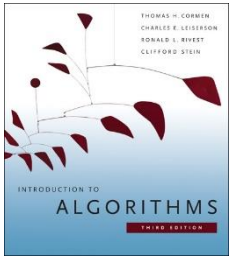
# YMH 317
# Algoritma Analizi

**Prof.Dr. Erkan TANYILDIZI**

# 1.Hafta Algoritmaların Analizi

Algoritma Analizine Giriş

# Ders Kitapları ve Yardımcı Kaynaklar




- Introduction To Algorithms, Third Edition:
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein
- Adnan YAZICI – Sinan KALKAN, ODTÜ
  - Algoritmalar
- M.Ali Akcayol, Gazi Üniversitesi
  - Algoritma Analizi Ders Notları
- Prof.Dr. Ali Karcı (İnönü Ünv.)
  - Algoritma Analizi Ders Notları
- Ayrıca internet üzerinden çok sayıda kaynağa ulaşabilirsiniz.

# Dersin Gereksinimleri

- Bu dersteki öğrencilerin Nesne tabanlı programlama dillerinden birisini (Java, C++ veya C#) ve Veri Yapıları dersini almış olması gerekir.

# Ders İşleme Kuralları

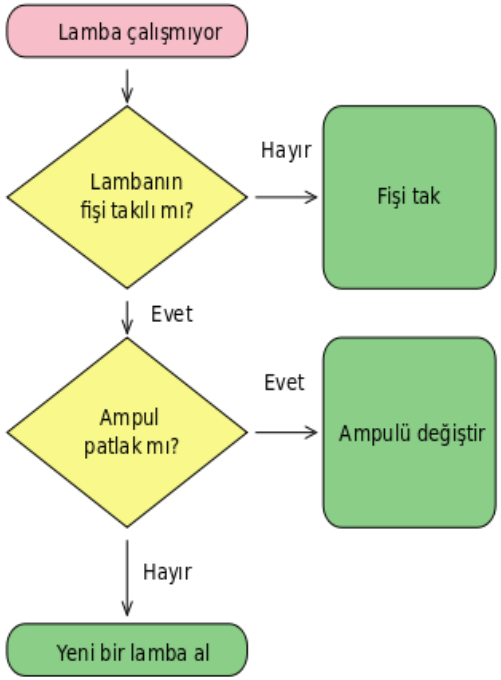
- Derse devam zorunludur. Ders başlangıç saatlerine özen gösteriniz. Derse geç gelen öğrenci ara verilinceye kadar bekleyecektir.
- Her ders iki imza alınacaktır.
- Ödevler zamanında teslim edilecektir. Verilen tarihten sonra getirilen ödevler kabul edilmeyecektir.
- Ders esnasında lütfen kendi aranızda konuşmayın, fısıldaşmayın, mesajlaşmayın v.s. Dersi anlatan ve dinleyen kişilere lütfen saygı gösterin.
- Ders ile ilgili merak ettiğiniz her konuda soru sormaktan çekinmeyin.
- Cep telefonu v.b kişisel taşınabilir iletişim cihazlarınızı ders süresince mutlaka kapalı tutunuz.

# Algoritma

Bölüm 1

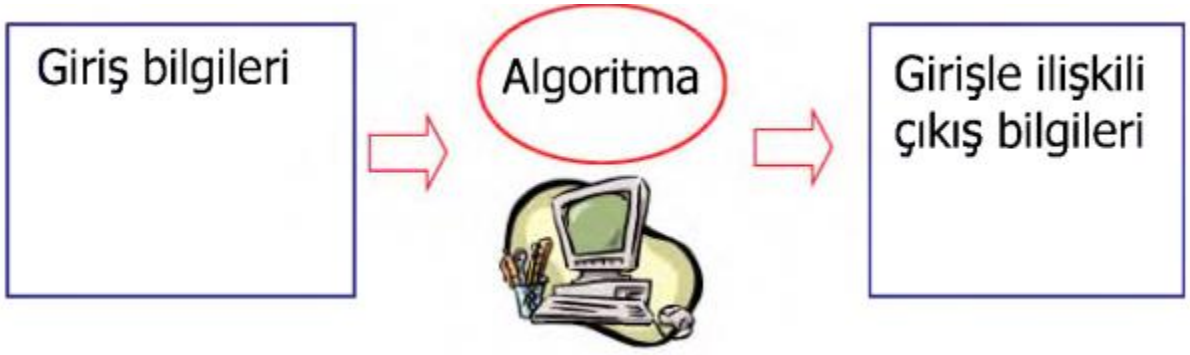
# Algoritma Nedir?

- **Algoritma,**
  - bir problemin çözmek için bir prosedür veya formüldür.
  - problemi çözmek için takip edilmesi gereken yönergeler kümesidir.
  - matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler kümesidir.




- **Program**
  - Bir programlama dilinde algoritmanın gerçekleştirilmesidir.

# Algoritmik çözüm




- Aynı algoritmik problem için farklı algoritmalar olabilir.
- Algoritmaların özellikleri nelerdir?

# 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.
  - **3. Çıkış (Output)**
    - Her algoritmanın bir veya daha fazla çıkış değeri vardır. Çıkış değerleri ile giriş değerleri arasında bağıntılar vardır.
  - **4. Etkililik (Efficiency)**
    - Olabildiğince hızlı çalışmalıdır, olabildiğince az hafıza kullanmalıdır. Bunun anlamı yapılan işlemler yeterince temel işlemler olacak ve sınırlı zaman süresince işleyip bitmelidir.
  - **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.

# Algoritmaların Özellikleri

- Diğer bazı kriterler ise **algoritmanın bilgisayar ortamına aktarılabilme özelliği**, **basitliği**, vb. gibi özelliklerdir.
- Bir problem için birden fazla algoritma verilmişse, bu algoritmalardan en iyisinin seçilmesi gerekir.
- İyi algoritmayı belirlemek için uygulanan testler veya yapılan işlemler **Algoritma Analizi**' nin konusudur.

# Algoritmaların Yönleri

- **1- Algoritmaları Tasarlama**
  - Bulmacaların (puzzel) parçalarını birleştirme,
  - Veri yapılarını seçme,
  - Problemin çözümü için temel yaklaşımlar seçme.
  - En popüler tasarım stratejileri böl ve fethet (divide&conquer), açgözlü(greedy), dinamik programlama, özyineleme (recursive).
- **2- Algoritma ifadesi ve uygulanması**
  - Algoritmayı tasarladıktan sonra sözde kod (pseudocode) ifadesinin belirlenmesi ve problem için uygulanması
  - Bu konudaki endişeler, netlik, özlülük, etkinlik vb.

# Algoritmaların Yönleri

- **3-Algoritma Analizi (Çözümlenmesi)**
  - Algoritma analizi, algoritmayı gerçekte uygulamadan, bir algoritmayı çalıştırabilmek için gereken kaynakların (zaman, yer gibi) araştırılması demektir.

- **4- Çözümünüzün yeterince iyi olup olmadığını görmek için alt ve üst sınırları karşılaştırma**
  - Algoritma analizi problemi çözmek için bize alt ve üst sınırları verir.

# Algoritmaların Yönleri

- **5- Algoritma veya programı doğrulama**
  - Algoritmanın verilen tüm olası girişler için hesaplama yaptığını ve doğru çıkış ürettiğini göstermektir.

- **6- Algoritmaların test edilmesi**
  - Test için iki aşama vardır;
  - **Hata ayıklama (Debugging):** Programın örnek veriler üzerinde çalıştırılması sırasında oluşan hataları tespit etme ve onları düzeltme işlemi.
  - **Profil oluşturma (Profiling):** Çeşitli veriler üzerinde programın çalıştırılması ve sonuçların hesaplaması için gerekli zamanın (ve alan) ölçülmesi işlemi.

# Algoritma Tasarımı

- Algoritmaları tasarlamada kullanılacak yöntemler
  - Özyineleme
  - Böl ve fethet
  - Bilinen probleme indirgeme
  - Dinamik programlama
  - Kaba Seçim veya Haris (Greedy) algoritması
  - Bir veri yapısı icat etme
  - İhtimali (olasılıksal) çözümler
  - Yaklaşım çözümleri

# Algoritmaları tasarlamada kullanılacak yöntemler:

- **Özyineleme**
  - Problemin çözümünün tekrarlı olması durumunda bilinen bir veya birkaç çözümden faydalanarak bir sonraki çözümü elde etme ve elde edilen çözüm ile önceki çözümlerin birkaçının kullanılması ile bir sonraki çözümün elde edilmesi ile problemi çözme işlemine **özyineleme yöntemi** denir.
- **Böl ve fethet**
  - Kompleks problemlerin bir bütün olarak çözülmeleri çok zor olacağından dolayı, bu problemler alt problemlere bölünürler. Bu bölünme işleminin yapılabilmesi için alt problemlerin bir üst seviyedeki problem ile aynı özelliği sağlamalıdır. Bu yöntem ile algoritma tasarımı yapmaya **böl ve fethet yöntemi** denir.
- **Dinamik programlama**
  - Böl ve yönet yöntemine benzer olarak alt problemlerin çözümlerini birleştirerek çözüme gitme mantığına sahip olup alt problem tekrarı varsa, bunlardan bir tanesi çözülür ve bu çözüm diğer tekrarlarda kullanılır. Bu yöntem ile yapılan algoritma tasarım yöntemine **dinamik programlama yöntemi** denir.
- **Kaba Seçim veya Haris (Greedy) algoritması**
  - Optimizasyon problemlerinin çözümü için yerel optimumların seçilmesi ilkesinden yola çıkar ve veriyi belli bir kritere göre düzenledikten sonra ilk veri her zaman optimum çözüme götürür mantığına sahiptir. Temel amaç en iyi sonucu elde etmek için en iyi ara adım çözümlerini seçmeye yönelik bir yöntem olduğundan bu yönteme **haris algoritması yöntemi** denir.

# Algoritmaları tasarlamada kullanılacak yöntemler:

- **Bir veri yapısı icat etme**
  - O ana kadar var olamayan bir veri yapısının icat edilmesi ile problemin çözülmesine **veri yapısı icat etme yöntemi** denir.
- **Bilinen probleme indirgeme**
  - Kompleks olan bir problemin çözümünü yapmak için çözümü bilinen bir veya birden fazla başka probleme dönüştürüp bu şekilde problemi çözme işlemine **bilinen probleme indirgeme yöntemi** denir.
- **İhtimali (olasılıksal) çözümler**
  - Bazı durumlarda gelişigüzellik ilkesi ile etkili bir şekilde problem çözümü yapılabilmektedir. Bunlara örnek olarak Las Vegas polinom-zamanlı ve Monte Carlo polinom-zamanlı algoritmalar verilebilir. Gelişigüzellik kullanılarak yapılan problem çözümlerine **ihtimali çözümler yöntemi** denir.
- **Yaklaşım çözümleri**
  - Çözümü deterministik Turing makinası ile yapılamayan yani karmaşık hesaplamaların belirli bir yöntem ile çözülemediği bu problemlerin bir kısmına bazı kriterler uygulayarak yaklaşım mantığı ile çözüm üretilebilmektedir. Bundan dolayı bu mantık ile yapılan algoritma tasarımına **yaklaşım çözümler yöntemi** adı verilir.

# Algoritma Analizi

- Algoritma analizi, bilgisayar programının performansı (başarım) ve kaynak kullanımı konusunda teorik çalışmalardır.
- Bir başka ifadeyle, 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, farklı çözüm yöntemlerinin verimliğini analiz eder.
- Biz bu derste performans yani başarım üzerine yoğunlaşacağız.

# Algoritma Analizi

- Performanstan daha önemli ne vardır ?
  - modülerlik
  - doğruluk
  - bakım kolaylığı
  - işlevsellik
  - sağlamlık
  - kullanıcı dostluğu
  - programcı zamanı (fiyat)
  - basitlik
  - genişletilebilirlik
  - güvenilirlik

# Neden algoritmalar ve başarımla uğraşırız?

- Başarım (performans) genelde yapılabilir olanla imkansızın arasındaki çizgiyi tanımlar.
- Algoritmik matematik program davranışlarını açıklamak için ortak dil oluşturur.
- Başarım **bilgi işleme**'nin para birimidir.
- Program başarımından alınan dersler diğer bilgi işleme kaynaklarına genellenebilir.
- Hız eğlencelidir!

# Algoritmik Performans

- Algoritmik performansın iki yönü vardır:
  - **Zaman (Time)**
    - Yönergeler veya talimatlar zaman alabilir.
    - Algoritma ne kadar hızlı bir performans gösteriyor?
    - Algoritmanın çalışma zamanını (runtime) ne etkiler?
    - Bir algoritma için gerekli olan zaman  nasıl tahmin edilir?
    - Gerekli olan zaman nasıl azaltılabilir?
  - **Alan (Space)**
    - Veri yapıları yer kaplar.
    - Ne tür veri yapıları kullanılabilir?
    - Veri yapılarının seçimi çalışma zamanını nasıl etkiler?

# Algoritma Analizi

- Bir algoritmanın analizinin yapılabilmesi için matematiksel bilgilere (temel olasılık, kümeler, cebir, v.b.) ihtiyaç duyulduğu gibi bazı terimlerin formül olarak ifade edilmesi gereklidir. Çünkü her giriş için algoritmanın davranışı farklı olabilir.

- Benzer problemi çözmek için iki algoritmanın zaman verimliliğini nasıl karşılaştırabiliriz?
  - **Naif( Basit ) yaklaşım:** bir programlama dilinde bu algoritmaların uygulanması ve çalışma zamanlarının karşılaştırılması.

# Algoritma Analizi

- Algoritmalar yerine programların karşılaştırılmasında aşağıda belirtilen zorluklar vardır.
  - **Programın kullanabileceği veri nedir?**
    - Analiz yöntemi veriye bağımlı olmamalıdır. Çalışma zamanı verinin büyüklüğü ile değişebilir.
  - **Hangi bilgisayarı kullanmak gerekir?**
    - Algoritmaların verimliliği belirli bir bilgisayara bağımlı olmadan karşılaştırılmalıdır. Çünkü, aynı algoritmanın işlemci hızları farklı iki bilgisayarda çalışma zamanı aynı olmaz.
  - **Algoritma nasıl kodlanmalıdır?**
    - Çalışma zamanını karşılaştırmak, uygulamaları karşılaştırmak anlamına gelir. Uygulamalar, programlama tarzına duyarlı olduğundan karşılaştıramayız. Programlama tarzı çok verimli bir algoritmanın çalışma zamanını bile etkileyebilir.
  - Programları karşılaştırmak, bir algoritmanın kesin ölçümü için uygun değildir.

# Algoritmaların Analizi

- Algoritma analizi, özel uygulamalardan, bilgisayarlardan veya veriden bağımsızdır.
- Algoritma analizi, tasarlanan program veya fonksiyonun belirli bir işleme göre matematiksel ifadesini bulmaya dayanır.
- Algoritmaları analiz etmek;
  - İlk olarak, algoritmanın etkinliğini değerlendirmek için belirli bir çözümde anlamlı olan işlemlerin kaç adet olduğu sayılır.
  - Daha sonra büyüme fonksiyonları kullanılarak algoritmanın verimliliği ifade edilir.

# Algoritma Analizi

| Problem | n elemanlı giriş | Temel işlem |
|---|---|---|
| Bir listede arama | liste ***n*** elemanlı | karşılaştırma |
| Bir listede sıralama | liste ***n*** elemanlı | karşılaştırma |
| İki matrisi çarpma | ***n x n*** boyutlu iki matris | çarpma |
| Bir ağaçta dolaşma | ***n*** düğümlü ağaç | Bir düğüme erişme |
| Hanoi kulesi | ***n*** disk | Bir diski taşıma |

- Not: Temel işlem tanımlayarak bir algoritmanın karmaşıklığını ölçebiliriz ve giriş büyüklüğü ***n*** için bu temel işlemi algoritmanın kaç kez gerçekleştirdiğini sayabiliriz.

# Algoritmaların Analizi

- Anlamlı olan işlemler hakkında önemli not:
  - Eğer problemin boyutu çok küçük ise algoritmanın verimliliğini muhtemelen ihmal edebiliriz.
  - Algoritmanın zaman ve bellek gereksinimleri arasındaki ağırlığı dengelemek zorundayız.
  - Dizi tabanlı listelerde geri alma işlemleri ***O(1)***'dir. Bağlı listelerde geri alma işlemi ise ***O(n)***'dir. Fakat eleman ekleme ve silme işlemeleri bağlı liste uygulamalarında çok daha kolaydır.

# Algoritmaların Analizi: Çalışma Zamanı fonksiyonu :T(n)

- **Çalışma zamanı veya koşma süresi (running time) fonksiyonu:**
- ***'n'*** boyutlu bir problemin algoritmasını çalıştırmak için gerekli zamandır ve **T(n)** ile gösterilir.

- Başka bir ifadeyle **T(n)**: **bir programın veya algoritmanın işlevini yerine getirebilmesi için, döngü sayısı, toplama işlemi sayısı, atama sayısı gibi işlevlerden kaç adet yürütülmesini veren bir bağıntıdır**.

# Algoritmaların Çalışma Zamanı

- **Örnek:** Basit ***if*** bildirimi

| | Cost | Times |
|---|---|---|
| `if (n < 0)` | c1 | 1 |
| `  absval = -n;` | c2 | 1 |
| `else` | | |
| `  absval = n;` | c3 | 1 |

- Toplam maliyet <=c1+max(c2,c3)

# Tahmin için Genel Kurallar

- **Döngüler (Loops)**
  - Bir döngünün çalışma zamanı en çok döngü içindeki deyimlerin çalışma zamanının iterasyon sayısıyla çarpılması kadardır.
- **İç içe döngüler (Nested Loops)**
  - İç içe döngülerde grubunun içindeki deyimin toplam çalışma zamanı, deyimlerin çalışma sürelerinin bütün döngülerin boyutlarının çarpımı kadardır. Bu durumda analiz içten dışa doğru yapılır.
- **Ardışık deyimler**
  - Her deyimin zamanı birbirine eklenir.
- **if/else**
  - En kötü çalışma zamanı: test zamanına **then** veya **else** kısmındaki çalışma zamanının hangisi büyükse o kısım eklenir.

# Algoritmaların Çalışma Zamanı

- **Örnek:** Basit bir döngü

| | Maliyet | Tekrar |
|---|---|---|
| i = 1; | c1 | 1 |
| sum = 0; | c2 | 1 |
| while (i <= n) { | c3 | n+1 |
| i = i + 1; | c4 | n |
| sum = sum + i; | c5 | n |
| } | | |

  - Toplam maliyet=c1 + c2 + (n+1)*c3 + n*c4 + n*c5 =3n+3
  - T(n)=3n+3
- Bu algoritma için gerekli zaman ***n*** ile doğru orantılıdır.

# Algoritmaların Çalışma Zamanı

- **Örnek:** İç içe döngü

| | Maliyet | Tekrar |
|---|---|---|
| i=1; | c1 | 1 |
| sum = 0; | c2 | 1 |
| while (i <= n) { | c3 | n+1 |
| j=1; | c4 | n |
| while (j <= n) { | c5 | n*(n+1) |
| sum = sum + i; | c6 | n*n |
| j = j + 1; | c7 | n*n |
| } | | |
| i = i +1; | c8 | n |
| } | | |

- Toplam maliyet= c1 + c2 + (n+1)*c3 + n*c4 + n*(n+1)*c5+ n*n*c6+ n*n*c7+ n*c8
- Bu algoritma için gerekli zaman $n^2$ ile doğru orantılıdır.