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

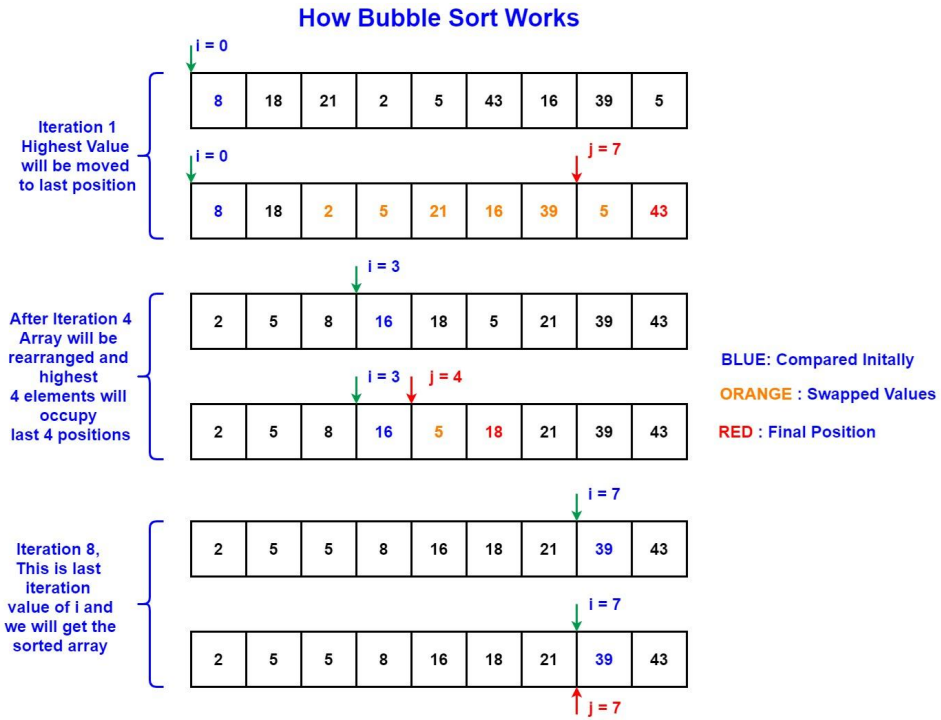
Elementary sorts

- 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



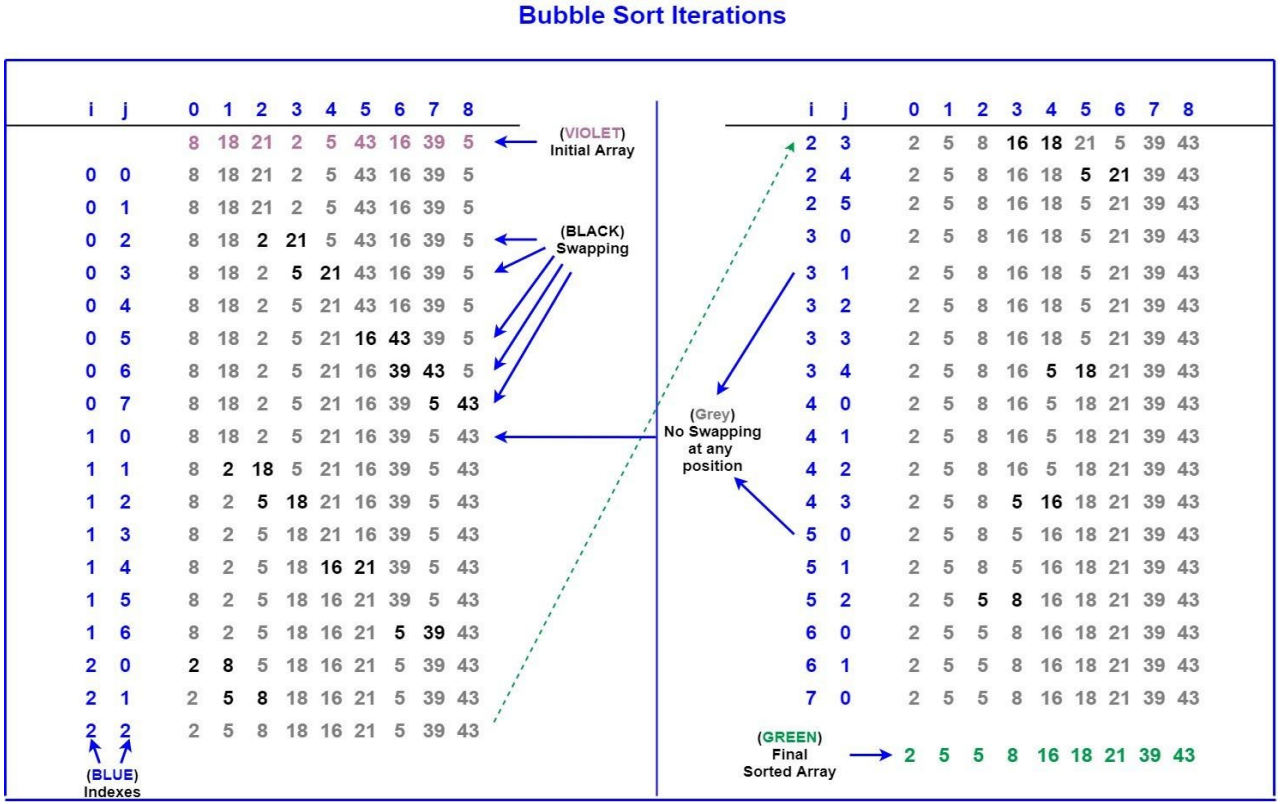
Bubble sort pseudocode

Code For Bubble Sort

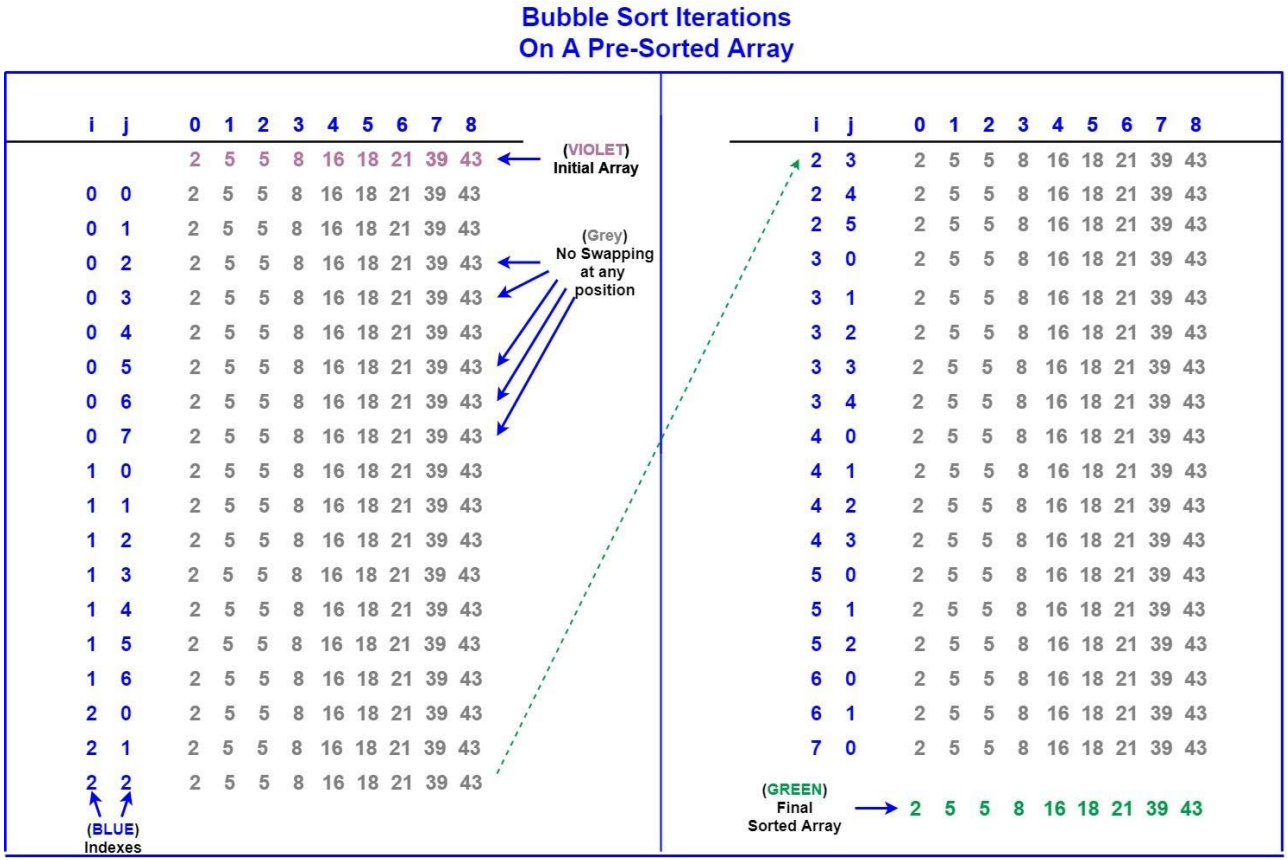
```
def swap(array, i, j):
    temp = array[i]
    array[i] = array[j]
    array[j] = temp
```

```
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
```

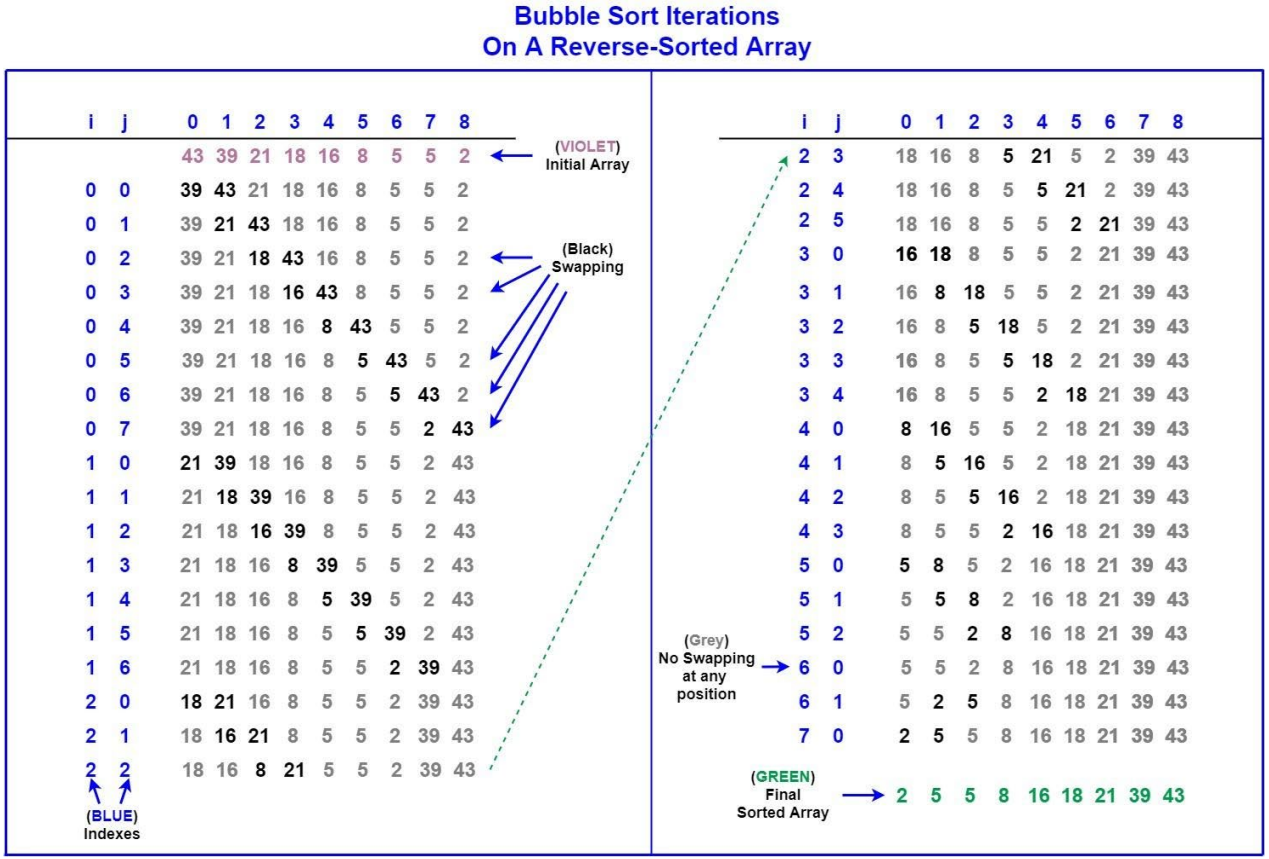
Bubble sort iterations



Bubble sort iterations



Bubble sort iterations



Bubble sort: Complexity

- In Bubble sort algorithm discussed above;
  - The first element is conmpared 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) + …… + 1
    - Proportional to N^2
  - Best Case, Worst case, and Average Case:
    - Number of comparisons is proportional to N^2

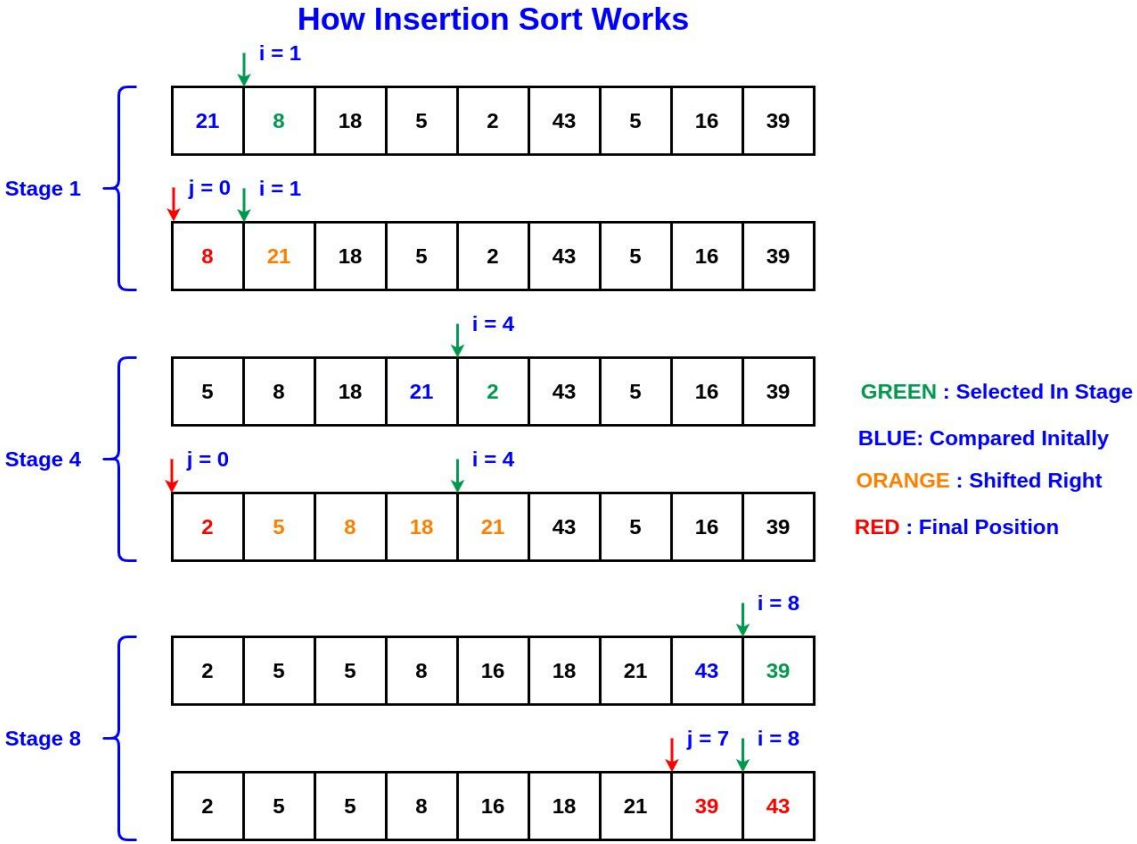
Bubble sort: Complexity

- 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...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



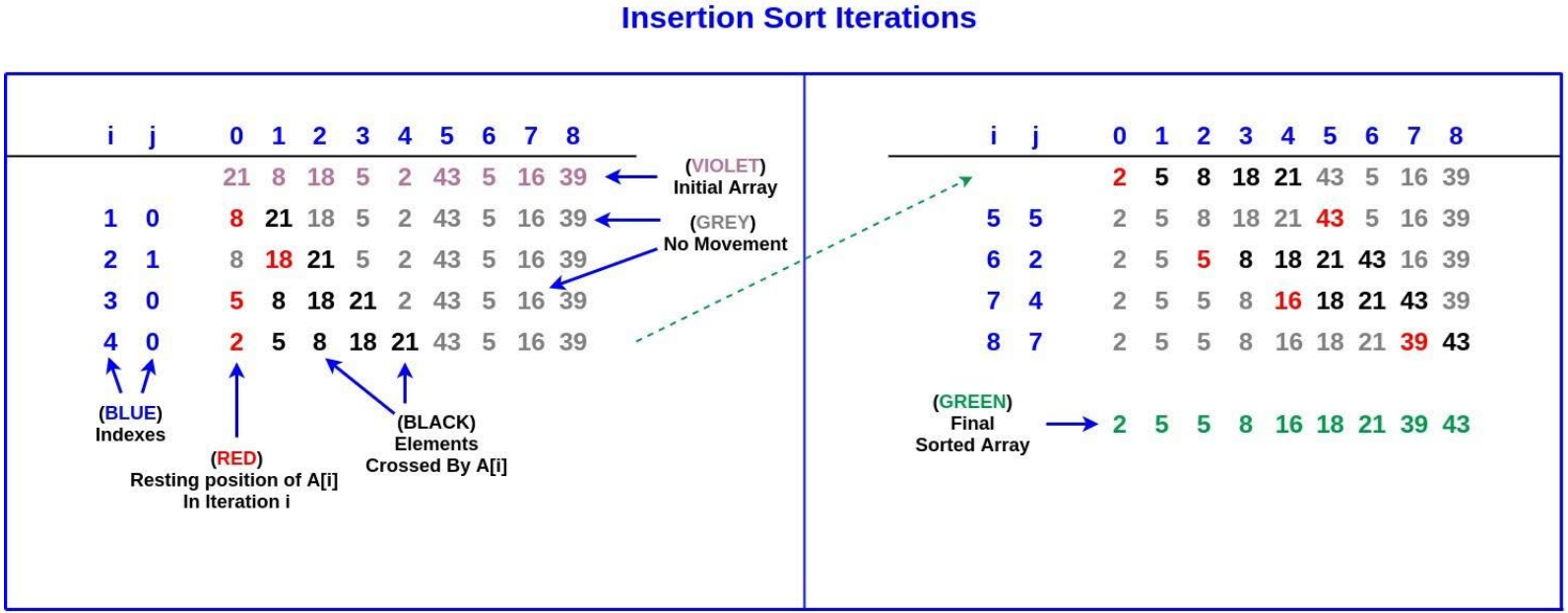
Insertion sort pseudocode

Code For Insertion Sort

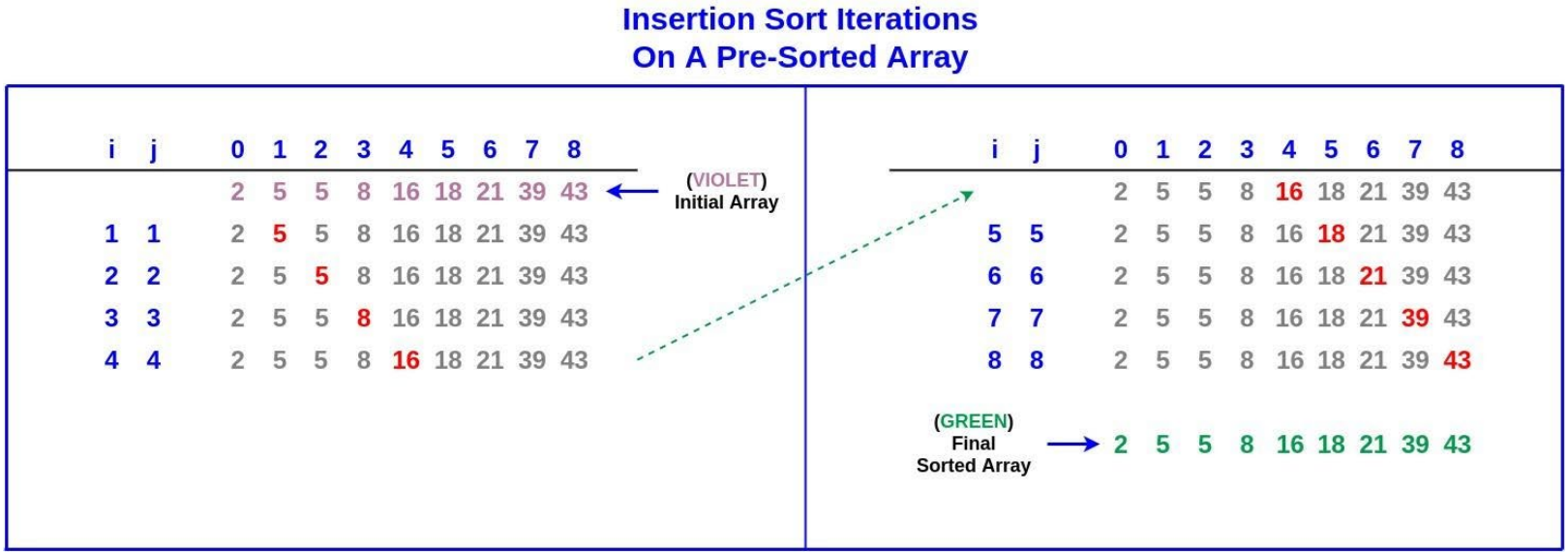
```
def swap(array, i, j):
    temp = array[i]
    array[i] = array[j]
    array[j] = temp
```

```
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
```

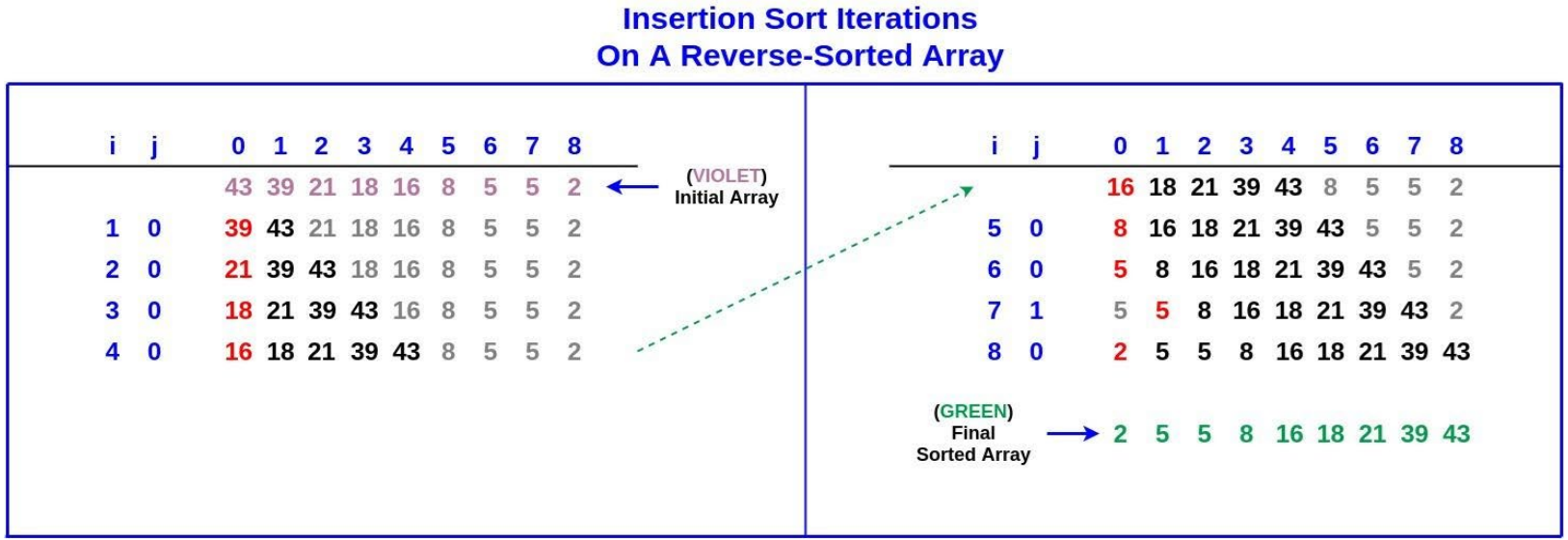
Insertion sort iterations



Insertion sort iterations



Insertion sort iterations



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

Summary

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