# YZM 2116- VERİ YAPILARI

## DERS#8: HEAP VERİ YAPISI

# İÇERİK

Bu bölümde,

- Hatırlatmalar
- Tam İkili Ağaç
- Eksiksiz İkili Ağaç
- Dizi Kullanarak İkili Ağaç Gerçekleştirimi
- Heap, Max Heap, Min Heap
- Öncelik Kuyruğu
- HeapSort

konusuna değinilecektir.

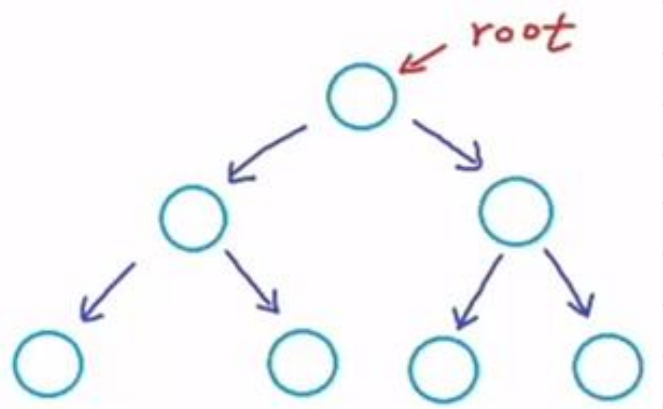
# Hatırlatmalar…

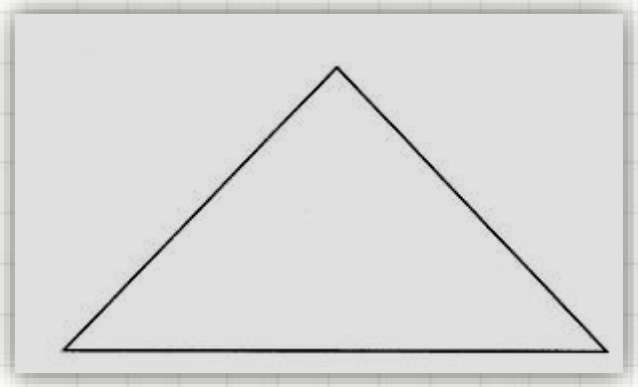
- Tam İkili Ağaç (Full Binary Tree)
- Eksiksiz İkili Ağaç (Complete Binary Tree)

**Neydi?**

# Tam İkili Ağaç (Full Binary Tree)

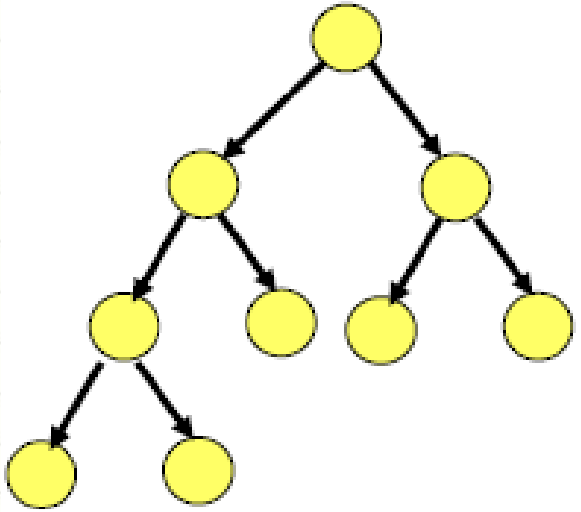
- Her bir düğümün (i)**net olarak iki çocuk düğüme** sahip olduğu ve (ii)**yaprak düğümlerin aynı seviyede** olduğu ikili ağaçtır.
- Her düğüm, eşit şekilde sağ ve sol alt-ağaçlara sahiptir.

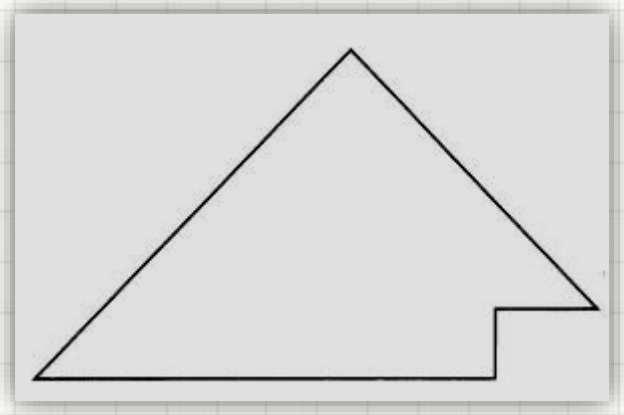


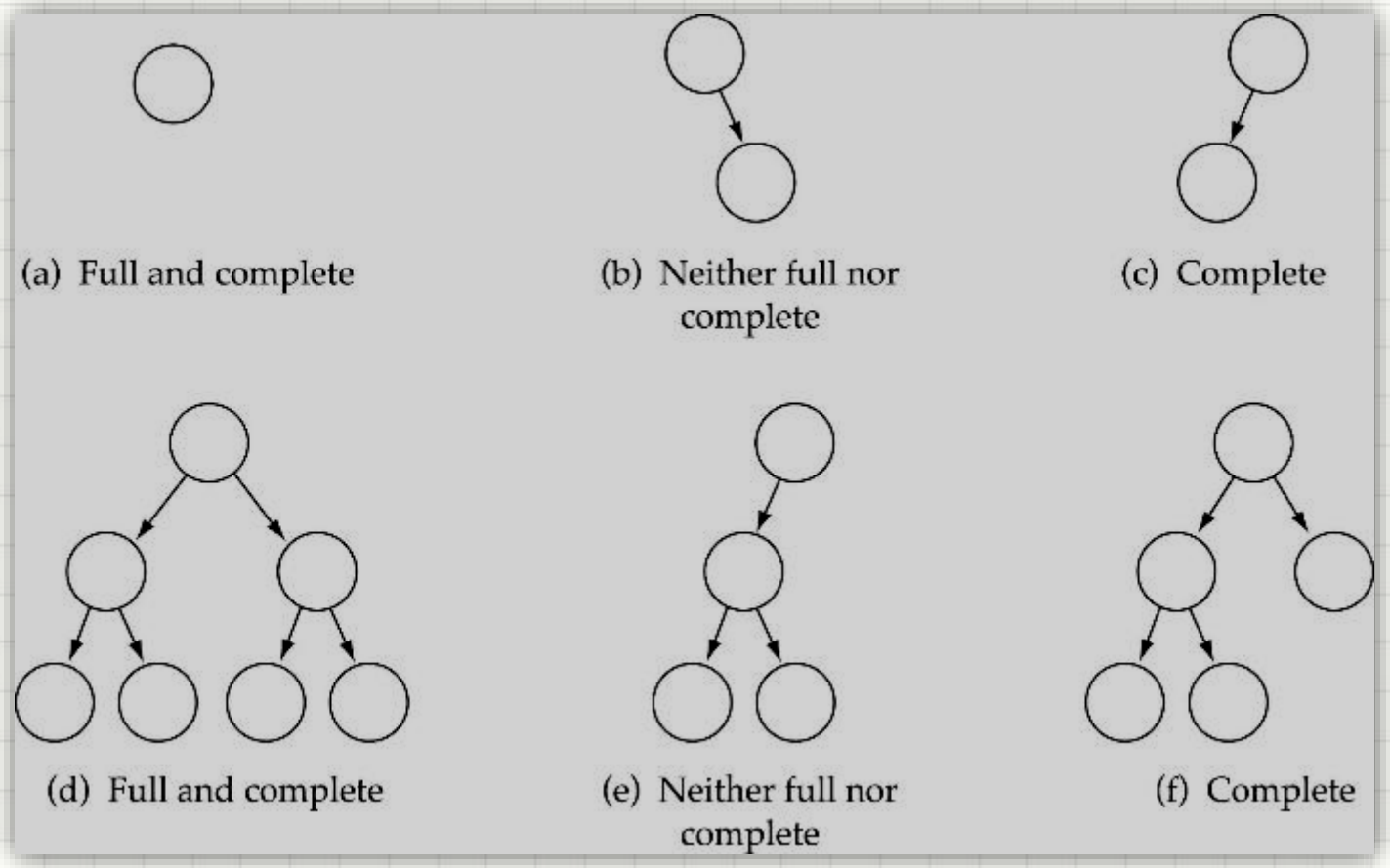
# Eksiksiz İkili Ağaç (Complete Binary Tree)

- **Son seviye dışındaki** tüm seviyelerin tam (full) olduğu ikili ağaç türüdür.
- Düğümleri **sol taraftan** (düğüme göre) doldurulur.
- ***Yeni bir derinliğe soldan sağa doğru ekleme başlanır.***





# Eksiksiz İkili Ağaç (Complete Binary Tree) (devam…)



# SORU

*DİZİ KULLANARAK İKİLİ AĞAÇ GERÇEKLEŞTİREBİLİR MİYİZ?*

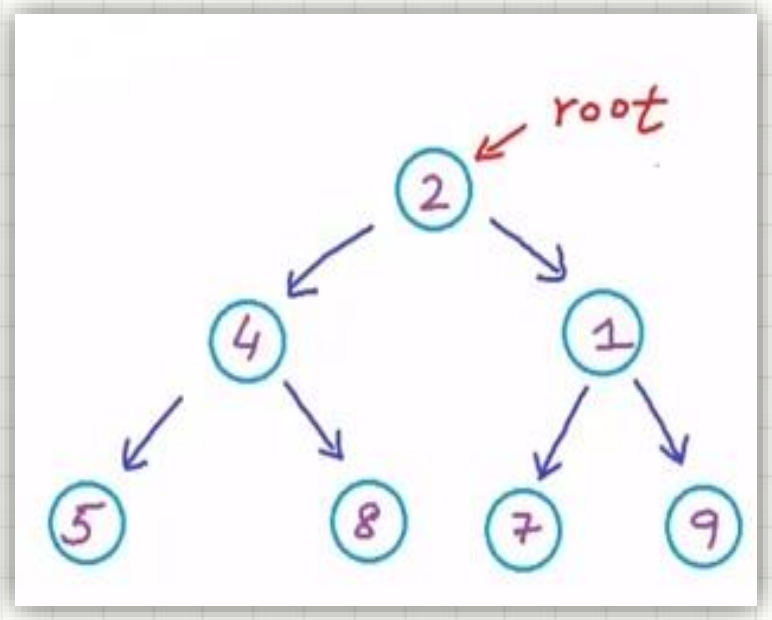
# Dizi Kullanarak İkili Ağaç Gerçekleştirimi

- *Bellek* kullanım *tasarrufu* sağlar (işaretçi yok vs.).
- Düğümler arası linkler olmadığı halde, ağaç üzerindeki düğümlerde aile ilişkisi korunur (**parent-child**).
- Gezinmeyi **basitleştirir**.

- **Veriyi Saklama Şekli**
  i. Seviye-seviye (level-by-level)
  ii. Soldan-sağa (left-to-right)

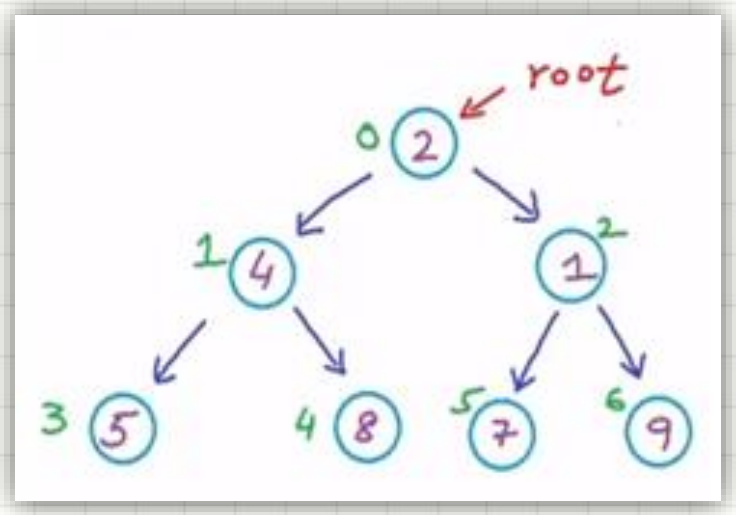
# Dizi Kullanarak İkili Ağaç Gerçekleştirimi (devam…)

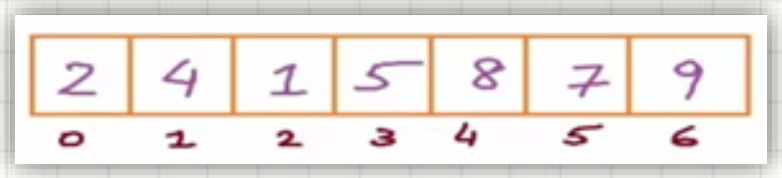
**Soru:** Aşağıdaki ikili ağacı dizide nasıl tutarız?



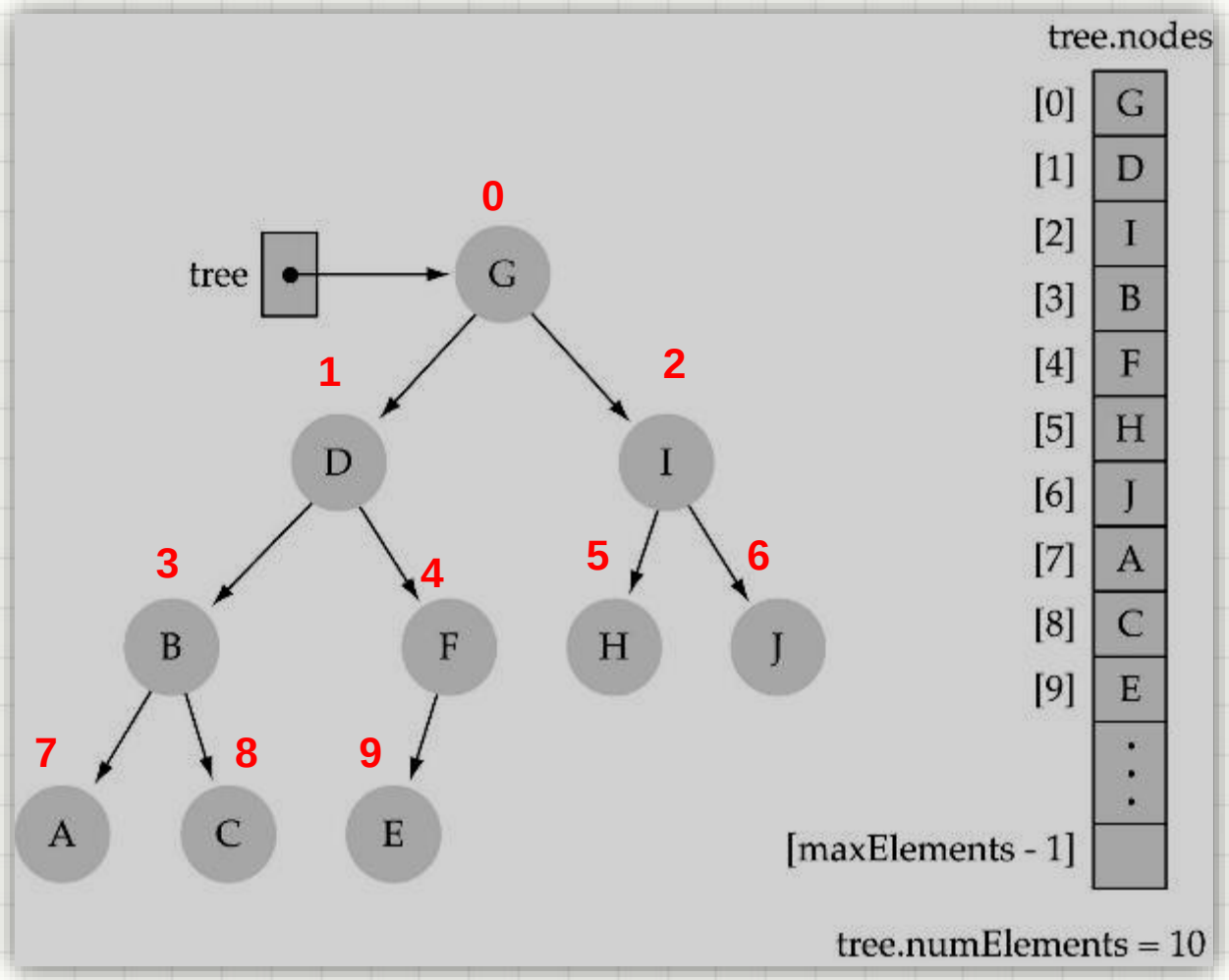
# Dizi Kullanarak İkili Ağaç Gerçekleştirimi (devam…)

**Cevap:** Kök'ü 0'dan başlatarak, sol-sağ kolları her seviyede dizi indisleri şeklinde arttırarak numaralayıp diziye atabiliriz.



# Dizi Kullanarak İkili Ağaç Gerçekleştirimi (devam…)



# Dizi Kullanarak İkili Ağaç Gerçekleştirimi (devam…)

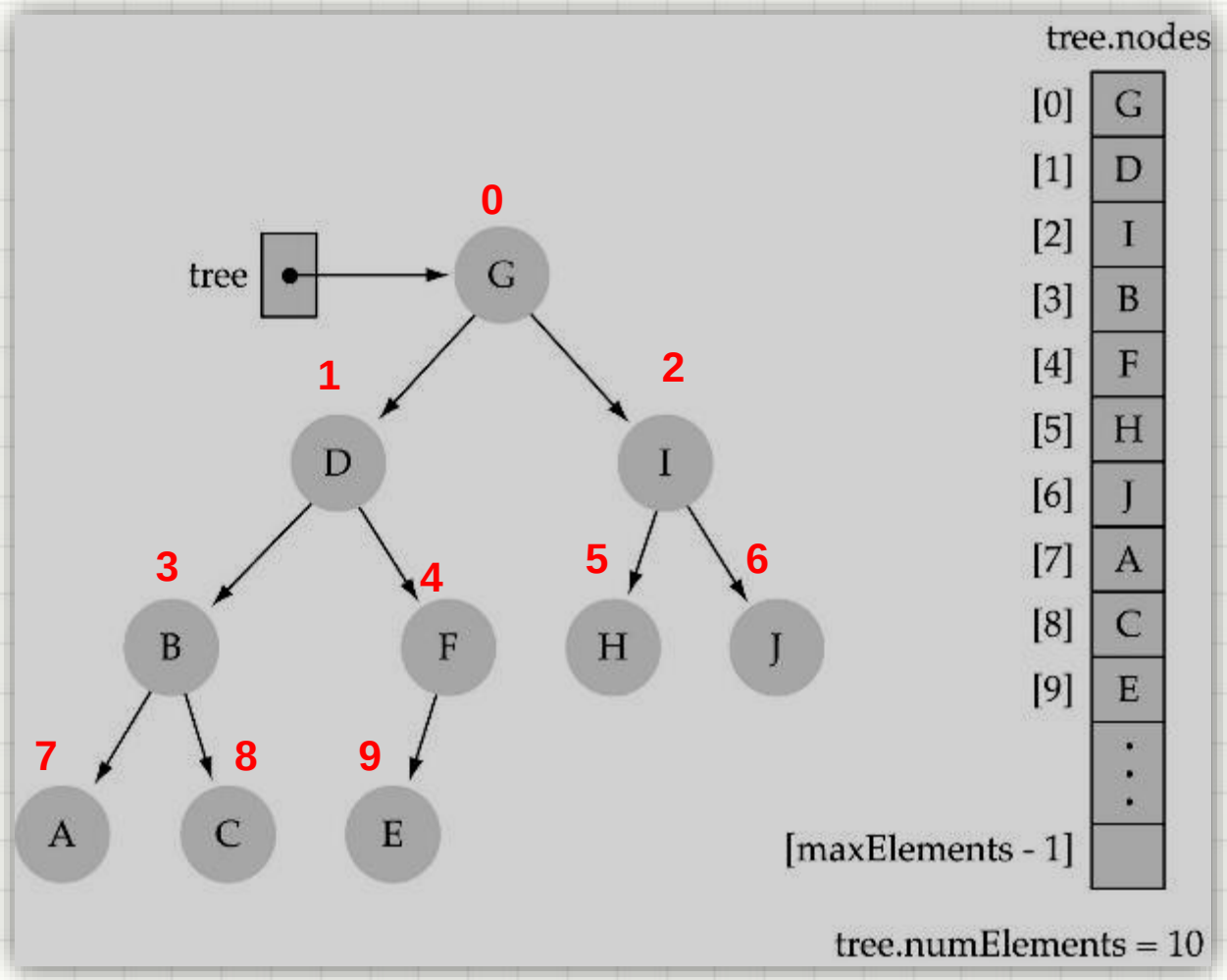
- **tree.nodes[index]** *düğümü ifade etmek üzere*
- Parent-child ilişkisi:
  - Sol çocuk = *tree.nodes[2*index+1]*
  - Sağ çocuk = *tree.nodes[2*index+2]*
  - Parent düğüm = *tree.nodes[(index-1)/2]*

(tam sayı bölüm)

- Yaprak düğümler:
  - *tree.nodes[numElements/2] - tree.nodes[numElements-1]*

# Dizi Kullanarak İkili Ağaç Gerçekleştirimi (devam…)

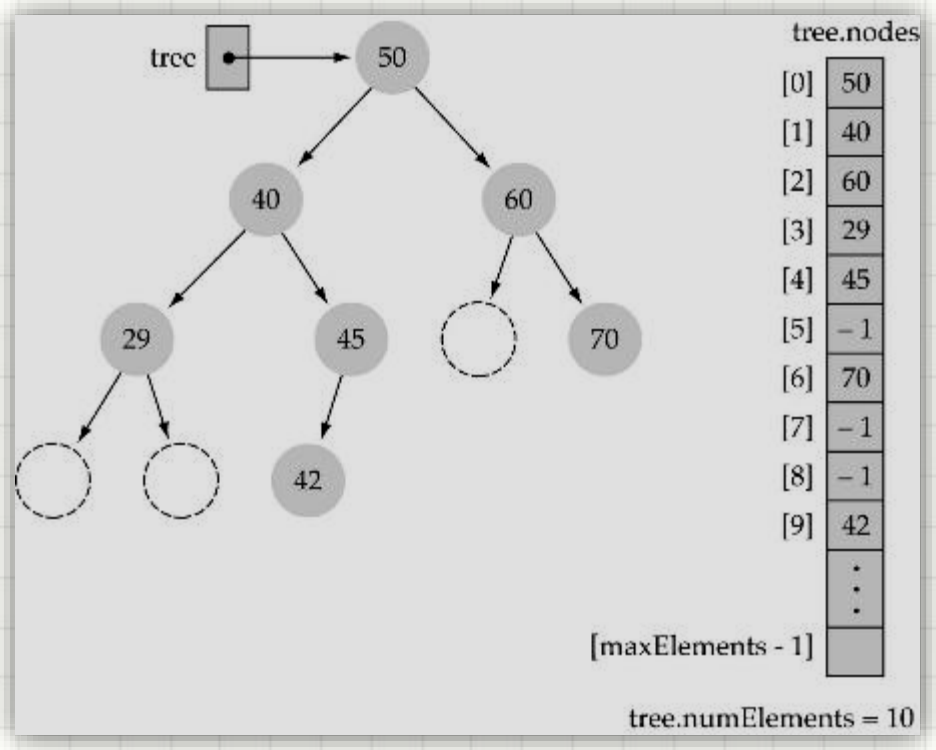
Örnek: 1 indisli D düğümü için …



# Dizi Kullanarak İkili Ağaç Gerçekleştirimi (devam…)

- Tam veya eksiksiz ikili ağaçlar, diziler ile **çok basit** ve **etkin şekilde** gerçekleştirilebilirler.
- Dizide boş alan *kalmaz*.

- Tam tersi durumdaki ikili ağaçlarda **mutlaka boş yer kullanmak** gerekir.

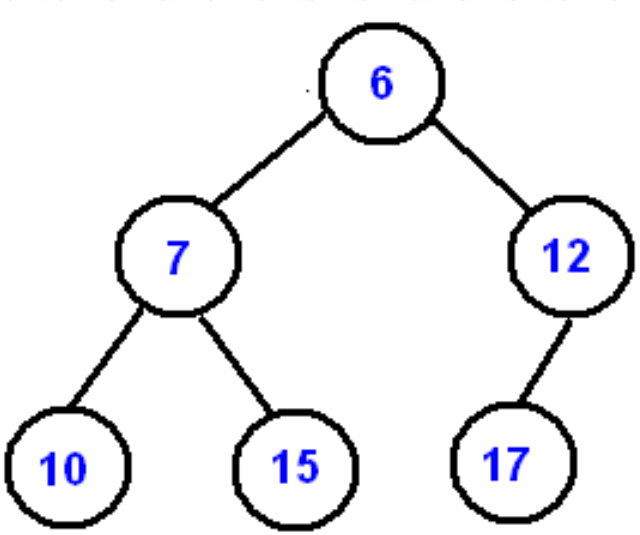


# Heap Nedir?

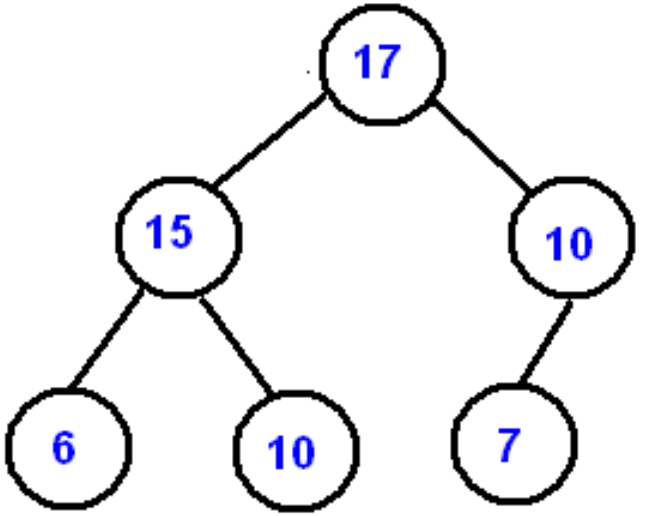
- Aşağıdaki özelliklere sahip **ikili ağaçtır**:
  - **Özellik 1:** Eksiksiz (complete) ikili ağaçtır.
  - **Özellik 2 (Heap özelliği):** Ebeveyn (parent) düğümün değeri her zaman çocukların değerinden büyük (küçük) veya eşittir.
    - **Max Heap:** Kök en büyük değere sahiptir.
    - **Min Heap:** Kök en küçük değere sahiptir.

# Max ve Min Heap

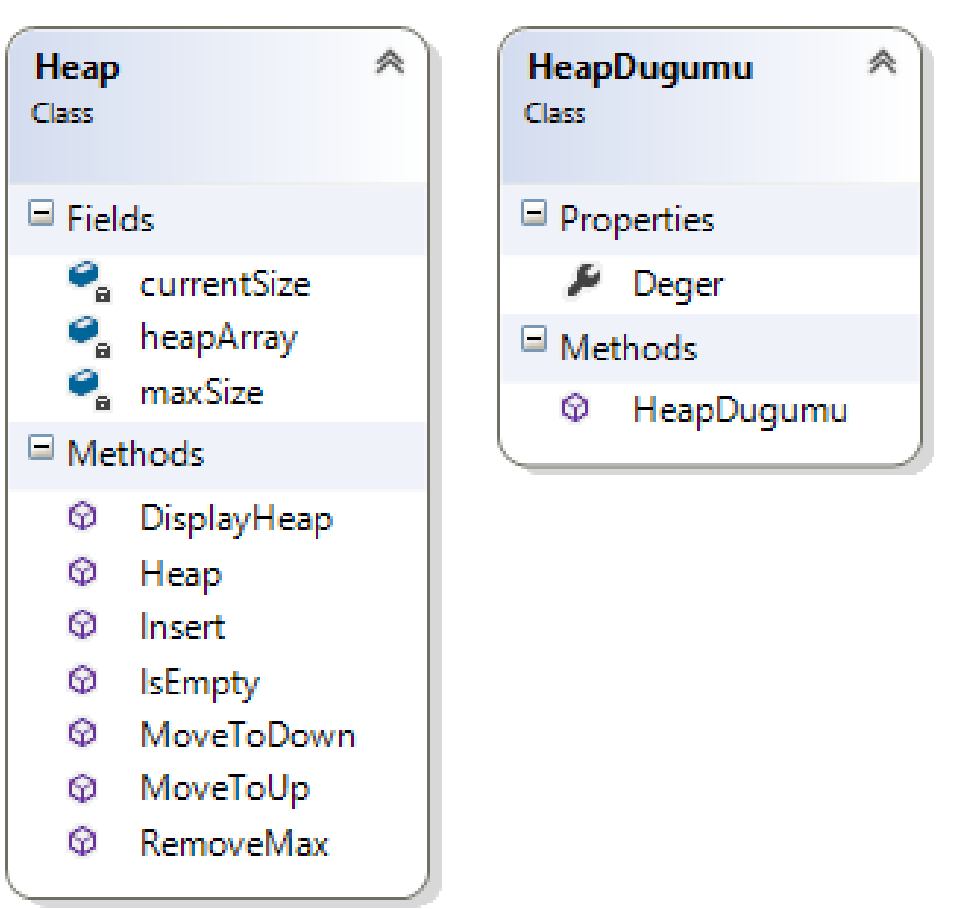
**Min Heap**




**Max Heap**

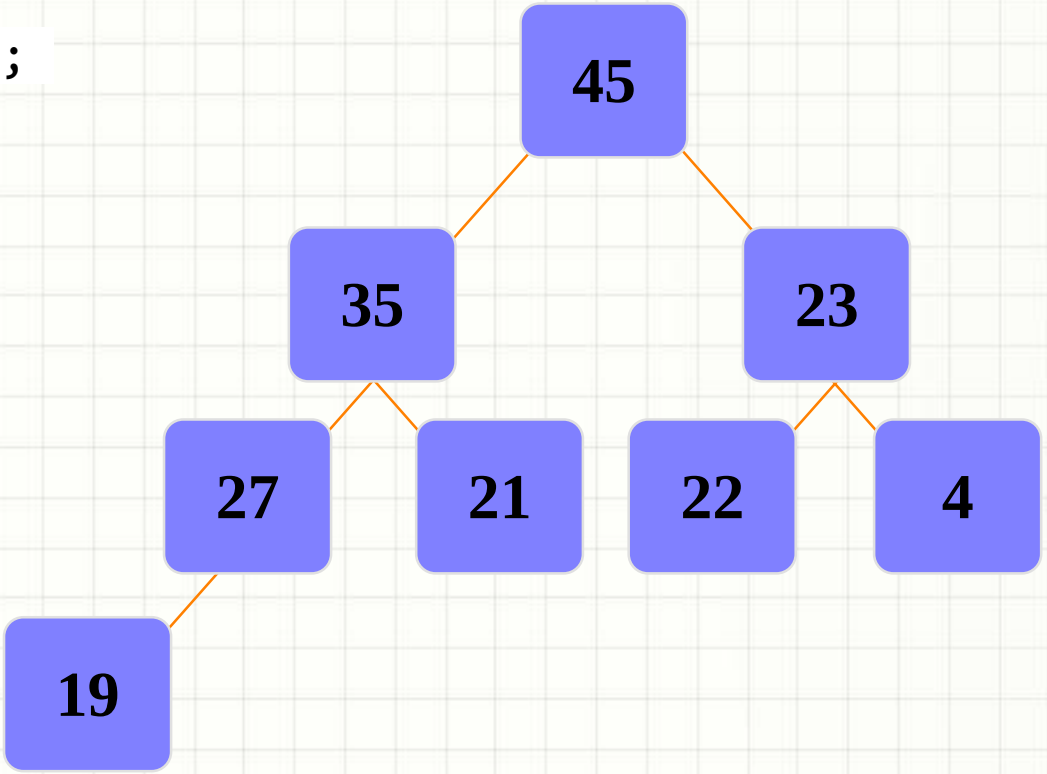


# Heap Gerçekleştirimi



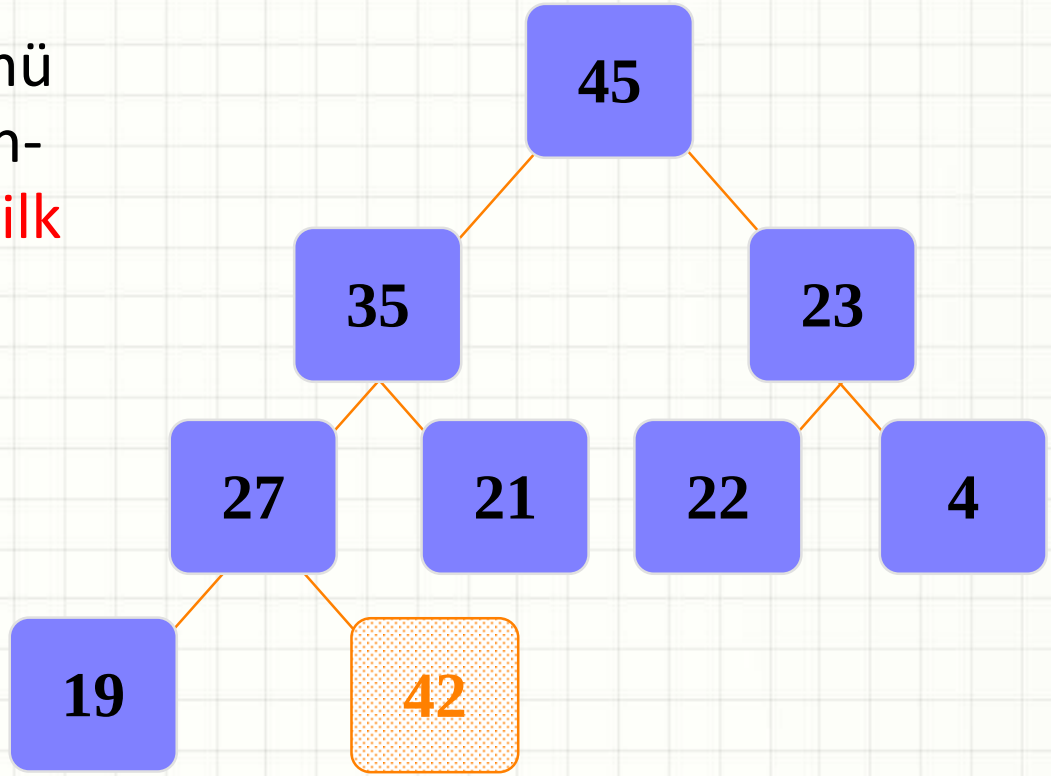
# Heap Oluşturma

```
Heap h = new Heap(100);
h.Insert(45);
h.Insert(35);
h.Insert(23);
h.Insert(27);
h.Insert(21);
h.Insert(22);
h.Insert(4);
h.Insert(19);
```



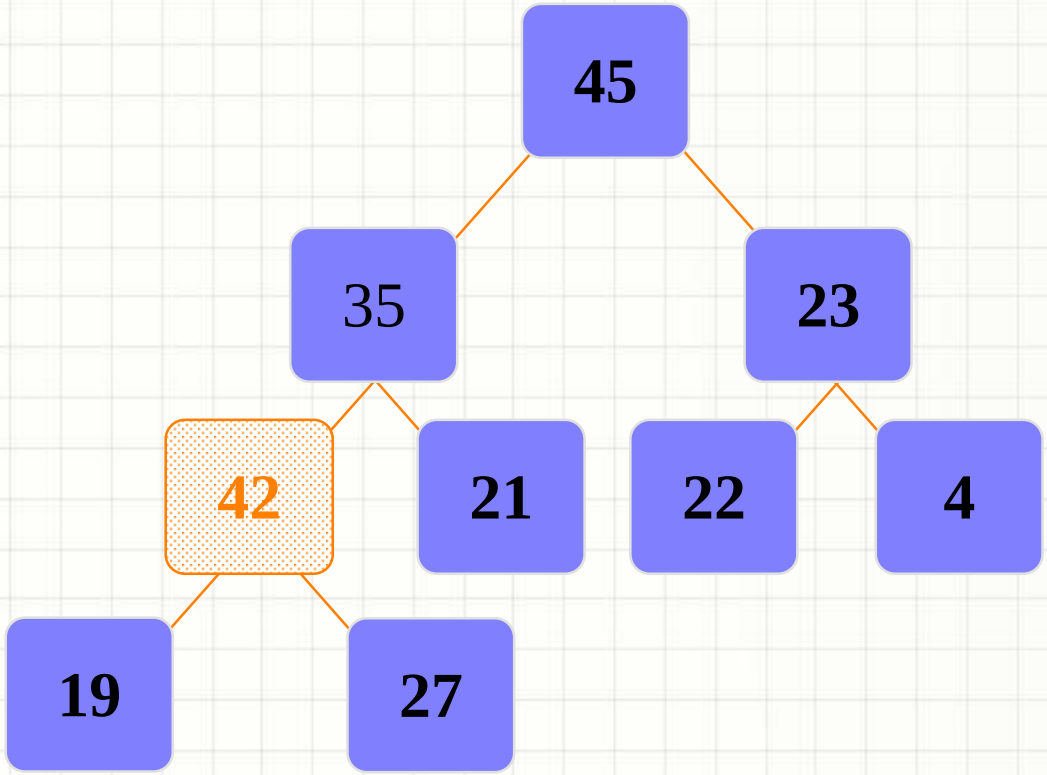
# Heap Insert(42)

- **<u>Adım 1:</u>** Yeni düğümü seviye-seviye soldan-sağa olacak şekilde ilk uygun yere ekle.



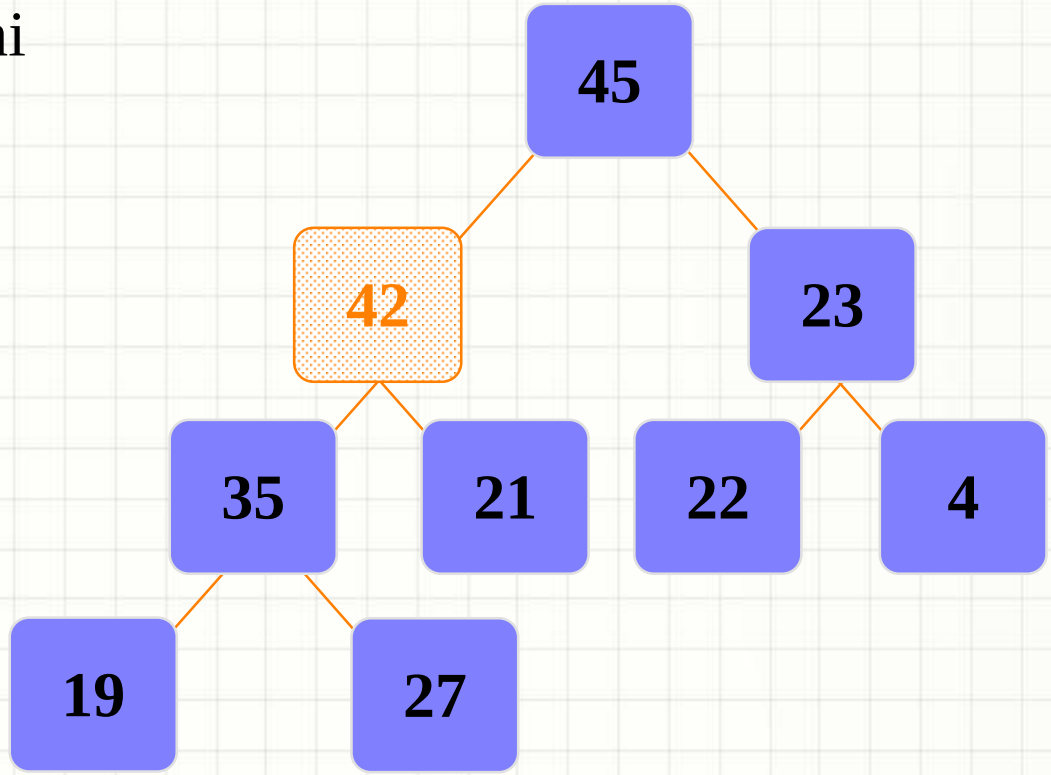
# Heap Insert(42)

- **Adım 2:** Eklenen yeni düğümü, ebeveyn ile karşılaştırarak yukarı doğru taşı.
- **Adım 3:** Uygun yer bulana kadar Adım 2 tekrarla.



# Heap Insert(42)

- **Adım 2:** Eklenen yeni düğümü, ebeveyn ile karşılaştırarak yukarı doğru taşı.
- **Adım 3:** Uygun yer bulana kadar Adım 2 tekrarla.



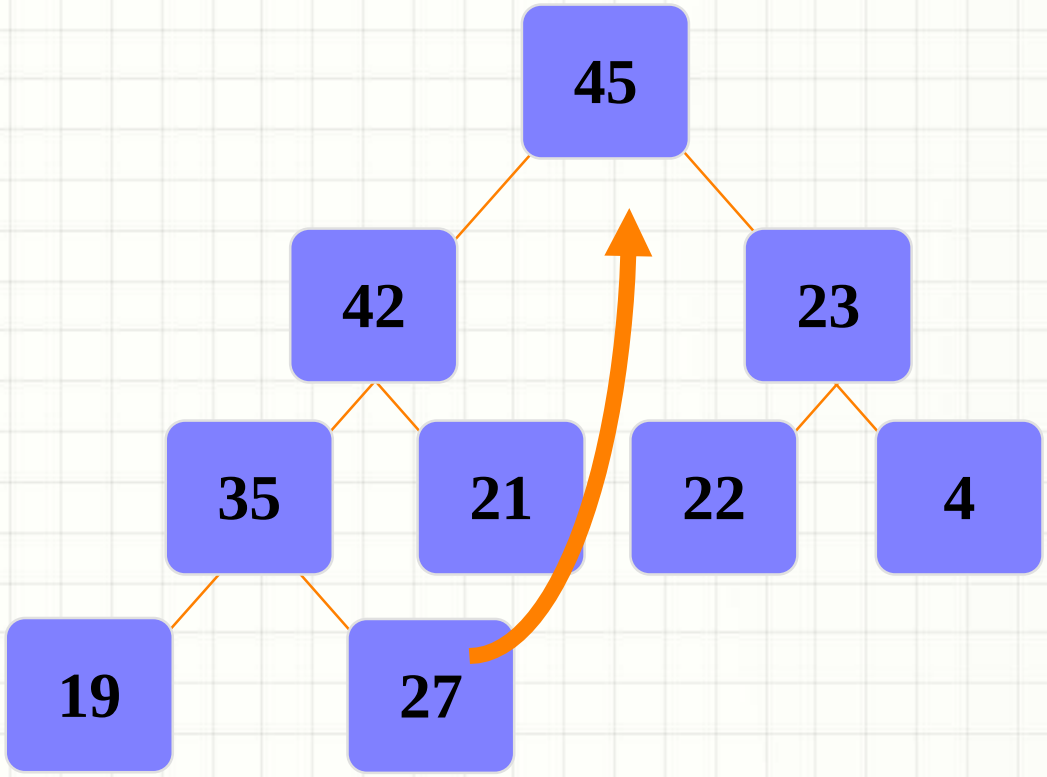
# Heap Insert Gerçekleştirimi

```
public bool Insert(int value)
{
    if (currentSize == maxSize)
        return false;
    HeapDugumu newHeapDugumu = new HeapDugumu(value);
    heapArray[currentSize] = newHeapDugumu;
    MoveToUp(currentSize++);
    return true;
}
```

```
public void MoveToUp(int index)
{
    int parent = (index - 1) / 2;
    HeapDugumu bottom = heapArray[index];
    while (index > 0 && heapArray[parent].Deger < bottom.Deger)
    {
        heapArray[index] = heapArray[parent];
        index = parent;
        parent = (parent - 1) / 2;
    }
    heapArray[index] = bottom;
}
```

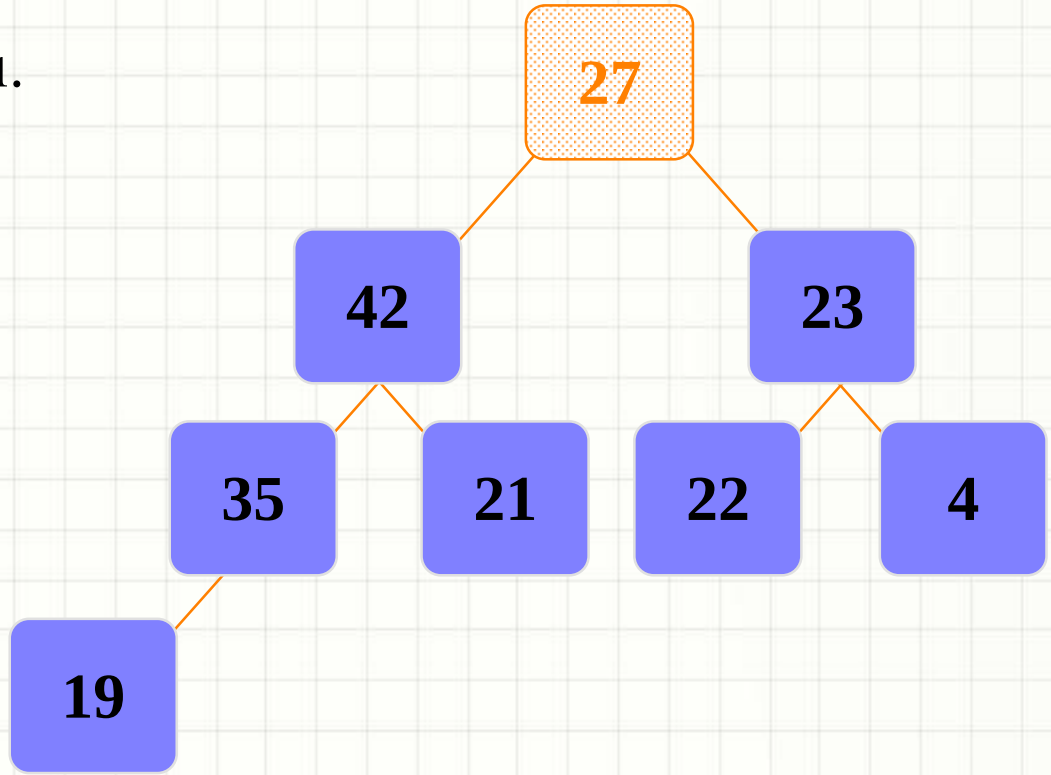
# Max Elemanı Silme (RemoveMax)

- **Adım 1:** Son eleman kök eleman yerine taşınır.
- Örnekte, 27 elemanı köke taşınır.



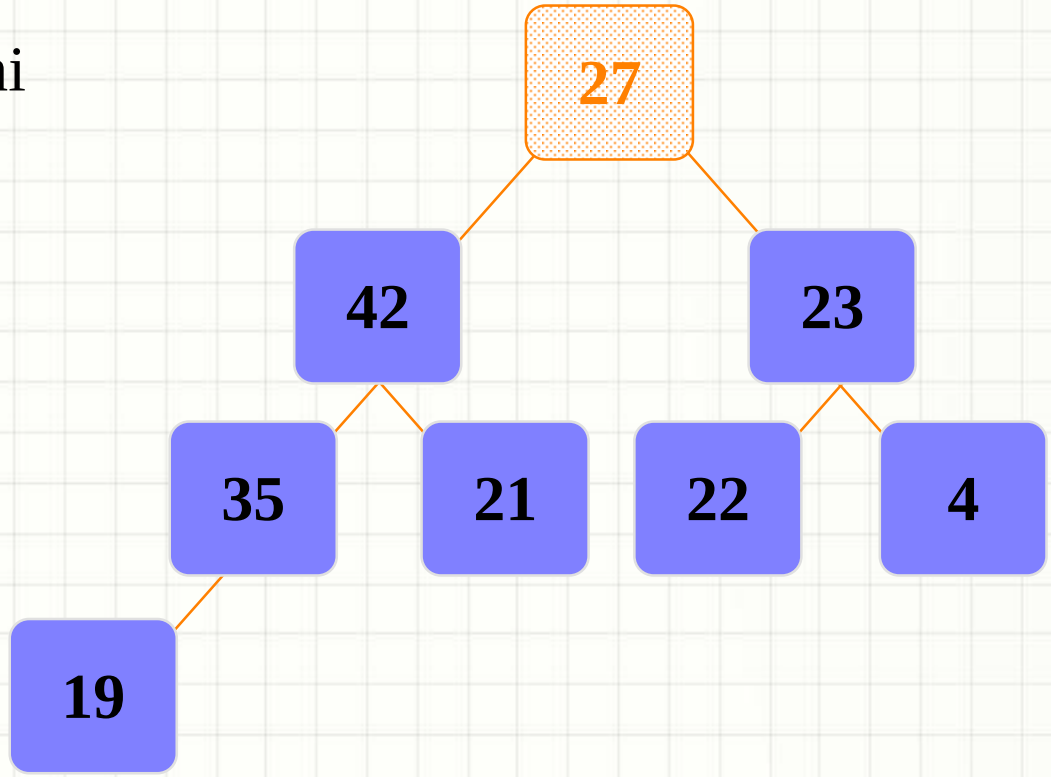
# Max Elemanı Silme (RemoveMax)

- **Adım 1:** Tamamlandı. Ancak, **Heap özelliğinin *tekrar kurulması*** gerekiyor.



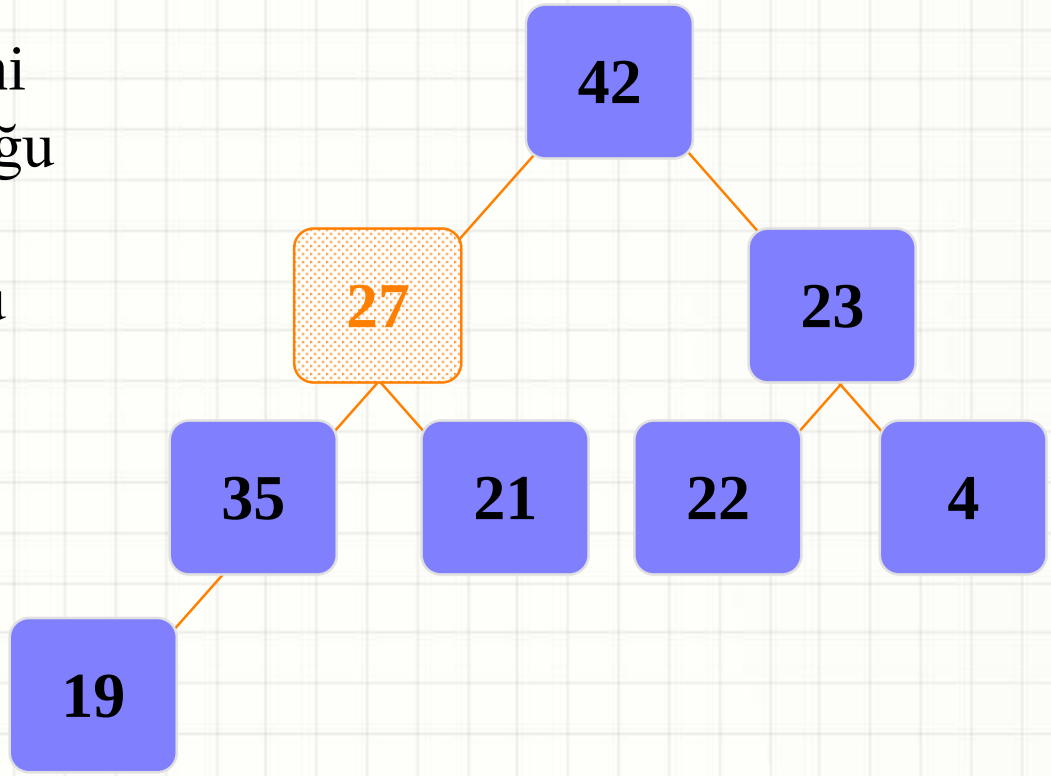
# Max Elemanı Silme (RemoveMax)

- **Adım 2:** Eklenen yeni düğümü, **büyük çocuğu** ile yer değiştirecek şekilde aşağıya doğru taşı.



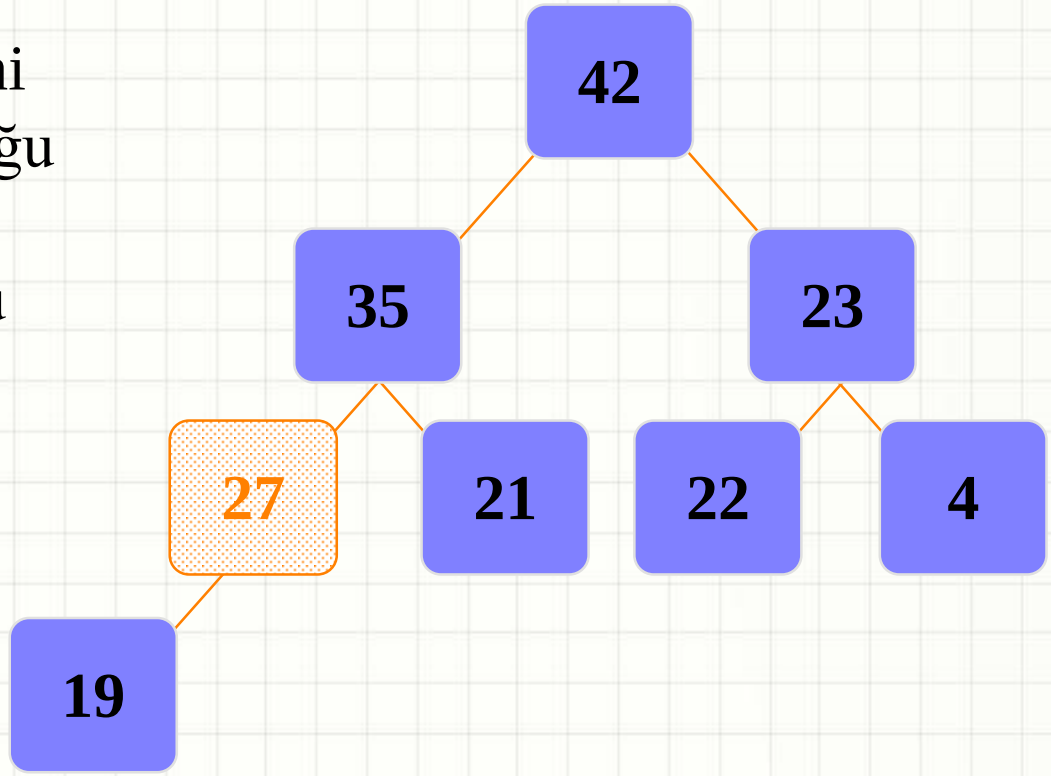
# Max Elemanı Silme (RemoveMax)

- **Adım 2:** Eklenen yeni düğümü, büyük çocuğu ile yer değiştirecek şekilde aşağıya doğru taşı.
- **Adım 3:** Adım 2'yi tekrarla.



# Max Elemanı Silme (RemoveMax)

- **Adım 2:** Eklenen yeni düğümü, büyük çocuğu ile yer değiştirecek şekilde aşağıya doğru taşı.
- **Adım 3:** Adım 2'yi tekrarla.



# Heap RemoveMax Gerçekleştirimi

```
public HeapDugumu RemoveMax() // Remove maximum value HeapDugumu
{
    HeapDugumu root = heapArray[0];
    heapArray[0] = heapArray[--currentSize];
    MoveToDown(0);
    return root;
}
```

# Heap RemoveMax Gerçekleştirimi

```
public void MoveToDown(int index)
{
    int largerChild;
    HeapDugumu top = heapArray[index];
    while (index < currentSize / 2)
    {
        int leftChild = 2 * index + 1;
        int rightChild = leftChild + 1;
        //Find larger child
        if (rightChild < currentSize && heapArray[leftChild].Deger < heapArray[rightChild].Deger)
            largerChild = rightChild;
        else
            largerChild = leftChild;
        if (top.Deger >= heapArray[largerChild].Deger)
            break;
        heapArray[index] = heapArray[largerChild];
        index = largerChild;
    }
    heapArray[index] = top;
}
```

# Heap Ağacı Uygulamaları

- Öncelik Kuyruğu (Priority Queue)
- Heap Sıralama Algoritması (Heap Sort)

# Öncelik Kuyruğu

- Öncelik kuyrukları,
  - **artan** ve **azalan**

  olmak üzere ikiye ayrılırlar.
- Diğer veri yapılarında olduğu gibi kuyrukta bulunan elemanlar, `string veya integer` gibi **basit veri türünde** olabileceği gibi *özelliklere (attribute) sahip* bir **nesne** de olabilir.
- Öncelik kriterinin ne olacağı kuyruktan kuyruğa değişkenlik gösterir.
- Kuyruğa eklenen *elemanın kendisi* veya herhangi bir özelliği, öncelik kriteri olabilir.

# Öncelik Kuyruğu (devam…)

- **Örneğin;** telefon rehberi uygulamasında,
  - Kuyruktaki her eleman soyad, ad, adres ve telefon numarası özelliklerinden oluşmakta ve kuyruk **soyada** göre sıralanmaktadır.
- Öncelik kuyrukları;
  - *Dizi*
  - *Bağlı Liste*
  - ***Binary Heap***

  kullanılarak implemente edilebilir.

# Öncelik Kuyruğu (devam…)

- Öncelik kuyruğunu **gerçekleştirmek için**
- Heap gerçekleştirimindeki:
  - **Insert()** ve
  - **RemoveMax()**

fonksiyonlarını **aynen veya biraz değişiklikle** kullanabiliriz.

# Queue İşlem Karmaşıklığı

| İşlem | Dizi Öncelik Kuyruğu | Heap Öncelik Kuyruğu |
|---|---|---|
| **Insert** | O(n) | O(logn) |
| **Remove** | O(1) | O(logn) |

# Heap Sort

- Heap ağacı kullanarak nasıl sıralama yaparız?

**Adım 1:** Heap ağacı oluştur ve **sıralanmamış** diziyi okuyarak, her bir elemanı ağaca **ekle**.

**Adım 2:**

1. Ağacın *en büyük elemanını* oku.
2. Elemanı yeni bir diziye ata.
3. Ağacı yeniden oluştur.
4. **İlk 3 adımı** ağaçta eleman kalmayana kadar tekrarla.

# Heap Sort (devam…)

```
public class HeapSort
{
    private int[] dizi;
    1 reference
    public HeapSort(int[] dizi)
    {
        this.dizi = dizi;
    }

    1 reference
    public int[] Sort()
    {
        Heap h = new Heap(dizi.Length);
        int[] sorted = new int[dizi.Length];
        //Heap Ağacı Oluştur
        foreach (var item in dizi)
            h.Insert(item);
        int i = 0;
        //Ağaçtaki maksimum elemanı al ve yeni diziye ekle
        while (!h.IsEmpty())
            sorted[i++] = h.RemoveMax().Deger;
        return sorted;
    }

}
```

# Heap Sort (devam…)

- Big O karmaşıklığı nedir?

**O(nlogn)**