Bubble sort.

5 | 4 | 3 | 2 | 1
0 1 2 3 4

sort. → 1 | 2 | 3 | 4 | 5
0 1 2 3 4

:- Swap the adjacents, if needed till we get all the array sorted.

→ i/p 5 | 4 | 3 | 2 | 1
0 1 2 3 4

(1) I Largest element at it's position.

(1) 5 | 4 | 3 | 2 | 1
(2) 4 | 5 | 3 | 2 | 1
(3) 4 | 3 | 5 | 2 | 1
(4) 4 | 3 | 2 | 5 | 1

4 comparisons.

4 | 3 | 2 | 1 | 5

(2) second Largest at it's pos.

(1) 4 | 3 | 2 | 1 | 5
(2) 3 | 4 | 2 | 1 | 5
(3) 3 | 2 | 4 | 1 | 5

3 Comparisons.

3 | 2 | 1 | 4 | 5

(3) Third Largest at it's pos.

① 3|2|1|4|5 2 comparisons.

② 2|3|1|4|5

$\boxed{2|1|3|4|5}$

④ fourth Largest at its pos.

① 2|1|3|4|5 1 Comparison

$\boxed{1|2|3|4|5}$ → Array is sorted.

Observations.

N=5

Iterations.	Comparisons.	Generalize
I.	4	1 → (n-1) Comp.
II.	3	2 → (n-2) Comp.
III.	2	⋮
IV.	1.	(n-1) → 1 comp.

$S_n = 1 + 2 + 3 + \text{-----} + (n-2) + (n-1)$

$$S_n = \frac{n(n-1)}{2} = \frac{n^2-n}{2}$$

$$O(n) = O\left(\frac{n^2}{2} - \frac{n}{2}\right)$$

$T.C = O(n^2)$

$S.C = O(1)$

① outer loop (n-1) times.
② inner loop (n-i-1) times.

code:

```
void Bubblesort(vector<int>&nums)
{
    int n = nums.size();
    for(int i=0; i<n-1; i++)
    {
        for(int j=0; j<n-i-1; j++)
        {
            if(nums[j] > nums[j+1])
            {
                swap(nums[j], nums[j+1])
            }
        }
    }
}.
```

selection sort.

what if, I select the minimum element and put it @ Right position.

I/P Array: 44 | 33 | 55 | 22 | 11.

① 44 | 33 | 55 | 22 | 11 → smallest element ↳ 11.

11 | 33 | 55 | 22 | 44.

② 11 | 33 | 55 | 22 | 44 → smallest element ↳ 22.

11 | 22 | 55 | 33 | 44.

③ 11 | 22 | 55 | 33 | 44 – smallest element ↳ 33.

11 | 22 | 33 | 55 | 44.

④ 11 | 22 | 33 | 55 | 44 – smallest element ↳ 44.

11 | 22 | 33 | 44 | 55

↳ Array is sorted.

obs: for ith iteration, pick smallest element from i to (n-1) index.
&
swap it with ith element.

① Outer loop : i→ (0 → <n-1) { i∈ [0, n-1]

② inner loop : j→ (0 →<n) { j∈ [0 , n).

Code

```
void selectionSort(vector<int> &nums) {
    int n = nums.size();
    for( int i=0; i<n-1; i++ ) {

        int minIndex = i; // to store min val
        for( int j= i+1; j<n; j++ ) {
            if (nums[j] <nums[minIndex])
                minIndex = j;
        }
        swap (nums[i], nums[midIndex]);

    }

},
```

Time complexity = $O(n^2)$

Space complexity :- $O(1)$.

1 Insertion sort.

just insert element at their right position and move other elements.

i/p :- 5 | 4 | 3 | 2 | 1.

⟷ compare

① 5 | 4 | 3 | 2 | 1   Key = 4
0 1 2 3 4.
↑ ignore.

4 | 5 | 3 | 2 | 1

② insert
3 | 4 | 5 | 3 | 2 | 1   Key = 3
move.

③ 3 | 4 | 5 | 2 | 1   Key = 2
2

④ 2 | 3 | 4 | 5 | 1   Key = 1

O/P :- 1 | 2 | 3 | 4 | 5.

① outer loop → 1 → n−1 } i ∈ ~~(1, n)~~ [1, n).

② inner loop.

```
j = i-1.
while( j<=0 && nums[j] > key) {
    nums[j+1] = nums[j]
    j--;
}
nums[j+1] = key.
```

I. i = 1 → 1 N = 5

II. i = 2 → 2

III. i = 3 → 4

IV. i = 4 → 4

(n - 1)

$$\frac{n(n-1)}{2} = \frac{n^2}{2} - \frac{n}{2}$$

Time complexity → O(n²)

Space complexity → O(1).

Code

```
void insertionSort(vector<int>&nums){
    int n = nums.size();
    for(int i=1; i<n; i++){
        int key = nums[i];
        int j = i-1;
        while(j>=0 && nums[j] > key)
        {
            nums[j+1] = nums[j];
            j--;
        }
        nums[j+1] = key;
    }.
```

▣ custom comparator.

C++ → STL.

↳ sort () (builtin functions).

in vector → sort (v.begin(), v.end())

in array → sort (a, a+n) .

↓

base address.

:- In C++, a custom comparator refers to a user deffined function object user for comparing elements in data structures like vector, sets, maps & priority queues.

These data structures often require a way to compare elements to maintain their order or uniqueness.

→ By default these data structures use the less-than ('<') operator for comparison.

However in some cases, you might want to sort or order elements based on criteria other than the default less than comparison.

This is where custom comparators come in handy.

```
#include<iostream>
# include <vector>
# include < algorithm> → Header file for
Using namespace std ;            builtin functions.
// Custom comparator function for sorting in descending.
bool descendingorder(int a, int b){

        return a>b;

};

int main(){

  vector<int> numbers = {5, 2, 9, 1, 7};

 // Normal sorting in Ascending order
  sort(numbers.begin(), numbers.end());

 // sorting now in descending order using
 //   custom comparator.

Sort(numbers.begin(), numbers.end(), descendingorder);

 // output sorted numbers

    for (int num : numbers){
          cout<<num<<" ";
    }
 return 0;

}
```

it has always bool return type.