



## *Arrays: Elementary sorts*

# Agenda

---

- Explore the elementary sort algorithms:
  - Bubble sort
  - Insertion sort
- Understand the following aspects
  - Algorithm mechanism and pseudocode
  - Algorithm iterations on varying input
  - Algorithm time and space complexity



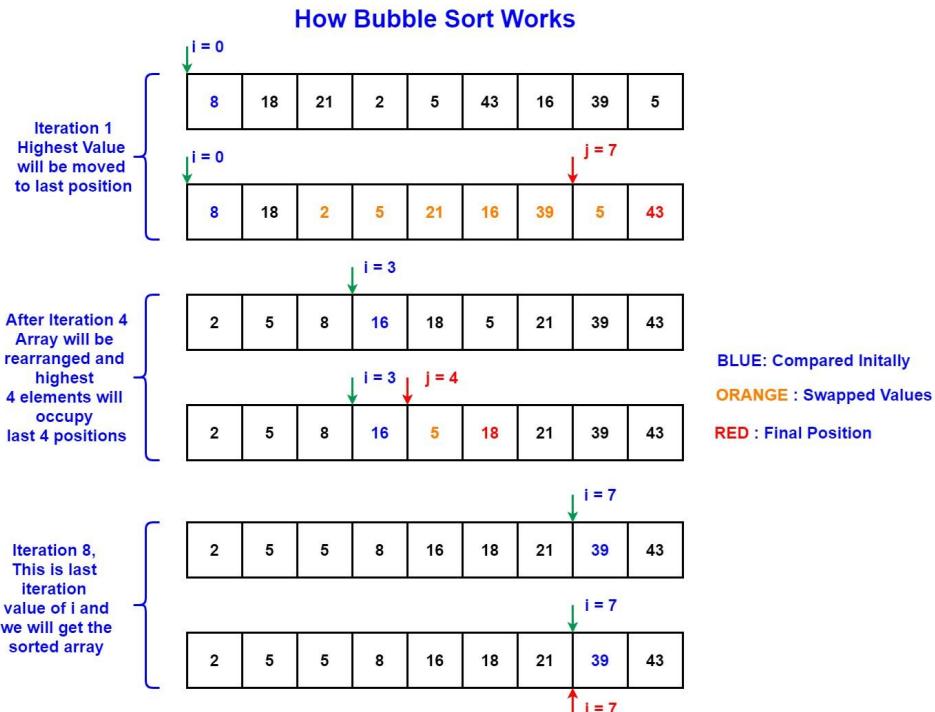
# Elementary sorts

- Historically, two categories of sort algorithms:
  - Comparison based sorting:
    - Bubble sort
    - Insertion sort
    - Selection sort
    - Merge sort
    - Quick sort
  - Non comparison based Sorting:
    - Bucket sort
    - Radix sort

# Bubble sort

- A comparison-based sorting algorithm.
- Comparison operation:Defined as needed by data type:
  - Numbers
  - Strings
  - Objects: by attributes
- Algorithm scheme: Run N iterations of the following:
  - i. Current iteration:
    - Start with the first 2 index elements.
    - If first greater then second, swap them.
    - Next, exclude first index, continue comparing.
  - ii. In each iteration, the next greatest element is bubbled-up to its final position.

# Bubble sort



# Bubble sort pseudocode

## Code For Bubble Sort

```
def swap(array, i, j):
    temp = array[i]
    array[i] = array[j]
    array[j] = temp
```

```
def bubble_sort(array):
    size = len(array)
    for i in range(0, size):
        for j in range(size-i-1):
            if array[j] > array[j+1]:
                swap(array, j, j+1)
            else:
                break
```

# Bubble sort iterations

Bubble Sort Iterations

i	j	0	1	2	3	4	5	6	7	8
		8	18	21	2	5	43	16	39	5
0	0	8	18	21	2	5	43	16	39	5
0	1	8	18	21	2	5	43	16	39	5
0	2	8	18	2	21	5	43	16	39	5
0	3	8	18	2	5	21	43	16	39	5
0	4	8	18	2	5	21	43	16	39	5
0	5	8	18	2	5	21	16	43	39	5
0	6	8	18	2	5	21	16	39	43	5
0	7	8	18	2	5	21	16	39	5	43
1	0	8	18	2	5	21	16	39	5	43
1	1	8	2	18	5	21	16	39	5	43
1	2	8	2	5	18	21	16	39	5	43
1	3	8	2	5	18	21	16	39	5	43
1	4	8	2	5	18	16	21	39	5	43
1	5	8	2	5	18	16	21	39	5	43
1	6	8	2	5	18	16	21	5	39	43
2	0	2	8	5	18	16	21	5	39	43
2	1	2	5	8	18	16	21	5	39	43
2	2	2	5	8	18	16	21	5	39	43

(VIOLET) Initial Array

(BLACK) Swapping

(Blue) Indexes

(Grey) No Swapping at any position

(GREEN) Final Sorted Array → 2 5 5 8 16 18 21 39 43

# Bubble sort iterations

Bubble Sort Iterations  
On A Pre-Sorted Array

i	j	0	1	2	3	4	5	6	7	8
0	0	2	5	5	8	16	18	21	39	43
0	1	2	5	5	8	16	18	21	39	43
0	2	2	5	5	8	16	18	21	39	43
0	3	2	5	5	8	16	18	21	39	43
0	4	2	5	5	8	16	18	21	39	43
0	5	2	5	5	8	16	18	21	39	43
0	6	2	5	5	8	16	18	21	39	43
0	7	2	5	5	8	16	18	21	39	43
1	0	2	5	5	8	16	18	21	39	43
1	1	2	5	5	8	16	18	21	39	43
1	2	2	5	5	8	16	18	21	39	43
1	3	2	5	5	8	16	18	21	39	43
1	4	2	5	5	8	16	18	21	39	43
1	5	2	5	5	8	16	18	21	39	43
1	6	2	5	5	8	16	18	21	39	43
2	0	2	5	5	8	16	18	21	39	43
2	1	2	5	5	8	16	18	21	39	43
2	2	2	5	5	8	16	18	21	39	43

(VIOLET) Initial Array

(Grey) No Swapping at any position

(GREEN) Final Sorted Array → 2 5 5 8 16 18 21 39 43

(BLUE) Indexes

# Bubble sort iterations

Bubble Sort Iterations  
On A Reverse-Sorted Array

i	j	0	1	2	3	4	5	6	7	8
		43	39	21	18	16	8	5	5	2
(VIOLET) Initial Array										
0	0	39	43	21	18	16	8	5	5	2
0	1	39	21	43	18	16	8	5	5	2
0	2	39	21	18	43	16	8	5	5	2
0	3	39	21	18	16	43	8	5	5	2
0	4	39	21	18	16	8	43	5	5	2
0	5	39	21	18	16	8	5	43	5	2
0	6	39	21	18	16	8	5	5	43	2
0	7	39	21	18	16	8	5	5	2	43
1	0	21	39	18	16	8	5	5	2	43
1	1	21	18	39	16	8	5	5	2	43
1	2	21	18	16	39	8	5	5	2	43
1	3	21	18	16	8	39	5	5	2	43
1	4	21	18	16	8	5	39	5	2	43
1	5	21	18	16	8	5	5	39	2	43
1	6	21	18	16	8	5	5	2	39	43
2	0	18	21	16	8	5	5	2	39	43
2	1	18	16	21	8	5	5	2	39	43
2	2	18	16	8	21	5	5	2	39	43
(BLUE) Indexes										
(Black) Swapping										
(Grey) No Swapping at any position										
(GREEN) Final Sorted Array										
		2	5	5	8	16	18	21	39	43

# Bubble sort: Complexity

- In Bubble sort algorithm discussed above;
  - The first element is compared with  $(N-1)$  elements;
  - The next element is compared with the  $(N-2)$  elements;
  - So overall if we calculate the total number of comparisons:
    - Total Comparison:  $(N-1) + (N-2) + (N-3) + \dots + 1$
    - Proportional to  $N^2$
  - Best Case, Worst case, and Average Case:
    - Number of comparisons is proportional to  $N^2$

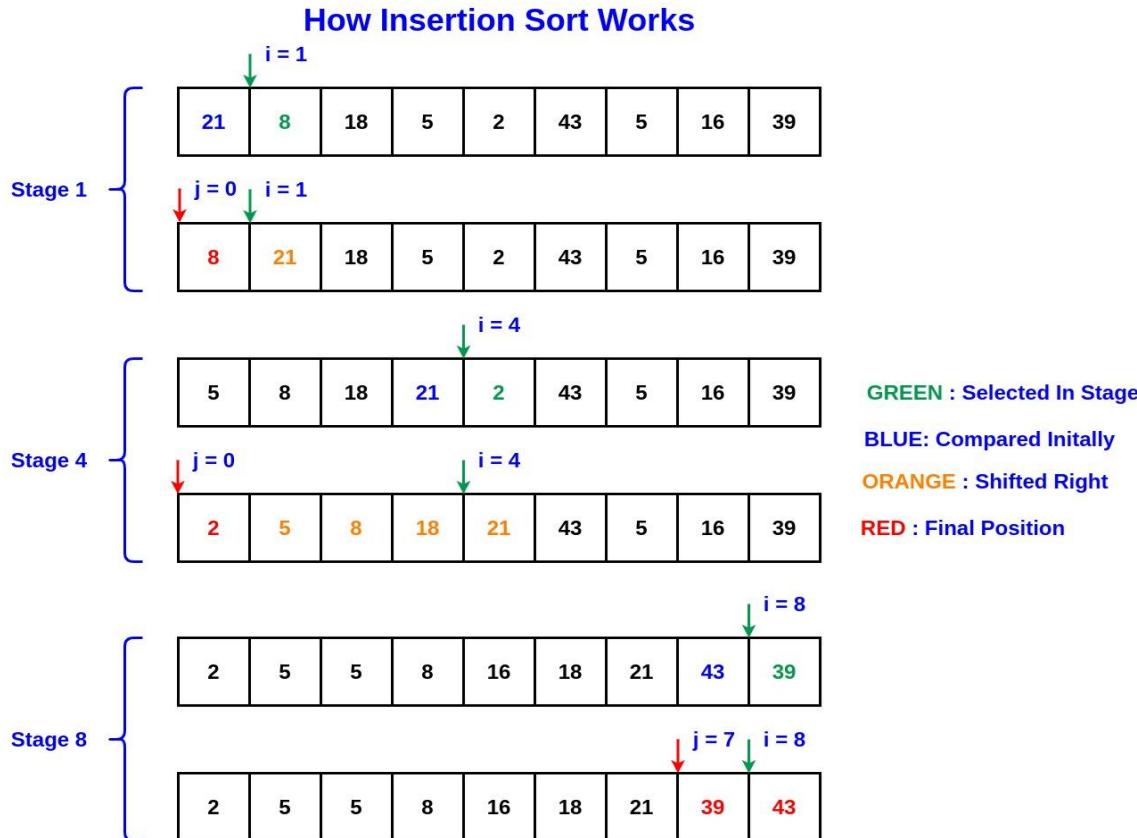
# Bubble sort: Complexity

- Analyzing space complexity
  - Bubble sort is an in-place, iterative sort algorithm - no additional space needed.
    - Best case: Constant space
    - Worst case: Constant space
    - Average case: Constant space

# Insertion sort

- A comparison-based sort algorithm.
- Algorithm scheme:
  - i. Make  $N$  iterations over the given array  $A[1\dots N-1]$
  - ii. In iteration  $i$ :
    - Swap  $a[i]$  with each element to its left, that is larger.
    - Iteration puts  $a[i]$  in its right place.

# Insertion sort



# Insertion sort pseudocode

## Code For Insertion Sort

```
def swap(array, i, j):  
    temp = array[i]  
    array[i] = array[j]  
    array[j] = temp
```

```
def insertion_sort(array):  
    size = len(array)  
    for i in range(0, size):  
        for j in range(i, 0, -1):  
            if array[j] < array[j-1]:  
                swap(array, j, j-1)  
            else:  
                break
```

# Insertion sort iterations

## Insertion Sort Iterations

Initial Array (VIOLET):

i	j	0	1	2	3	4	5	6	7	8
		21	8	18	5	2	43	5	16	39
1	0	8	21	18	5	2	43	5	16	39
2	1	8	18	21	5	2	43	5	16	39
3	0	5	8	18	21	2	43	5	16	39
4	0	2	5	8	18	21	43	5	16	39

(GREY):

i	j	0	1	2	3	4	5	6	7	8
		21	8	18	5	2	43	5	16	39
1	0	8	21	18	5	2	43	5	16	39
2	1	8	18	21	5	2	43	5	16	39
3	0	5	8	18	21	2	43	5	16	39
4	0	2	5	8	18	21	43	5	16	39

No Movement:

i	j	0	1	2	3	4	5	6	7	8
		21	8	18	5	2	43	5	16	39
1	0	8	21	18	5	2	43	5	16	39
2	1	8	18	21	5	2	43	5	16	39
3	0	5	8	18	21	2	43	5	16	39
4	0	2	5	8	18	21	43	5	16	39

(Blue) Indexes:

(Red) Resting position of  $A[i]$  in Iteration i

(Black) Elements Crossed By  $A[i]$

(VIOLET) Initial Array

(Grey) (No Movement)

(Green) Final Sorted Array

# Insertion sort iterations

Insertion Sort Iterations  
On A Pre-Sorted Array

i	j	0	1	2	3	4	5	6	7	8
		2	5	5	8	16	18	21	39	43
1	1	2	5	5	8	16	18	21	39	43
2	2	2	5	5	8	16	18	21	39	43
3	3	2	5	5	8	16	18	21	39	43
4	4	2	5	5	8	16	18	21	39	43

(VIOLET) Initial Array

i	j	0	1	2	3	4	5	6	7	8		
		2	5	5	8	16	18	21	39	43		
		5	5	2	5	5	8	16	18	21	39	43
		6	6	2	5	5	8	16	18	21	39	43
		7	7	2	5	5	8	16	18	21	39	43
		8	8	2	5	5	8	16	18	21	39	43

(GREEN) Final Sorted Array → 2 5 5 8 16 18 21 39 43

# Insertion sort iterations

Insertion Sort Iterations  
On A Reverse-Sorted Array

i	j	0	1	2	3	4	5	6	7	8
		43	39	21	18	16	8	5	5	2
1	0	39	43	21	18	16	8	5	5	2
2	0	21	39	43	18	16	8	5	5	2
3	0	18	21	39	43	16	8	5	5	2
4	0	16	18	21	39	43	8	5	5	2

(VIOLET) Initial Array

i	j	0	1	2	3	4	5	6	7	8
		16	18	21	39	43	8	5	5	2
5	0	8	16	18	21	39	43	5	5	2
6	0	5	8	16	18	21	39	43	5	2
7	1	5	5	8	16	18	21	39	43	2
8	0	2	5	5	8	16	18	21	39	43

(GREEN) Final Sorted Array → 2 5 5 8 16 18 21 39 43

# Insertion sort: complexity

- Analyzing time complexity:
  - Number of comparisons
  - Number of exchanges
- Best case: When the array is pre-sorted in ascending order.
  - Comparisons:  $N-1$
  - Exchanges: 0

# Insertion sort: complexity

- Worst case: The array is pre-sorted in descending order.
  - Comparisons: Proportional to  $N^2$
  - Exchanges: Proportional to  $N^2$
- Average case: Array with random ordering.
  - Comparisons: Proportional to  $N^2$  (Differing constants)
  - Exchanges: Proportional to  $N^2$  (Differing constants)

# Insertion sort: complexity

- Analyzing space complexity
  - Insertion sort is an in-place, iterative sort algorithm - no additional space needed.
  - Best case: Constant space.
  - Worst case: Constant space
  - Average Case: Constant space

We looked at the following aspects of bubble sort and insertion sort:

- Algorithm mechanism and pseudocode
- Algorithm iterations on different input
- Time and space complexity



# Thank You