Quick sort is also a "divide and conquer" algorithm, like Merge sort. It uses pivoting technique to break the main list into smaller ones until it gets sorted.

It requires to chose a pivot item. Then, it places all items smaller than the pivot item to its left and all the larger ones to its right. So, the original array is split in the pivot position and we get an array on the left and another on the right, which will follow the same pivoting technique, until both arrays are sorted. After each pass, the pivot element occupies its correct position in the array.

The time complexity in Best and Average cases is of O(n log n), whereas in Worst case it jumps up to O(n^2). The space complexity is O(log n).

In this implementation, we will take the last element as the pivot item:

In [1]:
```
count = 0

def division(array, left, right):
    smaller_index = left - 1
    pivot = array[right]
    for i in range(left, right):
        global count
        count += 1
        if array[i] < pivot:
            smaller_index += 1
            array[smaller_index], array[i] = array[i], array[smaller_index
    array[smaller_index + 1], array[right] = array[right], array[smaller_i
    print(array)
    return (smaller_index + 1)

def quick_sort(array, left, right):
    if left < right:
        dividing_index = division(array, left, right)
        print(dividing_index)
        quick_sort(array,left,dividing_index - 1)
        quick_sort(array,dividing_index + 1,right)
```

In [2]:
```
new_array = [5,9,3,10,45,2,0]
quick_sort(new_array, 0, (len(new_array)-1))
print(f'\nSorted array: {new_array}\nNumber of comparisons = {count}')
```

```
[0, 9, 3, 10, 45, 2, 5]
0
[0, 3, 2, 5, 45, 9, 10]
3
[0, 2, 3, 5, 45, 9, 10]
1
[0, 2, 3, 5, 9, 10, 45]
5

Sorted array: [0, 2, 3, 5, 9, 10, 45]
Number of comparisons = 14
```

In [3]:

```
sorted_array = [5,6,7,8,9]
quick_sort(sorted_array, 0, (len(sorted_array)-1))
print(f'\nSorted array: {sorted_array}\nNumber of comparisons = {count}')
```

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

Sorted array: [5, 6, 7, 8, 9]
Number of comparisons = 24
```

In [4]:

```
reverse_sorted_array = [9,8,7,6,5,4,3,2,1,0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10
quick_sort(reverse_sorted_array, 0, len(reverse_sorted_array) - 1)
print(f'\nSorted array: {reverse_sorted_array}\nNumber of comparisons = {c
```

```
[-10, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, 9]
0
[-10, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, 9]
19
[-10, -9, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, 8, 9]
1
[-10, -9, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, 8, 9]
18
[-10, -9, -8, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, 7, 8, 9]
2
[-10, -9, -8, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, 7, 8, 9]
17
[-10, -9, -8, -7, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, 6, 7, 8, 9]
3
[-10, -9, -8, -7, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, 6, 7, 8, 9]
16
[-10, -9, -8, -7, -6, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, 5, 6, 7, 8, 9]
4
[-10, -9, -8, -7, -6, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, 5, 6, 7, 8, 9]
15
[-10, -9, -8, -7, -6, -5, 3, 2, 1, 0, -1, -2, -3, -4, 4, 5, 6, 7, 8, 9]
5
[-10, -9, -8, -7, -6, -5, 3, 2, 1, 0, -1, -2, -3, -4, 4, 5, 6, 7, 8, 9]
14
[-10, -9, -8, -7, -6, -5, -4, 2, 1, 0, -1, -2, -3, 3, 4, 5, 6, 7, 8, 9]
6
[-10, -9, -8, -7, -6, -5, -4, 2, 1, 0, -1, -2, -3, 3, 4, 5, 6, 7, 8, 9]
13
[-10, -9, -8, -7, -6, -5, -4, -3, 1, 0, -1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
7
[-10, -9, -8, -7, -6, -5, -4, -3, 1, 0, -1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
12
[-10, -9, -8, -7, -6, -5, -4, -3, -2, 0, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
8
[-10, -9, -8, -7, -6, -5, -4, -3, -2, 0, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
11
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
9

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 = 214
```

As we defined our pivot item to be the last one of a given array, we can observe the time complexity increasing when the pivot item is either the biggest or the smallest item of said array. This is because of the number of comparisons performed.

However, if we define our pivot item to be one from the middle of the array, the number of comparisons will decrease.

In [5]:

```
count = 0

def partition(array, left, right):
    smaller_index = left - 1
    pivot = array[(right-left)//2] # changed pivot to be an item in the mi
    for i in range(left, right):
        global count
        count += 1
        if array[i] < pivot:
            smaller_index += 1
            array[smaller_index], array[i] = array[i], array[smaller_index
    array[smaller_index + 1], array[right] = array[right], array[smaller_i
    print(array)
    return (smaller_index + 1)

def quickest_sort(array, left, right):
    if left < right:
        dividing_index = division(array, left, right)
        print(dividing_index)
        quick_sort(array,left,dividing_index - 1)
        quick_sort(array,dividing_index + 1,right)
```

In [6]:

```
reverse_sorted_array = [9,8,7,6,5,4,3,2,1,0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10
quickest_sort(reverse_sorted_array, 0, len(reverse_sorted_array) - 1)
print(f'\nSorted array: {reverse_sorted_array}\nNumber of comparisons = {c
```

```
[-10, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, 9]
0
[-10, 8, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, -9, 9]
19
[-10, -9, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, 8, 9]
1
[-10, -9, 7, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, -8, 8, 9]
18
[-10, -9, -8, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, 7, 8, 9]
2
[-10, -9, -8, 6, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, -7, 7, 8, 9]
17
[-10, -9, -8, -7, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, 6, 7, 8, 9]
3
[-10, -9, -8, -7, 5, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, -6, 6, 7, 8, 9]
16
[-10, -9, -8, -7, -6, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, 5, 6, 7, 8, 9]
4
[-10, -9, -8, -7, -6, 4, 3, 2, 1, 0, -1, -2, -3, -4, -5, 5, 6, 7, 8, 9]
15
[-10, -9, -8, -7, -6, -5, 3, 2, 1, 0, -1, -2, -3, -4, 4, 5, 6, 7, 8, 9]
5
[-10, -9, -8, -7, -6, -5, 3, 2, 1, 0, -1, -2, -3, -4, 4, 5, 6, 7, 8, 9]
14
[-10, -9, -8, -7, -6, -5, -4, 2, 1, 0, -1, -2, -3, 3, 4, 5, 6, 7, 8, 9]
6
[-10, -9, -8, -7, -6, -5, -4, 2, 1, 0, -1, -2, -3, 3, 4, 5, 6, 7, 8, 9]
13
[-10, -9, -8, -7, -6, -5, -4, -3, 1, 0, -1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
7
[-10, -9, -8, -7, -6, -5, -4, -3, 1, 0, -1, -2, 2, 3, 4, 5, 6, 7, 8, 9]
12
[-10, -9, -8, -7, -6, -5, -4, -3, -2, 0, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
8
[-10, -9, -8, -7, -6, -5, -4, -3, -2, 0, -1, 1, 2, 3, 4, 5, 6, 7, 8, 9]
11
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
9

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 = 190
```

It is possible to verify that the number of comparisions performed on the reverse_sorted_array between functions has decreased by 24.