



2.Hafta

Algoritmaların Analizi

Araya Yerleştirme Sırlaması
(Insert Sort)

Birleştirme Sıralaması (Merge Sort)
Yinelemeler

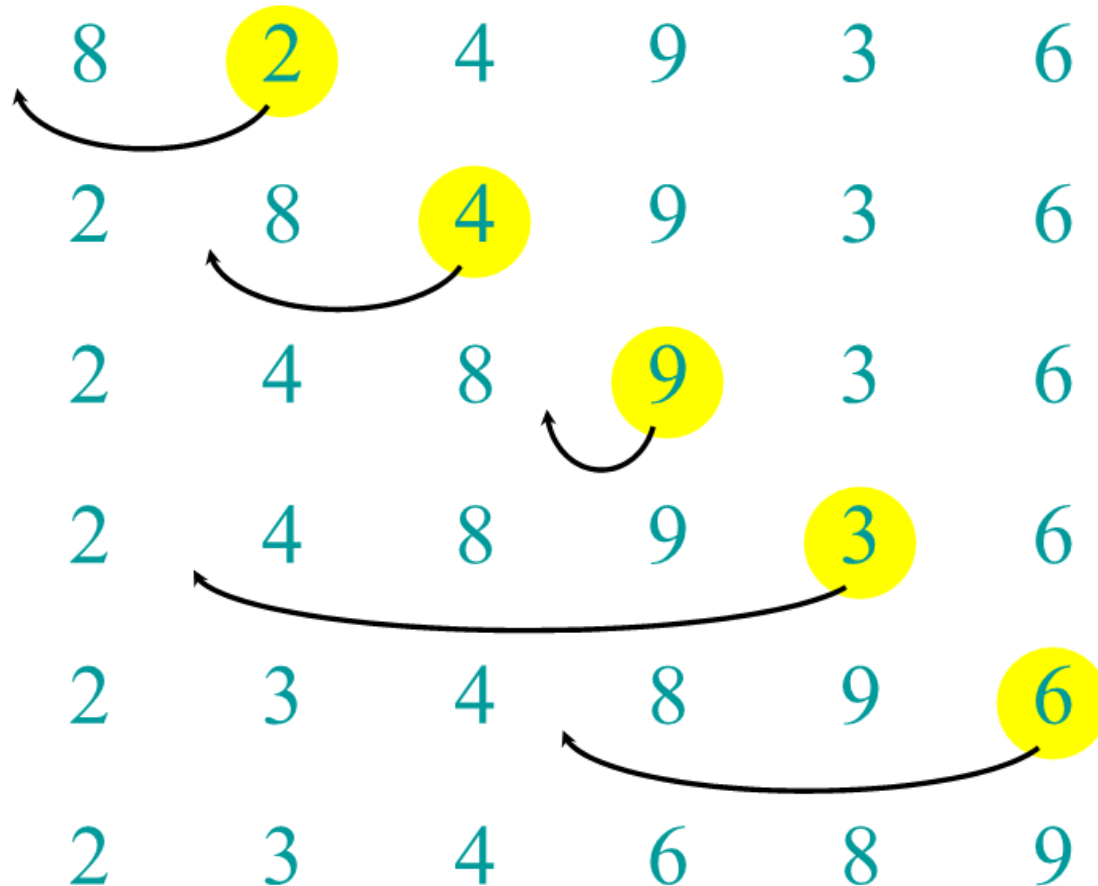
Sıralama (sorting) problemi

- **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

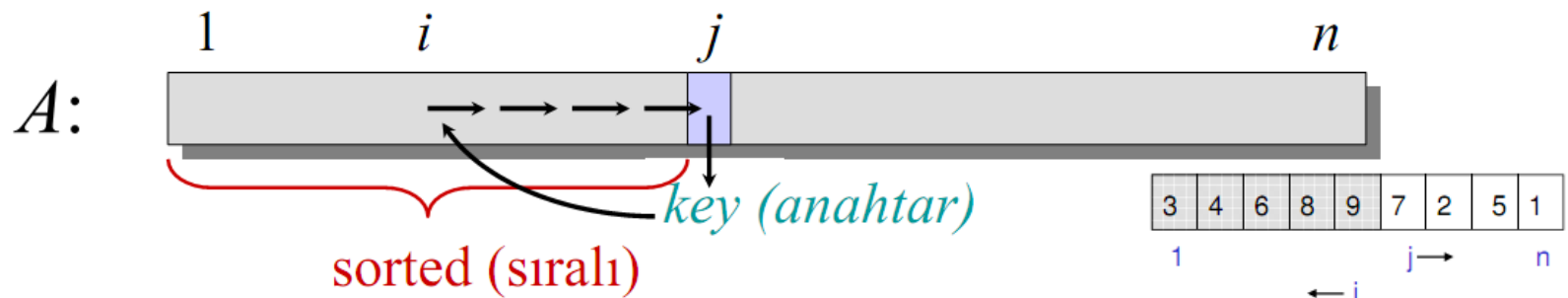


Araya yerleştirme sıralaması (Insertion sort)

“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 |

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

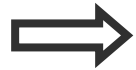


```
InsertionSort(A, n) {  
    for i = 2 to n {  
        key = A[i]  
        j = i - 1;  
        while (j > 0) and (A[j] > key) {  
            A[j+1] = A[j]  
            j = j - 1  
        }  
        A[j+1] = key  
    }  
}
```

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 |

| |
|---|
| $i = 2$ $j = 0$ $\text{key} = 10$ $A[j] = \emptyset$ $A[j+1] = 10$ |
|---|

```

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 |

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



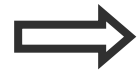
```

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 |

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



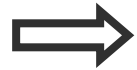
```

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 |

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



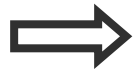
```

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

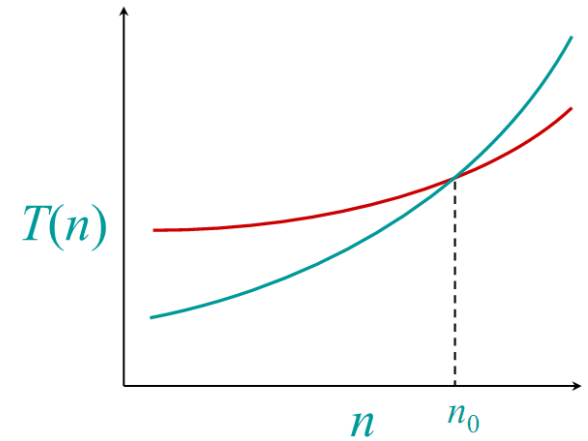
Makineden-bağımsız zaman

- 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

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

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

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

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

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> |
|----------------------------------|-------------|--------------|
| <code>i=1;</code> | c_1 | 1 |
| <code>sum = 0;</code> | c_2 | 1 |
| <code>while (i <= n) {</code> | c_3 | $n+1$ |
| <code>j=1;</code> | c_4 | n |
| <code>while (j <= n) {</code> | c_5 | $n*(n+1)$ |
| <code>sum = sum + i;</code> | c_6 | $n*n$ |
| <code>j = j + 1;</code> | c_7 | $n*n$ |
| <code>}</code> | | |
| <code>i = i + 1;</code> | c_8 | n |
| <code>}</code> | | |

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

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

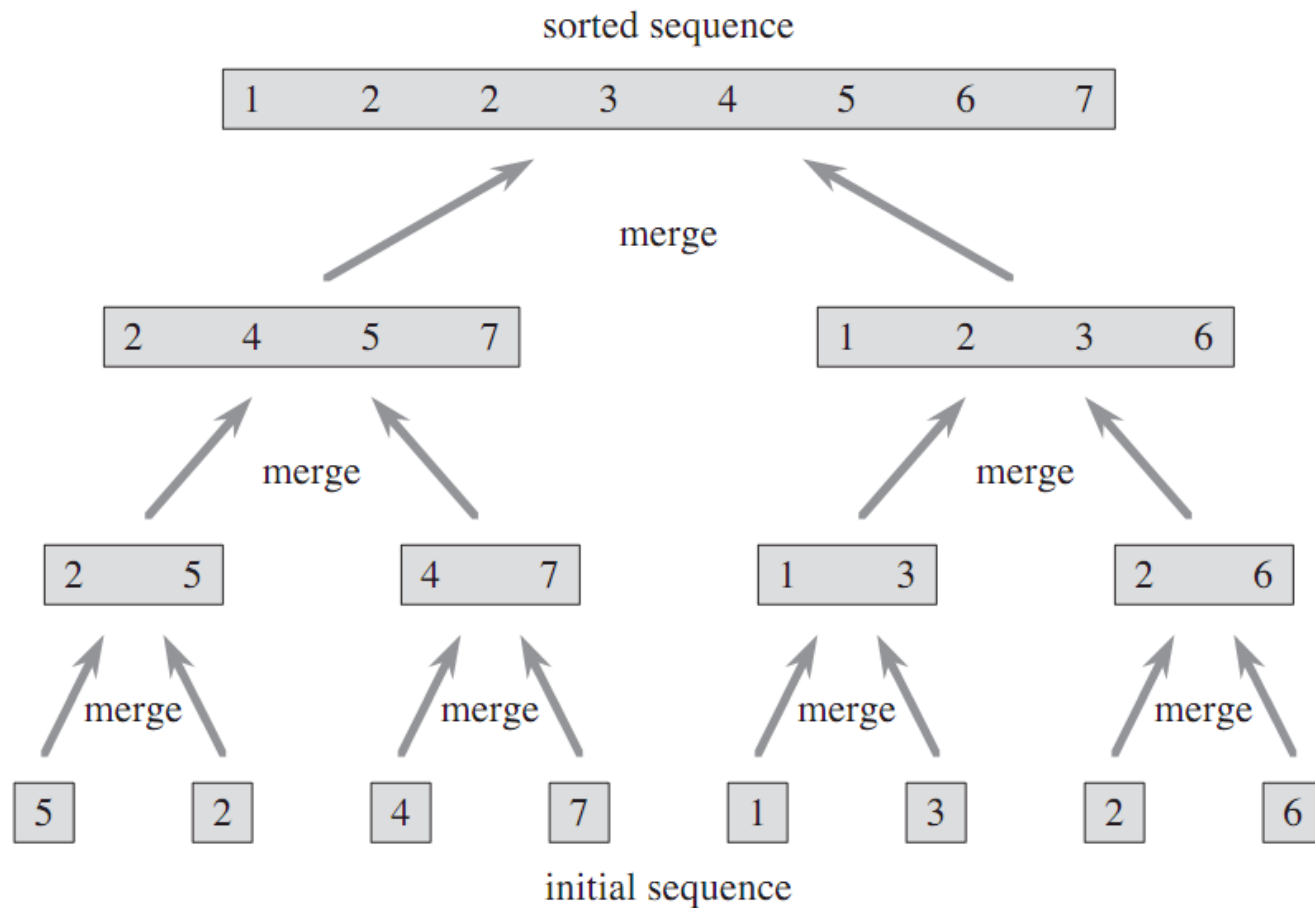
Hatırlatma-Özyinelemeli Tanımlar

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

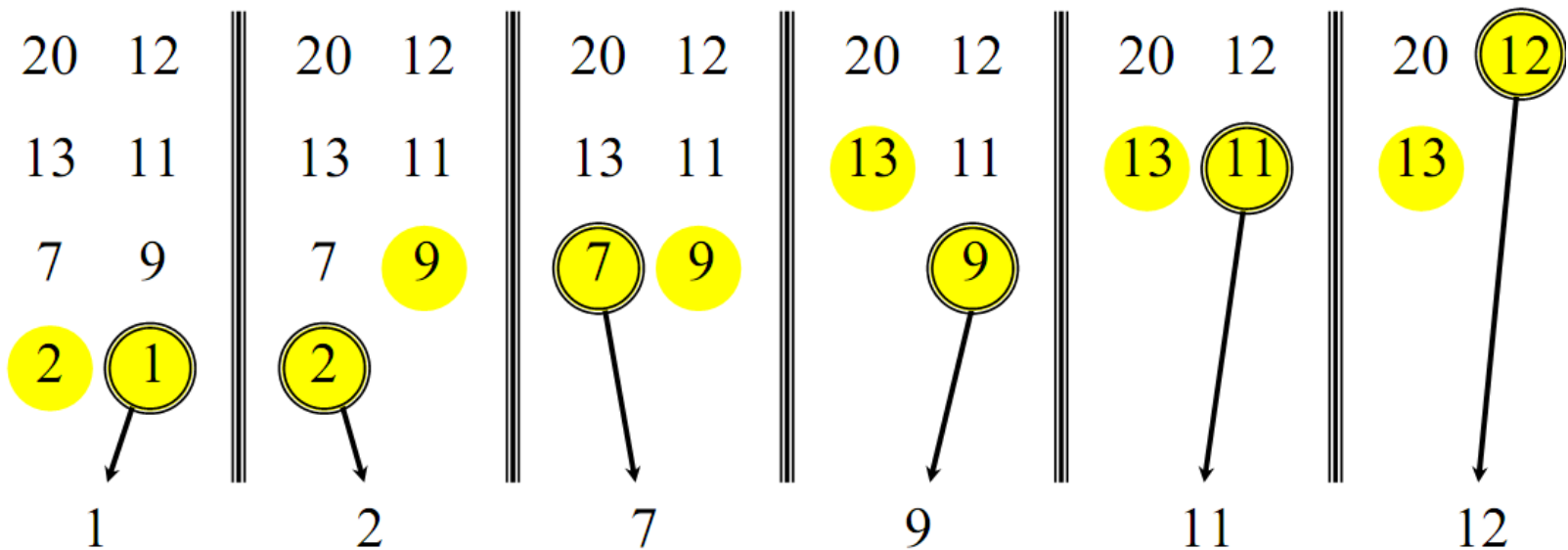
$A[k] = L[i]$

$i = i + 1$

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.

Yinelemelerin çözümü

- Yinelemeler entegral, türev, v.s. denklemlerinin çözümlerine benzer. Yinelemeleri çözmek konusunda kullanacağımız 3 ana yöntem vardır. Bu yöntemler;
 - **Yerine koyma metodu (substitution)**
 - **İterasyon (Yineleme) metodu (iteration metod)**
 - Özyineleme ağacı (recursion tree)
 - **Ana Metot (Master metod)**
 - Ana yöntemlerin dışında tekrarlı bağıntılar **Karakteristik denklemler kullanarak** çözülebilir.

Yerine koyma metodu (yöntemi)

- Bazı tekrarlı bağıntıların çözümü yapılmadan çözümünün nasıl olabileceği hakkında tahmin yapılabilir. Çözüm tahmini yapılır ve yerine konulur. Yerine koyma (tahminler) metodu bir sınırdır ve tahmini ispatlamak için tümevarım yöntemi kullanılır.
- En genel yöntem:
 - 1. Çözümün şeklini **tahmin** edin.
 - 2. Tümevarım ile **doğrulayın**.
 - 3. Sabitleri **çözün**.
- Örnek: $T(n)=T(n/2)+c$, $n \geq 2$, ve $T(1)=1$.
- $T(2)=1+c$, $T(4)=1+2c$, $T(8)=1+3c$, ...
- $T(2^k)=1+kc$, burada $n=2^k$, $T(n)=1+c \log n$ olur.

Yerine koyma metodu (yöntemi)

- **Tümevarım ile İspat:**
- Temel durum: $n=1$, $T(1)=1$ verildi.
- Tümevarım hipotezi: $n=2^k$, $T(2^k)=1+kc$, burada $n=2^k$,
 $T(n)=T(n/2)+c$ için doğrudur.
- Tümevarım adımı: $n=2^{k+1}$, $T(n+1)=1+(k+1)*c = (1+kc)+c$
- İspat (proof):
- $T(n+1)=T(2^{k+1})=T(2^{k+1}/2)+c$
- $T(n+1)=T(2^k*2/2)+c$
- $=T(2^k) + c = 1+kc + c$, olur. Tümevarım adımı ispatlanmış olunur.
- Burada, $n=2^k$ için, $T(n)=1+c\log n$ ve
- $T(n) \in O(\log n)$ dir.

Yerine koyma metodu (yöntemi)

- Örnek: $T(n)=3T(n/2)+cn$, $n \geq 2$, ve $T(1)=1$.
- $T(2)=3+2c$
- $T(4)=3(T(2))+4c=3(3+2c)+4c=9+10c=3^2+[3^1 2^1 c]+3^0 2^2 c$
- $T(8)=27+38c=3^3+[3^2 2^1 c+3^1 2^2 c]+3^0 2^3 c$
- $T(16)=81+130c$
- ...
- $T(2^k)=3^k+[3^{k-1} 2^1 c+3^{k-2} 2^2 c+\dots+3^1 2^{k-1} c]+3^0 2^k c$, $2^k c$ parantezine alalım
- $T(2^k)=3^k+2^k c[(3/2)^{k-1}+(3/2)^{k-2}+\dots+(3/2)]$, serisinin genel denklemi
- $T(2^k)=3^k+2^k c[((3/2)^k-1)/((3/2)-1)]$, burada $n=2^k$ ve $k=\log n$
- $T(n)=3^{\log n}+cn[((3^{\log n}/n-1)/(1/2))]$, burada $a^{\log_b x} = x^{\log_b a}$
- $T(n)=n^{\log 3}+2cn(n^{\log 3-1}-1) = n^{1,59}+2cn(n^{0,59}-1)$
- $T(n)=n^{1,59}(1+2c)-2cn$
- $T(n) \in O(n^{1,59})$ dır.

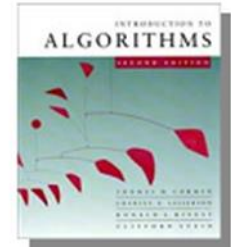
$$f(n) = \sum_{0 \leq i \leq n} x^i = (x^{n+1} - 1) / (x - 1)$$

Yerine koyma metodu (yöntemi)

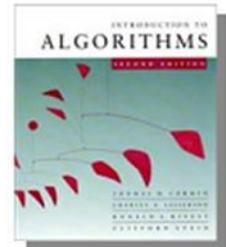
- **İspat:** $T(n)=3T(n/2)+cn$, $n \geq 2$, ve $T(1)=1$. (1)
- $T(2^k)=3^k+2^k c [((3/2)^k-1)/((3/2)-1)]$, burada $n=2^k$ ve $k=\log n$ (2)
- Temel durum: $n=1$, $T(1)=1$ ve $n=2 \rightarrow T(2)=3+2c$
- Tümevarım hipotezi: $n=2^k \rightarrow T(2^k)=3^k+2^k c [((3/2)^k-1)/((3/2)-1)]$
- Tümevarım adımı: $n=2^{k+1} \rightarrow T(2^{k+1})=3^{k+1}+2^{k+1} c [((3/2)^{k+1}-1)/((3/2)-1)]$ veya
 $\rightarrow T(2^{k+1})=3^{k+1}+2^{k+2} c [(3/2)^{k+1}-2]$ (3)
- Şimdi (1) nolu denklemi kullanarak (3) nolu denklemi ispatlayalım
- $T(2^k)=3^k.T(2^{k-1})+c2^k \rightarrow n=2^k$ için
- $T(2^{k+1})=3.T(2^k)+c2^{k+1} = 3.(3^k+c2^k [((3/2)^k-1)/((3/2)-1)])+c2^{k+1}$
- $T(2^{k+1})=3^{k+1}+2^k c [3.(3/2)^k-3]/(1/2) + 2^{k+1}c = 3^{k+1}+2^{k+1} c [3.(3/2)^k-3] + 2^{k+1}c$
- $T(2^{k+1})=3^{k+1}+2^{k+1} c [3.(3/2)^k-3 + 1] = 3^{k+1}+2^{k+1} c [(3/2)^{k+1}.2-2]$
- $T(2^{k+1})=3^{k+1}+2^{k+2} c [(3/2)^{k+1}-2]$, (3) nolu denklemi ispatlamış oluyoruz.
- $T(n) \in O(n^{1,59})$ dır.

Yerine koyma metodu (yöntemi)

Üst sınırı tahmin ederek çözüm



- Örnek: $T(n) = 4T(n/2) + n$, ise
 - [$T(1) = \Theta(1)$ olduğunu varsayın.]
 - $O(n^3)$ 'ü tahmin edin. (O ve Ω ayrı ayrı kanıtlayın.)
 - $k < n$ için $T(k) \leq ck^3$ olduğunu varsayın.
 - $T(n) \leq cn^3$ 'ü tümevarımla kanıtlayın.



Yerine koyma örneği

$$T(n) \leq cn^3$$

$$T(n) = 4T(n/2) + n$$

$$\leq 4c(n/2)^3 + n$$

$$= (c/2)n^3 + n$$

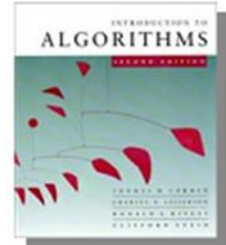
$$= cn^3 - ((c/2)n^3 - n) \leftarrow \text{istenen} - \text{kalan}$$

$$\leq cn^3 \leftarrow \text{istenen}$$

ne zaman ki $(c/2)n^3 - n \geq 0$, örneğin,
eğer $c \geq 2$ ve $n \geq 1$.

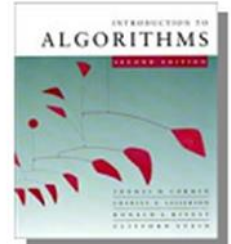
kalan

Yerine koyma örneği



- Başlangıç koşullarını da ele almalı, yani, tümevarımı taban şıklarına (base cases) dayandırmalıyız.
-
- **Taban:** $T(n) = \Theta(1)$ tüm $n < n_0$ için, ki n_0 uygun bir sabittir.
- $1 \leq n < n_0$ için, elimizde “ $\Theta(1)$ ” $\leq cn^3$, olur; yeterince büyük bir c değeri seçersek.

Bu, sıkı bir sınır değildir !

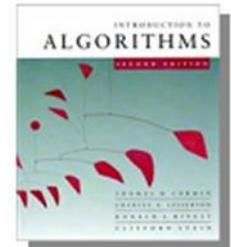


Yerine koyma örneği-Daha sıkı bir üst sınır?

$T(n) = O(n^2)$ olduğunu kanıtlayacağız.

Varsayın ki $T(k) \leq ck^2$, $k < n$ için olsun :

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^2 + n \\ &= cn^2 + n \\ &= O(n^2) \end{aligned}$$



Yerine koyma örneği-Daha sıkı bir üst sınır?

$T(n) = O(n^2)$ olduğunu kanıtlayacağız.

Varsayın ki $T(k) \leq ck^2$; $k < n$: için

$$\begin{aligned} T(n) &= 4T(n/2) + n \\ &\leq 4c(n/2)^2 + n \\ &= cn^2 + n \end{aligned}$$

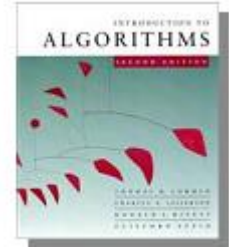
~~$= O(n^2)$~~ Yanlış! I.H.(tümevarım hipotezini) kanıtlamalıyız.

$$= cn^2 - (-n) \quad [\text{istenen} - \text{kalan}]$$

$\leq cn^2$ seçeneksiz durum $c > 0$. Kaybettik!

$-n \geq 0$ sağlanmaz (n değeri negatif olamaz)

Yerine koyma örneği-Daha sıkı bir üst sınır?



FİKİR: Varsayım hipotezini güçlendirin.

- Düşük-düzeyli bir terimi *çıkartın*.

Varsayım hipotezi: $T(k) \leq c_1 k^2 - c_2 k$; $k < n$ için.

$$\begin{aligned}
 T(n) &= 4T(n/2) + n \\
 &= 4(c_1(n/2)^2 - c_2(n/2)) + n \\
 &= c_1 n^2 - 2c_2 n + n \\
 &= c_1 n^2 - c_2 n - (c_2 n - n) \\
 &\leq c_1 n^2 - c_2 n \text{ eğer } c_2 \geq 1.
 \end{aligned}$$

c_1 'i başlangıç koşullarını karşılayacak kadar büyük seçin.

Yerine koyma metodu (yöntemi)

- $T(n)=3T(n/2)+cn$,
- Tahminimiz $=T(k) \in O(n^{1,59})$, $n \geq 2$, ve $T(1)=1$.
- İspatı yapabilmek için tahminimizi $T(n)$ de yerine yazıp istediğimizi elde etmeye çalışmak. **İstenen –kalan**. Kalan, 0'a eşit yada büyük olmalı
- $T(n) \leq c_1 \cdot n^{1,59}$ olması
- $T(n)=3 \cdot (c_1 n^{1,59}/2^{1,59})+cn$
- $T(n)=c_1 n^{1,59} - c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}+cn$
- **İstenen** $=cn^{1,59}$, **Kalan** $= - c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}+cn$
- **İstenen –kalan** $=c_1 n^{1,59} - (c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}-cn)$
- **Kalan** $\rightarrow c_1 n^{1,59}(2^{1,59}+3)/2^{1,59}-cn \geq 0$ sağlayacak c , ve n değeri var mı?
- $n=1$ için, $c_1(2^{1,59}+3)/2^{1,59}-c \geq 0$, $c \geq 1$,ve $c_1 \geq 2$, için sağlarız

Yerine koyma metodu (yöntemi)

● Hanoi kulesi

| | |
|---|-------------|
| <code>void hanoi(int n, char source, char dest, char spare) {</code> | <u>Cost</u> |
| <code>if (n > 0) {</code> | c1 |
| <code>hanoi(n-1, source, spare, dest);</code> | c2 |
| <code>cout << "Move top disk from pole " << source</code> | c3 |
| <code><< " to pole " << dest << endl;</code> | |
| <code>hanoi(n-1, spare, dest, source);</code> | c4 |
| <code>}}</code> | |

when n=0

$$T(0) = c1$$

when n>0

$$T(n) = c1 + c2 + T(n-1) + c3 + c4 + T(n-1)$$

$$= 2 * T(n-1) + (c1 + c2 + c3 + c4)$$

$$= 2 * T(n-1) + c \quad \leftarrow \text{recurrence equation for the growth-rate function of hanoi-towers algorithm}$$

Yerine koyma metodu (yöntemi)

- Hanoi kulesi
- $T(n) = 2T(n-1) + 1$ kabul edersek en iyi durumda $T(0)=0$ dır.
- $T(0) = 0, T(1) = 1, T(2) = 3, T(3) = 7, T(4) = 15, T(5) = 31, T(6) = 63, \dots$
- $T(0)=0$
- $T(1)=2T(0)+1=1$
- $T(2)=2T(1)+1=2+1$
- $T(3)=2T(2)+1=2.(2+1)+1=1+2+2^2$
- $T(4)=2T(3)+1=1+2+2^2+2^3$
- $T(n)=2T(n-1)+1=1+2+\dots+2^{n-1}$ $f(n) = \sum_{0 \leq i \leq n} 2^i = (2^{n+1}-1) / (2-1) = 2^{n+1}-1$
- $T(n)=2^n-1$
- olur

İterasyon metodu (Tümden gelim)

- Bu yöntemde verilen bağıntının çözümünü bulmak için büyük endeksli terimin yerine küçük endeksli terim yazılarak, genel terimin çözümü için bir yargıya varılıncaya kadar bu işleme devam edilir veya başlangıç şartlarına kadar devam edilir.
- İterasyon metodu, bir toplam içerisine yinelemeleri dönüştürür ve yinelemeleri çözmek için toplamaları sınırlayıcı teknikleri kullanır.
 - Yineleme işlemini açık hale getir
 - Matematiksel işlemlerle göster
 - Toplamı hesapla.

$$T(n) = \begin{cases} \text{denemeyle çözülür, } n=1 \\ \text{Alt problem sayısı} * T(n/\text{alt problem boyutu}) + \text{bölme} + \text{birleşim, } n=1 \end{cases}$$

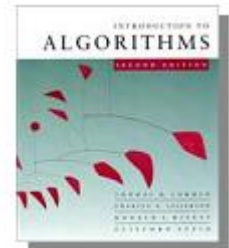
■ Örnek

$$T(n) = \begin{cases} \Theta(1) & \text{if } n = 1 \\ 2T(n/2) + \Theta(n) & \text{if } n > 1 \end{cases}$$

İterasyon metodu

- Örnek: Merge Sort, $T(n)=2*T(n/2)+n$, $n>1$, ve $T(1)=1$ için yinelemeyi çözünüz
- $n=2^k \rightarrow k = \log n$ hatırlayın,
 - $T(n)=2*T(n/2)+n$ substitute (yerine kullan)
 - $T(n)=2*(2*T(n/4)+n/2)+n$ expand
 - $T(n)=2^2*T(n/4)+2*n$ substitute
 - $T(n)=2^2*(2*T(n/8)+n/4)+2n = 8*T(n/8)+3*n$ expand
 - $T(n)=2^3*T(n/2^3)+3*n$ observe
 -
 - $T(n)=2^k*T(n/n)+k*n$
 - $T(n)=2^k * T(1)+k*n$
 - $T(n)=\theta(n)+ \theta(n\log n)$
 - $T(n) \in \theta(n\log n)$

İterasyon metodu

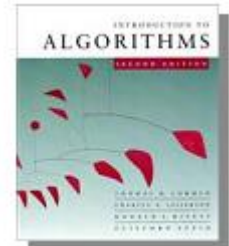


- Merge sort için daha kesin çalışma süresi bulma (bazı b değerleri için $n=2^b$ olduğunu varsayalım).

$$T(n) = \begin{cases} 2 & \text{if } n = 1 \\ 2T(n/2) + 2n + 3 & \text{if } n > 1 \end{cases}$$

$$T(n) = 5n + 2n \lg n - 3$$

iterasyon metodu



$$\begin{aligned}
 T(n) &= 2 T(n / 2) + 2n + 3 \\
 &= 2 (2 T(n / 4) + n + 3) + 2n + 3 \\
 &= 2^2 T(n / 4) + 4n + 2 \cdot 3 + 3 \\
 &= 2^2 T(n / 2^2) + 2^2 n + 2^1 \cdot 3 + 2^0 \cdot 3 \\
 &= 2^2 (2 T(n / 8) + n / 2 + 3) + 4n + 2 \cdot 3 + 3 \\
 &= 2^3 (T(n / 2^3) + 2 \cdot 3n + (2^2 + 2^1 + 2^0) \cdot 3) \\
 &= 2^b T(n / 2^b) + 2 \cdot b n + 3 \sum_{j=0}^{b-1} 2^j \\
 &= n T(n / n) + 2n \lg n + 3(2^b - 1) \\
 &= 2n + 2n \lg n + 3n - 3 \\
 &= 5n + 2n \lg n - 3
 \end{aligned}$$

İterasyon metodu

- Örnek: $T(n)=T(n-1)+n$, $n>1$, ve $T(1)=1$; için yinelemeyi çözünüz?
-
- $T(n)=T(n-1) + n$
- $T(n)=T(n-2) + n-1+ n$
- $T(n)=T(n-3) + n-2+ n-1+ n$
- ...
- $T(n)=T(1) + 2+ n-2+ n-1+ n$
- $T(n)=1 + 2+ n-2+ n-1+ n \longrightarrow f(n) = \sum_{1 \leq i \leq n} i = n(n+1)/2$
- $T(n)=n*(n+1)/2$
- $T(n) \in \theta(n^2)$

İterasyon metodu

- Hanoi Kulesi
- $T(n)=2*T(n-1)+1$, $n>1$, ve $T(1)=1$; için yinelemeyi çözünüz?
- Her bir hareket $O(1)$ zaman gerektirir.


○

- $T(n)=2*T(n-1)+1=2^2 * T(n-2)+2+1 = \dots$

- $T(n)=2^{n-1} * T(1)+2^{n-2}+\dots+2+1$

- $T(n)=2^n-1$

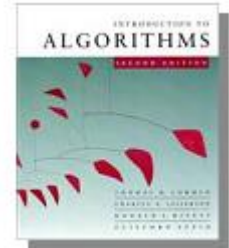
- $T(n) \in \theta(2^n-1)$



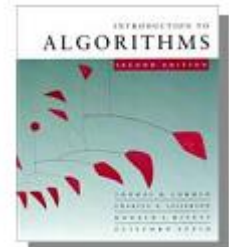
$$f(n) = \sum_{0 \leq i \leq n} 2^i = (2^{n+1}-1) / (2-1) = 2^{n+1}-1$$

- $n=64$ için, her 1000 taşımanın 1 sn olduğu düşünülürse çözümü $584*10^6$ yıl alır.

Özyineleme-ağacı metodu

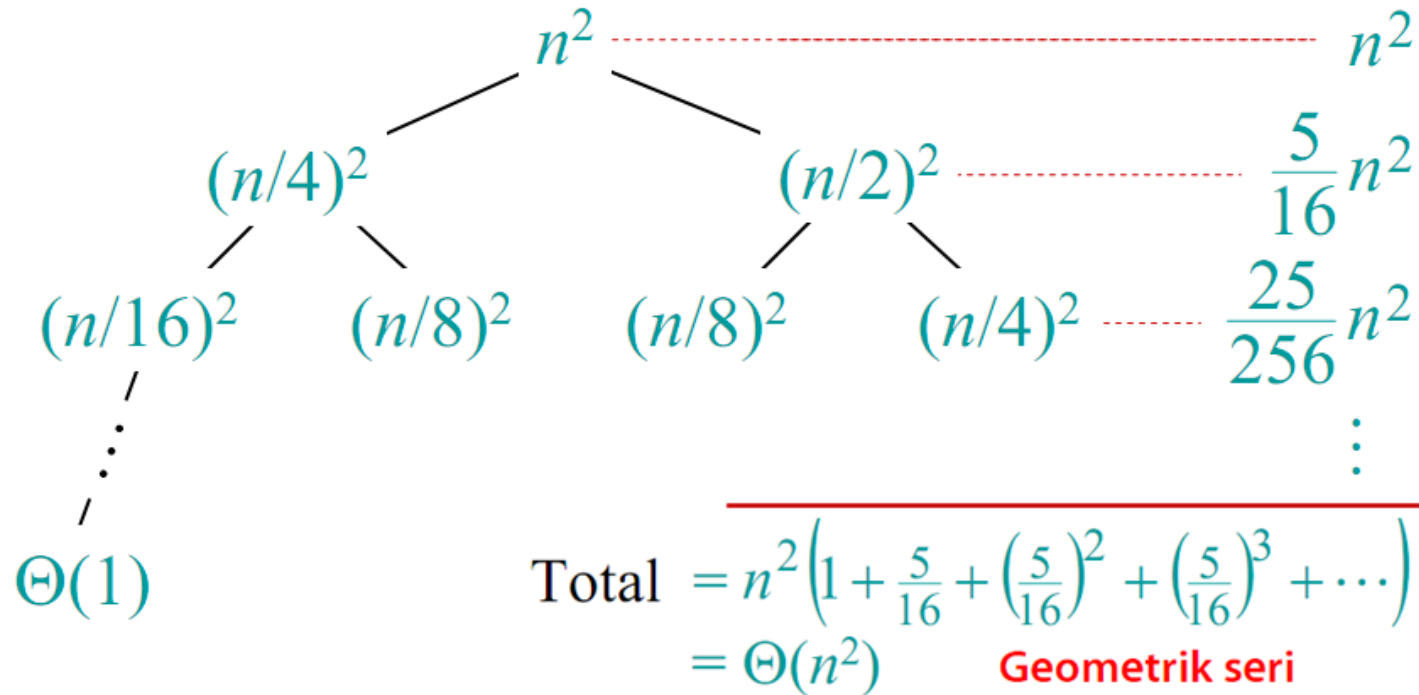


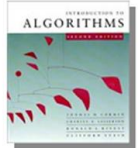
- İterasyon yönteminin çözüm adımları ağaç şeklinde gösterilebilir ve elde edilen ağaca **Özyineleme-ağacı** denir
- Özyineleme-ağacı, bir algorithmadaki özyineleme uygulamasının maliyetini (zamanı) modeller.
- Özyineleme-ağacı metodu, diğer yöntemler gibi, güvenilir olmayabilir.
- Öte yandan özyineleme-ağacı metodu "öngörü" olgusunu geliştirir.
- Özyineleme-ağacı metodu "yerine koyma metodu" için gerekli tahminlerinde yararlıdır.



Özyineleme-ağacı örneği

$$T(n) = T(n/4) + T(n/2) + n^2:$$



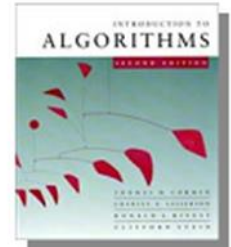


Birleştirme sıralamasının çözümlenmesi

| | | |
|--------------------|-------------|--|
| | $T(n)$ | BİRLEŞTİRME-SIRALAMASI $A[1 \dots n]$ |
| | $\Theta(1)$ | |
| <i>Suistimal</i> ↗ | $2T(n/2)$ | |
| | $\Theta(n)$ | |

1. Eğer $n = 1$ 'se, bitir.
2. Yinelemeli olarak $A[1 \dots \lceil n/2 \rceil]$ ve $A[\lceil n/2 \rceil + 1 \dots n]$ 'yi sırala.
3. 2 sıralı listeyi ***"Birleştir"***

Özensizlik: $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$ olması gerekir, ama asimptotik açıdan bu önemli değildir.



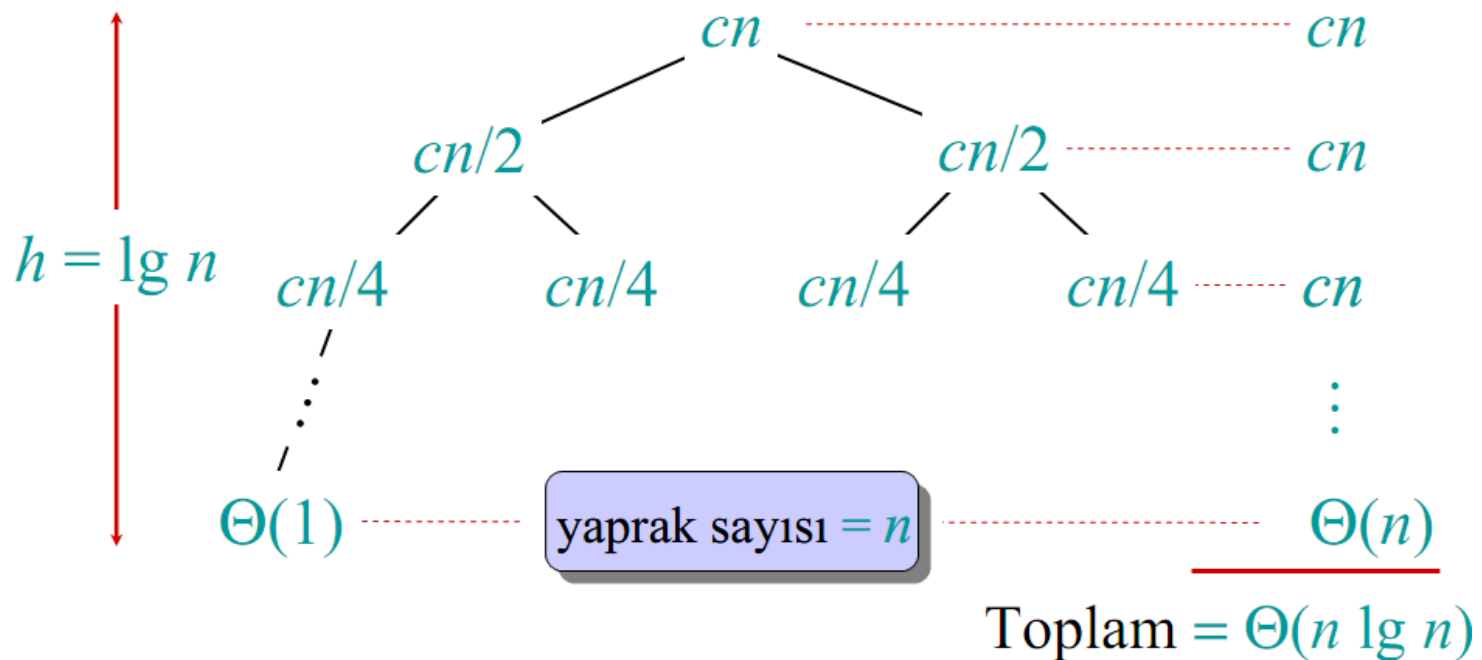
Birleştirme sıralaması için yineleme

$$T(n) = \begin{cases} \Theta(1) & \text{eğer } n = 1 \text{ ise;} \\ 2T(n/2) + \Theta(n) & \text{eğer } n > 1 \text{ ise.} \end{cases}$$

- Genellikle n 'nin küçük değerleri için taban durumu (base case) olan $T(n) = \Theta(1)$ 'i hesaplara katmayacağız; ama bunu sadece yinelemenin asimptotik çözümünü etkilemiyorsa yapacağız.

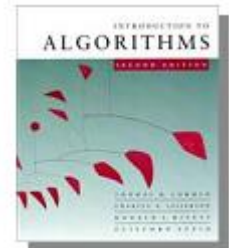
Yineleme ağacı

- $T(n) = 2T(n/2) + cn$ 'yi çözün; burada $c > 0$ bir sabittir.



Sonuçlar- Insert Sort –Merge Sort

- $\Theta(n \lg n)$ 'nin, büyüme oranı $\Theta(n^2)$ 'den daha yavaştır (yani küçüktür).
- En kötü durumda, birleştirme sıralaması asimptotik olarak araya yerleştirme sıralamasından daha iyidir.
- Pratikte, birleştirme sıralaması(Merge Sort) araya yerleştirme sıralamasını (Insert Sort) $n > 30$ değerlerinde geçer.



Yinelemelerin çözümü için değişken değiştirme

- Reküranslar değişken değiştirme ile daha basit hale dönüştürülebilir.
- $T(n) = 2T(\sqrt{n}) + \log n$.

$$m = \log n \Rightarrow 2^m = n \Rightarrow \sqrt{n} = 2^{m/2}$$

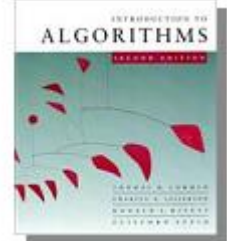
$$T(n) = 2T(\sqrt{n}) + \log n \Rightarrow T(2^m) = 2T(2^{m/2}) + m$$

$$S(m) = T(2^m)$$

$$T(2^m) = 2T(2^{m/2}) + m \Rightarrow S(m) = 2S(m/2) + m$$

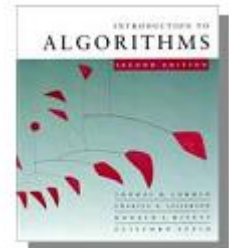
$$\Rightarrow S(m) = O(m \log m)$$

$$\Rightarrow T(n) = T(2^m) = S(m) = O(m \log m) = O(\log n \log \log n)$$



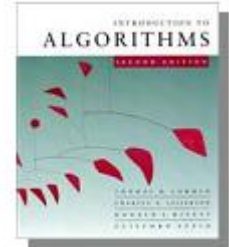
Sorular

- 1- $T(n) = T\left(\frac{n}{2}\right) + \sqrt{n} + \sqrt{6046}$ çözünüz.
- 2- $T(n) = T\left(\frac{n}{5}\right) + T\left(\frac{4n}{5}\right) + \theta(n)$ özyineleme ağacı kullanarak çözünüz.
- 3- Asimptotik notasyonlardan hangilerinin geçişme özelliği (transitivity) vardır.



Sorular

- 4- Aşağıdaki tekrarlı bağıntıların asimptotik davranışlarını inceleyiniz. ($T(0)=1$, $T(1)=1$)
 - a) $T(n)=2T(n-1)-T(n-2)+\Theta(n)$
 - b) $T(n)=3T(n-1)-2T(n-2)+\Theta(n2^n)$
 - c) $T(n)=4T(n/2)-4T(n/4)+\Theta(n \lg n)$
 - d) $T(n)=5T(n/2)-6T(n/4)+\Theta(n)$
- 5- $T(n)=T(\lceil n/2 \rceil)+1$ tekrarlı bağıntısı için $T(n)=O(\lg n)$ olduğunu gösteriniz.
- 6- $T(n)=T(n/2)+T(n/4)+T(n/8)+n$ öz yineleme ağacı ile çözünüz



Sorular

- 7- $T(n) = 2T(\lfloor n/2 \rfloor) + n$ tekrarlı bağıntısı için $T(n) = \Omega(n \lg n)$ ve $T(n) = O(n \lg n)$ olduğunu gösteriniz. Son olarak $T(n) = \Theta(n \lg n)$ asimptotik davranışı gösterip göstermediğini açıklayınız.
- 8- Aşağıdaki tekrarlı bağıntıyı çözünüz. ($0 < p \leq q < 1$)
 - $T(n) = T(pn) + T(qn) + \Theta(n)$
 - $T(1) = \Theta(1)$
- 9- i değişkeni 1 ile $n-1$ arasında değer alan düzenli dağıtık bir gelişigüzel değişken olsun.
- $f(n) = (i+1)f(i)$ ve $f(1) = 1$
tekarlı bağıntısı için $f(n)$ nin mümkün olan değerleri nelerdir ve ortalama değeri nedir?
- 10- $T(n) = 2T(\lfloor n/2 \rfloor + 17) + n$ tekrarlı bağıntısının çözümünün $O(n \lg n)$ olduğunu gösteriniz.

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

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

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. $\sum_{i=l}^u a_i = \sum_{i=l}^m a_i + \sum_{i=m+1}^u a_i$, 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)

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

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



3.Hafta

Master Teorem ve Böl-Fethet Metodu