Merge Sort implements a "divide and conquer" method, so it will break an array into two roughly equal parts. If the array has an odd number of items, one of those "halves" is larger than the other by one item. The sub-arrays are divided into halves over and over again, until we end up with arrays that have only one item each. Then, we combine two of these "one-item" arrays into a pair, sorting them in the process. These sorted pairs merge into "four-item" arrays and so on, until you end up with the initial array, but sorted.

Its main advantage is the reliable runtime of the algorithm and its efficiency when sorting large arrays. However, the additional memory that Merge Sort uses to store the temporary copies of arrays before merging them is its main disadvantage.

The time complexity for this operations is of O(n log n), and space complexity is of O(n).

In [2]:

```
count = 0


def merge(left, right):
    l = len(left)
    r = len(right)
    left_index = 0
    right_index = 0
    sorted_array = []
    while (left_index < l and right_index < r):
        global count
        count += 1
        if left[left_index] < right[right_index]:
            sorted_array.append(left[left_index])
            left_index += 1
        else:
            sorted_array.append(right[right_index])
            right_index += 1
    print(sorted_array + left[left_index:] + right[right_index:])
    return sorted_array + left[left_index:] + right[right_index:]

def merge_sort(array):
    if len(array) == 1:
        return array
    else:
        mid = len(array)//2
        left_array = array[:mid]
        right_array = array[mid:]
        print(f'Left : {left_array}')
        print(f'Right : {right_array}\n')
        return merge(merge_sort(left_array),merge_sort(right_array))
```

In [3]:

```
array = [5,9,3,10,45,2,0]
print(merge_sort(array))
print(f'Number of comparisons: {count}')
```

```
Left : [5, 9, 3]
Right : [10, 45, 2, 0]

Left : [5]
Right : [9, 3]

Left : [9]
Right : [3]

[3, 9]
[3, 5, 9]
Left : [10, 45]
Right : [2, 0]

Left : [10]
Right : [45]

[10, 45]
Left : [2]
Right : [0]

[0, 2]
[0, 2, 10, 45]
[0, 2, 3, 5, 9, 10, 45]
[0, 2, 3, 5, 9, 10, 45]
Number of comparisons: 12
```

In [4]:

```
sorted_array = [5,6,7,8,9]
print(merge_sort(sorted_array))
print(f'Number of comparisons: {count}')
```

```
Left : [5, 6]
Right : [7, 8, 9]

Left : [5]
Right : [6]

[5, 6]
Left : [7]
Right : [8, 9]

Left : [8]
Right : [9]

[8, 9]
[7, 8, 9]
[5, 6, 7, 8, 9]
[5, 6, 7, 8, 9]
Number of comparisons: 17
```

In [5]:

```
reverse_sorted_array = [9,8,7,6,5,4,3,2,1,0,-1,-2,-3,-4,-5,-6,-7,-8,-9,-10
print(merge_sort(reverse_sorted_array))
print(f'Number of comparisons: {count}')
```

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

Left : [9, 8, 7, 6, 5]
Right : [4, 3, 2, 1, 0]
```

```
Left : [9, 8]
Right : [7, 6, 5]

Left : [9]
Right : [8]

[8, 9]
Left : [7]
Right : [6, 5]

Left : [6]
Right : [5]

[5, 6]
[5, 6, 7]
[5, 6, 7, 8, 9]
Left : [4, 3]
Right : [2, 1, 0]

Left : [4]
Right : [3]

[3, 4]
Left : [2]
Right : [1, 0]

Left : [1]
Right : [0]

[0, 1]
[0, 1, 2]
[0, 1, 2, 3, 4]
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Left : [-1, -2, -3, -4, -5]
Right : [-6, -7, -8, -9, -10]

Left : [-1, -2]
Right : [-3, -4, -5]

Left : [-1]
Right : [-2]

[-2, -1]
Left : [-3]
Right : [-4, -5]

Left : [-4]
Right : [-5]

[-5, -4]
[-5, -4, -3]
[-5, -4, -3, -2, -1]
Left : [-6, -7]
Right : [-8, -9, -10]

Left : [-6]
Right : [-7]

[-7, -6]
Left : [-8]
```

```
Right : [-9, -10]

Left : [-9]
Right : [-10]

[-10, -9]
[-10, -9, -8]
[-10, -9, -8, -7, -6]
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1]
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
[-10, -9, -8, -7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
Number of comparisons: 65
```