

3.Hafta

Algoritmaların

Analizi

Araya Yerleştirme Sırlaması
(Insert Sort)

Birleştirme Sıralaması (Merge Sort)
Yinelemeler

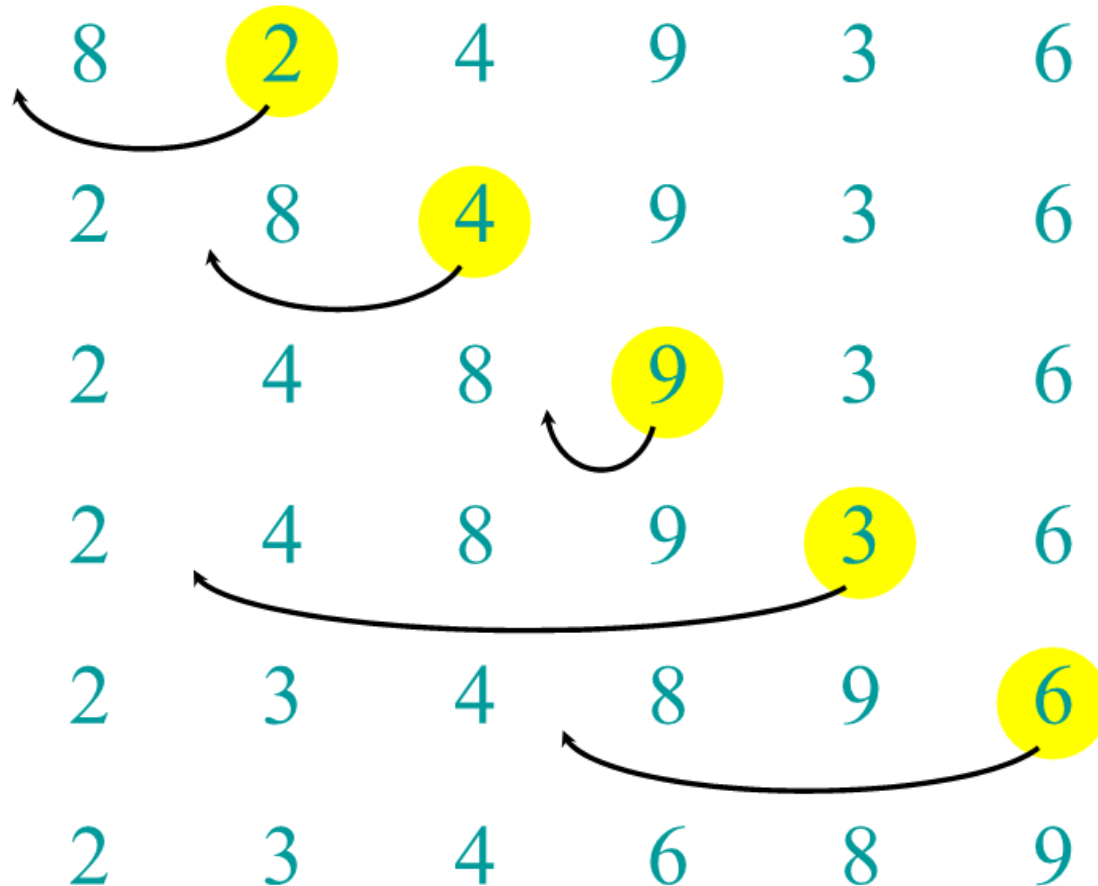
Sıralama (sorting) problemi

- **Girdi:** dizi $\langle a_1, a_2, \dots, a_n \rangle$ sayıları.
- **Çıktı:** $\langle a'_1, a'_2, \dots, a'_n \rangle$ elemanları a_1, a_2, \dots, a_n elemanlarının bir permütasyonudur ve $a'_1 \leq a'_2 \leq \dots \leq a'_n$.
- 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

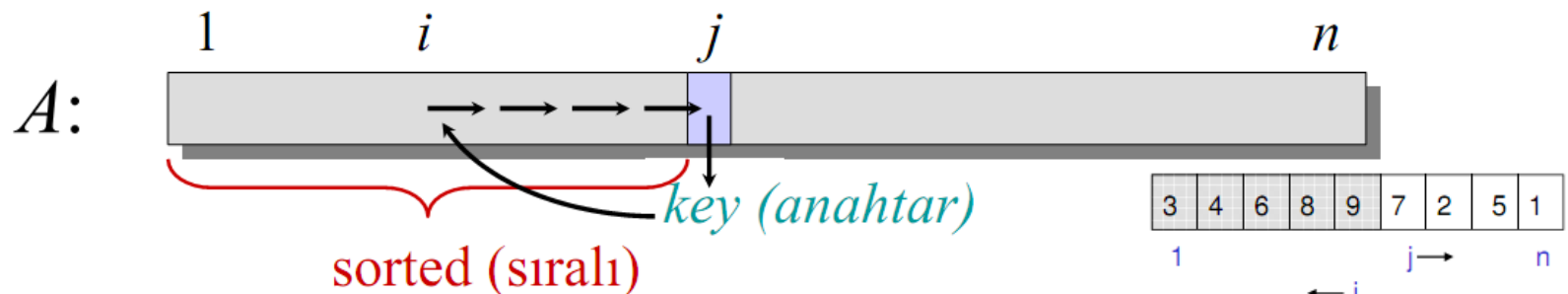


Araya yerleştirme sıralaması (Insertion sort)

“pseudocode”
(sözde kod)

```

INSERTION-SORT ( $A, n$ )    ▷  $A[1 \dots n]$ 
  for  $j \leftarrow 2$  to  $n$ 
    do  $key \leftarrow A[j]$ 
       $i \leftarrow j - 1$ 
      while  $i > 0$  and  $A[i] > key$ 
        do  $A[i+1] \leftarrow A[i]$ 
           $i \leftarrow i - 1$ 
       $A[i+1] = key$ 
  
```



Örnek: Insertion Sort

30	10	40	20
1	2	3	4

$i = \emptyset$	$j = \emptyset$	$key = \emptyset$
$A[j] = \emptyset$	$A[j+1] = \emptyset$	

➔

```

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

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

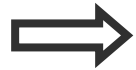


```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

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

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 30$	

```

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

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 30$	

```

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

$i = 2$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 10$	

```

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

$i = 3$	$j = 0$	$key = 10$
$A[j] = \emptyset$	$A[j+1] = 10$	



```

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

$i = 3$	$j = 0$	$key = 40$
$A[j] = \emptyset$	$A[j+1] = 10$	



```

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

$i = 3$	$j = 0$	$key = 40$
$A[j] = \emptyset$	$A[j+1] = 10$	



```

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

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



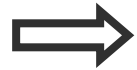
```

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

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



```
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	20
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	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

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



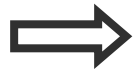
```

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 bağlıdır: 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

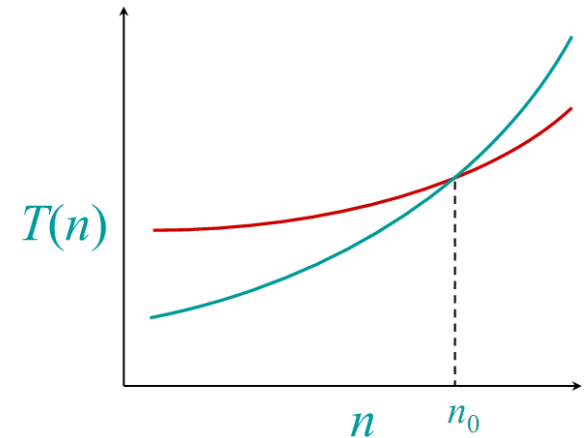
Makineden-bağımsız zaman

- 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 başarımlar

- 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 //

	<u>cost</u>	<u>times</u>
1 for $j \leftarrow 2$ to $\text{length}[A]$	c_1	n
2 do $\text{key} \leftarrow A[j]$	c_2	$n-1$
// insert $A[j]$ into the sorted sequence $A[1..j-1]$		
3 $i \leftarrow j-1$	c_4	$n-1$
4 while $i > 0$ and $A[i] > \text{key}$	c_5	$\sum_{2 \leq j \leq n} t_j$
5 $A[i+1] \leftarrow A[i]$	c_6	$\sum_{2 \leq j \leq n} (t_j - 1)$
6 $i \leftarrow i-1$	c_7	$\sum_{2 \leq j \leq n} (t_j - 1)$
done		
7 $A[i+1] \leftarrow \text{key}$	c_8	$n-1$
done		

Insertion sort algoritmasının analizi

	<u>cost</u>	<u>times</u>
// sort in increasing order //		
1 for $j \leftarrow 2$ to $\text{length}[A]$	c_1	n
2 do $\text{key} \leftarrow A[j]$	c_2	$n-1$
// insert $A[j]$ into the sorted sequence $A[1..j-1]$		
3 $i \leftarrow j-1$	c_4	$n-1$
4 while $i > 0$ and $A[i] > \text{key}$	c_5	$\sum_{2 \leq j \leq n} t_j$
5 $A[i+1] \leftarrow A[i]$	c_6	$\sum_{2 \leq j \leq n} (t_j - 1)$
6 $i \leftarrow i-1$	c_7	$\sum_{2 \leq j \leq n} (t_j - 1)$
done		
7 $A[i+1] \leftarrow \text{key}$	c_8	$n-1$
done		

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

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

Araya yerleştirme sıralaması

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

$$T(n) = c_1n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1)$$

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

$$T(n) = an + b, \text{ Lineer fonksiyon}$$

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

Araya yerleştirme sıralaması

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

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

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

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

$$T(n) = an^2 + bn + c \quad (\text{Quadratic Function})$$

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

Araya yerleştirme sıralaması analizi özet

- En kötü durum: Giriş tersten sıralıysa.

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

- Ortalama durum: Tüm permutasyonlar eşit olasılıklı.

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

- 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:

$$1. \quad f(n) = \sum_{1 \leq i \leq n} i = n(n+1)/2$$

$$2. \quad f(n) = \sum_{1 \leq i \leq n} i^2 = n(n+1)(2n+1)/6$$

$$3. \quad f(n) = \sum_{0 \leq i \leq n} x^i = (x^{n+1} - 1) / (x - 1)$$

$$4. \quad f(n) = \sum_{0 \leq i \leq n} 2^i = (2^{n+1} - 1) / (2 - 1) = 2^{n+1} - 1$$

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

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

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

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-1

```
for k = 1 to n/2 do
```

```
{
```

```
....
```

—————→ c_1

```
}
```

```
for j = 1 to n*n do
```

```
{
```

```
----- ....
```

—————→ 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
  {
    ----
  }
}

```

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

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-3

	<u>Cost</u>	<u>Times</u>
<code>i = 1;</code>	c_1	1
<code>sum = 0;</code>	c_2	1
<code>while (i <= n) {</code>	c_3	$n+1$
<code>i = i + 1;</code>	c_4	n
<code>sum = sum + i; }</code>	c_5	n

$$\begin{aligned}
 T(n) &= c_1 + c_2 + c_3(n+1) + c_4n + c_5n \\
 &= (c_3 + c_4 + c_5)n + (c_1 + c_2 + c_3)
 \end{aligned}$$

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

$T(n) \rightarrow$ Bu algoritmanın büyüme oran fonksiyonu, $O(n)$ dir

Büyüme oranı (Growth-Rate) fonksiyonları

Örnek-4

	<u>Cost</u>	<u>Times</u>
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
}		

$$\begin{aligned}
 T(n) &= c_1 + c_2 + c_3(n+1) + c_4n + c_5n(n+1) + c_6n(n) + c_7n(n) + c_8n \\
 &= (c_5+c_6+c_7)n^2 + (c_3+c_4+c_5+c_8)n + (c_1+c_2+c_3) \\
 &= an^2 + bn + c
 \end{aligned}$$

➔ Bu algoritmanın büyüme oran fonksiyonu, **O(n²)** 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) ← $0 + 1 + 2 + \dots + (N - 1) = \frac{1}{2} N (N - 1)$
- count++; $= \binom{N}{2}$

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

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

$$\begin{aligned}
 T(n) &= c_1(n+1) + c_2 \sum_{j=1}^n (j+1) + c_3 \sum_{j=1}^n \sum_{k=1}^j (k+1) + c_4 \sum_{j=1}^n \sum_{k=1}^j k \\
 &= an^3 + bn^2 + cn + d
 \end{aligned}$$

→ 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
  
```

$$\begin{aligned}
 T(n) &= \sum_{i=1}^n \sum_{j=i}^n \sum_{k=i}^j 3 \\
 &= \sum_{i=1}^n \sum_{j=i}^n 3(j-i+1) = \sum_{i=1}^n 3 \left[\sum_{j=i}^n j - \sum_{j=i}^n i + \sum_{j=i}^n 1 \right] \\
 &= \sum_{i=1}^n 3 \left[\left(\sum_{j=1}^n j - \sum_{j=1}^{i-1} j \right) - i(n-i+1) + (n-i+1) \right] \\
 &= \sum_{i=1}^n 3 \left[[n(n+1)/2 - i(i-1)/2] - i(n-i+1) + (n-i+1) \right] \\
 &\approx n^3/10 \text{ additions, that is, } O(n^3)
 \end{aligned}$$



Özyinelemeli Algoritmaların Analizi

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

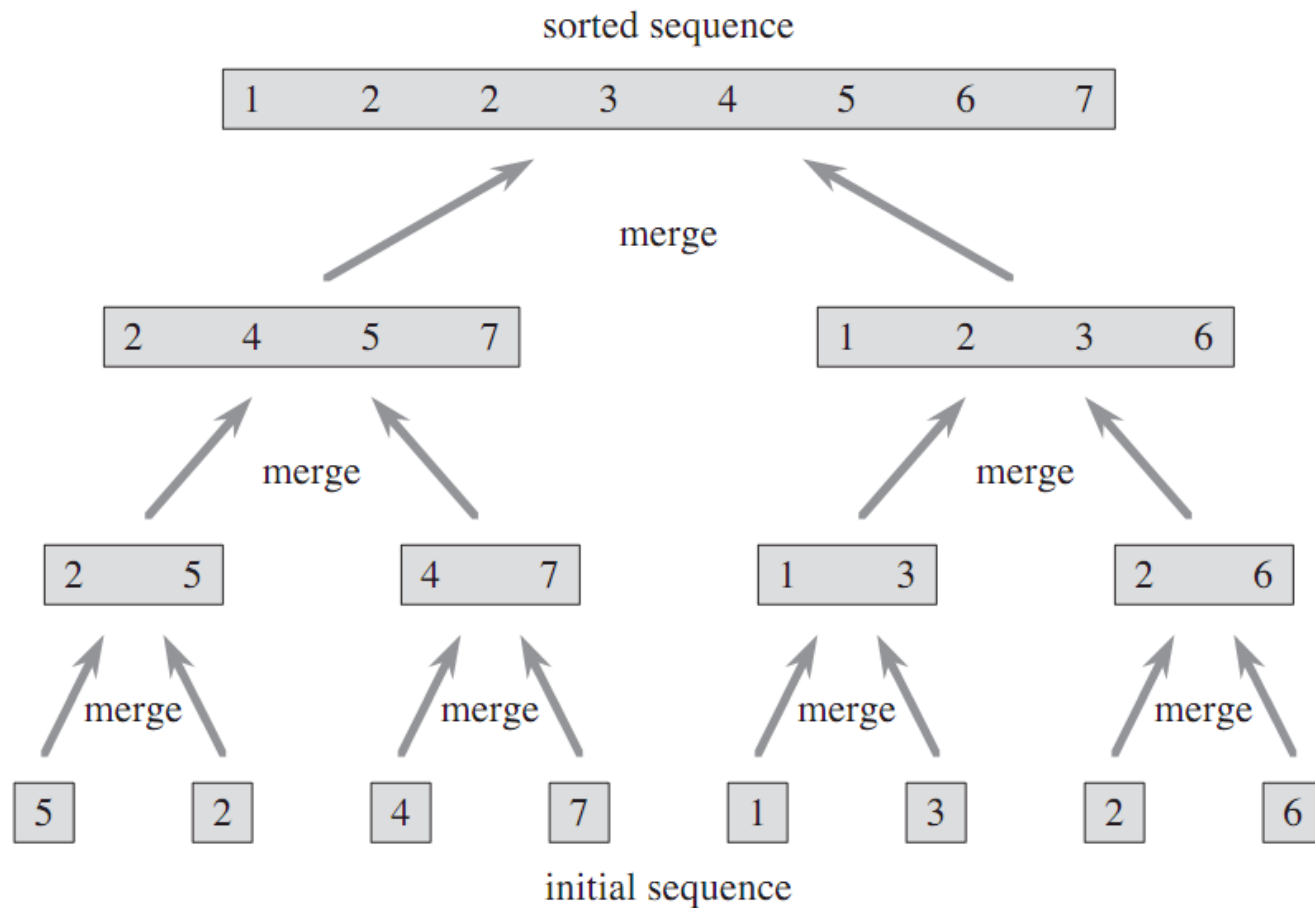
Hatırlatma-Özyinelemeli Tanımlar

- 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.
 - İkinci 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 $A[1 \dots n]$

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

Anahtar altyordam: Birleştirme

Birleştirme sıralaması-Merge Sort

MERGE-SORT(A, p, r)

if $p < r$

$q = \lfloor (p + r) / 2 \rfloor$

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)

$n_1 = q - p + 1$

$n_2 = r - q$

let $L[1..n_1 + 1]$ and $R[1..n_2 + 1]$ be new arrays

for $i = 1$ **to** n_1

$L[i] = A[p + i - 1]$

for $j = 1$ **to** n_2

$R[j] = A[q + j]$

$L[n_1 + 1] = \infty$

$R[n_2 + 1] = \infty$

$i = 1$

$j = 1$

for $k = p$ **to** r

if $L[i] \leq R[j]$

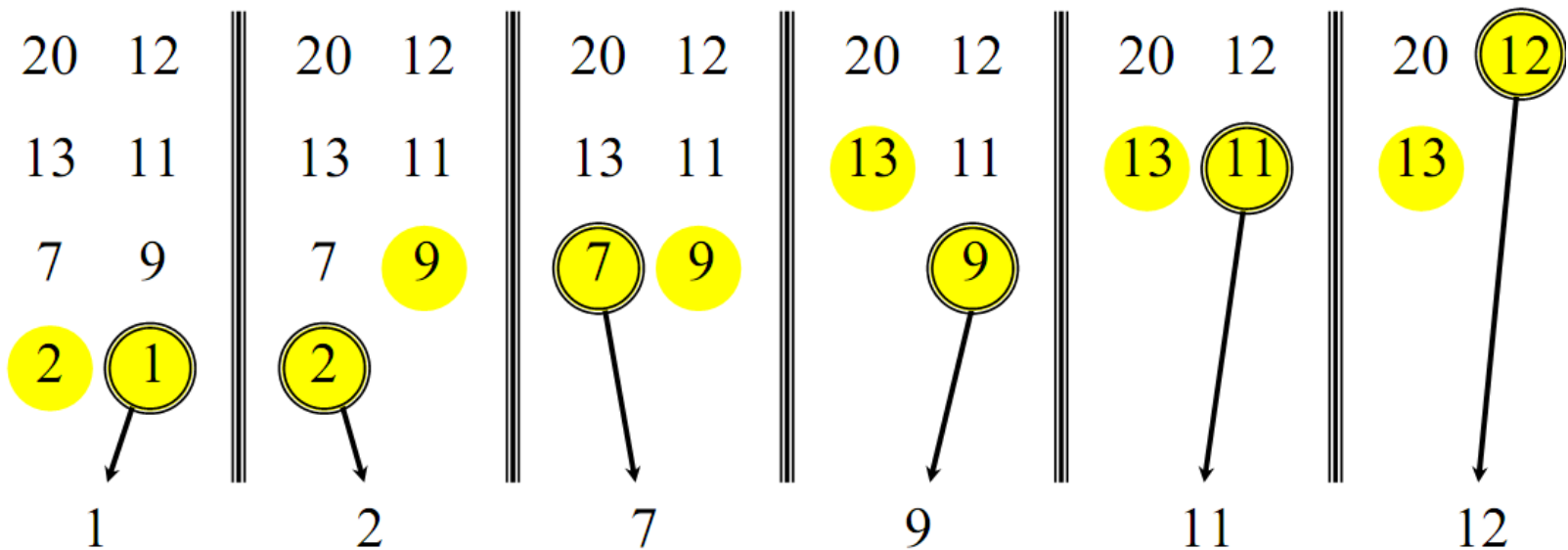
$A[k] = L[i]$

$i = i + 1$

else $A[k] = R[j]$

$j = j + 1$

Sıralı iki dizilimi birleştirme



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

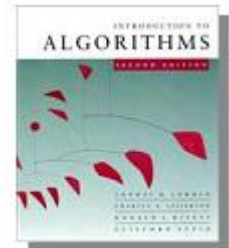
Birleştirme sıralaması-Merge Sort

- MergeSort(A, left, right) { $T(n)$
- if (left < right) { $\Theta(1)$
- mid = floor((left + right) / 2); $\Theta(1)$
- MergeSort(A, left, mid); $T(n/2)$
- MergeSort(A, mid+1, right); $T(n/2)$
- Merge(A, left, mid, right); $\Theta(n)$
- }
- }

- 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- $f(n)=n^2+4n\log n+900$ ve $g(n)=n^2+45\log n+h(n)$ 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?

$$1. \quad T(n) = T\left(\frac{n}{2}\right) + \sqrt{n} + \sqrt{6046} = O(\sqrt{n})$$

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

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

Bazı Matematiksel İfadeler

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

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

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

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

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

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

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

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

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

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

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

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

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

Bazı Önemli Matematiksel İfadeler

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

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

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

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

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

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

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

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

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.
$$\sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=m+1}^u a_i, \text{ where } l \leq m < u$$
4.
$$\sum_{i=l}^u (a_i - a_{i-1}) = a_u - a_{l-1}$$

Approximation of a Sum by a Definite Integral

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

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

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. $x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$
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 $n \rightarrow \infty$ (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