Bubble Sort

In this algorithm,

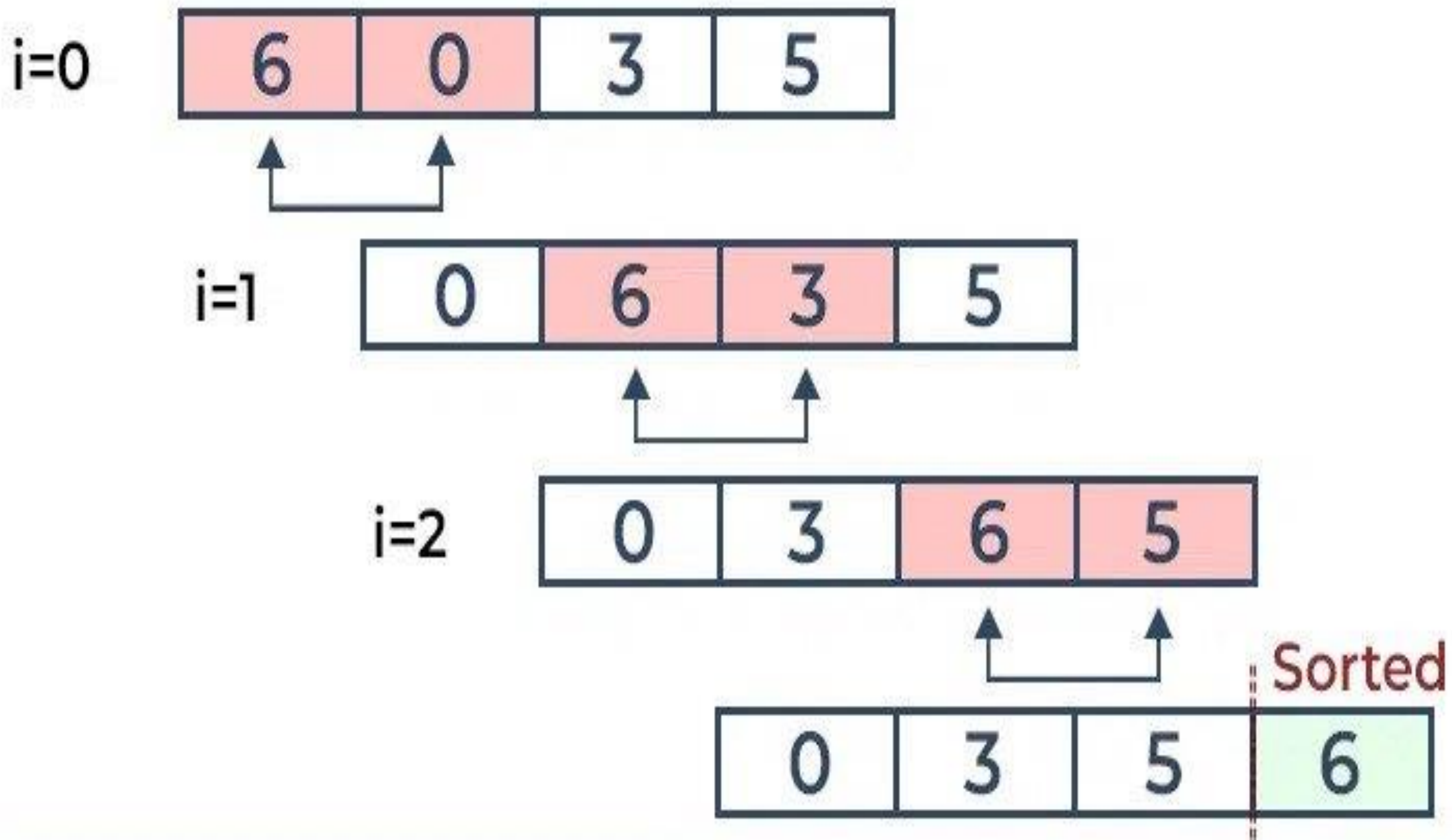
- *traverse from left and compare adjacent elements and the higher one is placed at right side.*

- *In this way, the largest element is moved to the rightmost end at first.*

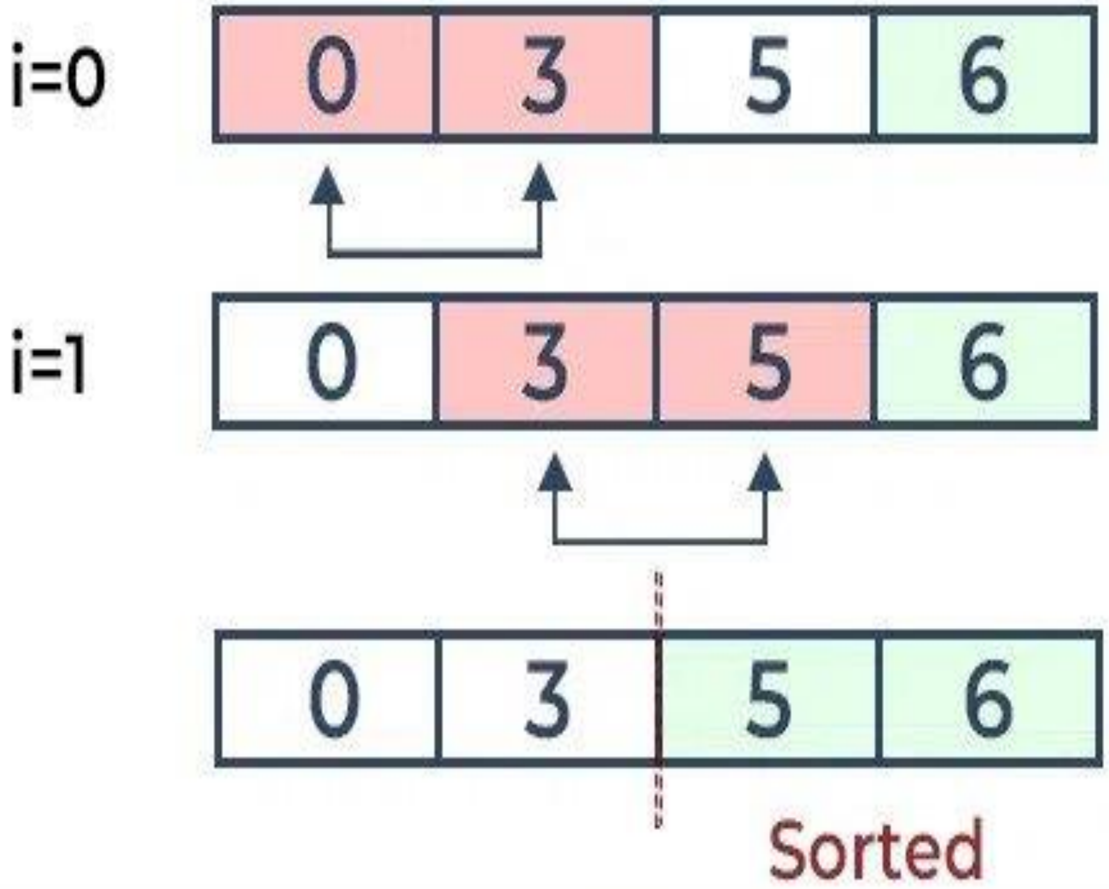
- *This process is then continued to find the second largest and place it and so on until the data is sorted.*

STEP 01

Placing the 1st largest element at Correct position



STEP 02 Placing 2nd largest element at Correct position



i=0 | 0 | 3 | 5 | 6

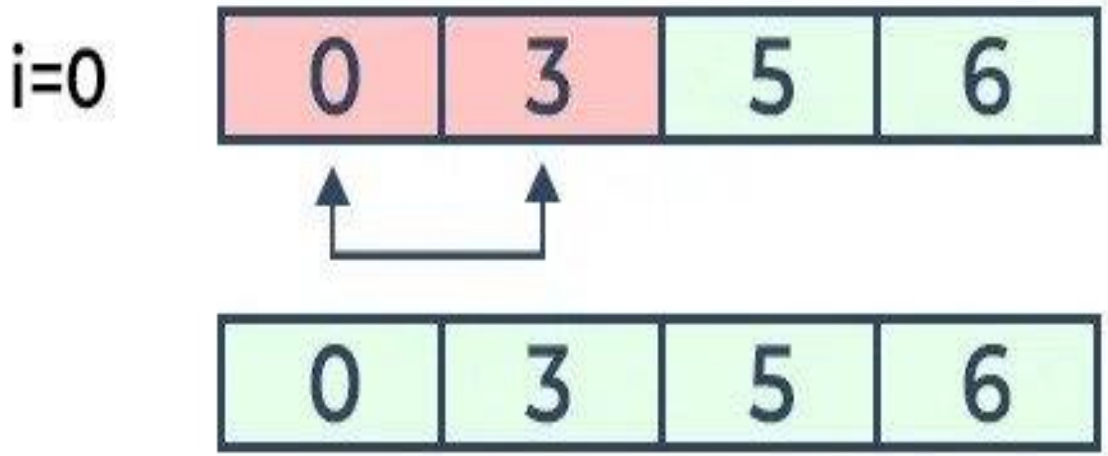
i=1 | 0 | 3 | 5 | 6

0 | 3 | 5 | 6

Sorted

STEP 03

Placing 3rd largest element at Correct position



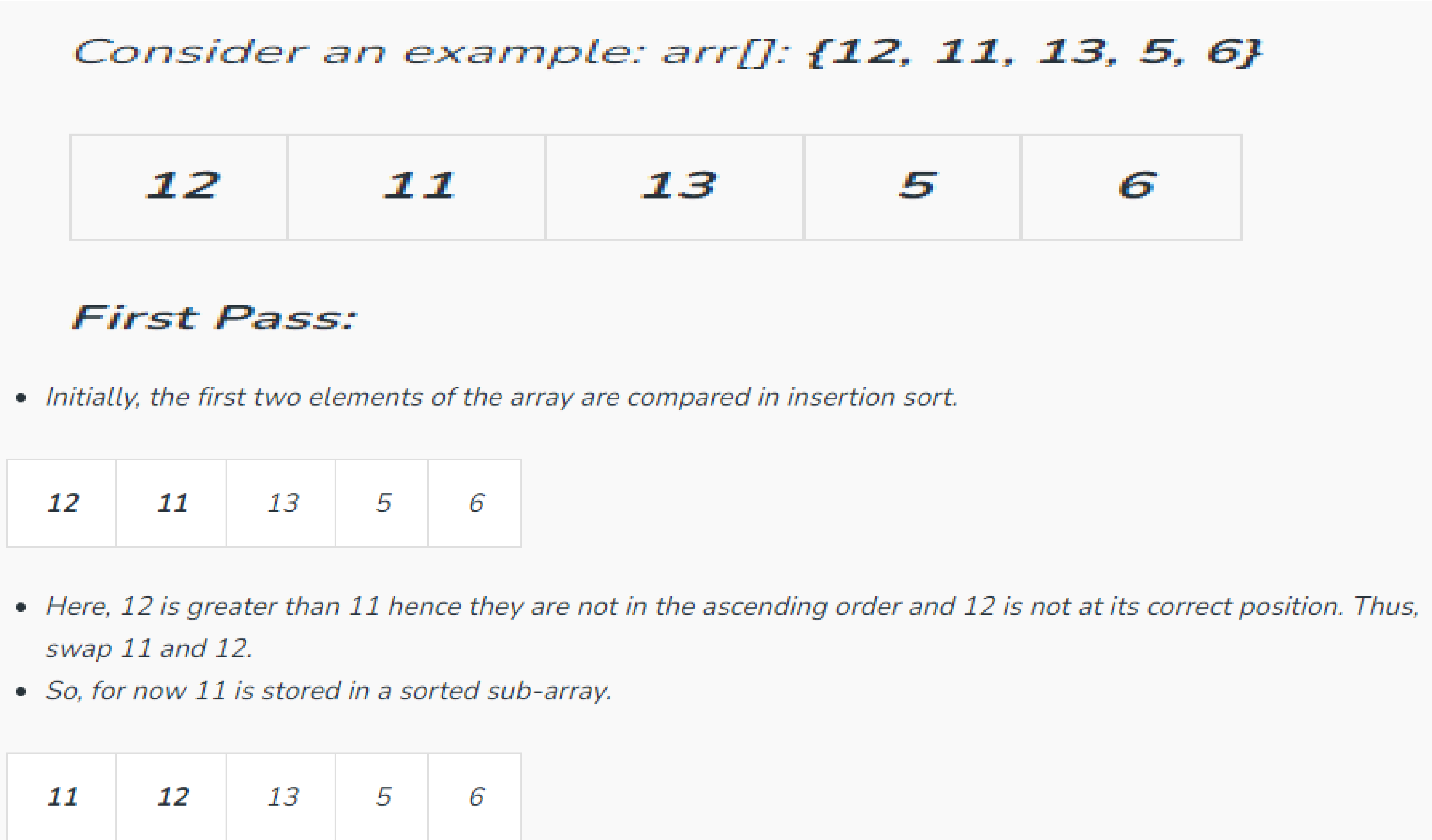
i=0

0	3	5	6

0	3	5	6

Sorted array

Insertion Sort



Consider an example: arr[]: {12, 11, 13, 5, 6}

12	**11**	**13**	**5**	**6**

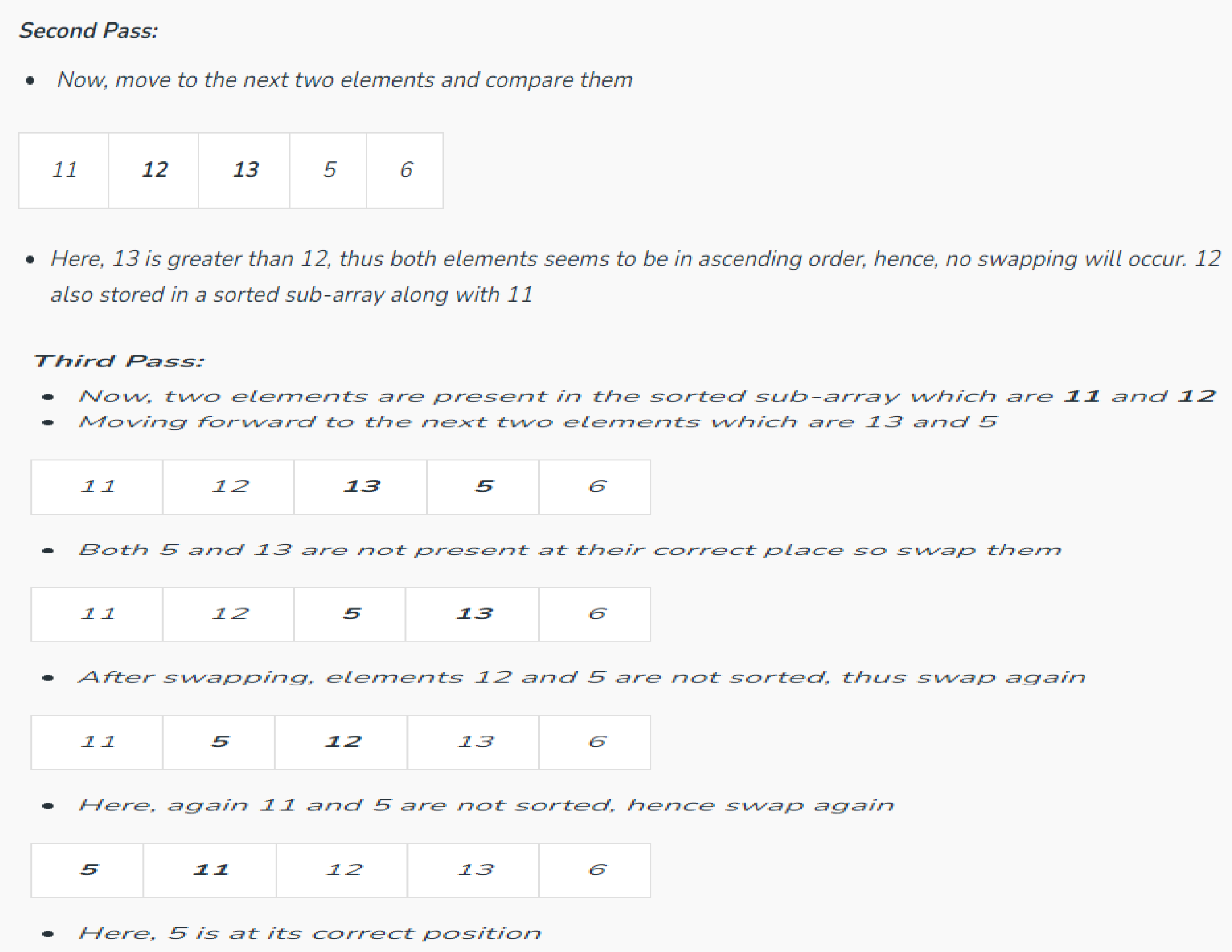
First Pass:

- *Initially, the first two elements of the array are compared in insertion sort.*

12	**11**	13	5	6

- *Here, 12 is greater than 11 hence they are not in the ascending order and 12 is not at its correct position. Thus, swap 11 and 12.*
- *So, for now 11 is stored in a sorted sub-array.*

11	**12**	13	5	6



Second Pass:

- *Now, move to the next two elements and compare them*

11	**12**	**13**	5	6

- *Here, 13 is greater than 12, thus both elements seems to be in ascending order, hence, no swapping will occur. 12 also stored in a sorted sub-array along with 11*

Third Pass:

- *Now, two elements are present in the sorted sub-array which are **11** and **12***
- *Moving forward to the next two elements which are 13 and 5*

11	12	**13**	**5**	6

- *Both 5 and 13 are not present at their correct place so swap them*

11	12	**5**	**13**	6

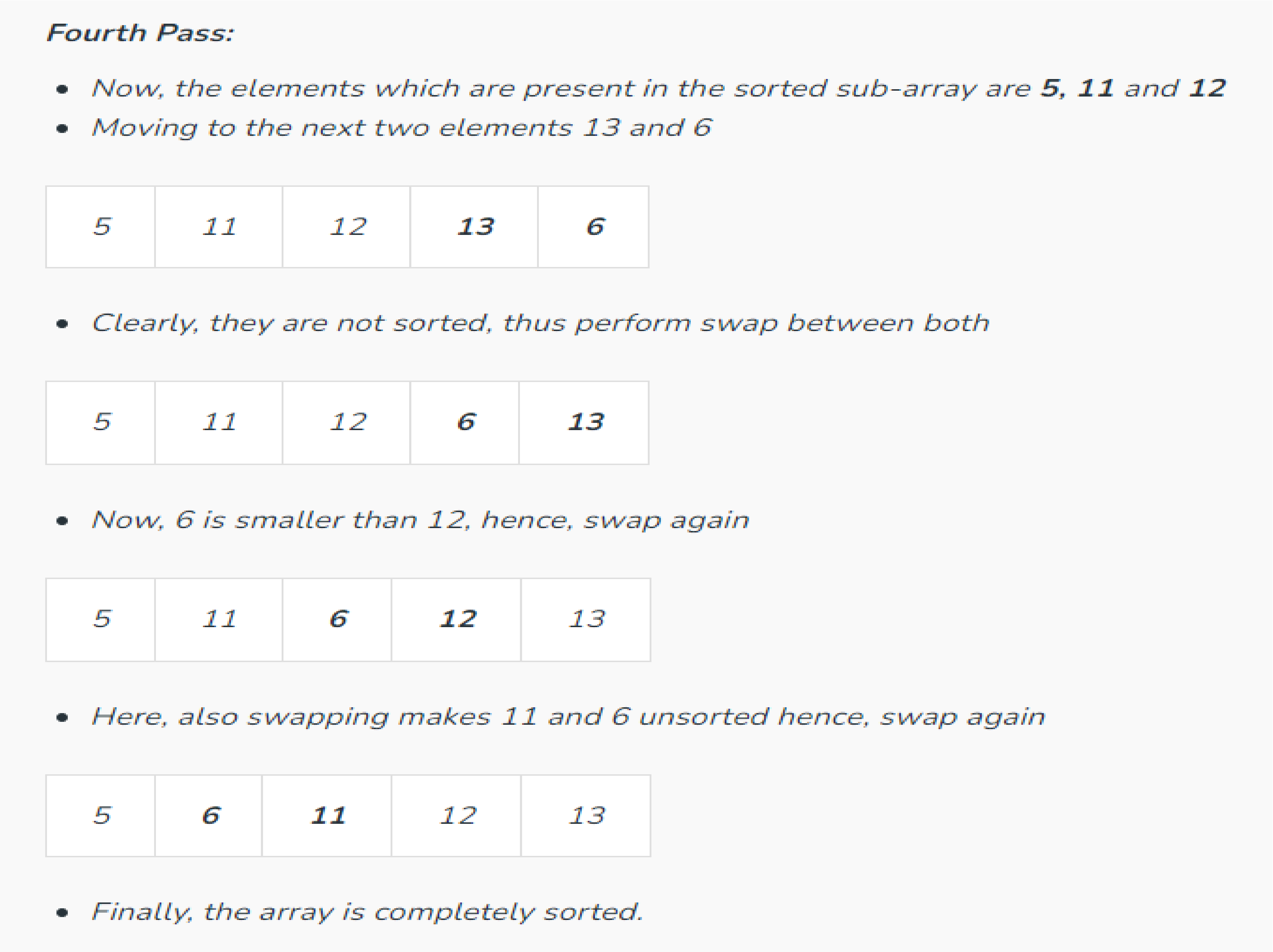
- *After swapping, elements 12 and 5 are not sorted, thus swap again*

11	**5**	**12**	13	6

- *Here, again 11 and 5 are not sorted, hence swap again*

5	**11**	12	13	6

- *Here, 5 is at its correct position*



Fourth Pass:

- *Now, the elements which are present in the sorted sub-array are **5, 11** and **12***
- *Moving to the next two elements 13 and 6*

5	*11*	*12*	***13***	***6***

- *Clearly, they are not sorted, thus perform swap between both*

5	*11*	*12*	***6***	***13***

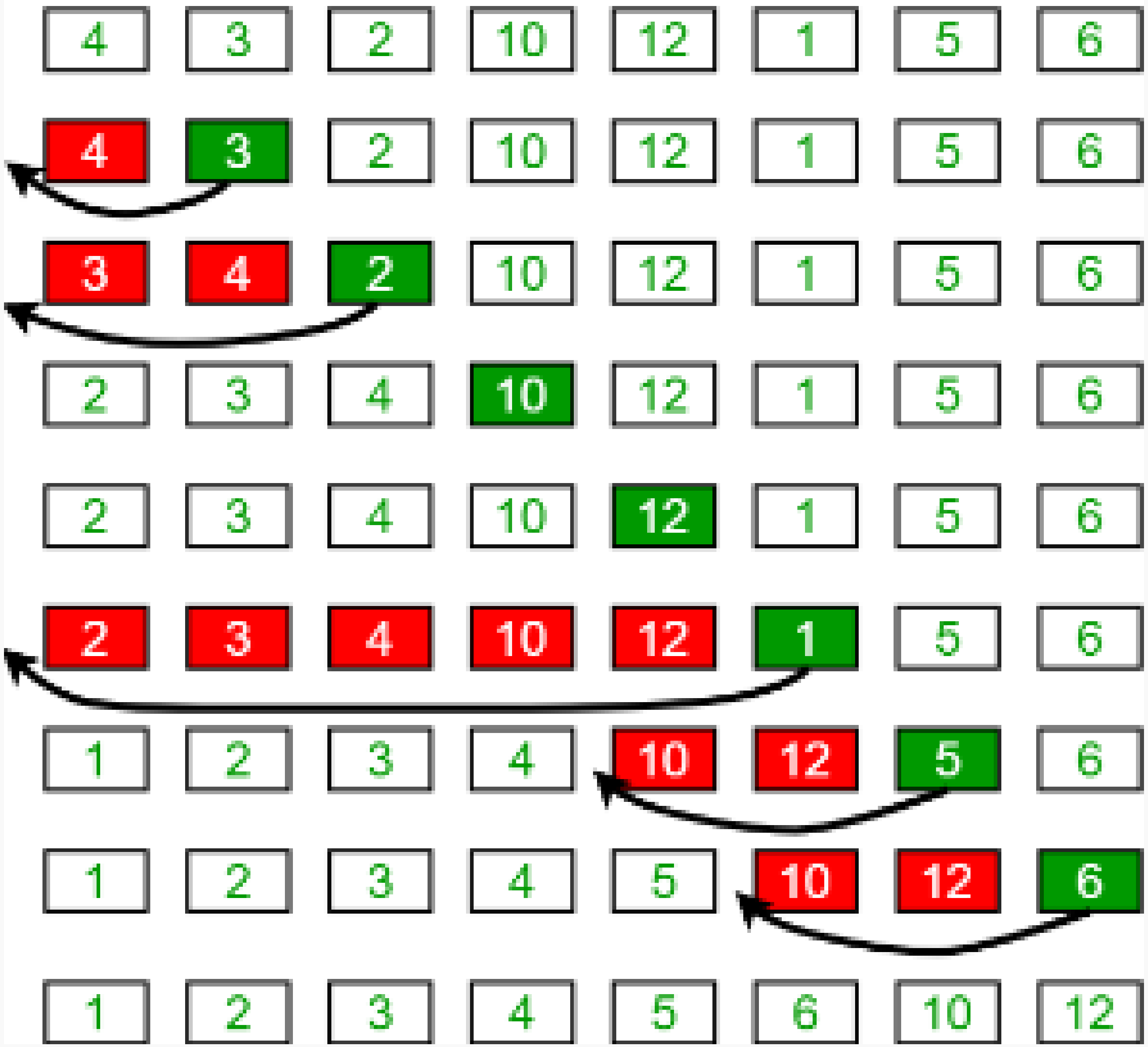
- *Now, 6 is smaller than 12, hence, swap again*

5	*11*	***6***	***12***	*13*

- *Here, also swapping makes 11 and 6 unsorted hence, swap again*

5	***6***	***11***	*12*	*13*

- *Finally, the array is completely sorted.*



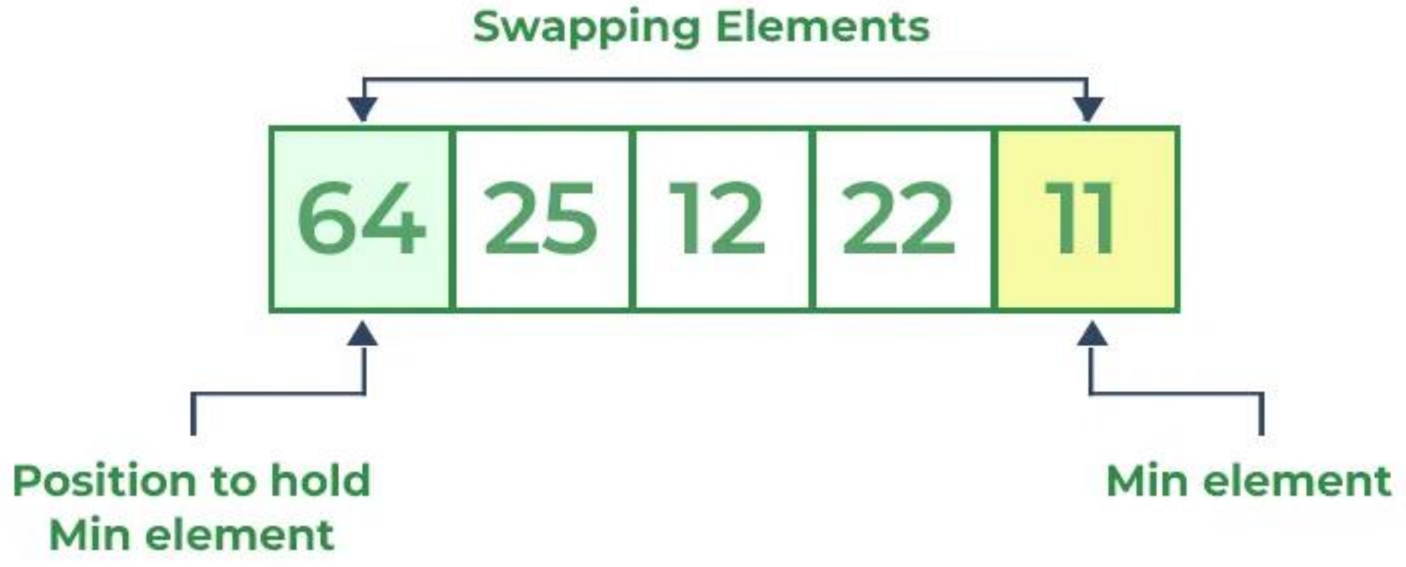
4	3	2	10	12	1	5	6
4	3	2	10	12	1	5	6
3	4	2	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
2	3	4	10	12	1	5	6
1	2	3	4	10	12	5	6
1	2	3	4	5	10	12	6
1	2	3	4	5	6	10	12

Selection Sort

Lets consider the following array as an example: ***arr[] = {64, 25, 12, 22, 11}***

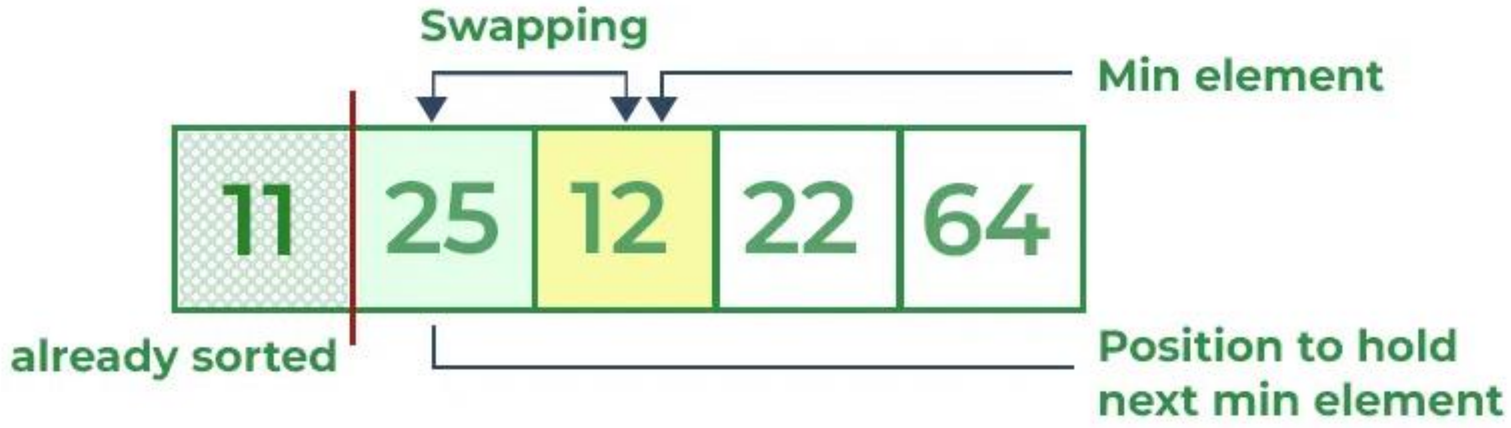
First pass:

- *For the first position in the sorted array, the whole array is traversed from index 0 to 4 sequentially. The first position where* ***64*** *is stored presently, after traversing whole array it is clear that* ***11*** *is the lowest value.*
- *Thus, replace 64 with 11. After one iteration* ***11****, which happens to be the least value in the array, tends to appear in the first position of the sorted list.*



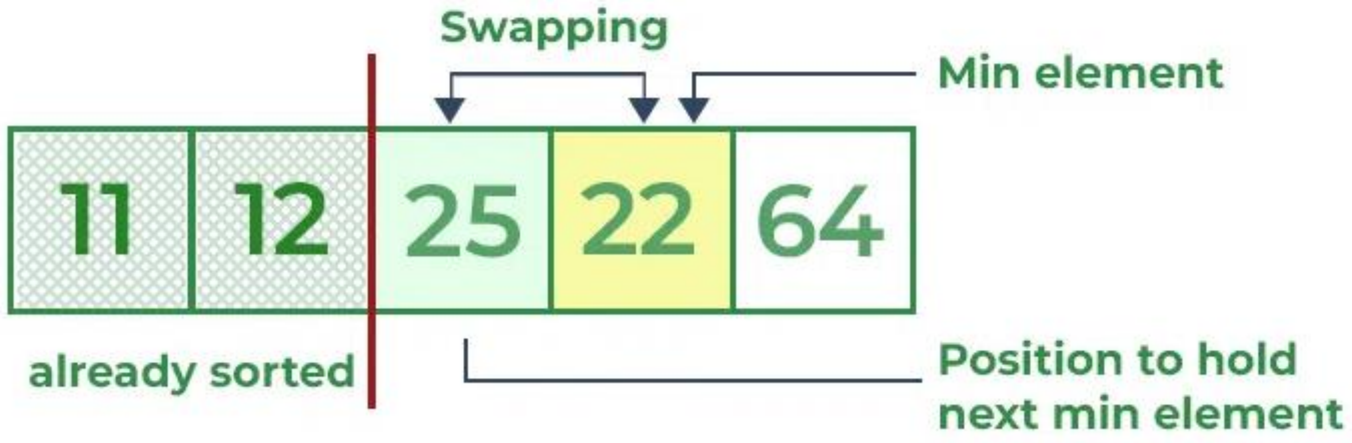
Second Pass:

- *For the second position, where 25 is present, again traverse the rest of the array in a sequential manner.*
- *After traversing, we found that **12** is the second lowest value in the array and it should appear at the second place in the array, thus swap these values.*



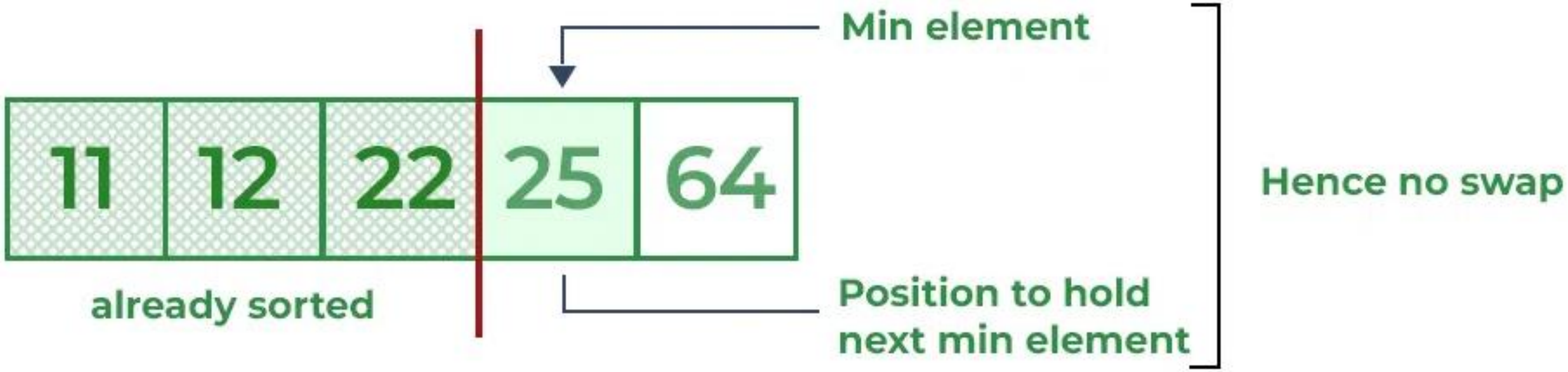
Third Pass:

- *Now, for third place, where **25** is present again traverse the rest of the array and find the third least value present in the array.*
- *While traversing, **22** came out to be the third least value and it should appear at the third place in the array, thus swap **22** with element present at third position.*



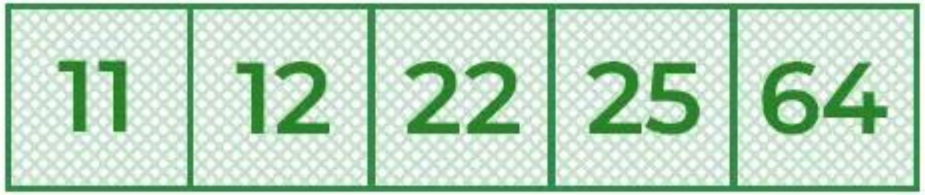
Fourth pass:

- *Similarly, for fourth position traverse the rest of the array and find the fourth least element in the array*
- *As **25** is the 4th lowest value hence, it will place at the fourth position.*



Fifth Pass:

- *At last the largest value present in the array automatically get placed at the last position in the array*
- *The resulted array is the sorted array.*



11	12	22	25	64

Sorted array