Sorting Algorithms

- ❑ **Sorting Algorithm** is an algorithm made up of a series of instructions that takes an array as input, and outputs a sorted array.

- ❑ There are many sorting algorithms, such as:
  - Selection Sort, Bubble Sort, Insertion Sort, Merge Sort, Heap Sort, QuickSort, Radix Sort, Counting Sort, Bucket Sort, ShellSort, Comb Sort, Pigeonhole Sort, Cycle Sort

Bubble Sort

Bubble Sort

- Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order.

Bubble Sort

❑ **Algorithm:**

- **Step1**: Compare each pair of adjacent elements in the list
- **Step2**: Swap two element if necessary
- **Step3**: Repeat this process for all the elements until the entire array is sorted

Bubble Sort

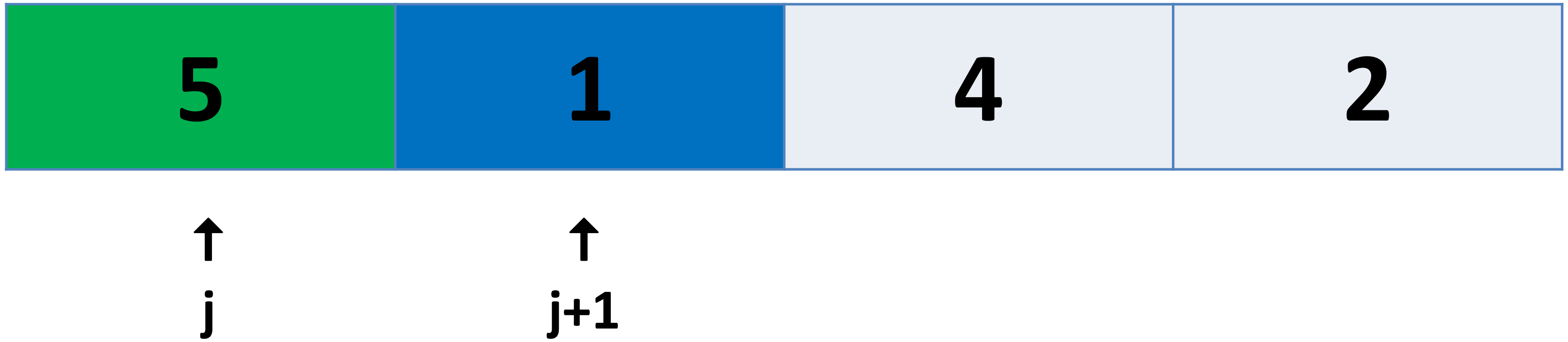
- Example 1 Assume the following Array:

5	1	4	2

Bubble Sort

❑ **First Iteration:**

❑ Compare

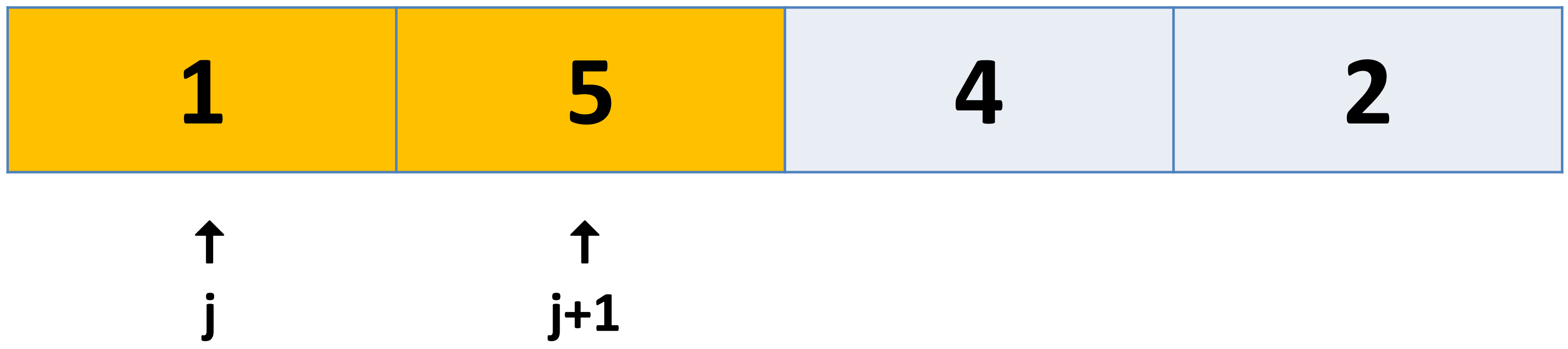


5	1	4	2
↑ j	↑ j+1		

Bubble Sort

□ First Iteration:

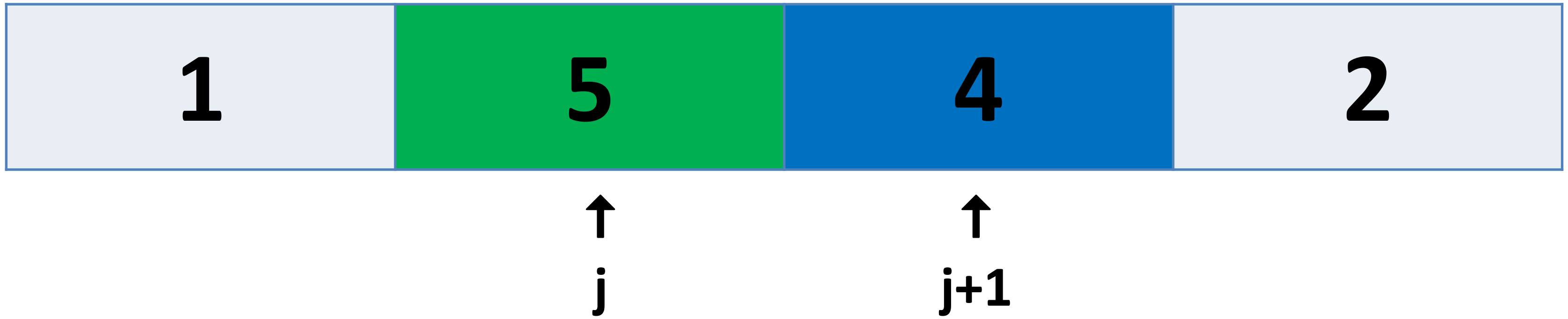
□ Swap



1	5	4	2

Bubble Sort

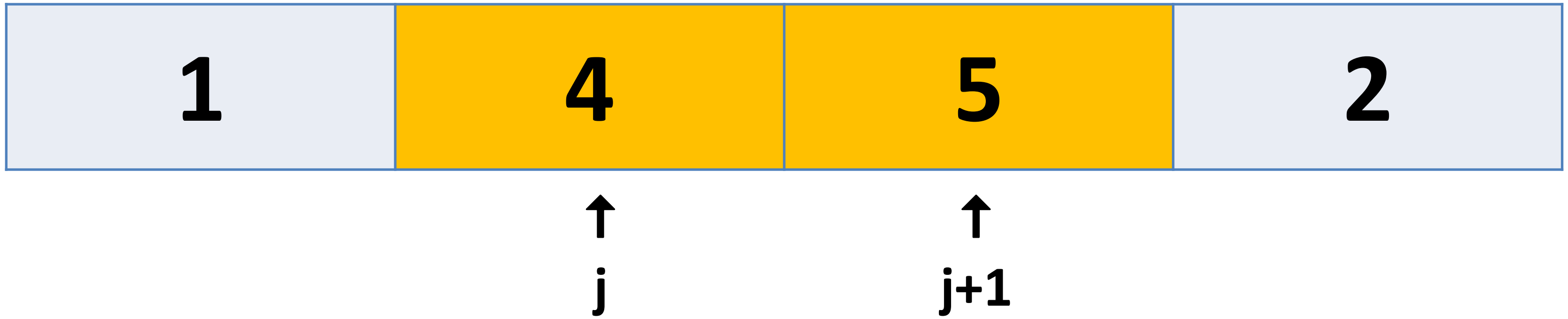
- □ First Iteration:
- □ Compare



1	5	4	2

Bubble Sort

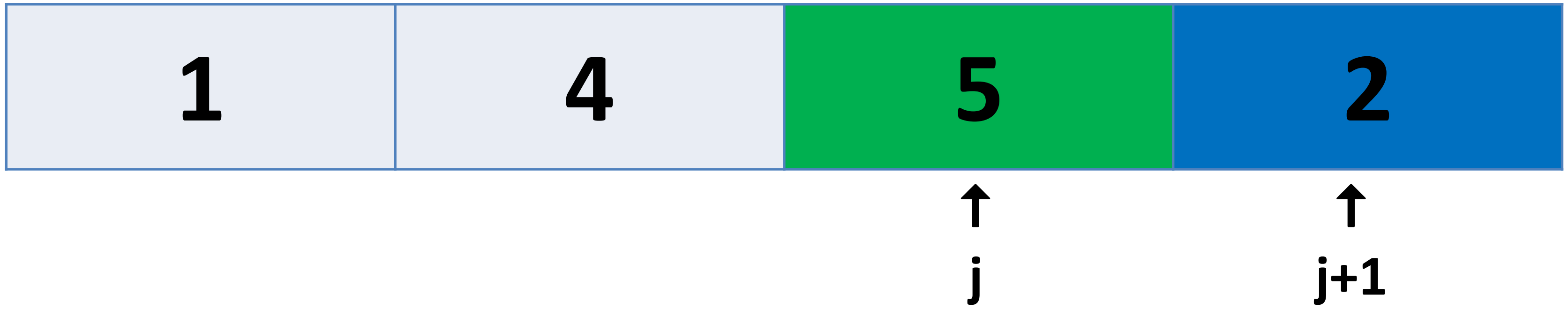
- ❑ First Iteration:
- ❑ Swap



1	4	5	2
	↑ j	↑ j+1	

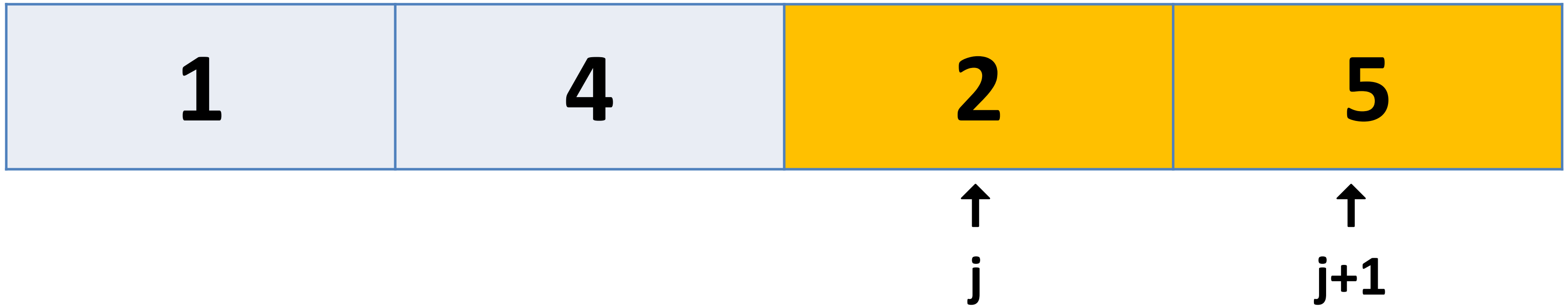
Bubble Sort

- □ First Iteration:
- □ Compare



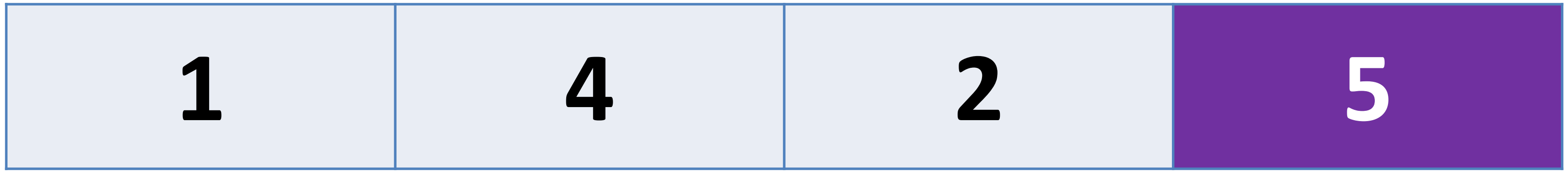
Bubble Sort

- First Iteration:
- Swap



1	4	2	5

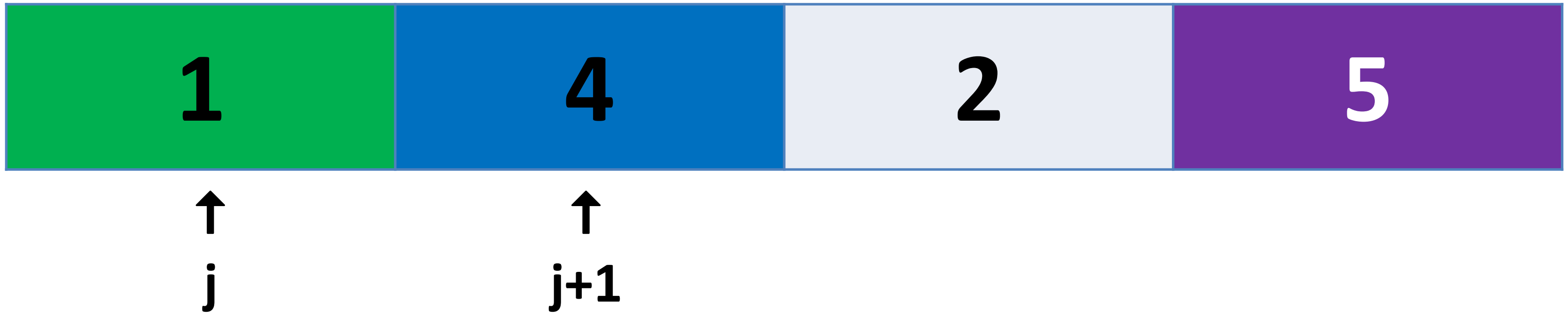
Bubble Sort



1	4	2	5

Bubble Sort

- ❑ Second Iteration:
- ❑ Compare

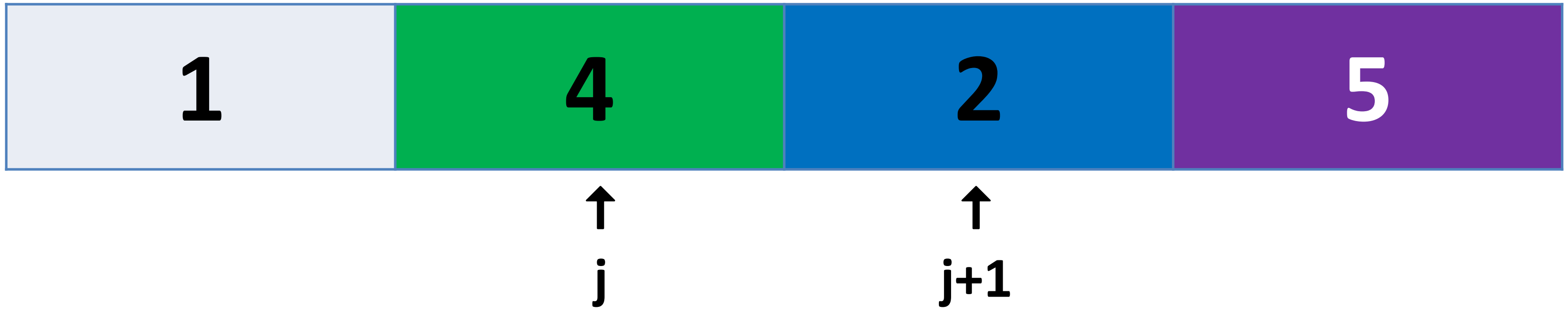


1	4	2	5
j	j+1		

Bubble Sort

- ❑ **Second Iteration:**

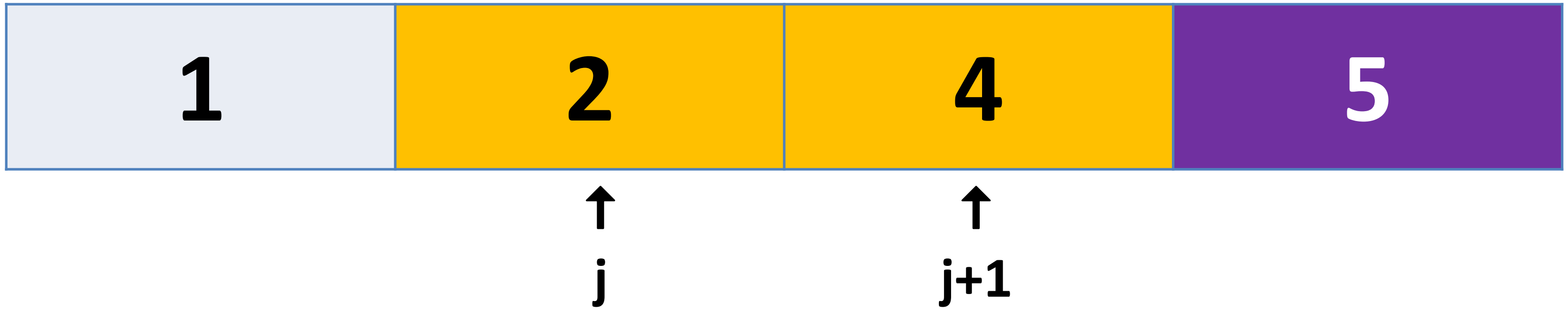
- ❑ Compare



1	4	2	5

Bubble Sort

- ❑ Second Iteration:
- ❑ Swap



1 2 4 5

j j+1

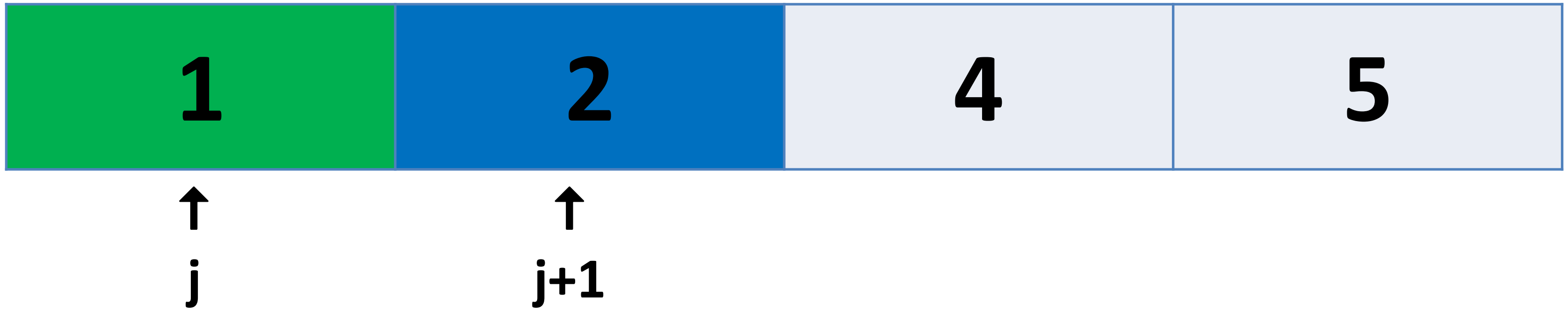
Bubble Sort

1	2	4	5

Bubble Sort

- ❑ **Third Iteration:**

- ❑ Compare



1	2	4	5

Bubble Sort

1	2	4	5

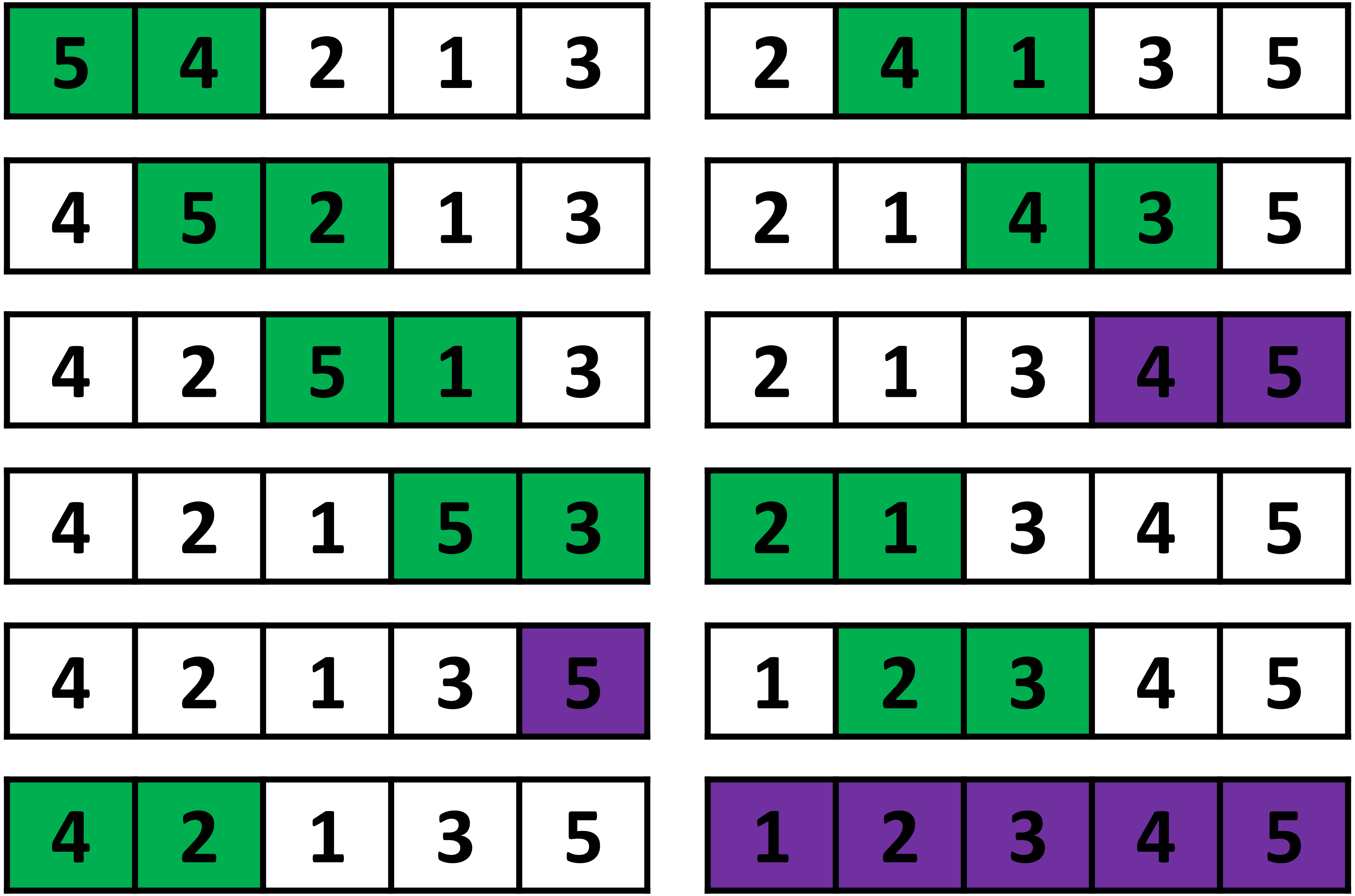
Bubble Sort

❑ Array is now sorted

1	2	3	4

Bubble Sort

- ❑ Example 2:



5 4 2 1 3

4 5 2 1 3

4 2 5 1 3

4 2 1 5 3

4 2 1 3 5

4 2 1 3 5

2 4 1 3 5

2 1 4 3 5

2 1 3 4 5

2 1 3 4 5

1 2 3 4 5

1 2 3 4 5

Bubble Sort

- ❑ What is the output of bubble sort after the 1st iteration given the following sequence of numbers: **13 2 9 4 18 45 37 63**

  a) 2 4 9 13 18 37 45 63

  b) 2 9 4 13 18 37 45 63

  c) 13 2 4 9 18 45 37 63

  d) 2 4 9 13 18 45 37 63

Bubble Sort

- □ What is the output of bubble sort after the 1st iteration given the following sequence of numbers: **13 2 9 4 18 45 37 63**

  a) 2 4 9 13 18 37 45 63

  b) 2 9 4 13 18 37 45 63

  c) 13 2 4 9 18 45 37 63

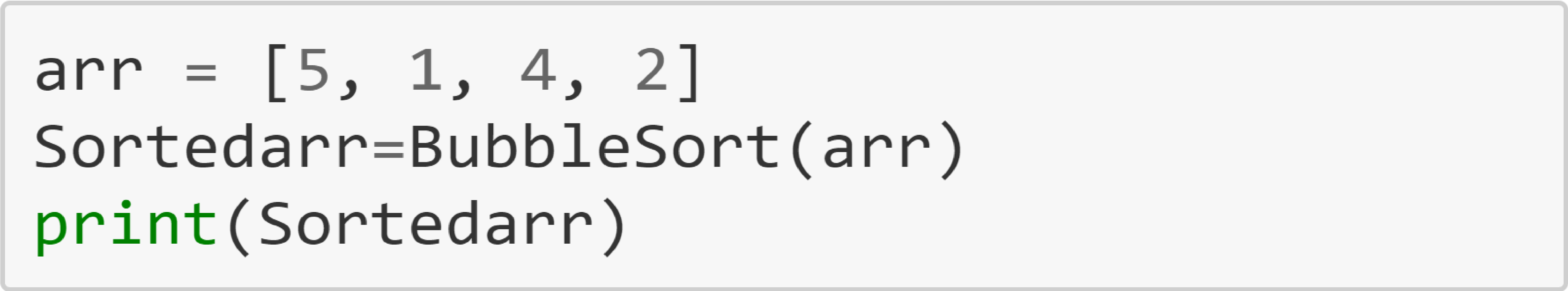
  d) 2 4 9 13 18 45 37 63

Bubble Sort

- ❑ Python Code

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

Bubble Sort



```
arr = [5, 1, 4, 2]
Sortedarr=BubbleSort(arr)
print(Sortedarr)
```

Bubble Sort

- **Time Complexity:** $O(n^2)$ as there are two nested loops

- Example of worst case

5	**4**	**3**	**2**	**1**

Selection Sort

Selection Sort

- ❑The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning.

Selection Sort

- **Algorithm:**

- **Step1**: Find the minimum value in the list
- **Step2:** Swap it with the value in the current position
- **Step3:** Repeat this process for all the elements until the entire array is sorted

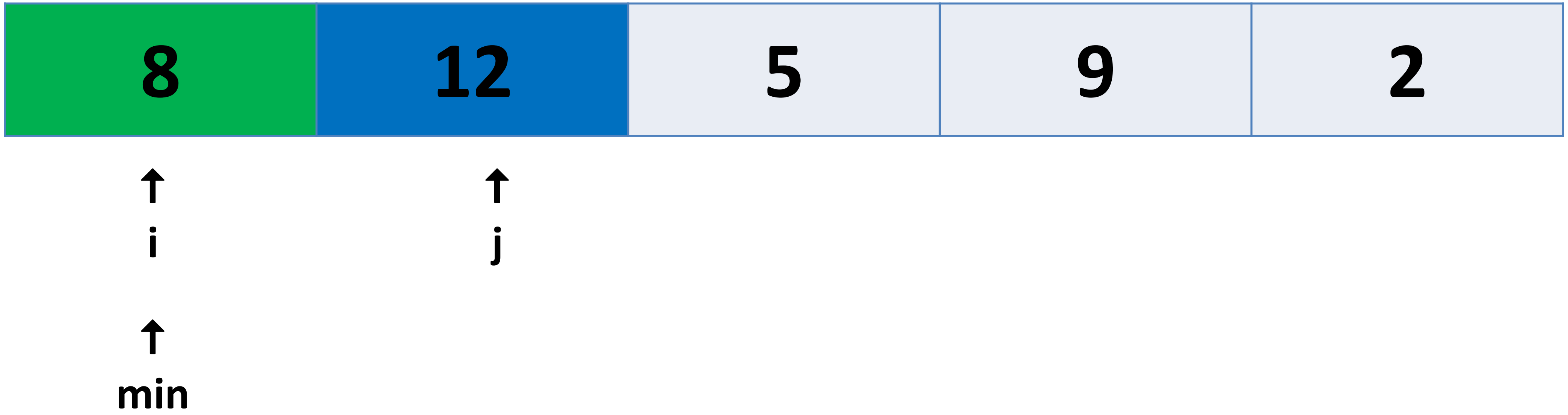
Selection Sort

- Example 1 Assume the following Array:

8	12	5	9	2

Selection Sort

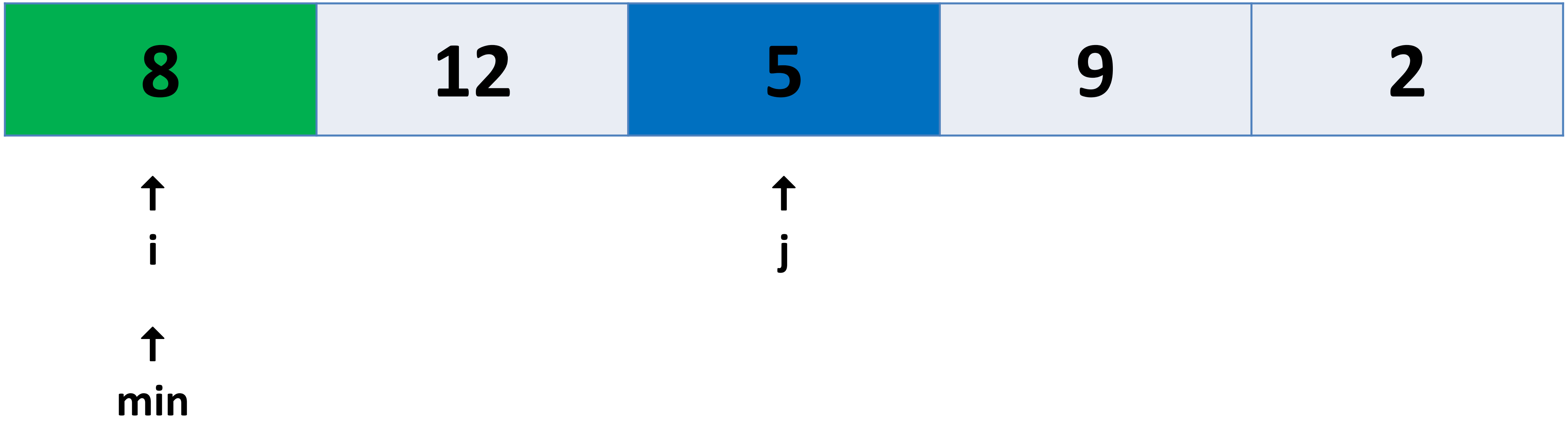
- ❑ Compare



8	12	5	9	2

Selection Sort

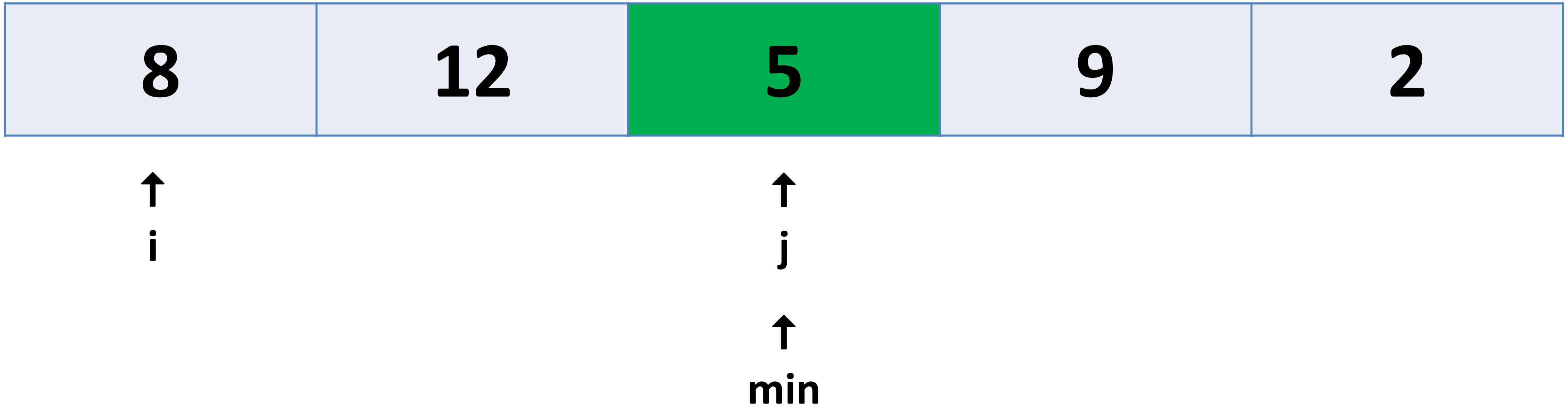
- Compare



8	12	5	9	2

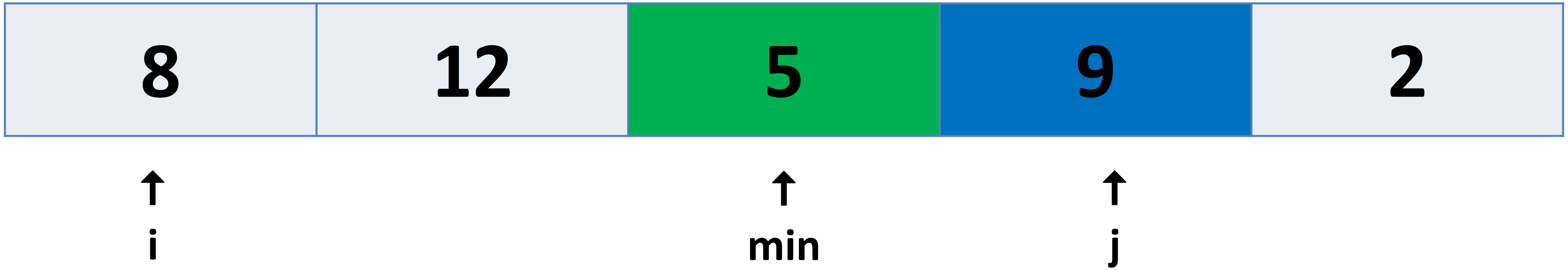
Selection Sort

❑ Move



Selection Sort

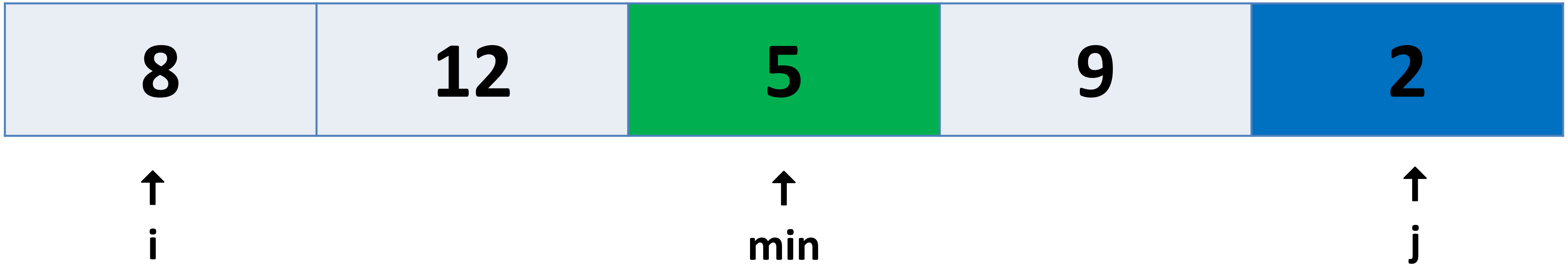
- ❑ Compare



8	12	5	9	2
↑ i		↑ min	↑ j	

Selection Sort

- ❑ Compare



8	12	5	9	2

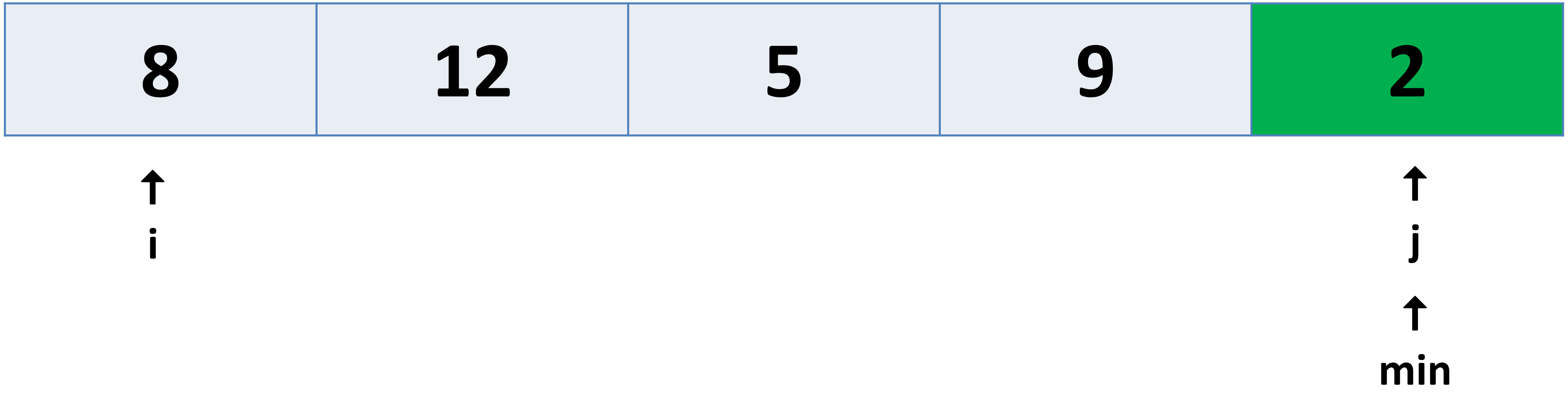
i

min

j

Selection Sort

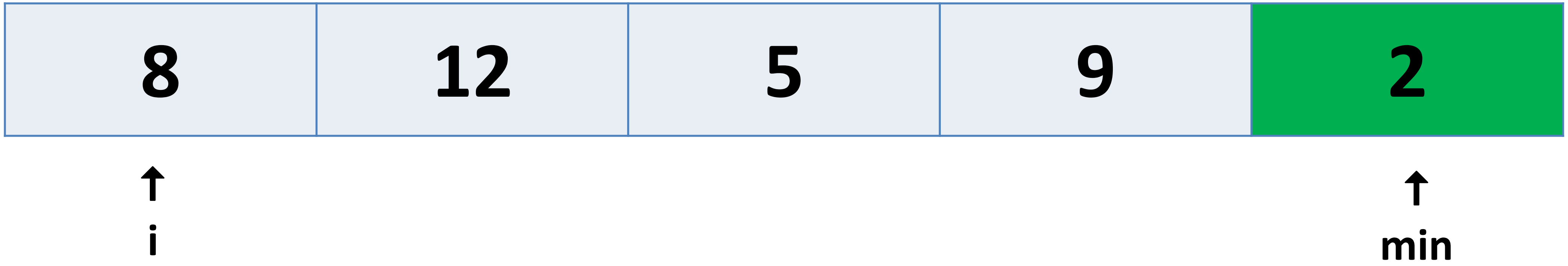
- Move



8	12	5	9	2

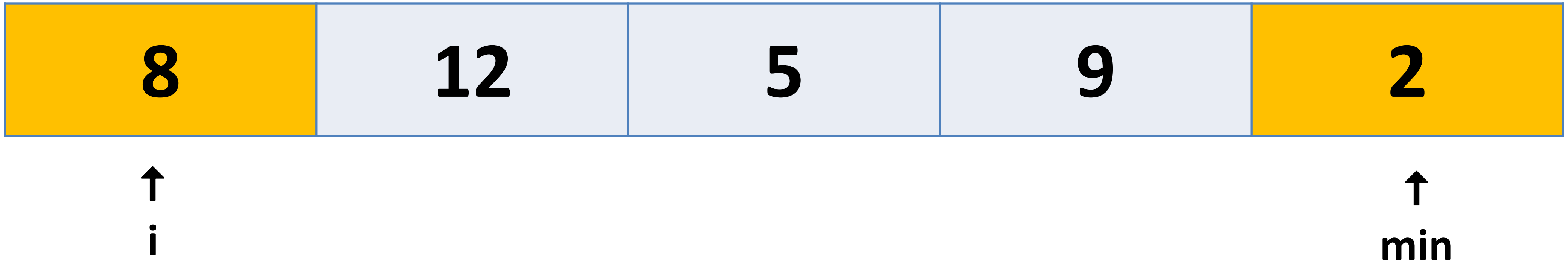
Selection Sort

- ❑ Smallest



Selection Sort

- ❑ Swap

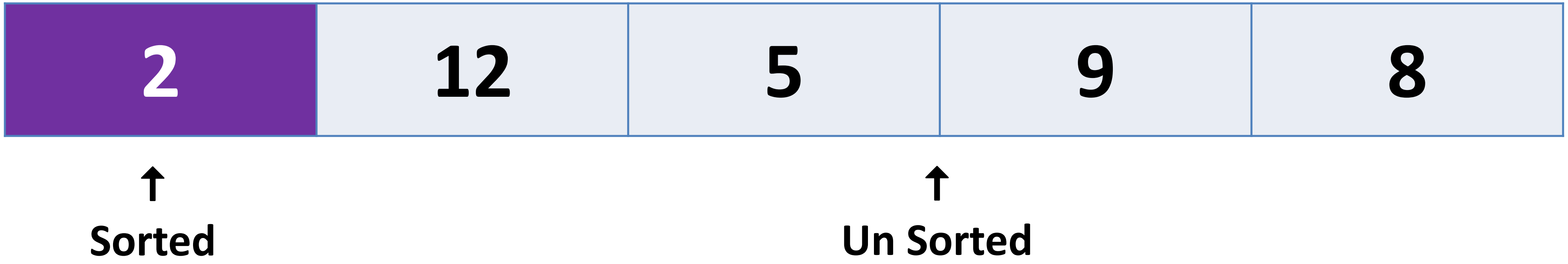


8	12	5	9	2
↑ i				↑ min

Selection Sort

□ Sorted

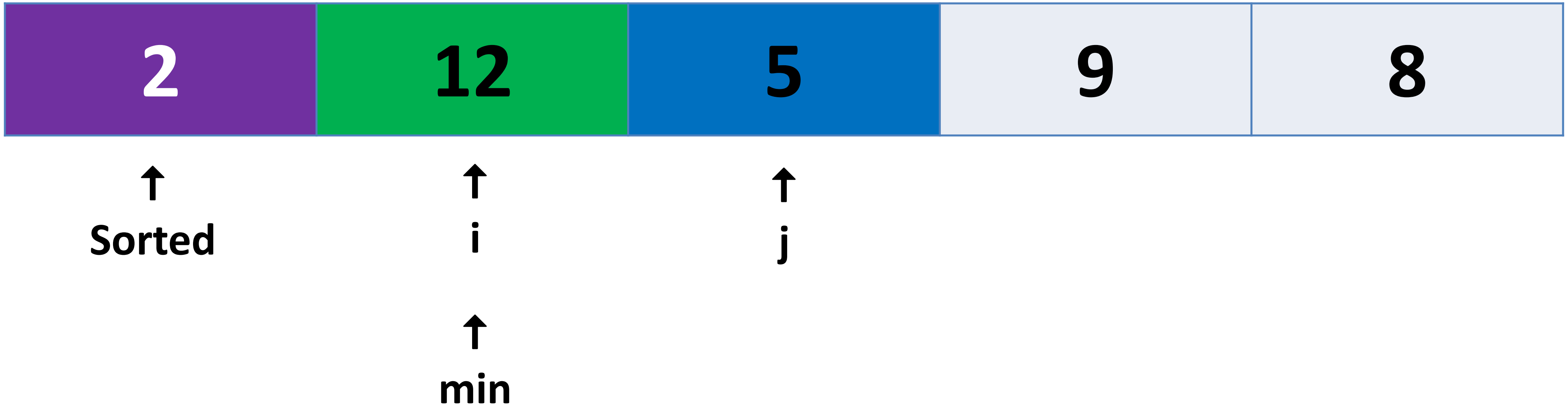
□ Un Sorted



2	12	5	9	8

Selection Sort

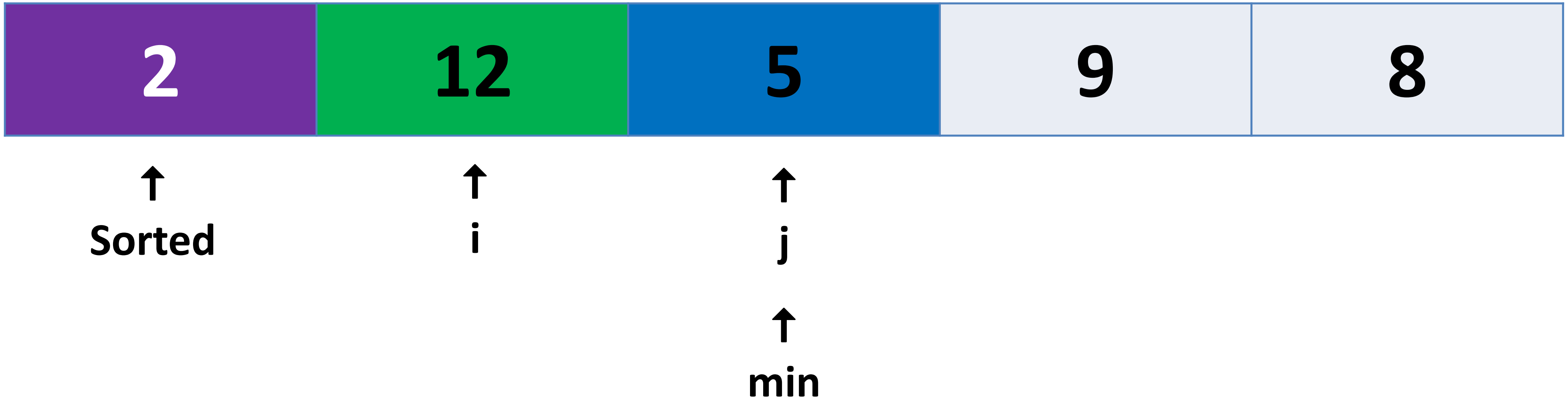
- ❑ Compare



2	12	5	9	8

Selection Sort

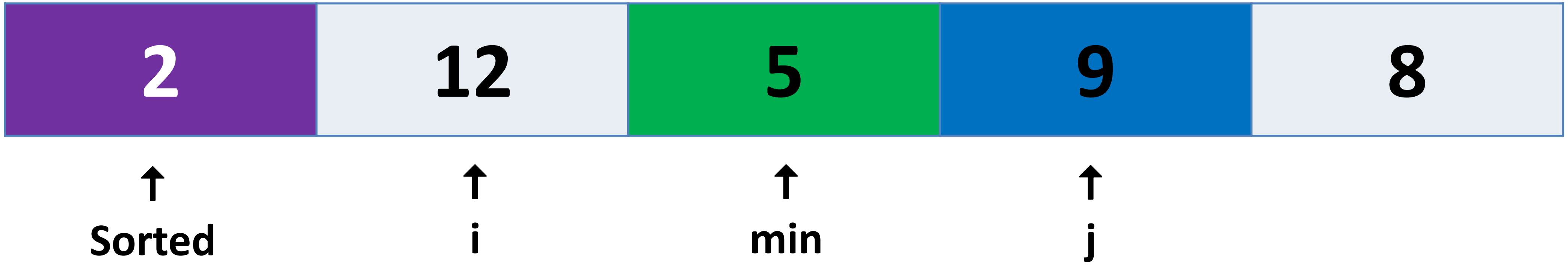
- Move



2	12	5	9	8

Selection Sort

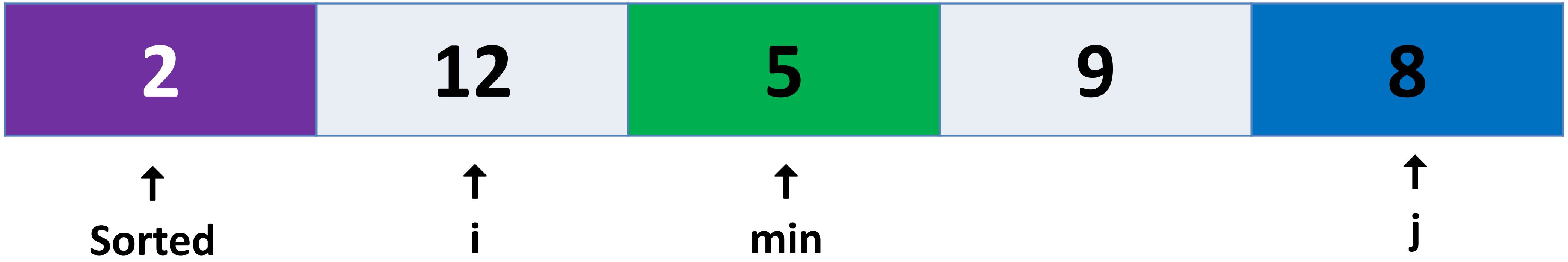
- ❑ Compare



2	12	5	9	8
↑ Sorted	↑ i	↑ min	↑ j	

Selection Sort

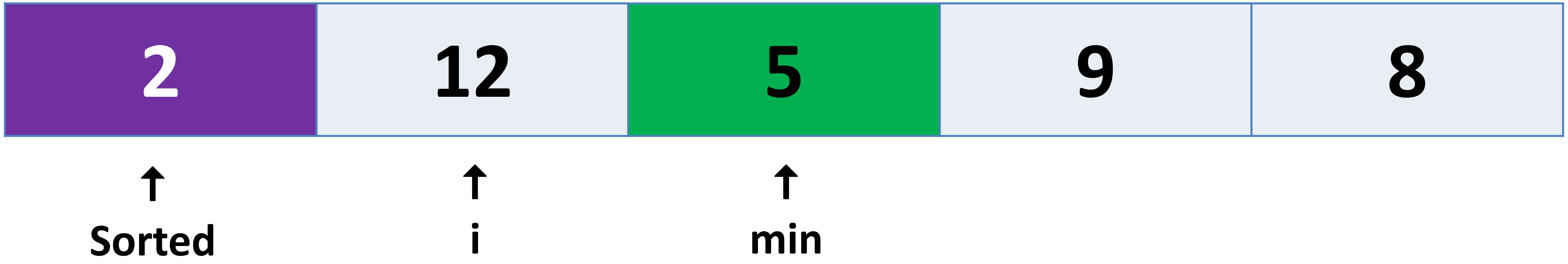
- ❑ Compare



2	12	5	9	8
↑ Sorted	↑ i	↑ min		↑ j

Selection Sort

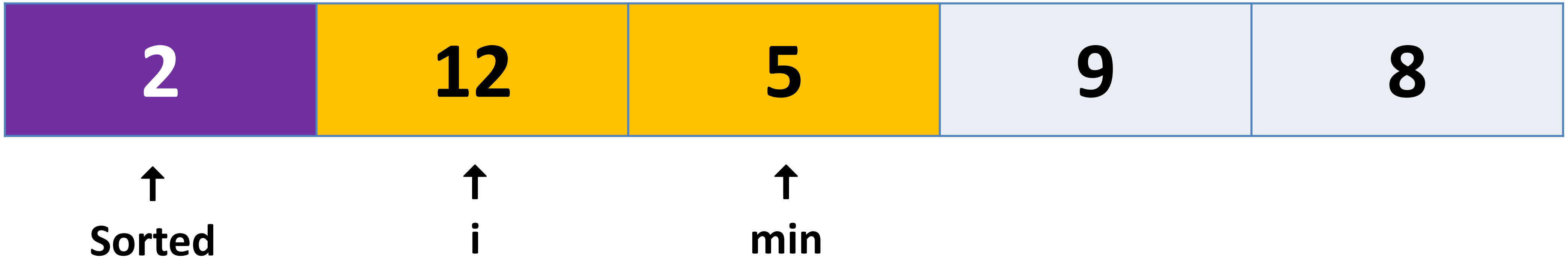
- ❑ Smallest



2	12	5	9	8
↑ Sorted	↑ i	↑ min		

Selection Sort

- ❑ Swap

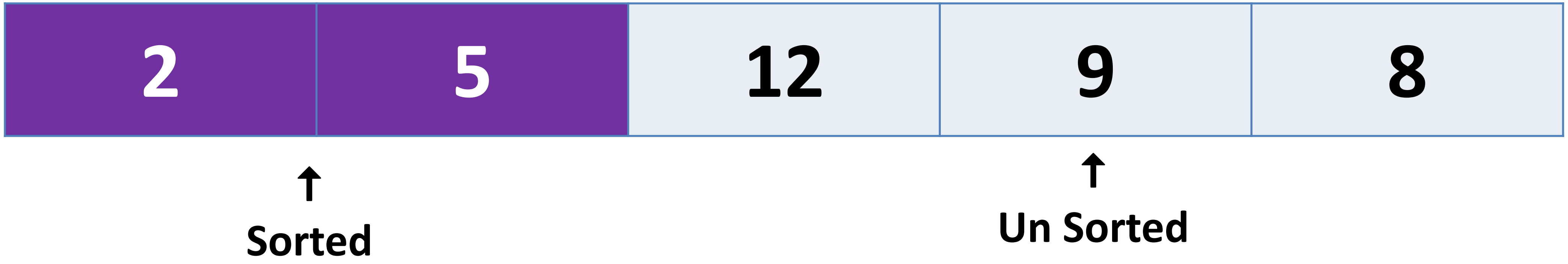


2	12	5	9	8

Selection Sort

□ Sorted

□ Un Sorted



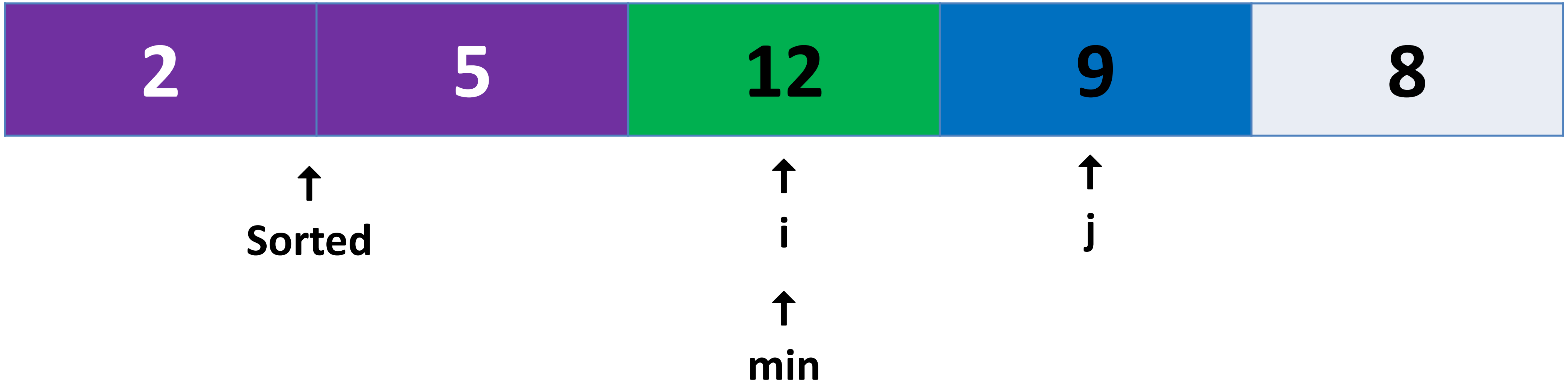
2	5	12	9	8

Sorted

Un Sorted

Selection Sort

- ❑ Compare



2	5	12	9	8

Sorted

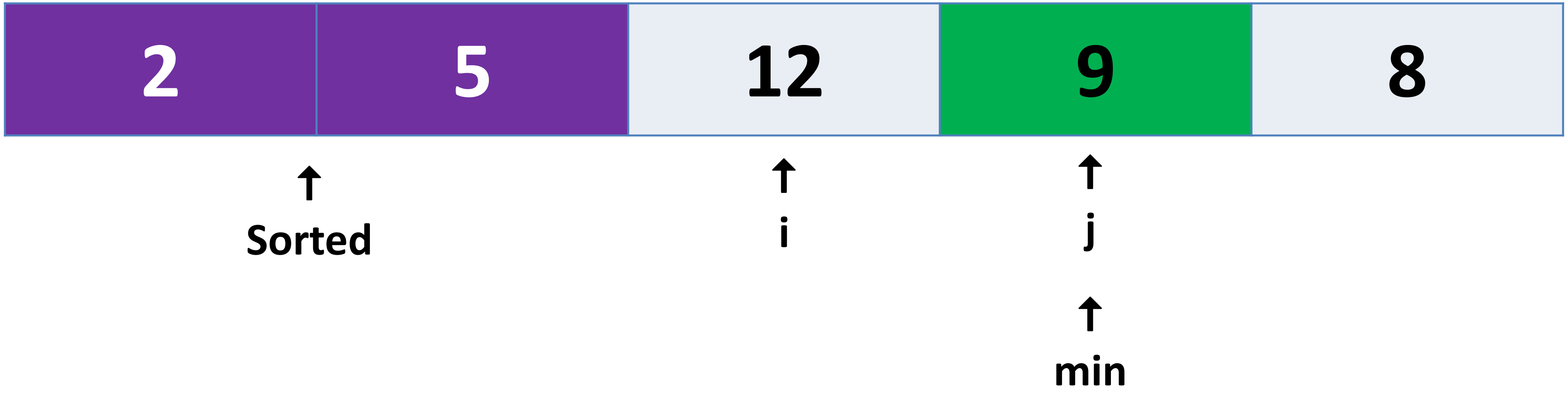
i

j

min

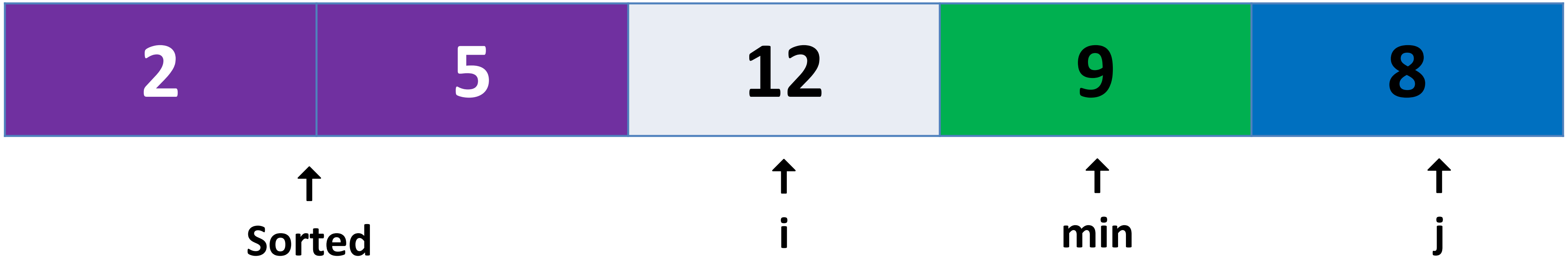
Selection Sort

- ❑ Move



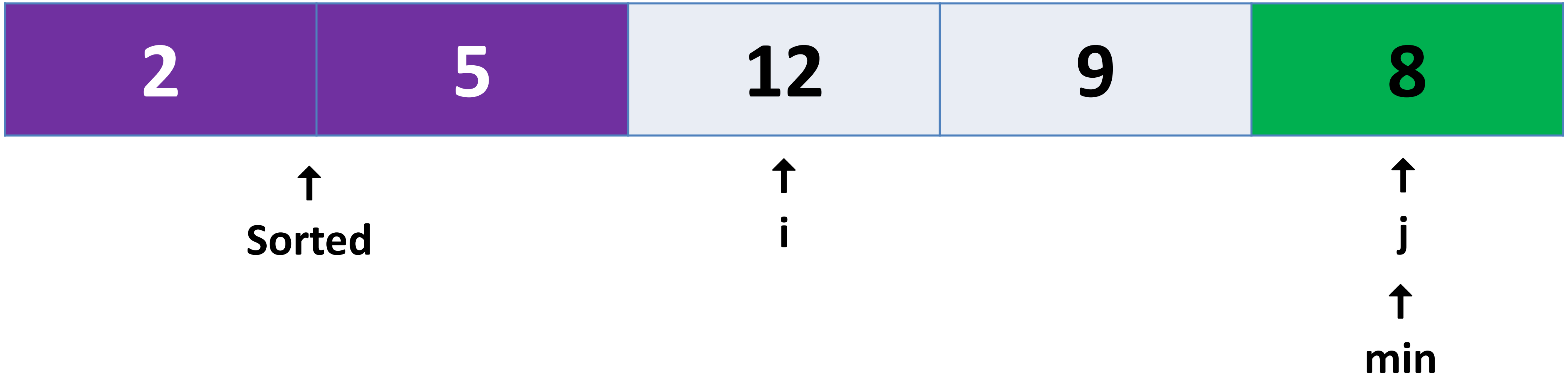
Selection Sort

- Compare



Selection Sort

- ❑ Move



2 5 12 9 8

Sorted

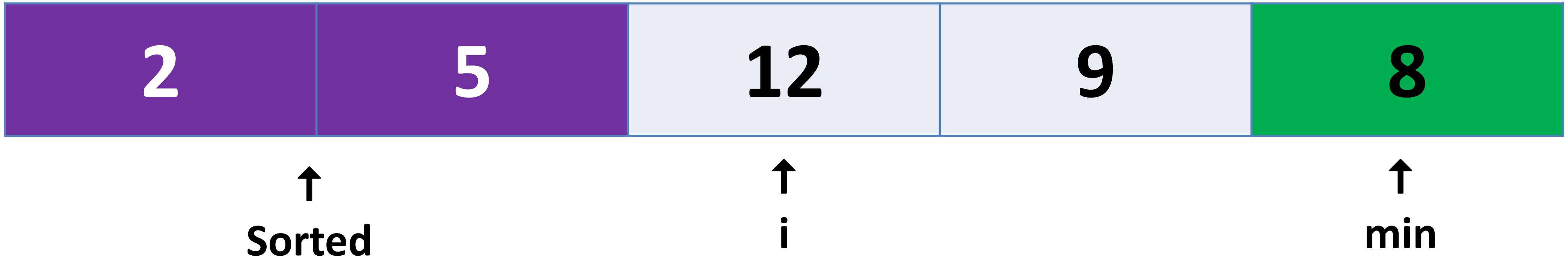
i

j

min

Selection Sort

- Smallest



2	5	12	9	8

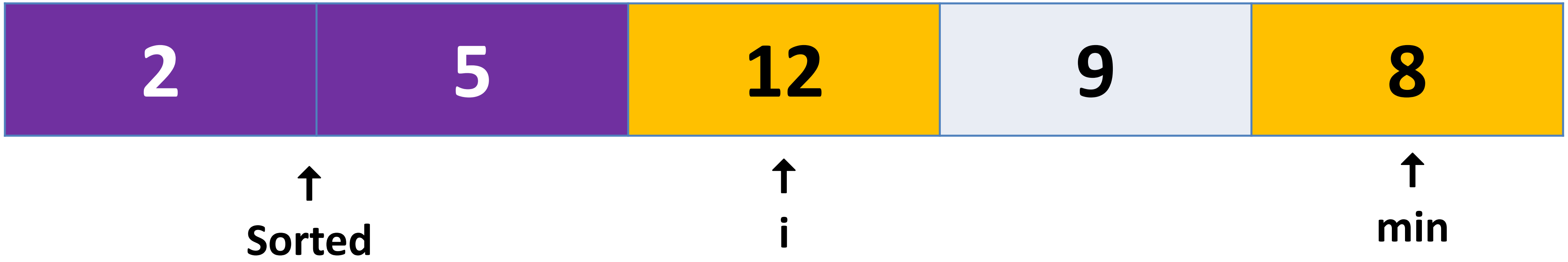
Sorted

i

min

Selection Sort

- Swap



2	5	12	9	8

Sorted

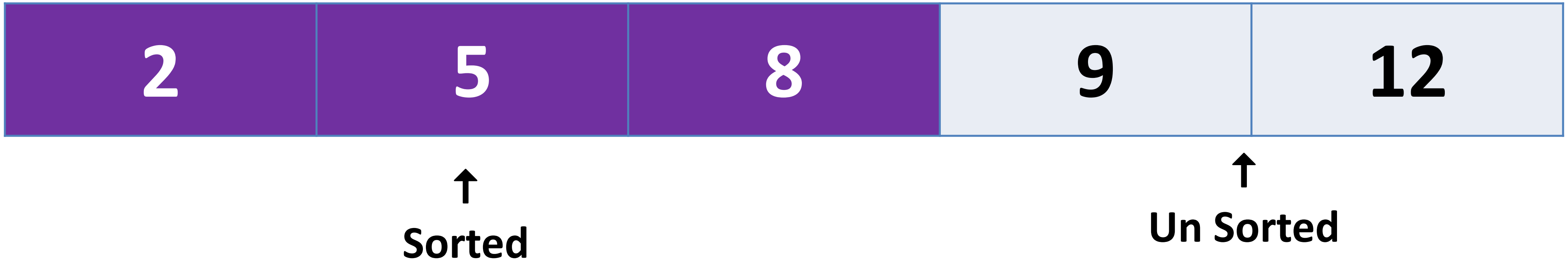
i

min

Selection Sort

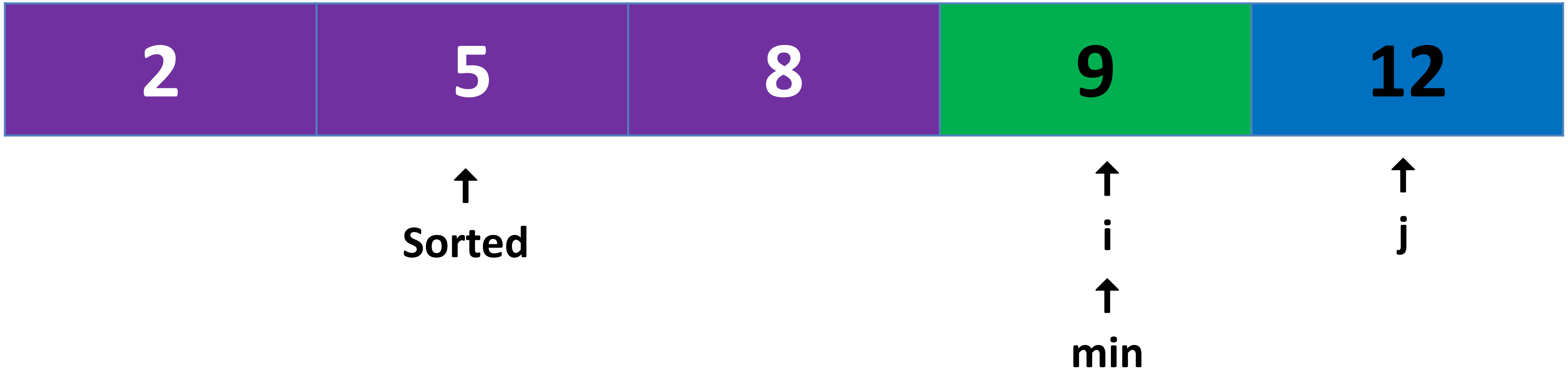
- ❑ Sorted

- ❑ Un Sorted



Selection Sort

❑ Compare

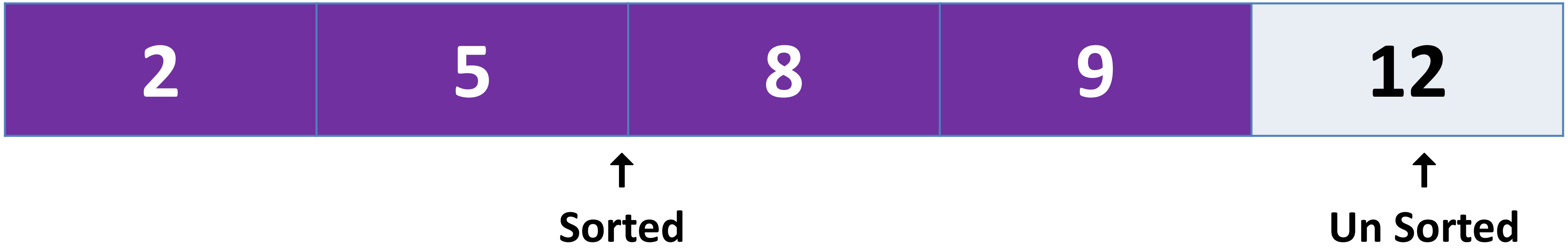


2	5	8	9	12

Selection Sort

□ Sorted

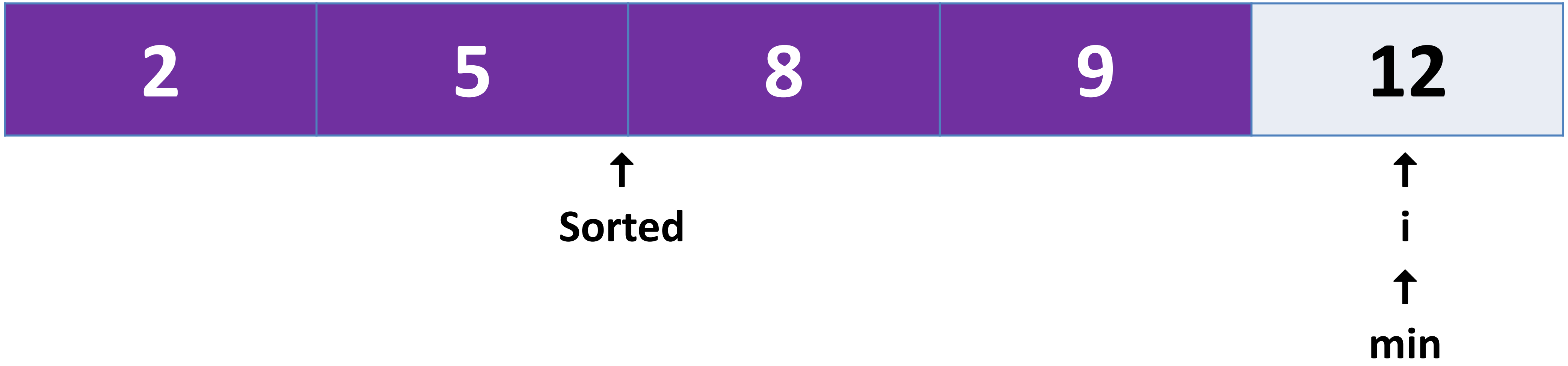
□ Un Sorted



2	5	8	9	12

Selection Sort

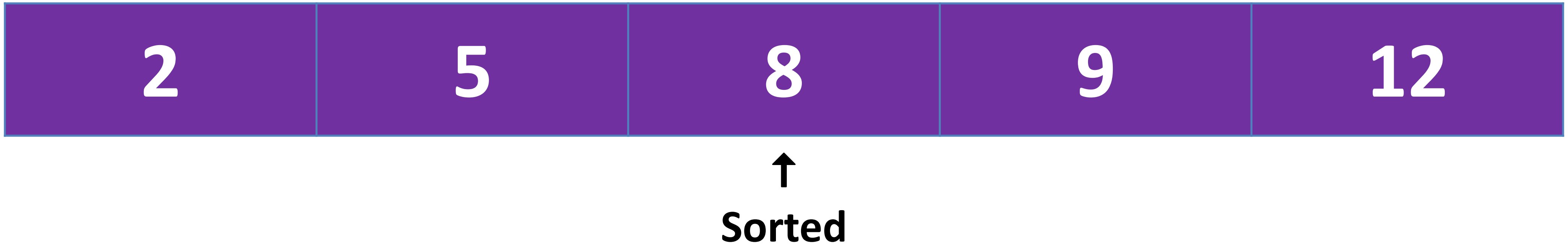
- ❑ Sorted
- ❑ Un Sorted



2	5	8	9	12

Selection Sort

- ❑ Array is now sorted

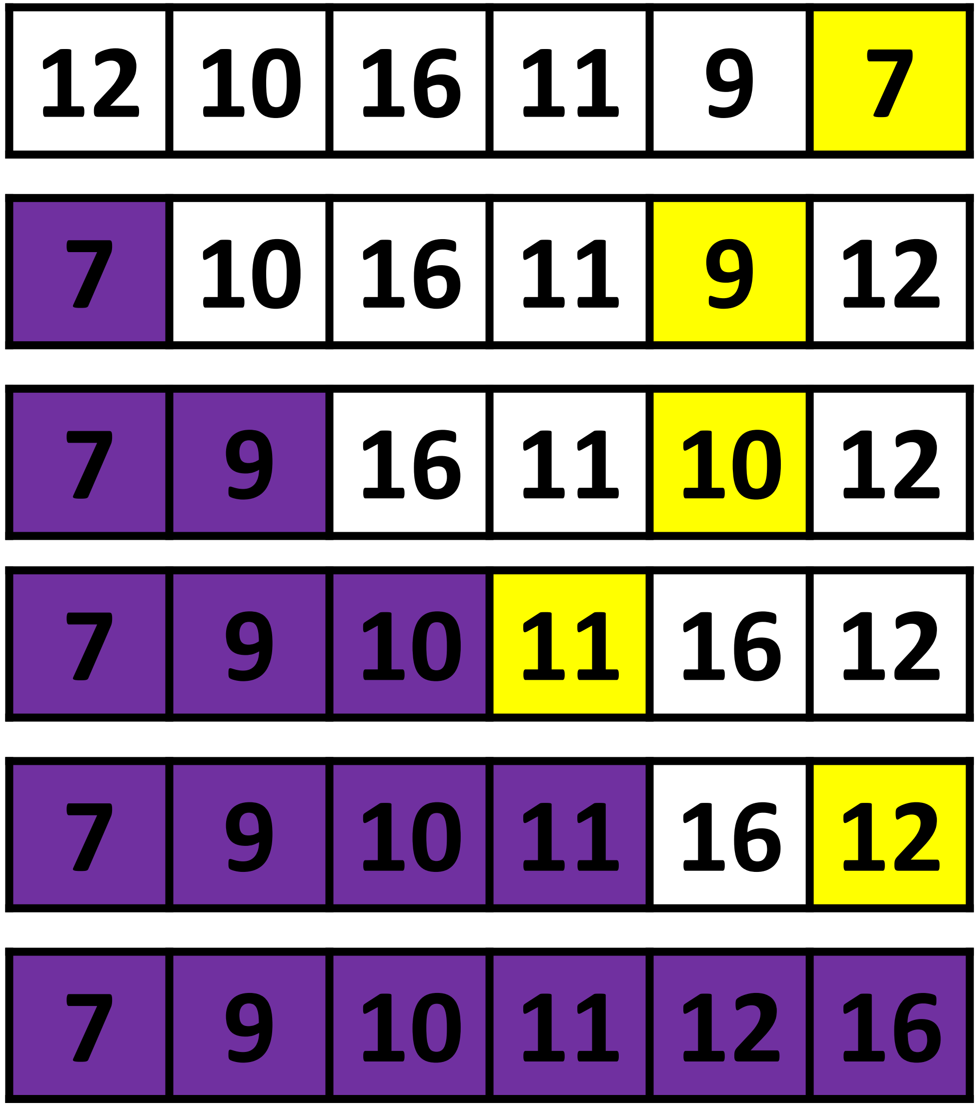


2	5	8	9	12

Selection Sort

- ❑ Example 2:

12	10	16	11	9	7



12	10	16	11	9	7
7	10	16	11	9	12
7	9	16	11	10	12
7	9	10	11	16	12
7	9	10	11	16	12
7	9	10	11	12	16

Selection Sort

- ❑ What is the output of selection sort after the 2nd iteration given the following sequence of numbers: **13 2 9 4 18 45 37 63**

a) 2 4 9 13 18 37 45 63

b) 2 9 4 13 18 37 45 63

c) 13 2 4 9 18 45 37 63

d) 2 4 9 13 18 45 37 63

Selection Sort

□ What is the output of selection sort after the 2nd iteration given the following sequence of numbers: **13 2 9 4 18 45 37 63**

a) 2 4 9 13 18 37 45 63

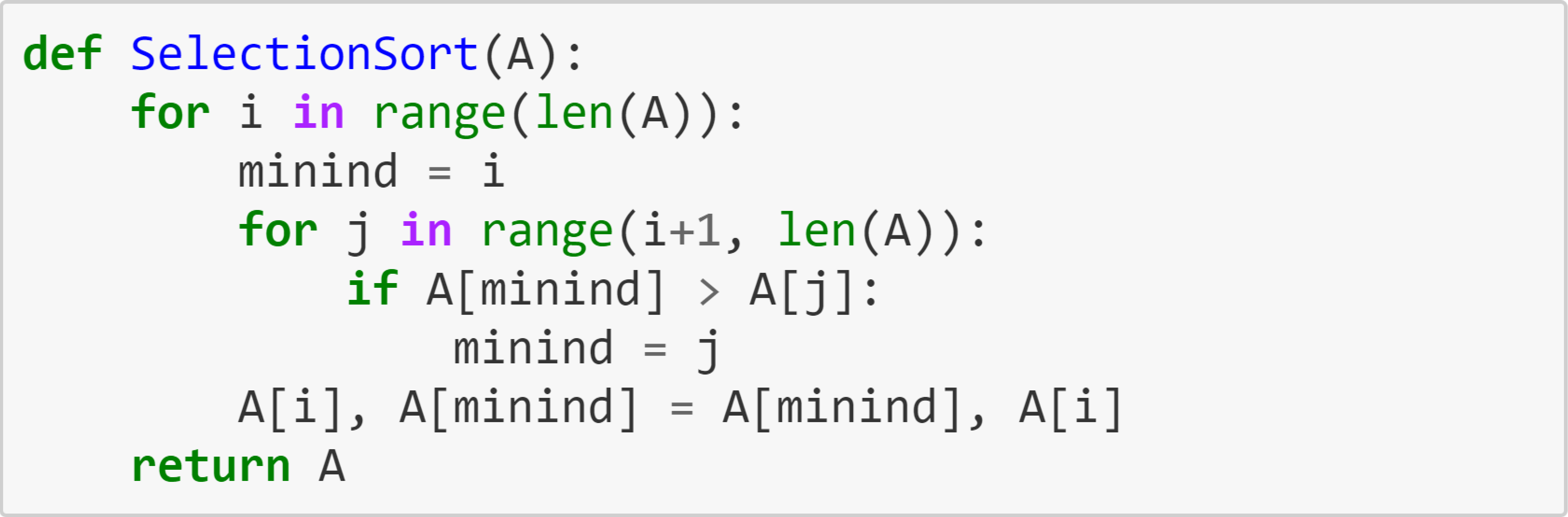
b) 2 9 4 13 18 37 45 63

c) 13 2 4 9 18 45 37 63

d) 2 4 9 13 18 45 37 63

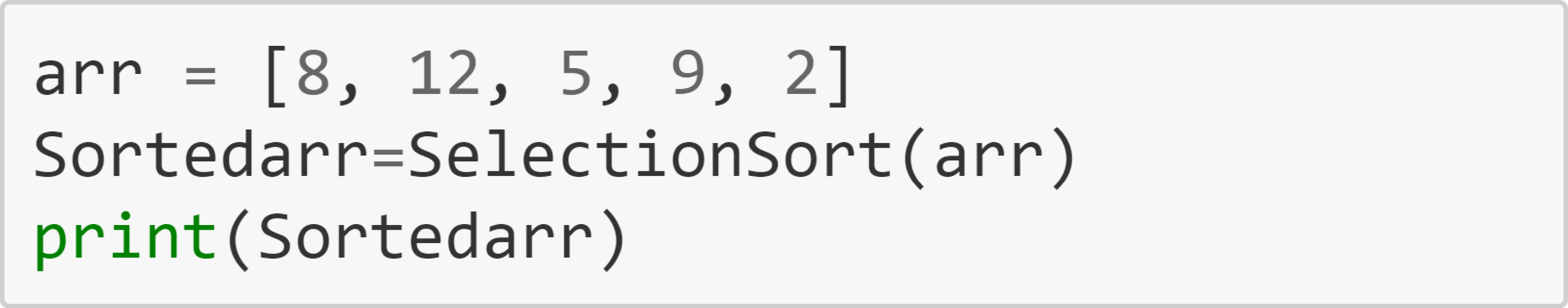
Selection Sort

❑ Python Code



```python
def SelectionSort(A):
    for i in range(len(A)):
        minind = i
        for j in range(i+1, len(A)):
            if A[minind] > A[j]:
                minind = j
        A[i], A[minind] = A[minind], A[i]
    return A
```

Selection Sort



```
arr = [8, 12, 5, 9, 2]
Sortedarr=SelectionSort(arr)
print(Sortedarr)
```

Selection Sort

- ❑ **Time Complexity:** O(n^2) as there are two nested loops

- ❑ Example of worst case

2	**3**	**4**	**5**	**1**

Insertion Sort

Insertion Sort

❑**Insertion sort** is a simple sorting algorithm that works the way we sort playing cards in our hands.

Insertion Sort

❑ Algorithm:

- **Step1**: Compare each pair of adjacent elements in the list
- **Step2**: Insert element into the sorted list, until it occupies correct position.
- **Step3**: Swap two element if necessary
- **Step4**: Repeat this process for all the elements until the entire array is sorted

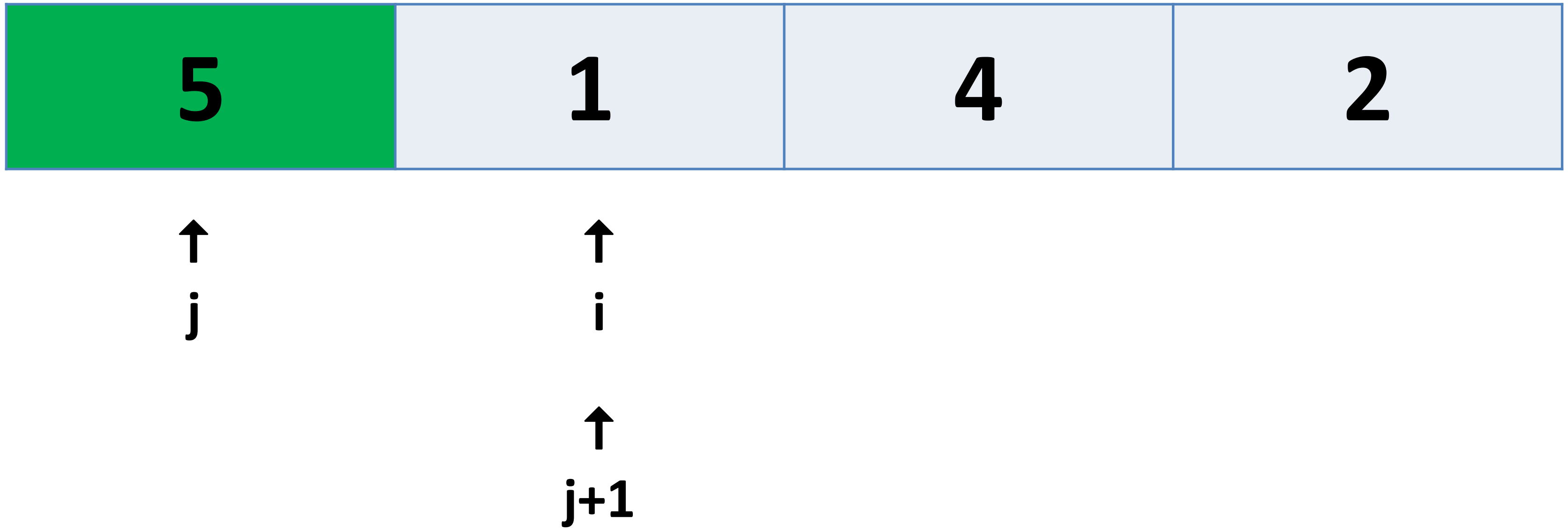
Insertion Sort

- Assume the following Array:

5	1	4	2

Insertion Sort

- ❑ Compare
- ❑ Store= 1



5	1	4	2

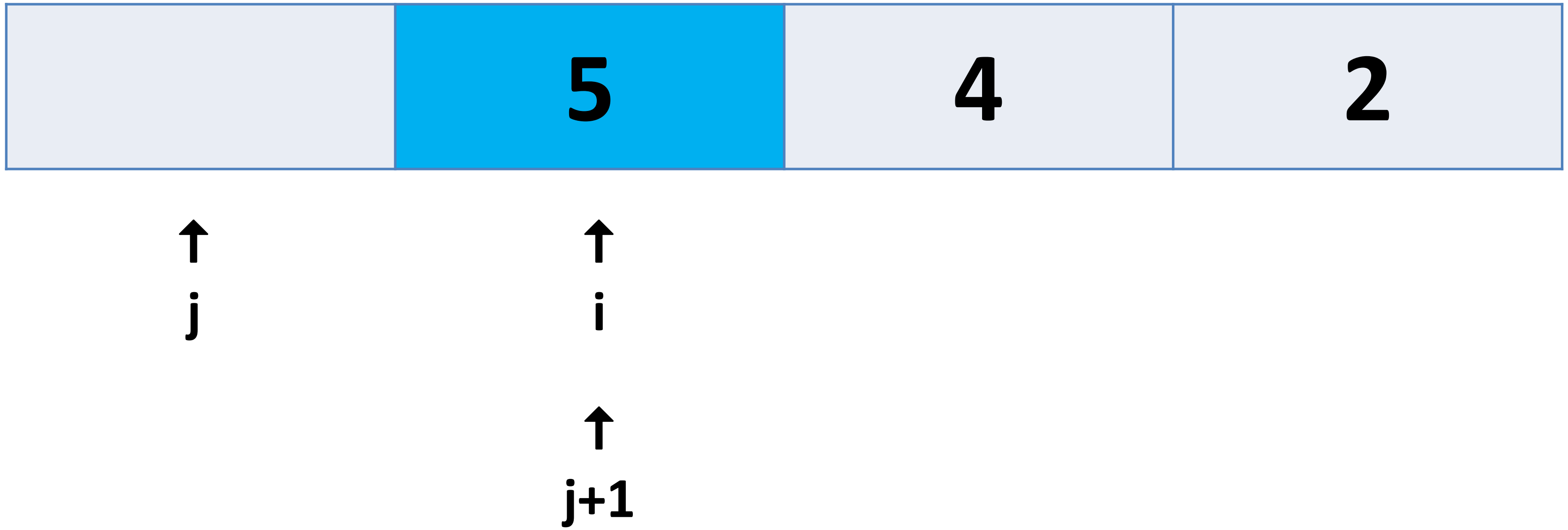
↑ j

↑ i

↑ j+1

Insertion Sort

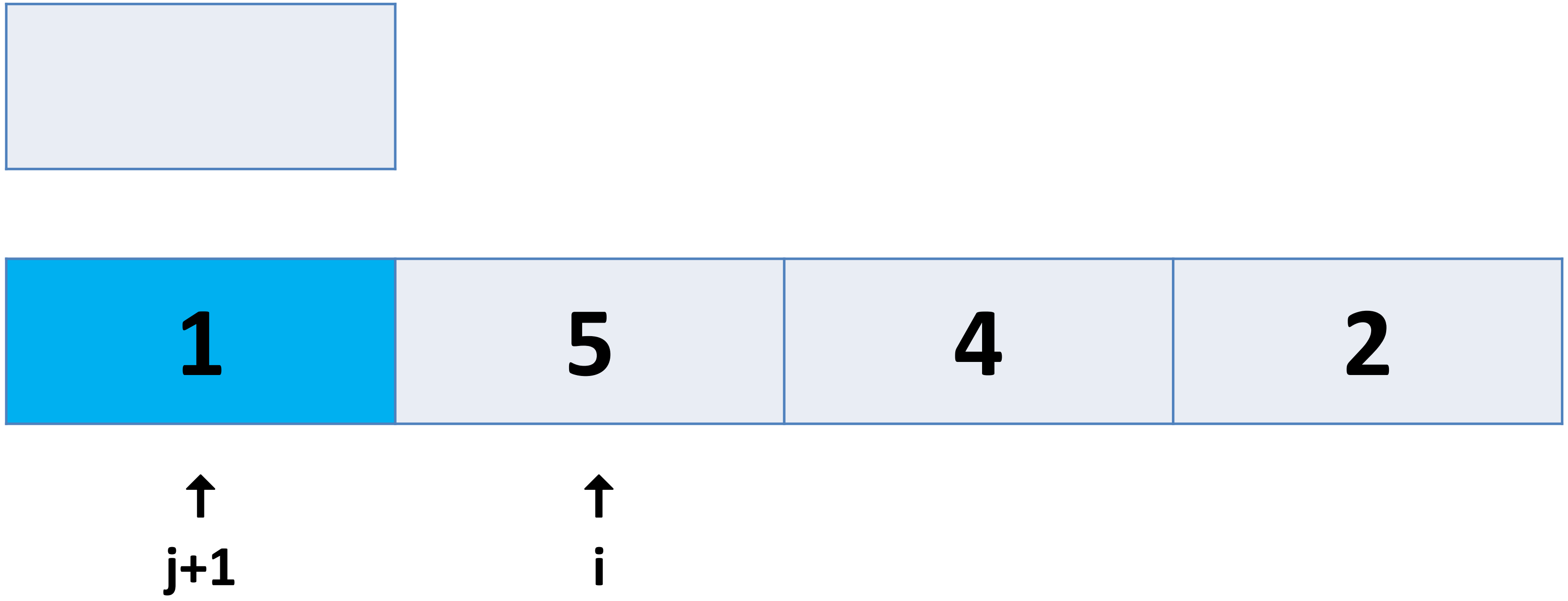
- □ Move
- □ Store= 1



Insertion Sort

□ Move

□ Store=



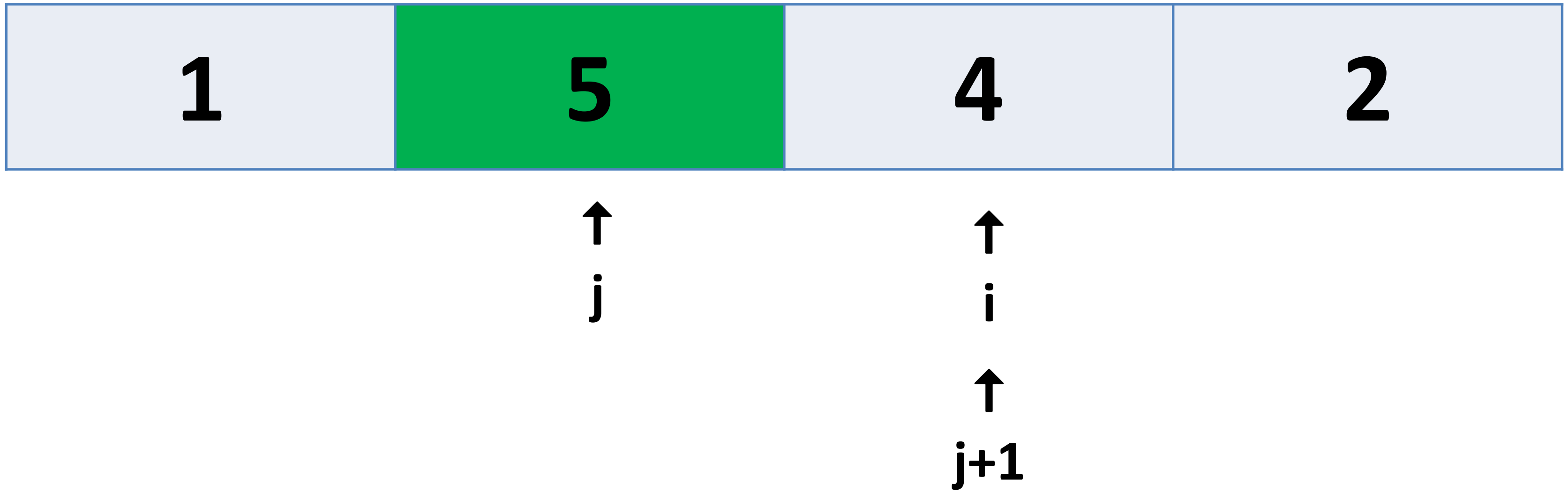
1	5	4	2

↑ j+1

↑ i

Insertion Sort

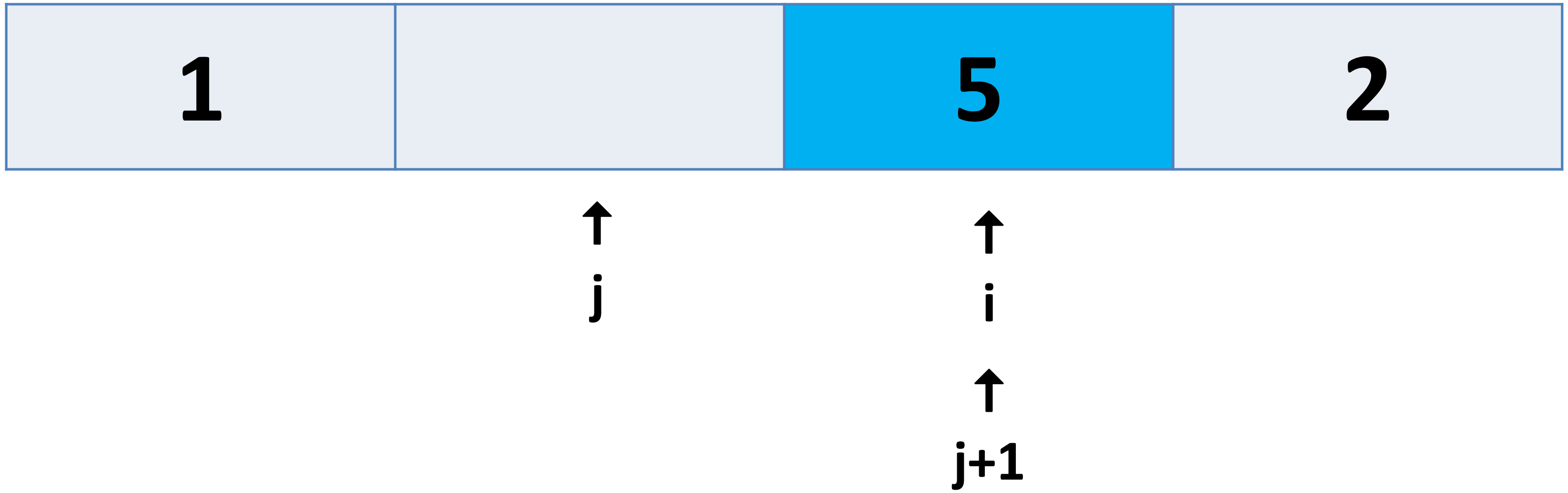
- ❑ Compare
- ❑ Store= 4



1	5	4	2

Insertion Sort

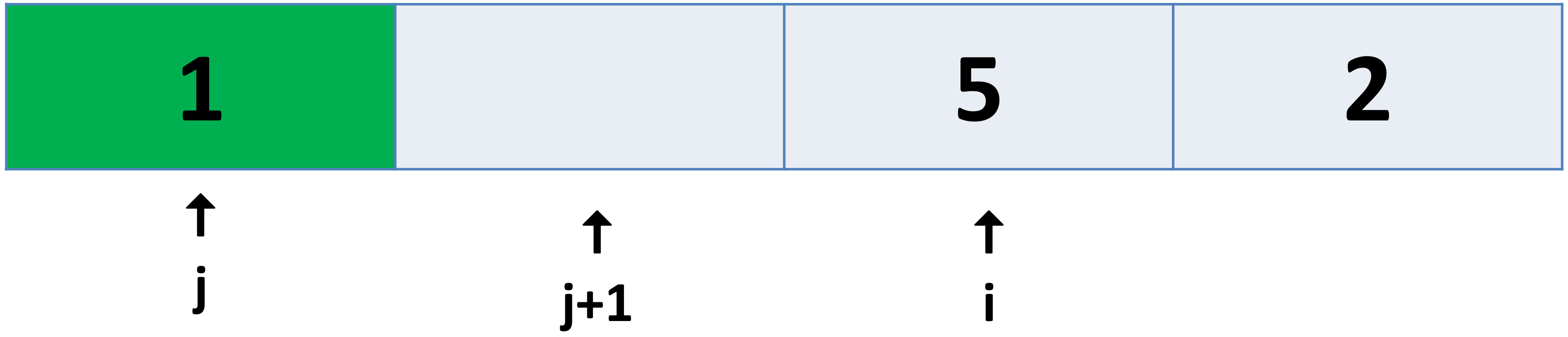
- ❑ Move
- ❑ Store= 4



1		5	2

Insertion Sort

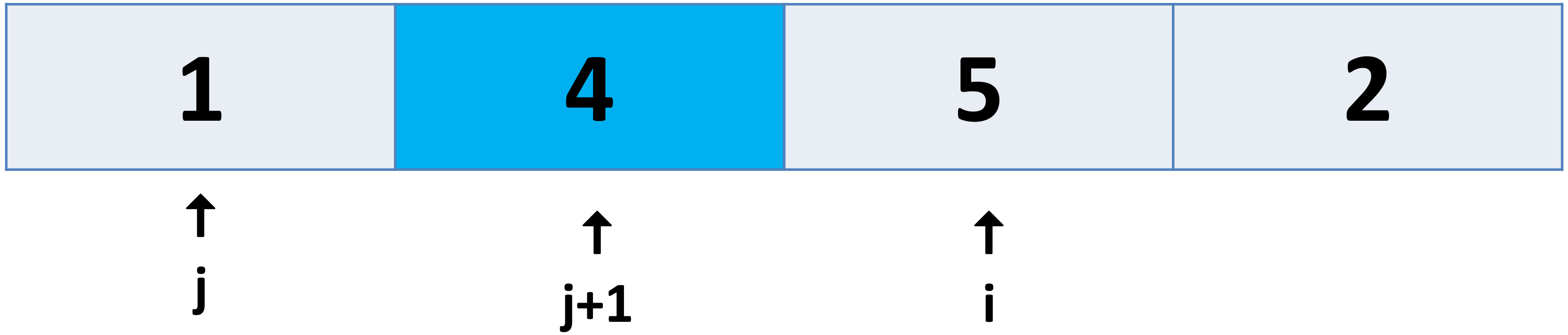
- □ Compare
- □ Store= **4**



1		5	2
j	j+1	i	

Insertion Sort

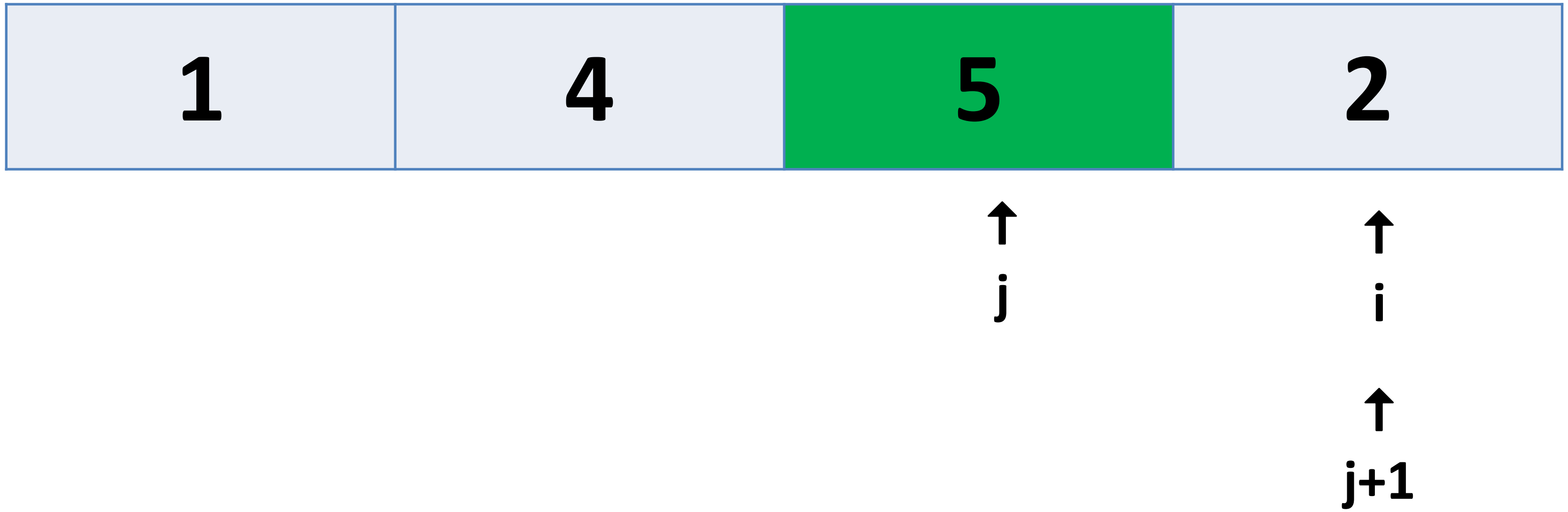
- □ Move
- □ Store=



1	4	5	2
↑ j	↑ j+1	↑ i	

Insertion Sort

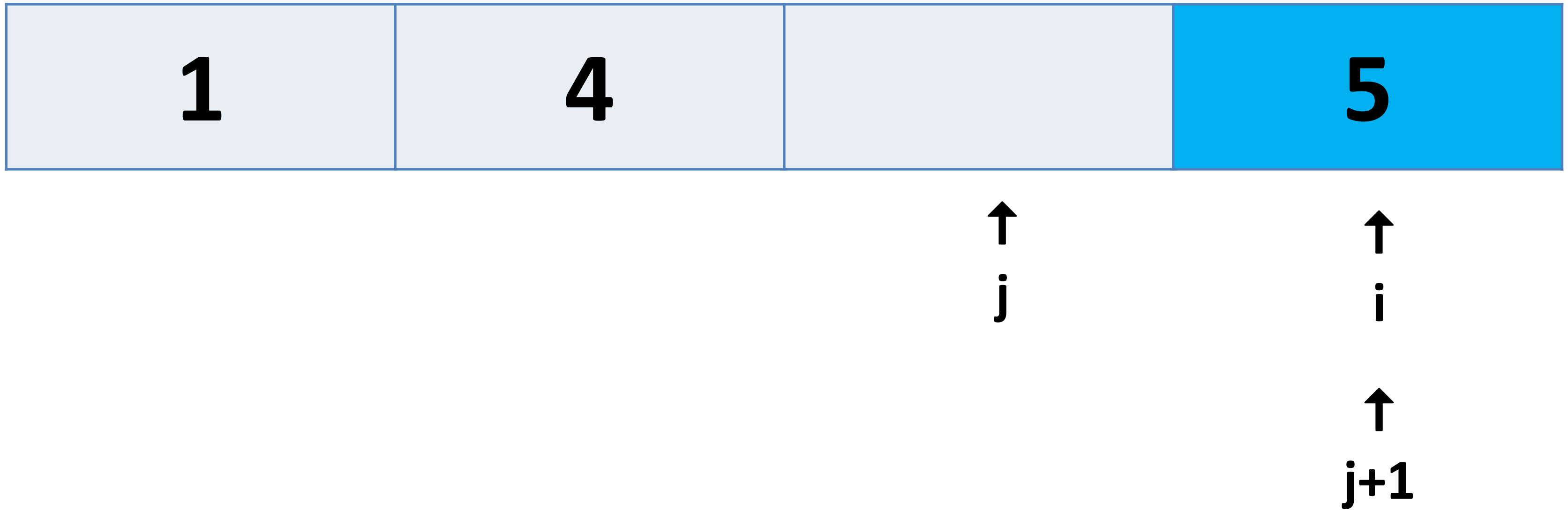
- ❑ Compare
- ❑ Store= 2



1	4	5	2

Insertion Sort

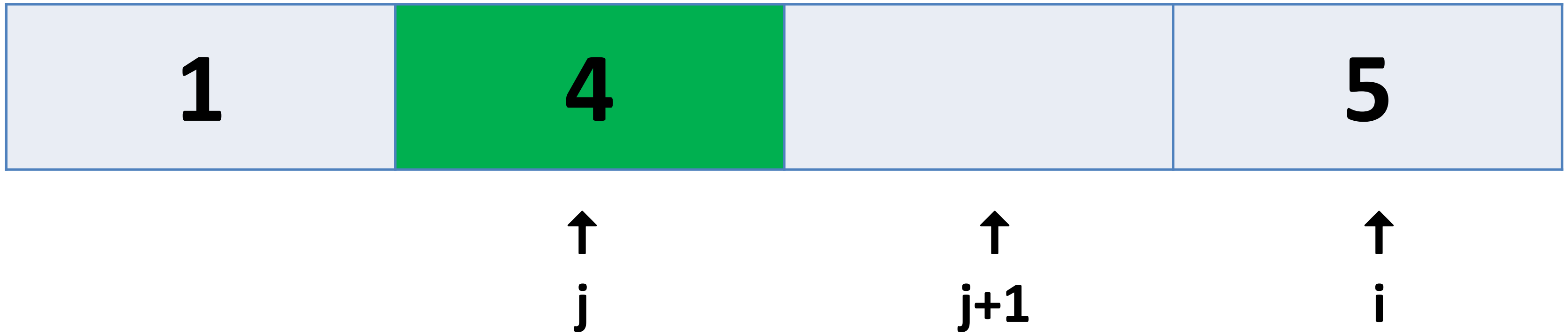
- ❑ Move
- ❑ Store= 2



Insertion Sort

❑ Compare

❑ Store= 2

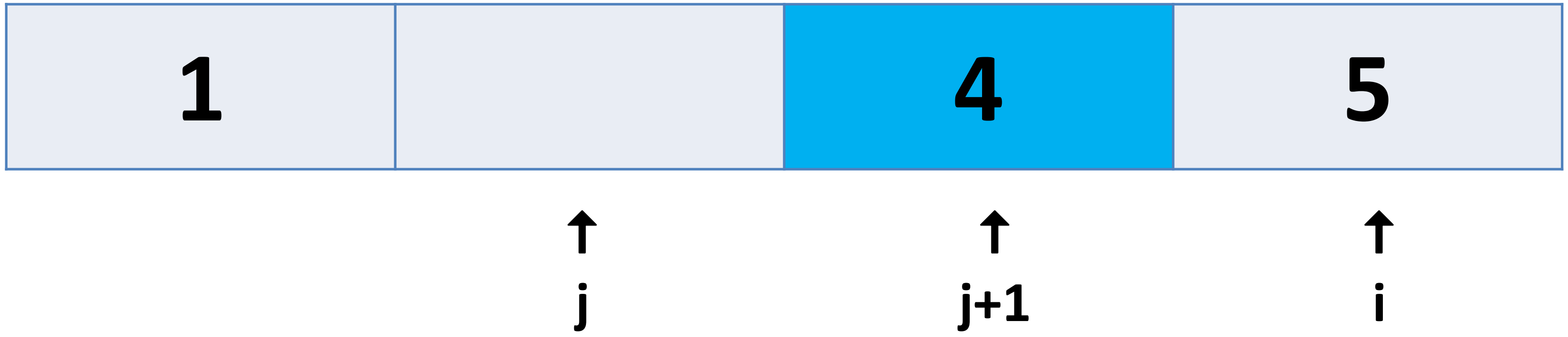


1	4		5
	↑ j	↑ j+1	↑ i

Insertion Sort

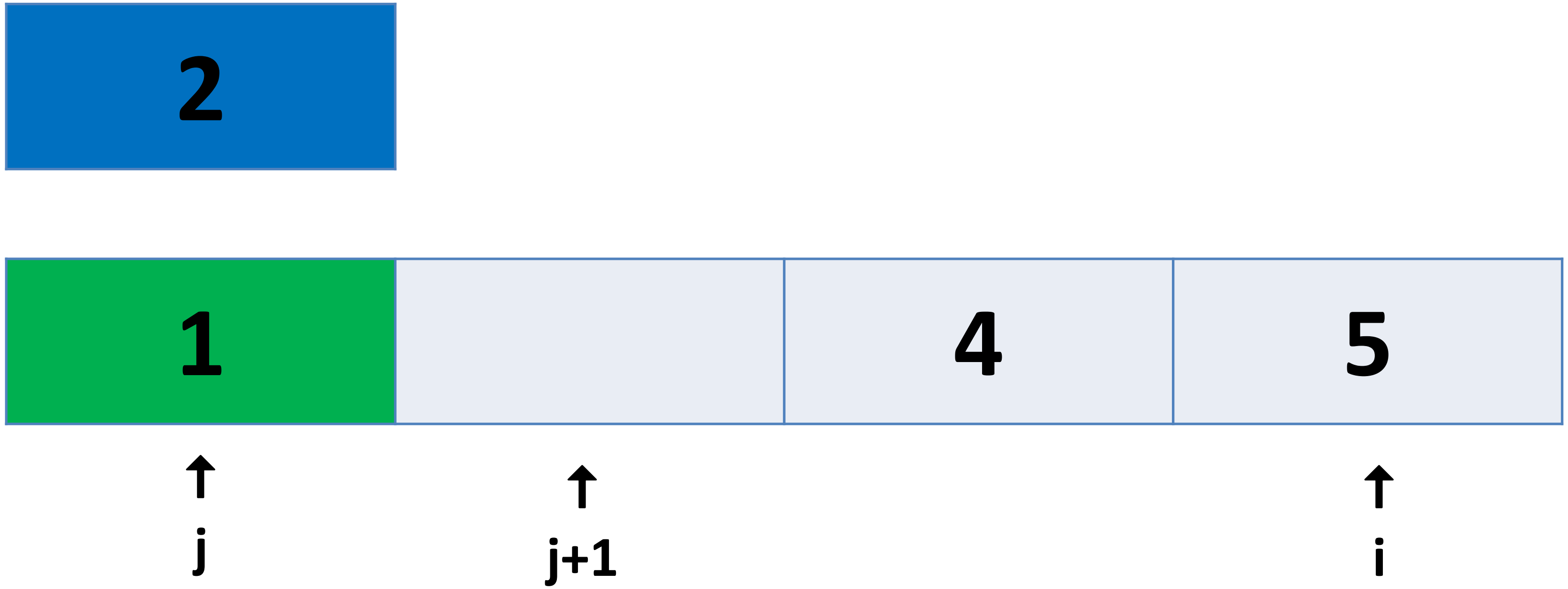
□ Move

□ Store= 2



Insertion Sort

- □ Compare
- □ Store=



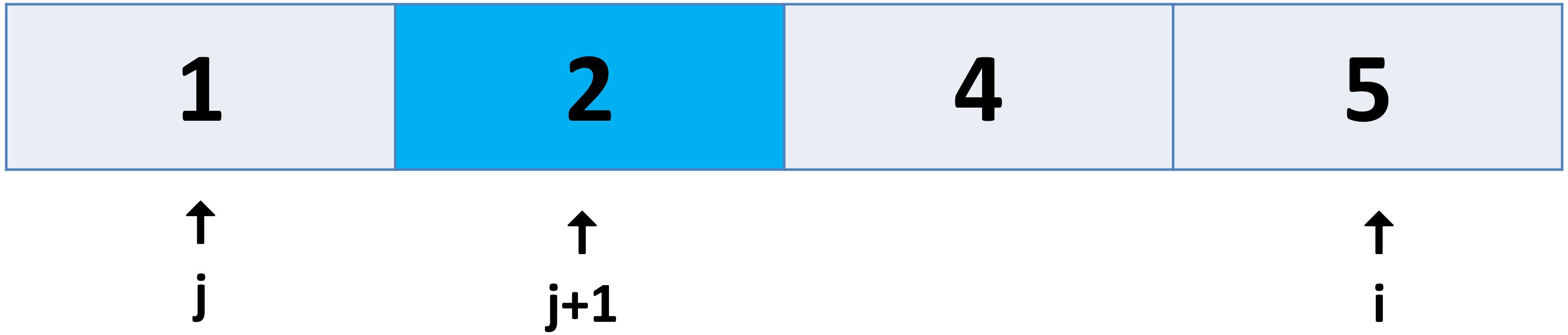
2

1		4	5

Insertion Sort

□ Compare

□ Store=



1	2	4	5

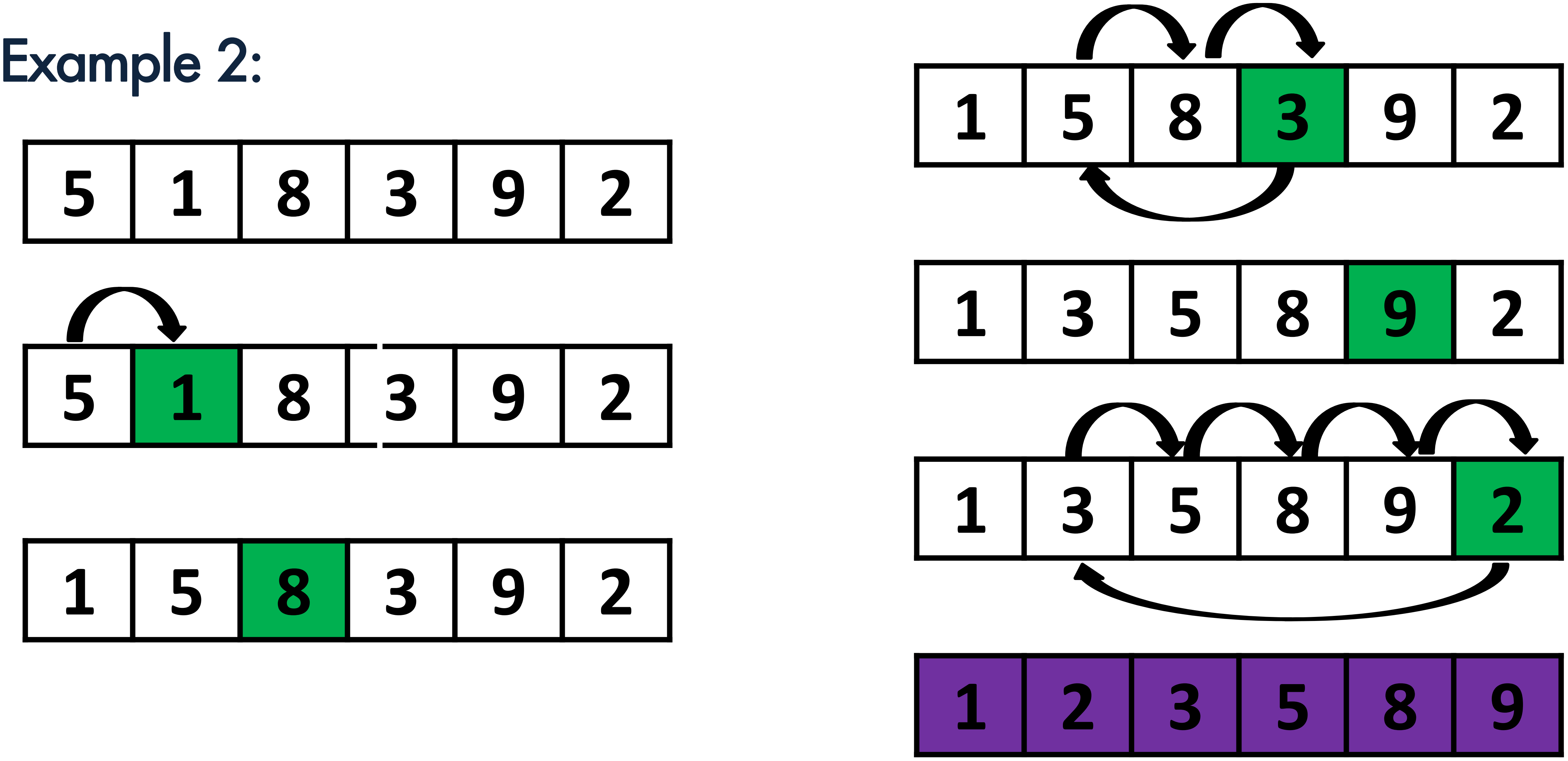
Insertion Sort

- Array is now sorted

1	2	4	5

Selection Sort

- Example 2:



5 1 8 3 9 2

5 1 8 3 9 2

1 5 8 3 9 2

1 5 8 3 9 2

1 3 5 8 9 2

1 3 5 8 9 2

1 2 3 5 8 9

Insertion Sort

- ❑ What is the output of insertion sort after the 1st iteration given the following sequence of numbers: 7 3 5 1 9 8 4 6

  a) 3 7 5 1 9 8 4 6

  b) 1 3 7 5 9 8 4 6

  c) 3 4 1 5 6 8 7 9

  d) 1 3 4 5 6 7 8 9

Insertion Sort

- ❑ What is the output of insertion sort after the 1st iteration given the following sequence of numbers: **7 3 5 1 9 8 4 6**

    a) 3 7 5 1 9 8 4 6

    b) 1 3 7 5 9 8 4 6

    c) 3 4 1 5 6 8 7 9

    d) 1 3 4 5 6 7 8 9

Insertion Sort

❑ What is the output of insertion sort after the 2nd iteration given the following sequence of numbers: **7 3 5 1 9 8 4 6**

a) 3 5 7 1 9 8 4 6

b) 1 3 7 5 9 8 4 6

c) 3 4 1 5 6 8 7 9

d) 1 3 4 5 6 7 8 9

Insertion Sort

❑ What is the output of insertion sort after the 2nd iteration given the following sequence of numbers: 7 3 5 1 9 8 4 6

a) 3 5 7 1 9 8 4 6

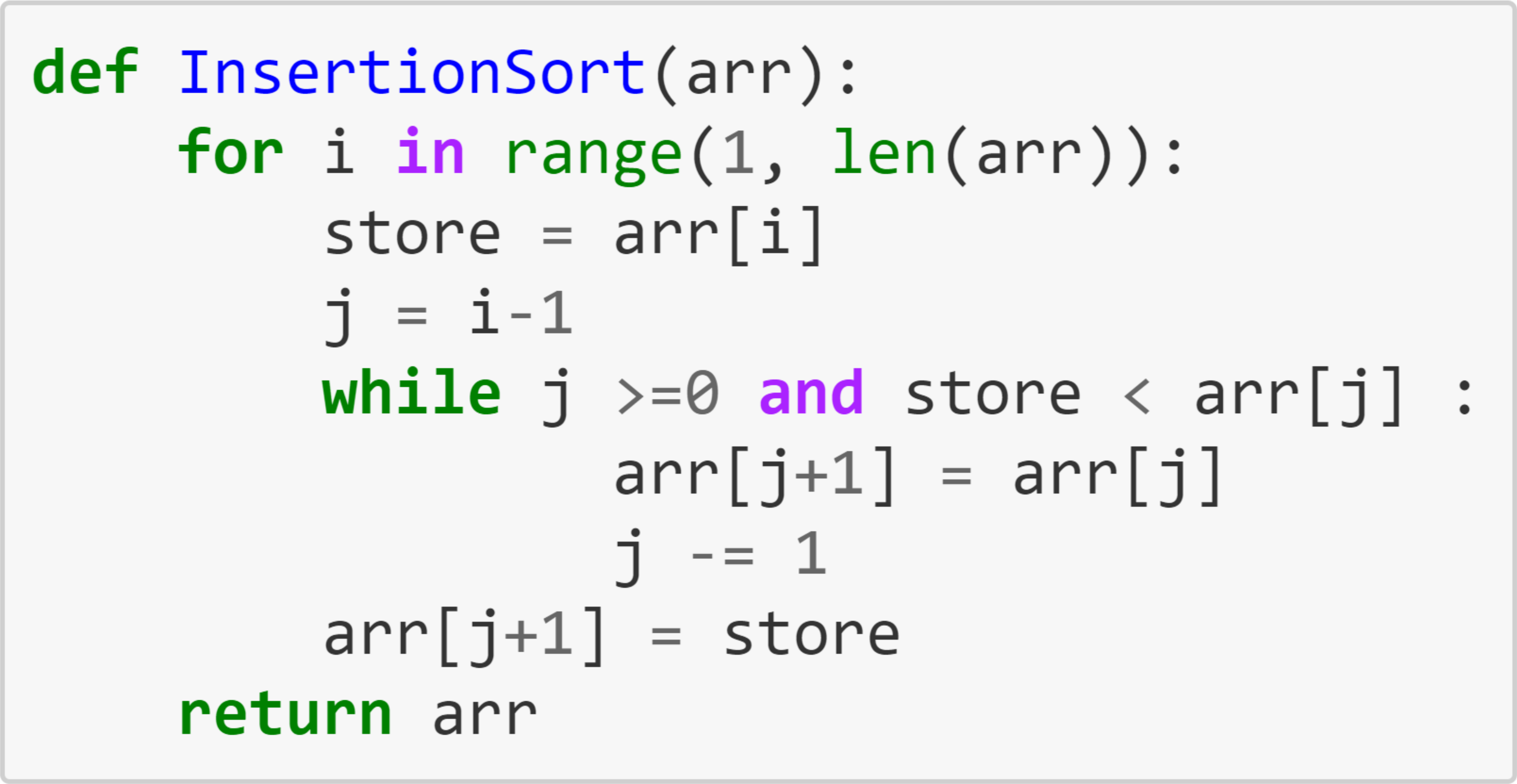
b) 1 3 7 5 9 8 4 6

c) 3 4 1 5 6 8 7 9

d) 1 3 4 5 6 7 8 9

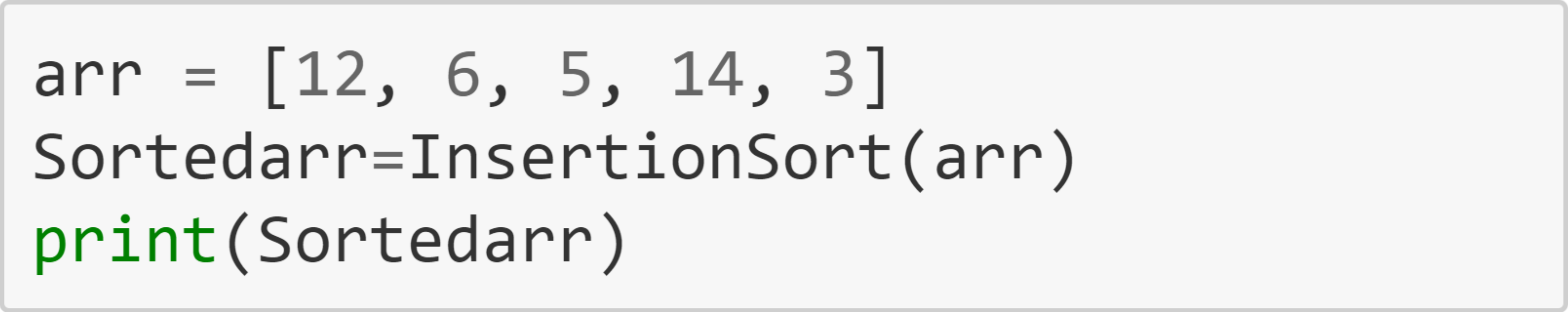
Insertion Sort

- ❑ Python Code



```
def InsertionSort(arr):
    for i in range(1, len(arr)):
        store = arr[i]
        j = i-1
        while j >=0 and store < arr[j] :
                arr[j+1] = arr[j]
                j -= 1
        arr[j+1] = store
    return arr
```

Insertion Sort



```
arr = [12, 6, 5, 14, 3]
Sortedarr=InsertionSort(arr)
print(Sortedarr)
```

Insertion Sort

- Time Complexity: O(n^2)

- Example of worst case

5	**4**	**3**	**2**	**1**