1) Selection Sort: Time complexity - Big O(n^2)

Min element comes to the first and we iterate leaving the sorted elements at first positions

index =	0	1	2	3	4

13 9 0 1 3 → 0 was the min element, 0 went to 13th index and 13 went to 0th index

$\boxed{0}$ 9 13 1 3 → 1 ⇄ 9

$\boxed{0 \ 1}$ 13 9 3 → 3 ⇄ 13

$\boxed{0 \ 1 \ 3}$ 9 13 → No need to swap as 9 is at the min-index.

Pseudo code:

```
for (i=0; i<n-1; i+=1)        // i will run from 0 to 3 because when j=i, j will compare the second last with last element
                              //   → There's no need to compare last element with the last element
    min-index = i             // min-index will be the first i-th value i.e, if i=0, min-index=0
                              //                                              i=1,     "     = 1
    for (j=i; j<n; j+=1)      // j will run from 0 to 4
        if (arr[j] < arr[min-index])
            min-index = j     // j will check for the min.element from i to j<n and update the min-index value
    temp = arr[i]
    arr[i] = arr[min-index]
    arr[min-index] = temp     // Ex: temp = arr[0] = 13
                              //     arr[0] = arr[2] ⇒ [0 9 0 1 3]
                              //     arr[2] = temp ⇒ [0 9 13 1 3]
return arr                    // → return the updated array
```

2) Bubble sort - Pushes the max element at the last position using adjacent swaps

index =	0	1	2	3	4

$\boxed{10 \ 0}$ 3 9 2 → 10 ⇄ 0

0 $\boxed{10 \ 3}$ 9 2 ⇒ 10 ⇄ 3

0 3 $\boxed{10 \ 9}$ 2 → 10 ⇄ 9

0 3 9 $\boxed{10 \ 2}$ → 10 ⇄ 2 ⇒ 10 got shifted to the last position

$\boxed{0 \ 3}$ 9 2 10 → No swap

0 $\boxed{3 \ 2}$ 9 10 → 3 ⇄ 2

0 2 $\boxed{3 \ 9}$ 10 → No swap

0 2 3 $\boxed{9 \ 10}$ → No swap

0 2 3 9 10 ⇒ Elements are Sorted

Pseudocode:

```
for (i=n; i>1; i-=1)
    for (j=0; j<i-1; j+=1)
        if (arr[j] > arr[j+1])
            int temp = arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = temp
return arr;
```

Worst complexity case:

Outer loop:	Inner loop:
n	$0 - n-1$
$n-1$	$0 - n-2$
$n-2$	$0 - n-3$
$\vdots$	$\vdots$
$n > 1 \Rightarrow 2$	$j < i-1 \Rightarrow j < 2-1 \Rightarrow j < 1$

$$n + n-1 + n-2 + \cdots + 2$$

$$\Rightarrow n(1 - 1 - 2 + \cdots + 2)$$

$$\Rightarrow \frac{n(n+1)}{2} = \frac{n^2+\cancel{n}}{\cancel{2}}$$

Big O $(n^2) \Rightarrow$ Upper limit bound

Optimization:

```
for (i=n ; i>1 ; i-=1)
    int did_swap = 0
    for (j=0 ; j<i-1 ; j+=1)
        if (arr[j] > arr[j+1])
            int temp = arr[j]
            arr[j] = arr[j+1]
            arr[j+1] = temp
            did_swap = 1        // if swap happened
    if (did_swap == 0)
        break;                  // else break from the loop if didn't
                                   happen
return arr;
```

$\therefore$ Best case is $O(n)$.

3) Insertion sort

index =	0	1	2	3	4

5 3 9 2 1 $\Rightarrow$ 3 < 5 swap

3 5 9 2 1 $\Rightarrow$ 9 < 5 No , 5 < 3

3 5 2 9 1 $\Rightarrow$ 9 < 2 No , 2 < 5 swap . 2 < 3 swap

3 2 5 9 1 $\Rightarrow$

2 3 5 9 1 $\Rightarrow$

2 3 5 1 9 $\Rightarrow$ 9 < 1 No , 1 < 5 swap , 1 < 3 swap , 1 < 2 swap

2 3 1 5 9 $\Rightarrow$

2 1 3 5 9 $\Rightarrow$

1 2 3 5 9

Pseudocode :

worst case complexity : Big O (n^2)

Best case : Big O(n)

```
for (i = 1 ; i<n ; i+=1)
    for (j = i ; j>0 and arr[j-1] > arr[j] ; j -= 1)

        int temp = arr[j]
        arr[j] = arr[j-1]
        arr[j-1] = temp;
return arr;
```