# Veri Yapıları

## Algoritma Analizi, Algoritmalarda Karmaşıklık ve Zaman Karmaşıklığı

Dr. Sinan TUNCEL

## Algoritmaların Özellikleri

Algoritmalarda bulunması gerekli özellikleri;

- **Giriş :** Belirlenen veri kümesinden algoritma giriş değerleri alır.
- **Çıkış :** Algoritma her bir giriş kümesinde çıkış değerleri üretir. Bu
- değerler problemin çözümüdür.
- **Açıklık :** Algoritmanın adımları açık olarak tanımlanmalıdır.
- **Doğruluk :** Her bir giriş kümesi için doğru çıkış üretmelidir.
- **Sonluluk :** Algoritma, her bir giriş kümesi için amaçlanan çıkışı, sonlu işlem adımı(büyük olabilir) sonunda üretmelidir
- **Verimlilik :** Algoritmanın her bir adımı tam ve sonlu bir zaman
- diliminde gerçeklenmelidir.
- **Genellik :** Yordam formdaki her probleme uygulanabilecek şekilde genel olmalıdır.



## Algoritmaların Özellikleri

Verilen listedeki en büyük tamsayıyı bulma algoritması bu kriterlere göre değerlendirilirse;

- Giriş : Sonlu sayıda tamsayı kümesi
- Çıkış : Kümedeki en büyük tamsayı
- Açık olarak adımlar tanımlanmıştır ve doğru sonuç üretir.
- Algoritma sonlu işlem adımı kullanır (n adım)
- Algoritma her bir adımda bir karşılaştırma işlemi yapar (verimlilik).
- Algoritma bu tür kümelerdeki en büyük tamsayıyı bulacak
- şekilde geneldir.



## Algoritma Analizi

- Algoritmaların kullandığı bilgisayar kaynaklarının (işletim süresi, bellek miktarı vb.) önemli olduğu uygulamalarda etkili algoritmalar geliştirmek gerekir.
- Bir problemin çözümünde, kullanılabilecek olan algoritmalardan en etkin olanı seçilmelidir.
- Daha kısa zaman alan bir algoritma yazmak için daha çok kod yazmak veya daha çok bellek kullanmak gerekebilir.
- Bazı durumlarda da en az bellek harcayan algoritmanın kullanılması gerekebilir.
- Derleyici özellikleri, bilgisayar mimarisi, gerçekleştirim kalitesi vb. konular algoritmanın performansını etkiler.



## Algoritma Analizi

Neden algoritmayı analiz ederiz?

- Algoritmanın performansını ölçmek için
- Farklı algoritmalarla karşılaştırmak için
- Daha iyisi mümkün mü?
- Olabileceklerin en iyisi mi?

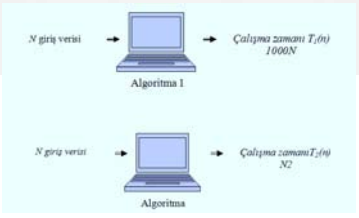
Özelliklerinin analizi

- Algoritmanın çalışma zamanı
- Hafızada kapladığı alan



## Algoritma Analizi

Aynı işi yapan fakat farklı yazılmış iki algoritmaya yanı verileri girdi olarak verdiğimizde farklı çalışma zamanlarında işlerini bitirdiklerini gözleriz.

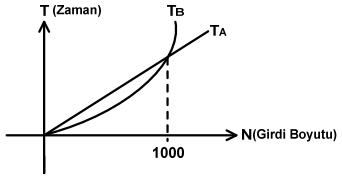


## Algoritma Analizi

**A Algoritması** $T_A(N) = 1000N$

**B Algoritması** $T_B(N) = N^2$

Yukarıda A ve B algoritmaları için geçerli olan zaman formülleri verilmiştir. N, girdi boyutunu ifade etmektedir. Her iki algoritma için harcanan zaman farklıdır. Girdi boyutuna göre değişiklik göstermektedir.




## Algoritma Analizi

Problemi çözmek için algoritmanın
- harcadığı zamanın analizi
  **zaman karmaşıklığı**,
- gerekli belleğin analizi ise
  **yer (space) karmaşıklığın**ın

hesabını gerektirir.

• Yer karmaşıklığı algoritmayı gerçeklerken kullanılan veri yapıları ile bağlantılıdır.

• Zaman karmaşıklığı ise, belirli miktardaki giriş verisine karşılık, yapılan karşılaştırma, tamsayı işlemleri ile diğer basit işlemlerin sayısı olarak hesaplanır.

## Algoritma Analizi

Bilgisayarda hesaplamalar genellikle
- Atama
- Karşılaştırma (**==, ≠, >, <, ≤, ≥**)
- Aritmetik işlemler (**+, -, ×, /**)
- Mantıksal operasyonlar

Yardımıyla gerçekleştirilir:

Bu işlemlerden
- Genelde, atama çok hızlı yapılır ve diğer işlemler daha yavaştır.
- Çarpma ve bölme de toplama ve çıkarmadan daha yavaştır.
- Kayan noktalarla işlem yapmak, tamsayılarla işlem yapmakta daha yavaştır.

• Bir çok algoritmada çokça kullanılan yukarıdaki işlemlerin hepsinin ne kadar zaman alacağını teker teker hesaplamak zor olacağından bir çok **algoritmada zaman karmaşıklığı, baskın olan tek bir işlem** ile belirlenebilmektedir.

## İşletim Zamanı (Running Time) Karmaşıklığı

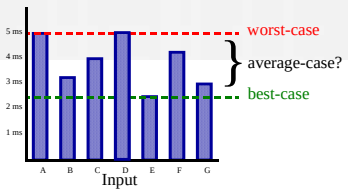
Zamanın girdi boyutunun bir fonksiyonu olarak ele alınmasıdır. Tüm girdilerin tek değere indirilmesi, değişik algoritmaları karşılaştırmayı kolaylaştırır.

**En Kötü Durum İşletim Süresi (Worst-Case Running Time):**
Bir programın en kötü ihtimalle ne kadar süreceğinin tahmin edilmesi istenen bir durumdur.
n elemanlı bir listede sıralı arama en kötü ihtimalle?

*Yani En Kötü Durum İşletim Süresi T(n) = n'dir.*

## Zaman Karmaşıklığı

Tüm problemlerde sadece en kötü girdi dikkate alındığı için en kötü durum işletim süresi değerini hesaplamak göreceli olarak kolaydır.




## Zaman Karmaşıklığı

**Ortalama Durum İşletim Süresi (Average-Case Running Time):**
Her girdi boyutundaki tüm girdilerin ortalamasıdır.
n elemanın her birinin aranma olasılığının eşit olduğu ve liste dışından bir eleman aranmayacağı varsayıldığında
ortalama işletim süresi (n+1)/2'dir.

## Zaman Karmaşıklığı

$O(n), \Theta(n), \Omega(n)$

**Big Oh Notasyonu O(n)**

- Paul Bachman tarafından tanıtılmıştır.
- *Zaman karmaşıklığında üst sınırı* gösterir.
- Bu notasyon bir çok ifadeyi sadeleştirerek göstermemizi sağlar.
- Ör. n^3 + n^2+3n gibi bir ifadeyi O(n^3) olarak ifade edilir.

**Big Omega Notasyonu**

- Big Oh notasyonunun tam tersidir.
- Zaman karmaşıklında alt sınırı gösterir.
- Omega ile ölçülen değerden daha hızlı bir değer elde edilemez.

**Big Theta Notasyonu**

- Bu notasyon big Oh notasyonu ile big Omega notasyonu arasında ortalama bir karmaşıklığı ifade eder.

13

## Zaman Karmaşıklığı

Zaman karmaşıklığı, girdi boyutu sonsuza yaklaşırken işletim süresinin artışını temsil eder.

girdi boyutunun büyük olduğunu kabul ederek, fonksiyonda **en hızlı artış gösteren terimin belirlenmesidir**.

Bu fonksiyon yaklaşımı matematiksel gösterim kullanarak **Big-O gösterimi veya Zaman Karmaşıklığı** olarak ifade edilir.

Örnek

$f(n) = n^4 + 100n^3 + 50n^2 + 20n + 60 = O(n^4)$

n'in çok büyük değerleri için $n^4$, diğer terimlere göre çok büyük olacağından daha düşük dereceli terimler dikkate alınmayabilir.

Bu diğer terimlerin, işlem süresini **etkilemedikleri anlamına gelmez**; **n'in çok büyük değerlerinde önem taşımadıkları** anlamına gelir.

14

## Zaman Karmaşıklığı

Bilgisayarın yaptığı isin programın boyutu / satır sayısı ile ilgili olması gerekmez.

N elemanlı bir diziyi 0'layan iki program da O(n) olduğu halde kaynak kodlarının satır sayıları oldukça farklıdır :

*Program 1 :*

```
toplam = 0;
for(int i=0; i<n; ++i)
toplam = toplam + i;
```

Program 1, O(n)'dir. n=50 olursa programın çalısması sırasında n=5 için harcanan sürenin yaklasık 10 katı süre harcanacaktır. Program 2 ise

*Program 2 :*

```
toplam = n * (n+1) / 2;
```

O(1)'dir. n=1 de olsa n=50'de olsa program aynı sürede biter.

15

## Zaman karmaşıklığı hesaplamadaki kurallar

1. Tek satırlık, dizinin boyutuyla ilişkisi olmayan komutlar O(1) zamanı alır.
2. Döngüler içerdikleri döngü komutlarının belli bir şarta kadar icrasını sağlarlar, bu nedenle O(n) zamanı alır.
3. İçiçe döngüler zaman karmaşıklıklarının çarpılması şeklinde hesaplanır. Birinci döngü icra sayısı n, ikinci döngü m ise, O(nxm); ikisi de n ise $O(n^2)$ zamanı alır.
4. Ardarda döngüler zaman karmaşıklıklarının toplanması şeklinde hesaplanır.
5. If-then-else gibi şartlı yapılarda koşullardan zaman karmaşıklığı yüksek olan alınır.
6. Her bir icra, problemi dizi boyutunun yarısı şeklinde ikiye bölüyorsa, zaman karmaşıklığı O(log2n) değerini alır.

Not 1: Katsayısı daha küçük olan zaman karmaşıklıkları ihmal edilir (O(n+n2)= O(n2) )
Değerlerinin önlerindeki sabit çarpanlar önemsenmez. (O(6n) = O(n) gibi)

16

## Büyük-O nasıl hesaplanır?

Bir program kodunun zaman karmaşıklığını hesaplamak için 5 kural

1. Döngüler
2. İç içe Döngüler
3. Ardışık deyimler
4. If-then-else deyimleri
5. Logaritmik karmaşıklık

17

## Büyük-O nasıl hesaplanır?

Kural 1: Döngüler

Bir döngünün çalışma zamanı en çok döngü içindeki deyimlerin çalışma zamanının iterasyon sayısıyla çarpılması kadardır.

*n* defa çalışır

```
for (i=1; i<=n; i++)
{
    m = m + 2;   <-- Sabit zaman
}
```

Toplam zaman = sabit c * n = cn = **O(*N*)**

18

## Büyük-O nasıl hesaplanır?

Kural 2:İç içe Döngüler

Toplam zaman bütün döngülerin çalışma sayılarının çarpımına eşittir

Dış döngü *n* defa çalışır

```
for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++) {
        k = k+1;
    }
}
```

Sabit zaman

iç döngü *n* defa çalışır

Toplam zaman = $c * n * n = cn^2 = O(N^2)$

19

## Büyük-O nasıl hesaplanır?

Kural 3: Ardışık deyimler

Her deyimin zamanı birbirine eklenir.

Sabit zaman

```
x = x +1;
```

Sabit zaman

```
for (i=1; i<=n; i++) {
    m = m + 2;
}
```

*n* defa çalışır

Dış döngü *n* defa çalışır

```
for (i=1; i<=n; i++) {
    for (j=1; j<=n; j++) {
        k = k+1;
    }
}
```

Sabit zaman

iç döngü *n* defa çalışır

toplam zaman = $c_0 + c_1n + c_2n^2 = O(N^2)$

20

## Kural 4: If-then-else deyimleri

En kötü çalışma zamanı:test zamanına then veya else kısmındaki çalışma zamanının hangisi büyükse o kısım eklenir.

test: sabit

```
if (depth( ) != otherStack.depth( ) ) {
    return false;
}
else {
    for (int n = 0; n < depth( ); n++) {
    if (!list[n].equals(otherStack.list[n]))
        return false;
    }
}
```

**then**: sabit

**else**: (sabit +sabit) * n

Diğer if : sabit+sabit (else yok)

Toplam zaman = $c_0 + c_1 + (c_2 + c_3) * n = O(N)$

21

## Büyük-O nasıl hesaplanır?

Kural 5: Logaritmik karmaşıklık

Problemin büyüklüğünü belli oranda (genelde ½) azaltmak için sabit bir zaman harcanıyorsa bu algoritma O(log *N*)'dir.

Örnek algoritma (binary search)

22

## Büyük-O nasıl hesaplanır?

O notasyonu yazılırken en basit şekilde yazılır.

$3n^2+2n+5 = O(n^2)$

Aşağıdaki gösterimlerde doğrudur fakat kullanılmaz.

$3n^2+2n+5 = O(3n^2+2n+5)$

$3n^2+2n+5 = O(n^2+n)$

$3n^2+2n+5 = O(3n^2)$

23

## Örnek

```
1 for (i=1; i<n; i++) {
2     a[i] = 0;
3 }
4 for (i=1; i<n; i++) {
5     for (j=1; j<n; j++) {
6         a[i] = a[i] + i + j;
7     }
8 }
```

O(1) } O(n)

O(1) } O(n) } O(n²)

O(n²)

24

## Örnek

```
1 for (i=1; i<n-1; i++) {
2    for (j=n; j>i+1; j--) {
3       if (a[j-1] > a[j]) {
4          temp = a[j-1];        O(1)
5          a[j-1] = a[j];        O(1)   O(1)   O(n)   O(n²)
6          a[j] = temp;          O(1)
7       }
8    }
9 }
```

25

## Başlıca zaman karmaşıklığı değerleri ve incelenmesi

O(1) : Sabit zaman
Örnek : n elemanlı bir dizinin i. elemanına bir değer atanması O(1)'dir.

O(n) : Doğrusal zaman
Örnek : n elemanlı bir dizinin tüm elemanlarının ekrana yazdırılması O(n)'dir.

O(log2 n) : Logaritmik zaman, O(1)'den fazla O(n)'den azdır.
Örnek : Sıralı bir listenin elemanları içinde ikili arama (binary search) O(log2 n)

$O(n^2)$ : Karesel zaman
Örnek : Basit sıralama algoritmalarının birçoğu (selection sort)

O(n.log2 n) : Bazı hızlı sıralama algoritmaları
Örnek : (merge sort) O(n.log2 n)

$O(n^3)$ : Kübik zaman
Örnek : Üç boyutlu bir tamsayı tablosundaki her elemanın değerini artıran algoritma.

26

## Başlıca zaman karmaşıklığı değerleri ve incelenmesi

Değişik artış fonksiyonlarının aldıkları değerlere göre zaman karmaşıklıkları aşağıdaki tabloda gösterilmiştir

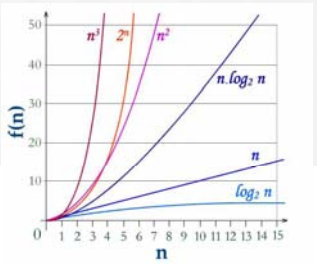
Değişik fonksiyonların f(n) değişik girdi boyutlarına (n) göre değerleri

| $\log_2 n$ | n | $n.\log_2 n$ | $n^2$ | $n^3$ | $2^n$ |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 2 |
| 1 | 2 | 2 | 4 | 8 | 4 |
| 2 | 4 | 8 | 16 | 84 | 16 |
| 3 | 8 | 24 | 64 | 512 | 256 |
| 4 | 16 | 64 | 256 | 4096 | 65536 |
| 5 | 32 | 160 | 1024 | 32768 | 4294967296 |

27

## Başlıca zaman karmaşıklığı değerleri ve incelenmesi

Tabloda görüldüğü gibi n değeri büyük olan problemlerde, O(n3), O(2n), O(n2) vb. zaman karmaşıklığına sahip algoritmaların oldukça yavaş sonuç ürettiklerini görülmektedir.




N değerinin fazla olduğu problemlerde, zaman karmaşıklığı parabol şeklinde olan algoritmaları kaçınmalı, O(n.log2 n) zaman karmaşıklığını geçmemeye dikkat etmli

28

İyi çalışmalar…

29