Data Structures and Algorithms

Insertion sort, Selection sort and Bubble sort

Content

- Insertion sort
- Selection sort
- Bubble sort

Insertion Sort

- Insertion sort is a simple sorting algorithm that places an unsorted element at its suitable place in each iteration.
- It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort.
- Simple implementation
- Efficient for (quite) small data sets, much like other sorting algorithms such as selection sort or bubble sort

Insertion sort

Given Array

6	3	10	1	4	9	7

Step 1: The first element in the array is assumed to be sorted. Take the second element and store it separately in temporally variable.

6	3	10	1	4	9	7

3

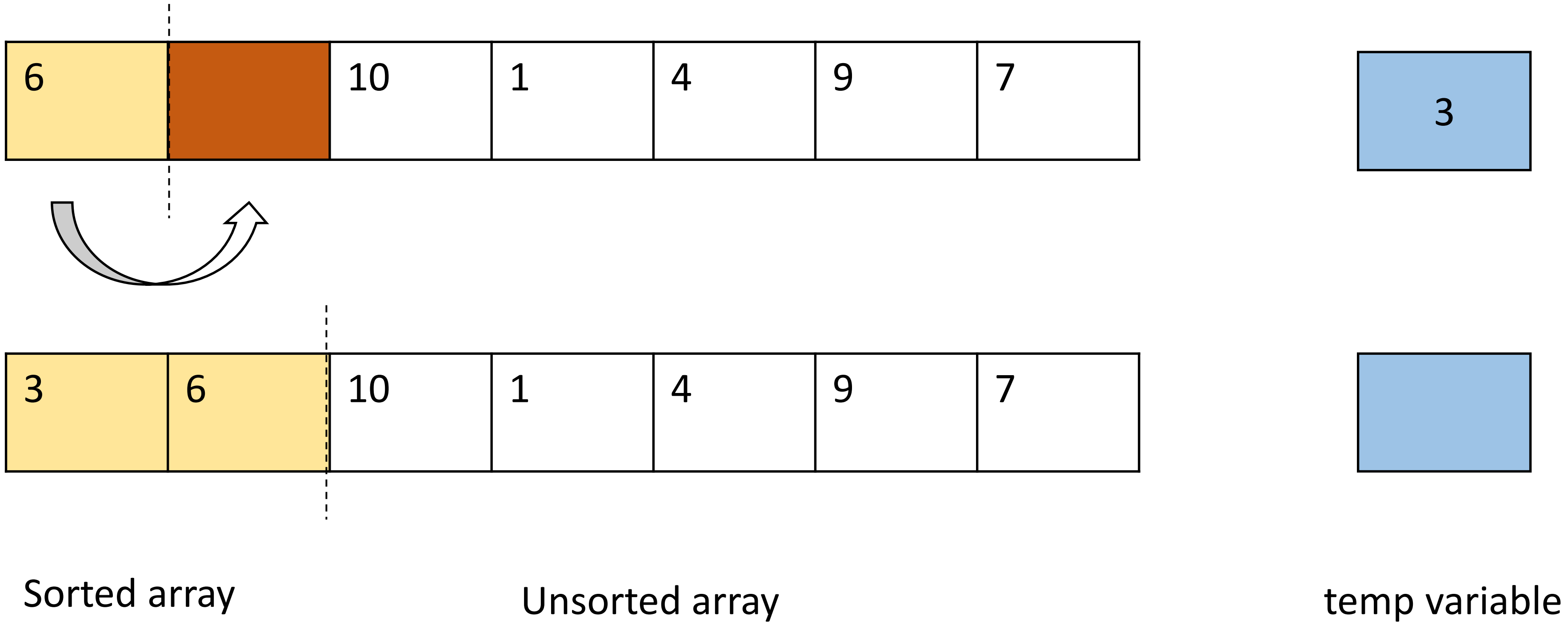
Sorted array

Unsorted array

temp variable

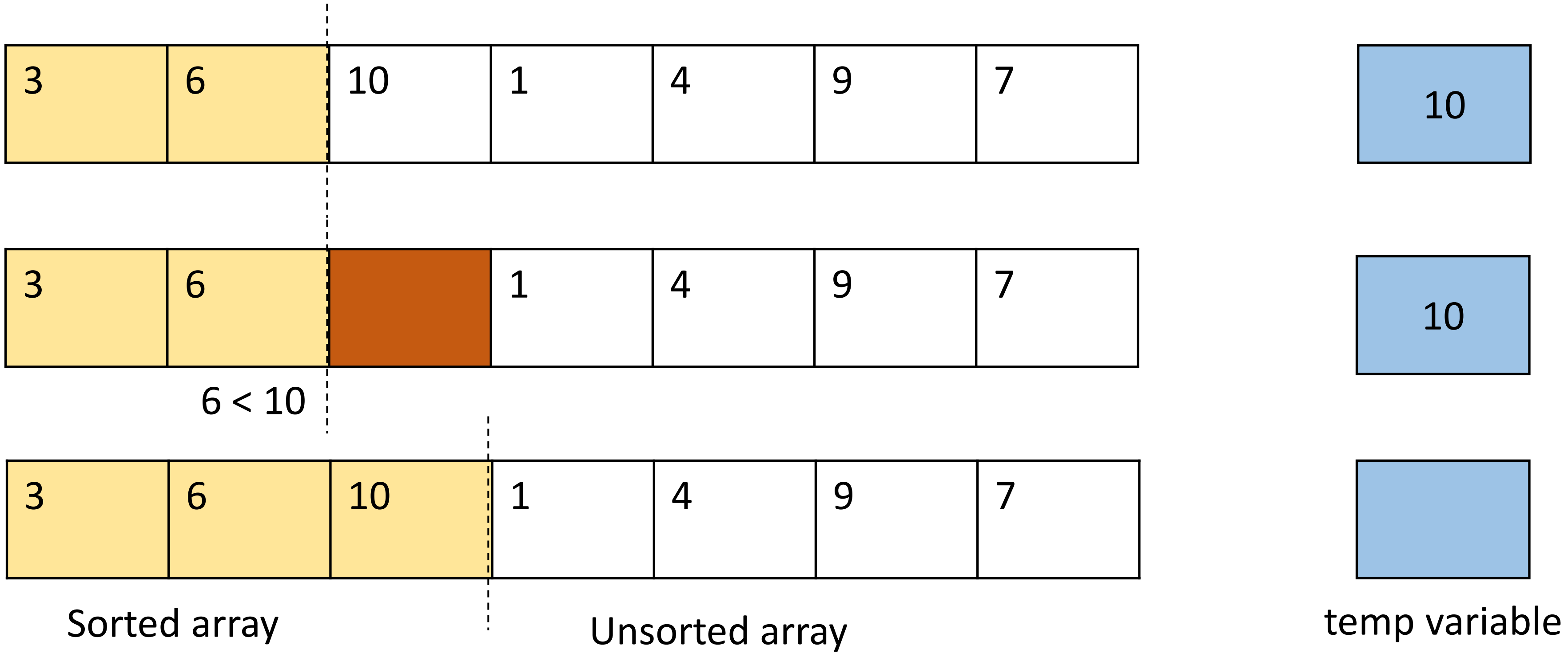
Insertion sort

Step 2: Compare temp with the first element. If the first element is greater than temp, then temp is placed in front of the first element.



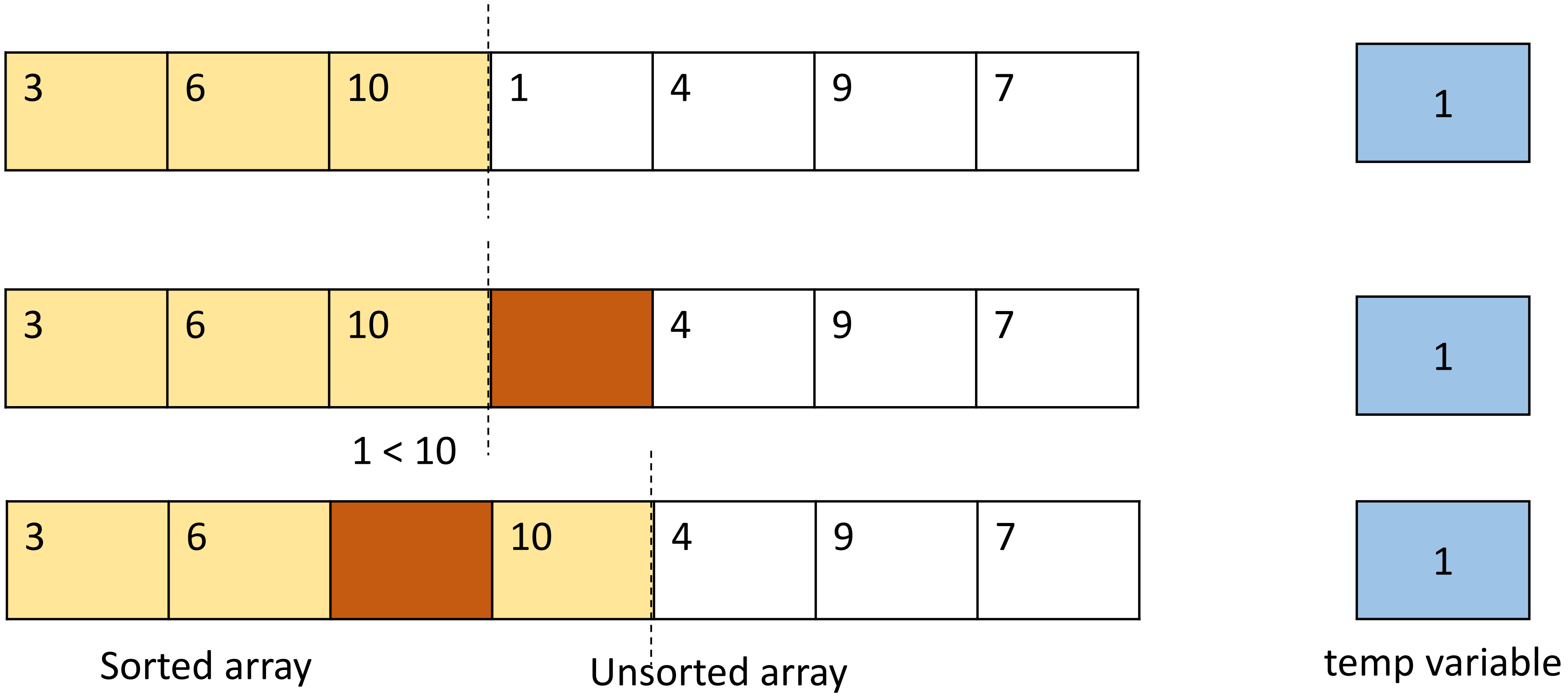
Insertion sort

Step 3: Take the third element and compare it with the elements on the left of it. Placed it just behind the element smaller than it. If there is no element smaller than it, then place it at the beginning of the array.

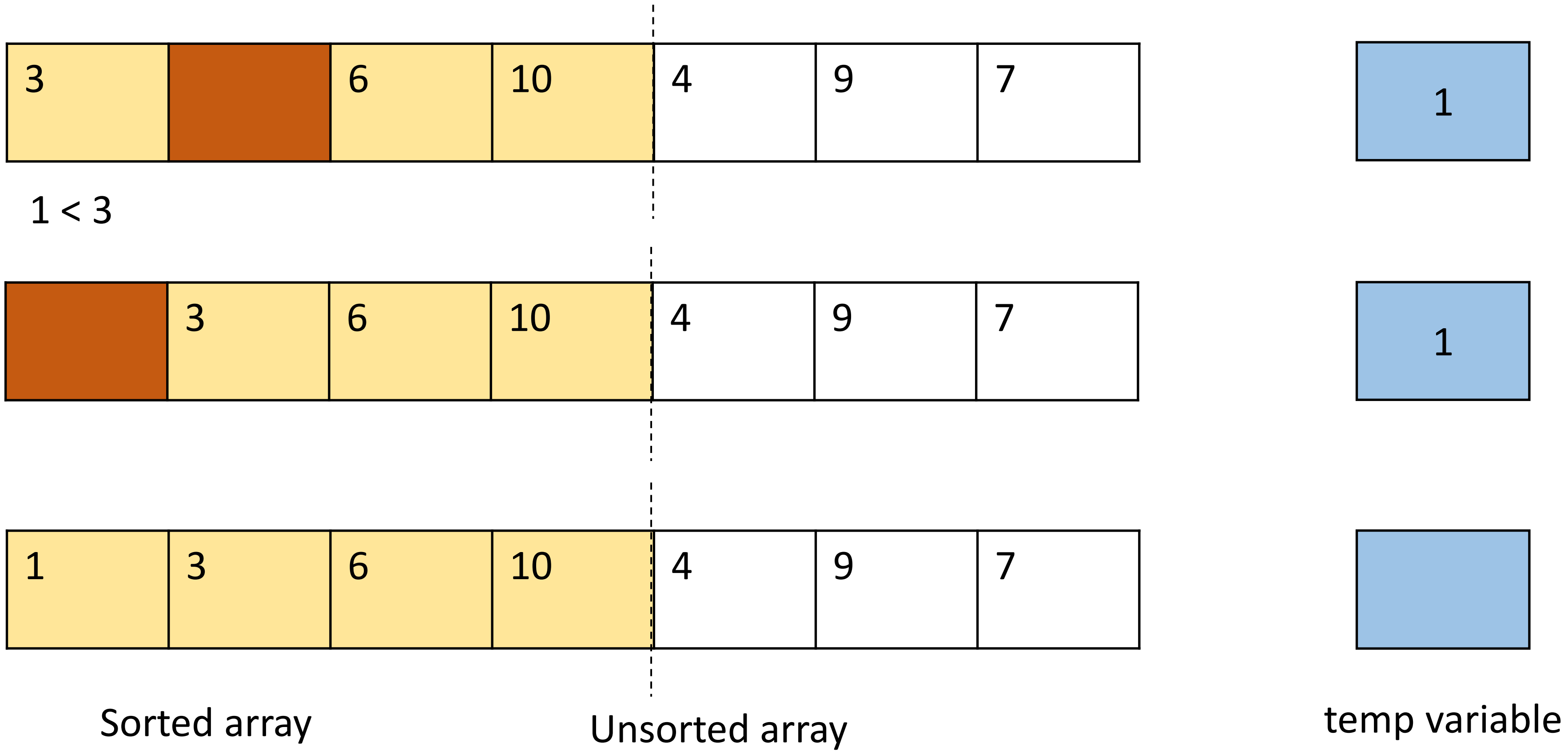


Insertion sort

Step 4: Similarly, place every unsorted element at its correct position.

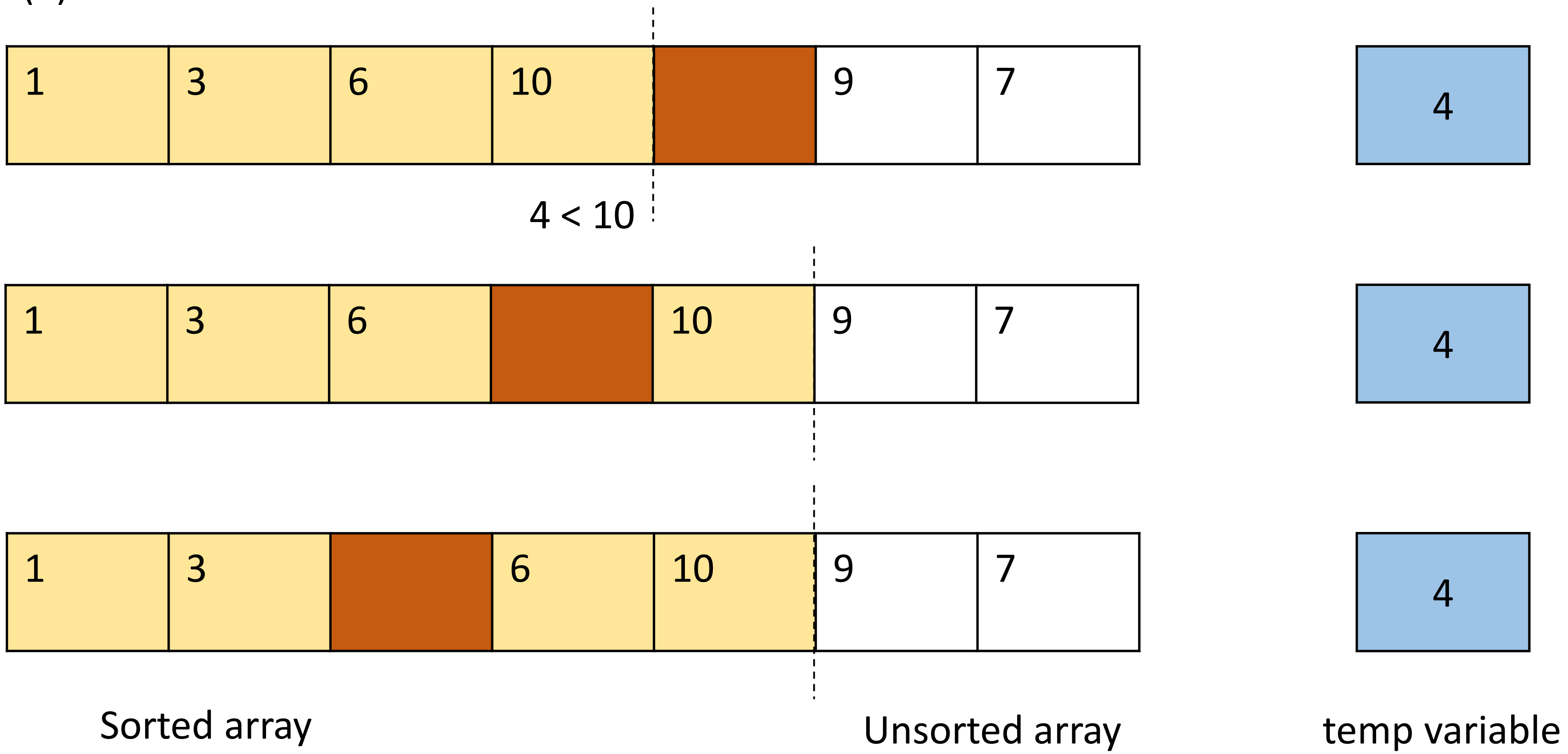


Insertion sort

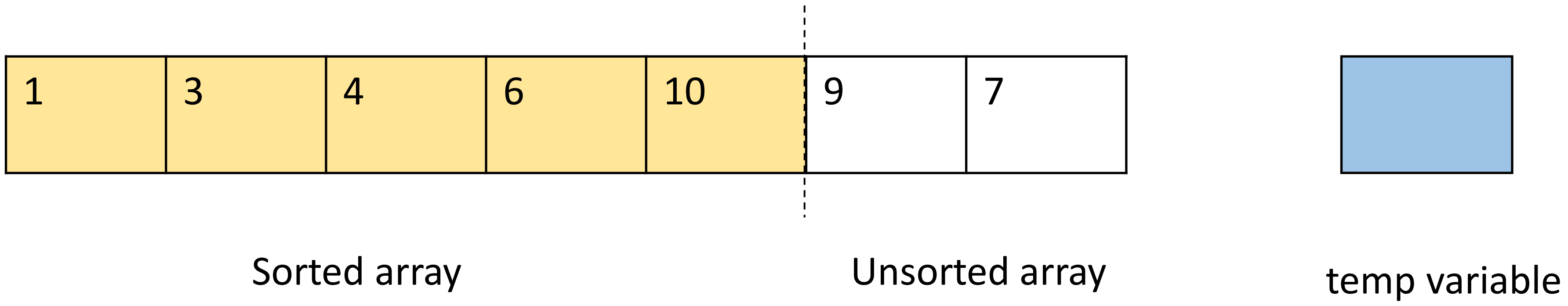


Insertion sort

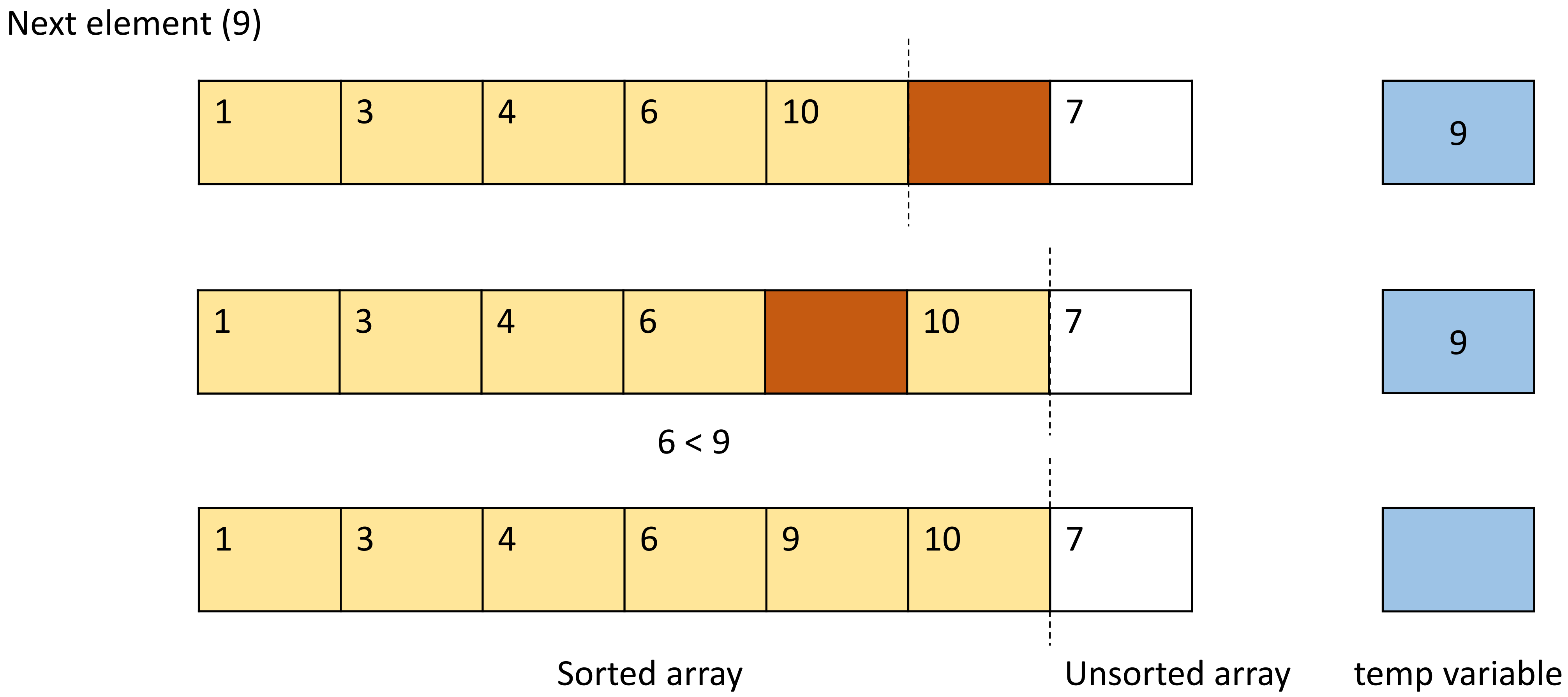
Next element (4)



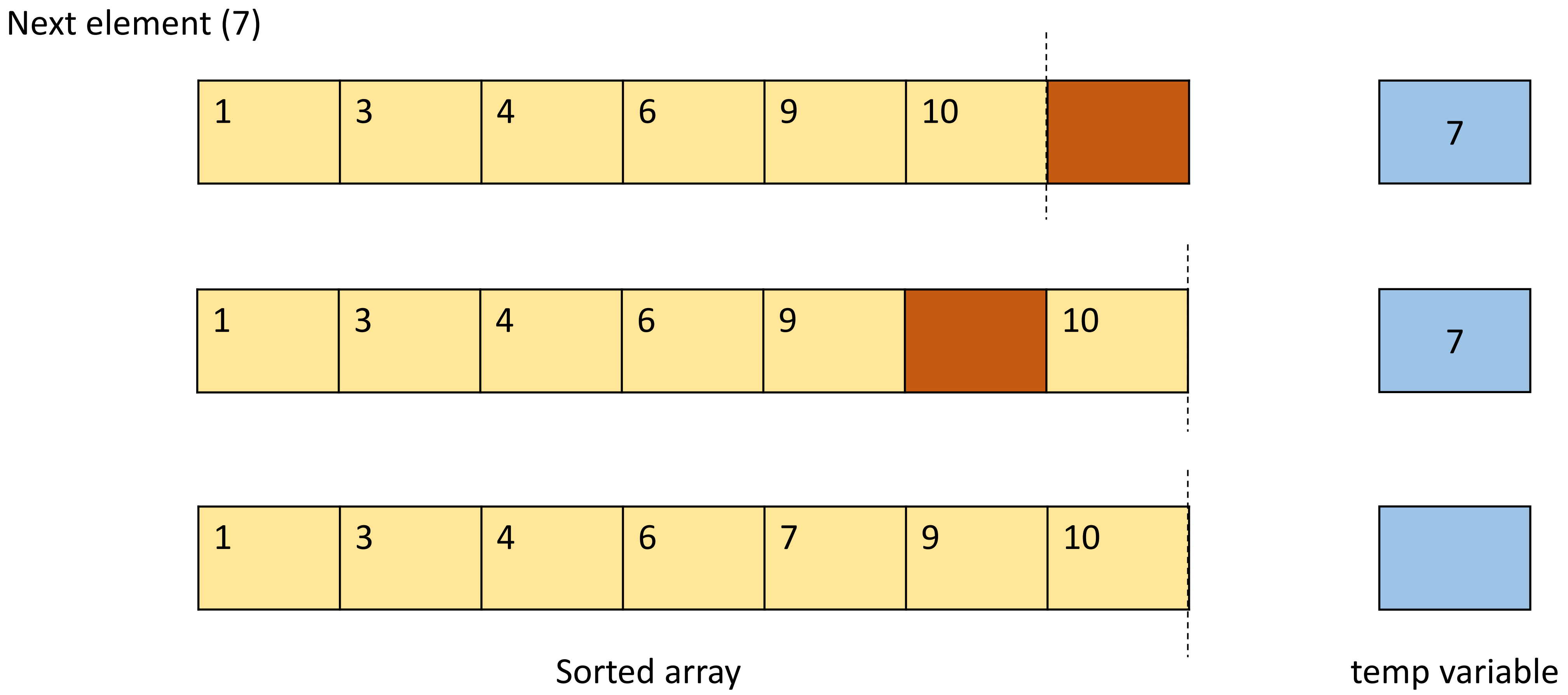
Insertion sort



Insertion sort



Insertion sort



Insertion sort ctd..

6 5 3 1 8 7 2 4

Insertion sort - Complexity

- **Time Complexity**
- Best O(n)
- Worst O(n^2)
- Average O(n^2)

- **Space complexity** O(1)

Selection sort

- Selection sort is an in-place comparison sorting algorithm.
- Has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.
- Simple algorithm
- Inefficient on large lists, and generally performs worse than the similar insertion sort.

Selection sort

5	2	8	10	4	1

Step 1: Select the first element as minimum.

5	2	8	10	4	1

Selection sort

Step 2: Compare minimum with the second element. If the second element is smaller than minimum, assign the second element as minimum

5	2	8	10	4	1

Compare 5 with 2
&
Assign 2 as minimum

Step 3: Compare minimum with the third element. Again, if the third element is smaller, then assign minimum to the third element otherwise do nothing. The process goes on until the last element.

5	2	8	10	4	1

Compare 2 with 8

$2 < 8$

Selection sort

5	2	8	10	4	1

Compare 2 with 10

2 < 10

5	2	8	10	4	1

Compare 2 with 4

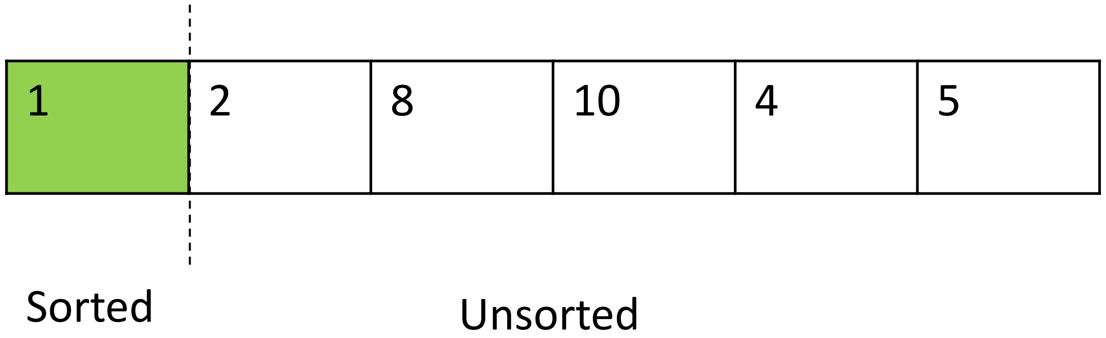
2 < 4

5	2	8	10	4	1

Compare 2 with 1
&
Assign 1 as minimum

Selection sort

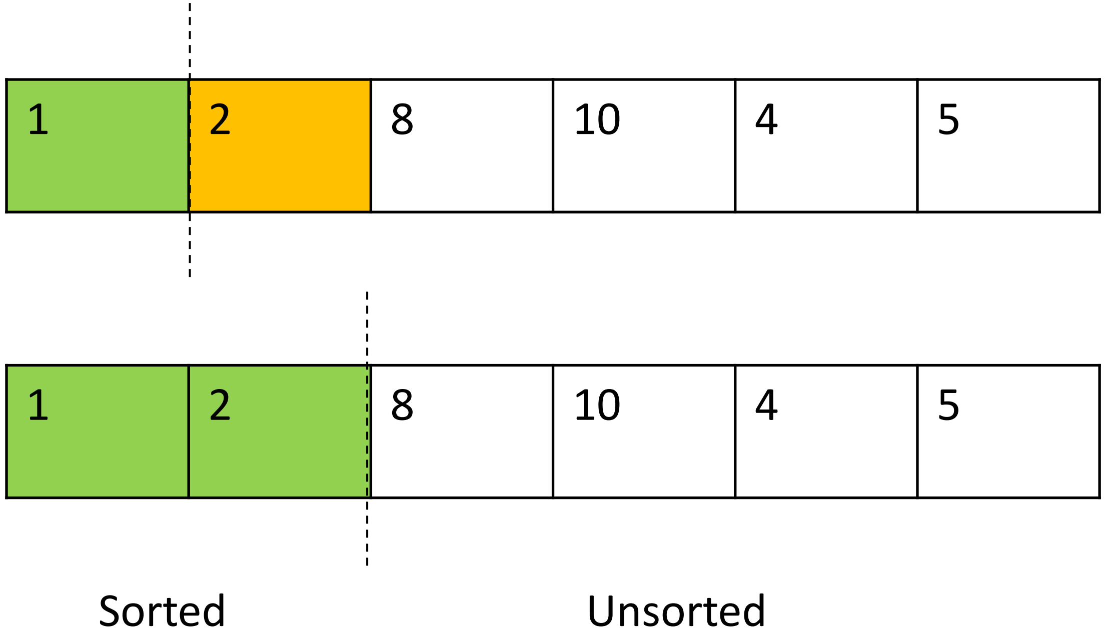
Step 4: After each iteration, minimum is placed in the front of the unsorted list.



Swap operation

Selection sort

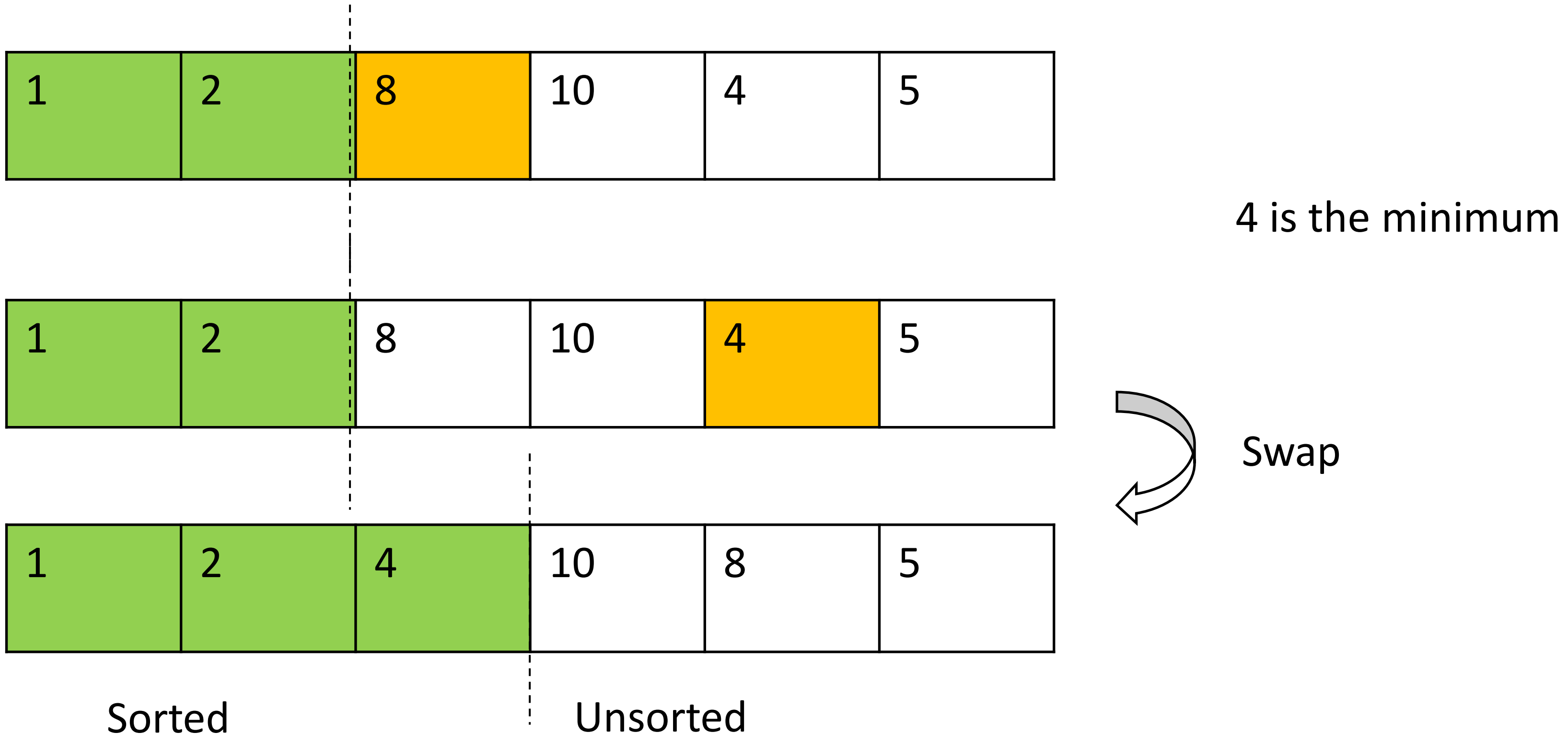
Step 5: For each iteration, indexing starts from the first unsorted element. Step 1 to 3 are repeated until all the elements are placed at their correct positions.



2 is the minimum

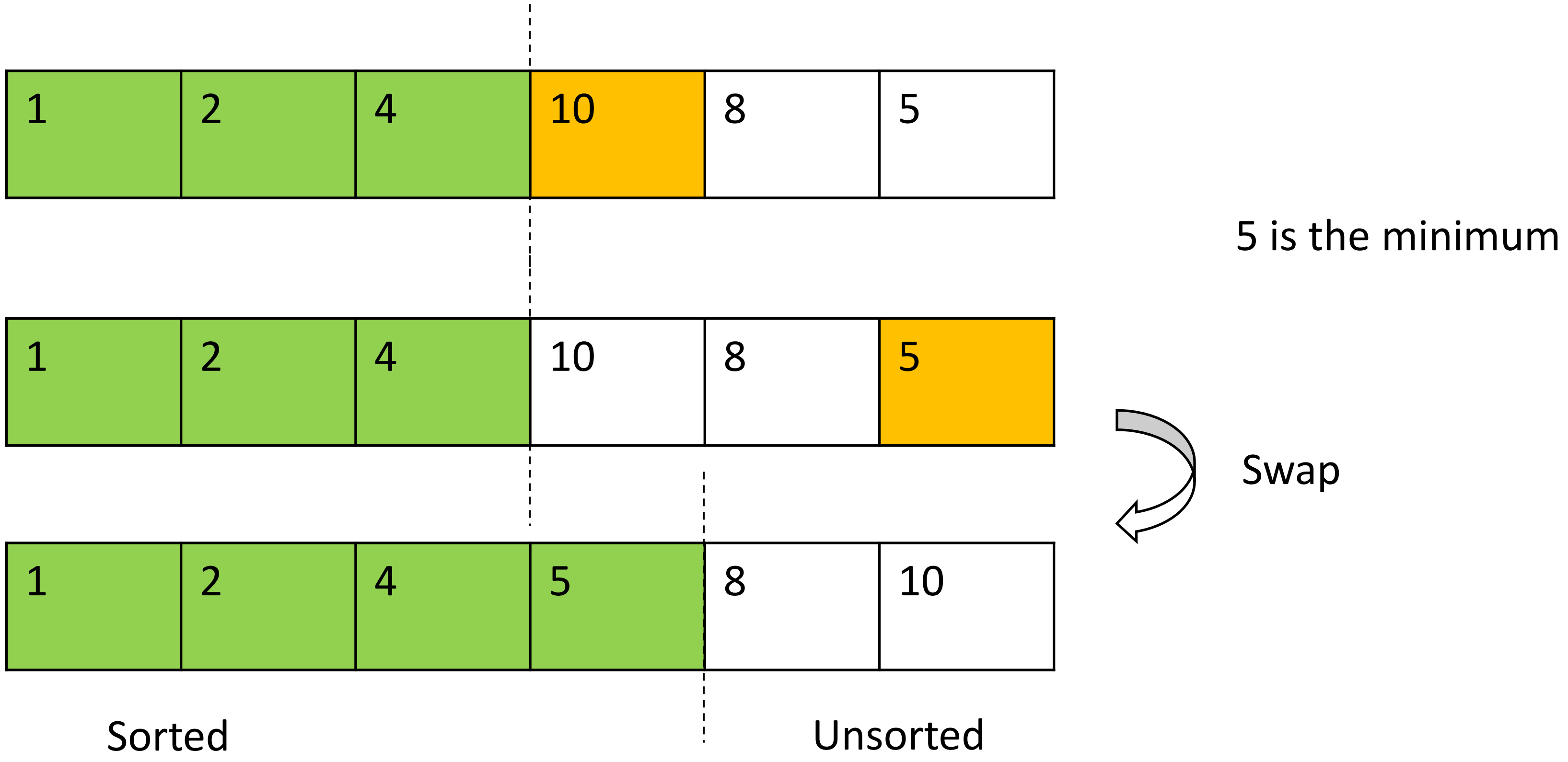
Selection sort

Next Element



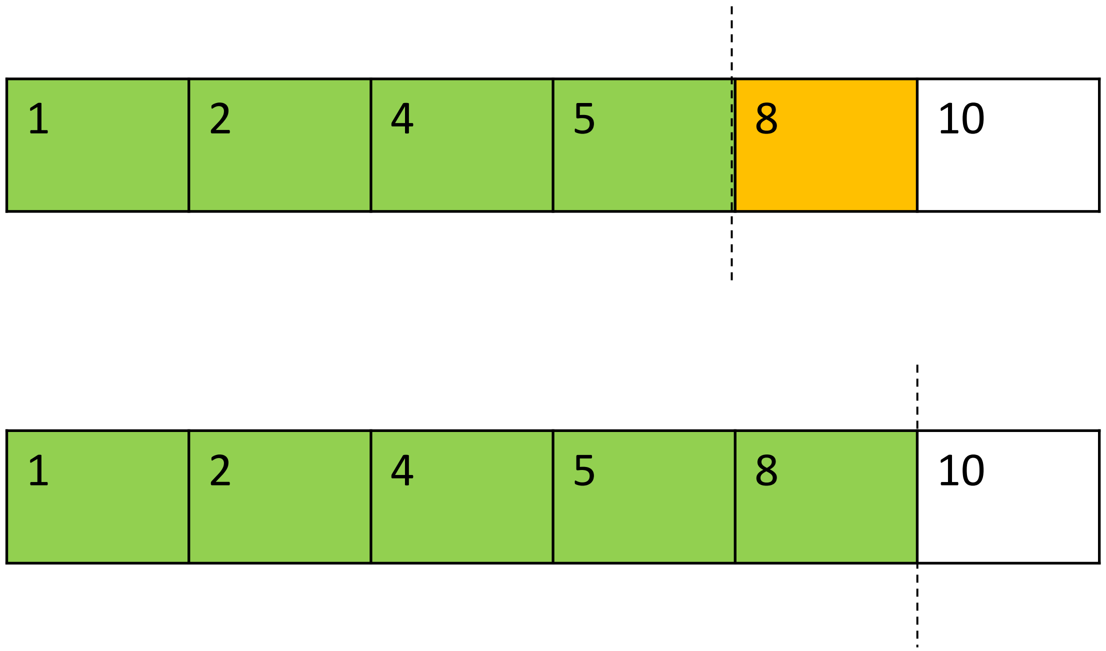
Selection sort

Next Element



Selection sort

Next Element

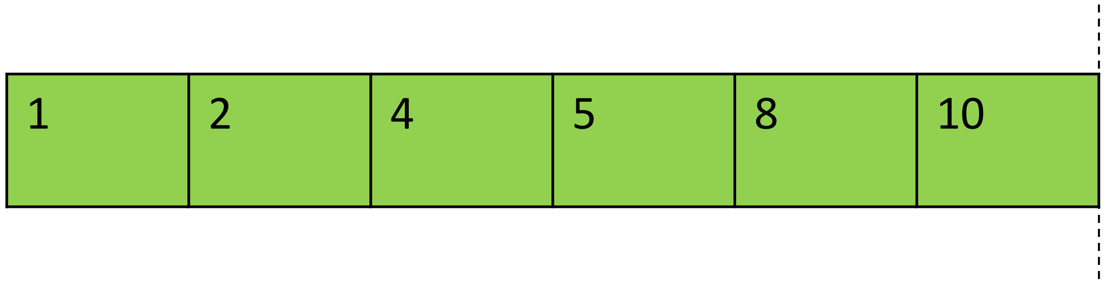


8 is the minimum

Sorted

Unsorted

Selection sort



Sorted

Selection sort

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7

Red color element is the minimum value

Selection sort - Complexity

- No of comparisons

```
(n-1) + (n-2) + (n-3) +.....+ 1 = n(n-1)/2
```

- $\simeq n^2$

Cycle	Number of Comparisons
1st	(n-1)
2nd	(n-2)
3rd	(n-3)
....	......
last	1

Selection sort

- **Time Complexity**
- Best   O(n^2)
- Worst O(n^2)
- Average O(n^2)

- **Space complexity** O(1)

Bubble sort

- **Bubble sort** is a sorting algorithm that repeatedly steps through the list, compares adjacent elements and swaps them if they are in the wrong order.
- The pass through the list is repeated until the list is sorted.
- Simple Algorithm

Bubble sort

Given Array

8	6	4	12	17	1

Bubble sort

First Iteration

Step 1: Starting from the first index, compare the first and the second elements.

8	6	4	12	17	1

Step 2: If the first element is greater than the second element, they are swapped.

6	8	4	12	17	1

Bubble sort

Step 3: Now, compare the second and the third elements. Swap them if they are not in order.

6	8	4	12	17	1

6	4	8	12	17	1

Bubble sort

Step 4: The above process goes on until the last element.

6	4	8	12	17	1

6	4	8	12	17	1

6	4	8	12	17	1

Bubble sort

6	4	8	12	1	17

Bubble sort

Remaining Iteration

The same process goes on for the remaining iterations.
After each iteration, the largest element among the unsorted elements is placed at the end.

6	4	8	12	1	17

Swap operation

4	6	8	12	1	17

Bubble sort

4	6	8	12	1	17

4	6	8	12	1	17

4	6	8	12	1	17

Swap

4	6	8	1	12	17

Bubble sort

Next Iteration

4	6	8	1	12	17

4	6	8	1	12	17

4	6	8	1	12	17

Swap

4	6	1	8	12	17

Bubble sort

Finally

1	4	6	8	12	17

Bubble sort

- **Time Complexity**
- Best O(n)
- Worst O(n^2)
- Average O(n^2)

- **Space complexity** O(1)

Bubble sort

6 5 3 1 8 7 2 4

Thank You