BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ · 1. DÖNEM

# Bilgisayar Mühendisliğine Giriş

Ders notları ve alıştırmalar


0 Bilgisayarın yapısı

1 Sayı sistemleri

2 Verinin saklanması

3 Algoritmaya giriş

4 Girdi-çıktı ve koşullar

5 Döngüler

6 Temel veri tipleri

7 Diziler ve karakter dizileri

8 Arama

9 Sıralama ve karmaşıklık

10 İşletim sistemleri

11 Bilgisayar ağları


HEKZON

hekzon.com/dersler

12 bölüm · 37 konu

# İçindekiler

Bu notlar, YouTube'daki Bilgisayar Mühendisliğine Giriş dersinin yazılı hâlidir. Her konunun yanındaki zaman, uzun videoda o konunun başladığı yeri gösterir. Her bölümün sonunda sorular var; cevaplarını ve çözümlerini en sonda bulabilirsin. Aynı soruları hekzon.com/dersler adresinde, cevabını hemen göreceğin alıştırmalar olarak da çözebilirsin.


0 **Bilgisayarın yapısı** 3
Kasada ne olur? · Dokununca ne olur?

1 **Sayı sistemleri** 10
Turuncu neden FF8000? · On her zaman on mu? · Onda bir neden bitmez? · İkilikte hesap · 1 TB neden 931 GB?

2 **Verinin saklanması** 23
255 + 1 = 0? · −5 nasıl saklanır? · 0,1 + 0,2 = ? · Bir bayt, sekiz ayar · Her şey sayı

3 **Algoritmaya giriş** 37
Algoritma nedir? · Bu çizimi oku · 100 faktöriyel

4 **Girdi-çıktı ve koşullar** 46
x = x + 1 · 135 dakika · İndirim var mı?

5 **Döngüler** 55
Kaç kez merhaba? · Şifre · 91 asal mı?

6 **Temel veri tipleri** 66
65 mi, A mı? · 7 / 2 neden 3?

7 **Diziler ve karakter dizileri** 73
Neden 0? · En büyük not? · Kelime nerede biter?

8 **Arama** 81
Var mı, yok mu? · Milyonda kaç adım?

9 **Sıralama ve karmaşıklık** 88
Nasıl sıraya dizer? · Kartları nasıl dizersin? · Bir milyon sayı · Hız neyle ölçülür?

10 **İşletim sistemleri** 99
Aynı anda mı? · Dosya nerede?

11 **Bilgisayar ağları** 106
Doğru telefonu bulmak · Bir fotoğrafın yolculuğu · Sayfa nereden gelir?

**Cevaplar ve çözümler** 115

BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ · 1. DÖNEM

# 0 Bilgisayarın yapısı

Bu bölümde bir bilgisayarın veriyi nasıl bilgiye çevirdiğini, hangi parçalardan kurulduğunu ve bir programın bu parçalar üzerinde nasıl çalıştığını göreceksin. İki örnek izleyeceğiz: market kasasında okutulan bir süt ve telefonda açılan bir adımsayar.

## Kasada ne olur?

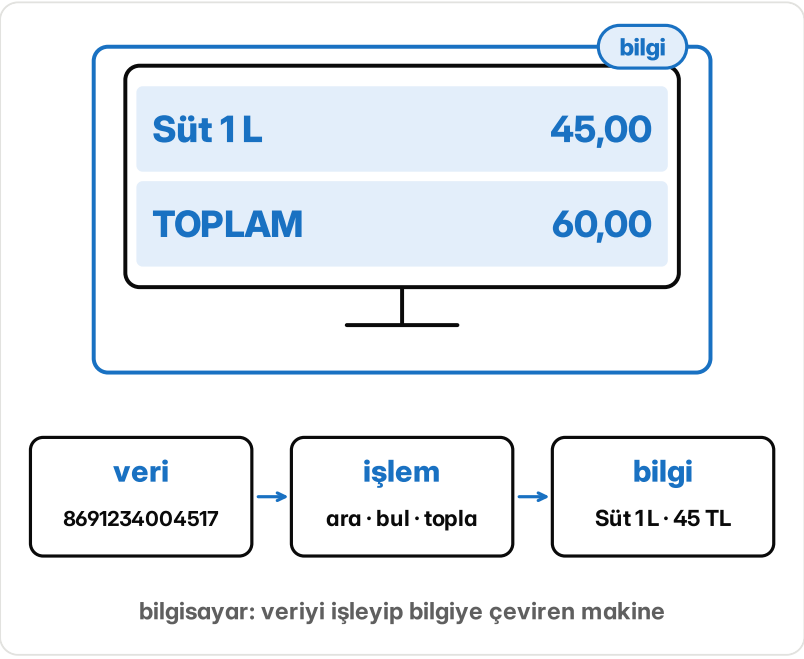
videoda 0:11

Market kasasında süt okutulduğunda ekrana ürünün adı ve fiyatı bir anda gelir. Kasadaki bilgisayar bu işi üç adımda yapar.

İlk adımda okuyucu, barkoddaki çizgileri bir numaraya çevirir: 8691234004517. Bu numara tek başına hiçbir şey anlatmaz; hangi ürün olduğunu da kaç lira tuttuğunu da söylemez. Anlamı henüz çıkarılmamış böyle sayı ve harflere **veri** denir.

İkinci adımda bilgisayar numarayı fiyat listesinde arar. Listede her barkodun karşısında bir ürün adı ve fiyatı yazılıdır: ekmek 15 TL, 10'lu yumurta 95 TL, 1 litre süt 45 TL, peynir 180 TL. Bilgisayar satırları tek tek karşılaştırır, numarayı süt satırında bulur ve 45 lirayı sepetin toplamına ekler. Sepetin toplamı 15 lira idiyse yeni toplam 15 + 45 = 60 lira olur. Veriyle yapılan bu arama ve hesaplara **işlem** denir.

Üçüncü adımda ekrana anlamlı bir sonuç çıkar: "Süt 1 L 45,00" ve "TOPLAM 60,00". Ad, fiyat ve toplamdan oluşan bu anlamlı sonuca **bilgi** denir; kasiyer de müşteri de ondan ne olduğunu anlar. Buradan bilgisayarın kısa bir tanımı çıkar: bilgisayar, veriyi işleyip bilgiye çeviren makinedir.



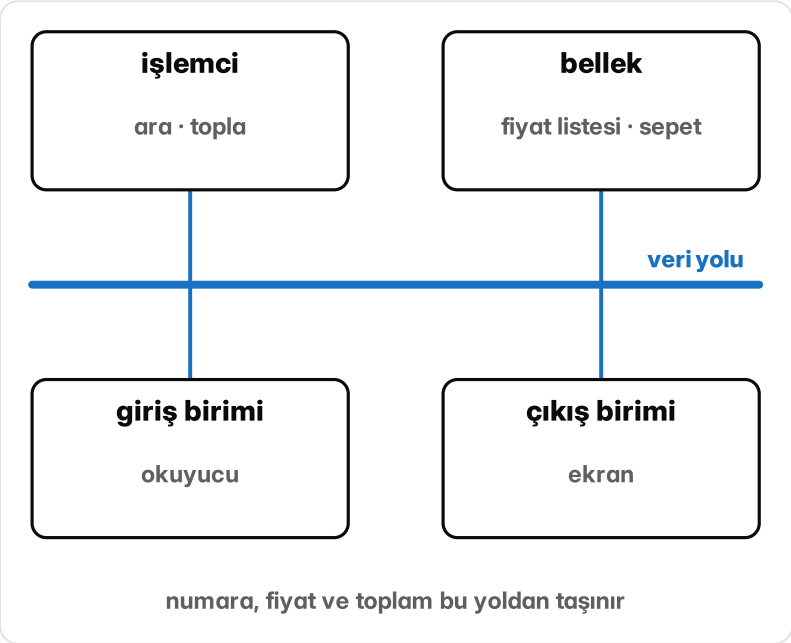

**Şekil 0.1** Kasa ekranındaki sonuç ve altında veri, işlem, bilgi adımları: barkod numarası aranıp toplanarak "Süt 1 L · 45 TL" bilgisine dönüşüyor.

Bu işin iki tarafı vardır. Okuyucu, ekran ve kasanın içindeki çipler elle dokunabildiğin parçalardır; bunlara **donanım** denir. "Numarayı listede ara, fiyatı topla, ekrana yaz" gibi, donanıma ne yapacağını söyleyen komutlara ise **yazılım** denir.

Kasada da telefonda da bilgisayarların neredeyse hepsi aynı düzenle kurulur. Bu düzen, 1945'te onu bir raporda tarif eden matematikçi John von Neumann'ın adıyla **Von Neumann modeli** diye anılır. Modelde dört parça vardır:

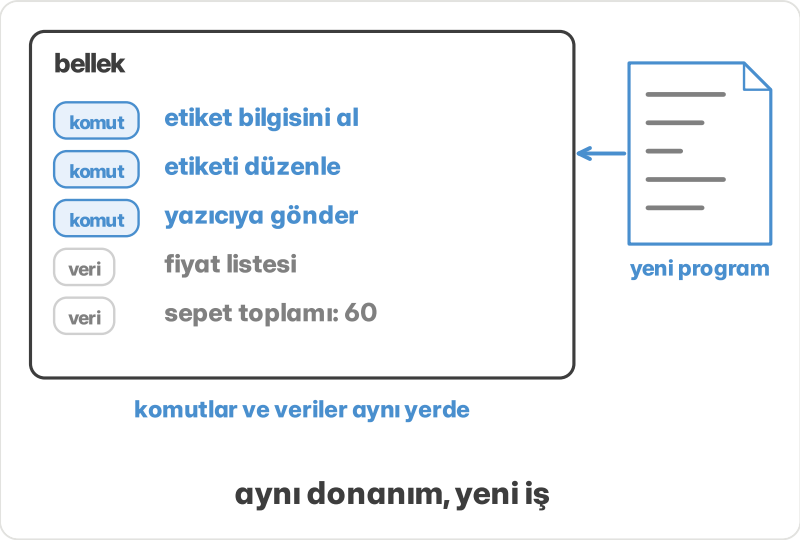
- **Giriş birimi** veriyi içeri alır; kasada bu, barkod okuyucudur.
- **Çıkış birimi** sonucu dışarı verir; kasada bu, ekrandır.
- **Bellek** fiyat listesini ve sepeti tutar.
- **İşlemci** aramayı ve toplamayı yapar.

Bu dört parça, **veri yolu** denen ortak bir hatla birbirine bağlıdır. Okunan numara, bulunan fiyat ve yeni toplam bu yoldan bir parçadan ötekine taşınır.




**Şekil 0.2** Von Neumann modeli: işlemci ve bellek üstte, giriş ve çıkış birimleri altta, hepsi ortadaki veri yoluna bağlı.

Modelin asıl fikri şudur: yazılımın komutları da verilerle aynı bellekte durur. Bellekte fiyat listesinin ve sepet toplamının yanında, bu verilerle ne yapılacağını söyleyen komutlar da saklanır. Bu yüzden aynı donanıma yeni bir iş yaptırmak için parçaları değiştirmek gerekmez; belleğe yeni bir program yüklemek yeter. Örneğin belleğe "etiket bilgisini al, etiketi düzenle, yazıcıya gönder" komutlarından oluşan bir program yüklenirse aynı donanım etiket basmaya başlar.




**Şekil 0.3** Bellekte komut ve veri satırları yan yana; yeni program yüklenince komutlar etiket basma işine dönüşüyor, veriler yerinde kalıyor.

Kasadaki her bipte bu dört parça birlikte çalışır: veri girer, işlenir, bilgi olarak çıkar.

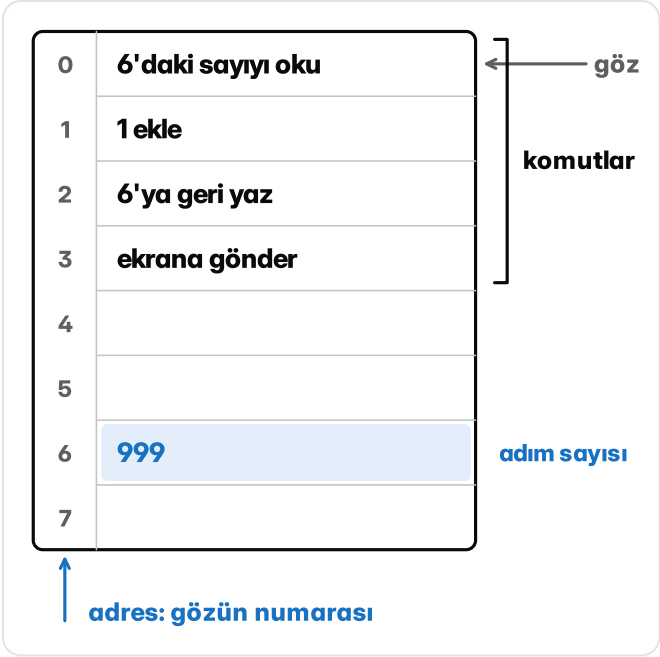
## Dokununca ne olur?

videoda 1:37

Telefonda bir uygulamaya dokunduğunda uygulama bir anda açılır. O anda içeride neler olduğunu bir adımsayar üzerinden izleyelim.

Uygulama, depolamada duran bir dosyadır. Dosyanın içinde işlemcinin anlayacağı komutlar birler ve sıfırlar hâlinde yazılıdır. Uygulamaya dokunduğun an bu komutlar depolamadan belleğe kopyalanır; buna **programı belleğe yüklemek** denir. Komutların önce belleğe taşınmasının nedeni şudur: işlemci komutları bellekten okur ve bellek, depolamadan çok daha hızlıdır.

Bellek, numaralı gözlerden oluşan bir raf gibi düşünülebilir. Her gözün numarasına **adres** denir. Adımsayarın dört komutu gözlere sırayla dizilir: 0. gözde "6'daki sayıyı oku", 1. gözde "1 ekle", 2. gözde "6'ya geri yaz", 3. gözde "ekrana gönder". O ana kadar atılan adım sayısı, 999, başka bir gözde, 6. gözde durur.

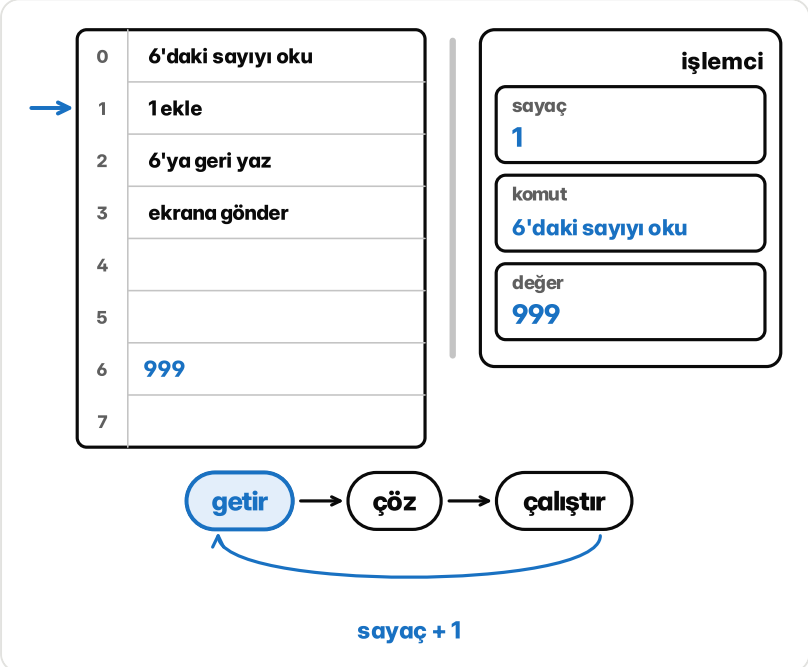



**Şekil 0.4** Sekiz gözlü bellek rafı: 0-3. gözlerde adımsayarın komutları, 6. gözde adım sayısı 999.

İşlemcinin içinde, sıradaki komutun adresini tutan bir sayaç vardır. Başta sayaç 0'ı, yani ilk komutun adresini gösterir. İşlemci her komut için aynı üç adımı izler:

- Getirir: sayacın gösterdiği gözdeki komutu bellekten alır.
- Çözer: komutun ne istediğini anlar; ilk komutta istenen, 6. gözdeki sayıyı okumaktır.
- Çalıştırır: komutu yerine getirir; burada 999'u bellekten içeri alır.

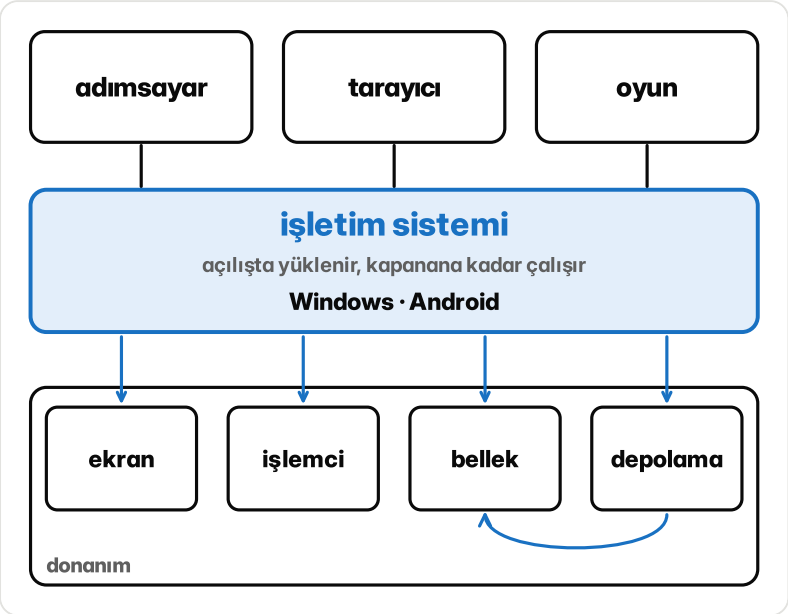
Ardından sayaç 1 artar ve döngü sıradaki komutla baştan başlar. İşlemcinin bu sürekli döngüsüne **getir, çöz, çalıştır** döngüsü denir.




**Şekil 0.5** İlk turun sonunda işlemci: sayaç 1'i gösteriyor, komut "6'daki sayıyı oku", değer 999; altta getir, çöz, çalıştır döngüsü.

Adımsayarın dört komutu dört turda biter. İlk turda 999 okunur. İkinci turda 1 eklenir ve değer 1.000 olur. Üçüncü turda 1.000, 6. göze geri yazılır. Dördüncü turda sayı ekrana gönderilir ve ekranda "1.000 adım" görünür. İşlemci bu döngüyü saniyede milyarlarca kez döner; bu yüzden dört tur sana bir an gibi gelir.

Peki dosyayı depolamadan belleğe kim yükledi? Bu işi **işletim sistemi** yapar. İşletim sistemi, bilgisayar açılırken yüklenen ve kapanana kadar çalışan yazılımdır. Donanımı yönetir; belleği ve ekranı uygulamalar arasında paylaştırır. Windows ve Android birer işletim sistemidir.




**Şekil 0.6** Üç katman: üstte uygulamalar, ortada işletim sistemi, altta ekran, işlemci, bellek ve depolamadan oluşan donanım.

**Uygulama** ise belli bir iş için açtığın programdır: adımsayar, tarayıcı, oyun. Uygulama donanıma doğrudan dokunmaz. Ekrana 1.000 sayısını yazmak için bile işletim sisteminden ister; ekrana yazma işini işletim sistemi yapar.

Bir uygulamaya her dokunuşunda bu olur: işletim sistemi onu belleğe yükler, işlemci de komutlarını tek tek getirir, çözer ve çalıştırır.

**Özet**

- Veri, anlamı henüz çıkarılmamış sayı ve harflerdir; işlem onu arar ve hesaplar; çıkan anlamlı sonuç bilgidir.
- Donanım elle dokunabildiğin parçalar, yazılım ise donanıma ne yapacağını söyleyen komutlardır.
- Von Neumann modelinde işlemci, bellek, giriş ve çıkış birimleri ortak bir veri yoluyla bağlıdır.
- Komutlar verilerle aynı bellekte durur; aynı donanıma yeni bir iş için yeni bir program yüklemek yeter.
- İşlemci her komutu getirir, çözer ve çalıştırır; sayaç sıradaki komutun adresini tutar.
- İşletim sistemi programları belleğe yükler ve donanımı yönetir; uygulama donanımı ondan ister.

## Sorular

**1** **Kasa ekranında görünen "Süt 1 L 45,00" satırı neye örnektir?**

A Veri
B Donanım
C Bilgi
D Veri yolu

**2** **Aşağıdakilerden hangisi yazılımdır?**

A Barkod okuyucu
B Kasa ekranı
C Kasanın içindeki çip
D "Numarayı listede ara, fiyatı topla, ekrana yaz" komutları

**3** **Von Neumann modelinde kasadaki fiyat listesi ve sepet hangi birimde tutulur?**

A Bellek
B İşlemci
C Giriş birimi
D Çıkış birimi

**4** **Von Neumann modelinin asıl fikri hangisidir?**

A İşlemci ile bellek tek bir parçada birleşir.
B Programın komutları da verilerle aynı bellekte durur.
C Her yeni iş için yeni bir donanım kurulur.
D Giriş birimi ile çıkış birimi aynı cihazdır.

**5** **Bir uygulamaya dokunduğunda komutları neden depolamadan belleğe kopyalanır?**

**A** Depolama bellekten daha hızlı olduğu için.

**B** Komutlar ancak bellekte birler ve sıfırlara çevrilebildiği için.

**C** İşlemci komutları bellekten okuduğu ve bellek depolamadan çok daha hızlı olduğu için.

**D** İşletim sistemi depolamada yer açmak istediği için.

**6** **Adımsayarın dört komutu 0-3. gözlerde duruyor ve sayaç başta 0'ı gösteriyor. Dört komut da getirilip çalıştırıldıktan sonra sayaç hangi sayıyı gösterir?**

Cevap: _____

**7** **Adımsayar ekrana 1.000 sayısını yazmak istediğinde ne olur?**

**A** Ekranı doğrudan kendisi kullanır.

**B** Bunu işletim sisteminden ister; ekrana yazma işini işletim sistemi yapar.

**C** Sayıyı depolamaya yazar, ekran onu oradan okur.

**D** Belleğe yeni bir işletim sistemi yükler.

**8** **Adımsayarın dört komutu (6'daki sayıyı oku, 1 ekle, 6'ya geri yaz, ekrana gönder) çalışınca 6. gözdeki 999, 1.000 oldu. Bir adım daha atılınca aynı dört komut yeniden çalışıyor. Bu kez ekranda hangi sayı görünür?**

Cevap: _____

BİLGİSAYAR MÜHENDİSLİĞİNE GİRİŞ · 1. DÖNEM

# 1 Sayı sistemleri

Bu bölümde bilgisayarın sayıları nasıl yazdığını göreceksin: onluk, ikilik, sekizlik ve onaltılık sistemler, bir tabandan ötekine çevirme, kesirli sayılar, ikilik sistemde toplama, çıkarma ve çarpma, son olarak da bit, bayt ve bellek birimleri.

## Turuncu neden FF8000?

videoda 3:08

Bilgisayarda turuncunun bir tonu FF8000 diye yazılır. İçinde harf var ama bu bir sayıdır. Nedenini anlamak için önce kendi sayılarımıza bakalım.

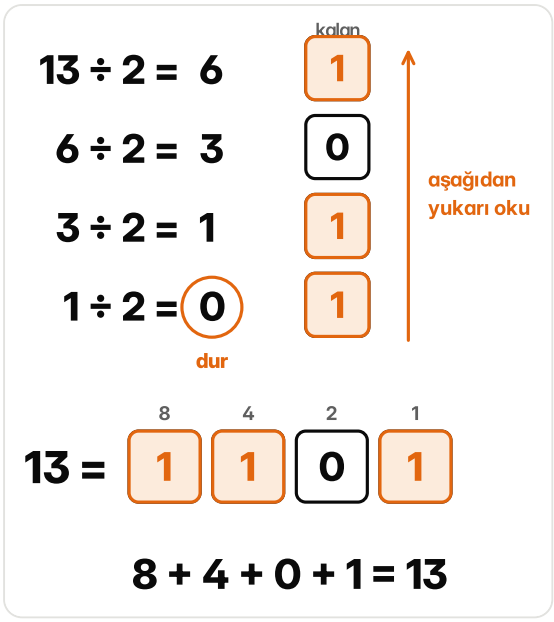
253 sayısını yazarken üç rakam kullanırız: 2, 5 ve 3. Her rakamın değeri durduğu yerden gelir: en sağdaki birler basamağıdır, solundaki onlar, onun solundaki yüzler. Yani 253 = 200 + 50 + 3. 10 rakamımız olduğu için her basamak sağındakinin 10 katıdır. Buna **onluk sistem** denir.

Bilgisayarın ise yalnızca iki rakamı vardır, 0 ve 1; çünkü içindeki her minik anahtar ya kapalıdır ya açık: akım yoksa 0, akım varsa 1. Bu yüzden onun her basamağı sağındakinin 2 katıdır: 1, 2, 4, 8, 16. Buna **ikilik sistem** denir.

Peki 13 ikilik sistemde nasıl yazılır? Önce onluk sistemdeki bir hileye bakalım. 253'ü 10'a böl: bölüm 25, kalan 3. Kalan, sayının en sağdaki rakamıdır. 25'i yine 10'a böl: bölüm 2, kalan 5; bu da sağdan ikinci rakamdır. Yani 10'a bölmek rakamları sağdan birer birer koparır. İkilik sistemde aynı işi 2'ye bölerek yaparsın; her bölmenin kalanı, ikilik sayının sağdan sıradaki rakamıdır.

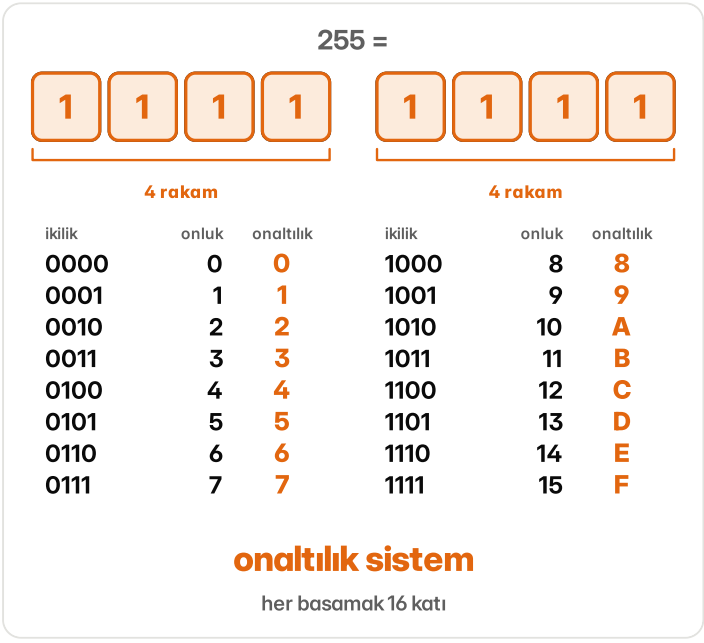
```
13 ÷ 2 = 6   kalan 1
 6 ÷ 2 = 3   kalan 0
 3 ÷ 2 = 1   kalan 1
 1 ÷ 2 = 0   kalan 1
```

Bölüm 0 olunca durursun. İlk kalan en sağdaki rakam olduğu için kalanları aşağıdan yukarı okursun: 1, 1, 0, 1. Yani 13, ikilikte 1101 olur. Sağlama: 8 + 4 + 0 + 1 = 13.




**Şekil 1.1** 13'ün ikiliğe çevrilmesi: art arda 2'ye bölme, aşağıdan yukarı okunan kalanlar 1101 ve sağlaması 8 + 4 + 0 + 1 = 13.

İkilik sayılar çabuk uzar. 255, ikilikte yan yana 8 tane 1'dir: 11111111. Okuması da yazması da zordur; programcılar bu yüzden ikilik rakamları dörder dörder gruplar. 4 ikilik rakamla, 0000'dan 1111'e, yani 0'dan 15'e kadar 16 farklı sayı yazılabilir. Her grubu tek bir rakamla yazmak için 16 rakam gerekir: 0'dan 9'a kadar bildiklerimiz, ardından sırayla 10'dan 15'e kadar değerler için A, B, C, D, E ve F. Buna **onaltılık sistem** denir; her basamak sağındakinin 16 katıdır.



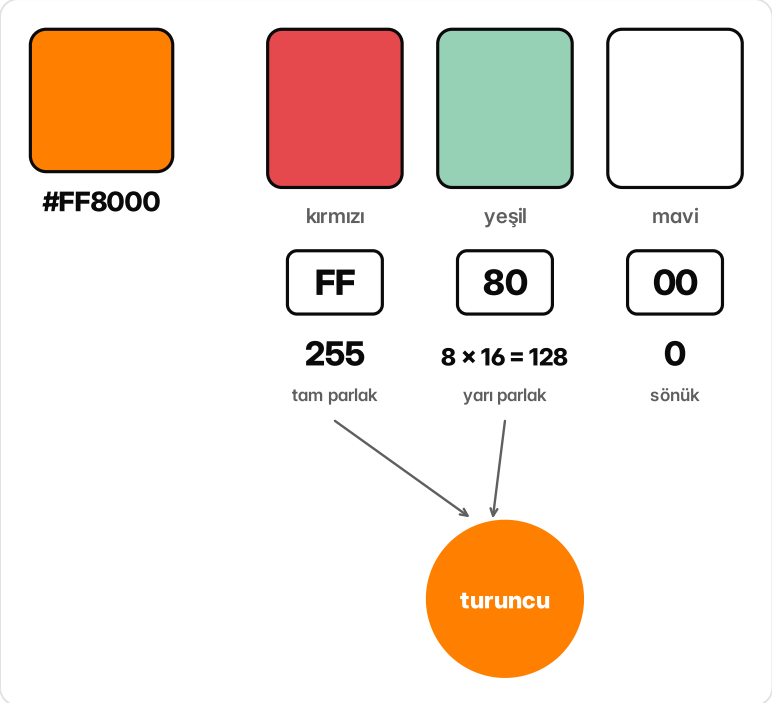

**Şekil 1.2** 255'in sekiz ikilik rakamı dörtlü iki gruba ayrılmış; altında 0000-1111 arasındaki 16 grubun onluk ve onaltılık karşılıkları.

11111111 iki gruba ayrılır: 1111 ve 1111. Her grup 8 + 4 + 2 + 1 = 15, yani F eder. Böylece 255, onaltılıkta FF olur: 8 ikilik rakam, 2 onaltılık rakama sığar.

Şimdi turuncuya dönelim. Ekrandaki her nokta üç küçük ışıkla yanar: kırmızı, yeşil ve mavi. Her ışığın parlaklığı 0 ile 255 arasında bir sayıdır; yani 8 ikilik rakam, onaltılıkta 2 rakam. Renk kodu bu üç sayının yan yana yazılmasıdır (web sayfalarında başına # konur: #FF8000). Kod ikişer rakamlık üç parçaya ayrılır:

- FF kırmızıdır: 255, tam parlak.
- 80 yeşildir: 8 × 16 + 0 = 128, yarı parlak.
- 00 mavidir: 0, sönük.

Tam parlak kırmızıyla yarı parlak yeşil birleşince göz turuncu görür.




**Şekil 1.3** FF8000 kodunun üç parçası: kırmızı FF = 255, yeşil 80 = 128, mavi 00 = 0; ışıklar karışınca turuncu.

Ekranındaki her renk böyle üç sayıdan oluşur. Bilgisayar bu sayıları birler ve sıfırlarla saklar (turuncu için 11111111, 10000000 ve 00000000), programcılar da onaltılıkla kısaca yazar.

## On her zaman on mu?

videoda 5:48

"10" yazısı her zaman on demek değildir. İkilikte 10, iki demektir; sekizlikte sekiz, onaltılıkta on altı. Rakamlar aynıdır; değeri, sayının yazıldığı **taban** belirler. Taban, sistemde kaç rakam olduğunu ve her basamağın sağındakinin kaç katı olduğunu söyler: onlukta 10, ikilikte 2, sekizlikte 8, onaltılıkta 16.

Her tabanda **basamak değerleri** aynı kuralla büyür: en sağdaki basamak 1'dir, solundaki tabanın kendisi, onun solundaki taban × taban.

- İkilikte: 1, 2, 4, 8.
- Sekizlikte: 1, 8, 64.
- Onaltılıkta: 1, 16, 256.

Bu kural, başka tabanda yazılmış bir sayıyı onluğa çevirmenin yolunu da verir: her rakamı kendi basamak değeriyle çarpar, sonuçları toplarsın. Üç örnek:

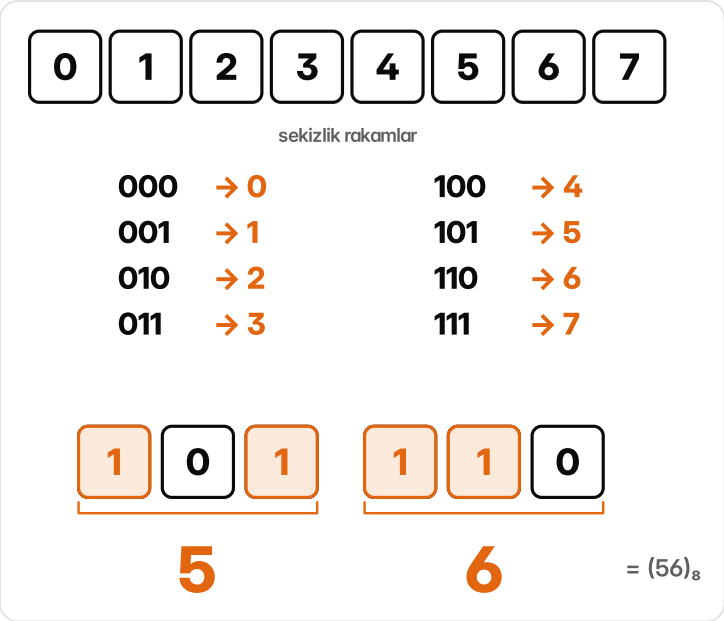
```
ikilik 1011    = 1 × 8 + 0 × 4 + 1 × 2 + 1 × 1 = 11
onaltılık 2F   = 2 × 16 + 15 × 1 = 32 + 15 = 47
sekizlik 17    = 1 × 8 + 7 × 1 = 15
```

Onaltılıkta F rakamının değeri 15 olduğu için ikinci örnekte F, 15 diye hesaba girer.




**Şekil 1.4** Onluğa çevirme: ikilik 1011 = 11, onaltılık 2F = 47, sekizlik 17 = 15; her rakamın üstünde basamak değeri.

**Sekizlik sistem** 0'dan 7'ye 8 rakam kullanır. 3 ikilik rakam da 000'dan 111'e, yani 0'dan 7'ye kadar 8 farklı değer alır. Bu yüzden ikilik bir sayı sağdan üçer üçer gruplanır ve her grup tek bir sekizlik rakam olur; onaltılıkta dörder dörder gruplamanın aynısıdır. Örneğin ikilik 101110 sayısı 101 ve 110 diye ikiye ayrılır: 101 = 5, 110 = 6. Yani ikilik 101110, sekizlikte 56'dır. Sağlama: 5 × 8 + 6 = 46 ve 32 + 8 + 4 + 2 = 46.




**Şekil 1.5** Üç ikilik rakamın sekiz sekizlik rakama karşılığı; ikilik 101110 üçlü gruplanarak sekizlik 56 oluyor.

Sunucu bilgisayarlarında dosya izinleri tam bu şekilde yazılır. Bir dosyanın sahibi, grubu ve geri kalan herkes için üç soru sorulur: okuyabilir mi, yazabilir mi, çalıştırabilir mi? Evet 1, hayır 0 diye yazılır. Sahibine üçü de açıksa 111, yani 7. Gruba ve herkese yalnız okuma ve çalıştırma açıksa 101, yani 5. İzin 755 diye yazılır: üç sekizlik rakam, dokuz evet-hayır cevabını taşır.




**Şekil 1.6** Dosya izni 755: sahibi için oku, yaz, çalıştır 111 = 7; grubu ve herkes için 101 = 5.

## Onda bir neden bitmez?

videoda 7:11

0,1 sayısı ikilik sistemde yazılmaya çalışılınca hiç bitmez. Nedenini görmek için önce bir kesrin ikilikte nasıl yazıldığına bakalım.

İkilikte virgülün solunda basamaklar ikişer kat büyür, sağında ikişer kat küçülür: yarım, çeyrek, sekizde bir, on altıda bir, yani 1/2, 1/4, 1/8, 1/16. Örneğin ikilik 0,11 sayısı yarım artı çeyrektir: 1/2 + 1/4 = 0,75.

Onluk bir kesri ikiliğe çevirmek için onu 2 ile çarparsın. Çünkü ikilikte 2 ile çarpmak virgülü bir basamak sağa kaydırır; virgülün önüne geçen rakam, sıradaki ikilik rakamdır. Tam kısmı rakam olarak yazar, kesirli kısımla devam edersin; kesirli kısım 0 olunca kesir biter. İlk çıkan rakam virgülden hemen sonraki rakam olduğu için rakamlar yukarıdan aşağı okunur.

```
0,75 × 2 = 1,5   rakam 1, kalan 0,5
0,5  × 2 = 1,0   rakam 1, kalan 0
```

Sonuç: 0,75, ikilikte 0,11 olur.

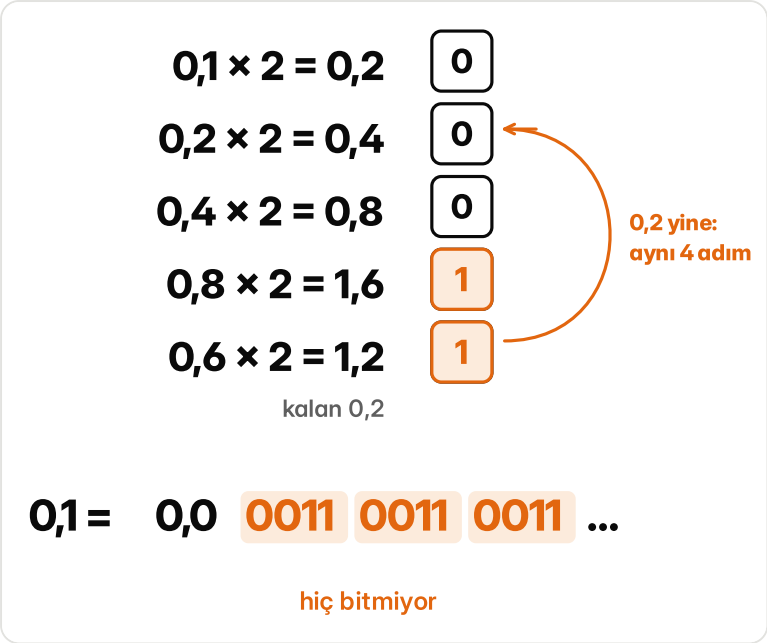



**Şekil 1.7** 0,75'in ikiliğe çevrilmesi: 2 ile çarpınca tam kısımlar sırayla 1 ve 1, sonuç ikilik 0,11.

Şimdi aynı yöntemi 0,1 için deneyelim:

```
0,1 × 2 = 0,2   rakam 0
0,2 × 2 = 0,4   rakam 0
0,4 × 2 = 0,8   rakam 0
0,8 × 2 = 1,6   rakam 1, kalan 0,6
0,6 × 2 = 1,2   rakam 1, kalan 0,2
```

Kalan 0,2'yi daha önce gördük. Buradan sonra aynı dört adım (0,2'den 0,4'e, 0,8'e, 0,6'ya ve yine 0,2'ye) sonsuza kadar tekrar eder ve her seferinde 0011 rakamlarını verir. 0,1, ikilikte 0,000110011... diye hiç bitmeden sürer.




**Şekil 1.8** 0,1'in 2 ile çarpılarak çevrilmesi: 0,2 yeniden çıkınca aynı dört adım dönüyor, sonuç 0,0 0011 0011 0011... diye sürüyor.

Neden? En sade hâliyle yazılmış bir kesir, ikilikte ancak paydası 2'nin bir kuvvetiyse biter: 1/2, 1/4, 1/8 ikilikte 0,1, 0,01 ve 0,001'dir. 0,1 ise 1/10'dur ve paydası 10 = 2 × 5'tir. 2 ile ne kadar çarparsan çarp paydadaki 5'ten kurtulamazsın; kesirli kısım hiçbir zaman 0'a inmez. Onlukta 1/3'ün 0,333... diye bitmemesi de aynı sebeptendir: 3, onluk sistemin tabanı 10'un çarpanı değildir.

Bilgisayar sonsuz rakamı saklayamaz; bellekte sınırlı sayıda kutu vardır ve rakamları bir yerde keser. Yani bellekteki 0,1, 0,1'in kendisi değil, ona çok yakın bir sayıdır. Örneğin bugün programların çoğunun kullandığı biçimde bellekteki değer 0,1000000000000000055511... olur.




**Şekil 1.9** Bellekteki sınırlı sayıda kutuya sığmayan rakamlar kesiliyor; saklanan değer 0,1000000000000000055511..., yani yaklaşık 0,1.

## İkilikte hesap

videoda 8:40

İşlemci toplamayı, çıkarmayı ve çarpmayı okulda öğrendiğin yöntemlerle yapar; tek farkı, yalnızca iki rakam kullanmasıdır.

Toplamada dört durum vardır:

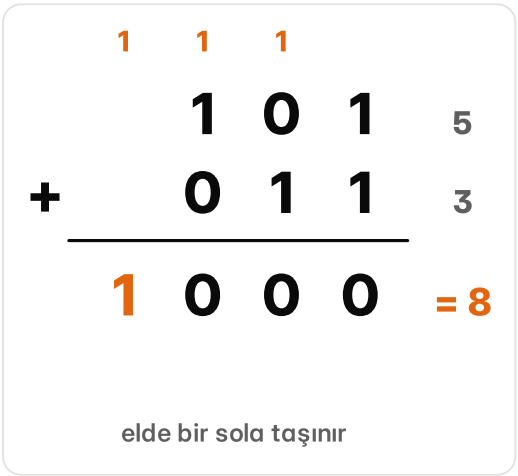
```
0 + 0 = 0
0 + 1 = 1
1 + 1 = 10        2 diye rakam yok: 0 yaz, elde 1
1 + 1 + 1 = 11    elde de gelirse 3: 1 yaz, elde 1
```

1 + 1 = 2 eder ama ikilikte 2 diye bir rakam yoktur; 2 ikilikte 10 diye yazılır. Bu yüzden 0 yazılır ve 1 elde olarak sola geçer, tıpkı onlukta 5 + 5'te 0 yazıp 1'i elde tutmak gibi.

Örnek olarak 5 ile 3'ü toplayalım. İkilikte 5 = 101, 3 = 011.

```
1 1 1          elde
  1 0 1        5
+ 0 1 1        3
-------
1 0 0 0        8
```

Sağdan başlarsın: 1 + 1 = 10, 0 yaz, elde 1. Ortada 0 + 1 + elde = 10, 0 yaz, elde 1. Solda 1 + 0 + elde = 10, 0 yaz, elde 1. Elde en sola iner: 1000, yani 8.

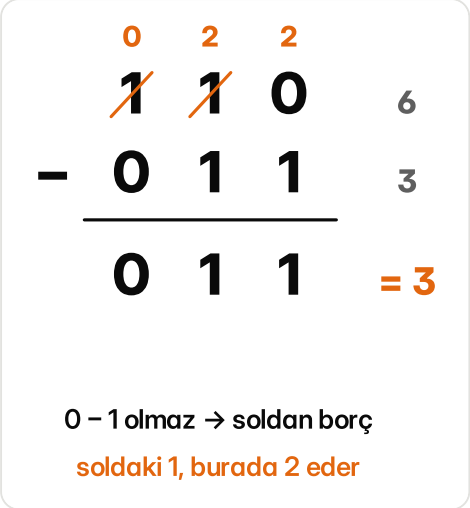



**Şekil 1.10** İkilikte 101 + 011: her basamakta elde bir sola taşınıyor, sonuç 1000 = 8.

Çıkarmada 0'dan 1 çıkmaz; soldan borç alınır. Soldaki basamaktaki 1, bu basamakta 2 eder, çünkü her basamak sağındakinin 2 katıdır; 2'den 1 çıkınca 1 kalır. Örnek olarak 6'dan 3'ü çıkaralım: 110 − 011.

```
  1 1 0       6
- 0 1 1       3
-------
  0 1 1       3
```

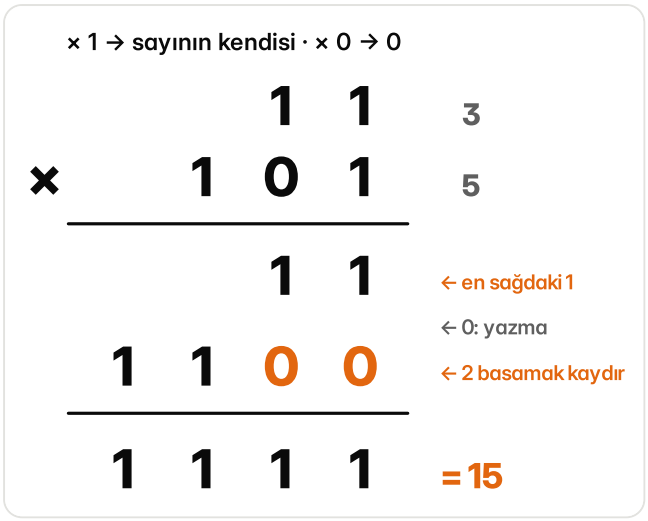
Sağdan: 0'dan 1 çıkmaz, soldan borç alırsın: 2 − 1 = 1. Ortadaki 1 borç verdiği için 0 kaldı; 0'dan 1 yine çıkmaz, soldan borç alırsın: 2 − 1 = 1. Soldaki 1 de borç verdi, 0 kaldı; 0 − 0 = 0. Sonuç 011, yani 3.




**Şekil 1.11** İkilikte 110 − 011: borç veren basamaklar 0'a iniyor, borç alanlar 2 sayılıyor; sonuç 011 = 3.

Çarpma en kolayıdır, çünkü bir rakamla çarpmak ya sayının kendisini verir (1 ile) ya 0 (0 ile). Örnek olarak 3 ile 5'i çarpalım: 11 × 101. Alttaki sayının her 1'i için üstteki sayıyı o basamağa kadar sola kaydırıp yazarsın; 0'lar için hiçbir şey yazmazsın. En sağdaki 1 için 11 yazılır, ortadaki 0 için hiçbir şey yazılmaz, soldaki 1 için 11 iki basamak kaydırılarak 1100 diye yazılır. Sonra ikisi toplanır:

```
      1 1      3
×   1 0 1      5
---------
      1 1      en sağdaki 1 için
  1 1 0 0      soldaki 1 için, 2 basamak kaydırılmış
---------
  1 1 1 1      15
```




**Şekil 1.12** İkilikte 11 × 101: 11 ile iki basamak kaydırılmış 1100 toplanıyor, sonuç 1111 = 15.

Buradan bir kural çıkar: 2 ile çarpmak, sayıyı bir basamak sola kaydırıp sağına 0 eklemekten ibarettir. Örneğin 101 (5) × 2 = 1010 (10). Onlukta 10 ile çarpmak da böyledir: 37 × 10 = 370. İşlemcinin içindeki çarpma devresi de temelde bu kaydırıp toplama işini yapar.

## 1 TB neden 931 GB?

videoda 10:19

1 terabaytlık bir disk bilgisayara takılınca 931 gigabayt görünür; aradaki 1.000 − 931 = 69 gigabayt kaybolmuş gibidir. Cevap, bitlerin ve baytların nasıl sayıldığındadır.

En küçük birim **bit** adını alır: tek bir 0 ya da 1. 8 bit 1 **bayt** eder. Bir bayt 00000000'dan 11111111'e, yani 0'dan 255'e kadar 256 farklı değer alır. Ekrandaki bir noktanın tek bir ışığının parlaklığı tam 1 bayttır.

Baytlardan sonra kilobayt, megabayt, gigabayt ve terabayt gelir. Onluk sistemde kilo 1.000 demektir; 1 kilogram 1.000 gramdır. Ama bellek ikilik çalıştığı için yazılımcılar uzun süre kiloyu 2 üzeri 10, yani 1.024 saydı. 1.024 hem 2'nin bir kuvvetidir hem de 1.000'e çok yakındır:

```
1 kilobayt = 1.024 bayt
1 megabayt = 1.024 kilobayt
1 gigabayt = 1.024 megabayt
1 terabayt = 1.024 gigabayt
```

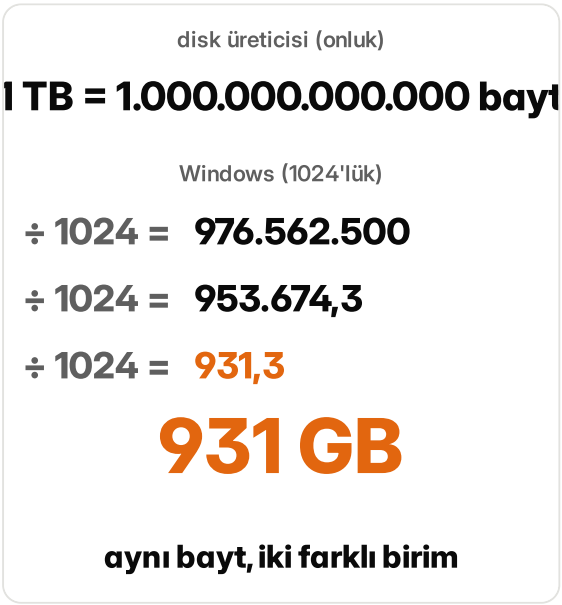



**Şekil 1.13** Onluk sistemde kilo = 1.000, bellekte 2 üzeri 10 = 1.024; her birim bir öncekinin 1.024 katı.

Disk üreticileri ise onluk sayar: 1 terabayt tam 1 trilyon, yani 1.000.000.000.000 bayttır. Windows aynı baytları 1.024'lük birimlerle böler. Bir trilyonu üç kez 1.024'e bölünce gigabayt cinsinden değer çıkar:

```
1.000.000.000.000 ÷ 1.024 = 976.562.500   kilobayt
      976.562.500 ÷ 1.024 = 953.674,3     megabayt
        953.674,3 ÷ 1.024 = 931,3         gigabayt
```

Kaybolan yer yoktur; aynı bayt sayısı iki farklı birimle okunur. Aynı hesap her diskte geçerlidir. Örneğin 500 gigabaytlık bir disk 500.000.000.000 bayttır; bunu 1.024 × 1.024 × 1.024 = 1.073.741.824'e bölünce yaklaşık 465,7 çıkar, yani Windows bu diski 465 GB civarında gösterir.




**Şekil 1.14** Üreticinin 1 TB'ı 1.000.000.000.000 bayt; üç kez 1.024'e bölününce 931,3 çıkıyor, yani Windows'taki 931 GB.

Karışıklığı bitirmek için 1.024'lük birimlere ayrı adlar verildi: **kibibayt** (KiB), **mebibayt** (MiB), **gibibayt** (GiB). Kilobayt (KB), megabayt (MB) ve gigabayt (GB) ise 1.000'lik birimlerin adıdır. Yani o disk 1 terabayt ya da 931 gibibayttır; Windows'un "931 GB" diye gösterdiği değer aslında 931 GiB'dir.




**Şekil 1.15** 1.000'lik birimler (KB, MB, GB) ve 1.024'lük birimler (KiB, MiB, GiB); 1 TB = 931 GiB.

### Özet

- Bir rakamın değeri basamağından gelir; basamak değerleri sağdan 1, taban, taban × taban diye büyür.
- Onluktan başka tabana çevirmek için sayı art arda tabana bölünür ve kalanlar aşağıdan yukarı okunur: 13 = 1101.
- Onluğa çevirmek için her rakam basamak değeriyle çarpılıp toplanır; ikilik rakamlar dörderli gruplanınca onaltılık, üçerli gruplanınca sekizlik olur.
- Kesirler 2 ile çarpılarak çevrilir; paydası 2'nin kuvveti olmayan 0,1 gibi kesirler ikilikte bitmez ve bellekte yaklaşık saklanır.
- İkilikte 1 + 1 = 10'dur; toplamada elde, çıkarmada borç vardır; 2 ile çarpmak bir basamak sola kaydırmaktır.
- 8 bit 1 bayttır; 1.024'lük birimler (KiB, MiB, GiB) ile 1.000'lik birimler (KB, MB, GB) aynı değildir.

## Sorular

**1** **İkilik sistemde yazılmış 10 sayısının onluk değeri kaçtır?**

**A** 10
**B** 1
**C** 2
**D** 8

**2** **Onaltılık sistemde 15 değerini tek bir rakamla gösteren hangisidir?**

**A** F
**B** E
**C** G
**D** 15

**3** **25 sayısını ikilik tabanda yaz.**

Cevap:

**4** **Onaltılık 3C sayısının onluk değeri kaçtır?**

**A** 42
**B** 61
**C** 312
**D** 60

**5** **Bir dosyanın sahibi okuyabilir ve yazabilir ama çalıştıramaz; grubu ve herkes yalnızca okuyabilir. Bu izin üç sekizlik rakamla nasıl yazılır?**

Cevap:

**6** **1 TB'lık bir disk Windows'ta neden 931 GB görünür?**

**A** Windows, diskin 69 GB'ını kendi dosyaları için ayırır.
**B** Üretici 1 TB'ı 1.000.000.000.000 bayt sayar; Windows aynı baytları 1.024'lük birimlerle böler.
**C** Disk zamanla yer kaybeder.
**D** Windows boyutu bayt yerine bit cinsinden gösterir.

**7** **0,1 sayısı ikilik sistemde neden bitmeyen bir rakam dizisiyle yazılır?**

**A** Bilgisayar ikilikte kesirli sayı yazamadığı için.
**B** 0,1 çok küçük bir sayı olduğu için.
**C** Paydası 10 = 2 × 5 olduğu için; 2 ile çarpmak paydadaki 5'ten kurtulamaz.
**D** İkilikte virgülün sağında yalnızca bir basamak olabildiği için.

**8** **İkilikte 1011 + 0110 toplamını ikilik olarak yaz.**

Cevap:

# 2 Verinin saklanması

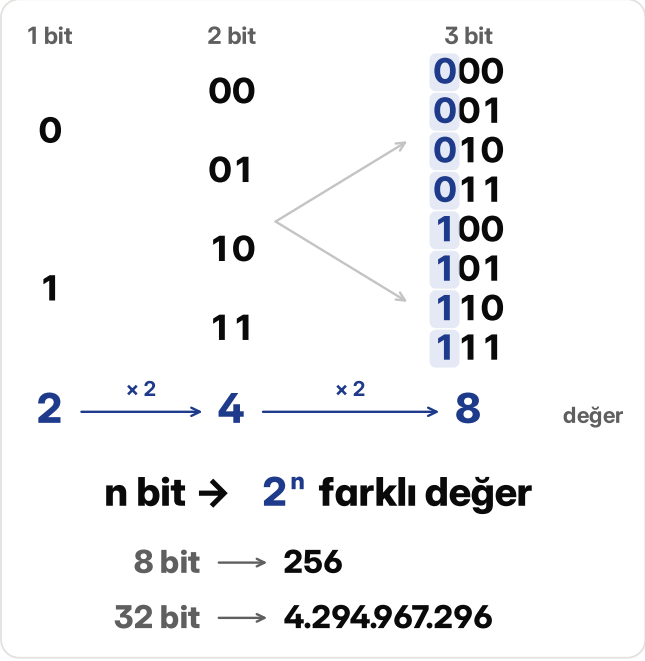
Bellekte yalnızca 0'lar ve 1'ler vardır. Bu bölümde tam sayıların, eksi sayıların ve kesirli sayıların bu bitlerle nasıl saklandığını, tek tek bitlerin nasıl işlendiğini ve harf, fotoğraf ve sesin nasıl sayıya dönüştüğünü öğreneceksin.

## 255 + 1 = 0?

videoda 11:22

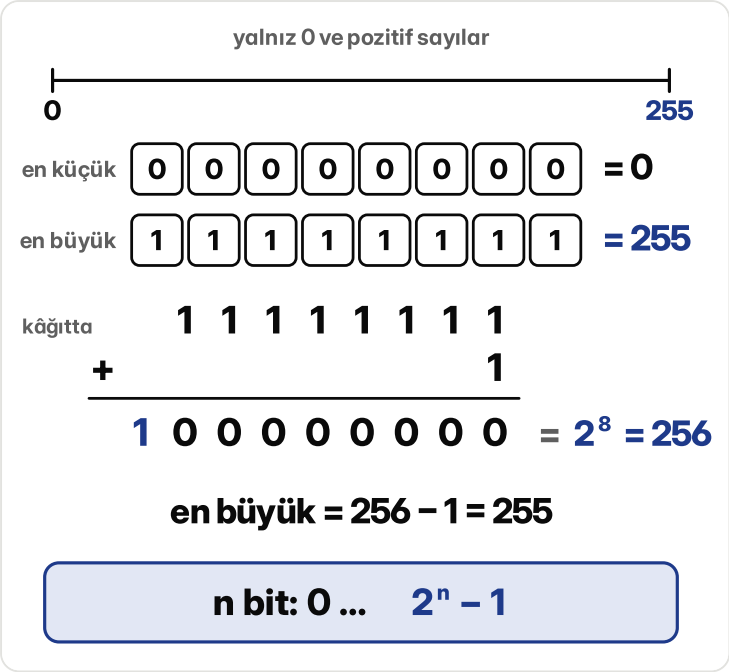
8 bitlik bir sayaç 255'e ulaştıktan sonra bir kez daha artırılırsa 256'yı değil, 0'ı gösterir. Nedenini anlamak için önce 8 bitle kaç farklı sayı yazılabildiğine bakmak gerekir.

1 bit 2 değer alır: 0 ya da 1. 2 bit 4 değer alır: 00, 01, 10 ve 11. 3. bit eklenince bu 4 değer 2 kez yazılır: bir kez başlarına 0, bir kez başlarına 1 konarak; 8 değer olur. Her yeni bit, var olan bütün değerleri 0 ile başlayan ve 1 ile başlayan 2 kopyaya ayırdığı için değer sayısını 2 katına çıkarır. Kural şudur: n bitle $2^n$ farklı değer yazılır. 8 bitle $2^8 = 256$, 32 bitle $2^{32} = 4.294.967.296$, yani 4 milyardan fazla değer.




**Şekil 2.1** 1, 2 ve 3 bitle yazılabilen değerler; her yeni bitle değer sayısı 2'den 4'e, 4'ten 8'e çıkıyor ve n bitle $2^n$ oluyor.

Bu değerlerin hepsini 0'a ve pozitif sayılara ayırırsak **işaretsiz tam sayı** elde ederiz. En küçüğü bütün bitleri 0 olan 00000000, yani 0'dır. En büyüğü bütün bitleri 1 olan 11111111'dir. Bunun kaç ettiğini bulmanın kısa yolu kâğıtta ona 1 eklemektir: sonuç 100000000, yani 1 ve ardından sekiz 0 çıkar. Bu sayı $2^8 = 256$'dır; demek ki en büyük değer onun bir eksiği, 255'tir. Genel olarak n bitlik işaretsiz bir sayı 0'dan $2^n - 1$'e kadar uzanır. 0 da bir değer olduğu için 0 ile 255 arasında tam 256 sayı vardır.

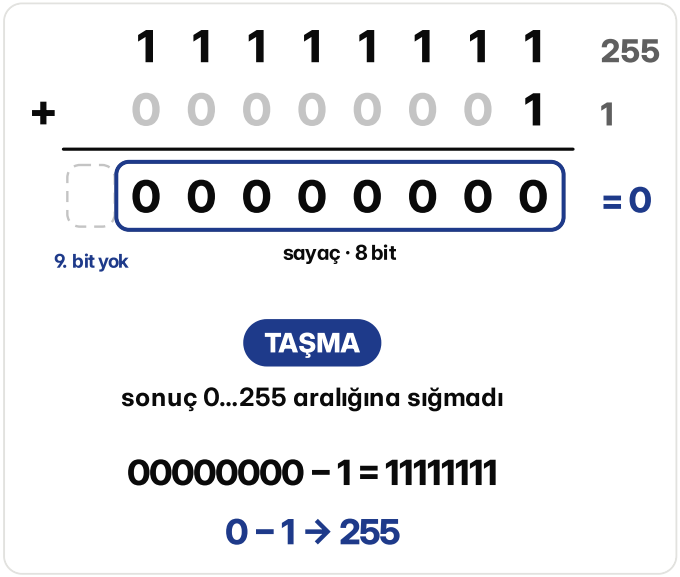



**Şekil 2.2** 8 bitlik işaretsiz sayının en küçüğü 00000000 = 0, en büyüğü 11111111 = 256 − 1 = 255; genel aralık 0 … $2^n - 1$.

Kâğıtta 9. basamağı yazabiliriz, ama sayaçta yalnızca 8 bit vardır. 11111111'e 1 eklerken her basamakta 1 + 1 olur: o basamağa 0 yazılır, elde 1 bir sola geçer. Elde sonuna kadar yürür ve 9. bite ulaşır; orada yer olmadığı için düşer. Sayaçta 00000000, yani 0 kalır.

```
  11111111     (255)
+ 00000001     (1)
1 00000000     9. bit yok: sayaçta 0 kalır
```

Bu duruma **taşma** denir: işlemin sonucu, bitlerin yazabileceği aralığın dışına çıkmıştır. Taşma ters yönde de olur: 0'dan 1 çıkarılınca sayaç 11111111'e, yani 255'e döner.




**Şekil 2.3** 11111111 + 1 işleminde elde 9. bite gidiyor, 8 bitlik sayaçta yer olmadığı için düşüyor ve sayaç 0 oluyor; 0 − 1 ise 255 veriyor.

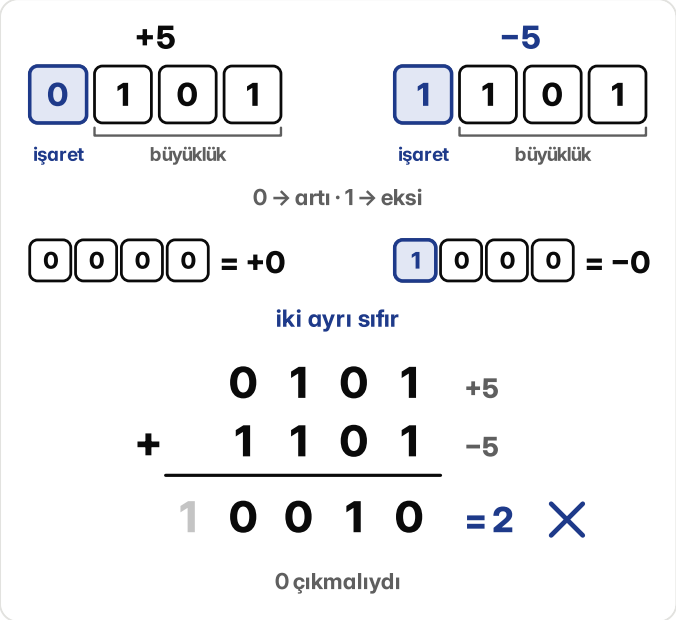
Taşma küçük bir hata sayılmaz. 1996'da Avrupa'nın yeni roketi Ariane 5, yatay hızla ilgili bir değer 16 bitlik bir tam sayıya sığmadığı için kalkıştan 40 saniye geçmeden rotasından çıktı ve imha edildi. Bu yüzden program yazılırken her sayıya, ulaşabileceği en büyük değere yetecek kadar bit ayrılır. Örneğin en çok 1.000.000'a kadar sayacak bir sayaç için 20 bit yeter, çünkü $2^{20}$ = 1.048.576'dır; 19 bit yetmez, çünkü $2^{19}$ = 524.288'dir.

## −5 nasıl saklanır?

videoda 12:56

Bellekte eksi işareti için ayrı bir sembol yoktur; yalnızca 0'lar ve 1'ler vardır. Eksi sayıları saklamak için 3 yol denenmiştir, bugün yalnızca biri kullanılır. Üçünü de 4 bitle ve 5 = 0101 sayısıyla görelim.

İlk akla gelen yol **işaret ve büyüklük** yöntemidir: en soldaki bit işarete ayrılır, 0 ise artı, 1 ise eksi demektir; kalan 3 bit sayının büyüklüğünü verir. Böylece +5 = 0101, −5 = 1101 olur. Okuması kolaydır, ama iki sorunu vardır. Birincisi, sıfırın iki ayrı yazımı çıkar: 0000 = +0 ve 1000 = −0. İkincisi, toplama devresi işaret bitini sıradan bir bit gibi toplar: 0101 + 1101 = 10010 eder, 4 bite sığmayan en soldaki 1 düşer ve sonuç 0010 = 2 olur. Oysa 5 ile −5'in toplamı 0 çıkmalıydı.



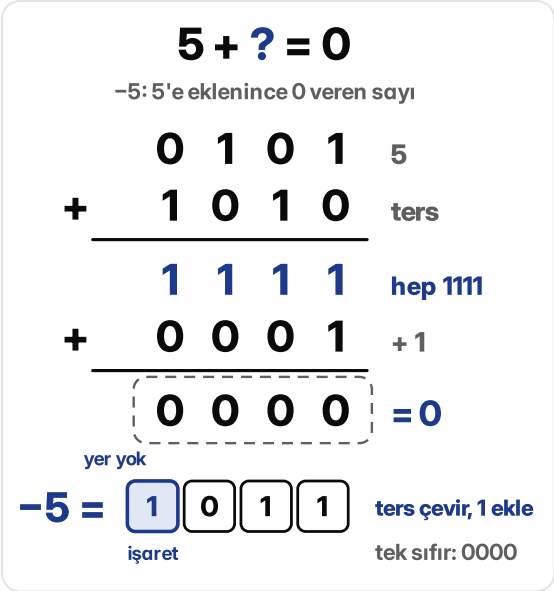

**Şekil 2.4** İşaret ve büyüklük yönteminde −5 = 1101; 0000 ve 1000 iki ayrı sıfır, 0101 + 1101 ise 0 yerine 2 veriyor.

İkinci yol **birin tümleyeni** adını taşır: bir sayının eksisini bulmak için bütün bitleri ters çevrilir. Bu yöntemde −5 = 1010 olur. 0101 + 1010 = 1111 eder; bu da −0'dır. Yani sıfır yine iki tanedir: 0000 ve 1111.

Bugün her bilgisayarın kullandığı yol **ikinin tümleyeni** adını taşır. Fikri şudur: −5, 5'e eklendiğinde 0 veren sayı olmalıdır. Bir sayıyla bitleri ters çevrilmiş hâli toplanınca her basamakta bir 0 ile bir 1 toplanır, sonuç hep 1111 olur. Buna 1 daha eklenirse 10000 çıkar; 4 bitte en soldaki 1'e yer yoktur, düşer ve geriye 0000 kalır. Demek ki −5'i bulmak için 5'in bitleri ters çevrilir ve 1 eklenir:

```
5        0101
ters     1010
+ 1      1011   →  -5
```

Bu yöntemde sıfırın tek yazımı vardır (0000) ve en soldaki bit yine işareti gösterir.



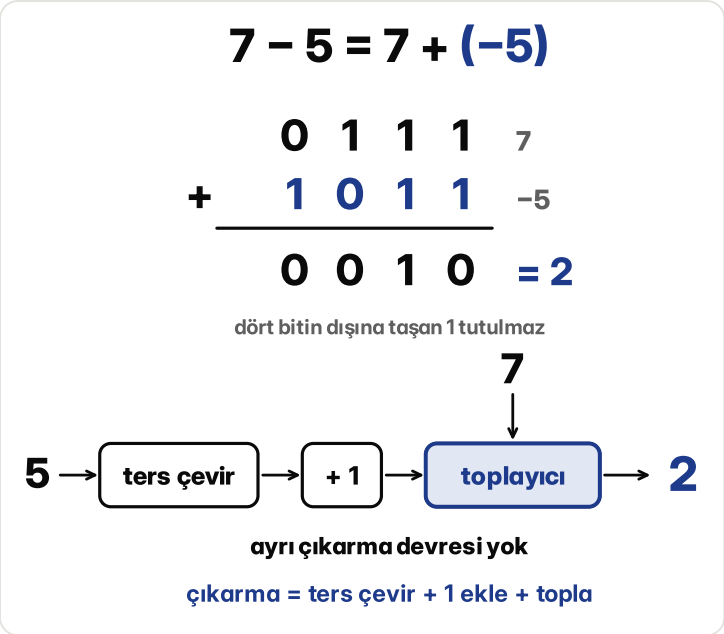

**Şekil 2.5** İkinin tümleyeni: 0101 + 1010 = 1111, 1 eklenince taşan 1 düşüyor ve 0 kalıyor; buradan −5 = 1011 çıkıyor.

4 bitin 16 değeri böylece ikiye ayrılır: 0 ile başlayan 0000 … 0111 sayıları 0'dan 7'ye, 1 ile başlayan 1000 … 1111 sayıları −8'den −1'e karşılık gelir. Aralık −8'den +7'ye uzanır. Aynı kuralla 32 bitte en büyük sayı $2^{31} - 1 = 2.147.483.647$ olur. YouTube izlenme sayısını böyle 32 bitlik bir sayıyla tutuyordu; 2014'te bir klip 2 milyarı aşan bu sınıra yaklaşınca sayaç 64 bite çıkarıldı.

İkinin tümleyeninin asıl kazancı çıkarmadadır: 7 − 5 = 7 + (−5). 7 ile −5'in bitleri toplanır, en soldan taşan 1 düşer:

```
  0111     (7)
+ 1011     (-5)
 10010     4 bitte 0010 = 2
```

Bu yüzden işlemcide ayrı bir çıkarma devresi gerekmez: çıkarılacak sayının bitleri ters çevrilir, 1 eklenir ve sonuç toplayıcıya verilir.




**Şekil 2.6** 7 − 5 = 7 + (−5) hesabı 0111 + 1011 = 0010 = 2 veriyor; altta 5'in ters çevir, 1 ekle ve topla adımlarından geçtiği boru hattı.

## 0,1 + 0,2 = ?

videoda 14:38

Bir bilgisayara 0,1 + 0,2'yi sorarsan 0,3 demez; 0,30000000000000004 der, yani 0,3'ün çok az fazlasını söyler. Bunun nedeni kesirli sayıların bellekte nasıl durduğundadır.

En basit yol **sabit nokta** denen yöntemdir: virgülün yeri baştan bellidir. Örneğin 8 bitin 4'ü tam kısma, 4'ü kesre ayrılabilir. Tam kısımdaki basamakların değerleri 8, 4, 2, 1; virgülden sonrakilerin değerleri 1/2, 1/4, 1/8, 1/16'dır. En küçük adım 0000,0001 = 1/16 = 0,0625'tir; en büyük sayı 1111,1111 = 15,9375'tir, 16'ya bile ulaşmaz. Parayı kuruşla tutmak da sabit noktadır: virgülden sonra hep 2 basamak vardır, 12,50 TL aslında 1250 kuruştur. Sorun aralığın darlığıdır: aynı bitlerle hem en yakın yıldızın uzaklığını (yaklaşık 40.000.000.000.000 km) hem bir atomun boyunu (yaklaşık 0,0000000001 m) yazamazsın.




**Şekil 2.7** Sabit nokta: 8 bitin 4'ü tam kısma, 4'ü kesre ayrılmış; en büyük sayı 1111,1111 = 15,9375, yıldızın uzaklığı ve atomun boyu bu bitlere sığmıyor.

Fizikte bu sorunun çözümü hazırdır: 300.000 km, $3 \times 10^5$ km diye yazılır. Rakamlar ayrı, virgülün yeri ayrı tutulur; üs değişince virgül kayar ($30.000 = 3 \times 10^4$). **Kayan nokta** denen yöntemde bilgisayar aynısını ikilikte yapar. 6,5 ikilikte 110,1'dir; virgül 2 basamak sola kaydırılınca $1{,}101 \times 2^2$ olur. Saklanan 3 parça vardır: işaret (artı olduğu için 0), üs (2) ve virgülden sonraki rakamlar. Bu son parçaya **mantis** denir.

```
6,5  =  110,1 (ikilik)  =  1,101 × 2²
işaret: 0     üs: 2     mantis: 101
```

Virgülün solundaki rakam bu yazımda hep 1 olduğu için saklanmaz.



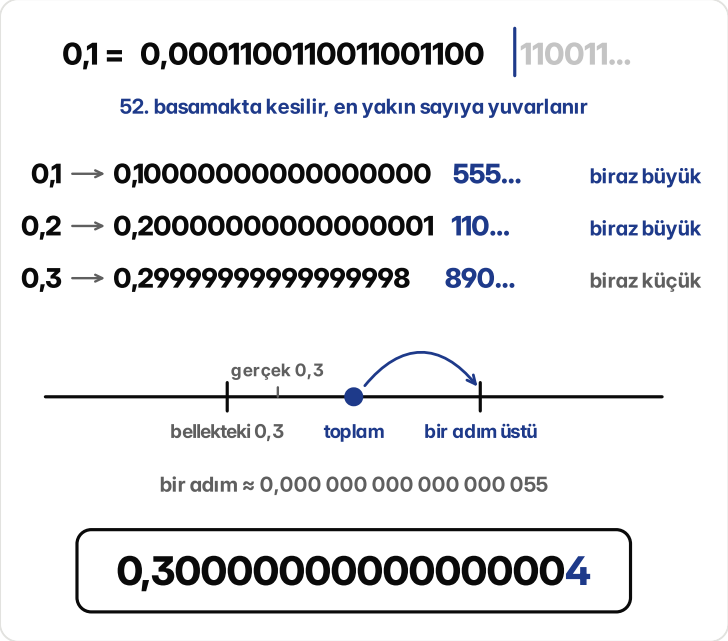

**Şekil 2.8** 6,5 ikilikte $1{,}101 \times 2^2$ diye yazılıyor; saklanan parçalar işaret 0, üs 2 ve mantis 101, baştaki 1 saklanmıyor.

**Tek duyarlık** denen biçimde bu parçalar 32 bite sığar: 1 bit işaret, 8 bit üs, 23 bit mantis. **Çift duyarlık** denen biçimde 64 bit vardır: 1 bit işaret, 11 bit üs, 52 bit mantis. Sayının inceliğini mantis belirler: tek duyarlık yaklaşık 7, çift duyarlık 15-16 basamağı doğru tutar. Örneğin π tek duyarlıkta 3,141593'e kadar, çift duyarlıkta 3,141592653589793'e kadar doğru kalır.

0,1 ise ikilikte hiç bitmez: 0,0001100110011001100110011... diye sürer. Çift duyarlıkta mantis 52. basamakta kesilir ve sayı en yakın değere yuvarlanır. Bu yüzden bellekteki 0,1 gerçek 0,1'den çok az büyüktür, 0,2 de öyle; bellekteki 0,3 ise çok az küçüktür:

```
0,1  →  0,1000000000000000555…   biraz büyük
0,2  →  0,2000000000000001110…   biraz büyük
0,3  →  0,2999999999999998889…   biraz küçük
```

İki fazlalık toplanır ve sonuç yeniden yuvarlanınca bellekteki 0,3'e değil, onun bir adım üstündeki sayıya düşer. Bu civarda iki komşu sayı arasındaki adım yaklaşık 0,00000000000000055'tir. Ekranda 0,3, ardından 15 sıfır ve bir 4 görürsün.




**Şekil 2.9** 0,1'in ikilik açılımı 52. basamakta kesiliyor; bellekteki 0,1 ve 0,2 biraz büyük, 0,3 biraz küçük, toplam ise bellekteki 0,3'ün bir adım üstüne düşüyor.

Bu bir arıza değil, sonlu sayıda bitin bedelidir: çoğu kesir bellekte yalnızca yaklaşık saklanır. Bu yüzden programlarda iki kesirli sayının eşitliğine bakılmaz; farklarının yeterince küçük olup olmadığına bakılır (|...| farkın işaretsiz büyüklüğüdür):

```
eğer 0,1 + 0,2 = 0,3 ise                  yanlış sonuç verir
eğer |0,1 + 0,2 − 0,3| < 0,000001 ise     doğru sonuç verir
```

Para hesapları da bu yüzden kayan noktayla değil, kuruş cinsinden tam sayıyla yapılır: 12,50 TL, 1250 kuruş olarak saklanır ve yuvarlama olmaz.

## Bir bayt, sekiz ayar

videoda 16:28

Bir bayt 8 bittir; her biti ayrı bir açık-kapalı ayarı tutabilir. Örneğin 10010110 baytında 1 olan bitler açık, 0 olanlar kapalı ayarlardır; bitler sağdan sayılır. Peki yalnızca bir ayarı, ötekilere dokunmadan nasıl okur ya da değiştirirsin? Bunun için sayıları bit bit işleyen işlemler vardır:

```
VE    VEYA    ÖZEL VEYA    DEĞİL    kaydırma
&     |       ^            ~        <<  >>      (C dilinde)
```

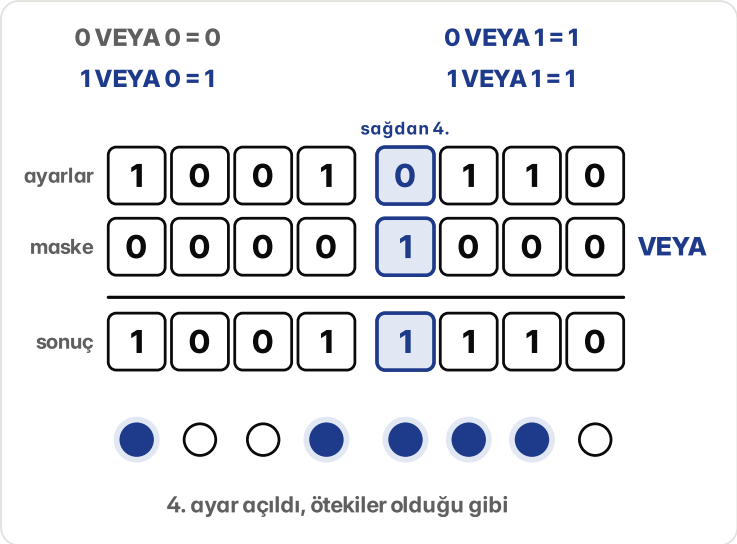
VE işlemi iki sayının aynı sıradaki bitlerine bakar: ikisi de 1 ise sonuç 1, değilse 0'dır. Sağdan 3. ayarın açık olup olmadığını öğrenmek için bayt, yalnız o biti 1 olan 00000100 sayısıyla VE işlemine sokulur. Yalnız ilgilenilen biti 1 olan böyle bir sayıya **maske** denir. Maskenin 0'ları öteki bitlerin hepsini 0 yapar, yani siler; geriye yalnız bakılan bit kalır. Sonuç 0 değilse ayar açıktır.

```
ayarlar   10010110
maske     00000100
VE        00000100     0 değil: 3. ayar açık
```




**Şekil 2.10** Ayarlar baytı 10010110, 00000100 maskesiyle VE işlemine sokuluyor; sonuç 00000100, 0 olmadığı için sağdan 3. ayar açık.

VEYA işleminde bitlerden en az biri 1 ise sonuç 1'dir. Sağdan 4. ayarı açmak için bayt, o biti 1 olan 00001000 maskesiyle VEYA işlemine sokulur. Maskede 1 olan yer, önceden ne olursa olsun 1 olur; maskede 0 olan yerlerde baytın biti olduğu gibi kalır: 10010110 VEYA 00001000 = 10011110.

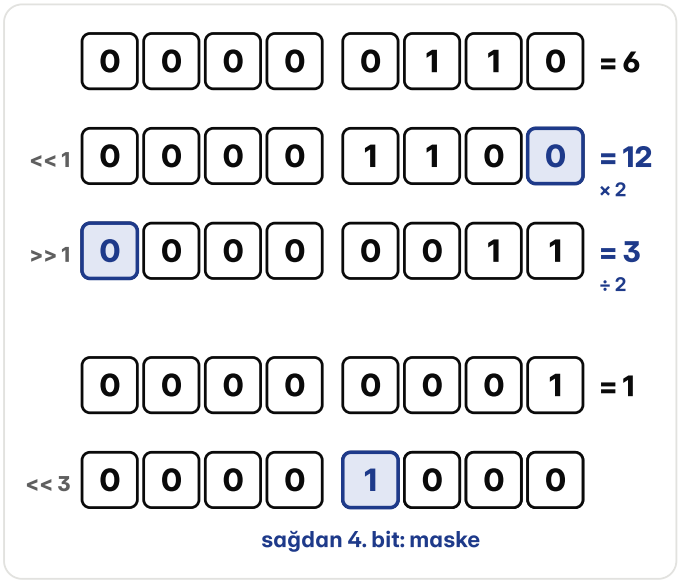



**Şekil 2.11** 10010110 VEYA 00001000 = 10011110; sağdan 4. ayar açılıyor, öteki bitler olduğu gibi kalıyor.

ÖZEL VEYA işleminde bitler farklıysa sonuç 1, aynıysa 0'dır. Bu yüzden maskedeki 1, baytın o bitini ters çevirir: açıksa kapanır, kapalıysa açılır. Maskedeki 0'lar öteki bitlere dokunmaz. DEĞİL ise tek bir sayıyla çalışır ve bütün bitleri ters çevirir.

```
10010110 ÖZEL VEYA 00000010  =  10010100     sağdan 2. ayar kapandı
DEĞİL 10010110               =  01101001     bütün bitler ters
```

Son işlem kaydırmadır: bütün bitler bir yana itilir, boşalan yere 0 girer. Onluk sistemde bir sayının sonuna 0 eklemek onu 10 katına çıkarır; ikilikte de bitleri sola 1 basamak kaydırmak sayıyı 2 katına çıkarır. Sağa 1 basamak kaydırmak ise sayıyı yarıya indirir (tek sayıda en sağdaki 1 düşer, küsurat atılır). Örneğin 00000110 = 6 sola 1 basamak kaydırılınca 00001100 = 12, sağa 1 basamak kaydırılınca 00000011 = 3 olur. Maskeler de böyle yapılır: 1 sayısı (00000001) sola 3 basamak kaydırılınca sağdan 4. biti 1 olan 00001000 maskesi çıkar.




**Şekil 2.12** 00000110 = 6 sola 1 basamak kaydırılınca 12, sağa 1 basamak kaydırılınca 3 oluyor; 00000001 sola 3 basamak kaydırılınca sağdan 4. biti 1 olan maske çıkıyor.

Programlar açık-kapalı seçenekleri sık sık böyle tek bir sayının bitlerinde tutar ve her birini maskeyle yönetir:

- Okumak için VE ve maske kullanılır.
- Açmak için VEYA ve maske kullanılır.
- Ters çevirmek, yani açığı kapatmak ya da kapalıyı açmak için ÖZEL VEYA ve maske kullanılır.

## Her şey sayı

videoda 17:45

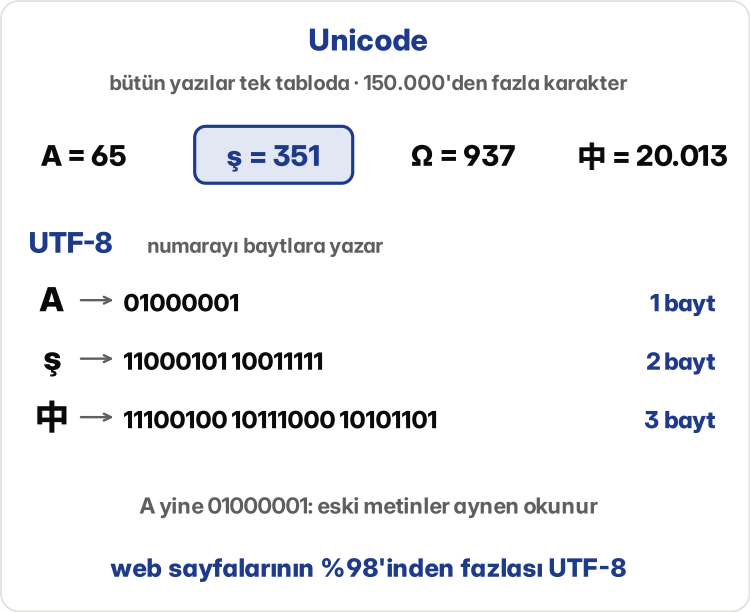
Bellek yalnızca sayı tutar; peki bir harf, bir fotoğraf ya da bir şarkı nasıl sayı olur? Üçünde de yol aynıdır: önce küçük parçalara bölünür, sonra her parça bir sayıyla yazılır.

Metinde parça karakterdir ve her karaktere bir numara verilir. 1963'te çıkan **ASCII** tablosunda büyük A 65 (1000001), küçük a 97 (1100001), boşluk 32 (0100000) numarasını alır. ASCII 7 bit kullanır; $2^7 = 128$ karakterlik yer vardır ve ş, ğ, ı gibi harflere bu 128'in içinde yer yoktu. Türkçe için 8 bit kullanan ayrı tablolar yapıldı, ama aynı sayı başka tabloda başka harf oldu: 254 numarası Türkçe tabloda ş, Batı Avrupa tablosunda þ'dir. Türkçe tabloyla yazılmış "Teşekkürler", Batı Avrupa tablosuyla okununca "Teþekkürler" görünür.

**Unicode** bu karmaşayı tek tabloyla bitirdi: dünyadaki yazıların 150.000'den fazla karakteri birer numara aldı. A yine 65'tir; ş 351, Ω 937, Çince 中 karakteri 20.013 numarasını alır. **UTF-8** bu numaraları baytlara yazmanın yoludur: İngilizce harfler eskisi gibi tek bayt, ş 2 bayt, bir Çince karakter 3 bayt tutar.

```
A    01000001                      1 bayt
ş    11000101 10011111             2 bayt
中   11100100 10111000 10101101    3 bayt
```

A'nın UTF-8 yazımı ASCII'dekiyle aynı olduğu için eski metinler değişmeden okunur; bugün web sayfalarının neredeyse hepsi UTF-8 ile yazılır.

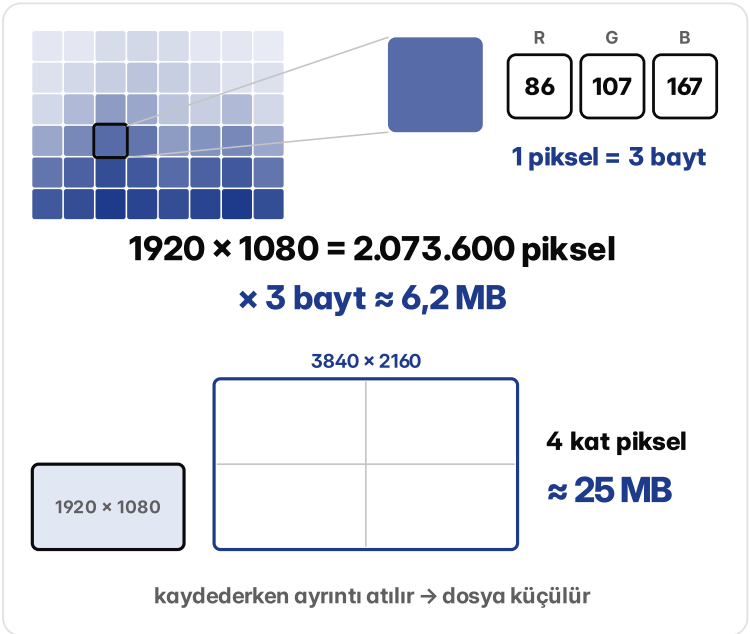



**Şekil 2.13** Unicode numaraları A = 65, ş = 351, Ω = 937, 中 = 20.013; UTF-8 bunları 1, 2 ve 3 bayta yazıyor.

Görüntüde parça pikseldir: fotoğraf, **piksel** denen küçük karelerden oluşan bir ızgaradır. Her pikselin rengi kırmızı, yeşil ve mavi miktarını veren 3 baytla yazılır. Yatayda 1920, dikeyde 1080 piksellik bir görüntü için:

```
1920 × 1080 = 2.073.600 piksel
2.073.600 × 3 bayt = 6.220.800 bayt ≈ 6,2 MB
```

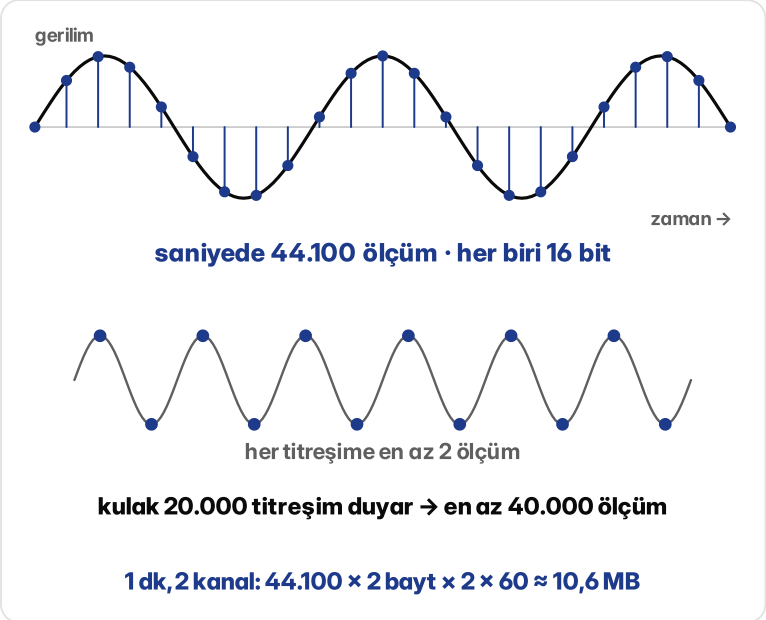
Eni de boyu da 2 katına çıkarsa (3840 × 2160) piksel sayısı 4 katına çıkar: 24.883.200 bayt, yani 25 MB'a yakın. Fotoğraf dosyaları bundan küçüktür, çünkü kaydedilirken gözün fark etmeyeceği ayrıntılar atılır.




**Şekil 2.14** Bir piksel R, G, B olarak 3 bayt; 1920 × 1080 görüntü yaklaşık 6,2 MB, eni ve boyu 2 katı olan 3840 × 2160 görüntü yaklaşık 25 MB.

Seste parça ölçümdür. Mikrofon havadaki ses titreşimini sürekli değişen bir gerilime çevirir. Müzik kaydında bu gerilim saniyede 44.100 kez ölçülür ve her ölçüm 16 bitlik (2 baytlık) bir sayıyla yazılır; buna **örnekleme** denir. Neden bu kadar sık? Bir titreşimi yakalamak için saniyede onun en az 2 katı ölçüm gerekir; insan kulağı saniyede 20.000 titreşime kadar duyduğu için saniyede en az 40.000 ölçüm gerekir. Bir dakikalık çift kanallı müzik:

```
44.100 × 2 bayt × 2 kanal × 60 saniye = 10.584.000 bayt ≈ 10,6 MB
```




**Şekil 2.15** Ses dalgası saniyede 44.100 kez ölçülüyor, her titreşime en az 2 ölçüm düşüyor; bir dakikalık çift kanallı kayıt yaklaşık 10,6 MB.

Yani yazı, fotoğraf ve şarkı bellekte aynı şeydir: sayılar. 01000001 = 65 sayısı bir metin programı için A harfi, bir resim programı için kırmızısı, yeşili ve mavisi 65 olan koyu gri bir pikselin parçası, bir ses programı için dalgadan tek bir ölçümdür. Onları harf, renk ya da ses yapan, programın onları nasıl okuduğudur.

### Özet

- n bitle $2^n$ farklı değer yazılır; n bitlik işaretsiz tam sayı 0'dan $2^n - 1$'e uzanır ve sonuç bu aralığın dışına çıkınca taşma olur.
- Eksi sayılar ikinin tümleyeniyle saklanır: bitler ters çevrilir ve 1 eklenir; 4 bitte aralık −8'den +7'ye uzanır ve çıkarma toplamayla yapılır.
- Kesirli sayılar kayan noktayla işaret, üs ve mantis olarak saklanır; tek duyarlık 32, çift duyarlık 64 bit kullanır.
- 0,1 gibi çoğu kesir ikilikte tam yazılamaz; bu yüzden kesirli sayıların eşitliğine değil, farklarının küçüklüğüne bakılır.
- VE, VEYA, ÖZEL VEYA, DEĞİL ve kaydırma, bir sayının bitlerini maskeyle okumaya, açmaya ve ters çevirmeye yarar.
- Harf, piksel ve ses ölçümü bellekte sayıdır; o sayıya anlamı, onu okuyan program verir.

## Sorular

**1** **8 bitle kaç farklı değer yazılabilir?**

**A** 255
**B** 256
**C** 16
**D** 512

**2** **UTF-8 için hangisi doğrudur?**

**A** Her karakteri 2 baytla yazar.
**B** 7 bitlik, 128 karakterlik bir tablodur.
**C** A harfini ASCII'deki gibi 1 baytla, ş harfini 2 baytla yazar.
**D** Türkçe harfler için 254 gibi tek baytlık numaralar kullanır.

**3** **4 bitlik ikinin tümleyeninde hangi aralıktaki sayılar yazılabilir?**

**A** −7'den +7'ye
**B** 0'dan 15'e
**C** −7'den +8'e
**D** −8'den +7'ye

**4** **4 bitlik ikinin tümleyeniyle −3 nasıl yazılır? 4 biti yaz.**

Cevap:

**5** **Bilgisayarda 0,1 + 0,2 neden tam 0,3 çıkmaz?**

**A** 0,1 ve 0,2 ikilikte tam yazılamadığı için bellekte en yakın değere yuvarlanmıştır; toplam da yeniden yuvarlanır.

**B** Toplama sırasında elde, sayının dışına taşar ve düşer.

**C** Kesirli sayılar ikinin tümleyeniyle saklandığı için sonuca 1 eklenir.

**D** Tek duyarlık kullanıldığı içindir; çift duyarlıkta sonuç tam 0,3 çıkar.

**6** **Bir bayt 01101100. Bu bayt 00000100 maskesiyle ÖZEL VEYA işlemine sokulursa sonuç ne olur?**

**A** 00000100

**B** 01101000

**C** 01101100

**D** 10010011

**7** **En çok 1.000.000'a kadar sayacak işaretsiz bir sayaca en az kaç bit ayrılmalıdır?**

Cevap: _____

**8** **Her pikseli 3 bayt olan 1280 × 720 piksellik bir görüntü, hiç ayrıntı atılmadan saklanırsa kaç bayt tutar?**

Cevap: _____

# 3 Algoritmaya giriş

Bu bölümde bir bilgisayara iş yaptırmanın ilk adımını göreceksin: işi adım adım tarif etmek. Algoritmanın ne olduğunu ve iyi bir algoritmanın özelliklerini, algoritmayı akış diyagramı ve sözde kodla yazmayı, döngüyü ve bir algoritmanın programa nasıl dönüştüğünü öğreneceksin.

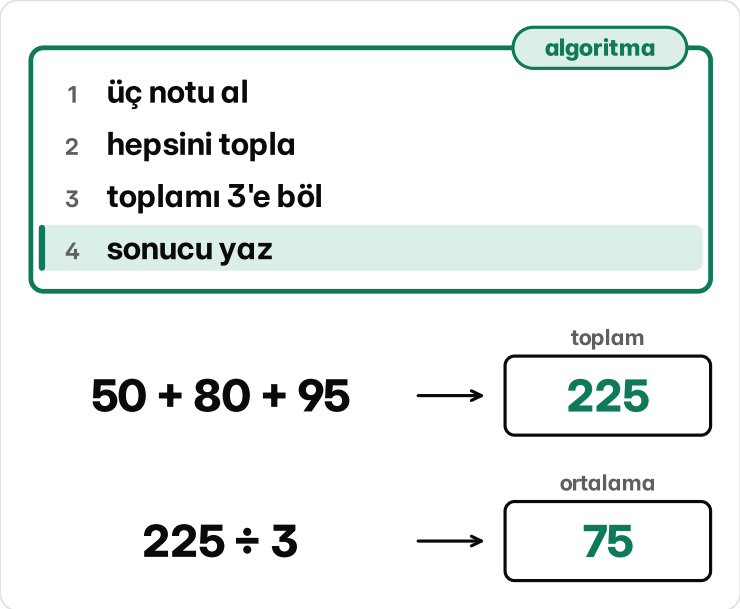
## Algoritma nedir?

videoda 19:33

Saniyede milyarlarca işlem yapan bir bilgisayar, üç notun ortalamasını kendi başına bulamaz. Çünkü ortalamanın ne demek olduğunu bilmez; yalnızca ona söylenen adımları, söylendiği sırayla yapar. Bu adımları biz yazarız:

```
1  üç notu al
2  hepsini topla
3  toplamı 3'e böl
4  sonucu yaz
```

Notlar 50, 80 ve 95 ise toplam 50 + 80 + 95 = 225 eder; 225 ÷ 3 = 75, yani ortalama 75'tir. Bir işi yapmak için sırayla izlenen böyle adımlara **algoritma** denir.




**Şekil 3.1** Ortalamanın dört adımlı algoritması ve 50, 80, 95 notlarıyla uygulanışı: toplam 225, ortalama 75.

İyi bir algoritmanın üç özelliği vardır.

Birincisi, **açıktır**: her adımı tek bir anlama gelir. "Toplamı uygun bir sayıya böl" dersek bilgisayar hangi sayıya böleceğini bilemez; "toplamı 3'e böl" dememiz gerekir. Bir insan neyin uygun olduğunu tahmin edebilir, bilgisayar tahmin etmez.

İkincisi, **sonludur**: belli sayıda adımdan sonra durur. Ortalama tarifi 4 adımda biter. Bölmeden sonra bir de "başa dön" adımı koysaydık, bilgisayar sonsuza kadar toplayıp bölerdi ve sonucu hiç yazamazdı.

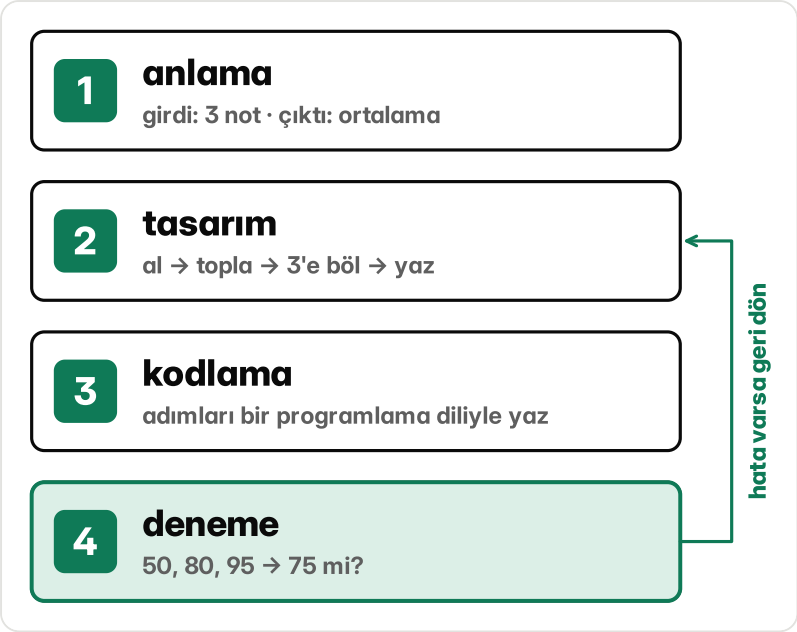



**Şekil 3.2** Bölmeden sonra eklenen "başa dön" adımı: algoritma ilk adımlara sonsuza kadar geri dönüyor, "sonucu yaz" adımına hiç gelinmiyor.

Üçüncüsü, **girdisi ve çıktısı** bellidir. Girdi, algoritmaya verdiğimiz şeydir; burada üç not. Çıktı, algoritmanın bize verdiği sonuçtur; burada ortalama. Girdi değişince çıktı da değişir: notlar 60, 70 ve 80 olsaydı aynı adımlar 210 ÷ 3 = 70 sonucunu verirdi.

Bir problemi algoritmayla çözerken dört aşamadan geçeriz:

- Anlama: elimizde ne var, bizden ne isteniyor? Burada girdi 3 not, istenen çıktı ortalama.
- Tasarım: adımları sırayla yazarız: al, topla, 3'e böl, yaz.
- Kodlama: bu adımları bir programlama diliyle yazarız.
- Deneme: sonucunu bildiğimiz bir örnek veririz. 50, 80 ve 95 için 75 çıkmazsa hata adımlardadır; geri dönüp düzeltiriz.




**Şekil 3.3** Problem çözmenin dört aşaması: anlama, tasarım, kodlama ve deneme; denemede hata çıkarsa geri dönülüyor.

Bilgisayarın yaptığı her işin arkasında, birinin adım adım yazdığı böyle bir algoritma vardır. Bilgisayar o adımları yalnızca çok hızlı uygular.

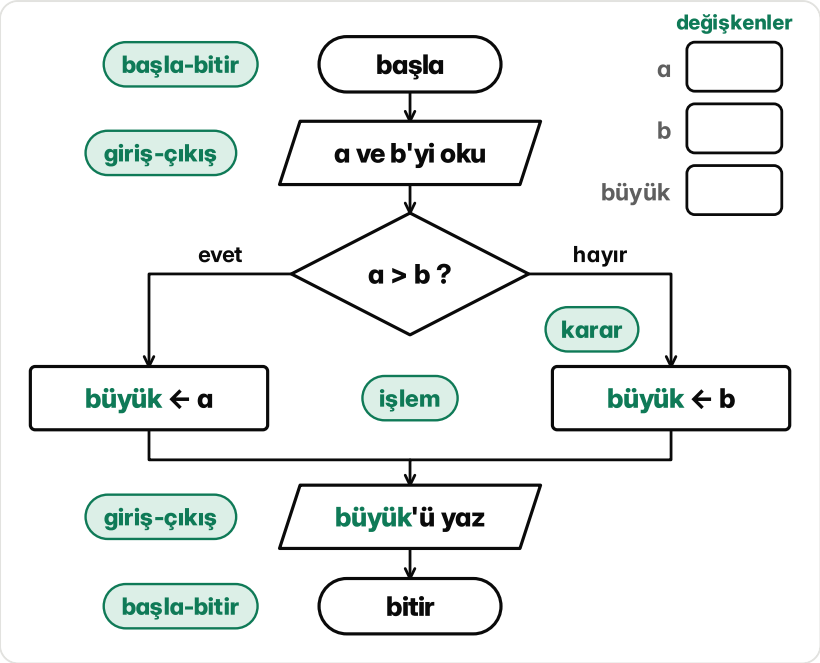
## Bu çizimi oku

videoda 21:02

Bir algoritma çizimle de gösterilebilir. Adımları şekillerle, sırayı oklarla gösteren çizime **akış diyagramı** denir. Okları izleyen kişi, algoritmanın nasıl ilerlediğini bir bakışta görür. Örnek olarak iki sayıdan büyüğünü bulan bir algoritmayı ele alalım. Akış diyagramında her şeklin ayrı bir anlamı vardır:

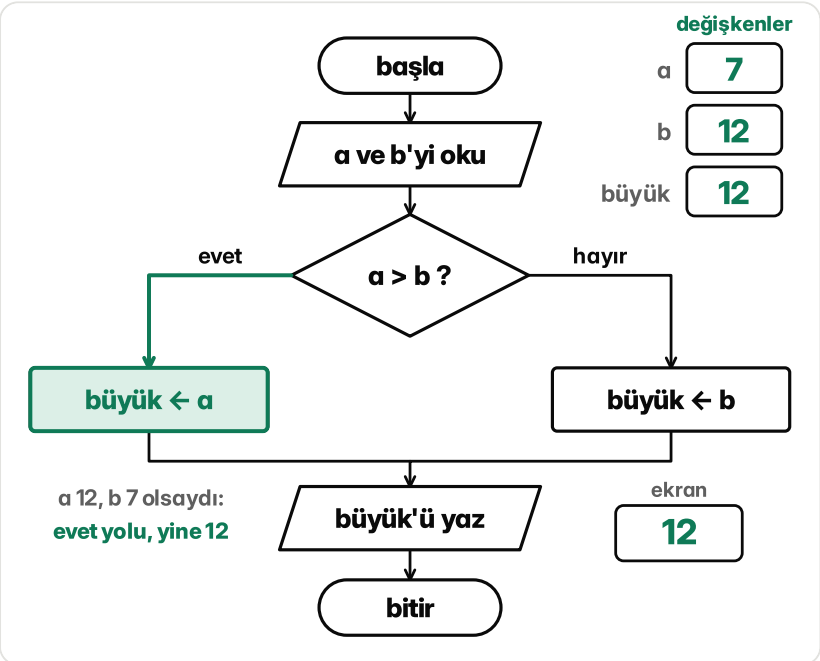
- Oval başlangıcı ve bitişi gösterir: "başla" ve "bitir".
- Paralelkenar giriş ve çıkıştır: burada a ve b sayılarını okuruz, sonda da büyük olanı yazarız.
- Eşkenar dörtgen bir karardır: içinde bir soru vardır (a > b ?), cevabı evet ya da hayırdır ve her cevap için ayrı bir ok çıkar.
- Dikdörtgen bir işlemdir: "büyük ← a" yazan kutudaki ok, sağdaki değerin soldaki kutuya konacağını söyler.

a, b ve büyük gibi değer saklayan bu kutulara **değişken** denir.




**Şekil 3.4** İki sayının büyüğünü bulan akış diyagramı; yanında şekillerin anlamları (başla-bitir, giriş-çıkış, karar, işlem) ve a, b, büyük değişkenleri.

Şimdi algoritmayı çalıştıralım: a 7, b 12 olsun. "Başla"dan çıkıp iki sayıyı okuruz. Karar kutusu sorar: 7, 12'den büyük mü? Cevap hayır; hayır okunu izleriz ve büyük değişkenine 12 konur. Sonra 12 yazılır ve algoritma biter. Sayılar tersine olsaydı, yani a 12, b 7 olsaydı, evet okuna gidilir ve yine büyük olan, 12 yazılırdı.




**Şekil 3.5** a = 7, b = 12 ile çalışma: karar kutusundan hayır yolu izleniyor, büyük değişkenine ve ekrana 12 yazılıyor.

Aynı algoritma yazıyla da anlatılabilir; her şekil bir satır olur. Buna **sözde kod** (pseudocode) denir: hiçbir programlama dilinin kuralına uymaz ama her adımı açıkça söyler.

```
1  başla
2  a ve b'yi oku
3  eğer a > b ise
4      büyük ← a
5  değilse
6      büyük ← b
7  büyük'ü yaz
8  bitir
```

"Eğer" ile başlayan satır karar kutusudur. Cevap evetse hemen altındaki satır, yani 4. satır çalışır; hayırsa "değilse" satırının altındaki 6. satır çalışır. İçeri girintili satırlar, hangi satırın hangi cevaba ait olduğunu gösterir.

Bu kez a 15, b 9 olsun. Sayılar okunur, sonra soru sorulur: 15, 9'dan büyük mü? Evet; büyük 15 olur, "değilse" kolu atlanır ve 15 yazılır.




**Şekil 3.6** Sözde kodun a = 15, b = 9 ile çalışması: evet kolu çalışıyor, değilse kolu atlanıyor; büyük ve ekran 15.

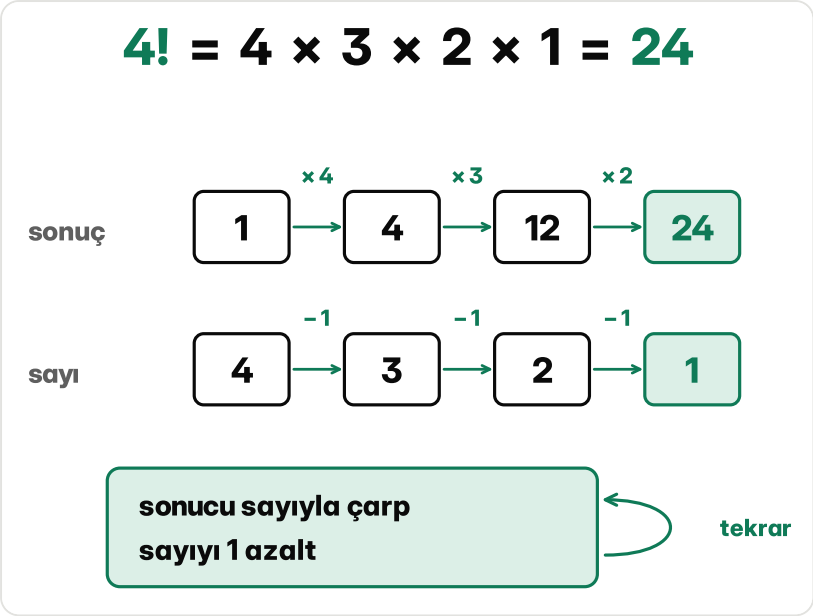
Programcılar bir işi koda dökmeden önce akışını çoğu zaman böyle çizer ya da yazar. Çünkü bir hatayı kâğıt üstünde bulmak, programın içinde bulmaktan çok daha kolaydır.

## 100 faktöriyel

videoda 22:32

100 **faktöriyel**, kısaca 100!, 100'den 1'e kadar bütün sayıların çarpımıdır: 100 × 99 × 98 × ... × 2 × 1. Bunu bilgisayara yaptırmak için 99 çarpmanın hepsini tek tek yazmamız gerekmez. Önce küçük bir örneğe bakalım.

4 faktöriyel 4 × 3 × 2 × 1 = 24 eder. Hesaplarken her adımda aynı şeyi yaparız: elimizdeki sonucu sayıyla çarpar, sonra sayıyı 1 azaltırız. Sonuç 1'den başlayıp 4, 12, 24 diye büyür; sayı da 4, 3, 2, 1 diye iner. Yani aynı iki adımı tekrar ettirmek yeter.

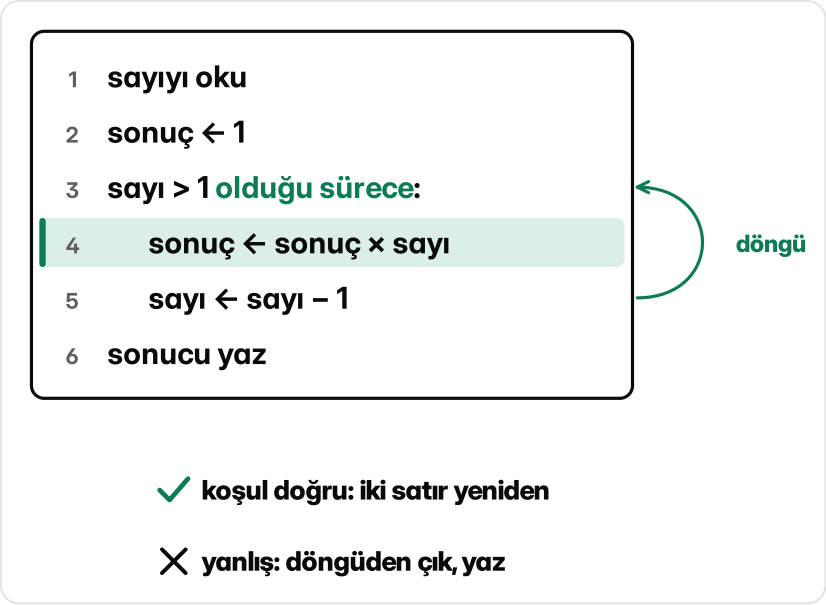



**Şekil 3.7** 4! = 24: sonuç 1, 4, 12, 24 diye büyürken sayı 4, 3, 2, 1 diye iniyor; her adımda aynı iki iş tekrar ediliyor.

Bunu sözde kodla yazalım. Satırlardaki ok (←), sağdaki değerin soldaki kutuya konduğunu gösterir.

```
1  sayıyı oku
2  sonuç ← 1
3  sayı > 1 olduğu sürece:
4      sonuç ← sonuç × sayı
5      sayı ← sayı − 1
6  sonucu yaz
```

Önce sayıyı okuruz ve sonuç kutusuna 1 koyarız; çünkü 1, çarpmada sonucu değiştirmez. Sonra şunu deriz: sayı 1'den büyük olduğu sürece, sonucu sayıyla çarp ve sayıdan 1 çıkar. Koşul doğru oldukça 4. ve 5. satırlar yeniden çalışır; koşul yanlış olunca döngüden çıkılır ve sonuç yazılır. Bu geri dönüşe **döngü** denir.




**Şekil 3.8** Faktöriyelin altı satırlık sözde kodu: koşul doğruysa 4. ve 5. satırlar yeniden çalışıyor, yanlışsa döngüden çıkılıp sonuç yazılıyor.

Algoritmayı 4 ile çalıştıralım; her satır, o turdan sonraki değerleri gösterir:

```
başta           sayı 4   sonuç 1
4 > 1, evet     sayı 3   sonuç 4
3 > 1, evet     sayı 2   sonuç 12
2 > 1, evet     sayı 1   sonuç 24
1 > 1, hayır    döngü biter, 24 yazılır
```

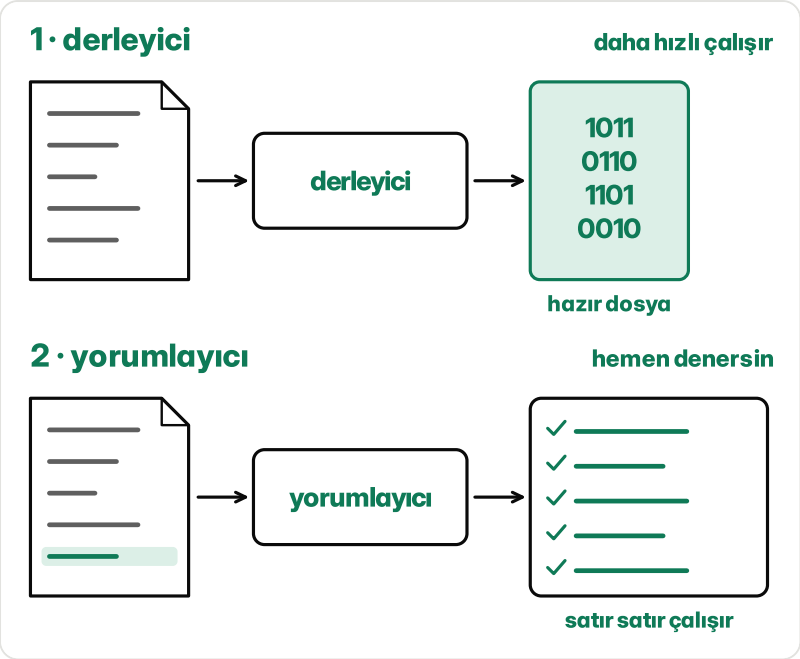
1 ile çarpmak sonucu değiştirmediği için 1'de durmak yeter. Aynı 6 satır 100 faktöriyeli de hesaplar; döngü, sayı 100'den 2'ye inene kadar yalnızca 99 kez döner.

Ama bilgisayar bu satırları olduğu gibi anlamaz; işlemcisi yalnızca birler ve sıfırlarla yazılmış kendi komutlarını çalıştırır. Bu yüzden algoritma önce bir **programlama dili** ile yazılır: kuralları kesin olan, insanın okuyabildiği ve makinenin diline çevrilebilen bir dil.

Çeviriyi iki tür program yapar. Bir kitap düşün: ya bütün kitap önceden çevrilip basılır ya da bir çevirmen okurken cümle cümle çevirir.

- **Derleyici** (compiler) birincisini yapar: bütün programı bir kerede makinenin diline çevirir ve çalışmaya hazır bir dosya çıkarır.
- **Yorumlayıcı** (interpreter) ikincisini yapar: programı satır satır çevirir ve her satırı hemen çalıştırır.

Derlenen program her çalışmada yeniden çevrilmediği için daha hızlı çalışır; yorumlayıcıda ise yazdığını beklemeden hemen denersin.




**Şekil 3.9** Derleyici bütün programı çevirip hazır bir dosya çıkarıyor; yorumlayıcı programı satır satır çevirip çalıştırıyor.

Telefonundaki her uygulama da böyle doğdu: önce adım adım bir algoritma, sonra bir programlama dili, en sonda işlemcinin anladığı birler ve sıfırlar.

### Özet

- Algoritma, bir işi yapmak için sırayla izlenen adımlardır; bilgisayar yalnızca ona söylenen adımları uygular.
- İyi bir algoritma açıktır, sonludur, girdisi ve çıktısı bellidir.
- Problem çözme dört aşamadan geçer: anlama, tasarım, kodlama, deneme.
- Akış diyagramı adımları şekillerle, sözde kod yazıyla anlatır; karar kutusu ve "eğer" satırı evet ya da hayıra göre yol ayırır.
- Döngü, bir koşul doğru oldukça aynı satırları yeniden çalıştırır; 6 satırlık faktöriyel algoritması 100! için 99 kez döner.
- Programlama diliyle yazılan programı derleyici bir kerede, yorumlayıcı satır satır makinenin diline çevirir.

## Sorular

**1** **Ortalama programına 50, 80 ve 95 notlarını verip sonucun 75 çıkıp çıkmadığına bakmak, problem çözmenin hangi aşamasıdır?**

**A** Anlama
**B** Tasarım
**C** Kodlama
**D** Deneme

**2** **Ortalama algoritmasında "toplamı uygun bir sayıya böl" adımının sorunu nedir?**

**A** Algoritmayı sonsuza kadar döndürür.

**B** Algoritmanın girdisini belirsiz bırakır.

**C** Adım açık değildir; bilgisayar hangi sayıya böleceğini bilemez.

**D** Sonucun yazılmasını engeller.

**3** **Ortalama algoritmasına bölmeden sonra "başa dön" adımı eklenirse ne olur?**

**A** Bilgisayar sonsuza kadar toplayıp böler, sonuç hiç yazılmaz.

**B** Ortalama iki kez yazılır.

**C** Ortalama daha doğru hesaplanır.

**D** Bilgisayar bu adımı atlayıp sonucu yazar.

**4** **Akış diyagramında içinde "a > b ?" yazan eşkenar dörtgen neyi gösterir?**

**A** Giriş ve çıkış

**B** Karar

**C** Başlangıç ve bitiş

**D** İşlem

**5** **İki sayının büyüğünü bulan sözde kod a = 3, b = 8 ile çalıştırılıyor. 4. satır (büyük ← a) ile 6. satırdan (büyük ← b) hangisi çalışır? Satır numarasını yaz.**

Cevap: _____

**6** **Faktöriyel algoritmasında sonuç kutusuna başta neden 1 konur?**

**A** Okunan sayı en az 1 olduğu için.

**B** Döngü yalnızca 1 kez döneceği için.

**C** 0 konursa döngü hiç çalışmayacağı için.

**D** 1 ile çarpmak sonucu değiştirmediği için.

**7** **Faktöriyel algoritması sayı 6 ile çalıştırılırsa döngü kaç kez döner?**

Cevap: _____

**8** **Derleyici ile yorumlayıcı arasındaki fark hangisinde doğru verilmiştir?**

**A** Derleyici programı satır satır çevirip hemen çalıştırır; yorumlayıcı bütün programı bir kerede çevirir.

**B** Derleyici sözde kodu, yorumlayıcı akış diyagramını makinenin diline çevirir.

**C** Derleyici bütün programı bir kerede çevirip hazır bir dosya çıkarır; yorumlayıcı programı satır satır çevirip çalıştırır.

**D** İkisi de programı her çalıştırmada baştan sona yeniden çevirir.

# 4 Girdi-çıktı ve koşullar

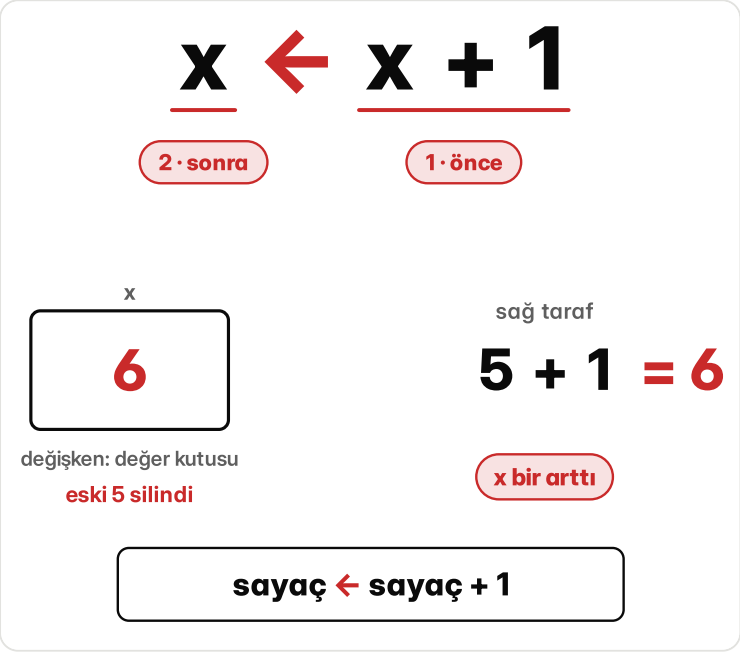
Bir program veriyi alır, saklar, işler ve sonucu gösterir; yol üstünde de kararlar verir. Bu bölümde değişkenleri, atamayı, girdi ve çıktıyı, aritmetik işlemleri ve programın koşullara göre nasıl karar verdiğini öğreneceksin.

## x = x + 1

videoda 24:31

Matematikte x = x + 1 satırı yanlıştır, çünkü hiçbir sayı kendisinin bir fazlasına eşit olamaz: x 5 ise 5 = 6 demek olur. Oysa programlarda en sık yazılan satırlardan biridir: sayaç = sayaç + 1, puan = puan + 10. Neden?

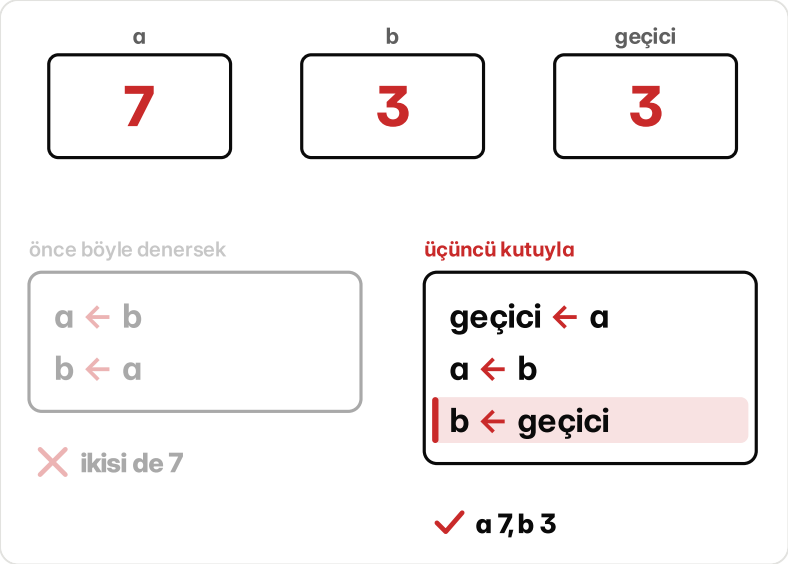
Çünkü buradaki eşittir bir eşitlik değil, bir emirdir; bu emre **atama** denir. Programlama dilleri onu çoğunlukla = ile yazar; sözde kodda eşitlikle karışmasın diye okla yazarız: x ← x + 1. Burada x, değer saklayan bir kutudur; böyle kutulara **değişken** denir. Ok, sağdaki değeri soldaki kutuya koy der. Bilgisayar önce sağ tarafı hesaplar, sonra sonucu kutuya koyar. x 5 iken sağ taraf 5 + 1 = 6 eder; kutuya 6 girer, eski 5 silinir. Bir kutu bir anda tek değer tutar; yani bu satır x'i 1 artırır. Sayaçlar da böyle sayar: sayaç ← sayaç + 1.




**Şekil 4.1** Atamada önce sağ taraf hesaplanıyor (5 + 1 = 6), sonra sonuç x kutusuna giriyor ve eski 5 siliniyor.

Eski değerin silinmesi yüzünden iki kutunun değerini değiştirmek göründüğü kadar kolay değildir. a 3, b 7 olsun. Önce a ← b yazarsak a 7 olur ve 3 kaybolur; sonra b ← a yazsak ikisi de 7 kalır. Çözüm üçüncü bir kutudur: önce a'yı geçici kutuya alırız, sonra a'ya b'yi, en son b'ye geçiciyi koyarız.

```
geçici ← a      geçici 3
a ← b           a 7
b ← geçici      b 3
```



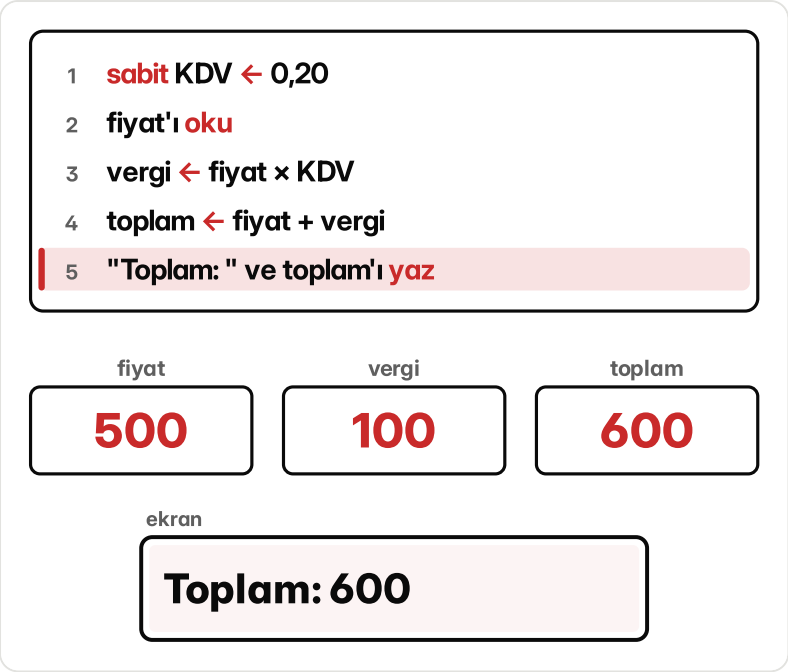

**Şekil 4.2** a ← b ve b ← a denemesi iki kutuyu da 7 yapıyor; geçici kutuyla yapılan 3 atamadan sonra a 7, b 3 oluyor.

Bazı değerler ise program boyunca hiç değişmez; bunlara **sabit** denir. Türkiye'de genel KDV oranı %20'dir. Bir mağaza programı bu oranı 100 ayrı yerde kullanıyorsa, her yere 0,20 yazmak yerine KDV adında bir sabit tanımlanır. Oran değişirse yalnız tanımın yapıldığı tek satır düzeltilir. Program çalışırken ise sabitin değeri değiştirilemez; KDV ← 0,25 gibi bir atama yapılamaz.

Şimdi KDV dahil fiyatı bulan küçük bir program yazalım. Fiyatı programın içine yazarsak (fiyat ← 500) program yalnız tek bir ürün için çalışır; bu yüzden fiyatı kullanıcıdan alırız. Oku komutu programı durdurur, kullanıcının yazacağı sayıyı bekler ve onu fiyat kutusuna koyar; bu, programın girdisidir. Sonra hesap yapılır. Yaz komutu sonucu ekrana çıkarır; bu da programın çıktısıdır. Yaz komutunda tırnak içindeki yazı olduğu gibi çıkar, tırnaksız bir ad ise kutusundaki değerle çıkar.

```
1   sabit KDV ← 0,20
2   fiyat'ı oku
3   vergi ← fiyat × KDV
4   toplam ← fiyat + vergi
5   "Toplam: " ve toplam'ı yaz
```

Kullanıcı 500 yazarsa vergi 500 × 0,20 = 100, toplam 500 + 100 = 600 olur ve ekranda "Toplam: 600" yazar.




**Şekil 4.3** 5 satırlık KDV programı fiyat 500 ile çalışıyor: vergi 100, toplam 600 ve ekranda "Toplam: 600".

Hesap makinesinden bankamatiğe kadar her program bu üç adımla çalışır: girdi alır, kutularda işler, sonucu yazar. Hesap makinesinde girdi tuşlardan gelir, işlem hesaptır, çıktı ekrandadır. Bankamatikte girdi kart ve şifredir, işlem bakiyenin hesabıdır, çıktı para ve fiştir.

## 135 dakika

videoda 26:25

Bir film 135 dakika sürüyor. Program bunu "2 saat 15 dakika" diye yazmak için hangi işlemi yapar?

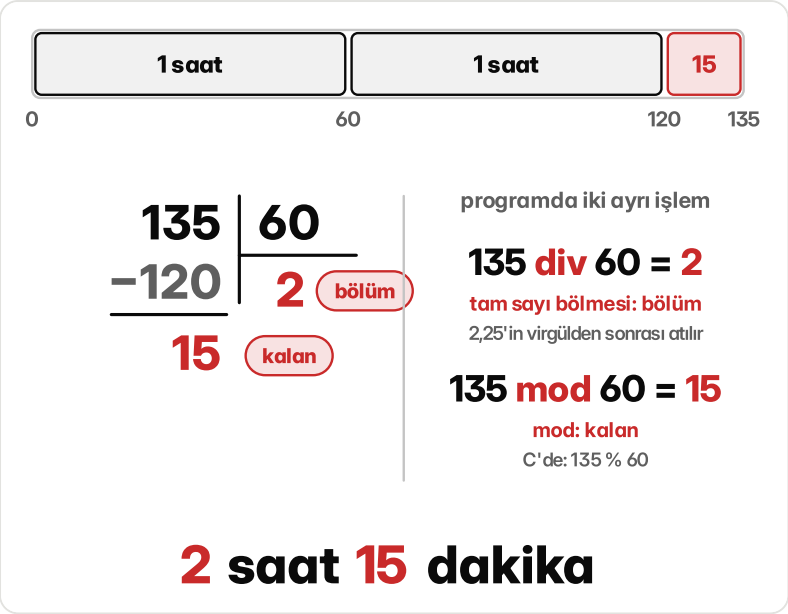
Toplama, çıkarma, çarpma ve bölme programlarda da vardır. Yalnız klavyede × ve ÷ tuşu olmadığı için programlarda çarpma yıldızla, bölme eğik çizgiyle yazılır:

```
+   toplama      -   çıkarma      *   çarpma      /   bölme
```

Bölmenin ise iki sonucu vardır. 135 dakikanın içine 60'ar dakikalık 2 tam saat sığar, geriye 15 dakika kalır. Bu ilkokuldaki bölmedir: bölüm 2, kalan 15.

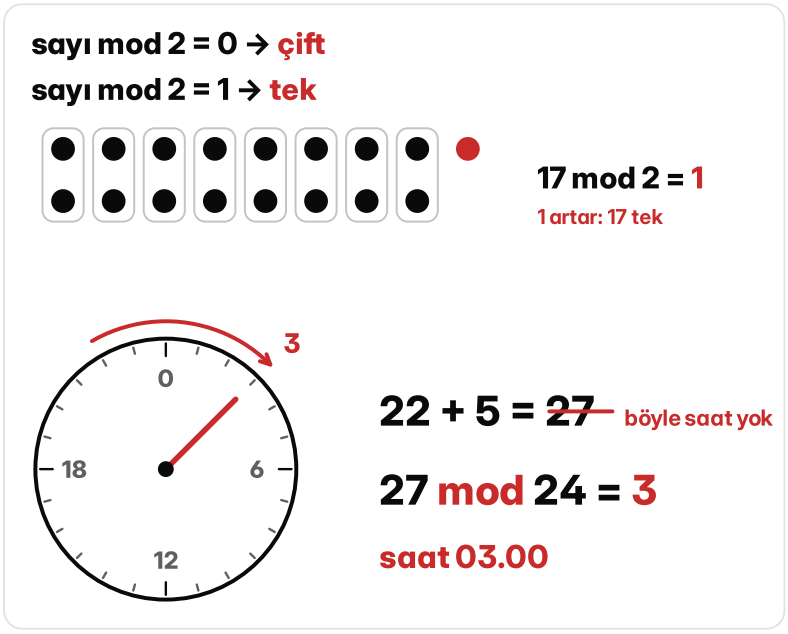
```
 135 | 60
-120 | 2      bölüm
  15          kalan
```

Programlarda bu iki sonuç ayrı işlemlerle bulunur. **Tam sayı bölmesi** yalnız bölümü verir, virgülden sonrasını atar: 135 / 60 = 2,25 eder, ama tam sayı bölmesinde sonuç 2'dir. Sözde kodda bunu 135 div 60 = 2 diye yazarız. Kalanı veren işleme **mod** denir: 135 mod 60 = 15 (C dilinde 135 % 60). Saat bölümden, dakika kalandan gelir: 2 saat 15 dakika. Aynı yolla 500 dakika, 500 div 60 = 8 ve 500 mod 60 = 20 olduğu için 8 saat 20 dakikadır.




**Şekil 4.4** 135 dakika 60 + 60 + 15 diye bölünüyor; ilkokul bölmesinde bölüm 2, kalan 15, programda 135 div 60 = 2 ve 135 mod 60 = 15.

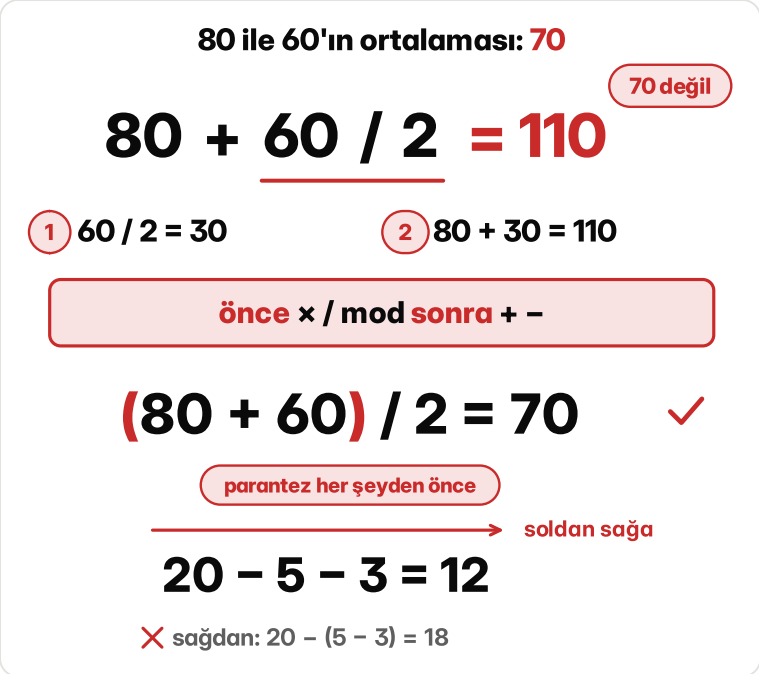
Mod birçok yerde işe yarar. Bir sayının 2 ile modu 0 ise sayı çifttir, 1 ise tektir; çünkü çift sayı 2'ye kalansız bölünür. 17 = 8 × 2 + 1 olduğu için 17 mod 2 = 1'dir; 17 tektir. Saat de modla döner: 22.00'den 5 saat sonra saat 27 olmaz. 22 + 5 = 27 ve 27 mod 24 = 3 olduğu için saat 03.00'tür.




**Şekil 4.5** 17 mod 2 = 1 olduğu için 17 tek; 24 saatlik kadranda 22'den 5 saat sonrası 27 mod 24 = 3, yani saat 03.00.

Şimdi 80 ile 60'ın ortalamasını bulalım. 80 + 60 / 2 yazarsak sonuç 70 değil, 110 çıkar. Çünkü programlarda da matematikteki **işlem önceliği** geçerlidir: çarpma, bölme ve mod, toplama ve çıkarmadan önce yapılır. Önce 60 / 2 = 30, sonra 80 + 30 = 110 hesaplanır. Önce toplamak için parantez kullanılır; parantez her şeyden önce yapılır: (80 + 60) / 2 = 140 / 2 = 70. Aynı düzeydeki işlemler soldan sağa yapılır:

```
20 - 5 - 3     =  15 - 3  =  12     soldan sağa
20 - (5 - 3)   =  20 - 2  =  18     sağdan başlasaydık
```




**Şekil 4.6** 80 + 60 / 2 = 110 çıkıyor, çünkü bölme önce yapılıyor; (80 + 60) / 2 = 70 ve soldan sağa 20 − 5 − 3 = 12.

Video oynatıcı da süreyi saniye olarak sayar. 135 saniyeyi ekranda göstermek için 135 div 60 = 2 ile dakikayı, 135 mod 60 = 15 ile saniyeyi bulur ve 2:15 yazar.

## İndirim var mı?

videoda 27:58

Bir sinema 12 yaşından küçüklere ve 65 yaşından büyüklere indirim yapıyor. Kasadaki program, kime indirim yapacağına nasıl karar verir?

Önce bir soru sorar: yaş 12'den küçük mü? Buna **karşılaştırma** denir. Karşılaştırmanın sonucu bir sayı değil, doğru ya da yanlıştır. 6 karşılaştırma vardır:

```
<   küçük               >   büyük
≤   küçük ya da eşit    ≥   büyük ya da eşit
=   eşit                ≠   eşit değil
```

Yaş 8 ise 8 < 12 sorusunun cevabı doğrudur.

İndirimin iki koşulu olduğu için iki soru mantıksal işlemlerle birleştirilir. VEYA, sorulardan en az biri doğruysa doğru verir: yaş < 12 VEYA yaş > 65. Günlük dilde "ve" deriz, ama burada VEYA gerekir; çünkü kimse hem 12'den küçük hem 65'ten büyük olamaz. VE ise ancak iki soru da doğruysa doğru verir: tam bilet için yaş ≥ 12 VE yaş ≤ 65 olmalıdır. DEĞİL cevabı tersine çevirir: doğruyu yanlış, yanlışı doğru yapar. İki sorunun (A ve B) 4 olası cevabını yan yana yazınca **doğruluk tablosu** çıkar (D doğru, Y yanlış):

```
A   B   A VE B   A VEYA B   DEĞİL A
D   D     D         D          Y
D   Y     Y         D          Y
Y   D     Y         D          D
Y   Y     Y         Y          D
```

VE yalnız ikisi de doğruyken doğrudur; VEYA yalnız ikisi de yanlışken yanlıştır.

Cevaba göre ne yapılacağını eğer ve değilse söyler: koşul doğruysa eğerin satırı, yanlışsa değilsenin satırı çalışır. Eğer yaş < 12 VEYA yaş > 65 ise bilet 60 lira, değilse 100 liradır. 70 yaşında biri gelirse ilk soru yanlış, ikincisi doğrudur; VEYA doğru verir ve bilet 60 lira olur.




**Şekil 4.7** Eğer-değilse kodu yaş 70 ile çalışıyor: 70 < 12 yanlış, 70 > 65 doğru, VEYA doğru veriyor ve bilet 60 lira.

Peki çocuğa 50, 65 yaş üstüne 60 lira olsaydı? O zaman koşullar zincir gibi dizilir; buna **eğer-değilse zinciri** denir: eğer yaş < 12 ise bilet 50, değilse eğer yaş > 65 ise 60, değilse 100. Sorular yukarıdan aşağı sorulur; ilk doğru cevabın satırı çalışır, gerisi atlanır. 8 yaşında biri için ilk soru doğrudur: bilet 50 lira olur, öteki sorular hiç sorulmaz.




**Şekil 4.8** 3 fiyatlı eğer-değilse zinciri yaş 8 ile çalışıyor: ilk soru doğru, bilet 50 lira, kalan satırlar atlanıyor.

Bazı sorular ise ancak başka bir sorunun cevabı evetse sorulur; buna **iç içe koşul** denir. Salonda boş koltuk yoksa yaşı sormanın anlamı yoktur; bu yüzden yaş sorusu, koltuğu soran eğerin içine yazılır (koltuk, boş koltuk sayısıdır):

```
eğer koltuk > 0 ise
    eğer yaş < 12 VEYA yaş > 65 ise
        bilet ← 60
    değilse
        bilet ← 100
değilse
    "Salon dolu" yaz
```

Koltuk 0 ise ekrana "Salon dolu" yazılır ve yaş hiç sorulmaz.

Kasadaki ekranda üç tuş olsun: 1 bilet, 2 mısır, 3 iptal. Bunu zincirle yazsak aynı soruyu üç kez sorardık: tuş 1 mi, 2 mi, 3 mü? Tek bir değerin hangi sayı olduğuna bakılacaksa **çoklu seçim** yazılır: her değerin kendi satırı vardır, değer hangisiyse o satır çalışır; hiçbirine uymazsa diğer adlı satır çalışır.




**Şekil 4.9** Kasa ekranındaki 1, 2 ve 3 tuşları için çoklu seçim: her durumun kendi satırı var, 1, 2 ve 3 dışındaki tuşlar diğer satırına gidiyor.

Bir bilet uygulamasında yaşını girince fiyatın değişmesi de arkada çalışan böyle eğer satırlarının işidir.

### Özet

- Atamada önce sağ taraf hesaplanır, sonuç soldaki değişkene konur ve eski değer silinir; x ← x + 1 x'i 1 artırır.
- Sabit program boyunca değişmez; oku girdiyi alır, yaz çıktıyı ekrana çıkarır ve her program girdi, işlem, çıktı adımlarıyla çalışır.
- Bölmenin iki sonucu ayrı işlemlerle bulunur: tam sayı bölmesi bölümü, mod kalanı verir.
- Çarpma, bölme ve mod toplamadan ve çıkarmadan önce, parantez hepsinden önce yapılır; aynı düzeydeki işlemler soldan sağa yapılır.
- Karşılaştırmanın sonucu doğru ya da yanlıştır; VE yalnız ikisi doğruyken doğru, VEYA yalnız ikisi yanlışken yanlış verir, DEĞİL cevabı ters çevirir.
- Eğer-değilse zincirinde ilk doğru koşulun satırı çalışır, iç içe koşulda içteki soru ancak dıştaki doğruysa sorulur, tek değere bakılırken çoklu seçim yazılır.

## Sorular

**1** **Sözde koddaki x ← x + 1 satırı ne yapar?**

**A** x'in kendisinin 1 fazlasına eşit olup olmadığını sorar.

**B** x'teki değeri 1 artırır.

**C** x kutusuna 1 değerini koyar.

**D** x'e 1 ekler, eski değeri de kutuda yanında saklar.

**2** **a 3, b 7 iken önce a ← b, sonra b ← a çalışırsa a ve b ne olur?**

**A** a 7, b 3

**B** a 3, b 7

**C** a 7, b 7

**D** a 3, b 3

**3** **A doğru, B yanlış ise hangisi doğrudur?**

**A** A VEYA B doğrudur.

**B** A VE B doğrudur.

**C** DEĞİL A doğrudur.

**D** A VEYA B yanlıştır.

**4** **Kasadaki çoklu seçimde 1 bilet, 2 mısır ve 3 iptal satırları ile bir diğer satırı var. Kullanıcı 4'e basarsa ne olur?**

**A** 3 iptal satırı çalışır, çünkü 4'e en yakın değer 3'tür.

**B** 1, 2 ve 3 satırları sırayla çalışır.

**C** Diğer satırı çalışır.

**D** Program 1 bilet satırından başlar ve bilet satar.

**5** **Bir şarkı 200 saniye sürüyor. Oynatıcı bu süreyi dakika:saniye biçiminde nasıl yazar?**

Cevap: _____

**6** **Sözde kodda 10 + 6 / 2 × 3 işleminin sonucu kaçtır?**

Cevap: _____

**7** **Bir programda eğer koltuk > 0 ise yaş sorulup bilet hesaplanıyor, değilse "Salon dolu" yazılıyor. Koltuk 0 ise ne olur?**

**A** Önce yaş sorulur, sonra "Salon dolu" yazılır.

**B** Ekrana "Salon dolu" yazılır, yaş hiç sorulmaz.

**C** Yaş sorulur ve bilet 100 lira olur.

**D** Yaş sorulmaz ama bilet 100 lira olarak hesaplanır.

**8** **Zincir şöyle: eğer yaş < 12 ise bilet ← 50, değilse eğer yaş > 65 ise bilet ← 60, değilse bilet ← 100. Hangi yaştaki biri 60 lira öder?**

**A** 8

**B** 12

**C** 65

**D** 66

# 5 Döngüler

Bu bölümde bir işi tekrar tekrar yaptıran döngüleri öğreneceksin: tur sayısını belirleyen üç parçayı, koşulu başta ve sonda soran döngüleri, döngüden erken çıkmayı ve iç içe döngüleri. Örnekler ekrana merhaba yazmak, bir şifre ekranı ve asal sayı bulmak.

## Kaç kez merhaba?

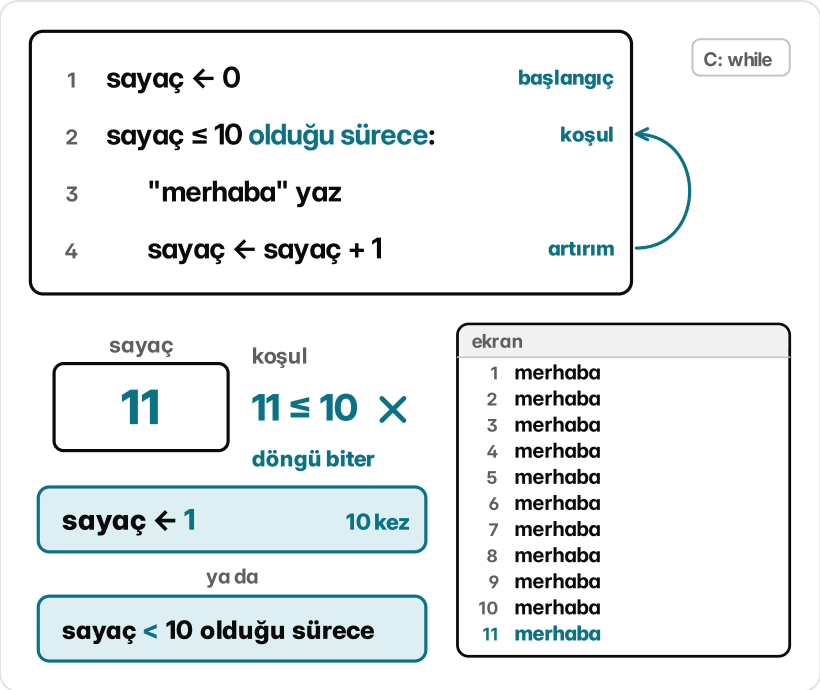
videoda 30:19

Şu 4 satırlık sözde kod ekrana kaç kez merhaba yazar: 10 mu, 11 mi?

```
sayaç ← 0
sayaç ≤ 10 olduğu sürece:
    "merhaba" yaz
    sayaç ← sayaç + 1
```

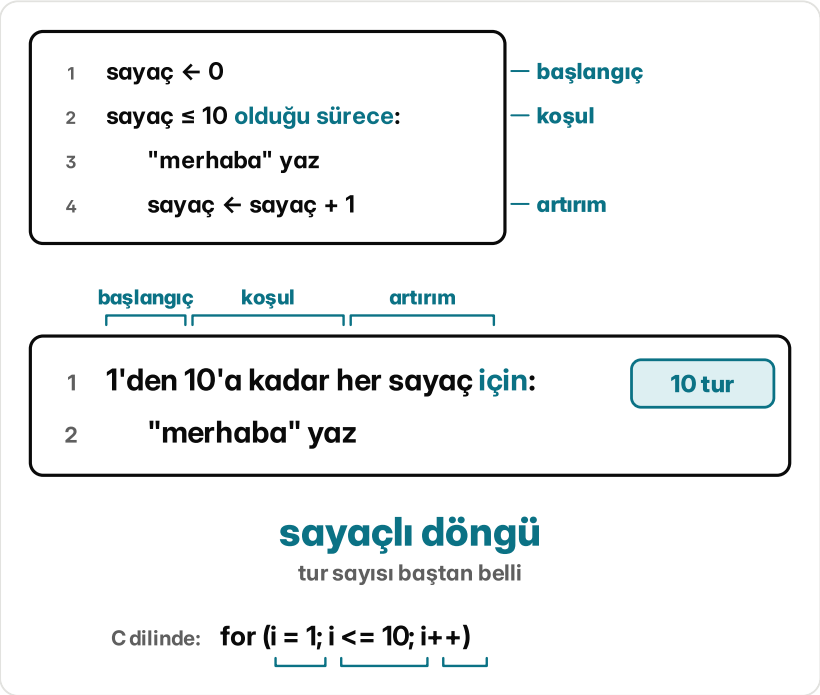
Aynı işi 100 kez alt alta yazmak yerine bir kez yazıp bir **döngü** içine koyarız. Döngü, koşulu doğru oldukça aynı satırları yeniden çalıştırır; kaç tur döneceğini ise üç parça belirler. **Başlangıç**, sayacın ilk değeridir; burada 0. **Koşul**, her turdan önce sorulan sorudur: sayaç en fazla 10 mu? **Artırım**, her turun sonunda sayacı 1 artıran satırdır; sayaç artmasaydı koşulun cevabı hiç değişmezdi.

Kodu satır satır çalıştıralım. Sayaç 0, koşul doğru: 1. merhaba yazılır ve sayaç 1 olur. Sayaç 1, koşul doğru: 2. merhaba yazılır, sayaç 2 olur. Bu böyle sürer. Sayaç 10 olduğunda da koşul doğrudur, çünkü 10, 10'a eşittir: 11. merhaba yazılır. Sayaç 11 olunca koşul yanlış olur ve döngü biter. Cevap 11'dir, çünkü 0'dan 10'a kadar 11 sayı vardır. 10 kez yazdırmak için ya sayaç 1'den başlamalı ya da koşul "sayaç < 10" olmalıdır; ikinci durumda sayaç 0'dan 9'a kadar 10 değer alır.




**Şekil 5.1** Sayaç 11 olunca 11 ≤ 10 koşulu yanlış olur ve döngü biter; ekranda 11 merhaba vardır.

Bu üç parça o kadar sık kullanılır ki tek satıra sığdırılır: "1'den 10'a kadar her sayaç için". Sayaç 1'den başlar, 10'a kadar gider ve her turda 1 artar; döngünün kaç tur döneceği ilk bakışta okunur. Tur sayısı baştan belli olan bu döngüye **sayaçlı döngü** denir. Programlama dillerinde de bu satır aynı üç parçayı aynı sırayla taşır; C dilinde for (i = 1; i <= 10; i++) diye yazılır.



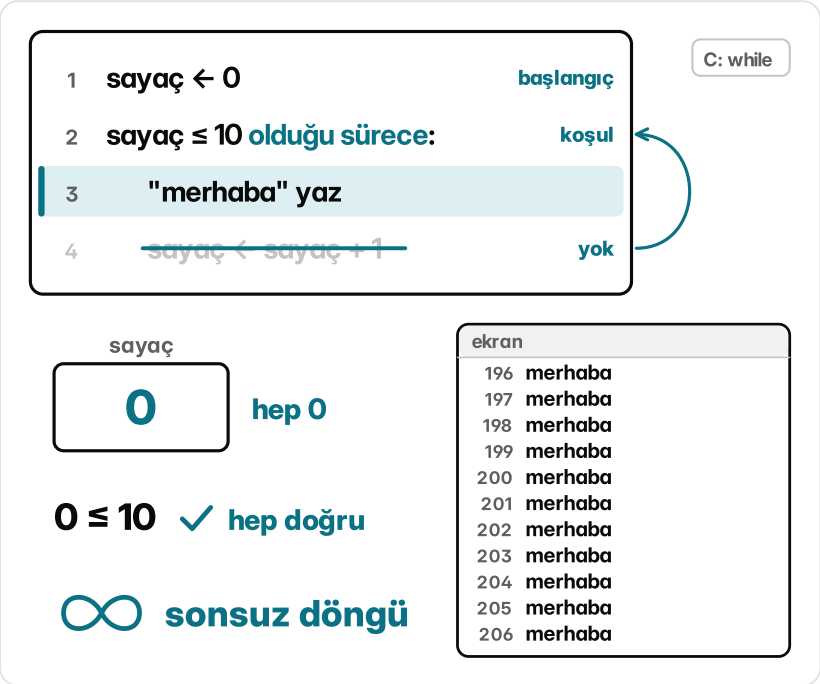

**Şekil 5.2** Sayaçlı döngü başlangıç, koşul ve artırımı tek satırda taşır: 10 tur; C dilindeki karşılığı for (i = 1; i <= 10; i++).

Sayaç yalnızca tur saymaz, her turda bir değer olarak da kullanılabilir. 1'den 100'e kadar sayıları toplayalım:

```
toplam ← 0
1'den 100'e kadar her sayaç için:
    toplam ← toplam + sayaç
toplamı yaz
```

Toplam 0'dan başlar ve her turda o anki sayaç eklenir: 1, 3, 6, 10 diye büyür. 100. turun sonunda toplam 5.050 olur.

Peki ilk koddaki artırım satırını unutursak? Sayaç hep 0 kalır, 0 ≤ 10 koşulu hep doğrudur ve döngü hiç bitmez. Buna **sonsuz döngü** denir: program durmadan merhaba yazar, başka hiçbir işe geçemez. Bir uygulama donup yanıt vermediğinde, bazen içeride koşulu hiç yanlış olmayan böyle bir döngü dönüyordur.




**Şekil 5.3** Artırım satırı olmayınca sayaç hep 0 kalır, koşul hep doğru olur ve merhaba satırları durmadan akar.

## Şifre

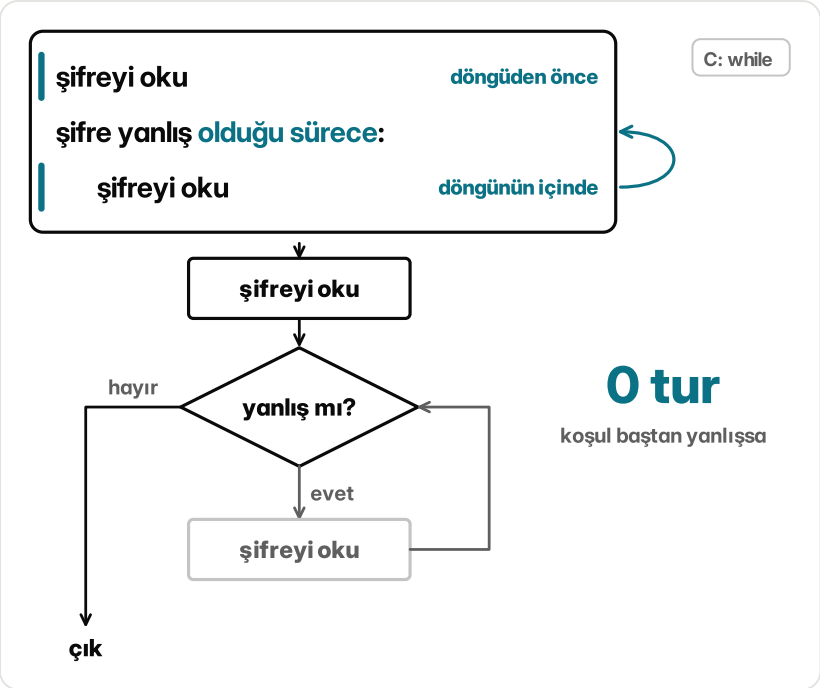
videoda 32:03

Bankamatik, şifreni yanlış girince yeniden sorar. Bu döngü kaç tur döneceğini önceden bilemez, çünkü tur sayısı baştan belli değildir. Döngü bir koşula bakar: şifre yanlış olduğu sürece sormayı sürdürür.

Koşulu en başta sorarsak bir sorun çıkar: daha hiç şifre okunmadan "şifre yanlış mı?" diye soramayız. Bu yüzden şifreyi bir kez döngüden önce, bir kez de döngünün içinde okuruz; aynı satır iki kez yazılır:

```
şifreyi oku
şifre yanlış olduğu sürece:
    şifreyi oku
```

Koşulu başta soran bu döngü (C dilinde while), koşul baştan yanlışsa hiç dönmez: ilk şifre doğruysa içine hiç girilmez, döngü 0 tur döner.

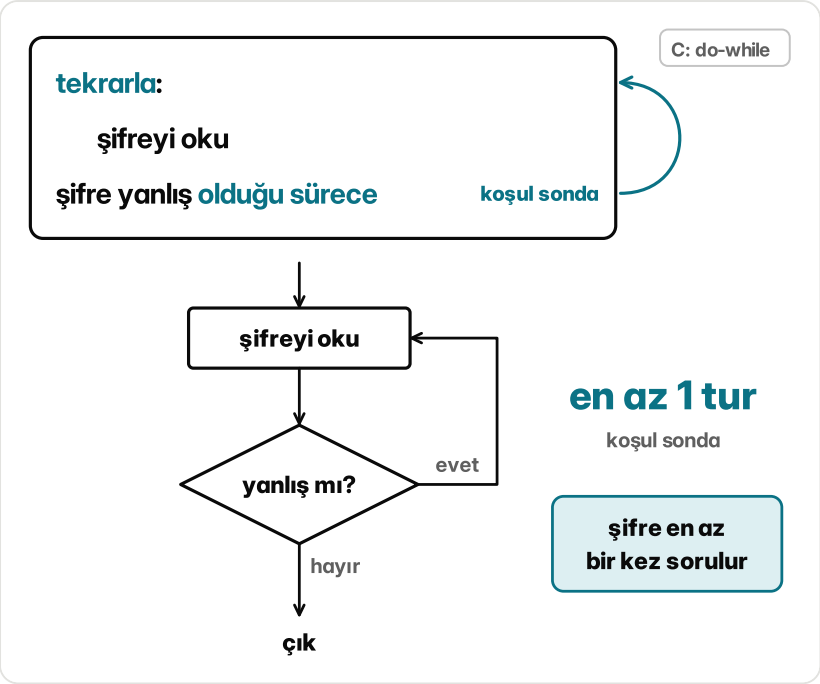



**Şekil 5.4** Koşulu başta soran döngüde şifre döngüden önce ve döngünün içinde okunur; koşul baştan yanlışsa döngü 0 tur döner.

İkinci tür döngü koşulu sonda sorar: önce şifreyi okur, sonra yanlış mı diye bakar, yanlışsa başa döner.

```
tekrarla:
    şifreyi oku
şifre yanlış olduğu sürece
```

Koşul sonda olduğu için içindeki satırlar en az bir kez çalışır (C dilinde do-while). Şifreyi en az bir kez sormak da zaten istediğimiz şeydir; üstelik okuma satırı yalnız bir kez yazılır.



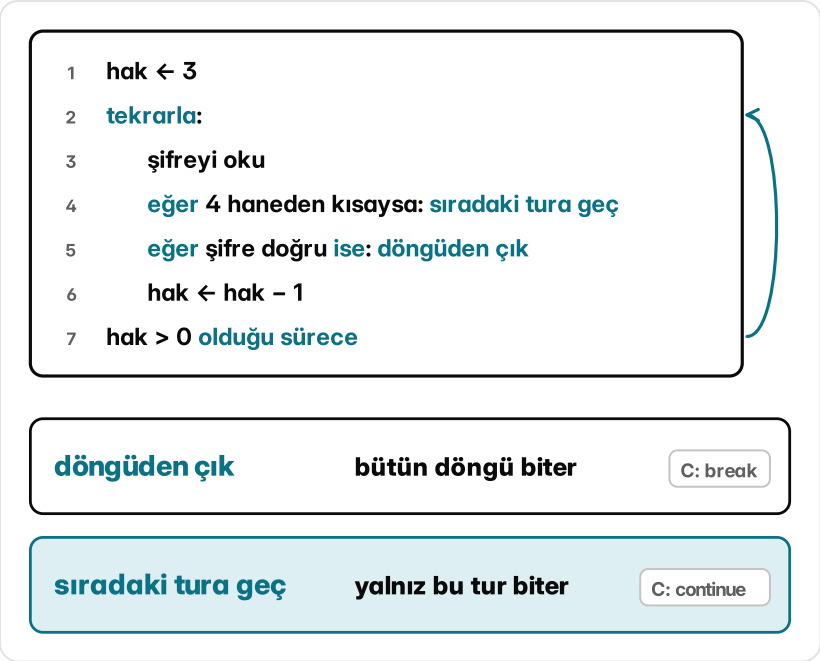

**Şekil 5.5** Koşulu sonda soran döngü önce şifreyi okur, sonra koşula bakar; bu yüzden en az 1 tur döner.

Şimdi kullanıcıya 3 hak verelim: hak sayacı 3'ten başlar, her yanlışta 1 azalır, döngü hak bitene kadar döner.

```
hak ← 3
tekrarla:
    şifreyi oku
    eğer 4 haneden kısaysa: sıradaki tura geç
    eğer şifre doğru ise: döngüden çık
    hak ← hak − 1
hak > 0 olduğu sürece
```

Diyelim ki ilk girişte yanlış, ikincide doğru girdin. İlk turda şifre yanlış olduğu için hak 2'ye iner. İkinci turda şifre doğrudur; **döngüden çık** komutu (C dilinde break) döngüyü o anda bitirir, kalan satırlar ve turlar atlanır.

Şimdi 4 hane yerine 3 hane girdiğini düşün; bu yanlış bir şifre değil, eksik tuşlamadır. **Sıradaki tura geç** komutu (C dilinde continue) turun kalan satırlarını atlar: hak azaltma satırına gelinmez, hak 3'te kalır. Ardından sondaki koşula bakılır; hak hâlâ 0'dan büyük olduğu için yeni tur başlar ve şifre yeniden sorulur. Kısacası döngüden çık komutu bütün döngüyü, sıradaki tura geç komutu yalnızca o turu bitirir.




**Şekil 5.6** Döngüden çık bütün döngüyü bitirir (C: break), sıradaki tura geç yalnız o turu bitirir (C: continue).

Bankamatik şifreni her yeniden sorduğunda, ekranın arkasında böyle bir döngü bir tur daha döner.

## 91 asal mı?

videoda 33:22

91 asal mı? Çift değil; rakamlarının toplamı 9 + 1 = 10 olduğu için 3'e bölünmüyor; 0 ya da 5 ile bitmediği için 5'e de bölünmüyor. Asal gibi duruyor. Ama bilgisayar tahmin etmez, dener.

**Asal sayı**, yalnızca 1'e ve kendisine tam bölünen sayıdır. Tanım bize bir yöntem verir: 2'den başlayıp her sayıyı bölen olarak deneriz ve 91'in o sayıya tam bölünüp bölünmediğine, yani bölümden kalanın 0 olup olmadığına bakarız. 2'ye bölümden kalan 1, 3'e 1, 4'e 3, 5'e 1, 6'ya 1'dir; hiçbiri tam bölmüyor. 7'de kalan 0'dır, çünkü 91 = 7 × 13. Demek ki 91 asal değil.

Bir bölen bulunduysa ötesini denemenin anlamı yoktur. Döngüden çık komutu döngüyü orada bitirir:

```
2'den 90'a kadar her bölen için:
    eğer 91, bölene tam bölünüyorsa:
        "asal değil" yaz
        döngüden çık
```

2'den 90'a kadar 89 olası bölen vardır, yani döngü en çok 89 tur dönebilir. Bölen 7'de bulunduğu için yalnızca 6 tur döner (2, 3, 4, 5, 6 ve 7); kalan 83 sayı hiç denenmez.




**Şekil 5.7** Bölen 7'de bulununca döngüden çıkılır; 89 olası turdan yalnızca 6'sı döner, kalan 83 sayı denenmez.

Bir döngünün içine başka bir döngü de konabilir; buna **iç içe döngü** denir. En bilinen örneği çarpım tablosudur:

```
1'den 10'a kadar her satır için:
    1'den 10'a kadar her sütun için:
        satır × sütun yaz
```

Dıştaki döngü satırları, içteki döngü sütunları dolaşır. Dıştaki sayaç 1 iken içteki 1'den 10'a gider: 1 × 1, 1 × 2, böylece 1 × 10'a kadar. Sonra dıştaki 2 olur ve içteki yeniden 1'den başlar. Dıştaki döngü bir tur atarken içteki baştan sona 10 tur döner: 10 satır, her satırda 10 çarpma, toplam 100 çarpma.



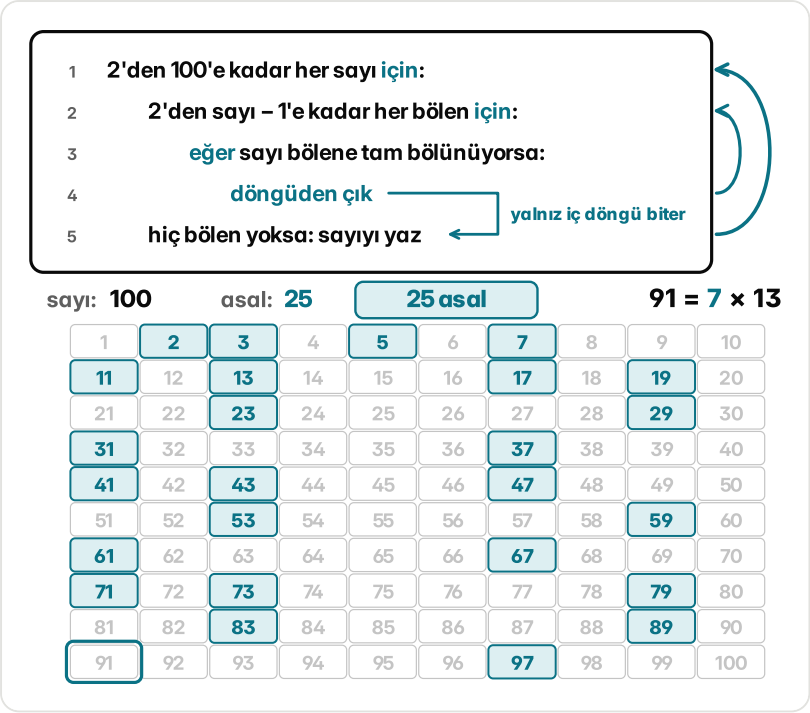

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| 2 | 2 | 4 | 6 | 8 | 10 | 12 | 14 | 16 | 18 | 20 |
| 3 | 3 | 6 | 9 | 12 | 15 | 18 | 21 | 24 | 27 | 30 |
| 4 | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 |
| 5 | 5 | 10 | 15 | 20 | 25 | 30 | 35 | 40 | 45 | 50 |
| 6 | 6 | 12 | 18 | 24 | 30 | 36 | 42 | 48 | 54 | 60 |
| 7 | 7 | 14 | 21 | 28 | 35 | 42 | 49 | 56 | 63 | 70 |
| 8 | 8 | 16 | 24 | 32 | 40 | 48 | 56 | 64 | 72 | 80 |
| 9 | 9 | 18 | 27 | 36 | 45 | 54 | 63 | 72 | 81 | 90 |
| 10 | 10 | 20 | 30 | 40 | 50 | 60 | 70 | 80 | 90 | 100 |

**Şekil 5.8** Dış döngü satırları, iç döngü sütunları dolaşır; 10 × 10 çarpım tablosu 100 çarpmayla dolar.

Asal testini de bir dış döngünün içine koyarsak 100'e kadar bütün asalları buluruz:

```
2'den 100'e kadar her sayı için:
    2'den sayı − 1'e kadar her bölen için:
        eğer sayı bölene tam bölünüyorsa:
            döngüden çık
    hiç bölen yoksa: sayıyı yaz
```

Dıştaki döngü 2'den 100'e kadar her sayıyı sırayla dener, içteki döngü o sayı için bölen arar. Bölen bulununca döngüden çık komutu yalnızca içteki döngüyü bitirir; dıştaki döngü sıradaki sayıya geçer. Hiç bölen bulunmayan sayı asaldır. 100'e kadar 25 asal çıkar; 91 listede yoktur, onu 7 eler.




**Şekil 5.9** 1'den 100'e kadar sayıların tablosunda asallar vurgulu; tarama bitince 25 asal sayılmış.

## Özet

- Döngü, koşulu doğru oldukça aynı satırları yeniden çalıştırır; tur sayısını başlangıç, koşul ve artırım belirler.
- Sayaç 0'dan başlayıp "sayaç ≤ 10" koşuluyla dönen döngü 11 tur döner; 10 tur için sayaç 1'den başlamalı ya da koşul "sayaç < 10" olmalıdır.
- Artırım unutulursa koşul hep doğru kalır ve döngü hiç bitmez: sonsuz döngü.
- Koşulu başta soran döngü hiç dönmeyebilir; koşulu sonda soran döngü en az bir kez döner.
- Döngüden çık bütün döngüyü, sıradaki tura geç yalnızca o turu bitirir; iç içe döngüde döngüden çık yalnız içteki döngüyü bitirir.
- 91 = 7 × 13 olduğu için asal değildir; 100'e kadar 25 asal sayı vardır.

## Sorular

**1** **"sayaç ← 0" ile başlayan ve "sayaç ≤ 10 olduğu sürece" merhaba yazan döngüde artırım satırı (sayaç ← sayaç + 1) unutulursa ne olur?**

- **A** Döngü hiç çalışmaz.
- **B** Döngü bir kez çalışıp biter.
- **C** Sayaç hep 0 kalır, koşul hep doğru olur ve döngü hiç bitmez.
- **D** Program hata verip hemen durur.

**2** **"toplam ← 0" ile başlayıp "1'den 10'a kadar her sayaç için: toplam ← toplam + sayaç" döngüsü bitince toplam kaç olur?**

Cevap:

**3** **Şu döngü ekrana kaç kez merhaba yazar? sayaç ← 1; sayaç < 10 olduğu sürece: "merhaba" yaz, sayaç ← sayaç + 1**

Cevap:

**4** **Bankamatik şifreyi en az bir kez sormalı ve şifre yanlış olduğu sürece yeniden sormalı. Şifre okuma satırını yalnız bir kez yazmak için hangi döngü uygundur?**

- **A** Koşulu başta soran döngü (C'de while)
- **B** Koşulu sonda soran döngü (C'de do-while)
- **C** 1'den 3'e kadar dönen sayaçlı döngü
- **D** İç içe iki döngü

**5** **3 haklı şifre döngüsünde kullanıcı 4 hane yerine 3 hane girer ve program "sıradaki tura geç" komutunu çalıştırır. Ne olur?**

- **A** Döngü tamamen biter.
- **B** Hak 1 azalır ve şifre yeniden sorulur.
- **C** Program şifreyi doğru sayar ve döngüden çıkar.
- **D** Turun kalan satırları atlanır; hak azalmaz ve şifre yeniden sorulur.

**6** **85'in asal olup olmadığını, 2'den başlayarak bölen deneyen ve ilk böleni bulunca döngüden çıkan bir döngüyle sınıyoruz. Döngü kaç tur döner?**

Cevap:

**7** **Dıştaki döngü 1'den 5'e, içteki döngü 1'den 10'a kadar dönüyor. İç döngünün içindeki çarpma satırı toplam kaç kez çalışır?**

**A** 50

**B** 15

**C** 10

**D** 5

**8** **100'e kadar asalları bulan programda dıştaki döngü sayıları, içteki döngü bölenleri dolaşıyor. İçteki döngüde bölen bulununca çalışan "döngüden çık" komutu ne yapar?**

**A** İki döngüyü birden bitirir; program durur.

**B** Yalnızca içteki döngüyü bitirir; dıştaki döngü sıradaki sayıya geçer.

**C** Yalnızca o turu atlar; içteki döngü sıradaki bölenle sürer.

**D** Yalnızca dıştaki döngüyü bitirir.

# 6 Temel veri tipleri

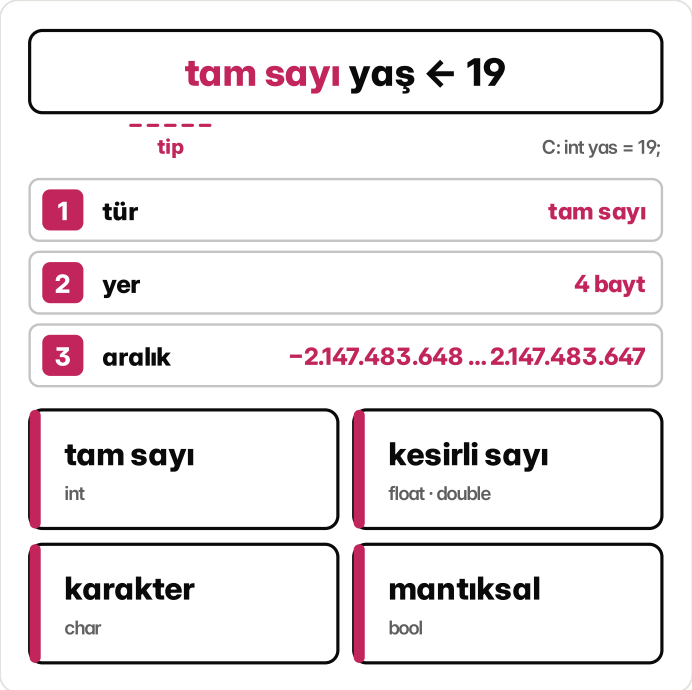
Bu bölümde bellekteki bitlere anlam veren veri tiplerini öğreneceksin: 4 temel tipi, bellekte kapladıkları yeri ve alabildikleri değerleri. Ardından tiplerin bir hesabın sonucunu nasıl değiştirdiğini, tip dönüşümünü ve tip taşmasını göreceksin.

## 65 mi, A mı?

videoda 34:41

Bellekte yan yana 8 bit duruyor: 01000001. Bunlar 65 sayısı da olabilir, A harfi de. Bilgisayar hangisi olduğunu bitlere bakarak bilemez; bunu ona verinin tipi söyler.

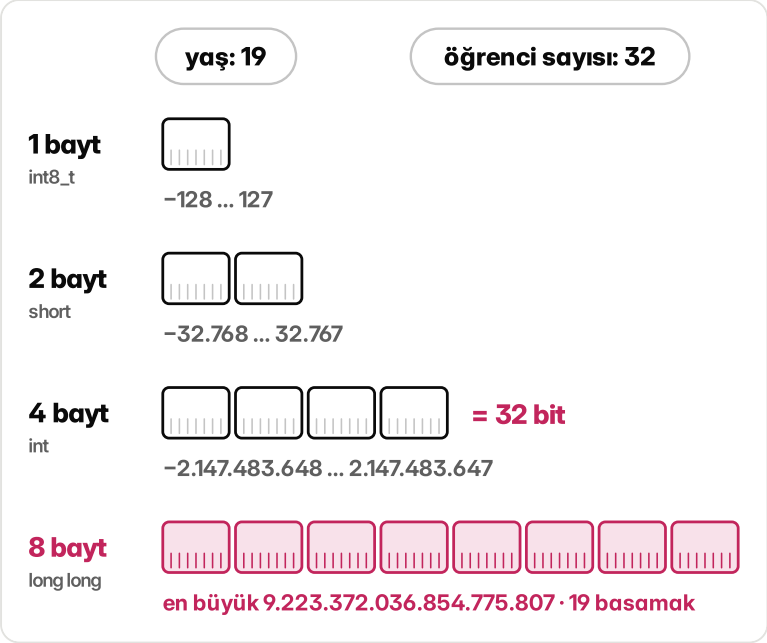
Program bellekte bir değişken açarken onun tipini de söyler. "tam sayı yaş ← 19" satırı, yaş adında bir tam sayı değişkeni açar ve içine 19 koyar (C dilinde int yas = 19;). **Veri tipi** 3 şeyi belirler: değerin türünü, bellekte kaç bayt yer kaplayacağını ve hangi değerleri alabileceğini. 4 temel tip vardır: tam sayı, kesirli sayı, karakter ve mantıksal.




**Şekil 6.1** Tip değerin türünü, bellekte kapladığı yeri ve aralığını belirler; altta 4 temel tip: tam sayı, kesirli sayı, karakter, mantıksal.

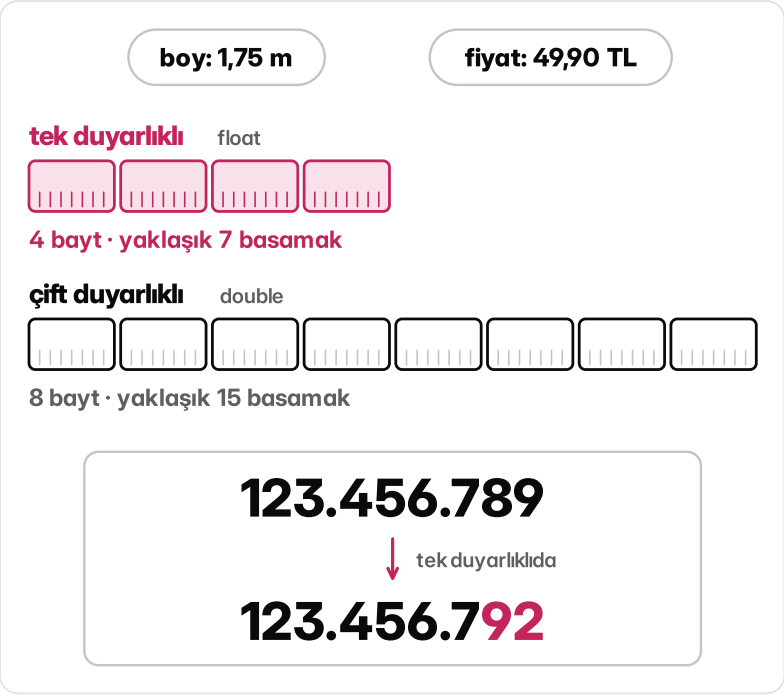
**Tam sayı** tipi, yaş ya da öğrenci sayısı gibi küsuratsız sayılar içindir. En çok kullanılanı 4 bayt, yani 32 bit yer kaplar ve yaklaşık −2 milyardan +2 milyara kadar değer alır. 1, 2 ve 8 baytlık türleri de vardır; 8 baytlık olan 19 basamaklı sayılara kadar gider:

```
1 bayt    −128 … 127
2 bayt    −32.768 … 32.767
4 bayt    −2.147.483.648 … 2.147.483.647
8 bayt    en büyük 9.223.372.036.854.775.807
```




**Şekil 6.2** 1, 2, 4 ve 8 baytlık tam sayıların değer aralıkları; 4 bayt 32 bittir.

**Kesirli sayı** tipi, boy (1,75 m) ya da fiyat (49,90 TL) gibi virgüllü sayılar içindir. Tek duyarlıklı olanı (float) 4 bayt kaplar ve yaklaşık 7 basamağı doğru tutar; çift duyarlıklı olanı (double) 8 bayt kaplar ve yaklaşık 15 basamağı doğru tutar. Bu yüzden 9 basamaklı bir sayıyı tek duyarlıklı tipe koyarsan son rakamları değişir: 123.456.789, tek duyarlıklı tipte 123.456.792 olarak saklanır.




**Şekil 6.3** Tek duyarlıklı tip 4 bayt ve yaklaşık 7 basamak, çift duyarlıklı tip 8 bayt ve yaklaşık 15 basamak tutar; 123.456.789 tek duyarlıklıda 123.456.792 olur.

**Karakter** tipi tek bir harf, rakam ya da işaret saklar ve 1 bayt yer kaplar. Bellekte harfin kendisi değil, kodu durur: büyük A'nın kodu 65'tir. Baştaki 8 bit de buydu: 01000001 = 64 + 1 = 65. Sayı olarak okunursa 65, karakter olarak okunursa A.

**Mantıksal** tip yalnızca 2 değer alır: doğru ya da yanlış. "Kapı kilitli mi?", "Öğrenci sınavı geçti mi?" gibi evet-hayır sorularının cevabını tutar. Bunun için tek bit yeter, ama genelde 1 bayt kaplar; çünkü bellekte adresi olan en küçük birim bayttır.

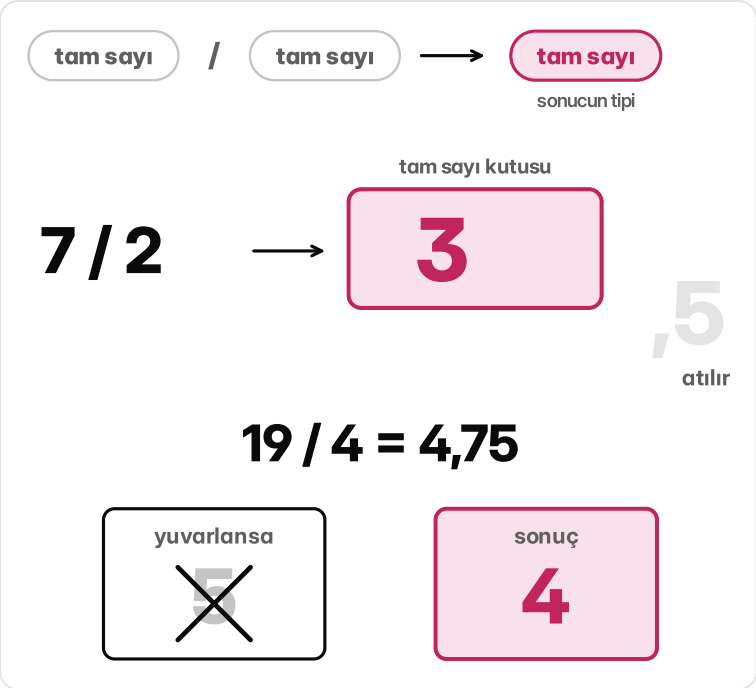
Peki neden her sayıya 8 bayt vermiyoruz? 1 milyon kişinin yaşını 8 baytlık tam sayılarla saklarsan 8 × 1.000.000 = 8.000.000 bayt, yani 8 megabayt harcarsın. Yaşa 1 bayt yeter, çünkü 1 baytlık tam sayı 127'ye kadar gider; toplam 1 megabayt olur. Bellekteki bitler kendi başına bir şey söylemez; onlara anlam ve yer veren, programcının seçtiği tiptir.

## 7 / 2 neden 3?

videoda 36:26

Hesap makinesinde 7 / 2 = 3,5 eder. C, C++, Java ve C# gibi birçok programlama dili ise 3 der. Neden? Çünkü sorduğun iki sayı da tam sayıydı.

İki tam sayıyla yapılan işlemin sonucu da tam sayı tipinde olur; sonucun tipini işlemdeki sayıların tipi belirler. 3,5 tam sayıya sığmaz; virgülden sonrası atılır, geriye 3 kalır. Yuvarlama da yapılmaz: 19 / 4 = 4,75 eder, ama sonuç 5 değil, 4 çıkar.




**Şekil 6.4** İki tam sayının bölümü tam sayıdır: 7 / 2'de ,5 atılır ve 3 kalır; 19 / 4 yuvarlanıp 5 olmaz, 4 çıkar.

Bu kural en çok ortalama hesabında can yakar:

```
tam sayı toplam ← 7
tam sayı adet ← 2
kesirli ortalama ← toplam / adet
yaz(ortalama)
```

Ortalamayı kesirli bir değişkene koysan bile ekrana 3,0 yazılır. Çünkü bölme önce yapılır: iki tam sayı bölünür, 3 çıkar; bu sonuç kesirli kutuya ancak sonra girer. Küsurat kutuya girmeden önce kaybolmuştur.

Sayılardan biri kesirliyse sonuç değişir. 7,0 / 2 yazarsan bilgisayar 2'yi kendiliğinden kesirliye, 2,0'a çevirir ve 3,5 bulur. Buna **kendiliğinden dönüşüm** denir. Karışık işlemde tam sayı kesirliye çevrilir, tersi olmaz; çünkü küsuratı tutabilen tip kesirli olandır.

Sayılar değişkenlerde duruyorsa sonlarına virgül ekleyemezsin; dönüşümü sen istersin. Toplamı kesirliye çevirir, sonra adede bölersin:

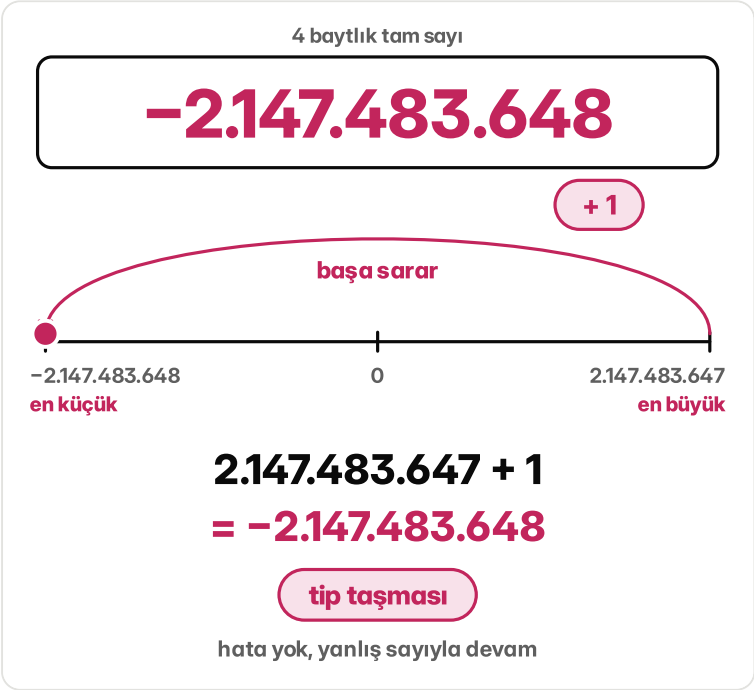
```
ortalama ← kesirli(toplam) / adet        sonuç 3,5
```

Buna **açık dönüşüm** denir; C dilinde (double) toplam / adet diye yazılır. Kesirliden tam sayıya çevirirken de küsurat atılır: tam sayı(3,9) sonucu 4 değil, 3 olur.




**Şekil 6.5** Toplamı kesirliye çevirmek ortalamayı 3,5 yapar; kesirliden tam sayıya çevirmede tam sayı(3,9) 4 değil, 3 olur.

Her tipin bir de sınırı vardır. 4 baytlık tam sayının en büyük değeri 2.147.483.647'dir. Buna 1 eklersen sayı büyümez; dolan bir sayaç gibi başa sarar. Tam sayıda baş en küçük değerdir: −2.147.483.648. Buna **tip taşması** denir; program çoğu zaman hata vermez, yanlış sayıyla yoluna devam eder.




**Şekil 6.6** 2.147.483.647 + 1, sayı doğrusunun en büyük ucundan en küçük ucuna, −2.147.483.648'e sarar.

1996'da Avrupa'nın yeni roketi Ariane 5, ilk uçuşunda kalkıştan yaklaşık 40 saniye sonra parçalandı. Yazılımı, yatay hızla ilgili 8 baytlık bir kesirli sayıyı 2 baytlık tam sayıya çeviriyordu; 2 bayta en çok 32.767 sığar. Kod, önceki roket Ariane 4'ten alınmıştı. Yeni roketin yatay hızı çok

daha büyüktü; sayı sınırı aştı, dönüşüm hata verdi ve yön ölçen sistem kapandı. Roketi parçalayan, 2 bayta sığmayan tek bir sayıydı.

**Özet**

- Bellekteki bitlerin anlamını veri tipi belirler: 01000001 sayı olarak 65, karakter olarak A'dır.
- Tip, değerin türünü, kapladığı yeri ve alabileceği değerleri belirler; 4 temel tip tam sayı, kesirli sayı, karakter ve mantıksaldır.
- En yaygın tam sayı 4 bayttır ve −2.147.483.648 ile 2.147.483.647 arasında değer alır.
- İki tam sayının bölümü tam sayıdır; küsurat yuvarlanmadan atılır: 7 / 2 = 3.
- Karışık işlemde tam sayı kendiliğinden kesirliye çevrilir; değişkenlerde dönüşümü açıkça istemek gerekir.
- Sınırı aşan tam sayı başa sarar; bu tip taşması çoğu zaman hata vermeden yanlış sonuç üretir.

## Sorular

**1** **Bellekte 01000001 bitleri duruyor. Bunun 65 sayısı mı yoksa A harfi mi olduğunu ne belirler?**

**A** Bitlerin kaç tane olduğu

**B** Bitlerin bellekteki adresi

**C** Verinin tipi

**D** En soldaki bitin 0 olması

**2** **Mantıksal tip yalnızca doğru ya da yanlış değerini alır ve bunun için tek bit yeter. Yine de genelde neden 1 bayt kaplar?**

**A** Doğru ve yanlış değerleri ayrı ayrı 4'er bit tuttuğu için

**B** Bellekte adresi olan en küçük birim bayt olduğu için

**C** Mantıksal değer bir harfin kodu gibi saklandığı için

**D** Mantıksal değer 0 ile 255 arasında bir sayı olduğu için

**3** **1 milyon kişinin yaşını 4 baytlık tam sayılarla saklarsan kaç megabayt yer harcarsın?**

Cevap:

**4** **İki tam sayıyla yapılan 23 / 4 işleminin sonucu C ya da Java gibi bir dilde kaç çıkar?**

Cevap:

**5** **4 baytlık bir tam sayı, en büyük değeri olan 2.147.483.647'yi tutuyor. Buna 1 eklenirse çoğu zaman ne olur?**

**A** Sayı başa sarar ve −2.147.483.648 olur.

**B** Program hata verip durur.

**C** Sayı 2.147.483.648 olur.

**D** Sayı en büyük değerde kalır.

**6** **1996'da Ariane 5 roketinin ilk uçuşunda parçalanmasına hangi yazılım sorunu yol açtı?**

**A** Tam sayı bölmesinde küsuratın atılması

**B** Tek duyarlıklı kesirli tipin 9 basamaklı bir sayının son rakamlarını değiştirmesi

**C** 8 baytlık kesirli bir sayının 2 baytlık tam sayıya çevrilirken sınırı aşması

**D** Bir harfin kodunun sayı olarak okunması

**7** **tam sayı toplam ← 9; tam sayı adet ← 2; kesirli ortalama ← toplam / adet. Bu satırlardan sonra ortalama kaçtır?**

**A** 4,5

**B** 5,0

**C** 0,5

**D** 4,0

**8** **toplam = 7 ve adet = 2 değişkenlerinin ikisi de tam sayı. Hangi yazım ortalamayı 3,5 bulur?**

**A** toplam / adet

**B** kesirli(toplam) / adet

**C** tam sayı(toplam / adet)

**D** tam sayı(kesirli(toplam) / adet)

# 7 Diziler ve karakter dizileri

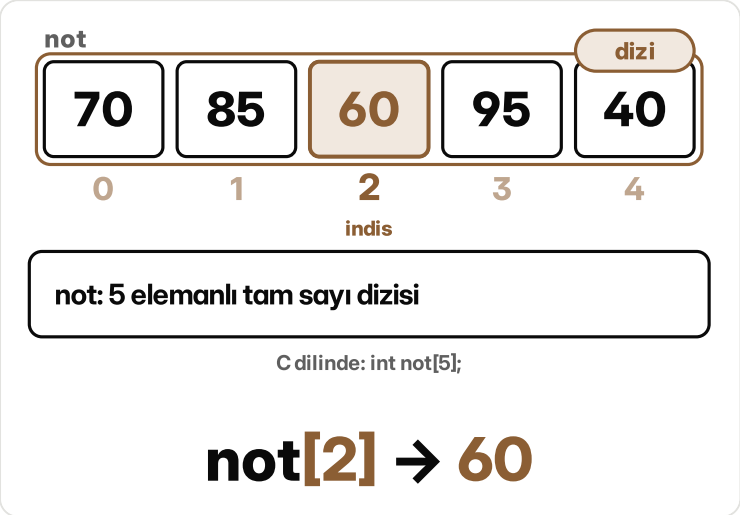
Bu bölümde aynı türden değerleri tek bir adla tutan dizileri öğreneceksin: elemanların bellekte nasıl yerleştiğini, saymanın neden 0'dan başladığını, dizide dolaşmayı, iki boyutlu dizileri ve kelimeleri tutan karakter dizilerini.

## Neden 0?

videoda 38:17

Bir programda 5 öğrencinin notu yan yana tutulur: 70, 85, 60, 95, 40. İlk nota not[1] değil, not[0] denir. Bilgisayarda sayma neden 0'dan başlar?

5 notu 5 ayrı değişkende tutmak yerine hepsini tek bir adla tutarız. Aynı türden değerleri böyle sıra sıra tutan yapıya **dizi** denir. Not adında 5 elemanlı bir tam sayı dizisi tanımlarız (C dilinde int not[5];) ve her elemana köşeli parantez içindeki sıra numarasıyla ulaşırız. Bu numaraya **indis** denir: not[2] dersek 60 gelir.



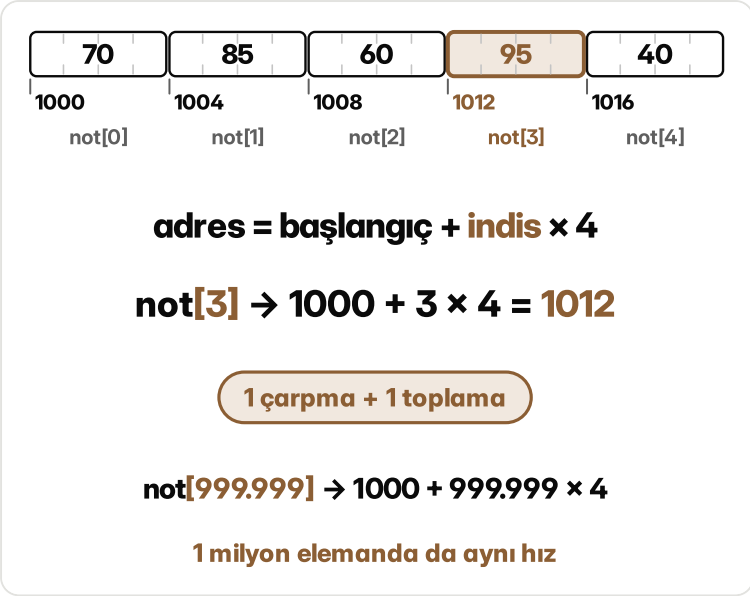

**Şekil 7.1** 5 not, not adlı tek bir dizide; indisler 0'dan 4'e gider ve not[2] → 60.

Bellek, 1 baytlık küçük gözlerden oluşur ve her gözün bir numarası vardır; bu numaraya **adres** denir. Dizinin elemanları bu gözlere arka arkaya, boşluk bırakmadan yerleşir. Bir tam sayı 4 göz tutsun: dizi 1000 adresinden başlıyorsa ilk not 1000, ikincisi 1004, üçüncüsü 1008 adresindedir. 5 notluk dizi 1000'den 1019'a kadar 20 göz kaplar.

Bu yüzden bilgisayar bir elemanı aramaz, adresini hesaplar:

```
adres = başlangıç adresi + indis × 4
not[3]  →  1000 + 3 × 4 = 1012
```

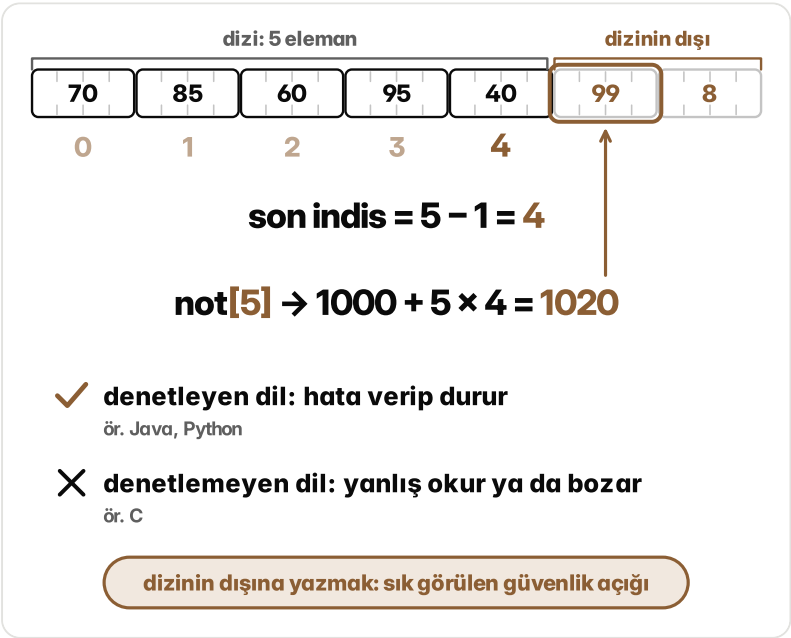
1 çarpma ve 1 toplama yeter. Dizi 1 milyon elemanlı olsa da her elemana aynı hızda ulaşılır; son eleman not[999.999] için de yalnızca 1000 + 999.999 × 4 hesaplanır.




**Şekil 7.2** Her not 4 gözlük bir grupta; adres = başlangıç + indis × 4 ile not[3] → 1000 + 3 × 4 = 1012.

0'ın sebebi de bu formüldedir. İndis, elemanın dizinin başından kaç eleman ileride olduğunu söyler. İlk eleman başlangıcın tam kendisindedir, uzaklığı 0'dır; bu yüzden adı not[0]'dır ve adresi 1000 + 0 × 4 = 1000'dir. Sayma 1'den başlasaydı formül başlangıç + (indis − 1) × 4 olurdu; her erişimde önce 1 çıkarmak zorunda kalırdık.

0'dan başladığımız için 5 elemanlı dizide son indis 5 − 1 = 4'tür. Peki not[5]'i istersek? Bilgisayar yine hesaplar: 1000 + 5 × 4 = 1020. Orası dizinin dışıdır, başka bir verinin yeridir. Bazı diller (örneğin Java ve Python) bunu denetler ve programı hata vererek durdurur. Bazıları (örneğin C) hiç denetlemez; program yanlış sayıyı okur ya da oraya yazıp başka bir veriyi bozar. Dizinin dışına yazmak, bugün de en sık görülen güvenlik açıklarından biridir.




**Şekil 7.3** not[5] → 1000 + 5 × 4 = 1020, dizinin dışına, başka verinin yerine düşer.

## En büyük not?

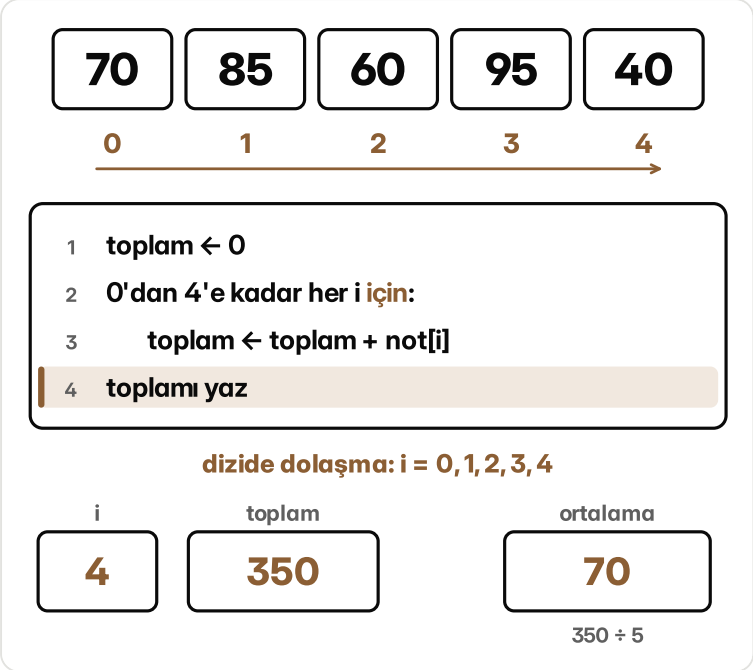
videoda 39:56

5 notun en büyüğünü biz bir bakışta görürüz. Bilgisayar ise dizinin elemanlarına tek tek bakar; bir anda yalnızca bir elemanı görür. O hâlde en büyüğü nasıl bulur?

Önce daha kolay bir işle başlayalım: notların toplamı. Toplam kutusuna 0 koyarız, sonra indisi 0'dan 4'e kadar birer artırıp her notu toplamın üstüne ekleriz:

```
toplam ← 0
0'dan 4'e kadar her i için:
    toplam ← toplam + not[i]
toplamı yaz
```

Dizinin elemanlarını böyle sırayla gezmeye **dizide dolaşma** denir. Kutu sırayla 70, 155, 215, 310 ve 350 olur. Toplam 350; 5'e bölünce ortalama 70.

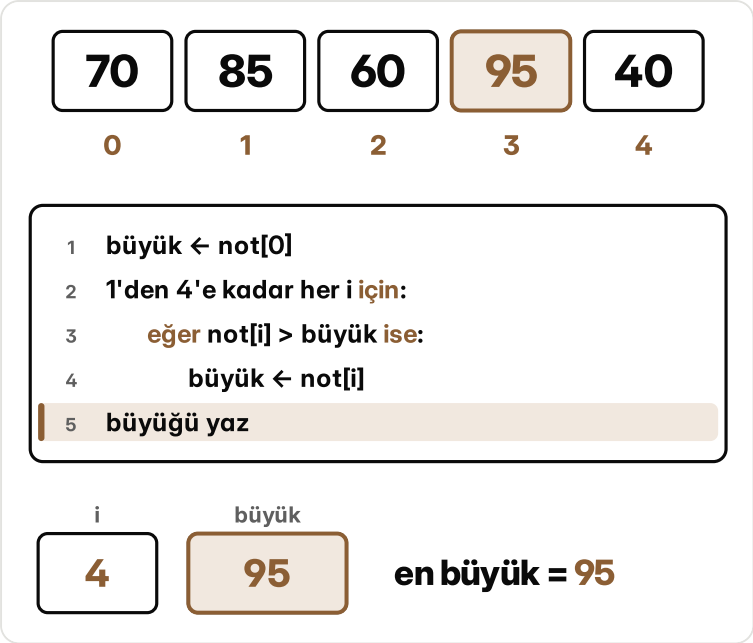



**Şekil 7.4** i 0'dan 4'e ilerlerken toplam kutusu 350'ye ulaşır; ortalama 350 ÷ 5 = 70.

En büyük için de aynı dolaşmayı kullanırız, yalnız kutunun işi değişir. Kutuya ilk notu koyarız; sonra her notu kutudakiyle karşılaştırır, daha büyükse kutuya onu yazarız:

```
büyük ← not[0]
1'den 4'e kadar her i için:
    eğer not[i] > büyük ise:
        büyük ← not[i]
büyüğü yaz
```

Böylece kutuda hep o ana kadar görülen en büyük not durur. 70, yerini 85'e bırakır; 60 geçemez, 95 geçer, 40 geçemez. Dizi bitince kutuda 95 kalır.



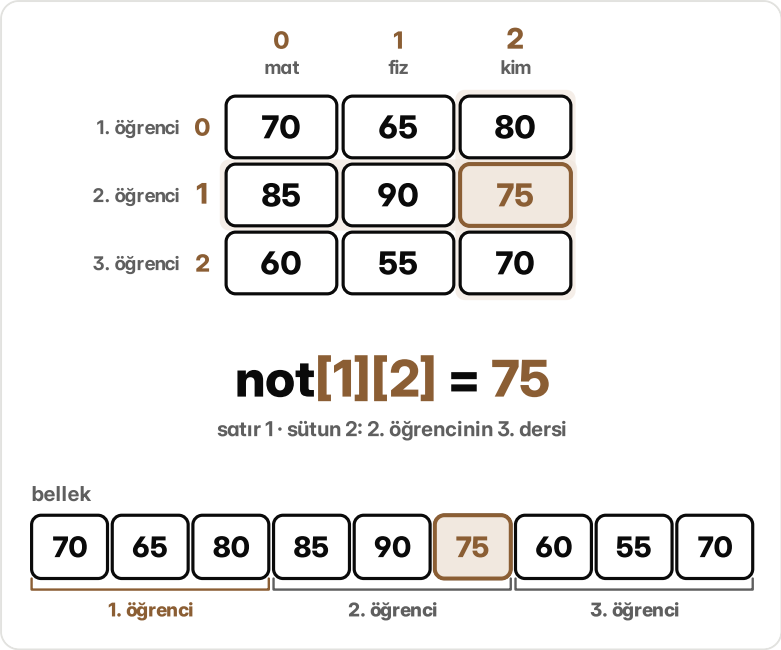

**Şekil 7.5** Dolaşma bitince büyük kutusunda 95 kalır.

En küçükte tek fark karşılaştırmanın yönüdür: not[i] kutudakinden küçükse kutuya yazarız. 70, 60'a, 60 da 40'a yerini bırakır; en küçük not 40. Kutuya baştan 0 koysaydık yanılırdık: hiçbir not 0'dan küçük olmadığı için kutuda 0 kalırdı. Kutuya ilk notun konmasının nedeni budur.

Peki her öğrencinin 3 dersi varsa? Notları bir tablo gibi satır ve sütunlara dizeriz; buna **iki boyutlu dizi** ya da **matris** denir. Aşağıda her satır bir öğrenci, her sütun bir derstir (matematik, fizik, kimya):

```
70  65  80
85  90  75
60  55  70
```

Bir elemana iki indisle ulaşırız, önce satır sonra sütun. İndisler 0'dan başladığı için not[1][2], ikinci öğrencinin üçüncü dersidir: 75. Bellekte gözler tek sıra olduğu için matris satır satır yerleşir: önce birinci öğrencinin 3 notu, sonra ikincininki, sonra üçüncününkü.




**Şekil 7.6** 3 × 3 not matrisi ve bellekte satır satır dizilişi; satır 1, sütun 2 ikinci öğrencinin üçüncü dersidir.

Telefonundaki her fotoğraf da böyle bir matristir: satır satır dizilmiş noktalar ve her birinin renk değeri.

## Kelime nerede biter?

videoda 41:29

Bellekte "bilgi" kelimesinin harfleri yan yana durur, hemen arkasında da başka veriler. Bilgisayar kelimenin nerede bittiğini nereden bilir?

Bir kelime aslında harflerden oluşan bir dizidir; buna **karakter dizisi** denir. Her harf bellekte bir sayı olarak saklanır (b = 98, i = 105, l = 108, g = 103) ve dizinin bir gözüne oturur; ilk harf 0. göze.

Kelimenin bittiği yeri göstermek için son harften sonra değeri 0 olan özel bir karakter konur; buna **sonlandırıcı** denir ve \0 diye gösterilir. Bu yüzden 5 harfli "bilgi" bellekte 6 göz tutar. Bazı diller sonlandırıcı yerine kelimenin uzunluğunu ayrıca saklar.




**Şekil 7.7** b, i, l, g, i harflerinin ardından sonlandırıcı \0 gelir: 5 harf, 6 göz.

Uzunluğu bulmak için ilk gözden başlar, sonlandırıcıya rastlayana kadar sayarız:

```
uzunluk ← 0
kelime[uzunluk] ≠ 0 olduğu sürece:
    uzunluk ← uzunluk + 1
uzunluğu yaz
```

1, 2, 3, 4, 5, sonra 0: "bilgi" 5 harflidir. Sonlandırıcı sayılmaz, ama yer ayırırken ona da bir göz bırakılır: 5 + 1 = 6 göz.




**Şekil 7.8** Sayma \0'da durur: uzunluk 5, ayrılan yer 5 + 1 = 6 göz.

İki kelimeyi karşılaştırırken de harfleri aynı sırayla, tek tek karşılaştırırız. "bilgi" ve "bilge" kelimelerinin ilk 4 harfi aynıdır, 5. harfte ayrılırlar. Bu harfte e'nin sayısı (101), i'ninkinden (105) küçüktür; bu yüzden "bilge", "bilgi"den önce gelir. Karar ilk farklı harfte verilir. Bütün harfler aynıysa ve iki kelime aynı yerde bitiyorsa, yani sonlandırıcılar da aynı gözdeyse, kelimeler eşittir.

Birleştirmede ikinci kelimenin harfleri, birincinin sonlandırıcısının yerinden başlayarak kopyalanır. "bilgi" ile "sayar" birleşince sonlandırıcının yerine "sayar"ın ilk harfi s gelir, en sona yeni bir sonlandırıcı konur: "bilgisayar". Sonuç 10 harf ve 1 sonlandırıcı, yani 11 göz ister. Ayrılan yer 11 gözden azsa harfler dizinin dışına taşar ve yanındaki veriyi bozar; bu yüzden önce yer olup olmadığına bakılır.




**Şekil 7.9** 11 gözlük yerde bilgi ile sayar birleşip bilgisayar olur; 6 gözlük yerde harfler taşar ve komşu veri bozulur.

Telefonda yazdığın her mesaj da bellekte böyle yan yana dizilmiş harflerdir; uzunluğu, karşılaştırması ve birleştirmesi hep bu adımlarla yapılır.

### Özet

- Dizi aynı türden değerleri tek bir adla tutar; elemanlara 0'dan başlayan indisle ulaşılır.
- Elemanlar bellekte arka arkaya durur; 4 gözlük tam sayılarda adres = başlangıç + indis × 4.
- İndis baştan uzaklıktır; 5 elemanlı dizide son indis 4'tür, not[5] dizinin dışına düşer.
- Dizide dolaşarak toplam, en büyük ve en küçük bulunur; kutuya baştan ilk eleman konur.
- Matriste bir elemana önce satır, sonra sütun indisiyle ulaşılır; matris bellekte satır satır yerleşir.
- Karakter dizisi değeri 0 olan sonlandırıcıyla biter; uzunluk ona kadar sayılır, yer ayırırken ona da bir göz bırakılır.

## Sorular

**1** 10 elemanlı bir dizinin son elemanının indisi kaçtır?

**A** 10
**B** 9
**C** 11
**D** 1

**2** **Her tam sayının 4 göz tuttuğu bir dizi 2000 adresinden başlıyor. dizi[7] elemanının adresi kaçtır?**

Cevap: _____

**3** **Notlar 70, 85, 60, 95, 40. "büyük ← not[0]" ile başlayan en büyük bulma döngüsünde büyük kutusu sırasıyla hangi değerleri alır?**

**A** 70, 85, 60, 95, 40
**B** 0, 70, 85, 95
**C** 70, 85, 95
**D** 95

**4** **Satırların öğrenci, sütunların ders olduğu bir not matrisinde not[1][2] neyi gösterir?**

**A** 1. öğrencinin 2. dersini
**B** 2. öğrencinin 1. dersini
**C** 1. dersin 2. öğrencisini
**D** 2. öğrencinin 3. dersini

**5** **1000 adresinden başlayan ve 4 gözlük 5 tam sayıdan oluşan not dizisinde not[5] istenirse ne olur?**

**A** Bilgisayar 1020 adresini hesaplar; orası dizinin dışıdır, başka bir verinin yeridir.
**B** Bilgisayar son eleman olan not[4]'ü verir.
**C** Bilgisayar her zaman 0 değerini verir.
**D** Dizi kendiliğinden 6 elemana büyür.

**6** **"merhaba" kelimesi sonlandırıcısıyla birlikte bellekte kaç göz tutar?**

Cevap: _____

**7** **Harflerin sayıları a = 97 ve e = 101'dir. "bal" ve "bel" harf harf karşılaştırılırsa sonuç ne olur?**

**A** İlk ve son harfleri aynı olduğu için eşittirler.
**B** bel önce gelir, çünkü e'nin sayısı a'nınkinden büyüktür.
**C** bal önce gelir, çünkü 2. harfte a'nın sayısı (97) e'ninkinden (101) küçüktür.
**D** Önce harf sayılarına bakılır; ikisi de 3 harfli olduğu için eşittirler.

**8** **"ders" ile "hane" birleştirilip "dershane" elde edilecek. Sonuç için en az kaç göz ayrılmalıdır?**

Cevap: _____

# 8 Arama

Bu bölümde bir listede bir sayıyı aramanın iki yolunu öğreneceksin: sayılara baştan sona tek tek bakan doğrusal aramayı ve yalnızca sıralı listede işleyen, her adımda kalanı yarıya indiren ikili aramayı. İkisinin de kaç adım sürdüğünü sayacağız; her karşılaştırma bir adımdır.

## Var mı, yok mu?

videoda 43:04

Karışık bir listede bir sayının olmadığını söylemek, olduğunu söylemekten neden daha uzun sürer? Çünkü bilgisayar listeyi bir bütün olarak göremez; her adımda tek bir sayıyı alır ve aradığıyla karşılaştırır.

Şu 8 sayının içinde 37'yi arayalım:

```
14  52  8  37  90  23  71  45
```

Bilgisayar baştan başlar: 14, 37 mi? Hayır. 52, hayır; 8, hayır; 37, evet. 4. adımda buldu ve durdu; arkadaki 4 sayıya hiç bakmadı. Baştan başlayıp sayılara sırayla tek tek bakmaya **doğrusal arama** denir, **sıralı arama** da denir.




**Şekil 8.1** 37, 4. adımda bulunur; arkadaki 4 sayıya bakılmaz.

Peki 60'ı ararsak? 14 değil, 52 değil, 8 değil; bilgisayar 8 sayının hepsine tek tek bakar, hiçbiri tutmaz. "Yok" diyebilmesi için sona kadar gitmesi gerekir, çünkü bakmadığı tek bir sayı aradığı olabilirdi. Sorunun cevabı da bu: bulunca hemen durursun, yok demek içinse hepsine bakmak zorundasın.




**Şekil 8.2** Listede olan 37 için 4 adım yeter; olmayan 60 için 8 sayının hepsine bakılır.

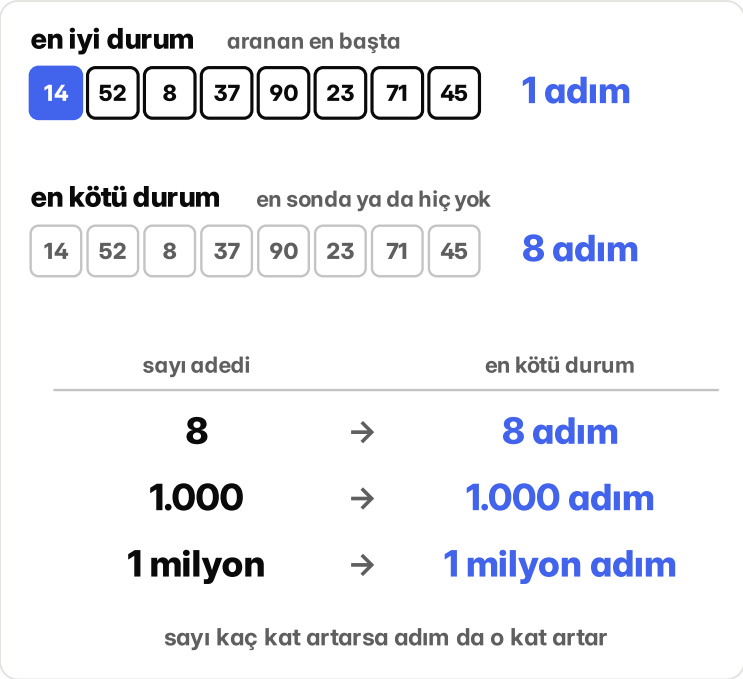
Bunu sözde kodla yazalım. Sıra numarasını tutan sayaç i, 0'dan başlar; n, sayı adedidir:

```
i ← 0
i < n olduğu sürece:
    eğer dizi[i] = aranan ise:
        yaz i ; bitir
    i ← i + 1
yaz "yok"
```

Sayaç sayı adedinden küçük olduğu sürece, o sıradaki sayı arananla aynıysa yerini yaz ve bitir. Değilse sayacı 1 artır ve sonrakine geç. Döngü sonuna kadar dönüp bittiyse aranan yoktur; "yok" yaz. 60 arandığında i sonunda 8 olur, 8 < 8 yanlış çıkar ve döngü biter.

Peki bu arama kaç adım sürer? Aradığın en baştaysa tek adım yeter; buna **en iyi durum** denir. En sondaysa ya da hiç yoksa 8 sayının 8'ine de bakılır; bu da **en kötü durum**dur. Liste büyüdükçe en kötü durum da aynı oranda büyür; sayı kaç kat artarsa adım da o kat artar:

```
8 sayı           8 adım
1.000 sayı       1.000 adım
1 milyon sayı    1 milyon adım
```




**Şekil 8.3** Doğrusal aramada en iyi durum 1 adım, en kötü durum 8 adım; sayı adedi kaç kat artarsa en kötü durum da o kat artar.

Karışık bir listede bundan kısa bir yol yoktur; bilgisayar da sırayla bakar, ama saniyede yüz milyonlarca sayıya. Bir belgede bir sözcüğü aradığında da program metni baştan sona böyle tarar.

## Milyonda kaç adım?

videoda 44:35

1 milyon sayının içinde aradığını en çok kaç adımda bulursun? Sayılar karışıksa tek tek bakmak 1 milyon adım sürebilir. Ama sayılar küçükten büyüğe sıralıysa 20 adım yeter.

İşin sırrı, ortadaki sayıya bakmakta. Sayılar sıralıysa ortadakinin solundakiler ondan küçük, sağındakiler ondan büyüktür. Aradığın sayı ortadakinden büyükse sol yarıda olamaz; o yarının hepsi tek adımda elenir.

15 sıralı sayıda 63'ü arayalım:

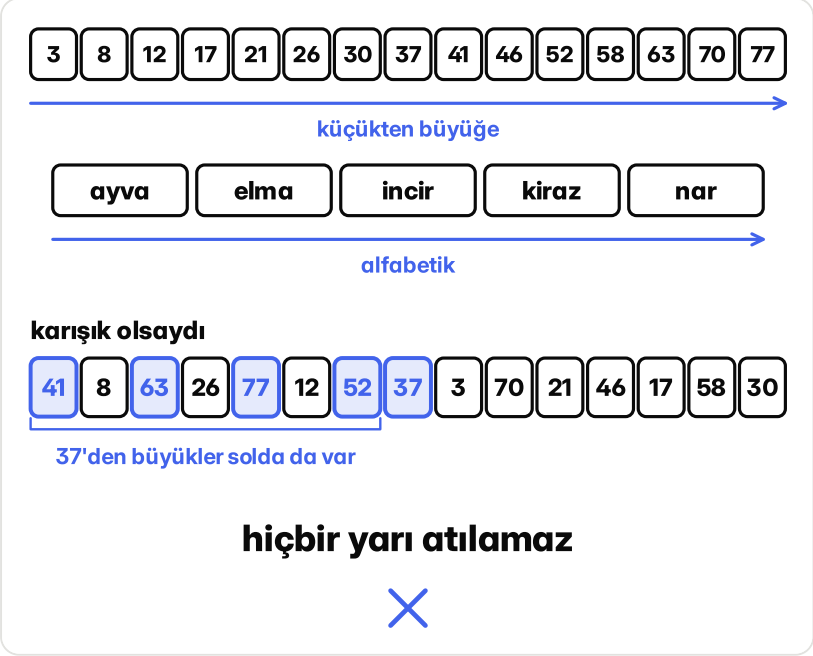
```
3  8  12  17  21  26  30  37  41  46  52  58  63  70  77
```

Ortadaki sayı 37; 63 ondan büyük, sol yarı gider. Kalan 7 sayının ortası 58; yine büyük, yine sol yarı gider. Kalan 3 sayının ortası 70; bu kez küçük, sağdaki gider. Tek sayı kaldı: 63. 4 adımda bulduk. Her adımda kalanı ikiye böldüğü için bu yönteme **ikili arama** denir.




**Şekil 8.4** 63, 4 adımda bulunur: ortalar 37, 58, 70, 63; kalan sayı 15 → 7 → 3 → 1.

Bu yöntem yalnızca sıralı dizide çalışır: küçükten büyüğe dizilmiş sayılarda ya da alfabetik dizilmiş sözcüklerde. Sayılar karışık olsaydı 37'den büyük bir sayı solda da durabilirdi; hiçbir yarıyı gözü kapalı atamazdık.




**Şekil 8.5** Aynı 15 sayı karışık dizilince 37'den büyük sayılar solda da durur; hiçbir yarı atılamaz.

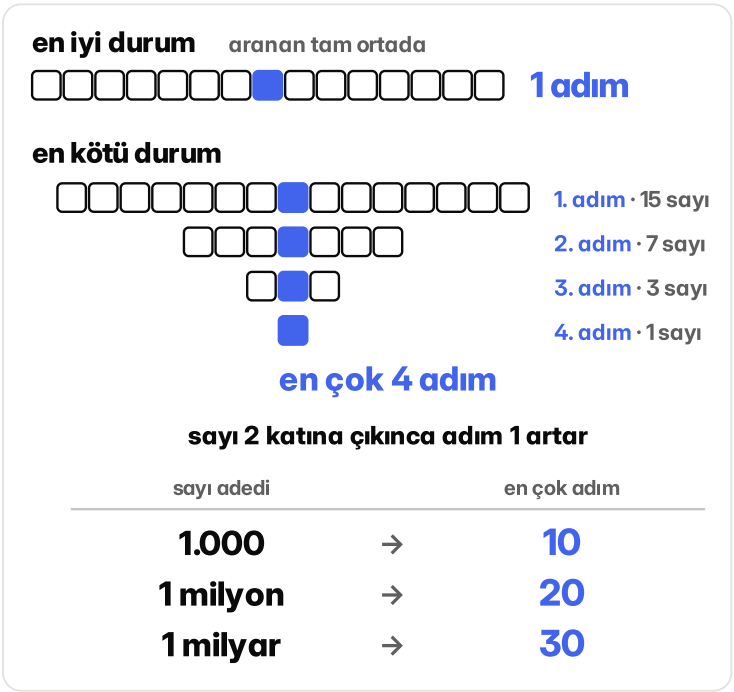
Sözde kodda iki sınır tutarız: alt ve üst.

```
alt ← 0 ; üst ← n − 1
alt ≤ üst olduğu sürece:
    orta ← (alt + üst) ÷ 2
    eğer dizi[orta] = aranan ise:
        yaz orta ; bitir
    değilse eğer dizi[orta] < aranan ise:
        alt ← orta + 1
    değilse:
        üst ← orta − 1
yaz "yok"
```

Alt üstü geçmediği sürece ikisinin ortasındaki sayıya bakarız. Ortanın yeri, alt ile üstün toplamının yarısıdır, küsurat atılır: 0 ile 14 için 7. Ortadaki sayı arananın kendisiyse yerini yazıp bitiririz. Aranandan küçükse alt sınırı ortanın bir sağına, büyükse üst sınırı ortanın bir soluna çekeriz; örnekte dizi[7] = 37 < 63 olduğu için alt = 7 + 1 = 8 olur. Alt üstü geçerse aranan dizide yoktur.

Peki kaç adım sürer? En iyi durumda aradığın tam ortadadır: tek adım. En kötü durumda tek sayı kalana kadar yarıya bölersin: 15 sayı 7'ye, 7 sayı 3'e, 3 sayı 1'e iner; en çok 4 adım. Her adım kalanı yarıya indirdiği için, dizi 2 katına çıkınca yalnızca 1 adım eklenir:

```
1.000 sayı        en çok 10 adım
1 milyon sayı     en çok 20 adım
1 milyar sayı     en çok 30 adım
```




**Şekil 8.6** En iyi durum 1 adım; en kötü durum 15 sayıda 4, 1.000 sayıda 10, 1 milyon sayıda 20, 1 milyar sayıda 30 adım.

Kâğıt sözlükte bir kelime ararken sen de buna benzer bir şey yaparsın: bir sayfa açar, kelime önde mi arkada mı kalıyor diye bakar, öbür tarafı hiç okumazsın. Sıralı bir listede bilgisayar da böyle yapar; 1 milyon kayıtta bile aradığını 20 adımda bulur.

**Özet**

- Bilgisayar listeyi bir bütün olarak göremez; her adımda tek bir sayıyı arananla karşılaştırır.
- Doğrusal arama sayılara baştan sona tek tek bakar; bulunca durur, "yok" demek içinse hepsine bakar.
- Doğrusal aramada en iyi durum 1 adım, en kötü durum sayı adedi kadar adımdır.
- İkili arama yalnızca sıralı dizide çalışır; her adımda ortaya bakar ve kalanın yarısını eler.
- İkili aramada dizi 2 katına çıkınca en kötü durum yalnızca 1 adım artar: 1 milyon sayıda en çok 20 adım.

## Sorular

**1** **Karışık 8 sayılık bir listede aranan sayının olmadığını söyleyebilmek için bilgisayar kaç sayıya bakmalıdır?**

A Yalnızca ilk sayıya
B Ortadaki sayıya
C 8 sayının hepsine
D Sayıların yarısına

**2** **14 52 8 37 90 23 71 45 dizisinde doğrusal aramayla 23 kaçıncı adımda bulunur?**

Cevap:

**3** **2.000 sayılık karışık bir listede doğrusal arama en kötü durumda kaç adım sürer?**

A 2.000
B 1.000
C 11
D 1

**4** **İkili aramanın doğru çalışması için hangisi şarttır?**

A Sayı adedinin tek olması
B Aranan sayının listede bulunması
C Listenin 1.000 sayıdan uzun olması
D Sayıların küçükten büyüğe sıralı olması

**5** **Aynı 15 sayı karışık dizilmiş ve ortadaki sayı 37. 63'ü ikili aramayla aramak neden yanlış sonuç verebilir?**

A Karışık dizide ortadaki sayı hesaplanamaz.
B 15 tek sayı olduğu için dizi yarıya bölünemez.
C 37'den büyük sayılar solda da durabilir; sol yarıyı atmak 63'ü de atabilir.
D İkili arama yalnızca alfabetik dizilmiş sözcüklerde çalışır.

**6** İkili aramada alt = 0 ve üst = 9 iken orta kaçtır?

Cevap: _____

**7** 3 8 12 17 21 26 30 37 41 46 52 58 63 70 77 dizisinde (indis 0-14) ikili aramayla 21 aranıyor. Sırasıyla hangi ortalara bakılır?

**A** 37, 21

**B** 37, 17, 26, 21

**C** 3, 8, 12, 17, 21

**D** 37, 26, 21

**8** Sıralı 1.000 sayıda ikili arama en çok 10 adım sürer. Sıralı 4.000 sayıda en çok kaç adım sürer?

Cevap: _____

# 9 Sıralama ve karmaşıklık

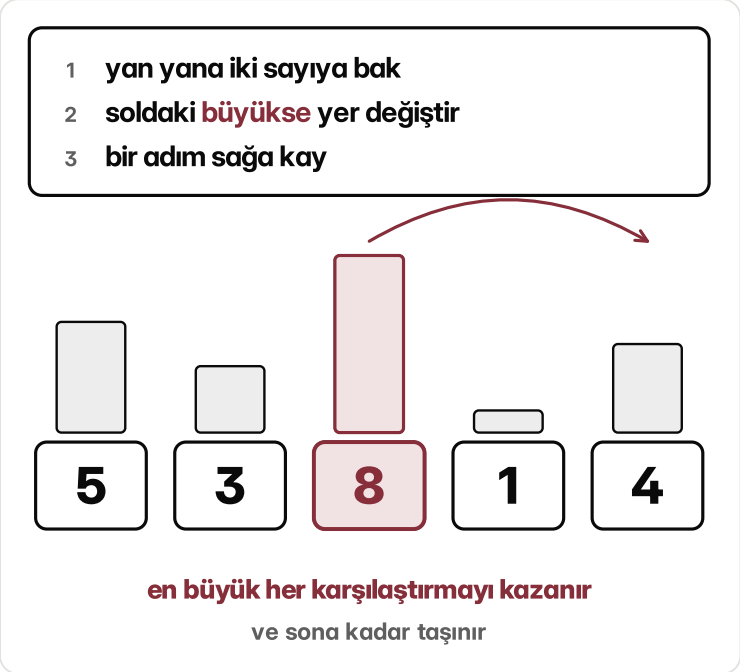
Bu bölümde bilgisayarın karışık bir listeyi nasıl sıraya dizdiğini önce 3 basit, sonra 2 hızlı yöntemle göreceksin. Ardından bir algoritmanın hızını saniye yerine adım sayısıyla ölçmeyi, bunu büyük O gösterimiyle yazmayı ve zamanın yanında belleği de hesaba katmayı öğreneceksin.

## Nasıl sıraya dizer?

videoda 46:29

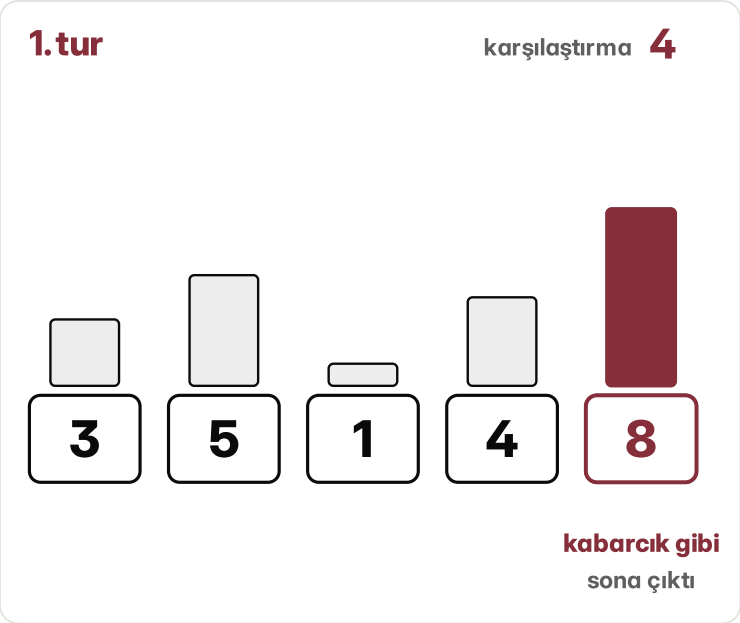
Bilgisayar her seferinde yalnızca iki sayıyı karşılaştırabiliyorsa karışık bir listeyi nasıl sıraya dizer? Listeye bakıp en küçüğü bir bakışta göremez; ama iki sayıdan hangisinin büyük olduğunu bilir. Bu bölümdeki bütün sıralama yöntemleri, bu tek işlemi akıllıca tekrarlamanın yollarıdır.

Elimizde 5, 3, 8, 1 ve 4 olsun. En basit yönteme **kabarcık sıralaması** (bubble sort) denir. Kuralı 3 adımdır: yan yana iki sayıya bak; soldaki büyükse yerlerini değiştir; sonra bir adım sağa kay. Bu kural en büyük sayıyı sona taşır, çünkü en büyüğe ulaşıldığı andan itibaren o her karşılaştırmayı kazanır ve her yer değiştirmede bir adım sağa geçer.




**Şekil 9.1** Kabarcık sıralamasının 3 satırlık kuralı; 5, 3, 8, 1, 4 listesinde en büyük sayı 8 vurgulu ve sona giden bir okla gösterilmiş.

Birinci turu adım adım izleyelim. 5 ile 3: 5 büyük, yer değiştirirler. 5 ile 8: sıra doğru, dokunulmaz. 8 ile 1 yer değiştirir, 8 ile 4 de. Tur 4 karşılaştırmayla biter ve liste 3, 5, 1, 4, 8 olur. 8, sudaki bir kabarcığın yüzeye çıkması gibi en sona çıkmıştır; yöntemin adı da buradan gelir.




**Şekil 9.2** Birinci turun sonu: liste 3, 5, 1, 4, 8; 4 karşılaştırma yapılmış ve 8 koyu renkle en sona yerleşmiş.

8 artık yerindedir, ona bir daha bakılmaz; her tur bir öncekinden bir sayı eksik listeyle çalışır. İkinci tur kalan 4 sayıya bakar ve 5'i sona taşır. Üçüncü turda 1 ile 3 yer değiştirir. Son turda tek çift kalır, o da doğru sıradadır. Bütün turlar şöyle:

```
başlangıç   5  3  8  1  4
1. tur      3  5  1  4  8     4 karşılaştırma, 3 yer değiştirme
2. tur      3  1  4  5  8     3 karşılaştırma, 2 yer değiştirme
3. tur      1  3  4  5  8     2 karşılaştırma, 1 yer değiştirme
4. tur      1  3  4  5  8     1 karşılaştırma, 0 yer değiştirme
```

Kaç karşılaştırma yaptık? 4 + 3 + 2 + 1 = 10. Genel kural da buradan çıkar: n sayılık bir listede ilk tur n − 1 karşılaştırma yapar, sonraki her tur bir eksik. 100 sayı olsaydı 99'dan 1'e kadar inerdik: 99 + 98 + … + 1 = 100 × 99 / 2 = 4.950 karşılaştırma. 1.000 sayıda 1.000 × 999 / 2 = 499.500, yani yaklaşık 500.000. Sayılar 10 katına çıkınca karşılaştırma yaklaşık 100 katına çıkıyor.




**Şekil 9.3** Turlara göre karşılaştırma sayısı 4, 3, 2, 1, toplam 10; tabloda 5 sayı için 10, 100 sayı için 4.950, 1.000 sayı için yaklaşık 500.000 karşılaştırma.

Bu yüzden kabarcık sıralaması anlaşılması en kolay yöntemlerden biridir, ama büyük listelerde çok yavaş kalır.

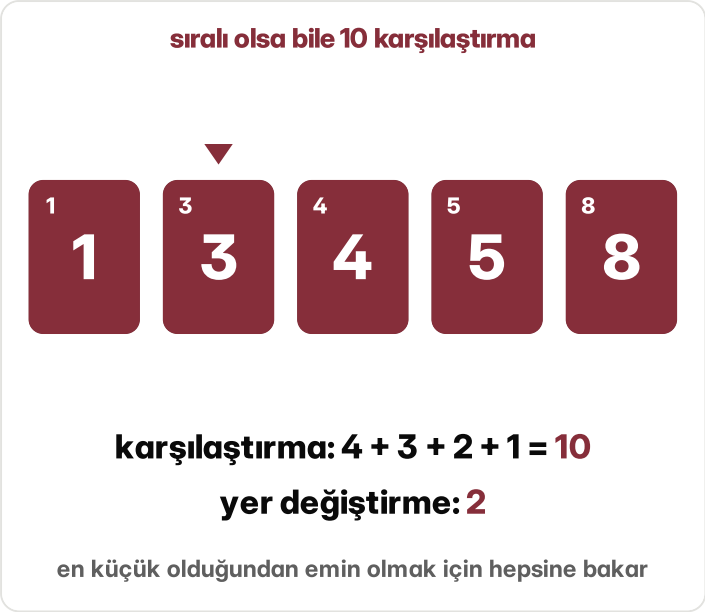
## Kartları nasıl dizersin?

videoda 47:50

Masadaki karışık kartları küçükten büyüğe dizerken ne yaparsın? Çoğumuz iki yoldan birini izleriz; ikisi de bilgisayarın kullandığı birer sıralama algoritmasıdır. Kartlarda yine 5, 3, 8, 1, 4 yazsın.

Birinci yol **seçmeli sıralama** (selection sort): en küçüğü seç, en başa koy. En küçük sayı nasılsa başa gelecektir; onu yerine koyunca bir daha oynatmayız ve iş bir sayı azalır. Ama bilgisayar en küçüğü bir bakışta göremez; soldan sağa yürür ve o ana kadar gördüğü en küçük sayıyı aklında tutar.

İlk taramada önce 5 en küçüktür; 3 daha küçük; 8 değil; 1 daha küçük; 4 değil. 4 karşılaştırmada en küçük bulunur: 1. Baştaki 5'le yer değiştirir ve liste 1, 3, 8, 5, 4 olur. Kalan 4 sayıda en küçük 3'tür, zaten yerindedir. Kalan 3 sayıda en küçük 4'tür, 8'le yer değiştirir: 1, 3, 4, 5, 8. Son ikide 5 zaten yerindedir. Toplam 4 + 3 + 2 + 1 = 10 karşılaştırma, ama yalnızca 2 yer değiştirme. Liste baştan sıralı olsaydı bile yine 10 karşılaştırma yapardı; en küçüğü bulduğundan emin olmak için kalan her sayıya bakmak zorundadır.




**Şekil 9.4** Seçmeli sıralama bitmiş: 10 karşılaştırma ve 2 yer değiştirme; liste sıralı olsa bile en küçüğü bulmak için kalanların hepsine bakılır.

İkinci yol, elinde kart dizen birinin yoludur: **eklemeli sıralama** (insertion sort). Eldeki kartlar hep sıralıdır. Masadan yeni kart alınca onu sağdan sola doğru, kendinden büyükleri geçerek yerine sokarsın. Tek kartlık bir el zaten sıralı olduğu için oradan başlanır: elde 5, masada 3, 8, 1, 4.




**Şekil 9.5** Eklemeli sıralamanın başı: elde tek kart (5) zaten sıralı; masadaki kartlar büyükleri geçerek ele girecek.

3 gelir, 5'ten küçüktür, önüne girer. 8 gelir, 5'ten büyüktür, hemen sona oturur. 1 gelir; 8'i, 5'i ve 3'ü geçer, en başa girer. 4 gelir; 8'i ve 5'i geçer, 3'ten büyük olduğu için orada durur. Her kartın kaç karşılaştırma yaptığı:

```
el: 5                masa: 3 8 1 4
3 gelir → 3 5          1 karşılaştırma
8 gelir → 3 5 8        1 karşılaştırma
1 gelir → 1 3 5 8      3 karşılaştırma
4 gelir → 1 3 4 5 8    3 karşılaştırma
                       toplam 8
```

Liste zaten sıralıysa her yeni kart tek karşılaştırmayla yerini bulur: 5 sayıda 4 karşılaştırma yeter. Ters sıralıysa her kart en başa kadar gider: 1 + 2 + 3 + 4 = 10. Karışık listede iki yöntem de her sayıyı neredeyse bütün öbürleriyle karşılaştırır; ama sıralanmaya az kalmış bir listede eklemeli sıralama çok daha çabuk biter.




**Şekil 9.6** Karşılaştırma sayıları: karışık listede seçmeli 10, eklemeli 8; sıralı listede 10 ve 4; ters sıralı listede 10 ve 10.

## Bir milyon sayı

videoda 49:32

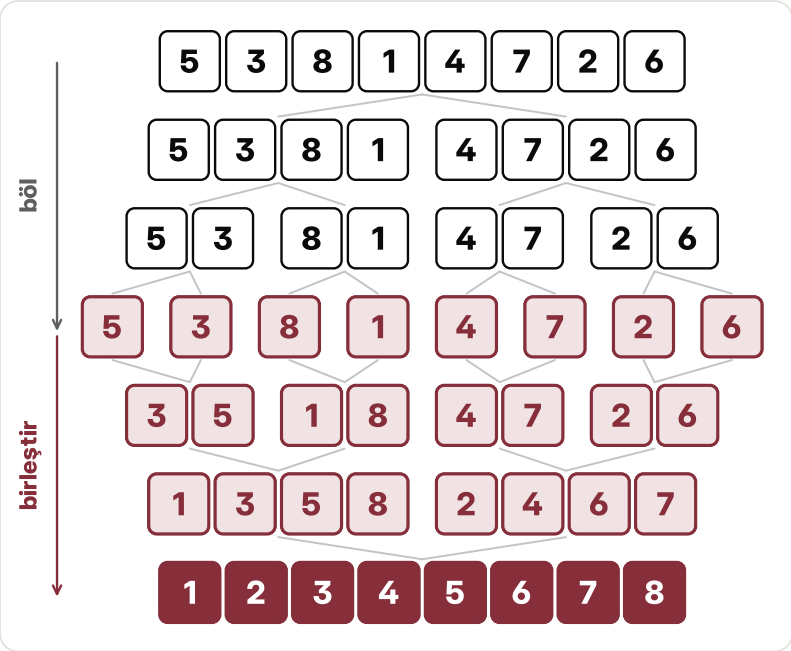
Bir milyon sayıyı sıralamak için kaç karşılaştırma gerekir? Her sayıyı öbür bütün sayılarla karşılaştıran basit yöntemler n(n – 1)/2 kuralıyla yaklaşık 500 milyar karşılaştırma yapar. Daha akıllı yöntemler, örneğin birazdan göreceğin birleştirmeli sıralama, aynı işi 20 milyondan az karşılaştırmayla bitirir. Sırları listeyi bölmektir.

Önce kolay bir iş: iki sıralı listeyi tek bir sıralı listede birleştirmek. 1, 3, 5, 8 ile 2, 4, 6, 7 olsun. İki listenin yalnızca başına bakılır ve küçük olan alınır: 1 ile 2'den 1 alınır; 3 ile 2'den 2 alınır; sonra sırasıyla 3, 4, 5, 6 ve 7. Bir liste bitince öbüründe kalan 8 karşılaştırmasız sona eklenir.

```
1 < 2 → 1     3 > 2 → 2     3 < 4 → 3     5 > 4 → 4
5 < 6 → 5     8 > 6 → 6     8 > 7 → 7     kalan → 8
sonuç: 1 2 3 4 5 6 7 8   (7 karşılaştırma)
```

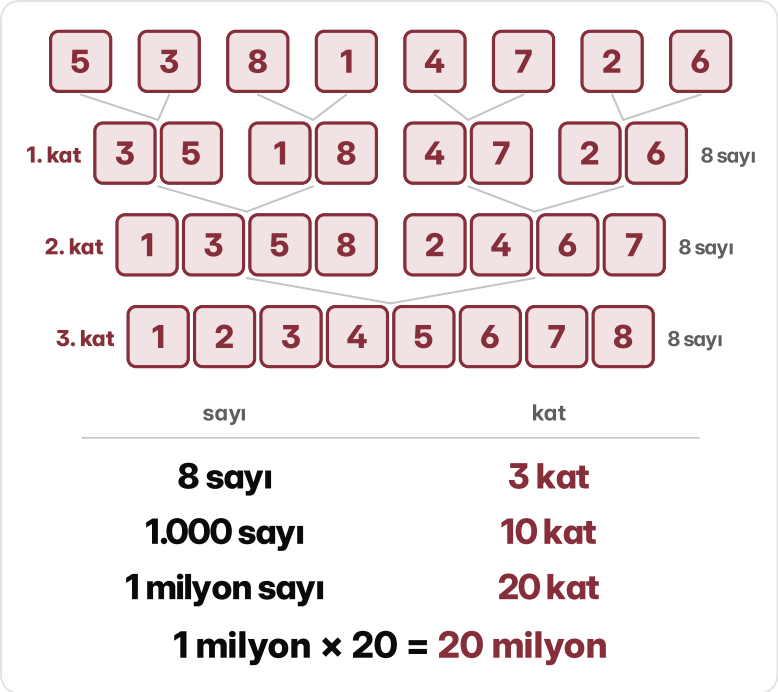
Her karşılaştırma bir sayıyı yerine koyar; birleştirmek bu yüzden hızlıdır.

**Birleştirmeli sıralama** (merge sort) bu kolaylığı kullanır. Karışık listeyi ikiye, her yarıyı yine ikiye böler; tek sayı kalana kadar. Tek sayılık bir liste zaten sıralıdır. Sonra parçaları ikişer ikişer birleştirerek yukarı çıkar: tekler ikili, ikililer dörtlü, dörtlüler sekizli sıralı liste olur. 5, 3, 8, 1, 4, 7, 2, 6 listesinde son adımda birleşen iki dörtlü, yukarıdaki 1, 3, 5, 8 ile 2, 4, 6, 7'dir.




**Şekil 9.7** Birleştirmeli sıralama ağacı: 5, 3, 8, 1, 4, 7, 2, 6 3 katta tek sayılara bölünür, 3 katta birleşerek 1'den 8'e sıralanır.

Neden hızlı? Her birleştirme katında her sayı bir kez yerine konur. Liste her seferinde yarıya indiği için kat sayısı çok yavaş artar: 8 sayı 3 bölmede teke iner (8 → 4 → 2 → 1), 1.000 sayı 10, 1 milyon sayı 20 bölmede. Kat sayısı bu kadardır. 1 milyon sayı çarpı 20 kat: 20 milyon. Basit yöntemlerin 500 milyarıyla karşılaştırınca 25.000 kat daha az iş.




**Şekil 9.8** 3 birleştirme katının her birinde 8 sayının hepsi bir kez yerine konur; tabloda 8 sayı için 3, 1.000 için 10, 1 milyon için 20 kat ve 1 milyon × 20 = 20 milyon.

**Hızlı sıralama** (quick sort) da böler, ama başka türlü: listeden bir sayı seçer; bu sayıya **pivot** denir. Pivottan küçükleri sola, büyükleri sağa ayırır. Pivot 5 olursa 3, 1, 4 ve 2 sola; 8, 7 ve 6 sağa geçer. 5 artık kesin yerindedir: solundakilerin hepsi küçük, sağındakilerin hepsi büyüktür. İki grup ayrı ayrı aynı yolla sıralanır. Pivot hep en küçük ya da en büyük sayı çıkarsa gruplar bölünmez: zaten sıralı 1'den 8'e listede pivot 1 seçilirse sol grup boş kalır, sağda 7 sayı birden durur. Her adımda yalnız bir sayı ayrıldığı için yöntem yavaşlar; bu yüzden pivot çoğu zaman rastgele seçilir.




**Şekil 9.9** Pivot 5: küçükler (3, 1, 4, 2) sola, büyükler (8, 7, 6) sağa; altta sıralı listede pivot 1 seçilince sol boş kalır, sağdaki 7 sayı bölünmez.

Programlama dillerinin hazır sıralama işlevleri bu iki fikrin, bölüp birleştirmenin ve pivotla ayırmanın üstüne kurulur; bir milyon sayıyı saniyeden kısa sürede sıralarlar.

## Hız neyle ölçülür?

videoda 51:11

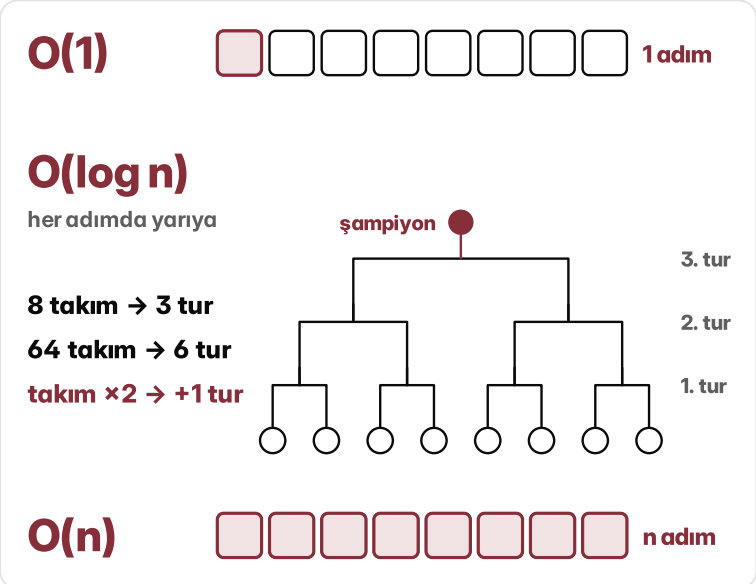
Bir algoritmanın hızını saniyeyle ölçebilir miyiz? Ölçemeyiz: aynı program eski bir bilgisayarda 8 saniye, yenisinde 1 saniye sürebilir. Ama iki bilgisayarda da attığı adım sayısı aynıdır; bu yüzden adım sayarız.

Bir listedeki sayıları toplayalım: her sayı için bir toplama yaparız. 4, 7, 2, 9, 3 listesinde toplam sırasıyla 4, 11, 13, 22 ve 25 olur; 5 sayı, 5 toplama. Listede n tane sayı varsa n toplama: 1.000 sayıda 1.000. Liste iki katına çıkınca iş de iki katına çıkar; bizi ilgilendiren bu büyümedir.

```
toplam ← 0
her sayı için:
    toplam ← toplam + sayı
yaz toplam
```

Tam sayım biraz daha uzundur: başta toplam sıfırlanır, sonda sonuç yazılır; 1.000 sayıda 1 + 1.000 + 1 = 1.002 adım. Ama liste büyüdükçe bu 2 adım önemsizleşir. Her sayı için 2 iş yapsak bile büyüme biçimi değişmez: n 1.000'den 2.000'e çıkınca n + 2 de 1.002'den 2.002'ye, 2n de 2.000'den 4.000'e çıkar; hepsi yaklaşık iki katına. **Büyük O gösterimi** (big O notation) bu yüzden yalnız büyüyen kısmı yazar: liste toplamı O(n)'dir, yani adım sayısı n ile aynı oranda büyür.

Sık görülen 5 sınıfın ilk 3'ü yavaş büyür. O(1): listenin ilk sayısını okumak; liste ne kadar uzun olursa olsun tek adım. O(log n): her adımda işi yarıya indirmek, eleme turnuvası gibi. 8 takım 3 turda, 64 takım 6 turda şampiyonunu bulur; takım sayısı iki katına çıkınca yalnızca bir tur eklenir. O(n): her elemana bir kez dokunmak, toplamadaki gibi.



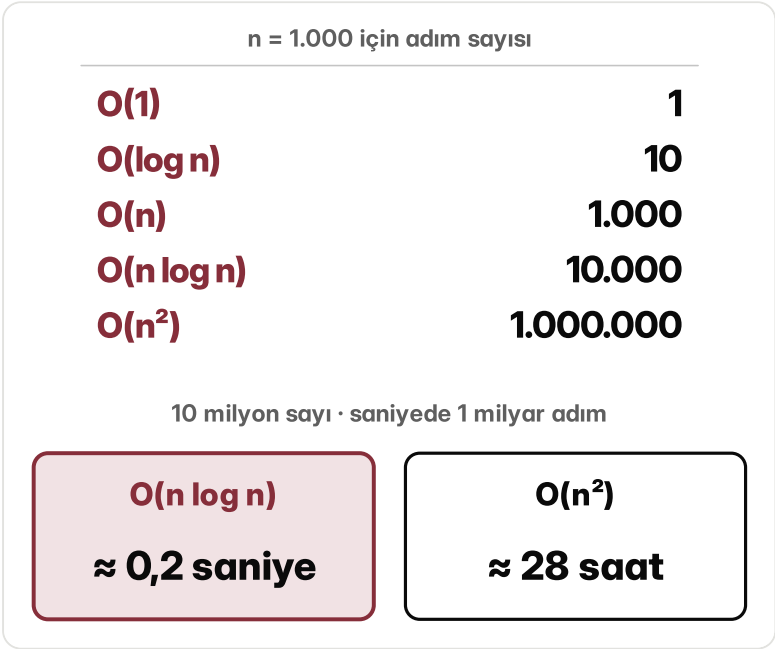

**Şekil 9.10** Yavaş büyüyen sınıflar: O(1) için tek hücre, O(log n) için 8 takımlı eleme ağacı (3 tur), O(n) için 8 hücrenin hepsi.

Sonraki ikisi hızlı büyür. O(n log n): n tane işin her biri log n adım sürer; listeyi bölerek sıralayan yöntemler bu sınıftadır (8 sayıda 8 × 3 = 24 adım). O(n²): herkesin herkesle tokalaşması gibi. 10 kişide 10 × 9 / 2 = 45 tokalaşma, 100 kişide 4.950; kişi 10 katına çıkınca iş yaklaşık 100 katına çıkar.

1.000 elemanlı bir girdide fark şöyledir (log 1.000 yaklaşık 10):

```
O(1)          1 adım
O(log n)      10
O(n)          1.000
O(n log n)    10.000
O(n²)         1.000.000
```

Saniyede 1 milyar adım atan bir bilgisayar 10 milyon sayıyı O(n log n) süren bir yöntemle yaklaşık 10.000.000 × 23 = 230 milyon adımda, yani 0,2 saniye kadar sürede sıralar. O(n²) süren bir yöntemle 10.000.000 × 10.000.000 = 100.000 milyar adım gerekir: 100.000 saniye, yaklaşık 28 saat.




**Şekil 9.11** n = 1.000 için adım sayıları ve 10 milyon sayıda saniyede 1 milyar adımla O(n log n) için yaklaşık 0,2 saniye, O(n²) için yaklaşık 28 saat.

Büyük O yalnızca adımları değil, algoritmanın istediği ek yeri de ölçer. 6, 1, 9, 4, 7, 2 listesini ters çevirelim. Yeni bir liste açıp sayıları sondan başa kopyalarsak n kadar ek yer gerekir. Baştakiyle sondakini yer değiştirip içeri doğru ilerlersek (6 ile 2, 1 ile 7, 9 ile 4) tek bir geçici kutu yeter. İkisi de O(n) adım sürer; ama birincinin ek belleği O(n), ikincininki O(1). Adım sayısının büyümesine **zaman karmaşıklığı**, ek yerin büyümesine **bellek karmaşıklığı** denir.




**Şekil 9.12** Listeyi ters çevirmenin iki yolu: yeni listeye kopyalama (zaman O(n), bellek O(n)) ve uçlardan yer değiştirme (zaman O(n), bellek O(1)).

Bir algoritma seçerken sorulan iki soru da budur: girdi büyüdükçe ne kadar uzar, ne kadar yer ister?

### Özet

- Kabarcık sıralaması yan yana sayıları karşılaştırıp her turda en büyüğü sona taşır; n sayıda $n(n-1)/2$ karşılaştırma yapar.
- Seçmeli sıralama her turda en küçüğü bulup başa koyar ve liste sıralı olsa bile aynı sayıda karşılaştırma yapar; eklemeli sıralama sıralanmaya az kalmış listede çok çabuk biter.
- Birleştirmeli sıralama listeyi yarıya bölüp sıralı parçaları birleştirir; bir milyon sayıyı 20 milyondan az karşılaştırmayla sıralar.
- Hızlı sıralama pivottan küçükleri sola, büyükleri sağa ayırır; pivot hep en küçük ya da en büyük çıkarsa yavaşlar.
- Büyük O gösterimi adım sayısının girdiyle nasıl büyüdüğünü yazar: $O(1)$, $O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$.
- Zaman karmaşıklığı adım sayısının, bellek karmaşıklığı ek yerin büyümesini ölçer.

## Sorular

**1** **Kabarcık sıralamasında bir tur bittiğinde hangi sayı kesin olarak yerine oturmuş olur?**

**A** Turda bakılan sayıların en küçüğü

**B** Turda bakılan sayıların en büyüğü

**C** Listenin ilk sayısı

**D** Turda en çok yer değiştiren sayı

**2** **Seçmeli sıralama, zaten sıralı olan 5 sayılık bir listede kaç karşılaştırma yapar?**

**A** 0

**B** 4

**C** 10

**D** 5

**3** **Kabarcık sıralaması 6 sayılık bir listeyi sıralarken toplam kaç karşılaştırma yapar?**

Cevap:

**4** **Bir algoritma n sayılık bir listede 3n + 5 adım atıyor. Büyük O gösterimiyle bu algoritma hangi sınıftadır?**

**A** $O(n)$

**B** $O(3n + 5)$

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

**D** $O(1)$

**5** **Hızlı sıralama hangi durumda yavaşlar?**

**A** Pivot, sayıları iki eşit gruba ayırdığında

**B** Pivot rastgele seçildiğinde

**C** Listede çift sayıda sayı olduğunda

**D** Pivot hep en küçük ya da en büyük sayı çıktığında

**6** **Bir listeyi, baştakiyle sondakini yer değiştirip içeri doğru ilerleyerek ters çeviriyoruz. Bu yöntemin zaman ve ek bellek karmaşıklığı nedir?**

**A** Zaman $O(n)$, bellek $O(n)$

**B** Zaman $O(n)$, bellek $O(1)$

**C** Zaman $O(1)$, bellek $O(1)$

**D** Zaman $O(n^2)$, bellek $O(1)$

**7** **Eklemeli sıralama 1, 3, 5, 4, 8 listesini sıralarken toplam kaç karşılaştırma yapar?**

Cevap:

**8** **Saniyede 1 milyar adım atan bir bilgisayar, $O(n^2)$ adım süren bir yöntemle 100.000 sayıyı yaklaşık kaç saniyede sıralar?**

Cevap:

# 10 İşletim sistemleri

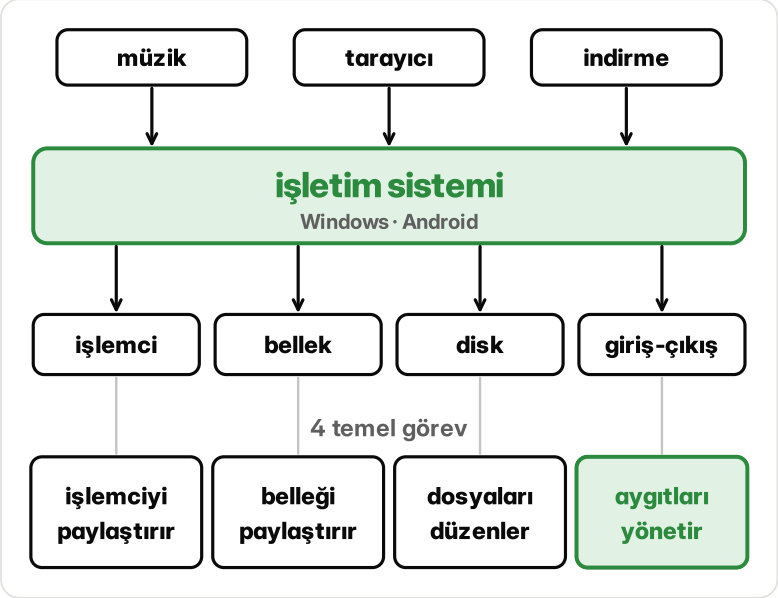
Bu bölümde bilgisayarı yöneten ana programı, işletim sistemini tanıyacaksın: birkaç çekirdeği yüzlerce program arasında nasıl paylaştırdığını, belleği nasıl böldüğünü, dosyaları diskte nasıl bulduğunu ve aygıtlarla nasıl konuştuğunu göreceksin.

## Aynı anda mı?

videoda 53:17

Müzik dinlerken tarayıcıda geziniyorsun, bir yandan da dosya iniyor. Arkada, sen açmasan bile yüzlerce program daha çalışıyor. Oysa işlemcinde yalnızca birkaç çekirdek var, örneğin 4. Bu kadar program nasıl aynı anda çalışıyor?

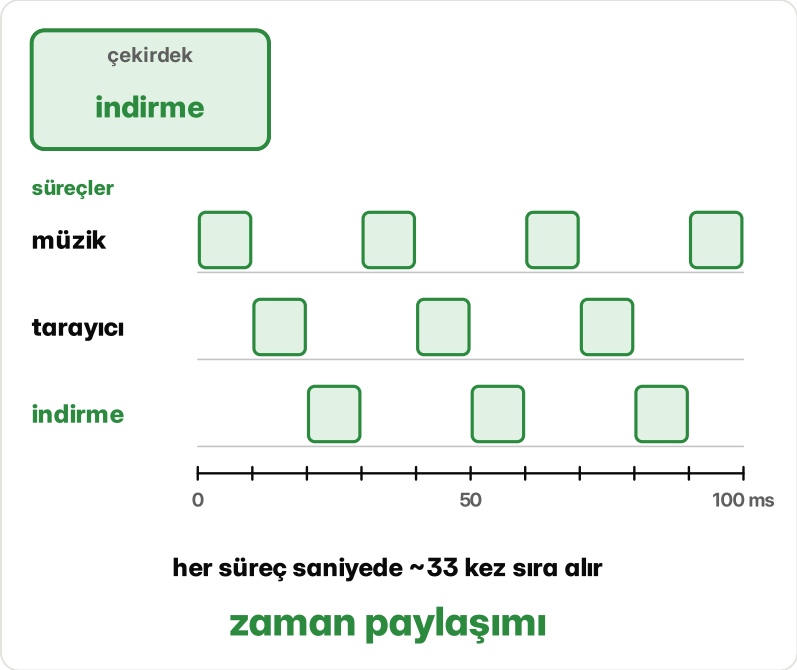
Bunu **işletim sistemi** yönetir: bilgisayardaki Windows, telefondaki Android birer işletim sistemidir. Öteki programlar donanıma doğrudan değil, işletim sistemi aracılığıyla ulaşır. 4 temel görevi vardır: işlemciyi paylaştırmak, belleği paylaştırmak, dosyaları düzenlemek ve giriş-çıkış aygıtlarını yönetmek.




**Şekil 10.1** Programlar ile işlemci, bellek, disk ve giriş-çıkış arasında işletim sistemi katmanı; altta işletim sisteminin 4 temel görevi.

Çalışmakta olan her programa **süreç** (process) denir. Bir çekirdek bir anda tek bir süreci yürütür; işletim sistemi çekirdeği süreçlere sırayla verir. Müzik, diyelim saniyenin yüzde biri (10 milisaniye) kadar çalışır, sonra durdurulur; hangi komutta kaldığı ve işlemcideki değerler kaydedilir. Sıra tarayıcıya geçer, sonra indirmeye, sonra yine müziğe. Müzik kaydedilen yerden devam eder, kesildiğini fark etmez. Her süreç saniyede onlarca kez sıra aldığı için hepsinin aynı anda çalıştığını sanırız. Buna **zaman paylaşımı** (time sharing) denir. 3 süreçle sayalım:

```
1 saniye = 1.000 ms
1.000 ms / 10 ms = 100 dilim
100 dilim / 3 süreç ≈ 33 sıra (her süreç için, saniyede)
```




**Şekil 10.2** Müzik, tarayıcı ve indirme çekirdeği 10 ms'lik dilimlerle sırayla kullanır; her süreç saniyede yaklaşık 33 kez sıra alır.

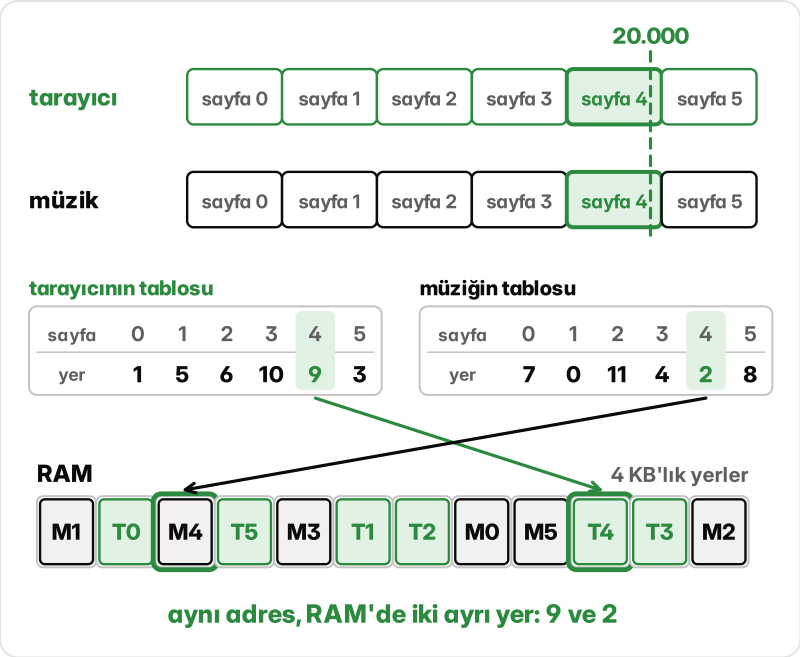
Peki birkaç çekirdek yüzlerce sürece nasıl yetişir? Çünkü süreçlerin çoğu bir şey bekler: bir tuşa basılmasını, ağdan gelecek veriyi. Bir süreç her an 3 durumdan birindedir: hazır, çalışıyor ya da bekliyor. Durumlar arasındaki geçişler şöyledir:

- Hazır süreç sırası gelince çalışmaya başlar.
- Çalışan sürecin süresi dolunca yeniden hazıra döner.
- Çalışan süreç bir şey beklemeye başlayınca bekliyor durumuna geçer.
- Bekleyen süreç, beklediği şey gelince yeniden hazır olur.

Bekleyen süreç sıraya girmez; çekirdek yalnızca hazır olanlar arasında paylaştırılır. Yüzlerce sürecin çoğu bir şey beklediği için çekirdeği paylaşan hazır süreç sayısı azdır.

Bellek de paylaştırılmalıdır. Tarayıcı da müzik de diyelim 20.000 numaralı adrese yazıyor; aynı yere yazsalar birbirinin verisini bozarlardı. Bu yüzden işletim sistemi RAM'i 4 kilobaytlık (4.096 bayt) parçalara böler; bu parçalara **sayfa** (page) denir. Her sürecin sayfalarını RAM'deki boş yerlere koyar ve hangi sayfanın nereye konduğunu sürecin **sayfa tablosu**na yazar. Süreç bir adrese yazarken işlemci önce o tabloya bakar. Böylece aynı 20.000 adresi iki süreçte RAM'in farklı yerlerine düşer:

```
20.000 / 4.096 → bölüm 4, kalan 3.616
adres 20.000, sayfa 4'tedir (sayfa 4: 16.384 ile 20.479 arası)
tarayıcının tablosu: sayfa 4 → RAM'de 9. yer
müziğin tablosu:     sayfa 4 → RAM'de 2. yer
```




**Şekil 10.3** Tarayıcının ve müziğin 6'şar sayfası RAM'deki 4 KB'lık yerlere dağılmış; iki sayfa tablosu aynı 20.000 adresini RAM'de 9. ve 2. yerlere götürür.

RAM dolarsa ne olur? Bir süredir kullanılmayan sayfalar diske yazılır, tabloya da "diskte" diye not edilir. Süreç o sayfaya yeniden dokununca sayfa RAM'e geri getirilir; gerekirse yerine kullanılmayan başka bir sayfa diske iner. Programlar gerçek RAM'i hiç görmez; kendilerine ait, RAM'den büyük olabilen bir bellek görürler. Buna **sanal bellek** (virtual memory) denir. Disk RAM'den çok yavaş olduğu için RAM yetmeyince sayfalar sık sık gidip gelir ve bilgisayar ağırlaşır.

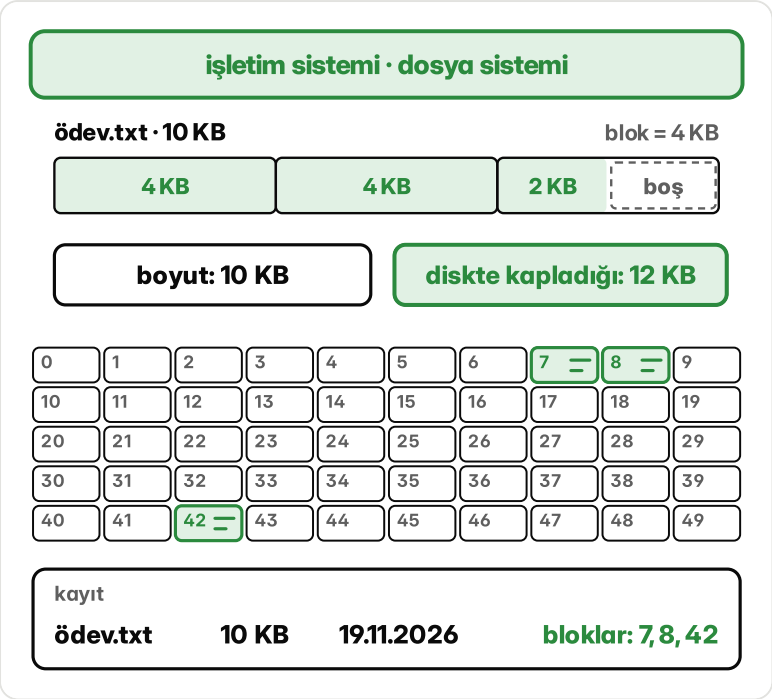
Yüzlerce program birkaç çekirdekte, birbirine karışmadan böyle çalışır: çekirdeği sırayla, belleği sayfa sayfa işletim sistemi paylaştırır.

## Dosya nerede?

videoda 55:16

Disk, dosya nedir bilmez; onun gözünde yalnızca numaralı küçük bloklar vardır. 1 TB'lık bir diskte 4 KB'lık bloklardan yaklaşık 244 milyon tane bulunur (1.000.000.000.000 bayt / 4.096 bayt ≈ 244.140.625). Peki kaydettiğin ödevi bilgisayar milyonlarca blok arasında nasıl yeniden bulur?

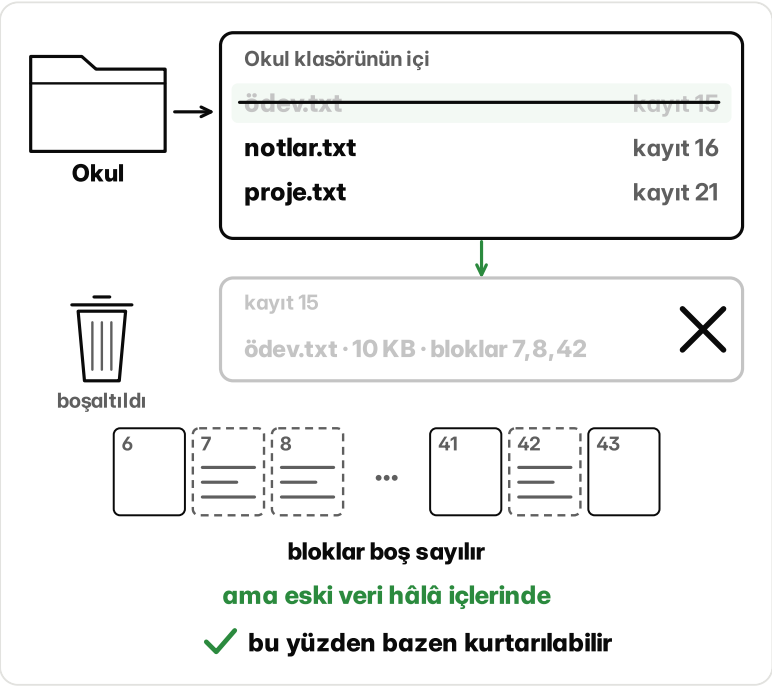
Bunu işletim sisteminin **dosya sistemi** (file system) yapar. Diyelim her blok 4 KB, ödevin 10 KB. Ödev 3 bloğa bölünür: 4 + 4 + 2 KB; son blokta 2 KB boş kalır. Dosya blok blok yer kapladığı için dosya özelliklerinde boyut 10 KB, diskte kapladığı yer 12 KB görünür. Bloklar yan yana olmak zorunda değildir: 7., 8. ve 42. blok olabilir. Her dosya için bir kayıt tutulur: adı, boyutu, tarihi ve hangi bloklarda durduğu.




**Şekil 10.4** 10 KB'lık ödev.txt 4 + 4 + 2 KB'lık parçalara bölünüp 7, 8 ve 42 numaralı bloklara yazılmış; kayıtta ad, boyut, tarih ve bloklar var.

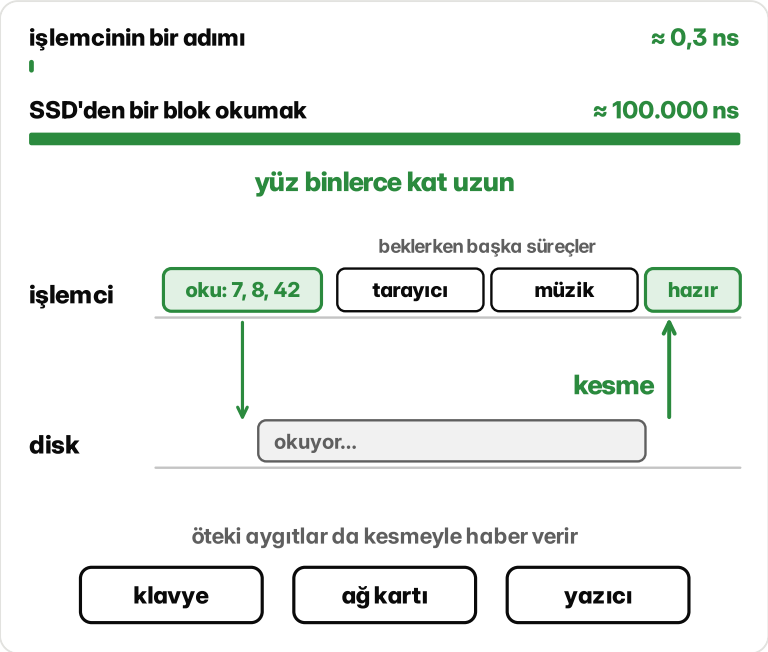
**Klasör** de aslında bir dosyadır; içinde dosyaların adları ve kayıtlarının yeri yazar. Okul klasöründe ödev.txt'nin karşısında kayıt 15, notlar.txt'nin karşısında kayıt 16 yazar. Ödevi açınca önce klasörde kaydı bulunur, kayıttan bloklar öğrenilir, sonra bloklar okunur.

Geri dönüşüm kutusunu boşaltınca bile bloklar hemen silinmez; yalnız kayıt kaldırılır ve bloklar boş sayılır. Üstlerine yeni bir dosya yazılana kadar eski veri orada kalabilir; silinen dosyalar bazen bu yüzden kurtarılabilir.




**Şekil 10.5** Kutu boşaltılınca ödev.txt'nin kaydı kaldırılmış; 7, 8 ve 42 numaralı bloklar boş sayılıyor ama eski veri hâlâ içlerinde.

Diskten blok okumak, işlemcinin hızına göre çok uzun sürer. İşlemcinin bir adımı yaklaşık 0,3 nanosaniye (bir nanosaniye saniyenin milyarda biridir), SSD'den bir blok okumak yaklaşık 100.000 nanosaniyedir: yüz binlerce kat uzun. İşletim sistemi diske hangi blokları istediğini söyler, beklerken işlemciyi başka süreçlere verir. Disk işini bitirince işlemciye haber verir; bu habere **kesme** (interrupt) denir. Klavye, ağ kartı ve yazıcı gibi öteki aygıtlarla da iş böyle yürür.




**Şekil 10.6** İşlemcinin bir adımı 0,3 ns, SSD'den bir blok okumak 100.000 ns; disk okurken işlemci tarayıcı ve müziği çalıştırır, disk bitince kesmeyle haber verir.

Yeni bir yazıcı taktığında önce sürücüsü kurulur, çünkü her aygıt farklı komutlarla konuşur: bir marka yazıcı PCL, öbürü PostScript adlı komut dilini anlar. **Sürücü** (driver), işletim sistemiyle aygıt arasında ikisinin dilini bilen küçük bir programdır. Yazı programı yazıcının markasını bilmez; işletim sistemine "yazdır" der, gerisini sürücü halleder.

İşletim sistemine iki yoldan iş yaptırırsın. **Grafik arayüz**de ödevi fareyle sürükleyip Okul klasörüne bırakırsın; **komut satırı**nda aynı işi tek satır yazarak yaparsın. İkisi de işletim sistemine aynı isteği iletir: bu dosyayı şu klasöre taşı. Komut satırı zor görünür ama tekrar eden işte hızlıdır: 100 dosyayı tek tek sürüklemek yerine tek satırla taşırsın. Windows komut isteminde ilk satır ödevi taşır, ikincisi adı .txt ile biten bütün dosyaları:

```
> move ödev.txt C:\Okul
> move *.txt C:\Okul
```

Bir dosyayı kaydetmekten yazıcıdan çıktı almaya kadar her işte işletim sistemi aradadır: dosya adlarını bloklara, istekleri aygıtların diline o çevirir.

**Özet**

- İşletim sistemi programlarla donanım arasında durur; işlemciyi ve belleği paylaştırır, dosyaları düzenler, aygıtları yönetir.
- Çalışan programa süreç denir; çekirdek yalnız hazır süreçler arasında kısa dilimlerle el değiştirir, buna zaman paylaşımı denir.
- RAM 4 KB'lık sayfalara bölünür; her sürecin sayfa tablosu aynı adresi RAM'in farklı yerlerine götürür.
- Sanal bellekte kullanılmayan sayfalar diske iner; RAM yetmezse bilgisayar ağırlaşır.
- Dosya sistemi her dosyanın adını, boyutunu ve bloklarını bir kayıtta tutar; silmek kaydı kaldırır, veriyi hemen silmez.
- Aygıtlar işini bitirince kesmeyle haber verir; sürücüler isteği aygıtın diline çevirir.

## Sorular

**1** **Bilgisayarda çalışmakta olan bir programa ne denir?**

A Sürücü
B Sayfa
C Süreç
D Kesme

**2** **Diskten blok okunurken işletim sistemi işlemciyi neden başka süreçlere verir?**

A İşlemciyi okuma bitene kadar bekletmek diski yavaşlattığı için
B Blok okumak, işlemcinin bir adımından yüz binlerce kat uzun sürdüğü için
C Disk aynı anda yalnızca tek bir süreçle çalışabildiği için
D Okunan veri önce sanal belleğe yazılmak zorunda olduğu için

**3** **Bir yazı programı, bağlı yazıcının markasını bilmeden nasıl çıktı alır?**

A Her yazı programı bütün yazıcıların komutlarını bilir
B Yazıcılar her programın dilini anlayacak biçimde üretilir
C Program çıktıyı diske yazar, yazıcı onu oradan kendisi okur
D Program işletim sistemine yazdır der; isteği yazıcının diline sürücü çevirir

**4** **Klavyeden bir tuşa basılmasını bekleyen bir süreç için hangisi doğrudur?**

A Sıraya girmez; tuşa basılınca yeniden hazır olur ve sıraya girer
B Çekirdekten sırasını alır ve tuşu beklerken çalışmayı sürdürür
C İşletim sistemi onu kapatır, tuşa basılınca yeniden açar
D Bütün sayfaları diske taşınır ve tablodan silinir

**5** **RAM dolduğunda işletim sistemi ne yapar?**

**A** Yeni bir programın açılmasına izin vermez

**B** Bir süredir kullanılmayan sayfaları diske yazar ve tabloya diskte diye not eder

**C** En eski süreci kapatarak yer açar

**D** Sayfaların boyunu küçülterek daha çok sayfa sığdırır

**6** **Geri dönüşüm kutusunu boşalttığında silinen dosyanın bloklarına ne olur?**

**A** Blokların içi hemen sıfırlarla doldurulur

**B** Bloklar diskin sonuna taşınır

**C** Bloklar boş sayılır; üstlerine yeni veri yazılana kadar eski veri kalabilir

**D** Bloklar, dosyanın kaydıyla birlikte klasörde durmaya devam eder

**7** **Blok boyu 4 KB olan bir diskte 9 KB'lık bir dosya diskte kaç KB yer kaplar?**

Cevap: _____

**8** **Sayfa boyu 4 KB (4.096 bayt) ve sayfalar 0'dan numaralanıyor. 13.000 numaralı adres kaç numaralı sayfadadır?**

Cevap: _____

# 11 Bilgisayar ağları

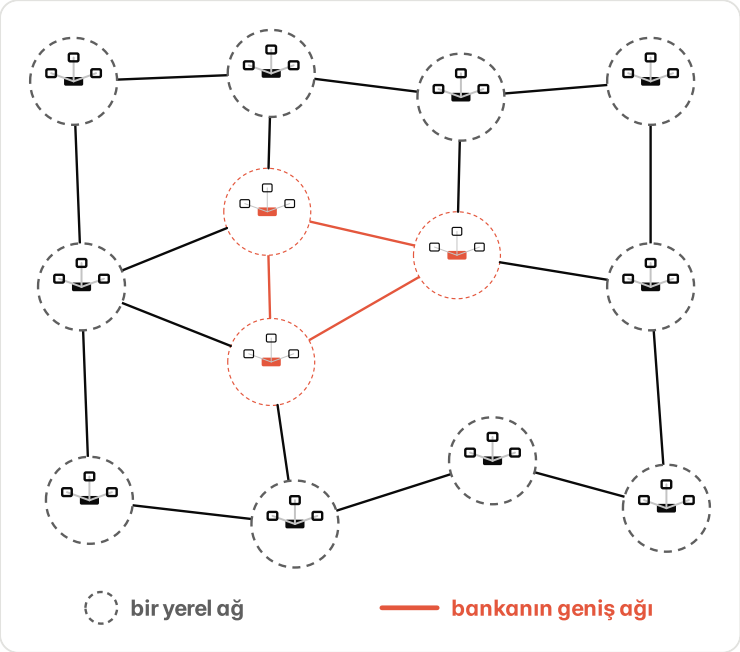
Bu bölümde cihazların ağda birbirini nasıl bulduğunu, verinin internette paket paket nasıl yol aldığını ve bir web sayfasının senin ekranına nasıl geldiğini göreceksin.

## Doğru telefonu bulmak

videoda 57:07

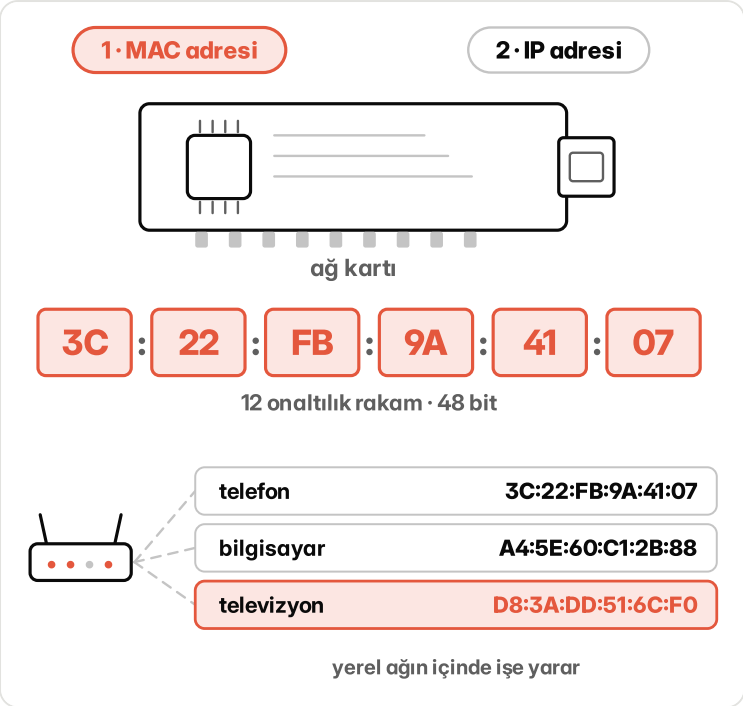
İnternete bağlı milyarlarca cihaz var. Bir mesaj, bunların arasından tam senin telefonunu nasıl bulur? Cevap iki adreste; ama önce ağın ne olduğuna bakalım.

Evinde telefonun, bilgisayarın ve televizyonun aynı modeme bağlıdır; kimi kabloyla, kimi kablosuz. Birbiriyle veri alışverişi yapabilen cihazların böyle bağlanmasına **ağ** denir. Bir ev, okul ya da ofis büyüklüğündeki ağa **yerel ağ** (LAN) denir. Şehirleri ve ülkeleri birbirine bağlayan ağlara **geniş ağ** (WAN) denir; bir bankanın İstanbul, Ankara ve İzmir şubelerini bağlayan ağ gibi. **İnternet** ise sayısız yerel ve geniş ağın birbirine bağlanmasıyla oluşur; yani ağların ağıdır.



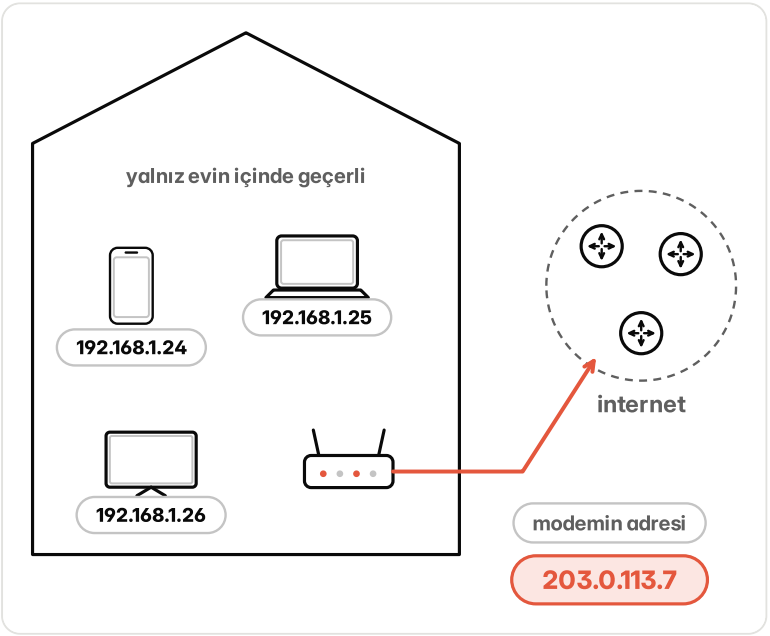

**Şekil 11.1** Kesikli dairelerle gösterilen birçok yerel ağ birbirine bağlı; aralarında bankanın 3 şubesini bağlayan geniş ağ vurgulu.

Doğru cihazı bulmak için her cihazın iki adresi vardır. Birincisi **MAC adresi**dir. Cihazı ağa bağlayan parçaya **ağ kartı** denir; üretici her karta ayrı bir numara verir. Bu numara 12 onaltılık rakamla yazılır, örneğin 3C:22:FB:9A:41:07. Her onaltılık rakam 4 bit olduğu için MAC adresi 12 × 4 = 48 bittir. Bu adres yerel ağın içinde işe yarar: modem evdeki telefonu, bilgisayarı ve televizyonu MAC adresleriyle ayırt eder.




**Şekil 11.2** Ağ kartı ve MAC adresi 3C:22:FB:9A:41:07 (12 onaltılık rakam, 48 bit); altta modemin evdeki 3 cihazı MAC adresleriyle ayırt ettiği tablo.

İkincisi **IP adresi**dir. Bunu cihaza, bağlandığı ağ verir: evde modem, kafede kafenin ağı. Aynı telefon evde 192.168.1.24, kafede 10.0.0.57 adresini alabilir. IP adresi noktalarla ayrılmış 4 sayıdır; her biri 8 bittir, yani 0 ile 255 arasında bir değer alır (00000000 = 0, 11111111 = 255). Bu yüzden 10.0.0.57 geçerli bir adrestir, 10.0.0.300 değildir. Evdeki adresler yalnızca evin içinde geçerlidir; internete bütün ev, modemin kendi adresiyle çıkar.




**Şekil 11.3** Evdeki cihazların 192.168.1.24, 192.168.1.25 ve 192.168.1.26 adresleri yalnız evin içinde geçerli; internete ev, modemin 203.0.113.7 adresiyle çıkar.

İkisi birlikte bir kargo gibi çalışır. Kargo şehirden şehre adrese bakarak gider ve apartmanı bulur: "Gül Sokak No 12, Ankara". Bu, IP adresidir. Apartmanda kurye zillerdeki isimlere bakıp doğru daireyi bulur: "Kaya". Bu da MAC adresidir.

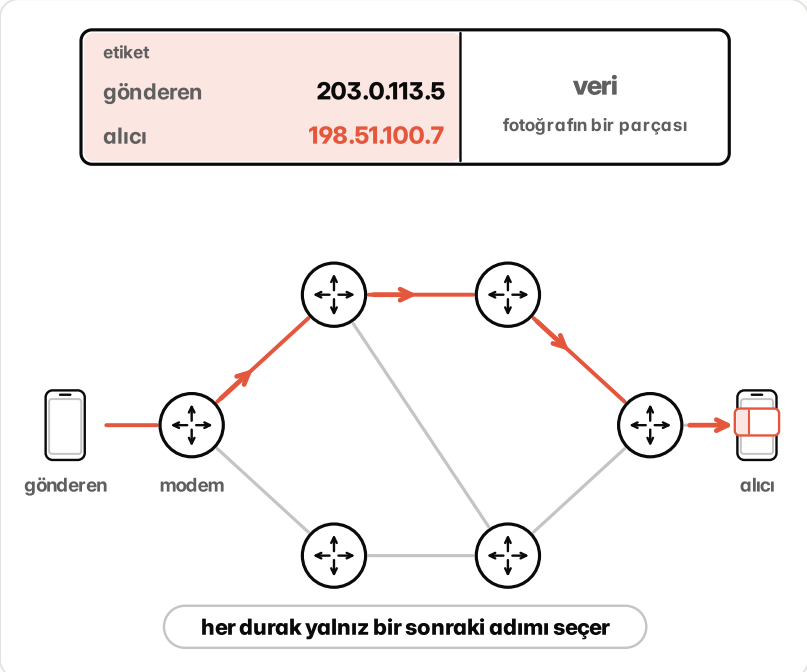
Evde bir video izlediğinde veri sana böyle ulaşır: internette IP adresiyle modemine kadar gelir, evin içinde MAC adresiyle cihazını bulur.

## Bir fotoğrafın yolculuğu

videoda 58:35

Arkadaşına bir fotoğraf gönderdiğinde, o fotoğraf yola tek parça hâlinde mi çıkar? Hayır; **paket** denen küçük parçalara bölünür. Bir paket en çok 1.500 bayt kadardır ve bunun bir kısmını paketin etiketleri kaplar. Bu yüzden 3 megabaytlık bir fotoğraf 2.000'den fazla paket eder: 3.000.000 / 1.500 = 2.000, etiketler yüzünden biraz daha fazla. Neden bölünür? Çünkü hatlar ortaktır: büyük bir dosya hattı uzun süre tek başına kaplardı, öbürleri beklerdi; küçük paketler ise başkalarınınkiyle sırayla aynı hattan geçer.

Her paketin başında bir etiket vardır: gönderenin ve alıcının internetteki adresi, yani IP adresi; örneğin gönderen 203.0.113.5, alıcı 198.51.100.7. Yolda **yönlendirici** (router) denen cihazlar durur; evindeki modem de bu işi görür. Her yönlendirici alıcının adresine bakar ve paketi hedefe daha yakın bir komşusuna verir. Yolun tamamını bilen yoktur; her biri yalnızca bir sonraki adımı seçer.




**Şekil 11.4** Etiketinde gönderen 203.0.113.5 ve alıcı 198.51.100.7 yazan paket, modemden yönlendiricilere durak durak ilerleyip alıcıya ulaşır.

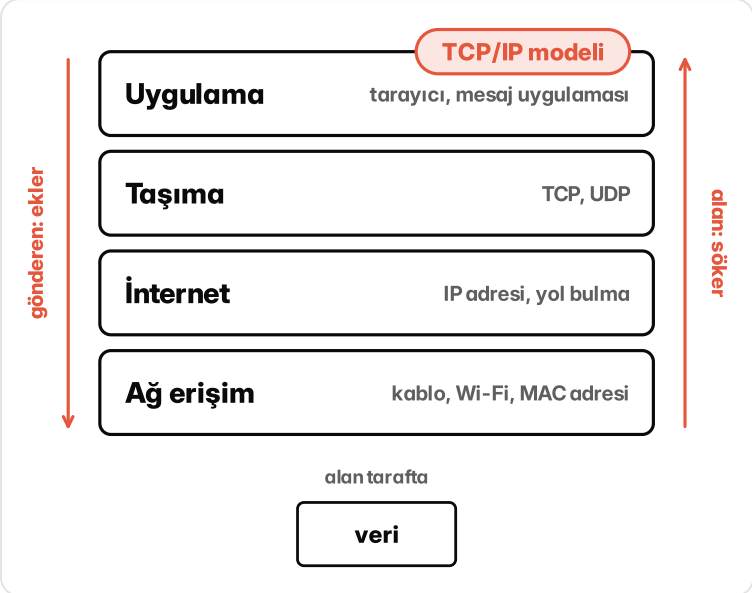
Bir hat tıkanır ya da koparsa sonraki paketler başka yoldan gider. Bu yüzden paketler karışık sırayla varabilir, örneğin 1, 3, 2, 5, 6; bazıları da yolda kaybolabilir, örneğin 4.

Bunları düzenlemek için ağdaki iş katmanlara bölünür. Her katmanın tek bir işi ve **protokol**ü, yani iki tarafın önceden anlaştığı kuralları vardır. Aşağıdan yukarı 4 katman:

- Ağ erişim katmanı aynı ağdaki iki cihaz arasında taşır; kablo, kablosuz bağlantı (Wi-Fi) ve MAC adresi buradadır.
- İnternet katmanı IP adresiyle ağlar arasında yol bulur.
- Taşıma katmanı verinin karşıdaki programa nasıl teslim edileceğini belirler.
- Uygulama katmanı tarayıcı gibi programların konuştuğu dildir.

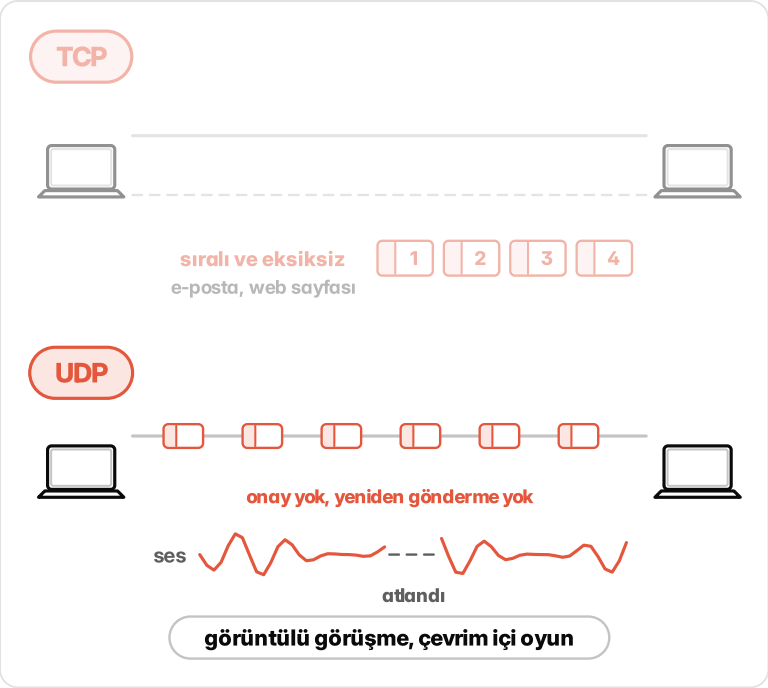
Gönderirken her katman kendi etiketini ekler, alan tarafta her katman kendi etiketini söker. Bu 4 katmanlı düzene **TCP/IP modeli** denir.

```
gönderen:  veri → [TCP][veri] → [IP][TCP][veri] → [MAC][IP][TCP][veri]
alan:      [MAC][IP][TCP][veri] → [IP][TCP][veri] → [TCP][veri] → veri
```




**Şekil 11.5** TCP/IP modelinin 4 katmanı: uygulama, taşıma, internet ve ağ erişim; gönderen her katmanda etiket ekler, alan söker.

Taşıma katmanında iki protokol öne çıkar. **TCP** her pakete bir sıra numarası koyar; alan taraf aldığını bildirir, gelmeyen paket yeniden gönderilir ve hepsi numarasına göre dizilir. Diyelim 1, 2, 3, 4 gönderildi ve 3 yolda kayboldu: alan taraf 1, 2 ve 4'ü aldığını bildirir, 3 yeniden gönderilir, 1, 2, 4, 3 diye gelenler 1, 2, 3, 4 diye dizilir. E-postalar ve web sayfalarının çoğu böyle gelir; tek bir bayt eksik olsa içerik bozulur. **UDP** ise onay beklemeden gönderir; kaybolanı yeniden yollamaz ama daha hızlıdır. Görüntülü görüşme ve çevrim içi oyunlar bunu kullanır; geç gelen bir ses parçası artık işe yaramaz, beklemektense atlamak iyidir.




**Şekil 11.6** Üstte TCP: numaralı paketler sıralı ve eksiksiz; altta UDP: paketler onaysız akar, kaybolan ses parçası atlanır.

Arkadaşın fotoğrafı açtığında, 2.000'den fazla paket çoktan numarasına göre dizilmiş ve yeniden tek bir fotoğraf olmuştur.

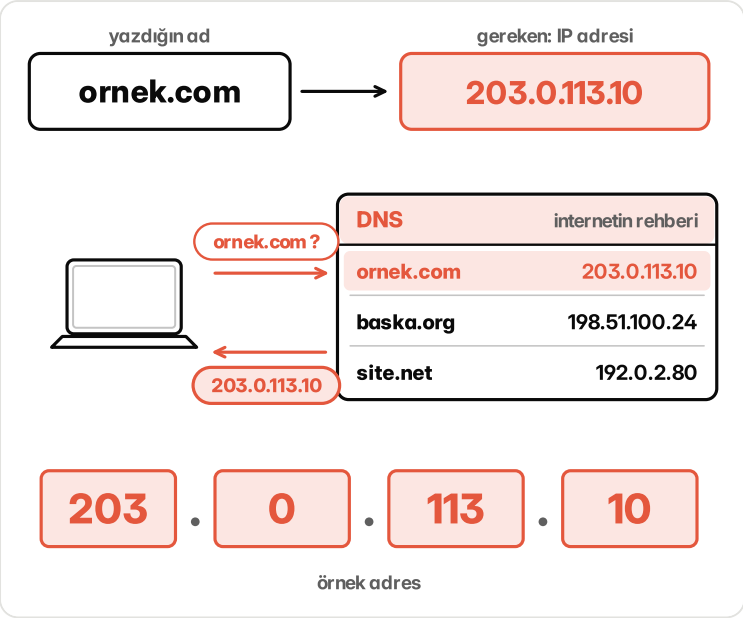
## Sayfa nereden gelir?

videoda 1:00:30

Tarayıcına bir site adı yazıp Enter'a bastığında o sayfa nereden gelir? Senin bilgisayarında değil; dünyanın bir yerinde, tam şu anda açık bekleyen başka bir bilgisayarda durur.

İnternetteki işlerin çoğu iki taraf arasında geçer. İsteyen tarafa **istemci** (client) denir; senin tarayıcın bir istemcidir. Cevap veren tarafa **sunucu** (server) denir: sayfaları ve videoları saklayan, gece gündüz açık duran bilgisayarlardır. Bir sunucu aynı anda binlerce istemciye cevap verebilir.

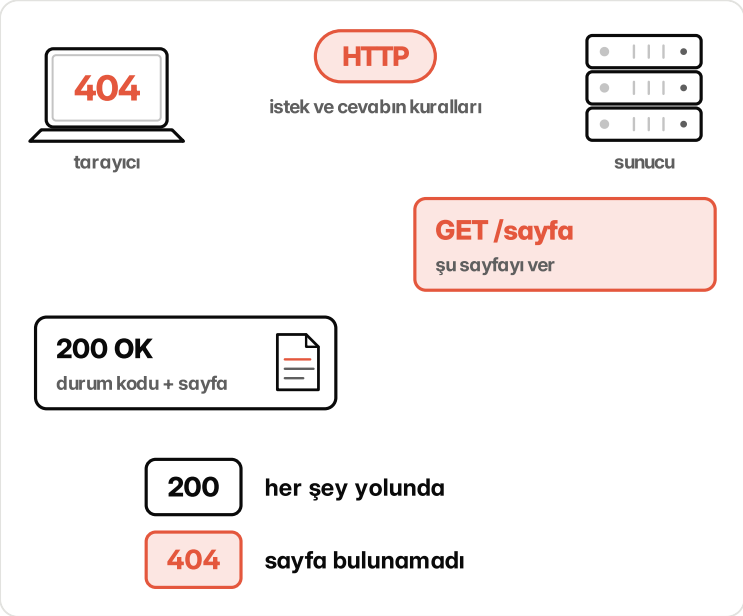
Ama sunucuya ulaşmak için onun IP adresi, yani internetteki sayısal adresi gerekir; sen ise bir ad yazdın. Adı adrese çeviren sisteme **DNS** denir; internetin telefon rehberi gibidir. Bilgisayarın önce bir DNS sunucusuna sorar: bu adın adresi ne? Cevap, noktalarla ayrılmış 4 sayıdır; örneğin ornek.com için 203.0.113.10.




**Şekil 11.7** DNS rehberinde ornek.com satırı bulunur ve bilgisayara 203.0.113.10 adresi döner.

Adres gelince tarayıcı sunucuya bir istek gönderir. Bu konuşmanın kurallarına **HTTP** denir. Tarayıcı "şu sayfayı ver" der; sunucu önce bir **durum kodu**, ardından sayfanın kendisini gönderir. 200, her şey yolunda demektir; ekranda bazen gördüğün 404 hatası da sunucunun "sayfa bulunamadı" cevabıdır.

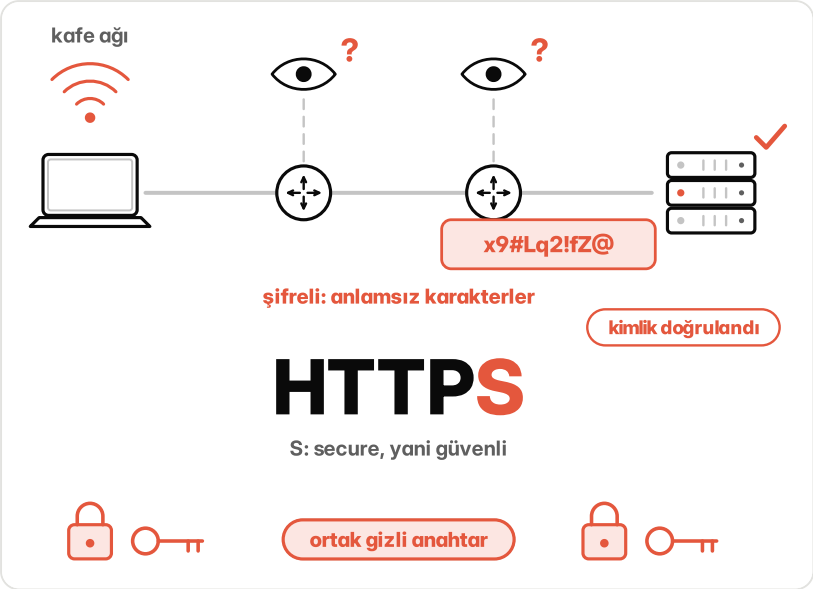
```
istek:  GET /sayfa    (şu sayfayı ver)
cevap:  200 OK        (durum kodu, ardından sayfanın kendisi)
```




**Şekil 11.8** Tarayıcı GET /sayfa isteği gönderir, sunucu 200 OK ile sayfayı yollar; 200 her şey yolunda, 404 sayfa bulunamadı demektir.

Ama HTTP'de istek de cevap da açık yazıyla gider; yoldaki her cihaz okuyabilir. Kafenin ortak kablosuz ağında şifreni böyle gönderdiğini düşün: yoldaki her yönlendirici "şifre: kedi123" satırını okuyabilir. **HTTPS**'teki S, İngilizce secure sözcüğünden gelir ve güvenli demektir. Tarayıcı önce

sunucunun kimliğini doğrular; sonra ikisi gizli bir anahtar üzerinde anlaşır ve her şey bu anahtarla şifrelenerek gider. Yoldakiler yalnızca x9#Lq2!fZ@ gibi anlamsız karakterler görür.




**Şekil 11.9** HTTPS'te kimliği doğrulanan sunucuyla ortak gizli anahtar kurulur; kafe ağındaki cihazlar yalnız x9#Lq2!fZ@ gibi anlamsız karakterler görür.

Bir siteyi her açtığında bu yolculuk çoğu zaman bir saniyeden kısa sürede baştan sona yaşanır:

- Ad adrese çevrilir: DNS'e sorulur.
- İstek şifrelenip gider: HTTPS.
- Sunucu cevabını yollar: 200 durum kodu ve sayfanın kendisi.
- Sayfa ekranında belirir: tarayıcı sayfayı çizer.

### Özet

- Yerel ağ bir ev, okul ya da ofisi, geniş ağ şehirleri ve ülkeleri bağlar; internet ağların ağıdır.
- MAC adresi ağ kartının 48 bitlik numarasıdır ve yerel ağda cihazı bulur; IP adresini bağlanılan ağ verir ve ağlar arasında yolu o buldurur.
- Veri en çok 1.500 baytlık paketlere bölünür; yönlendiriciler her paketi bir sonraki durağa iletir.
- TCP/IP modeli işi 4 katmana böler: ağ erişim, internet, taşıma ve uygulama.
- TCP paketleri numaralar, onay ister ve eksiği yeniden gönderir; UDP beklemeden gönderir ve daha hızlıdır.
- DNS adı IP adresine çevirir; HTTP istek ve cevabın kurallarıdır, HTTPS aynı konuşmayı şifreler.

## Sorular

**1** **Bir bankanın İstanbul, Ankara ve İzmir şubelerini birbirine bağlayan ağ hangi türdendir?**

A Yerel ağ
B Geniş ağ
C İnternetin kendisi
D Kablosuz ağ

**2** **Kargo benzetmesinde kuryenin zillerdeki isimlere bakıp doğru daireyi bulması hangi adrese karşılık gelir?**

A IP adresi
B Modemin adresi
C Gönderenin adresi
D MAC adresi

**3** **IP adresi noktalarla ayrılmış 4 sayıdan oluşur ve her sayı 8 bittir. Bir IP adresi toplam kaç bittir?**

Cevap:

**4** **Etiketleri yok sayalım ve her paketin 1.500 bayt veri taşıdığını düşünelim. 6.000.000 baytlık bir video kaç pakete bölünür?**

Cevap:

**5** **Paketin ağlar arasındaki yolunu IP adresine bakarak bulmak, TCP/IP modelinin hangi katmanının işidir?**

A İnternet katmanı
B Taşıma katmanı
C Uygulama katmanı
D Ağ erişim katmanı

**6** **Görüntülü görüşme uygulamaları neden çoğunlukla UDP kullanır?**

A UDP her pakete sıra numarası koyup kaybolanı yeniden gönderdiği için
B UDP paketleri yolda hiç kaybolmadığı için
C Geç gelen bir ses parçası artık işe yaramadığından, beklemek yerine atlamak daha iyi olduğu için
D UDP yalnızca yerel ağın içinde çalıştığı için

**7** **Tarayıcına ornek.com yazıp Enter'a bastığında aşağıdakilerden hangisi en önce olur?**

A Tarayıcı sunucuya GET /sayfa isteği gönderir
B Bilgisayar bir DNS sunucusuna ornek.com'un IP adresini sorar
C Tarayıcı ile sunucu gizli bir anahtar üzerinde anlaşır
D Sunucu 200 durum koduyla cevap verir

**8** **Kafenin ortak kablosuz ağında bir siteye HTTPS ile şifreni gönderiyorsun. Yoldaki bir cihaz ne görür?**

**A** Şifreni açık yazıyla görür; HTTPS yalnızca sunucunun kimliğini doğrular

**B** Hiçbir şey göremez; HTTPS paketleri yönlendiricilerden geçmez

**C** Şifreni açık yazıyla görür; şifreleme yalnızca evdeki ağda işe yarar

**D** Yalnızca anlamsız karakterler görür; veri ortak gizli anahtarla şifrelenmiştir

# Cevaplar ve çözümler

## 0 Bilgisayarın yapısı

1. **C** · Bilgi

   Bu satır, barkod numarasının fiyat listesinde aranmasıyla elde edilmiş anlamlı bir sonuçtur; böyle sonuçlara bilgi denir. Okuyucunun ürettiği ham numara ise veridir.

2. **D** · "Numarayı listede ara, fiyatı topla, ekrana yaz" komutları

   Okuyucu, ekran ve çip elle dokunabildiğin parçalardır, yani donanımdır. Donanıma ne yapacağını söyleyen komutlar ise yazılımdır.

3. **A** · Bellek

   Fiyat listesini ve sepeti bellek tutar. İşlemci arama ve toplamayı yapar ama listeyi saklamaz; giriş birimi veriyi içeri alır, çıkış birimi sonucu dışarı verir.

4. **B** · Programın komutları da verilerle aynı bellekte durur.

   Komutlar verilerle aynı bellekte saklanır. Bu yüzden aynı donanıma yeni bir iş yaptırmak için parça değiştirmek gerekmez; belleğe yeni bir program yüklemek yeter.

5. **C** · İşlemci komutları bellekten okuduğu ve bellek depolamadan çok daha hızlı olduğu için.

   İşlemci komutları bellekten getirir ve bellek depolamadan çok daha hızlıdır. Komutlar depolamada da zaten birler ve sıfırlar hâlinde yazılıdır; belleğe çevrilmek için değil, okunmak için gelir.

6. **4**

   Her turun sonunda sayaç 1 artar. 0'dan başlayan sayaç dört turdan sonra 0 + 4 = 4 olur; yani bir sonraki getirme 4. gözden yapılır.

7. **B** · Bunu işletim sisteminden ister; ekrana yazma işini işletim sistemi yapar.

   Uygulama donanıma doğrudan dokunmaz; ekrana bir sayı yazmak için bile işletim sisteminden ister. Donanımı yöneten ve ekranı uygulamalar arasında paylaştıran işletim sistemidir.

8. **1.001**

   6. gözde artık 1.000 var. Birinci turda 1.000 okunur, ikinci turda 1 eklenip 1.001 olur, üçüncü turda 1.001 geri yazılır, dördüncü turda ekrana gönderilir.

## 1 Sayı sistemleri

1. **C** · 2

   İkilikte basamak değerleri sağdan 1, 2, 4… diye gider. 10'daki 1, 2'ler basamağındadır: 1 × 2 + 0 × 1 = 2. Rakamlar onluktakiyle aynı görünse de değeri taban belirler.

2. **A** · F

   Onaltılıkta 10'dan 15'e kadar değerler sırayla A, B, C, D, E, F ile yazılır; 15 = F. E 14'tür, G bu sistemde yoktur. Onaltılıkta yazılmış 15 ise iki rakamlıdır ve 1 × 16 + 5 = 21 demektir.

3. **11001**

   25'i art arda 2'ye böl: 25 → 12 kalan 1, 12 → 6 kalan 0, 6 → 3 kalan 0, 3 → 1 kalan 1, 1 → 0 kalan 1. Kalanları aşağıdan yukarı oku: 11001. Sağlama: 16 + 8 + 1 = 25.

4. **D** · 60

   Basamak değerleri sağdan 1 ve 16'dır; C'nin değeri 12. 3 × 16 + 12 × 1 = 48 + 12 = 60. 3'ü 10 ile çarpmak (42) onluk basamak değerini kullanmak olur; C'yi 13 saymak 61'i verir.

5. **644**

   Her biri için sıra okuma, yazma, çalıştırmadır; evet 1, hayır 0. Sahibi 110 = 6, grubu 100 = 4, herkes 100 = 4. İzin 644 diye yazılır.

6. **B** · Üretici 1 TB'ı 1.000.000.000.000 bayt sayar; Windows aynı baytları 1.024'lük birimlerle böler.

   Üretici onluk sayar: 1 TB = 1.000.000.000.000 bayt. Windows bu sayıyı üç kez 1.024'e böler ve 931,3 bulur. Kaybolan yer yoktur; aynı bayt sayısı iki farklı birimle okunur.

7. **C** · Paydası 10 = 2 × 5 olduğu için; 2 ile çarpmak paydadaki 5'ten kurtulamaz.

   En sade hâliyle bir kesir ikilikte ancak paydası 2'nin kuvvetiyse biter. 1/10'un paydasında 5 çarpanı vardır ve 2 ile çarpmak onu gidermez. İkilikte kesir yazılabilir (0,75 = 0,11 gibi); küçüklük de sebep değildir, 1/16 daha küçüktür ama 0,0001 diye biter.

8. **10001**

   Sağdan: 1 + 0 = 1. 1 + 1 = 10, 0 yaz, elde 1. 0 + 1 + elde = 10, 0 yaz, elde 1. 1 + 0 + elde = 10, 0 yaz, elde 1; elde en sola iner: 10001. Sağlama: 11 + 6 = 17 = 16 + 1.

## 2 Verinin saklanması

1. **B** · 256

   Her yeni bit değer sayısını iki katına çıkarır, bu yüzden 8 bitle $2^8$ = 256 değer yazılır. 255, 8 bitlik işaretsiz sayının en büyük değeridir; 0 da bir değer olduğu için 0'dan 255'e kadar 256 değer vardır.

2. **C** · A harfini ASCII'deki gibi 1 baytla, ş harfini 2 baytla yazar.

   UTF-8, Unicode numaralarını baytlara yazar: A eskisi gibi 01000001 (1 bayt), ş 2 bayt, bir Çince karakter 3 bayt tutar. 7 bitlik, 128 karakterlik tablo ASCII'dir; ş için 254 numarasını kullanan da eski Türkçe tablodur.

3. **D** · −8'den +7'ye

   4 bitle 16 değer yazılır. 0 ile başlayan 0000 … 0111 sayıları 0'dan 7'ye, 1 ile başlayan 1000 … 1111 sayıları −8'den −1'e karşılık gelir. Sıfırın tek yazımı olduğu için eksi tarafta bir sayı fazladır; −7'den +7'ye yalnız 15 değer ederdi.

4. **1101**

   3 = 0011. Bitleri ters çevir: 1100. 1 ekle: 1101. Sağlama: 0011 + 1101 = 10000; taşan 1 düşer, 0000 kalır.

5. **A** · 0,1 ve 0,2 ikilikte tam yazılamadığı için bellekte en yakın değere yuvarlanmıştır; toplam da yeniden yuvarlanır.

   0,1 ikilikte 0,000110011… diye hiç bitmez; mantis kesilir ve sayı en yakın değere yuvarlanır. Bellekteki 0,1 ve 0,2 biraz büyük olduğu için toplam, bellekteki 0,3'ün bir adım üstüne düşer. Ekranda görülen 0,30000000000000004 zaten 64 bitlik çift duyarlıkta çıkar.

6. **B** · 01101000

   ÖZEL VEYA'da bitler farklıysa sonuç 1'dir, bu yüzden maskedeki tek 1 baytın sağdan 3. bitini ters çevirir: 1 olan bit 0 olur ve 01101000 çıkar. 01101100 VEYA işleminin, 00000100 VE işleminin, 10010011 ise DEĞİL işleminin sonucudur.

**7. 20**

19 bitle $2^{19}$ = 524.288 değer yazılır; 0'dan 1.000.000'a kadar saymaya yetmez. 20 bitle $2^{20}$ = 1.048.576 değer yazılır, sayaç 0'dan 1.048.575'e kadar gidebilir; 20 bit yeter.

**8. 2764800**

Önce piksel sayısı bulunur: 1280 × 720 = 921.600. Her piksel 3 bayt olduğu için 921.600 × 3 = 2.764.800 bayt eder, yani yaklaşık 2,8 MB.

## 3 Algoritmaya giriş

**1. D** · Deneme

Deneme aşamasında sonucunu bildiğimiz bir örnek veririz; 75 çıkmazsa hata adımlardadır ve geri dönüp düzeltiriz. Anlama girdiyi ve istenen çıktıyı belirler, tasarım adımları sıralar, kodlama onları bir programlama diliyle yazar.

**2. C** · Adım açık değildir; bilgisayar hangi sayıya böleceğini bilemez.

İyi bir algoritmanın her adımı tek bir anlama gelmelidir. "Uygun sayı" belli değildir; doğrusu "toplamı 3'e böl" demektir. Adım sayısı değişmediği için algoritma yine sonludur, girdi de yine üç nottur.

**3. A** · Bilgisayar sonsuza kadar toplayıp böler, sonuç hiç yazılmaz.

Başa dön adımı her seferinde ilk adıma götürür ve sonucu yazma adımına hiç gelinmez. Algoritma durmadığı için sonlu olma özelliği bozulur. Bilgisayar adımları söylendiği sırayla yaptığı için hiçbir adımı kendiliğinden atlamaz.

**4. B** · Karar

Eşkenar dörtgen bir karardır: içinde bir soru vardır, cevabı evet ya da hayırdır ve her cevap için ayrı bir ok çıkar. Giriş ve çıkış paralelkenarla, başlangıç ve bitiş ovalle, işlem dikdörtgenle gösterilir.

**5. 6**

3. satırdaki soru: 3, 8'den büyük mü? Cevap hayır. Hayırsa "değilse" satırının altındaki 6. satır çalışır: büyük ← b, yani büyük 8 olur ve 8 yazılır.

**6. D** · 1 ile çarpmak sonucu değiştirmediği için.

Sonuç her turda sayıyla çarpılır; 1 ile başlamak ilk çarpımı değiştirmez: 1 × 4 = 4. 0 konsaydı döngü yine çalışırdı, çünkü koşul sonuca değil sayıya bakar; ama her çarpım 0 çıkardı.

**7. 5**

Döngü sayı 1'den büyük oldukça döner: sayı 6, 5, 4, 3 ve 2 iken. Toplam 5 tur; sonuç 6 × 5 × 4 × 3 × 2 = 720 olur. Sayı 1'e inince koşul yanlış olur ve döngü biter.

**8. C** · Derleyici bütün programı bir kerede çevirip hazır bir dosya çıkarır; yorumlayıcı programı satır satır çevirip çalıştırır.

Derleyici, kitabı önceden çevirip basan çevirmen gibidir: hazır bir dosya çıkarır ve her çalışmada yeniden çevrilmediği için program daha hızlı çalışır. Yorumlayıcı, okurken cümle cümle çeviren çevirmen gibidir; yazdığını hemen denersin.

## 4 Girdi-çıktı ve koşullar

**1. B** · x'teki değeri 1 artırır.

Ok bir eşitlik değil, atamadır: önce sağ taraf hesaplanır, sonuç x kutusuna konur ve eski değer silinir. x 5 ise kutuda 6 kalır. Kutu bir anda tek değer tuttuğu için eski değer saklanmaz.

2. **C** · a 7, b 7

   a ← b ile a 7 olur ve 3 silinir. b ← a çalışınca a'da artık 7 vardır, b de 7 olur. Değerleri yer değiştirmek için önce a'yı geçici bir kutuya almak gerekir.

3. **A** · A VEYA B doğrudur.

   VEYA, sorulardan en az biri doğruysa doğru verir; A doğru olduğu için A VEYA B doğrudur. VE için ikisinin de doğru olması gerekir; DEĞİL A ise doğruyu yanlışa çevirir.

4. **C** · Diğer satırı çalışır.

   Çoklu seçimde yalnız değeri tutan satır çalışır. 4 hiçbir satırın değerine uymadığı için diğer satırı çalışır.

5. **3:20**

   Dakika tam sayı bölmesinden gelir: 200 div 60 = 3. Saniye kalandan gelir: 3 × 60 = 180 ve 200 − 180 = 20 olduğu için 200 mod 60 = 20. Sonuç 3:20.

6. **19**

   Bölme ve çarpma toplamadan önce yapılır; ikisi aynı düzeyde olduğu için soldan sağa gidilir: 6 / 2 = 3, sonra 3 × 3 = 9. En son 10 + 9 = 19. Toplamayla başlamak (16 / 2 × 3) yanlışlıkla 24 verirdi.

7. **B** · Ekrana "Salon dolu" yazılır, yaş hiç sorulmaz.

   Yaş sorusu, koltuğu soran eğerin içindedir; ancak koltuk > 0 doğruysa sorulur. Koltuk 0 iken bu koşul yanlıştır, dıştaki değilse çalışır ve yalnız "Salon dolu" yazılır.

8. **D** · 66

   66 için 66 < 12 yanlış, 66 > 65 doğrudur; bilet 60 lira olur. 65 > 65 yanlıştır, bu yüzden 65 yaşındaki biri 100 lira öder; 12 de 12'den küçük olmadığı için 100 lira öder. 8 için ilk soru doğrudur: 50 lira.

## 5 Döngüler

1. **C** · Sayaç hep 0 kalır, koşul hep doğru olur ve döngü hiç bitmez.

   Artırım olmadan sayaç 0'da kalır ve 0 ≤ 10 koşulu her turda doğru çıkar. Koşul hiç yanlış olmadığı için döngü bitmez; buna sonsuz döngü denir. Program hata vermez, durmadan merhaba yazar.

2. **55**

   Toplam her turda o anki sayaç kadar büyür: 1, 3, 6, 10, 15, 21, 28, 36, 45, 55. 10. turun sonunda toplam 55 olur.

3. **9**

   Sayaç 1'den başlar ve koşul 10'dan küçük olmasını ister; gövde sayaç 1, 2, ..., 9 için çalışır. Sayaç 10 olunca 10 < 10 yanlış olur ve döngü biter. 1'den 9'a kadar 9 sayı vardır, yani 9 merhaba yazılır.

4. **B** · Koşulu sonda soran döngü (C'de do-while)

   Koşulu sonda soran döngü önce şifreyi okur, sonra koşula bakar; bu yüzden içi en az bir kez çalışır. Koşulu başta soran döngüde koşul sorulmadan önce şifre okunmuş olmalıdır, bu yüzden okuma satırı döngüden önce bir kez daha yazılır. Sayaçlı döngü uymaz, çünkü kaç deneme olacağı baştan belli değildir.

5. **D** · Turun kalan satırları atlanır; hak azalmaz ve şifre yeniden sorulur.

   Sıradaki tura geç yalnızca o turun kalan satırlarını atlar. Hak azaltma satırına gelinmediği için hak aynı kalır ve yeni turda şifre yeniden sorulur. Döngüyü tamamen bitiren komut döngüden çık komutudur.

**6. 4**

85'in 2'ye, 3'e ve 4'e bölümünden kalan 1'dir. 5'te kalan 0'dır, çünkü 85 = 5 × 17. Bölen olarak 2, 3, 4 ve 5 denenir, 5'te döngüden çıkılır: 4 tur.

**7. A** · 50

Dıştaki döngünün her turunda içteki döngü baştan sona 10 tur döner. Dıştaki 5 tur attığı için çarpma 5 × 10 = 50 kez çalışır. 15, iki tur sayısını toplamaktan gelen yaygın bir yanlıştır.

**8. B** · Yalnızca içteki döngüyü bitirir; dıştaki döngü sıradaki sayıya geçer.

Döngüden çık, içinde bulunduğu döngüyü, yani bölen arayan içteki döngüyü bitirir. Dıştaki döngü sürer ve sıradaki sayıyı denemeye geçer. Yalnızca o turu atlayan komut sıradaki tura geç komutudur.

## 6 Temel veri tipleri

**1. C** · Verinin tipi

Aynı 8 bit sayı olarak okunursa 64 + 1 = 65, karakter olarak okunursa A'dır. Bitler kendi başına bir şey söylemez; nasıl okunacaklarını verinin tipi söyler.

**2. B** · Bellekte adresi olan en küçük birim bayt olduğu için

Bellekte adresi olan en küçük birim bayttır; tek bir bite ayrı bir adres verilemez. Bu yüzden mantıksal değer genelde 1 bayt kaplar, doğru ya da yanlış için bu baytın tek biti yeterdi.

**3. 4**

Her yaş 4 bayt tutar: 4 × 1.000.000 = 4.000.000 bayt, yani 4 megabayt. Yaş için 1 baytlık tam sayı (−128 … 127) yeterdi; o zaman 1 megabayt harcanırdı.

**4. 5**

23 / 4 = 5,75 eder; ama iki tam sayının bölümü tam sayı tipindedir. Virgülden sonrası atılır, yuvarlama yapılmaz: sonuç 6 değil, 5.

**5. A** · Sayı başa sarar ve −2.147.483.648 olur.

4 bayta 2.147.483.647'den büyük tam sayı sığmaz. Sayı dolan bir sayaç gibi başa sarar ve en küçük değere, −2.147.483.648'e döner. Buna tip taşması denir; program çoğu zaman hata vermeden yanlış sayıyla devam eder.

**6. C** · 8 baytlık kesirli bir sayının 2 baytlık tam sayıya çevrilirken sınırı aşması

Yatay hızla ilgili değer 8 baytlık kesirli sayıydı ve 2 baytlık tam sayıya çevriliyordu; 2 bayta en çok 32.767 sığar. Ariane 4'ten alınan kod, yatay hızı çok daha büyük olan Ariane 5'te bu sınırı aştı; dönüşüm hata verdi ve yön ölçen sistem kapandı.

**7. D** · 4,0

Önce bölme yapılır: iki tam sayının bölümü 9 / 2 = 4 çıkar, küsurat atılır. Bu 4 kesirli kutuya ancak sonra girer ve 4,0 olur. Sonucu kesirli bir değişkene koymak kaybolan küsuratı geri getirmez.

**8. B** · kesirli(toplam) / adet

kesirli(toplam), 7'yi 7,0 yapar; karışık işlemde adet de kendiliğinden kesirliye çevrilir ve 7,0 / 2 = 3,5 bulunur. toplam / adet iki tam sayıyı böldüğü için 3 verir; sonucu tam sayıya çeviren yazımlar da küsuratı atar ve 3 verir.

## 7 Diziler ve karakter dizileri

1. **B** · 9

   İndis baştan uzaklıktır ve 0'dan başlar: ilk elemanın indisi 0, sonuncununki 10 − 1 = 9'dur. 10 indisi dizinin dışına düşer.

2. **2028**

   Adres = başlangıç + indis × 4 = 2000 + 7 × 4 = 2000 + 28 = 2028. dizi[7] baştan 7 eleman ileridedir; her eleman 4 göz olduğu için 28 göz ileri gidilir.

3. **C** · 70, 85, 95

   Kutu 70 ile başlar. 85 > 70 olduğu için 85 yazılır; 60 geçemez; 95 > 85 olduğu için 95 yazılır; 40 geçemez. Kutu yalnızca daha büyük bir not görünce değişir: 70, 85, 95.

4. **D** · 2. öğrencinin 3. dersini

   Önce satır, sonra sütun indisi yazılır ve ikisi de 0'dan başlar. Satır 1 ikinci öğrenci, sütun 2 üçüncü derstir.

5. **A** · Bilgisayar 1020 adresini hesaplar; orası dizinin dışıdır, başka bir verinin yeridir.

   Bilgisayar yine adresi hesaplar: 1000 + 5 × 4 = 1020. Dizi 1000 ile 1019 arasındadır; 1020 başka bir verinin yeridir. Java ve Python gibi diller burada hata verip durur, C gibi diller denetlemez ve yanlış sayıyı okur.

6. **8**

   merhaba 7 harflidir: m, e, r, h, a, b, a. Son harften sonra değeri 0 olan sonlandırıcı da bir göz tutar: 7 + 1 = 8 göz.

7. **C** · bal önce gelir, çünkü 2. harfte a'nın sayısı (97) e'ninkinden (101) küçüktür.

   İlk harfler aynıdır (b); kelimeler 2. harfte ayrılır. Karar ilk farklı harfte verilir: 97 < 101 olduğu için bal, bel'den önce gelir. Eşit olmaları için bütün harflerin aynı olması gerekirdi.

8. **9**

   dershane 8 harflidir; en sona yeni bir sonlandırıcı konur: 8 + 1 = 9 göz. Daha az yer ayrılırsa harfler dizinin dışına taşar ve yanındaki veriyi bozar.

## 8 Arama

1. **C** · 8 sayının hepsine

   Bilgisayar her adımda tek bir sayıyı arananla karşılaştırır. Bakmadığı tek bir sayı aranan olabileceği için "yok" diyebilmesi için listenin sonuna kadar gitmesi gerekir: 8 adım.

2. **6**

   Arama baştan başlar: 14, 52, 8, 37 ve 90 tutmaz. 6. sayı 23'tür; 6. karşılaştırmada bulunur ve arama durur.

3. **A** · 2.000

   En kötü durumda aranan en sondadır ya da hiç yoktur; her sayıya bakılır, yani sayı adedi kadar adım: 2.000. 11, sıralı listede ikili aramanın en kötü durumudur; karışık listede ikili arama kullanılamaz.

4. **D** · Sayıların küçükten büyüğe sıralı olması

   İkili arama ortadaki sayıya bakıp bir yarıyı atar; bu ancak soldakiler küçük, sağdakiler büyükse güvenlidir. Aranan sayı listede yoksa da arama çalışır: alt üstü geçince "yok" yazar.

5. **C** · 37'den büyük sayılar solda da durabilir; sol yarıyı atmak 63'ü de atabilir.

   63 > 37 olduğu için ikili arama sol yarıyı atar. Ama dizi karışıksa 63 tam da o sol yarıda durabilir; arama onu kaybeder ve "yok" diyebilir. Ortayı hesaplamak her dizide mümkündür, sorun yarının güvenle atılamamasıdır.

6. **4**

   orta ← (alt + üst) ÷ 2 = (0 + 9) ÷ 2 = 4,5. Küsurat atılır, orta 4 olur.

7. **B** · 37, 17, 26, 21

   (0 + 14) ÷ 2 = 7: 37 > 21, üst = 6. (0 + 6) ÷ 2 = 3: 17 < 21, alt = 4. (4 + 6) ÷ 2 = 5: 26 > 21, üst = 4. (4 + 4) ÷ 2 = 4: 21 bulunur; 4 adım.

8. **12**

   Her adım kalanı yarıya indirdiği için dizi 2 katına çıkınca en kötü durum yalnızca 1 adım artar. 1.000 sayıda 10, 2.000 sayıda 11, 4.000 sayıda 12 adım.

## 9 Sıralama ve karmaşıklık

1. **B** · Turda bakılan sayıların en büyüğü

   En büyük sayıya ulaşıldığı andan sonra o her karşılaştırmayı kazanır ve her adımda bir sağa geçer, böylece tur bitince en sondadır. En küçük sayı ise bir turda yalnızca bir adım sola gidebilir; yerine oturması birkaç tur sürebilir.

2. **C** · 10

   Seçmeli sıralama en küçüğü bulduğundan emin olmak için her turda kalan bütün sayılara bakar: 4 + 3 + 2 + 1 = 10. Liste sıralı olsa da bu değişmez. 4 karşılaştırma, sıralı listede eklemeli sıralamanın yaptığı sayıdır.

3. **15**

   İlk tur 5 karşılaştırma yapar, sonraki her tur bir eksik: 5 + 4 + 3 + 2 + 1 = 15. Aynı sonuç n(n − 1)/2 kuralından da çıkar: 6 × 5 / 2 = 15.

4. **A** · O(n)

   Büyük O yalnız büyüyen kısmı yazar. Sabit 5 adım liste büyüdükçe önemsizleşir; 3 katsayısı da büyüme biçimini değiştirmez, çünkü n iki katına çıkınca 3n + 5 de yaklaşık iki katına çıkar. Bu yüzden O(n) yazılır, O(3n + 5) değil.

5. **D** · Pivot hep en küçük ya da en büyük sayı çıktığında

   Pivot en küçük ya da en büyük sayıysa gruplardan biri boş kalır, öbürü yalnızca bir sayı küçülür; liste bölünmediği için yöntem yavaşlar. İki eşit grup en iyi durumdur; pivotun çoğu zaman rastgele seçilmesi de bu kötü durumu önlemek içindir.

6. **B** · Zaman O(n), bellek O(1)

   Her sayıya bir kez dokunulur, adım sayısı n ile aynı oranda büyür: zaman O(n). Liste ne kadar uzun olursa olsun tek bir geçici kutu yeter: ek bellek O(1). Sayıları yeni bir listeye kopyalasaydık ek bellek O(n) olurdu.

7. **5**

   Elde 1 ile başlanır. 3, 1 ile karşılaştırılır ve durur (1). 5, 3 ile karşılaştırılır ve durur (1). 4 önce 5'i geçer, sonra 3'ten büyük olduğu için durur (2). 8, 5 ile karşılaştırılır ve durur (1). Toplam 1 + 1 + 2 + 1 = 5.

8. **10**

   $n^2$ = 100.000 × 100.000 = 10.000.000.000 adım. Saniyede 1.000.000.000 adımla 10.000.000.000 / 1.000.000.000 = 10 saniye sürer.

## 10 İşletim sistemleri

1. **C** · Süreç

   Çalışmakta olan her programa süreç denir. Sürücü aygıtın dilini bilen programdır, sayfa belleğin 4 KB'lık parçasıdır, kesme ise aygıtın işini bitirince işlemciye verdiği haberdir.

2. **B** · Blok okumak, işlemcinin bir adımından yüz binlerce kat uzun sürdüğü için

   İşlemcinin bir adımı yaklaşık 0,3 ns, SSD'den bir blok okumak yaklaşık 100.000 ns sürer. Bu sürede boş beklemek yerine işlemci başka süreçleri çalıştırır; disk işini bitirince kesmeyle haber verir.

3. **D** · Program işletim sistemine yazdır der; isteği yazıcının diline sürücü çevirir

   Her aygıt farklı komutlarla konuşur; örneğin bir yazıcı PCL, öbürü PostScript anlar. Sürücü, işletim sistemiyle aygıt arasında ikisinin dilini bilen küçük bir programdır; yazı programı yalnızca yazdır der.

4. **A** · Sıraya girmez; tuşa basılınca yeniden hazır olur ve sıraya girer

   Bekleyen süreç sıraya girmez; çekirdek yalnızca hazır süreçler arasında paylaştırılır. Beklediği şey gelince süreç yeniden hazır olur. Birkaç çekirdeğin yüzlerce sürece yetişmesi de bu sayededir.

5. **B** · Bir süredir kullanılmayan sayfaları diske yazar ve tabloya diskte diye not eder

   Kullanılmayan sayfalar diske iner; süreç o sayfaya yeniden dokununca sayfa RAM'e geri getirilir. Programlar bu yüzden RAM'den büyük olabilen bir sanal bellek görür. Disk çok yavaş olduğu için RAM yetmeyince bilgisayar ağırlaşır.

6. **C** · Bloklar boş sayılır; üstlerine yeni veri yazılana kadar eski veri kalabilir

   Silmede yalnızca dosyanın kaydı kaldırılır ve blokları boş sayılır; veri hemen silinmez. Üstüne yeni dosya yazılana kadar orada kalabildiği için silinen dosyalar bazen kurtarılabilir.

7. **12**

   9 KB, 4 KB'lık bloklara 4 + 4 + 1 diye bölünür; 3 blok gerekir. Son blokta 3 KB boş kalsa da blok bütün olarak dosyaya ayrılır: 3 × 4 = 12 KB.

8. **3**

   13.000'i 4.096'ya böl: bölüm 3, kalan 712. Sayfa 3, 3 × 4.096 = 12.288 ile 4 × 4.096 − 1 = 16.383 arasındaki adresleri kapsar; 13.000 bu aralıktadır.

## 11 Bilgisayar ağları

1. **B** · Geniş ağ

   Şehirleri ve ülkeleri birbirine bağlayan ağa geniş ağ (WAN) denir. Yerel ağ bir ev, okul ya da ofis büyüklüğündedir; internet ise sayısız yerel ve geniş ağın birbirine bağlanmasından oluşur.

2. **D** · MAC adresi

   Şehirden şehre adrese bakarak apartmanı bulmak IP adresinin işidir. Apartmanın içinde doğru daireyi bulmak ise yerel ağın içinde doğru cihazı bulan MAC adresine benzer.

3. **32**

   4 sayının her biri 8 bit: 4 × 8 = 32 bit.

4. **4000**

   6.000.000 / 1.500 = 4.000 paket. Gerçekte her paketin bir kısmını etiketler kapladığı için paket sayısı biraz daha fazla olur.

5. **A** · İnternet katmanı

   İnternet katmanı IP adresiyle ağlar arasında yol bulur. Ağ erişim katmanı yalnızca aynı ağdaki iki cihaz arasında taşır ve MAC adresi oradadır; taşıma katmanı verinin karşıdaki programa nasıl teslim edileceğini belirler.

6. **C** · Geç gelen bir ses parçası artık işe yaramadığından, beklemek yerine atlamak daha iyi olduğu için

   UDP onay beklemez ve kaybolanı yeniden yollamaz; bu yüzden daha hızlıdır. Görüşmede geç gelen ses parçası artık işe yaramaz. Paketleri numaralayıp eksiği yeniden göndermek TCP'nin işidir.

7. **B** · Bilgisayar bir DNS sunucusuna ornek.com'un IP adresini sorar

   Sunucuya ulaşmak için önce onun IP adresi gerekir; adı adrese DNS çevirir. Anahtar üzerinde anlaşmak, istek göndermek ve 200 cevabını almak ancak adres bilindikten sonra olur.

8. **D** · Yalnızca anlamsız karakterler görür; veri ortak gizli anahtarla şifrelenmiştir

   HTTPS'te tarayıcı önce sunucunun kimliğini doğrular, sonra ikisi gizli bir anahtar üzerinde anlaşır ve her şey bu anahtarla şifrelenerek gider. Paketler yine yönlendiricilerden geçer, ama yoldakiler yalnızca anlamsız karakterler görür.

## Öğrendiğini pekiştir

Bu notlardaki bütün soruları ve daha fazlasını sitede, her cevabın hemen ardından çözümüyle birlikte çözebilirsin.

# hekzon.com/dersler

HEKZON