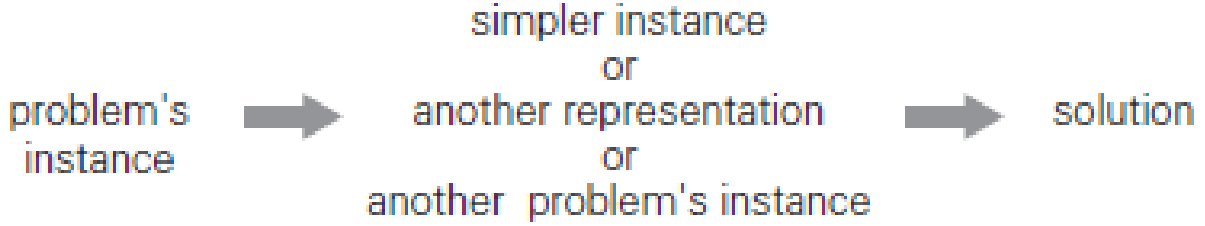
YZM 3207- ALGORİTMA ANALİZİ VE TASARIM

DERS#7: DÖNÜŞTÜR VE FETHET YÖNTEMİ

Dönüştür ve Fethet Algoritmaları

- Bu algoritma teknikleri problemi dönüştürerek çözüm arar.
  - **Problemi daha basit, daha uygun bir duruma dönüştürmek, durumu basitleştirme** (*instance simplification*)
  - **Başka bir gösterime dönüştürmek** (*representation change*)
  - **Algoritması bilinen bir probleme dönüştürmek** (*problem reduction*)



Durumu Basitleştirme - Önsıralama

- Listeler ile ilgili birçok problem liste sıralı ise daha kolay çözülür.
  - Arama
  - Ortanca bulma
  - Elemanların farklı olup olmadığını belirleme
  - Topolojik sıralama bir çok problemin çözülmesini kolaylaştırır
  - Birçok geometri probleminin çözümünü ön sıralama kolaylaştırır

Durumu Basitleştirme - Önsıralama

- **Orta seviye sıralama algoritmaları**
  - Selection, bubble, insertion
    - En kötü ve ortalama durumlarda üssel
- **Gelişmiş sıralama algoritmaları**
  - Merge, Quick
    - $\Theta(n \log n)$,

Durum Basitleştirme

- Dizi elemanları farklı mı?
  - Kaba kuvvetle çözüm n^2
  - Diziyi sıralayıp kontrol edersek?

ALGORITHM *PresortElementUniqueness*$(A[0..n-1])$
//Solves the element uniqueness problem by sorting the array first
//Input: An array $A[0..n-1]$ of orderable elements
//Output: Returns "true" if A has no equal elements, "false" otherwise
sort the array A
for $i \leftarrow 0$ **to** $n-2$ **do**
  if $A[i] = A[i+1]$ **return false**
return true

  - Algoritmanın çalışma zamanı sıralama işleminin ve kontrol işleminin toplamı
  - Dizi sıralı olduğu için kontrol işlemi n-1 kez yapılıyor
  - Algoritma etkinliğini sıralama belirleyecek
    - Uygun bir sıralama algoritması seçilirse (n log n ?)

$$T(n) = T_{sort}(n) + T_{scan}(n) \in \Theta(n \log n) + \Theta(n) = \Theta(n \log n)$$

Durum Basitleştirme

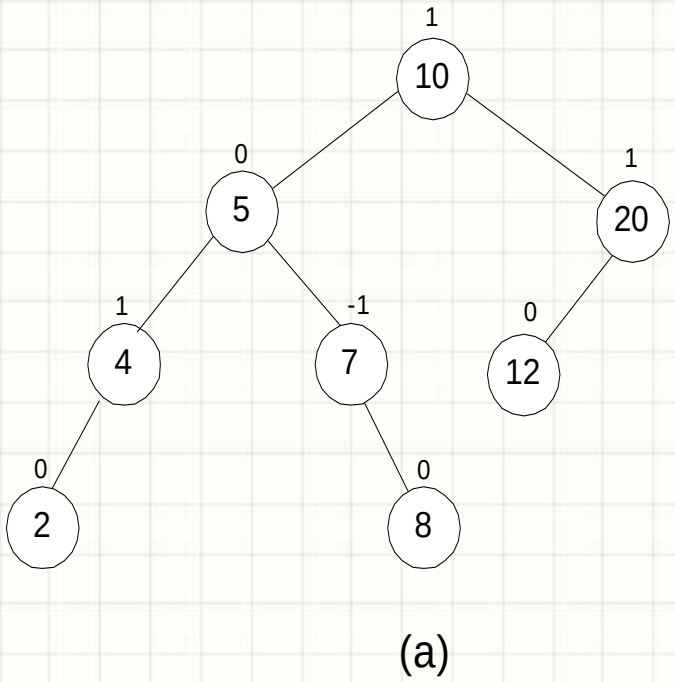
- Arama Problemi
  - Diziyi sırala
  - İkili arama uygula

$$T(n) = T_{sort}(n) + T_{search}(n) = \Theta(n \log n) + \Theta(\log n) = \Theta(n \log n)$$

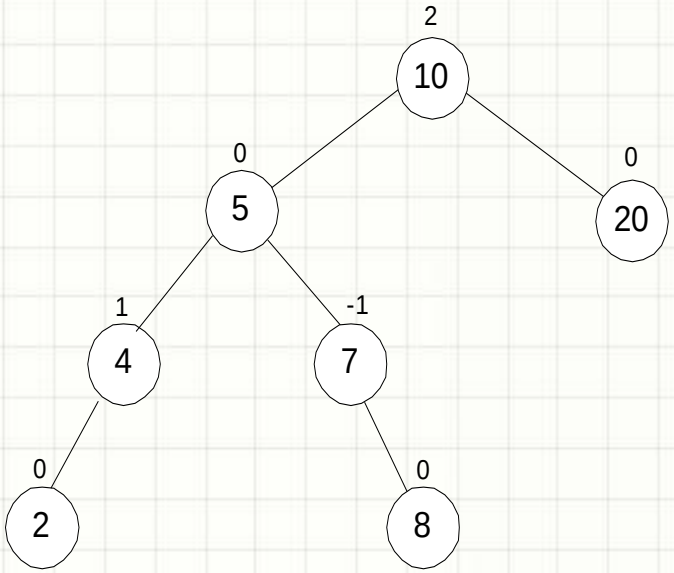
AVL Ağaçları

- AVL Ağacı: AVL Ağaçları sürekli olarak dengeli olan ikili arama ağaçlarıdır.
- Algoritmada basitçe, bir düğümün kolları arasındaki derinlik farkı 2 ise bu durumda dengeleme işlemi yapılır.
- Eğer fark 2'den az ise (yani 1 veya 0) ise bu durumda bir dengeleme işlemine gerek yoktur.

AVL Ağaçları

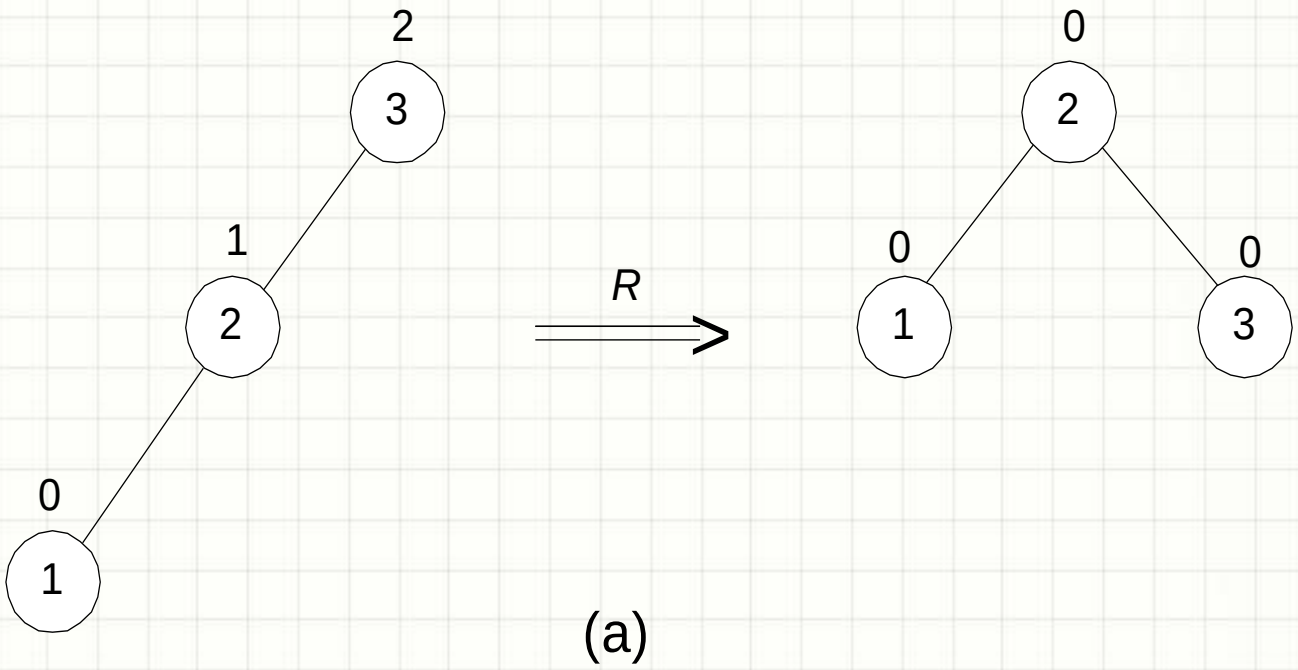


(a)



(b)

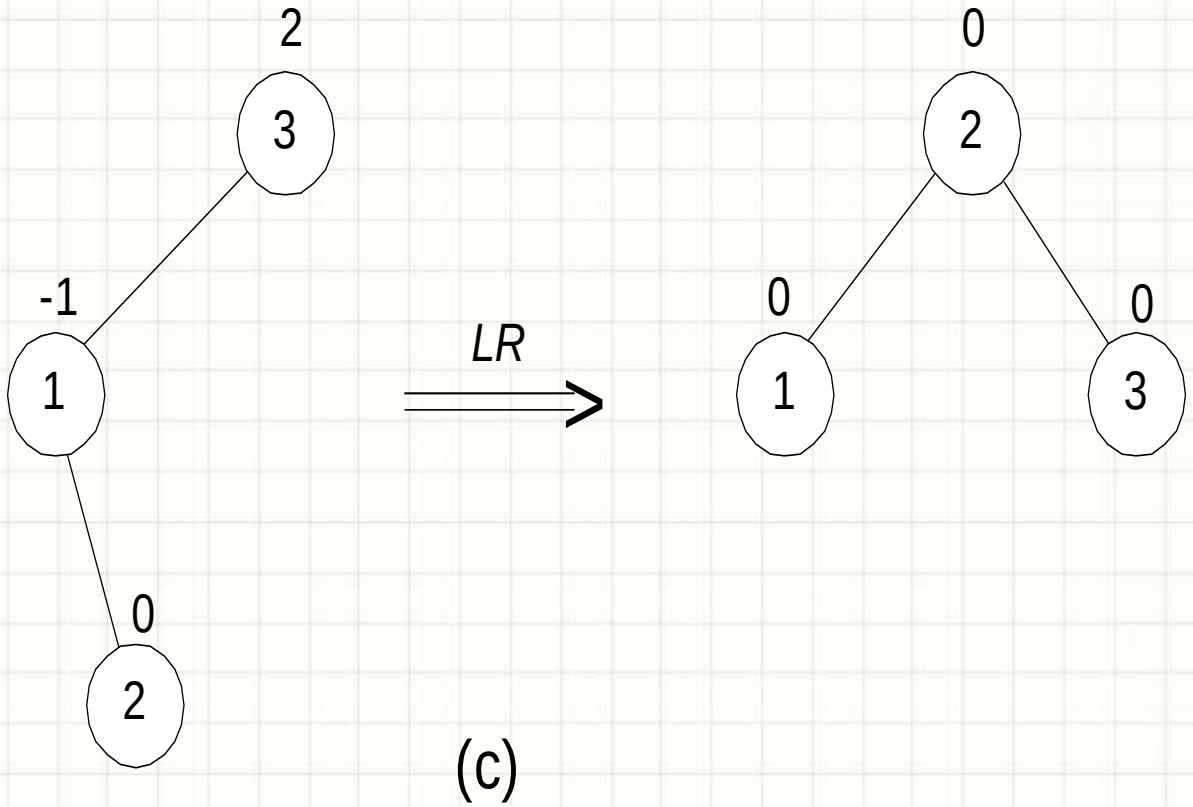
Dengeleme--- Tek Sağa Çevirme



(a)

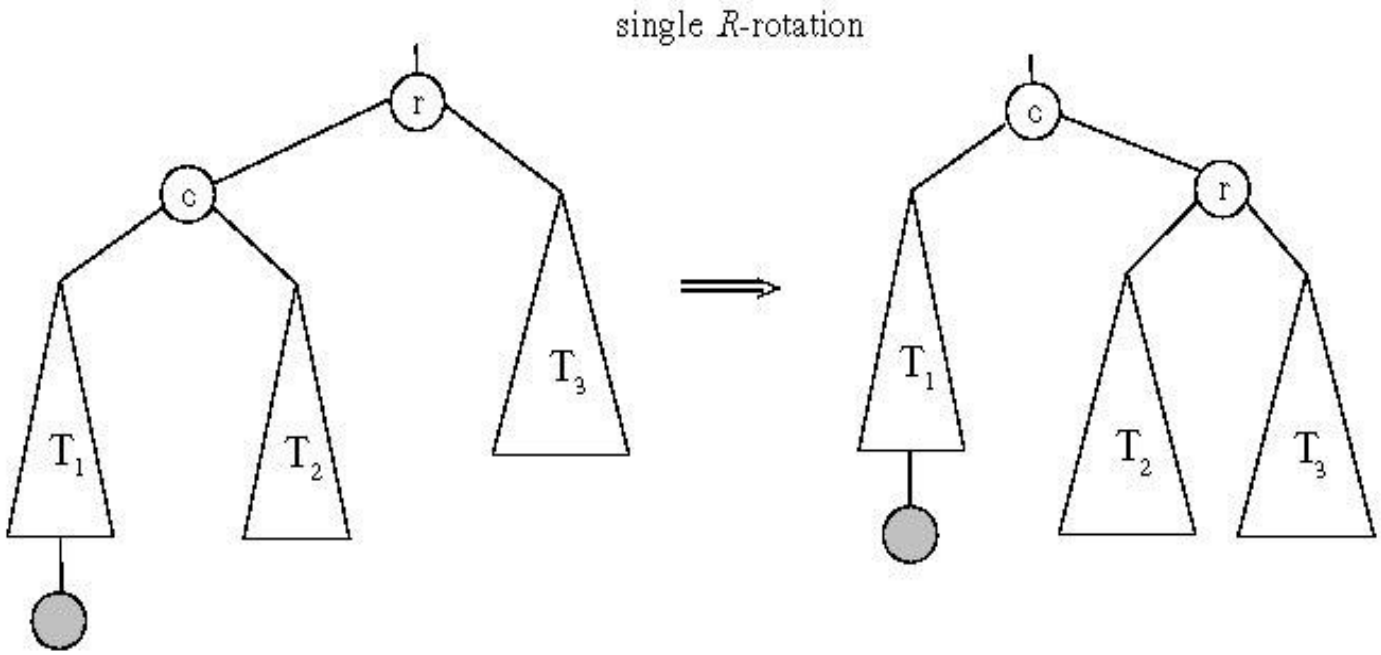
R Rotasyonu

Dengeleme--- Sol ve Sağa Çevirme

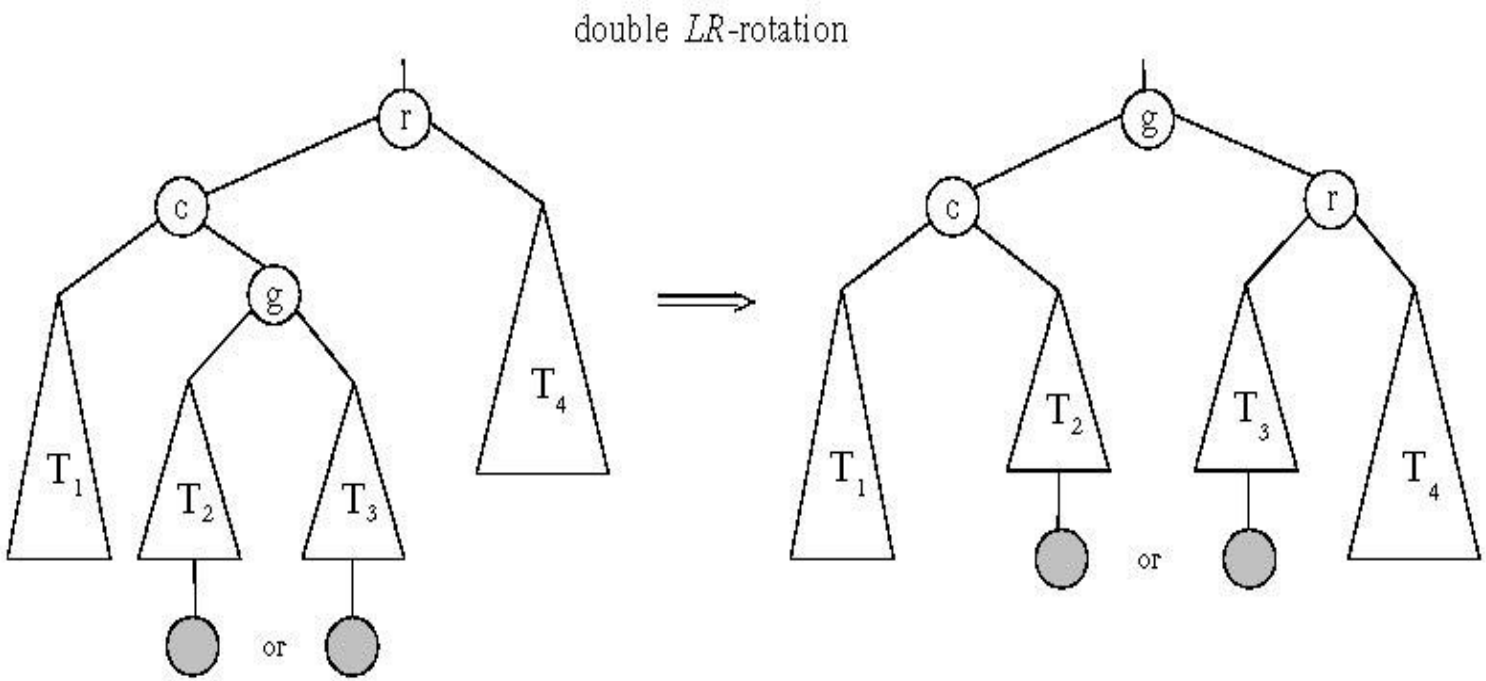


LR Rotasyonu

Dengeleme--- Tek Sağa Çevirme

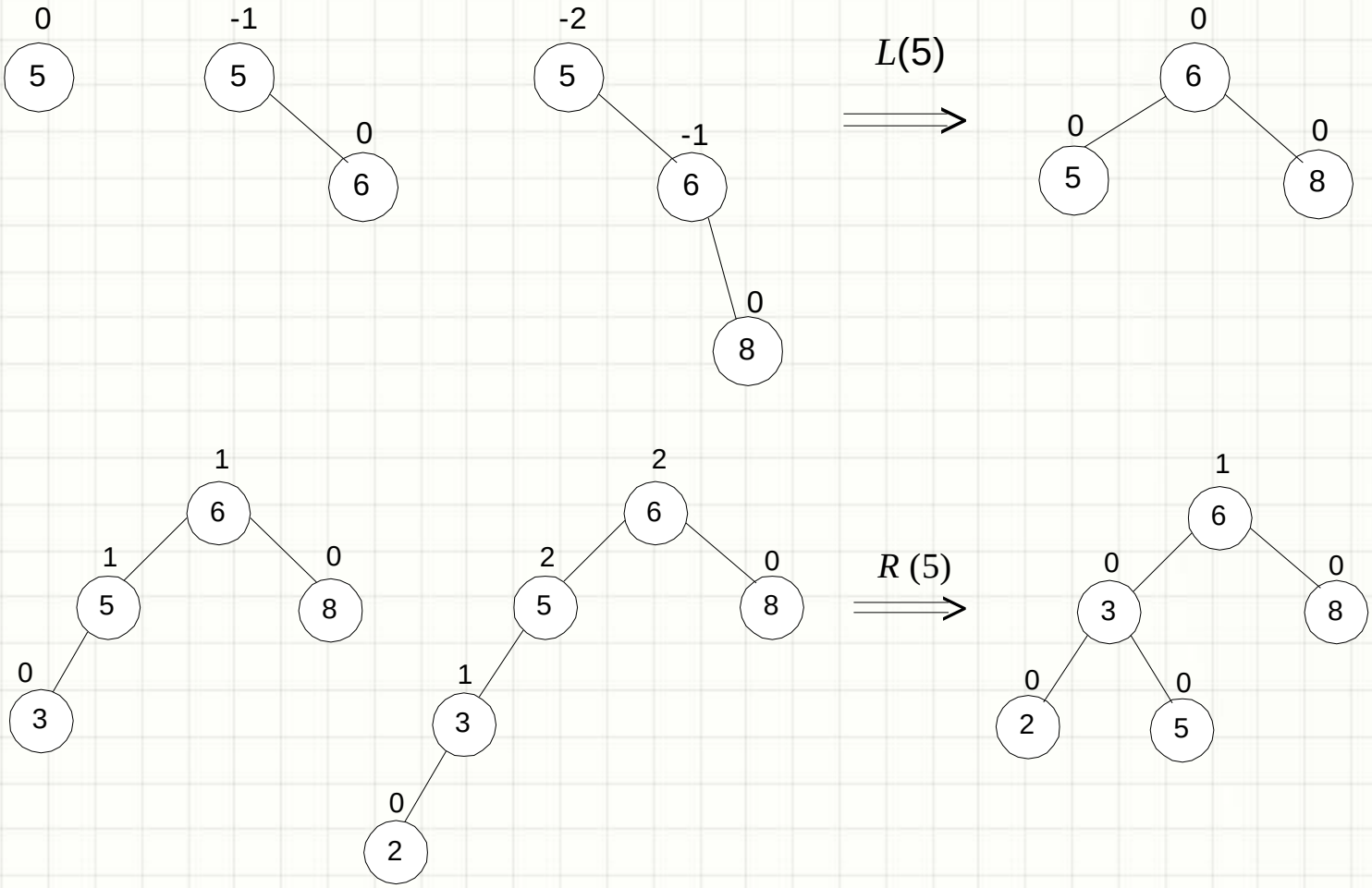


Dengeleme--- Sol ve Sağa Çevirme

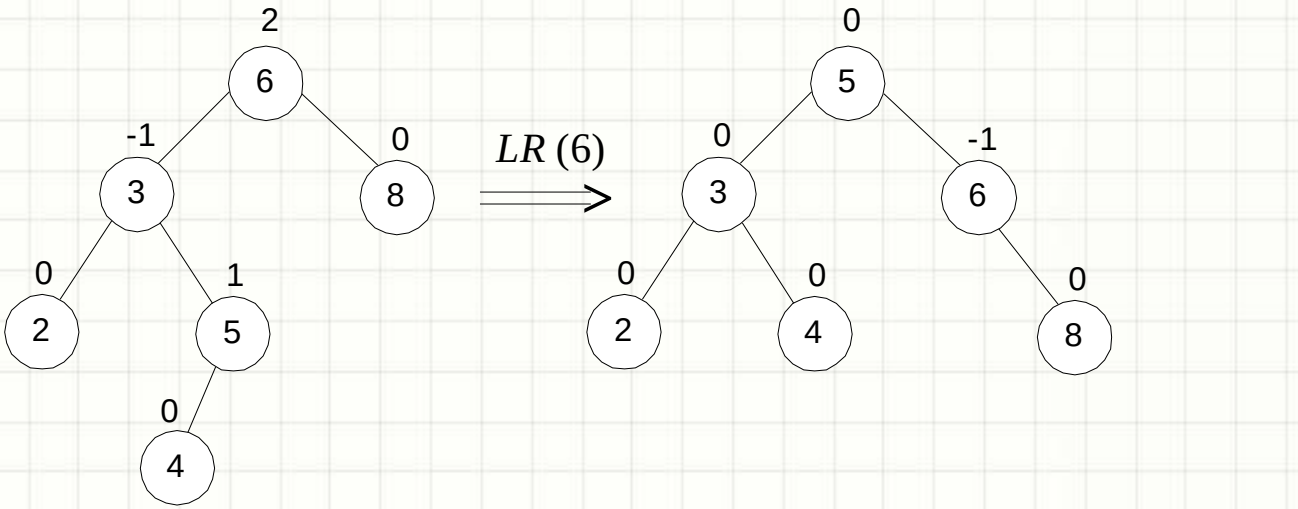


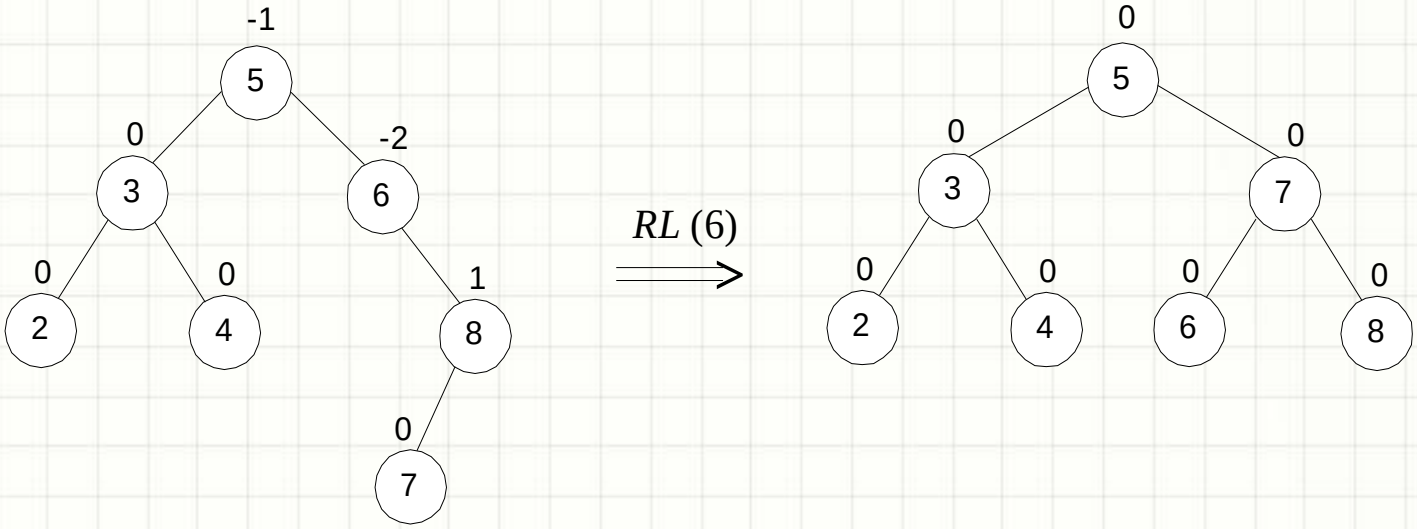
AVL Ağacı Oluşturma

5, 6, 8, 3, 2, 4, 7



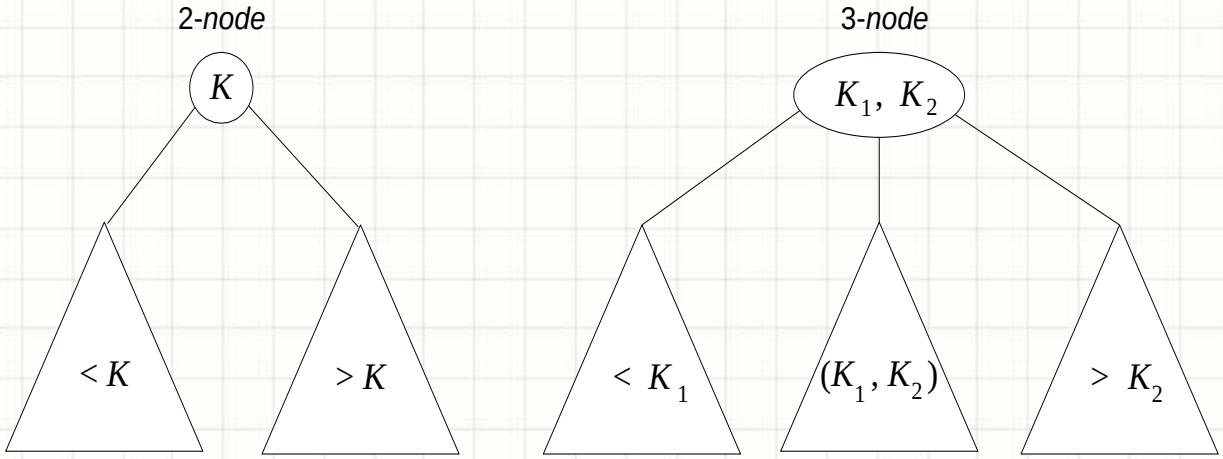
AVL Ağacı Oluşturma (devam)





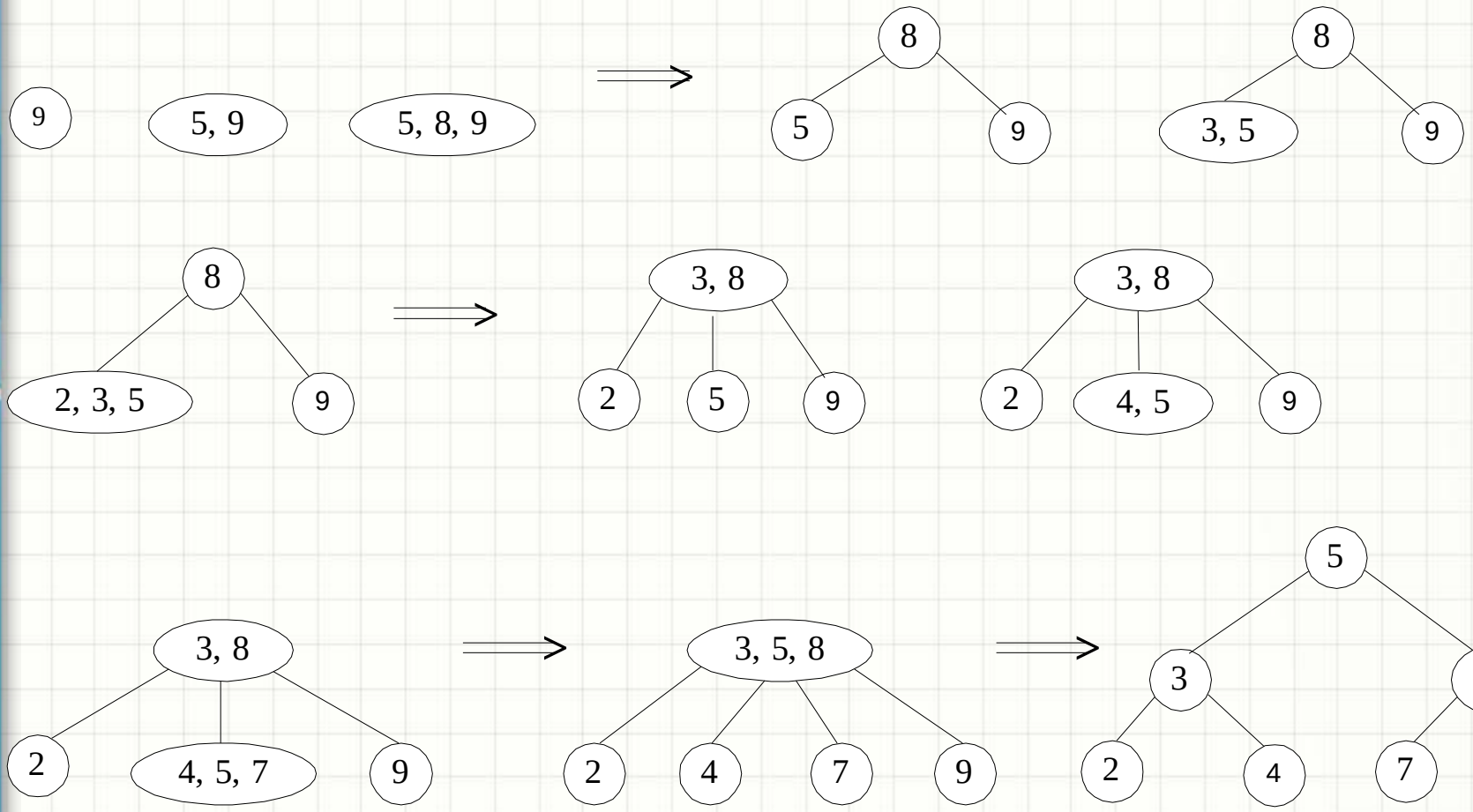
2-3 Ağaçları

- Özel bir ağaç yapısıdır ve amaç ağacı sürekli olarak dengeli tutmaktır.
- **2 düğümleri (2 nodes) :** 2 adet çocuğu ve bir veri elemanı bulunan düğüm yapısıdır.
- **3 düğümleri (3 nodes) :** 3 adet çocuğu ve iki veri elemanı bulunan düğüm yapısıdır.



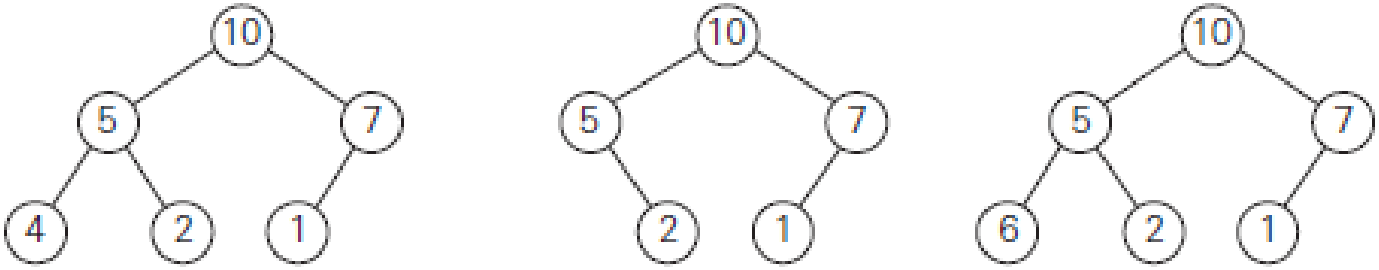
2-3 Ağaçları --- Örnek

9, 5, 8, 3, 2, 4, 7

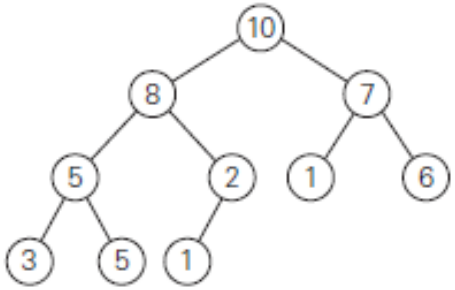


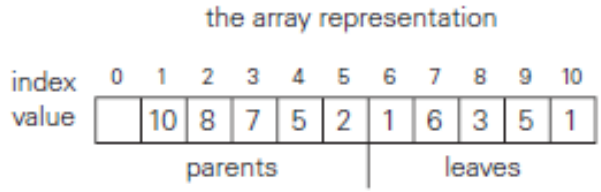
Heap Sıralama

- Heap (Yığın Ağacı)
  - Kurallı yığın
  - Öncelik (priority) kuyrukları
- Bir ikili ağacın heap olabilmesi için
  - **Şekil özelliği**
    - Tüm düzeylerde eşit olması gerekir
    - En sağdaki yaprak istisna olabilir
  - **Ebeveyn baskınlığı**
    - Bir düğümün değeri çocuk düğümlerden büyük veya eşit olmalıdır



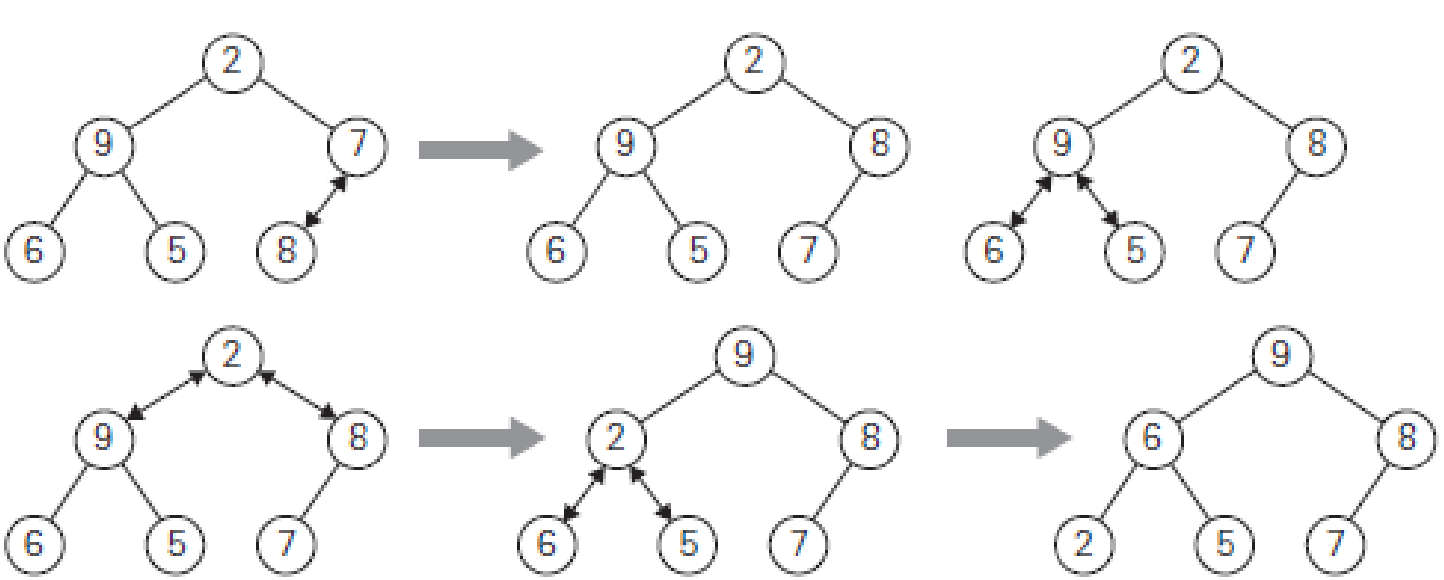
Heap Oluşturma





- **Heap Kuralları**
  - *n* adet düğümü olan bir tam ikili ağacın yüksekliği $\log_2 n$'dir
  - Heap kökü her zaman en büyük elemanı tutar
  - Bir heap düğümünün alt düğümleride heapdir
  - Bir heap dizi şeklinde gösterilecekse;
    - Yukarıdan aşağı ve soldan sağa doğru yerleştirilmeli
    - 0. eleman boş bırakılmalı veya en yüksek değerin yerini göstermeli
    - İlk **n/2** elemanda ebeveyn, son **n/2** elemanda çocuk düğüm değerleri olmalı
    - **i** pozisyonundaki ebeveynin çocukları **2i** ve **2i+1** pozisyonlarında olmalı
    - **i** pozisyonundaki çocuğun ebeveni **i/2** pozisyonunda olmalı

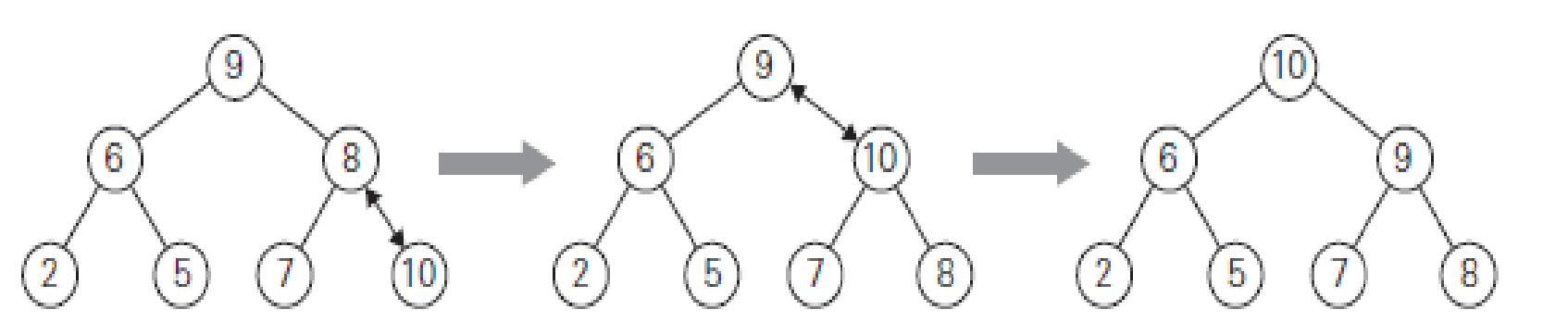
Heap Oluşturma



- **Aşağıdan yukarıya oluşturma**
  - Dizi elemanları ikili ağaca yerleştirilir
  - Son ebeveynden başlanılarak ebeveyn değeri ile çocuk değeri karşılaştırılır
    - Çocuk değeri büyük ise yer değiştirilir
    - Kök ebeveyne kadar devam eder

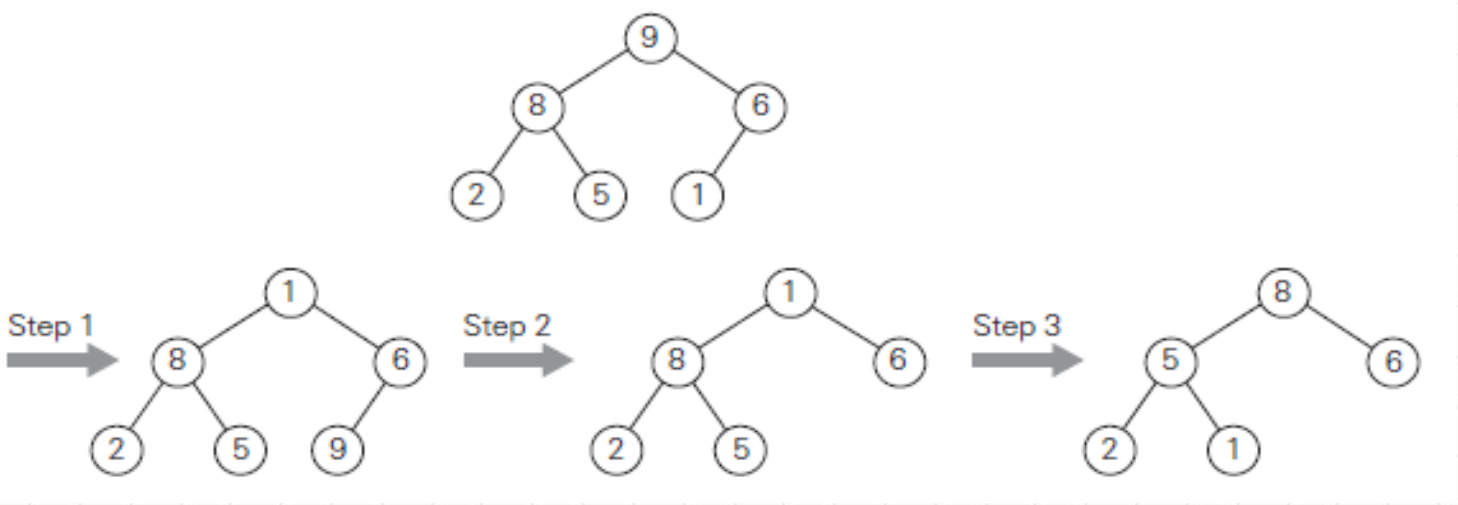
Heap Oluşturma

- Yukarıdan aşağı oluşturma
  - Her elemanın sırayla ağaca eklenmesi
  - Her eklemede heap yeniden düzenlenir



Heap’ten Eleman Silme

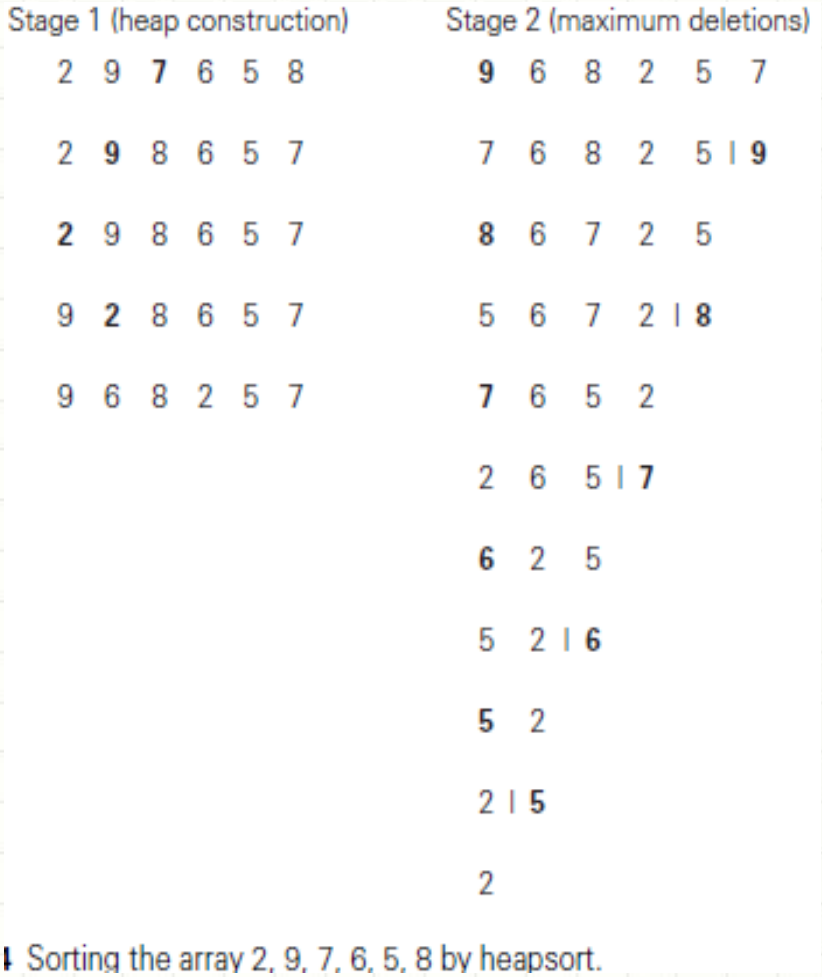
- Heap’ten eleman silinmesi
  - Silinecek eleman ağacın son elemanıyla yer değiştirir
  - Eleman silinir, ağaç boyutu 1 azaltılır
  - Ağaç tekrar düzenlenir



Heap Sıralama

- İki aşamadan oluşan bir sıralama algoritması
  - Diziden bir heap oluştur
  - Kök düğümü (n-1) kez sil
  - Ters sıralı olarak dizi düzenlenmiş olur

$$O(n) + O(n \log n) = O(n \log n)$$



Sorting the array 2, 9, 7, 6, 5, 8 by heapsort.