# Algoritmalar, Akış Şemaları ve O() Karmaşıklık Notasyonu

Öğr. Gör. M. Ozan AKI

r1.0

# Algoritmalar (Algorithms)

Algoritma, bir problemin çözümünü sağlayan ancak deneme-yanılma ve sezgisel çözüme karşıt bir yöntemdir. (V. Nabiyev)
Problemin çözümü için yapılması gereken işlemleri adım adım ve sırasıyla açıklar.

Algoritmalar, herhangi programlama diline bağımlı değildir. Bu nedenle *Pseudo Code* olarak yazılırlar.

Pseudo Code, herhangi programlama dili için geçersiz, ancak konuşma diline yakın, işlemleri açıkça tanımlayan komutlar olarak tanımlanabilir.

# Algoritmalar

Algoritmalar, genellikle spesifik bir işi açıklarlar. Bu iş parçaları büyük bir yazılımın ya da programın bir parçası olabilirler.

Günlük hayatımızdaki bir kek tarifi, algoritma olarak düşünülebilir. Tarifte işlemler ve malzemeler vardır,
bilgisayarda ise, işlemler ve veriler.

# Algoritmalar

Sonlu sayıdaki bir dizideki maksimum sayının bulunması için örnek bir algoritma şöyle yazılabilir;

1. Dizinin ilk elemanını al.
2. Dizinin sonraki elemanı ile karşılaştır
3. Karşılaştırılan eleman daha büyükse onu al
4. Kontrol edilmemiş eleman kaldıysa 2'ye git
5. Son

# Akış Şemaları (Flow Charts)

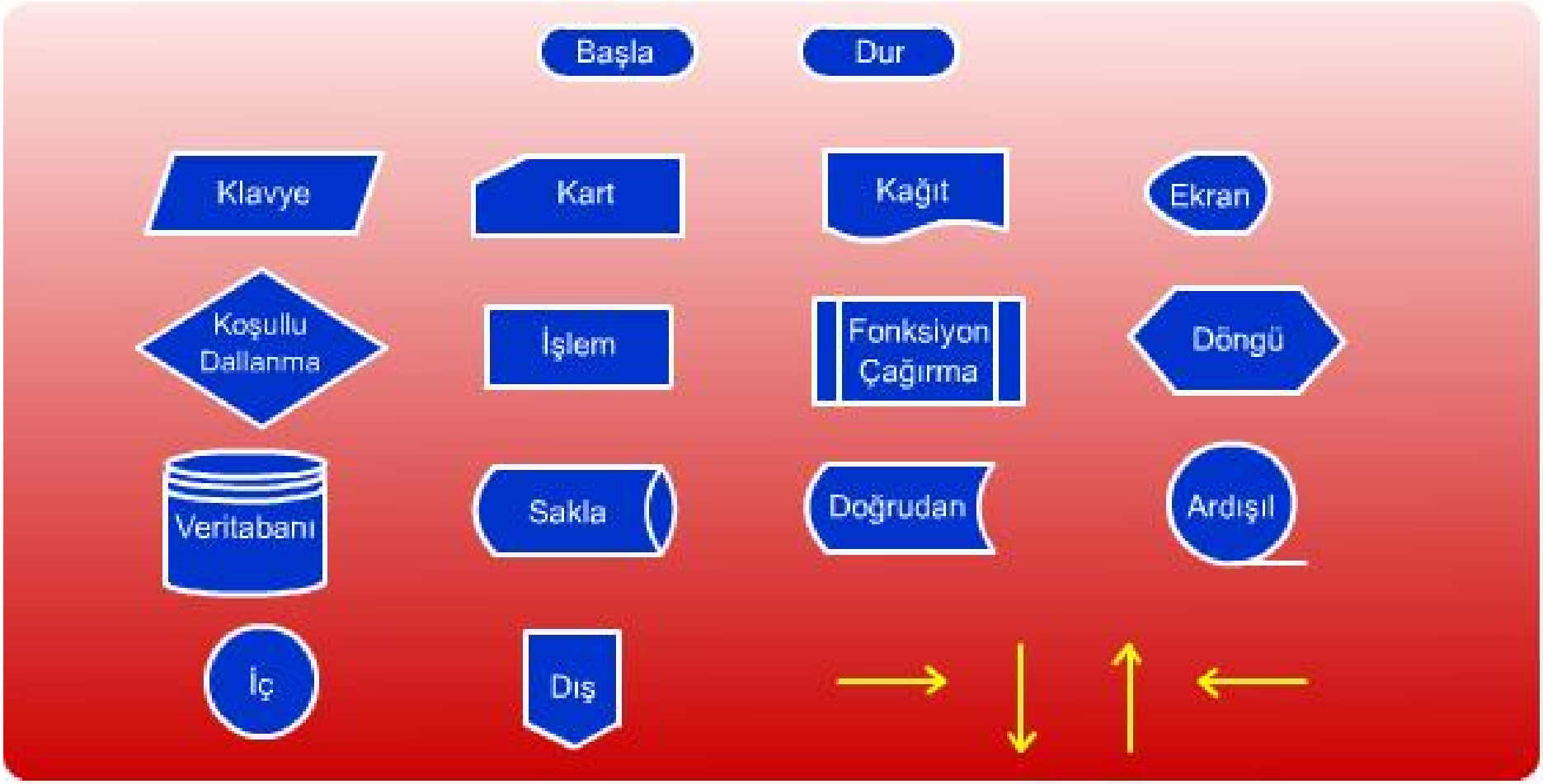
Akış şemaları, algoritmaların şematik olarak gösterilmesidir.

Ancak akış şemalarındaki ifadeler, daha çok kullanılacak programlama dilindeki ifadelere yakındır.

Akış şemalarında, farklı şekiller, farklı görevleri sembolize eder.

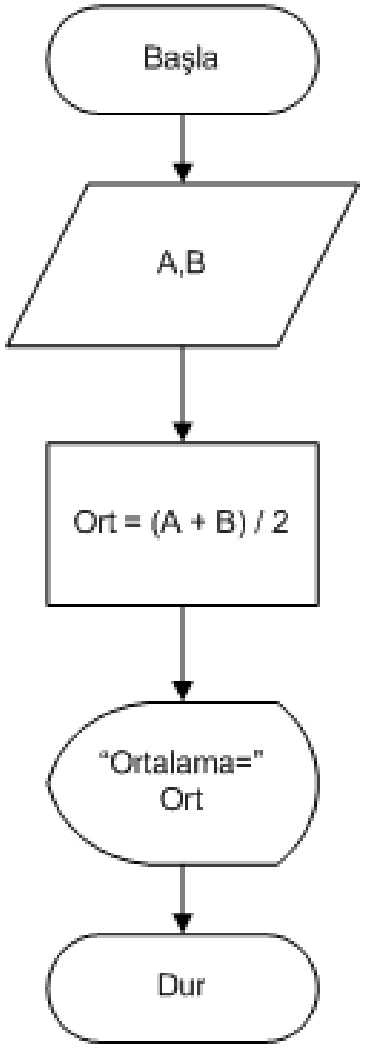
# Akış Şemaları

Akış şemaları, algoritmaların şematik olarak gösterilmesidir.



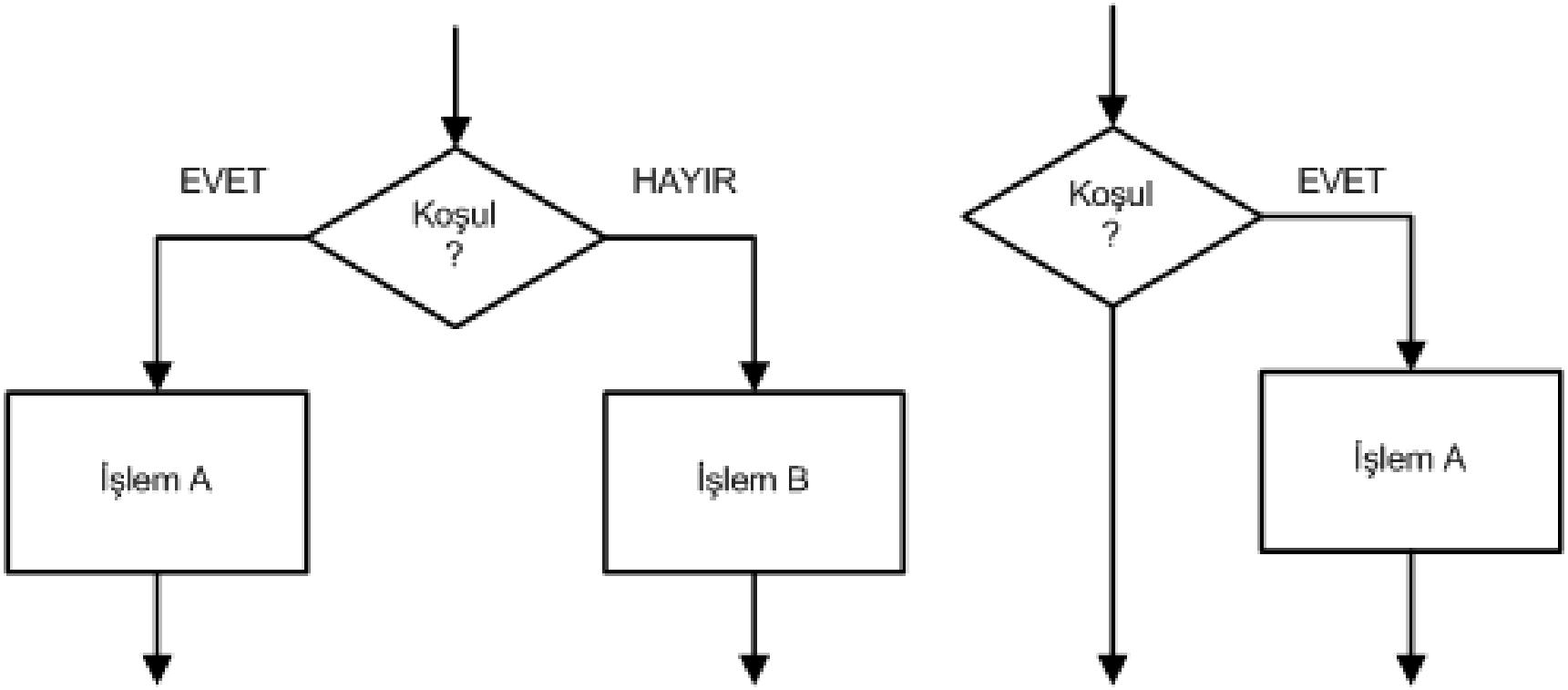
# Akış Şemaları

İki sayının aritmetik ortalamasını bulan akış şeması

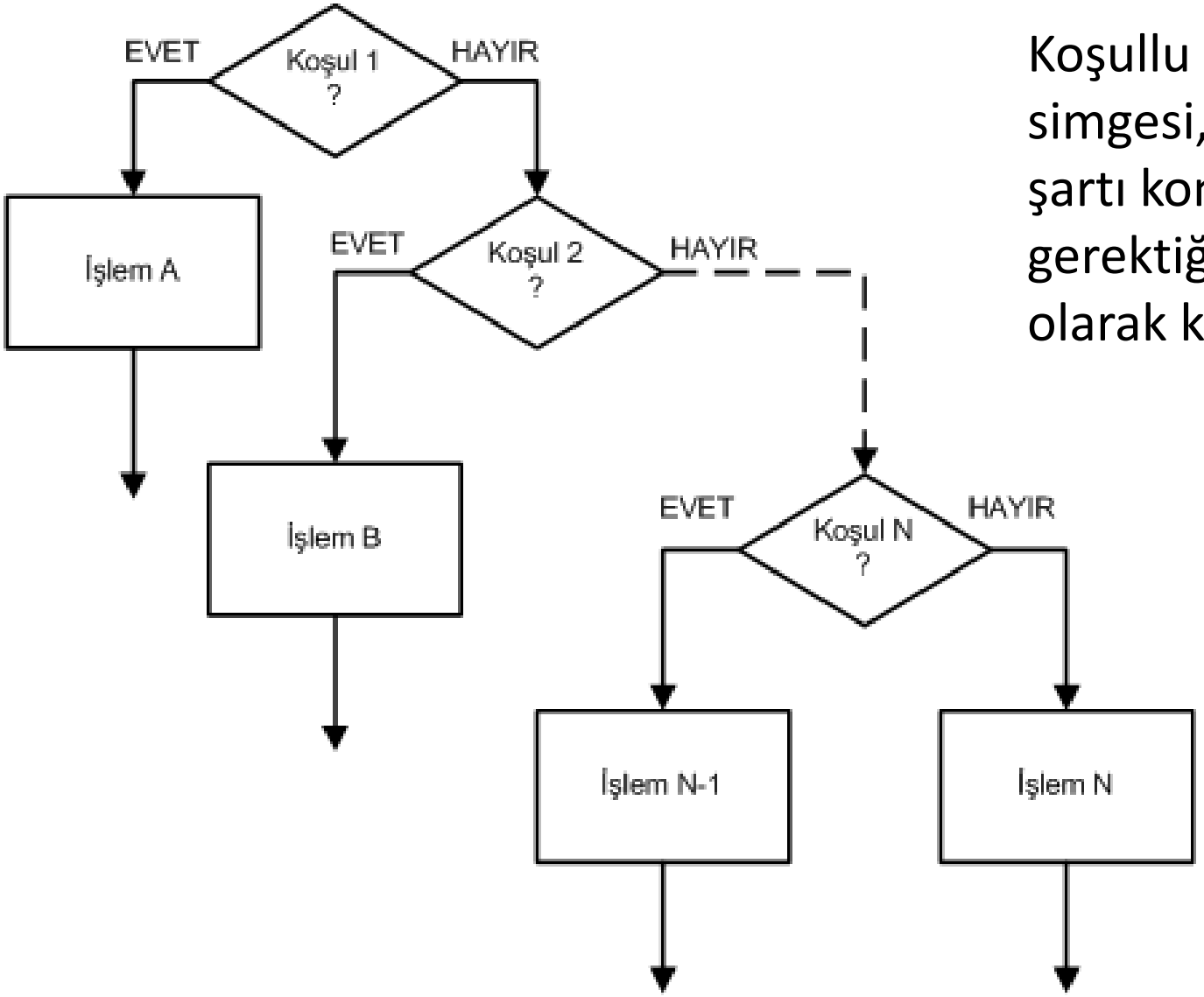


# Koşullu Dallanma Simgesi

Koşullu Dallanma simgesi baklava dilimi şeklindedir.  Bir girişi ve iki çıkışı vardır. Çıkışlardan biri doğru ya da evet, diğeri yanlış ya da hayır çıkışıdır.



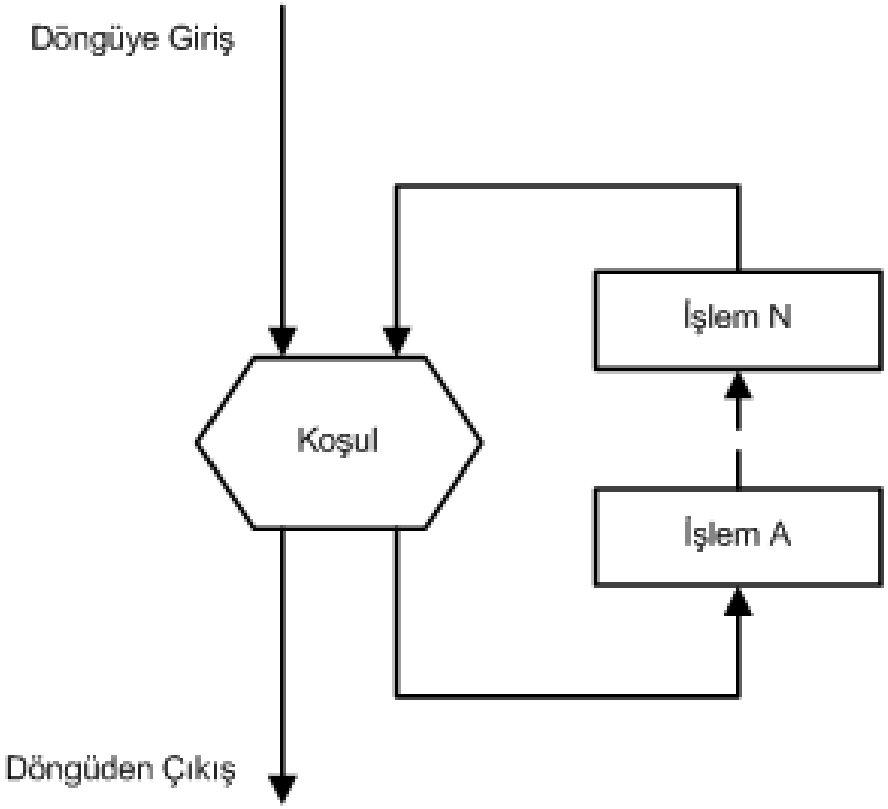
# Koşullu Dallanma Simgesi

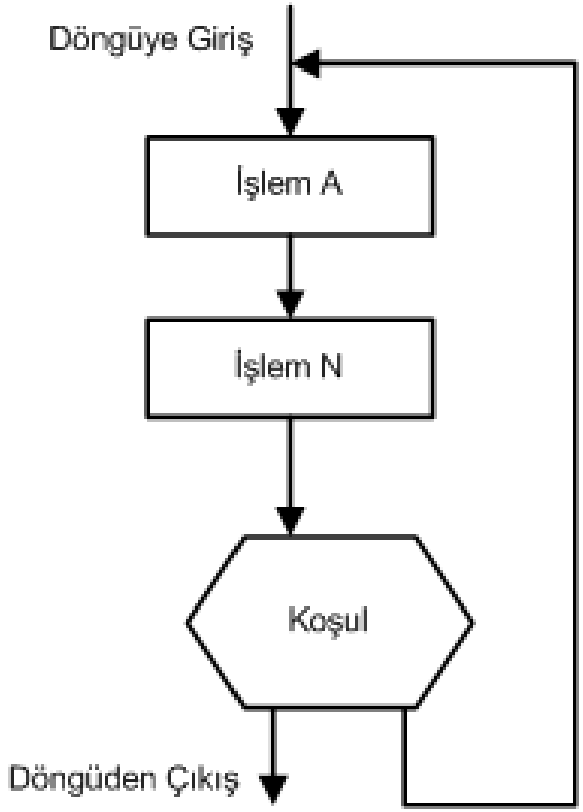



Koşullu dallanma simgesi, çok sayıda şartı kontrol etmek gerektiğinde ardaşık olarak kullanılabilir.

# Döngü Simgesi

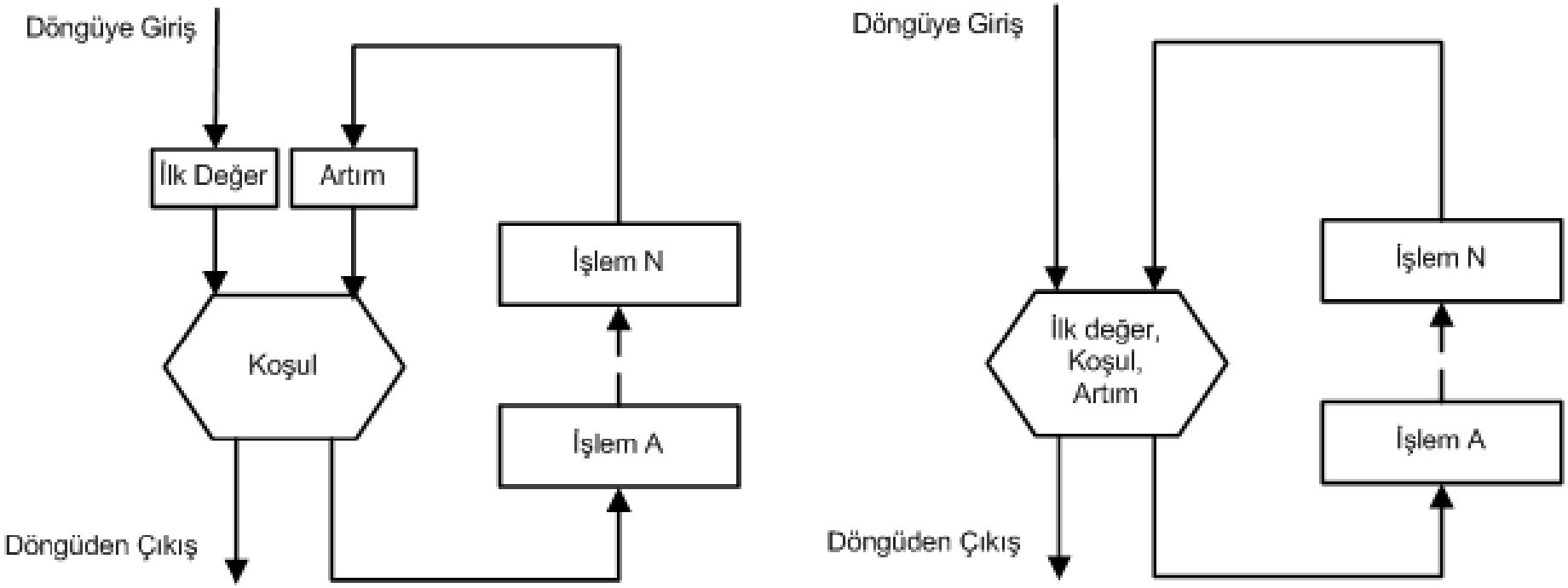
Döngü simgesi, basık altıgen şeklindedir. Göngüde belirtilen koşul doğru olduğu sürece döngü devam edecektir.



# Döngü Simgesi

Tipik bir for döngüsü ise aşağıdaki gibi gösterilebilir.

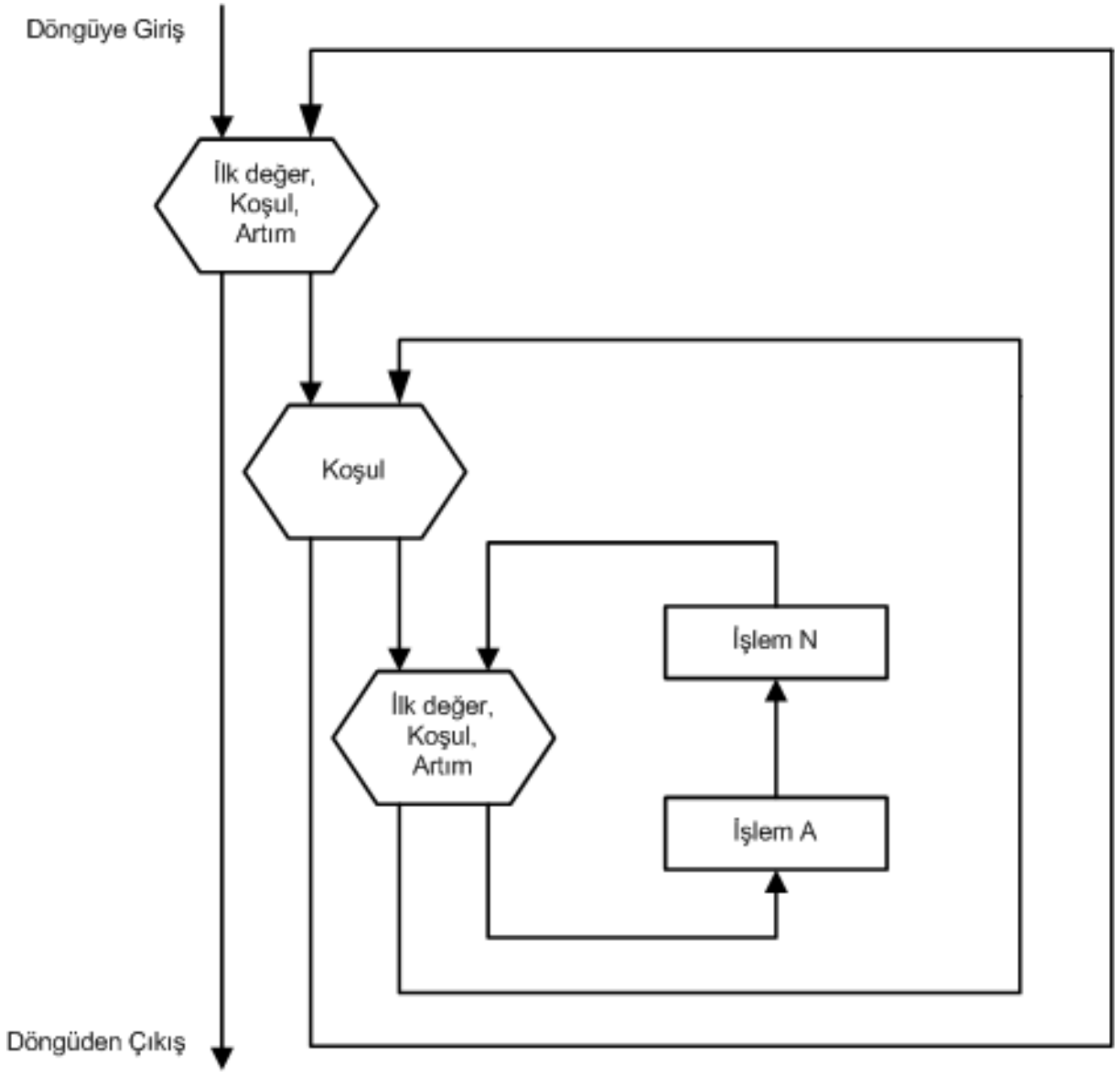



Ancak, ilk değer ve artım işlemlerini her defasında ayrıca göstermek yerine, bunları döngü simgesi içerisine yazmak daha pratik olacakır.

# İç İçe Döngüler

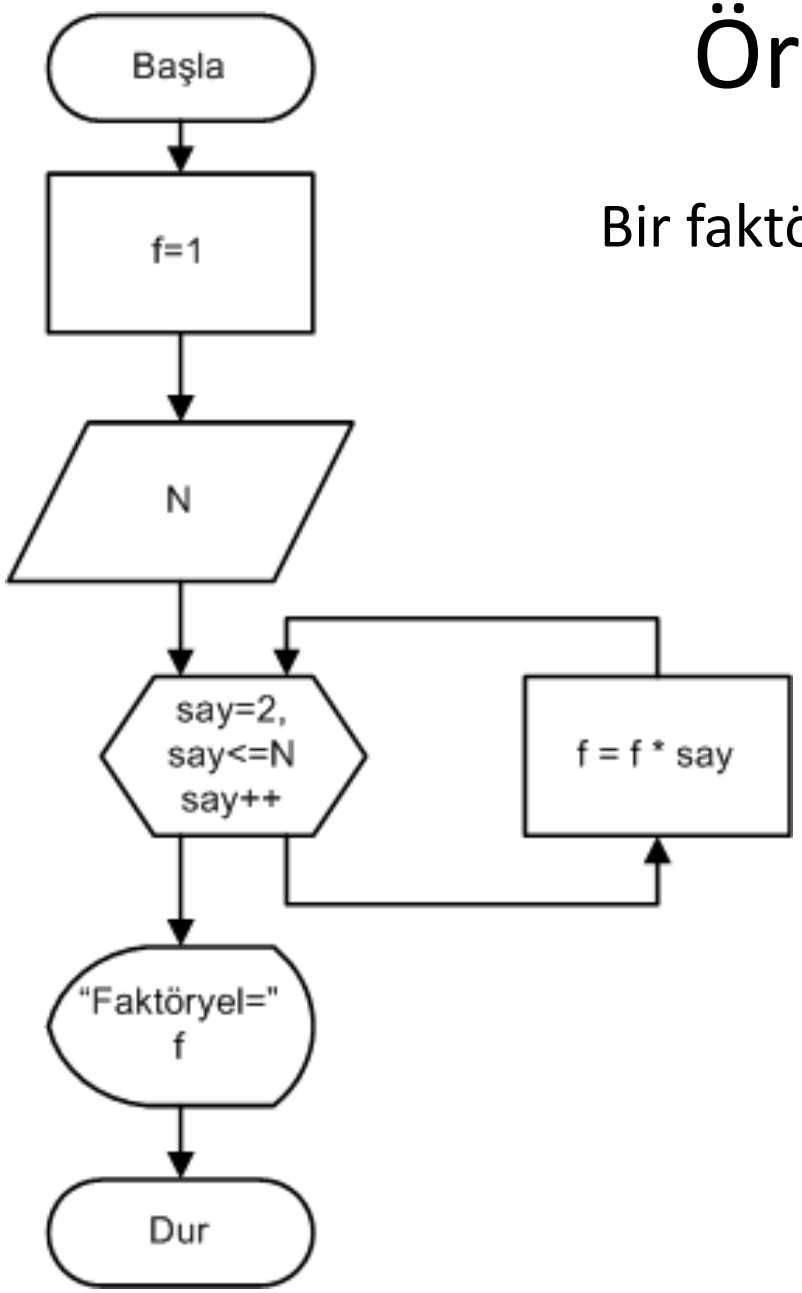
İç içe döngüler kurulabilir.

İç içe döngüler aynı tipte olabileceği gibi, farklı tipte döngülerden de oluşabilir.



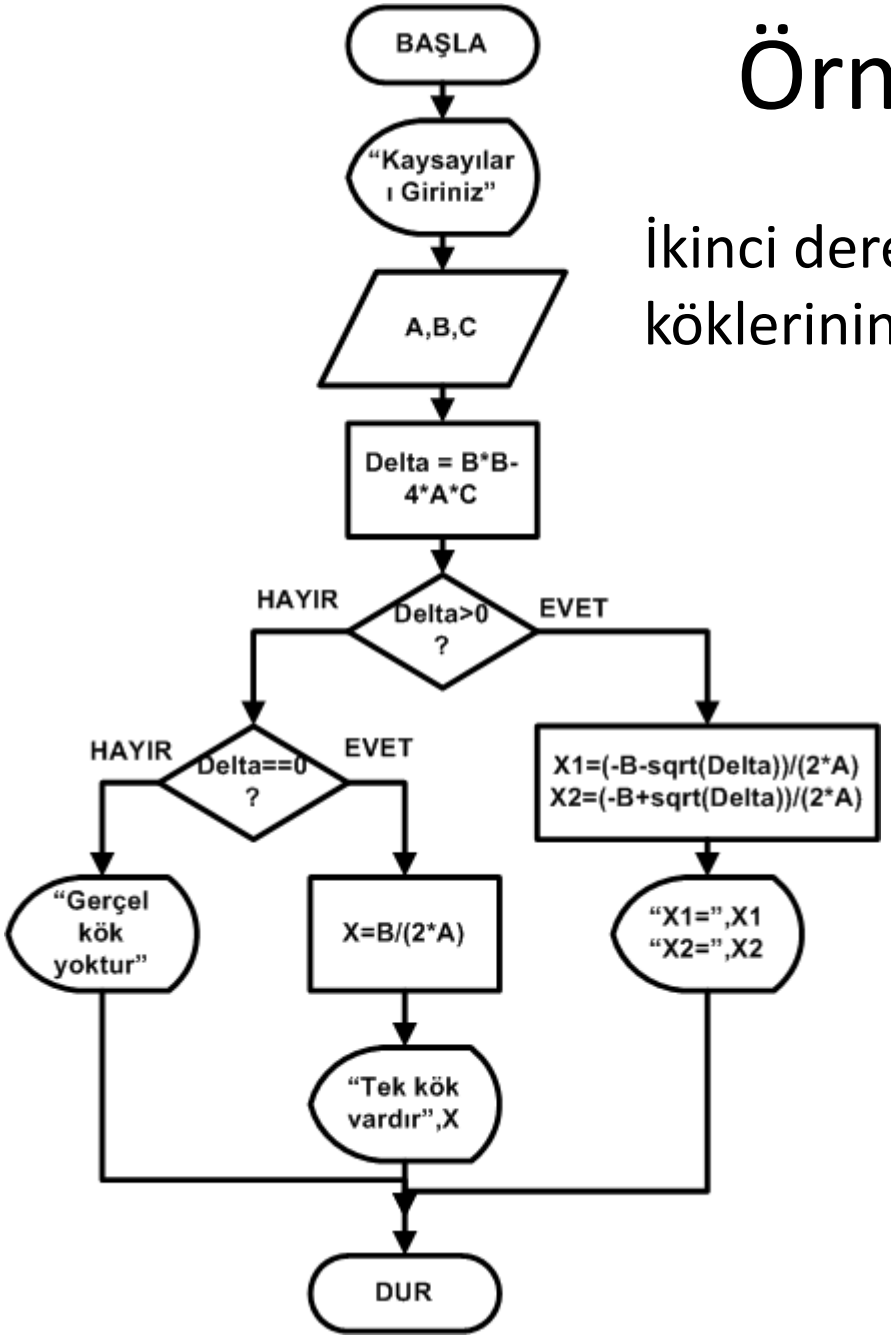
# Örnekler

Bir faktöryel hesabının akış şemasını çiziniz.



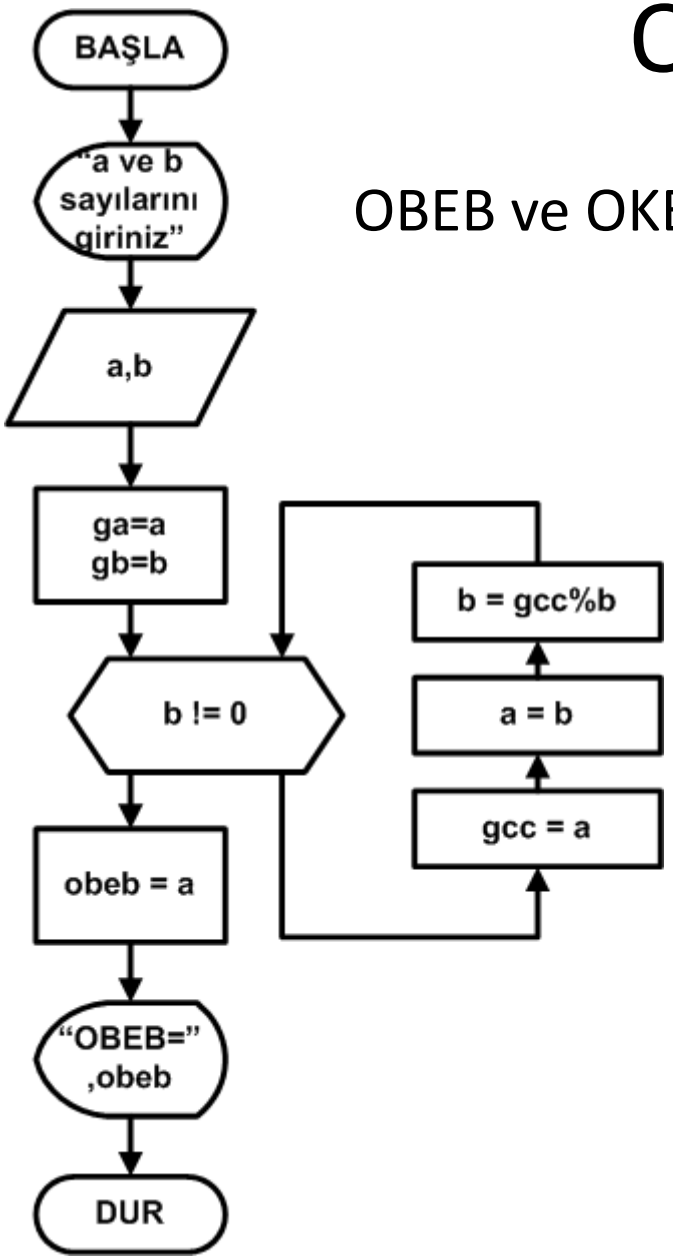
# Örnekler

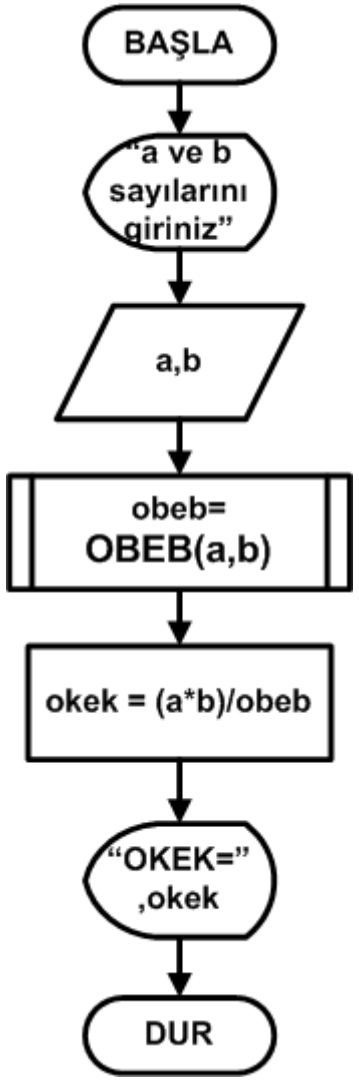
İkinci dereceden iki bilinmeyenli denklemin köklerinin hesaplanması



# Örnekler

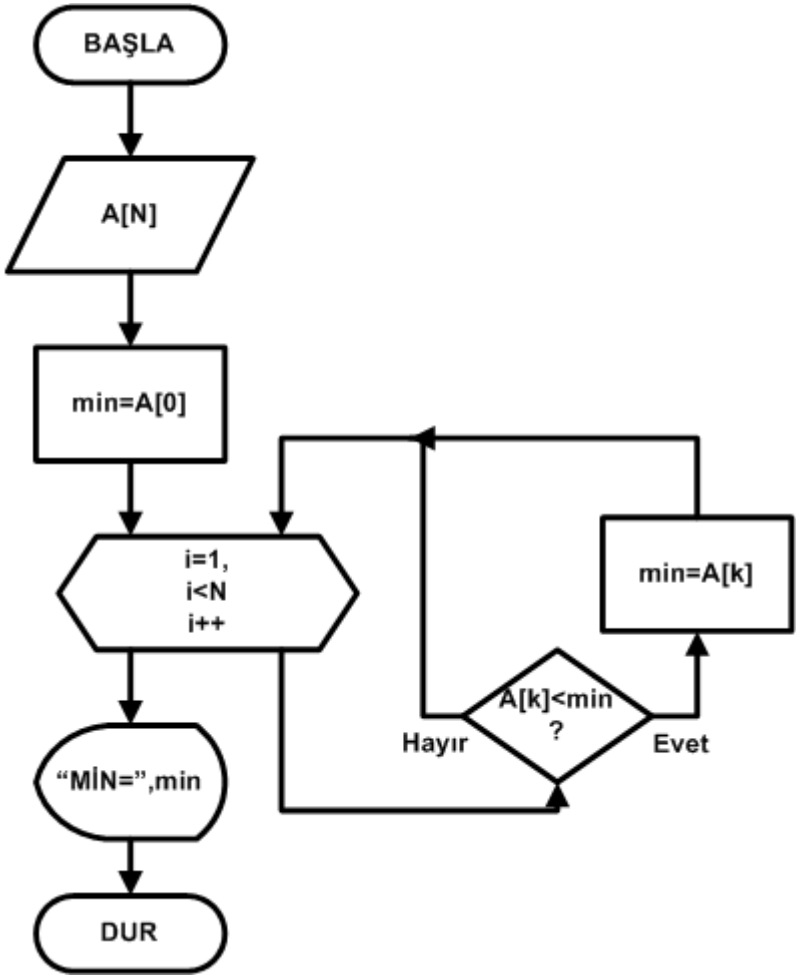
OBEB ve OKEK hesaplanması

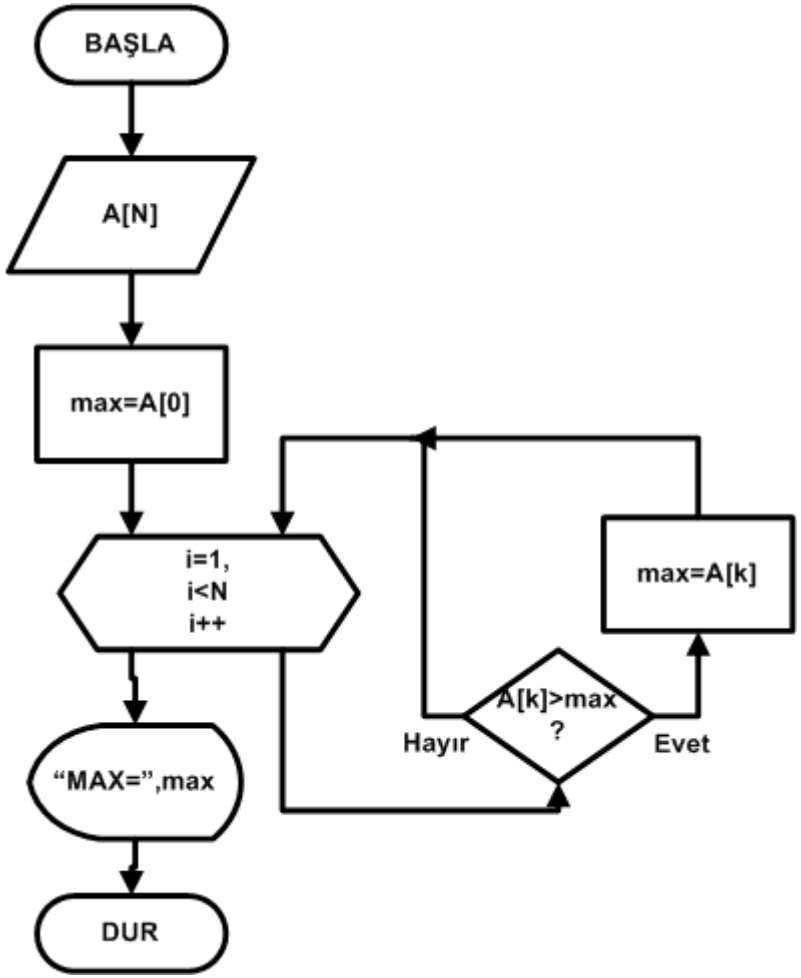


# Örnekler

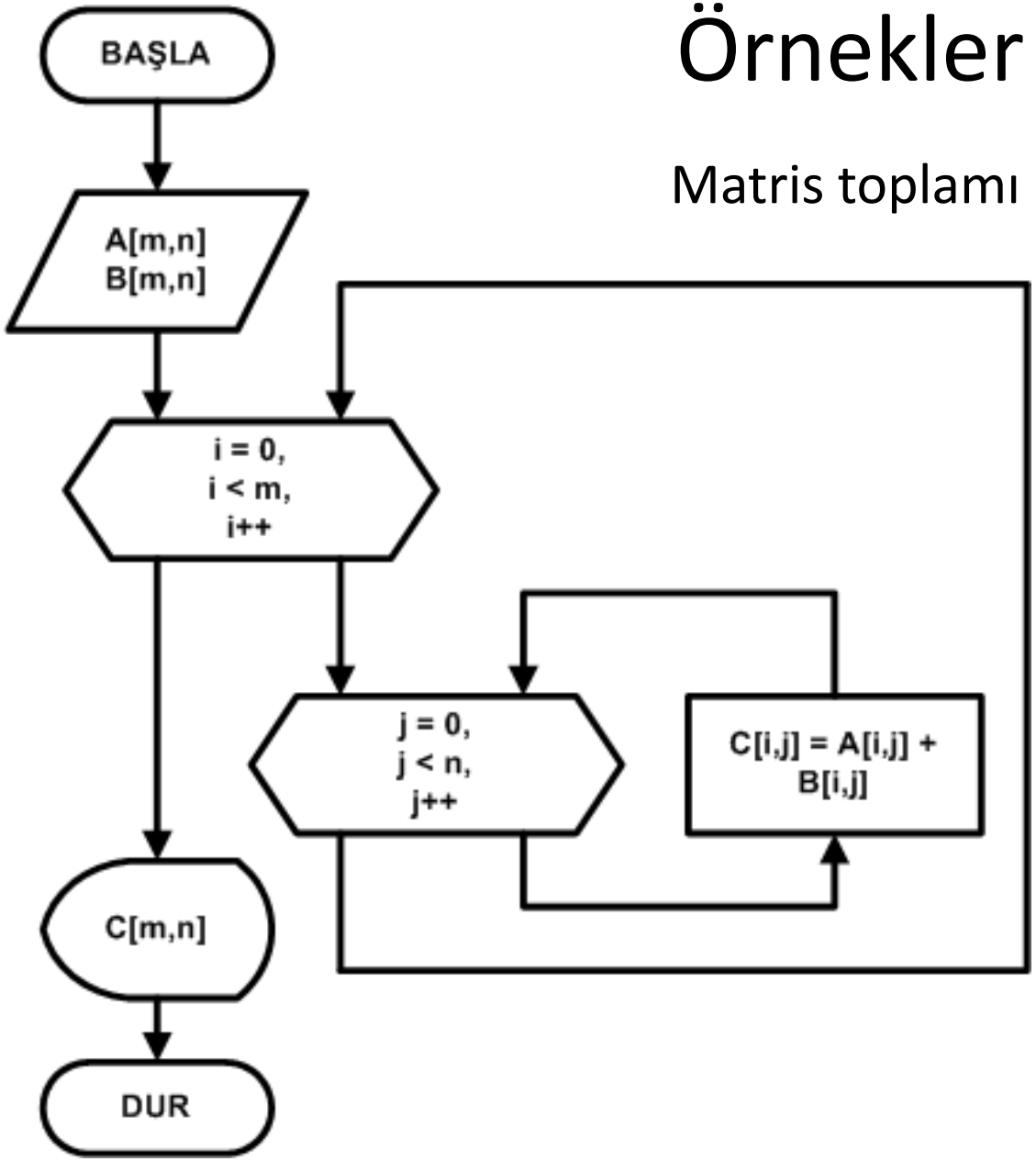
En küçük ve en büyük dizi elemanının bulunması



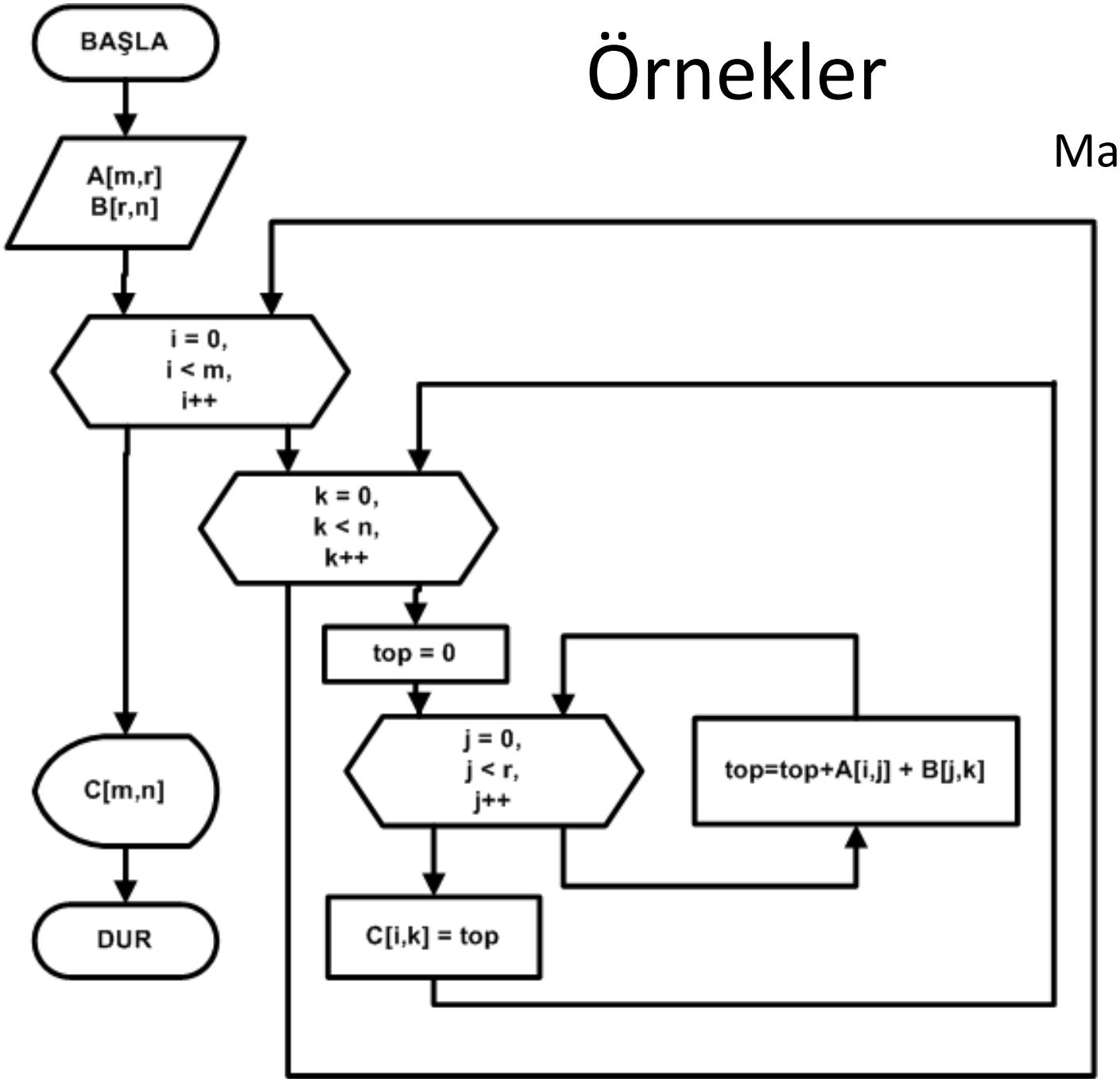
# Örnekler

Matris toplamı



# Örnekler

Matris çarpımı



# Yürütme Zamanı T(n)

Bir yazılım kodunda, toplam kaç birim işlem yapıldığının göstergesidir.

Buradaki her bir birim işlem, her bir dil (örn, C dili) ifadesi olarak ele alınabilir.

n olarak ifade edilen birim işlemlerin toplamı T(n) yürütme zamanı fonksiyonunu verecektir.

Ancak birçok algoritma veri işleridğinden dolayı, n'i belirleyen en büyük faktör veri sayısı olacaktır.

# Yürütme Zamanı T(n)

```
int ortalama(int dizi[], int len)
{
        int i,ort,top=0;

        for(i=0; i<len; i++)            1+(n+1)+n
                top += dizi[i];         n

        ort = top / len;                1

        return ort;                     1
}
```

**T(n) = 2n + 2 + n + 1 + 1 , T(n) = 3n + 4**

# Yürütme Zamanı T(n)

```
void matristopla (int A[m,n], int B[m,n], int C[m,n])
{
        int i,j;
        for(i=0; i<m; i++)                      1+(m+1)+m
                for(j=0; j<n; j++)              m*(1+(n+1)+n)
                        C[i,j] = A[i,j]+B[i,j]; m*n
}
```

**T(n) = 2m+2 + m*(2n+2) + mn**

**T(n) = 3mn + 4m + 2,**

**Eğer m == n olursa; T(n) = $3n^2 + 4n + 2$ olur.**

# Karmaşıklık (Complexity)

Karmaşıklık ifadesi, özellikle çok sayıda ya da büyüyen veri sayısı karşısında faklı algoritmaların nasıl davrandığını gösteren bir kavramsal ifadedir.

Karmaşıklığı ifade etmek için asimptotik ifadeler kullanılmaktadır. En çok kullanılan asimptotik karmaşılık ifadesi «büyük O» notasyonudur ve **O()** ile gösterilir.

O(n) fonksiyonuna verilen n değişkeni, algoritmanın işlediği veri sayısını göstermektedir.

# Karmaşıklık (Complexity)

Bir algoritmanın çalışma zamanı bulunduktan sonra, ne tür bir karmaşıklığa sahip algortma olduğunu çıkarsamak için;

T(n) bulunduktan sonra, tüm değişkenler tek tip değişkene dönüştürülür, sabitler ve düşük dereceli terimler atılar ve  en yüksek dereceli terim, O(n) karmaşıklık foksiyonunu ifade eder.

# Karmaşıklık (Complexity)

Örneğin,

Daha önce ortalama alan fonksiyonun bulunan T(n) yürütme zamanı,

**T(n) = 3n + 4**

Karmaşıklık notasyonu;

**O(n)**

Olur.

# Karmaşıklık (Complexity)

Örneğin, Daha önce matris toplayan fonksiyonun bulunan T(n) yürütme zamanı,

**$T(n) = 3mn + 4m + 2$**

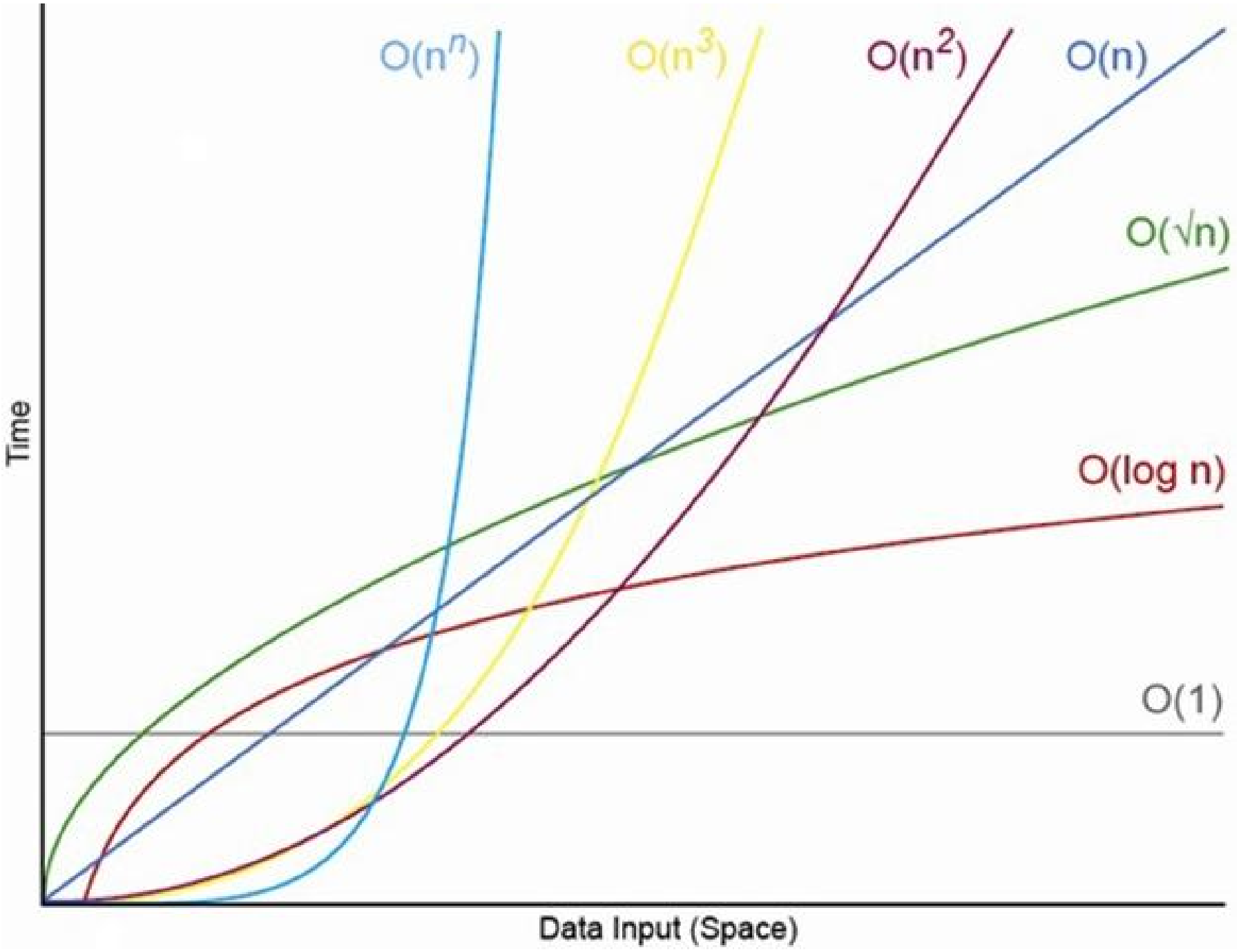
Tek tip değişkene dönüştürürsek,

**$T(n) = 3n^2 + 4n + 2$**

Buradan;

**$O(n^2)$**

Bulunur.


O(n^n)
O(n^3)
O(n^2)
O(n)
O(√n)
O(log n)
O(1)
Time
Data Input (Space)

# Örnek Karmaşılık ve Yürütme Zamanları

**1GHz işlemcili bir bilgisayarda, 1 saniyede 1.000.000.000 işlem (instruction) yürütüldüğü varsayıldığında, her bir veriyi işlemek için $10^{-9}$ saniye zaman harcanır. Buna göre, çeşitli karmaşıklığa sahip algoritmalar karşılaştırıldığında;**

| O() | n | | | |
|---|---|---|---|---|
| | 10.000 | 100.000 | 1.000.000 | 250.000.000 |
| O(log2n) | 13 ns | 17 ns | 20 ns | 28 ns |
| O(n) | 0.00001 s | 0.0001 s | 0.001 s | 0.25 s |
| $O(n\log_2 n)$ | 0.00013 s | 0.00166 s | 0.01993 s | 6.97 s |
| $O(N^2)$ | 0.1 s | 10.0 s | 16 d 40 s | ~1.98 yıl |
| $O(N^3)$ | 16 d 40 s | ~11 gün 14 sa | ~32 yıl | ~500.000.000 yıl |
| $O(2^n)$ | $\sim 6*10^{2993}$ yıl | ? | ? | ? |

# Çalışma Zamanını Ölçülmesi

Bir algoritmanın çalışma zamanı teorik olarak bulunabileceği gibi,  ölçülerek te bulunabilir.

Bunun için basitçe, algoritma başlamadan önce sistem saati okunur, algoritma çalıştırılır ve algoritma sonlandığında sistem saati tekrar okunur.

İki zaman arasındaki fark, algoritmamızın, o bilgisayardaki çalışma zamanını ölçer.

Ancak bu zaman, sadece o bilgisayar için geçerli olup, bire bir gerçek algoritma zamanını vermez. Çünkü işlemci aynı zamanda çok görevli (multitasking) işletim sisteminde diğer arkaplan proseslerini de çalıştırmaktadır.

# Çalışma Zamanını Ölçülmesi

Ancak, farklı algoritmaları birbirleriyle kıyaslamak (**benchmark**ing) için, aynı bilgisayarda çalıştırmak koşulu ile bu çalışma zamanı kullanılabilir. Basitçe;

```
#include <time.h>
time_t begin, end;
...
time(&begin);
suresi_olculecek_algoritma();
time(&end);

printf(«%.3f saniye\n», difftime(end,begin));
```

# Çalışma Zamanını Ölçülmesi

difftime ile zamanı ölçmenin bir dezavantajı, zaman çözünürlüğünün saniye bazında olmasıdır. Saniyeden daha küçük zaman dilimlerini ölçebilmek için clock() fonksiyonu kullanılabilir. clock() fonksiyonu, geçen süreyi işlemci saat darbe sayısı olarak ölçer. CLOCKS_PER_SEC ile bölünerek saniye cinsinden süre elde edilir.

```
#include <time.h>
clock_t sure;
...
sure=clock();
suresi_olculecek_algoritma();
sure=clock()-sure;

printf(«%.3f saniye\n», (double)sure/CLOCKS_PER_SEC);
```