Bölüm 7. Yığın

Olcay Taner Yıldız

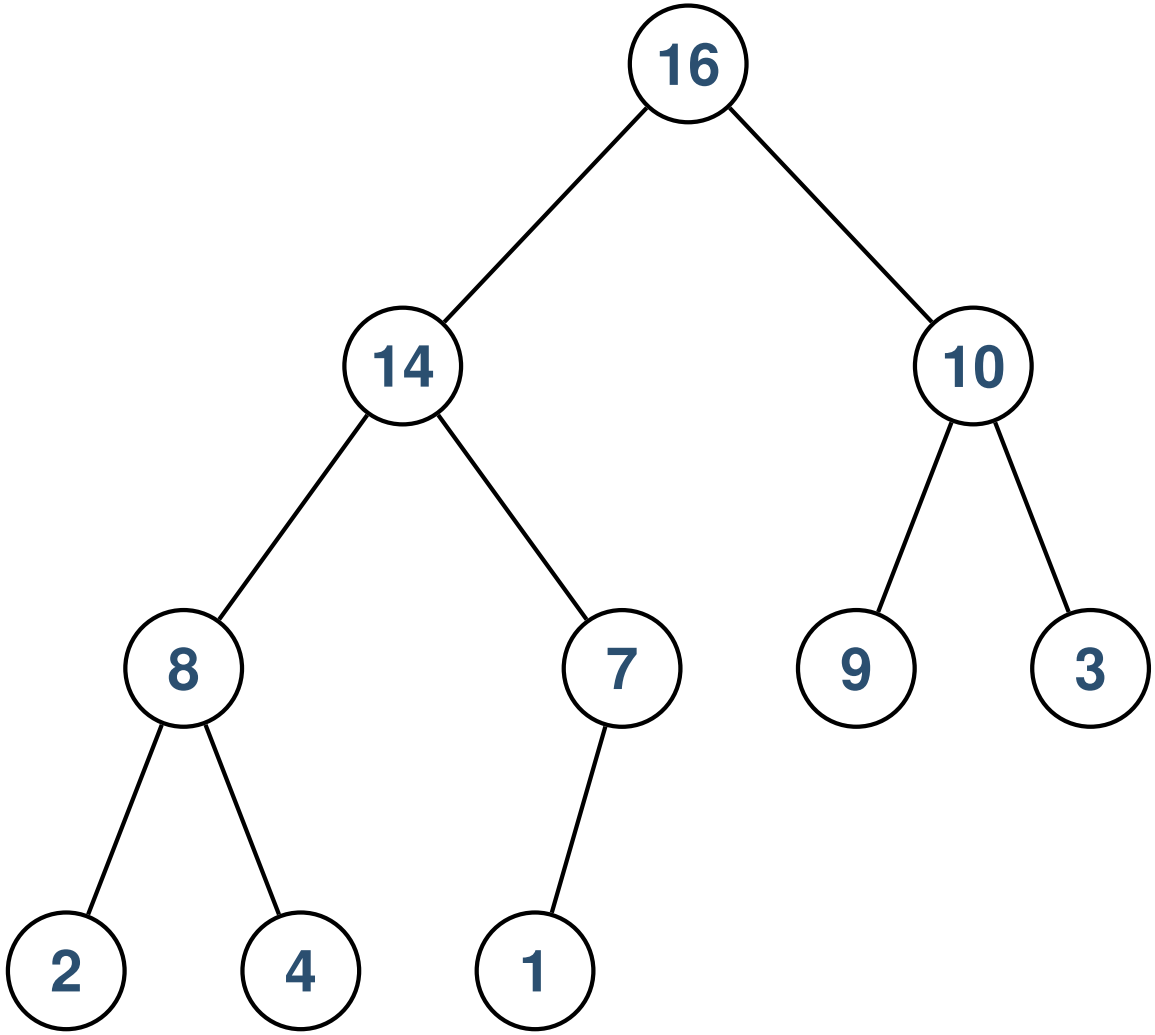
2014

Giriş

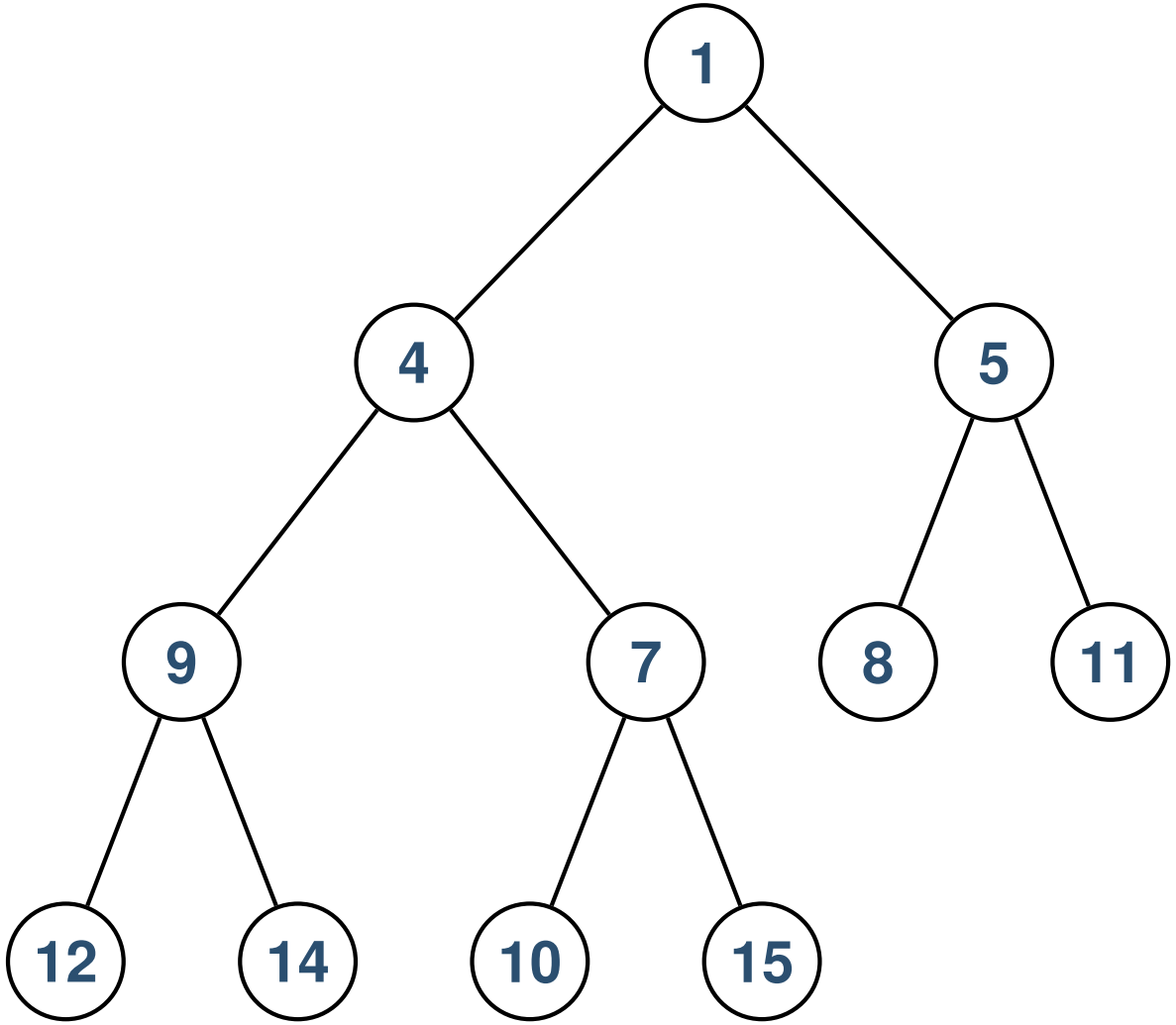
Yığın vs. İkili Arama Ağacı

- İkili arama ağacı, arama temel işlemini hızlı yapmak için üretilmiş bir veri yapısı iken yığın, bir grup elemanın arasından önceliği en yüksek olan elemanı hızlı bulmak ve bu elemanı yapıyı bozmadan hızlıca silmek için üretilmiş bir veri yapısıdır.
- İkili arama ağacında her düğüm sol alt ağacındaki tüm düğümlerden büyük, sağ alt ağacındaki tüm düğümlerden küçük iken, yığında her düğüm sol ve sağ alt ağacındaki tüm düğümlerden büyüktür. Sol ve sağ alt ağaç arasında büyüklük küçüklük ilişkisi açısından bir zorlama yoktur.
- İkili arama ağacında düğümlerin sol ve/veya sağ çocukları olabileceği gibi, hiç çocuğu da olmayabilir. Buna karşılık yığında elemanlar seviye seviye yerleştirilmiştir. Üstteki seviye dolmadan alttaki seviyede eleman bulunmaz. Dolayısıyla son seviyeden önceki seviyedeki tüm düğümlerin iki çocuğu vardır.

Örnek bir azami-yığın



Örnek bir asgari-yığın



Yığın Tanımı

Yığında tutulan elemanların tanımı


```
1  public class Nokta{
2      int icerik;
3      int ad;
4      public Nokta(int icerik, int ad){
5          this.icerik = icerik ;
6          this.ad = ad;
7      }
8  }
```

Örnek azami yığının dizi halinde gösterimi

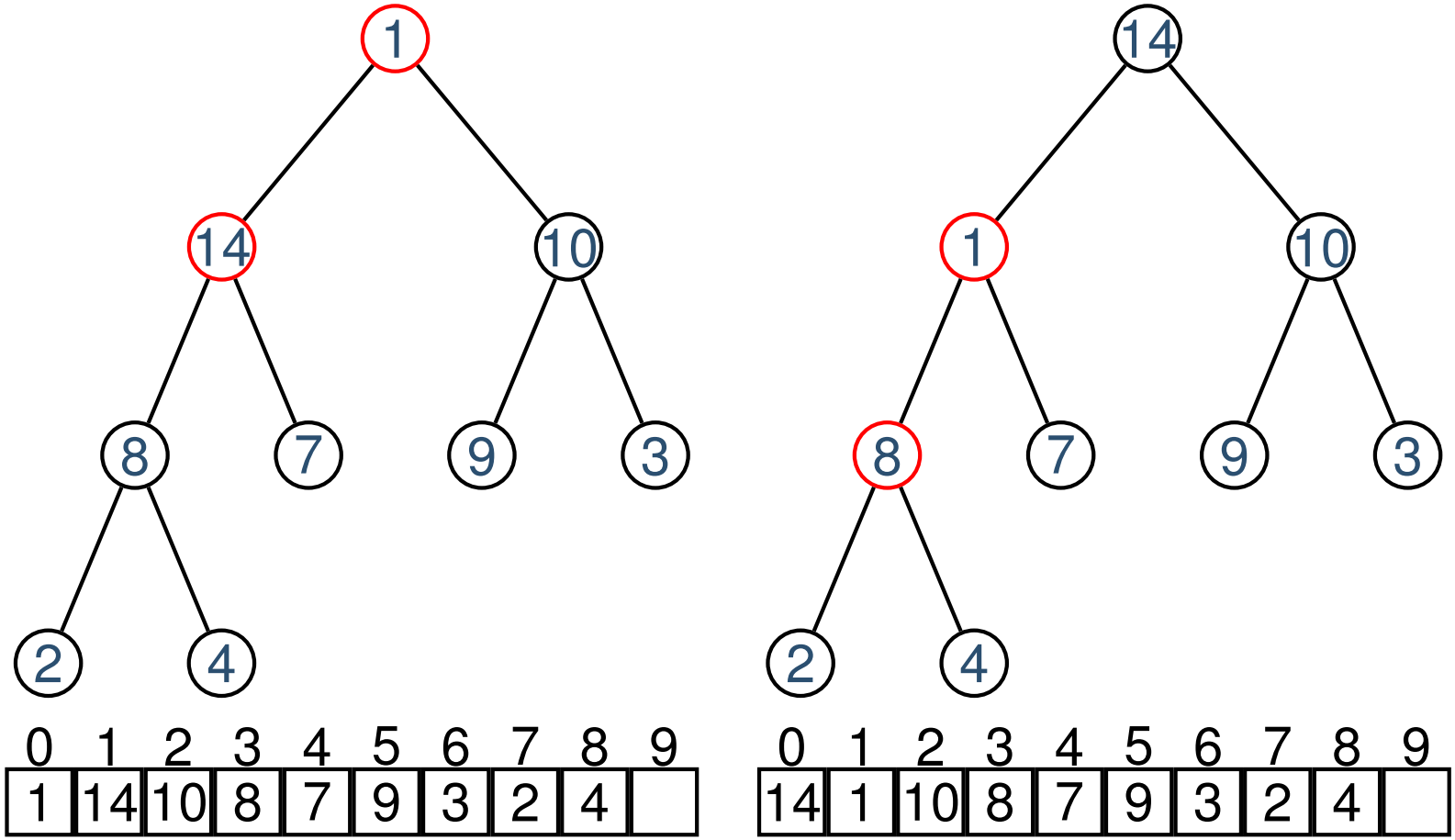
0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

Yığın tanımı

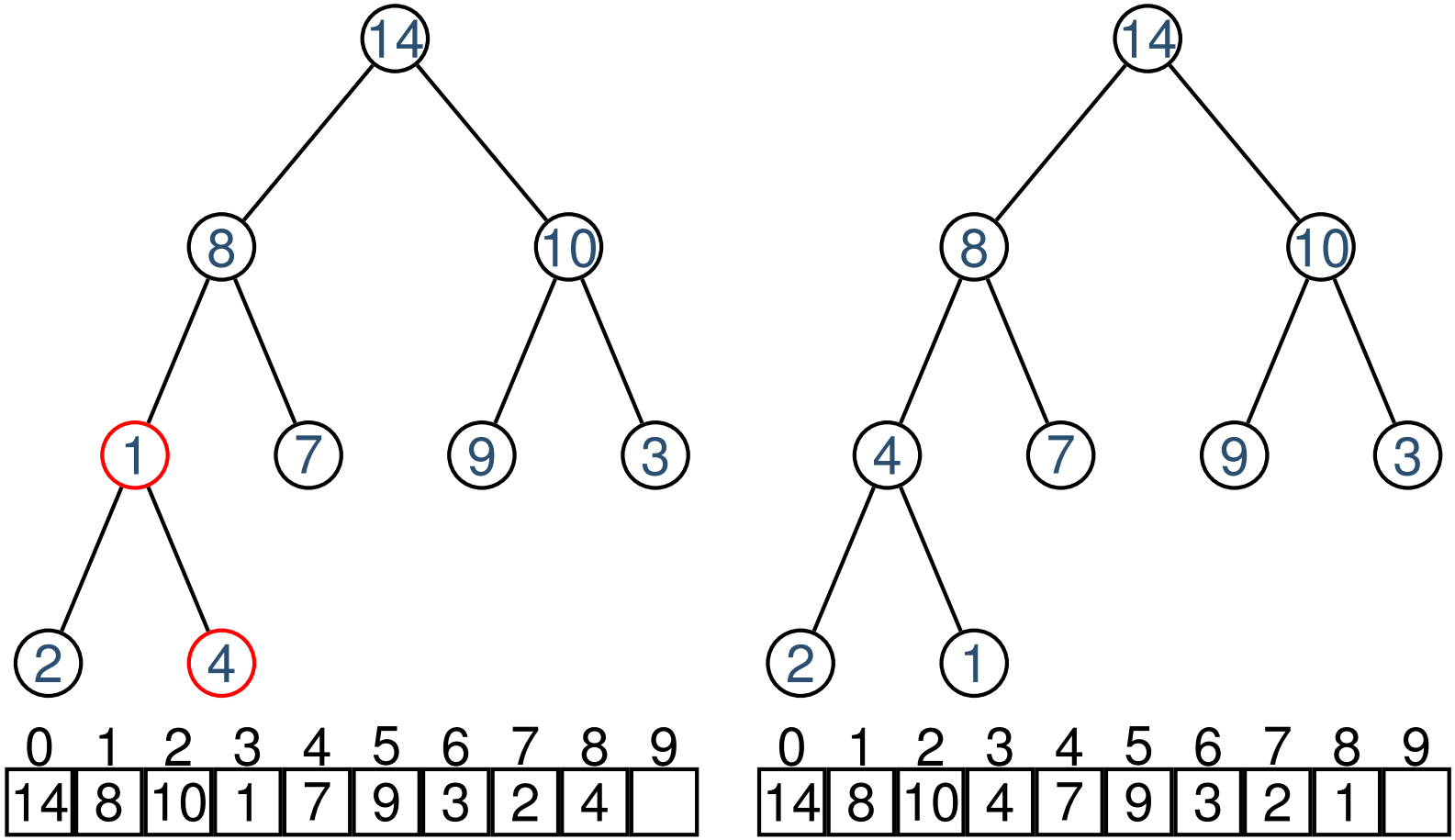

```
1  public class Yigin{
2      Nokta dizi [];
3      int tane;
4      public Yigin(int N){
5          dizi  = new Nokta[N];
6          tane = 0;
7      }
8      boolean yiginBos(){
9          if (tane == 0)
10             return true;
11         else
12             return false;
13     }
14 }
```

Temel Yığın İşlemleri

Örnek yığının ilk elemanı alındıktan sonra yapısının yeniden oluşturulması (1)



Örnek yığının ilk elemanı alındıktan sonra yapısının yeniden oluşturulması (2)



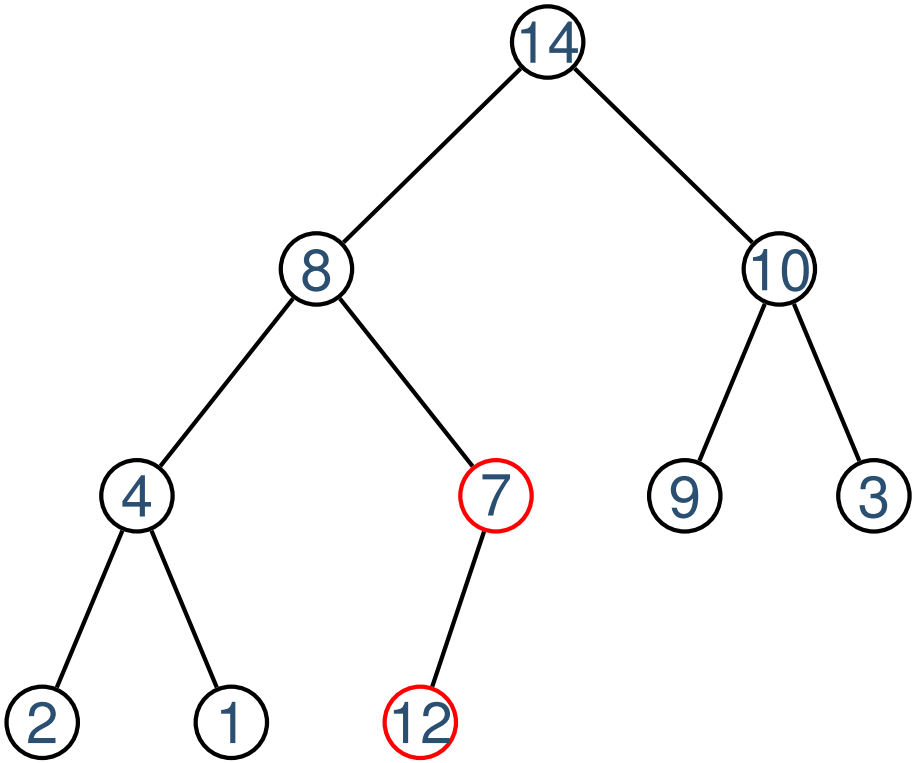
Yığın özelliğinin tekrar oluşturulması için belirli bir düğümden aşağıya doğru inme

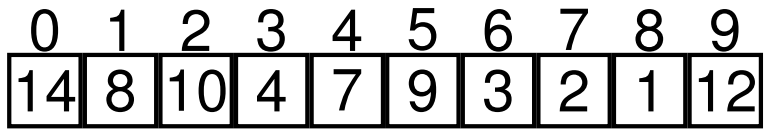

```
1  void asagiIn(int no){
2      int altsol, altsag;
3      altsol = 2 * no + 1;
4      altsag = 2 * no + 2;
5      while ((altsol < tane && dizi[no].icerik < dizi[altsol].icerik) ||
6      (altsag < tane && dizi[no].icerik < dizi[altsag].icerik )){
7          if (altsag >= tane || dizi[altsol].icerik > dizi[altsag].icerik ){
8              yerDegistir(no, altsol);
9              no = altsol;
10         }
11         else{
12             yerDegistir(no, altsag);
13             no = altsag;
14         }
15         altsol = 2 * no + 1;
16         altsag = 2 * no + 2;
17     }
18 }
```

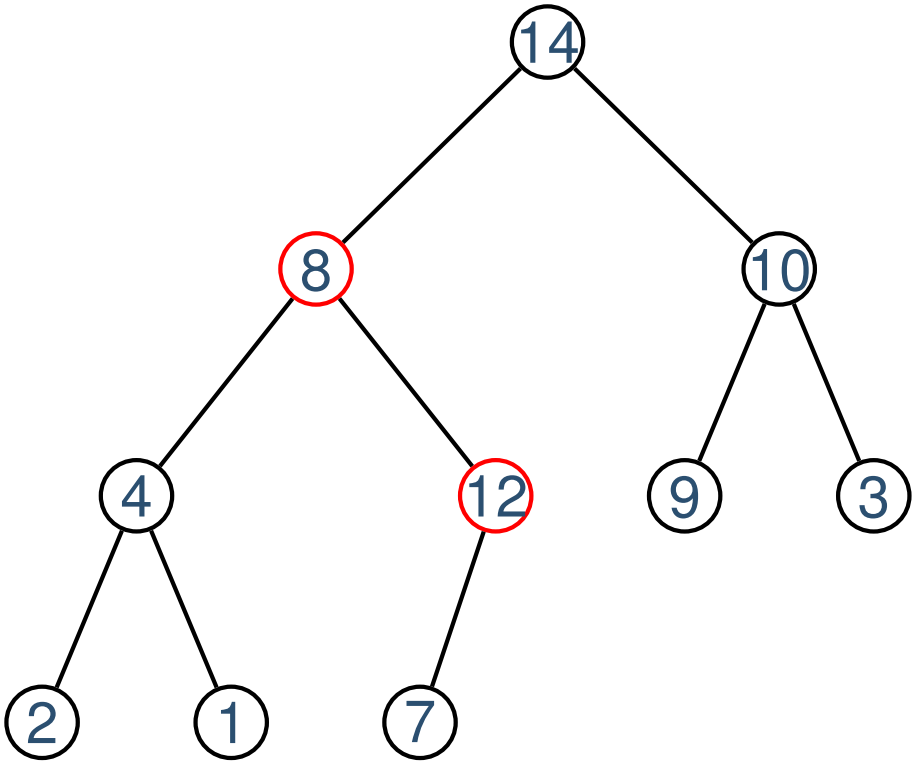
Yığının ilk elemanı silindikten sonra yeniden yapılandırılması

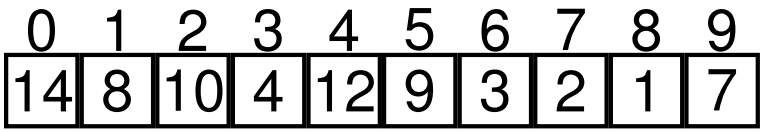

```
Nokta azamiDondur(){
    Nokta tmp;
    tmp = dizi [0];
     dizi [0]  =  dizi [tane-1];
    asagiIn(0);
    tane--;
    return tmp;
}
```

Yeni eleman eklenen yığının yeniden yapılandırılması (1)

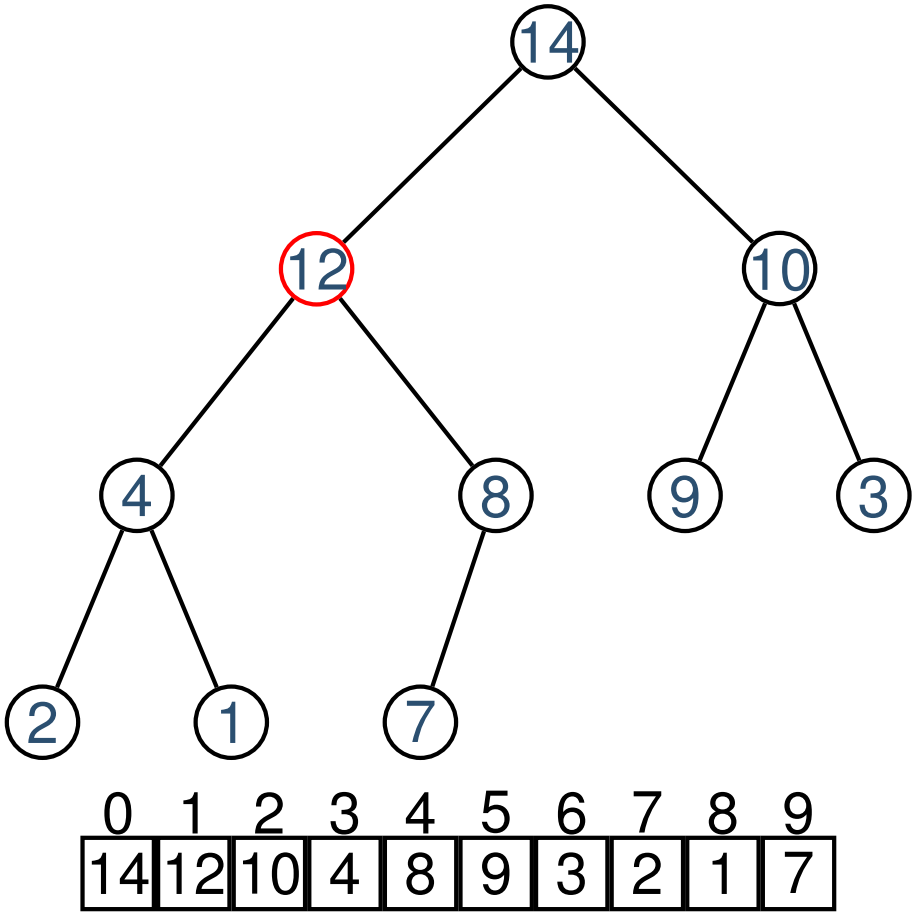








Yeni eleman eklenen yığının yeniden yapılandırılması (2)



Yığına eleman ekleme algoritması


```
void yiginEkle(Nokta yeni){
    tane++;
     dizi [tane-1] = yeni;
    yukariCik(tane - 1);
}
```

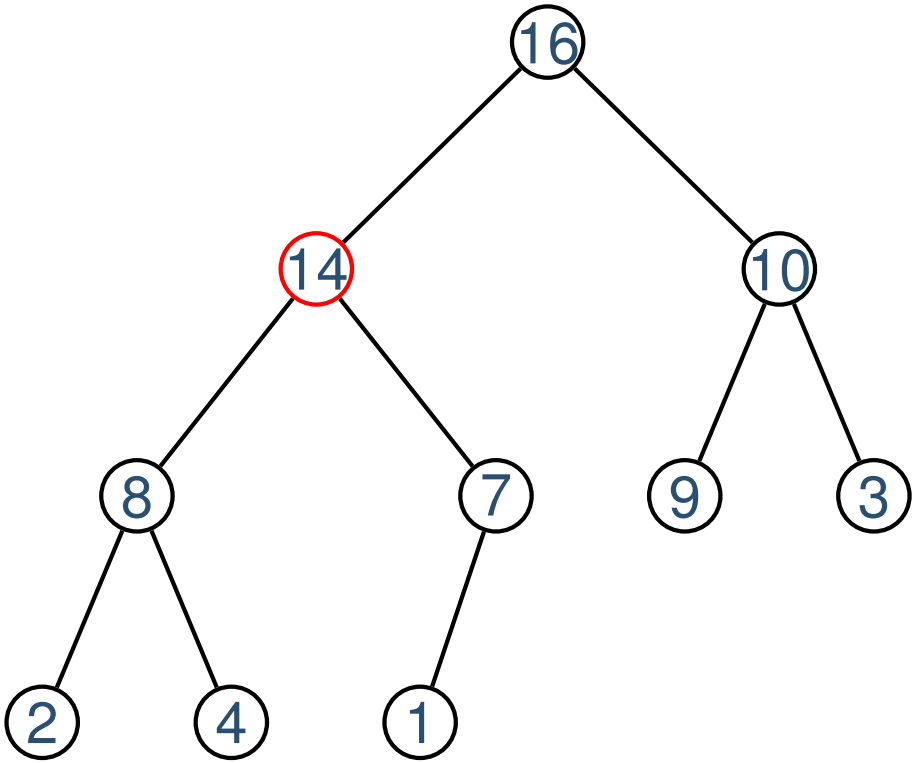
Yığın özelliğinin tekrar oluşturulması için belirli bir düğümden yukarıya doğru çıkma


```
1  void yukariCik(int no){
2      int ustdal;
3      ustdal = (no - 1) / 2;
4      while (ustdal >= 0 && dizi[ustdal].icerik < dizi[no].icerik ){
5          yerDegistir(ustdal, no);
6          no = ustdal;
7          ustdal = (no - 1) / 2;
8      }
9  }
```

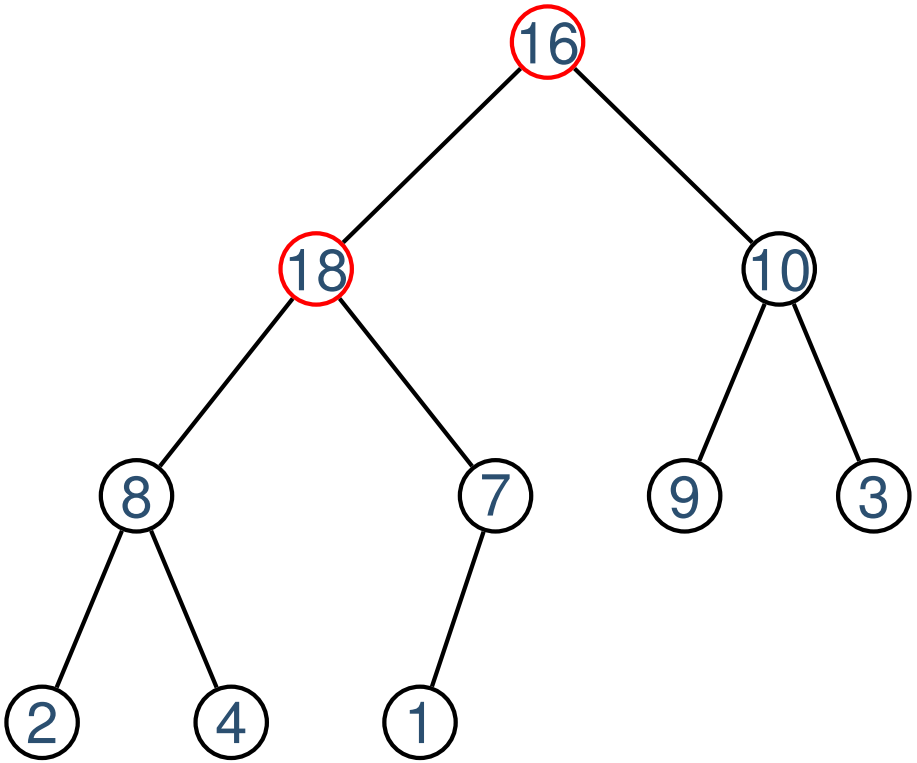
Yığında belirli bir elemanı arama


```
int yiginArama(int ad){
   int i;
   for (i = 0; i < tane; i++)
      if (dizi[i].ad == ad)
         return i;
   return -1;
}
```

Değeri 14 olan elemanın değerinin 18 yapılması (1)

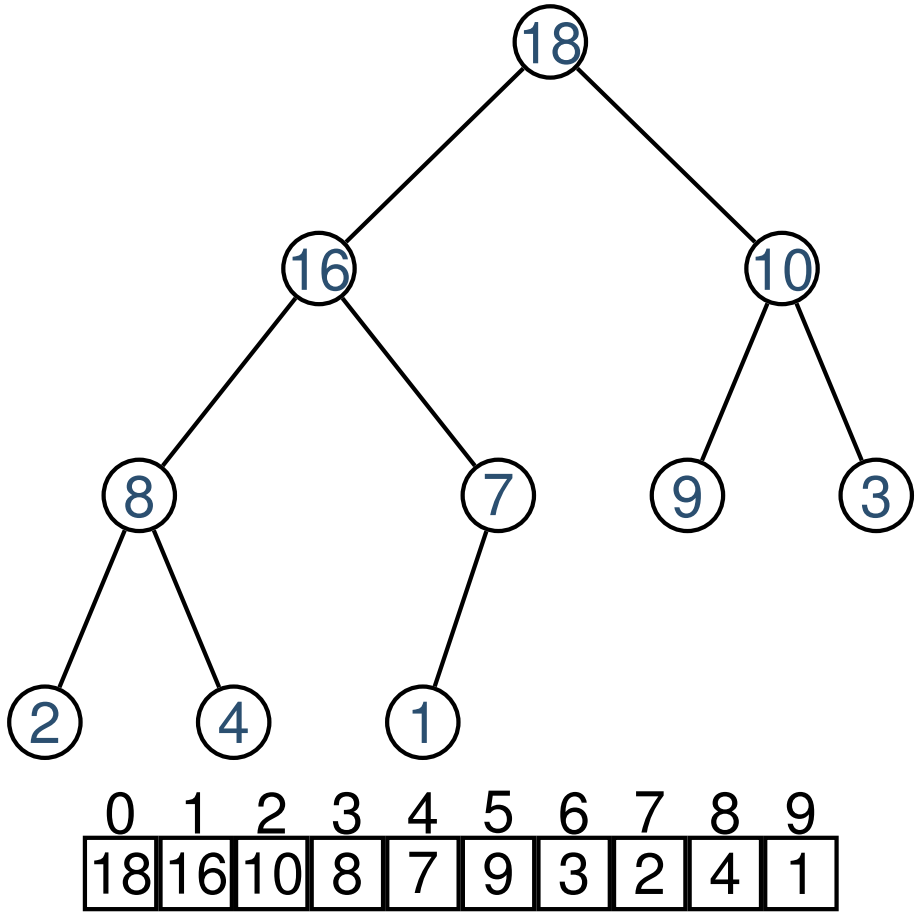


0	1	2	3	4	5	6	7	8	9
16	14	10	8	7	9	3	2	4	1

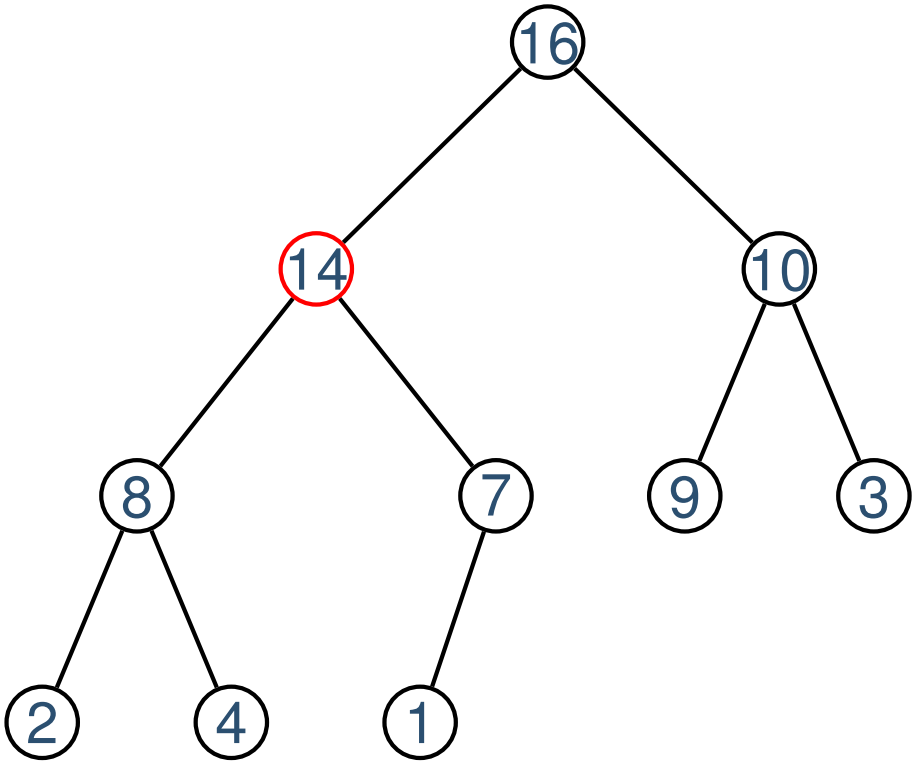


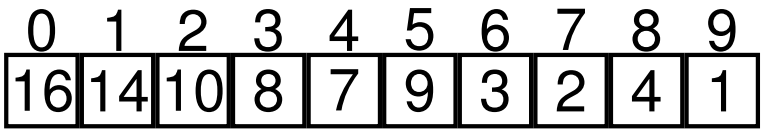
0	1	2	3	4	5	6	7	8	9
16	18	10	8	7	9	3	2	4	1

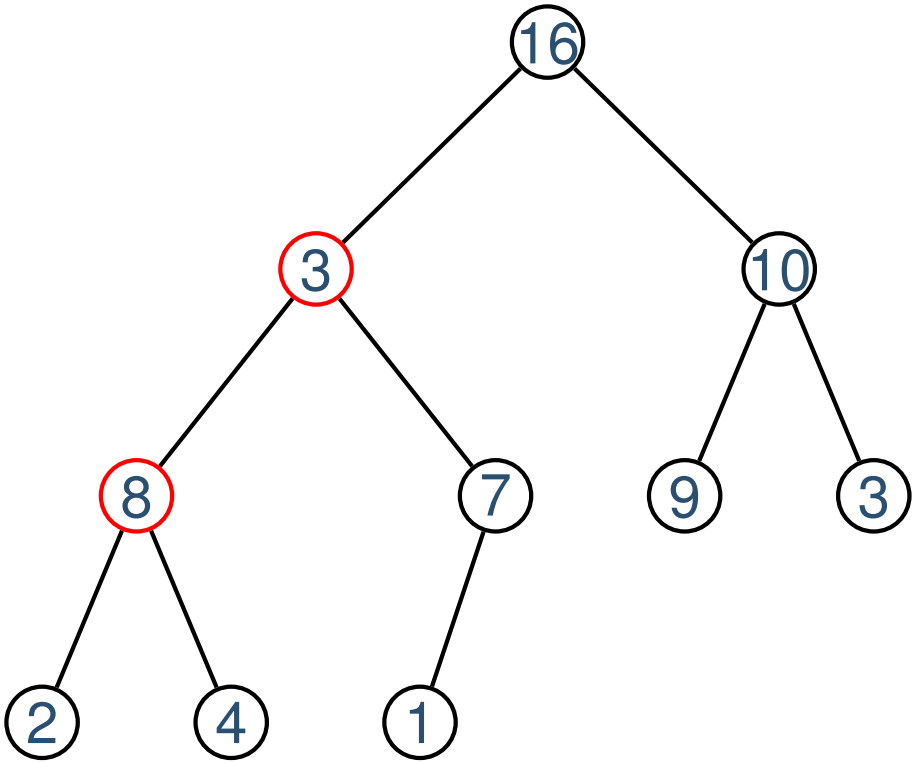
Değeri 14 olan elemanın değerinin 18 yapılması (2)

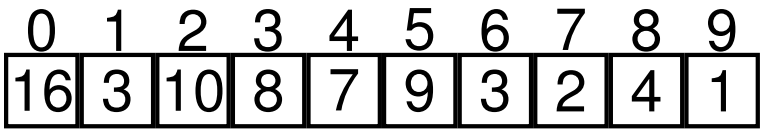


Değeri 14 olan elemanın değerinin 3 yapılması

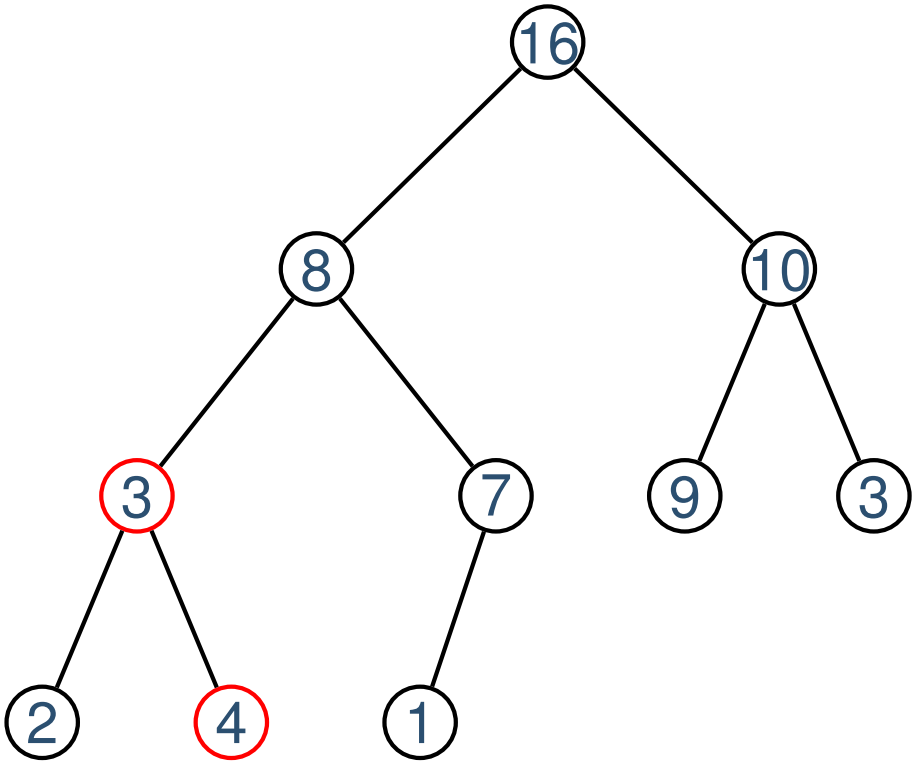


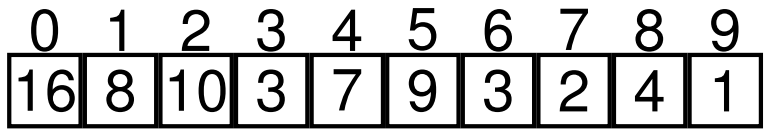


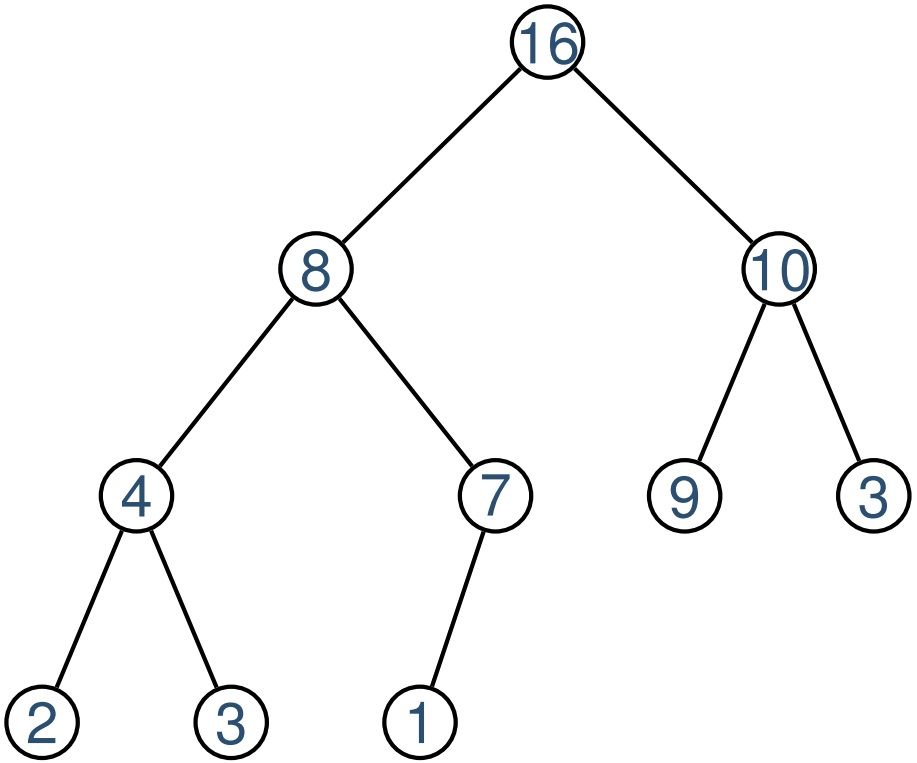


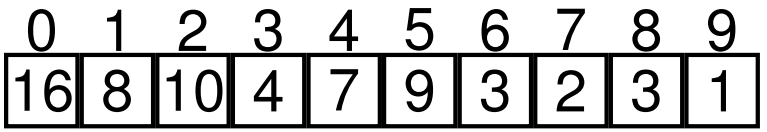


Değeri 14 olan elemanın değerinin 3 yapılması (2)









Yığındaki bir elemanın değerinin değiştirilmesi

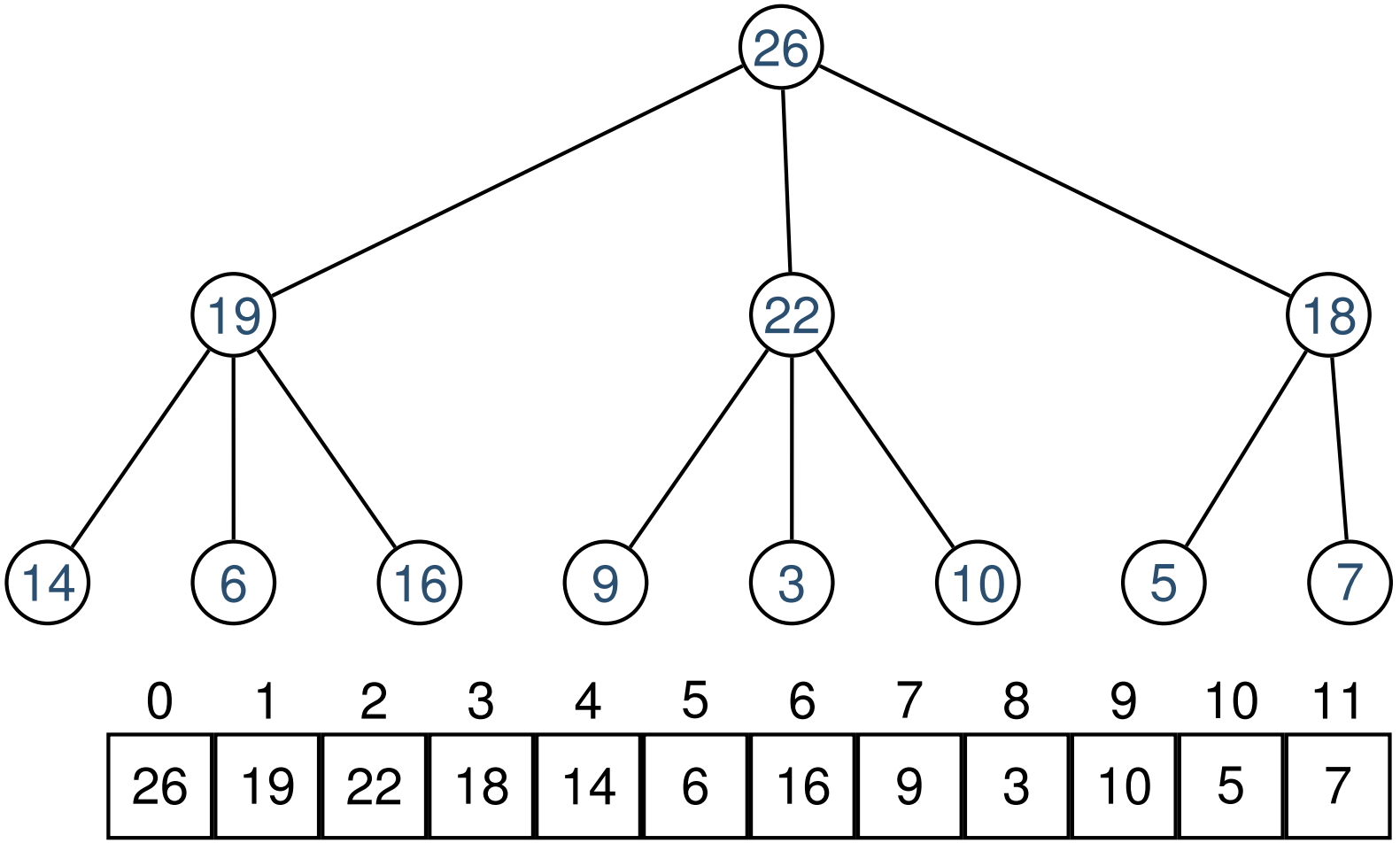

```
1 void degerDegistir(int k, int yeni){
2     int eski = dizi[k]. icerik ;
3     dizi[k]. icerik  = yeniDeger;
4     if (eski > yeni)
5         asagiIn(k);
6     else
7         yukariCik(k);
8 }
```

Yığın İşlemleri

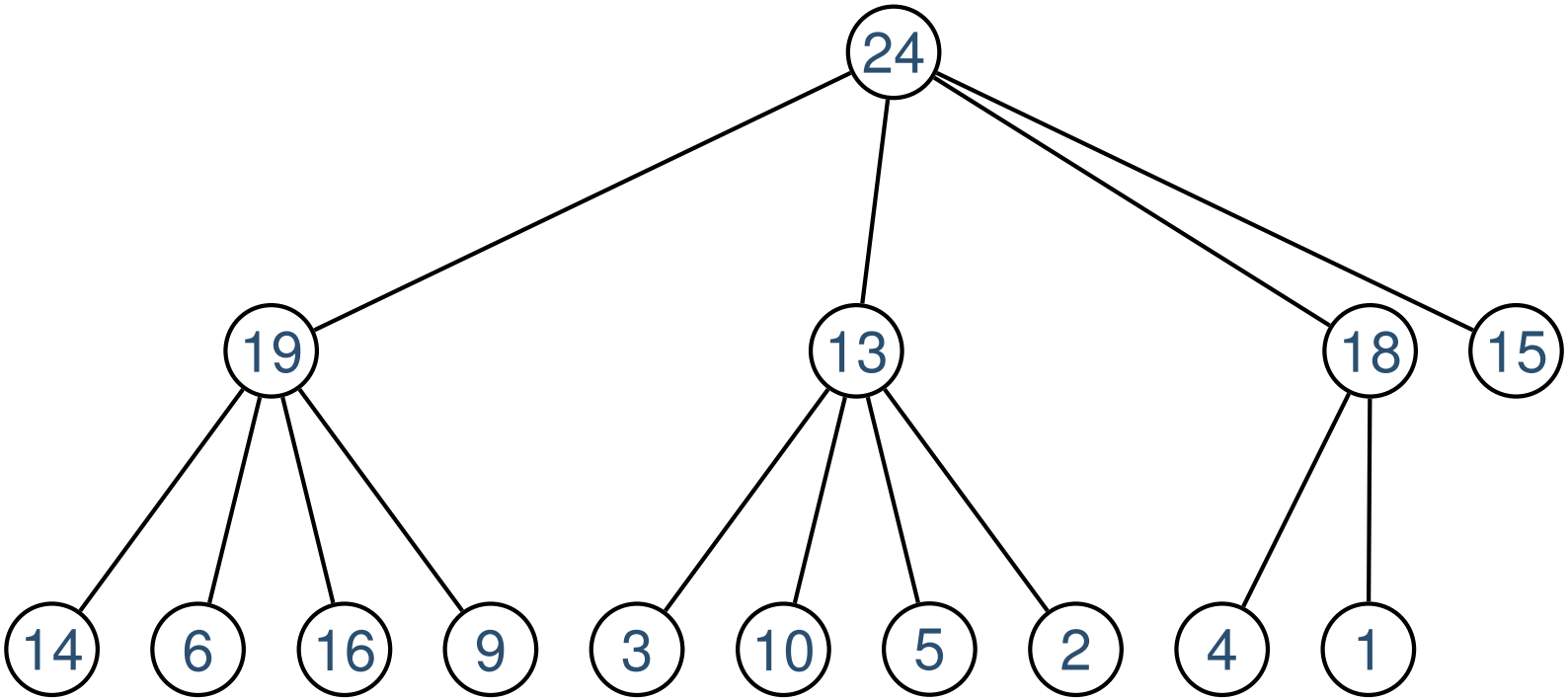
- Ekleme: $\mathcal{O}(\log N)$
- Silme: $\mathcal{O}(\log N)$
- Değer Değiştirme: $\mathcal{O}(\log N)$
- Arama: $\mathcal{O}(N)$

d-li Yığın

Örnek 3'lü-yığın



Örnek 4'lü-yığın



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
24	19	13	18	15	14	6	16	9	3	10	5	2	4	1

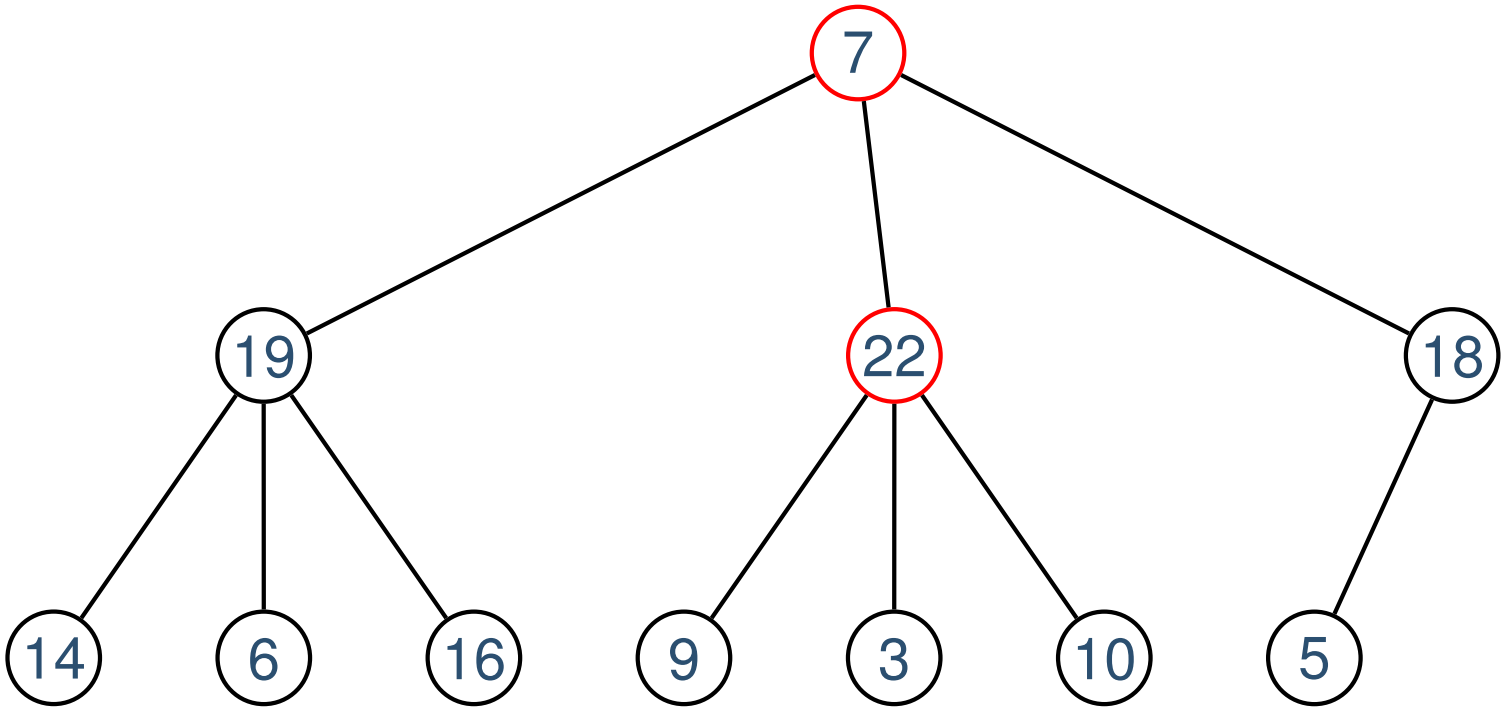
d-li Yığın tanımı


```
1  public class Yigin{
2      Nokta dizi [];
3      int tane;
4      int d;
5      public Yigin(int N, int d){
6          dizi  = new Nokta[N];
7          tane = 0;
8          this.d = d;
9      }
10 }
```

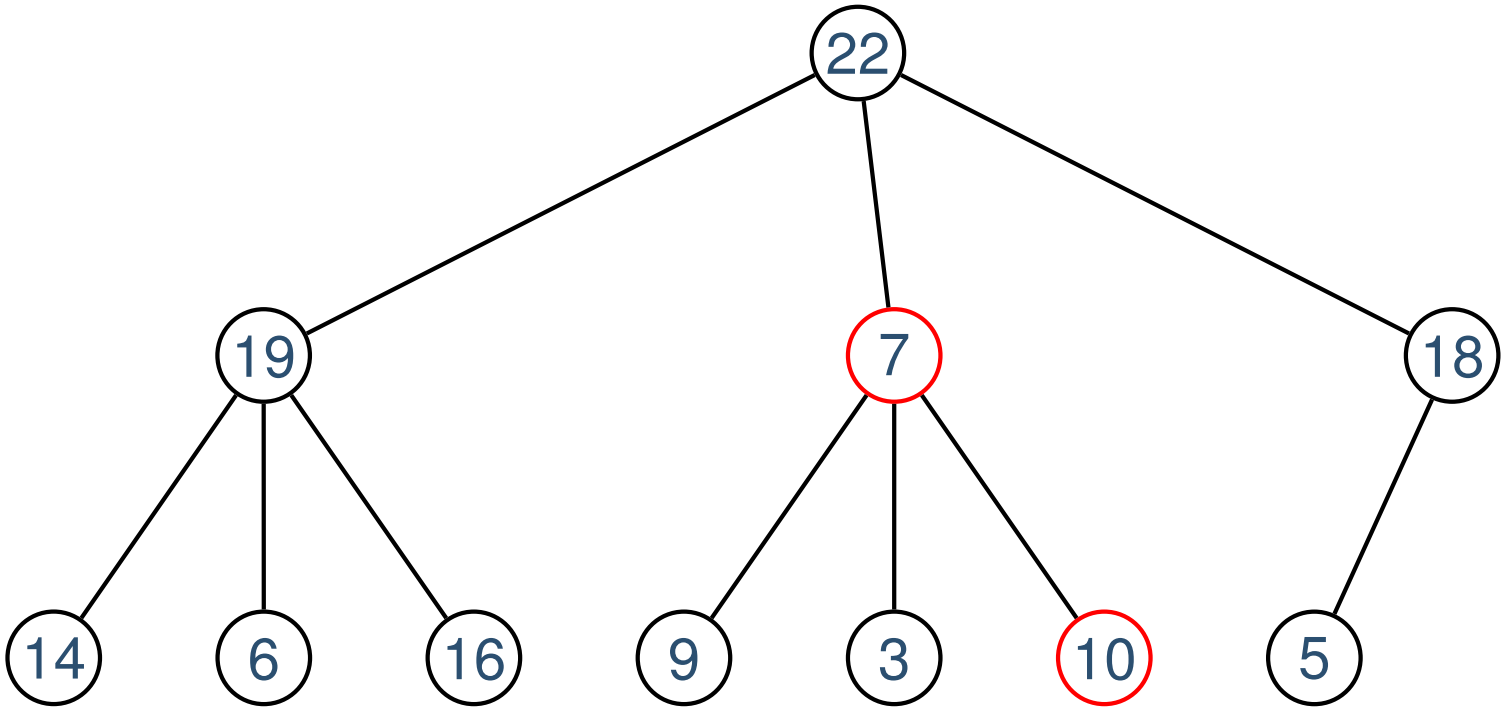
d-li yığın özelliğinin tekrar oluşturulması için belirli bir düğümden aşağıya doğru inme

```
1  void asagiIn(int no){
2      int i, cocuk, buyukCocuk, deger;
3      do{
4          deger = dizi[no].icerik;
5          for (i = 1; i <= d && d * no + i < tane; i++){
6              cocuk = d * no + i;
7              if (deger < dizi[cocuk].icerik){
8                  buyukCocuk = cocuk;
9                  deger = dizi[cocuk].icerik;
10             }
11         }
12         if (deger != dizi[no].icerik){
13             yerDegistir(no, buyukCocuk);
14             no = buyukCocuk;
15         }
16     }while (deger != dizi[no].icerik);
17 }
```

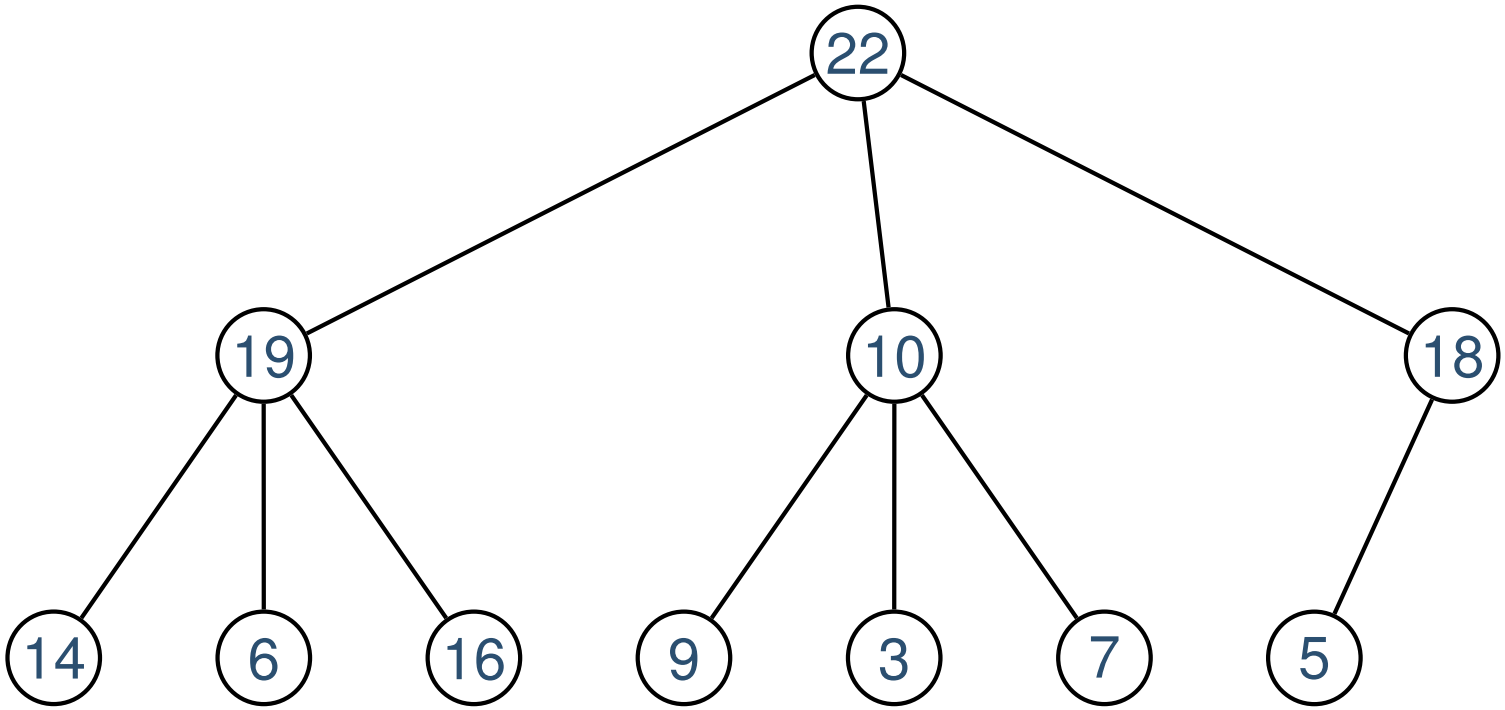
İlk eleman silindikten sonra 3'lü yığın yapısının yeniden oluşturulması (1)



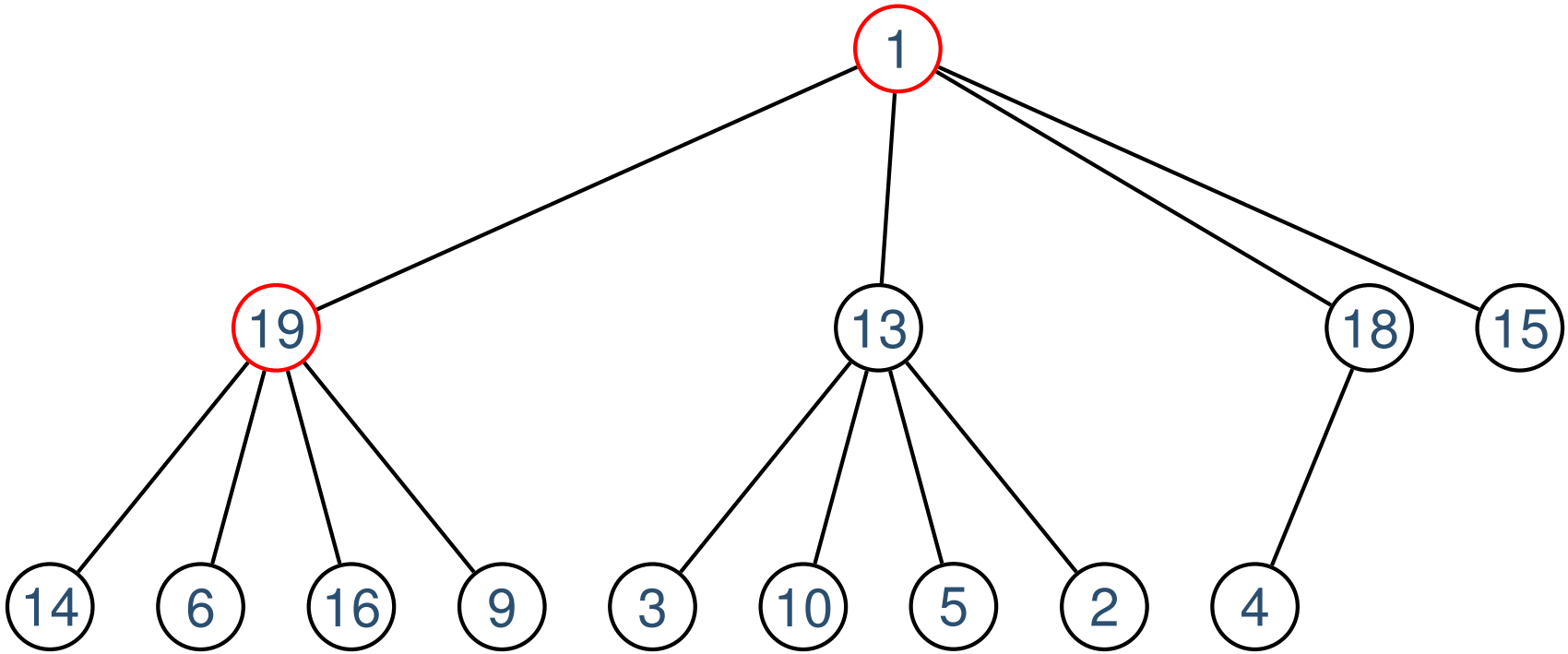
İlk eleman silindikten sonra 3'lü yığın yapısının yeniden oluşturulması (2)



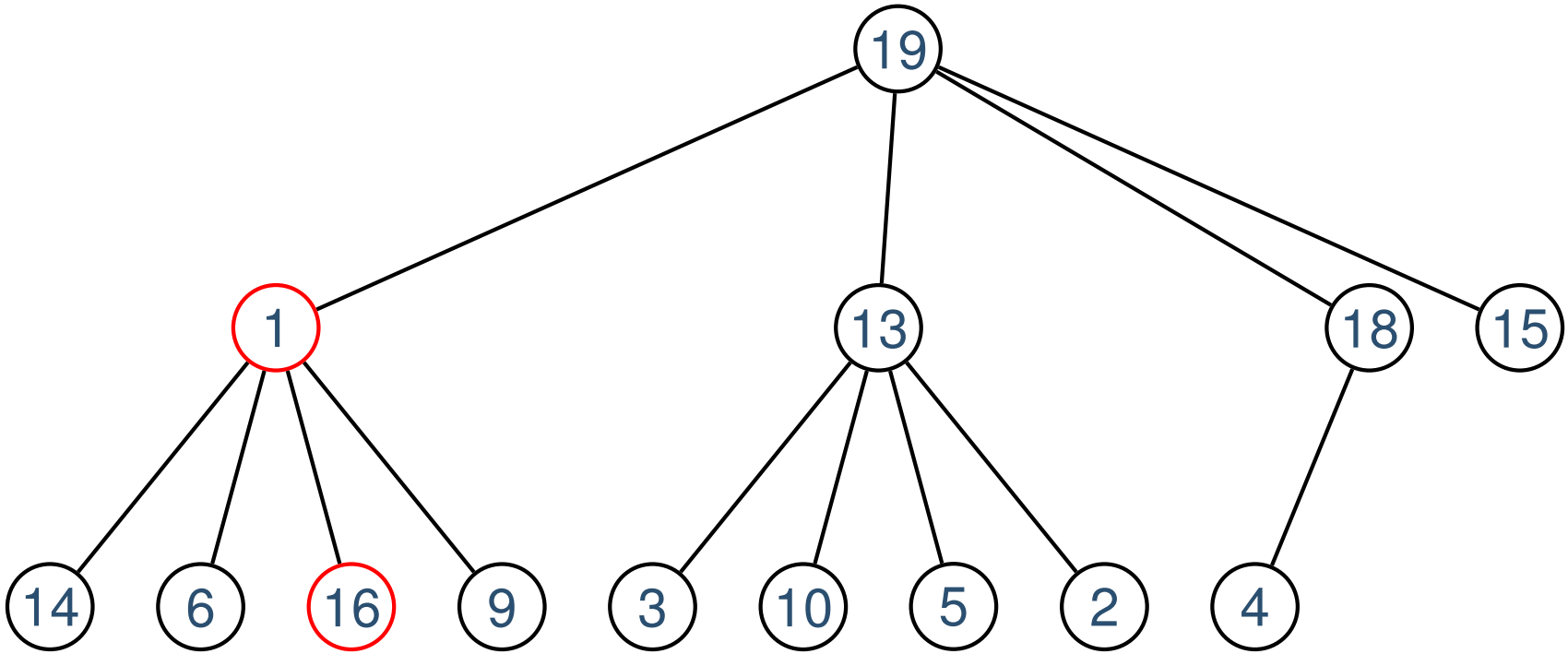
İlk eleman silindikten sonra 3'lü yığın yapısının yeniden oluşturulması (3)



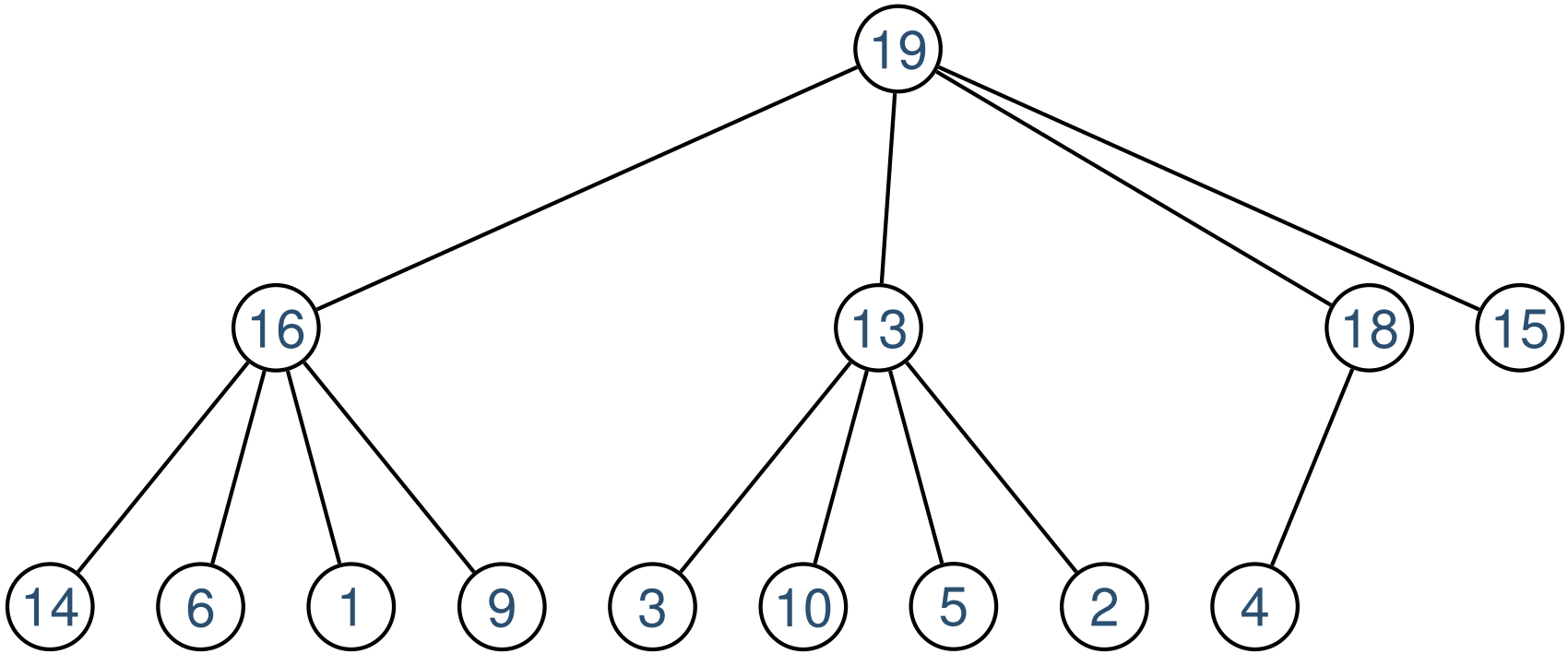
İlk eleman silindikten sonra 4'lü yığın yapısının yeniden oluşturulması (1)



İlk eleman silindikten sonra 4'lü yığın yapısının yeniden oluşturulması (2)



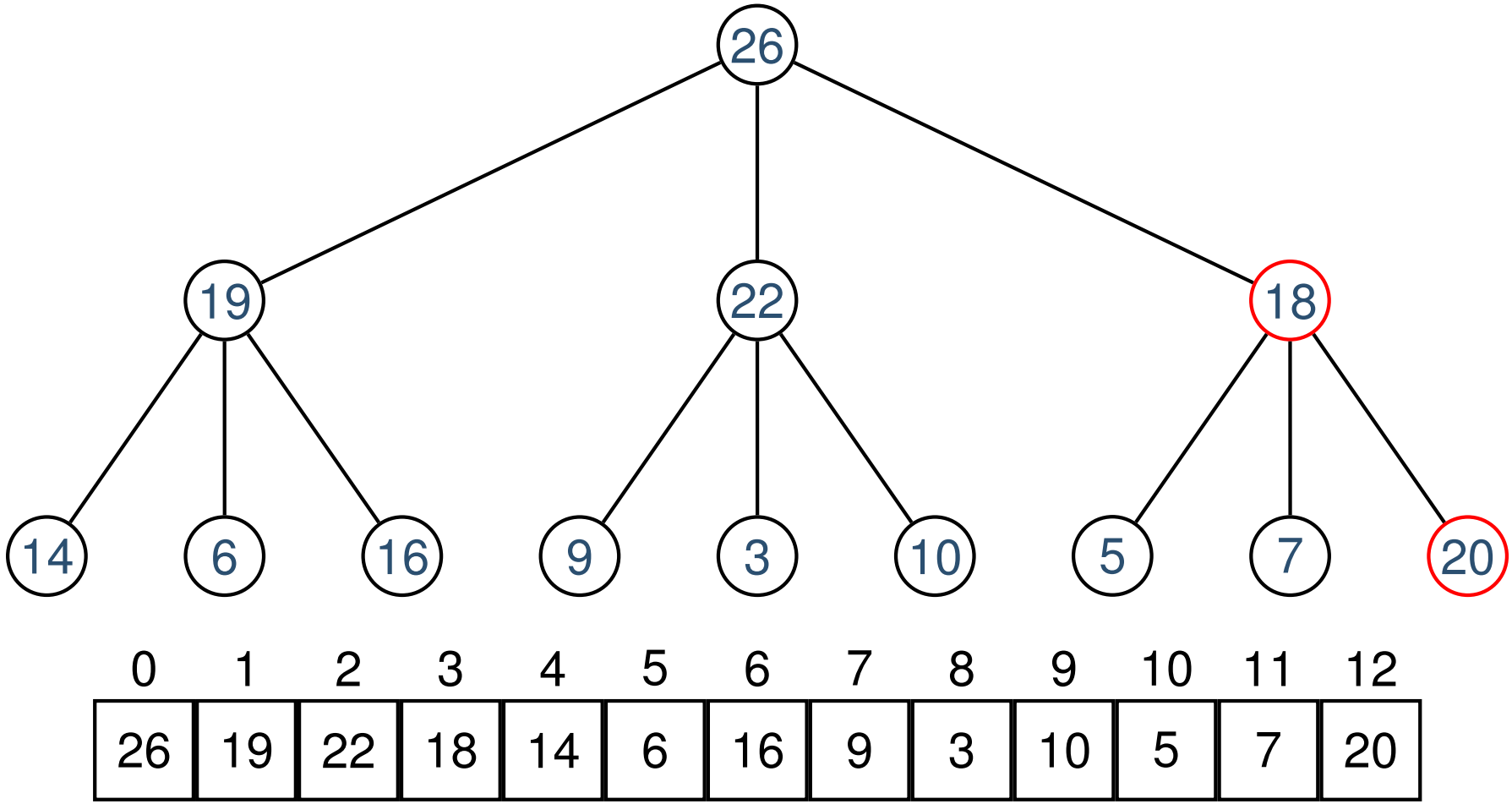
İlk eleman silindikten sonra 4'lü yığın yapısının yeniden oluşturulması (3)



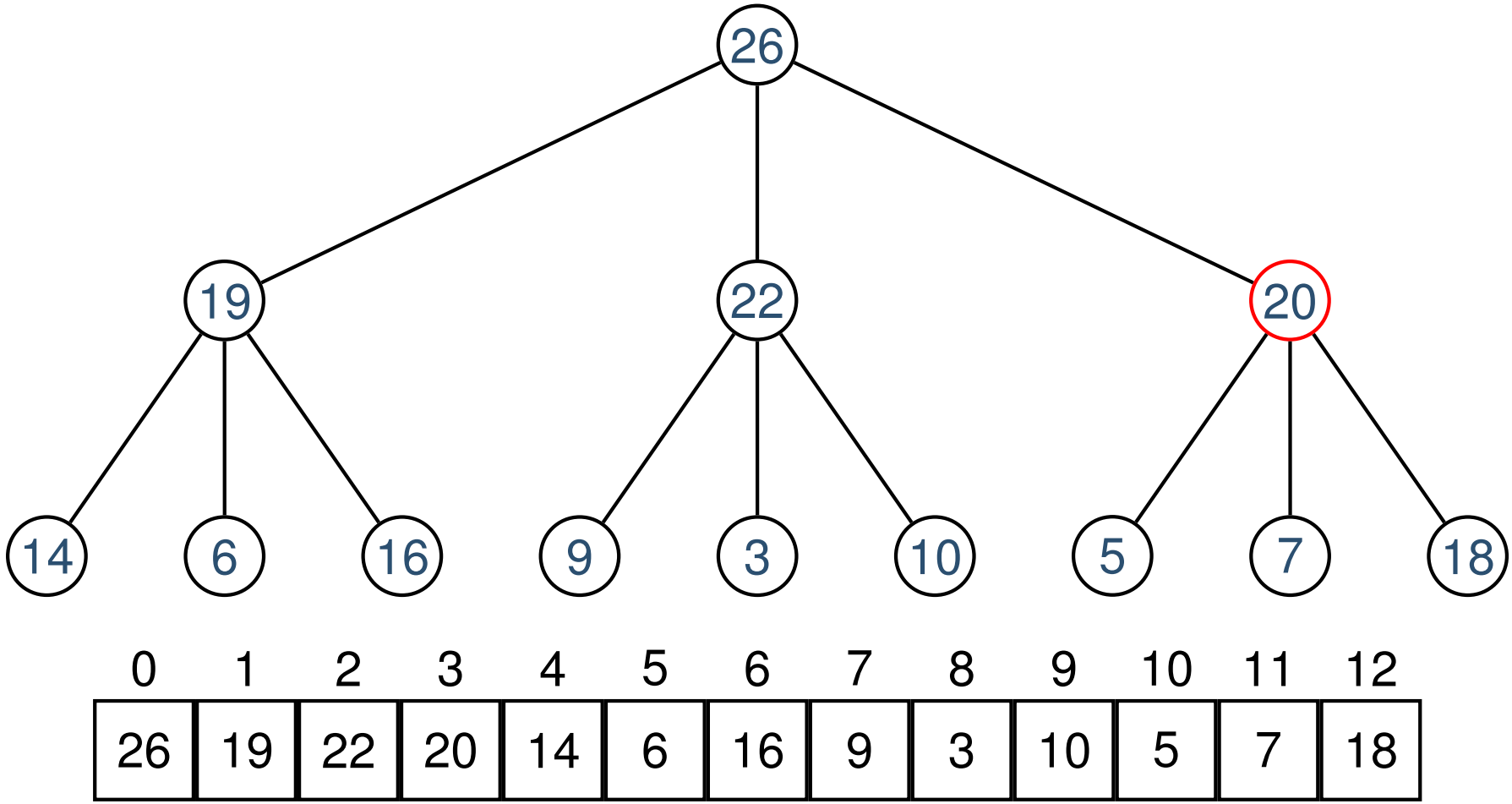
d-li yığın özelliğinin tekrar oluşturulması için belirli bir düğümden yukarıya doğru çıkma

```
1  void yukariCik(int no){
2      int ebeveyn;
3      ebeveyn = (no - 1)  /  d;
4      while (ustdal >= 0 && dizi[ebeveyn].icerik  <  dizi [no]. icerik ){
5          yerDegistir (ebeveyn, no);
6          no = ebeveyn;
7          ebeveyn = (no - 1)  /  d;
8      }
9  }
```

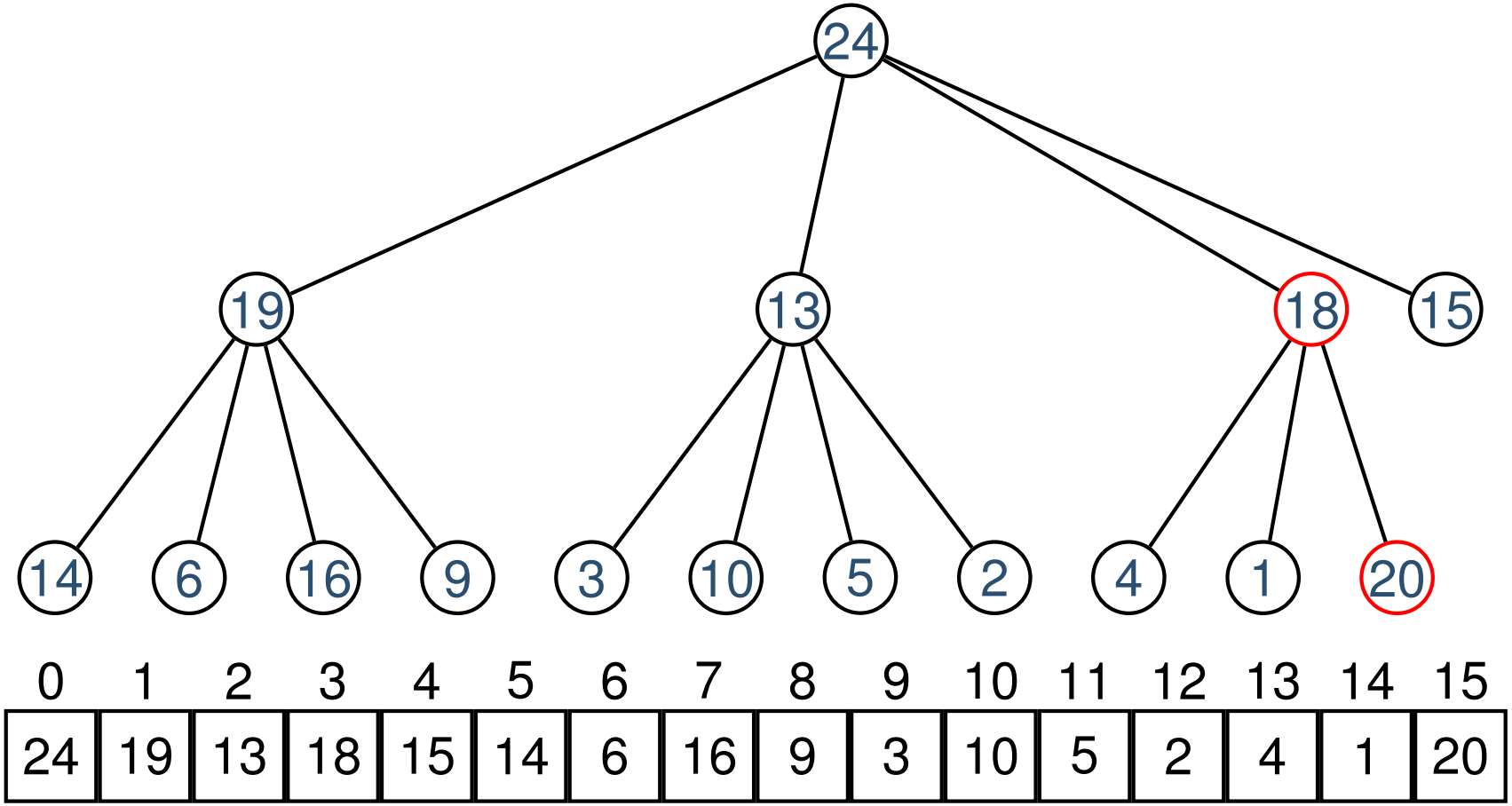
Yeni eleman eklenen üçlü yığının yeniden yapılandırılması (1)



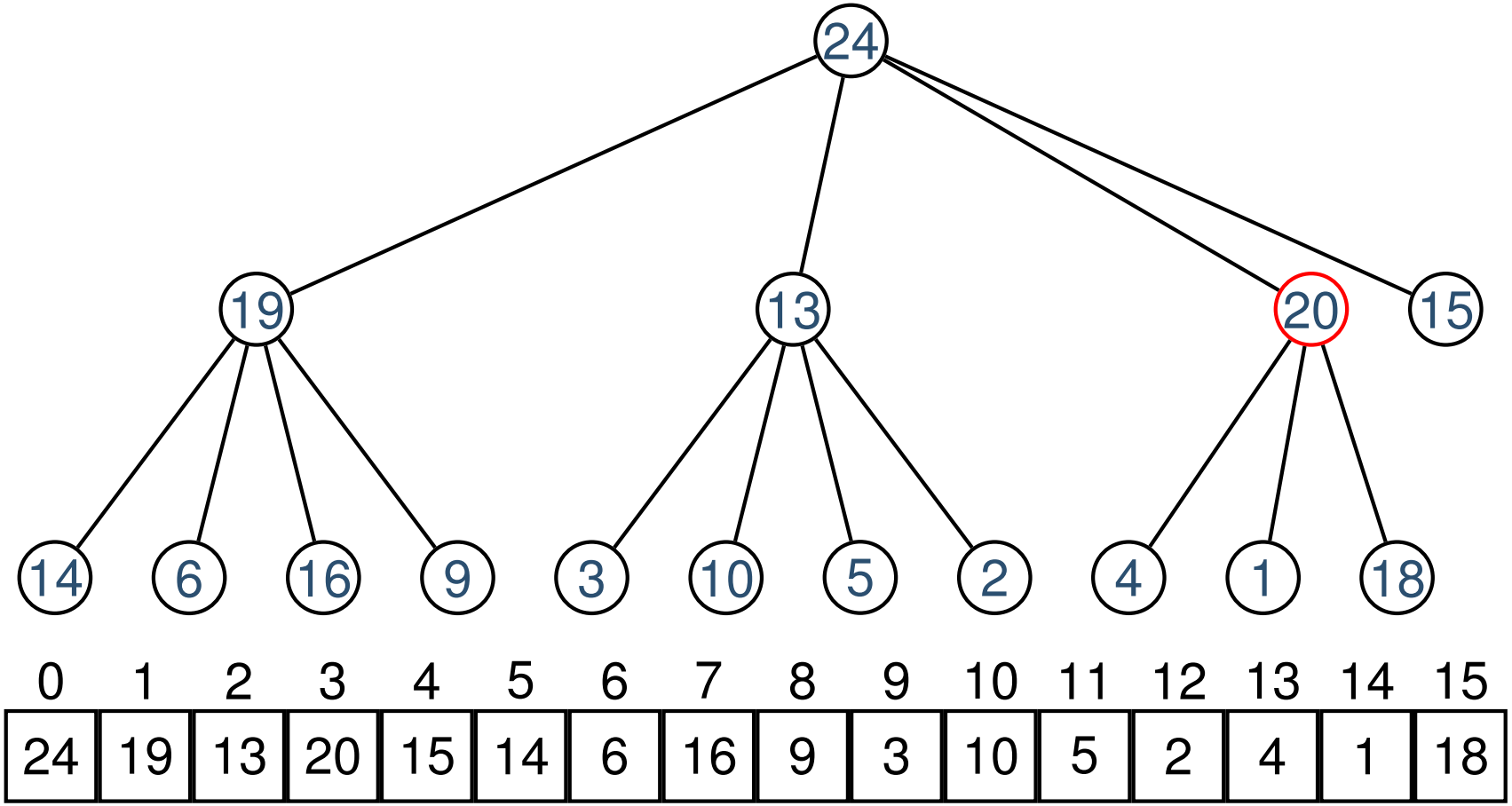
Yeni eleman eklenen üçlü yığının yeniden yapılandırılması (2)



Yeni eleman eklenen dörtlü yığının yeniden yapılandırılması (1)



Yeni eleman eklenen dörtlü yığının yeniden yapılandırılması (2)



d-li Yığın İşlemleri

- Ekleme: $\mathcal{O}(\log_d N)$
- Silme: $\mathcal{O}(\log_d N)$
- Değer Değiştirme: $\mathcal{O}(\log_d N)$
- Arama: $\mathcal{O}(N)$

Uygulama: Hedef Tahtası

Hedef tahtası probleminin geniş arama yöntemiyle çözümü (1)


```
1  boolean hedefTahtasi(int[] tahta){
2      int i, t;
3      Nokta e;
4      Yigin y;
5      Ornek o;
6      Karma kt;
7      e = new Nokta(0, 0);
8      y = new Yigin();
9      y.yiginEkle(e);
10     o = new Ornek(0);
11     kt = new Karma();
12     kt.karmaEkle(o);
```

Hedef tahtası probleminin geniş arama yöntemiyle çözümü (2)


```
13      while (!y.yiginBos()){
14          e = y.azamiDondur();
15          if (e.icerik == 100)
16              return true;
17          for (i = 0; i < 5; i++){
18              t = e.icerik + tahta[i];
19              if (t <=100)
20                  if (kt.karmaAra(t) != null){
21                      e = new Nokta(t, t);
22                      y.yiginEkle(e);
23                      o = new Ornek(t);
24                      kt.karmaEkle(o);
25                  }
26          }
27      }
28      return false;
29  }
```