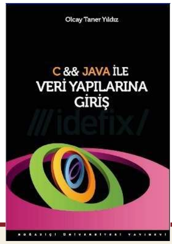


Bölüm 1. Algoritma Analizi

Olcay Taner Yıldız

2014

Büyük-O Gösterimi

Bir dizinin en büyük elemanını bulan algoritma

```
int diziEnBuyuk(int[] dizi){
    int i, enBuyuk;
    enBuyuk = dizi[0];
    for (i = 1; i < dizi.length; i++){
        if (dizi[i] > enBuyuk)
            enBuyuk = dizi[i];
    }
    return enBuyuk;
}
```

İşlem Sayısı (En Fazla)

- 1 defa `enBuyuk = dizi[0]` atama komutu
- 1 defa `i = 1` atama komutu
- N defa `i < N` karşılaştırma komutu
- N - 1 defa `i++` atama komutu
- N - 1 defa `if (dizi[i] > enBuyuk)` karşılaştırma komutu
- N - 1 defa `enBuyuk = dizi[i]` atama komutu

$\mathcal{O}(n)$

Tanım 1 *$f(n)$ ve $g(n)$ pozitif tamsayılardan reel sayılara tanımlı iki fonksiyon olsun. Eğer her $n > n_0$ tamsayısı için $f(n) < cg(n)$ olacak şekilde c ve n_0 sabitleri varsa, $f = \mathcal{O}(g)$'dir ve f fonksiyonu için g fonksiyonu bir üst sınır belirler.*

İşlem Sayısı (En Az)

- 1 defa `enBuyuk = dizi[0]` atama komutu
- 1 defa `i = 1` atama komutu
- N defa `i < N` karşılaştırma komutu
- N - 1 defa `i++` atama komutu
- N - 1 defa `if (dizi[i] > enBuyuk)` karşılaştırma komutu

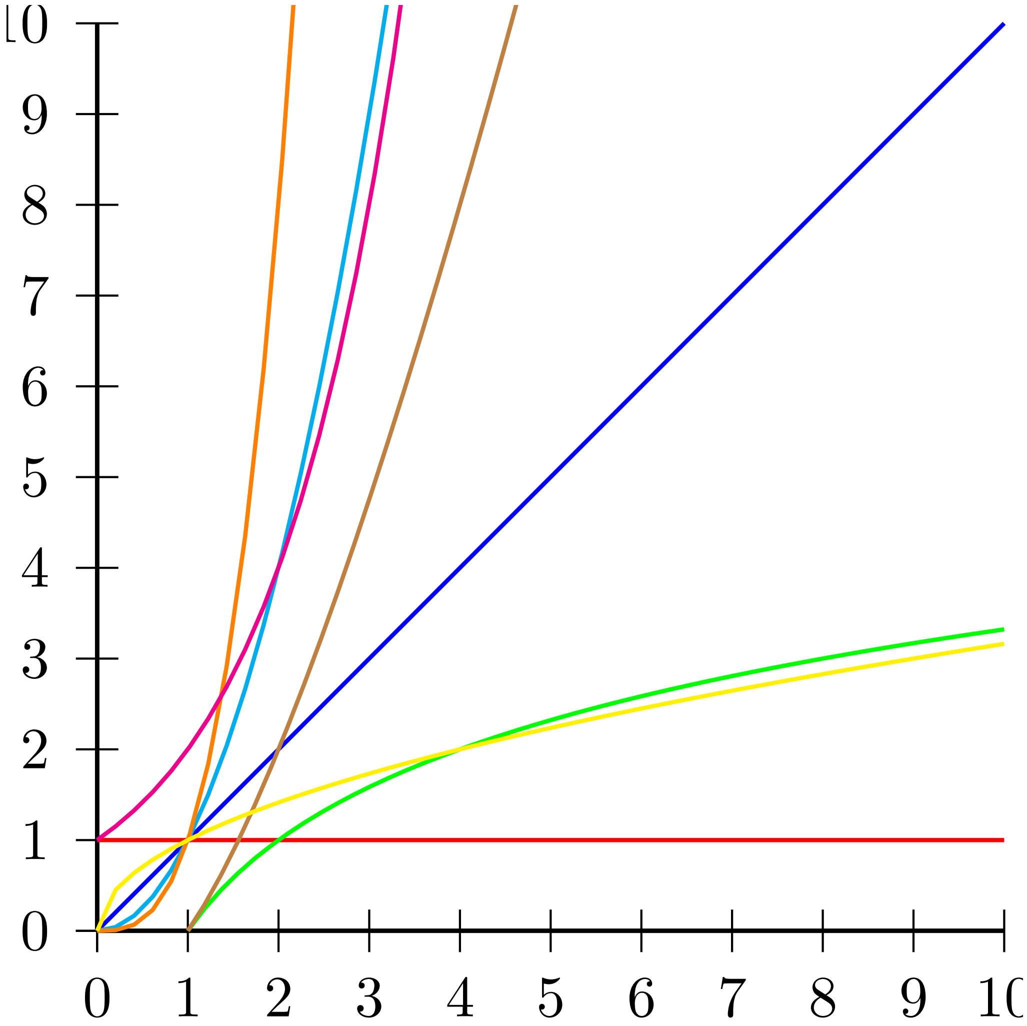
$\Omega(n)$

Tanım 2 $f(n)$ *ve* $g(n)$ *pozitif tamsayılardan reel sayılara tanımlı iki fonksiyon olsun. Eğer her* $n > n_0$ *tamsayısı için* $f(n) > cg(n)$ *olacak şekilde* c *ve* n_0 *sabitleri varsa,* $f = \Omega(g)$*'dir ve* f *fonksiyonu için* g *fonksiyonu bir alt sınır belirler.*

$\Theta(n)$

Tanım 3 *$f(n)$ ve $g(n)$ pozitif tamsayılardan reel sayılara tanımlı iki fonksiyon olsun. Eğer her $n > n_0$ tamsayısı için $c_1 g(n) > f(n) > c_2 g(n)$ olacak şekilde c_1, c_2 ve n_0 sabitleri varsa, $f = \Theta(g)$'dir ve f fonksiyonu için g fonksiyonu hem üst hem de bir alt sınır belirler.*

Algoritma analizinde sıklıkla kullanılan 8 temel fonksiyonun büyüme oranları



Yinelemesiz Programların Analizi

Püf Nokta (1)

Bir for döngüsünün çalışma süresi döngünün içindeki satırların çalışma süreleri ile döngünün tekrar sayısının çarpımı kadardır.

```
int kare_toplami(int N){
   int i, toplam = 0;
   for (i = 1; i <= N; i++)
      toplam += i * i;
   return toplam;
}
```

$$
\begin{aligned}
T(N) &= \sum_{i=1}^{N} 1 \\
&= N \in \mathcal{O}(N)
\end{aligned}
$$

Püf Nokta (2)

Birden fazla döngü iç içe olduğunda fonksiyonun çalışma süresi bütün döngülerin çalışma sürelerinin çarpımı kadardır.

```
toplam = 0;
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        toplam++;
```

Püf Nokta (2)

$$
\begin{aligned}
T(N) &= \sum_{i=0}^{N-1} \underbrace{\sum_{j=0}^{N-1} 1}_{N} \\
&= \sum_{i=0}^{N-1} N \\
&= N \underbrace{\sum_{i=0}^{N-1} 1}_{N} \\
&= N^2 \in \mathcal{O}(N^2)
\end{aligned}
$$

Püf Nokta (3)

Ardışık program parçaları söz konusu olduğunda çalışma süresi en fazla olan program parçasının çalışma süresi tüm programın çalışma süresi olarak kabul edilir.

```
toplam = 0;
for (i = 1; i <= N; i++)
    toplam += i * i;
for (i = 0; i < N; i++)
    for (j = 0; j < N; j++)
        toplam++;
```

$$T(N) = \mathcal{O}(N) + \mathcal{O}(N^2) \in \mathcal{O}(N^2)$$

Püf Nokta (4)

If/Else koşul satırının çalışma süresi, koşulun doğru veya yanlış olduğu durumların çalışma sürelerinin en fazlası kadardır.

```
toplam = 0;
if (N % 2 == 0)
    for (i = 1; i <= N; i++)
        toplam += i * i;
else
    for (i = 0; i < N; i++)
        for (j = 0; j < N; j++)
            toplam++;
```

$$T(N) = \mathcal{O}(N^2)$$

Püf Nokta (5)

Döngü değişkeninin birer birer artmadığı / azalmadığı durumlarda döngü değişkeninin döngü süresince aldığı değerleri belirlemek döngünün çalışma süresini belirlemede yardımcı olacaktır.

```
int basamak_sayisi(int N){
   int i, sayi = 1;
   while (N > 1){
      sayi++;
      N = N / 2;
   }
   return sayi;
}
```

Püf Nokta (5)

İlk döngü sonundaki değeri $\frac{N}{2}$, ikinci döngü sonundaki değeri $\frac{N}{4} = \frac{N}{2^2}$, üçüncü döngü sonundaki değeri $\frac{N}{8} = \frac{N}{2^3}$, ..., k'nıncı döngü sonundaki değeri ise $\frac{N}{2^k}$ olacaktır.

$$
\begin{aligned}
\frac{N}{2^k} &= 1 \\
N &= 2^k \\
k &= \log_2 N
\end{aligned}
$$

sonucunda toplam tekrar sayısı $\log_2 N$, fonksiyonun çalışma süresi de $\mathcal{O}(\log N)$ olur.

Özyinelemeli Programların Analizi

Özyinelemeli Programların Analizi

- Özyinelemeli programların analizi, özyinelemeli olmayan programların analizi gibi yapılamaz.
- Bunun temel nedeni, programın çalışma süresinin sadece yapılan işlemlere değil aynı zamanda programın kendi çalışma süresinin bir fonksiyonuna da bağlı olmasıdır.

Örnek (1)

```
int faktoryel(int N){
   if (N <= 1)
      return 1;
   else
      return N * faktoryel(N - 1);
}
```

N girdisi için çalışma süresi yine faktöryel fonksiyonunun $N-1$ girdisi için çalışma süresine bağlıdır.

Örnek (1)

$$
\begin{aligned}
f(1) &= 1 \\
f(N) &= f(N-1) + 1
\end{aligned}
$$

- Birinci denklem $N = 1$ için fonksiyonun yaptığı işlem sayısını, yani `return 1;` satırındaki döndürme işlemini
- İkinci denklemdeki +1 ise `return N * faktoryel(N - 1);` satırındaki çarpma işlemini göstermektedir.

Örnek (1)

İkinci denklemdeki N yerine sırayla $N-1$, $N-2$, ..., 2 koyarak

$$\begin{aligned} f(N) &= f(N-1)+1 \\ f(N-1) &= f(N-2)+1 \\ f(N-2) &= f(N-3)+1 \\ & \cdots \\ f(2) &= f(1)+1 \end{aligned}$$

elde ederiz. Eşitlikleri taraf tarafa topladığımızda, $f(N-1)$, $f(N-2)$, ..., $f(2)$ sadeleşir ve geriye

$$\begin{aligned} f(N) &= f(1)+N-1 \\ f(N) &= N \in \mathcal{O}(N) \end{aligned}$$

Örnek (2)

```
int basamak_sayisi(int N){
   if (N == 1)
      return 1;
   else
      return 1 + basamak_sayisi(N / 2);
}
```

Bu fonksiyonun çalışma süresi (girdi N için) yine bu fonksiyonun çalışma süresine (girdi N / 2 için) bağlıdır.

Örnek (2)

$$\begin{aligned} b(1) &= 1 \\ b(N) &= b(N/2) + 1 \end{aligned}$$

- Birinci denklem $N = 1$ için fonksiyonun yaptığı işlem sayısını, yani `return 1;` satırındaki döndürme işlemini
- İkinci denklemdeki +1 ise `return 1 + basamak_sayisi(N / 2);` satırındaki toplama işlemini göstermektedir.

Örnek (2)

$N = 2^k$ varsayıp N yerine sırayla $N/2 = 2^{k-1}$, $N/2^2 = 2^{k-2}$, $\ldots$, $N/2^{k-1} = 2$ koyarak

$$\begin{aligned} b(2^k) &= b(2^{k-1}) + 1 \\ b(2^{k-1}) &= b(2^{k-2}) + 1 \\ b(2^{k-2}) &= b(2^{k-3}) + 1 \\ &\cdots \\ b(2^1) &= b(1) + 1 \end{aligned}$$

elde ederiz. Eşitlikleri taraf tarafa topladığımızda, $b(2^{k-1})$, $b(2^{k-2})$, ..., $b(2)$ sadeleşir ve geriye

$$\begin{aligned} b(2^k) &= b(1) + k \\ b(2^k) &= k + 1 \\ b(N) &= \log_2 N + 1 \in \mathcal{O}(\log N) \end{aligned}$$

Örnek (3)

```
void hanoi(int N, int sutun1, int sutun2){
   sutun3 = 6 - sutun1 - sutun2;
   if (N = 1)
       hareketEttir (sutun1, sutun2);
   else{
      hanoi(N - 1, sutun1, sutun3);
       hareketEttir (sutun1, sutun2);
      hanoi(N - 1, sutun3, sutun2);
   }
}
```

Hanoi fonksiyonunun çalışma süresi (girdi N için) yine bu fonksiyonun çalışma süresine (girdi $N-1$ için) bağlıdır.

Örnek (3)

$$
\begin{aligned}
h(1) &= 1 \\
h(N) &= 2h(N-1)+1
\end{aligned}
$$

- Birinci denklem N= 1 için fonksiyonun yaptığı işlem sayısını, yani `hareketEttir (1, 3);` satırındaki işlemi
- İkinci denklemdeki 2h(N - 1), 2 kere özyinelemeli çağırılan `hanoi(N - 1, ...)` satırını; +1 ise `hareketEttir (1, 3);` satırındaki işlemi göstermektedir

Örnek (3)

İkinci denklemdeki N yerine sırayla $N-1$, $N-2$, ..., 2 koyarak

$$
\begin{aligned}
h(N) &= 2h(N-1)+1 \\
h(N-1) &= 2h(N-2)+1 \\
h(N-2) &= 2h(N-3)+1 \\
&\cdots \\
h(2) &= 2h(1)+1
\end{aligned}
$$

elde ederiz.

Örnek (3)

İkinci denklemi 2 ile, üçüncü denklemi 4= 2^2 ile, . . . çarparsak,

$$
\begin{aligned}
h(N) &= 2h(N-1)+1 \\
2h(N-1) &= 2^2h(N-2)+2 \\
2^2h(N-2) &= 2^3h(N-3)+2^2 \\
&\dots \\
2^{N-1}h(2) &= 2^Nh(1)+2^{N-1}
\end{aligned}
$$

olur. Eşitlikleri taraf tarafa topladığımızda, $h(N-1)$, $h(N-2)$, . . ., $h(2)$ sadeleşir ve geriye

$$
\begin{aligned}
h(N) &= 2^Nf(1)+2^{N-1}+2^{N-2}+\ldots+1 \\
h(N) &= \sum_{i=0}^{N} 2^i \\
h(N) &= 2^{N+1}-1 \in \mathcal{O}(2^N)
\end{aligned}
$$

Temel Teorem

Özyinelemeli Problem Bölümü

N büyüklüğünde bir problemi özyinelemeli çözmek için

- N / b büyüklüğünde a tane alt problem çözüyor
- bu alt problemlerin çözümlerini de $\mathcal{O}(N^d)$ zamanda birleştirip ana probleme çözüm buluyorsak

ana problemi çözmek için harcayacağımız zaman

$$T(N) = aT(N/b) + \mathcal{O}(N^d)$$

Temel Teorem

Teorem 1 *a, b ve d $a > 0$, $b > 1$, $d \geq 0$ koşullarını sağlayan birer reel sayı olmak üzere,*

$$T(N) = aT(N/b) + \mathcal{O}(N^d)$$

denkleminin çözümü

$$T(N) = \begin{cases} \mathcal{O}(N^d) & \text{eğer } d > \log_b a \\ \mathcal{O}(N^d \log N) & \text{eğer } d = \log_b a \\ \mathcal{O}(N^{\log_b a}) & \text{eğer } d < \log_b a \end{cases}$$

olarak verilir.