BMB2006
VERİ YAPILARI

Doç. Dr. Murtaza CİCİOĞLU

Bursa Uludağ Üniversitesi

Bilgisayar Mühendisliği Bölümü

Bu ders notları Olcay Taner Yıldız ile Erkan TANYILDIZI hocaların notları kullanılarak hazırlanmıştır.

Hafta 2: Algoritma Analizi

Amaç:

- Çalışma süresi???
- Asimptotik Notasyonlar

Yol haritası:

- Büyük-O Gösterimi
- Yinelemesiz Programların Analizi
- Özyinelemeli Programların Analizi
- Temel Teorem

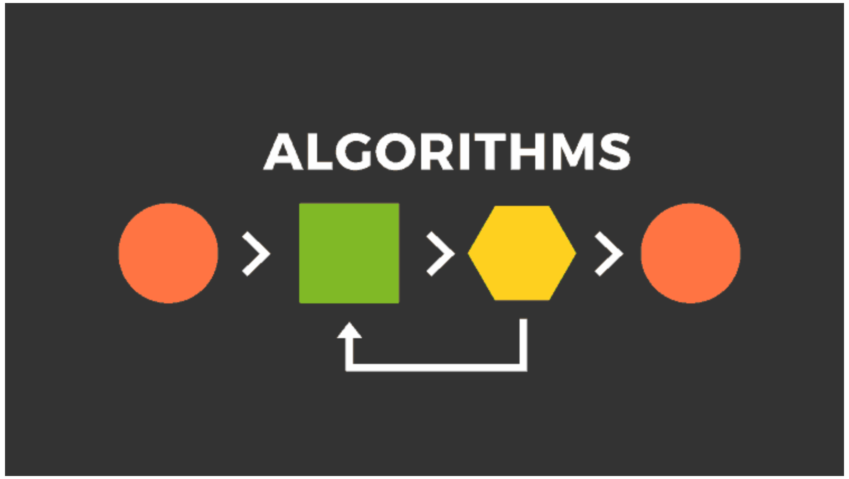
PrOGRAM GELİŞTİRME AŞAMLARI

1. Analiz
2. Algoritma oluşturma
3. Kodlama
4. Derleme ya da Yorumlama
5. Hata Ayıklama

- Algoritma, bir sorunun çözümüne gidebilmek için tasarlanan yollar, yöntemlerdir.

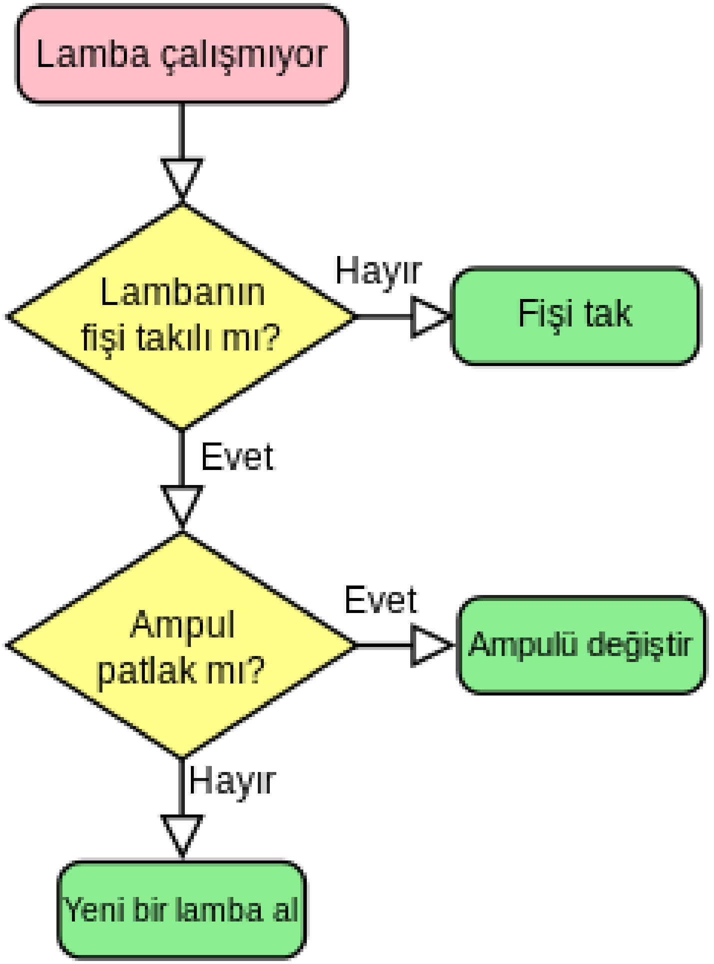
Algoritmik Program Tasarımı

- Algoritmik program tasarımı, verilen bir problemin bilgisayar ortamında çözülecek biçimde adım adım ortaya koyulması ve herhangi bir programlama aracıyla kodlanması sürecidir.
- Akış şeması(flow chart), yapılacak bir işin veya programın şekilsel/grafiksel olarak ortaya koyulmasıdır.



Algoritmik Program Tasarımı

- Etkin ve Genel Olma
- Sonlu Olma
- Kesinlik
- Doğruluk
- Giriş/Çıkış Tanımlı Olma
- Başarım (Verimli)



Algoritma Süreci

- Tasarım (design)
- Doğruluğunu ispat etme (validation)
- Analiz (analysis)
- Uygulama (implementation)
- Test

Kaba-Kod (Pseudo Code)

- Kaba-kod, bir algoritmanın yarı programlama dili kuralı, yarı konuşma diline dönük olarak ortaya koyulması/tanımlanmasıdır.
- Kaba-kod, çoğunlukla, bir veri yapısına dayandırılmadan algoritmayı genel olarak tasarlanır.
- Gerçek kod ise, algoritmanın herhangi bir programlama diliyle, belirli bir veri yapısı üzerinde gerçekleştirilmiş halidir. Bir algoritmanın gerçek kodu, yalnızca, tasarlandığı veri yapısı üzerinde koşar; veri yapısı değiştirildiğinde algoritmanın gerçek kodu üzerinde oynamalar yapılmalıdır.

Kaba-Kod (Pseudo Code)

1. Bir değer atamak için genellikle **:=** kullanılır. = işareti ise eşitlik kontrolü için kullanılır.
2. Metot, fonksiyon, yordam isimleri:
Algoritma Adı ({parametre listesi})
3. Program yapısı şu şekilde tanımlanır::
   - Karar yapıları: **if … then … else …**
   - while döngüleri: **while … do {döngü gövdesi}**
   - Tekrar döngüleri: **repeat {döngü gövdesi} until …**
   - for döngüleri: **for … do {döngü gövdesi}**
   - Dizi indeksleri: **A[i]**
4. Metotların çağrılması: **Metot adı ({değişken listesi})**
5. Metotlardan geri dönüş: **return değer**

Kaba-Kod (Pseudo Code)

- Örnek: Bir dizideki elemanların toplam ve çarpımını hesaplayan algoritmayı kaba-kod kullanarak tanımlayınız.
  - **ToplamCarpimHesapla** (dizi, toplam, çarpım)
  - **Girdi:** n sayıdan oluşan dizi.
  - **Çıktı:** dizi elemanlarının toplam ve çarpım sonucu
  - toplam=0, çarpım = 1
  - **for** i:= 1 **to** n **do**
    - toplam:= toplam + dizi[i]
    - çarpım:= çarpım* dizi[i]
  - **endfor**

Kaba-Kod (Pseudo Code) – Gerçek Kod

```
for d ← 1 to n − 1 do    //diagonal count
    for i ← 1 to n − d do
        j ← i + d
        minval ← ∞
        for k ← i to j do
            if C[i, k − 1] + C[k + 1, j] < minval
                minval ← C[i, k − 1] + C[k + 1, j];   kmin ← k
        R[i, j] ← kmin
        sum ← P[i];   for s ← i + 1 to j do sum ← sum + P[s]
        C[i, j] ← minval + sum
return C[1, n], R
```

```java
public Node search(Node root,
                                  int key)
{
    // Base Cases: root is null
    // or key is present at root
    if (root == null ||
        root.key == key)
        return root;

   // Key is greater than root's key
    if (root.key < key)
       return search(root.right, key);

    // Key is smaller than root's key
    return search(root.left, key);
}
```

Algoritma Analizi

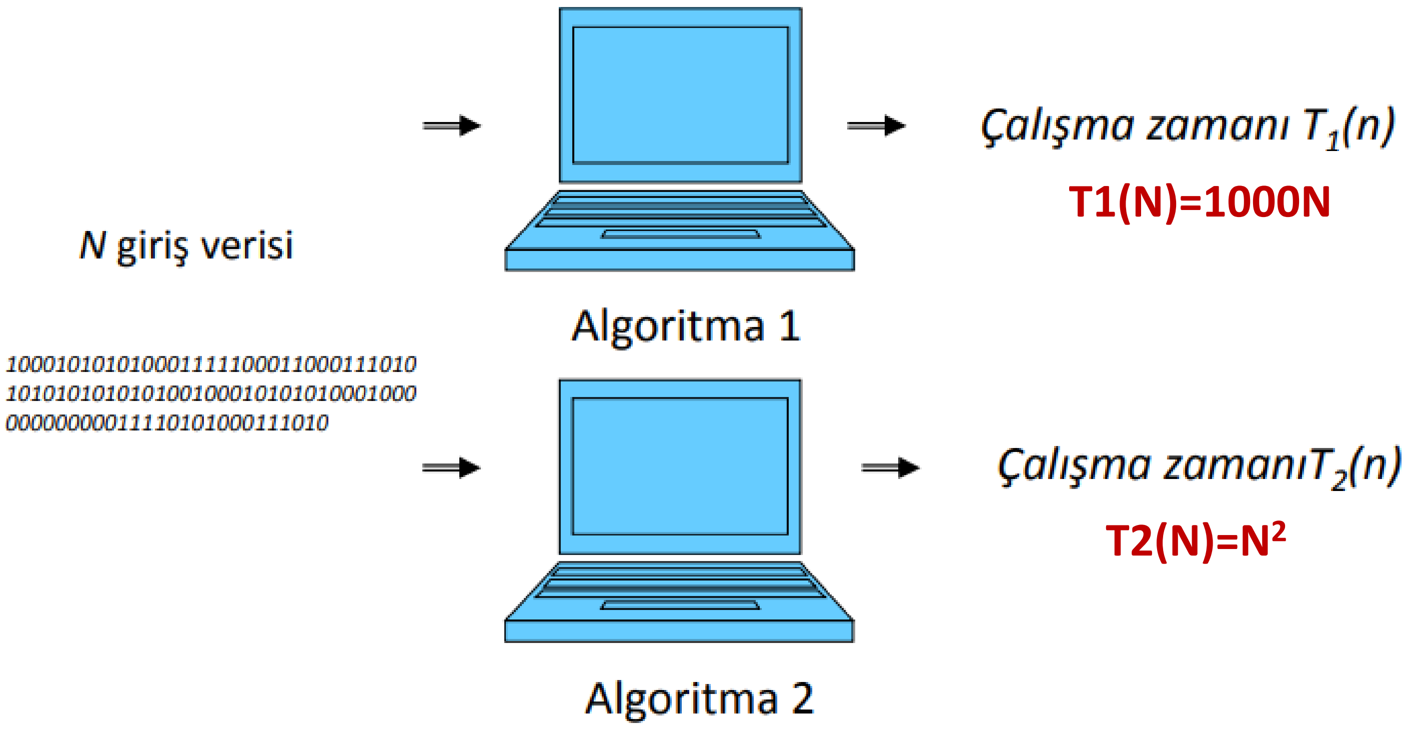
- Herhangi bir programlama dilinde yazılmış bir algoritmanın ne kadar hızlı çalıştığını veya ne kadar sürede çalıştığını o algoritmayı analiz ederek yapabiliriz.
- Peki, algoritma analizi nedir?
  - *tasarlanan program veya fonksiyonun belirli bir işleme göre matematiksel ifadesini bulmak*
- Burada temel hesap birimi seçilir ve programın görevini yerine getirebilmesi için bu işlemden kaç adet yapılması gerektiğini bulmaya yarayan bir bağıntı hesaplanır.

Algoritma Analizi

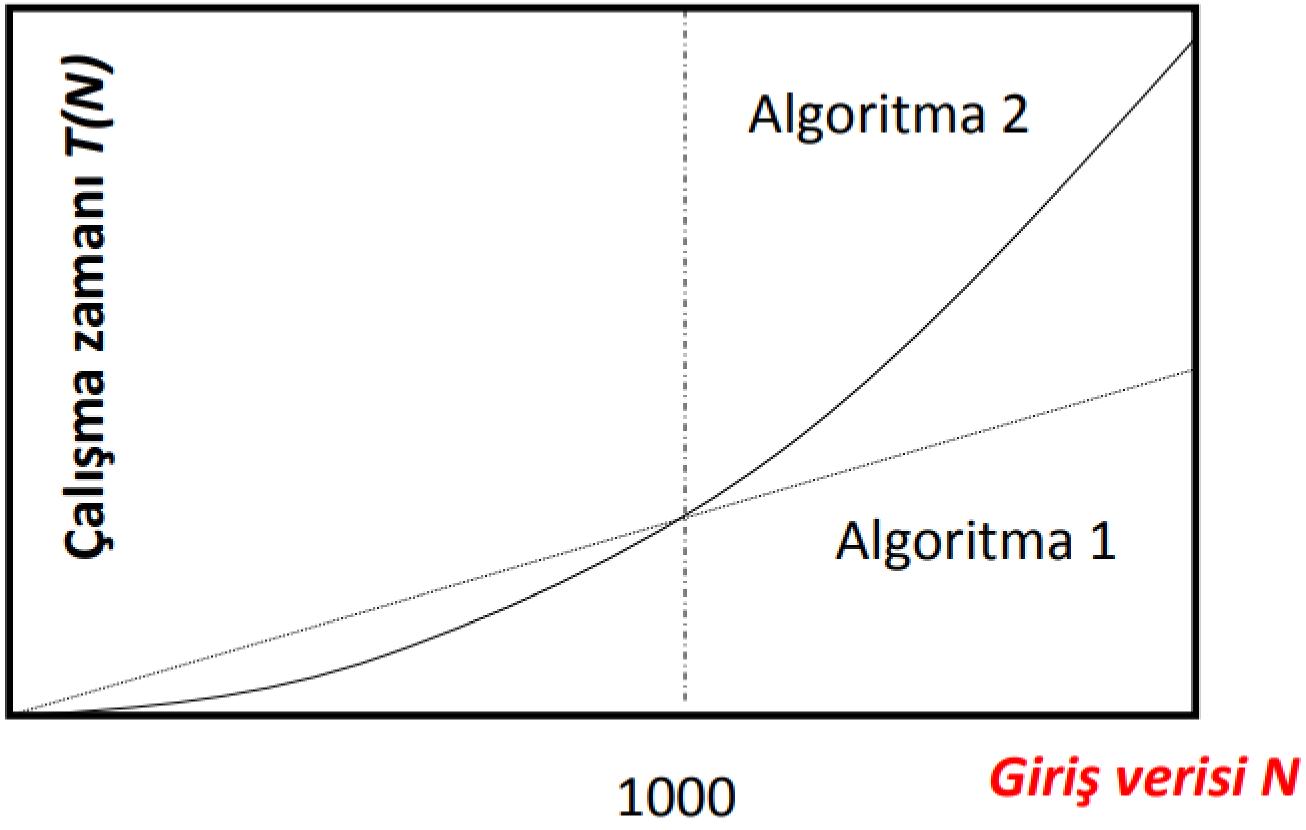
- Algoritma analizi denince akla iki önemli kavram gelir bunlar alan ve zaman karmaşıklığıdır.
- Alan karmaşıklığı yazdığınız algoritma bellekten ne kadar yer kullanıyor, zaman karmaşıklığı ise yazdığınız algoritmanın çalışma süresini ifade eder.
- Algoritma analizine neden ihtiyaç duyarız çünkü yazdığımız algoritmanın performansını bilmek isteriz, farklı algoritmalarla karşılaştırmak isteriz ve daha iyisi mümkün mü sorusuna ancak analiz yaparak cevap verebiliriz.

Algoritma Analizi

- Aynı işi yapan fakat farklı yazılmış iki algoritmaya yani verileri girdi olarak verdiğimizde farklı çalışma zamanlarında işlerini bitirdiklerini gözleriz.



Yürütme (Çalışma) Zamanı Analizi



Yürütme (Çalışma) Zamanı Analizi

N	T1	T2
10	10^{-2} sec	10^{-4} sec
100	10^{-1} sec	10^{-2} sec
1000	1 sec	1 sec
10000	10 sec	100 sec
100000	100 sec	10000 sec

Algoritma Analizi

- Yürütme zamanı(Running Time)-T(n)
  - Bir algoritma çalışmasını bitirene kadar geçen süre yürütme zamanı olarak adlandırılır. Ve algoritmada genelde eleman sayısı n olarak gösterilir ve yürütme zamanı da T(n) ile ifade edilir.
- Algoritmanın işlevini yerine getirmesi için kullandığı bellek miktarına alan maliyeti denir. Örneğin n elemanlı bir dizi, elemanlarının her biri 4 byte ise bu dizi için alan maliyeti bağıntısı;

$$T(n) = 4n$$

Algoritma Analizi

- Aynı şekilde alan karmaşıklığı da eleman sayısı çok büyük olduğu zaman alan maliyetini ifade eden asimptotik notasyonlara bakılır.
  - **Giriş verilerine bağlı olan en iyi durum** (best case)
  - **Ortalama durum** (average case); hesaplanması zordur
  - **Diğerlerine göre en kötü durum** (worst case); hesaplanması kolaydır
  - Algoritmadaki eleman sayısı çok fazla olduğunda yürütme zamanı, zaman karmaşıklığı olarak adlandırılır. Ve derecesi asimptotik notasyon ile verilir. O(o), Θ(o) veya Ω(o) gibi notasyonlar kullanılmaktadır.

Algoritma Analizi

- zaman karmaşıklığı **derleyiciden bağımsız** olmalı ve yine hesap yaparken bir çok ayrıntıyı da göz ardı etmemiz gerekir.
- Big Oh Notasyonu O(n) Paul Bachman tarafından tanıtılmıştır. Zaman karmaşıklığında üst sınırı gösterir. Bu notasyon bir çok ifadeyi sadeleştirerek göstermemizi sağlar.
- Big Omega Notasyonu Ω(o) Big Oh notasyonunun tam tersidir. Zaman karmaşıklında alt sınırı gösterir. Yani Omega ile ölçülen değerden daha hızlı bir değer elde etmeniz mümkün değildir.
- Big Theta Notasyonu Θ(o) Bu notasyon big Oh notasyonu ile big Omega notasyonu arasında ortalama bir karmaşıklığı ifade eder.

Aritmetik Ortalama için T(n) Hesabı

- Aşağıda bir dizinin aritmetik ortalamasını bulan ve sonucu çağırana gönderen bulOrta() fonksiyonun kodu verilmiştir. Bu fonksiyonun yürütme zamanını gösteren T(n) bağıntısını ayrık C dili deyimlerine göre belirleyiniz.

```
    float bulOrta(float A[], int n) {
    {
    float ortalama, toplam=0;
    int k ;
1-  for(k=0;k<n;k++)
2-          toplam+=A[k];
3-  ortalama=toplam/n
4-  return ortalama;
    }
```

Aritmetik Ortalama için T(n) Hesabı

Temel Hesap Birimi	Birim Zaman (Unit Time)	Frekans(Tekrar) (Frequency)	Toplam (Total)
float bulOrta(float A[], int n)	-	-	-
{	-	-	-
float ortalama, toplam=0;	-	-	-
int k ;	-	-	-
1- for(k=0;k<n;k++)	1,1,1	1, (n+1), n	2n+2
2- toplam+=A[k];	1	n	n
3- ortalama=toplam/n	1	1	1
4- return ortalama;	1	1	1
}	-	-	-
			T(n)=3n+4

En Büyük Eleman Bulma için T(n) Hesabı

```
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
- 1 defa return enBuyuk
- T(n)=4N

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
- 1 defa return enBuyuk
- $T(n)=3N+1$

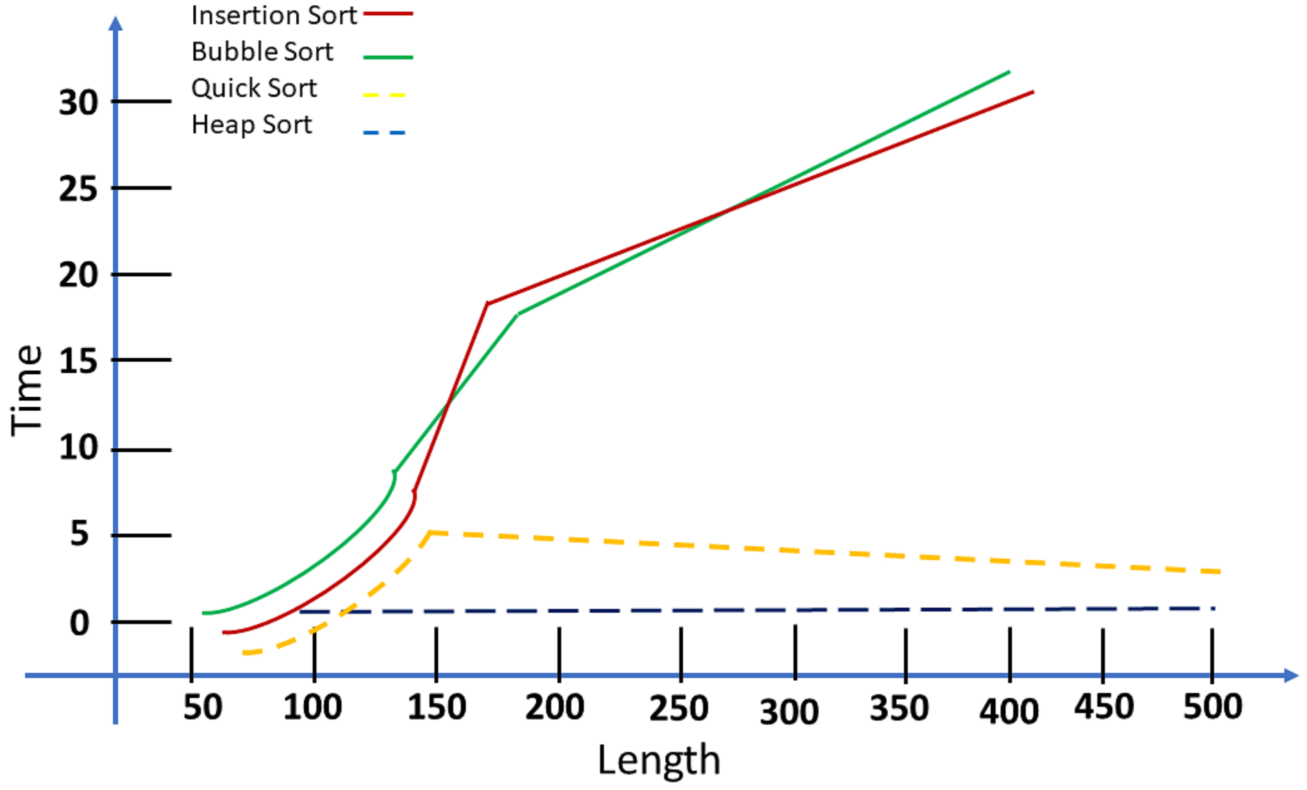
Matris Toplama için T(n) Hesabı

Temel Hesap Birimi	Birim Zaman (Unit Time)	Frekans(Tekrar) (Frequency)	Toplam (Total)
void toplaMatris (A,B,C) {	-	-	-
int A[n][m], B[n][m], C[n][m];	-	-	-
int i,j ;	-	-	-
1- for(i=0;i<n;i++)	1,1,1	1,(n+1),n	2n+2
2- for(j=0;j<m;j++)	1,1,1	n(1,(m+1),m)	n(2m+2)
3- C[i][j]=A[i][j]+B[i][j];	1	nm	nm
}	-	-	-
	T(n,m)=3nm+4n+2		
n=m ise	$T(n)=3n^2+4n+2$		

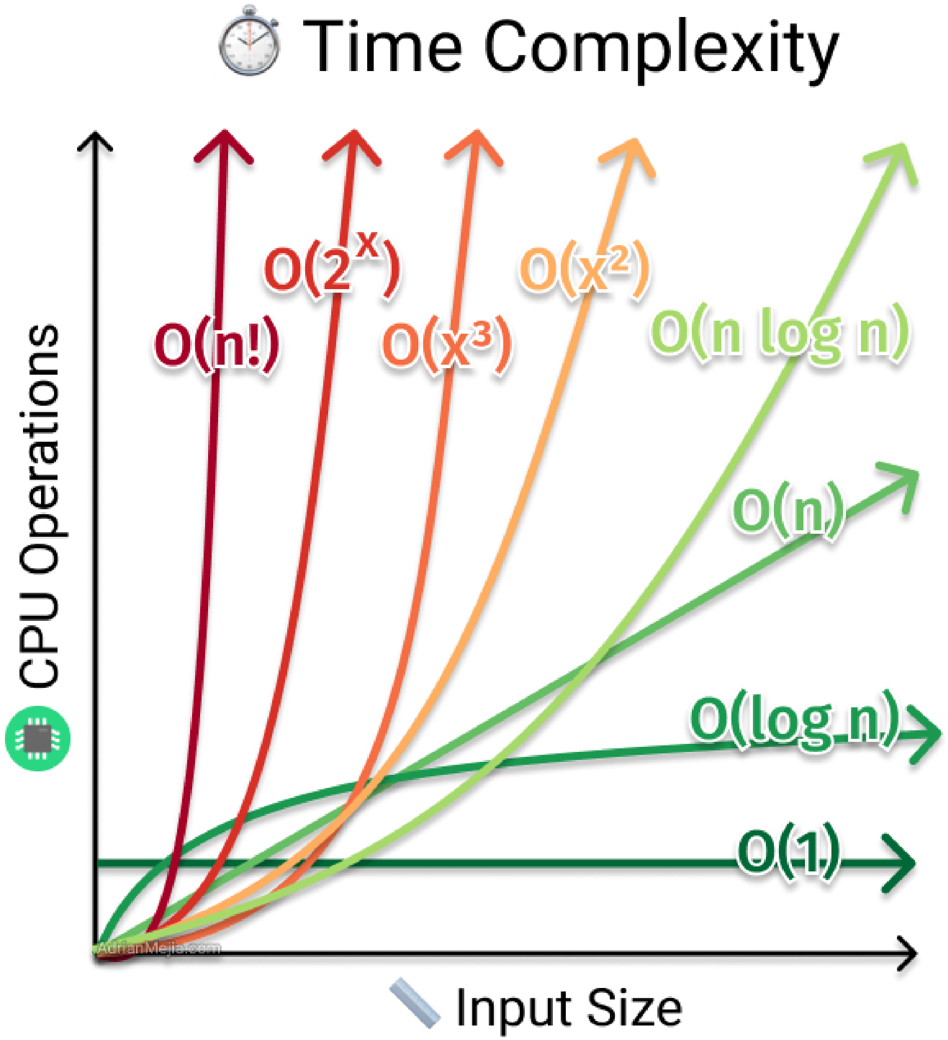
$$T(N)=\sum_{i=1}^{N}\sum_{j=1}^{N}1=\sum_{i=1}^{N}N=N*N=N^2$$

Karmaşıklık (Complexity)

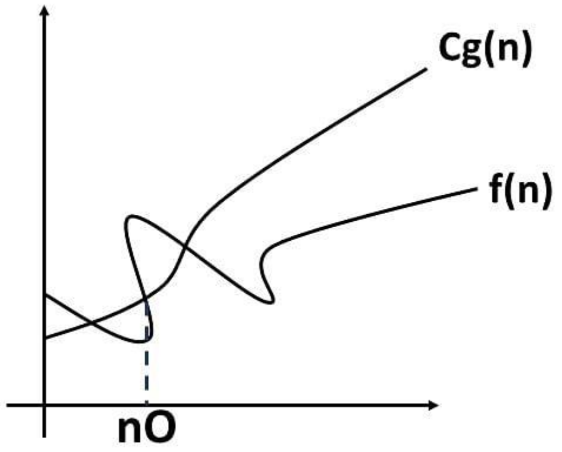
- Karmaşıklık; bir algoritmanın çok sayıda parametre karşısındaki değişimini gösteren ifadelerdir.



Karşılaşılan Genel Fonksiyonlar



Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır (En kötü durum analizi)



f(n) = O(g(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.*

Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır (En kötü durum analizi)

- x algoritması için Tx(N)=1000N
- O notasyonu cinsinden çalışma süresi Tx(N)=1000N → O(N)
- 1000N = O(N) ifadesini ispatlamak için;
- 1000N<=cN $\forall N>=n_0$ eşitsizliğini belirli bir c ve n_0 sabitleri için ispatlamak gerekir.
- c=2000 ve n_0 = 1 seçildiği zaman eşitsizliğin n_0 dan büyük her N değeri için sağlandığı görülür.

Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır (En kötü durum analizi)

- y algoritması için $Ty(N)=N^2$
- O notasyonu cinsinden çalışma süresi $Ty(N)=N^2 \rightarrow O(N^2)$
- $N^2 \rightarrow O(N^2)$ ifadesini ispatlamak için;
- $N^2 <= cN^2$ $\forall N >= n_0$ eşitsizliğini belirli bir c ve n_0 sabitleri için ispatlamak gerekir.
- $c=1$ ve $n_0 = 1$ seçildiği zaman eşitsizliğin n_0 dan büyük her N değeri için sağlandığı görülür.

O Notasyonu- Asimtotik Üst Limit

- $7n^2 + 5=O(n)$ ifadesinin doğruluğunu ispatlayınız.
- O notasyonuna göre $7n^2 + 5<=cn \quad \forall n>=n_0$ olması gerekir. Bu şartı sağlayacak c ve n_0 değerleri bulabilir miyiz?
- Her iki tarafı n sayısına bölersek;
- ***7n+5/n <=c*** elde edilir.
- Buna göre eşitliğin sağlanabilmesi için n sayısı değiştikçe c de değişmelidir. Sabit bir c ve n_0 çifti yoktur.
- Eşitsizlik doğru değildir.

O Notasyonu- Asimtotik Üst Limit

- $7n^2 + 5 = O(n^2)$ ifadesinin doğruluğunu ispatlayınız.
- O notasyonuna göre $7n^2 + 5 <= cn^2$ $\forall n >= n_0$ olması gerekir. Bu şartı sağlayacak c ve n_0 değerleri bulabilir miyiz?

- c=12 $n_0 = 1$ veya
- c=8 $n_0 = 5$ değerleri eşitsizliği n_0 dan büyük her n değeri için sağlanmaktadır.
- Çözüm kümesini sağlayacan kaç tane c ve n çifti olduğu önemli değil, tek bir çift olması notasyonun doğruluğunu ispatlamak için yeterlidir.

O Notasyonu- Asimtotik Üst Limit

- $T(n) = 2n+5$ is $O(n^2)$ Neden?
- $n>=n_0$ şartını sağlayan tüm sayılar için $T(n) = 2n+5 <= c*n^2$ şartını sağlayan c ve n_0 değerlerini arıyoruz.
- $n>=4$ için $2n+5 <= 1*n^2$
  - $c = 1, n_0 = 4$
- $n>=3$ için $2n+5 <= 2*n^2$
  - $c = 2, n_0 = 3$
- Diğer c ve n_0 değerleri de bulunabilir.

O Notasyonu- Asimtotik Üst Limit

- O notasyonunda yazarken en basit şekilde yazarız.
- $3n^2+2n+5 = O(n^2)$
- Aşağıdaki gösterimlerde doğrudur fakat kullanılmaz
  - $3n^2+2n+5 = O(3n^2+2n+5)$
  - $3n^2+2n+5 = O(n^2+n)$
  - $3n^2+2n+5 = O(3n^2)$

O Notasyonu- Asimtotik Üst Limit

- $3n^2+2n+5 = O(n^2)$ ifadesinin doğru olup olmadığını ispatlayınız.
- $10\,n^2 = 3\,n^2 + 2\,n^2 + 5\,n^2$
- $\qquad\qquad >= 3\,n^2 + 2n + 5$ için $n >= 1$
- $c = 10$, $n_0 = 1$

O Notasyonu- Asimtotik Üst Limit

- $T(N)=O(7n^2+5n+4)$ olarak ifade edilebiliyorsa, $T(N)$ fonksiyonu aşağıdakilerden herhangi biri olabilir.
- $T(N)=n^2$
- $T(N)=4n+7$
- $T(N)=1000n^2+2n+300$

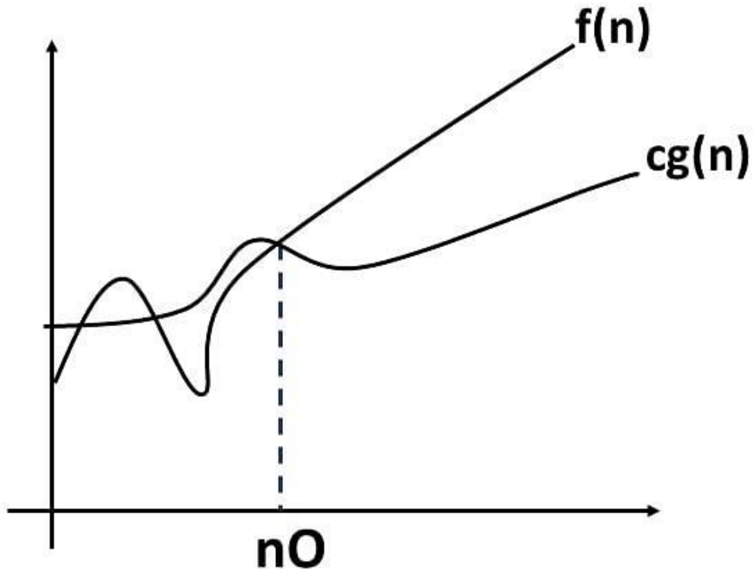
- $T(N)= O(7n^2+5n+4) =O(n^2)$

O Notasyonu- Asimtotik Üst Limit

- Fonksiyonların harcadıkları zamanları O notasyonuna göre yazınız.
- $f1(n) = 10\ n + 25\ n^2$
- $f2(n) = 20\ n \log n + 5\ n$
- $f3(n) = 12\ n \log n + 0.05\ n^2$
- $f4(n) = n^{1/2} + 3\ n \log n$

Notasyonu- Asimtotik Alt Limit (En iyi durum analizi) Ω(n)

- O notasyonun tam tersidir.



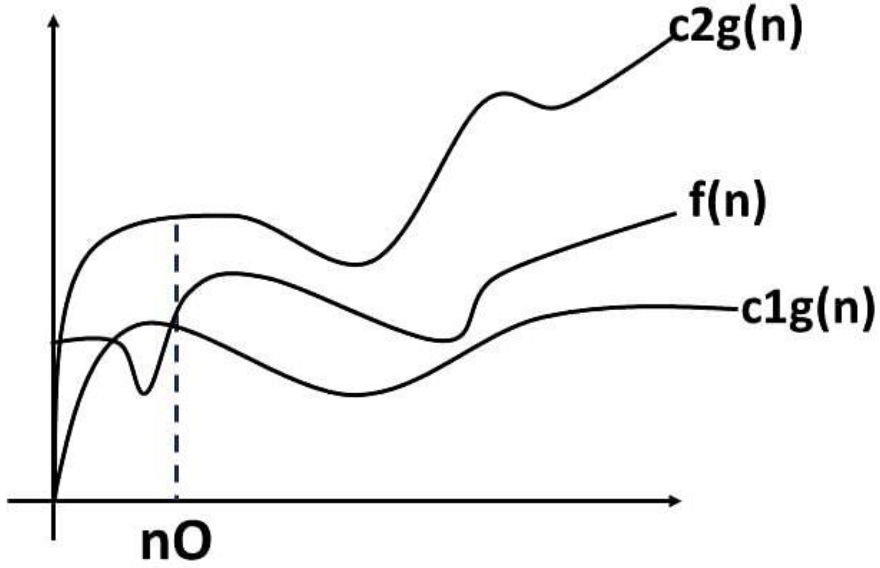
f(n) = Ω(g(n))

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.*

Ω(n) Notasyonu

- $T(n) = 2n + 5 \rightarrow \Omega(n)$ Neden?
  - $2n+5 >= 2n$, tüm $n >= 1$ için
- $T(n) = 5*n^2 - 3*n \rightarrow \Omega(n^2)$. Neden?
  - $5* n^2 - 3*n >= 4* n^2$, tüm $n >= 4$ için
- $5*n^2 \rightarrow \Omega(n)$ Neden?
  - $\exists$ c, n_0 var ki: $0<=cn<=5n^2 \rightarrow c=1$ ve $n_0=1$

Θ(n) Notasyonu (Ortalama durum analizi)



f(n) = θ(g(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.*

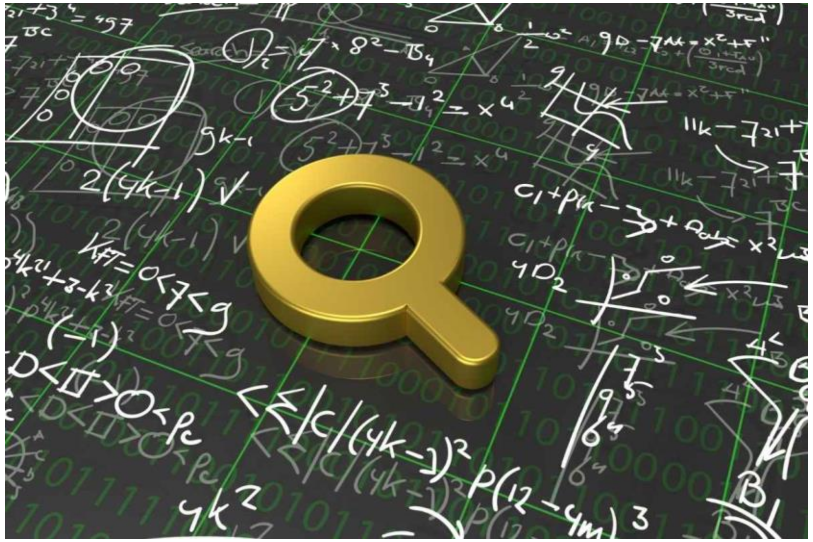
$\Theta(n)$ Notasyonu (Ortalama durum analizi)

- $T(n) = 2n + 5 \rightarrow \Theta(n)$. Neden?
  - $2n \le 2n+5 \le 3n$, tüm $n \ge 5$ için

- $T(n) = 5n^2 - 3n \rightarrow \Theta(n^2)$. Neden?
  - $4n^2 \le 5n^2 - 3n \le 5n^2$, tüm $n \ge 4$ için

Hafta 2: Algoritma Analizi

- Büyük-O Gösterimi
- Yinelemesiz Programların Analizi
- Özyinelemeli Programların Analizi
- Temel Teorem



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 oldugunda fonksiyonun çalışma süresi bütün döngülerin çalışma sürelerinin çarpımı kadardır.

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

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

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 degişkeninin birer birer artmadığı/azalmadığı durumlarda döngü degişkeninin 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)

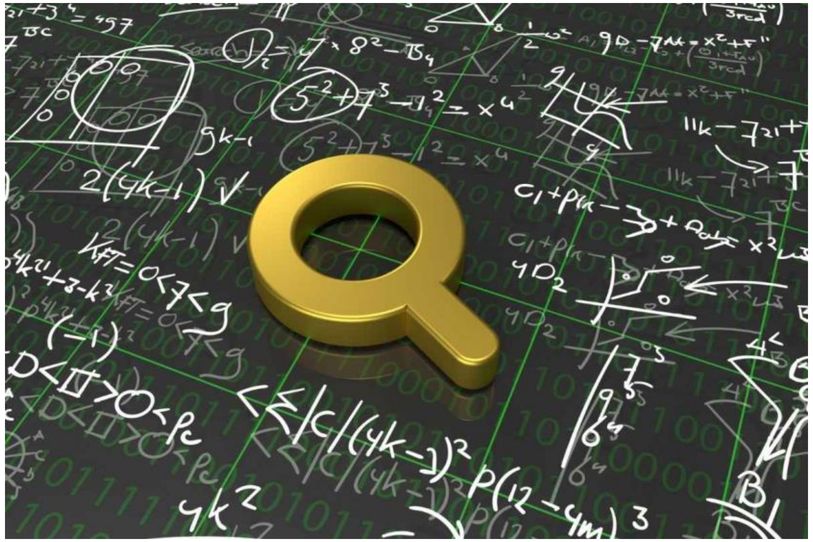
İlk döngü sonundaki değeri $\frac{N}{2}$, ikinci döngü sonundaki değeri $\frac{N}{4} = \frac{N}{2^2}$, üçüncü döngü sonundaki değeri $\frac{N}{8} = \frac{N}{2^3}$, ..., k'nıncı döngü sonundaki değeri ise $\frac{N}{2^k}$ olacaktır.

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

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

Hafta 2: Algoritma Analizi

- Büyük-O Gösterimi
- Yinelemesiz Programların Analizi
- Özyinelemeli Programların Analizi
- Temel Teorem



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);
}
```

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

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

Örnek (1)

- ikinci 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);
}
```

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

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

Örnek (2)

- $N = 2^k$ varsayıp N yerine sırayla $N/2 = 2^{k-1}$, $N/2^2 = 2^{k-2}$, . . . , $N/2^{k-1} = 2$ 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);
    }
}
```

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

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

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)

- ikinci denklemi 2 ile, üçüncü denklemi 4= 2^2 ile, . . . çarparsak,

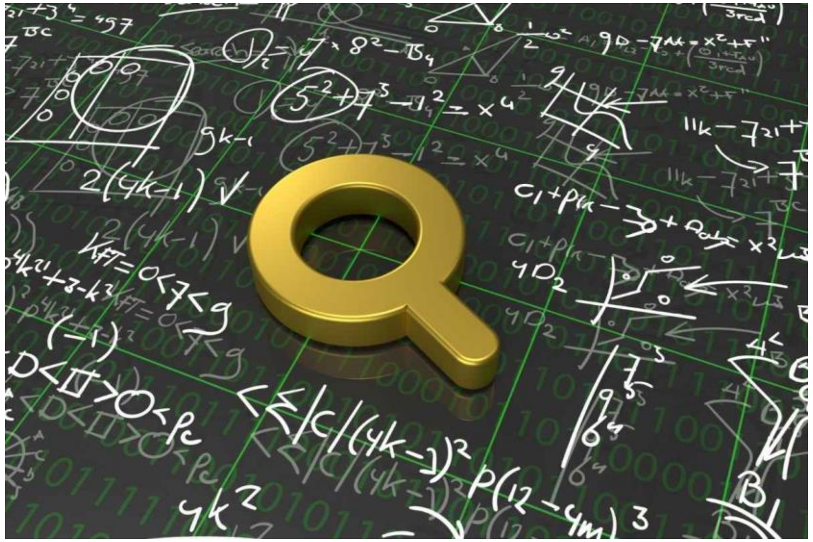
$$\begin{aligned}
h(N) &= 2h(N-1)+1 \\
2h(N-1) &= 2^2h(N-2)+2 \\
2^2h(N-2) &= 2^3h(N-3)+2^2 \\
&\cdots \\
2^{N-1}h(2) &= 2^Nh(1)+2^{N-1}
\end{aligned}$$

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

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

Hafta 2: Algoritma Analizi

- Büyük-O Gösterimi
- Yinelemesiz Programların Analizi
- Özyinelemeli Programların Analizi
- Temel Teorem (Master)



Özyinelemeli Problem

- N büyüklüğünde bir problemi özyinelemeli çözmek için
- N / b büyüklüğünde a tane alt problem çözüyor
- bu alt problemlerin çözümlerini de $O(N^d)$ 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.

Özyinelemeli Problem

- $T(n) = 9T(n/3) + n$
- a=9, b=3, d=1
- $\log_b a = \log_3 9 = 2 > d \rightarrow \mathcal{O}(N^{\log_b a})$ eğer $d < \log_b a$
- $T(n) = O(n^2)$