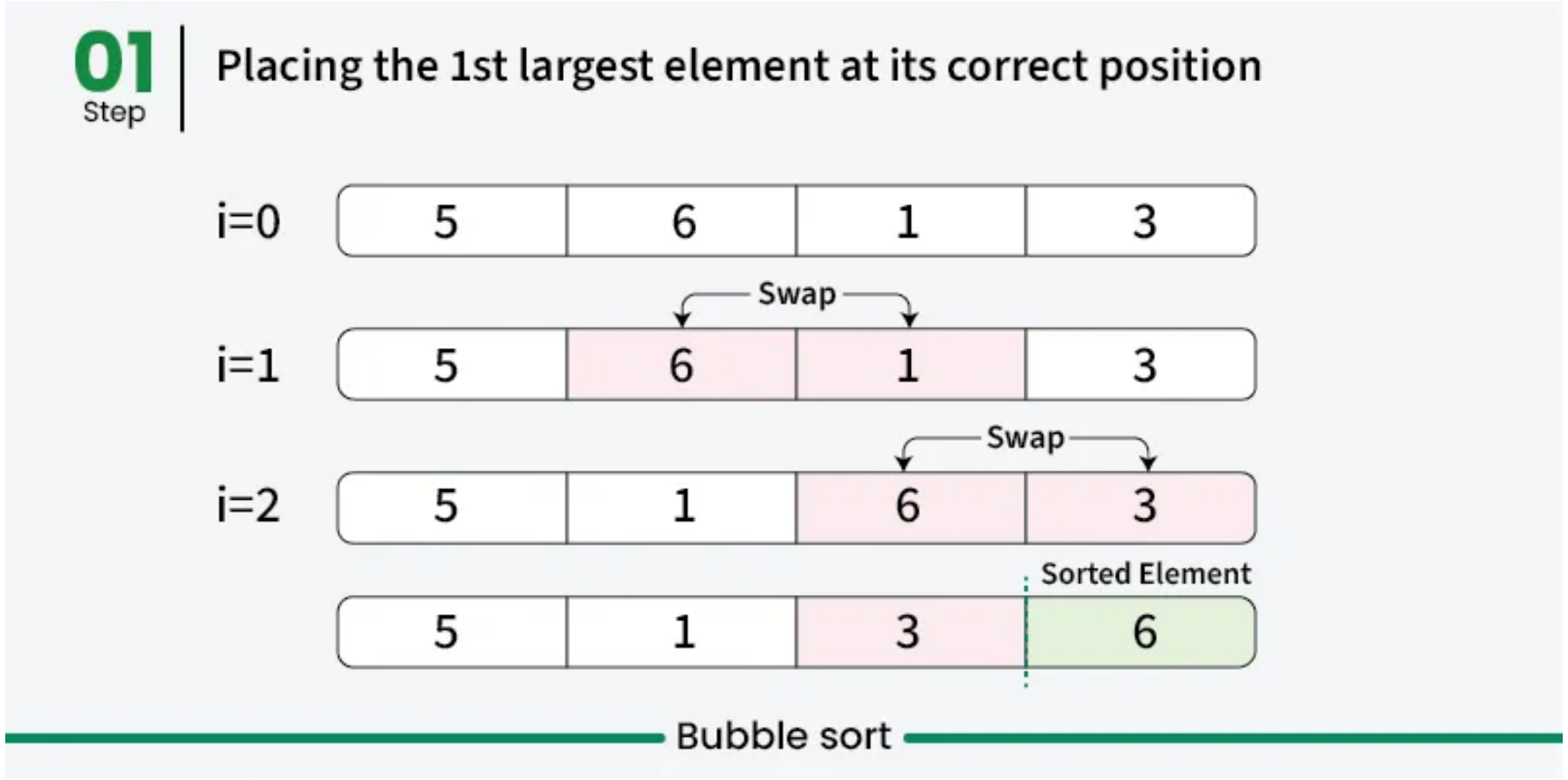


"Bubbling Up" the Largest Element

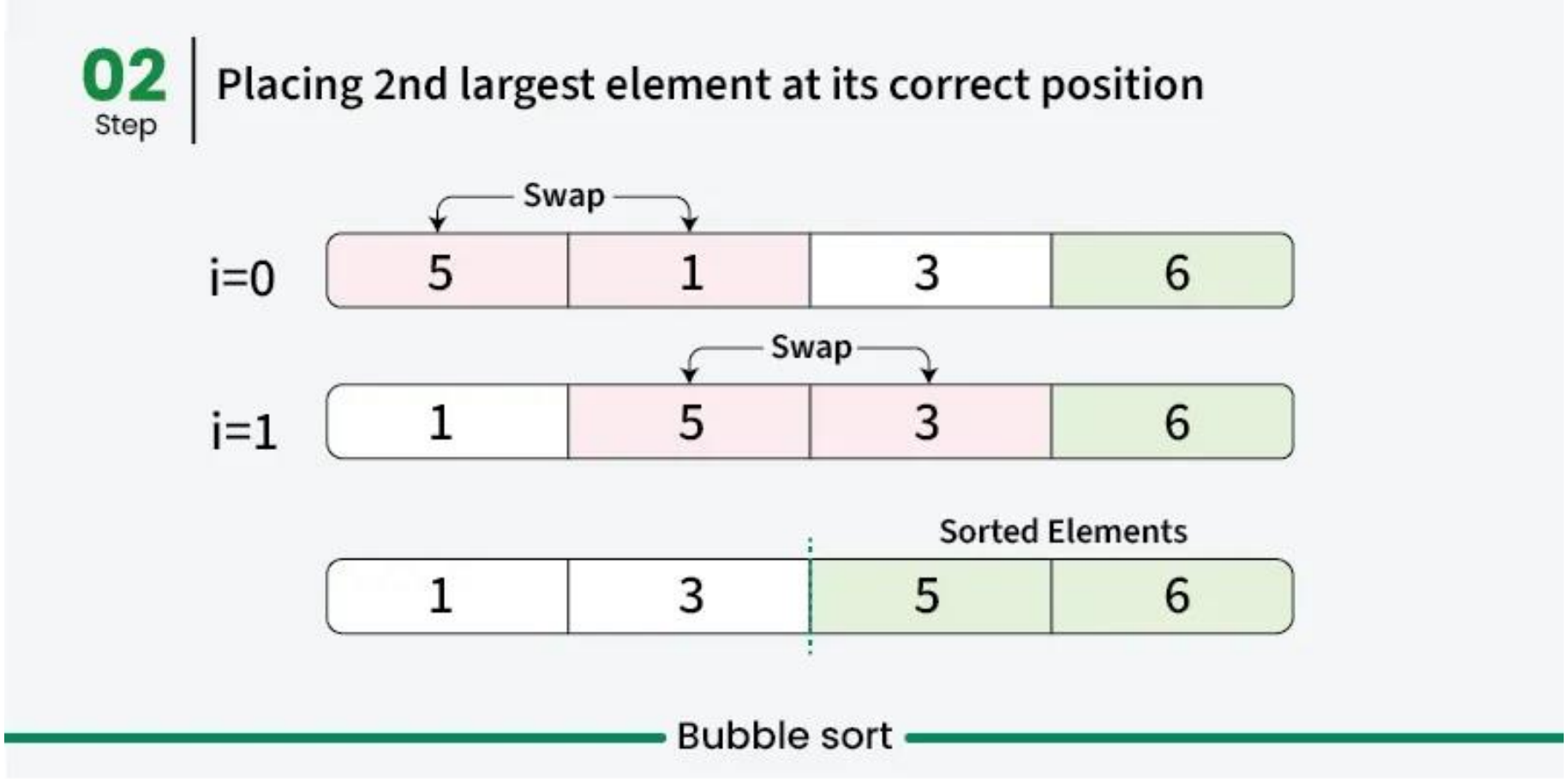
- **Traverse a collection of elements**
  - **Move from the front to the end**
  - **“Bubble” the largest value to the end using pair-wise comparisons and swapping**



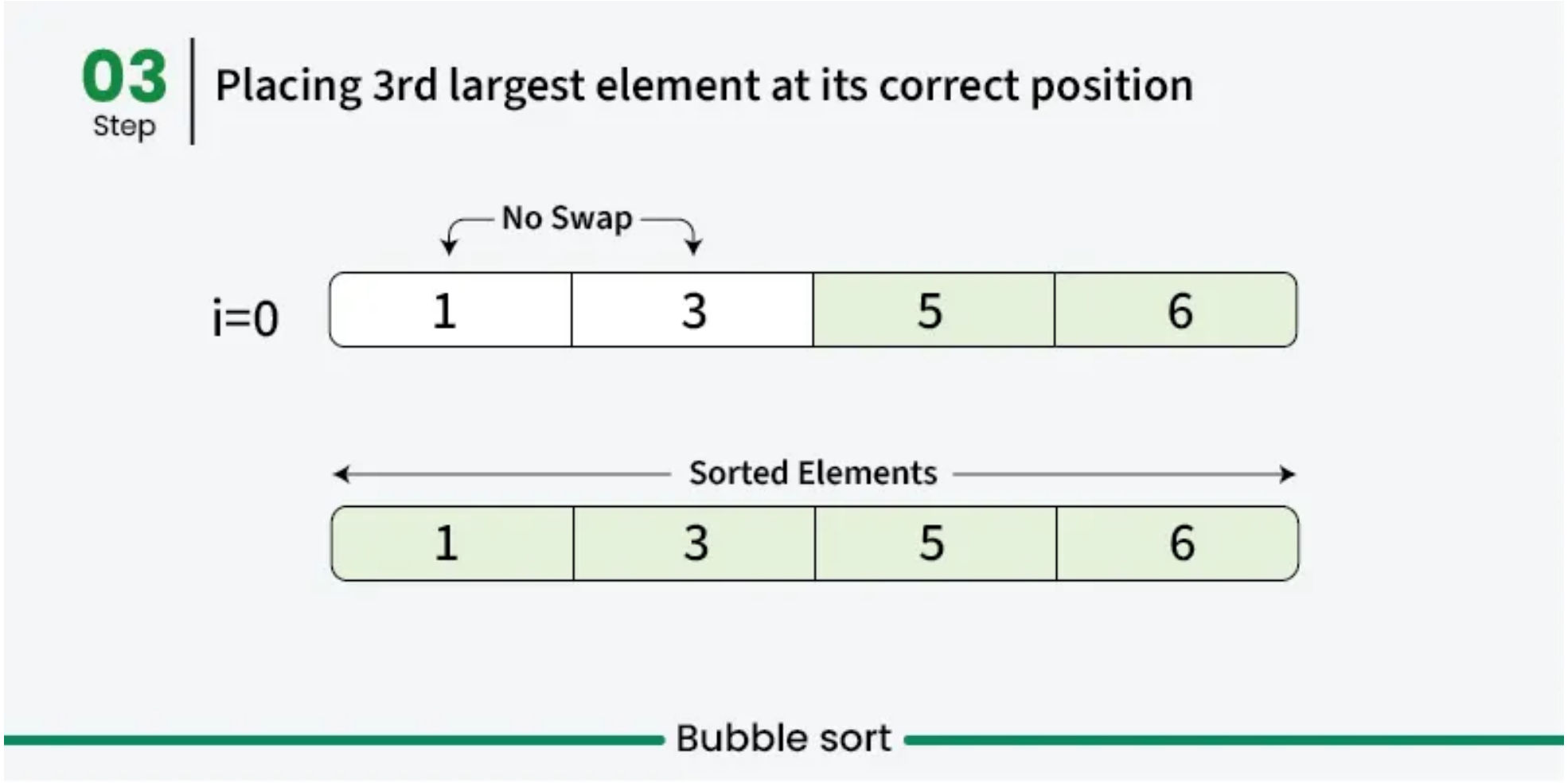
Bubble Sort (Simulation)



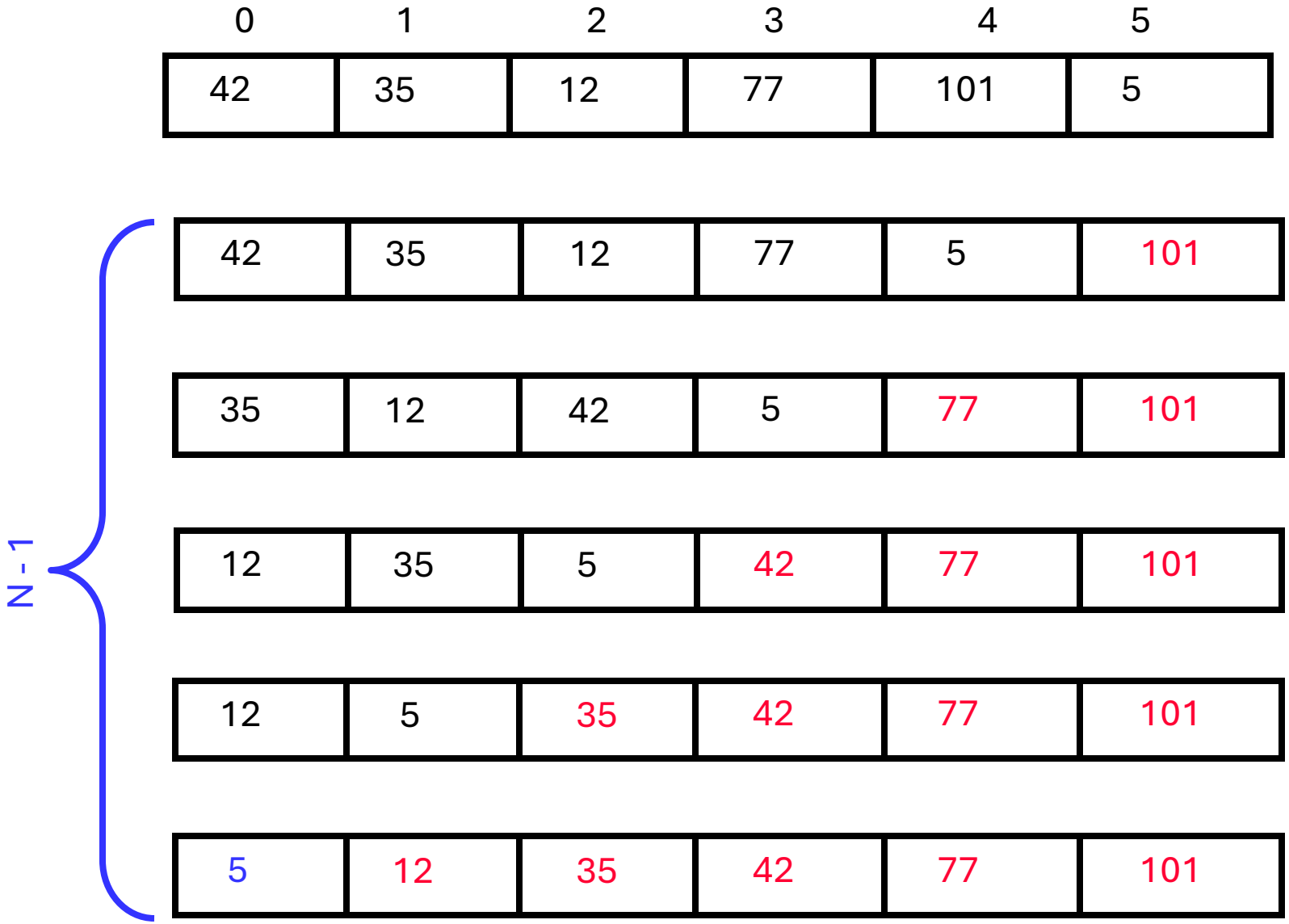
Bubble Sort (Simulation)



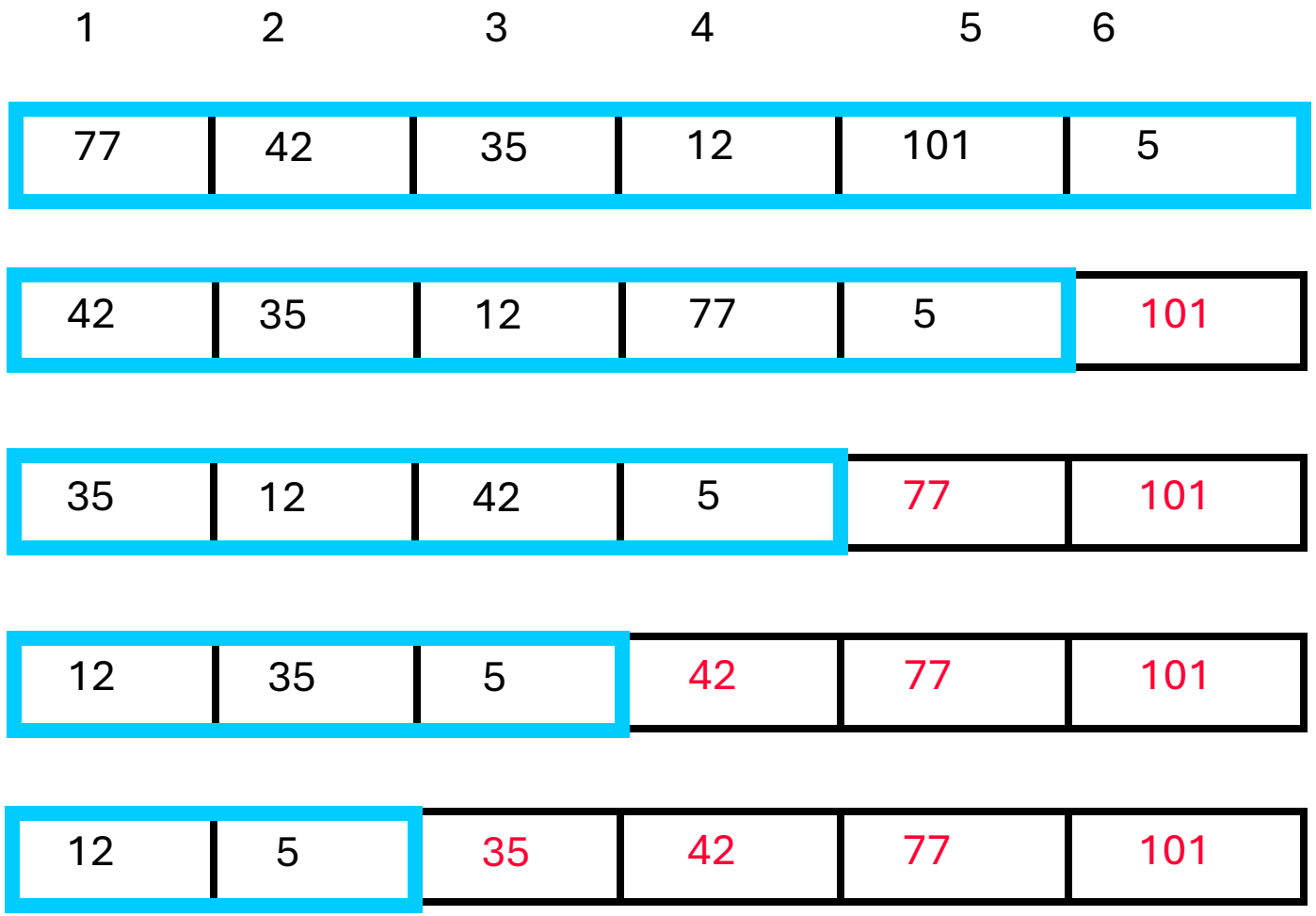
Bubble Sort (Simulation)



“Bubbling” All the Elements



Reducing the Number of Comparisons



Complexity

- Worst case: array is in **reverse order**.
- Number of Iterations: N-1
- Number of comparisons in each pass:
  - 1st pass → n-1 comparisons
  - 2nd pass → n-2 comparisons
  - …
  - Last pass → 1 comparison
- Total comparisons = (n-1) + (n-2) + … + 1 = n(n-1)/2
- **Worst-case time complexity:** $O(n^2)$

Pseudocode

```
Algorithm BubbleSort(A, n)
Input: Array A of size n
Output: Sorted array A in ascending order

1. for i = 0 to n-1 do
2.     flag = 0
3.     for j = 0 to n-i-2 do
4.         if A[j] > A[j+1] then
5.             // Swap A[j] and A[j+1]
6.             temp = A[j]
7.             A[j] = A[j+1]
8.             A[j+1] = temp
9.             flag = 1
10.        end if
11.    end for
12.    // If no elements were swapped, array is already sorted
13.    if (!flag then
14.        break
15.    end if
16. end for
17. return A
```

Exercises

- Write a C program to find whether a given number exists in an array of **N** integers using **linear search**. Print the index if found, else print “Not found.”
- Modify the linear search program to count how many times a given number appears in the array.
- You have a list of student roll numbers. Write a program to check if a new student’s roll number already exists in the list using linear search.
- Given a sorted array of **N** integers, write a program to search for a number using **binary search**. Print its index if found.
- Modify the binary search program to find the first occurrence of a repeated element in a sorted array.
- Write a binary search program that also counts how many comparisons are performed during the search.
- Write a program to sort an array of **N** integers in **ascending order** using **bubble sort**.
- Modify the bubble sort program to sort the array in **descending order**.
- Write a program for bubble sort to count the number of swaps needed to sort the array.
- You have a list of students with roll numbers and scores. You have to find the student with the highest score and display their roll number and score.

References

- **Chapter 10: Data Structures using C** by E. Balagurusamy

Thank You