Newton School of Technology

10

Sorting Algorithms

by Gladden Rumao

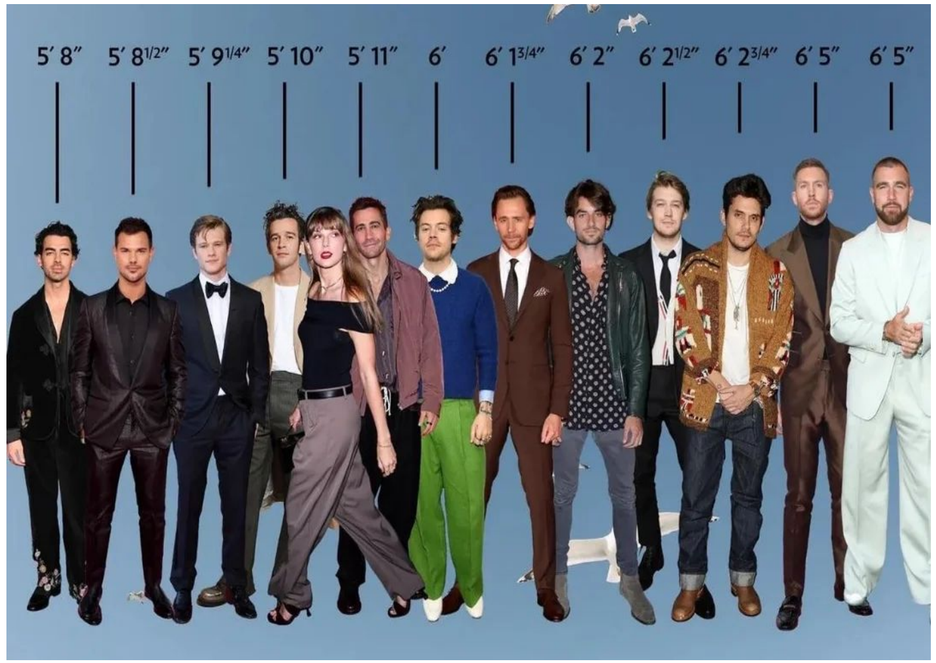
CSA121 : DSA

Sorting

The process of **arranging the elements** in a specific order.
Imagine sorting a list of names, numbers, or search results.

Example :

Example : We arrange students according to their height for a prayer assembly in the school.



What if we have just have the data of students in a list ?

A **sorting algorithm** can be used to find the sorted order of the student's height.

Newton School of Technology

Bubble Sort

Why is it called Bubble Sort ?

Newton School of Technology

- Works similarly to bubbles rising to the surface in water.
- During each pass through the list, larger elements "bubble up" to their correct positions
- This happens by repeatedly comparing and swapping adjacent elements, just like a bubble rising through liquid.

Working principle of Bubble Sort

Working principle

Newton School of Technology

It works by **repeatedly stepping through the list**, **comparing adjacent** elements and swapping them if they are in the wrong order.

Step 1: Comparison

The algorithm compares the **first two adjacent elements** in the list.

Step 2: Swapping

If the elements are in the **wrong order**, they are swapped.

Step 3: Iteration

The process is **repeated for the next pair of elements**, and so on.

Newton School of Technology

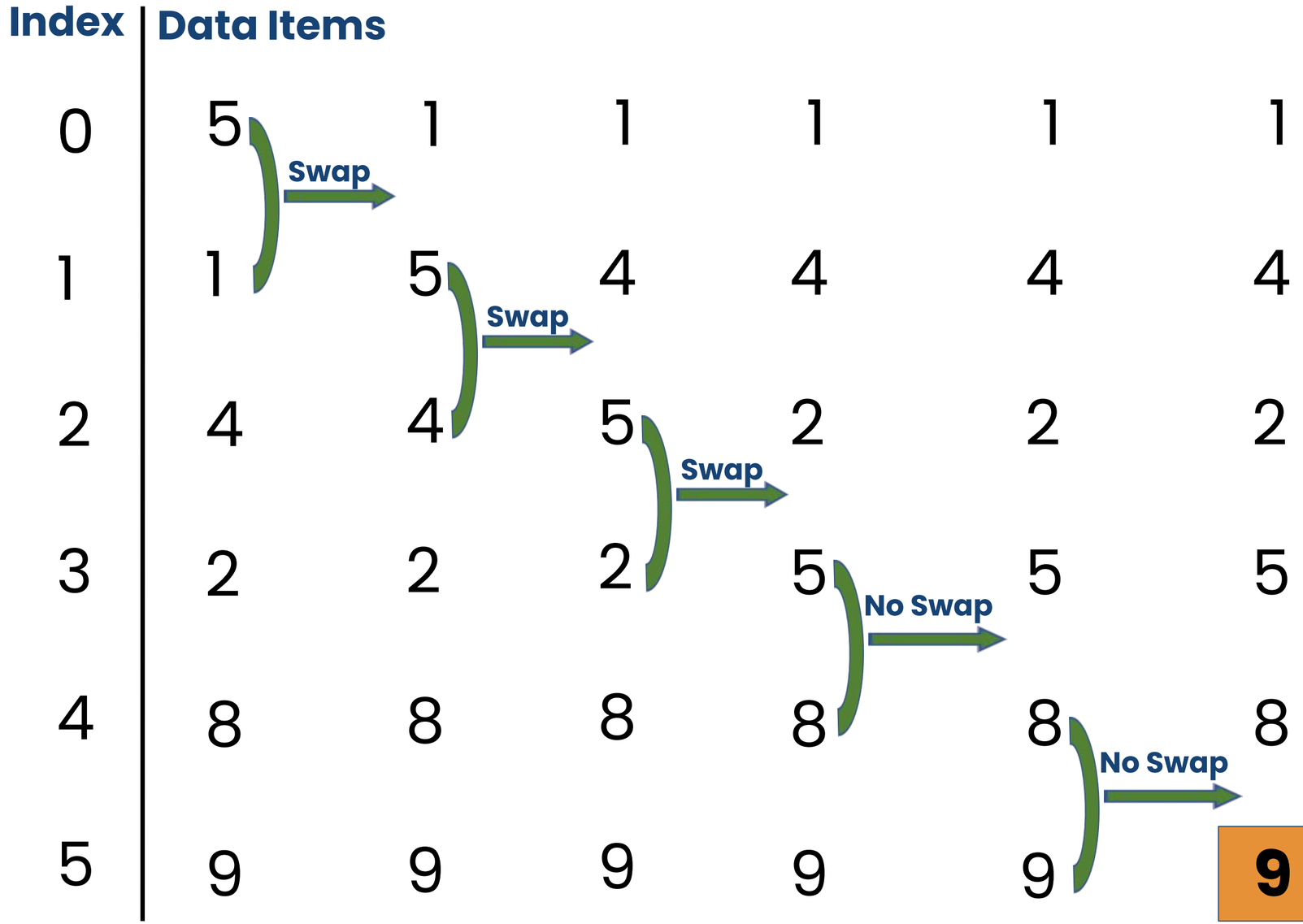
Bubble Sort – Example

Example – Bubble Sort

Index	Data Items
0	5
1	1
2	4
3	2
4	8
5	9

Iteration-1 :

1	4	2	5	8	9



9

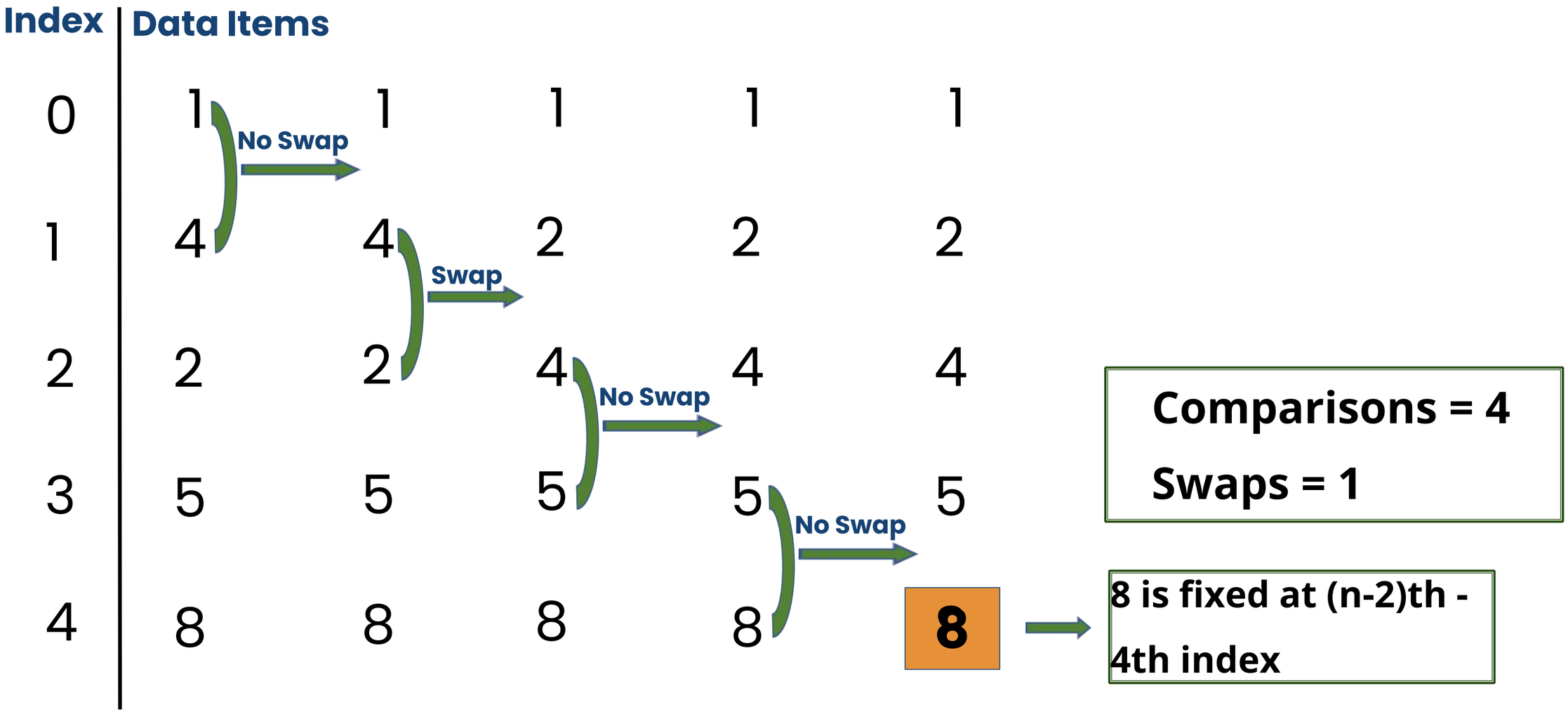
Comparisons = 5

Swaps = 3

9 is fixed at (n-1)th - 5th index

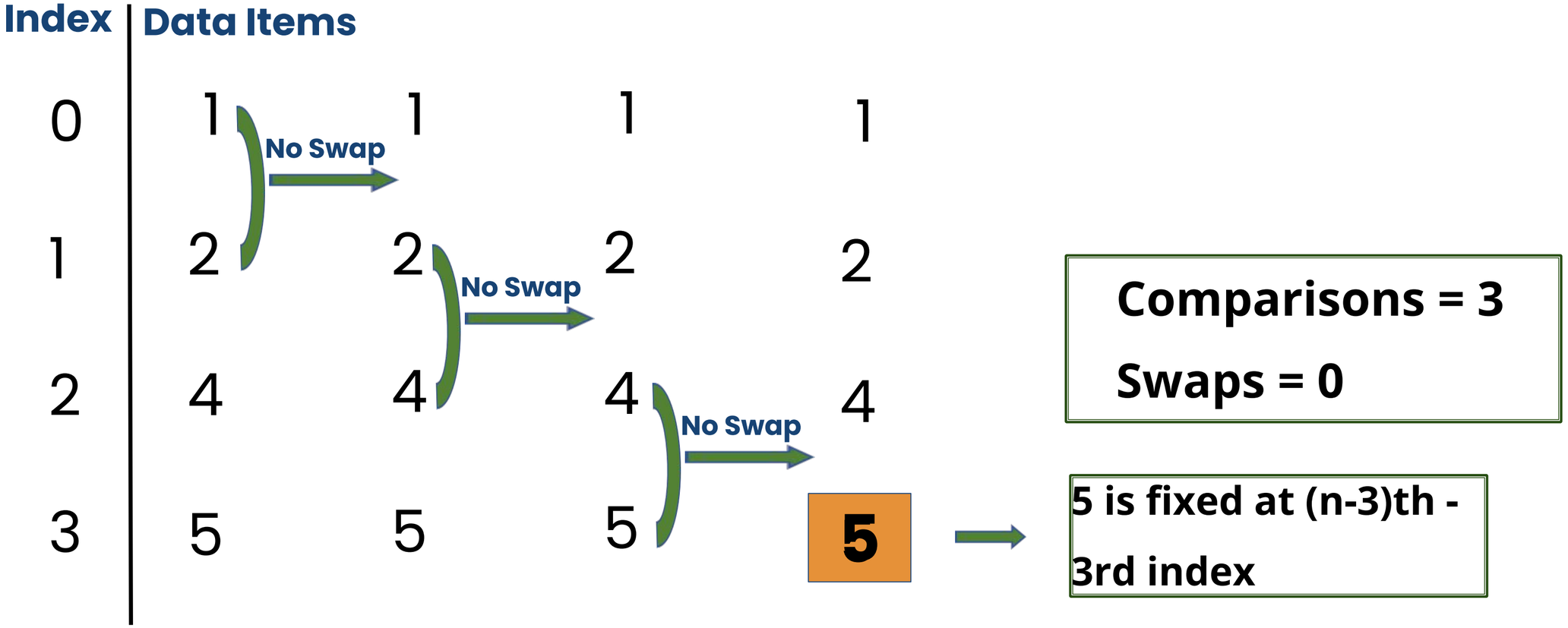
Iteration-2 :

1	2	4	5	8	9



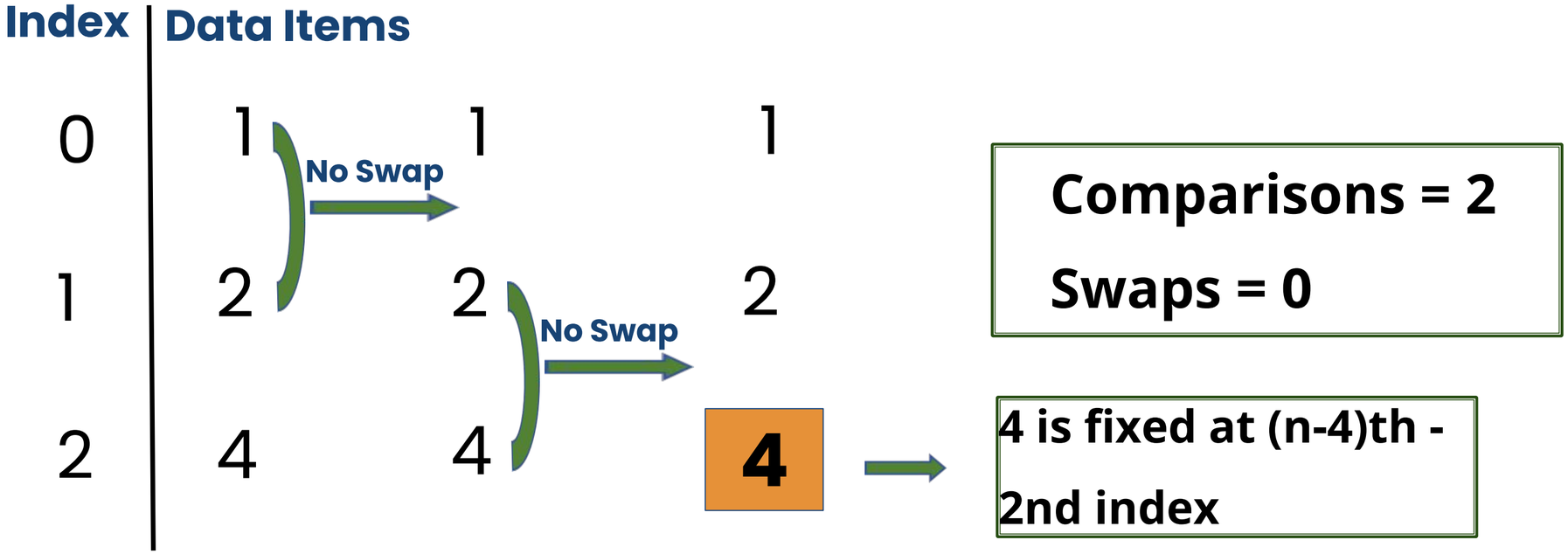
Iteration-3 :

1	2	4	5	8	9



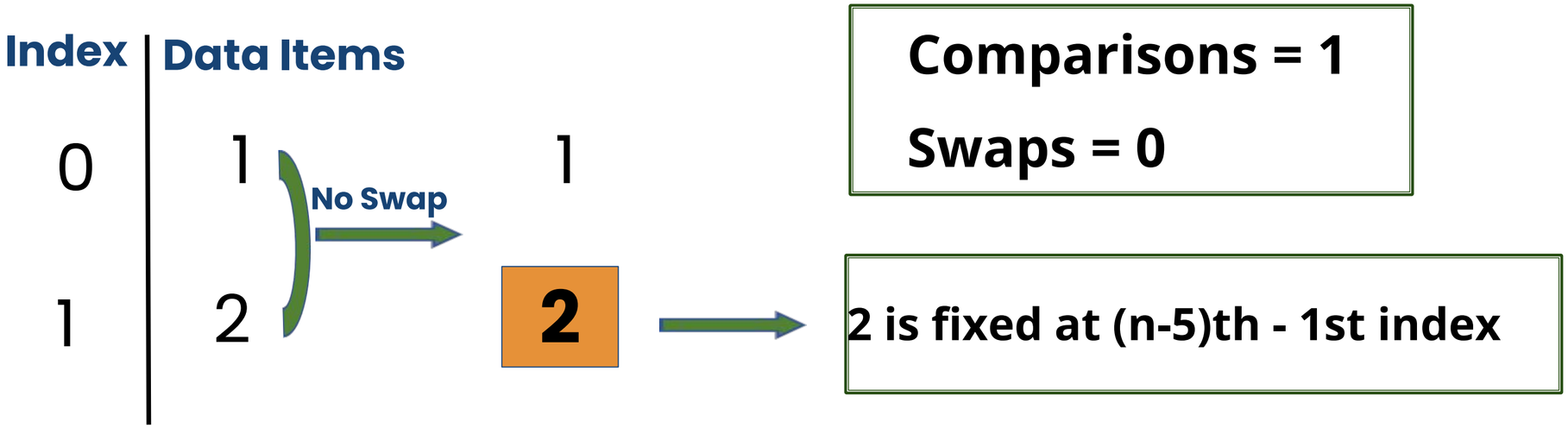
Iteration-4 :

1	2	4	5	8	9



Iteration-5 :

1	2	4	5	8	9



Post Iteration – 5 :

After Iteration 5, only one element remains unsorted in the array.

A **single element is inherently sorted**, as there are no other elements to compare or swap with.

Index	Data Items
0	1

Newton School of Technology

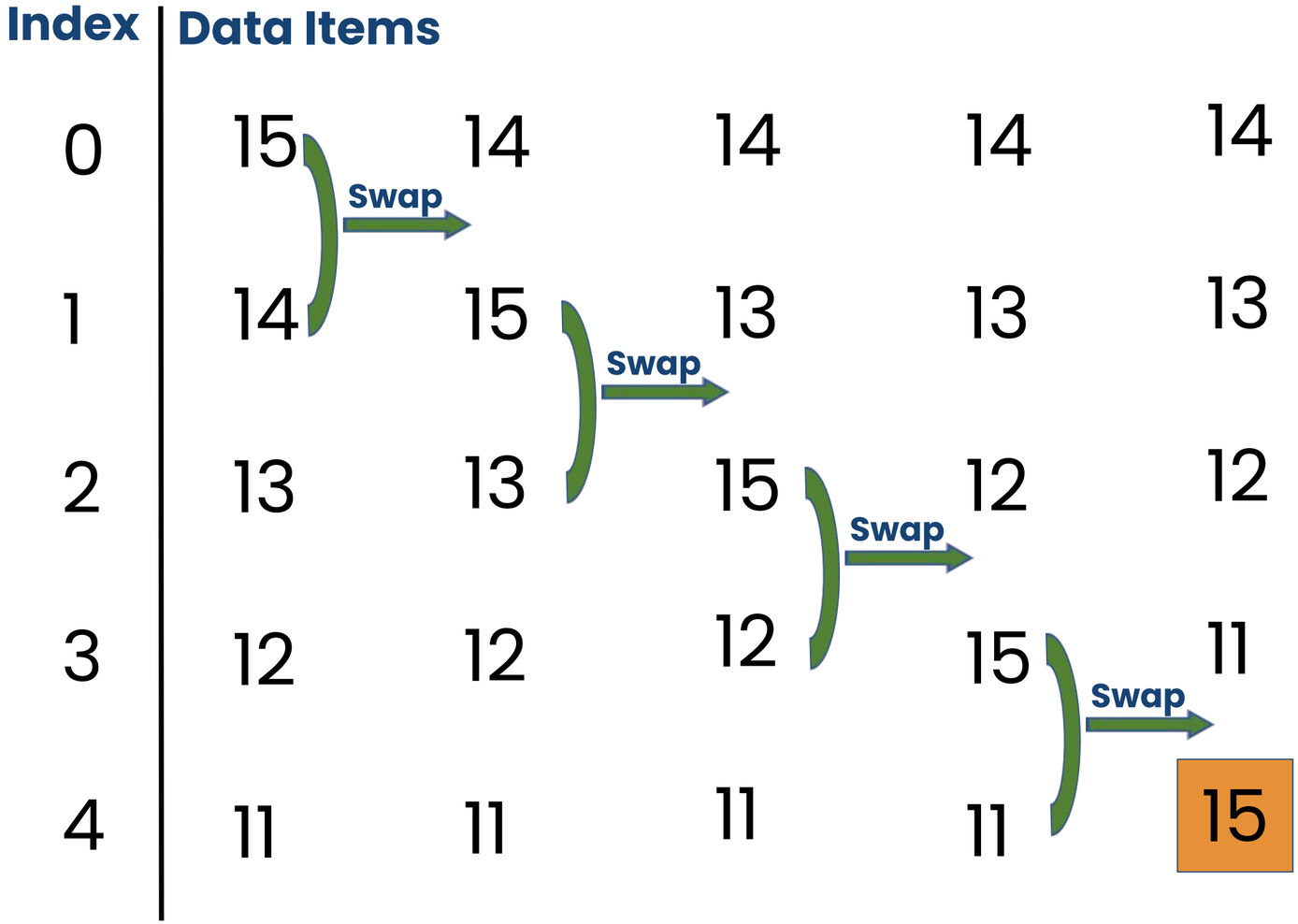
Bubble Sort – Worst Case

Example – Bubble Sort

Index	Data Items
0	15
1	14
2	13
3	12
4	11

Iteration-1 :

14	13	12	11	15



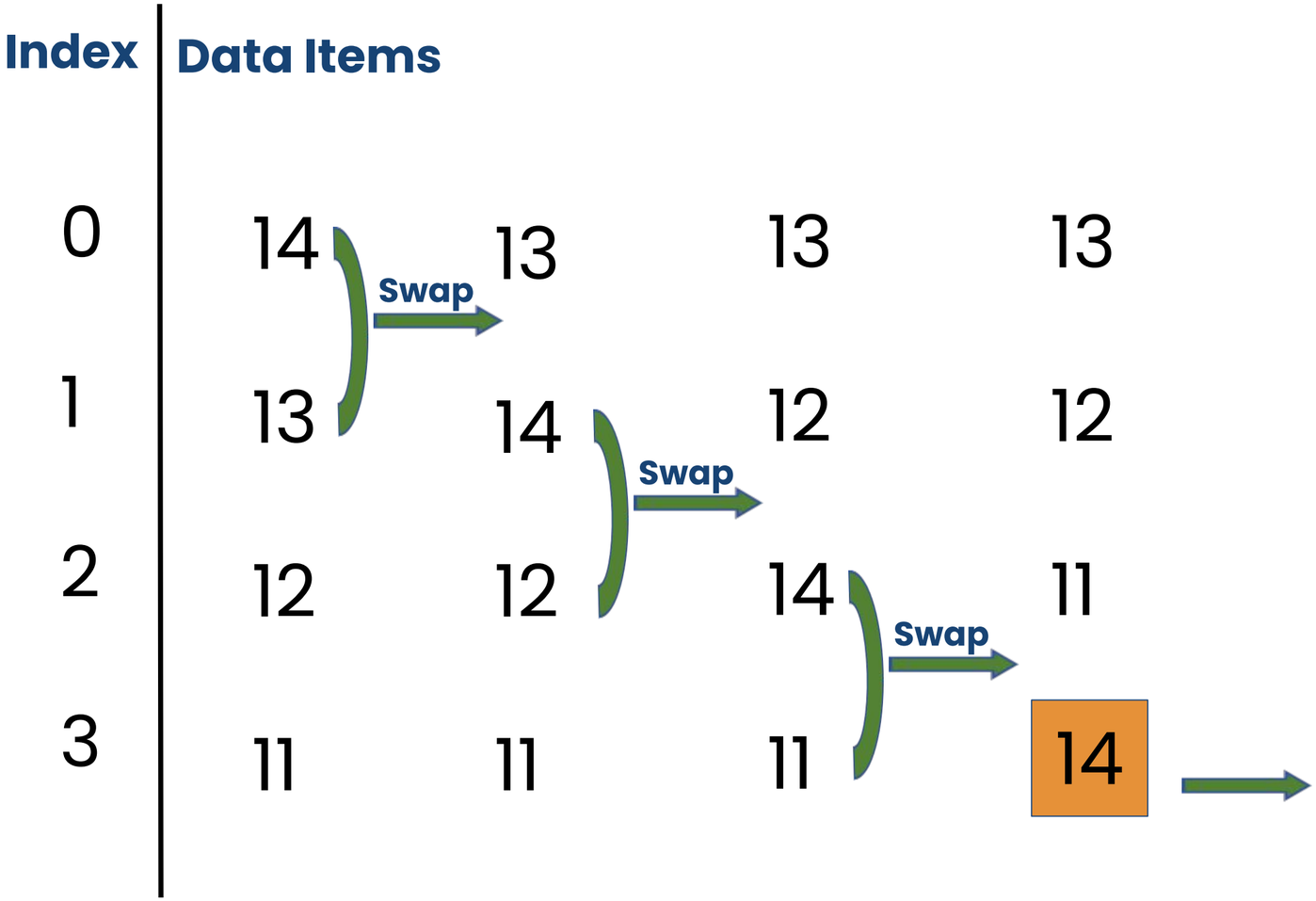
Comparisons = 4

Swaps = 4

15 is fixed at (n-1)th - 4th index

Iteration-2 :

13	12	11	14	15



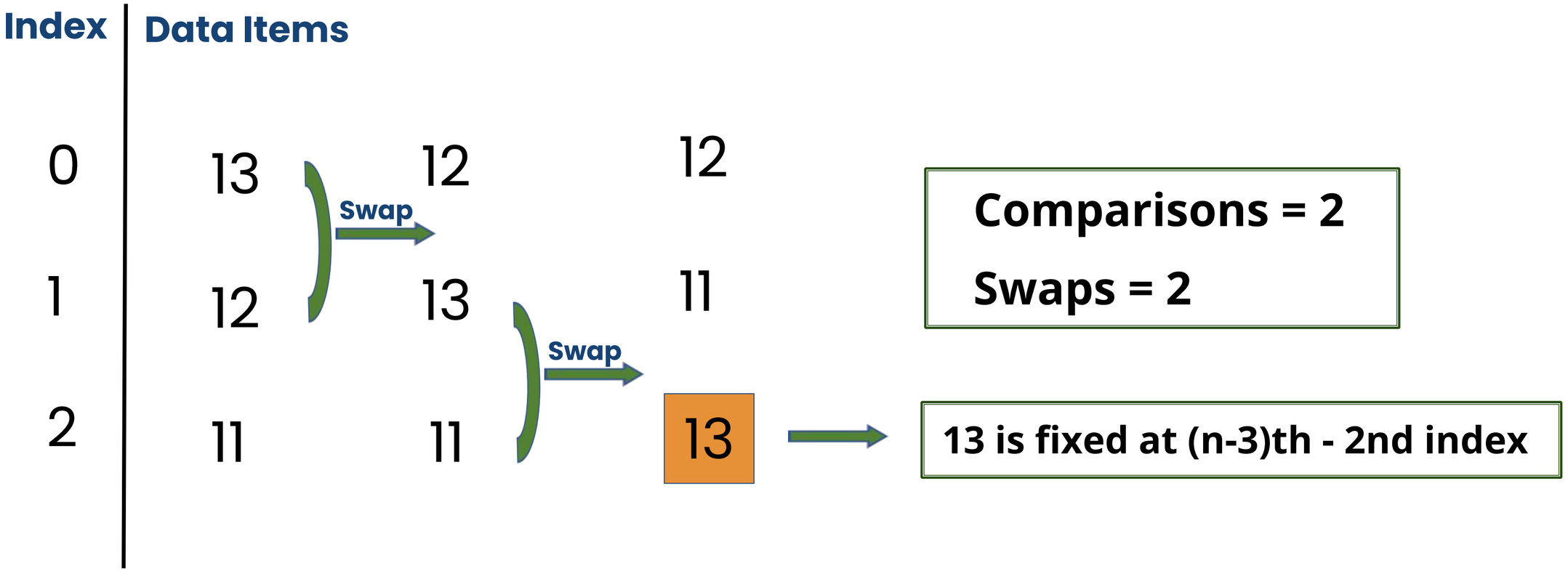
Comparisons = 3

Swaps = 3

14 is fixed at (n-2)th - 3rd index

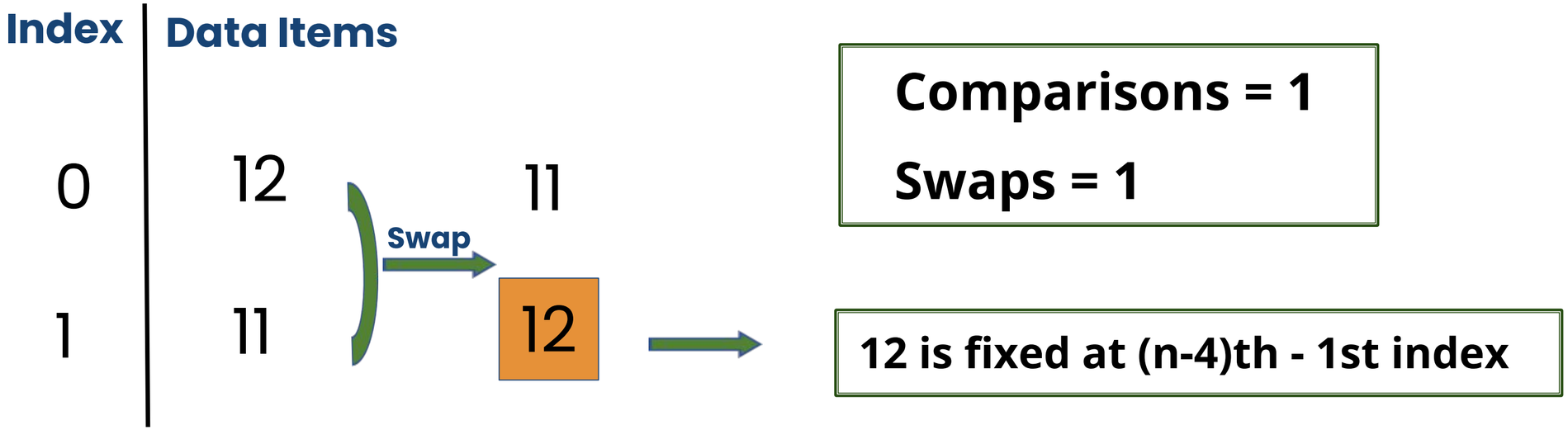
Iteration-3 :

12	11	13	14	15



Iteration-4 :

11	12	13	14	15



Comparisons = 1

Swaps = 1

12 is fixed at (n-4)th - 1st index

Post Iteration-4 :

After Iteration 4, only one element remains unsorted in the array.

A **single element is inherently sorted**, as there are no other elements to compare or swap with.

Index	Data Items
0	11

Code Implementation :

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

Newton School of Technology

Performance Analysis - Worst Case

	Number of Comparisons	Number of Swaps
1st Iteration	n-1	n-1
2nd Iteration	n-2	n-2
3rd Iteration	n-3	n-3
. .	. .	. .
(N-1)th Iteration	1	1

Newton School of Technology

Key Observations :

- **After the first pass**, the largest element reaches its rightful spot at the end.
- **After each pass, the "comparison window"** shrinks since the last elements are already sorted.
- **After each subsequent pass**, the next largest is placed before it, and so on.

Performance Analysis - Worst Case

The worst-case scenario for Bubble Sort occurs when the input list is sorted in reverse order :

5	4	3	2	1

- For each element, the algorithm needs to compare it with every other element in the list to ensure proper placement.
- The total number of **comparisons and swaps required is maximum**.

Performance Analysis - Worst Case

Total number of comparisons = n (n-1) /2

Total number of swaps = n (n-1) /2

$$\text{Worst Case TC} = \frac{n(n-1)}{2} + \frac{n(n-1)}{2} = n(n-1) = O(n^2)$$

Newton School of Technology

How many passes would be required , if input array is already sorted ?

```
def bubble_sort(arr):
    n = len(arr)
    for i in range(n):
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
    return arr
```

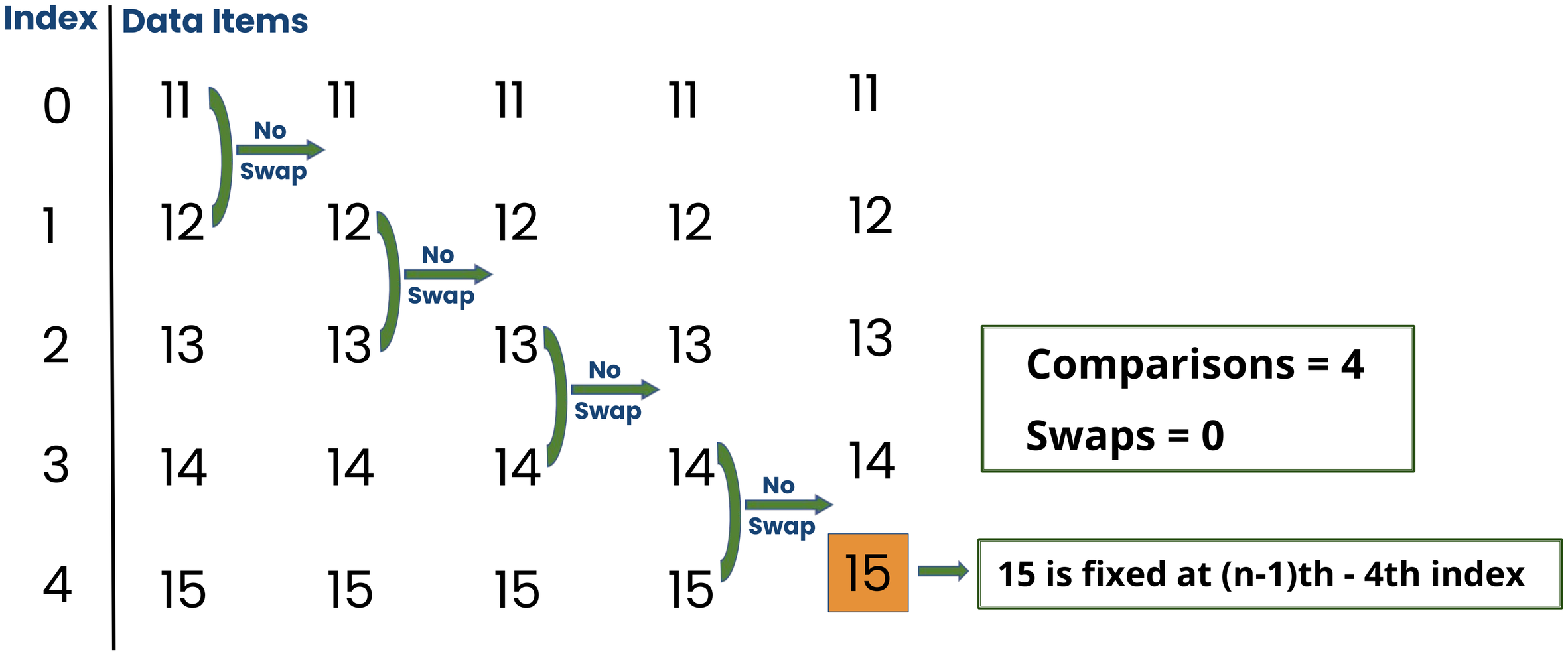
Newton School of Technology

Bubble Sort – Best Case

Iteration - 1

Index	Data Items
0	11
1	12
2	13
3	14
4	15

Iteration - 1


Index
Data Items
0
1
2
3
4
11
12
13
14
15
No Swap
No Swap
No Swap
No Swap
11
12
13
14
15
11
12
13
14
15
11
12
13
14
15
15
Comparisons = 4
Swaps = 0
15 is fixed at (n-1)th - 4th index

Performance Analysis - Best Case :

Input list is already sorted in ascending order

	Number of Comparisons	**Number of Swaps**
1st Iteration	n-1	0

Total number of comparisons = n-1

Total number of swaps = 0

Best Case Time Complexity = **O(n)**

Newton School of Technology

Early exit – Optimisation :

```python
def bubble_sort_optimized(arr):
    n = len(arr)
    for i in range(n):
        swapped = False
        for j in range(n - i - 1):
            if arr[j] > arr[j + 1]:
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
                swapped = True
        if not swapped:  # Exit early if no swaps occurred in this pass
            break
    return arr
```

Newton School of Technology

Time Complexity of Bubble Sort :

Worst Case	Average Case	Best Case
$O(n^2)$	$O(n^2)$	$O(n)$

Space Complexity of Bubble Sort

Bubble Sort has a space complexity of O(1)

O(1) because it is an **in-place sorting algorithm**.

It **does not require any additional space** proportional to the input size.

Sorting is achieved by swapping adjacent elements directly within the input list.

About Bubble Sort

Newton School of Technology

- **Stable** Sorting Algorithm.
- **Inplace** Sorting Algorithm.
- **Rarely used** due to poor performance.

Advantages and Disadvantages :

Advantages	Disadvantages
Simple and easy to implement.	Inefficient for large datasets due to O(n2) time complexity.
Requires only O(1) additional space (in-place sorting).	High number of comparisons and swaps in the worst case.

Newton School of Technology

END