

Odev_4: Zaman Karmaşıklığı ve Çalışma Zamanı

Yusuf Üzeyir Kaya / 211307012

1. Aşağıdaki kavramları kısaca açıklayınız:

- **Big O** : Bir algoritmanın işlem adımlarının büyüklüğünü, en kötü durumda gösteren bir gösterimdir. Algoritmanın çalışma süresi (zaman karmaşıklığı) veya kullanılan bellek miktarı (alan karmaşıklığı) gibi performans ölçüleri için kullanılır.
- **Big Theta**: Bir algoritmanın işlem adımlarının büyüklüğünü, hem en iyi durumda hem de en kötü durumda gösteren bir gösterimdir. Bu gösterim, algoritmanın performansını daha kesin bir şekilde gösterir ve Big O'dan daha sık kullanılır.
- **Big Omega**: Bir algoritmanın işlem adımlarının büyüklüğünü, en iyi durumda gösteren bir gösterimdir. Bu gösterim, algoritmanın performansını en iyimser senaryoda gösterir.
- **Best Case**: Bir algoritmanın en iyi durumu, algoritmanın işlem adımlarının sayısının, girdinin boyutuna göre en az olduğu senaryodur.
- **Worst Case**: Bir algoritmanın en kötü durumu, algoritmanın işlem adımlarının sayısının, girdinin boyutuna göre en fazla olduğu senaryodur.
- **Expected Case**: Bir algoritmanın beklenen durumu, algoritmanın işlem adımlarının sayısının, girdinin boyutuna göre ortalama olarak ne kadar olacağıdır.
- **Time Complexity**: Bir algoritmanın çalışma süresinin, girdinin boyutuna göre nasıl değiştiğini ifade eden bir ölçüdür.
- **Space Complexity** : Bir algoritmanın çalışması için gereken bellek miktarının, girdinin boyutuna göre nasıl değiştiğini ifade eden bir ölçüdür.

2. Aşağıdaki kod parçacığı ne işe yarar ve big O zamanı nedir ?

```
int topN (int n)
{
    int sum = 0;
    while (n > 0) {
        sum += n % 10;
        n /= 10;
    }
    return sum;
}
```

Kod bloğu n sayısının rakamlarının toplamı için oluşturulmuş ama döngü içinde $n/10$ ifadesi de olması lazım ki her döngü de sayımızın son rakamını atalım. Big O zamanı da $O(\log n)$ dir.

3. Aşağıdaki kod parçacığı ne işe yarar ve çalışma süresi (runtime), zaman karmaşıklığı nedir?

```
int mod (int a, int b){
    if(b <= 0) {
        return -1;
    }
    int bol = a / b;
    return a - bol*b;
}
```

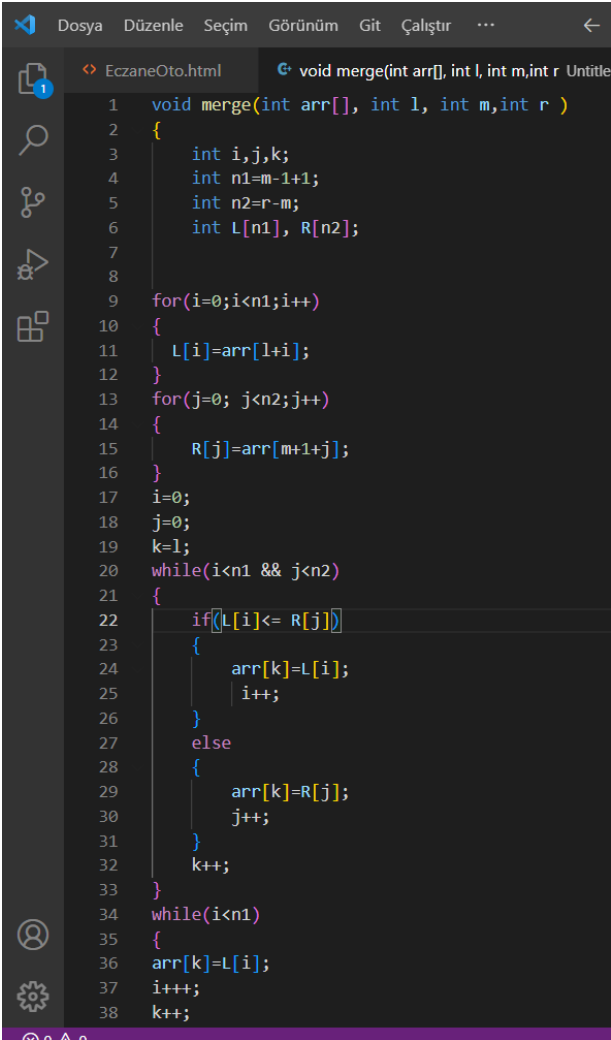
Kod bloğu a ve b değişkenlerini alır, eğer b 0 dan küçük ya da eşitse -1 döndürür. Eğer 0 dan büyükse a sayısının b ye göre modu hesaplanır. Zaman karmaşıklığı $O(1)$ 'dir. İşlem sayısı girdi boyutundan bağımsızdır.

4. Aşağıda verilen kod için zaman karmaşıklığını $T(n)$, çalışma süresini hesaplayınız ve gerekçenizi açıklayınız.

```
void bubble_sort(int arr[], int n)
{
    for (int i = 0; i < n - 1; i++)
    {
        for (int j = 0; j < n - i - 1; j++)
        {
            if (arr[j] > arr[j + 1])
            {
                int temp = arr[j];
                arr[j] = arr[j + 1];
                arr[j + 1] = temp;
            }
        }
    }
}
```

Koda bloğunun zaman karmaşıklığı $O(n^2)$ 'dir. Dizi n kez taranır ve en kötü durumda n^2 karşılaştırma yapılır.

5. Aşağıda verilen kod için zaman karmaşıklığını $T(n)$ ve çalışma zamanını hesaplayınız ve gerekçenizi açıklayınız



```
1 void merge(int arr[], int l, int m, int r)
2 {
3     int i, j, k;
4     int n1 = m - l + 1;
5     int n2 = r - m;
6     int L[n1], R[n2];
7
8
9     for (i = 0; i < n1; i++)
10    {
11        L[i] = arr[l + i];
12    }
13    for (j = 0; j < n2; j++)
14    {
15        R[j] = arr[m + 1 + j];
16    }
17    i = 0;
18    j = 0;
19    k = l;
20    while (i < n1 && j < n2)
21    {
22        if (L[i] <= R[j])
23        {
24            arr[k] = L[i];
25            i++;
26        }
27        else
28        {
29            arr[k] = R[j];
30            j++;
31        }
32        k++;
33    }
34    while (i < n1)
35    {
36        arr[k] = L[i];
37        i++;
38        k++;
39    }
40    while (j < n2)
41    {
42        arr[k] = R[j];
43        j++;
44        k++;
45    }
46 }
```

Kod bloğunun zaman karmaşıklığı $O(n \log n)$ 'dir. merge_sort, bir diziyi n elemanlı iki alt diziye bölerek başlar ve daha küçük parçalar halinde sıralar. Bu, her seviyede n elemanı işlemek için n adım gerektirir. Toplam seviye sayısı, n elemanının tamamının tek bir diziye birleştirildiği son seviyede $\log n$ adımına kadar sürer. Bu yüzden $n \cdot \log n$ karmaşıklığı oluşur.