

Son Teslim tarihi: 11.11.2024

## Ödev #2

OĞUZHAN TOPALOĞLU  
Ç19052025 – Grup 1

*Bilgisayar Mühendisliği Bölümü,  
Elektrik-Elektronik Fakültesi,  
Yıldız Teknik Üniversitesi*



Istanbul, 2024

---

**Video Linki:** <https://youtu.be/-0aRKnWz5bM>

# İÇERİK

1. Problemin Çözümü
2. Karşılaşılan Sorunlar
3. Karmaşıklık Analizi
4. Ekran Görüntüleri

## 1. Problemin Çözümü

Çözümümde büyük bir diziyi daha küçük parçalara bölüp bu parçaları ayrı ayrı sıraladım ve ardından bu parçaları birleştirdim yani merge ettim. Bu çözüm, k-way merge sort olarak bilinen algoritmanın bir implementasyonudur.

Çözümümü psödö kod olarak aşağıya yazıyorum:

```
function mergeSort(arr, size, k, result):
```

```
    if size <= 1:
```

```
        result[0] = arr[0] if size == 1
```

```
        return
```

```
    mid = (size + k - 1) / k
```

```
    create arrays `subarrs` and `sizes` of length k
```

```
    for i = 0 to k - 1:
```

```
        start = i * mid
```

```
        end = min(start + mid, size)
```

```
        sizes[i] = end - start
```

```
        subarrs[i] = copy of arr[start to end]
```

```
        mergeSort(subarrs[i], sizes[i], k, subarrs[i])
```

```
    mergeKSortedArrays(subarrs, sizes, k, result, size)
```

## 2. Karşılaşılan Sorunlar

İmplementasyonum sırasında en çok `runBenchmark()` fonksiyonunu yazarken zorlandım. PDF’te özellikle aynı dizilerin  $k$  değerleri için kullanılması gerektiğini demesi biraz kafamı karıştırdı. Kaç tane döngü olacağını ve hangi döngünün içte hangisinin dışta olacağını pek anlamamıştım. Sonradan dış döngünün  $k=[2,10]$  ve iç döngünün `HOW_MANY` sefer olması gerektiğini anladım ve kodu yazdım.

Burada `HOW_MANY` sayısı 12 olarak seçtiğim 10’dan büyük bir sayıdır. Her  $(N,k)$  ikilisi için bir zaman metriği hesaplarken 12 kere test yapıp bunların ortalamasını alıyorum.

### 3. Karmaşıklık Analizi

mergeSort fonksiyonu verilen diziye her seferinde  $k$  alt diziye bölerek rekürsif olarak kendini çağırıyor. size boyutundaki diziye, her bölmede yaklaşık  $size / k$  uzunluğunda  $k$  alt parçaya ayırıyor. Her alt parça için mergeSort aynı işlemi uyguluyor, böylece her rekürsif adımda  $k$  alt dizi elde ediyoruz. Rekürsif adımların her birinin maliyeti  $O(N)$  ile ifade edilebilir, çünkü mergeKSortedArrays fonksiyonunu her bölme işleminin sonunda çağırıyoruz.

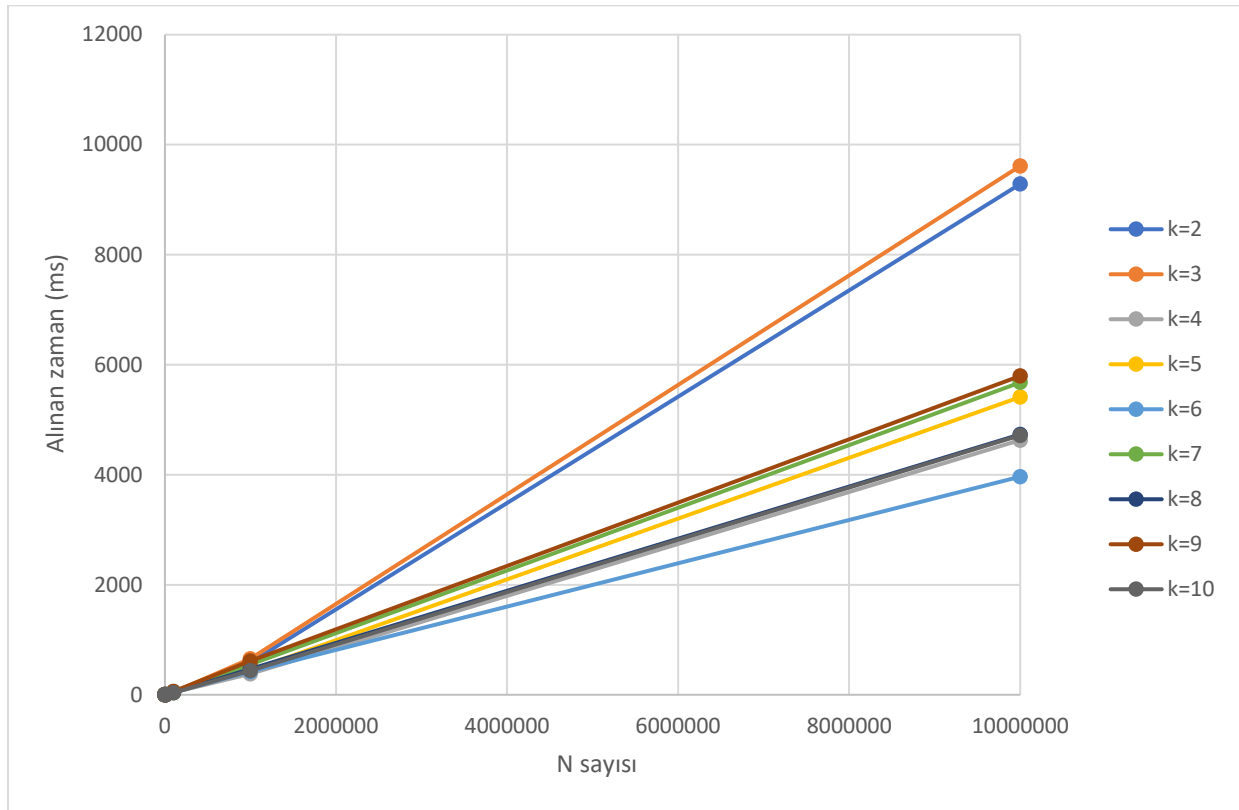
Her bir mergeSort çağrısında,  $k$  yeni alt diziye bölme işlemi yapıldığından, fonksiyonun zaman karmaşıklığı  $T(N) = k * T(N/k) + O(N)$  olarak yazılır. Master Teoremi ile bunu çözersek de  $T(N) = O(N \log_k N)$  buluruz, yani çözüm  $n \log n$  büyüklüğündedir.

mergeKSortedArrays için analiz (gerekiyor mu bilmiyorum):

Bu döngü toplamda  $totalSize = N$  eleman kadar sürer, çünkü mergeKSortedArrays fonksiyonu her bir index değeri için bir eleman ekleyerek result dizisini sıralı hale getirir. Bu yüzden  $index < totalSize$  döngüsünün kendisi  $O(N)$  adımda tamamlanacaktır.

for döngüsü her index adımında  $k$  sıralı dizi arasında en küçük elemanı bulmak için çalışır. Bu for döngüsünün maliyeti  $O(k)$ 'dir çünkü her index adımında  $k$  dizinin başındaki en küçük elemanı ararız. Bu  $O(k)$  maliyeti, index döngüsü boyunca  $N$  kez tekrarlandığı için toplam maliyet  $O(N * k)$  olur.

#### 4. Ekran Görüntüleri



The image shows a Visual Studio Code editor with three tabs: `C:\9052025.c`, `benchmark.txt`, and `Kontroller.txt`. The `benchmark.txt` tab is active, displaying a script that benchmarks a function for various values of `N` and `k`. The script consists of two main loops: one for `N=100` and `N=1000`, each with `k` values from 2 to 10. The output shows the average time taken for each configuration.

```
1 => N: 100, k: 2, avgtime: 0.083333
2 => N: 100, k: 3, avgtime: 0.000000
3 => N: 100, k: 4, avgtime: 0.000000
4 => N: 100, k: 5, avgtime: 0.083333
5 => N: 100, k: 6, avgtime: 0.000000
6 => N: 100, k: 7, avgtime: 0.083333
7 => N: 100, k: 8, avgtime: 0.000000
8 => N: 100, k: 9, avgtime: 0.083333
9 => N: 100, k: 10, avgtime: 0.000000
10
11 => N: 1000, k: 2, avgtime: 0.416667
12 => N: 1000, k: 3, avgtime: 0.416667
13 => N: 1000, k: 4, avgtime: 0.250000
14 => N: 1000, k: 5, avgtime: 0.416667
15 => N: 1000, k: 6, avgtime: 0.250000
16 => N: 1000, k: 7, avgtime: 0.416667
17 => N: 1000, k: 8, avgtime: 0.416667
18 => N: 1000, k: 9, avgtime: 0.500000
19 => N: 1000, k: 10, avgtime: 0.250000
20
21 => N: 10000, k: 2, avgtime: 4.750000
22 => N: 10000, k: 3, avgtime: 4.333333
23 => N: 10000, k: 4, avgtime: 3.500000
24 => N: 10000, k: 5, avgtime: 3.166667
25 => N: 10000, k: 6, avgtime: 4.666667
26 => N: 10000, k: 7, avgtime: 3.333333
```

The terminal at the bottom shows the command `C:\Users\oguzh\Desktop\Kodlama\zSpecial\university-stuff\algoritma_analizi\hw2>C:\9052025.exe` and the subsequent output of the program, which matches the content of `benchmark.txt`.