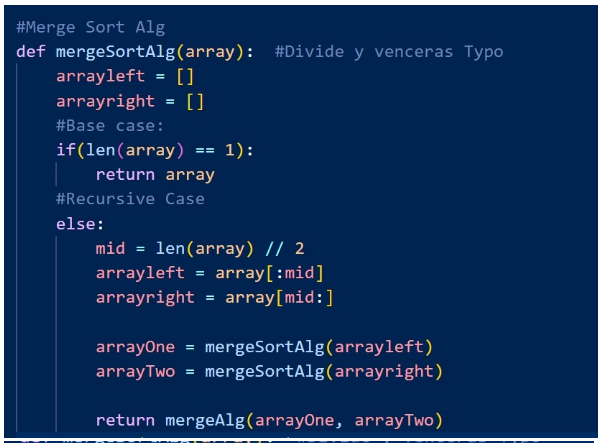
Merge Sort

Recursion

Merge sort is defined as a sorting algorithm that works by dividing an array into smaller subarrays, sorting each subarray, and then merging the sorted subarrays back together to form the final sorted array.



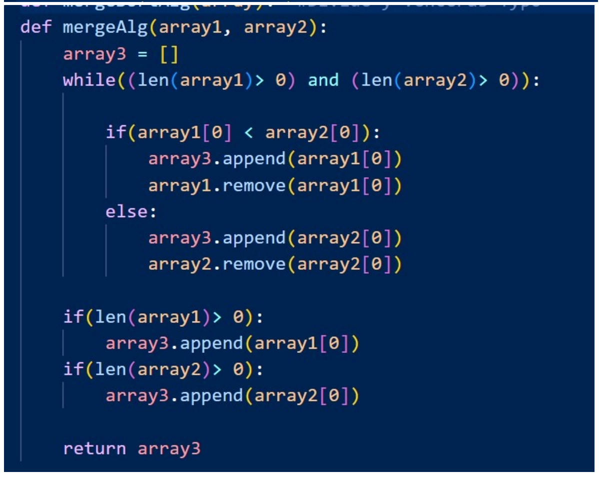
```python
#Merge Sort Alg
def mergeSortAlg(array):  #Divide y venceras Typo
    arrayleft = []
    arrayright = []
    #Base case:
    if(len(array) == 1):
        return array
    #Recursive Case
    else:
        mid = len(array) // 2
        arrayleft = array[:mid]
        arrayright = array[mid:]

        arrayOne = mergeSortAlg(arrayleft)
        arrayTwo = mergeSortAlg(arrayright)

        return mergeAlg(arrayOne, arrayTwo)
```

The first function splits the array in two and keeps calling itself (recursively) till the base case, where the array length == 1

Right after, the sorting function is call for both of the sub arrays.



```python
def mergeAlg(array1, array2):
    array3 = []
    while((len(array1)> 0) and (len(array2)> 0)):

        if(array1[0] < array2[0]):
            array3.append(array1[0])
            array1.remove(array1[0])
        else:
            array3.append(array2[0])
            array2.remove(array2[0])

    if(len(array1)> 0):
        array3.append(array1[0])
    if(len(array2)> 0):
        array3.append(array2[0])

    return array3
```

While both arrays contain at least one element, we'll compare them.

Depending on which element is minor, we'll add it to the new array and remove it from its original "home"

Finally, when there is an array that still contains an element, we'll add it and return the new all sorted array.

Time Complexity: O(N log(N))

Great for sorting large data sets, which also means not so optimal with very small datasets.
A drawback is the space complexity, considering the additional memory used to store the merged sub arrays