

Bubble Sort

Shusen Wang

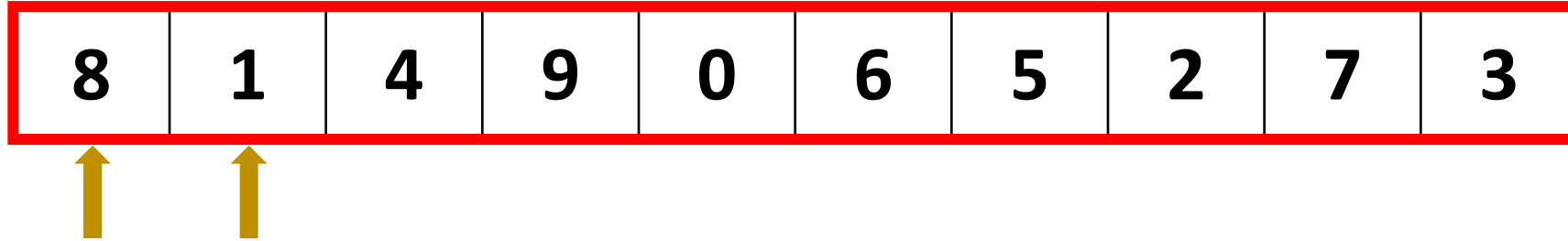
Sorting

8	1	4	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---

- **Input:** An array with n elements.
- **Goal:** Sort the array so that the elements are in the ascending (or descending) order.

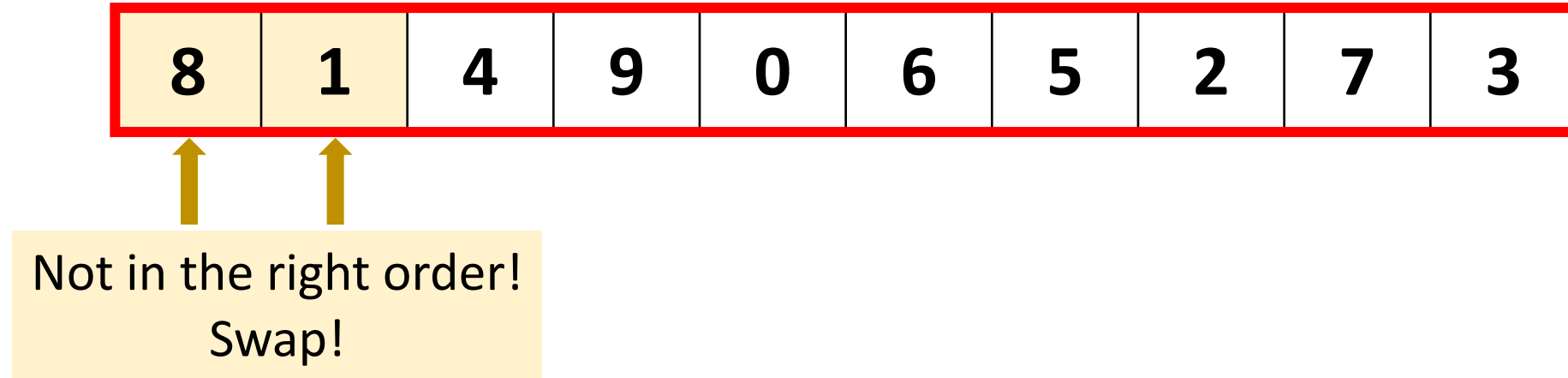
Iteration 1

8	1	4	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

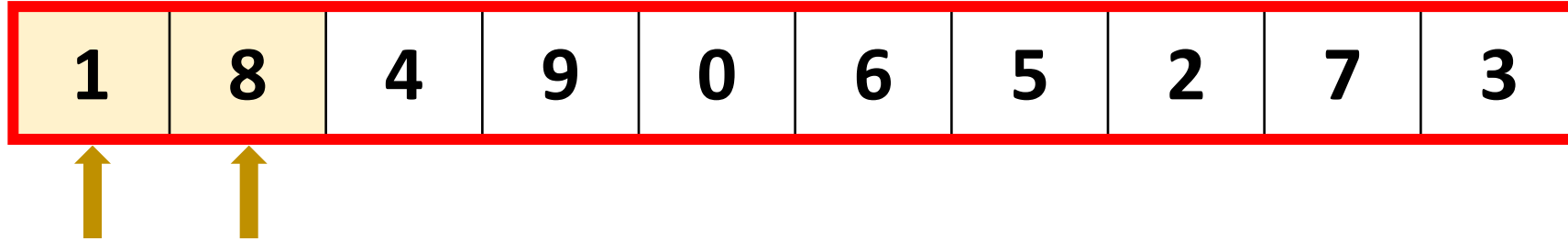
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1

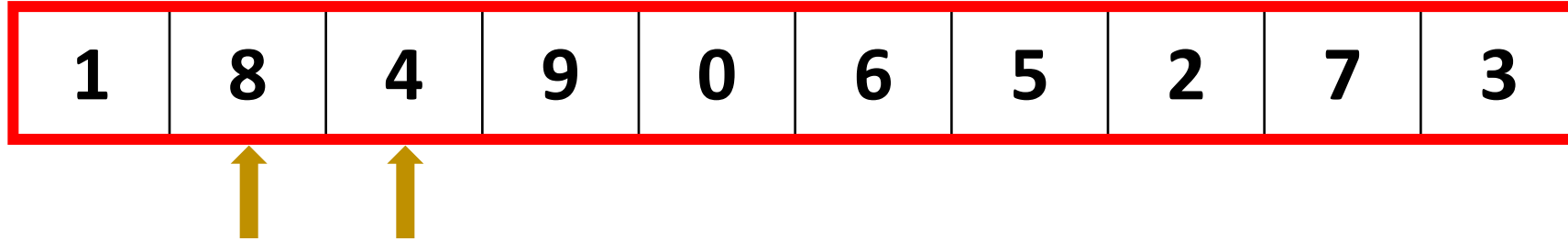
1	8	4	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

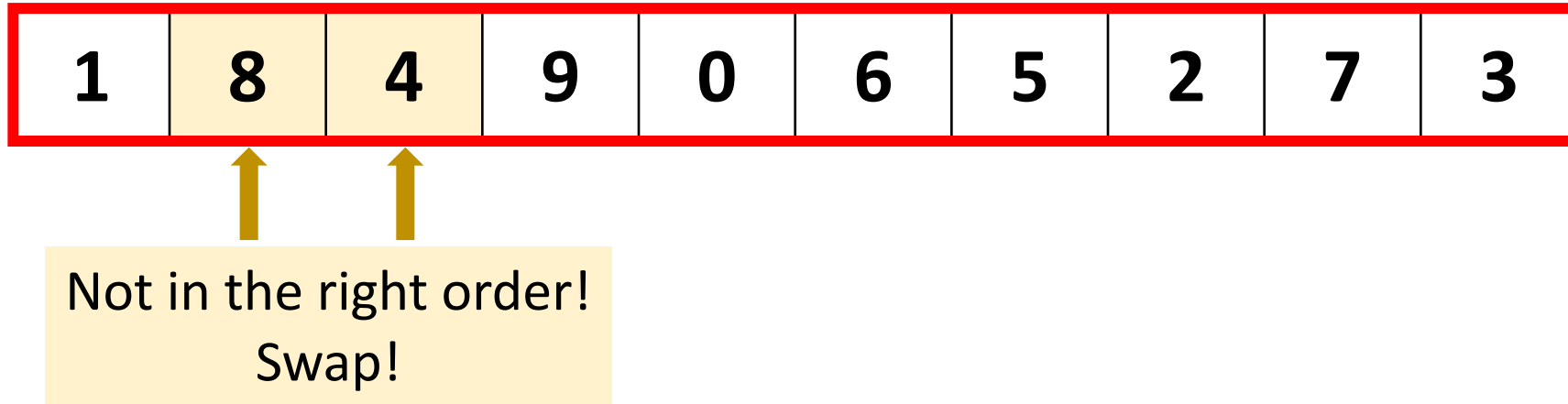
Iteration 1

1	8	4	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---



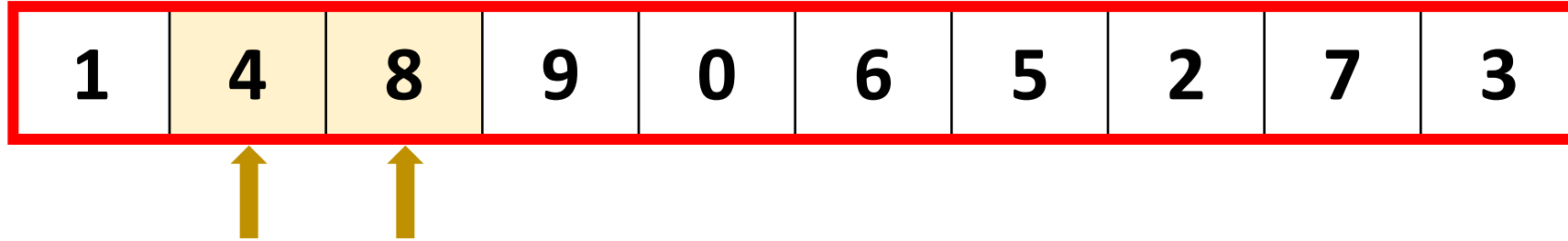
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1



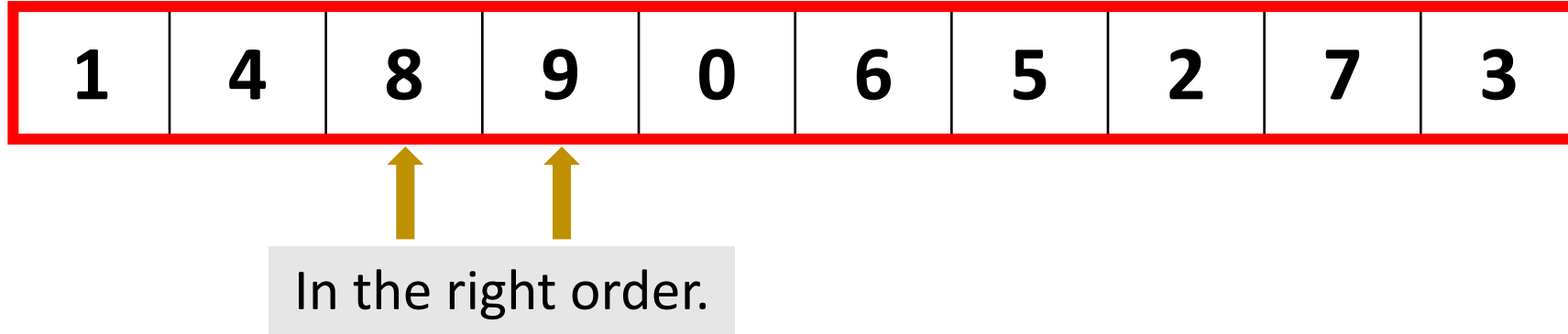
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

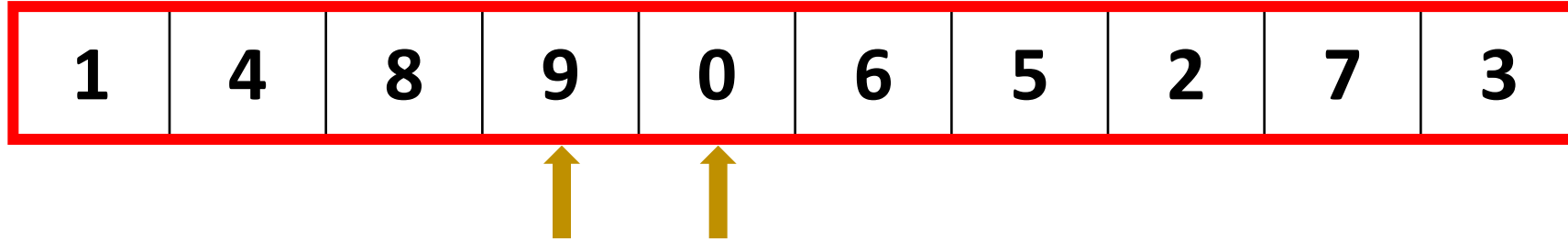
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

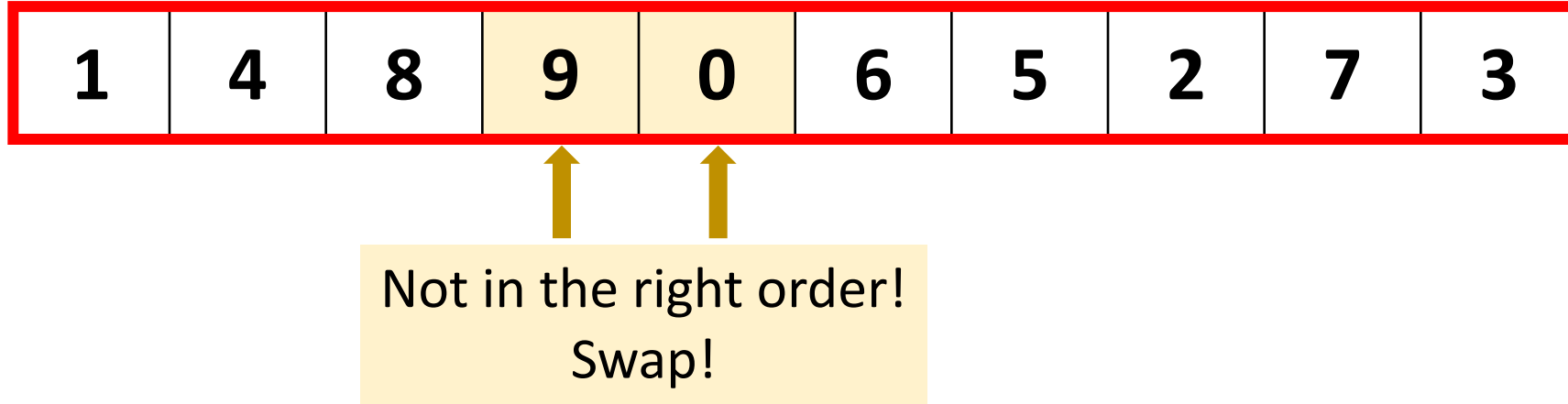
Iteration 1

1	4	8	9	0	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---



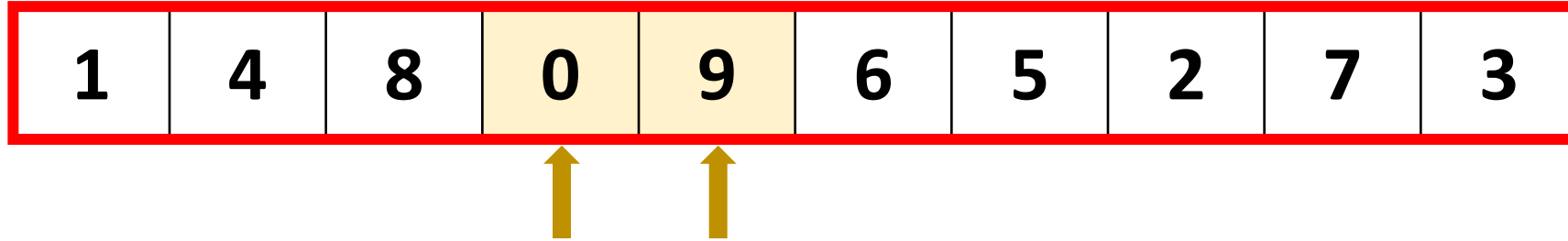
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

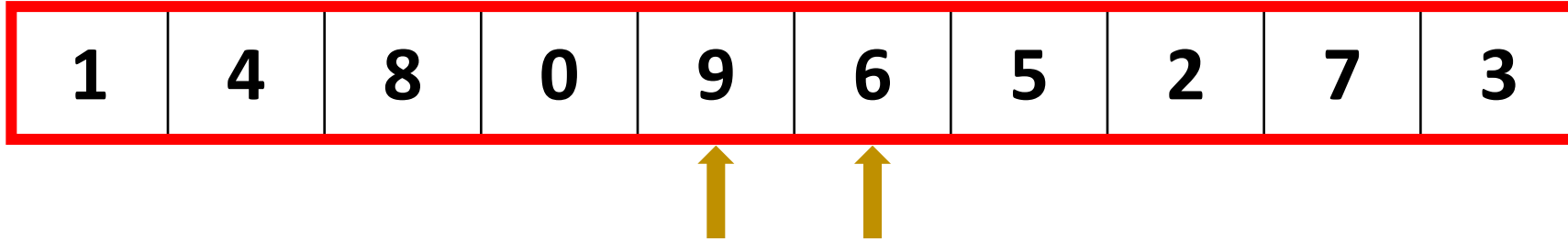
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

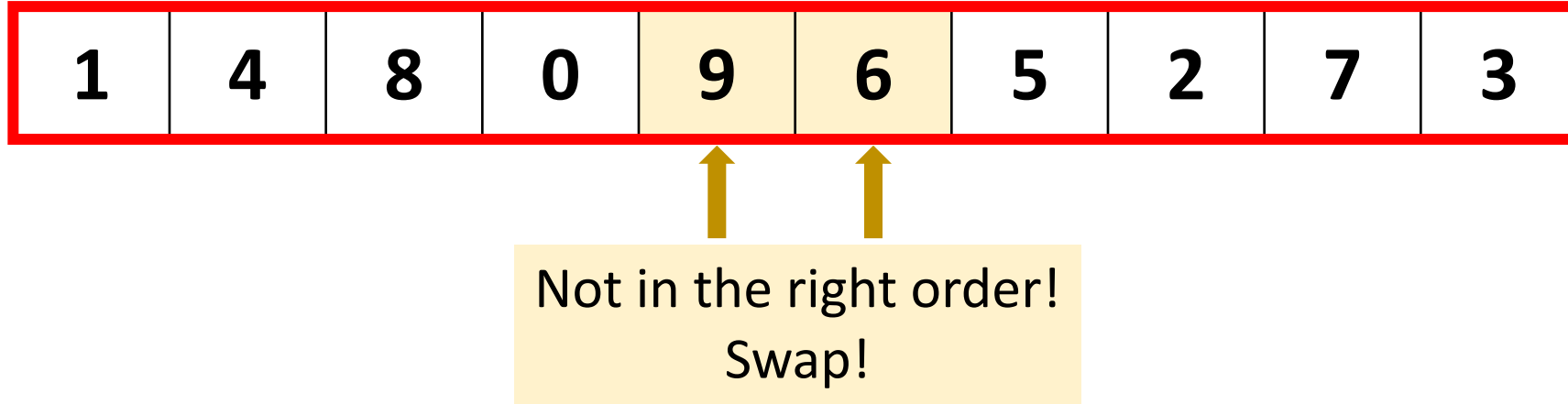
Iteration 1

1	4	8	0	9	6	5	2	7	3
---	---	---	---	---	---	---	---	---	---



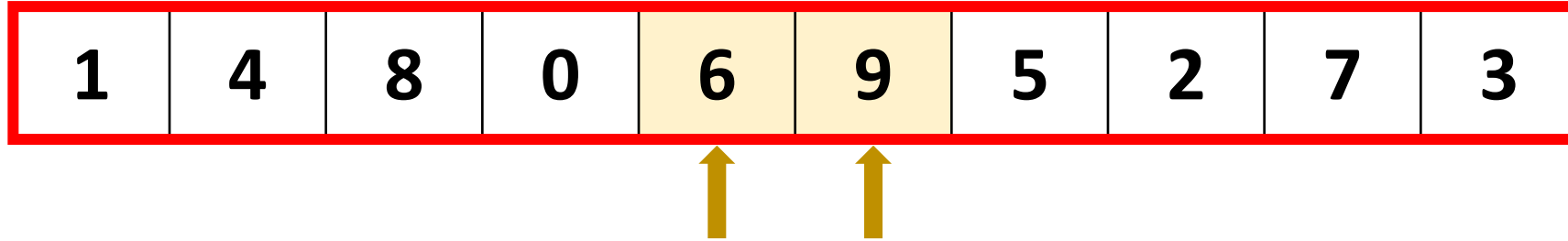
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

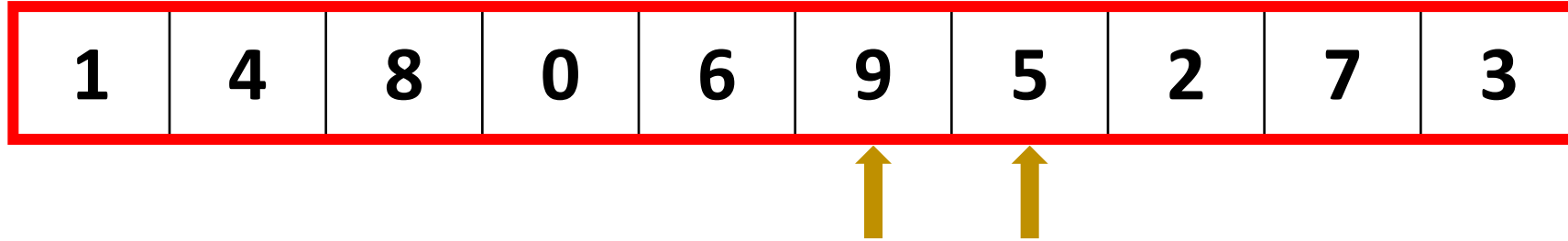
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

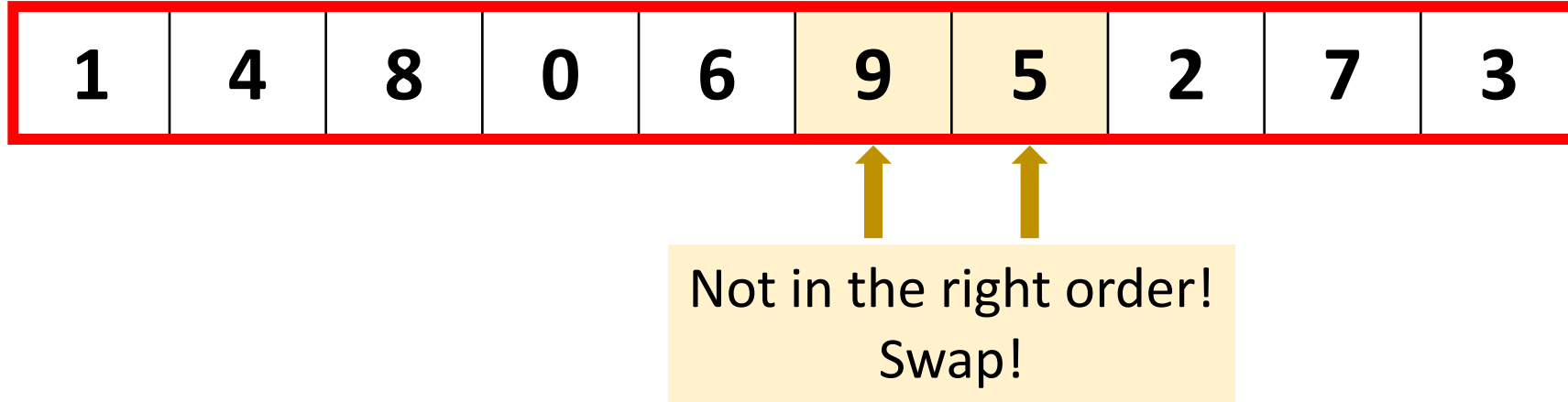
Iteration 1

1	4	8	0	6	9	5	2	7	3
---	---	---	---	---	---	---	---	---	---



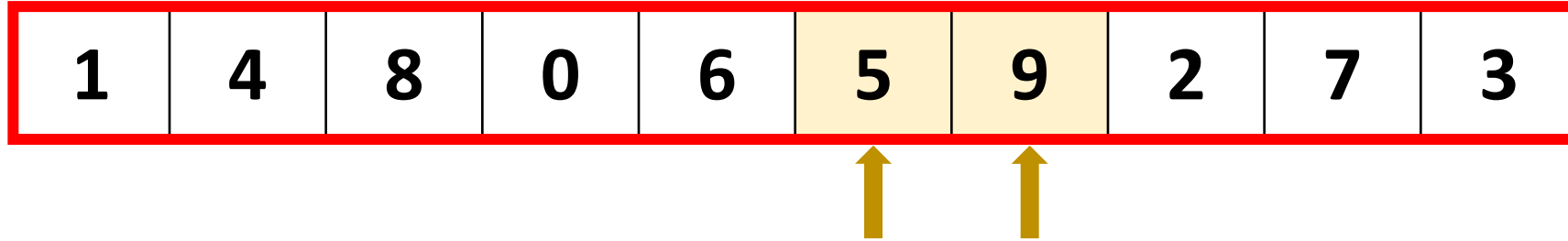
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

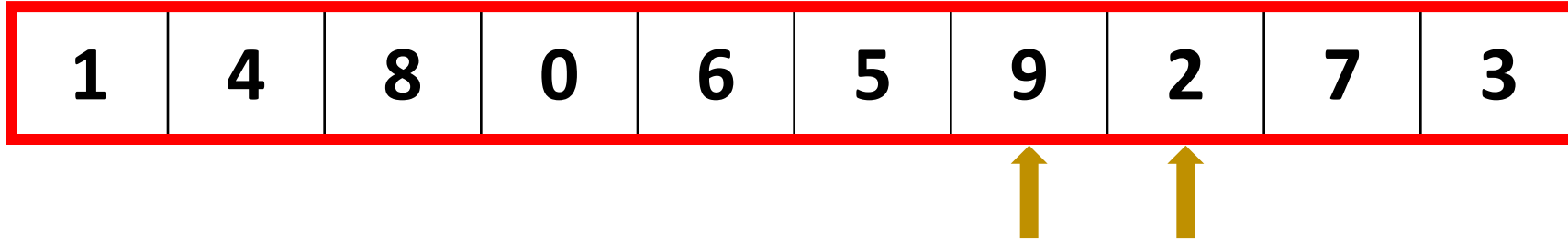
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

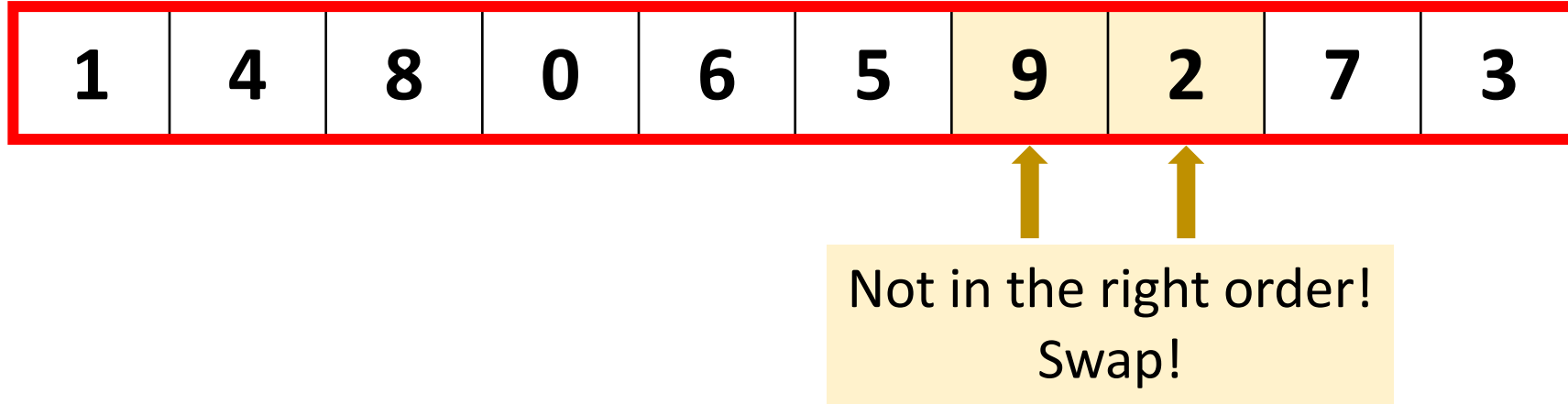
Iteration 1

1	4	8	0	6	5	9	2	7	3
---	---	---	---	---	---	---	---	---	---



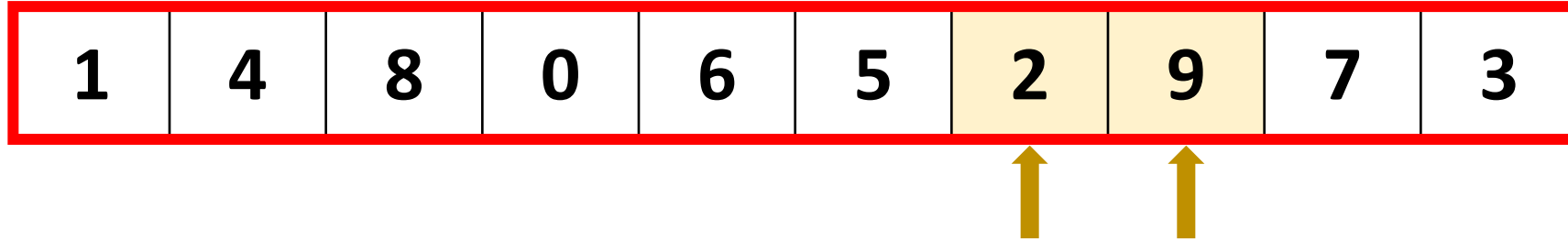
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

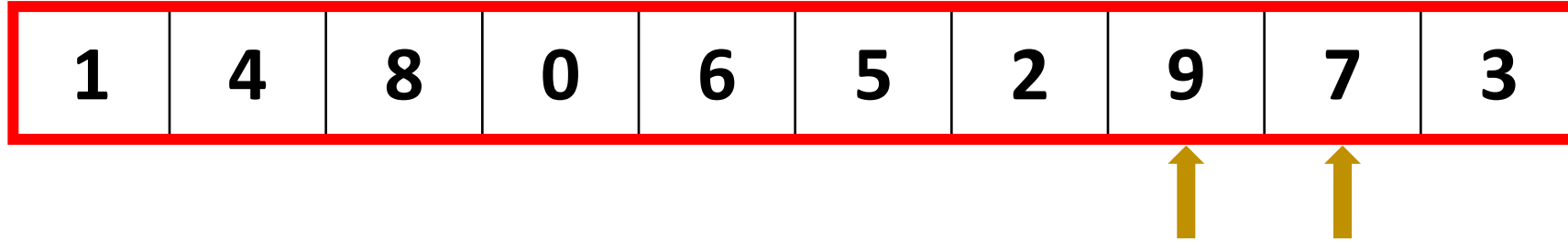
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

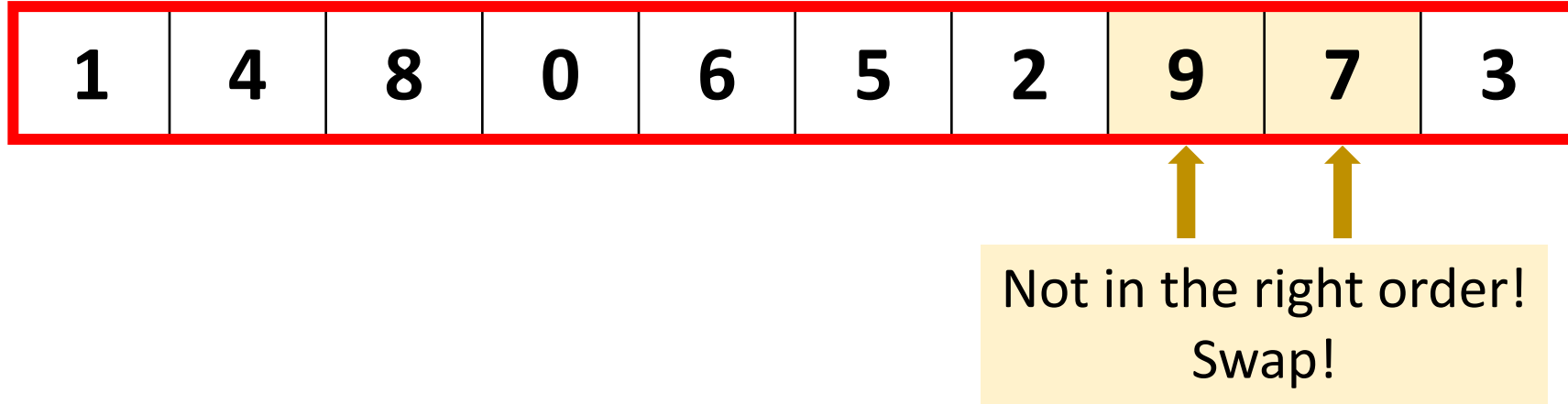
Iteration 1

1	4	8	0	6	5	2	9	7	3
---	---	---	---	---	---	---	---	---	---



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

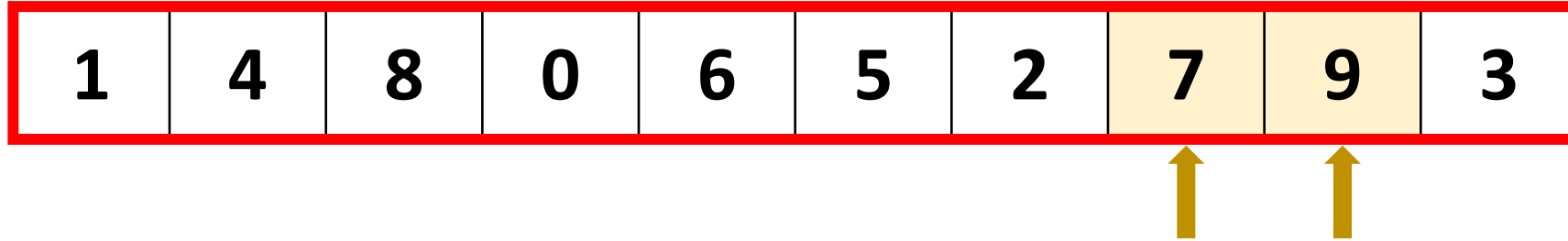
Iteration 1



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1

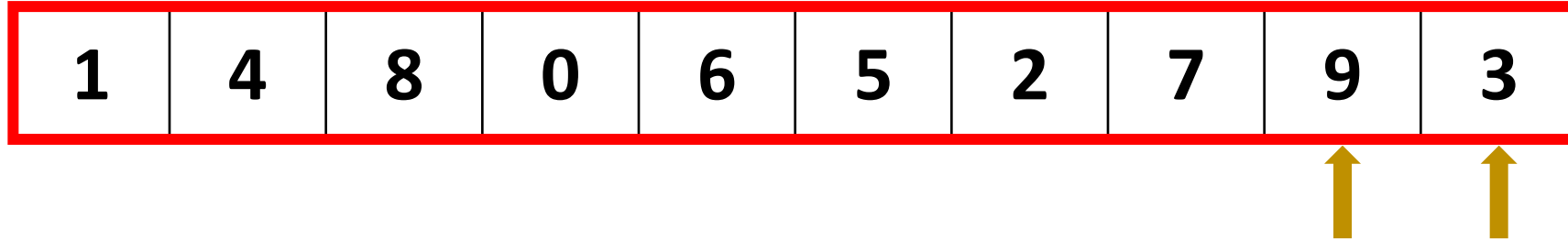
1	4	8	0	6	5	2	7	9	3
---	---	---	---	---	---	---	---	---	---



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1

1	4	8	0	6	5	2	7	9	3
---	---	---	---	---	---	---	---	---	---



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1

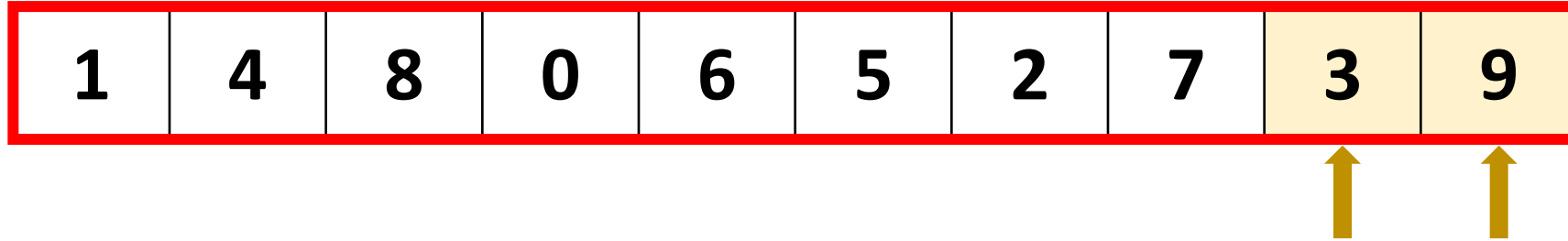
1	4	8	0	6	5	2	7	9	3
---	---	---	---	---	---	---	---	---	---

Not in the right order!
Swap!

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 1

1	4	8	0	6	5	2	7	3	9
---	---	---	---	---	---	---	---	---	---



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 1

1	4	8	0	6	5	2	7	3	9
---	---	---	---	---	---	---	---	---	---



This is the largest number.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2

1	4	8	0	6	5	2	7	3	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 1$ elements.

↑
This is the largest number.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2

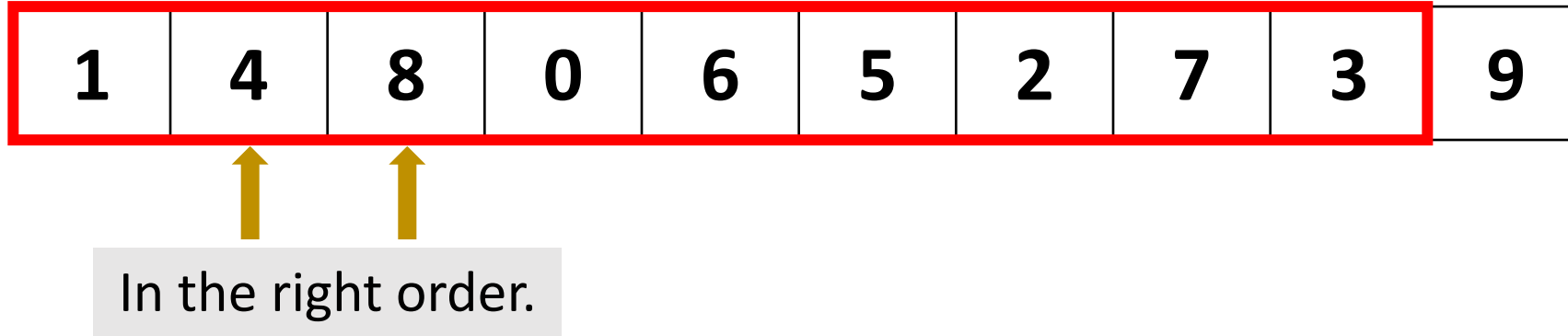
1	4	8	0	6	5	2	7	3	9
---	---	---	---	---	---	---	---	---	---



In the right order.

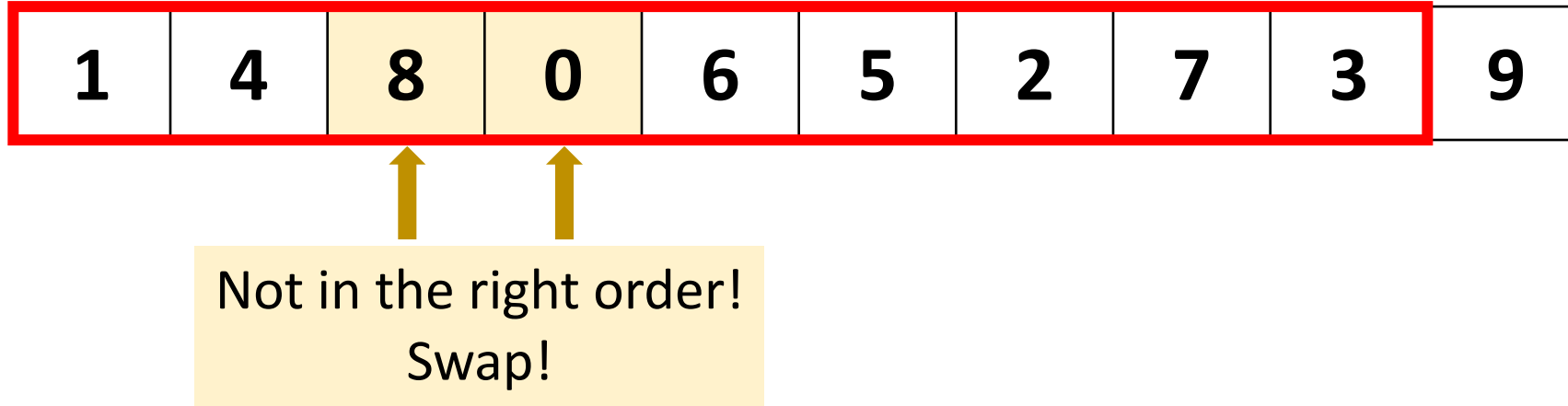
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



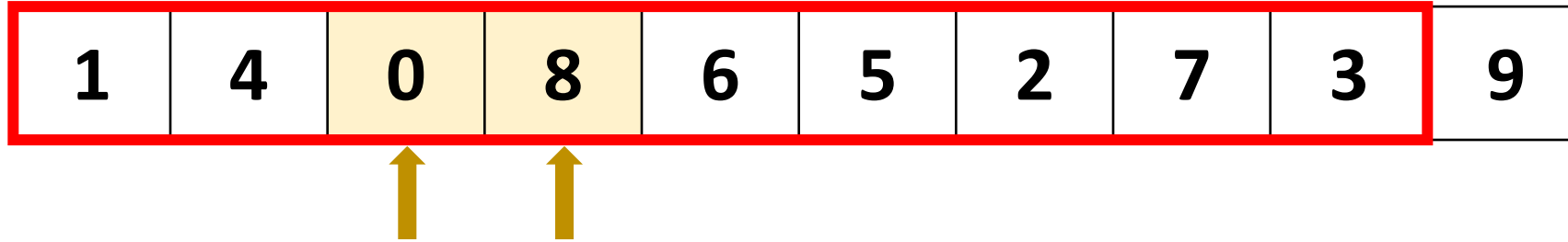
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



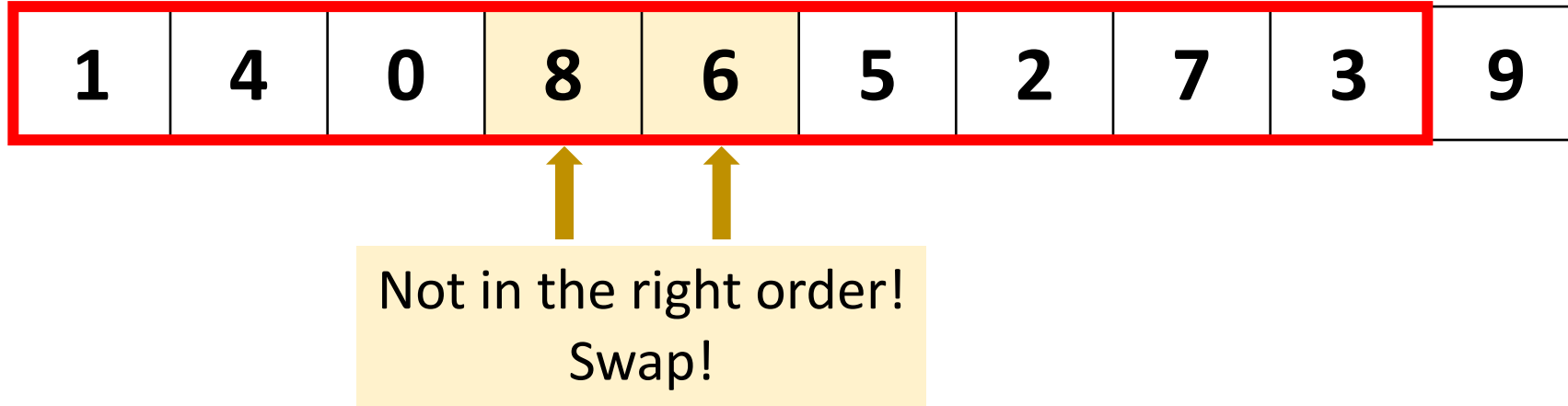
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



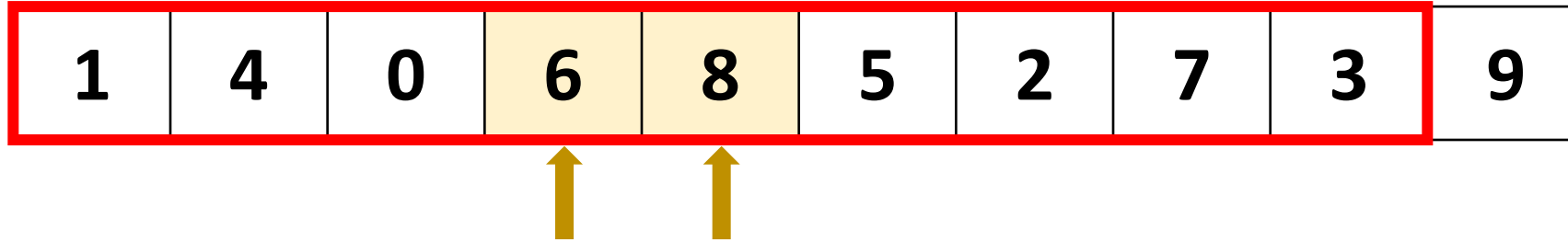
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



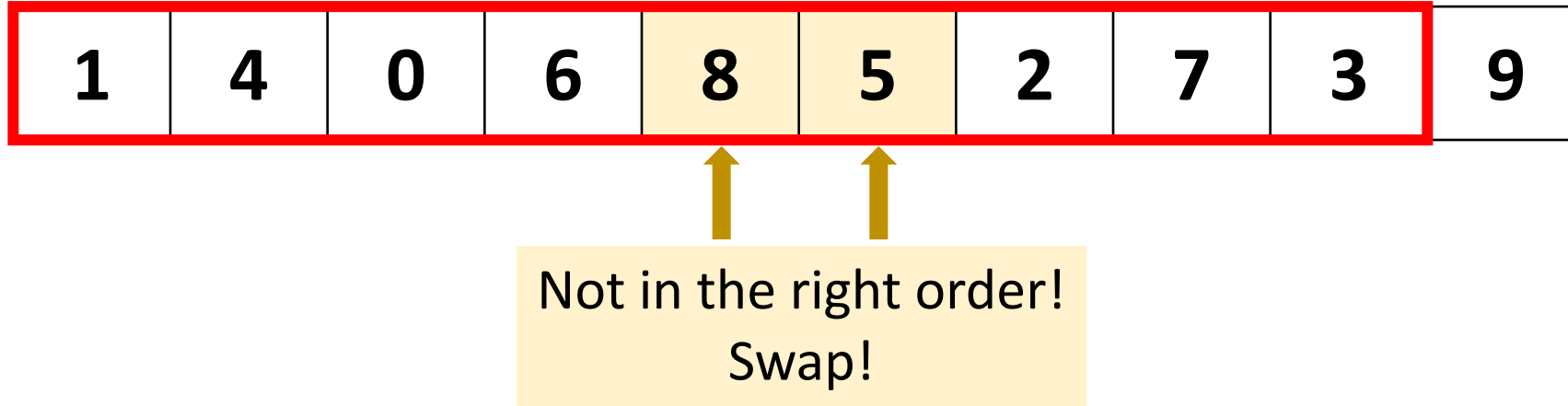
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



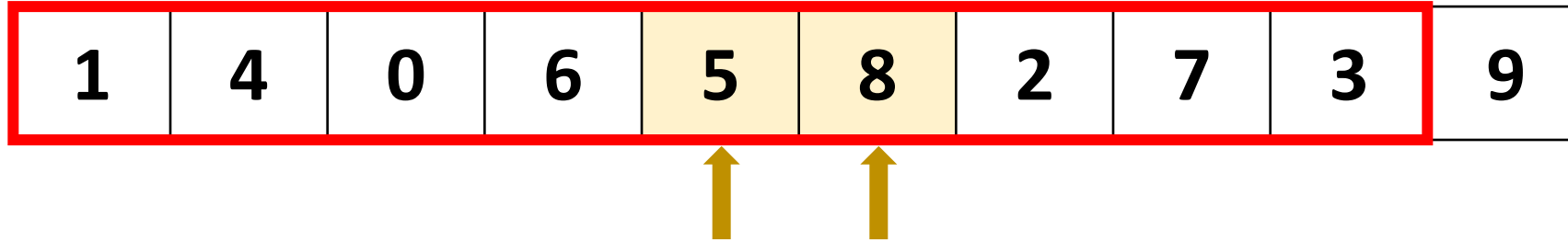
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



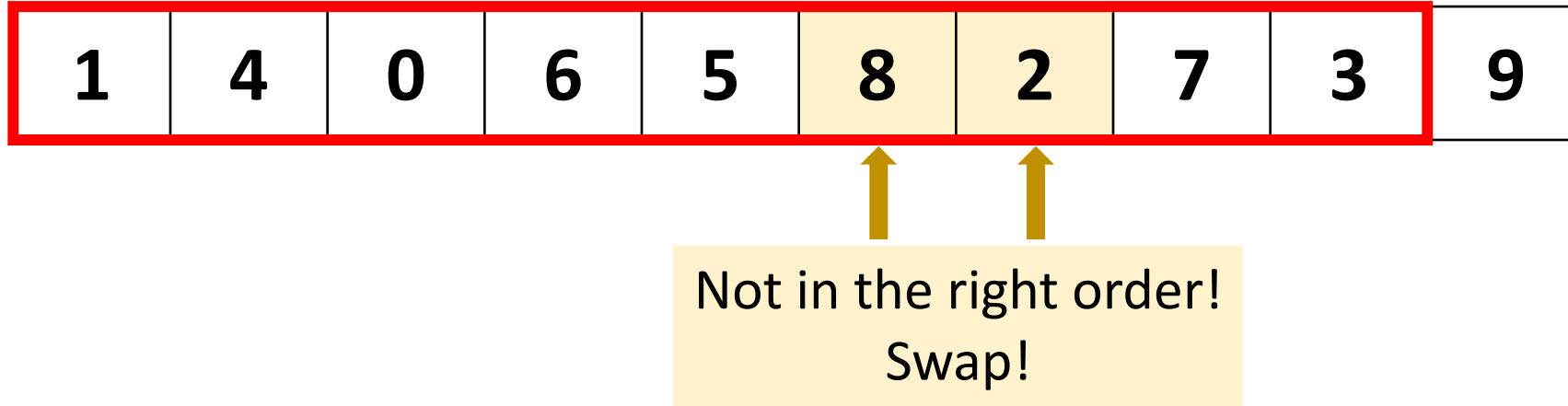
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



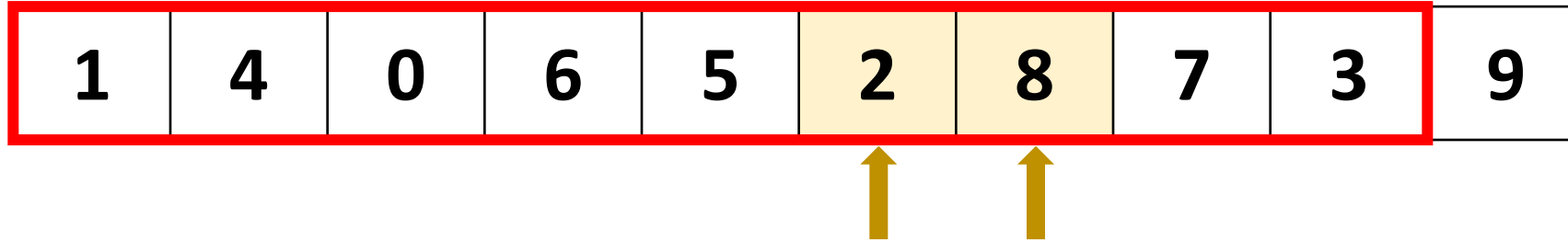
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



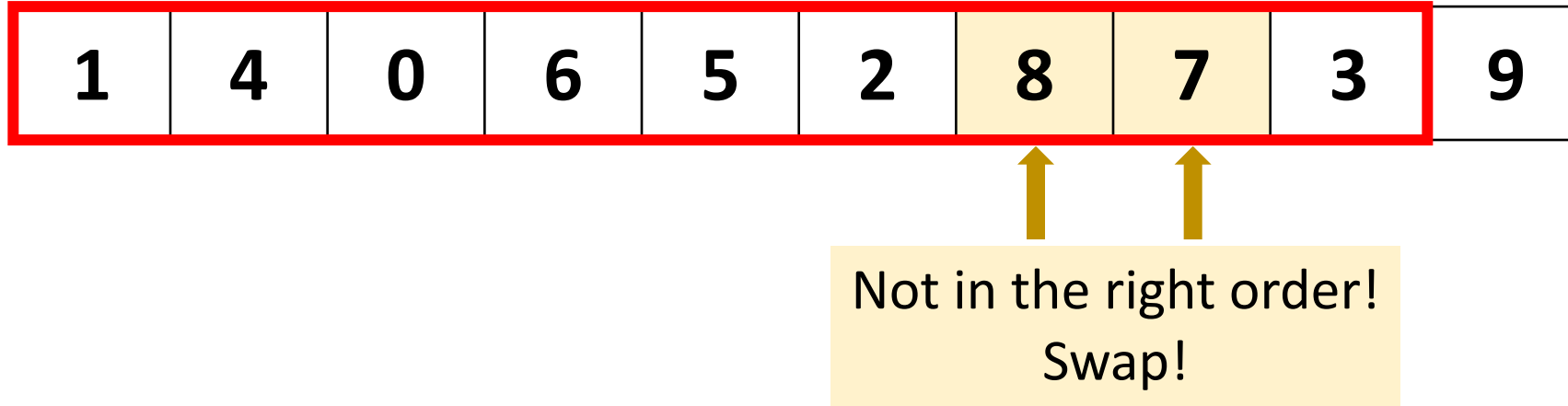
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



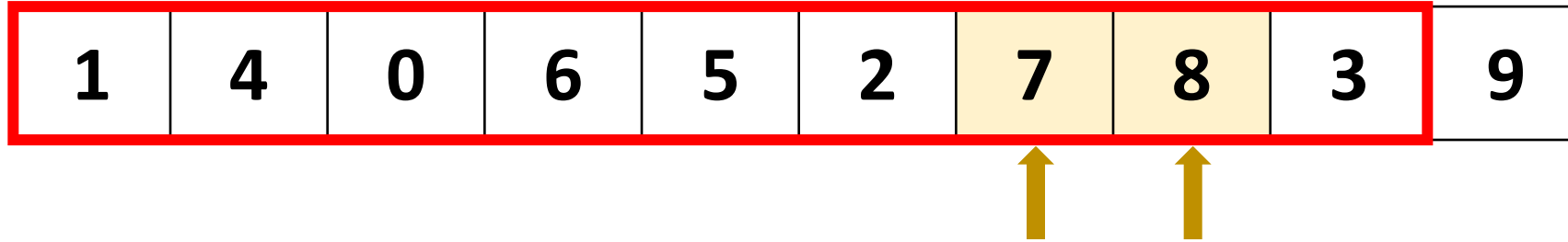
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



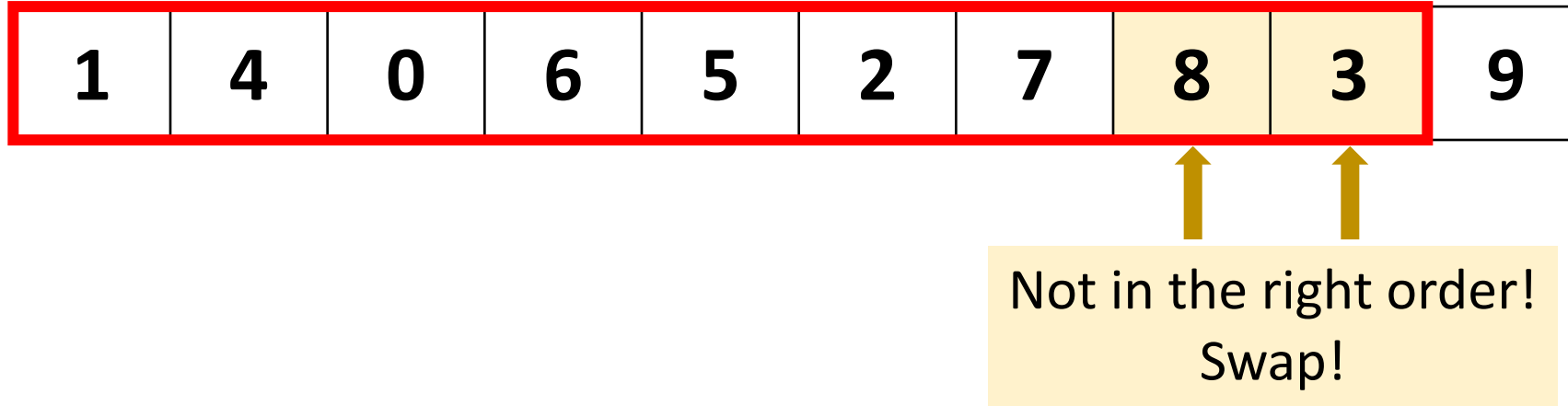
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



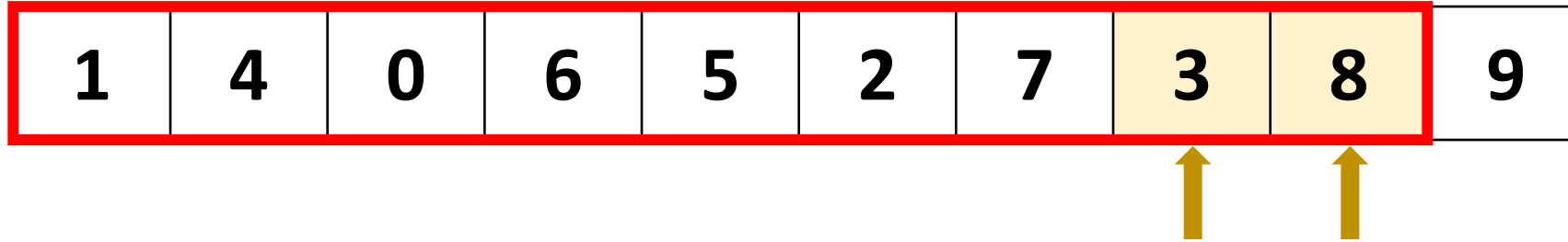
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 2



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 2

1	4	0	6	5	2	7	3	8	9
---	---	---	---	---	---	---	---	---	---



This is the 2nd largest number.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3

1	4	0	6	5	2	7	3	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 2$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3

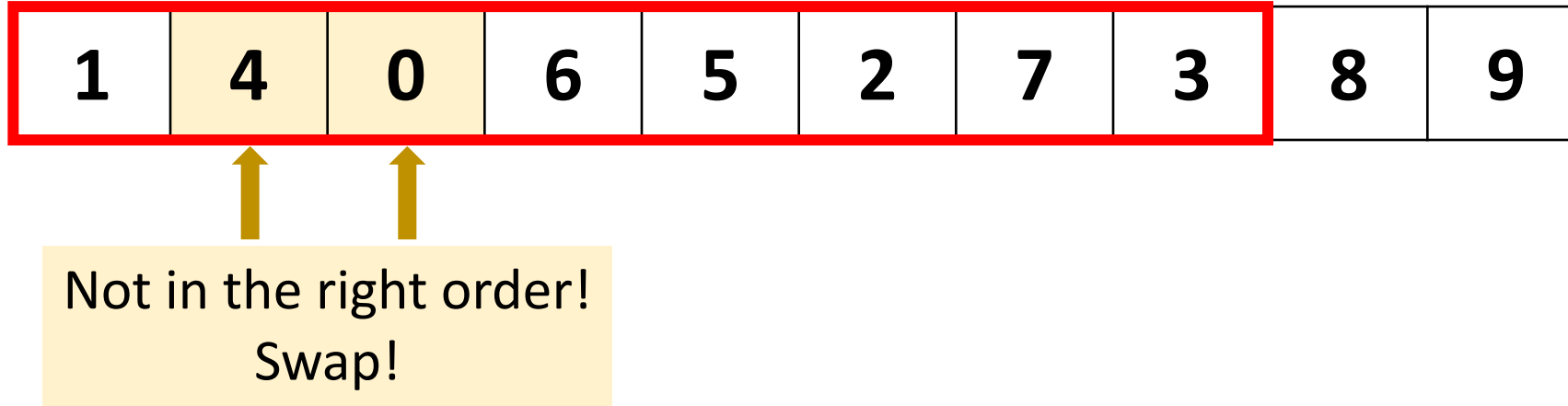
1	4	0	6	5	2	7	3	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

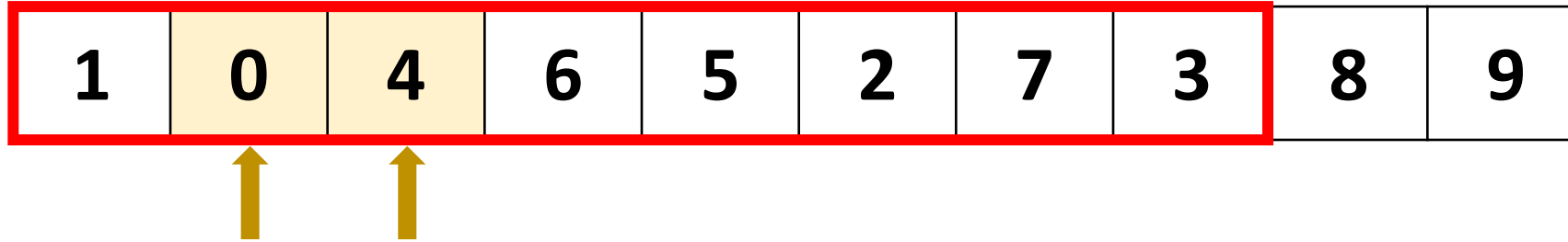
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3

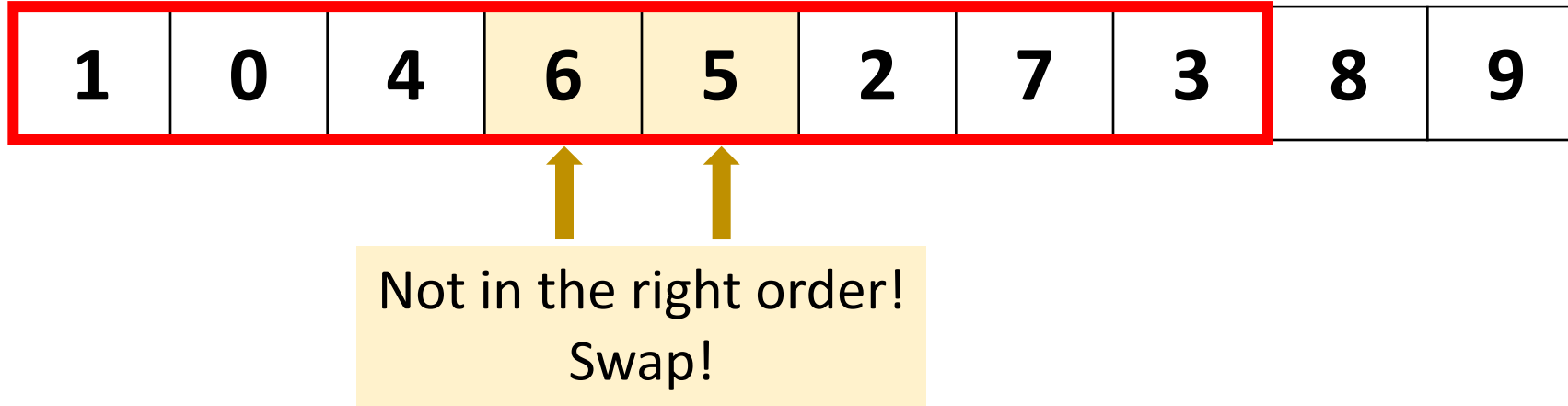
1	0	4	6	5	2	7	3	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

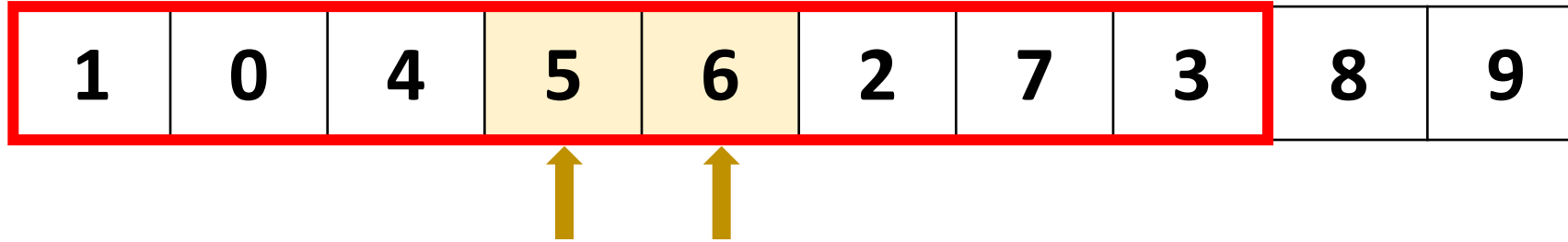
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



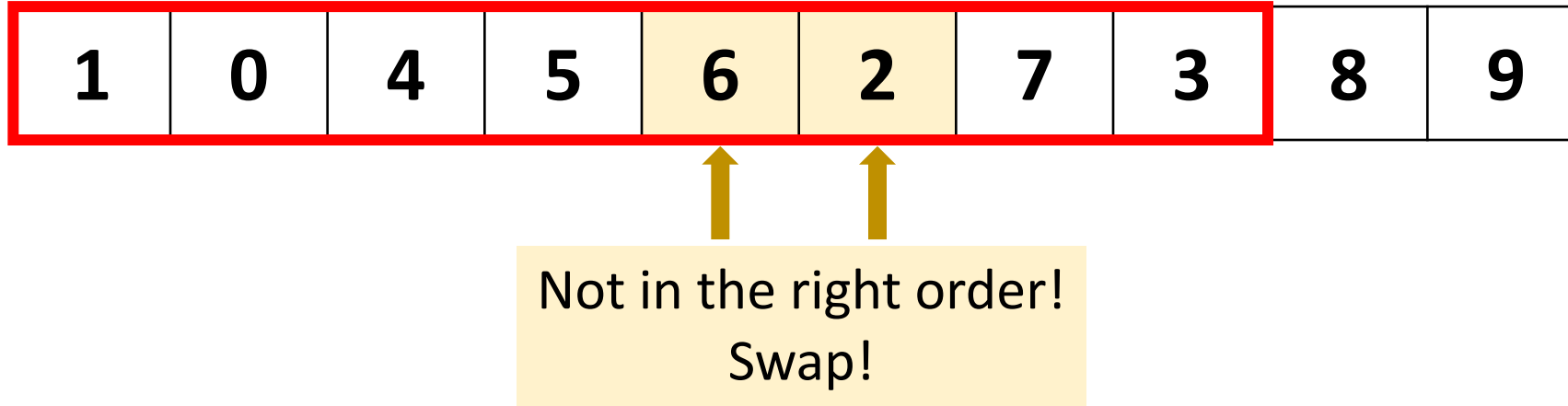
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



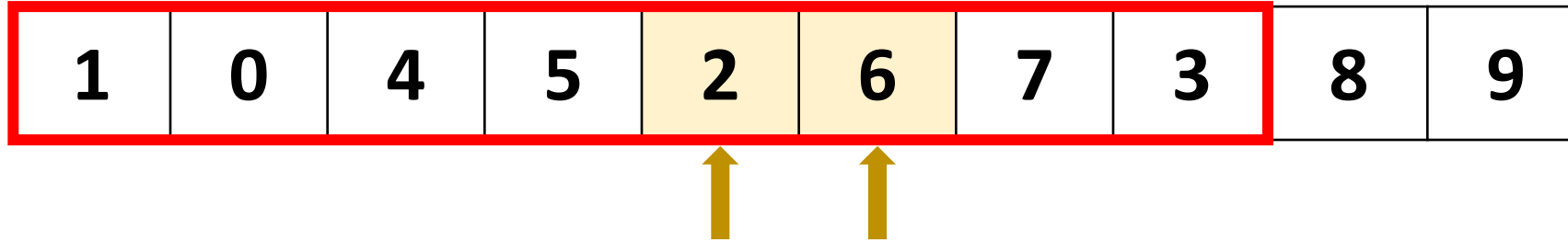
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

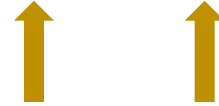
Iteration 3



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3

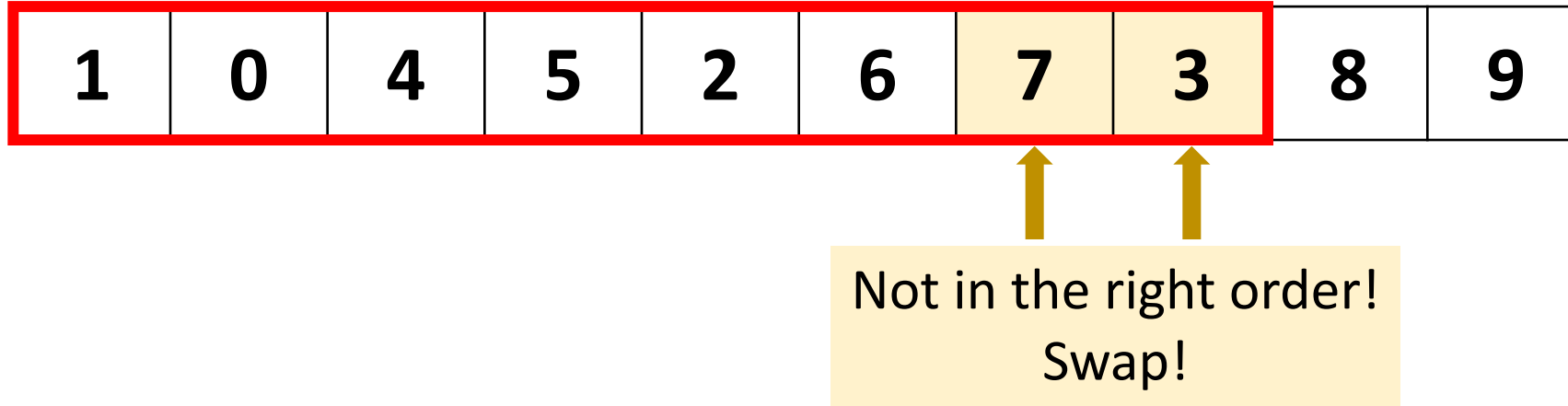
1	0	4	5	2	6	7	3	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

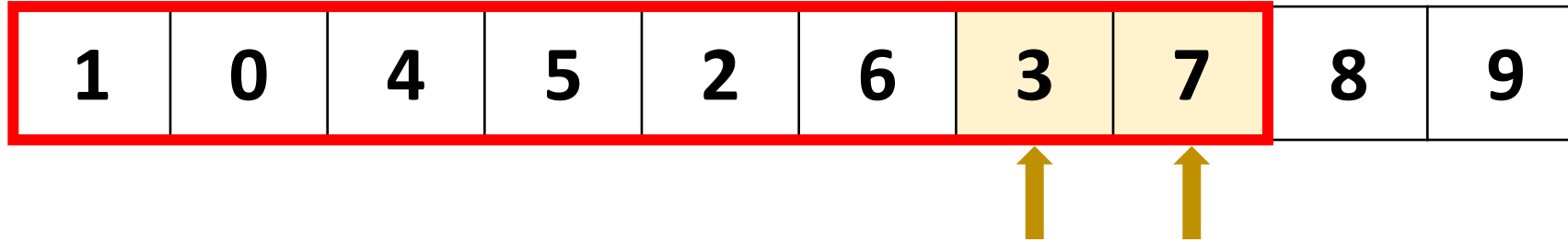
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 3



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 3

1	0	4	5	2	6	3	7	8	9
---	---	---	---	---	---	---	---	---	---



This is the 3rd largest number.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 4

1	0	4	5	2	6	3	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 3$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 4

0	1	4	2	5	3	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 3$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 4

0	1	4	2	5	3	6	7	8	9
---	---	---	---	---	---	---	---	---	---



This is the 4th largest number.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 5

0	1	4	2	5	3	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 4$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 5

0	1	2	4	3	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 4$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 5

0	1	2	4	3	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



This is the 5th largest number.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 6

0	1	2	4	3	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 5$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

After Iteration 6

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 5$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 7

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

Work on the first $n - 6$ elements.

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 7

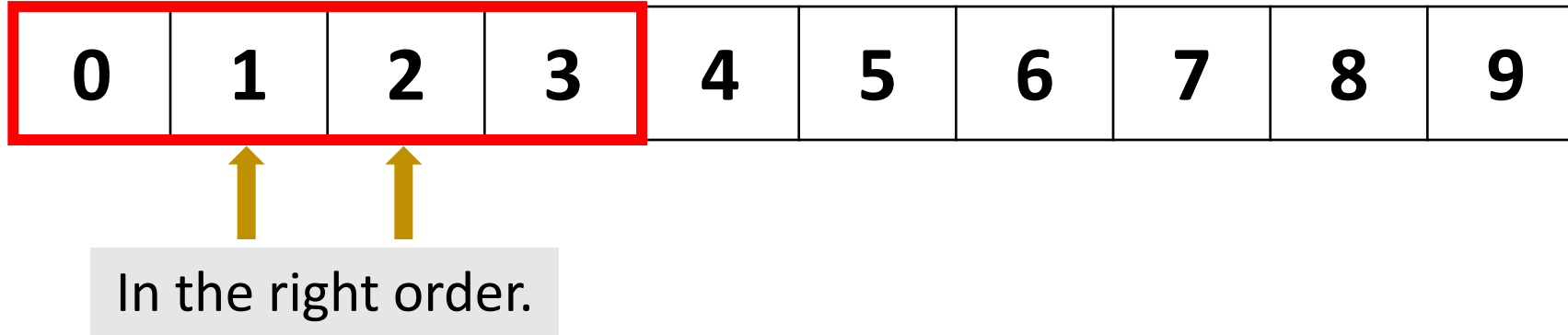
0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

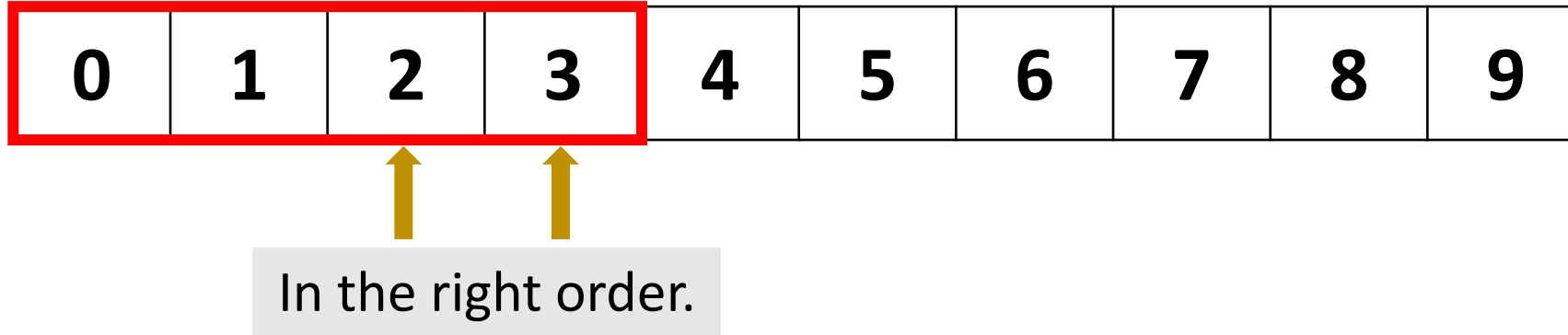
- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 7



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

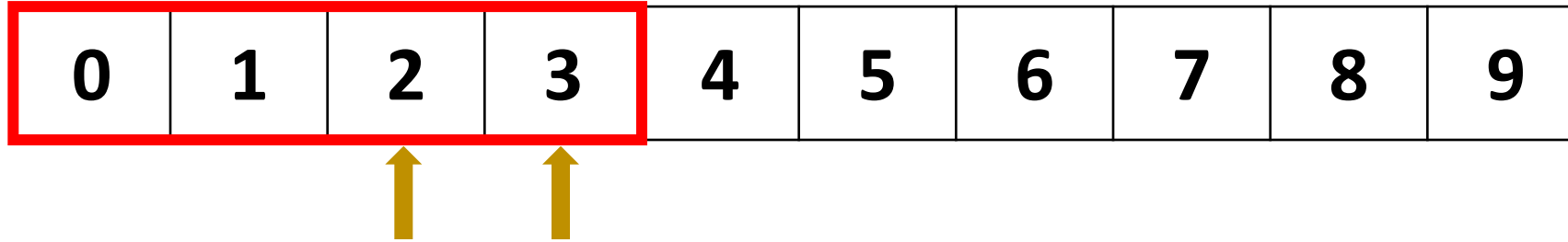
Iteration 7



- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Iteration 7

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



No swap in the 7th iteration → These elements are in the ascending order.

Stop!

- Start from the beginning of the array.
- Compare every adjacent pair.
- Swap them if they are not in the right order.
- After k iterations, the last k elements are the biggest.

Naïve Implementation

```
void bubblesort(int arr[], int n) {  
    int i, j;  
    for (i = 0; i < n-1; i++)  
        for (j = 0; j < n-i-1; j++)  
            if (arr[j] > arr[j+1])  
                swap(&arr[j], &arr[j+1]);  
}
```

Improved Implementation

```
void bubblesort(int arr[], int n) {  
    int i, j;  
    bool swapped;  
    for (i = 0; i < n-1; i++) {  
        swapped = false;  
        for (j = 0; j < n-i-1; j++) {  
            if (arr[j] > arr[j+1]) {  
                swap(&arr[j], &arr[j+1]);  
                swapped = true;  
            }  
        }  
        if (swapped == false) break;  
    }  
}
```


Time Complexity

Worst-Case Time Complexity

6	5	4	3	2	1	0	7	8	9
---	---	---	---	---	---	---	---	---	---

Iteration 4: Work on the first $n - 3$ elements.

- The k -th iteration performs $n - k$ operations.
- In the worst case, $n - 1$ iterations are needed.
- Time complexity:

$$T(n) = \sum_{k=1}^{n-1} (n - k) = O(n^2).$$

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

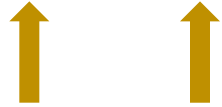


In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---



In the right order.

- In the **best case**, the input array is in the ascending order.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

- In the **best case**, the input array is in the ascending order.
- No swap during the 1st iteration.
- Thus we know the array is already sorted.

Best-Case Time Complexity

0	1	2	3	4	5	6	7	8	9
---	---	---	---	---	---	---	---	---	---

- In the **best case**, the input array is in the ascending order.
- No swap during the 1st iteration.
- Thus we know the array is already sorted.
- Return after the 1st iteration.
- Best-case time complexity: $O(n)$.

Average-Case Time Complexity

- **Best case:** the initial input is in the right order.
- **Worst case:** the initial input is in the reverse order.
- **Average case:** the input array is like random shuffled.
- Time complexity: $O(n^2)$.

Thank you!