Birleştirme Sıralaması

→ Merge Sort(birleştirme sıralaması), bir dizi elemanını sıralamak için "böl ve fethet" (divide and conquer) yaklaşımını kullanan bir sıralama algoritmasıdır. Diziyi sürekli olarak ikiye böler, ardından her bir parçayı sıralar ve son olarak birleştirir.

1- İlk Adım: Sıralanacak olan diziyi ortadan ikiye böleriz. Bu işlem, dizinin uzunluğu 1 veya 0 olana kadar tekrarlanır.

2- İkinci Adım: İlk adımın sonunda, her bir alt dizi artık sıralı haldedir. Bu adımda, alt dizileri birleştirerek sıralı bir ana dizi oluştururuz. İki alt dizi sıralı olarak birleştirilirken, alt dizilerin elemanları karşılaştırılır ve küçük olan eleman ana diziye eklenir. Bu işlem, tüm alt diziler birleşene kadar tekrarlanır.

3- Son Adım: İkinci adımın sonunda, tüm alt diziler birleşmiş olur ve ana dizi sıralanmış hale gelir.

Zaman Karmaşıklığı: Merge Sort algoritmasının zaman karmaşıklığı O(n log n)'dir, burada "n" sıralanacak eleman sayısını temsil eder. Merge Sort, her bir bölme işlemi için O(log n) zaman karmaşıklığına sahiptir ve her bölmede tüm elemanları karşılaştırarak birleştirme işlemi yapar, bu da O(n) zaman karmaşıklığına sahiptir. Bu nedenle, bölme ve birleştirme işlemlerini topladığınızda, Merge Sort'un toplam zaman karmaşıklığı O(n log n) olur.

Not: Merge Sort'un zaman karmaşıklığı, diğer sıralama algoritmalarına göre daha iyidir, özellikle büyük veri kümesi için daha etkilidir. Ancak, Merge Sort, ek bir bellek gerektiren bir algoritmadır, çünkü sıralama işlemi için ek bir diziye ihtiyaç duyar. Bu nedenle, bellek kullanımı açısından bazı kısıtlamaları olabilir.

Örnek:

Dizi: [8, 4, 2, 9, 3, 1, 5, 7]

İlk Adım:

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

İkinci Adım:

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

Üçüncü Adım:

[8] [4] [2] [9] [3] [1] [5] [7]

Dördüncü Adım:

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

Beşinci Adım:

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

Son Adım:

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

Pseudocode Kod:

```
function mergeSort(arr)
    if length of arr is less than or equal to 1
        return arr

    // Step 1: Divide the array into two halves
    middle = length of arr divided by 2
    leftHalf = mergeSort(arr[0 to middle-1])
    rightHalf = mergeSort(arr[middle to end])

    // Step 2: Merge the two sorted halves
    sortedArr = merge(leftHalf, rightHalf)

    return sortedArr

function merge(leftHalf, rightHalf)
    mergedArr = empty array
    leftIndex = 0
    rightIndex = 0

    while leftIndex is less than length of leftHalf and rightIndex is less than length of rightHalf
        if leftHalf[leftIndex] <= rightHalf[rightIndex]
            add leftHalf[leftIndex] to mergedArr
            increment leftIndex
        else
```

```
        add rightHalf[rightIndex] to mergedArr
        increment rightIndex

// Add remaining elements from leftHalf (if any)
while leftIndex is less than length of leftHalf
    add leftHalf[leftIndex] to mergedArr
    increment leftIndex

// Add remaining elements from rightHalf (if any)
while rightIndex is less than length of rightHalf
    add rightHalf[rightIndex] to mergedArr
    increment rightIndex

return mergedArr
```

Python Kod:

```
def mergeSort(arr):
    if len(arr) <= 1:
        return arr

    # Step 1: Divide the array into two halves
    middle = len(arr) // 2
    leftHalf = mergeSort(arr[:middle])
    rightHalf = mergeSort(arr[middle:])

    # Step 2: Merge the two sorted halves
    sortedArr = merge(leftHalf, rightHalf)

    return sortedArr

def merge(leftHalf, rightHalf):
    mergedArr = []
    leftIndex = 0
    rightIndex = 0

    while leftIndex < len(leftHalf) and rightIndex < len(rightHalf):
        if leftHalf[leftIndex] <= rightHalf[rightIndex]:
            mergedArr.append(leftHalf[leftIndex])
            leftIndex += 1
        else:
```

```
            mergedArr.append(rightHalf[rightIndex])
            rightIndex += 1

    # Add remaining elements from leftHalf (if any)
    while leftIndex < len(leftHalf):
        mergedArr.append(leftHalf[leftIndex])
        leftIndex += 1

    # Add remaining elements from rightHalf (if any)
    while rightIndex < len(rightHalf):
        mergedArr.append(rightHalf[rightIndex])
        rightIndex += 1

    return mergedArr

arr = [8, 4, 2, 9, 3, 1, 5, 7]
sortedArr = mergeSort(arr)
print(sortedArr)
```