

VERİ YAPILARI VE ALGORİTMALAR

BLM2512 Gr.2

2020-2021 Bahar Yarıyılı (Uzaktan Eğitim)

Dr.Öğr.Üyesi Göksel Biricik

HEAP

(Binary) Heap

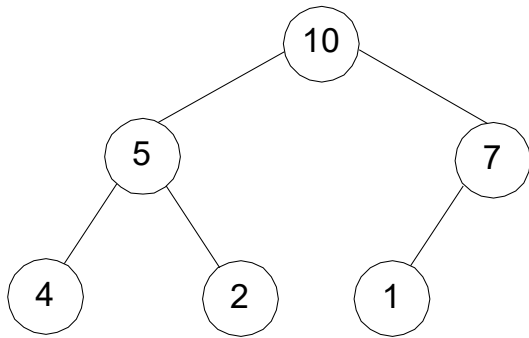
- Neredeyse “complete binary tree” şeklinde olan veri yapılarıdır.
 - Complete Binary Tree: Son hariç tüm düzeylerin (level) tam dolu olduğu ve son düzeyin de en soldan dolduğu ikili ağaçlar
- Java, Lisp gibi dillerde olan “garbage-collected” depolama alanı değildir.
- Örneğin, Heap ile $O(n \lg n)$ karmaşıklıkta (en kötü durumda) sıralama yapabiliriz.

(Binary) Heap

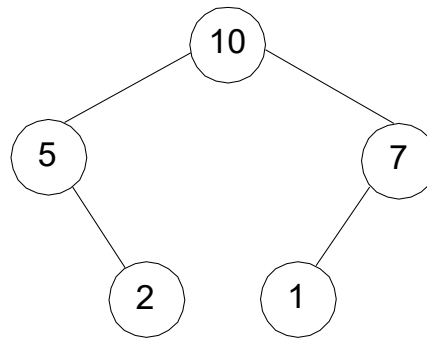
- Heap tanımı: Bir düğümündeki değerin çocuklarından daha küçük (ya da daha büyük) olmasını garanti eden ağaç temelli bir veri yapısı.
 - Heap'in herhangi bir düğümünü kök olarak alan alt ağaç da heap'tir.
- İki çeşit heap vardır, ikisi de heap tanımına uygundur
- **Max-heap**
 $A[\text{parent}(i)] \geq A[i]$
En büyük eleman köktedir.
- **Min-heap**
 $A[\text{parent}(i)] \leq A[i]$
En küçük eleman köktedir.

(Binary) Heap

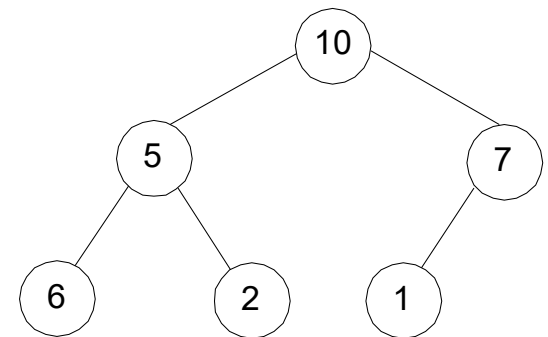
Heap



Heap Değil



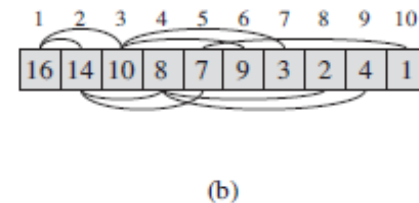
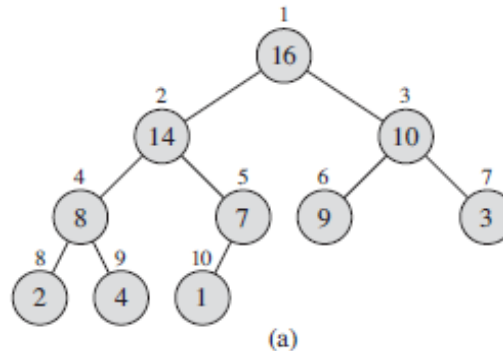
Heap Değil



(Binary) Heap

- Her düğüm, bir dizi elemanına karşılık gelir.
 - $A[1..A.length]$
 - $0 \leq A.heap_size \leq A.length$
 - Ağacın kökü $A[1]$
- i gözündeki bir elemanın ebeveyn ve çocukları:

- Node: $A[i]$
- Parent: $A[i/2]$
- Left(i): $A[2i]$
- Right(i): $A[2i+1]$



- Ebeveyn düğümler dizinin **ilk $n/2$** elemanında yer alır.
- Heap yüksekliği = Kökün yüksekliği = **$\lg n$**

En büyük (ya da küçük) elemanı bulmak


- Heap'in kök elemanı 😊


```
HEAP_MAXIMUM ( A )  
    return A [ 1 ]
```

- $O(1)$

(Binary) Heap Nasıl Oluşturabiliriz?

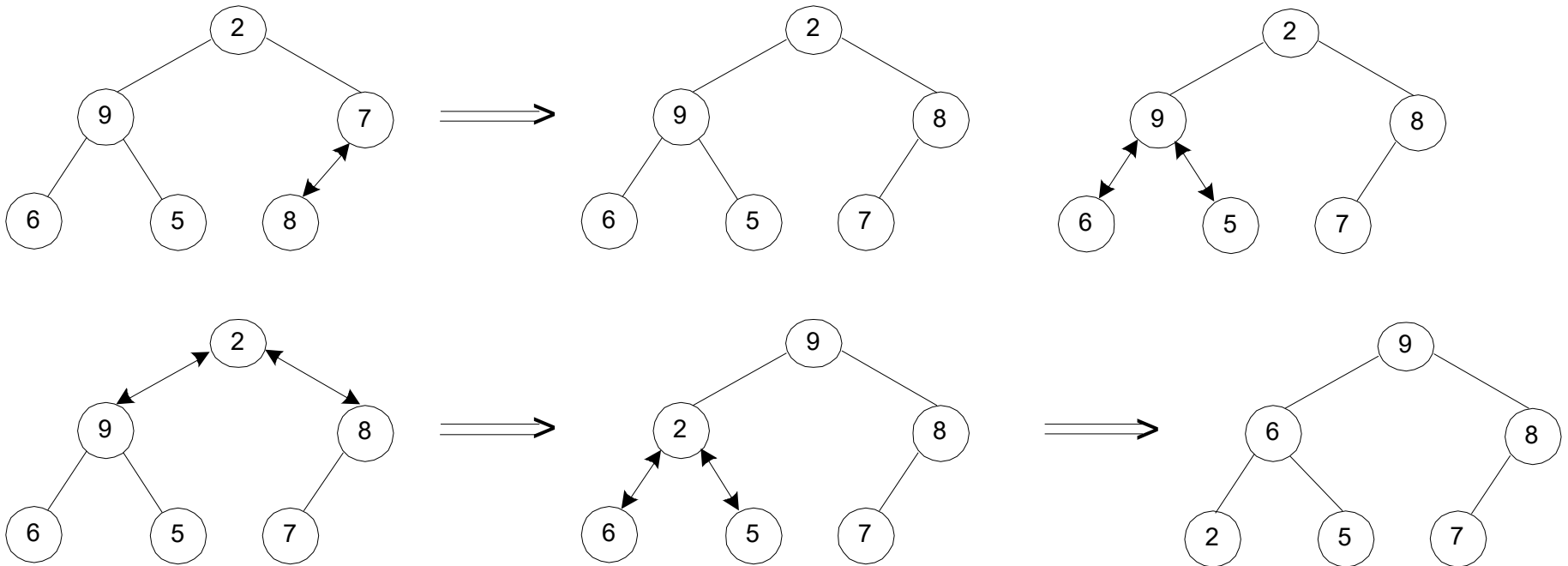
- Bottom-Up:
 - Adım 0: Verilen sıra ile veri yapısını(diziyi) oluşturun.
 - Adım 1: En son (en sağdaki) ebeveyn düğümünden başlayarak, heap parçasını kontrol edip heap haline getirin (MAX_HEAPIFY)
 - Eğer heap şartını sağlamıyorsa, şartı sağlayan kadar en büyük çocuğuyla yer değiştirin.
 - Adım 2: Bir önceki ebeveyn düğüm için adım 1'i işletin.

BUILD-MAX-HEAP(A, n)  $O(N)$
for $i \leftarrow n/2$ **downto** 1
 do MAX-HEAPIFY(A, i, n)

MAX-HEAPIFY(A, i, n)  $O(\log N)$
 $l \leftarrow \text{LEFT}(i)$
 $r \leftarrow \text{RIGHT}(i)$
if $l \leq n$ and $A[l] > A[i]$ **then** $\text{largest} \leftarrow l$ **else** $\text{largest} \leftarrow i$
if $r \leq n$ and $A[r] > A[\text{largest}]$ **then** $\text{largest} \leftarrow r$
if $\text{largest} = i$ **then**
 exchange $A[i] \leftrightarrow A[\text{largest}]$
 MAX-HEAPIFY($A, \text{largest}, n$)

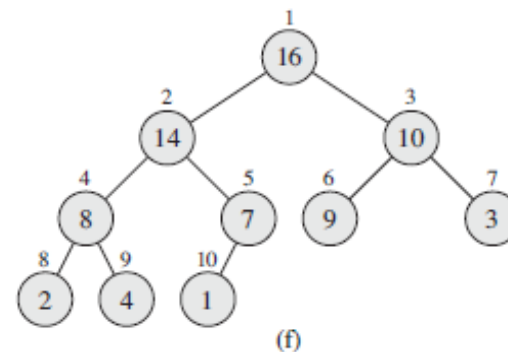
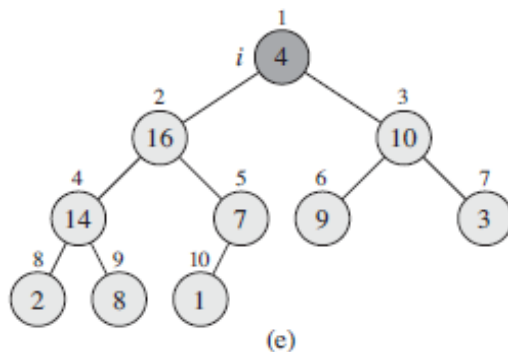
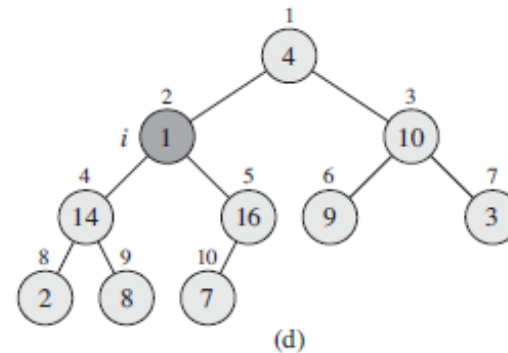
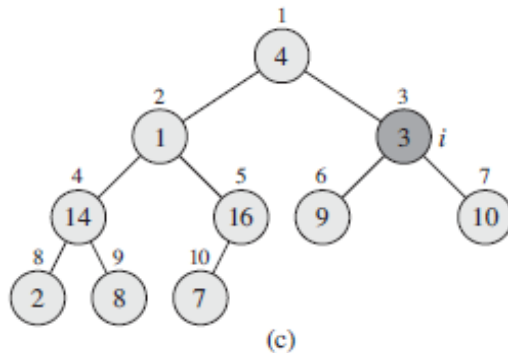
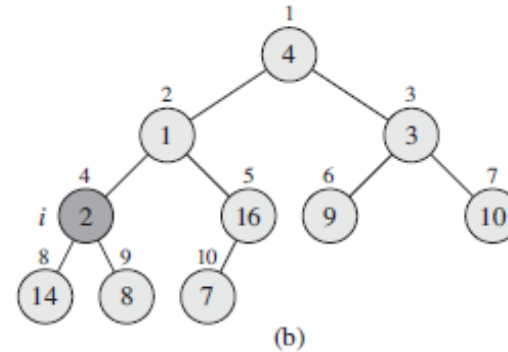
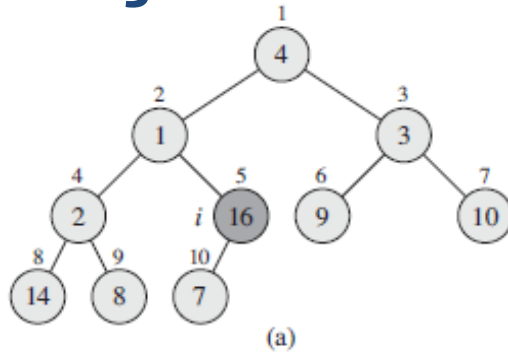
Heap Oluşturma: Örnek-1

- 2, 9, 7, 6, 5, 8



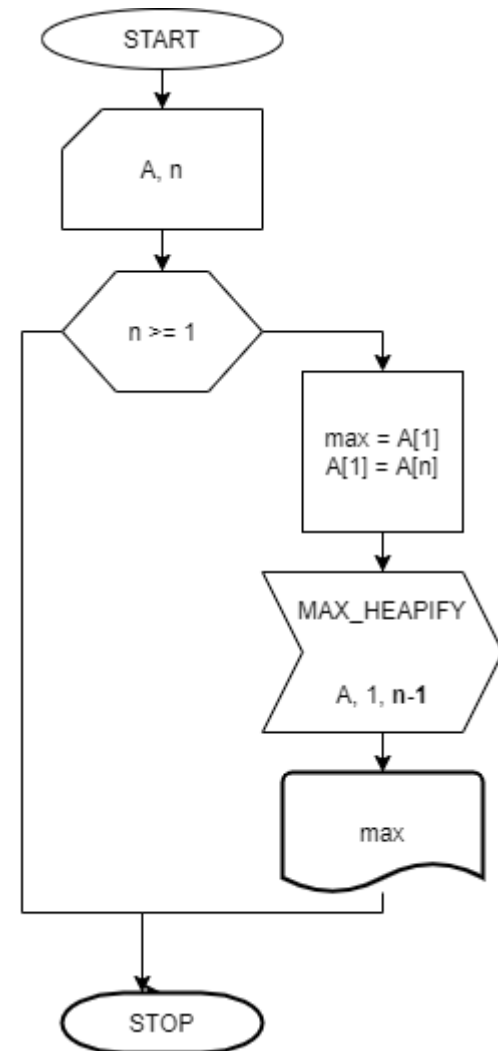
Heap Oluşturma: Örnek-2

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

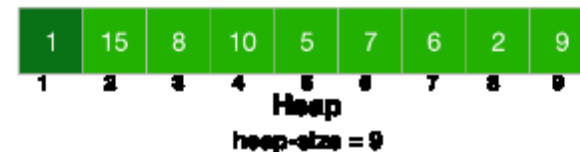
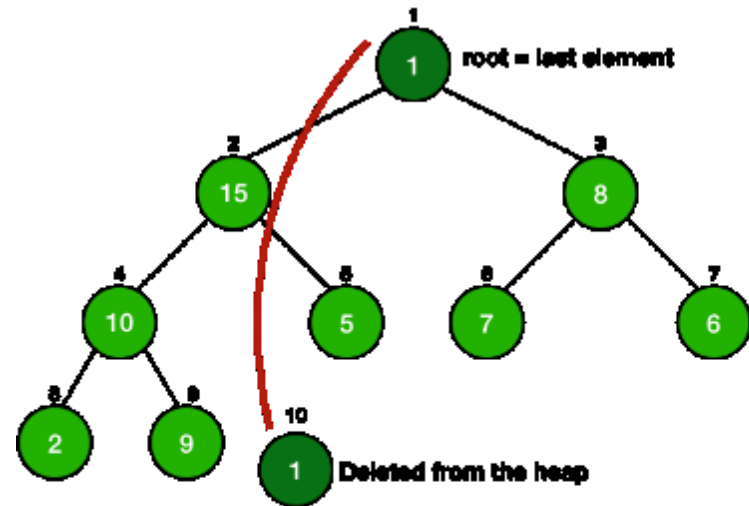
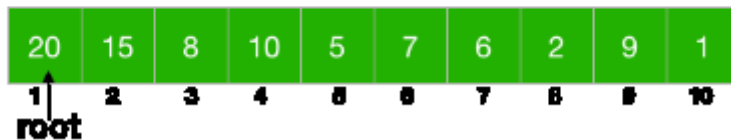
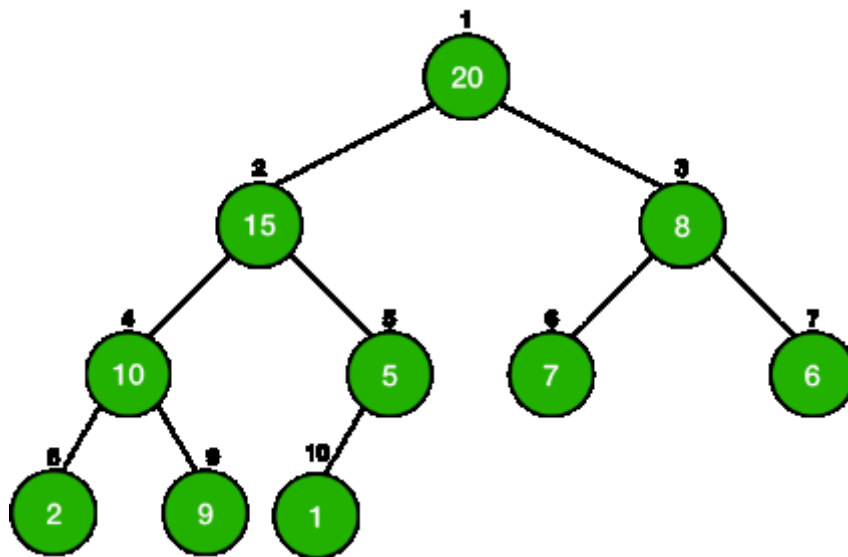


En büyük elemanı Heap'ten almak

- Heap boş olmamalı
- Kök elemanın kopyası çıkarılır (max)
- Son düğüm kök düğümü yapılır
- Bir az eleman sayısı ile «MAX_HEAPIFY» yapılır (heap boyutu 1 azaltılır)
- max değeri işlemler için çağırana döndürülür
- $O(\lg N)$

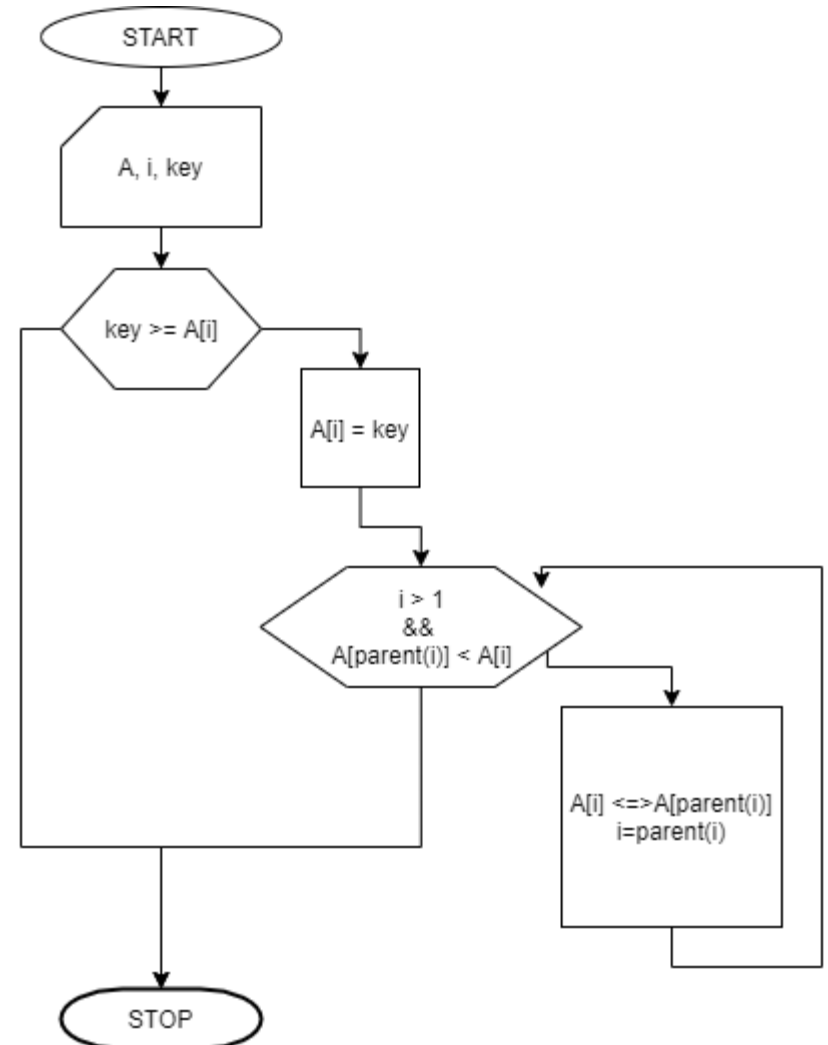


En büyük elemanı Heap'ten almak

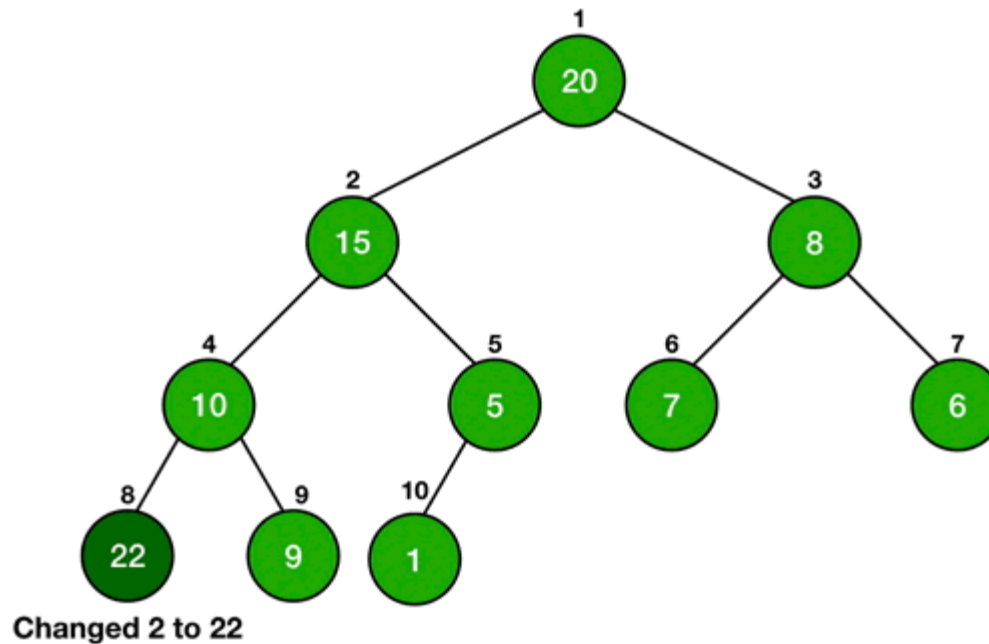


Bir düğümün değerini arttırmak

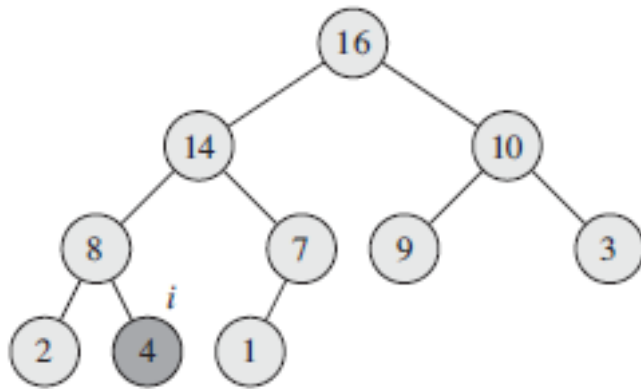
- k (yeni değer), mevcuttan büyük-eşit olmalı
- x düğümünün değeri k yapılır
- ebeveyn(ler)ile karşılaştırılarak, gerektiğinde (ebeveyninden büyük olduğu halde) yeri değiştirilir
- $O(\lg N)$



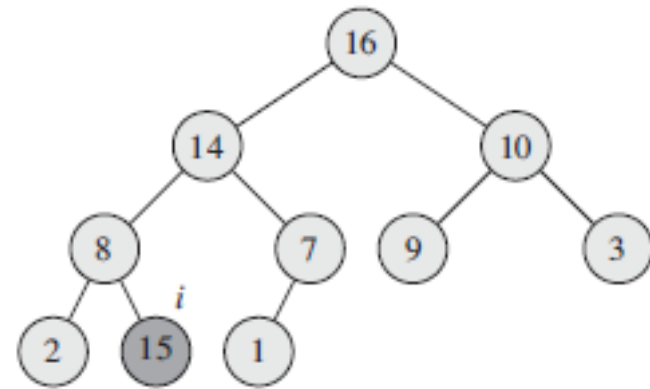
Bir düğümün değerini arttırmak



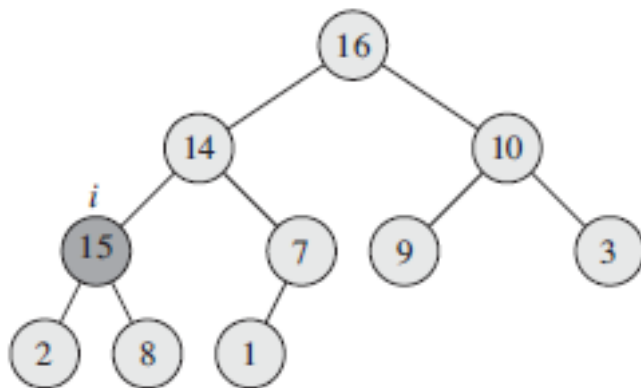
Bir düğümün değerini arttırmak



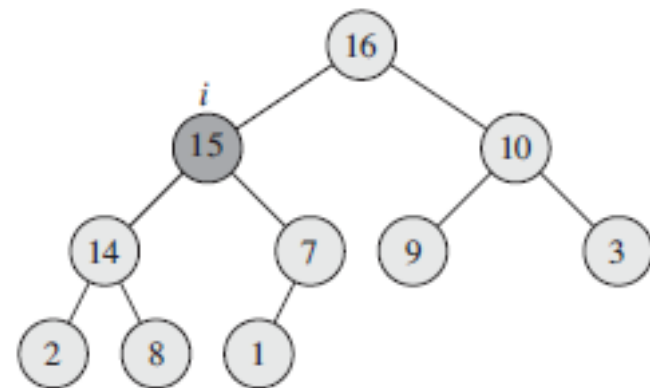
(a)



(b)



(c)



(d)

Heap'e yeni bir eleman eklemek

- Elimizde HEAP_INCREASE_KEY fonksiyonu var.
- Max_Heap'in boyu bir arttırılır, son elemanın anahtar değerine $-\infty$ atanır.
- $-\infty$ yerine «k» değeri HEAP_INCREASE_KEY ile atanır.

MAX_HEAP_INSERT(A, key, n)

$A[n+1] \leftarrow -\infty$

HEAP_INCREASE_KEY (A, n+1, key)

- $O(\lg N)$

Alternatif Heap Yaratma Yöntemi

BUILD-MAX-HEAP(A)

1 *A.heap-size = A.length*

2 **for** *i = A.length/2* **downto** 1

3 MAX-HEAPIFY(A,i)

BUILD-MAX-HEAP-2(A)

1 *A.heap-size = 1*

2 **for** *i = 2* **to** *A.length*

3 MAX-HEAP-INSERT(A,A[i])

İki yöntem, $A=1,2,3$ giriş dizisi için
aynı çıktıyı üretir mi?

Alternatif Heap Yaratma Yöntemi

Bottom-Up:

BUILD-MAX-HEAP(A)

1 $A.heap-size = A.length$

2 **for** $i = A.length/2$ **downto** 1

3 MAX-HEAPIFY(A,i)

Top-Down:

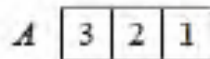
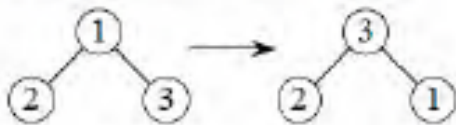
BUILD-MAX-HEAP-2(A)

1 $A.heap-size = 1$

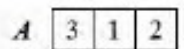
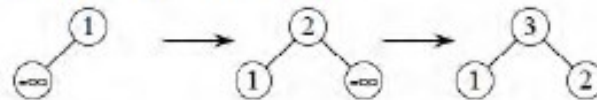
2 **for** $i = 2$ **to** $A.length$

3 MAX-HEAP-INSERT(A,A[i])

BUILD-MAX-HEAP(A):



BUILD-MAX-HEAP'(A):



HEAPSORT

HeapSort

- $A[1..n]$ giriş dizisini ilk olarak max-heap'e yerleştiririz (BUILD_MAX_HEAP)
- En büyük eleman $A[1]$, dizinin o andaki en son elemanı ile yer değiştirir.
- Son eleman en büyük olduğu için görmezden gelip (heap boyutunu 1 küçültüp) yeni kök üzerinde MAX_HEAPIFY işlemini gerçekleştiririz.
- En küçük eleman kalana kadar işleme devam ederiz.

HEAPSORT(A, n)

BUILD-MAX-HEAP(A, n)

for $i \leftarrow n$ **downto** 2 **do**

 exchange $A[1] \leftrightarrow A[i]$

 MAX-HEAPIFY($A, 1, i - 1$)

BUILD-MAX-HEAP: $O(n)$

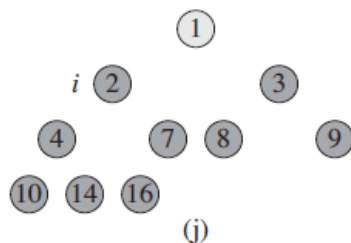
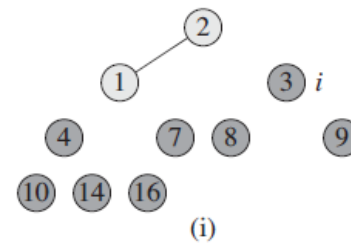
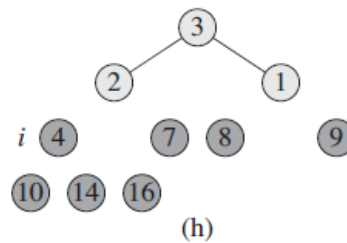
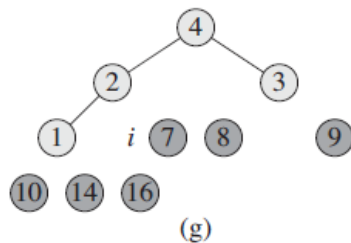
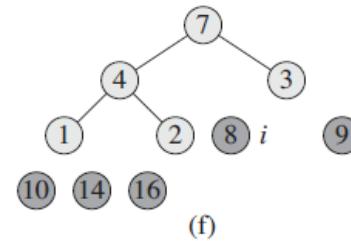
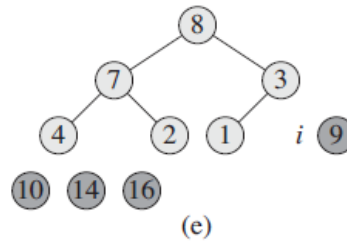
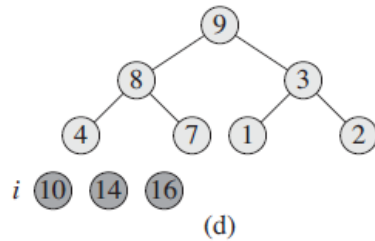
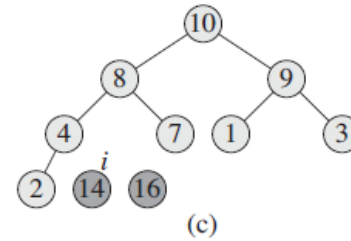
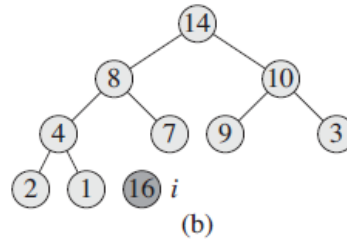
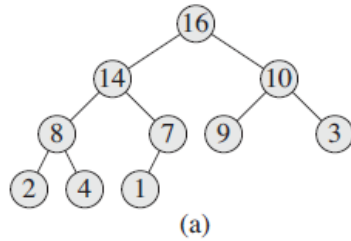
for loop: $n - 1$ tekrar

 exchange s: $O(1)$

MAX-HEAPIFY: $O(\lg n)$

Toplam Karmaşıklık: $O(n \lg n)$.

HeapSort - Örnek-1



A

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

(k)

HeapSort – Örnek-2

HeapSort 2, 9, 7, 6, 5, 8

Aşama 1 (heap oluşturma)

2	9	<u>7</u>	6	5	8
2	<u>9</u>	8	6	5	7
<u>2</u>	9	8	6	5	7
9	<u>2</u>	8	6	5	7
9	6	8	2	5	7

Aşama 2 (kök/max değiştirme)

<u>9</u>	6	8	2	5	7
7	6	8	2	5	9
<u>8</u>	6	7	2	5	9
5	6	7	2	8	9
<u>7</u>	6	5	2	8	9
2	6	5	7	8	9
<u>6</u>	2	5	7	8	9
5	2	6	7	8	9
<u>5</u>	2	6	7	8	9
2	5	6	7	8	9