Bubble Sort compares all the element one by one and sort them based on their values. If the given array has to be sorted in ascending order, then bubble sort will start by comparing the first element of the array with the second element, if the first element is greater than the second element, it will swap both the elements, and then move on to compare the second and the third element, and so on. This way, after every pass, the largest element reaches to the end of the array.

The time complexity of Bubble Sort in Worst and Average cases is O(n^2), whereas in Best case it's O(n).

In [27]:
```
def bubble_sort(data):
    count = 0
    for i in range(len(data)-1):
        print(data)
        for j in range(len(data)-i-1):
# In every iteration of the outer loop, one number gets sorted.
# So the inner loop will run only for the unsorted part
            count += 1
            if data[j] > data[j+1]:
# If two adjacent elements are in the wrong order, they get swapped.
                data[j], data[j+1] = data[j+1], data[j]

    return (f'\nSorted array: {data} \nNumber of comparisons = {count}')
```

In [28]:
```
new_list = [19, 13, 6, 2, 18, 8]

print(bubble_sort(new_list))
```

```
[19, 13, 6, 2, 18, 8]
[13, 6, 2, 18, 8, 19]
[6, 2, 13, 8, 18, 19]
[2, 6, 8, 13, 18, 19]
[2, 6, 8, 13, 18, 19]

Sorted array: [2, 6, 8, 13, 18, 19]
Number of comparisons = 15
```

We can optimize the bubble sorting implementation, by adding a boolean variable to keep track of the number of swaps performed during an iteration. If somewhere through the loops the array becomes completely sorted, the function won't have to do unnecessary comparisons thanks to this boolean variable:

In [29]:

```
def opt_bubble_sort(data):
    count = 0
    for i in range(len(data) - 1):
        swap = False
        print(data)
        for j in range(len(data) - i - 1):
            count += 1
            if data[j] > data[j+1]:
                data[j], data[j+1] = data[j+1], data[j]
                swap = True
        if swap==False:
            return (f'\nSorted array:{data} \nNumber of comparisons = {cou
    return (f'\nSorted array:{data} \nNumber of comparisons = {count}')
```

In [30]:

```
new_data = [5,9,3,10,45,2,0]
print(opt_bubble_sort(new_data))
```

```
[5, 9, 3, 10, 45, 2, 0]
[5, 3, 9, 10, 2, 0, 45]
[3, 5, 9, 2, 0, 10, 45]
[3, 5, 2, 0, 9, 10, 45]
[3, 2, 0, 5, 9, 10, 45]
[2, 0, 3, 5, 9, 10, 45]

Sorted array:[0, 2, 3, 5, 9, 10, 45]
Number of comparisons = 21
```

In [31]:

```
nearly_sorted = [5,6,7,8,9]
print(opt_bubble_sort(nearly_sorted))
```

```
[5, 6, 7, 8, 9]

Sorted array:[5, 6, 7, 8, 9]
Number of comparisons = 4
```