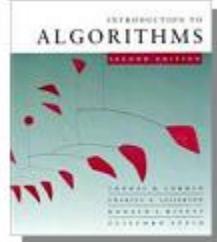


6.Hafta

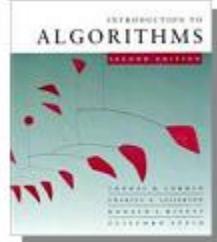
**Sıralama Algoritmaları
Çabuk Sıralama, Rastgele
Algoritmalar**

Sıralama Algoritmaları



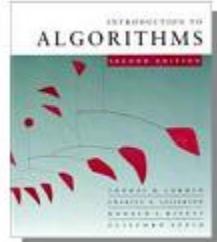
- Sıralama algoritmaların tipleri:
 - Karşılaştırmaya dayalı sıralama algoritmaları
 - Heap Sort, quicksort, insertion sort, bubble sort, merge sort,...
 - En iyi çalışma zamanı $\Theta(n \log n)$
 - Doğrusal zaman sıralama algoritmaları
 - Counting sort (sayma), radix(taban) sort, bucket (sepet) sort.

Yerinde Sıralama :In-place Sorting



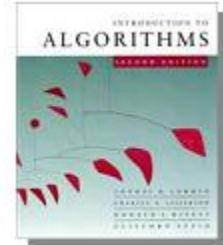
- Yerinde Sıralama: Algoritmanın, boyutu $\Theta(n)$ olan ekstra depolama (tek değişkenli ve register (kayıtlar) dışında) alan gerektirmemesi.
 - Algoritma: Yerinde Sıralama
 - - Bubble sort Evet
 - - Insertion sort Evet
 - - Selection sort Evet
 - - Merge sort Hayır(ek alan gereklidir)
 - - Heap sort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

Heap (Yığın ağacı)

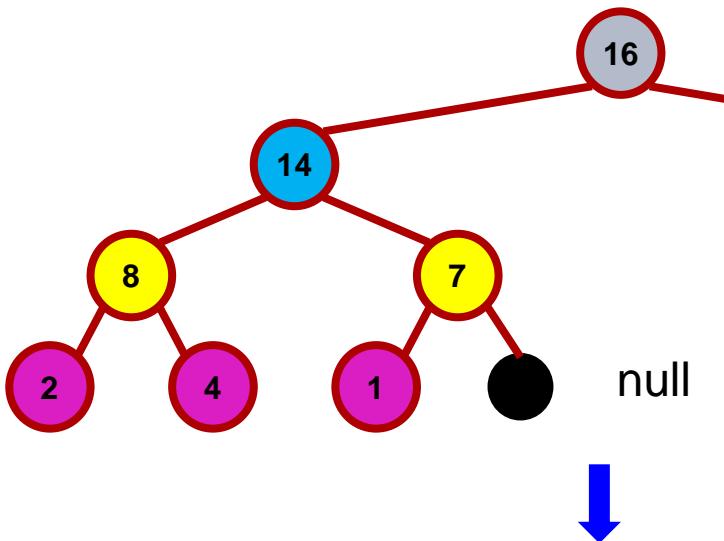


- Heap, ikili ağaç (binary tree) olarak düşünebileceğimiz bir veri yapısıdır.
 - Dizi
 - Complete binary tree yakın bir ağaç olarak görülebilir.
 - En düşük seviye hariç bütün seviyeler doludur.
 - Her düğümdeki veri kendi çocuk düğümlerinden büyük (max-heap) veya küçüktür (min-heap).

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

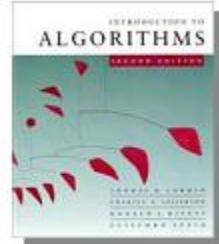
Heap (Yığın ağacı)

- 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 $A[2i + 1]$

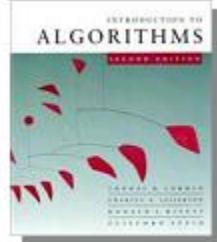
- $\text{Left}(i)=2i$
- $\text{Right}(i)=2i+1$
- $\text{Parent}(i)=\lfloor i / 2 \rfloor$

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



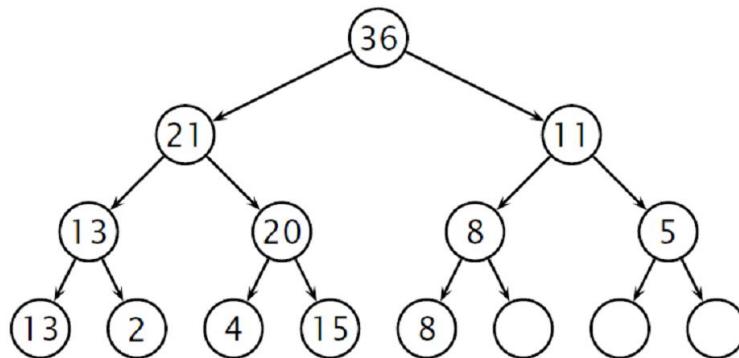
```

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

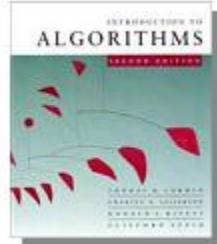


Max-Heap Özelliği

- 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?

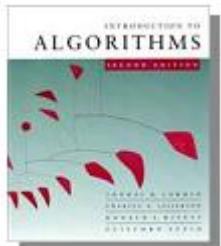


Heap Yüksekliği



○ 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;
```

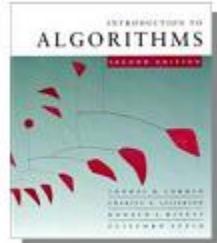
```
    if (R <= heap_size(A) && A[R] > A[largest])
```

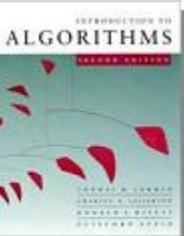
```
        largest = R;
```

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

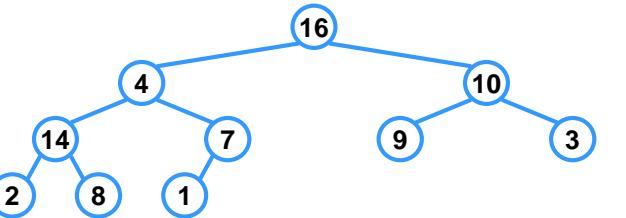
```
        Swap(A, i, largest); Max_Heapify (A,largest);
```

```
}
```

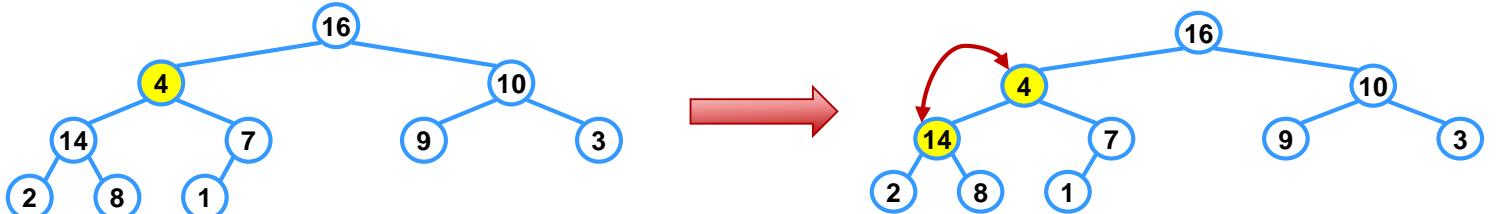




Heapify(): Örnek

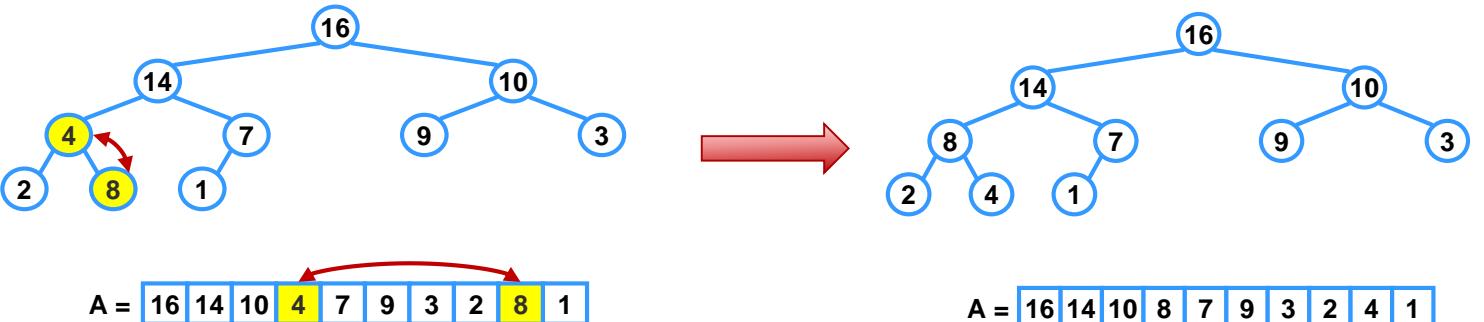


$A = [16 \ 4 \ 10 \ 14 \ 7 \ 9 \ 3 \ 2 \ 8 \ 1]$



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

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

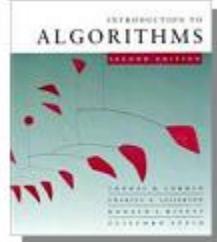


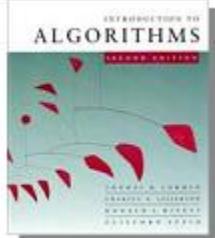
$A = [16 \ 14 \ 10 \ 4 \ 7 \ 9 \ 3 \ 2 \ 8 \ 1]$

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

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





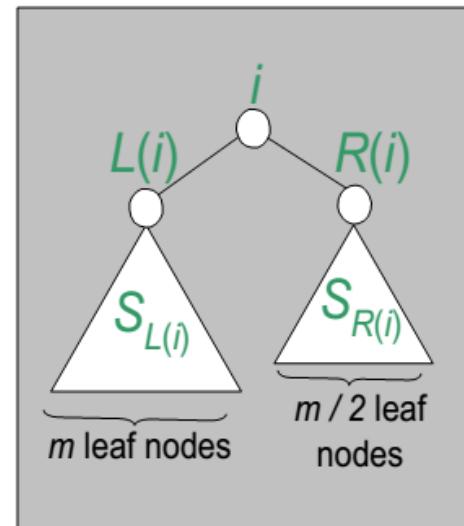
Heapify() Analizi: ispat

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

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

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

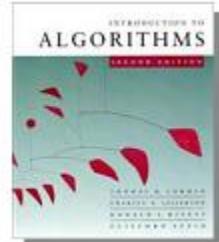
- $T(n) \leq T(2n/3) + \Theta(1) \Rightarrow T(n) = O(\lg n)$



By case 2 of
Master Thm

Heap İşlemleri :Heap Yapılandırması

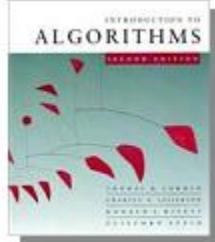
BuildHeap()



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

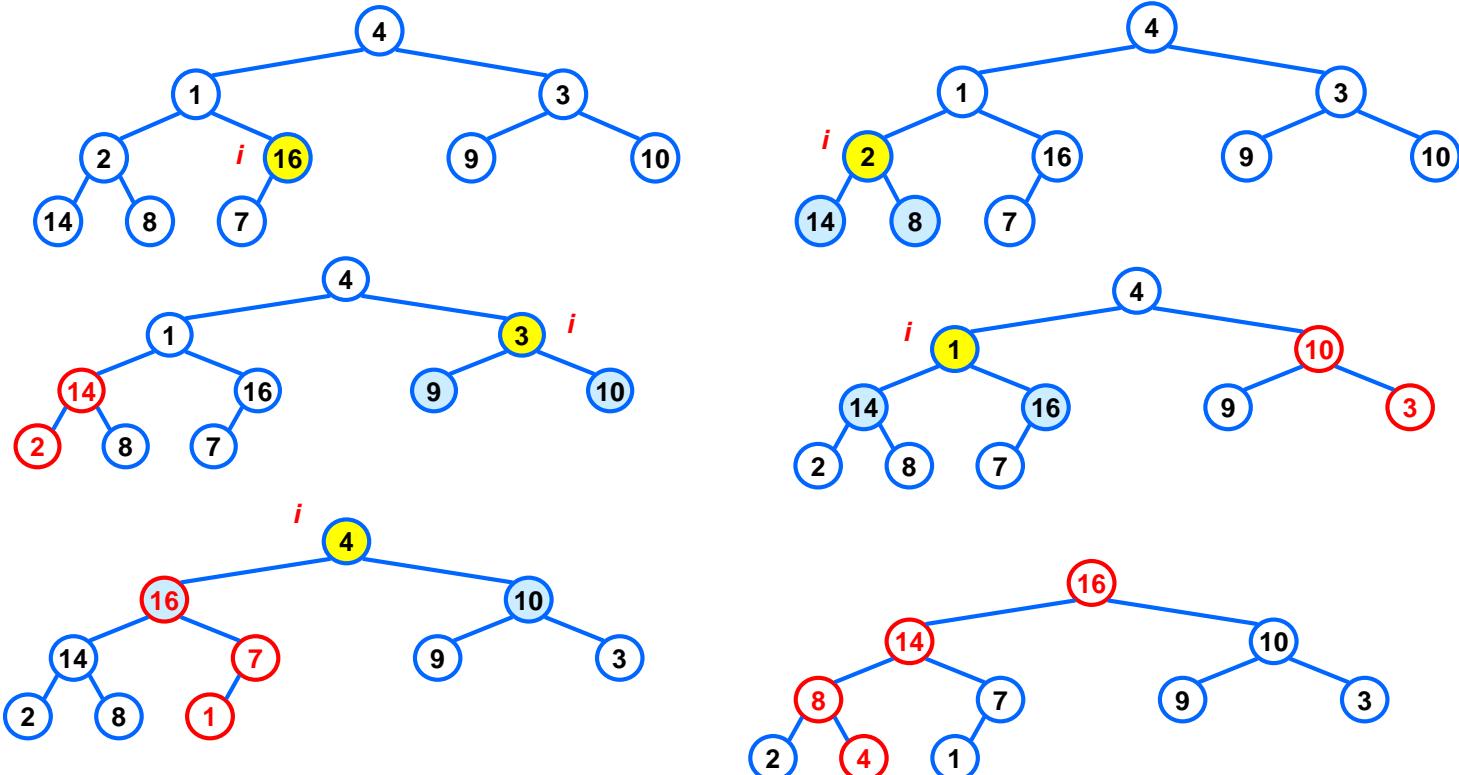
BuildHeap(A)

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

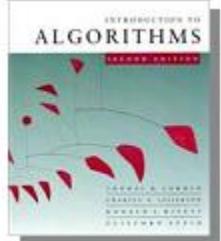


Örnek:BuildHeap()

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

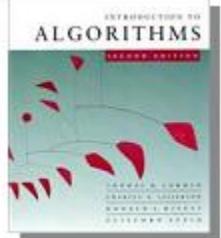


BuildHeap() Analizi

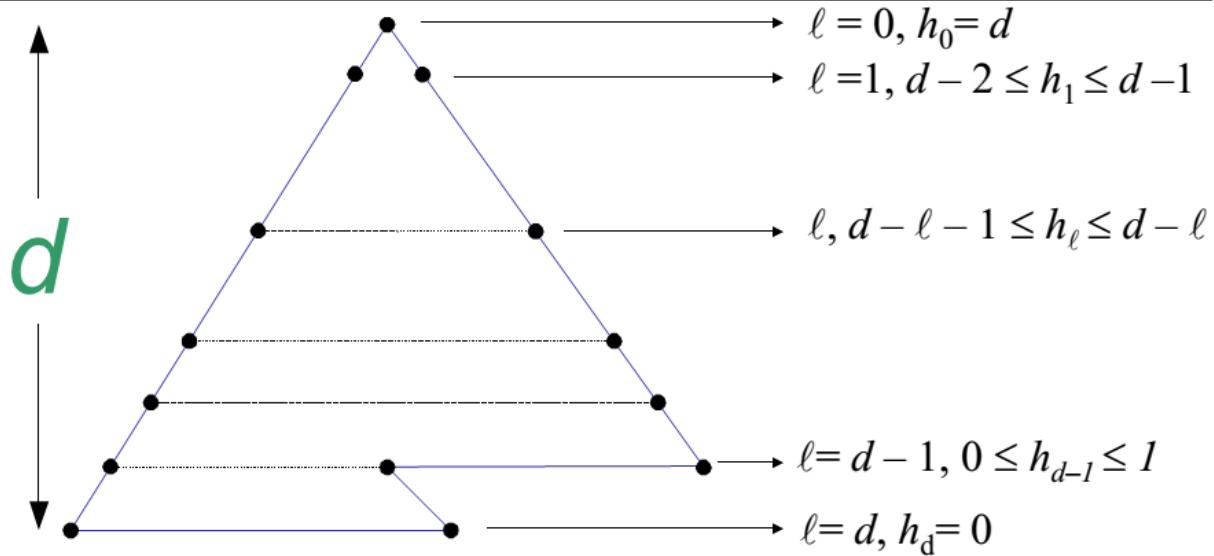


- **Heapify()** her çağrıldığında $O(\lg n)$ zaman alır.
- **Max_Heapify (A, i)**, $\rightarrow O(h)$ zaman gerektirir, burada $h \rightarrow i$.
düğümün yüksekliği yani $\log n$ dir.
- Her seviyede $O(n/2^{h+1})$, **Max_Heapify** çağrıları yapılınrsa $O(n/2^h)$,
(özellikle $\lfloor n/2 \rfloor$ çağrılarında)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

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

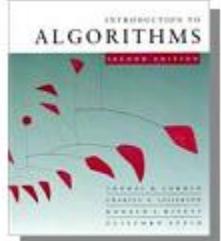
$$\therefore T(n) = O\left(\sum_{\ell=0}^{d-1} 2^\ell (d - \ell)\right)$$

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



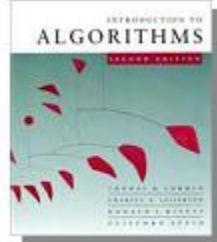
$$- T(n) = \sum_{h=0}^{\lfloor \lg n \rfloor} O(n/2^h)O(h) = O(n \times \left[\frac{1}{2} + \frac{2}{4} + \dots + \frac{\lg n}{n} \right])$$

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

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

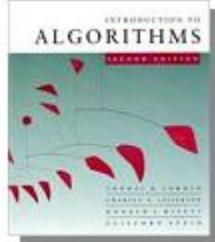
$$T(n) = O(n)$$

Heap Sort Algoritması



Heapsort (A)

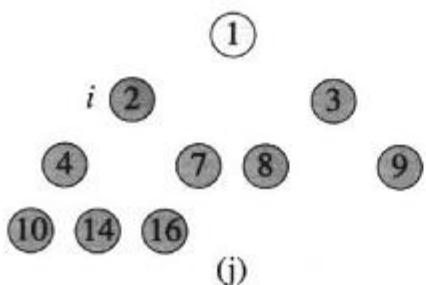
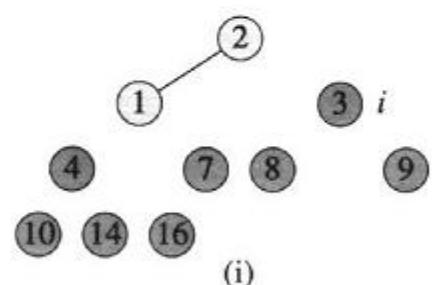
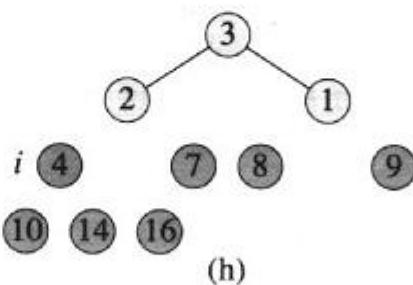
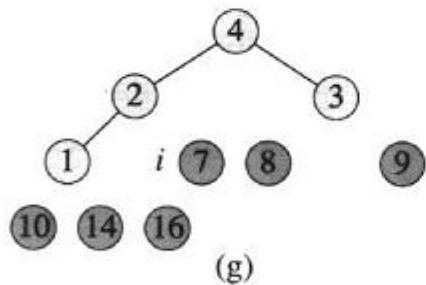
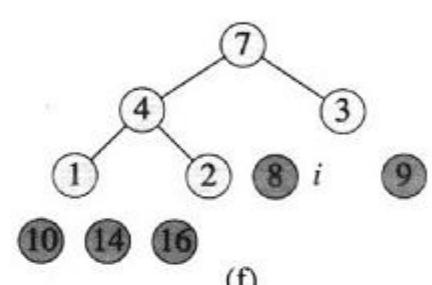
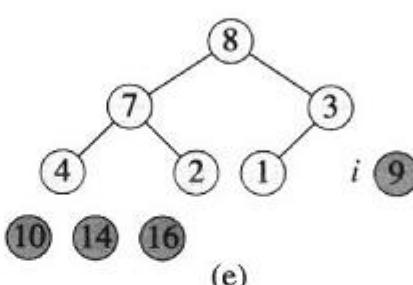
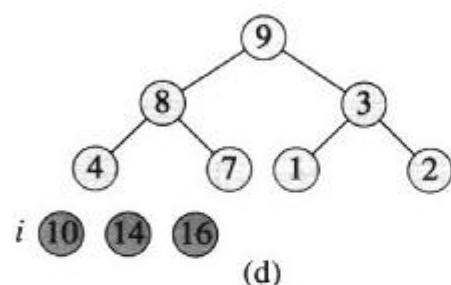
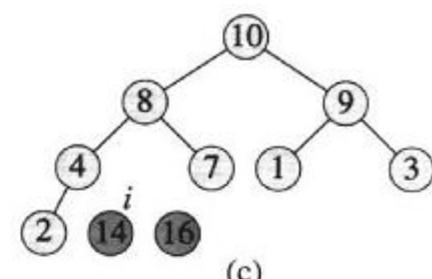
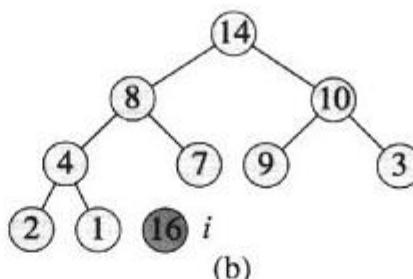
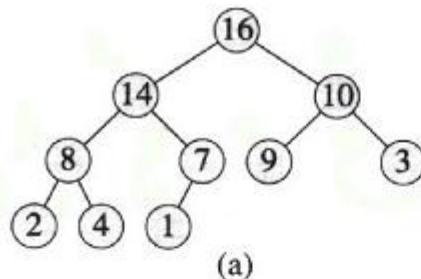
```
{      BuildHeap (A) ;  
        for  (i = length (A)  downto  2)  
        {  
            Swap (A[1] ,  A[i]) ;  
            heap_size (A)  -=  1;  
            Heapify (A,  1) ;  
        }  
    }
```



Heap Sort Algoritması Analizi

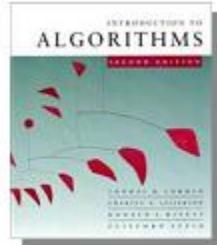
- **BuildHeap()** çağrılmaması: $O(n)$ time
 - **Heapify()** $n - 1$ çağrılmaması : $O(\lg n)$ time
 - **HeapSort()** toplam çalışma zamanı
-
- $T(n) = O(n) + (n - 1) O(\lg n)$
 - $T(n) = O(n) + O(n \lg n)$
 $T(n) = O(n \lg n)$

Heap Sort



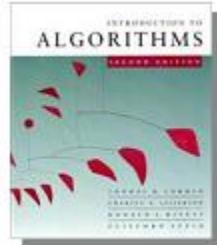
A	1	2	3	4	7	8	9	10	14	16
-----	---	---	---	---	---	---	---	----	----	----

(k)



Heap Sort

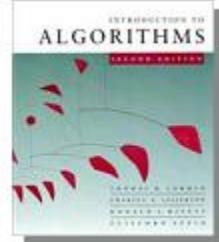
- Heapsort iyi bir algoritmadır fakat pratikte genelde Quicksort daha hızlıdır.
- Ancak heap veri yapısı, öncelik sırası uygulaması (*priority queues*) için inanılmaz faydalıdır.
 - Her biri ilişkili bir anahtar (key) veya değer olan elamanların oluşturduğu **A** dizisini muhafaza etmek için bir veri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



Heap Sort

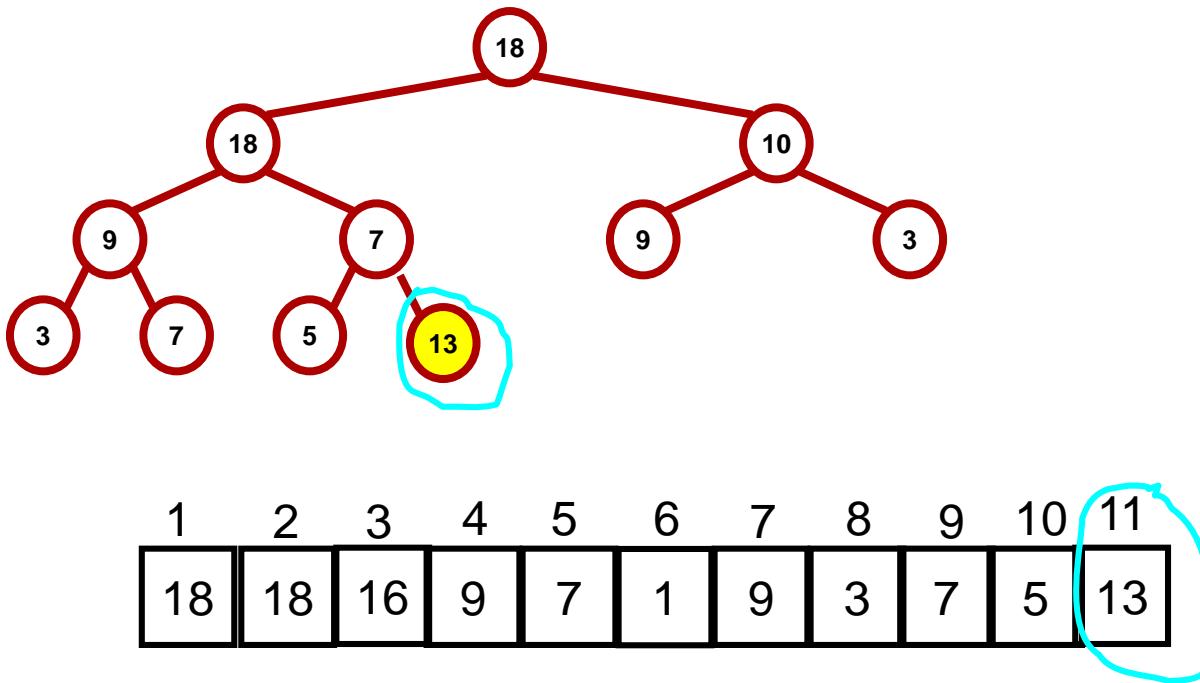
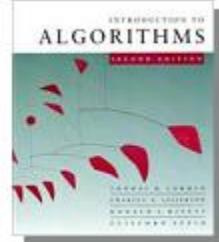
- **Insert(S, x)** : S dizisine x elemanını ekler
- **Maximum(S)**: S dizisindeki maksimum elamanı geri döndürür
- **ExtractMax(S)** S dizisindeki maksimum elamanı geri döndürür ve elamanı diziden çıkarır

Öncelikli Kuyruk Uygulamaları (Implementing Priority Queues)

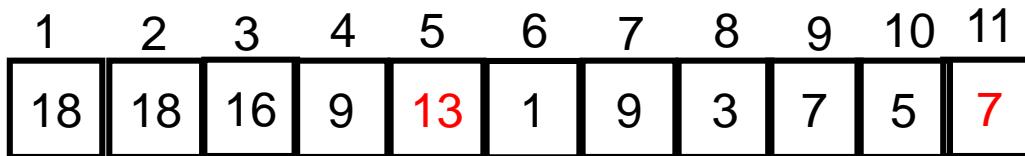
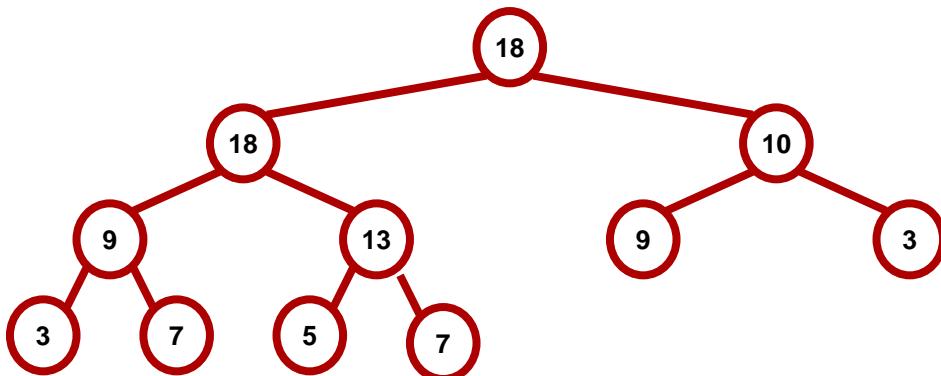
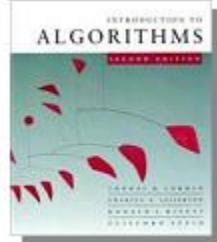


```
HeapInsert(A, key)
{
    heap_size[A]++;
    i = heap_size[A];
    while (i > 1 AND A[Parent(i)] < key)
    {
        A[i] = A[Parent(i)];
        i = Parent(i);
    }
    A[i] = key;
}
```

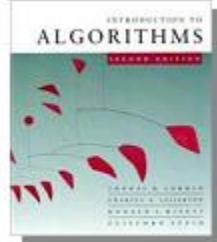
Öncelikli Kuyruk Uygulamaları (Implementing Priority Queues)



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)

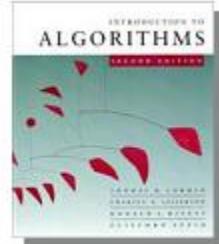
- C.A.R. Hoare tarafından 1962'de önerildi.
- Böl ve fethet algoritması.
- "Yerinde" sıralar (araya yerleştirme sıralamasında olduğu gibi; birleştirme sıralamasından farklı).
- (Ayar yapılrsa) çok pratik.



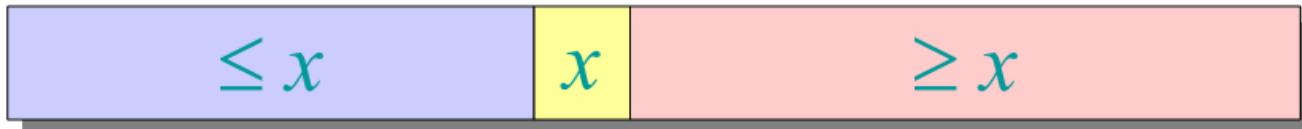
Sir Charles Antony Richard
Hoare
1934 -

Çabuk sıralama (Quick Sort)

Böl ve fethet



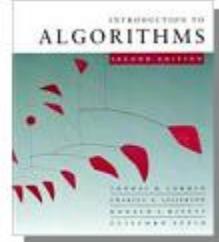
- **n -elemanlı bir dizilimin çabuk sıralanması:**
- **1. Böl:** Dizilimi **pivot (eksen sabit) x** 'in etrafında iki altdizilime bölüntüle; burada soldaki altdizilim elemanları $\leq x \leq$ sağdaki altdizilim elemanları olsun.



- **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

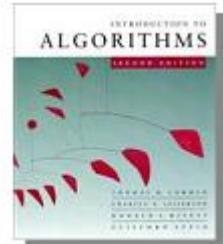
QUICKSORT(A, p, r)

if $p < r$

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

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

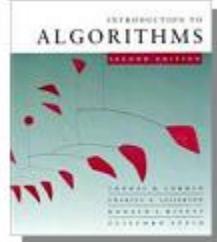
QUICKSORT($A, q+1, r$)



İlk arama: QUICKSORT($A, 1, n$)

Çabuk sıralama (Quick Sort)

Bölbüntüleme örneği



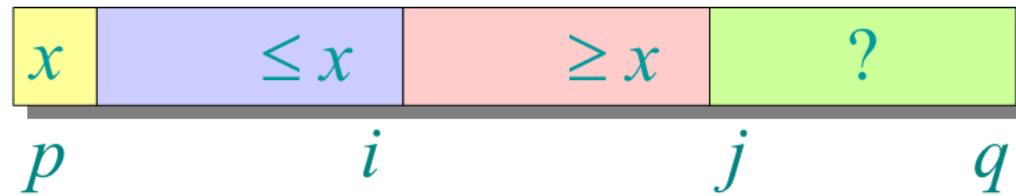
```

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)

```

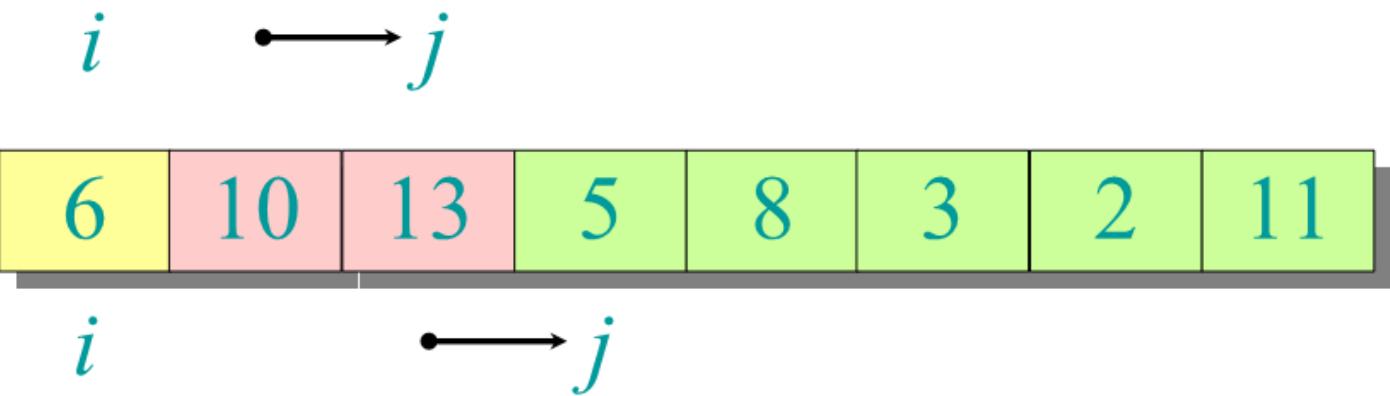
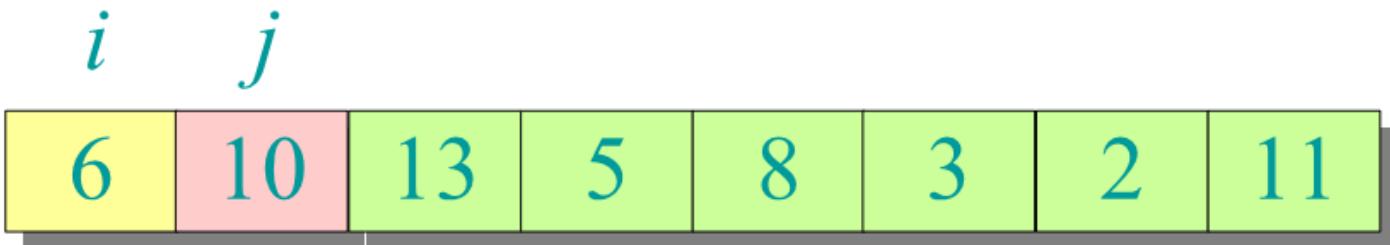
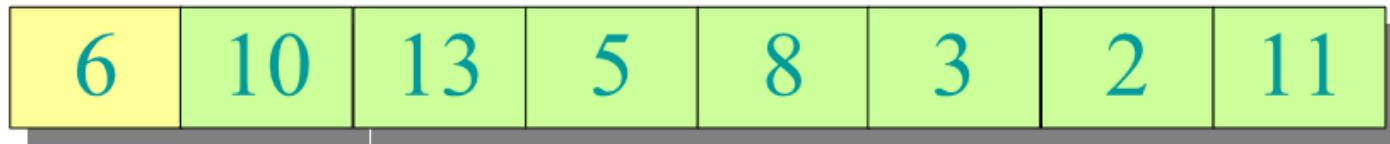
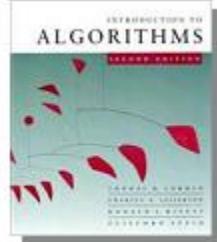
Koşma süresi
 $=\mathcal{O}(n)$
 n eleman için

Değişmez:



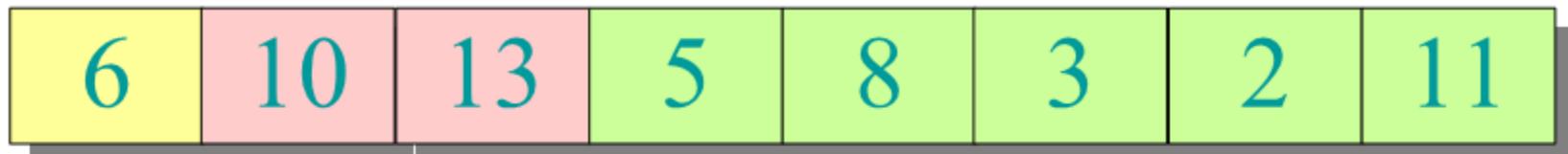
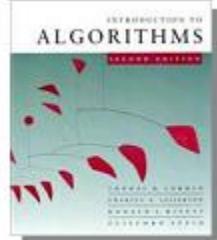
Çabuk sıralama (Quick Sort)

Bölüntüleme örneği



Çabuk sıralama (Quick Sort)

Bölüntüleme örneği

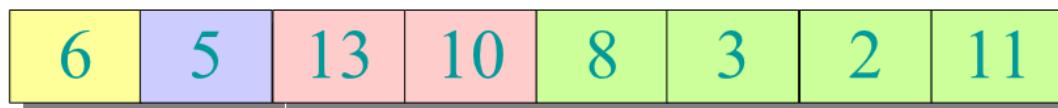
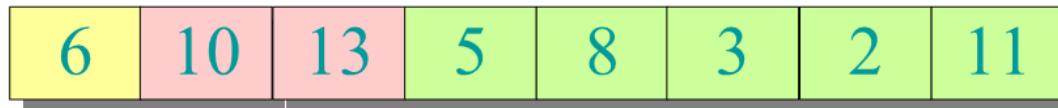
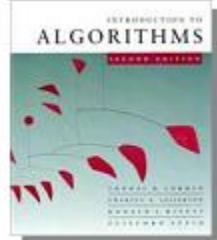


→ i

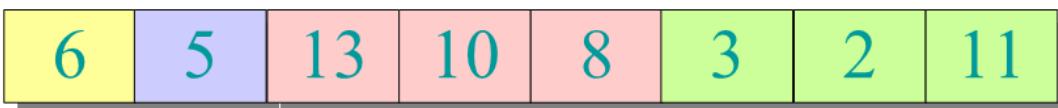
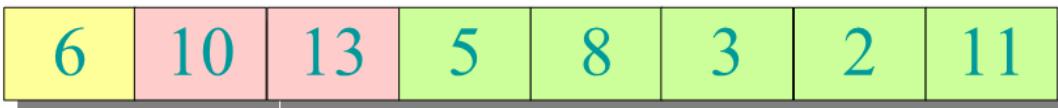
j

Çabuk sıralama (Quick Sort)

Bölüntüleme örneği



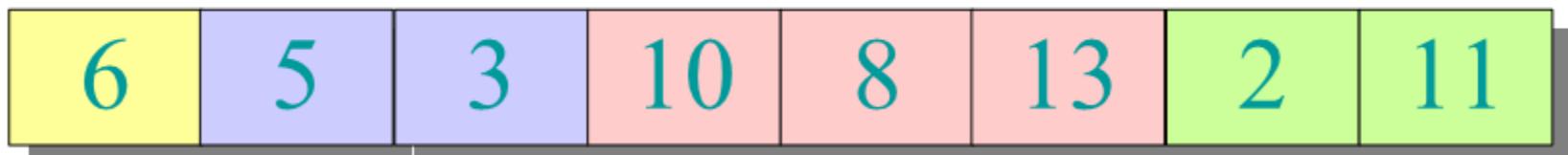
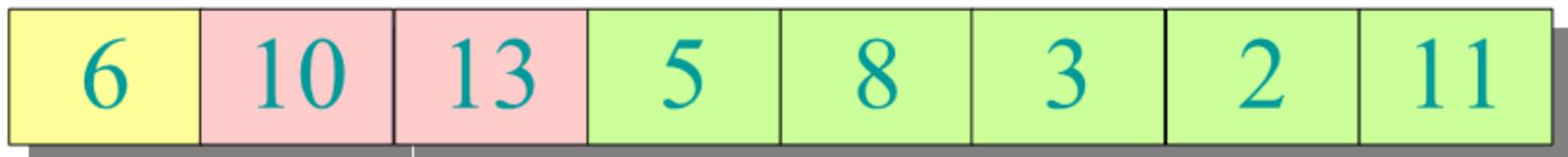
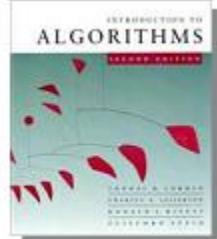
i $\longrightarrow j$



i $\longrightarrow j$

Çabuk sıralama (Quick Sort)

Bölüntüleme örneği

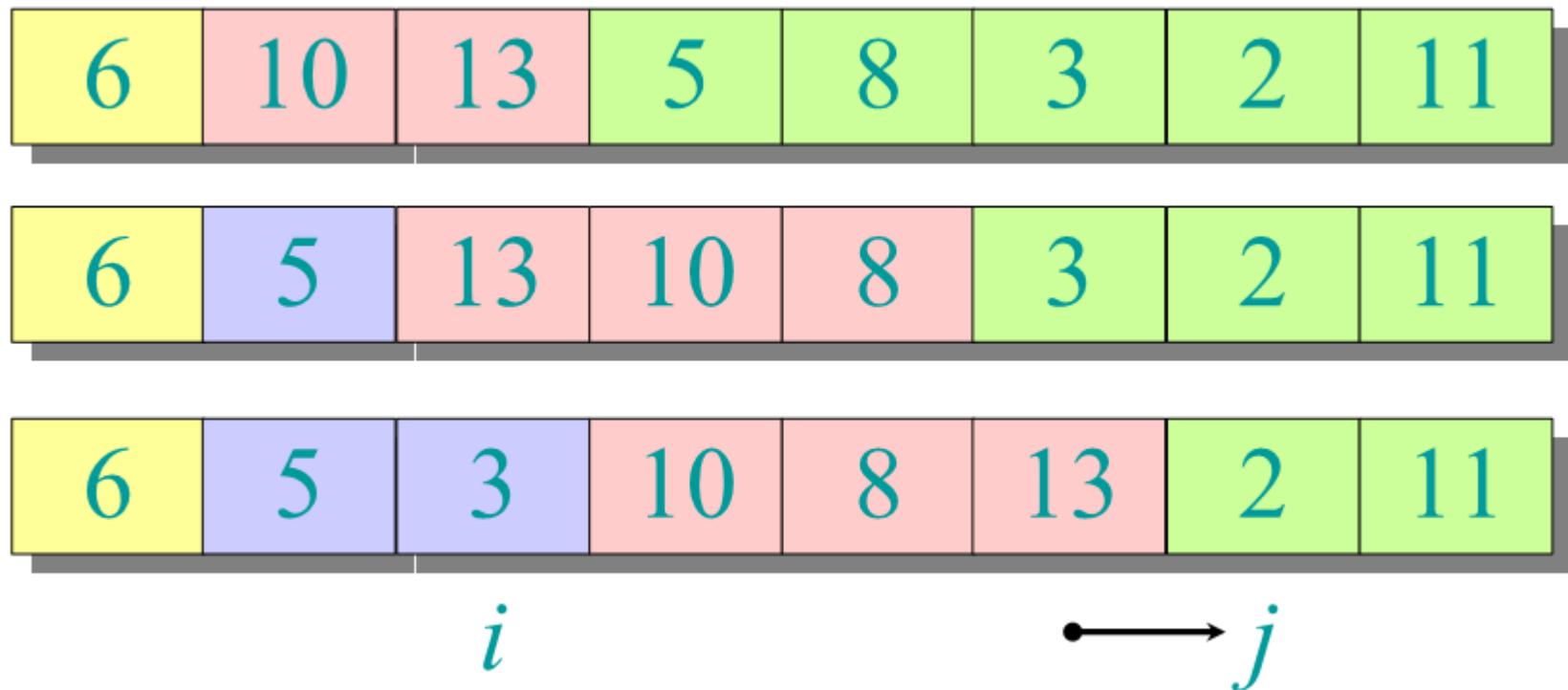
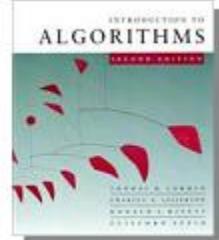


i

j

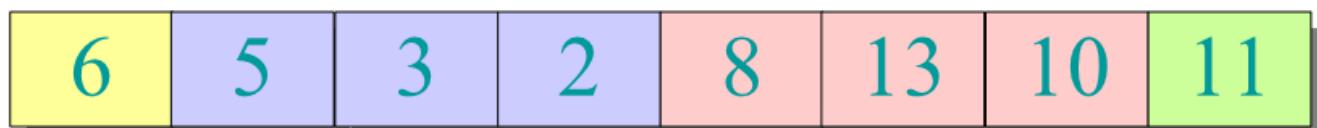
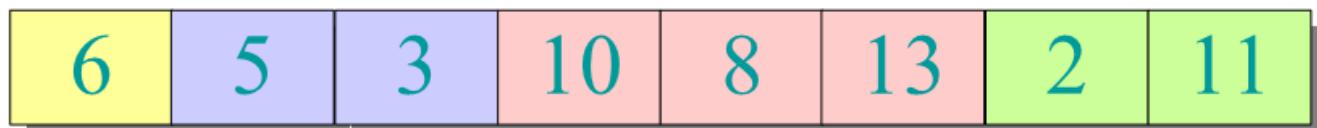
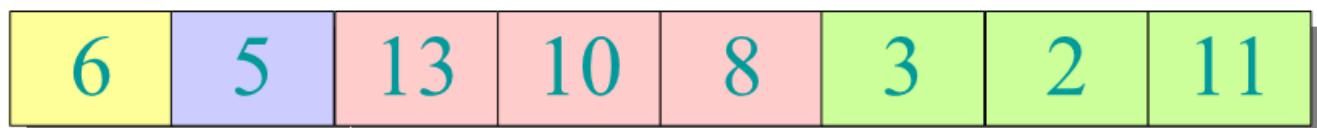
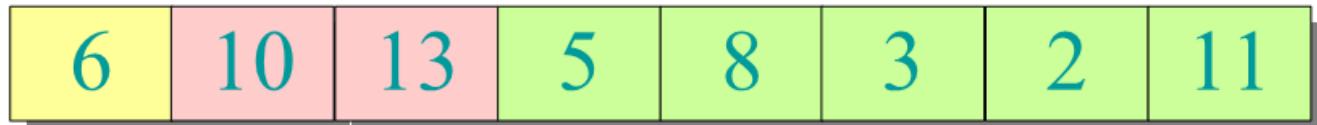
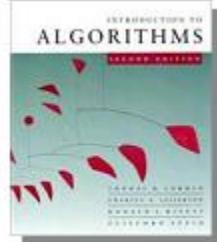
Çabuk sıralama (Quick Sort)

Bölüntüleme örneği



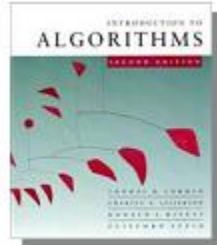
Çabuk sıralama (Quick Sort)

Bölüntüleme örneği



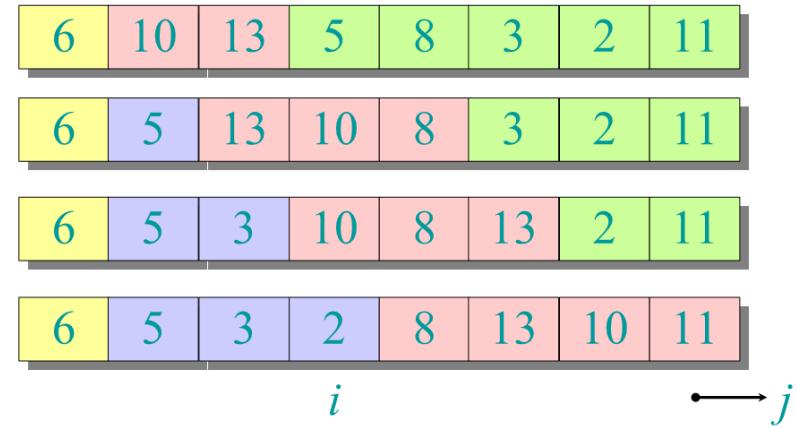
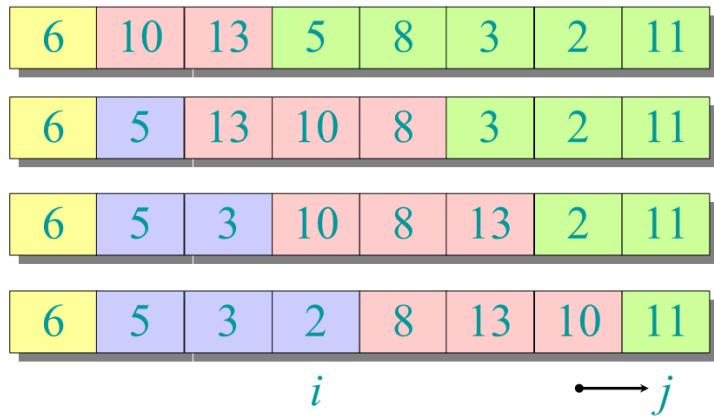
→ i

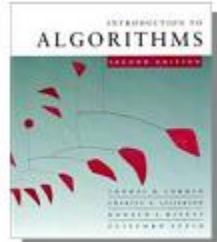
j



Çabuk sıralama (Quick Sort)

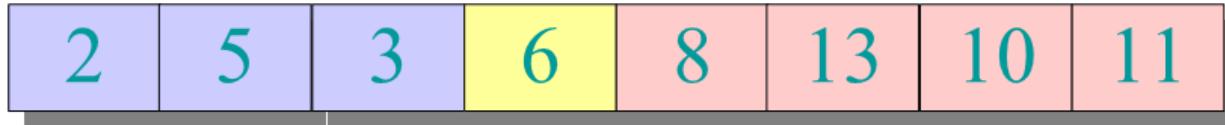
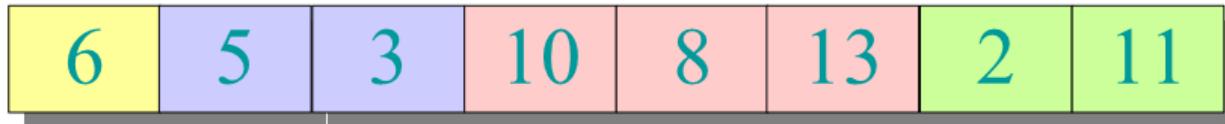
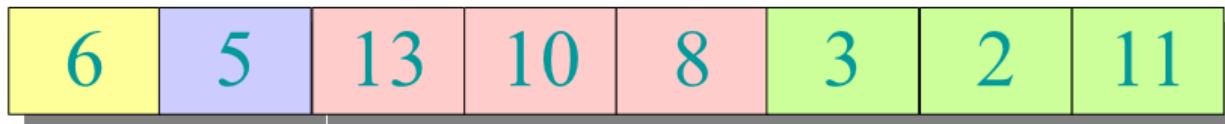
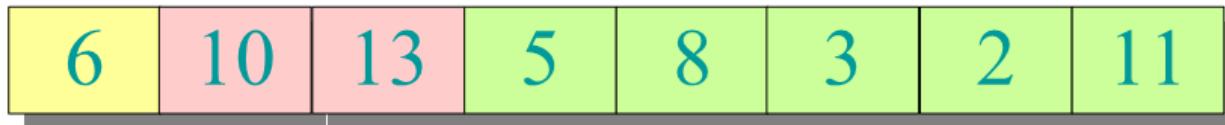
Bölbüntüleme örneği





Çabuk sıralama (Quick Sort)

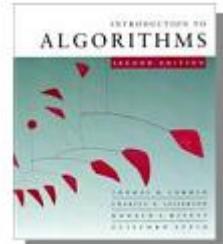
Bölbüntüleme örneği



i

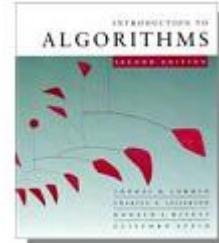
Çabuk sıralamanın çözümlemesi

- 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.



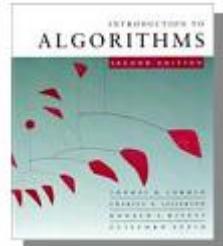
Çabuk sıralamanın en kötü durumu (worst-case)

- 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

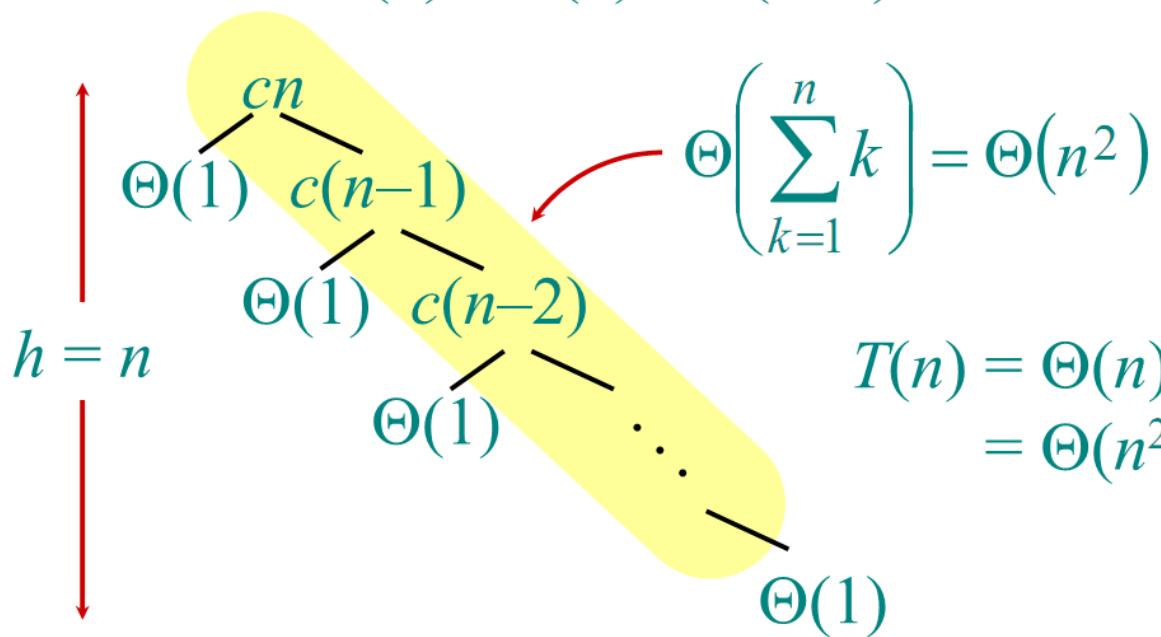


$$\begin{aligned} T(n) &= T(0) + T(n-1) + \Theta(n) \\ &= \Theta(1) + T(n-1) + \Theta(n) \\ &= T(n-1) + \Theta(n) \\ &= \Theta(n^2) \quad (\text{aritmetik seri}) \end{aligned}$$

Çabuk sıralamanın En kötü durum özyineleme ağacı



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



Rastgele çabuk sıralama çözümlemesi (analizi)

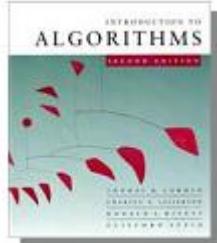
- En kötü durum:

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

- 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:

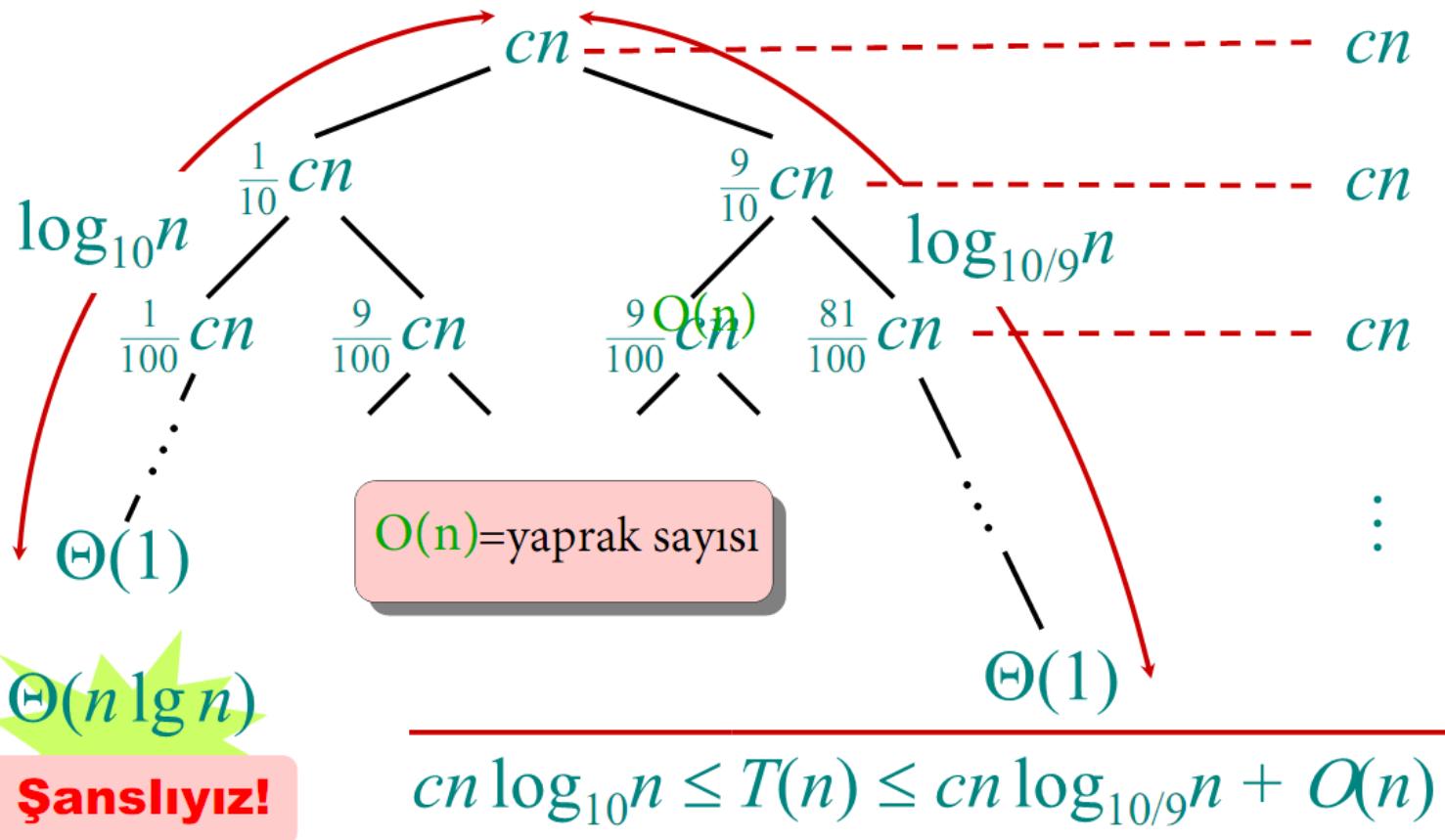
$$\begin{aligned} T(n) &= 2T(n/2) + \Theta(n) \\ &= \Theta(n \lg n) \quad (\text{birleştirme sıralamasındaki gibi}) \end{aligned}$$

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

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

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:

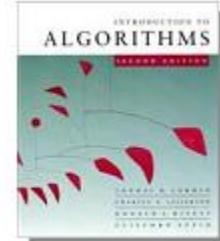
HOARE-PARTITION(A, p, r)

```
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

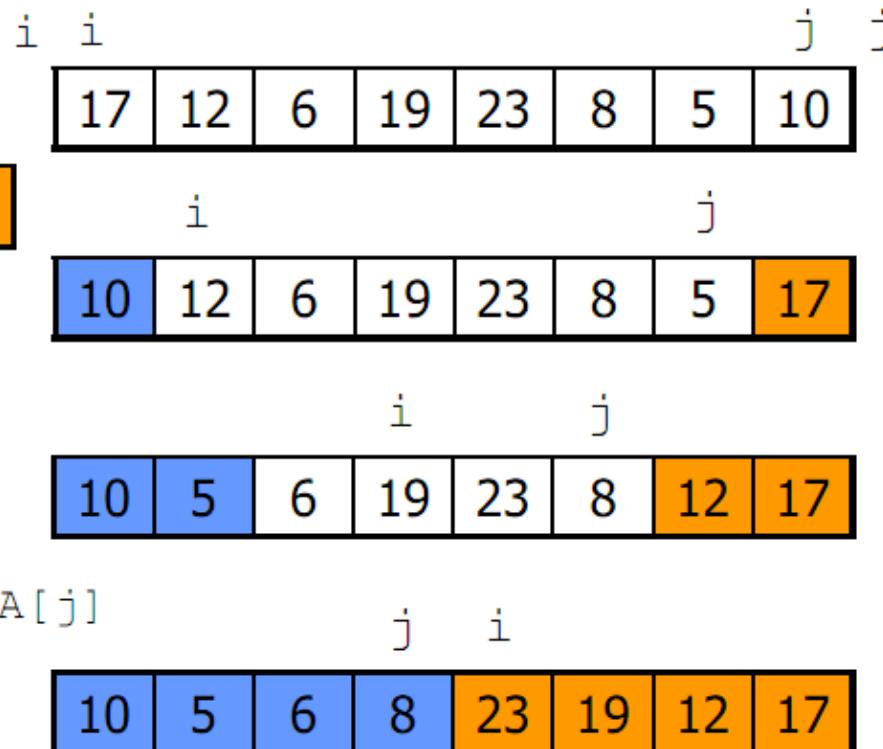


Partition(A, p, r)

```

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://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

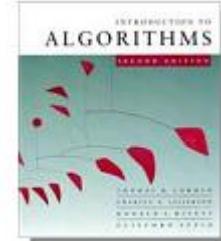
Alt Sınırları Sıralama

- Karar ağaçları

Doğrusal-Zaman Sıralaması

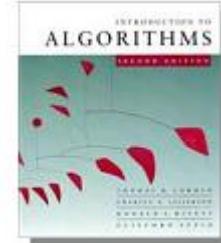
- Sayma sıralaması
- Taban sıralaması
- Kova sıralaması

Ne kadar hızlı sıralayabiliriz?

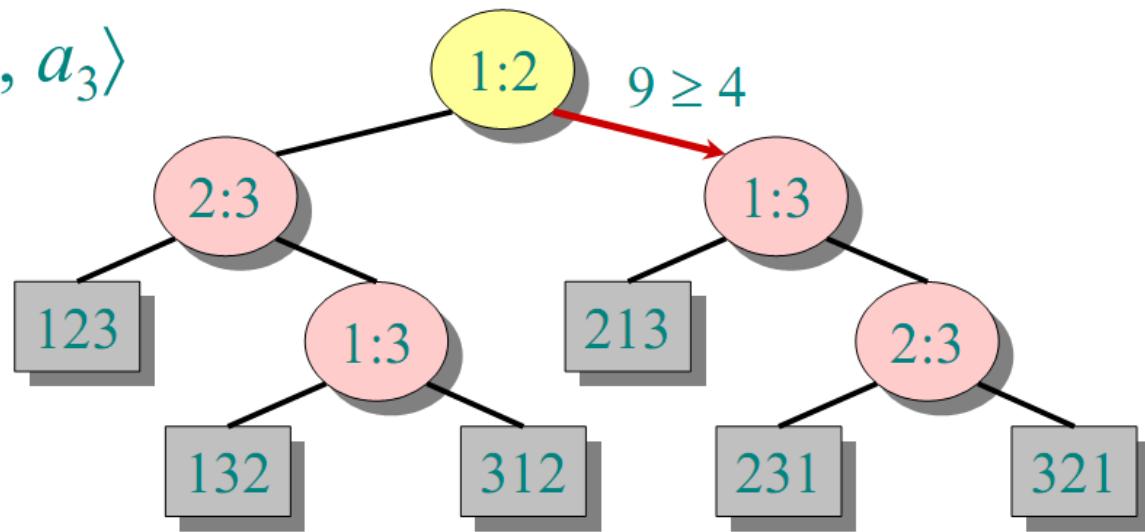


- Şu ana kadar gördüğümüz tüm sıralama algoritmaları **karşılaştırma sıralamalarıydı**. Elemanların bağıl düzenlerini saptamakta yalnız karşılaştırma kullanırlar.
- Örneğin, araya yerleştirme, birleştirme sıralamaları, çabuk sıralama, yiğin sıralaması.
- Karşılaştırma sıralamalarında gördüğümüz en iyi 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



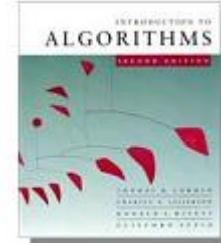
Sırala $\langle a_1, a_2, a_3 \rangle$
 $= \langle 9, 4, 6 \rangle$:



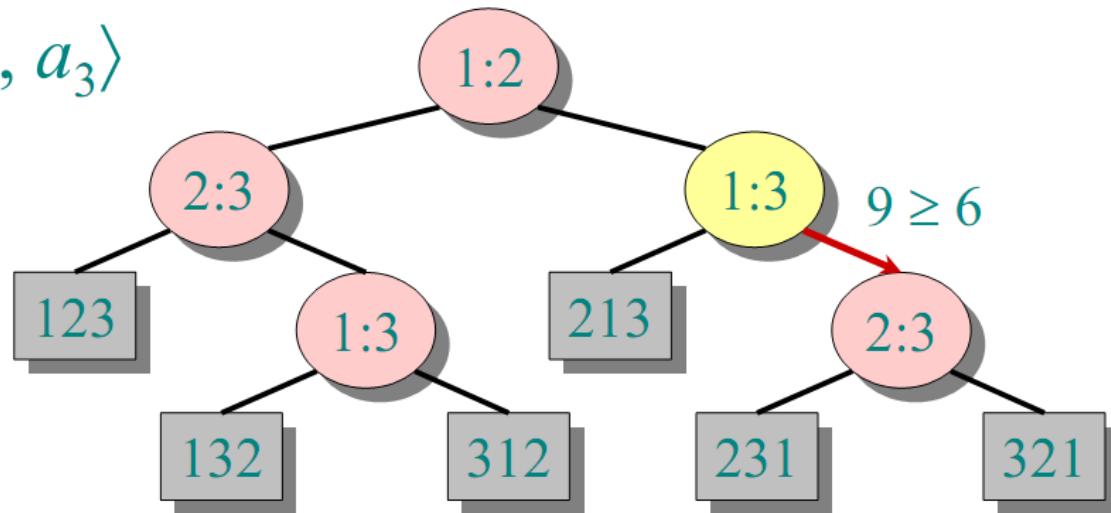
Her iç boğumun etiketlenmesi $i:j$; $i, j \in \{1, 2, \dots, n\}$ için.

- 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



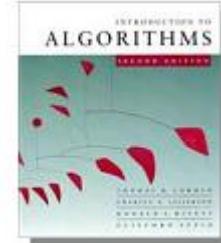
Sırala $\langle a_1, a_2, a_3 \rangle$
 $= \langle 9, 4, 6 \rangle$:



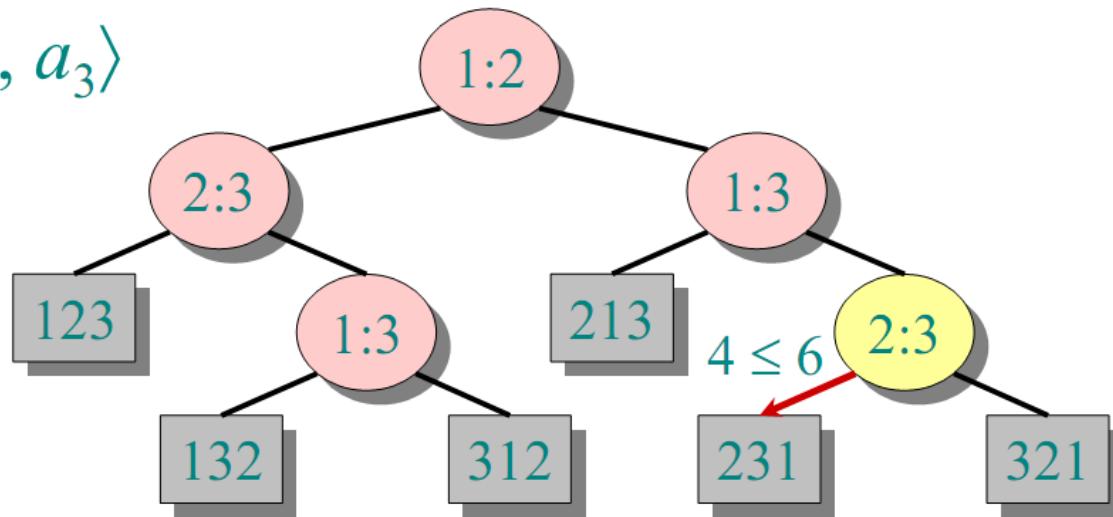
Her iç boğumun etiketlenmesi $i:j$; $i, j \in \{1, 2, \dots, n\}$ için:

- 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



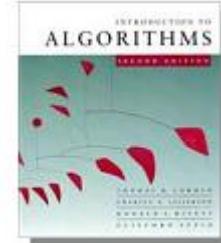
Sırala $\langle a_1, a_2, a_3 \rangle$
 $= \langle 9, 4, 6 \rangle$:



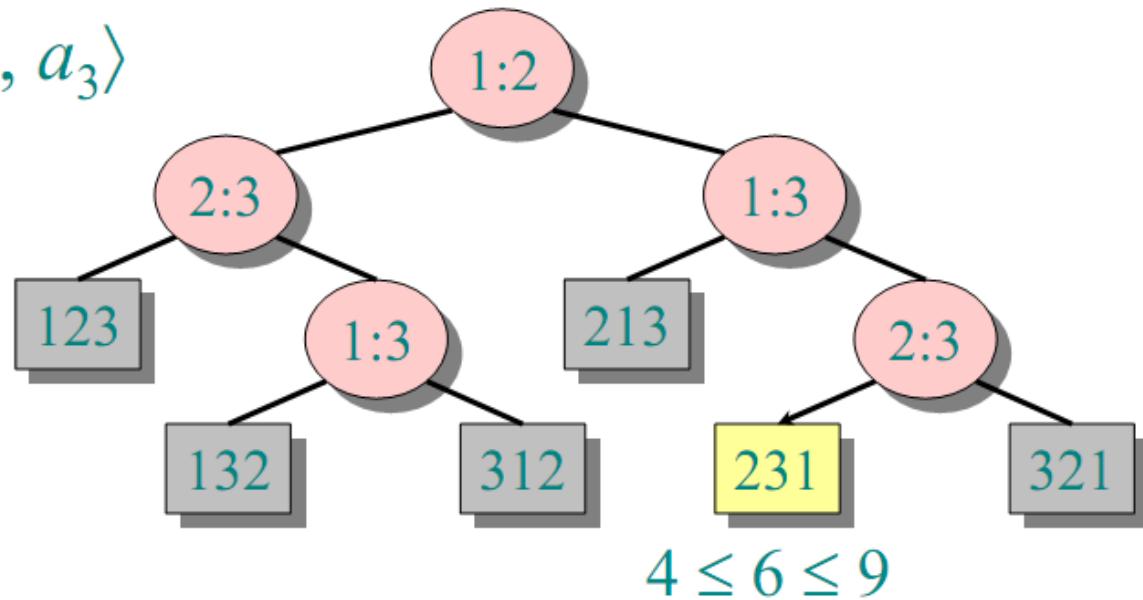
Her iç boğumun etiketlenmesi $i:j$; $i, j \in \{1, 2, \dots, n\}$ için.

- 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

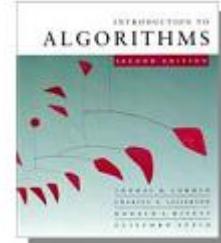


Sırala $\langle a_1, a_2, a_3 \rangle$
 $= \langle 9, 4, 6 \rangle$:



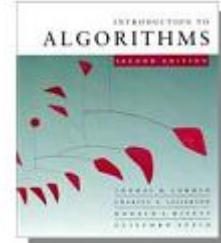
Her yaprakta $\langle \pi(1), \pi(2), \dots, \pi(n) \rangle$ permütasyonu vardır bu $a_{\pi(1)} \leq a_{\pi(2)} \leq \dots \leq a_{\pi(n)}$ sıralamasının tamamlanmış olduğunu gösterir.

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)

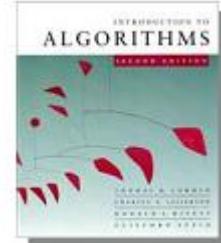
$$= n \lg n - n \lg e$$

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

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



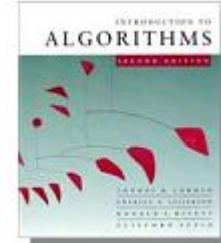
Karar-ağacı sıralamasında alt sınır



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

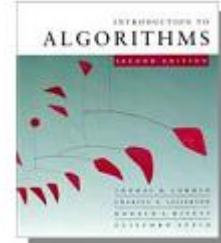
- Randomize Quick Sort da asimtotik olarak en iyi karşılaştırma sıralama algoritması olduğu söylenebilir.

Doğrusal zamanda sıralama



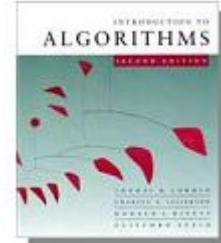
- **Sayma sıralaması (Counting Sort):** Elemanlar arası karşılaştırma yok.
- **Giriş:** $A[1 \dots n]$, burada $A[j] \in \{1, 2, \dots, k\}$.
- **k** , küçük ise iyi bir algoritma olur, **k** , büyük ise çok kötü bir algoritma olur ($n \log n$ daha kötü)
- **Cıkış:** $B[1 \dots n]$, sıralı.
- **Yedek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

Sayma sıralaması



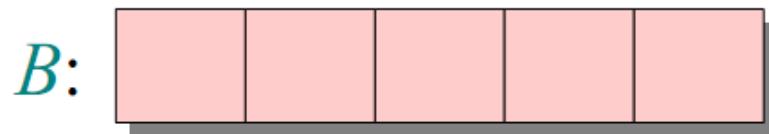
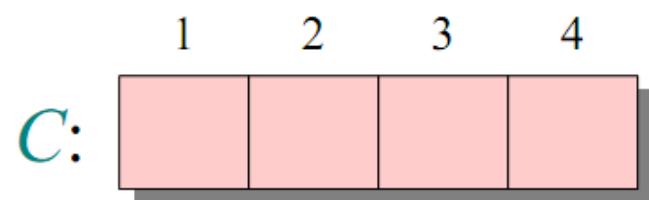
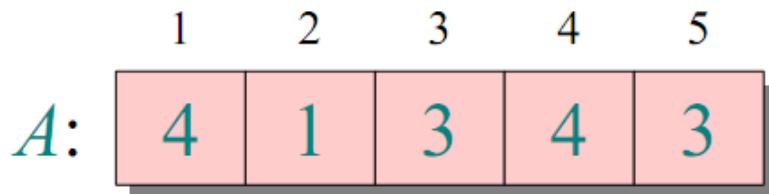
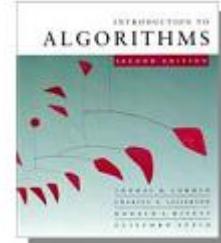
- n adet girişin tamsayı olduğu kabul edilir.
- Girişlerin **0** ile **k** arasında olduğu kabul edilir.
- Temel olarak bir **x** elemanı için kendisinden küçük elemanların sayısını bulmayı amaçlar. Örneğin x elemanından küçük 17 eleman varsa x elemanın doğru yeri 18 olur.
- Girilen dizi boyutunda bir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

Sayma sıralaması



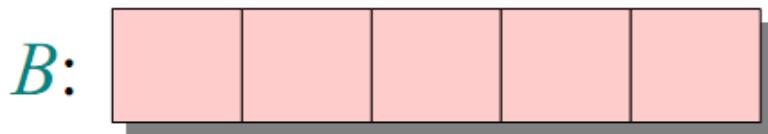
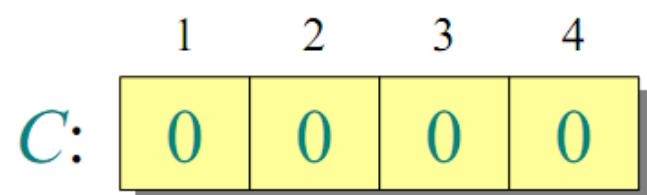
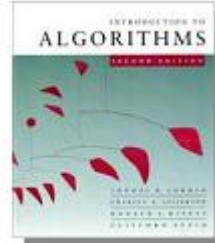
```
for i ← 1 to k
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]] + 1    ▷ C[i] = |{key = i}|
for i ← 2 to k
    do C[i] ← C[i] + C[i−1]      ▷ C[i] = |{key ≤ i}|
for j ← n down to 1          (down to 1: 1'e inene kadar)
    do B[C[A[j]]] ← A[j]
        C[A[j]] ← C[A[j]] − 1
```

Sayma sıralaması



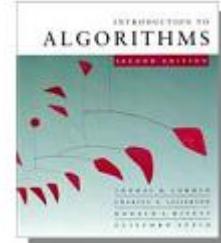
- Dizi girişi 1 ile 4 arasındadır. O zaman $k=4$ olur.

Döngü 1



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

Döngü 2



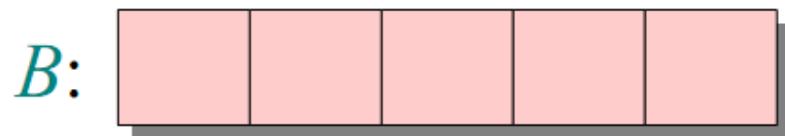
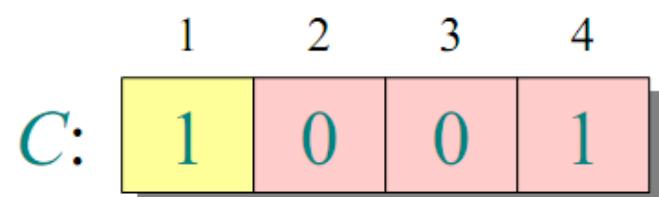
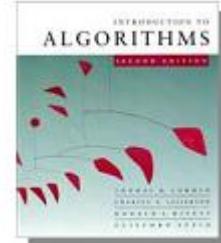
1	2	3	4	5
A: 4	1	3	4	3

1	2	3	4
C: 0	0	0	1

B:				

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

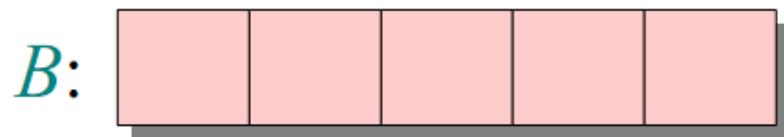
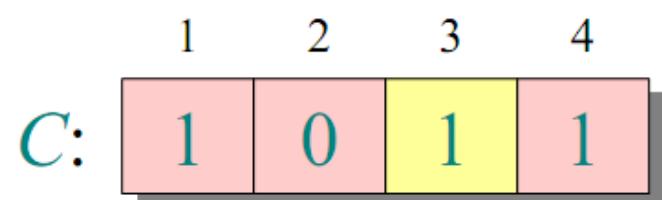
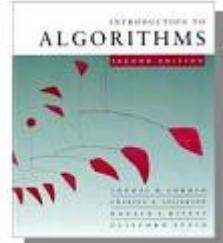
Döngü 2



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

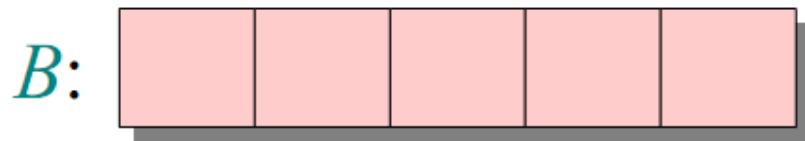
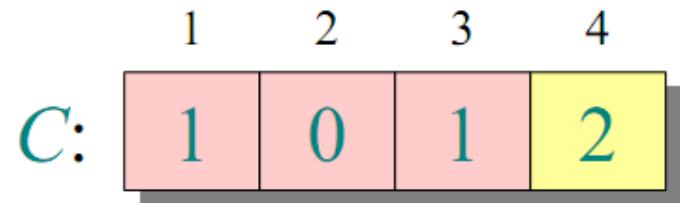
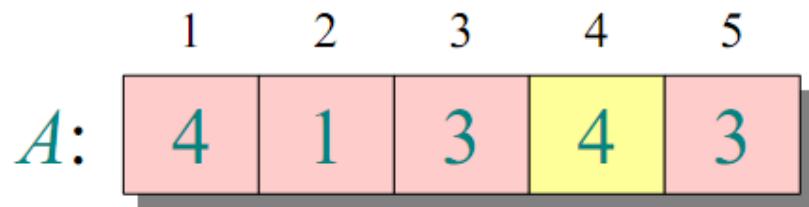
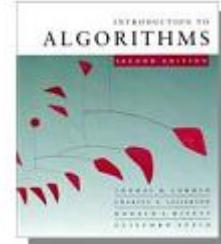
do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleright C[i] = |\{ \text{key} = i \}|$

Döngü 2

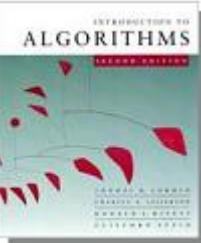


for $j \leftarrow 1$ **to** n
do $C[A[j]] \leftarrow C[A[j]] + 1$ $\triangleright C[i] = |\{ \text{key} = i \}|$

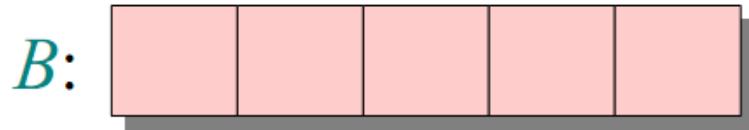
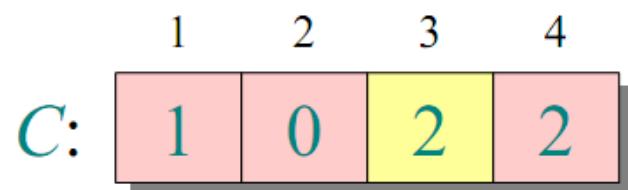
Döngü 2



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

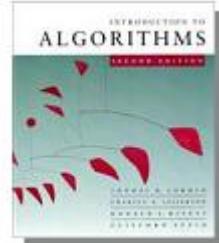


Döngü 2



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

Döngü 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

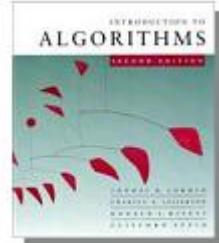
<i>B:</i>					
-----------	--	--	--	--	--

<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



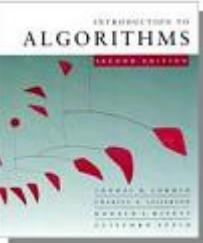
	1	2	3	4	5
$A:$	4	1	3	4	3

	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

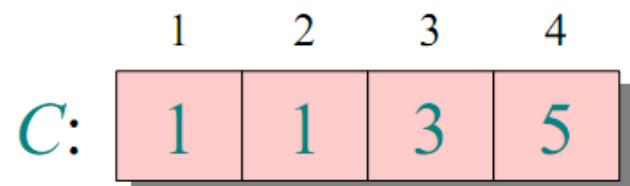
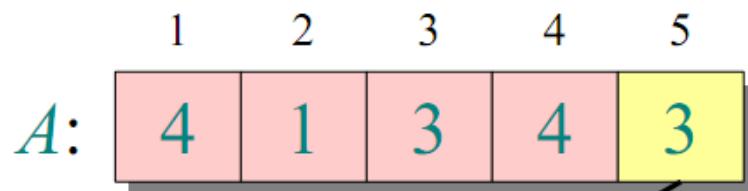
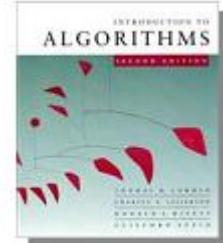
	1	2	3	4
<i>C:</i>	1	0	2	2

<i>B:</i>					

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

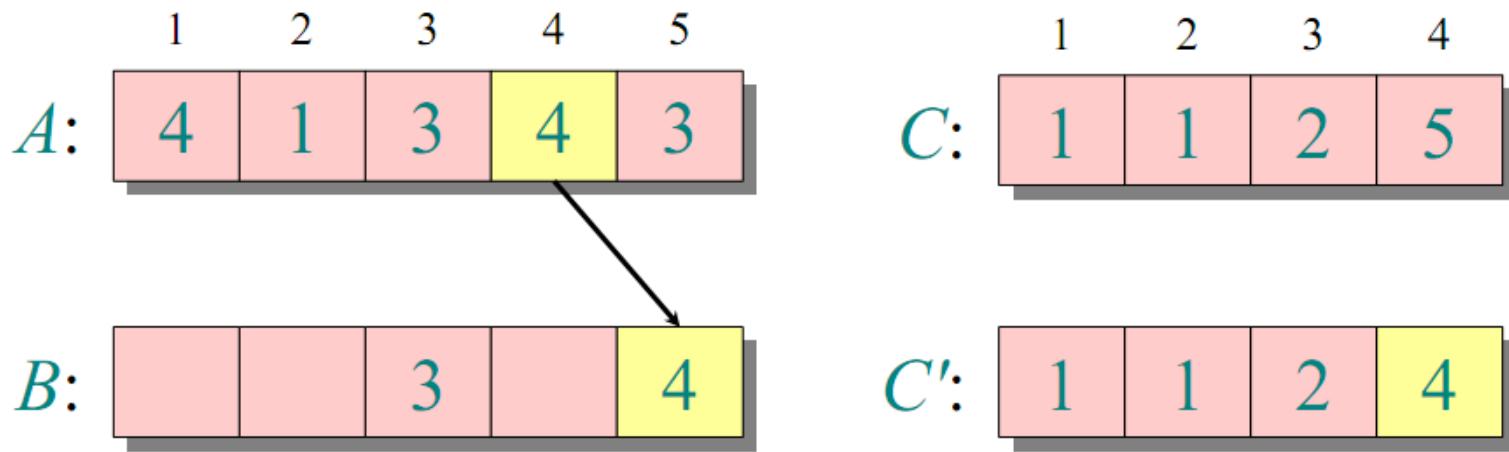
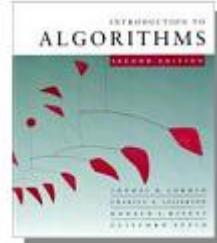
for $i \leftarrow 2$ **to** k
do $C[i] \leftarrow C[i] + C[i-1]$ $\triangleright C[i] = |\{\text{key} \leq i\}|$

Döngü 4



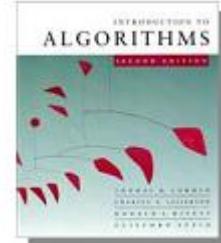
```
for  $j \leftarrow n$  down to 1  
  do  $B[C[A[j]]] \leftarrow A[j]$   
       $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

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

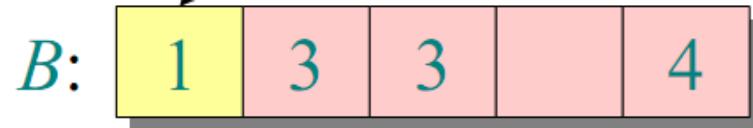
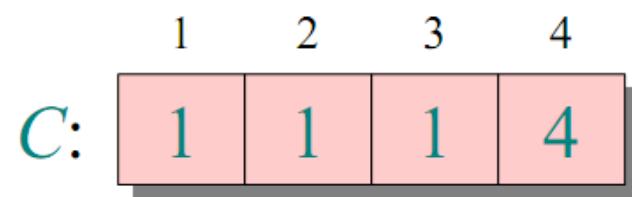
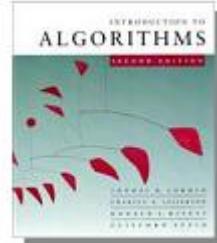
	1	2	3	4
$C:$	1	1	2	4

$C':$	1	1	1	4

```

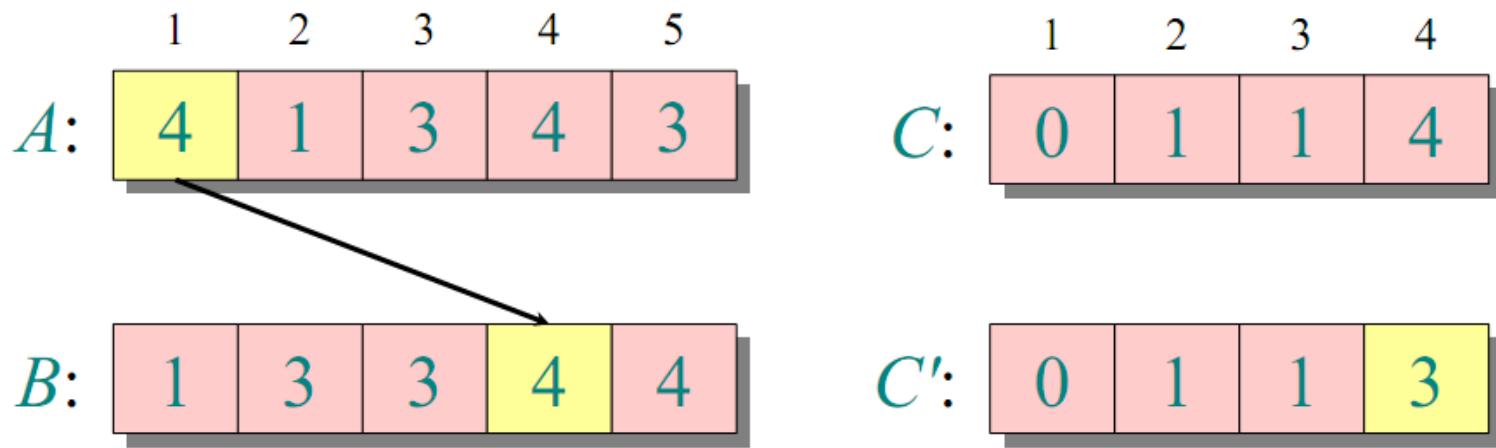
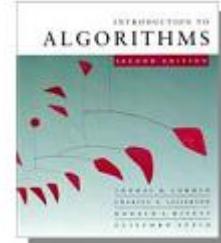
for  $j \leftarrow n$  down to 1
  do  $B[C[A[j]]] \leftarrow A[j]$ 
         $C[A[j]] \leftarrow C[A[j]] - 1$ 
    
```

Döngü 4

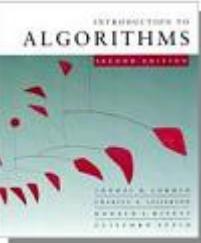


```
for  $j \leftarrow n$  down to 1  
do  $B[C[A[j]]] \leftarrow A[j]$   
     $C[A[j]] \leftarrow C[A[j]] - 1$ 
```

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

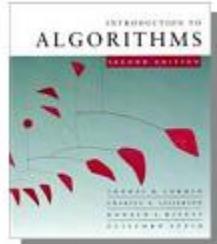
$$\Theta(k) \quad \left\{ \begin{array}{l} \textbf{for } i \leftarrow 1 \text{ to } k \\ \quad \textbf{do } C[i] \leftarrow 0 \end{array} \right.$$

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

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

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

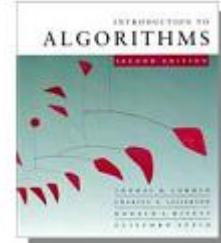
$$\Theta(n + k)$$



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!

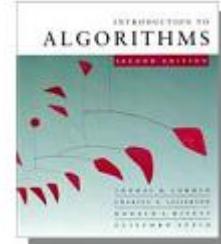
Çalışma Zamanı



```
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

$$\begin{aligned}
 T(n) &= \frac{2}{n} \sum_{k=1}^{n-1} (ak \lg k + b) + \Theta(n) && \text{Yineleme ile çözüm} \\
 &= \frac{2}{n} \sum_{k=1}^{n-1} ak \lg k + \frac{2}{n} \sum_{k=1}^{n-1} b + \Theta(n) && \text{Toplamı dağıt} \\
 &= \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + \frac{2b}{n} (n-1) + \Theta(n) && \text{Toplamı değerlendir:} \\
 &\leq \frac{2a}{n} \sum_{k=1}^{n-1} k \lg k + 2b + \Theta(n) && \mathbf{b+b+\dots+b = b (n-1)} \\
 &&& \text{Çünkü } n-1 < n, \frac{2b(n-1)}{n} < 2b
 \end{aligned}$$

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.