Özyineli Olmayan (Nonrecursive) Algoritmaların Matematiksel Analizi

- En büyük elemanı bulma problemi

ALGORITHM $MaxElement(A[0..n-1])$
//Determines the value of the largest element in a given array
//Input: An array $A[0..n-1]$ of real numbers
//Output: The value of the largest element in A
$maxval \leftarrow A[0]$
for $i \leftarrow 1$ **to** $n-1$ **do**
    if $A[i] > maxval$
        $maxval \leftarrow A[i]$
return $maxval$

En Büyük Elemanı Bulma Problemi

```
ALGORITHM  MaxElement(A[0..n − 1])
    //Determines the value of the largest element in a given array
    //Input: An array A[0..n − 1] of real numbers
    //Output: The value of the largest element in A
    maxval ← A[0]
    for i ← 1 to n − 1 do
        if A[i] > maxval
            maxval ← A[i]
    return maxval
```

- Girdi büyüklüğü :
  - n elemanlı dizi
- En çok gerçekleştirilen işlem:
  - Döngünün içerisindeki işlemler
    - Karşılaştırma
    - Atama
- En iyi, en kötü, ortalama durum
  - Tüm elemanlar için karşılaştırma yapılacağından söz konusu değil

- Karşılaştırma işlemi kaç kere yapılıyor?
  - Döngünün her turunda 1 kez

$$C(n) = \sum_{i=1}^{n-1} 1.$$

- Bu toplamın sonucu

$$C(n) = \sum_{i=1}^{n-1} 1 = n - 1 \in \Theta(n)$$

Özyineli Olmayan Algoritmaların Zaman Etkinliğinin Analizinin Genel Planı

1. Girdi büyüklüğünü gösteren parametrelerin belirlenmesi
2. Algoritmanın temel işleminin belirlenmesi
3. Girdi büyüklüğüne bağımlı olarak temel işlemin kaz kez gerçekleştiğinin bulunması
   – Başka parametrelere bağlı ise en kötü, en iyi, ortalama durum incelemeleri
4. Temel işlemin kaç kez gerçekleştiğinin bir toplam formülüyle gösterilmesi
5. Toplam formülünün büyüme derecesini gösterecek forma dönüştürülmesi

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. $\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$

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

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 $l \le m < u$

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

Dizi Elemanlarının Eşsizliği

- Bir dizinin tüm elemanlarının birbirinden farklı olması

```
ALGORITHM  UniqueElements(A[0..n − 1])
    //Determines whether all the elements in a given array are distinct
    //Input: An array A[0..n − 1]
    //Output: Returns "true" if all the elements in A are distinct
    //            and "false" otherwise
    for i ← 0 to n − 2 do
        for j ← i + 1 to n − 1 do
            if A[i] = A[j] return false
    return true
```

- Girdi büyüklüğü :
  - n
- En çok gerçekleştirilen işlem:
  - Döngünün içerisindeki işlemler
    - Karşılaştırma
- Karşılaştırma işleminin gerçekleşme sayısı
  - Sadece n'e bağlı değil
  - Eşit eleman olmasına da bağlı
  - İnceleme en kötü duruma göre yapılmalı
- En kötü durum
  - Dizide eşit eleman olmaması
  - Dizinin son iki elemanının eşit olması
- Karşılaştırma işleminin gerçekleşme sayısının hesabı
  - limitleri i+1'den n-1'e kadar olan içteki j döngüsünün her tekrarında 1 karşılaştırma yapılıyor
  - İç döngü dıştaki i döngüsünün her değeri için tekrarlanıyor

Dizi Elemanlarının Eşsizliği

$$C_{worst}(n) = \sum_{i=0}^{n-2} \sum_{j=i+1}^{n-1} 1 \qquad = \sum_{i=0}^{n-2} [(n-1) - (i+1) + 1] = \sum_{i=0}^{n-2} (n-1-i)$$

$$= \sum_{i=0}^{n-2} (n-1) - \sum_{i=0}^{n-2} i \qquad = (n-1) \sum_{i=0}^{n-2} 1 - \frac{(n-2)(n-1)}{2}$$

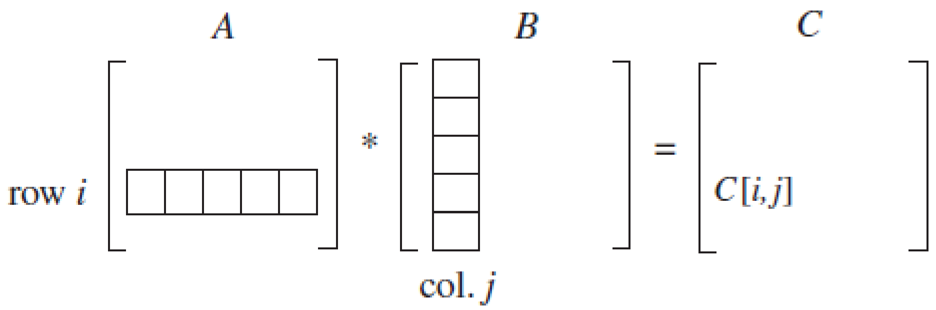
$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} \qquad = \frac{(n-1)n}{2} \approx \frac{1}{2} n^2 \in \Theta(n^2).$$

ALGORITHM *UniqueElements*$(A[0..n-1])$

//Determines whether all the elements in a given array are distinct
//Input: An array $A[0..n-1]$
//Output: Returns "true" if all the elements in A are distinct
//   and "false" otherwise
for $i \leftarrow 0$ **to** $n-2$ **do**
    for $j \leftarrow i+1$ **to** $n-1$ **do**
        if $A[i] = A[j]$ **return false**
return true

n*n Matris Çarpımı

- n*n boyutlarındaki iki matrisin çarpımı



$$C[i,\ j] = A[i, 0]B[0,\ j] + \cdots + A[i, k]B[k,\ j] + \cdots + A[i, n-1]B[n-1,\ j]$$

ALGORITHM *MatrixMultiplication*($A[0..n-1, 0..n-1]$, $B[0..n-1, 0..n-1]$)

    //Multiplies two square matrices of order n by the definition-based algorithm
    //Input: Two $n \times n$ matrices A and B
    //Output: Matrix $C = AB$
    for $i \leftarrow 0$ **to** $n-1$ **do**
        for $j \leftarrow 0$ **to** $n-1$ **do**
            $C[i,\ j] \leftarrow 0.0$
            for $k \leftarrow 0$ **to** $n-1$ **do**
                $C[i,\ j] \leftarrow C[i,\ j] + A[i,\ k] * B[k,\ j]$
    return C

n*n Matris Çarpımı

```
ALGORITHM MatrixMultiplication(A[0..n − 1, 0..n − 1], B[0..n − 1, 0..n − 1])
    //Multiplies two square matrices of order n by the definition-based algorithm
    //Input: Two n × n matrices A and B
    //Output: Matrix C = AB
    for i ← 0 to n − 1 do
        for j ← 0 to n − 1 do
            C[i, j] ← 0.0
            for k ← 0 to n − 1 do
                C[i, j] ← C[i, j] + A[i, k] ∗ B[k, j]
    return C
```

- Girdi büyüklüğü :
  - n. Dereceden matris
- En çok gerçekleştirilen işlem:
  - En iç döngünün içerisindeki işlemler
    - Toplama ve Çarpma
  - Döngünün her turunda 1 kez gerçekleşiyorlar
  - Birini seçmek yeterli (Çarpma)
- En iyi, en kötü, ortalama durum incelemesi
  - Matrisin tüm elemanları için işlem yapılacağından gerek yok

- Temel işlem çarpma (M(n)) En içteki k döngüsünün her tekrarında 1 kez gerçekleşiyor
  - Alt sınır 0, üst sınır n-1

$$\sum_{k=0}^{n-1} 1,$$

- Toplam çarpma sayısı

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1.$$

$$M(n) = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \sum_{k=0}^{n-1} 1 = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} n = \sum_{i=0}^{n-1} n^2 = n^3.$$

Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

- Bir pozitif onluk sayının ikili sayı sisteminde kaç basamaklı olduğunun bulunması

```
ALGORITHM Binary(n)
    //Input: A positive decimal integer n
    //Output: The number of binary digits in n's binary representation
    count ← 1
    while n > 1 do
        count ← count + 1
        n ← ⌊n/2⌋
    return count
```

- Temel işlem : Karşılaştırma
  - Döngünü içinde değil
  - Döngü içeriğinin icra edilip edilmeyeceğini belirliyor
  - Karşılaştırma döngü içindeki işlemlerden 1 kez fazla yapılıyor
- Gerçekleşme Sayısı
  - Döngü değişkeni alt ve üst sınırları arasında çok az değer alıyor
  - Her tekrarda yarılanmasından dolayı
  - Bu durumda n girdi boyutu için $\log_2 n$ olmalıdır
  - Temel işlem (n>1) $\log_2 n + 1$ kez gerçekleşir
- Bu tür durumların incelenmesi özyineli algoritmalar ile daha sağlıklı olur

Özyineli Algoritmaların Matematiksel Analizi

- F(n) = n! Değerinin hesaplanması

$$n! = 1 \cdot \ldots \cdot (n-1) \cdot n = (n-1)! \cdot n \quad \text{for } n \geq 1$$

ALGORITHM $F(n)$
//Computes $n!$ recursively
//Input: A nonnegative integer n
//Output: The value of $n!$
if $n = 0$ **return** 1
else return $F(n-1) * n$

- F(n) = F(n-1).n
- Temel işlem: Çarpma M(n)

$$n! = 1 \cdot \ldots \cdot (n-1) \cdot n = (n-1)! \cdot n \quad \text{for } n \geq 1$$

$$M(n) = \underbrace{M(n-1)}_{\substack{\text{to compute} \\ F(n-1)}} + \underbrace{1}_{\substack{\text{to multiply} \\ F(n-1) \text{ by } n}} \quad \text{for } n > 0.$$

Faktöriyel Alma

$$M(n) = \underbrace{M(n-1)}_{\substack{\text{to compute}\\ F(n-1)}} + \underbrace{1}_{\substack{\text{to multiply}\\ F(n-1) \text{ by } n}} \quad \text{for } n > 0.$$

- M(n) n’ e bağlı bir fonksiyon
  - Dolaylı olarak aynı zamanda n-1’e bağlı bir fonksiyondur
- Bu duruma özyineleme denir
- Yapılması gereken $M(n) = M(n-1) + 1$ serisinin çözülmesidir

Faktöriyel Alma

- **if** *n* = 0 **return** 1. satırı n =0 olduğunda özyineleme çağırımının duracağını ve çarpma işlemi yapılmayacağını belirtir

$$M(0) = 0.$$

the calls stop when $n = 0$

no multiplications when $n = 0$

- Bu durum özyineleme ilişkisini ve çarpma sayısı algoritması için başlangıç koşulunu verir

$$M(n) = M(n-1) + 1 \quad \text{for } n > 0,$$
$$M(0) = 0.$$

$$F(n) = F(n-1) \cdot n \quad \text{for every } n > 0,$$
$$F(0) = 1.$$

Faktöriyel Alma

- Özyineleme ilişkisinin çözümü için kullanılan yöntemlerden biri
  - Backward Substitution (Geriye doğru değiştirme)

$$
\begin{aligned}
M(n) &= M(n-1)+1 && \text{substitute } M(n-1) = M(n-2)+1 \\
&= [M(n-2)+1]+1 = M(n-2)+2 && \text{substitute } M(n-2) = M(n-3)+1 \\
&= [M(n-3)+1]+2 = M(n-3)+3.
\end{aligned}
$$

  - Seri incelendiğinde şu örüntü görülebilir.

    $M(n) = M(n-i)+i.$

  - n = 0’dan i = n’e kadar gidildiğinde

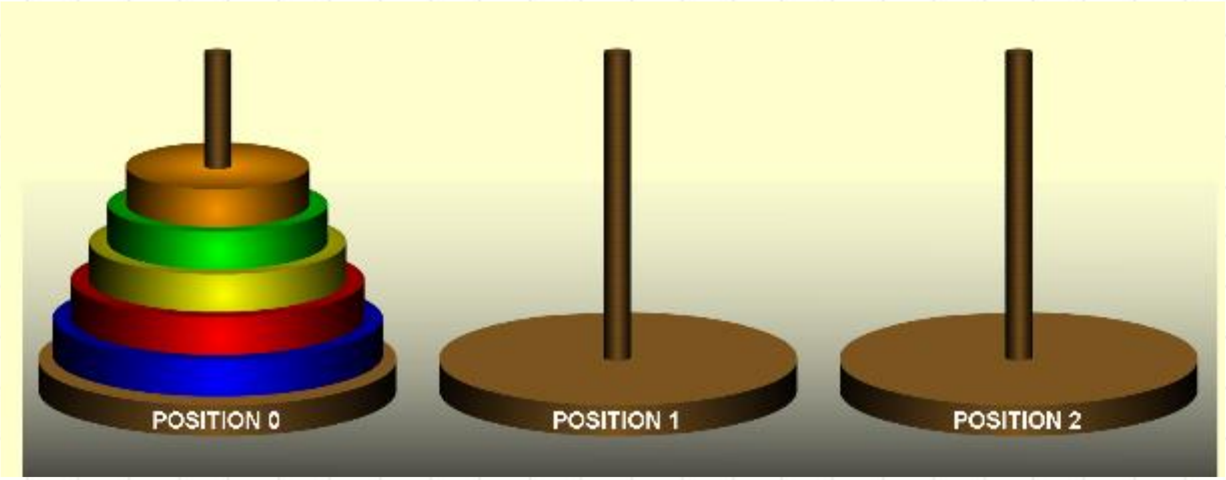
$$M(n) = M(n-1)+1 = \cdots = M(n-i)+i = \cdots = M(n-n)+n = n.$$

Özyineli Algoritmaların Zaman Etkinliğinin Analizinin Genel Planı

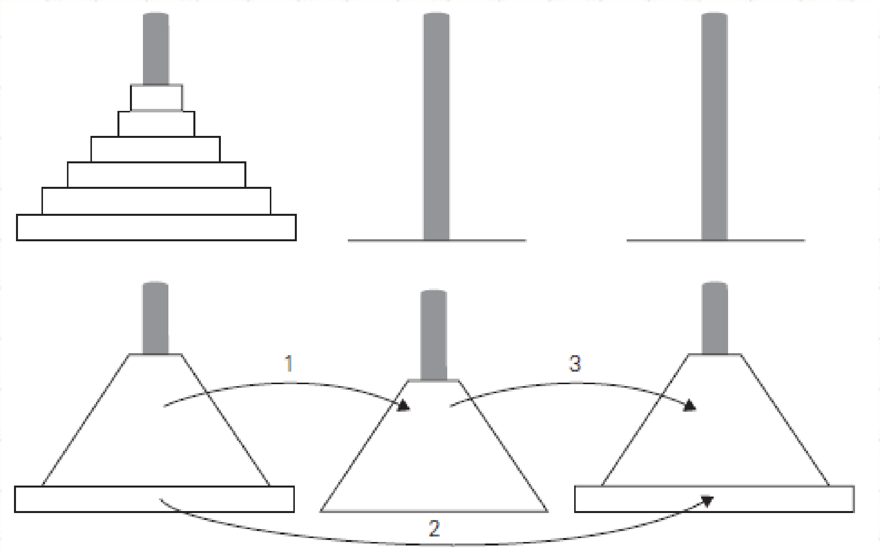
1. Girdi büyüklüğünü gösteren parametrelerin belirlenmesi
2. Algoritmanın temel işleminin belirlenmesi
3. Girdi büyüklüğüne bağımlı olarak temel işlemin kaz kez gerçekleştiğinin bulunması
   – Aynı büyüklükteki girdiler için farklı sayıda gerçekleşiyorsa en kötü, en iyi, ortalama durum incelemeleri
4. Temel işlemin kaç kez gerçekleştiğinin bir özyineleme ilişkisiyle gösterilmesi
5. Özyinelemenin çözülmesi ve büyüme derecesinin araştırılması

Hanoi Kuleleri Bulmacası

- Hanoi Kuleleri Bulmacası
  - Farklı büyüklüklerde diskler
  - 1. Çubuktan 3. çubuğa aynı sırayla taşınacak
  - Her defasında bir disk hareket ettirilecek
  - Büyük disk küçük diskin üzerine gelmeyecek



Hanoi Kuleleri Bulmacası



- n>1 adet disk Ç1’den Ç3’e özyineli olarak taşınması
  - Önce n-1 disk Ç1’den Ç2’ye özyineli olarak taşınır
  - n. Disk Ç1’den Ç3’e taşınır
  - Son olarak n-1 disk Ç2’den Ç3’e özyineli olarak taşınır
  - n=1 ise 1 hareketle işlem gerçekleşir.

- Önce n-1 disk Ç1’den Ç2’ye özyineli olarak taşınır
- n. Disk Ç1’den Ç3’e taşınır
- Son olarak n-1 disk Ç2’den Ç3’e özyineli olarak taşınır

- Girdi büyüklüğü: Disk sayısı
- Hamle Sayısı (M(n)) n’e bağımlı

$$M(n) = M(n-1) + 1 + M(n-1) \quad \text{for } n > 1.$$

- Başlangıç koşulu M(1)=1 olduğuna göre özyineleme ilişkisi

$$M(n) = 2M(n-1) + 1 \quad \text{for } n > 1,$$
$$M(1) = 1.$$

- Backward Substitution (Geriye doğru değiştirme)

$$M(n) = 2M(n-1) + 1 \qquad \text{sub. } M(n-1) = 2M(n-2) + 1$$
$$= 2[2M(n-2) + 1] + 1 = 2^2 M(n-2) + 2 + 1 \quad \text{sub. } M(n-2) = 2M(n-3) + 1$$
$$= 2^2[2M(n-3) + 1] + 2 + 1 = 2^3 M(n-3) + 2^2 + 2 + 1.$$
$$2^4 M(n-4) + 2^3 + 2^2 + 2 + 1,$$

- Örüntü

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^i M(n-i) + 2^i - 1.$$

Hanoi Kuleleri Bulmacası

- Örüntü

$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^i M(n-i) + 2^i - 1.$$

- Başlangıç koşulu n=1, i=n-1 tekrar sayısı formülde yerine konursa:

$$\begin{aligned} M(n) &= 2^{n-1} M(n-(n-1)) + 2^{n-1} - 1 \\ &= 2^{n-1} M(1) + 2^{n-1} - 1 = 2^{n-1} + 2^{n-1} - 1 = 2^n - 1. \end{aligned}$$

- Algoritmanın büyüme derecesi üssel bir fonksiyondur.
  - Büyük n değerleri için çözüm çok uzun sürecektir.
  - Bu durum algoritmanın etkin olmadığını göstermez

Fibonacci Sayıları

Fibonacci Sayıları:

0, 1, 1, 2, 3, 5, 8, 13, 21, …

Fibonacci Özyinelemesi:

$F(n) = F(n\text{-}1) + F(n\text{-}2)$

$F(0) = 0$

$F(1) = 1$