3.Hafta Algoritmaların Analizi

Araya Yerleştirme Sırlaması (Insert Sort)
Birleştirme Sıralaması (Merge Sort)
Yinelemeler

Sıralama (sorting) problemi

- **Girdi:** dizi $\langle \mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n} \rangle$ sayıları.

- **Çıktı:** $\langle \mathbf{a'_1}, \mathbf{a'_2}, \ldots, \mathbf{a'_n} \rangle$ elemanları $\mathbf{a_1}, \mathbf{a_2}, \ldots, \mathbf{a_n}$ elamanlarının bir permütasyonudur ve $\mathbf{a'_1 \le a'_2 \le \ldots \le a'_n}$.
- Permütasyon, rakamların sıralamasının değiştirilmesidir.

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

Araya yerleştirme sıralaması (Insertion sort)

- Insert Sort, artımsal tasarım yaklaşımını kullanır: A[1...j-1] alt dizisi sıralanmış.

- Strateji:
- Bir elemanı olduğu sıraya ekle
- Bütün elemanları sıralayana kadar devam et

Araya yerleştirme sıralaması örneği

8 2 4 9 3 6

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 3 4 8 9 6

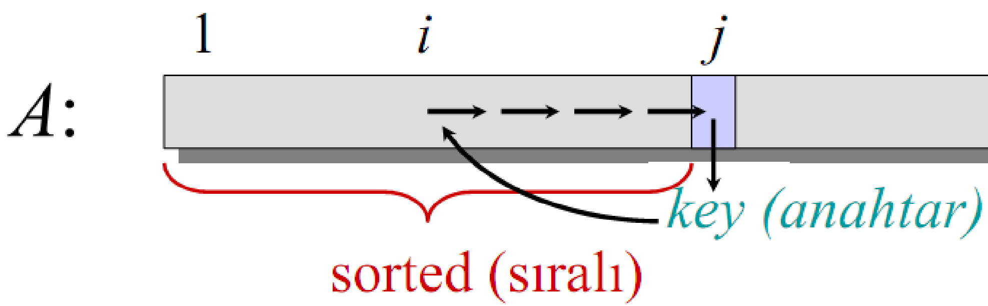
2 3 4 6 8 9

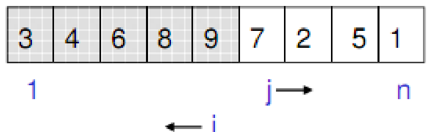
Araya yerleştirme sıralaması (Insertion sort)

"pseudocode"

(sözde kod)

```
INSERTION-SORT (A, n)        ▷ A[1 . . n]
    for j ← 2 to n
        do key ← A[ j]
           i ← j – 1
           while i > 0 and A[i] > key
               do A[i+1] ← A[i]
                  i ← i – 1
           A[i+1] = key
```





Örnek: Insertion Sort

30	10	40	20
1	2	3	4

$i = \varnothing \quad j = \varnothing \quad key = \varnothing$

$A[j] = \varnothing \quad A[j+1] = \varnothing$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

30	10	40	20
1	2	3	4

i = 2 j = 1 key = 10
A[j] = 30 A[j+1] = 10

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

30	30	40	20
1	2	3	4

i = 2 j = 1 key = 10

A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
⇨              A[j+1] = A[j]
              j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

30	30	40	20
1	2	3	4

i = 2 j = 1 key = 10

A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

30	30	40	20
1	2	3	4

$i = 2 \quad j = 0 \quad key = 10$

$A[j] = \varnothing \quad A[j+1] = 30$

```
InsertionSort(A, n) {
  for i = 2 to n {
       key = A[i]
       j = i - 1;
       while (j > 0) and (A[j] > key) {
              A[j+1] = A[j]
⇨             j = j - 1
       }
       A[j+1] = key
  }
}
```

Örnek : Insertion Sort

30	30	40	20
1	2	3	4

$i = 2 \quad j = 0 \quad key = 10$

$A[j] = \varnothing \quad A[j+1] = 30$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

$i = 2 \quad j = 0 \quad key = 10$

$A[j] = \varnothing \quad A[j+1] = 10$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

1	2	3	4
10	30	40	20

$i = 3$ $j = 0$ $key = 10$

$A[j] = \varnothing$ $A[j+1] = 10$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$ $j = 0$ key $= 40$

$A[j] = \varnothing$ $A[j+1] = 10$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

$i = 3$ $j = 0$ $key = 40$

$A[j] = \varnothing$ $A[j+1] = 10$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

i = 3   j = 2   key = 40

A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

i = 3   j = 2   key = 40

A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

i = 3   j = 2   key = 40

A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
     key = A[i]
     j = i - 1;
     while (j > 0) and (A[j] > key) {
          A[j+1] = A[j]
          j = j - 1
     }
     A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

i = 4   j = 2   key = 40

A[j] = 30   A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

$i = 4 \quad j = 2 \quad key = 20$

$A[j] = 30 \quad A[j+1] = 40$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

$i = 4 \quad j = 2 \quad key = 20$

$A[j] = 30 \quad A[j+1] = 40$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

i = 4   j = 3   key = 20

A[j] = 40   A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	20
1	2	3	4

i = 4 j = 3 key = 20

A[j] = 40 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 3 key = 20

A[j] = 40 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
⇨             A[j+1] = A[j]
             j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	40
1	2	3	4

i = 4   j = 3   key = 20
A[j] = 40   A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	40	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 40

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$ $j = 2$ $key = 20$

$A[j] = 30$ $A[j+1] = 30$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	30	40
1	2	3	4

i = 4 j = 2 key = 20
A[j] = 30 A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	30	40
1	2	3	4

$i = 4$ $j = 1$ key $= 20$

$A[j] = 10$ $A[j+1] = 30$

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	30	30	40
1	2	3	4

i = 4   j = 1   key = 20
A[j] = 10   A[j+1] = 30

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	20	30	40
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Örnek : Insertion Sort

10	20	30	40
1	2	3	4

i = 4 j = 1 key = 20
A[j] = 10 A[j+1] = 20

```
InsertionSort(A, n) {
  for i = 2 to n {
      key = A[i]
      j = i - 1;
      while (j > 0) and (A[j] > key) {
            A[j+1] = A[j]
            j = j - 1
      }
      A[j+1] = key
  }
}
```

Bitti!

Hatırlatma
Çalışma Zamanı

- Çalışma zamanı veya koşma süresi girişe bağımlıdır: Önceden sıralanmış bir diziyi sıralamak daha kolaydır.
- Çalışma zamanının girişin boyutuna göre parametrelenmesi yararlıdır, çünkü kısa dizileri sıralamak uzun dizilere oranla daha kolaydır.
- Genellikle, çalışma zamanında üst sınır aranır.

Hatırlatma

Çözümleme türleri

- **En kötü durum (Worst-case):** (genellikle bununla ilgilenilecek)
  - **T(n) = n** boyutlu bir girişte algoritmanın maksimum süresi. (Programın her durumda çalışması)
- **Ortalama durum:** (bazen ilgilenilecek)
  - **T(n) = n** boyutlu her girişte algoritmanın beklenen süresi (ağırlıklı ortalama).
  - Girişlerin istatistiksel dağılımı için varsayım gerekli.
  - En çok kullanılan varsayımlardan biri **n** boyutunda her girişin eşit oranda olasılığının olduğunu kabul etmek.
- **En iyi durum:** (gerçek dışı)
  - Sadece bir giriş yapısında hızlı çalışan fakat yavaş bir algoritma ile hile yapmak.

Hatırlatma
Makineden-bağımsız zaman

- Araya yerleştirme sıralamasının en kötü zamanı nedir?
- Bilgisayarın hızına bağlıdır:
  - bağıl (rölatif) zaman (aynı makinede),
  - mutlak (absolüt) zaman (farklı makinelerde).
- BÜYÜK FİKİR:
  - Makineye bağımlı sabitleri görmezden gel.
  - n $\rightarrow\infty$ 'a yaklaştıkça, T(n)'nin büyümesine bak.

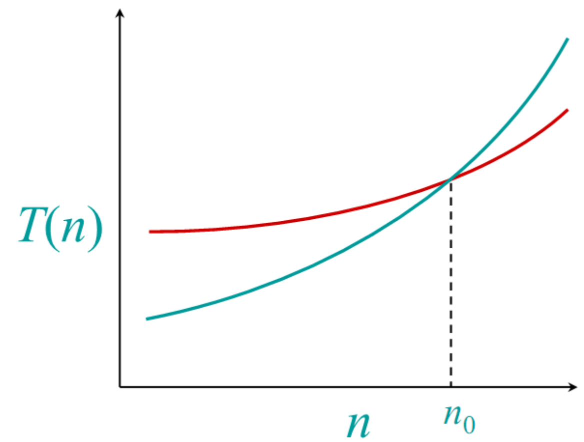
" Asimptotik Çözümleme"

Hatırlatma
Asimptotik başarım

- n yeterince büyürse, $\Theta(n^2)$ algoritması bir $\Theta(n^3)$ algoritmasından her zaman daha hızlıdır.

  - Öte yandan asimptotik açıdan yavaş algoritmaları ihmal etmemeliyiz.

  - Asimptotik çözümleme düşüncemizi yapılandırmada önemli bir araçtır.



Insertion sort algoritmasının analizi

// sort in increasing order //	cost	times
for j ← 2 **to** length[A]	c_1	n
do key ← A[j]	c_2	$n-1$
// insert A[j] into the sorted sequence A[1..j-1]		
i ← j –1	c_4	$n-1$
while i > 0 **and** A[i] > key	c_5	$\sum_{2 \le j \le n} t_j$
do A[i+1] ← A[i]	c_6	$\sum_{2 \le j \le n} (t_j-1)$
i ← i –1	c_7	$\sum_{2 \le j \le n} (t_j-1)$
done		
A[i+1] ← key	c_8	$n-1$
done		

Insertion sort algoritmasının analizi

// sort in increasing order //	cost	times
for j ← 2 to length[A]	c_1	n
do key ← A[j]	c_2	n-1
// insert A[j] into the sorted sequence A[1..j-1]		
i ← j –1	c_4	n-1
while i > 0 and A[i] > key	c_5	$\sum_{2\le j\le n} t_j$
do A[i+1] ← A[i]	c_6	$\sum_{2\le j\le n} (t_j-1)$
i ← i –1	c_7	$\sum_{2\le j\le n} (t_j-1)$
done		
A[i+1] ← key	c_8	n-1
done		

- Insert sort algoritmasının çalışma zamanı (T(n)), cost(maliyet), times (tekrar) toplamıdır.

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} t_j + c_6\sum_{j=2}^{n}(t_j-1) + c_7\sum_{j=2}^{n}(t_j-1) + c_8(n-1)$$

Araya yerleştirme sıralaması

- **En iyi durum:** dizi sıralı ise
- $j=2,\dots n$ için $t_j=1$ ise

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

$$T(n) = (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)$$

$T(n) = an + b$, Lineer fonksiyon

$$T(n) = \theta(n)$$

Araya yerleştirme sıralaması

- **En kötü durum:** dizi ters sıralı
- *j=2,…n* için $t_j=j$ ise

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \; ve \sum_{j=2}^{n} j - 1 = \frac{n(n-1)}{2}$$

$$T(n) = c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1)$$

$$T(n) = \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n - (c_2 + c_4 + c_5 + c_8)$$

$$T(n) = an^2 + bn + c$$ **(Quadratic Function)**

$$T(n) = \theta(n^2)$$

Araya yerleştirme sıralaması analizi özet

- **En kötü durum:** Giriş tersten sıralıysa.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta\left(n^2\right) \qquad \text{[aritmetik seri]}$$

- **Ortalama durum:** Tüm permutasyonlar eşit olasılıklı.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta\left(n^2\right)$$

- Araya yerleştirme sıralaması hızlı bir algoritma mıdır ?
  - Küçük n değerleri için olabilir.
  - Büyük n değerleri için asla!

Bazı fonksiyonların büyüme oranları

- Aşağıda verilen fonksiyonlar için yandaki kurallar uygulanacak:

1. $f(n) = \sum_{1 \le i \le n} i = n(n+1)/2$
2. $f(n) = \sum_{1 \le i \le n} i^2 = n(n+1)(2n+1)/6$
3. $f(n) = \sum_{0 \le i \le n} x^i = (x^{n+1} - 1) / (x - 1)$
4. $f(n) = \sum_{0 \le i \le n} 2^i = (2^{n+1}-1) / (2-1) = 2^{n+1}-1$

$$\sum_{i=1}^{n} i = 1 + 2 + \dots + n = \frac{n*(n+1)}{2} \approx \frac{n^2}{2}$$

$$\sum_{i=0}^{n-1} 2^i = 0 + 1 + 2 + \dots + 2^{n-1} = 2^n - 1$$

$$\sum_{i=1}^{n} i^2 = 1 + 4 + \dots + n^2 = \frac{n*(n+1)*(2n+1)}{6} \approx \frac{n^3}{3}$$

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-1

```
for k = 1 to n/2 do
  {
    ....
  }
for j = 1 to n*n do
  {
    ----  ....
  }
```

$$\text{....} \longrightarrow c_1$$

$$\text{---- ....} \longrightarrow c_2$$

$$\sum_{k=1}^{n/2} c_1 + \sum_{j=1}^{n*n} c_2 = c_1 n/2 + c_2 n^2 = O(n^2)$$

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-2

```
for k = 1 to n/2 do
{
    for j = 1 to n*n do
    {
            ----
    }
}
```

$$\sum_{k=1}^{n/2} \sum_{j=1}^{n*n} c = c\, n/2 * n^2 = O(n^3)$$

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-3

Code	Cost	Times
`i = 1;`	c_1	1
`sum = 0;`	c_2	1
`while (i <= n) {`	c_3	n+1
`    i = i + 1;`	c_4	n
`    sum = sum + i;}`	c_5	n

$$T(n) \quad = c_1 + c_2 + c_3(n+1) + c_4 n + c_5 n$$
$$= (c_3 + c_4 + c_5)n + (c_1 + c_2 + c_3)$$

$$T(n) \quad = an + b \rightarrow O(n)$$

T(n) → Bu algoritmanın büyüme oran fonksiyonu, O(n) dir

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-4

Code	Cost	Times
`i=1;`	c_1	1
`sum = 0;`	c_2	1
`while (i <= n) {`	c_3	n+1
`    j=1;`	c_4	n
`    while (j <= n) {`	c_5	n*(n+1)
`        sum = sum + i;`	c_6	n*n
`        j = j + 1;`	c_7	n*n
`    }`		
`    i = i +1;`	c_8	n
`}`		

$$\begin{aligned} T(n) &= c_1 + c_2 + c_3(n+1) + c_4 n + c_5 n(n+1) + c_6 n(n) + c_7 n(n) + c_8 n \\ &= (c_5 + c_6 + c_7)n^2 + (c_3 + c_4 + c_5 + c_8)n + (c_1 + c_2 + c_3) \\ &= an^2 + bn + c \end{aligned}$$

➔ Bu algoritmanın büyüme oran fonksiyonu, $O(n^2)$ dir.

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-5

```
for (int i = 0; i < N; i++)
    for (int j = i+1; j < N; j++)
        if (a[i] + a[j] == 0)
            count++;
```

$$
\begin{aligned}
0 + 1 + 2 + \ldots + (N-1) &= \frac{1}{2} N(N-1) \\
&= \binom{N}{2}
\end{aligned}
$$

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-6

```
int count = 0;
for (int i = 0; i < N; i++)
   for (int j = i+1; j < N; j++)
      for (int k = j+1; k < N; k++)
         if (a[i] + a[j] + a[k] == 0)
            count++;
```

$$\binom{N}{3} = \frac{N(N-1)(N-2)}{3!} \sim \frac{1}{6}N^3$$

Büyüme oranı (Growth-Rate) fonksiyonları Örnek-7

	Cost	Times
`for (i=1; i<=n; i++)`	c_1	n+1
`for (j=1; j<=i; j++)`	c_2	$\sum_{j=1}^{n}(j+1)$
`for (k=1; k<=j; k++)`	c_3	$\sum_{j=1}^{n}\sum_{k=1}^{j}(k+1)$
`x=x+1;`	c_4	$\sum_{j=1}^{n}\sum_{k=1}^{j}k$

$$T(n) = c_1(n+1) + c_2\sum_{j=1}^{n}(j+1) + c_3\sum_{j=1}^{n}\sum_{k=1}^{j}(k+1) + c_4\sum_{j=1}^{n}\sum_{k=1}^{j}k$$

$$= an^3 + bn^2 + cn + d$$

➔ **Bu algoritmanın büyüme oran fonksiyonu, $O(n^3)$ dir.**

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-8

```
for i = 1 to n do
  for j = i to n do
        for k = i to j do
                m = m + i + j + k
```

$$T(n) = \sum^{n}_{i=1}\sum^{n}_{j=i}\sum^{j}_{k=i}3$$

$$= \sum^{n}_{i=1}\sum^{n}_{j=i}3(j-i+1) = \sum^{n}_{i=1}3[\sum^{n}_{j=i}j - \sum^{n}_{j=i}i + \sum^{n}_{j=i}1]$$

$$= \sum^{n}_{i=1}3[(\sum^{n}_{j=1}j - \sum^{i-1}_{j=1}j) - i(n-i+1) + (n-i+1)]$$

$$= \sum^{n}_{i=1}3[[n(n+1)/2 - i(i-1)/2] - i(n-i+1) + (n-i+1)]$$

$\approx n^3/10$ additions, that is, $O(n^3)$

Özyinelemeli Algoritmaların Analizi

- Yerine koyma metodu
- Yineleme döngüleri
- Özyineleme ağacı
- Ana Metot (Master metod)

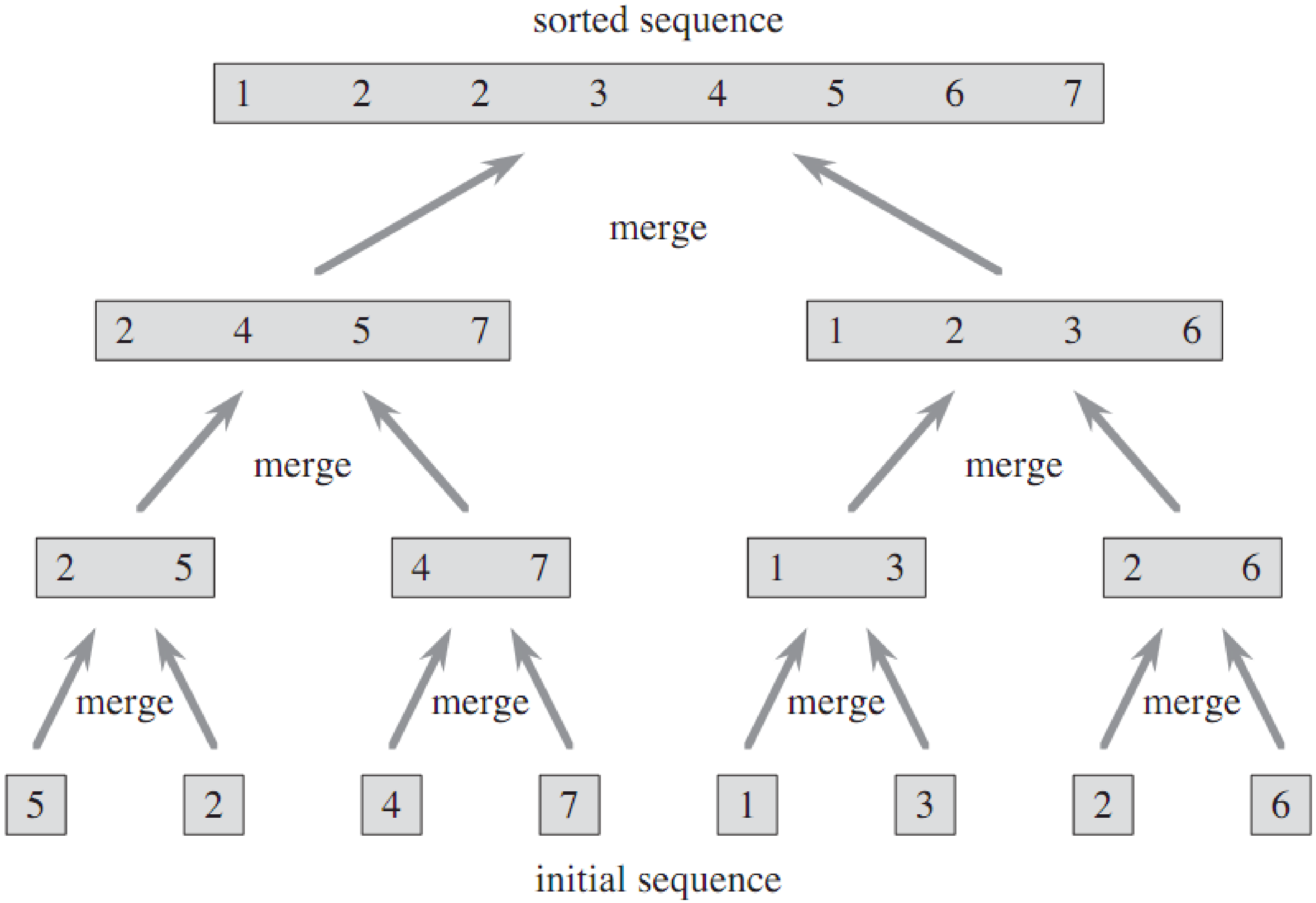
Hatırlatma-Özyinelemeli Tanımlar

- Bir dizi, seri, fonksiyon ve algoritmanın kendi cinsinden tanımlanmasına ***özyineleme*** denir.
- Tanım bölgesi negatif tamsayılar olmayan fonksiyon tanımlanırken:
  - **Temel Adım:** Fonksiyonun sıfırdaki değeri belirtilir.
  - **Özyinelemeli adım:** Fonksiyonun bir tamsayıdaki değeri hesaplanırken, fonksiyonun daha küçük tamsayılardaki değer(ler)ini kullanarak bu değeri veren kural belirtilir.
    - İkinin kuvvetlerinden oluşan dizi aşağıdaki gibi ifade edilebilir
    - $a_n=2^n$
    - Fakat bu dizi özyinelemeli olarak aşağıdaki gibi ifade edilebilir.
    - $a_0=1,\ a_{n+1}=2a_n$

Hatırlatma-Özyinelemeli Tanımlar

- Örnek: ***f*** fonksiyonu öz yinelemeli olarak aşağıdaki tanımlanmış olsun;
- *$f(0)=3$, temel durum*
- *$f(n+1)=2f(n)+3$, $f(1)$, $f(2)$, $f(3)$* değerleri nedir?
  - $f(1) = 2f(0) + 3 = 2 * 3 + 3 = 9$
  - $f(2) = 2f(1) + 3 = 2 * 9 + 3 = 21$
  - $f(3) = 2f(2) + 3 = 2 * 21 + 3 = 45$
  - $f(4) = 2f(3) + 3 = 2 * 45 + 3 = 93$

Birleştirme sıralaması-Merge Sort



Birleştirme sıralaması-Merge Sort

BİRLEŞTİRME-SIRALAMASI $A[1\,..\,n]$

1. Eğer $n = 1$ ise, işlem bitti.
2. $A[\,1\,..\,\lceil n/2 \rceil\,]$ve
   $A[\,\lceil n/2 \rceil + 1\,..\,n\,]$'yi özyinelemeli sırala.
3. 2 sıralanmış listeyi "***Birleştir***".

Anahtar altyordam: **Birleştirme**

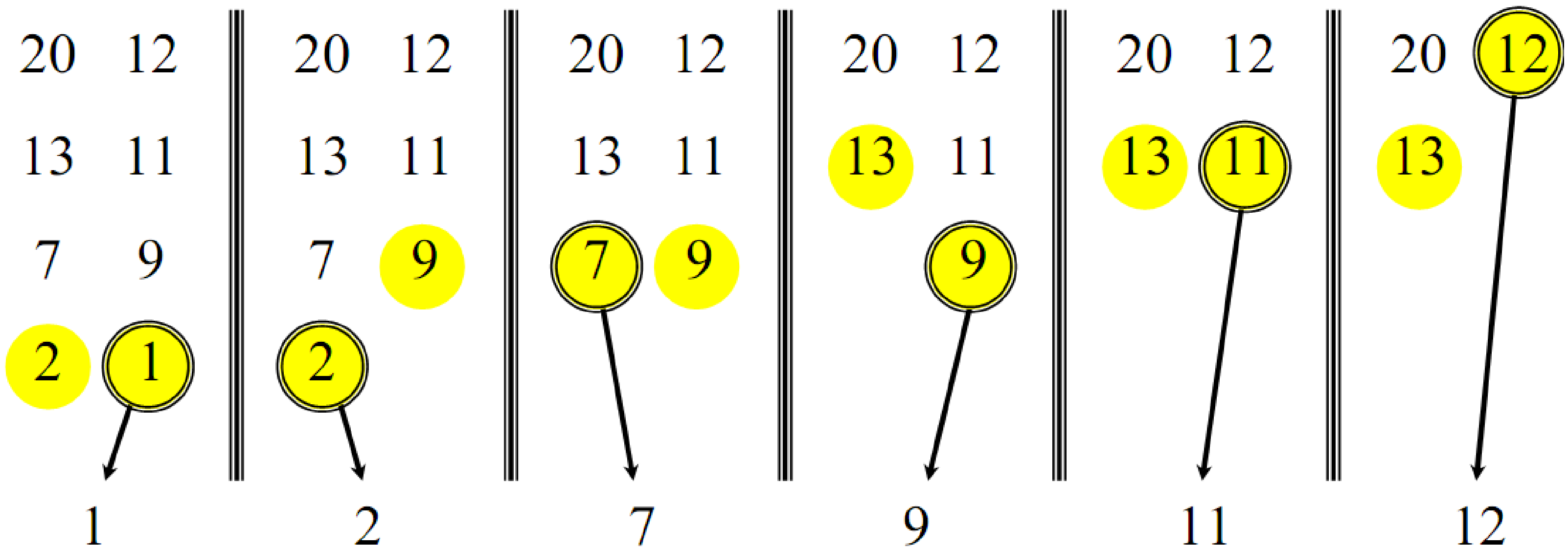
Birleştirme sıralaması-Merge Sort

```
MERGE-SORT(A, p, r)
  if p < r
      q = ⌊(p + r)/2⌋
      MERGE-SORT(A, p, q)
      MERGE-SORT(A, q + 1, r)
      MERGE(A, p, q, r)
```

Merge(), A dizisinin iki tane sıralı alt dizisini alır ve onları tek bir sıralı alt dizi içerisinde birleştirir.
Bu işlem ne kadar zaman alır?

```
MERGE(A, p, q, r)
  n1 = q - p + 1
  n2 = r - q
  let L[1 .. n1 + 1] and R[1 .. n2 + 1] be new arrays
  for i = 1 to n1
      L[i] = A[p + i - 1]
  for j = 1 to n2
      R[j] = A[q + j]
  L[n1 + 1] = ∞
  R[n2 + 1] = ∞
  i = 1
  j = 1
  for k = p to r
      if L[i] ≤ R[j]
          A[k] = L[i]
          i = i + 1
      else A[k] = R[j]
          j = j + 1
```

Sıralı iki dizilimi birleştirme



Süre = $\Theta(n)$, toplam n elemanı birleştirmek için (doğrusal zaman).

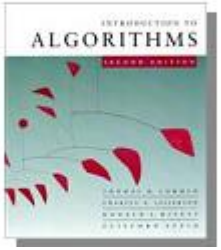
Birleştirme sıralaması-Merge Sort

```
MergeSort(A, left, right) {                T(n)
 if (left < right) {                       Θ(1)
   mid = floor((left + right) / 2);        Θ(1)
   MergeSort(A, left, mid);                T(n/2)
   MergeSort(A, mid+1, right);             T(n/2)
   Merge(A, left, mid, right);             Θ(n)
   }
}
```

- Birleştirme sıralaması çözümlemesi bir yinelemeyi çözmemizi gerektirir.

Yinelemeler –(Reküranslar)

- Yinelemeli algoritmalarda çalışma süresi reküranslar kullanılarak tanımlanabilir.
- Bir **yineleme** herhangi bir fonksiyonu kendisinin küçük girişleriyle tanımlar.
- Yinelemeleri çözmek için genel bir prosedür yada yöntem yoktur.
- Malesef yinelemeleri çözmek için iyi bir algoritma da yoktur. Sadece birtakım teknikler vardır. Bunların bazıları bazen işe yarar ve eğer şanslıysanız sizin yinelemeniz için bunlardan biri çalışır.



Sorular

- 1- Asimptotik notasyonlardan hangilerinin geçişme özelliği (transitivity) vardır.

- 2-f(n)=n^2+4nlogn+900 ve g(n)=n^2+45logn+h(n) fonksiyonları verilmiştir ve h(n) lineer olan bir polinomdur. f(n) ile g(n) arasındaki asimptotik ilişki nedir? f(n) ve g(n) arasındaki asimptotik ilişkiyi belirlerken h(n) polinomuna ihtiyaç var mıdır? Hangi durumlarda ihtiyaç duyulur veya duyulmaz?

1. $T(n) = T\left(\frac{n}{\ }\right) + \sqrt{n} + \sqrt{6046}$ $O(\sqrt{n})$

$$1+x+x^2+\cdots+x^n=\frac{1-x^{n+1}}{1-x} \quad ; x \neq 1 \text{ için}$$

$$1+x+x^2+\cdots=\frac{1}{1-x} \quad ; |x|<1 \text{ için}$$

Bazı Matematiksel İfadeler

S(N) = 1 + 2 + 3 + 4 + … N = $\sum_{i=1}^{N} i = \frac{N(N+1)}{2}$

Karelerin Toplamı: $\sum_{i=1}^{N} i^2 = \frac{N*(N+1)*(2n+1)}{6} \approx \frac{N^3}{3}$

Geometrik Seriler: $\sum_{i=0}^{N} A^i = \frac{A^{N+1}-1}{A-1}$ A > 1

$\sum_{i=0}^{N} A^i = \frac{1-A^{N+1}}{1-A} = \Theta(1)$ |A| < 1

Bazı Matematiksel İfadeler

- İki sınır arasındaki sayıların toplamı: $\sum_{i=a}^{b} f(i) = \sum_{i=0}^{b} f(i) - \sum_{i=0}^{a-1} f(i)$

$$\sum_{i=1}^{n} (4i^2 - 6i) = 4\sum_{i=1}^{n} i^2 - 6\sum_{i=1}^{n} i$$

- Combinatorics

1. Number of permutations of an n-element set: $P(n) = n!$

2. Number of k-combinations of an n-element set: $C(n, k) = \frac{n!}{k!(n-k)!}$

3. Number of subsets of an n-element set: 2^n

Bazı Matematiksel İfadeler

- Properties of Logarithms

1. $\log_a 1 = 0$

2. $\log_a a = 1$

3. $\log_a x^y = y \log_a x$

4. $\log_a xy = \log_a x + \log_a y$

5. $\log_a \frac{x}{y} = \log_a x - \log_a y$

6. $a^{\log_b x} = x^{\log_b a}$

7. $\log_a x = \frac{\log_b x}{\log_b a} = \log_a b \log_b x$

Bazı Önemli Matematiksel İfadeler

- Important Summation Formulas

1. $\sum_{i=l}^{u} 1 = \underbrace{1+1+\cdots+1}_{u-l+1 \text{ times}} = u - l + 1$ (l, u are integer limits, $l \le u$); $\sum_{i=1}^{n} 1 = n$

2. $\sum_{i=1}^{n} i = 1 + 2 + \cdots + n = \frac{n(n+1)}{2} \approx \frac{1}{2}n^2$

3. $\sum_{i=1}^{n} i^2 = 1^2 + 2^2 + \cdots + n^2 = \frac{n(n+1)(2n+1)}{6} \approx \frac{1}{3}n^3$

4. $\sum_{i=1}^{n} i^k = 1^k + 2^k + \cdots + n^k \approx \frac{1}{k+1}n^{k+1}$

5. $\sum_{i=0}^{n} a^i = 1 + a + \cdots + a^n = \frac{a^{n+1}-1}{a-1}$ $(a \ne 1)$; $\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$

6. $\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$ $\quad \sum_{i=0}^{n-1} ix^i = x + 2x^2 + 3x^3 \cdots + nx^n = \frac{(n-1)x^{(n+1)} - nx^n + x}{(x-1)^2}.$

7. $\sum_{i=1}^{n} \frac{1}{i} = 1 + \frac{1}{2} + \cdots + \frac{1}{n} \approx \ln n + \gamma$, where $\gamma \approx 0.5772\ldots$ (Euler's constant)

8. $\sum_{i=1}^{n} \lg i \approx n \lg n$

Bazı Önemli Matematiksel İfadeler

1. $\sum_{k=1}^{n} k = 1+2+3+\ldots+n = \frac{n(n+1)}{2}$

2. $\sum_{k=1}^{n} 2k = 2+4+6+\ldots+2n = n(n+1)$

3. $\sum_{k=1}^{n} (2k-1) = 1+3+5+\ldots+(2n-1) = n^2$

4. $\sum_{k=1}^{n} k^2 = 1^2+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6}$

5. $\sum_{k=1}^{n} k^3 = 1^3+2^3+3^3+\ldots+n^3 = \left[\frac{n(n+1)}{2}\right]^2$

6. $\sum_{k=1}^{n} r^{k-1} = 1+r+r^2+r^3+\ldots+r^{n-1} = \frac{1-r^n}{1-r}, \ (r \neq 1)$

7. $\sum_{k=1}^{n} \frac{1}{k\cdot(k+1)} = \frac{1}{1\cdot 2} + \frac{1}{2\cdot 3} + \ldots + \frac{1}{n\cdot(n+1)} = \frac{n}{n+1}$

8. $\sum_{k=1}^{n} k\cdot k! = (n+1)! - 1$

Bazı Matematiksel İfadeler

- Sum Manipulation Rules

1. $$\sum_{i=l}^{u} ca_i = c\sum_{i=l}^{u} a_i$$

2. $$\sum_{i=l}^{u} (a_i \pm b_i) = \sum_{i=l}^{u} a_i \pm \sum_{i=l}^{u} b_i$$

3. $$\sum_{i=l}^{u} a_i = \sum_{i=l}^{m} a_i + \sum_{i=m+1}^{u} a_i, \text{ where } l \le m < u$$

4. $$\sum_{i=l}^{u} (a_i - a_{i-1}) = a_u - a_{l-1}$$

Approximation of a Sum by a Definite Integral

$$\int_{l-1}^{u} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l}^{u+1} f(x)dx \quad \text{for a nondecreasing } f(x)$$

$$\int_{l}^{u+1} f(x)dx \le \sum_{i=l}^{u} f(i) \le \int_{l-1}^{u} f(x)dx \quad \text{for a nonincreasing } f(x)$$

Floor and Ceiling Formulas

The *floor* of a real number x, denoted $\lfloor x \rfloor$, is defined as the greatest integer not larger than x (e.g., $\lfloor 3.8 \rfloor = 3$, $\lfloor -3.8 \rfloor = -4$, $\lfloor 3 \rfloor = 3$). The *ceiling* of a real number x, denoted $\lceil x \rceil$, is defined as the smallest integer not smaller than x (e.g., $\lceil 3.8 \rceil = 4$, $\lceil -3.8 \rceil = -3$, $\lceil 3 \rceil = 3$).

1. $x - 1 < \lfloor x \rfloor \le x \le \lceil x \rceil < x + 1$

2. $\lfloor x + n \rfloor = \lfloor x \rfloor + n$ and $\lceil x + n \rceil = \lceil x \rceil + n$ for real x and integer n

3. $\lfloor n/2 \rfloor + \lceil n/2 \rceil = n$

4. $\lceil \lg(n + 1) \rceil = \lfloor \lg n \rfloor + 1$

Miscellaneous

1. $n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$ as $n \to \infty$ (Stirling's formula)

2. Modular arithmetic (n, m are integers, p is a positive integer)

$$(n + m) \bmod p = (n \bmod p + m \bmod p) \bmod p$$

$$(nm) \bmod p = ((n \bmod p)(m \bmod p)) \bmod p$$

4.Hafta Yinelemelerin çözümü

Yerine Koyma, İterasyon, Master Metot