Bölüm 1: Giriş

Algoritmalar

Algoritma

- Bir problemi çözmek veya görevi yerine getirmek için,
  - adım adım yönergeler dizisidir.
- Her adım belirli, açık ve anlaşılır olmalıdır.
- Girdi verileri alınır, işlemler yapılır, çıktı elde edilir.
- Algoritma, sınırlı sayıda adımda sonlanmalıdır.

Çay Hazırlama Algoritması

Girdi: Su, Çay yaprakları, Şeker (isteğe bağlı), Süt (isteğe bağlı)

Çıktı: Çay

1. Suyu kaynat.
2. Kaynamış suyu, demliğe koy.
3. Demliğin içine çay yapraklarını ekle.
4. Çayın demlenmesini bekle.
5. Demlenen çayı fincana süzerek koy.
6. İsteğe bağlı olarak şeker ve süt ekle.

Ortalama Bulma Algoritması

Girdi: Sayı1, Sayı2, Sayı3

Çıktı: Ortalama

1. Toplam = Sayı1 + Sayı2 + Sayı3
2. Ortalama = Toplam / 3
3. Ortalama değerini ekrana yazdır.

Ortalama Bulma

```
sayi1 = float(input("Birinci sayıyı girin: "))
sayi2 = float(input("İkinci sayıyı girin: "))
sayi3 = float(input("Üçüncü sayıyı girin: "))
toplam = sayi1 + sayi2 + sayi3
ortalama = toplam / 3
print(f"Girilen sayıların ortalaması: {ortalama}")
```

Ortalama Bulma

```
Scanner scanner = new Scanner(System.in);
System.out.print("Birinci sayıyı girin: ");
double sayi1 = scanner.nextDouble();
System.out.print("İkinci sayıyı girin: ");
double sayi2 = scanner.nextDouble();
System.out.print("Üçüncü sayıyı girin: ");
double sayi3 = scanner.nextDouble();
double toplam = sayi1 + sayi2 + sayi3;
double ortalama = toplam / 3;
System.out.println("Sayıların ortalaması: " + ortalama);
scanner.close();
```

Algoritma Analizi

- Bir algoritmanın performansını anlamak ve değerlendirmek için kullanılır.
- İşlemci ve bellek gibi kaynakların verimli kullanılmasını sağlar.
- Zaman Analizi:
  - Algoritmanın çalışma süresini ölçer.
  - Genellikle *Big-O* notasyonu ile gösterilir.
- Bellek Analizi:
  - Algoritmanın bellek kullanımını değerlendirir.
  - Bellek karmaşıklığını gösterir.
- *Big-O notasyonu*, algoritmaların en kötü durum performansını ifade eder.

Asimptotik Gösterim Türleri

- *Omega* Ω Gösterimi
- *Big* O Gösterimi
- *Theta* Θ Gösterimi

Omega (Ω) Gösterimi

- Bir algoritmanın *en iyi durum* çalışma süresini ifade eder.
- Çalışma süresinin alt sınırını ifade eder.
- **Örnek**:
  - Bir algoritmanın en iyi durum çalışma süresi *100 saniye* ise,
  - Algoritmanın çalışma süresi *100 saniyeden* daha uzun sürebilir.
  - Ancak, geçen süre *100 saniyeden* kısa olamaz.

Büyük (Big) O Gösterimi

- Bir algoritmanın *en kötü durum* çalışma süresini ifade eder.
- Çalışma süresinin üst sınırını ifade eder.
- **Örnek**:
  - Bir algoritmanın en kötü durum çalışma süresi *100 saniye* ise,
  - Algoritma çalışma süresi *100 saniyeden* daha kısa sürebilir.
  - Ancak, geçen süre *100 saniyeden* uzun olamaz.

Theta (Θ) Gösterimi

- Algoritmanın hem *en kötü* hem *en iyi durum* çalışma süresini ifade eder.
- Örneğin bir sıralama algoritması,
  - En kötü durum karmaşıklığı O(*n log n*)
  - En iyi durum karmaşıklığı Ω(*n*) ise,
  - Çalışma süresi Θ(*n log n*) ile gösterilir.

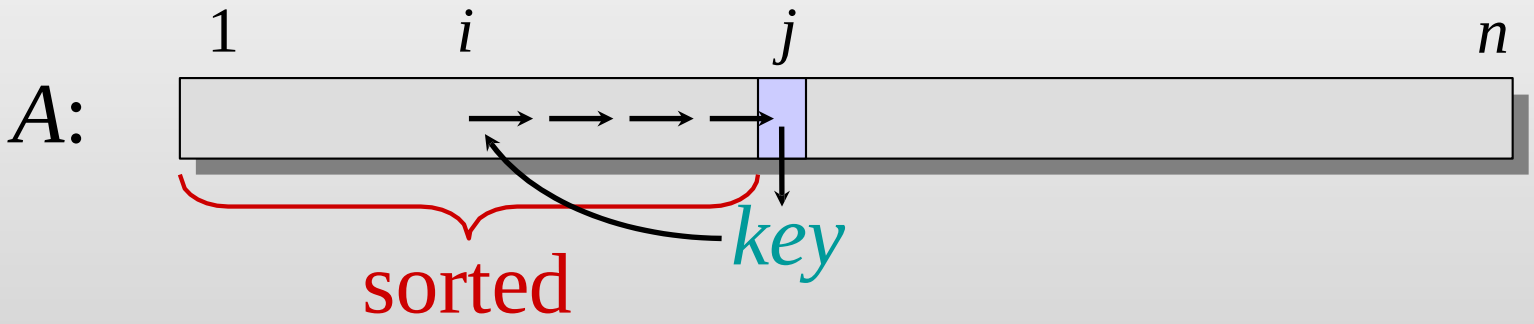
Sıralama Problemi

- **Girdi**: $<a_1, a_2, \ldots, a_n>$ sayı dizisi.
- **Çıktı**: $<a'_1, a'_2, \ldots, a'_n>$ sayı dizisi $(a'_1 \leq a'_2 \leq \ldots \leq a'_n)$

- **Örnek**:
  - Girdi: 8 2 4 9 3 6
  - Çıktı: 2 3 4 6 8 9

Araya Sokarak Sıralama

- Insertion sort
- Algoritma, bir diziyi sıralamak için iç içe geçmiş iki döngü kullanır.
- Dıştaki döngü, dizinin her elemanını bir kere geçer.
- İçteki döngü ise, mevcut elemanı uygun konuma yerleştirir.
- Zaman karmaşıklığı $O(n^2)$.



Araya Sokarak Sıralama

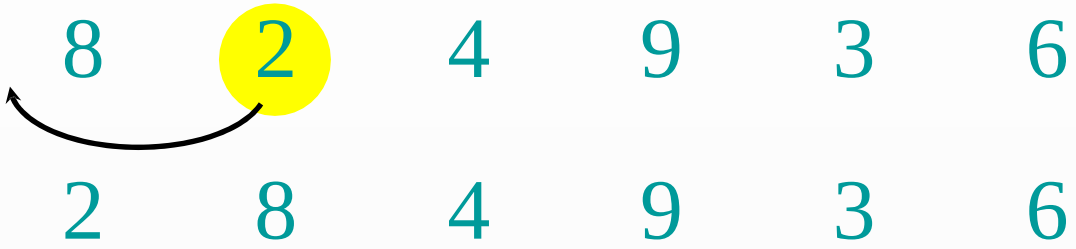
```
def Sirala(A):
    for i in range(1, len(A)):
        key = A[i]
        j = i - 1
        // A[0..i-1] alt dizisi sıralı.
        // Key, alt dizideki uygun konuma yerleştirilir.
        while j >= 0 and A[j] > key:
            A[j + 1] = A[j]
            j = j - 1
        A[j + 1] = key
```

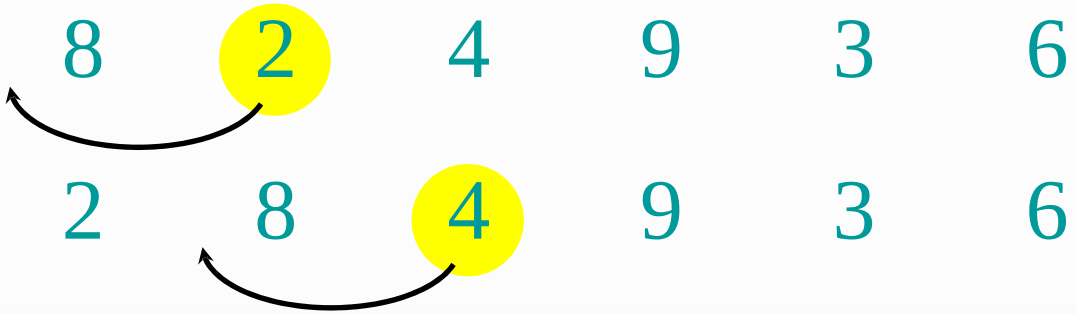
Eklemeli Sıralama

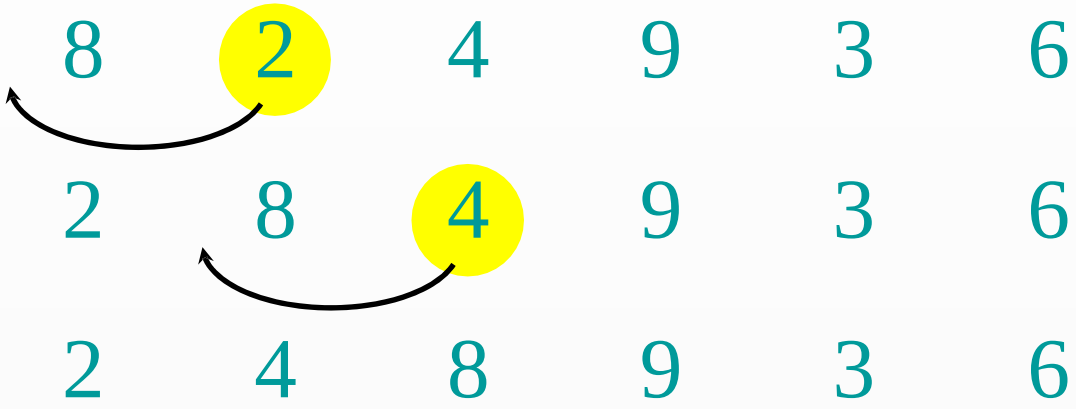
```
public void Sirala(int[] A) {
    for (int i = 1; i < A.length; ++i) {
        int key = A[i];
        int j = i - 1;
        while (j >= 0 && A[j] > key) {
            A[j + 1] = A[j];
            j = j - 1;
        }
        A[j + 1] = key;
    }
}
```

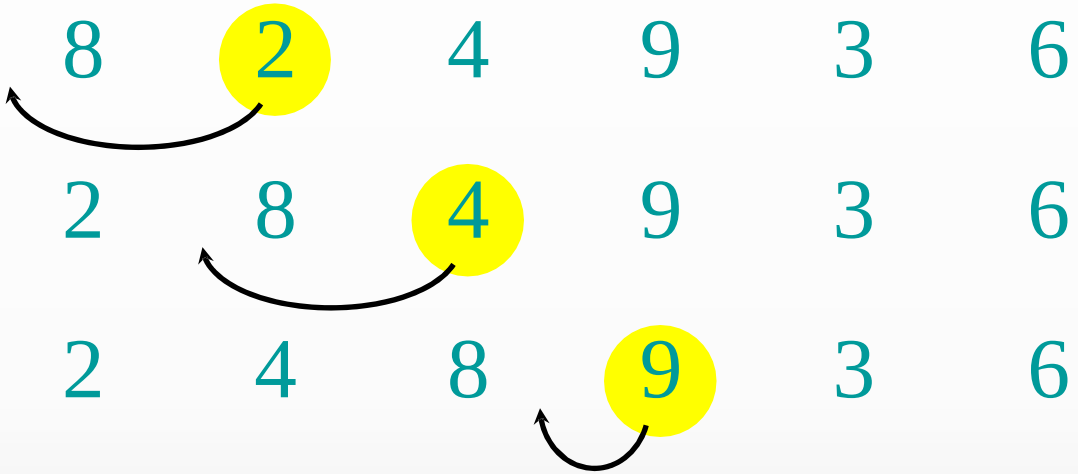
Örnek

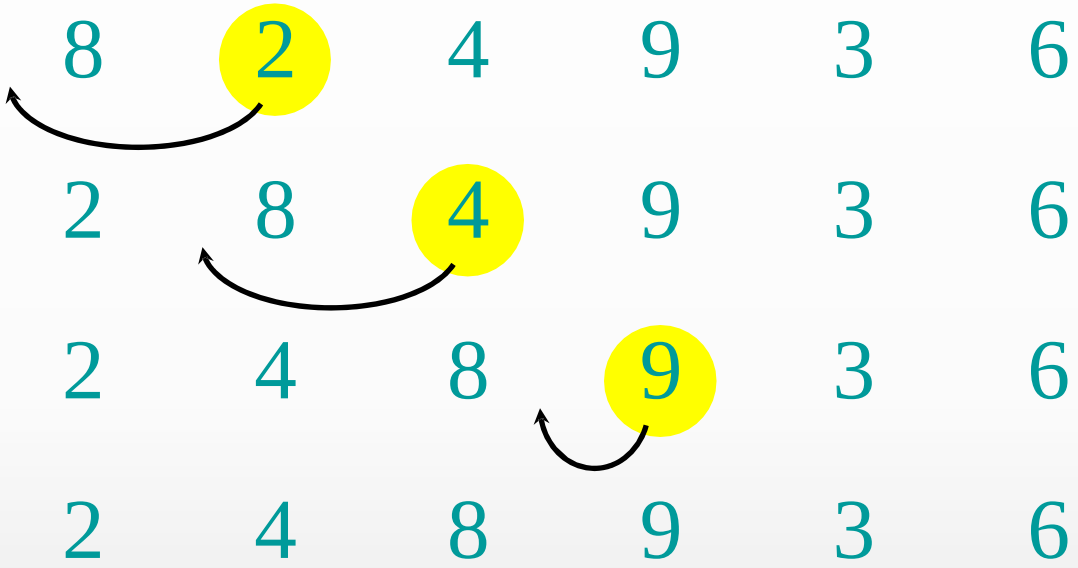
8 2 4 9 3 6

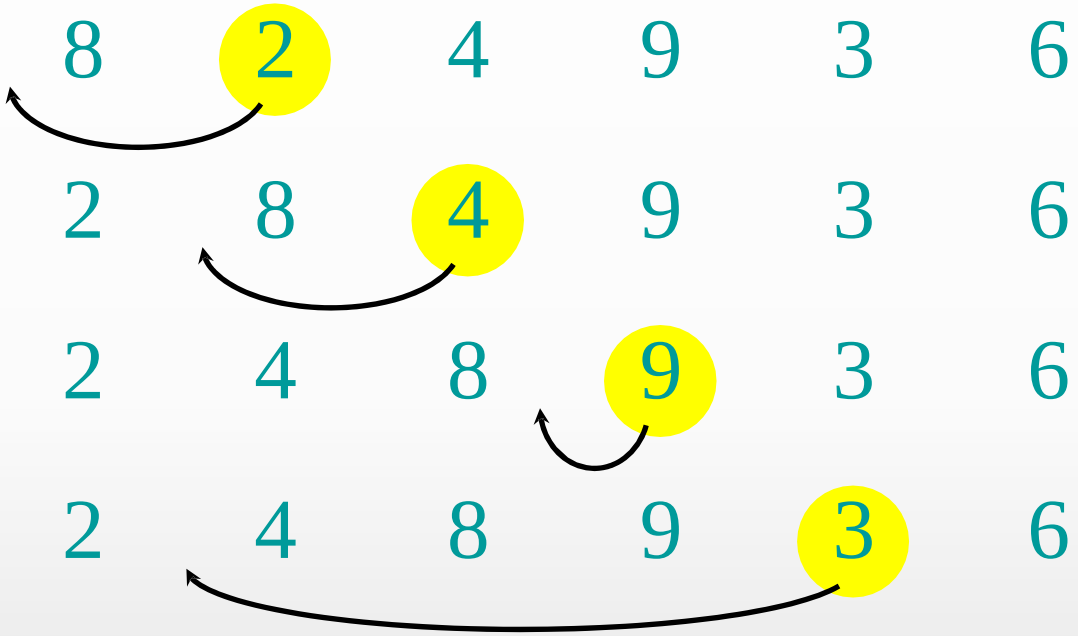


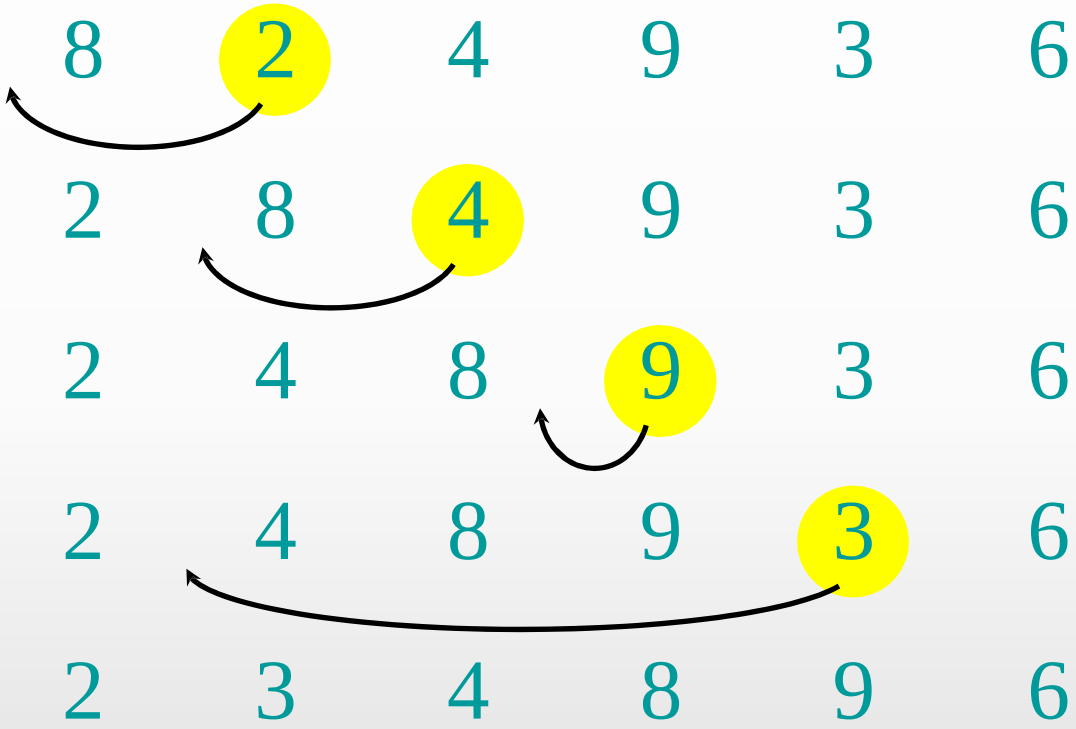


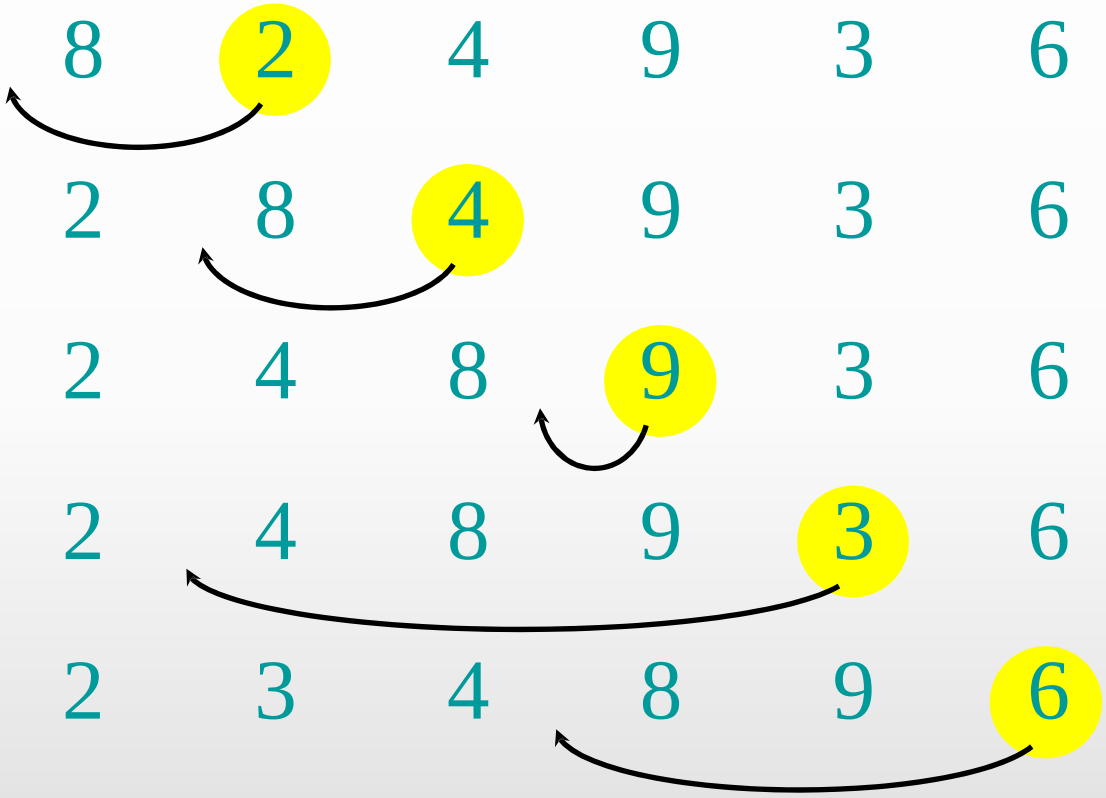


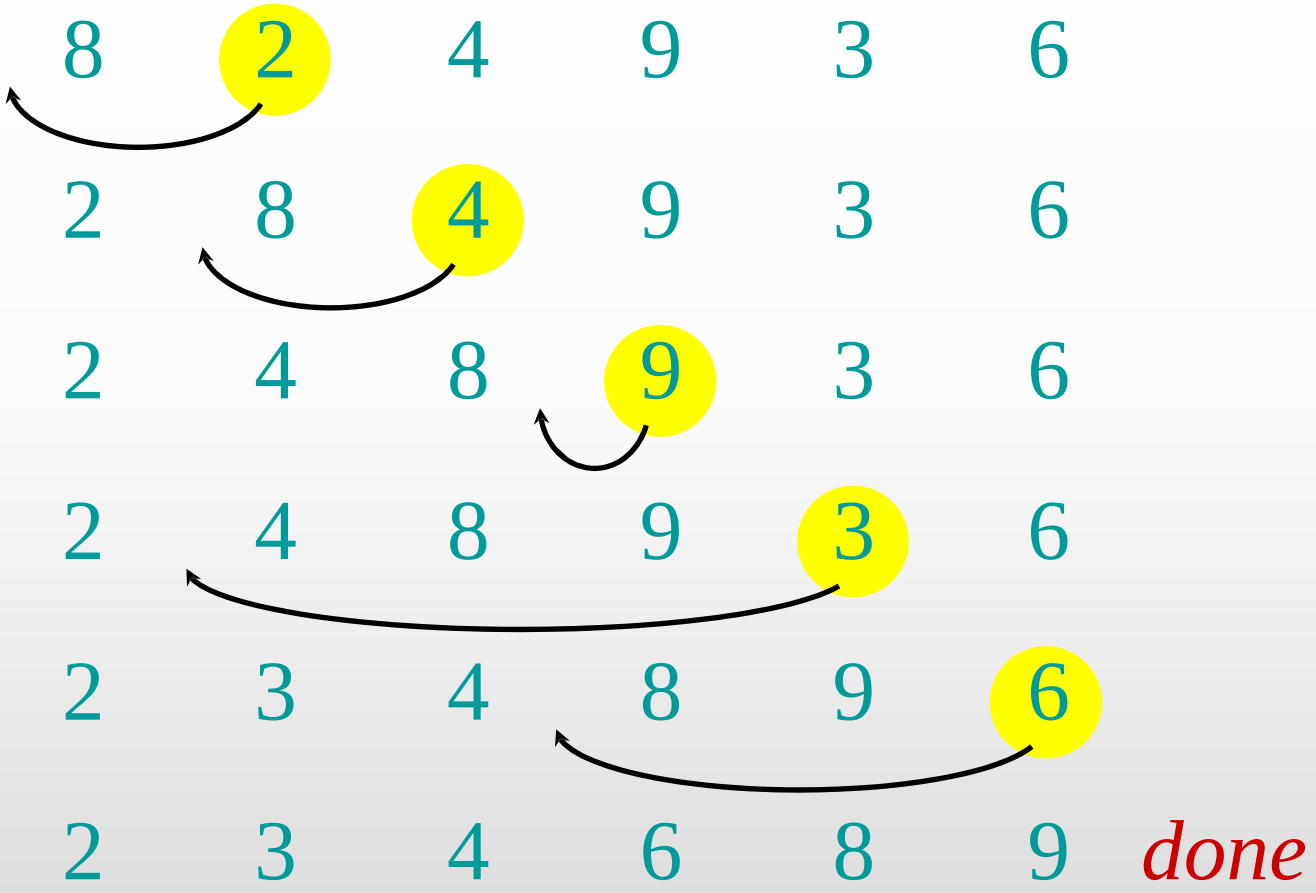












Çalışma Süresi

- Girdi durumuna bağlı.
  - Dizi zaten sıralı ise, sıralamak kolay. ☺
- Girdi boyutuna bağlı.
  - Dizi az elemanlı ise, sıralamak kısa sürer. ☺
- Çalışma süresi olarak en kötü durum ele alınır.
  - Çünkü herkes garanti ister. ☺

Analiz Türleri

- Kötü Durum: (*usually*)
  - *T(n)* = *n* boyutlu diziyi sıralamak için gereken maksimum süre
- Orta Durum: (*sometimes*)
  - *T(n)* = *n* boyutlu diziyi sıralamak için beklenen süre
- İyi Durum: (*bogus*)
  - Özel veya hileli bir girdiyi sıralamak için geçen süre

Makineden Bağımsız Çalışma Süresi

- Araya sokarak sıralamanın en kötü durum zaman karmaşıklığı: $O(n^2)$
  - Girdi *tersten sıralı* ise gerçekleşir.
- Bilgisayarın hızına göre çalışma süresi değişebilir.
  - *Asimptotik analiz*, büyüme oranına odaklanır.
  - Makineye bağlı *sabit faktörler* göz ardı edilir.
- Analiz için $T(n)$'nin büyüme oranına bakılır.
  - Girdi boyutu n sonsuza gittikçe, $(n \to \infty)$
  - Yalnızca, en önemli (*üssü büyük olan*) terim değerlendirilir.
- $T(n) = 3n^2 + 2n + 5 = O(n^2)$

Θ-notation

- $\Theta(g(n)) = \{\ f\ (n)$: tüm $n \geq n_0$ değerleri için $0 \leq c_1\, g(n) \leq f\ (n) \leq c_2\, g(n)$ şartını sağlayan c_1, c_2, ve n_0 pozitif katsayılar vardır. $\}$
- Örnek:
  - $T(n) = 3n^3 + 90n^2 - 5n + 6046$
- Düşük kuvvetli terimleri sil.
  - $T(n) = 3n^3$
- Katsayıları kaldır.
  - $T(n) = \Theta(n^3)$

Birleştirerek Sıralama

MergeSort A[1 . . n]

n = 1 ise işlem tamam.

liste1 = MergeSort A[1 . . n/2]

liste2 = MergeSort A[n/2 +1 . . n]

2 sıralı listeyi birleştir. (*merge*)

Birleştirerek Sıralama

MergeSort A[1 . . n] *T(n)*

n = 1 ise işlem tamam.

liste1 = MergeSort A[1 . . n/2]

liste2 = MergeSort A[n/2 +1 . . n]

2 sıralı listeyi birleştir. (*Merge*)

Birleştirerek Sıralama

MergeSort A[1 . . n] *T(n)*

n = 1 ise işlem tamam. *Θ(1)*

liste1 = MergeSort A[1 . . n/2]

liste2 = MergeSort A[n/2 +1 . . n]

2 sıralı listeyi birleştir. (*Merge*)

Birleştirerek Sıralama

MergeSort A[1 . . n] *T(n)*

n = 1 ise işlem tamam. *Θ(1)*

liste1 = MergeSort A[1 . . n/2] *T(n/2)*

liste2 = MergeSort A[n/2 +1 . . n]

2 sıralı listeyi birleştir. (*Merge*)

Birleştirerek Sıralama

MergeSort A[1 . . n] *T(n)*

n = 1 ise işlem tamam. *Θ(1)*

liste1 = MergeSort A[1 . . n/2] *T(n/2)*

liste2 = MergeSort A[n/2 +1 . . n] *T(n/2)*

2 sıralı listeyi birleştir. (*Merge*)

Birleştirerek Sıralama

MergeSort A[1 . . n] *T(n)*

n = 1 ise işlem tamam. *Θ(1)*

liste1 = MergeSort A[1 . . n/2] *T(n/2)*

liste2 = MergeSort A[n/2 +1 . . n] *T(n/2)*

2 sıralı listeyi birleştir. (*Merge*) *Θ(n)*

Birleştirme İşlemi (Merge)

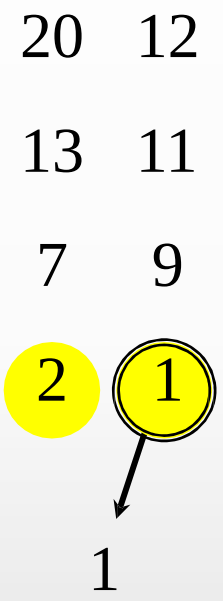
20 12

13 11

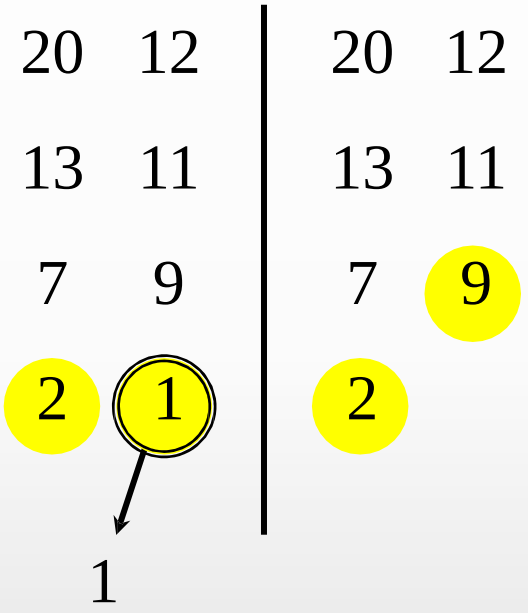
7 9

2 1

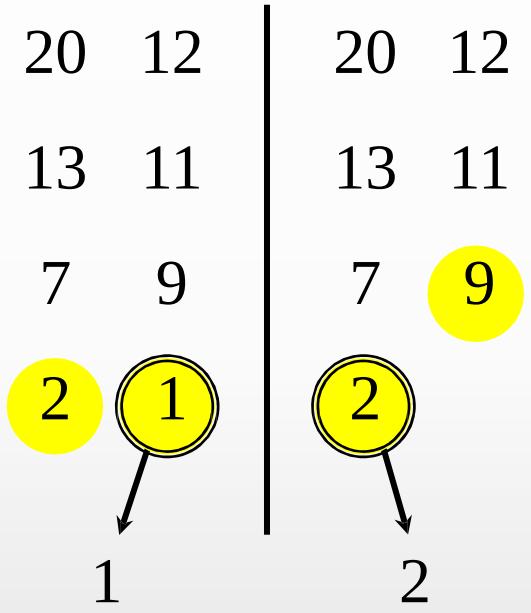
Birleştirme İşlemi (Merge)



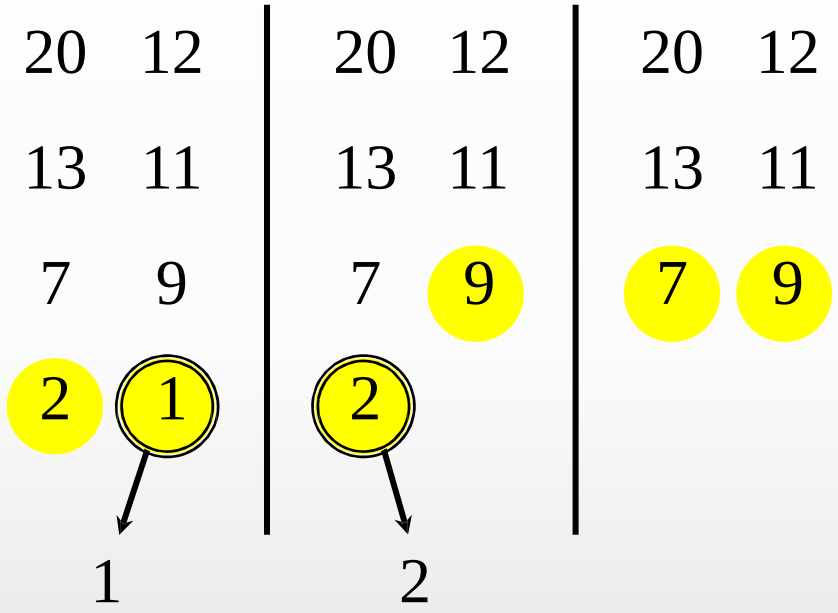
Birleştirme İşlemi (Merge)



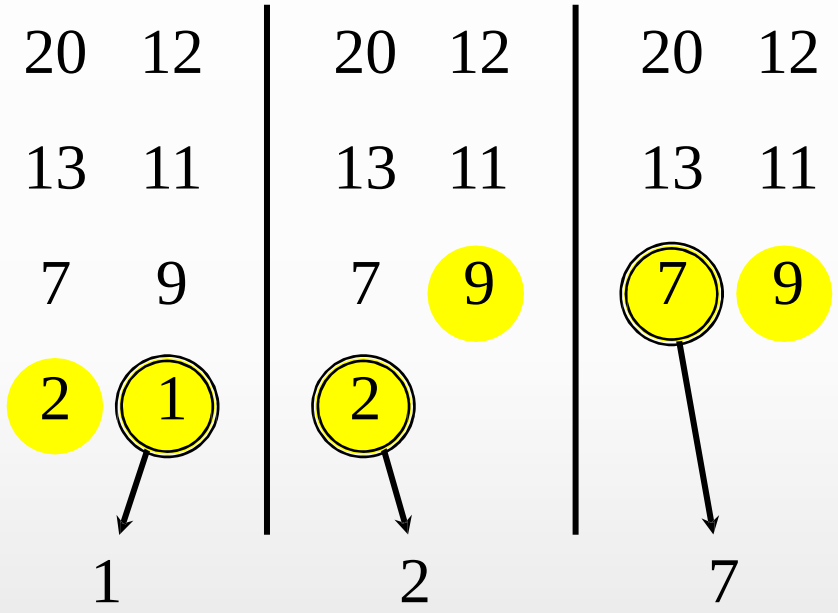
Birleştirme İşlemi (Merge)



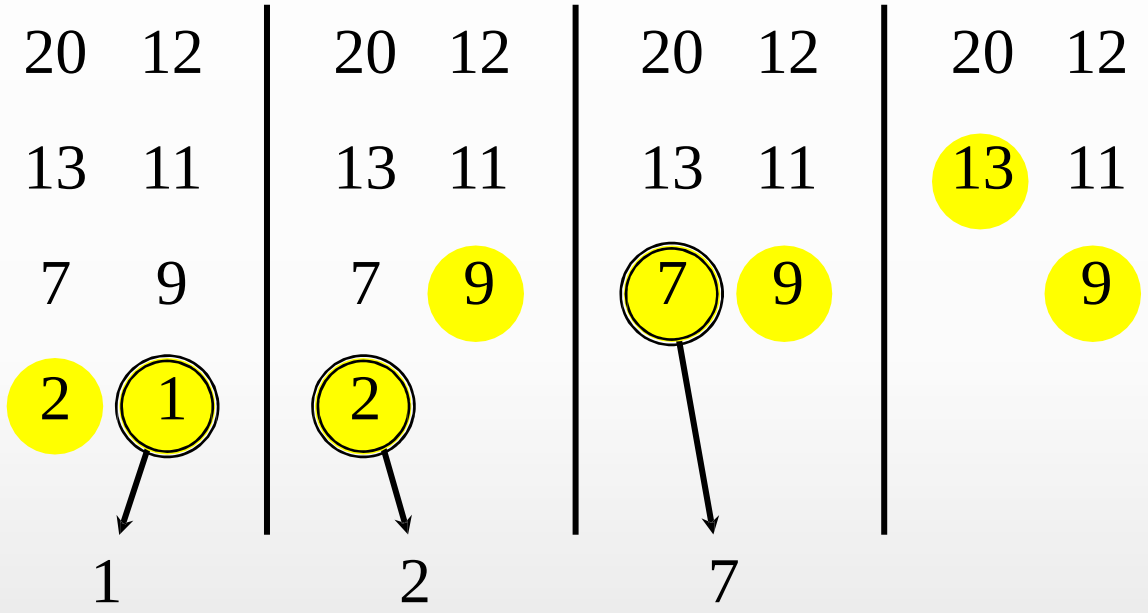
Birleştirme İşlemi (Merge)



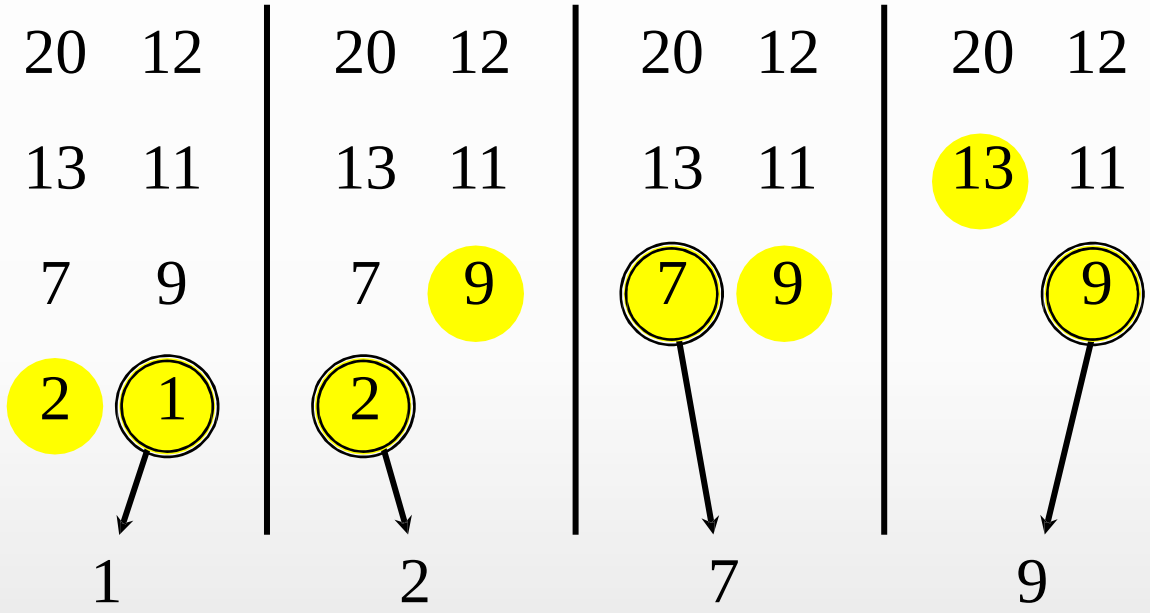
Birleştirme İşlemi (Merge)



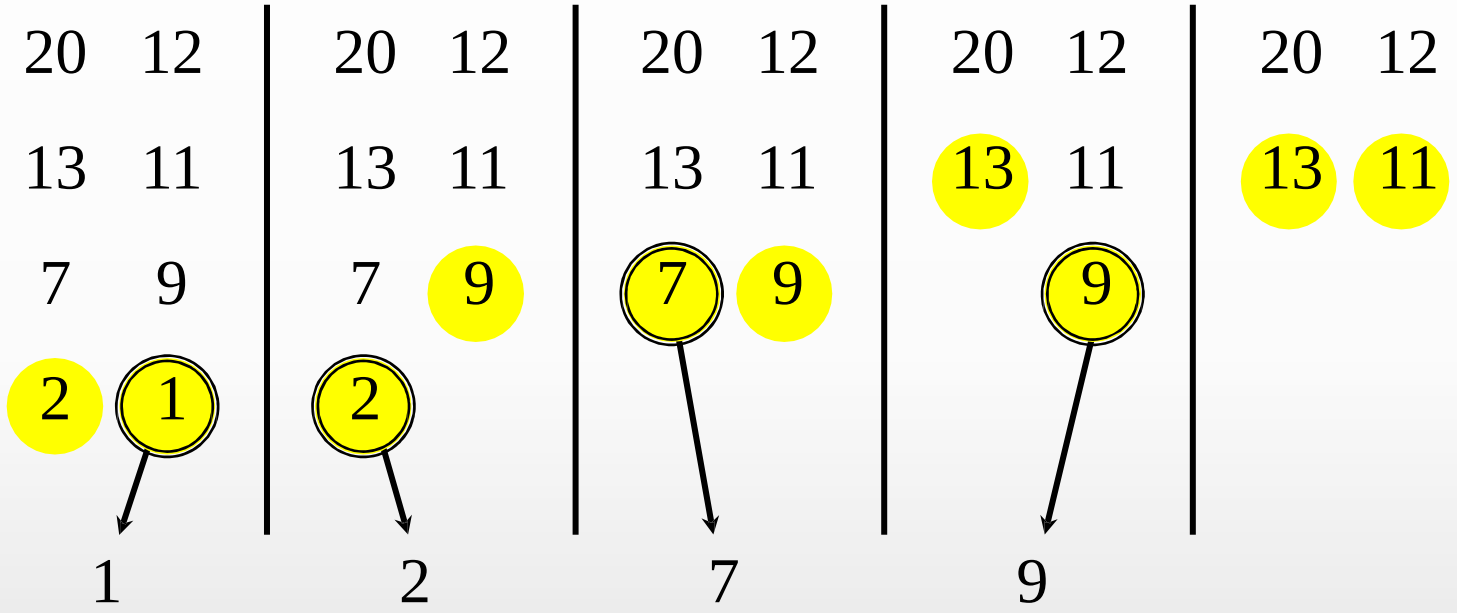
Birleştirme İşlemi (Merge)



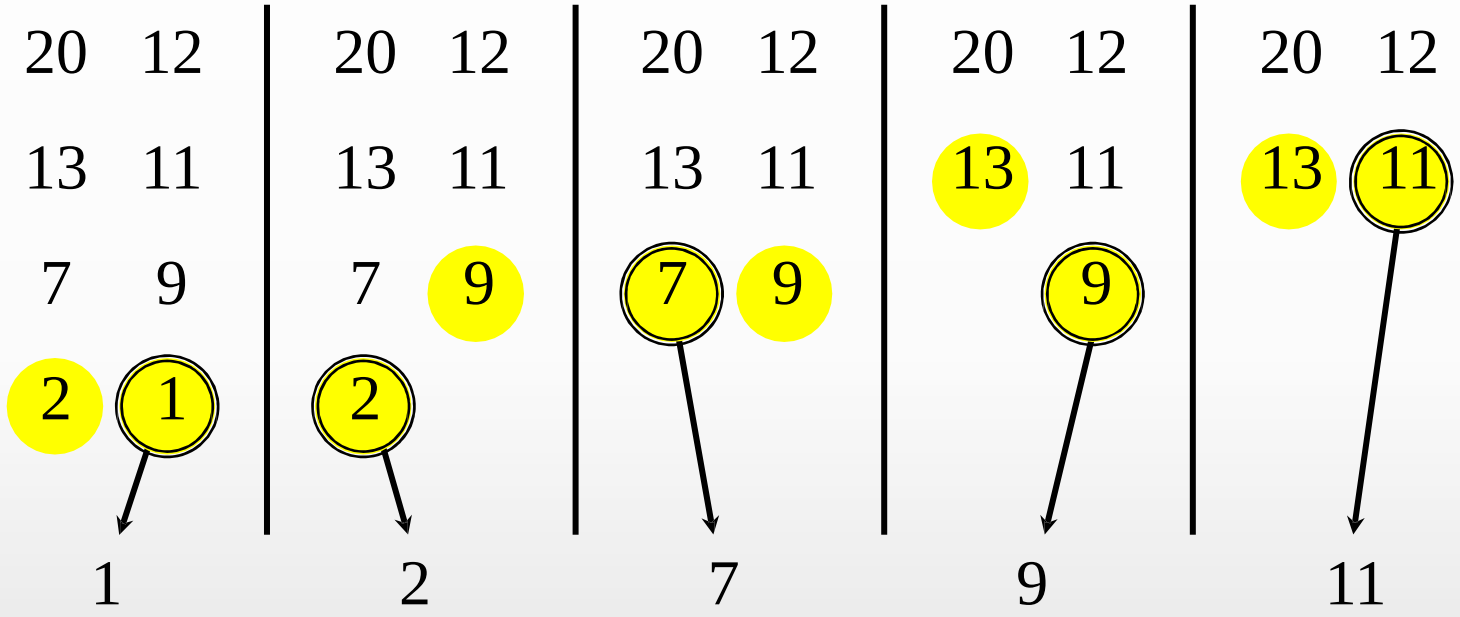
Birleştirme İşlemi (Merge)



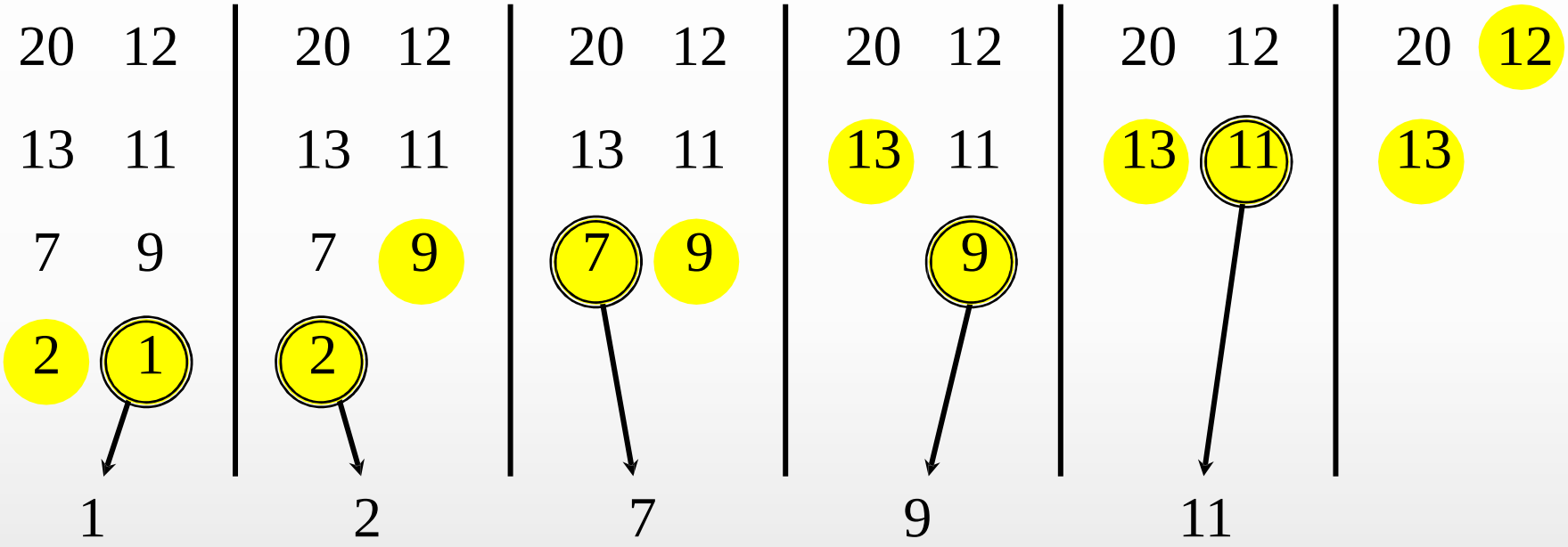
Birleştirme İşlemi (Merge)



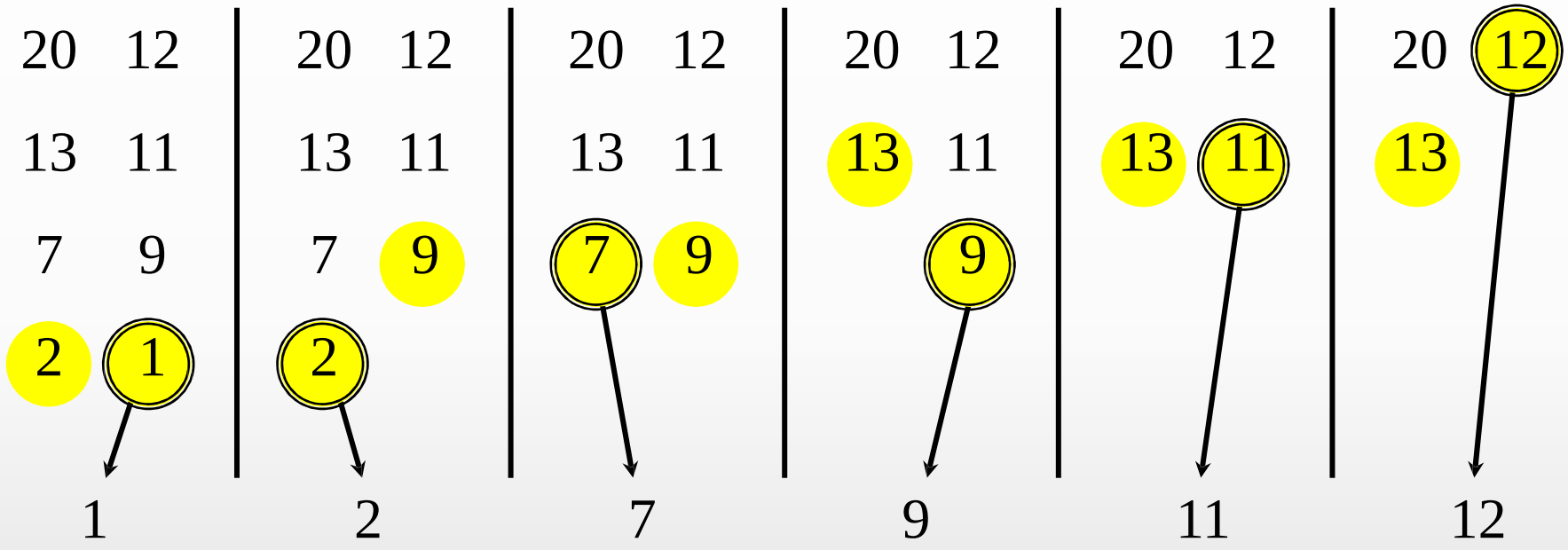
Birleştirme İşlemi (Merge)



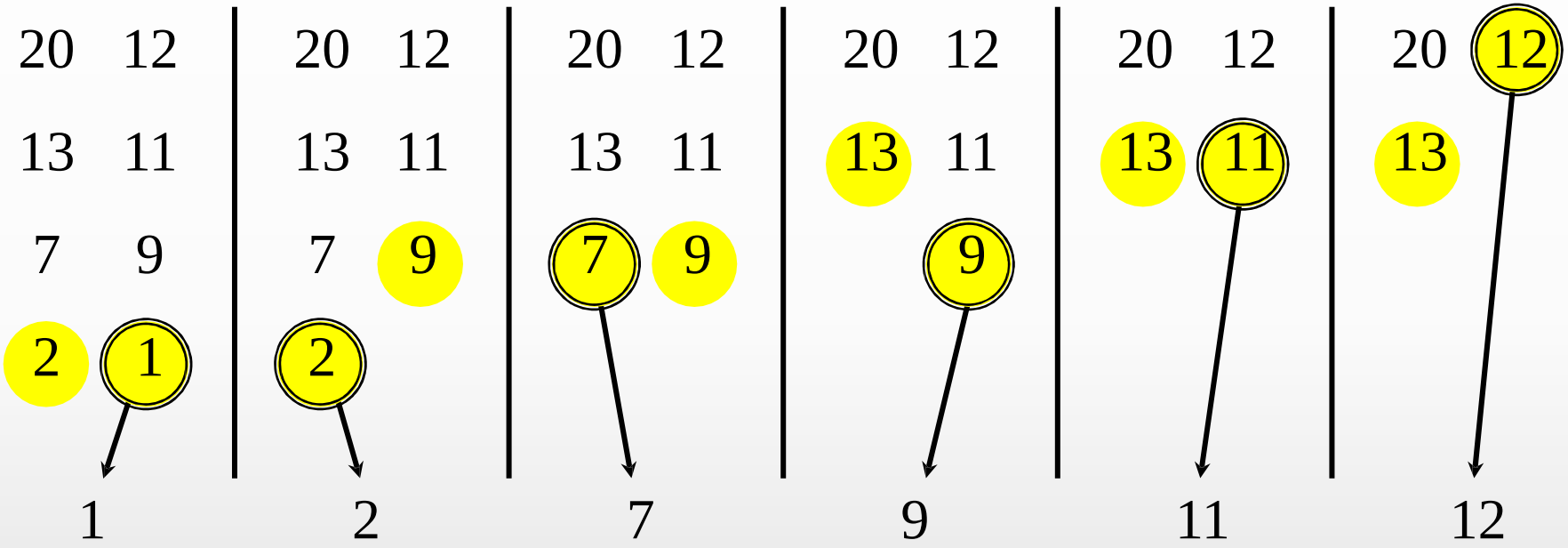
Birleştirme İşlemi (Merge)



Birleştirme İşlemi (Merge)



Birleştirme İşlemi (Merge)



- n elemanlı listeyi birleştirmek için $T(n) = \Theta(n)$.

İkili Arama

- Binary Search
- Sıralı listede bir elemanı arar.
  - Böl (*Divide*): orta elemanı kontrol et.
  - Fethet (*Conquer*): seçilen alt diziyi özyinelemeli olarak ara.
  - Birleştir (*Combine*): önemsiz (*trivial*).

İkili Arama

- **Örnek**: 9'u bul.

3	5	7	8	9	12	15

İkili Arama

- **Örnek**: 9'u bul.

3	5	7	8	9	12	15

3	5	7	8	9	12	15

İkili Arama

- **Örnek**: 9'u bul.

3	5	7	8	9	12	15

3	5	7	8	9	12	15

3	5	7	8	9	12	15

İkili Arama

- **Örnek**: 9'u bul.

3	5	7	8	9	12	15
3	5	7	**8**	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	**12**	15

İkili Arama

- **Örnek**: 9'u bul.

3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	12	15

İkili Arama

- **Örnek**: 9'u bul.

3	5	7	8	9	12	15
3	5	7	**8**	9	12	15
3	5	7	8	9	12	15
3	5	7	8	9	**12**	15
3	5	7	8	**9**	12	15

İkili Arama

- Binary Search
- Sıralı listede bir elemanı arar. *T(n)*
  - Böl (*Divide*): orta elemanı kontrol et. *Θ(1)*
  - Fethet (*Conquer*): seçilen alt diziyi özyinelemeli olarak ara. *T(n/2)*
  - Birleştir (*Combine*): önemsiz (*trivial*). *Θ(1)*

$$T(n) = 1T(n/2) + \Theta(1)$$

alt problem sayısı

alt problem büyüklüğü

bölme ve birleştirme

Matris Çarpımı

Girdi: $A = [a_{ij}], B = [b_{ij}].$

Çıktı: $C = [c_{ij}] = A \cdot B.$

$$\left.\begin{array}{l} \text{Girdi: } A = [a_{ij}], B = [b_{ij}]. \\ \text{Çıktı: } C = [c_{ij}] = A \cdot B. \end{array}\right\} \; i, j = 1, 2, \ldots, n.$$

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} \\ c_{21} & c_{22} & \cdots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{n1} & c_{n2} & \cdots & c_{nn} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \cdot \begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1n} \\ b_{21} & b_{22} & \cdots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{nn} \end{bmatrix}$$

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

Matris Çarpımı

```
for i = 1 to n
  do for j = 1 to n
    do c_ij = 0
      for k = 1 to n
        do c_ij = c_ij + a_ik · b_kj
```

Matris Çarpımı

```
for i = 1 to n
  do for j = 1 to n
    do c_ij = 0
      for k = 1 to n
        do c_ij = c_ij + a_ik · b_kj
```

Çalışma süresi $T(n) = \Theta(n^3)$

Bir Sayının Üssünü Alma

- **Problem**: a^n 'i hesapla (n € N).
- Doğal algoritma:
  - $a^n = a \; x \; a \; x \; a \; ....$
  - $\Theta(n)$.
- Böl ve Fethet algoritması:

$$a^n = \begin{cases} a^{n/2} \cdot a^{n/2} & n \text{ çift ise;} \\ a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a & n \text{ tek ise.} \end{cases}$$

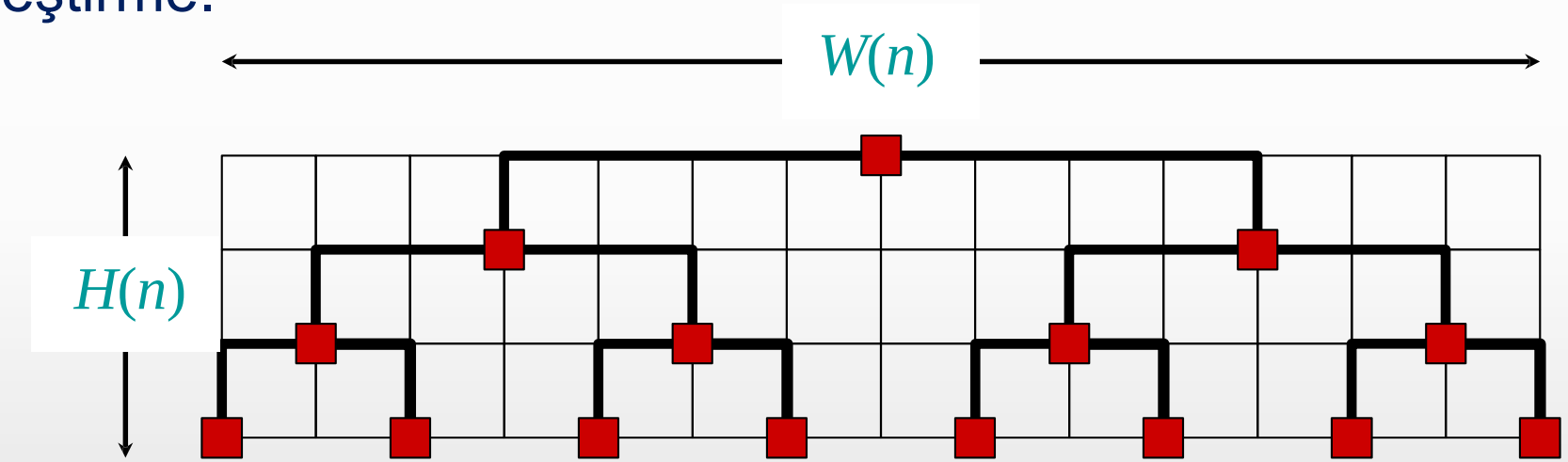
$$T(n) = T(n/2) + \Theta(1) \quad \Rightarrow \quad T(n) = \Theta(\lg n) \; .$$

Algoritma Çalışma Süresi

- Temel işlemler (atama, karşılaştırma, döngüler gibi) belirlenir.
- Her temel işlemin algoritma içinde kaç kez gerçekleştiği belirlenir.
- Her bir temel işlem adımının sayısı toplanarak birleştirilir.
- Toplam işlem sayısında sabit terim ve düşük kuvvetliler yok edilir.
- Çalışma süresi büyüme oranına odaklanarak asimptotik analizi yapılır.
- *Big-O, Omega, ve Theta* gösterimleri ile ifade edilir.

VLSI (Very Large Scale Integration) Yerleşimi

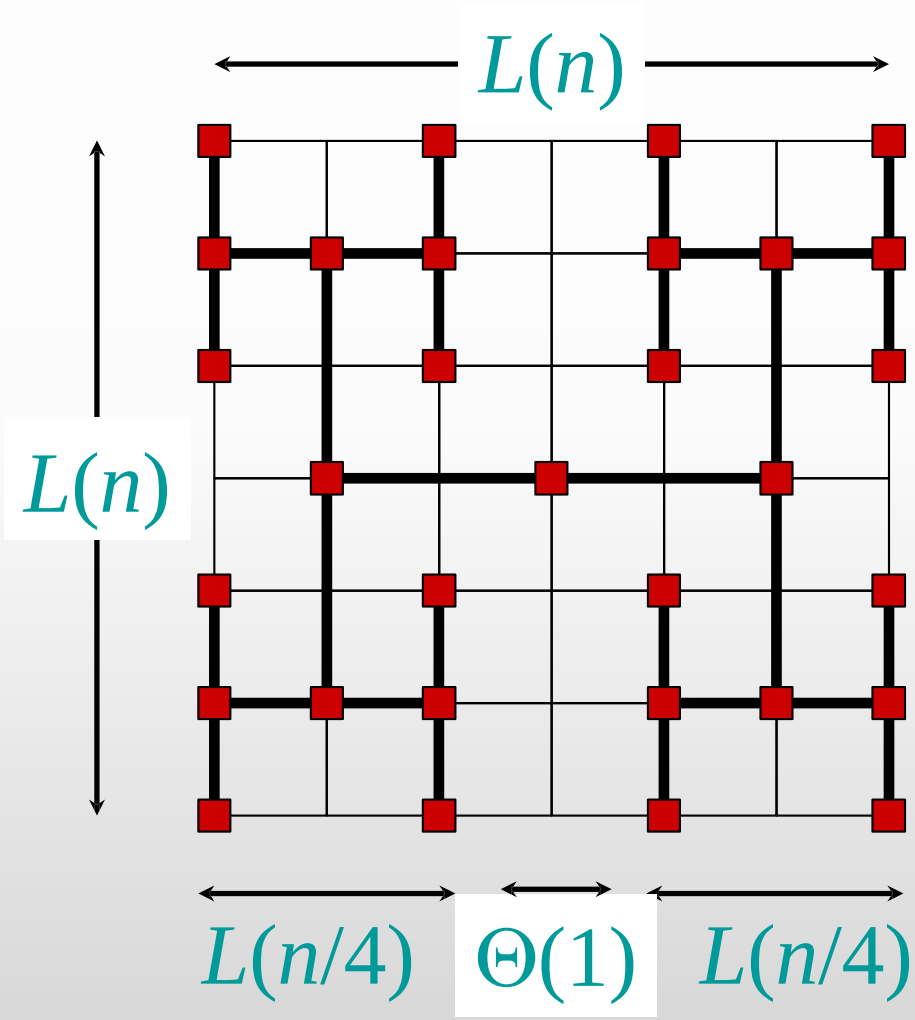
- **Problem**: n yapraklı ikili tam ağacı ızgara düzeninde minimum alana yerleştirme.



$$H(n) = H(n/2) + \Theta(1) \qquad W(n) = 2\,W(n/2) + \Theta(1)$$
$$= \Theta(\lg n) \qquad = \Theta(n)$$

$$\text{Alan} = \Theta(n \lg n)$$

H Ağacı Gömme (Embedding)



$$L(n) = 2L(n/4) + \Theta(1)$$
$$= \Theta(\sqrt{n})$$

Alan = $\Theta(n)$

SON