5.7 Öncelik Kuyruğu

- → kuyruk gibi davranır
- → kuyruk: arkadan sıraya gir, önden çık
- → mantıksal sırası ise, önceliklerince belirlenir
- → en yüksek öncelikli kuyruk başında
- → öncelik kuyruğu özellikle _graph_ algoritmaları için yararlıdır

öncelik kuyruğu

- → gerçekleme: sıralama + liste kullanımı
- → öncelik için sıralama; elemanları tutmak için liste
- → listeye ekleme O(n), sıralama ise O(nlog n) maliyetli
- → daha iyileştirmek mümkün mü?
- → yanıtı: **ikil yığın (heap)** yapısıdır
- → bu yapıda ekleme-çıkarma O(n) maliyetli olacak

ikil yığın

- → çalışması ilginçtir
- → gösterilimi ağaca benzer, iç temsili ise tek bir listedir
- → iki varyasyonu vardır: min heap, max heap
- → en küçük anahtarlı en önde/sonda
- → biz min heap gerçekleyeceğiz
- → siz max heap gerçekleyeceksiniz

5.7.1 ikil yığın işlevleri

BinaryHeap(): boş ikil yığın

insert(k): yeni öğeyi – k yığına koy

findMin(): minimum anahtarlı öğeyi bul

delMin(): minimum anahtarlı değeri bul, sil, yığından uzaklaştır

isEmpty(): yığın boş mu?

size(): yığında kaç eleman var?

buildHeap(list): list'eden yığın oluştur

decreaseKey(k): k anahtarlı keyi bul ve değerini bir azalt

demo

bitince böyle olacak

```
>>> bh = BinaryHeap()
>>> bh.insert(5)
>>> bh.insert(7)
>>> bh.insert(3)
>>> bh.insert(11)
>>> print bh.delMin()
3
>>> print bh.delMin()
5
>>> print bh.delMin()
7
>>> print bh.delMin()
11
```

5.7.2 ikil yığın gerçekleme

todo

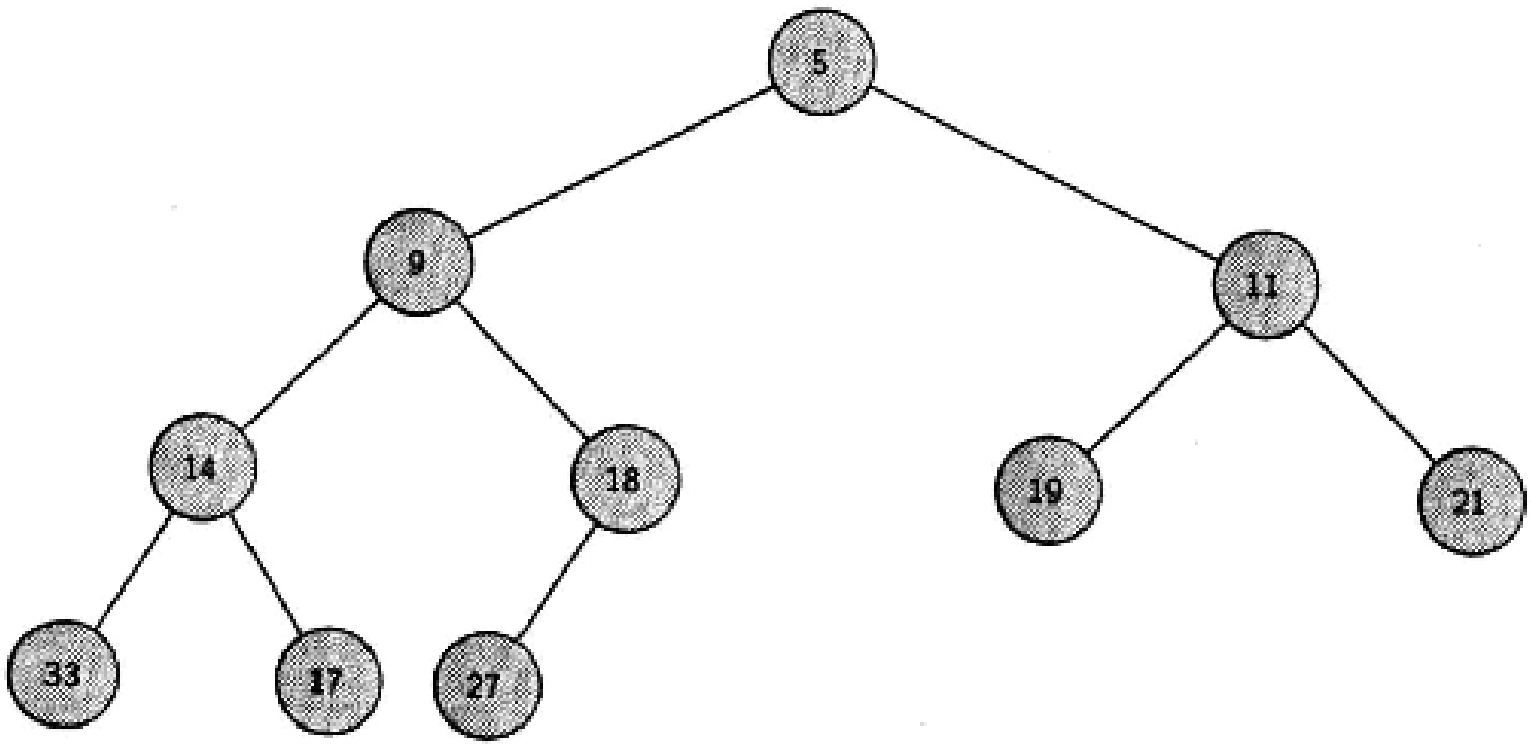
5.7.2.1 yapısal özellik

→ dengeli ağaçtır

→ tam bir ikil (complete binary tree) ağaç oluşturacağız

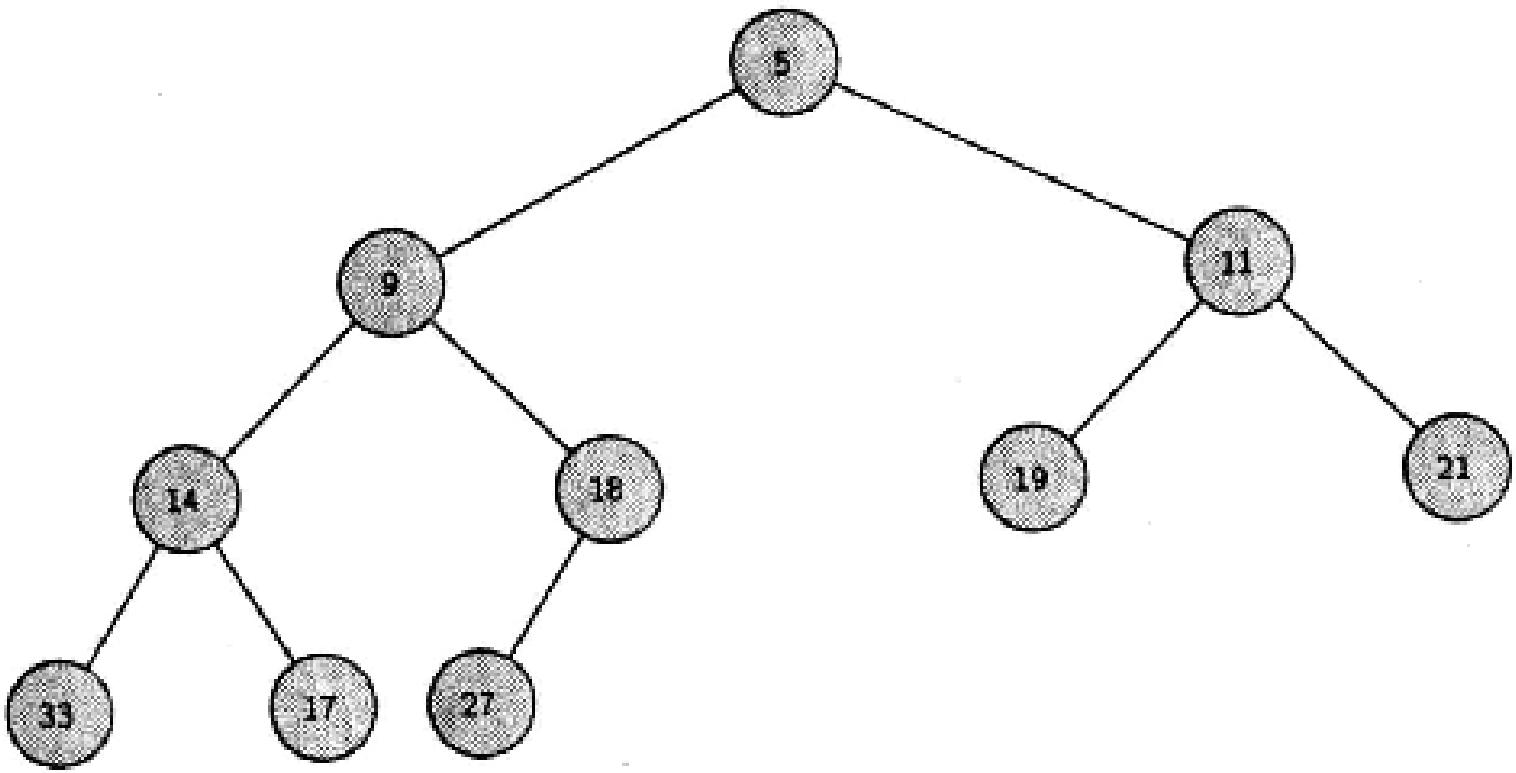
→ her bir seviyesindeki tüm düğümlere sahiptir

→ en alttakinin olmayabilir

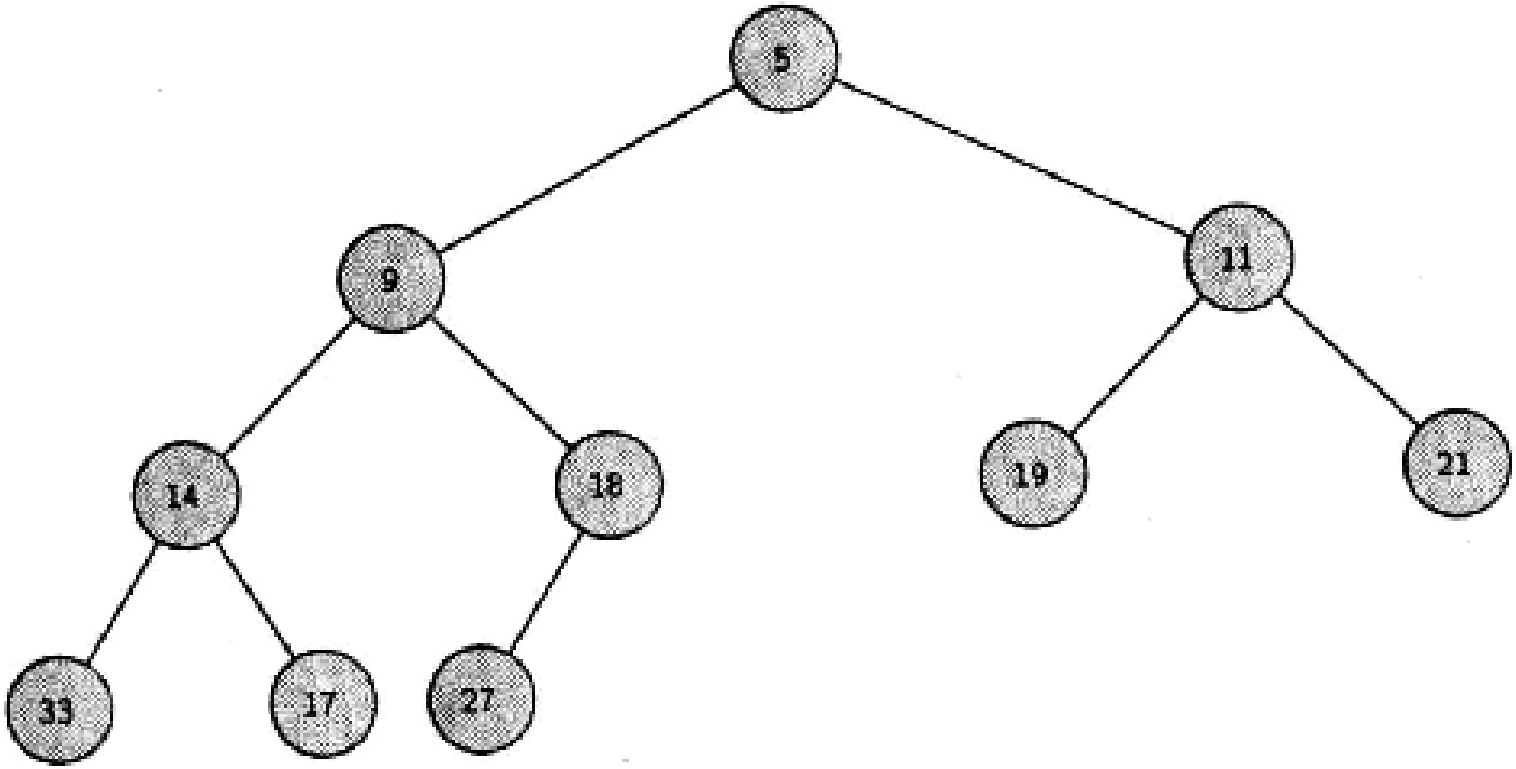


özellik 2: tek listeyle temsil

- → tek listeyle temsil
- → ağaç tam olduğundan (complete), p pozisyonundaki sol çocuk listenin 2*p
- → sağ çocuk ise 2*p+1 konumundadır
- → düğümün ebeveynini bulmak için tamsayı bölme yeterlidir
- → listenin n konumundaki çocuğun ebeveyni n/2 konumundadır



örnek



0	5	9	11	14	18	19	21	33	17	27	
0	1	2	3	4	5	6	7	8	9	10	11

→ 14 değerli düğümün çocukları ve ebeveyni listenin hangi düğümündedir?

özet

→ dolaşması **kolay**

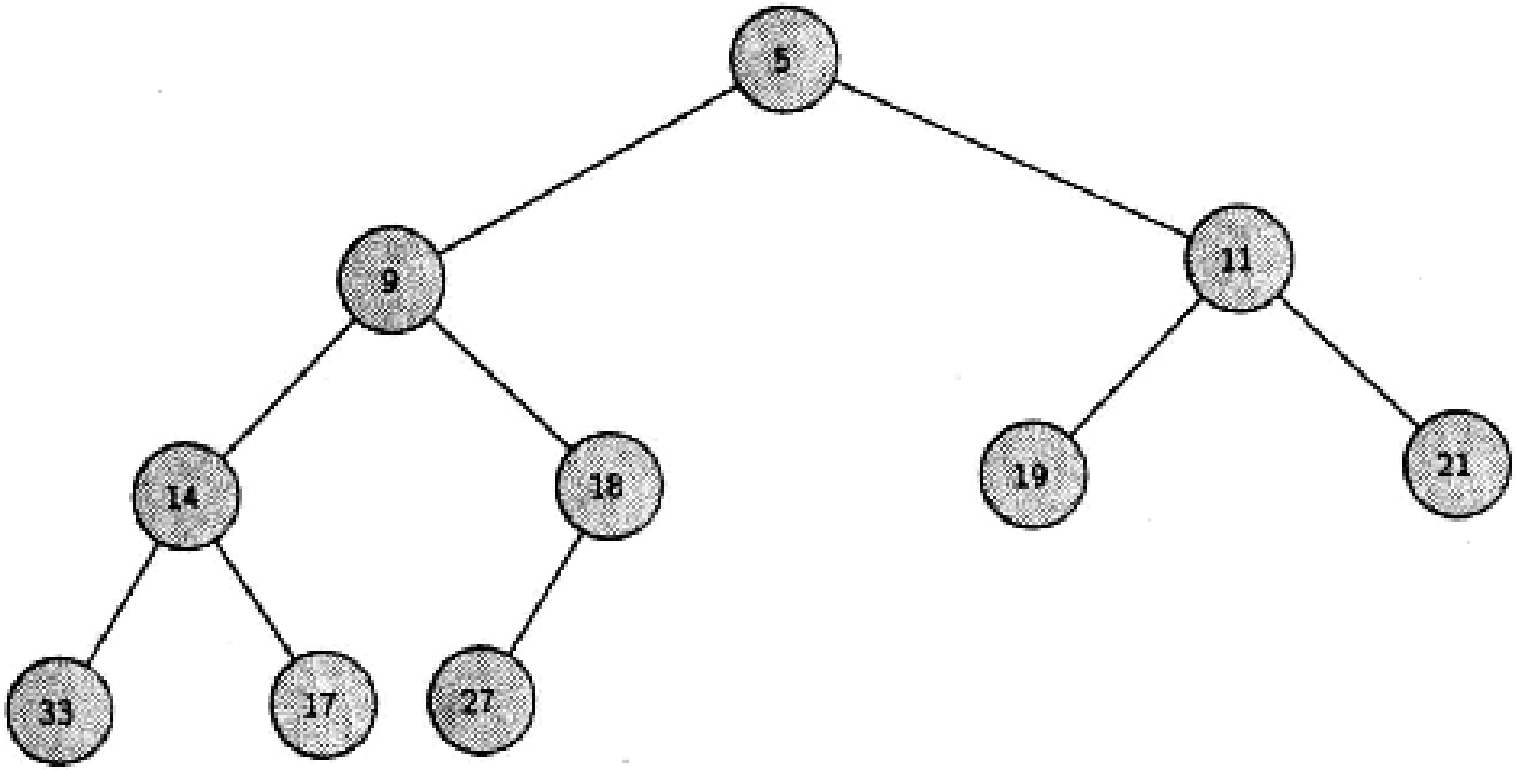
→ oluşturması **sorun**

5.7.2.2 yığın sıra özelliği

→ **yığına öğeleri saklama (sıralı tutma) işlemlerin belkemiği**

→ yığının sıralılık özelliği

→ p ebeveynli her bir x düğümü için, p'deki anahtar x'dekinden küçük veya eşittir



0	5	9	11	14	18	19	21	33	17	27	
	1	2	3	4	5	6	7	8	9	10	11

5.7.2.3 yığın işlevleri

→ yapıcı işlevi

```
1         def __init__(self):
2             self.heapList = [0]
3             self.currentSize = 0
```

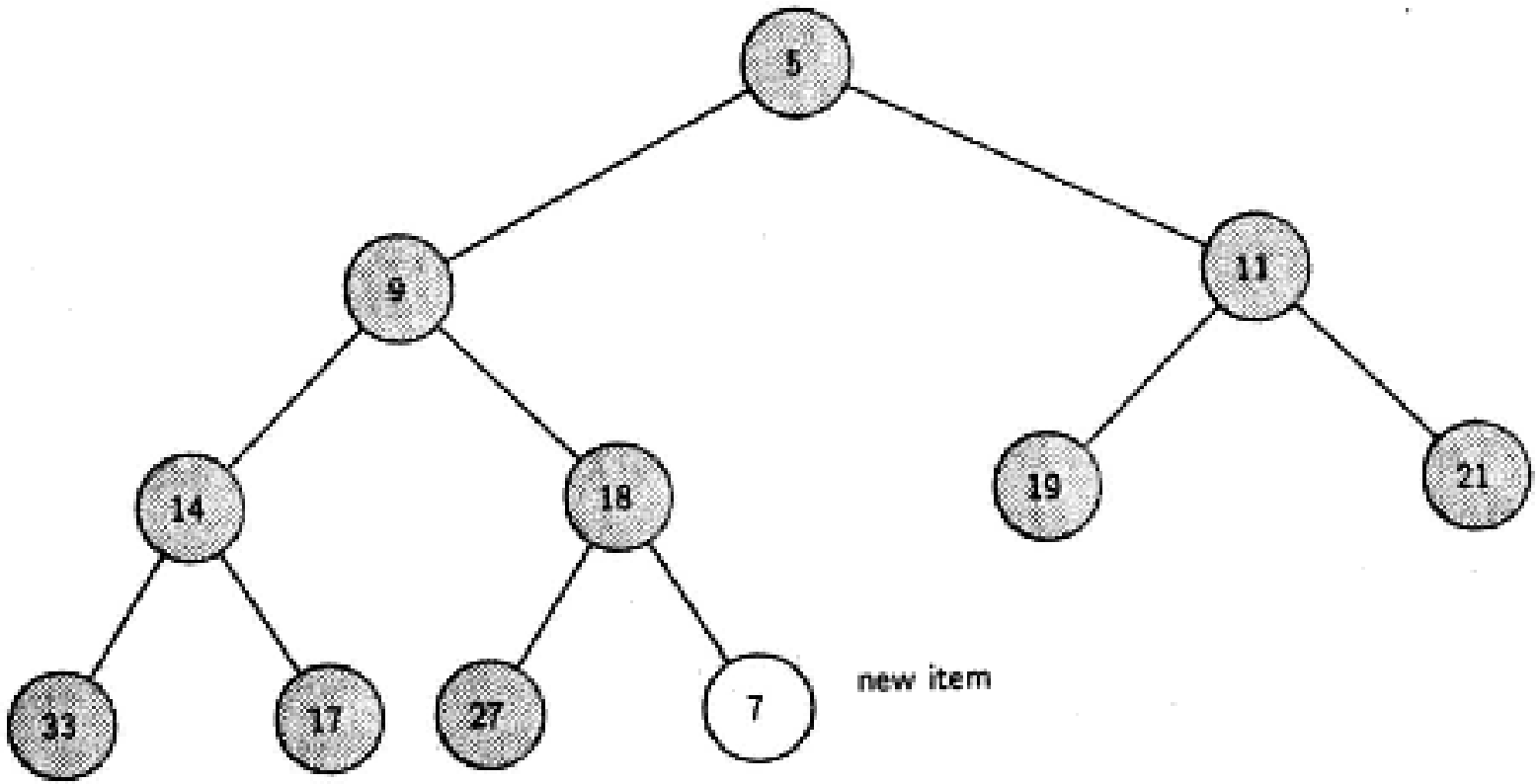
→ s2: heapList = [0]

→ başlangıç kökü

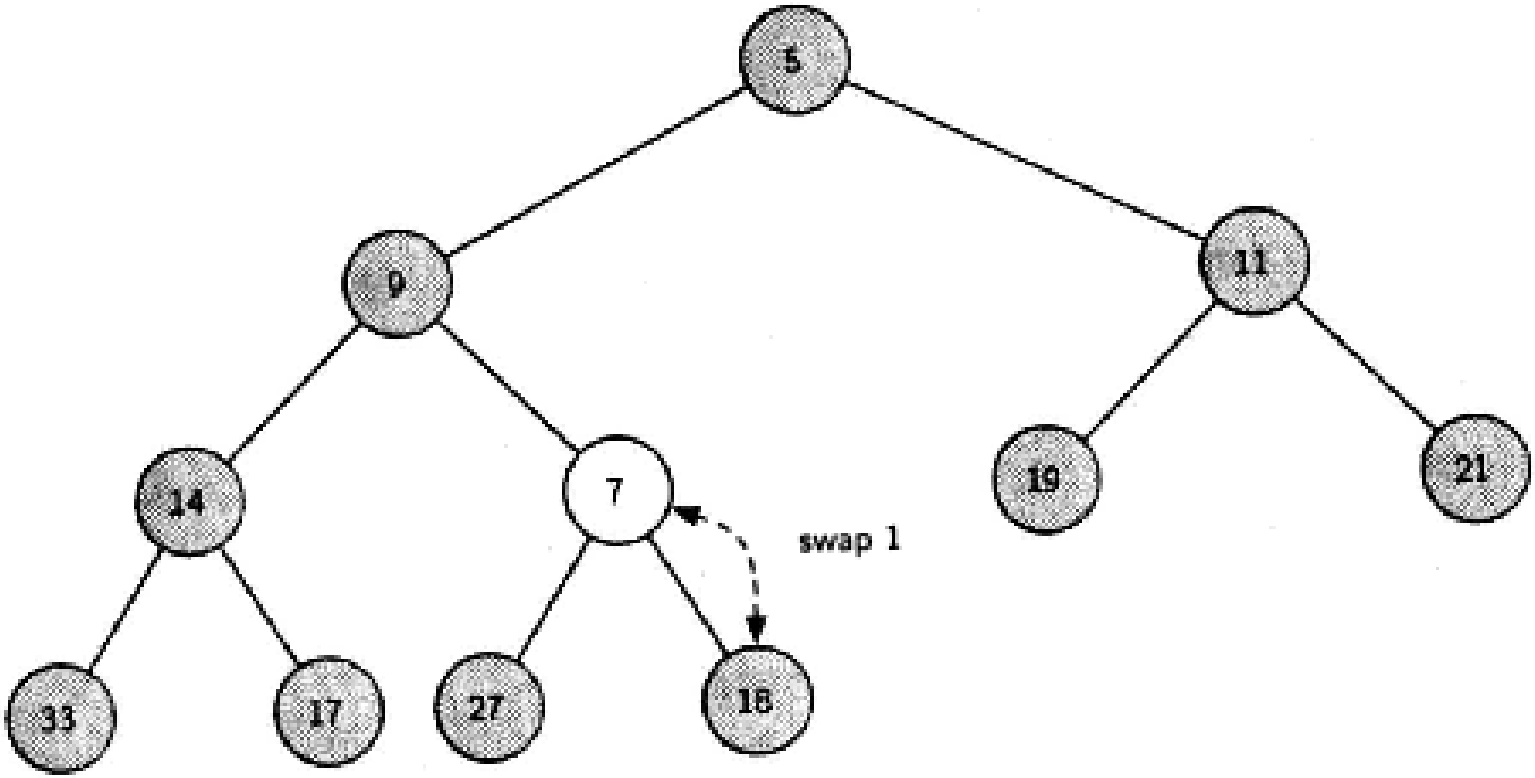
insert

→ yeni elemanı listenin sonuna ekle, tam ağaç özelliği korunur

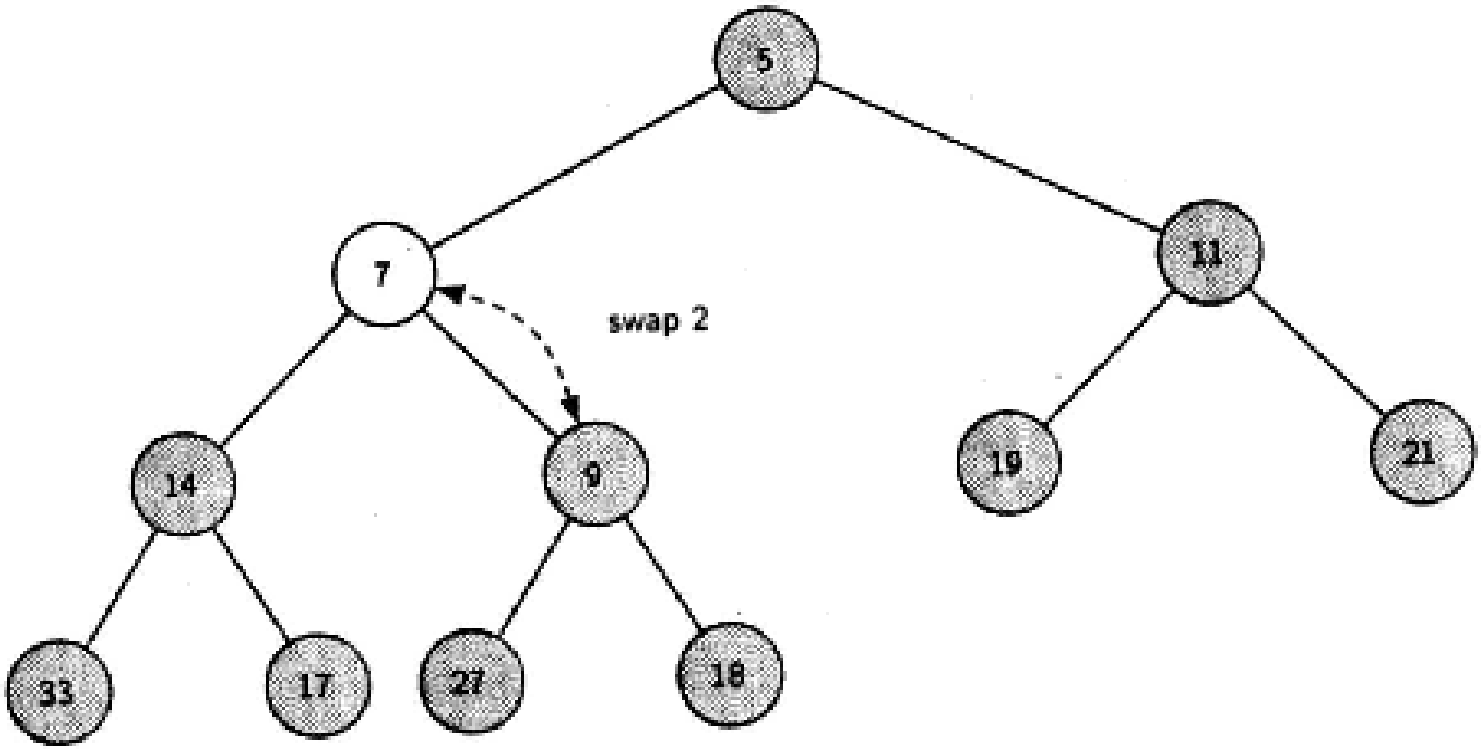
→ sıra özelliği ise (yapısallık) bozulur



insert



insert



insert: percUp (yukarıya süzül)

- → yeni ekleneni doğru yerine koymak gerek
- → bunun için yeni ekleneni ebeveyniyle karşılaştır
- → eğer küçükse takas et

gerçekleme

gerçekleme

```
1           def percUp(self,i):
2               while i > 0:
3                   if self.heapList[i] < self.heapList[i/2]:
4                       tmp = self.heapList[i/2]
5                       self.heapList[i/2] = self.heapList[i]
6                       self.heapList[i] = tmp
7                   i = i/2
```

→ s4-6: takas işlevi

→ ör. 7 eklendiğinde i=11, i/2==>5:ebeveyn

insert

gerçekleme

```
1           def insert(self,k):
2               self.heapList.append(k)
3               self.currentSize = self.currentSize + 1
4               self.percUp(self.currentSize)
```

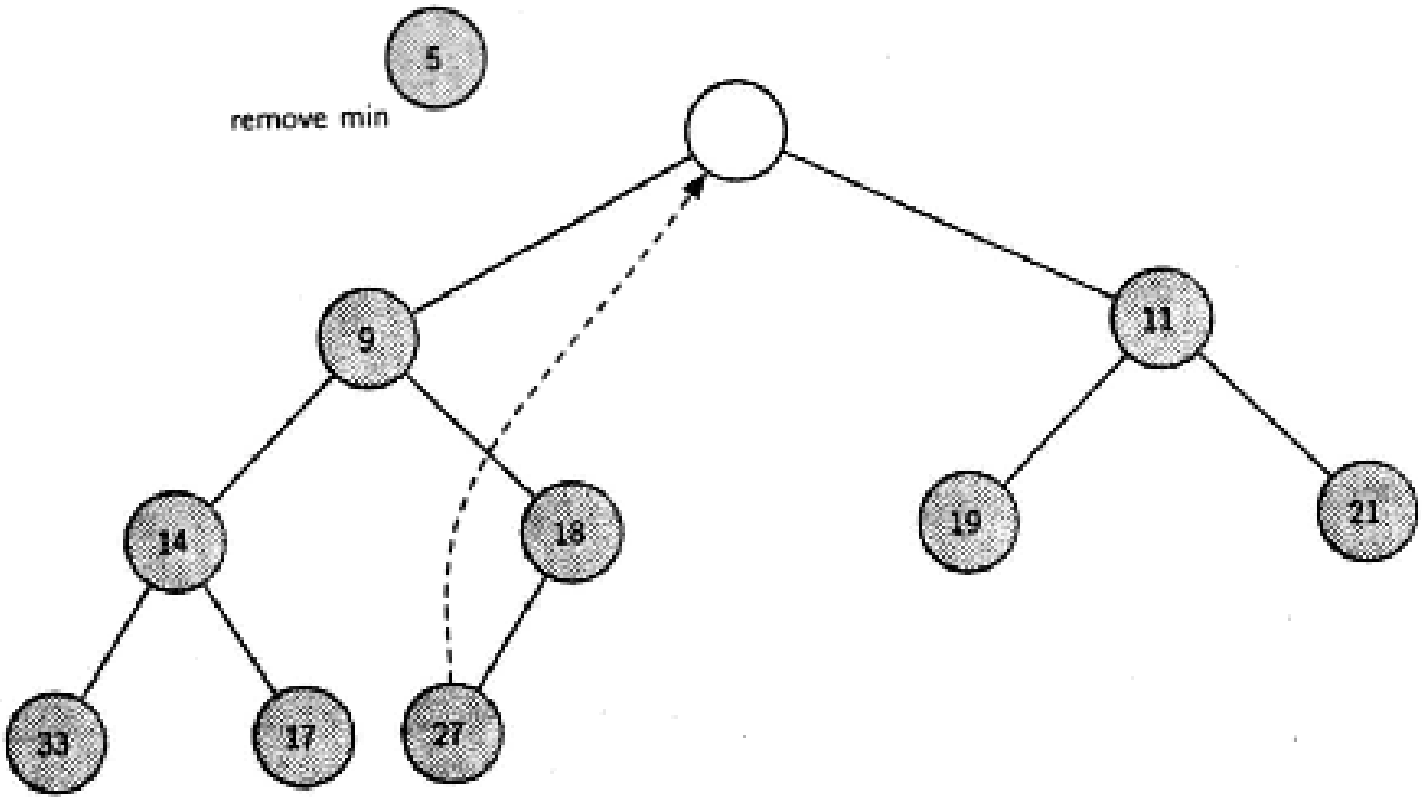
→ s2: listenin sonuna ekle

→ bu işlem ya yeni çocuk (sol çocuk) ya da ikinci çocuk (sağ çocuk) olmasını sağlayacak

→ yeni ekleneni doğru yerine kadar taşı (percUp)

→ ileride aynı öncelikliden ilk gelen üstte olacak ve silinecek

delMin



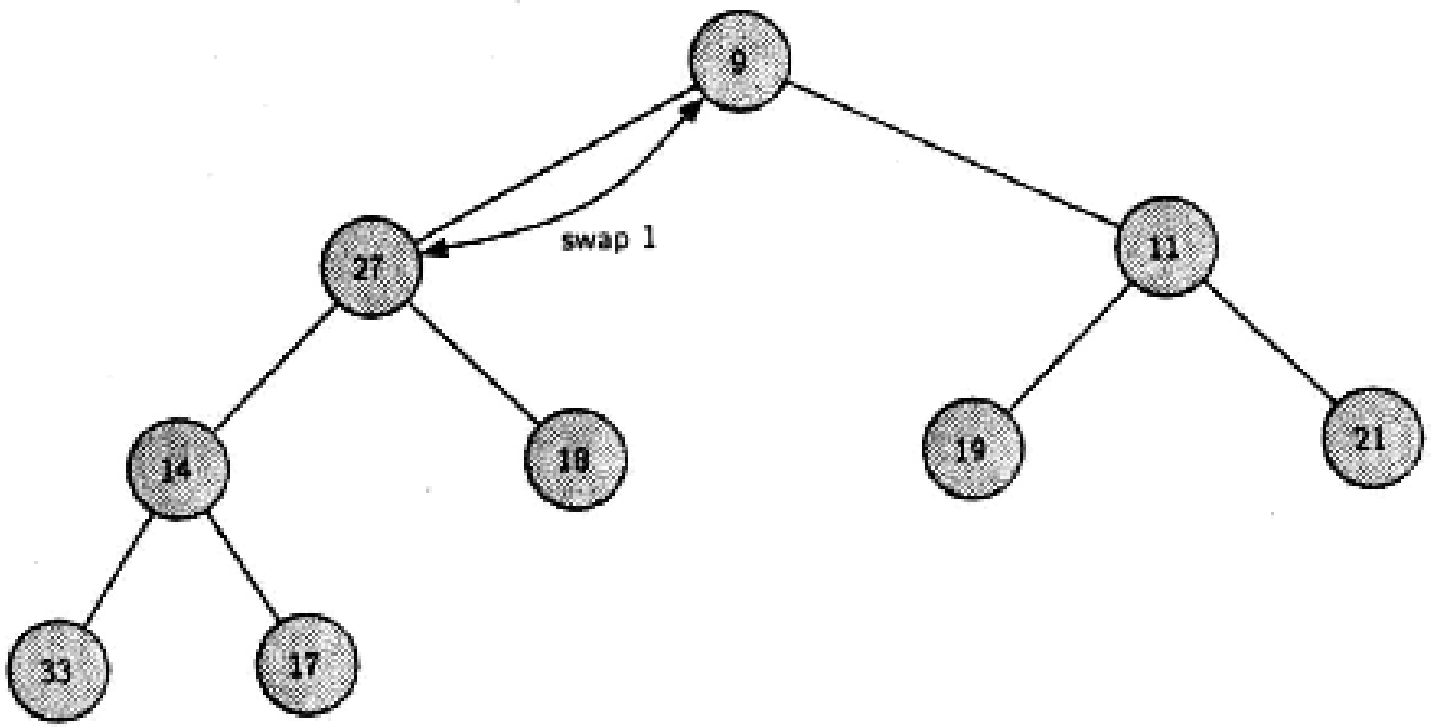
→ minimum eleman köktedir (i=1, 0 değil!)

→ onu sil, yerine listenin son elemanını koy

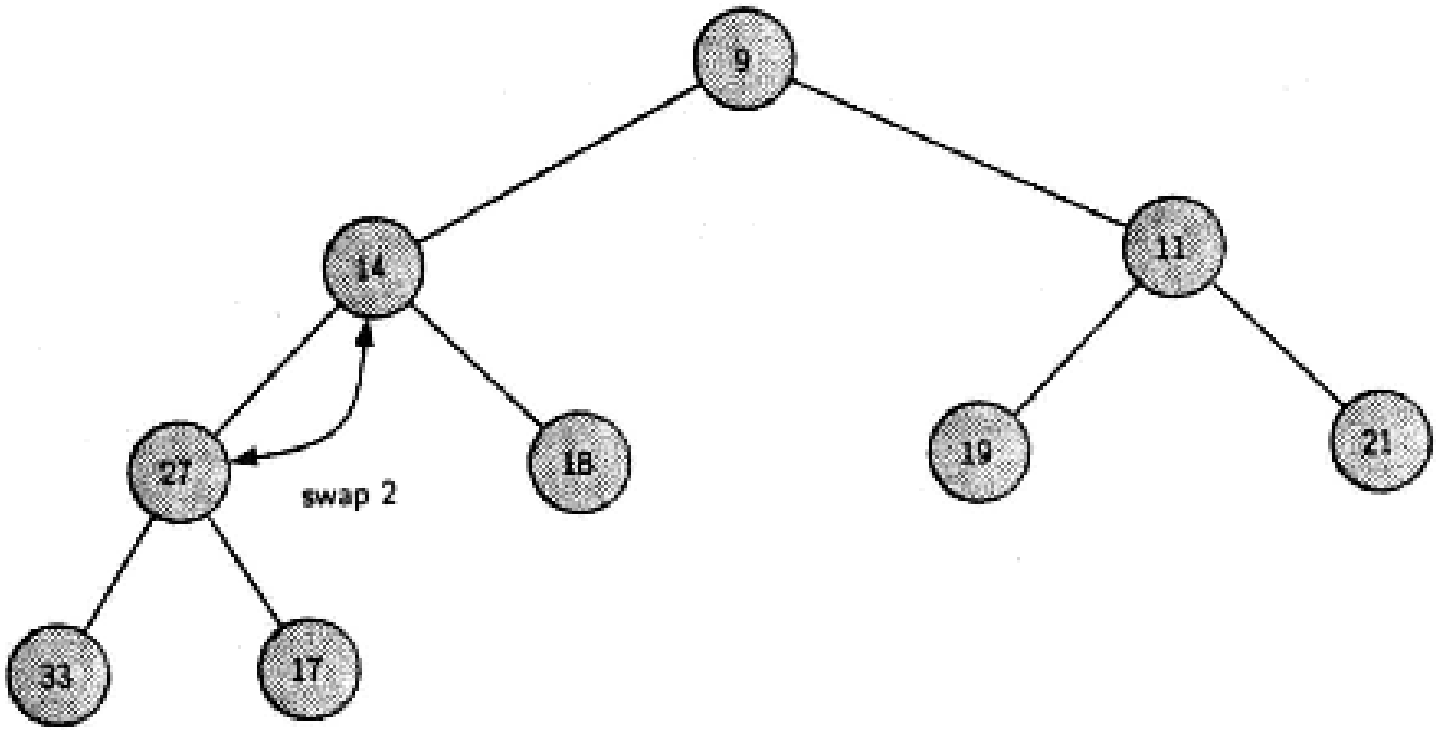
→ böylelikle tam ağaç özelliği korunur

delMin

- → yeni koyulanı çocuklarıyla karşılaştırarak takaslarla doğru yerine taşı
- → yapısallığı/sıralılığı koruma
- → aşağı doğru taşıma: percDown



delMin



gerçekleme: percDown, minChild

→ gerçekleme

```
1       def percDown(self,i):
2           while (i * 2) <= self.currentSize:
3               child = i * 2
4               mc = self.minChild(i)
5               if self.heapList[i] > self.heapList[mc]:
6                   tmp = self.heapList[i]
7                   self.heapList[i] = self.heapList[mc]
8                   self.heapList[mc] = tmp
9               i = mc
10
11
12      def minChild(self,i):
13          if i*2 > self.currentSize:
14              return -1
15          else:
16              if i*2 + 1 > self.currentSize:
17                  return i*2
18              else:
19                  if self.heapList[i*2] < self.heapList[i*2+1]:
20                      return i*2
21                  else:
22                      return i*2+1
```

açıklama

- → minimum eleman köktedir (i=1, 0 değil!)
- → onu sil, yerine listenin son elemanını koy
- → böylelikle tam ağaç özelliği korunur
- → s4: yeni koyulanı çocuklarıyla karşılaştırarak
- → s6-8: takaslarla doğru yerine taşı
- → yapısallığı/sıralılığı koruma
- → aşağı doğru taşıma: percDown

gerçekleme: delMin

gerçekleme

```
1       def delMin(self):
2           retval = self.heapList[1]
3           self.heapList[1] = self.heapList[self.currentSize]
4           self.currentSize = self.currentSize - 1
5           self.percDown(1)
6           return retval
```

buildHeap

→ gerçekleme

```
1       def buildHeap(self,alist):
2           i = len(alist) / 2
3           self.currentSize = len(alist)
4           self.heapList = [0] + alist[:]
5           while (i > 0):              self.percDown(i)
6               i = i - 1
```

→ maliyeti: O(n)

→ SS: listeyi sıralayıp (O(nlogn)) o şekilde yığına itmeyi gerçekleyin.

örnek

→ Liste: [9, 5, 6, 2, 3]

```
1  i = 2,  [0, 9, 5, 6, 2, 3]
2  i = 1,  [0, 9, 2, 6, 5, 3]
3  i = 0,  [0, 2, 3, 6, 5, 9]
```

→ başlangıçta i=2 (=5/2)

→ heapList[i=2]'deki 5 değerini aşağı doğru taşı

→ ...

→ defterden çizim (sayfa 23)

sıra sizde

- alist= [9,5,6,2,3,7] ise (6 elemanlı) ikil ağaç?
- alist= [9,5,6,2,3,7, 8] ise (7 elemanlı) ikil ağaç?
- ağaçtan --> listeye?
- fig 5.12 benzeri, listeden-->yığına teker teker ekle, ağacı göster

5.8 özet

- → ifadelerin parsing ve değerlendirilmesinde ikil ağaç kullanımı
- → sözlük yapısının oluşturulmasında ikil ağaçlar
- → min heap gerçeklemesinde ikil ağaçlar
- → min heap yardımıyla öncelik kuyruğu kullanımı

ödev

BinaryHeap

→ limitli yığın boyutu

→ buildHeap, sıralama eklentisi

→ min yerine max heap

PriorityQueue

→ yapıcı, enqueue, dequeue yöntemleri