Table of Contents

Table of Figures

Table of Tables

Task 01 – Algorithm Recording

1.1 Series of Instructions (Step-by-Step Algorithm)

The goal of this algorithm is to process multiple sensor readings (each being an 8×8 matrix of plant heights), determine the tallest plant in each reading, and finally calculate the average of these maximum heights.

Input

- A number of readings (N)
- Each reading is an 8×8 matrix containing plant heights

Process

1. Start the program.
2. Ask the user to enter the number of readings (N).
3. Create an empty list called ‘MaxValues’ to store the tallest plant from each reading.
4. Repeat the following steps for each reading (from 1 to N):
    a. Input the 8×8 matrix of plant heights.
    b. Set ‘max_height’ to the first value in the matrix.
    c. Examine each of the 64 plant heights:
        i. If the current height is greater than ‘max_height’, update ‘max_height’.
    d. Store ‘max_height’ in the list ‘MaxValues’.
5. After processing all readings:
    a. Calculate the average of all values in ‘MaxValues’.
    b. Display the average maximum plant height.
6. End the program.

Output

- The average of the maximum heights from all readings.

1.2 Structure Chart

A structure chart is a hierarchical representation of modules, showing the logical breakdown of the process.

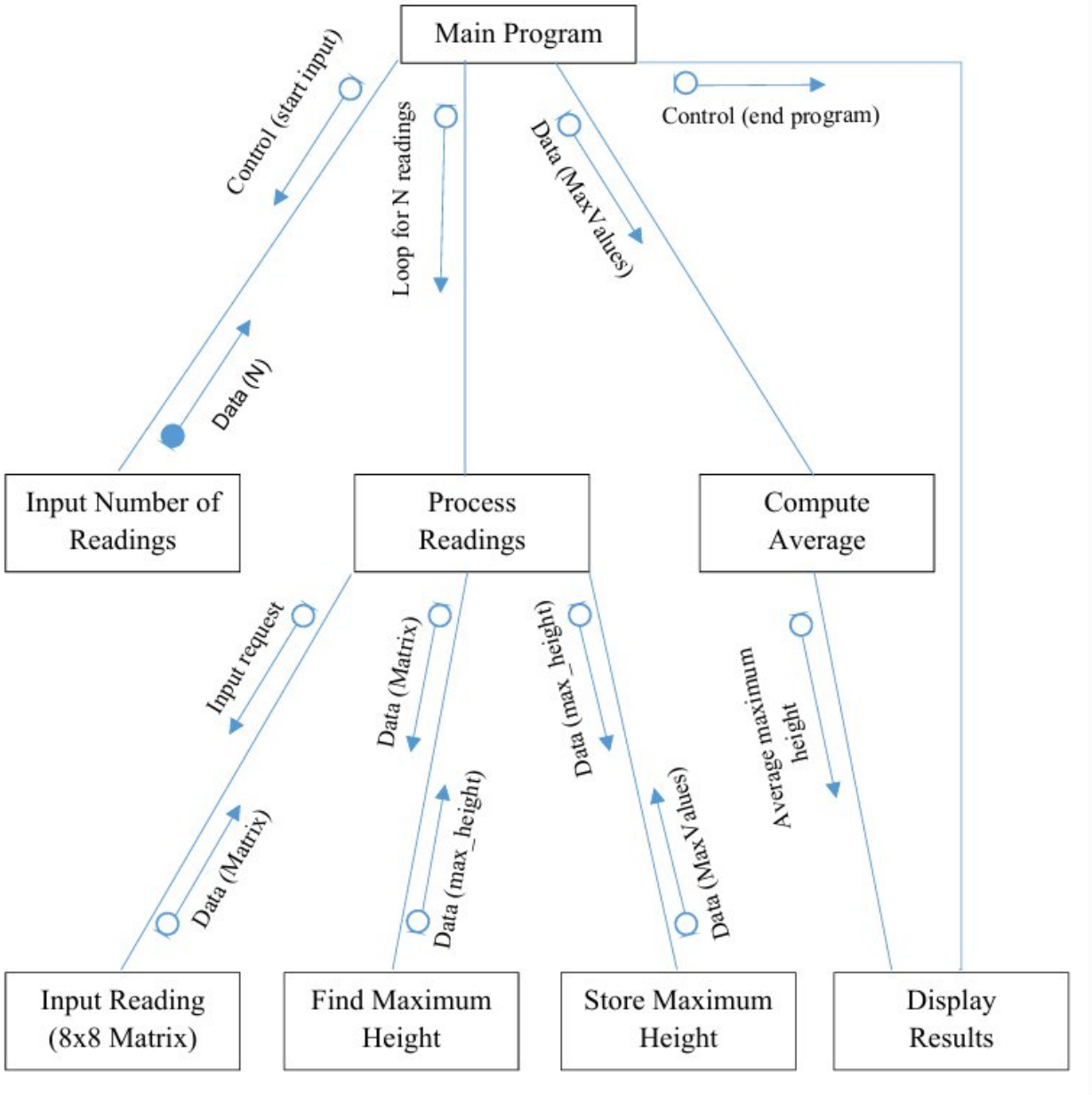

Figure 1 Structure chart

1.3 Flowchart

The flowchart below describes the step-by-step flow of the algorithm.

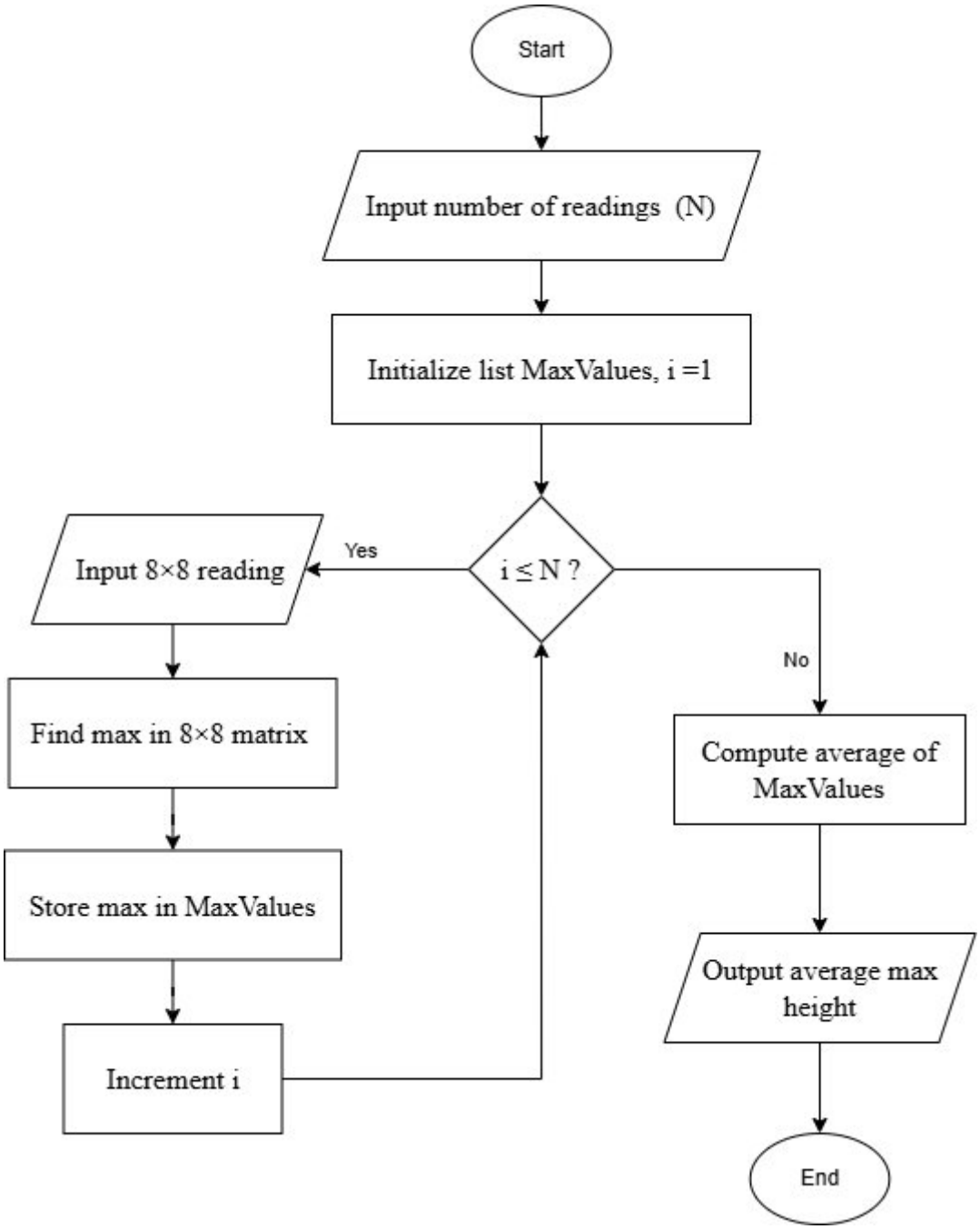

Figure 2 Flowchart

The process begins with the **Start** of the program. The user is prompted to enter the number of readings (**N**). An empty list called 'MaxValues' is initialized to store the maximum height from each matrix, and a counter 'i' is set to 1.

The program then enters a loop that continues as long as the condition ‘i ≤ N’ is true, indicating that more readings are to be processed. If this condition is true, the user is asked to input an 8×8 matrix of plant heights. The algorithm scans all 64 values in the matrix to find the maximum height, which is then added to the ‘MaxValues’ list. Afterward, the counter ‘i’ is incremented by 1, and control returns to the loop condition to process the next reading.

Once the condition is false (i.e., all readings have been processed), the program calculates the average of the values stored in ‘MaxValues’ and displays this result: the average maximum plant height. The program then reaches the **End** and terminates.

Task 02 – Pseudocode and Python Implementation

2.1 Pseudocode

```
BEGIN
    INPUT number_of_readings (N)
    CREATE empty list MaxValues

    FOR reading_index = 1 TO N DO
        INPUT Reading[8][8]
        SET max_height = Reading[0][0]

        FOR row = 0 TO 7 DO
            FOR col = 0 TO 7 DO
                IF Reading[row][col] > max_height THEN
                    SET max_height = Reading[row][col]
                ENDIF
            ENDFOR
        ENDFOR

        APPEND max_height TO MaxValues
    ENDFOR

    SET total = 0
    FOR i = 0 TO LENGTH(MaxValues) - 1 DO
        total = total + MaxValues[i]
    ENDFOR

    average = total / LENGTH(MaxValues)

    OUTPUT "Average of maximum heights = ", average
END
```

Explanation:

The pseudocode provides a clear and systematic method for calculating the average maximum plant height from multiple sensor readings. It begins by prompting the user to input the total number of readings, referred to as 'N', that will be analyzed. An empty list, 'MaxValues', is created to store the highest plant height found in each of these readings.

For each reading, the program accepts an 8×8 matrix of plant heights. It initializes a variable called 'max_height' with the first value from this matrix, establishing a baseline for comparison.

Using nested loops, the algorithm traverses all the heights in the matrix. For each value encountered, it checks if that value exceeds the current 'max_height'. If it does, the 'max_height' variable is updated to this new maximum value. By the end of this traversal, 'max_height' holds the tallest plant height for the current reading.

This maximum value is then appended to the 'MaxValues' list. Once all readings are processed and their maxima collected, the algorithm calculates the total sum of all maximum heights. Finally, the average maximum height is determined by dividing this total by the number of readings. The algorithm outputs this average, representing the overall tallest plant height observed across all sensor data.

2.2 Python Implementation

main.py

```python
def calculate_average_of_max_heights(readings):
    """
    readings: list of 2D lists (each 8x8 matrix of plant heights)
    returns: average of maximum heights
    """
    max_values = []

    for reading in readings:
        max_height = reading[0][0]
        for row in reading:
            for value in row:
                if value > max_height:
                    max_height = value
        max_values.append(max_height)

    # Compute average
    average = sum(max_values) / len(max_values)
    return average


# ---- Example Run using Table 1 ----
reading1 = [
    [0.67, 0.27, 0.67, 3.04, 0.5, 1.25, 0.37, 1.56],
    [0.13, 1.6, 1.23, 2.11, 3.11, 0.11, 1.56, 2.56],
    [0.33, 3.2, 2.1, 2.24, 3.0, 1.2, 1.44, 2.1],
    [0.88, 0.82, 0.81, 0.74, 0.8, 0.9, 1.22, 2.2],
    [1.56, 1.56, 1.56, 1.56, 3.2, 2.1, 2.24, 3.0],
    [2.56, 2.56, 2.56, 2.56, 0.81, 0.74, 0.8, 0.9],
    [2.1, 2.1, 2.1, 2.1, 1.56, 1.56, 3.2, 2.1],
    [2.2, 2.2, 2.2, 2.2, 3.2, 2.1, 2.24, 3.0]
]

# Suppose we only have 1 reading
all_readings = [reading1]

average = calculate_average_of_max_heights(all_readings)
print("Average of maximum heights =", average)
```

```
Average of maximum heights = 3.2


...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 3 Python implementation code for single reading

main.py

```python
def calculate_average_of_max_heights(readings):
    max_values = []
    for reading in readings:
        max_height = reading[0][0]
        for row in reading:
            for value in row:
                if value > max_height:
                    max_height = value
        max_values.append(max_height)
    return sum(max_values) / len(max_values)


# ---- Example Run ----
reading1 = [
    [0.67, 0.27, 0.67, 3.04, 0.5, 1.25, 0.37, 1.56],
    [0.13, 1.6, 1.23, 2.11, 3.11, 0.11, 1.56, 2.56],
    [0.33, 3.2, 2.1, 2.24, 3.0, 1.2, 1.44, 2.1],
    [0.88, 0.82, 0.81, 0.74, 0.8, 0.9, 1.22, 2.2],
    [1.56, 1.56, 1.56, 1.56, 3.2, 2.1, 2.24, 3.0],
    [2.56, 2.56, 2.56, 2.56, 0.81, 0.74, 0.8, 0.9],
    [2.1, 2.1, 2.1, 2.1, 1.56, 1.56, 3.2, 2.1],
    [2.2, 2.2, 2.2, 2.2, 3.2, 2.1, 2.24, 3.0]
]

reading2 = [
    [0.5, 1.2, 2.3, 0.7, 1.5, 0.8, 2.0, 1.1],
    [1.0, 1.8, 0.9, 2.4, 2.8, 1.1, 0.5, 1.6],
    [0.6, 0.7, 1.9, 2.2, 2.1, 1.7, 0.8, 0.9],
    [0.5, 0.6, 1.2, 1.1, 0.8, 1.3, 0.7, 1.0],
    [1.4, 0.9, 0.8, 2.0, 1.5, 1.6, 2.1, 1.9],
    [1.0, 1.1, 1.2, 1.3, 0.9, 1.5, 1.6, 1.7],
    [0.7, 1.0, 1.2, 1.5, 1.8, 2.0, 2.2, 2.3],
    [1.0, 1.3, 1.6, 1.9, 2.1, 2.2, 2.4, 2.5]
]

all_readings = [reading1, reading2]

average = calculate_average_of_max_heights(all_readings)
print("Average of maximum heights =", average)
```

input

```
Average of maximum heights = 3.0


...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 4 Python implementation code for multiple readings

Task 03 – Evaluation of Algorithm

3.1 Restate the Algorithm in Brief

The algorithm is designed to:

1. Accept multiple plant height readings, each structured as an 8×8 matrix.
2. For each reading, scan all 64 height values one by one.
3. Identify the maximum plant height for that reading.
4. Store the maximum values from all readings in a list.
5. After processing all readings, compute the average of the stored maximum heights.
6. Output the final average maximum height.

This approach ensures that the algorithm captures the tallest plant in each individual reading while providing an overall average across all datasets.

3.2 Desk Check (Step-by-Step Trace with Sample Data)

To verify the correctness of the algorithm, a desk check was performed using the provided Table 1 dataset (an 8×8 grid of plant heights).

- Initial value: max_height = 0
- The algorithm sequentially compares each cell's height with the current max_height.
- If a larger value is found, max_height is updated.
- After scanning all 64 cells, the final value of max_height is recorded as the tallest plant in that reading.

Trace Table for Reading 1

Step	Current Cell Value	max_height Before	Condition (cell > max_height?)	max_height After
1	0.67	0.0	Yes	0.67
2	0.27	0.67	No	0.67
3	0.67	0.67	No	0.67
4	3.04	0.67	Yes	3.04
5	0.5	3.04	No	3.04
6	1.25	3.04	No	3.04
7	0.37	3.04	No	3.04
8	1.56	3.04	No	3.04
9	0.13	3.04	No	3.04

10	1.6	3.04	No	3.04
11	1.23	3.04	No	3.04
12	2.11	3.04	No	3.04
13	3.11	3.04	Yes	3.11
14	0.11	3.11	No	3.11
15	1.56	3.11	No	3.11
16	2.56	3.11	No	3.11
17	0.33	3.11	No	3.11
18	3.2	3.11	Yes	3.2
19	2.1	3.2	No	3.2
20	2.24	3.2	No	3.2
21	3.0	3.2	No	3.2
22	1.2	3.2	No	3.2
23	1.44	3.2	No	3.2
24	2.1	3.2	No	3.2
25	0.88	3.2	No	3.2
26	0.82	3.2	No	3.2
27	0.81	3.2	No	3.2
28	0.74	3.2	No	3.2
29	0.8	3.2	No	3.2
30	0.9	3.2	No	3.2
31	1.22	3.2	No	3.2
32	2.2	3.2	No	3.2
33	1.56	3.2	No	3.2
34	1.56	3.2	No	3.2
35	1.56	3.2	No	3.2
36	1.56	3.2	No	3.2
37	3.2	3.2	No	3.2
38	2.1	3.2	No	3.2
39	2.24	3.2	No	3.2
40	3.0	3.2	No	3.2
41	2.56	3.2	No	3.2
42	2.56	3.2	No	3.2
43	2.56	3.2	No	3.2
44	2.56	3.2	No	3.2
45	0.81	3.2	No	3.2
46	0.74	3.2	No	3.2
47	0.8	3.2	No	3.2
48	0.9	3.2	No	3.2
49	2.1	3.2	No	3.2
50	2.1	3.2	No	3.2
51	2.1	3.2	No	3.2
52	2.1	3.2	No	3.2
53	1.56	3.2	No	3.2
54	1.56	3.2	No	3.2
55	3.2	3.2	No	3.2
56	2.1	3.2	No	3.2
57	2.2	3.2	No	3.2
58	2.2	3.2	No	3.2
59	2.2	3.2	No	3.2

60	2.2	3.2	No	3.2
61	3.2	3.2	No	3.2
62	2.1	3.2	No	3.2
63	2.24	3.2	No	3.2
64	3.0	3.2	No	3.2

Table 1 Trace Table for Reading 1

Final Result of Desk Check

- Maximum plant height found in Table 1 = 3.2
- If multiple readings were processed, their respective maximums would be averaged to produce the final result.

3.3 Evaluation of the Algorithm

The desk check demonstrates that the algorithm accurately identifies the tallest plant in the dataset:

- **Correctness:** The algorithm systematically scans all 64 cells without skipping or duplicating values. It updates the maximum height whenever it encounters a larger value. In the sample dataset, the tallest plant height of 3.2 was correctly identified.
- **Logical Flow:** The initialization of `max_height` with the first value ensures that comparisons are valid from the start. The use of nested loops allows for the entire matrix to be covered. After processing all readings, storing the maximum values in a list and calculating their average is both logical and precise.
- **Scalability:** The algorithm can handle any number of readings (N). Each reading is processed independently, and the averaging step ensures that it can adapt to larger datasets.
- **Reliability:** The results from the desk check confirm that the algorithm operates without errors. Even if there are repeated values or multiple cells containing the same maximum (for instance, several instances of 3.2), the algorithm consistently returns the correct maximum.

The evaluation confirms that the algorithm is accurate, logically consistent, and efficient. It performs reliably across various datasets, making it suitable for real-world monitoring of greenhouse plants. The systematic desk check validates both the correctness of the results and the robustness of the algorithm.

Task 04 – Sorting Algorithm

4.1 Name of the Algorithm: Bubble Sort

Bubble Sort is a straightforward comparison-based sorting algorithm. It functions by repeatedly swapping adjacent elements if they are in the wrong order. Each pass through the list moves the largest unsorted element to its correct position at the end of the array. This process continues until the entire array is sorted.

4.2 Pseudocode (Bubble Sort)

```
BEGIN
  INPUT Array A of size n

  FOR i = 0 TO n-1 DO
    FOR j = 0 TO n-i-2 DO
      IF A[j] > A[j+1] THEN
        SWAP A[j] and A[j+1]
      ENDIF
    ENDFOR
  ENDFOR

  OUTPUT A
END
```

4.3 C++ Implementation

main.cpp

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

// Function to perform Bubble Sort
void bubbleSort(int arr[], int n) {
    for (int i = 0; i < n - 1; i++) {
        // Flag to check if any swapping happened
        bool swapped = false;

        for (int j = 0; j < n - i - 1; j++) {
            if (arr[j] > arr[j + 1]) {
                // Swap elements
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
                swapped = true;
            }
        }

        // If no two elements were swapped → array is already sorted
        if (!swapped) break;
    }
}

// Function to print the array
void printArray(int arr[], int n) {
    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}

// Main function
int main() {
    int arr[] = {15, 8, 2, 10, 3, 8, 1, 5, 2};
    int n = sizeof(arr) / sizeof(arr[0]);

    cout << "Original Array: ";
    printArray(arr, n);

    bubbleSort(arr, n);

    cout << "Sorted Array: ";
    printArray(arr, n);

    return 0;
}
```

```
Original Array: 15 8 2 10 3 8 1 5 2
Sorted Array: 1 2 2 3 5 8 8 10 15


...Program finished with exit code 0
Press ENTER to exit console.
```

Figure 5 C++ implementation code

Task 05 – Applying Bubble Sort

Initial List : [15, 8, 2, 10, 3, 8, 1, 5, 2]

	Sorted items
	Pair of items under comparison

1st Pass

15	8	2	10	3	8	1	5	2	->	8	15	2	10	3	8	1	5	2
8	15	2	10	3	8	1	5	2	->	8	2	15	10	3	8	1	5	2
8	2	15	10	3	8	1	5	2	->	8	2	10	15	3	8	1	5	2
8	2	10	15	3	8	1	5	2	->	8	2	10	3	15	8	1	5	2
8	2	10	3	15	8	1	5	2	->	8	2	10	3	8	15	1	5	2
8	2	10	3	8	15	1	5	2	->	8	2	10	3	8	1	15	5	2
8	2	10	3	8	1	15	5	2	->	8	2	10	3	8	1	5	15	2
8	2	10	3	8	1	5	15	2	->	8	2	10	3	8	1	5	2	15

Table 2 1st Pass for Bubble sort

Largest element (15) moved to the end.

2nd Pass

8	2	10	3	8	1	5	2	15	->	2	8	10	3	8	1	5	2	15
2	8	10	3	8	1	5	2	15	->	2	8	10	3	8	1	5	2	15
2	8	10	3	8	1	5	2	15	->	2	8	3	10	8	1	5	2	15
2	8	3	10	8	1	5	2	15	->	2	8	3	8	10	1	5	2	15
2	8	3	8	10	1	5	2	15	->	2	8	3	8	1	10	5	2	15
2	8	3	8	1	10	5	2	15	->	2	8	3	8	1	5	10	2	15
2	8	3	8	1	5	10	2	15	->	2	8	3	8	1	5	2	10	15
2	8	3	8	1	5	2	10	15	->	2	8	3	8	1	5	2	10	15

Table 3 2nd Pass for Bubble sort

Second largest element 10 placed at position 7.

3rd Pass

2	8	3	8	1	5	2	10	15	->	2	8	3	8	1	5	2	10	15
2	8	3	8	1	5	2	10	15	->	2	3	8	8	1	5	2	10	15
2	3	8	8	1	5	2	10	15	->	2	3	8	8	1	5	2	10	15
2	3	8	8	1	5	2	10	15	->	2	3	8	1	8	5	2	10	15
2	3	8	1	8	5	2	10	15	->	2	3	8	1	5	8	2	10	15
2	3	8	1	5	8	2	10	15	->	2	3	8	1	5	2	8	10	15
2	3	8	1	5	2	8	10	15	->	2	3	8	1	5	2	8	10	15
2	3	8	1	5	2	8	10	15	->	2	3	8	1	5	2	8	10	15

Table 4 3rd Pass for Bubble sort

Third largest element (8) has bubbled into correct region.

4th Pass

2	3	8	1	5	2	8	10	15	->	2	3	8	1	5	2	8	10	15
2	3	8	1	5	2	8	10	15	->	2	3	8	1	5	2	8	10	15
2	3	8	1	5	2	8	10	15	->	2	3	1	8	5	2	8	10	15
2	3	1	8	5	2	8	10	15	->	2	3	1	5	8	2	8	10	15
2	3	1	5	8	2	8	10	15	->	2	3	1	5	2	8	8	10	15
2	3	1	5	2	8	8	10	15	->	2	3	1	5	2	8	8	10	15
2	3	1	5	2	8	8	10	15	->	2	3	1	5	2	8	8	10	15
2	3	1	5	2	8	8	10	15	->	2	3	1	5	2	8	8	10	15

Table 5 4th Pass for Bubble sort

Another 8 placed in order.

5th Pass

2	3	1	5	2	8	8	10	15	->	2	3	1	5	2	8	8	10	15
2	3	1	5	2	8	8	10	15	->	2	1	3	5	2	8	8	10	15
2	1	3	5	2	8	8	10	15	->	2	1	3	5	2	8	8	10	15
2	1	3	5	2	8	8	10	15	->	2	1	3	2	5	8	8	10	15
2	1	3	2	5	8	8	10	15	->	2	1	3	2	5	8	8	10	15
2	1	3	2	5	8	8	10	15	->	2	1	3	2	5	8	8	10	15
2	1	3	2	5	8	8	10	15	->	2	1	3	2	5	8	8	10	15
2	1	3	2	5	8	8	10	15	->	2	1	3	2	5	8	8	10	15

Table 6 5th Pass for Bubble sort

Element 5 placed correctly.

6th Pass

2	1	3	2	5	8	8	10	15	->	1	2	3	2	5	8	8	10	15
1	2	3	2	5	8	8	10	15	->	1	2	3	2	5	8	8	10	15
1	2	3	2	5	8	8	10	15	->	1	2	2	3	5	8	8	10	15
1	2	2	3	5	8	8	10	15	->	1	2	2	3	5	8	8	10	15
1	2	2	3	5	8	8	10	15	->	1	2	2	3	5	8	8	10	15
1	2	2	3	5	8	8	10	15	->	1	2	2	3	5	8	8	10	15
1	2	2	3	5	8	8	10	15	->	1	2	2	3	5	8	8	10	15
1	2	2	3	5	8	8	10	15	->	1	2	2	3	5	8	8	10	15

Table 7 6th Pass for Bubble sort

Now the array is sorted.

7th Pass

No swaps → List already sorted

Final Sorted List : [1, 2, 2, 3, 5, 8, 8, 10, 15]