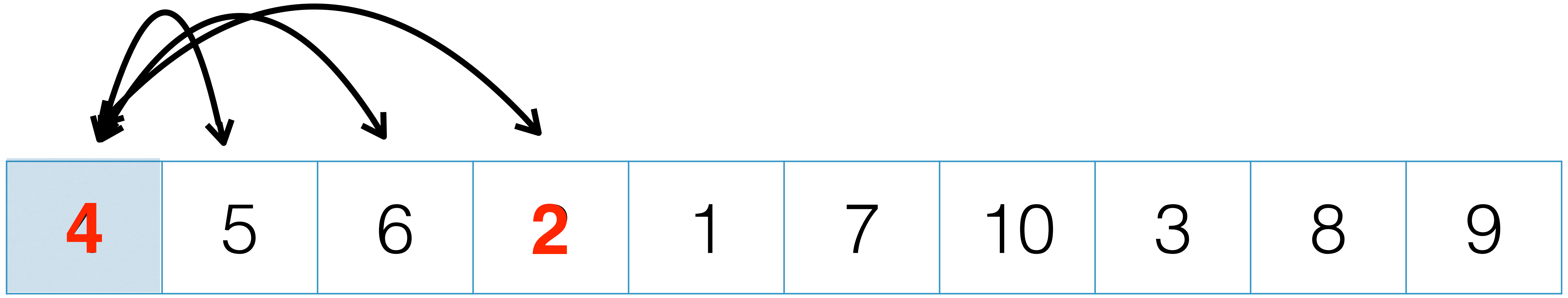
LET'S SEE SOME SORTING ALGORITHMS
- THE SIMPLE ONES FIRST

SELECTION SORT

AT EACH ITERATION 1 ELEMENT IS SELECTED AND COMPARED WITH EVERY OTHER ELEMENT IN THE LIST TO FIND THE SMALLEST ONE

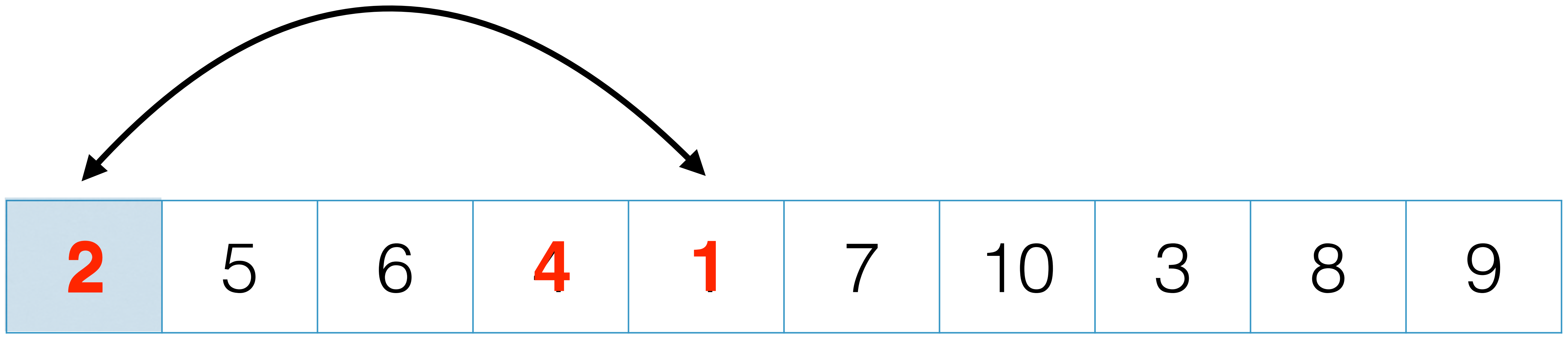
FIRST WE FIND THE SMALLEST ELEMENT, GET IT INTO THE FIRST POSITION, NEXT WE FIND THE SECOND SMALLEST TILL THE ENTIRE LIST IS SORTED

SELECTION SORT



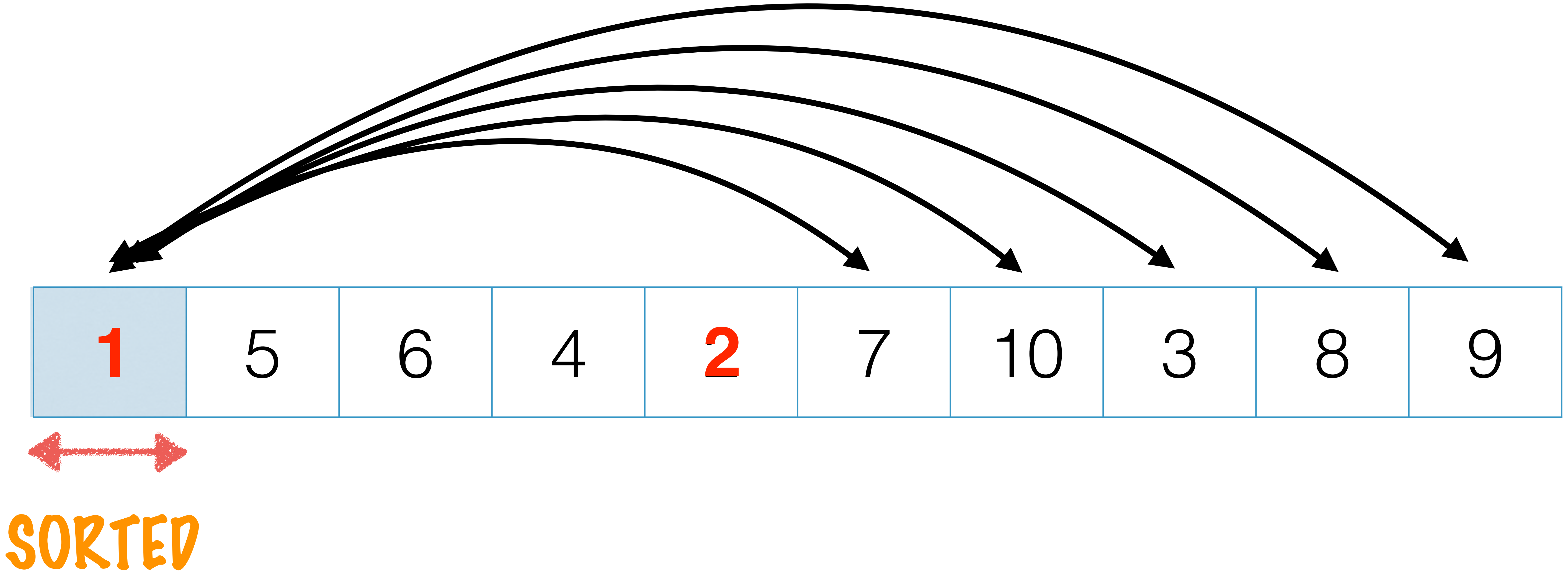
4	5	6	2	1	7	10	3	8	9

SELECTION SORT



2	5	6	4	1	7	10	3	8	9

SELECTION SORT

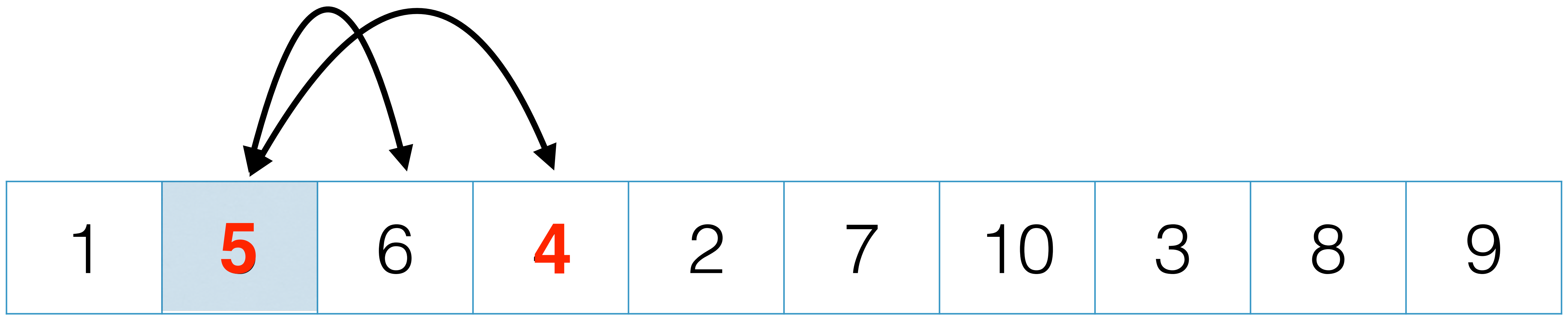


1	5	6	4	2	7	10	3	8	9

SORTED

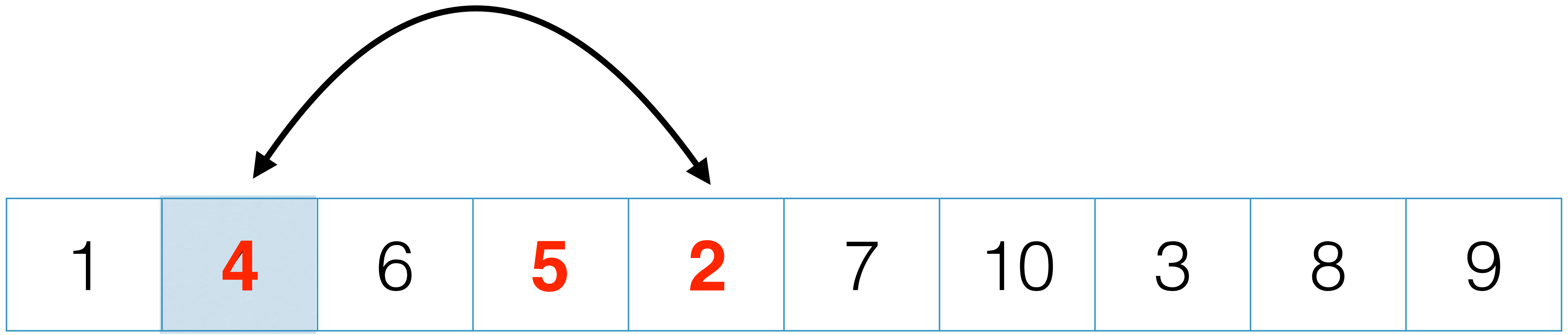
1 IS NOW IN THE
CORRECT POSITION

SELECTION SORT



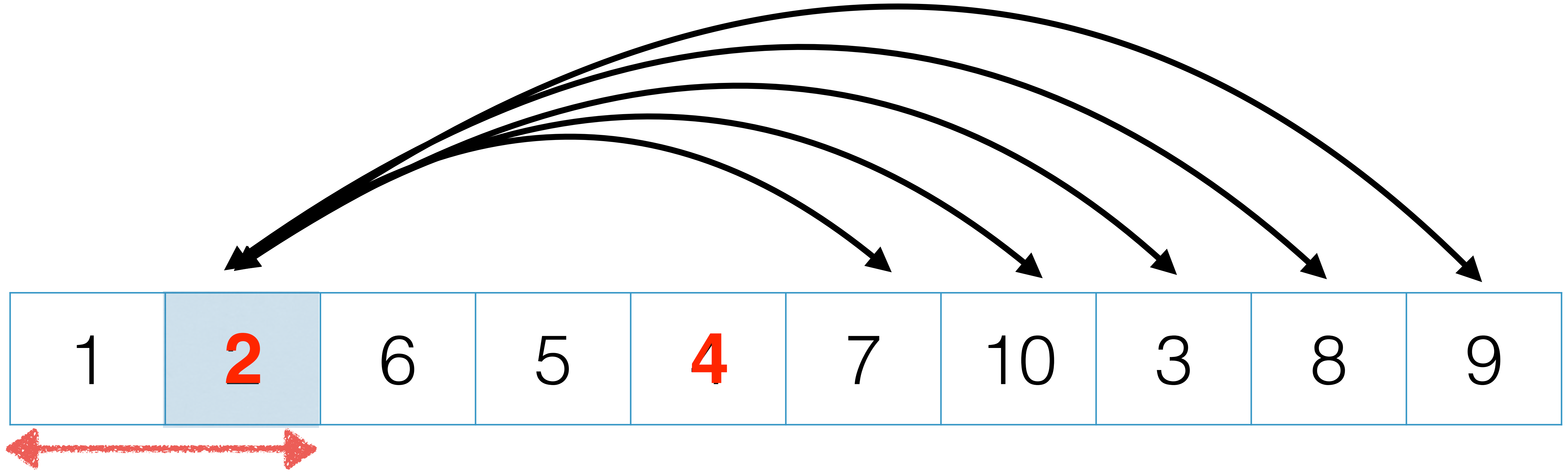
1	**5**	6	**4**	2	7	10	3	8	9

SELECTION SORT



1	**4**	6	**5**	**2**	7	10	3	8	9

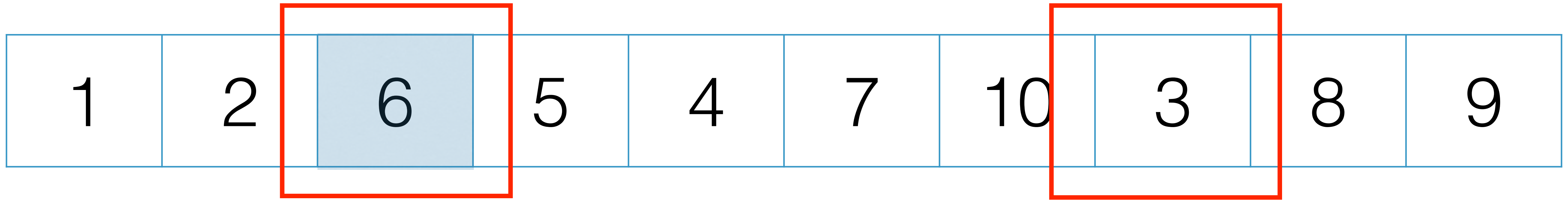
SELECTION SORT



SORTED

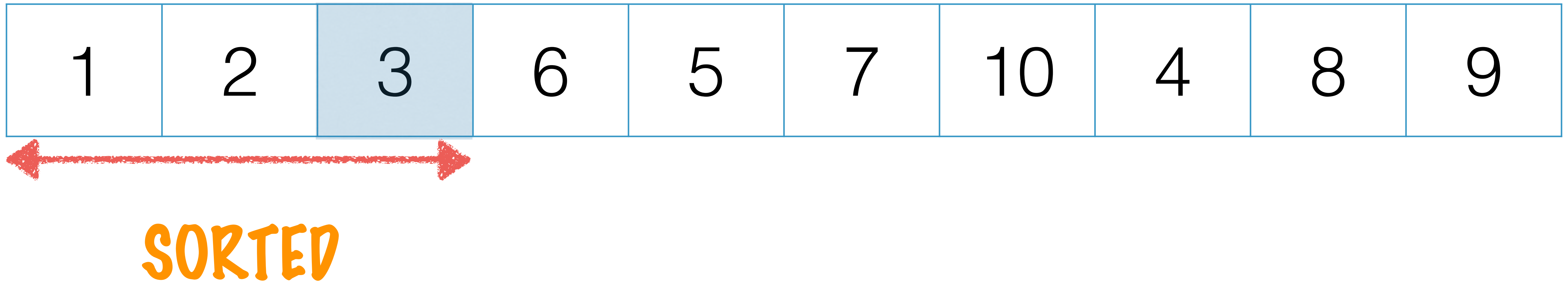
1 AND 2 ARE NOW IN THE CORRECT POSITION

SELECTION SORT



1	2	6	5	4	7	10	3	8	9

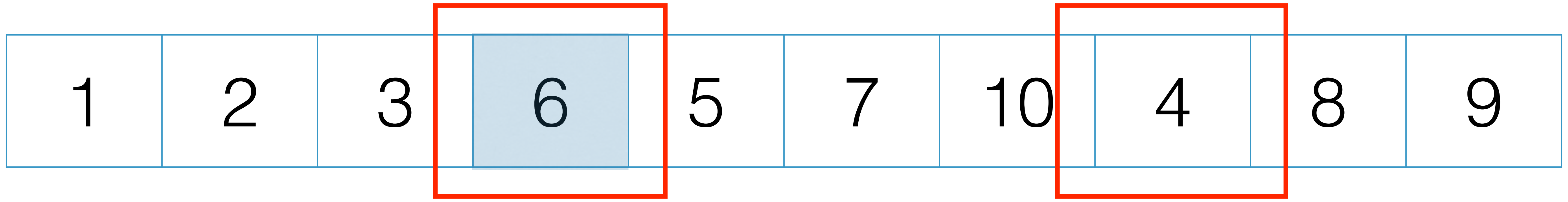
SELECTION SORT



1	2	3	6	5	7	10	4	8	9

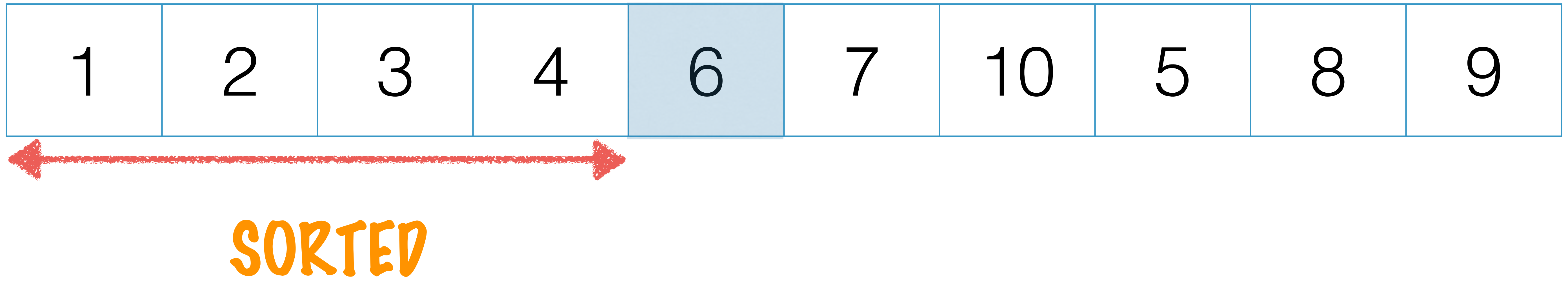
1, 2 AND 3 ARE NOW IN
THE CORRECT POSITION

SELECTION SORT



1	2	3	6	5	7	10	4	8	9

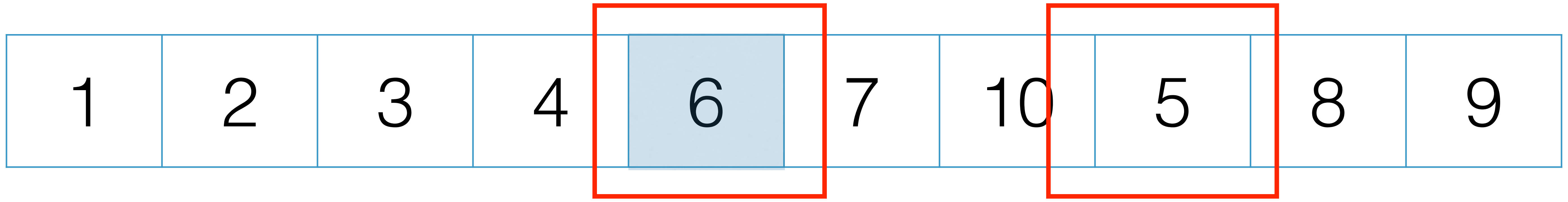
SELECTION SORT



1	2	3	4	6	7	10	5	8	9

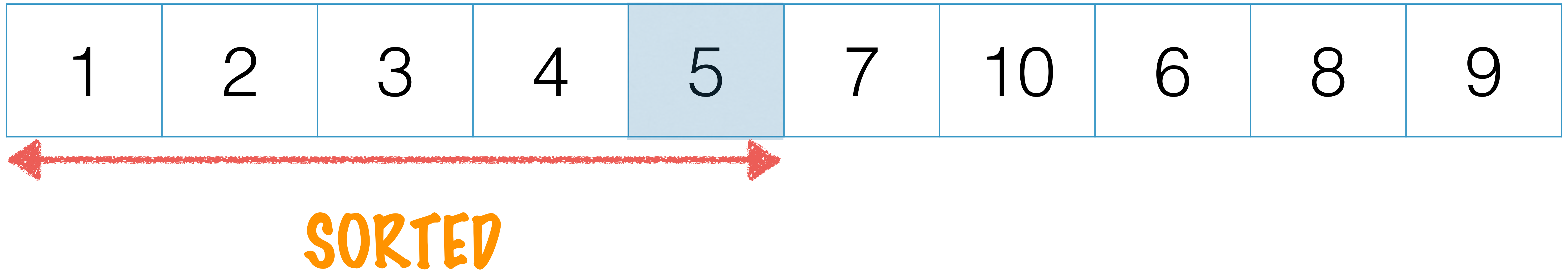
SORTED

SELECTION SORT



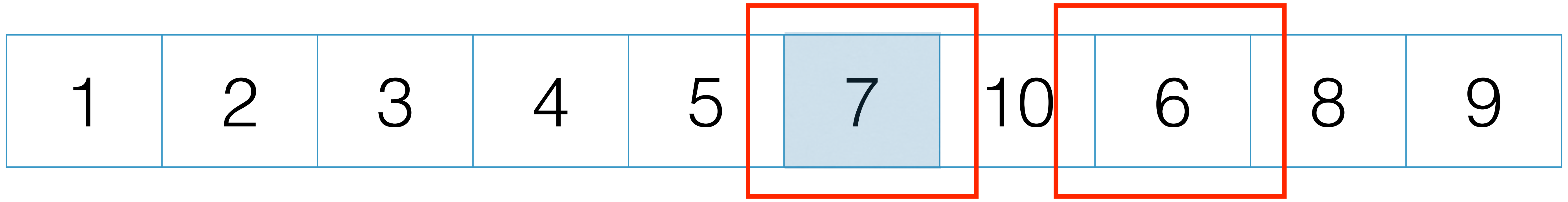
1	2	3	4	6	7	10	5	8	9

SELECTION SORT



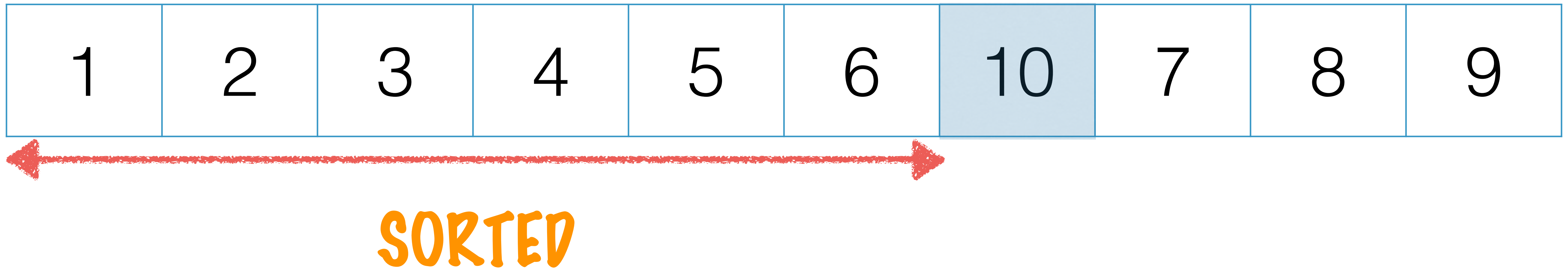
1	2	3	4	5	7	10	6	8	9

SELECTION SORT



1	2	3	4	5	7	10	6	8	9

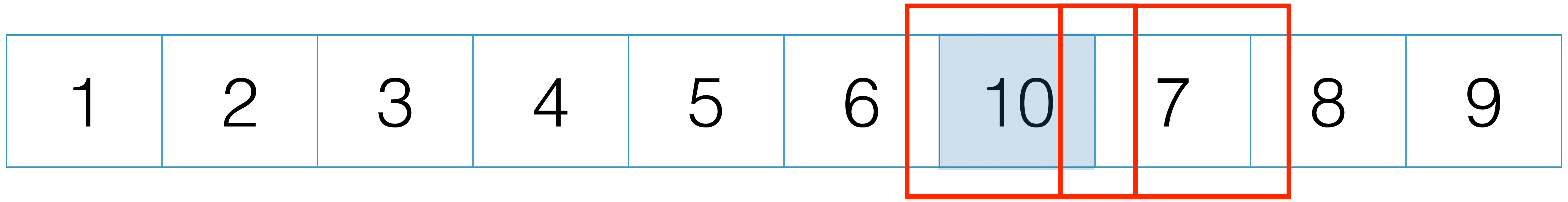
SELECTION SORT



1	2	3	4	5	6	10	7	8	9

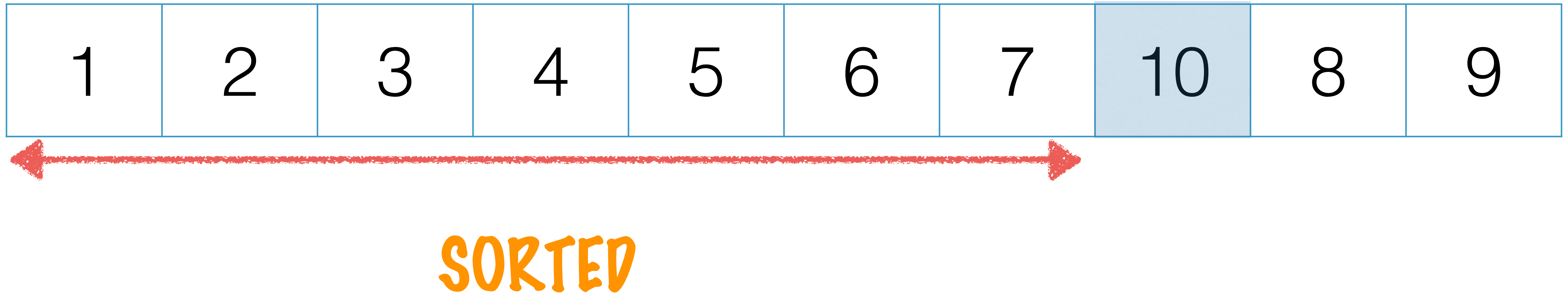
SORTED

SELECTION SORT



1	2	3	4	5	6	10	7	8	9

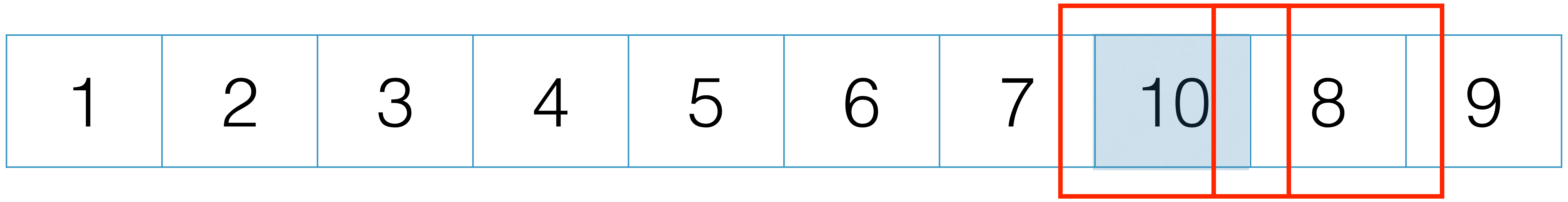
SELECTION SORT



1	2	3	4	5	6	7	10	8	9

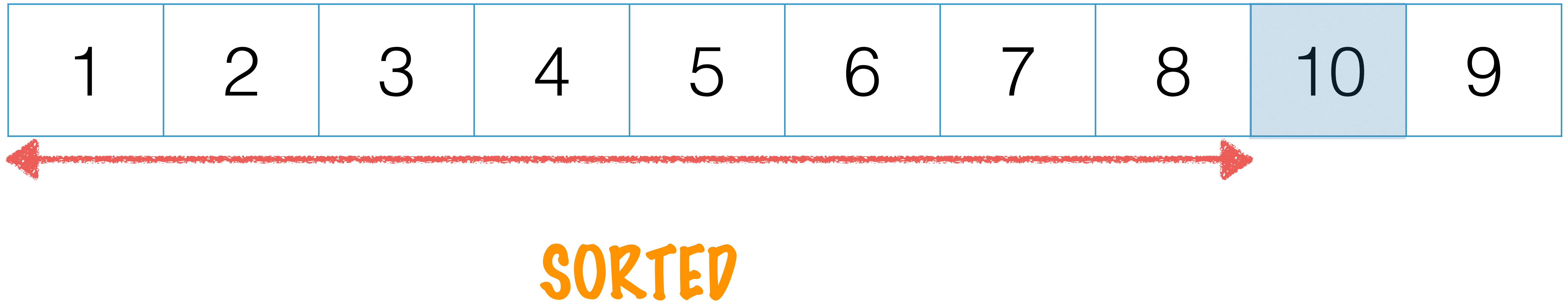
SORTED

SELECTION SORT



1	2	3	4	5	6	7	10	8	9

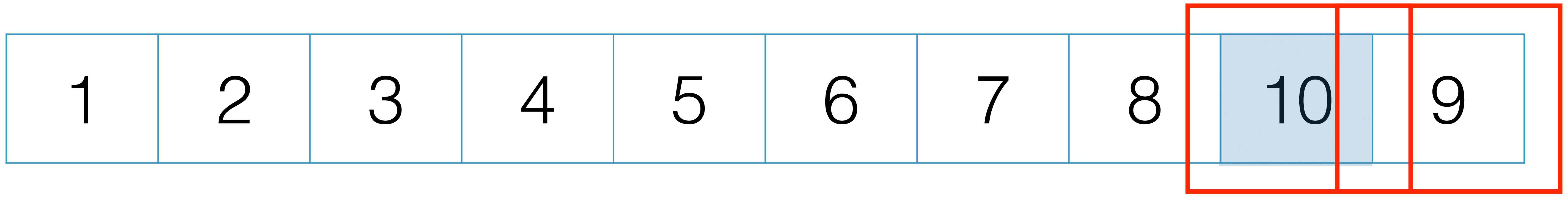
SELECTION SORT



1	2	3	4	5	6	7	8	10	9

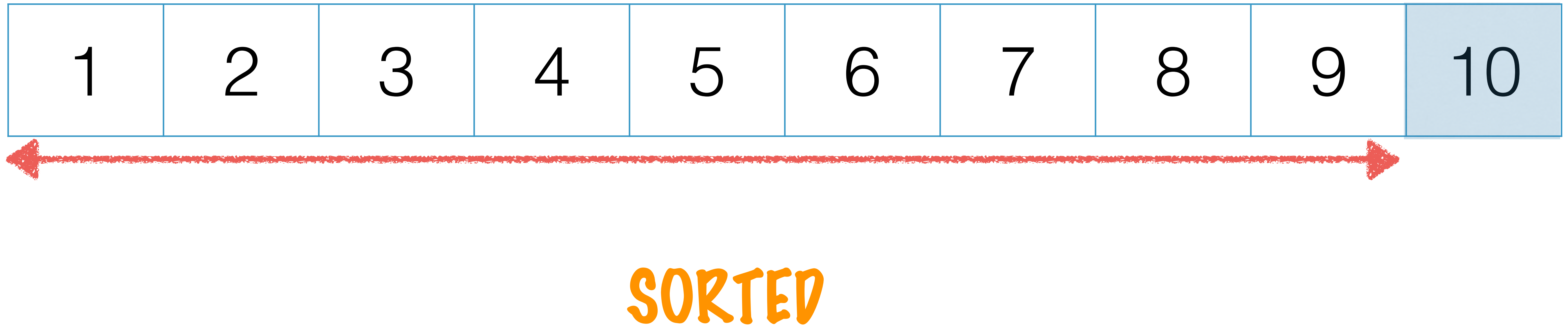
SORTED

SELECTION SORT



1	2	3	4	5	6	7	8	10	9

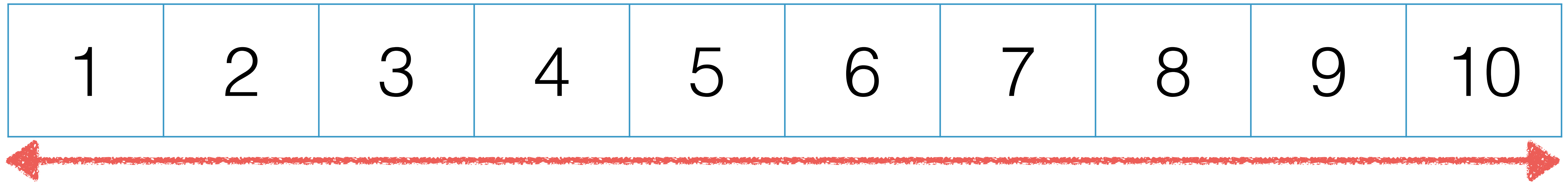
SELECTION SORT



1	2	3	4	5	6	7	8	9	10

SORTED

SELECTION SORT



1	2	3	4	5	6	7	8	9	10

SORTED

FULLY SORTED LIST!

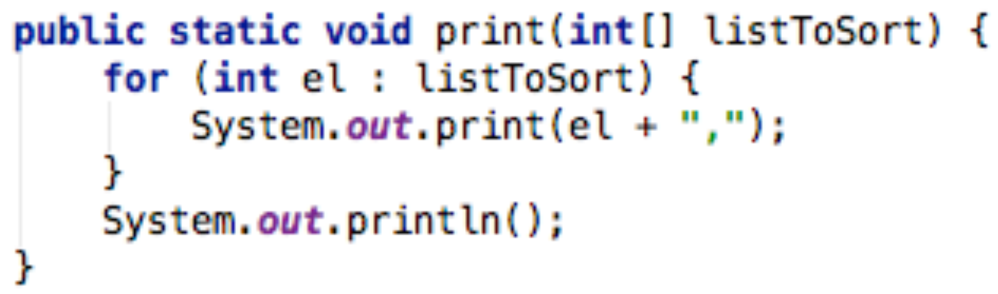
SELECTION SORT SELECTS ONE ELEMENT AT A TIME, COMPARES IT TO ALL OTHER ELEMENTS IN THE LIST

THE CORRECT POSITION FOR THAT SELECTED ELEMENT IS FOUND BEFORE MOVING ON TO THE NEXT ELEMENT

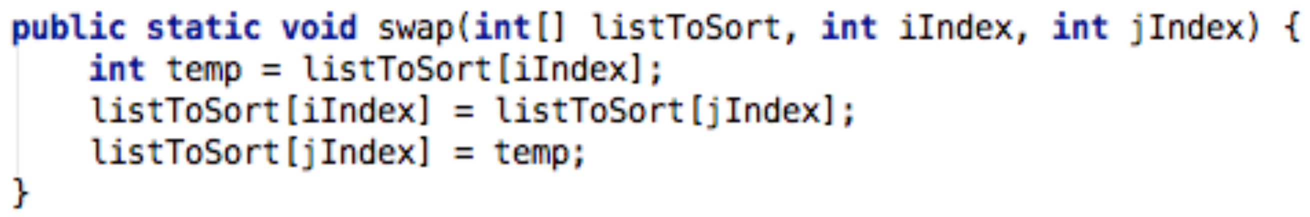
LET'S LOOK AT WHAT THE CODE FOR SELECTION SORT LOOKS LIKE

BUT FIRST... HELPER METHODS

PRINT THE LIST, SO WE CAN SEE HOW THE SORT HAPPENS



```java
public static void print(int[] listToSort) {
    for (int el : listToSort) {
        System.out.print(el + ",");
    }
    System.out.println();
}
```

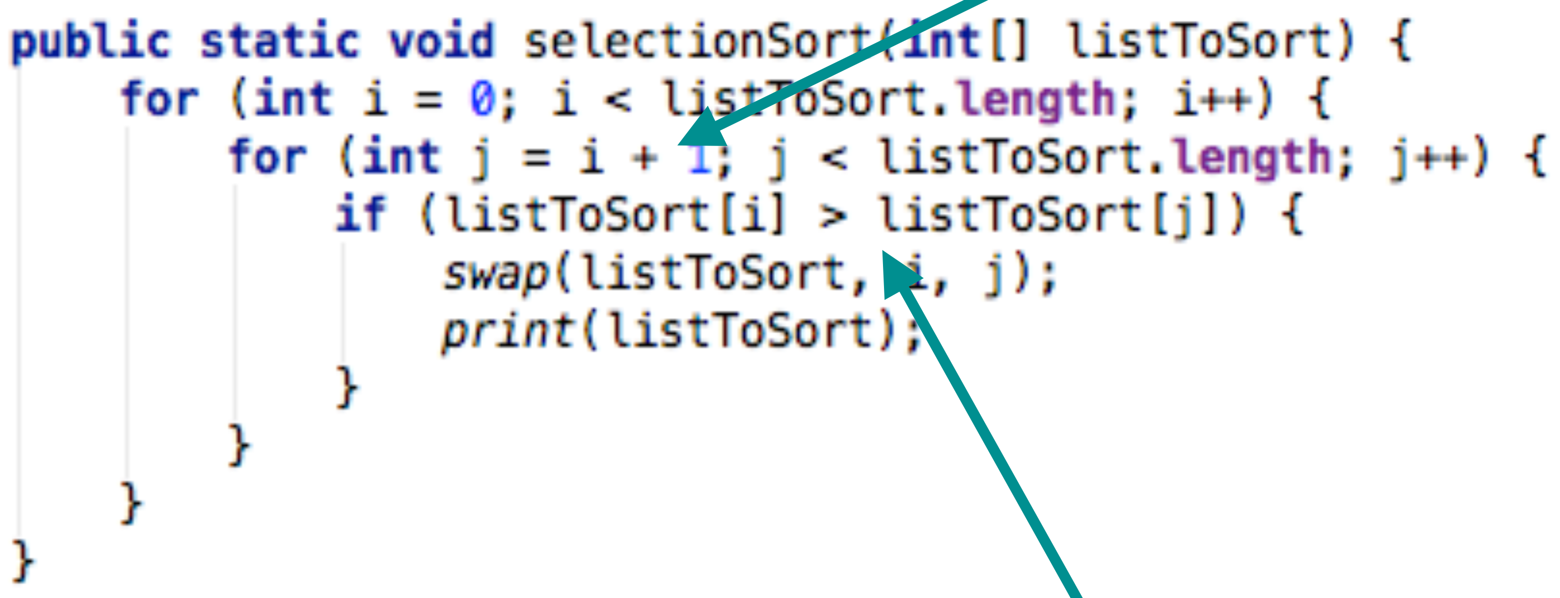


```java
public static void swap(int[] listToSort, int iIndex, int jIndex) {
    int temp = listToSort[iIndex];
    listToSort[iIndex] = listToSort[jIndex];
    listToSort[jIndex] = temp;
}
```

SWAP TWO ELEMENTS IN THE LIST

SELECTION SORT - CODE

NOTE THAT THE SECOND LOOP STARTS FROM THE ELEMENT AFTER `i`



```java
public static void selectionSort(int[] listToSort) {
    for (int i = 0; i < listToSort.length; i++) {
        for (int j = i + 1; j < listToSort.length; j++) {
            if (listToSort[i] > listToSort[j]) {
                swap(listToSort, i, j);
                print(listToSort);
            }
        }
    }
}
```

IF THE ELEMENT AT INDEX AT `i` IS GREATER THAN THE ELEMENT AT INDEX `j`, SWAP THE ELEMENTS

FOR EACH ELEMENT THE ENTIRE LIST IS CHECKED TO FIND THE SMALLEST ELEMENT

SO IN THE WORST CASE "N" ELEMENTS ARE CHECKED FOR EVERY SELECTED ELEMENT

THE COMPLEXITY OF SELECTION SORT IS $O(N^2)$

IT IS NOT A STABLE SORT - ENTITIES WHICH ARE EQUAL MIGHT BE RE-ARRANGED

IT TAKES $O(1)$ EXTRA SPACE, IT SORTS IN PLACE

IT MAKES $O(N^2)$ COMPARISONS AND $O(N)$ SWAPS