# Algoritmik Program Tasarımı ve Analizi

Doç. Dr. Fatih ÖZYURT

# Algoritmik Program Tasarımı Nedir?

- Verilen bir problemin bilgisayar ortamında çözülecek biçimde adım adım ortaya koyulması ve herhangi bir programlama aracıyla kodlanması sürecidir.
- Çözüm için yapılması gereken işlemler hiçbir alternatif yoruma izin vermeksizin sözel olarak ifade edilir.
- Verilerin, bilgisayara hangi çevre biriminden girileceğinin, problemin nasıl çözüleceğinin, hangi basamaklardan geçirilerek sonuç alınacağının, sonucun nasıl ve nereye yazılacağının sözel olarak ifade edilmesi biçiminde de tanımlanabilir.

# Algoritmaların Özellikleri

- Bir algoritmanın taşıması gereken beş tane temel özelliği vardır.

- 1. Giriş (Input)
  - Bir algoritmanın sıfır veya daha fazla giriş değişkeni vardır. Giriş değişkenleri algoritma işlemeye başlamadan önce, algoritmaya verilen değerler kümesidir veya değer kaydetmesi için verilen hafıza bölgesidir.
- 2. Belirlilik (Definiteness)
  - Bir algoritmanın her adımı için kesin olarak ne iş yapacağı belirlenmelidir ve belirsizlik olmamalıdır. Her durum için hangi işlem gerçekleştirilecekse, o açık olarak tanımlanmalıdır.

# Algoritmaların Özellikleri

- 3.Çıkış (Output))
  - Her algoritmanın bir veya daha fazla çıkış değeri vardır. Çıkış değerleri ile giriş değerleri arasında bağıntılar vardır.

- 4.Etkililik (Efficiency)
  - Doğal olarak her algoritmanın etkili olarak işlem yapması beklenir. Olabildiğince hızlı çalışmalıdır, olabildiğince az hafıza kullanmalıdır.
  - Bunun anlamı yapılan işlemler yeterince temel işlemler olacak ve sınırlı zaman süresince işleyip bitmelidir.

# Algoritmaların Özellikleri

- 5. Sınırlılık (Boundedness)
  - Her algoritma sınırlı sayıda çalışma adımı sonunda bitmelidir.

- Bir algoritma için sınırlılık çok önemlidir. Aynı işlemi yapan iki algoritmadan biri bir milyar adımda bitiyor olsun ve diğeri de yüz adımda bitiyor olsun. Bu durumda yüz adımda biten algoritma her zaman daha iyidir. Bunun anlamı sınırlılık kavramı ile anlatılmak istenen mümkün olan en az sayıda adım ile işlemin bitirilmesidir. Diğer bazı kriterler ise algoritmanın bilgisayar ortamına aktarılabilme özelliği, basitliği, vb. gibi özelliklerdir.

# Algoritma Süreci

- Tasarım(design)
- Doğruluğunu ispat etme (validation)
- Analiz(analysis)
- Uygulama(implementation)
- Test

# Kaba-Kod (Pseudo Code)

- Kaba-kod, bir algoritmanın yarı programlama kuralı, yarı konuşma diline dönük olarak ortaya koyulması, tanımlanması, ifade edilmesidir.

- Kaba-kod, çoğunlukla, bir veriyapısına dayandırılmadan algoritmayı genel olarak tasarlamaya yardımcı olur.

# Gerçek Kod

- Algoritmanın herhangi bir programlama diliyle, belirli bir veri yapısı üzerinde gerçekleştirilmiş halidir.
- Bir algoritmanın gerçek kodu, yalnızca, tasarlandığı veri yapısı üzerinde çalışır.
- Bir algoritma kaba-kod ile verilirse gerçek kod verilmesin den daha kolay anlaşılır.

# Kaba-kod: temel gösterim

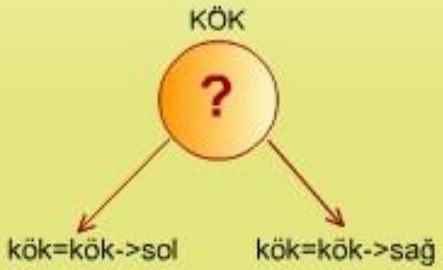
- 1. Bir değer atamak için genellikle " := " kullanılır, "=" işareti ise eşitlik kontrolü için kullanılır.
- 2. Metot, fonksiyon, yordam isimleri:
  Algoritma Adı ({parametre listesi})
- 3. Program yapısı şu şekilde tanımlanır::
  - Karar yapıları: if … then … else …
  - while döngüleri: while … do {döngü gövdesi}
  - Tekrar döngüleri: repeat {döngü gövdesi} until …
  - for döngüleri: for … do {döngü gövdesi}
  - Dizi indeksleri: A[i]
- 4. Metotların çağrılması: Metot adı ({değişken listesi})
- 5. Metotlardan geri dönüş: return değer

# Algoritmaların kaba-kod olarak ifade edilmesi

- Örnek: Bir dizideki elemanların toplam ve çarpımını hesaplayan algoritmayı kaba-kod kullanarak tanımlayınız.
  - Toplam ve Çarpım Hesapla (dizi, toplam, çarpım)
  - Girdi: n sayıdan oluşan dizi.
  - Çıktı: dizi elemanlarının toplam ve çarpım sonucu
  - for i← 1 to n do
  - toplam← toplam + dizi[i]
  - çarpım← çarpım* dizi[i]
  - end for

# Kaba-kod ve Gerçek Kod

```
/* İkili arama ağacına düğüm ekleme algoritmasının kaba-kodu */
if (kök boş ağacı gösteriyorsa)
   Düğümü köke ekle;
else {
    if (eklenen kökten küçükse)
     sol altağaca dallan; (kök=kökün sol çocuğu adresi )
     başa dön;
    else
     sağ altağaca dallan; (kök=kökün sağ çocuğu adresi )
     başa dön;
    }
```




```
/* Ikili arama agacina düğüm ekleme fonksiyonu */
void ekle (agacKok, eklenen )
            AGAC2 *agacKok, *eklenen;
{
  if (agacKok==NULL )
     kok=eklenen;
     else {
        if (eklenen->bilgi <= agacKok->bilgi)
          {
           if (agacKok->sol==NULL)
              agacKok->sol=eklenen;
           else ekle (agacKok->sol, eklenen );
          }
        else
          {
            if (agacKok->sag==NULL )
               agacKok->sag=eklenen;
            else ekle (agacKok->sag, eklenen );
           }
        }
}
```

# Algoritma Analizi

- Algoritmanın icra edilmesi sırasında duyacağı kaynak miktarının tahmin edilmesine Algoritma Analizi denir.
- Kaynak denildiğinde, bellek, iletişim bant genişliği, mantık kapıları akla gelebilir, fakat en önemli kaynak algoritmanın icra edilebilmesi için gerekli olan zamanın tahmin edilmesidir.
- Algoritma analizi, tasarlanan program veya fonksiyonun belirli bir işleme göre matematiksel ifadesini bulmaya dayanır.
- Burada temel hesap birimi seçilir ve programın görevini yerine getirebilmesi için bu işlemden kaç adet yapılması gerektiğini bulmaya yarayan bir bağıntı hesaplanır.
- Eğer bu bağıntı zamanla ilgiliyle çalışma hızını, bellek gereksinimiyle ilgiliyse bellek gereksinimi ortaya koyar.

# Algoritma Analizi

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

- Algoritmayı nasıl analiz ederiz?
  - Çalışma zamanı (Running Time)-T(n)
  - Karmaşıklık (Complexity) -Notasyonlar

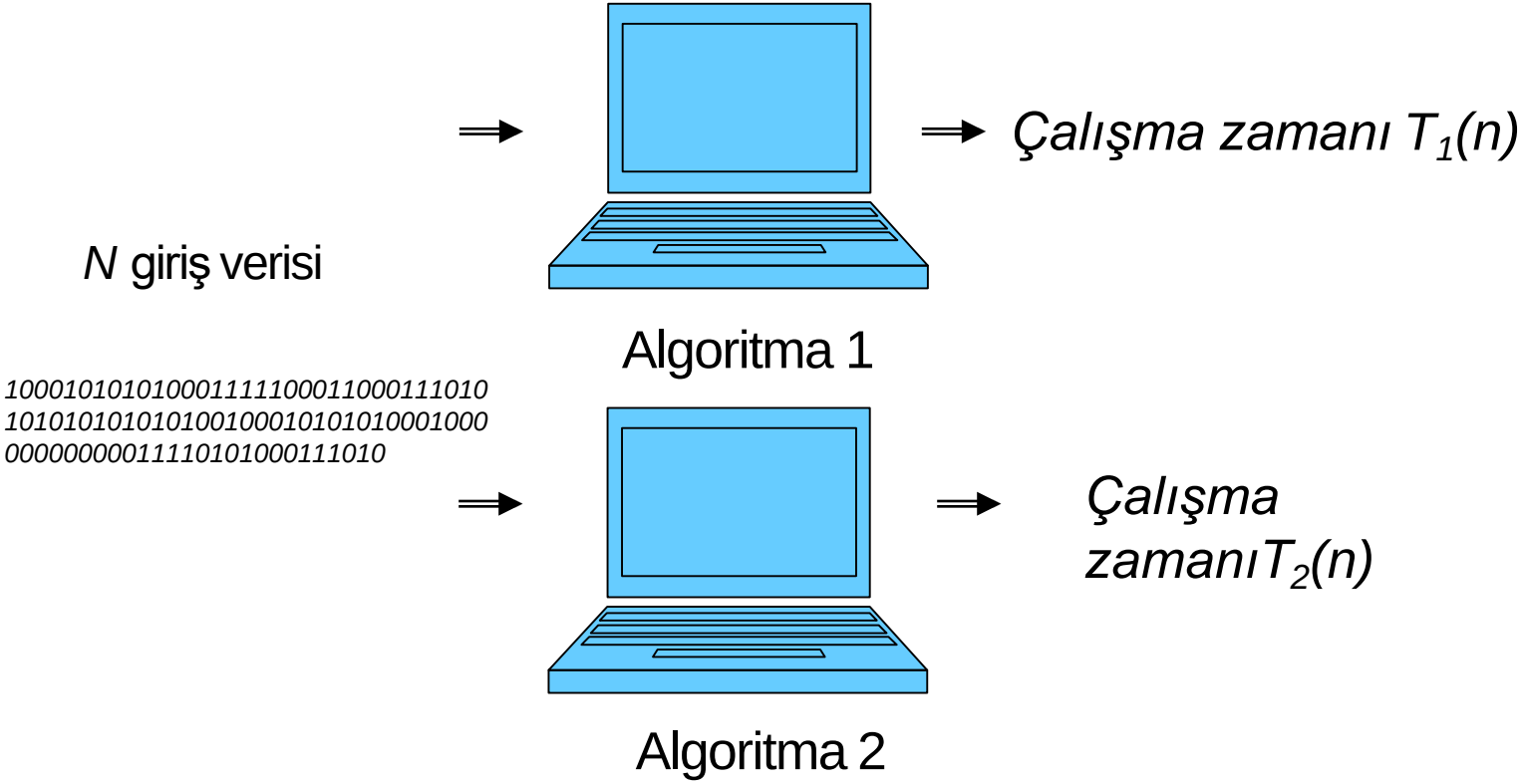
# Çalışma Zamanı Analizi (Running Time)

- Çalışma Zamanı; Bir programın veya algoritmanın işlevini yerine getirebilmesi için, temel kabul edilen işlevlerden kaç adet yürütülmesini veren bir bağıntıdır ve T(n) ile gösterilir.

- Temel hesap birimi olarak, programlama dilindeki deyimler seçilebildiği gibi döngü sayısı, toplama işlemi sayısı, atama sayısı, dosyaya erişme sayısı gibi işler de temel hesap birimi olarak seçilebilir.

# Çalışma Zamanı Analizi

- Algoritma 1, $T_1(N) = 1000*N$
- Algoritma 2, $T_2(N) = N^2$



# Örnek I: Dizideki sayıların toplamını bulma

```
int Topla(int A[], int N)
{
  int toplam = 0;

  for  (i=0; i < N; i++){
    toplam += A[i];
  } //Bitti-for

  return toplam;
} //Bitti-Topla
```

- Bu fonksiyonun yürütme zamanı ne kadardır?

# Örnek I: Dizideki sayıların toplamını bulma

```
int Topla(int A[], int N)
{
  int topla = 0;

  for (i=0; i < N; i++){
    topla += A[i];
  } //Bitti-for

  return topla;
} //Bitti-Topla
```

İşlem sayısı

1

N

N

1

Toplam: 1 + N + N + 1 = 2N + 2

- Çalışma zamanı: T(N) = 2N+2
  - N dizideki sayı sayısı

# Örnek II: Dizideki bir elemanın aranması

```
int Arama(int A[], int N,
                  int sayi) {
  int i = 0;

  while (i < N){
    if (A[i] == sayi) break;
    i++;
  } //bitti-while

  if (i < N) return i;
  else return -1;
} //bitti-Arama
```

İşlem sayısı

1

1<=L<=N

1<=L<=N

0<=L<=N

1

1

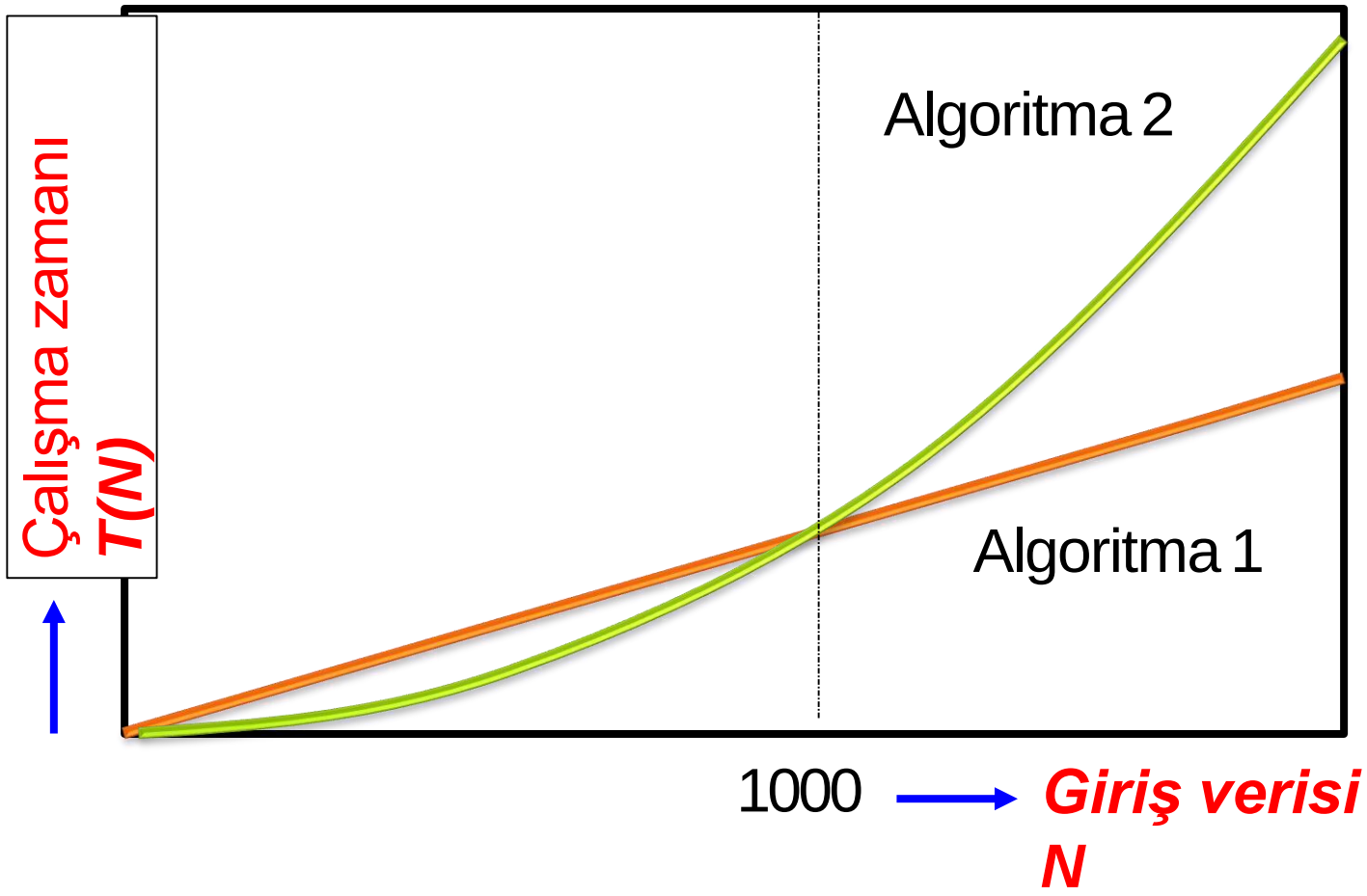
---

Toplam: 1+3*L+1+1 = 3L+3

# Örnek II: Dizideki bir elemanın aranması

- **En iyi çalışma zamanı nedir?**
  - Döngü sadece bir kez çalıştı=>T(n) = 6
- **Ortalama(beklenen) çalışma zamanı nedir?**
  - Döngü N/2 kez çalıştı =>T(n)=3*n/2+3 = 1.5n+3
- **En kötü çalışma zamanı nedir?**
  - Döngü N kez çalıştı =>T(n) = 3n+3

# Çalışma Zamanı Analizi



# Çalışma Zamanlarının Özeti

| N | T1 | T2 |
|---|---|---|
| 10 | $10^{-2}$ sec | $10^{-4}$ sec |
| 100 | $10^{-1}$ sec | $10^{-2}$ sec |
| 1000 | 1 sec | 1 sec |
| 10000 | 10 sec | 100 sec |
| 100000 | 100 sec | 10000 sec |

N değerinin 1000'den küçük olduğu durumlarda iki algoritma arasındaki çalışma zamanı ihmal edilebilir büyüklüktedir.

# Karmaşıklık (Copmlexity)

- Karmaşıklık; bir algoritmanın çok sayıda parametre karşısındaki değişimini gösteren ifadelerdir. Çalışma (Yürütme) zamanını daha doğru bir şekilde bulmak için kullanılırlar.

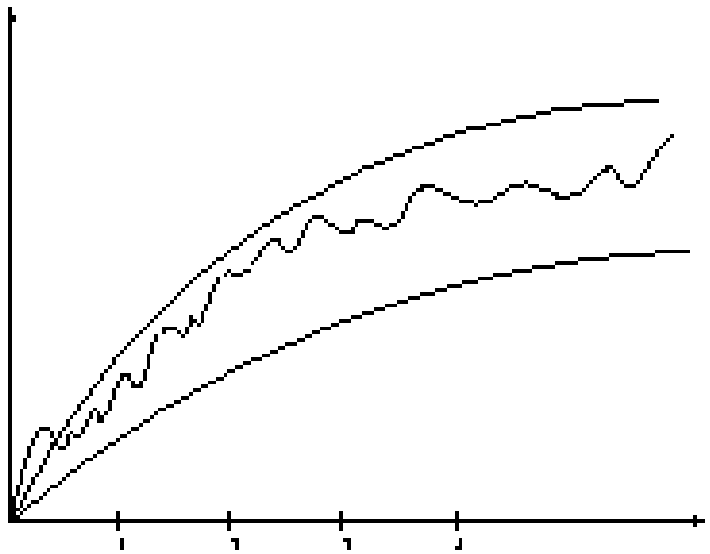
- Genel olarak, az sayıda parametreler için karmaşıklıkla ilgilenilmez; eleman sayısı ***n*** nin sonsuza gitmesi durumunda algoritmanın maliyet hesabının davranışını görmek veya diğer benzer işleri yapan algoritmalarla karşılaştırmak için kullanılır.

- Karmaşıklığı ifade edebilmek için asimtotik ifadeler kullanılmaktadır.

- Bu amaçla O(n) (O notasyonu), $\Omega$(n) (Omega notasyonu), $\theta$(n) (Teta notasyonu) gibi tanımlamalara baş vurulur.

# Karmaşıklık (Copmlexity)

- Strateji: Alt ve üst limitlerin bulunması



Üst limit – O(n)

Algoritmanın gerçek fonksiyonu

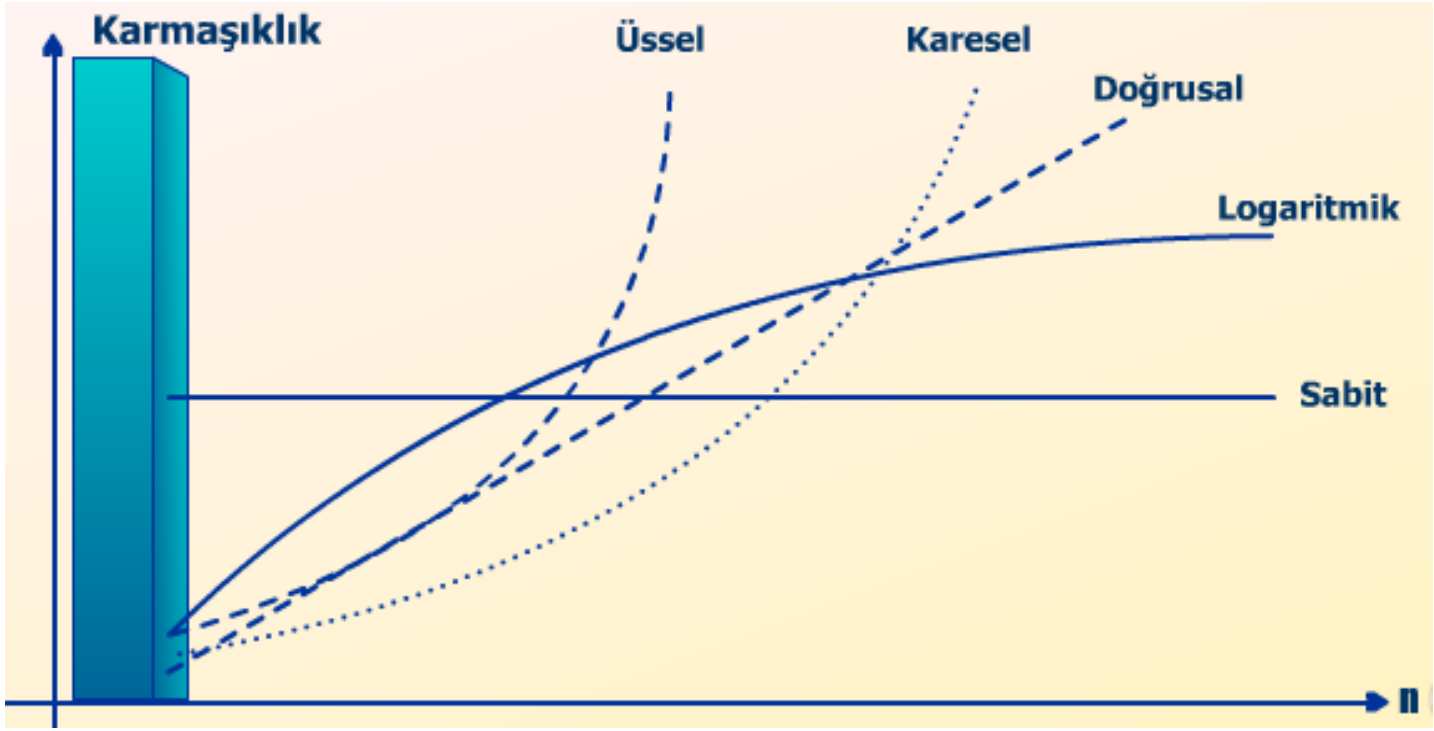
Alt limit- $\Omega$ (n)

# Karşılaşılan Genel Fonksiyonlar

Maliyet artar

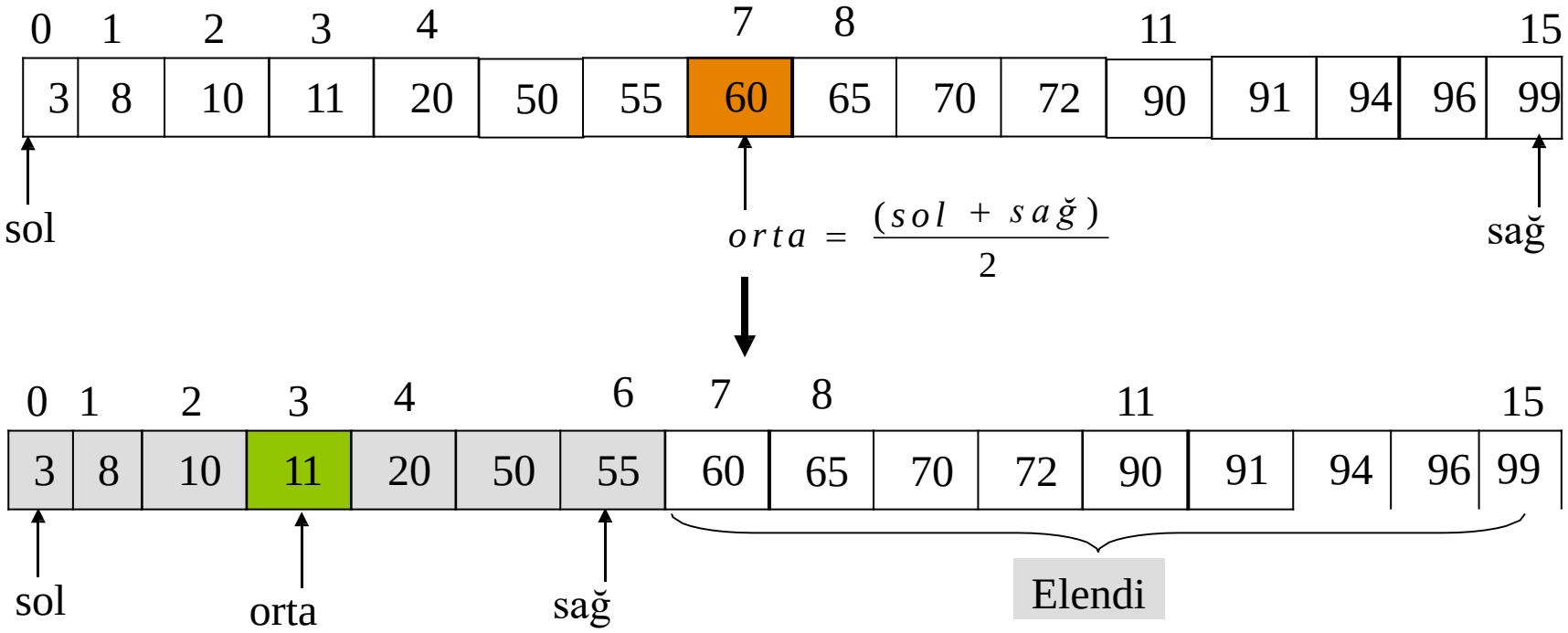
| Big-Oh | Değişim Şekli |
|---|---|
| O(1) | Sabit, komut bir veya birkaç kez çalıştırılır. Yenilmez! |
| $O(\log_n)$ | Logaritmik, Büyük bir problem, her bir adımda sabit kesirler tarafından orijinal problemin daha küçük parçalara ayrılması ile çözülür. İyi hazırlanmış arama algoritmalarının tipik zamanı |
| O(n) | Doğrusal, Küçük problemlerde her bir eleman için yapılır. Hızlı bir algoritmadır. N tane veriyi girmek için gereken zaman. |
| $O(n\log_n)$ | Doğrusal çarpanlı logaritmik. Çoğu sıralama algoritması |
| $O(n^2)$ | Karasel. Veri miktarı az olduğu zamanlarda uygun (N<1000) |
| $O(n^3)$ | Kübik. Veri miktarı az olduğu zamanlarda uygun (N<1000) |
| $O(2^n)$ | İki tabanında üssel.Veri miktarı çok az olduğunda uygun (N<=20) |
| $O(10^n)$ | On tabanında üssel |
| O(n!) | Faktöriyel |

# Karmaşıklık (Copmlexity)

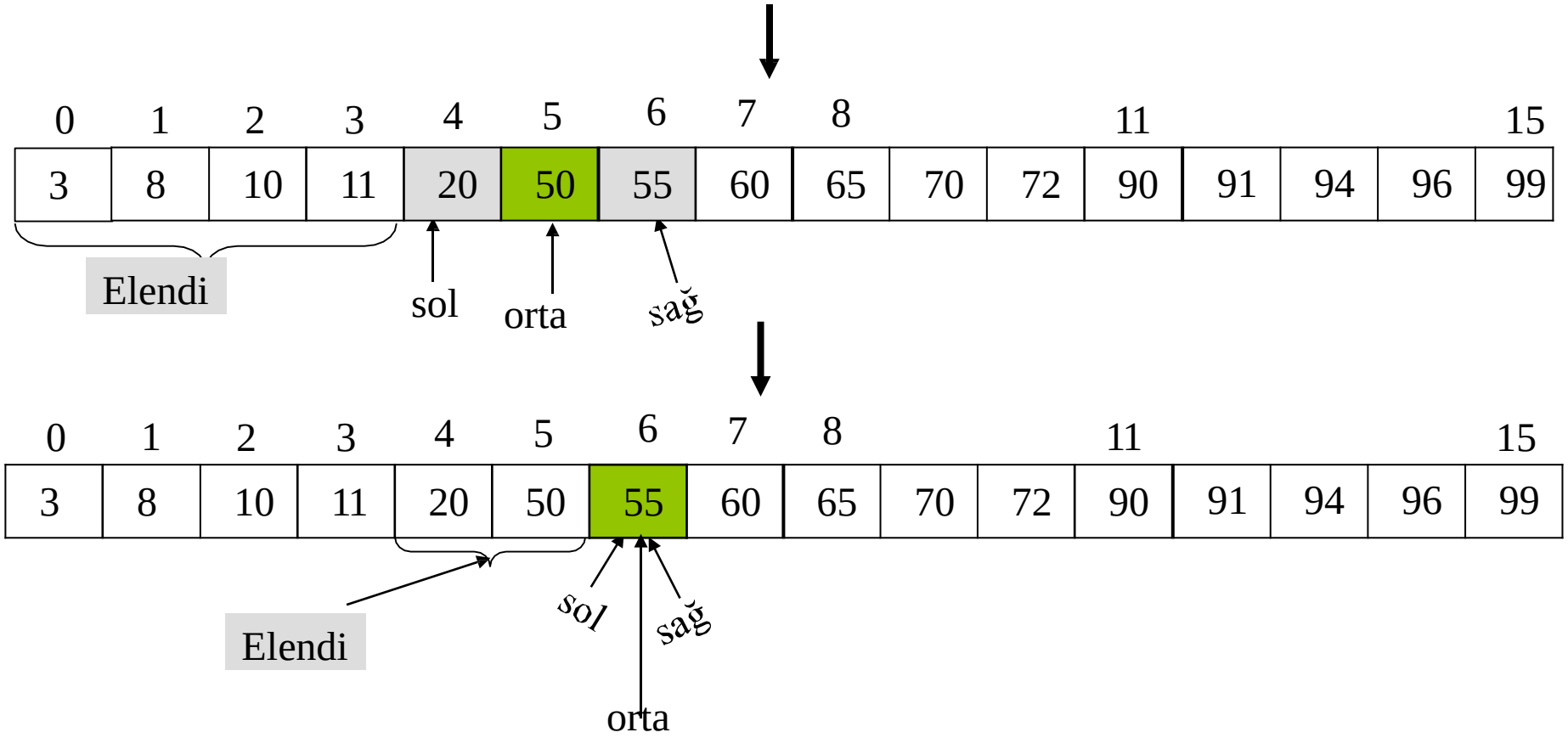


# Örnek : İkili Arama

- Dizi sıralanmış olduğundan, dizinin ortasında bulunan sayı ile aranan sayıyı karşılaştırarak arama boyutunu yarıya düşürülür ve bu şekilde devam edilir.
- Örnek: 55'i arayalım

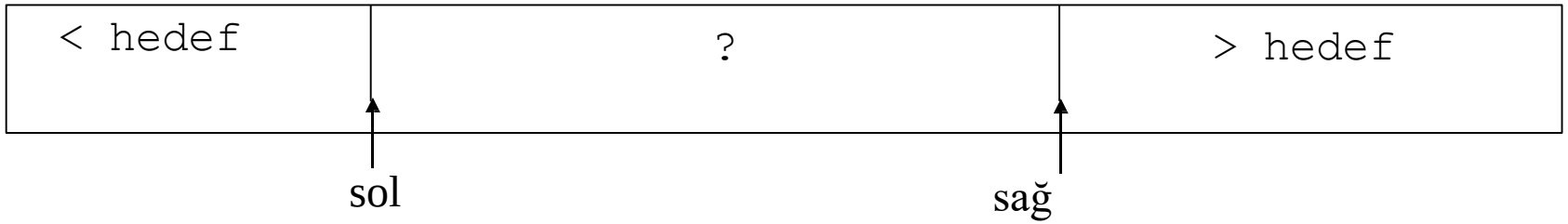


# İkili arama (devam)




- 55'i bulduk→ Başarılı arama
- 57'yi aradığımızda, bir sonraki işlemde başarısız bir şekilde sonlanacak.

# İkili Arama (devam)




- Hedefi ararken herhangi bir aşamada, arama alanımızı "sağ" ile "sol" arasındaki alana kısıtlamış oluyoruz.

- "sol" 'un solunda kalan alan hedeften küçüktür ve bu alan arama alanından çıkarılır.

- "sağ" ın sagında kalan alan hedeften büyüktür ve bu alan arama alanından çıkarılır.

# Veri Yapıları- Bölüm Özeti

- Veri modelleri ve onlara ait veri yapıları yazılım geliştirmenin temel noktalarıdır; problemlerin en etkin şekilde çözülebilmesi için ona algoritmik ifadenin doğasına yakın bulunmasıdır. Kısaca, veri yapıları, verinin saklanma kalıbını, veri modelleri de veriler arasındaki ilişkiyi gösterir.

- Bilinen ve çözümlerde sıkça başvurulan veri modelleri, genel olarak, bağlantılı liste (link list), ağaç (tree), graf (graph), durum makinası (state machine), ağ (network) ve veritabanı-ilişkisel (database-relation) şeklinde verilebilir.

- Her veri modelinin, altında duran veri yapısına bağlı olarak, işlem zaman maliyetleri ve bellek gereksinimleri farklıdır. Program geliştirilirken, zaman ve bellek alanı maliyetlerini dengeleyecek çözüm üretilmeye çalışılır. Genel olarak, RAM türü ardışıl erişimlerin yapılabildiği bellek üzerinde, maliyeti ile bellek gereksinim ters orantılı olduğu söylenebilir.

# Kaynaklar

Veri Yapıları ve Algoritmalar  – Dr. Rifat ÇÖLKESEN, Papatya yayıncılık
Veri Yapıları ve Algoritmalar-Dr. Öğ. Üyesi Ömer ÇETİN
Veri Yapıları – Prof. Dr. Erkan TANYILDIZI