



YMT219

Veri Yapıları

Yrd.Doç.Dr. Erkan TANYILDIZI

Ders Kitapları ve Yardımcı Kaynaklar

- Veri Yapıları ve Algoritmalar

- Dr. Rifat ÇÖLKESEN, Papatya yayıncılık



- Data Structures and Problem Solving Using Java

- Mark Allen Weiss, Pearson International Edition

- Ahmet Yesevi Üniversitesi Ders Notları

- Ayrıca internet üzerinden çok sayıda kaynağa ulaşabilirsiniz.

Dersin Gereksinimleri

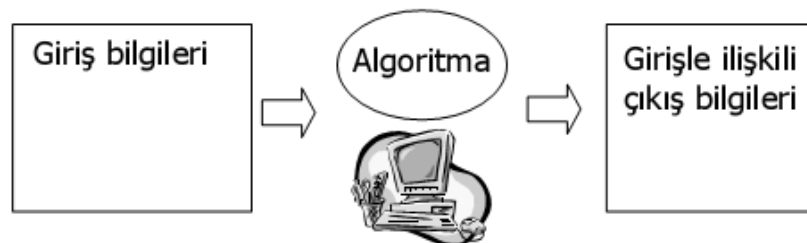
- Bu dersteki öğrencilerin Nesne tabanlı programlama dillerinden birisini(Java, C++, C#) veya yordamsal programlama dillerinden birisini(C, Pascal) bildiği varsayılmıştır.
- Bilinmesi gereken konular:
 - Temel veri türleri (int, float)
 - Kontrol yapısı (if else yapısı)
 - Döngüler
 - Fonksiyonlar(Methods)
 - Giriş çıkış işlemleri
 - Basit düzeyde diziler ve sınıflar

Veri Yapıları ve Modelleri

Bölüm 1

Algoritma

- **Algoritma**, bir problemin çözümünde izlenecek yol anlamına gelir. Algoritma, bir programlama dilinde (Java, C++, C# gibi) ifade edildiğinde **program** adını alır.
- Algoritma, belirli bir problemin sonucunu elde etmek için art arda uygulanacak adımları ve koşulları kesin olarak ortaya koyar. Herhangi bir giriş verisine karşılık, çıkış verisi elde edilmesi gereklidir. Bunun dışındaki durumlar algoritma değildir.



Algoritma

- Bilgisayar uygulamasında, bir yazılım geliştirirken birçok algoritmaya ihtiyaç duyulur. Örneğin,
 - arama algoritması
 - sıralama algoritması
 - matris veya vektörel işlem algoritması
 - graf algoritması
 - bir matematiksel modelin çözülmesi algoritması
- gibi birçok algoritma türü vardır ve uygulama geliştirirken, bunların biri veya birkaçı her zaman kullanılır.

Veri

- **Veri**, algoritmalar tarafından işlenen en temel elemanlardır (sayısal bilgiler, metinsel bilgiler, resimler, sesler ve girdi, çıktı olarak veya ara hesaplamalarda kullanılan diğer bilgiler...).
- Bir algoritmanın etkin, anlaşılır ve doğru olabilmesi için, algoritmanın işleyeceği verilerin düzenlenmesi gerekir.

Veri Yapısı ve Veri Modeli

- **Veri yapısı** (Data Structure) verinin veya bilginin bellekte tutulma şeklini veya düzenini gösterir.
 - Tüm programlama dillerinin, genel olarak, tamsayı, kesirli sayı, karakter ve sözcük saklanması için temel veri yapıları vardır. Bir program değişkeni bile **basit bir veri yapısı** olarak kabul edilebilir.
- **Veri modeli** (Data Model), verilerin birbirleriyle ilişkisel veya sırasal durumunu gösterir; problemin çözümü için kavramsal bir yaklaşım yöntemidir denilebilir.
 - Bilgisayar ortamında uygulanacak tüm matematik ve mühendislik problemleri bir veri modeline yaklaştırılarak veya yeni veri modelleri tanımlaması yapılarak çözülebilmektedir.

Veri Yapılarına Neden İhtiyaç Vardır?

- Bilgisayar yazılımları gün geçtikçe daha karmaşık bir hal almaktadır.
 - Örneğin 8 milyar sayfanın indekslenmesi (Google)
- Yazılımların programlanması ve yönetimi zorlaşmaktadır.
- **Temiz kavramsal yapılar** ve bu yapıları sunan çerçeve programları, **daha etkin ve daha doğru program yazmayı sağlar.**

Veri Yapılarına Neden İhtiyaç Vardır?

- İyi bir yazılım için gereksinimler:

- Temiz bir tasarım
- Kolay bakım ve yönetim
- Güvenilir
- Kolay kullanımlı
- Hızlı algoritmalar

- Verimli Veri Yapıları

- Verimli Algoritmalar

Veri Yapılarına Neden İhtiyaç Vardır?

○ Örnek

- Her biri satır başına ortalama 10 kelimeden ve yine ortalama 20 satırdan oluşan 3000 metin koleksiyonu olduğunu düşünelim.
 - →600,000 kelime
- Bu metinler içinde “dünya” kelimesi ile eşleşecek bütün kelimeleri bulmak isteyelim
- Doğru eşleştirme için yapılacak karşılaştırmanın 1 sn. sürdüğünü varsayalım.
- Ne yapılmalıdır?

Veri Yapılarına Neden İhtiyaç Vardır?

- Örnek
- Çözüm. 1:
- Sıralı eşleştirme: 1 sn. x 600,000 kelime= 166 saat
- Çözüm. 2:
- İkili Arama (Binary searching):
 - - kelimeler sıralanır
 - - sadece tek yarıda arama yapılır
 - toplam adım sayısı $\log_2 N = \log_2 600000$ yaklaşık 20 adım (çevrim) 20 sn.
- 20 saniye veya 166 saat!

Veri Yapılarına Neden İhtiyaç Vardır?

- **Örnek** :25 değerini 5 8 12 15 15 17 23 25 27 dizisinde arayalım. Kaç adımda sonuç bulunur?
 - 25 ? 15 15 17 23 25 27
 - 25 ? 23 23 25 27
 - 25 ? 25

Veri Yapılarının Sınıflandırılması

- Veri yapıları,
 - **Temel Veri Yapıları**
 - **Tanımlamalı (Bileşik) Veri Yapıları**
- Temel veri yapıları, daha çok programlama dilleri tarafından doğrudan değişken veya sabit bildirimi yapılarak kullanılır.
- Tanımlamalı veri yapıları, kendisinden önceki tanımlamalı veya temel veri yapıları üzerine kurulurlar; yani, önceden geçerli olan veri yapıları kullanılarak sonradan tanımlanırlar.

Veri Yapılarının Sınıflandırılması

- Programlama dilinin elverdiği ölçüde, hemen her tür veri yapısı tanımlanabilir. C Programlama dilinde yeni veri yapısı tanımlamak için struct, union gibi birkaç deyim vardır.
- Aşağıdaki bildirime göre **tam**, **kr** ve **kesirli** adlı değişkenler, C programlama dilinde birer temel veri yapısıdır; ancak, **toplam** adlı değişken ise, tanımlamalı veri yapısı şeklindedir. struct karmasik adlı veri yapısının 2 tane üyesi vardır; biri **gercel**, diğeri **sanal** kısmı tutmak için kullanılır.

```
int tam ;  
char kr ;  
float kesirli;
```

```
struct karmasik {  
    float gercel;  
    float sanal;    };  
struct karmasik toplam;
```

Temel Veri Yapıları

- Tüm programlama dillerinin, genel olarak, karakter, tamsayı, kesirli sayı ve sözcük (karakter katarı) saklanması için temel veri yapıları vardır. Veri yapısı, aslında, ham olarak 1 ve 0'lardan oluşan verinin yorumlanmasını belirleyen biçimleme (formatting) düzenidir. Örneğin, 62 sayısının ikili tabandaki karşılığı, 111110 olarak bellekte saklanır.
- Temel veri yapıları aşağıdaki gibi sınıflanabilir:

- **Karakter** (Character) - A, B,C, @, ?, >, ∞...
- **Tamsayı** (Integer) - 1923, 19, 29, 23, 5, - 153
- **Gerçek Sayı** (Real Number) - 3.14, 2.71, 1.53×10^8 ...
- **Katar** (String) - "Üniversite", "Oğuzhan", "Can", "A"
- **Dizi/Matris** (Array/Matrix) - [23 4 1923], $\begin{bmatrix} 19 & 23 \\ 29 & 5 \end{bmatrix}$

Tanımlamalı Veri Yapıları

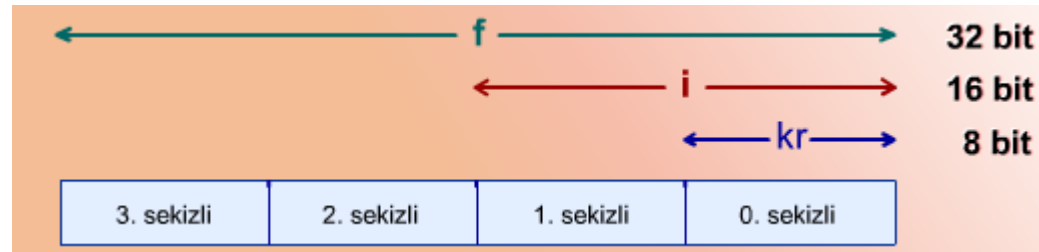
- Tanımlamalı veri yapısı, temel veya daha önceden tanımlanmış veri yapılarının kullanılıp yeni veri yapıları oluşturulmasıdır. Üç değişik şekilde yapılabilir:
 - **Topluluk (Struct) Oluşturma:** Birden çok veri yapısının bir araya getirilip yeni bir veri yapısı ortaya çıkarmaktır. (Java dilinde sınıflar)
 - **Ortaklık (Union) Oluşturma:** Birden çok değişkenin aynı bellek alanını kullanmasını sağlayan veri yapısı tanımlamasıdır. Ortaklıkta en fazla yer işgal eden veri yapısı hangisi ise, ortaklık içerisindeki tüm değişkenler orayı paylaşır.
 - **Bit Düzeyinde Erişim:** Verinin her bir bit'i üzerinde diğerlerinden bağımsız olarak işlem yapılması olanağı sunar.
- Her birinin kullanım amacı farklı farklı olup uygulamaya göre bir tanesi veya hepsi bir arada kullanılabilir. Genel olarak, en çok kullanılanı topluluk oluşturmadır; böylece birden fazla veri yapısı bir araya getirilip/paketlenip yeni bir veri yapısı/türü ortaya çıkarılır.

Tanımlamalı Veri Yapıları

- C dilinde tanımlamalı veri yapılarına örnek aşağıda verilmiştir.

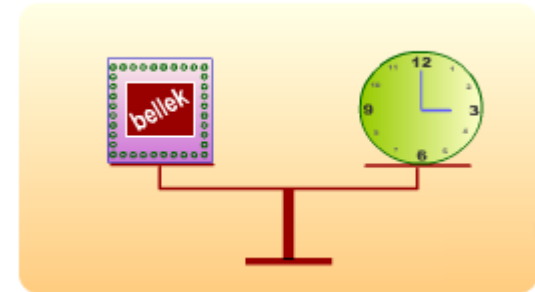
```
struct kimlik {
    char ad[15];
    char soyad[20];
    int yas;
    char adres[50];
};
```

```
union paylas {
    int i;
    flat f;
    char kr;
};_
```



Veri Modelleri

- Veri modelleri, tasarımı yapılacak programın en uygun ve etkin şekilde olmasını sağlar ve daha baştan programın çalışma hızı ve bellek gereksinimi hakkında bilgi verir. Çoğu zaman, programın çalışma hızıyla bellek gereksinimi miktarı doğru orantılıdır denilebilir.
- Veri modeller, genel olarak, aşağıdaki gibi verilebilir:
 - Bağlantılı Liste (Link List)
 - Ağaç (Tree)
 - Graf (Graph)
 - Durum Makinası (State Machine)
 - Veritabanı-İlişkisel (Database Relational)
 - Ağ Bağlantı (Network Connection)



Hız ile Bellek Miktarı arasında denge kurulması

Liste ve Bağlantılı Liste Veri Modeli

- Liste veri modeli, aynı kümeye ait olan verilerin bellekte art arda tutulması ilkesine dayanır. Veriler belirli bir düzen içerisinde (sıralı vs.) olabilir veya olmayabilir; önemli olan tüm verilerin art arda gelen sırada tutulmasıdır.

Ardışıl Erişim

Dinamik Yaklaşım

Arama Maliyeti $O(n)$

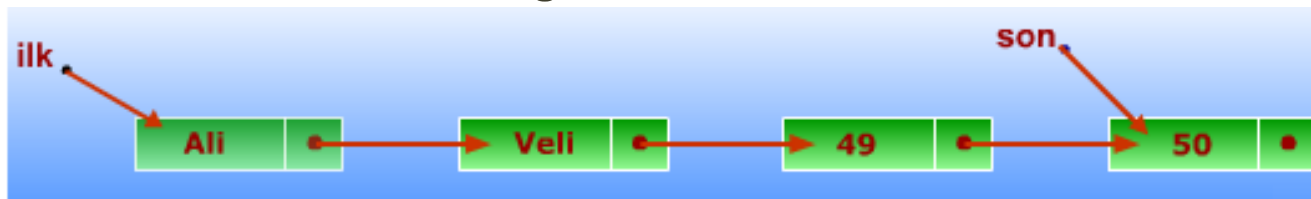
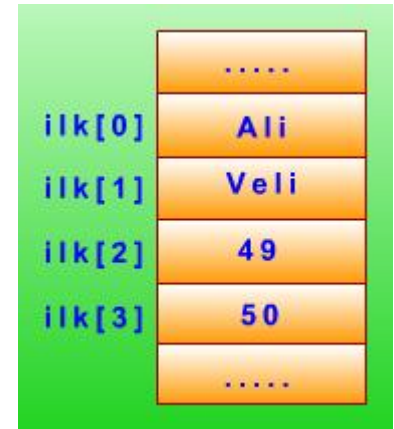
Ekleme Maliyeti $O(1)$

Silme Maliyeti $O(1)$ veya $O(n)$

Esnek Bellek Kullanımı

Liste ve Bağlantılı Liste Veri Modeli

- En yalın liste veri modeli bir boyutlu dizi üzerinde tutulandır. Böylesi bir listeye eleman ekleme işlemi oldukça kolaydır; genel olarak, yeni gelen elemanlar listenin sonuna eklenir. Yalın listede bir sonraki eleman hemen o elemanın işgal ettiği bellek alanından sonradır.
- Bağlantılı liste (link list) ise, elemanların kendi değerlerine ek olarak bir de bağlantı bilgisinin kullanılmasıyla sağlanır; bağlantı bilgisi bir sonraki elemanın adresi niteliğindedir.



Ağaç Veri Modeli

- Ağaç veri modeli, düğümlerden ve dallardan oluşur; düğümlerde verilerin kendileri veya bir kısmı tutulurken, dallar diğer düğümlere olan bağlantı ilişkilerini gösterir. Ağaç veri modeli, özellikle kümenin büyük olduğu ve arama işleminin çok kullanıldığı uygulamalarda etkin bir çözüm sunar.
- En üstteki düğüm kök (root), kendisine alttan hiçbir bağlantının olmadığı düğüm yaprak (leaf), diğerleri de ara düğüm (internal node) olarak adlandırılır. Bir düğüme alttan bağlı düğümlere çocuk (child), üsten bağlı düğüme de o düğümün ailesi (parent) denilir.

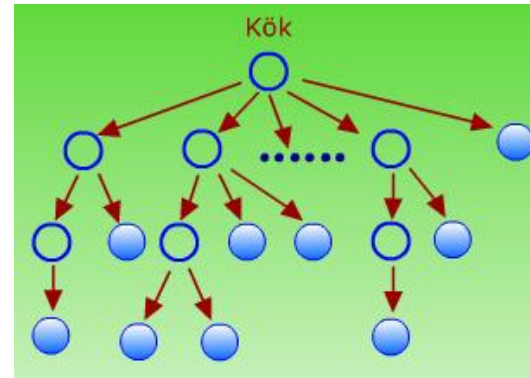
Hızlı Erişim

Esnek Bellek Kullanımı

Arama-Ekleme Maliyetleri

Tasarım Esnekliği

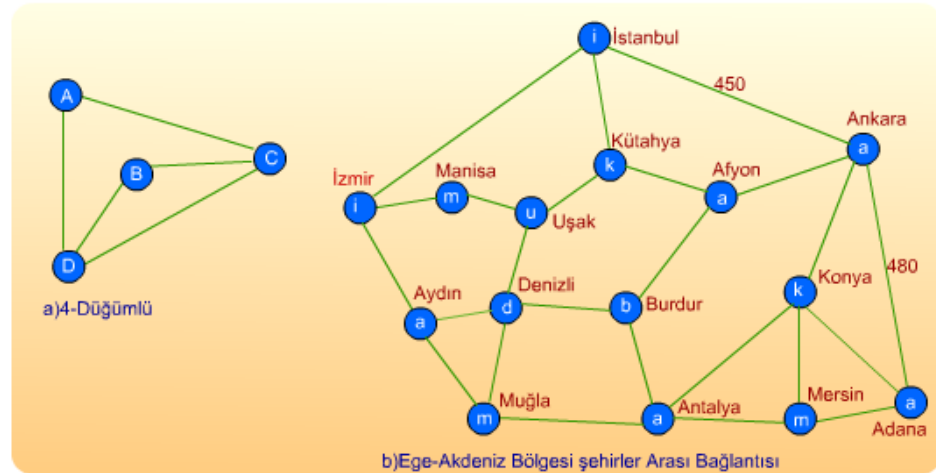
Çok Farklı problemlere Model



Graf Veri Modeli

- Graf veri modeli, aynı kümeye ait olan verilerin şekilde görüldüğü gibi düğümler, ayrıtlar (kenarlar) ve bunların birleştirilmesinden oluşur. Düğümler birleşme noktasını ayrıtlar da düğümlerin bağlantı ilişkisini gösterir. Verilerin kendileri veya bir kısmı hem düğümlerde hem de ayrıtların bilgi kısmında tutulabilir.
- Graflar, yazılım dünyasından önemli bir yere sahiptir. Örneğin, bir şehrin trafik altyapısından en yüksek akışın sağlanması, taşıma şirketinin en verimli taşıma şekli veya network bağlantılarında yüksek başarımlar elde edilmesi gibi problemler.

Modelleme Esnekliği
Üzerine Tasarlanmış Algoritma Çokluğu
En kısa Yol (Shortest Path)
Yol Ağacı (Spanning Tree)
Greedy Yöntemi
DFS ve BFS Yaklaşımları
Kruksal Algoritması



Durum Makinası Veri Modeli

- Durum makinası veri modeli, bir sistemin davranışını tanımlamak ve ortaya çıkarmak için kullanılan bir yaklaşım şeklidir; işletim sistemlerinde, derleyici ve yorumlayıcılarda, kontrol amaçlı yazılımlarda sistemin davranışını durumlara indirger ve durumlar arası geçiş koşullarıyla sistemi ortaya koyar.
- Durum makinası, yazılım uygulamasında birçok alanda kullanılabilir. Örneğin bir robot kolunun hareketi, şifre çözme, gerçek zamanlı işletim sistemlerinde proses kontrolü ve genel olarak kontrol alt sistemlerinin yazılımla uygulamayı başarılı bir şekilde sonuçlandırma durumlarında çözüm olur.

Durum Makinası Veri Modeli

- Durum makinası veri modeli şeklen yönlü graflara benzer, ancak, birleşme noktaları graflarda olduğu gibi düğüm olarak değil de **durum**, ayrıtlar da **geçiş eğrileri** olarak adlandırılır. Durumlar arasındaki geçişler, sistemin o ana kadar ki durumlarına ve giriş parametrelerine bağlıdır.

Davranış Modelleme

Ardaşıl Yaklaşım

Desen Uyuşması

Katar Arama

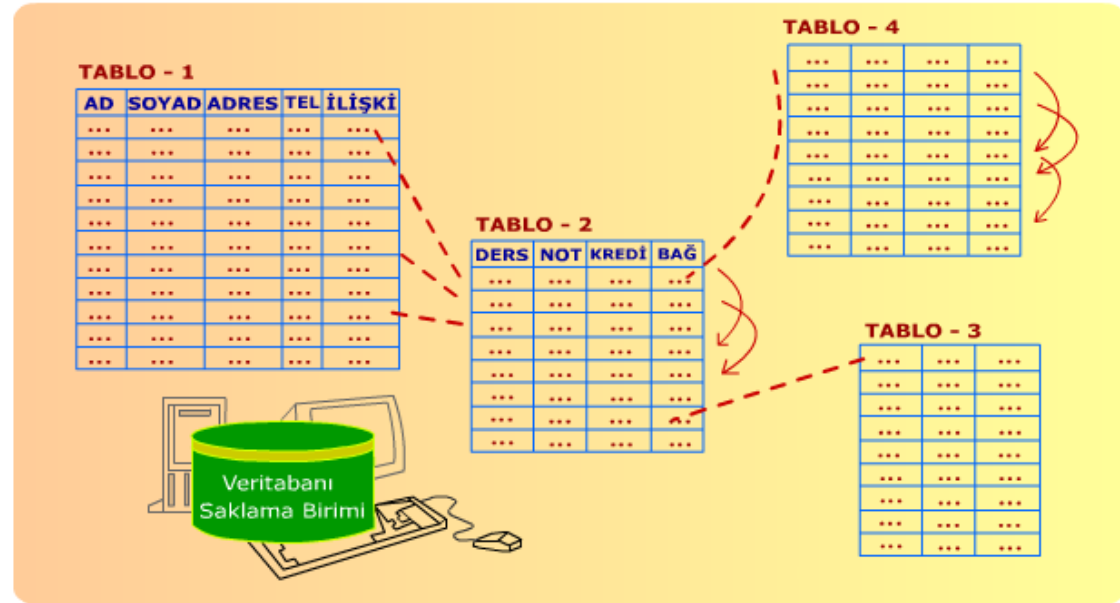
Gramer Çözümü



Veritabanında İlişkisel Veri Modeli

- Veritabanı ilişkisel veri modeli veritabanı uygulamalarında var olan dört beş sınıftan birisidir; veriler şekilde gösterildiği gibi tablolar üzerinden kurulan ilişkilere dayanmaktadır.

Bilgilerin Düzenli Saklanması
Hızlı Arama/Sorulama
Veriler Arasında İlişki Oluşturulması
İnternet Tarayıcısı Ortamında Bilgi Sorgulama
Koruma
Verilerin Arşivlenmesi



Yukarıda tipik olarak veritabanı ilişkisel veri modelindeki tablolar ve alanların durumu görülmektedir.

Veritabanında İlişkisel Veri Modeli

- SQL (Structured Query Language), sorgulama dili kullanılarak veritabanı üzerinde sorgulama yapılabilir. Access, Microsoft SQL, ORACLE, SYBASE, Ingres gibi birçok veritabanı yönetim sistemleri ilişkisel veri modelini desteklemektedir.
- Veritabanı yönetim sistemleri, veritabanı oluşturma, tablo yaratma, alanları tanımlama gibi işlerin başarılı bir şekilde sonuçlandırmasını ve genel olarak veritabanı yönetimini sağlarlar.

Ağ Veri Modeli

- Ağ veri modeli, katmalı ağ mimarilerinde, bilgisayarlar arasında eş katmanlar (peer layers) düzeyinde veri alış-verişini sağlayan dilim (segment), paket (packet) ve çerçeve yapılarını ortaya koyar ve iletişim için gerekli davranışı tanımlar. Veri haberleşmesinde hemen hemen tüm mimariler katmanlı yapıdadır. Tüm mimariler örnek temsil eden OSI 1, başvuru modeli 7, TCP/IP (Transmission Control Protocol / Internet Protocol) protokol kümesi 4 katmanlıdır.




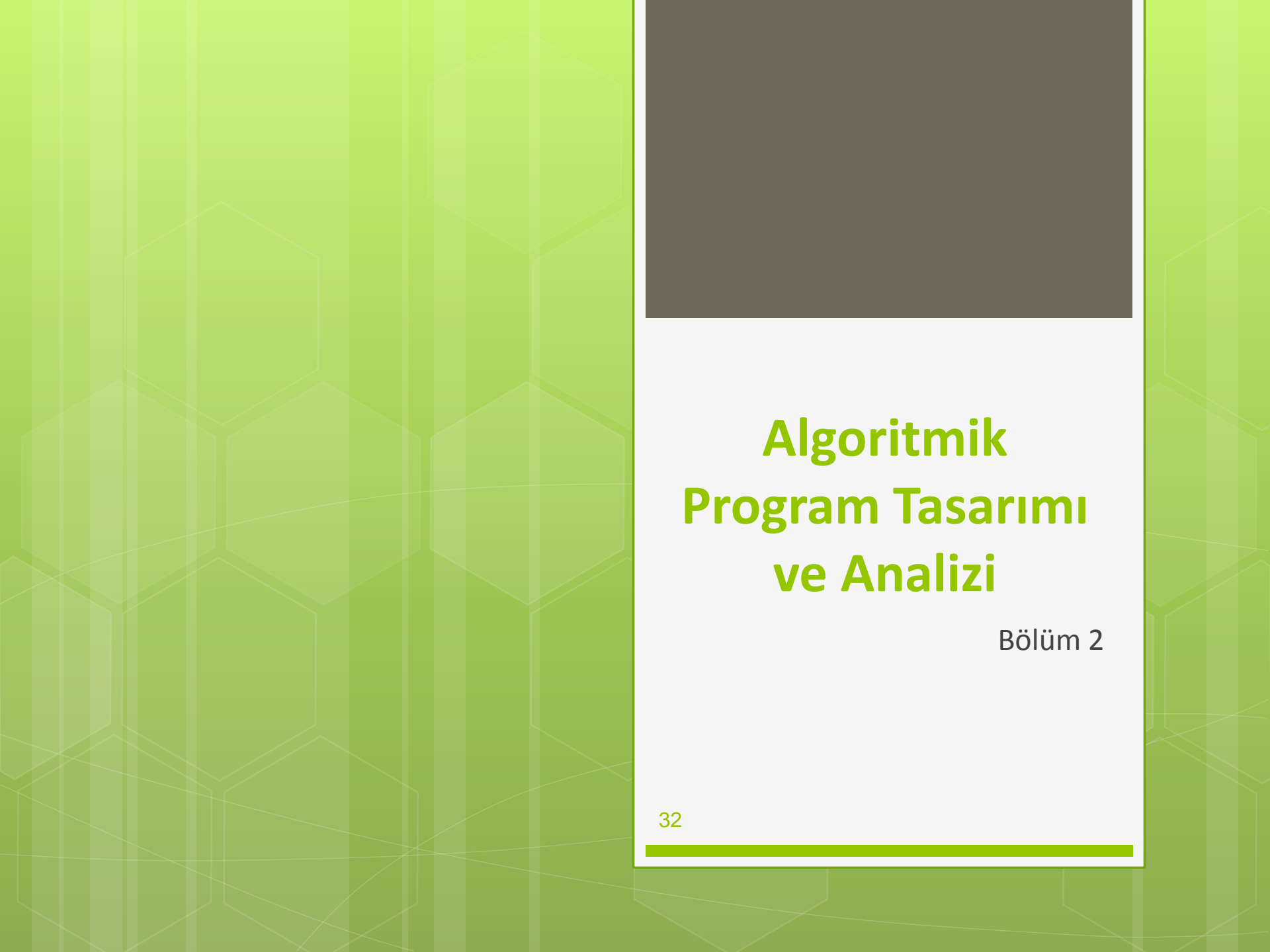
Veri Modelleri

- Bu derste aşağıdaki veri modelleri detaylı ele alınacaktır;
- **Liste**
 - Sonlu sayıda elemandan oluşan ve elemanları **doğrusal sırada** yerleştirilmiş veri modeli. Herhangi bir elemanına erişimde sınırlama yoktur.
- **Yığıt veya Yığın**
 - Elemanlarına erişim sınırlaması olan, liste uyarlı veri modeli (Last In First Out-LIFO listesi).
- **Kuyruk**
 - Elemanlarına erişim sınırlaması olan, liste uyarlı veri modeli. (First In First Out-FIFO listesi).
- **Ağaç**
 - **Doğrusal olmayan** belirli niteliklere sahip veri modeli
- **Çizge (Graph)**
 - Köşe adı verilen düğümleri ve kenar adı verilen köşeleri birbirine bağlayan bağlantılardan oluşan **doğrusal olmayan** veri modeli

Veri Yapısı	Artıları	Eksileri
Dizi	Hızlı ekleme ve çok hızlı erişim(indis biliniyorsa).	Yavaş arama, yavaş silme ve sabit boyut.
Sıralı Dizi	Sıralanmamış diziye göre daha hızlı arama.	Yavaş arama, yavaş silme ve sabit boyut.
Yığın	Son giren, ilk çıkar(last-in, first-out) erişimi sağlar.	Diğer öğelere yavaş erişim.
Kuyruk	İlk giren, ilk çıkar(first-in, first-out) erişimi sağlar.	Diğer öğelere yavaş erişim.
Bağlı Liste	Hızlı ekleme ve silme.	Yavaş arama.
Hash Tablosu	Hızlı ekleme ve anahtar bilindiğinde çok hızlı erişim.	Yavaş silme, anahtar bilinmediğinde yavaş erişim ve verimsiz bellek kullanımı.
Küme(Heap)	Hızlı ekleme ve silme.	Diğer öğelere yavaş erişim. Başta en büyük öğeye erişim.
İkili Ağaç	Hızlı arama, ekleme ve silme(ağaç dengeli kalmışsa).	Silme algoritması karmaşık.
Graf	Gerçek-dünya problemlerini modelleyebilmesi.	Bazı algoritmaları yavaş çalışmakta ve karmaşıklığı yüksek.

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 başvurulanan 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.



Algoritmik Program Tasarımı ve Analizi

Bölüm 2

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 alınacağını, sonucun nasıl ve nereye yazılacağını sözel olarak ifade edilmesi biçiminde de tanımlanabilir.

Algoritmanın Önemli Özellikleri

- Algoritma hazırlanırken, çözüm için yapılması gerekli işlemler, öncelik sıraları göz önünde bulundurularak ayrıntılı bir biçimde tanımlanmalıdırlar.
- Yazılan komutun tek bir anlama gelmesi ve herkes tarafından anlaşılır olması gereklidir.
- Yazılan komutların uygulanabilir olması gereklidir.
- Her algoritmanın sonlanması, çalıştırılan komut sayısının sonsuz olmaması gereklidir.

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 veri yapısına 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 verilmesinden 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 çağırılması: **Metot adı ({değişken listesi})**
- 5. Metotlardan geri dönüş: **return değer**

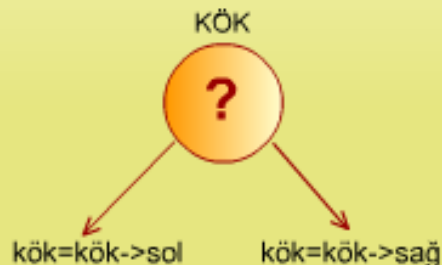
Algoritmaların kaba-kod olarak ifade edilmesi

- 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] toplam: toplam + dizi[i]
 - çarpım:= çarpım* dizi[i]
 - **endfor**

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;
}
```



/ İkili arama ağacına 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

- 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 işlemde kaç adet yapılması gerektiğini bulmaya yarayan bir bağıntı hesaplanır.
- Eğer bu bağıntı zamanla ilgiliyse çalışma hızını, bellek gereksinimiyle ilgiliyse bellek gereksinimi ortaya koyar.

Algoritma Analizi

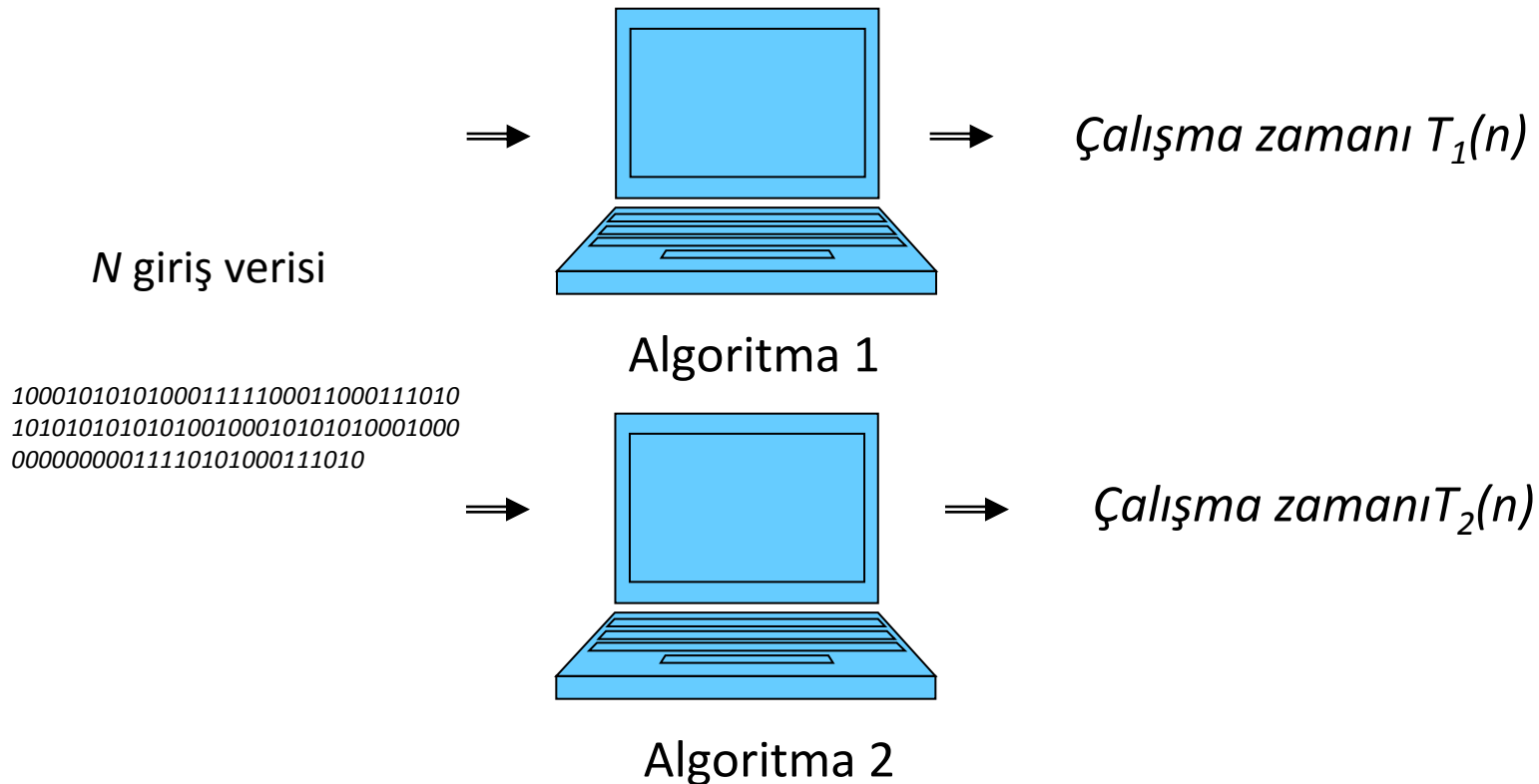
- Neden algoritmayı analiz ederiz?
 - Algoritmanın performansını ölçmek için
 - Farklı algoritmalarla karşılaştırmak için
 - Daha iyisi mümkün mü? Olabileceklerin en iyisi mi?
- Algoritmayı nasıl analiz ederiz?
 - Yürütme zamanı(Running Time)- $T(n)$
 - Karmaşıklık (Complexity) -Notasyonlar

Yürütme Zamanı Analizi (Running Time)

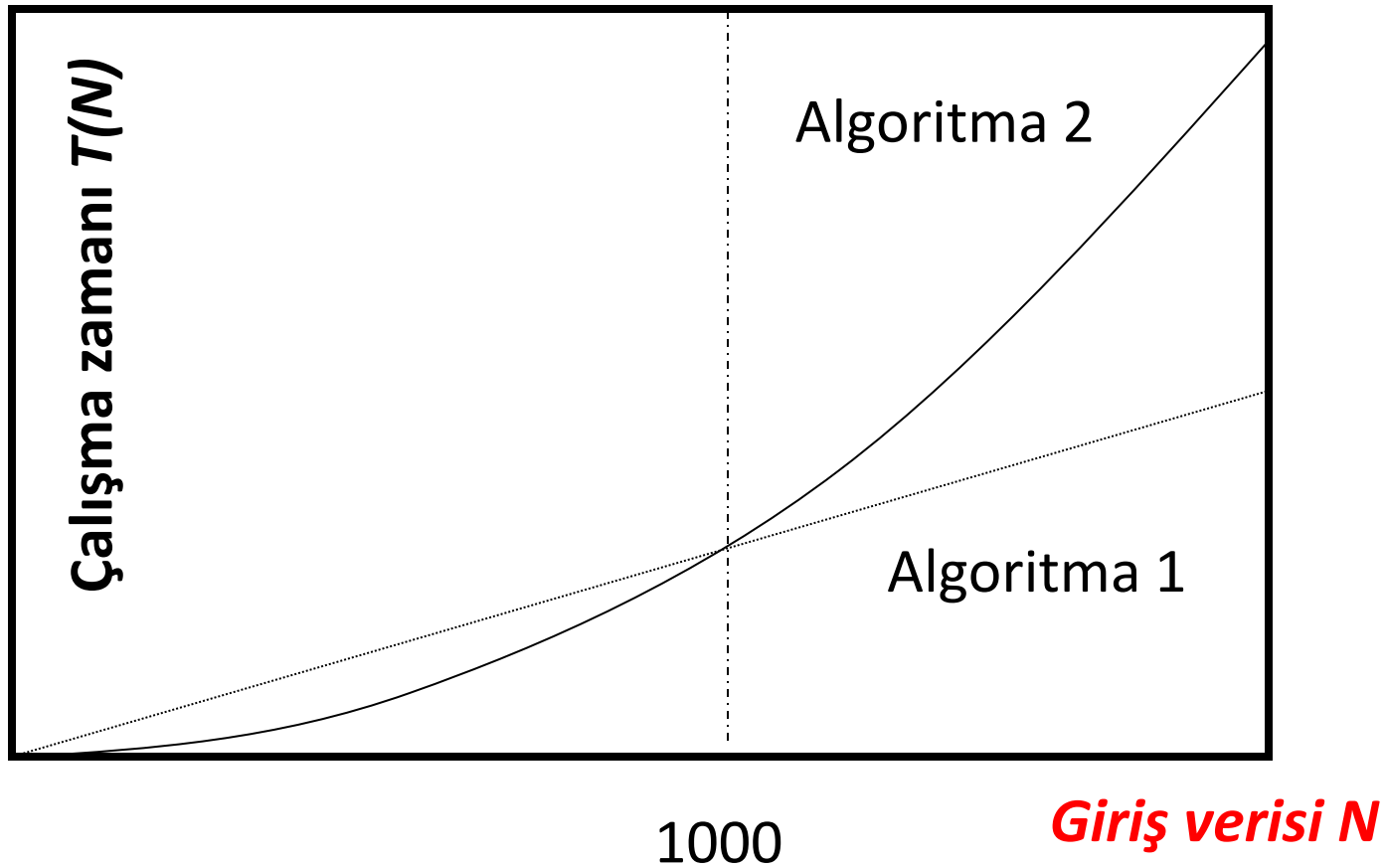
- **Yürütme Zamanı;** Bir programın veya algoritmanın işlevini yerine getirebilmesi için, temel kabul edilen işlevlerden kaç adet yürütülmesini veren bir bağıntıdır ve $T(n)$ ile gösterilir.
- Temel hesap birimi olarak, programlama dilindeki deyimler seçilebildiği gibi döngü sayısı, toplama işlemi sayısı, atama sayısı, dosyaya erişme sayısı gibi işler de temel hesap birimi olarak seçilebilir.

Yürütme Zamanı Analizi

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



Yürütme (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.

Analiz

- Çalışma zamanının kesin olarak belirlenmesi zordur
 - **Giriş verilerine bağlı olan en iyi durum** (best case)
 - **Ortalama durum** (average case); hesaplanması zordur
 - **Diğerlerine göre en kötü durum** (worst case); hesaplanması kolaydır
- Bunun için çeşitli notasyonlardan faydalanılır.

Aritmetik Ortalama için $T(n)$ Hesabı

- **Örnek 1:** Aşağıda bir dizinin aritmetik ortalamasını bulan ve sonucu çağırana gönderen **bulOrta()** fonksiyonun kodu verilmiştir. Bu fonksiyonun yürütme zamanını gösteren **$T(n)$** bağıntısını ayırık C dili deyimlerine göre belirleyiniz.

```
float bulOrta(float A[], int n) {  
    {  
        float ortalama, toplam=0;  
        int k ;  
1-      for(k=0;k<n;k++)  
2-          toplam+=A[k];  
3-      ortalama=toplam/n  
4-      return ortalama;  
    }  
}
```


Aritmetik Ortalama için $T(n)$ Hesabı

o Çözüm 1:

Temel Hesap Birimi	Birim Zaman (Unit Time)	Frekans(Tekrar) (Frequency)	Toplam (Total)
float bulOrta(float A[], int n)	-	-	-
{	-	-	-
float ortalama, toplam=0;	-	-	-
int k ;	-	-	-
1- for(k=0;k<n;k++)	1,1,1	1, (n+1), n	2n+2
2- toplam+=A[k];	1	n	n
3- ortalama=toplam/n	1	1	1
4- return ortalama;	1	1	1
}	-	-	-
			$T(n)=3n+4$

En Küçük Eleman Bulma için $T(n)$ Hesabı

- **Örnek 2:** Aşağıda bir dizi içerisindeki en küçük elemanı bulan **bulEnKucuk()** adlı bir fonksiyonun kodu görülmektedir. Bu fonksiyonun yürütme zamanını gösteren $T(n)$ bağıntısı ayrık C dili deyimlerine göre belirleyiniz.
- ```
float bulEnKucuk(float A[])
{
 float enkucuk;
 int k ;
 1- enkucuk=A[0];
 2- for(k=1;k<n;k++)
 3- if (A[k]<enkucuk)
 4- enkucuk=A[k];
 5- return enkucuk;
}
```

## En Küçük Eleman Bulma için $T(n)$ Hesabı

### o Çözüm 2:

| Temel Hesap Birimi            | Birim Zaman<br>(Unit Time) | Frekans(Tekrar)<br>(Frequency) | Toplam<br>(Total) |
|-------------------------------|----------------------------|--------------------------------|-------------------|
| float bulEnKucuk(float A[]) { | -                          | -                              | -                 |
| float enkucuk;                | -                          | -                              | -                 |
| int k ;                       | -                          | -                              | -                 |
| 1- enkucuk=A[0];              | 1                          | 1                              | 1                 |
| 2- for(k=1;k<n;k++)           | 1,1,1                      | 1, n, (n-1)                    | 2n                |
| 3-       if (A[k]<enkucuk)    | 1                          | n-1                            | n-1               |
| 4-       enkucuk=A[k];        | 1                          | n-1                            | n-1               |
| 5- return enkucuk;            | 1                          | 1                              | 1                 |
| }                             | -                          | -                              | -                 |
|                               | $T(n)=4n$                  |                                |                   |

# Matris Toplama için $T(n)$ Hesabı

| Temel Hesap Birimi             | Birim Zaman<br>(Unit Time) | Frekans(Tekrar)<br>(Frequency) | Toplam<br>(Total) |
|--------------------------------|----------------------------|--------------------------------|-------------------|
| void toplaMatris (A,B,C) {     | -                          | -                              | -                 |
| int A[n][m], B[n][m], C[n][m]; | -                          | -                              | -                 |
| int i,j ;                      | -                          | -                              | -                 |
| 1- for(i=0;i<n;i++)            | 1,1,1                      | 1,(n+1),n                      | 2n+2              |
| 2- for(j=0;j<m;j++)            | 1,1,1                      | n(1,(m+1),m)                   | n(2m+2)           |
| 3- C[i][j]=A[i][j]+B[i][j];    | 1                          | nm                             | nm                |
| }                              | -                          | -                              | -                 |
|                                | $T(n,m)=3nm+4n+2$          |                                |                   |
| n=m ise                        | $T(n)=3n^2+4n+2$           |                                |                   |

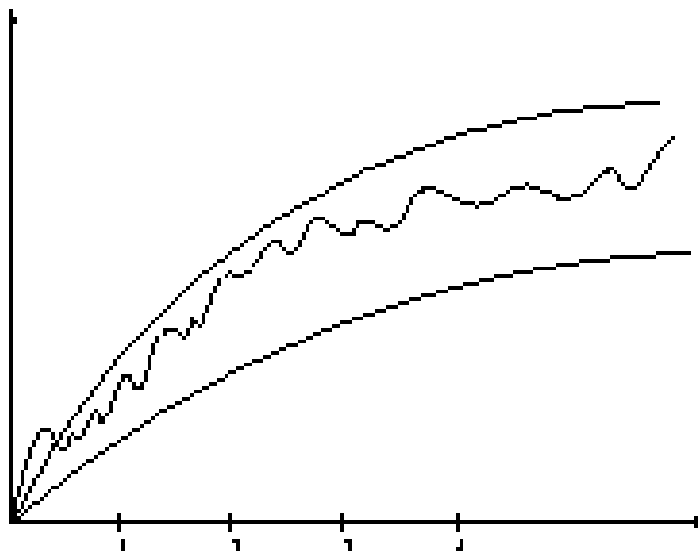
$$T(N) = \sum_{i=1}^N \sum_{j=1}^N 1 = \sum_{i=1}^N N = N * N = N^2$$

# Karmaşıklık (Complexity)

- 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 (Complexity)

- Strateji: Alt ve üst limitlerin bulunması



Üst limit –  $O(n)$

Algoritmanın gerçek fonksiyonu

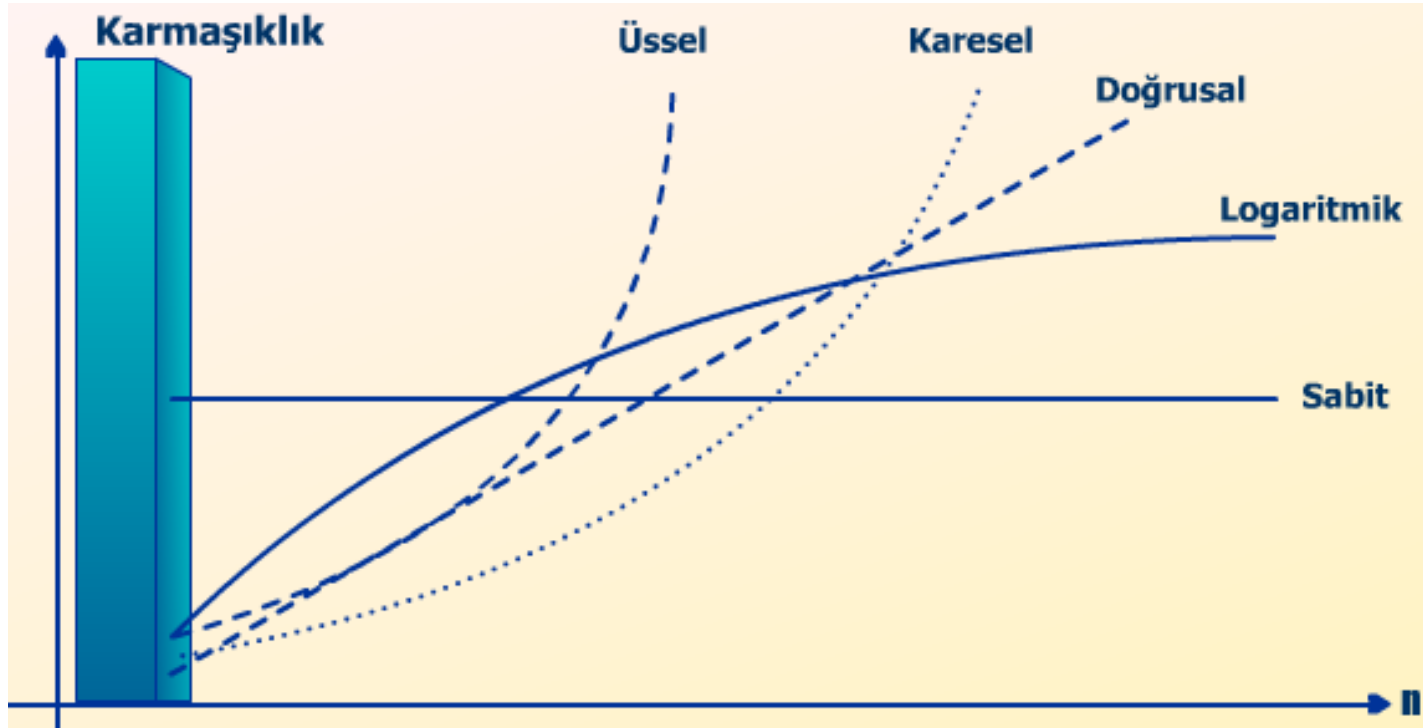
Alt limit-  $\Omega(n)$

# Karşılaşılan Genel Fonksiyonlar

Maliyet artar

| Büyük O       | 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 ( $N \leq 20$ )                                                                                                                   |
| $O(10^n)$     | On tabanında üssel                                                                                                                                                                         |
| $O(n!)$       | Faktöriyel                                                                                                                                                                                 |

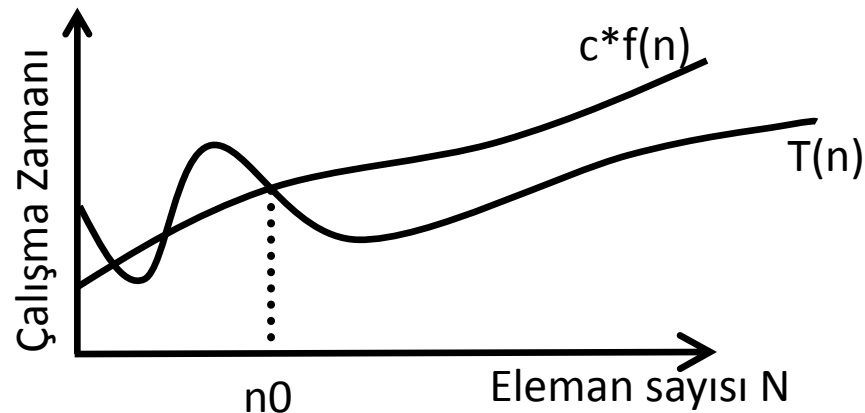
# Karmaşıklık (Complexity)





## Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır (En kötü durum analizi)

- Bir algoritmanın çalışma süresi,
- $T(N)=O(f(n))$  **O**, bir fonksiyon değil, sadece gösterimdir.
- $T(N) \leq c f(n)$  ve  $N \geq n_0$  koşullarını sağlayan  $c$  ve  $n_0$  değerleri varsa  $T(N) \leq c f(n)$  ifadesi doğrudur.
- $f(n)$ ,  $T(N)$ 'in asimptotik üst limiti olarak adlandırılır.
- $T(N)=O(f(n))$



# O Notasyonu- Asimtotik Üst Limit

A algoritması için  $T_A(N) = 1000N$  olarak verilmiştir.  $O$  notasyonu cinsinden bu çalışma süresi  $T_A(N) = 1000N = O(N)$  dir.

Yukarıdaki ifadenin doğruluğunu ispat etmek için  $O$  notasyonunun tanımını kullanacağız. Bu tanıma göre  $1000N = O(N)$  ifadesini ispatlamak için,

$1000N \leq cN \quad \forall N \geq n_0$  eşitsizliğini belirli bir  $c$  ve  $n_0$  sabitleri için ispatlamak gerekir.

$c = 2000$  ve  $n_0 = 1$  seçildiği zaman eşitsizliğin  $n_0$  dan büyük her  $N$  değeri için sağlandığı aşıkardır.

## O Notasyonu- Asimtotik Üst Limit

B algoritması için  $T_B(N) = N^2$  olarak verilmiştir.  $O$  notasyonu cinsinden bu çalışma süresi  $T_B(N) = N^2 = O(N^2)$  dir.

Yukarıdaki ifadenin doğruluğunu ispat etmek için  $O$  notasyonunun tanımını kullanacağız. Bu tanıma göre  $N^2 = O(N^2)$  ifadesini ispatlamak için,

$N^2 \leq cN^2$   $\forall N \geq n_0$  eşitsizliğini belirli bir  $c$  ve  $n_0$  sabitleri için

ispatlamak gerekir.

$c = 1$  ve  $n_0 = 1$  seçildiği zaman eşitsizlik  $n_0$  dan büyük her  $N$  değeri için sağlanmaktadır.

# O Notasyonu- Asimtotik Üst Limit

$7n^2 + 5 = O(n)$  ifadesinin doğru olup olmadığını ispatlayın.

O notasyonuna göre

$7n^2 + 5 \leq cn \quad \forall n \geq n_0$  olması gereklidir. Bu şartı sağlayacak  $c$  ve  $n_0$  değerleri bulabilir miyiz?

Her iki tarafı  $n$  sayısına bölersek;

$7n + \frac{5}{n} \leq c$  elde edilecektir. Buna göre eşitliğin sağlanabilmesi için  $n$  sayısı değiştikçe  $c$

de değişmelidir. Sabit bir  $c$  ve  $n_0$  çifti yoktur; dolayısıyla eşitsizlik doğru değildir.

# O Notasyonu

$7n^2 + 5 = O(n^2)$  ifadesinin doğruluğunu ispatlayın.

O notasyonu tanımına göre,

$7n^2 + 5 \leq cn^2 \quad \forall n \geq n_0$  ifadesini sağlayan bir  $c$  ve  $n_0$  değeri var mı?

$c = 12 \quad n_0 = 1$  veya

$c = 8 \quad n_0 = 5$  değerleri eşitsizliği  $n_0$  dan büyük her  $n$  değeri için sağlamaktadır.

Not: Çözüm kümesini sağlayan kaç tane  $c$  ve  $n$  çifti olduğu önemli değildir. Tek bir çift olması notasyonun doğruluğunu ispatlamak için yeterlidir.

## Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır

- Örnek:  $T(n) = 2n+5$  is  $O(n^2)$  Neden?
  - $n \geq n_0$  şartını sağlayan tüm sayılar için  $T(n) = 2n+5 \leq c \cdot n^2$  şartını sağlayan  $c$  ve  $n_0$  değerlerini arıyoruz.
  - $n \geq 4$  için  $2n+5 \leq 1 \cdot n^2$ 
    - $c = 1, n_0 = 4$
  - $n \geq 3$  için  $2n+5 \leq 2 \cdot n^2$ 
    - $c = 2, n_0 = 3$
  - Diğer  $c$  ve  $n_0$  değerleri de bulunabilir.

## Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır

- Örnek:  $T(n) = n(n+1)/2 \rightarrow O(?)$ 
  - $T(n) = n^2/2 + n/2 \rightarrow O(N^2)$ . Neden?
  - $n \geq 1$  iken  $n^2/2 + n/2 \leq n^2/2 + n^2/2 \leq n^2$
  - Böylece,  $T(n)=n*(n+1)/2 \leq 1 * n^2$  for all  $n \geq 1$ 
    - $c=1, n_0=1$
- Not:  $T(n)$  ayrıca  $O(n^3)$  tür.

# O Notasyonunun Önemi

- İki algoritma karşılaştırılırken zaman mertebesinden konuşulur. Mertebesi büyük olanın daha yavaş olduğu kolaylıkla anlaşılır.
- Makineler arasındaki fark, katsayılar olarak düşünüldüğünde  $O(7n^2)$  yerine  $O(n^2)$  ifadesi geçerli olur. Dolayısıyla sabitler ihmal edilebilir ve birimlerden kurtulur



# O Notasyonu

- O notasyonunda yazarken en basit şekilde yazarız.
  - Örneğin
    - $3n^2+2n+5 = O(n^2)$
  - Aşağıdaki gösterimlerde doğrudur fakat kullanılmaz.
    - $3n^2+2n+5 = O(3n^2+2n+5)$
    - $3n^2+2n+5 = O(n^2+n)$
    - $3n^2+2n+5 = O(3n^2)$

## O Notasyonu-Örnek 1

- $3n^2+2n+5 = O(n^2)$  ifadesinin doğru olup olmadığını ispatlayınız.

$$\begin{aligned} 10n^2 &= 3n^2 + 2n^2 + 5n^2 \\ &\geq 3n^2 + 2n + 5 \text{ için } n \geq 1 \\ c &= 10, n_0 = 1 \end{aligned}$$

Çözüm kümesini sağlayan kaç tane  $n_0$  ve  $c$  çifti olduğu önemli değildir. Tek bir çift olması notasyonun doğruluğu için yeterlidir.

## O Notasyonu-Örnek 2

- $T(N)=O(7n^2+5n+4)$  olarak ifade edilebiliyorsa,  $T(N)$  fonksiyonu aşağıdakilerden herhangi biri olabilir.
- $T(N)=n^2$
- $T(N)=4n+7$
- $T(N)=1000n^2+2n+300$
- $T(N)= O(7n^2+5n+4) =O(n^2)$

## O notasyonu- Örnek 3

- Fonksiyonların harcadıkları zamanları O notasyonuna göre yazınız.

- $f_1(n) = 10n + 25n^2$

- $f_2(n) = 20n \log n + 5n$

- $f_3(n) = 12n \log n + 0.05n^2$

- $f_4(n) = n^{1/2} + 3n \log n$

- $O(n^2)$

- $O(n \log n)$

- $O(n^2)$

- $O(n \log n)$

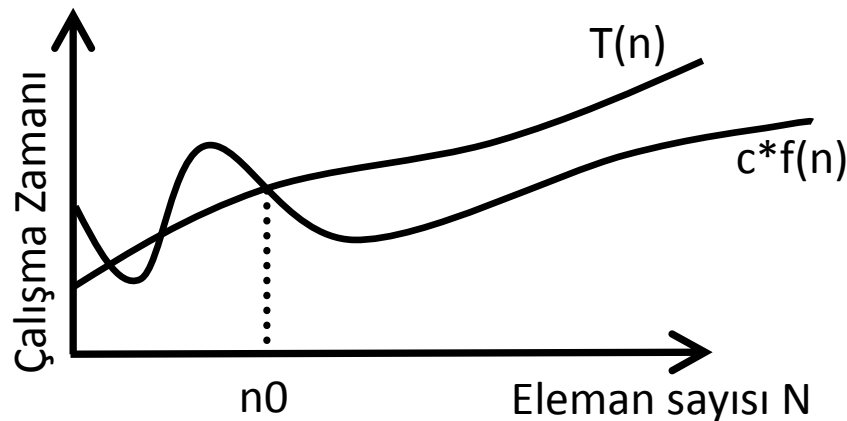
- $2n^2 = O(n^3)$ :  $2n^2 \leq cn^3 \Rightarrow 2 \leq cn \Rightarrow c = 1$  and  $n_0 = 2$
- $n^2 = O(n^2)$ :  $n^2 \leq cn^2 \Rightarrow c \geq 1 \Rightarrow c = 1$  and  $n_0 = 1$
- $1000n^2 + 1000n \neq O(n^2)$ : Şart doğrumu

- $n = O(n^2)$ :

$$n \leq cn^2 \Rightarrow cn \geq 1 \Rightarrow c = 1 \text{ and } n_0 = 1$$

# $\Omega$ Notasyonu- Asimtotik Alt Limit (En iyi durum analizi)

- O notasyonun tam tersidir.
- Her durumda  $T(N) \geq c f(n)$  ve  $N \geq n_0$  koşullarını sağlayan pozitif, sabit  $c$  ve  $n_0$  değerleri bulunabiliyorsa  $T(N)=\Omega(f(n))$  ifadesi doğrudur.
- $f(n)$ ,  $T(N)$ 'in asimtotik alt limiti olarak adlandırılır.



## $\Omega$ notasyonu-Örnek

- $T(n) = 2n + 5 \rightarrow \Omega(n)$ . Neden?
  - $2n+5 \geq 2n$ , tüm  $n \geq 1$  için
- $T(n) = 5*n^2 - 3*n \rightarrow \Omega(n^2)$ . Neden?
  - $5*n^2 - 3*n \geq 4*n^2$ , tüm  $n \geq 4$  için

- $7n^2+3n+5 = O(n^4)$
- $7n^2+3n+5 = O(n^3)$
- $7n^2+3n+5 = O(n^2)$
- $7n^2+3n+5 = \Omega(n^2)$
- $7n^2+3n+5 = \Omega(n)$
- $7n^2+3n+5 = \Omega(1)$

## Examples

- $5n^2 = \Omega(n)$

$\exists c, n_0$  such that:  $0 \leq cn \leq 5n^2 \Rightarrow cn \leq 5n^2 \Rightarrow c = 1$  and  $n_0 = 1$

- $100n + 5 \neq \Omega(n^2)$

$\exists c, n_0$  such that:  $0 \leq cn^2 \leq 100n + 5$

$$100n + 5 \leq 100n + 5n \quad (\forall n \geq 1) = 105n$$

$$cn^2 \leq 105n \Rightarrow n(cn - 105) \leq 0$$

Since  $n$  is positive  $\Rightarrow cn - 105 \leq 0 \Rightarrow n \leq 105/c$

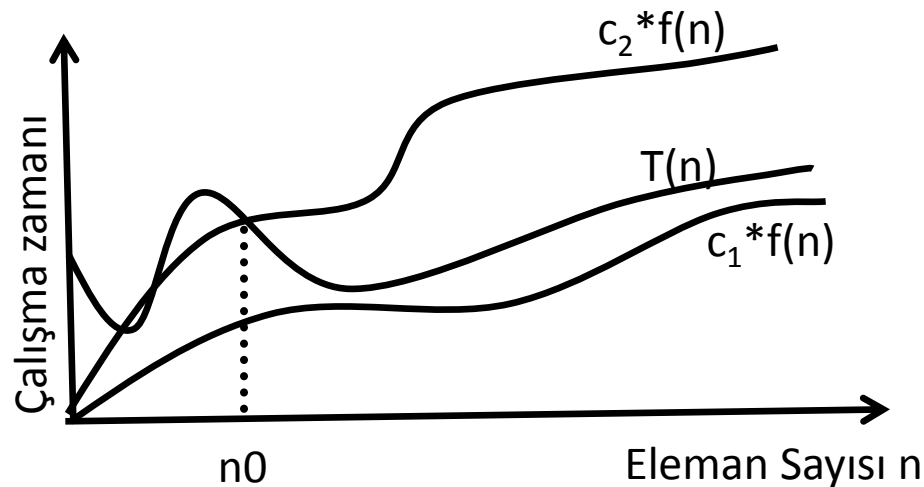
$\Rightarrow$  contradiction:  $n$  cannot be smaller than a constant

- $n = \Omega(2n), n^3 = \Omega(n^2), n = \Omega(\log n)$



## ⊕ Notasyonu (Ortalama durum analizi)

- Her durumda  $c_1 f(n) \geq T(n) \geq c_2 f(n)$  ve  $n \geq n_0$  koşullarını sağlayan pozitif, sabit  $c_1, c_2$  ve  $n_0$  değerleri bulunabiliyorsa  $T(n) = \Theta(f(n))$  ifadesi doğrudur.



## $\Theta$ notasyonu- Örnek

- $T(n) = 2n + 5 \rightarrow \Theta(n)$ . Neden?  
 $2n \leq 2n+5 \leq 3n$ , tüm  $n \geq 5$  için
- $T(n) = 5*n^2 - 3*n \rightarrow \Theta(n^2)$ . Neden?
  - $4*n^2 \leq 5*n^2 - 3*n \leq 5*n^2$ , tüm  $n \geq 4$  için

## Examples

- $n^2/2 - n/2 = \Theta(n^2)$

- $\frac{1}{2} n^2 - \frac{1}{2} n \leq \frac{1}{2} n^2 \quad \forall n \geq 0 \quad \Rightarrow \quad c_2 = \frac{1}{2}$

- $\frac{1}{2} n^2 - \frac{1}{2} n \geq \frac{1}{2} n^2 - \frac{1}{2} n * \frac{1}{2} n \quad ( \forall n \geq 2 ) = \frac{1}{4} n^2 \Rightarrow \quad c_1 = \frac{1}{4}$

- $n \neq \Theta(n^2): c_1 n^2 \leq n \leq c_2 n^2 \Rightarrow \text{only holds for: } n \leq 1/c_1$

- $6n^3 \neq \Theta(n^2): c_1 n^2 \leq 6n^3 \leq c_2 n^2 \Rightarrow \text{only holds for:}$

$$n \leq c_2 / 6$$

N değerini keyfi olarak belirlemek imkansızdır. Çünkü  $c_2$  sabittir.

## Another example

- Prove that  $\frac{1}{2}n^2 - 3n = \Theta(n^2)$

- Determine  $c_1$ ,  $c_2$  and  $n_0$  such that

$$c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$\frac{1}{2} - \frac{3}{n} \leq c_2 \rightarrow n \geq 1, c_2 \geq \frac{1}{2}$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \rightarrow n \geq 7, c_1 \leq \frac{1}{14}$$

- $c_1 = 1/14$ ,  $c_2 = 1/2$ ,  $n_0 = 7$

For any polynomial  $p(n) = \sum_{i=0}^d a_i n^i$   
 $p(n) = \Theta(n^d)$

# Büyük-Oh, Theta, Omega

- İpucu:
- $O(f(N))$  düşünürsek  $f(N)$  ile “eşit veya küçük”
  - Üstten sınır:  $f(N)$  ile “yavaş veya aynı hızda büyür”
- $\Omega(f(N))$  düşünürsek  $f(N)$  ile “eşit veya büyük”
  - Altan sınır:  $f(N)$  ile “aynı hızda veya hızlı büyür”
- $\Theta(f(N))$  düşünürsek  $f(N)$  ile “eşit”
  - Altan ve Üsten sınır : büyüme oranları eşit
- ( $N$ 'nin büyük olduğu ve sabitlerin elendiği durumlarda)

## Sıkça Yapılan Hatalar

- Karmaşıklığı bulmak için sadece döngüleri saymakla yetinmeyin.
  - 2 iç içe döngünün 1 den  $N^2$  kadar döndüğünü düşünürsek karmaşıklık  $O(N^4)$  olur.
- $O(2N^2)$  veya  $O(N^2+N)$  gibi ifadeler kullanmayın.
  - Sadece baskın terim kullanılır.
  - Öndeki sabitler kaldırılır.
- İç içe döngüler karmaşıklığı direk etkilerken art arda gelen döngüler karmaşıklığı etkilemez.

## Bazı Matematiksel İfadeler

$$S(N) = 1 + 2 + 3 + 4 + \dots N = \sum_{i=1}^N i = \frac{N(N+1)}{2}$$

$$\text{Karelerin Toplamı: } \sum_{i=1}^N i^2 = \frac{N * (N+1) * (2n+1)}{6} \approx \frac{N^3}{3}$$

$$\text{Geometrik Seriler: } \sum_{i=0}^N A^i = \frac{A^{N+1} - 1}{A - 1} \quad A > 1$$

$$\sum_{i=0}^N A^i = \frac{1 - A^{N+1}}{1 - A} = \Theta(1) \quad A < 1$$

## Bazı Matematiksel İfadeler

Lineer Geometrik

seriler: 
$$\sum_{i=0}^n ix^i = x + 2x^2 + 3x^3 + \dots + nx^n = \frac{(n-1)x^{(n+1)} - nx^n + x}{(x-1)^2}$$

Harmonik seriler: 
$$H_n = \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \frac{1}{3} + \dots + \frac{1}{n} = (\ln n) + O(1)$$

$$\log A^B = B * \log A$$

Logaritma: 
$$\log(A * B) = \log A + \log B$$

$$\log\left(\frac{A}{B}\right) = \log A - \log B$$



## Bazı Matematiksel İfadeler

- İki sınır arasındaki sayıların toplamı:

$$\sum_{i=a}^b f(i) = \sum_{i=0}^b f(i) - \sum_{i=0}^{a-1} f(i)$$

$$\sum_{i=1}^n (4i^2 - 6i) = 4 \sum_{i=1}^n i^2 - 6 \sum_{i=1}^n i$$

## En iyi zaman ( $T_{best}$ )

- Bir algoritma için, yürütme zamanı, maliyet veya karmaşıklık hesaplamalarında en iyi sonucun elde edildiği duruma “en iyi zaman” denir.
- Örneğin bir dizide yapılan aramanın en iyi durumu, aranan elemanın dizinin ilk elemanı olmasıdır.

## En iyi zaman (Tbest)

|                                                                                         | Birim Zaman<br>(Unit Time)                             | Frekans<br>(Frequency) | Toplam<br>(Total) |
|-----------------------------------------------------------------------------------------|--------------------------------------------------------|------------------------|-------------------|
| <b>i=1</b>                                                                              | <b>1</b>                                               | <b>1</b>               | <b>1</b>          |
| <b>while (a[i] ≠key) and (i ≤ N)</b>                                                    | <b>2</b>                                               | <b>1</b>               | <b>2</b>          |
| <b>i++</b>                                                                              | <b>1</b>                                               | <b>0</b>               | <b>0</b>          |
| <b>if (a[i]=key) print found(veya return i)<br/>else print not found(veya return 0)</b> | <b>2</b>                                               | <b>1</b>               | <b>2</b>          |
|                                                                                         | <b><math>T_{\text{best}}(N) = 5 = \Theta(1)</math></b> |                        |                   |

## En kötü zaman (Tworst)

- En kötü zaman, tüm olumsuz koşulların oluşması durumunda algoritmanın çözüm üretmesi için gerekli hesaplama zamanıdır.
- Örneğin bir dizi içinde arama yapılması durumunda en kötü durum aranan elemanın dizide olmamasıdır. Çünkü aranan elemanın dizide olmadığına anlaşılması için bütün elemanlara tek tek bakılması gerekir.

## En kötü zaman (Tworst)

|                                                                                         | Birim Zaman<br>(Unit Time)                                                   | Frekans<br>(Frequency) | Toplam<br>(Total) |
|-----------------------------------------------------------------------------------------|------------------------------------------------------------------------------|------------------------|-------------------|
| <b>i=1</b>                                                                              | <b>1</b>                                                                     | <b>1</b>               | <b>1</b>          |
| <b>while (a[i] ≠key) and (i≤ N)</b>                                                     | <b>2</b>                                                                     | <b>N+1</b>             | <b>2N+2</b>       |
| <b>i++</b>                                                                              | <b>1</b>                                                                     | <b>N</b>               | <b>N</b>          |
| <b>if (a[i]=key) print found(veya return i)<br/>else print not found(veya return 0)</b> | <b>2</b>                                                                     | <b>1</b>               | <b>2</b>          |
|                                                                                         | $T_{\text{worst}}(N) = 3N+5 = \Theta(N)$ $T(N) \neq \Theta(N)$ $T(N) = O(N)$ |                        |                   |

## Ortalama zaman (Taverage)

- Ortalama zaman, giriş parametrelerin en iyi ve en kötü durum arasında gelmesi ile ortaya çıkan durumda harcanan zamandır.
- Bu işletim süresi, her girdi boyutundaki tüm girdilerin ortalamasıdır.  $n$  elemanın her birinin aranma olasılığının eşit olduğu varsayıldığında ve liste dışından bir eleman aranmayacağı varsayıldığında ortalama işletim süresi  $(n+1)/2$ 'dir. İkinci varsayım kaldırıldığında ortalama işletim süresi  $[(n+1)/2, n]$  aralığındadır (aranan elemanların listede olma eğilimine bağlı olarak). Ortalama durum analizi basit varsayımlar yapıldığında bile zordur ve varsayımlar da gerçek performansın iyi tahmin edilememesine neden olabilir.

## Ortalama zaman (Taverage)

|                                                                                         | Birim Zaman<br>(Unit Time) | Frekans<br>(Frequency) | Toplam<br>(Total) |
|-----------------------------------------------------------------------------------------|----------------------------|------------------------|-------------------|
| <b>i=1</b>                                                                              | <b>1</b>                   | <b>1</b>               | <b>1</b>          |
| <b>while (a[i] ≠key) and (i ≤ N)</b>                                                    | <b>2</b>                   | <b>k+1</b>             | <b>2k+2</b>       |
| <b>i++</b>                                                                              | <b>1</b>                   | <b>k</b>               | <b>k</b>          |
| <b>if (a[i]=key) print found(veya return i)<br/>else print not found(veya return 0)</b> | <b>2</b>                   | <b>1</b>               | <b>2</b>          |
|                                                                                         | <b>=3k+5</b>               |                        |                   |

## Ortalama zaman (Taverage)

$$T_{average}(N) = \sum_{k=0}^{N-1} \frac{1}{2N} (3k + 5) + \frac{1}{2} (3N + 5)$$

$$T_{average}(N) = \frac{1}{2N} 5 + \frac{1}{2N} (3 + 5) + \frac{1}{2N} (3 \cdot 2 + 5) + \dots + \frac{1}{2N} (3(N-1) + 5) + \frac{1}{2} (3N + 5)$$

$$T_{average}(N) = \frac{1}{2N} \left( 3 \frac{N(N-1)}{2} + 5N \right) + \frac{3}{2} N + \frac{5}{2}$$

$$T_{average}(N) = \frac{3}{4} N - \frac{3}{4} + \frac{5}{2} + \frac{3}{2} N + \frac{5}{2} \cong 2N + 4 = \theta(N)$$



# Algoritma Analizinde Bazı Kurallar

- **For Döngüsü:**
- Bir **For** döngüsü için yürütme zamanı en çok **For** döngüsünün içindeki (test dahil) deyimlerin yinelenme sayısı kadardır.
- **İç içe döngüler (Nested Loops)**
- İç içe döngülerde grubunun içindeki deyimın toplam yürütme zamanı, deyimlerin yürütme sürelerinin bütün **For** döngülerinin boyutlarının çarpımı kadardır. Bu durumda analiz içten dışa doğru yapılır.

$$T(N) = \sum_{i=1}^N \sum_{j=1}^N 1 = \sum_{i=1}^N N = N * N = N^2$$

# Algoritma Analizinde Bazı Kurallar

## For Döngüsü:

```

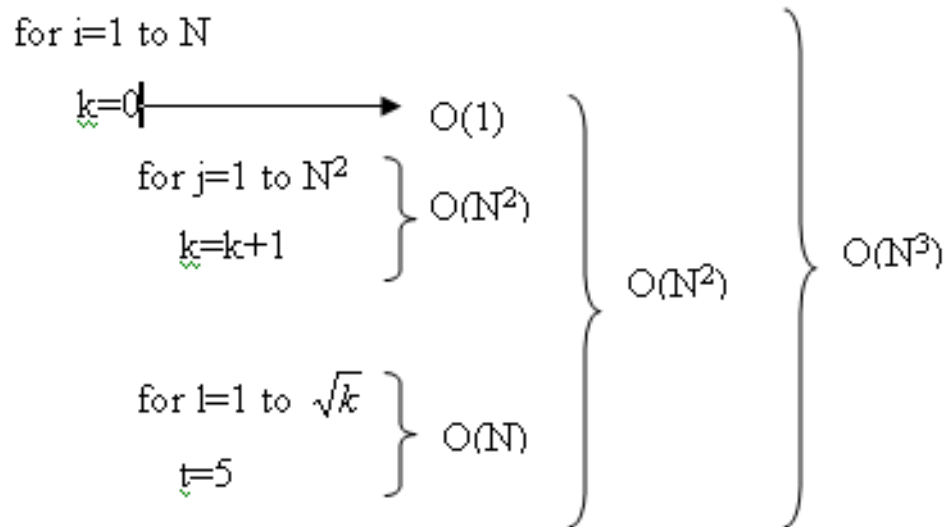
For i=1 to N
 For j=1 to N2
 k=k+1 -> O(1)
 } O(N2)
} O(N3)

```

# Algoritma Analizinde Bazı Kurallar

## For Döngüsü:

Ardışık  
deyimlerin  
toplam yürütme  
zamanını  
bulabilmek için  
sadece toplama  
yapılır.



$$\left. \begin{array}{l} T_1 = O(1) \\ T_2 = O(N^2) \\ T_3 = O(N) \end{array} \right\} T_1 + T_2 + T_3 = O(N^2)$$

# Algoritma Analizinde Bazı Kurallar

If (Durum)

~~~~~  
~~~~~  
~~~~~ }  $T_2(N)$

else

~~~~~  
~~~~~  
~~~~~ }  $T_1(N)$

Durumun maksimum olduğu değerle

durumu hesaplama süresinin toplamıdır.

Dolayısıyla değerlendirme zamanı  $T + \max(T_1(N), T_2(N))$

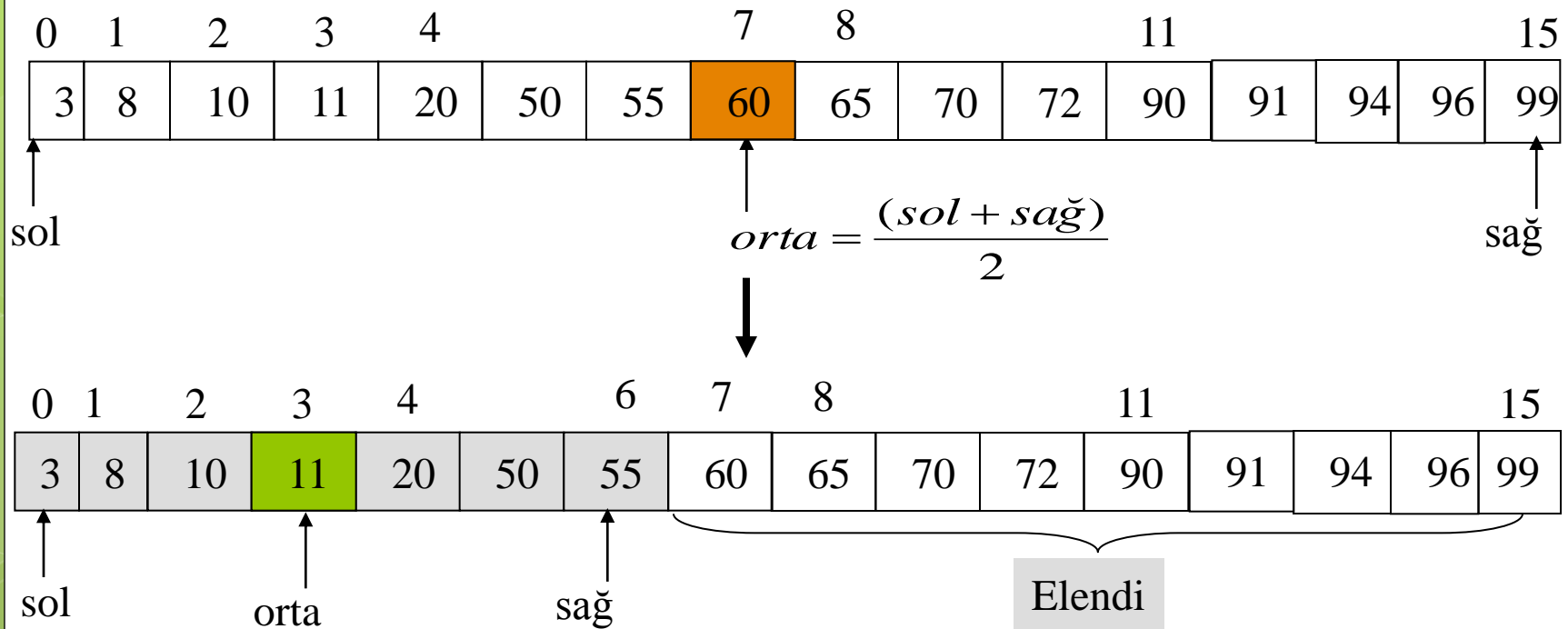
$O(\log N) + \max(O(N), O(N^2)) = O(N^2)$  olur.

# Algoritma Analizinde Bir Örnek: İkili Arama Algoritması

- İkili arama algoritması, sıralı dizilerde kullanılan bir arama metodur. Algoritma iteratif veya tekrarlamalı (recursive) olabilir.
- İterasyon ifadeleri (döngü) belirli bir koşul sağlanana kadar, verilen talimatların çalıştırılmasını sağlar. Belirlenen koşul, “for” döngüsündeki gibi, önceden belirlenebilir veya “while-do” döngüsünde olduğu gibi net olarak belirlenmemiş, uçu açık da olabilir.
- Aşağıda iteratif olarak tasarlanmış ikili arama algoritmasına bir örnek verilmiştir.

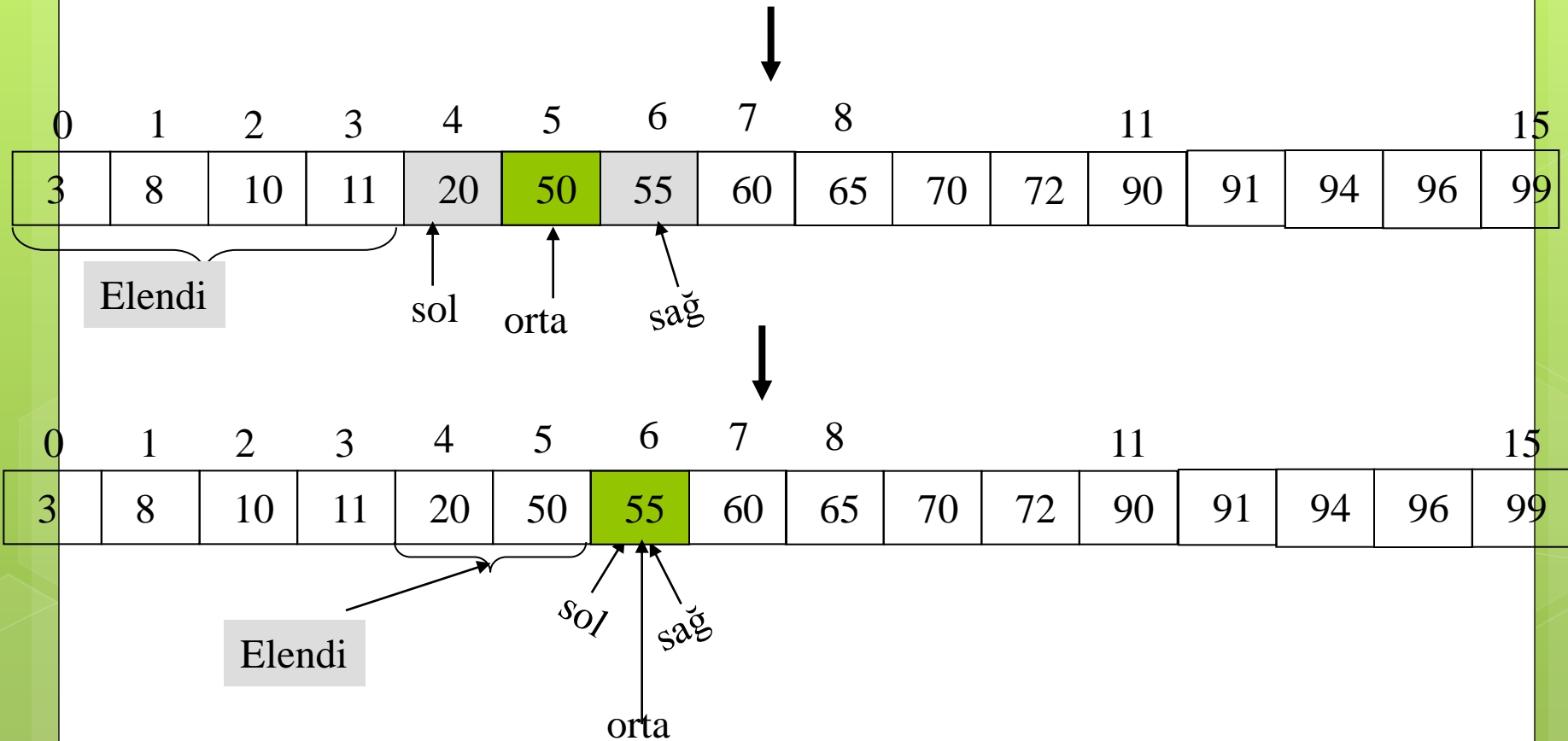
# Örnek : İkili Arama

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



## İkili arama (devam)

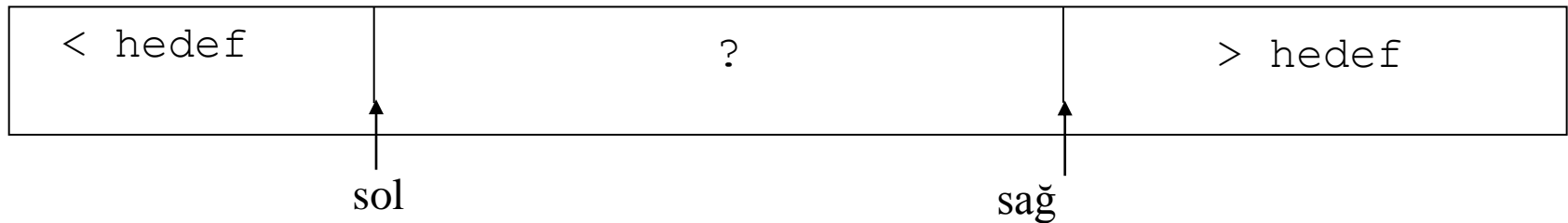
96



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

## İkili Arama (devam)

97



- 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” in sağında kalan alan hedeften büyüktür ve bu alan arama alanından çıkarılır.



# İkili Arama - Algoritma

98

// Aranan sayının indeksini döndürür aranan sayı bulunamazsa -1 döndürür.

**int ikiliArama(int A[], int N, int sayi){**

sol = 0;

sag = N-1;

while (sol <= sag){

int orta = (sol+sag)/2; // Test edilecek sayının indeksi

if (A[orta] == sayi) return orta; // Aranan sayı bulundu. İndeksi döndür

else if (sayi < A[orta]) sag = orta - 1; // Sağ tarafı ele

else sol = orta+1; // Sol tarafı ele

} //bitti-while

return -1; // Aranan sayı bulunamadı

**} //bitti-ikiliArama**

- En kötü çalışma zamanı:  $T(n) = 3 + 5 \cdot \log_2 N$ . Neden?

## Algoritma Analizinde Bir Örnek: İkili Arama Algoritması

```

int binary search(A, key, N)
 low=0, high=N-1 → O(1)
 while(low ≤ high)
 mid=(low+high)/2 → O(1)
 if(A[mid] < key) → O(1)
 low=mid+1
 if(A[mid] > key) → O(1)
 high=mid-1;
 if(A[mid] = key) → O(1)
 return mid
 Return not found → O(1)

```

Complexity Analysis:

- The inner loop body (from `mid=(low+high)/2` to `if(A[mid] = key)`) is grouped by a brace and labeled  $O(1)$ .
- The `while` loop is grouped by a brace and labeled  $O(\log N)$ .
- The entire function is grouped by a brace and labeled  $O(\log N)$ .

# Algoritma Analizinde Bir Örnek: İkili Arama Algoritması

- İlk çalışmada N eleman var.
- 2. de  $\frac{N}{2^1} \rightarrow 3.de \frac{N}{2^3} \rightarrow \dots \frac{N}{2^k} = 1 \rightarrow 0$
- Yukarıdaki ifadeden  $N=2^k$  olduğu anlaşılmaktadır. Bu durumda  $k=\log N$  olarak belirlenir.  $N=1$  için k değeri 0 olacağından,
- $\text{Tworst} = \log N + 1$  olacaktır.

# Örnekler

## 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

İşlem

sayısı

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

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

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

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

$1 \leq L \leq N$

$1 \leq L \leq N$

$0 \leq L \leq N$

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  $\Rightarrow T(n) = 6$
- Ortalama(beklenen) çalışma zamanı nedir?
  - Döngü **N/2 kez** çalıştı  $\Rightarrow T(n) = 3 * n/2 + 3 = 1.5n + 3$
- En kötü çalışma zamanı nedir?
  - Döngü **N kez** çalıştı  $\Rightarrow T(n) = 3n + 3$



## Örnek III: Matris Çarpımı

```
/* İki boyutlu dizi A, B, C. Hesapla C = A*B */
for (i=0; i<N; i++) {
 for (j=0; j<N; j++) {
 C[i][j] = 0;
 for (int k=0; k<N; k++){
 C[i][j] += A[i][k]*B[k][j];
 } //bitti-en içteki for
 } //bitti-içteki for
} //bitti-dıştaki for
```

$$T(N) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left(1 + \sum_{k=0}^{N-1} 1\right) = N^3 + N^2$$