The insertion sort algorithm concept can be described similarly to the way we sort a deck of cards. We assume that the first card is already sorted, and the rest isn't. If the unsorted card we selected next is greater than the first card, it is placed on the right bunch, otherwise, it goes to the left bunch.

This is a more straightforward and efficient algorithm than the bubble sort algorithm, but it is more beneficial for nearly-sorted lists/arrays or ones with fewer items, because it uses nested loops for sorting. Its time complexity is O(n^2) in Worst case.

To implement it, the first iteration will define the first item as the only one that is sorted and in the correct position. Then, it loops through the rest of the items and inserts them in their correct position, with respect to the already sorted part of the array.

In [1]:

```python
def insertion_sort(array):
    count = 0
    size = len(array)
    for i in range(1, size):
        print(array)
        last_sorted_position = array[i-1]  # A)
        count += 1
        if array[i] < last_sorted_position: # B)
            temp = array[i] # C)
            for j in range(i-1,-1,-1):  # D)
                count += 1
                if temp < array[j]: # E)
                    if j == 0: # F)
                        array[j+1] = array[j]
                        array[j] = temp
                    else:
                        array[j+1] = array[j]
                else:
                    array[j+1] = temp # G
                    break
    return (f'\nSorted array: {array} \nNumber of comparisons = {count}')
```

In [2]:

```python
array = [5,9,3,10,45,2,0]
print(insertion_sort(array))
```

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

Sorted array: [0, 2, 3, 5, 9, 10, 45]
Number of comparisons = 19
```

In [3]:

```
reverse_sorted_array = [9,8,7,6,5,4,3,2,1,0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10]
print(insertion_sort(reverse_sorted_array))
```

```
[9, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[8, 9, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[7, 8, 9, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[6, 7, 8, 9, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[5, 6, 7, 8, 9, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[4, 5, 6, 7, 8, 9, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[3, 4, 5, 6, 7, 8, 9, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[2, 3, 4, 5, 6, 7, 8, 9, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -1, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[-1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -2, -3, -4, -5, -6, -7, -8, -9, -10]
[-2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -3, -4, -5, -6, -7, -8, -9, -10]
[-3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -4, -5, -6, -7, -8, -9, -10]
[-4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -5, -6, -7, -8, -9, -10]
[-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -6, -7, -8, -9, -10]
[-6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -7, -8, -9, -10]
[-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -8, -9, -10]
[-8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -9, -10]
[-9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, -10]

Sorted array: [-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Number of comparisons = 209
```

Here's a summary of the function above:

A) We start by storing the last item which is considered already sorted;

B) Then, we check if the current item is smaller than the last sorted item;

C) If so, we store the curent item in a temporary variable;

D) After that, we loop backwards through the sorted part of the array to check where the current element can fit;

E) Within the sorted part of the array, every item that is greater than the current one gets shifted one position to the right, in order to make room for the current element;

F) If we reach the beginning of the array in the process, we shift all items to the right and assign the current element to the 0th position. Otherwise, we just keep shifting;

G) Once we find an item that is smaller than the current element, then we found the right position to insert our current item in, so we assign it to that position and break out of the loop.