Algorithm Analysis And Design

With Infinity Teams

Eng: Abdulrahman Hamdi

INFINITY TEAM
Since 2020

Table of contents

Eng: Abdulrahman Hamdi

05

Mathematical Recursion and Iteration Analysis

Eng: Abdulrahman Hamdi

Özyineli Olmayan (Nonrecursive)

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 $maxval$

- 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

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

Eng: Abdulrahman Hamdi

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 kaç 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

Eng: Abdulrahman Hamdi

Önemli formüller

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

Eng: Abdulrahman Hamdi

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

Eng: Abdulrahman Hamdi

Dizi Elemanlarının Eşsizliği

- Pseudo code ve önemli formüller

```
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
```

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

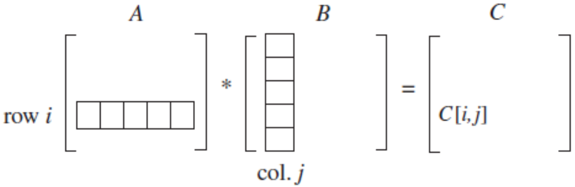
$$= (n-1)^2 - \frac{(n-2)(n-1)}{2} = \frac{(n-1)n}{2} \approx \frac{1}{2}n^2 \in \Theta(n^2).$$

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

Eng: Abdulrahman Hamdi

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

Eng: Abdulrahman Hamdi

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

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

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

Eng: Abdulrahman Hamdi

Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

- Soru: Aldığı bir pozitif tamsayının ikilik sistemde kaç basamaklı olacağına dönen aşağıdaki algoritmanın zaman verimliliğini bulunuz.
- 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
```

```
ikiliSistem(n)
//Girdi: n tam sayısı (10'luk sistemde)
//Çıktı: n'nin ikilik sistemdeki toplam basamak sayısı
    sayac ← 1
while n>1
sayac ← sayac+1
n ← ⌊ n/2 ⌋ //n/2'nin tam sayı kısmını al
return sayac
```

Eng: Abdulrahman Hamdi

Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

• 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

Eng: Abdulrahman Hamdi

Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

- **Bu algoritmanın nasıl çalıştığını şu şekilde görebiliriz:**

35/2= 17 → 35 = n (ilk n)
17/2=8 → 17 ikinci n
8/2=4→ 8 üçüncü n
4/2=2→ 4 dördüncü n
2/2=1→ 2 beşinci n
1 → altıncı n ve son n çünkü n=1 olduğunda while'dan çıkıyoruz
Şimdi buradaki while'ın kaç defa döneceğine bakalım

1.iterasyonda n'nın yarısı gider $\frac{n}{2}$ kalır.

2.iterasyonda $\frac{n}{2}$'nin yarısı gider $\frac{n}{2} \cdot \frac{1}{2} = \frac{n}{2^2}$ kalır.

3.iterasyonda $\frac{n}{2^2}$'nin yarısı gider $\frac{n}{2^2} \cdot \frac{1}{2} = \frac{n}{2^3}$ kalır

. . .

i.İterasyonda . . . . . $\frac{n}{2^i}$ kalır.

Eng: Abdulrahman Hamdi

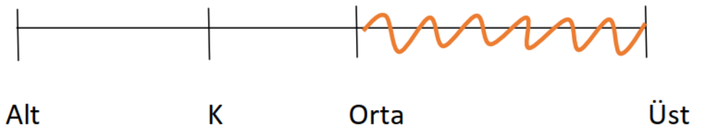
Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

Genel kural şudur i.iterasyonda $\frac{n}{2^i}$ kalır.
While'ın durması için kalanın yani elimizdeki sayının 1 olması gerekir. Kaçıncı iterasyondan sonra 1 kalır, yani while durur, anlayabilmemiz için $\frac{n}{2^i}$ 'yi 1'e eşitlememiz gerekir.

$$\frac{n}{2^i} = 1$$

ise i= $\log_2 n$ olur. Yani bir n tamsayısı için toplamda $\log_2 n$ adımda algoritma n'nın ikilik sistemde toplam kaç basamaklı olacağını bulur.
O halde bu algoritma için $T(n) = \log_2 n$ 'dır. Bu da $O(\log_2 n)$ 'e aittir.
T(n)= $\log_2 n \in O(\log_2 n)$

Soru: n uzunluğunda sıralı bir dizi alıp bu dizide bir K değerini arayan binarySearch algoritmasının zaman verimliliğine bakalım.
Önce bu algoritmanın çalışma mantığına bakalım:
Eğer dizide aradığımız K değeri orta değerin altında kalıyorsa yani

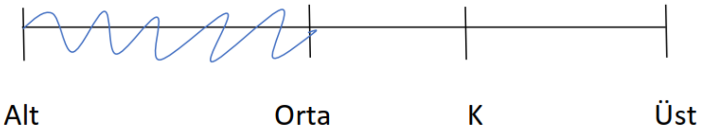


Turuncu rengi bu durumda üst indis ortanın indisinin bir eksiği olur. Bu diziden orta–üst kısmın yani buranın atılması demektir. Dizinin mevcut büyüklüğü yarıya iner.

Eng: Abdulrahman Hamdi

Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

Eğer dizide aradığımız K değeri dizinin ortanca indisteki değerinden daha büyük kalıyorsa yani



Mavi rengi Bu kısım atılır. Yeni alt indis (eski) orta indisinin 1 fazlası olur. Bu şekilde yani her defasında mevcut dizinin yarısı atılarak devam edilir. Ortanca indisteki değer K'ya denk getirildiğinde algoritma sonlanır.

binarySearch(A[0, ……. ,n-1],K)
//Girdi: n uzunluğunda küçükten büyüğe sıralı bir A dizisi ve bir K değeri
//Çıktı: K'nın olduğu A'nın indisi
alt ← 0
üst ← n-1
while alt ≤ üst
orta ⌊ alt+üst/2 ⌋
if A[orta]=K **return** True
else if A[orta]> K // K alt ile orta arasında
üst ← orta-1
else
üst ← orta-1
return False

Eng: Abdulrahman Hamdi

Onluk Sayının İkili Sayı Sisteminde Basamak Sayısının Bulunması

1.iterasyonda $\frac{n}{2}$ eleman atılır

2.iterasyonda $\frac{n}{2}$'nin elemanın yarısı atılır $\frac{n}{2^2}$ kalır.

3.iterasyonda $\frac{n}{2^2}$'nin elemanın yarısı atılır $\frac{n}{2^3}$ kalır

. . .

i.İterasyonda . . . . . $\frac{n}{2^i}$ kalır.

En kötü durumda kalan eleman sayısı 1 olur.

Yani $\frac{n}{2^i}$ =1 → i= $\log_2 n$ olur.

En kötü durumda while loopu $\log_2 n$ kez çalışır o halde T_{worst} = $\log_2 n \in O(\log_2 n)$

Eng: Abdulrahman Hamdi

Ödev 04 Çözümü

ALGORITHM *Mystery*(n)

//Input: A nonnegative integer n
$S \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $S \leftarrow S + i * i$
return S

a. Bu algoritma neyi hesaplıyor?

Algoritma, ilk n pozitif tam sayının karelerinin toplamını hesaplar. Matematiksel olarak:

$$S = 1^2 + 2^2 + 3^2 + \cdots + n^2$$

Bu, kareler toplamı formülüyle de ifade edilebilir:

$$S = \frac{n(n+1)(2n+1)}{6}$$

Eng: Abdulrahman Hamdi

b. Temel işlemi nedir?

Temel işlem, S ← S + i * i yani bir sayının karesini alıp toplama ekleme işlemidir.

c. Temel işlem kaç kez yürütülür?

Döngü n kez çalışır, dolayısıyla temel işlem tam olarak n kez yürütülür.

d. Bu algoritmanın verimlilik sınıfı nedir?

En dıştaki döngü n kez çalışıyor, dolayısıyla T(n) = O(n) olur.Yani algoritma doğrusal zaman karmaşıklığına (O(n)) sahiptir.

Ödev 04 Çözümü

ALGORITHM *Mystery(n)*

//Input: A nonnegative integer n
$S \leftarrow 0$
for $i \leftarrow 1$ **to** n **do**
    $S \leftarrow S + i * i$
return S

e. Bir iyileştirme veya daha iyi bir algoritma önerin ve verimlilik sınıfını belirtin. Bunu yapamazsanız, bunun yapılamayacağını kanıtlamaya çalışın.

Kareler toplamı için kapalı bir formül var:

$$S = \frac{n(n+1)(2n+1)}{6}$$

Bu formülle döngüye gerek kalmaz, çünkü işlem sadece sabit sayıda çarpma ve bölme işlemi gerektirir.Bu yöntem, O(1) (sabit zamanlı) bir algoritmadır, yani çok daha verimlidir.

Orijinal algoritma O(n) iken, formülü kullanarak O(1) zamanda çözebiliriz. Bu yüzden, daha iyi bir algoritma mümkündür ve önerilmiştir.

Eng: Abdulrahman Hamdi

Ödev 05

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

```
//Input: An array A[0..n − 1] of n real numbers
minval ← A[0]; maxval ← A[0]
for i ← 1 to n − 1 do
    if A[i] < minval
        minval ← A[i]
    if A[i] > maxval
        maxval ← A[i]
return maxval − minval
```

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

```
//Input: A matrix A[0..n − 1, 0..n − 1] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        if A[i, j] ≠ A[j, i]
            return false
return true
```

a. Bu algoritma neyi hesaplıyor?

b. Temel işlemi nedir?

c. Temel işlem kaç kez yürütülür?

d. Bu algoritmanın verimlilik sınıfı nedir?

e. Bir iyileştirme veya daha iyi bir algoritma önerin ve verimlilik sınıfını belirtin. Bunu yapamazsanız, bunun yapılamayacağını kanıtlamaya çalışın.

Eng: Abdulrahman Hamdi

Ödev 05

ALGORITHM $GE(A[0..n-1, 0..n])$

```
//Input: An n × (n + 1) matrix A[0..n − 1, 0..n] of real numbers
for i ← 0 to n − 2 do
    for j ← i + 1 to n − 1 do
        for k ← i to n do
            A[j, k] ← A[j, k] − A[i, k] ∗ A[j, i] / A[i, i]
```

a. Bu algoritmanın zaman verimliliği sınıfını bulun.

b. Bu sözde kodda belirgin bir verimsizlik nedir ve algoritmayı hızlandırmak için nasıl ortadan kaldırılabilir?

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

- F(n) = n! Değerinin hesaplanması

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

and $0! = 1$ by definition, we can compute $F(n) = F(n-1) \cdot n$ with the following recursive algorithm.

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

$M(0) = 0.$

the calls stop when $n = 0$ no multiplications when $n = 0$

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

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

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

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

- 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

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

Eng: Abdulrahman Hamdi

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 kaç 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. Özyinelemenin çözülmesi ve büyüme derecesinin araştırılması

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

1'den n'e kadar olan sayıları rekürsif olarak toplayan algoritmaya bakalım

```
topla(n)
if n=1
  return 1
else
  return n+topla(n-1)
```

1'den n'e kadar olan sayıların toplanması için yani topla(n)'nin çalışması için gereken zamana T(n) dersek, bu 1'den n–1'e kadar olan sayıların toplanması için gereken zamana bir birim fazladan zaman eklenmesi ile bulunur; bu bir birimlik artış var olan toplama n'nin eklenmesi için gereken zamandır.

O halde; T(n)=1+T(n–1) diyebiliriz.

T(n)=1+T(n-1)	→	1.iterasyon
1+T(n-2)	→	2.iterasyon
1+T(n-3)	→	3.iterasyon
1+T(n-4)	→	4.iterasyon

Genellersek i.iterasyon için T(n)=i+T(n–i) olur. i=n–1 değerini aldığında yani bu algoritma n–1 defa çağırıldığında

T(n)=n-1+T(1)

↓

Bunun çalışma zamanı 1'dir

O halde T(n)=n ϵ O(n) olur.

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

```
ALGORITHM algo(n)
if n = 1 then
    return 4
else
    return 3 * algo(n - 1)
```

Bu algoritmanın zaman verimliliğini hesaplayalım

Her ne kadar algo(n)=3.algo(n-1) (n>1) şeklinde olsa da bir önceki örnekle benzer olarak algo (n)'i hesaplamak için geçen süre algo(n-1)'i hesaplamak için gereken sürenin 1 fazlasıdır.(Burada 3 ile çarpmak işlem sayısını 1 arttırır.) O halde yine T(n)=1+T(n-1)

n büyüklüğündeki bir girdi ile, n-1 büyüklüğündeki bir girdinin çalışma süreleri arasında aynı rekürsif ilişki vardır. Sonuç olarak bu algoritma içinde T(n)=n ϵ O(n) bulunur.

Kural: Her rekürsif iterasyonda sabit/aynı sayıda işlem yapılıyorsa örneğin bir önceki örnekte algo'yu her çağırdımızda algo'yu bir düşük girdi için tekrar çağırıyor ve bunu 3 ile çarpıyoruz, yani her iterasyonda 2 işlem yapıyoruz, böyle durumlarda algoritmanın zaman verimliliği direkt olarak algoritmanın kaç kez çağırıldığına eşittir.

Eng: Abdulrahman Hamdi

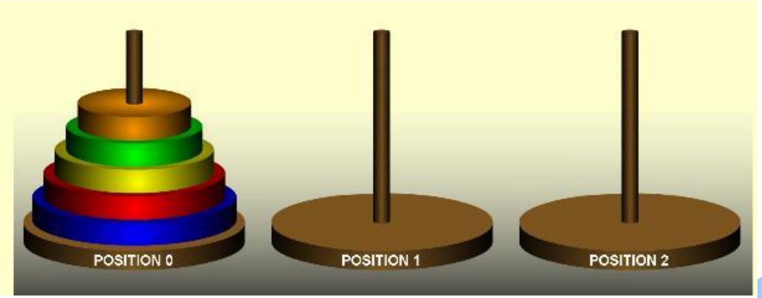
Özyineli Algoritmaların Matematiksel Analizi

```
ALGORITHM algo(n)
if n ≤ 1 then
    return 1
else
    return 2 * algo(n - 1)
```

Algoritmanın zaman verimliliği nedir?
algo(n) çağırılır → algo(n-1) çağırılır → algo(n-2) çağrılır ……….. → algo(1) çağrılır
Toplam n adet çağırma olur $T(n)=n \in O(n)$ olur.
Şimdi bu algoritmanın rekürsif kısmını 2*algo(n-1) yerine algo (n-1)+algo(n-1) şeklinde yazalım.
Bakalım zaman verimliliği değişecek mi?

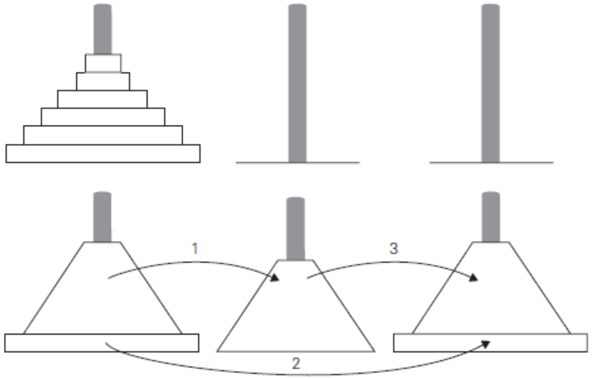
Hanoi Kuleleri Bulmacası

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



Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi



- 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.

- 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)

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

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

- Örüntü

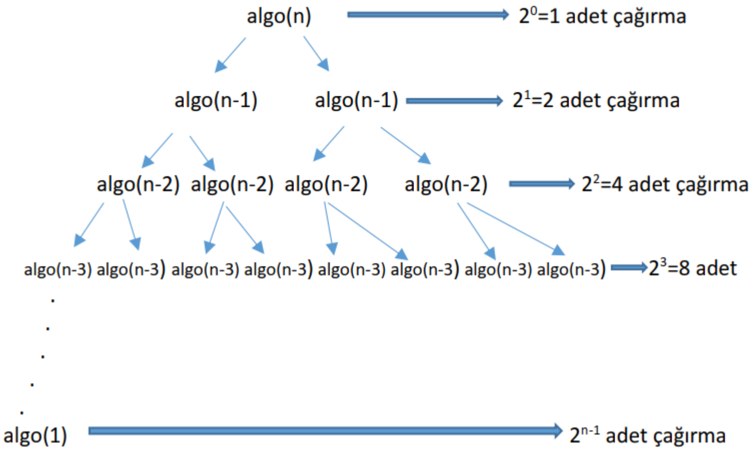
$$M(n) = 2^i M(n-i) + 2^{i-1} + 2^{i-2} + \cdots + 2 + 1 = 2^i M(n-i) + 2^i - 1.$$

- 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

```
ALGORITHM algo(n)
if n ≤ 1 then
    return 1
else
    return algo(n - 1) + algo(n - 1)
```



Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

Toplam çağırmalar

$2^0+2^1+.....+2^{n-1}=\frac{2^n-1}{2-1}=2^n-1$

Geometrik seri toplamı $= a+ar+ar^2+.......+ar^{n-1}$

$= a(1+r+r^2+.......+r^{n-1})$

$= \frac{ar^n-a}{r-1}$

Toplam 2^n-1 adet fonksiyon yani algoritma çağırılır o halde $T(n)=2^n-1 \in O(2^n)$'dir.

Dikkat edersek algoritmada yaptığımız ufak bir değişiklik ile O değeri n'den 2^n'e çıktı !

En bilinen rekürsif algoritmalardan biri olan Fibonacci'ninde zaman verimliliği $O(2^n)$'dir.

Özyineli Algoritmaların Matematiksel Analizi

Fibonacci Sayıları:

0, 1, 1, 2, 3, 5, 8, 13, 21, ...

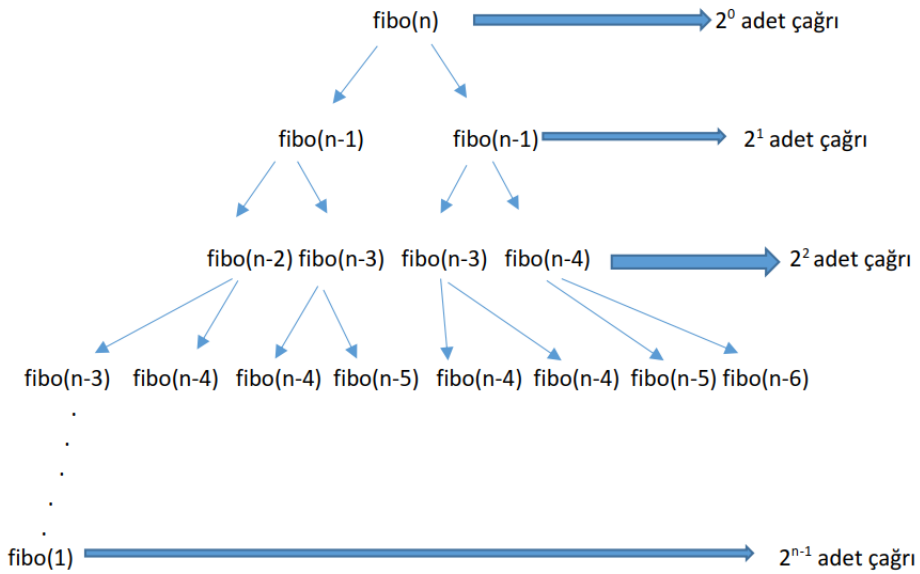
Fibonacci Özyinelemesi:

$F(n) = F(n-1) + F(n-2)$

$F(0) = 0$

$F(1) = 1$

```
ALGORITHM fibo(n)
if n ≤ 1 then
    return 1
else
    return fibo(n - 1) + fibo(n - 2)
```



fibo'nun toplam çağrılma sayısı

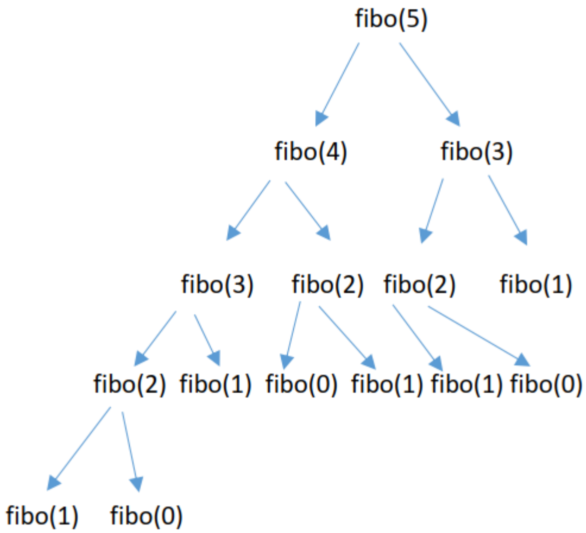
$2^0+2^1+ \ldots\ldots\ldots\ldots\ldots\ldots +2^{n-1}=2^n-1$

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

Fibonacci Sayıları:

Fakat şu örneğe bakalım:



fibo(5) için yapılan toplam çağrı 15 adettir. Formüle göre bunun 25-1=31 olması gerekirdi. Bu bakımdan formülümüz yanlışmış gibi görülebilir. Fakat n çok büyüdüğünde toplam fonksiyon çağrılma sayısı 2n−1'ye yaklaşır, aradaki fark azalır. Zaten bizim yaptığımız asimptotik analizdir, yani çok büyük girdiler için algoritmanın çalışma zamanını tahmin etmektir. O yüzden bulduğumuz T(n) değeri yaklaşık değerdir, kesin sonuç vermeyebilir.

Eng: Abdulrahman Hamdi

Özyineli Algoritmaların Matematiksel Analizi

Rekürsif Olmayan Fibonacci Sayıları:

```
ALGORITHM fibo(n)
if n ≤ 1 then
    return 1
else
    a ← 1
    b ← 1
    for i ← 2 to n do
        c ← a + b
        a ← b
        b ← c
    return c
```

Her iterasonda a, bir önceki iterasyonun b'si, b'de bir önceki iterasyonun c'sidir.

1	1	2	3	5	8
a	b	c			
	a	b	c		
		a	b	c	
			a	b	c

Büyük n'ler için fibo'nun içindeki for loopu n−2 kez döner. O halde T(n)=n−2 ϵ O(n)'dir. Aynı işi yapan, yani n. Fibonacci sayısını hesaplayan algoritmaların ilki (rekürsif olan) $O(2^n)$ iken diğeri O(n)'dir

Özyineli Algoritmaların Matematiksel Analizi

Rekürsif Olmayan Fibonacci Sayıları:
Son olarak algoritma her çağırıldığında farklı sayıda işlem yapılan bir algoritmayı inceleyelim.

```
algo(n)
if n ≤ 1
  return 1
else
  for i=1'den n'ye kadar
    // bazı işlemler (bunun ne olduğu önemli değil)
    .
    .
    .
algo(n-1)
```

algo(n) için n adet işlem yapılır (for n kez döner) + algo(n-1) çağrılır, toplamda n+1 adet işlem yapılır.
algo(n-1) için n-1 adet işlem yapılır, algo(n-2) çağrılır, toplam n adet işlem yapılır.
. . .
algo(2) için 2 adet işlem + algo(1)'in çağırılması toplamda 3 işlem yapılır.

algo(1) için 1 adet işlem yapılır.

Toplam işlem sayısı

Burda olmayan 2'nin çıkartılması

$$1+3+4+\ldots\ldots\ldots+n+n+1 = \frac{(n+1).(n+2)}{2} - 2$$

$$= \frac{n^2+3n+3}{2} - 2 = \frac{n^2+3n-1}{2} \in O(n^2)$$

Eng: Abdulrahman Hamdi

Algorithm Analysis And Design

Thanks for listening!

Eng: Abdulrahman Hamdi