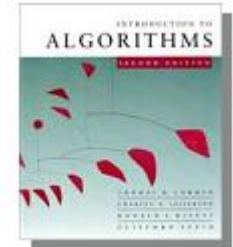
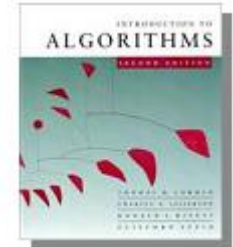


## **6.Hafta Bilinen Probleme İndirgeme Tasarım Yöntemi**



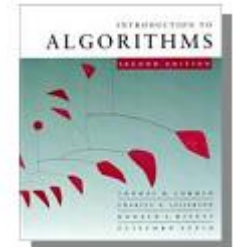
## Bilinen Probleme İndirgeme

- 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 işlemdir, 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;



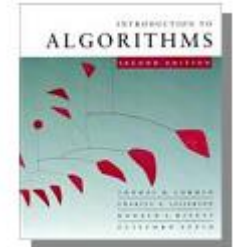
# Bilinen Probleme İndirgeme

## ○ I. YOL

► A parametresi içinde tekrar eden sayıların olup olmadığının kontrol edileceği dizidir ve B parametresinin i. elemanı A dizisinin i. elemanından kaç tane olduğunu tutan bir dizidir.

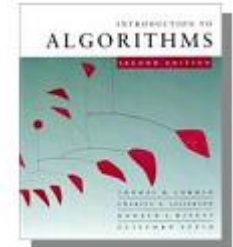
TekrarBul(A,B)

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

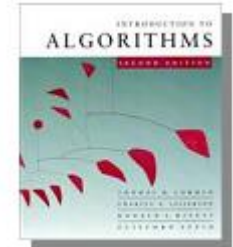
- I. YOL: En kötü çalışma zamanı
- Bütün sayılar ikili olarak karşılaştırılırlar.
- 1. sayı için  $n-1$  karşılaştırma
- 2. sayı için  $n-2$  karşılaştırma
- ....
- $(n-1)$ . sayı için 1 karşılaştırma yapılır.
- Bunun sonucunda elde edilen karşılaştırma toplamaları
- $(n-1)+(n-2)+\dots+1 = \frac{n(n-1)}{2}$
- $T(n)=O(n^2)$  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

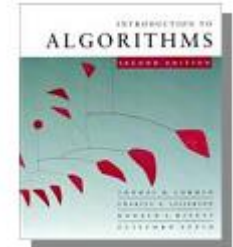


- **II. YOL: Bilinen probleme dönüştürme**
- Aynı problemin bilinen bir yöntemle çözülmesi daha iyi sonuç verebilir. Bilinen problem sıralama işlemidir. A dizisi içindeki bütün sayılar sıralanır ve ondan sonra birinci elemandan başlanarak sona doğru ardışıl olan elemanlar karşılaştırılır. Bu şekilde 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

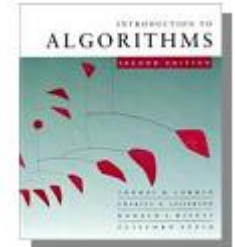


## ○ II. YOL

- Bu algoritma iki kısımdan oluşmaktadır. Birinci kısmı A dizisinin sıralanması ve ikinci kısımda ise bir döngü ile tekrar sayısının bulunması işlemidir. YığınSıralama algoritmasının mertebesinin  $T_1(n) = \Theta(n \lg n)$  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  $T(n)$
- $T(n) = T_1(n) + T_2(n)$
- $= \Theta(n \lg n) + \Theta(n)$
- $= \Theta(n \lg n)$
- 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

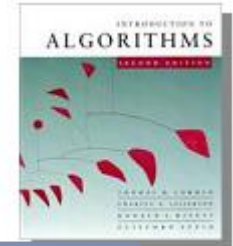


# Bilinen Probleme İndirgeme



- **Örnek 2:**
- İki boyutlu bir uzayda  $n$  tane noktadan hangi üç noktanın aynı doğru üzerinde olup olmadığı kontrolü yapılmak 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

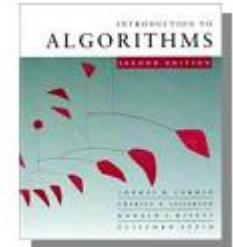
▷ **A parametresi, her elemanı iki tane gerçel sayıdan oluşan bir iki boyutlu uzay noktaları kümesidir.**

**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



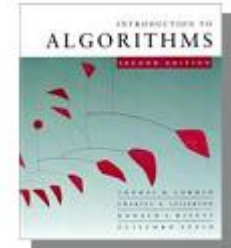
## II. YOL

- 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



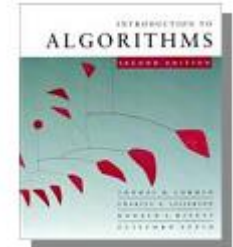
## ○ II. YOL :Bilinen probleme indirgeme

► 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.

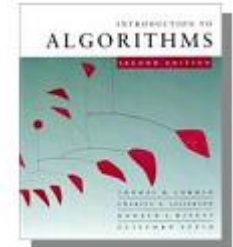
Dogru\_Uz\_Noktalar(A)

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

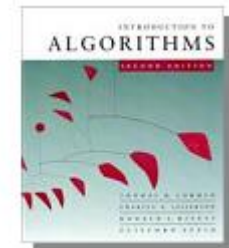


- II. YOL
- C dizisindeki elemanlar kendisi ile aynı endekse sahip B dizisinin o elemanından kaç tane olduğunu tutmaktadır.
- Bu algoritmanın mertebesi hesaplanacak olursa, algoritmada üç parçadan oluşan bir zaman bağıntısı elde edilir.
- İlk parça eğimleri hesaplama zamanı ve bu zaman  $T_1(n)$  olsun.
- 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 değinilmişti.
- **Örnek 4:** Bir kümenin maksimum ve minimum elemanlarının belirlenmesi için gerekli algoritmanın kaba kodunu yazınız.



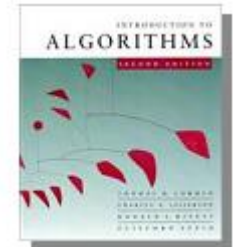
# Bilinen Probleme İndirgeme

- 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.



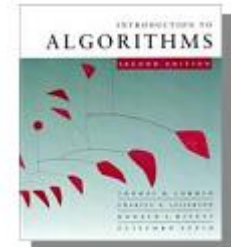
- Daha sonra her bir çift karşılaştırılır.  $\lfloor n/2 \rfloor, \lceil n/2 \rceil$



● = 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

$S = S1 \cup S2$

$(\min1, \max1) \leftarrow \text{MaxMin}(S1)$

$(\min2, \max2) \leftarrow \text{MaxMin}(S2)$

if  $\min1 \leq \min2$  then sonuç( $\min = \min1$ )

else sonuç( $\min = \min2$ )

if  $\max \geq \max2$  then sonuç ( $\max = \max1$ )

else sonuç ( $\max = \max2$ )  $T(n) = O(n)$

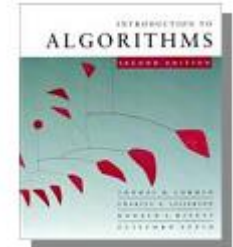
n için Sıkı sınır

$$T(n) = \lceil 3n/2 \rceil - 2$$

```

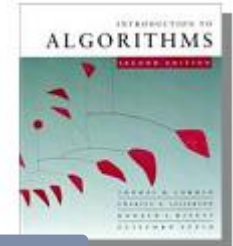
MAXMIN(i, j, fmax, fmin)
1 if (i=j)
2   then fmax ← fmin ← a[i]
3 if (i=(j-1)) then
4   if a[i]<a[j]
5     then fmax ← a[j]
6         fmin ← a[i]
7   else fmax ← a[i]
8         fmin ← a[j]
9 else
10   mid ← ⌊(i+j)/2⌋
11   MAXMIN(i, mid, gmax, gmin)
12   MAXMIN(mid+1, j, hmax, hmin)
13   fmax ← max{gmax, 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 gerekir. Binada kirişler ve kolonlar ayırıt olarak düşünüldüğünde, kiriş ve kolonların birleştiği noktalar da düğüm olarak düşünülebilir. Bu şekilde binanın çizgesi çıkarılmış olur. Binaya yerleştirilecek kameraların görmediği kiriş veya kolon kalmamalı. Kiriş ve kolonlar ayırıt olduklarına göre çözüm minimum-düğüm kapsama probleminin çözümü olur. Binayı modelleyen çizge  $G=(V,E)$  olmak üzere problemin çözümü aşağıdaki algoritma ile yapılır.
- (Graflara sonra değinilecektir)

# 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 düğümüne çakışık olan ayrıtların hepsini sil.



# **Amortize Edilmiş Analiz Amortized Analysis**

- Dinamik Tablolar
- Birleşik Metod
- Hesaplama Metodu
- Potansiyel Metodu

## Amortize edilmiş çözümleme

- **Amortize edilmiş analiz**, bir dizi işlem içindeki tek bir işlemin pahalı olması durumunda bile, ortalama işlem maliyetinin küçük olduğunu göstermek için kullanılan bir yöntemdir.
- Ortalama durum analizlerinden olasılık içermemesi bakımından farklılık gösterir. Ortalama durum analizinde bütün ihtimallerden ortalama maliyette bir ihtimal üzerinden analiz yapılır. Bütün ihtimaller hesaplanarak ortalamaları bulunur.
- Amortize edilmiş çözümleme ise, her bir işlemin en kötü durumdaki ortalama performansını garanti eder ve en kötü durumun beklenenden daha iyi olduğunu ispatlar.

## Amortize edilmiş çözümleme

- Amortize analiz çözümlerinin üç metodu vardır
  1. **Aggregate Metod** (Toplam, Birleşik veya Topak Metodu)
  2. **Accounting Metod** (Hesaplama veya Muhasebe Metodu)
  3. **Potantiel Metod** (Potansiyel (kapasite) Metodu)
- Birleşik metodu basit olmasına karşın, diğer iki metodun kesinliğine sahip değildir. Hesaplama ve potansiyel metotları, her bir işleme belli bir ***amortize edilmiş maliyet*** atanmasına izin verir.
- Birleşik metodunda ise bir işlemin gerçekte neye mal olduğu belirlenemiyor. Temelde  $n$  işlemin ne kadar süre aldığını çözümlemidir.

## Amortize edilmiş çözümleme çeşitleri

- **Aggregate Metod (Birleşik Metodu):**
- Her  $n$  için,  $n$  tane işlemin en kötü durum zamanı  $T(n)$  olur. Ortalama maliyet veya amortize maliyet  $T(n)/n$  olur. Birden fazla işlem olsa dahi amortize maliyet her işlem başına olan maliyettir.
- Başlangıçta  $D(0)$  gibi bir veri vardır ve bu veri boştur. Aynı zamanda verilere erişmek için işlemler kümesi bulunmaktadır.
- $D(1)$  verisine ulaşmak için bir işlem seçilir.
- $D(0) \rightarrow D(1) \rightarrow D(2) \rightarrow \dots \rightarrow D(n)$
- $T(n)$ : Toplam en kötü durum zamanı olsun ve buradan işlem başına zaman  $T(n)/n$  olur.

## Amortize edilmiş çözümleme çeşitleri

- Örnek: Yığın
- Yığın veri yapısına yeni bir işlem eklenmesi ile ele alınsın.
- $PUSH(S,x) \rightarrow \Theta(1)$
- $POP(S) \rightarrow \Theta(1)$
- Her birinin maliyeti 1 olduğu kabul edilsin.  $n$  tane PUSH ve POP işleminin maliyeti  $n$  olur.
- $T(n) = \Theta(n)$
- olur. Böylece bir işlemin maliyeti  $\frac{T(n)}{n} = O(1)$  olur.



## Amortize edilmiş çözümleme çeşitleri

- Örnek: İkili sayıcıyı artırma
- Sıfırdan başlayan ve k-bit olan bir sayıcının değerinin artırılması problemi düşünölsün  $A[0 \dots k-1]$  : bitler dizisi ve  $\text{Uzunluk}(A)=k$  olur. Sayıcıda saklanan değör x olsun ve en önemsiz biti  $A[0]$  olur ve en önemli biti  $A[k-1]$ 'de olur. Bu sayıcıda herhangi bir andaki sayısal değör

$$x = \sum_{j=0}^{k-1} A[j] * 2^j$$

- olur. Başlangıçta  $x=0$  dır ve böylece  $j=0,1,2,3,\dots,k-1$  için  $A[j]=0$  olur. Bu sayıya  $1 \pmod{2^k}$  sayısının sayıcıya eklenmesi için aşağıdaki algoritma kullanılır.

## Amortize edilmiş çözümleme çeşitleri

- Örnek: İkili sayıcıyı artırma

Artır (A)

► Yerel Değişkenler: A dizisi.

1.  $j \leftarrow 0$
2. for  $j < \text{uzunluk}(A)$  ve  $A[j]=1$  olduğu sürece
3.      $A[j] \leftarrow 0$
4.      $j \leftarrow j+1$
5. if  $j < \text{uzunluk}(A)$  ise
6.      $A[j] \leftarrow 1$

- Bu algoritmanın analizi yapılacak olursa, en kötü durumda bir sefer artırmanın maliyeti  $\Theta(k)$  olur. En kötü durum A dizisinin bütün elemanlarının 1 olduğunda ortaya çıkar. Böylece n tane artırma işleminin maliyeti  $O(n^k)$  olur. Bu sınır değeri geniş bir aralık içerir. Daha dar bir aralık belirlenebilir.

## Amortize edilmiş çözümleme çeşitleri

- Örnek: İkili sayıcıyı artırma

- İlk adım  $A[0] \leftarrow 1$  olur  $x=1$   $A[0] \rightarrow 0$
- İkinci adım
- $A[0] \leftarrow 0$
- $A[1] \leftarrow 1$   $x=2$   $A[1] A[0] \rightarrow 10$
- Üçüncü adım
- $A[0] \leftarrow 1$   $x=3$   $A[1] A[0] \rightarrow 11$
- Dördüncü adım
- $A[0] \leftarrow 0$
- $A[1] \leftarrow 0$
- $A[2] \leftarrow 1$   $x=4$   $A[2] A[1] A[0] \rightarrow 100$

## Amortize edilmiş çözümleme çeşitleri

- **Örnek: İkili sayıcıyı artırma**
- A[0] her defasında değer değiştirir. n sefer artırma işleri yapılırsa A[1] elemanı  $\lfloor n/2 \rfloor$  sefer değer değiştirir. A[2] elemanı  $\lfloor n/4 \rfloor$  defa değer değiştirir.
- Eğer  $j > \lfloor \log n \rfloor$  ise A[j] hiçbir zaman değer değiştirmez. Toplam değer değişimi

$$\sum_{j=0}^{\lfloor \lg n \rfloor} \left\lfloor \frac{n}{2^j} \right\rfloor < n \sum_{j=0}^{\infty} \frac{1}{2^j} = 2n$$

- Böylece toplam n adım zamanının en kötü durum zamanı  $O(n)$  olur ve bir adım zamanı

$$\frac{O(n)}{n} = O(1)$$

- olur.

## Amortize edilmiş çözümleme çeşitleri

- **Accounting Metod (Hesaplama Metodu):**
- Her işlemin gerçek maliyeti yüklenir (Farklı işlemlere, farklı maliyetler yüklenebilir). Daha sonra bu işlemler için hayali bir amortize maliyet tanımlanır.
- $c_i = \text{gerçek maliyet},$   $\hat{c}_i = \text{amortize maliyet}$
- Amortize maliyeti, gerçek maliyeti geçtiği zaman, aradaki fark kredi denilen bir veri yapısına yüklenir. Bu değer daha sonraki işlemlerde amortize maliyeti, gerçek maliyetten küçük olan işlemlere eklenir.
- Böylece amortize maliyeti gerçek maliyet ve kredi arasında paylaştırılır(ya depolanır yada tüketilir). Birleşik yöntemde bütün işlemlerin maliyeti aynı iken, burada farklıdır.

## Amortize edilmiş çözümleme çeşitleri

- Accounting Metod (Hesaplama Metodu):

$$\sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i$$

- Gerçek maliyetler toplamı amortize maliyet toplamlarından büyük olamaz.
- Amortize maliyet çok dikkatli seçilmelidir. En kötü durumda her işlemin maliyetinin küçük(ortalama maliyet) olduğu gösterilirse, toplam amortize maliyet bir üst sınır belirlemelidir. Kredilerin negatif olmamasına dikkat edilmelidir.

## Amortize edilmiş çözümleme çeşitleri

- Örnek: Yığıt işlemler
- Yığıt veri yapısının boyutu  $S$  olmak üzere her ekleme ve çıkarma maliyeti
- $PUSH(S,x) \rightarrow 1$
- $POP(S) \rightarrow 1$
- olur. Amortize maliyetler ise
- $PUSH(S,x) \rightarrow 2$
- $POP(S) \rightarrow 0$
- Her  $PUSH$  işleminde 1 birim maliyet için harcanırken, 1 birim maliyet kredi olarak o elemanda saklanır. Böylece  $n$  tane  $PUSH$  ve  $POP$  işleminin maliyeti  $O(n)$  olur.

## Amortize edilmiş çözümleme çeşitleri

- Örnek : İkili sayıcı artırma
- Sayıcının ilk değeri sıfır olmak üzere bu problemde icra zamanı değer değiştiren bit sayısı ile orantılıdır.
- Her bit değişiminin ( $0 \rightarrow 1$ ) maliyeti 2 olsun.
- Her bitin  $0 \rightarrow 1$  değişiminde 1 birim maliyet gerçek maliyet için harcanırken, 1 birim maliyet kredi olarak saklanır ve bu değer  $1 \rightarrow 0$  işleminin maliyetini karşılar.  $n$  tane artışın maliyeti  $O(n)$  olur.



## Amortize edilmiş çözümleme çeşitleri

- **Potansiyel Metot:**

- Potansiyel metot da, belli bir değerde amortize maliyet (kredi) tutmak yerine amortize maliyetin işlemlerin potansiyel enerjisi ve gerçek maliyet toplamı olduğu kabul edilir. Bu enerji belli bir veri yapısına değil, bütün veriye aittir.
- İlk olarak  $D_0$  gibi bir veri ile başlanır ve  $n$  tane işlem uygulanacaktır.  $i=1,2,3,\dots,n$  için  $i$ . işlemin gerçek maliyeti  $c_i$  olsun ve  $D_i'$  de  $i$  tane işlemin  $D_0$  uygulanmasından elde edilen veri değeri olsun ve

$$D_i \xleftarrow{i} D_{i-1} \xleftarrow{i-1} D_{i-2} \xleftarrow{i-2} \dots \xleftarrow{2} D_1 \xleftarrow{1} D_0$$

- şeklinde gösterilir.

## Amortize edilmiş çözümleme çeşitleri

- $\Phi$  : *Potansiyel fonksiyonu* ve  $\Phi: D_i \rightarrow \Phi(D_i) \in R$  şeklinde tanımlanır ve  $D_i$  veri yapısının potansiyelini verir.
- Potansiyel fonksiyonu verinin değerini veya durumunu parametre olarak alır ve sonuç olarak bir gerçel sayı verir ve bu sayı değeri verinin potansiyelini gösterir.  $i$  işlemi için amortize maliyet  $\hat{c}_i$  ile gösterilir ve
- $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$  şeklinde tanımlanır.

## Amortize edilmiş çözümleme çeşitleri

- Amortize maliyet, gerçek maliyet ile veri yapısındaki potansiyel artışının toplamına eşittir. **n tane işlemin amortize maliyeti,**

$$\sum_{i=1}^n \hat{C}_i = \underbrace{\sum_{i=1}^n (C_i + \Phi(D_i) - \Phi(D_{i-1}))}_{\text{teleskop serisi}} \quad \text{şeklinde tanımlanır ve}$$

$$\sum_{i=1}^n \hat{C}_i = \sum_{i=1}^n C_i + \Phi(D_n) - \Phi(D_0) \quad \text{olur.}$$

## Amortize edilmiş çözümleme çeşitleri

- Eğer  $\Phi$  potansiyel fonksiyonu  $\Phi(D_n) \geq \Phi(D_0)$  şeklinde tanımlanabilirse,
- $\sum_{i=1}^n \hat{C}_i$
- üst sınırı teşkil eder. Pratikte her zaman ne kadar işlem gerçekleştirileceği bilinmediğinden bütün i'ler için
- $\Phi(D_i) \geq \Phi(D_0)$
- olmalıdır.
- Genellikle  $\Phi(D_0)=0$  olarak tanımlanır ve  $\forall i, \Phi(D_i) \geq 0$  olur.

## Amortize edilmiş çözümleme çeşitleri

- Eğer  $\Phi(D_i) - \Phi(D_{i-1}) > 0$  ise,  $\hat{c}_i'$  de aşırı artış var demektir ve verinin potansiyeli artar.
- Eğer  $\Phi(D_i) - \Phi(D_{i-1}) < 0$  ise,  $\hat{c}_i'$  de aşırı azalma var demektir ve verinin potansiyeli her işlemten (i. işlem) sonra azalır.

## Amortize edilmiş çözümleme çeşitleri

- Örnek Yığıt işlemleri,
- $\Phi$ : yığıtta bulunan eleman sayısını versin.  $D_0$  için  $\Phi(D_0)=0$  olur, çünkü yığıt boştur.
- Böylece  $\forall i$  için  $\Phi(D_i) \geq 0$  olur.  $n$  tane işlemin amortize maliyeti bir üst sınır teşkil eder. Eğer  $i$ . işlem PUSH ise ( $s$  tane eleman var)
- $\Phi(D_i)-\Phi(D_{i-1}) = (s+1)-s = 1$  olur.
- Amortize maliyet  $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- $= 1 + 1$
- $= 2$  olur.

## Amortize edilmiş çözümleme çeşitleri

- Eğer  $i$ . işlem POP ise,  $\Phi(D_i) - \Phi(D_{i-1}) = (s-1) - s = -1$  olur.  
Amortize maliyeti
- $\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$
- $= 1 - 1$
- $= 0$  olur.
- Her iki işlemin amortize maliyeti  $O(1)$  olduğundan  $n$  tane işlem için toplam amortize maliyet  $O(n)$  olur.  
 $\Phi(D_i) \geq \Phi(D_0)$  olduğundan toplam amortize maliyet üst sınır teşkil eder. Böylece  $n$  tane işlemin en kötü durum maliyeti  $O(n)$  olur.

## Amortize edilmiş çözümleme çeşitleri

- Örnek: İkili sayıcı artırma
- Sayıcı için potansiyelin tanımlanması gerekir.  $i$ . artırma işleminin potansiyeli  $b_i$  olsun ve  $b_i$  sayıcıdaki 1 sayılarının toplamıdır. Daha sonra artırma işleminin amortize maliyeti hesaplınsın.
- $i$ . işlemde  $t_i$  tane  $1 \rightarrow 0$  olsun. Bu işlemin gerçek maliyeti en fazla  $t_{i+1}$  olur.  $i$ . işlemden sonra sayıcıdaki 1 sayısı
- $b_i \leq b_{i-1} - t_i + 1$
- olur ve potansiyel farkı
- $\Phi(D_i) - \Phi(D_{i-1}) \leq (b_{i-1} - t_i + 1) - b_{i-1}$
- $= 1 - t_i$  olur.



## Amortize edilmiş çözümleme çeşitleri

- Amortize maliyet,
- $\hat{C}_i = C_i + \Phi(D_i) - \Phi(D_{i-1})$
- $\leq (t_i + 1) + (1 - t_i)$
- $= 2$  olur.
- Sayıcı sıfırdan başlarsa  $\Phi(D_0) = 0$  olur.  $\forall i$  için  $\Phi(D_i) \geq 0$  olduğundan  $n$  tane artırma işleminin amortize maliyet toplamı, toplam gerçek maliyet için bir üst sınır teşkil eder. Böylece  $n$  tane işlemin en kötü durum üst sınırı  $O(n)$  olur.

## Amortize edilmiş çözümleme çeşitleri

- Eğer sayıcı sıfırdan başlamazsa,  $b_0$  tane 1 vardır.  $n$  tane artırma işleminden sonra  $b_n$  tane 1 olur ve  $0 \leq b_0, b_n \leq k$  olur.

$$\sum_{i=1}^n C_i = \sum_{i=1}^n \hat{C}_i - \Phi(D_n) + \Phi(D_0)$$

- $\hat{C}_i \leq 2$  olduğundan ve  $\Phi(D_0) = b_0$  ve  $\Phi(D_n) = b_n$  gerçek maliyet ( $n$  tane işlemden sonra)

$$\begin{aligned} \sum_{i=1}^n C_i &\leq \sum_{i=1}^n 2 + b_n - b_0 \\ &= 2n - b_n + b_0 \end{aligned}$$

- olur.  $b_0 \leq k$  olduğundan eğer  $n = \Omega(k)$  işlem uygulanırsa, toplam gerçek maliyet  $O(n)$  olur.

## Dinamik Tablolar

- Bazı uygulamalarda tabloda kaç tane nesne olduğu bilinmez. Zaman içerisinde tablonun boyutu az gelirse, boyutu daha büyük olan bir tablo oluşturulur ve yeni değerler ile eski tablodaki değerler yeni tabloya kopyalanır.
- Benzer şekilde tablodan değer silindiğinde, tablonun boyutu küçülebiliyorsa, tablonun boyutu yarıya indirilir.
- Amaç, bu tablolara eleman ekleme ve eleman silme işlemlerinin amortize maliyetlerinin  $O(1)$  olduğunun gösterilmesidir.

## Dinamik Tablolar

- Gerçek maliyetlere gelince tablonun genişlemesi veya daralması işlemlerinin maliyeti  $O(1)$ 'den daha büyüktür.
- Hash (Kıyım) fonksiyonunun kullanıldığını kabul edin. Hash tabloları ne kadar büyük olmalıdır?
- Mümkün olduğu kadar büyük(zaman)
- Mümkün olduğu kadar küçük(alan)
- $n$  değerleri için  $O(n)$

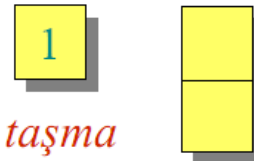
## Dinamik Tablolar

### Kıyım tablosu ne kadar büyük olmalı?

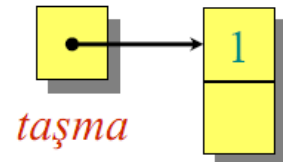
- **Amaç** : Tabloyu olabildiğince küçük yapın, ama yeterince de büyük olmalı ki taşma olmasın.(Aksi takdirde verimsiz olur.)
- **Problem** : Uygun boyutun ne olması gerektiğini önceden bilemiyorsak ne olacak?
- **Çözüm** : Dinamik Tablolar
- **FİKİR** : Her ne zaman tablo taşarsa, (**malloc** veya **new** kullanarak) yeni ve daha büyük bir tablo oluşturun. Eski tablodaki bütün elemanları yenisine taşıyıp, eski tablonun depolama yerini boşaltın.

# Dinamik Tablo Örneği

1. Insert/ Ekle

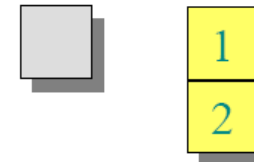


2. Insert /Ekle



1. Ekle

2. Ekle

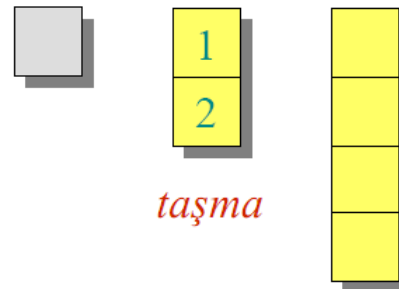


1. Ekle

2. Ekle

1. Ekle

2. Ekle



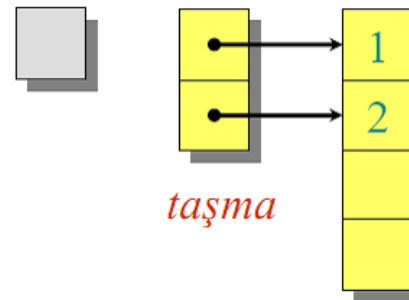
3. Ekle

## Dinamik Tablo Örneği

1. Ekle

2. Ekle

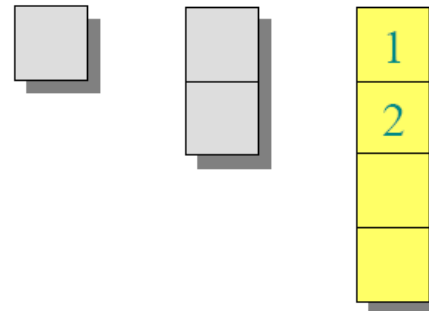
3. Ekle



1. Ekle

2. Ekle

3. Ekle



## Dinamik Tablo Örneği

1. Ekle

2. Ekle

3. Ekle

4. Ekle

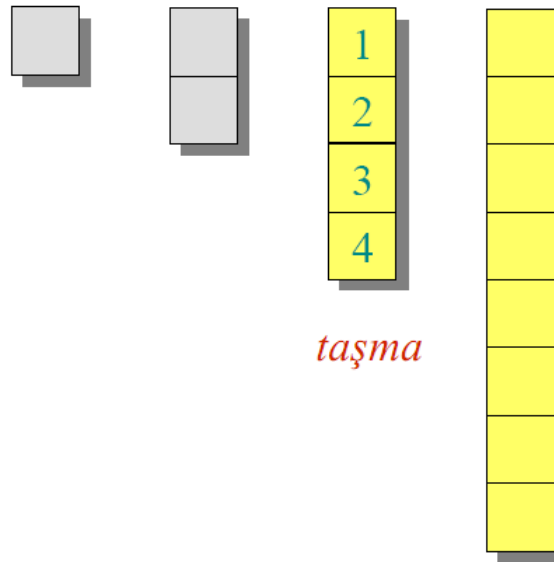
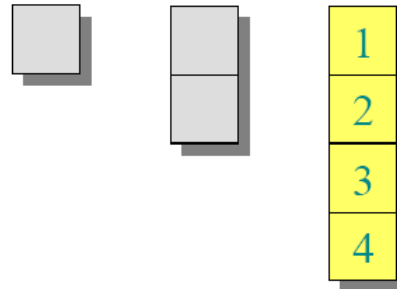
1. Ekle

2. Ekle

3. Ekle

4. Ekle

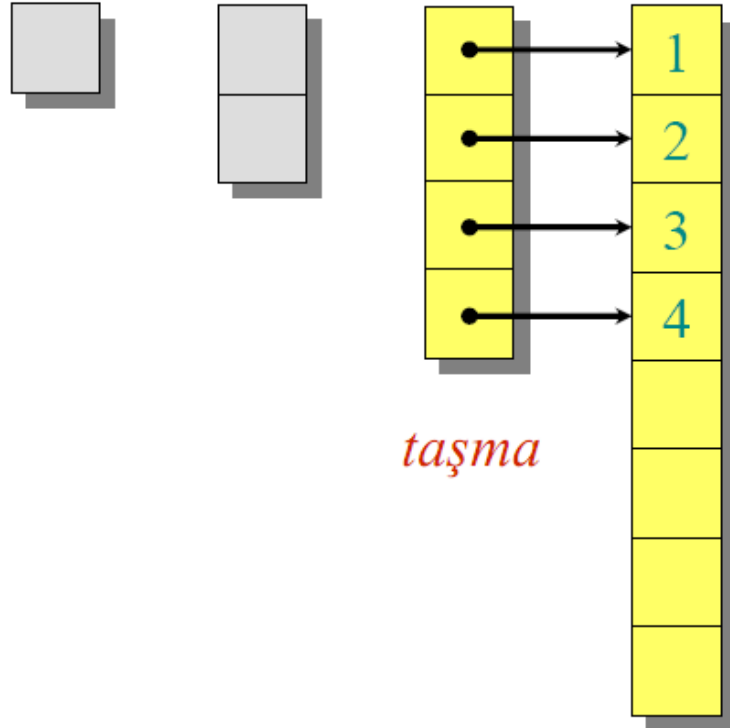
5. Ekle





## Dinamik Tablo Örneği

1. Ekle
2. Ekle
3. Ekle
4. Ekle
5. Ekle



## Dinamik Tablo Örneği

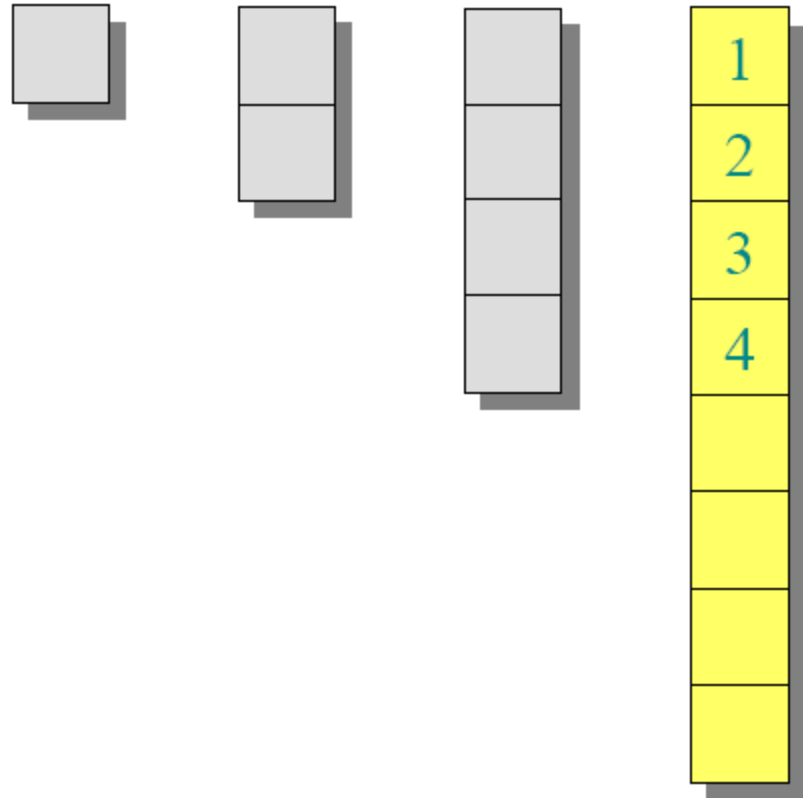
1. Ekle

2. Ekle

3. Ekle

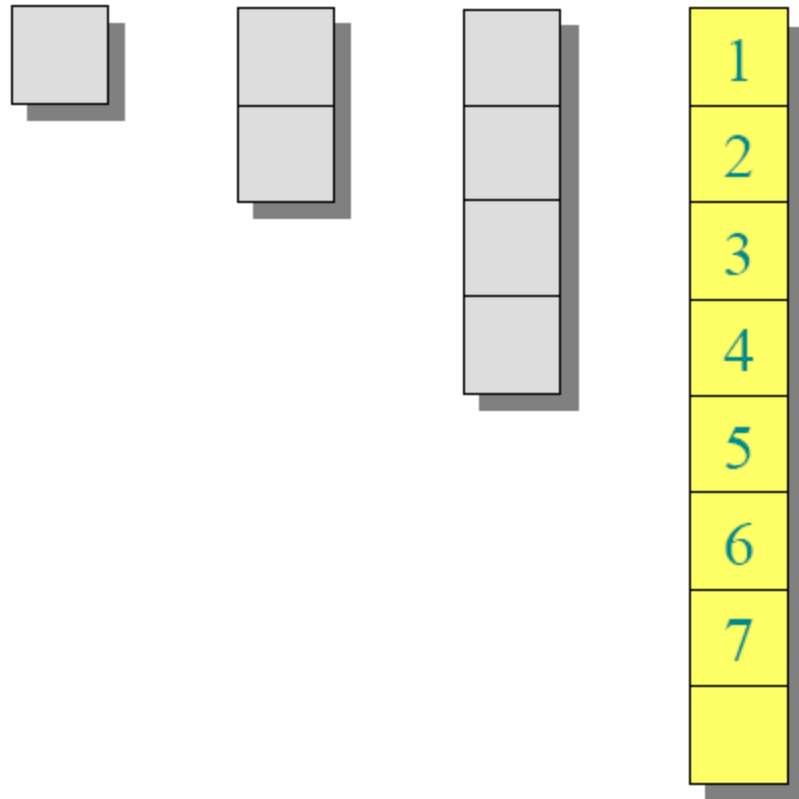
4. Ekle

5. Ekle



## Dinamik Tablo Örneği

1. Ekle
2. Ekle
3. Ekle
4. Ekle
5. Ekle
6. Ekle
7. Ekle



## En kötü durum çözümlemesi

$n$  tane ekleme olduğunu düşünün. Bir eklemeyi yapmak için gerekecek en kötü zaman  $\Theta(n)$ . Dolayısıyla  $n$  tane ekleme için en kötü zaman  $n \cdot \Theta(n) = \Theta(n^2)$ .

**YANLIŞ!** Aslında  $n$  tane ekleme için gerekecek en kötü durum süresi sadece  $\Theta(n) \ll \Theta(n^2)$ 'dir.

Neden olduğunu görelim!

## Daha sıkı bir çözümleme (Topak veya Birleşik Metoda göre)

$c_i$  =  $i$ 'nci eklemenin maliyeti olsun.

$$= \begin{cases} i & \text{eğer } i-1, 2\text{'nin tam kuvvetiyse,} \\ 1 & \text{Aksi takdirde} \end{cases}$$

| $i$          | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  | 10 |
|--------------|---|---|---|---|---|---|---|---|----|----|
| $büyüklik_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$        | 1 | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9  | 1  |

## Birleşik Metoda göre

$c_i$  =  $i$ 'nci eklemenin maliyeti olsun.

$$= \begin{cases} i & \text{eğer } i-1, 2\text{'nin tam kuvvetiyse,} \\ 1 & \text{Aksi takdirde} \end{cases}$$

| $i$      | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  | 10 |
|----------|---|---|---|---|---|---|---|---|----|----|
| $size_i$ | 1 | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$    | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1  | 1  |
|          |   | 1 | 2 |   | 4 |   |   |   | 8  |    |

## Birleşik veya topak(aggregate) Metoda göre amortize analiz

$n$  tane eklemenin maliyeti  $= \sum_{i=1}^n c_i$

Bir dizi işlemde bir veya birden fazla işlem pahalı olsa bile ortalama işlem maliyetinin küçük olduğu görülür. Genelde ortalama durum hesaplanırken olasılık kullanılır. Amortize analizde ise olasılık yok.  $n$  tane işlem en kötü durumda sabit miktarda zamana mal olur.  $\longrightarrow$

$$\leq n + \sum_{j=0}^{\lfloor \lg(n-1) \rfloor} 2^j$$

$$\leq 3n$$

$$= \Theta(n).$$

Bu durumda, her bir dinamik tablo işleminin ortalama maliyeti  $\Theta(n)/n = \Theta(1)$ .

## Hesaplama (Accounting) metodu

- $i$ 'nci işleme, hayali *amortize edilmiş* bir  $\hat{c}_i$  maliyeti yükler, 1 birim işe 1\$ öder. (örnek: zaman)
- Bu miktar işlemin yapılması için harcanır.
- Harcanmayan her hangi bir ödeme, **banka**da daha sonraki işlemlerde kullanılmak üzere saklanır.
- Banka hesabı hiçbir zaman eksi olamaz. Bütün  $n$ 'ler için

$$\sum_{i=1}^n c_i \leq \sum_{i=1}^n \hat{c}_i$$

olduğundan emin olmalıyız.

- Toplam amortize edilmiş maliyet, gerçek bütün maliyetler için bir üst sınır oluşturur.



# Dinamik tabloların hesaplama çözümlemesi

$i$ 'nci ekleme için **amortize edilmiş** bir  $\hat{c}_i = \$3$  maliyeti ekleyin.

- \$1 anında ekleme için harcanır.
- \$2 ise daha sonraki tablo büyütme (iki katına) işlemleri için saklanır.

Tablo iki katına çıkartıldığında \$1 son elemanın yerleştirilmesi, \$1 da eski bir elemanın yerleştirilmesi için harcanır.

### ÖRNEK:

|     |     |     |     |     |     |     |     |
|-----|-----|-----|-----|-----|-----|-----|-----|
| \$0 | \$0 | \$0 | \$0 | \$2 | \$2 | \$2 | \$2 |
|-----|-----|-----|-----|-----|-----|-----|-----|

*taşma*

[illegible]

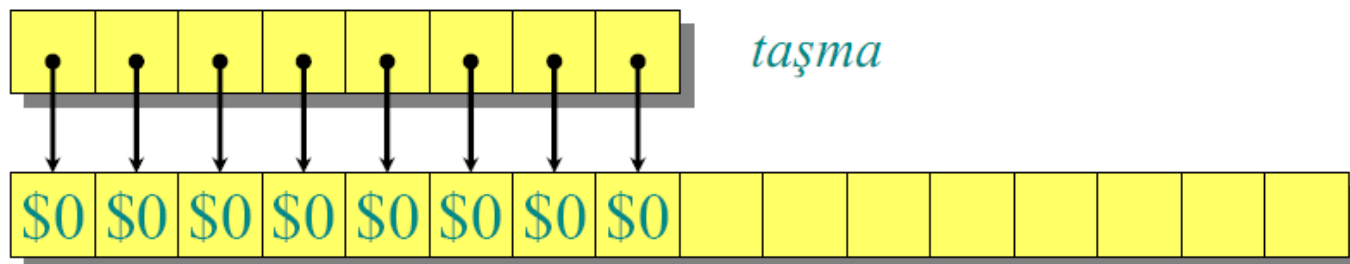
## Dinamik tabloların hesaplama çözümlemesi

$i$ 'nci ekleme için *amortize edilmiş* bir  $\hat{c}_i = \$3$  maliyeti ekleyin.

- \$1 anında ekleme için harcanır.
- \$2 ise daha sonraki tablo büyütme (iki katına) işlemleri için saklanır.

Tablo iki katına çıkartıldığında \$1 son elemanın yerleştirilmesi, \$1 da eski bir elemanın yerleştirilmesi için harcanır.

### ÖRNEK:



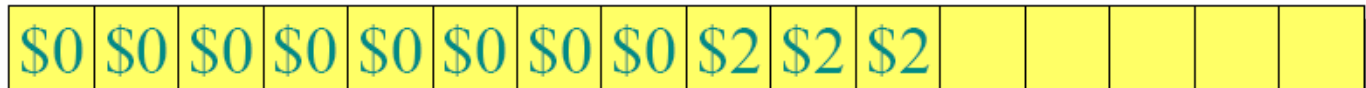
## Dinamik tabloların hesaplama çözümlemesi

$i$ 'nci ekleme için *amortize edilmiş* bir  $\hat{c}_i = \$3$  maliyeti ekleyin.

- \$1 anında ekleme için harcanır.
- \$2 ise daha sonraki tablo büyütme (iki katına) işlemleri için saklanır.

Tablo iki katına çıkartıldığında \$1 son elemanın yerleştirilmesi, \$1 da eski bir elemanın yerleştirilmesi için harcanır.

### ÖRNEK:



## Hesaplama çözümlemesi (devamı)

**Anahtar değişmezi:** Banka hesabı hiçbir zaman 0'ın altına düşemez. Amortize edilmiş maliyetlerin toplamı, gerçek maliyetler için bir üst sınır sağlar.

| $i$          | 1  | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9  | 10 |
|--------------|----|---|---|---|---|---|---|---|----|----|
| $büyüklik_i$ | 1  | 2 | 4 | 4 | 8 | 8 | 8 | 8 | 16 | 16 |
| $c_i$        | 1  | 2 | 3 | 1 | 5 | 1 | 1 | 1 | 9  | 1  |
| $\hat{c}_i$  | 2* | 3 | 3 | 3 | 3 | 3 | 3 | 3 | 3  | 3  |
| $banka_i$    | 1  | 2 | 2 | 4 | 2 | 4 | 6 | 8 | 2  | 4  |

\* İlk işlem \$3'a değil, \$2'a mal olur.

$$c_i = \begin{cases} i & \text{eğer } i-1, 2\text{'nin tam kuvvetiyse,} \\ 1 & \text{Aksi takdirde} \end{cases}$$

## Potansiyel metodu

**FİKİR :** Banka hesabını dinamik kümenin potansiyel enerjisi olarak düşünün.

### Çerçeve :

- Bir  $D_0$  başlangıç veri yapısı seçin.
  - $i$  işlemi  $D_{i-1}$ 'i  $D_i$ 'ye çevirir.
  - $i$  işleminin maliyeti  $c_i$ 'dir.
  - Bir potansiyel fonksiyon tanımlayın.  $\Phi : \{D_i\} \rightarrow \mathbb{R}$ .
- Öyle ki bütün  $i$ 'ler için;  $\Phi(D_0) = 0$  ve  $\Phi(D_i) \geq 0$
- $\Phi$  ye göre amortize edilmiş maliyet şu şekilde tanımlanır:
 
$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1})$$

## Potansiyelleri Anlamak

$$\hat{c}_i = c_i + \underbrace{\Phi(D_i) - \Phi(D_{i-1})}_{\text{Potansiyel fark } \Delta\Phi_i}$$

- Eğer  $\Delta\Phi_i > 0$  ise  $\hat{c}_i > c_i$  dir.  $i$  işlemi; işi, daha sonra kullanmak üzere veri yapısında saklar.
- Eğer  $\Delta\Phi_i < 0$  ise  $\hat{c}_i < c_i$  dir. Veri yapısı, daha önce saklanan işi,  $i$  işleminde kullanılmak üzere getirir.

## Amortize edilmiş maliyetler, gerçek maliyet için sınır oluşturur.

$n$  tane işlem için amortize edilmiş maliyet şudur :

$$\sum_{i=1}^n \hat{c}_i = \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1}))$$

İki tarafı da topluyoruz...

$$\begin{aligned} \sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\ &= \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0) \end{aligned}$$

Seri teleskop gibi uzar...

## Amortize edilmiş maliyetler, gerçek maliyet için sınır oluşturur.

$n$  tane işlem için amortize edilmiş maliyet şudur :

$$\begin{aligned}
 \sum_{i=1}^n \hat{c}_i &= \sum_{i=1}^n (c_i + \Phi(D_i) - \Phi(D_{i-1})) \\
 &= \sum_{i=1}^n c_i + \Phi(D_n) - \Phi(D_0) \\
 &\geq \sum_{i=1}^n c_i \quad \Phi(D_n) \geq 0 \text{ ve } \Phi(D_0) = 0 \text{ iken..}
 \end{aligned}$$



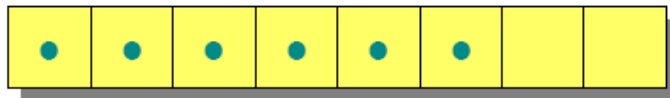
## Tabloyu ikiye katlamanın potansiyel analizi

$\Phi(D_i) = 2i - 2^{\lceil \lg i \rceil}$  ile  $i$ 'inci eklemekten sonra, tablonun potansiyelini tanımlayın. (  $2^{\lceil \lg 0 \rceil} = 0$  olduğunu varsayın.)

**Not :**

- $\Phi(D_0) = 0$
- Bütün  $i$ ' ler için  $\Phi(D_i) \geq 0$

**Örnek :**



$$\Phi = 2 \cdot 6 - 2^3 = 4$$



**Hesaplama Yöntemi**

)

## Amortize edilmiş maliyetlerin hesaplanması

$i$ 'inci eklemenin maliyeti

$$\begin{aligned}
 \hat{c}_i &= c_i + \Phi(D_i) - \Phi(D_{i-1}) \\
 &= \left\{ \begin{array}{ll} i & i-1, 2\text{'nin tam kuvveti ise,} \\ 1 & \text{değilse} \end{array} \right\} \\
 &\quad + (2i - 2^{\lceil \lg i \rceil}) - (2(i-1) - 2^{\lceil \lg(i-1) \rceil}) \\
 &= \left\{ \begin{array}{ll} i & i-1, 2\text{'nin tam kuvveti ise,} \\ 1 & \text{değilse} \end{array} \right\} \\
 &\quad + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg(i-1) \rceil}.
 \end{aligned}$$

## Hesaplama

**Durum 1 :**  $i-1, 2$ 'nin tam kuvvetidir.

$$\begin{aligned}
 \hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
 &= i + 2 - 2(i-1) + (i-1) \\
 &= i + 2 - 2i + 2 + i - 1 \\
 &= 3
 \end{aligned}$$

**Durum 2 :**  $i-1, 2$ 'nin tam kuvveti değilse.

$$\begin{aligned}
 \hat{c}_i &= 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\
 &= 3 \quad ( 2^{\lceil \lg i \rceil} = 2^{\lceil \lg (i-1) \rceil} \text{ iken } )
 \end{aligned}$$

## Hesaplama

**Durum 1 :**  $i-1, 2$ 'nin tam kuvvetidir.

$$\begin{aligned}\hat{c}_i &= i + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= i + 2 - 2(i-1) + (i-1) \\ &= i + 2 - 2i + 2 + i - 1 \\ &= 3\end{aligned}$$

**Durum 2 :**  $i-1, 2$ 'nin tam kuvveti değilse.


$$\begin{aligned}\hat{c}_i &= 1 + 2 - 2^{\lceil \lg i \rceil} + 2^{\lceil \lg (i-1) \rceil} \\ &= 3 \quad (2^{\lceil \lg i \rceil} = 2^{\lceil \lg (i-1) \rceil} \text{ iken})\end{aligned}$$

$n$  tane ekleme en kötü durumda  $\Theta(n)$ 'ya mal olur.

**Egzersiz :** Birinci eklemenin amortize edilmiş maliyetinin sadece 2 olduğunu, bu analizdeki hatayı bularak gösterin...

## Sonuçlar

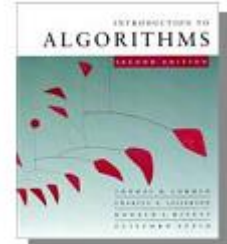
- Amortize edilmiş maliyetler, veri yapısı performansı ile ilgili açık bir soyutlama sağlar.
- Amortize edilmiş analiz kullanılırken, herhangi bir metod kullanılabilir. Ancak, her metodun daha basit ve özet kullanım durumları mevcuttur.
- Hesaplama metodunda veya Potansiyel metodunda, ciddi anlamda farklı sınırlar yaratan, amortize edilmiş maliyetlerin atanması için farklı şemalar bulunabilir.



## İkili Arama Ağaçları (BST)

Rastgele yapılanmış ikili  
arama ağaçları

- Beklenen düğüm derinliği
- Yüksekliği çözümlmek



# İkili-arama-ağacı sıralaması

$T \leftarrow \emptyset$       $i = 1$  den  $n'$  ye kadar değiştiğinde,

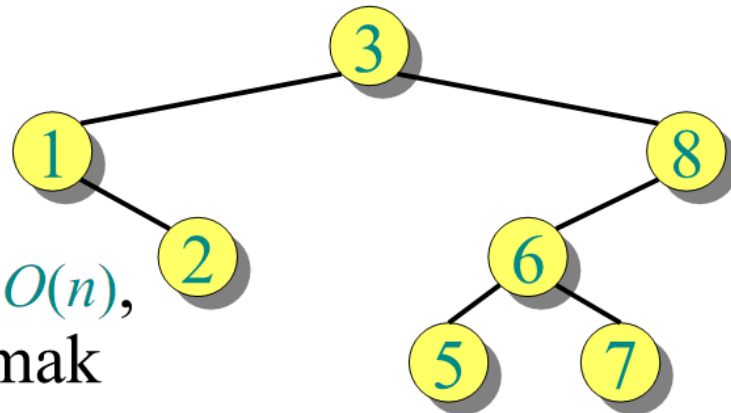
▷ Boş bir BST (ikili arama ağacı) yarat.

AĞAÇ ARAYA YERLEŞTİRMESİ **YAP** ( $T, A[i]$ )

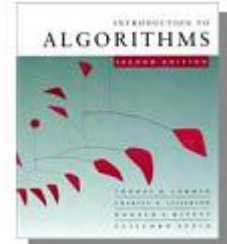
$T$ 'nin içinde sıralı adımlama yap.

**Örnek:**

$A = [3 \ 1 \ 8 \ 2 \ 6 \ 7 \ 5]$

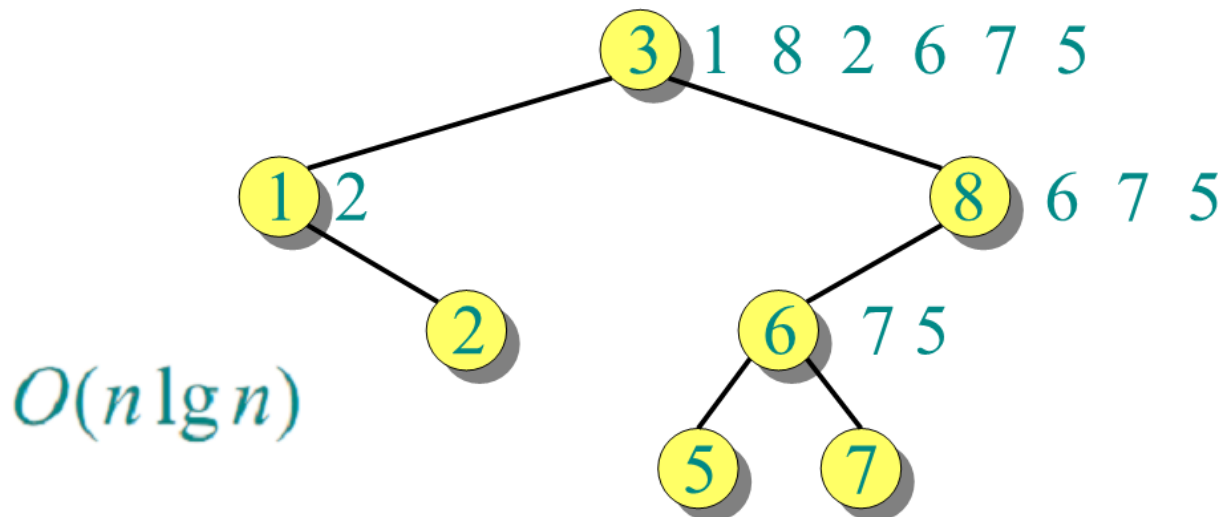


Ağaç adımlama süresi =  $O(n)$ ,  
ancak BST'yi oluşturmak  
ne kadar zaman alır?



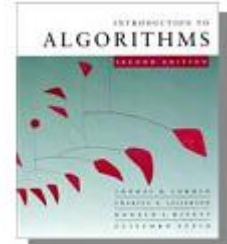
# BST sıralaması çözümlemesi

BST sıralaması çabuk sıralama karşılaştırmalarının aynısını, başka bir düzende yapar!



Ağacı oluşturmanın beklenen süresi asimptotik olarak çabuk sıralamanın koşma süresinin aynıdır.





## Düğüm derinliği

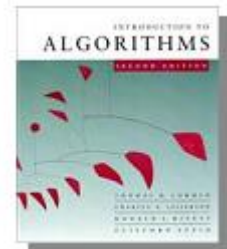
Bir düğüm derinliği = AĞAÇ ARAYA YERLEŞTİRMESİ için yapılan karşılaştırmalar. Tüm girdi permütasyonları eşit olasılıklı varsayılırsa:

Ortalama düğüm derinliği

$$= \frac{1}{n} E \left[ \sum_{i=1}^n \text{(Boğum } i' \text{ yi araya yerleştirmek için gerekli karşılaştırmaların sayısı)} \right]$$

$$= \frac{1}{n} O(n \lg n) \quad (\text{Çabuk sıralama analizi})$$

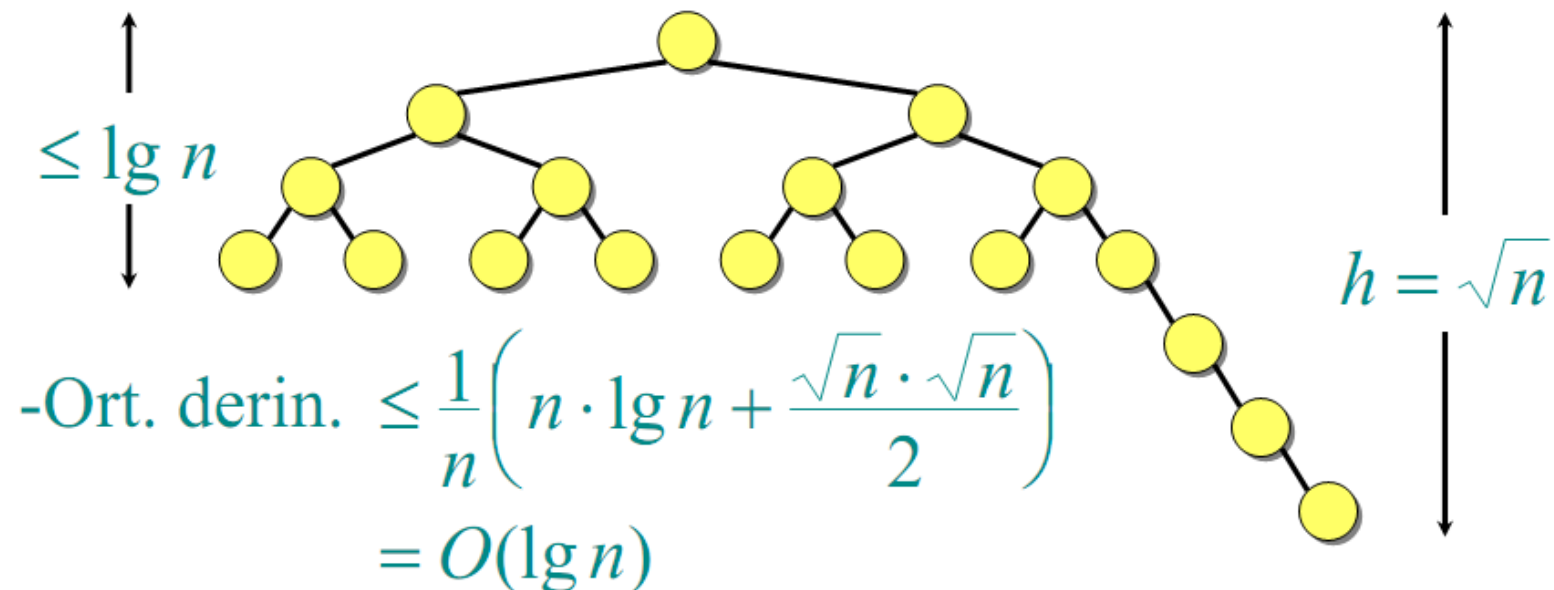
$$= O(\lg n) .$$



## 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.**





## **7.Hafta**

# **Dengeli Arama Ağaçları (Red - Black Tree)**

- Kırmızı-siyah ağaçlar
  - Kırmızı-siyah ağacın yüksekliği
  - Rotation / Dönme
  - Insertion / araya yerleştirme
-