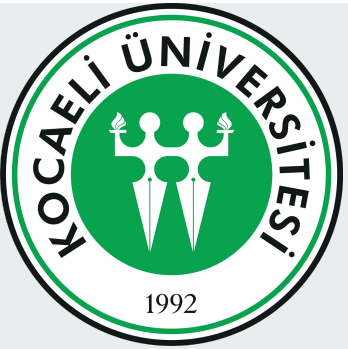


Kesikli Matematik

Algoritma Analizi

Dr. Öğr. Üyesi Mehmet Ali ALTUNCU
Bilgisayar Mühendisliği

Algoritma Analizi

- Algoritma analizi, algoritmanın yürütülmesi için gerekli kaynak miktarının belirlenmesidir.
- Belirli bir problemi çözen herhangi bir algoritmanın ihtiyaç duyduğu kaynaklar için teorik tahminler sağlar.
- Başka bir ifadeyle, algoritmanın performansı ve kaynak kullanımı konusunda yapılan teorik çalışmaların tümüne **algoritma analizi** denir.
- Buradaki çalışmalar, herhangi bir programlama dilinden bağımsız bir şekilde yürütülür ki, gerçek anlamda sadece algoritmanın kendisi analiz edilip bilimsel bir yaklaşım benimsenebilsin.

Algoritmaların Performans Kriterleri

- Algoritma verimliliğinde ya da performansında iki kriter vardır:
- **Zaman Karmaşıklığı (Time)**
- **Bellek Alan Karmaşıklığı (Memory)**
- Zaman karmaşıklığı bir algoritmanın girdi ile çıktı arasında geçen zamanı hesaplar.
- Bellek karmaşıklığı ise harcanan bellek alanını hesaplar. Veri büyüdükçe bu zaman ve belleğin nasıl değiştiğini analiz eder.

Çalışma Zamanı

- Çalışma zamanı, ‘n’ boyutlu bir problemin algoritmasını çalıştırmak için gerekli olan zamana denir.
- Başka bir ifadeyle karşılaştırma, döngü çevirimi, aritmetik işlemler gibi algoritmanın işlevini yerine getirmesi için temel kabul edilen işlemlerin kaç kere yürütüldüğünü veren bir bağıntıdır.
- n boyutlu bir veride algoritmanın yaptığı temel işlemlerin sayısı **T(n)** ile gösterilir.

Çalışma zamanı hesabında dikkat edilmesi gereken temel hesap birimleri şu şekildedir:

- Programlama dilindeki deyimler
- Döngü sayıları
- Toplam işlem sayısı
- Dosyaya erişim sayısı
- Atama sayısı

Çalışma Zamanı

Örnek: Dizideki en küçük elemanı bulan bir algoritmaya bakalım.

```
int bulEnkucuk(int A[], int n){
   int enkucuk;
   int k;
   enkucuk = A[0];                  // 1 işlem
   for (k = 0; k < n; k++)          //  k değerine 0 ataması 1 kez, k değerinin n değerinden küçük olup   olmadığı kontrolü n+1 kez ve k değerini artırma işlemi n kez
         if (A[k] < enkucuk)        // n kez
         enkucuk = A[k];            // n-1 kez
   return enkucuk;                  // 1 işlem
}
```

- Algoritmanın çalışma zamanı **T(n) = 4n + 3** olur.

Zaman Karmaşıklığı

- Zaman karmaşıklığı, bir algoritmadaki yürütme zamanının derecesinin asimptotik notasyonlarla gösterilmesine denir. Başka bir ifadeyle, algoritmanın asimptotik notasyona göre karmaşıklık mertebesidir.
- Burada amaç, algoritmadaki eleman sayısı çok fazla olduğunda ya da veri seti sonsuza gittiğinde, giriş boyutu ve zaman arasındaki karmaşıklığı azaltmaktır.
- Yani detaylardan kurtularak çalışma süresi analizini basitleştirmektir.
- Bunun için asimptotik gösterimde, fonksiyon içerisindeki sabitler ve katsayılar gibi sonsuza giderken büyümeye pek etkisi olmayan önemsiz kısımlar atılır.
- Böylelikle geriye, verinin büyümesine bağlı olarak fonksiyonun büyümesinde en büyük etkiye sahip olan parametre kalır ve gerçek fonksiyona göre yaklaşık bir değer bulunabilir.

Zaman Karmaşıklığı

- Zaman karmaşıklığı genel olarak özel gösterimler ile tanımlanmaktadır.

  Bunlardan bazıları;

  - Büyük - *O* Notasyonu
  - Büyük - Omega (Ω) Notasyonu
  - Büyük Theta (Θ) Notasyonu

şeklindedir.

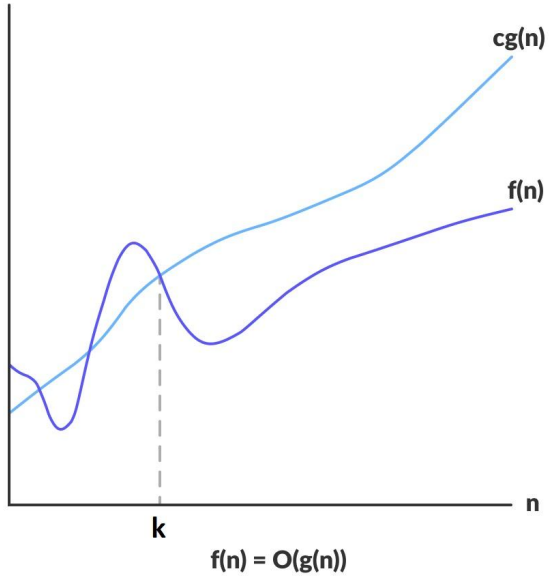
Büyük - *O* Notasyonu

Tanım: *f* ve *g* fonksiyonları reel sayılardan reel sayılara tanımlı iki fonksiyon olsun.

- *f(x)* fonksiyonu için c ve k sabit olmak üzere;
- $|f(x)| \leq c|g(x)|$ ve $x>k$ durumunda;
- ***f(x) = O(g(x))*** *olur. Yani; f(x)* fonksiyonun büyüme hızı (Büyük-*O*) *g(x)* olmuş olur.
- *f(x)*'in = *O*(g(x)) olduğunu göstermek istiyorsak bunu sağlayan c ve k ikilisi bulmamız gerekir.

Büyük - *O* Notasyonu

- Büyük-*O* Notasyonu, algoritma analizindeki zaman karmaşıklığında üst sınırı tanımlayan, matematiksel bir gösterimdir.



Büyük - *O* Notasyonu

Örnek: $f(n) = 3n^2+8n$ 'in $O(n^2)$ olduğunu gösteriniz.

Çözüm: $3n^2+8n \leq 3n^2+8n^2$ ⇨ $3n^2+8n \leq \mathbf{11}n^2$ olduğuna göre $n \geq 1$ ve $c=11$ olması durumunda $f(n)$, $O(n^2)$ olur.

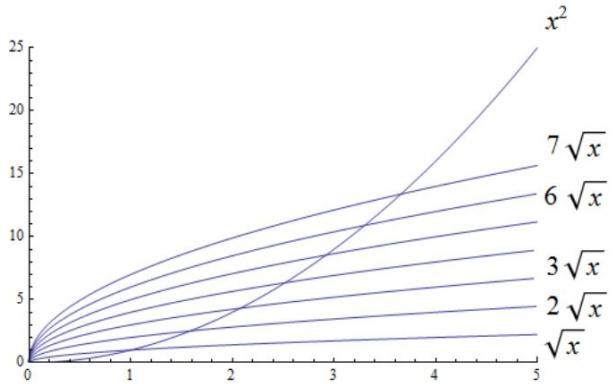
Örnek: $f(n) = 3n^2+8n$ 'in $O(n^3)$ olduğunu gösteriniz.

Çözüm: $3n^2+8n \leq 3n^3+8n^3$ ⇨ $3n^2+8n \leq \mathbf{11}n^3$ olduğuna göre $n \geq 1$ ve $c=11$ olması durumunda $f(n)$, $O(n^3)$ olur.

Büyük - *O* Notasyonu

Örnek: *f*(n) = x^2 'nin *O*(√x) **olmadığını** gösteriniz.

Çözüm: √x, √2x, √3x ve x^2 grafiğine baktığımızda x^2'nin aşağıdaki şekilde olduğu gibi herhangi bir √Ax 'den daha büyük olduğuna dikkat edin. Büyük-*O* Notasyonu, algoritma analizindeki zaman karmaşıklığında üst sınırı tanımlar. √Ax fonksiyonları ise x^2 fonksiyonunun altındadır.



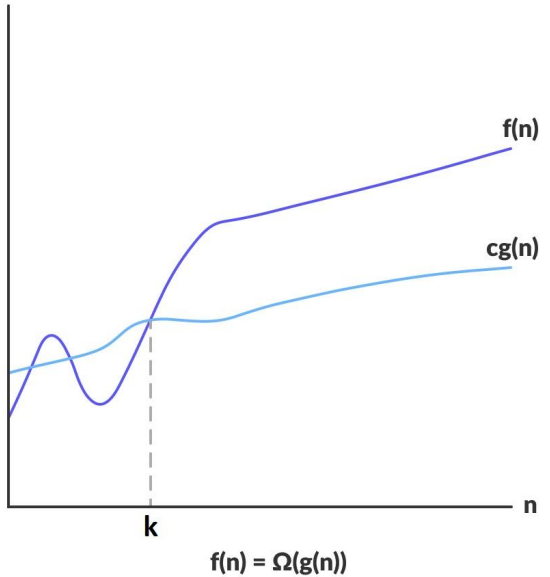
Büyük - Omega (Ω) Notasyonu

Tanım: f ve g fonksiyonları reel sayılardan reel sayılara tanımlı iki fonksiyon olsun.

- $f(x)$ fonksiyonu için c ve k sabit olmak üzere;
- $|f(x)| \geq c|g(x)|$ ve $x>k$ durumunda;
- $\boldsymbol{f(x) = \Omega(g(x))}$ olur. Yani; f(x) fonksiyonun büyüme hızı (Büyük-Omega) $g(x)$ olmuş olur.
- $f(x)$'in $= \Omega(g(x))$ olduğunu göstermek istiyorsak bunu sağlayan c ve k ikilisi bulmamız gerekir.

Büyük - Omega (Ω) Notasyonu

- Büyük-Omega Notasyonu, algoritma analizindeki zaman karmaşıklığında alt sınırı tanımlayan, matematiksel bir gösterimdir.



Büyük - Omega (Ω) Notasyonu

Örnek: $f(n) = 2n+5$ 'in $\Omega(n)$ olduğunu gösteriniz.

Çözüm: $2n+5 \geq \mathbf{2}n$ ⇨ $n \geq 1$ ve $c=2$ olması durumunda $f(n)$, $\Omega(n)$ olur.

Örnek: $f(n) = 5n^2-3n$ 'in $\Omega(n^2)$ olduğunu gösteriniz.

Çözüm: $5n^2-3n \geq \mathbf{4}n^2$ ⇨ $n \geq 3$ ve $c=4$ olması durumunda $f(n)$, $\Omega(n^2)$ olur.

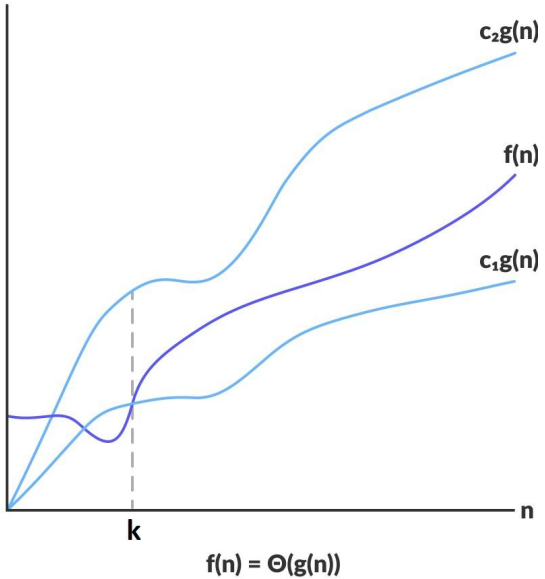
Büyük - Theta (Θ) Notasyonu

Tanım: *f* ve *g* fonksiyonları reel sayılardan reel sayılara tanımlı iki fonksiyon olsun.

- *f(x)* fonksiyonu için c ve k sabit olmak üzere;
- $c_1|g(x)| \leq |f(x)| \leq c_2|g(x)|$ ve x>k durumunda;
- ***f(x) = Θ(g(x))*** *olur. Yani; f(x)* fonksiyonun büyüme hızı (Büyük-Theta) *g(x)* olmuş olur.
- *f(x)*'in = Θ(g(x)) olduğunu göstermek istiyorsak bunu sağlayan c_1 , c_2 ve k üçlüsü bulmamız gerekir.

Büyük - Theta (Θ) Notasyonu

- Büyük-Theta Notasyonu, algoritma analizindeki zaman karmaşıklığında alt ve üst sınırı tanımlayan, matematiksel bir gösterimdir.



Büyük - Theta (Θ) Notasyonu

Örnek: $f(n) = 2n+5$ 'in $\Theta(n)$ olduğunu gösteriniz.

Çözüm: $\mathbf{2}n \leq 2n+5 \leq \mathbf{3}n$ ⇨ $n \geq 5$, $c_1=2$ ve $c_2=3$ olması durumunda $f(n)$, $\Theta(n)$ olur.

Örnek: $f(n) = 5n^2-3n$ 'in $\Theta(n^2)$ olduğunu gösteriniz.

Çözüm: $\mathbf{4}n^2 \leq 5n^2-3n \leq \mathbf{5}n^2$ ⇨ $n \geq 4$, $c_1=4$ ve $c_2=5$ olması durumunda $f(n)$, $\Theta(n^2)$ olur.

İteratif Fonksiyonların Zaman Analizi

Örnek:

for (int i = 1; i <= n; i++) {
  // code
}
 	
O(n)
	for (int i = 1; i <= n; i=i+3) {
	// code
	}
	
	O(n/20) = O(n)

for (int i = 1; i <= n; i++) {
for (int j = 1; j <= n; j++) {
  // code
}}
 	
$O(n^2)$
	for (int i = 1; i <= n; i=i*3) {
	// code
	}
	
	$O(\log_3 n)$

İteratif Fonksiyonların Zaman Analizi

Örnek (devam):

```
int search(int arr[], int N, int x)
{
   for (i = 0; i < n; i++)
      if (arr[i] == x)
         return i;
   return -1;
}
```

O(n)

```
int fact()
{

   for(i=1; i<=n; i++)
   {
      fact=fact*i;
   }
   return fact;
}
```

O(n)

Rekürsif Fonksiyonların Zaman Analizi

Örnek: Insertion Sort Algoritması

```
insertion(int arr[], int n)
{
    if (n <= 1)
      return;

   insertion(arr, n - 1);
   int last = arr[n - 1];
   int j = n - 2;
   while (j >= 0 && last < arr[j]) {
      arr[j + 1] = arr[j];
      j--;
   arr[j + 1] = last;
```

- **Özyineleme İlişkisi;**
  - **Temel Durum : T(0) =1**
  - **n>0 için; T(n) = T(n-1) + n**

Rekürsif Fonksiyonların Zaman Analizi

Örnek: Insertion Sort Algoritması

- **Özyineleme İlişkisi : T(0) =1 , T(n) = T(n-1) + n**
- T(n-1) = T(n-2)+n-1
- T(n) = T(n-2)+n-1+n = T(n-2)+n+(n-1)
- T(n-2) = T(n-3)+n-2
- T(n) = T(n-3)+n+(n-1)+(n-2)
- T(n) = T(n-k)+n+(n-1)+(n-2)+...+(n-k+1)
- Temel duruma T(0) anında ulaşılır.
- T(0) = 1 ⇨ n-k = 0 ⇨ k = n
- T(n) = T(0)+1+2+3+...+n = 1+[n(n+1)/2] ⇨ **T(n) = *O* (n^2)**

Rekürsif Fonksiyonların Zaman Analizi

Örnek: Binary Search Algoritması

```
BinarySearch(A, start, end, target)
  if start > end then
    return NOT FOUND
  mid = floor((start + end)/2)
  if A[mid] = target then
    return mid
  else if target < A[mid] then
     return BinarySearch(A, start, mid-1, target)
   else
     return BinarySearch(A, mid+1, end, target)
```

- **Özyineleme İlişkisi;**
  - **Temel Durum : T(1) =1**
  - **n>1 için; T(n) = T (n/2) + 1**

Rekürsif Fonksiyonların Zaman Analizi

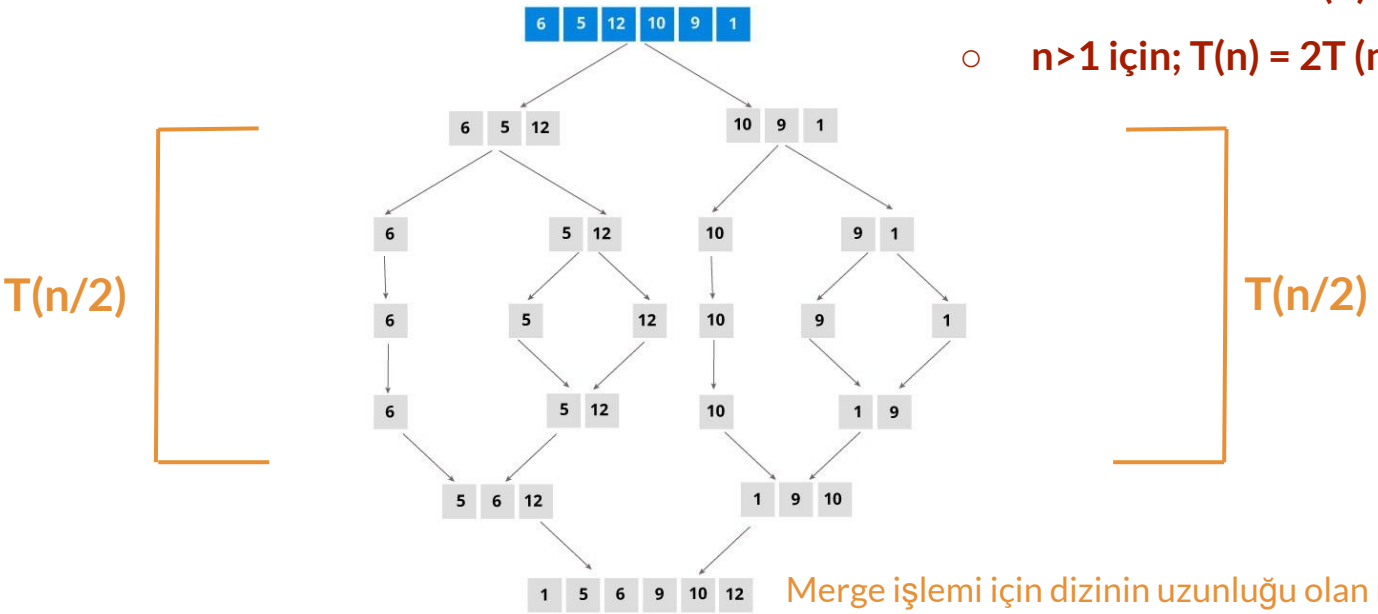
Örnek: Binary Search Algoritması

- **Özyineleme İlişkisi : T(1) =1 , T(n) = T (n/2) + 1**
- T(n/2) = T(n/4)+1
- T(n) = T(n/4)+1+1 = T(n/4)+2
- T(n/4) = T(n/8)+1
- T(n) = T(n/8)+1+1+1 = T(n/8)+3
- $T(n) = T(n/2^k)+k$
- Temel duruma T(1) anında ulaşılır.
- $T(1) = 1 \Rightarrow n/2^k = 1 \Rightarrow k = \log_2 n$
- $T(n) = T(1)+\log_2 n \Rightarrow T(n) = 1+\log_2 n \Rightarrow$ **T(n) = *O* (logn)**

Rekürsif Fonksiyonların Zaman Analizi

Örnek: Merge Sort Algoritması

- **Özyineleme İlişkisi**
  - **Temel Durum : T(1) =1**
  - **n>1 için; T(n) = 2T (n/2) + n**



Merge işlemi için dizinin uzunluğu olan n işlem

Rekürsif Fonksiyonların Zaman Analizi

Örnek: Merge Sort Algoritması

- **Özyineleme İlişkisi : $T(1) = 1$, $T(n) = 2T(n/2) + n$**
- $T(n/2) = 2T(n/4)+n/2$
- $T(n) = 2[2T(n/4)+n/2]+n = 2^2T(n/4)+2n$
- $T(n/4) = 2^*T(n/8)+n/4$
- $T(n) = 2^2[2T(n/8)+n/4]+2n = 2^3T(n/8)+3n$
- $T(n) = 2^kT(n/2^k)+kn$
- Temel duruma T(1) anında ulaşılır.
- $T(1) = 1 \Rightarrow n/2^k = 1 \Rightarrow k = \log_2 n$
- $T(n) = 2^{\log n}T(n/2^k)+n\log_2 n = n+n\log_2 n \Rightarrow$ **$T(n) = O(n\log n)$**

Rekürsif Kalıpları

- **T(n) = aT(n-b) *O*g(n)** olduğunda temel durumun T(0) için oluşması gerekir.
- **T(n) = aT(n/b) *O*g(n)** olduğunda temel durumun T(1) için oluşması gerekir.

Kaynaklar

- **Kenneth Rosen,** “Discrete Mathematics and Its Applications”, 7th Edition , McGraw Hill Publishing Co., 2012.
- https://birhankarahasan.com/algoritma-analizi-nedir-zaman-karmasikligi-big-o-gosterimi#:~:text=Belirli%20bir%20problemi%20%C3%A7%C3%B6zen%20herhangi,%C3%A7al%C4%B1%C5%9Fmalar%C4%B1n%20t%C3%BCm%C3%BCne%20algoritma%20analizi%20denir.
- chrome-extension://efaidnbmnnnibpcajpcglclefindmkaj/https://web.ogu.edu.tr/Storage/egulbandilar/Uploads/AlgoritmaAnalizi.pdf