

YMH 317

Algoritma Analizi

Prof.Dr. Erkan TANYILDIZI

1.Hafta

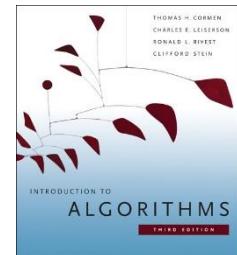
Algoritmaların

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 gereklidir.

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, fısıldasmayın, 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

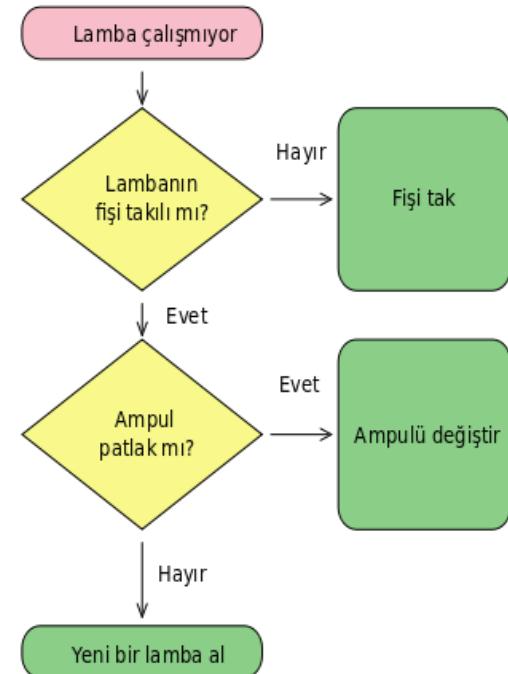
Algoritma Nedir?

- **Algoritma,**

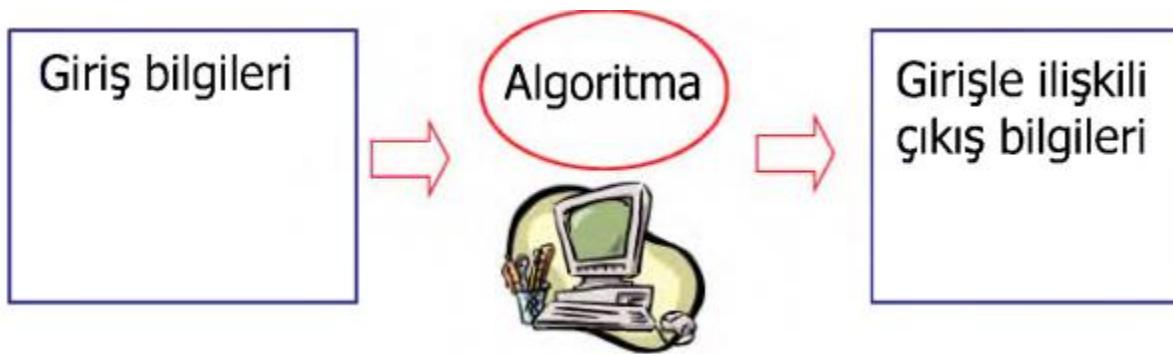
- bir problemin çözmek için bir prosedür veya formüldür.
- problemi çözmek için takip edilmesi gereken yönergeler kümesidir.
- matematikte ve bilgisayar biliminde bir işi yapmak için tanımlanan, bir başlangıç durumundan başladığında, açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler kümesidir.

- **Program**

- Bir programlama dilinde algoritmanın gerçekleştirilemesidir.



Algoritmik çözüm



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

Algoritmaların Özellikleri

- Bir algoritmanın taşımıası 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 bölgeleridir.

- **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 algoritmalarдан en iyisinin seçilmesi gereklidir.
- 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 yaklaşımalar seçme.
- En popüler tasarım stratejileri böl ve fethet (divide&conquer),
ağzılı(greedy), dinamik programlama, özyineleme (recursive).

○ 2- Algoritma ifadesi ve uygulanması

- Algoritmayı tasarladıkten sonra sözde kod (pseudocode)
ifadesinin belirlenmesi ve problem için uygulanması
- Bu konudaki endişeler, netlik, özlülük, etkinlik vb.

Algoritmaların Yonleri

○ 3-Algoritma Analizi (Çözümlenmesi)

- Algoritma analizi, algoritmayı gerçekte uygulamadan, bir algoritmayı çalıştırabilme 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 Yonleri

- **5- Algoritma veya programı doğrulama**
 - Algoritmanın verilen tüm olası girişler için hesaplama yaptığı 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 hesaplanması 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 faydalananarak bir sonraki çözümü elde etme ve elde edilen çözüm ile önceki çözümlerin birkaçının kullanılması ile bir sonraki çözümün elde edilmesi ile problemi çözme işlemeye **özyineleme yöntemi** denir.

○ Böl ve fethet

- Kompleks problemlerin bir bütün olarak çözülmeleri çok zor olacağından dolayı, bu problemler alt problemlere bölünürler. Bu bölümme işleminin yapılabilmesi için alt problemlerin bir üst seviyedeki problem ile aynı özelliği sağlamalıdır. Bu yöntem ile algoritma tasarıımı 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 gelişigüzelik ilkesi ile etkili bir şekilde problem çözümü yapılmaktadır. Bunlara örnek olarak Las Vegas polinom-zamanlı ve Monte Carlo polinom-zamanlı algoritmalar verilebilir. Gelişigüzelik 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 çalışmalarıdır.
- 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 gereklidir?**
 - 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 büyümeye 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	$n \times n$ 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 işlemleri 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 çalıştmak 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

	<u>Cost</u>	<u>Times</u>
if (n < 0)	c1	1
absval = -n;	c2	1
else		
absval = n;	c3	1

- Toplam maliyet $\leq c_1 + \max(c_2, c_3)$

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

- $i = 1;$
- $sum = 0;$
- $while (i \leq n) \{$
- $i = i + 1;$
- $sum = sum + i;$
- }

Maliyet	Tekrar
c1	1
c2	1
c3	$n+1$
c4	n
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 |
|--|---|-----------|
| ○ <code>i=1;</code> | c_1 | 1 |
| ○ <code>sum = 0;</code> | c_2 | 1 |
| ○ <code>while (i <= n) {</code> | c_3 | $n+1$ |
| ○ <code> j=1;</code> | c_4 | n |
| ○ <code> while (j <= n) {</code> | c_5 | $n*(n+1)$ |
| ○ <code> sum = sum + i;</code> | c_6 | $n*n$ |
| ○ <code> j = j + 1;</code> | c_7 | $n*n$ |
| ○ <code> }</code> | | |
| ○ <code> i = i +1;</code> | c_8 | n |
| ○ <code>}</code> | | |
| ○ Toplam maliyet= | $c_1 + c_2 + (n+1)*c_3 + n*c_4 + n*(n+1)*c_5 + n*n*c_6 +$ | |
| | $n*n*c_7 + n*c_8$ | |
| ○ Bu algoritma için gerekli zaman n^2 ile doğru orantılıdır. | | |

2.Hafta

Asimptotik Analiz (Notasyonlar)

O, Ω , ve Θ Notasyonu

Algoritmanın Büyüme Oranları

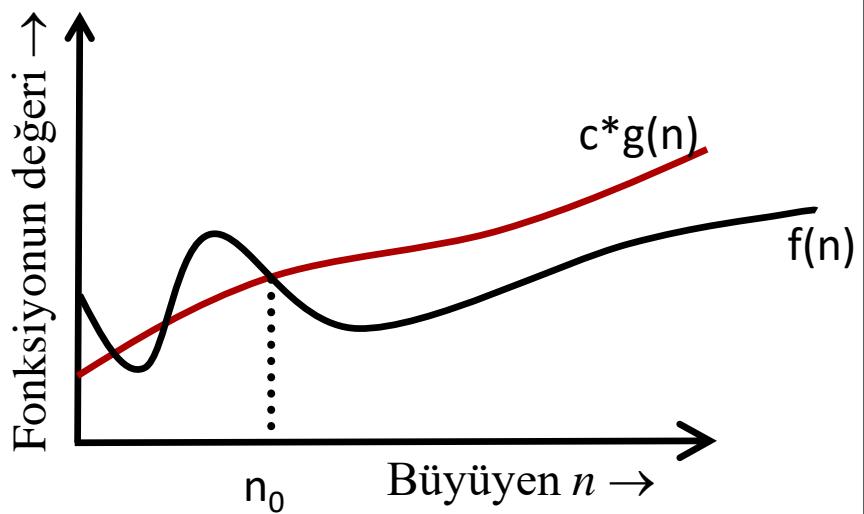
- Bir algoritmanın orantılı zaman gereksinimi büyüme oranı (veya büyümeye hızı) olarak bilinir.
- $T(n)$ nin büyümeye oranı, algoritmanın hesaplama karmaşıklığıdır.
- **Hesaplama karmaşıklığı** belirli bir uygulamadan bağımsız olarak n ile değişen $T(n)$ ' nin çalışma zamanını daha doğru bir şekilde bulmayı sağlar.
 - Genel olarak, az sayıda parametreler için karmaşıklıkla ilgilenilmez; eleman sayısı n 'nin sonsuza gitmesi durumunda $T(n)$ büyümeye bakılır.
 - Karmaşıklığı belirtmek için asimtotik notasyon (simgelem) ifadeleri kullanılmaktadır.

Asimptotik simgelem (notasyon)

O-simgelemi (üst sınırlar)

Tüm $n \geq n_0$ değerleri için sabitler $c > 0$, $n_0 > 0$ ise
 $0 \leq f(n) \leq cg(n)$ durumunda
 $f(n) = O(g(n))$ yazabiliriz.

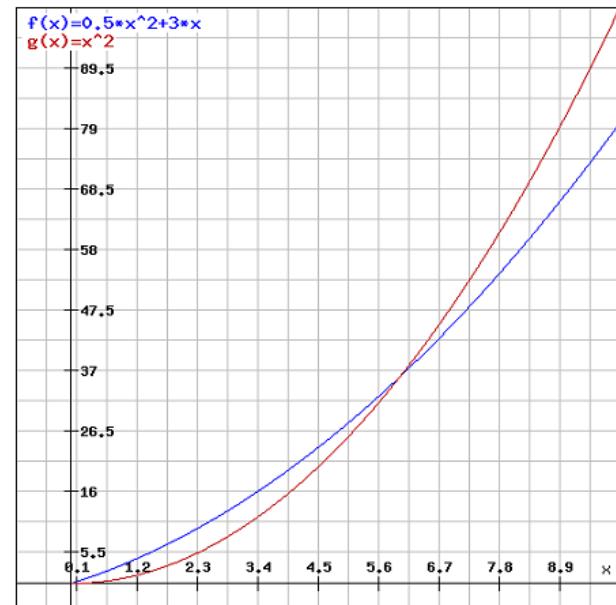
- $f(n)$ ve $g(n)$ verilen iki çalışma zamanı fonksiyonudur.
- $f(n) \leq c.g(n)$ ve $n \geq n_0$ koşullarını sağlayan c ve n_0 değerleri varsa $f(n)$ zaman karmaşıklığı $O(g(n))$ dir.
- Başka bir deyişle, n sayısı yeteri kadar büyük olduğunda, $f(n)$, $g(n)$ ile aynı büyüklüktedir.
- O -notasyonu sabit bir katsayı içinde bir fonksiyon için üst sınırı verir



O-simgeleri (üst sınırlar)

- Örnek: $2n^2 = O(n^3)$ için c , n_0 değerlerini bulunuz?
- $0 \leq f(n) \leq c.g(n)$, $0 \leq 2n^2 \leq cn^3$
- $c=1$ için $n_0=2$, şartı sağlar.
- Örnek: $(1/2)n^2 + 3n$ için üst sınırın $O(n^2)$ olduğunu gösteriniz.
- $c=1$ için
- $(1/2)n^2 + 3n \leq n^2$
- $3n \leq 1/2n^2$
- $6 \leq n$,
- $n_0=6$

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



Algoritmanın Büyüme Oranları

- Büyüme oranlarına bakarak iki algoritmanın verimliliğini karşılaştırabiliriz.
 - n' nin yeterince büyük değerleri için düşük büyümeye oranına sahip algoritma her zaman daha hızlıdır.
 - Örneğin; $f(n)=n^2+3n+5$ ifadesinin büyümeye oranı $O(n^2)$ dir.
- Algoritma tasarımcılarının amacı, çalışma zaman fonksiyonu olan $f(n)$ nin mümkün olduğu kadar düşük büyümeye oranı sahip bir algoritma olmasıdır.

O-notasyonu

- Örnek
- $3n^2 + 2n + 5 = O(n^2)$ olduğunu gösteriniz
- $10 n^2 = 3n^2 + 2n^2 + 5n^2$
- $\geq 3n^2 + 2n + 5 , n \geq 1$
- $c = 10, n_0 = 1$

Ortak Büyüme Oranları

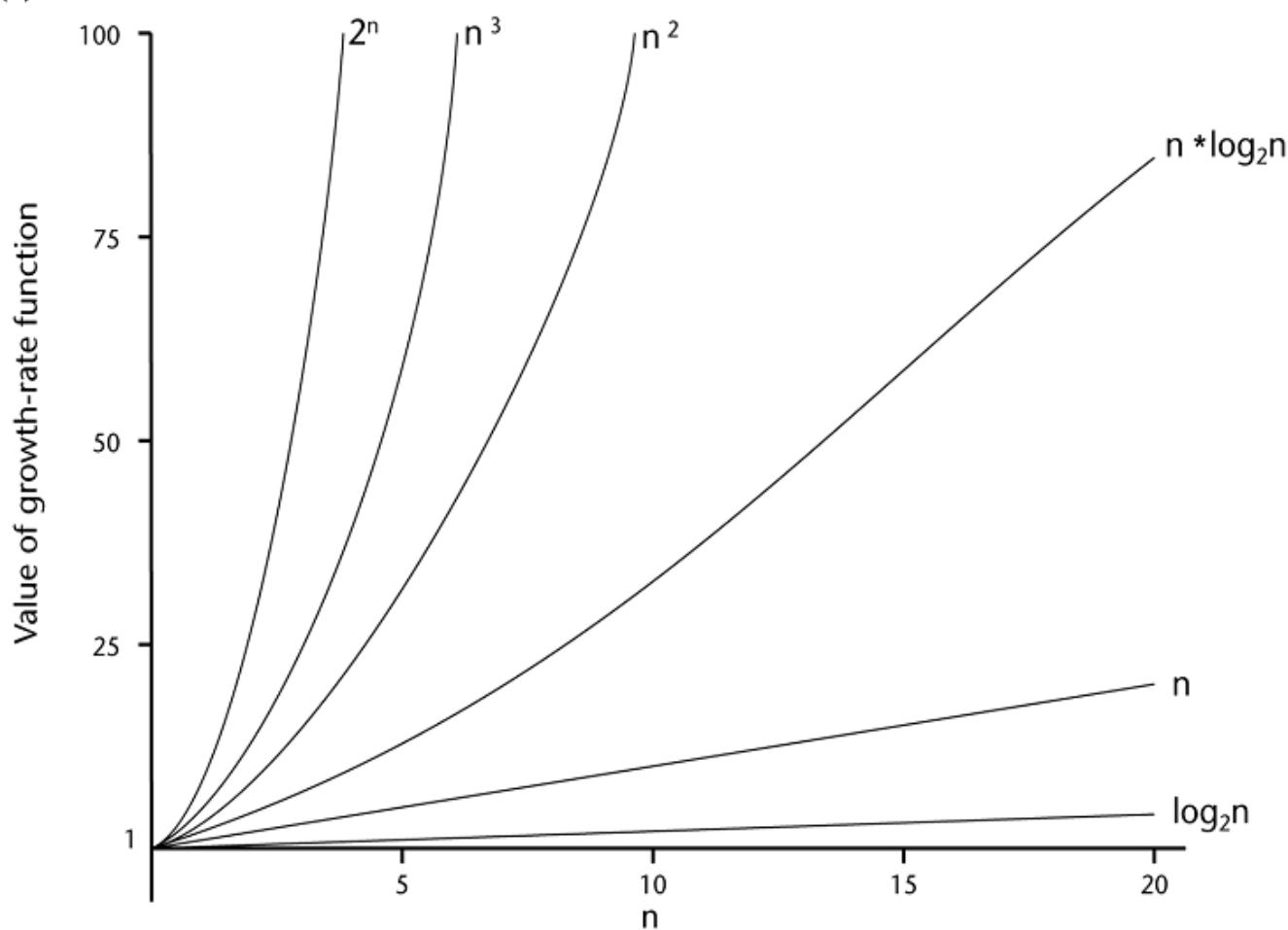
Maliyet artar

Fonksiyon	Büyüme oranı ismi
c	Sabit, komut bir veya birkaç kez çalıştırılır. Yenilmez!
$\log n$	Logaritmik, Büyük bir problem, her bir adımda sabit kesirler tarafından orijinal problemin daha küçük parçalara ayrılması ile çözülür. İyi hazırlanmış arama algoritmalarının tipik zamanı
$\log^2 n$	Karasel logaritmik
n	Doğrusal, Küçük problemlerde her bir eleman için yapılır. Hızlı bir algoritmadır. N tane veriyi girmek için gereken zaman.
$n \log n$	Doğrusal çarpanlı logaritmik. Çoğu sıralama algoritması
n^2	Karasel. Veri miktarı az olduğu zamanlarda uygun ($n < 1000$)
n^3	Kübik. Veri miktarı az olduğu zamanlarda uygun ($n < 1000$)
2^n	İki tabanında üssel. Veri miktarı çok az olduğunda uygun ($n \leq 20$)
10^n	On tabanında üssel
$n!$	Faktöriyel
n^n	n tabanında üstel (çoğu ilginç problem bu kategoride)

Büyüme oranı fonksiyonlarının karşılaştırılması

Function	n					
	10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

Büyüme oranı fonksiyonlarının karşılaştırılması



Büyüme oranı fonksiyonlarının karşılaştırılması

- Verimli algoritmaları geliştirmenin önemi:

Dizi boyutu	Sıralı arama	İkili (Binary) arama
128	128	8
1,048,576	1,048,576	21

- Verilen zaman karmaşıklığı ile algoritmalar için yürütme zamanı

n	$f(n)=n$	$f(n)=n\log n$	$f(n)=n^2$	$f(n)=2^n$
20	0.02 μ s	0.086 μ s	0.4 μ s	1 ms
10^6	1 μ s	19.93 ms	16.7 dk	31.7 yıl
10^9	1s	29.9 s	31.7 yıl	!!! yüzyıllar

Büyüme oranı fonksiyonlarının özellikleri

- 1- Bir algoritmanın büyümeye oranı fonksiyonunda düşük dereceli terimler, sabitler ve katsayılar ihmal edilebilir.
 - $O(n^3+4n^2+3n) \rightarrow O(n^3)$
 - $O(8n^4) \rightarrow O(n^4)$

- 2- Algoritmanın büyümeye fonksiyonlarını birleştirebiliriz.
 - $O(f(n))+O(g(n)) = O(f(n)+g(n))$
 - $O(n^3)+O(4n^2) \rightarrow O(n^3+4n^2) \rightarrow O(n^3)$
 - Çarpma içinde benzer kurallara sahiptir.

Büyüme oranı analizi ile ilgili problemler

- 1- Daha küçük büyümeye oranına sahip bir algoritma yeterince büyük olmayan belirli n değerleri için daha hızlı büyümeye oranına sahip algoritmadan hızlı çalışmaz.

- 2- Aynı büyümeye oranına sahip algoritmalar çalışma zamanı fonksiyonlarındaki sabitlerden dolayı farklı çalışma zamanlarına sahip olabilirler. Ama iki algoritmanın da kırılma noktası aynı n değerine sahiptir.

Notasyonlarda eşitlik "=" gösterimi

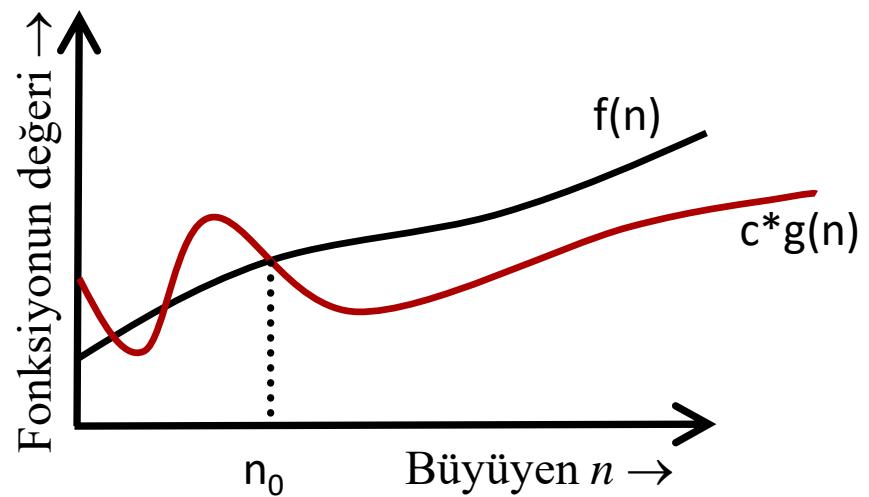
- A=B ise B = A anlamında değil mi?
- Fakat, $f(n) = O(g(n))$, $O(g(n)) = f(n)$ anlamına gelmez. Burada tek eşitlik söz konusudur.
- Burada "=", üyelik işlemi (\in) olarak tercih edilmiştir.
 - $f(n) = O(g(n)) \rightarrow f(n) \in O(g(n))$ dir
 - $O(g(n))$ bir küme anlamına gelir.
 - $f(n) = O(g(n)) \rightarrow O(g(n)) = \{f(n)\}$ gösterimi doğrudur.

Diğer Asimptotik Notasyonlar

Ω -simgelemi (alt sınırlar)

$\Omega(g(n)) = \{ f(n) : \text{tüm } n \geq n_0 \text{ değerlerinde}$
 $c > 0, n_0 > 0 \text{ ise,}$
 $0 \leq cg(n) \leq f(n) \}$

- Her durumda $f(n) \geq c g(n)$ ve $n \geq n_0$ koşullarını sağlayan pozitif, sabit c ve n_0 değerleri bulunabiliyorsa $f(n)=\Omega(g(n))$ dir.



Ω notasyonu-Örnek

- $2n + 5 \in \Omega(n)$ olduğunu gösteriniz
 - $n_0 \geq 0$, $2n+5 \geq n$, olduğundan sonuç elde etmek için $c=1$ ve $n_0 = 0$ alabiliriz.
- $5*n^2 - 3*n = \Omega(n^2)$ olduğunu gösteriniz.
 - $5*n^2 - 3*n \geq n^2$, $c=1$, $n_0 = 0$ değerleri için sağlar

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: $0 \leq cn^2 \leq 100n + 5$

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

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

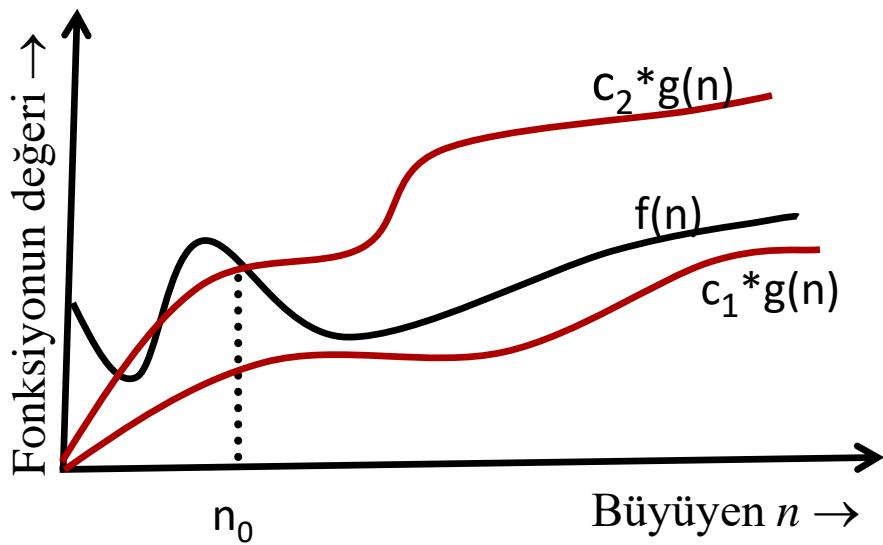
Since n is positive $\Rightarrow cn - 105 \leq 0 \Rightarrow n \leq 105/c$

\Rightarrow contradiction: n cannot be smaller than a constant

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

Θ Notasyonu – Sıkı Sınırlar

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



Θ notasyonu- Örnek

- $f(n) = 2n + 5 \in \Theta(n)$.
 - $2n \leq 2n+5 \leq 3n$, tüm $n \geq 5$ için

- $f(n) = 5*n^2 - 3*n \in \Theta(n^2)$.
 - $4*n^2 \leq 5*n^2 - 3*n \leq 5*n^2$, tüm $n \geq 4$ için

Examples

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

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

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

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

- $6n^3 \neq \Theta(n^2): c_1 n^2 \leq 6n^3 \leq c_2 n^2 \Rightarrow$ 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 $p(n) = \sum_{i=0}^d a_i n^i$
 $p(n) = \Theta(n^d)$

O, Ω ve Θ notasyonları arasındaki ilişkiler

- Eğer $g(n) = \Omega(f(n))$ ise $\rightarrow f(n)=O(g(n))$
- Eğer $f(n)= O(g(n))$ ve $f(n)= \Omega(g(n))$ ise $\rightarrow f(n)= \Theta(g(n))$
- $\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$

O, Ω ve Θ notasyonları arasındaki ilişkiler

- n 'nin büyük olduğu ve sabitlerin elendiği durumlarda.
- $O(g(n))$ düşünürsek $f(n)$ ile “eşit veya büyük”
 - Üstten sınır: $f(n)$ ile “aynı hızda veya hızlı büyür”
- $\Omega(g(n))$ düşünürsek $f(n)$ ile “eşit veya küçük”
 - Altan sınır: $f(n)$ ile “aynı hızda veya yavaş büyür”
- $\Theta(g(n))$ düşünürsek $f(n)$ ile “eşit”
 - Altan ve Üsten sınır : büyümeye oranları eşit
- Her bağıntı için alt ve üst sınırlar aynı zamanda belirlenemiyorsa veya bu sınırlardan sadece biri belirlenebiliyorsa, o sınıra göre notasyon gösterimi yapılır.

Θ Notasyonunun Özellikleri

- $f(n) = \Theta(f(n))$, yansıtma (reflexivity) özelliği
- $g(n) = \Theta(f(n))$ olduğu durumda $f(n) = \Theta(g(n))$ dir. simetri (symmetry) özelliği
- Eğer $f(n)=\Theta(g(n))$ ve $g(n)=\Theta(h(n))$ ise $f(n)=\Theta(h(n))$ geçişme (transitivity) özelliği
- $c > 0$ olduğu herhangi bir durum için, bu fonksiyon $c.f(n) = \Theta(f(n))$ dir.
- Eğer $f_1=\Theta(g_1(n))$ ve $f_2(n)=\Theta(g_2(n))$ ise $(f_1+f_2)(n)=\Theta(\max\{g_1(n), g_2(n)\})$
- Eğer $f_1=\Theta(g_1(n))$ ve $f_2(n)=\Theta(g_2(n))$ ise $(f_1.f_2)(n)=\Theta((g_1.g_2)(n))$
- Simetri özelliği dışındaki diğer özellikler **O** ve **Ω** notasyonlarında da vardır.

o-notasyonu ve ω -notasyonu

- O-notasyonu ve Ω -notasyonu \leq ve \geq gibidir.
- o-notasyonu ve ω -notasyonu $<$ ve $>$ gibidir.
- o-notasyonunda üst sınıra, ω notasyonununda ise alt sınıra eşitlik yoktur. Bundan dolayı üst ve alt sınırları sıkı bir asimptotik notasyon değildir.
- Oncekinden tek farklılığı, c katsayısı ve bir n_0 değeri var demek yerine, her c katsayısı için başka bir n_0 olacağını kabul etmek.

o-notasyonu

- $o(g(n)) = \{ f(n) : \text{tüm } n \geq n_0 \text{ değerlerinde}$
 $c > 0$ sabiti için n_0 sabiti varsa
 $0 \leq f(n) < cg(n).\}$
- Çalışma sürelerinin karşılaştırılması için kullanılır. Eğer $f(n) = o(g(n))$, ise $g(n)$, $f(n)$ fonksiyonundan daha ağırlıklıdır. Sıkı bir üst sınır vermez.
- $\lim_{n \rightarrow \infty} f(n)/g(n) = 0$
- o-notasyonunda n sonsuza gittiğinde $f(n)$ fonksiyonu $g(n)$ fonksiyonu karşısında önemini kaybeder.

o-notasyonu

- Örnek: $n^2/2 \in o(n^3)$,
 - $\lim_{n \rightarrow \infty} (n^2/2)/n^3 = \lim_{n \rightarrow \infty} 1/2n = 0$
- Örnek: $2n = o(n^2)$, fakat $2n^2 \neq o(n^2)$
- Örnek: $2n^2 = o(n^3)$ ($n_0 = 2/c$)
- **Önerme:** $f(n) \in o(g(n)) \Rightarrow O(f(n)) \subset O(g(n))$

ω-notasyonu

- $\omega(g(n)) = \{ f(n) : \text{tüm } n \geq n_0 \text{ değerlerinde}$
 $c > 0$ sabiti için n_0 sabiti varsa
 $0 \leq c.g(n) < f(n)$ }
- Çalışma sürelerinin karşılaştırılması için kullanılır. Eğer $f(n)=\omega(g(n))$, ise $g(n)$, $f(n)$ fonksiyonundan daha ağırlıklıdır. Sıkı bir alt sınır vermez.
 - $\lim_{n \rightarrow \infty} f(n)/g(n) = \infty$, eğer limit varsa
- Örnek: $n^2/2 \in \omega(n)$,
 - $\lim_{n \rightarrow \infty} (n^2/2)/n = \infty$, fakat $n^2/2 \neq \omega(n^2)$
- Örnek: $n^{1/2} = \omega(\log n)$, ($n_0 = 1+1/c$)

Asimptotik Notasyonlar

- $f(n) = O(g(n)) \quad \approx \quad f \leq g$
- $f(n) = \Omega(g(n)) \quad \approx \quad f \geq g$
- $f(n) = \Theta(g(n)) \quad \approx \quad f = g$
- $f(n) = o(g(n)) \quad \approx \quad f < g$
- $f(n) = \omega(g(n)) \quad \approx \quad f > g$

Asimptotik Notasyonlarının Karşılaştırılması

- Geçişlilik(Transitivity):

- $f(n) = \Theta(g(n))$ ve $g(n) = \Theta(h(n))$ ise $f(n) = \Theta(h(n))$,
- $f(n) = O(g(n))$ ve $g(n) = O(h(n))$ ise $f(n) = O(h(n))$,
- $f(n) = \Omega(g(n))$ ve $g(n) = \Omega(h(n))$ ise $f(n) = \Omega(h(n))$,
- $f(n) = o(g(n))$ ve $g(n) = o(h(n))$ ise $f(n) = o(h(n))$,
- $f(n) = \omega(g(n))$ ve $g(n) = \omega(h(n))$ ise $f(n) = \omega(h(n))$.

- Dönüşlülük veya yansima(Reflexivity):

- $f(n) = \Theta(f(n))$,
- $f(n) = O(f(n))$,
- $f(n) = \Omega(f(n))$.

- Simetri(Symmetry):

- $g(n) = \Theta(f(n))$ olduğu durumda, $f(n) = \Theta(g(n))$

- Transpose symmetry:

- $g(n) = \Omega(f(n))$ olduğu durumda, $f(n) = O(g(n))$,
- $g(n) = \omega(f(n))$ olduğu durumda, $f(n) = o(g(n))$.

En iyi (Best), En kötü (Worst), Ortalama(Average) Durum Analizi

- **En iyi durum (best case):** Bir algoritma için, çalışma zamanı, maliyet veya karmaşıklık hesaplamalarında en iyi sonucun elde edildiği duruma “en iyi durum” denir. Bir giriş yapısında hızlı çalışan yavaş bir algoritma ile hile yapmak. (gerçek dışı) Ör: Bütün elemanların sıralı olduğu durum.
- **En kötü durum (worst case):** Tüm olumsuz koşulların olması durumunda algoritmanın çözüm üretmesi için gerekli maksimum çalışma zamanıdır. (genellikle). Ör: Bütün elemanlar ters sıralı.
- **Ortalama durum (avarage case):** Giriş parametrelerin en iyi ve en kötü durum arasında gelmesi ile ortaya çıkan durumda harcanan zamandır. Fakat bu her zaman ortalama durumu vermeyebilir. (bazen) Ör: Elemanların yaklaşık yarısı kendi sırasındadır.

En iyi (Best), En kötü (Worst), Ortalama(Average) Durum Analizi

- **Dizi arama**

- Worst case = $O(n)$ Average case = $O(n)$

- **Quick sort**

- Worst case = $O(n^2)$ Average case = $O(n \log n)$

- **Merge Sort, Heap Sort**

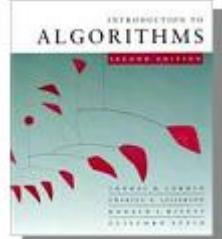
- Worst case = $O(n \log n)$ Average case = $O(n \log n)$

- **Bubble sort**

- Worst case = $O(n^2)$ Average case = $O(n^2)$

- **Binary Search Tree:** Bir elaman için arama

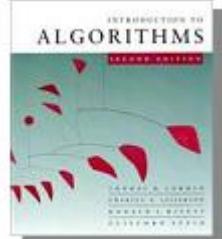
- Worst case = $O(\log n)$ Average case = $O(\log n)$



Diziler (Arrays)

- Diziler, tipleri homojen olan birden fazla elemandan oluşan veri grubudur. Diziler tek boyutlu, iki boyutlu, üç boyutlu, vb. şeklinde tanımlanabilirler.
- Diziler genelde diğer veri yapılarında kullanılan bir veri yapısıdır. Diziler **statik veriler** olarak da isimlendirilebilirler.
- Örnek: Tek boyutlu A dizisi

0	1	2	3	4	5	6	7	8	9
6	7	12	23	46	78	12	5	8	2



Diziler-Doğrusal Arama (Search)

- Statik ve sıralanmamış veri dizisi üzerinde arama algoritması (Lineer arama).

Lineer Arama(n, x)

1. $i \leftarrow 1$, $Veri_Var \leftarrow 0$
2. ($i \leq n$) ve $\sim (Veri_Var)$ olduğu sürece devam et
3. eğer $Dizi[i] = x$ ise
4. $Veri_Var \leftarrow 1$
5. $i \leftarrow i + 1$

- x , dizi içinde aranacak veri ve n dizinin boyutudur.
- \sim simgesinin anlamı önüne geldiği mantıksal ifadenin deęilini almaktır.

Diziler-Arama (Search) Analizi

- Başarısız Arama : $O(n)$

- Başarılı Arama:

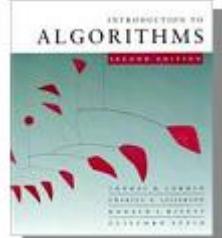
- Best-Case: x dizinin ilk elemanı ise: $O(1)$

- Worst-Case: x dizinin son elemanı ise: $O(n)$

- Average-Case: Listedeki her bir sayıyı bir kez aradığımızı düşünelim. Anahtar karşılaştırmalarının sayısı $1, 2, \dots, n$ ise : $O(n)$

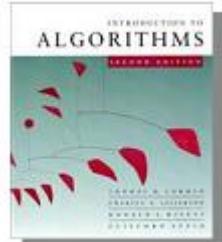
$$T(n) = \begin{cases} \Theta(1) & \text{en iyi durum} \\ \Theta(n) & \text{en kötü durum} \\ \Theta(n) & \text{ortalama durumu} \end{cases}$$

$$\frac{\sum_{i=1}^n i}{n} = \frac{(n^2 + n)/2}{n}$$



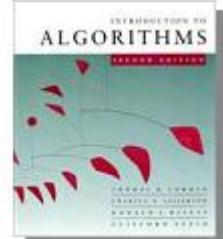
Diziler-Arama (Search) Analizi

- Lineer arama algoritması, herhangi bir dizi üzerinde yapılan bir aramadır.
- Dizinin elemanları sıralı olması veya olmaması herhangi bir anlam ifade etmez. Bu aramada aranacak eleman dizinin bütün elemanları ile karşılaşılır ve bu işleme, aranan eleman dizide bulununcaya kadar veya dizide olmadığı kesinlik kazanıncaya kadar devam edilir.
- Eğer dizinin elemanları sıralı ise, bu durumda mertebesi daha iyı olan bir algoritma kullanılabilmektedir.
 - **İkili Arama (Binary Search)**

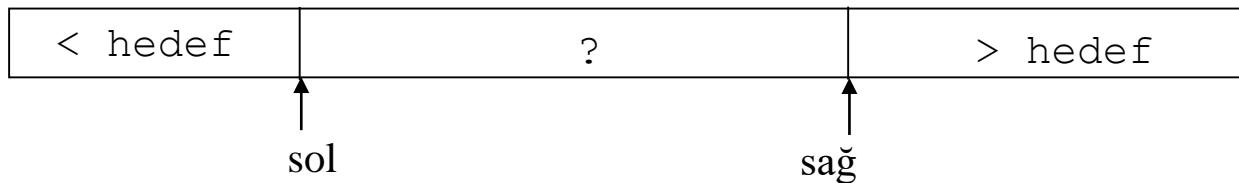


İkili Arama (Binary Search)

- İkili aramada dizi sıralanmış olduğundan, dizinin ortasında bulunan sayı ile aranan sayıyı karşılaştırarak arama boyutunu yarıya düşürülür ve bu şekilde devam edilir.
- Bu algoritmanın temel mantığı aranacak elemanı dizinin ortasındaki eleman ile karşılaştırıp, aranacak eleman eğer bu elemana eşitse, amaca ulaşılmıştır. Eğer eşit değilse, bu durumda aranacak eleman dizinin hangi parçası içinde olabileceği kararı verilir. Bu sayede arama boyutu yarıya düşürülür ve bu şekilde devam edilir.



İkili Arama (Binary Search)

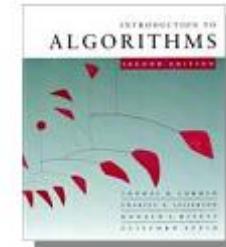


- Statik ve sıralı veri dizisi üzerinde ikili arama algoritması.

İkili Arama(Dizi,n,x,bulundu, yeri)

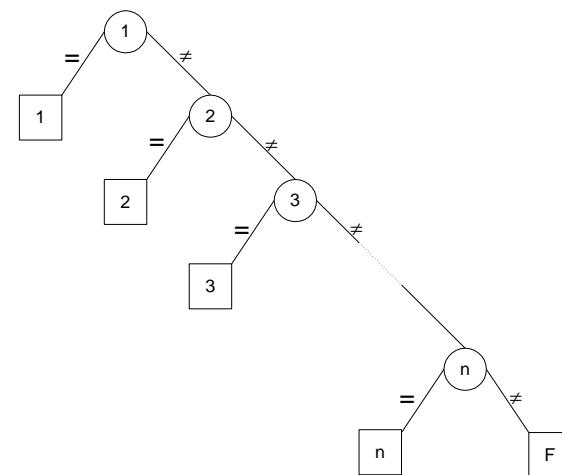
1. Yerel değişkenler
alt, ust, orta : integer;
2. Ust \leftarrow n, alt \leftarrow 1, bulundu \leftarrow 0
3. (bulundu = 0) ve (ust \geq alt) olduğu sürece devam et
4. orta $\leftarrow \lfloor (\text{alt} + \text{ust}) / 2 \rfloor$
5. eğer x = Dizi[orta] ise
 bulundu \leftarrow 1
7. değil ve eğer x < Dizi₂ [orta] ise
 ust \leftarrow orta-1
9. değilse
 alt \leftarrow orta+1
11. yeri \leftarrow orta

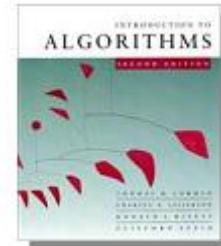
İkili Arama (Binary Search)



- İkili aramanın mantığı veya işleyiş şekli ağaç şeklinde gösterilebilir ve bu ağaca **karşılaştırma ağıacı** veya **karar ağıacı** denir. **Karşılaştırma ağıacı**, bir algoritmanın yapmış olduğu karşılaştırmaların hepsinin temsil edildiği bir ağaçtır.
- Örneğin, statik veri yapıları üzerinde Lineer arama algoritmasının karşılaştırma ağıacı;

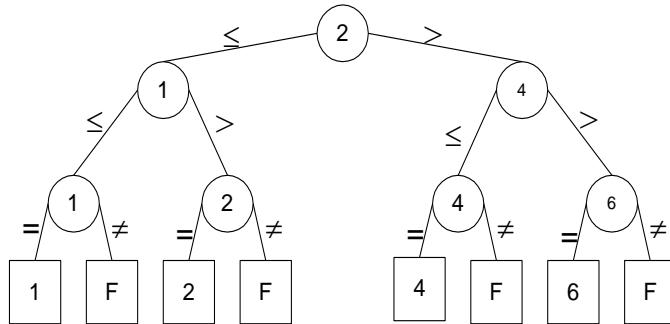
Aranacak bir elemanın dizinin içinde olup olmadığını test etmek amacıyla dizinin her elemanı ile karşılaştırma yapıldığından ağaç tek dal üzerine büyümektedir ve ağaçın bir tarafı hiç yok iken, diğer tarafı çok büyüyebilir. Buradan da rahatlıkla görülebileceği gibi Lineer arama algoritması iyi bir algoritma değildir.





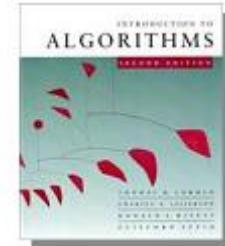
İkili Arama (Binary Search)

- Statik veri yapıları üzerinde yapılan ikili arama algoritması için karşılaştırma ağacı;



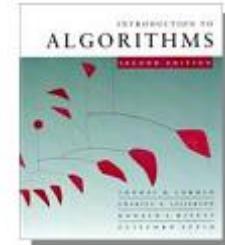
- Görülen karşılaştırma ağacı dengeli bir ağaçtır. Bazı durumlarda karşılaştırma ağacının bütün yaprakları aynı seviyede olmayıabilir.

İkili Arama (Binary Search)



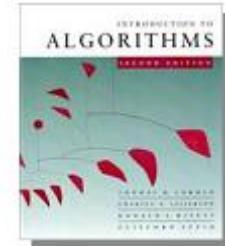
- Bu algoritmanın analizini yapmak için ilk olarak $n=2^k$ kabulu yapılsın.
- $\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \dots, \frac{n}{2^k} = 1, n=2^k$
- Bu kabulün yapılması genelliliği bozmayacaktır. Algoritma çalışlığında ilk bakılacak eleman 2^{k-1} endeksli eleman olacaktır. Eğer eşitlik varsa, amaca ulaşılmıştır. Eşitlik yoksa geriye kalan ve her birinin boyutu 2^{k-1} olan dizi parçalarının hangisinde aranan elemanın olacağına karar verilir.

İkili Arama (Binary Search)



- Eğer x değişkeninin değeri Dizi[2^{k-1}] elemanın değerinden küçükse,
- | | | | |
|---|---|-------|-----------|
| | | | |
| 1 | 2 | | 2^{k-1} |
- x elemanı bu parçanın içinde olabilir, diğer parçanın içinde olması mümkün değildir. Eğer x değişkeninin değeri Dizi[2^{k-1}] elemanın değerinden büyükse,
- | | | | |
|-------------|-------------|-------|-------|
| | | | |
| $2^{k-1}+1$ | $2^{k-1}+2$ | | 2^k |
- x elemanı bu parçanın içinde olabilir, diğer parçanın içinde olması olasılığı sıfırdır. Bu şekilde geriye kalan parçanın içindeki eleman sayısı $2'$ nin kuvveti kadar eleman kalacaktır. Bundan dolayı ortadaki elemanı bulmak için taban veya tavan fonksiyonuna ihtiyaç duyulmayacaktır. Dizi üzerinde yapılacak bölme sayısı logn olacaktır.

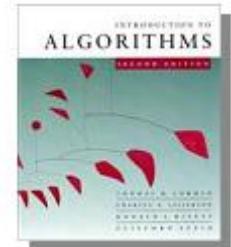
İkili Arama (Binary Search)



- Bir dizi için başarılı arama sayısı n ve 1 tanede başarısız arama sayısı yapılabilir. Toplam $n+1$ tane arama yapılabilir.
- Bu algoritmada en iyi durum aramasında döngü kısmı bir sefer çalışır ve en kötü durumda döngü kısmı $\lceil \lg n + 1 \rceil$ sefer çalışır.
- Ortalama arama zamanına bakılacak olursa, toplam arama sayısı ile arama adım sayısı çarpılır ve n değerine bölünür.
- Asimptotik notasyonda sabit katsayılar ihmal edildiğinden dolayı, ikili arama algoritmasının mertebesi $T(n)$,

$$T(n) = \begin{cases} \Theta(1) & \text{en iyi durum} \\ \Theta(\lg n) & \text{en kötü durum} \\ \Theta(\lg n) & \text{ortalama durum} \end{cases} \quad \frac{\sum_{i=1}^n \log n}{n} = \frac{n \log n}{n} = \log n$$

İkili Arama (Binary Search)



- Bu algoritmanın da en kötü durumu ile ortalama durumun asimptotik davranışları aynıdır. Fakat en kötü durum ve ortalama durum mertebeleri (çalışma zamanı) logaritmik olduğundan, lineer aramaya göre çok iyi olan bir algoritmadır.
- Eğer bir dizi içindeki veriler sıralı ise, her zaman ikili arama algoritmasını kullanmak sistemin performansını arttıracaktır.

Binary Arama Analizi

- Başarısız Arama : $O(n)$
- Başarılı Arama:
- Best-Case: $O(1)$
- Worst-Case: $O(\log n)$
- Average-Case: $O(\log n)$

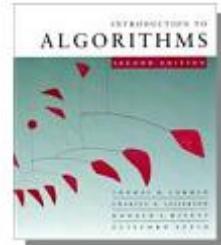
$$\sum_{i=1}^n i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \dots + n2^n = (n - 1)2^{n+1} + 2$$

$$\frac{1 \times 1 + 2 \times 2 + 3 \times 4 + \dots + \log n 2^{\log n - 1}}{n} = \frac{1}{n} \sum_{i=1}^{\log n} i2^{i-1}$$

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^{\log n} i2^{i-1} &= \frac{(\log n - 1)2^{\log n} + 1}{n} \\ &= \frac{n(\log n - 1) + 1}{n} \\ &\approx \log n - 1 \end{aligned}$$

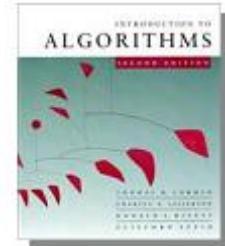
```
int ikiliArama(int A[], int N, int sayı)
{
    sol = 0;
    sag = N-1;
    while (sol <= sag){
        int orta = (sol+sag)/2;
        if (A[orta] == sayı) return orta;
        else if (sayı < A[orta]) sag = orta - 1;
        else sol = orta+1;
    }
    return -1;
}
```

İkili Arama (Binary Search)



- İkili arama algoritmasının performansını ölçmek için, bu algoritmanın $T(n)$ zaman bağıntısının tekrarlı bağıntısı (özyinelemeli bağıntı) elde edilebilir. İkili arama algoritmasının $T(n)$ bağıntısı
- $T(n)=T(\lfloor n/2 \rfloor)+\Theta(1)$ şeklinde olur. Bu tekrarlı bağıntının çözümü iteratif (iterasyon) yapılrsa (taban fonksiyonunu ihmal ederek),
 - $T(n)=T(n/2)+\Theta(1)$
 - $=\Theta(1)+(\Theta(1)+T(n/2^2))$
 - $=\Theta(1)+(\Theta(1)+(\Theta(1)+T(n/2^3)))$
 - $=(\lg n-1)\Theta(1)+T(n/2^{\lg n})$
 - $=(\lg n-1)\Theta(1)+T(1)$
- elde edilir.

İkili Arama (Binary Search)



- İkili arama için $T(1)=\Theta(1)$ olur, bundan dolayı
- $T(n) = (\lg n - 1)\Theta(1) + \Theta(1)$
- $= (\lg n)\Theta(1)$
- $= \Theta(\lg n)$ olur.
- Master yöntemi kullanılırsa, $f(n)=\Theta(1)$ ve $a=1$, $b=2$,
- $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$ olduğundan olur. Bundan dolayı $f(n)=\Theta(n^{\log_b a})$ olur. Master yönteminde bu şart sağlandığında
- $T(n) = \Theta(n^{\log_b a} \lg n)$
- $= \Theta(n^0 \lg n)$
- $= \Theta(\lg n)$
- elde edilir.
- Bu yöntemler ilerleyen bölümde detaylı olarak ele alınacaktır.

3.Hafta

Algoritmaların

Analizi

Araya Yerleştirme Sıralaması
(Insert Sort)
Birleştirme Sıralaması (Merge Sort)
Yinelemeler

Sorular

- 1. $f(n)$ ve $g(n)$ asimptotik negatif olmayan fonksiyonlar olsunlar.
 $\max(f(n), g(n)) = \Theta(f(n) + g(n))$ olduğunu gösteriniz.
- 2. $b > 0$ olmak üzere reel a ve b sabitleri için $(n+a)^b = \Theta(n^b)$ olduğunu gösteriniz.
- 3. $f(n) = 2^{n+1}$ ve $g(n) = 2^n$ ise $f(n) = O(g(n))$ midir? $f(n) = 3^{2n}$ olduğu durumda $f(n) = O(n)$ midir?
- 4. Bir algoritmanın en iyi çalışma performansında çalışma zamanı $\Omega(g(n))$ ve en kötü çalışma durumunda çalışma zamanı $O(g(n))$ ise, algoritmanın mertebesi $f(n) = \Theta(g(n))$ olduğunu gösteriniz.
- 5. Aşağıdaki kümelerden hangisi veya hangileri boş kümedir?
 - $\omega(f(n)) \cap o(f(n))$
 - $\Omega(f(n)) \cap O(f(n))$

Sorular

- 6. Asimptotik notasyonların tanımı bir parametre için verilmiştir. Eğer algoritmada birden fazla parametre varsa, bu durumda tanımlama şu şekilde yapılabilir. $f(m,n)$ ve $g(m,n)$ asimptotik negatif olmayan fonksiyonlar olsunlar. Bu durumda O -notasyonun tanımı
- $O(g(m,n)) = \{ f(m,n) : c, n_0 \text{ ve } m_0 \text{ pozitif sabitler olmak üzere } 0 \leq f(m,n) \leq cg(m,n), n \geq n_0 \text{ ve } m \geq m_0 \}$.
- Benzer tanımlamaları Θ , Ω , ω ve o için yapınız.
- 7. $\lg(n!) = \Theta(n \lg n)$ ve $n! = o(n^n)$ olduğunu gösteriniz.
- 8 . Eğer bir algoritmanın çalışma ortalama zaman bir $k > 0$ sabiti için $T(n) = O(n^k)$ ise $T(n) = n^{O(1)}$ olduğunu gösteriniz. Tersinin de doğruluğunu gösteriniz.
- 9. $\lceil \lg n \rceil!$ fonksiyonu asimptotik olarak sınırlı mıdır? $n > 0$ için $\lfloor \lg n \rfloor!$ fonksiyonu polinom olarak sınırlı mıdır?

Sorular

- 10. $P(n)$, derecesi d olan bir polinom olsun. $k > 0$ bir sabit olmak üzere $P(n) = O(n^k)$, $P(n) = \Theta(n^k)$, $P(n) = \Omega(n^k)$, $P(n) = \omega(n^d)$ ve $P(n) = o(n^d)$ durumları için d ve k arasındaki ilişkileri gösteriniz.
- 11. $f(n)$ ve $g(n)$ asimptotik pozitif fonksiyonlar olsunlar. Aşağıdakilerin doğruluğunu veya yanlışlığını gösteriniz.
 - $f(n) = O(g(n)) \rightarrow g(n) = O(f(n))$
 - $f(n) + g(n) = \Theta(\min(f(n), g(n)))$
 - $f(n) = O(g(n)) \rightarrow 2^{f(n)} = O(2^{g(n)})$
 - $f(n) = O((f(n))^2)$
 - $f(n) = O(g(n)) \rightarrow g(n) = \Omega(f(n))$
 - $f(n) = \Theta(f(n/2))$
 - $f(n) + o(f(n)) = \Theta(f(n))$

Sorular

- 12. $f(n)$ ve $g(n)$ asimptotik pozitif fonksiyonlar ve yeterince büyük bütün n' ler için $f(n) \geq 1$ ve $\lg(g(n)) > 0$ olsun. $f(n)=O(g(n))$ ise $\lg(f(n))=O(\lg(g(n)))$ olduğunu gösteriniz.
- 13. Eğer $f(n)=O(F(n))$ ve $g(n)=O(G(n))$ ise

$$\frac{f(n)}{g(n)} = O\left(\frac{F(n)}{G(n)}\right)$$

olur. Bu iddianın doğru ya da yanlış olduğunu gösteriniz.

- 14. Aşağıda iddiaların doğru ya da yanlış olduğunu gösteriniz.
- Eğer $f(n)=O(g(n))$ ve $g(n)=O(h(n))$ ise, $f(n)=O(h(n))$ olur.
- Eğer $f(n)=\Theta(g(n))$ ve $g(n)=\Theta(h(n))$ ise, $f(n)=\Theta(h(n))$ olur.

Ek-

52

Algoritmaların doğruluğu

- Bir algoritma geçerli olan herhangi bir giriş için sonlanmakta ve istenen bir sonucu üretmekte ise doğrudur.
- Algoritmaların doğruluğunun kontrolünde pratik teknikler kullanılır.

Döngü Değişmezleri (Loop Invariants)

- **Değişmezler (Invariants)**- Herhangi bir zamanda ulaşıldıklarında veya işlem yapıldıklarında doğru oldukları varsayıılır (algoritma çalışma sırasında tekrarlı gerçekleşen işlemler, Örnek: döngülerde)
- Döngü değişmezleri için üç şeyi göstermek zorunludur:
 - **Başlatma(Initialization)** - ilk iterasyondan önce doğrudur
 - **Koruma(Maintenance)** - bir iterasyondan önce doğruysa bir sonraki iterasyondan önce doğruluğunu korur
 - **Sonlandırma (Termination)** – döngünün değişmezleri bitirmesi algoritmanın doğruluğunu gösterir

Örnek: Binary Search (1)

- null değeri döndüğünde q değerinin A dizisinde olmadığından emin olmak istiyoruz.

- Değişmez : her **while** döngüsünün başlangıcında,

$A[i] < q$ bütün $i \in [1 \dots left-1]$ ve

$A[i] > q$ bütün $i \in [right+1 \dots n]$

- Başlatma: $left=1$, $right=n$ olarak seçilir ve $left'$ in solunda ve $right'$ in sağında hiçbir elaman kalmaz.

```
left ← 1
right ← n
do
    j ← ⌊ (left + right ) / 2 ⌋
    if A[j]=q then return j
    else if A[j]>q then right ← j-1
    else left= j+1
while left<=right
return null
```

Örnek: Binary Search (2)

- **Koruma** : Eğer $A[j]>q$ ise $A[i]>q$ olur her $i \in [j...n]$, çünkü dizi sıralıdır. Algoritma $right$ değişkenine $j-1$ değerini atar.
 - **Sonlandırma**: $left>right$ olduğunda döngü bitirilir. q değeri $left$ solundaki A'nın tüm değerlerinden büyuktur ve $right'$ in sağındaki A'nın tüm değerlerinden küçüktür. Bu A'nın tüm elemanlarından q değerinin küçük yada büyük olduğunu gösterir.
- ```

left ← 1
right←n
do
 j← ⌊ (left + right) / 2 ⌋
 if A[j]=q then return j
 else if A[j]>q then right ←j-1
 else left= j+1
 while left<=right
 return null

```

## Örnek: Insert Sort

- **Değişmez:** her for döngüsü *başlangıcında*,  $A[1\dots j-1]$  dizisi sıralı olarak  $A[1\dots j-1]$  aralığındaki elemanlardan ibarettir.
- **Başlatma:**  $j=2$ , olarak alınır. Çünkü  $A[1]$  zaten sıralıdır.
- **Koruma :** İçteki while döngü elemanlar arasında  $A[j-1]$ ,  $A[j-2]$ , ...,  $A[j-k]$  şeklinde sırasını değiştirmeden bir önceki elemana gider. Daha sonra  $A[j]$  elemanı  $k$ . Pozisyon'a yerleştirilir, böylece  $A[k-1] < A[k] < A[k+1]$  olur.  
 $A[1\dots j-1]$  sıralı +  $A[j]$   $\rightarrow A[1\dots j]$  sıralı
- **Sonlandırma:**  $j=n+1$  olduğunda döngü bitirilir. Böylece  $A[1\dots n]$  orijinal olarak alınmış olan  $A[1\dots n]$  elemanla aynıdır ancak sıralanmış şekildedir.

```

for j $\leftarrow 2$ to length[A]
do key $\leftarrow A[j]$
 i $\leftarrow j-1$
 while i > 0 and A[i] > key
 do A[i+1] $\leftarrow A[i]$
 i $\leftarrow i-1$
 done
 A[i+1] \leftarrow key
done

```

# **3.Hafta**

# **Algoritmaların**

# **Analizi**

Araya Yerleştirme Sıralaması  
(Insert Sort)  
Birleştirme Sıralaması (Merge Sort )  
Yinelemeler

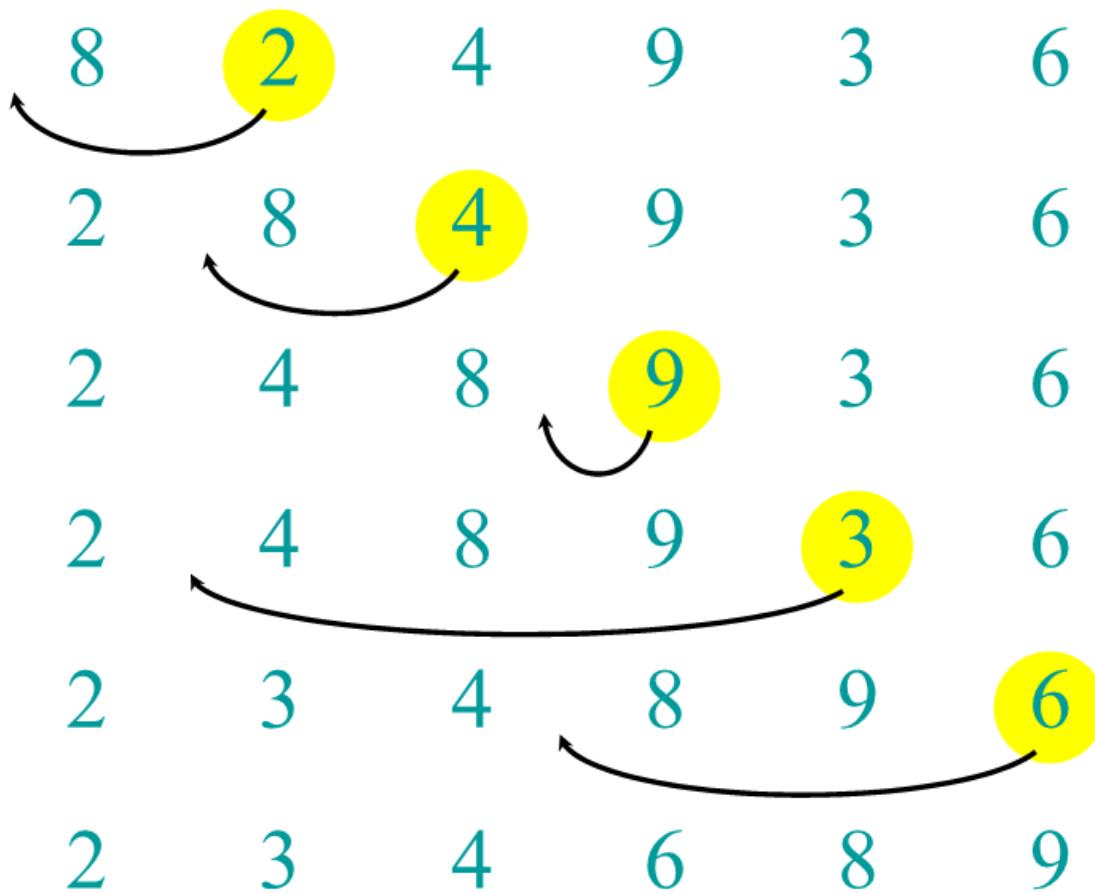
# Sıralama (sorting) problemi

- **Girdi:** dizi  $\langle a_1, a_2, \dots, a_n \rangle$  sayıları.
- **Çıktı:**  $\langle a'_1, a'_2, \dots, a'_n \rangle$  elemanları  $a_1, a_2, \dots, a_n$  elamanlarının bir permütasyonudur ve
$$a'_1 \leq a'_2 \leq \dots \leq a'_n .$$
- Permütasyon, rakamların sıralamasının değiştirilmesidir.
- Örnek:
- **Girdi:** 8 2 4 9 3 6
- **Çıktı:** 2 3 4 6 8 9

# Araya yerleştirme sıralaması (Insertion sort)

- Insert Sort, artımsal tasarım yaklaşımını kullanır:  
A[1...j-1] alt dizisi sıralanmış.
  
- Strateji:
- Bir elemanı olduğu sıraya ekle
- Bütün elemanları sıralayana kadar devam et

## Araya yerleştirme sıralaması örneği



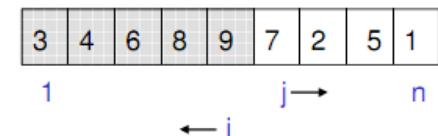
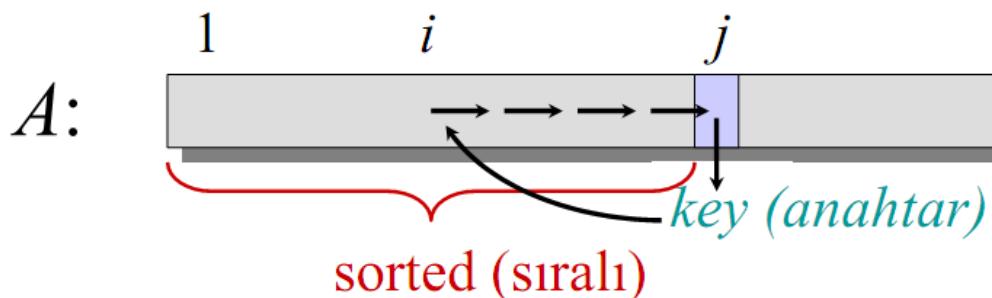
# Araya yerleştirme sıralaması (Insertion sort)

“pseudocode” (sözde kod) {

```

 INSERTION-SORT (A, n) ▷ $A[1 \dots n]$
 for $j \leftarrow 2$ to n
 do $key \leftarrow A[j]$
 $i \leftarrow j - 1$
 while $i > 0$ and $A[i] > key$
 do $A[i+1] \leftarrow A[i]$
 $i \leftarrow i - 1$
 $A[i+1] = key$

```



## Örnek: Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 30 | 10 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|                    |                      |                   |
|--------------------|----------------------|-------------------|
| $i = \emptyset$    | $j = \emptyset$      | $key = \emptyset$ |
| $A[j] = \emptyset$ | $A[j+1] = \emptyset$ |                   |

```
→ InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 30 | 10 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 2$     | $j = 1$ | $key = 10$    |
| $A[j] = 30$ |         | $A[j+1] = 10$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 2$     | $j = 1$ | $key = 10$    |
| $A[j] = 30$ |         | $A[j+1] = 30$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 2$     | $j = 1$ | $key = 10$    |
| $A[j] = 30$ |         | $A[j+1] = 30$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

$i = 2 \quad j = 0 \quad \text{key} = 10$   
 $A[j] = \emptyset \quad A[j+1] = 30$

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 30 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

$i = 2 \quad j = 0 \quad \text{key} = 10$   
 $A[j] = \emptyset \quad A[j+1] = 30$

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



# Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|                    |         |               |
|--------------------|---------|---------------|
| $i = 2$            | $j = 0$ | $key = 10$    |
| $A[j] = \emptyset$ |         | $A[j+1] = 10$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

# Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|                    |         |               |
|--------------------|---------|---------------|
| $i = 3$            | $j = 0$ | $key = 10$    |
| $A[j] = \emptyset$ |         | $A[j+1] = 10$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

# Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

$i = 3 \quad j = 0 \quad \text{key} = 40$   
 $A[j] = \emptyset \quad A[j+1] = 10$

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|                    |         |               |
|--------------------|---------|---------------|
| $i = 3$            | $j = 0$ | $key = 40$    |
| $A[j] = \emptyset$ |         | $A[j+1] = 10$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 3$     | $j = 2$ | $key = 40$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 3$     | $j = 2$ | $key = 40$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 3$     | $j = 2$ | $key = 40$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

$i = 4 \quad j = 2 \quad \text{key} = 40$   
 $A[j] = 30 \quad A[j+1] = 40$

→

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 2$ | $key = 20$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 2$ | $key = 20$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

$$\begin{array}{lll} i = 4 & j = 3 & \text{key} = 20 \\ A[j] = 40 & & A[j+1] = 20 \end{array}$$

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 20 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 3$ | $key = 20$    |
| $A[j] = 40$ |         | $A[j+1] = 20$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 3$ | $key = 20$    |
| $A[j] = 40$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 3$ | $key = 20$    |
| $A[j] = 40$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

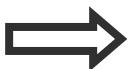


## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 2$ | $key = 20$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 40 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 2$ | $key = 20$    |
| $A[j] = 30$ |         | $A[j+1] = 40$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |                   |
|-------------|---------|-------------------|
| $i = 4$     | $j = 2$ | $\text{key} = 20$ |
| $A[j] = 30$ |         | $A[j+1] = 30$     |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1  | 2  | 3  | 4  |

$i = 4 \quad j = 2 \quad \text{key} = 20$   
 $A[j] = 30 \quad A[j+1] = 30$

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

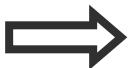


## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 1$ | $key = 20$    |
| $A[j] = 10$ |         | $A[j+1] = 30$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 30 | 30 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 1$ | $key = 20$    |
| $A[j] = 10$ |         | $A[j+1] = 30$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



## Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 20 | 30 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 1$ | $key = 20$    |
| $A[j] = 10$ |         | $A[j+1] = 20$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```



# Örnek : Insertion Sort

|    |    |    |    |
|----|----|----|----|
| 10 | 20 | 30 | 40 |
| 1  | 2  | 3  | 4  |

|             |         |               |
|-------------|---------|---------------|
| $i = 4$     | $j = 1$ | $key = 20$    |
| $A[j] = 10$ |         | $A[j+1] = 20$ |

```
InsertionSort(A, n) {
 for i = 2 to n {
 key = A[i]
 j = i - 1;
 while (j > 0) and (A[j] > key) {
 A[j+1] = A[j]
 j = j - 1
 }
 A[j+1] = key
 }
}
```

Bitti!

# Hatırlatma

## Çalışma Zamanı

- Çalışma zamanı veya koşma süresi girişe bağlıdır: Önceden sıralanmış bir diziyi sıralamak daha kolaydır.
- Çalışma zamanının girişin boyutuna göre parametrelenmesi yararlıdır, çünkü kısa dizileri sıralamak uzun dizilere oranla daha kolaydır.
- Genellikle, çalışma zamanında üst sınır aranır.

# Hatırlatma

## Çözümleme türleri

- **En kötü durum (Worst-case):** (genellikle bununla ilgilenilecek)
  - $T(n) = n$  boyutlu bir girişte algoritmanın maksimum süresi. (Programın her durumda çalışması)
- **Ortalama durum:** (bazen ilgilenilecek)
  - $T(n) = n$  boyutlu her girişte algoritmanın beklenen süresi (ağırıklı ortalama).
  - Girişlerin istatistiksel dağılımı için varsayıım gereklili.
  - En çok kullanılan varsayımlardan biri  $n$  boyutunda her girişin eşit oranda olasılığının olduğunu kabul etmek.
- **En iyi durum:** (gerçek dışı)
  - Sadece bir giriş yapısında hızlı çalışan fakat yavaş bir algoritma ile hile yapmak.

# Hatırlatma

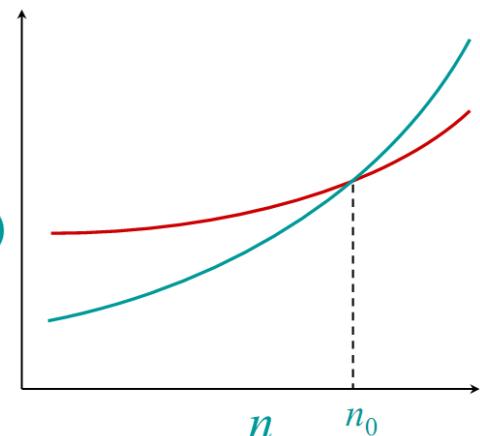
## Makineden-bağımsız zaman

- Araya yerleştirme sıralamasının en kötü zamanı nedir?
- Bilgisayarın hızına bağlıdır:
  - bağıl ( rölatif ) zaman ( aynı makinede),
  - mutlak (absolut ) zaman (farklı makinelerde).
- **BÜYÜK FİKİR:**
  - Makineye bağlı sabitleri görmezden gel.
  - $n \rightarrow \infty$  'a yaklaşıkça,  $T(n)$ 'nin büyümeye bak.  
" Asimptotik Çözümleme"

# Hatırlatma

## Asimptotik başarım

- $n$  yeterince büyürse,  $\Theta(n^2)$  algoritması bir  $\Theta(n^3)$  algoritmasından her zaman daha hızlıdır.
- Öte yandan asimptotik açıdan yavaş  $T(n)$  algoritmaları ihmal etmemeliyiz.
- Asimptotik çözümleme düşüncemizi yapılandırmada önemli bir araçtır.



# Insertion sort algoritmasının analizi

// sort in increasing order //

|                                                            | <u>cost</u> | <u>times</u>                       |
|------------------------------------------------------------|-------------|------------------------------------|
| 1 <b>for</b> $j \leftarrow 2$ <b>to</b> $\text{length}[A]$ | $c_1$       | $n$                                |
| 2 <b>do</b> $\text{key} \leftarrow A[j]$                   | $c_2$       | $n-1$                              |
| // insert $A[j]$ into the sorted sequence $A[1..j-1]$      |             |                                    |
| 3 $i \leftarrow j-1$                                       | $c_4$       | $n-1$                              |
| 4 <b>while</b> $i > 0$ <b>and</b> $A[i] > \text{key}$      | $c_5$       | $\sum_{2 \leq j \leq n} t_j$       |
| 5 <b>do</b> $A[i+1] \leftarrow A[i]$                       | $c_6$       | $\sum_{2 \leq j \leq n} (t_j - 1)$ |
| 6 $i \leftarrow i-1$                                       | $c_7$       | $\sum_{2 \leq j \leq n} (t_j - 1)$ |
| <b>done</b>                                                |             |                                    |
| 7 $A[i+1] \leftarrow \text{key}$                           | $c_8$       | $n-1$                              |
| <b>done</b>                                                |             |                                    |

# Insertion sort algoritmasının analizi

|                                                            | <u>cost</u> | <u>times</u>                       |
|------------------------------------------------------------|-------------|------------------------------------|
| 1 <b>for</b> $j \leftarrow 2$ <b>to</b> $\text{length}[A]$ | $c_1$       | $n$                                |
| 2 <b>do</b> $\text{key} \leftarrow A[j]$                   | $c_2$       | $n-1$                              |
| // insert $A[j]$ into the sorted sequence $A[1..j-1]$      |             |                                    |
| 3 $i \leftarrow j-1$                                       | $c_4$       | $n-1$                              |
| 4 <b>while</b> $i > 0$ <b>and</b> $A[i] > \text{key}$      | $c_5$       | $\sum_{2 \leq j \leq n} t_j$       |
| 5 <b>do</b> $A[i+1] \leftarrow A[i]$                       | $c_6$       | $\sum_{2 \leq j \leq n} (t_j - 1)$ |
| 6 $i \leftarrow i-1$                                       | $c_7$       | $\sum_{2 \leq j \leq n} (t_j - 1)$ |
| <b>done</b>                                                |             |                                    |
| 7 $A[i+1] \leftarrow \text{key}$                           | $c_8$       | $n-1$                              |
| <b>done</b>                                                |             |                                    |

- Insert sort algoritmasının çalışma zamanı ( $T(n)$ ), cost(maliyet), times (tekrar) toplamıdır.

$$T(n)$$

$$\begin{aligned}
 &= c_1 n + c_2(n-1) + c_4(n-1) \\
 &+ c_5 \sum_{j=2}^n t_j + c_6 \sum_{j=2}^n (t_j - 1) + c_7 \sum_{j=2}^n (t_j - 1) + c_8(n-1)
 \end{aligned}$$

## Araya yerleştirme sıralaması

- **En iyi durum:** dizi sıralı ise
- $j=2, \dots, n$  için  $t_j=1$  ise

$$T(n) = c_1n + c_2(n - 1) + c_4(n - 1) + c_5(n - 1) + c_8(n - 1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$$T(n) = an + b, \text{ Lineer fonksiyon}$$

$$T(n) = \theta(n)$$

## Araya yerleştirme sıralaması

- **En kötü durum:** dizi ters sıralı
- $j=2, \dots, n$  için  $t_j=j$  ise

$$\sum_{j=2}^n j = \frac{n(n+1)}{2} - 1 \text{ ve } \sum_{j=2}^n j - 1 = \frac{n(n-1)}{2}$$

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

$$T(n) = an^2 + bn + c \quad (\text{Quadratic Function})$$

$$T(n) = \theta(n^2)$$

## Araya yerleştirme sıralaması analizi özeti

- En kötü durum: Giriş tersten sıralıysa.

$$T(n) = \sum_{j=2}^n \Theta(j) = \Theta(n^2) \quad [\text{aritmetik seri}]$$

- Ortalama durum: Tüm permutasyonlar eşit olasılıklı.

$$T(n) = \sum_{j=2}^n \Theta(j/2) = \Theta(n^2)$$

- Araya yerleştirme sıralaması hızlı bir algoritma mıdır ?
  - Küçük n değerleri için olabilir.
  - Büyük n değerleri için asla!

## Bazı fonksiyonların büyümeye oranları

- Aşağıda verilen fonksiyonlar için yandaki kurallar uygulanacak:

$$1. \quad f(n) = \sum_{1 \leq i \leq n} i = n(n+1)/2$$

$$2. \quad f(n) = \sum_{1 \leq i \leq n} i^2 = n(n+1)(2n+1)/6$$

$$3. \quad f(n) = \sum_{0 \leq i \leq n} x^i = (x^{n+1}-1) / (x - 1)$$

$$4. \quad f(n) = \sum_{0 \leq i \leq n} 2^i = (2^{n+1}-1) / (2-1) = 2^{n+1}-1$$

$$\sum_{i=1}^n i = 1 + 2 + \dots + n = \frac{n * (n+1)}{2} \approx \frac{n^2}{2}$$

$$\sum_{i=0}^{n-1} 2^i = 0 + 1 + 2 + \dots + 2^{n-1} = 2^n - 1$$

$$\sum_{i=1}^n i^2 = 1 + 4 + \dots + n^2 = \frac{n * (n+1) * (2n+1)}{6} \approx \frac{n^3}{3}$$

## Büyüme oranı (Growth-Rate) fonksiyonları

### Örnek-1

```
for k = 1 to n/2 do
{
 -----> c1
}
for j = 1 to n*n do
{
 ---- ... -----> c2
}

$$\sum_{k=1}^{n/2} c_1 + \sum_{j=1}^{n*n} c_2 = c_1 n/2 + c_2 n^2 = O(n^2)$$

```

## Büyüme oranı (Growth-Rate) fonksiyonları Örnek-2

```
for k = 1 to n/2 do
{
 for j = 1 to n*n do
 {

 }
}
```

$$\sum_{k=1}^{n/2} \sum_{j=1}^{n*n} c = c n/2 * n^2 = O(n^3)$$

## Büyüme oranı (Growth-Rate) fonksiyonları

### Örnek-3

|                  | <u>Cost</u> | <u>Times</u> |
|------------------|-------------|--------------|
| i = 1;           | $c_1$       | 1            |
| sum = 0;         | $c_2$       | 1            |
| while (i <= n) { | $c_3$       | n+1          |
| i = i + 1;       | $c_4$       | n            |
| sum = sum + i; } | $c_5$       | n            |

$$\begin{aligned}
 T(n) &= c_1 + c_2 + c_3(n+1) + c_4n + c_5n \\
 &= (c_3 + c_4 + c_5)n + (c_1 + c_2 + c_3)
 \end{aligned}$$

$$T(n) = an + b \rightarrow O(n)$$

$T(n) \rightarrow$  Bu algoritmanın büyümeye oran fonksiyonu,  $O(n)$  dir

# Büyüme oranı (Growth-Rate) fonksiyonları

## Örnek-4

|                  | <u>Cost</u> | <u>Times</u> |
|------------------|-------------|--------------|
| i=1;             | $c_1$       | 1            |
| sum = 0;         | $c_2$       | 1            |
| while (i <= n) { | $c_3$       | $n+1$        |
| j=1;             | $c_4$       | $n$          |
| while (j <= n) { | $c_5$       | $n*(n+1)$    |
| sum = sum + i;   | $c_6$       | $n*n$        |
| j = j + 1;       | $c_7$       | $n*n$        |
| }                |             |              |
| i = i +1;        | $c_8$       | $n$          |
| }                |             |              |

$$\begin{aligned}
 T(n) &= c_1 + c_2 + c_3(n+1) + c_4n + c_5n(n+1) + c_6n(n) + c_7n(n) + c_8n \\
 &= (c_5 + c_6 + c_7)n^2 + (c_3 + c_4 + c_5 + c_8)n + (c_1 + c_2 + c_3) \\
 &= an^2 + bn + c
 \end{aligned}$$

→ Bu algoritmanın büyümeye oran fonksiyonu,  $O(n^2)$  dir.

## Büyüme oranı (Growth-Rate) fonksiyonları

### Örnek-5

- `for (int i = 0; i < N; i++)`
- `for (int j = i+1; j < N; j++)`
- `if (a[i] + a[j] == 0)` ←
- `count++;`

$$\begin{aligned}0 + 1 + 2 + \dots + (N - 1) &= \frac{1}{2} N(N - 1) \\&= \binom{N}{2}\end{aligned}$$

## Büyüme oranı (Growth-Rate) fonksiyonları

### Örnek-6

- int count = 0;
- for (int i = 0; i < N; i++)
- for (int j = i+1; j < N; j++)
- for (int k = j+1; k < N; k++)
- if (a[i] + a[j] + a[k] == 0)
- count++;

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!}$$
$$\sim \frac{1}{6}N^3$$

# Büyüme oranı (Growth-Rate) fonksiyonları

## Örnek-7

|                                      | <u>Cost</u> | <u>Times</u>                      |
|--------------------------------------|-------------|-----------------------------------|
| <code>for (i=1; i&lt;=n; i++)</code> | $c_1$       | $\sum_{j=1}^{n+1} (j+1)$          |
| <code>for (j=1; j&lt;=i; j++)</code> | $c_2$       | $\sum_{j=1}^n \sum_{k=1}^j (k+1)$ |
| <code>for (k=1; k&lt;=j; k++)</code> | $c_3$       | $\sum_{j=1}^n \sum_{k=1}^j k$     |
| <code>x=x+1;</code>                  | $c_4$       |                                   |

$$\begin{aligned}
 T(n) &= c_1(n+1) + c_2 \sum_{j=1}^n (j+1) + c_3 \sum_{j=1}^n \sum_{k=1}^j (k+1) + c_4 \sum_{j=1}^n \sum_{k=1}^j k \\
 &= an^3 + bn^2 + cn + d
 \end{aligned}$$

→ Bu algoritmanın büyümeye oran fonksiyonu,  $O(n^3)$  dir.

## Büyüme oranı (Growth-Rate) fonksiyonları

### Örnek-8

```

for i = 1 to n do
 for j = i to n do
 for k = i to j do
 m = m + i + j + k

```

$$\begin{aligned}
 T(n) &= \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j 3 \\
 &= \sum_{i=1}^n \sum_{j=i}^n 3(j-i+1) = \sum_{i=1}^n 3 \left[ \sum_{j=i}^n j - \sum_{j=i}^n i + \sum_{j=i}^n 1 \right] \\
 &= \sum_{i=1}^n 3 \left[ \left( \sum_{j=1}^n j - \sum_{j=1}^{i-1} j \right) - i(n-i+1) + (n-i+1) \right] \\
 &= \sum_{i=1}^n 3 \left[ [n(n+1)/2 - i(i-1)/2] - i(n-i+1) + (n-i+1) \right] \\
 &\approx n^3/10 \text{ additions, that is, } O(n^3)
 \end{aligned}$$

# **Özyinelemeli Algoritmaların Analizi**

- Yerine koyma metodu
- Yineleme döngüleri
- Özyineleme ağaçları
- Ana Metot (Master metod)

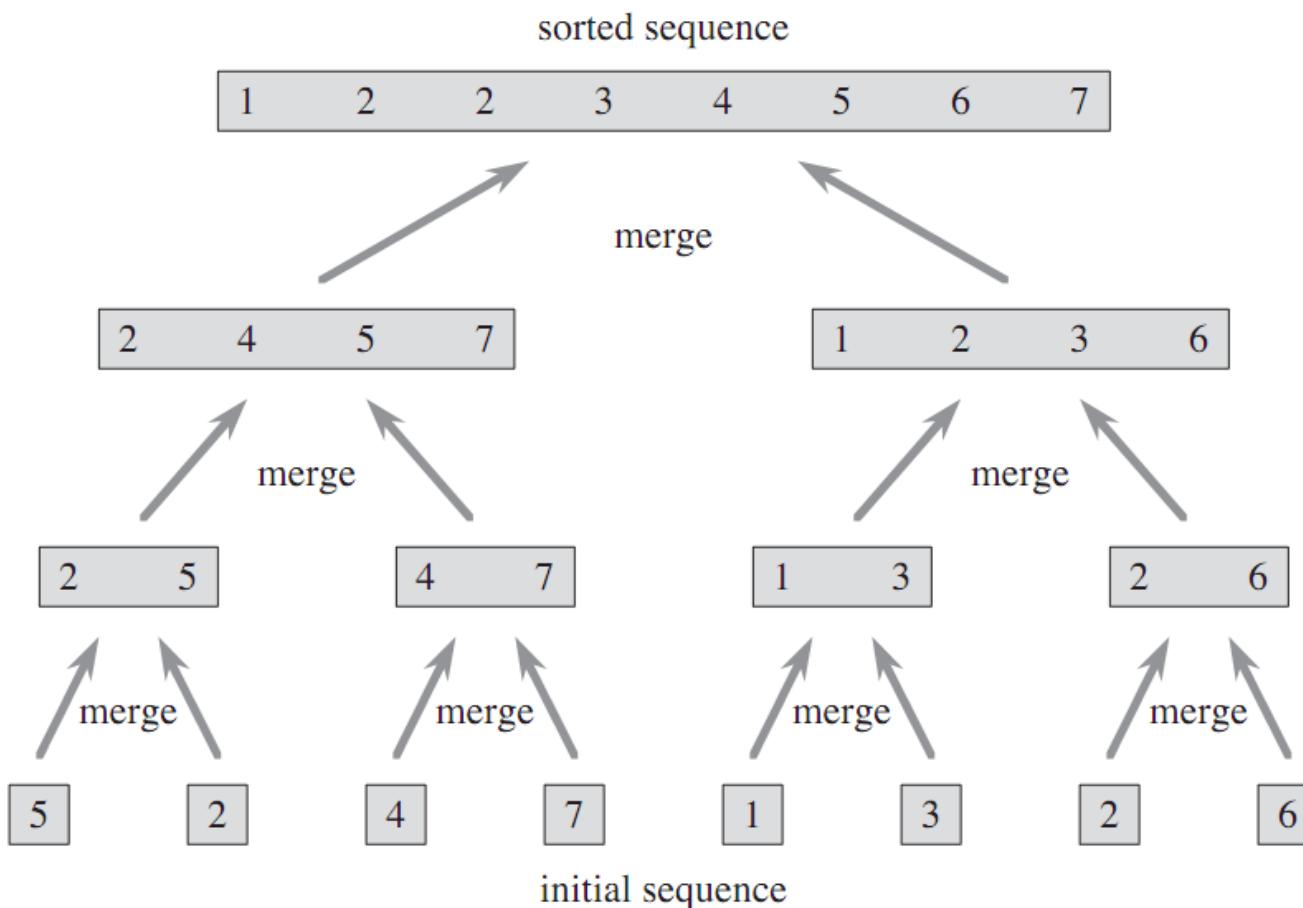
## Hatırlatma-Özyinelemeli Tanımlar

- Bir dizi, seri, fonksiyon ve algoritmanın kendi cinsinden tanımlanmasına **özyineleme** denir.
- Tanım bölgesi negatif tamsayılar olmayan fonksiyon tanımlanırken:
  - **Temel Adım:** Fonksiyonun sıfırdaki değeri belirtilir.
  - **Özyinelemeli adım:** Fonksiyonun bir tamsayıdaki değeri hesaplanırken, fonksiyonun daha küçük tamsayılardaki değer(ler)ini kullanarak bu değeri veren kural belirtilir.
    - İkinin kuvvetlerinden oluşan dizi aşağıdaki gibi ifade edilebilir
    - $a_n=2^n$
    - Fakat bu dizi özyinelemeli olarak aşağıdaki gibi ifade edilebilir.
    - $a_0=1, a_{n+1}=2a_n$

## Hatırlatma-Özyinelemeli Tanımlar

- Örnek:  $f$  fonksiyonu öz yinelemeli olarak aşağıdaki tanımlanmış olsun;
- $f(0)=3$ , temel durum
- $f(n+1)=2f(n)+3$ ,  $f(1)$ ,  $f(2)$ ,  $f(3)$  değerleri nedir?
  - $f(1) = 2f(0) + 3 = 2 * 3 + 3 = 9$
  - $f(2) = 2f(1) + 3 = 2 * 9 + 3 = 21$
  - $f(3) = 2f(2) + 3 = 2 * 21 + 3 = 45$
  - $f(4) = 2f(3) + 3 = 2 * 45 + 3 = 93$

# Birleştirme sıralaması-Merge Sort



## Birleştirme sıralaması-Merge Sort

**BİRLEŞTİRME-SIRALAMASI**  $A[1 \dots n]$

1. Eğer  $n = 1$  ise, işlem bitti.
2.  $A[1 \dots \lceil n/2 \rceil]$  ve  $A[\lceil n/2 \rceil + 1 \dots n]$ 'yi özyinelemeli sırala.
3. 2 sıralanmış listeyi “**Birleştir**”.

*Anahtar altyordam:* Birleştirme

# Birleştirme sıralaması-Merge Sort

MERGE-SORT( $A, p, r$ )

**if**  $p < r$

$$q = \lfloor (p + r)/2 \rfloor$$

MERGE-SORT( $A, p, q$ )

MERGE-SORT( $A, q + 1, r$ )

MERGE( $A, p, q, r$ )

Merge(), A dizisinin iki tane sıralı alt dizisini alır ve onları tek bir sıralı alt dizi içerisinde birleştirir.

Bu işlem ne kadar zaman alır?

MERGE( $A, p, q, r$ )

$$n_1 = q - p + 1$$

$$n_2 = r - q$$

let  $L[1..n_1 + 1]$  and  $R[1..n_2 + 1]$  be new arrays

**for**  $i = 1$  **to**  $n_1$

$$L[i] = A[p + i - 1]$$

**for**  $j = 1$  **to**  $n_2$

$$R[j] = A[q + j]$$

$$L[n_1 + 1] = \infty$$

$$R[n_2 + 1] = \infty$$

$$i = 1$$

$$j = 1$$

**for**  $k = p$  **to**  $r$

**if**  $L[i] \leq R[j]$

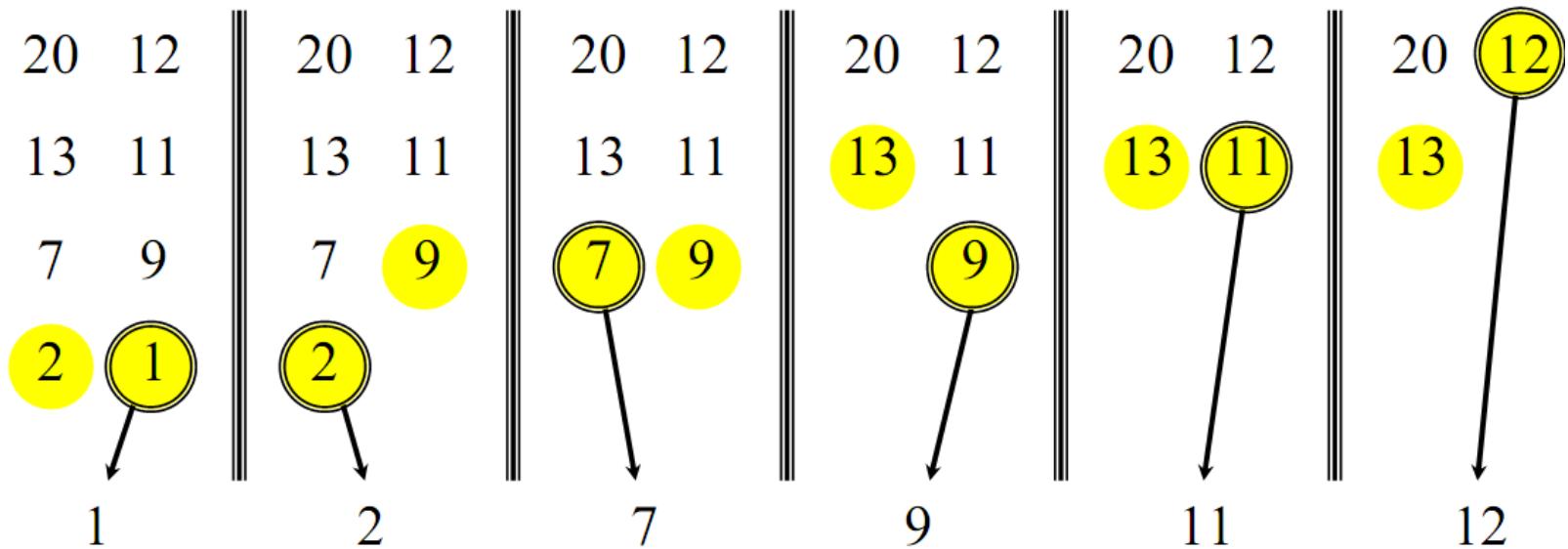
$$A[k] = L[i]$$

$$i = i + 1$$

**else**  $A[k] = R[j]$

$$j = j + 1$$

## Sıralı iki dizilimi birleştirme



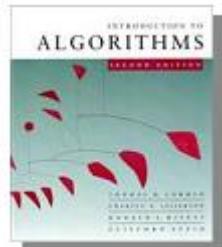
Süre =  $\Theta(n)$ , toplam  $n$  elemanı  
birleştirmek için (doğrusal zaman).

## Birleştirme sıralaması-Merge Sort

- `MergeSort(A, left, right) {`  $T(n)$
- `if (left < right) {`  $\Theta(1)$
- `mid = floor((left + right) / 2);`  $\Theta(1)$
- `MergeSort(A, left, mid);`  $T(n/2)$
- `MergeSort(A, mid+1, right);`  $T(n/2)$
- `Merge(A, left, mid, right);`  $\Theta(n)$
- `}`
- `}`
  
- Birleştirme sıralaması çözümlemesi bir yinelemeyi çözmemizi gerektirir.

## Yinelemeler –(Reküranslar)

- Yinelemeli algoritmalar çalışma süresi reküranslar kullanılarak tanımlanabilir.
- Bir **yineleme** herhangi bir fonksiyonu kendisinin küçük girişleriyle tanımlar.
- Yinelemeleri çözmek için genel bir prosedür yada yöntem yoktur.
- Malesef yinelemeleri çözmek için iyi bir algoritma da yoktur. Sadece birtakım teknikler vardır. Bunların bazıları bazen işe yarar ve eğer şanslıysanız sizin yinelemeniz için bunlardan biri çalışır.



## Sorular

- 1- Asimptotik notasyonlardan hangilerinin geçişme özelliği (transitivity) vardır.
  
- 2- $f(n)=n^2+4n\log n+900$  ve  $g(n)=n^2+45\log n+h(n)$  fonksiyonları verilmiştir ve  $h(n)$  lineer olan bir polinomdur.  $f(n)$  ile  $g(n)$  arasındaki asimptotik ilişki nedir?  $f(n)$  ve  $g(n)$  arasındaki asimptotik ilişkiyi belirlerken  $h(n)$  polinomuna ihtiyaç var mıdır? Hangi durumlarda ihtiyaç duyulur veya duyulmaz?

$$1. T(n) = T\left(\frac{n}{4}\right) + \sqrt{n} + \sqrt{6046} \cdot O(\sqrt{n})$$

$$1 + x + x^2 + \dots + x^n = \frac{1 - x^{n+1}}{1 - x} ; x \neq 1 \text{ için}$$

$$1 + x + x^2 + \dots = \frac{1}{1 - x} ; |x| < 1 \text{ için}$$

## Bazı Matematiksel İfadeler

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

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

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

- İki sınır arasındaki sayıların toplamı:  $\sum_{i=a}^b f(i) = \sum_{i=0}^b f(i) - \sum_{i=0}^{a-1} f(i)$

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

- Combinatorics

1. Number of permutations of an  $n$ -element set:  $P(n) = n!$
2. Number of  $k$ -combinations of an  $n$ -element set:  $C(n, k) = \frac{n!}{k!(n-k)!}$
3. Number of subsets of an  $n$ -element set:  $2^n$

# Bazı Matematiksel İfadeler

## ● Properties of Logarithms

1.  $\log_a 1 = 0$
2.  $\log_a a = 1$
3.  $\log_a x^y = y \log_a x$
4.  $\log_a xy = \log_a x + \log_a y$
5.  $\log_a \frac{x}{y} = \log_a x - \log_a y$
6.  $a^{\log_b x} = x^{\log_b a}$
7.  $\log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$

# Bazı Önemli Matematiksel İfadeler

## ○ Important Summation Formulas

$$1. \sum_{i=l}^u 1 = \underbrace{1 + 1 + \cdots + 1}_{u-l+1 \text{ times}} = u - l + 1 \quad (l, u \text{ are integer limits}, l \leq u); \quad \sum_{i=1}^n 1 = n$$

$$2. \sum_{i=1}^n i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$$

$$3. \sum_{i=1}^n i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$$

$$4. \sum_{i=1}^n i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1}n^{k+1}$$

$$5. \sum_{i=0}^n a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \quad (a \neq 1); \quad \sum_{i=0}^n 2^i = 2^{n+1} - 1$$

$$6. \sum_{i=1}^n i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2 \quad \sum_{i=0}^{n-1} ix^i = x + 2x^2 + 3x^3 + \cdots + nx^n = \frac{(n-1)x^{(n+1)} - nx^n + x}{(x-1)^2}.$$

$$7. \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma, \text{ where } \gamma \approx 0.5772 \dots \text{ (Euler's constant)}$$

$$8. \sum_{i=1}^n \lg i \approx n \lg n$$

# Bazı Önemli Matematiksel İfadeler

$$1. \sum_{k=1}^n k = 1+2+3+\dots+n = \frac{n(n+1)}{2}$$

$$2. \sum_{k=1}^n 2k = 2+4+6+\dots+2n = n(n+1)$$

$$3. \sum_{k=1}^n (2k-1) = 1+3+5+\dots+(2n-1) = n^2$$

$$4. \sum_{k=1}^n k^2 = 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$5. \sum_{k=1}^n k^3 = 1^3 + 2^3 + 3^3 + \dots + n^3 = \left[ \frac{n(n+1)}{2} \right]^2$$

$$6. \sum_{k=1}^n r^{k-1} = 1+r+r^2+r^3+\dots+r^{n-1} = \frac{1-r^n}{1-r}, \quad (r \neq 1)$$

$$7. \sum_{k=1}^n \frac{1}{k \cdot (k+1)} = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \dots + \frac{1}{n \cdot (n+1)} = \frac{n}{n+1}$$

$$8. \sum_{k=1}^n k \cdot k! = (n+1)! - 1$$

# Bazı Matematiksel İfadeler

## ● Sum Manipulation Rules

$$1. \sum_{i=l}^u ca_i = c \sum_{i=l}^u a_i$$

$$2. \sum_{i=l}^u (a_i \pm b_i) = \sum_{i=l}^u a_i \pm \sum_{i=l}^u b_i$$

$$3. \sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=m+1}^u a_i, \text{ where } l \leq m < u$$

$$4. \sum_{i=l}^u (a_i - a_{i-1}) = a_u - a_{l-1}$$

## Approximation of a Sum by a Definite Integral

$$\int_{l-1}^u f(x)dx \leq \sum_{i=l}^u f(i) \leq \int_l^{u+1} f(x)dx \quad \text{for a nondecreasing } f(x)$$

$$\int_l^{u+1} f(x)dx \leq \sum_{i=l}^u f(i) \leq \int_{l-1}^u f(x)dx \quad \text{for a nonincreasing } f(x)$$

## Floor and Ceiling Formulas

The *floor* of a real number  $x$ , denoted  $\lfloor x \rfloor$ , is defined as the greatest integer not larger than  $x$  (e.g.,  $\lfloor 3.8 \rfloor = 3$ ,  $\lfloor -3.8 \rfloor = -4$ ,  $\lfloor 3 \rfloor = 3$ ). The *ceiling* of a real number  $x$ , denoted  $\lceil x \rceil$ , is defined as the smallest integer not smaller than  $x$  (e.g.,  $\lceil 3.8 \rceil = 4$ ,  $\lceil -3.8 \rceil = -3$ ,  $\lceil 3 \rceil = 3$ ).

1.  $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
2.  $\lfloor x + n \rfloor = \lfloor x \rfloor + n$  and  $\lceil x + n \rceil = \lceil x \rceil + n$  for real  $x$  and integer  $n$
3.  $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
4.  $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$

## Miscellaneous

1.  $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$  as  $n \rightarrow \infty$  (Stirling's formula)
2. Modular arithmetic ( $n, m$  are integers,  $p$  is a positive integer)

$$(n+m) \bmod p = (n \bmod p + m \bmod p) \bmod p$$

$$(nm) \bmod p = ((n \bmod p)(m \bmod p)) \bmod p$$

# **4.Hafta**

## **Yinelemelerin çözümü**

Yerine Koyma, İterasyon, Master  
Metot

# **4.Hafta**

# **Algoritmaların**

# **Analizi**

**Yinelemelerin çözümü**

## Yinelemelerin çözümü

- Yinelemeler entegral, türev, v.s. denklemlerinin çözümlerine benzer. Yinelemeleri çözmek konusunda kullanacağımız 3 ana yöntem vardır. Bu yöntemler;
  - **Yerine koyma metodu (substitution)**
  - **İterasyon (Yineleme) metodu (iteration metod)**
    - **Özyineleme ağacı (recursion tree)**
  - **Ana Metot (Master metod)**
  - Ana yöntemlerin dışında tekrarlı bağıntılar **Karakteristik denklemler** kullanarak çözülebilir.

## Yerine koyma metodu (yöntemi)

- Bazı tekrarlı bağıntıların çözümü yapılmadan çözümünün nasıl olabileceği hakkında tahmin yapılabilir. Çözüm tahmini yapılır ve yerine konulur. Yerine koyma (tahminler) metodu bir sınırdır ve tahmini ispatlamak için tümevarım yöntemi kullanılır.
- En genel yöntem:
  - 1. Çözümün şeklini **tahmin** edin.
  - 2. Tümevarım ile **doğrulayın**.
  - 3. Sabitleri **çözün**.
- Örnek:  $T(n)=T(n/2)+c$ ,  $n \geq 2$ , ve  $T(1)=1$ .
- $T(2)=1+c$ ,  $T(4)=1+2c$ ,  $T(8)=1+3c$ , ...
- $T(2^k)=1+kc$ , burada  $n=2^k$ ,  $T(n)=1+c\log n$  olur.

# Yerine koyma metodu (yöntemi)

- Tümevarım ile İspat:
- Temel durum:  $n=1$ ,  $T(1)=1$  verildi.
- Tümevarım hipotezi:  $n=2^k$ ,  $T(2^k)=1+kc$ , burada  $n=2^k$ ,  
 $T(n)=T(n/2)+c$  için doğrudur.
- Tümevarım adımı:  $n= 2^{k+1}$ ,  $T(n+1)=1+(k+1)*c= (1+ kc)+c$
- İspat (proof):
- $T(n+1)=T(2^{k+1})=T(2^{k+1}/2)+c$
- $T(n+1)= T(2^k*2/2)+c)$
- $=T(2^k) + c= 1+ kc +c$ , olur. Tümevarım adımı ispatlanmış olunur.
- Burada,  $n=2^k$  için,  $T(n)=1+c\log n$  ve
- $T(n)\in O(\log n)$  dir.

## Yerine koyma metodu (yöntemi)

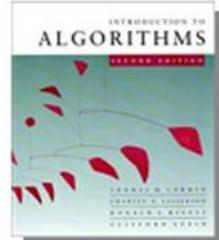
- Örnek:  $T(n)=3T(n/2)+cn$ ,  $n \geq 2$ , ve  $T(1)=1$ .
  - $T(2)=3+2c$
  - $T(4)=3(T(2))+4c=3(3+2c)+4c=9+10c=3^2+[3^12^1c]+3^02^2c$
  - $T(8)=27+38c=3^3+[3^22^1c+3^12^2c]+3^02^3c$
  - $T(16)=81+130c$
  - ...
  - $T(2^k)=3^k+[3^{k-1}2^1c+3^{k-2}2^2c+\dots+3^12^{k-1}c]+3^02^kc$ ,  $2^kc$  parantezine alalım
  - $T(2^k)=3^k+2^kc[(3/2)^{k-1}+(3/2)^{k-2}+\dots+(3/2)]$ , serisinin genel denklemi
  - $T(2^k)=3^k+2^kc[((3/2)^k-1)/((3/2)-1)]$ , burada  $n=2^k$  ve  $k=\log n$
  - $T(n)=3^{\log n}+cn[((3^{\log n})/n-1)/(1/2)]$ , burada  $a^{\log_b x} = x^{\log_b a}$
  - $T(n)=n^{\log 3}+2cn(n^{\log 3-1}-1) = n^{1,59}+2cn(n^{0,59}-1)$
  - $T(n)=n^{1,59}(1+2c)-2cn$
  - $T(n) \in O(n^{1,59})$  dır.
- $f(n) = \sum_{0 \leq i \leq n} x^i = (x^{n+1} - 1) / (x - 1)$
-

## Yerine koyma metodu (yöntemi)

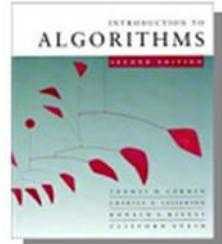
- İspat:  $T(n)=3T(n/2)+cn$ ,  $n \geq 2$ , ve  $T(1)=1$ . (1)
- $T(2^k)=3^k+2^k c[((3/2)^k-1)/((3/2)-1)]$ , burada  $n=2^k$  ve  $k=\log n$  (2)
- Temel durum:  $n=1$ ,  $T(1)=1$  ve  $n=2 \rightarrow T(2)=3+2c$
- Tümivarım hipotezi:  $n=2^k \rightarrow T(2^k)=3^k+2^k c[((3/2)^k-1)/((3/2)-1)]$
- Tümivarım adımı:  $n=2^{k+1} \rightarrow T(2^{k+1})=3^{k+1}+2^{k+1} c[((3/2)^{k+1}-1)/((3/2)-1)]$  veya  
 $\rightarrow T(2^{k+1})=3^{k+1}+2^{k+2} c[(3/2)^{k+1}2-2]$  (3)
- Şimdi (1) nolu denklemi kullanarak (3) nolu denklemi ispatlayalım
- $T(2^k)=3^k \cdot T(2^{k-1})+c2^k \rightarrow n=2^k$  için
- $T(2^{k+1})=3 \cdot T(2^k)+c2^{k+1}=3 \cdot (3^k+c2^k[((3/2)^k-1)/((3/2)-1)])+c2^{k+1}$
- $T(2^{k+1})=3^{k+1}+2^k c[3 \cdot (3/2)^k - 3]/(1/2) + 2^{k+1}c = 3^{k+1}+2^{k+1} c[3 \cdot (3/2)^k - 3] + 2^{k+1}c$
- $T(2^{k+1})=3^{k+1}+2^{k+1} c[3 \cdot (3/2)^k - 3] + 1 = 3^{k+1}+2^{k+1} c[(3/2)^{k+1} \cdot 2 - 2]$
- $T(2^{k+1})=3^{k+1}+2^{k+2} c[(3/2)^{k+1}2-2]$ , (3) nolu denklemi ispatlamış oluyoruz.
- $T(n) \in O(n^{1.59})$  dir.

# Yerine koyma metodu (yöntemi)

## Üst sınırı tahmin ederek çözüm



- Örnek:  $T(n) = 4T(n/2) + n$ , ise
  - [  $T(1) = \Theta(1)$  olduğunu varsayıñ.]
  - $O(n^3)$ 'ü tahmin edin. ( $O$  ve  $\Omega$  ayrı ayrı kanıtlayın.)
  - $k < n$  için  $T(k) \leq ck^3$  olduğunu varsayıñ.
  - $T(n) \leq cn^3$ 'ü tümevarımla kanıtlayın.



## Yerine koyma örneği

$$T(n) \leq cn^3$$

$$T(n) = 4T(n/2) + n$$

$$\leq 4c(n/2)^3 + n$$

$$= (c/2)n^3 + n$$

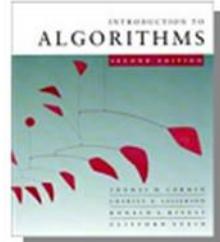
$$= cn^3 - ((c/2)n^3 - n) \leftarrow \text{istenen} - \text{kalan}$$

$$\leq cn^3 \leftarrow \text{istenen}$$

ne zaman ki  $(c/2)n^3 - n \geq 0$ , örneğin,

eğer  $c \geq 2$  ve  $n \geq 1$ .

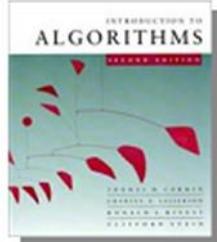
*kalan*



## Yerine koyma örneği

- Başlangıç koşullarını da ele almalı, yani, tümevarımı taban şıklarına (base cases) dayandırmalıyız.
- 
- **Taban:**  $T(n) = \Theta(1)$  tüm  $n < n_0$  için, ki  $n_0$  uygun bir sabittir.
- $1 \leq n < n_0$  için, elimizde " $\Theta(1)$ "  $\leq cn^3$ , olur; yeterince büyük bir  $c$  değeri seçersek.

**Bu, sıkı bir sınır değildir !**

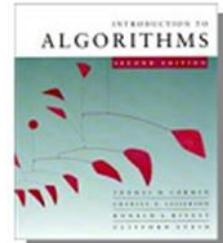


## Yerine koyma örneği-Daha sıkı bir üst sınır?

$T(n) = O(n^2)$  olduğunu kanıtlayacağız.

Varsayıñ ki  $T(k) \leq ck^2$ ,  $k < n$  için olsun :

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \end{aligned}$$



## Yerine koyma örneği-Daha sıkı bir üst sınır?

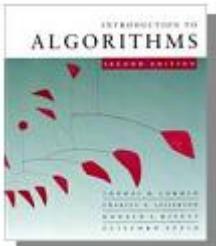
$T(n) = O(n^2)$  olduğunu kanıtlayacağız.

Varsayıñ ki  $T(k) \leq ck^2$ ;  $k < n$ : için

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= \cancel{O(n^2)} \quad \text{Yanlış! I.H.(tümeyarım hipotezini) kanıtlamalıyız.} \\ &= cn^2 - (-n) \quad [\text{istenen } -\text{kalan}] \\ &\leq cn^2 \quad \text{seçeneksiz durum } c > 0. \quad \text{Kaybettik!} \end{aligned}$$

$-n \geq 0$  sağlanmaz ( $n$  değeri negatif olamaz)

## Yerine koyma örneği-Daha sıkı bir üst sınır?



**FIKİR:** Varsayımdı hipotezini güçlendirin.

- Düşük-düzenli bir terimi *çıkartın*.

*Varsayımdı hipotezi:*  $T(k) \leq c_1k^2 - c_2k$  ;  $k < n$  için.

$$\begin{aligned}
 T(n) &= 4T(n/2) + n \\
 &= 4(c_1(n/2)^2 - c_2(n/2)) + n \\
 &= c_1n^2 - 2c_2n + n \\
 &= c_1n^2 - c_2n - (c_2n - n) \\
 &\leq c_1n^2 - c_2n \text{ eğer } c_2 \geq 1.
 \end{aligned}$$

$c_1$ 'ı başlangıç koşullarını karşılayacak kadar büyük seçin.

## Yerine koyma metodu (yöntemi)

- $T(n)=3T(n/2)+cn$ ,
- Tahminimiz = $T(k) \in O(n^{1,59})$ ,  $n \geq 2$ , ve  $T(1)=1$ .
- İspatı yapabilmek için tahminimizi  $T(n)$  de yerine yazıp istediğimizi elde etmeye çalışmak. **İstenen – kalan**. Kalan, 0'a eşit yada büyük olmalı
- $T(n) \leq c_1 \cdot n^{1,59}$  olması
- $T(n)=3 \cdot (c_1 n^{1,59}/2^{1,59})+cn$
- $T(n)=c_1 n^{1,59} - c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}+cn$
- **İstenen=** $c_1 n^{1,59}$ , **Kalan=**  $- c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}+cn$
- **İstenen – kalan=** $c_1 n^{1,59} - (c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}-cn)$
- **Kalan** →  $c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}-cn \geq 0$  sağlayacak  $c$ , ve  $n$  değeri var mı?
- **n=1 için**,  $c_1(2^{1,59}+3)/2^{1,59}-c \geq 0$ ,  $c \geq 1$ , ve  $c_1 \geq 2$ , için sağlarız

# Yerine koyma metodu (yöntemi)

- Hanoi kulesi

```
void hanoi(int n, char source, char dest, char spare) { Cost
 if (n > 0) {
 hanoi(n-1, source, spare, dest); c1
 cout << "Move top disk from pole " << source c2
 << " to pole " << dest << endl;
 hanoi(n-1, spare, dest, source); c3
 }
}
```

c4

when  $n=0$

$$T(0) = c_1$$

when  $n>0$

$$\begin{aligned} T(n) &= c_1 + c_2 + T(n-1) + c_3 + c_4 + T(n-1) \\ &= 2*T(n-1) + (c_1+c_2+c_3+c_4) \\ &= 2*T(n-1) + c \quad \leftarrow \text{recurrence equation for the growth-rate function of hanoi-towers algorithm} \end{aligned}$$

## Yerine koyma metodu (yöntemi)

- Hanoi kulesi
- $T(n) = 2T(n - 1) + 1$  kabul edersek en iyi durumda  $T(0)=0$  dır.
- $T(0) = 0, T(1) = 1, T(2) = 3, T(3) = 7, T(4) = 15, T(5) = 31, T(6) = 63, \dots$
- $T(0)=0$
- $T(1)=2T(0)+1=1$
- $T(2)=2T(1)+1=2+1$
- $T(3)=2T(2)+1=2.(2+1)+1=1+2+2^2$
- $T(4)=2T(3)+1=1+2+2^2+2^3$
- $T(n)=2T(n-1)+1=1+2+\dots 2^{n-1}$      $f(n) = \sum_{0 \leq i \leq n} 2^i = (2^{n+1}-1) / (2-1) = 2^{n+1}-1$
- $T(n)=2^n-1$
- olur

# İterasyon metodu (Tümden gelim)

- Bu yöntemde verilen bağıntının çözümünü bulmak için büyük endeksli terimin yerine küçük endeksli terim yazılarak, genel terimin çözümü için bir yargıya varılınca kadar bu işleme devam edilir veya başlangıç şartlarına kadar devam edilir.
- İterasyon metodu, bir toplam içerisinde yinelemeleri dönüştürür ve yinelemeleri çözmek için toplamları sınırlayıcı teknikleri kullanır.
  - Yineleme işlemini açık hale getir
  - Matematiksel işlemlerle göster
  - Toplamı hesapla.

$$T(n) = \begin{cases} \text{denemeyle çözülür, } n=1 \\ \text{Alt problem sayısı * } T(n/\text{alt problem boyutu}) + \text{bölme+birleşim, } n=1 \end{cases}$$

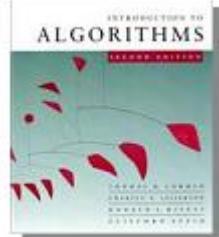
■ Örnek

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

# İterasyon metodu

- Örnek: Merge Sort,  $T(n)=2*T(n/2)+n$ ,  $n>1$ , ve  $T(1)=1$  için yinelemeyi çözünüz
- $n=2^k \rightarrow k = \log n$  hatırlayın,
  - $T(n)=2*T(n/2)+n$  substitute (yerine kullan)
  - $T(n)=2*(2*T(n/4)+n/2)+n$  expand
  - $T(n)=2^2*T(n/4)+2*n$  substitute
  - $T(n)=2^2*(2*T(n/8)+n/4)+2n =8*T(n/8)+3*n$  expand
  - $T(n)=2^3*T(n/2^3)+3*n$  observe
  - ....
  - $T(n)=2^k*T(n/n)+k*n$
  - $T(n)=2^k * T(1)+k*n$
  - $T(n)=\theta(n) + \theta(n\log n)$
  - $T(n) \in \Theta(n\log n)$

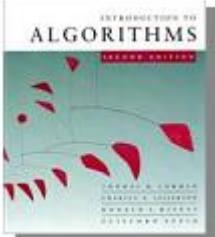
# İterasyon metodu



- Merge sort için daha kesin çalışma süresi bulma (bazı  $b$  değerleri için  $n=2^b$  olduğunu varsayılmı).

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}$$

$$T(n) = 5n + 2nlgn - 3$$



# İterasyon metodu

$$\begin{aligned}
 T(n) &= 2 T(n/2) + 2n + 3 \\
 &= 2(2 T(n/4) + n + 3) + 2n + 3 \\
 &= 2^2 T(n/4) + 4n + 2 \cdot 3 + 3 \\
 &= 2^2 (2 T(n/8) + n/2 + 3) + 4n + 2 \cdot 3 + 3 \\
 &= 2^3 (T(n/2^3) + 2 \cdot 3n + (2^2 + 2^1 + 2^0) \cdot 3) \\
 &= 2^b T(n/2^b) + 2 \cdot bn + 3 \sum_{j=0}^{b-1} 2^j \\
 &= n T(n/n) + 2n \lg n + 3(2^b - 1) \\
 &= 2n + 2n \lg n + 3n - 3 \\
 &= 5n + 2n \lg n - 3
 \end{aligned}$$

# İterasyon metodu

- Örnek:  $T(n)=T(n-1)+n$ ,  $n>1$ , ve  $T(1)=1$ ; için yinelemeyi çözünüz?
- 
- $T(n)=T(n-1) + n$
- $T(n)=T(n-2) + n-1+ n$
- $T(n)=T(n-3) + n-2+ n-1+ n$
- ...
- $T(n)=T(1) + 2+ \dots n-2+ n-1+ n$
- $T(n)=1 + 2+ \dots n-2+ n-1+ n \longrightarrow f(n) = \sum_{1 \leq i \leq n} i = n(n+1)/2$
- $T(n)=n*(n+1)/2$
- $T(n) \in \Theta(n^2)$

# İterasyon metodu

- Hanoi Kulesi
- $T(n)=2*T(n-1)+1$ ,  $n>1$ , ve  $T(1)=1$ ; için yinelemeyi çözünüz?
- Her bir hareket  $O(1)$  zaman gerektirir.
- 

- $T(n)=2*T(n-1)+1= 2^2 * T(n-2)+2+1 = \dots$

- $T(n)=2^{n-1} * T(1)+2^{n-2}+\dots+2+1$

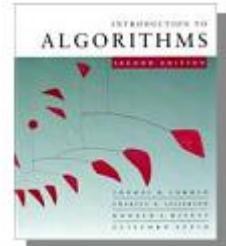
- $T(n)=2^{n-1}$

- $T(n) \in \Theta(2^{n-1})$

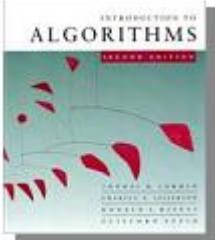
$$f(n) = \sum_{0 \leq i \leq n} 2^i = (2^{n+1}-1) / (2-1) = 2^{n+1}-1$$

- $n=64$  için, her 1000 taşımının 1 sn olduğu düşünülürse çözümü  $584*10^6$  yıl alır.

# Özyineleme-ağacı metodu

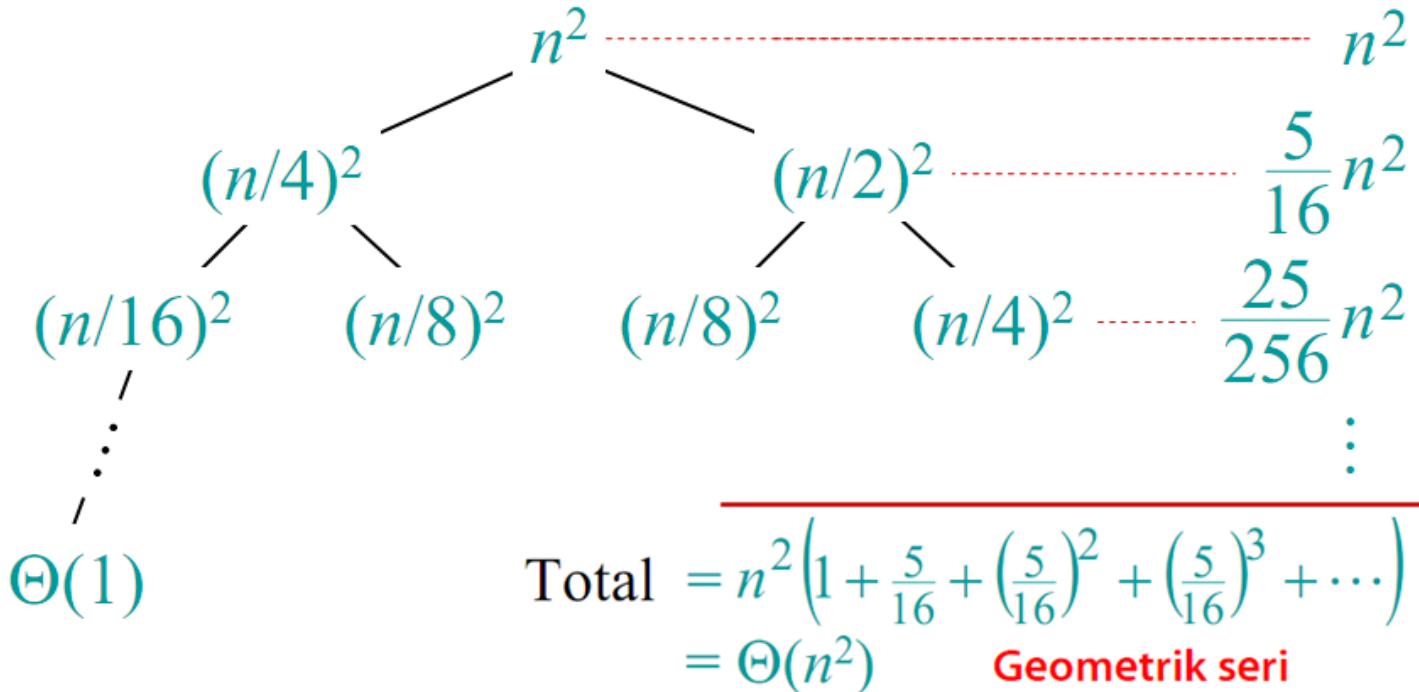


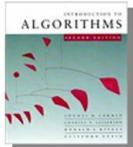
- İterasyon yönteminin çözüm adımları ağaç şeklinde gösterilebilir ve elde edilen ağaca Özyineleme-ağacı denir
- Özyineleme-ağacı, bir algoritmadaki özyineleme uygulamasının maliyetini (zamanı) modeller.
- Özyineleme-ağacı metodu, diğer yöntemler gibi, güvenilir olmayıabilir.
- Öte yandan özyineleme-ağacı metodu "öngörü" olusunu geliştirir.
- Özyineleme-ağacı metodu "yerine koyma metodu" için gerekli tahminlerinde yararlıdır.



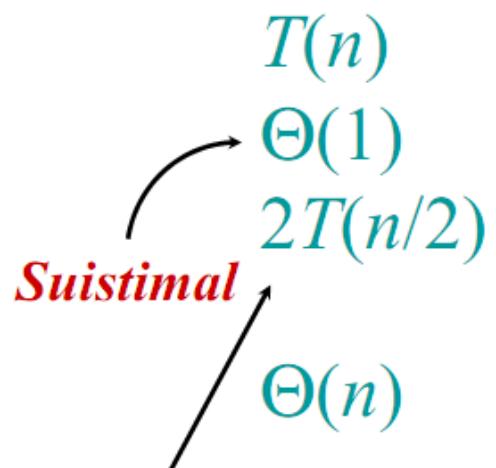
# Özyineleme-ağacı örneği

$$T(n) = T(n/4) + T(n/2) + n^2:$$





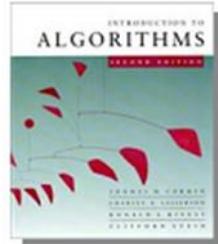
# Birleştirme sıralamasının çözümlenmesi



**BİRLEŞTİRME-SIRALAMASI**  $A[1 \dots n]$

1. Eğer  $n = 1$ 'se, bitir.
2. Yinelemeli olarak  $A[1 \dots \lceil n/2 \rceil]$  ve  $A[\lceil n/2 \rceil + 1 \dots n]$ 'yi sırala.
3. 2 sıralı listeyi "**Birleştir**"

**Özensizlik:**  $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$  olması gereklidir, ama asimptotik açıdan bu önemli değildir.



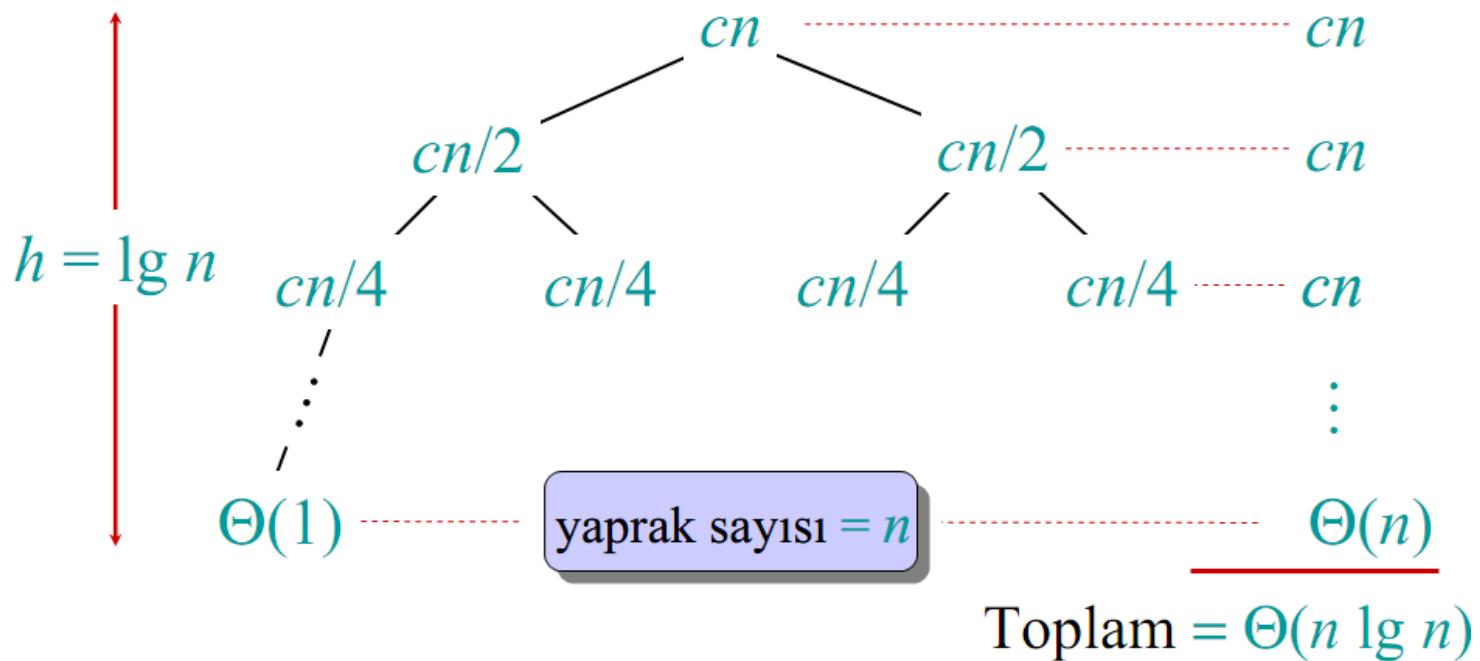
## Birleştirme sıralaması için yineleme

$$T(n) = \begin{cases} \Theta(1) & \text{eğer } n = 1 \text{ ise;} \\ 2T(n/2) + \Theta(n) & \text{eğer } n > 1 \text{ ise.} \end{cases}$$

- Genellikle  $n$ 'nin küçük değerleri için taban durumu ( base case ) olan  $T(n) = \Theta(1)$  'i hesaplara katmayacağız; ama bunu sadece yinelemenin asimptotik çözümünü etkilemiyorsa yapacağız.

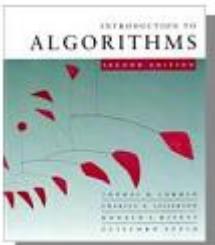
## Yineleme ağacı

- $T(n) = 2T(n/2) + cn$ 'yi çözün; burada  $c > 0$  bir sabittir.



## Sonuçlar- Insert Sort –Merge Sort

- $\Theta(n \lg n)$ 'nin, büyümeye oranı  $\Theta(n^2)$ 'den daha yavaştır (yani küçüktür).
- En kötü durumda, birleştirme sıralaması asimptotik olarak araya yerleştirme sıralamasından daha iyidir.
- Pratikte, birleştirme sıralaması(Merge Sort) araya yerleştirme sıralamasını (Insert Sort)  $n > 30$  değerlerinde geçer.



## Yinelemelerin çözümü için değişken değiştirme

- Reküranslar değişken değiştirme ile daha basit hale dönüştürülebilir.
- $T(n) = 2T(\sqrt{n}) + \log n.$

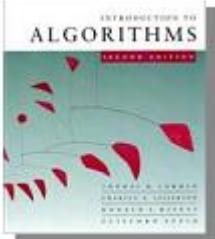
$$m = \log n \Rightarrow 2^m = n \Rightarrow \sqrt{n} = 2^{m/2}$$

$$T(n) = 2T(\sqrt{n}) + \log n \Rightarrow T(2^m) = 2T(2^{m/2}) + m$$

$$S(m) = T(2^m)$$

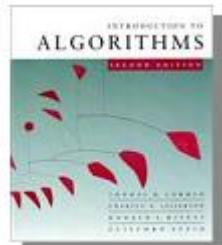
$$\begin{aligned} T(2^m) &= 2T(2^{m/2}) + m \Rightarrow S(m) = 2S(m/2) + m \\ &\Rightarrow S(m) = O(m \log m) \\ &\Rightarrow T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n) \end{aligned}$$

## Master Teorem



# Ana Metod (The Master Method)

- Ana method aşağıda belirtilen yapıdaki yinelemelere uygulanır:
- $T(n) = aT(n/b) + f(n)$  , burada  $a \geq 1$ ,  $b > 1$ , ve  $f$  asimptotik olarak pozitiftir.
- $T(n)$  bir algoritmanın çalışma süresidir.
  - $n/b$  boyutunda  $a$  tane alt problem recursive olarak çözülür ve her biri  $T(n/b)$  süresindedir.
  - $f(n)$  problemin bölünmesi ve sonuçların birleştirilmesi için geçen süredir.
  - Örnek: Merge-sort için  $T(n)=2T(n/2)+\Theta(n)$  yazılabilir.



# Ana Metod (The Master Method)

## Üç yaygın uygulama

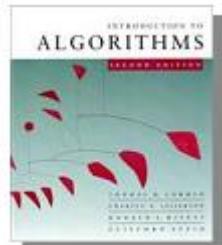
$f(n)$ 'i  $n^{\log_b a}$  ile karşılaştırın:

1.  $f(n) = O(n^{\log_b a - \varepsilon})$   $\varepsilon > 0$  sabiti durumunda;
  - $f(n)$  polinomsal olarak  $n^{\log_b a}$  göre daha yavaş büyür ( $n^\varepsilon$  faktörü oranında).

**Çözüm:**  $T(n) = \Theta(n^{\log_b a})$ .

2.  $f(n) = \Theta(n^{\log_b a} \lg^k n)$   $k \geq 0$  sabiti durumunda;
  - $f(n)$  ve  $n^{\log_b a}$  benzer oranlarda büyürler.

**Çözüm:**  $T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .



# Ana Metod (The Master Method)

## Üç yaygın uygulama

$f(n)$ 'i  $n^{\log_b a}$  ile karşılaştırın:

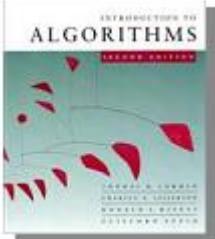
3.  $f(n) = \Omega(n^{\log_b a + \varepsilon})$   $\varepsilon > 0$  sabiti durumunda;

- $f(n)$  polinomsal olarak  $n^{\log_b a}$  'ye göre daha hızlı büyür ( $n^\varepsilon$  faktörü oranında),

ve  $f(n)$ , **düzenlilik koşulunu**  $af(n/b) \leq cf(n)$  durumunda,  $c < 1$  olmak kaydıyla karşılar.

**Çözüm:**  $T(n) = \Theta(f(n))$ .

$$c=(1-\varepsilon), \varepsilon > 0$$



## Örnekler

**Ör.**  $T(n) = 4T(n/2) + n$

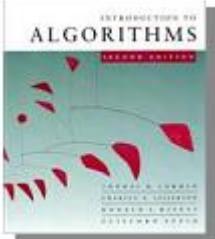
$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n.$$

**Durum 1:**  $f(n) = O(n^{2-\varepsilon})$      $\varepsilon = 1$  için.  
 $\therefore T(n) = \Theta(n^2)$ .

**Ör.**  $T(n) = 4T(n/2) + n^2$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2.$$

**Durum 2:**  $f(n) = \Theta(n^2 \lg^0 n)$ , yani,  $k = 0$ .  
 $\therefore T(n) = \Theta(n^2 \lg n)$ .



## Örnekler

**Ör.**  $T(n) = 4T(n/2) + n^3$

$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^3.$$

**DURUM 3:**  $f(n) = \Omega(n^{2+\varepsilon})$     $\varepsilon = 1$  için

ve  $4(n/2)^3 \leq cn^3$  (düz. koş.)    $c = 1/2$  için

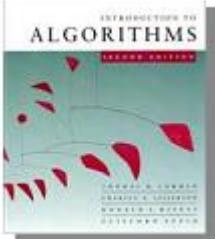
$$\therefore T(n) = \Theta(n^3).$$

**Ör.**  $T(n) = 4T(n/2) + n^2/\lg n$

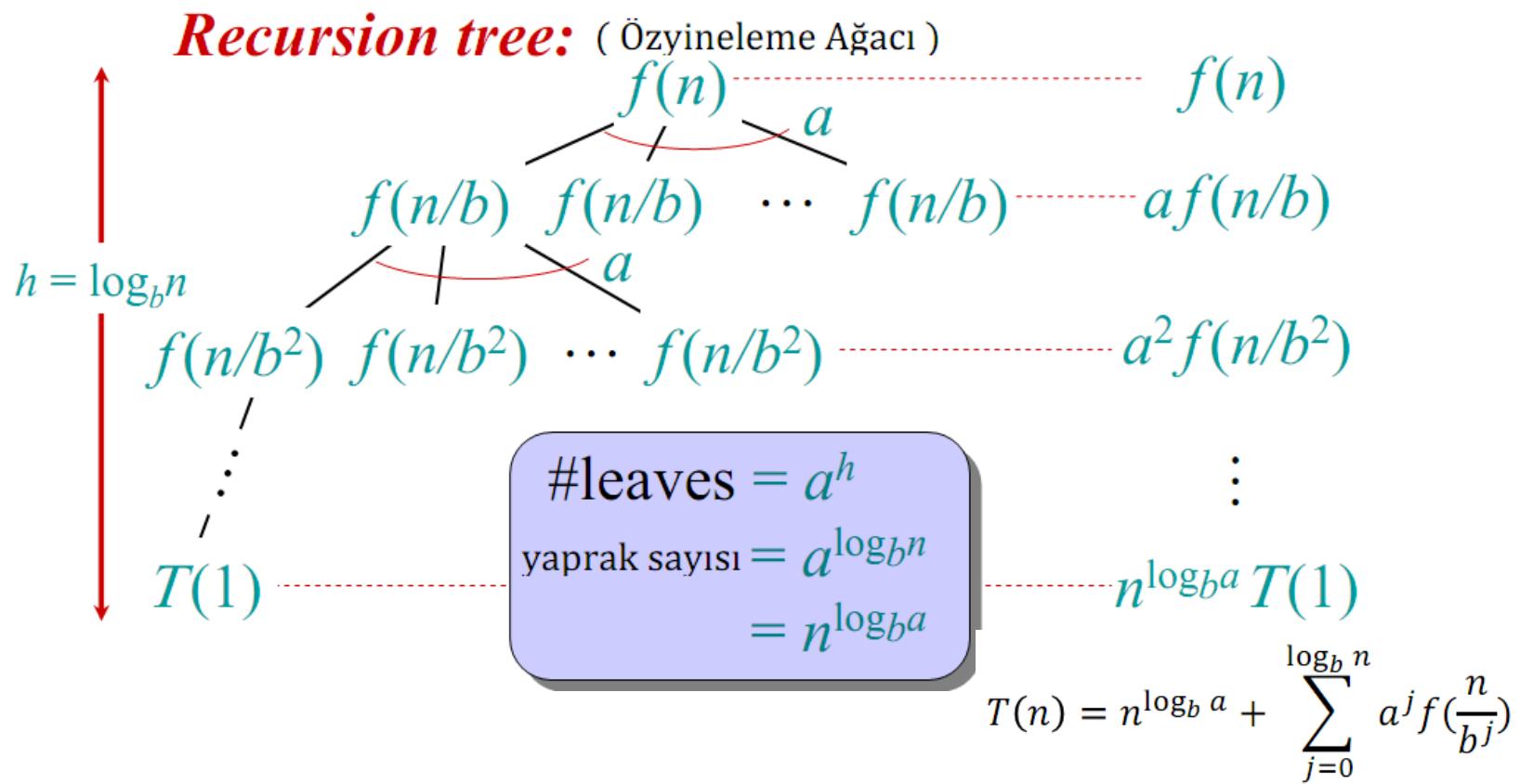
$$a = 4, b = 2 \Rightarrow n^{\log_b a} = n^2; f(n) = n^2/\lg n.$$

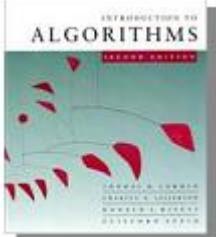
Ana metod geçerli değil. Özellikle,

$\varepsilon > 0$  olan sabitler için  $n^\varepsilon = \omega(\lg n)$  elde edilir.

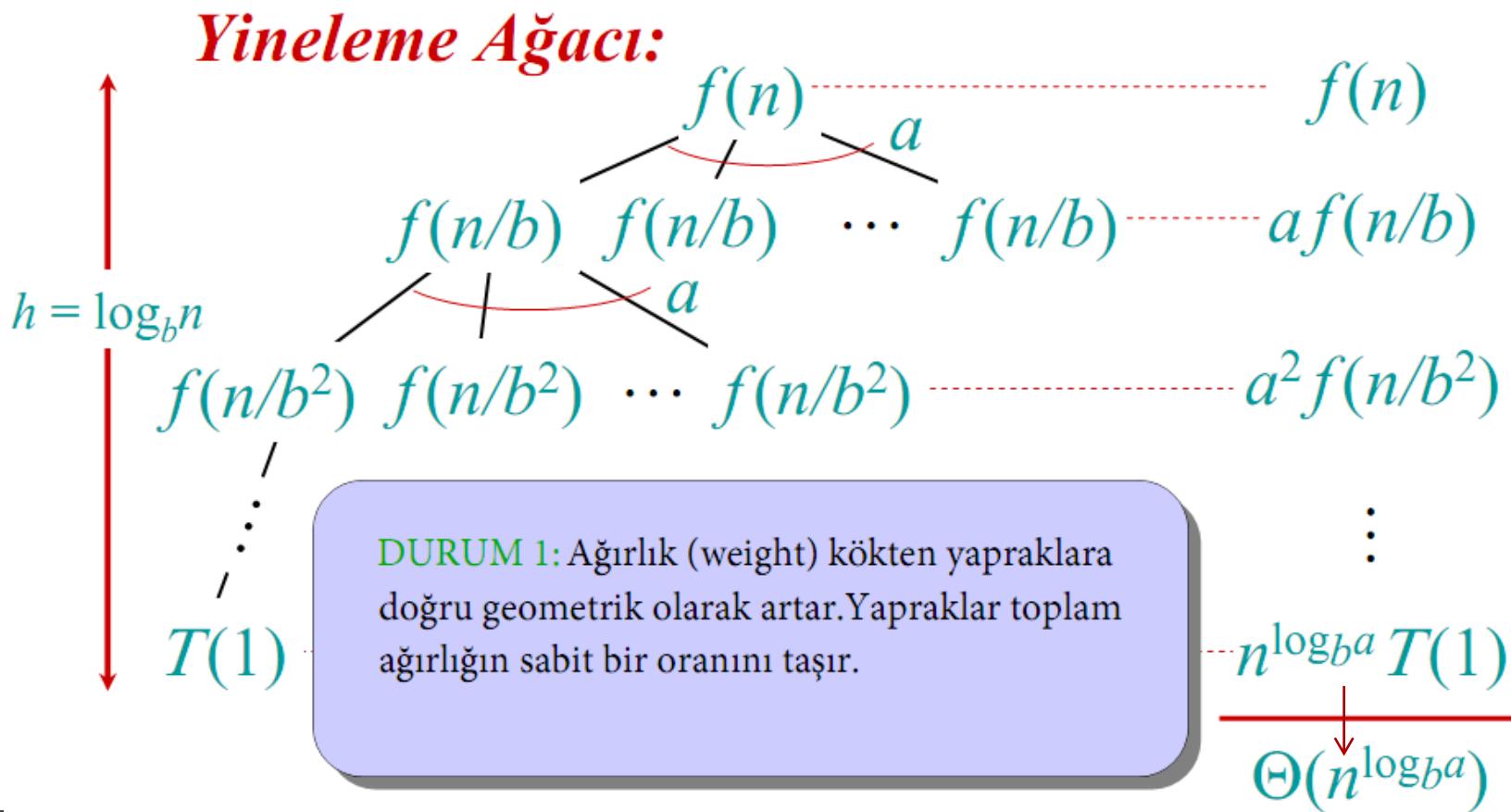


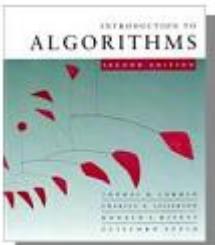
# Master teoremdeki düşünce



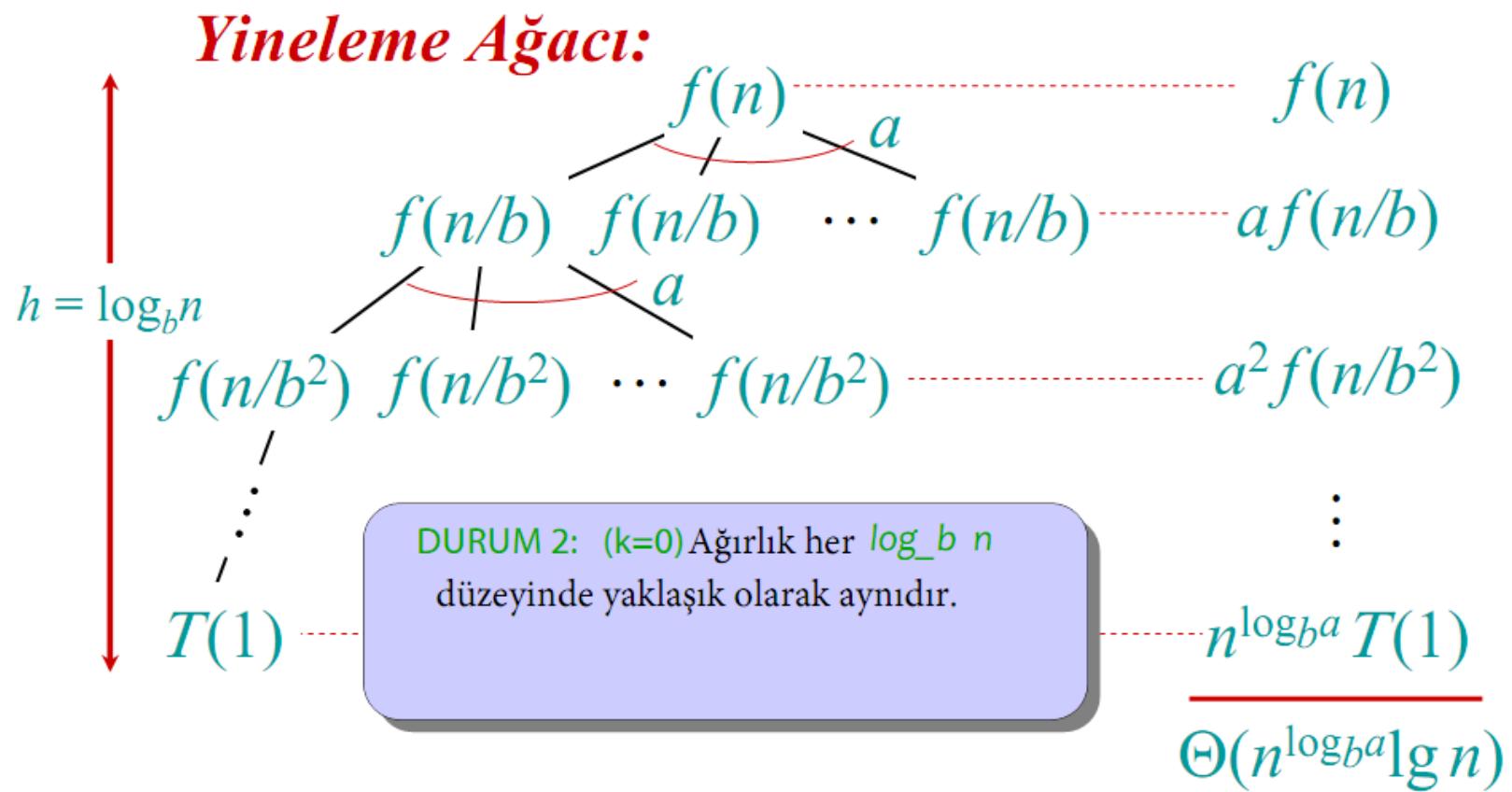


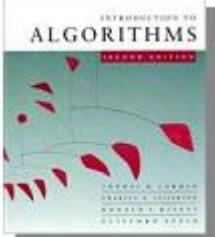
# Master teoremdeki düşünce



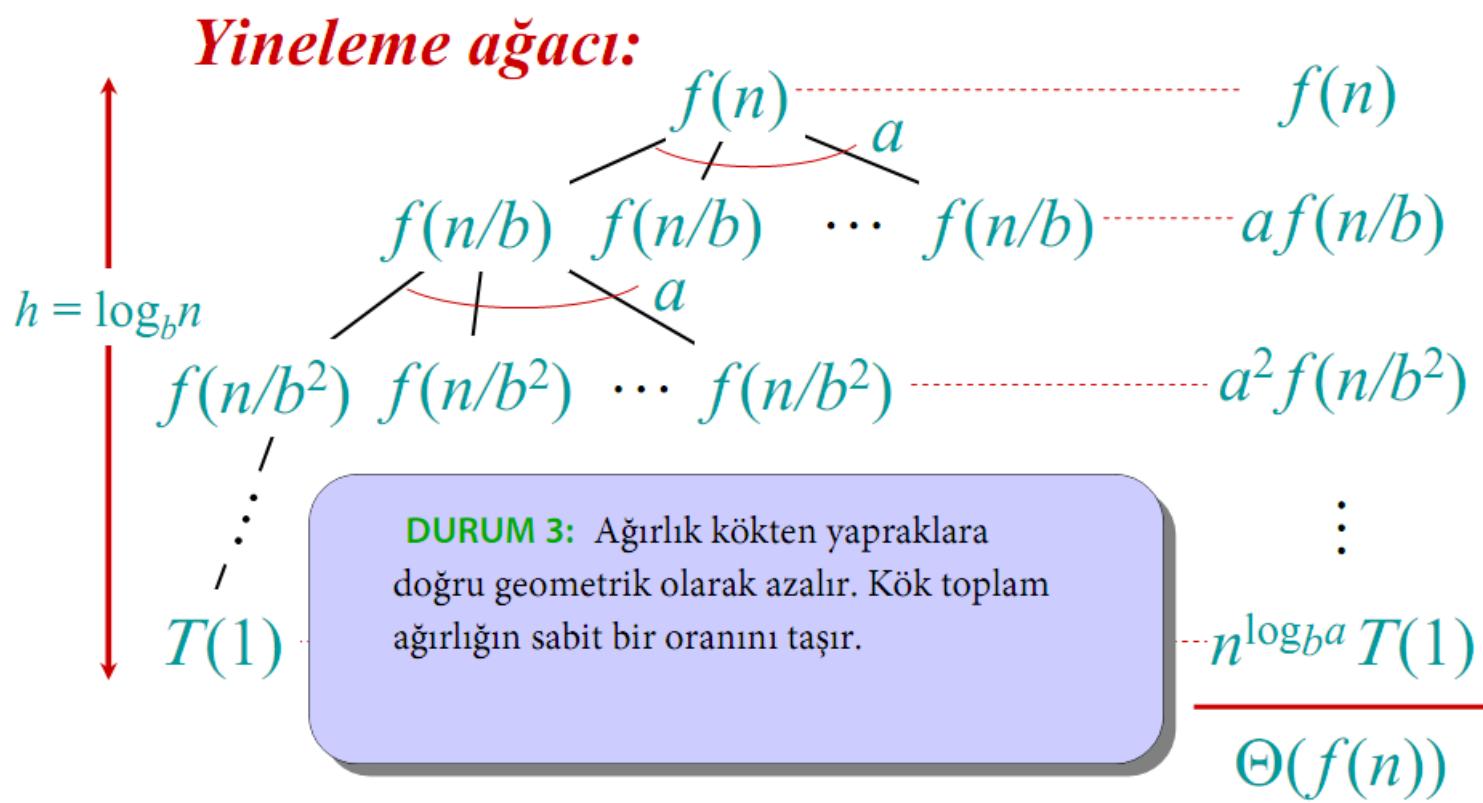


# Master teoremdeki düşünce

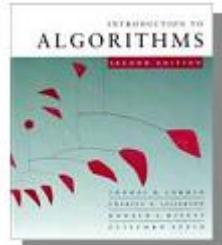




# Master teoremdeki düşünce

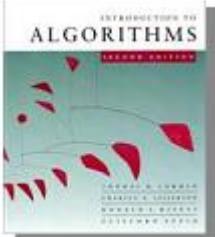


# Master teoremi ispat



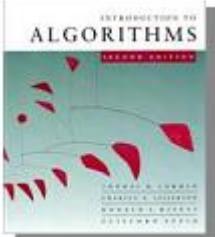
- **Durum 2:** Eğer  $f(n) = \theta(n^{\log_b a})$ , ise  $T(n) = \theta(n^{\log_b a} \log n)$
- **İspat:** Eğer  $f(n) = \theta(n^{\log_b a})$ , o zaman  $f(n) \leq cn^{\log_b a}$  olur
  
- $$T(n) = n^{\log_b a} + \sum_{j=0}^{\log_b n - 1} a^j f\left(\frac{n}{b^j}\right)$$
- $$\leq n^{\log_b a} + c \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{n}{b^j}\right)^{\log_b a}$$
- $$= n^{\log_b a} + cn^{\log_b a} \sum_{j=0}^{\log_b n - 1} a^j \left(\frac{1}{b^{\log_b a}}\right)^j \text{a}$$

j=0  $\xrightarrow{\quad}$  a
- $$= n^{\log_b a} + cn^{\log_b a} \sum_{j=0}^{\log_b n - 1} 1 = n^{\log_b a} + cn^{\log_b a} \log_b n$$
- $\leq cn^{\log_b a} \log n$
- Bu yüzden,  $f(n) = \theta(n^{\log_b a})$  ise  $T(n) = O(n^{\log_b a} \log n)$  dir.
- Durum 1 ve Durum 3 te benzerdir.( Önerilen ders kitabı inceleyiniz)



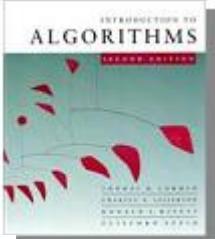
# Master teoremi

- Örnek:  $T(n) = 9T\left(\frac{n}{3}\right) + n, n \geq 3$  ve  $T(1) = 1$  ise çalışma zamanını bulunuz?
- Çözüm:  $a=9, b=3, f(n) = n$  ve  $n^{\log_b a} = n^{\log_3 9} = \theta(n^2)$
- Durum 1:  $f(n) = O(n^{\log_3 9 - \varepsilon})$ ,  $\varepsilon = 1$  için
- $T(n) = \theta(n^{\log_3 9}) = \theta(n^2)$



# Master teoremi

- **Örnek:**  $T(n) = T\left(\frac{2n}{3}\right) + 1, n \geq 3$  ve  $T(2) = 1$  ise çalışma zamanını bulunuz?
- **Çözüm:**  $a=1, b=3/2$  ,  $f(n) = 1$  ve  
 $n^{\log_b a} = n^{\log_{3/2} 1} = \theta(n^0)$
- **Durum 2:**  $f(n) = \Theta(n^{\log_b a} \lg^k n)$   $k \geq 0$   
 $T(n) = \theta(\log n)$



## Master teoremi

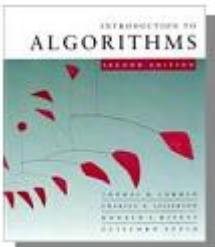
- Örnek:  $T(n) = 3T\left(\frac{n}{4}\right) + n \log n$ ,  $n \geq 4$  ve  $T(1) = 1$  ise çalışma zamanını bulunuz?

- Çözüm:  $a=3$ ,  $b=4$ ,  $f(n) = n \log n$  ve

$$n^{\log_b a} = n^{\log_4 3} = O(n^{0.793})$$

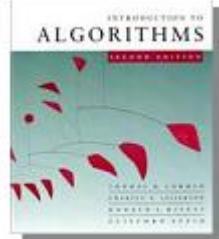
- Durum 3:  $f(n) = \Omega(n^{\log_4 3 + \varepsilon})$ , burada  $\varepsilon \approx 0.2$  için düzenlilik koşulu,  $a*f(n/b) \leq c*f(n)$ , büyük  $n$  değerleri ve  $c < 1$  olmak koşuluyla
- $3(n/4)\log(n/4) \leq (3/4) n \log n$ ,  $c=3/4 < 1$  için

$$T(n) = \theta(n \log n)$$



# Master teoremi ispat

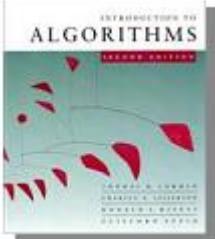
- ➊ Örnek:  $T(n) = 2T\left(\frac{n}{2}\right) + n \log n$  için master-metot durumları uygulanmaz?
  - Burada  $a=2$ ,  $b=2$ ,  $n^{\log_b a} = n^{\log_2 2} = O(n)$  ve  $f(n) = n \log n$  dir.
  - $f(n)$ , polinomsal olarak  $n^{\log_b a}$  göre hızlı büyündüğünden **Durum 3** uygulanır.
  - Büyüme oranı, asimptotik olarak çok büyük olmasına rağmen polinomsal olarak çok ta büyük değildir.
  - Büyüme oranı  $f(n)/n^{\log_b a} = (n \log n)/n = \log n$  dir. Bu oran herhangi bir pozitif  $\varepsilon$  sabiti için  $n^\varepsilon$  den asimptotik olarak azdır.
  - Sonuç olarak çözüm Durum2 ve Durum3 arasına düşer.



## Sorular

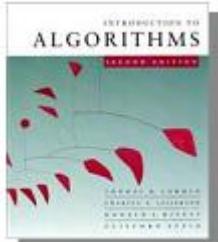
- 1- $T(n) = T\left(\frac{n}{2}\right) + \sqrt{n} + \sqrt{6046}$  çözümü.
- 2- $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + \theta(n)$  özyineleme ağacı kullanarak çözümü.
- 3- Asimptotik notasyonlardan hangilerinin geçişme özelliği (transitivity) vardır.

$$1-T(n) = T\left(\frac{n}{2}\right) + \sqrt{n} + \sqrt{6046} = O(\sqrt{n})$$



## Sorular

- 4- Aşağıdaki tekrarlı bağıntıların asimptotik davranışlarını inceleyiniz. ( $T(0)=1$ ,  $T(1)=1$ )
  - a)  $T(n)=2T(n-1)-T(n-2)+\Theta(n)$
  - b)  $T(n)=3T(n-1)-2T(n-2)+\Theta(n2^n)$
  - c)  $T(n)=4T(n/2)-4T(n/4)+\Theta(nlgn)$
  - d)  $T(n)=5T(n/2)-6T(n/4)+\Theta(n)$
- 5-  $T(n)=T(\lceil n/2 \rceil)+1$  tekrarlı bağıntısı için  $T(n)=O(lgn)$  olduğunu gösteriniz.
- 6-  $T(n)=T(n/2)+T(n/4)+T(n/8)+n$  öz yineleme ağaçları ile çözümüz

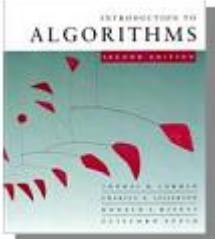


## Sorular

- 7-  $T(n)=2T(\lfloor n/2 \rfloor)+n$  tekrarlı bağıntısı için  $T(n)=\Omega(nlgn)$  ve  $T(n)=O(nlgn)$  olduğunu gösteriniz. Son olarak  $T(n)=\Theta(nlgn)$  asimptotik davranışını gösterip göstermediğini açıklayınız.
- 8- Aşağıdaki tekrarlı bağıntıyı çözünüz. ( $0 < p \leq q < 1$ )
  - $T(n)=T(pn)+T(qn)+\Theta(n)$
  - $T(1)=\Theta(1)$
- 9-  $i$  değişkeni 1 ile  $n-1$  arasında değer alan düzenli dağıtık bir gelişigüzel değişken olsun.
- $f(n)=(i+1)f(i)$  ve  $f(1)=1$   
tekrarlı bağıntısı için  $f(n)$  nin mümkün olan değerleri nelerdir ve ortalama değeri nedir?
- 10-  $T(n) = 2 T(\lfloor n/2 \rfloor + 17) + n$  tekrarlı bağıntısının çözümünün  $O(nlgn)$  olduğunu gösteriniz.

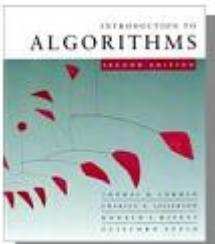
# **5.Hafta**

## **Karakteristik denklemler kullanarak yinelemeleri çözme**



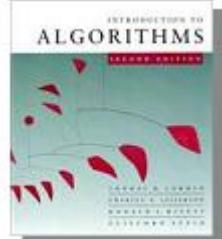
## Sorular

- $1-T(n) = T\left(\frac{n}{2}\right) + \sqrt{n} + \sqrt{6046}$  çözümü.
- $2-T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + \theta(n)$  özyineleme ağacı kullanarak çözümü.
- 3-Aşağıdaki tekrarlı bağıntıların asimptotik davranışlarını inceleyiniz. ( $T(0)=1$ ,  $T(1)=1$ )
  - a)  $T(n)=2T(n-1)-T(n-2)+\Theta(n)$
  - b)  $T(n)=3T(n-1)-2T(n-2)+\Theta(n2^n)$
  - c)  $T(n)=4T(n/2)-4T(n/4)+\Theta(nlgn)$
  - d)  $T(n)=5T(n/2)-6T(n/4)+\Theta(n)$



## Sorular

- 4-  $T(n)=T(\lceil n/2 \rceil)+1$  tekrarlı bağıntısı için  $T(n)=O(lgn)$  olduğunu gösteriniz.
- 5-  $T(n)=T(n/2)+T(n/4)+T(n/8)+n$  öz yineleme ağacı ile çözünüz
- 6-  $T(n)=2T(\lfloor n/2 \rfloor)+n$  tekrarlı bağıntısı için  $T(n)=\Omega(nlgn)$  ve  $T(n)=O(nlgn)$  olduğunu gösteriniz. Son olarak  $T(n)=\Theta(nlgn)$  asimptotik davranışını gösterip göstermediğini açıklayınız.



## Sorular

- 7- Aşağıdaki tekrarlı bağıntıyı çözünüz. ( $0 < p \leq q < 1$ )
  - $T(n) = T(pn) + T(qn) + \Theta(n)$
  - $T(1) = \Theta(1)$
- 8-  $i$  değişkeni 1 ile  $n-1$  arasında değer alan düzenli dağıtık bir gelişigüzel değişken olsun.
  - $f(n) = (i+1)f(i)$  ve  $f(1) = 1$   
tekrarlı bağıntısı için  $f(n)$  nin mümkün olan değerleri nelerdir ve ortalama değeri nedir?
- 9-  $T(n) = 2 T(\lfloor n/2 \rfloor + 17) + n$  tekrarlı bağıntısının çözümünün  $O(nlgn)$  olduğunu gösteriniz.

$$1 + x + x^2 + \dots + x^n = \frac{1 - x^{n+1}}{1 - x} ; x \neq 1 \text{ için}$$

$$1 + x + x^2 + \dots = \frac{1}{1 - x} ; |x| < 1 \text{ için}$$

## Bazı Matematiksel İfadeler

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

Karelerin ToplAMI:

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

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

- İki sınır arasındaki sayıların toplamı:  $\sum_{i=a}^b f(i) = \sum_{i=0}^b f(i) - \sum_{i=0}^{a-1} f(i)$

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

- Combinatorics

1. Number of permutations of an  $n$ -element set:  $P(n) = n!$
2. Number of  $k$ -combinations of an  $n$ -element set:  $C(n, k) = \frac{n!}{k!(n-k)!}$
3. Number of subsets of an  $n$ -element set:  $2^n$

# Bazı Matematiksel İfadeler

- Properties of Logarithms

$$1. \log_a 1 = 0$$

$$2. \log_a a = 1$$

$$3. \log_a x^y = y \log_a x$$

$$4. \log_a xy = \log_a x + \log_a y$$

$$5. \log_a \frac{x}{y} = \log_a x - \log_a y$$

$$6. a^{\log_b x} = x^{\log_b a}$$

$$7. \log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$$

# Bazı Önemli Matematiksel İfadeler

## ● Important Summation Formulas

$$1. \sum_{i=l}^u 1 = \underbrace{1 + 1 + \cdots + 1}_{u-l+1 \text{ times}} = u - l + 1 \quad (l, u \text{ are integer limits}, l \leq u); \quad \sum_{i=1}^n 1 = n$$

$$2. \sum_{i=1}^n i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$$

$$3. \sum_{i=1}^n i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$$

$$4. \sum_{i=1}^n i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1}n^{k+1}$$

$$5. \sum_{i=0}^n a^i = 1 + a + \cdots + a^n = \frac{a^{n+1} - 1}{a - 1} \quad (a \neq 1); \quad \sum_{i=0}^n 2^i = 2^{n+1} - 1$$

$$6. \sum_{i=1}^n i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2 \quad \sum_{i=0}^{n-1} ix^i = x + 2x^2 + 3x^3 + \cdots + nx^n = \frac{(n-1)x^{(n+1)} - nx^n + x}{(x-1)^2}.$$

$$7. \sum_{i=1}^n \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma, \text{ where } \gamma \approx 0.5772 \dots \text{ (Euler's constant)}$$

$$8. \sum_{i=1}^n \lg i \approx n \lg n$$

# Bazı Önemli Matematiksel İfadeler

$$1. \sum_{k=1}^n k = 1+2+3+\dots+n = \frac{n(n+1)}{2}$$

$$2. \sum_{k=1}^n 2k = 2+4+6+\dots+2n = n(n+1)$$

$$3. \sum_{k=1}^n (2k-1) = 1+3+5+\dots+(2n-1) = n^2$$

$$4. \sum_{k=1}^n k^2 = 1^2 + 2^2 + 3^2 + \dots + n^2 = \frac{n(n+1)(2n+1)}{6}$$

$$5. \sum_{k=1}^n k^3 = 1^3 + 2^3 + 3^3 + \dots + n^3 = \left[ \frac{n(n+1)}{2} \right]^2$$

$$6. \sum_{k=1}^n r^{k-1} = 1+r+r^2+r^3+\dots+r^{n-1} = \frac{1-r^n}{1-r}, \quad (r \neq 1)$$

$$7. \sum_{k=1}^n \frac{1}{k \cdot (k+1)} = \frac{1}{1 \cdot 2} + \frac{1}{2 \cdot 3} + \dots + \frac{1}{n \cdot (n+1)} = \frac{n}{n+1}$$

$$8. \sum_{k=1}^n k \cdot k! = (n+1)! - 1$$

# Bazı Matematiksel İfadeler

## ● Sum Manipulation Rules

$$1. \sum_{i=l}^u ca_i = c \sum_{i=l}^u a_i$$

$$2. \sum_{i=l}^u (a_i \pm b_i) = \sum_{i=l}^u a_i \pm \sum_{i=l}^u b_i$$

$$3. \sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=m+1}^u a_i, \text{ where } l \leq m < u$$

$$4. \sum_{i=l}^u (a_i - a_{i-1}) = a_u - a_{l-1}$$

## Approximation of a Sum by a Definite Integral

$$\int_{l-1}^u f(x)dx \leq \sum_{i=l}^u f(i) \leq \int_l^{u+1} f(x)dx \quad \text{for a nondecreasing } f(x)$$

$$\int_l^{u+1} f(x)dx \leq \sum_{i=l}^u f(i) \leq \int_{l-1}^u f(x)dx \quad \text{for a nonincreasing } f(x)$$

## Floor and Ceiling Formulas

The *floor* of a real number  $x$ , denoted  $\lfloor x \rfloor$ , is defined as the greatest integer not larger than  $x$  (e.g.,  $\lfloor 3.8 \rfloor = 3$ ,  $\lfloor -3.8 \rfloor = -4$ ,  $\lfloor 3 \rfloor = 3$ ). The *ceiling* of a real number  $x$ , denoted  $\lceil x \rceil$ , is defined as the smallest integer not smaller than  $x$  (e.g.,  $\lceil 3.8 \rceil = 4$ ,  $\lceil -3.8 \rceil = -3$ ,  $\lceil 3 \rceil = 3$ ).

1.  $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
2.  $\lfloor x + n \rfloor = \lfloor x \rfloor + n$  and  $\lceil x + n \rceil = \lceil x \rceil + n$  for real  $x$  and integer  $n$
3.  $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$
4.  $\lceil \lg(n+1) \rceil = \lfloor \lg n \rfloor + 1$

## Miscellaneous

1.  $n! \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n$  as  $n \rightarrow \infty$  (Stirling's formula)
2. Modular arithmetic ( $n, m$  are integers,  $p$  is a positive integer)

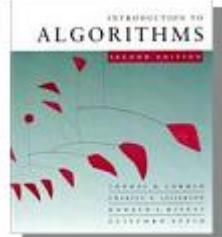
$$(n+m) \bmod p = (n \bmod p + m \bmod p) \bmod p$$

$$(nm) \bmod p = ((n \bmod p)(m \bmod p)) \bmod p$$

# **5.Hafta**

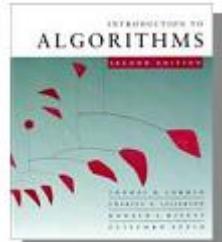
## **Karakteristik denklemler kullanarak yinelemeleri çözme**

# Karakteristik denklemler kullanarak yinelemeleri çözme



- **Doğrusal Yineleme (Rekürans) Bağıntısı**
- Bir yinelemeli bağıntıda  $t_n$ , dizinin önceki terimlerinin katlarının toplamına eşitse doğrusal (lineer) dır. ( $t_n \rightarrow T(n)$ )
  - $t_n = t_{n-1} + t_{n-2}$  doğrusal
  - $t_n = t_{n-1} + t^2_{n-2}$  doğrusal değildir,  $t^2_{n-2}$  önceki terimin katı değildir.
- **Homojen Yineleme (Rekürans) Bağıntısı:**
- $t_n$  sadece önceki terimlerin katlarına bağlı ise homojen (türdes) olarak adlandırılır.
  - $t_n = t_{n-1} + t_{n-2}$  homojen
  - $t_n = 2t_{n-1} + 1$  homojen değildir. "+1" terimi  $t_j$  katı değildir.

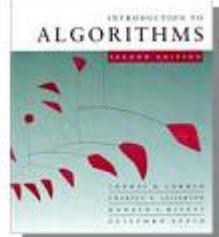
## Karakteristik denklemler kullanarak özyinelemeleri çözme



- Yinelemeli bağıntıdaki terimlerin katsayıları sabit ise; sabit katsayılı homojen doğrusal yineleme formu aşağıdaki gibidir.

$$c_0 t_n + c_1 t_{n-1} + \dots + c_k t_{n-k} = 0$$

- Burada,
  - $t_i$ : özyinelemeli bağıntının değerlerini,
  - $c_i$ : sabit katsayılı terimlerini ifade eder.
    - $c_i$ , reel sayılardır ve  $c_i \neq 0$ .
  - $k$ : ise özyinelemeli bağıntının derecesidir.



## Karakteristik denklemler kullanarak özyinelemeleri çözme

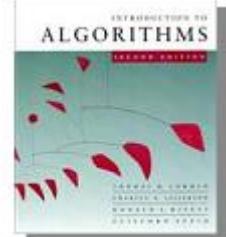
- Doğrusal özyinelemelerde  $t_{i+j}$ ,  $t_i^2$  şeklinde terimler bulunmaz.
- Öz yineleme homojendir, çünkü  $t_i$  nin doğrusal kombinasyonundan dolayı  $0$  (sıfır)' a eşittir.
- Bu öz yinelemeler  $k$  başlangıç koşullarını içerir.

$$t_n=c_0 \quad t_1=c_1 \quad \dots \quad t_k=c_k$$

- Fibonacci dizisi için özyineleme

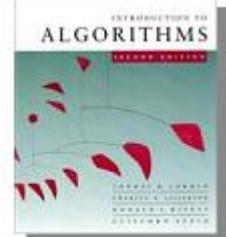
- $f_n=f_{n-1}+f_{n-2}, \rightarrow f_n-f_{n-1}-f_{n-2}=0,$   
burada  $k=2$ ,  $c_0=1$  ve  $c_1=c_2=-1$  dir.

## Karakteristik denklemler kullanarak özyinelemeleri çözme



- Sabit katsayılı homojen doğrusal yineleme bağıntılarını çözmek için basit bir yöntem vardır. Bu yöntem;
  - $k$  bir sabit olmak üzere,  $t_k = x^k$ ;
  - $t_n = c_1 t_{n-1} + c_2 t_{n-2} + \dots + c_k t_{n-k}$ 'nın bir çözümü kabul edilir ve bağıntıda yerine konulursa
  - $x^n = c_1 x^{n-1} + c_2 x^{n-2} + \dots + c_k x^{n-k}$  elde edilir. Burada,  $x$  bilinmeyen bir sabit ve  $x \neq 0$  dır.
- Bu ifadenin her iki yanını  $x^{n-k}$  ile bölersek:
  - $x^k - c_1 x^{k-1} - c_2 x^{k-2} - \dots - c_k = 0$  bulunur ve derecesi  $k$  olan ve genelde  $k$  adet kökü olan bu polinoma yineleme bağıntısının **karakteristik denklemi** (characteristic equation) adı verilir. Bu denklemin kökü birden fazla veya karmaşık olabilir.

# Karakteristik denklemler kullanarak özyinelemeleri çözme



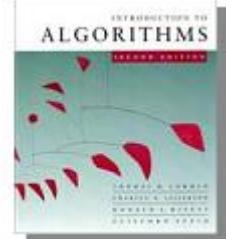
## • İşlem Adımları

### ○ Adım 1

- $x^2 - c_1x - c_2 = 0$  karakteristik denklemi için
- İkinci dereceden bir denklem olduğundan karakteristik kökleri  $r_1$  ve  $r_2$  olup
- $x_{1,2} = \frac{c_1 \pm \sqrt{(c_1)^2 + 4c_2}}{2}$

### ○ Adım 2

- **Durum 1:** Köklerin hiç biri aynı değilse
- $t_n = c_1 x_1^n + c_2 x_2^n$
- **Durum 2:** Köklerde aynı olan değer var ise ( $x_1 = x_2$ )
- $t_n = c_1 x_0^n + c_2 x_1^n + c_3 n x_1^n$



# Karakteristik denklemler kullanarak özyinelemeleri çözme

## • Adım 3

- Bir önceki adımda elde edilen denklemlere ilk koşulları uygulayınız.
- Durum I: Kökler eşit değil
  - $t_0 = c_1x_1^0 + c_2x_2^0 = c_1 + c_2$
  - $t_1 = c_1x_1^1 + c_2x_2^1 = c_1x_1 + c_2x_2$
  - Durum 2: Kökler eşit ( $x_2 = x_1 = x_0$ )
    - $t_0 = c_1x_0^0 + c_2 \cdot 0 \cdot x_0^0 = c_1$
    - $t_1 = c_1x_0^1 + c_2 \cdot 1 \cdot x_0^1 = (c_1 + c_2)x_0$

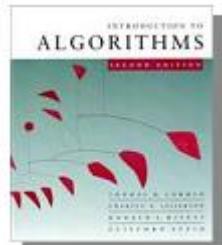
## ○ Adım 4

- $c_1, c_2$ 'yi bulunuz

## ○ Adım 5

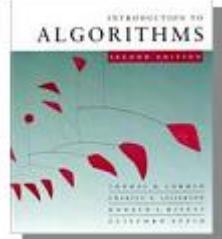
$t_n$  için genel çözümü yaz

## Karakteristik denklemler kullanarak özyinelemeleri çözme



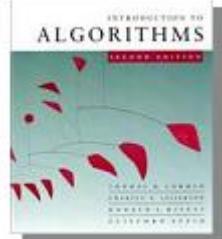
- ➊ Örnek: İlk koşullar  $t_0 = 2$  ve  $t_1 = 7$  olarak verildiğine göre
  - ➋  $t_n = t_{n-1} + 2t_{n-2}$  yinelemeli bağıntıyı çözünüz
  - ⌾ Karakteristik denklem:  $x^2 - x - 2 = 0$
  - ⌽ Kökler  $x_1 = 2$  ve  $x_2 = -1$ , kökler eşit değil. Durum 1'i kullanılacak.
  - ⌽  $t_n = c_1 2^n + c_2 (-1)^n$
  - ⌽  $t_0 = 2 = c_1 + c_2$ ,  $t_1 = 7 = c_1 \cdot 2 + c_2 \cdot (-1)$
  - ⌽  $c_1 = 3$  ve  $c_2 = -1$  olarak bulunur.
  - ⌽ Bu değerleri yerine yazarsak
  - ⌽  $t_n = 3 \cdot 2^n + (-1) \cdot (-1)^n = 3 \cdot 2^n - (-1)^n$  olarak bulunur.
  - ⌽  $t_n \in \theta(2^n)$

## Karakteristik denklemler kullanarak özyinelemeleri çözme



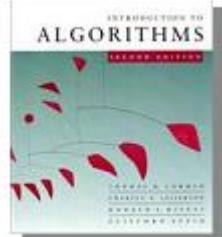
- ➊ Örnek: İlk koşullar  $t_0 = 0$  ve  $t_1 = 1$  ve  $n \geq 2$  olarak verildiğine göre
  - ➋  $t_n - 3t_{n-1} - 4t_{n-2} = 0$  için yinelemeli bağıntıyı çözünüz
  - ⌾ Karakteristik denklem:  $x^2 - 3x - 4 = 0$
  - ⌽ Kökler  $x_1 = -1$  ve  $x_2 = 4$ , kökler eşit değil. Durum 1'i kullanılacak.
  - ⌽  $t_n = c_1(-1)^n + c_24^n$
  - ⌽  $t_0 = 0 = c_1 + c_2, t_1 = 1 = c_1 \cdot (-1) + 4c_2$
  - ⌽  $c_1 = -1/5$  ve  $c_2 = 1/5$  olarak bulunur.
  - ⌽ Bu değerleri yerine yazarsak
  - ⌽  $t_n = 1/5[4^n - (-1)^n]$  olarak bulunur.
  - ⌽  $t_n \in \theta(4^n)$

## Karakteristik denklemler kullanarak özyinelemeleri çözme



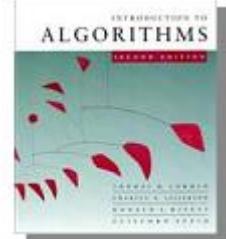
- ➊ Örnek: Fibonacci özyinelemeli bağıntı
- ➋  $t_0 = 0$  ve  $t_1 = 1$  ve  $n \geq 2$  olarak verildiğine göre
- ⌂  $t_n - t_{n-1} - t_{n-2} = 0$  için yinelemeli bağıntıyı çözünüz
- ⌃ Karakteristik denklem:  $x^2 - x - 1 = 0$
- ⌄ Kökler  $x_1 = \frac{1+\sqrt{5}}{2}$  ve  $x_2 = \frac{1-\sqrt{5}}{2}$ , kökler eşit değil. ( $\frac{1+\sqrt{5}}{2}$ , altın oran)
- ⌅ Durum 1'i kullanılacak.  $t_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$
- ⌆  $t_n(0) = 0 = c_1 + c_2$ ,  $t_1 = 1 = c_1 \cdot \frac{1+\sqrt{5}}{2} + c_2 \cdot \frac{1-\sqrt{5}}{2}$
- ⌇  $c_1 = 1/\sqrt{5}$  ve  $c_2 = -1/\sqrt{5}$  olarak bulunur.
- ⌈ Bu değerleri yerine yazarsak
- ⌋  $t_n = 1/\sqrt{5} \left[ \left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right]$  olarak bulunur ve sonuç olarak
- ⌌  $t_n \in \theta\left(\left((1 + \sqrt{5})/2\right)^n\right)$

## Karakteristik denklemler kullanarak özyinelemeleri çözme



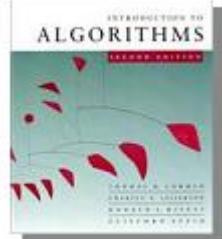
- ➊ Örnek:  $t_0 = 0$ ,  $t_1 = 1$  ve  $t_2 = 2$  ve  $n \geq 3$  olarak verildiğine göre
  - $t_n = 5t_{n-1} - 8t_{n-2} + 4t_{n-3}$  için yinelemeli bağıntıyı çözünüz
  - Karakteristik denklem:  $x^3 - 5x^2 + 8x - 4 = 0$ , veya  $(x-1)(x-2)^2$
  - Kökler  $x_1 = 1$ , ve  $x_2 = x_3 = 2$ , (iki kök eşit). Eşit kökler bulunduğuundan Durum 2 kullanılacak
  - $t_n = c_1(1)^n + c_2(2)^n + c_3n(2)^n$
  - Başlangıç şartlarına göre;
  - $c_1 + c_2 = 0$ ; ( $n = 0$ ),
  - $c_1 + 2c_2 + 2c_3 = 1$ ; ( $n = 1$ ),
  - $c_1 + 4c_2 + 8c_3 = 2$ ; ( $n = 2$ ),
  - $c_1 = -2$ ,  $c_2 = 2$  ve  $c_3 = -1/2$  olarak bulunur.
  - Bu değerleri yerine yazarsak
  - $t_n = 2^{n+1} - n2^{n-1} - 2$  olarak bulunur.

# Karakteristik denklemler kullanarak yinelemeleri çözme



- **Homojen Olmayan Yineleme Bağıntıları**
- $t_n$  sadece önceki terimlerin katlarına bağlı değil ise homojen olmayan bağıntı olarak adlandırılır.
  - $t_n = t_{n-1} + t_{n-2}$  homojen
  - $t_n = 2t_{n-1} + 1$  homojen değildir. "+1" terimi  $t_j$  katı değildir.
- Yinelemeli bağıntıların genel formu  $c_0t_n + c_1t_{n-1} + \dots + c_k t_{n-k} = f(n)$  şeklinde ifade edilir.  $f(n) = 0$  eşit ise homojen, sıfırdan farklı ise homojen olmayan yinelemeli bağıntıdır.
- **$f(n) = b^n p(n)$**
- şeklinde ifade edilirse  $b$  sıfırdan farklı bir sabiti  $p(n)$  ise  $d$ . dereceden  $n$  nin bir polinomudur.

## Karakteristik denklemler kullanarak yinelemeleri çözme



- Örnek: Aşağıda verilen reküransı çözünüz
- $t_n - 2t_{n-1} = 3^n$  burada b=3,  $p(n)^d=1$  ve polinom derecesi d=0' dır.

- İlk olarak her iki tarafı 3 ile çarpalım:

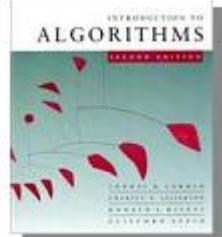
$$3t_n - 6t_{n-1} = 3^{n+1}$$

- Eğer n, n+1 ile yer değiştirirsek:

$$t_{n+1} - 2t_n = 3^{n+1} \quad \text{denklemini elde ederiz.}$$

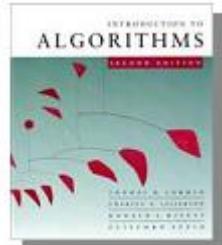
- Her iki denklemi bir birinden çıkarırsak yeni denklem
- $t_{n+1} - 5t_n + 6t_{n-1} = 0$  olur.

## Karakteristik denklemler kullanarak yinelemeleri çözme



- Homojen durumda olduğu gibi çözüm yaparsak karakteristik denklem
- $x^2 - 5x + 6 = 0, \quad \rightarrow \quad (x-2)(x-3)=0$
- Dikkat edilecek olursa  $(x-2)$  değeri orijinal rekürans ta sol tarafı,  $x-3$  ise sağ taraftaki polinomu ifade etmekte. Buna göre karakteristik denklemin basit genel formunu aşağıdaki şekilde ifade edebiliriz:
- $(c_0 x^k + c_1 x^{k-1} + c_2 x^{k-2} + \dots + c_k)(x-b)^{d+1} = 0,$
- burada  $d$ ,  $p(n)$  polinomunun derecesidir. Bu denklem elde edildikten sonra homojen durumunda olduğu gibi çözüm yapılır.

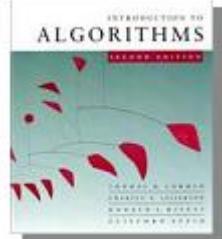
## Karakteristik denklemler kullanarak yinelemeleri çözme



➊ Örnek: Aşağıda verilen Hanoi Kulesi problemine ait reküransı çözünüz

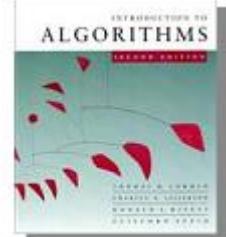
- $t_n = 2t_{n-1} + 1$ ;  $n \geq 1$  ve  $t_0 = 0$ ;
- Burada  $b=1$   $p(n)=1$  ve polinom derecesi 0 dır.
- Karakteristik denklem:  $(x-2)(x-1)=0$  olur.
- $t_n = c_1 1^n + c_2 2^n$ ,  $t_0 = c_1 + c_2 = 0$ ,  $t_1 = 2t_0 + 1 = 1$  ise
- $t_1 = c_1 1 + 2c_2 = 1$  olur.  $c_1 = -1$ ,  $c_2 = 1$  bulunur
- $t_n = 2^n - 1$  elde ederiz. Sonuç olarak ;
- $t_n \in \theta(2^n)$  olur.

## Karakteristik denklemler kullanarak yinelemeleri çözme



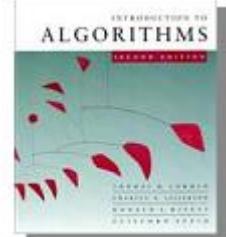
- $c_0t_n+c_1t_{n-1}+\dots+c_kt_{n-k}=b^n p(n)$  homojen olmayan denklemler için verilen basit genel formu daha da genelleştirirsek
- $c_0t_n+c_1t_{n-1}+\dots+c_kt_{n-k}=b_1^n p_1(n)+b_2^n p_2(n)+\dots$  formunu elde ederiz. Buna göre karakteristik denklem:  
  
$$(c_0x^k + c_1x^{k-1} + c_2x^{k-2} + \dots + c_k)(x-b_1)^{d_1+1}(x-b_2)^{d_2+1}\dots = 0,$$

## Karakteristik denklemler kullanarak yinelemeleri çözme



- ➊ Örnek:  $n \geq 1$  ve  $t_0 = 0$  başlangıç şartları için  $t_n = 2t_{n-1} + n + 2^n$  problemine ait reküransı çözümünüz
- ➋  $t_n - 2t_{n-1} = n + 2^n$ , burada  $b_1 = 1$ ,  $p_1(n) = n$ ,  $b_2 = 2$ ,  $p_2(n) = 1$ ,  $d_1 = 1$ ,  $d_2 = 0$ ,  $n$  polinom derecesidir.
- ➌ Karakteristik denklem:  $(x-2)(x-1)^2(x-2) = 0$  olur. Kökler, 1, 1, 2, 2 dir.
- ➍ Buna göre genel çözüm  $t_n = c_1 1^n + c_2 n 1^n + c_3 2^n + c_4 n 2^n$
- ➎  $t_1 = 0 + 1 + 2^1 = 3$ ,  $t_2 = 12$ ,  $t_3 = 35$
- ➏  $n = 0$  için  $c_1 + c_3 = 0$ ,
- ➐  $n = 1$  için  $c_1 + c_2 + 2c_3 + 2c_4 = 3$ ,
- ➑  $n = 2$  için  $c_1 + 2c_2 + 4c_3 + 8c_4 = 12$ ,
- ➒  $n = 3$  için  $c_1 + 3c_2 + 8c_3 + 24c_4 = 35$ ,
- ➓  $t_n = -2 - n + 2^{n+1} + n2^n$  elde ederiz. Sonuç olarak ;
- ➔  $t_n \in \theta(n2^n)$  olur.

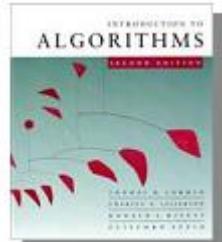
# Karakteristik denklemler kullanarak yinelemeleri çözme



- Çözüm yolu: Gaus yok etme yöntemi , bilinmeyenlerin ileriye doğru elenmesi. İlk adım ilk bilinmeyeni ( $c_1$ ), 2. denklemden n. Denkleme kadar elemektir.
- 2.denklem
- $a_{21} - (a_{21}/a_{11}) * a_{11} + a_{22} - (a_{21}/a_{11}) * a_{12} + \dots + a_{2n} - (a_{21}/a_{11}) * a_{1n} = c_2 - (a_{21}/a_{11}) * c_1$
- 3. denklem
- $a_{31} - (a_{31}/a_{11}) * a_{11} + a_{32} - (a_{31}/a_{11}) * a_{12} + \dots + a_{3n} - (a_{31}/a_{11}) * a_{1n} = c_3 - (a_{31}/a_{11}) * c_1$
- n.denklem
- $a_{n1} - (a_{n1}/a_{11}) * a_{11} + a_{n2} - (a_{n1}/a_{11}) * a_{12} + \dots + a_{nn} - (a_{n1}/a_{11}) * a_{1n} = c_n - (a_{n1}/a_{11}) * c_1$
- Buna göre ilk durumda matrisimiz

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 1 & 2 & 2 \\ 1 & 2 & 4 & 8 \\ 1 & 3 & 8 & 24 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 12 \\ 35 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 2 & 3 & 8 \\ 0 & 3 & 7 & 24 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 12 \\ 35 \end{bmatrix}$$

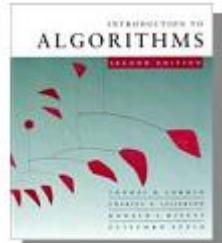
# Karakteristik denklemler kullanarak yinelemeleri çözme



- İkinci adım ikinci bilinmeyeni ( $c_2$ ), 3. denklemden n. denkleme katar elemektir.
- 3.denklem
- $a_{32} - (a_{32}/a_{22}) * a_{22} + a_{33} - (a_{32}/a_{22}) * a_{23} + \dots + a_{3n} - (a_{32}/a_{22}) * a_{2n} = c_3 - (a_{32}/a_{22}) * c_2$
- n.denklem
- $a_{n2} - (a_{n2}/a_{22}) * a_{22} + a_{n3} - (a_{n2}/a_{22}) * a_{23} + \dots + a_{nn} - (a_{n2}/a_{22}) * a_{2n} = c_n - (a_{n2}/a_{22}) * c_2$

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 2 & 3 & 8 \\ 0 & 3 & 7 & 24 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 12 \\ 35 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 4 & 18 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 6 \\ 26 \end{bmatrix}$$

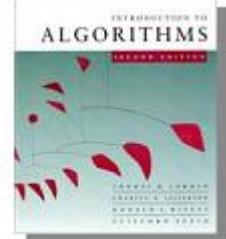
## Karakteristik denklemler kullanarak yinelemeleri çözme



- Diğer adımlarda benzer şekilde yapılır.

$$\begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 4 & 18 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 6 \\ 26 \end{bmatrix} \rightarrow \begin{bmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 4 \\ 0 & 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 3 \\ 6 \\ 2 \end{bmatrix}$$

- $c_4 = 1, c_3 = 2, c_2 = -1, c_1 = -2$  olur.
- $T(n) = -2 - n + 2^{n+2} + n2^n$
- $T(n) \in \theta(n2^n)$

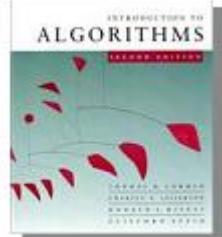


## Karakteristik denklemler kullanarak yinelemeleri çözme

### ② Değişkenlerin Değişimi:

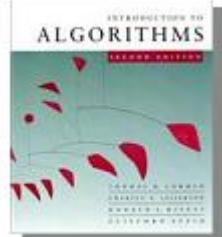
- $T(n)$  şeklinde verilen bir yinelemeyi değişkenlerin değişimi ile  $t_k$  şeklinde yeni bir yineleme yazılabilir.
- **Örnek:** 2 nin kuvveti şeklinde verilen n için aşağıda verilen yinelemeyi çözünüz.
- $T(n)=4T(n/2)+n$ ,  $n>1$
- $n$ , değerini  $2^k$  (burada  $k=\log n$  dir) ile yer değiştirirsek  $T(2^k)=4T(2^{k-1})+2^k$ , elde ederiz.
- Eğer  $t_k = T(2^k) = T(n)$  ise bunu ,  $t_k = 4t_{k-1} + 2^k$  şeklinde yazabiliriz. Yeni yinelemeyi çözersek  $(x-4)(x-2)=0$  karakteristik denklemini elde ederiz.
- $t_k = c_1 4^k + c_2 2^k$
- $k$  yerine  $n$  değerini yazarsak,
- $T(n) = c_1 n^2 + c_2 n$  buluruz.  $T(n) \in O(n^2)$

## Karakteristik denklemler kullanarak yinelemeleri çözme



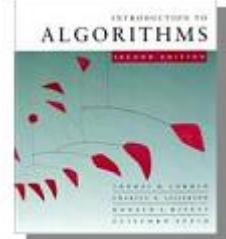
- ➊ Örnek:  $2$  nin kuvveti şeklinde verilen  $n$  için aşağıda verilen yinelemeyi çözünüz.  $T(n) = 2T(n/2) + n\log n$ ,  $n > 1$
- ➋  $n$ , değerini  $2^k$  (burada  $k = \log n$  dir) ile yer değiştirirsek  $T(2^k) = 2T(2^{k-1}) + k2^k$ , elde ederiz.
- ➌ Eğer  $t_k = T(2^k) = T(n)$  ise bunu,  $t_k = 2t_{k-1} + k2^k$  şeklinde yazabiliriz. Yeni yinelemeyi çözersek: ( $t_k - 2t_{k-1} = k2^k$ , burada  $b=2$ ,  $p(k)=k$  ve  $d=1$  olduğundan)
- ➍  $(x-2)^3 = 0$  karakteristik denklemini elde ederiz ve
- ➎  $t_k = c_1 2^k + c_2 k 2^k + c_3 k^2 2^k$
- ➏  $k$  yerine  $n$  değerini yazarsak,
- ➐  $T(n) = c_1 n + c_2 n \log n + c_3 n \log^2 n$  buluruz.
- ➑  $T(n) \in O(n \log^2 n)$

## Karakteristik denklemler kullanarak yinelemeleri çözme



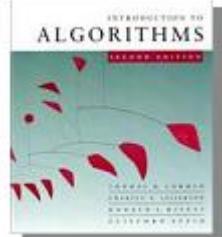
- ➊ Örnek:  $2$  nin kuvveti şeklinde verilen  $n$  için aşağıda verilen yinelemeyi çözünüz.  $T(n)=3T(n/2)+cn$  ( $c$  bir sabittir ve  $n = 2^k > 1$ )
  - $n$ , değerini  $2^k$  (burada  $k=\log n$ ' dir) ile yer değiştirirsek  $T(2^k)=3T(2^{k-1})+c2^k$ , elde ederiz.
  - Eğer  $t_k = T(2^k) = T(n)$  ise bunu,  $t_k = 3t_{k-1} + c2^k$  şeklinde yazabiliriz. Yeni yinelemeyi çözersek:  $(t_k - 3t_{k-1}) = c2^k$ , burada  $b=2$ ,  $p(k)=c$  ve  $d=0$  olduğundan  $(x-3)(x-2)=0$  karakteristik denklemini elde ederiz ve
  - $t_k = c_13^k + c_22^k$
  - $k$  yerine  $n$  değerini yazarsak,
  - $T(n) = c_13^{\log n} + c_2n$ , ( $a^{\log b} = b^{\log a}$  olduğundan)
  - $T(n) = c_1n^{\log 3} + c_2n$  buluruz.
  - $T(n) \in O(n^{\log 3})$

## Karakteristik denklemler kullanarak yinelemeleri çözme

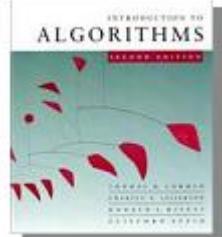


- Aralık dönüşümleri (Range Transformations): Yinelemelerin çözümünde değişkenlerin değişimi yerine bazen aralık dönüşümü kullanmak daha faydalı olabilir.
- Örnek:  $T(n) = nT(n/2)^2$ ,  $n > 1$ ,  $T(1) = 6$
- $n$ , değerini  $2^k$  (burada  $k = \log n$  dir) ile yer değiştirirsek  $T(2^k) = 2^k T(2^{k-1})^2$ , elde ederiz.
- $t_k = T(2^k) = T(n)$  ise,  $t_k = 2^k t_{k-1}^2$ ,  $k > 0$  için  $t_0 = 6$
- İlk bakışta gördüğümüz tekniklerin hiç biri bu yineleme için uygulanamaz çünkü hem doğrusal değil, hem de katsayılarından biri sabit değildir.
- Aralık dönüşümü yapmak için  $V_k = \log t_k$  koyarak yeni bir yineleme oluşturulur.

## Karakteristik denklemler kullanarak yinelemeleri çözme



- $V_k = k + 2V_{k-1}$ ,  $k > 0$  için başlangıç şartları  $V_0 = \log 6 = \log 2 * 3 = 1 + \log 3$
- $V_k = k + 2V_{k-1}$  için karakteristik denklem  $(x - 2)(x - 1)^2 = 0$  ve
- $V_k = c_1 2^k + c_2 1^k + c_3 k 1^k$
- $V_k = k + 2V_{k-1}$  denkleminden  $V_0 = 1 + \log 3$ ,  $V_1 = 3 + 2\log 3$ ,  $V_2 = 8 + 4\log 3$  bulunur ve  $V_k = c_1 2^k + c_2 1^k + c_3 k 1^k$
- $V_0 = 1 + \log 3 = c_1 + c_2$
- $V_1 = 3 + 2\log 3 = 2c_1 + c_2 + c_3$
- $V_2 = 8 + 4\log 3 = 4c_1 + c_2 + 4c_3$
- $c_1 = 3 + \log 3$ ,  $c_2 = -2$ ,  $c_3 = -1$
- $V_k = (3 + \log 3)2^k - k - 2$

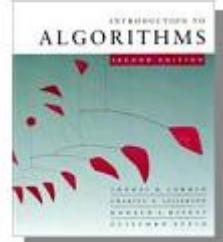


## Karakteristik denklemler kullanarak yinelemeleri çözme

Sonunda  $t_k = 2^{V^k}$

- $2^{V^k} = 2^{(3+\log 3)*2^k - k - 2} \rightarrow t_k = 2^{(3+\log 3)*2^k - k - 2}$
- $k$  yerine  $n$  değerini yazarsak,
- $T(n) = 2^{(3+\log 3)*n - \log n - 2} \rightarrow T(n) = 2^{3n + n\log 3} / (2^{\log n} * 2^2)$
- $T(n) = 2^{3n-2} * \frac{3^{n\log 2}}{n} = 2^{3n-2} * \frac{3^n}{n}$
- $T(n) = (2^{3n-2} 3^n)/n$
- $T(n) \in O(2^{3n} 3^n)$

# **Böl-ve-Fethet (Divide & Conquer) Tasarım Yöntemi**



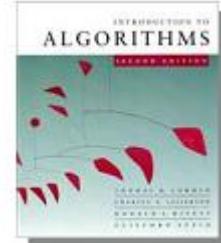
# Böl-ve-Fethet (Divide & Conquer)

- Böl ve fethet teknigiyle algoritma tasarıımı:
  - Problem kendisine benzer küçük boyutlu alt problemlere bölünür. Alt problemler çözülür ve bulunan çözümler birleştirilir.
  - **Divide:** Problem iki veya daha fazla alt problemlere bölünür.
  - **Conquer:** Alt problemleri özyinelemeli olarak çözüp, onları **fethet**.
  - **Combine:** Alt problem çözümlerini **birleştir**.

# Merge Sort (Birleştirme sıralaması) Algoritması

- **1. Böl:** Eğer  $S$  en az iki elemana sahipse ( $S$  sıfır veya bir elemana sahipse hiçbir işlem yapılmaz), bütün elemanlar  $S$ 'e n alınır ve  $S_1$  ve  $S_2$  adlı iki alana yerleştirilir, her biri  $S$  dizisinin yarısına sahiptir, (örn.  $S_1$  ilk  $\lceil n/2 \rceil$  elemana ve  $S_2$  ise ikinci  $\lfloor n/2 \rfloor$  elemana sahiptir).
- **2. Fethet:**  $S_1$  ve  $S_2$  Merge Sort kullanılarak sıralanır.
- **3. Birleştir:**  $S_1$  and  $S_2$  içindeki sıralı elemanlar tekrar  $S$  içerisinde tek bir sıralı dizi oluşturacak şekilde aktarılır.

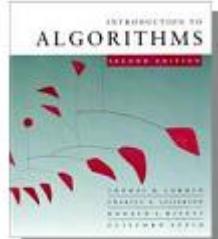
# Birleştirme sıralaması



- 1. Bölmek: Kolay.
  - 2. Hükmetmek: 2 alt dizilimi özyinelemeli sıralama.
  - 3. Birleştirmek: Doğrusal-zamanda birleştirme.

A diagram illustrating the recurrence relation  $T(n) = 2T(n/2) + \Theta(n)$ . At the top, the equation is shown with three yellow circles highlighting the term  $2T(n/2)$ . Below the equation, three arrows point downwards to the terms: the left arrow points to the text "altproblem sayısı", the middle arrow points to the text "altproblem boyutu", and the right arrow points to the text "bölme ve birleştirme işi".

# Master teoremi (hatırlatma)



$$T(n) = a T(n/b) + f(n)$$

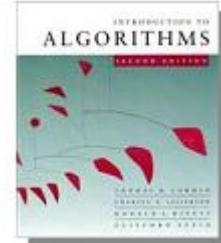
**DURUM 1:**  $f(n) = O(n^{\log_b a - \varepsilon})$ , sabit  $\varepsilon > 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a})$ .

**DURUM 2:**  $f(n) = \Theta(n^{\log_b a} \lg^k n)$ , sabit  $k \geq 0$   
 $\Rightarrow T(n) = \Theta(n^{\log_b a} \lg^{k+1} n)$ .

**DURUM 3:**  $f(n) = \Omega(n^{\log_b a + \varepsilon})$ , sabit  $\varepsilon > 0$ ,  
ve düzenleyici koşul (regularity condition).  
 $\Rightarrow T(n) = \Theta(f(n))$ .

*Birleştirme sıralaması:*  $a = 2$ ,  $b = 2 \Rightarrow n^{\log_b a} = n^{\log_2 2} = n$   
 $\Rightarrow$  DURUM 2 ( $k = 0$ )  $\Rightarrow T(n) = \Theta(n \lg n)$ .

# İkili arama (Binary Search)



- Sıralı dizilimin bir elemanını bulma:

*INPUT:*  $A[1..n]$  – sıralı (azalmayan) integer sayı dizisi,  $s$  – aranan integer sayı.

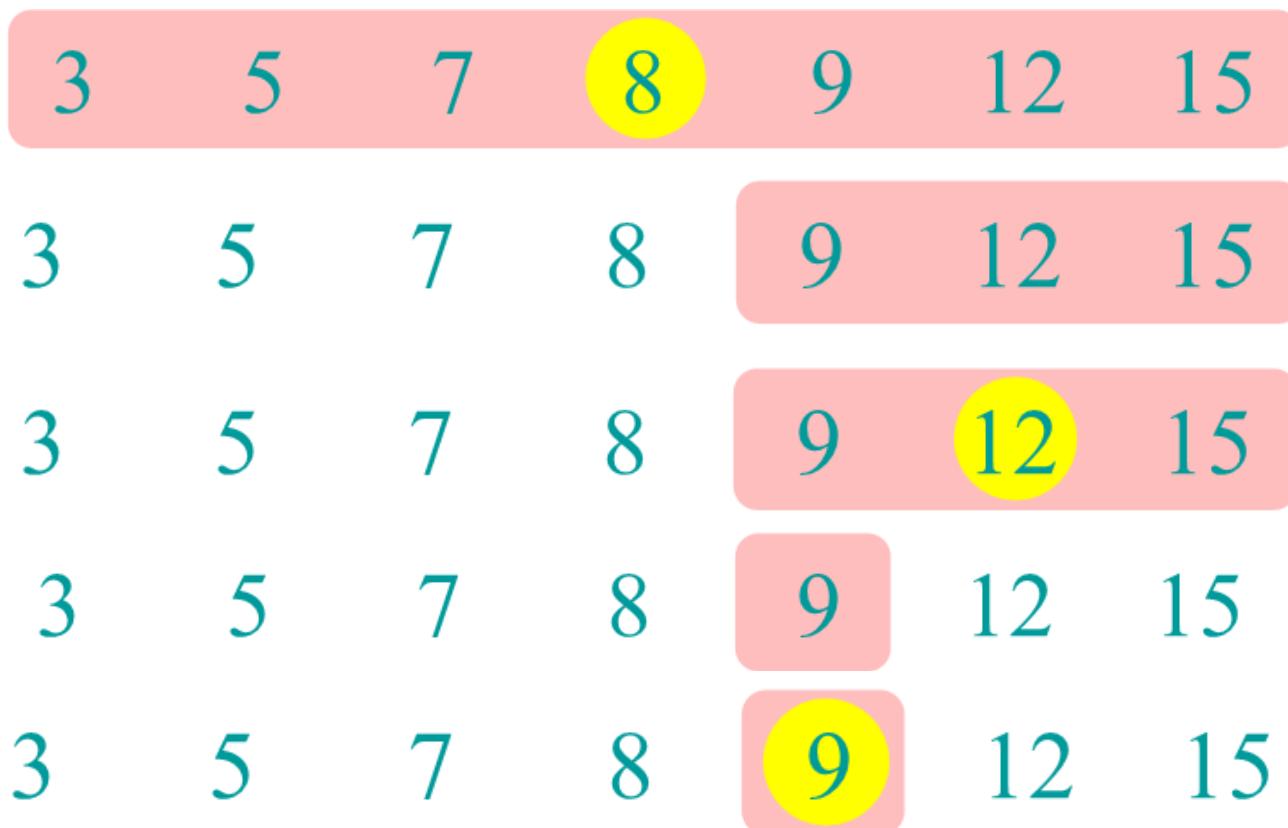
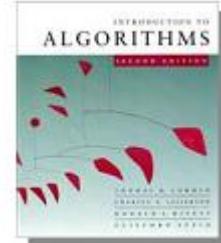
*OUTPUT:*  $j$  bulunan sayının indeksi  $A[j] = s$ .  $\text{NIL}$ , if  $\forall j (1 \leq j \leq n): A[j] \neq s$

```
Binary-search(A, p, r, s):
 if p = r then
 if A[p] = s then return p
 else return NIL
 q ← ⌊(p+r)/2⌋
 ret ← Binary-search(A, p, q, s)
 if ret = NIL then
 return Binary-search(A, q+1, r, s)
 else return ret
```

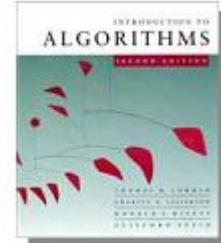
- 1. **Böl**: Orta elemanı belirle.
- 2. **Hükmet**: 1 alt dizilimde özyinelemeli arama yap.
- 3. **Birleştir**: Kolay.
- Örnek: 9' u bul.

3    5    7    8    9    12    15

# İkili arama (Binary Search)



# İkili arama için yineleme

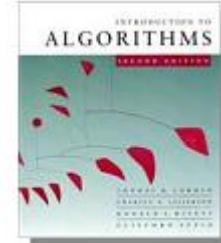


$$T(n) = \Theta(1) T(n/2) + \Theta(1)$$

altproblem sayısı       $\nearrow$   
*altproblem boyutu*       $\nwarrow$   
*bölme ve  
birleştirme işi*

$$\begin{aligned}
 n^{\log_b a} &= n^{\log_2 1} = n^0 = 1 \Rightarrow \text{DURUM 2 } (k=0) \\
 \Rightarrow T(n) &= \Theta(\lg n) .
 \end{aligned}$$

# Bir sayının üstellenmesi



○ **Problem:**  $a^n$  'yi  $n \in N$  iken hesaplama.

○ **Saf (Naive) algorithm:**  $\Theta(n)$ .

○ **Böl-ve-fethet algoritması:**

**Algorithm Power(x, n):**

**Input:** x sayısı ve n tamsayısi,  $n \geq 0$

**Output:**  $x^n$  değeri

**if**  $n = 0$  **then**

**return** 1

**if**  $n$  is odd **then**

$y = \text{Power}(x, (n - 1)/2)$

**return**  $x \cdot y \cdot y$

**else**

$y = \text{Power}(x, n/2)$

**return**  $y \cdot y$

Her özyineli çağrımda n sayısını 2'ye böliyoruz; dolayısıyla,  $\log n$  özyineli çağrı yaparız. Bu metod  $O(\log n)$  zamanına sahiptir.

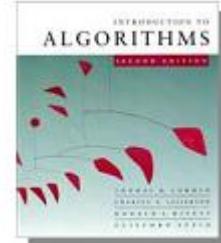
Burada ara sonucu y değişkeni ile göstermemiz önemli; şayet metod çağrıma yazarsak metod 2 defa çağrılmış olur.

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & n \text{ çift sayıysa;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & n \text{ tek sayıysa.} \end{cases}$$

$$T(n) = T(n/2) + \Theta(1) \Rightarrow T(n) = \Theta(\lg n).$$

# Fibonacci sayıları

## Özyinelemeli tanım:



$$F_n = \begin{cases} 0 & \text{eğer } n = 0; \\ 1 & \text{eğer } n = 1; \\ F_{n-1} + F_{n-2} & \text{eğer } n \geq 2 \text{ ise.} \end{cases}$$

0    1    1    2    3    5    8    13    21    34    ...

**Saf özyinelemeli algoritma:**  $\Omega(\phi^n)$   
 (üstel zaman), buradaki  $\phi = (1 + \sqrt{5})/2$   
**altın oran'dır (*golden ratio*).**

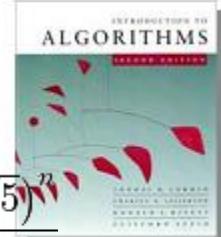
$$\varphi = \frac{1 + \sqrt{5}}{2} \approx 1.6180339887\ldots$$

$$\psi = \frac{1 - \sqrt{5}}{2} = 1 - \varphi = -\frac{1}{\varphi} \approx -0.6180339887\ldots$$

# Fibonacci sayılarını hesaplama

$$F_n = F(n) = \begin{cases} 0 & n = 0; \\ 1 & n = 1; \\ F(n-1) + F(n-2) & n > 1. \end{cases}$$

$$n = 0; \quad n = 1; = \frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}} = \frac{\varphi^n - (\varphi - \sqrt{5})^n}{\sqrt{5}}$$



- **Aşağıdan yukarıya algortiması:**

- $F_0, F_1, F_2, \dots, F_n$ 'i sırayla, her sayı iki öncekinin toplamı olacak şekilde hesaplayın.

- Yürütmü süresi:  $\Theta(n)$ .

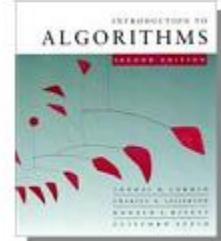
- **Saf özyinelemeli kare alma (Naive recursive squaring) algortiması:**

- $F_n = \varphi^n / \sqrt{5}$  yakın tamsayı yuvarlaması.

- **Özyinelemeli kare alma algortması:**  $\Theta(\lg n)$  zamanı.

- Bu yöntem güvenilir değildir, çünkü yüzey-nokta aritmetiği yuvarlama hatalarına gebedir.

# Özyineleme ile kare alma (Recursive squaring)



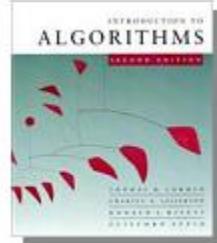
**Teorem:** 
$$\begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n.$$

**Algoritma:** Özyineleme ile kare alma.  
Süre =  $\Theta(\lg n)$ .

*Teoremin ispatı.* ( $n$  'de tümevarım)

Taban ( $n = 1$ ): 
$$\begin{bmatrix} F_2 & F_1 \\ F_1 & F_0 \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^1$$

# Özyineleme ile kare alma (Recursive squaring)

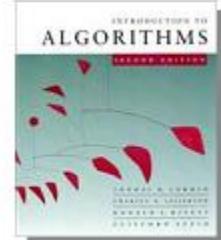


Tümevarım adımı ( $n \geq 2$ ):

$$\begin{aligned}
 \begin{bmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{bmatrix} &= \begin{bmatrix} F_n & F_{n-1} \\ F_{n-1} & F_{n-2} \end{bmatrix} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{n-1} \cdot \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^n
 \end{aligned}$$

■

# Matrislerde çarpma



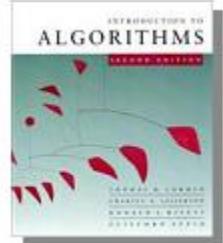
**Girdi:**  $A = [a_{ij}], B = [b_{ij}]$ .    **Cıktı:**  $C = [c_{ij}] = A \cdot B$ .     $\left. \begin{array}{l} \\ \end{array} \right\} i, j = 1, 2, \dots, n.$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^n a_{ik} \cdot b_{kj}$$

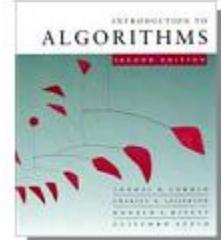
# Matrislerde çarpma

## Standart algoritma



```
for $i \leftarrow 1$ to n (i 1'den n 'ye kadar)
 do for $j \leftarrow 1$ to n (j 1'den n 'ye kadar)
 do $c_{ij} \leftarrow 0$
 for $k \leftarrow 1$ to n
 do $c_{ij} \leftarrow c_{ij} + a_{ik} \cdot b_{kj}$
```

Koşma süresi =  $\Theta(n^3)$



# Böl-ve-fethet algoritması

**FIKIR:**

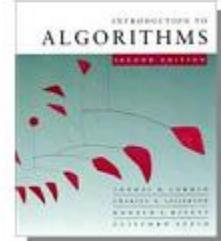
$n \times n$  matris =  $(n/2) \times (n/2)$  altmatrisin  $2 \times 2$  matrisi:

$$\begin{bmatrix} r & s \\ t & u \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \cdot \begin{bmatrix} e & f \\ g & h \end{bmatrix}$$

$$C = A \cdot B$$

$$\left. \begin{array}{l} r = ae + bg \\ s = af + bh \\ t = ce + dh \\ u = cf + dg \end{array} \right\} \begin{array}{l} \text{recursive (özyinelemeli)} \\ 8 \text{ çarpma } (n/2) \times (n/2) \text{ altmatriste,} \\ 4 \text{ toplama } (n/2) \times (n/2) \text{ altmatriste.} \end{array}$$

# Böl-ve-Fethet algoritmasının çözümlemesi



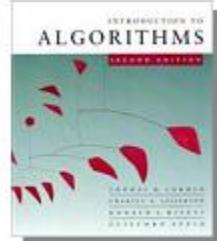
$$T(n) = 8T(n/2) + \Theta(n^2)$$

altmatris sayısı      altmatris boyutu      altmatrisleri toplama işi

$$n^{\log_b a} = n^{\log_2 8} = n^3 \Rightarrow \text{DURUM 1} \Rightarrow T(n) = \Theta(n^3)$$

*Sıradan algoritmadan daha iyi değil.*

# Strassen'in fikri



- **2×2** matrisleri yalnız **7** özyinelemeli çarpmayla çöz.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$s = P_1 + P_2$$

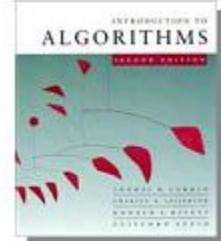
$$t = P_3 + P_4$$

$$u = P_5 + P_1 - P_3 - P_7$$

**7** çarp., **18** topl. /çıkar.

**Not:** Çarpma işleminde sırasımsızlık yok!

# Strassen'in fikri



- **2×2** matrisleri yalnız **7** özyinelemeli çarpmayla çöz.

$$P_1 = a \cdot (f - h)$$

$$P_2 = (a + b) \cdot h$$

$$P_3 = (c + d) \cdot e$$

$$P_4 = d \cdot (g - e)$$

$$P_5 = (a + d) \cdot (e + h)$$

$$P_6 = (b - d) \cdot (g + h)$$

$$P_7 = (a - c) \cdot (e + f)$$

$$r = P_5 + P_4 - P_2 + P_6$$

$$= (a + d)(e + h)$$

$$+ d(g - e) - (a + b)h$$

$$+ (b - d)(g + h)$$

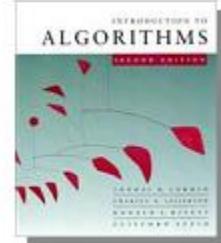
$$= ae + ah + de + dh$$

$$+ dg - de - ah - bh$$

$$+ bg + bh - dg - dh$$

$$= ae + bg$$

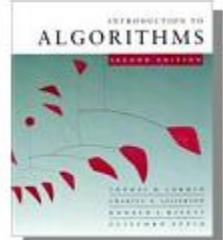
# Strassen'in algoritması



- **1. Böl:** A ve B'yi  $(n/2) \times (n/2)$  altmatrislere böl. + ve - kullanarak çarpılabilen terimler oluştur. ( $\Theta(n^2)$ )
- **2. Fethet:**  $(n/2) \times (n/2)$  altmatrislerde özyinelemeli 7 çarpma yap ( $P_1, P_2, P_3, \dots, P_7$ )
- **3. Birleştir:** + ve - kullanarak  $(n/2) \times (n/2)$  altmatrislerde C'yi oluştur. ( $\Theta(n^2)$ )

$$T(n) = 7 T(n/2) + \Theta(n^2)$$

# Strassen'in algoritması



$$T(n) = 7 T(n/2) + \Theta(n^2)$$

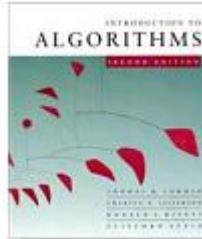
$$n^{\log_b a} = n^{\log_2 7} \approx n^{2.81} \Rightarrow \text{DURUM 1} \Rightarrow T(n) = \Theta(n^{\lg 7})$$

- **2.81** değeri **3'** den çok küçük görünmeyebilir ama, fark üstelde olduğu için, yürütüm süresine etkisi kayda değerdir. Aslında,  **$n \geq 32$**  değerlerinde Strassen'in algoritması günün makinelerinde normal algoritmadan daha hızlı çalışır. **Bugünün en iyi değeri** (teorik merak açısından):  **$\Theta(n^{2.376\dots})$**

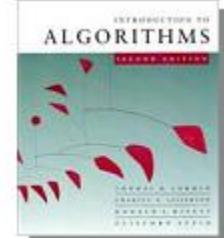
## Böl ve Fethet

### VLSI (Very Large Scale Integration) yerleşimi (Çok Büyük Çapta Tümleşim)

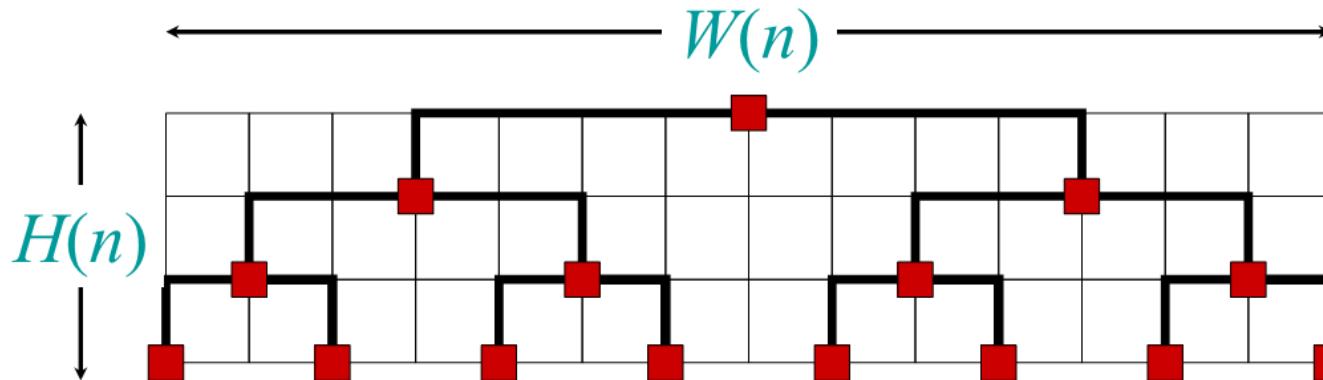
- Bilgisayar çipleri yada yongaları bildiğiniz gibi çok büyük çapta tümleşim kullanırlar.
- Elimizde bir devre olduğunu düşünelim ve bu devrenin de bir ikili ağaç olduğunu kabul edelim. Ama şimdilik bu devrenin bir kısmını ele alalım ama siz bunu tüm devre kabul edin.
- **Problem:**  $n$  yaprağı olan tam bir ikili ağaç en az alan kullanarak bir ızgaraya gömmek.



# VLSI (Very Large Scale Integration) yerleşimi (Çok Büyük Çapta Tümleşim)

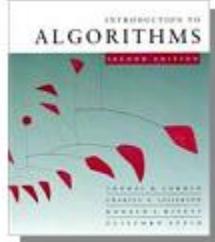


- Problem:  $n$  yaprağı olan tam bir ikili ağacı en az alan kullanarak bir ızgaraya gömmek.

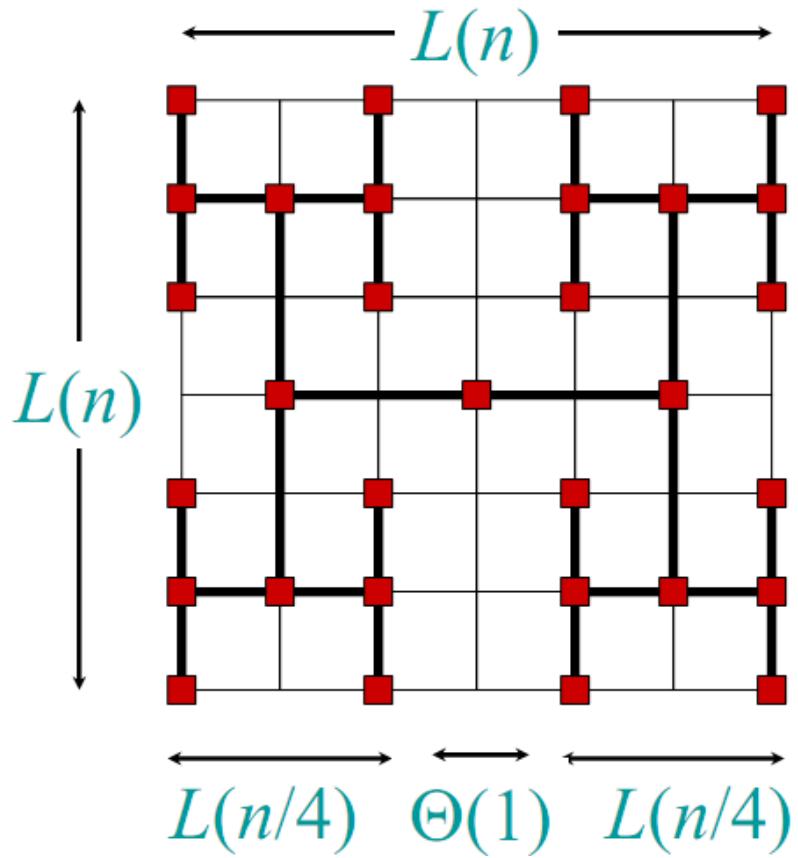


$$\begin{aligned}
 H(n) &= H(n/2) + \Theta(1) & W(n) &= 2W(n/2) + \Theta(1) \\
 &= \Theta(\lg n) & &= \Theta(n)
 \end{aligned}$$

Alan =  $\Theta(n \lg n)$



## H-ağacını gömme

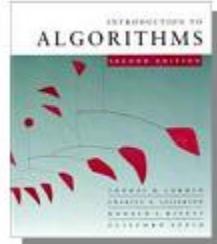


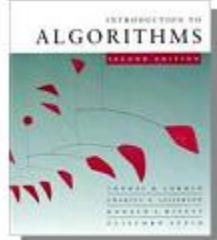
$$\begin{aligned}L(n) &= 2L(n/4) + \Theta(1) \\&= \Theta(\sqrt{n})\end{aligned}$$

Alan =  $\Theta(n)$

# Sonuç

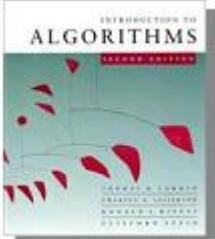
- Böl ve Fethet algoritma tasarımının güçlü tekniklerinden sadece biridir.
- Böl ve Fethet algoritmaları yinelemeler ve Ana (Master) metot kullanarak çözümlenebilir.
- Böl ve Fethet stratejisi genellikle verimli algoritmala görevidir.





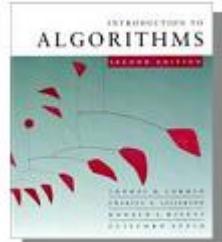
# Ortak Reküranslar

| Rekürans İlişkisi                          | Kapalı Form          | Örnek         |
|--------------------------------------------|----------------------|---------------|
| $c(1) = a$                                 |                      |               |
| $c(n) = b + c(n-1)$                        | $c(n) = O(n)$        | Linear search |
| $c(n) = b*n + c(n-1)$                      | $c(n) = O(n^2)$      | Quicksort     |
| $c(n) = b + c(n/2)$                        | $c(n) = O(\log(n))$  | Binary search |
| $c(n) = b*n + c(n/2)$                      | $c(n) = O(n)$        |               |
| $c(n) = b + kc(n/k)$                       | $c(n) = O(n)$        |               |
| $c(n) = b*n + 2c(n/2)$                     | $c(n) = O(n\log(n))$ | Mergesort     |
| $c(n) = b*n + kc(n/k)$                     | $c(n) = O(n\log(n))$ |               |
| $c(2) = b$<br>$c(n) = c(n-1) + c(n-2) + d$ | $c(n) = O(2^n)$      | Fibonacci     |

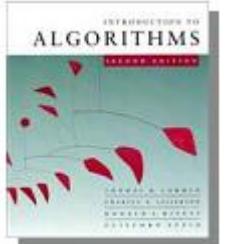


## Sorular

- 1.  $T(n)=3T(\sqrt{2n})+2$  tekrarlı bağıntısını çözünüz.
- 2.  $T(n)=3T(\lfloor n/5 \rfloor )+n$  tekrarlı bağıntısının çözümünü iteratif yolla gerçekleştiriniz. Bu bağıntının Özyineleme ağacı nedir?
- 3. Özyineleme ağacını kullanarak  $T(n)=T(n/3)+T(2n/3)+n$  bağıntısının çözümünü elde ediniz.
- 4.  $b \geq 1$  bir sabit olmak üzere  $T(n)=T(n/b)+T(b)+n$  tekrarlı bağıntısının Özyineleme ağacını elde ediniz ve bu bağıntının çözümü nedir?
- 5.  $0 < a < 1$  sabit olmak üzere  $T(n)=T(an)+T((1-a)n)+n$  tekrarlı bağıntısının Özdevinim ağacını elde ediniz ve asimptotik davranışı hakkında bilgi veriniz.



# Sorular



## Sorular

- 9. Aşağıdaki tekrarlı bağıntıları karakteristik denklem ve üreten fonksiyon yöntemleri ile çözünüz.
  - a)  $a_n = 5a_{n-1} - 6a_{n-2}$ ,  $a_1 = 36$  ve  $a_0 = 0$
  - b)  $a_n = 3a_{n-1} - 2a_{n-2} + 2^{n-1} + 2 \cdot 3^n$ ,  $a_1 = 29$  ve  $a_0 = 9$
  - c)  $a_n = a_{n-2} + 4n$ ,  $a_1 = 4$  ve  $a_0 = 1$
  - d)  $a_n = 3a_{n-1} - 2a_{n-2} + 3 \cdot 2^{2n-1}$ ,  $a_1 = 12$  ve  $a_0 = 0$
- 10. Master teoremini kullanarak aşağıdaki bağıntının mertebesini (çalışma zamanını) elde ediniz.
  - $T(n) = 16T(n/4) + O(n^2)$

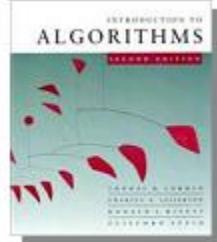
# **6.Hafta**

**Sıralama Algoritmaları  
Çabuk Sıralama, Rastgele  
Algoritmalar**

# **6.Hafta**

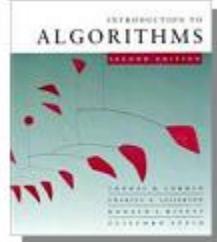
**Sıralama Algoritmaları  
Çabuk Sıralama, Rastgele  
Algoritmalar**

# Sıralama Algoritmaları



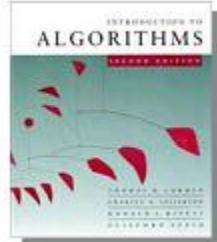
- Sıralama algoritmaların tipleri:
  - Karşılaştırmaya dayalı sıralama algoritmaları
    - Heap Sort, quicksort, insertion sort, bubble sort, merge sort,...
    - En iyi çalışma zamanı  $\Theta(n \log n)$
  - Doğrusal zaman sıralama algoritmaları
    - Counting sort (sayma), radix(taban) sort, bucket (sepet) sort.

# Yerinde Sıralama :In-place Sorting



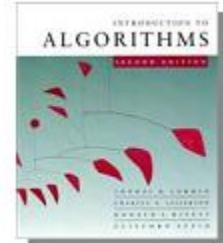
- Yerinde Sıralama: Algoritmanın, boyutu  $\Theta(n)$  olan ekstra depolama (tek değişkenli ve register (kayıtlar) dışında) alan gerektirmemesi.
  - Algoritma: Yerinde Sıralama
  - - Bubble sort Evet
  - - Insertion sort Evet
  - - Selection sort Evet
  - - Merge sort Hayır(ek alan gereklidir)
  - - Heap sort Evet
  - - Quick sort Evet

# Heap (Yığın ağacı)

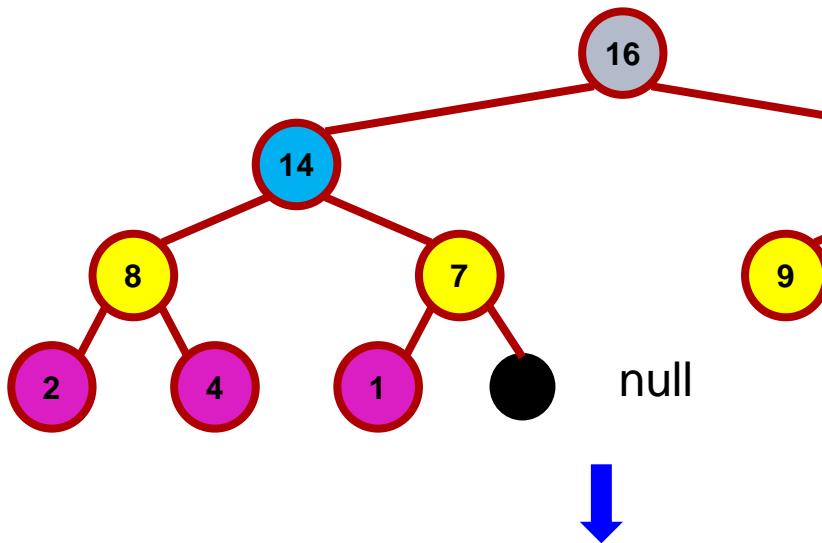


- Heap, ikili ağaç (binary tree) olarak düşünebileceğimiz bir veri yapısıdır.
  - Dizi
  - Complete binary tree yakın bir ağaç olarak görülebilir.
    - En düşük seviye hariç bütün seviyeler doludur.
  - Her düğümdeki veri kendi çocuk düğümlerinden büyük (max-heap) veya küçüktür (min-heap).

# Heap (Yığın ağacı)



- Complete binary tree: null değeri dolu olursa



|                    |                                      |
|--------------------|--------------------------------------|
| $\text{Parent}(i)$ | $\text{return } \lfloor i/2 \rfloor$ |
| $\text{Left}(i)$   | $\text{return } 2i$                  |
| $\text{Right}(i)$  | $\text{return } 2i + 1$              |

|    |    |    |   |   |   |   |   |   |    |
|----|----|----|---|---|---|---|---|---|----|
| 1  | 2  | 3  | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 16 | 14 | 10 | 8 | 7 | 9 | 3 | 2 | 4 | 1  |

Seviye: 3

2

1

0

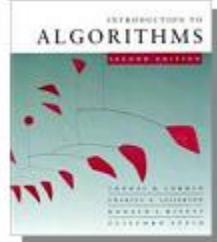
# Heap (Yığın ağacı)

- Heap, bir dizi olarak tasarlarsak

- Kök düğüm  $A[1]$ 'dir.
- $i.$  düğüm  $A[i]$
- $i.$  düğümün ebeveyni  $A[i/2]$  (tam bölme)
- $i.$  düğümün sol çocuğu  $A[2i]$
- $i.$  düğümün sağ çocuğu  $A[2i + 1]$

- $\text{Left}(i)=2i$
- $\text{Right}(i)=2i+1$
- $\text{Parent}(i)=\lfloor i / 2 \rfloor$

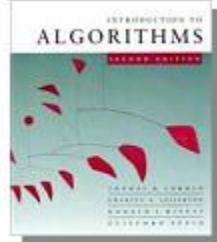
- Kök düğüm çocuklarından büyük ise max-heap
  - $A[\text{Parent}(i)] \geq A[i]$
- Kök düğüm çocuklarından küçük ise min-heap
  - $A[\text{Parent}(i)] \leq A[i]$



```

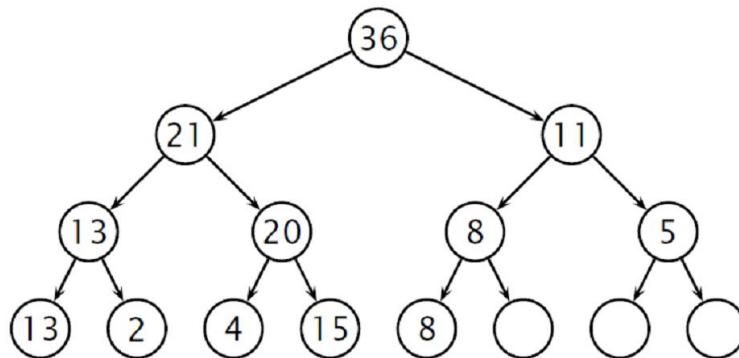
Parent(i) { return ⌊i/2⌋; }
Left(i) { return 2*i; }
Right(i) { return 2*i + 1; }

```

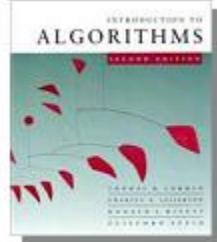


# Max-Heap Özelliği

- Kök düğüm çocuklarından büyük ise max-heap
  - $A[\text{Parent}(i)] \geq A[i]$ , bütün düğümler için  $i > 1$
  - Diğer bir deyişle düğümün değeri aynı zamanda onun ebeveynin değeridir.
  - Heap'in en büyük elamanı nerededir?

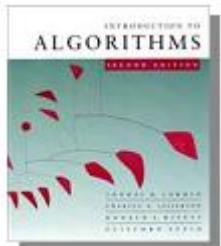


# Heap Yüksekliği



## ○ Tanım:

- Ağaçtaki bir düğümün yüksekliği; en alt seviyedeki yaprağa doğru gidilen yol üzerindeki kenarların sayısıdır.
- Ağacın yüksekliği; kök düğümün yüksekliğidir.
- $n$  elamanlı bir heap ağacının yüksekliği, temel heap işlemlerinin aldığı zaman ile orantılıdır.  $\Theta(\lg n)$



## Heap İşlemleri: Heapify()

- **Max\_Heapify()**: Temel heap özelliğini korumak.  
( $A[i]$  elamanını aşağıya taşıma)
  - Verilen:  $i$  düğümü ( $i$  ve  $r$  çocuklarına sahip)
  - Verilen:  $l$  ve  $r$  düğümleri (iki alt heap ağacının kökleri)
  - Eylem: Eğer  $A[i] < A[l]$  veya  $A[i] < A[r]$  ise,  $A[i]$  değerini,  $A[l]$  ve  $A[r]$  nin en büyük değeri ile yer değiştir.
  - Çalışma zamanı:  **$O(h)$ ,  $h = \text{height of heap} = O(\lg n)$**

# Heap İşlemleri: Heapify()

```
Max_Heapify (A, i)
```

```
{
```

```
 L = Left(i); R = Right(i);
```

```
 if (L <= heap_size(A) && A[L] > A[i])
```

```
 largest = L;
```

```
 else
```

```
 largest = i;
```

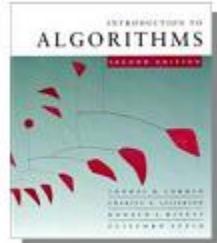
```
 if (R <= heap_size(A) && A[R] > A[largest])
```

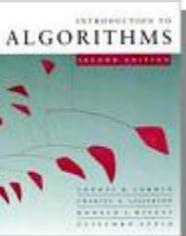
```
 largest = R;
```

```
 if (largest != i)
```

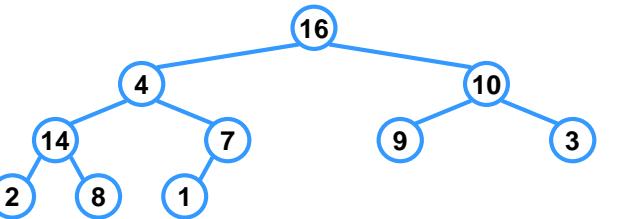
```
 Swap(A, i, largest); Max_Heapify (A,largest);
```

```
}
```

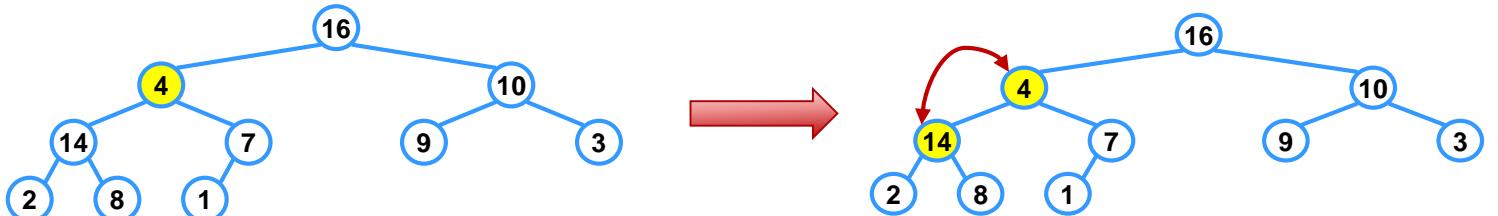




# Heapify(): Örnek

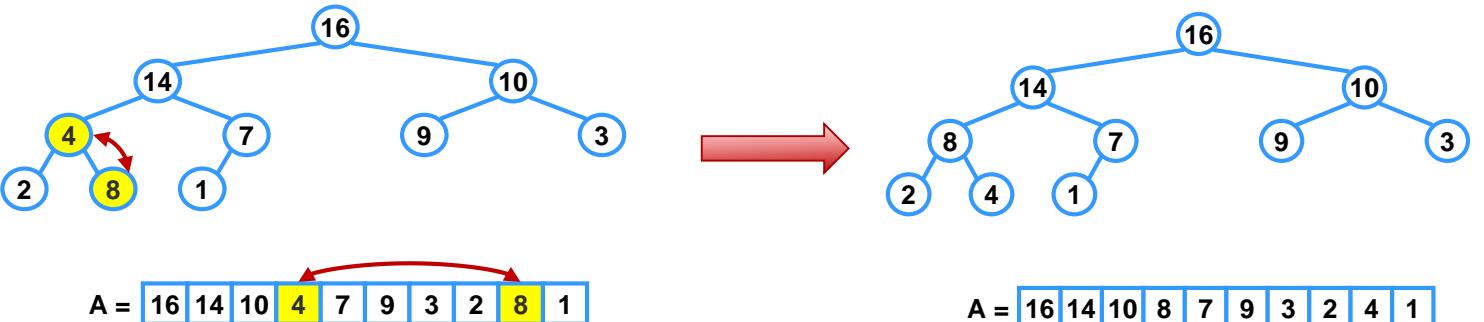


$A = [16 \ 4 \ 10 \ 14 \ 7 \ 9 \ 3 \ 2 \ 8 \ 1]$



$A = [16 \ 4 \ 10 \ 14 \ 7 \ 9 \ 3 \ 2 \ 8 \ 1]$

$A = [16 \ 4 \ 10 \ 14 \ 7 \ 9 \ 3 \ 2 \ 8 \ 1]$

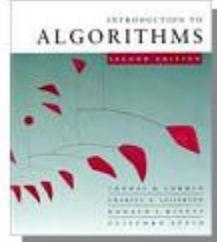


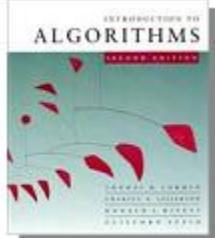
$A = [16 \ 14 \ 10 \ 4 \ 7 \ 9 \ 3 \ 2 \ 8 \ 1]$

$A = [16 \ 14 \ 10 \ 8 \ 7 \ 9 \ 3 \ 2 \ 4 \ 1]$

# Heapify() Analizi

- Heapify çalışma zamanı: i kök düğümüne sahip n boyutlu bir alt ağaç için
  - Elamanlar arasındaki ilişkiyi bulma:  $\Theta(1)$
  - En kötü durumda bir alt ağaç için en fazla  $2n/3$  düşüme sahiptir. Böylece en kötü durum için aşağıdaki rekürans ifade edilebilir.
  - $T(n) \leq T(2n/3) + \Theta(1)$
- Burada  $a=1$ ,  $b=3/2$  dir. Böylece  $n^{\log_b a} = n^{\log_{3/2} 1} = \theta(n^0) = 1$  ve  $f(n) = 1$  olduğundan master teoriminde **Durum 2** uygulanır ve çözüm,  $T(n) = \theta(\log n)$





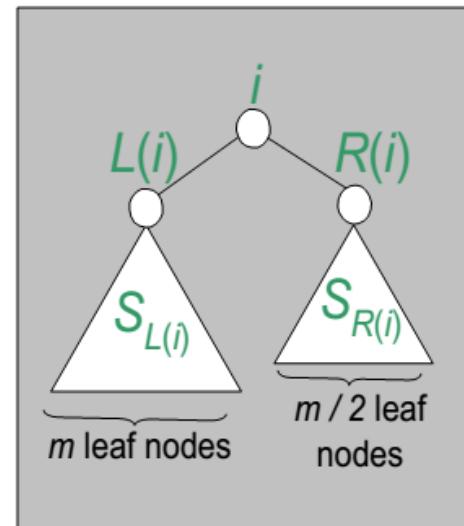
# Heapify() Analizi: ispat

- Let  $m$  be the number of leaf nodes in  $S_{L(i)}$
- $|S_{L(i)}| = \underbrace{m}_{\text{ext}} + \underbrace{(m - 1)}_{\text{int}} = 2m - 1$  ;
- $|S_{R(i)}| = \overbrace{m/2}^{\text{ext}} + \overbrace{(m/2 - 1)}^{\text{int}} = m - 1$
- $|S_{L(i)}| + |S_{R(i)}| + 1 = n$

$$(2m - 1) + (m - 1) + 1 = n \Rightarrow m = (n+1)/3$$

$$|S_{L(i)}| = 2m - 1 = 2(n+1)/3 - 1 = (2n/3 + 2/3) - 1 = 2n/3 - 1/3 \leq 2n/3$$

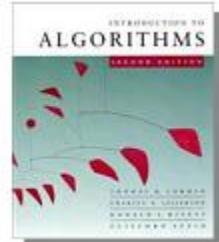
- $T(n) \leq T(2n/3) + \Theta(1) \Rightarrow T(n) = O(\lg n)$



By case 2 of  
Master Thm

# Heap İşlemleri :Heap Yapılandırması

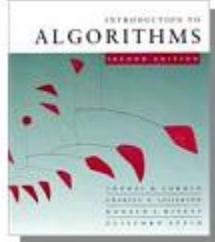
## BuildHeap()



- A[1..n] dizisinin  $n = \text{length}[A]$  uzunlığında olan bir heap' dönüştürülmesi.
- Alt dizideki  $A[\lfloor n / 2 \rfloor + 1] \dots n]$  elemanlar heap durumundadır.

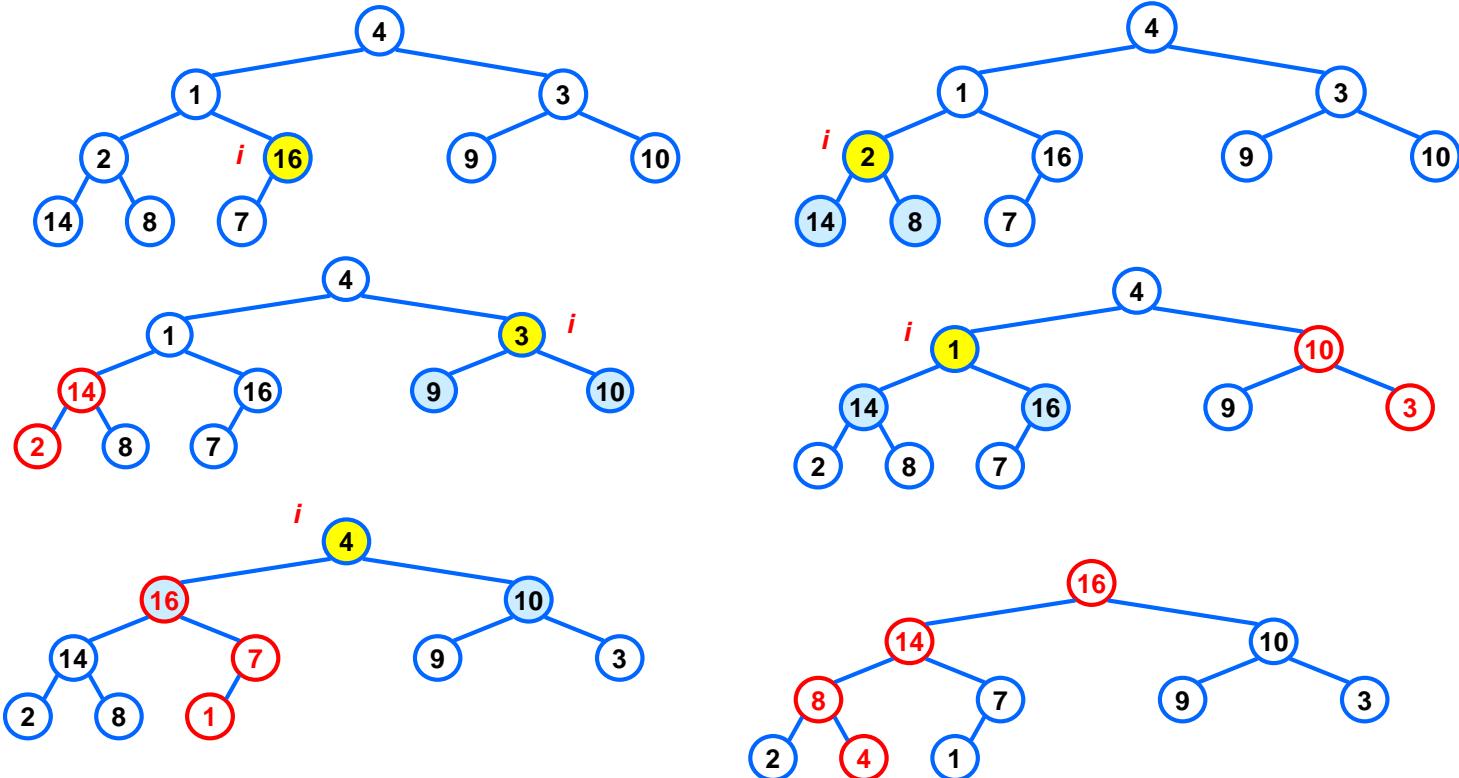
**BuildHeap(A)**

```
{
 heap_size(A) = length(A);
 for (i = ⌊length[A]/2⌋ downto 1)
 Max_Heapify(A, i);
}
```

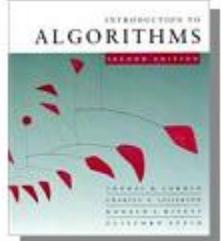


## Örnek:BuildHeap()

- A = {4, 1, 3, 2, 16, 9, 10, 14, 8, 7}

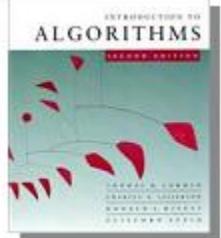


# BuildHeap() Analizi

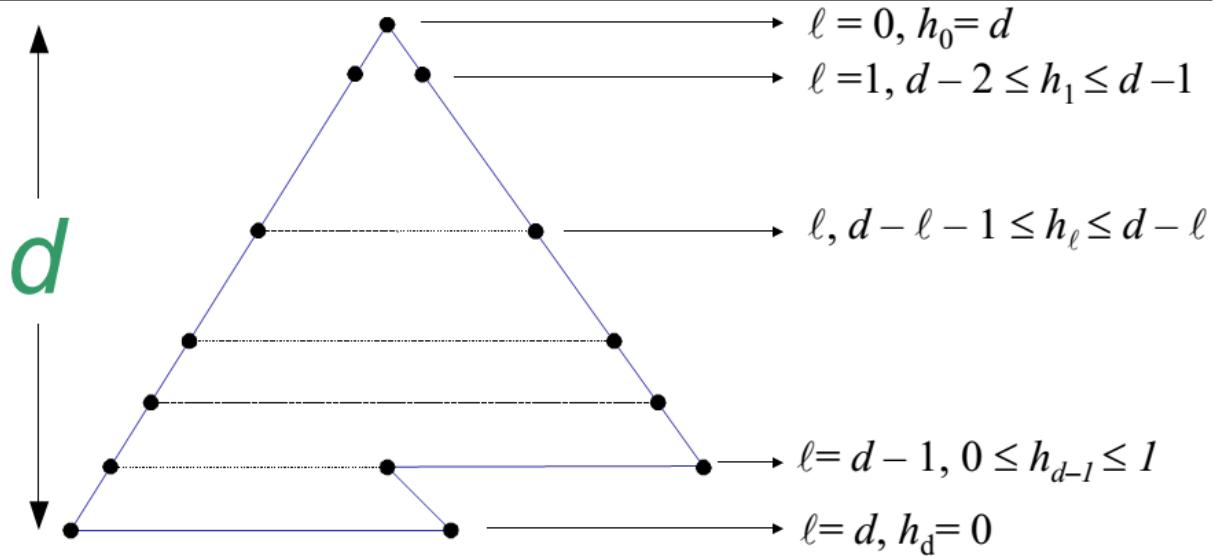


- **Heapify()** her çağrıldığında  $O(\lg n)$  zaman alır.
- **Max\_Heapify ( $A, i$ )**,  $\rightarrow O(h)$  zaman gerektirir, burada  $h \rightarrow i$ .  
düğümün yüksekliği yani  $\log n$  dir.
- Her seviyede  $O(n/2^{h+1})$ , **Max\_Heapify** çağrıları yapılınrsa  $O(n/2^h)$ ,  
(özellikle  $\lfloor n/2 \rfloor$  çağrılarında) BuildHeap çalışma zamanı aşağıdaki  
şekilde hesaplanır.

# BuildHeap() Analizi: İspat



## Build-Heap: tighter running time analysis



If the heap is complete binary tree then  $h_\ell = d - \ell$

Otherwise, nodes at a given level do not all have the same height

But we have  $d - \ell - 1 \leq h_\ell \leq d - \ell$

# BuildHeap() Analizi: İspat

Assume that all nodes at level  $\ell = d - 1$  are processed

$$T(n) = \sum_{\ell=0}^{d-1} n_\ell O(h_\ell) = O\left(\sum_{\ell=0}^{d-1} n_\ell h_\ell\right) \quad \begin{cases} n_\ell = 2^\ell = \# \text{ of nodes at level } \ell \\ h_\ell = \text{height of nodes at level } \ell \end{cases}$$

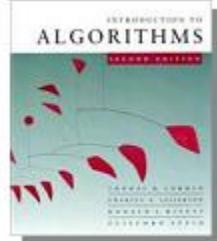
$$\therefore T(n) = O\left(\sum_{\ell=0}^{d-1} 2^\ell (d - \ell)\right)$$

Let  $h = d - \ell \Rightarrow \ell = d - h$  (change of variables)

$$T(n) = O\left(\sum_{h=1}^d h 2^{d-h}\right) = O\left(\sum_{h=1}^d h 2^{d-h}\right) = O\left(2^d \sum_{h=1}^d h (1/2)^h\right)$$

$$\text{but } 2^d = \Theta(n) \Rightarrow T(n) = O\left(n \sum_{h=1}^d h (1/2)^h\right)$$

# BuildHeap() Analizi



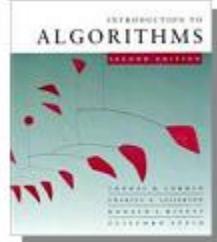
$$- T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} O(n/2^h)O(h) = O(n \times \left[ \frac{1}{2} + \frac{2}{4} + \dots + \frac{\lg n}{n} \right])$$

$$\sum_{h=0}^{\lfloor \lg n \rfloor} \left\lceil \frac{n}{2^{h+1}} \right\rceil O(h) = O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right)$$

$$\begin{aligned} O\left(n \sum_{h=0}^{\lfloor \lg n \rfloor} \frac{h}{2^h}\right) &= O\left(n \sum_{h=0}^{\infty} \frac{h}{2^h}\right) \quad \sum_{h=0}^{\infty} \frac{h}{2^h} &= \frac{1/2}{(1 - 1/2)^2} \\ &= 2 . \end{aligned}$$

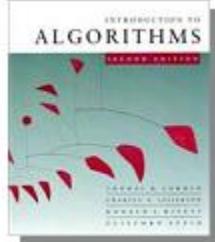
$$T(n) = O(n)$$

# Heap Sort Algoritması



**Heapsort (A)**

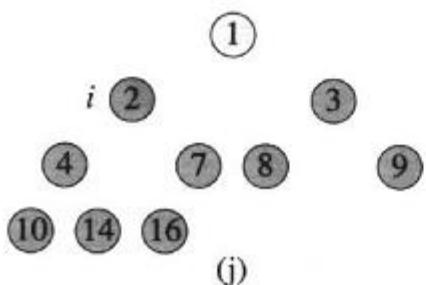
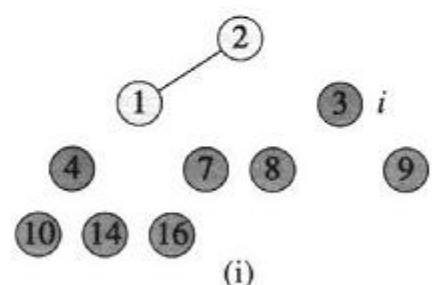
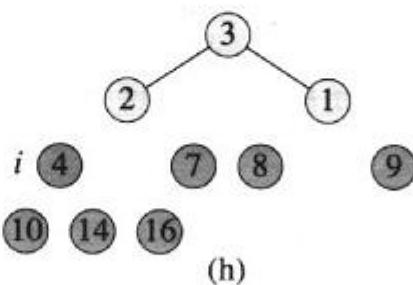
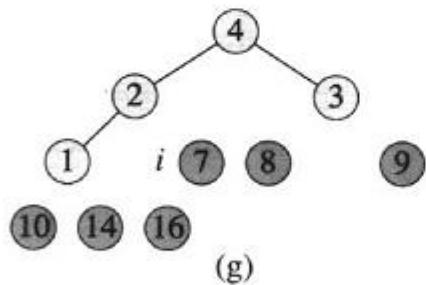
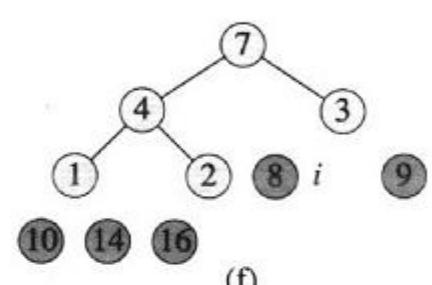
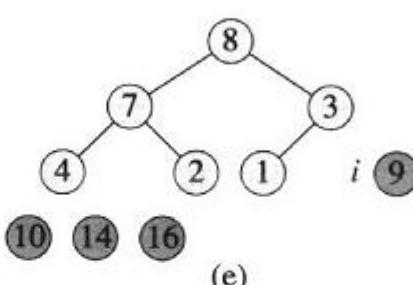
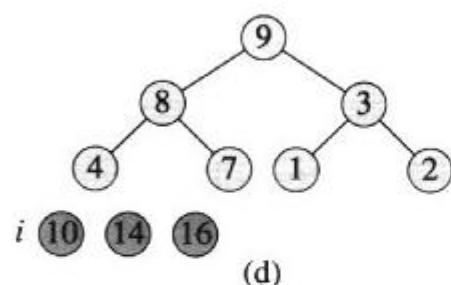
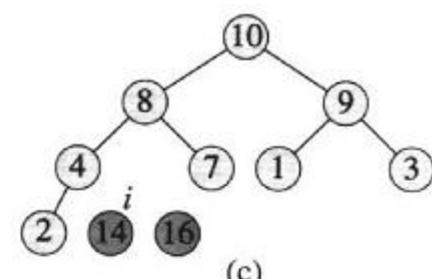
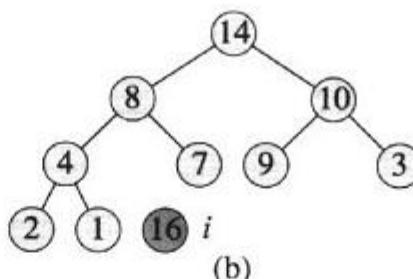
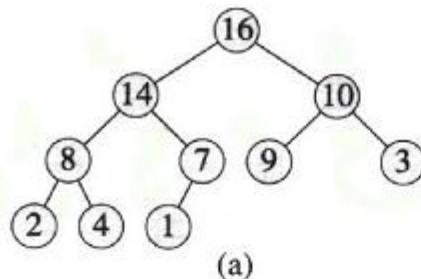
```
{ BuildHeap (A) ;
 for (i = length (A) downto 2)
 {
 Swap (A[1] , A[i]) ;
 heap_size (A) -= 1 ;
 Heapify (A, 1) ;
 }
 }
```



## Heap Sort Algoritması Analizi

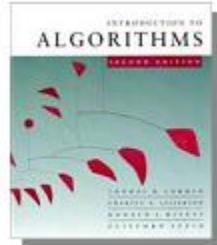
- **BuildHeap()** çağrılmaması:  $O(n)$  time
  - **Heapify()**  $n - 1$  çağrılmaması :  $O(\lg n)$  time
  - **HeapSort()** toplam çalışma zamanı
- 
- $T(n) = O(n) + (n - 1) O(\lg n)$
  - $T(n) = O(n) + O(n \lg n)$   
 $T(n) = O(n \lg n)$

# Heap Sort



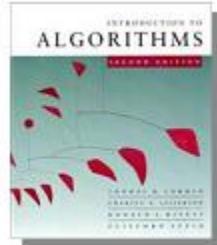
|     |   |   |   |   |   |   |   |    |    |    |
|-----|---|---|---|---|---|---|---|----|----|----|
| $A$ | 1 | 2 | 3 | 4 | 7 | 8 | 9 | 10 | 14 | 16 |
|-----|---|---|---|---|---|---|---|----|----|----|

(k)



# Heap Sort

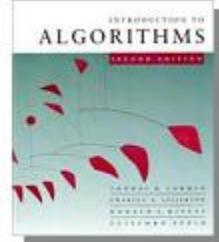
- Heapsort iyi bir algoritmadır fakat pratikte genelde Quicksort daha hızlıdır.
- Ancak heap veri yapısı, öncelik sırası uygulaması (*priority queues*) için inanılmaz faydalıdır.
  - Her biri ilişkili bir anahtar (key) veya değer olan elamanların oluşturduğu **A** dizisini muhafaza etmek için bir veri yapısı.
  - Desteklenen işlemler **Insert()**, **Maximum()**, ve **ExtractMax()**



# Heap Sort

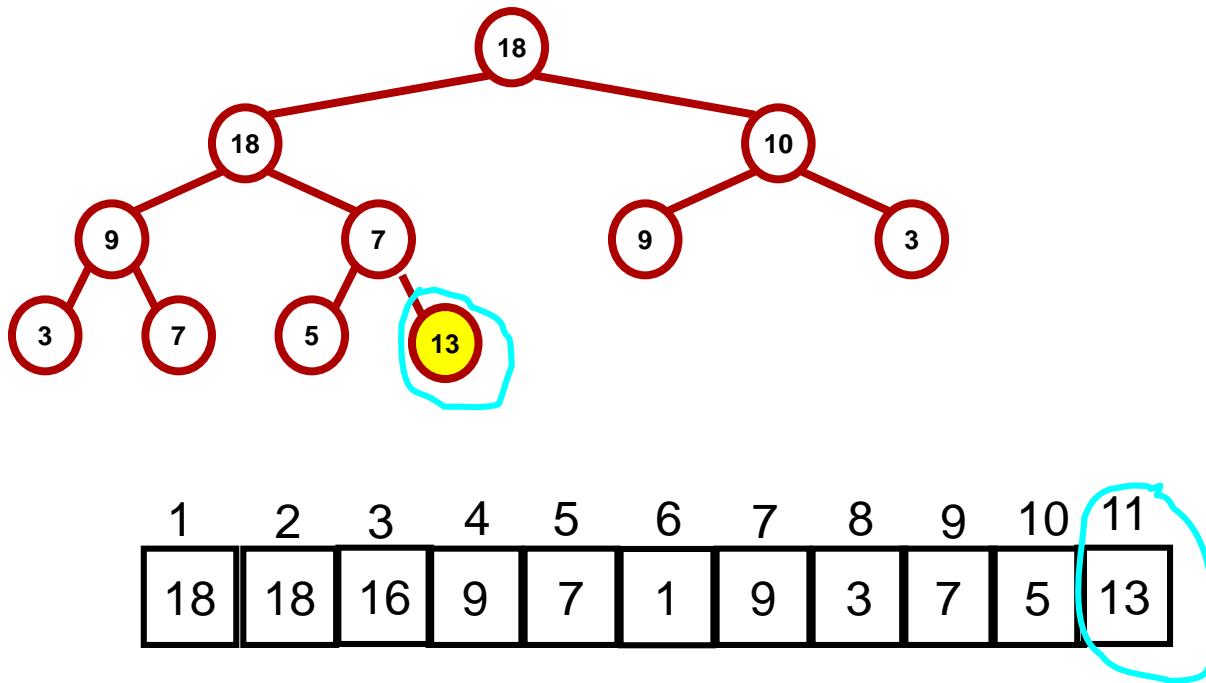
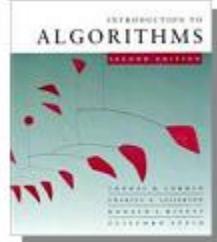
- **Insert( $S, x$ )** :  $S$  dizisine  $x$  elemanını ekler
- **Maximum( $S$ )**:  $S$  dizisindeki maksimum elamanı geri döndürür
- **ExtractMax( $S$ )**  $S$  dizisindeki maksimum elamanı geri döndürür ve elamanı diziden çıkarır

# Öncelikli Kuyruk Uygulamaları (Implementing Priority Queues)

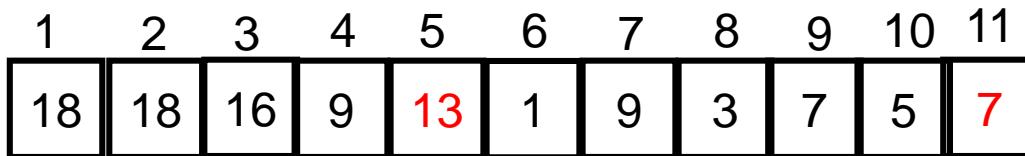
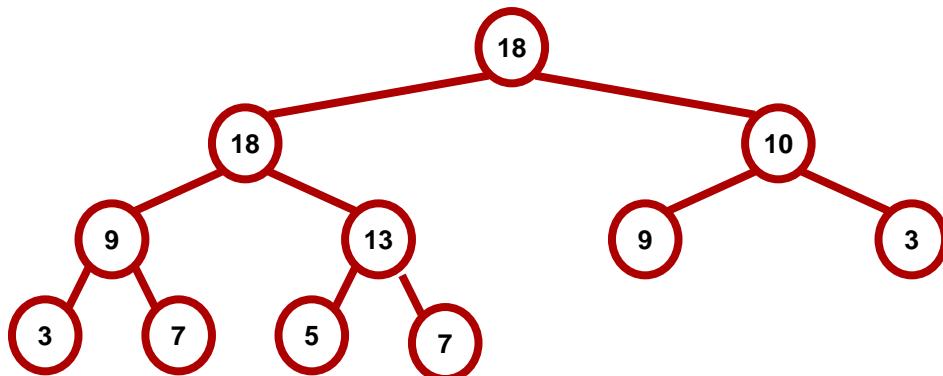
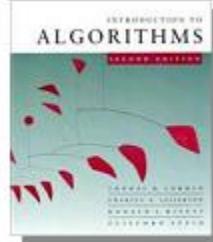


```
HeapInsert(A, key)
{
 heap_size[A]++;
 i = heap_size[A];
 while (i > 1 AND A[Parent(i)] < key)
 {
 A[i] = A[Parent(i)];
 i = Parent(i);
 }
 A[i] = key;
}
```

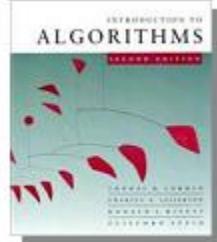
# Öncelikli Kuyruk Uygulamaları (Implementing Priority Queues)



# Öncelikli Kuyruk Uygulamaları (Implementing Priority Queues)



# Öncelikli Kuyruk Uygulamaları (Implementing Priority Queues)



**HeapMaximum (A)**

```
{ return A[1]; }
```

**HeapExtractMax (A)**

```
{
 if (heap_size[A] < 1) { error; }
 max = A[1];
 A[1] = A[heap_size[A]];
 heap_size[A] --;
 Heapify(A, 1);
 return max;
}
```

# Çabuk Sıralama, Rastgele Algoritmalar

- Böl ve fethet
- Böülüntüler
- En kötü durum çözümlemesi
- Sezgi (Öngörü)
- Rastgele çabuk sıralama
- Çözümleme

# Çabuk sıralama (Quick Sort)

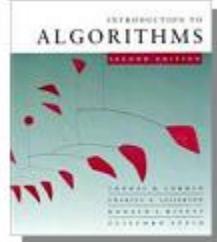
- C.A.R. Hoare tarafından 1962'de önerildi.
- Böl ve fethet algoritması.
- "Yerinde" sıralar (araya yerleştirme sıralamasında olduğu gibi; birleştirme sıralamasından farklı).
- (Ayar yapılrsa ) çok pratik.



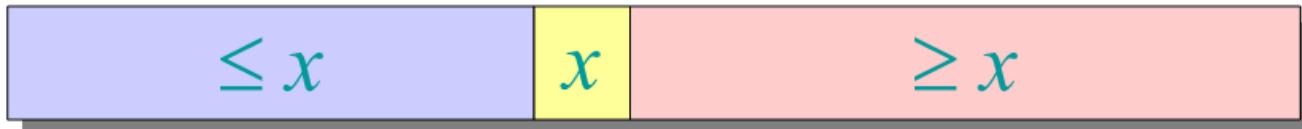
Sir Charles Antony Richard  
Hoare  
1934 -

# Çabuk sıralama (Quick Sort)

## Böl ve fethet



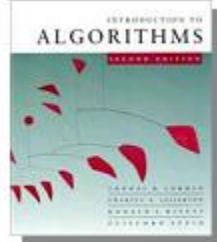
- **$n$ -elemanlı bir dizilimin çabuk sıralanması:**
- **1. Böl:** Dizilimi **pivot (eksen sabit)  $x$** 'in etrafında iki altdizilime bölüntüle; burada soldaki altdizilim elemanları  $\leq x \leq$  sağdaki altdizilim elemanları olsun.



- **2. Fethet:** İki altdizilimi özyinelemeli sırala.
- **3. Birleştir:** Önemsiz (yerinde sıraladığı için)  
**Anahtar:** Doğrusal-zamanlı ( $\Theta(n)$ )bölüntü altyordamı.

# Çabuk sıralama (Quick Sort)

## Böl ve fethet



- Quicksort algoritmasında yapılan ana iş öz yinelemede bölüntülere ayırma işlemidir. Bütün iş bölüntüleme de yapılmaktadır.
- Buradaki anahtar olay bölüntü alt yordamı doğrusal zamanda yani  $\Theta(n)$  olması.
- Merge sort algoritmasında ana iş ise öz yinelemeli birleştirme yapmadır.

## Çabuk sıralama (quicksort) için sözdekode

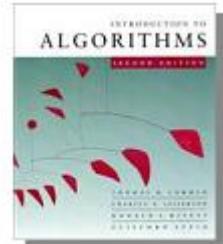
QUICKSORT( $A, p, r$ )

if  $p < r$

then  $q \leftarrow \text{PARTITION}(A, p, r)$

QUICKSORT( $A, p, q-1$ )

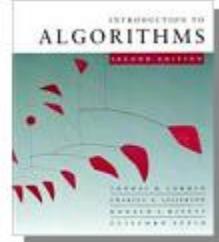
QUICKSORT( $A, q+1, r$ )



İlk arama: QUICKSORT( $A, 1, n$ )

# Çabuk sıralama (Quick Sort)

## Bölbüntüleme örneği



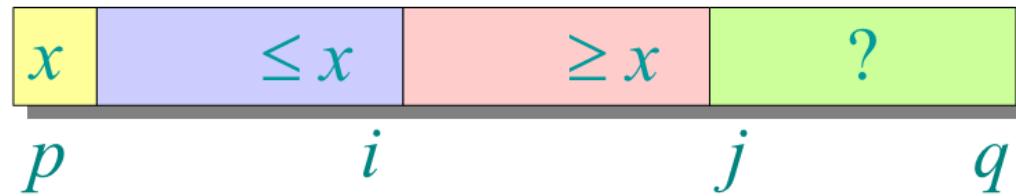
```

PARTITION (Bölbüntü)(A, p, q) $\triangleright A[p \dots q]$
 $x \leftarrow A[p]$ \triangleright pivot = $A[p]$
 $i \leftarrow p$ (eksen sabit)
 for $j \leftarrow p + 1$ to q
 do if $A[j] \leq x$ (öyleyse yap)
 then $i \leftarrow i + 1$
 exchange $A[i] \leftrightarrow A[j]$ (değiştir)
 exchange $A[p] \leftrightarrow A[i]$ (değiştir)
 return i (dön)

```

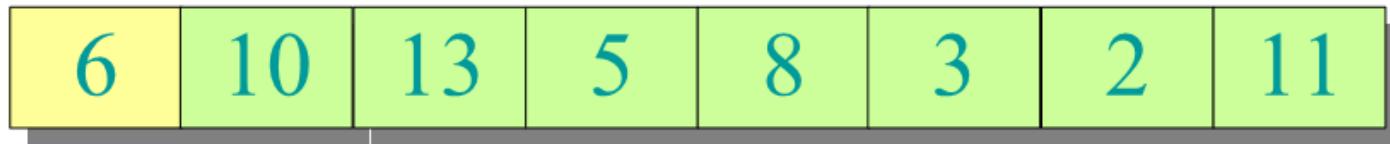
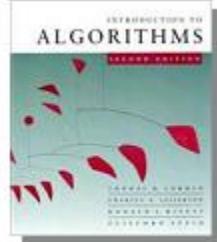
Koşma süresi  
 $=\mathcal{O}(n)$   
 $n$  eleman için

*Değişmez:*

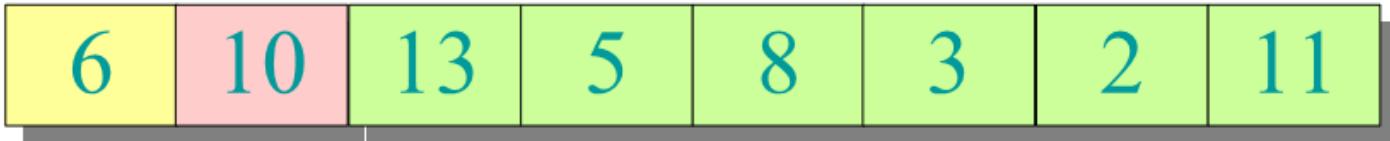


# Çabuk sıralama (Quick Sort)

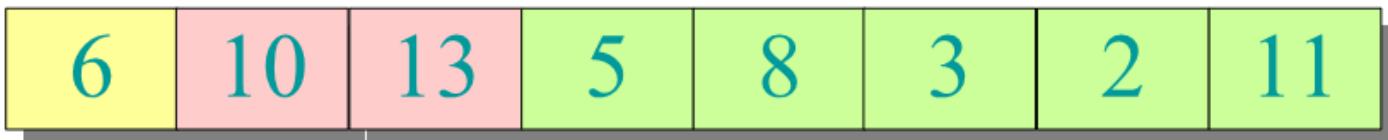
## Bölüntüleme örneği



$i$        $j$



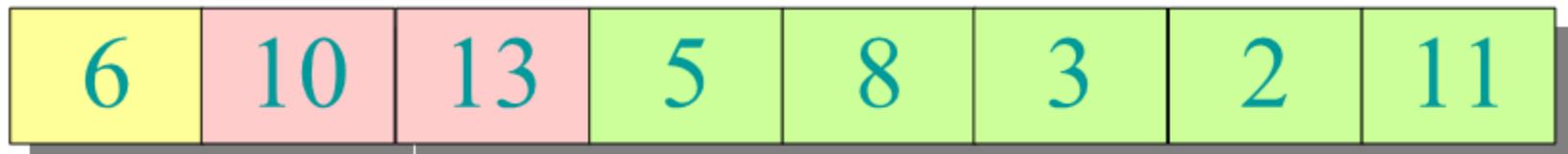
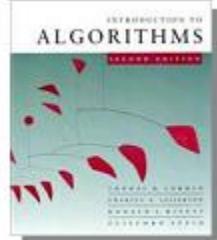
$i$        $\xrightarrow{} j$



$i$        $\xrightarrow{} j$

# Çabuk sıralama (Quick Sort)

## Bölüntüleme örneği

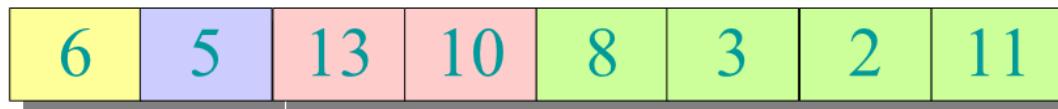
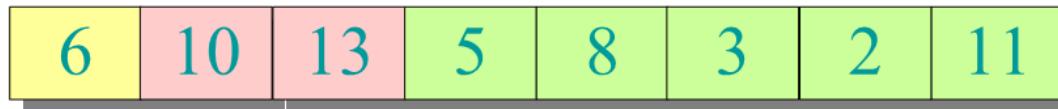
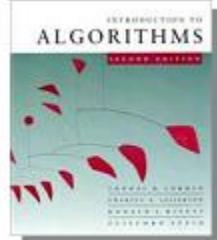


→  $i$

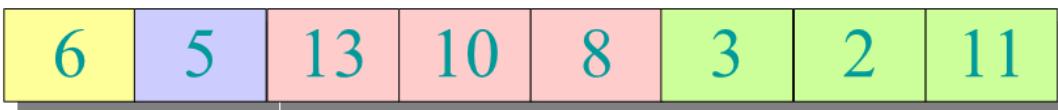
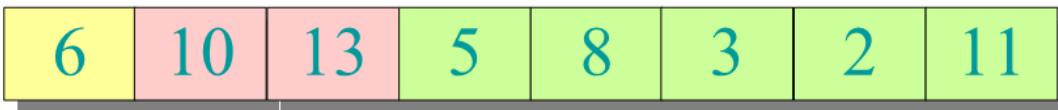
$j$

# Çabuk sıralama (Quick Sort)

## Bölüntüleme örneği



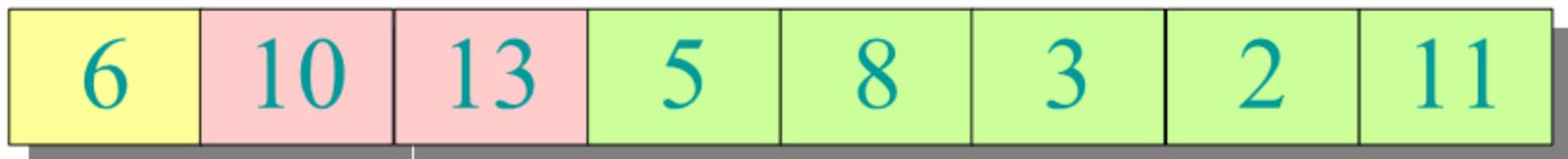
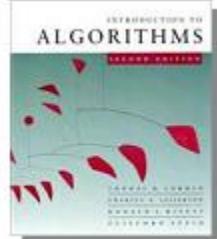
$i$                        $\longrightarrow j$



$i$                        $\longrightarrow j$

# Çabuk sıralama (Quick Sort)

## Bölüntüleme örneği

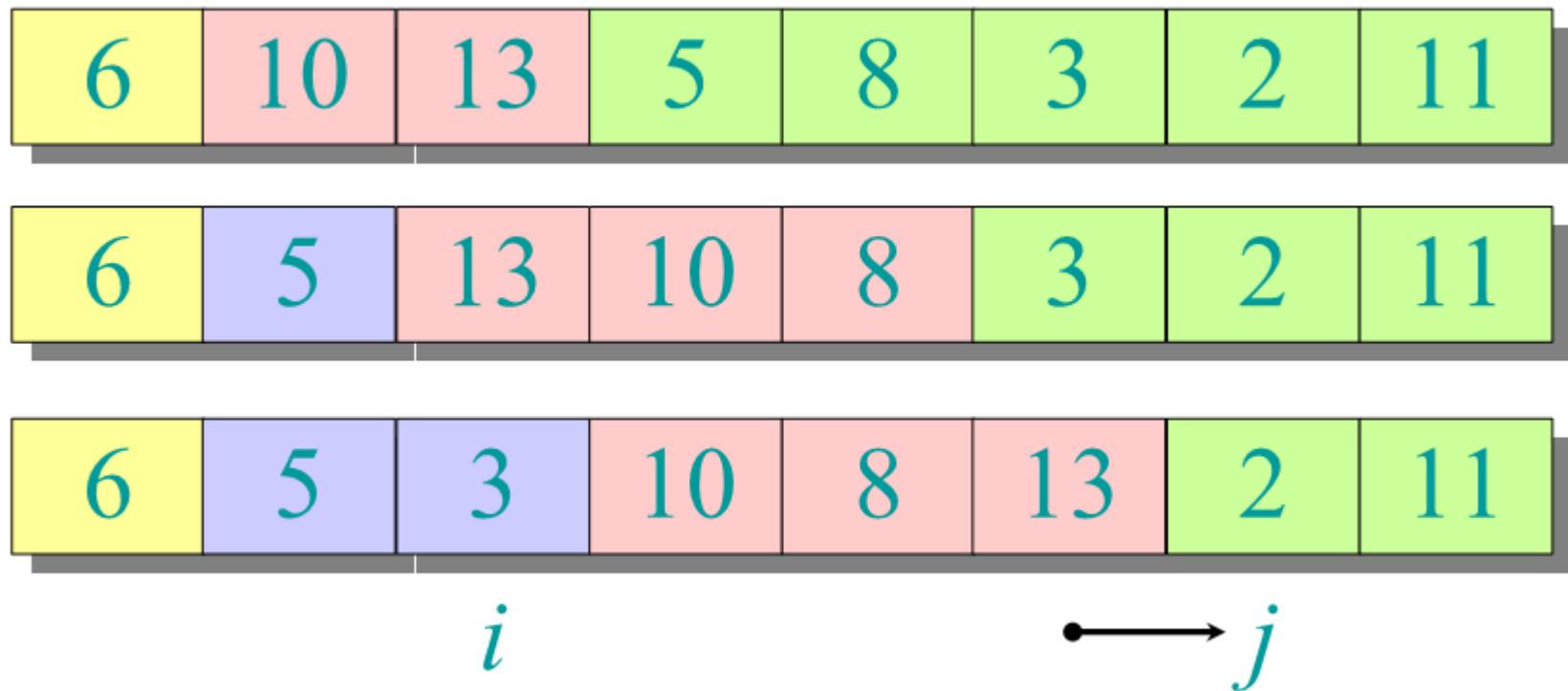
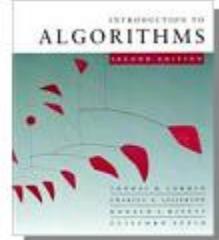


→ *i*

*j*

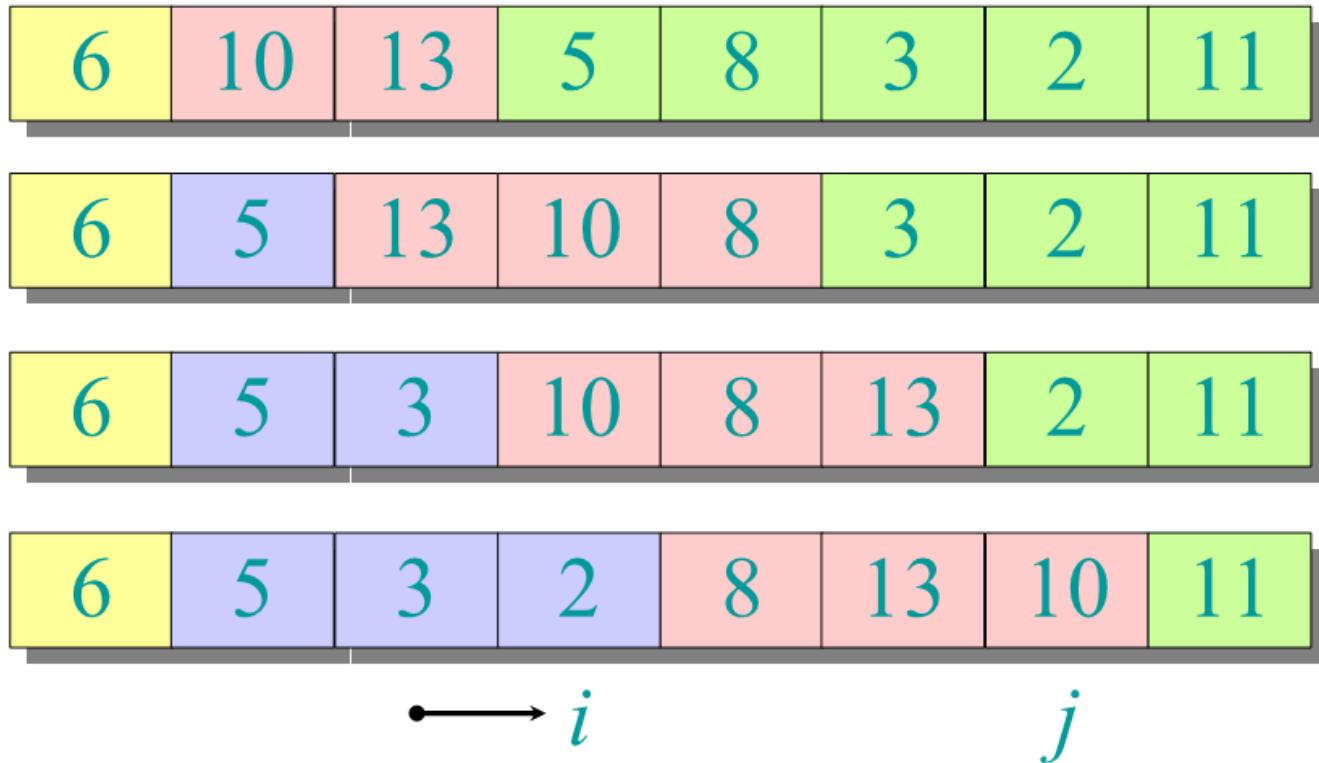
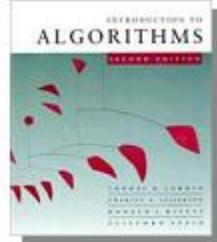
# Çabuk sıralama (Quick Sort)

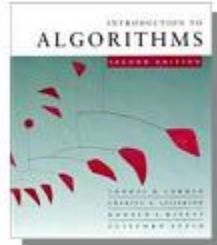
## Bölüntüleme örneği



# Çabuk sıralama (Quick Sort)

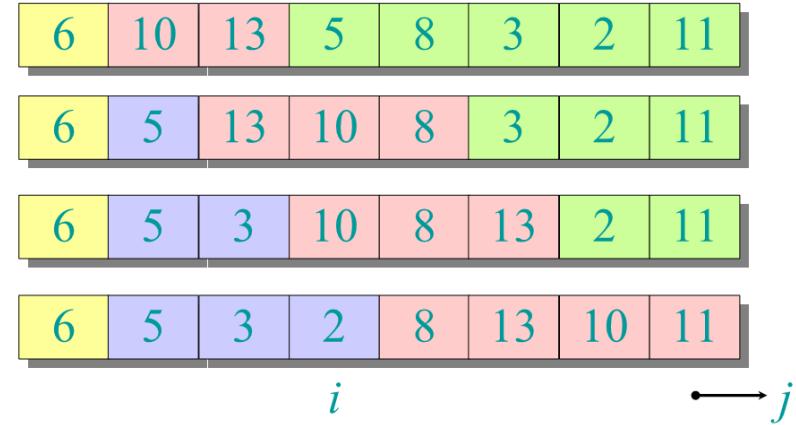
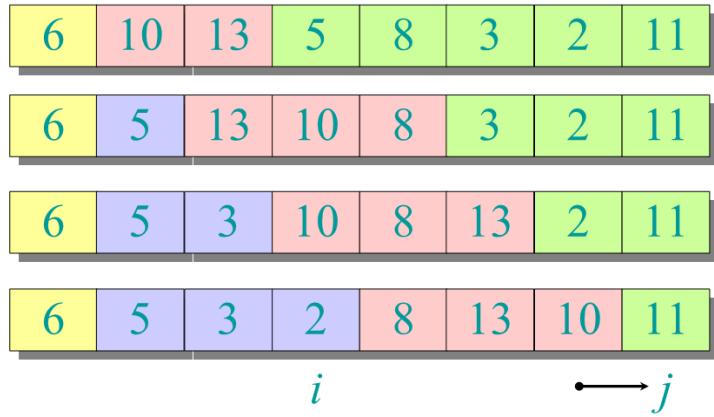
## Bölüntüleme örneği

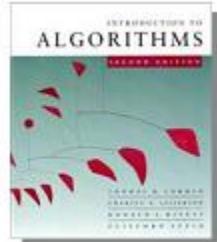




# Çabuk sıralama (Quick Sort)

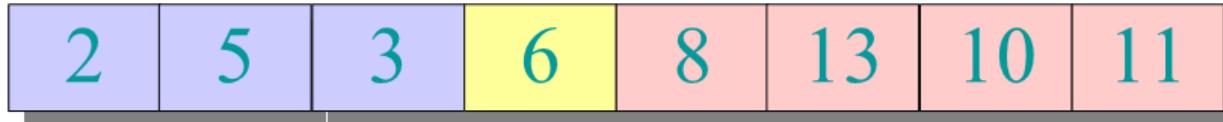
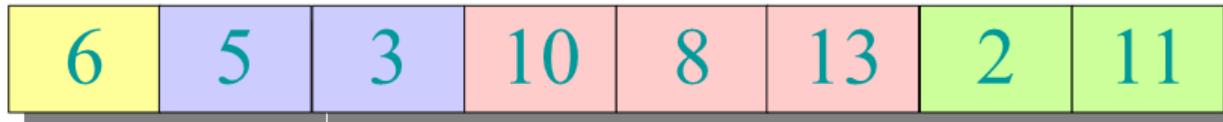
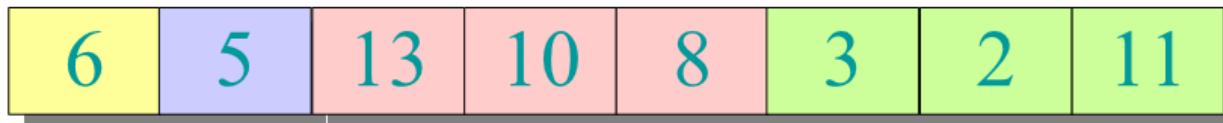
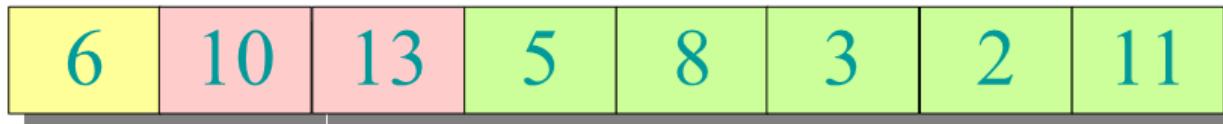
## Bölbüntüleme örneği





# Çabuk sıralama (Quick Sort)

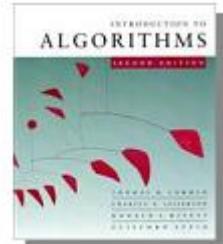
## Bölbüntüleme örneği



*i*

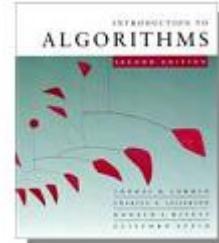
## Çabuk sıralamanın çözümlemesi

- Bütün girişlerin bir birinden farklı olduğu kabul edilirse çalışma zamanı parçaların dağılımına bağlıdır.
  - Pratikte, tekrarlayan girdi elemanları varsa, daha iyi algoritmalar vardır.
  - $n$  elemanı olan bir dizilimde
  - $T(n)$ , en kötü koşma süresi olsun.



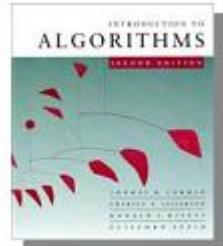
# Çabuk sıralamanın en kötü durumu (worst-case)

- Girdiler sıralı ya da ters sıralı. (Ancak sıralı girişler insert sort için en iyi durum olur)
- En küçük yada en büyük elemanların etrafında bölüntüleme.
- Bölüntünün bir yanında hiç eleman yok veya parçalardan biri sadece bir elemana sahip

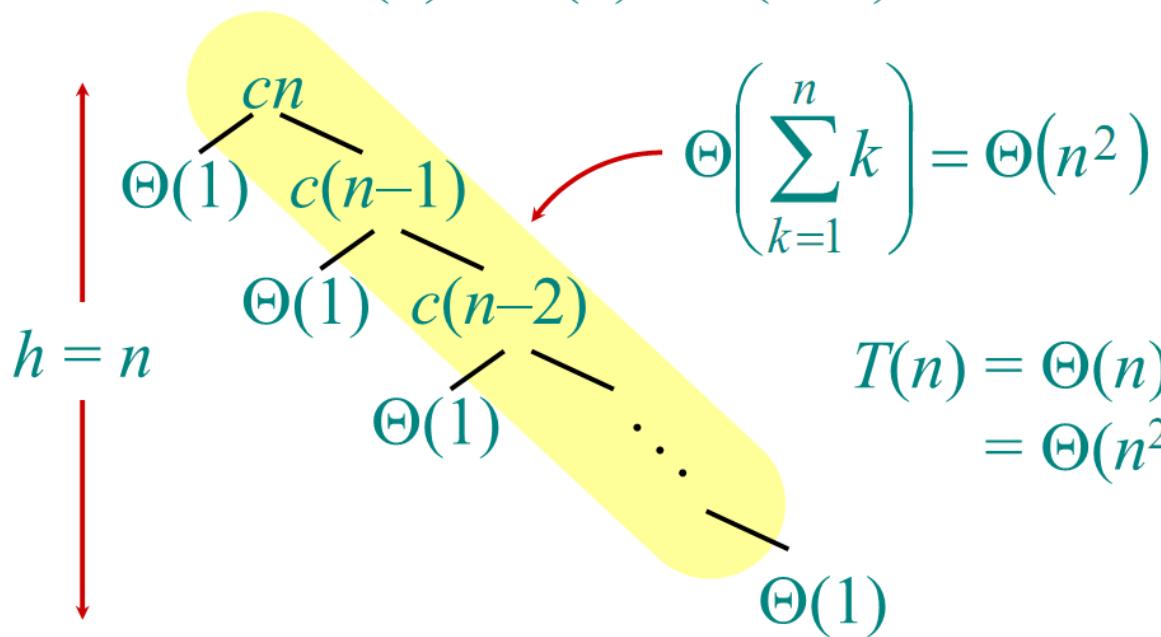


$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ &= \Theta(1) + T(n-1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \quad (\text{aritmetik seri}) \end{aligned}$$

# Çabuk sıralamanın En kötü durum özyineleme ağacı



$$T(n) = T(0) + T(n-1) + cn$$



# Rastgele çabuk sıralama çözümlemesi (analizi)

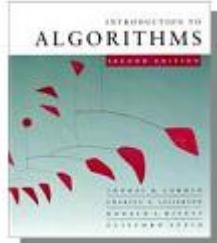
- En kötü durum:

- $T(\theta)=1$
- $T(n) = \max_{0 \leq k \leq n-1} (T(k) + T(n - k) + \theta(n))$

- Çözüm:

- $T(n) \leq cn^2$  olduğu kabul edilirse,
- $T(n) = \max_{1 \leq k \leq n-1} (ck + c(n - k)^2 + \theta(n))$
- $T(n) = c \max_{1 \leq k \leq n-1} (k + (n - k)^2 + \theta(n))$
- $T(n) = c(1 + (n - 1)^2 + \theta(n)), T(n) \leq cn^2 - 2c(n - 1) + \theta(n)$
- $T(n) \leq cn^2$  olur.

# Çabuk sıralamanın En iyi durum (Best Case) çözümlemesi ( Yalnızca sezgi gelişimi amaçlı!)



Eğer şanslıysak, BÖLÜNTÜ dizilimi eşit böler:

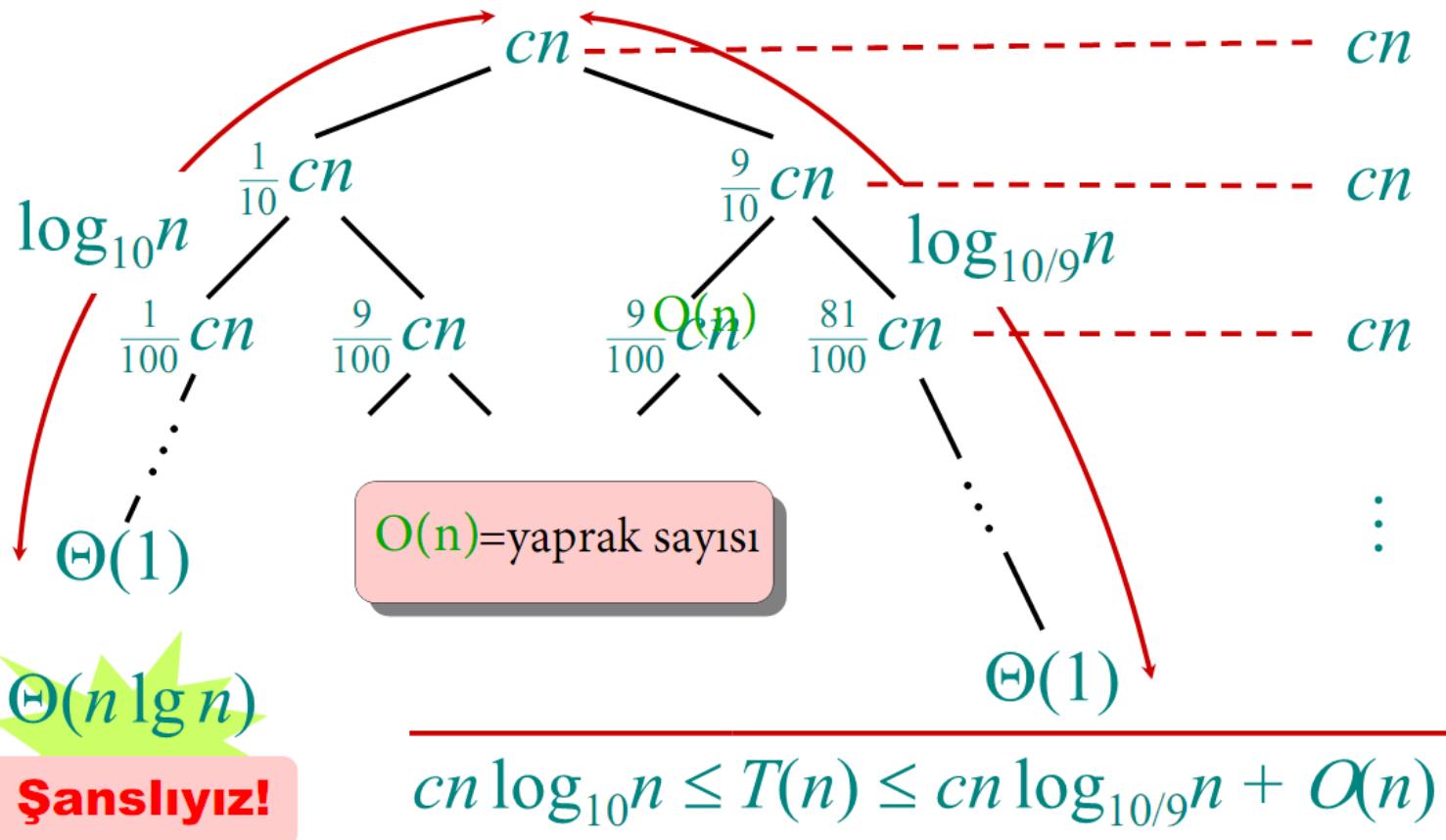
$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \quad (\text{birleştirme sıralamasındaki gibi}) \end{aligned}$$

Ya bölünme her zaman  $\frac{1}{10} : \frac{9}{10}$  oranındaysa?

$$T(n) = T\left(\frac{1}{10}n\right) + T\left(\frac{9}{10}n\right) + \Theta(n)$$

Bu yinelemenin çözümü nedir?

# Çabuk sıralamanın “En iyiye yakın” durumun (Average Case) çözümlemesi



# Çabuk sıralamanın “En iyiye yakın” durumun (Average Case) çözümlemesi: Daha fazla sezgi

- En iyi ve en kötü durumların birleşimi: average case

Şanslı ve şanssız durumlar arasında sırayla gidip geldiğimizi varsayalım ...

$$L(n) = 2U(n/2) + \Theta(n) \quad \text{şanslı durum}$$

$$U(n) = L(n - 1) + \Theta(n) \quad \text{şanssız durum}$$

Çözelim:

$$\begin{aligned} L(n) &= 2(L(n/2 - 1) + \Theta(n/2)) + \Theta(n) \\ &= 2L(n/2 - 1) + \Theta(n) \\ &= \Theta(n \lg n) \end{aligned}$$

Şanslı!

Genellikle şanslı olmayı nasıl garanti ederiz?

## Rastgele çabuk sıralama

- Genelde şanslı olmak için
  - Ortadaki elamanın yakınından ( $n/2$ ) bölme yapılır
  - Rastgele seçilen bir elamana göre bölme yapılır (Pratik daha iyi çalışır.)
- **FIKİR:** Rastgele bir eleman çevresinde bölüntü yap.
  - Çalışma zamanı girişin sırasından bağımsızdır.
  - Girişteki dağılım konusunda herhangi bir varsayıma gerek yoktur.
  - Hiçbir girdi en kötü durum davranışına neden olmaz.
  - En kötü durum yalnızca rasgele sayı üreticinin çıkışına bağlıdır.

## Rastgele çabuk sıralama (Randomized Quicksort)

- Bütün elemanların farklı olduğu kabul edilir
- Rastgele seçilen elemanın yakınından bölünür
- Bütün bölme ( $1:n-1$ ,  $2:2-2,\dots,n-1:1$ ) durumları  $1/n$  oranında eşit olasılığa sahiptir.
- Rastgele seçilen algoritmanın average-case durumunu iyileştirir.

# Rastgele çabuk sıralama (Randomized Quicksort)

**Randomized-Partition (A, p, r)**

```
01 i←Random (p, r)
02 exchange A[r] ↔A[i]
03 return Partition (A, p, r)
```

**Randomized-Quicksort (A, p, r)**

```
01 if p<r then
02 q←Randomized-Partition (A, p, r)
03 Randomized-Quicksort (A, p, q)
04 Randomized-Quicksort (A, q+1, r)
```

## Rastgele çabuk sıralama çözümlemesi (analizi)

$n$  boyutlu ve sayıların bağımsız varsayıldığı bir girdinin, rastgele çabuk çözümlemesi için  $T(n) =$  koşma süresinin rastgele değişkeni olsun.

$k = 0, 1, \dots, n-1$ , için **indicator random variable (göstergesel rastgele değişken)**'i tanımlayın

$$X_k = \begin{cases} 1 & \text{eğer } BÖLÜNTÜ \text{ bir } k : n-k-1 \text{ bölünme yaratıyorsa,} \\ 0 & \text{diğer durumlarda.} \end{cases}$$

$E[X_k] = \Pr\{X_k = 1\} = 1/n$ , elemanların farklı olduğu varsayılrsa, her bölünme işleminin olasılığı aynıdır.

## Rastgele çabuk sıralama çözümlemesi

$$\begin{aligned} T(n) &= \begin{cases} T(0) + T(n-1) + \Theta(n) & \text{eğer } 0 : n-1 \text{ bölünme,} \\ T(1) + T(n-2) + \Theta(n) & \text{eğer } 1 : n-2 \text{ bölünme,} \\ \vdots \\ T(n-1) + T(0) + \Theta(n) & \text{eğer } n-1 : 0 \text{ bölünme varsa,} \end{cases} \\ &= \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n)) \end{aligned}$$

# Rastgele çabuk sıralama Beklenenin hesaplanması

$$E[T(n)] = E \left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n)) \right]$$

Bekleneni her iki tarafta alın.

$$E[T(n)] = E \left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n)) \right]$$

$$\rightarrow = \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))]$$

Beklenenin doğrusallığı.

## Rastgele çabuk sıralama Beklenenin hesaplanması

$$\begin{aligned} E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n))\right] \\ &= \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))] \\ \rightarrow &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \end{aligned}$$

$X_k$  'nın diğer değişken seçeneklerinden bağımsızlığı.

## Rastgele çabuk sıralama Beklenenin hesaplanması

$$\begin{aligned}
 E[T(n)] &= E\left[ \sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n)) \right] \\
 &= \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))] \\
 &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \\
 \rightarrow &= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)
 \end{aligned}$$

Beklenenin doğrusallığı;  $E[X_k] = 1/n$ .

# Rastgele çabuk sıralama Beklenenin hesaplanması

$$\begin{aligned}
 E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k (T(k) + T(n-k-1) + \Theta(n))\right] \\
 &= \sum_{k=0}^{n-1} E[X_k (T(k) + T(n-k-1) + \Theta(n))] \\
 &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(k) + T(n-k-1) + \Theta(n)] \\
 &= \frac{1}{n} \sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n} \sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \\
 \Rightarrow &= \frac{2}{n} \sum_{k=1}^{n-1} E[T(k)] + \Theta(n)
 \end{aligned}$$

Toplamlarda  
benzer terimler var.

# Rastgele çabuk sıralama

## Beklenenin hesaplanması

### Karmaşık yineleme

---

$$E[T(n)] = \frac{2}{n} \sum_{k=2}^{n-1} E[T(k)] + \Theta(n)$$

(  $k = 0, 1$  terimleri  $\Theta(n)$  içine yedirilebilir.)

**Kanıtla:**  $E[T(n)] \leq an \lg n$  ( $a > 0$  sabiti için)

- $a$ 'yı öyle büyük seçin ki, yeterince küçük  $n \geq 2$  için  $an \lg n$ ,  $E[T(n)]$  'ye göre büyük olsun.

**Kullan:**  $\sum_{k=2}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2$  (egzersiz).

# Rastgele çabuk sıralama Yerine koyma metodu

$$E[T(n)] \leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n)$$

Tümevarım hipotezini yerine koyun.

$$\begin{aligned} E[T(n)] &\leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\ &\leq \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n) \end{aligned}$$

Bilineni kullanın.

## Rastgele çabuk sıralama Yerine koyma metodu

$$\begin{aligned} E[T(n)] &\leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\ &\leq \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n) \\ &= an \lg n - \left( \frac{an}{4} - \Theta(n) \right) \end{aligned}$$

İstenen (*desired*) – kalan (*residual*) olarak ifade edin.

## Rastgele çabuk sıralama Yerine koyma metodu

$$\begin{aligned}
 E[T(n)] &\leq \frac{2}{n} \sum_{k=2}^{n-1} ak \lg k + \Theta(n) \\
 &= \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + \Theta(n) \\
 &= an \lg n - \left( \frac{an}{4} - \Theta(n) \right) \\
 \rightarrow &\quad \leq an \lg n,
 \end{aligned}$$

eğer  $a$  yeterince büyük seçilir ve  $an/4 - \Theta(n)$ 'e göre büyükolursa.

## Daha sıkı bir üst sınır

$$\begin{aligned}
 \sum_{k=1}^{n-1} k \lg k &= \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg k \\
 &\leq \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \sum_{k=\lceil n/2 \rceil}^{n-1} k \lg n \\
 &= \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k
 \end{aligned}$$

$$\sum_{i=1}^n \lg i \approx n \lg n$$

Daha sıkı bir sınır için toplamı böl

İkinci terimdeki  $\lg k$ ,  $\lg n$  ile sınırlanır.

$\lg n$  i toplamın dışına taşıyın

# Daha sıkı bir üst sınır

$$\sum_{k=1}^{n-1} k \lg k \leq \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

**Şimdiye kadarki toplamın sınırı**

$$\leq \sum_{k=1}^{\lceil n/2 \rceil - 1} k \lg(n/2) + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

**İlk terimdeki  $\lg k$ ,  $\lg n/2$  ile sınırlanır**

$$= \sum_{k=1}^{\lceil n/2 \rceil - 1} k(\lg n - 1) + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

**$\lg n/2 = \lg n - 1$  ile sınırlıdır  
ve**

$$= (\lg n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k$$

**( $\lg n - 1$ ) i toplamın dışına taşıyın**

# Daha sıkı bir üst sınır

$$\begin{aligned}
 \sum_{k=1}^{n-1} k \lg k &\leq (\lg n - 1) \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k \\
 &= \lg n \sum_{k=1}^{\lceil n/2 \rceil - 1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k + \lg n \sum_{k=\lceil n/2 \rceil}^{n-1} k && (\lg n - 1)'i \text{ dağıtn} \\
 &= \lg n \sum_{k=1}^{n-1} k - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \\
 &= \lg n \left( \frac{(n-1)(n)}{2} \right) - \sum_{k=1}^{\lceil n/2 \rceil - 1} k && \text{Guassian serisi}
 \end{aligned}$$

# Daha sıkı bir üst sınır

$$\begin{aligned}\sum_{k=1}^{n-1} k \lg k &\leq \left( \frac{(n-1)(n)}{2} \right) \lg n - \sum_{k=1}^{\lceil n/2 \rceil - 1} k \\&\leq \frac{1}{2} [n(n-1)] \lg n - \sum_{k=1}^{n/2-1} k \\&\leq \frac{1}{2} [n(n-1)] \lg n - \frac{1}{2} \left( \frac{n}{2} \right) \left( \frac{n}{2} - 1 \right) \quad \text{X Guassian series} \\&\leq \frac{1}{2} \left( n^2 \lg n - n \lg n \right) - \frac{1}{8} n^2 + \frac{n}{4}\end{aligned}$$

## Daha sıkı bir üst sınır

$$\begin{aligned}\sum_{k=1}^{n-1} k \lg k &\leq \frac{1}{2} (n^2 \lg n - n \lg n) - \frac{1}{8} n^2 + \frac{n}{4} \\ &\leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \text{ when } n \geq 2\end{aligned}$$

olur!!!

# Orjinal Partition Algoritması

## 7-1 Hoare partition correctness

The version of PARTITION given in this chapter is not the original partitioning algorithm. Here is the original partition algorithm, which is due to C. A. R. Hoare:

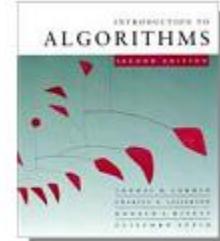
HOARE-PARTITION( $A, p, r$ )

```
1 $x = A[p]$
2 $i = p - 1$
3 $j = r + 1$
4 while TRUE
5 repeat
6 $j = j - 1$
7 until $A[j] \leq x$
8 repeat
9 $i = i + 1$
10 until $A[i] \geq x$
11 if $i < j$
12 exchange $A[i]$ with $A[j]$
13 else return j
```

# Çabuk sıralama (Quick Sort)

## Bölüntüleme örneği-2

### pivot son elaman

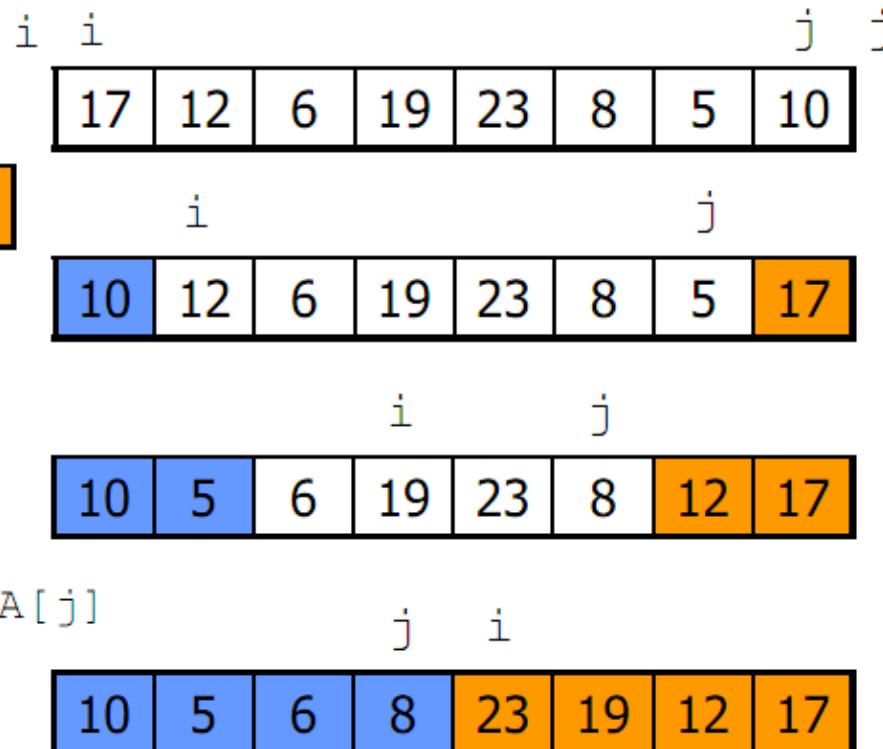


**Partition**(A, p, r)

```

01 x←A[r]
02 i←p-1
03 j←r+1
04 while TRUE
05 repeat j←j-1
06 until A[j] ≤x
07 repeat i←i+1
08 until A[i] ≥x
09 if i < j
10 then exchange A[i]↔A[j]
11 else return j

```



## Pratikte çabuk sıralama

- Çabuk sıralama önemli bir genel maksatlı sıralama algoritmasıdır.
- Çabuk sıralama tipik olarak birleştirme (Merge Sort) sıralamasından iki kat daha hızlıdır.
- Çabuk sıralama önbellekleme ve sanal bellek uygulamalarında oldukça uyumludur.

## Quick sort ile Heapsort karşılaştırma

- Analiz sonuçlarına göre heap sort 'un en kötü durumu, hızlı sıralamanın ortalama durumundan kötüdür ancak hızlı sıralamanın en kötü durumu çok daha kötüdür.
  - Heapsort'un ortalama durum analizi çok karmaşıktır fakat en kötü durum ile ortalama durum arasında çok az fark vardır.
  - Heapsort genelde Quicksort tan 2 kat daha fazla zaman alır. Ortalama olarak maliyeti pahalı olmasına rağmen  $O(n^2)$  olasılığını önler.
  - Quicksort random bölütleme yapılrsa en kötü durumda  $n \log n$  olur

# Ek: Sıralama Algoritmaları Analiz

[http://tr.wikipedia.org/wiki/Sıralama\\_algoritması](http://tr.wikipedia.org/wiki/Sıralama_algoritması)

[http://en.wikipedia.org/wiki/Sorting\\_algorithm](http://en.wikipedia.org/wiki/Sorting_algorithm)

| Adı                           | Ortalama      | En Kötü         | Bellek      | Kararlı mı? | Yöntem        |
|-------------------------------|---------------|-----------------|-------------|-------------|---------------|
| Kabarcık Sıralaması           | —             | $O(n^2)$        | $O(1)$      | Evet        | Değiştirme    |
| Kokteyl Sıralaması            | —             | $O(n^2)$        | $O(1)$      | Evet        | Değiştirme    |
| Tarak Sıralaması              | $O(n \log n)$ | $O(n \log n)$   | $O(1)$      | Hayır       | Değiştirme    |
| Cüce Sıralaması               | —             | $O(n^2)$        | $O(1)$      | Evet        | Değiştirme    |
| Seçmeli Sıralama              | $O(n^2)$      | $O(n^2)$        | $O(1)$      | Hayır       | Seçme         |
| Eklemeli Sıralama             | $O(n + d)$    | $O(n^2)$        | $O(1)$      | Evet        | Ekleme        |
| Kabuk Sıralaması              | —             | $O(n \log^2 n)$ | $O(1)$      | Hayır       | Ekleme        |
| Ağaç Sıralaması               | $O(n \log n)$ | $O(n \log n)$   | $O(n)$      | Evet        | Ekleme        |
| Kütüphane Sıralaması          | $O(n \log n)$ | $O(n^2)$        | $O(n)$      | Evet        | Ekleme        |
| Birleşirmeli Sıralama         | $O(n \log n)$ | $O(n \log n)$   | $O(n)$      | Evet        | Birleştirme   |
| Yerinde Birleşirmeli Sıralama | $O(n \log n)$ | $O(n \log n)$   | $O(1)$      | Evet        | Birleştirme   |
| Yığın Sıralaması              | $O(n \log n)$ | $O(n \log n)$   | $O(1)$      | Hayır       | Seçme         |
| Rahat Sıralama                | —             | $O(n \log n)$   | $O(1)$      | Hayır       | Seçme         |
| Hızlı Sıralama                | $O(n \log n)$ | $O(n^2)$        | $O(\log n)$ | Hayır       | Bölümlendirme |
| İçgözlemle Sıralama           | $O(n \log n)$ | $O(n \log n)$   | $O(\log n)$ | Hayır       | Melez         |
| Sabır Sıralaması              | —             | $O(n^2)$        | $O(n)$      | Hayır       | Ekleme        |
| İplik Sıralaması              | $O(n \log n)$ | $O(n^2)$        | $O(n)$      | Evet        | Seçme         |

# **Alt Sınırları Sıralama Doğrusal-Zaman (linear time) Sıralaması**

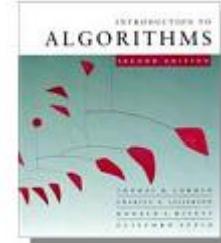
**Alt Sınırları Sıralama**

- Karar ağaçları

**Doğrusal-Zaman Sıralaması**

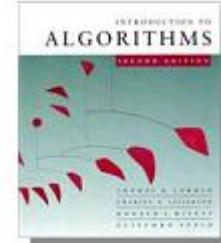
- Sayma sıralaması
- Taban sıralaması
- Kova sıralaması

# Ne kadar hızlı sıralayabiliriz?

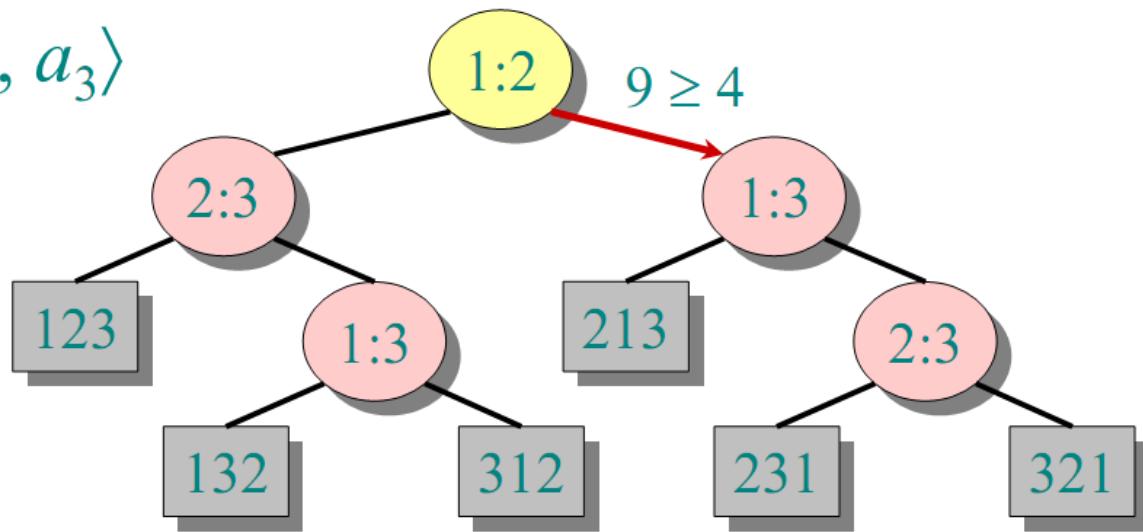


- Şu ana kadar gördüğümüz tüm sıralama algoritmaları **karşılaştırma sıralamalarıydı**. Elemanların bağıl düzenlerini saptamakta yalnız karşılaştırma kullanırlar.
- Örneğin, araya yerleştirme, birleştirme sıralamaları, çabuk sıralama, yiğin sıralaması.
- Karşılaştırma sıralamalarında gördüğümüz en iyi en-kötü-durum koşma süresi  **$O(nlgn)$**  idi.
- **$O(nlgn)$  elde edebileceğimizin en iyisi mi?**
- **Karar ağaçları** bu sorunun yanıtına yardımcı olur.

# Karar-ağacı örneği



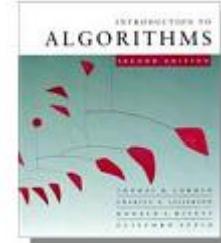
Sırala  $\langle a_1, a_2, a_3 \rangle$   
 $= \langle 9, 4, 6 \rangle$ :



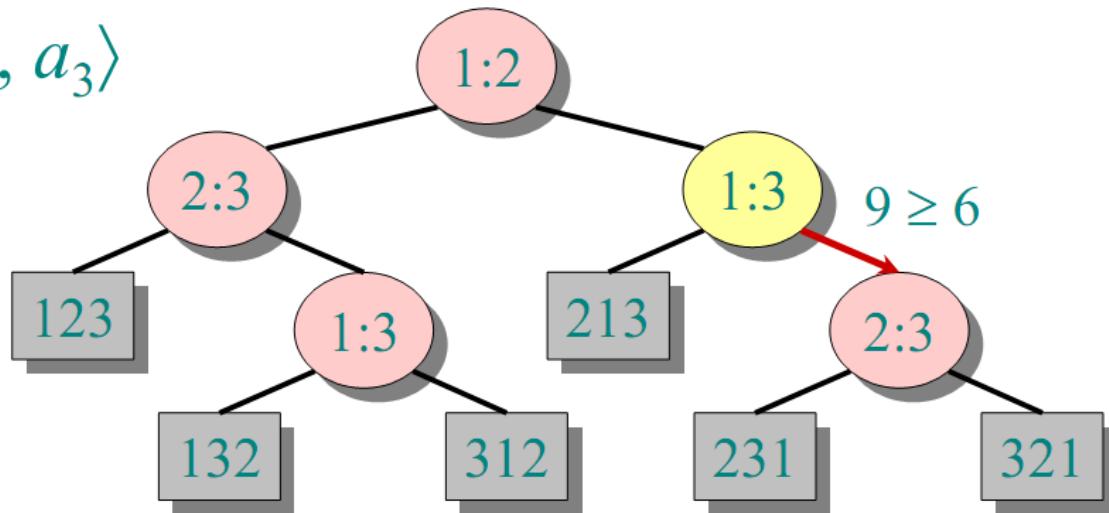
Her iç boğumun etiketlenmesi  $i:j$ ;  $i, j \in \{1, 2, \dots, n\}$  için.

- Sol alt-ağacı  $a_i \leq a_j$  ise, ardarda karşılaştırmaları gösterir.
- Sağ alt-ağacı  $a_i \geq a_j$  ise, ardarda karşılaştırmaları gösterir.

# Karar-ağacı örneği



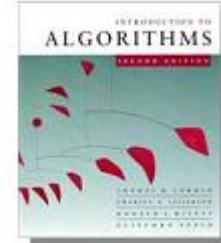
Sırala  $\langle a_1, a_2, a_3 \rangle$   
 $= \langle 9, 4, 6 \rangle$ :



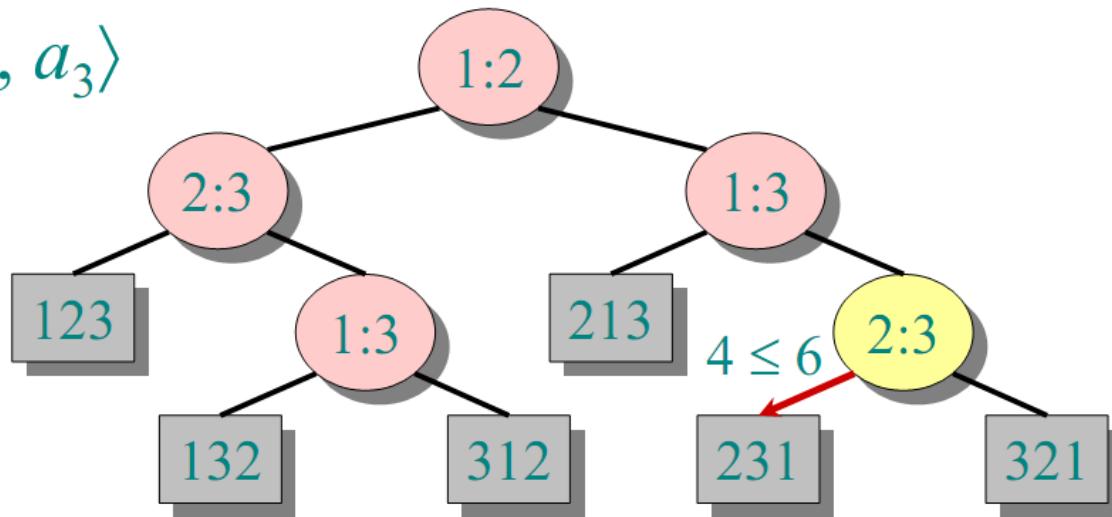
Her iç boğumun etiketlenmesi  $i:j$  ;  $i, j \in \{1, 2, \dots, n\}$  için:

- Sol alt-ağaç  $a_i \leq a_j$  ise, ardarda karşılaştırmaları gösterir.
- Sağ alt-ağaç  $a_i \geq a_j$  ise, ardarda karşılaştırmaları gösterir.

# Karar-ağacı örneği



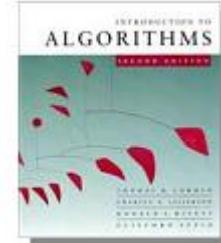
Sırala  $\langle a_1, a_2, a_3 \rangle$   
=  $\langle 9, 4, 6 \rangle$ :



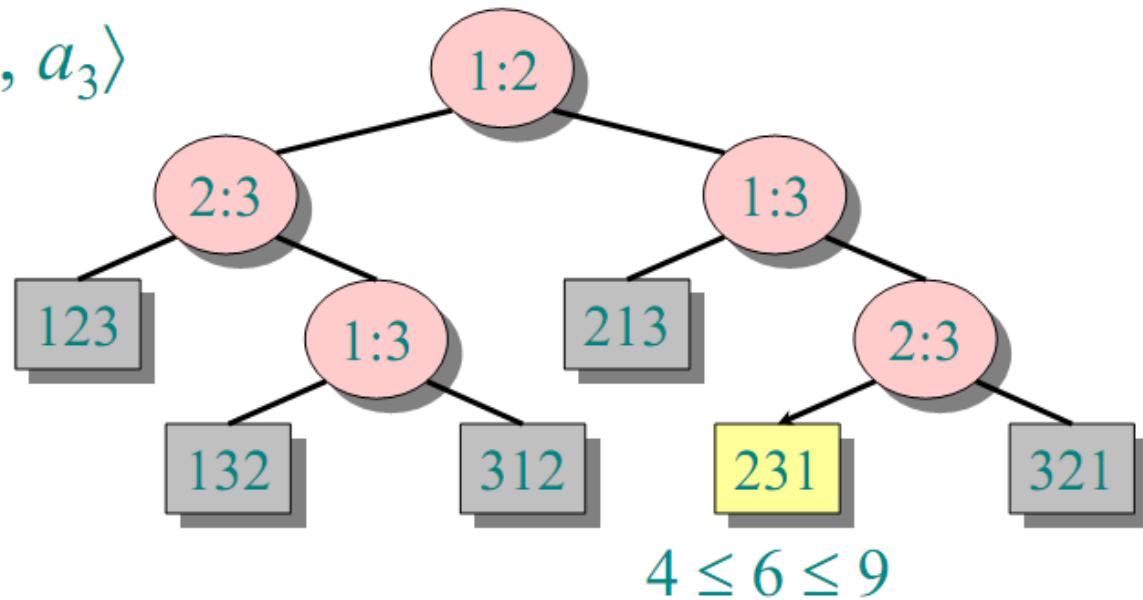
Her iç böğümün etiketlenmesi  $i:j$ ;  $i, j \in \{1, 2, \dots, n\}$  için.

- Sol alt-ağaç  $a_i \leq a_j$  ise, ardarda karşılaştırmaları gösterir.
  - Sağ alt-ağaç  $a_i \geq a_j$  ise, ardarda karşılaştırmaları gösterir.

# Karar-ağacı örneği

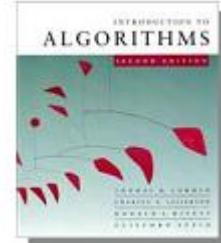


Sırala  $\langle a_1, a_2, a_3 \rangle$   
 $= \langle 9, 4, 6 \rangle$ :



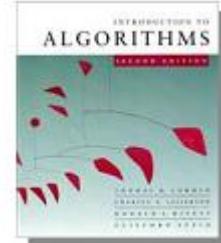
Her yaprakta  $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$  permütasyonu vardır bu  $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$  sıralamasının tamamlanmış olduğunu gösterir.

# Karar-ağacı modeli



- Bir karar ağacı her karşılaştırma sıralaması uygulanmasını modelleyebilir:
  - Her **n** giriş boyutu için bir ağaç.
  - Algoritmayı iki elemanı karşılaştırdığında bölündüyormuş gibi görün.
  - Ağaç tüm olası komut izlerindeki karşılaştırmalar içerir.
  - Algoritmanın çalışma zamanı = takip edilen yolun uzunluğu.
  - En kötü-durum çalışma zamanı = ağacın boyu.

# Karar-ağacı sıralamasında alt sınır



**Teorem.**  $n$  elemanı sıralayabilen bir karar-ağacının yüksekliği (boyu)  $\Omega(n \lg n)$  olmalıdır.

*Kanıtlama.* Ağacın  $\geq n!$  yaprağı olmalıdır, çünkü ortada  $n!$  olası permütasyon vardır. Boyu  $h$  olan bir ikili ağacın  $\leq 2^h$  yaprağı olur. Böylece,  $n! \leq 2^h$ .

$$\therefore h \geq \lg(n!)$$

( $\lg$  monoton artışlı)

$$\geq \lg ((n/e)^n)$$

(Stirling'in formülü)

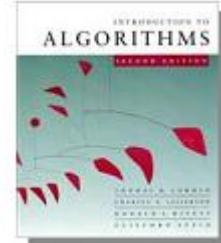
$$= n \lg n - n \lg e$$

$$n! > \left(\frac{n}{e}\right)^n$$

$$= \Omega(n \lg n).$$



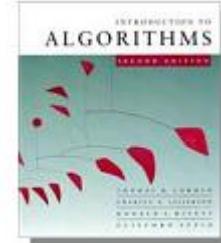
# Karar-ağacı sıralamasında alt sınır



**Doğal sonuç.** Yığın sıralaması ve birleştirme sıralaması asimptotik olarak en iyi karşılaştırma sıralaması algoritmalarıdır. □

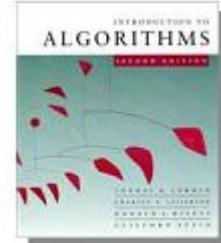
- Randomize Quick Sort da asimtotik olarak en iyi karşılaştırma sıralama algoritması olduğu söylenebilir.

# Doğrusal zamanda sıralama



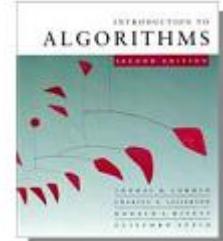
- **Sayma sıralaması (Counting Sort):** Elemanlar arası karşılaştırma yok.
- **Giriş:**  $A[1 \dots n]$ , burada  $A[j] \in \{1, 2, \dots, k\}$ .
- **$k$ , küçük ise** iyi bir algoritma olur,  **$k$ , büyük ise** çok kötü bir algoritma olur ( $n \log n$  daha kötü)
- **Cıkış:**  $B[1 \dots n]$ , sıralı.
- **Yedek depolama:**  $C[1 \dots k]$ .

# Sayma sıralaması



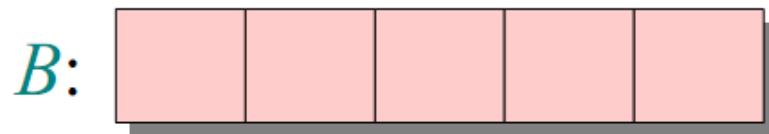
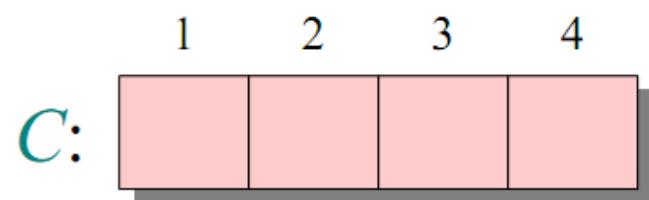
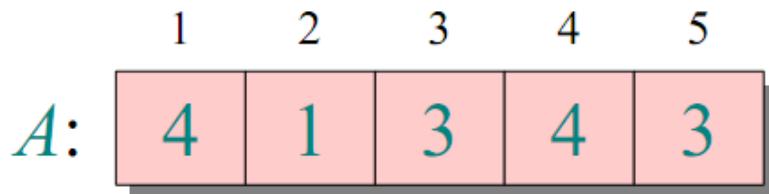
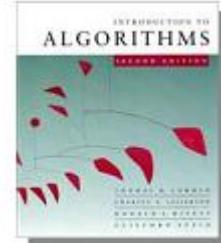
- $n$  adet girişin tamsayı olduğu kabul edilir.
- Girişlerin **0** ile **k** arasında olduğu kabul edilir.
- Temel olarak bir **x** elemanı için kendisinden küçük elemanların sayısını bulmayı amaçlar. Örneğin  $x$  elemanından küçük 17 eleman varsa  $x$  elemanın doğru yeri 18 olur.
- Girilen dizi boyutunda bir ek dizİYE ihtiyaç duyar
- Elemanların aralığı kadar elemana sahip ikinci bir ek dizİYE ihtiyaç duyar.

# Sayma sıralaması



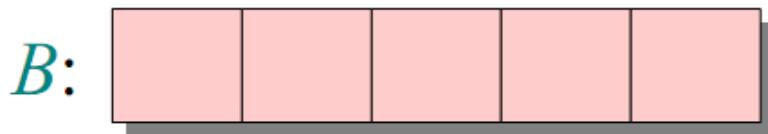
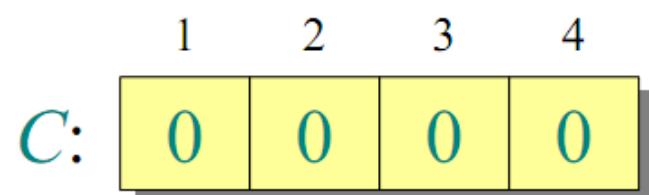
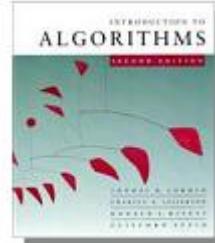
```
for i ← 1 to k
 do C[i] ← 0
for j ← 1 to n
 do C[A[j]] ← C[A[j]] + 1 ▷ C[i] = |{key = i}|
for i ← 2 to k
 do C[i] ← C[i] + C[i−1] ▷ C[i] = |{key ≤ i}|
for j ← n down to 1 (down to 1: 1'e inene kadar)
 do B[C[A[j]]] ← A[j]
 C[A[j]] ← C[A[j]] − 1
```

# Sayma sıralaması



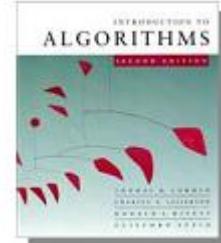
- Dizi girişi 1 ile 4 arasındadır. O zaman  $k=4$  olur.

# Döngü 1



```
for $i \leftarrow 1$ to k
 do $C[i] \leftarrow 0$
```

# Döngü 2



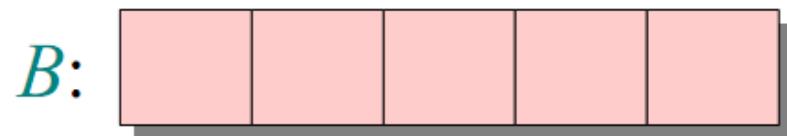
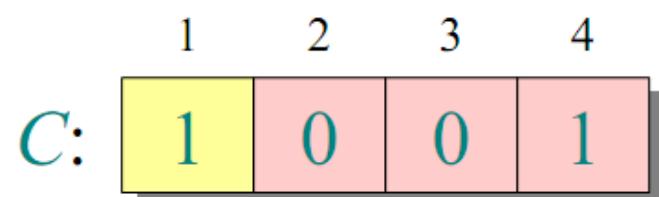
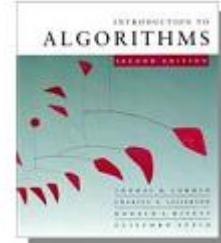
| 1    | 2 | 3 | 4 | 5 |
|------|---|---|---|---|
| A: 4 | 1 | 3 | 4 | 3 |

| 1    | 2 | 3 | 4 |
|------|---|---|---|
| C: 0 | 0 | 0 | 1 |

|  |  |  |  |  |
|--|--|--|--|--|
|  |  |  |  |  |
|--|--|--|--|--|

**for**  $j \leftarrow 1$  **to**  $n$   
**do**  $C[A[j]] \leftarrow C[A[j]] + 1$   $\triangleright C[i] = |\{ \text{key} = i \}|$

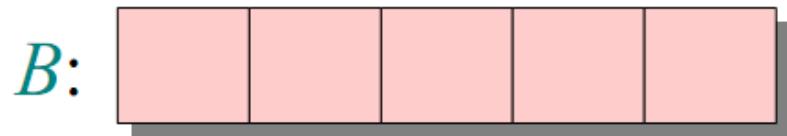
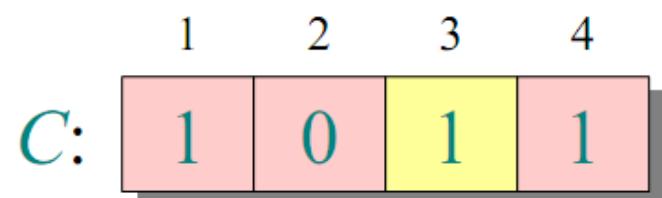
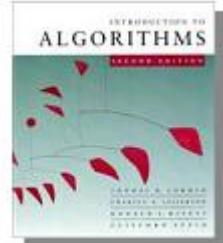
# Döngü 2



**for**  $j \leftarrow 1$  **to**  $n$

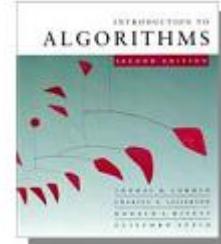
**do**  $C[A[j]] \leftarrow C[A[j]] + 1$   $\triangleright C[i] = |\{ \text{key} = i \}|$

# Döngü 2



**for**  $j \leftarrow 1$  **to**  $n$   
**do**  $C[A[j]] \leftarrow C[A[j]] + 1$   $\triangleright C[i] = |\{ \text{key} = i \}|$

# Döngü 2

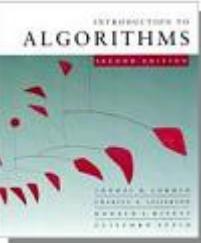


| 1    | 2 | 3 | 4 | 5 |
|------|---|---|---|---|
| A: 4 | 1 | 3 | 4 | 3 |

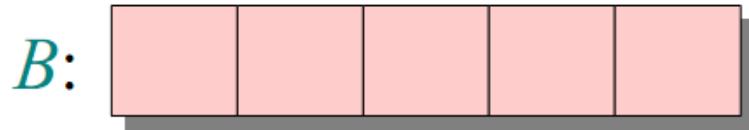
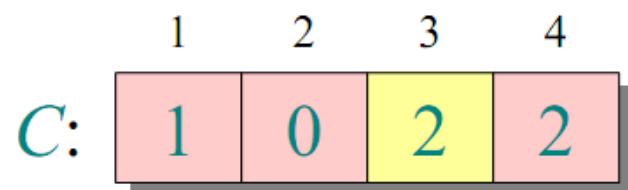
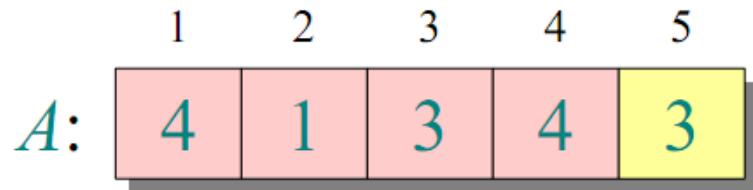
| 1    | 2 | 3 | 4 |
|------|---|---|---|
| C: 1 | 0 | 1 | 2 |

|    |  |  |  |  |  |
|----|--|--|--|--|--|
| B: |  |  |  |  |  |
|----|--|--|--|--|--|

```
for $j \leftarrow 1$ to n
 do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{key = i\}|$
```

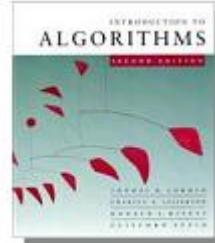


# Döngü 2



```
for $j \leftarrow 1$ to n
do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleright C[i] = |\{ \text{key} = i \}|$
```

# Döngü 3



|           | 1 | 2 | 3 | 4 | 5 |
|-----------|---|---|---|---|---|
| <i>A:</i> | 4 | 1 | 3 | 4 | 3 |

|           | 1 | 2 | 3 | 4 |
|-----------|---|---|---|---|
| <i>C:</i> | 1 | 0 | 2 | 2 |

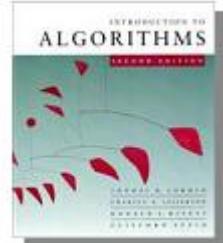
| <i>B:</i> |  |  |  |  |  |
|-----------|--|--|--|--|--|
|-----------|--|--|--|--|--|

| <i>C':</i> | 1 | 1 | 2 | 2 |
|------------|---|---|---|---|
|------------|---|---|---|---|

**for**  $i \leftarrow 2$  **to**  $k$   
**do**  $C[i] \leftarrow C[i] + C[i-1]$

▷  $C[i] = |\{\text{key} \leq i\}|$

# Döngü 3



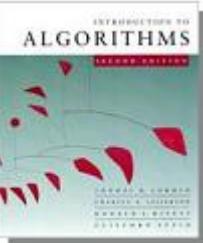
|      | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|
| $A:$ | 4 | 1 | 3 | 4 | 3 |

|      | 1 | 2 | 3 | 4 |
|------|---|---|---|---|
| $C:$ | 1 | 0 | 2 | 2 |

| $B:$ |  |  |  |  |  |
|------|--|--|--|--|--|
|      |  |  |  |  |  |

| $C':$ | 1 | 1 | 3 | 2 |
|-------|---|---|---|---|
|       |   |   |   |   |

**for**  $i \leftarrow 2$  **to**  $k$   
**do**  $C[i] \leftarrow C[i] + C[i-1]$        $\triangleright C[i] = |\{\text{key} \leq i\}|$



# Döngü 3

|           | 1 | 2 | 3 | 4 | 5 |
|-----------|---|---|---|---|---|
| <i>A:</i> | 4 | 1 | 3 | 4 | 3 |

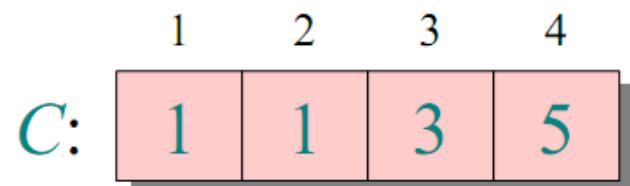
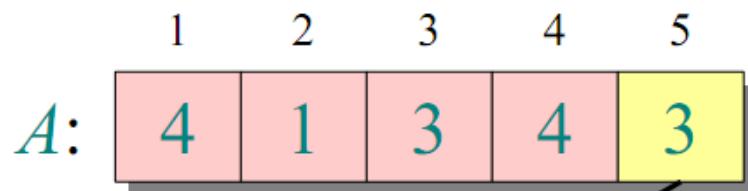
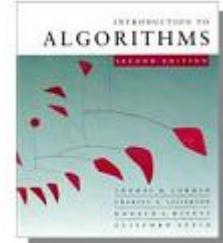
|           | 1 | 2 | 3 | 4 |
|-----------|---|---|---|---|
| <i>C:</i> | 1 | 0 | 2 | 2 |

|           |  |  |  |  |  |
|-----------|--|--|--|--|--|
| <i>B:</i> |  |  |  |  |  |
|           |  |  |  |  |  |

|            |   |   |   |   |
|------------|---|---|---|---|
| <i>C':</i> | 1 | 1 | 3 | 5 |
|            |   |   |   |   |

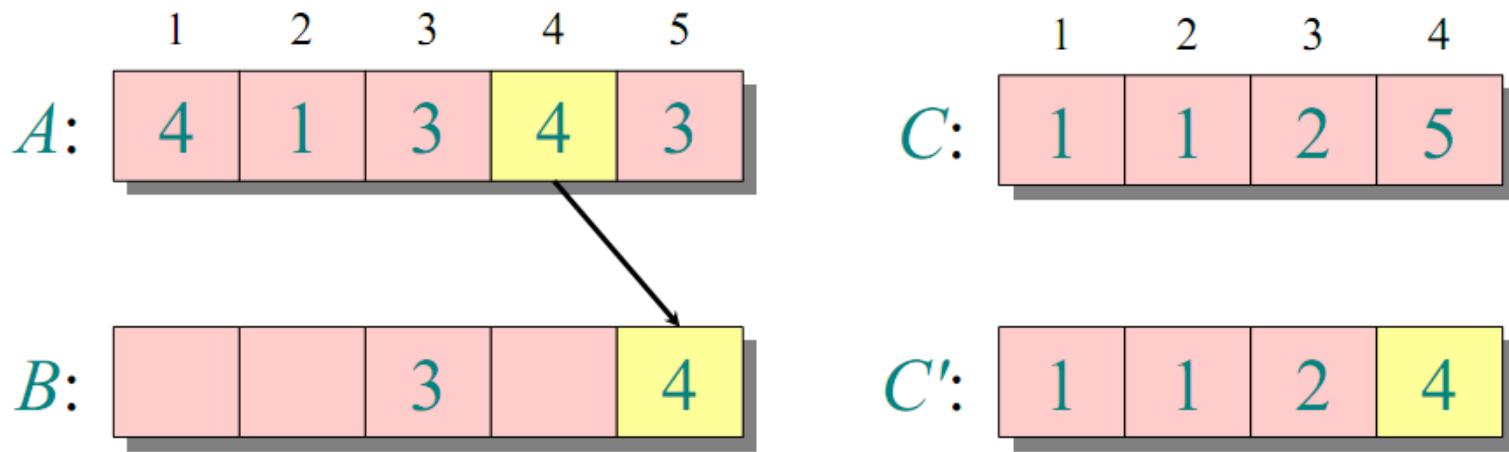
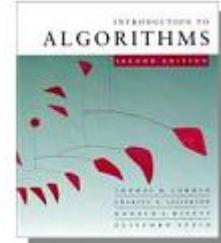
**for**  $i \leftarrow 2$  **to**  $k$   
**do**  $C[i] \leftarrow C[i] + C[i-1]$        $\triangleright C[i] = |\{\text{key} \leq i\}|$

# Döngü 4



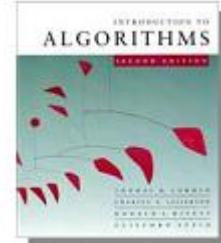
```
for $j \leftarrow n$ down to 1
 do $B[C[A[j]]] \leftarrow A[j]$
 $C[A[j]] \leftarrow C[A[j]] - 1$
```

# Döngü 4



```
for $j \leftarrow n$ down to 1
 do $B[C[A[j]]] \leftarrow A[j]$
 $C[A[j]] \leftarrow C[A[j]] - 1$
```

# Döngü 4



|      | 1 | 2 | 3 | 4 | 5 |
|------|---|---|---|---|---|
| $A:$ | 4 | 1 | 3 | 4 | 3 |

| $B:$ |  | 3 | 3 |  | 4 |
|------|--|---|---|--|---|
|      |  |   |   |  |   |

|      | 1 | 2 | 3 | 4 |
|------|---|---|---|---|
| $C:$ | 1 | 1 | 2 | 4 |

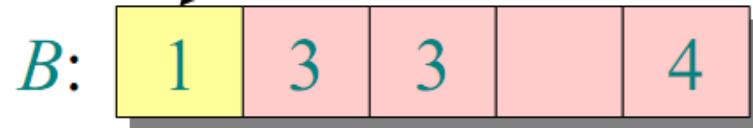
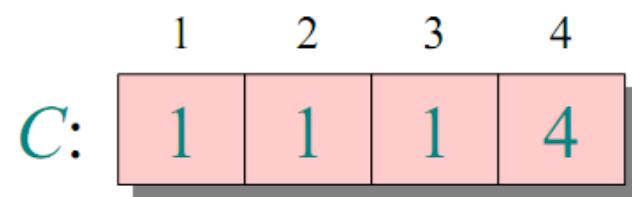
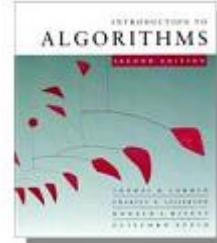
| $C':$ | 1 | 1 | 1 | 4 |
|-------|---|---|---|---|
|       |   |   |   |   |

```

for $j \leftarrow n$ down to 1
 do $B[C[A[j]]] \leftarrow A[j]$
 $C[A[j]] \leftarrow C[A[j]] - 1$

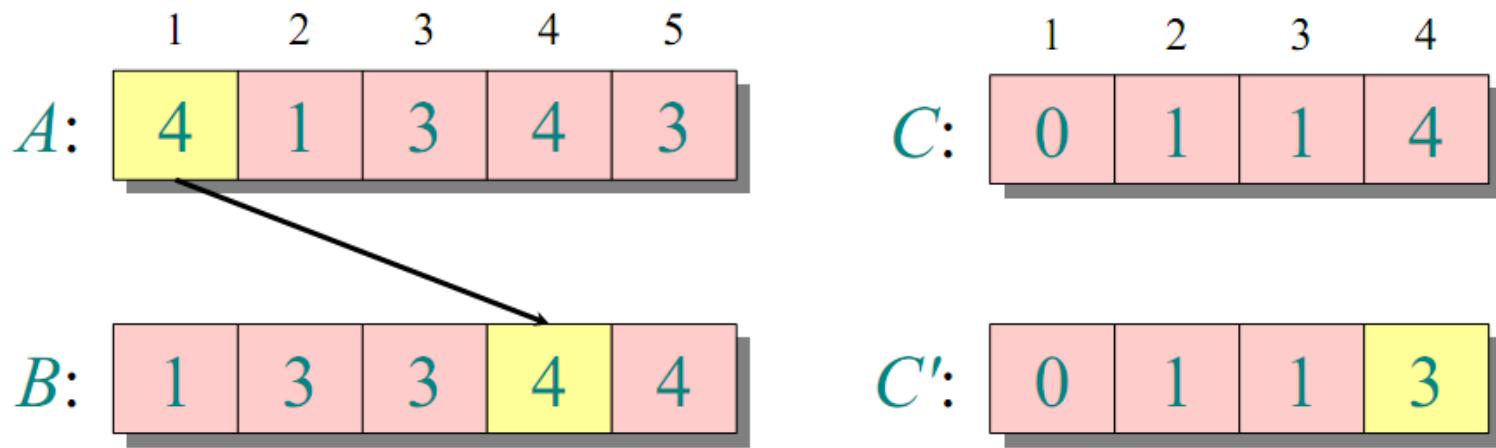
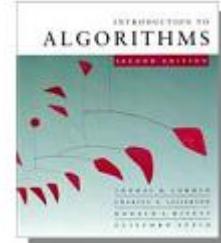
```

# Döngü 4

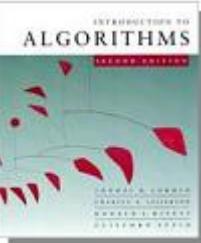


```
for $j \leftarrow n$ down to 1
do $B[C[A[j]]] \leftarrow A[j]$
 $C[A[j]] \leftarrow C[A[j]] - 1$
```

# Döngü 4



```
for $j \leftarrow n$ down to 1
 do $B[C[A[j]]] \leftarrow A[j]$
 $C[A[j]] \leftarrow C[A[j]] - 1$
```



# Çözümleme

$$\Theta(k) \quad \left\{ \begin{array}{l} \textbf{for } i \leftarrow 1 \text{ to } k \\ \quad \textbf{do } C[i] \leftarrow 0 \end{array} \right.$$

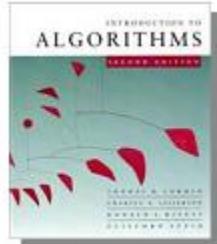
$$\Theta(n) \quad \left\{ \begin{array}{l} \textbf{for } j \leftarrow 1 \text{ to } n \\ \quad \textbf{do } C[A[j]] \leftarrow C[A[j]] + 1 \end{array} \right.$$

$$\Theta(k) \quad \left\{ \begin{array}{l} \textbf{for } i \leftarrow 2 \text{ to } k \\ \quad \textbf{do } C[i] \leftarrow C[i] + C[i-1] \end{array} \right.$$

$$\Theta(n) \quad \left\{ \begin{array}{l} \textbf{for } j \leftarrow n \text{ down to } 1 \\ \quad \textbf{do } B[C[A[j]]] \leftarrow A[j] \\ \quad \quad C[A[j]] \leftarrow C[A[j]] - 1 \end{array} \right.$$


---

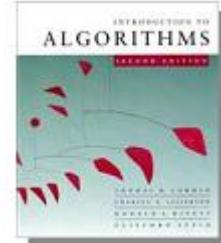

$$\Theta(n + k)$$



# Çalışma Zamanı

- $k = O(n)$  ise, sayma sıralaması  $\Theta(n)$  süresi alır. Eğer  $k=n^2$  veya  $k=2^n$  çok kötü bir algoritma olur.
- $k$  tamsayı olmalı.
- Ama sıralamalar  $\Omega(n \lg n)$  süresi alıyordu! (karar ağacı)
- Hata nerede?
- **Yanıt:**
- Karşılaştırma sıralaması  $\Omega(n \lg n)$  süre alır.
- Sayma sıralaması bir karşılaştırma sıralaması değildir.
- Aslında elemanlar arasında bir tane bile karşılaştırma yapılmaz!

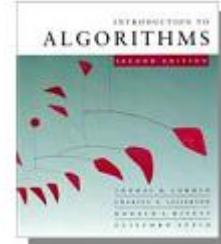
# Çalışma Zamanı



```
o using System;
o class Program
o { static void Main(string[] args)
o { Random rand = new Random();
o int[] arr = new int[8];
o for (int i = 0; i < 8; i++) { arr[i] = rand.Next(0, 10);
o Console.Write(" "+arr[i]);} Console.WriteLine();
o int[] newarr = countingSort(arr, arr.Min(), arr.Max());
o foreach(int x in arr) Console.Write(" "+x);
o }
o private static int[] countingSort(int[] arr, int min, int max) {
o int[] count = new int[max - min + 1];
o int z = 0;
o for (int i = 0; i < count.Length; i++) { count[i] = 0; }
o for (int i = 0; i < arr.Length; i++) { count[arr[i] - min]++; }

o for (int i = min; i <= max; i++)
o { while (count[i - min]-- > 0) { arr[z] = i; z++; }
o }
o return arr;
o }
o }
```

# Sayma sıralamanın artıları eksileri



## ○ Artıları:

- $n$  ve  $k$  da doğrusaldır (lineer).
- Kolay uygulanır.

## ○ Eksileri:

- Yerinde sıralama yapmaz. Ekstra depolama alanına ihtiyaç duyar.
- Sayıların küçük tam sayı olduğu varsayıılır.
- Byte ise ek dizinin boyutu en fazla  $2^8 = 256$  olur fakat sayılar int ise yani 32 bit lik sayılar ise  $2^{32} = 4.2$  milyar sayı eder oda yaklaşık 16 Gb yer tutar.

# **7. Hafta Sıra İstatistikleri, Bilinen Probleme İndirgeme**

(Devam)

Doğrusal-Zaman Sıralaması

- **Taban sıralaması**
- **Kova sıralaması**

# Quicksort Analizi: Ortalama Durum

- $(0:n-1, 1:n-2, 2:n-3, \dots, n-2:1, n-1:0)$  bölünme üretiliyor ise her bir bölünmenin  $1/n$  olasılığı vardır.
- $T(n)$ 'nin beklenen çalışma zamanı

$$\begin{aligned} T(n) &= \frac{1}{n} \sum_{k=0}^{n-1} [T(k) + T(n-1-k)] + \Theta(n) \\ &= \frac{2}{n} \sum_{k=0}^{n-1} T(k) + \Theta(n) \end{aligned}$$

$T(k)$  değerinden 2 tane var

- Çözümün  $T(n) \leq an\log n + b$  olduğu kabul edilsin

# Quicksort Analizi: Ortalama Durum

$$T(n) = \frac{2}{n} \sum_{k=0}^{n-1} T(k) + \Theta(n)$$

$$\leq \frac{2}{n} \sum_{k=0}^{n-1} (ak \lg k + b) + \Theta(n)$$

$$\leq \frac{2}{n} \left[ b + \sum_{k=1}^{n-1} (ak \lg k + b) \right] + \Theta(n)$$

$$= \frac{2}{n} \sum_{k=1}^{n-1} (ak \lg k + b) + \frac{2b}{n} + \Theta(n)$$

$$= \frac{2}{n} \sum_{k=1}^{n-1} (ak \lg k + b) + \Theta(n)$$

Yineleme ile çözüm

Tümevarım hipotezi yerleştir

k=0 durumundan genişlet

2b/n sabit olduğundan  
 $\Theta(n)$  içerisinde dahil et

## Quicksort Analizi: Ortalama Durum

$$\begin{aligned}
 T(n) &= \frac{2}{n} \sum_{k=1}^{n-1} (ak \lg k + b) + \Theta(n) && \text{Yineleme ile çözüm} \\
 &= \frac{2}{n} \sum_{k=1}^{n-1} ak \lg k + \frac{2}{n} \sum_{k=1}^{n-1} b + \Theta(n) && \text{Toplamı dağıt} \\
 &= \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + \frac{2b}{n} (n-1) + \Theta(n) && \text{Toplamı değerlendir:} \\
 &\leq \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + 2b + \Theta(n) && \mathbf{b+b+...+b = b (n-1)} \\
 &&& \text{Çünkü } n-1 < n, \frac{2b(n-1)}{n} < 2b
 \end{aligned}$$

# Quicksort Analizi: Ortalama Durum

$$T(n) \leq \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + 2b + \Theta(n), \quad \sum_{k=1}^{n-1} k \lg k \leq \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \text{ oldugundan}$$

$$\leq \frac{2a}{n} \left( \frac{1}{2} n^2 \lg n - \frac{1}{8} n^2 \right) + 2b + \Theta(n)$$

$$= an \lg n - \frac{a}{4} n + 2b + \Theta(n)$$

$$= an \lg n + b + \left( \Theta(n) + b - \frac{a}{4} n \right) \quad \text{Ispat: } T(n) \leq an \lg n + b$$

$\leq an \lg n + b$  olur.

$\sum_{k=1}^{n-1} k \lg k$  toplamındaki terimler en fazla  $n \lg n$  olur. Bu durumda en fazla  $n$  terim vardır.

$\sum_{k=1}^{n-1} k \lg k \leq n^2 \lg n$  olur.

# 7.Hafta

## Alt Sınırları Sıralama Doğrusal-Zaman (linear time) Sıralaması (devam)

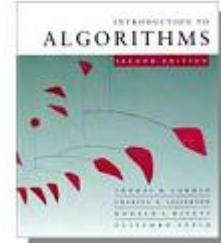
Alt Sınırları Sıralama

- Karar ağaçları

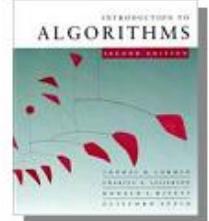
Doğrusal-Zaman Sıralaması

- Sayma sıralaması
- Taban sıralaması
- Kova sıralaması

# Sayma Sıralaması



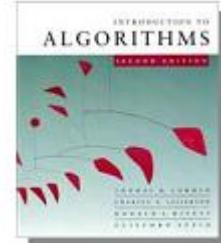
$$\begin{array}{l}
 \Theta(k) \quad \left\{ \begin{array}{l} \textbf{for } i \leftarrow 1 \text{ to } k \\ \quad \textbf{do } C[i] \leftarrow 0 \end{array} \right. \\
 \Theta(n) \quad \left\{ \begin{array}{l} \textbf{for } j \leftarrow 1 \text{ to } n \\ \quad \textbf{do } C[A[j]] \leftarrow C[A[j]] + 1 \end{array} \right. \\
 \Theta(k) \quad \left\{ \begin{array}{l} \textbf{for } i \leftarrow 2 \text{ to } k \\ \quad \textbf{do } C[i] \leftarrow C[i] + C[i-1] \end{array} \right. \\
 \Theta(n) \quad \left\{ \begin{array}{l} \textbf{for } j \leftarrow n \text{ down to } 1 \\ \quad \textbf{do } B[C[A[j]]] \leftarrow A[j] \\ \quad \quad C[A[j]] \leftarrow C[A[j]] - 1 \end{array} \right. \\
 \hline
 \Theta(n + k)
 \end{array}$$



# Taban (Radix) sıralaması

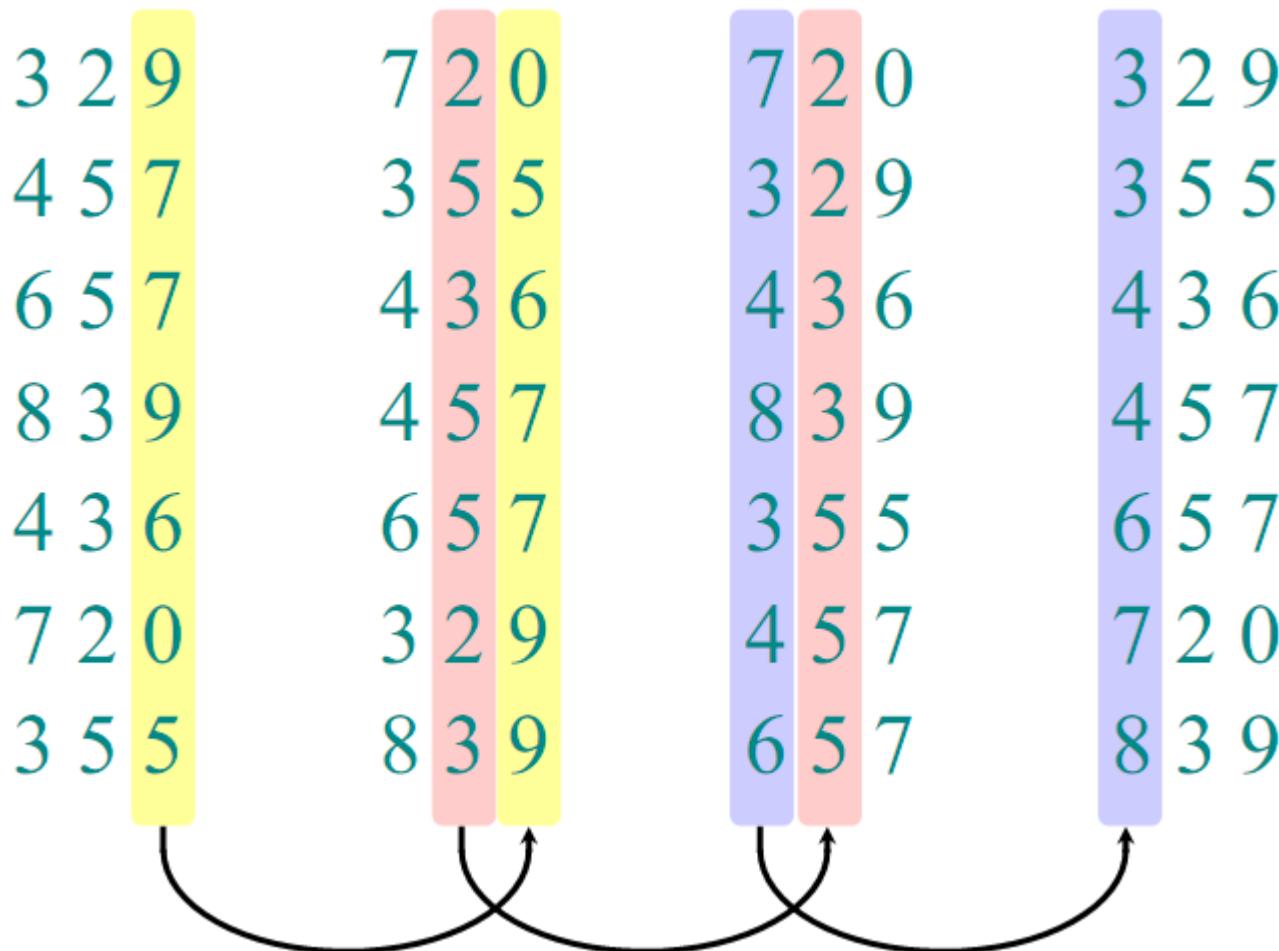
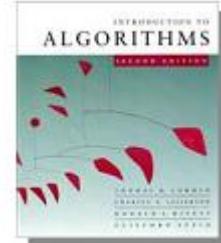
- En az öneme sahip basamaktan başlayarak sıralama yapar.
- Birden fazla anahtara göre sıralama gerektiğinde kolaylıkla kullanılabilir.
- Ör: tarihe göre sıralamada yıl, ay, gün'e göre sıralama yapılır. Tarih sıralamasında önce gün, sonra ay ve yıl' a göre kolayca sıralanır.

# Taban (Radix) sıralaması

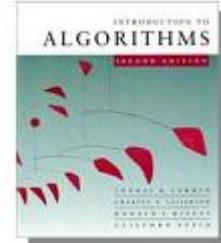


- Basamak basamak sıralama.
- Kötü fikir: sıralamaya önceli en önemli basamaktan başlamak.
- İyi fikir: Sıralamaya **en önemsiz basamaktan** başlamak ve **ek kararlı** sıralama uygulamak.

# Taban sıralaması uygulaması

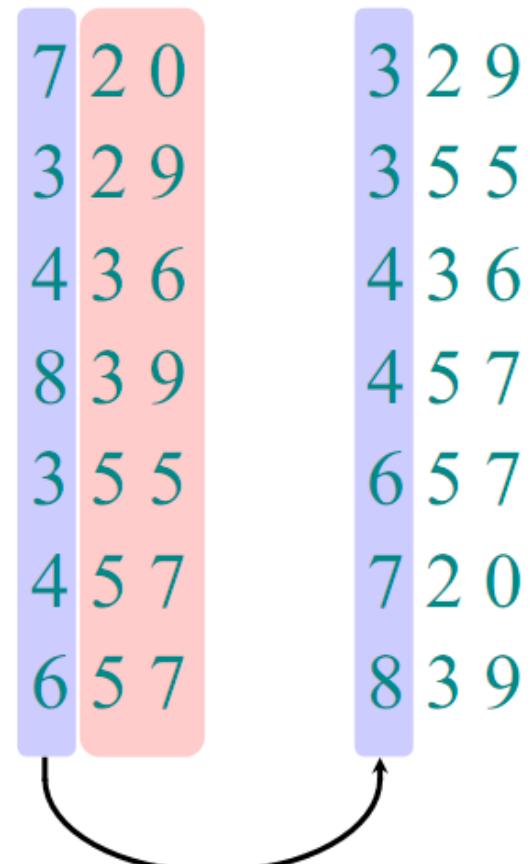


# Taban sıralaması uygulaması

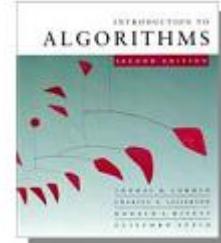


*Basamak konumunda tümevarım*

- Sayıların düşük düzeyli  $t - 1$  basamaklarına göre sıralandığını varsayıın.
- $t$  basamağında sıralama yapın.

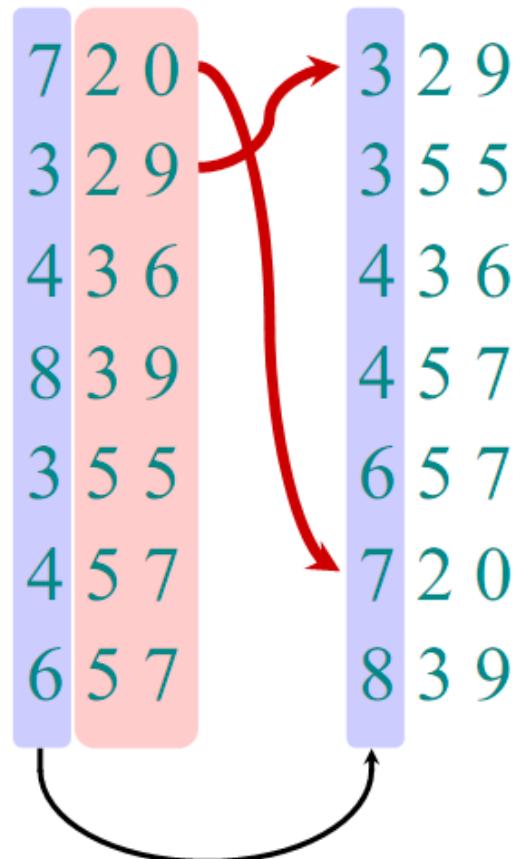


# Taban sıralaması uygulaması

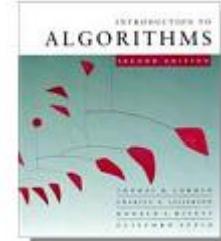


*Basamak konumunda tümevarım*

- Sayıların düşük düzeyli  $t-1$  basamaklarına göre sıralandığını varsayıın.
- $t$  basamağında sıralama yapın.
  - $t$  basamağında farklı olan iki sayı doğru sıralanmış.

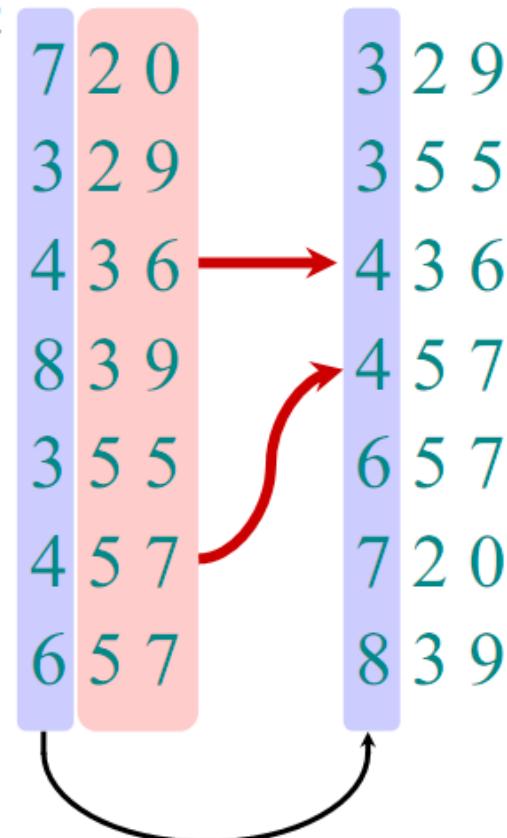


# Taban sıralaması uygulaması

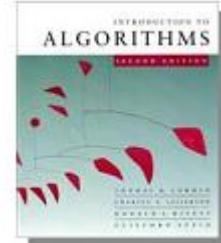


*Basamak konumunda tümevarım*

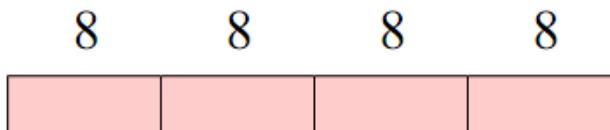
- Sayıların düşük düzeyli  $t - 1$  basamaklarına göre sıralandığını varsayıın.
- $t$  basamağında sıralama yapın.
  - $t$  basamağında farklı olan iki sayı doğru sıralanmış.
  - $t$  basamağındaki iki eşit sayının girişteki sıraları muhafaza edilmiş  $\Rightarrow$  doğru sıra.



# Taban sıralamasının çözümlemesi



- Sayma sıralamasını ek kararlı sıralama varsayıın.
- Herbiri  $b$  bit olan  $n$  bilgiişlem sözcüğünü sıralayıın.
- Her sözcüğün basamak yapısı  $b/r$  taban- $2^r$  olarak görülebilir.

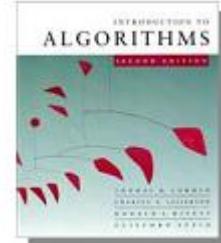


**Örnek:** 32-bit sözcük

$r = 8 \Rightarrow b/r = 4$  ise, taban- $2^8$  basamak durumunda sıralama 4 geçiş yapar; veya  $r = 16 \Rightarrow b/r = 2$  ise, taban- $2^{16}$  basamakta 2 geçiş yapar.

*Kaç geçiş yapmalıyız?*

# Taban sıralamasının çözümlemesi

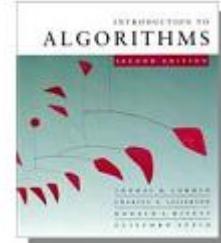


**Hatırla:** Sayma sıralaması  $\Theta(n + k)$  süresini alır; ( $0$  ile  $k - 1$  aralığında  $n$  sayıyı sıralamak için). Her  $b$ -bitlik sözcük  $r$ -bitlik parçalara ayrılrsa, sayma sıralamasının her geçisi  $\Theta(n + 2^r)$  süre alır. Bu durumda  $b/r$  geçiş olduğundan, elimizde:

$$T(n, b) = \Theta\left(\frac{b}{r}(n + 2^r)\right) \text{ olur.}$$

- $r'$  yi,  $T(n, b)$ ' yi en aza düşürecek gibi seçin:
- $r'$ yi arttırmak daha az geçiş demektir, ama  $r >> \lg n$  olduğundan, süre üstel olarak artar.

# r' yi seçmek



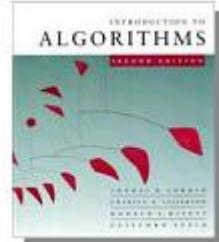
(r'ye göre)  
 $T(n, b)$ 'yi türevini alıp 0' a eşitleyerek en aza düşürün.

Veyahut da , istemediğimiz değer  $2^r >> n$  olduğundan, bu sınırlamaya bağlı kalarak r'yi olabildiğince büyük seçmenin asimptotik bir sakıncası olmadığını gözleyin.

$r = \lg n$  seçimi  $T(n, b) = \Theta(bn/\lg n)$  anlamına gelir.

- 0 ile  $n^d - 1$  aralığındaki sayılarla  $b = d \lg n$  'yi elde ederiz.  $\Rightarrow$  taban sıralaması  $\Theta(dn)$  süresini alır.

Not: Değer aralığı  $0 \dots 2^b - 1 = 0 \dots n^d$  kabul edip her iki tarasın logaritması alınır.  
 $b = d \lg n$  olur. Burada d basamak sayısıdır.



# Sonuçlar

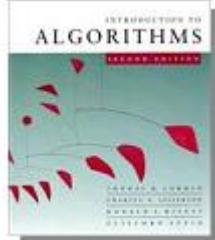
Pratikte taban sıralaması büyük girişler için hızlıdır; aynı zamanda kod yazması ve bakımı kolaydır.

**Örnek** (32-bitlik sayılar için):

- En çok 3 geçiş ( $\geq 2000$  sayının sıralanmasında).
- Birleştirme sıralaması /çabuk sıralama $\lceil \lg 2000 \rceil$  en az 11 geçiş yaparlar.

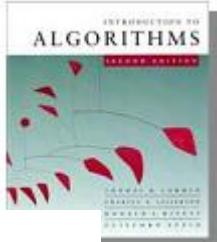
**Dezavantajı:** Çabuk sıralamanın aksine, taban sıralamasının yer referansları zayıftır ve bu nedenle ince ayarlı bir çabuk sıralama, dik bellek sıradüzeni olan günümüz işlemcilerinde daha iyi çalışır.

Pratikte radix sort yani taban sıralaması, sayılarınız gerçekten küçük değilse çok hızlı bir algoritma değildir.



# Radix Sort

```
● public void RadixSort(int[] Dizi) {
 ● // Yardımcı dizimiz
 ● int[] t = new int[Dizi.Length];
 ● // her defasında kaç bit işleme alınacak
 ● int r = 4; // 2, 8 veya 16 bit ile kaç geçiş yapılacağı denenebilir
 ● // int 4 byte yani 32 bit
 ● int b = 32;
 ● // counting ve prefix dizileri
 ● // (unutmayın bu dizlerin boyutu 2^r düzeyinde olacaktır.
 ● // Bu değerin uygun seçilmesi gereklidir r=4 için dizi boyutu 16
 ● // r=16 için dizi boyutu 65535)
 ● int[] count = new int[1 << r];
 ● int[] pref = new int[1 << r];
 ● // grupların sayısı yani geçiş bulunuyor
 ● int groups = (int)Math.Ceiling((double)b / (double)r);
 ● // gruplara uygulanacak maske
 ● int mask = (1 << r) - 1; //r=4 için 15
```



# Radix Sort

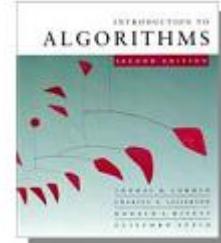
```

o for (int c = 0, shift = 0; c < groups; c++, shift += r)
o { // count dizisini resetleme
o for (int j = 0; j < count.Length; j++) count[j] = 0;
o // cinci grubun elamanlarının sayılması
o for (int i = 0; i < Dizi.Length; i++)
o count[(Dizi[i] >> shift) & mask]++;
o // prefix dizisinin hesaplanması
o pref[0] = 0;
o for (int i = 1; i < count.Length; i++)
o pref[i] = pref[i - 1] + count[i - 1];
o // t[] dizisine cinci grupta sıralanmış elamanların indisine
o // uygun değerin Dizi den atanması
o for (int i = 0; i < Dizi.Length; i++)
o t[pref[(Dizi[i] >> shift) & mask]] = Dizi[i];
o // Dizi[]=t[] cinci guruba göre sıralanmış değerlerin diziye aktarılması
o t.CopyTo(Dizi, 0);
o }
o // Dizi sıralı
o }

```

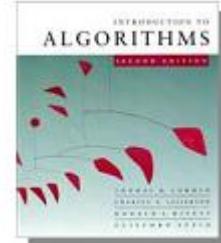
Çıktı: 32/4= 8 geçiş  
256 , 1 , 120 , 10 , 235 , 987  
256 , 1 , 10 , 120 , 987 , 235  
1 , 10 , 120 , 235 , 256 , 987  
1 , 10 , 120 , 235 , 256 , 987  
1 , 10 , 120 , 235 , 256 , 987  
1 , 10 , 120 , 235 , 256 , 987  
1 , 10 , 120 , 235 , 256 , 987  
1 , 10 , 120 , 235 , 256 , 987

# Kova Sıralama (Bucket Sort)



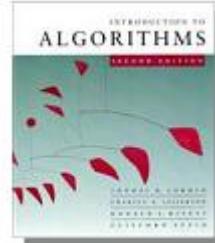
- Kova Sıralaması (ya da *sepet sıralaması*), sıralanacak bir diziyi parçalara ayırarak sınırlı sayıdaki *kovalara* (ya da *sepetlere*) atan bir sıralama algoritmasıdır.
- Ayışma işleminin ardından her kova kendi içinde ya farklı bir algoritma kullanılarak ya da kova sıralamasını özyinelemeli olarak çağırarak sıralanır.
- Kova sıralaması aşağıdaki biçimde çalışır:
- Başlangıçta boş olan bir "kovalar" dizisi oluştur.
- Asıl dizinin üzerinden geçerek her öğeyi ilgili aralığa denk gelen kovaya at.
- Boş olmayan bütün kovaları sırala.
- Boş olmayan kovalardaki bütün öğeleri yeniden diziye al.

# Kova Sıralama (Bucket Sort)

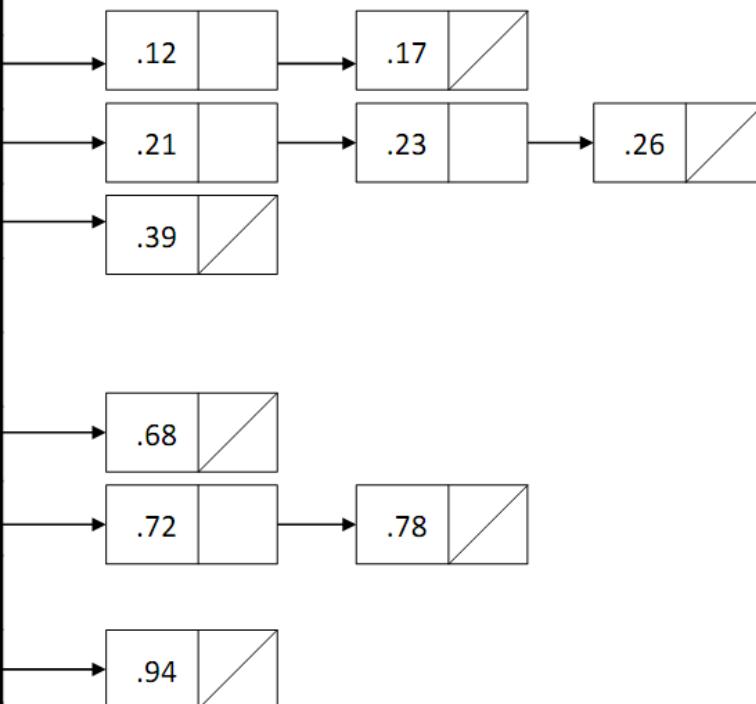


- Kova sıralaması doğrusal zamanda çalışır.
- Girişin düzgün dağılımlı olduğu kabul edilir.
- Random olarak  $[0,1]$  aralığında oluşturulmuş giriş bilgileri olduğu kabul edilir.
- Temel olarak  $[0, 1)$  aralığını  $n$  eşit alt aralığa böler ve girişi bu aralıklara dağıtır.
- Aralıklardaki değerleri insert sort ile sıralar.
- Aralıkları bir biri ardına ekleyerek sıralanmış diziyi elde eder.

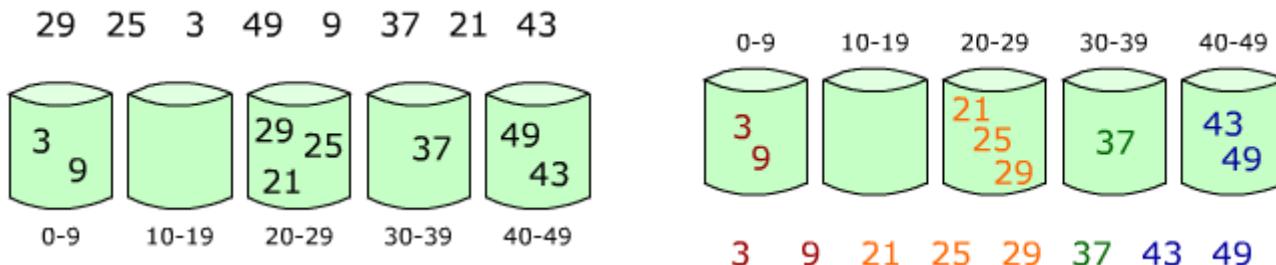
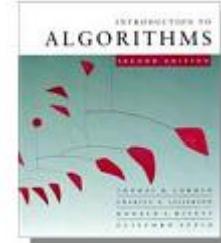
# Kova Sıralama:



|    | A   | B |
|----|-----|---|
| 1  | .78 | 0 |
| 2  | .17 | 1 |
| 3  | .39 | 2 |
| 4  | .26 | 3 |
| 5  | .72 | 4 |
| 6  | .94 | 5 |
| 7  | .21 | 6 |
| 8  | .12 | 7 |
| 9  | .23 | 8 |
| 10 | .68 | 9 |



# Kova Sıralama (Bucket Sort)



## BUCKET-SORT (A)

- $\Theta(n)$  1.  $n \leftarrow \text{length}(A)$
- $\Theta(n)$  2. **for**  $i \leftarrow 0$  **to**  $n$
- $\Theta(n)$  3.     **do** insert  $A[i]$  into list  $B[\lfloor n \times A[i] \rfloor]$
- $\Theta(n)$  4. **for**  $i \leftarrow 0$  **to**  $n$
- $\Theta(n^2)$  5.     **do** sort  $B[i]$  with insertion sort
- $\Theta(n)$  6. concatenate the lists  $B[0], B[1], \dots, B[n-1]$  together in order

**Çalışma zamanı=  $\Theta(n)$**

$\Theta(n_i^2)$   $B[i]$ . Yerdeki eleman sayısını ifade eder. Beklenen çalışma süresi doğrusaldır

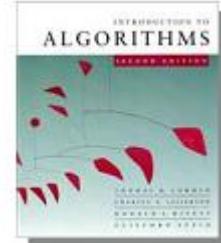
Worst case performance

$O(n^2)$

Average case performance

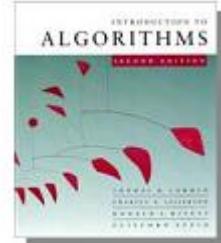
$O(n + k)$

# Kova Sıralama (Bucket Sort)



- Değer aralıkları çok büyük seçilirse veya girişler düzgün dağılımlı değil ise sıralama  $O(n^2)$  olur. Değer aralığını yani kova sayısını bulmak için
- Dizi boyutu\*i. Dizi elamanı/(Girişlerin maksimumu+x\_sayı)
- ile bulunabilir.
- Aralık-kova sayısı  $m \rightarrow n * \max(A[i]) / (\max(A[i]) + x)$
- 29 25 3 49 9 37 21 43
- 0 1 2 3 4 5 6 7
- $n=8$ , giriş maksimum değeri 49,  $k > \text{maks} \rightarrow k=50$  üst sınır
- $29 * 8 / 50 = 4,64 \rightarrow \text{Kova}[4] = 29$
- $25 * 8 / 50 = 4 \rightarrow \text{Kova}[4] = 25 \rightarrow 29$
- $3 * 8 / 50 = 0,48 \rightarrow \text{Kova}[0] = 3$
- $49 * 8 / 50 = 7,84 \rightarrow \text{Kova}[7] = 49$

# Kova Sıralama:



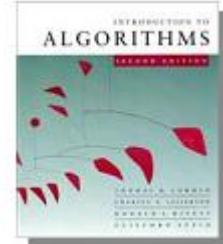
| Algorithm     | Worst-case running time | Average-case/ expected running time |
|---------------|-------------------------|-------------------------------------|
| Insert Sort   | $\Theta(n^2)$           | $\Theta(n^2)$                       |
| Merge Sort    | $\Theta(n \log n)$      | $\Theta(n \log n)$                  |
| Heap Sort     | $\Theta(n \log n)$      | $\Theta(n \log n)$                  |
| QuickSort     | $\Theta(n^2)$           | $\Theta(n \log n)$                  |
| Counting Sort | $\Theta(n+k)$           | $\Theta(n+k)$                       |
| Radix Sort    | $\Theta(d(n+k))$        | $\Theta(d(n+k))$                    |
| Bucket Sort   | $\Theta(n^2)$           | $\Theta(n)$                         |

# Sıra İstatistikleri

## Order Statistics

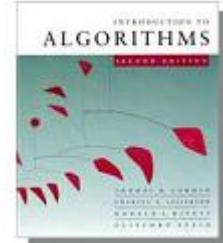
Rastgele böl ve fethet

- Beklenen sürenin çözümlemesi
- En kötü durum doğrusal-süre sıra istatistikleri
- Çözümleme



# Sıra İstatistikleri

- Doğrusal zaman çözümüne gereksinim duyulur.
- $n$  elamanlı bir dizide  $i$  'inci sıra istatistiği,  $i$ 'inci en küçük elemanı bulmak
- $i=1$  ise *minimum*
- $i=n$  ise *maximum*
- $i=n/2$  orta değeri (*medyan*)
  - Eğer  $n$  tek ise, 2 medyan vardır.
- *Sıra istatistiğini nasıl hesaplayabiliriz?*
- *Çalışma zamanı nedir?*



# Sıra istatistikleri (doğrusal zamanda)

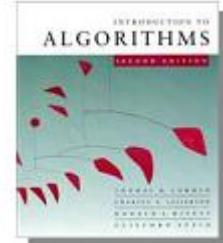
$n$  elemanın  $i$ 'inci küçük değerini seçin  
( $i$  ranklı eleman).

- $i = 1$ : **minimum**; (en az)
- $i = n$ : **maximum**; (en çok)
- $i = \lfloor (n+1)/2 \rfloor$  veya  $\lceil (n+1)/2 \rceil$ : **median**. (ortanca)

*Saf algoritma*:  $i$ 'inci elemanı sırala ve dizinle.

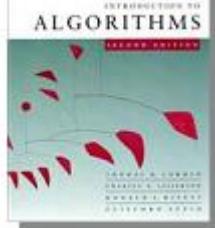
En kötü koşma süresi =  $\Theta(n \lg n) + \Theta(1)$   
 $= \Theta(n \lg n)$ ,

birleştirme veya yiğin sıralaması kullan (*çabuk sıralamayı değil*).



# Sıra İstatistikleri

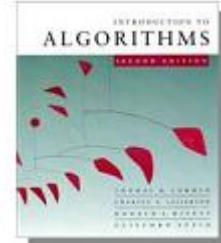
- **$O(n \lg n)$**  daha iyisi olabilir mi?
- Bir dizideki minimum elemanı bulmak için kaç karşılaştırma gereklidir?
- Minimum ve Maksimumu, 2 kez daha az maliyetli bulabilir miyiz?
- Evet:
  - Çiftler şeklinde ilerleyerek
    - Diğer çifteki her bir eleman ile karşılaştır.
    - Maksimum için en büyük, minimum için en küçük elamanılaştır.
  - Toplam maliyet: 2 elaman başına 3 karşılaştırma =  $O(3n/2)$



# Sıra istatistiklerinin Bulunması: Seçim Problemi

- Seçme daha ilginç problemdir: Bir dizideki  $i$ 'inci en küçük elemanı bulma. Bunun için iki algoritma;
  - Beklenen çalışma zamanı  $O(n)$  olan bir pratik rastgele algoritması
  - En kötü çalışma zamanı sadece  $O(n)$  ile ilgili teorik algoritma
- Anahtar Fikir: Quicksort algoritmasındaki rastgele bölüntüyü kullanmak.
  - Fakat, sadece bir altdizi incelememiz gereklidir
  - Bu işlem çalışma zamanında tasarruf sağlar:  $O(n)$

# Rastgele böl-ve-fethet algoritması

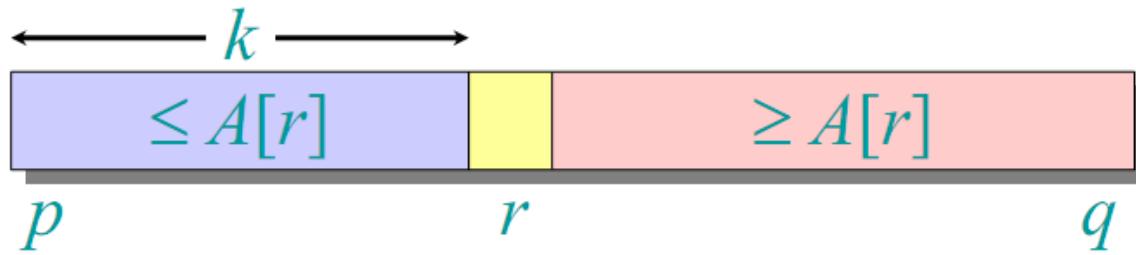


RAND-SELECT( $A, p, q, i$ )  $\triangleright A[p \dots q]$ 'nın  $i$ 'ninci en küçüküğü

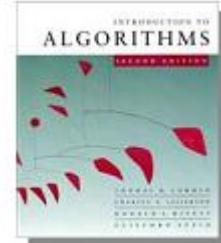
```

if $p = q$ then return $A[p]$
 $r \leftarrow \text{RAND-PARTITION}(A, p, q)$ (Rastgele bölüntü)
 $k \leftarrow r - p + 1$ $\triangleright k = \text{rank}(A[r])$ (rütbeli)
if $i = k$ then return $A[r]$
if $i < k$
 then return RAND-SELECT($A, p, r - 1, i$)
 else return RAND-SELECT($A, r + 1, q, i - k$)

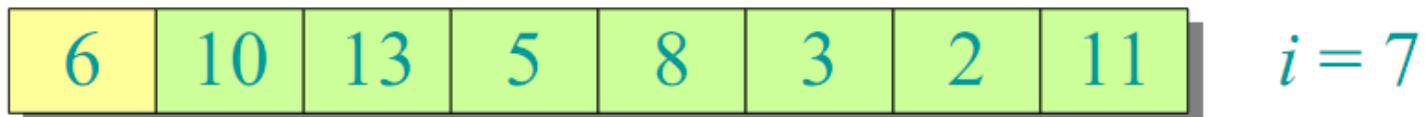
```



# Örnek

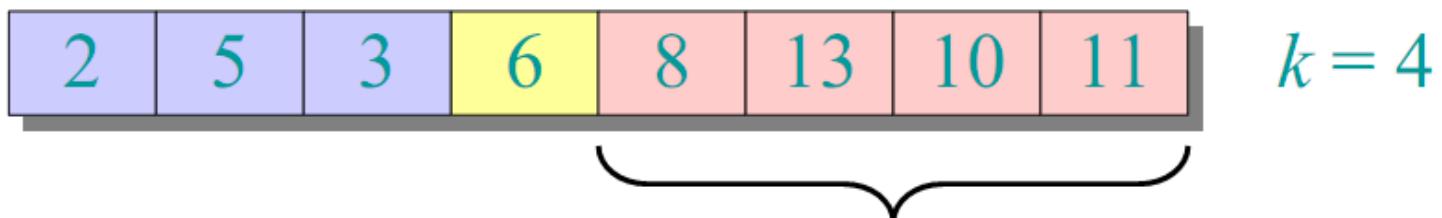


$i = 7$ 'inci en küçük olarak seçin:



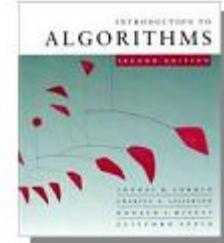
*pivot (esas eleman)*

Partition (Bölüntü):



$7 - 4 = 3$  'üncü küçüğü özyinelemeyle seçin.

# Rastgele Seçme Analizi: Çözümlemede sezgi (öngörü)



- (Çözümlemelerin hepsinde tüm elemanların farklı olduğu varsayılıyor.)

## Şanslı durum:

$$\begin{aligned} T(n) &= T(9n/10) + \Theta(n) \\ &= \Theta(n) \end{aligned}$$

En iyi durum (Best case):  
9:1 bölüntü (partition)  
olduğunu farz edin

$$n^{\log_{10/9} 1} = n^0 = 1$$

DURUM 3

## Şanssız durum:

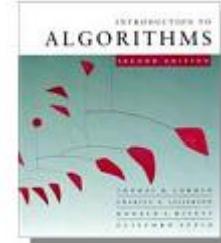
$$\begin{aligned} T(n) &= T(n - 1) + \Theta(n) \\ &= \Theta(n^2) \end{aligned}$$

aritmetik seri

*Sıralamadan daha kötü!*

En kötü durum(Worst case):  
Bölüntü daima 0:n-1

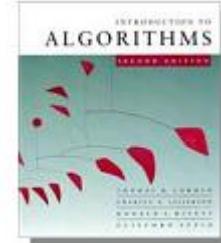
# Beklenen süre çözümlemesi: (Average Case)



- Çözümleme rastgele çabuk sıralamanın benzeri ama bazı farkları var.
- $T(n)$ , Rastgele-seçim çalışma zamanının rastgele değişkeni olsun ( $n$  boyutlu bir girişte), ve rastgele sayılar birbirinden bağımsız olsun.
- $k = 0, 1, \dots, n-1$  için **göstergesel rastgele değişkeni** tanımlayın.

$$X_k = \begin{cases} 1 & \text{eğer } BÖLÜNTÜ \ k : n-k-1 \text{ bölmeli ise,} \\ 0 & \text{diğer durumlarda.} \end{cases}$$

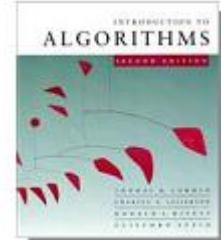
# Beklenen süre çözümlemesi



- Bir üst sınır elde etmek için,  $i$ 'inci elemanın her zaman bölüntünün büyük bölgesinde olduğunu varsayıın:

$$\begin{aligned}
 T(n) &= \begin{cases} T(\max\{0, n-1\}) + \Theta(n), & 0 : n-1 \text{ bölünmesi,} \\ T(\max\{1, n-2\}) + \Theta(n), & 1 : n-2 \text{ bölünmesi,} \\ \vdots \\ T(\max\{n-1, 0\}) + \Theta(n), & n-1 : 0 \text{ bölünmesi,} \end{cases} \\
 &= \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)).
 \end{aligned}$$

# Beklenenin hesaplanması



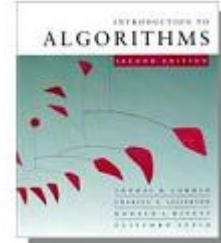
$$E[T(n)] = E\left[ \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)) \right]$$

Her iki taraftaki beklenenleri bulun.

$$\begin{aligned} E[T(n)] &= E\left[ \sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n)) \right] \\ &= \sum_{k=0}^{n-1} E[X_k (T(\max\{k, n-k-1\}) + \Theta(n))] \end{aligned}$$

Beklenenin doğrusallığı.

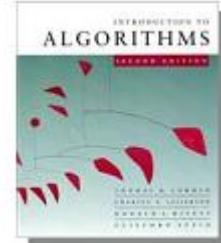
# Beklenenin hesaplanması



$$\begin{aligned}E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right)\right] \\&= \sum_{k=0}^{n-1} E[X_k \left(T(\max\{k, n-k-1\}) + \Theta(n)\right)] \\&= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)]\end{aligned}$$

$X_k$ 'nın diğer rastgele seçimlerden bağımsızlığı.

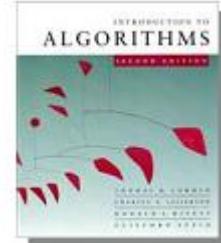
# Beklenenin hesaplanması



$$\begin{aligned}
 E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n))\right] \\
 &= \sum_{k=0}^{n-1} E[X_k (T(\max\{k, n-k-1\}) + \Theta(n))] \\
 &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \\
 &= \frac{1}{n} \sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n)
 \end{aligned}$$

Beklenenin doğrusallığı;  $E[X_k] = 1/n$ .

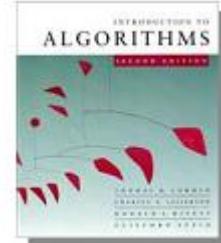
# Beklenenin hesaplanması



$$\begin{aligned}
 E[T(n)] &= E\left[\sum_{k=0}^{n-1} X_k (T(\max\{k, n-k-1\}) + \Theta(n))\right] \\
 &= \sum_{k=0}^{n-1} E[X_k (T(\max\{k, n-k-1\}) + \Theta(n))] \\
 &= \sum_{k=0}^{n-1} E[X_k] \cdot E[T(\max\{k, n-k-1\}) + \Theta(n)] \\
 &= \frac{1}{n} \sum_{k=0}^{n-1} E[T(\max\{k, n-k-1\})] + \frac{1}{n} \sum_{k=0}^{n-1} \Theta(n) \\
 &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)
 \end{aligned}$$

Üstteki terimler  
iki kez görünüyor.

# Karmaşık yineleme



(Ama çabuk sıralamanınki kadar karmaşık değil.)

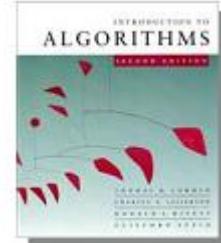
$$E[T(n)] = \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} E[T(k)] + \Theta(n)$$

**Kanıtla:**  $E[T(n)] \leq cn$  sabiti için  $c > 0$ .

- $c$  sabiti öyle büyük seçilebilir ki,  
 $E[T(n)] \leq cn$  tüm taban durumlarında geçerli olur.

**Veri:**  $\sum_{k=\lfloor n/2 \rfloor}^{n-1} k \leq \frac{3}{8}n^2$  (alıştırma).

# Yerine koyma metodu



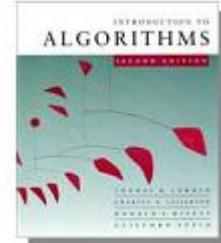
$$E[T(n)] \leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n)$$

Tümevarım hipotezini yerleştirin.

$$\begin{aligned} E[T(n)] &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n) \\ &\leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \end{aligned}$$

Veriyi kullanın.

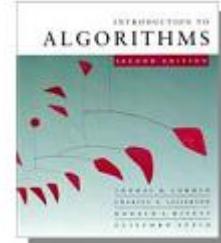
# Yerine koyma metodu



$$\begin{aligned}E[T(n)] &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n) \\&\leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \\&= cn - \left( \frac{cn}{4} - \Theta(n) \right)\end{aligned}$$

*istenen – kalan* şeklinde gösterin.

# Yerine koyma metodu



$$\begin{aligned}
 E[T(n)] &\leq \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + \Theta(n) \\
 &\leq \frac{2c}{n} \left( \frac{3}{8} n^2 \right) + \Theta(n) \\
 &= cn - \left( \frac{cn}{4} - \Theta(n) \right) \\
 &\leq cn,
 \end{aligned}$$

$c$  yeterince büyük seçilirse  $cn/4$ ,  $\Theta(n)$ 'nin üstünde olur.

## İspat, Yerine koyma 2. yöntem

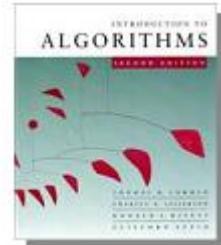
- $T(n) \leq cn$ ,  $c$  sabitini çok büyük seç:

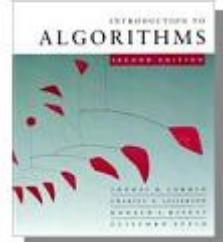
$$\begin{aligned}
 T(n) &\leq \frac{2}{n} \sum_{k=n/2}^{n-1} T(k) + \Theta(n) \\
 &\leq \frac{2}{n} \sum_{k=n/2}^{n-1} ck + \Theta(n) \\
 &= \frac{2c}{n} \left( \sum_{k=1}^{n-1} k - \sum_{k=1}^{n/2-1} k \right) + \Theta(n) \\
 &= \frac{2c}{n} \left( \frac{1}{2}(n-1)n - \frac{1}{2}\left(\frac{n}{2}-1\right)\frac{n}{2} \right) + \Theta(n) \quad \text{Aritmetik seriyi genişlet} \\
 &= c(n-1) - \frac{c}{2}\left(\frac{n}{2}-1\right) + \Theta(n)
 \end{aligned}$$

**Rekürans ile başlanıldı**

**T(k) için alt durum  $T(n) \leq cn$**

**Reküransı böl**





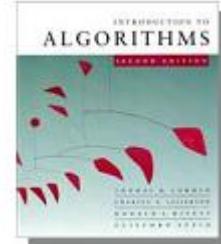
## İspat, Yerine koyma 2. yöntem

- $T(n) \leq cn$ ,  $c$  sabitini çok büyük seç:

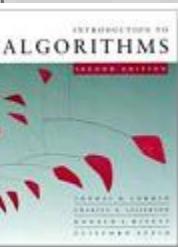
$$\begin{aligned}
 T(n) &\leq c(n-1) - \frac{c}{2} \left( \frac{n}{2} - 1 \right) + \Theta(n) \\
 &= cn - c - \frac{cn}{4} + \frac{c}{2} + \Theta(n) \\
 &= cn - \frac{cn}{4} - \frac{c}{2} + \Theta(n) \\
 &= cn - \left( \frac{cn}{4} + \frac{c}{2} - \Theta(n) \right) \\
 &\leq cn \quad (c, yeterince büyük ise)
 \end{aligned}$$

İspat

# Rastgele sıra istatistik seçiminin özeti



- Hızlı çalışır: doğrusal beklenen süre.
- Pratikte mükemmel bir algoritma.
- Ama, en kötü durumu **çok** kötü:  $\Theta(n^2)$  .
  - **Q.** En kötü durumda doğrusal zamanda çalışan bir algoritma var mıdır?
  - **A.** Evet, Blum, Floyd, Pratt, Rivest ve Tarjan [1973] sayesinde vardır. Çok karmaşık bir algoritmadır.
  - **FIKİR:** İyi bir pivotu yinelemeyle üretmek.
  - $n=100$  elaman olduğunu düşünün



# En kötü durum doğrusal-zaman sıra istatistikleri

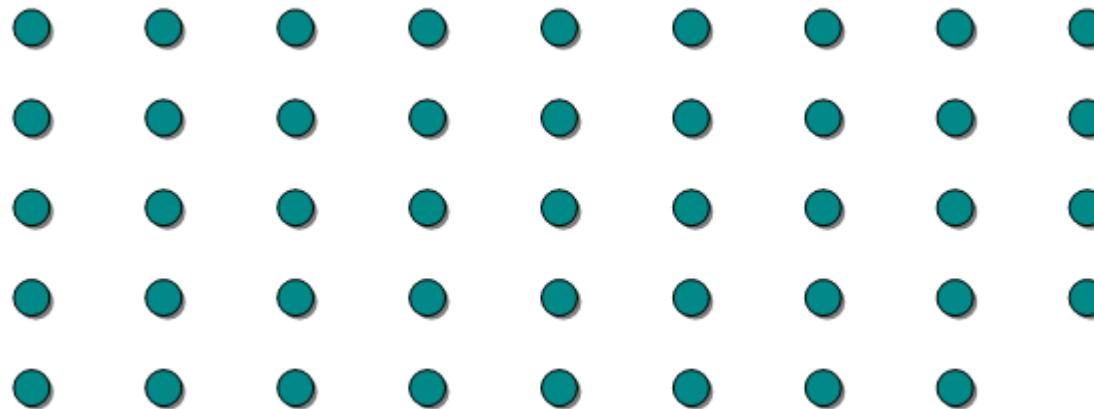
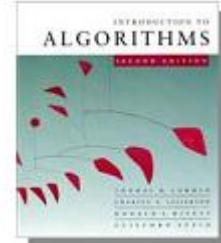
SEÇ ( $i, n$ )

1.  $n$  elemanı  $5'$  li gruplara bölün. Her  $5'$  li grubun ortancasını ezbere bulun.
2.  $\lfloor n/5 \rfloor$  gruplarının ortancası olacak  $x'$  i yinelemeli SEÇME ile pivot olarak belirleyin.
3. Pivot  $x$  etrafında bölüntü yapın.  $k = \text{rank}(x)$ .
 

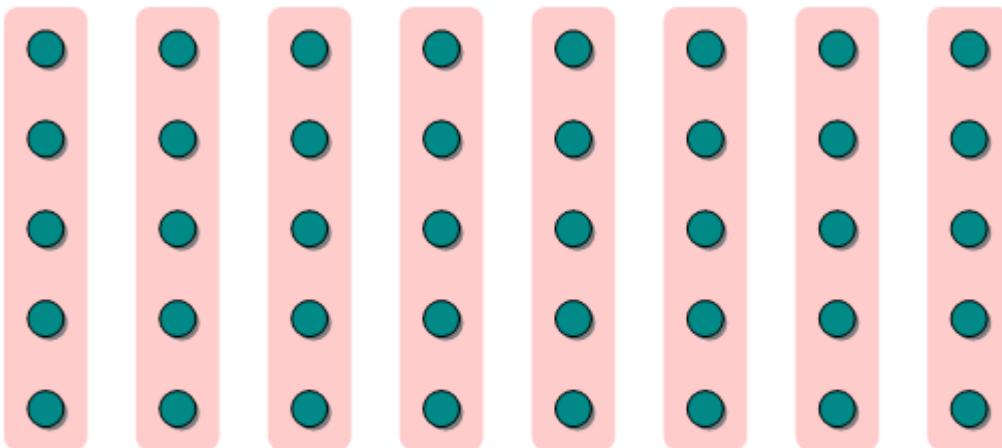
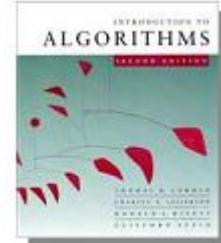
```
if i = k then return x (eğer / öyleyse çıkar)
elseif i < k (diğer durumlarda)
 then i' ninci en küçük elemanı alt bölgede
 yinelemeyle SEÇİN.
 else (i-k)' ninci en küçük elemanı
 üst bölgede yinelemeyle SEÇİN.
```

} RASTGELE-  
SEÇİMİN  
aynısı

# Pivot seçimi

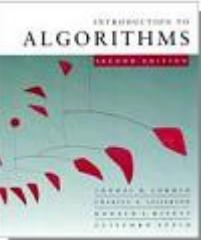


# Pivot seçimi

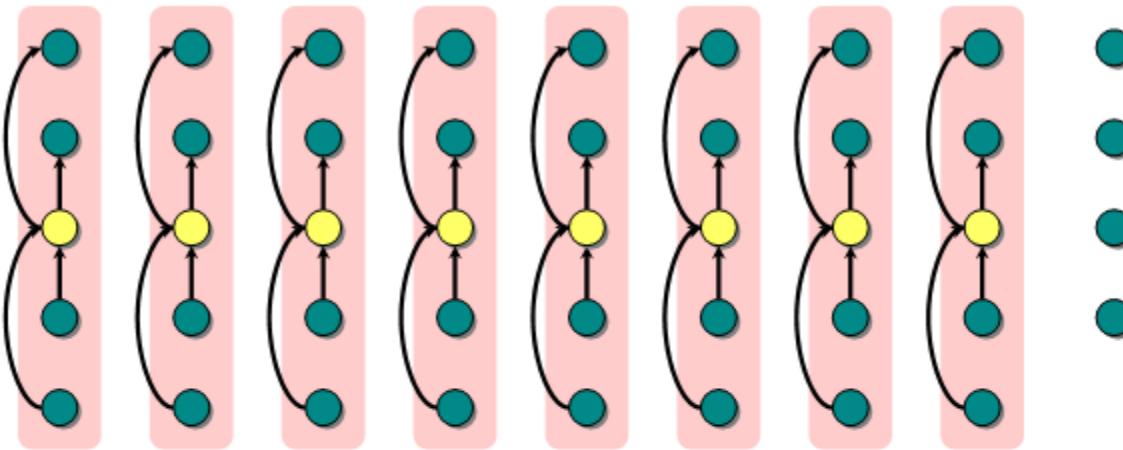


- $n$  her zaman tam bölünmeyebilir son grupta elemanlar eksik kalabilir bu durumda o sütun dikkate alınmaz.

1.  $n$  elemanı  $5'$  li gruplara bölün.



# Pivot seçimi



1.  $n$  elemanlarını  $5'$  li gruplara bölün. 5-elemanlı *daha az* grupların ortancasını ezbere bulun.

$n/5$  öbek, her birinde beş eleman var; her birinin ortancasını hesaplamak ne kadar zaman alır?

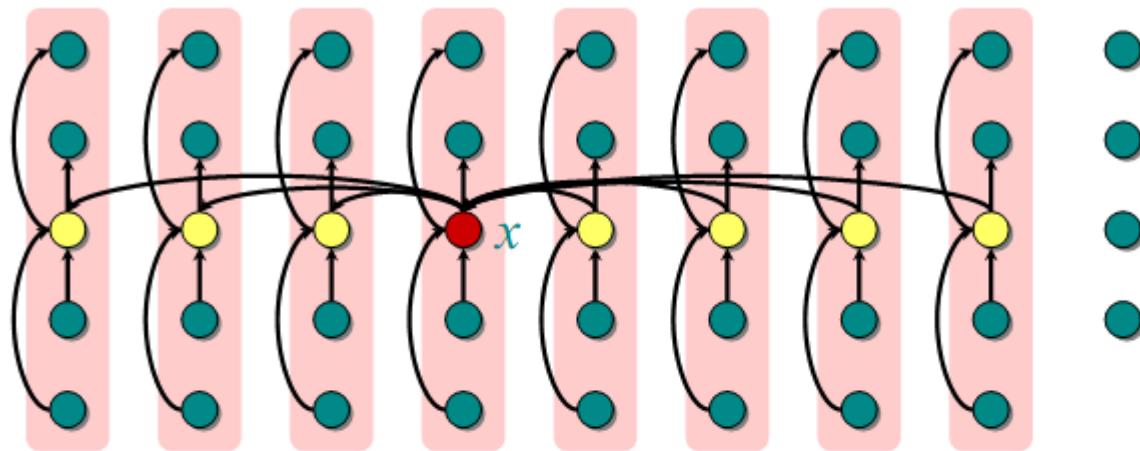
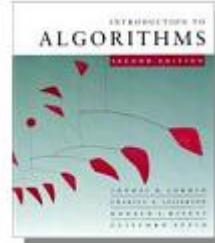
2 kere  $n/5$ . Yani, karşılaştırmaları sayıyorsunuz ve bu  $\Theta(n)$  'dir.

Sonuçta her grupta 5 sayı var ve sabit sayıda karşılaştırma yapılır.



*daha fazla*

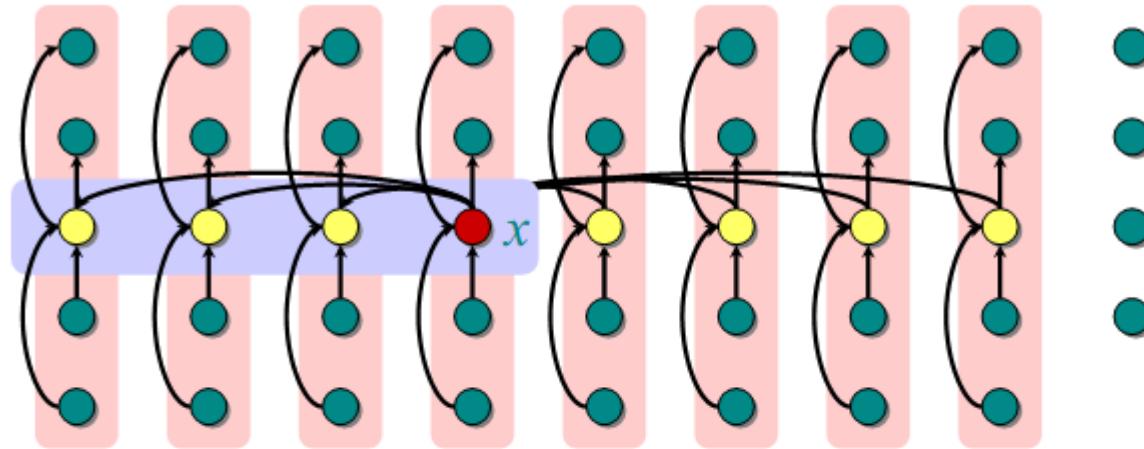
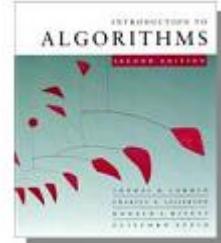
# Pivot seçimi



1.  $n$  elemanlarını  $5'$  li gruplara bölün. 5 elemanlı grupların ortancalarını ezbere bulun.
2.  $\lfloor n/5 \rfloor$  gruplarının ortacısı olacak  $x'$  i, yinelemeli SEÇME ile pivot olarak belirleyin.

*daha az*  
  
*daha çok*

# Çözümleme



Grup ortancalarının en az yarısı  $\leq x$ , bu da  
en az  $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$  grup ortancası eder.

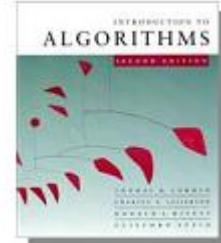
$n=100$ ,  $n/5= 20$ ,  $20/2 = 10$  ortanca değer

*daha az*

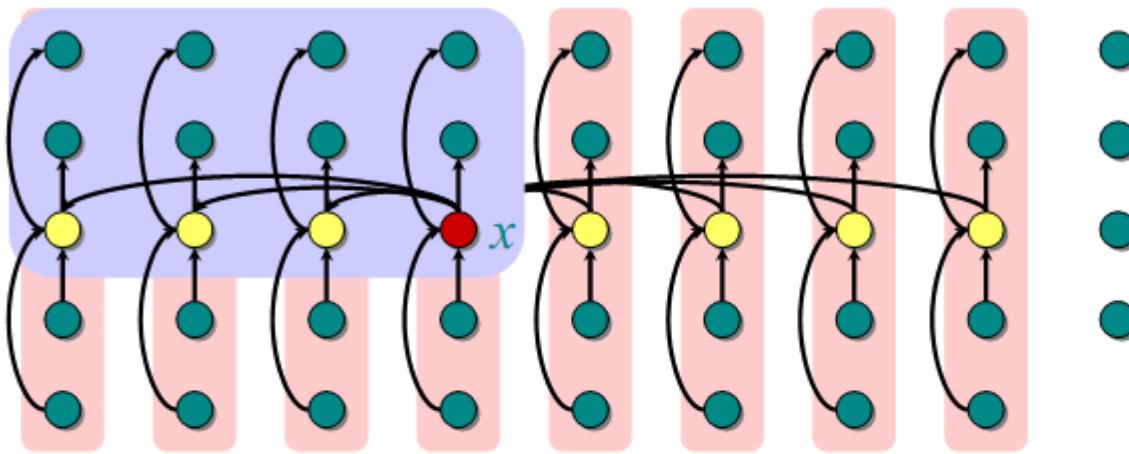


*daha çok*

# Çözümleme (Tüm elemanları farklı varsayı.)



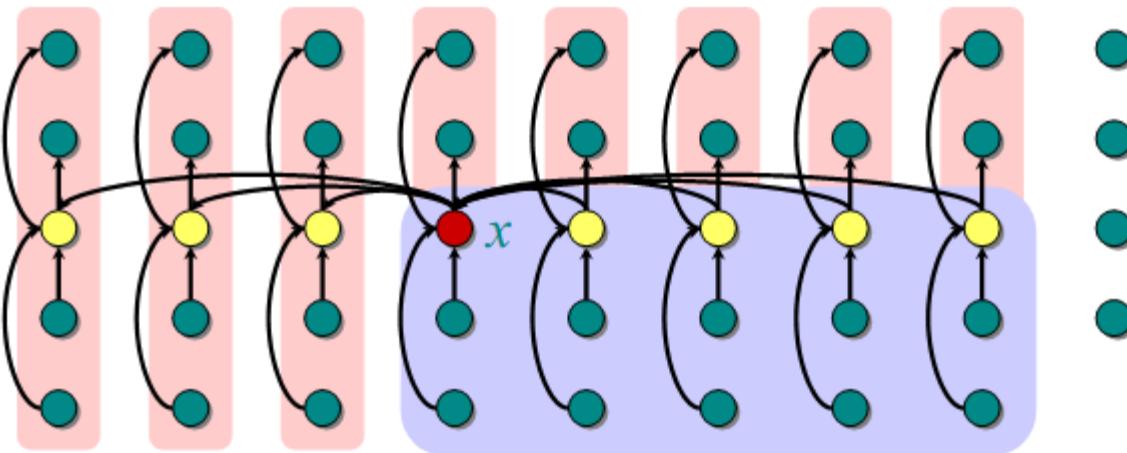
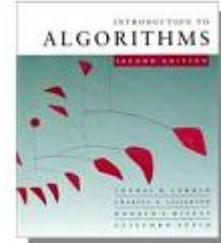
Her grubun  
içinde 3  
eleman var



Grup ortancalarının en az yarısı  $\leq x$ , bu da  
en az  $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$  grup ortancası eder.  
• Bu nedenle, en az  $3\lfloor n/10 \rfloor$  eleman  $\leq x$ .

*daha az*  
  
*daha çok*

# Çözümleme (Tüm elemanları farklı varsayı.)



Grup ortancalarının en az yarısı  $\leq x$ , bu da en az  $\lfloor \lfloor n/5 \rfloor /2 \rfloor = \lfloor n/10 \rfloor$  grup ortancası eder.

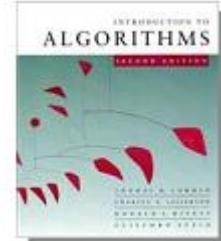
- Bu nedenle, en az  $3\lfloor n/10 \rfloor$  eleman  $\leq x$ .
- Benzer şekilde, en az  $3\lfloor n/10 \rfloor$  eleman  $\geq x$ .

*daha az*



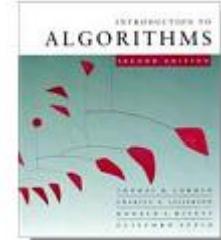
*daha çok*

# Önemsiz basitleştirme



- $n \geq 50$  için,  $3\lfloor n/10 \rfloor \geq n/4$  olur.
- Bu nedenle,  $n \geq 50$  için Adım 4'teki SEÇİM özyinelemeli olarak  $\leq 3n/4$  eleman kapsamında yapılır.  
Kalan  $7n/10$  alınsa da çözüm değişmez
- Böylece, koşma süresinin yinelemesinde Adım 4'ün en kötü durumda  $T(3n/4)$  zamanı alacağı farz edilebilir.
- $n < 50$  için en kötü sürenin  $T(n) = \Theta(1)$  olduğunu biliyoruz.

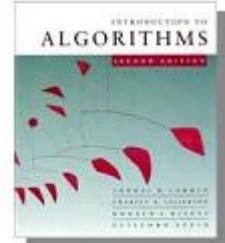
# Yinelemeyi geliştirmek



$T(n)$       SELECT ( $i, n$ )    (SEÇİN)

|             |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
|-------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| $\Theta(n)$ | <ol style="list-style-type: none"> <li>1. <math>n</math> elemanı 5' li gruptara ayırin. 5-elemanlı grupların ortancasını ezberden bulun.</li> </ol>                                                                                                                                                                                                                                                                                                         |
| $(n/5)$     | <ol style="list-style-type: none"> <li>2. <math>\lfloor n/5 \rfloor</math> gruplarının ortancası olacak <math>x'</math> i, yinelemeli SEÇME ile pivot olarak belirleyin.</li> </ol>                                                                                                                                                                                                                                                                         |
| $\Theta(n)$ | <ol style="list-style-type: none"> <li>3. Pivot <math>x</math> etrafında bölüntü yapın. <math>k = \text{rank}(x)</math> olsun.</li> <li>4. <b>if</b> <math>i = k</math> <b>then return</b> <math>x</math><br/><b>elseif</b> <math>i &lt; k</math><br/>    <b>then</b> <math>i'</math> ninci en küçük elemanı alt bölgede yinelemeli olarak SEÇİN.<br/><b>else</b> <math>(i-k)</math>'ninci en küçük elemanı üst bölgede yinelemeli olarak SEÇİN.</li> </ol> |
| $T(3n/4)$   |                                                                                                                                                                                                                                                                                                                                                                                                                                                             |

# Yinelemeyi çözmek



$$T(n) = T\left(\frac{1}{5}n\right) + T\left(\frac{3}{4}n\right) + \Theta(n)$$

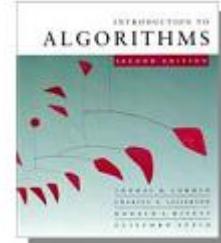

---

**Yerine koyma:**

$$\begin{aligned} T(n) &\leq \frac{1}{5}cn + \frac{3}{4}cn + \Theta(n) \\ T(n) &\leq cn \quad , \\ &= \frac{19}{20}cn + \Theta(n) \\ &= cn - \left( \frac{1}{20}cn - \Theta(n) \right) \\ &\leq cn \quad , \end{aligned}$$

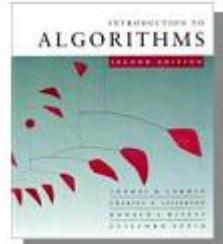
$c$ , hem  $\Theta(n)$ 'i hem de başlangıç koşullarını göztererek yeterince büyük seçilirse...

# Sonuçlar



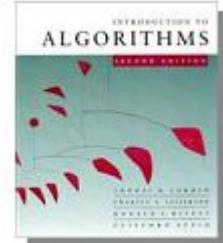
- Yinelemenin her düzeyindeki iş sabit bir kesir (**19/20**) oranında küçüldüğünden, düzeylerdeki iş bir geometrik seri gibidir ve kökteki doğrusal iş ön plana çıkar.
- Pratikte bu algoritma yavaş çalışır, çünkü **n'** nin önündeki sabit büyktür (1 yakın bir değer, Eğer 1 olsaydı  $T(n) \leq cn$  olmazdı)
- Rastgele algoritma çok daha pratiktir.
- **Alıştırma:** Neden **3'** lü gruplara bölmüyoruz?

# **Bilinen Probleme İndirgeme Tasarım Yöntemi**



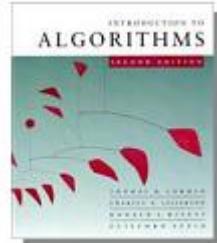
# Bilinen Probleme İndirgeme

- Bu yöntemde, karmaşık olan problem çözümü yapılmadan önce problem bilinen problemlerden birine dönüştürülür ve ondan sonra bilinen problemin çözümü nasıl yapılıyorsa, bu problemin de çözümü benzer şekilde yapılır.
- Problemi bilinen bir probleme dönüştürme işlemi sofistike(yapmacık) bir işlemidir, bundan dolayı çok karmaşık problemlerde her zaman başarılı olmak mümkün olmayabilir. Belki de problem alt problemlere bölündükten sonra her alt problemin bilinen probleme dönüşümü yapılacaktır.
- Çözümü yapılmış problemlerin algoritmalarının daha etkili hale getirilmesi için de bu tasarım yöntemine başvurulabilir.



# Bilinen Probleme İndirgeme

- **Örnek 1:** Verilen bir dizi ya da liste içerisinde tekrar eden sayılar var mıdır? Tekrar varsa, tekrar eden sayıdan kaç tane vardır? Birbirinden farklı kaç tane tekrar eden sayı vardır ve her birinden kaç tane vardır?
- Bu sorulara cevap vermenin farklı yolları olabilir. Bunlar içinde en etkili algoritma hangi yöntemle elde edilmişse, o çözüm en iyi çözüm olarak kabul edilir.
  
- **I. YOL**
- Birinci yol olarak bütün ikililer birbiri ile karşılaştırılırlar. Bu işlemi yapan algoritma;



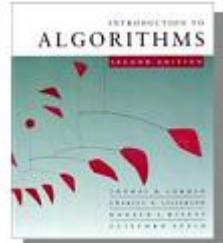
# Bilinen Probleme İndirgeme

## ● I. YOL

► A parametresi içinde tekrar eden sayıların olup olmadığını kontrol edileceği dizidir ve B parametresinin i. elemanı A dizisinin i. elemanından kaç tane olduğunu tutan bir dizidir.

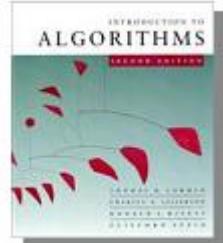
TekrarBul(A,B)

1. **for**  $i \leftarrow 1 \dots n$
2.     $B[i] \leftarrow -1$
3. **for**  $i \leftarrow 1 \dots (n-1)$
4.    **if**  $B[i]=-1$  **then**
5.      $B[i] \leftarrow 1$
6.     **for**  $j \leftarrow (i+1) \dots n$
7.       **if**  $A[i]=A[j]$  **then**
8.          $B[i] \leftarrow B[i]+1$
9.          $B[j] \leftarrow 0$



# Bilinen Probleme İndirgeme

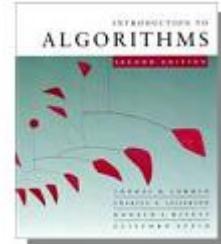
- I. YOL: En kötü çalışma zamanı
- Bütün sayılar ikili olarak karşılaştırılırlar.
- 1. sayı için n-1 karşılaştırma
- 2. sayı için n-2 karşılaştırma
- ....
- (n-1). sayı için 1 karşılaştırma yapılır.
- Bunun sonucunda elde edilen karşılaştırma toplamları
  
- $(n-1)+(n-2)+\dots+1 = \frac{n(n - 1)}{2}$
  
- $T(n)=O(n^2)$  olur.



# Bilinen Probleme İndirgeme

- I. YOL En iyi çalışma zamanı
- Bu mertebe bu algoritmanın en kötü durum analizidir ve en kötü durum A dizisi içindeki bütün sayıların farklı çıkması durumudur. Eğer A dizisi içindeki bütün sayılar aynı iseler, dış döngünün değişkeninin ilk değeri için iç döngü baştan sona kadar çalışır ve ondan sonraki değerler için iç döngü hiç çalışmaz. Bunun sonucunda en iyi durum elde edilir ve en iyi durumun mertebesi  $\Theta(n)$  olur.
- $T(n) = \Theta(n)$
- Bu algoritmadan daha iyisi var mı?

# Bilinen Probleme İndirgeme

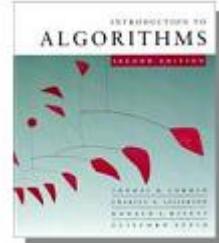


- II. YOL: Bilinen probleme dönüştürme
- Aynı problemin bilinen bir yöntemle çözülmesi daha iyi sonuç verebilir. Bilinen problem sıralama işlemidir. A dizisi içindeki bütün sayılar sıralanır ve ondan sonra birinci elemandan başlanarak sona doğru ardışıl olan elemanlar karşılaştırılır. Bu şekilde kaç tane tekrar olduğu bulunur.

**TekrarBul(A,B)**

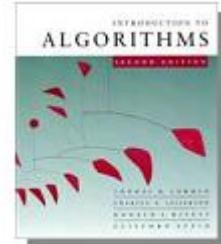
1. **HeapSort(A)**
2. **for**  $j \leftarrow 1 \dots n$
3.      $B[j] \leftarrow 1$
4.      $i \leftarrow 1$
5.     **for**  $j \leftarrow 1 \dots (n-1)$
6.         **if**  $A[j] = A[j+1]$  **then**
7.              $B[i] \leftarrow B[i] + 1$
8.              $B[j+1] \leftarrow 0$
9.         **else**
10.         $i \leftarrow j+1$

# Bilinen Probleme İndirgeme



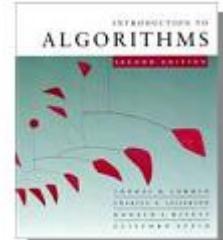
- II. YOL
- Bu algoritma iki kısımdan oluşmaktadır. Birinci kısmı A dizisinin sıralanması ve ikinci kısımda ise bir döngü ile tekrar sayısının bulunması işlemidir. YığınSıralama algoritmasının mertebesinin  $T_1(n)=\Theta(nlgn)$  olduğu daha önceden bilinmektedir ve ikinci kısımda ise bir tane döngü olduğundan, bu kısmın mertebesi  $T_2(n)=\Theta(n)$  olur. Bunun sonucunda algoritmanın zaman bağıntısı  $T(n)$
- $T(n)=T_1(n)+T_2(n)$
- $=\Theta(nlgn)+ \Theta(n)$
- $=\Theta(nlgn)$
- sınıfına ait olur. Dikkat edilirse, ikinci yol ile elde edilen çözüm birinci yol ile elde edilen çözümden daha iyidir. Bilinen probleme indirgeme yapılarak elde edilen algoritma birinci algoritmaya göre daha etkili bir algoritmadır.
- Daha iyisi var mı? Araştırma

# Bilinen Probleme İndirgeme



- **Örnek 2:**
- İki boyutlu bir uzayda  $n$  tane noktadan hangi üç noktanın aynı doğru üzerinde olup olmadığı kontrolü yapmak isteniyor. Bu problemin çözümü için en etkili algoritma nedir?
  
- **I. YOL**
- İlk olarak tasarlanacak olan algoritma klasik mantık olarak bütün nokta ikilileri arasındaki eğimler hesaplanır ve bu eğimler birbiri ile karşılaştırılarak hangi üç noktanın aynı doğru üzerinde olduğu belirlenir. Bu işlemi yapan algoritma;

# Bilinen Probleme İndirgeme

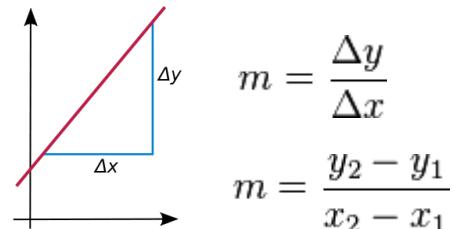


- I. YOL

► A parametresi, her elemanı iki tane gerçek sayıdan oluşan bir iki boyutlu uzay noktaları kümesidir.

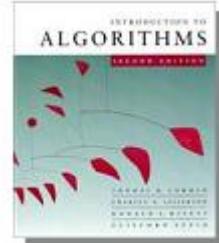
Dogru\_Uz\_Noktalar(A)

1. for  $k \leftarrow 1 \dots n$
2.   for  $j \leftarrow 1 \dots n$
3.     for  $i \leftarrow 1 \dots n$
4.       if  $k \neq j \neq i$  then
5.          $m_1 = \text{eğim}(A[k], A[j])$
6.          $m_2 = \text{eğim}(A[k], A[i])$
7.         if  $m_1 = m_2$  then
8.              $(A[k], A[j])$  ve  $(A[k], A[i])$  noktaları aynı doğru üzerindedir.



- Bu algoritma, iç içe üç tane döngüden oluşmaktadır ve her döngü  $n$  kez çalışmaktadır. En kötü durumda çalışma zamanı
- $T(n) = \Theta(n^3)$  olur.

# Bilinen Probleme İndirgeme

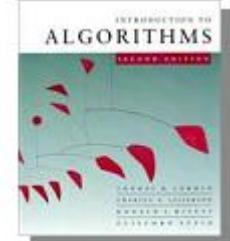


- II. YOL
- İkinci çözüm şeklinde ise, ilk önce oluşabilecek iki nokta arasındaki eğimlerin hepsi hesaplanır. Meydana gelebilecek eğim sayısı n tane noktanın 2' li kombinasyonu olur ve eğim sayısı M olmak üzere

$$M = \binom{n}{2} = \frac{n(n-1)}{2}$$

- olur. Bundan sonraki işlem M tane eğimi sıralamaktır ve ondan sonra M tane elemanlı dizide tekrar eden elemanın olup olmadığı kontrol edilir. Bu işlemleri yapan algoritma ;

# Bilinen Probleme İndirgeme



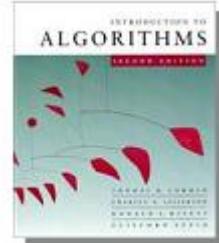
- II. YOL :Bilinen probleme indirgeme

► A parametresi, her elemanı iki tane gerçel sayıdan oluşan bir iki boyutlu uzay noktaları kümesidir. Bu dizideki her eleman çifti arasındaki eğim hesaplanır ve bu eğim B dizisine atılır. Ondan sonra B dizisi sıralanır ve bu dizinin tekrar eden elemanı olup olmadığı kontrol edilir.

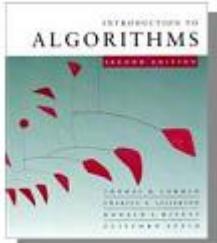
Dogru\_Uz\_Noktalar(A)

1. for  $k \leftarrow 1 \dots n$
2.     for  $j \leftarrow (k+1) \dots n$
3.          $m = \text{eğim}(A[k], A[j])$
4.          $B[i] = m$
5.          $i \leftarrow i + 1$
6.     HeapSort(B,M)
7.     for  $j \leftarrow 1 \dots M$
8.          $C[j] \leftarrow 1$
9.          $i \leftarrow 1$
10.    for  $j \leftarrow 1 \dots M-1$
11.      if  $B[j] = B[j+1]$
12.          $C[i] \leftarrow C[i] + 1$
13.          $C[j+1] \leftarrow 0$
14.      else
15.          $i \leftarrow j + 1$

# Bilinen Probleme İndirgeme

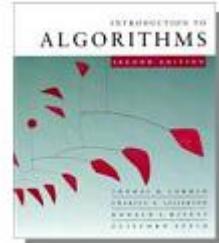


- II. YOL
- C dizisindeki elemanlar kendisi ile aynı endekse sahip B dizisinin o elemanından kaç tane olduğunu tutmaktadır.
- Bu algoritmanın mertebesi hesaplanacak olursa, algoritmada üç parçadan oluşan bir zaman bağıntısı elde edilir.
- İlk parça eğimleri hesaplama zamanı ve bu zaman  $T_1(n)$  olsun.
- İkinci parça B dizisini sıralama zamanı ve bu zaman  $T_2(n)$  olsun.
- Son parçada ise sıralı B dizisi içinde tekrar eden eleman olup olmadığını kontrol etme zamanıdır ve bu zaman  $T_3(n)$  olsun.
- Bu zamanlar
- $T_1(n)=\Theta(n^2)$
- $T_2(n)=\Theta(M \lg M)=\Theta(n^2 \lg n)$
- $T_3(n)=\Theta(M)=\Theta(n^2)$
- Algoritmanın mertebesi  $T(n)=T_1(n)+T_2(n)+T_3(n)=\Theta(n^2 \lg n)$  olur.



# Bilinen Probleme İndirgeme

- **Örnek 3:**  $n \times n$  boyutlarında kare matrislerin çarpımı. Klasik yöntemle  $O(n^3)$  çarpma ve  $O(n^3)$  toplama vardır.
- **Çözüm:** Strassen'in fikri daha önce debynilmiştir.
  
- **Örnek 4:** Bir kümenin maksimum ve minumum elemanlarının belirlenmesi için gerekli algoritmanın kaba kodunu yazınız.



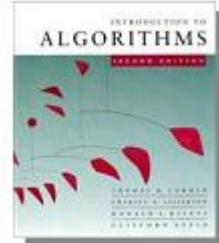
# Bilinen Probleme İndirgeme

- Uygulama çözüm:
- I. Yol: İlk olarak  $n-1$  karşılaştırma yapılarak maksimum bulunur ve  $n-2$  karşılaştırma yapılarak minimum bulunur. Buradan  $T(n)=2n-3$  olur ve Çalışma zamanı  $T(n)=O(n)$  olur.

## MAXIMUM-MINIMUM(A)

```
MAXIMUM-MINIMUM (A)
1 max ← min ← A[1]
2 for i ← 2 to length[A]
3 do if A[i] > max
4 then max ← A[i]
5 else if A[i] < min
6 then min ← A[i]
7 return min & max
```

- Daha iyisi olan bir algoritma tasarlayıp çalışma zamanını bulunuz?



# Bilinen Probleme İndirgeme

- II.Yol Çözüm:
- Eğer  $n$  sayısı çift ise(  $\lg n$  sayısının katı): a) İlk olarak  $n/2$  çift elamanlar bulunur.

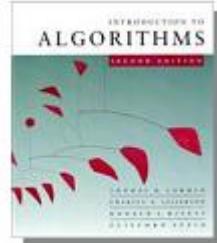


- Daha sonra her bir çift karşılaştırılır.  $\lfloor n/2 \rfloor$ ,  $\lceil n/2 \rceil$ , çiftler arasında en fazla 3 karşılaştırma yapılır.



= larger

= smaller



# Bilinen Probleme İndirgeme

- n çift ise  $T(n)=(3/2)n-2$  olur.

$$T(n) = \begin{cases} 0 & n=1 \\ 1 & n=2 \\ T(\lfloor n/2 \rfloor) + T(\lceil n/2 \rceil) + 2 & \text{olur} \end{cases}$$

(Eğer n sayısı tek ise:  $3(n-1)/2$  olur.)

## Algoritma(S)

```

if |S|=1 or |S|=2 then bir karşılaştırma yapılır
elseif |S|>2 then
 S=S1∪S2
 (min1, max1) ← MaxMin(S1)
 (min2, max2) ← MaxMin(S2)
 if min1<=min2 then sonuç(min=min1)
 else sonuç(min=min2)
 if max1>=max2 then sonuç (max=max1)
 else sonuç (max=max2) T(n)=O(n)

```

n için Sıkı sınır

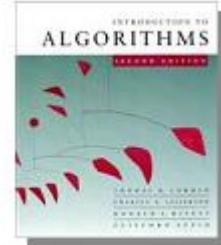
$$T(n) = \lceil 3n/2 \rceil - 2$$

```

MAXMIN(i,j,fmax,fmin)
1 if (i=j)
2 then fmax ← fmin ← a[i]
3 if (i=(j-1)) then
4 if a[i]<a[j]
5 then fmax ← a[j]
6 fmin ← a[i]
7 else fmax ← a[i]
8 fmin ← a[j]
9 else
10 mid ← ⌊(i+j)/2⌋
11 MAXMIN(i,mid,gmax,gmin)
12 MAXMIN(mid+1,j,hmax,hmin)
13 fmax ← max{gmax,hmax}
14 fmin ← min{gmin,hmin}

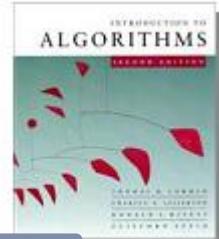
```

# Bilinen Probleme İndirgeme



- **Örnek 4:**
- Bir binanın güvenlik işlemleri kamera tertibatı ile yapılmak isteniyor ve kurulacak olan kamera sistemi, en az sayıda kamera içerecek ve binada görüş alanı dışında da yer kalmayacak şekil olacaktır. Bu problem nasıl çözülür?
- **Çözüm**
- İlk olarak problemin bilinen bir probleme dönüştürülmesi gereklidir. Binada kirişler ve kolonlar ayrı olarak düşünüldüğünde, kiriş ve kolonların birleştiği noktalar da düğüm olarak düşünülebilir. Bu şekilde binanın çizgesi çıkarılmış olur. Binaya yerleştirilecek kameraların görmediği kiriş veya kolon kalmamalıdır. Kiriş ve kolonlar ayrı olduklarına göre çözüm minimum-düğüm kapsama problemiinin çözümü olur. Binayı modelleyen çizge  $G=(V,E)$  olmak üzere problemin çözümü aşağıdaki algoritma ile yapılabilir.
- (Graflara sonra debynilecektir)

# Bilinen Probleme İndirgeme



## ○ Çözüm

▷ C kümesi hangi köşelere kamera konulacaksa, o köşeleri temsil eden düğümleri içerir.

Düğüm\_Kapsama(G)

1.  $C \leftarrow \emptyset$
2.  $E' \leftarrow E$
3.  $E' \neq \emptyset$  olduğu sürece devam et
  4.  $(u,v) \in E'$  olan bir ayrıt seç ve
  5.  $C \leftarrow C \cup \{u,v\}$
  6.  $E'$  kümesinde u veya v düşümüne çakışık olan ayrıtların hepsini sil.

# **8.Hafta**

## **Kıym Fonksiyonu (Hashing),İkili Arama Ağaçları (BST)**

Rastgele yapılanmış ikili  
arama ağaçları

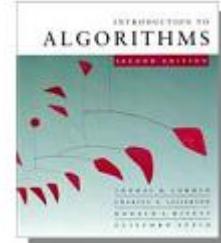
- Beklenen düğüm derinliği
- Yüksekliği çözümlemek

# 8.Hafta

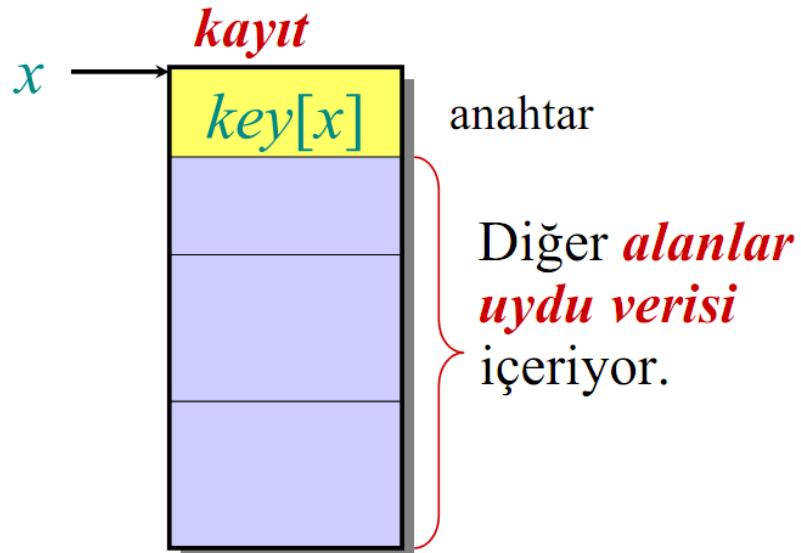
## Kıym Fonksiyonu (Hashing), BST

- Doğrudan erişim tabloları
- Çarpışmaları ilmekleme ile çözmek
- Kıym fonksiyonu seçimi
- Açık adresleme

# Sembol-tablosu problemi



Sembol tablosu  $S$  'nin içinde  $n$  **kayıt** var:

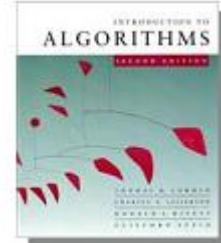


$S$  'deki işlemler:

- ARAYA SOKMA( $S, x$ )
- SILME( $S, x$ )
- ARAMA( $S, k$ )

Veri yapısı  $S$  nasıl organize edilmelidir?

# Doğrudan erişim tablosu



**FIKİR:** Anahtarların  $U \subseteq \{0, 1, \dots, m-1\}$  setinden seçildiğini ve birbirlerinden farklı olduklarını varsayıñ.  $T[0 \dots m-1]$  dizilimini oluşturun:

$$T[k] = \begin{cases} x & \text{eğer } x \in K \text{ ve } \text{key}[x] = k \text{ ise,} \\ 0 & \text{diğer durumlarda.} \end{cases}$$

Burada işlemler  $\Theta(1)$  zamanı alır.

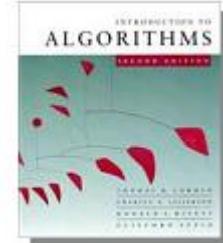
**Problem:** Anahtarların değer kümesi büyük olabilir:

- 64-bit sayılar ( $18,446,744,073,709,551,616$  farklı anahtarları temsil eder),
- (daha da fazla) karakter dizgisini içerebilir.

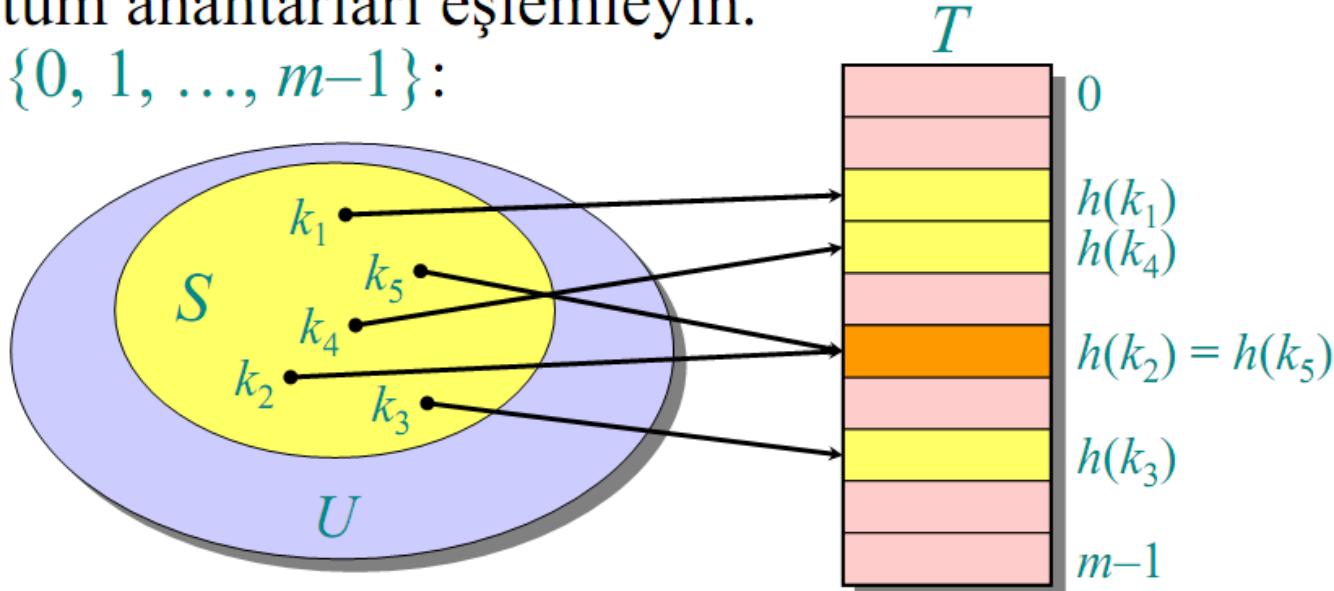
# Çözüm HASHING

- Hashing, elimizdeki veriyi kullanarak o veriden elden geldiği kadar benzersiz bir tamsayı elde etme işlemidir.
- Bu elde edilen tamsayı, dizi şeklinde tutulan verilerin indisleri gibi kullanılarak verilere tek seferde erişmemizi sağlar.

# HASHING (KIYIM FONKSİYONU)

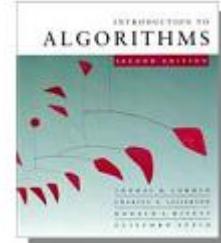


**Çözüm:** *Kiyim fonksiyonu*  $h$  ile  $U$  evrenindeki tüm anahtarları eşlemleyin.  
 $\{0, 1, \dots, m-1\}$ :

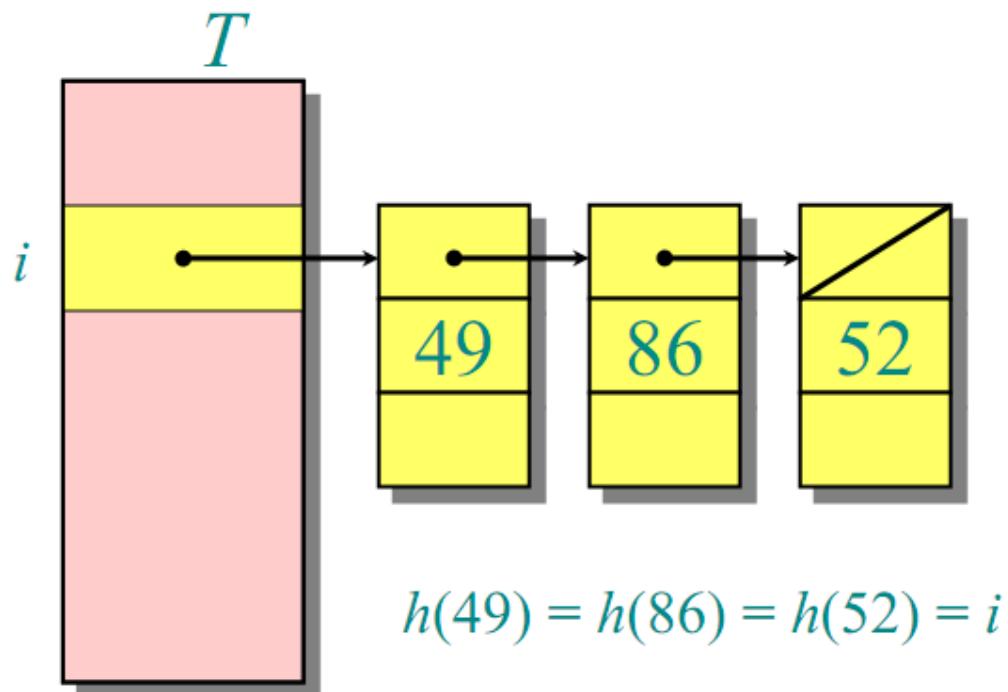


Araya yerleştirilecek kayıt  $T$ ' deki dolu bir yuvaya eşlemlendiğinde, bir **çarpışma** oluşur.

# Çarpışmaları ilmeklemeyle (Chaining) çözme

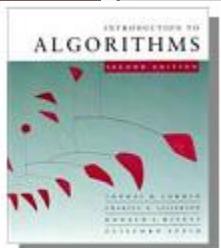


- Aynı yuvadaki kayıtları bir listeye ilişkilendirin.



*En kötü durum:*

- Tüm anahtarlar aynı yuvaya kıyımlanır.
- Erişim süresi =  $\Theta(n)$  eğer  $|S| = n$  ise



# İlmeklemede Ortalama Durum Çözümlemesi

*Basit tekbiçimli kıyımlama için* şu varsayıımı yaparız:

- Her anahtar  $k \in S$ ,  $T$  tablosunun her yuvasına diğer anahtarların nereye kıyımlandığından bağımsız olarak kıyımlanır.

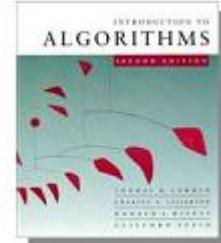
$n$  bu tablodaki anahtarların sayısı ve  $m$  de yuvaların sayısı olsun.

$T$ 'nin *yük oranını* tanımlarken;

$$\alpha = n/m$$

= yuva başına ortalama anahtar sayısıdır.

# Arama maliyeti



Belirli bir anahtar kaydı için **başarısız** bir aramadaki beklenen süre

$$= \Theta(1 + \alpha)$$

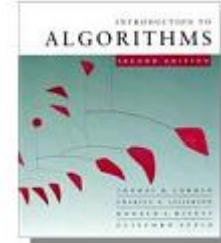
*listeyi  
arama*

*kıym fonksiyonu uygulama  
ve yuvaya erişim*

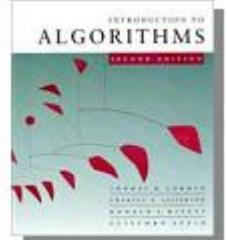
Beklenen arama süresi =  $\Theta(1)$ , eğer  $\alpha = O(1)$ , veya eğer  $n = O(m)$  ise..

**Başarılı** bir arama da aynı asimptotik sınıra sahiptir, ama çok sıkı bir argüman biraz daha karmaşıktır.

# Bir kiyim fonksiyonu seçmek



- Basit tek biçimli kiyimlamanın varsayımini garanti etmek zordur, ama eksikliklerinden kaçınılabildiği sürece pratikte iyi çalışan bazı ortak teknikler vardır.
- **İstenilenler:**
  - İyi bir kiyim fonksiyonu, anahtarları tablonun yuvalarına tek biçimli dağıtabilmelidir.
  - Anahtar dağılımındaki düzenlilik bu tek biçimliliği etkilememelidir.



# Bölme metodu

Tüm anahtarların tam sayı olduğunu kabul edin ve şöyle tanımlayın:  $h(k) = k \bmod (\text{ölçke}) m$ .

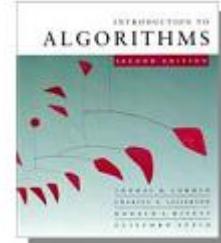
**Sakınca:**  $m'$  yi küçük bir  $d$  böleni olacak şekilde seçmeyin. Anahtarlardan çoğu ölçke (modulo)  $d$  ile çakışırsa, bu durum tekbiçimliliği olumsuz etkiler.

**Uç sakınca:** Eğer  $m = 2^r$  ise, kıyım fonksiyonu  $k'$  nin bütün bitlerine bağımlı bile olmaz:

- Eğer  $k = 1011000111\overbrace{011010}_2$  ve  $r = 6$  ise,  $m=2^6$   
 $h(k) = 011010_2$  dır.  $\quad h(k)$

# Bölme metodu

$$h(k) = k \bmod m.$$



$m$ 'yi , 2 veya 10' un bir kuvveti olmayacak şekilde ve bilgisayar dünyasında yaygın kullanılmayan asal sayılar arasından seçin.

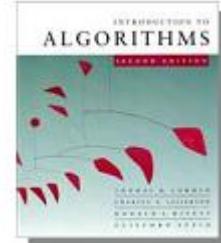
## Rahatsızlık:

- Bazen, tablo boyutunu asal oluşturmak uygun değildir.

Buna rağmen bu metot yaygındır ama bir sonraki metot daha üstündür.

Not:  $m$  çift ve değerlerde çift sayı ise anahtarların hepsi aynı yuvayı işaret eder. Tek sayılı yuvalara hiçbir zaman kiyim olmaz. Yuvaların yarısı boş olur.  $m$  yi asal seçmek daha uygundur ama her zaman değil, asal sayı 2 ve 10 nun kuvvetlerine yakın olmazsa iyidir.

# Çarpma metodu

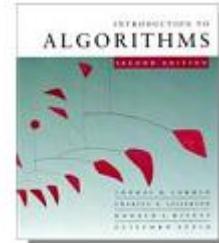


Tüm anahtarların tamsayı  $m = 2^r$ , ve bilgisayar sözcüklerinin de  $w$ -bit olduğunu kabul edin.

$h(k) = (A \cdot k \bmod 2^w)$  rsh  $(w - r)$  'yi tanımlayın, burada rsh “bit bazında sağa kayma” işlemcisi ve  $A$  da  $2^{w-1} < A < 2^w$  aralığında tek tamsayı olsun.

- $A$  'yı  $2^{w-1}$  veya  $2^w$  'ye çok yakın seçmeyin.
- Ölçek (modulo)  $2^w$  ile çarpma bölmeye oranla daha hızlıdır.
- rsh ( bit bazında sağa kaydırma) operatörü hızlıdır.

## Çarpma metodu örneği

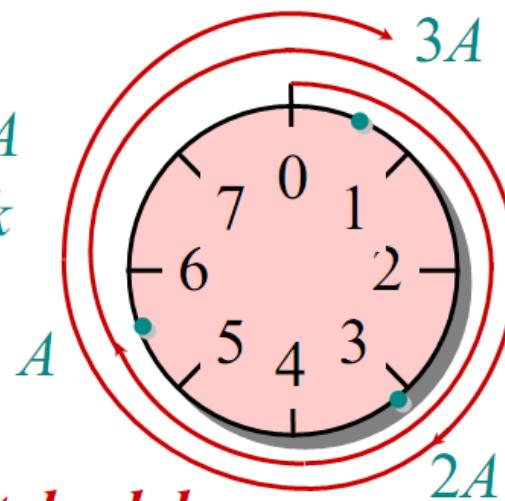


$$h(k) = (A \cdot k \bmod 2^w) \text{ rsh } (w - r)$$

$m = 8 = 2^3$  olsun ve bilgisayarımızda da  $w = 7$ -bit sözcükler olsun:

$$\begin{array}{r}
 & 89 & 1011001 = A \\
 \times & 107 & 1101011 = k \\
 \hline
 9523 & \underbrace{1001010}_{\text{ }} \underbrace{011}_{\text{ }} \underbrace{0011}_{\text{ }} &
 \end{array}$$

mod  $2^w$  alınırsa bu kısım  
ihmal edilir. Düşük değerli  
bitler kalır.

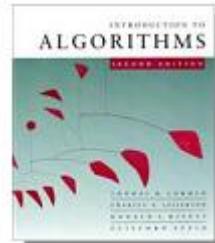


Burada A  
kesirli sayı  
düşünüldü  
(A=11/2)

## *Modüler tekerlek*

Eğer A, örneğin tek sayı ise ve ikinin kuvvetlerinden birine çok yakın değilse, atamayı başka bir yerdeki farklı yuvaya yapar. Böylece etrafta dolaşırken k çok büyük bir değerse, k çarpı A çevrede k kere döner.

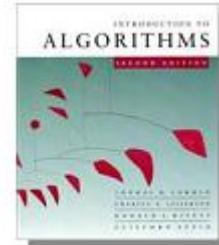
# Çarpma metodu örneği



## Multiplication Method

- Hash function is defined by size plus a parameter  $A$   
$$h_A(k) = \lfloor \text{size} * (k * A \bmod 1) \rfloor \text{ where } 0 < A < 1$$
- Example: size = 10, A = 0.485  
$$\begin{aligned} h_A(50) &= \lfloor 10 * (50 * 0.485 \bmod 1) \rfloor \\ &= \lfloor 10 * (24.25 \bmod 1) \rfloor = \lfloor 10 * 0.25 \rfloor = 2 \end{aligned}$$

# Açık adresleme ile çarşımları çözmek



Kıymı tablosunun dışında depo alanı kullanılmaz.

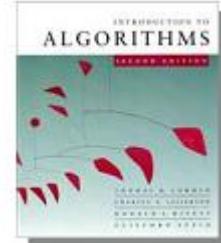
- Araya yerleştirme boş bir yuva bulunana kadar tabloyu sistematik biçimde sondalar.
- Kıymı fonksiyonu hem anahtara hem de sonda sayısına bağlıdır:

$$h : U \times \{0, 1, \dots, m-1\} \rightarrow \{0, 1, \dots, m-1\}.$$

- Sonda dizisi  $\langle h(k,0), h(k,1), \dots, h(k,m-1) \rangle$   $\{0, 1, \dots, m-1\}$ ' in bir permütasyonu olmalıdır.
- Tablo dolabilir ve silme işlemi zordur, (ama imkansız değildir).

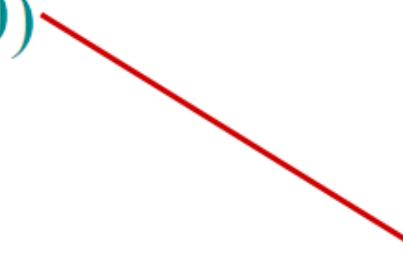
Tablo dolabilir olduğundan  $n \leq m$  olmalıdır. Tablo dolarsa her yerde arama yapmak zorunda kalırız ve aradığımız elemanı bulamayabiliriz.

# Açık adresleme için örnek



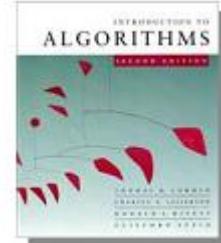
Anahtarı  $k = 496$  araya yerleştirin:  $T$

0. Sonda  $h(496, 0)$



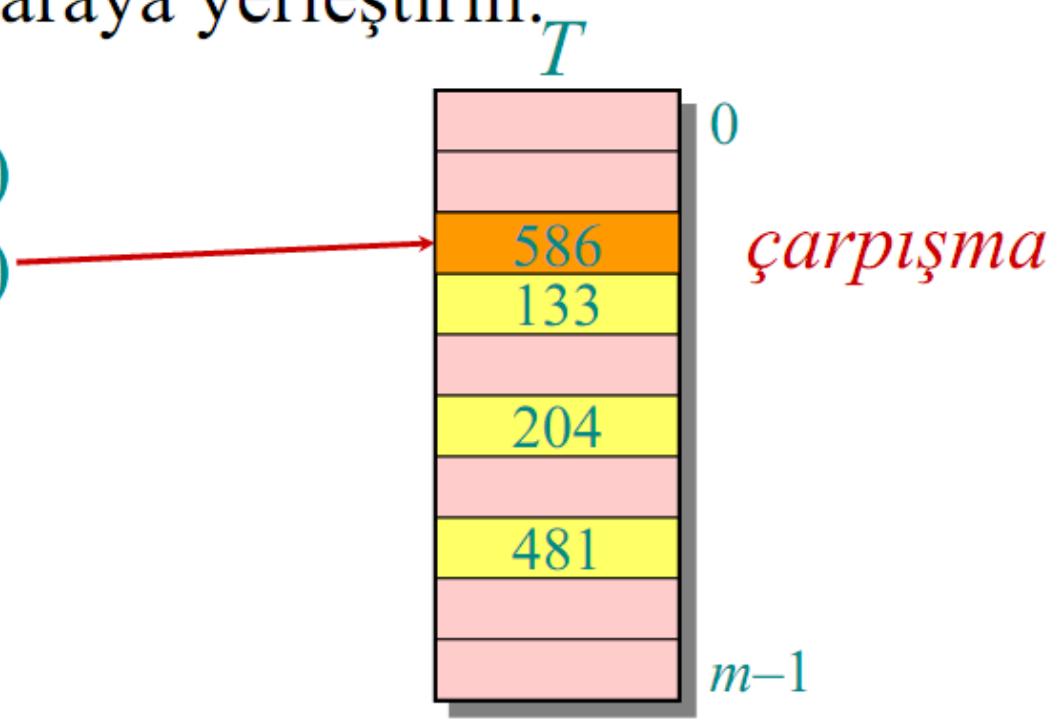
0  
*çarpışma*  
 $m-1$

# Açık adresleme için örnek

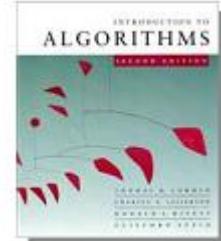


Anahtarı  $k = 496$  araya yerleştirin:

0. Sonda  $h(496, 0)$
1. Sonda  $h(496, 1)$

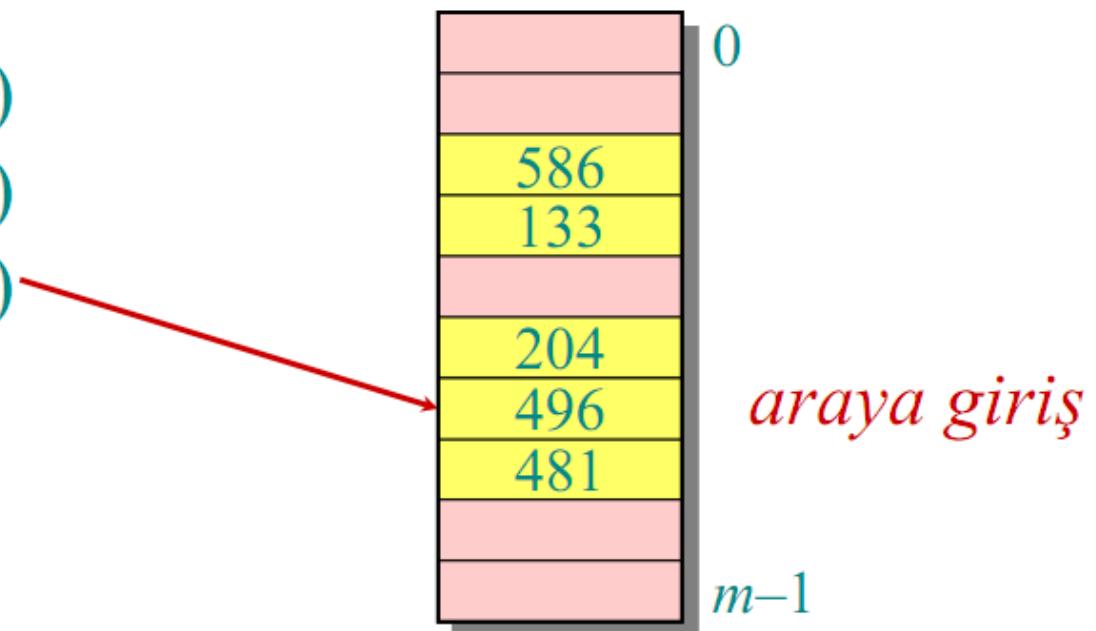


# Açık adresleme için örnek

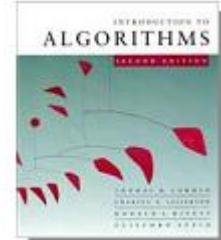


Anahtarı  $k = 496$  araya yerleştirin:

0. Sonda  $h(496,0)$
1. Sonda  $h(496,1)$
2. Sonda  $h(496,2)$

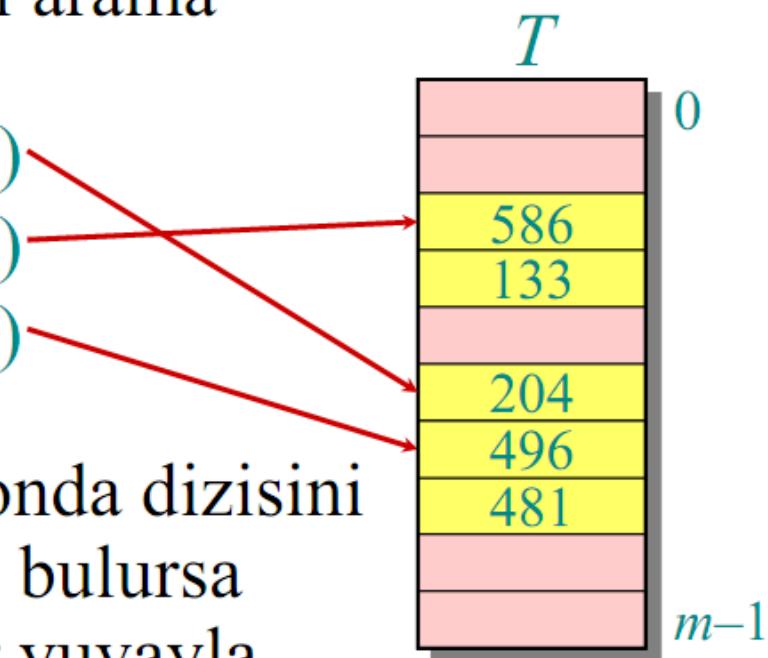


# Açık adresleme için örnek



$k = 496$ : Anahtarı arama

0. Sonda  $h(496, 0)$
1. Sonda  $h(496, 1)$
2. Sonda  $h(496, 2)$



Arama da aynı sonda dizisini kullanır; anahtarı bulursa başarıyla, boş bir yuvaya karşılaşırsa başarısızlıkla sona erer.

## Sondalama (Probing) Stratejileri

- Doğrusal Sondalama (Linear Probing)

- $- h(k,i) = (h'(k) + i) \bmod m \rightarrow h(k,0)$

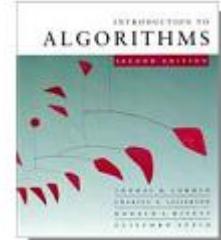
- İkinci Dereceden Sondalama(Quadratic probing)

- $- h(k,i) = (h'(k) + c_1i + c_2i^2) \bmod m$

- Çift Kiyım (Double hashing)

- $- h(k,i) = (h1(k) + i * h2(k)) \bmod m$

# Sonda stratejileri



## Doğrusal sondalama:

$h'(k)$ , gibi basit bir kiyim fonksiyonu verildiğinde, doğrusal sondalama şu kiyim fonksiyonunu kullanır:

$$h(k,i) = (h'(k) + i) \text{ mod } m.$$

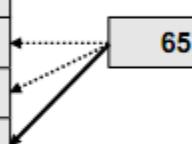
Bu metot basit olmakla birlikte **asal gruplandırma** sıkıntı yaratır; dolu yuvalar uzun sıralar oluşturur ve ortalama arama süresi artar. Ayrıca, dolu yuvaların sıra uzunluğu giderek artar.

# Hash fonksiyonları

## Çakışmanın giderilmesi (Linear Probing)

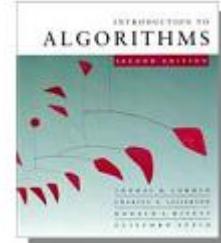
- Aynı pozisyon'a gelen ikinci kayıt ilgili pozisyondan sonraki ilk boş pozisyon'a yerleştirilir.
- Ekleme: Boş bir alan bulunarak yapılır.
- Silme/Erişim: İlk boş alan bulunana kadar devam edebilir.

|    |      |
|----|------|
| 0  | -    |
| 1  | -    |
| 2  | 47   |
| 3  | -    |
| 4  | -    |
| 5  | 35   |
| 6  | 36   |
| 7  | 65   |
| 8  | -    |
| 9  | 129  |
| 10 | 25   |
| 11 | 2501 |
| 12 | -    |
| 13 | -    |
| 14 | -    |



# Sonda stratejileri

## Linear Probing: Example



|   |    |
|---|----|
| 0 |    |
| 1 |    |
| 2 |    |
| 3 | 3  |
| 4 | 13 |
| 5 | 5  |
| 6 | 6  |
| 7 | 23 |
| 8 | 15 |
| 9 |    |

$$h(k,i) = (h'(k) + i) \bmod m$$

Ex:  $m = 10$

Input =  $<5, 3, 6, 13, 23, 15>$

$$h(3,0) = (h'(3) + 0) \bmod 10 = 3$$

$$h(13,0) = (h'(13) + 0) \bmod 10 = 3 ?$$

$$h(13,1) = (h'(13) + 1) \bmod 10 = 4$$

$$h(5,0) = (h'(5) + 0) \bmod 10 = 5$$

$$h(6,0) = (h'(6) + 0) \bmod 10 = 6$$

$$h(23,0) = (h'(23) + 0) \bmod 10 = 3 ?$$

$$h(23,1) = (h'(23) + 1) \bmod 10 = 4 ?$$

---

$$h(23,4) = (h'(23) + 4) \bmod 10 = 7$$

## Hash fonksiyonları

### Çakışmanın giderilmesi (Linear Probing)

- Linear Probing metodunun avantajları / dezavantajları
  
- Bağlı listeler gibi ayrı bir veri yapısına ihtiyaç duyulmaz.
  
- Kayıtların yoğun şeklinde toplanmasına sebep olur.
  
- Silme ve arama işlemleri için gereken zaman aynı hash değeri sayısı arttıkça artar.

## Hash fonksiyonları Çakışmanın giderilmesi (Quadratic Probing)

- Aynı pozisyon'a gelen ikinci kayıt Quadratic Fonksiyonla yerleştirilir.
- En çok kullanılan hash fonksiyonu
- $h(k,i) = (h'(k) + c_1i + c_2i^2) \text{ mod } m$
- Burada  $h'$ , yardımcı hash fonksiyonu,  $c_1$  ve  $c_2 \neq 0$
- ve  $i = 0, 1, \dots, M-1$ .
- Sondalamanın başlangıç posizyonu:  $t = [h'(k)]$
- $h(k,i) = (t + c_1i + c_2i^2) \text{ mod } m$

# Quadratic Probing: Example

|   |    |
|---|----|
| 0 | 12 |
| 1 |    |
| 2 | 2  |
| 3 | 3  |
| 4 |    |
| 5 | 22 |
| 6 |    |
| 7 |    |
| 8 | 18 |
| 9 |    |

$$h(k,i) = (h'(k) + c_1 i + c_2 i^2) \bmod M$$

Ex:  $M = 10$ ,  $c_1 = 2$ ,  $c_2 = 1$

Input =  $<2, 3, 22, 12, 18>$

$$h(2,0) = (h'(2) + 2*0+1*0) \bmod 10 = 2$$

$$h(3,0) = (h'(3) + 2*0+1*0) \bmod 10 = 3$$

$$h(22,0) = (h'(22)+2*0+1*0) \bmod 10 = 2 ?$$

$$h(22,1) = (h'(22)+2*1+1*1) \bmod 10 = 5$$

$$h(12,0) = (h'(12)+2*0+1*0) \bmod 10 = 2 ?$$

$$h(12,1) = (h'(12)+2*1+1*1) \bmod 10 = 5 ?$$

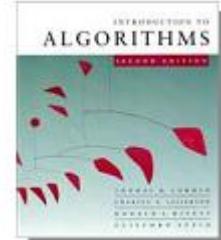
$$h(12,2) = (h'(12)+2*2+1*4) \bmod 10 = 0$$

$$h(18,0) = (h'(18) + 2*0+1*0) \bmod 10 = 8$$

## Hash fonksiyonları Çakışmanın giderilmesi (Quadratic Probing)

- Quadratic Probing metodunun avantajları / dezavantajları
- Anahtar değerlerini linear probing metoduna göre daha düzgün dağıtır.
- Yeni eleman eklenmede tablo boyutuna dikkat edilmezse sonsuza kadar çalışma riski vardır.

# Sonda stratejileri



## Çifte kıymımlama

$h_1(k)$  ve  $h_2(k)$ , gibi iki basit kıymımlama fonksiyonu varsa, çifte kıymımlama şu kıymımlama fonksiyonunu kullanır:

$$h(k,i) = (h_1(k) + i \cdot h_2(k)) \text{ mod } m.$$

Bu metot genelde mükemmel sonuçlar verir, ama  $h_2(k)$ ,  $m$  'e göre asal olmalıdır. Bunun bir yolu  $m$  'yi, 2 'nin bir kuvveti yapmak ve  $h_2(k)$  'yı sadece tek sayılar üretecek şekilde tasarlamaktır.

|    |    |
|----|----|
| 0  |    |
| 1  | 79 |
| 2  |    |
| 3  |    |
| 4  |    |
| 5  | 98 |
| 6  |    |
| 7  |    |
| 8  |    |
| 9  | 14 |
| 10 |    |
| 11 |    |
| 12 |    |

# Double Hashing

$$h(k,i) = (h_1(k) + i * h_2(k)) \bmod M$$

Ya da

- $M=2^d$  ve  $h_2$  çift sayı üretecek şekilde tasarlanabilir
- $M$  asaldır ve  $h_2$ ,  $M$  'den daha küçük pozitif tam sayı üretecek şekilde tasarlanır.

Ex:  $M = 13$ ,  $M' = 11$      $\leftarrow M'$  should be slightly less than  $M$   
 ➔  $h_1(k) = k \bmod M$ ,  $h_2(k) = 1 + (k \bmod M')$ .

Input =  $<98, 79, 14>$

$$h_1(98) = 98 \bmod 13 = 5$$

$$h_1(79) = 79 \bmod 13 = 1$$

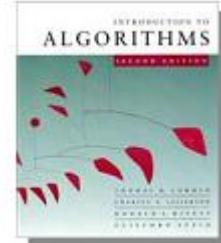
$$h_1(14) = 14 \bmod 13 = 1$$

$$h_2(14) = 1 + 14 \bmod 11 = 4$$

$$h(14, 1) = (h_1(14) + 1 * h_2(14)) \bmod 13 = 1+1*4=5$$

$$h(14, 2) = (1 + 2*4 ) \bmod 13 = 9$$

# Çifte Kıyımlama Teoremin kanıtlanması

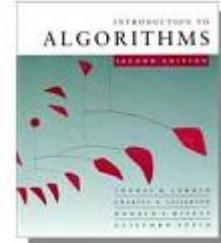


## *Kanıt.*

- En az bir sondalama mutlaka gereklidir.
- $n/m$  olasılığıyla, ilk sonda dolu bir yuvaya gider ve ikinci sondalama gereklili olur.
- $(n-1)/(m-1)$  olasılığıyla, ikinci sonda dolu bir yuvaya gider ve üçüncü sondalama gereklili olur.
- $(n-2)/(m-2)$  olasılığıyla, üçüncü sonda da dolu bir yuvaya gider, v.b.

Gözlemle:  $\frac{n-i}{m-i} < \frac{n}{m} = \alpha$  ;  $i = 1, 2, \dots, n$  için...

# Teoremin kanıtlanması



Bu nedenle, beklenen sonda sayısı:

$$1 + \frac{n}{m} \left( 1 + \frac{n-1}{m-1} \left( 1 + \frac{n-2}{m-2} \left( \dots \left( 1 + \frac{1}{m-n+1} \right) \dots \right) \right) \right)$$

$$\leq 1 + \alpha (1 + \alpha (1 + \alpha (\dots (1 + \alpha) \dots)))$$

$$\leq 1 + \alpha + \alpha^2 + \alpha^3 + \dots$$

$$= \sum_{i=0}^{\infty} \alpha^i$$

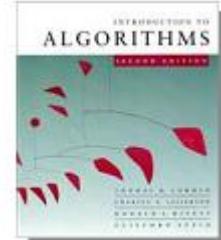
$$= \frac{1}{1 - \alpha}.$$
□

Başlangıçta 1 sondalama olacaktır.  
 $n/m$  çarpışma olacaktır.  
 2.sondada çarpışma olasılığı  $(n-1)/(m-1)$  olacaktır. Böyle devam eder....

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

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

# Teoremin açılımları

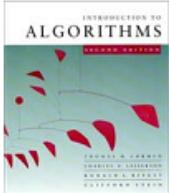


- Eğer  $\alpha$  bir sabitse, açık adresli bir kiyim tablosuna erişim sabit zaman alır.
- Eğer tablo yarı doluysa, beklenen sonda sayısı  $1/(1-0.5) = 2'$  dir.
- Eğer tablo % 90 doluysa, beklenen sonda sayısı  $1/(1-0.9) = 10'$  dur.

## Hash fonksiyonları

### Çakışmanın giderilmesi (Double Hashing)

- Double Hashing metodunun avantajları / dezavantajları
  
- Çok iyi bir kiyim fonksiyonudur
- Anahtar değerlerini linear probing metoduna göre daha düzgün dağıtır ve gruplar oluşmaz.
- Quadratic probing metoduna göre daha yavaştır çünkü ikinci bir hash fonksiyonu hesaplanır.



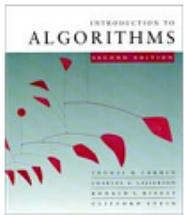
# Kıymı̄m fonksiyonunun bir zaafı

**Problem:** Her kıymı̄m fonksiyonu  $i$  için, kıymı̄m tablosuna ortalama erişim süresini çok büyük ölçüde artıracak bir anahtarlar kümesi vardır.

- Rakibiniz bir  $i$  yuvası için tüm anahtarları  $\{k \in U : h(k) = i\}$ 'den elde edebilir.

**FIKİR:** Kıymı̄m fonksiyonunu tüm anahtarlardan bağımsız olacak şekilde rastgele seçin.

- Rakibiniz kodunuzu görüyor olsa bile, hangi kıymı̄m fonksiyonunun seçileceğini kesinlikle bilmeliğinden, kötü bir anahtarlar kümesi bulamayacaktır.



# Bir evrensel kiyim fonksiyonları setini yapılandırmak

$m$  asal sayı olsun.  $k$  anahtarını  $r + 1$  basamağa ayırtırın; herbirinin set içinde değeri  $\{0, 1, \dots, m-1\}$  olsun. Yani,  $k = \langle k_0, k_1, \dots, k_r \rangle$  ve  $0 \leq k_i < m$  olsun.

## Rastgele yapma stratejisi:

$a = \langle a_0, a_1, \dots, a_r \rangle$  olsun; burada  $a_i \in \{0, 1, \dots, m-1\}$  arasından rastgele seçilmiştir.

Tanım:  $h_a(k) = \sum_{i=0}^r a_i k_i \bmod m$ . **Nokta çarpım, mod  $m$**  (ölçke)

## Evrensel Kiyim

- Good Hashing:
- Universal Hash Function
- • Parameterized by prime size and vector:  
 $a = \langle a_0 \ a_1 \ \dots \ a_r \rangle$  where  $0 \leq a_i < \text{size}$
- • Represent each key as  $r + 1$  integers where  $k_i < \text{size}$ 
  - size = 11, key = 39752 ==>  $\langle 3, 9, 7, 5, 2 \rangle$
  - size = 29, key = “hello world” ==>  
 $\langle 8, 5, 12, 12, 15, 23, 15, 18, 12, 4 \rangle$

$$h_a(k) = \left( \sum_{i=0}^r a_i k_i \right) \bmod \text{size}$$

# Universal Hash Function: Example

Context: hash strings of length 3 in a table of size 131

let  $a = \langle 35, 100, 21 \rangle$

$$\begin{aligned} h_a("xyz") &= (35 * 120 + 100 * 121 + 21 * 122) \% 131 \\ &= 129 \end{aligned}$$

Dezavantajı:  $k_i$  değeri tablo boyutundan büyük olabilir.  
Bu yüzden tablo boyutunu  $k_i$  değerinden büyük seçilmeli

## Mükemmel Kiyım

- Şu ana kadar yaptıklarımız beklenen zamanda başarımla ilgiliydi. Kiyım, beklenen süre bağlamında iyi bir uygulama. **Mükemmel kiyım** ise şu sorulara ilgilenir: Farz edin ki size bir anahtar kümlesi verildi ve bana statik bir tablo oluşturmanız istendi. Böylece en kötü zamanda tabloda anahtarı arayabileyim.
- Bir iyi birde en kötü zamanda. Dolayısıyla elimde sabit bir anahtar kümlesi var. Aynı İngilizcedeki en sık kullanılan 100 veya 1000 sözcük gibi bir şey.

## Mükemmel Kiyım

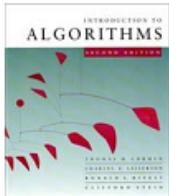
- Bir sözcük ele alındığında, sözcüğün İngilizcede sık kullanılıp kullanılmadığına tabloya bakarak hızlı bir şekilde anlamalıyız. Bu işi beklenen başarıyla değil de garantilenmiş en kötü durum zamanında yapabilmeliyiz.
- Problem şu; verilen  $n$  adet anahtar için statik bir kiyım tablosu yaratmak. Diğer bir deyişle, yeni girdi veya silme yapılmayacak. Sadece elemanları oraya koyacağiz. Büyüklüğü ise,  $m = O(n)$ .
- $m = O(n)$  boyutunda bir tablo ve en kötü durumda arama  $O(1)$  zamanı alacak. Ortalama durumu biliyor olacağız, bu çok zor değil, ama en kötü durumda değerlerin yiğilip, fazla zaman kaybına neden olacağı bir nokta olmayacağından emin olmalıyız. Herhangi bir noktada bu olmamalı; her bir arama  $O(1)$  zamanında olmalı.

## Mükemmel Kiyım

- Buradaki fikir iki aşamalı bir veri tanımlaması yapmaktadır. Fikir, kiyım yapmak; bir kiyım tablomuz olacak, yuvalara kiyım yapacağız, ancak zincirleme işlemini kullanmak yerine ikinci bir kiyım tablosu daha olacak. İkinci tabloya ikinci bir kiyım daha yapacağız. Ve buradaki fikir ikinci düzeyde hiç çarışma olmadan kiyım yapmak.
- Dolayısıyla birinci düzeyde çarışma olabilir. Birinci tabloda çarışan her şeyi ikinci düzeydeki tabloya koyacağız, ama bu tabloda çarışma olmayacak.
- Dolayısıyla evrensel bir kiyım fonksiyonu bulalım. Rastgele bir fonksiyon seçiyoruz. Yapacağımız bu düzeye yani ilk düzeye kiyım yapmak.

## Mükemmel Kiyım

- Bundan sonra iki şeyi takip edeceğiz. Birincisi, diğer düzeydeki kiyım tablomuzun büyülüğü. Bu durumda, kiyım tablomuzun büyülüüğünü yuva sayısıyla adlandıracağız. Örneğin 1. düzeyde kiyım fonksiyonu 1. yuvaya sondalansın ve değeri 4 olsun. İkinci düzey içinse farklı bir kiyım anahtarı kullanacağız.
- Dolayısıyla, ikinci düzeyde her yuhanın farklı bir kiyım fonksiyonu olacak. Mesela, bir yuva rastgele seçilmiş 31 değerini taşıyabilir. Sonra, kiyım tablosuna bir işaretçi koyayım; buna büyük S1 diyeyim. Bu 4 yuvaya sahip olacak ve 14 ile 27'yi saklayacak. Bu
- $h(14) = h(27)$  o da 1'e eşit. Çünkü birinci yuvadayız. Şimdi bu ikisi birinci düzeyde kiyım tablosunda aynı yuvaya kiyiliyor. Bu birinci düzeyde..

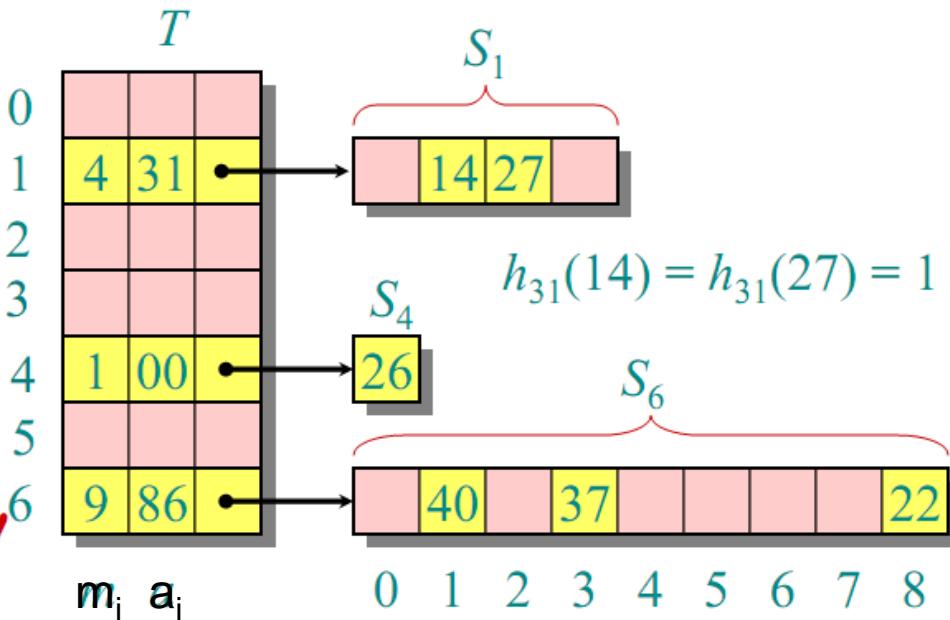


# Mükemmel kiyim fonksiyonu

$n$  anahtarlı bir set verilirse, bir statik kiyim tablosunu boyutu  $m = O(n)$  olacak şekilde yapılandırın ve ARAMA (SEARCH) *en kötü durumda*  $\Theta(1)$  süre alsın.

**FİKİR:** Her iki düzeyde de evrensel kiyim ile 2 düzeyli veri tanımlama.

*2. düzey çarpışması yok!*



## Mükemmel Kiyım

- Buradaki de ikinci düzey. Yani 14 ve 27 birinci seviyede çarpıştılar ve aynı yuvaya gittiler. Ancak ikinci seviyede farklı yuvalara kiyıldılar. Seçtiğim kiyım fonksyonu seçtiğim rastgele sayılarla göre anahtar listesi oluşturarak bu yapıyı yarattı.
- İkinci düzeyde,  $h_{31}(14)$  sayısı için, 1'e eşit ve  $h_{31}(27)$  sayısı için 2 değerlerini aldı.
- Eğer kiyım tablosunun  $i$ . yuvasına kiyılan  $n_i$  tane elaman varsa, ikinci düzeydeki tabloda  $m_i$  sayıda yuva kullanırız ve burada  $m_i$ ,  $n_i$  'nin karesi kadar yuvaya eşit olarak seçilir.
- Örnek olarak, 2 elemanım varsa 4 büyülüüğünde bir kiyım tablom olur. 3 elemanım varsa 9 yuvalı bir kiyım tablosuna ihtiyacım olur.

## Mükemmel Kiyım

- Örnek :  $K=\{10,22,37,40,52,60,70,72,75\}$  9 elamanlı bir anahtar kümesi mod yani  $m=n=9$  olur. Hash fonksiyonumuz:  $h(k)=((a*k+b) \text{ mod } p) \text{ mod } m$
- $a=3$ ,  $b=42$ ,  $p =101$ ,  $m=9$  (a ve b değerleri 0-101 arasında rastgele üretilen sayılar) Öncelikle ilk kiyım tablomuzda çakışmaların sayısını bulalım
- $h(10)=0$       0. ve 5. indiste 1 çakışma
- $h(60)=2$       2. indiste 3 çakışma
- $h(72)=2$       7. indiste 4 çakışma
- $h(75)=2$
- $h(70)=5$
- $h(22)=7$
- $h(37)=7$
- $h(40)=7$
- $h(52)=7$

|   | $n_i$ |
|---|-------|
| 0 | 1     |
| 1 |       |
| 2 | 3     |
| 3 |       |
| 4 |       |
| 5 | 1     |
| 6 |       |
| 7 | 4     |
| 8 |       |

## Mükemmel Kiyım

- Çakışmaları bulduktan sonra tek çakışmaya sahip değerler için  $a_i$  ve  $b_i$  değerlerini 0, diğerleri için ise 0-p arasında random seçelim, 2.kiyım ( $S_i$ ) tablosunun büyülüğu ise  $m_i = n_i^2$  olacak
- 2.kiyım fonksiyonunda çakışma olmayacak şekilde yapılandırıralım
- $h_i(k) = ((a_i * k + b_i) \bmod p) \bmod m_i$

|   | $n_i$ | $m_i$ | $a_i$ | $b_i$ |
|---|-------|-------|-------|-------|
| 0 | 1     | 1     | 0     | 0     |
| 1 |       |       |       |       |
| 2 | 3     | 9     | 10    | 18    |
| 3 |       |       |       |       |
| 4 |       |       |       |       |
| 5 | 1     | 1     | 0     | 0     |
| 6 |       |       |       |       |
| 7 | 4     | 16    | 23    | 88    |
| 8 |       |       |       |       |

|       |
|-------|
| 10    |
| 0     |
| $S_0$ |

|       |    |    |   |   |   |   |   |   |
|-------|----|----|---|---|---|---|---|---|
| 60    | 72 | 75 |   |   |   |   |   |   |
| 0     | 1  | 2  | 3 | 4 | 5 | 6 | 7 | 8 |
| $S_2$ |    |    |   |   |   |   |   |   |

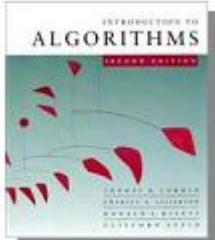
|       |
|-------|
| 70    |
| 0     |
| $S_5$ |

|       |    |    |    |   |   |   |   |   |   |    |    |    |    |    |    |
|-------|----|----|----|---|---|---|---|---|---|----|----|----|----|----|----|
| 40    | 52 | 22 | 37 |   |   |   |   |   |   |    |    |    |    |    |    |
| 0     | 1  | 2  | 3  | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $S_7$ |    |    |    |   |   |   |   |   |   |    |    |    |    |    |    |

# İkili Arama Ağaçları (BST)

Rastgele yapılanmış ikili  
arama ağaçları

- Beklenen düğüm derinliği
- Yüksekliği çözümlemek



# İkili-arama-ağacı sıralaması

$T \leftarrow \emptyset$        $i = 1$  den  $n'$  ye kadar değiştiğinde,

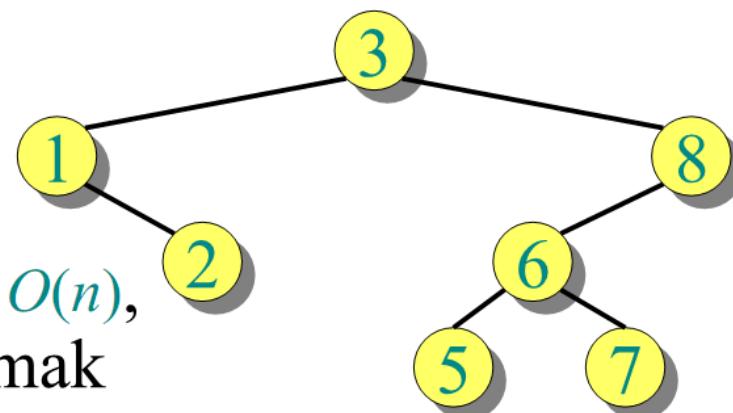
- ▷ Boş bir BST(ikili arama ağacı) yarat.

AĞAÇ ARAYA YERLEŞTİRMEŞİ YAP ( $T, A[i]$ )

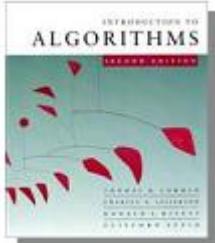
$T$ 'nin içinde sıralı adımlama yap.

## Örnek:

$A = [3 \ 1 \ 8 \ 2 \ 6 \ 7 \ 5]$

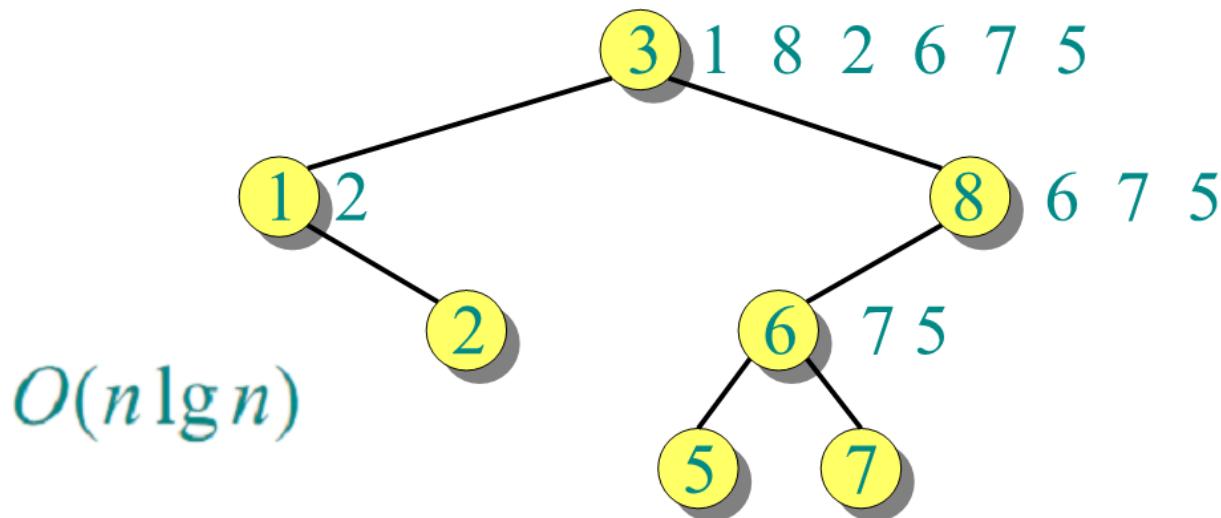


Ağaç adımlama süresi =  $O(n)$ ,  
ancak BST'yi oluşturmak  
ne kadar zaman alır?

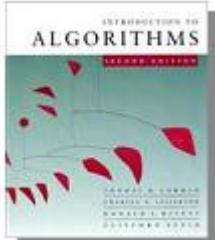


# BST sıralaması çözümlemesi

BST sıralaması çabuk sıralama karşılaştırmalarının aynısını, başka bir düzende yapar!



Ağacı oluşturanın beklenen süresi asimptotik olarak çabuk sıralamanın koşma süresinin aynıdır.



# Düğüm derinliği

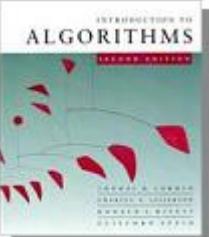
Bir düğüm derinliği = AĞAÇ ARAYA YERLEŞTİRMESİ için yapılan karşılaştırmalar. Tüm girdi permütasyonları eşit olasılıklı varsayılırsa:

Ortalama düğüm derinliği

$$= \frac{1}{n} E \left[ \sum_{i=1}^n \text{(Boğum } i \text{ yi araya yerleştirmek için gerekli karşılaştırmaların sayısı)} \right]$$

$$= \frac{1}{n} O(n \lg n) \quad (\text{Çabuk sıralama analizi})$$

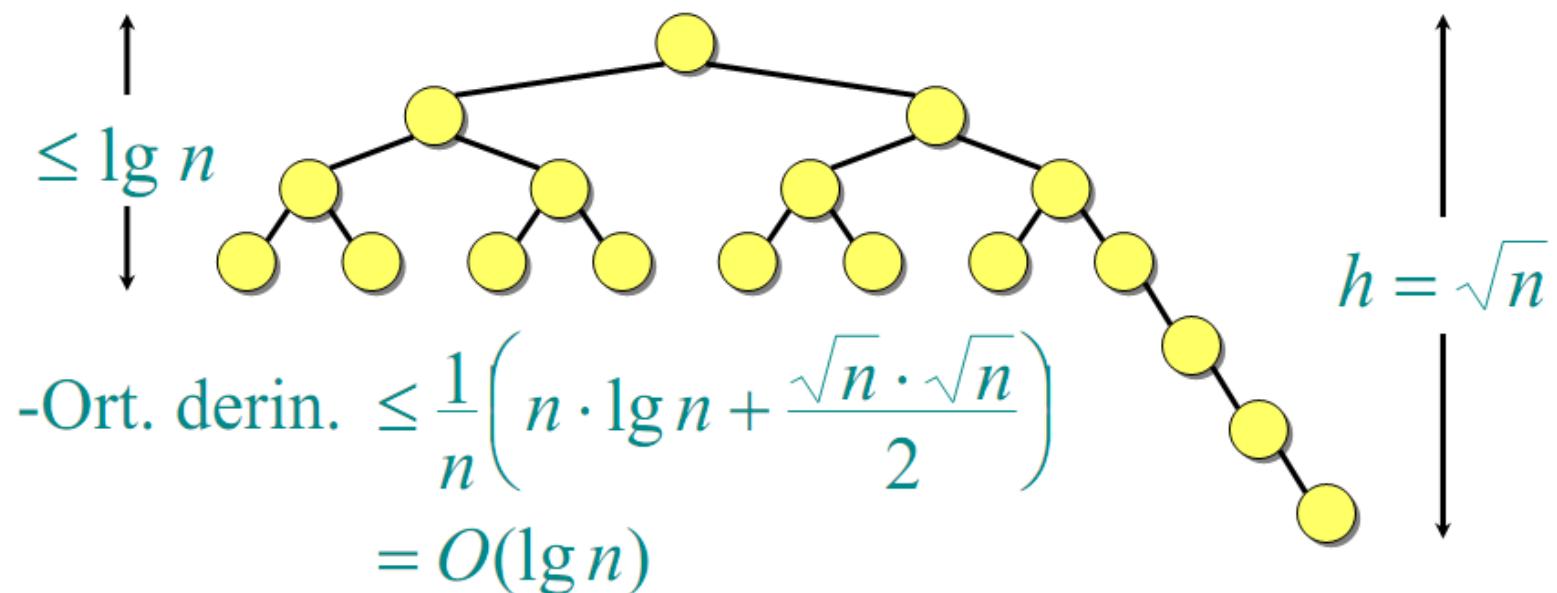
$$= O(\lg n) .$$



# Ağacın beklenen yüksekliği

Ama, ortalama düğüm derinliğinin rastgele yapılanmış bir ikili arama ağacında (BST) =  $O(\lg n)$  olması ağacın beklenen yüksekliğinin de  $O(\lg n)$  olduğu anlamına gelmeyebilir (buna rağmen öyledir).

## Örnek.



## Dengeli arama ağaçları

***Dengeli arama ağaçları:***  $n$  elemanlı bir değişken kümede işlem yaparken  $O(\lg n)$  yüksekliğinin garanti edildiği bir arama ağaçları veri yapısı.

- Örnekler:**
- AVL ağaçları
  - 2-3 ağaçları
  - 2-3-4 ağaçları
  - B-ağaçları
  - Kırmızı-siyah ağaçlar

# 9.Hafta

# Dengeli Arama

# Ağaçları

# (Red - Black Tree)

- Kırmızı-siyah ağaçlar
- Kırmızı-siyah ağacın yüksekliği
- Rotation / Dönme
- Insertion / araya yerleştirme

# 9.Hafta

# Dengeli Arama

# Ağaçları

# (Red - Black Tree)

- Kırmızı-siyah ağaçlar
- Kırmızı-siyah ağacın yüksekliği
- Rotation / Dönme
- Insertion / araya yerleştirme

## Dengeli arama ağaçları

***Dengeli arama ağaçları:***  $n$  elemanlı bir değişken kümede işlem yaparken  $O(\lg n)$  yüksekliğinin garanti edildiği bir arama ağaçları veri yapısı.

- AVL ağaçları
- 2-3 ağaçları
- 2-3-4 ağaçları
- B-ağaçları
- Kırmızı-siyah ağaçlar

# Red - Black Tree

- Kırmızı-siyah ağaç bilgisayar biliminde bir çeşit kendini-dengeleyen ikili arama ağaçları veri yapısıdır.
- Orijinali ilk olarak 1972 yılında yapıyı "simetrik ikili B-ağaçları" olarak adlandıran Rudolf Bayer tarafından bulunmuştur. Bugünkü ismini 1978 yılında Leo J. Guibas ve Robert Sedgewick tarafından yayımlanan bir makaleyle almıştır.
- Karmaşık ancak çalışma süresi en kötü durumda bile iyi ve pratikte verimlidir:  $O(\log n)$  ( $n$  ağaçtaki eleman sayısını gösterir) zamanda arama, ekleme ve çıkarma işlemleri yapabilir.
- Bir kırmızı-siyah ağaç, bilgisayar biliminde karşılaştırılabilir veri parçalarını (sayılar gibi) organize etmek için kullanılabilen özel bir ikili ağaç türüdür.

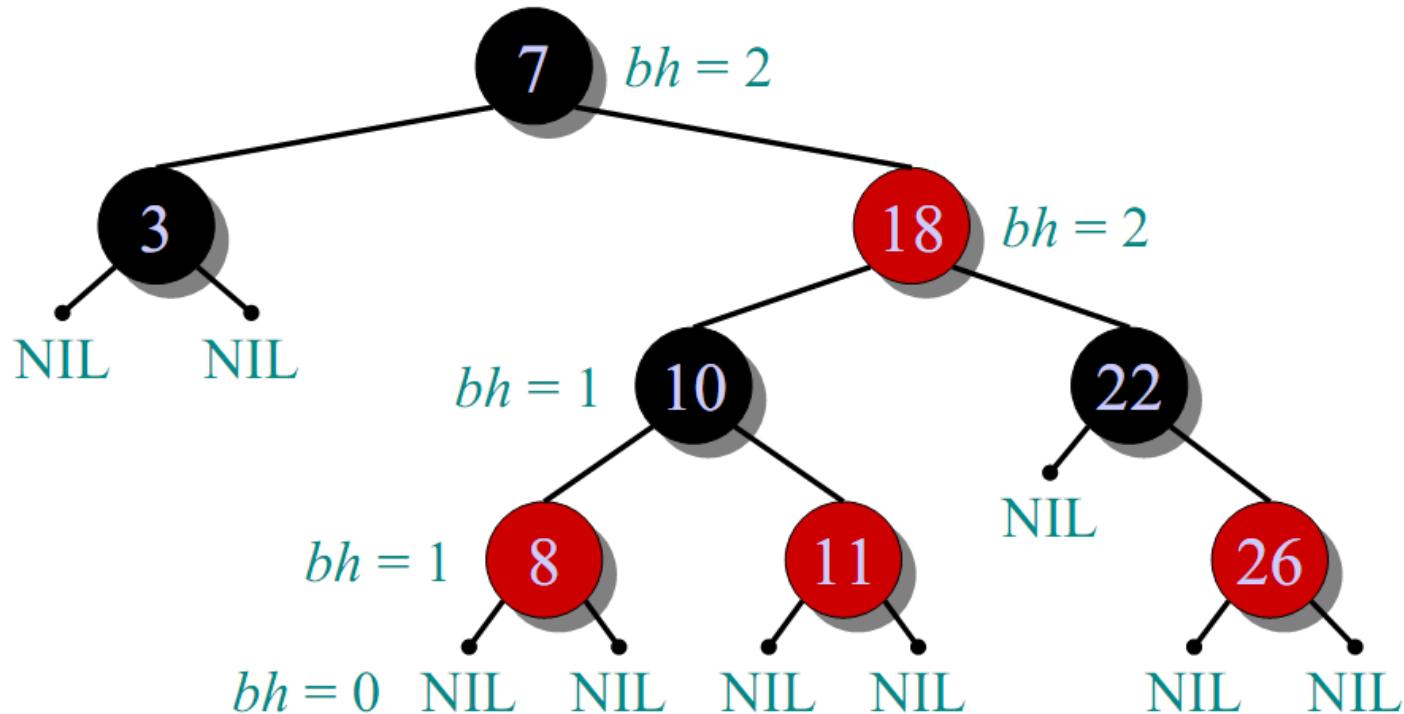
# Red - Black Tree

Bu veri yapısının her düğümünde bir-bitlik renk alanına ihtiyaç vardır.

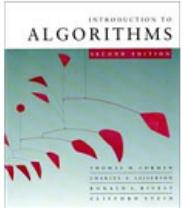
## *Kırmızı-siyah özellikler:*

1. Her düğüm ya kırmızı ya da siyahtır.
2. Kök ve yapraklar (**NIL**'ler yani sıfır'lar) siyahtır.
3. Eğer bir düğüm kırmızı ise, atası siyahtır.
4. Herhangi bir  $x$  düğümünden ardıl yaprağa giden basit yollarda aynı sayıda siyah düğüm vardır  
= **black-height( $x$ )** yani **siyah-yükseklik( $x$ )**.

# Bir kırmızı-siyah ağaç örneği



Herhangi bir  $x$  düğümünden ardıl yaprağa giden basit yollarda aynı sayıda siyah düğüm vardır     $bh = \text{siyah-yükseklik}(x)$ .



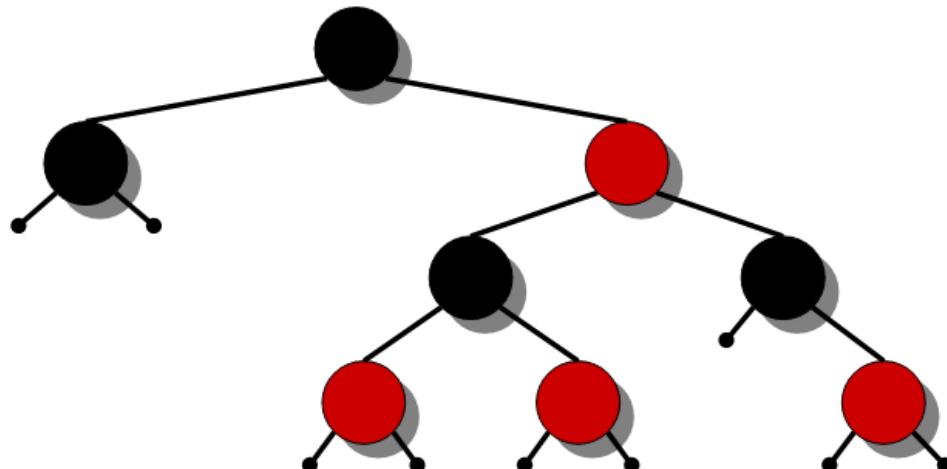
# Kırmızı-siyah ağacın yüksekliği

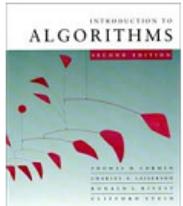
**Teorem.**  $n$  anahtarlı bir kırmızı-siyah ağacın yüksekliği  $h \leq 2 \lg(n + 1)$  dir.

*Kanıt.*

## SEZGİ YÖNTEMİ:

- Kırmızı  
düğümleri  
siyah atalarına  
yaklaştırın.





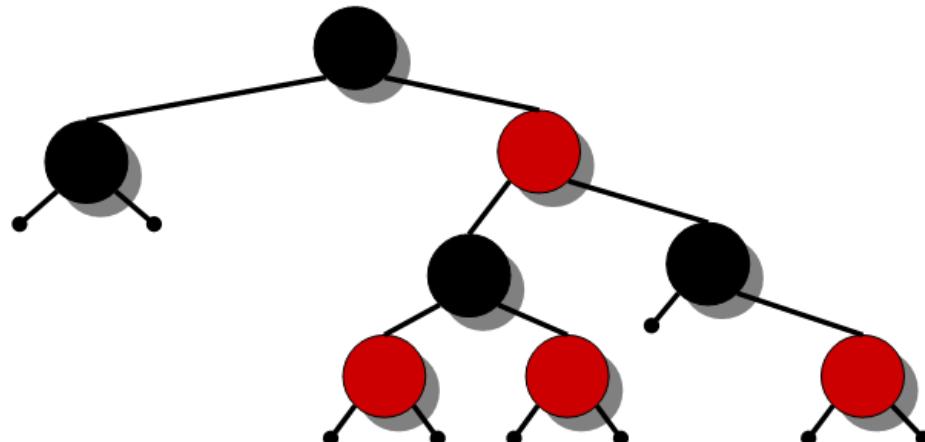
# Kırmızı-siyah ağacın yüksekliği

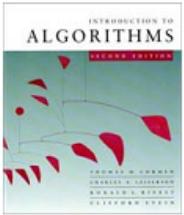
**Teorem.**  $n$  anahtarlı bir kırmızı-siyah ağacın yüksekliği  $h \leq 2 \lg(n + 1)$  dir.

*Kanıt*

**SEZGİ YÖNTEMİ:**

- Kırmızı düğümleri siyah atalarına yaklaştırin.





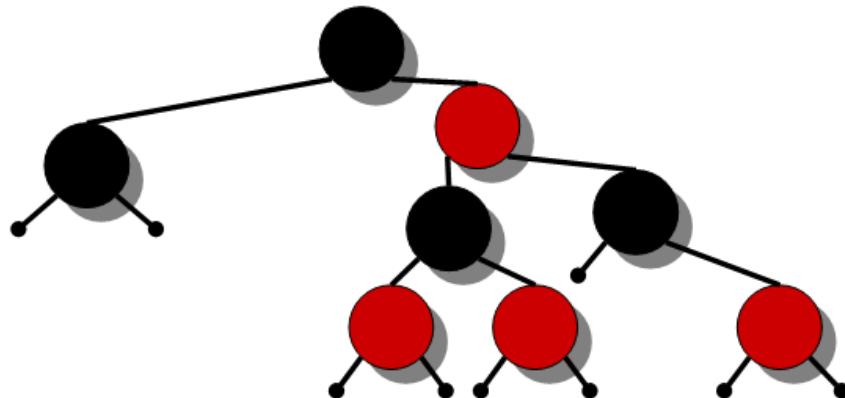
# Kırmızı-siyah ağacın yüksekliği

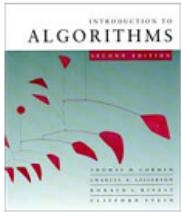
**Teorem.**  $n$  anahtarlı bir kırmızı-siyah ağacın yüksekliği  $h \leq 2 \lg(n + 1)$  dir.

*Kanıt.*

**SEZGI YÖNTEMİ:**

- Kırmızı düğümleri siyah atalarına yaklaştırın.





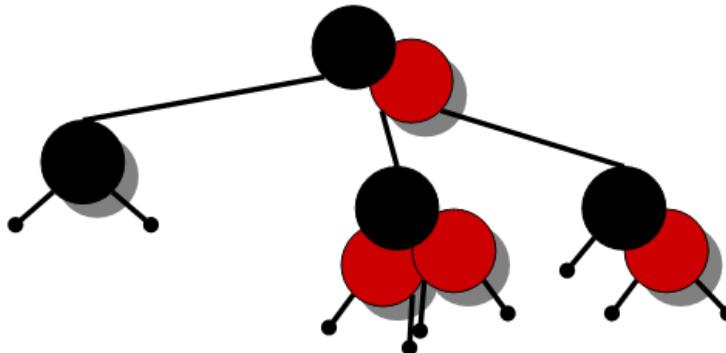
# Kırmızı-siyah ağacın yüksekliği

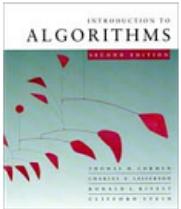
**Teorem.**  $n$  anahtarlı bir kırmızı-siyah ağacın yüksekliği  $h \leq 2 \lg(n + 1)$  dir.

*Kanıt.*

**SEZGI YÖNTEMİ:**

- Kırmızı düğümleri siyah atalarıyla birleştirin.





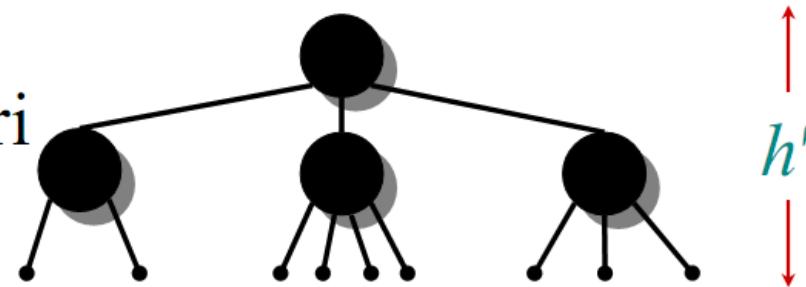
# Kırmızı-siyah ağacın yüksekliği

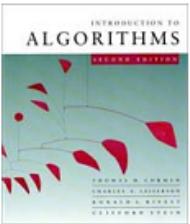
**Teorem.**  $n$  anahtarlı bir kırmızı-siyah ağacın yüksekliği  $h \leq 2 \lg(n + 1)$  dir.

*Kanıt.* (Kitap tümevarımı kullanıyor. Dikkatle okuyun.)

## SEZGI YÖNTEMİ:

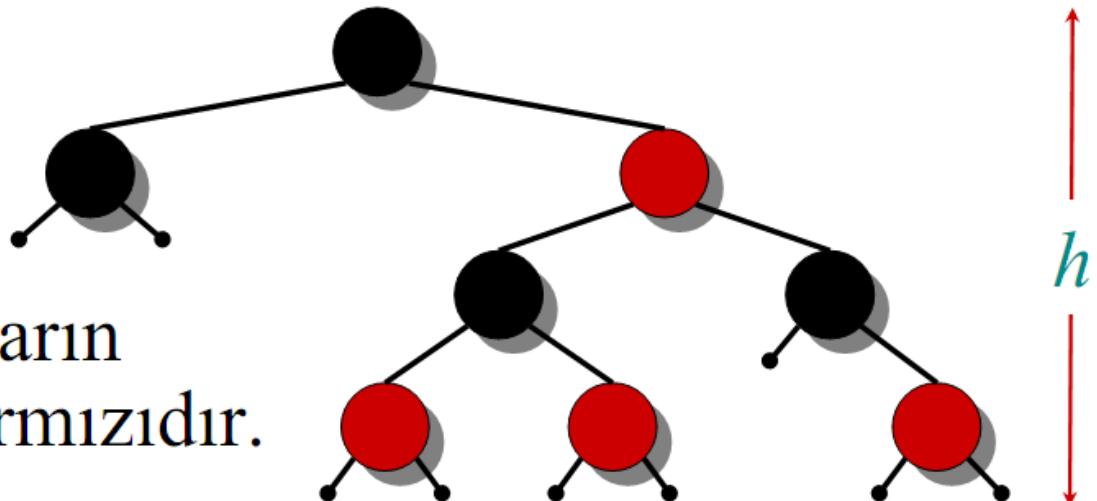
- Kırmızı düğümleri siyah atalarıyla bütünleştirin.
- Bu işlem sonucunda oluşan ağacın her düğümünün 2, 3, ya da 4 ardılı olur.
- 2-3-4 ağacının yapraklarının derinliği  $h'$  tekbiçimlidir.



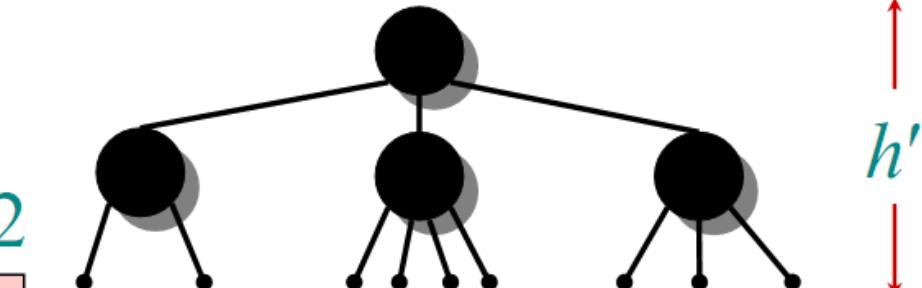


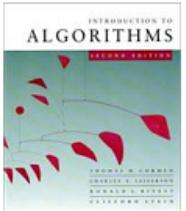
# Kanıtlama (devamı)

- Elimizde  $h' \geq h/2$  olur, çünkü her yoldaki yaprakların en çok yarısı kırmızıdır.



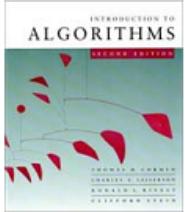
- Her ağaçtaki yaprakların sayısı:  $n + 1$   
 $\Rightarrow n + 1 \geq 2^{h'}$   
 $\Rightarrow \lg(n + 1) \geq h' \geq h/2$   
 $\Rightarrow h \leq 2 \lg(n + 1)$ .





# Sorgulama İşlemleri

**Corollary (Doğal sonuç).**  $n$  düğümlü bir kırmızı-siyah ağaçta SEARCH (ARAMA), MIN, MAX, SUCCESSOR (ARDIL) ve PREDECESSOR (ATA) sorgulamalarının hepsi  $O(\lg n)$  süresinde çalışırlar.

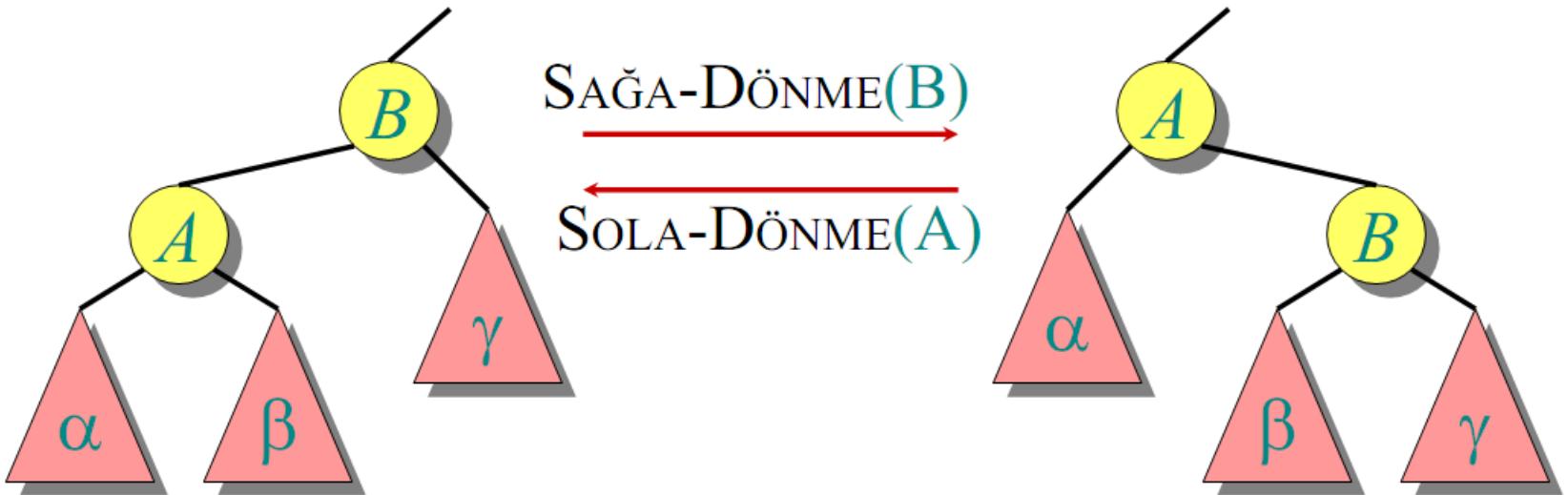


# Değiştirme işlemleri

INSERT (ARAYA YERLEŞTİRME) ve DELETE (SİLME) işlemleri kırmızı-siyah ağaçta değişime neden olur:

- işlemin kendi yapısı,
- renk değişimleri,
- ağacın bağlantılarının “*rotations/rotasyonlar*” yordamıyla yeniden yapılanması.

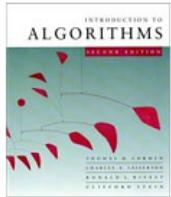
# Rotasyonlar / Dönmeler



Rotasyonlar anahtarların sıralı düzenini korurlar:

- $a \in \alpha, b \in \beta, c \in \gamma \Rightarrow a \leq A \leq b \leq B \leq c.$

Bir rotasyon  $O(1)$  sürede yapılabilir.



# Grafik simgelem

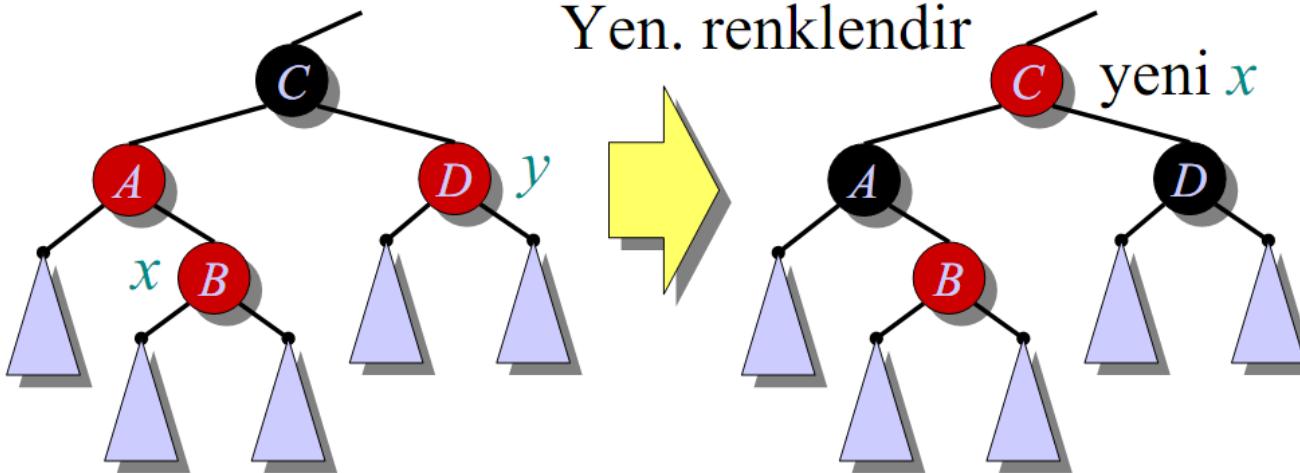


siyah kökü olan bir altağacı tanımlasın.



'ın tümünün siyah-yükseklikleri aynıdır.

# Durum 1:

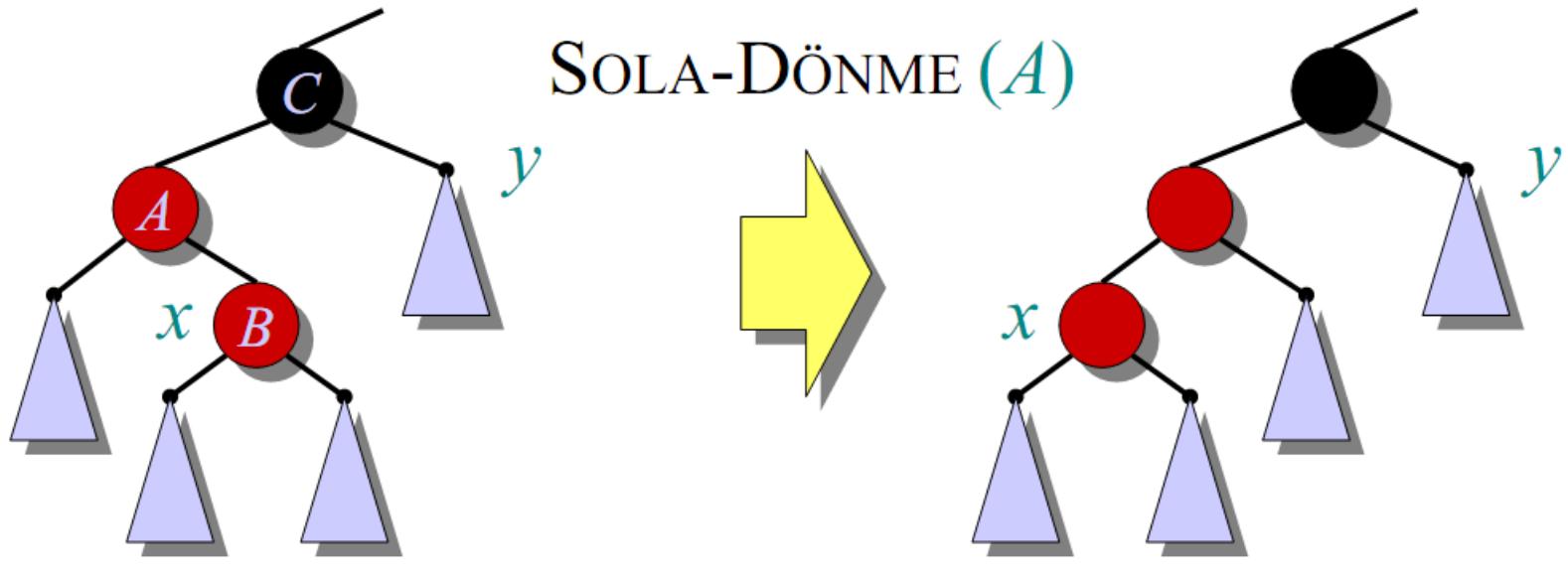


(veya,  $A$ 'nın ardılları  
yer değiştirir.)

$C$  'nin siyahını  $A$  ve  $D$  'ye  
doğru itin ve özyineleme  
yapın, çünkü  $C$ 'nin atası  
kırmızı olabilir.

- Özellik 3 bozuldu. Kırmızı düğümün çocukları siyah olmak zorunda.  $C$  düğümünün çocukların ikisi de kırmızı olduğundan döndürme işlemi yapılmadan yeniden renklendirilecek.  $C$  kırmızı ve çocukları siyah. (Eğer  $C$  kök olsaydı o da siyah olacaktı)

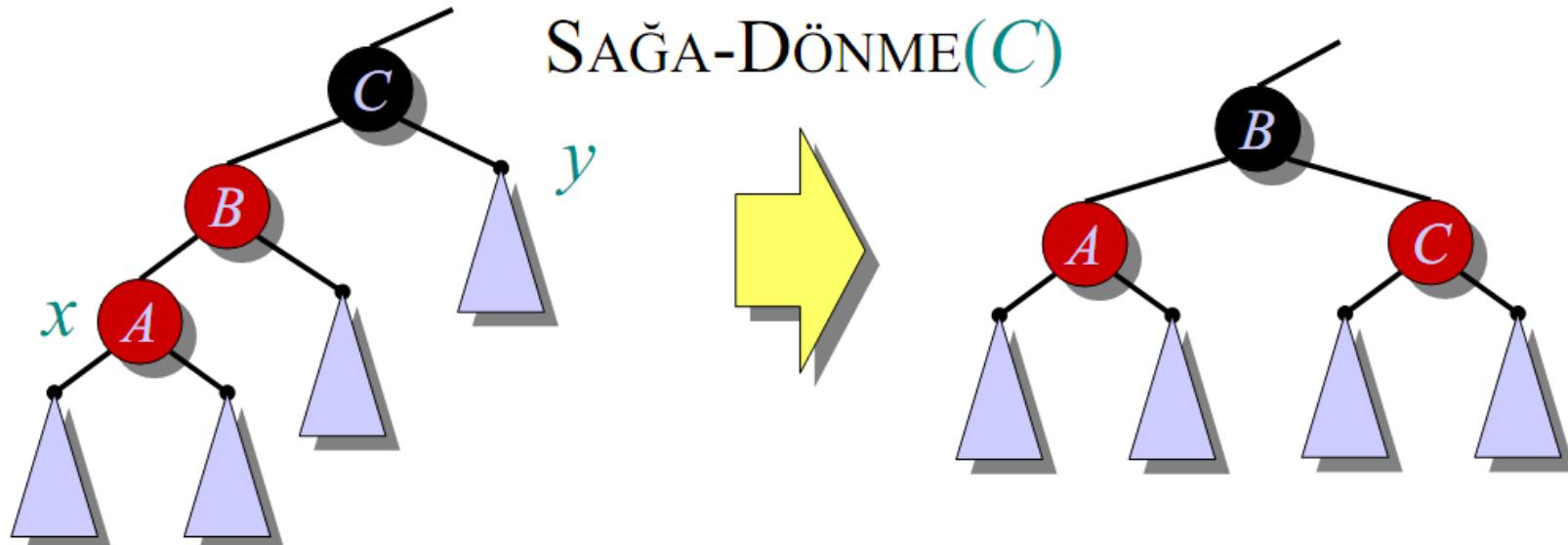
## Durum 2:



Durum 3'e dönüştürün.

- Ağaçta siyah düğümlerin sayısı bozulduysa (*bh*) veya *C*'nin bir siyah bir kırmızı çocuğu var ise döndürme işlemi gerçekleştirilecek. Yeni dönüşüm Durum 3 'ü meydana getirir.

## Durum 3:



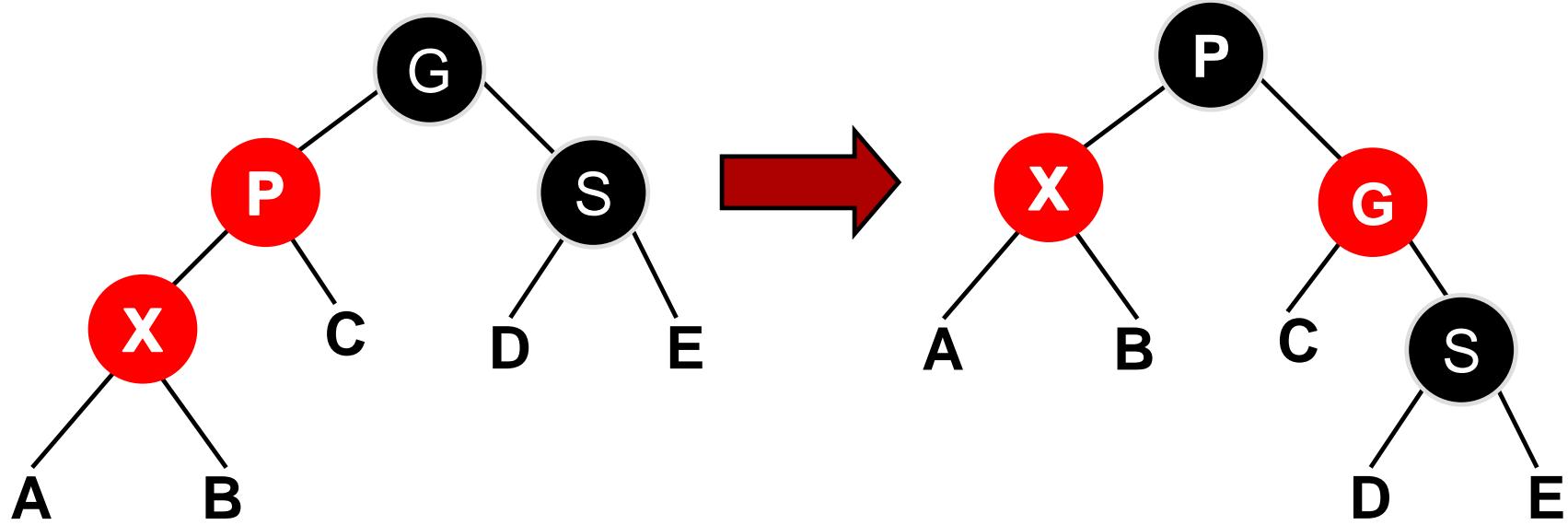
Bitti! RB ( Kırmızı-siyah) 3. özelliğin  
ihlali artık mümkün  
değil.

- Özellik 3 ihlali devam ettiğinden yeniden döndürme işlemi ve renklendirme yapılacak.

# Tek Döndürme(Single Rotation)

- Eklenmeden sonraki durum:
  - Ardışık red (P & X)
  - P'nin kardeşi S black
  - X dış düşüm (left-left veya right-right)

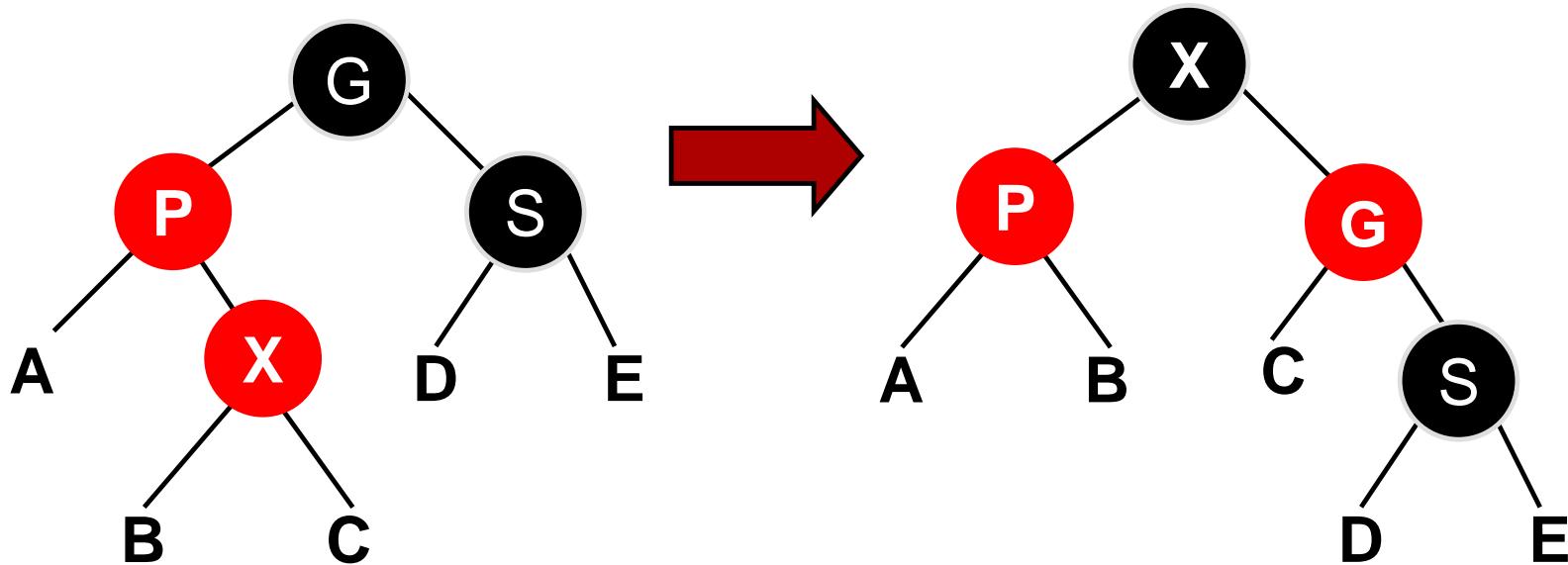
X: Yeni Düğüm  
P: Ebeveyn  
S: Kardeş  
G: Ata



# Çift Döndürme (Double Rotation)

- Eklenmeden sonraki durum:
  - Ardışık red (P & X)
  - P'nin kardeşi S black
  - X iç düğüm (left-right veya left)

X: Yeni Düğüm  
P: Ebeveyn  
S: Kardeş  
G: Ata



# Pseudocode

RB-INSERT( $T, x$ )

  TREE-INSERT( $T, x$ )

$\text{color}[x] \leftarrow \text{RED}$    ▷ only RB property 3 can be violated

**while**  $x \neq \text{root}[T]$  and  $\text{color}[p[x]] = \text{RED}$

**do if**  $p[x] = \text{left}[p[p[x]]]$

**then**  $y \leftarrow \text{right}[p[p[x]]]$            ▷  $y = \text{aunt/uncle of } x$

**if**  $\text{color}[y] = \text{RED}$

**then** ⟨Case 1⟩

**else if**  $x = \text{right}[p[x]]$

**then** ⟨Case 2⟩   ▷ Case 2 falls into Case 3

          ⟨Case 3⟩

**else** ⟨“then” clause with “left” and “right” swapped⟩

$\text{color}[\text{root}[T]] \leftarrow \text{BLACK}$

```

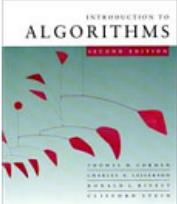
RB_Insert(x)
 Tree_Insert(x);
 x.color = RED;
 // 3. kural ihlali,
 while (x!=root && x.p.color == RED)
 if (x.p == x.p.p.left)
 y = x.p.p.right;
 if (y.color == RED)
 x.p.color = BLACK;
 y.color = BLACK;
 x.p.p.color = RED;
 } x = x.p.p;
 else // y.color == BLACK
 if (x == x.p.right)
 x = x.p;
 leftRotate(x);
 x.p.color = BLACK;
 x.p.p.color = RED;
 rightRotate(x.p.p);
 }
 else // x.p == x.p.p.right
 (yukarıdakine benzer, fakat "right" ve "left" yer değiştirecek)

```

Case 1: y= aunt/uncle of x is RED

Case 2

Case 3

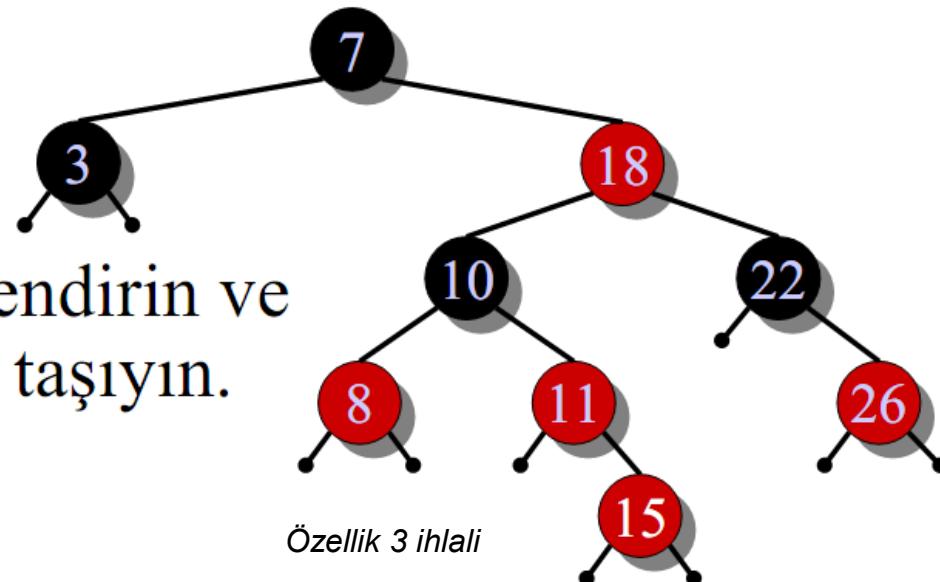


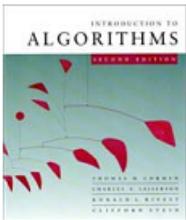
# Kırmızı-siyah ağaçta araya yerleştirme

**FIKİR:** Ağaçta  $x'$  i araya yerleştirin.  $x'$  i kırmızı yapın. Sadece kırmızı-siyah özellik 3 ihlal edilebilir. İhlali ağaç boyunca yukarı doğru, rotasyonlar ve yeniden renklendirmeyle düzelene kadar taşıyın.

## Örnek:

- Ar. Yer.  $x = 15$ .
- Yeniden renklendirin ve  
ihlali yukarıya taşıyın.



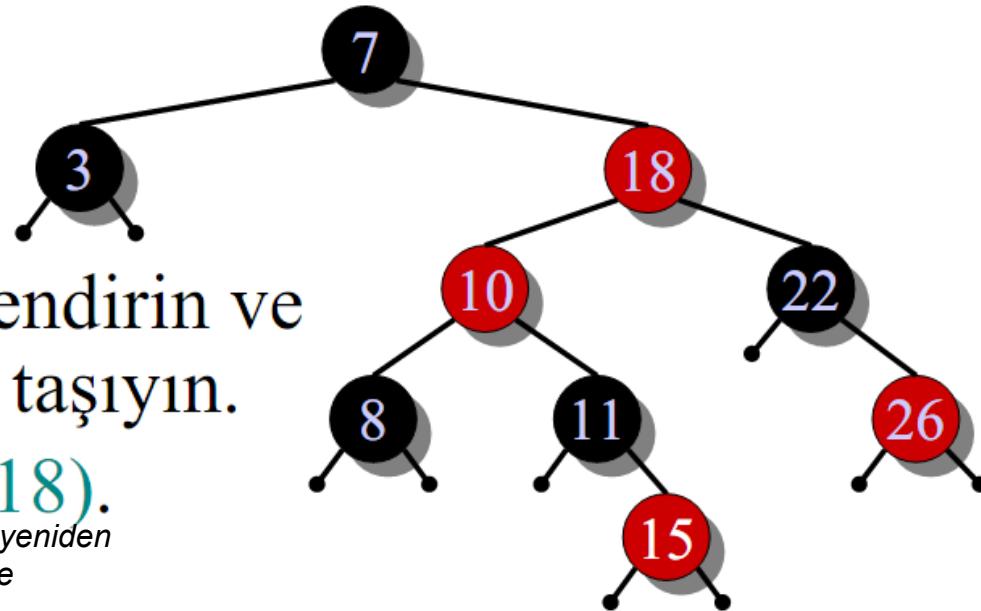


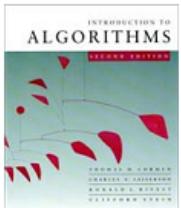
# Kırmızı-siyah ağaçta araya yerleştirme

**FİKİR:** Ağaçta  $x'$  i araya yerleştirin.  $x'$  i kırmızı yapın. Sadece kırmızı-siyah özellik **3** ihlal edilebilir. İhlali ağaç boyunca yukarı doğru, rotasyonlar ve yeniden renklendirmeyle düzelene kadar taşıyın.

## Örnek:

- Ar.Yer.  $x = 15$ .
- Yeniden renklendirin ve  
ihlali yukarıya taşıyın.
- SAĞA-DÖNME(18).  
Özellik 3 ihlali devam ettiğinden yeniden  
döndürme işlemi ve renklendirme  
yapılacak.



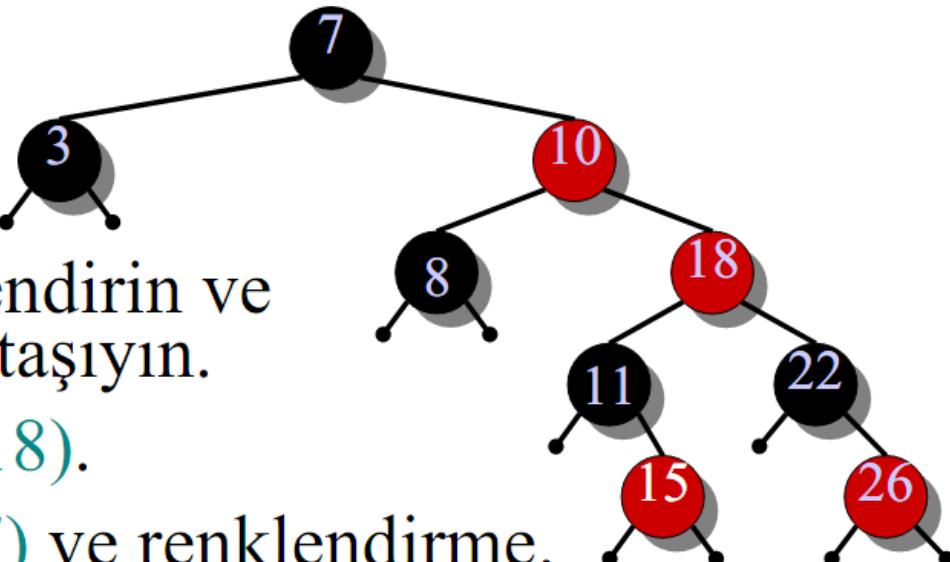


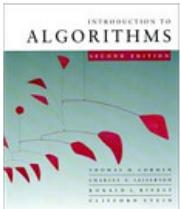
# Kırmızı-siyah ağaçta araya yerleştirme

**FİKİR:** Ağaçta  $x'$  i araya yerleştirin.  $x'$  i kırmızı yapın. Sadece kırmızı-siyah özellik **3** ihlal edilebilir. İhlali ağaç boyunca yukarı doğru, rotasyonlar ve yeniden renklendirmeyle düzelene kadar götürün.

## Örnek:

- Ar. Yer.  $x = 15$ .
- Yeniden renklendirin ve ihlali yukarıya taşıyın.
- SAĞA-DÖNME(18).
- SOLA-DÖNME(7) ve renklendirme.



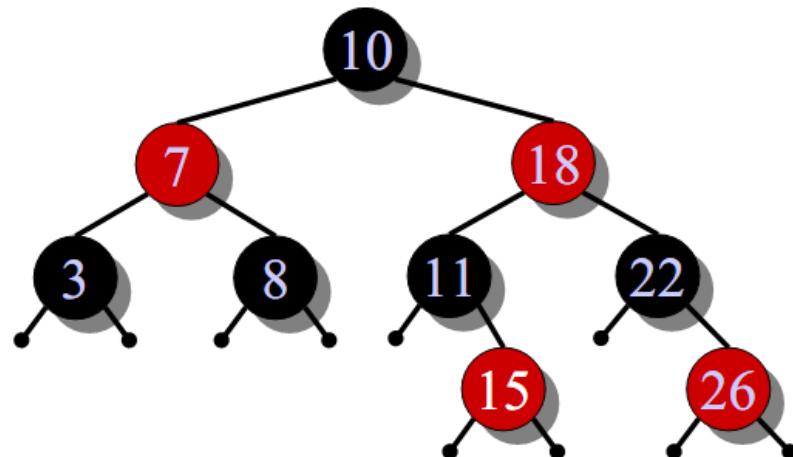


# Kırmızı-siyah ağaçta araya yerleştirme

**FIKİR:** Ağaçta  $x'$  i araya yerleştirin.  $x'$  i kırmızı yapın. Sadece kırmızı-siyah özellik **3** ihlal edilebilir. İhlali ağaç boyunca yukarı doğru, rotasyonlar ve yeniden renklendirmeyle düzeltilene kadar götürün .

## Örnek:

- Ar.Yer.  $x = 15$ .
- Yeniden renklendir, ihlali yukarıya taşı.
- SAĞA-DÖNME(18).
- SOLA-DÖNME(7) ve yeniden renklendirme.



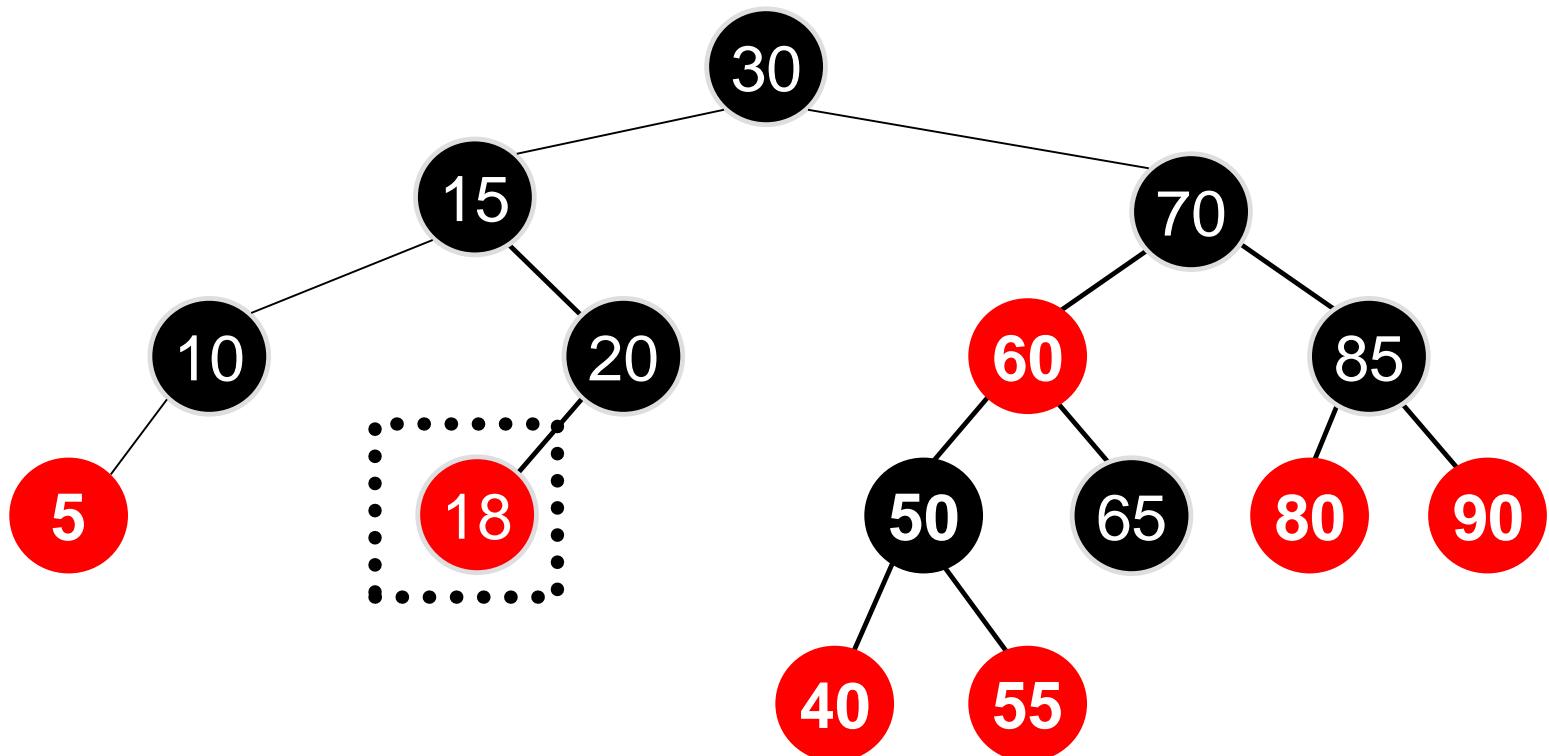
## Çözümleme

- Ağaçta yukarıya giderken Durum 1' i uygulayın; bu durumda sadece düğümler yeniden renklendirilir.
- Eğer Durum 2 veya 3 ile karşılaşırsanız, 1 ya da 2 rotasyon yapın ve işlemi sonlandırın.

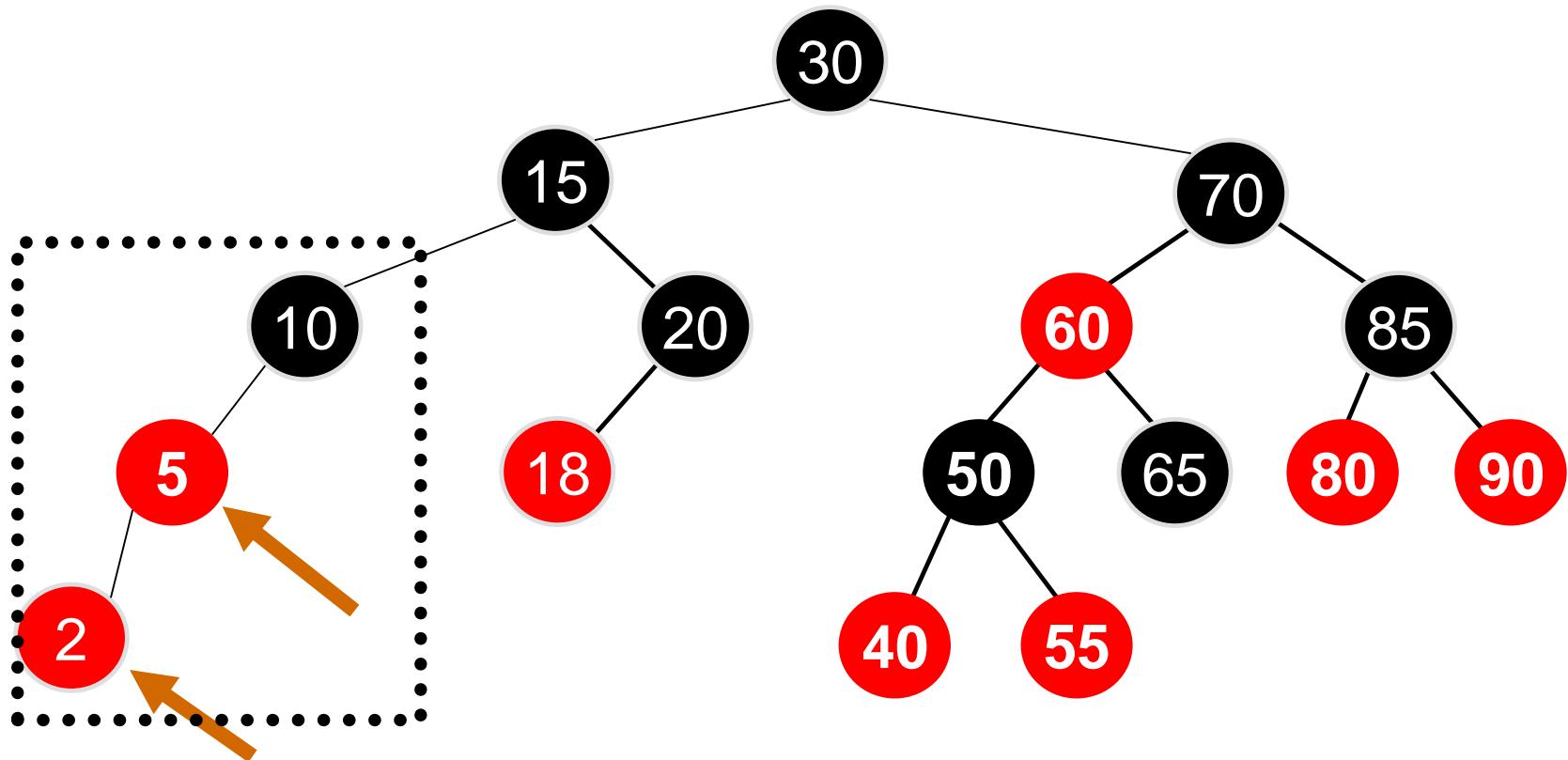
**Yürütmüş süresi:**  $O(\lg n)$  ve  $O(1)$  rotasyon.

RB-DELETE ( KIRMIZI\_SIYAH SİLME) — asimptotik koşma süresi ve rotasyonların sayısı RB-INSERT (KIRMIZI-SİYAH ARAYA YERLEŞTİRME) ile aynıdır.

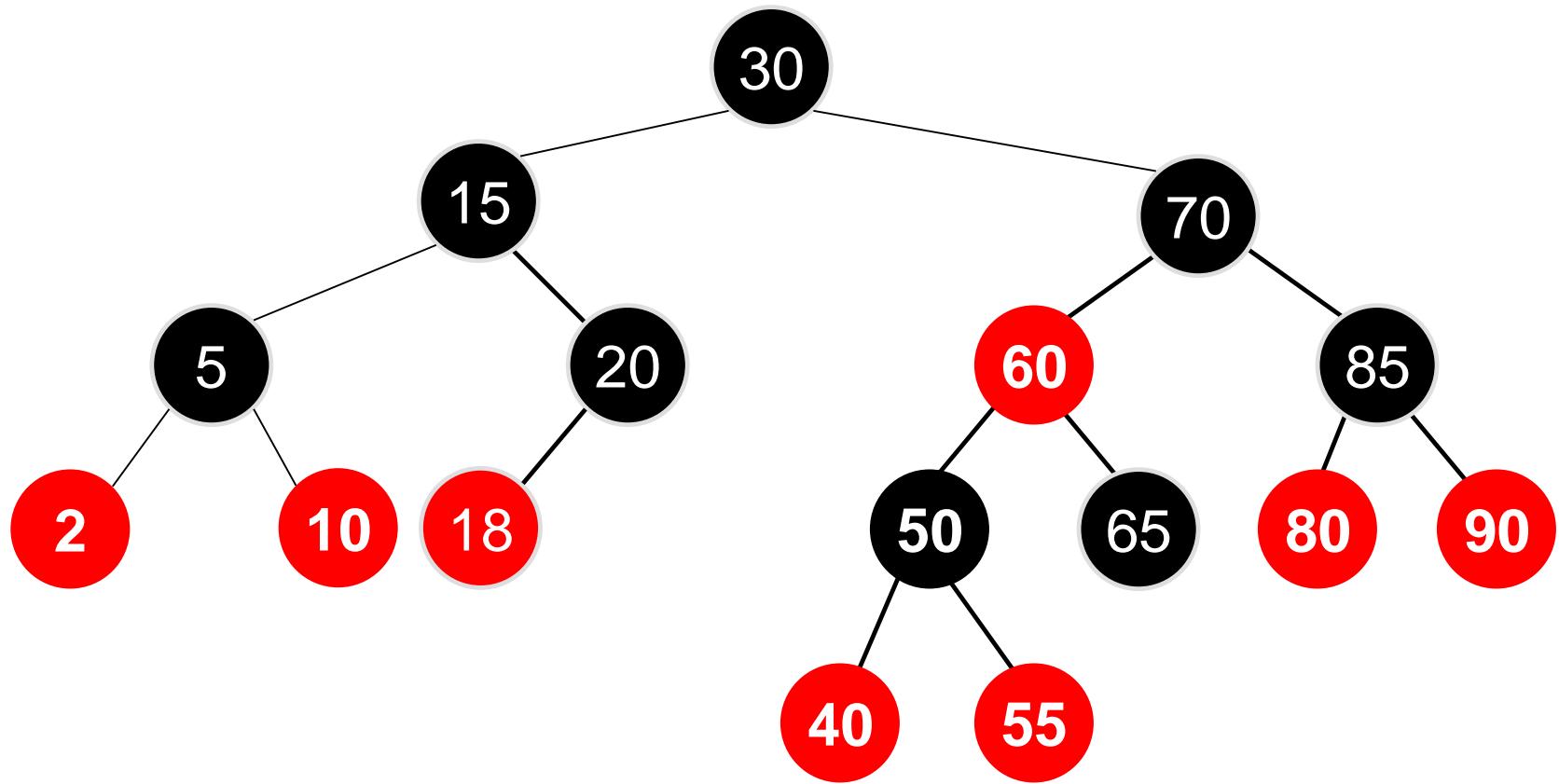
## Örnek: 18 değerinin eklenmesi



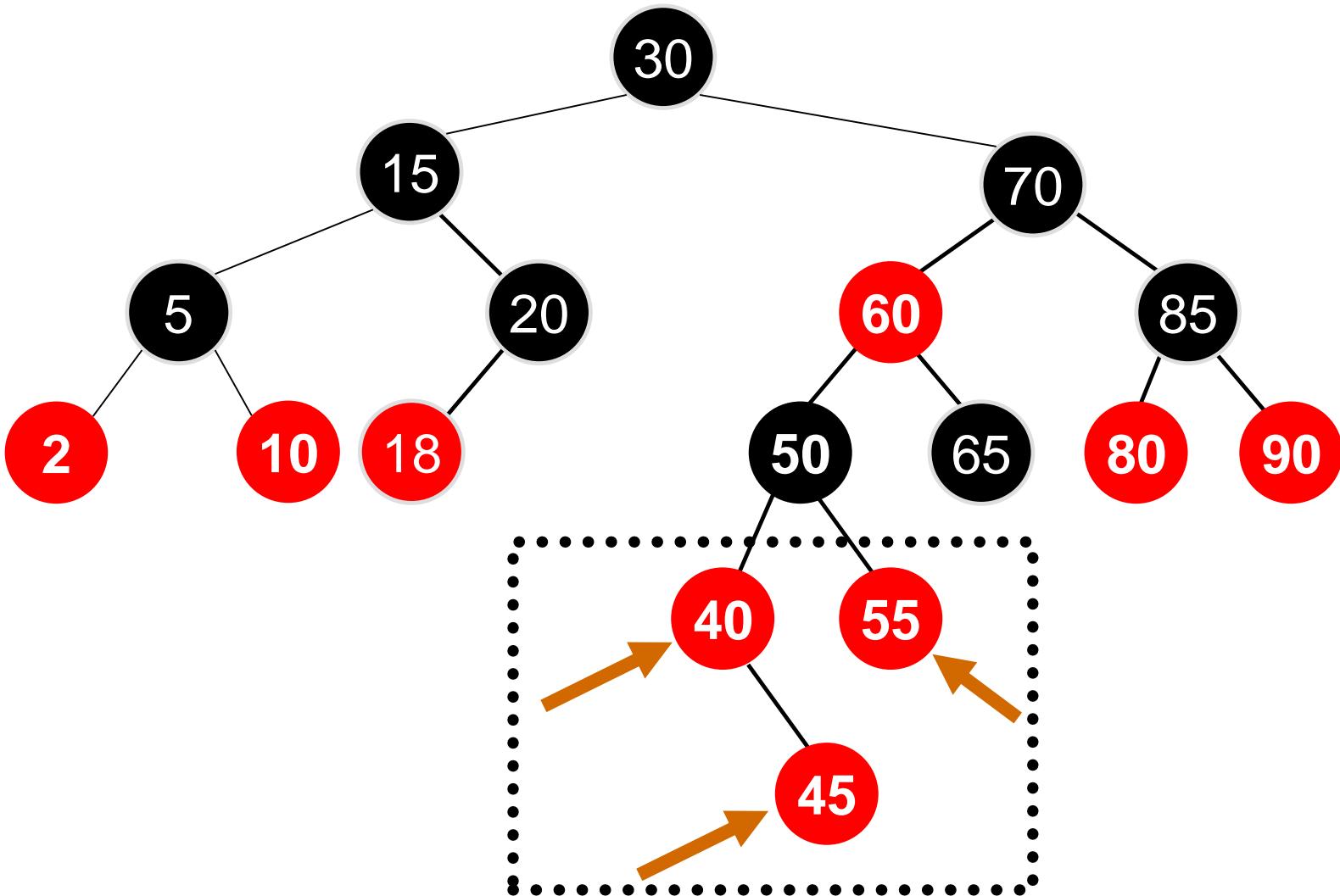
## Örnek: 2 değerinin eklenmesi



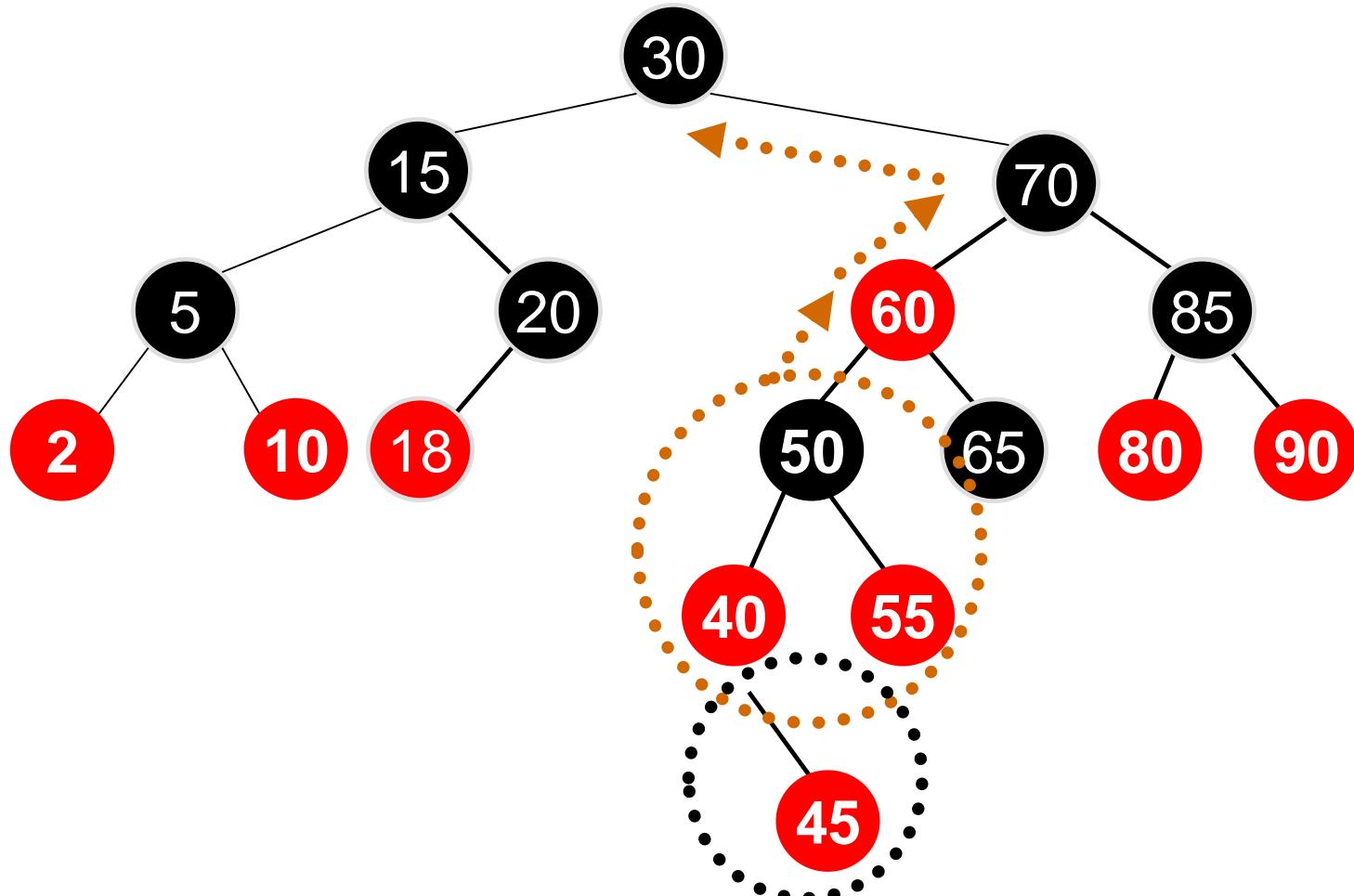
## Örnek: 2 değerinin eklenmesi



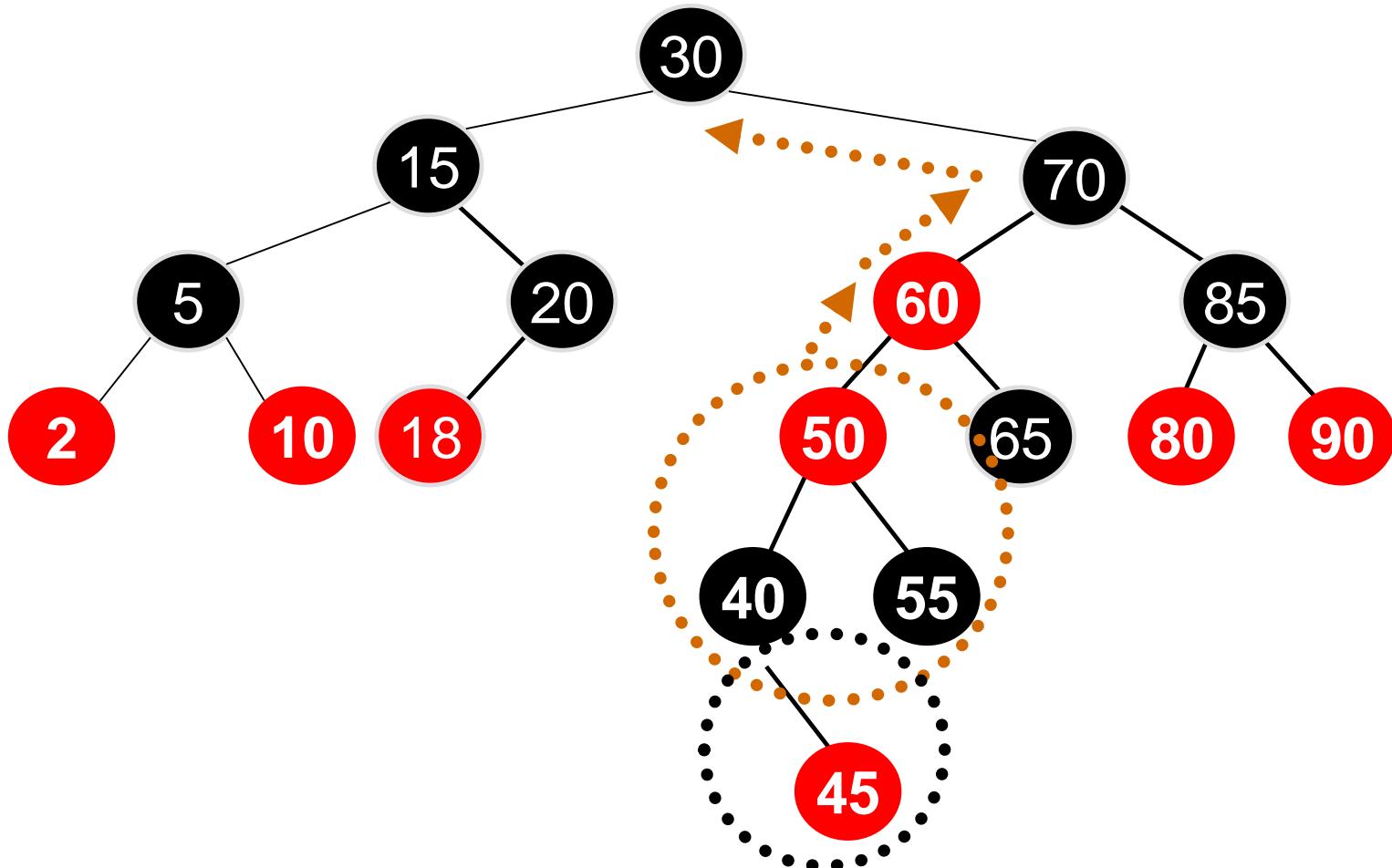
# Örnek: 45 değerinin eklenmesi



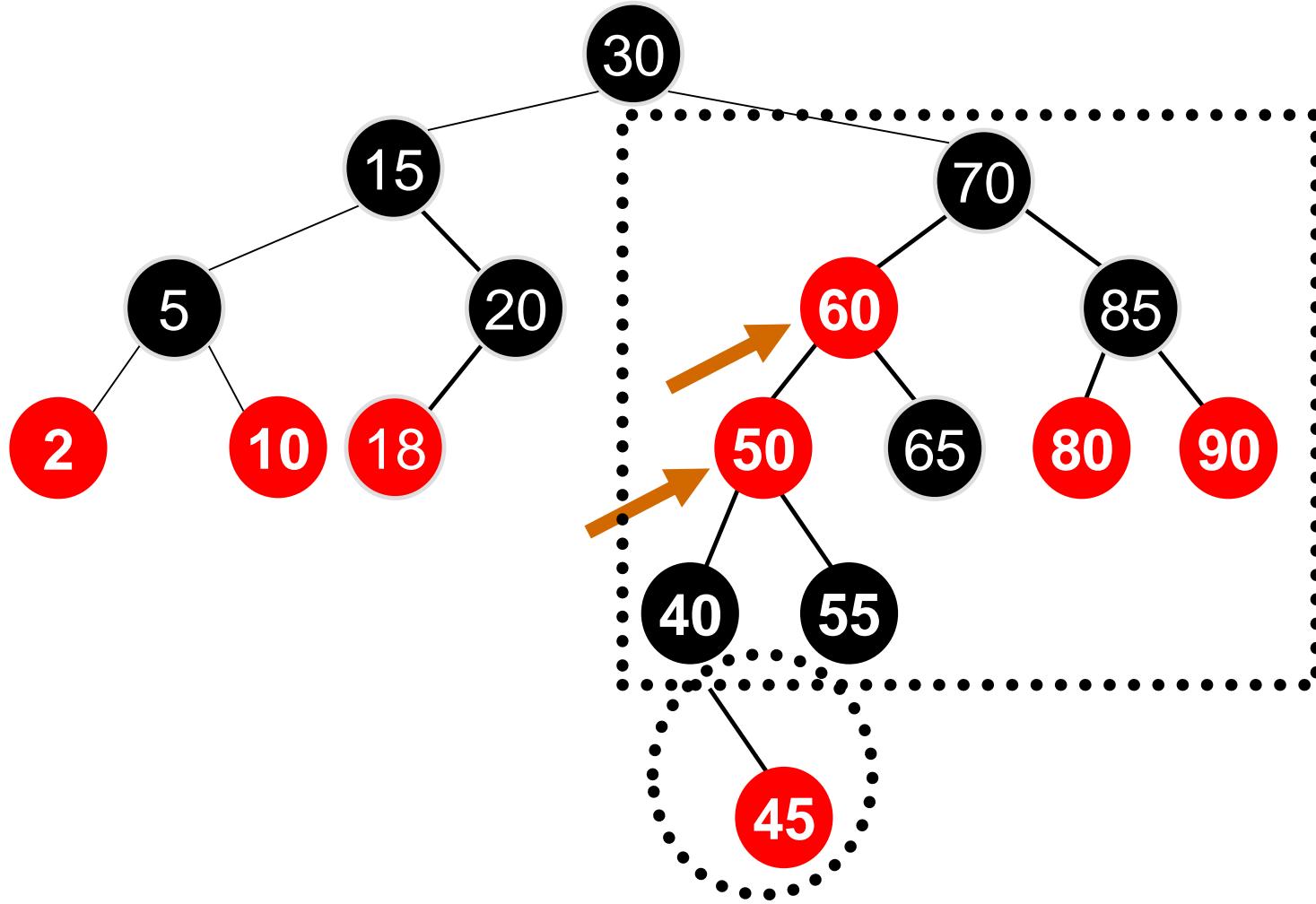
## Örnek: 45'i Ekle



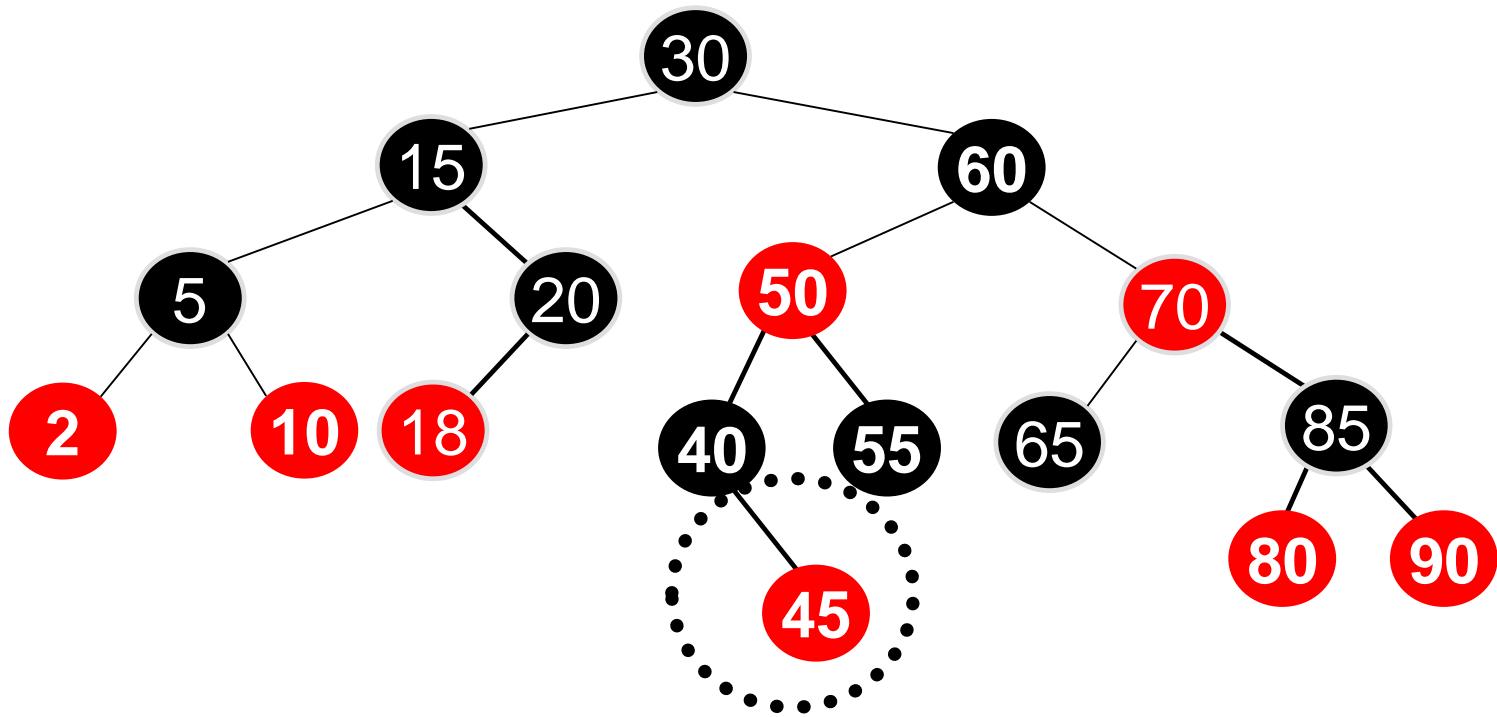
## Örnek: 45'i Ekle



## Örnek: 45'i Ekle (Tek Döndürme)

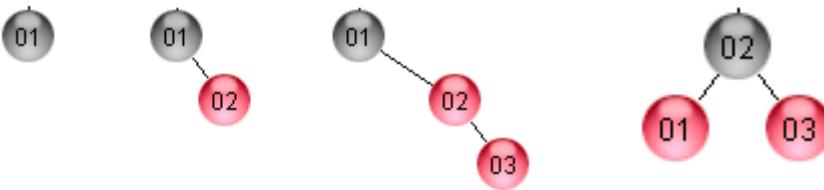


# Örnek: 45'i Ekle (Tek Döndürme)



# Örnek: Red-Black tree ekleme

- Eklenenek değerler: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 ,11, 12, 13, 14

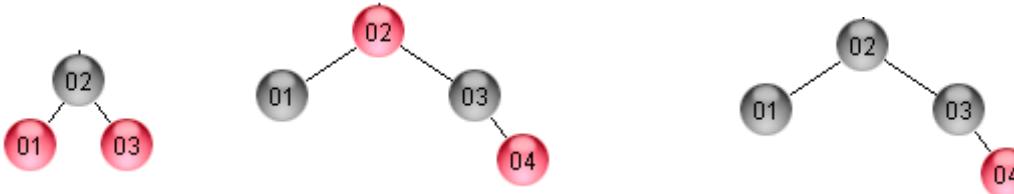


- 3. ve 4. özellik ihlali var. 3 nolu düğümün atası siyah olmalı. 2 nolu düğüm siyah 1 nolu düğüm kırmızı olacak şekilde yeniden renklendirme yapılır.
- 4. özellik dikkate alınırsa bu defa bh değeri bütün yollarda aynı değil bu defa sola döndürme işlemi yapılacak.
  - 3 eklendi, 2 Siyah, 1 Kırmızı, Root (kök) 1 Sola döndü

# Örnek: Red-Black tree ekleme

- Eklenenek değerler: 1,2,3,**4**,5,6,7,8,9,10,11,12,13,14

- 



- 4 nolu düğüm eklendiğinde 3 ve 4 nolu düğümler kırmızı olduğundan 3. özellik ihlali mevcut. Yeniden renklendir:
- 3 nolu düğüm siyah atası kırmızı yapılacak.
- Ata düğüm kırmızı olduğunda çocukları mutlaka siyah olmalı bu nedenle 1 nolu düğüm de siyah yapıldı. Ayrıca kök düğüm kırmızı olmayacağından 2 nolu düğümde siyah yapıldı. (bh değerleri aynı)
- 4 eklendi, 2 Kırmızı, 1 Siyah, 3 Siyah, 2 Siyah

# Örnek: Red-Black tree ekleme

- Eklenen değerler: 1,2,3,4,5,6,7,8,9,10,11,12,13,14

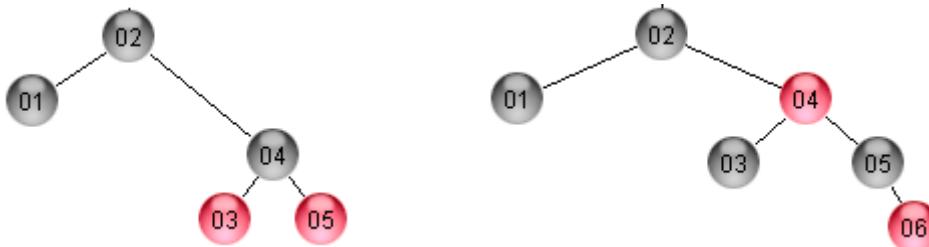


- 5 nolu düğüm eklendiğinde kırmızı olduğundan özellik 3 ihlali oluştu. 4 nolu düğümü siyah, atası olan 3 kırmızı olacak şekilde yeniden renklendirme yapıldı. Bu defa 4. özellik ihlali oluştu. Sola döndürme işlemi yapılarak ağaç dengelendi.
  - 5 eklendi, 4 B, 3R, 3 sola döndü

# Örnek: Red-Black tree ekleme

- Eklenecek değerler:

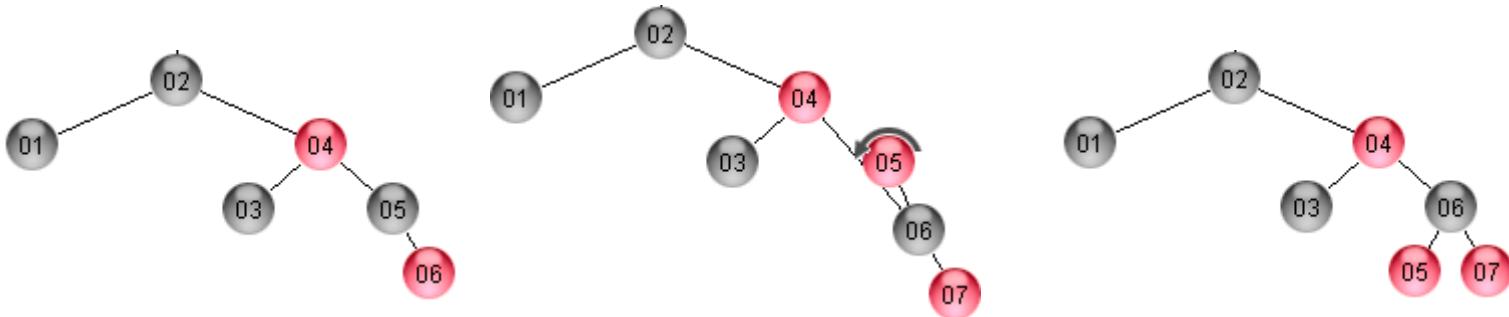
1,2,3,4,5,**6**,7,8,9,10,11,12,13,14



- 3. özellik ihlali yeniden renklendirme yapıldı.
  - 6 eklendi, 3B, 5B, 4R

# Örnek: Red-Black tree ekleme

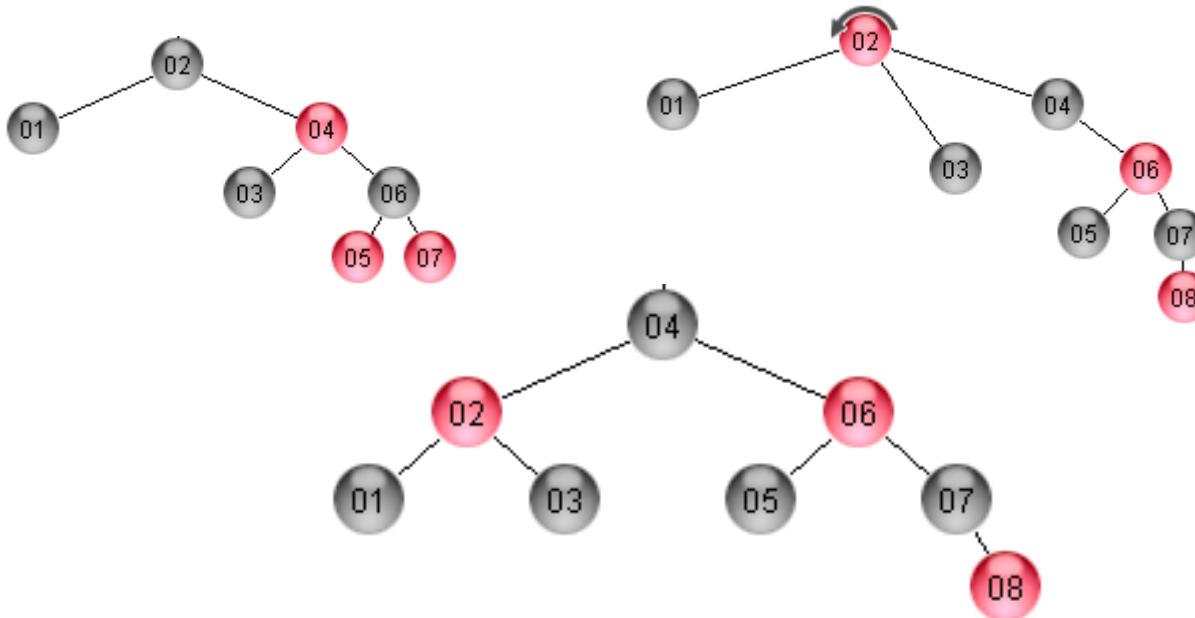
- Eklenenek değerler: 1,2,3,4,5,6,7,8,9,10,11,12,13,14



- 3. ve 4. özellik ihlali
  - 7 Eklendi, 6B, 5R, 5 sola

# Örnek: Red-Black tree ekleme

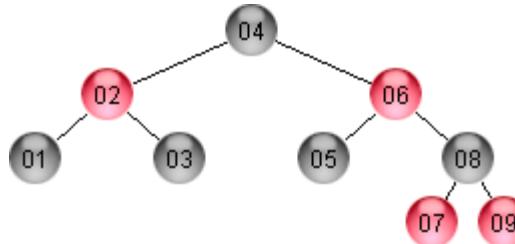
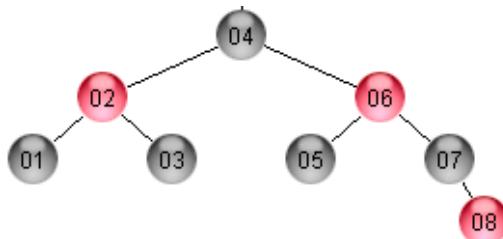
- Eklenen değerler: 1,2,3,4,5,6,7,8,9,10,11,12,13,14



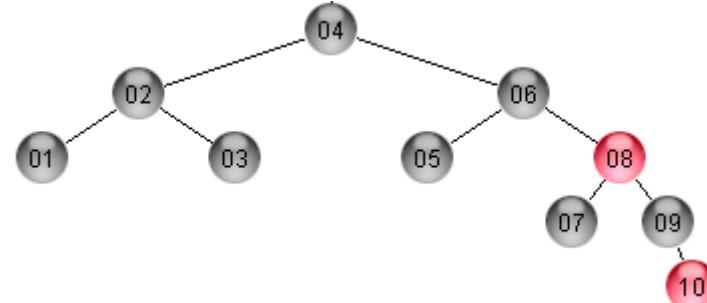
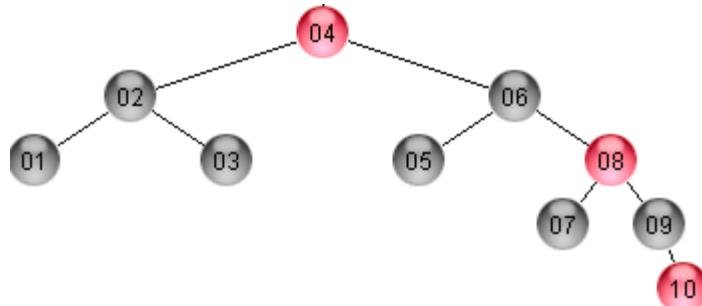
- 8 eklendi, 5B, 7B, 6R, 4B, 2R, 2 Sol

# Örnek: Red-Black tree ekleme

- Eklenenek değerler: 1,2,3,4,5,6,7,8,**9,10**,11,12,13,14



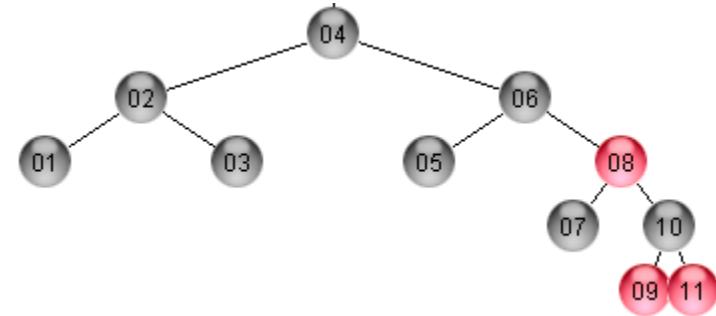
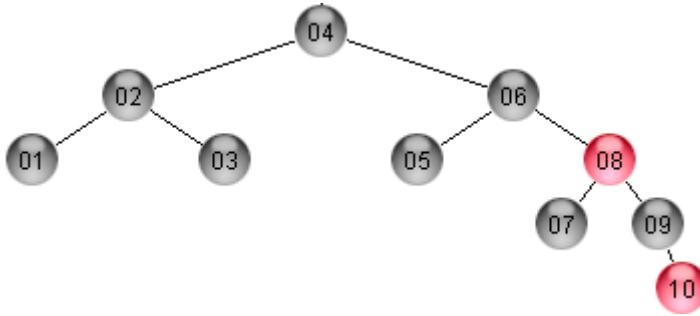
- 9 Eklendi, 8B, 7R, 7 Sol



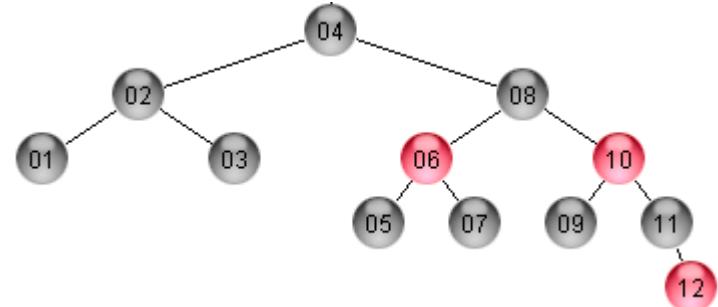
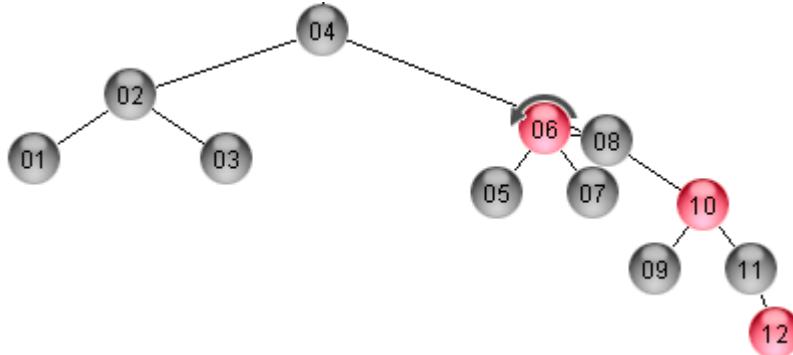
- 10 Eklendi, 7B, 9B, 8R, 2B, 6B, 4R, 4B

# Örnek: Red-Black tree ekleme

- Eklenecek değerler: 1,2,3,4,5,6,7,8,9,10,**11,12**,13,14



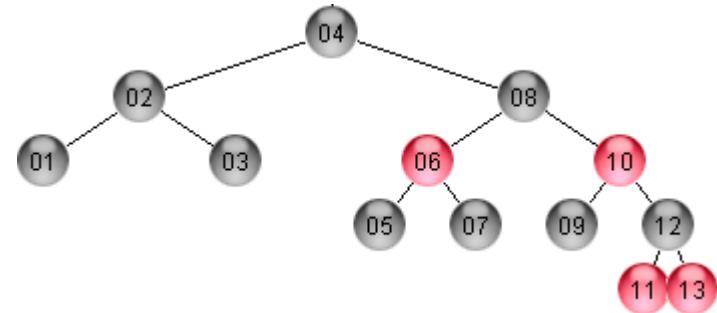
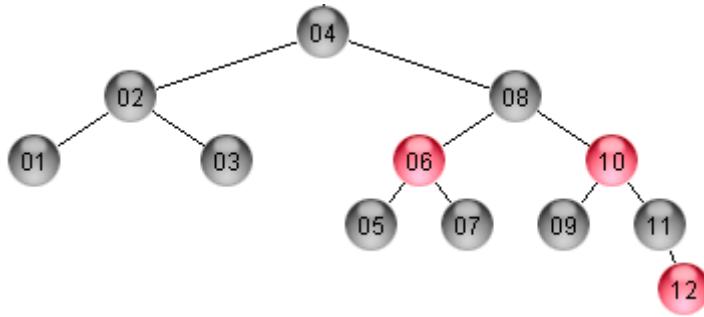
- 11 Eklendi, 10B,9R, 9 Sol



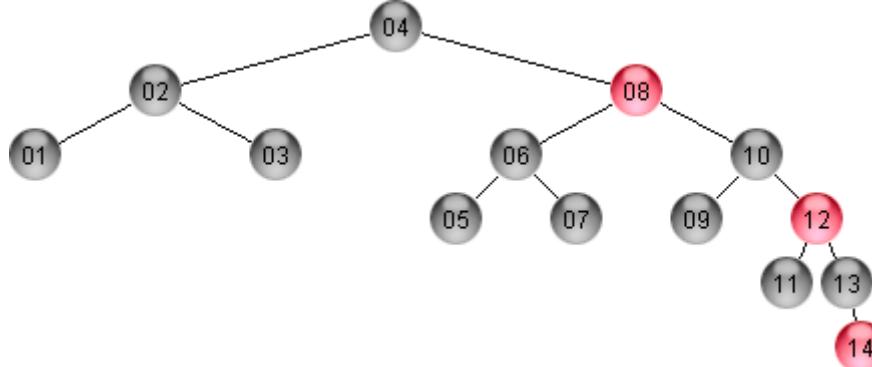
- 12 Eklendi, 9B,11B,10R, 8B,6R, 6 Sol

# Örnek: Red-Black tree ekleme

- Eklenecek değerler: 1,2,3,4,5,6,7,8,9,10,11,12, **13,14**



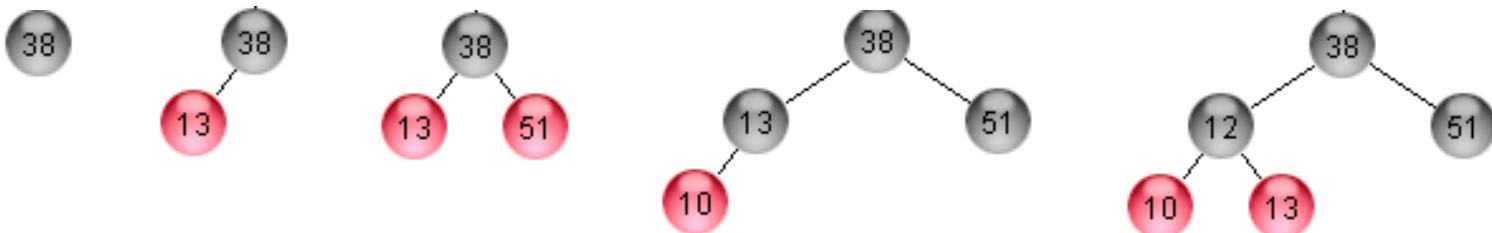
- 13 Eklendi, 12B,11R, 11 Sol



- 14 Eklendi,11B,13B,12R, 8R,6B, 10B

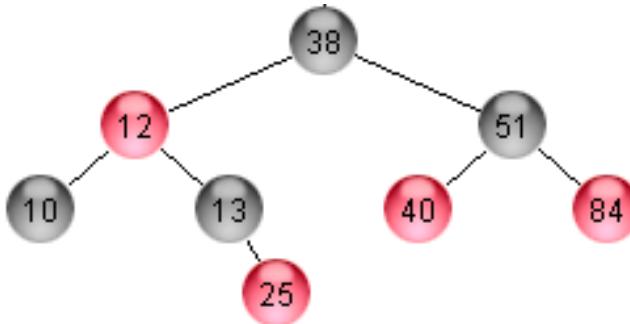
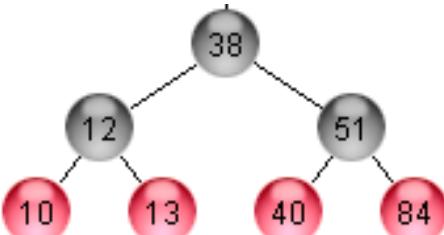
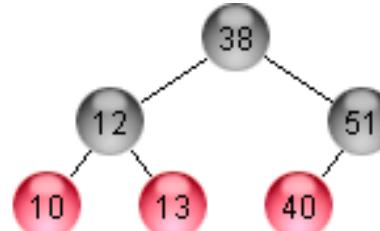
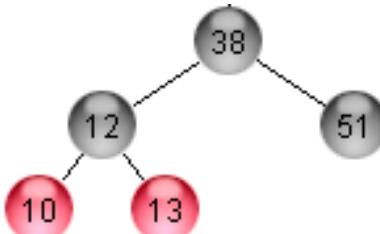
# Örnek: Red-Black tree ekleme

- Aşağıdaki değerleri her eklemeye Red-Black tree özelliğini uygulayınız.
- 38,13,51,10,12,40,84,25



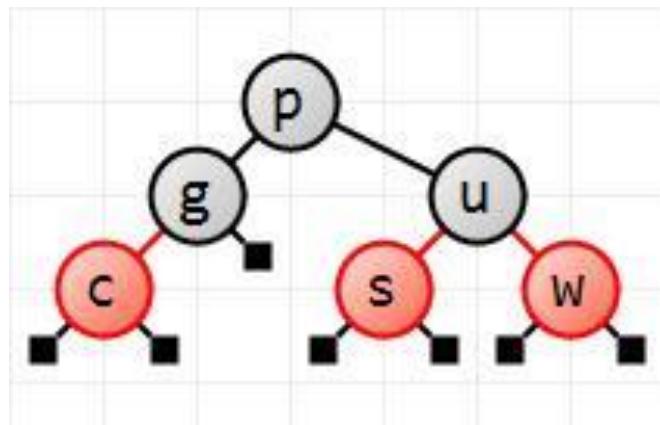
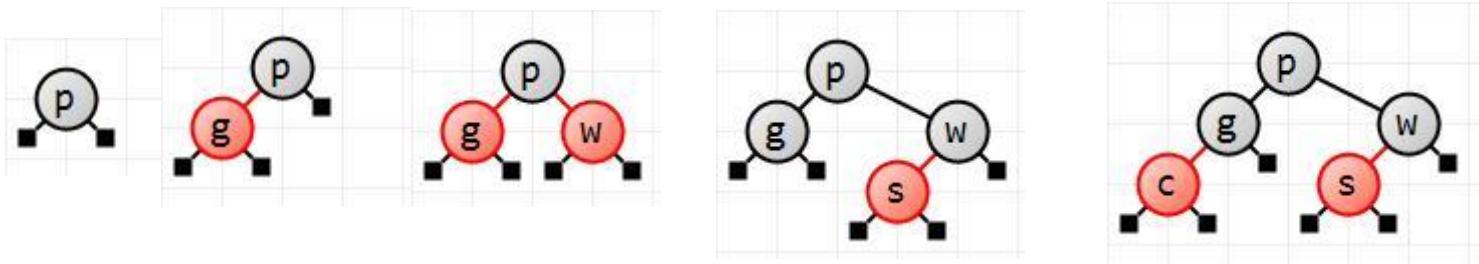
# Örnek: Red-Black tree ekleme

- 38,13,51,10,12,40,84,25

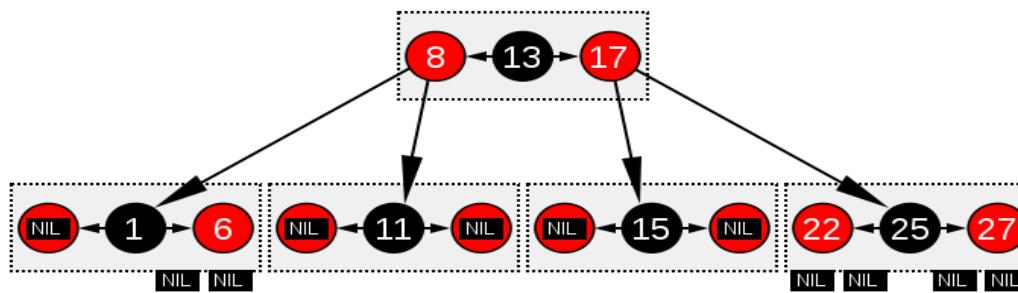
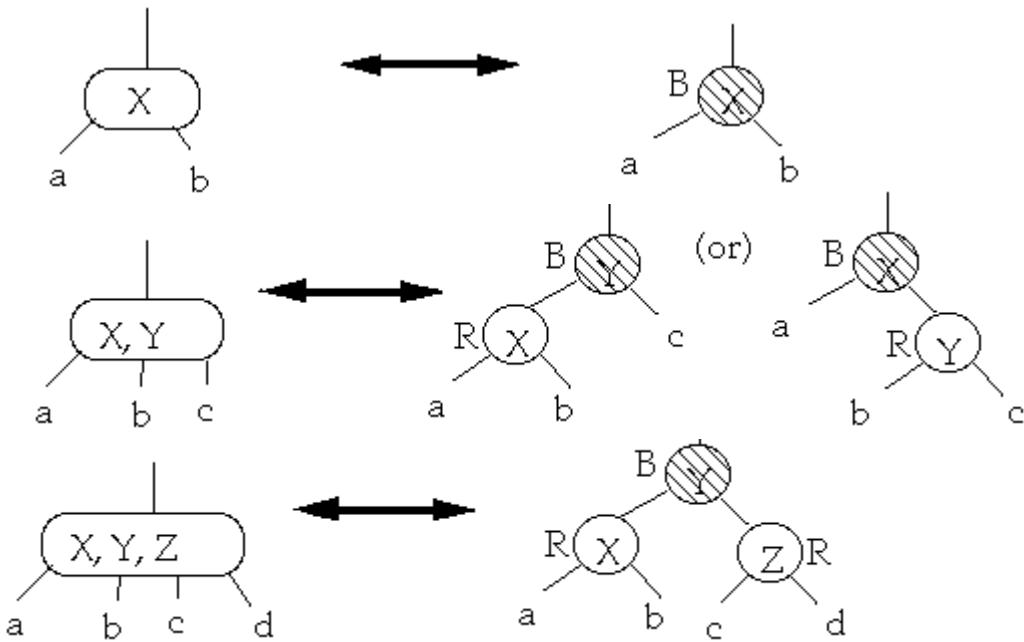


# Örnek: Red-Black tree ekleme

- p,g,w,s,c,u



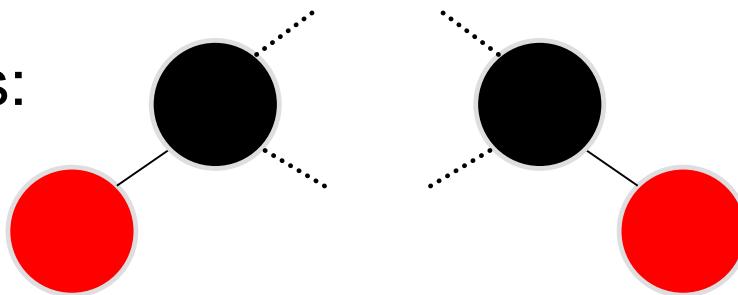
## 2-3-4-tree ve Red-Black tree gösterimi



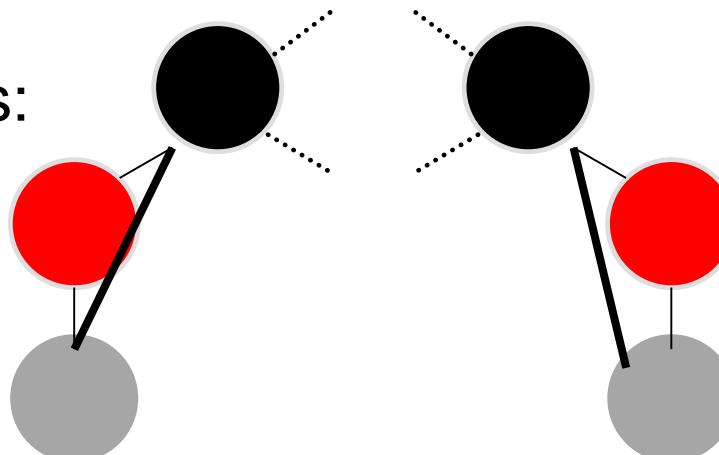
# Red-Black Tree: Silme

- AVL ağaçlarında olduğu gibi çocukları olan düğüm silindiğinde soldaki en büyük düğüm veya sağdaki en küçük düğüm alınır
- Eğer silinen düğüm kırmızı ise, problem yok.

Leaf nodes:

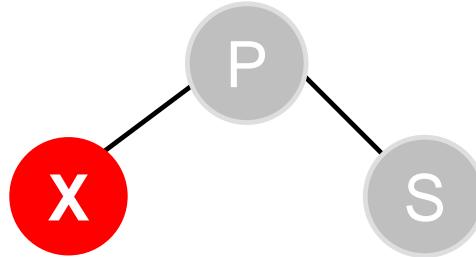
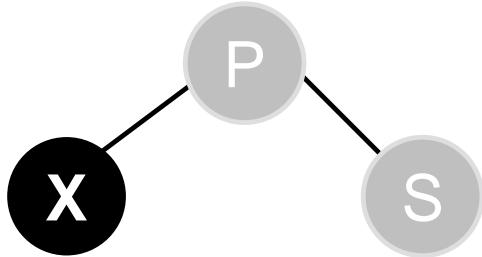


Single child nodes:



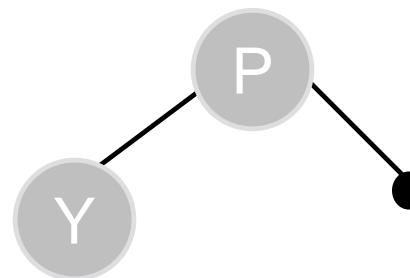
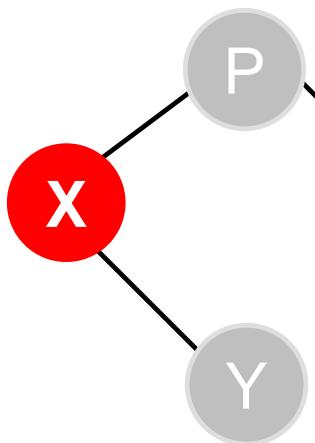
# Red-Black Tree: Silme

- Eğer silinen düğüm kırmızı ise, problem yok.
- Eğer siyah ise kırmızı yap ve sil



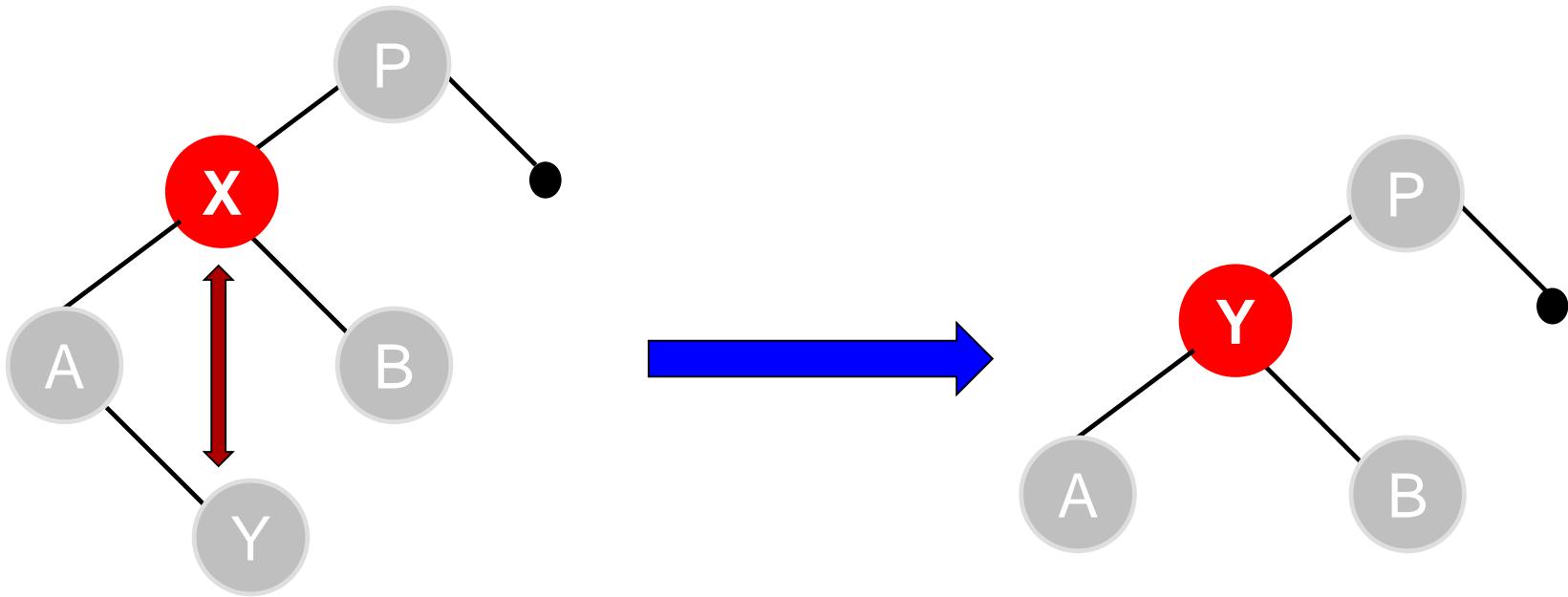
# Red-Black Tree: Deletion

- Silinen düğümün tek çocuğu var ise yer değiştir ve kırmızı ise sil.



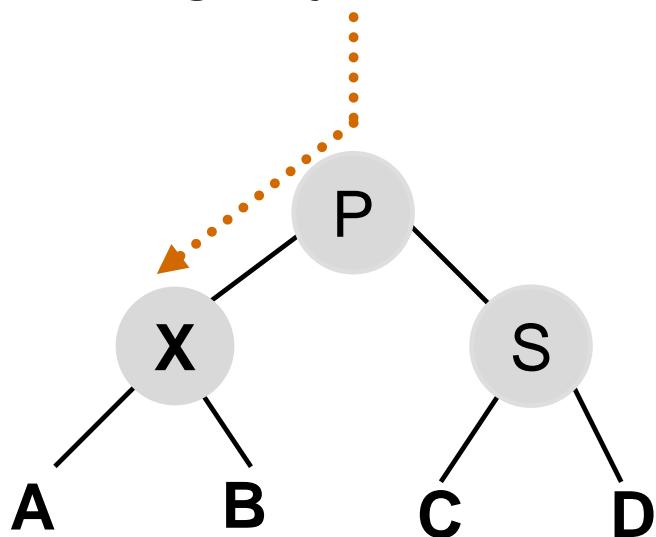
# Red-Black Tree: Deletion

- Silinen düğümün iki çocuğu var ise soldaki en büyük düğüm ile yer değiştirir ve kırmızı ise sil.



# Top-Down Deletion

- Silinecek düğüm siyah ise → 4. özellik ihlali
- Silinecek düğümün daima kırmızı olmasını sağla.
- Kökten başlayarak, yukarıdan-aşağı doğru seyahat ederek silinecek düğüm için bak.

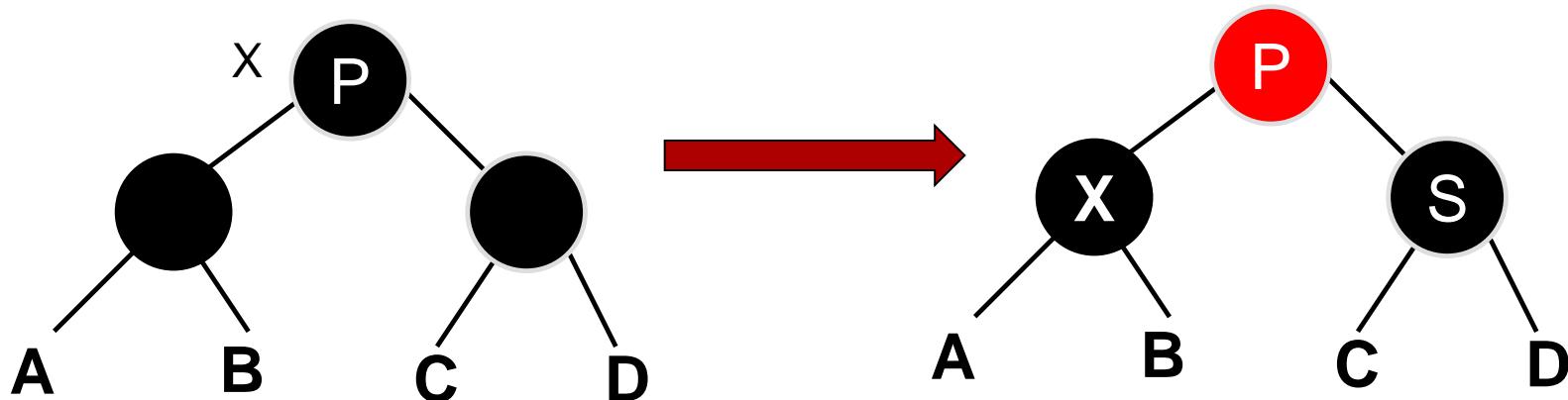


X: ziyaret edilen düğüm  
P: ebeveyn  
S: kardeş

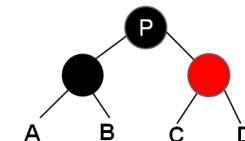
**Fikir:** X'i kırmızı yaptığından emin ol!

# Muhtemel Durumlar- Adım 1

- **Adım 1-A1:** Kökün her iki çocuğu da siyah ise;
  - a) Kökü kırmızı yap
  - b) X'i kökün uygun çocوغuna taşı
  - c) Adım 2'ye geç

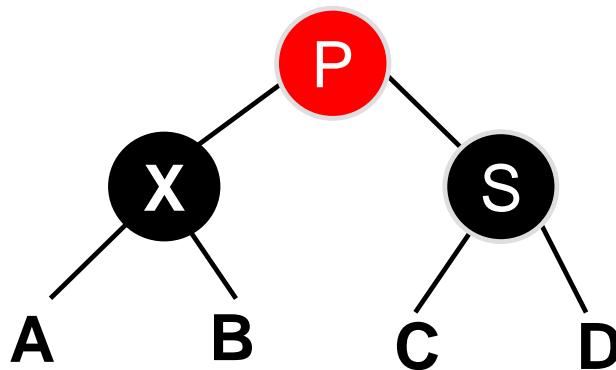


- **Adım 1-A2:** Kökün her iki çocuğu da siyah değil ise, X'i kök ( $X=P$ ) olarak tasarla ve **Adım 2-B**'yi işlet.



## Muhtemel Durumlar-Adım 2

- Silinecek düğüme ulaşıncaya kadar devam et; **P** kırmızı, **X** ve **S** siyah.



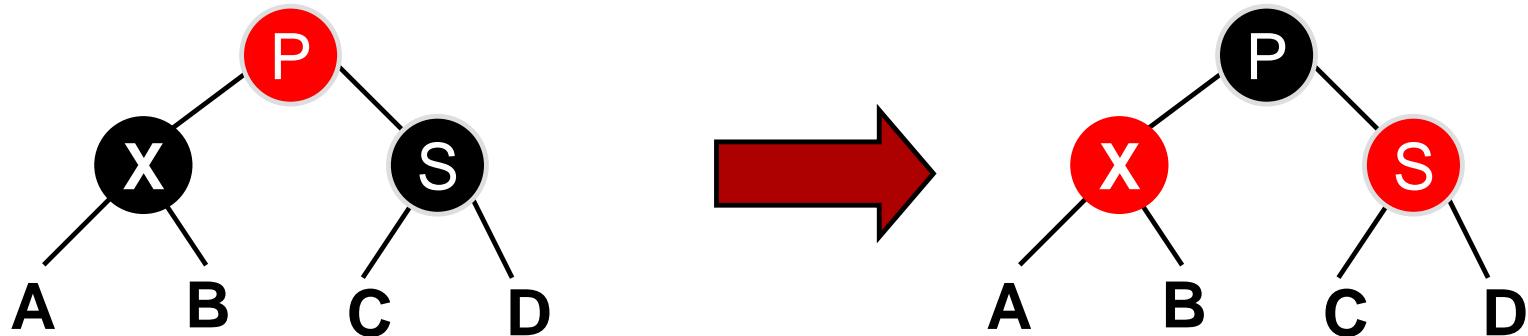
- X'i kırmızı yaptığımız zaman, X ve S'nin çocuklarına bağlı olarak muhtemelen döndürme işlemi yapılacak. Burada 2 durum mevcut.
  - Adım 2-A:** X'in çocukları (A & B) siyahdır (black)
  - Adım 2-B:** X'in çocuklarından en az biri kırmızı (red ) (A, B, veya ikisi)

## Adım 2-A

- **X'** in çocuklar (A ve B) siyah ise **S'** nin çocuklarına bağlıdır.
- **Adım 2-A1:** **S'** in her iki çocuğu da siyah, P, X, S yeniden renklendir.
- **Adım 2-A2:** **S'** in sol çocuğu kırmızı, çift döndürme
- **Adım 2-A3:** **S'** in sağ çocuğu kırmızı (Her iki çocuğu da kırmızı ise sağ çocuğa göre işlem yap), tek döndürme

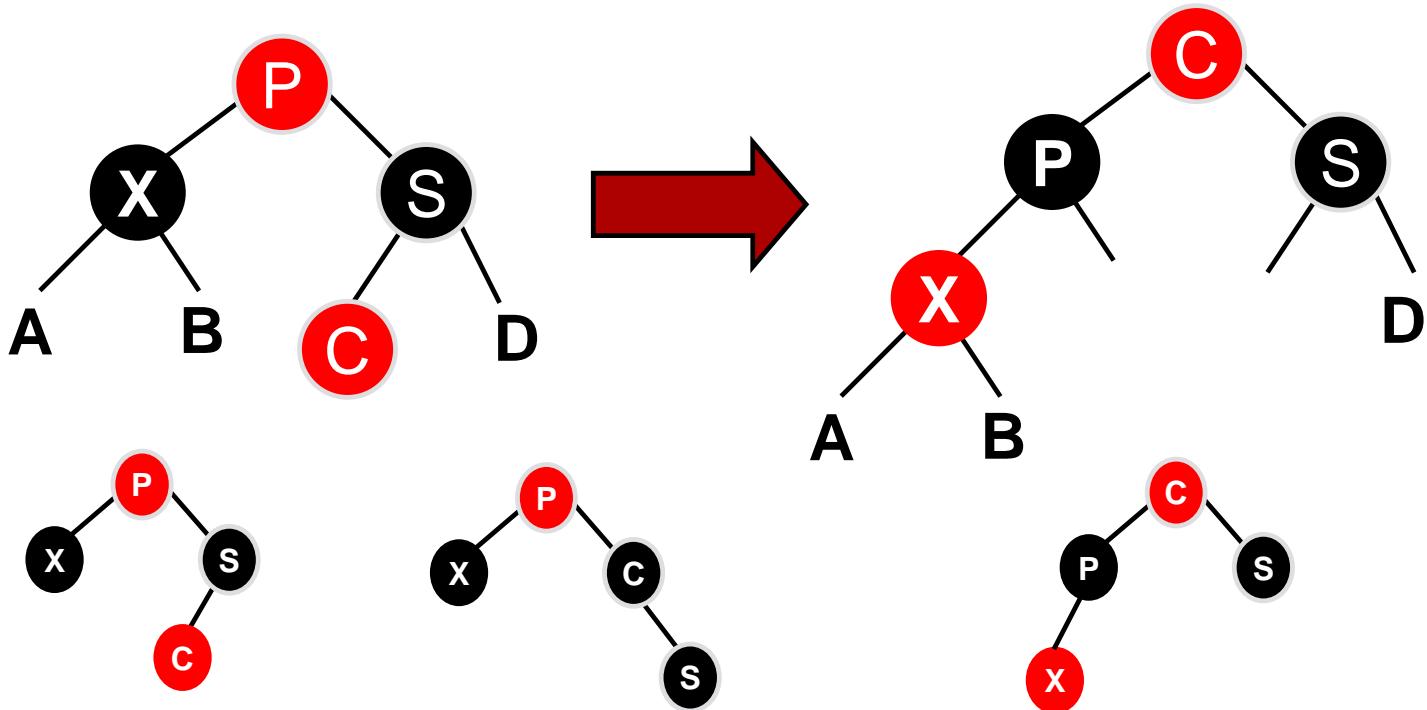
## Adım 2-A

- Adım 2-A1: S'in her iki çocuğu da siyah ise, yeniden renklendir. (Adım 1-A1 uygulanır)



## Adım 2-A

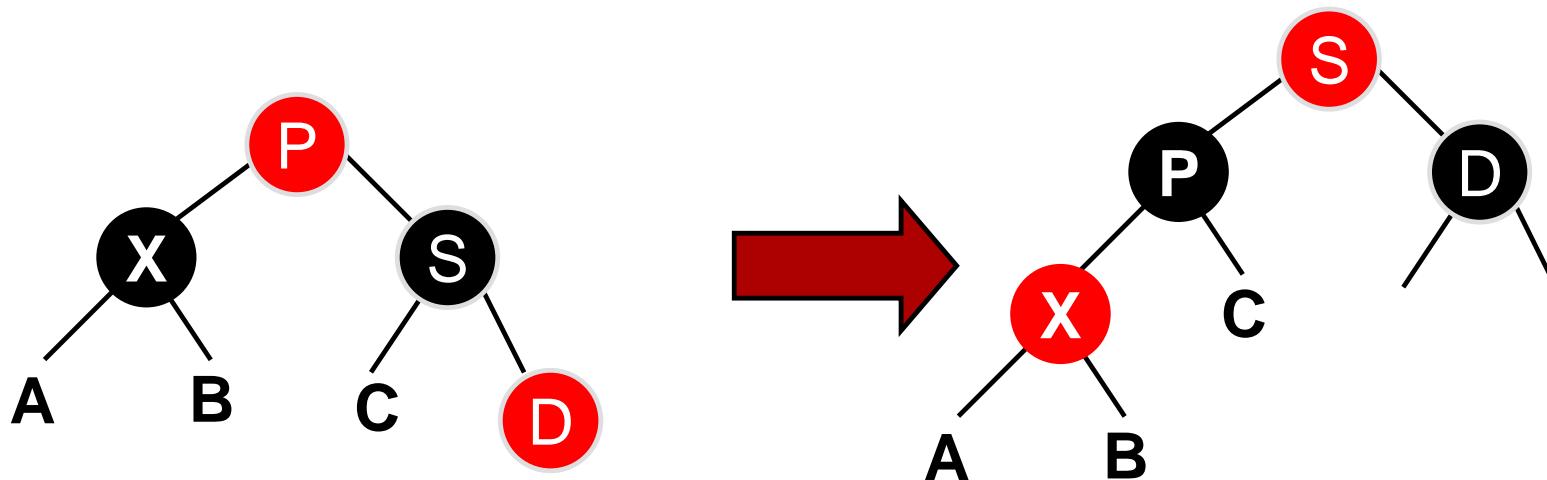
- Adım 2-A2: S'in sol çocuğu kırmızı ise, çift döndürme yap ve renklendir(P siyah ise sadece X kırmızı olur)



Her düğüm yerleştiği düğümün rengini alıyor, son adımda X direk kırmızı yapılıyor

## Adım 2-A

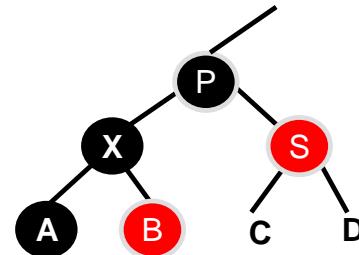
- **Adım 2-A3:** S'in sağ çocuğu kırmızı (Her iki çocuğu da kırmızı ise sağ çocuğa göre işlem yap) ise, tek döndürme yap ve renklendir



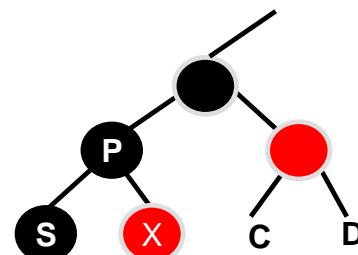
Her düğüm yerlesiği düğümün rengini alıyor, son adımda X direk kırmızı yapılıyor

## Adım 2-B

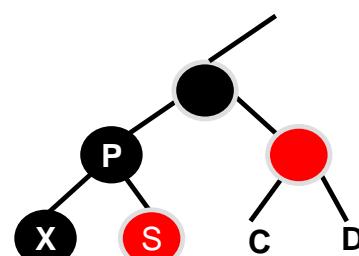
- X'in çocuklarından en az biri kırmızı ise; X'i uygun çocuğa taşı



- **Adım 2-B1:** Eğer yeni X kırmızı ise taşımaya devam et

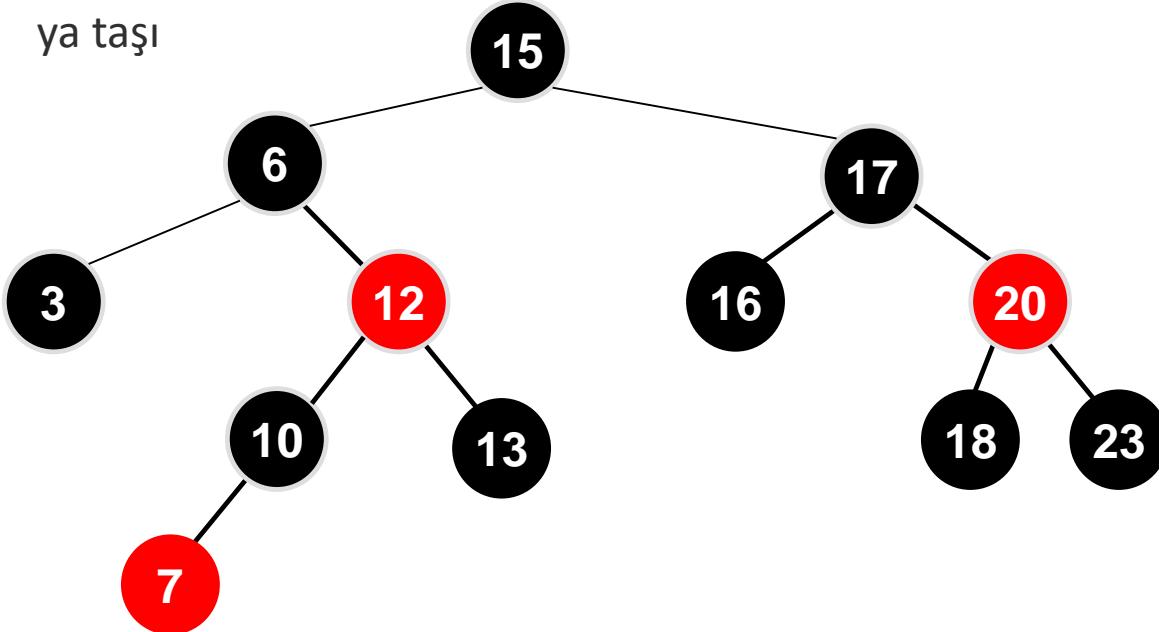


- **Adım 2-B2:** Eğer yeni X siyah ise ( P Siyah, S kırmızı), S'yi P' nin etrafında dönder P ve S'yi yeniden renklendir ve Adım 2'ye git.



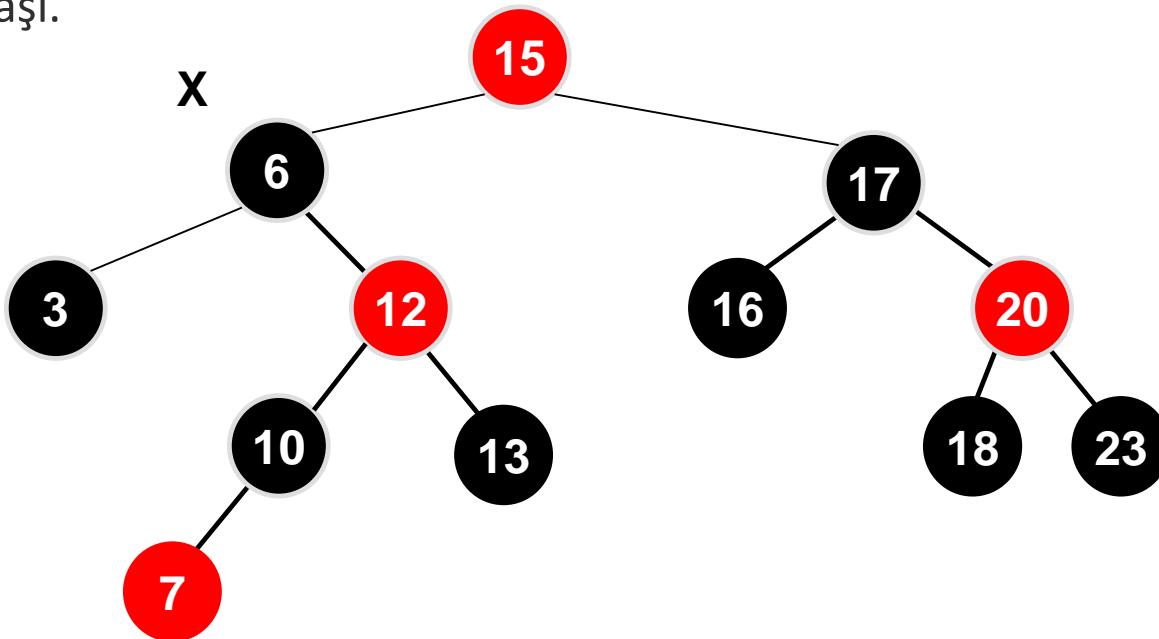
## Adım 3-4

- **Adım 3-** Sonunda silinecek düğümü bul ve sil
- **Adım 4-** Kökü yeniden renklendir (Siyah yap).
- Örnek: R-B ağacından **10** değerini siliniz.
  - **Adım 1:** A1-Kök iki tane siyah çocuğa sahip, kökü kırmızı yap X' 6 ya taşı



# Örnek

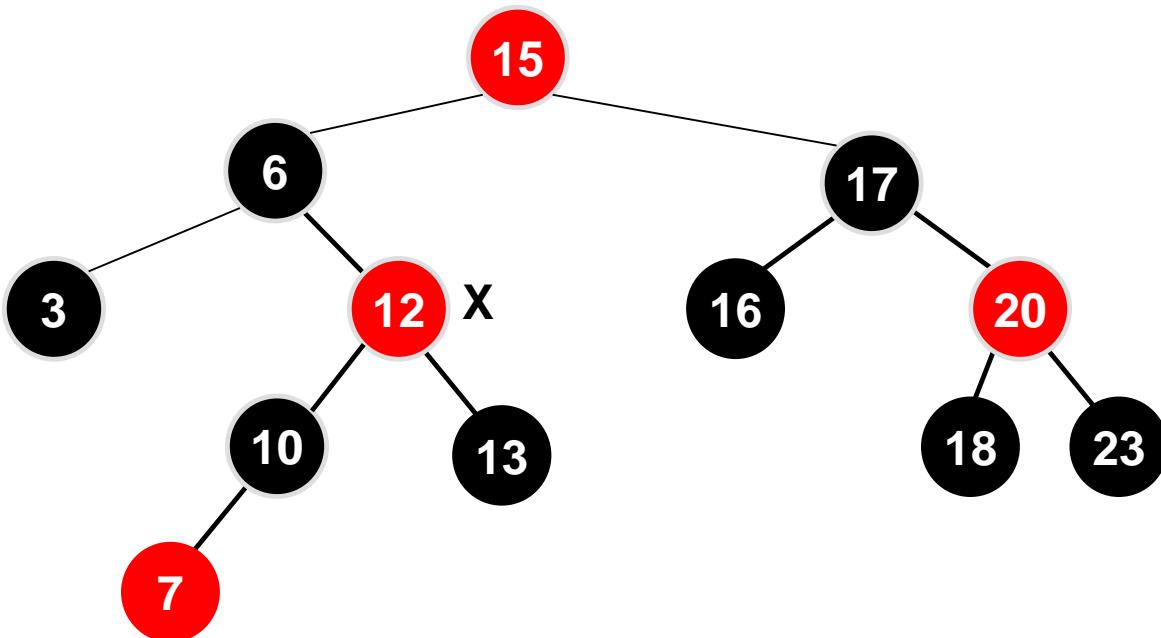
- Adım 1: Kök iki tane siyah çocuğa sahip, kökü kırmızı yap X' 6 ya taşı.



- X'in çocuklarında biri kırmızı (Adım 2). X'i uygun çocuğa taşı, yeni X(12) aynı zamanda kırmızı (Adım 2-B1)

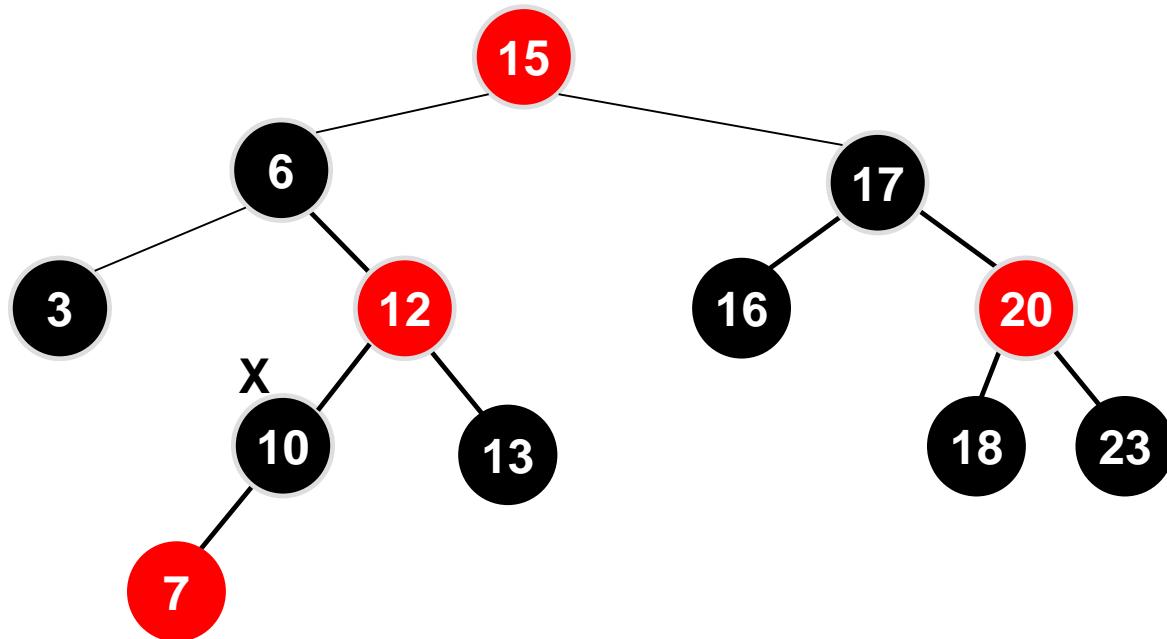
# Örnek

- X taşımaya devam et X (10) .



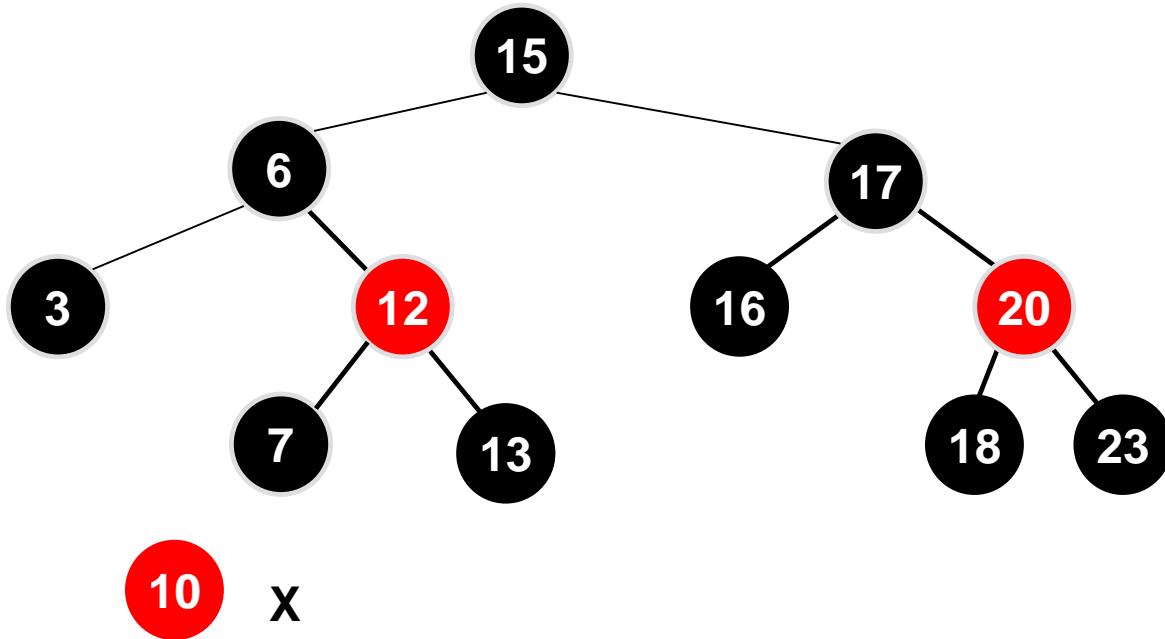
# Örnek

- Silinecek düğüm bulundu. **Adım 3**, çocuğu ile yer değiştirir ve sil.



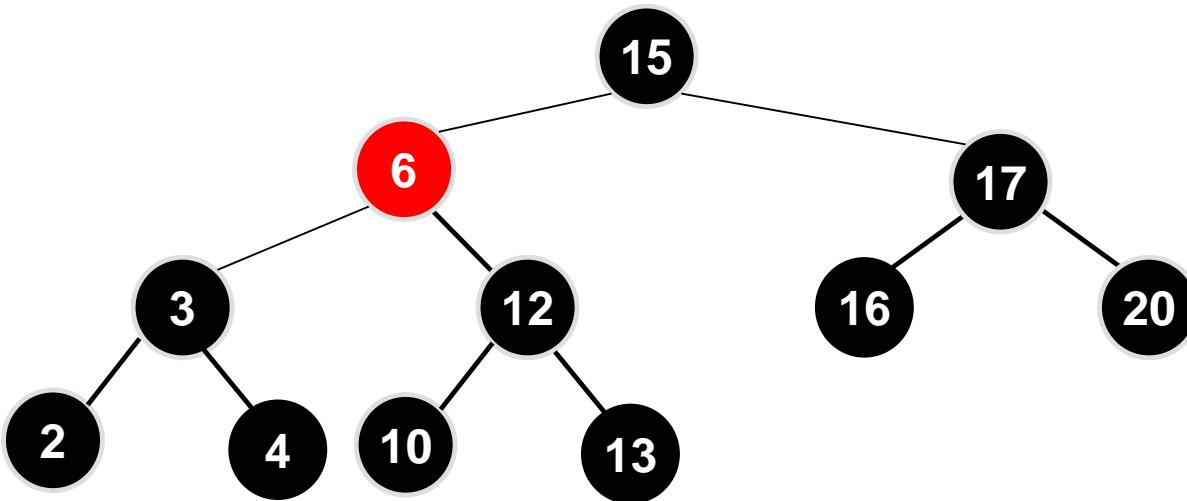
# Örnek

- Adım 4: Kök değerini siyah yap



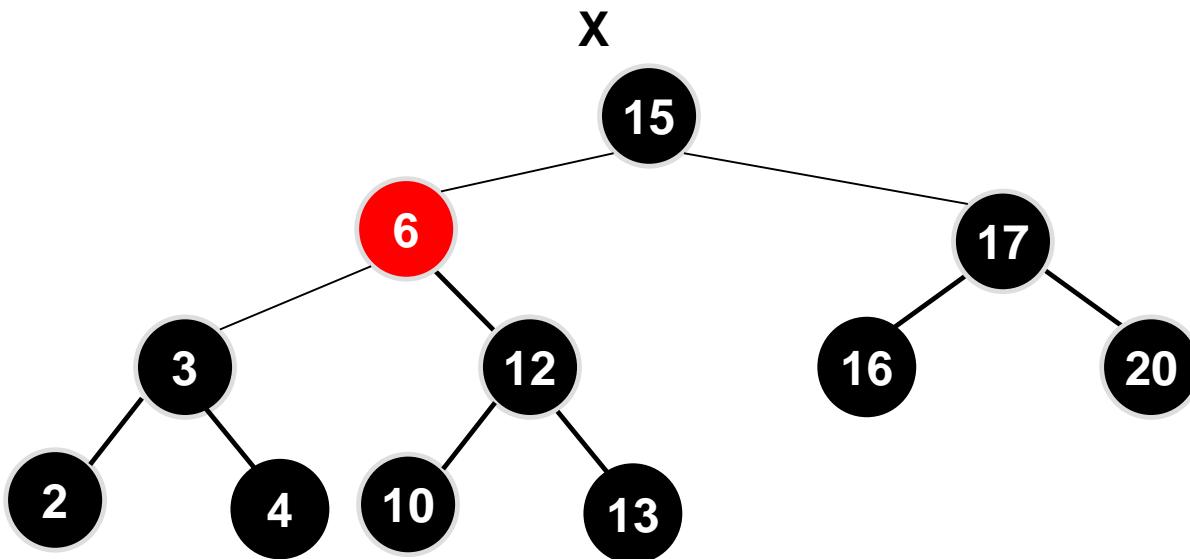
# Örnek

- Örnek: R-B ağacından 10 değerini siliniz.



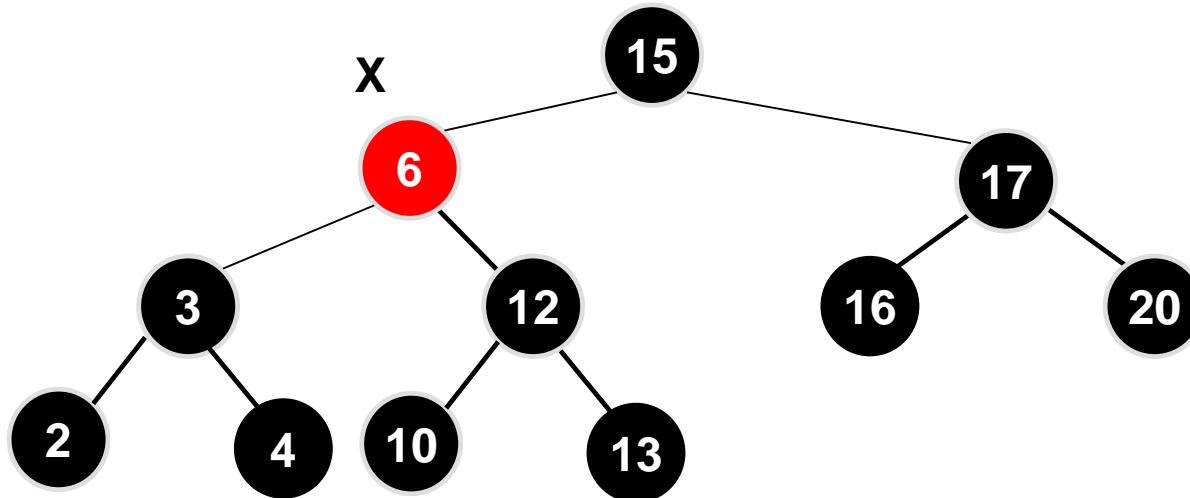
# Örnek

- **Adım 1:** A2-Kök bir tane siyah çocuğa sahip, X=kök yap
- **Adım 2-B:** X en az bir tane kırmızı çocuğa sahip. X uygun çocuğa taşı.



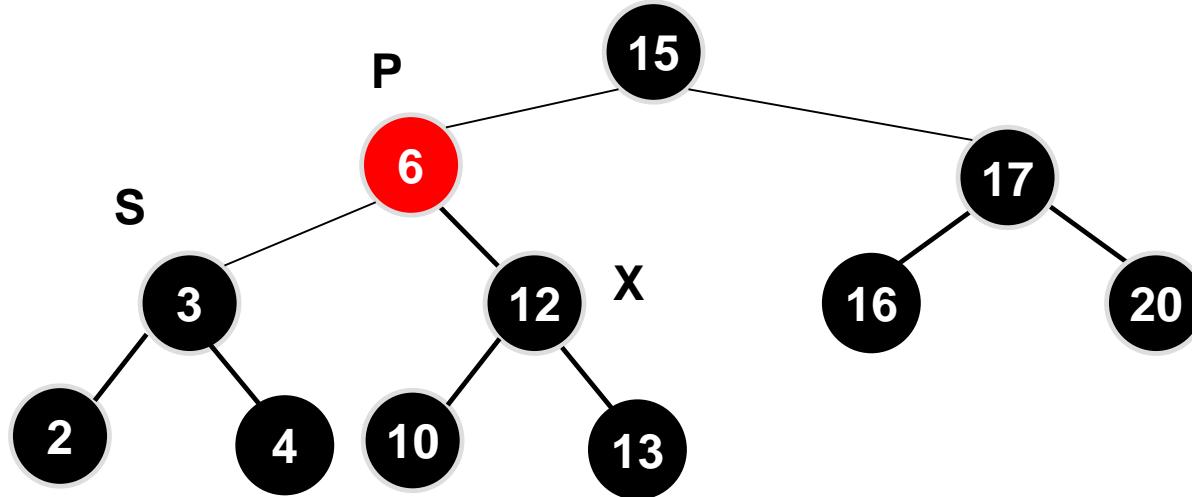
# Örnek

- Adım 2-B1: Yeni X (6) kırmızı olduğundan uygun çocuk  
düğümeye taşımaya devam et X(12)



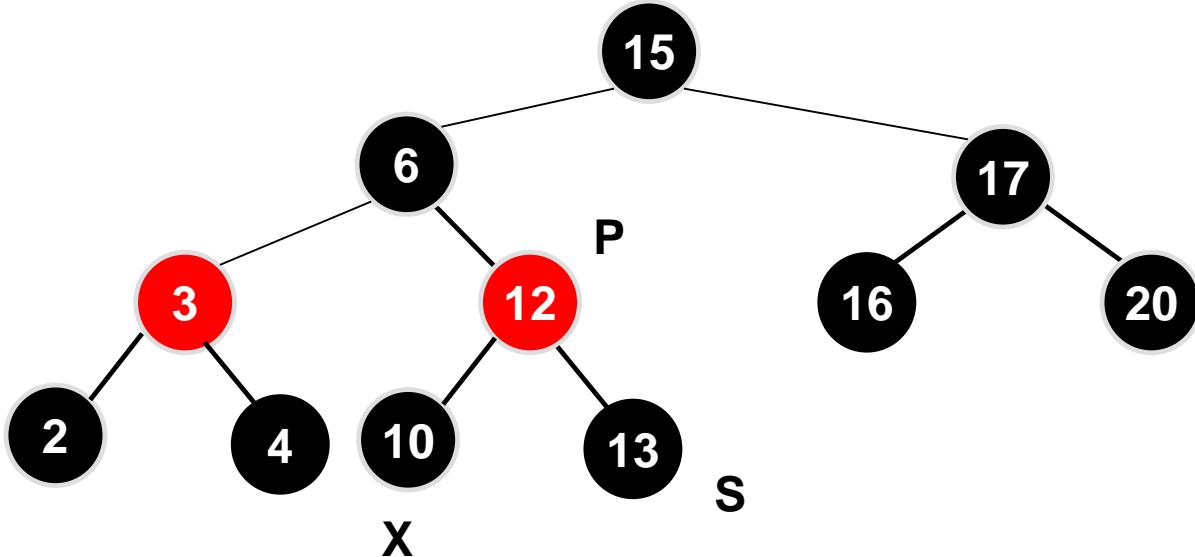
# Örnek

- X iki tane siyah düğüme sahip. X' in kardeşi S de 2 tane siyah düğüme sahip (Adım 2-A1).
- **Adım 2-A1:** S'in her iki çocuğu da siyah ise, X, P, S yeniden renklendir.
- X'i 10'na taşı.



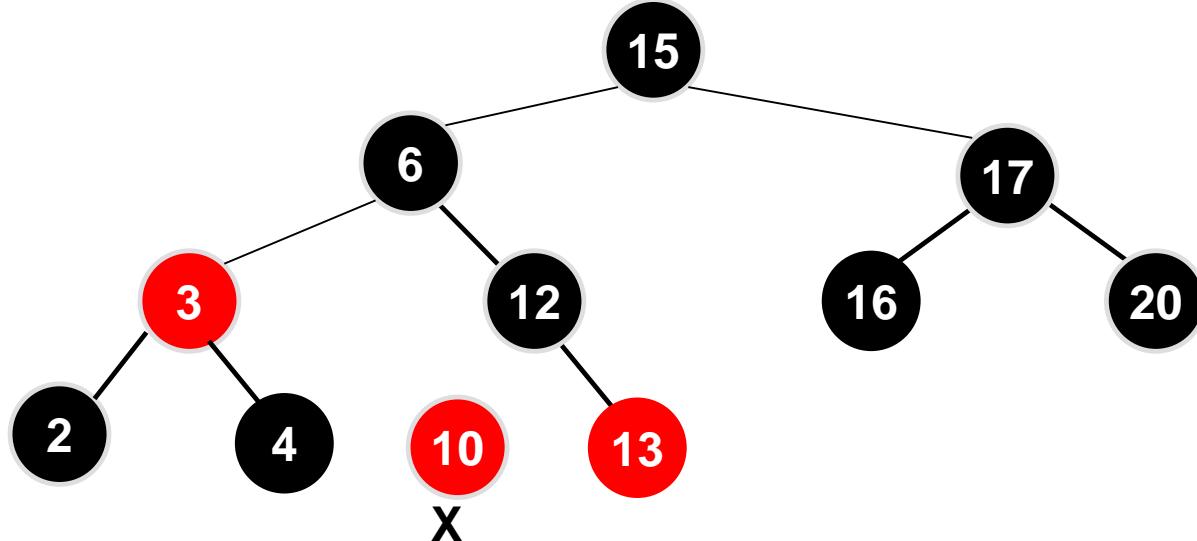
# Örnek

- X, hedef olan yaprak düğüm olduğundan silinebilir fakat rengi siyah Adım 2 ye git.
- X iki tane siyah düşüğe sahip. X'in kardeşi S de 2 tane siyah düşüğe sahip (Adım 2-A1).
- **Adım 2-A1:** S'in her iki çocuğu da siyah ise, X, P, S yeniden renklendir.



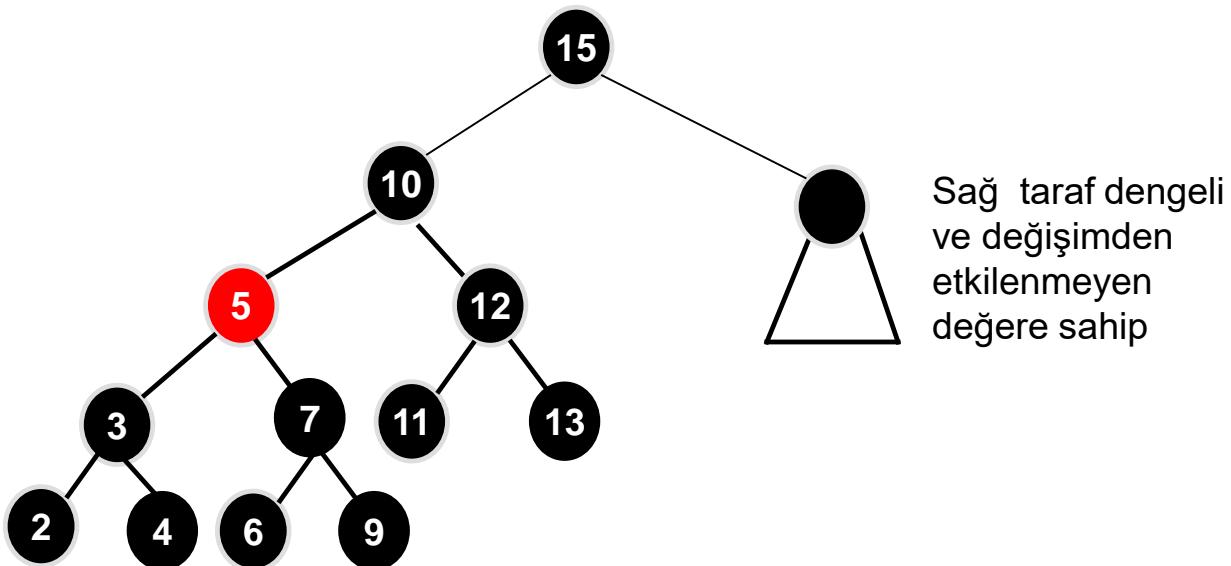
# Örnek

- Adım 3: Kırmızı olan 10 düğümü artık silinebilir.
- Adım 4: Kökü siyah yap.



# Örnek

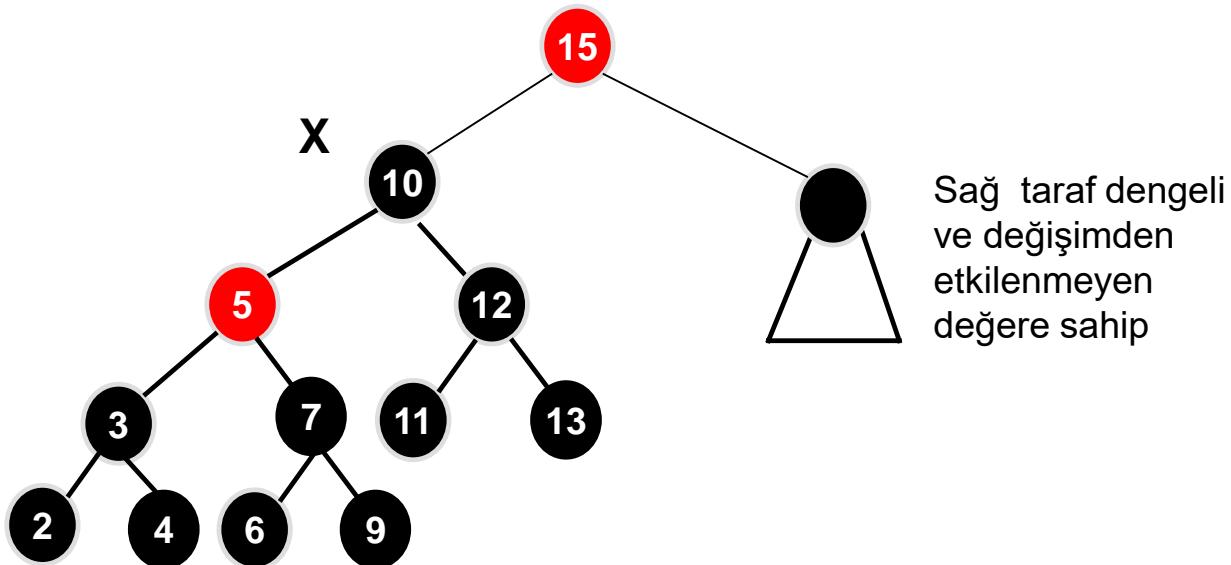
- Örnek: R-B ağacından 11 değerini siliniz.



- Adım 1: Kök iki tane siyah çocuğu sahip. Kökü kırmızı yap. X'i uygun çocuğu taşı (X(10)).

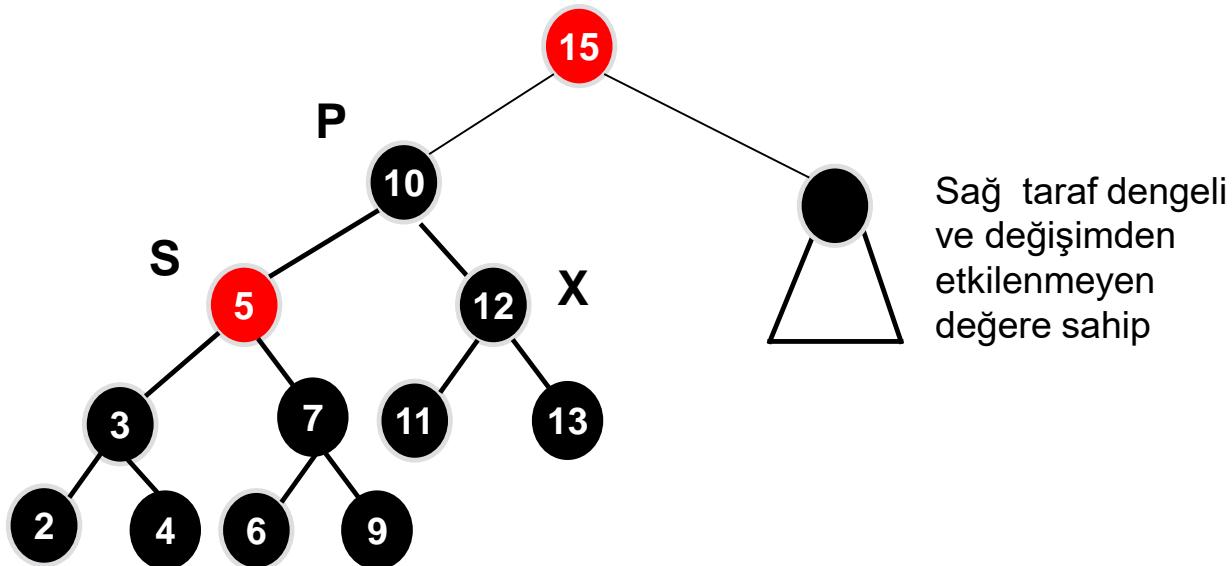
# Örnek

- X kırmızı bir çocuğa sahip (Adım 2). X'i 12 ye taşı .

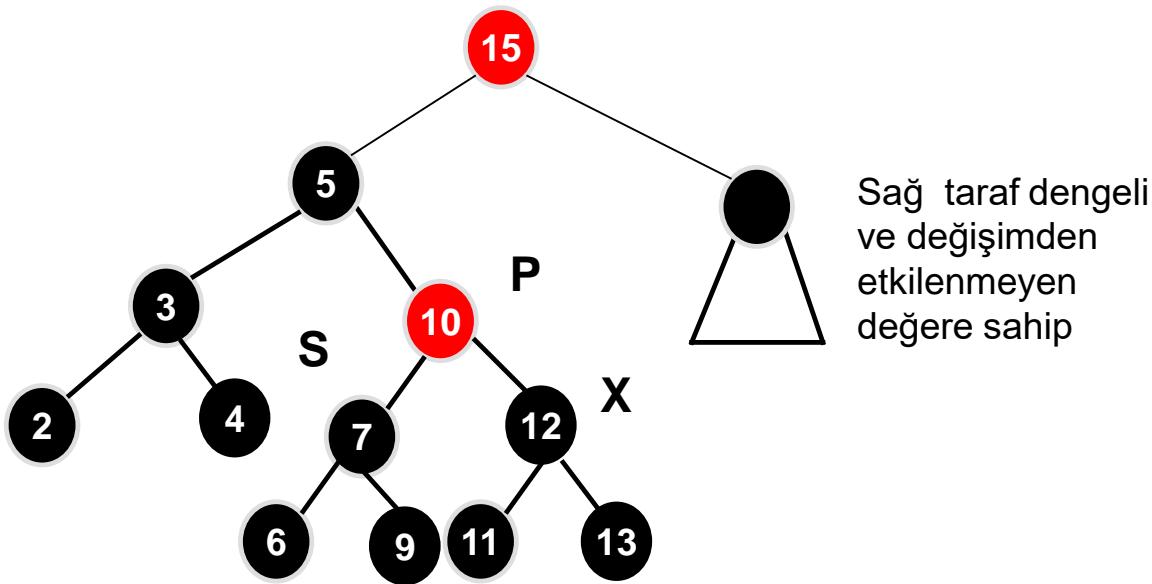


# Örnek

- Adım 2-B2: Eğer yeni X siyah ise ( P Siyah, S kırmızı), S'yi P' nin etrafında döndür P ve S'yi yeniden renklendir ve Adım 2'ye git.



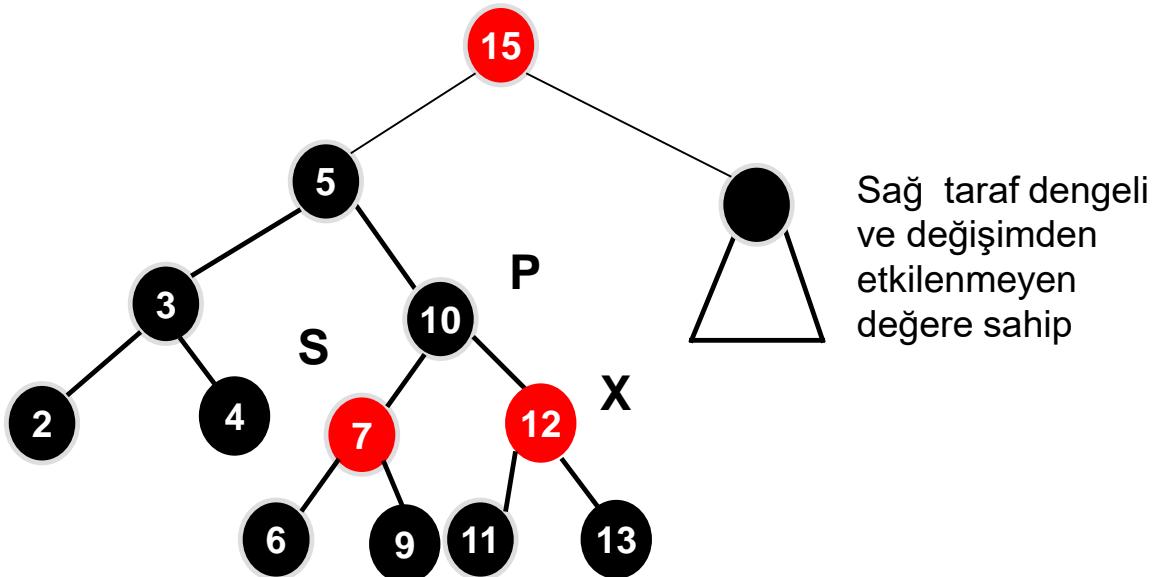
# Örnek



- X ve S siyah çocuklara sahip (Adım 2-A1). X,P ve S' yi yeniden renklendir.

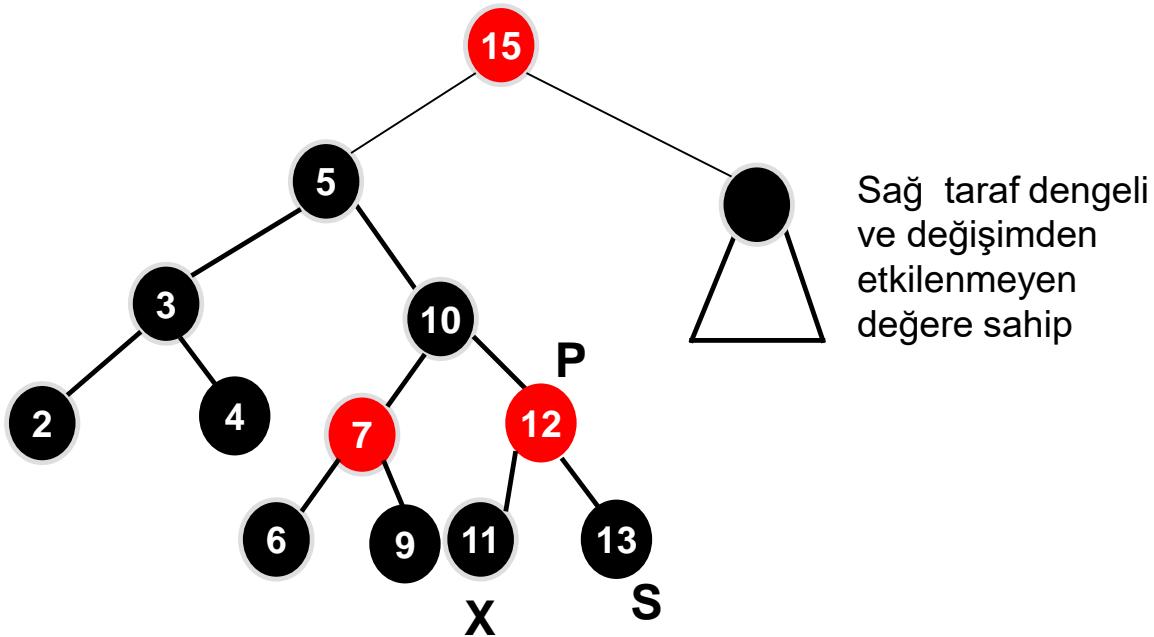
# Örnek

- X, P ve S'yi yeniden renklendir. X'i 11 taşı.



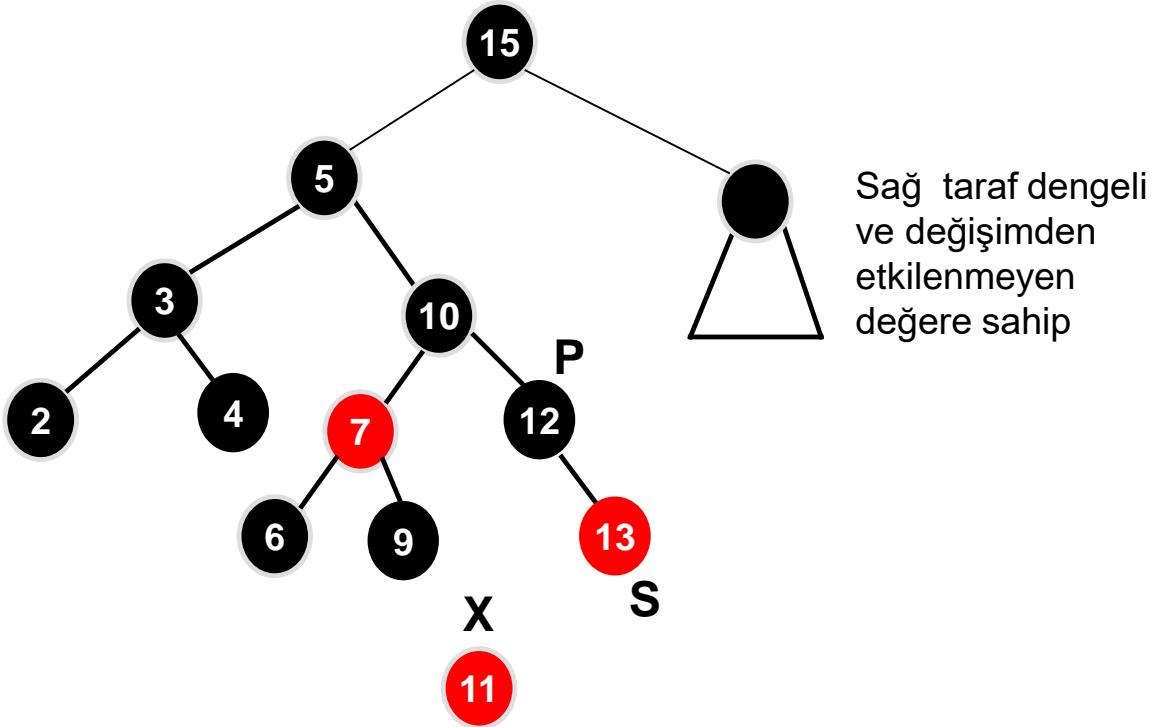
# Örnek

- X'i 11 taşı. X siyah olduğundan silinemez. X ve S siyah çocuklara sahip (Adım 2-A1). X,P ve S' yi yeniden renklendir.



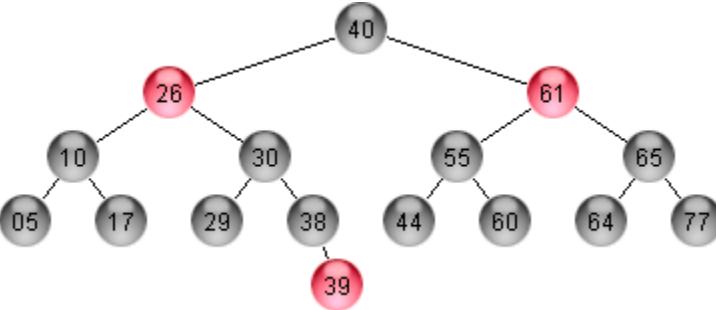
# Örnek

- Adım 3: X'i sil.
- Adım 4: Kökü siyah yap

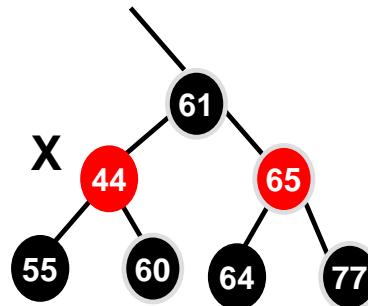
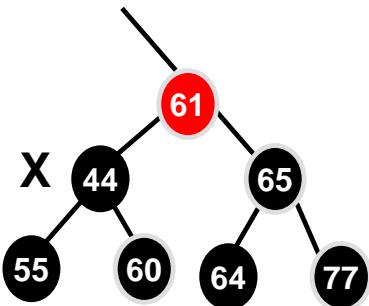
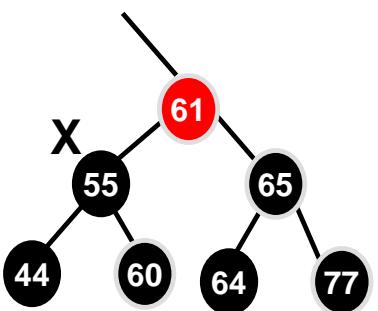


# Örnek

- Örnek: Aşağıda verilen red-black ağacından 55 nolu düğümü siliniz.

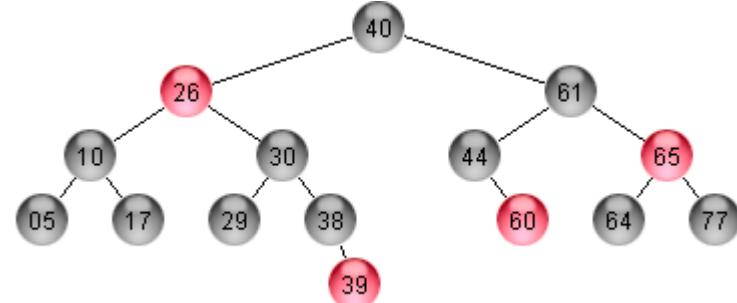
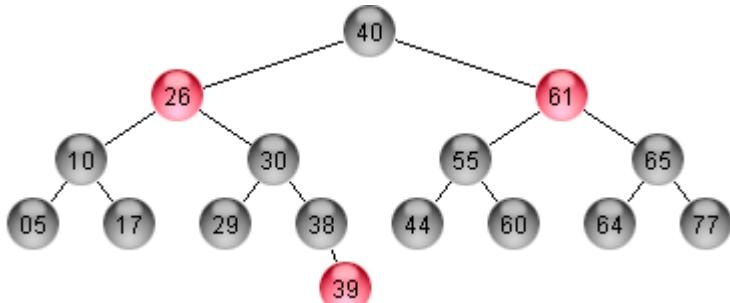


- Silinecek 55 nolu düğüm ile yerine gelecek düğümü (44) yer değiştir.
- Adım1: Kök düğümün çocukları kırmızı, kökü kırmızı yap  $X=\text{kök}$ .  $X'$  i uygun çocuğa taşı  $X(61)$ . Yeni durum Adım 2-A1'i uygula.

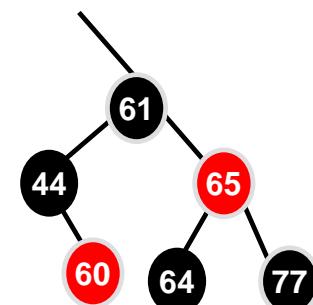
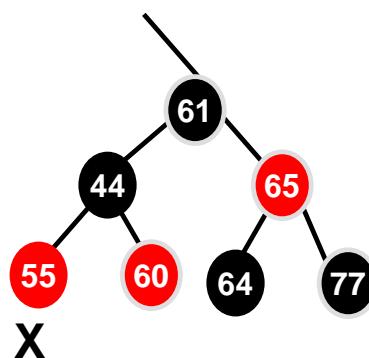
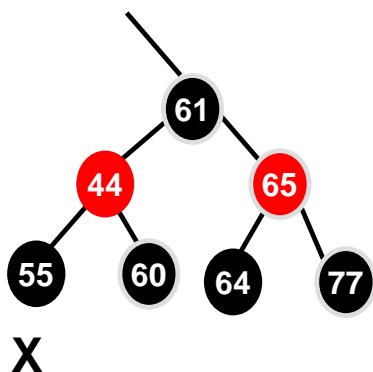


# Örnek

- Örnek: Aşağıda verilen red-black ağacından 55 nolu düğümü siliniz.

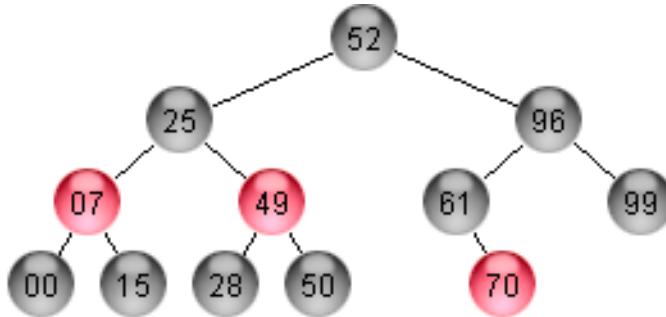
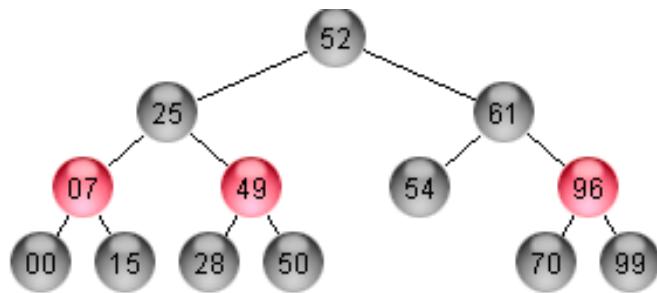


- X'i taşı X(55). Yeni durum Adım 2-A1'i uygula
- Adım 3 (X'i sil) ve Adım 4 uygula

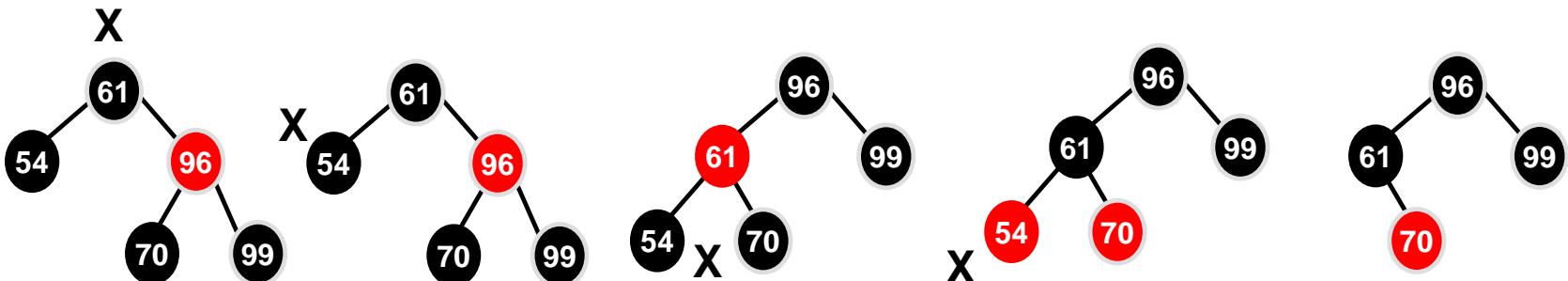


# Örnek

- Örnek: Aşağıda verilen red-black ağacından 54 nolu düğümü siliniz.

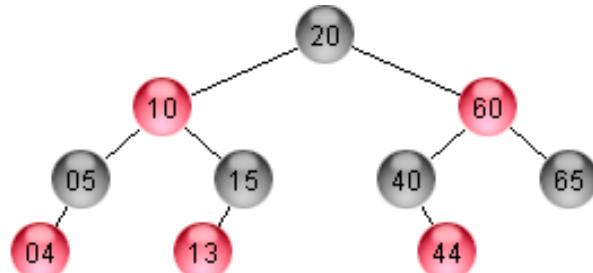
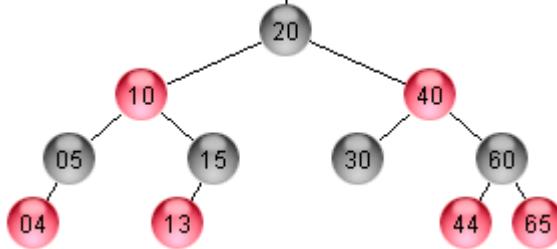


- Adım 1'i uygula X'i taşı (X(61)). Yeni durum Adım 2-B2'yi uygula (X(54))
- Yeni durum Adım 2-A1'i uygula. Adım 3 ve 4'ü uygula

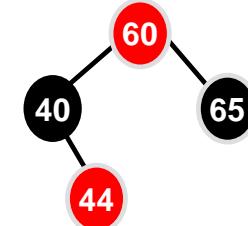
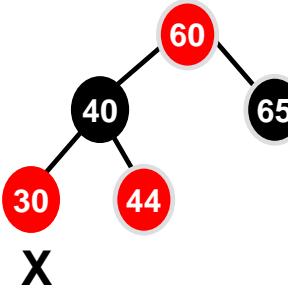
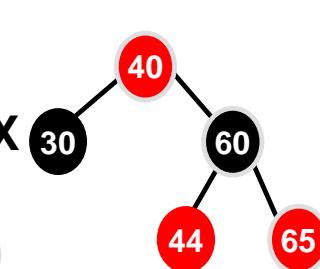
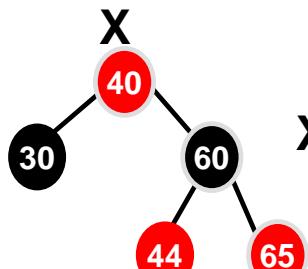


# Örnek

- Örnek: Aşağıda verilen red-black ağacından 30 nolu düğümü siliniz.

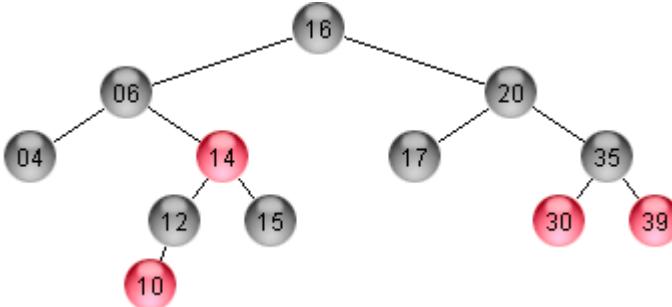


- Adım1'i uygula  $X=\text{kök}$ .  $X$ 'i ,i uygun çocuğa taşı  $X(30)$ . Yeni durum Adım 2-A3'ü uygula. Adım 3 ve 4'ü uygula.

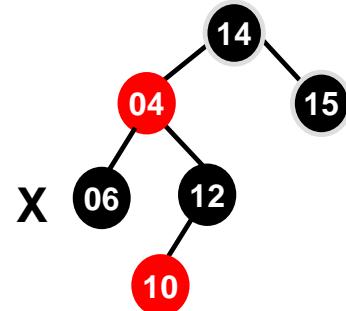
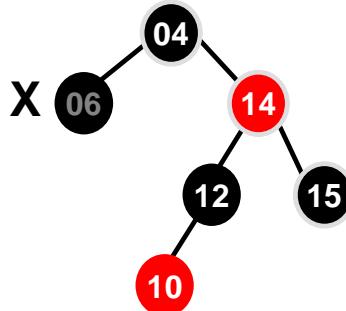
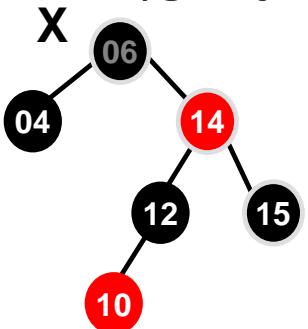


# Örnek

- Örnek: Aşağıda verilen red-black ağacından 6 nolu düğümü siliniz.

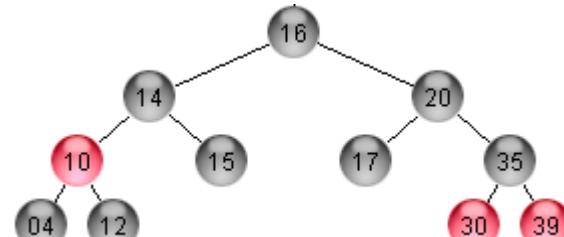
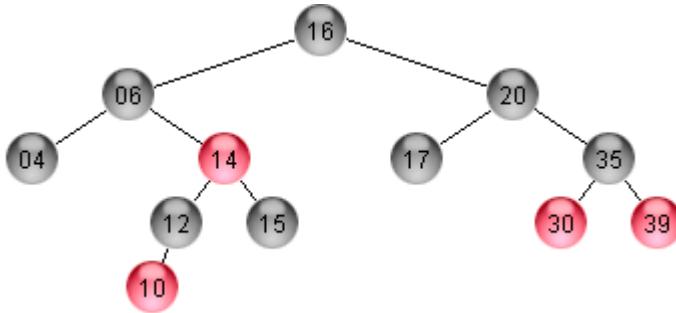


- 4 ile 6 yi yer değiştir.
- Adım 1' uygula
- X'i uygun çocuğa taşı X(4). Yeni durum Adım 2-B
- X'i uygun çocuğa taşı X(6). Yeni durum Adım 2-B2'yi uygula.

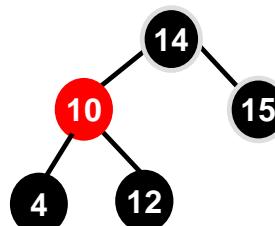
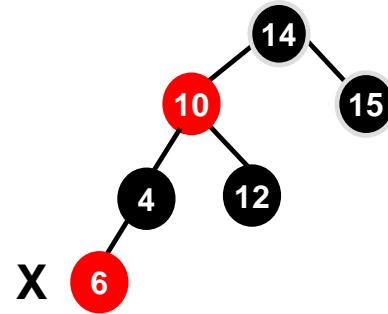
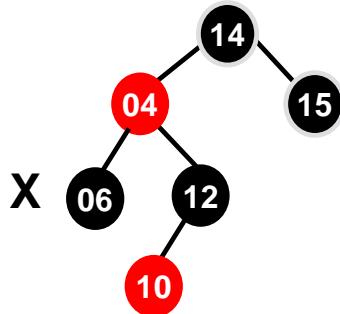


# Örnek

- Örnek: Aşağıda verilen red-black ağacından 6 nolu düğümü siliniz.

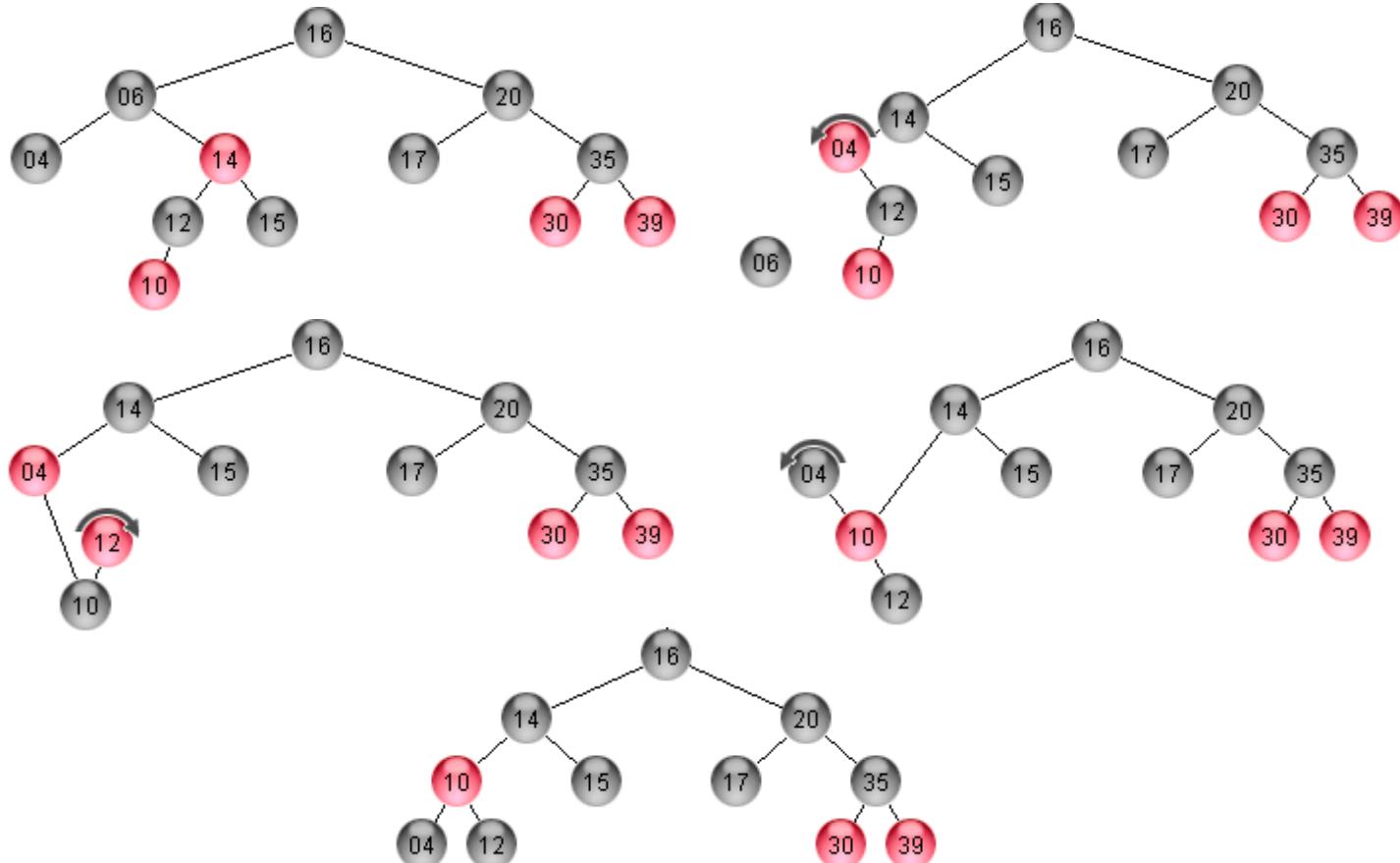


- X'i uygun çocuğu taşı X(6). Yeni durum Adım 2-B2'yi uygula.
- Yeni durum Adım 2-A2'i uygula. Adım 3 ve 4'ü uygula



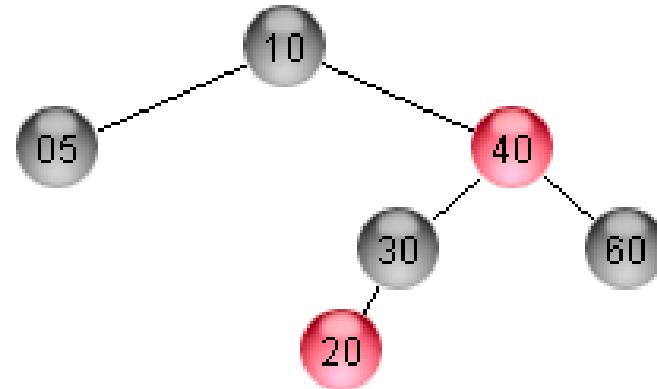
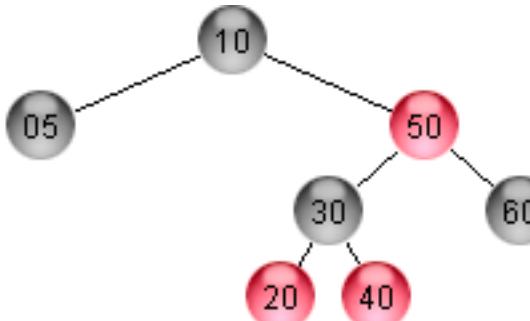
# Örnek

- Örnek: Aşağıda verilen red-black ağacından 6 nolu düğümü siliniz.



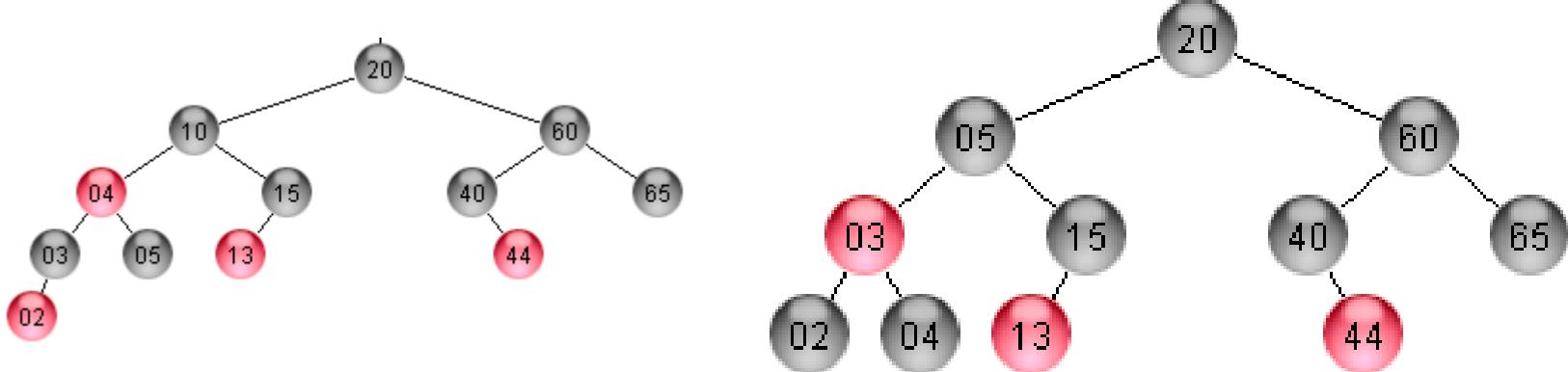
# Red-Black Tree: Deletion

- Silinecek düğüm kırmızı, siyah çocukları var ise renk değişmez. (50 silindi)



# Red-Black Tree: Deletion

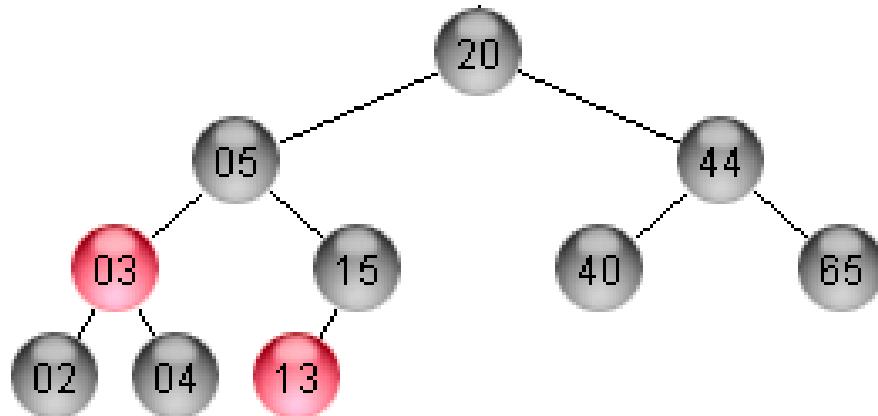
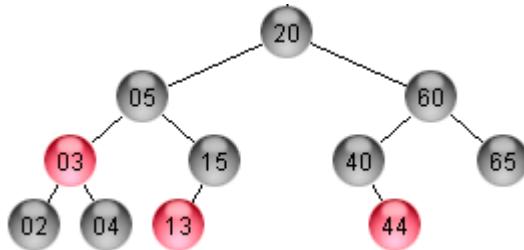
- Örnek: 10 değerini siliniz.



- 4B, 3R, 2B, 4 Sağ

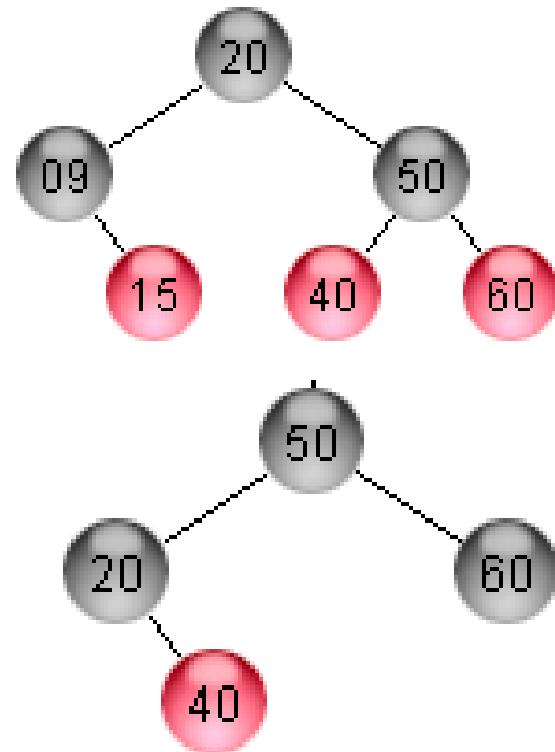
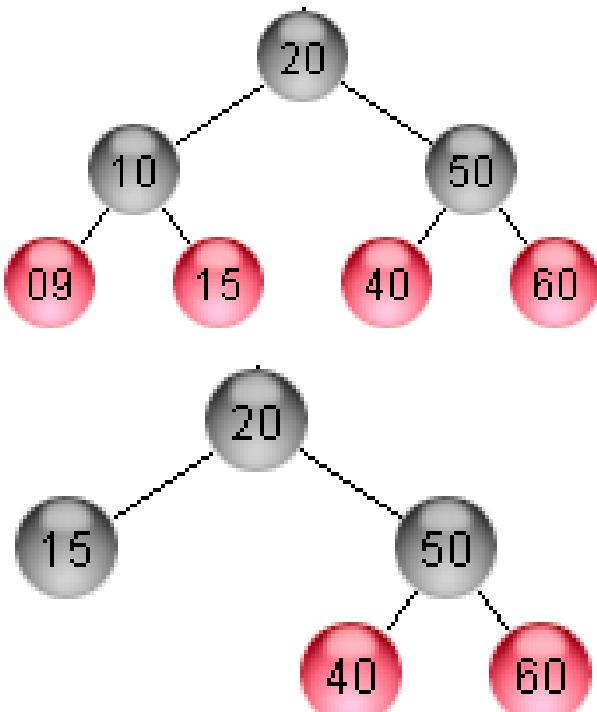
# Red-Black Tree: Deletion

- Örnek: 60 değerini siliniz



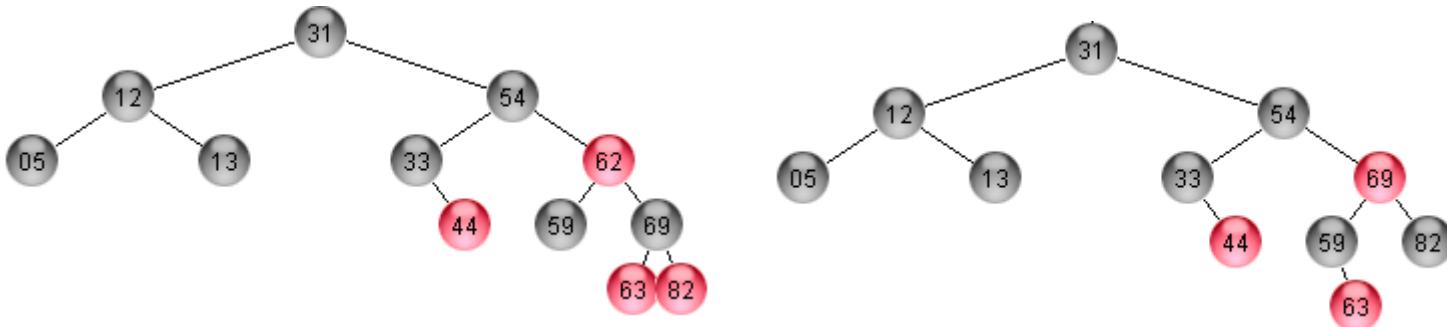
# Red-Black Tree: Deletion

- Örnek
- Sırasıyla 10, 9, 15, silindi.



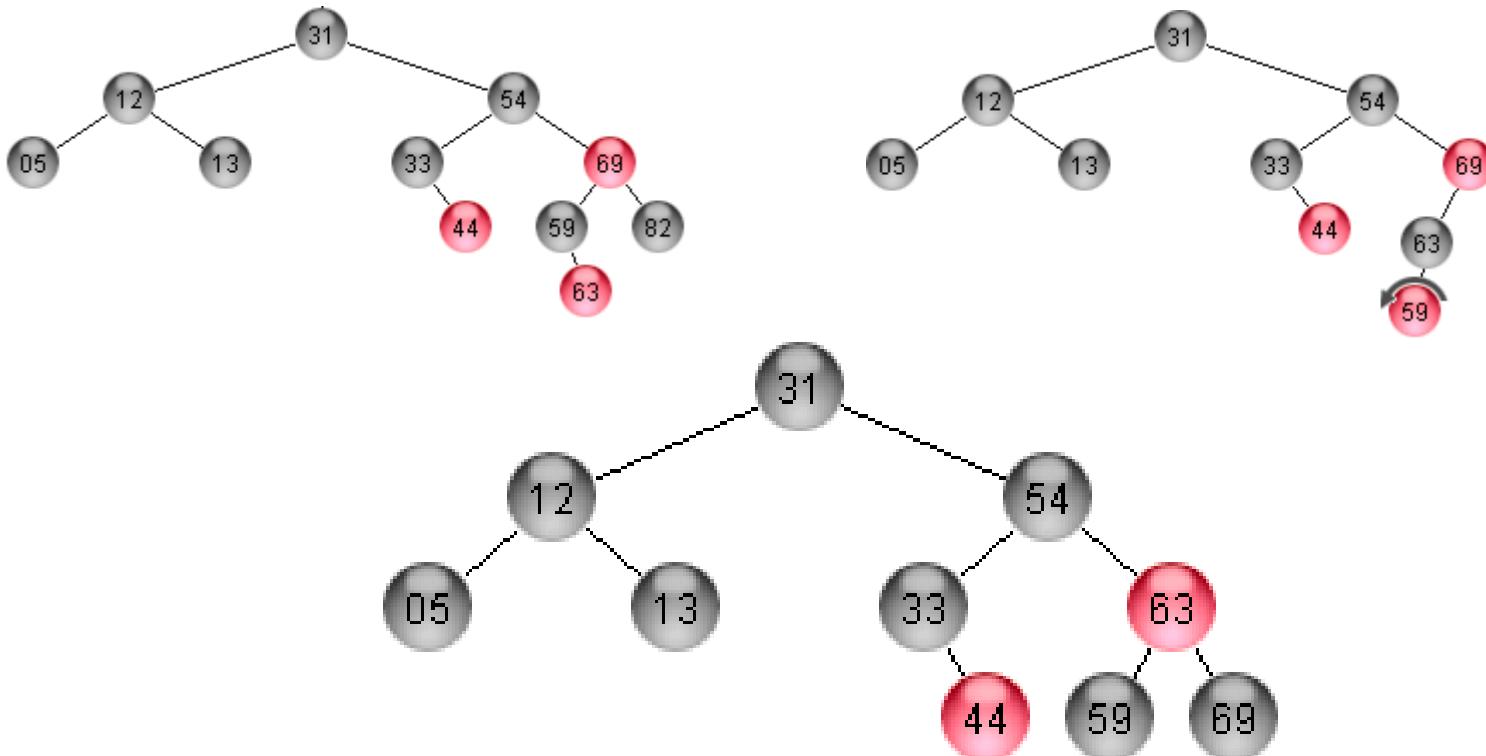
# Red-Black tree Silme

- Örnek: 62 silinecek



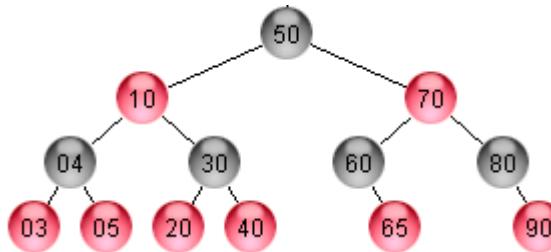
# Red-Black tree Silme

- Örnek: 82 silinecek



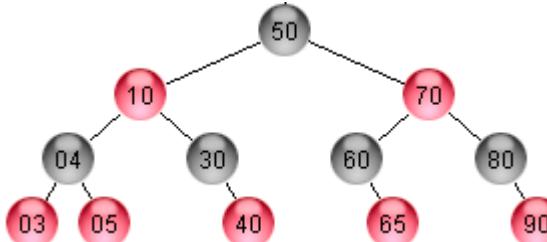
# Red-Black tree Silme

- (Standart olması açısından silinen düğüm yaprak değil ise genelde soldaki en büyük düğümü alacağız.)



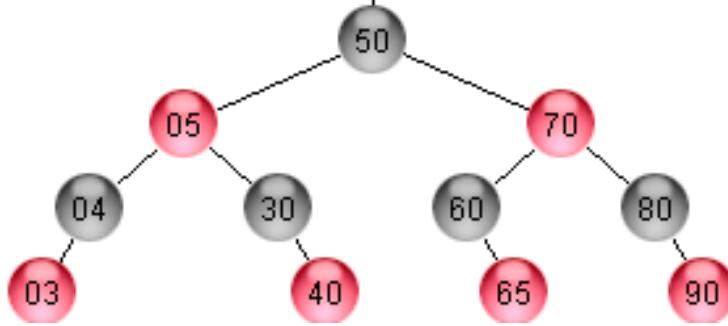
- Sil:20,10, 70, 30,50

- 20 silindi

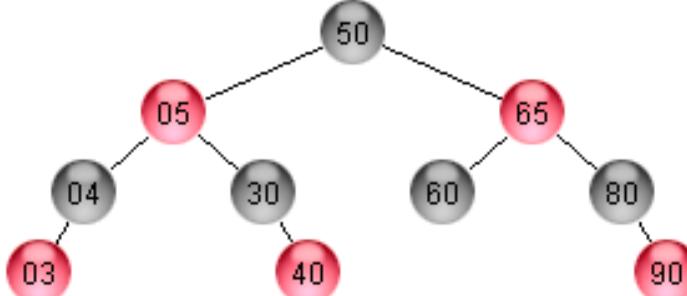


# Red-Black tree Silme

- Sil:10,70,30,50,
- 10 silindi

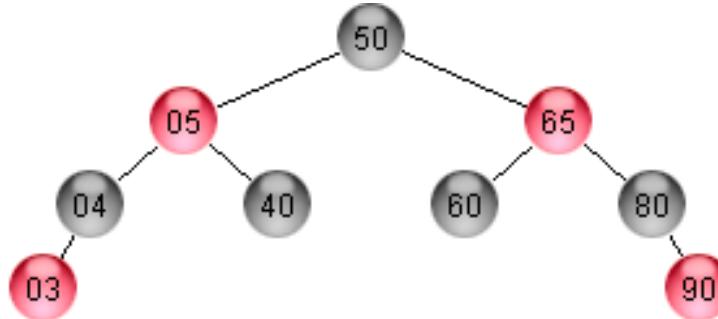


- 70 silindi

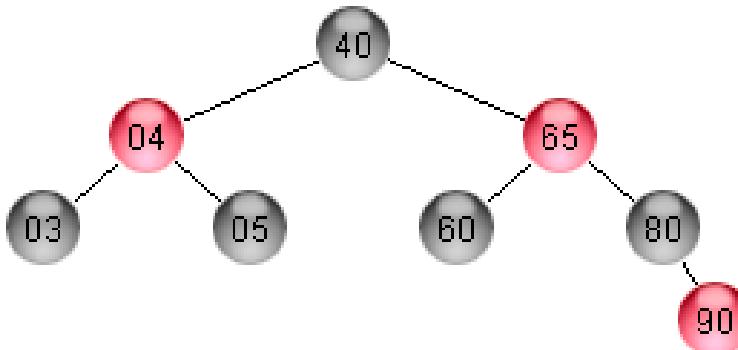


# Red-Black tree Silme

- Sil:30,50,



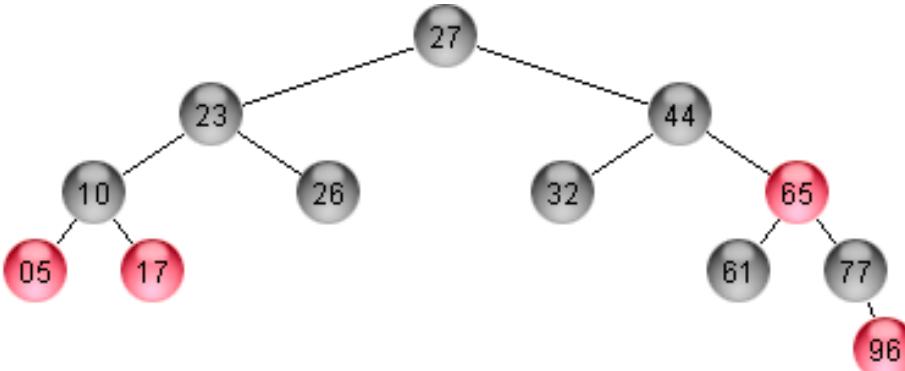
- 30 silindi.



- 50 silindi.

# Red-Black tree Silme

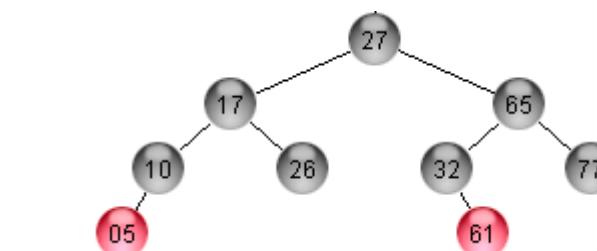
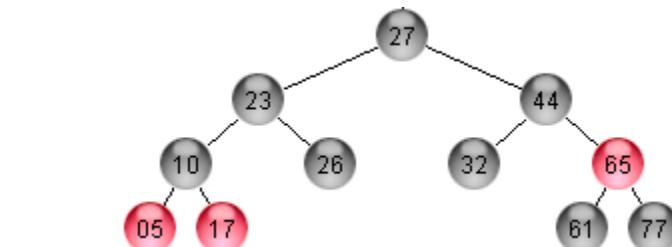
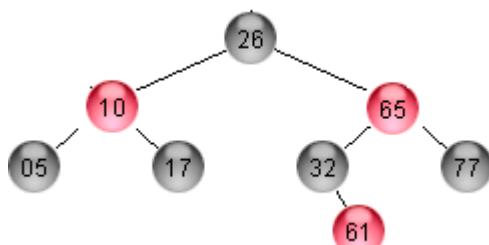
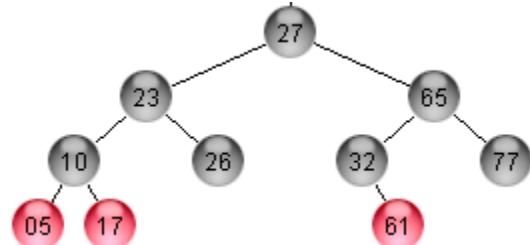
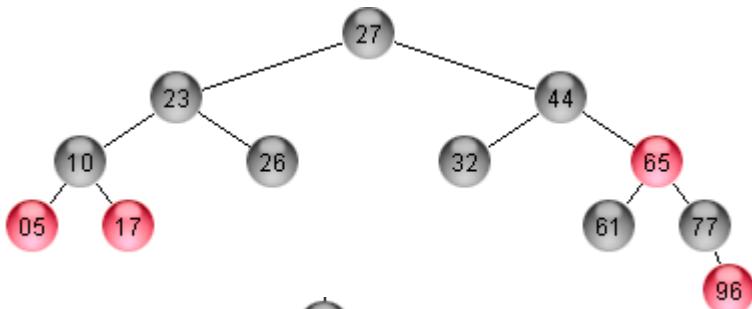
- Aşağıda verilen ağaçta sırasıyla 96, 44, 23, 27 düğümlerini silerek aşağıdaki soruları cevaplayınız.



- (Doğru değer dışında yazılan değerler olursa sonuç yanlış olarak kabul edilecektir.)
- A) Hangi düğümler kırmızı renge sahiptir?
- B) Kök düğümün değeri nedir?

# Red-Black tree Silme

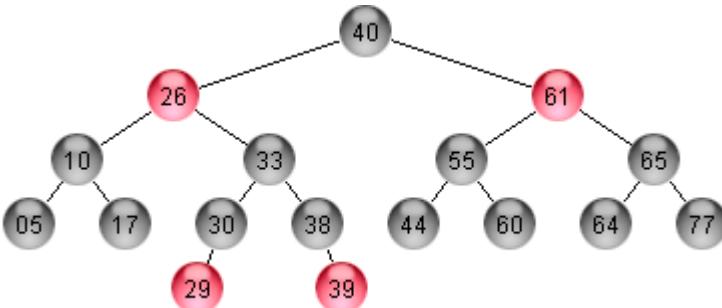
- 96, 44, 23, 27



A)10,61,65 B) 26

# Red-Black tree Silme

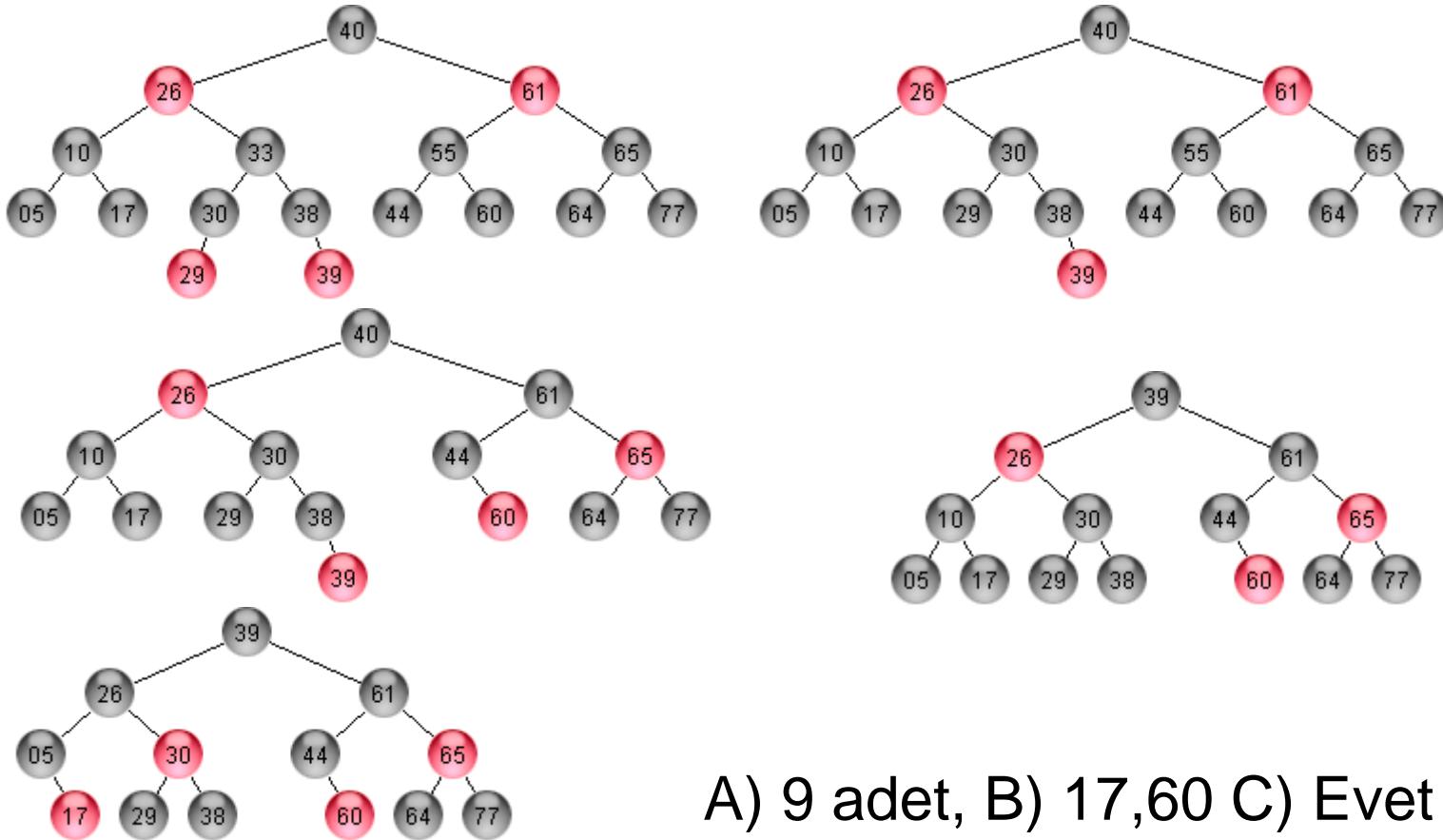
- Aşağıda verilen ağaçta sırasıyla 33, 55, 40, 10 düğümlerini silerek aşağıdaki soruları cevaplayınız.



- (Doğru değer dışında yazılan değerler olursa sonuç yanlış olarak kabul edilecektir.)
- A) Kaç adet Siyah renge sahip düğüm vardır?
- B) Yaprakta bulunan hangi düğümler kırmızı renge sahiptir? Kök düğümün değeri nedir?
- C) Ağacın son hali AVL ağacı olarak dengede midir?
- Dengede değil ise dengeleyiniz.

# Red-Black tree Silme

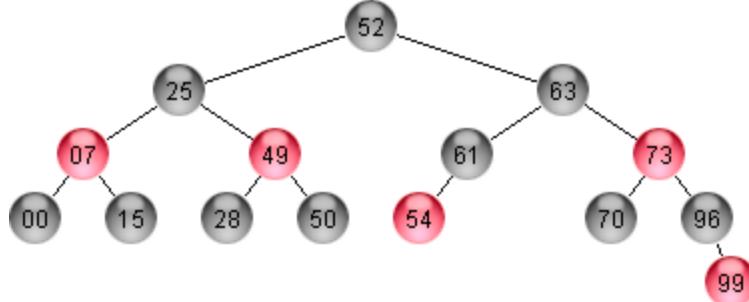
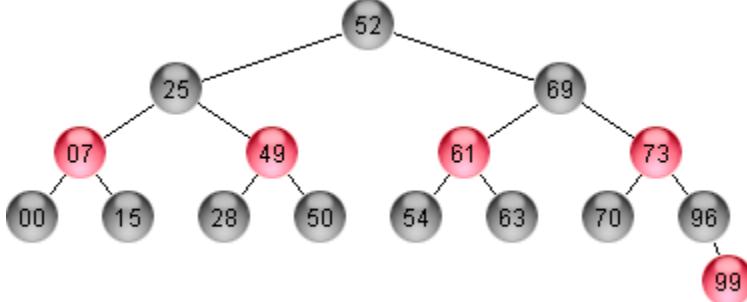
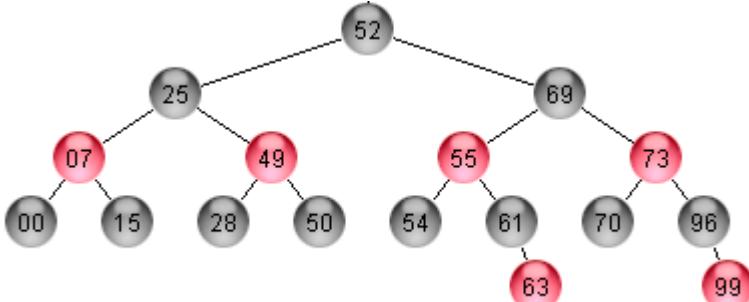
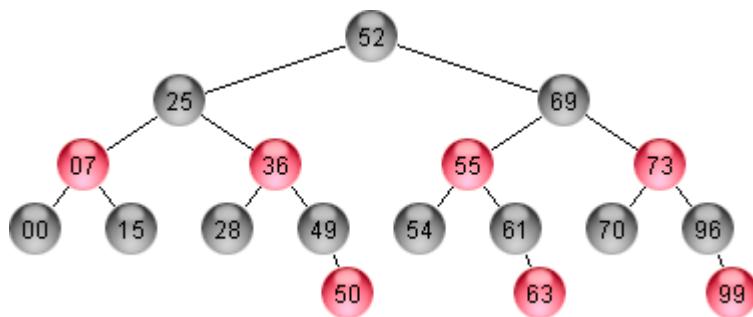
o 33, 55, 40, 10



A) 9 adet, B) 17,60 C) Evet

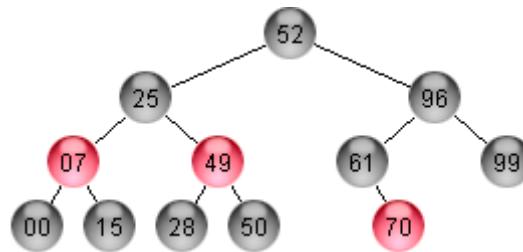
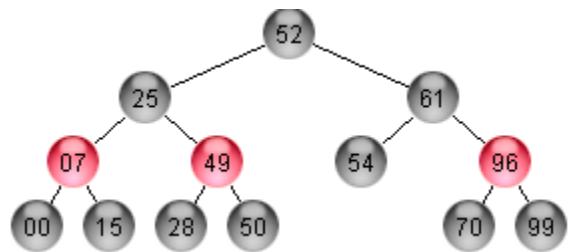
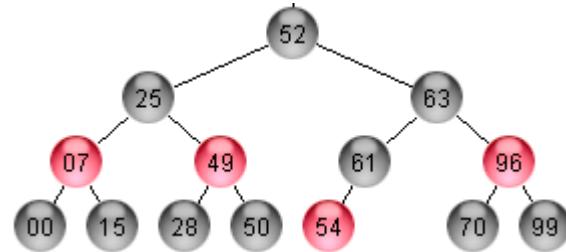
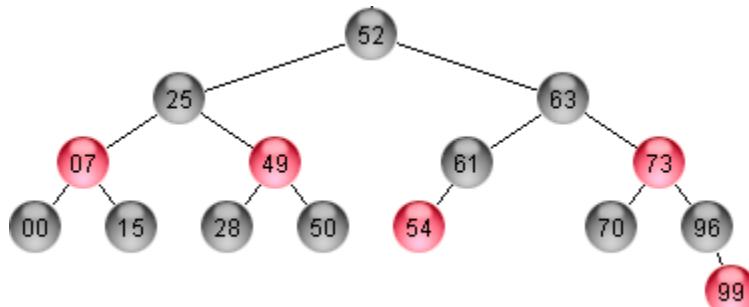
# Red-Black tree Silme

- Silinecek değerler :36,55,69



# Red-Black tree Silme

- Silinecek değerler: 73,63,54



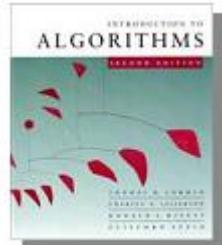
# **10.Hafta**

# **Dinamik**

# **Programlama**

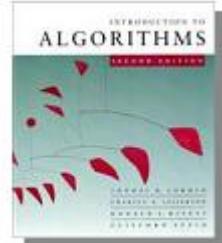
- En uzun ortak altdizi
- En uygun altyapı
- Altproblemlerin çakışması

# Dinamik Programlama (Dynamic programming)

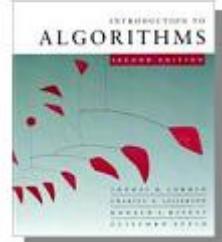


- Fibonacci sayıları örneği
- Optimizasyon problemleri
- Matris çarpım optimizasyonu
- Dinamik programlamaın prensipleri
- En uzun ortak alt sıra (Longest common subsequence)

# Dinamik programlama (Dynamic programming)

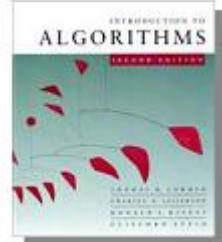


- Tarihçe:
- Richard Bellman tarafından 1950' li yıllarda bulunmuştur.  
"Programming" kelimesi aslında "planning" anlamında  
kullanılmaktadır (bilgisayar programlama değil)
- Dinamik Programlama (DP), böl ve yönet yöntemine benzer  
olarak alt problemlerin çözümlerini birleştirerek çözüm elde  
eden bir yöntemdir.



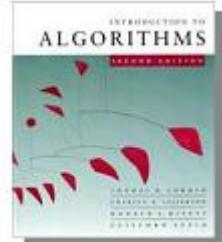
# Dinamik programlama

- Böl ve yönet yönteminde alt problemlerin bağımsız olması gerekiyor; bunun tersine DP' da alt problemler bağımsız değilse de, DP uygulanabilir.
- DP her alt problemi bir kez çözer ve çözümleri bir tabloda saklar ve bu şekilde aynı alt problemin birden fazla ortaya çıkma durumunda her seferinde tekrar çözüm yapmaktansa, tabloda saklamış olduğu değeri problemde yerine koyar. Bu şekilde işlemlerin çözümünün hızlanması sağlanmış olur.
- DP, genelde en iyileme (optimizasyon) problemlerine uygulanır. Bu tip problemlerin birden fazla çözümü olabilir. Amaç bu çözümler içinde en iyisini bulmaktır.



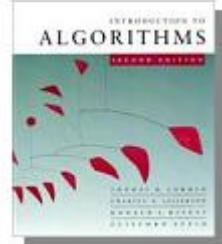
# Dinamik programlama

- Dinamik programlama uygulamalarında temel olarak 3 teknikten faydalanyılır.
  - 1- Çözümü aynı olan alt-problemler
  - 2- Büyük bir problemi küçük parçalara bölmek ve bu küçük parçaları kullanarak baştaki büyük problemimizin sonucuna ulaşmak
  - 3- Çözdüğümüz her alt-problemin sonucunu bir yere not almak ve gerektiğinde bu sonucu kullanarak aynı problemi tekrar tekrar çözmeyi engellemek.
- Genel olarak dinamik programlama kullanırken aşağıdaki dört adımı göz önüne almamız gereklidir ve dört adım DP' nin temelini oluştururlar.



# Dinamik Programlama

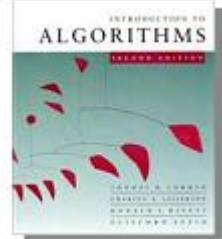
- 1 . Optimal çözümün yapısının karakteristiği ortaya çıkarılmalı.
- Optimal altyapısının gösterimi -bir optimal sonuç, alt problemlerin optimal sonuçlarını içerir
  - Bir problemin çözümü:
    - Birçok olası çözümden birisi seçilir
    - Seçilen çözüm bir veya daha fazla alt problemin çözümünün sonucudur.
  - Tüm çözümün optimal olması için alt problemlerin çözümlerinin de optimal olması zorunludur



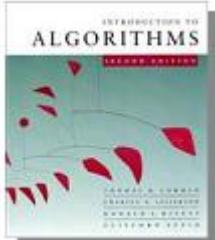
# Dinamik Programlama

- 2. Özyinelemeli olarak optimal çözümün değerini tanımlamalı.
  - Optimal çözümün değeri için bir öz yineleme (recurrence) ifade yazılır.
  - $M_{\text{opt}} = \text{Min}_{k \text{'nın tüm değerleri için}}$
- 3. Optimal çözümün değeri bottom-up(alttan-üste) yaklaşımıyla çözülür, böylece alt problemlerin daha önce çözülmesi gereklidir (veya memoization kullanılır)
  - Eğer bazı alt problemlerin çözümlerinin tutulmamasıyla alan gereksinimi azaltılabiliyorsa bu alt çözümlerin sonuçları hafızaya alınmamalıdır

# Dinamik Programlama



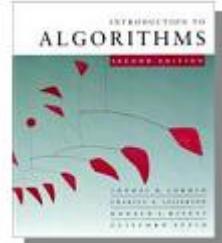
- 4. Elde edilen bilgilerden (optimal çözüme ulaşmak için yapılan seçimlerin sırasıdır) optimal çözüm yapılandırılır.
- Eğer problemin bir tane çözümü varsa, bu durumda bu adım atlanabilir.



# Algoritma tasarım teknikleri

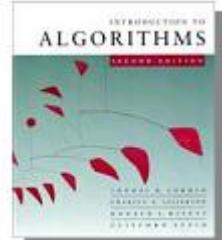
- Şimdiye kadar görülenler:
- Iterative (brute-force) algoritmalar
  - Örnek, insertion sort
- Diğer veri yapılarını kullanan algoritmalar
  - Örnek, heap sort
- Divide-and-conquer algoritmaları
  - Örnek, binary search, merge sort, quick sort

# Böl ve Fethet (Divide and Conquer)-Hatırlatma



- Böl ve Fethet metoduyla algoritma tasarıımı:
- **Böl (Divide):** Eğer giriş boyut çok büyükse, problemi iki veya daha fazla birleşik alt probleme böl.
- **Fethet (Conquer):** Alt problemleri çözmek için böl ve fethet metodunu özyinelemeli olarak kullan
- **Birleştir (Combine):** Orjinal problemin çözümünü oluşturmak için alt problemlerin çözümlerini birleştir "merge".

# Böl ve Fethet (Divide and Conquer)-Hatırlatma

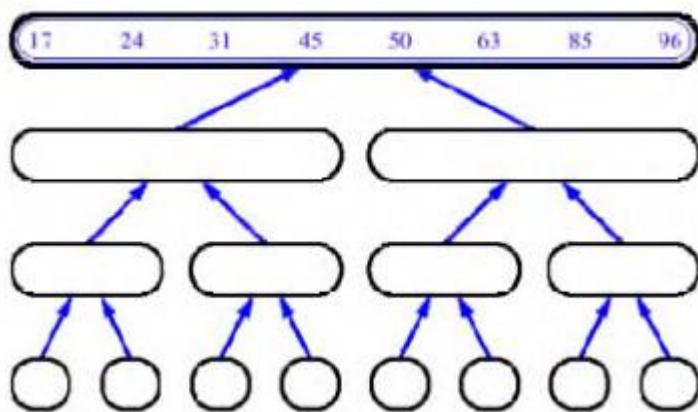


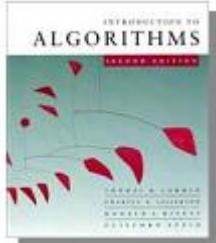
- Örnek, MergeSort
- Alt problemler bağımsızdır

```

Merge-Sort (A, p, r)
 if p < r then
 q←(p+r)/2
 Merge-Sort (A, p, q)
 Merge-Sort (A, q+1, r)
 Merge (A, p, q, r)

```





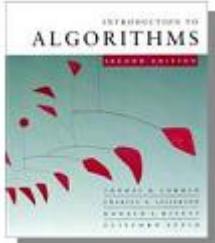
# Fibonacci Sayıları

- $F(n) = F(n-1) + F(n-2)$  seri
- $F(0) = 0, F(1) = 1$  başlangıç değerleri
- $0, 1, 1, 2, 3, 5, 8, 13, 21, 34 \dots$  seri açılım
- Bu serinin herhangi bir elemanını hesaplamak için iki farklı şekilde çözüm yapılabılır. Bunlardan biri özyineleme yöntemi ile ve diğerinin dinamik programlama mantığı ile yapılabilir.
- Recursive

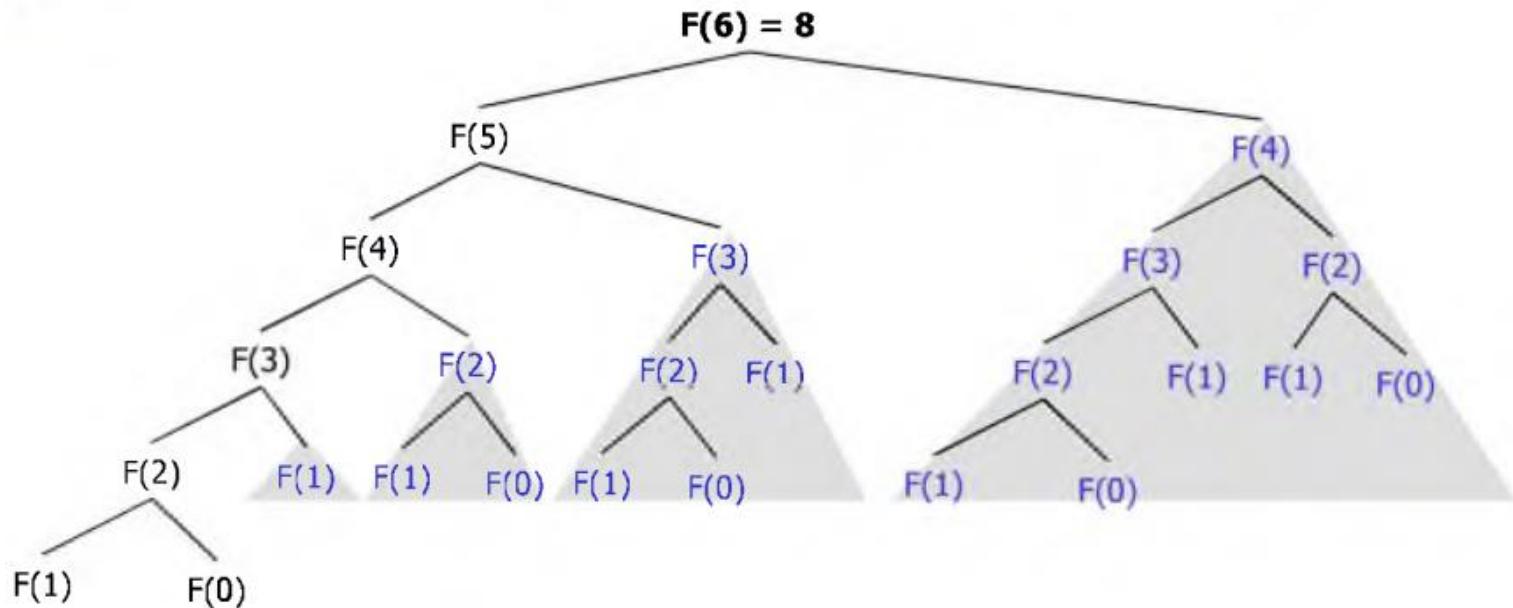
```
FibonacciR(n)
```

```
01 if n ≤ 1 then return n
02 else return FibonacciR(n-1) + FibonacciR(n-2)
```

- Recursive işlem çok yavaştır!  $T(n) \geq T(n-1) + T(n-2) + \Theta(1)$
- Niçin? Nasıl yavaş olur?

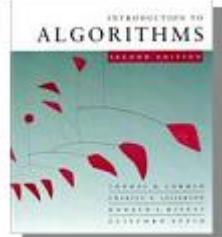


# Fibonacci Sayıları

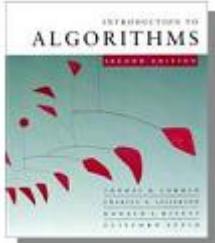


- Her seferinde  $F(n)$  sayısından küçük olan aynı değerler tekrar tekrar hesaplanmaktadır!

## Karakteristik denklemler kullanarak özyinelemeleri çözme



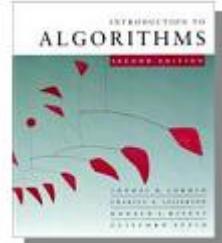
- ➊ Örnek: Fibonacci özyinelemeli bağıntı
- ➋  $t_0 = 0$  ve  $t_1 = 1$  ve  $n \geq 2$  olarak verildiğine göre
- ⌂  $t_n - t_{n-1} - t_{n-2} = 0$  için yinelemeli bağıntıyı çözünüz
- ⌃ Karakteristik denklem:  $x^2 - x - 1 = 0$
- ⌄ Kökler  $x_1 = \frac{1+\sqrt{5}}{2}$  ve  $x_2 = \frac{1-\sqrt{5}}{2}$ , kökler eşit değil. ( $\frac{1+\sqrt{5}}{2}$ , altın oran)
- ⌅ Durum 1'i kullanılacak.  $t_n = c_1\left(\frac{1+\sqrt{5}}{2}\right)^n + c_2\left(\frac{1-\sqrt{5}}{2}\right)^n$
- ⌆  $t_n(0) = 0 = c_1 + c_2$ ,  $t_1 = 1 = c_1 \cdot \frac{1+\sqrt{5}}{2} + c_2 \cdot \frac{1-\sqrt{5}}{2}$
- ⌇  $c_1 = 1/\sqrt{5}$  ve  $c_2 = -1/\sqrt{5}$  olarak bulunur.
- ⌈ Bu değerleri yerine yazarsak
- ⌋  $t_n = 1/\sqrt{5} \left[ \left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n \right]$  olarak bulunur ve sonuç olarak
- ⌌  $t_n \in \theta\left(\left((1 + \sqrt{5})/2\right)^n\right)$



# Fibonacci Sayıları

- $T(n) \approx \Theta(1.6^n)$  olur. Çalışma süresi **exponential** olur!
- Hesaplanan her terim, gerekli olduğu her yerde tekrar hesaplanmak yerine kullanıldığında bu yavaşlık ortadan kalkacaktır. Alt problemlerin çözümlerinin hafızada tutulmasıyla  $F(n)$  'i doğrusal çalışma süresiyle çözebiliriz – **dynamic programming**
- Çözüm bottom-up yaklaşımıyla çözülür
- Bu işlemi gerçekleştiren algoritma;

# Fibonacci Sayıları

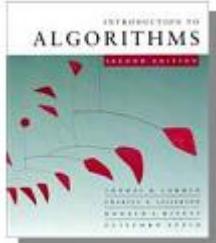


## Fibonacci( $n$ )

▷  $n$  hesaplanacak olan terimin numarasını gösterir.

1. Eğer  $n \leq 1$  ise
2. Sonuç  $\leftarrow 1$
3. değilse
4. Son  $\leftarrow 1$
5. Son2  $\leftarrow 1$
6. Cevap  $\leftarrow 1$
7.  $i \leftarrow 2, \dots, n$  kadar
8. Cevap  $\leftarrow$  Son + Son2
9. Son2  $\leftarrow$  Son
10. Son  $\leftarrow$  Cevap
11. Sonuç  $\leftarrow$  Cevap

- Gerçekte, herhangi bir iterasyonda sadece son iki değerin hafızada tutulması yeterlidir.
- Her adım için maliyet  $\Theta(1)$  ise en kötü durum için yani  $n$  tane elaman için  $T(n) = \Theta(n)$  olur. Çalışma süresi **doğrusal** olur!



# Binom Katsayıları

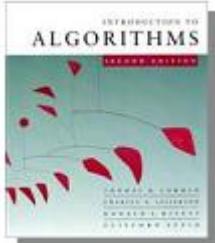
- Binom, polinomların özel hali, iki terimli polinom demek oluyor. Örnek verecek olursak.  $(x+y)$ ,  $(x-5)$ ,  $(3x^2+5y)$  gibi iki terimli ifadelere binom diyoruz. Binom açılımı ile,  $(x+y)^n$  ifadesinin tek tek terimlerin neler olduğunu gösteren formül kastediliyor.

Örneğin:

$$(x+y)^2 = x^2 + 2xy + y^2$$

$$(x+y)^3 = x^3 + 3x^2y + 3xy^2 + y^3$$

İfadelerinde hem her terimin katsayısı, hem de değişkenlerin üslerini hesaplayan formülün adı 'binom teoremi', 'binom formülü', terimlerin katsayıları da 'binom katsayıları' adlarıyla anılırlar. Binom açılımı birkaç değişik şekilde ifade edilebilir:



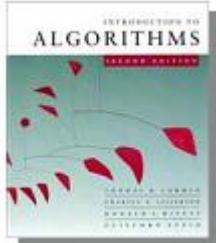
# Binom Katsayıları

- **Pascal Üçgeni:** Aşağıda görülen üçgene bakarak

|               |
|---------------|
| 1             |
| 1 2 1         |
| 1 3 3 1       |
| 1 4 6 4 1     |
| 1 5 10 10 5 1 |

.....

- $(x+y)^5$  ifadesini  $x^5+5x^4y+10x^3y^2+10x^2y^3+5xy^4+y^5$  olarak açabiliyoruz. Ancak,  $(x+y)^{18}$  gibi bir açılım yapmak istediğimizde, gerçekten büyük bir üçgen kurmak zorunda kalacağımızı da göz önünde tutalım.



# Binom Katsayıları

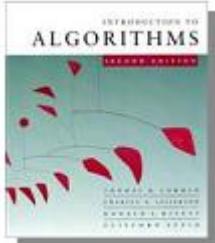
- **Binom Teoremi:**

$$(x+y)^n = x^n + nx^{n-1}y + n(n-1)x^{n-2}y^2/2! + n(n-1)(n-2)x^{n-3}y^3/3!$$

- $+ \dots + n(n-1)(n-2)\dots(n-k)x^{n-k}y^k/k! + \dots + nxy^{n-1} + y^n$

- $(x+y)^n = \sum_{k=0}^n [n!/k!(n-k)!]x^{n-k}y^k$  şeklinde kapalı olarak yazılabilir.

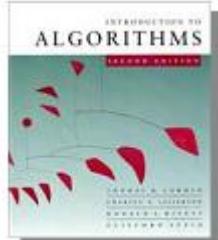
- ! İşareti faktöriyel işaretidir ve  $k!=1*2*3*\dots*(k-1)*k$  anlamını taşır.



# Binom Katsayıları

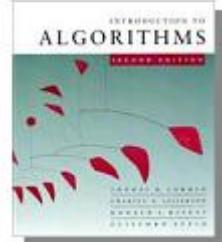
- Binom katsayıları hesaplama,  $C(k,n)$  
$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$
- $C(k,n)$ , toplam n nesneden mümkün k nesne seçenekleri sayıyor
- Özellikleri:
  - $C(k,n)=n!/k!(n-k)!$  ( $n!=1*2*3*...*(n-1)*n \rightarrow$  faktöriyel)
  - $C(k,n)=C(k,n-1)+C(k-1,n-1)$  (altyapı denklemi)
  - $C(0,n)=1; C(n,n)=1$
  - Büyük k ve n için, faktöriyel hesaplama zor olur, ve  $C(k,n)=C(k,n-1)+C(k-1,n-1)$  formül kullanılır.

$$(x + y)^n = \sum_{k=0}^n \binom{n}{k} x^k y^{n-k}$$



# Binom Katsayıları

- Direkt  $C(k,n) = C(k,n-1) + C(k-1,n-1)$  kullanma çok hesaplama gerekiyor, ama dinamik programlama için gereken bütün durumlar var:
  - Optimum altyapı var –  $C(k,n)$ , daha küçük  $k$  ve  $n$   $C$ -katsayıları kullanarak hesaplanır
  - (yani –  $C(k,n) = C(k,n-1) + C(k-1,n-1)$ )
- Örtüşen altproblemler özelliği var –  $C$  katsayılarının hepsi aynı şekilde hesaplanır (yani şu formül tekrar tekrar kullanılır,  $C(k,n) = C(k,n-1) + C(k-1,n-1)$ )



# Binom Katsayıları

$C(k,n)$  hesaplarken onları bir tabloda kaydedince, belirli  $C(k,n)$ ,  $O(kn)$  zamanla hesaplanabilir:

Sözde kodu:

$C(0,0)=1,$

**döngü**  $i=0$ 'dan  $n$ 'e **kadar** // döngü

**döngü**  $j=0$ 'den  $\min(i,k)$ 'ya **kadar** //döngü

**eğer**  $j=0$  veya  $j=i$  **ise**

$C(j,i)=1$

**aksi halde**

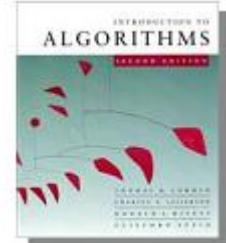
$C(j,i)=C(j-1,i-1)+C(j,i-1)$  //alt problemi hesaplama

**eğer** sonu //ve kaydetme

**döngü** sonu

**döngü** sonu

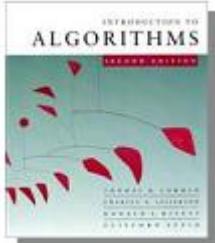
# Binom Katsayıları



- o `class pascal_ucgen`
- o `{ public void binom()`
- o  `{// değişkenleri tanımlama, maksimum değeri int için 19 olmalı, long tipinde daha`
- o  `// büyük rakamlar elde edilebilir`
- o  `Console.Clear();`
- o  `int[,] binom = new int[19, 19]; int n = 0;`
- o  `for (int i = 0; i < 19 - 1; i++)`
- o  `{ binom[i, 0] = 1; binom[i + 1, i + 1] = 1; }`
- o  `do { Console.WriteLine("Binom acilimini gormek istediginiz sayi n=");`
- o  `n = Convert.ToInt16(Console.ReadLine());`
- o  `} while (n < 0 || n >= 19);`
- o  `for (int k = 0; k < n; k++)`
- o  `{ for (int j = 0; j <= k + 1; j++)`
- o  `{ if (k + 1 == j || j == 0) binom[k, j] = 1;`
- o  `else binom[k + 1, j] = binom[k, j - 1] + binom[k, j];}`
- o  `Console.WriteLine();`
- o  `}`
- o  `for (int k = 0; k < n + 1; k++)`
- o  `{ for (int j = 0; j <= k; j++){ Console.WriteLine(binom[k, j] + "\t"); }`
- o  `Console.WriteLine();`
- o  `}`
- o `}`

Binom acilimini gormek istediginiz sayi n=6

|   |   |    |    |    |   |   |
|---|---|----|----|----|---|---|
| 1 |   |    |    |    |   |   |
| 1 | 1 |    |    |    |   |   |
| 1 | 2 | 1  |    |    |   |   |
| 1 | 3 | 3  | 1  |    |   |   |
| 1 | 4 | 6  | 4  | 1  |   |   |
| 1 | 5 | 10 | 10 | 5  | 1 |   |
| 1 | 6 | 15 | 20 | 15 | 6 | 1 |

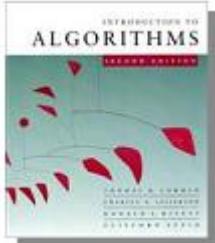


# Matrislerin Çarpımı

- **$A - nxm$**  ve  **$B - mxk$**  gibi iki matrisin çarpımından elde edilen  **$C-nxk$** , matrisi  **$n.m.k$**  adet çarpma işlemi gerektirir.

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} \begin{pmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{pmatrix} = \begin{pmatrix} \dots & \dots & \dots \\ \dots & c_{22} & \dots \\ \dots & \dots & \dots \end{pmatrix} \quad c_{i,j} = \sum_{l=1}^m a_{i,l} \cdot b_{l,j}$$

- **Problem:** Çok sayıdaki matrisin çarpım sonucunun en az çarpma yaparak elde edilmesi
- Matris, çarpma işleminin birleşme (associative) özelliğine sahiptir
  - $(AB)C = A(BC)$



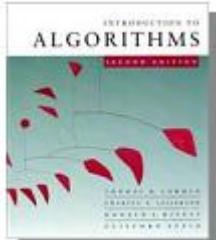
# Matrislerin Çarpımı

- $AxBxCxD$ , çarpımını düşünürsek
  - $A - 30x1, B - 1x40, C - 40x10, D - 10x25$

| <b>p0</b> | <b>p1</b> | <b>p2</b> | <b>p3</b> | <b>p4</b> |
|-----------|-----------|-----------|-----------|-----------|
| 30        | 1         | 40        | 10        | 25        |

- Çarpma sayısı:
  - $(AB)C)D = (30*1*40) + (30*40*10) + (30*10*25) = 20700$
  - $(AB)(CD) = (30*1*40) + (40*10*25) + (30*40*25) = 41200$
  - $A((BC)D) = 400 + 250 + 750 = 1400$
- Optimal parantezlemeye (işlem sırasına) ihtiyaç vardır.

$A_1 \times A_2 \times \dots \times A_n$ , burada  $A_i \ p_{i-1} \times p_i$  matristir



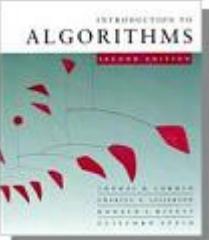
# Matrislerin Çarpımı

- $A_1 A_2 A_3 A_4$  matrislerin çarpımı 5 farklı şekilde elde edilebilir. ( $n=4$ )

$(A_1(A_2(A_3A_4)))$  ,  
 $(A_1((A_2A_3)A_4))$  ,  
 $((A_1A_2)(A_3A_4))$  ,  
 $((A_1(A_2A_3))A_4)$  ,  
 $(((A_1A_2)A_3)A_4)$  .

$$P(n) = \begin{cases} 1 & \text{if } n = 1 , \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n \geq 2 . \end{cases}$$

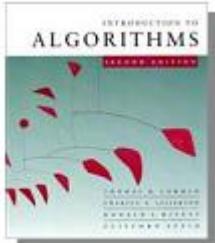
- Recurrence ifadenin çözümü,  $\Omega(2^n)$  olur,  $n \geq 7$  için
  - $n=2, 1$
  - $n=3, 2$
  - $n=4, 5$
  - $n=5, 14$
  - $n=6, 42$
  - $n=7, 132$



# Matrislerin Çarpımı

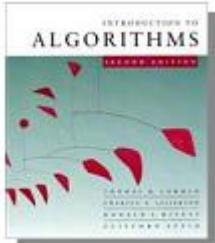
MATRIX-MULTIPLY( $A, B$ )

```
1 if $A.columns \neq B.rows$
2 error “incompatible dimensions”
3 else let C be a new $A.rows \times B.columns$ matrix
4 for $i = 1$ to $A.rows$
5 for $j = 1$ to $B.columns$
6 $c_{ij} = 0$
7 for $k = 1$ to $A.columns$
8 $c_{ij} = c_{ij} + a_{ik} \cdot b_{kj}$
9 return C
```



# Matrislerin Çarpımı

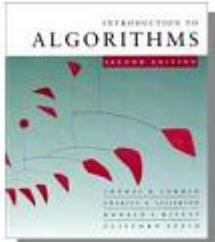
- Bu algoritma sadece optimal olarak kaç tane skaler çarpım yapılacağını hesaplar. Matrislerin çarpımını yapmaz.
- Dinamik programmanın dördüncü adımda olduğu gibi matris çarpımını yapan optimal çarpım algoritması hazırlanır.



# Matrislerin Çarpımı

- $m(i, j)$ ,  $\prod_{k=i}^j A_k$  çarpımlarını hesaplamak için gerekli olan **minimum** çarpma sayısını göstersin.
- **Çıkarımlar**
  - (i, j) matrislerinin belirlenen bir k ( $i \leq k < j$ ) değeri için en dıştaki parantezleri şu şekilde ifade edilebilir:  
 $(A_i \dots A_k)(A_{k+1} \dots A_j)$
  - (i, j) matrislerini optimal parantezlemek için k 'nın her iki tarafının da optimal parantezlenmesi gereklidir.
  - Minimum çarpma işlemi ifadesi

$$m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$$

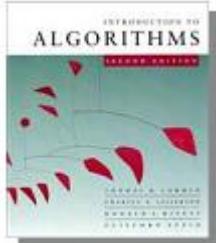


# Matrislerin Çarpımı

- Bütün olası  $k$  değerleri için denemeler yapılabilir.  
Özyinelemeli (Recurrence) ifade:

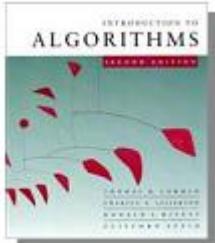
$$m[i, j] = \begin{cases} 0 & \text{if } i = j , \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\} & \text{if } i < j . \end{cases}$$

- Doğrudan recursive çalışma exponential çalışma süresine sahiptir - birçok işlem tekrar tekrar yapılır



# Matrislerin Çarpımı

- Optimal  $m(i, j)$  için  $O(n^2)$  kadar alana ihtiyaç duyulur ve  $m[1..n, 1..n]$  iki boyutlu dizisinin yarısı kullanılır.
  - $m(i,i) = 0, 1 \leq i \leq n$
  - Alt problemlerin çözümü büyüyen şekilde yapılır: önce 2 boyutunda alt problemler, sonra 3 boyutunda alt problemler ve 4, 5... şeklinde devam eder.
- Her  $(i, j)$  çifti için optimal parantezlemenin elde edilmesi için  $s[i, j] = k$  değeri kayıt edilir ve problem  $(i, k)$  ve  $(k+1, j)$  olarak iki alt probleme bölünür.



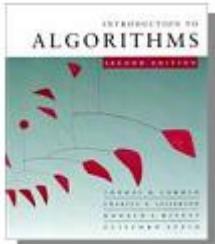
# Matrislerin Çarpımı

$$m[i, j] = \begin{cases} 0 & \text{if } i = j, \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j\} & \text{if } i < j. \end{cases}$$

MATRIX-CHAIN-ORDER( $p$ )

```

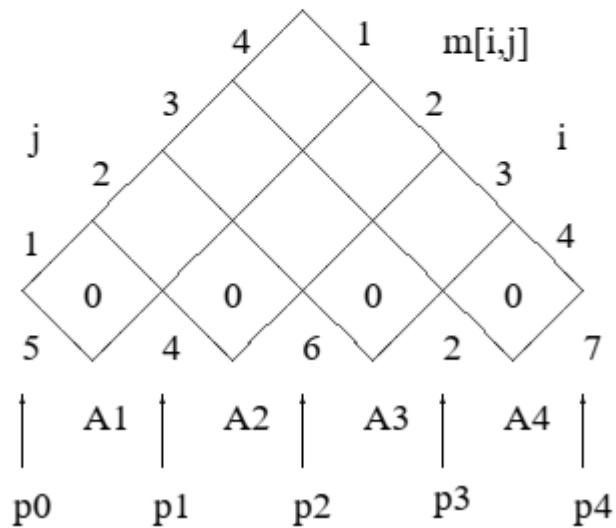
1 $n = p.length - 1$
2 let $m[1..n, 1..n]$ and $s[1..n - 1, 2..n]$ be new tables
3 for $i = 1$ to n
4 $m[i, i] = 0$ // i is the chain length
5 for $l = 2$ to n
6 for $i = 1$ to $n - l + 1$
7 $j = i + l - 1$
8 $m[i, j] = \infty$
9 for $k = i$ to $j - 1$
10 $q = m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j$
11 if $q < m[i, j]$
12 $m[i, j] = q$
13 $s[i, j] = k$
14 return m and s
```

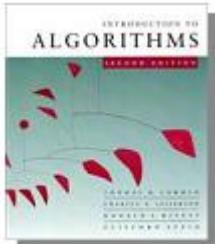


# Dynamic Programming: Minimum Weight Triangulation

**Example:** Given a chain of four matrices  $A_1, A_2, A_3$  and  $A_4$ , with  $p_0 = 5, p_1 = 4, p_2 = 6, p_3 = 2$  and  $p_4 = 7$ . Find  $m[1, 4]$ .

## S0: Initialization

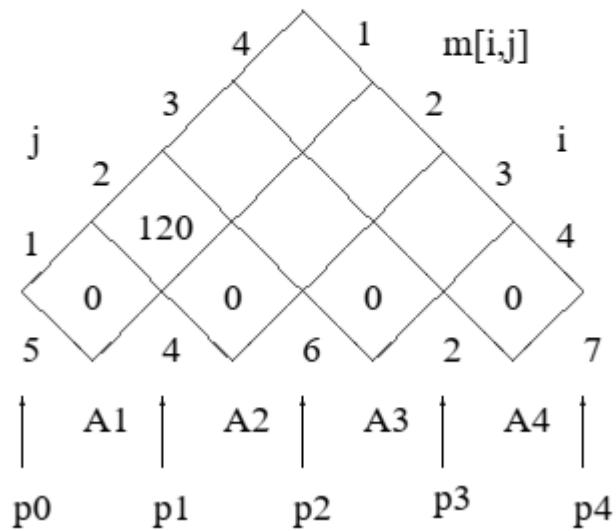


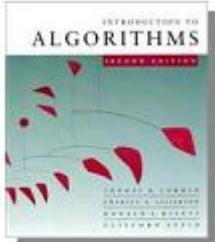


# Dynamic Programming: Minimum Weight Triangulation

**Step 1: Computing  $m[1, 2]$**  By definition

$$\begin{aligned} m[1, 2] &= \min_{1 \leq k < 2} (m[1, k] + m[k + 1, 2] + p_0 p_k p_2) \\ &= m[1, 1] + m[2, 2] + p_0 p_1 p_2 = 120. \end{aligned}$$

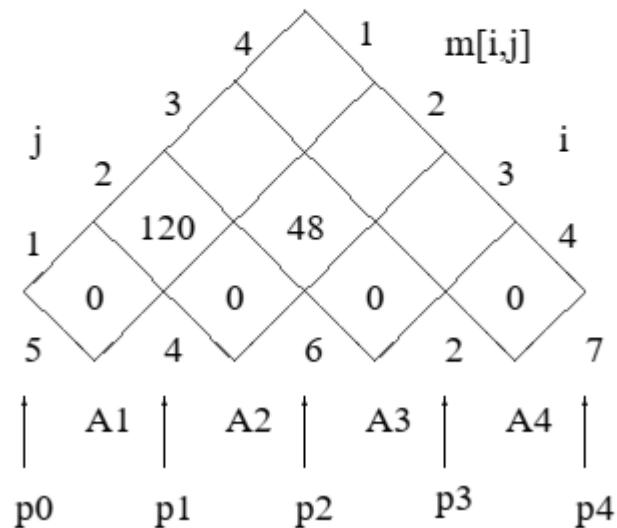


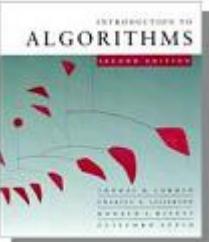


# Dynamic Programming: Minimum Weight Triangulation

**Step 2: Computing  $m[2, 3]$**  By definition

$$\begin{aligned} m[2, 3] &= \min_{2 \leq k < 3} (m[2, k] + m[k + 1, 3] + p_1 p_k p_3) \\ &= m[2, 2] + m[3, 3] + p_1 p_2 p_3 = 48. \end{aligned}$$

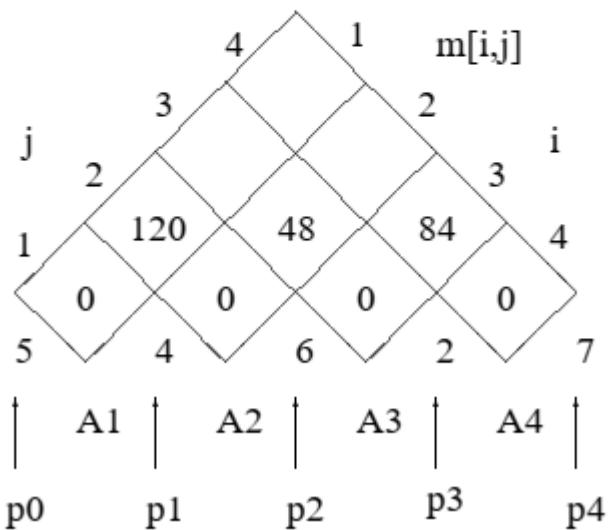


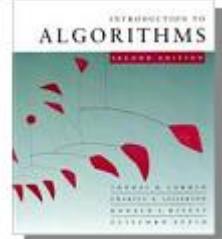


# Dynamic Programming: Minimum Weight Triangulation

**Step 3: Computing  $m[3, 4]$**  By definition

$$\begin{aligned} m[3, 4] &= \min_{3 \leq k < 4} (m[3, k] + m[k + 1, 4] + p_2 p_k p_4) \\ &= m[3, 3] + m[4, 4] + p_2 p_3 p_4 = 84. \end{aligned}$$

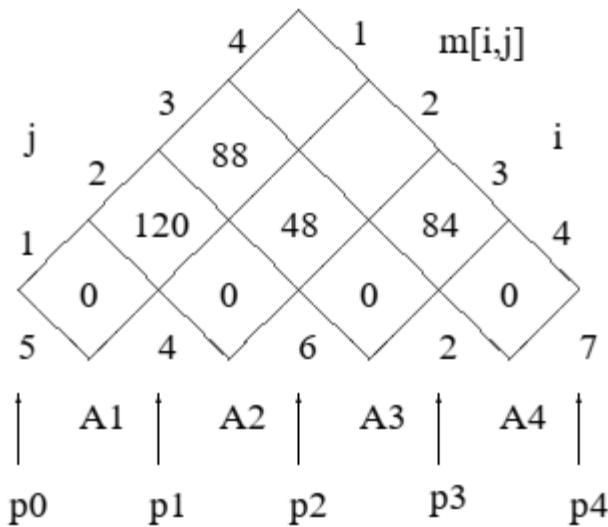


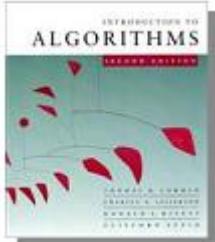


# Dynamic Programming: Minimum Weight Triangulation

**Step 4: Computing  $m[1, 3]$**  By definition

$$\begin{aligned}
 m[1,3] &= \min_{1 \leq k < 3} (m[1,k] + m[k+1,3] + p_0 p_k p_3) \\
 &= \min \left\{ \begin{array}{l} m[1,1] + m[2,3] + p_0 p_1 p_3 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 \end{array} \right\} \\
 &= 88.
 \end{aligned}$$

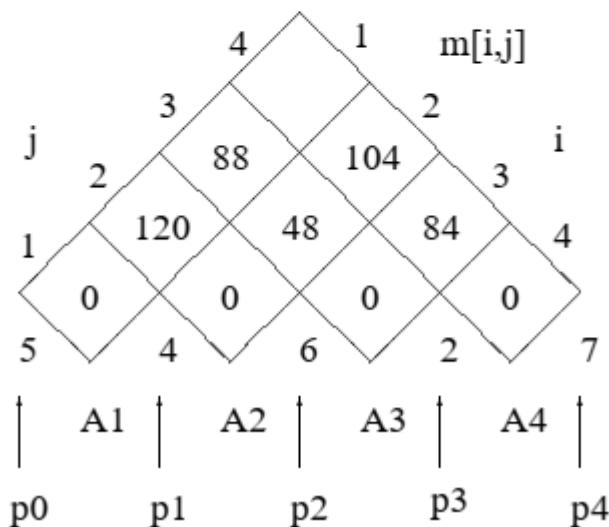


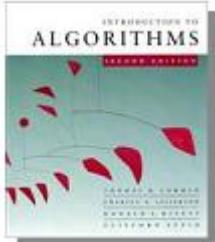


# Dynamic Programming: Minimum Weight Triangulation

**Step 5: Computing  $m[2, 4]$**  By definition

$$\begin{aligned}
 m[2, 4] &= \min_{2 \leq k < 4} (m[2, k] + m[k + 1, 4] + p_1 p_k p_4) \\
 &= \min \left\{ \begin{array}{l} m[2, 2] + m[3, 4] + p_1 p_2 p_4 \\ m[2, 3] + m[4, 4] + p_1 p_3 p_4 \end{array} \right\} \\
 &= 104.
 \end{aligned}$$

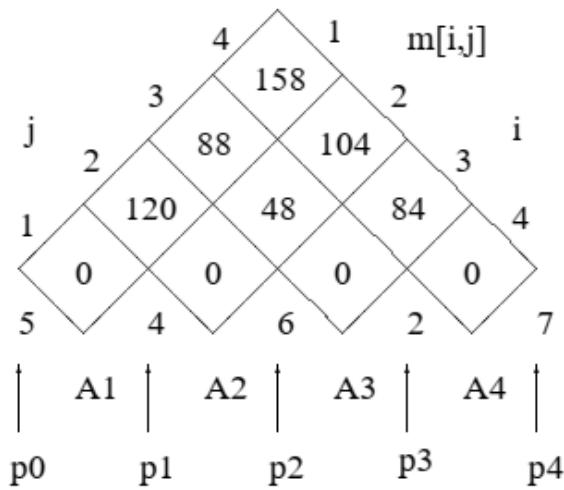


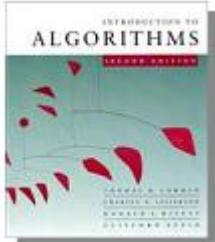


# Dynamic Programming: Minimum Weight Triangulation

**St6: Computing  $m[1, 4]$**  By definition

$$\begin{aligned}
 m[1, 4] &= \min_{1 \leq k < 4} (m[1, k] + m[k + 1, 4] + p_0 p_k p_4) \\
 &= \min \left\{ \begin{array}{l} m[1, 1] + m[2, 4] + p_0 p_1 p_4 \\ m[1, 2] + m[3, 4] + p_0 p_2 p_4 \\ m[1, 3] + m[4, 4] + p_0 p_3 p_4 \end{array} \right\} \\
 &= 158.
 \end{aligned}$$





# Dynamic Programming: Minimum Weight Triangulation

$A_1 (5 \times 4)$

$A_2 (4 \times 6)$

$A_3 (6 \times 2)$

$A_4 (2 \times 7)$

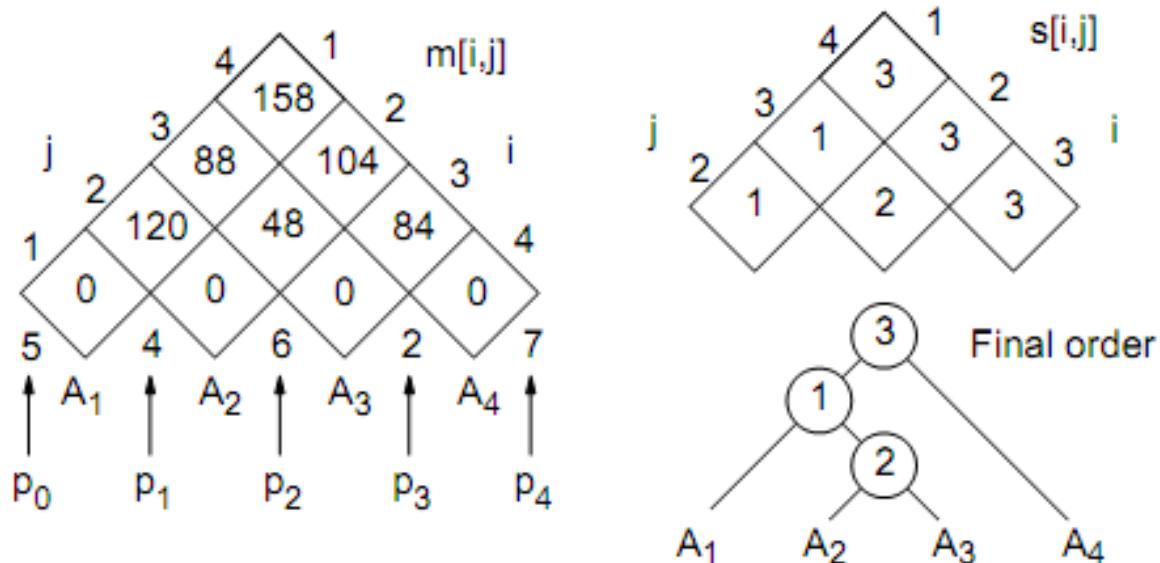
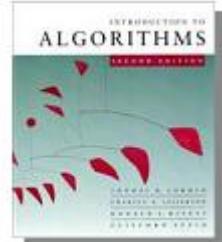


Fig. 9: Chain Matrix Multiplication Example.



# Matrislerin Çarpımı

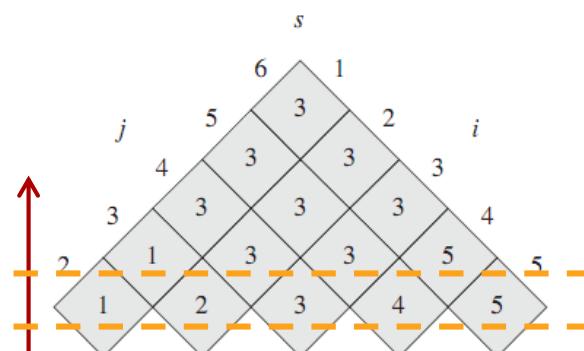
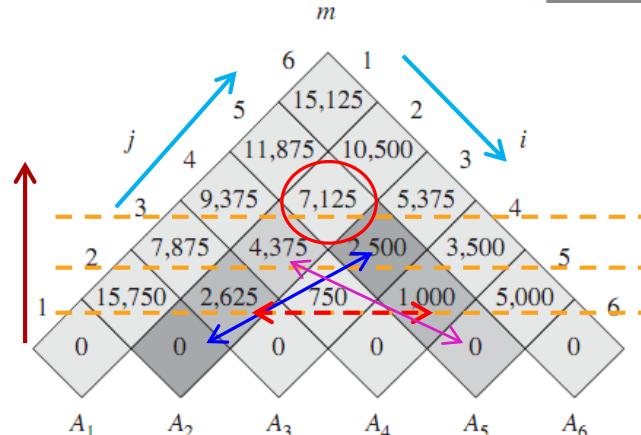
MATRIX-CHAIN-ORDER( $p$ )

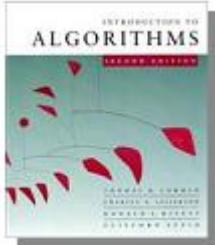
```

1 $n = p.length - 1$
2 let $m[1 \dots n, 1 \dots n]$ and $s[1 \dots n - 1, 2 \dots n]$ be new tables
3 for $i = 1$ to n
4 $m[i, i] = 0$
5 for $l = 2$ to n // l is the chain length
6 for $i = 1$ to $n - l + 1$
7 $j = i + l - 1$
8 $m[i, j] = \infty$
9 for $k = i$ to $j - 1$
10 $q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$
11 if $q < m[i, j]$
12 $m[i, j] = q$
13 $s[i, j] = k$
14 return m and s
```

| matrix dimension | $A_1$          | $A_2$          | $A_3$         | $A_4$         | $A_5$          | $A_6$          |
|------------------|----------------|----------------|---------------|---------------|----------------|----------------|
|                  | $30 \times 35$ | $35 \times 15$ | $15 \times 5$ | $5 \times 10$ | $10 \times 20$ | $20 \times 25$ |

$$m[2, 5] = \min \begin{cases} m[2, 2] + m[3, 5] + p_1 p_2 p_5 = 0 + 2500 + 35 \cdot 15 \cdot 20 = 13,000, \\ m[2, 3] + m[4, 5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \cdot 5 \cdot 20 = 7125, \\ m[2, 4] + m[5, 5] + p_1 p_4 p_5 = 4375 + 0 + 35 \cdot 10 \cdot 20 = 11,375 \\ \end{cases} = 7125.$$





# Matrislerin Çarpımı

- İç içe 3 tane for döngüsü vardır.
- Çalışma süresi  $O(n^3)$  olur.
- Exponential çalışma süresinden polynomial çalışma süresine indirildi.
- Çalışmanın sonucunda :  $m[1, n]$  optimal çözüm değerini ve  $s$  ise optimal alt parçaları ( $k$ ının seçimi) bulundurur.  $i=1, j=n$ ;

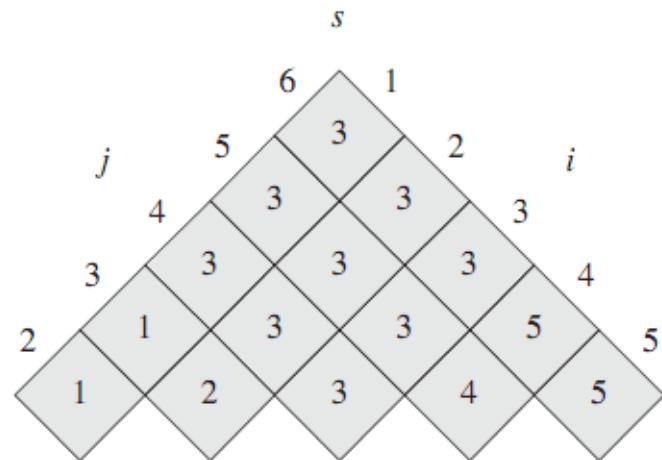
PRINT-OPTIMAL-PARENS( $s, i, j$ )

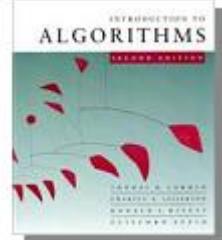
```

1 if $i == j$
2 print " A " $_i$
3 else print "("
4 PRINT-OPTIMAL-PARENS($s, i, s[i, j]$)
5 PRINT-OPTIMAL-PARENS($s, s[i, j] + 1, j$)
6 print ")"

```

$$((A_1(A_2A_3))((A_4A_5)A_6))$$





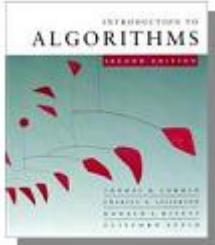
# Matrislerin Çarpımı

- Matris çarpımını gerçekleştiren algoritma

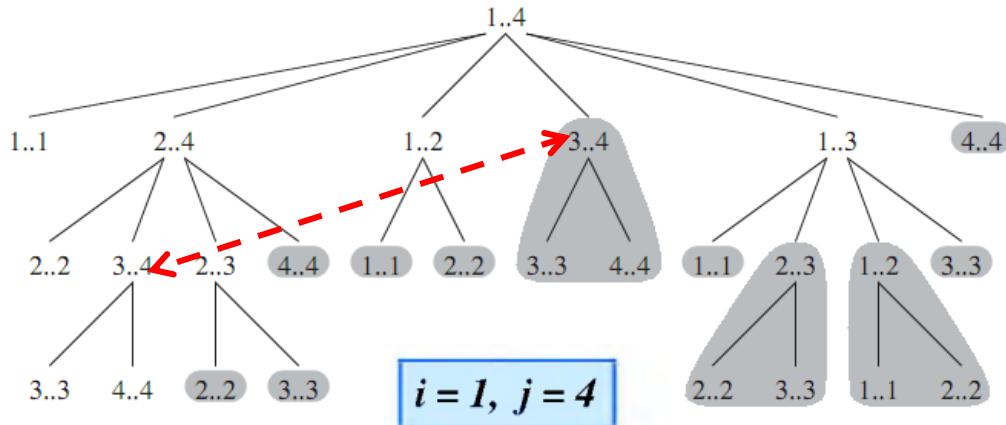
**Zincirleme\_matris\_çarpımı(A,s,i,j)**

▷  $A_1A_2\dots A_n$  matrislerinin optimal olarak çarpılmasını yapan bir algoritmadır. Bu işlemi gerçekleştirmek için m ve s tablolarını kullanmaktadır.

1. eğer  $j > i$  ise
2.     $X \leftarrow \text{Zincirleme_Matris_Çarpımı}(A, s, i, s[i, j])$
3.     $Y \leftarrow \text{Zincirleme_Matris_Çarpımı}(A, s, s[i, j] + 1, j)$
4.     $\text{Matris_Çarpımı}(X, Y)$



# Örtüşen alt problemler Overlapping Subproblems



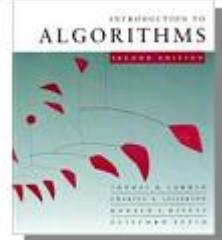
RECURSIVE-MATRIX-CHAIN( $p, i, j$ )

```

1 if $i == j$
2 return 0
3 $m[i, j] = \infty$
4 for $k = i$ to $j - 1$
5 $q = \text{RECURSIVE-MATRIX-CHAIN}(p, i, k)$
 + $\text{RECURSIVE-MATRIX-CHAIN}(p, k + 1, j)$
 + $p_{i-1} p_k p_j$
6 if $q < m[i, j]$
7 $m[i, j] = q$
8 return $m[i, j]$
```

Bir çok kez aynı değer çiftleri meydana gelir.

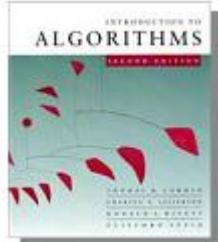
# Örtüşen alt problemler Overlapping Subproblems



- Gerçekte,  $m[1..n]$  çalışma süresi  $n$  üsteli şeklindedir.
- RECURSIVE-MATRIX-CHAIN tarafından,  $n$  matris zincirinin optimal çözüm değerini hesaplamak için geçen süreyi  $T(n)$  ile gösterebiliriz. O zaman, 1-2., 6-7. satırlar en az bir kez, 5.satır ise birden fazla işletilir ise;

$$T(1) \geq 1,$$

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \quad \text{for } n > 1.$$

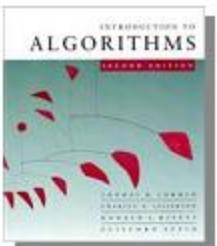


# Örtüşen alt problemler Overlapping Subproblems

- $i=1,2,\dots,n-1$  için  $T(i)$  bir kez,  $T(k)$  bir kez ve  $T(n-k)$  bir kez olarak ortaya çıkar.  $n-1$  tane bölme noktası vardır ve  $n-1$  tane 1 toplam içinde görünürken 1 tane 1 dışında görünmektedir. Buna göre denklem düzenlenliğinde

$$T(n) \geq 2 \sum_{k=1}^{n-1} T(k) + n$$

- olur. Bu durumda  $T(n)=\Omega(2^n)$  olduğu kabul edilsin. Buradan
- $T(n) \geq 2^{n-1}$  olsun ve  $T(1) \geq 2^0 = 1$  olur.  $T(n)$  ise
 
$$\begin{aligned} T(n) &\geq 2 \sum_{k=1}^{n-1} 2^{k-1} + n \\ &= 2 \sum_{k=0}^{n-2} 2^k + n \\ &= 2(2^{n-1} - 1) + n \\ &= 2^n - 2 + n \geq 2^{n-1} \end{aligned}$$
- şeklinde olur.



# Hatırlatma (Memoization) algoritması

Algoritma recursive olarak yapılandırılsa:

- Elemanları  $\infty$  olarak başlatılır ve **Lookup-Chain( $p$ ,  $i$ ,  $j$ )** metodu çağrırlar

MEMOIZED-MATRIX-CHAIN( $p$ )

```

1 $n = p.length - 1$
2 let $m[1..n, 1..n]$ be a new table
3 for $i = 1$ to n
4 for $j = i$ to n
5 $m[i, j] = \infty$
6 return LOOKUP-CHAIN($m, p, 1, n$)

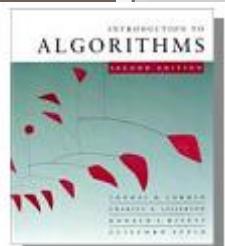
```

LOOKUP-CHAIN( $m, p, i, j$ )

```

1 if $m[i, j] < \infty$
2 return $m[i, j]$
3 if $i == j$
4 $m[i, j] = 0$
5 else for $k = i$ to $j - 1$
6 $q = \text{LOOKUP-CHAIN}(m, p, i, k)$
 + LOOKUP-CHAIN($m, p, k + 1, j$) + $p_{i-1} p_k p_j$
7 if $q < m[i, j]$
8 $m[i, j] = q$
9 return $m[i, j]$

```

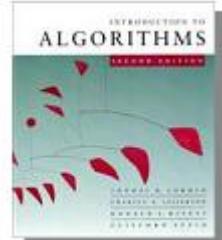


# Hatırlatma (Memoization) algoritması

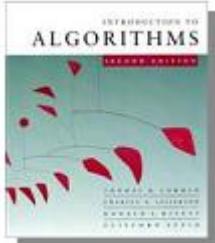
- **Hatırlatma:** Problemleri top-down yaklaşımıyla çözer, ancak alt problemlerin çözümleri bir tabloda tutulur.
- **Avantajları ve Dezavantajları :**
  - Recursion genellikle döngülerden daha yavaşır ve stack alanı kullanır
  - Anlaşılması kolaydır.
  - Eğer tüm alt problemler çözülmek zorunda değilse sadece gerekli olanlar çözülür.

# Dinamik Programlama

## En uzun ortak alt dizi (LCS-Longest Common Subsequence)



- $x$  ve  $y$  gibi iki text string verilirse:
- Birbirlerine benzerlikleri farklı şekillerde ifade edilebilir:
  - Birisi diğerinin substring' i ise benzerdirler
  - Birini diğerine dönüştürmek için gerekli olan değişiklik azaldıkça benzerlikleri artar
  - Her ikisinden oluşturulan bir  $z$  string'in her ikisinde de aynı sırada yer alması (yan yana olması gerekmez) benzerliklerini gösterir.
  - $z'$  nin boyutu arttıkça  $x$  ve  $y$ ' nin benzerliği artar.
- Benzerliklerin ölçümünde kullanılan yöntem **LCS** olarak adlandırılır.
  - Farklı canlılar için DNA sıralamasının benzerliklerinin araştırılmasında ve Yazım kontrolü (Spell checkers) yapılmasında kullanılır.



# Dinamik Programlama

*Böl-ve-fethet gibi bir tasarım tekniği.*

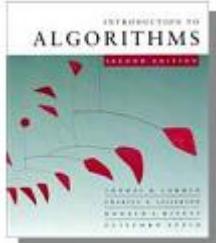
## Örnek: *En uzun ortak altdizi (LCS)*

- İki tane,  $x[1 \dots m]$  ve  $y[1 \dots n]$  dizisi verilmiş, ikisinde de ortak olan en uzun altdiziyi bulun.

En uzun altdizi tek degildir.

$x$ : A    B    C    B    D    A    B

$y$ : B    D    C    A    B    A



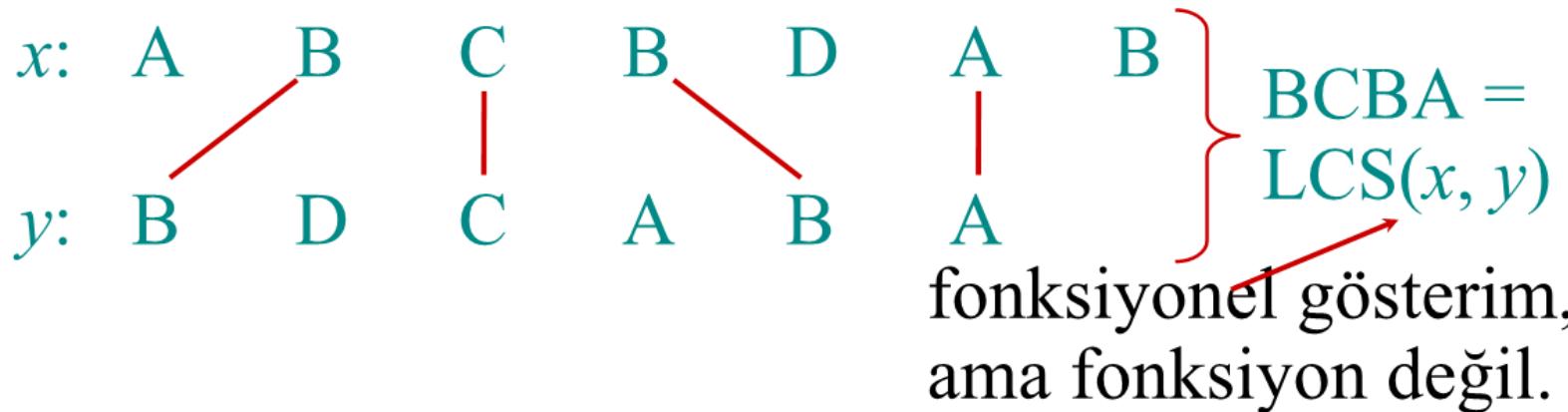
# Dinamik Programlama

*Böl-ve-fethet gibi bir tasarım tekniği.*

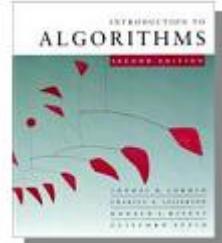
## Örnek: *En uzun ortak altdizi (LCS)*

- İki tane,  $x[1 \dots m]$  ve  $y[1 \dots n]$  dizisi verilmiş, ikisinde de ortak olan en uzun altdiziyi bulun.

En uzun altdizi tek değildir.

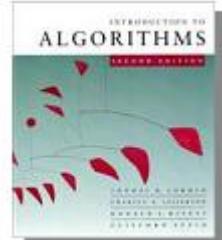


# Dinamik Programlama- En uzun ortak alt dizi (LCS)



- Eğer  $X'$  te bazı karakterleri atlayarak(veya hiç atlamayarak)  $Z$  oluşturulabilirse  $Z$ ,  $X'$  in alt dizisi olur.
  - Örnek:  $X = "ACGGTTA"$ ,  $Y = "CGTAT"$  ise
  - $LCS(X,Y) = "CGTA"$  veya  $"CGTT"$
- LCS problemini çözmek için  $LCS(X,Y)$  için  $X$  ve  $Y$ 'deki atlamaları oluşturmamız gerekmektedir.
- $X = \langle A, B, C, B, D, A, B \rangle$ ,  $Y = \langle B, D, C, A, B, A \rangle$
- $Z = \langle B, C, B, A \rangle$
- $LCS = \langle 2, 3, 4, 6 \rangle$

# Kaba-kuvvet LCS algoritması Brute-force LCS algorithm

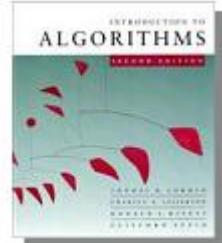


$x[1 \dots m]$ ' nin her altdizisini  
 $y[1 \dots n]$ ' nin de altdizisi mi diye kontrol edin.

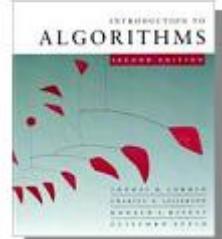
## Analiz

- Kontrol etme = her altdizi için  $O(n)$  zamanı.
- $x'$  in  $2^m$  sayıda altdizisi var. ( $m$  uzunluğunda her bir bit-vektörü,  $x$ 'in farklı bir altdizisini belirler.)  
En kötü durum koşma süresi =  $O(n2^m)$   
= üstel zaman.

# Dinamik Programlama- LCS: Optimal'in Bulunması



- $X = \langle x_1, x_2, \dots, x_m \rangle$  ve  $Y = \langle y_1, y_2, \dots, y_n \rangle$  string dizileri olmak üzere,  $Z = \langle z_1, z_2, \dots, z_k \rangle$   $X$  ve  $Y$  nin herhangi bir LCS olur.
- Eğer  $x_m = y_n$  ise, bu simbol  $z_k = x_m = y_n$  dizisine eklenir, ve  $Z_{k-1} = \text{LCS}(X_{m-1}, Y_{n-1})$  olur. ( $Z_{k-1}$ ,  $X_{m-1}$  ve  $Y_{n-1}$ 'in LCS' sidir)
- Eğer  $x_m \neq y_n$  ise,  $z_k \neq x_m$ ,  $Z = \text{LCS}(X_{m-1}, Y)$  olur
- Eğer  $x_m \neq y_n$  ise,  $z_k \neq y_n$ ,  $Z = \text{LCS}(X, Y_{n-1})$  olur bu durumda
  - $x$  veya  $y$  den bir karakter atlanır
  - $\text{LCS}(X, Y_{n-1})$  ve  $\text{LCS}(X_{m-1}, Y)$  'den büyük olan alınır.



# Daha iyi bir algoritmaya doğru

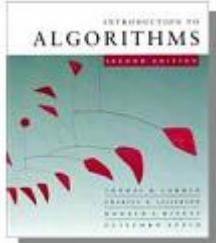
## Basitleştirme:

1. En uzun ortak bir altdizinin uzunluğuna bakalım.
2. Algoritmayı LCS' yi kendisi bulacak şekilde genişletin.

**Simgelem:**  $s$  dizisinin uzunluğunu  $|s|$  ile belirtin.

**Strateji:**  $x$  ve  $y$ 'nin öneklerini düşünün.

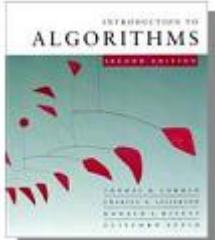
- Define(tanımlama)  $c[i, j] = |\text{LCS}(x[1 \dots i], y[1 \dots j])|$ .
- Then(sonra),  $c[m, n] = |\text{LCS}(x, y)|$ .



## LCS: Özyineleme

- $c[i,j] = \text{LCS}(x_i, y_j)$  olursa

$$c[i, j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0 \\ c[i - 1, j - 1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{c[i, j - 1], c[i - 1, j]\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

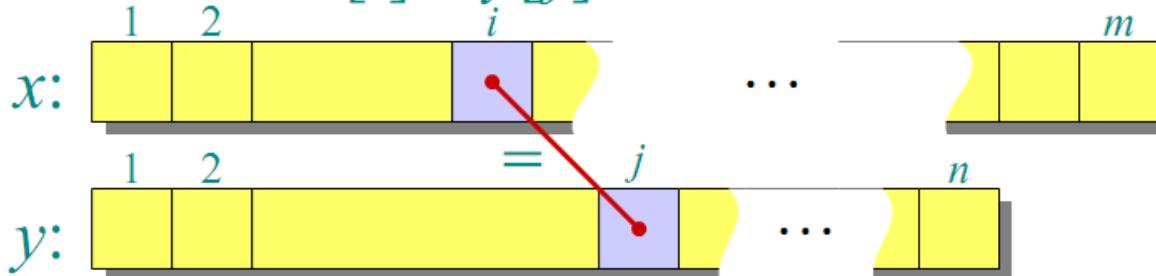


# Özyinelemeli formülleme

**Teorem.**

$$c[i, j] = \begin{cases} c[i-1, j-1] + 1 & \text{(eğer) if } x[i] = y[j] \text{ ise,} \\ \max\{c[i-1, j], c[i, j-1]\} & \text{aksi takdirde.} \end{cases}$$

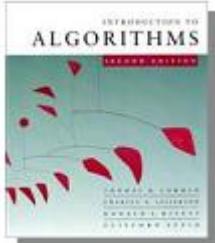
*Kanıt.* Durum  $x[i] = y[j]$ :



$c[i, j] = k$  iken  $z[1 \dots k] = \text{LCS}(x[1 \dots i], y[1 \dots j])$  olsun.

Sonra,  $z[k] = x[i]$ , ya da başka  $z$  genişletilebilir.

Böylece,  $z[1 \dots k-1]$ ,  $x[1 \dots i-1]$  ve  $y[1 \dots j-1]$ 'nin CS'sidir.



## Kanıt (devamı)

**İddia:**  $z[1 \dots k-1] = \text{LCS}(x[1 \dots i-1], y[1 \dots j-1])$ .

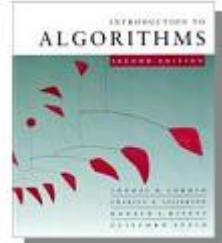
Diyelim ki  $w$ ,  $x[1 \dots i-1]$  ve  $y[1 \dots j-1]$ 'nin daha uzun bir CS'si, yani  $|w| > k-1$ . Sonra, **kes ve yapıştır**:  $w \parallel z[k]$  ( $w$ ,  $z[k]$  ile bitiştilmiş.)

$x[1 \dots i]$  ve  $y[1 \dots j]$ 'nin ortak altdizisiidir ve  $|w||z[k]| > k$ 'dır. Çelişki, iddiayı kanıtlar.

Böylece,  $c[i-1, j-1] = k-1$ 'dir ki bu  $c[i, j] = c[i-1, j-1] + 1$  anlamına gelir.

Diğer durumlar benzerdir.

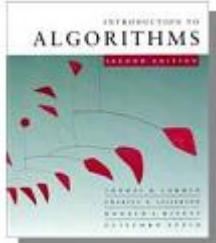
# Dinamik-programlama kalite işaretleri #1



## *Optimal altyapı*

*Bir probleme optimal çözüm (örnek) alt-problemlere optimal çözümler içerir.*

Eğer  $z = \text{LCS}(x, y)$  ise,  $z$  nin her öneki  $x'$  in ve  $y'$  nin öneklerinin LCS' sidir.



## LCS için özyinelemeli algoritma

$\text{LCS}(x, y, i, j)$

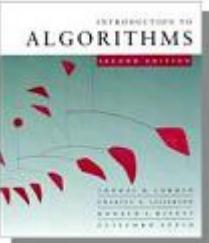
(eğer) **if**  $x[i] = y[j]$

(sonra) **then**  $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$

(başka) **else**  $c[i, j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j),$   
 $\text{LCS}(x, y, i, j-1) \}$

**return**  $c[i, j]$

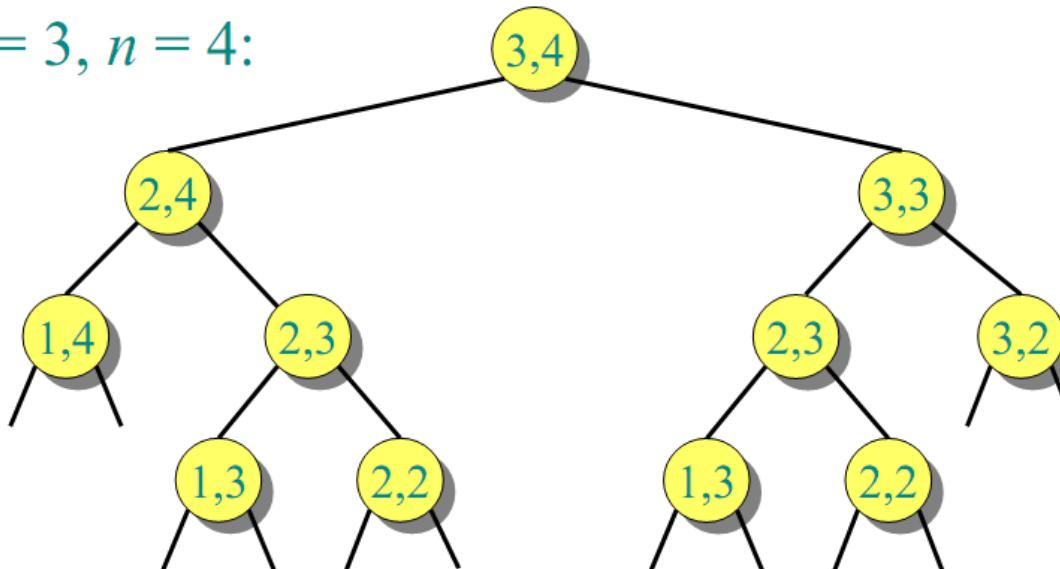
**En kötü durum:**  $x[i] \neq y[j]$ , bu durumda algoritma, tek parametrenin azaltıldığı iki alt-problemi değerlendirir.



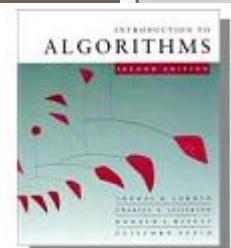
## Özyineleme ağacı

- Burada bu problem için ne olduğunu anlamamıza yarayacak öz yineleme ağacı çizilir. (m ve n örnek ağaç için en kötü durumda yürütmek için verilen değerleridir.)

$m = 3, n = 4$ :

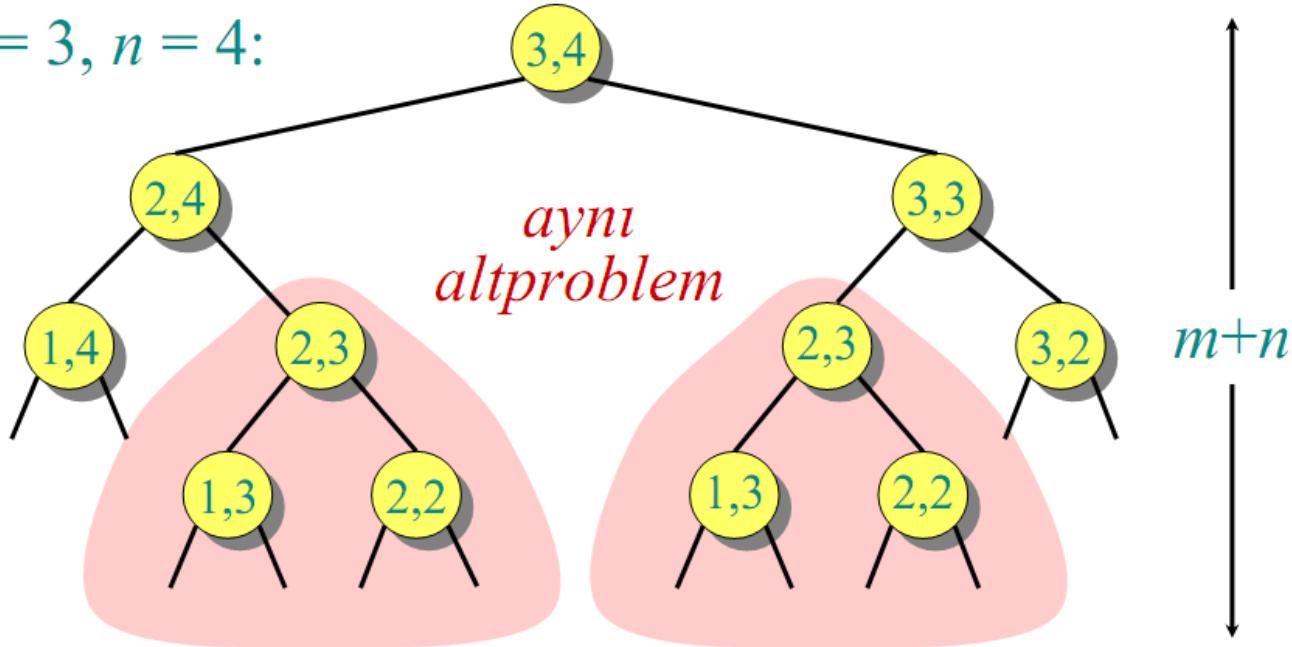


- Ağacın yüksekliği m ve n cinsinden ne olur?



# Özyineleme ağacı

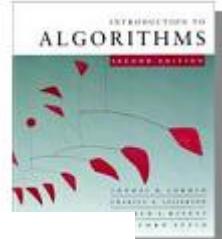
$m = 3, n = 4$ :



Yükseklik =  $m + n \Rightarrow$  iş potansiyel olarak üsteldir,  
ama biz zaten çözülmüş olan altproblemleri çözüyoruz!  $(2^{m+n})$

$2^{m+n}$  aynı zamanda toplam problem sayısıdır. Bir birinden farklı kaç alt problem var?

# Dinamik-programlama kalite işaretleri #2



## *Altproblemlerin çakışması*

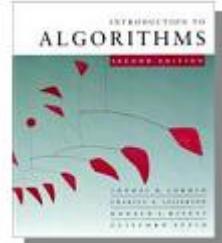
*Özyinelemeli bir çözüm, az sayıda  
birbirinden farklı altproblemin  
birçok kere tekrarlanması içindedir.*

$m$  ve  $n$  uzunluklarının 2 dizgisi için farklı LCS  
altproblemlerinin sayısı  $mn'$  dir.

- Bu  $2^m$  ve  $2^n$  den daha küçüktür. Çünkü  $i$  ve  $j$  ile  
karakterize ediliyor.  $i=<1..m>$ ,  $j=<1...n>$

# Hatırlama Algoritması

**Memoization(Hatırlama):** Bir altprobleme ait çözümü hesapladıkten sonra onu bir tabloda depolayın. Ardışık çağrımlar, işlemi tekrar yapmamak için tabloyu kontrol eder.



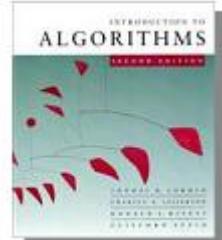
```

LCS(x, y, i, j)
(eğer) if $c[i, j] = \text{NIL}$
(sonra eğer) then if $x[i] = y[j]$
 (sonra) then $c[i, j] \leftarrow \text{LCS}(x, y, i-1, j-1) + 1$
 (başka) else $c[i, j] \leftarrow \max \{ \text{LCS}(x, y, i-1, j),$
 $\text{LCS}(x, y, i, j-1) \}$
return $c[i, j]$
 } önceki gibi

```

Eğer  $c[i, j] = \text{nil}$  ise hesaplıyoruz aksi takdirde hesaplamadan  $c[i, j]$  geri döndürüyoruz.

Süre =  $\Theta(mn)$  = her tablo girişi için sabit miktarda iş.  
 Yer(alan) =  $\Theta(mn)$ .



# Dinamik-programlama algoritması

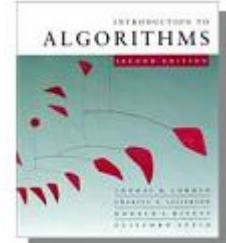
## Fikir:

Aşağıdan yukarıya tabloyu hesaplayın.

Süre =  $\Theta(mn)$ .

Başlangıçta sıfır  
uzunlığında bir  
dizi olduğundan  
ilk satır ve  
sütunlar 0 olur

|   | A | B | C | B | D | A | B |
|---|---|---|---|---|---|---|---|
| B | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| D | 0 | 0 | 1 | 1 | 1 | 2 | 2 |
| C | 0 | 0 | 1 | 2 | 2 | 2 | 2 |
| A | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| B | 0 | 1 | 2 | 2 | 3 | 3 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 3 | 4 |



# Dinamik-programlama algoritması

## Fikir:

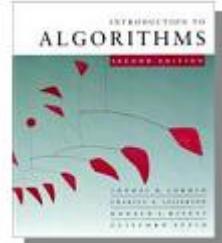
Aşağıdan yukarıya tabloyu hesaplayın.

Süre =  $\Theta(mn)$ .

LCS' yi, geriye giderek tekrar oluşturun.

Alan =  $\Theta(mn)$ .

|   | A | B | C | B | D | A | B |
|---|---|---|---|---|---|---|---|
| A | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| B | 0 | 0 | 1 | 1 | 1 | 1 | 1 |
| D | 0 | 0 | 1 | 1 | 1 | 2 | 2 |
| C | 0 | 0 | 1 | 2 | 2 | 2 | 2 |
| A | 0 | 1 | 1 | 2 | 2 | 3 | 3 |
| B | 0 | 1 | 2 | 2 | 3 | 3 | 4 |
| A | 0 | 1 | 2 | 2 | 3 | 3 | 4 |



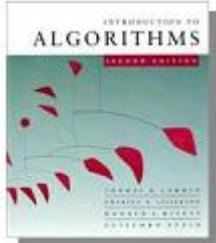
# LCS'nin Elde Edilmesi

LCS-LENGTH( $X, Y$ )

```

1 $m = X.length$
2 $n = Y.length$
3 let $b[1..m, 1..n]$ and $c[0..m, 0..n]$ be new tables
4 for $i = 1$ to m
5 $c[i, 0] = 0$
6 for $j = 0$ to n
7 $c[0, j] = 0$
8 for $i = 1$ to m
9 for $j = 1$ to n
10 if $x_i == y_j$
11 $c[i, j] = c[i - 1, j - 1] + 1$
12 $b[i, j] = "\nwarrow"$
13 elseif $c[i - 1, j] \geq c[i, j - 1]$
14 $c[i, j] = c[i - 1, j]$
15 $b[i, j] = "\uparrow"$
16 else $c[i, j] = c[i, j - 1]$
17 $b[i, j] = "\leftarrow"$
18 return c and b
```

| $i$ | $j$   | 0 | 1 | 2  | 3  | 4  | 5  | 6  |
|-----|-------|---|---|----|----|----|----|----|
|     | $y_j$ | B | D | C  | A  | B  | A  |    |
| 0   | $x_i$ | 0 | 0 | 0  | 0  | 0  | 0  | 0  |
| 1   | A     | 0 | 0 | 0  | 0  | 1  | -1 | 1  |
| 2   | B     | 0 | 1 | -1 | -1 | 1  | 2  | -2 |
| 3   | C     | 0 | 1 | 1  | 2  | -2 | 2  | 2  |
| 4   | B     | 0 | 1 | 1  | 2  | 2  | 3  | -3 |
| 5   | D     | 0 | 1 | 2  | 2  | 2  | 3  | 3  |
| 6   | A     | 0 | 1 | 2  | 2  | 3  | 3  | 4  |
| 7   | B     | 0 | 1 | 2  | 2  | 3  | 4  | 4  |



# LCS'nin Yazdırılması

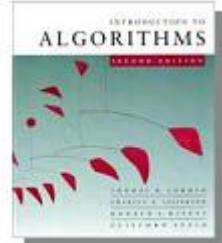
PRINT-LCS( $b, X, i, j$ )

```

1 if $i == 0$ or $j == 0$
2 return
3 if $b[i, j] == \nwarrow$
4 PRINT-LCS($b, X, i - 1, j - 1$)
5 print x_i
6 elseif $b[i, j] == \uparrow$
7 PRINT-LCS($b, X, i - 1, j$)
8 else PRINT-LCS($b, X, i, j - 1$)

```

|     | $j$   | 0 | 1 | 2  | 3  | 4  | 5  | 6  |
|-----|-------|---|---|----|----|----|----|----|
| $i$ | $y_j$ | B | D | C  | A  | B  | A  |    |
| 0   | $x_i$ | 0 | 0 | 0  | 0  | 0  | 0  | 0  |
| 1   | A     | 0 | 0 | 0  | 0  | 1  | -1 | 1  |
| 2   | B     | 0 | 1 | -1 | -1 | 1  | 2  | -2 |
| 3   | C     | 0 | 1 | 1  | 2  | -2 | 2  | 2  |
| 4   | B     | 0 | 1 | 1  | 2  | 2  | 3  | -3 |
| 5   | D     | 0 | 1 | 2  | 2  | 2  | 3  | 3  |
| 6   | A     | 0 | 1 | 2  | 2  | 3  | 3  | 4  |
| 7   | B     | 0 | 1 | 2  | 2  | 3  | 4  | 4  |



# Örnek

|     |                 |   |   |   |   |
|-----|-----------------|---|---|---|---|
|     | Y: 0 1 2 3 4 =n |   |   |   |   |
|     | B               | D | C | B |   |
| 0   | 0               | 0 | 0 | 0 | 0 |
| 1   | B               | 0 | 1 | 1 | 1 |
| X:  | A               | 0 | 1 | 1 | 1 |
| 3   | C               | 0 | 1 | 1 | 2 |
| 4   | D               | 0 | 1 | 2 | 2 |
| m=5 | B               | 0 | 1 | 2 | 2 |
|     | 3               |   |   |   |   |

X = BACDB

Y = BDCB

LCS = BCB

|     |                 |   |   |   |   |
|-----|-----------------|---|---|---|---|
|     | Y: 0 1 2 3 4 =n |   |   |   |   |
|     | B               | D | C | B |   |
| 0   | 0               | 0 | 0 | 0 | 0 |
| 1   | B               | 0 | 1 | 1 | 1 |
| X:  | A               | 0 | 1 | 1 | 1 |
| 3   | C               | 0 | 1 | 1 | 2 |
| 4   | D               | 0 | 1 | 2 | 2 |
| m=5 | B               | 0 | 1 | 2 | 2 |
|     | 3               |   |   |   |   |

start here

LCS Length Table

with back pointers included

Fig. 6: Longest common subsequence example for the sequences  $X = \langle BACDB \rangle$  and  $Y = \langle BCDB \rangle$ . The numeric table entries are the values of  $c[i, j]$  and the arrow entries are used in the extraction of the sequence.

## LCS Örnek

- $X = ABCB$
- $Y = BDCAB$
- $X$  ve  $Y$  nin LCS(En uzun ortak alt dizisi) nedir?

$$\text{LCS}(X, Y) = BCB$$

$X = A B C B$

$Y = B D C A B$

# LCS Örnek

|   |          |       |          |          |          |          |          |
|---|----------|-------|----------|----------|----------|----------|----------|
|   | j        | 0     | 1        | 2        | 3        | 4        | 5        |
| i |          | $Y_j$ | <b>B</b> | <b>D</b> | <b>C</b> | <b>A</b> | <b>B</b> |
| 0 | $X_i$    |       |          |          |          |          |          |
| 1 | <b>A</b> |       |          |          |          |          |          |
| 2 | <b>B</b> |       |          |          |          |          |          |
| 3 | <b>C</b> |       |          |          |          |          |          |
| 4 | <b>B</b> |       |          |          |          |          |          |

ABCB  
BDCAB

$$X = ABCB; \quad m = |X| = 4$$

$$Y = BDCAB; \quad n = |Y| = 5$$

Tahsis edilen dizi c[5,4]

# LCS Örnek

|   |          |          |          |          |          |          |          |       |
|---|----------|----------|----------|----------|----------|----------|----------|-------|
|   |          |          |          |          |          |          |          | ABCB  |
|   | j        | 0        | 1        | 2        | 3        | 4        | 5        | BDCAB |
| i |          | Yj       | <b>B</b> | <b>D</b> | <b>C</b> | <b>A</b> | <b>B</b> |       |
| 0 | Xi       | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> | <b>0</b> |       |
| 1 | <b>A</b> | <b>0</b> |          |          |          |          |          |       |
| 2 | <b>B</b> | <b>0</b> |          |          |          |          |          |       |
| 3 | <b>C</b> | <b>0</b> |          |          |          |          |          |       |
| 4 | <b>B</b> | <b>0</b> |          |          |          |          |          |       |

for  $i = 1$  to  $m$   
 for  $j = 1$  to  $n$

$c[i,0] = 0$   
 $c[0,j] = 0$

# LCS Örnek

ABCB  
BDCAB

| i | j  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----|---|---|---|---|---|---|
|   | Yj | B | D | C | A | B |   |
| 0 | Xi | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0 | 0 |   |   |   |   |
| 2 | B  | 0 |   |   |   |   |   |
| 3 | C  | 0 |   |   |   |   |   |
| 4 | B  | 0 |   |   |   |   |   |

```

if (Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

|   |    |    |   |   |   |   |   |
|---|----|----|---|---|---|---|---|
|   |    | 0  | 1 | 2 | 3 | 4 | 5 |
| i | j  | Yj | B | D | C | A | B |
| 0 | Xi | 0  | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0  | 0 | 0 | 0 |   |   |
| 2 | B  | 0  |   |   |   |   |   |
| 3 | C  | 0  |   |   |   |   |   |
| 4 | B  | 0  |   |   |   |   |   |

```

if($X_i == Y_j$)
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

```

# LCS Örnek

ABC<sub>B</sub>  
BDC<sub>A</sub>B

| i | j              | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----------------|---|---|---|---|---|---|
|   | Y <sub>j</sub> | B | D | C | A | B |   |
| 0 | X <sub>i</sub> | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A              | 0 | 0 | 0 | 0 | 1 |   |
| 2 | B              | 0 |   |   |   |   |   |
| 3 | C              | 0 |   |   |   |   |   |
| 4 | B              | 0 |   |   |   |   |   |

```

if (Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

ABCB  
BDCAB

| i | j  | 0 | 1 | 2 | 3 | 4 | 5   |
|---|----|---|---|---|---|---|-----|
|   | Yj | B | D | C | A | B |     |
| 0 | Xi | 0 | 0 | 0 | 0 | 0 | 0   |
| 1 | A  | 0 | 0 | 0 | 0 | 1 | → 1 |
| 2 | B  | 0 |   |   |   |   |     |
| 3 | C  | 0 |   |   |   |   |     |
| 4 | B  | 0 |   |   |   |   |     |

```

if(Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

ABC  
BDCAB

|   |          |          |   |   |   |          |   |
|---|----------|----------|---|---|---|----------|---|
|   | j        | 0        | 1 | 2 | 3 | 4        | 5 |
| i | Yj       | <b>B</b> | D | C | A | <b>B</b> |   |
| 0 | Xi       | 0        | 0 | 0 | 0 | 0        | 0 |
| 1 | A        | 0        | 0 | 0 | 0 | 1        | 1 |
| 2 | <b>B</b> | 0        | 1 |   |   |          |   |
| 3 | C        | 0        |   |   |   |          |   |
| 4 | B        | 0        |   |   |   |          |   |

```

if ($X_i == Y_j$)
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

```

# LCS Örnek

ABCB  
BDCAB

| i | j   | 0 | 1 | 2 | 3 | 4 | 5 |
|---|-----|---|---|---|---|---|---|
|   | Yj  | B | D | C | A | B |   |
| 0 | Xi  | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A   | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | (B) | 0 | 1 | 1 | 1 | 1 |   |
| 3 | C   | 0 |   |   |   |   |   |
| 4 | B   | 0 |   |   |   |   |   |

```

if (Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

|   |    |    |   |   |   |   |   |
|---|----|----|---|---|---|---|---|
|   | j  | 0  | 1 | 2 | 3 | 4 |   |
| i |    | Yj | B | D | C | A |   |
| 0 | Xi | 0  | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0  | 0 | 0 | 0 | 1 | 1 |
| 2 | B  | 0  | 1 | 1 | 1 | 1 | 2 |
| 3 | C  | 0  |   |   |   |   |   |
| 4 | B  | 0  |   |   |   |   |   |

ABCB  
BDCAB

5

B

2

```

if (Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

ABC  
BDCAB

| i | j              | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----------------|---|---|---|---|---|---|
|   | Y <sub>j</sub> | B | D | C | A | B |   |
| 0 | X <sub>i</sub> | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A              | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | B              | 0 | 1 | 1 | 1 | 1 | 2 |
| 3 | (C)            | 0 | 1 | 1 |   |   |   |
| 4 | B              | 0 |   |   |   |   |   |

```

if(Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

ABCB  
BDCAB

| i | j  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----|---|---|---|---|---|---|
|   | Yj | B | D | C | A | B |   |
| 0 | Xi | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | B  | 0 | 1 | 1 | 1 | 1 | 2 |
| 3 | C  | 0 | 1 | 1 | 2 |   |   |
| 4 | B  | 0 |   |   |   |   |   |

```

if(Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

ABC  
BDCAB

| i | j  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----|---|---|---|---|---|---|
|   | Yj | B | D | C | A | B |   |
| 0 | Xi | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | B  | 0 | 1 | 1 | 1 | 1 | 2 |
| 3 | C  | 0 | 1 | 1 | 2 | 2 | 2 |
| 4 | B  | 0 |   |   |   |   |   |

```

if($X_i == Y_j$)
 $c[i,j] = c[i-1,j-1] + 1$
else $c[i,j] = \max(c[i-1,j], c[i,j-1])$

```

# LCS Örnek

ABCB  
BDCAB

| i | j  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----|---|---|---|---|---|---|
|   | Yj | B | D | C | A | B |   |
| 0 | Xi | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | B  | 0 | 1 | 1 | 1 | 1 | 2 |
| 3 | C  | 0 | 1 | 1 | 2 | 2 | 2 |
| 4 | B  | 0 | 1 |   |   |   |   |

```

if (Xi == Yj)
 c[i,j] = c[i-1,j-1] + 1
else c[i,j] = max(c[i-1,j], c[i,j-1])

```

# LCS Örnek

ABCB  
BDCAB

| i | j  | 0 | 1 | 2 | 3 | 4 | 5 |
|---|----|---|---|---|---|---|---|
|   | Yj | B | D | C | A | B |   |
| 0 | Xi | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | A  | 0 | 0 | 0 | 0 | 1 | 1 |
| 2 | B  | 0 | 1 | 1 | 1 | 1 | 2 |
| 3 | C  | 0 | 1 | 1 | 2 | 2 | 2 |
| 4 | B  | 0 | 1 | 1 | 2 | 2 |   |

if ( $X_i == Y_j$ )  
 $c[i,j] = c[i-1,j-1] + 1$   
 else  $c[i,j] = \max( c[i-1,j], c[i,j-1] )$

# LCS Örnek

## LCS nin Bulunması

| j | 0  | 1 | 2   | 3   | 4   | 5   |   |
|---|----|---|-----|-----|-----|-----|---|
| i | Yj | B | D   | C   | A   | B   |   |
| 0 | Xi | 0 | 0   | 0   | 0   | 0   | 0 |
| 1 | A  | 0 | 0   | 0   | 0   | 1   | 1 |
| 2 | B  | 0 | 1 ← | 1 ↙ | 1   | 1   | 2 |
| 3 | C  | 0 | 1   | 1   | 2 ← | 2 ↗ | 2 |
| 4 | B  | 0 | 1   | 1   | 2   | 2   | 3 |

# LCS Örnek

## LCS nin Bulunması

| i  | j | 0     | 1 | 2     | 3 | 4 | 5 |
|----|---|-------|---|-------|---|---|---|
| Yj | B | D     | C | A     | B |   |   |
| Xi | 0 | 0     | 0 | 0     | 0 | 0 | 0 |
| A  | 0 | 0     | 0 | 0     | 1 | 1 |   |
| B  | 0 | 1 ← 1 | 1 | 1     | 1 | 2 |   |
| C  | 0 | 1     | 1 | 2 ← 2 | 2 | 2 |   |
| B  | 0 | 1     | 1 | 2     | 2 | 3 |   |

# **11.Hafta**

## **Veri sıkıştırma ve Açı gözlü algoritmalar**

# LCS C#

```
• using System;
• using System.Collections.Generic;
• using System.Linq;
• using System.Text;

• namespace LCS_ornek
• {
 class Program
 {
 int[,] c;
 string[,] b;
 public void Lcs(string[] x, string[] y)
 {
 int m = x.Length;
 int n = y.Length;
 c = new int[m + 1, n + 1];
 b = new string[m, n];
 for (int i = 1; i < c.GetLength(0); i++)
 c[i, 0] = 0;
 for (int j = 0; j < c.GetLength(1); j++)
 c[0, j] = 0;
 }

 for (int i = 0; i < m; i++)
 for (int j = 0; j < n; j++)
 if (x[i] == y[j])
 {
 c[i + 1, j + 1] = c[i, j] + 1;
 b[i, j] = "Çapraz";
 }
 else if (c[i, j + 1] >= c[i + 1, j])
 {
 c[i + 1, j + 1] = c[i, j + 1];
 b[i, j] = "Yukarı";
 }
 else
 {
 c[i + 1, j + 1] = c[i + 1, j];
 b[i, j] = "Sol";
 }
 }
}
```

# LCS C#

```

○ public void Print_Lcs(string[,] b, string[] x, int
 i, int j)
○ { if (i == 0 || j == 0)
○ { Console.WriteLine(x[i] + ", \t"); return; }
○ if (b[i, j] == "Çapraz")
○ {
○ Print_Lcs(b, x, i - 1, j - 1);
○ Console.WriteLine(x[i] + ", \t");
○ }
○ else if (b[i, j] == "Yukarı")
○ { Print_Lcs(b, x, i - 1, j); }
○ else
○ { Print_Lcs(b, x, i, j - 1); }
○ }
○
○ static void Main(string[] args)
{
○ Program lcs_xy = new Program();
○ string[] x = { "A", "B", "C", "B", "D", "A", "B" };
○ string[] y = { "B", "D", "C", "A", "B", "A" };
○ lcs_xy.Lcs(x,y);
○ for (int i = 0; i < x.Length + 1; i++)
○ { for (int j = 0; j < y.Length + 1; j++)
○ Console.WriteLine(lcs_xy.c[i, j] + "+" + "\t");
○ Console.WriteLine();
○ }
○
○ for (int i = 0; i < x.Length; i++)
○ {
○ for (int j = 0; j < y.Length; j++)
○ Console.Write(lcs_xy.b[i, j] + "," + "\t");
○ Console.WriteLine();
○ }
○ Console.WriteLine("\n");
○ lcs_xy.Print_Lcs(lcs_xy.b, x, x.Length-1,
y.Length-1);
○ }
○ }
○ }

```

# **11.Hafta**

## **Veri sıkıştırma ve Açı gözlü algoritmalar**

## Veri Sıkıştırma (Compression)

Kayıplı-Kayıpsız Veri Sıkıştırma  
Sabit ve Değişken Genişlikli  
Kodlama  
Huffman Algortiması (Greedy  
Algoithms)

# Veri Sıkıştırma

- Veri sıkıştırma;
- **1- Alan gereksinimini azaltmak için**
  - Hard disklerin boyutu büyümekle birlikte, yeni programların ve data dosyalarının boyutu da büyümektedir ve alan ihtiyacı artmaktadır.
- **2-Zaman ve bant genişliği gereksinimini azaltmak için**
  - Birçok program ve dosya internetten download edilmektedir.
  - Birçok kişi düşük hızlı bağlantıyu kullanmaktadır.
  - Sıkıştırılmış dosyalar transfer süresini kısaltmakta ve bir çok kişinin aynı server'ı kullanımına izin vermektedir.

# Kayıplı ve Kayıpsız Sıkıştırma

## ○ Kayıplı Sıkıştırma

- Bilginin bir kısmı geri elde edilemez (MP3, JPEG, MPEG)
- Genellikle ses ve görüntü uygulamalarında kullanılır. Çok büyük ses ve görüntü dosyalarında çok iyi sıkıştırma yapar ve kullanıcı kalitedeki azalmayı fark etmez.

## ○ Kayıpsız Sıkıştırma

- Orijinal dosyalar tekrar ve tam olarak elde edilir ( $D(C(X))=X$ , burada C sıkıştırılan dosyayı, D ise açılan dosyayı ifade eder.)
- .txt, .exe, .doc, .xls, .java, .cpp vs. gibi dosyalarda kullanılır.
- Ses ve görüntü dosyalarında da kullanılabilir ancak kayıplı sıkıştırma kadar yüksek sıkıştırma oranına sahip olmaz.

# Kayıpsız Sıkıştırma

- Kayıpsız sıkıştırma temel olarak üç algoritma altında toplanır
  - **Huffman** – her karakter için değişken genişlikli bir kelime kodu (codeword) kullanır .
  - **LZ277** – "sliding window (kayan pencere)" kullanır . Bir grup karakteri bir anda sıkıştırır.
  - **LZ278/LZW** – daha önce karşılaşılan işaretleri saklayan bir sözlük kullanır ve bir grup karakteri aynı anda sıkıştırır.
- Kayıpsız sıkıştırma uygulamaları
  - Unix compress, gif : LZW
  - pkzip, zip, winzip, gzip: Huffman+LZ277
  - ASCII, 8 bit
  - 29 farklı karakter 5 bit ile kodlanabilir.
  - $5/8 = \%62,5$  sıkıştırma oranı, kazanç  $\%37,5$  tır

## Huffman Kodlama

- Renksiz ve ozelliksiz karakterlerden oluşan dosyaların sıkıştırılması için kullanılan kodlardır. Dosyadaki bir karakterin tekrar sayısına o karakterin frekansı denir ve karakterlerin tekrar frekansından faydalananarak ikili kodlar üretilir.
- İki tip kodlama vardır: Sabit uzunluklu ve değişken uzunluklu.
- Toplam 1.000.000.000 karakterden oluşan bilginin sadece 6 harften {a, b, c, d, e, f} olduğunu varsayıyalım. Karakterlerin kullanım sıklıkları:

| a   | b   | c   | d   | e    | f    |                        |
|-----|-----|-----|-----|------|------|------------------------|
| .45 | .13 | .12 | .16 | .09  | .05  | <b>Frekans</b>         |
| 000 | 001 | 010 | 011 | 100  | 101  | <b>Fixed length</b>    |
| 0   | 101 | 100 | 111 | 1101 | 1100 | <b>Variable length</b> |

Fixed length      $3 \bullet 1.000.000.000 = 3.000.000.000 \text{ bits} \approx \mathbf{3GB}$

Variable length  $(.45 \bullet 1 + .13 \bullet 3 + .12 \bullet 3 + .16 \bullet 3 + .09 \bullet 3 + .05 \bullet 3) \bullet 1.000.000.000$   
 $= 2.240.000.000 \text{ bits} \approx \mathbf{2.24GB}$

## Huffman Kodlama

- Değişken uzunluklu kodlamada, frekansı en yüksek olan karaktere en kısa kod ve diğer karakterlere benzer mantıkla kod verilirse, sabit uzunluklu kodlamaya göre daha başarılı bir sonuç elde edilir.
- Kodlama ve kodlamayı çözmek için ön ek (prefix) kodlamadan faydalанılır.
- Kodlama her zaman için daha kolaydır. Kodu çözmek, sabit uzunluklu kodlamada oldukça kolaydır; değişken uzunluklu kodlamada ise, biraz daha zahmetlidir.
- Sıkıştırılmış bir dosyanın kod ağacı tam dolu ikili ağaç ise, bu sıkıştırma işlemi optimaldir; aksi halde optimal değildir.
- Sabit uzunluklu kodlama optimal değildir, çünkü ağacı tam dolu ikili ağaç değildir.

## Karakterlerin Kodunu Çözme

|   |      |
|---|------|
| E | 0    |
| T | 11   |
| N | 100  |
| I | 1010 |
| S | 1011 |

110|100|100|1010|1011  
T E N N I S

Prefix code

|   |      |
|---|------|
| E | 0    |
| T | 10   |
| N | 100  |
| I | 0111 |
| S | 1010 |

100|100|1010|10  
T E N N I S

Belirsiz

## Karakterlerin Kodunu Çözme

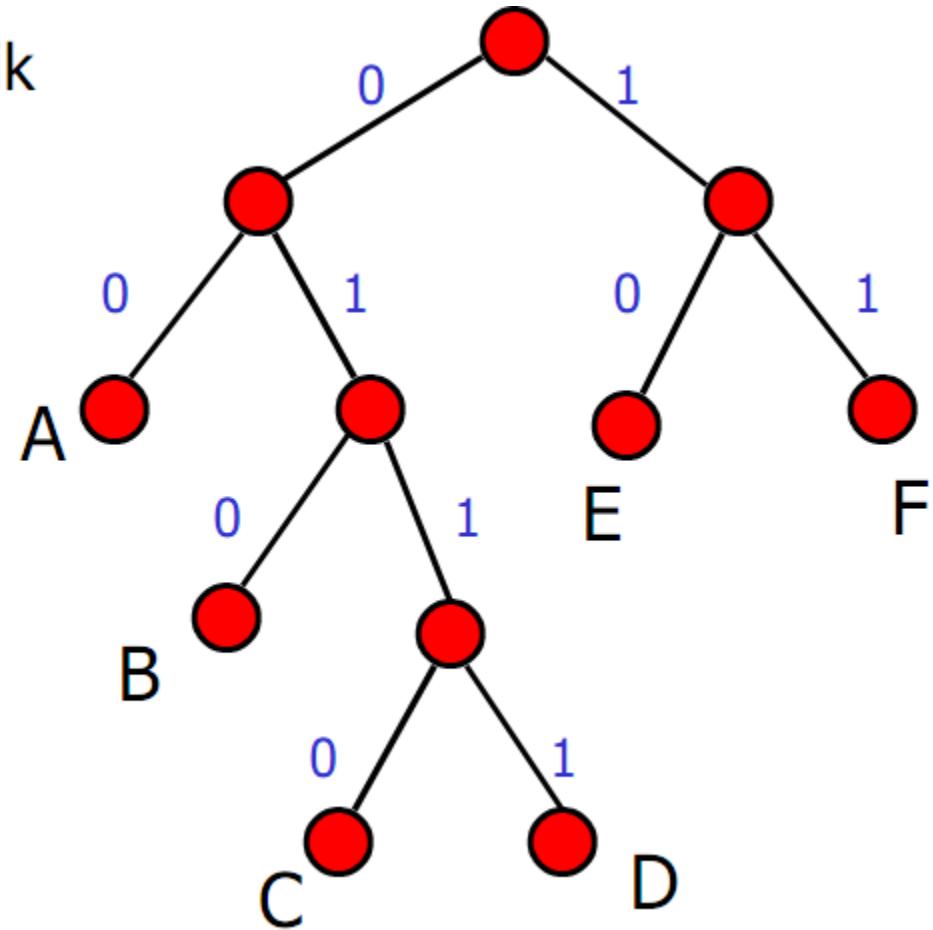
- Bir codeword başka bir codeword'ün prefix'i olamaz
- Kod çözülmesinde teklik olmalı

|   |     |        |       |
|---|-----|--------|-------|
| A | 00  | 1      | 00    |
| B | 010 | 01     | 10    |
| C | 011 | 001    | 11    |
| D | 100 | 0001   | 0001  |
| E | 11  | 00001  | 11000 |
| F | 101 | 000001 | 101   |

## Ön ek Kodları ve İkili ağaçta gösterimi

Prefix code'ların ikilik ağaçla gösterimi

|   |      |
|---|------|
| A | 00   |
| B | 010  |
| C | 0110 |
| D | 0111 |
| E | 10   |
| F | 11   |



## Kodlama için gereken boyut

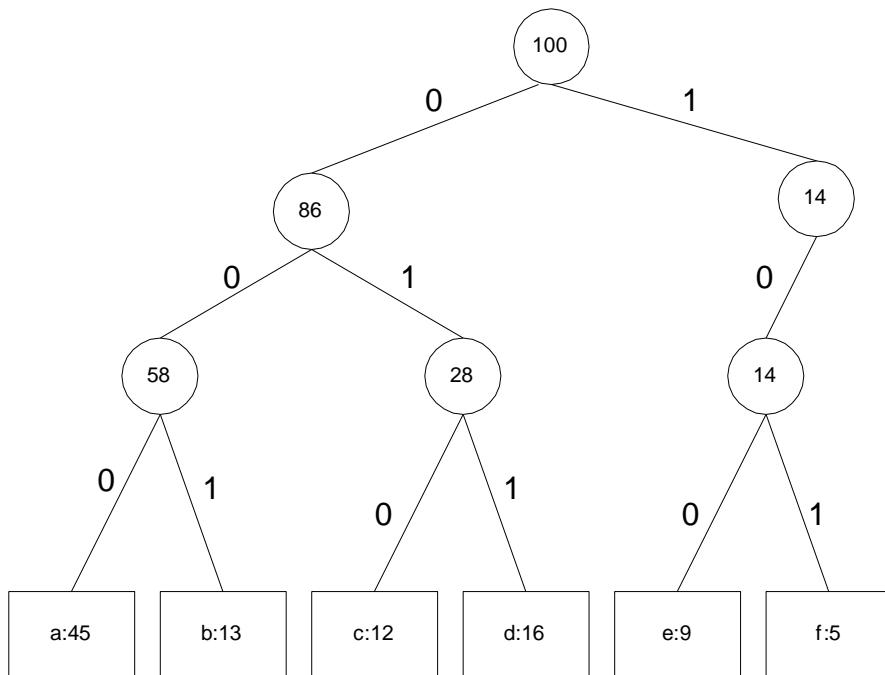
- Eğer ağaç **C** alfabetesinde üretilmiş ise, bu durumda yaprak sayısının  $|C|$  olması gereklidir ve ara düğümlerde  $|C|-1$  olmalıdır.
- $c \in C$  olsun ve  $f(c)$  bu karakterin frekansını göstersin. **T** ağaçında ön ek(prefix) kod ağaçları olsun.  $d_T(c)$  ise  $c$  karakterinin **T** ağacındaki derinliği olsun ve aynı zamanda  $c$  karakterinin kod uzunluğunu gösterir. Dosyanın kodlandıkten sonraki bit olarak uzunluğu  $B(T)$  aşağıdaki gibi ifade edilir

$$B(T) = \sum_{c \in C} f(c)d_T(c)$$

- olur.

## Kodlama ağacının oluşturulması

- Sabit uzunlukta kodlama



## Kodlama ağacının oluşturulması

- Değişken uzunlukta kodlama: Frekansa göre sıralanır ve her biri bir yaprağı temsil eder. En küçük iki değer bir atada toplanır ve geriye kalan yapraklar ile ata tekrar sıralanır.

f:5

e:9

c:12

b:13

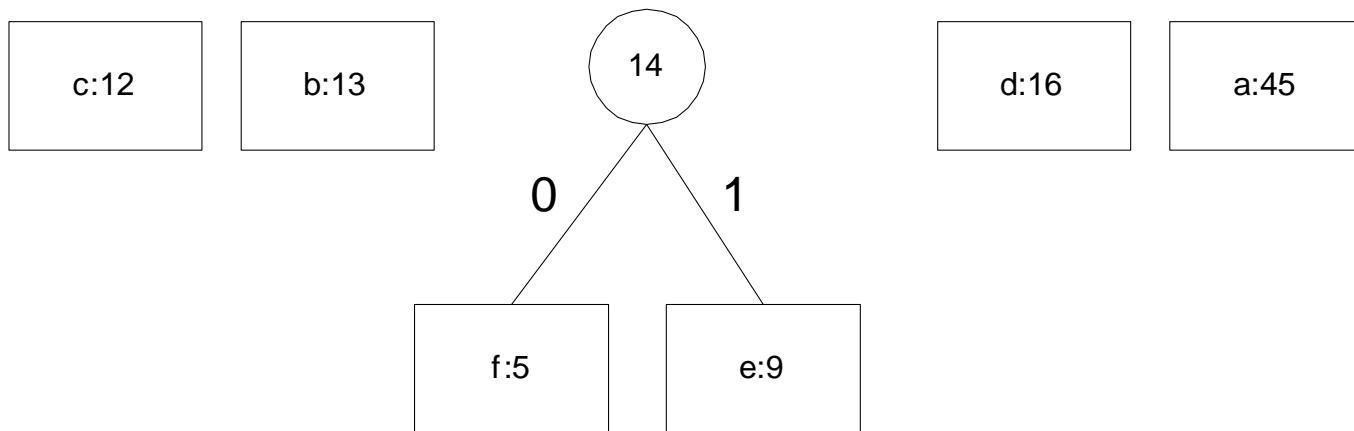
d:16

a:45

- Değişken uzunluklu kodlama için frekansların sıralanması.*

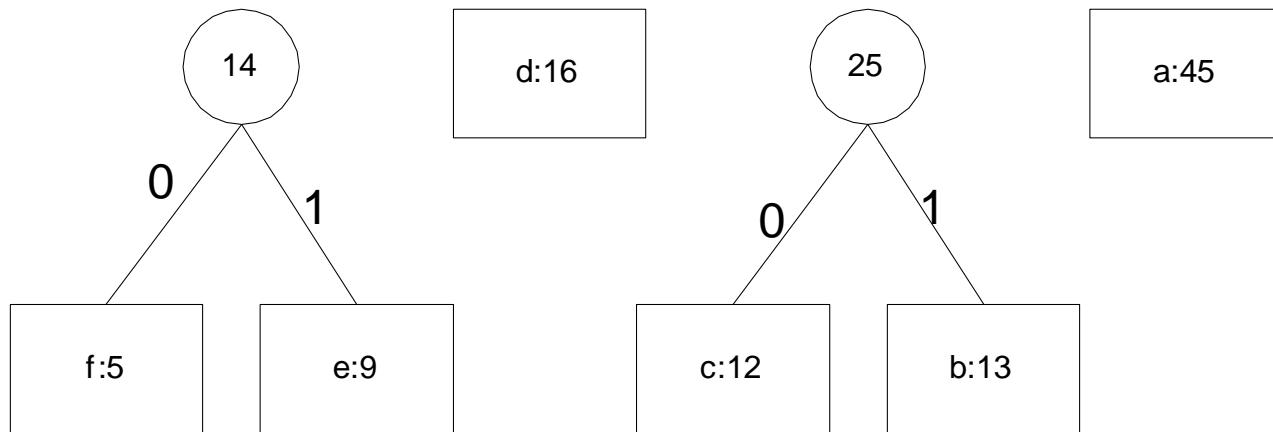
## Kodlama ağacının oluşturulması

- *Değişken uzunluklu kodlama*



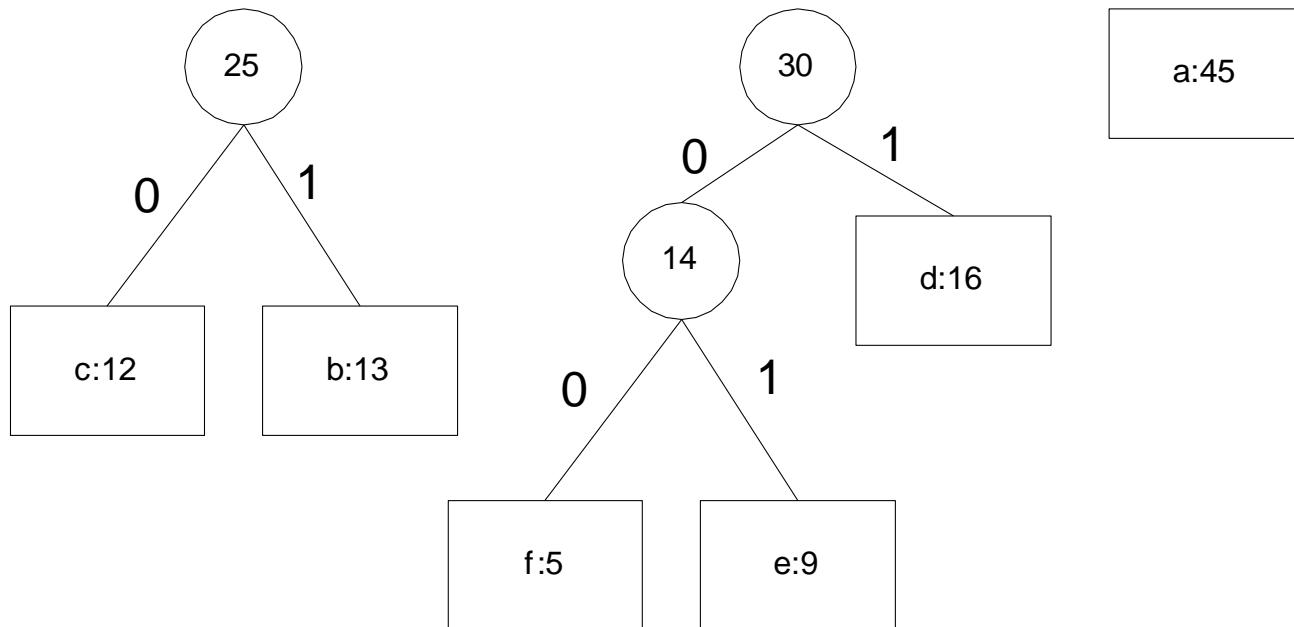
## Kodlama ağacının oluşturulması

- Değişken uzunluklu kodlama



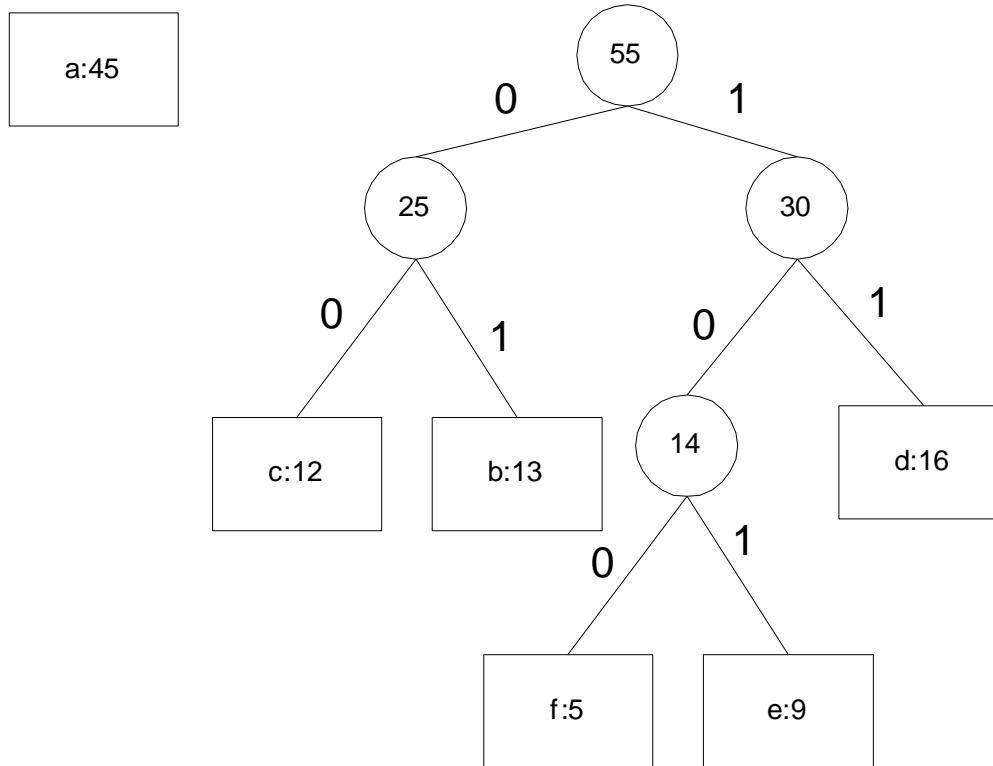
## Kodlama ağacının oluşturulması

- Değişken uzunluklu kodlama



## Kodlama ağacının oluşturulması

- Değişken uzunluklu kodlama



## Huffman Algoritması

$\text{HUFFMAN}(C)$

```
1 $n = |C|$
2 $Q = C$
3 for $i = 1$ to $n - 1$
4 allocate a new node z
5 $z.\text{left} = x = \text{EXTRACT-MIN}(Q)$
6 $z.\text{right} = y = \text{EXTRACT-MIN}(Q)$
7 $z.\text{freq} = x.\text{freq} + y.\text{freq}$
8 $\text{INSERT}(Q, z)$
9 return $\text{EXTRACT-MIN}(Q)$ // return the root of the tree
```

Farklı karakter sayısı

Min-Priority Queue

En düşük frekanslı iki düğüm

İki düğümün toplamı yeni düğüm

Yeni düğümü kuyruktaki yerine ekleme

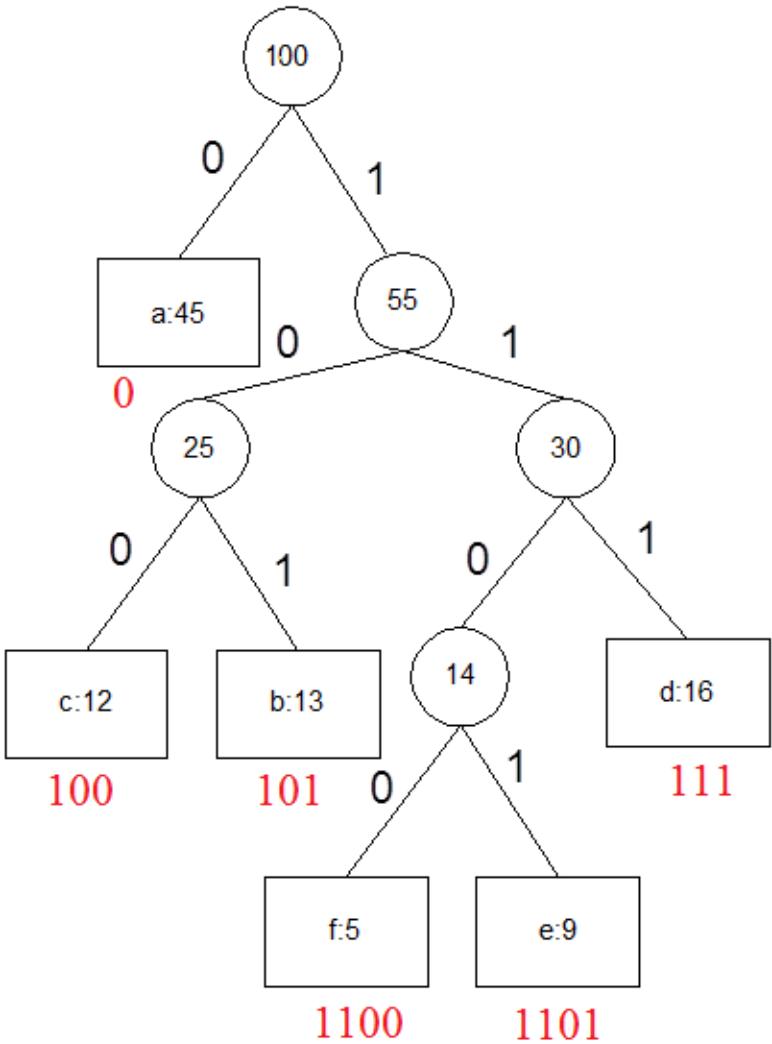
# Kodlama ağacının oluşturulması ve Huffman Algoritması

## ● Değişken uzunluklu kodlama

### Algoritma -Huffman(C)

► C alfabe olmak üzere bu algoritma Huffman kodu üretir.

1.  $n \leftarrow |C|$
2.  $Q \leftarrow C$
3.  $i \leftarrow 1, \dots, n-1$
4.  $z \leftarrow \text{Dügüm_Oluştur}()$
5.  $x \leftarrow \text{Sol}(z) \leftarrow \text{Minimum\_AI}(Q)$
6.  $y \leftarrow \text{Sağ}(z) \leftarrow \text{Minimum\_AI}(Q)$
7.  $f(z) \leftarrow f(x) + f(y)$
8.  $\text{Ekle}(Q, Z)$
9.  $\text{Sonuç} \leftarrow \text{Minimum\_AI}(Q)$



## Huffman Algoritması Analizi

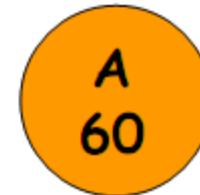
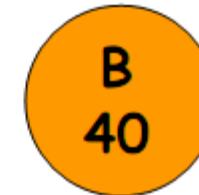
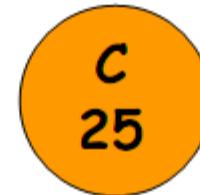
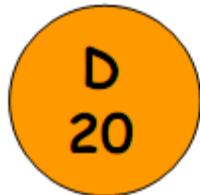
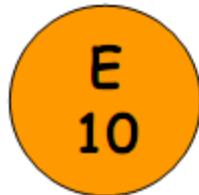
- Uygun veri yapısı olarak min-heap kullanılabilir. (Her seferinde frekansı düşük olan düğüm çıkarılır ve toplanarak ağaca tekrar eklenir)
- ([http://rosettacode.org/wiki/Huffman\\_coding](http://rosettacode.org/wiki/Huffman_coding))
- Heap oluşturma  $O(\log n)$  ve for döngüsünde  $n-1$  adet işlem yapılır
- Toplam Çalışma süresi  **$O(n \log n)$**  olur.
- Huffman algoritması az sayıda karakter çeşidine sahip ve büyük boyutlardaki verilerde çok kullanışlı olabilir. Fakat oluşturulan ağacın sıkıştırılmış veriye eklenmesi zorunludur. Bu da sıkıştırma verimini düşürür. Adaptive Huffman gibi teknikler bu sorunu halletmek için geliştirilmiştir.

## Örnek:

- Veri setine ait frekans tablosu aşağıdaki gibi olsun.

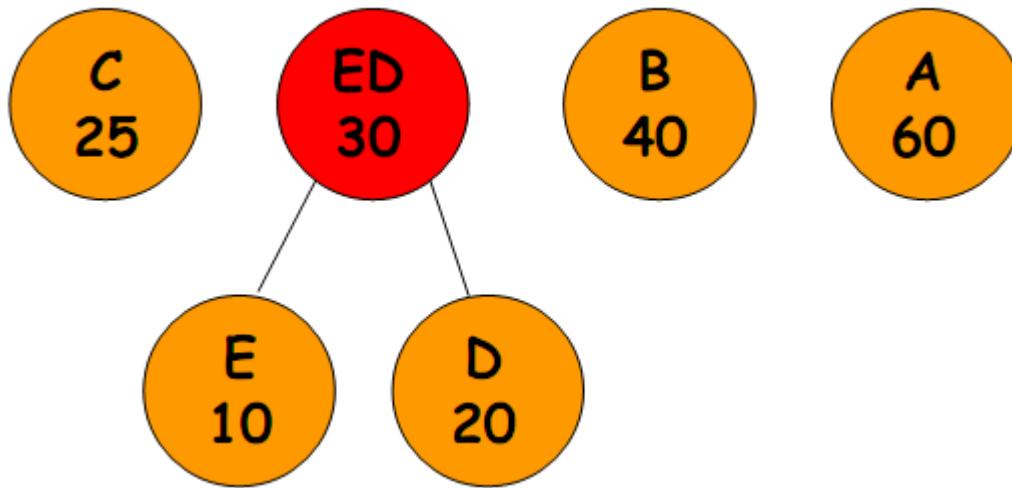
| Sembol | Frekans |
|--------|---------|
| A      | 60      |
| B      | 40      |
| C      | 25      |
| D      | 20      |
| E      | 10      |

- Adım 1:** Huffman ağacındaki en son düğümleri oluşturacak olan bütün semboller frekanslarına göre küçükten büyüğe doğru sıralanırlar.



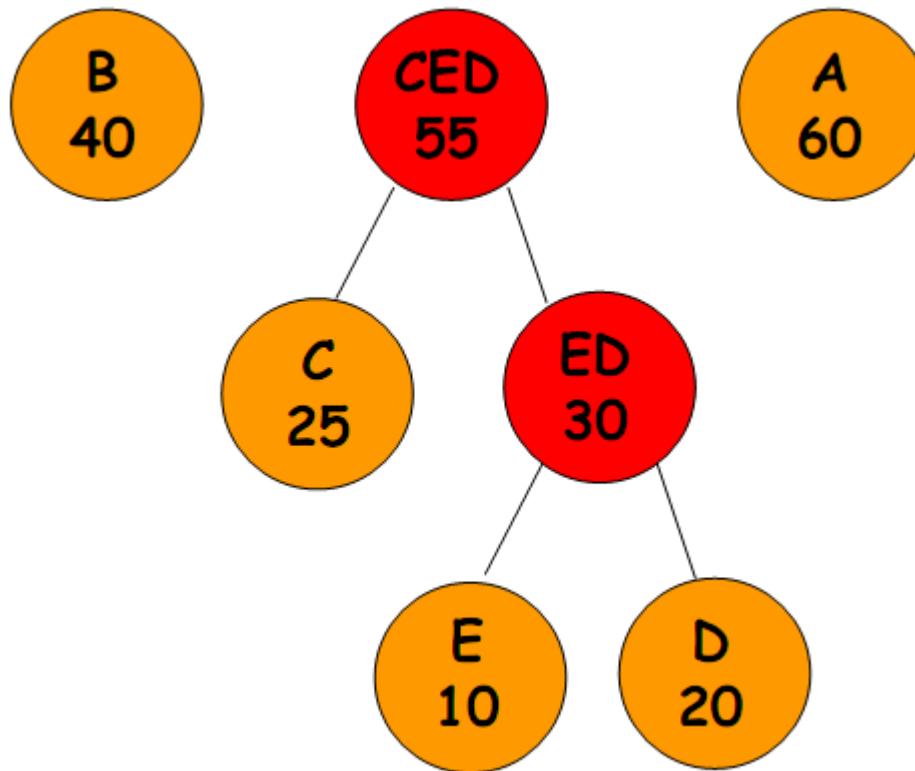
## Örnek: Adım 2

- En küçük frekansa sahip olan iki sembolün frekansları toplanarak yeni bir düğüm oluşturulur. Oluşturulan bu yeni düğüm var olan düğümler arasında uygun yere yerleştirilir. Bu yerleştirme frekans bakımından küçüklük veya büyülükle göredir.

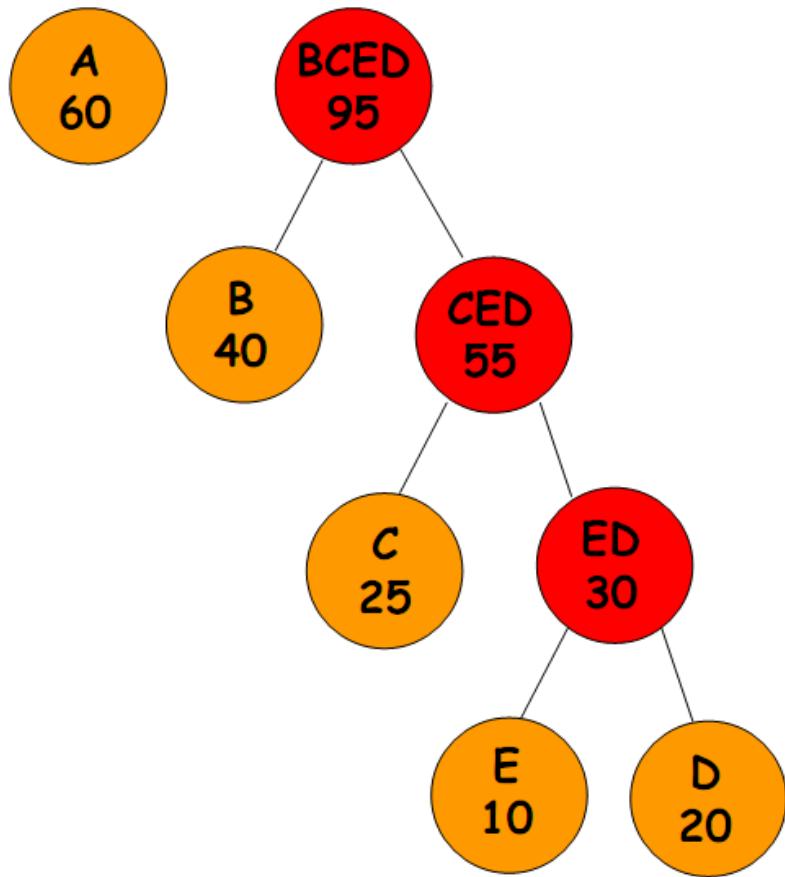


## Örnek: Adım 3

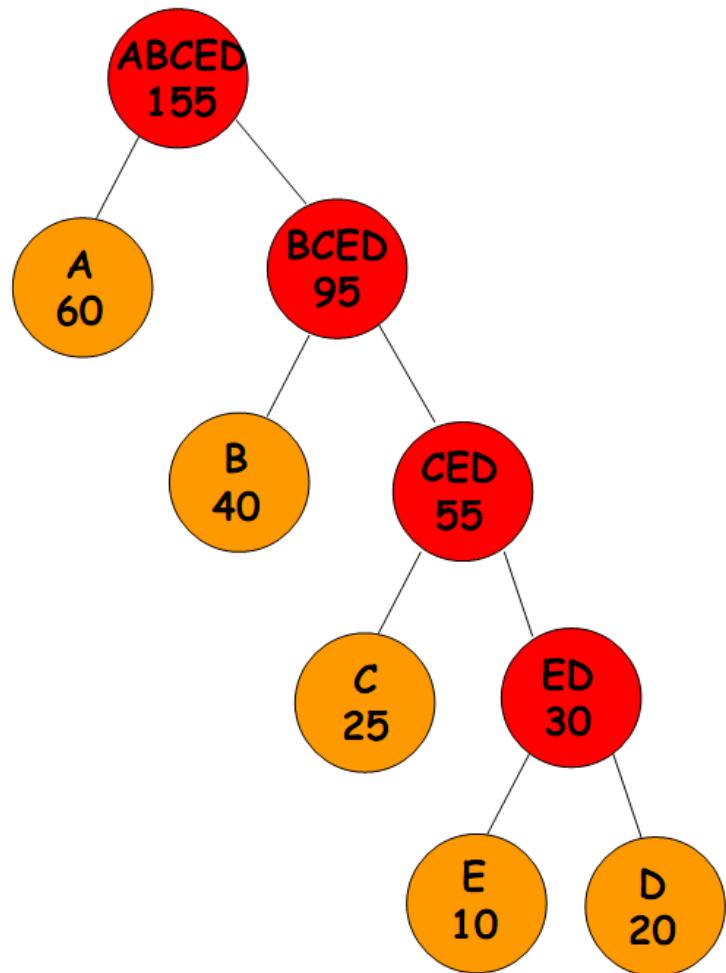
- Bir önceki adımdaki işlem terkarlanılarak en küçük frekanslı 2 düğüm tekrar toplanır ve yeni bir düğüm oluşturulur.



## Örnek: Adım 4



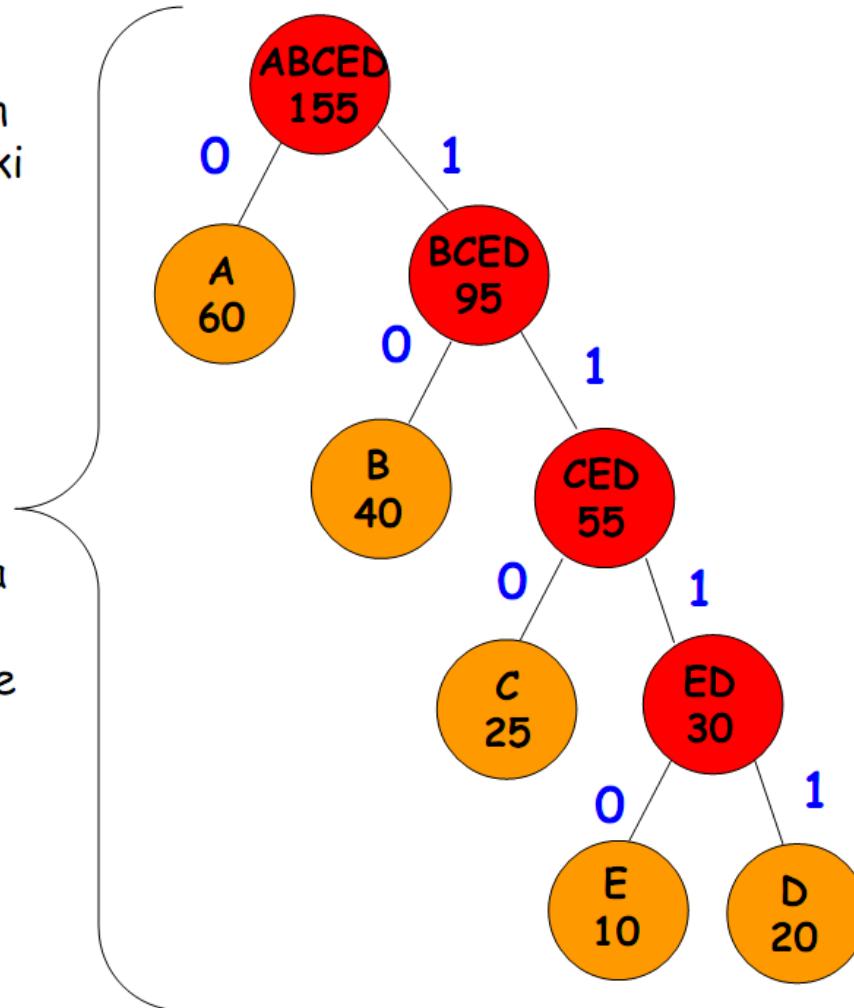
## Adım 5



## Örnek: Adım 6

Huffman ağacının kodu oluşturulurken ağacın en tepesindeki kök düğümden başlanır.

Kök düğümün soluna ve sağına giden dallara sırasıyla **0** ve **1** yazılır. Sırası seçimliktir.



## Örnek: Veri setinin kodlanmış hali

| Sembol | Huffman Kodu | Bit Sayısı | Frekans |
|--------|--------------|------------|---------|
| A      | 0            | 1          | 60      |
| B      | 10           | 2          | 40      |
| C      | 110          | 3          | 25      |
| D      | 1111         | 4          | 20      |
| E      | 1110         | 4          | 10      |

## Örnek: Karşılaştırma

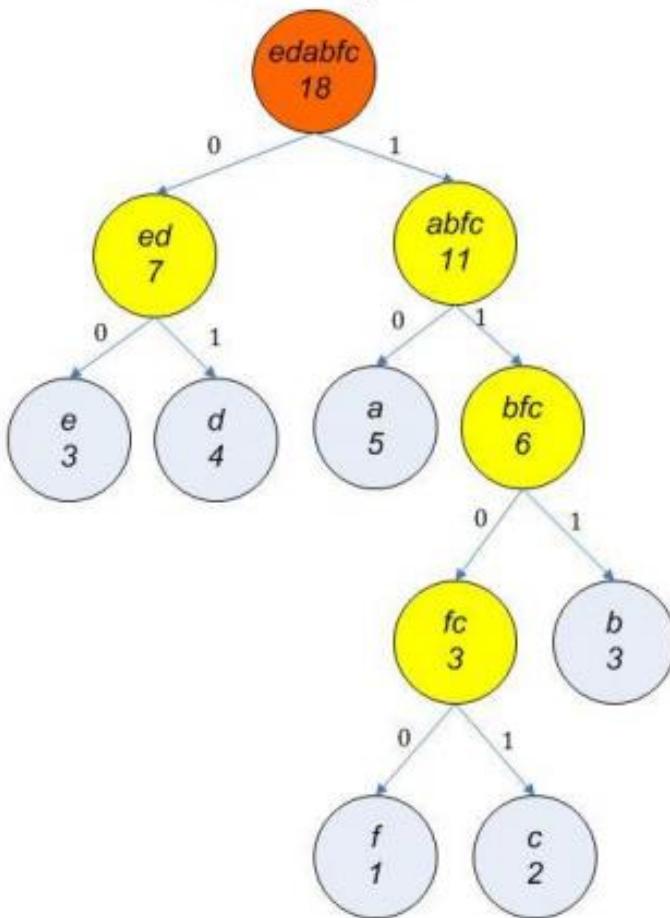
- Veri setini ASCII kodu ile kodlanırsa, her karakter için 1 byte(8 bit)'lık alan gereğinden toplamda 155 byte(1240 bit)'a ihtiyaç olacaktır.
- Huffman kodlaması ile bu sayı :
- $60 \times 1 + 40 \times 2 + 25 \times 3 + 20 \times 4 + 10 \times 4 = 335$  bittir.
- Görüldüğü gibi Huffman kodlaması ile  $335/1240 = \%27$ 'lik bir sıkıştırma oranı sağlanmaktadır. Bu oran, veri setindeki sembollerin frekansları arttıkça daha da artmaktadır.

## Örnek:

Kök Düğüm



Huffman Ağacı



- e: 00
- d: 01
- a: 10
- b: 111
- f: 1100
- c: 1101

## Örnek

- Elimizde sıkıştırılması istenen alttaki katarın olduğunu varsayıalım;
  - aaaabbbccddddeeefa
- Bu katardaki karakter sıklıkları hesaplandığında;
  - $f = 1, c = 2, b = 3, e = 3, d = 4, a = 5$
- ASCII olarak tutulduğunda bu katar toplam  $18 * 8 = 144$  bit yer kaplamaktadır.

## Örnek:

- Sık olan karakterler üst dallarda, seyrek olanlar alt dallarda.
  - Sıkıştırmadan önce
    - 144 bit ( $18 * 8$  bit)
  - Huffman kodu
    - 45 bit (sıklık \* kod sözcüğü uzunluğu) yer
  - Sıkıştırma oranı
    - $45 / 144 = \%31$
- a: 10                      sıklık:5
  - d: 01                      sıklık:4
  - e: 00                      sıklık:3
  - b: 111                     sıklık:3
  - c: 1101                   sıklık:2
  - f: 1100                   sıklık:1

Kodları elde ettikten sonra veri içerisindeki tüm veriler baştan okunur. Her karaktere karşılık gelen kod yerine yerleştirilir.

a a a a b b b c c d d d d e e e f a

10 10 10 10 111 111 111 1101 1101 01 01 01 01 00 00 00 1100 10

verimizin boyutu küçülmüş oldu

## Huffman Kodunun Çözümü

- Frekans tablosu ve sıkıştırılmış veri mevcutsa bahsedilen işlemlerin tersini yaparak kod çözülür. Diğer bir değişle:
- Sıkıştırılmış verinin ilk biti alınır. Eğer alınan bit bir kod sözcüğüne karşılık geliyorsa, ilgili kod sözcüğüne denk düşen karakter yerine konulur.
- Eğer alınan bit bir kod sözcüğü değil ise sonraki bit ile birlikte alınır ve yeni dizinin bir kod sözcüğü olup olmadığına bakılır. Bu işlem dizinin sonuna kadar yapılır. Böylece huffman kodu çözülerken karakter dizisi elde edilmiş olur.

# Açgözlü Algoritmalar (ve Grafikler) Greedy Algorithms (and Graphs)

- Grafik gösterim
- Minimum kapsayan ağaç
- Optimal altyapı
- Açıgözlü seçim
- Prim'in açgözlü MST algoritması

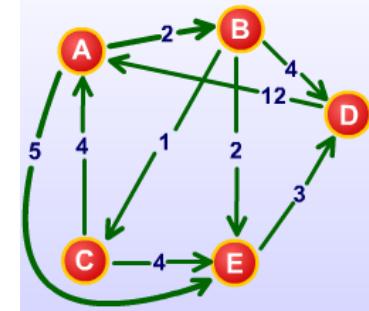
# Graflar (Graphs)

## Konular

- Graflar-Tanım
- Graf Gösterimleri
- Graflarda Dolaşma
  - Breadth- First Search
  - Depth Search
- Hamiltonian cycle
- Euler cycle

# Graflar (Graphs)

## Tanım



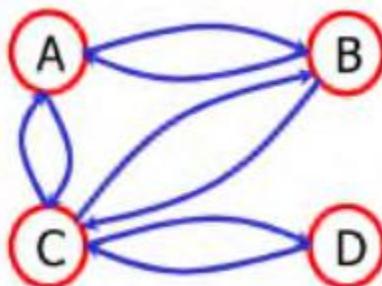
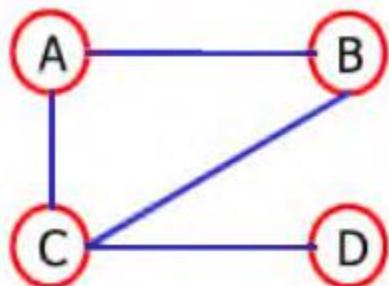
- Graf, matematiksel anlamda, **düğümlerden** ve bu düğümler arasındaki ilişkiyi gösteren **kenarlardan** oluşan bir kümedir. Mantıksal ilişki, düğüm ile düğüm veya düğüm ile kenar arasında kurulur.
  - **Bağlantılı listeler ve ağaçlar** grafların özel örneklerindendir.
- Fizik, Kimya gibi temel bilimlerde ve mühendislik uygulamalarında ve tıp biliminde pek çok problemin çözümü ve modellenmesi graflara dayandırılarak yapılmaktadır.
  - Elektronik devreler, networkler
  - Ulaşım ve iletişim network'leri
  - Herhangi bir türdeki ilişkilerin modellenmesi (parçalar, insanlar, süreçler, fikirler vs.)

# GRAFLAR

- Bir **G** grafi,  $V$  ile gösterilen **düğümlerden** (node veya vertex) ve  $E$  ile gösterilen **kenarlardan** (Edge) oluşur.
  - Her kenar iki düğümü birleştirir.
- Her kenar, iki bilgi (düğüm) arasındaki ilişkiyi gösterir ve  $(u,v)$  şeklinde ifade edilir.
  - $e=(u, v)$  iki düğümü göstermektedir.
  - Bir graf yönlü değil ise  $u$  ve  $v$  düğümleri arasındaki kenar  $(u,v) \in E$  ve  $(v,u) \in E$  olarak gösterilebilir.
- Bir graf üzerinde  $n$  tane düğüm ve  $m$  tane kenar varsa, matematiksel gösterilimi, düğümler ve kenarlar kümesinden elamanlarının ilişkilendirilmesiyle yapılır:
  - $V=\{v_0, v_1, v_2 \dots v_{n-1}, v_n\}$  Düğümler kümesi
  - $E=\{e_0, e_1, e_2 \dots e_{m-1}, e_m\}$  Kenarlar kümesi
  - $G=(V, E) \rightarrow$  Graf

# Graflar

- $G = (V, E)$  grafları aşağıda verilmiştir.

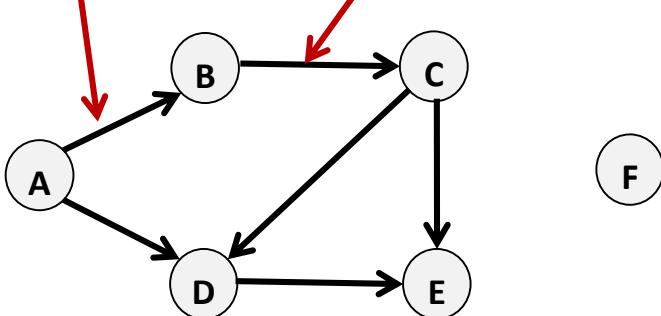


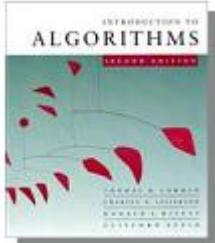
$$V = \{A, B, C, D\}$$

$$E = \{(A, B), (B, A), (A, C), (C, A), (C, D), (D, C), (B, C), (C, B)\}$$

- $V = \{A, B, C, D, E, F\}$

- $E = \{(A, B), (A, D), (B, C), (C, D), (C, E), (D, E)\}$





# Graflar

**Tanım.** *Yönlendirilmiş grafik (digraf)*

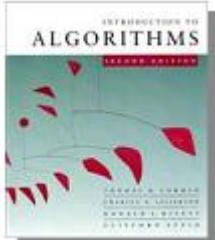
$G = (V, E)$ ,

- $V$  köşeler kümesi ve
- $E \subseteq V \times V$  kenarlar kümelerinden oluşur.

*Yönlendirilmemiş grafik*  $G = (V, E)$ 'de,  $E$  kenar kümesi, sıralı olmayan köşe çiftlerinden oluşur.

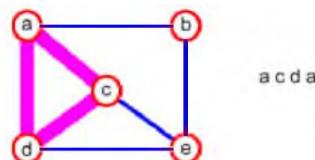
Her durumda, elimizde  $|E| = O(V^2)$  vardır.

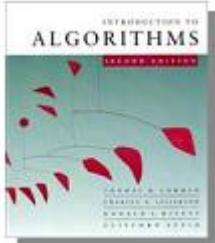
Ayrıca, eğer  $G$  bağılantılıysa,  $|E| \geq |V| - 1$ , ki bu da  $\lg |E| = \Theta(\lg V)$  anlamına gelir.



# Graflar-Tanımlar

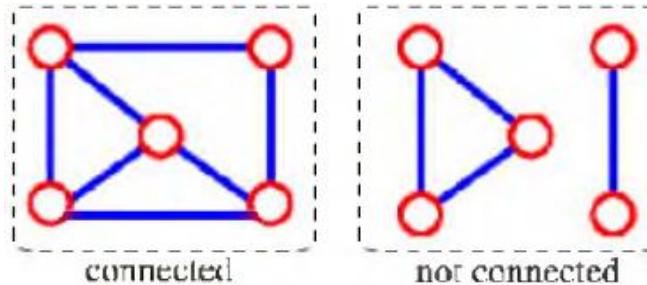
- **Komşu(adjacent):** Eğer  $(u,v) \in E$  ise, u ve düğümleri komşudur.
- **Derece(degree):** Bir düğümün derecesi, komşu düğüm sayısına eşittir.
- **Yol (path):**  $v_0, v_1, v_2 \dots v_n$ , düğümlerinin sırasıdır,  $v_{i+1}$  düğümü  $v_i$  düğümünün komşusudur. ( $i= 0..n-1$ )
- **Basit Yol (simple path):** düğüm tekrarı olmayan yoldur.
- **Döngü(cycle):** basit yoldur, sadece ilk ve son düğüm aynıdır.



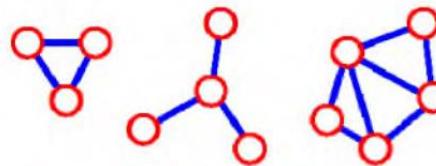


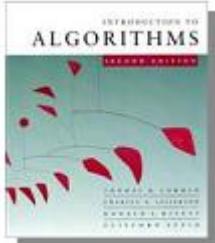
## Graflar-Tanımlar

- **Bağılı Graf(connected graph):** Bütün düğümler bir birine herhangi bir yol ile bağlıdır.



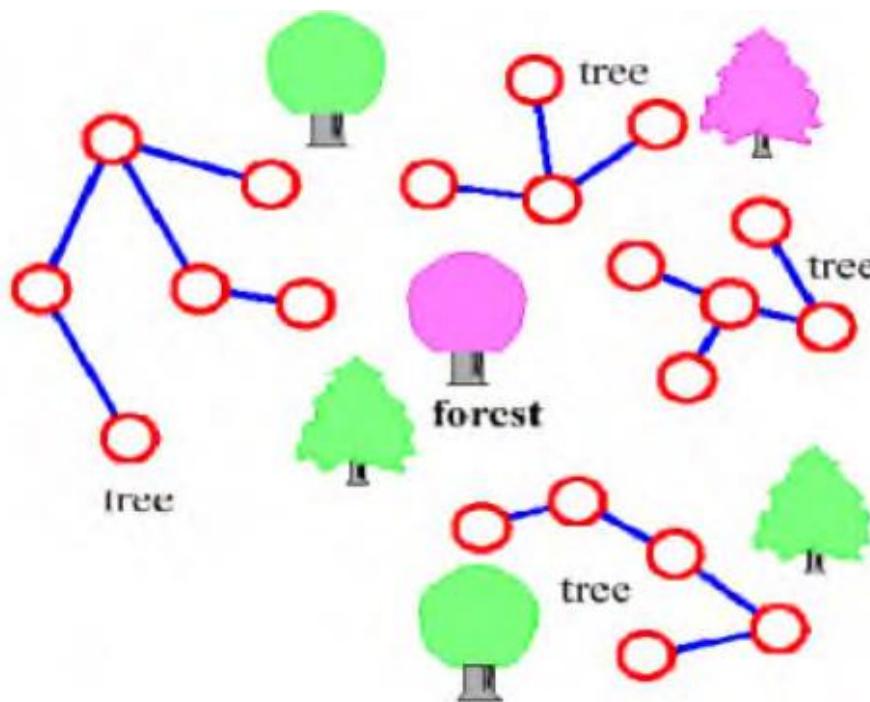
- **Alt graf(Subgraph):** Bir grafi oluştururan alt düğümler ve kenarlar kümesidir.
- **Bağılı elaman (Connected component):** bağlı alt graflar. Örnek: 3 tane bağlı elamana sahip graflar





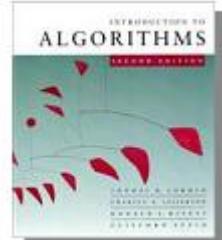
## Graflar-Tanımlar

- **Ağaç (tree):** Döngü olmayan bağlı graflardır.
- **Orman(forest):** Ağaçların toplamıdır.



# Graflar için Veri yapıları

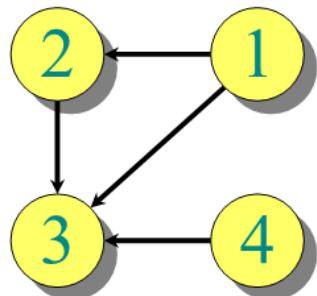
## Komşuluk matrisi gösterimi



Bir  $G = (V, E)$  grafiginin *komşuluk matrisi*  
 $V = \{1, 2, \dots, n\}$  iken,

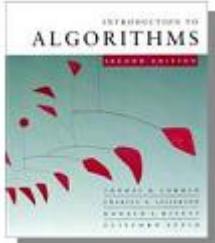
$$A[i, j] = \begin{cases} 1 & \text{eğer } (i, j) \in E \text{ ise,} \\ 0 & \text{eğer } (i, j) \notin E \text{ ise,} \end{cases}$$

$A[1 \dots n, 1 \dots n]$  matrisidir.



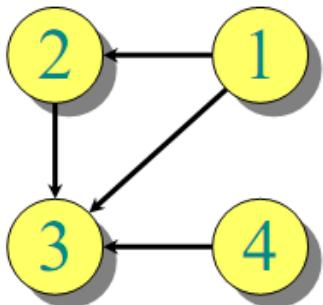
| $A$ | 1 | 2 | 3 | 4 |
|-----|---|---|---|---|
| 1   | 0 | 1 | 1 | 0 |
| 2   | 0 | 0 | 1 | 0 |
| 3   | 0 | 0 | 0 | 0 |
| 4   | 0 | 0 | 1 | 0 |

$\Theta(V^2)$  depolama:  
 $\Rightarrow$  **yoğun**  
gösterimi.



## Komşuluk listesi gösterimi

Bir  $v \in V$  köşesinin *komşuluk listesi*,  $v'$  ye komşu olan köşelerin  $Adj[v]$  listesidir.



$$\begin{aligned}Adj[1] &= \{2, 3\} \\Adj[2] &= \{3\} \\Adj[3] &= \{\} \\Adj[4] &= \{3\}\end{aligned}$$

Yönlendirilmemiş grafikler için,  $|Adj[v]| = \text{degree}(\text{derece})(v)$ .

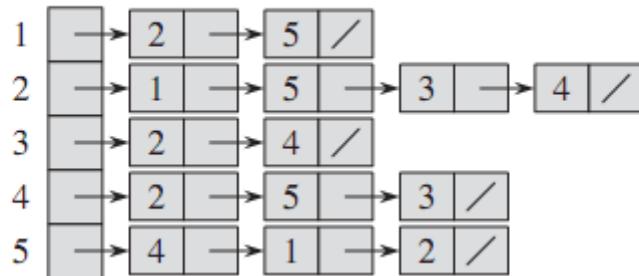
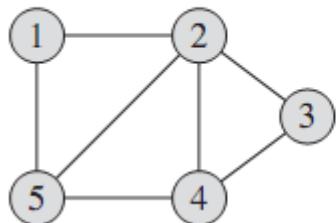
Yönlendirilmiş grafikler için,  $|Adj[v]| = \text{out-degree}(\text{dış-derece})(v)$ .

### Handshaking Lemma

**Tokalasma önkuramı:** Yönlendirilmemiş grafikler için  $\sum_{v \in V} |Adj[v]| = 2|E| \Rightarrow$  komşuluk listeleri,  $\Theta(V + E)$  depolama alanını kullanır. Bu seyrek gösterimdir. (her tür grafik için.)

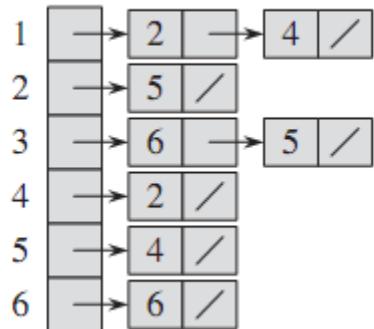
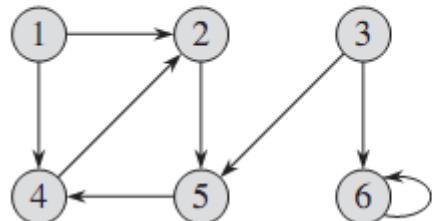
# Komşuluk listesi gösterimi

- Yönsüz graf komşuluk listesi



|   | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

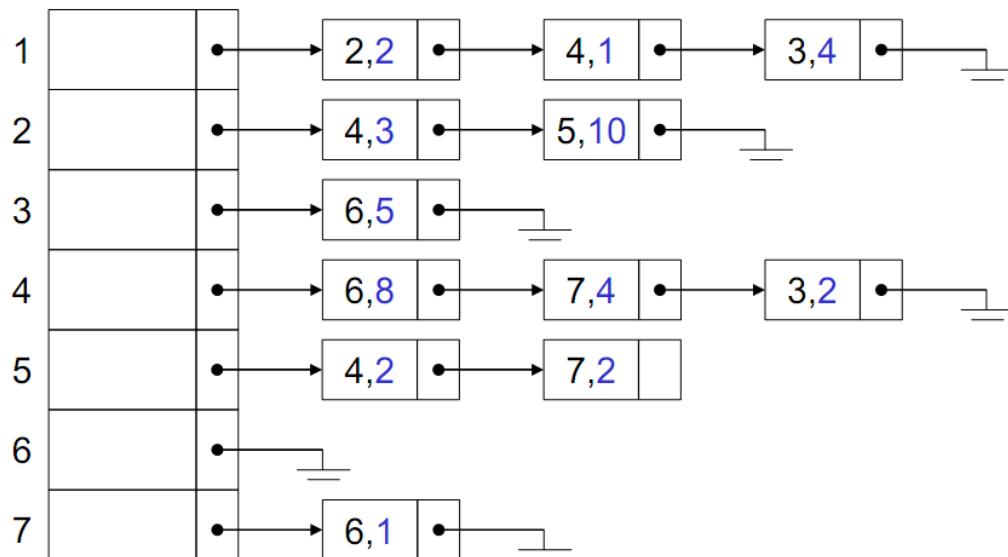
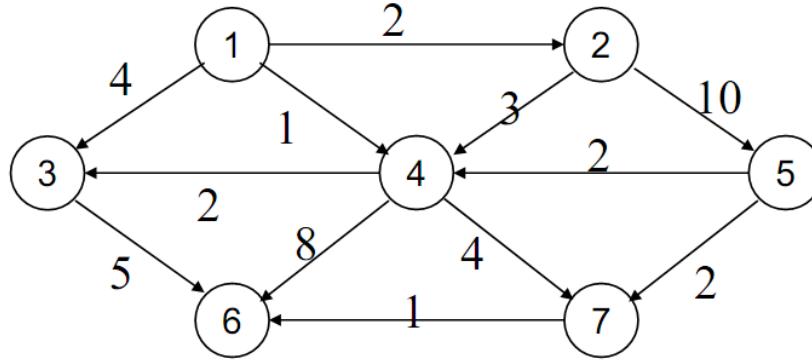
- Yönlü graf komşuk listesi



|   | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 1 | 0 | 0 |
| 2 | 0 | 0 | 0 | 0 | 1 | 0 |
| 3 | 0 | 0 | 0 | 0 | 1 | 1 |
| 4 | 0 | 1 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 1 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 1 |

# Komşuluk listesi gösterimi

- Her kenarı ağırlıklandırılmış yönlü graf komşuluk listesi



# Graf Arama Algoritmaları

- Graf içindeki her kenar ve düğüm için sistematik arama.
- $G=(V,E)$  grafi yönlü olabilir veya olamayabilir.
- Uygulamalar
  - Derleyiciler(Compilers)
  - Grafikler(Graphics)
  - Haritalama(Mapping)
  - Ağlar(Networks):routing, searchind, clustering, vs.

# Enine Arama

## Breadth First Search (BFS)

- BFS, bir grafın bağlı olan parçalarını dolasır ve bir kapsayan ağaç (spaning tree) oluşturur.
- BFS, arama ağaçlarındaki level order aramaya benzer.
- 1-Seçilen düğümün tüm komşuları sırayla seçilir ve ziyaret edilir.
- 2- Her komşu kuyruk içerisine atılır.
- 3- Komşu kalmadığında kuyruk içerisindeki ilk düğüm alınır ve 2. adıma gidilir.

# Breadth First Arama-düğüm renklendirme

- Ulaşılmayan bir düğüm beyaz renklidir.
- Bir düğüm, eğer kendine ulaşılmış ancak tüm kenarlarına bakılmamışsa gri renklidir.
- Bir düğüm, eğer kendisinin tüm komşu düğümlerine ulaşıldıysa (komşuluk listesi tamamen incelenir) siyah renklidir.

$\text{BFS}(G, s)$

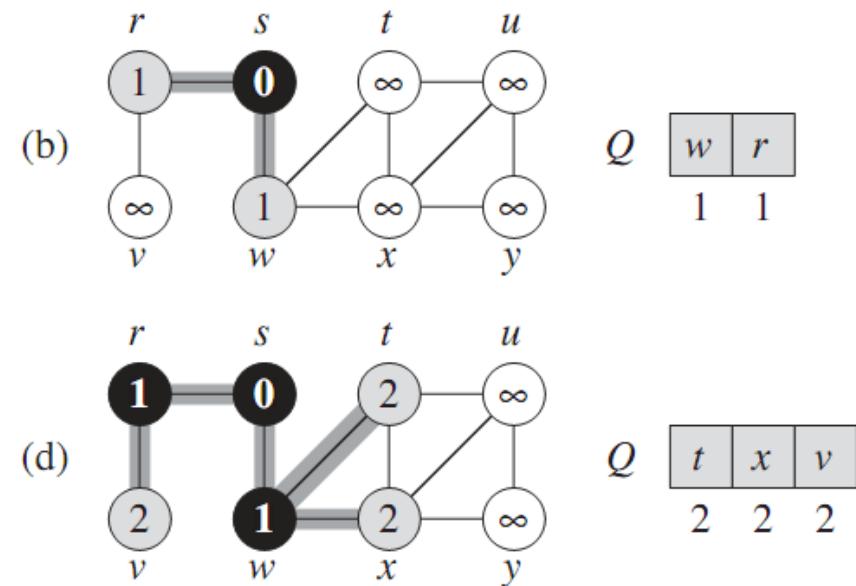
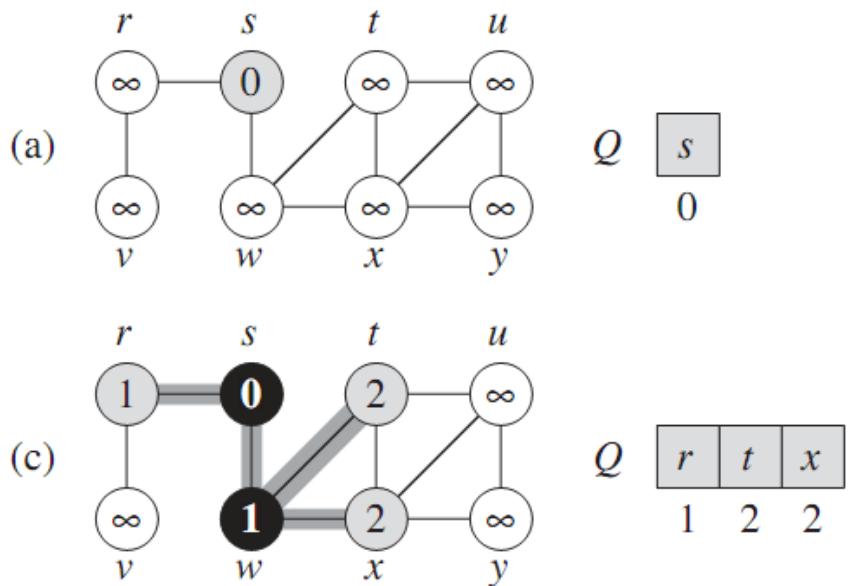
```

1 for each vertex $u \in G.V - \{s\}$
2 $u.\text{color} = \text{WHITE}$
3 $u.d = \infty$
4 $u.\pi = \text{NIL}$
5 $s.\text{color} = \text{GRAY}$
6 $s.d = 0$
7 $s.\pi = \text{NIL}$
8 $Q = \emptyset$
9 ENQUEUE(Q, s)
10 while $Q \neq \emptyset$
11 $u = \text{DEQUEUE}(Q)$
12 for each $v \in G.\text{Adj}[u]$
13 if $v.\text{color} == \text{WHITE}$
14 $v.\text{color} = \text{GRAY}$
15 $v.d = u.d + 1$
16 $v.\pi = u$
17 ENQUEUE(Q, v)
18 $u.\text{color} = \text{BLACK}$

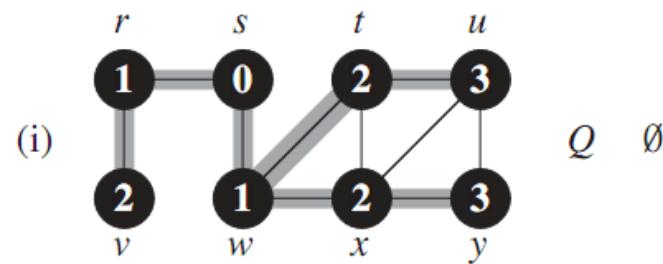
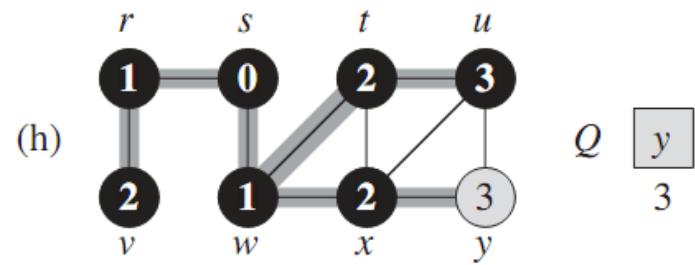
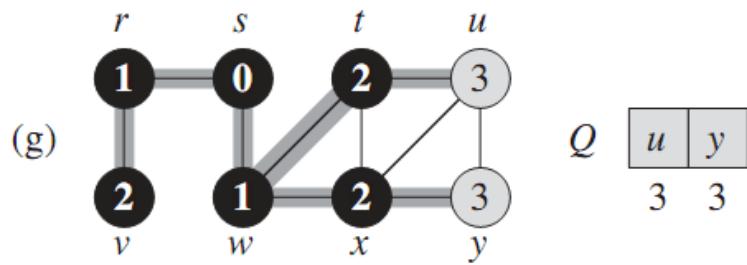
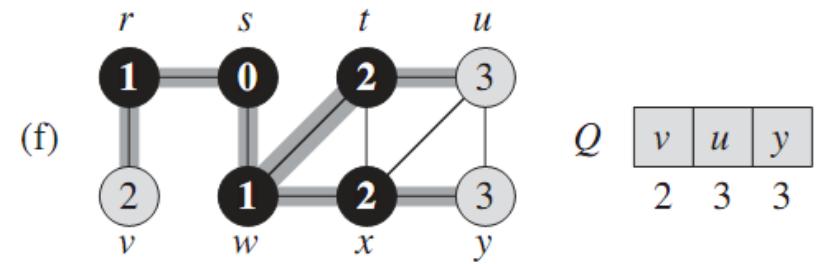
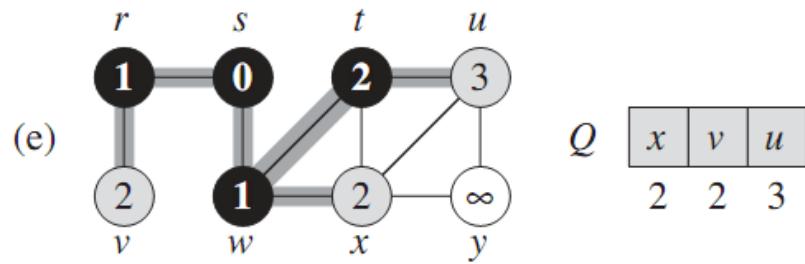
```

# Breadth First Arama Örnek

- S düğümü ile başla



# Breadth First Arama Örnek



# BFS algoritmasının Analizi

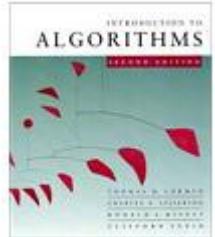
- $G = (V, E)$  grafi için
  - Beyaz renkli tüm düğümler sıraya konur. Kuyruğa ekleme ve çıkarma işlemleri  $O(1)$  süresinde yapılrsa toplam çalışma süresi düğüm sayısı kadardır,  $O(V)$ .
  - Bir düğüm kuyruktan çıkarıldığında komşuluk listesi taranır. Toplamda komşuluk listesini taramanın çalışma süresi  $O(E)$  olur.
  - BFS nin toplam çalışma zamanı:  $O(V+E)$  olur.

$\text{BFS}(G, s)$

```

1 for each vertex $u \in G.V - \{s\}$
2 $u.\text{color} = \text{WHITE}$
3 $u.d = \infty$
4 $u.\pi = \text{NIL}$
5 $s.\text{color} = \text{GRAY}$
6 $s.d = 0$
7 $s.\pi = \text{NIL}$
8 $Q = \emptyset$
9 ENQUEUE(Q, s)
10 while $Q \neq \emptyset$
11 $u = \text{DEQUEUE}(Q)$
12 for each $v \in G.\text{Adj}[u]$
13 if $v.\text{color} == \text{WHITE}$
14 $v.\text{color} = \text{GRAY}$
15 $v.d = u.d + 1$
16 $v.\pi = u$
17 ENQUEUE(Q, v)
18 $u.\text{color} = \text{BLACK}$

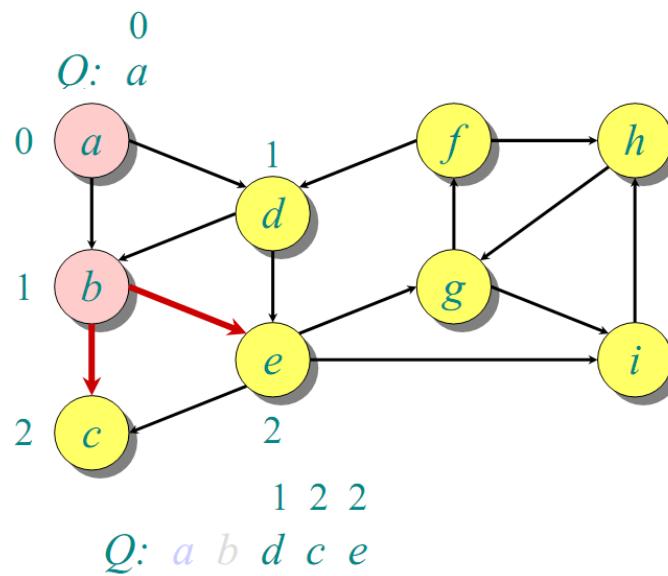
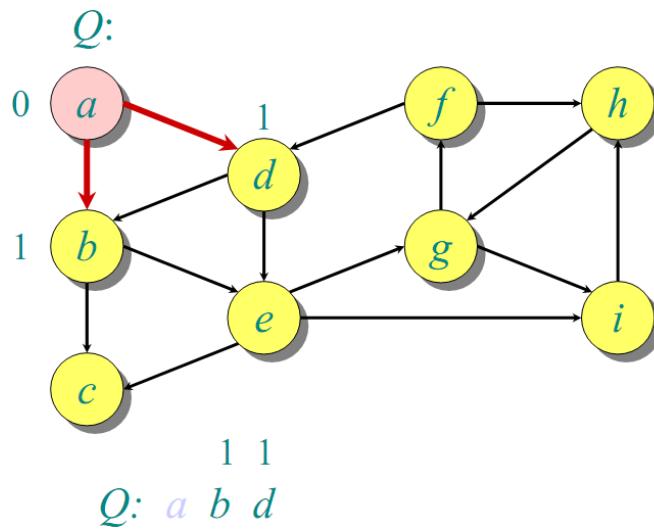
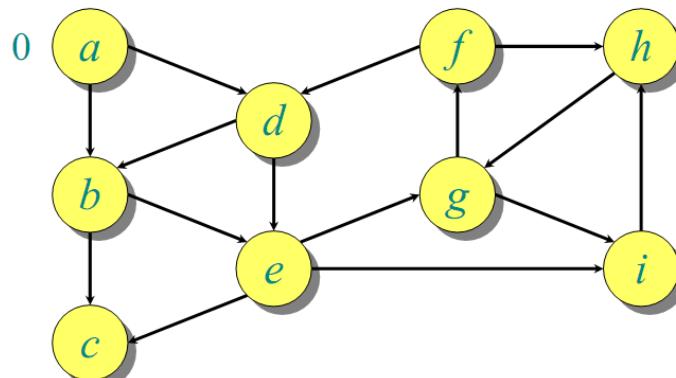
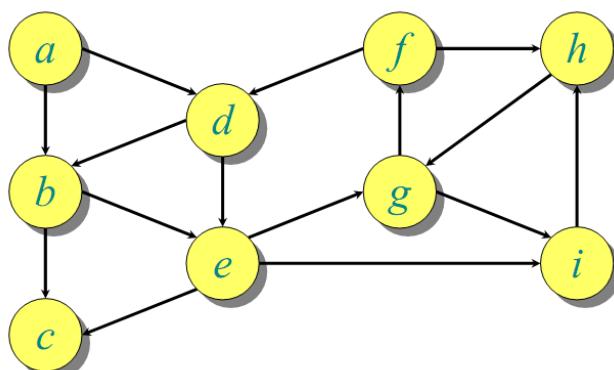
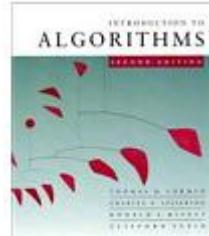
```

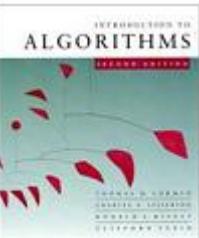


## BFS Özellikleri

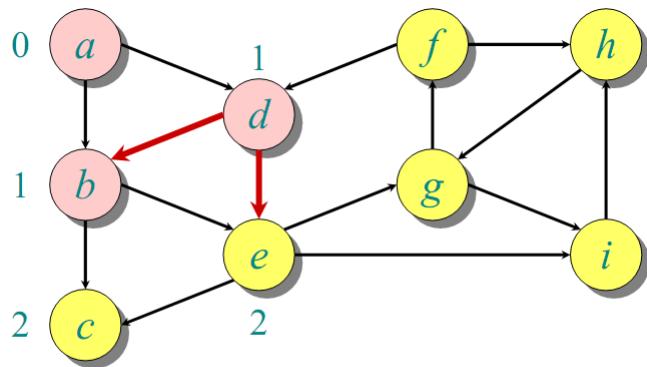
- $G=(V,E)$  grafi için BFS,  $s$  düğümünden ulaşılabilecek tüm düğümleri dolaşır.
- Ulaşılabilen düğümler için en kısa yolu hesaplayabilir.
- Ulaşılabilen tüm düğümler için BF ağacını oluşturur.
- $s$  düğümünden ulaşılan bir  $v$  düğümü için BF ağacı içindeki yol,  $G$  grafındaki en kısa yolu (shortest path) ifade eder.

# Yönlü Graf(Digraph) için Enine arama Örnek

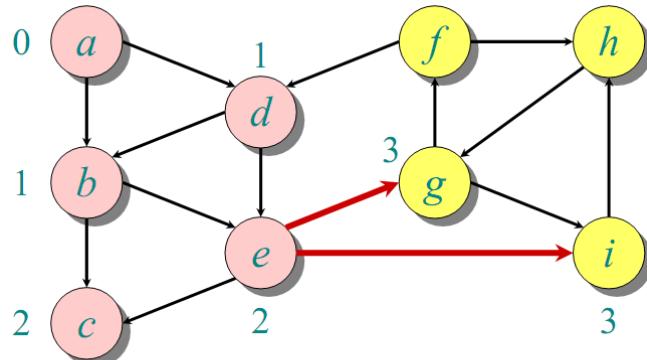
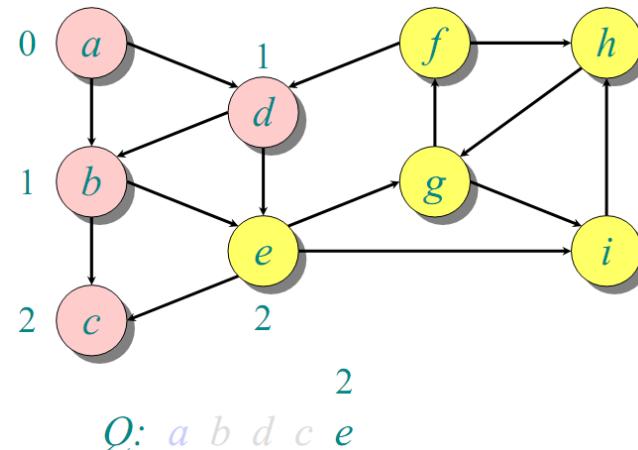




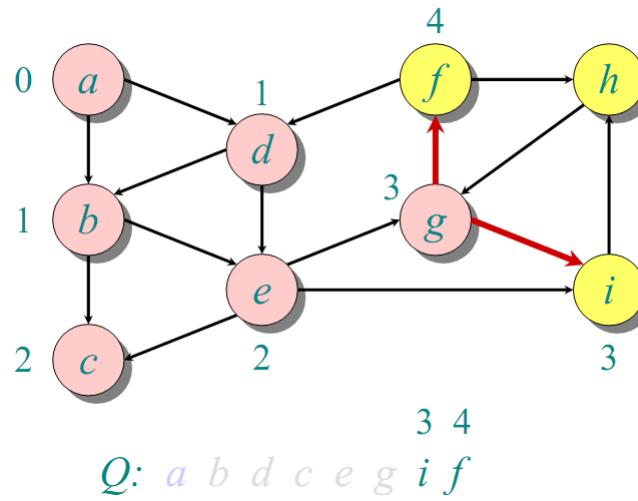
## Enine arama için örnek



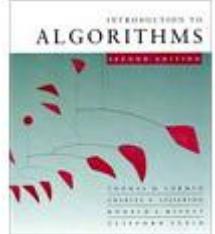
$Q: a \ b \ d \ c \ e$



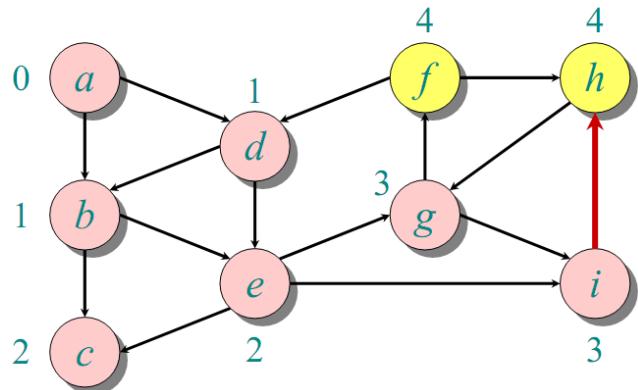
$Q: a \ b \ d \ c \ e \ g \ i$



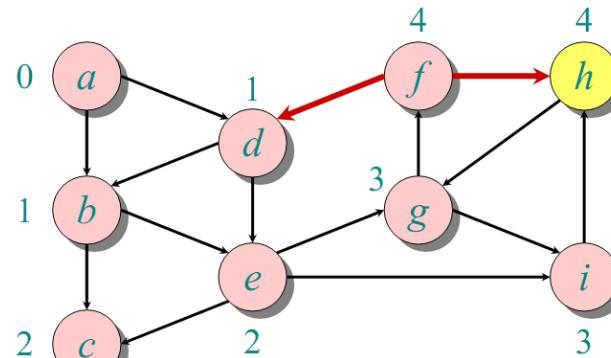
$f$   
 $h$   
 $g$   
 $i$



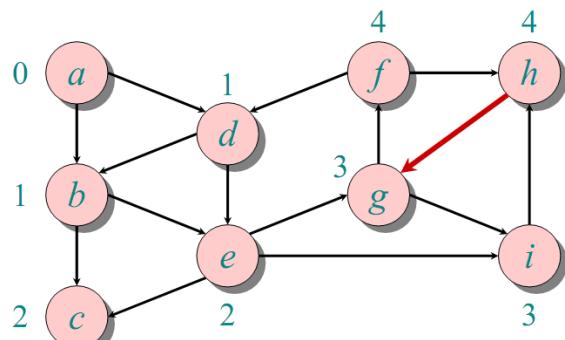
## Enine arama için örnek



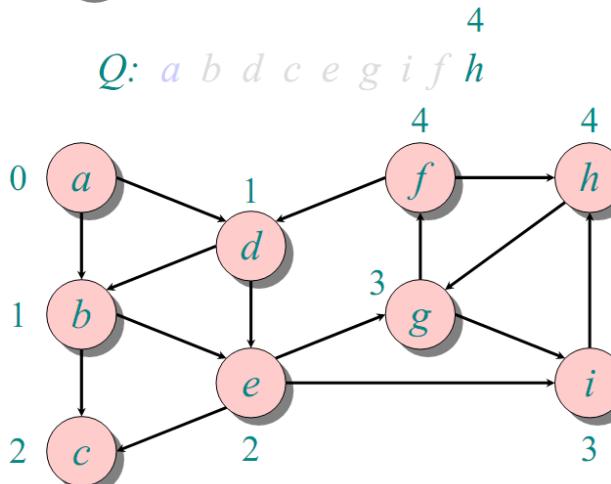
$Q: \text{a } b \text{ } d \text{ } c \text{ } e \text{ } g \text{ } i \text{ } f \text{ } h$



$Q: \text{a } b \text{ } d \text{ } c \text{ } e \text{ } g \text{ } i \text{ } f \text{ } h$

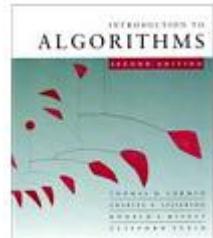


$Q: \text{a } b \text{ } d \text{ } c \text{ } e \text{ } g \text{ } i \text{ } f \text{ } h$



$Q: \text{a } b \text{ } d \text{ } c \text{ } e \text{ } g \text{ } i \text{ } f \text{ } h$

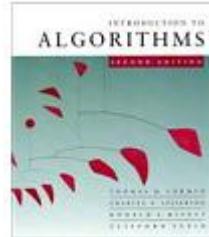
# Derinlemesine Arama Depth First Search(DFS)



- 1- DFS ile önce bir başlangıç düğümü seçilir ve ziyaret edilir.
- 2- Seçilen düğümün bir komşusu seçilir ve ziyaret edilir.
- 3- Seçilen komşu düğümün bir komşusu seçilir ve ziyaret edilir.
- 4- 3. adım koşu düğüm kalmayınca kadar devam eder.
- 5- Komşu kalmadığında geri dönülür (backtracking) ve her düğüm için yeniden 3.adıma gidilir.

# Derinlemesine Arama

## Depth First Search(DFS)



**DFS(G,s)**

```

for each vertex $u \in V[G]$
 color[u] \leftarrow WHITE
 $\Pi[u] \leftarrow$ NIL
 time $\leftarrow 0$
for each vertex $u \in V[G]$
 if color[u] == WHITE
 then DFS-VISIT(u)

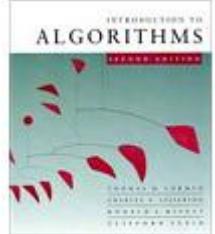
```

**DFS-VISIT( $u$ )**

```

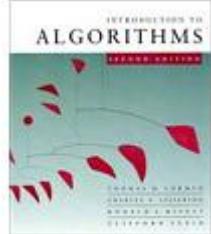
color[u] \leftarrow GRAY // white vertex u has just been discovered
d[u] \leftarrow time \leftarrow time + 1
for each vertex $v \in \text{Adj}[u]$ // explore edge (u,v)
 do if color[v] == WHITE
 then $\Pi[v] \leftarrow u$
 DFS-VISIT(v)
color[u] \leftarrow BLACK // Blaken u ; it is finished
f[u] \leftarrow time \leftarrow time + 1

```



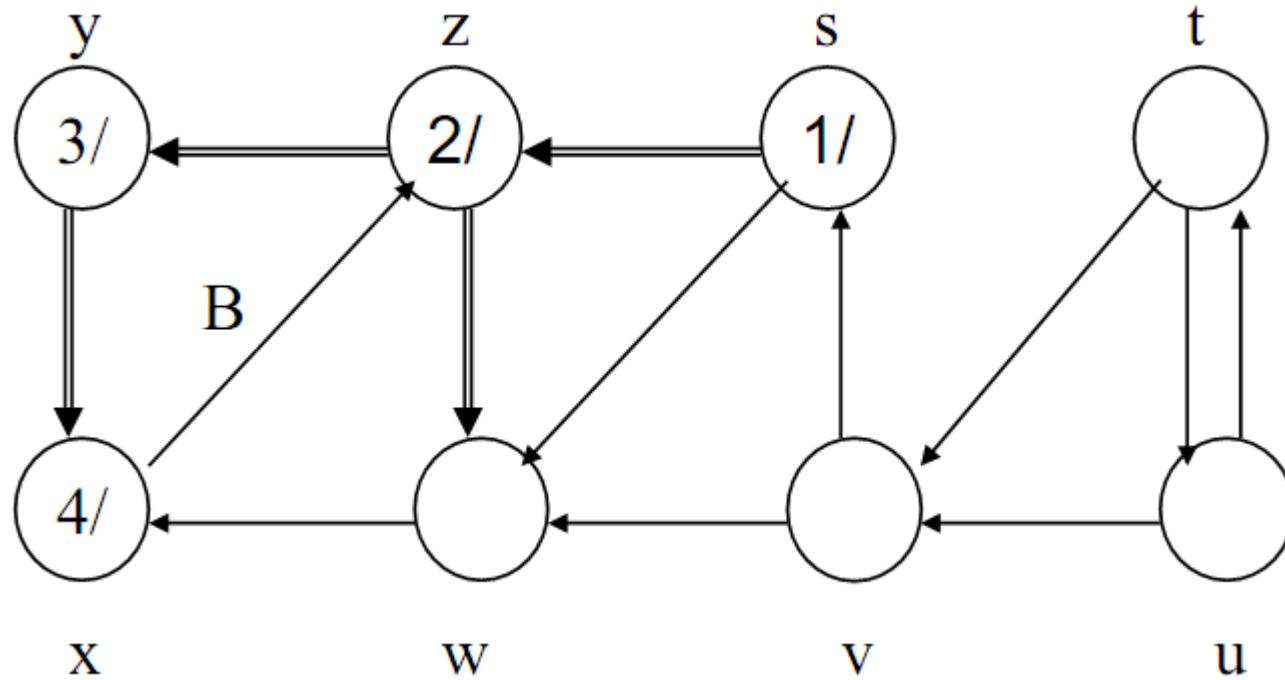
## Derinlemesine Arama-(DFS)

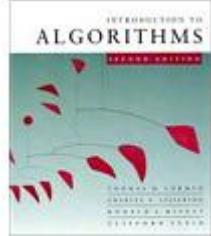
- Başlatma(initialize)- tüm düğümler beyaz yapılır.
- Tüm beyaz düğümler DFS-Visit kullanılarak dolaşılır.
- DFS-Visit( $u$ ) çağrıldığında  $u$  düğümü yeni ağacın kökü yapılır.
- DFS çalışmasını bitirince, her  $u$  düğümü için erişilme zamanı (discovery time)  $d[u]$ , ve bitirme zamanı (finishing time)  $f[u]$  değerleri atanır.
- **Back Edge(B):** DF ağacında  $u$  düğümünün  $v$  atasına (ancestor) bağlı olması.
- **Forward Edge (F):** DF ağacında  $u$  düğümünün,  $v$  toruna (descendant) bağlı olması.
- **Cross Edge(C):** Geri kalan tüm kenarlar. Bu kenarlar ile aynı yada farklı DF ağacında, atası olmadığı diğer düğümler arasında dolaşabilir.



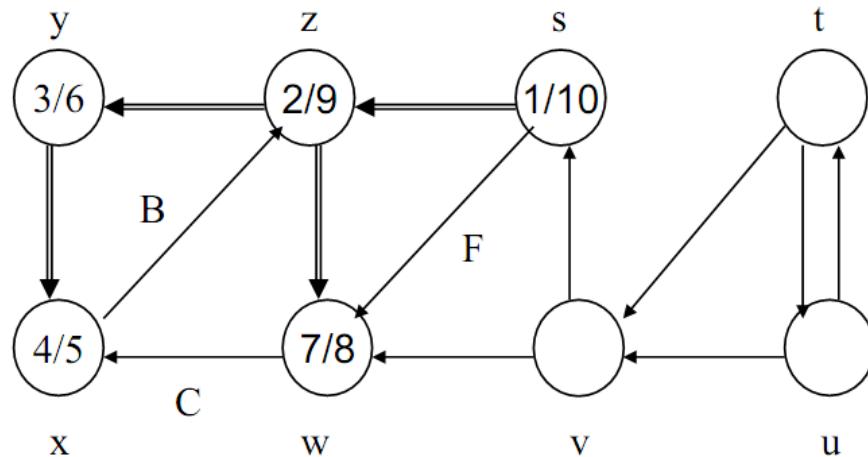
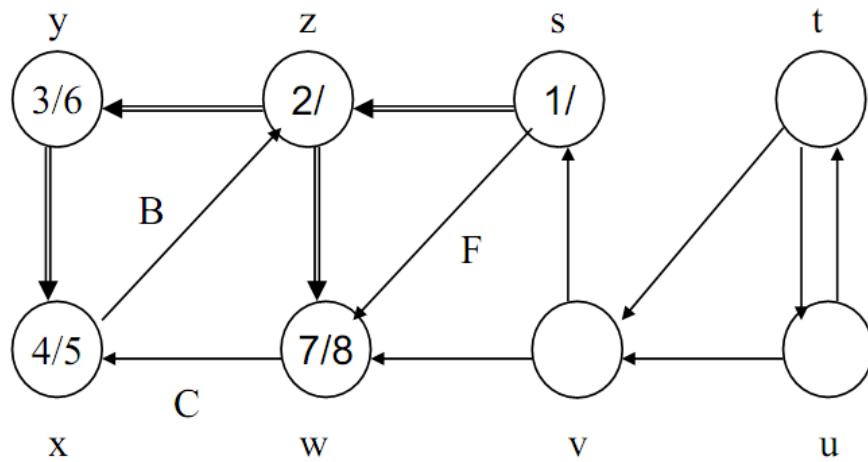
## DFS-Örnek

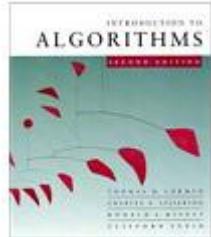
- S düğümü ile başla,
- d/f → d: erişilme zamanı, f: bitirme zamanı



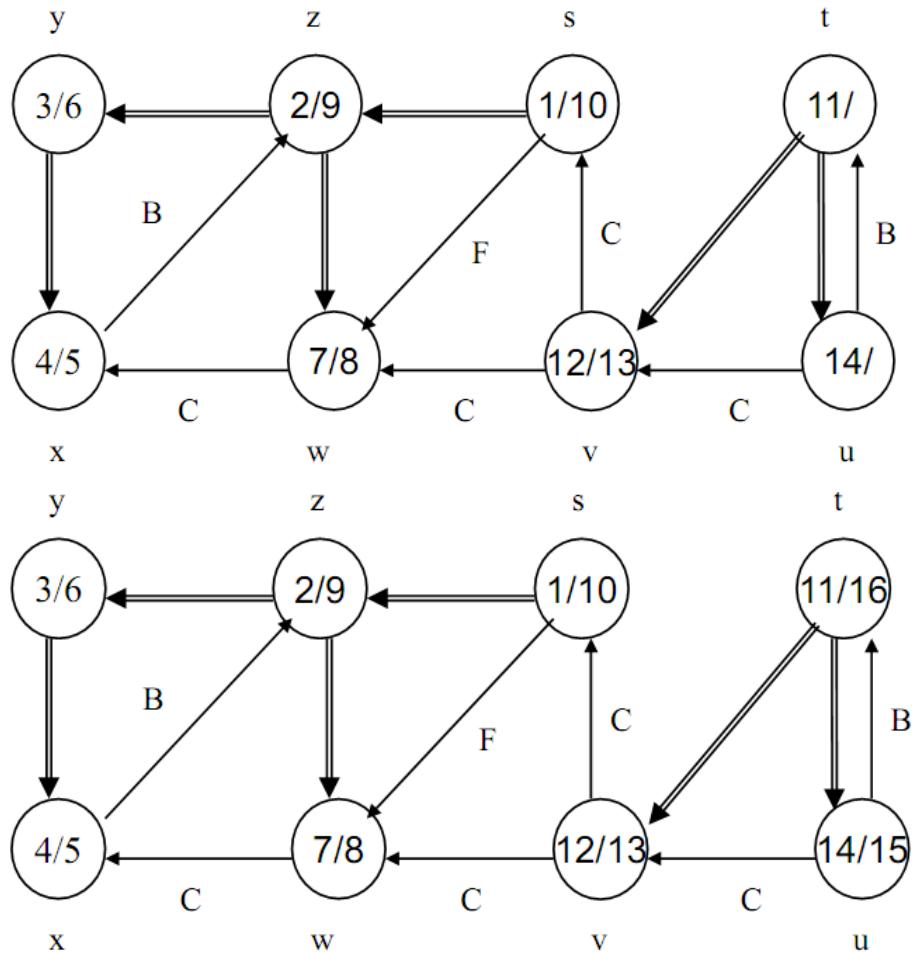


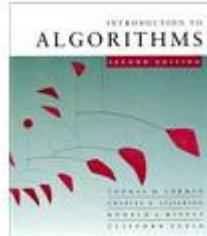
## DFS-Örnek



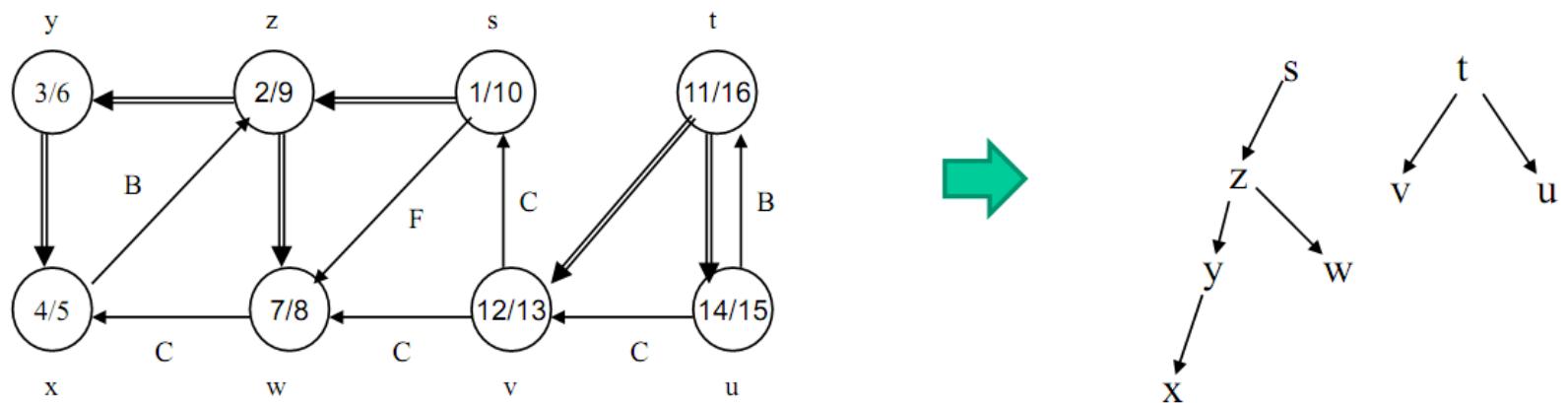


## DFS-Örnek

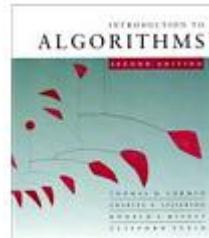




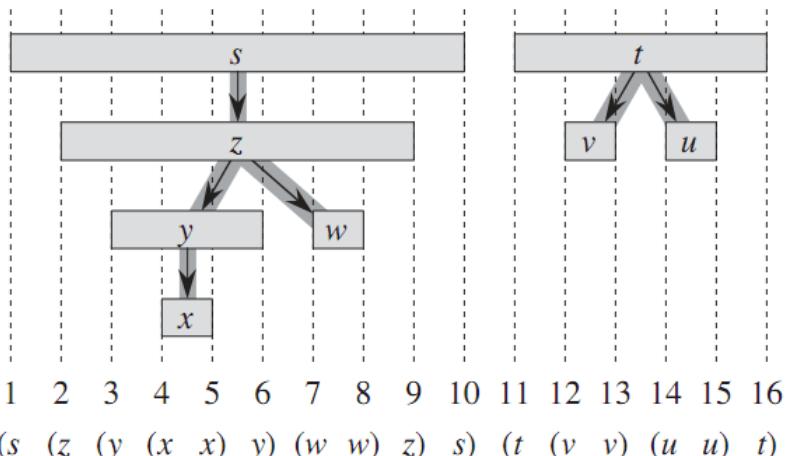
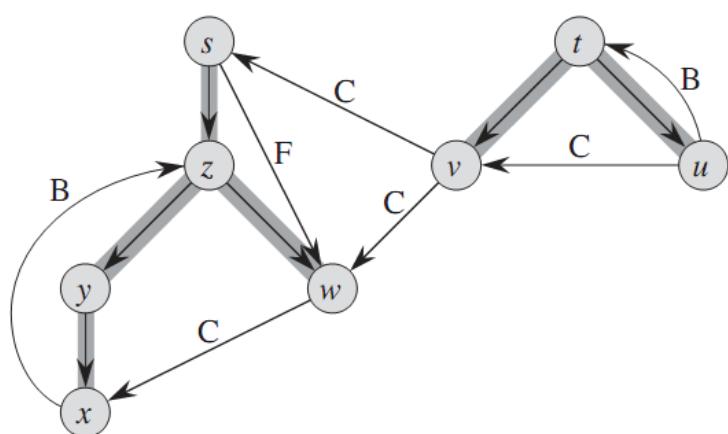
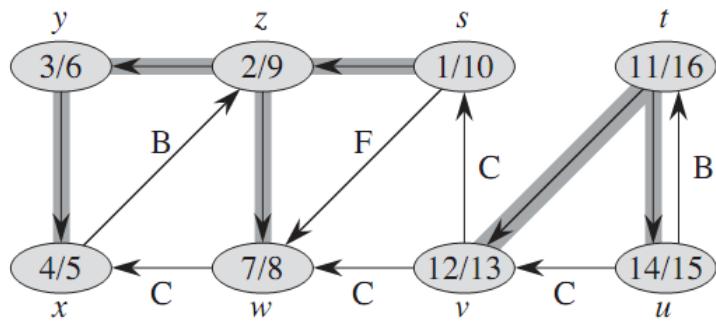
## DFS-Örnek

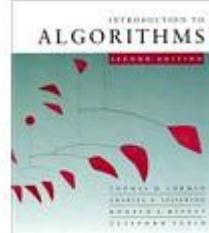


|    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| (s | (z | (y | (x | x) | y) | (w | w) | z) | s) | (t | (v | v) | (u | u) | t) |

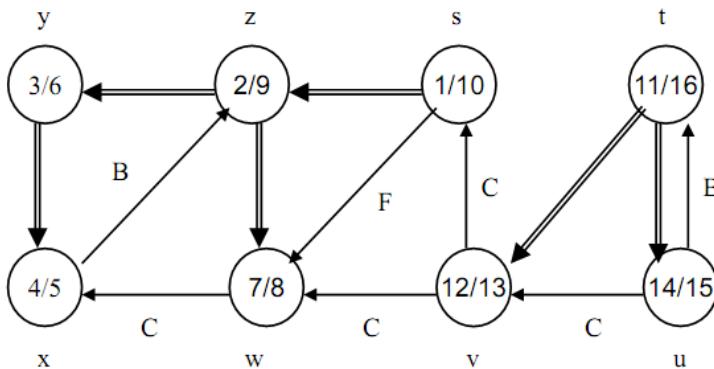


## DFS-Örnek

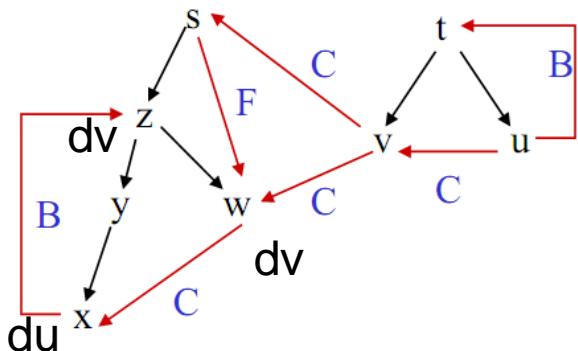




# DFS-Örnek



du



Edge (u,v) is:

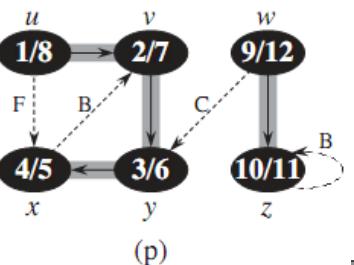
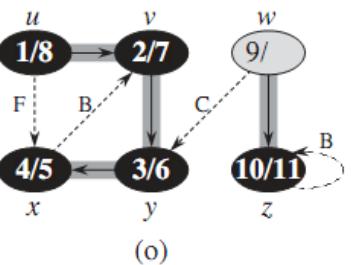
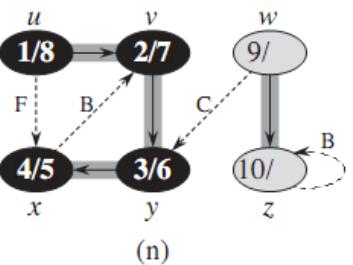
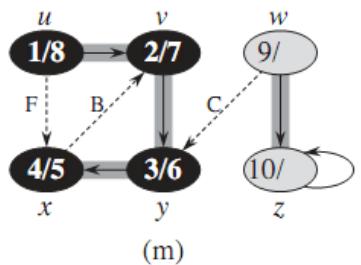
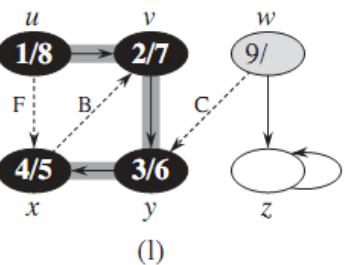
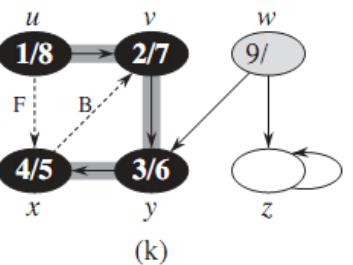
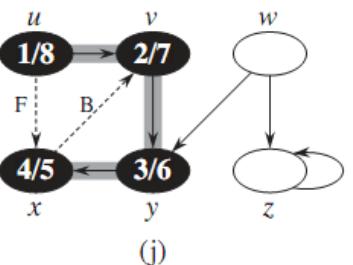
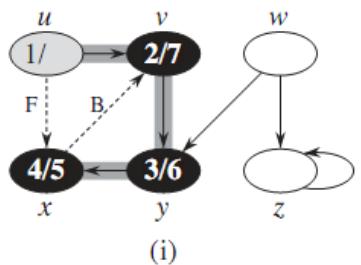
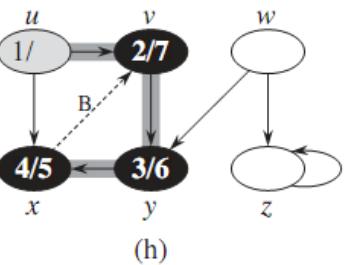
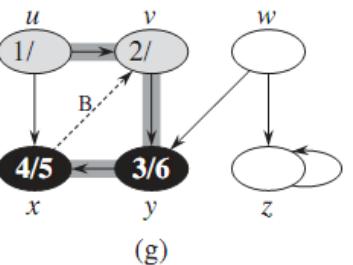
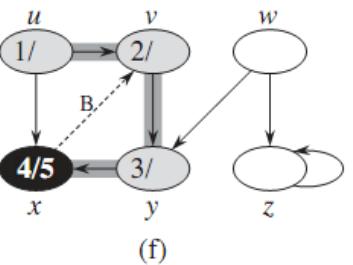
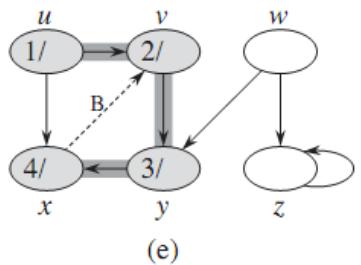
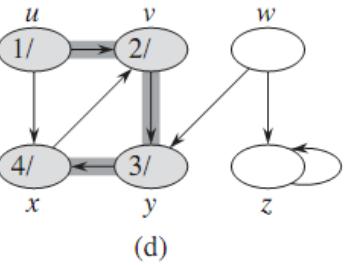
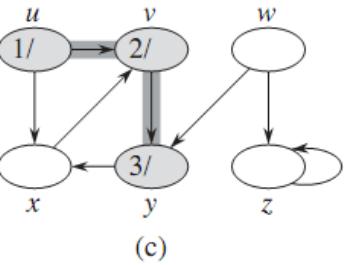
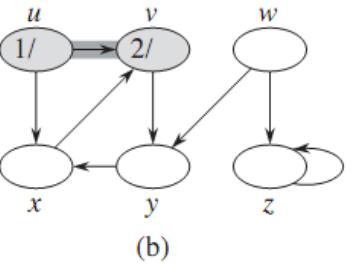
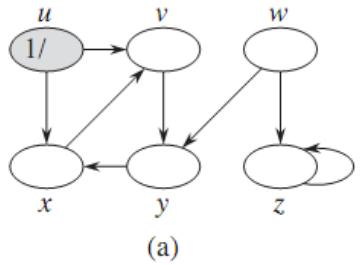
A **tree edge** if  $d[u] < d[v] < f[v] < f[u]$

A **forward edge** if  $d[u] < d[v] < f[v] < f[u]$   
and  $(u,v)$  is not a tree edge

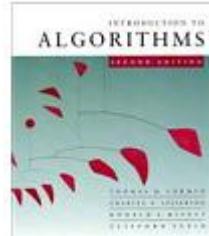
A **back edge** iff  $d[v] < d[u] < f[u] < f[v]$

A cross edge otherwise.

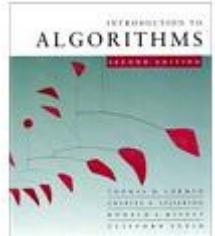
# DFS-Örnek



# DFS-Çalışma Zamanı ve düğüm renklendirme



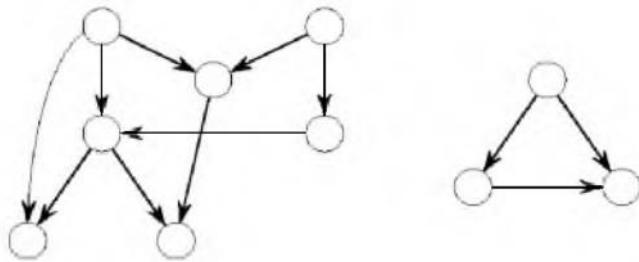
- Çalışma zamanı:
  - DFS içindeki döngü  $\Theta(V)$  süresi alır (DFS-Visit çalışma süresi hariç).
  - DFS-Visit her düğüm için bir kez çağrırlır.
    - Sadece beyaz renkteki düğümler için çağrırlır
    - Düğüm hemen gri renk yapılır.
  - DFS-Visit her çağrılmışında  $|adj[v]|$  kez çalışır      
$$\sum_{v \in V} |Adj[v]| = \Theta(E)$$
  - DFS –Visit için toplam çalışma zamanı  $\rightarrow \Theta(E)$
  - DFS için toplam çalışma zamanı  $\rightarrow \Theta(V+E)$
- Düğüm u,
- $d[u]$  zamanından önce beyazdır.
- $d[u]$  ve  $f[u]$  zamanda gridir, bundan sonra ise siyahdır.



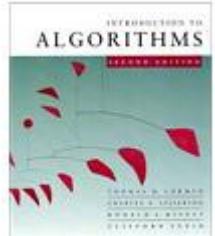
# Döngü olmayan yönlü graflar (DAG)

## DAG-Directed Acyclic Graphs

- Bir DAG döngü olmayan yönlü bir graftır.

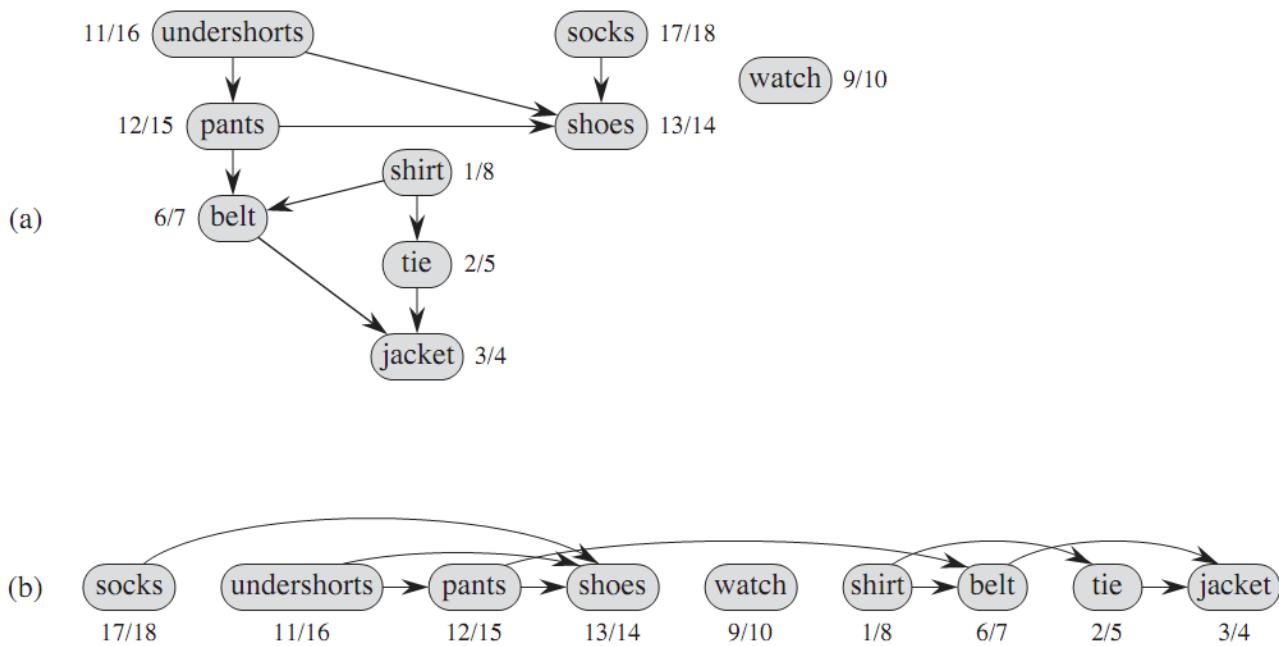


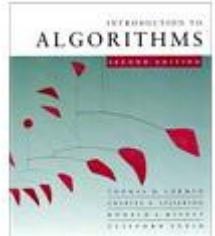
- Sıklıkla olaylar arasındaki öncelik sırasını göstermek için kullanılır. Örnek a olayı b olayından önce olmak zorundadır (paralel kod çalışma).
- Tüm sıra topolojik sıralamayla (topological sort) verilebilir.



# Topolojik Sıralama

- Öncelik İlişkileri (Precedence relations):  $x'$  ten  $y'$  ye bir kenar,  $x$  olayı  $y$  den önce olur anlamını içerir. Bir iş, kendisinin tüm alt işleri planladıkten sonra planlanabilir.





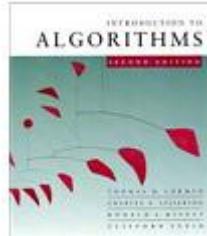
# Topolojik Sıralama

- Bir DAG'nın topolojik sıralaması tüm düğümlerin doğrusal sırasını ifade eder. **Grafta döngü yoktur!**
- Topolojik sıralamada herhangi bir  $(u,v)$  kenarında,  $u$ , sıralamada  $v$ 'den öncedir.
- Aşağıdaki algoritma bir DAG için topolojik sıralamayı yapar

TOPOLOGICAL-SORT( $G$ )

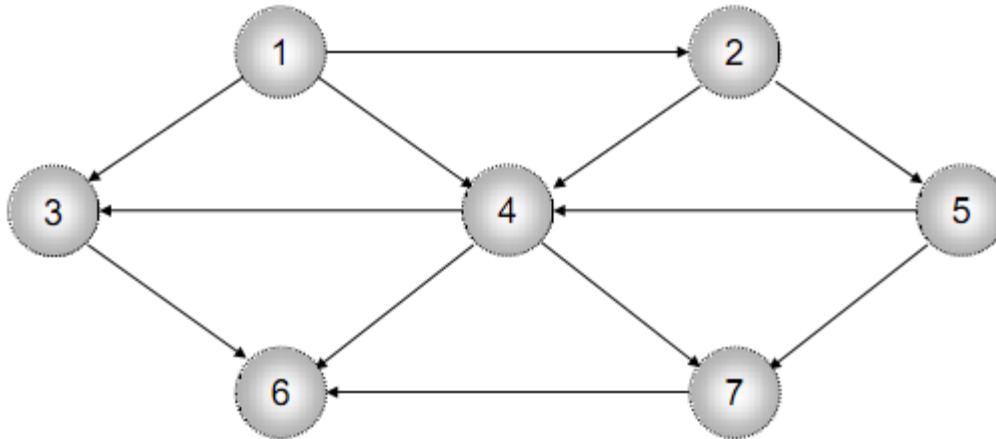
- 1 call  $\text{DFS}(G)$  to compute finishing times  $v.f$  for each vertex  $v$
- 2 as each vertex is finished, insert it onto the front of a linked list
- 3 **return** the linked list of vertices

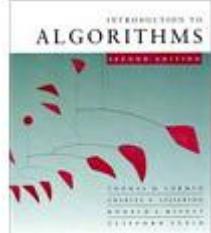
- Toplam sıralama bağlı dizilerle ile oluşturulur.



# Topolojik Sıralama

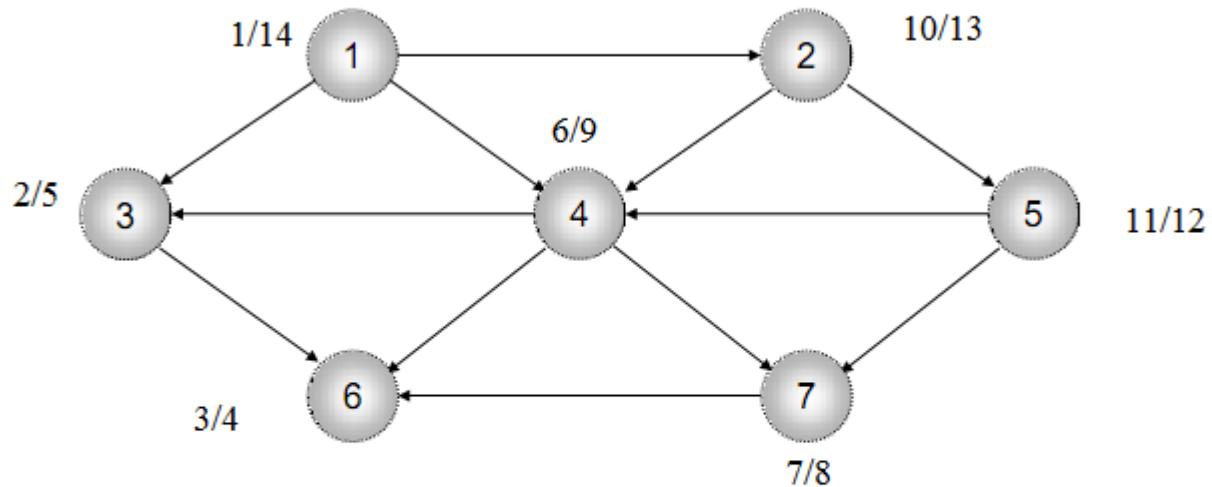
DFS- Topological Ordering





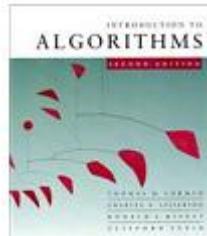
# Topolojik Sıralama

## DFS- Topological Ordering



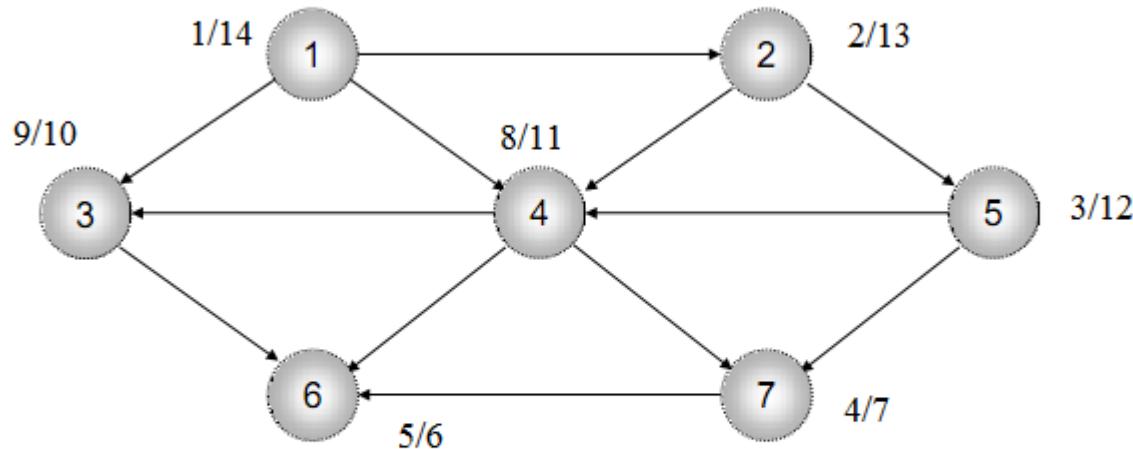
|           |            |            |          |          |          |          |
|-----------|------------|------------|----------|----------|----------|----------|
| 1<br>1/14 | 2<br>10/13 | 5<br>11/12 | 4<br>6/9 | 7<br>7/8 | 3<br>2/5 | 6<br>3/4 |
|-----------|------------|------------|----------|----------|----------|----------|

Topolojik sıralama için Sol düğüme göre DFS dolaşma



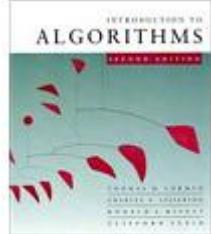
# Topolojik Sıralama

DFS- Topological Ordering



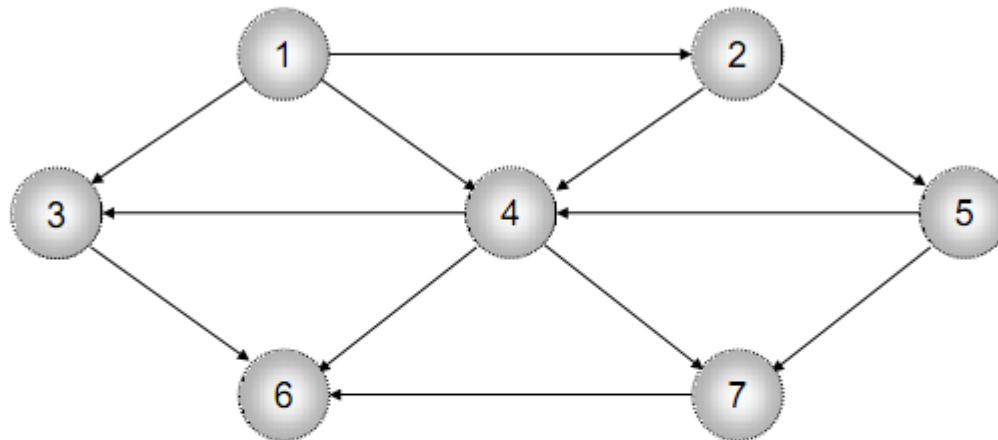
|      |      |      |      |      |     |     |
|------|------|------|------|------|-----|-----|
| 1    | 2    | 5    | 4    | 3    | 7   | 6   |
| 1/14 | 2/13 | 3/12 | 8/11 | 9/10 | 4/7 | 5/6 |

Topolojik sıralama için Sağ düğüme göre DFS dolaşma

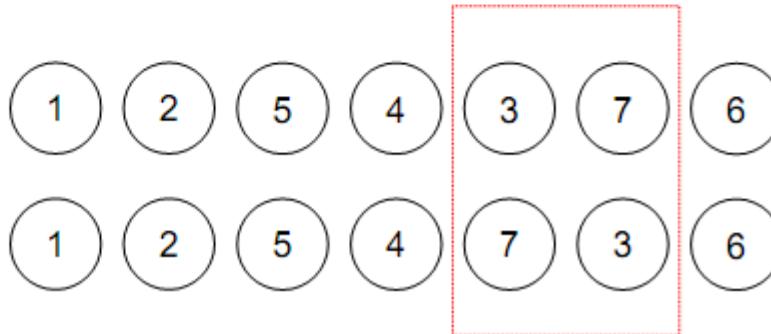


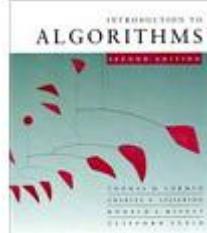
# Topolojik Sıralama Çalışma zamanı

## Topological Ordering



*A topological ordering*





# Topolojik Sıralama Çalışma zamanı

- Çalışma Zamanı:
- DFS:  $O(V+E)$
- Her bir  $|V|$  düğümünün bağlı dizi başına eklenmesi  $O(1)$  süresini alır (her ekleme için)
- Toplam çalışma zamanı:  $O(|V^2|)$  veya  $O(|V|+|E|)$  olur.

```

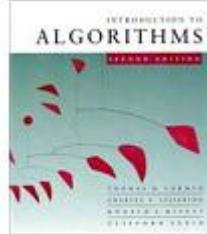
void Graph::topsort()
{
 Queue q(NUM_VERTICES);
 int counter = 0; //topological order of a vertex:1,2,3,...,NUM_VERTICES
 Vertex v, w;

 q.makeEmpty();
 for each vertex v
 if (v.indegree == 0)
 q.enqueue(v);
 while (! q.isEmpty())
 {
 v = q.dequeue();
 counter++;

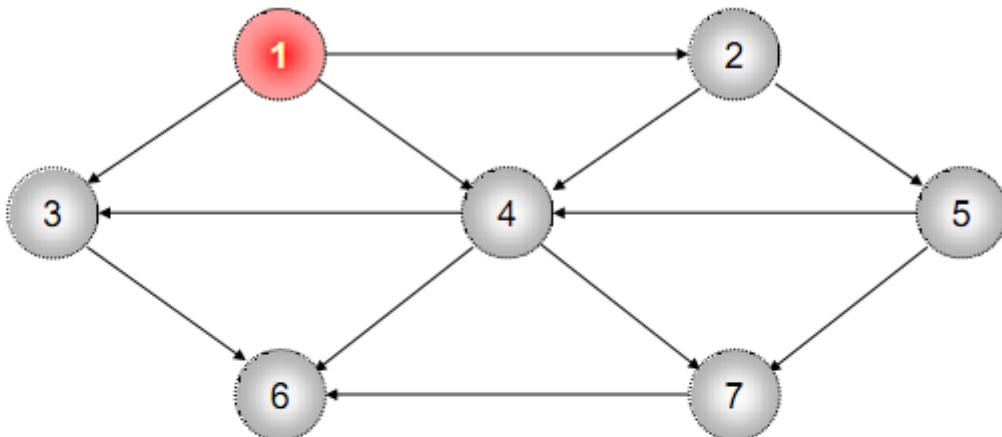
 for each w adjacent to v
 if (--w.indegree == 0)
 q.enqueue(w);
 }
 if (counter != NUM_VERTICES)
 throw CycleFound();
}

```

Pseudocode  
of a  
Topological Sort  
Algorithm

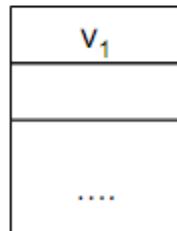


# Topolojik Sıralama

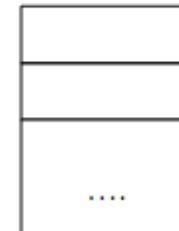


| Indegrees |   |
|-----------|---|
| $v_1$     | 0 |
| $v_2$     | 1 |
| $v_3$     | 2 |
| $v_4$     | 3 |
| $v_5$     | 1 |
| $v_6$     | 3 |
| $v_7$     | 2 |

After  
Enqueue

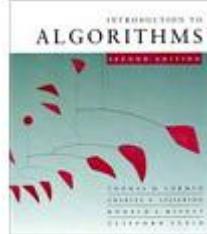


After  
Dequeue

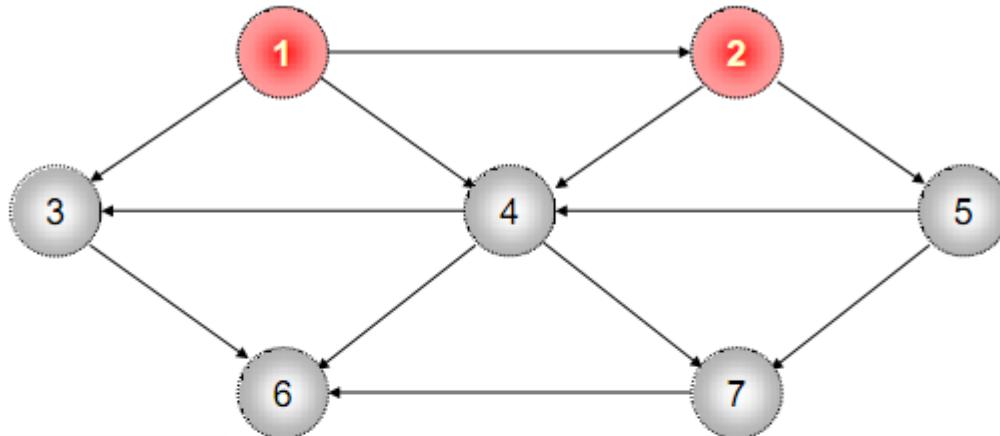


Print

$v_1$



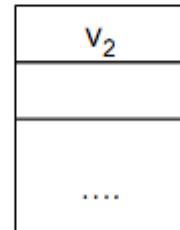
# Topolojik Sıralama



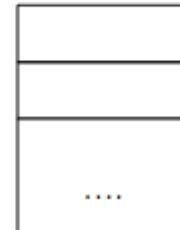
*Updated Indegrees*

|       |   |
|-------|---|
| $v_1$ | 0 |
| $v_2$ | 0 |
| $v_3$ | 1 |
| $v_4$ | 2 |
| $v_5$ | 1 |
| $v_6$ | 3 |
| $v_7$ | 2 |

After  
Enqueue

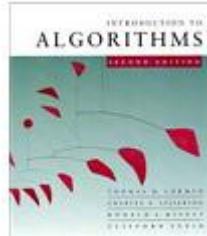


After  
Dequeue

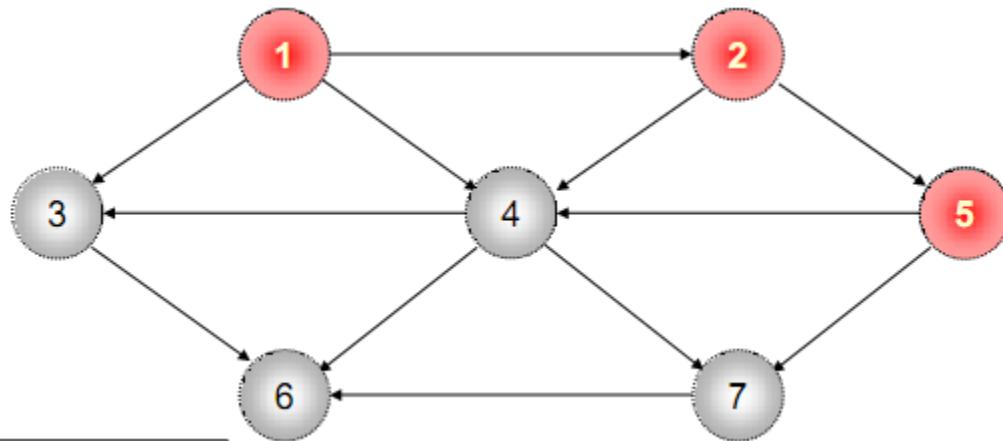


Print

$v_2$



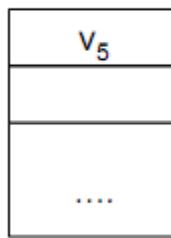
# Topolojik Sıralama



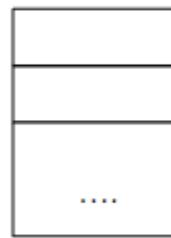
*Updated Indegrees*

|       |   |
|-------|---|
| $v_1$ | 0 |
| $v_2$ | 0 |
| $v_3$ | 1 |
| $v_4$ | 1 |
| $v_5$ | 0 |
| $v_6$ | 3 |
| $v_7$ | 2 |

After  
Enqueue

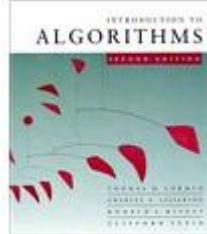


After  
Dequeue

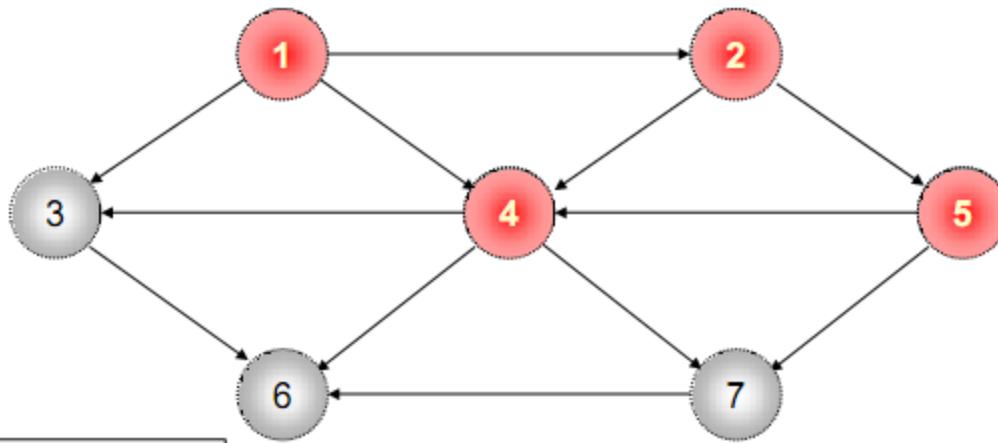


Print

$v_5$



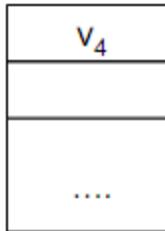
# Topolojik Sıralama



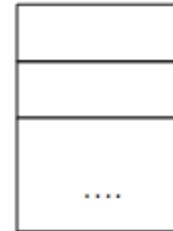
*Updated Indegrees*

|       |   |
|-------|---|
| $v_1$ | 0 |
| $v_2$ | 0 |
| $v_3$ | 1 |
| $v_4$ | 0 |
| $v_5$ | 0 |
| $v_6$ | 3 |
| $v_7$ | 1 |

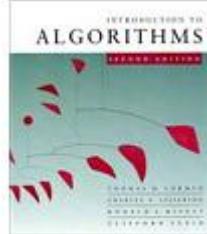
After  
Enqueue



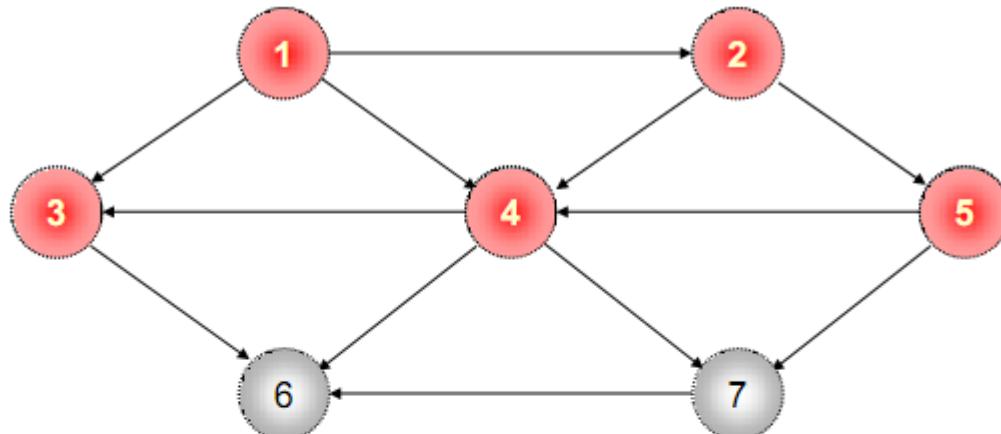
After  
Dequeue



Print  
 $v_4$



# Topolojik Sıralama



*Updated Indegrees*

|       |   |
|-------|---|
| $v_1$ | 0 |
| $v_2$ | 0 |
| $v_3$ | 0 |
| $v_4$ | 0 |
| $v_5$ | 0 |
| $v_6$ | 2 |
| $v_7$ | 0 |

After  
Enqueue

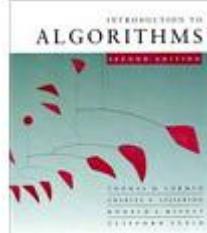
|       |
|-------|
| $v_3$ |
| $v_7$ |
| ....  |

After  
Dequeue

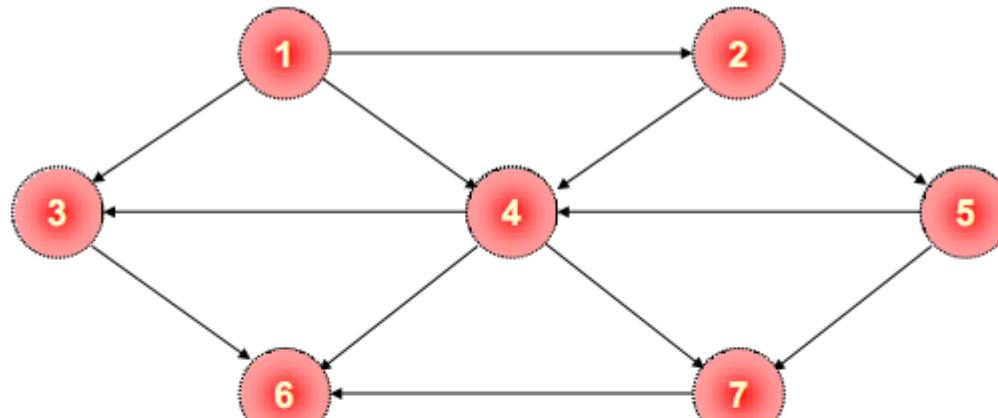
|       |
|-------|
| $v_7$ |
| ....  |
| ....  |

Print

$v_3$



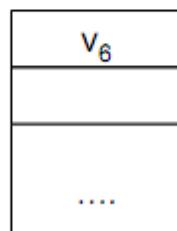
# Topolojik Sıralama



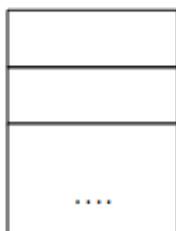
*Updated Indegrees*

|       |   |
|-------|---|
| $v_1$ | 0 |
| $v_2$ | 0 |
| $v_3$ | 0 |
| $v_4$ | 0 |
| $v_5$ | 0 |
| $v_6$ | 0 |
| $v_7$ | 0 |

After  
Enqueue



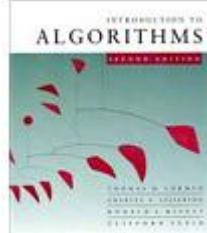
After  
Dequeue



Print

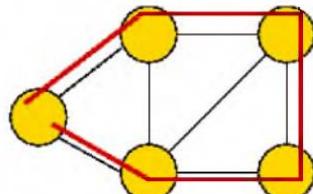
$v_6$

Finished! Result: 1,2,5,4,3,7,6

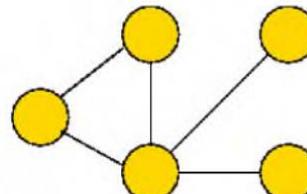


# Hamiltonian ve Euler Cycle

- **Hamiltonian döngüsü:** Bir  $G$  grafında, başlangıç ve bitiş düğümleri hariç her düğüm bir kez bulunduğu yoldur.

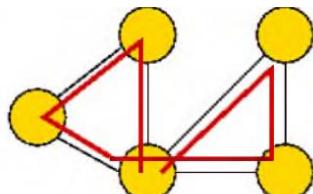


Hamiltonian cycle var

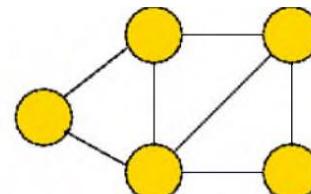


Hamiltonian cycle yok

- **Euler döngüsü:** Bir  $G$  grafında, bir düğümünden başlayıp yine kendisinde bitecek şekilde oluşturulan bir yoldur. Bu yolda bütün düğümler en az bir kez bulunur ancak her kenar mutlaka bir kez bulunur.



Euler cycle var



Euler cycle yok

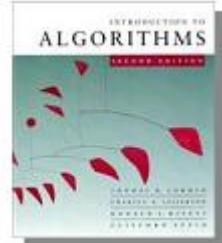
**12.Hafta**  
**Minimum kapsayan**  
**ağaçlar**  
**Minimum spanning trees**  
**(MST)**

# **12.Hafta**

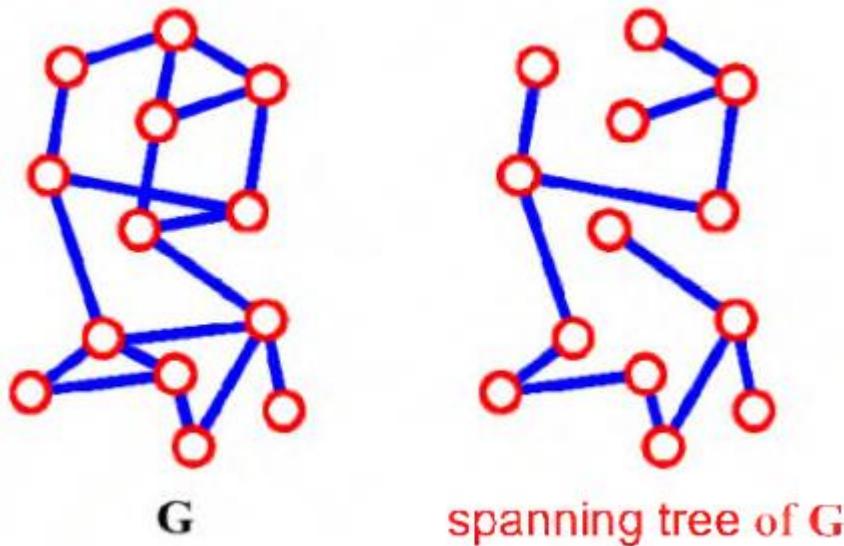
## **Minimum kapsayan ağaçlar**

## **Minimum spanning trees (MST)**

# Kapsayan ağaç Spanning Tree (ST)

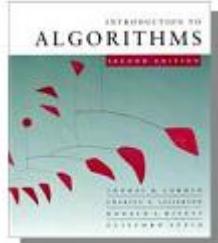


- Bir Kapsayan Ağaç (ST);  $G$ , grafındaki bir alt graftır ve aşağıdaki özelliklere sahiptir.
  - $G$  grafındaki tüm düğümleri içerir
  - Bir ağaç yapısı oluşturur.



# Minimum kapsayan ağaçlar

## Minimum spanning trees (MST)

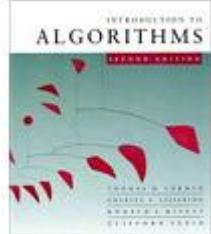


**Girdi:**  $w : E \rightarrow \mathbb{R}$  ağırlık fonksiyonlu,  $G = (V, E)$  bağlantılı, yönlendirilmemiş grafik.

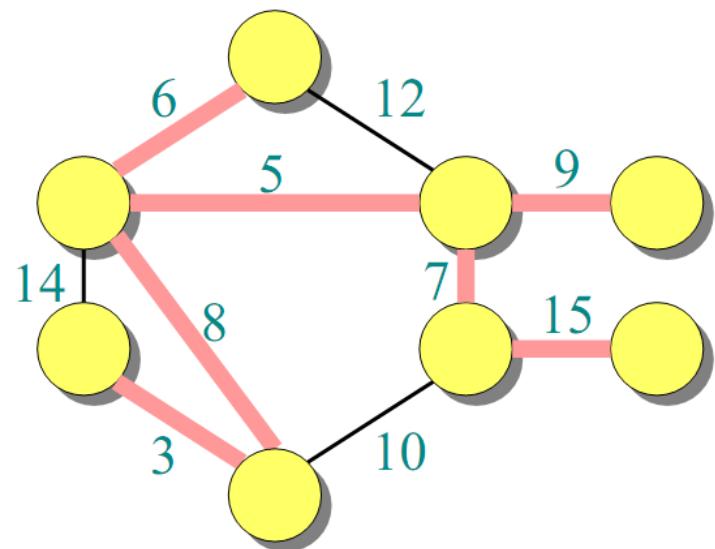
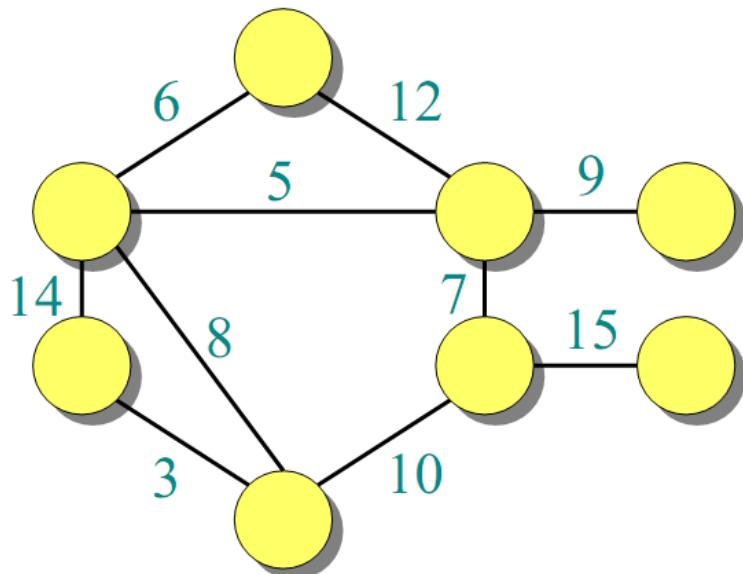
- Basitlik adına, farzedin ki tüm kenar ağırlıkları farklı olsun.

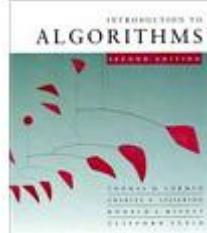
**Çıktı:**  $T$  kapsayan ağaç--en az ağırlıkla tüm köşeleri birleştiren bir ağaç:

$$w(T) = \sum_{(u,v) \in T} w(u,v).$$



# MST (minimum kapsayan ağaç) örneği

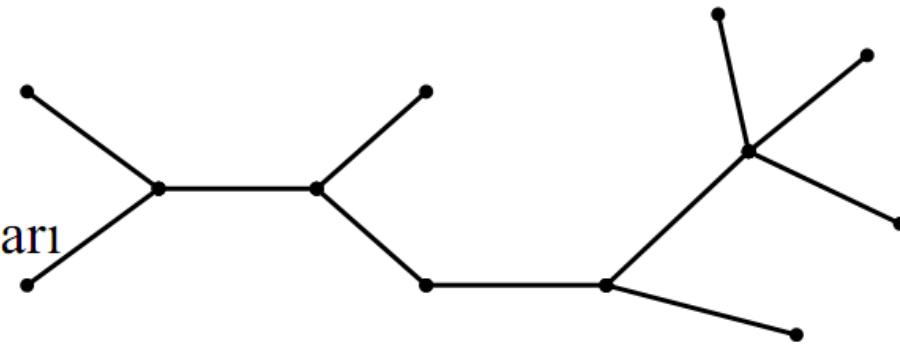




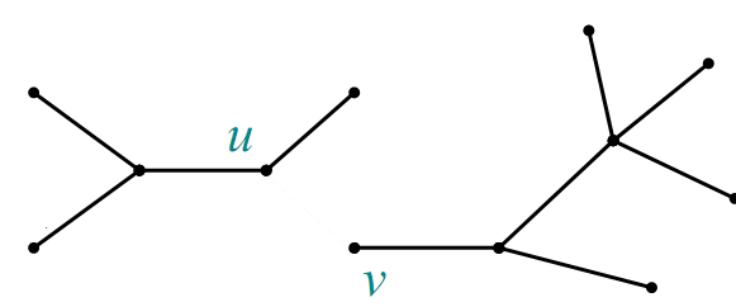
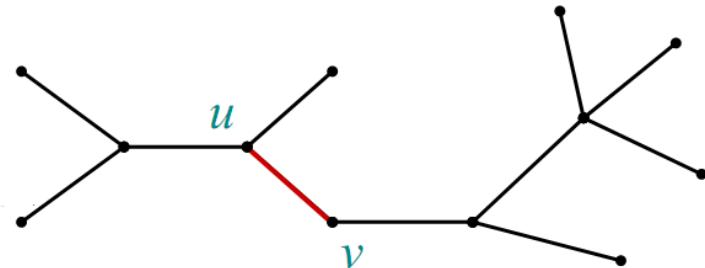
# Optimal altyapı

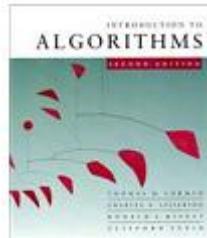
MST  $T$ :

( $G'$  nin diğer kenarları gösterilmemiştir.)



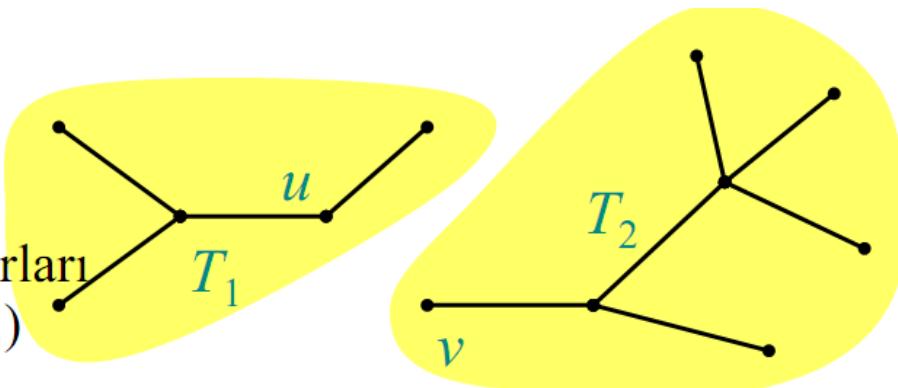
Herhangi bir  $(u, v) \in T$  kenarını kaldır.





# Optimal altyapı

MST  $T$ :  
 $(G'$  nin diğer kenarları  
gösterilmemiştir.)

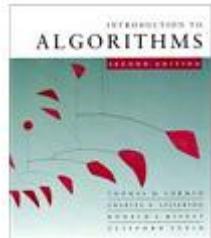


Herhangi bir  $(u, v) \in T$  kenarını kaldırın. Bu durumda  $T$ ,  $T_1$  and  $T_2$  olarak iki altağaca bölüntülenir.

**Teorem.** Altağac  $T_1$ ,  $G_1 = (V_1, E_1)$  'in bir MST'sidir ve  $T_1$ 'in köşeleri tarafından oluşturulan  $G'$  nin altgrafıgidir:

$$\begin{aligned} V_1 &= T_1 \text{' in köşeleri} \\ E_1 &= \{ (x, y) \in E : x, y \in V_1 \}. \end{aligned}$$

$T_2$  için de bu böyledir.



## Optimal altyapının kanıtı

*Kanıt.* Kes ve yapıştır

$$w(T) = w(u, v) + w(T_1) + w(T_2).$$

Eğer  $T_1'$   $G_1$  için  $T_1$  den daha az ağırlıklı bir kapsayan ağaçsa, bu durumda  $T' = \{(u, v)\} \cup T_1' \cup T_2$   $G$  için  $T'$  den daha az ağırlıklı bir kapsayan ağaç olur.

Aynı zamanda çakışan altproblemlerimiz de var mı?

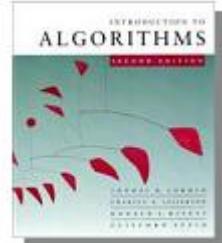
- Evet var.

Harika, o zaman dinamik programlama çalışabilir!

- Evet, fakat MST daha da verimli bir algoritmaya yol açan bir başka kuvvetli özelliğini sergiler.

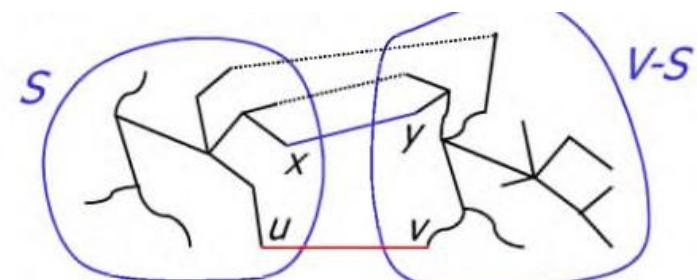
# Greedy Yaklaşımı/Yöntemi

- Dolaşma yapılırken bir sonraki düğümü belirlemek için kullanılan bir karar verme/seçme yöntemidir.
- O andaki seçenekler içerisinde en iyi olarak gözükeni seçer.
- Bölgesel/yerel değerlendirmeler yapar.
- Yerel optimum daima global optimum anlamına gelmez dolayısıyla en iyi sonuca götürmeyebilir.
- Fakat bazı durumlarda en iyi sonuca götürür. (MST, en kısa yol alg. , Huffman coding)

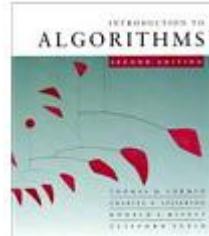


# Greedy (Aç gözlü) Seçim

- MST'nin kenarları için  $V-1$  tane seçim yapılması gerekmektedir.
- **Greedy seçim:** Lokal olarak optimal seçim (greedy) global olarak optimal çözümü oluşturur.
- **Teorem**
  - $(u,v)$  düşük ağırlıklı bir kenar ancak  $(u,v) \notin \text{MST}$  ise
  - MST içinde  $u$ 'dan  $v$ 'ye bir yol aranır. MST içinde  $(x,y)$  gibi bir yol bulunursa,  $(x,y)$  yerine  $(u,v)$  alınır.
  - Bu işlem MST yi geliştirir.

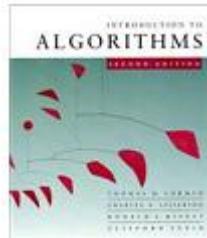


# Açgözlü algoritmalar için Kalite işareteti



**Açgözlü-seçim özelliği**  
*Yerel olarak en uygun seçim genel olarak da en uygundur.*

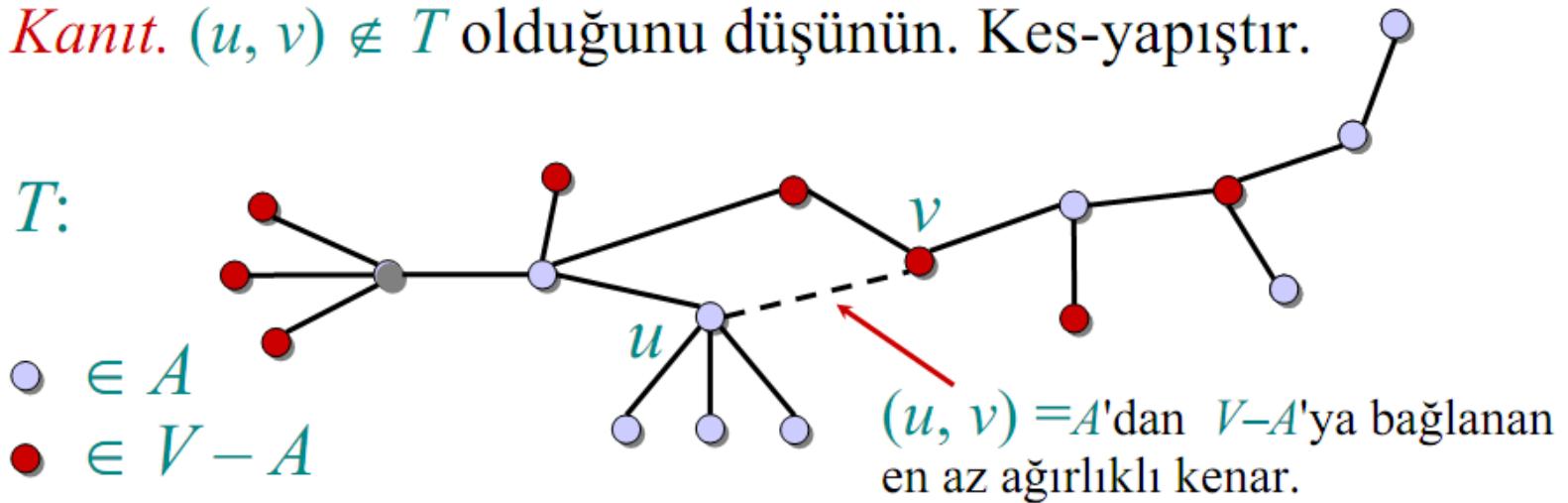
**Teorem.**  $T$ ,  $G = (V, E)$ 'nin MST'si olsun ve  $A \subseteq V$  olsun.  $(u, v) \in E$  'nin  $A$  yi  $V - A$ 'ya bağlayan en az ağırlıklı kenar olduğunu farzedin. O zaman  $(u, v) \in T$  olur.

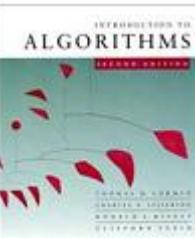


## Teorem' in kanıtı

*Kanıt.*  $(u, v) \notin T$  olduğunu düşünün. Kes-yapıştır.

$T$ :

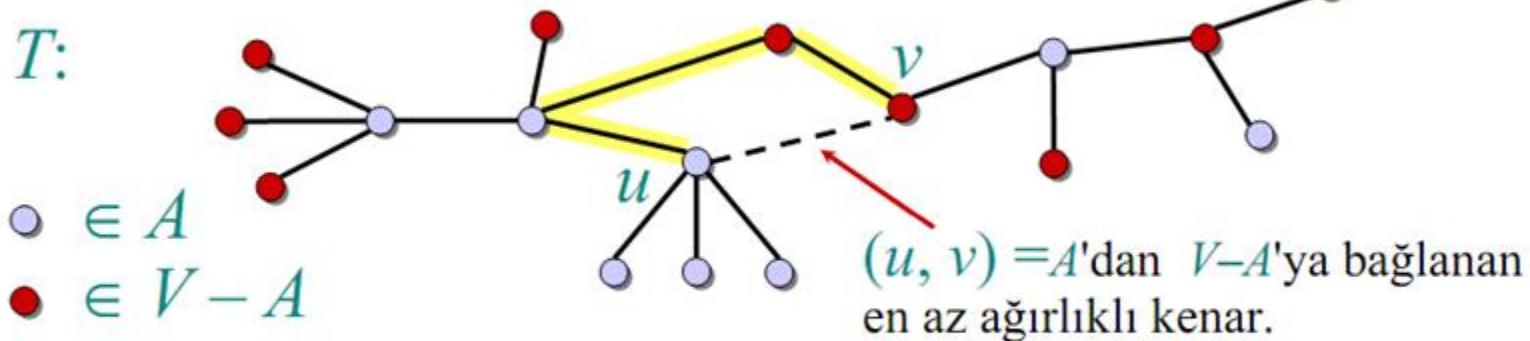




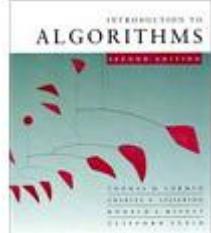
## Teorem' in kanıtı

*Kanıt.*  $(u, v) \notin T$  olduğunu düşünün. Kes-yapıştır.

$T$ :



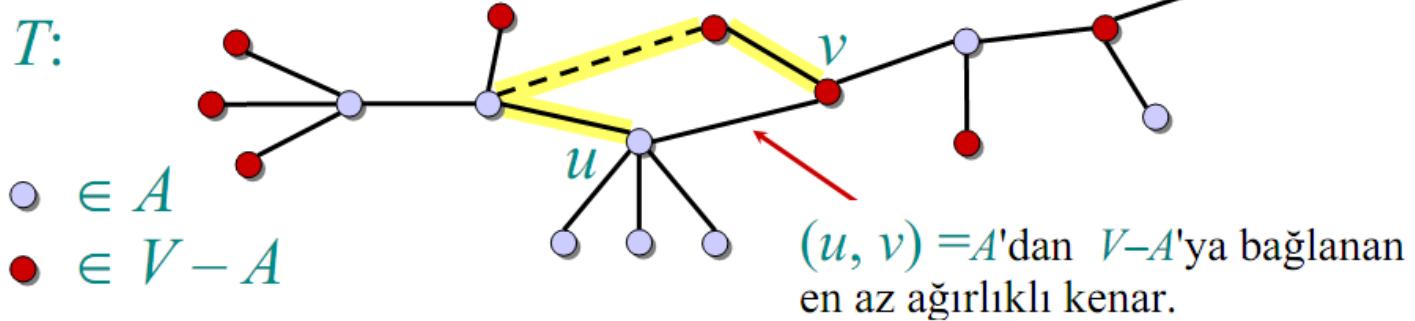
$T'$  de  $u$ 'dan  $v$ 'ye benzeri olmayan basit yolu düşünün.



## Teorem' in kanıtı

*Kanıt.*  $(u, v) \notin T$  olduğunu düşünün. Kes-yapıştır.

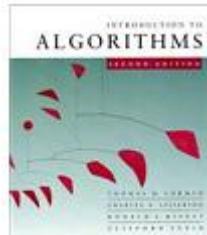
$T$ :



$T'$  de  $u$ 'dan  $v$ 'ye benzeri olmayan basit yolu düşünün.

$(u, v)$ 'yi, bu yolda  $A'$  daki bir köşeyi

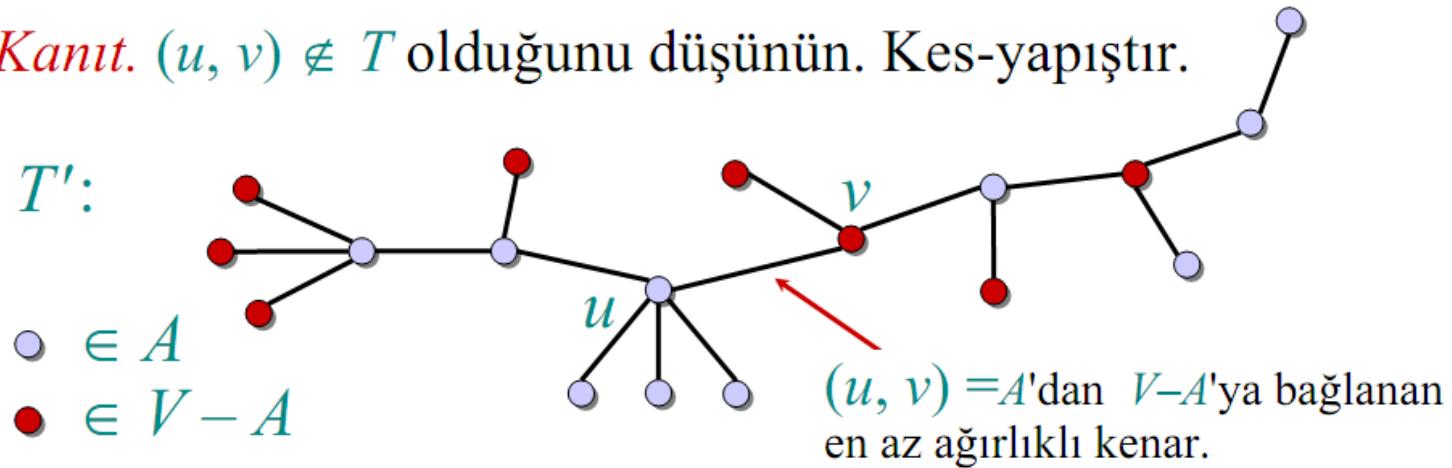
$V - A'$  daki bir köşeye bağlayan ilk kenarla değiştirin.



## Teorem' in kanıtı

*Kanıt.*  $(u, v) \notin T$  olduğunu düşünün. Kes-yapıştır.

$T'$ :



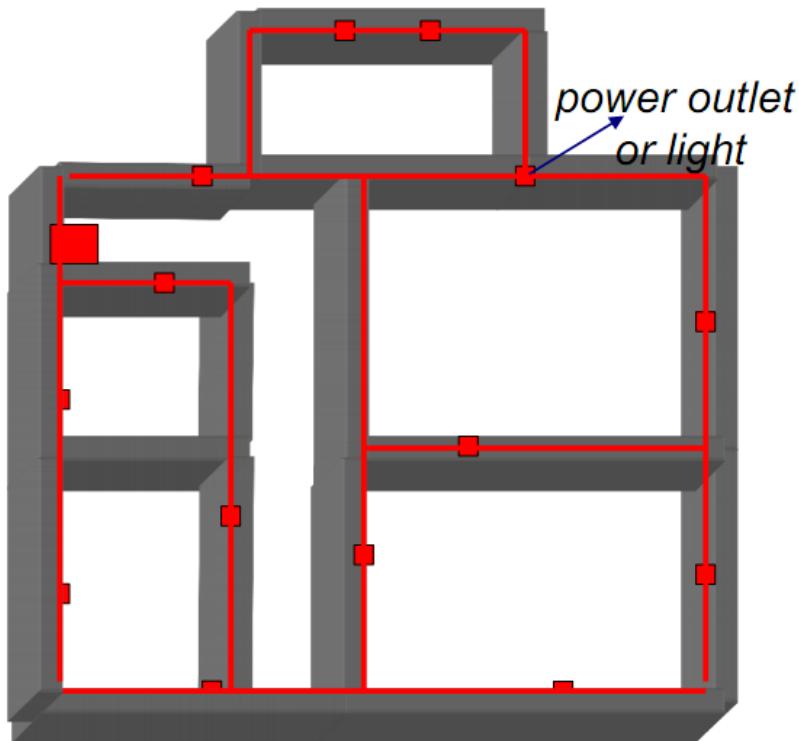
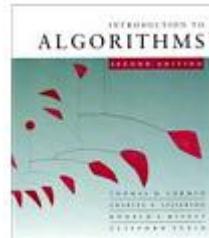
$T'$  de  $u$ 'dan  $v$ 'ye benzeri olmayan basit yolu düşünün.

$(u, v)$ 'yi, bu yolda  $A$ ' daki bir köşeyi

$V - A$ ' daki bir köşeye bağlayan ilk kenarla değiştirin.

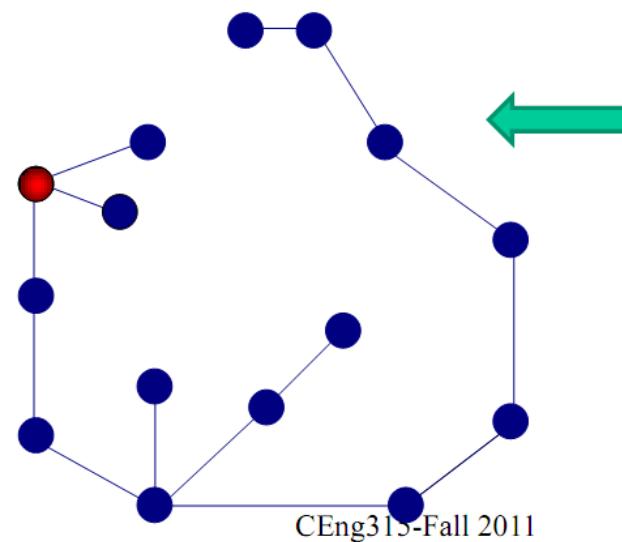
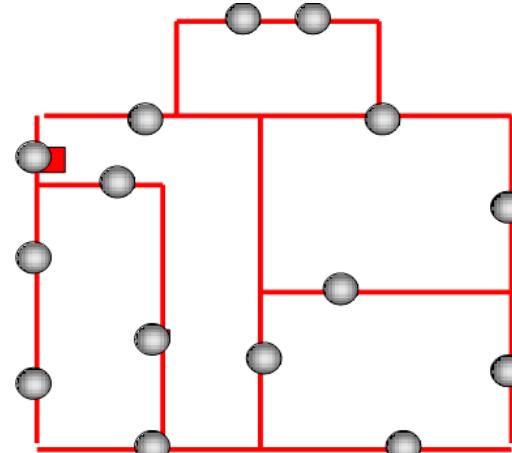
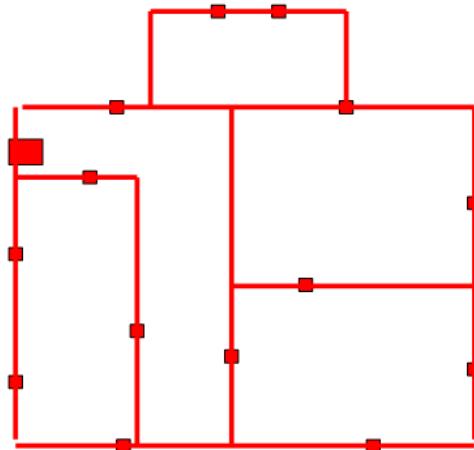
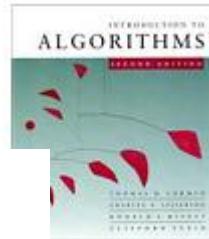
$T$ 'den daha az ağırlıklı bir kapsayan ağaç oluşur. □

# MST uygulaması için örnek



- Mininum kablo kullanarak bir evin elektriğini döseme

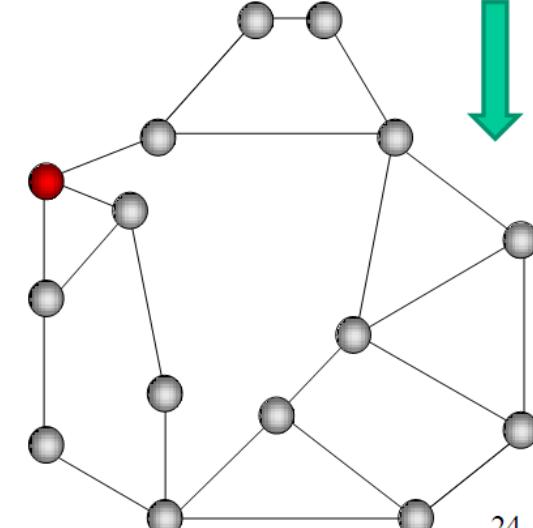
# MST uygulaması için örnek



o

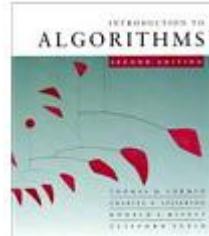
A.Yazici

CEng315-Fall 2011



24

# MST Algoritmaları



- En küçük yol ağacını belirlemek için birçok algoritma geliştirilmiştir.
  - **Prim'in Algoritması:** En az maliyetli kenardan başlayıp onun uçlarından en az maliyetle genişleyecek kenarın seçilmesine dayanır. Bir tane ağaç oluşur.
  - **Kruskal'ın Algoritması:** Daha az maliyetli kenarları tek tek değerlendирerek yol ağacını bulmaya çalışır. Ara işlemler birden çok ağaç oluşturabilir.
  - **Sollin'in Algoritması:** Doğrudan paralel programlamaya yatkındır. Aynı anda birden çok ağaçla başlanır ve ilerleyen adımlarda ağaçlar birleşerek tek bir yol ağacına dönüşür.

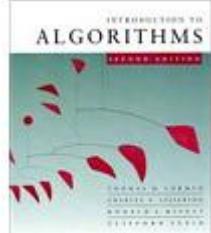
# Prim'in Algoritması

- Düğüm tabanlı bir algoritmadır. Bir T ağacını her adımda bir düğüm ekleyerek büyütür.
- Bir kök düğüm ile başlar ve bütün düğümleri içine alıncaya kadar ağaç büyütür.
- **Adım-1:** Başlangıçta, herhangi bir noktayı ağaç oluşturmaya başlamak için seç.
- **Adım-2:** Oluşturulan ağaç'a eklemek için, şu ana kadar oluşturulmuş **ağaç üzerinden erişilebilen** ve daha önceden ağaç'a katılmamış olan **en küçük ağırlıklı kenarı** seç.
- **Adım-3:** Eğer bu kenarın ağaç'a katılması, bir **çember oluşmasına sebep olmuyorsa**, ağaç'a ekle.
- **Adım-4:** Ağaçtaki kenar sayısı **(N-1)**'e ulaşana kadar ikinci adıma geri dön.

$A = \{(v, \pi[v]) : v \in V - \{r\}\}$ .

MST-PRIM( $G, w, r$ )

- 1 **for** each  $u \in V[G]$
- 2     **do**  $key[u] \leftarrow \infty$
- 3      $\pi[u] \leftarrow \text{NIL}$
- 4      $key[r] \leftarrow 0$
- 5      $Q \leftarrow V[G]$
- 6     **while**  $Q \neq \emptyset$
- 7         **do**  $u \leftarrow \text{EXTRACT-MIN}(Q)$
- 8         **for** each  $v \in \text{Adj}[u]$
- 9             **do if**  $v \in Q$  and  $w(u, v) < key[v]$
- 10                 **then**  $\pi[v] \leftarrow u$
- 11                  $key[v] \leftarrow w(u, v)$



## Prim'in Algoritması

**Fikir:**  $V - A'$ 'yı bir  $Q$  öncelikli sırası olarak koruyun.  
 $Q'$  daki her köşeyi,  $A'$  daki bir köşeye bağlayan en az ağırlıklı kenarın ağırlığıyla KEYleyin. (anahtarlayın)

$Q \leftarrow V$

$key[v] \leftarrow \infty$  tüm  $v \in V$ 'ler için

$key[s] \leftarrow 0$  rastgele  $s \in V$ 'için

(-iken) **while**  $Q \neq \emptyset$

(yap) **do**  $u \leftarrow \text{EXTRACT-MIN}(\text{en küçükü çıkar})(Q)$

her  $v \in Adj[u]$  için

(yap-eğer) **do if**  $v \in Q$  ve  $w(u, v) < key[v]$

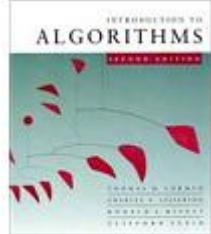
(sonra) **then**  $key[v] \leftarrow w(u, v)$

$\pi[v] \leftarrow u$

► DECREASE-KEY

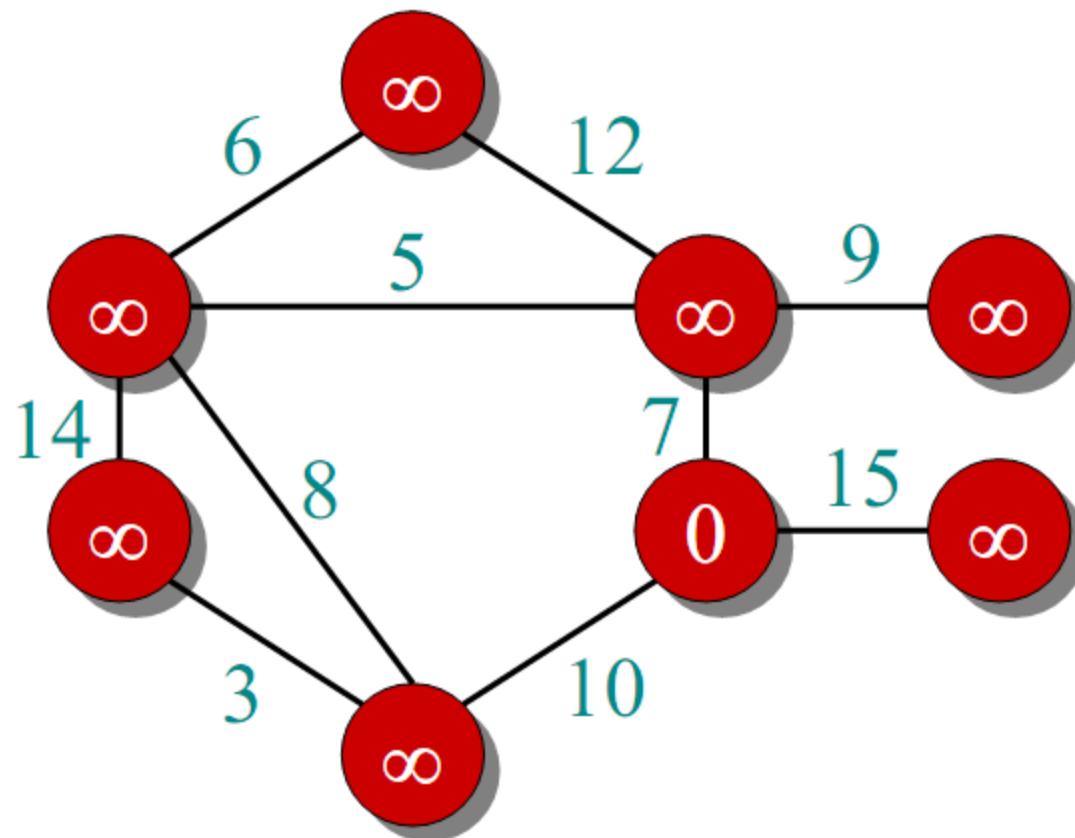
(anahtarı küçült)

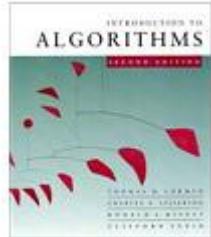
Sonunda,  $\{(v, \pi[v])\}$ , MST' yi biçimlendirir.



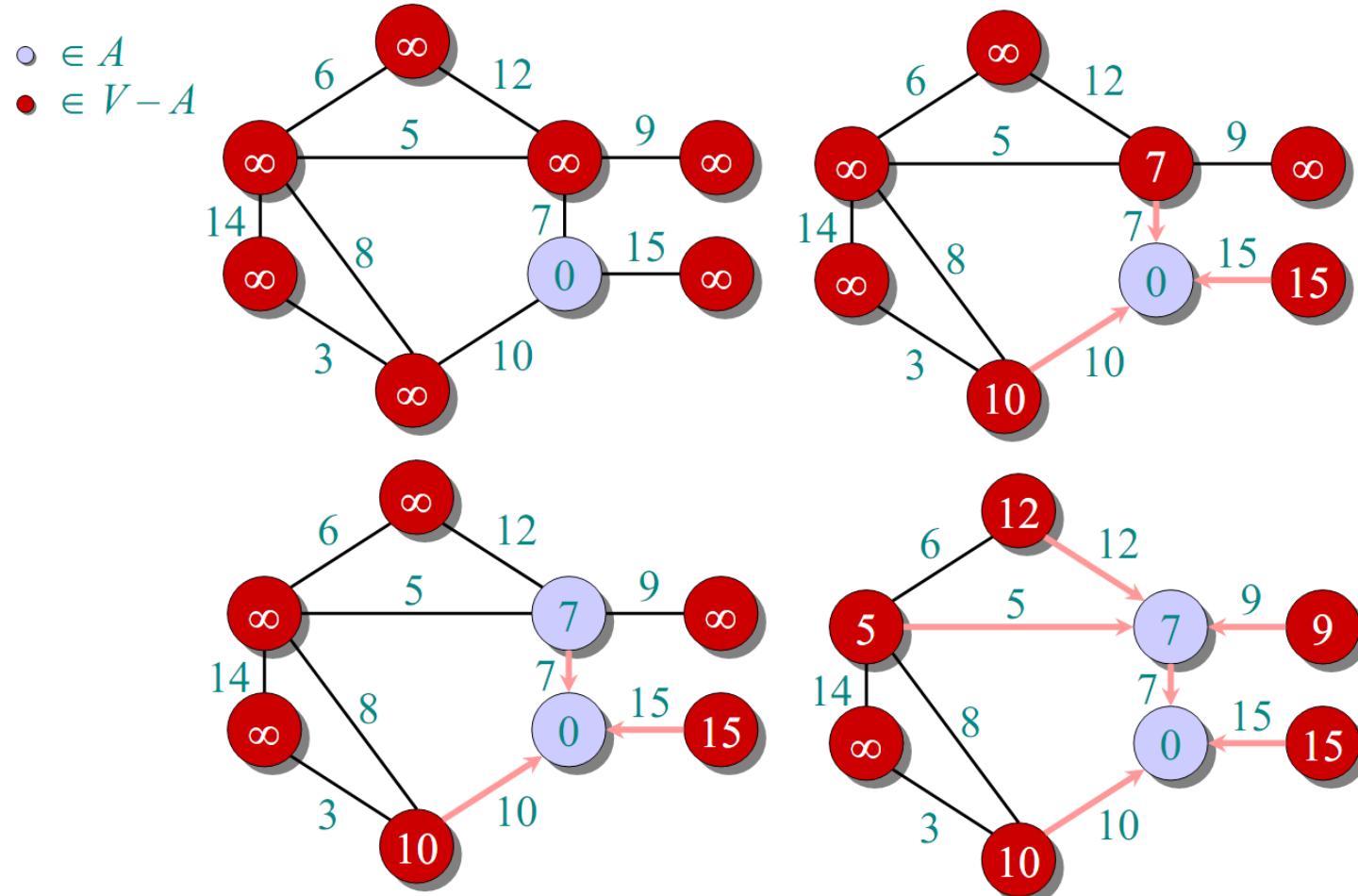
## Prim'in algoritmasına örnek

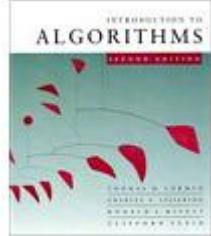
- $\in A$
- $\in V - A$



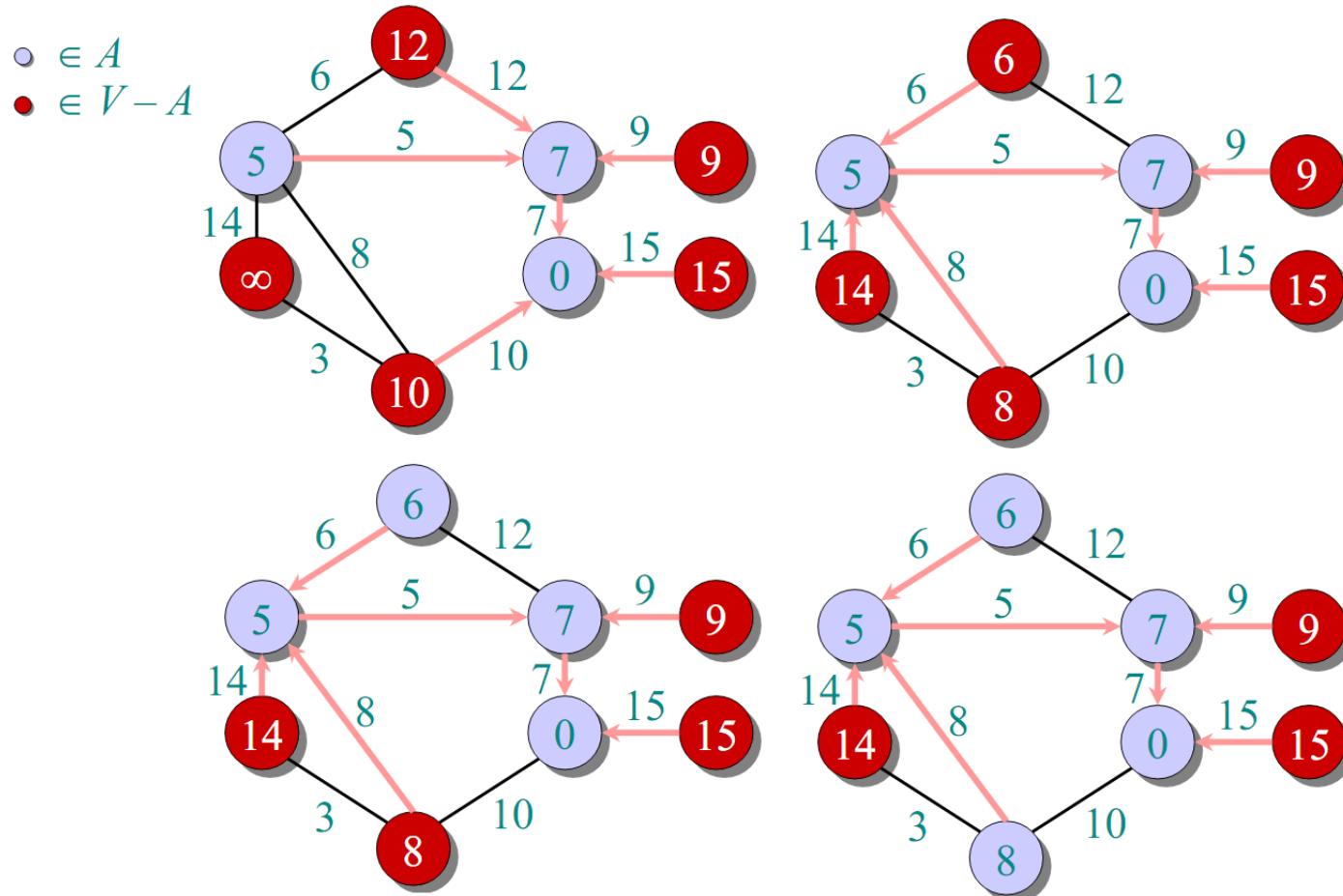


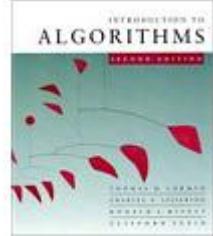
## Prim'in algoritmasına örnek



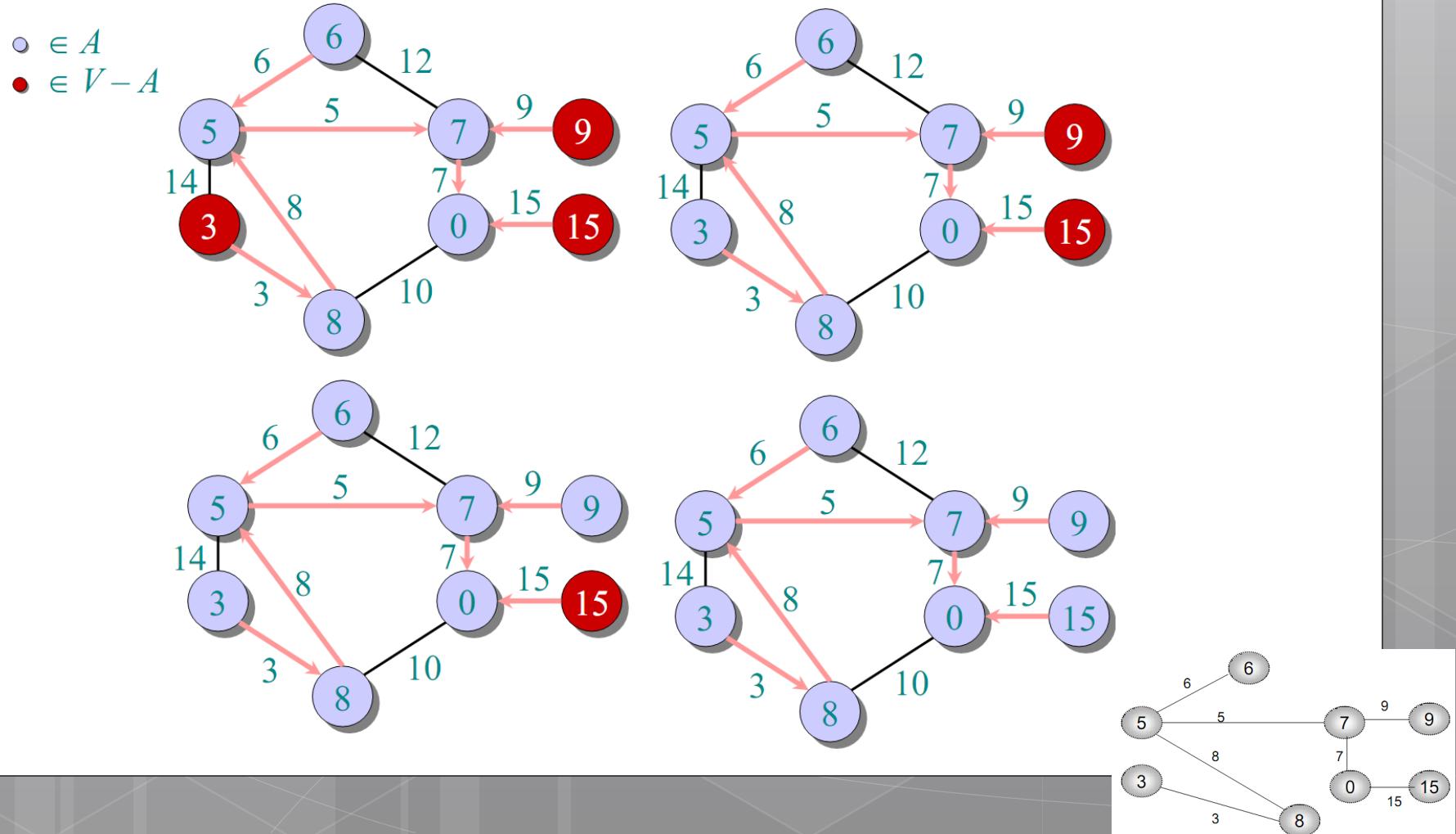


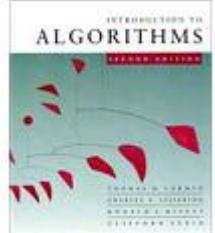
## Prim'in algoritmasına örnek





## Prim'in algoritmasına örnek



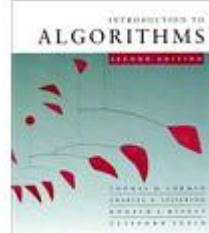


## Prim' in çözümlemesi (analizi)

$\Theta(V)$  toplam  $\left\{ \begin{array}{l} Q \leftarrow V \\ \text{key}[v] \leftarrow \infty \text{ tüm } v \in V \text{ ler için.} \\ \text{key}[s] \leftarrow 0 \text{ rastgele } s \in V \text{ ler için.} \end{array} \right.$   
 $|V|$  kere  $\left\{ \begin{array}{l} (-\text{iken}) \textbf{while } Q \neq \emptyset \\ \quad (\text{yap}) \textbf{do } u \leftarrow \text{En az } (Q) \text{ yu çıkar.} \\ \quad \quad \text{her } v \in \text{Adj}[u] \text{ için} \\ \quad \quad \quad \textbf{do if } v \in Q \text{ ve } w(u, v) < \text{key}[v] \\ \quad \quad \quad \quad (\text{yap-eğer}) \textbf{then } \text{key}[v] \leftarrow w(u, v) \\ \quad \quad \quad \quad (\text{sonra}) \quad \pi[v] \leftarrow u \end{array} \right.$

Tokalaşma önkuramı  $\Rightarrow \Theta(E)$  varsayılan DECREASE-KEY'ler.  
 (Anahtarı küçült)

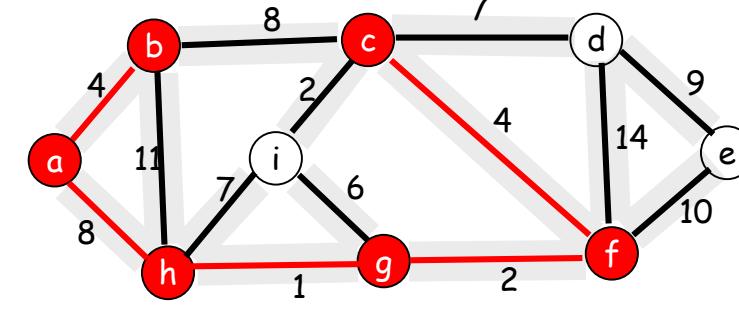
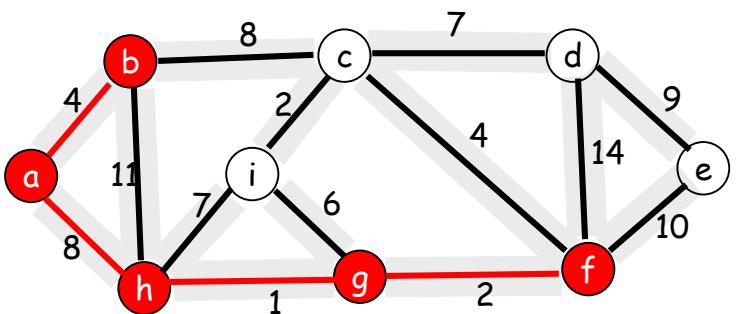
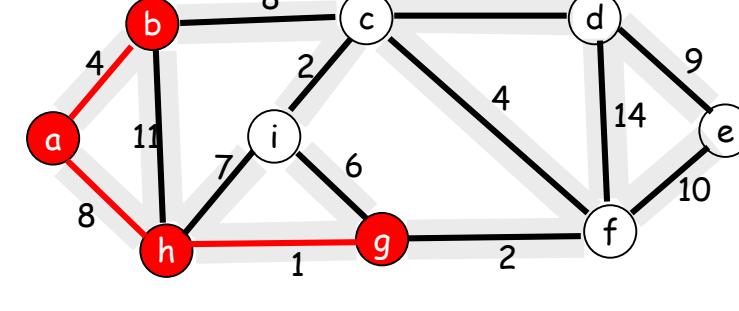
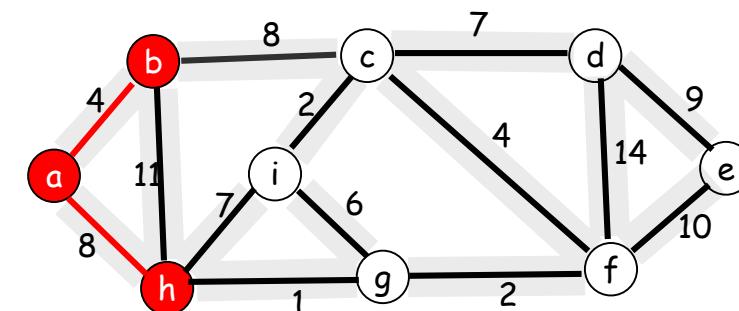
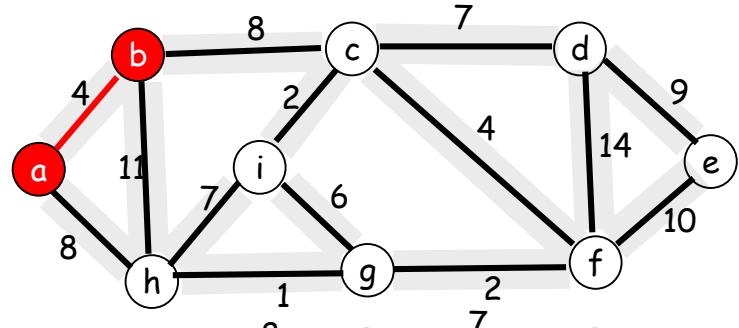
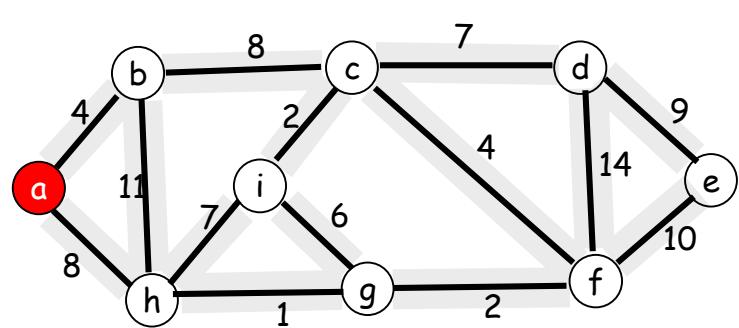
Time(süre) =  $\Theta(V) \cdot T_{\text{EXTRACT-MIN}} + \Theta(E) \cdot T_{\text{DECREASE-KEY}}$   
 (en küçükü çıkar)



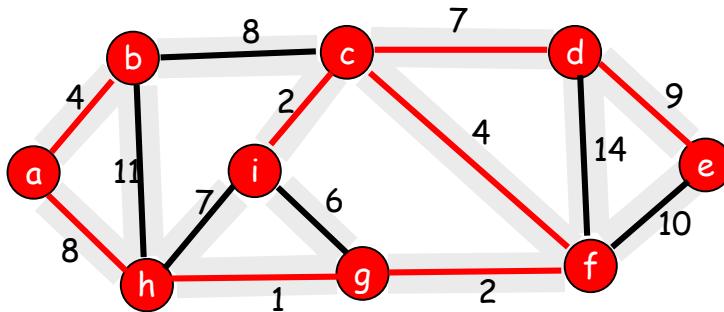
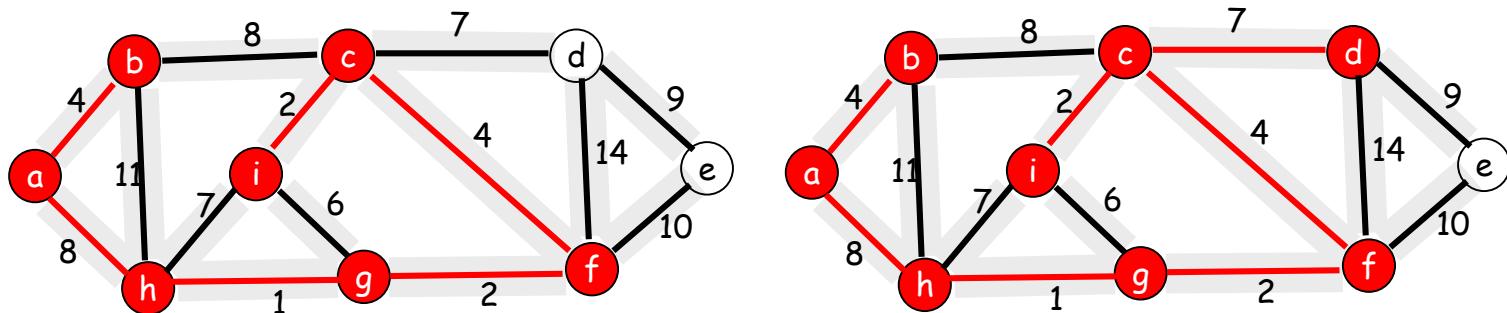
# Prim' in çözümlemesi (devamı)

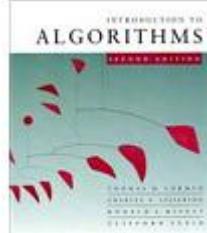
|                   | $Q$                 | $T_{\text{EXTRACT-MIN}}$          | $T_{\text{DECREASE-KEY}}$     | Toplam                                          |
|-------------------|---------------------|-----------------------------------|-------------------------------|-------------------------------------------------|
| array             | dizilim             | $O(V)$                            | $O(1)$                        | $O(V^2)$                                        |
| binary<br>heap    | ikili<br>yığın      | $O(\lg V)$                        | $O(\lg V)$                    | $O(E \lg V)$                                    |
| Fibonacci<br>heap | Fibonacci<br>yığını | $O(\lg V)$<br>amortize<br>edilmiş | $O(1)$<br>amortize<br>edilmiş | $O(E + V \lg V)$<br>worst case<br>en kötü durum |

# Prim'in Algoritması: Örnek



# Prim'in Algoritması: Örnek

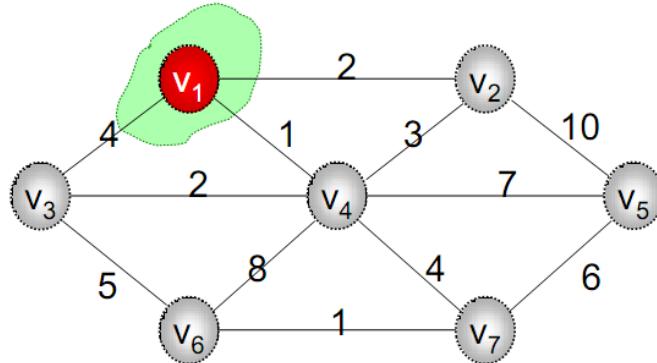




# Prim'in algoritmasına örnek-Sınav

## Prim's Algorithm: Example

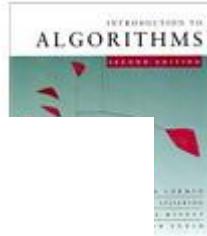
- Start with vertex  $v_1$ . It is the initial current tree which we will grow to an MST



|                |   |
|----------------|---|
| V <sub>1</sub> | 0 |
|----------------|---|

Content of the priority queue

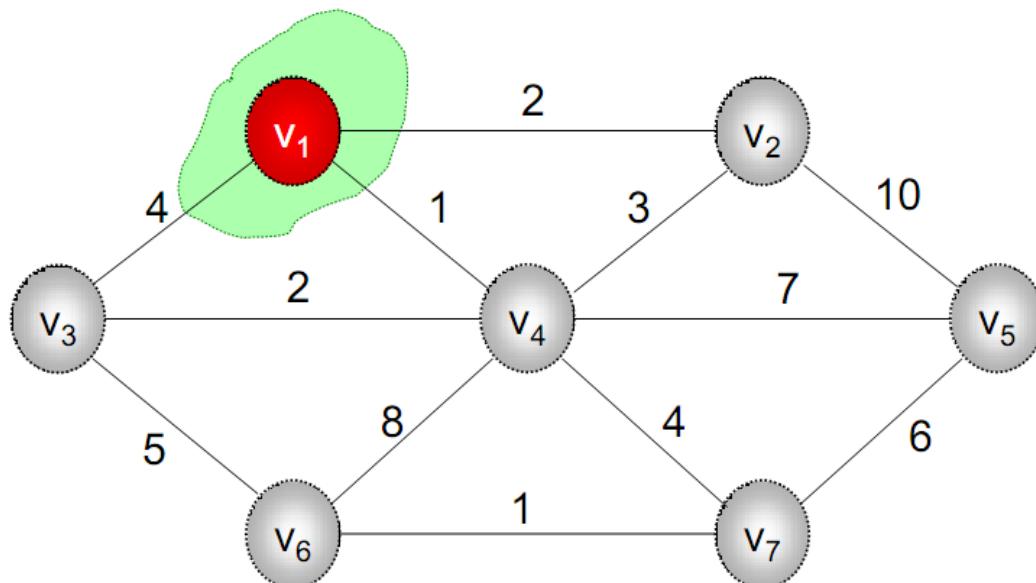
A connected, undirected graph G is given above.



## Prim's Algorithm: Example

### Step 1

Select an edge from graph: that is not in the current tree, that has the minimum cost, and that can be connected to the current tree.



|       |   |
|-------|---|
| $V_4$ | 1 |
| $V_2$ | 2 |
| $V_3$ | 4 |

Content of the priority queue

# Prim's Algorithm: Example

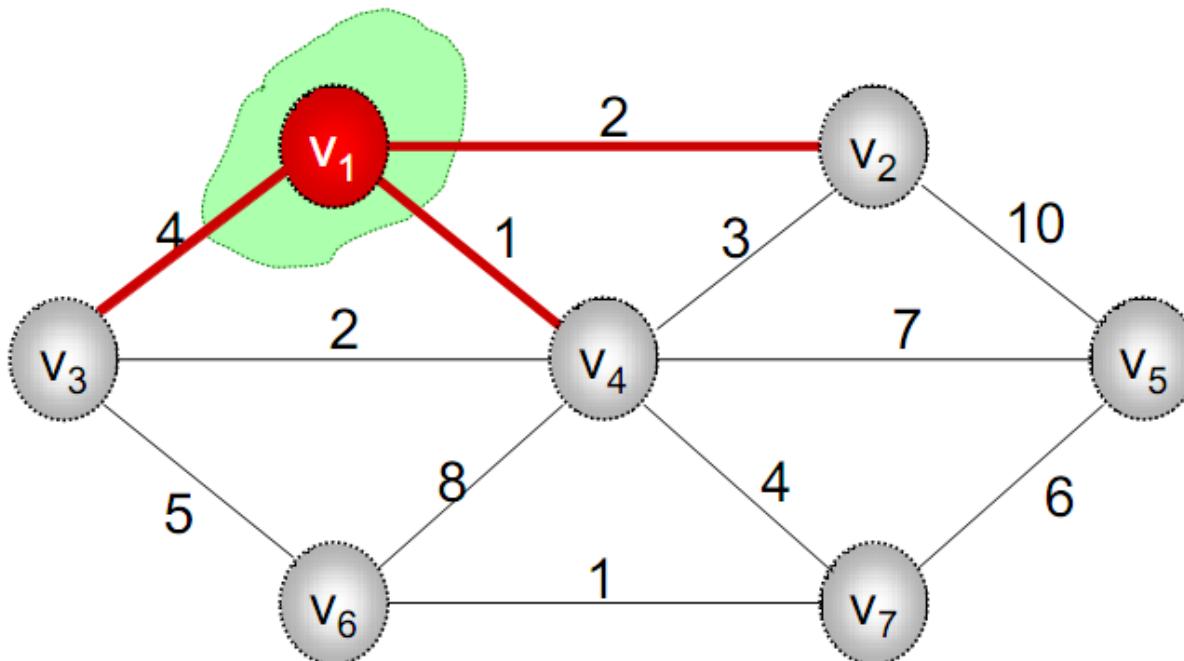
Step 1

The edges that can be connected are:

$(v_1, v_2)$ : cost 2

$(v_1, v_4)$ : cost 1

$(v_1, v_3)$ : cost 4



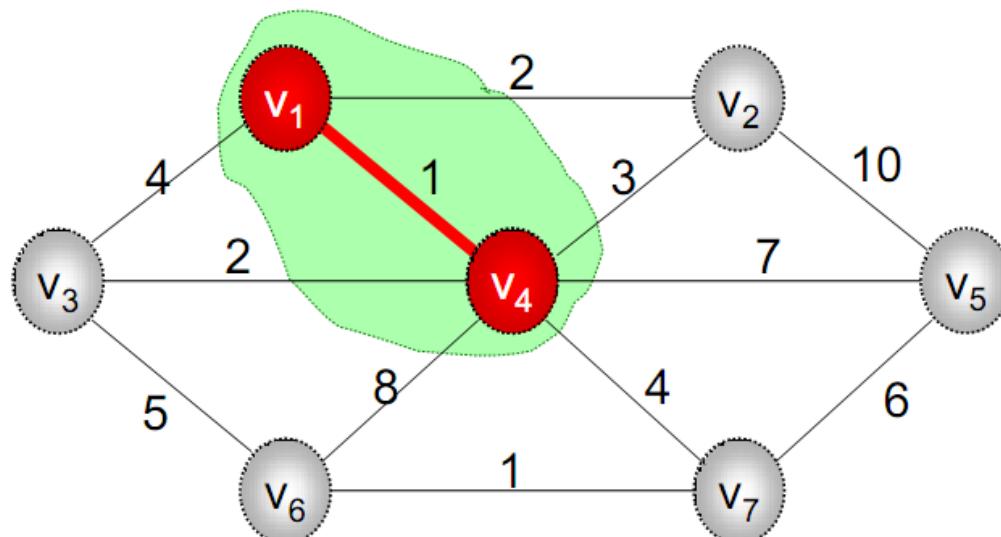
|       |   |
|-------|---|
| $V_4$ | 1 |
| $V_2$ | 2 |
| $V_3$ | 4 |

Content of the priority queue

# Prim's Algorithm: Example

## Step 2

We include the vertex  $v_4$ , that is connected to the selected edge, to the current tree. In this way we grow the tree.

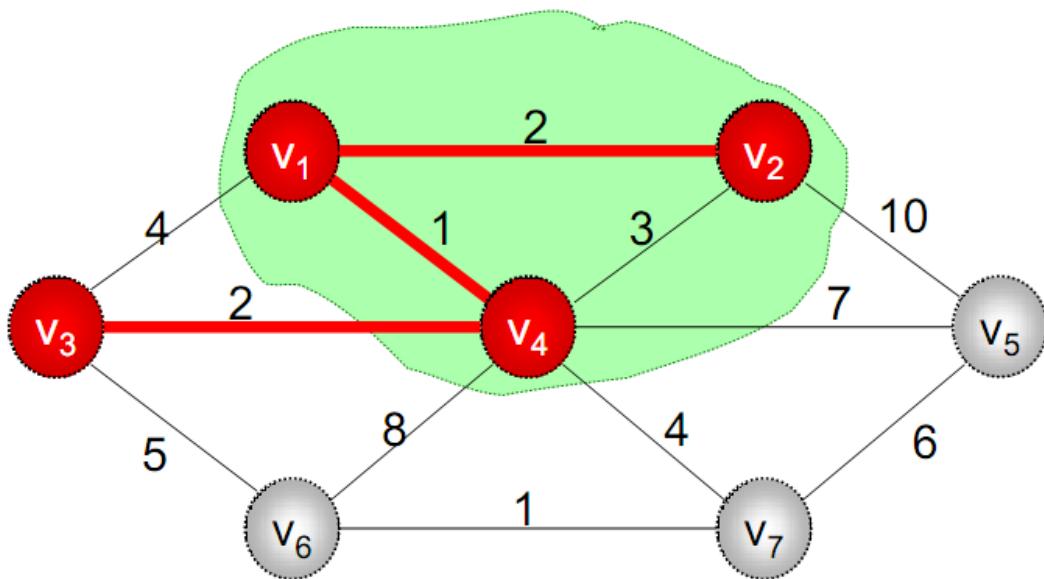


|                |   |
|----------------|---|
| V <sub>2</sub> | 2 |
| V <sub>3</sub> | 2 |
| V <sub>7</sub> | 4 |
| V <sub>5</sub> | 7 |
| V <sub>6</sub> | 8 |

Content of the priority queue

# Prim's Algorithm

*Repeat steps: 1, and 2  
Add either edge ( $v_4, v_3$ )*

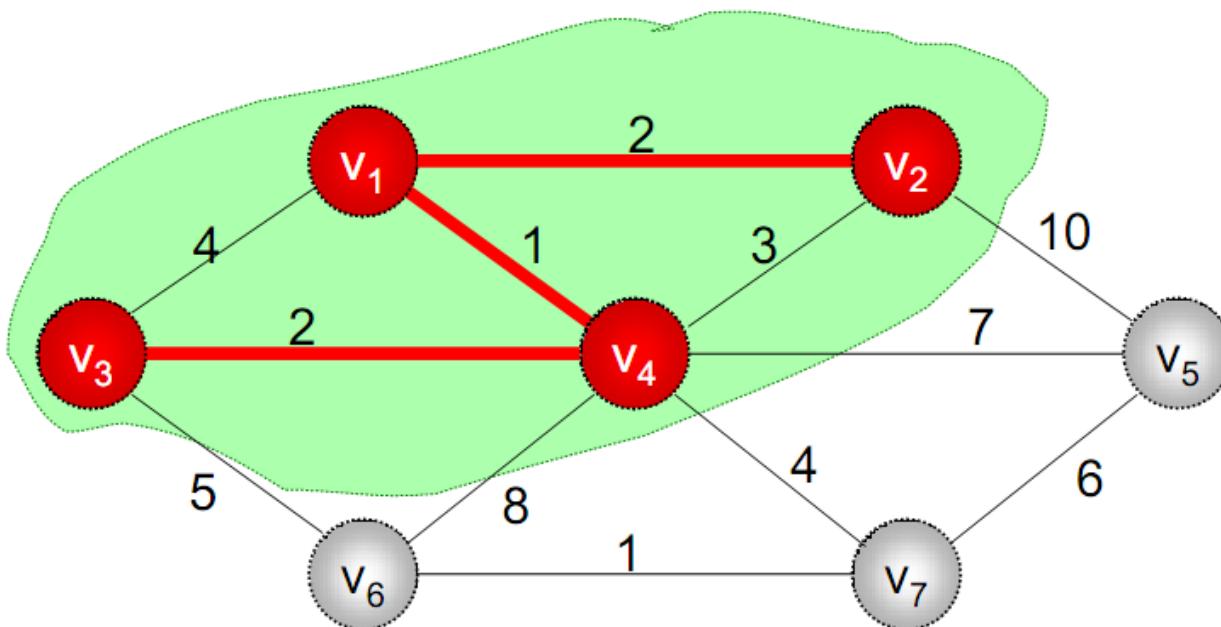


|       |   |
|-------|---|
|       |   |
| $V_3$ | 2 |
| $V_7$ | 4 |
| $V_5$ | 7 |
| $V_6$ | 8 |

Content of the priority queue

# Prim's Algorithm

*Grow the tree!*

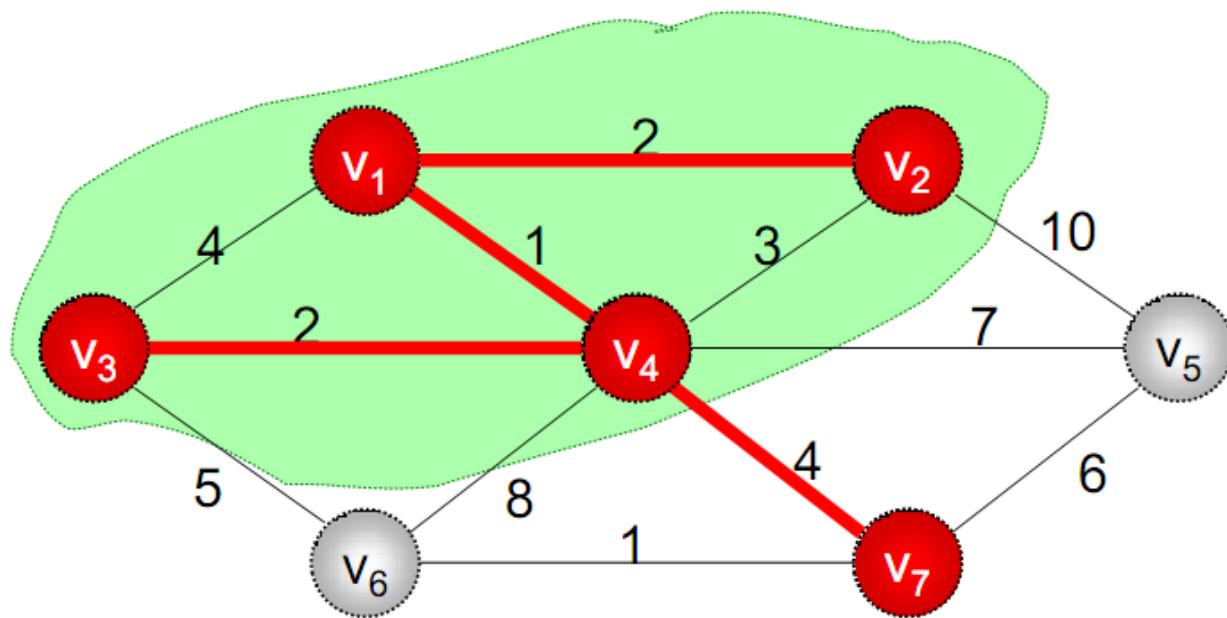


|       |   |
|-------|---|
|       |   |
| $V_7$ | 4 |
| $V_6$ | 5 |
| $V_5$ | 7 |

Content of the priority queue

# Prim's Algorithm

*Add edge  $(v_4, v_7)$*

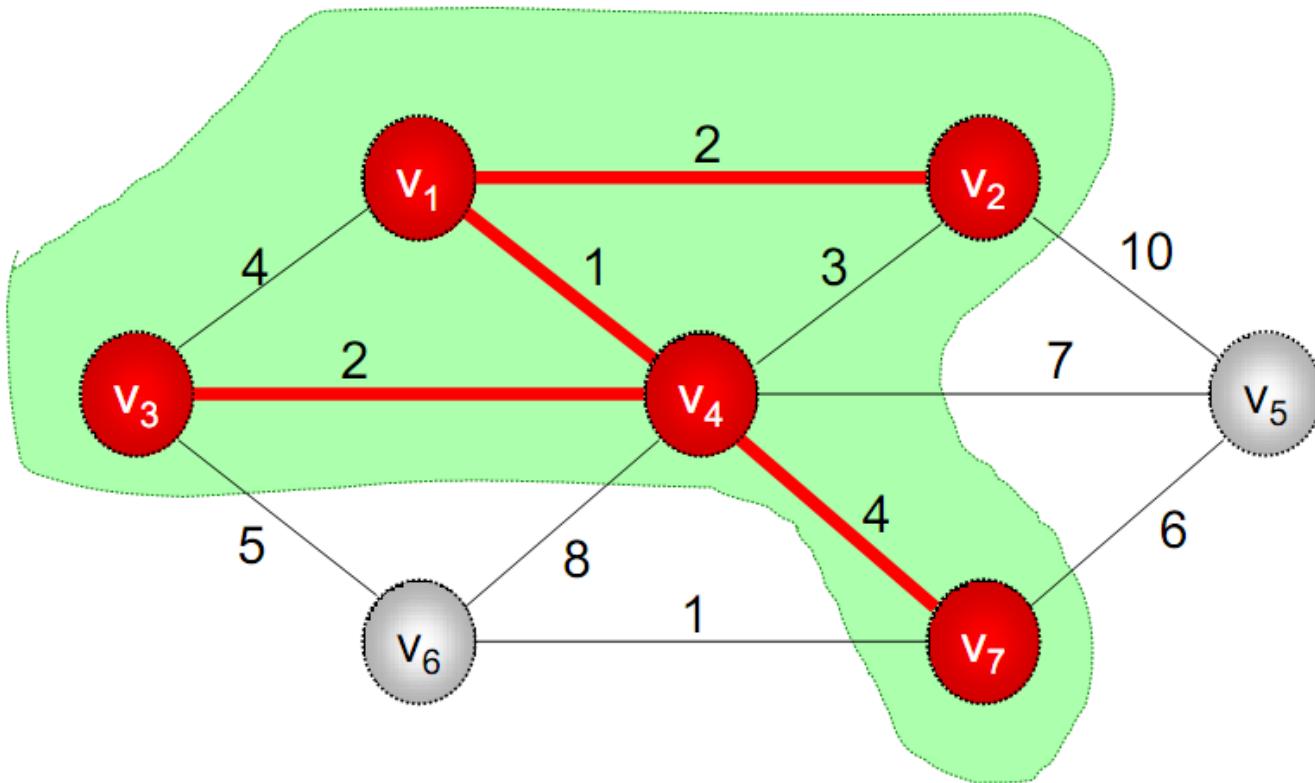


|       |   |
|-------|---|
|       |   |
| $V_7$ | 4 |
| $V_6$ | 5 |
| $V_5$ | 7 |

Content of the priority queue

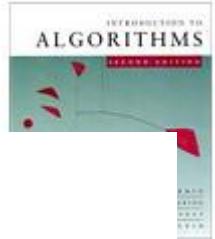
# Prim's Algorithm

*Grow the tree!*



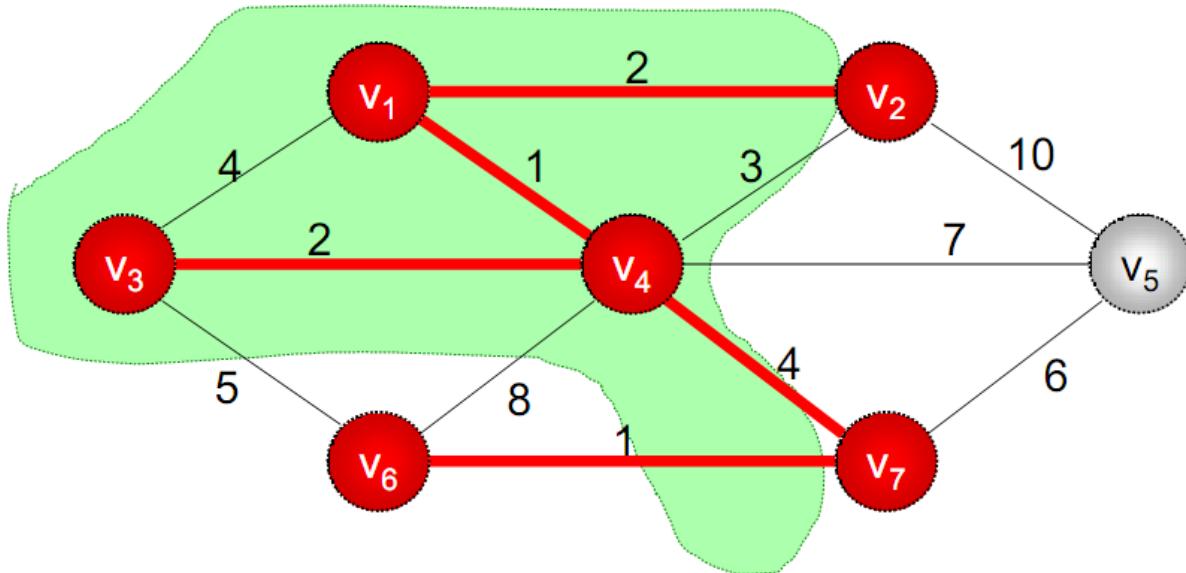
|       |   |
|-------|---|
|       |   |
|       |   |
| $V_6$ | 1 |
| $V_5$ | 6 |

Content of the priority queue



# Prim's Algorithm

*Add edge ( $v_7, v_6$ )*

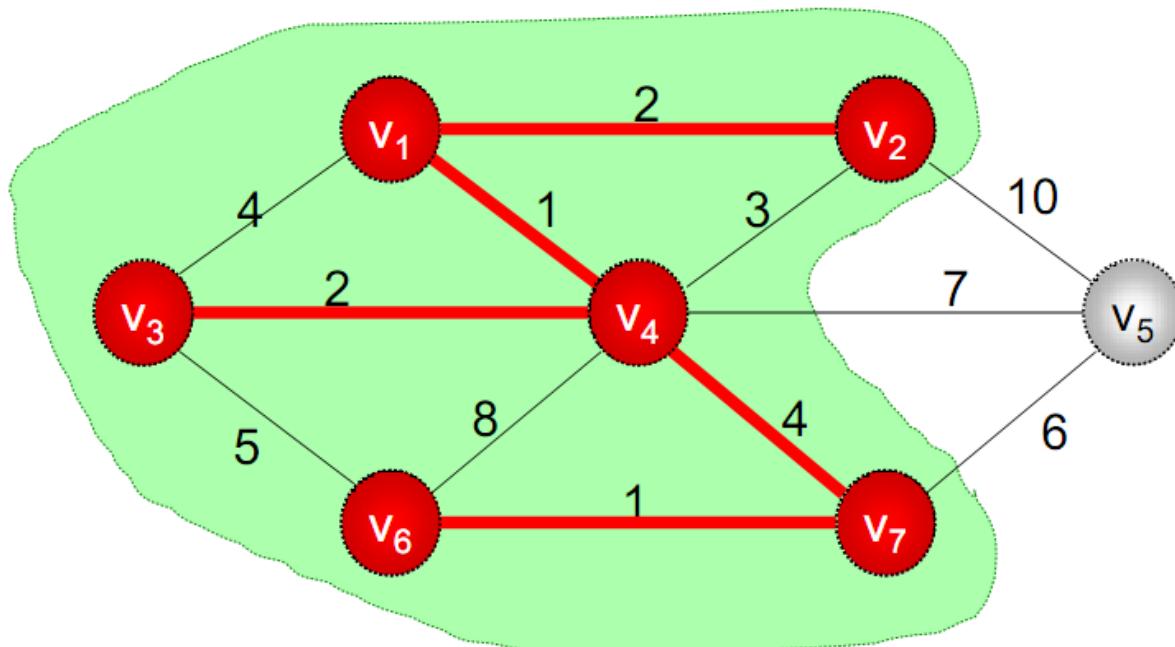


|                |   |
|----------------|---|
|                |   |
|                |   |
|                |   |
| V <sub>6</sub> | 1 |
| V <sub>5</sub> | 6 |

Content of the  
priority queue

# Prim's Algorithm

*Grow the tree!*

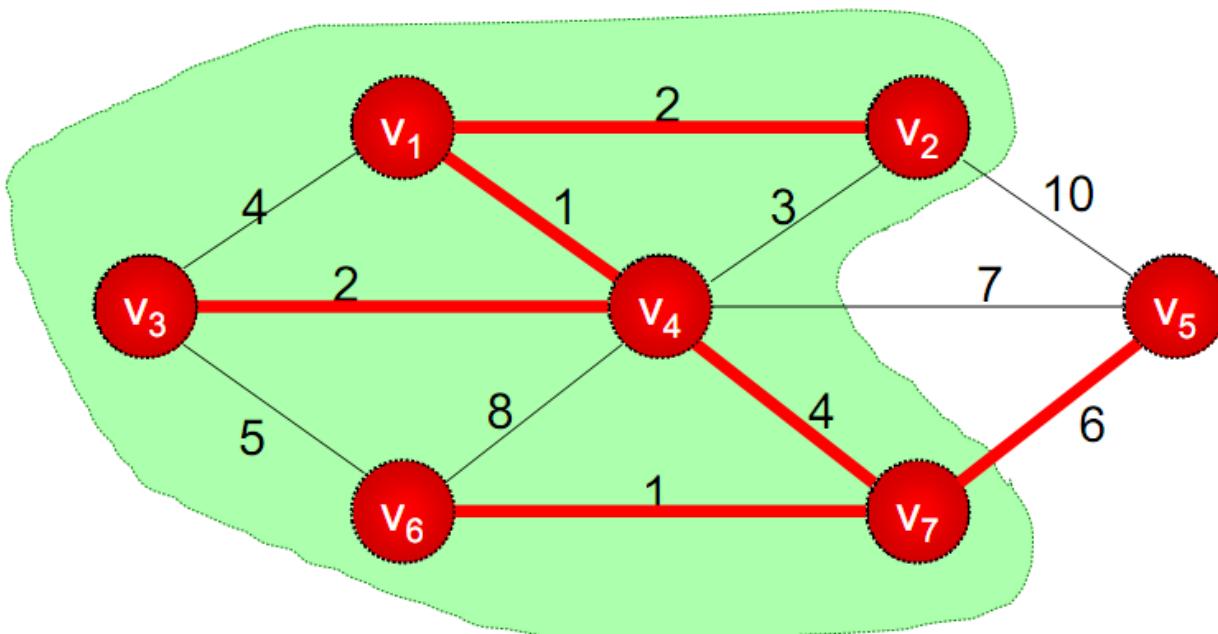


|                |   |
|----------------|---|
|                |   |
|                |   |
|                |   |
| V <sub>5</sub> | 6 |

Content of the  
priority queue

# Prim's Algorithm

Add edge  $(v_7, v_5)$

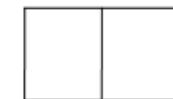
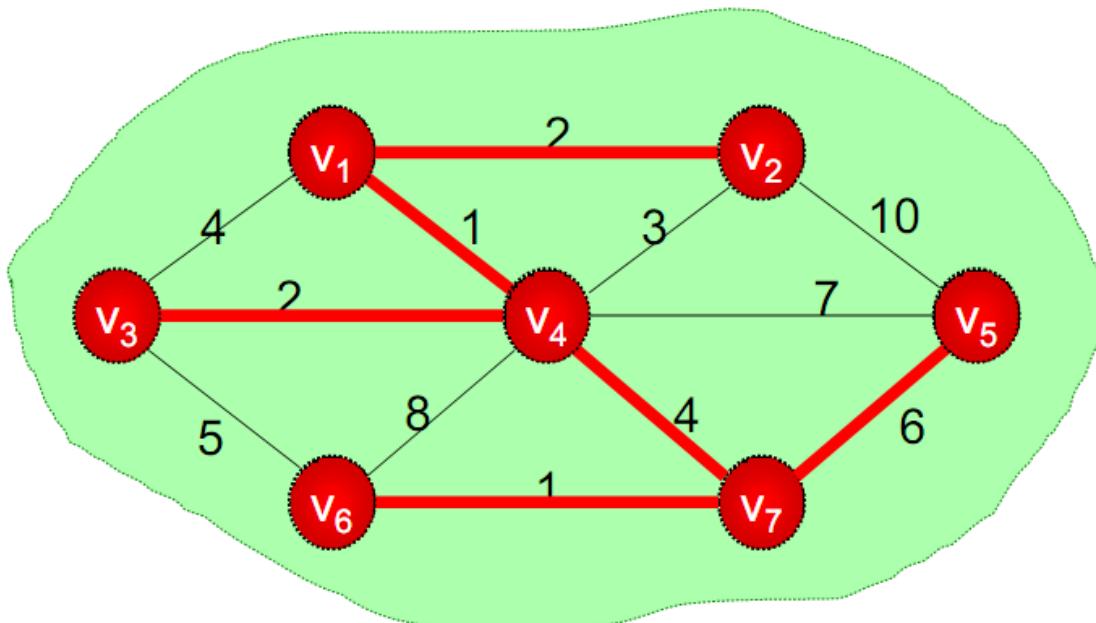


|                |   |
|----------------|---|
|                |   |
|                |   |
|                |   |
|                |   |
| V <sub>5</sub> | 6 |

Content of the priority queue

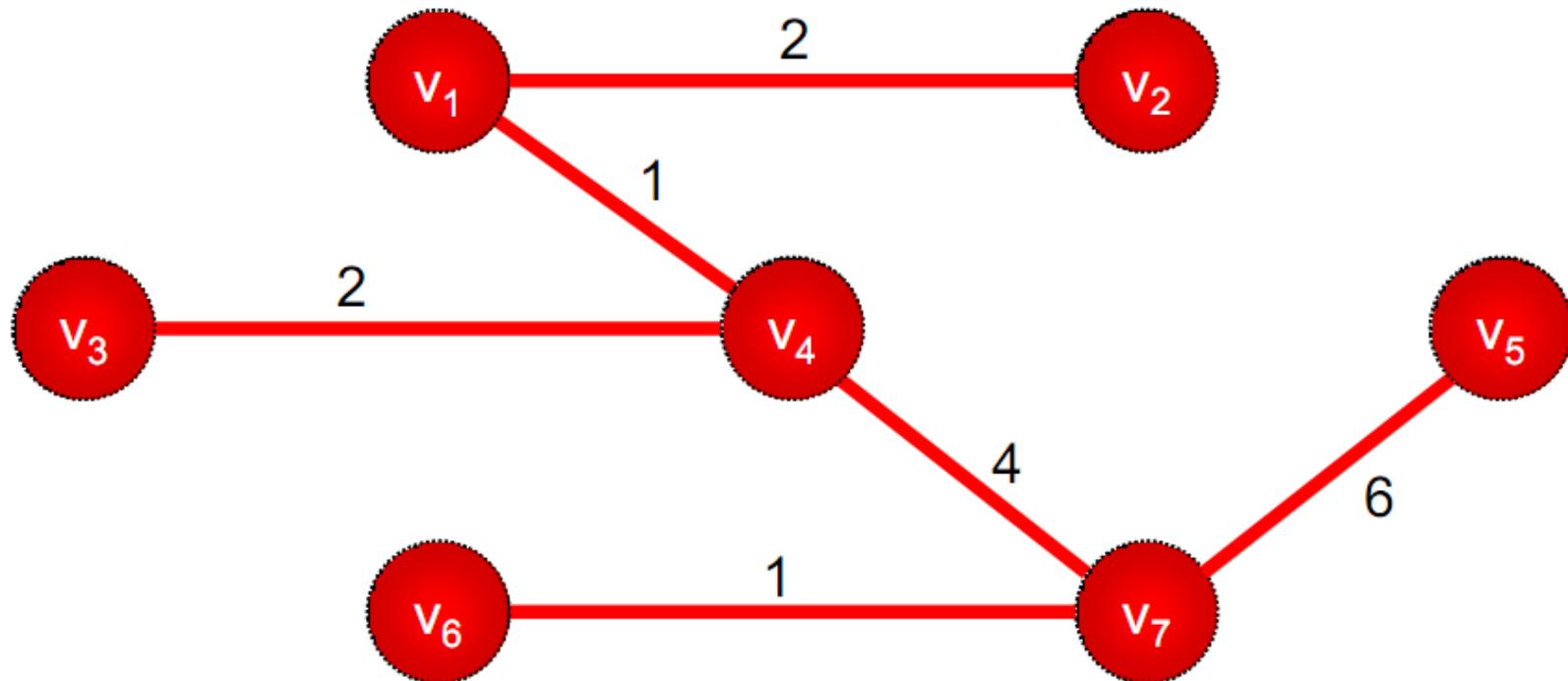
# Prim's Algorithm: Example

Grow the tree!

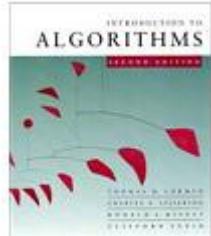


Content of the  
priority queue

# Prim's Algorithm: Example



Finished! The resulting MST is shown below!



## MST algoritmaları

Kruskal algoritması:

- *Kopuk-küme veri yapısı* 'nı kullanır.
- Koşma süresi =  $O(E \lg V)$ .

Bugüne kadar en iyisi:

- Karger, Klein, and Tarjan [1993].
- Rastgele algoritma.
- $O(V + E)$  beklenen süre.

# Kruskal'ın Algoritması

- Kenar tabanlı bir algoritmadır. Kenarlar küçükten büyüğe sıralanır.
- Graf üzerindeki düğümler, aralarında bağlantı olmayan  $N$  tane bağımsız küme gibi düşünülür.
- Daha sonra bu kümeler tek tek maliyeti en az olan kenarlarla birleştirilir (çevrim oluşturmayacak şekilde).
- Düğümler arasında bağlantı olan tek bir küme oluşturulmaya çalışılır.
- Küme birleştirme işleminde en az maliyetli olan kenardan başlanılır; daha sonra kalan kenarlar arasından en az maliyetli olanlar seçilir.

# Kruskal'ın Algoritması – Kaba Kod

- Yol ağacını oluşturan kenarların tutulduğu A dizisini oluşturur.
- Grafdaki kenarları içeren K dizisini oluşturur. yolUzunluğuna başlangıç değerini ver.

```

while(K≠{Ø} && yolUzunluğu < N) {
 K içerisinden en düşük maliyetli ki kenarını al ve
 onu K'den sil.

```

```
if(ki A'ya eklendiğinde çevrim oluşturmuyorsa) {
```

```
 ki'yi A'ya ekle.
```

```
 yolUzunluğu++
```

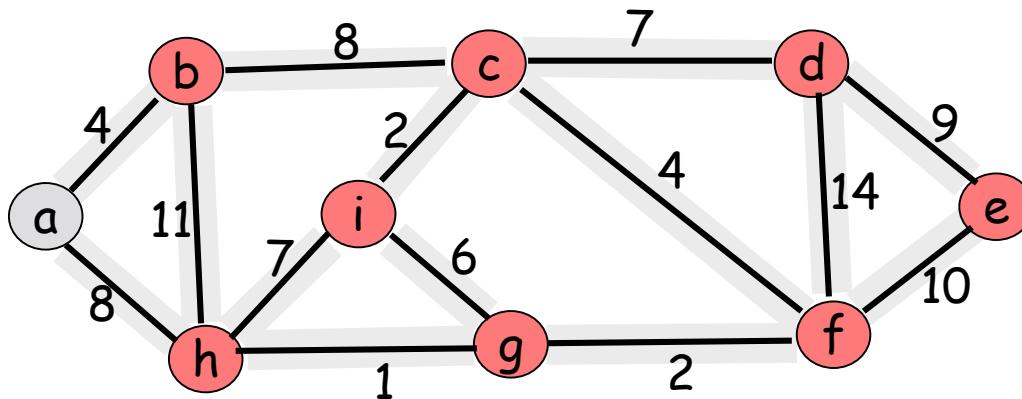
```
}
```

```
}
```

MST-KRUSKAL( $G, w$ )

- 1  $A = \emptyset$
- 2 **for** each vertex  $v \in G.V$
- 3     MAKE-SET( $v$ )
- 4     sort the edges of  $G.E$  into nondecreasing order by weight  $w$
- 5     **for** each edge  $(u, v) \in G.E$ , taken in nondecreasing order by weight
- 6         **if** FIND-SET( $u$ ) ≠ FIND-SET( $v$ )
- 7              $A = A \cup \{(u, v)\}$
- 8             UNION( $u, v$ )
- 9     **return**  $A$

# Kruskal'ın Algoritması: Örnek

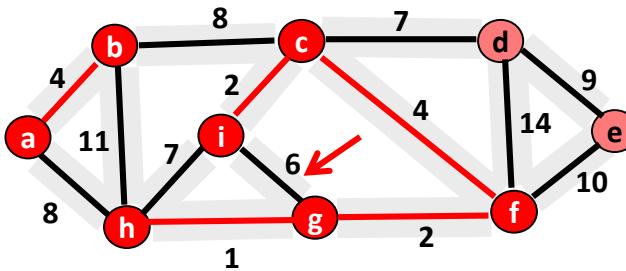
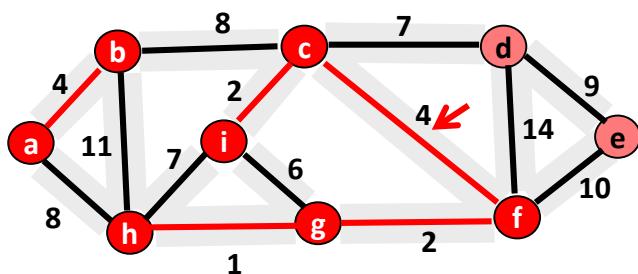
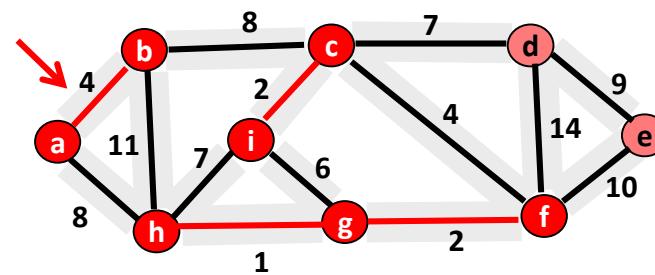
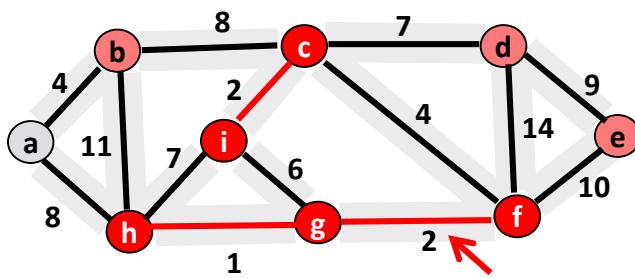
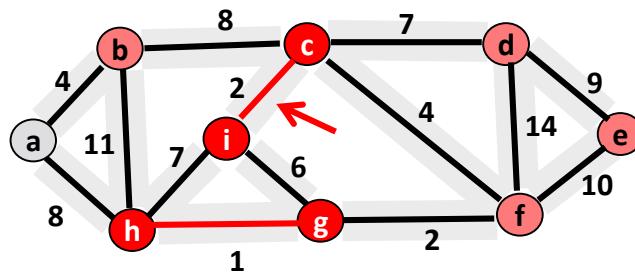
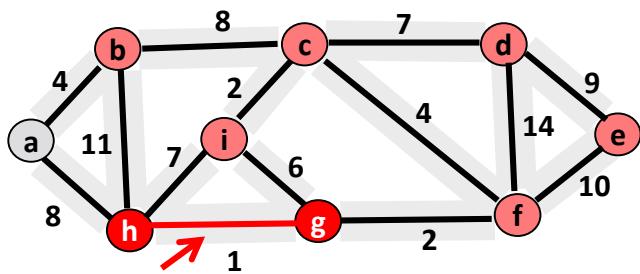


Sıralı kenar listesi

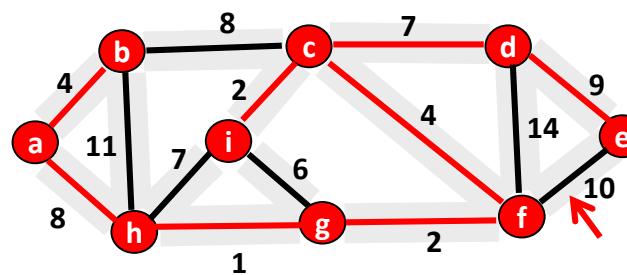
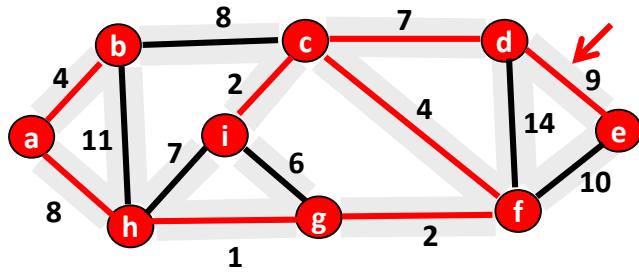
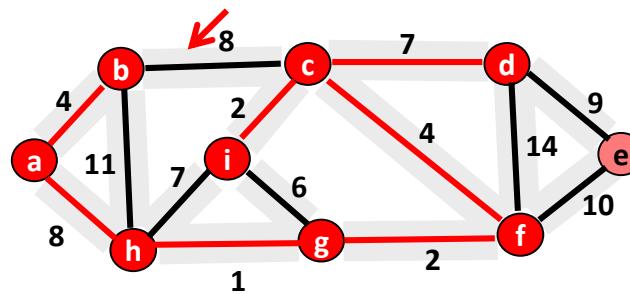
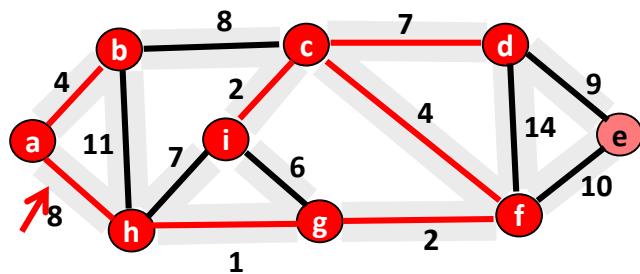
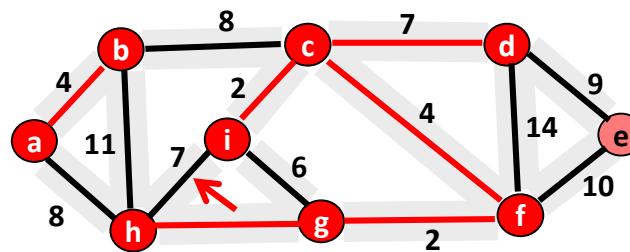
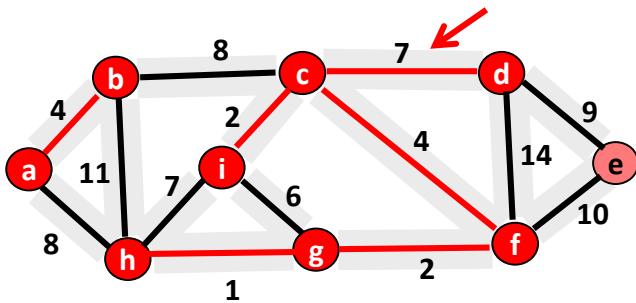
- (h, g)
- (i, c)
- (g, f)
- (a, b)
- (c, f)
- (i, g)
- (c, d)

- (i, h)
- (a, h)
- (b, c)
- (d, e)
- (e, f)
- (b, h)
- (d, f)

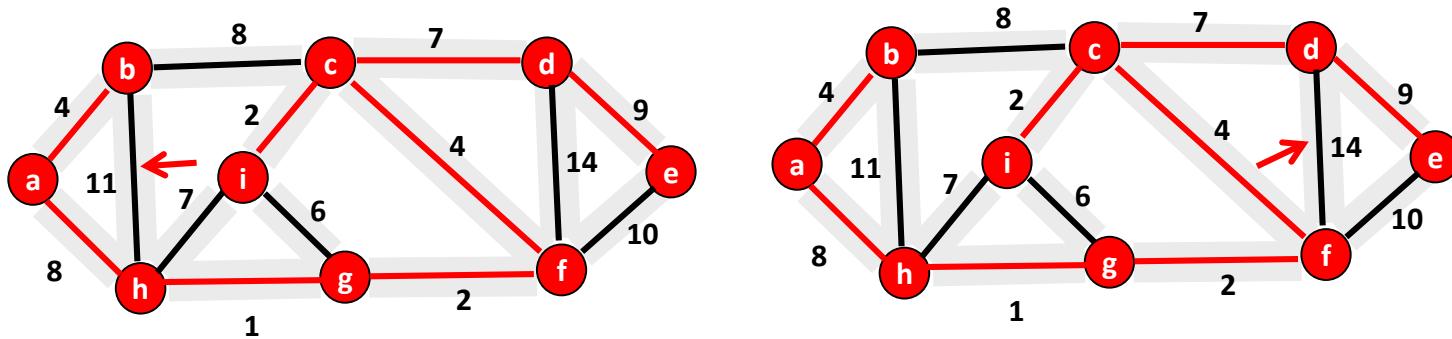
# Kruskal'ın Algoritması: Örnek



# Kruskal'ın Algoritması: Örnek



# Kruskal'ın Algoritması: Örnek

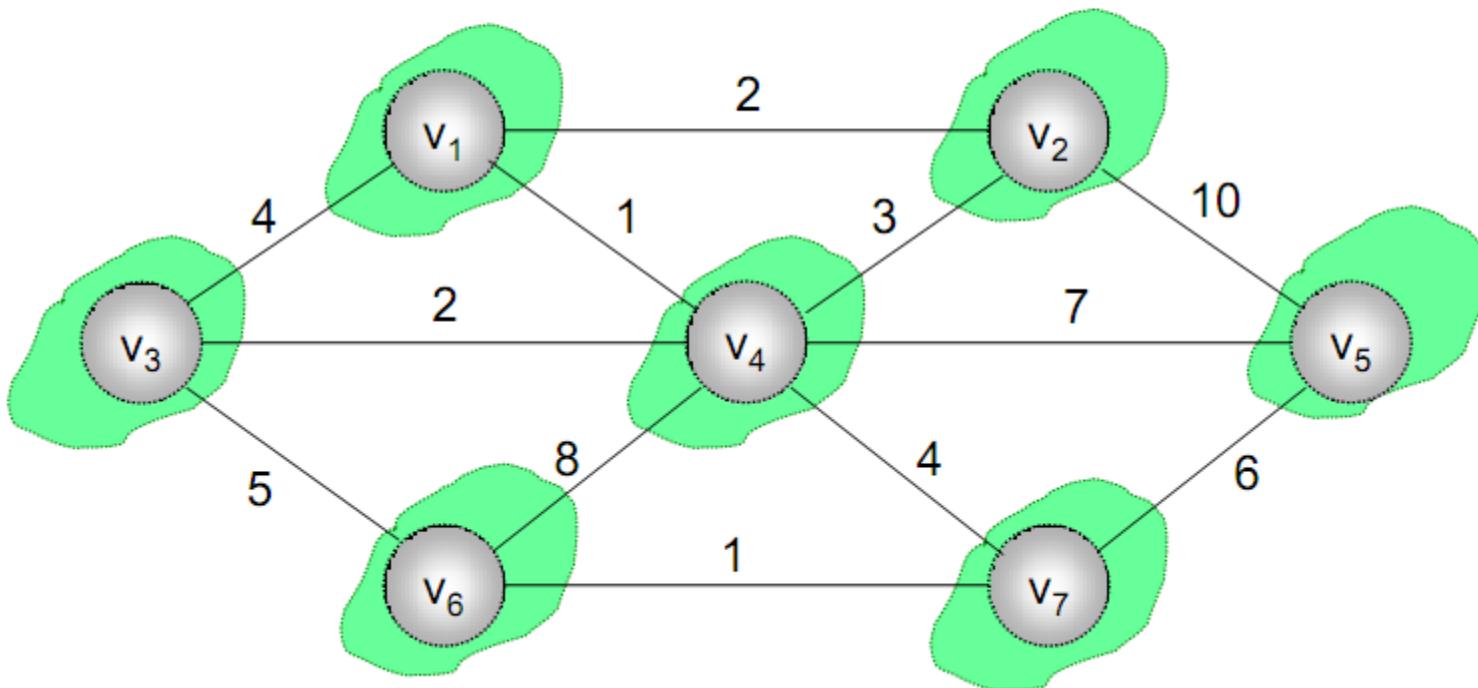


Sıralı kenar listesi

(h, g)    (i, c)    (g, f)    (a, b)    (c, f)    (i, g)    (c, d)

(i, h)    (a, h)    (b, c)    (d, e)    (e, f)    (b, h)    (d, f)

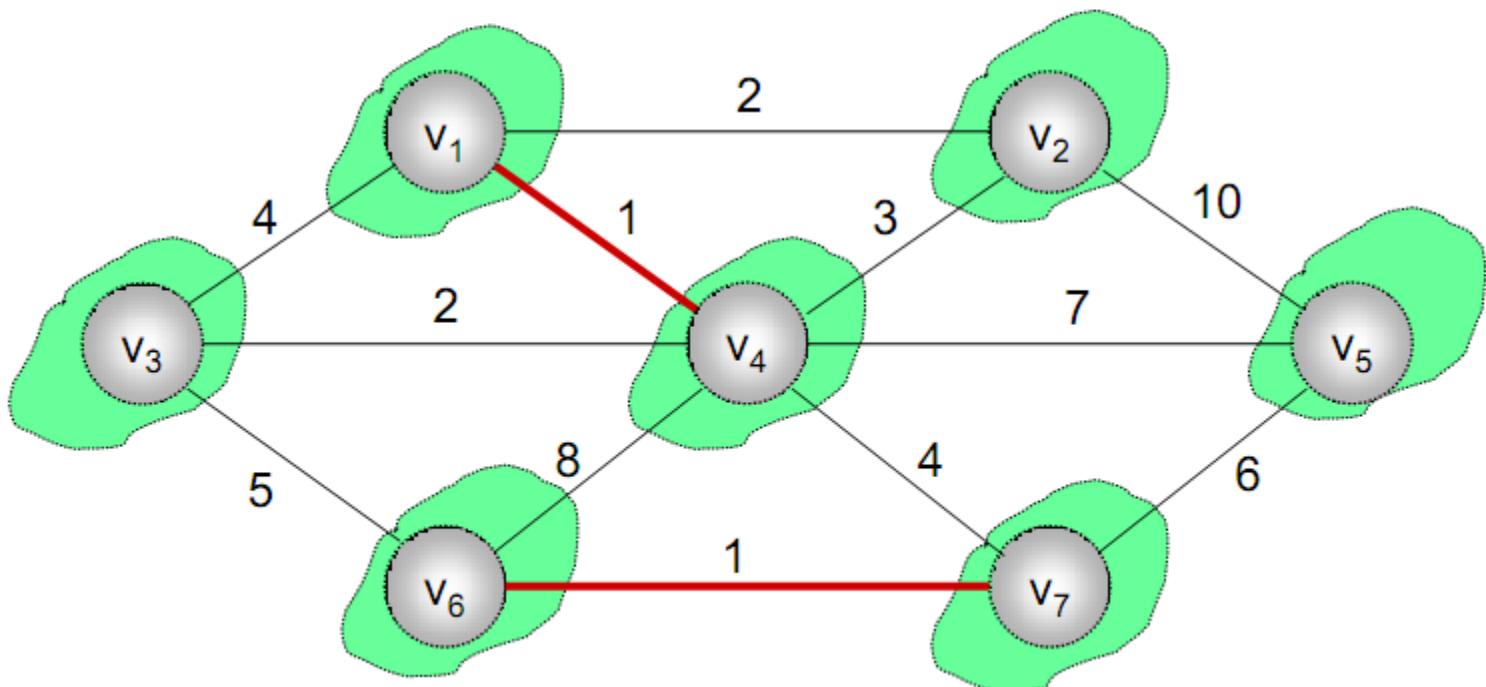
# Initial Forest



# Initial Forest

Step 1

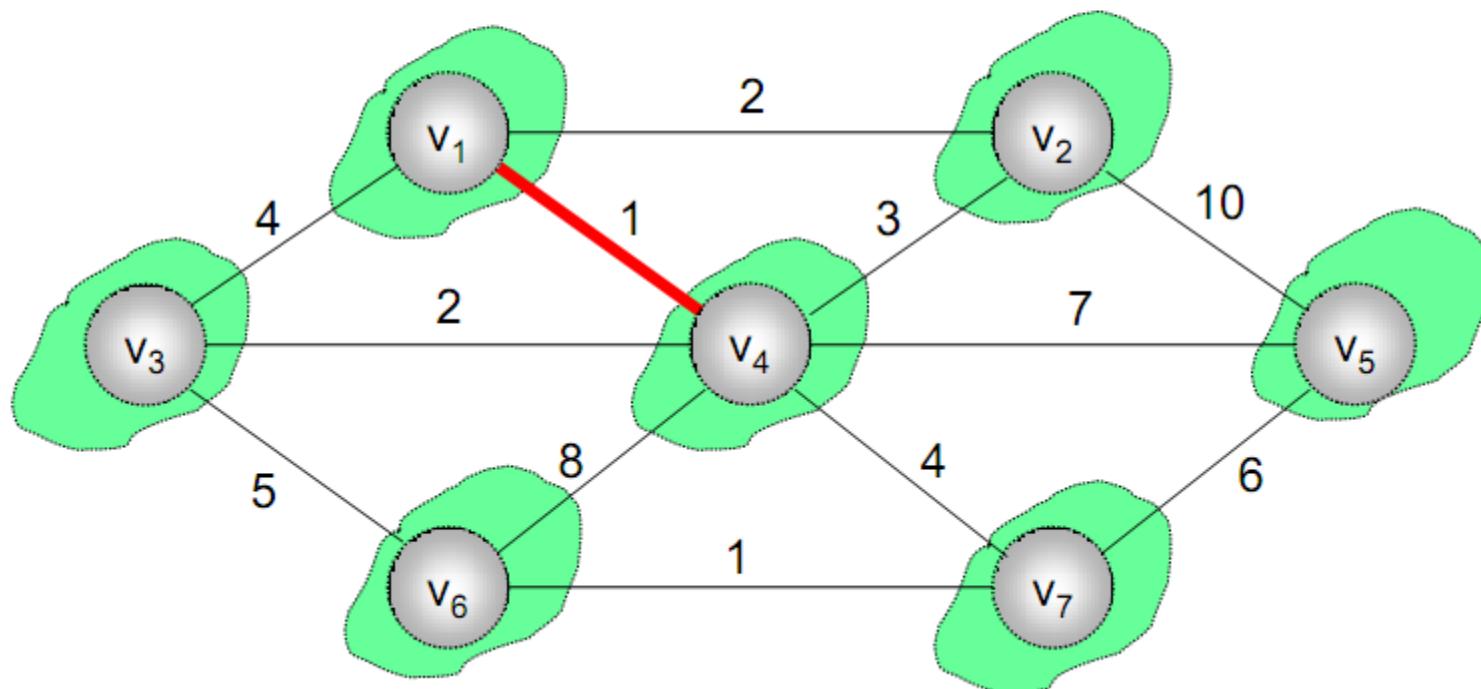
*Candidate edges are shown  
(edges that have low cost and  
edges that connect two trees)*



# Initial Forest

## Step 2

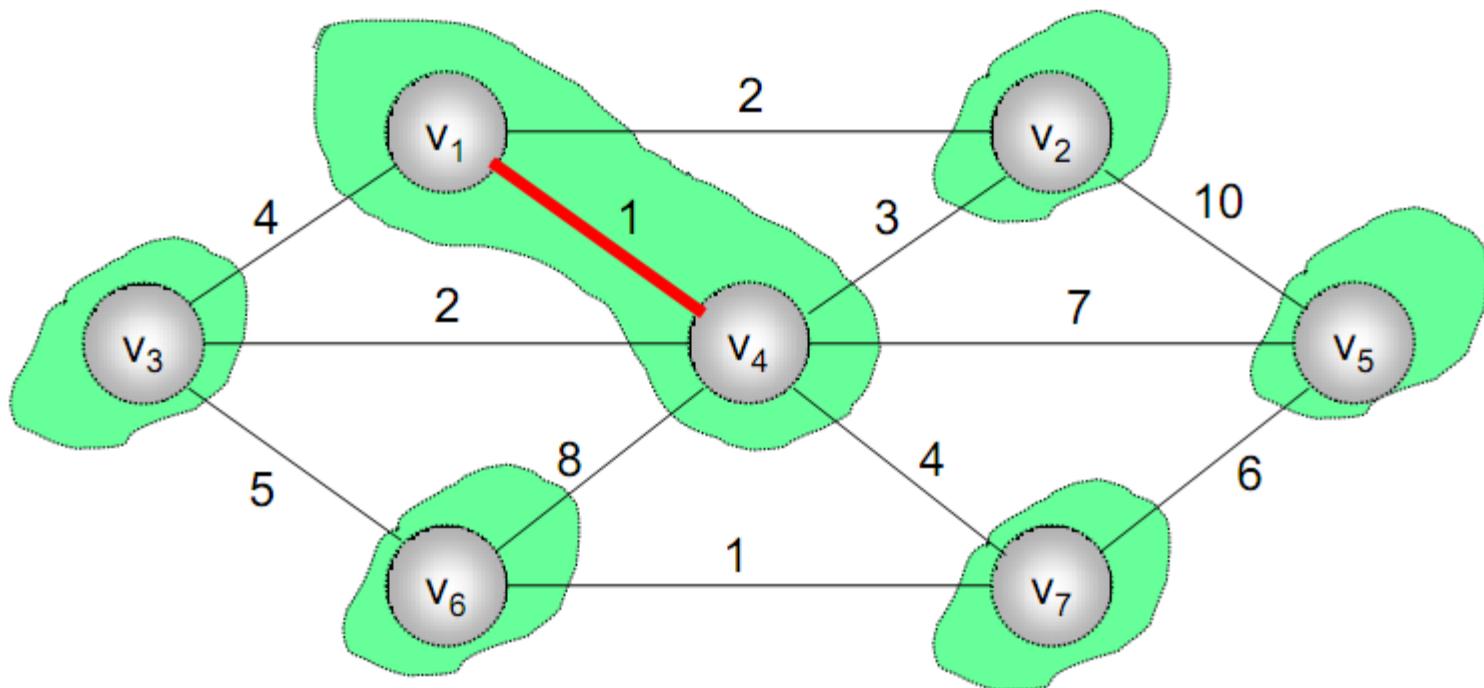
*Accept one of the candidate edges:  $(v_1, v_4)$   
(we can do random accept here).*



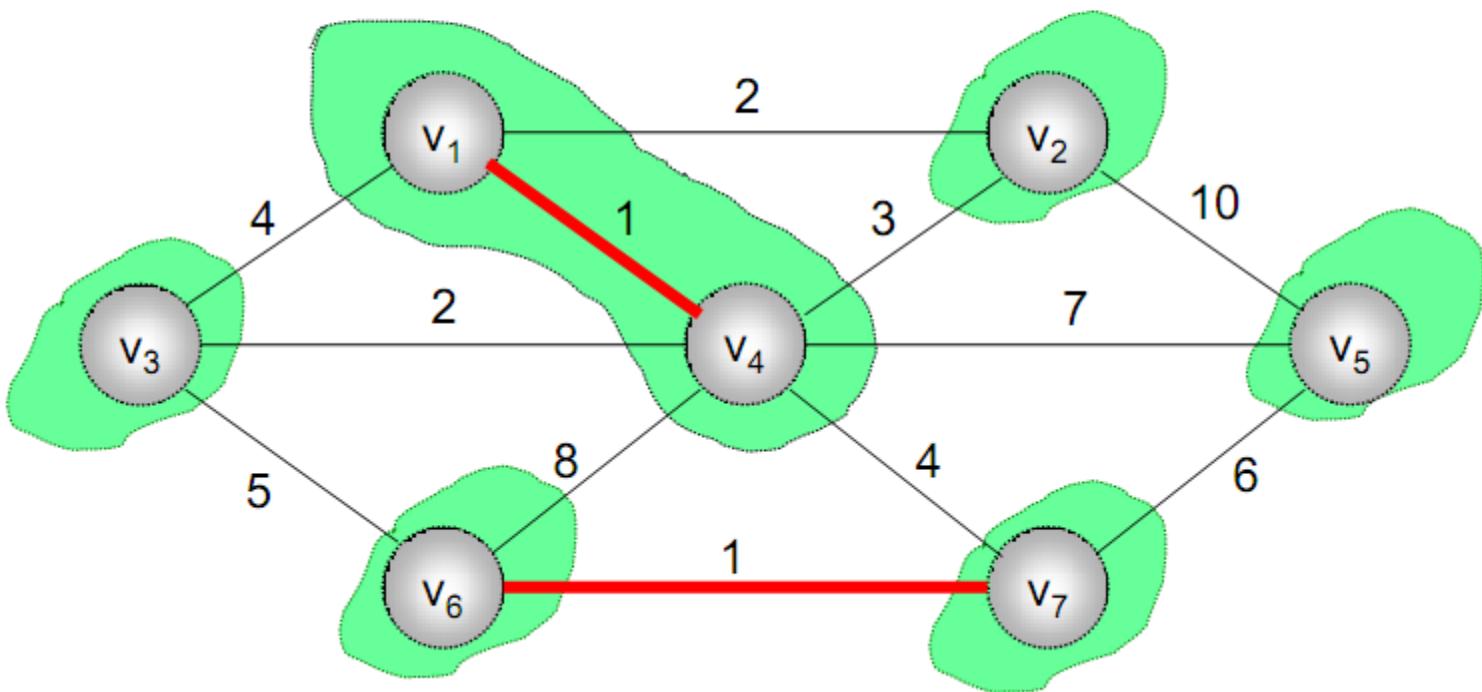
# Initial Forest

## Step 3

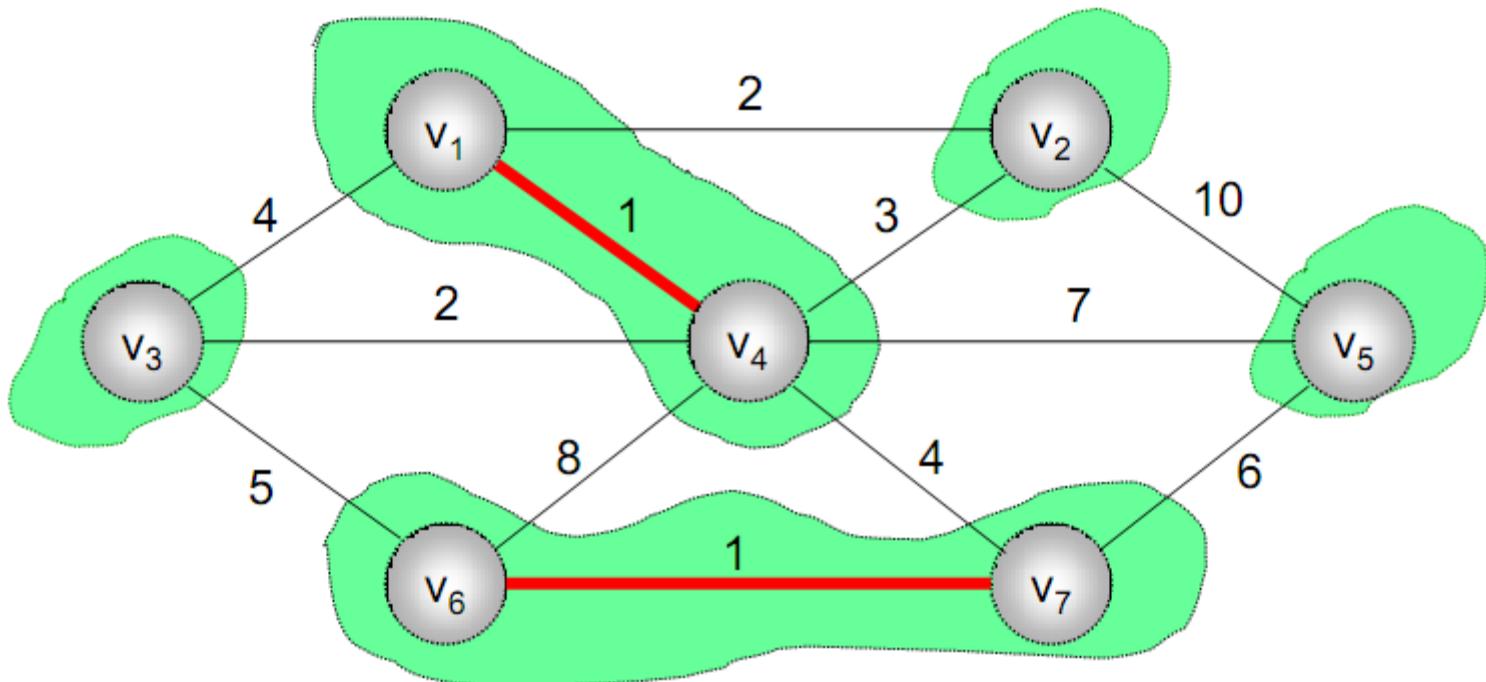
*Merge the two trees connected by that edge.  
Obtain a new tree in this way.*



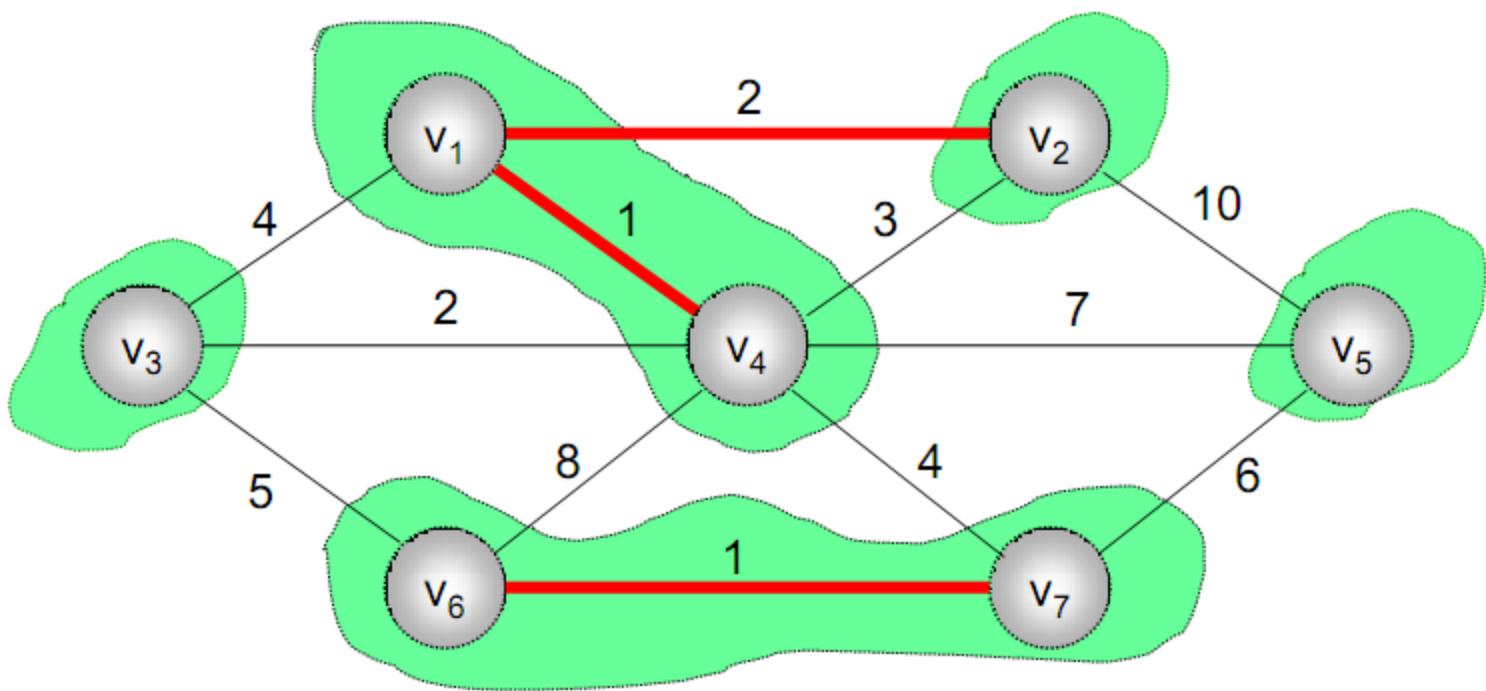
*Repeat previous steps!  
Edge  $(v_6-v_7)$  is accepted.*



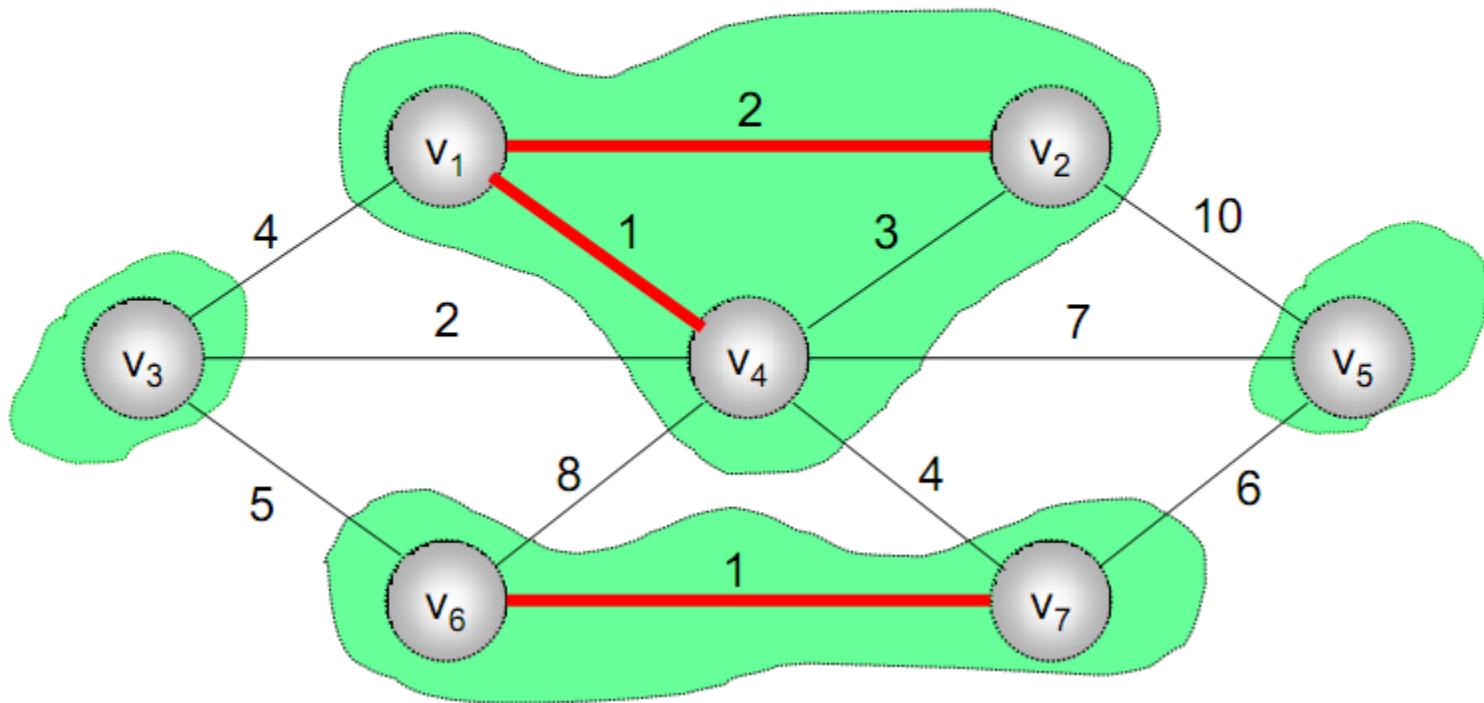
*Merge the two trees connected by that edge!*



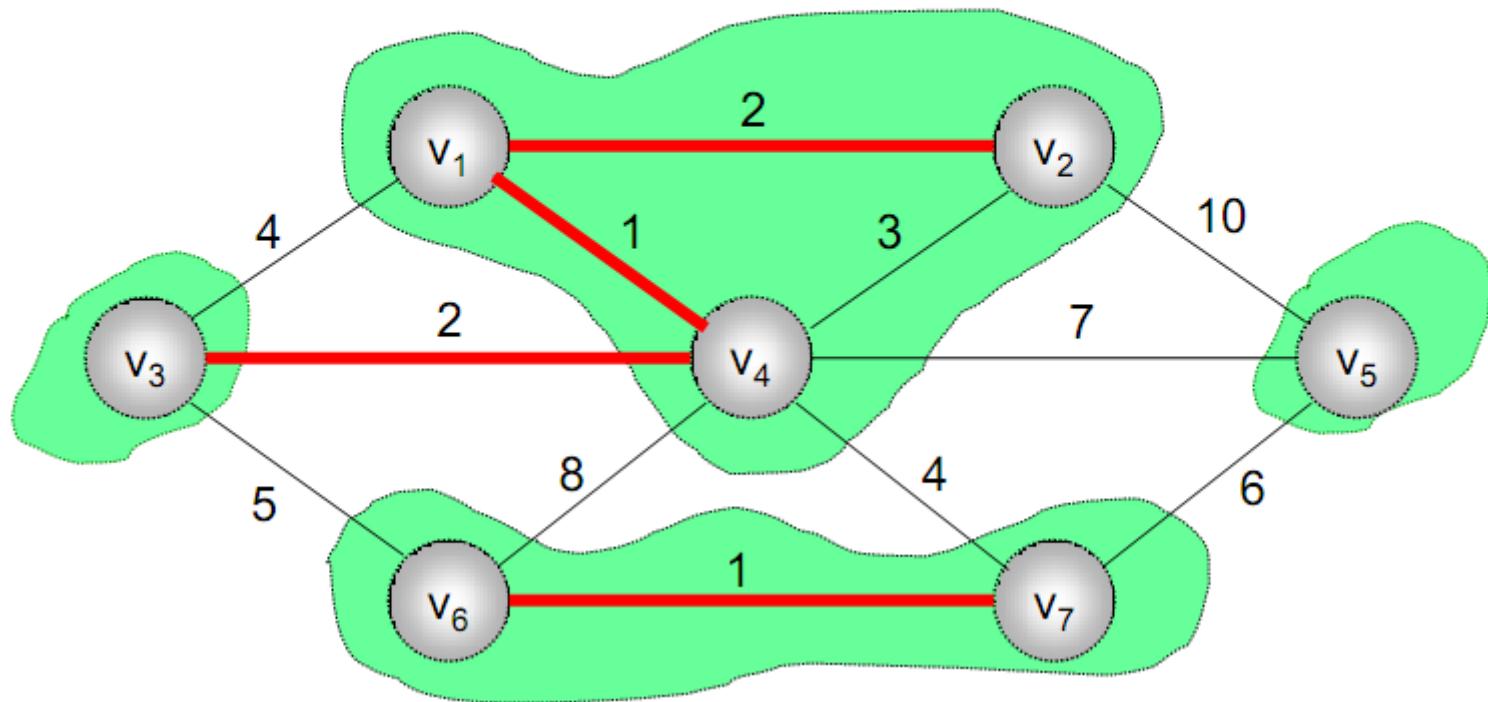
*Accept edge  $(v_1, v_2)$*



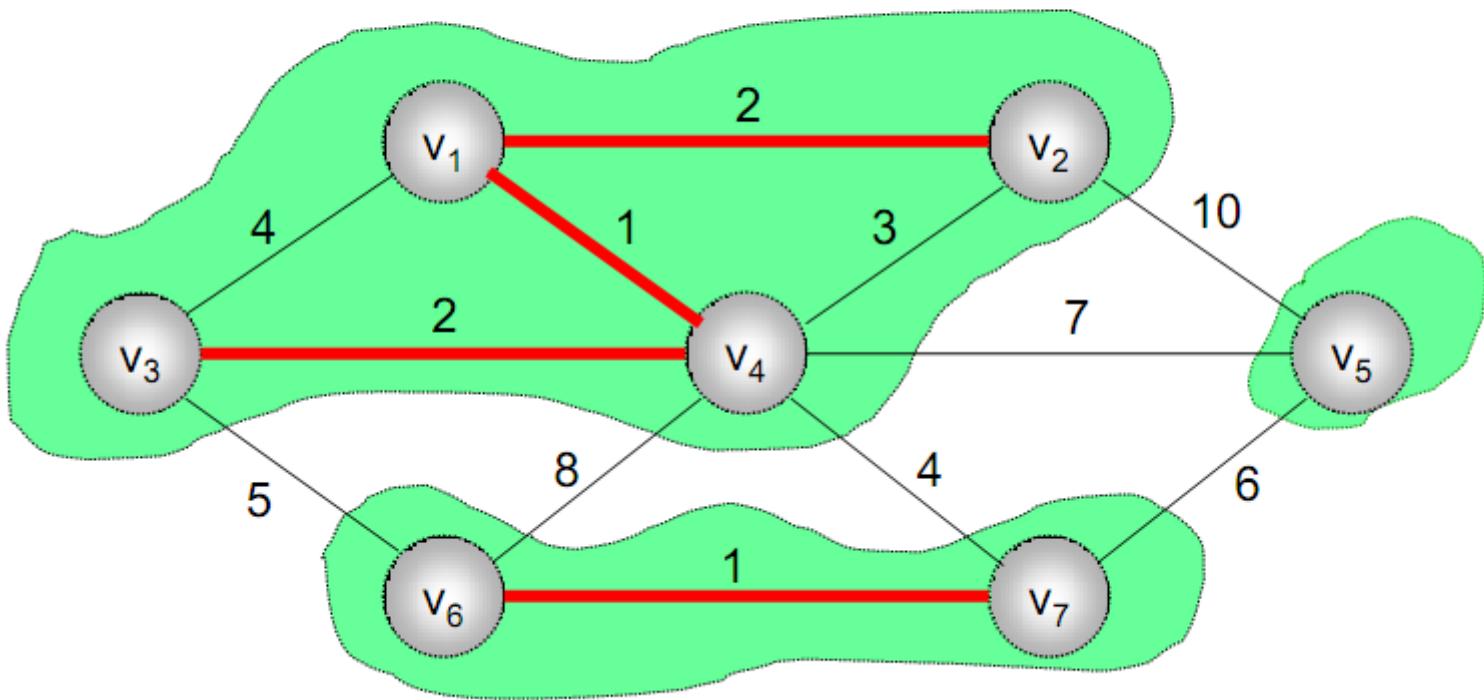
*Merge the two trees connected by that edge!*



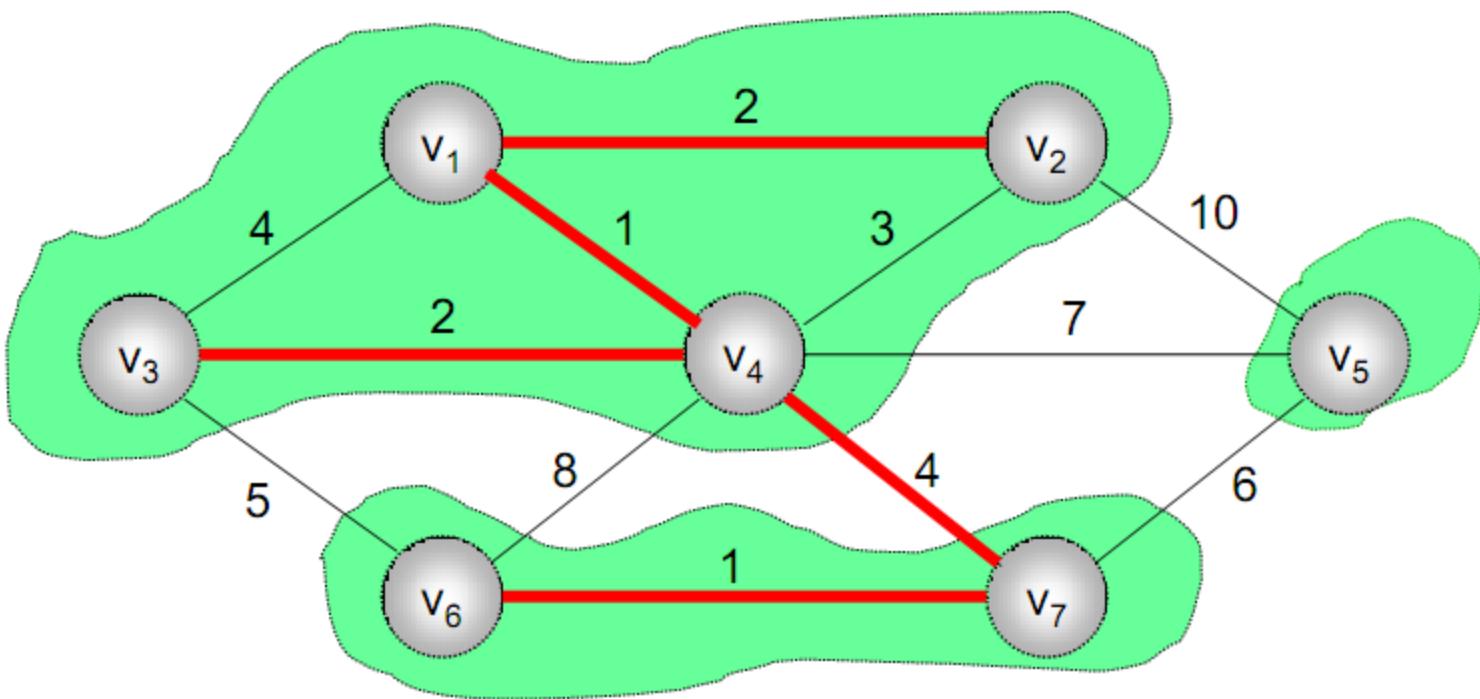
Accept edge  $(v_3, v_4)$



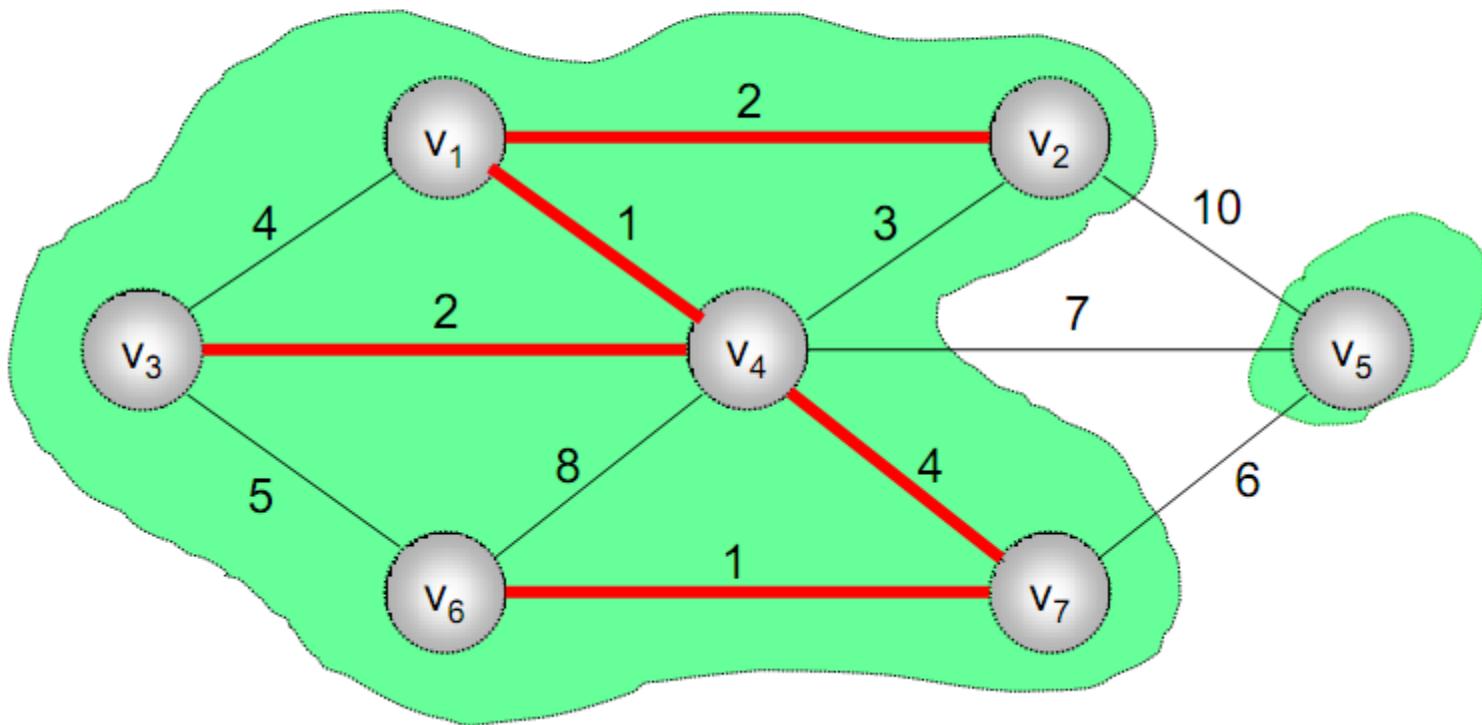
*Merge the two trees connected by that edge!*



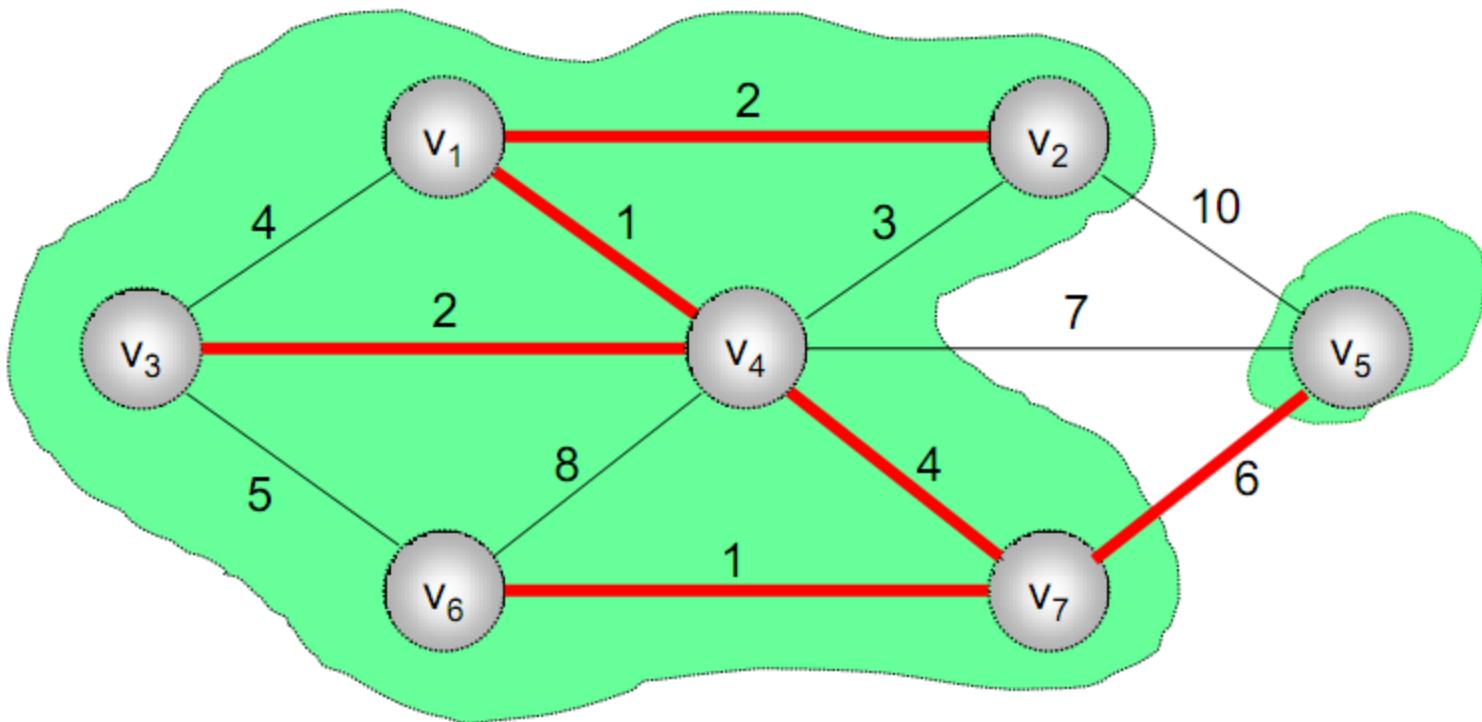
Accept edge  $(v_4, v_7)$



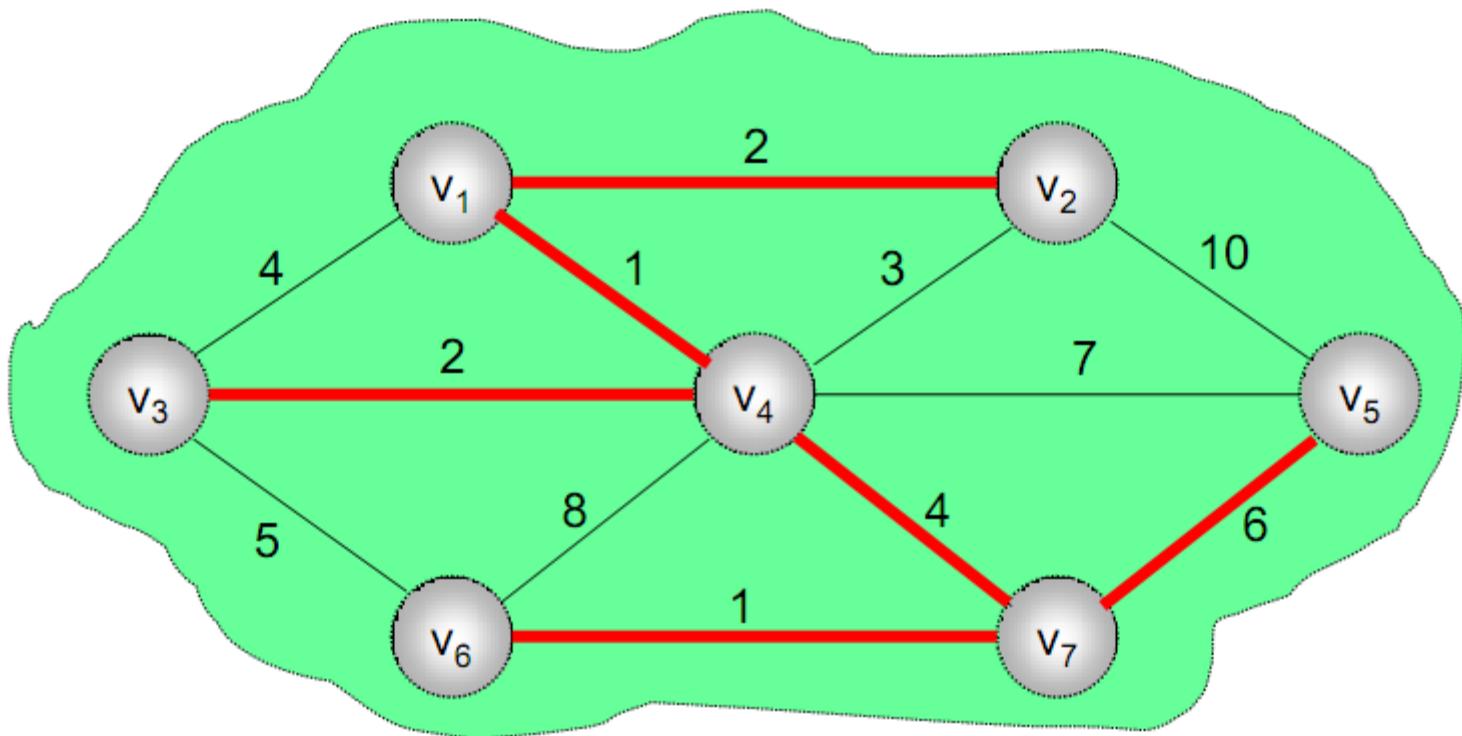
*Merge the two trees connected by that edge!*



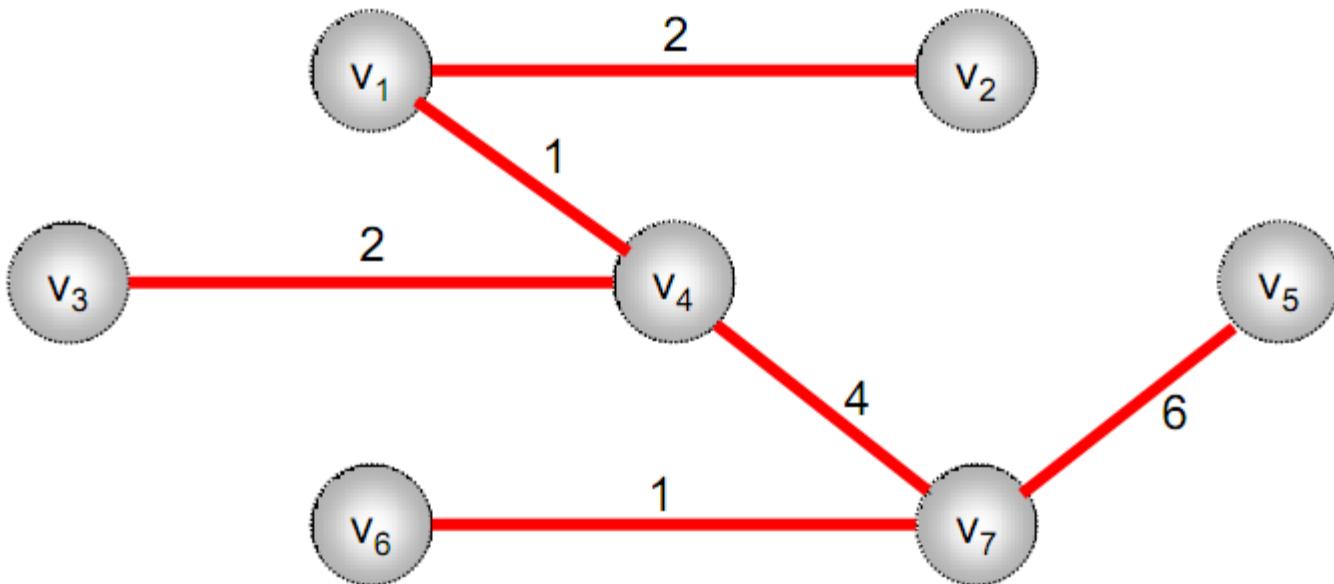
Accept edge  $(v_7, v_5)$



*Merge the two trees connected by that edge!*



Finished!  
The resulting MST is shown below!



# Sollin(Boruvka) 'in Algoritması

- Minimum geçişli ağaçların bulunmasına yönelik bu algoritma 1926 yılında Boruvka tarafından önerilmiştir.
- Sollin algoritması aynı anda birden çok ağaçla başlayan ve sonraki adımlarda ağaçların birleşerek tek bir ağaca dönüştüğü bir algoritmadır.
- Bu algoritmada, bir adımda birden fazla kenar seçilir ve yol ağacı ara işlemlerde alt ağaç olarak bulunan ağaçlara eklenmek suretiyle bulunur.
- Yani yol ağacı belirlenené kadar ara işlemlerde birden fazla ağaç bulunur.
- Kruskal ve Prim'in algoritmalarına göre sonuca daha az adımda ulaşan Sollin algoritmasının ara işlemleri bu algoritmala göre daha fazla olabilir.
- Çalışma zamanı  $O(E \log V)$

# Sollin 'in Algoritması- Kaba Kod

$SK \leftarrow \{ \emptyset \}$ ; boş seçilen kenarlar ( $SK$ ) dizisi oluşturur.

Başlangıç anında her düğüm için En Küçük Maliyetli (EKM) kenarı seç ve sonuçta tüm düğümleri içeren altağaçları (ormanı) oluşturur.

while (ormandaki ağaç sayısı > 1 VEYA kenar sayısı <  $N$ )

{

Her ağaç, genişletmek için, o ağaca herhangi bir yerden bağlı en az maliyetli kenarı al. O kenarı ilgili altağaca ekle.

Aynı kenar birden çok seçilmişse ilki dışındakileri önemseme.

}

# Sollin'in Algoritması

BORÜVKA( $V, E$ ):

$$F = (V, \emptyset)$$

while  $F$  has more than one component  
 choose leaders using DFS  
 $\text{FINDSAFEEDGES}(V, E)$   
 for each leader  $\bar{v}$   
     add  $\text{safe}(\bar{v})$  to  $F$

loop iterates  $O(\log V)$  times

FINDSAFEEDGES( $V, E$ ):

for each leader  $\bar{v}$

$$\text{safe}(\bar{v}) \leftarrow \infty$$

for each edge  $(u, v) \in E$

$$\bar{u} \leftarrow \text{leader}(u)$$

$$\bar{v} \leftarrow \text{leader}(v)$$

$$\text{if } \bar{u} \neq \bar{v}$$

$$\text{if } w(u, v) < w(\text{safe}(\bar{u}))$$

$$\text{safe}(\bar{u}) \leftarrow (u, v)$$

$$\text{if } w(u, v) < w(\text{safe}(\bar{v}))$$

$$\text{safe}(\bar{v}) \leftarrow (u, v)$$

Each call to FINDSAFEEDGES takes  $O(E)$  time

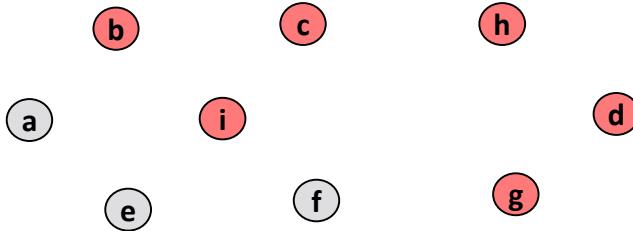
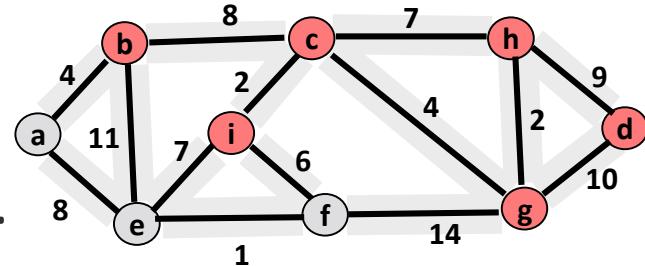
running time of Borùvka's algorithm is  $O(E \log V)$ .

# Sollin 'in Algoritması- Kaba Kod

- Adım 1: Algoritmada her düğüm en küçük maliyet değerine sahip komşu düğümü belirler.
- Adım2- Belirlediği bu düğümle kendisi arasında bir kenar oluşturur. Bu belirleme sürecinde aynı kenarın birden fazla seçildiği durumda, ilkinden sonraki durumlar grafın yönsüz graf olması nedeniyle önemsiz kabul edilir.
- Adım 3- Meydana gelen alt ağaçlar daha sonra kenar maliyeti değeri dikkate alınarak yeniden birleştirilir. Uygulama, sonunda tek bir ağaç kalana kadar devam eder.

# Sollin'in Algoritması: Örnek

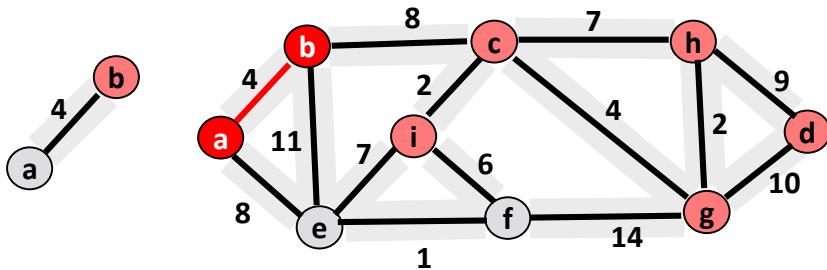
- Öncelikler tüm ayrıtları siliniz.



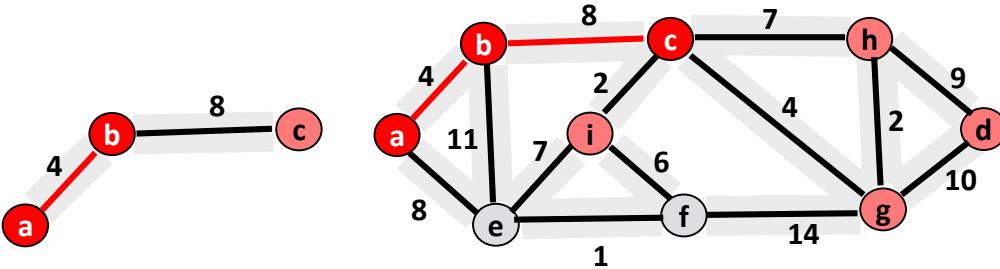
- Başlangıç durumu
- Düğümler sıralanır  $\{a,b,c,d,e,f,g,h,i\}$  ve sıradaki her düğüm için minimum maliyetli yollar seçilir.

# Sollin'in Algoritması: Örnek

- a düğümü için (a-b) seçilir (en düşük maliyetli kenar) bu kenar çizilerek maliyeti M dizisine eklenir  $M=\{4\}$

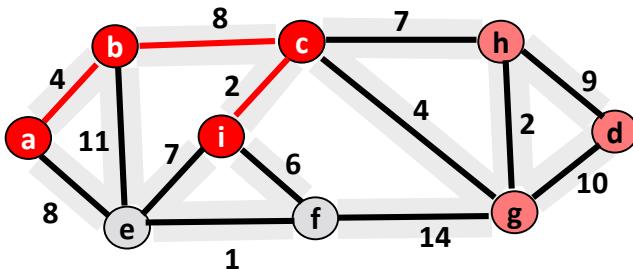


- b düğümü için (b-c) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir  $M=\{4+8\}=12$

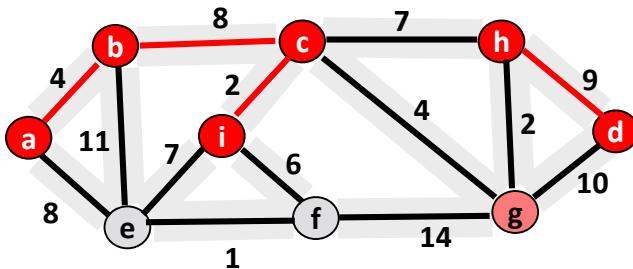


# Sollin'in Algoritması: Örnek

- c düğümü için ( $c-i$ ) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir,  $M=\{12+2\}=14$

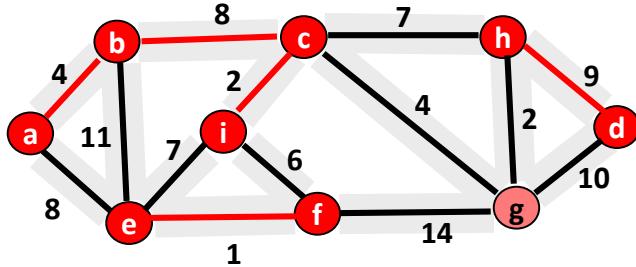


- d düğümü için ( $d-h$ ) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir,  $M=\{14+9\}=23$



# Sollin'in Algoritması: Örnek

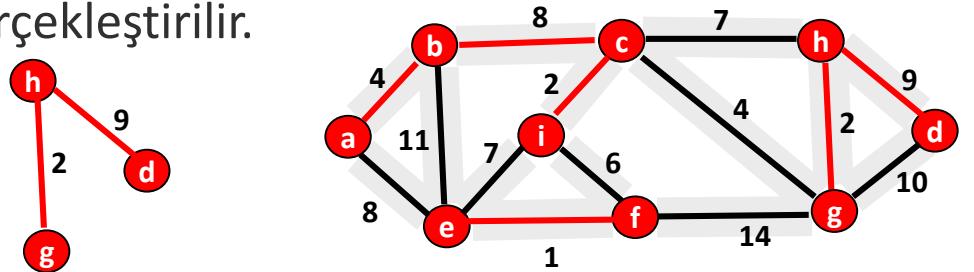
- e düğümü için (e-f) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir,  $M=\{23+1\}=24$



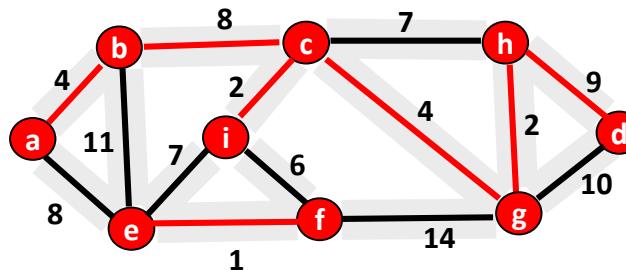
- f düğümü için (f-e) seçilir bu kenar daha önce çizildiği için maliyeti toplam minimum maliyet M dizisine eklenmez.

# Sollin'in Algoritması: Örnek

- g düğümü için (g-h) seçilir bu kenar çizilerek toplam minimum maliyet M dizisine eklenir,  $M=\{24+2\}=26$ . Bu aşamada d-g-h altağacı bulunur. Bu adımdan sonra altağaçlar birleştirilmelidir. Kalan düğümler içinden minimum maliyetlisi seçilerek birleştirme gerçekleştirilecektir.

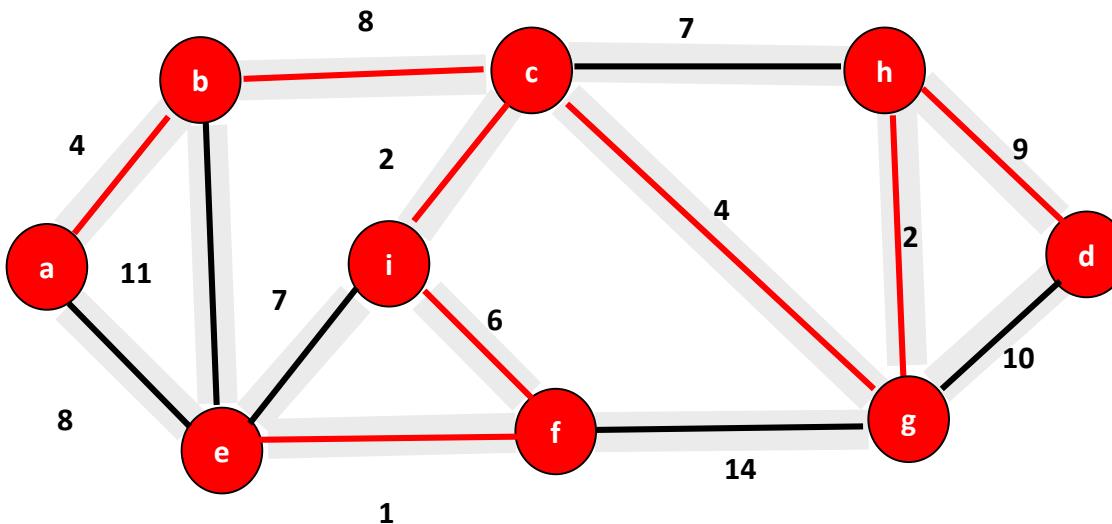


- g-c düğümleri en düşük olduğu için seçilip birleştirildi.  
 $M=\{26+4\}=30$



# Sollin'in Algoritması: Örnek

- e-f alt ağacı aynı mantıkla f-i düğümü ile birleştirilerek ağacın son hali çizilir.  $M=\{30+6\}=36$ .



# 13.Hafta

## En kısa yollar- I

En kısa yolların özellikleri

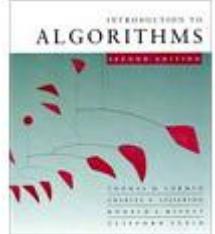
- Dijkstra algoritması
- Doğruluk
- Çözümleme
- Enine arama

# 13.Hafta

## En kısa yollar

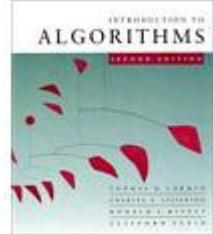
En kısa yolların özellikleri

- Dijkstra algoritması
- Doğruluk
- Çözümleme
- Enine arama



# Konular

- Ağırlıklandırılmış graflarda (weighted graphs) tek kaynaklı (single-source) en kısa yollar(shortest paths)
  - En kısa yol problemleri
  - En Kısa yol özellikleri ve gevşeme(relaxation)
  - Dijkstra algoritması
  - Bellman Ford algoritması

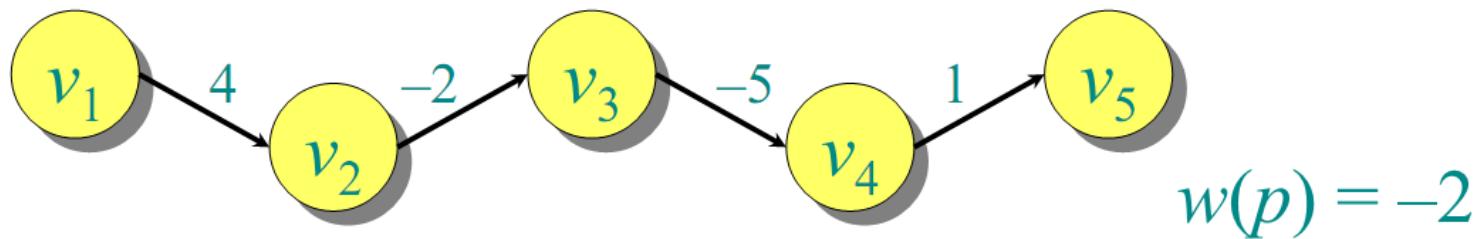


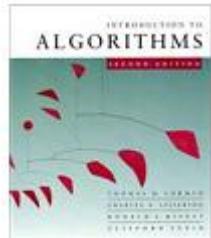
# Yönlü Grafiklerde yollar- En kısa yol

$w : E \rightarrow \mathbb{R}$  kenar-ağırlık fonksiyonu olan bir  $G = (V, E)$  yönlendirilmiş grafiği olduğunu düşünün. Yolun **ağırlığı** olan  $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$

$$w(p) = \sum_{i=1}^{k-1} w(v_i, v_{i+1}) \text{ olarak tanımlanır.}$$

## Örnek:





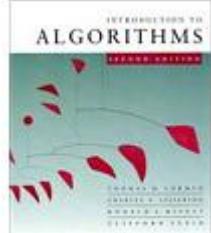
## En kısa yollar

$u'$  dan  $v'$  ye en kısa yol,  $u'$  dan  $v'$  ye en az ağırlıklı yoldur.

$u'$  dan  $v'$  ye en kısa yolun ağırlığı

$\delta(u, v) = \min\{w(p) \text{ olarak tanımlanır: } p, u \text{ dan } v \text{ ye bir yoldur}\}.$

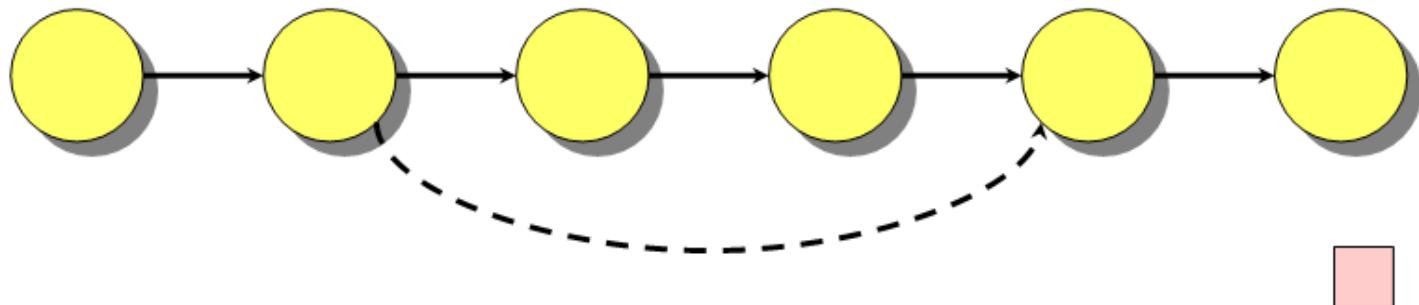
**Not:**  $u'$  dan  $v'$  bir yol yoksa  $\delta(u, v) = \infty$

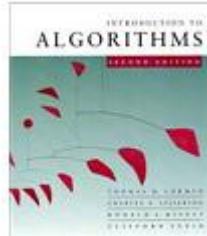


## En uygun altyapı

**Teorem.** En kısa yolun alt yolu, bir en kısa yoldur.

*Kanıt.* Kes ve yapıştır:



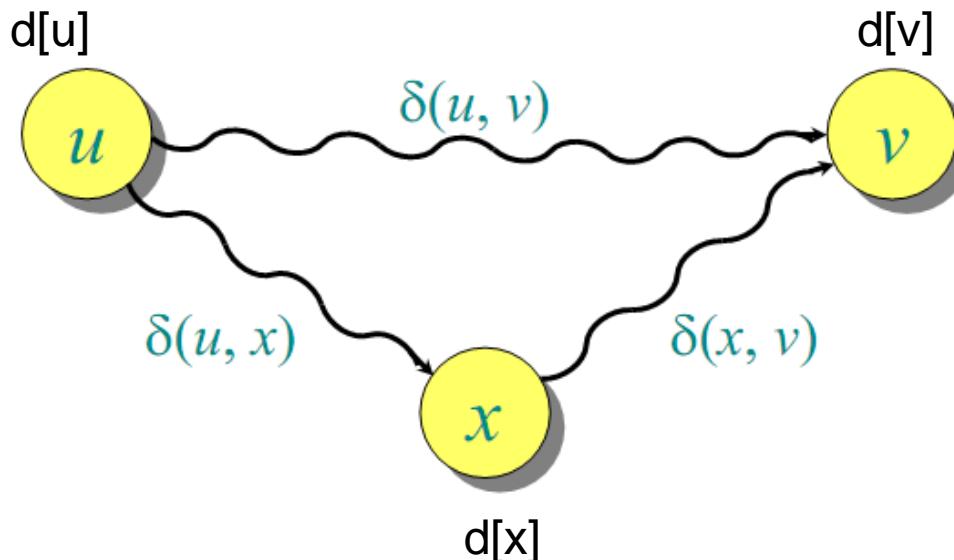


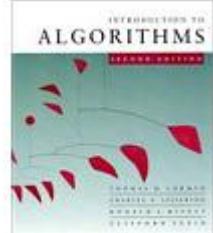
## Üçgen eşitsizliği

**Teorem.** Tüm  $u, v, x \in V$  ler için,

$$d[v] = \delta(u, v) \leq \delta(u, x) + \delta(x, v).$$

*Kanıt.*

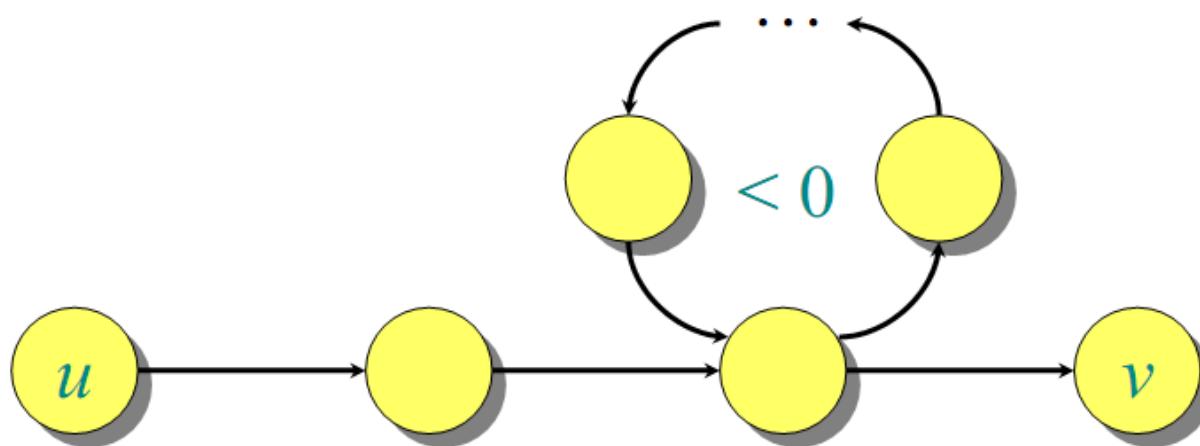


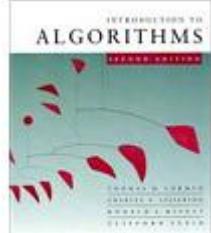


# En kısa yolların iyi tanımlanırlığı

Bir  $G$  grafiği negatif ağırlık döngüsü içeriyorsa, bazı en kısa yollar var olmayabilir.

**Örnek:**





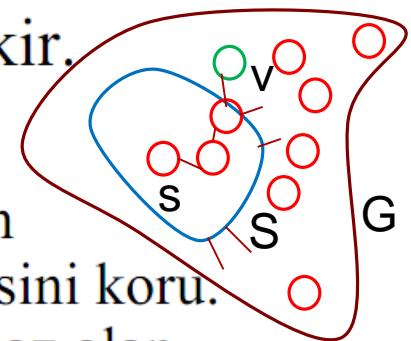
# Tek-kaynaklı (single-source) En kısa yollar

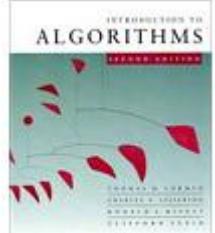
**Problem.**  $s \in V'$  deki verilen bir kaynak köşeden, tüm  $v \in V$  ler için,  $\delta(s, v)$  en kısa yol ağırlıklarını bulun.

Tüm  $w(u, v)$  kenar ağırlıkları *eksi* değilse bütün en kısa yol ağırlıklarının olması gereklidir.

**Fikir:** Açıgözlu.

1.  $s'$  den başlayan ve  $S$  içindeki tüm köşelere olan en kısa yol uzunlukları bilinen köşelerin kümesini koru.
2. Her adımda  $S'$  ye,  $s'$  ye olan uzaklık tahmini en az olan  $v \in V - S$  köşesine ekle.
3.  $v'$  ye bitişik köşelerin uzaklık tahminlerini güncelle.





# Ağırlıklı en kısa yol algoritmaları

## ○ Dijkstra Algoritması:

- Ağırlıklı ve yönlü graflar için geliştirilmiştir.
- Graf üzerindeki kenarların ağırlıkları 0 veya sıfırdan büyük sayılar olmalıdır.
- Negatif ağırlıklar için çalışmaz.

## ○ Bellman ve Ford Algoritması:

- Negatif ağırlıklı graflar için geliştirilmiştir.

## ○ Floyd-Warshall Algoritması

- Negatif ağırlıklı graflar için geliştirilmiştir.

## ○ Johnson Algoritması

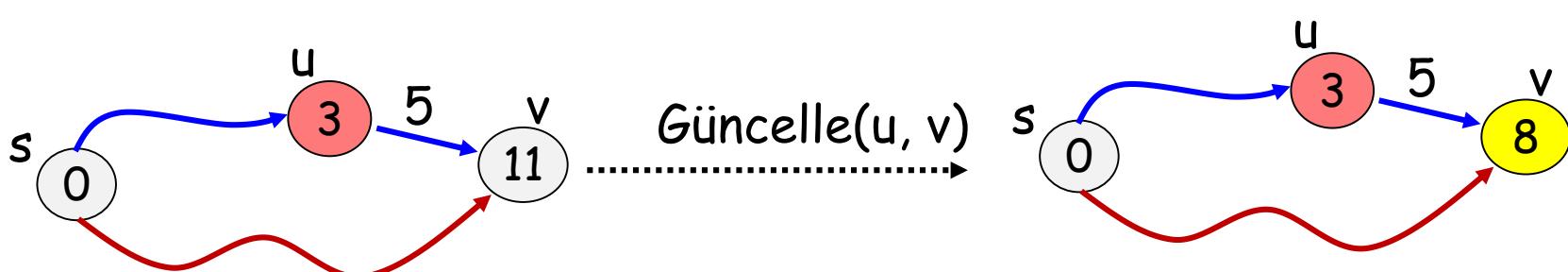
- Negatif ağırlıklı graflar için geliştirilmiştir.

# Dijkstra Algoritması

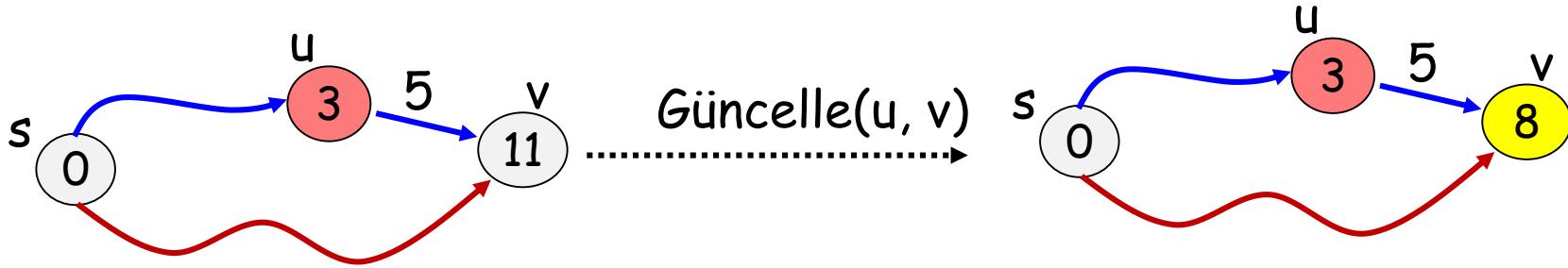
- Başlangıç olarak sadece başlangıç düğümünün en kısa yolu bilinir. (0 dır.)
- Tüm düğümlerin maliyeti bilinene kadar devam et.
  1. O anki bilinen düğümler içerisinde en iyi düğümü seç. (en az maliyetli düğümü seç, daha sonra bu düğümü bilinen düğümler kümese ekle)
  2. Seçilen düğümün komşularının maliyetlerini güncelle.

# Güncellemeye

- Adım-1 de seçilen düğüm **u** olsun.
- u düğümünün komşularının maliyetini güncelleme işlemi aşağıdaki şekilde yapılır.
  - s'den v'ye gitmek için iki yol vardır.
  - Kırmızı yol izlenebilir. Maliyet 11.
  - veya mavi yol izlenebilir. Önce s'den u'ya 3 maliyeti ile gidilir. Daha sonra (u, v) kenarı üzerinden 8 maliyetle v'ye ulaşılır.



# Güncelleme - Kaba Kod

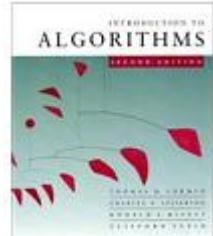


```

Guncelle(u, v){
 if (maliyet[u] + w(u, v) < maliyet[v]){
 maliyet[v] = maliyet[u] + w(u, v); // U üzerinden yol daha kısa ise
 pred[v] = u; // Evet! Güncelle
 }
}

```

# Dijkstra algoritması



$d[s] \leftarrow 0$

(her) **for** each  $v \in V - \{s\}$  (için)

(yap) **do**  $d[v] \leftarrow \infty$

$S \leftarrow \emptyset$

$Q \leftarrow V$       ▷  $Q$ ,  $V - S$ 'yi koruyan bir öncelikli sıradır.

**while**  $Q \neq \emptyset$  (-iken)

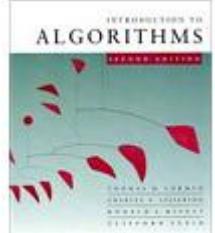
(yap) **do**  $u \leftarrow \text{EXTRACT-MIN}(Q)$  (en azı çıkar)

$S \leftarrow S \cup \{u\}$

(her) **for** each  $v \in Adj[u]$  (için)

(yap eğer) **do if**  $d[v] > d[u] + w(u, v)$

(sonra) **then**  $d[v] \leftarrow d[u] + w(u, v)$



# Dijkstra algoritması

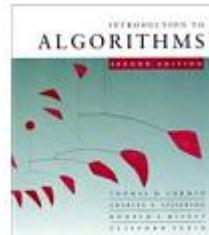
$d[s] \leftarrow 0$   
 (her) **for** each  $v \in V - \{s\}$  (için)  
 (yap) **do**  $d[v] \leftarrow \infty$   
 $S \leftarrow \emptyset$   
 $Q \leftarrow V$       ▷  $Q$ ,  $V - S$ 'yi koruyan bir öncelikli sıradır.

**while**  $Q \neq \emptyset$  (-iken)  
 (yap) **do**  $u \leftarrow \text{EXTRACT-MIN}(Q)$  (en azı çıkar)  
 $S \leftarrow S \cup \{u\}$

(her) **for** each  $v \in \text{Adj}[u]$  (için)  
 (yap eğer) **do if**  $d[v] > d[u] + w(u, v)$   
 (sonra) **then**  $d[v] \leftarrow d[u] + w(u, v)$

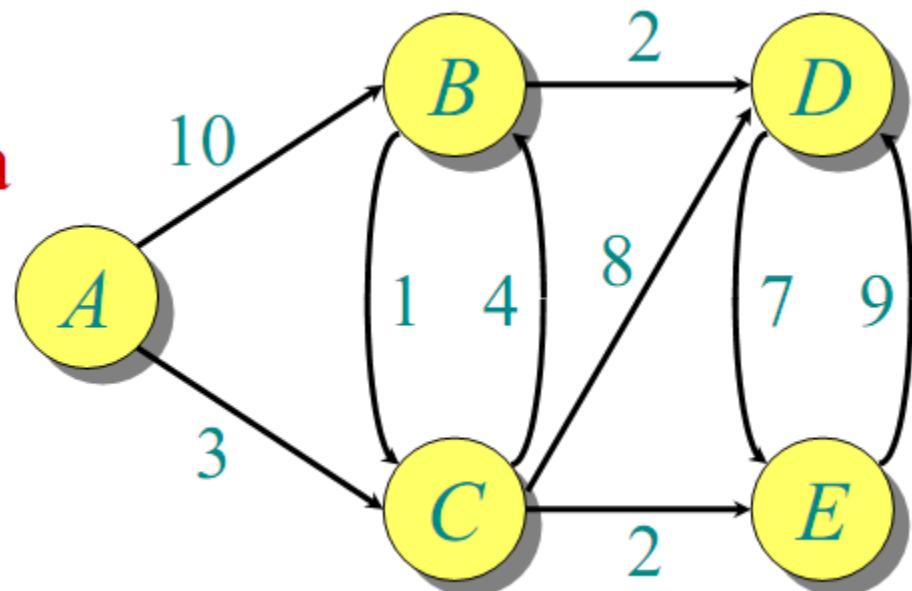
*Gevşeme  
Adımı*

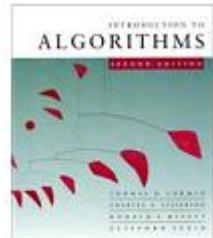
Implicit DECREASE-KEY(azaltılmış anahtar)



## Dijkstra algoritmasına örnek

Eksi olmayan  
kenar ağırlıklarıyla  
grafik:

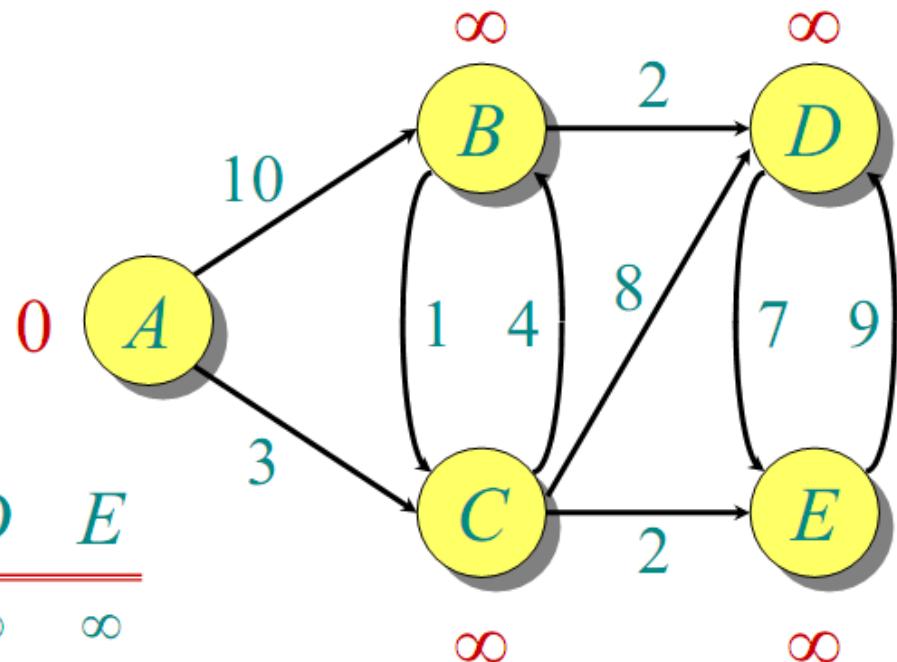




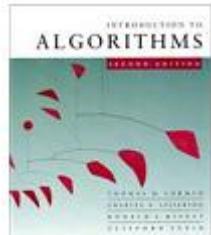
## Dijkstra algoritmasına örnek

İlklenidleme:

$$Q: \frac{A \ B \ C \ D \ E}{0 \ \infty \ \infty \ \infty \ \infty}$$



$$S: \{\}$$

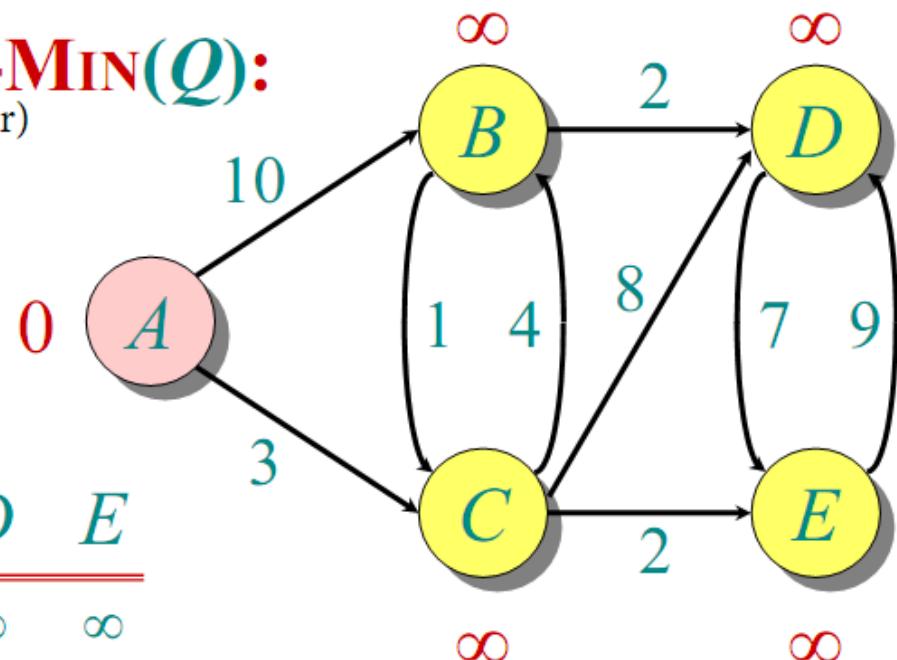


## Dijkstra algoritmasına örnek

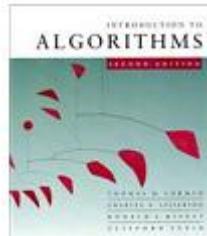
$"A" \leftarrow \text{EXTRACT-MIN}(Q)$ :

(en azı çıkar)

| $Q:$ | $A$ | $B$      | $C$      | $D$      | $E$      |
|------|-----|----------|----------|----------|----------|
|      | 0   | $\infty$ | $\infty$ | $\infty$ | $\infty$ |

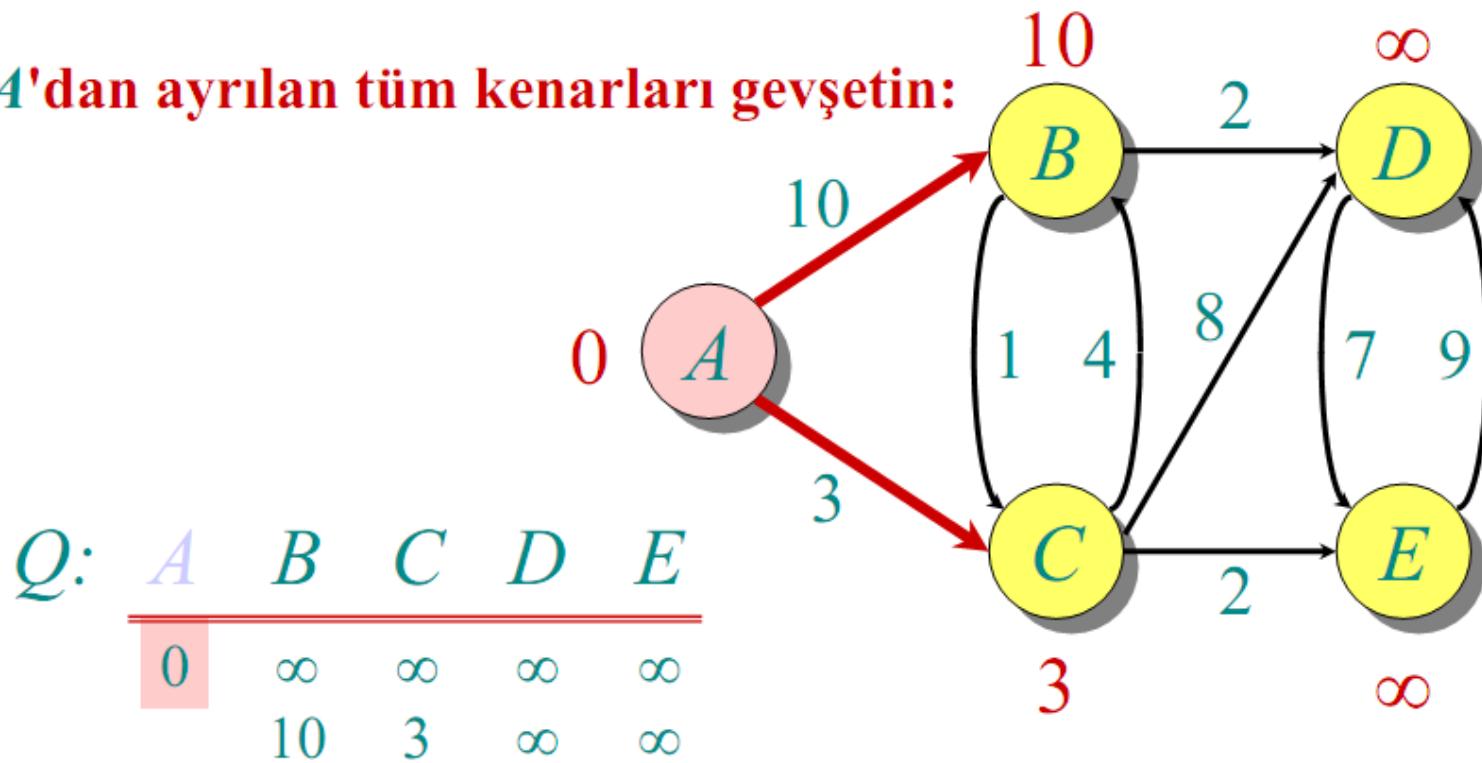


$S: \{ A \}$

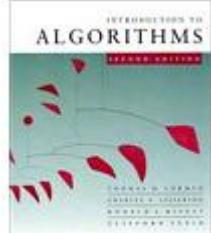


## Dijkstra algoritmasına örnek

*A'dan ayrılan tüm kenarları gevşetin:*



$S: \{ A \}$

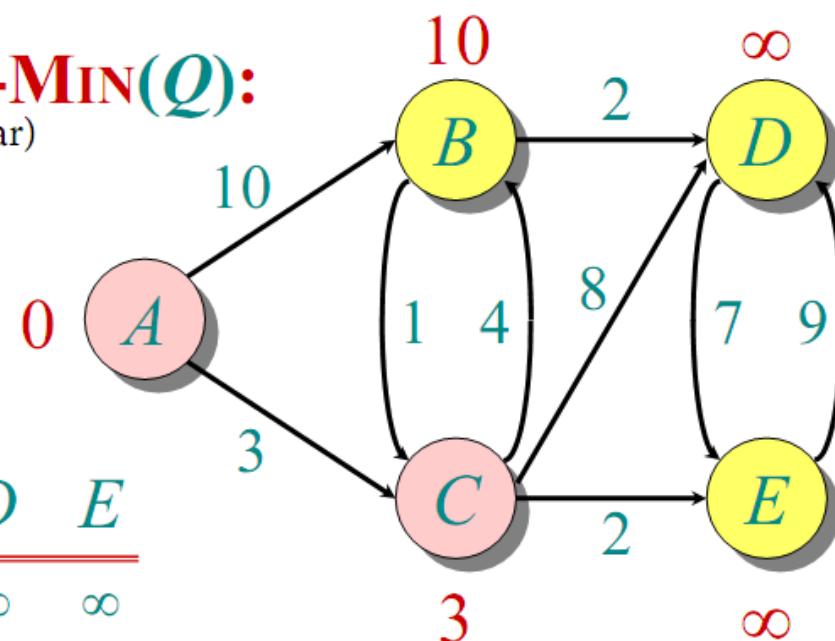


## Dijkstra algoritmasına örnek

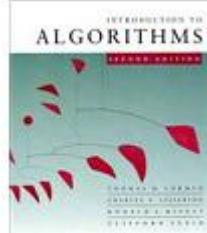
$C \leftarrow \text{EXTRACT-MIN}(Q)$ :

(en azı çıkar)

| $Q:$ | $A$ | $B$      | $C$      | $D$      | $E$      |
|------|-----|----------|----------|----------|----------|
|      | 0   | $\infty$ | $\infty$ | $\infty$ | $\infty$ |

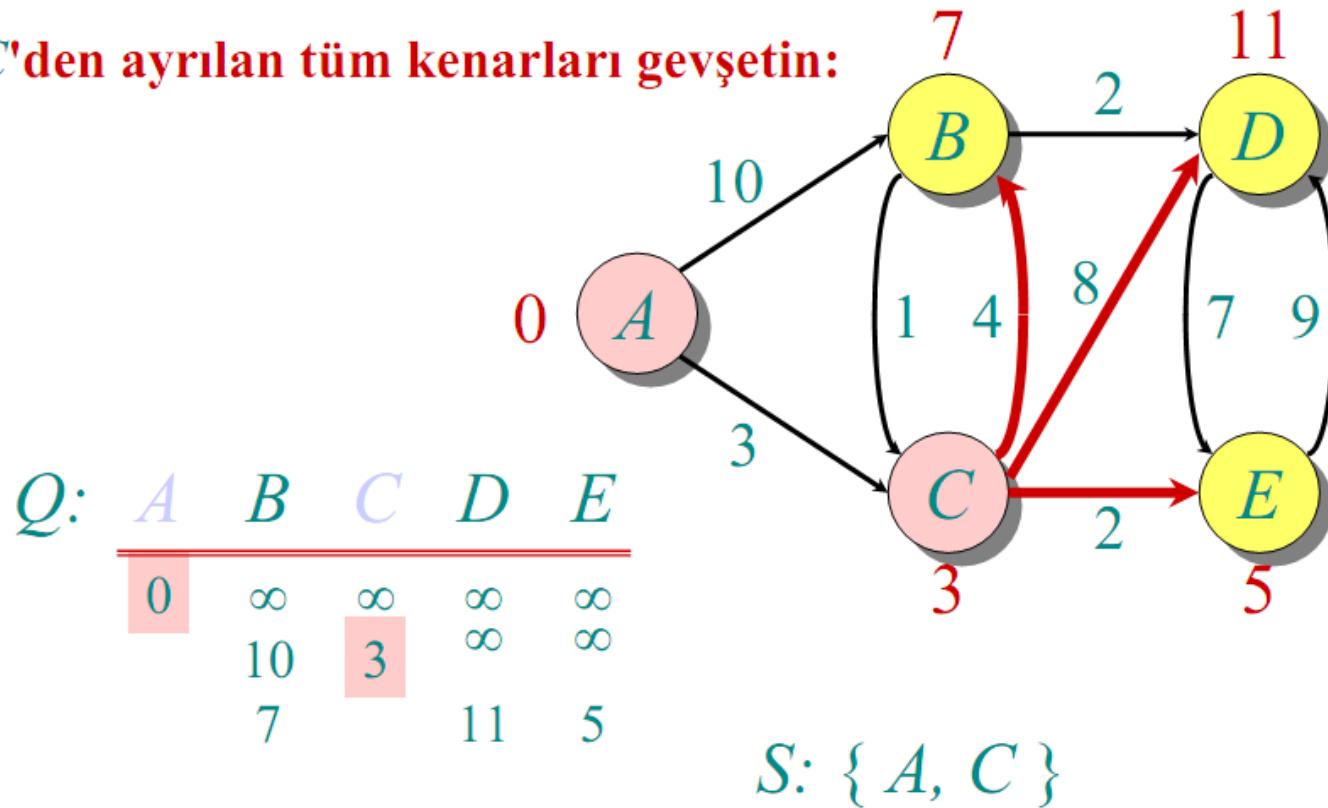


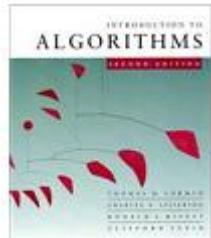
$S: \{ A, C \}$



## Dijkstra algoritmasına örnek

$C$ 'den ayrılan tüm kenarları gevşetin:



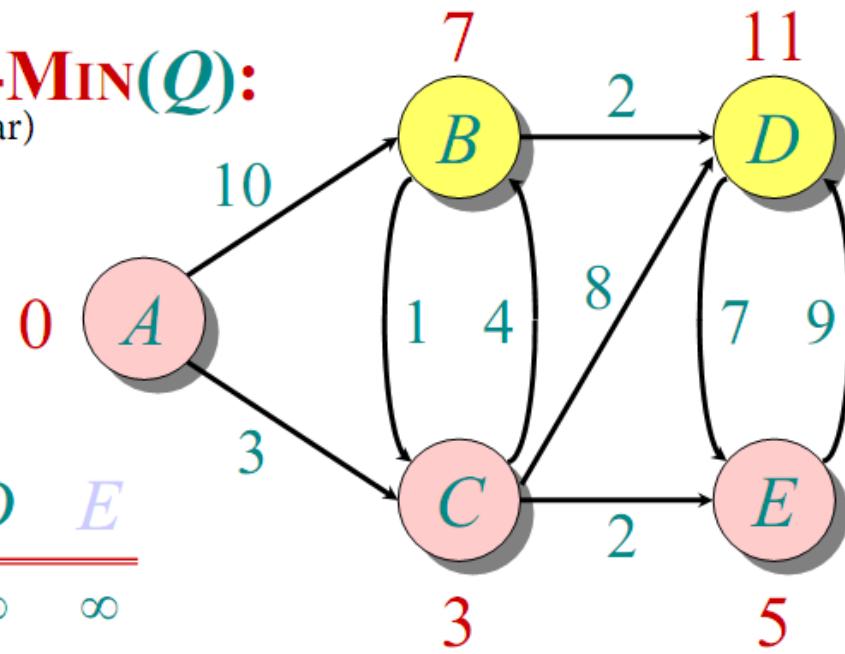


## Dijkstra algoritmasına örnek

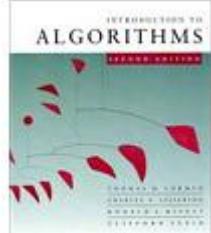
$E \leftarrow \text{EXTRACT-MIN}(Q)$ :

(en azı çıkar)

| $Q:$ | $A$ | $B$      | $C$      | $D$      | $E$      |
|------|-----|----------|----------|----------|----------|
|      | 0   | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
|      | 10  | 3        | $\infty$ | $\infty$ |          |
|      | 7   |          | 11       | 5        |          |



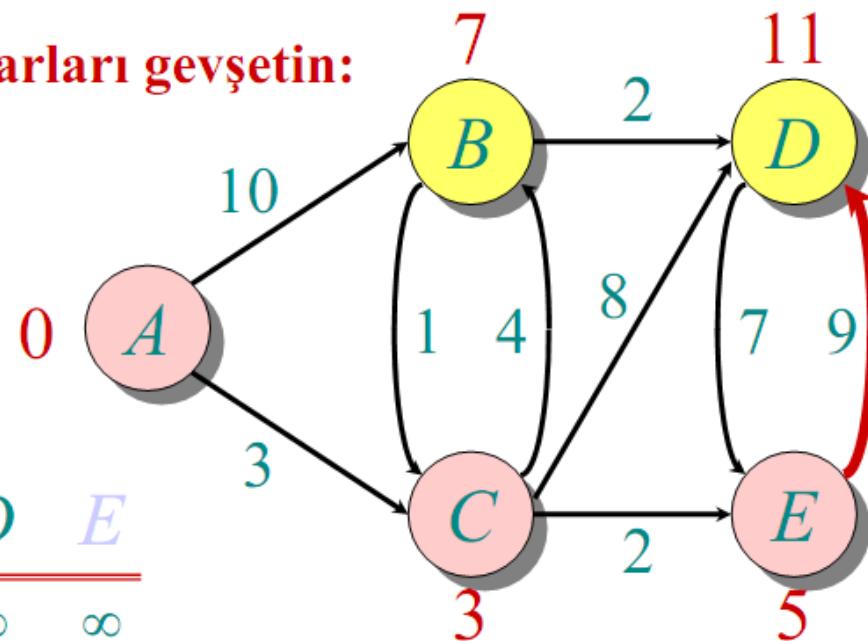
$S: \{ A, C, E \}$



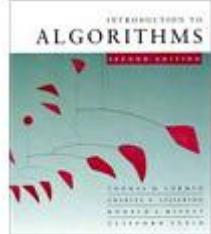
## Dijkstra algoritmasına örnek

*E'den ayrılan tüm kenarları gevşetin:*

| <i>Q:</i> | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|-----------|----------|----------|----------|----------|----------|
| 0         | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 10        |          | 3        |          | $\infty$ | $\infty$ |
| 7         |          |          | 11       |          | 5        |
| 7         |          |          | 11       |          |          |



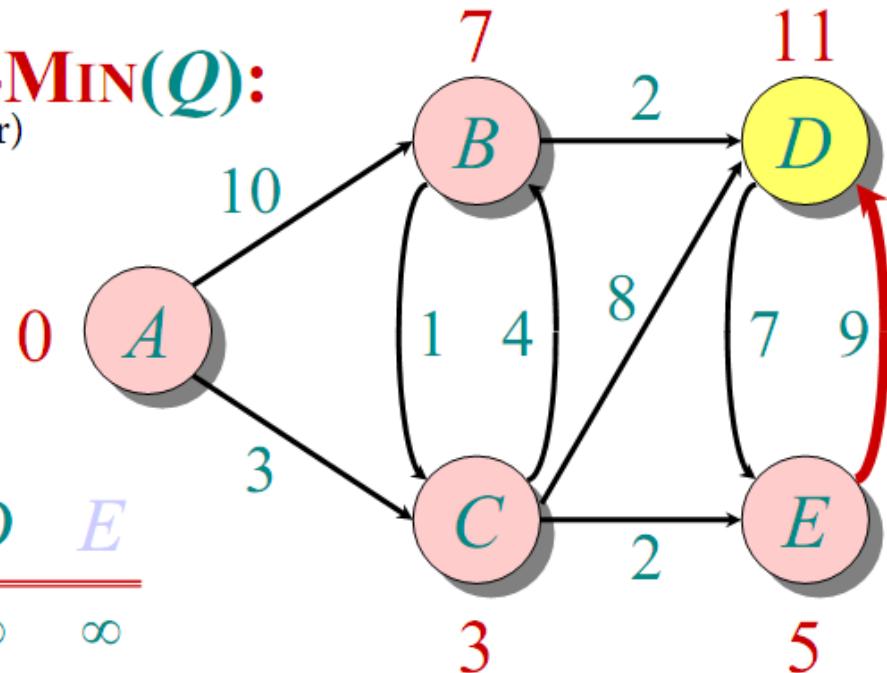
*S:* { *A, C, E* }



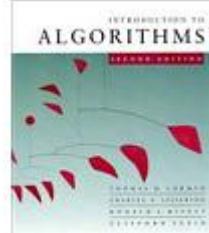
## Dijkstra algoritmasına örnek

**“B”  $\leftarrow \text{EXTRACT-MIN}(Q)$ :**  
 (en azı çıkar)

| $A$ | $B$      | $C$      | $D$      | $E$      |
|-----|----------|----------|----------|----------|
| 0   | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 10  | 3        | $\infty$ | $\infty$ |          |
| 7   | 7        | 11       | 5        | 11       |

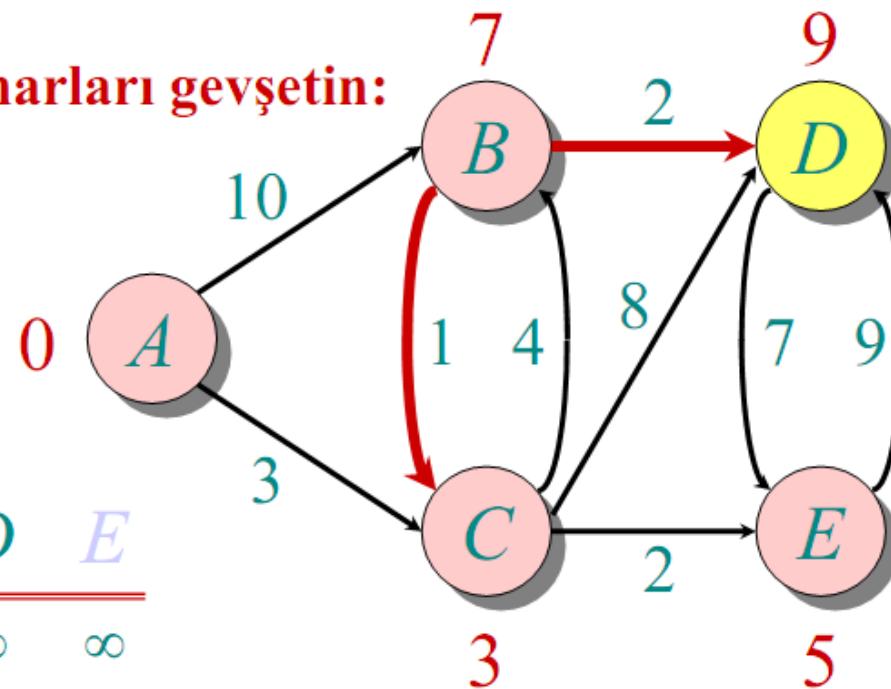


$S: \{ A, C, E, B \}$



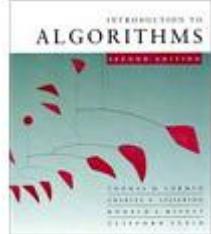
## Dijkstra algoritmasına örnek

*B'den ayrılan tüm kenarları gevşetin:*



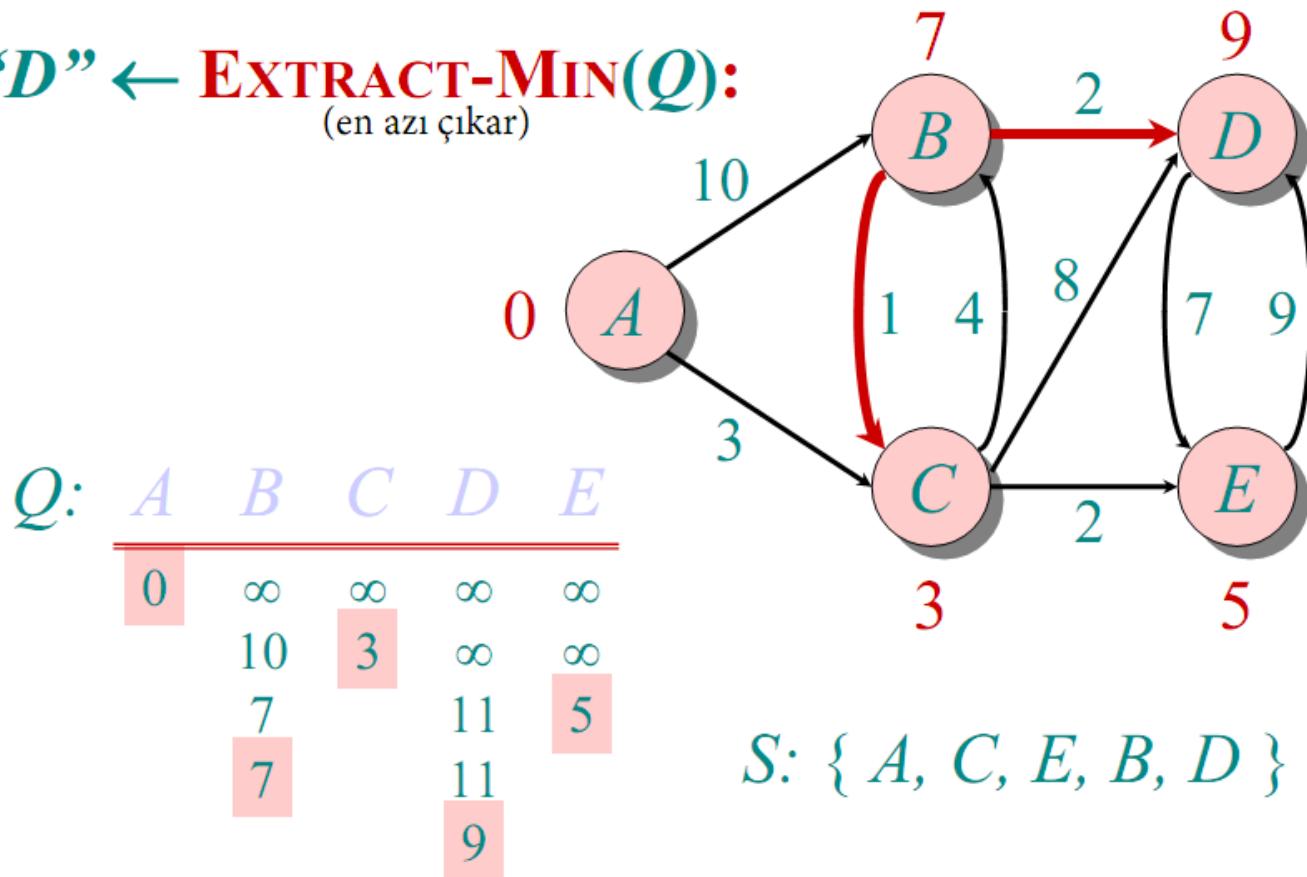
| $Q:$ | $A$ | $B$      | $C$      | $D$      | $E$      |
|------|-----|----------|----------|----------|----------|
|      | 0   | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
|      | 10  | 3        | $\infty$ | $\infty$ |          |
|      | 7   | 7        | 11       | 5        |          |
|      |     |          | 11       |          |          |
|      |     |          | 9        |          |          |

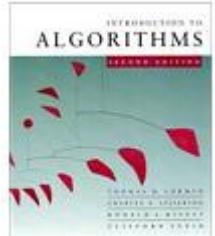
$S: \{ A, C, E, B \}$



## Dijkstra algoritmasına örnek

**“D”  $\leftarrow \text{EXTRACT-MIN}(Q)$ :**  
 (en azı çıkar)





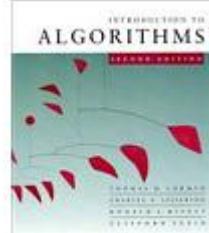
## Dijkstra' nın çözümlemesi (analizi)

$|V|$  kere { (-iken) **while**  $Q \neq \emptyset$   
 (yap) **do**  $u \leftarrow \text{EXTRACT-MIN}(Q)$  (en azı çıkar)  
 $S \leftarrow S \cup \{u\}$   
 ( $u$ ) derecesi { **for each**  $v \in \text{Adj}[u]$  (için)  
 kere { (her) **do if**  $d[v] > d[u] + w(u, v)$   
 (yap eğer) **then**  $d[v] \leftarrow d[u] + w(u, v)$   
 (sonra) }

Tokalaşma önkuramı  $\Rightarrow \Theta(E)$  implicit (gizli) DECREASE-KEY's.  
 (azaltılmış anahtar)

Time(süre) =  $\Theta(V \cdot T_{\text{EXTRACT-MIN}} + E \cdot T_{\text{DECREASE-KEY}})$

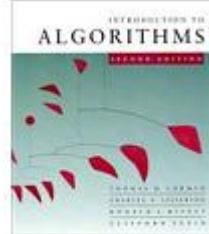
**Not:** Prim'in en az kapsayan ağaç algoritmasının çözümlemesinde de aynı formül.



# Dijkstra' nın çözümlemesi (analizi)

$$\text{Süre} = \Theta(V) \cdot T_{\substack{\text{EXTRACT-MIN} \\ (\text{en azı çıkar})}} + \Theta(E) \cdot T_{\substack{\text{DECREASE-KEY} \\ (\text{azaltılmış anahtar})}}$$

| $Q$                 | $T_{\text{EXTRACT-MIN}}$          | $T_{\text{DECREASE-KEY}}$     | Toplam                            |
|---------------------|-----------------------------------|-------------------------------|-----------------------------------|
| dizilim             | $O(V)$                            | $O(1)$                        | $O(V^2)$                          |
| ikili<br>yığın      | $O(\lg V)$                        | $O(\lg V)$                    | $O(E \lg V)$<br>$O((V+E)\log V)$  |
| Fibonacci<br>yığını | $O(\lg V)$<br>amortize<br>edilmiş | $O(1)$<br>amortize<br>edilmiş | $O(E + V \lg V)$<br>en kötü durum |



# Doğruluk – Bölüm I

**Önkuram.**  $d[s] \leftarrow 0$  ve  $d[v] \leftarrow \infty$ ' yi tüm  $v \in V - \{s\}$ ' ler için ilkendirme  $d[v] \geq \delta(s, v)$ ' yi sağlar- tüm  $v \in V$ ' ler için: Ve bu değişmez dizideki tüm gevşetme adımlarında korunur.

**Kanıt.** Sunun olmadığını düşünün.  $v$ ,  $d[v] < \delta(s, v)$ ' deki ilk köşe olsun ve  $u$ ' da  $d[v]$ ' yi değiştiren ilk köşe olsun:

$$d[v] = d[u] + w(u, v). \text{ O zaman,}$$

$$d[v] < \delta(s, v)$$

kabul

$$\leq \delta(s, u) + \delta(u, v)$$

üçgen eşitsizliği

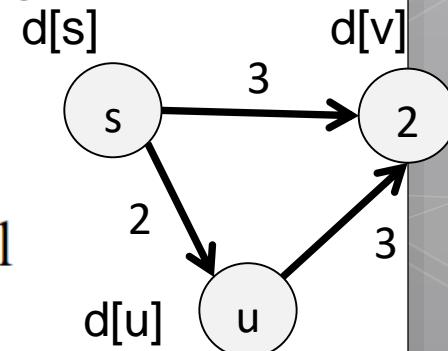
$$\leq \delta(s, u) + w(u, v)$$

kısa yol  $\leq$  özel yol

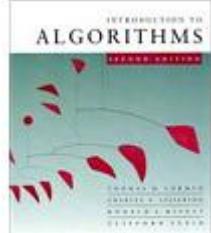
$$\leq d[u] + w(u, v)$$

$v$  ilk ihlal

Çelişki.



Genişletmede bir hata olduğunu gösterir

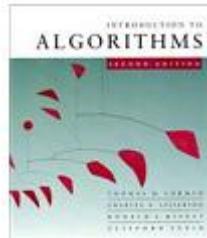


## Doğruluk – Bölüm II

**Ön kuram.**  $u$ ,  $s'$  den  $v'$  ye en kısa yolda  $v'$  nin atası olsun. O durumda, eğer  $d[u] = \delta(s, u)$  ve kenar  $(u, v)$  gevşetilmişse, gevşemeden sonra elimizde  $d[v] = \delta(s, v)$  olur.

**Kanıt.**  $\delta(s, v) = \delta(s, u) + w(u, v)$  olduğuna dikkat edin. Gevşetmeden önce  $d[v] > \delta(s, v)$  olduğunu farzedin. (Diğer türlü, bitirmiştik.) Sonra,  $d[v] > d[u] + w(u, v)$  testi başarılı, çünkü  $d[v] > \delta(s, v) = \delta(s, u) + w(u, v) = d[u] + w(u, v)$  ve algoritma  $d[v] = d[u] + w(u, v) = \delta(s, v)$ ' yi ayarlar.

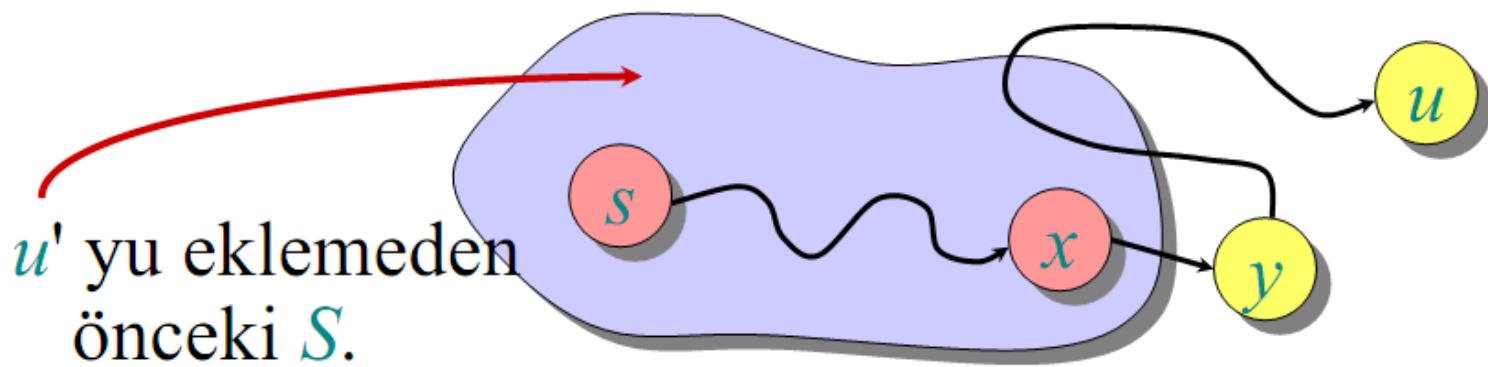


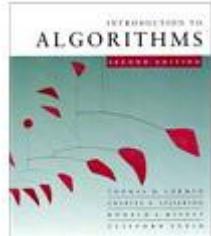


## Doğruluk – Bölüm III

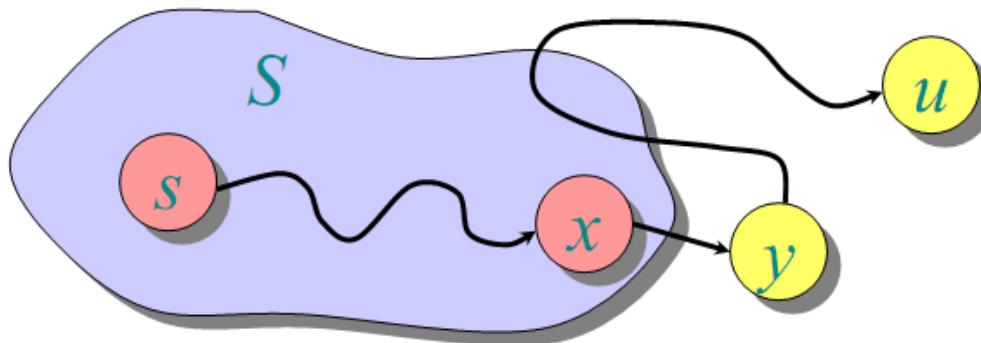
**Teorem.** Dijkstra algoritması tüm  $v \in V$  için  $d[v] = \delta(s, v)$  ile sonlanır.

*Kanıt.*  $v$ ,  $S'$  ye eklenirken, her  $v \in V$  için  $d[v] = \delta(s, v)$  olduğunu göstermek yeterlidir. Her  $d[u] > \delta(s, u)$  için  $u'$  nun  $S'$  ye eklenen ilk köşe olduğunu düşünün.  $y, s'$  den  $u'$  ya en kısa yol boyunca  $V - S$  de ilk köşe olsun,  $x$  de onun atası olsun:





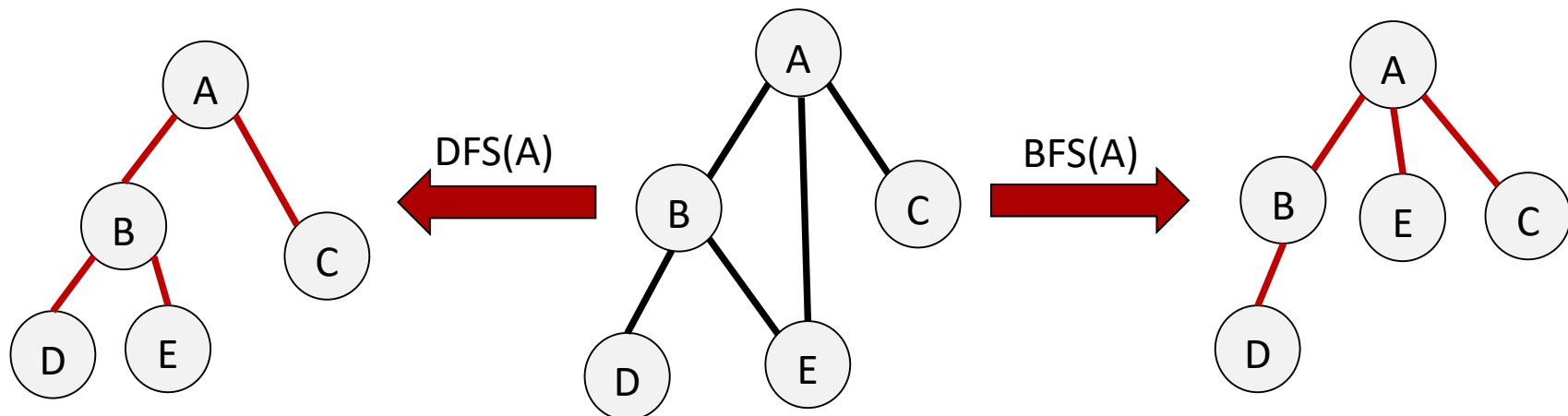
## Doğruluk – Bölüm III

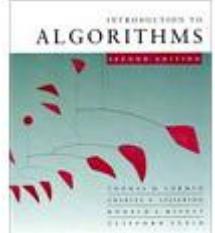


Eğer  $u$ , belirlenen değişmezi ihlal eden ilk köşe ise  $d[x] = \delta(s, x)$  elde ederiz.  $x$ ,  $S$ ' ye eklendiğinde kenar  $(x, y)$  gevşetildi ki bu  $d[y] = \delta(s, y) \leq \delta(s, u) < d[u]$  anlamına gelir. Fakat, bizim  $u$  seçimimizle  $d[u] \leq d[y]$  olur. Çelişki. ■

# MST Hesaplama – Ağırlıksız Graf

- Graf ağırlıksızsa veya tüm kenarların ağırlıkları eşit ise MST nasıl bulunur?
  - BSF veya DSF çalıştırın oluşan ağaç MST'dir





# Ağırlıklandırılmamış grafikler

Tüm  $(u, v) \in E$ 'ler için  $w(u, v) = 1$  olduğunu farzedin  
Dijkstra algoritması geliştirilebilir mi?

- Bir öncelik sırası yerine basit FIFO sırası kullanın.

## *Enine arama*

(-iken)**while**  $Q \neq \emptyset$

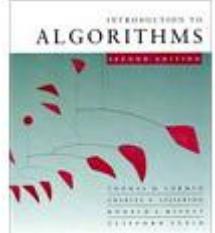
(yap) **do**  $u \leftarrow \text{DEQUEUE}(Q)$  (sıradan çıkar)

(her) **for each**  $v \in \text{Adj}[u]$  (için)

(yap eğer) **do if**  $d[v] = \infty$

(sonra) **then**  $d[v] \leftarrow d[u] + 1$

**ENQUEUE**( $Q, v$ ) (sıraya ekle)



## Ağırlıklandırılmamış grafikler

Tüm  $(u, v) \in E$ 'ler için  $w(u, v) = 1$  olduğunu farzedin.  
Dijkstra algoritması geliştirilebilir mi?

- Bir öncelik sırası yerine basit FIFO sırası kullanın.

### *Sığ öncelikli arama*

(-iken) **while**  $Q \neq \emptyset$

(yap) **do**  $u \leftarrow \text{DEQUEUE}(Q)$  (sıradan çıkar)

(her) **for each**  $v \in \text{Adj}[u]$  (için)

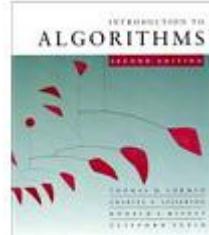
(yap eğer) **do if**  $d[v] = \infty$

(sonra) **then**  $d[v] \leftarrow d[u] + 1$

$\text{ENQUEUE}(Q, v)$  (sıraya ekle)

**Çözümleme:** Time(süre)=  $O(V + E)$ .

# (BFS)Enine arama'nın doğruluğu



(-iken) **while**  $Q \neq \emptyset$

(yap) **do**  $u \leftarrow \text{DEQUEUE}(Q)$  (sıradan çıkar)

(her) **for each**  $v \in \text{Adj}[u]$  (için)

(yap eğer) **do if**  $d[v] = \infty$

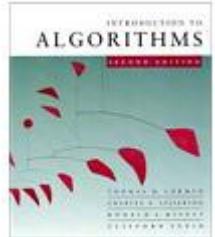
(sonra) **then**  $d[v] \leftarrow d[u] + 1$

$\text{ENQUEUE}(Q, v)$  (sıraya ekle)

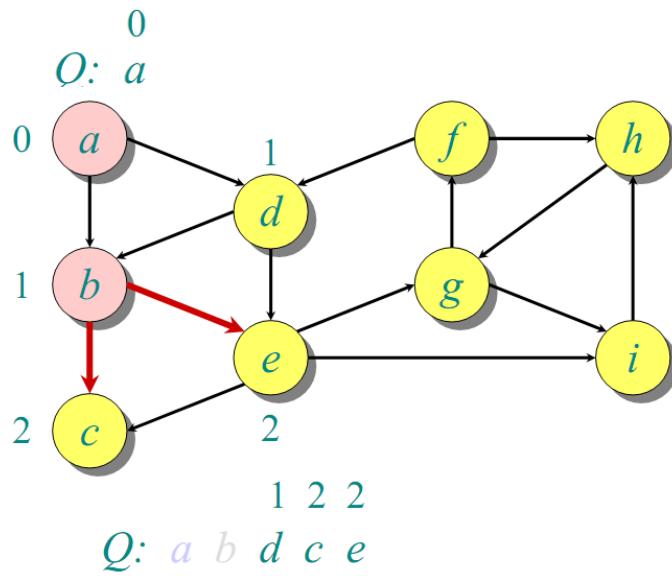
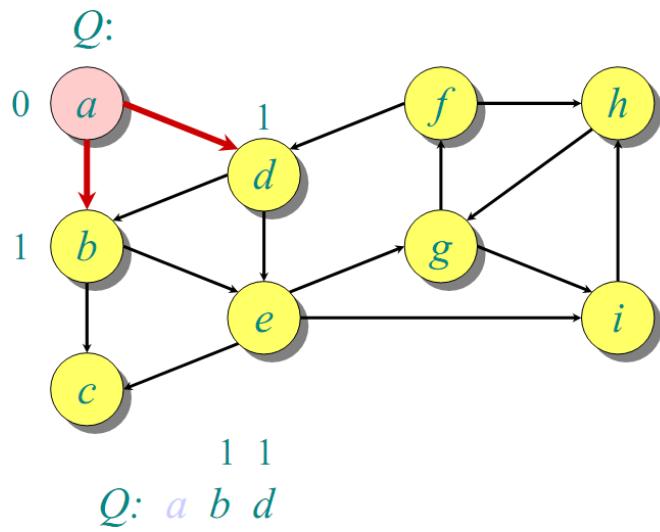
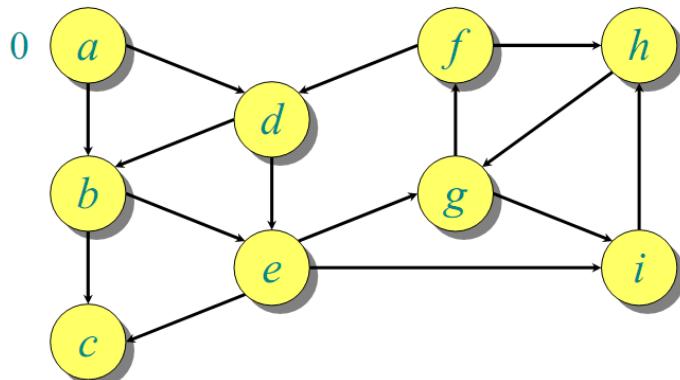
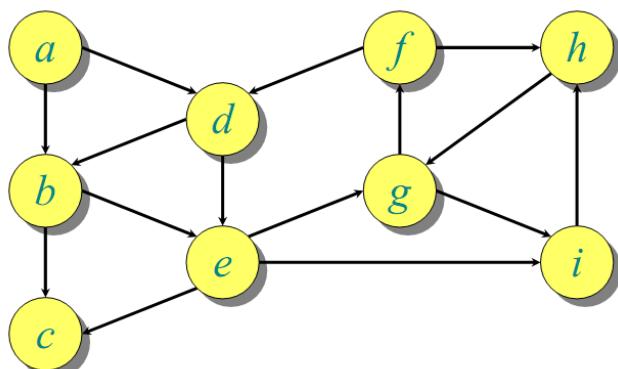
## Anahtar fikir:

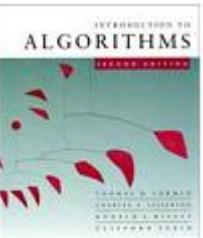
Enine aramadaki FIFO  $Q$ , Dijkstra' nın öncelikli sıralamasındaki kuyruk  $Q'$  yu taklit eder.

- **Değişmez:**  $Q'$  da  $v, u$ ' dan sonra gelirse bu  $d[v] = d[u]$  ya da  $d[v] = d[u] + 1$  anlamına gelir.

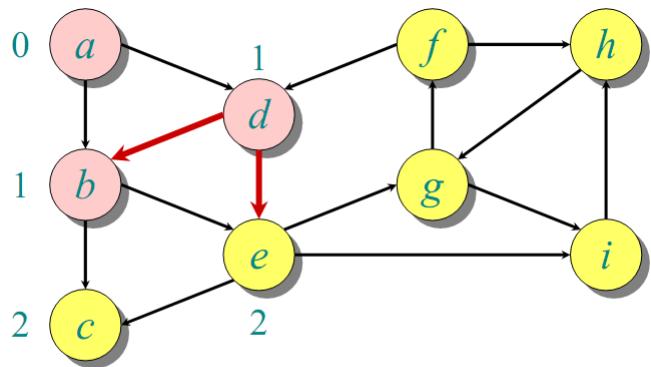


## Enine arama için örnek

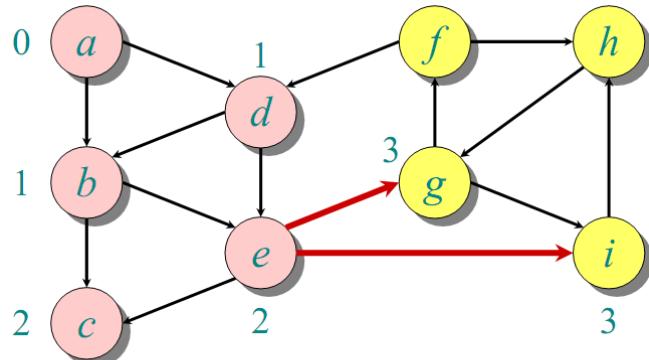
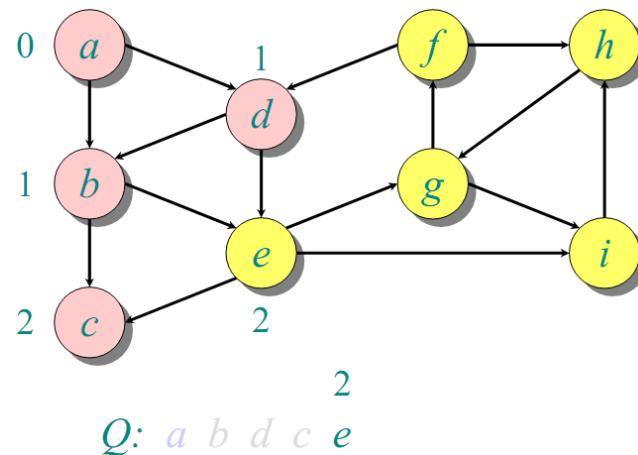




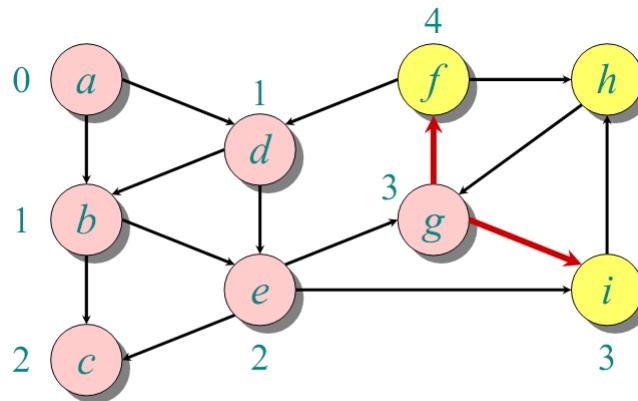
## Enine arama için örnek



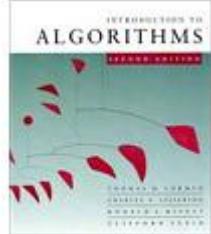
$Q: \text{a } b \text{ } d \text{ } c \text{ } e$



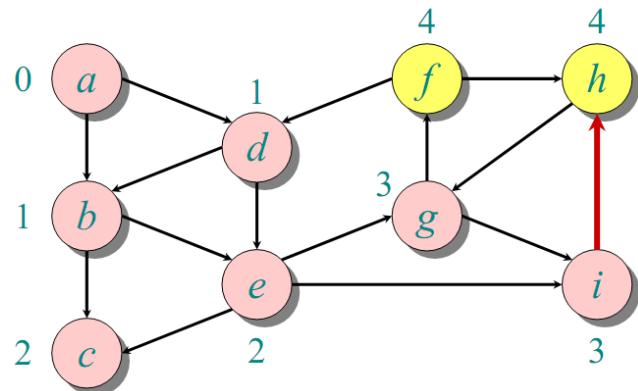
$Q: \text{a } b \text{ } d \text{ } c \text{ } e \text{ } g \text{ } i$



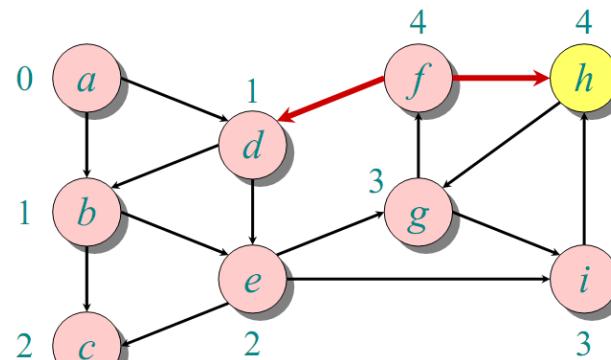
$Q: \text{a } b \text{ } d \text{ } c \text{ } e \text{ } g \text{ } i \text{ } f$



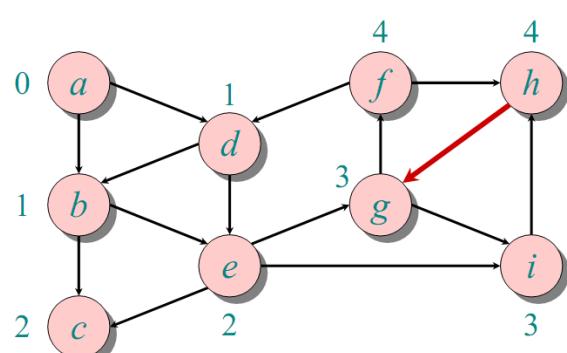
## Enine arama için örnek



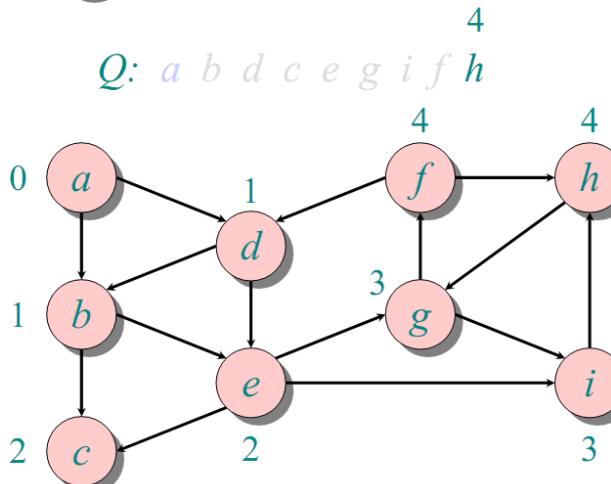
*Q:* *a b d c e g i f h*



*Q:* *a b d c e g i f h*



*Q:* *a b d c e g i f h*



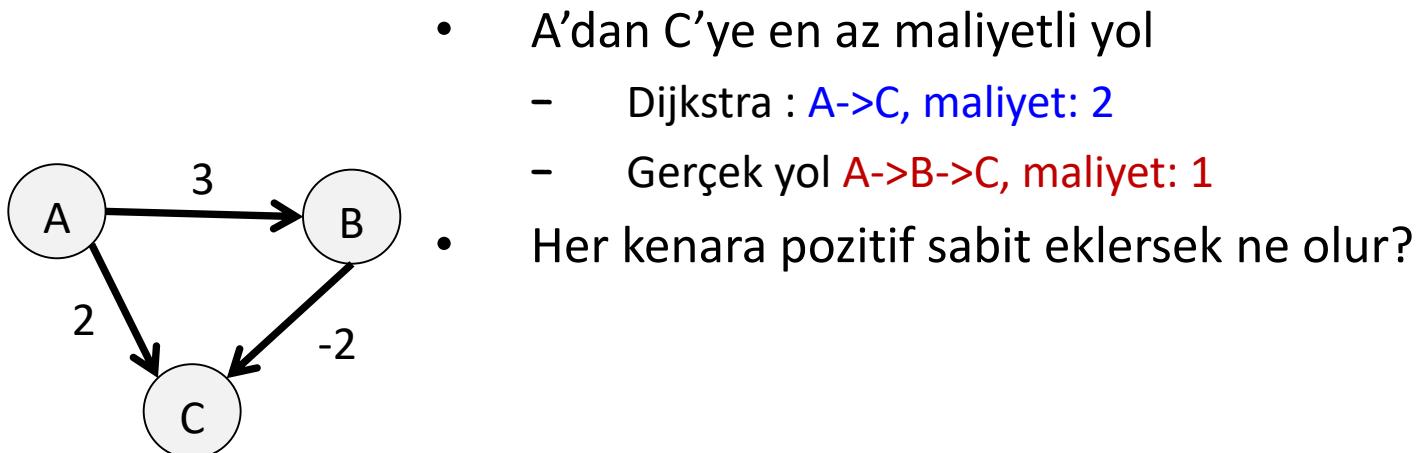
*Q:* *a b d c e g i f h*

## Negatif Ağırlıklı En Kısa Yollar

- Doğruluk
- Çözümleme

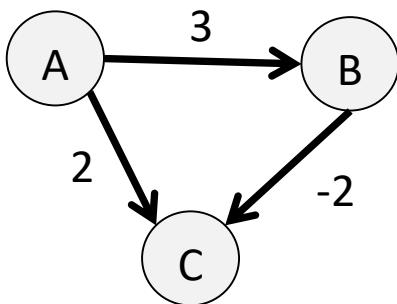
# Negatif Ağırlıklı Dijkstra

- Eğer kenarların ağırlıkları negatif ise Dijkstra algoritması en az maliyetli yolu bulmada başarısız oluyor.
- Bunun sebebi eksi (-) değerdeki kenarın sürekli olarak mevcut durumdan daha iyi bir sonuç üretmesi ve algoritmanın hiçbir zaman için kararlı hale gelememesidir.

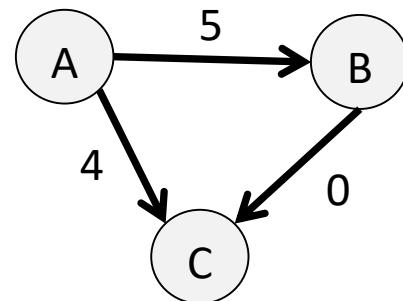


# Negatif Ağırlıklı Dijkstra

- Her kenara pozitif sabit eklersek ne olur?



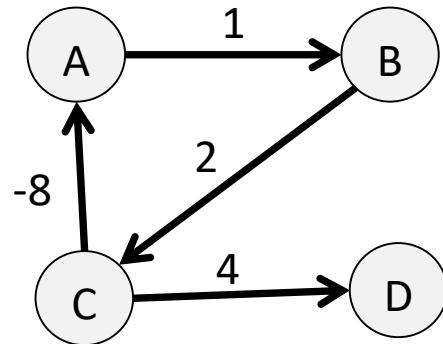
Her kenara  
2 ekle



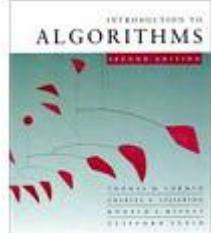
- A'dan C'ye en az maliyetli yol
  - Dijkstra: A->C
  - Gerçek Yol: A->B->C

# Negatif Maliyetli Çember

- Eğer graf negatif maliyetli çember içeriyorsa, en az maliyetli yol tanımlanamaz.



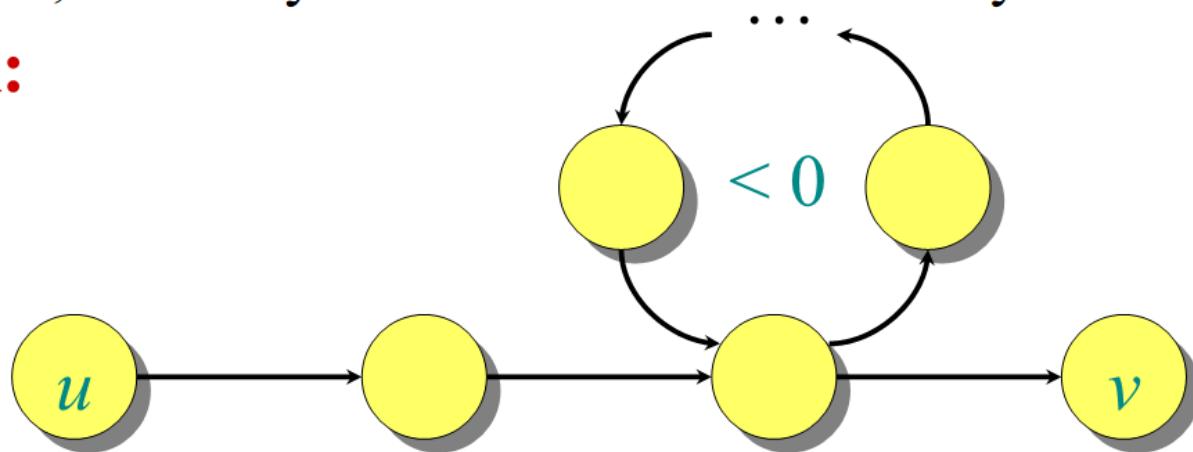
- A'dan D'ye en az maliyetli yol nedir?
  - veya B'den C'ye?



## Negatif-ağırlık çevrimleri

**Hatırlatma:** Eğer grafik  $G = (V, E)$  negatif ağırlık çevrimi içeriyorsa, en kısa yollardan bazıları bulunmayabilir.

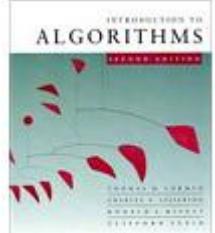
**Örnek:**



**Bellman-Ford algoritması:** Bir  $s \in V$  kaynağından tüm  $v \in V$  lere bütün kısa yol uzunluklarını bulur ya da bir negatif ağırlık çevrimi olduğunu saptar.

# Bellman-Ford Algoritması

- Ana mantık:
  - Her düğüm için bir uzaklık tahmini oluşturulur.
  - Başlangıç olarak maliyet( $s$ )=0 diğer düğümler için maliyet( $u$ )=  $\infty$  olarak atanır.
  - En az maliyetli yol hesaplanana kadar tüm kenarlar üzerinden güncelleme yapılır.
- Algoritma ayrıca grafin negatif maliyetli kenarının olup olmadığını da bulur.
- Dijkstra'nın algoritmasına göre daha yavaş çalışır.
  - (Dijkstra algoritması negatif kenarlarda kullanılmaz.)



## Bellman-Ford algoritması

$d[s] \leftarrow 0$

**for each**(her bir)  $v \in V - \{s\}$ (için)  
**do(yap)**  $d[v] \leftarrow \infty$

**for**(için)  $i \leftarrow 1$  **to**  $|V| - 1$  ' ( e )

**do for each edge**(her kenar için yap)  $(u, v) \in E$

**do(yap) if**(eğer)  $d[v] > d[u] + w(u, v)$

**then(sonra)**  $d[v] \leftarrow d[u] + w(u, v)$

} ilklandırmalı

} **Gevsetme adımı**

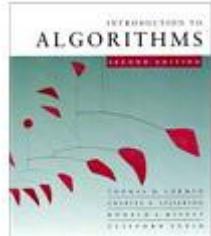
(her) **for each edge**  $(u, v) \in E$  (kenarı için)

(yap eğer) **do if**  $d[v] > d[u] + w(u, v)$

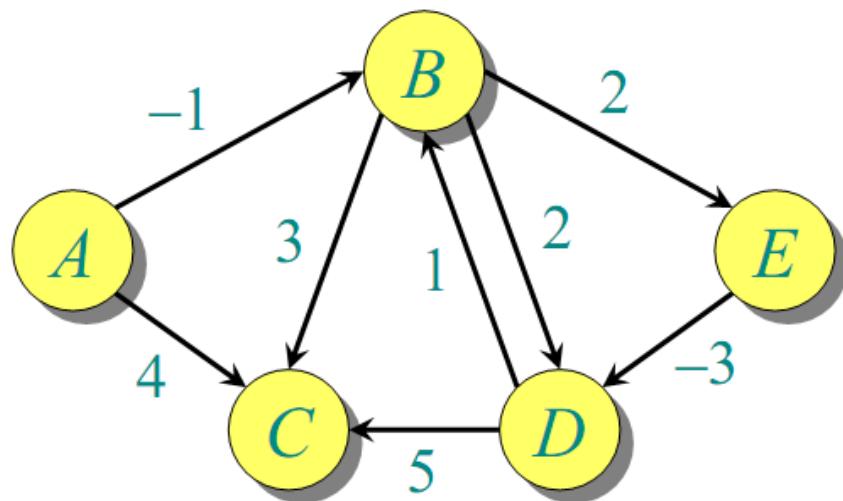
**sonra** bunu negatif ağırlık çevrimi var diyerek raporla

Sonunda,  $d[v] = \delta(s, v)$ , negatif ağırlık çevrimi yoksa.

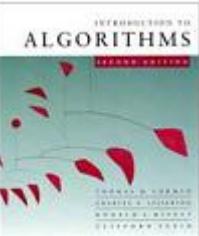
Süre =  $O(VE)$ .



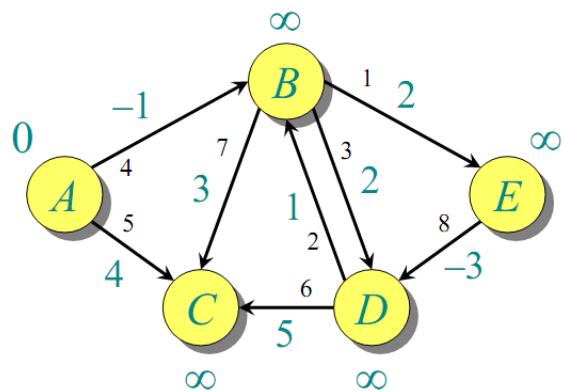
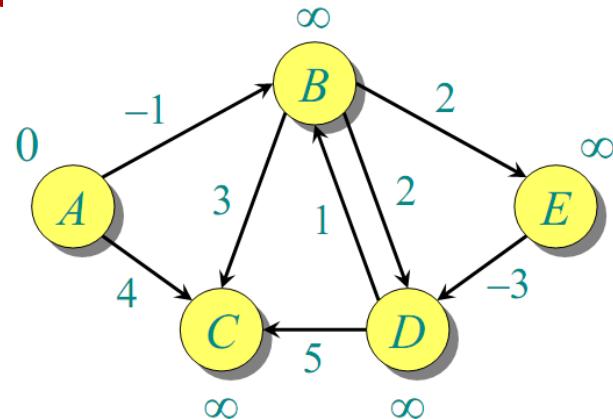
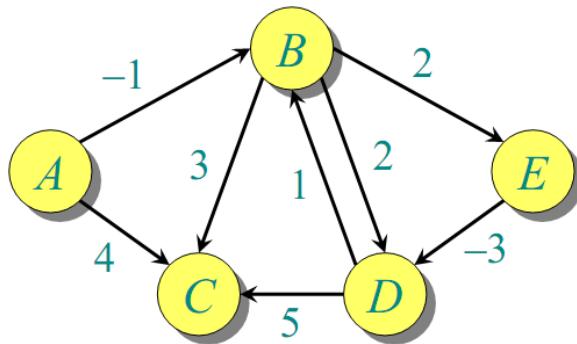
## Bellman-Ford örneği



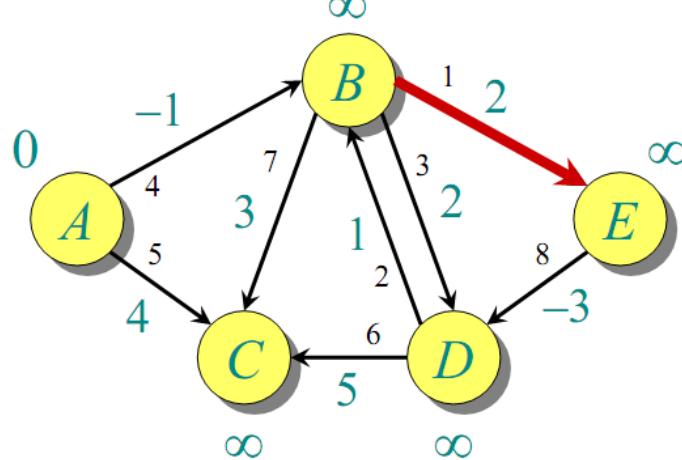
| A | B        | C        | D        | E        |
|---|----------|----------|----------|----------|
| 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ |



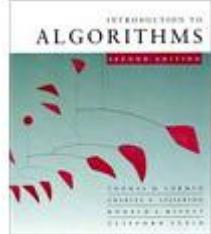
## Bellman-Ford örneği



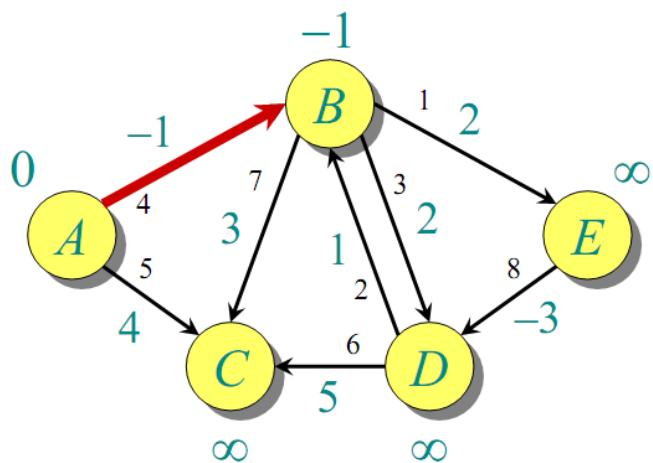
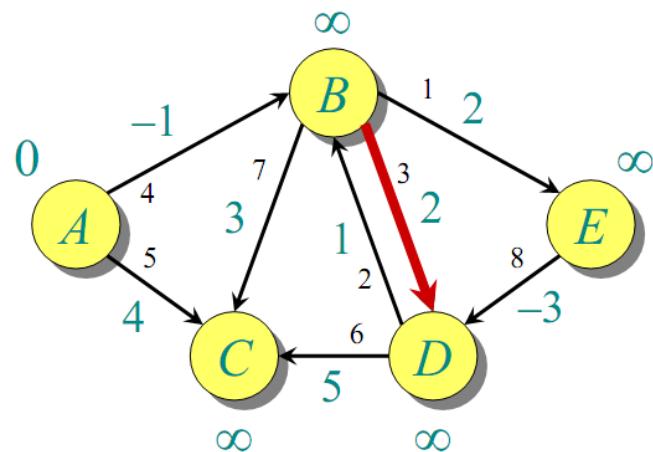
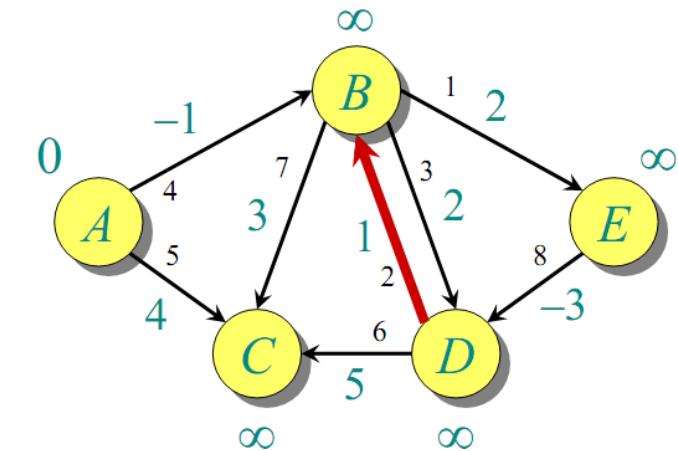
İlklandırmeye.



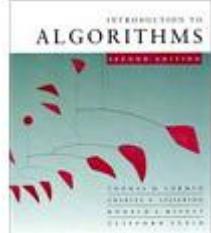
Köşe gevşetme düzeneşi



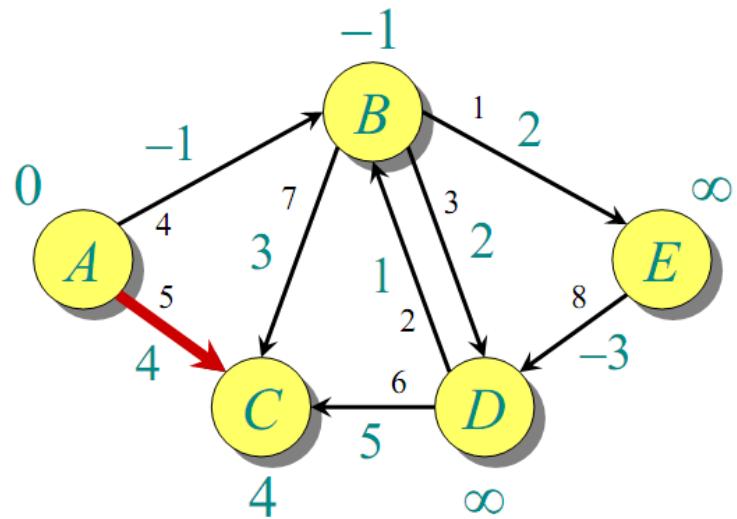
## Bellman-Ford örneği



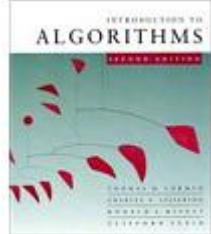
| <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|----------|----------|----------|----------|----------|
| 0        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | $\infty$ | $\infty$ | $\infty$ |



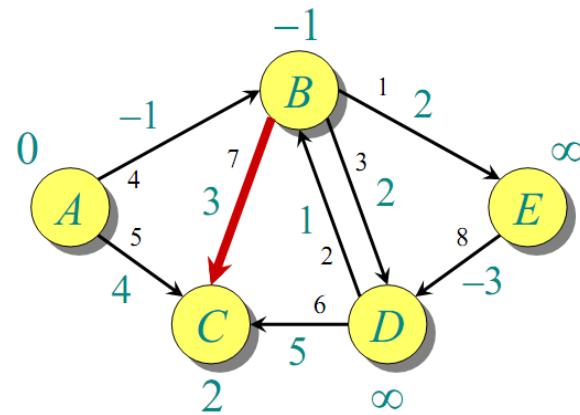
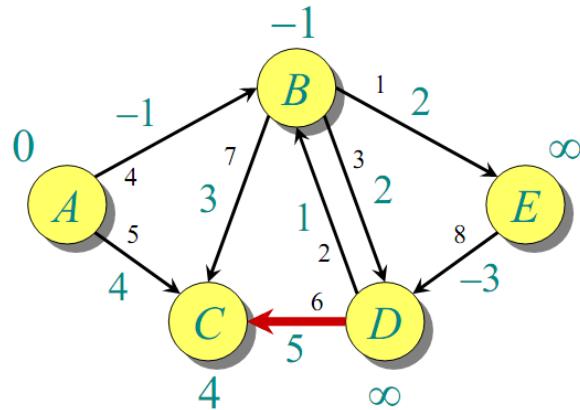
## Bellman-Ford örneği



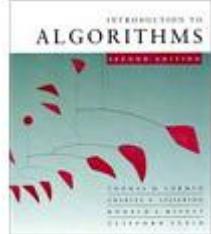
|   | A | B        | C        | D        | E        |
|---|---|----------|----------|----------|----------|
| 0 | 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1 | 0 | -1       | $\infty$ | $\infty$ | $\infty$ |
| 2 | 0 | -1       | 4        | $\infty$ | $\infty$ |



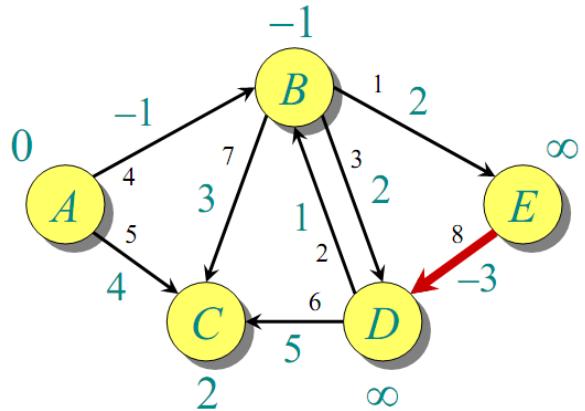
## Bellman-Ford örneği



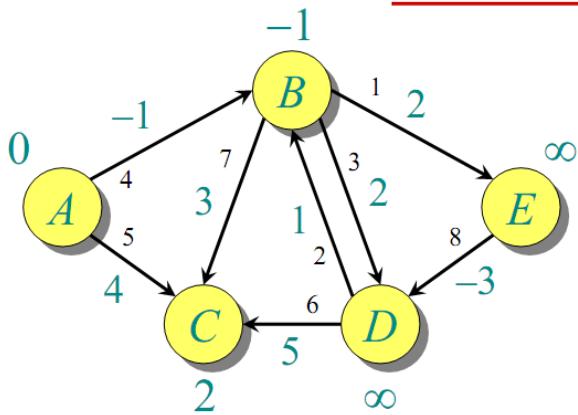
|   | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|---|----------|----------|----------|----------|----------|
| 0 | 0        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1 | 0        | -1       | $\infty$ | $\infty$ | $\infty$ |
| 2 | 0        | -1       | 4        | $\infty$ | $\infty$ |
| 3 | 0        | -1       | 2        | $\infty$ | $\infty$ |



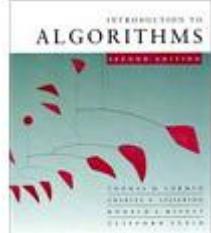
## Bellman-Ford örneği



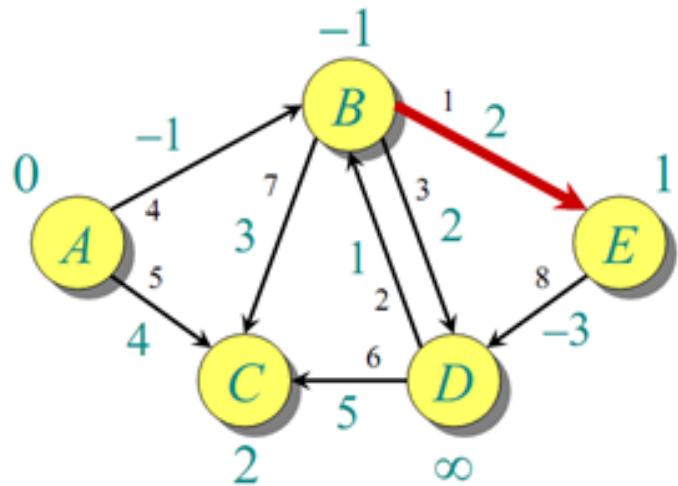
| <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|----------|----------|----------|----------|----------|
| 0        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | 4        | $\infty$ | $\infty$ |
| 0        | -1       | 2        | $\infty$ | $\infty$ |



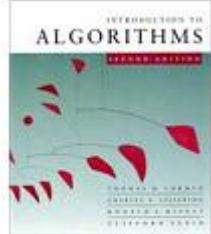
1. geçişin sonunda



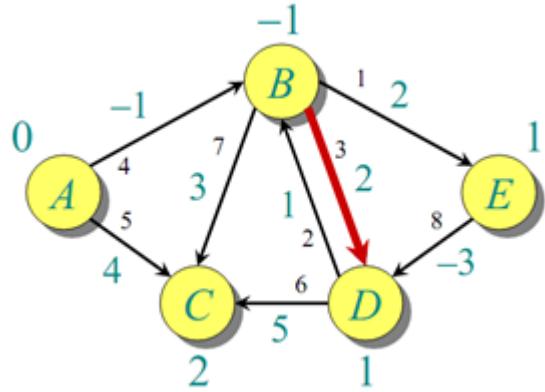
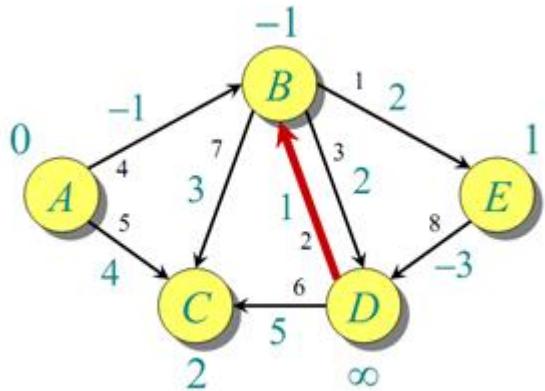
## Bellman-Ford örneği



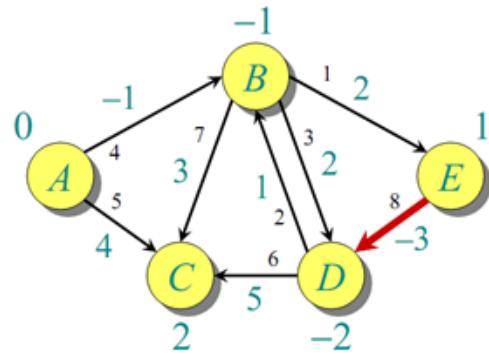
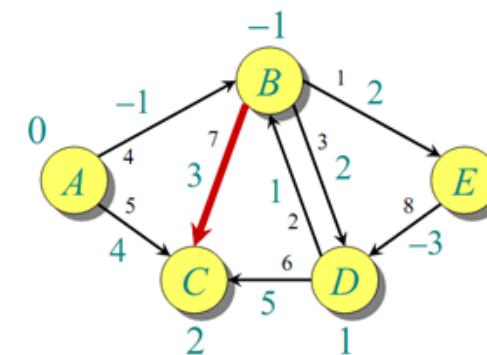
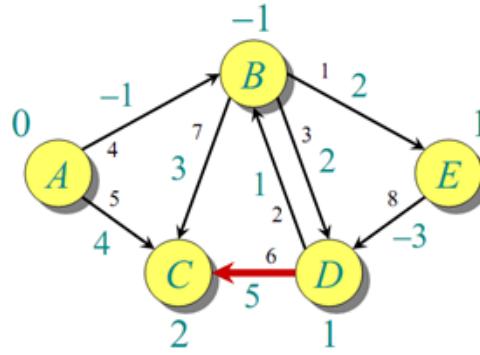
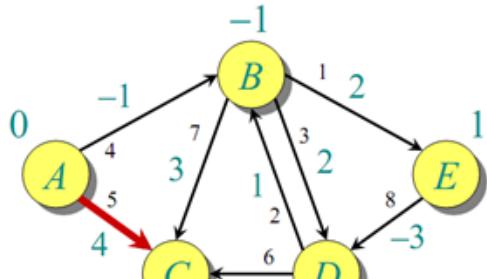
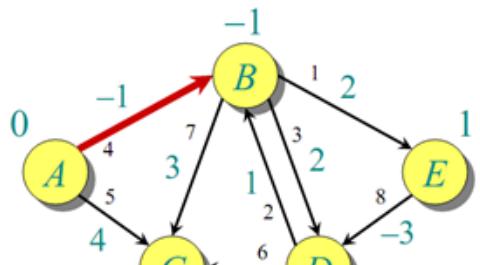
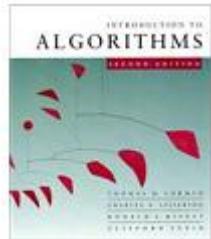
| <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|----------|----------|----------|----------|----------|
| 0        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | 4        | $\infty$ | $\infty$ |
| 0        | -1       | 2        | $\infty$ | $\infty$ |
| 0        | -1       | 2        | $\infty$ | 1        |



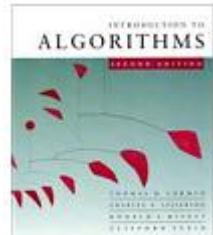
## Bellman-Ford örneği



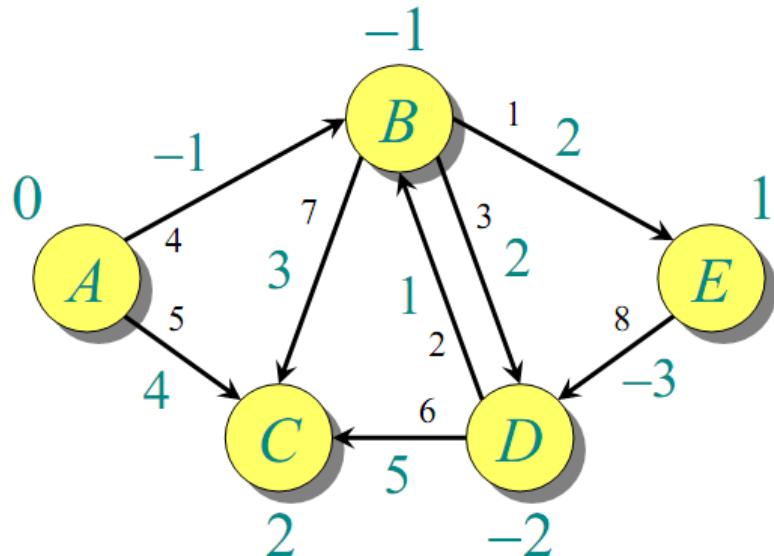
| <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|----------|----------|----------|----------|----------|
| 0        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | $\infty$ | $\infty$ | $\infty$ |
| 0        | -1       | 4        | $\infty$ | $\infty$ |
| 0        | -1       | 2        | $\infty$ | $\infty$ |
| 0        | -1       | 2        | $\infty$ | 1        |
| 0        | -1       | 2        | 1        | 1        |



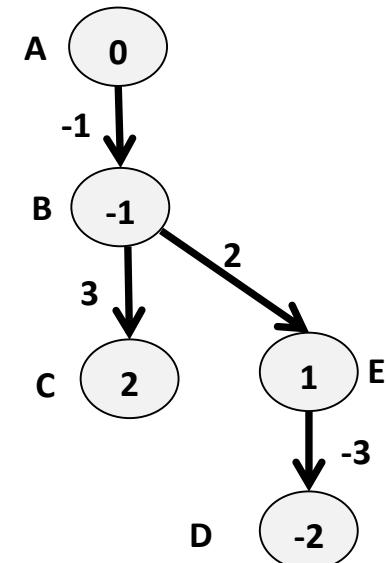
|   | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|---|----------|----------|----------|----------|----------|
| 0 | 0        | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 1 | -1       | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 2 | -1       | 4        | $\infty$ | $\infty$ | $\infty$ |
| 3 | -1       | 2        | $\infty$ | $\infty$ | 1        |
| 4 | -1       | 2        | $\infty$ | 1        | 1        |
| 5 | -1       | 2        | -2       | -2       | 1        |



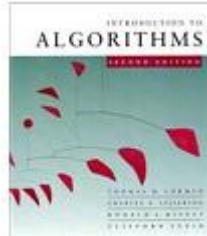
## Bellman-Ford örneği



|   | <i>A</i> | <i>B</i> | <i>C</i> | <i>D</i> | <i>E</i> |
|---|----------|----------|----------|----------|----------|
| 0 | $\infty$ | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0 | -1       | $\infty$ | $\infty$ | $\infty$ | $\infty$ |
| 0 | -1       | 4        | $\infty$ | $\infty$ | $\infty$ |
| 0 | -1       | 2        | $\infty$ | $\infty$ | $\infty$ |
| 0 | -1       | 2        | $\infty$ | 1        |          |
| 0 | -1       | 2        | 1        | 1        |          |
| 0 | -1       | 2        | -2       | 1        |          |



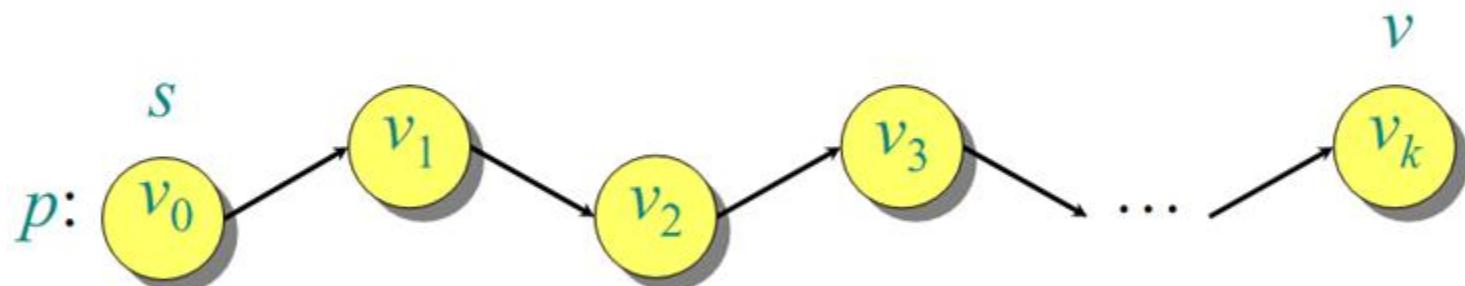
2. geçişin sonu (ve 3 ve 4).



# Doğruluk

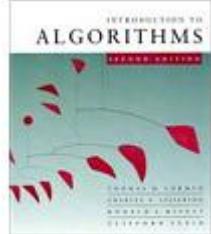
**Teorem.** Eğer  $G = (V, E)$  hiç negatif ağırlık çevrimi içermiyorsa, sonrasında Bellman-Ford algoritması bütün  $v \in V$  ler için  $d[v] = \delta(s, v)$  yi çalıştırır.

**Kanıt.**  $v \in V$  herhangi bir köşe olsun ve  $s'$  den  $v'$  ye, üzerinde en az sayıda köşe olan en kısa yolun  $p$  olduğunu farzedin.

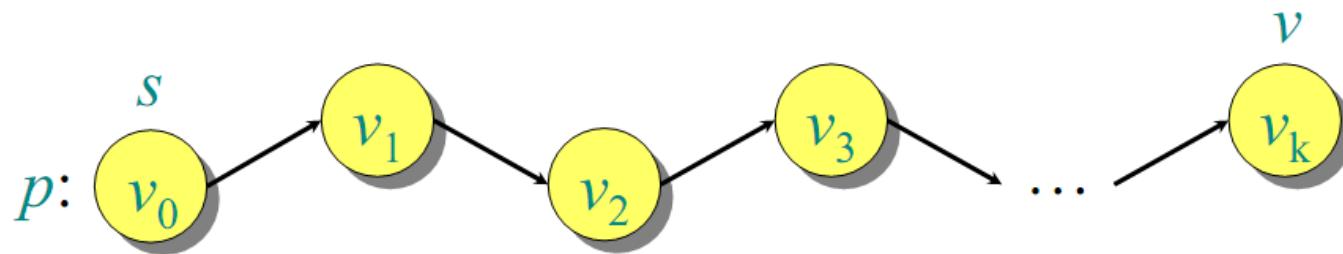


$p$  en kısa yol ise,

$$\delta(s, v_i) = \delta(s, v_{i-1}) + w(v_{i-1}, v_i).$$



## Doğruluk



İlk olarak,  $d[v_0] = 0 = \delta(s, v_0)$  ve  $d[v_0]$  sonraki gevşetmeler tarafından değiştirilmemiş.

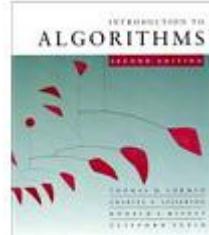
(Ders 14' teki  $d[v] \geq \delta(s, v)$  kuramı sebebiyle).

- $E'$  den 1 geçiş sonra,  $d[v_1] = \delta(s, v_1)$ .
- $E'$  den 2 geçiş sonra,  $d[v_2] = \delta(s, v_2)$ .
- ⋮
- $E'$  den  $k$  geçiş sonra,  $d[v_k] = \delta(s, v_k)$ .

Eğer  $G$  negatif ağırlık çevrimi içermiyorsa,  $p$  basittir.

En uzun basit yolun  $\leq |V| - 1$  kadar kenarı vardır.

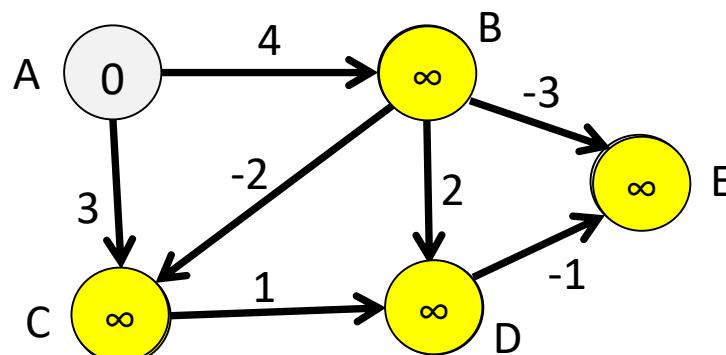
# Negatif ağırlık çevrimlerini bulma



**Doğal Sonuç.**  $|V|-1$  geçiş sonra  $d[v]$  değeri birleşmede başarısız olursa,  $G'$  de  $s'$  den erişilebilir bir negatif ağırlık çevrimi vardır.



# Bellman-Ford Alg: Örnek



| Düğüm | M        | Ata |
|-------|----------|-----|
| A     | 0        | -   |
| B     | $\infty$ |     |
| C     | $\infty$ |     |
| D     | $\infty$ |     |
| E     | $\infty$ |     |

1

2

3

4

5

6

7

(C, D)

(A, B)

(A, C)

(B, C)

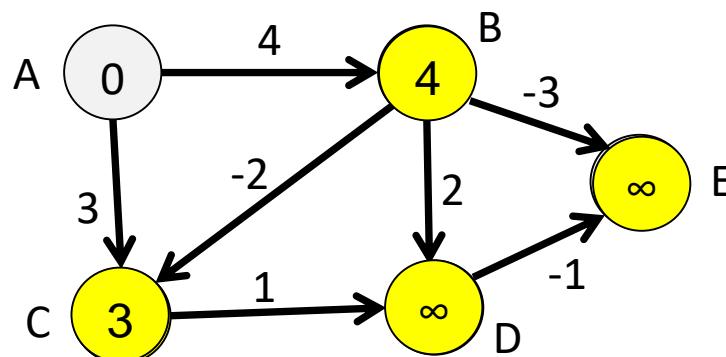
(B, D)

(B, E)

(D, E)

İlk Yineleme  
(İlklendirme)

# Bellman-Ford Alg: Örnek



| Düğüm | M        | Ata |
|-------|----------|-----|
| A     | 0        | -   |
| B     | 4        | A   |
| C     | 3        | A   |
| D     | $\infty$ |     |
| E     | $\infty$ |     |

1

2

3

4

5

6

7

(C, D)

(A, B)

(A, C)

(B, C)

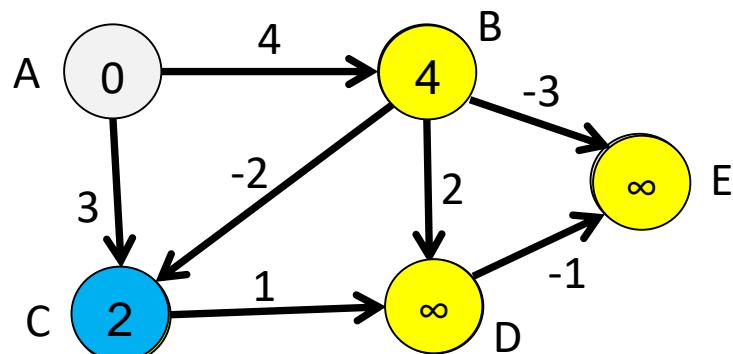
(B, D)

(B, E)

(D, E)

Köşe gevşetme düzeni

# Bellman-Ford Alg: Örnek



| Düğüm | M | Ata |
|-------|---|-----|
| A     | 0 | -   |
| B     | 4 | A   |
| C     | 2 | B   |
| D     | ∞ |     |
| E     | ∞ |     |

1

2

3

4

5

6

7

(C, D)

(A, B)

(A, C)

(B, C)

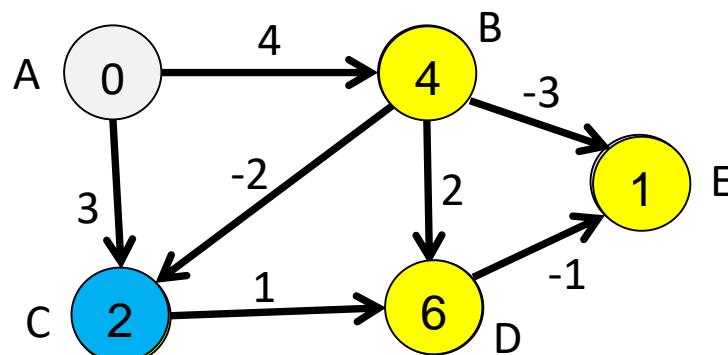
(B, D)

(B, E)

(D, E)

Köşe gevşetme düzeni

# Bellman-Ford Alg: Örnek



| Düğüm | M | Ata |
|-------|---|-----|
| A     | 0 | -   |
| B     | 4 | A   |
| C     | 2 | B   |
| D     | 6 | B   |
| E     | 1 | B   |

1

2

3

4

5

6

7

(C, D)

(A, B)

(A, C)

(B, C)

(B, D)

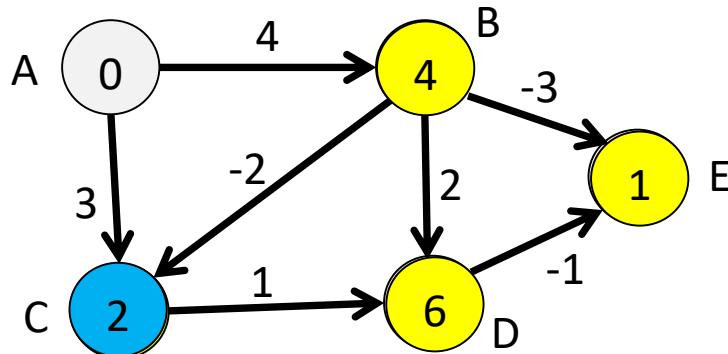
(B, E)

(D, E)

1.Tur tamamlandı

# Bellman-Ford Alg: Örnek

- $B = AB \ 4, \min(A,B) = 0 \rightarrow 0+4 = 4, \ C = AC \ 3, \min(A,C) = 0 \rightarrow 0+3 = 3$
- $C = BC \ -2, \min(B,C) = 4 \rightarrow 4-2 = 2, \ D = BD \ 2, \min(B,D) = 4 \rightarrow 4+2 = 6$
- $E = BE \ -3, \min(B,C) = 4 \rightarrow 4-3 = 1,$



| Düğüm | Ma. | Pred |
|-------|-----|------|
| A     | 0   | -    |
| B     | 4   | A    |
| C     | 2   | B    |
| D     | 6   | B    |
| E     | 1   | B    |

(C, D)

(A, B)

(A, C)

(B, C)

(B, D)

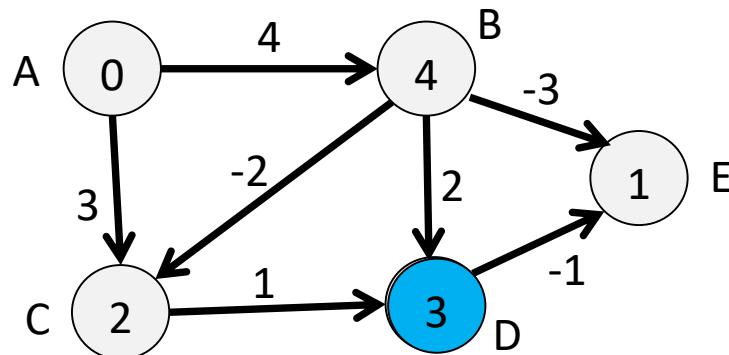
(B, E)

(D, E)

1.Tur sonu

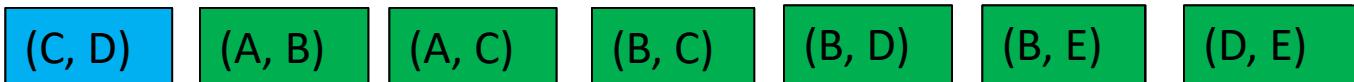
# Bellman-Ford Alg: Örnek

- D=CD 1,  $\min(C,D)=2 \rightarrow 2+1 = 3$



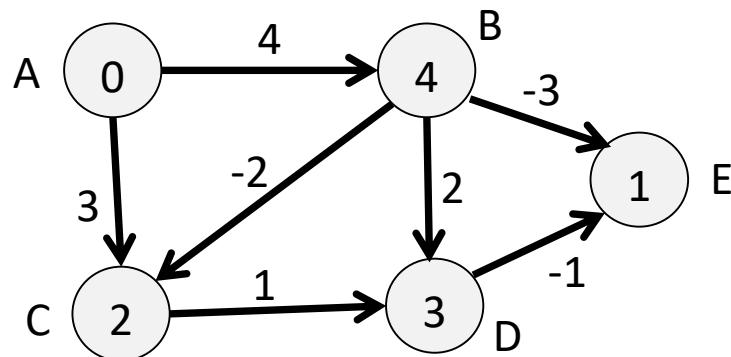
| Düğüm | Ma | Pred |
|-------|----|------|
| A     | 0  | -    |
| B     | 4  | A    |
| C     | 2  | B    |
| D     | 3  | C    |
| E     | 1  | B    |

1      2      3      4      5      6      7



2. tur yineleme

# Bellman-Ford Alg: Örnek



| Düğüm | Ma | Pred |
|-------|----|------|
| A     | 0  | -    |
| B     | 4  | A    |
| C     | 2  | B    |
| D     | 3  | C    |
| E     | 1  | B    |

(C, D)

(A, B)

(A, C)

(B, C)

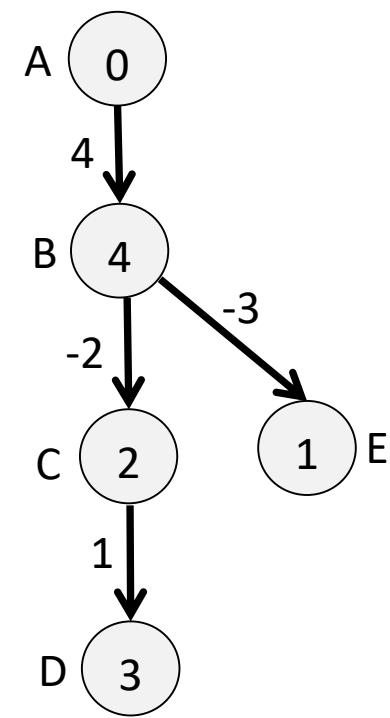
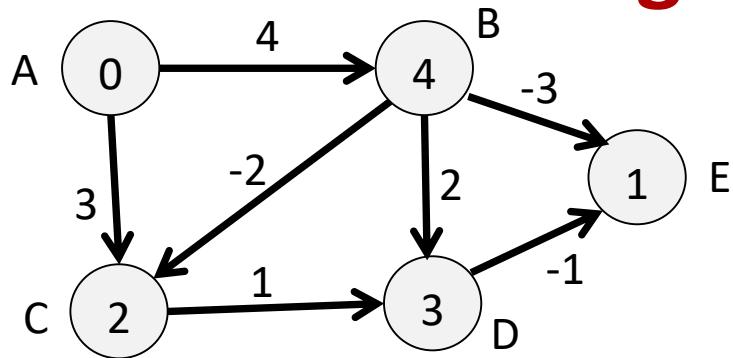
(B, D)

(B, E)

(D, E)

Üçüncü & Dördüncü Yineleme

## Bellman-Ford Alg: Sonuç

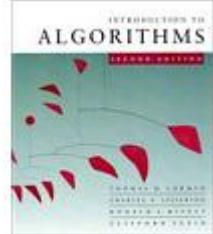


| Düğüm | Ma | Pred |
|-------|----|------|
| A     | 0  | -    |
| B     | 4  | A    |
| C     | 2  | B    |
| D     | 3  | C    |
| E     | 1  | B    |

# **14.Hafta**

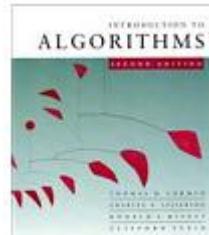
## **Tüm-ikili en kısa yollar**

- Tüm-ikili en kısa yollar (All-Pairs Shortest Paths )
  - Matris-çarpımı algoritması
  - Floyd-Warshall algoritması
  - Johnson algoritması



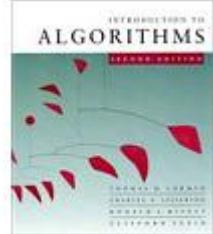
# Hatırlatma

- Şu ana kadar tek-kaynaklı-en kısa-yollar üzerinde kaynak bir köşeden diğer köşelerin her birine en kısa yolu bulmak için geliştirilen algoritmaların bahsedildi.
- Ağırlıksız durumda ve bütün kenar ağırlıkları bir olan graph için **BFS** (enine arama) uygulandı. Bu yapıda çalışma zamanı köşelerin sayıları ile kenarların sayılarının toplamından olustugundan doğrusal –zaman ( $O(V+E)$ ) 'a mal olur.
- İkinci en kolay durum ise negatif olmayan kenar ağırlıkları yani **Dijkstra** algoritması.
- Eğer iyi bir “fibonacci heap structure” yani yığın yapısı kullanılırsa, yaklaşık doğrusal zamana mal olmakta, yani  $O(E+V\log V)$ .



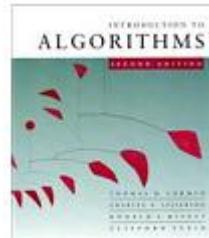
# Hatırlatma

- Ayrıca negatif ağırlıklar için geliştirilmiş genel ağırlıklar için **Bellman-Ford** algoritmasından bahsedildi ve çalışma zamanı  $O(VE)$ 'ye mal olmaktadır(bu biraz daha kötü)
- Eğer log faktörlerini hesaba alınmazsa, E'nin V düzeyinde olduğu, seyrek halde (komşuluk listesi) **Dijkstra**, doğrusal zamanlı ( $O(V+E)$ ), **Bellman-Ford** ise (eğer bağlantılı bir grafik varsa) en az karaseldir ( $O(V^2)$ ). Yoğun durumda (komşuluk matrisi) yani, E yaklaşık  $V^2$  olduğunda, Dijkstra karesel  $O(V^2)$ , Bellman-Ford ise kübik  $O(V^3)$  olur.



# Hatırlatma

- Dijkstra ve Bellman-Ford, birbirlerinden bir V faktörü kadar farklıdır ve bu da oldukça kötüdür.
- Ancak, tek kaynaklı en kısa yollarda, negatif kenar ağırlıklarının olduğu durumlarda bildiklerimizin en iyisi Bellman-Ford'dur.
- Bu bağlamda daha önce DAG-Directed Acyclic Graphs (Döngü Olmayan Yönüüt Graflar)rneğini de görmüştük ve orada Topolojik sıralama yapıyorduk.
- Demek oluyor ki, köşeler açısından bir düzenleme elde etmek için topolojik bir sıralama yapabiliriz. Sonra Bellman-Ford'u bir kere çalıştırırsak bir doğrusal zamanlı algoritma elde ederiz. DAG, ağırlıklarla dahi iyi bir uygulamanın nasıl yapılacağını bildiğimiz bir durum.



## En kısa yollar

### Tek-kaynaklı en kısa yollar

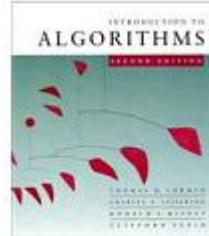
- Negatif olmayan kenar ağırlıkları
  - ◆ Dijkstra algoritması:  $O(E + V \lg V)$
- Genel
  - ◆ Bellman-Ford:  $O(VE)$
- DAG
  - ◆ Bellman-Ford' un bir turu:  $O(V + E)$

### Tüm-ikili en kısa yollar

- Negatif olmayan kenar ağırlıkları
  - ◆ Dijkstra algoritması çarpı  $|V|$ :  $O(VE + V^2 \lg V)$
- Genel
  - ◆ Bugün üç algoritma.

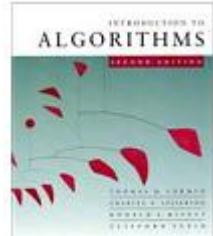
# Tüm-ikili en kısa yollar

## All-Pairs Shortest Paths



- Her iki köşe arasındaki en kısa yolun ağırlığını nasıl hesaplayabiliriz?
- Graph ağırlıksız ise, **BFS** algoritmasını kullanabiliriz.
- Çalışma zamanı  $|V| * \text{BFS}$  olur yani,  $O(V^2 + VE)$  olur. Yani en kısa yol ağırlığını hesaplayabilmemiz için kesinlikle, en az  $V^2$  gibi bir süreye ihtiyacımız var. Çünkü çıkışın boyutu  $V^2$ ; yani hesaplamanız gereken en kısa yol ağırlığı.
- Negatif olmayan kenar ağırlıkları durumunda, **Dijkstra'** yi,  $V$  defa uygulamak bunun koşma süresi de, gene  $O(VE + V^2 \log V)$  olur. **BFS** yi uygulamak ile aynı (Eğer log faktörünü dikkate almazsanız, ağırlıklı terim budur).
- Bu durum, **Bellman-Ford** artı bir log faktörü zaman alıyor, ki eğer negatif olmayan kenar ağırlıklarınız varsa, bu sürede “tüm-ikili-en-kısa-yollar”的 hepsini hesaplayabiliriz.

# Tüm-ikili en kısa yollar



- Negatif ağırlıkların olduğu durumda

**Girdi:**  $G = (V, E)$  yönlü grafiğinde,  $V = \{1, 2, \dots, n\}$  iken,  $w : E \rightarrow \mathbb{R}$

kenar-ağırlık fonksiyonuyla

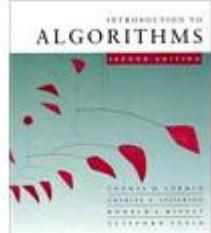
**Çıktı:**

Tüm  $i, j \in V$  için  $\delta(i, j)$  en kısa yol uzunlıklarının  $n \times n$  matrisi.

**Fikir:**

- Her köşeden Bellman-Ford' u bir tur çalıştır.
- Time (sure) =  $O(V^2E)$ .
- En kötü durumda yoğun grafik ( $n^2$  kenarlı)  $\Rightarrow \Theta(n^4)$  sure.

*İlk deneme için iyi!* Amacımız daha iyisini yapmak (Dinamik Programlama)



# Dinamik programlama

Yönlü grafikte,  $n \times n$  komşuluk matrisinin  $A = (a_{ij})$  olduğunu düşünün,

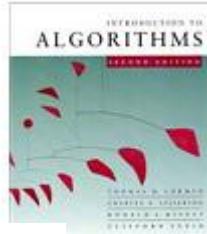
$d_{ij}^{(m)}$  =  $i$  'den  $j$  'ye en kısa yol ağırlığı-  
en çok  $m$  sayıda kenarda kullanıldığında.

**İddia:**

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if(eğer) } i = j \text{ ise,} \\ \infty & \text{if(eğer) } i \neq j \text{ ise;} \end{cases}$$

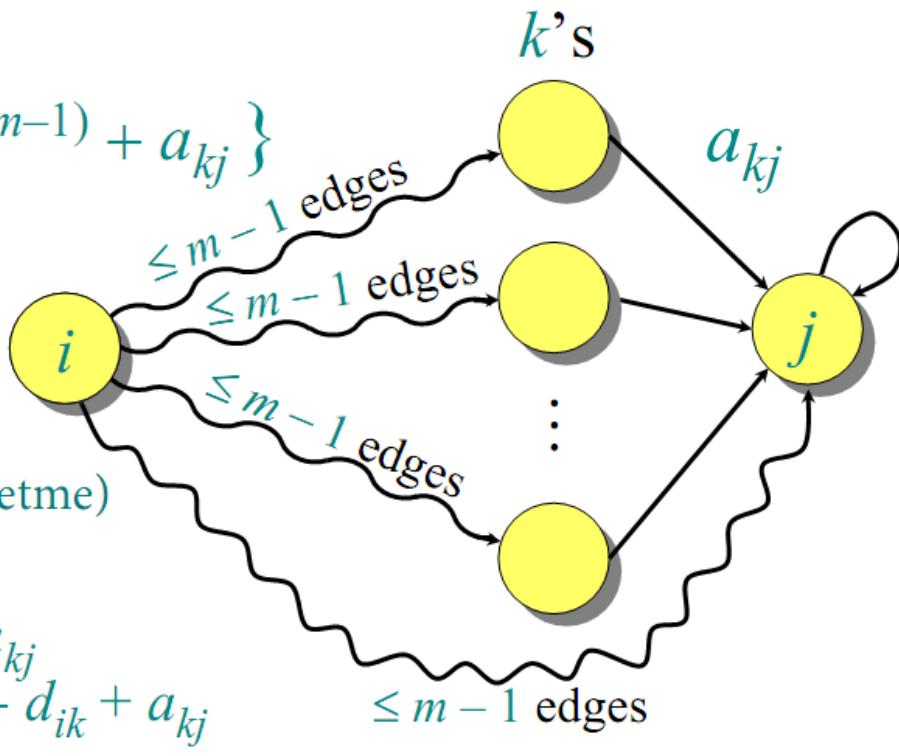
ve for(için)  $m = 1, 2, \dots, n - 1$ ,

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}.$$



# İddianın Kanıtı

$$d_{ij}^{(m)} = \min_k \{ d_{ik}^{(m-1)} + a_{kj} \}$$



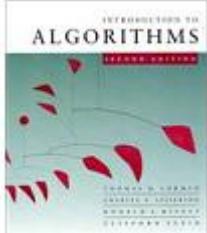
**Relaxation!** (gevsetme)

for  $k \leftarrow 1$  to  $n$   
 (için)  
 do if  $d_{ij} > d_{ik} + a_{kj}$   
 (yap eğer)  
 then  $d_{ij} \leftarrow d_{ik} + a_{kj}$   
 (sonra)

$O(n^4)$

**Not:** Negatif ağırlık çevrimi olmaması demek:

$$\delta(i, j) = d_{ij}^{(n-1)} = d_{ij}^{(n)} = d_{ij}^{(n+1)} = \dots$$



## Matris Çarpımı

$C, A,$  ve  $B$   $n \times n$  matrislerse  $C = A \cdot B$  yi hesapla:

$$c_{ij} = \sum_{k=1}^n a_{ik} b_{kj}.$$

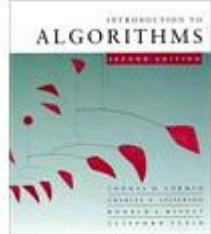
Time(süre) =  $\Theta(n^3)$  standart algoritmayı kullanıyor.

“+” → “min” ve “.” → “+” ya eşlemlersek?

$$c_{ij} = \min_k \{a_{ik} + b_{kj}\}.$$

Böylece,  $D^{(m)} = D^{(m-1)} \times A.$   $= A^m$

Özdeşlik matrisi =  $I = \begin{pmatrix} 0 & \infty & \infty & \infty \\ \infty & 0 & \infty & \infty \\ \infty & \infty & 0 & \infty \\ \infty & \infty & \infty & 0 \end{pmatrix} = D^0 = (d_{ij}^{(0)}).$   $= A^0$



## Matris Çarpımı

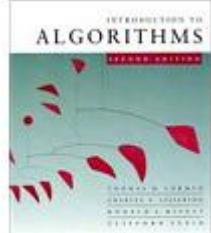
$(\min, +)$  çarpımı *çağrışimsal*dır, ve gerçek sayılarla, *kapalı semiring(closed semiring)* olarak adlandırılan cebirsel bir yapı oluşturur.

Sonuçta bunu hesaplayabiliriz

$$\begin{aligned} D^{(1)} &= D^{(0)} \cdot A = A^1 \\ D^{(2)} &= D^{(1)} \cdot A = A^2 \\ &\vdots && \vdots \\ D^{(n-1)} &= D^{(n-2)} \cdot A = A^{n-1}, \end{aligned}$$

yielding  $D^{(n-1)} = (\delta(i,j))$  verir.

Time(süre) =  $\Theta(n \cdot n^3) = \Theta(n^4)$ .  $n \times B-F'$  den daha iyi değil.



# Geliştirilmiş matris çarpım algoritması

**Tekrarlanan kareleme:**  $A^{2k} = A^k \times A^k$ .

Hesaplayın:  $\underbrace{A^2, A^4, \dots, A^{2^{\lceil \lg(n-1) \rceil}}}_{O(\lg n) \text{karelemeler}}$ .

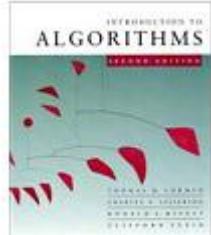
**Not:**  $A^{n-1} = A^n = A^{n+1} = \dots$

Time(süre) =  $\Theta(n^3 \lg n)$ .

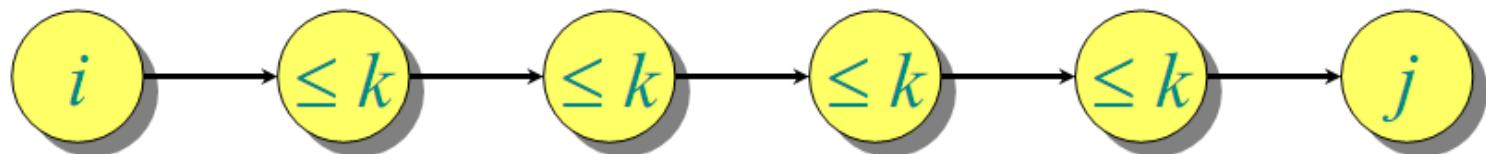
Negatif ağırlık çevrimlerini bulmak için, köşegendeki negatif değerleri  $O(n)$  ek zamanında kontrol edin.

# Tüm-ikili en kısa yollar

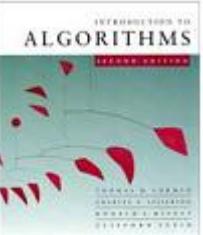
## Floyd-Warshall algoritması



Tanımlama  $c_{ij}^{(k)} = i'$  den  $j'$  ye, set  $\{1, 2, \dots, k\}$ ' e deki  
ara köşeleri olan en kısa yolun  
ağırlığı.

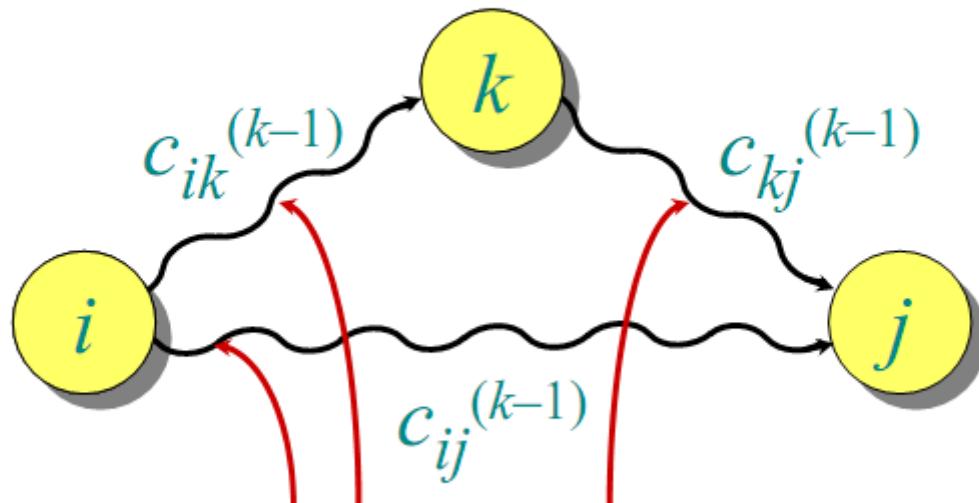


böylece,  $\delta(i, j) = c_{ij}^{(n)}$ . ve  $c_{ij}^{(0)} = a_{ij}$ .

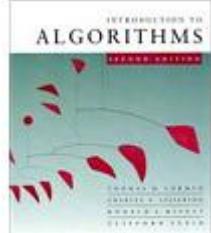


## Floyd-Warshall yinelemesi

$$c_{ij}^{(k)} = \min_k \{c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)}\}$$



$\{1, 2, \dots, k\}'$  deki ara köşeler



## Floyd-Warshall için sözde kod

```

for $k \leftarrow 1$ to n
 (icin)
 do for $i \leftarrow 1$ to n
 (icin yap)
 do for $j \leftarrow 1$ to n
 (icin yap)
 do if $c_{ij} > c_{ik} + c_{kj}$
 (yap eger)
 then $c_{ij} \leftarrow c_{ik} + c_{kj}$
 (sonra)
 }
 }
}

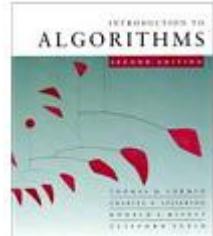
Gevsetme

```

### Notlar:

- Ekstra gevşetmelerin zararı olmayacağından üst simgeyi kullanmamak uygundur.
- $\Theta(n^3)$  zamanında çalışır.
- Kodlaması basittir.
- Pratikte verimlidir.

# Bir yönlendirilmiş grafigin geçişli kapanışı (transitive closure)



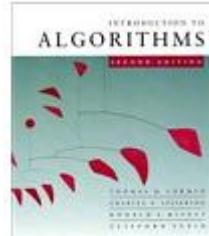
Hesaplayın  $t_{ij} = \begin{cases} 1 & i \text{ den } j \text{ ye bir yol varsa,} \\ 0 & \text{diğer durumda.} \end{cases}$

**Fikir:** Floyd-Warshall'ı ( $\min, +$ ) yerine ( $\vee, \wedge$ ) ile kullanın.

$$t_{ij}^{(k)} = t_{ij}^{(k-1)} \vee (t_{ik}^{(k-1)} \wedge t_{kj}^{(k-1)}).$$

Time(süre) =  $\Theta(n^3)$ .

# Floyd-Warshall Algoritması



**FLOYD-WARSHALL( $W$ )**

```

1 $n = W.\text{rows}$
2 $D^{(0)} = W$ $d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$
3 for $k = 1$ to n
4 let $D^{(k)} = (d_{ij}^{(k)})$ be a new $n \times n$ matrix
5 for $i = 1$ to n
6 for $j = 1$ to n
7 $d_{ij}^{(k)} = \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)})$
8 return $D^{(n)}$

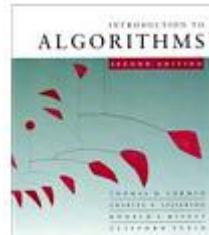
```

Constructing a shortest path

$$\pi_{ij}^{(0)} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } w_{ij} = \infty, \\ i & \text{if } i \neq j \text{ and } w_{ij} < \infty. \end{cases}$$

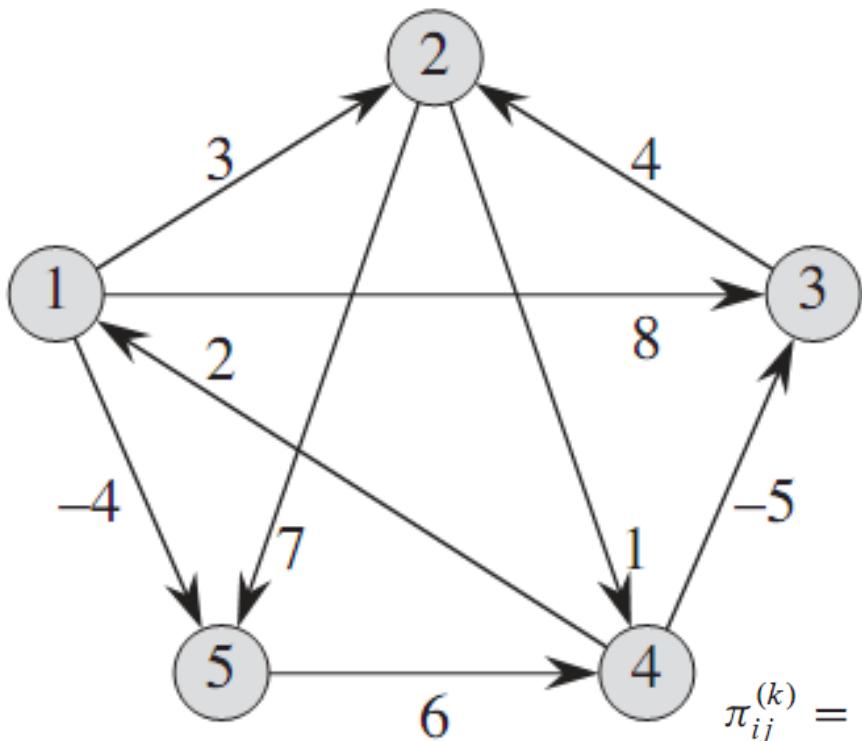
Atasını hesaplama

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$



# Floyd-Warshall Örnek

Kenar ağırlıkları



$$W = \text{matrix of weights} = \begin{bmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \end{bmatrix}$$

İkili en kısa yolları  
hesaplama

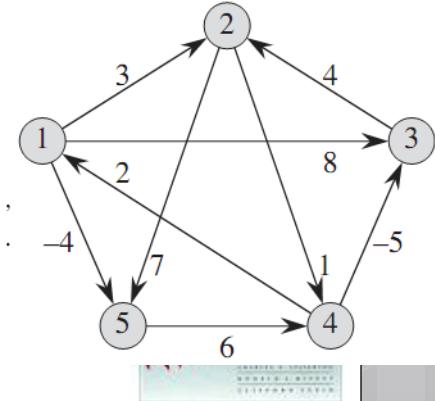
$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

Gidilecek düğümün  
atasını hesaplama

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}, \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}. \end{cases}$$



## Floyd-Warshall Örnek

$$D^{(0)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & \infty & -5 & 0 & \infty \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(0)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & \text{NIL} & 4 & \text{NIL} & \text{NIL} \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 3 & 8 & \infty & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & \infty & \infty \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(1)} = \begin{pmatrix} \text{NIL} & 1 & 1 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & \text{NIL} & \text{NIL} \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & 5 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(2)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 1 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

## Floyd-Warshall Örnek

$$D^{(3)} = \begin{pmatrix} 0 & 3 & 8 & 4 & -4 \\ \infty & 0 & \infty & 1 & 7 \\ \infty & 4 & 0 & 5 & 11 \\ 2 & -1 & -5 & 0 & -2 \\ \infty & \infty & \infty & 6 & 0 \end{pmatrix} \quad \Pi^{(3)} = \begin{pmatrix} \text{NIL} & 1 & 1 & 2 & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 2 & 2 \\ \text{NIL} & 3 & \text{NIL} & 2 & 2 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ \text{NIL} & \text{NIL} & \text{NIL} & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(4)} = \begin{pmatrix} 0 & 3 & -1 & 4 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(4)} = \begin{pmatrix} \text{NIL} & 1 & 4 & 2 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

$$D^{(5)} = \begin{pmatrix} 0 & 1 & -3 & 2 & -4 \\ 3 & 0 & -4 & 1 & -1 \\ 7 & 4 & 0 & 5 & 3 \\ 2 & -1 & -5 & 0 & -2 \\ 8 & 5 & 1 & 6 & 0 \end{pmatrix} \quad \Pi^{(5)} = \begin{pmatrix} \text{NIL} & 3 & 4 & 5 & 1 \\ 4 & \text{NIL} & 4 & 2 & 1 \\ 4 & 3 & \text{NIL} & 2 & 1 \\ 4 & 3 & 4 & \text{NIL} & 1 \\ 4 & 3 & 4 & 5 & \text{NIL} \end{pmatrix}$$

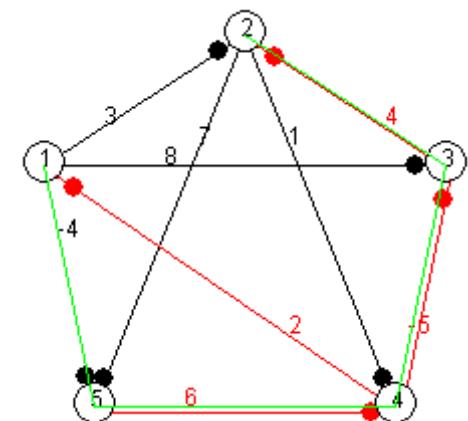
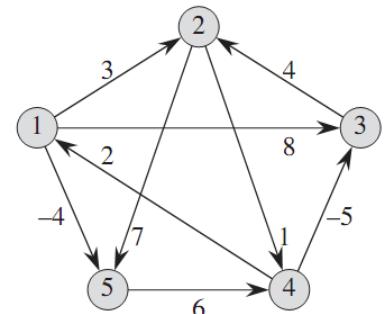
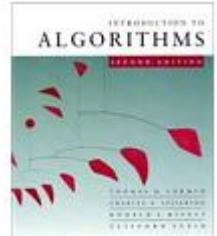


Figure 25.4 The sequence of matrices  $D^{(k)}$  and  $\Pi^{(k)}$  computed by the Floyd-Warshall algorithm

**1.Düğüm için:** 5'in atası 1, 4'ün atası 5, 3'ün atası 4, 2'nin atası da 3 tür.  
 $1-5=-4$ ,  $5-4 = -4+6=2$ ,  $4-3=-5+2=-3$ ,  $3-2=-3+4=1$ , Yol: 1-5-4-3-2

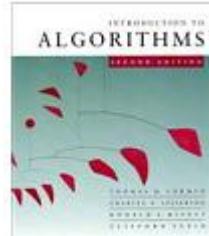
# Johnson algoritması



- 1977 yılında Donald B. Johnson tarafından geliştirilmiştir. **Bellman Ford, Reweighting ve Dijkstra Algoritması** tabanlı, All pairs problemini çözmek için kullanılan bir algoritmadır.
- Sparse(dağınık) ve directed(yönlü) graflar için kullanılan güzel bir çözüm yoludur.
- Bağlantıların negatif olmasına izin vermektedir ve bu en önemli özelliğidir. Negatif bağlantıları reweighting yöntemiyle işlem sırasında ağırlıkları yeniden hesaplayarak pozitif ağırlıklara güncellemektedir. Bu yönüyle Floyd Warshall'a benzemektedir fakat  $O(V^3)$  olan çalışma süresi Johnson Algoritması'nın tercih edilmesine neden olur.
- Ayrıca Floyd Warshall daha sık graflarda tercih edilirken, Johnson seyrek graflarda daha iyidir. Ağırlıkların pozitif olması durumunda Dijkstra'nın kullanılması daha iyi performans sağlar.

# Grafik yeniden ağırlıklandırması

## Johnson algoritması

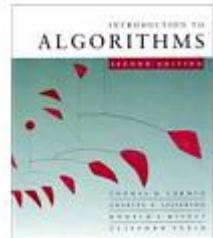


**Teorem.**  $h : V \rightarrow \mathbb{R}$ , fonksiyonu verilmiş, her  $(u, v) \in E$  kenarını  $w_h(u, v) = w(u, v) + h(u) - h(v)$  ile yeniden ağırlıklandırın. Bu durumda, her iki köşe arasındaki bütün yollar aynı miktarda yeniden ağırlıklandırılır.

**Kanıt.**  $p = v_1 \rightarrow v_2 \rightarrow \dots \rightarrow v_k$ ,  $G$  'de bir yol olsun.

$$\begin{aligned}
 w_h(p) &= \sum_{i=1}^{k-1} w_h(v_i, v_{i+1}) \\
 &= \sum_{i=1}^{k-1} (w(v_i, v_{i+1}) + h(v_i) - h(v_{i+1})) \\
 &= \sum_{i=1}^{k-1} w(v_i, v_{i+1}) + h(v_1) - h(v_k) \\
 &= w(p) + h(v_1) - h(v_k).
 \end{aligned}$$

*Aynı miktar!*



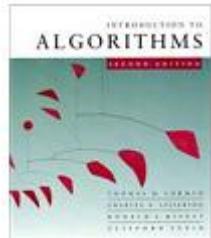
## Yeniden ağırlıklandırılan grafiklerde en kısa yollar

**D.Sonuç.**  $\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$ . □

**Fikir:**  $h : V \rightarrow \mathbb{R}$  fonksiyonunu bulun:  
Tüm  $(u, v) \in E$  ler için  $wh(u, v) \geq 0$  olduğunda.

Sonra da yeniden ağırlıklandırılmış grafikte,  
her köşeden Dijkstra'nın algoritmasını çalıştırın.

**NOT:**  $w_h(u, v) \geq 0$  iff(eğer ve sadece eğer)  
 $h(v) - h(u) \leq w(u, v)$ .



# Johnson algoritması

1. Şu fonksiyonu bulun  $h : V \rightarrow \mathbb{R}$ :

Tüm  $(u, v) \in E'$  ler için  $wh(u, v) \geq 0$  üzerinde Bellman-Ford' u çalıştırın  $h(v) - h(u) \leq w(u, v)$  fark kısıtlarını çözün veya bir negatif ağırlık çevrimi varsa saptayın.

- Time(süre) =  $O(VE)$ .

2. Dijkstra'nın algoritmasını  $w_h$ ' yi kullanarak, her köşeden  $(u \in V)$ ,  $\delta_h(u, v)$  ' hesaplayın (tüm  $v \in V$  için).

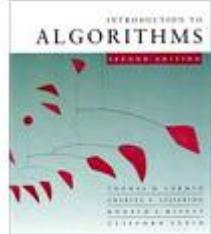
- Time(süre) =  $O(VE + V^2 \lg V)$ .

3. Her  $(u, v) \in V \times V$  için,

$$\delta(u, v) = \delta_h(u, v) - h(u) + h(v) \text{ hesaplayın.}$$

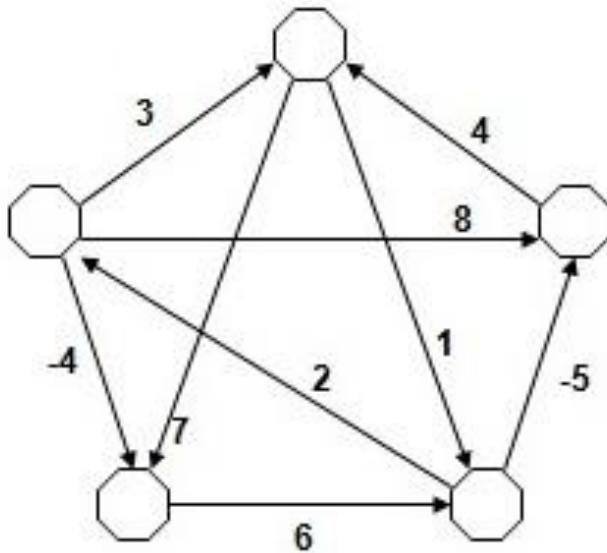
- Time(süre) =  $O(V^2)$ .

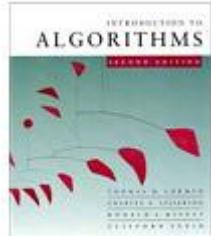
Toplam süre =  $O(VE + V^2 \lg V)$ .



# Johnson algoritması

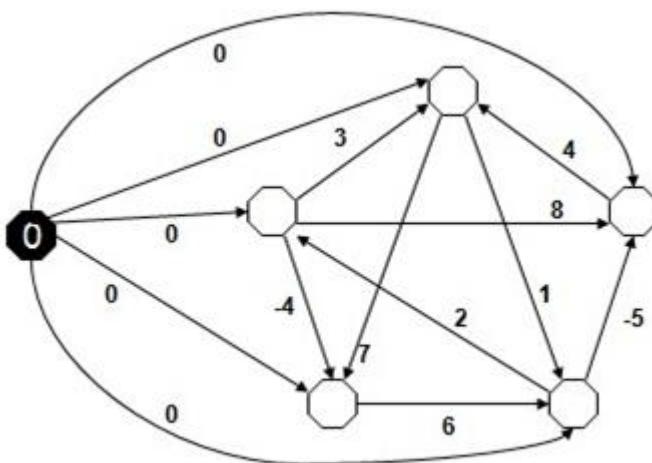
- Örnek: Algoritmanın adımlarını aşağıdaki graf üzerinde açıklayalım.

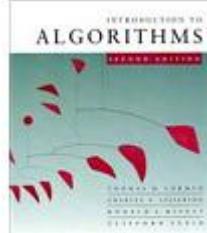




# Johnson algoritması

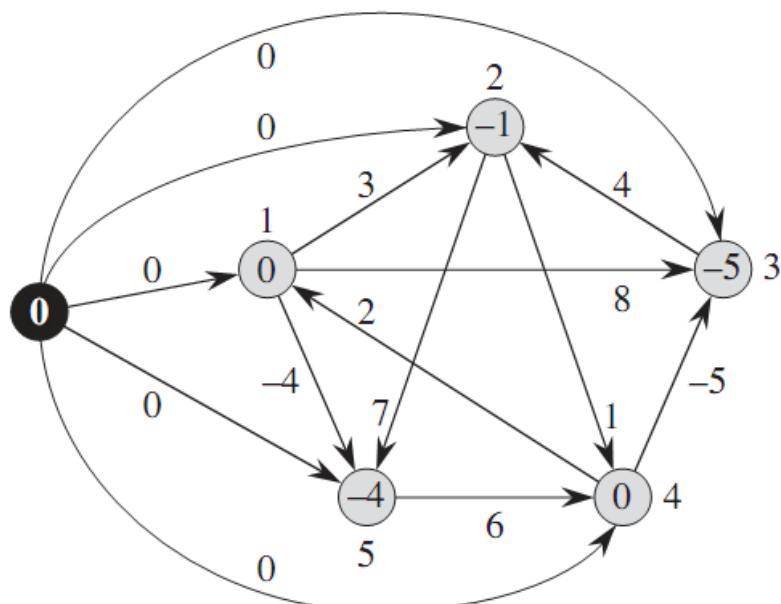
- **Adım 1:** Öncelikle grafa aşağıda görüldüğü üzere yeni bir düğüm ve bu düğümden grafta bulunan tüm düğümlere bağlantılar eklenir. Bu düğümün ve bağlantılarının ağırlığı sıfır olarak belirlenir.

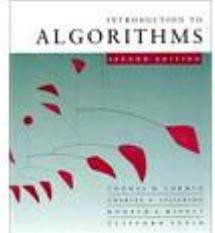




## Johnson algoritması

- **Adım 2 :** Her düğüm için Bellman-Ford Algoritması bir kez çalıştırılır ve düğümlerin ağırlıkları belirlenir. Aşağıda görüldüğü üzere her düğümün ağırlığı içerisinde yazıldı.

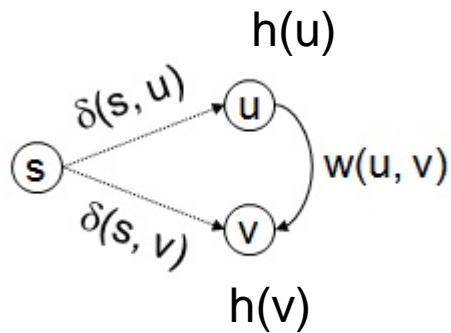




# Johnson algoritması

- Adım 3 : Adım 4'te Dijkstra Algoritması kullanılacaktır.

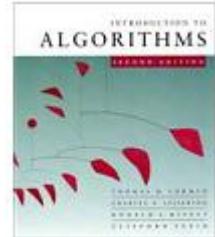
Bilindiği üzere Dijkstra Algoritması negatif bağlantı uzunluklarını kabul etmemektedir. Bu yüzden bu adımda ağırlıklar tekrar hesaplama yöntemiyle yenilecek ve negatif bağlantı kalmayacaktır. Yeniden hesaplama yöntemi şu şekildedir;  $\hat{w}(u, v) = w(u, v) + d(s, u) - d(s, v)$



$$\underbrace{w(u, v) + \delta(s, u) - \delta(s, v)}_{\hat{w}(u, v)} \geq 0$$

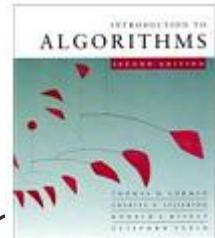
$$\begin{aligned}\delta_h(u, v) &= \delta(u, v) + h(u) - h(v) \\ \delta(u, v) &= \delta_h(u, v) - h(u) + h(v)\end{aligned}$$

# Johnson algoritması

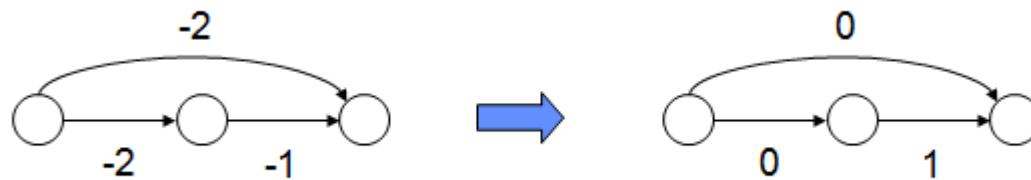


- Ağırlıkların yeniden hesaplanması Reweighting Teorem ile yapılır. Formülde yeni kenar ağırlığı eski kenar ağırlığı ile düğümün, algoritmanın birinci adımda eklenen yeni düğümüne olan ağırlığı ile toplanır. Bu değerden gidilecek olan düğümün S düşümüne olan ağırlığı çıkarılır ve yeni ağırlık bulunur. ( $d_h[u,v] = d[u,v] + h[u] - h[v]$ )
- Reweighting işlemi sonucunda shortest path değişmezken, ağırlıkların hepsi nonnegative olur. Burada akla şu soru gelebilir, ağırlıkları bu şekilde hesaplayacağımıza tüm düğümlere minimum bağlantı uzunluğunu eklesek olmaz mı? Olmaz çünkü bu en kısa yolun değişmesine sebep olabilir.

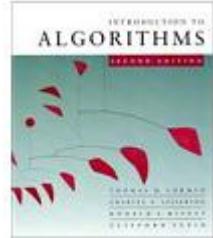
# Johnson algoritması



- Aşağıdaki grafikte tüm düğümlere minimum düğüm ağırlığı eklenmesi durumunda ortaya çıkacak bozulma görülmektedir. Birinci durumda kısa yol alttaki iken, ikinci durumda üstteki oluyor. Bu metod görüldüğü üzere kısa yolu değiştirmeye sebep oluyor.

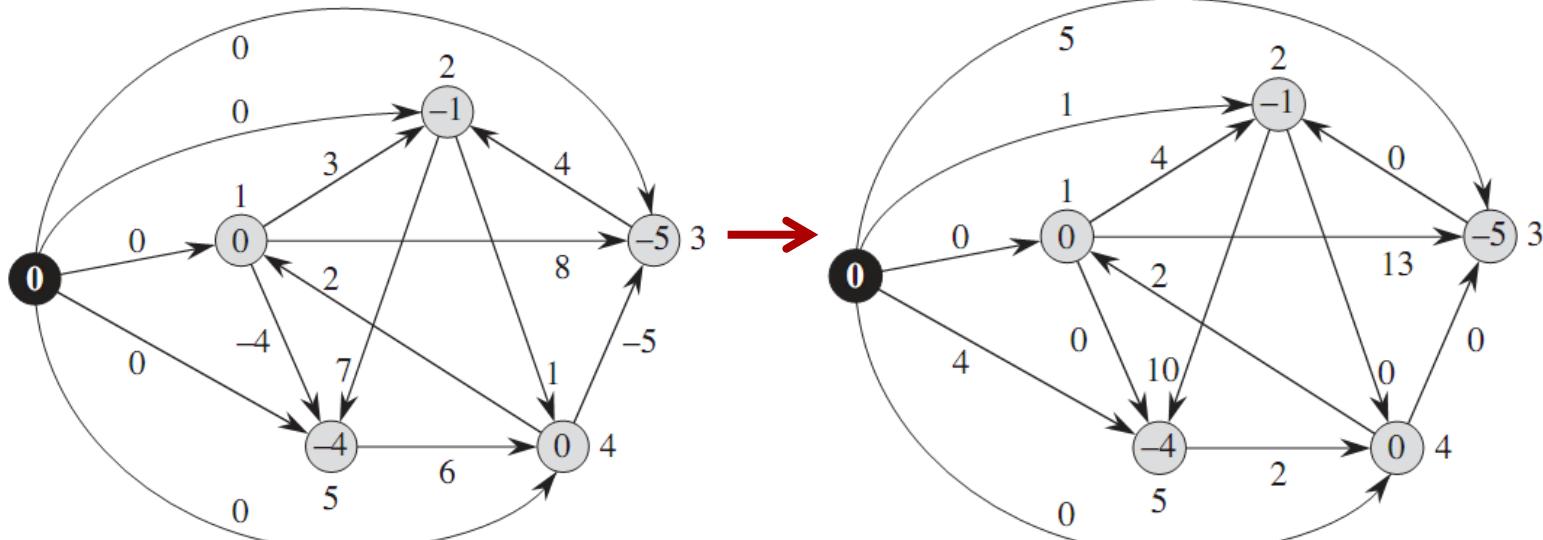


# Johnson algoritması

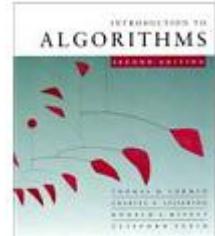


- Aşağıda ağırlıkların güncellenmiş hali bulunmaktadır.  
Kenarların yeni ağırlıklarının bulunma işlemine örnek verecek olursak ; eski ağırlığı 8 olan kenarın (1-3) yeni ağırlığı yukarıda bahsettiğimiz yöntemle  $8 + 0 - (-5) = 13$  'e olarak hesaplanır. (1-2 kenarı için: $3+0-(-1)=4$

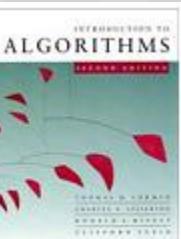
$$\delta_h(u, v) = \delta(u, v) + h(u) - h(v)$$



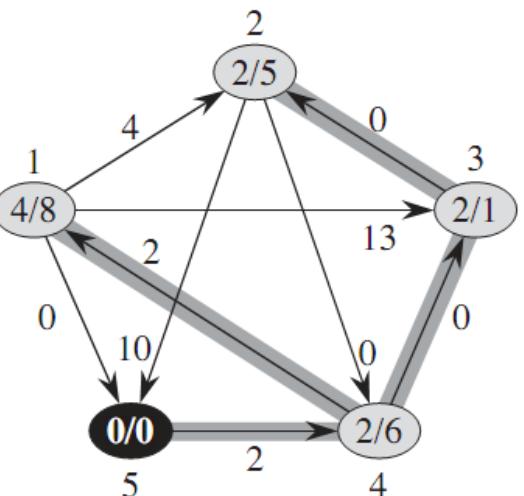
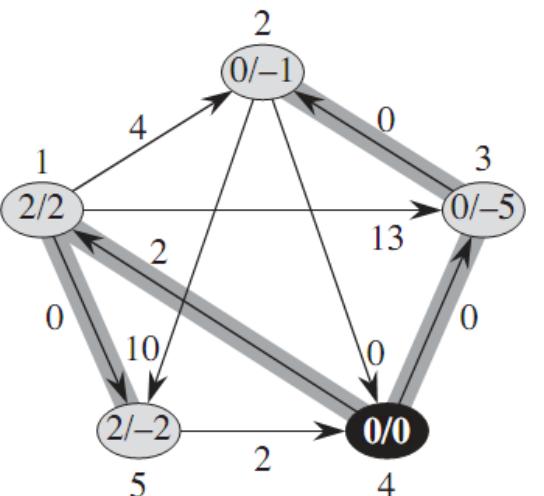
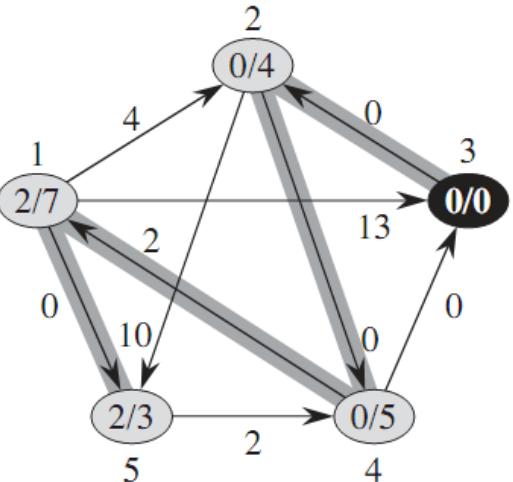
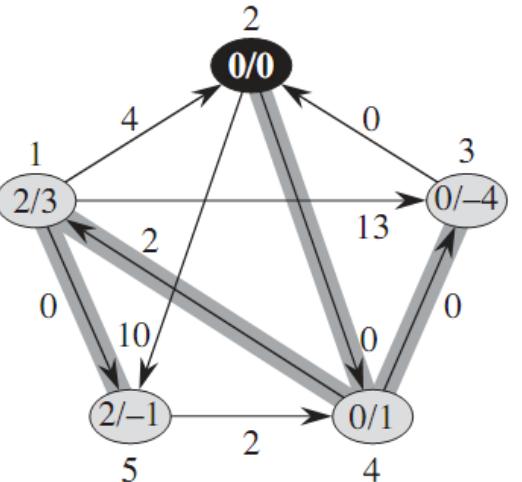
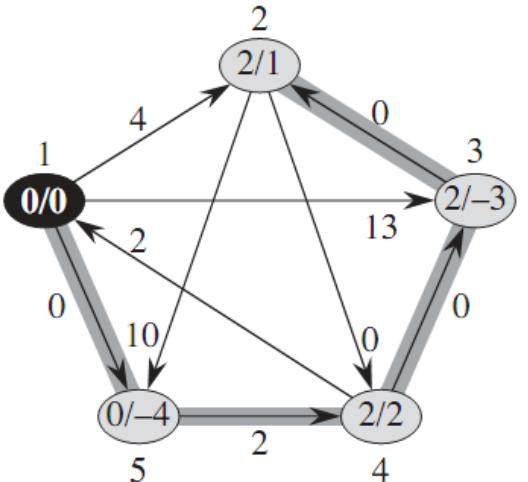
# Johnson algoritması

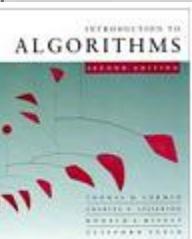


- **Adım 4 :** Birinci adımda eklenen düğüm graftan silinir ve geri kalan tüm düğümler için Dijkstra Algoritması uygulanır. Tüm çiftler arası en kısa yol bulunur. Aşağıda tüm düğümler için ağırlıkların Dijkstra Algoritması'yla yeniden hesaplanması görülmektedir.



# Johnson algoritması



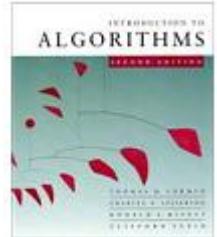


# Johnson algoritması

**JOHNSON( $G, w$ )**

- 1 compute  $G'$ , where  $G'.V = G.V \cup \{s\}$ ,  
 $G'.E = G.E \cup \{(s, v) : v \in G.V\}$ , and  
 $w(s, v) = 0$  for all  $v \in G.V$
- 2 **if** BELLMAN-FORD( $G', w, s$ ) == FALSE  
3     print “the input graph contains a negative-weight cycle”
- 4 **else for** each vertex  $v \in G'.V$   
5         set  $h(v)$  to the value of  $\delta(s, v)$   
6             computed by the Bellman-Ford algorithm
- 6 **for** each edge  $(u, v) \in G'.E$   
7          $\hat{w}(u, v) = w(u, v) + h(u) - h(v)$
- 8     let  $D = (d_{uv})$  be a new  $n \times n$  matrix
- 9     **for** each vertex  $u \in G.V$   
10         run DIJKSTRA( $G, \hat{w}, u$ ) to compute  $\hat{\delta}(u, v)$  for all  $v \in G.V$
- 11         **for** each vertex  $v \in G.V$   
12              $d_{uv} = \hat{\delta}(u, v) + h(v) - h(u)$
- 13     **return**  $D$

# Johnson algoritması analizi

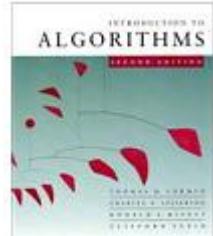


- Algoritmanın adımlarını kontrol edelim.
  - İlk adımda yeni düğüm ekleniyor ve her düğümün ağırlığı Bellman Ford ile hesaplanıyor. Bu durumun getirdiği karmaşıklık  $O(VE)$ 'dir.
  - Daha sonra negatif kenarlardan kurtarmak için reweighting işlemi yapılıyor. Bu durumun getirdiği karmaşıklık  $O(E)$ 'dir.
  - Her düğüm için Djikstra Algoritması'nın getirdiği karmaşıklık  $O(V^2 \log V + VE \log V)$ 'dir.
  - Bu durumda Johnson Algoritması'nın karmaşıklığı  $O(V^2 \log V + VE \log V)$  olarak hesaplanır.
  - ( $V$ : Düğümler Kümesi ,  $E$ : Bağlantılar Kümesi )

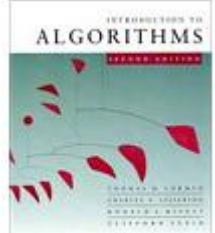
Örnek

36

# Floyd-Warshall Algoritması

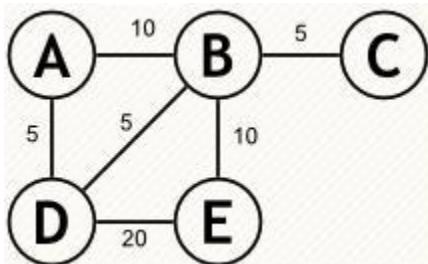


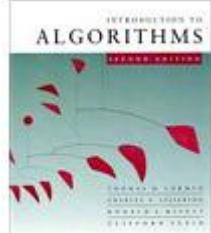
- for  $i = 1$  to  $N$
- for  $j = 1$  to  $N$
- if( $i$ 'den  $j$ 'ye bir yol varsa)
  - $yol[0][i][j] = i$  ile  $j$  arasındaki mesafe
  - else
    - $yol[0][i][j] = \infty$
- for  $k = 1$  to  $N$
- for  $i = 1$  to  $N$
- for  $j = 1$  to  $N$
- $yol[k][i][j] = \min(yol[k-1][i][j], yol[k-1][i][k] + yol[k-1][k][j])$



## Floyd-Warshall Örnek

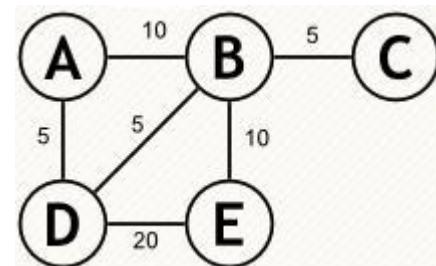
- Algoritmanın çalışmasını daha iyi anlayabilmek için aşağıdaki örnek üzerinden adım adım algoritmayı kullanarak en kısa yolu bulalım:

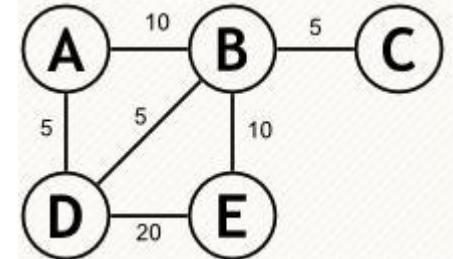




## Floyd-Warshall Örnek

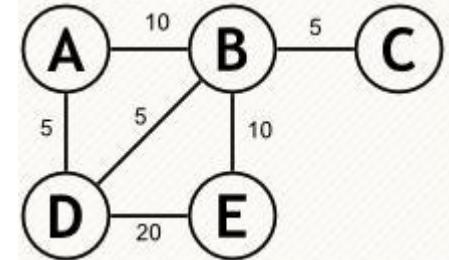
- Yukarıdaki şekil incelendiğinde A'dan E'ye giden birden çok yol bulunabilir:
  - Yol 1: A -> B -> E                    20
  - Yol 2: A -> D -> E                    25
  - Yol 3: A -> B -> D -> E        35
  - Yol 4: A -> D -> B -> E        20
- Yukarıdaki yollar çıkarıldıkten sonra en kısasının 20 uzunlığında olduğu bulunabilir. Şimdi bu yollardan en kısasını Floyd-Warshall algoritmasının nasıl bulduğunu adım adım inceleyelim:





## Floyd-Warshall Örnek

- 1. Adımda komşuluk listesine göre matris inşa edilir.
- Yukarıdaki şekilde doğrudan ilişkisi bulunan düğümler ve ağırlıkları aşağıda verilmiştir:
- |  |   |   |   |   |   |
|--|---|---|---|---|---|
|  | A | B | C | D | E |
|--|---|---|---|---|---|
- |   |   |    |          |   |          |
|---|---|----|----------|---|----------|
| A | 0 | 10 | $\infty$ | 5 | $\infty$ |
|---|---|----|----------|---|----------|
- |   |    |   |   |   |    |
|---|----|---|---|---|----|
| B | 10 | 0 | 5 | 5 | 10 |
|---|----|---|---|---|----|
- |   |          |   |   |          |          |
|---|----------|---|---|----------|----------|
| C | $\infty$ | 5 | 0 | $\infty$ | $\infty$ |
|---|----------|---|---|----------|----------|
- |   |   |   |          |   |    |
|---|---|---|----------|---|----|
| D | 5 | 5 | $\infty$ | 0 | 20 |
|---|---|---|----------|---|----|
- |   |          |    |          |    |   |
|---|----------|----|----------|----|---|
| E | $\infty$ | 10 | $\infty$ | 20 | 0 |
|---|----------|----|----------|----|---|
- Yukarıdaki grafta doğrudan ilişkisi bulunmayan düğümlerin değerleri  $\infty$  olarak gösterilmektedir. Diğer değerler doğrudan ağırlıkları göstermektedir.

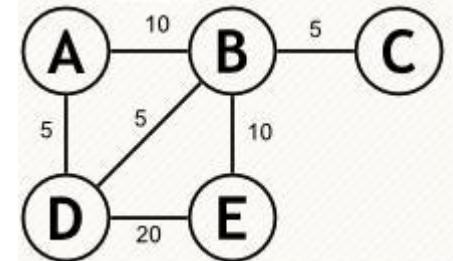


## Floyd-Warshall Örnek

- Şimdi algoritmanın 2. adımına geçerek yolların tutulduğu bu matrisi adım adım güncelleyelim:

- A B C D E
- A 0 10  $\infty$  5  $\infty$
- B 10 0 5 5 10
- C  $\infty$  5 0  $\infty$   $\infty$
- D 5 5  $\infty$  0 20
- E  $\infty$  10  $\infty$  20 0

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$



## Floyd-Warshall Örnek

- B üzerinden atlanarak ulaşılan düğümleri güncelleyelim
  - A B C D E
  - A 0 10 15 5 20
  - B 10 0 5 5 10
  - C 15 5 0 10 15
  - D 5 5 10 0 15
  - E 20 10 15 15 0

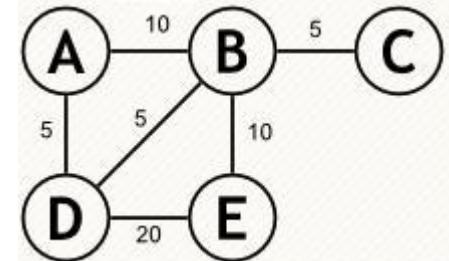
## Floyd-Warshall Örnek

- C üzerinden atlanan düğümler:

- A B C D E
- A 0 10 15 5 20
- B 10 0 5 5 10
- C 15 5 0 10 15
- D 5 5 10 0 15
- E 20 10 15 15 0

- D üzerinden atlanan düğümler:

- A B C D E
- A 0 10 15 5 20
- B 10 0 5 5 10
- C 15 5 0 10 15
- D 5 5 10 0 15
- E 20 10 15 15 0



## Floyd-Warshall Örnek

- E üzerinden atlanan düğümler:
  - A B C D E
  - A 0 10 15 5 20
  - B 10 0 5 5 10
  - C 15 5 0 10 15
  - D 5 5 10 0 15
  - E 20 10 15 15 0
- Yukarıda son elde edilen bu matriste görüldüğü üzere herhangi bir düğümden diğer bütün düğümlere giden en kısa yollar çıkarılmıştır. Örneğin A düğümünden E'ye 20 uzunlığında veya C düğümünden D'ye 10 uzunlığında yolla gidilebilir.
- Yukarıdaki matrislerde diyagona göre simetri bulunmasının sebebi grafın yönsüz graf (undirected graph) olmasıdır. Şayet graf yönlü graf (directed graph) olsaydı bu simetri bozulurdu (tabi yönlerin ağırlıklarının aynı olmaması durumunda).

