Sıralama Algoritmaları
Çabuk Sıralama,
Rastgele Algoritmaları

Dersi İçeriği

1. Hızlı Sıralama Algoritması
2. Rastgele hızlı sıralama algoritması
3. Heap Sort Algoritması

NİÇİN SIRALAMA

- **Veritabanında arama, sıralı veri üzerinde** binary search (ikili arama) yapabiliriz.
- Eleman tekliğinin sağlanması çiftlerin elimine edilmesi
- Bilgisayar grafik ve geometri hesaplama problemleri
    - En yakın çift closest pair
    - En kısa yol shortest path

Sıralama Algoritmaların Türleri

- **Farklı algoritma teknikleri ile çok sayıda sıralama algoritması geliştirilmiştir.**

Sıralama için alt sınır/en iyi çalışma zamanı (lower bound):

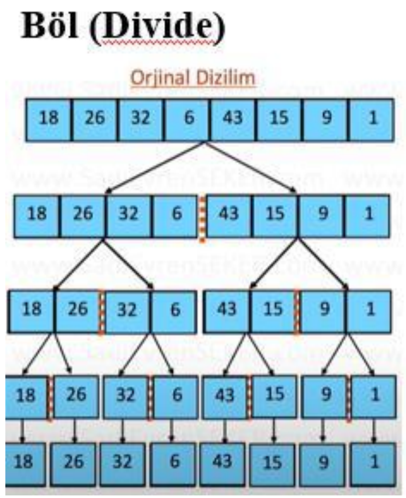
Ω(nlogn) olarak elde edilebilmektedir.

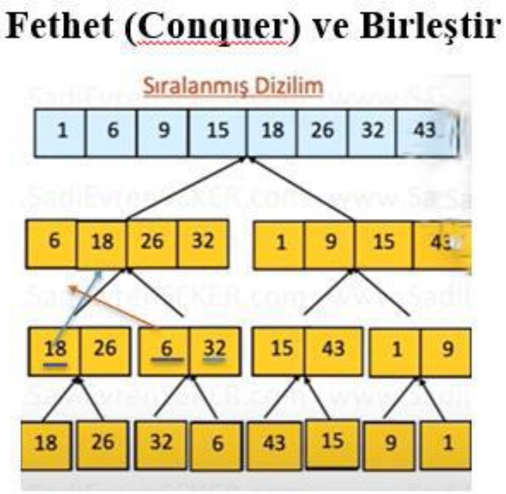
- ***Karşılaştırmaya dayalı sıralama algoritmalar***
  - *HeapSort, quicksort, insertionsort, bubblesort, mergesort,...*
  - *En iyi çalışma zamanı $\theta(n \log n)$*
- **Doğrusal zaman sıralama algoritmaları**
  - *Counting sort(sayma), radix(taban) sort, bucket(sepet) sort.*

Yerinde Sıralama (In-place Sorting)

❖**Yerinde Sıralama:** Algoritmanın boyutu, **θ(n)** olan ve ekstra depolama alan **gerektirmemektedir.**

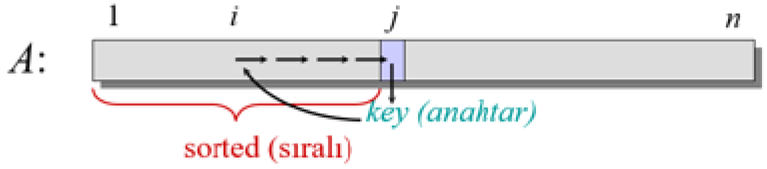
❖Birleştirme Sıralama Algoritması

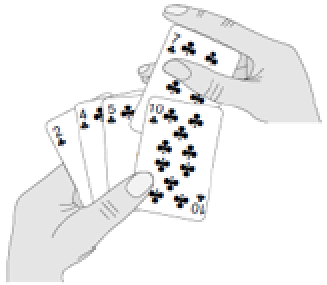




* Ekstra depolama alanı gerektirir.

❖Araya yerleştirme Algoritması





8 2 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

* Ekstra depolama alanı gerektirmez.

Yerinde Sıralama (In-place Sorting)

Algoritma	Yerinde Sıralama
- Bubble sort	Evet
- Insertion sort	Evet
- Selection sort	Evet
- Merge sort	Hayır(ek alan gerekir)
- Heap sort	Evet
- Quick sort	Evet

- **Insertion sort, quick sort , selection sort, bubble sort algoritmalarının**
  - **Worst-case(En kötü) çalışma süresi:** $\theta(n^2)$
- **Merge sort**
  - **Worst-case çalışma süresi:** θ(nlogn)'dir. **Ancak θ(n) boyutunda da ek alan gerektirir.**

1. Çabuk sıralama (QuickSort)

- C.A.R. Hoaretarafından 1962'de önerildi.
- **Böl ve fethet** algoritmasını kullanır.
- "Yerinde" sıralar.
- (Ayar yapılırsa) çok pratiktir.

Sir Charles Antony Richard Hoare
1934 -

1. Çabuk sıralama (QuickSort)

Böl ve fethet

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

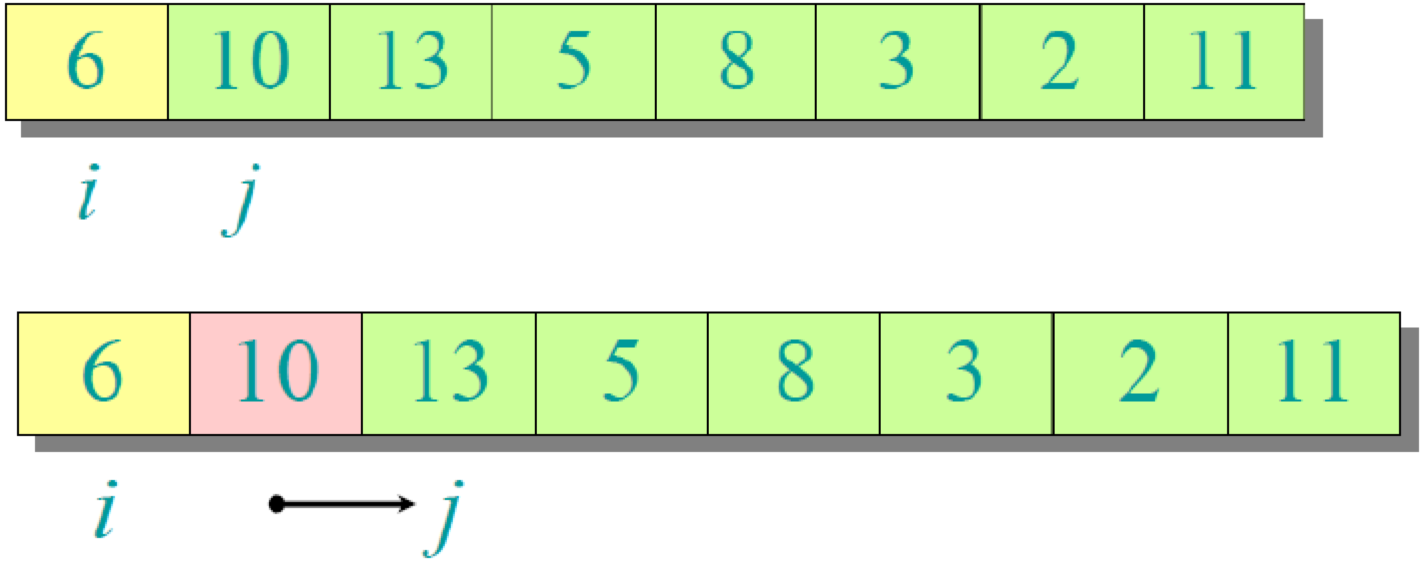
$\leq x$	x	$\geq x$

- ***2. Fethet:*** İki altdizilimi özyinelemeli sırala.
- ***3. Birleştir:*** Önemsiz (yerinde sıraladığı için)

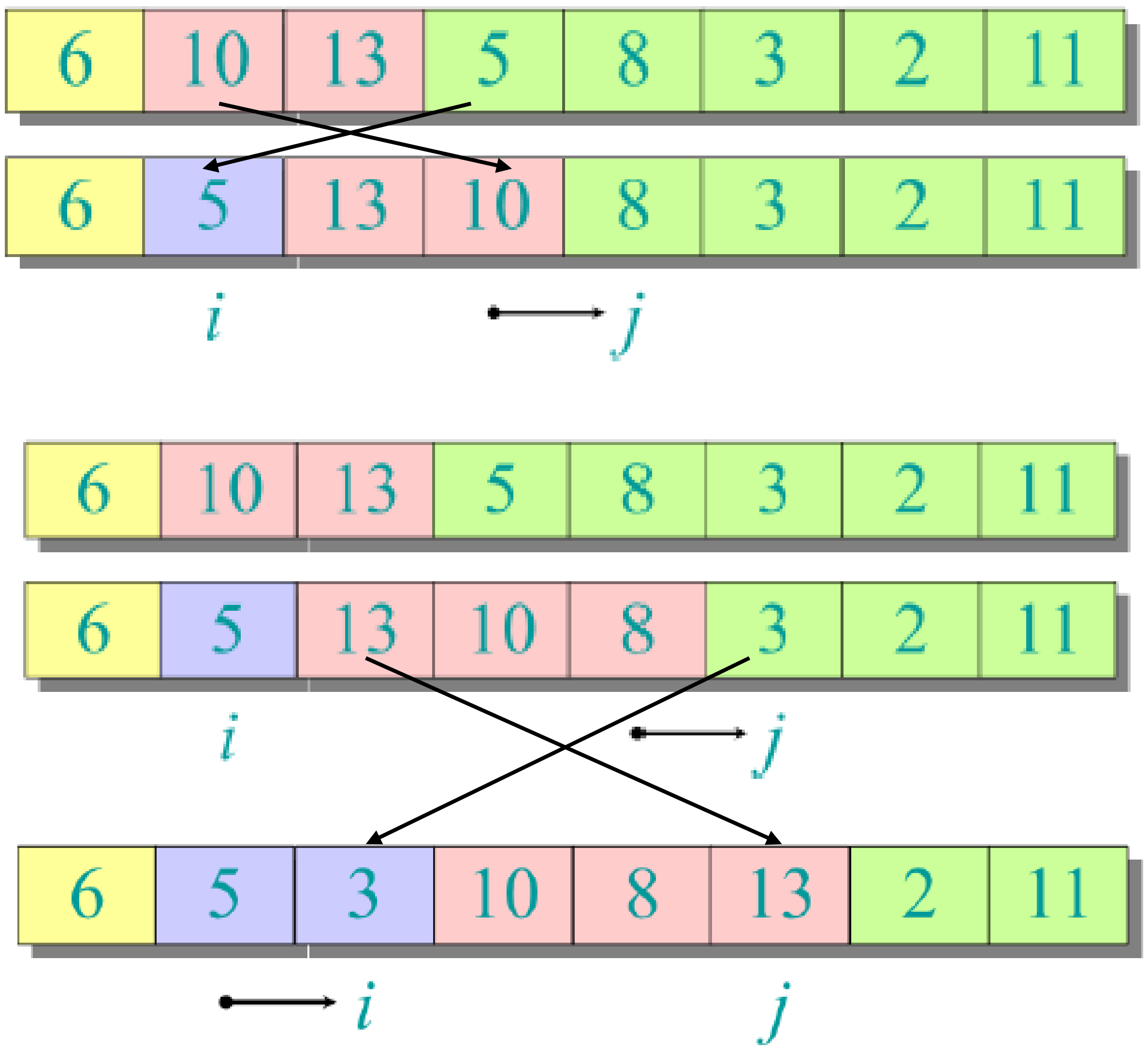
Anahtar: Doğrusal-zamanlı ($\Theta(n)$) bölüntü altyordamı.

Çabuk Sıralamada Bölüntüleme (Partition) Örneği

- **Pivot eleman olarak** ilk eleman seçilmiştir.
- **İki alt grup (sol ve sağ)** oluşturmak için iki pointer kullanılabilir.
- Burada **i** ve **j** indistir.
- Pivottan büyük bir sayı bulunursa **j sağa doğru hareket ettirilir** ve **yer değiştirme işlemi yapılır.**

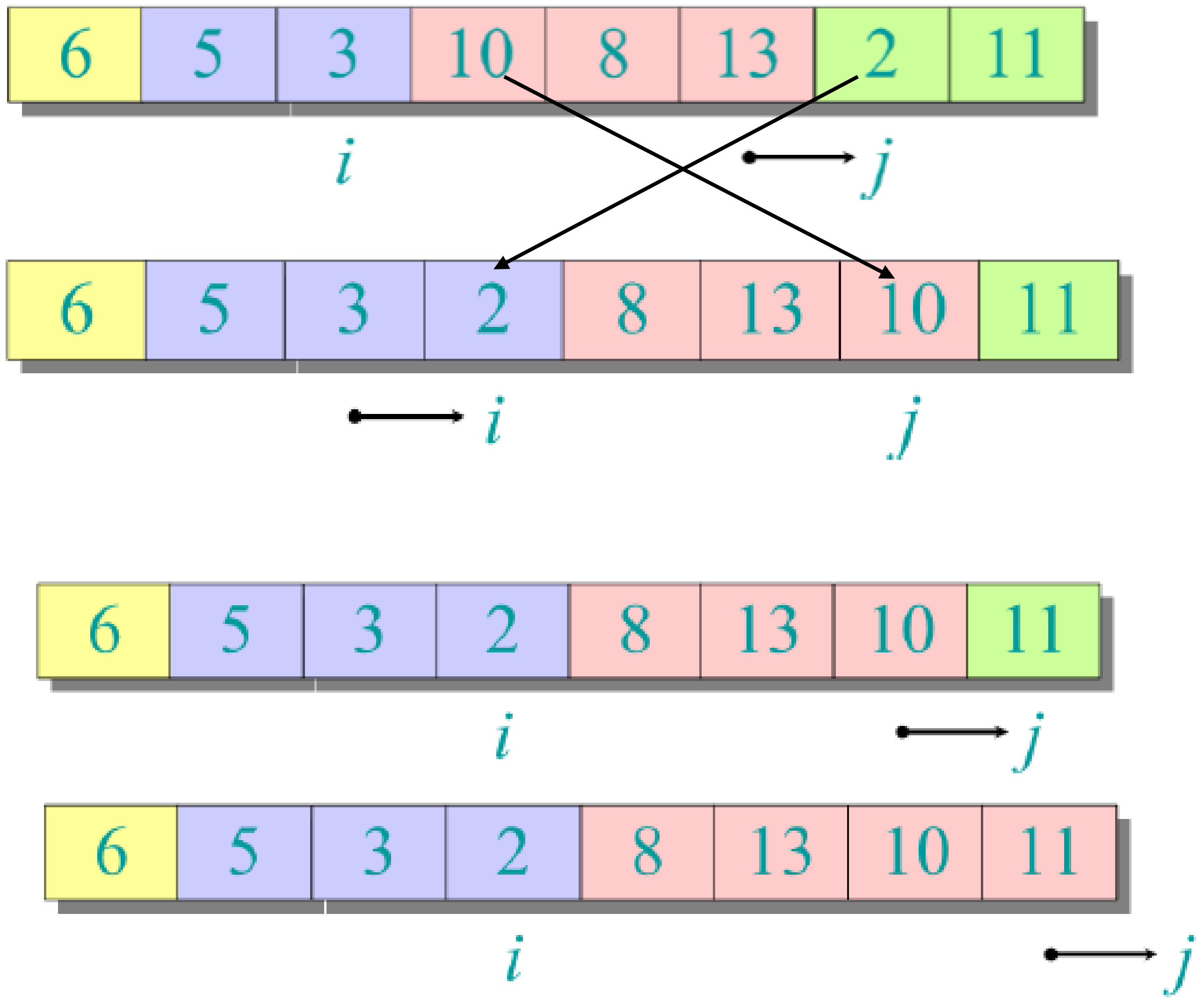


Çabuk Sıralamada Bölüntüleme (Partition) Örneği

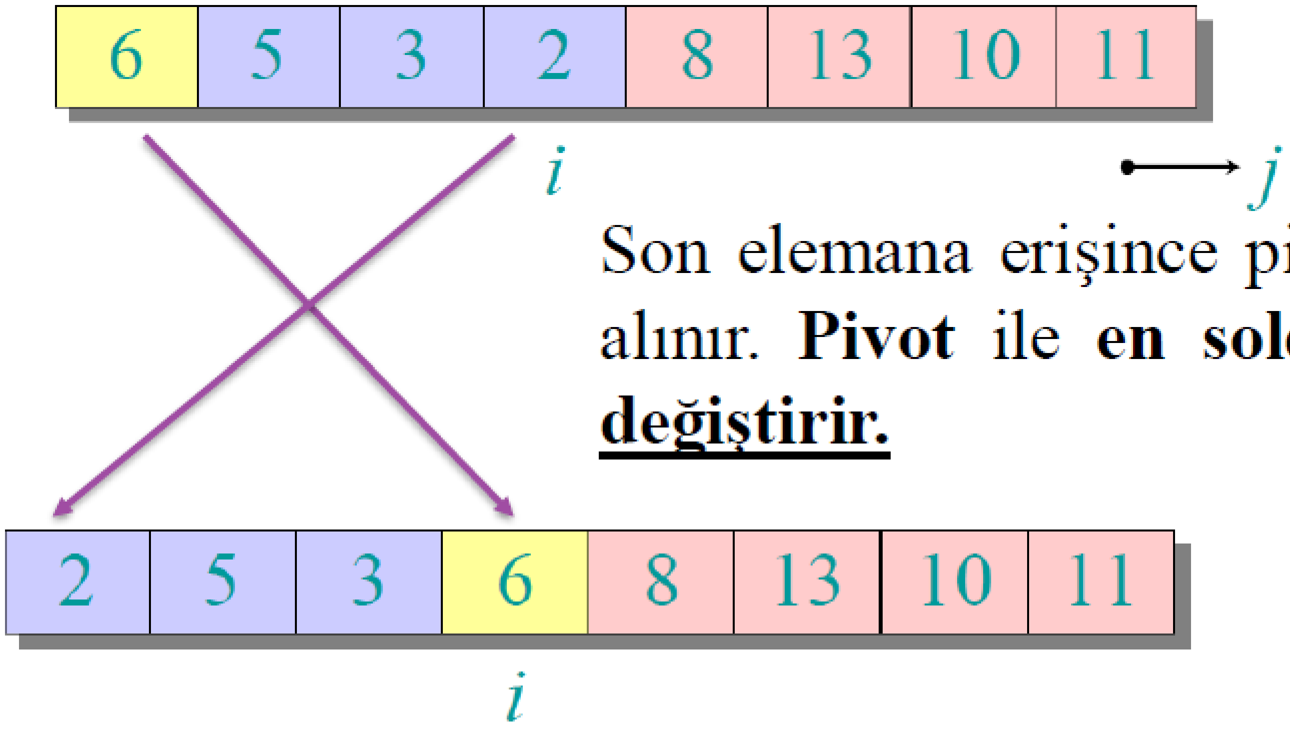


- i sağa kaydırıldı.
- Bir başka ifadeyle i+1. yere j yerleştirildi.

Çabuk Sıralamada Bölüntüleme (Partition) Örneği

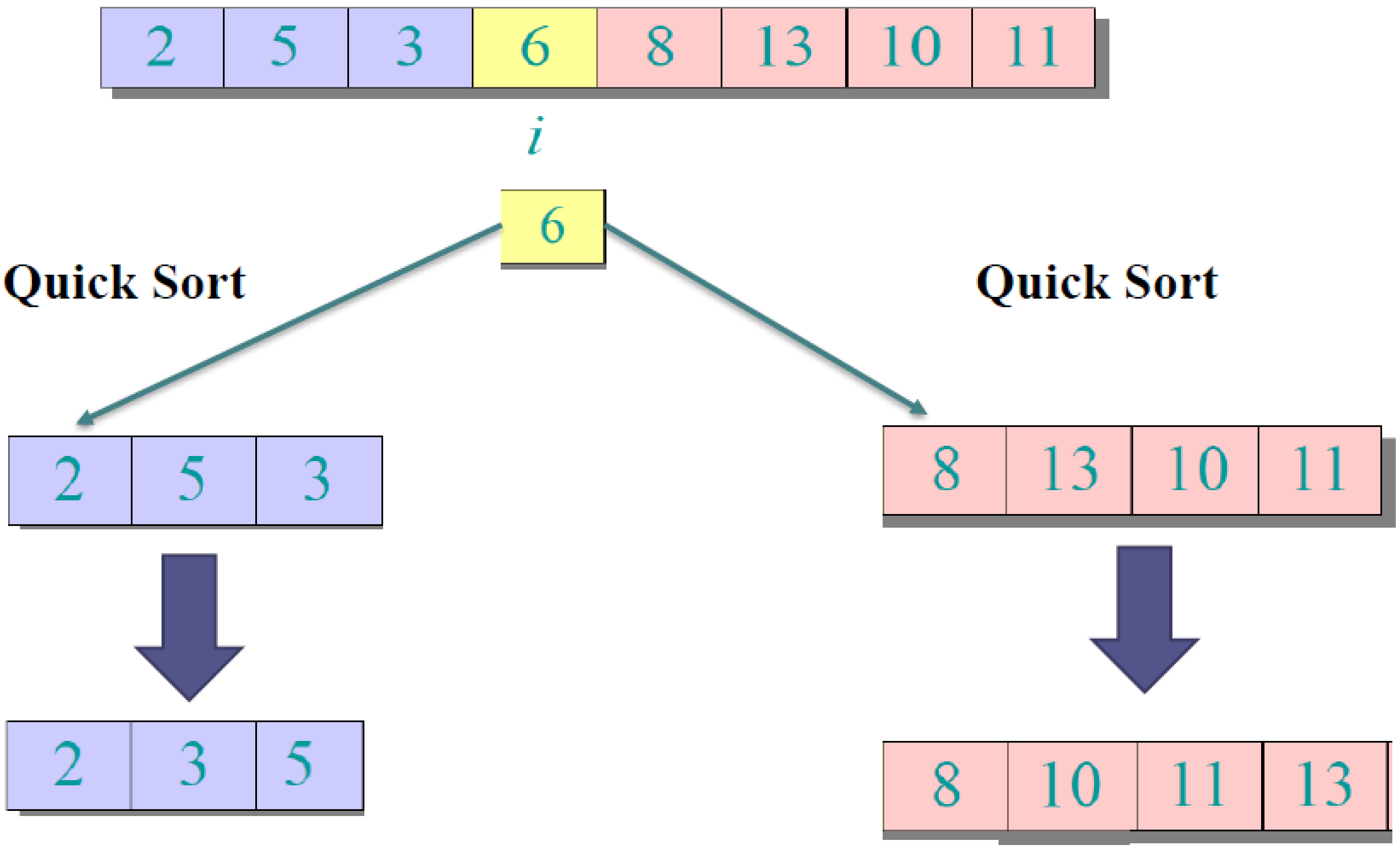


Çabuk Sıralamada Bölüntüleme (Partition) Örneği



Son elemana erişince pivot eleman araya alınır. **Pivot ile en soldaki eleman <u>yer değiştirir.</u>**

Çabuk Sıralamada Bölüntüleme (Partition) Örneği



* Özyinelemeli olarak dizinin orta elamanı seçilme işlemi **yapılır ve en iyi durumda O(lgn) olur.**

Çabuk Sıralama (Quick Sort) Sözde Kodu

```
QUICKSORT(A, p, r)
  if p < r
    then q ← PARTITION(A, p, r)
      QUICKSORT(A, p, q–1)
      QUICKSORT(A, q+1, r)
```

İlk arama: QUICKSORT(A, 1, n)

Çabuk Sıralama Bölüntüleme Örneği

PARTITION (Bölüntü)(A, p, q) $\triangleright$ $A[p \mathinner{.\,.} q]$

- $x \leftarrow A[p]$ $\triangleright$ 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

$= O(n)$

n eleman için

Değişmez:

x	$\leq x$	$\geq x$	?
p	i	j	q

1. Çabuk sıralama (QuickSort)

- 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, $\Theta(n)$.

ÇABUK SIRALAMANIN ÇÖZÜMLENMESİ

- **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 (araya yerleştirme) 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***

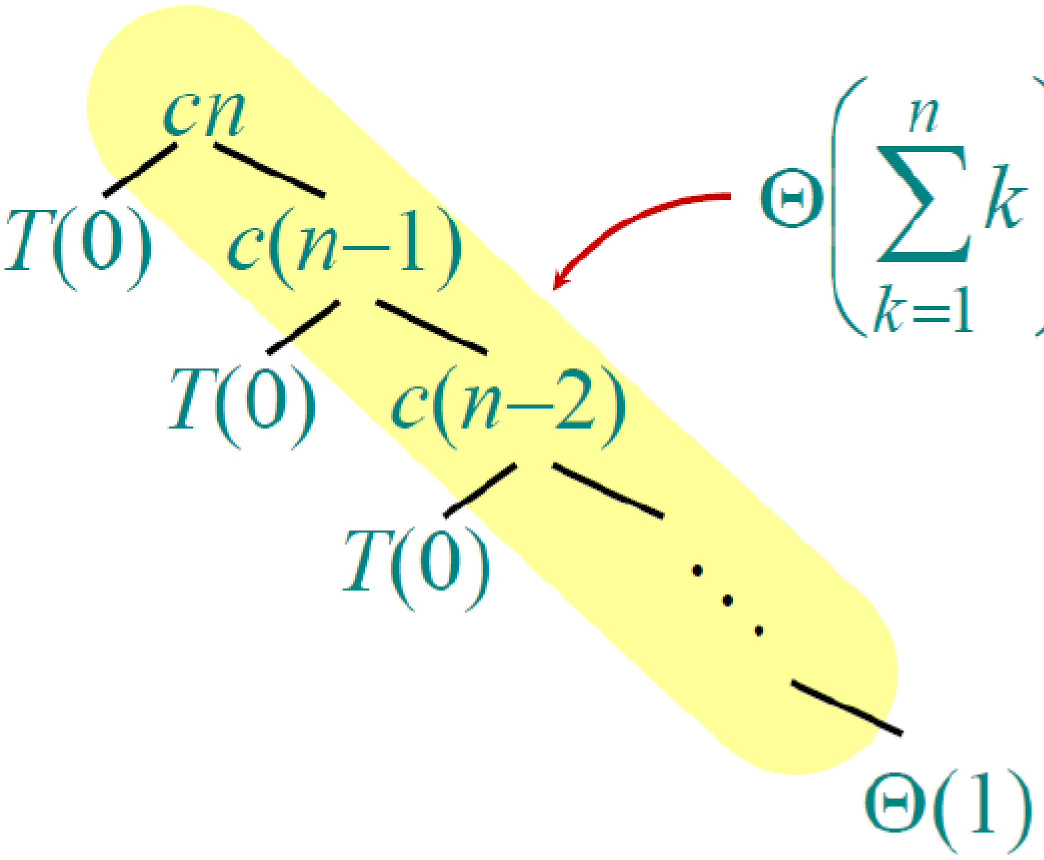
Çabuk Sıralamanın En Kötü Durumu (Worst Case)

- **Bölüntünün bir yanında hiç eleman yok.**

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

En kötü durum özyineleme ağacı

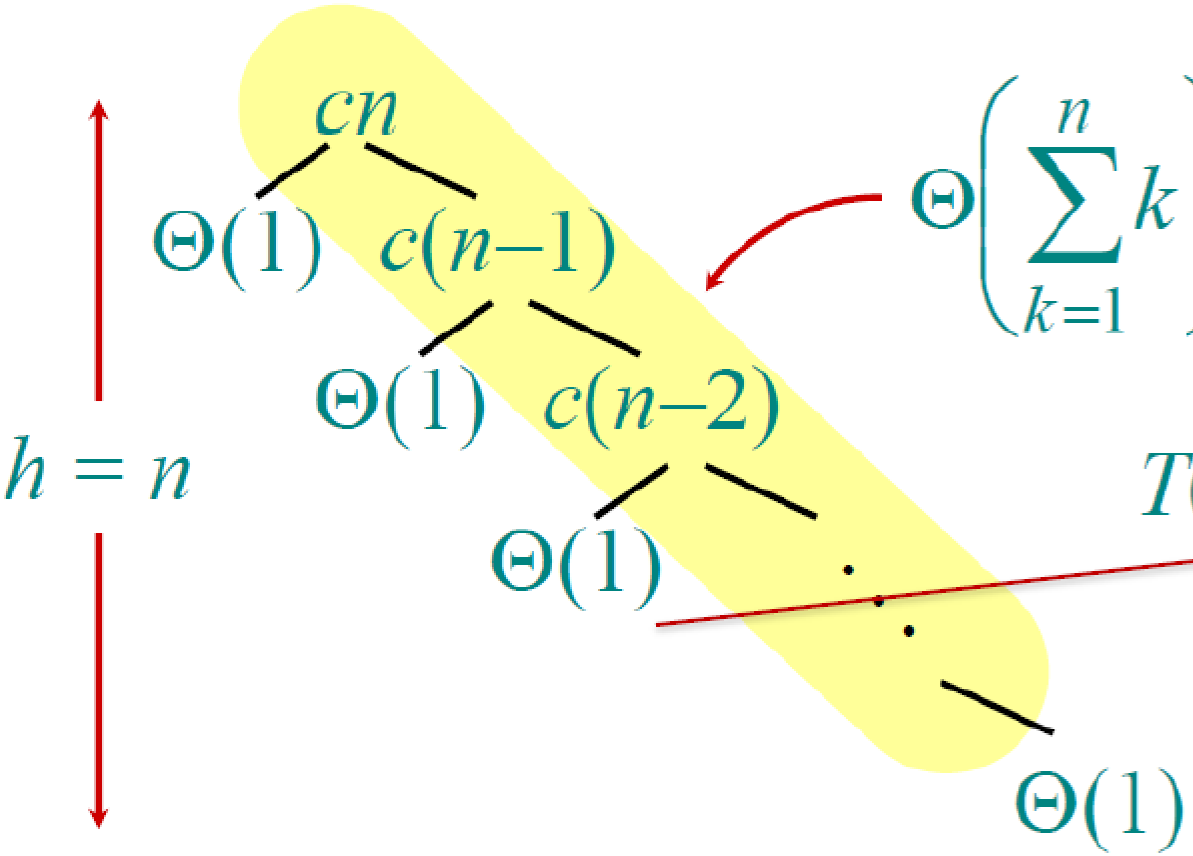
$$T(n) = T(0) + T(n-1) + cn$$



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta\left(n^2\right)$$

En kötü durum özyineleme ağacı

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



$$\Theta\left(\sum_{k=1}^{n} k\right) = \Theta(n^2)$$

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

❖ Çabuk Sıralamanın En İyi Durumu (Best Case)

Eğer şanslıysak, BÖLÜNTÜ dizilimi eşit böler:

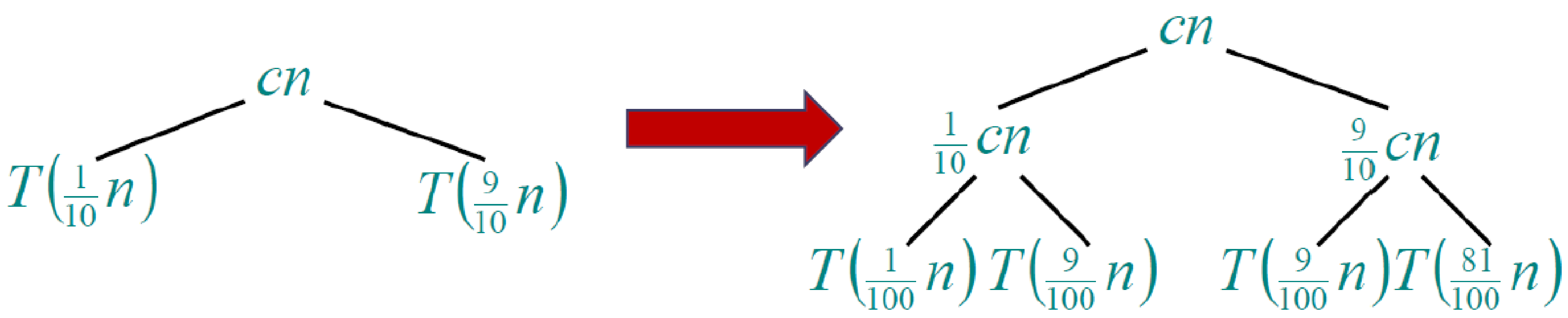
$$T(n) = 2T(n/2) + \Theta(n)$$
$$= \Theta(n \lg n)$$ (birleştirme sıralamasındaki gibi)

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

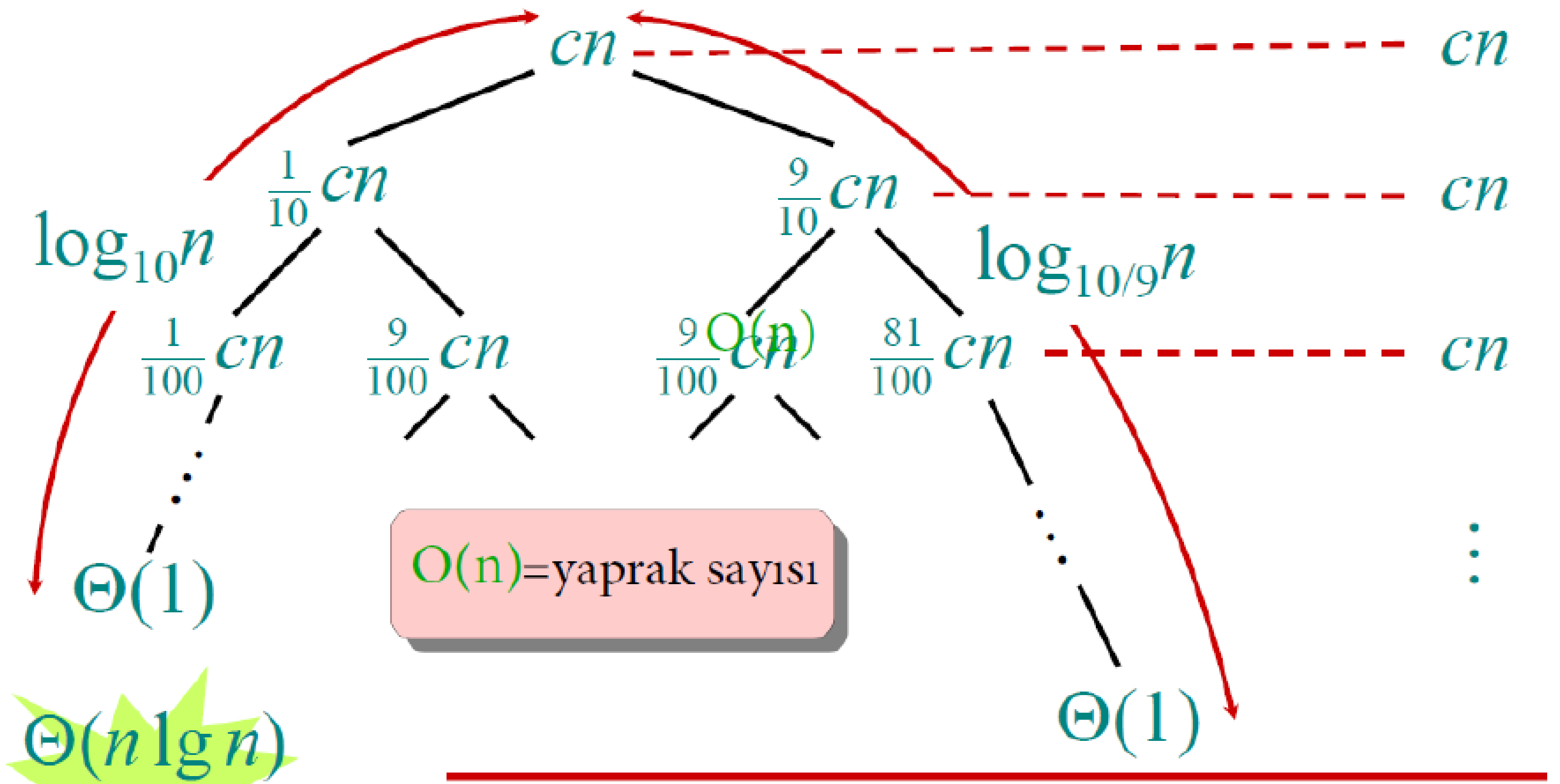
$$T(n) = T\left(\tfrac{1}{10}n\right) + T\left(\tfrac{9}{10}n\right) + \Theta(n)$$

Bu yinelemenin çözümü nedir?

❖ ’En iyiye yakın’ durumun çözümlemesi



’En iyiye yakın’ durumun çözümlemesi



Şanslıyız!

$$cn \log_{10} n \le T(n) \le cn \log_{10/9} n + O(n)$$

Daha fazla sezgi

- En iyi ve en kötü durumların birleşimi: average case

Şanslı ve şanssız durumlar arasında sırayla gidip geldiğimizi varsayalım …

$$L(n) = 2U(n/2) + \Theta(n) \quad \textit{şanslı durum}$$
$$U(n) = L(n-1) + \Theta(n) \quad \textit{şanssız durum}$$

Çözelim: **Yerine koyma metodu U(n/2) değerini hesaplarsak**

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

Şanslı!

Genellikle şanslı olmayı nasıl garanti ederiz?

2. 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.)
- FİKİR: **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 üretecinin çıkışına bağlıdır.

- Bütün elemanların farklı olduğu kabul edilir

- Rastgele seçilen elemanın yakınından bölünür

- Bütün bölme (1:n-1, 2:2-2,…,n-1:1) durumları **1/n** oranında eşit olasılığa sahiptir.

- Rastgele seçilen algoritmanın average-case durumunu iyileştirir.

Rastgele çabuk sıralama Sözde Kodu-Örnek

```
Randomized-Partition(A,p,r)
i←Random(p,r)
exchange A[r]↔A[i]
return Partition(A,p,r)
```

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

Rastgele çabuk sıralama çözümlemesi

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

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

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

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

Rastgele çabuk sıralama çözümlemesi

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

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

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.

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

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

Beklenenin doğrusallığı.

Beklenenin hesaplanması

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

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

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

X_k 'nın diğer değişken seçeneklerden bağımsızlığı.

Beklenenin hesaplanması

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

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

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

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

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

Beklenenin hesaplanması

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

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

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

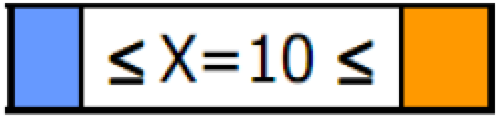
$$= \frac{1}{n}\sum_{k=0}^{n-1} E[T(k)] + \frac{1}{n}\sum_{k=0}^{n-1} E[T(n-k-1)] + \frac{1}{n}\sum_{k=0}^{n-1}\Theta(n)$$

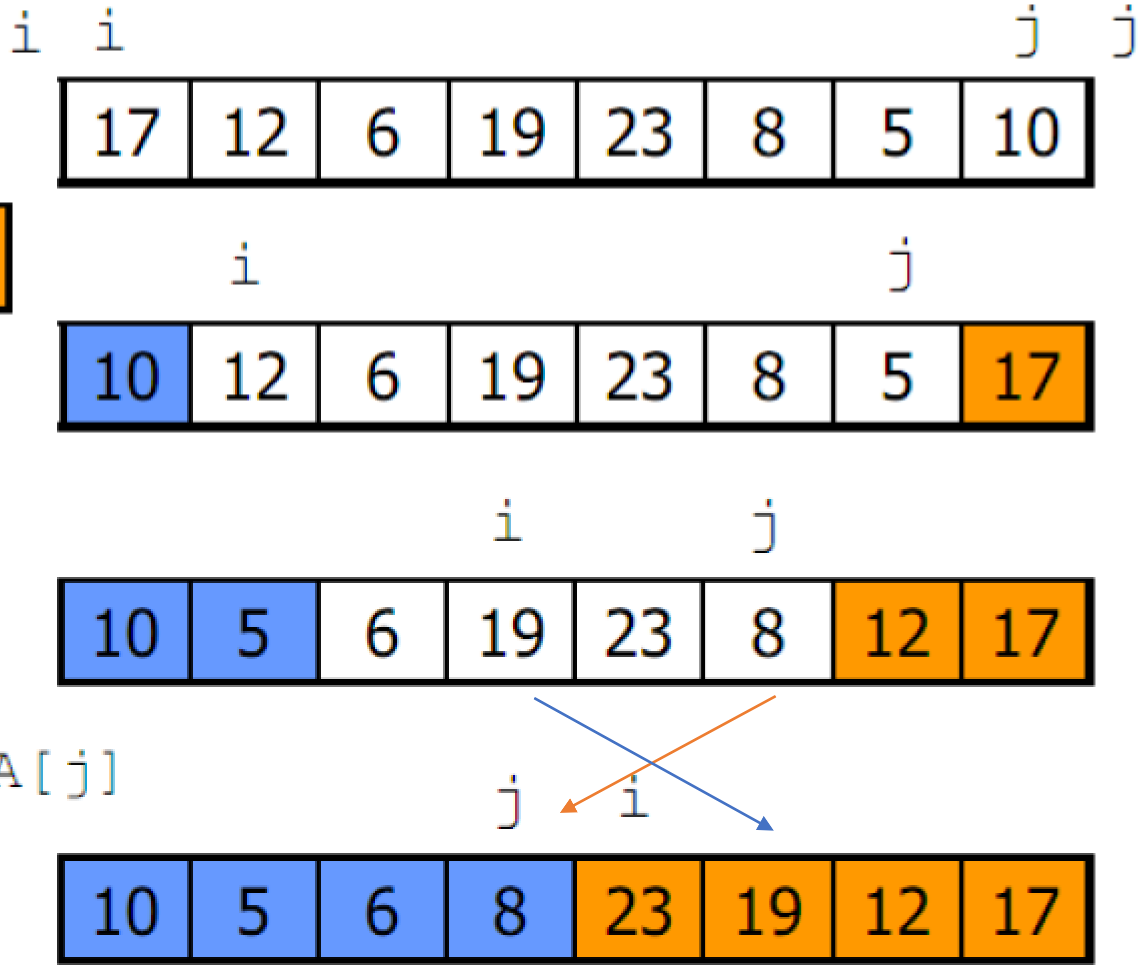
$$= \frac{2}{n}\sum_{k=1}^{n-1} E[T(k)] + \Theta(n)$$

Toplamlarda benzer terimler var.

❖ Çabuk sıralama Bölüntüleme örneği 2 (pivot son elaman)

```
Partition(A,p,r)
   x←A[r]
   i←p-1
   j←r+1
   while TRUE
       repeat j←j-1
           until A[j] ≤x
       repeat i←i+1
           until A[i] ≥x
       if i<j
           then exchange A[i]↔A[j]
           else return j
```





❖ Pratikte çabuk sıralama

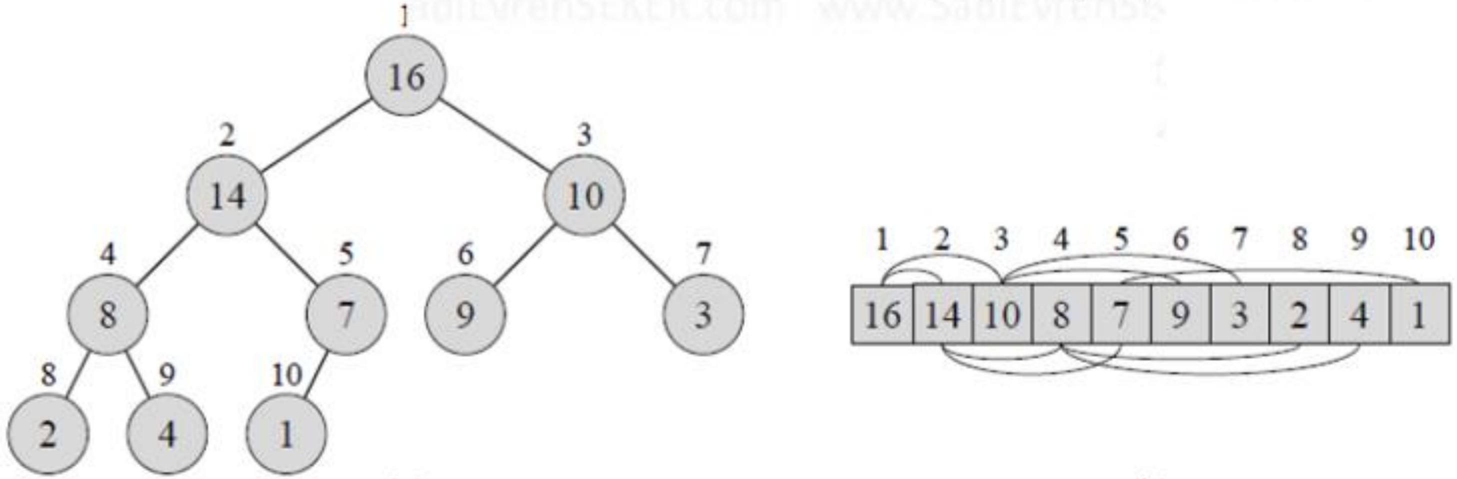
- ❖Çabuk sıralama önemli bir genel maksatlı sıralama algoritmasıdır.
- ❖Çabuk sıralama tipik olarak birleştirme (Merge Sort)sıralamasından iki kat daha hızlıdır.
- ❖Çabuk sıralama **ön bellekleme** ve **sanal bellek uygulamalarında oldukça uyumludur.**

3. Heap (Yığın) Ağacı

- Heap (Yığın), ikili ağaç (binary tree) olarak düşünebileceğimiz bir veri yapısıdır.
- Dizi
- Copmlete 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

– N elemanlı yığın derinliği= $\lfloor \log_2 N \rfloor$.



(a) (b)

Şekildeki a) ikili ağaç ve (b) bir dizi olarak görüntülenen bir maksimum yığın. Ağaçtaki her bir düğümdeki daire içindeki sayı, o düğümde depolanan değerdir. Bir düğümün üzerindeki sayı karşılık gelen sayıdır. Dizinin üstünde ve altında, ebeveyn-alt ilişkilerini gösteren çizgiler bulunur; anne babalar her zaman çocuklarının solundadır. Ağacın yüksekliği üç; indeks 4'teki (8 değerindeki) düğümün yüksekliği birdir.

PARENT(i)
1 **return** $\lfloor i/2 \rfloor$

LEFT(i)
1 **return** $2i$

RIGHT(i)
1 **return** $2i + 1$

Örneğin **i=2** için değer **14** olur.
14'ün **solundaki** değer **2i=4** yani **8**, sağındaki değer **2i+1=5** yani **7** olur.

Örnek

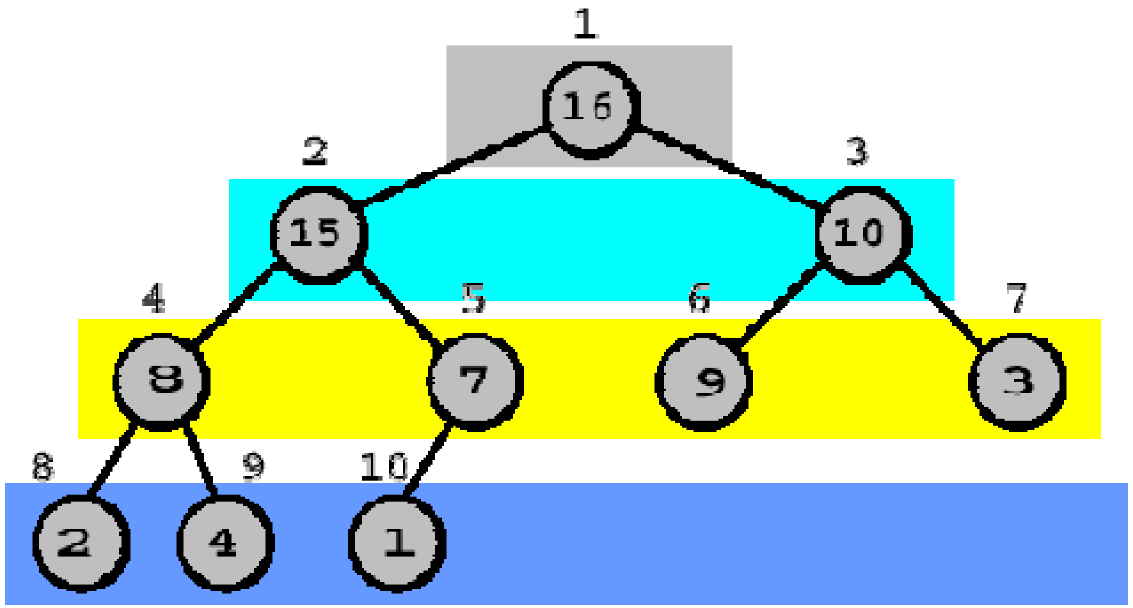
Parent (i)
  return $\lfloor i/2 \rfloor$

Left (i)
  return $2i$

Right (i)
  return $2i+1$

Heap özelliği:

$A[\text{Parent}(i)] \geq A[i]$



1	2	3	4	5	6	7	8	9	10
16	15	10	8	7	9	3	2	4	1

Level: 3 2 1 0

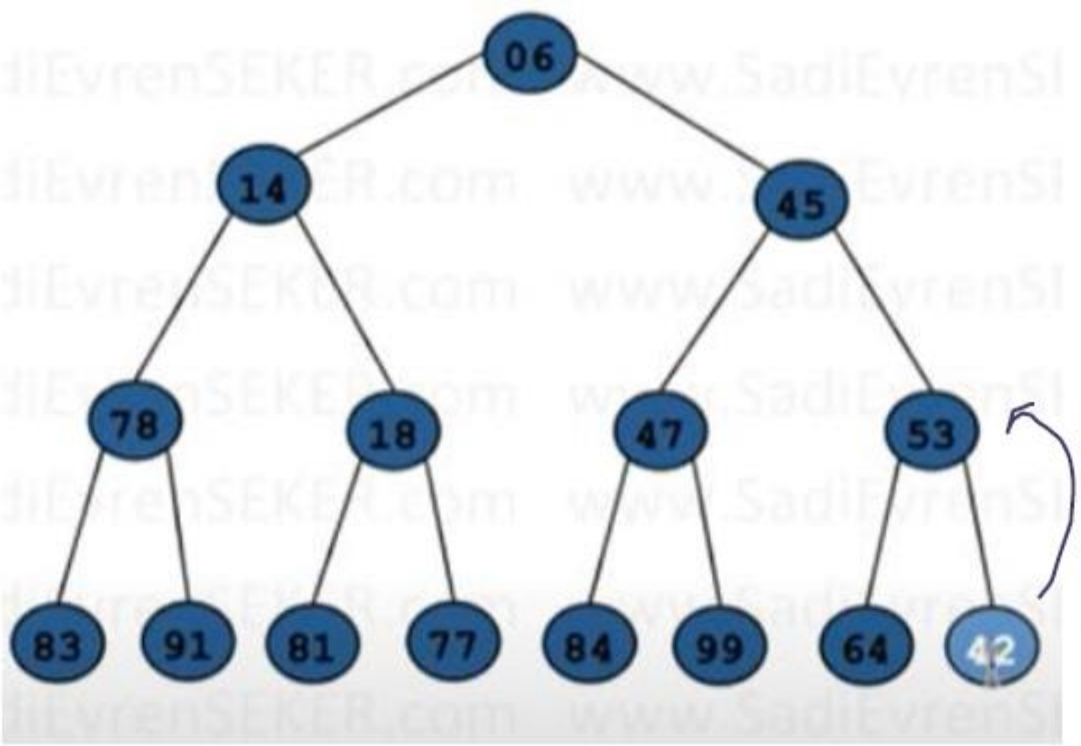
Heap İşlemleri: Heapify

- **`Heapify()`: Temel heap özelliğini korumayı amaçlar.** (A[i] elamanını aşağıya veya yukarı taşıma)
  - Verilen: *i* düğümü (*l* 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 (aşağı taşı).**
  - Çalışma zamanı: **O(*h*), h = height of heap = O(lg *n*)**

Heap İşlemleri: Heapify

1. Ekleme İşlemi

- Örneğin yığın ağacına bir eleman **eklendiği zaman** ilk boş olan yere eklenir.
- Bu durumda yığın ağacı yapısı bozulur. Yukarı taşıma işlemi gerektirir.

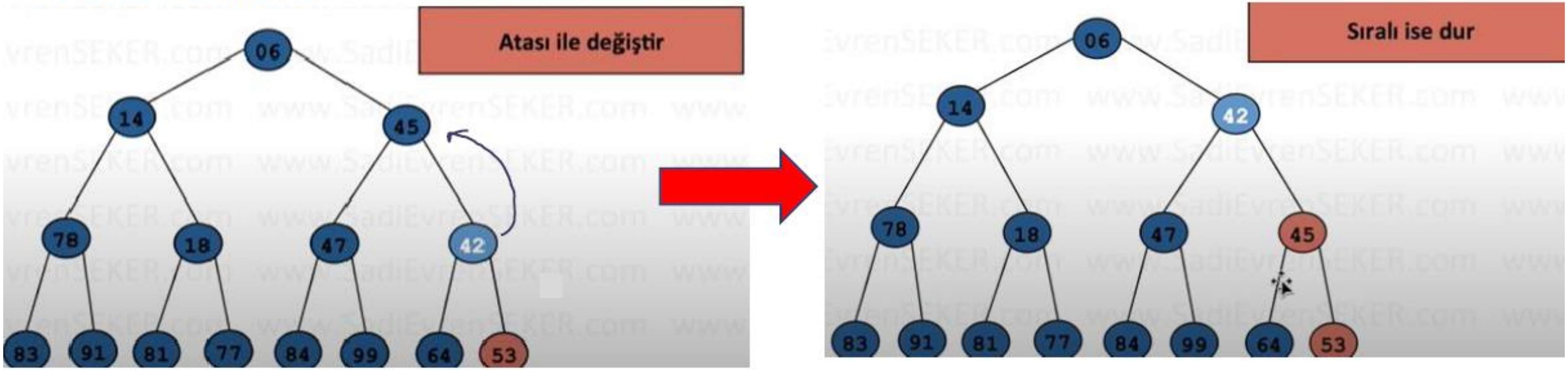


53' 42 den küçük değil!

Boş alan.

Heap İşlemleri: Heapify

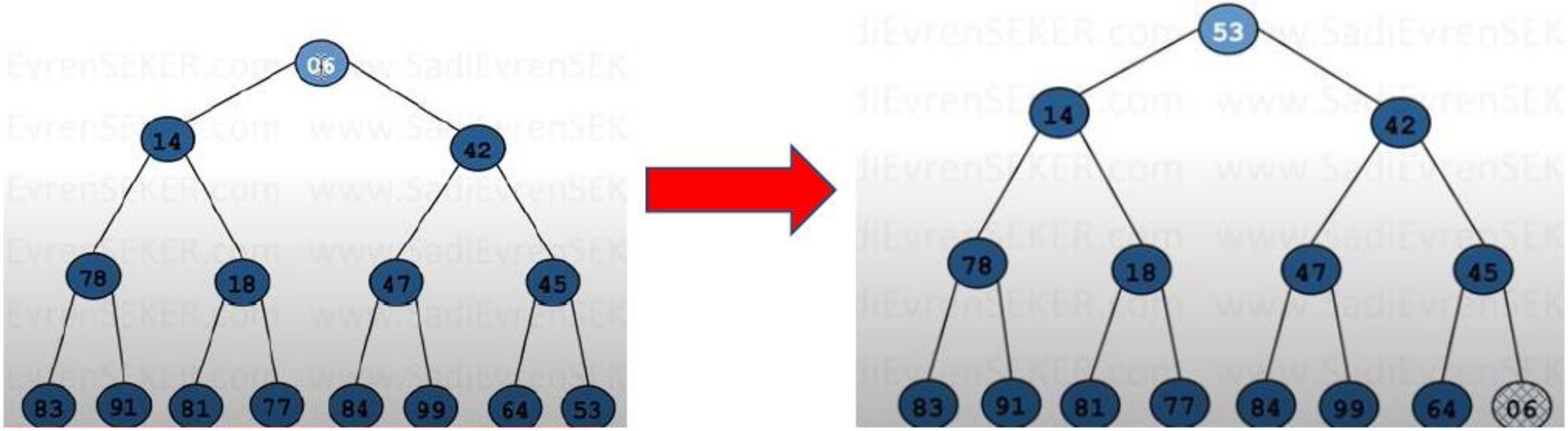
1. Ekleme İşlemi



Burada ağaç derinliği kadar karşılaştırma yapacaktır. Dolayısıyla O(lgn) adımda işlem biter.

Heap İşlemleri: Heapify

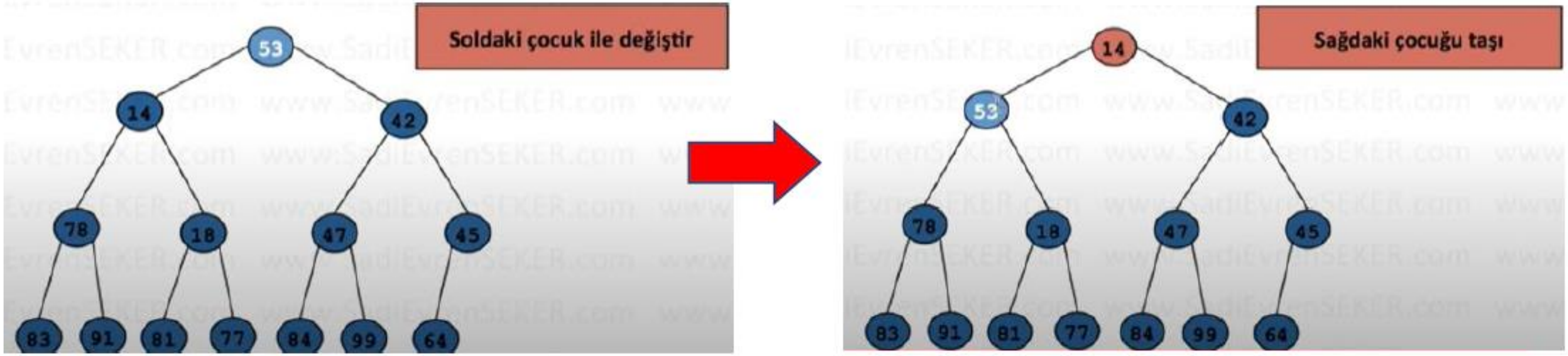
2. Silme İşlemi



*** Dizinin boyutu azaldığı için <u>en sondaki eleman</u> ile <u>kökteki eleman</u> yer değiştirir. Heapfiy işlemini uygula.**

Heap İşlemleri: Heapify

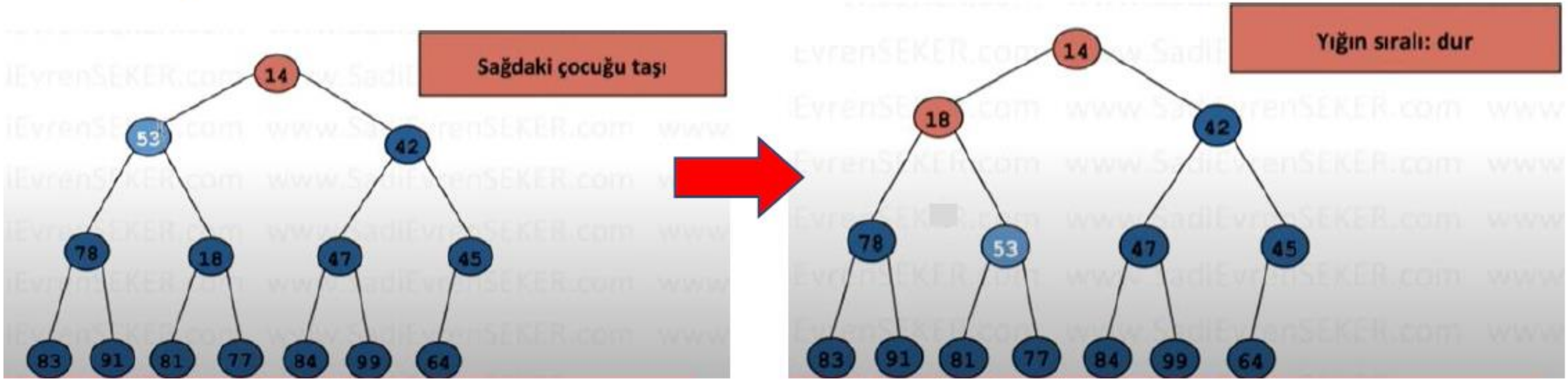
2. Silme İşlemi



* **53 en küçük olmadığı için yer değiştirme işlemini yapacağız.**
* Burada sol ve sağdaki değerlerden hangisi en küçükse onunla yer değiştireceğiz.
* 14 daha küçük olduğu için *53* ile *14* *yer değiştirir.*
* Eğer **53** ile **42'yi** değiştirmiş olsaydık bu seferde **42**, **14**'ten **küçük olmayacaktı.**

Heap İşlemleri: Heapify

2. Silme İşlemi



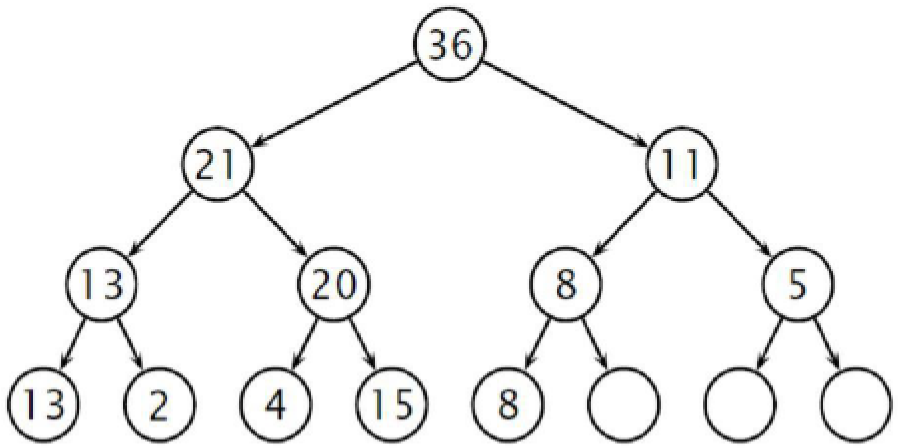
*** Burada kökten başlayıp yaprağa kadar gittiğimiz işlem maliyeti O(lgn) olur.**

Max Heap (Yığın)

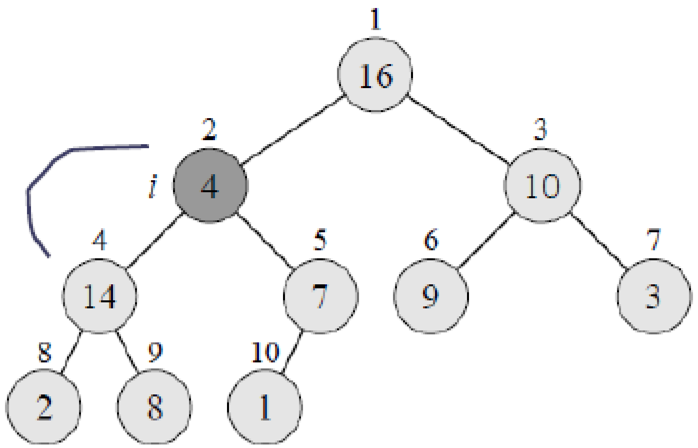
- **Bir maksimum yığında**, maksimum yığın özelliği, **kök dışındaki her i düğümü için,**

  $A[\text{PARENT}(i)] \geq A[i]$

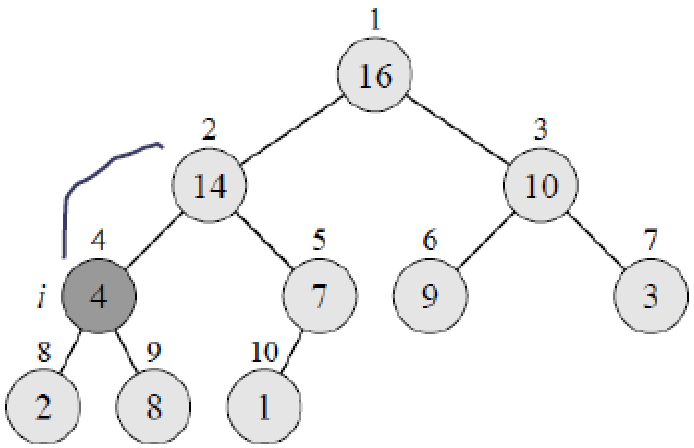
- yani, bir düğümün değeri, **en fazla ebeveyninin değeridir.**
- Böylece, ***bir maksimum yığındaki*** **en büyük öğe kökte depolanır ve alt ağaç köklenir.**



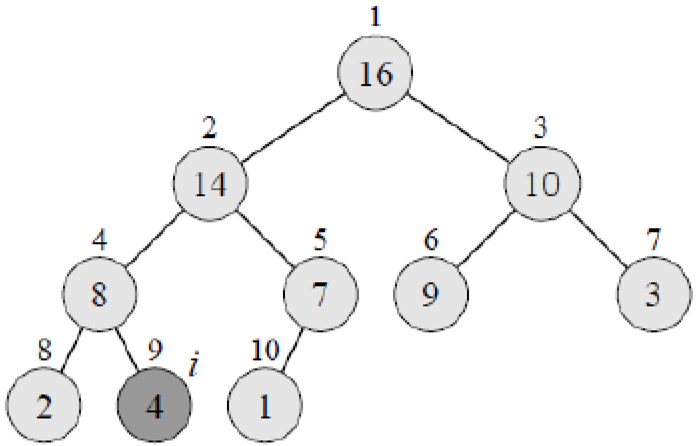
Max-Heapfiy (Yığın ağacı)



(a)



(b)



(c)

Şekil’de A.heapsize=10,
(a)MaxHeapSize(A,2) (b) A[2] ile A[4] yer değiştirmesi
(c) A[4] (8) ile A[9] (4) yer değiştirmesi

Max-Heapfiy Sözde Kod

```
MAX-HEAPIFY(A, i)
l = LEFT(i)
r = RIGHT(i)
if l ≤ A.heap-size and A[l] > A[i]
    largest = l
else largest = i
if r ≤ A.heap-size and A[r] > A[largest]
    largest = r
if largest ≠ i
    exchange A[i] with A[largest]
    MAX-HEAPIFY(A, largest)
```

MAX-HEAPFIY Analizi

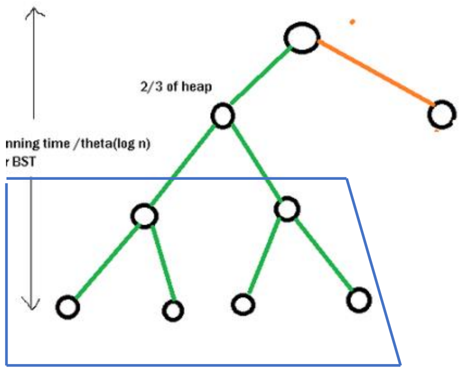
- Verilen bir i düğümde köklenmiş *n* boyutlu bir alt ağaç üzerinde MAX-HEAPFIY ‘nin çalışma süresi, **A[i]**, **A[LEFT(i)]** ve **A[RIGHT(i)]** arasındaki ilişkiyi sağlayan **θ(1)**’dir.
- Ayrıca bu zaman ***i*** düğümünün çocuklardan birinde köklenmiş ağaç üzerinde

MAX-HEAPFIY ‘yi çalıştırmak içindir (öz yinelemeli çağrının gerçekleştiğini varsayarak).

MAX-HEAPFIY Analizi

- *Çocuklardan alt ağaçlardan her biri* ***en fazla*** **2n/3** boyuta sahiptir.
- **Alt ağacın seviyesi** ***tam olarak yarı dolu*** olduğunda ***en kötü çalışma süresi*** meydana gelir.
- Bu nedenle MAX-HEAPFIY çalışma zamanını yenileyerek

$$T(n) \leq T(2n/3) + \Theta(1)$$

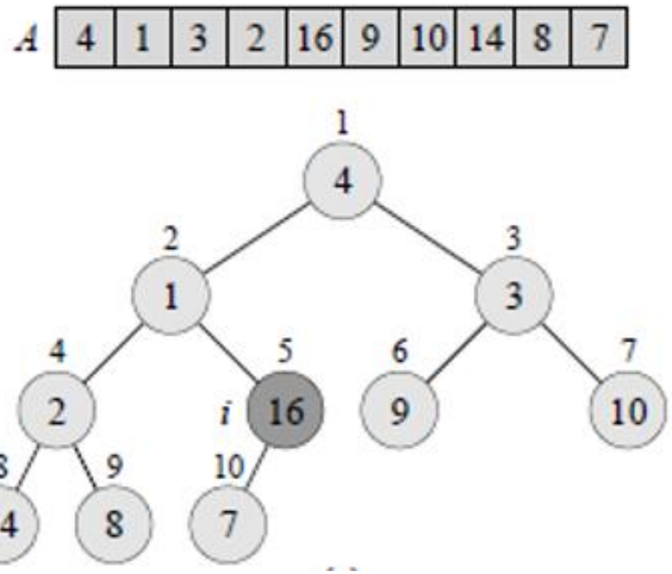


- Burada a=1, b=3/2 dir. Böylece $n^{log_b a} = n^{log_{3/2} 1} = \theta(n^0) = 1$ ve $f(n) = 1$ olduğundan master teoriminde **Durum 2** uygulanır ve çözüm, $T(n) = \theta(logn)$

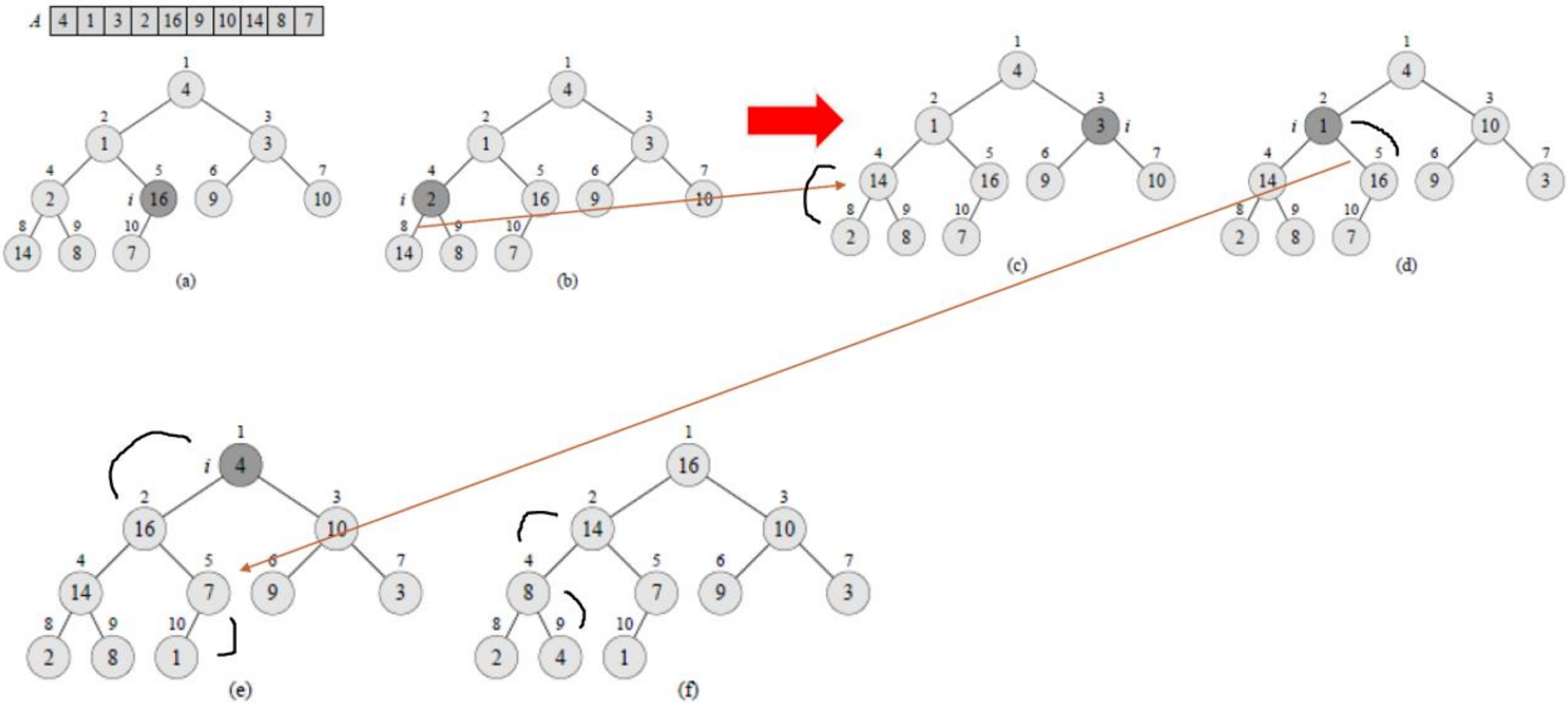
Yığın (Heap) İşlemleri : Heap Yapılandırması BuildHeap()

- A[1..n] dizisinin n=lenght[A] uzunluğunda olan bir heap' dönüştürülmesi.
- Alt dizideki A[(⌊ n / 2 ⌋+1)…n] elamanlar heap durumundadır.

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



Yığın (Heap) İşlemleri : Heap Yapılandırması BuildHeap()



BuildHeap()Analizi

- Tanımlamalar
  - node yüksekliği: node'dan yaprağa giden en uzun yol
  - tree yüksekliği: root'un yüksekliği

$$
\begin{array}{ll}
\multicolumn{2}{l}{\text{BUILD-HEAP}(A)} \\
1 & \textbf{for } i \leftarrow \lfloor n/2 \rfloor \textbf{ downto } 1 \\
2 & \quad\quad \textbf{do } \text{HEAPIFY}(A, i)
\end{array}
$$

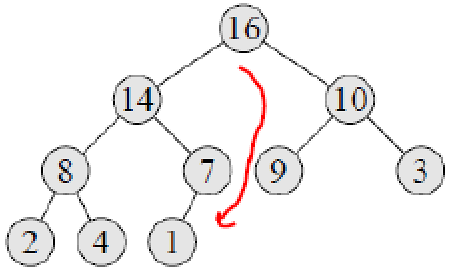
  - heapify süresi = O(i root node'unun subtree yüksekliği (k))
  - $n = 2^k - 1$ olur (complete binary tree $k = \lfloor lg\ n \rfloor$)

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

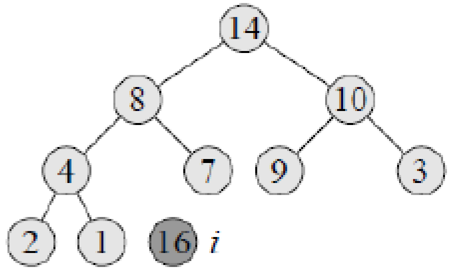
Heap Sort Algoritması

Dizinin 1. **elemanı olan 16**’yı sonuncu eleman yaptı. **Max-Heap’ta en tepede en büyük eleman** vardı.

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```



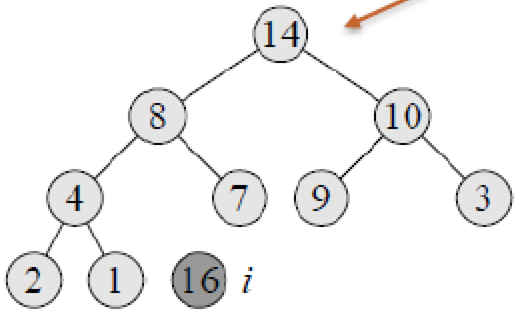
(a)



(b)

1. Eleman sona atılır. Sonuncu elaman 1. eleman yapılır.

Yığın boyutu **bir azaltılır**. Yani İlgili **düğüm (16) kaldırılır.**

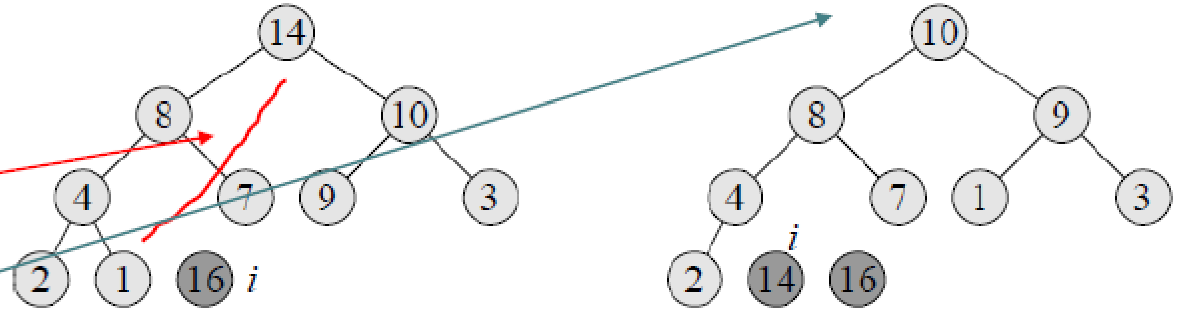


(b)

1. Elemandan itibaren Max-heapify uygulanır.

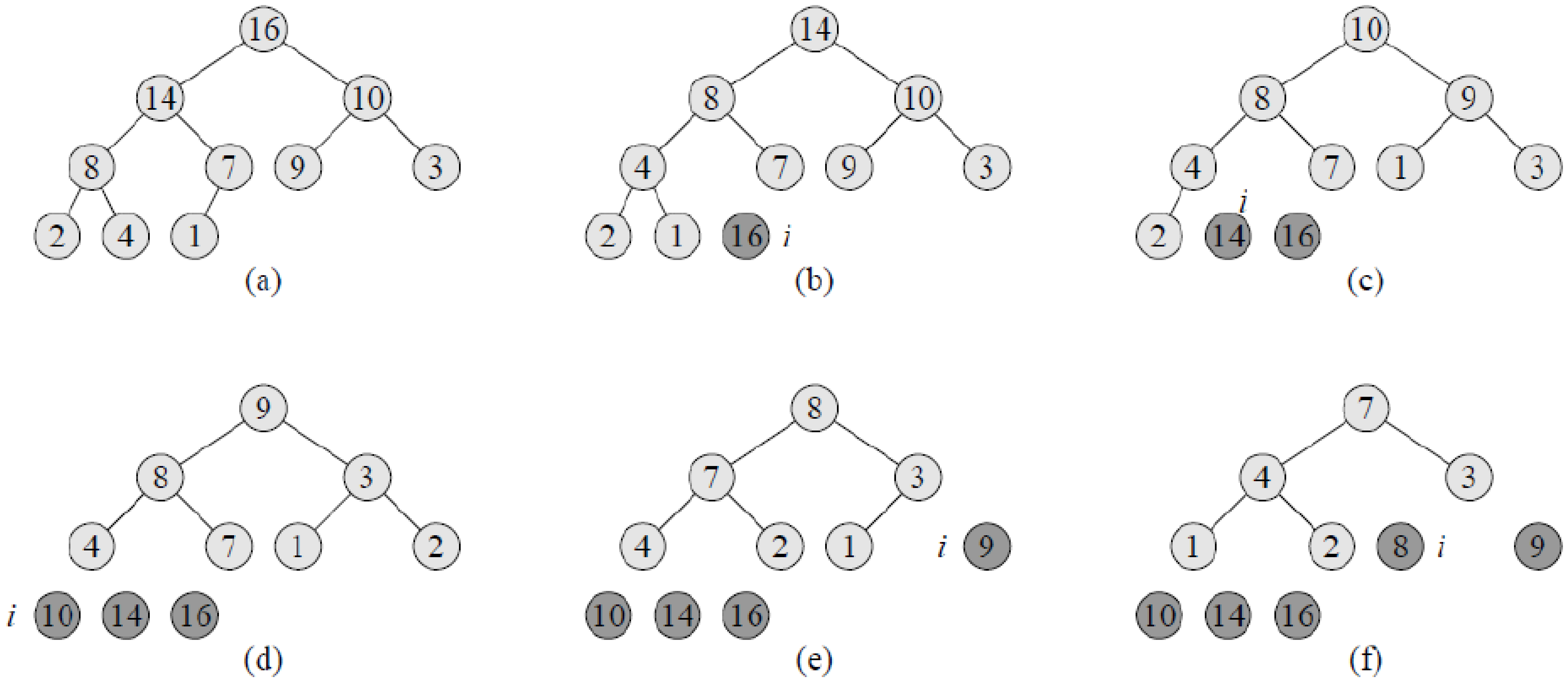
Heap Sort Algoritması

```
HEAPSORT(A)
BUILD-MAX-HEAP(A)
for i = A.length downto 2
    exchange A[1] with A[i]
    A.heap-size = A.heap-size − 1
    MAX-HEAPIFY(A, 1)
```

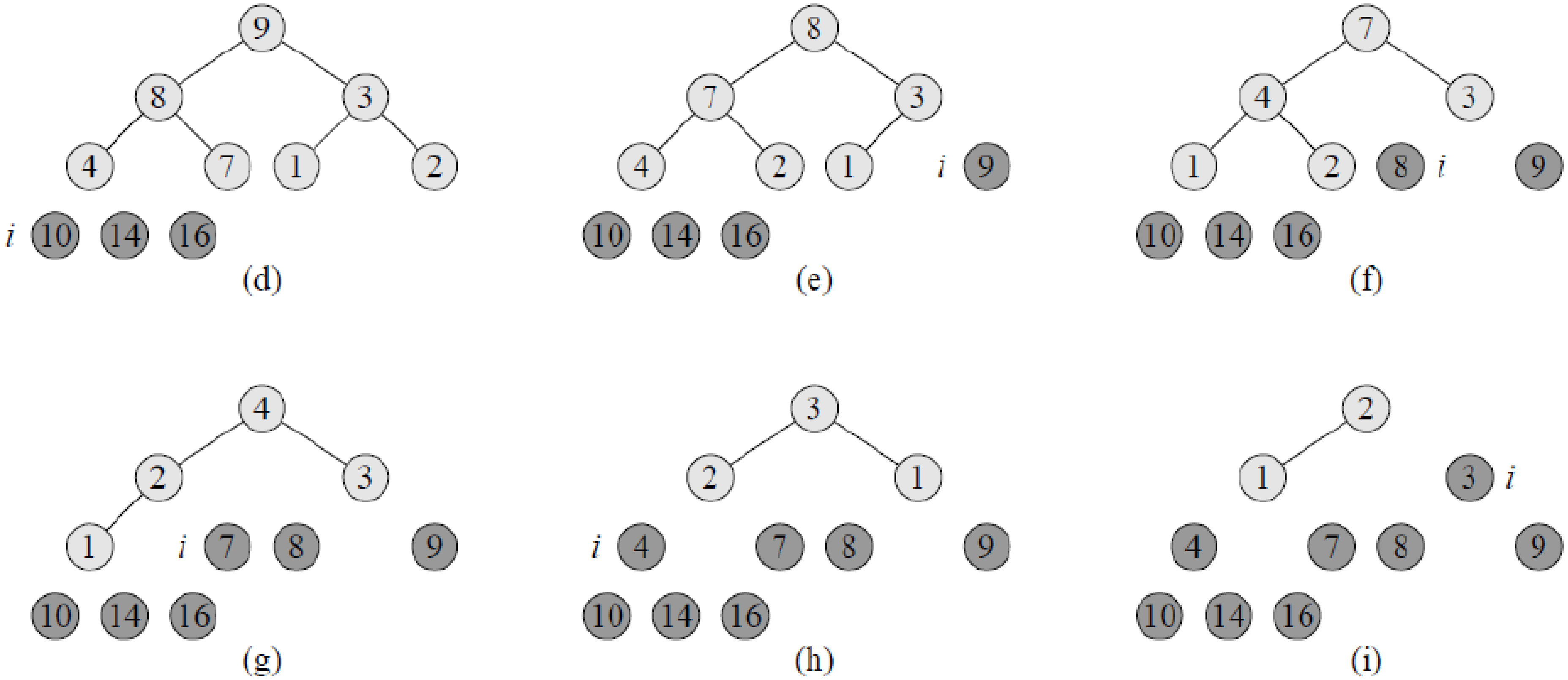


(b) (c)

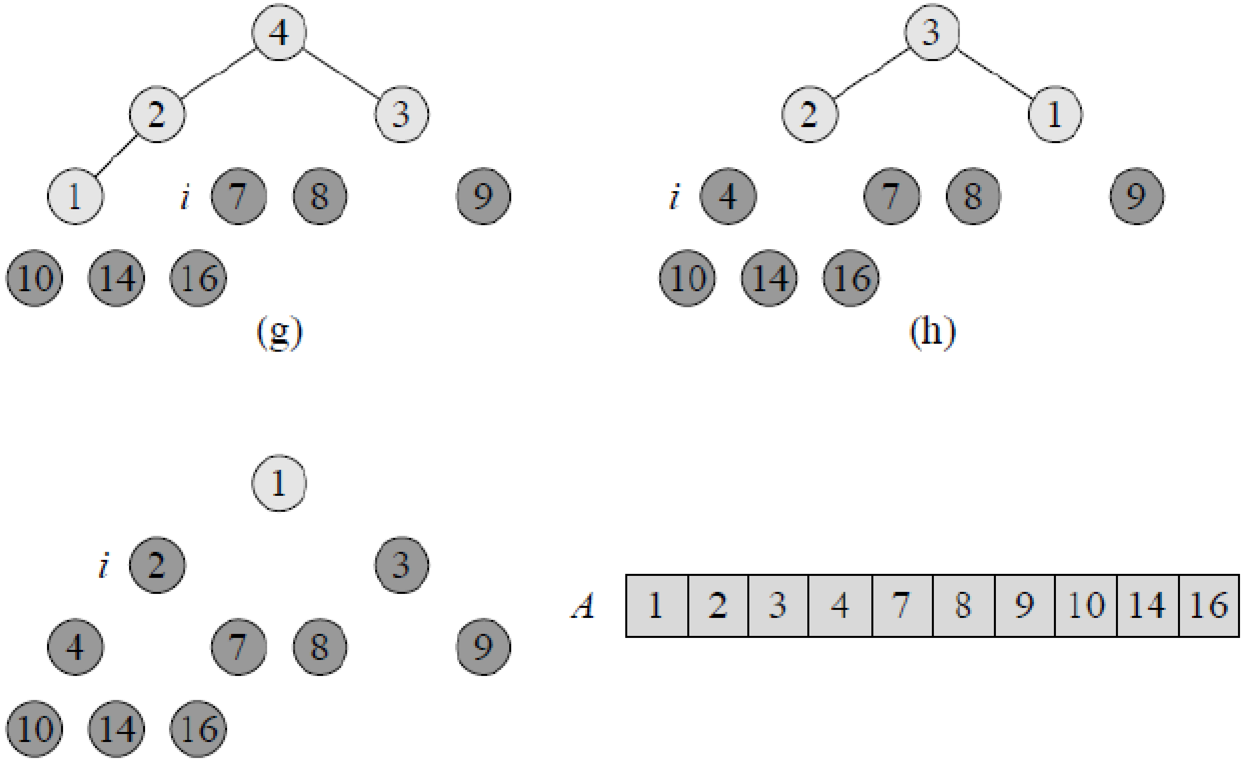
Heap Sort Algoritması



Heap Sort Algoritması



Heap Sort Algoritması



(g) (h) (i) (j) (k)

Heap Sort Algoritması

HEAPSORT(A)	<u>**ANALİZ**</u>
BUILD-MAX-HEAP(A)	**O(n)**
for $i = A.length$ **downto** 2	**n-1**
exchange $A[1]$ with $A[i]$	**O(1)**
$A.heap\text{-}size = A.heap\text{-}size - 1$	**O(1)**
MAX-HEAPIFY(A, 1)	**O(lg n)**

- ***T(n) = O(n) + (n -1) O(lgn)***
- ***T(n) = O(n) + O(n lgn)***
- ***T(n) = O(n lgn)***

Heap Sort Algoritması

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 Algoritması

- **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.

UYGULAMA SORULARI

1. Hızlı sıralama algoritmasını uygulayınız.
2. Rastgele hızlı sıralama algoritmasını uygulayınız.
3. Bu iki algoritmayı performans açısından karşılaştırınız.
4. Heap Sort algoritmasını uygulayınız.
5. Rastgele hızlı sıralama algoritması ile Heaps Sort algoritmasını performans açısından karşılaştırınız.

KAYNAKÇA

- ❖Algoritmalar : Prof. Dr. Vasif NABİYEV, Seçkin Yayıncılık
- ❖Algoritmalara Giriş : Thomas H. Cormen , Charles E. Leiserson , Ronald L. Rivest and
- ❖Clifford Stein , Palme YAYINCILIK
- ❖Algoritmalar : Robert Sedgewick , Kevin Wayne , Nobel Akademik Yayıncılık
- ❖M.Ali Akcayol , Gazi Üniversitesi, Algoritma Analizi Ders Notları
- ❖Doç. Dr. Erkan TANYILDIZI, Fırat Üniversitesi, Algoritma Analizi Ders Notları
- ❖http://www.bilgisayarkavramlari.com