Bölüm 4.1: Algoritmalar

Bilgisayar ve Programlamaya Giriş

Problem | Algoritma | İzleme | Arama | Sıralama

Sercan KÜLCÜ · Giresun Üniversitesi

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/computer/slides/Bolum_04_01_Algoritmalar.html

GİRİŞ

Öğrenme Çıktıları

Tanımlama

Girdi, çıktı ve kısıtları belirlemek.

Tasarım

Çözümü açık ve sonlu adımlara ayırmak.

Doğrulama

Algoritmayı örneklerle izlemek ve test etmek.

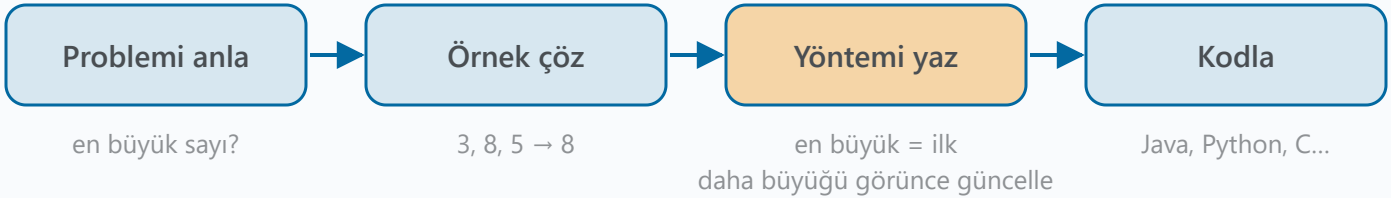
Karşılaştırma

Arama ve sıralama yaklaşımlarını değerlendirmek.

PROBLEM ÇÖZME

Önce Algoritma, Sonra Kod

- Kod, çözüm fikrinin belirli bir dildeki uygulamasıdır.
- Algoritma kurulmadan yazılan kod gereksiz denemelere ve gizli hatalara yol açabilir.
- Doğru sıra: problemi anla, örnek çöz, yöntemi yaz, sonra kodla.



Turuncu adım (algoritma) dilden bağımsızdır; aynı yöntem her dilde kodlanabilir.

PROBLEM ÇÖZME

Problem Tanımı

Amaç
Hangi sorun çözülecek?

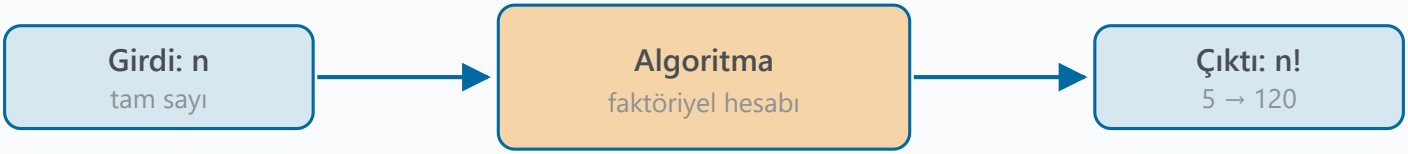
Girdi
Programa hangi bilgiler verilecek?

Çıktı
Hangi sonuç üretilecek?

Kısıt
Değerlerin sınırı ve geçerlilik koşulu nedir?

Örnek
Beklenen sonuç hangi küçük örnekle doğrulanabilir?

Hata durumu
Geçersiz veya eksik girdide ne yapılacak?

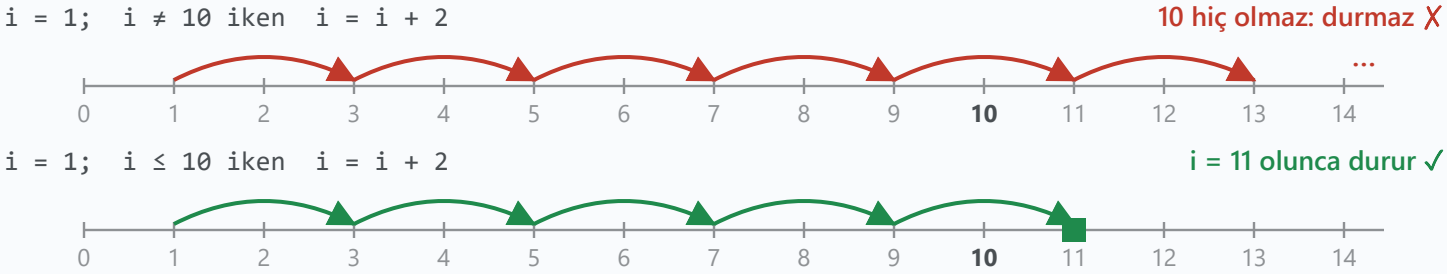


Kısıt: 0 ≤ n ≤ 20 (long tipine sığar) · n < 0 ise hata

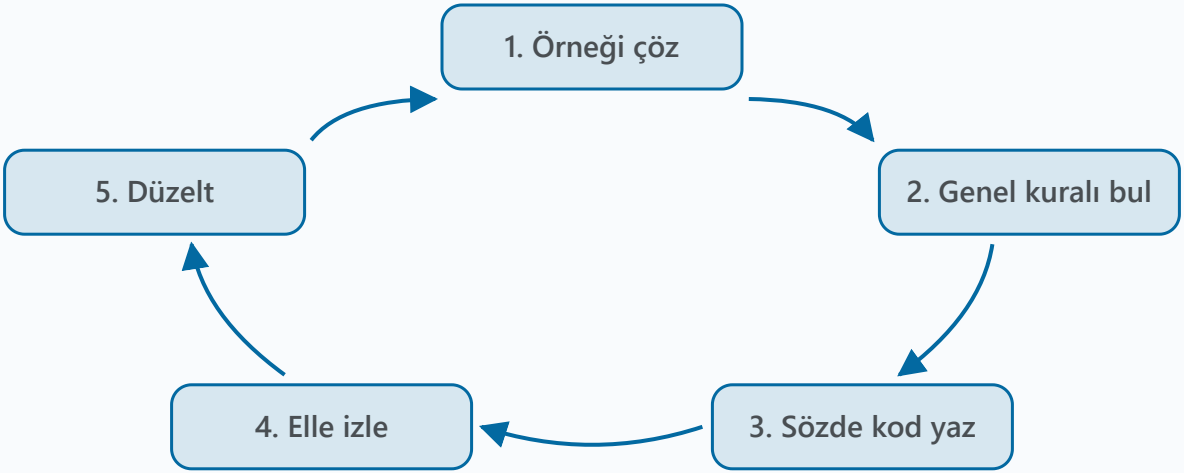
İyi Algoritmanın Özellikleri

Algoritma, belirli bir problemi çözmek için tanımlanmış sonlu adımlar dizisidir.

- Sıfır veya daha fazla girdi alır, en az bir çıktı üretir.
- Adımlar açık, sıralı ve uygulanabilir olmalıdır.
- Sonlu sayıda adımdan sonra durmalıdır.
- Geçerli girdiler için doğru sonucu üretmelidir.
- Kaynak kullanımı girdi boyutu büyüdüğünde kabul edilebilir olmalıdır.



Algoritma Tasarlama Döngüsü



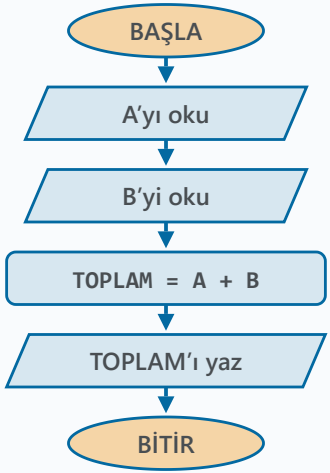
- Elle izlemede hata çıkarsa düzeltilir ve döngü yeniden döner; hata kalmayınca kodlamaya geçilir.
- Birden fazla çözüm varsa doğruluk, okunabilirlik ve verimlilik karşılaştırılır.

ALGORİTMA

Sözde Kod

Sözde kod, algoritmanın mantığını belirli bir programlama diline bağlı kalmadan açıklar.

```
BAŞLA
  A değerini oku
  B değerini oku
  TOPLAM = A + B
  TOPLAM değerini yaz
BİTİR
```



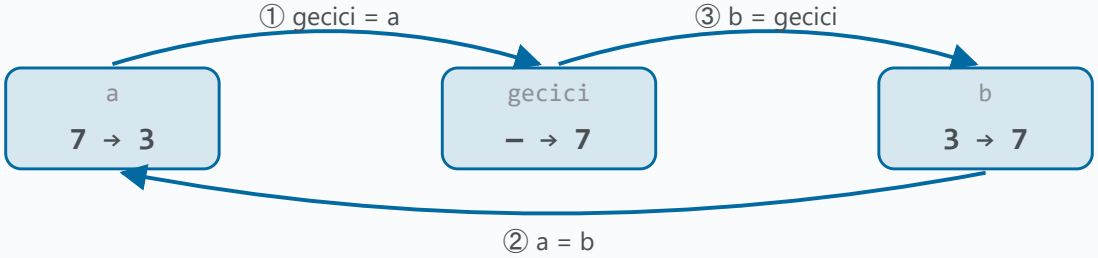
- Doğal dile yakın, kısa ve tutarlı ifadeler kullanılır.
- Girinti, koşul ve döngülerin kapsamını gösterir.
- Amaç söz dizimi değil, algoritmayı anlaşılır hale getirmektir.

ÖRNEKLER

Örnek: İki Sayıyı Sırala

```
oku a, b
eğer a > b ise
    gecici = a
    a = b
    b = gecici
yaz a, b
```

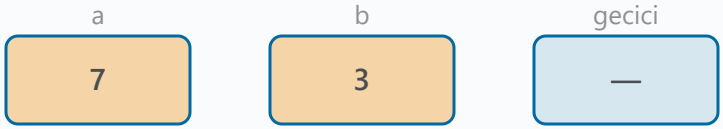
- Eşit değerlerde değişim gerekmez.
- Takas **üç atamayla** yapılır.
- Çıktı için a ≤ b koşulu sağlanmalıdır.



Neden `gecici` değişkeni? Doğrudan **a = b; b = a** yazılırsa ilk atama a'nın eski değerini siler: a = 7, b = 3 için sonuç **3 3** olur.

ÖRNEKLER · ADIM ADIM İZLEME

İki Sayıyı Sırala Adım Adım (a = 7, b = 3)



Girdiler okunur: **a = 7**, **b = 3**. Hedef: küçük olan önce yazılsın.

Adım 1 / 6

```
▶ oku a, b
2 eğer a > b ise
3     gecici = a
4     a = b
5     b = gecici
6 yaz a, b
```

DEĞİŞKENLER

a = 7 b = 3

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

İki Sayıyı Sırala Adım Adım (a = 7, b = 3)



Koşul: **7 > 3** → **doğru** → takas bloğuna girilir.

Adım 2 / 6

```
1 oku a, b
▶ eğer a > b ise
3     gecici = a
4     a = b
5     b = gecici
6 yaz a, b
```

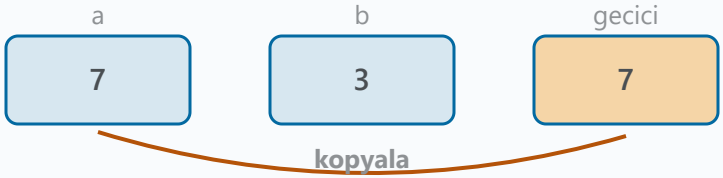
DEĞİŞKENLER

a = 7 | b = 3 | a > b = doğru

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

İki Sayıyı Sırala Adım Adım (a = 7, b = 3)



gecici = a = 7: a'nın değeri kaybolmasın diye önce yedeklenir.

Adım 3 / 6

```
1 oku a, b
2 eğer a > b ise
▶     gecici = a
4     a = b
5     b = gecici
6 yaz a, b
```

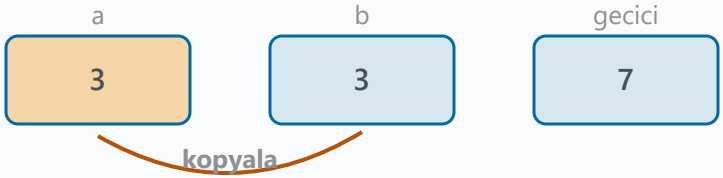
DEĞİŞKENLER

a = 7 | b = 3 | gecici = 7

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

İki Sayıyı Sırala Adım Adım (a = 7, b = 3)



a = b = 3. Eski 7 artık yalnızca gecici'de duruyor.

Adım 4 / 6

```
1 oku a, b
2 eğer a > b ise
3     gecici = a
▶     a = b
5     b = gecici
6 yaz a, b
```

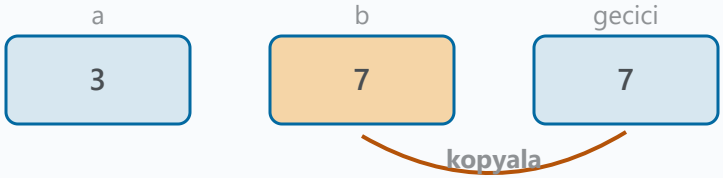
DEĞİŞKENLER

a = 3 | b = 3 | gecici = 7

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

İki Sayıyı Sırala Adım Adım (a = 7, b = 3)



b = gecici = 7. Üç atamayla değerler yer değiştirdi.

Adım 5 / 6

```
1 oku a, b
2 eğer a > b ise
3     gecici = a
4     a = b
▶     b = gecici
6 yaz a, b
```

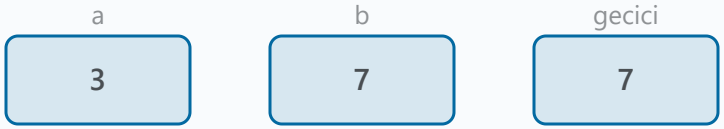
DEĞİŞKENLER

a = 3 | b = 7 | gecici = 7

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

İki Sayıyı Sırala Adım Adım (a = 7, b = 3)



Çıktı **3 7**: a ≤ b sağlandı. (a = 2, b = 9 girilseydi koşul yanlış olur, 3–5. satırlar atlanırdı.)

Adım 6 / 6

```
1 oku a, b
2 eğer a > b ise
3     gecici = a
4     a = b
5     b = gecici
▶ yaz a, b
```

DEĞİŞKENLER

a = **3** b = **7** gecici = **7**

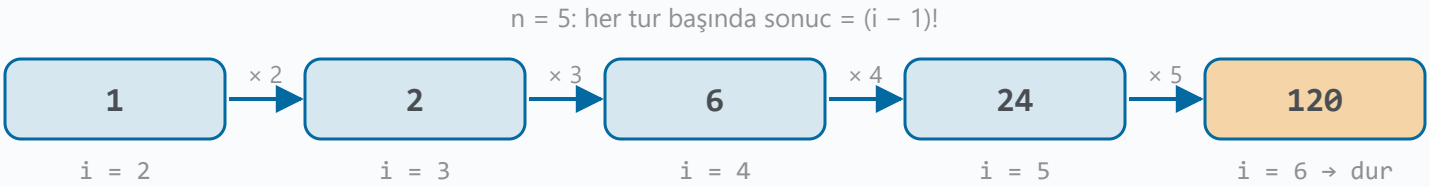
ÇIKTI

```
3 7
```

Örnek: Faktöriyel

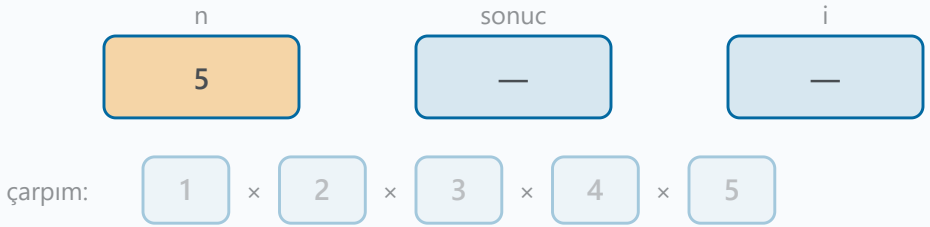
```
oku n
eğer n < 0 ise "hata" yaz ve dur
sonuc = 1
i = 2
i ≤ n iken
    sonuc = sonuc × i
    i = i + 1
yaz sonuc
```

- 0! ve 1! değeri 1'dir: döngü hiç dönmez, sonuc = 1 kalır.
- Döngü değişmezi: her tur başında **sonuc = (i − 1)!**
- Negatif girdide hata verilip durulur.



ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



n = 5 okunur. Beklenen: 5! = 1×2×3×4×5.

Adım 1 / 18

```
▶ oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

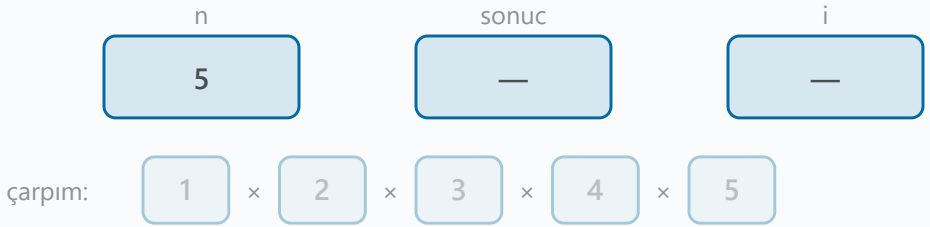
DEĞİŞKENLER

n = 5

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



Koşul: **5 < 0** → **yanlış**; hata yok, devam.

Adım 2 / 18

```
1 oku n
▶ eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

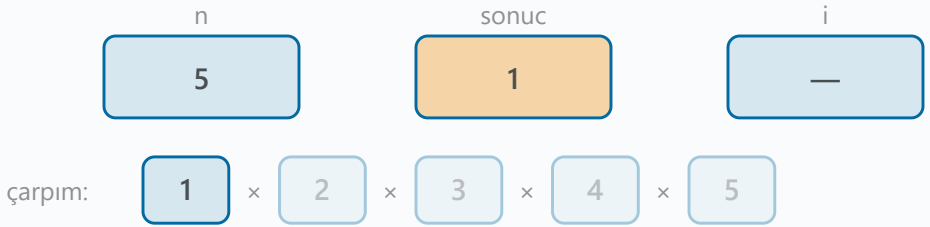
DEĞİŞKENLER

n = 5

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



Çarpım için başlangıç değeri **1** (0 olsaydı her şey 0 çıkardı).

Adım 3 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
▶ sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

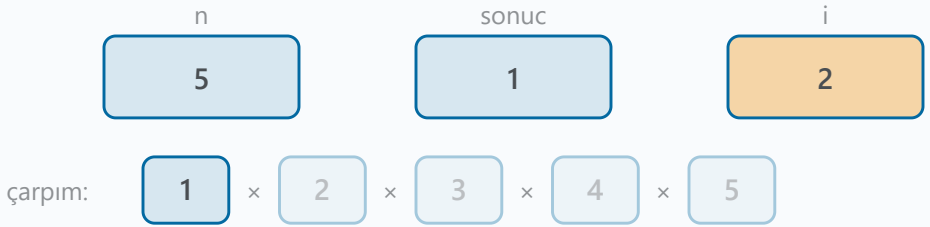
DEĞİŞKENLER

`n = 5` `sonuc = 1`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



i = 2: 1 ile çarpmak değeri değiştirmez, 2'den başlanır.

Adım 4 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
▶ i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

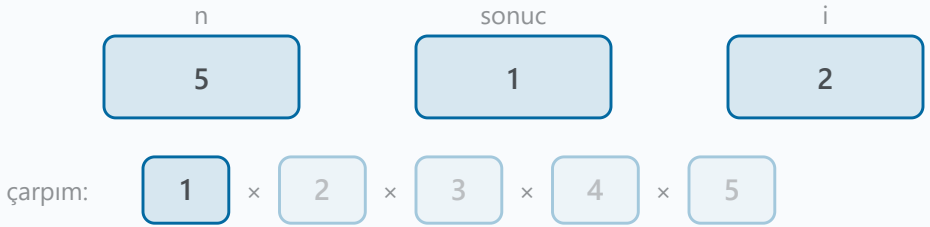
DEĞİŞKENLER

`n = 5` `sonuc = 1` `i = 2`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



Koşul: **2 ≤ 5** → **doğru**.

Adım 5 / 18

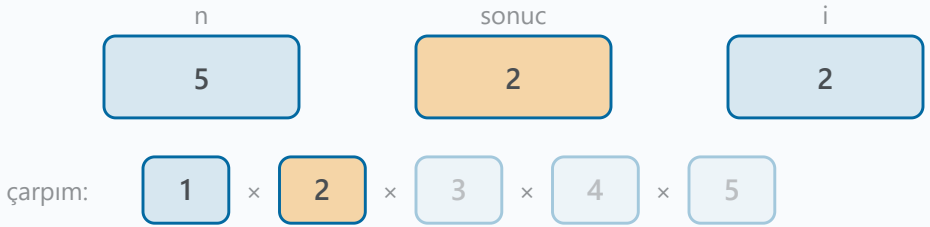
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
▶ i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 1` `i = 2`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



sonuc = 1 × 2 = 2 (= 2!).

Adım 6 / 18

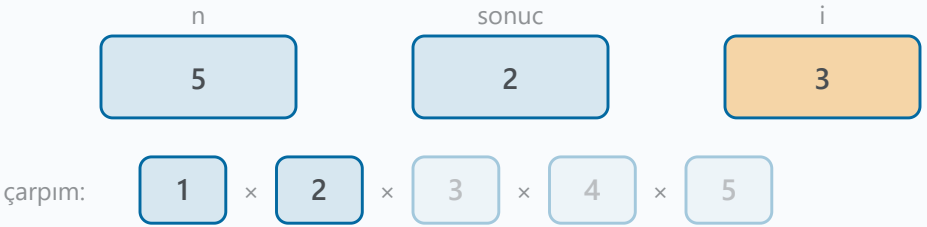
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
▶     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 2` `i = 2`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



i = 3.

Adım 7 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
▶     i = i + 1
8 yaz sonuc
```

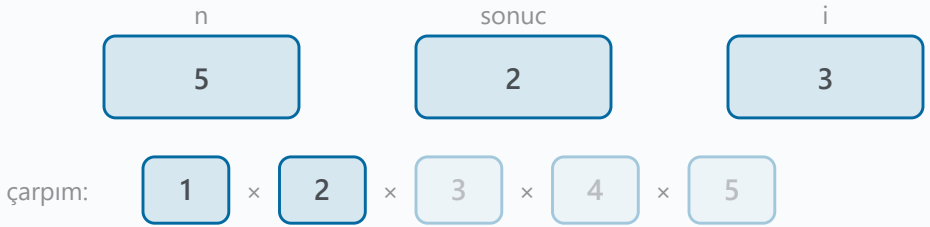
DEĞİŞKENLER

`n = 5` `sonuc = 2` `i = 3`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



Koşul: **3 ≤ 5 → doğru**.

Adım 8 / 18

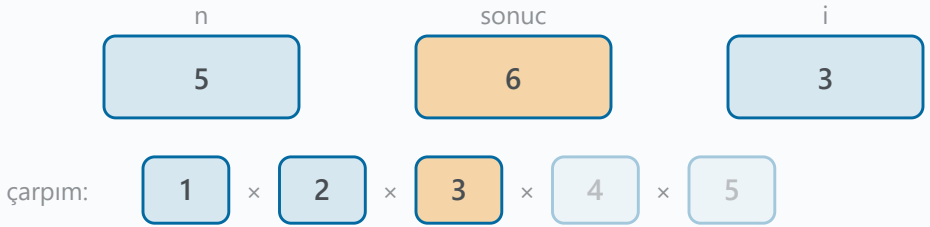
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
▶ i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 2` `i = 3`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



sonuc = 2 × 3 = 6 (= 3!).

Adım 9 / 18

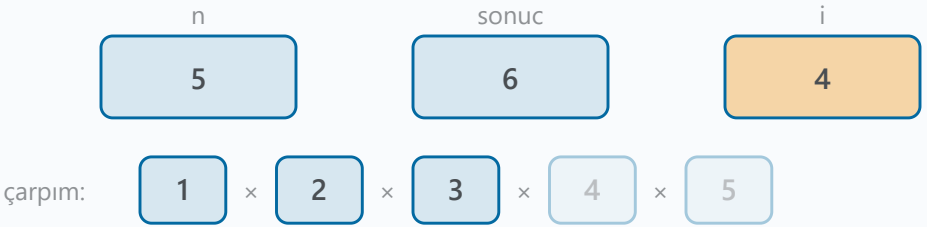
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
▶     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 6` `i = 3`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



i = 4.

Adım 10 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
▶     i = i + 1
8 yaz sonuc
```

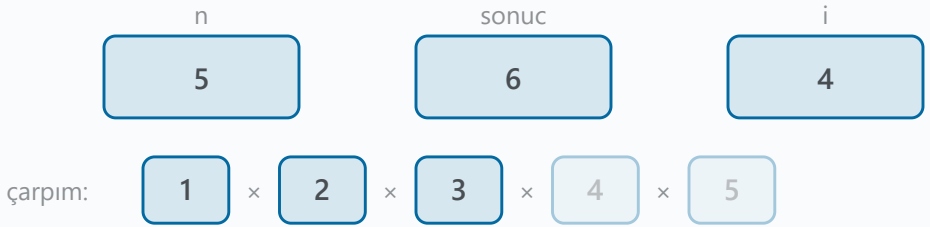
DEĞİŞKENLER

`n = 5` `sonuc = 6` `i = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



Koşul: **4 ≤ 5 → doğru**.

Adım 11 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
▶ i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

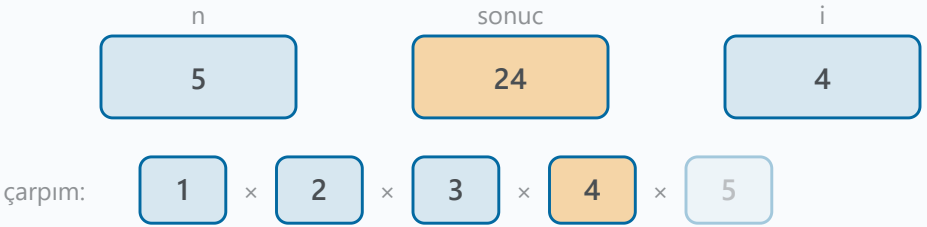
DEĞİŞKENLER

`n = 5` `sonuc = 6` `i = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



sonuc = 6 × 4 = 24 (= 4!).

Adım 12 / 18

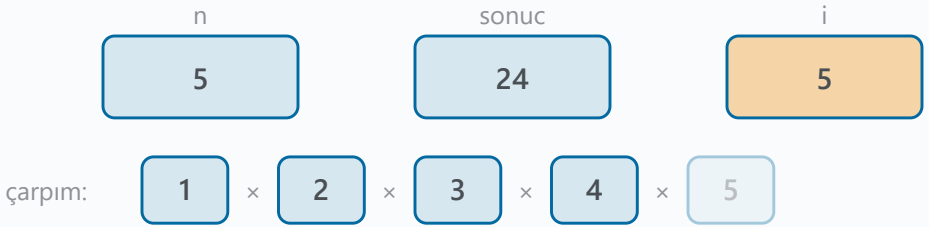
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
▶     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 24` `i = 4`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



i = 5.

Adım 13 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
▶     i = i + 1
8 yaz sonuc
```

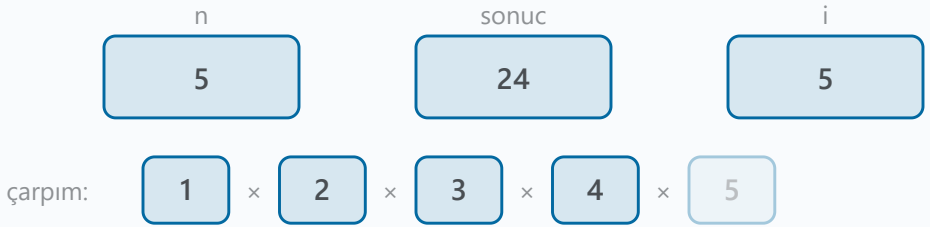
DEĞİŞKENLER

`n = 5` `sonuc = 24` `i = 5`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



Koşul: **5 ≤ 5** → **doğru**.

Adım 14 / 18

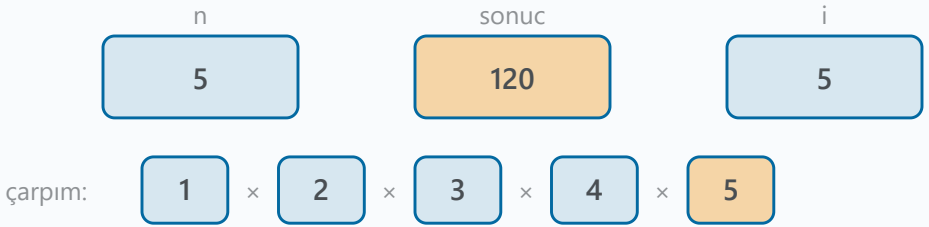
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
▶ i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 24` `i = 5`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



sonuc = 24 × 5 = 120 (= 5!).

Adım 15 / 18

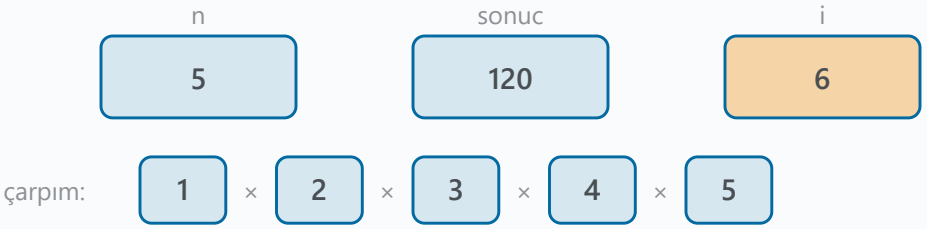
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
▶     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 120` `i = 5`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



i = 6.

Adım 16 / 18

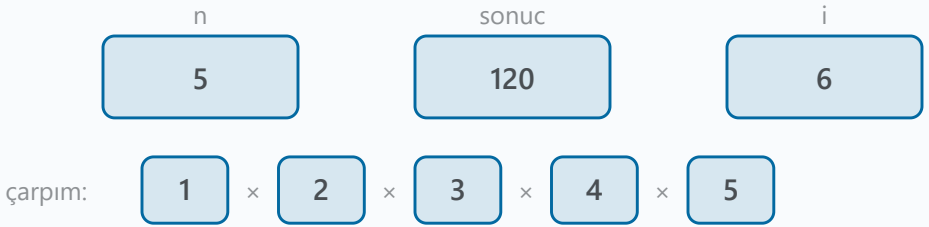
```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
▶     i = i + 1
8 yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 120` `i = 6`

ÇIKTI

Faktöriyel Adım Adım (n = 5)



Koşul: **6 ≤ 5** → **yanlış** → döngü biter.

Adım 17 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
▶ i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
8 yaz sonuc
```

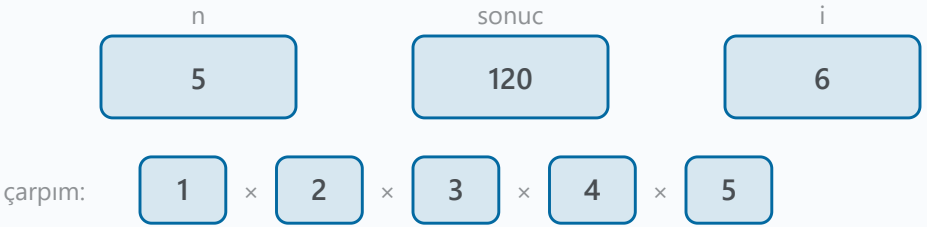
DEĞİŞKENLER

`n = 5` `sonuc = 120` `i = 6`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Faktöriyel Adım Adım (n = 5)



120 yazılır. Döngü n − 1 = 4 kez döndü.

Adım 18 / 18

```
1 oku n
2 eğer n < 0 ise "hata" yaz ve dur
3 sonuc = 1
4 i = 2
5 i ≤ n iken
6     sonuc = sonuc × i
7     i = i + 1
▶ yaz sonuc
```

DEĞİŞKENLER

`n = 5` `sonuc = 120` `i = 6`

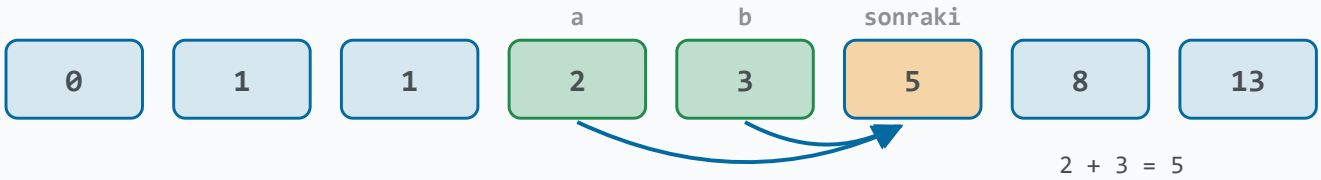
ÇIKTI

```
120
```

Örnek: Fibonacci

```
oku n
a = 0, b = 1
0'dan n-1'e kadar tekrarla
    yaz a
    sonraki = a + b
    a = b
    b = sonraki
```

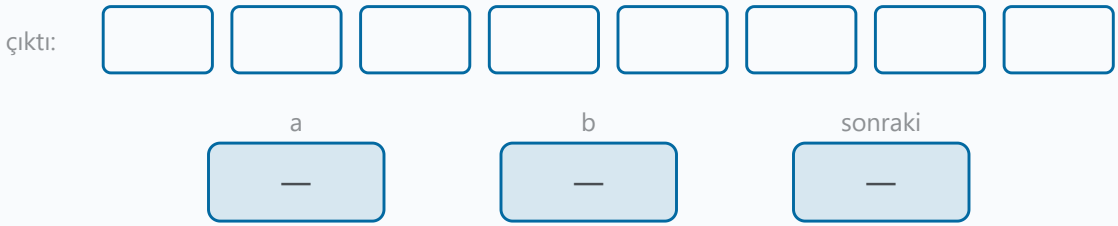
- İki önceki değer korunur: a ve b her turda bir sağa kayan pencere gibidir.
- Geçici değişken güncelleme sırasında bilgi kaybını önler.



Tur 3: a (2) yazılır, sonraki = a + b; sonra a = 3, b = 5 olur ve pencere bir sağa kayar.

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



n = 8: ilk 8 Fibonacci sayısı yazılacak.

Adım 1 / 14

```
▶ oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
7     b = sonraki
```

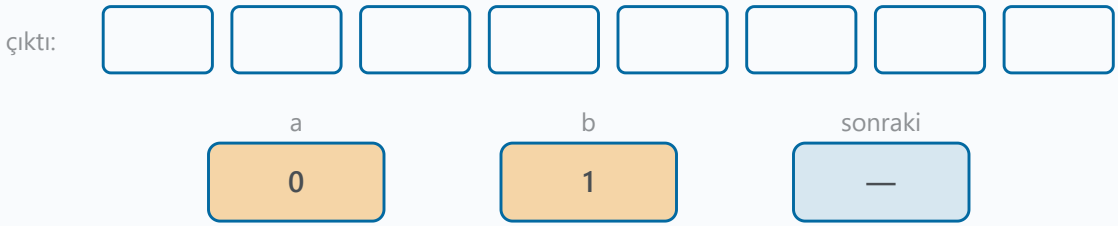
DEĞİŞKENLER

n = 8

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



Dizinin ilk iki terimi: **a = 0**, **b = 1**.

Adım 2 / 14

```
1 oku n
▶ a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
7     b = sonraki
```

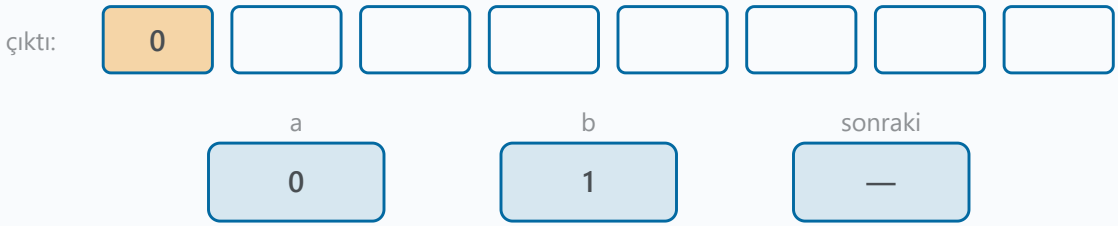
DEĞİŞKENLER

n = 8 | a = 0 | b = 1

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



Tur 0 (0…7): **a = 0** yazılır.

Adım 3 / 14

```
1 oku n
2 a = 0, b = 1
0'dan n-1'e kadar tekrarla
    yaz a
5   sonraki = a + b
6   a = b
7   b = sonraki
```

DEĞİŞKENLER

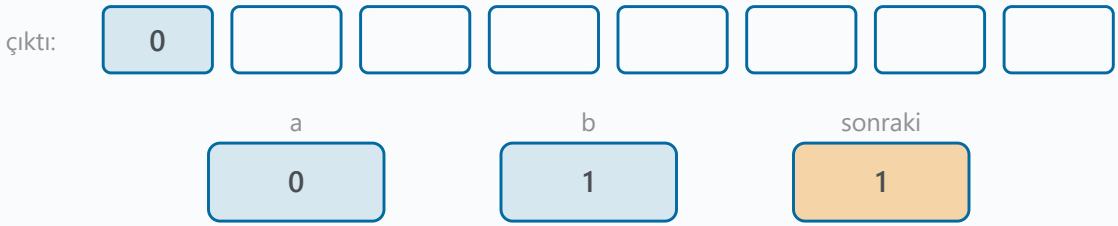
n = 8 | tur = 0 | a = 0 | b = 1 | sonraki = –

ÇIKTI

```
0
```

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



sonraki = 0 + 1 = 1.

Adım 4 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
      sonraki = a + b
6     a = b
7     b = sonraki
```

DEĞİŞKENLER

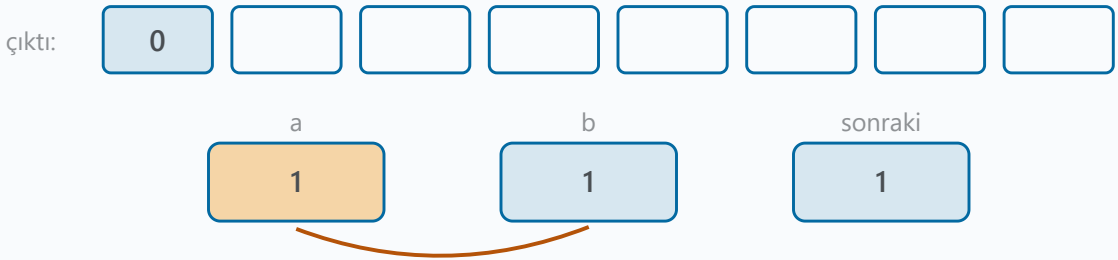
`n = 8` `tur = 0` `a = 0` `b = 1` `sonraki = 1`

ÇIKTI

```
0
```

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



a = b = 1: pencere bir kayar.

Adım 5 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

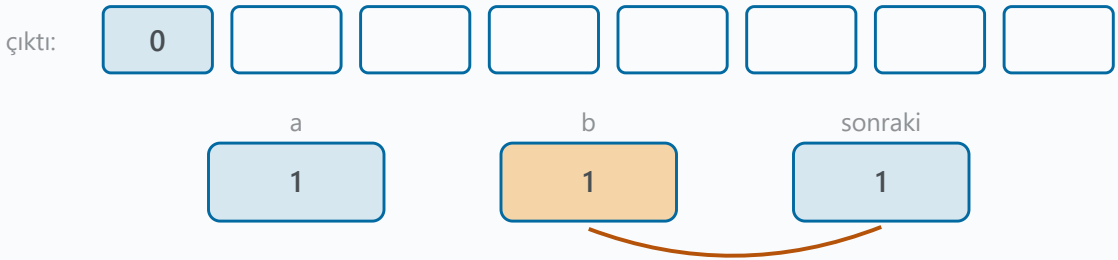
DEĞİŞKENLER

n = 8 | tur = 0 | a = 1 | b = 1 | sonraki = 1

ÇIKTI

```
0
```

Fibonacci Adım Adım (n = 8, tur 0–2)



b = sonraki = 1. Sıra önemli: önce a = b yapılmasaydı eski b kaybolurdu.

Adım 6 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

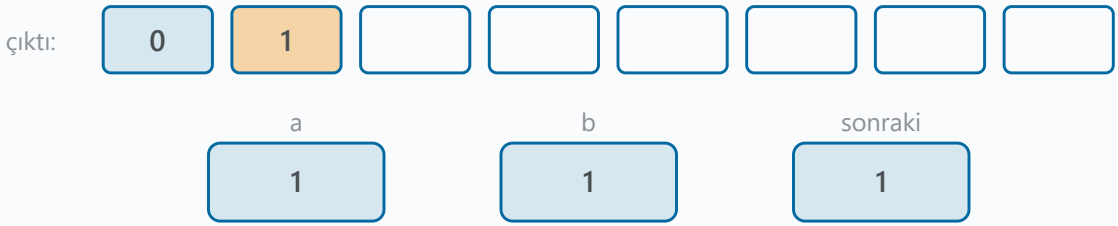
DEĞİŞKENLER

`n = 8` `tur = 0` `a = 1` `b = 1` `sonraki = 1`

ÇIKTI

```
0
```

Fibonacci Adım Adım (n = 8, tur 0–2)



Tur 1 (0...7): **a = 1** yazılır.

Adım 7 / 14

```
1 oku n
2 a = 0, b = 1
▶ 0'dan n-1'e kadar tekrarla
▶     yaz a
5     sonraki = a + b
6     a = b
7     b = sonraki
```

DEĞİŞKENLER

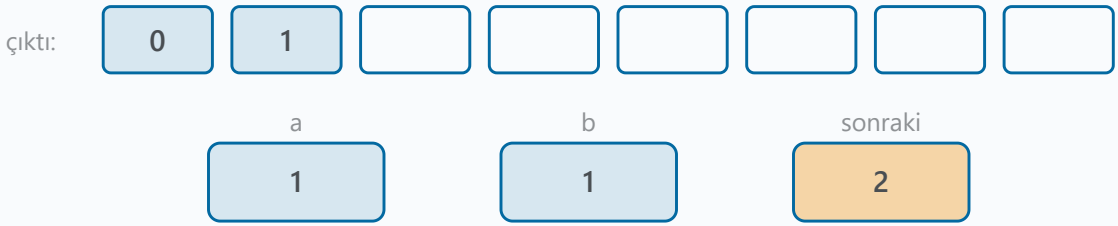
`n = 8` `tur = 1` `a = 1` `b = 1` `sonraki = 1`

ÇIKTI

```
0 1
```

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



sonraki = 1 + 1 = 2.

Adım 8 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
      sonraki = a + b
6     a = b
7     b = sonraki
```

DEĞİŞKENLER

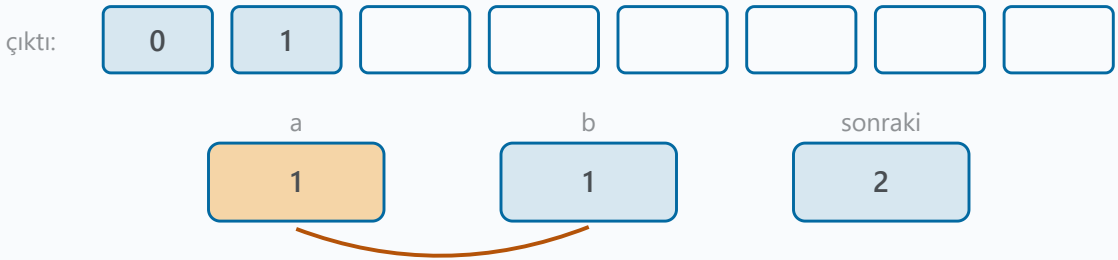
`n = 8` `tur = 1` `a = 1` `b = 1` `sonraki = 2`

ÇIKTI

```
0 1
```

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



a = b = 1: pencere bir kayar.

Adım 9 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

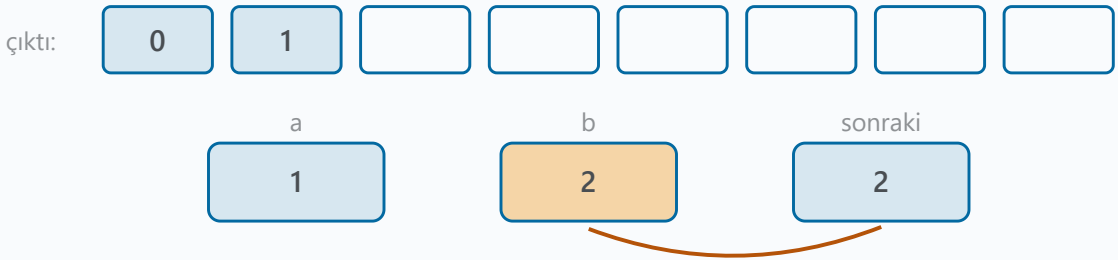
DEĞİŞKENLER

`n = 8` `tur = 1` `a = 1` `b = 1` `sonraki = 2`

ÇIKTI

```
0 1
```

Fibonacci Adım Adım (n = 8, tur 0–2)



b = sonraki = 2.

Adım 10 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

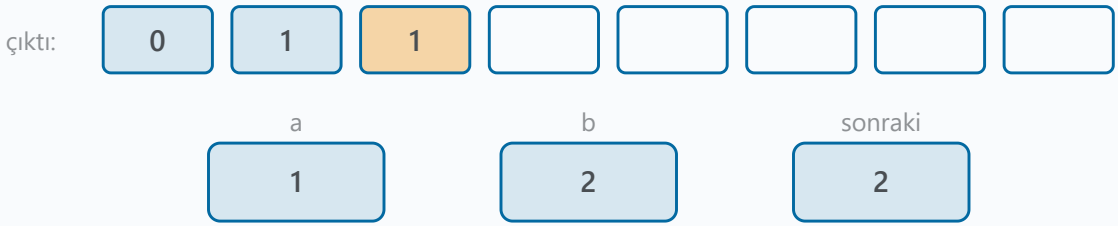
DEĞİŞKENLER

`n = 8` `tur = 1` `a = 1` `b = 2` `sonraki = 2`

ÇIKTI

```
0 1
```

Fibonacci Adım Adım (n = 8, tur 0–2)



Tur 2 (0...7): **a = 1** yazılır.

Adım 11 / 14

```
1 oku n
2 a = 0, b = 1
0'dan n-1'e kadar tekrarla
    yaz a
5   sonraki = a + b
6   a = b
7   b = sonraki
```

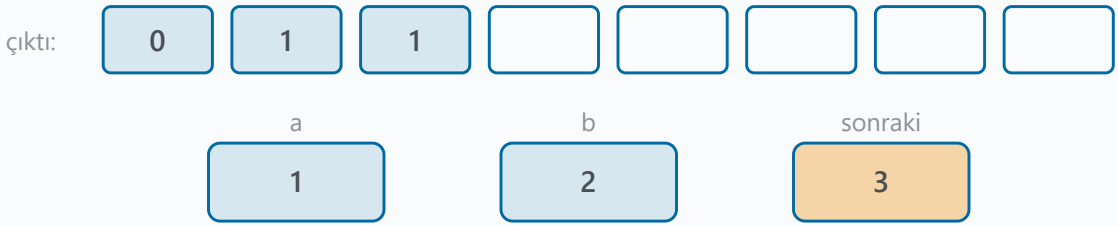
DEĞİŞKENLER

n = 8 | tur = 2 | a = 1 | b = 2 | sonraki = 2

ÇIKTI

```
0 1 1
```

Fibonacci Adım Adım (n = 8, tur 0–2)



sonraki = 1 + 2 = 3.

Adım 12 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
      sonraki = a + b
6     a = b
7     b = sonraki
```

DEĞİŞKENLER

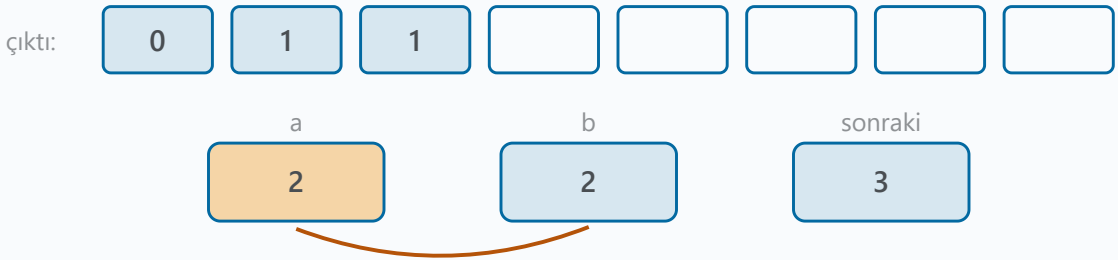
`n = 8` `tur = 2` `a = 1` `b = 2` `sonraki = 3`

ÇIKTI

```
0 1 1
```

ÖRNEKLER · ADIM ADIM İZLEME

Fibonacci Adım Adım (n = 8, tur 0–2)



a = b = 2: pencere bir kayar.

Adım 13 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

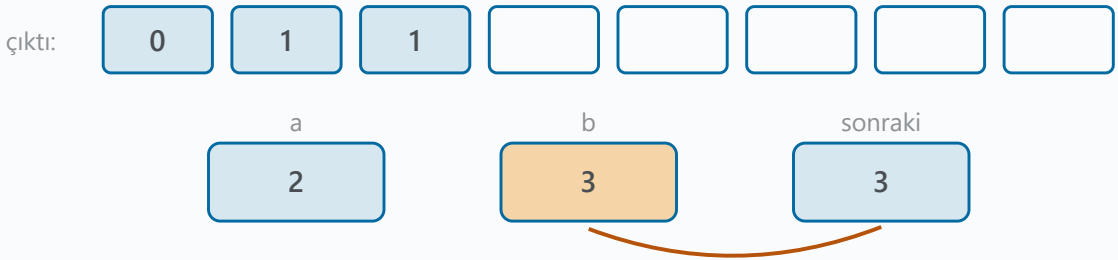
DEĞİŞKENLER

n = 8 | tur = 2 | a = 2 | b = 2 | sonraki = 3

ÇIKTI

```
0 1 1
```

Fibonacci Adım Adım (n = 8, tur 0–2)



b = sonraki = 3.

Adım 14 / 14

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

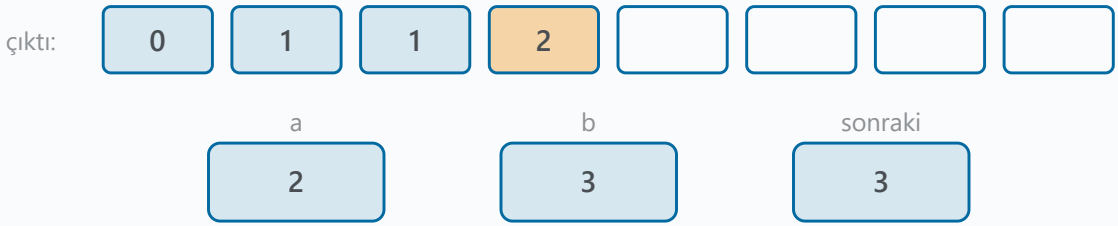
DEĞİŞKENLER

`n = 8` `tur = 2` `a = 2` `b = 3` `sonraki = 3`

ÇIKTI

```
0 1 1
```

Fibonacci Adım Adım (devam: tur 3–5)



Tur 3 (0…7): **a = 2** yazılır.

Adım 1 / 12

```
1 oku n
2 a = 0, b = 1
▶ 0'dan n-1'e kadar tekrarla
▶     yaz a
5     sonraki = a + b
6     a = b
7     b = sonraki
```

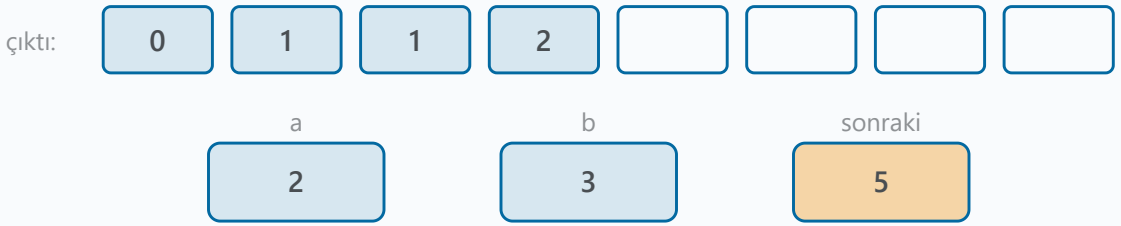
DEĞİŞKENLER

`n = 8` `tur = 3` `a = 2` `b = 3` `sonraki = 3`

ÇIKTI

```
0 1 1 2
```

Fibonacci Adım Adım (devam: tur 3–5)



sonraki = 2 + 3 = 5.

Adım 2 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
▶     sonraki = a + b
6     a = b
7     b = sonraki
```

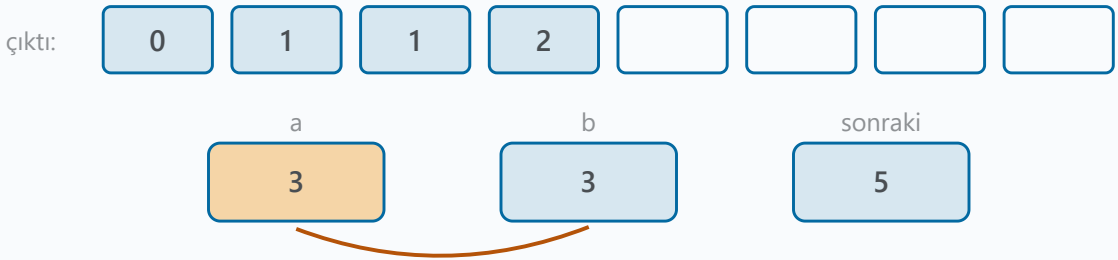
DEĞİŞKENLER

`n = 8` `tur = 3` `a = 2` `b = 3` `sonraki = 5`

ÇIKTI

```
0 1 1 2
```

Fibonacci Adım Adım (devam: tur 3–5)



a = b = 3: pencere bir kayar.

Adım 3 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

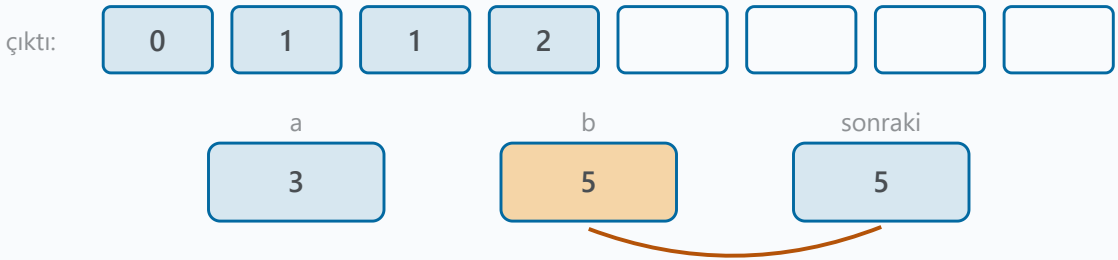
DEĞİŞKENLER

`n = 8` `tur = 3` `a = 3` `b = 3` `sonraki = 5`

ÇIKTI

```
0 1 1 2
```

Fibonacci Adım Adım (devam: tur 3–5)



b = sonraki = 5.

Adım 4 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

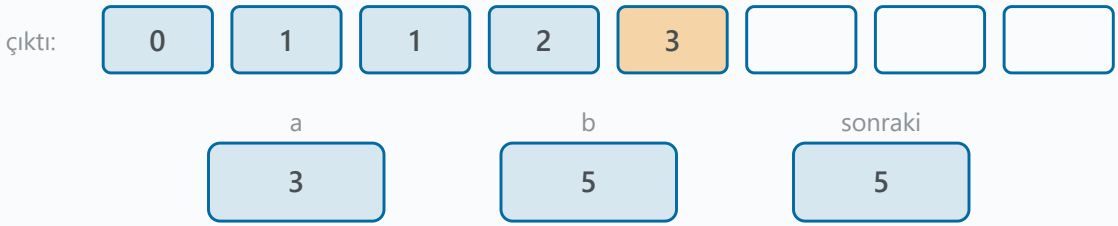
DEĞİŞKENLER

`n = 8` `tur = 3` `a = 3` `b = 5` `sonraki = 5`

ÇIKTI

```
0 1 1 2
```

Fibonacci Adım Adım (devam: tur 3–5)



Tur 4 (0…7): **a = 3** yazılır.

Adım 5 / 12

```
1 oku n
2 a = 0, b = 1
0'dan n-1'e kadar tekrarla
    yaz a
5   sonraki = a + b
6   a = b
7   b = sonraki
```

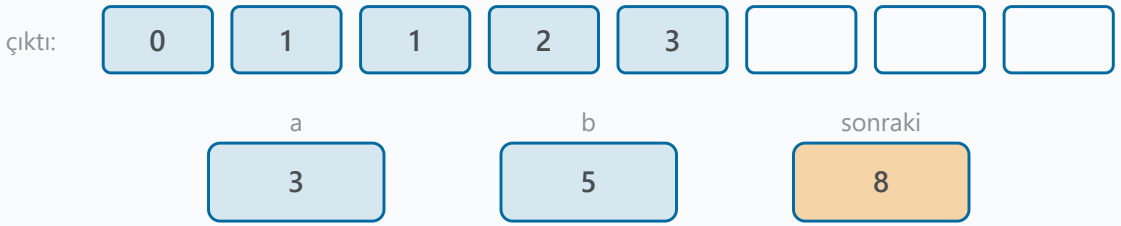
DEĞİŞKENLER

n = 8 | tur = 4 | a = 3 | b = 5 | sonraki = 5

ÇIKTI

```
0 1 1 2 3
```

Fibonacci Adım Adım (devam: tur 3–5)



sonraki = 3 + 5 = 8.

Adım 6 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
▶     sonraki = a + b
6     a = b
7     b = sonraki
```

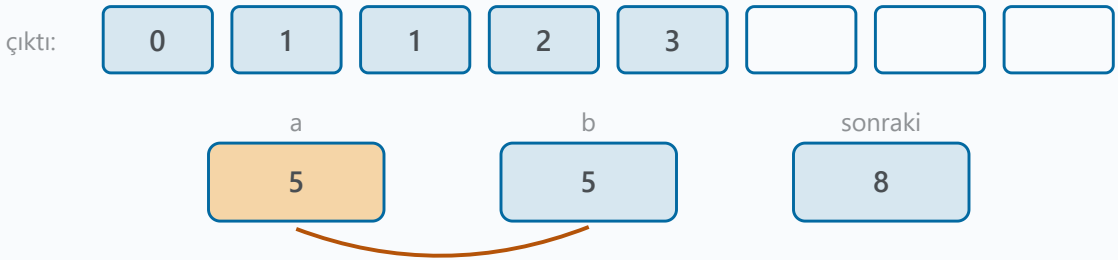
DEĞİŞKENLER

`n = 8` `tur = 4` `a = 3` `b = 5` `sonraki = 8`

ÇIKTI

```
0 1 1 2 3
```

Fibonacci Adım Adım (devam: tur 3–5)



a = b = 5: pencere bir kayar.

Adım 7 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

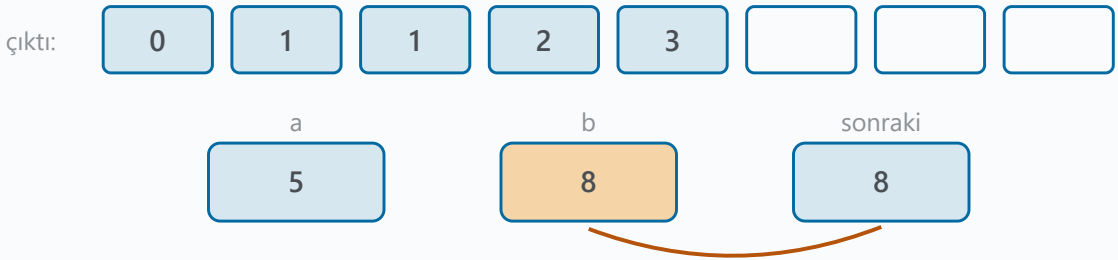
DEĞİŞKENLER

`n = 8` `tur = 4` `a = 5` `b = 5` `sonraki = 8`

ÇIKTI

```
0 1 1 2 3
```

Fibonacci Adım Adım (devam: tur 3–5)



b = sonraki = 8.

Adım 8 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

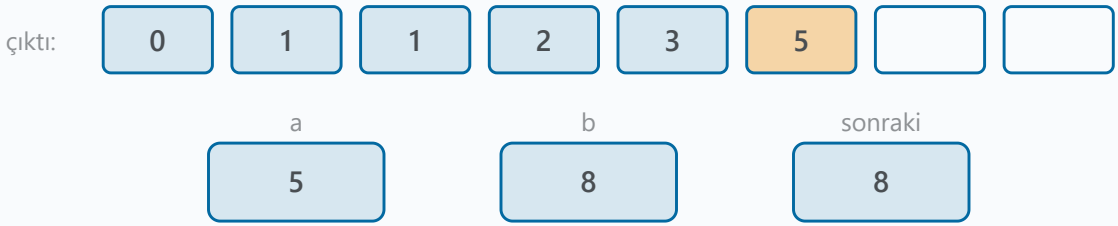
DEĞİŞKENLER

n = 8 | tur = 4 | a = 5 | b = 8 | sonraki = 8

ÇIKTI

```
0 1 1 2 3
```

Fibonacci Adım Adım (devam: tur 3–5)



Tur 5 (0…7): **a = 5** yazılır.

Adım 9 / 12

```
1 oku n
2 a = 0, b = 1
0'dan n-1'e kadar tekrarla
    yaz a
5   sonraki = a + b
6   a = b
7   b = sonraki
```

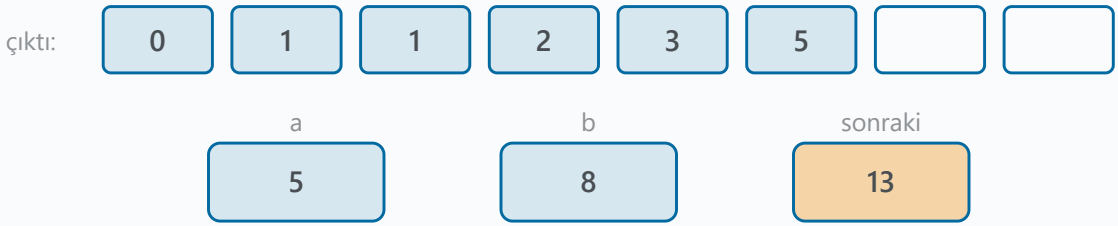
DEĞİŞKENLER

`n = 8` `tur = 5` `a = 5` `b = 8` `sonraki = 8`

ÇIKTI

```
0 1 1 2 3 5
```

Fibonacci Adım Adım (devam: tur 3–5)



sonraki = 5 + 8 = 13.

Adım 10 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
      sonraki = a + b
6     a = b
7     b = sonraki
```

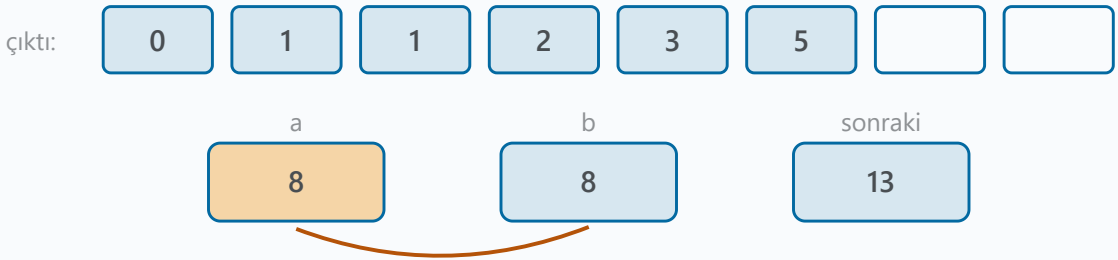
DEĞİŞKENLER

`n = 8` `tur = 5` `a = 5` `b = 8` `sonraki = 13`

ÇIKTI

```
0 1 1 2 3 5
```

Fibonacci Adım Adım (devam: tur 3–5)



a = b = 8: pencere bir kayar.

Adım 11 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

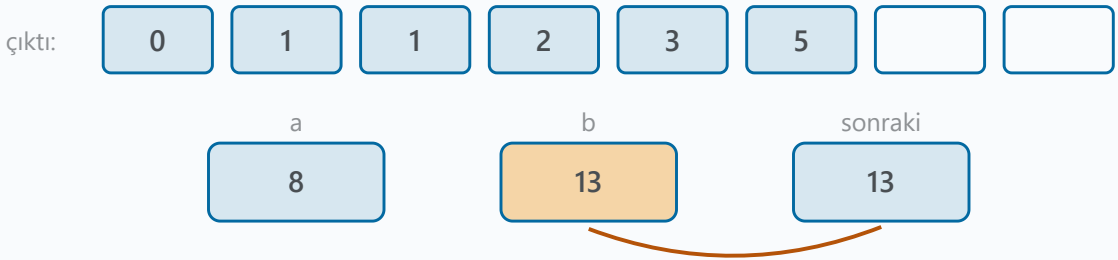
DEĞİŞKENLER

`n = 8` `tur = 5` `a = 8` `b = 8` `sonraki = 13`

ÇIKTI

```
0 1 1 2 3 5
```

Fibonacci Adım Adım (devam: tur 3–5)



b = sonraki = 13.

Adım 12 / 12

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

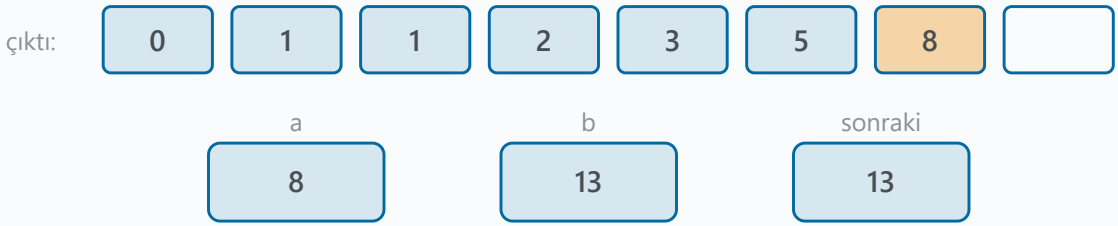
DEĞİŞKENLER

`n = 8` `tur = 5` `a = 8` `b = 13` `sonraki = 13`

ÇIKTI

```
0 1 1 2 3 5
```

Fibonacci Adım Adım (devam: tur 6–7)



Tur 6 (0...7): **a = 8** yazılır.

Adım 1 / 9

```
1 oku n
2 a = 0, b = 1
▶ 0'dan n-1'e kadar tekrarla
▶     yaz a
5     sonraki = a + b
6     a = b
7     b = sonraki
```

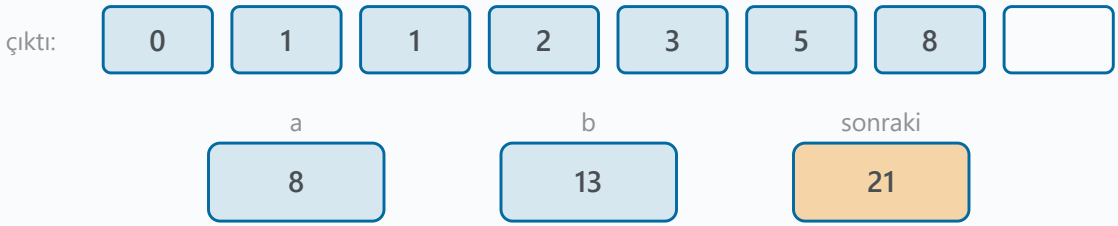
DEĞİŞKENLER

`n = 8` `tur = 6` `a = 8` `b = 13` `sonraki = 13`

ÇIKTI

```
0 1 1 2 3 5 8
```

Fibonacci Adım Adım (devam: tur 6–7)



sonraki = 8 + 13 = 21.

Adım 2 / 9

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
▶     sonraki = a + b
6     a = b
7     b = sonraki
```

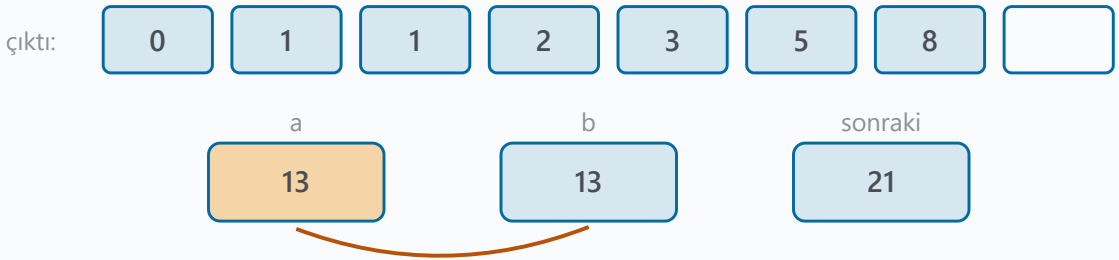
DEĞİŞKENLER

`n = 8` `tur = 6` `a = 8` `b = 13` `sonraki = 21`

ÇIKTI

```
0 1 1 2 3 5 8
```

Fibonacci Adım Adım (devam: tur 6–7)



a = b = 13: pencere bir kayar.

Adım 3 / 9

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

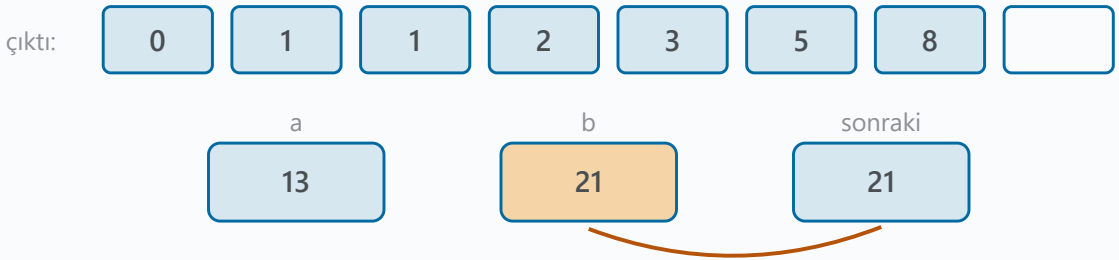
DEĞİŞKENLER

`n = 8` `tur = 6` `a = 13` `b = 13` `sonraki = 21`

ÇIKTI

```
0 1 1 2 3 5 8
```

Fibonacci Adım Adım (devam: tur 6–7)



b = sonraki = 21.

Adım 4 / 9

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

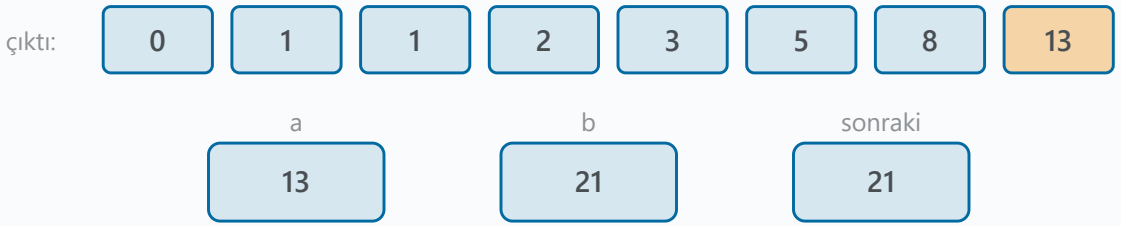
DEĞİŞKENLER

n = 8 | tur = 6 | a = 13 | b = 21 | sonraki = 21

ÇIKTI

```
0 1 1 2 3 5 8
```

Fibonacci Adım Adım (devam: tur 6–7)



Tur 7 (0…7): **a = 13** yazılır.

Adım 5 / 9

```
1 oku n
2 a = 0, b = 1
0'dan n-1'e kadar tekrarla
    yaz a
5   sonraki = a + b
6   a = b
7   b = sonraki
```

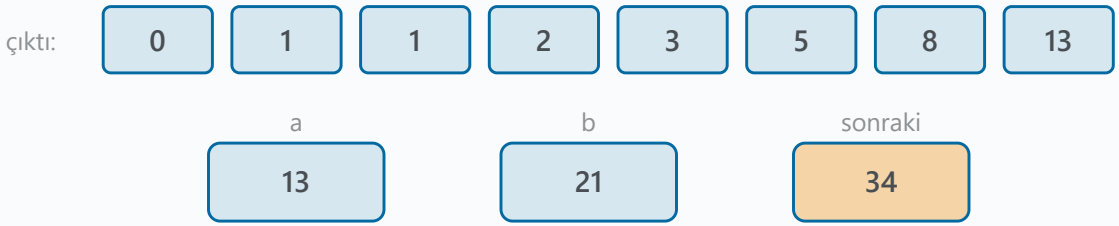
DEĞİŞKENLER

`n = 8` `tur = 7` `a = 13` `b = 21` `sonraki = 21`

ÇIKTI

```
0 1 1 2 3 5 8 13
```

Fibonacci Adım Adım (devam: tur 6–7)



sonraki = 13 + 21 = 34.

Adım 6 / 9

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
      sonraki = a + b
6     a = b
7     b = sonraki
```

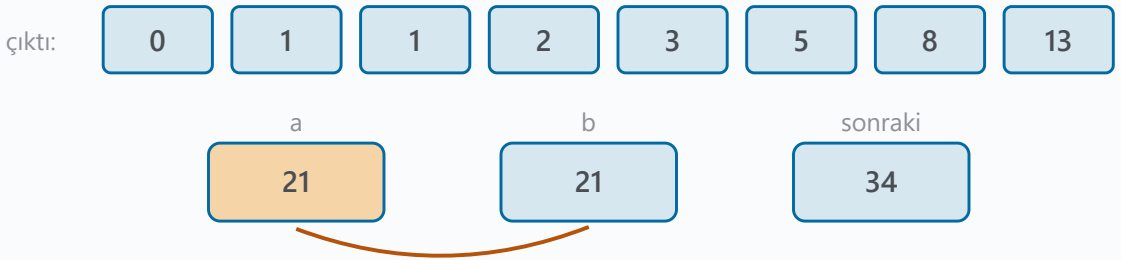
DEĞİŞKENLER

`n = 8` `tur = 7` `a = 13` `b = 21` `sonraki = 34`

ÇIKTI

```
0 1 1 2 3 5 8 13
```

Fibonacci Adım Adım (devam: tur 6–7)



a = b = 21: pencere bir kayar.

Adım 7 / 9

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
▶     a = b
7     b = sonraki
```

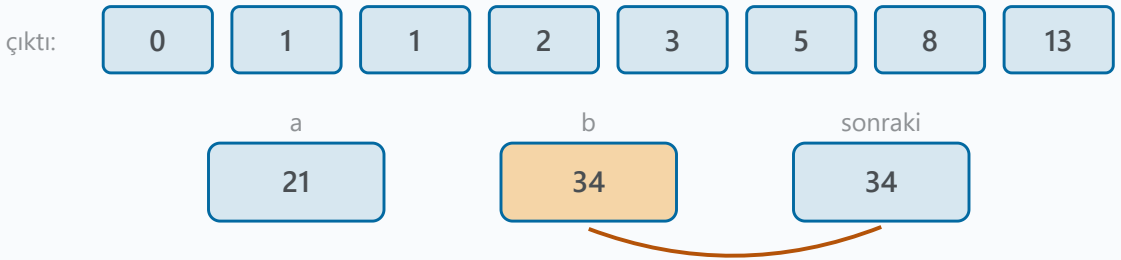
DEĞİŞKENLER

`n = 8` `tur = 7` `a = 21` `b = 21` `sonraki = 34`

ÇIKTI

```
0 1 1 2 3 5 8 13
```

Fibonacci Adım Adım (devam: tur 6–7)



b = sonraki = 34.

Adım 8 / 9

```
1 oku n
2 a = 0, b = 1
3 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
▶     b = sonraki
```

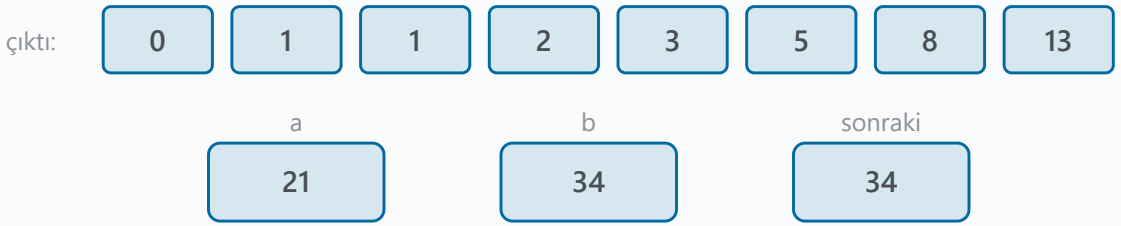
DEĞİŞKENLER

`n = 8` `tur = 7` `a = 21` `b = 34` `sonraki = 34`

ÇIKTI

```
0 1 1 2 3 5 8 13
```

Fibonacci Adım Adım (devam: tur 6–7)



8 tur tamamlandı, döngü biter. Çıktı: **0 1 1 2 3 5 8 13**. Son yazılan terim **13**.

Adım 9 / 9

```
1 oku n
2 a = 0, b = 1
▶ 0'dan n-1'e kadar tekrarla
4     yaz a
5     sonraki = a + b
6     a = b
7     b = sonraki
```

DEĞİŞKENLER

`n = 8` `a = 21` `b = 34` `sonraki = 34`

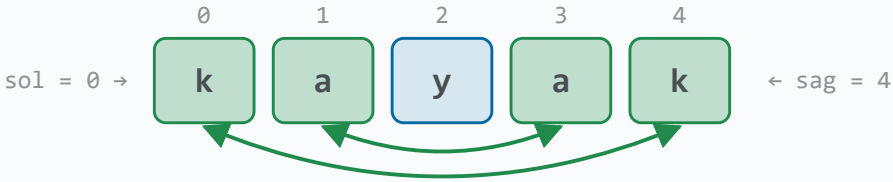
ÇIKTI

```
0 1 1 2 3 5 8 13
```

Örnek: Palindrom

```
metni oku
metni kurala göre temizle
sol = 0, sag = uzunluk - 1
sol < sag iken
    karakterler farklıysa yanlış döndür
    sol = sol + 1, sag = sag - 1
doğru döndür
```

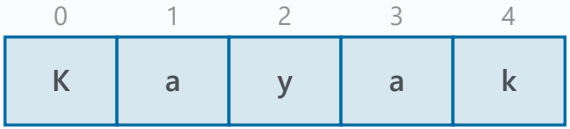
- Karşılaştırma iki uçtan merkeze ilerler.
- Büyük-küçük harf ve boşluk kuralı önceden belirlenmelidir (ör. "Kayak" → "kayak").
- En fazla uzunluk / 2 karşılaştırma yapılır.



Dıştan içe: k = k, a = a; ortadaki y ile karşılaştırma gerekmez → ⌊5 / 2⌋ = 2 karşılaştırma

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



Metin okunur: **"Kayak"**. İlk harf büyük.

Adım 1 / 11

```
▶ metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

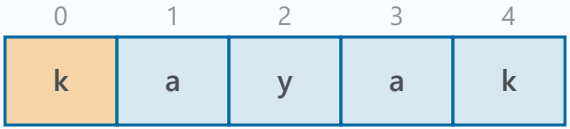
DEĞİŞKENLER

`metin = "Kayak"`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



Kural: küçük harfe çevir, boşlukları at → **"kayak"**. Yoksa K ≠ k sanılırdı.

Adım 2 / 11

```
1 metni oku
▶ metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

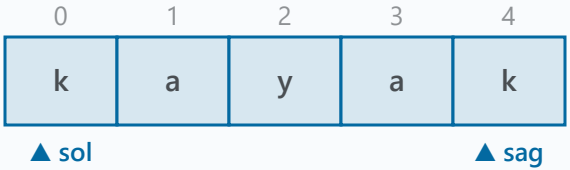
DEĞİŞKENLER

`metin = "kayak"`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



İki uç: **sol = 0**, **sag = 4**.

Adım 3 / 11

```
1 metni oku
2 metni kurala göre temizle
▶ sol = 0, sag = uzunluk - 1
4 sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

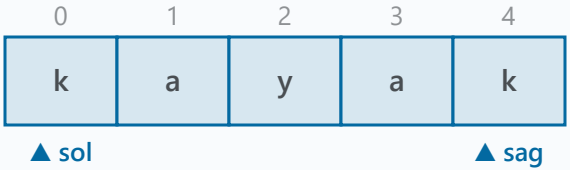
DEĞİŞKENLER

metin = "kayak" sol = 0 sag = 4

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



Koşul: **0 < 4** → **doğru**.

Adım 4 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
▶ sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

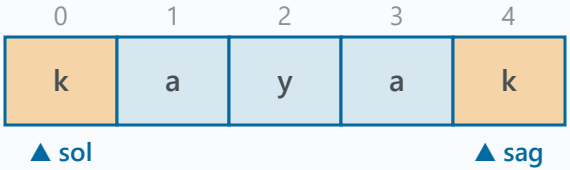
DEĞİŞKENLER

`metin = "kayak"` `sol = 0` `sag = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



metin[0] = 'k', metin[4] = 'k' → **aynı**, devam.

Adım 5 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
▶     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

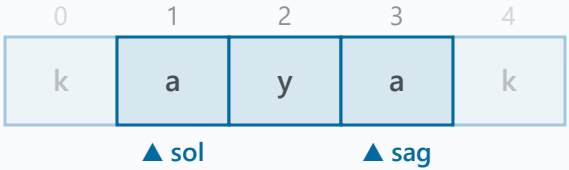
DEĞİŞKENLER

`metin = "kayak"` `sol = 0` `sag = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



İçe doğru bir adım: **sol = 1**, **sag = 3**.

Adım 6 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
5     karakterler farklıysa yanlış döndür
▶     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

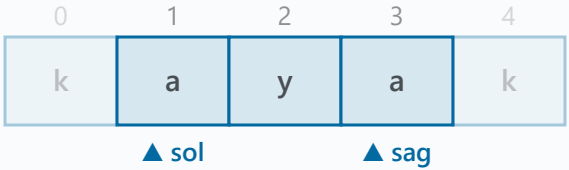
DEĞİŞKENLER

`metin = "kayak"` `sol = 1` `sag = 3`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



Koşul: **1 < 3** → **doğru**.

Adım 7 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
▶ sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

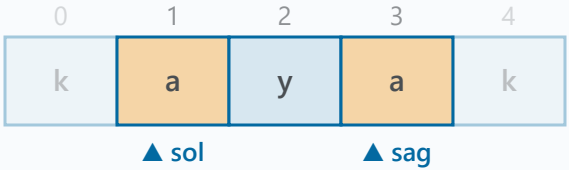
DEĞİŞKENLER

`metin = "kayak"` `sol = 1` `sag = 3`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



metin[1] = 'a', metin[3] = 'a' → **aynı**, devam.

Adım 8 / 11

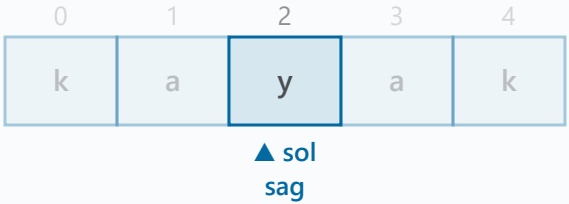
```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
▶     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

DEĞİŞKENLER

`metin = "kayak"` `sol = 1` `sag = 3`

ÇIKTI

Palindrom Adım Adım ("Kayak")



İçe doğru bir adım: **sol = 2**, **sag = 2**.

Adım 9 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
5     karakterler farklıysa yanlış döndür
▶     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

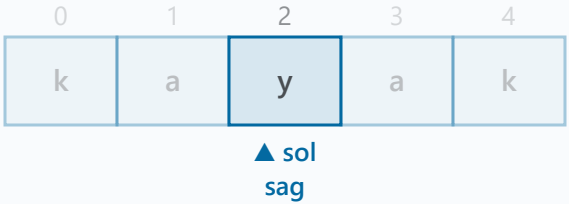
DEĞİŞKENLER

`metin = "kayak"` `sol = 2` `sag = 2`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



Koşul: **2 < 2** → **yanlış**: ortaya gelindi, döngü biter.

Adım 10 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
▶ sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
7 doğru döndür
```

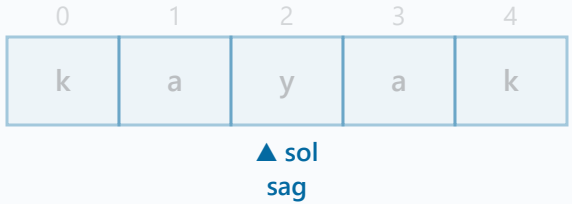
DEĞİŞKENLER

`metin = "kayak"` `sol = 2` `sag = 2`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Palindrom Adım Adım ("Kayak")



Farklı çift bulunmadı → **doğru**: "Kayak" bir palindromdur. 5 harf için yalnızca 2 karşılaştırma yapıldı.

Adım 11 / 11

```
1 metni oku
2 metni kurala göre temizle
3 sol = 0, sag = uzunluk - 1
4 sol < sag iken
5     karakterler farklıysa yanlış döndür
6     sol = sol + 1, sag = sag - 1
▶ doğru döndür
```

DEĞİŞKENLER

`metin = "kayak"` `sol = 2` `sag = 2`

ÇIKTI

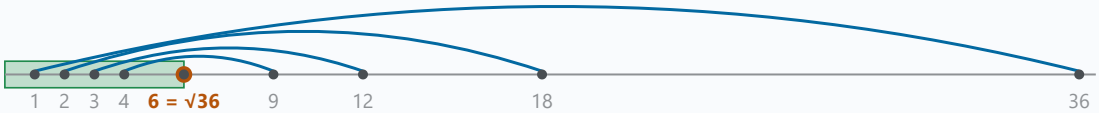
```
doğru
```

ÖRNEKLER

Örnek: Asal Sayı

```
oku n
eğer n < 2 ise "asal değil" yaz ve dur
i = 2
i × i ≤ n iken
    eğer n % i == 0 ise "asal değil" yaz ve dur
    i = i + 1
"asal" yaz
```

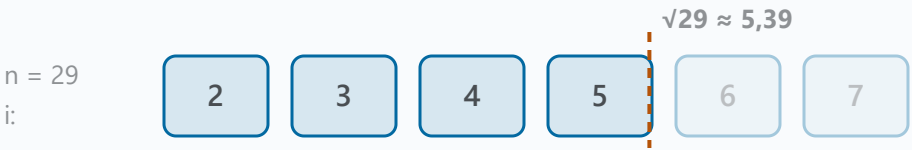
- Bir bölen bulununca arama durur.
- Karekök sınırı: n = a × b ise a ve b'nin **ikisi birden** √n'den büyük olamaz; küçük bölen varsa i × i ≤ n aralığında bulunur.
- % işlemi bölümden kalanı verir.



36 = 1×36 = 2×18 = 3×12 = 4×9 = 6×6: her çiftin küçüğü ≤ √36 → i = 2…6 denemek yeter

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



n = 29 okunur.

Adım 1 / 17

```
▶ oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

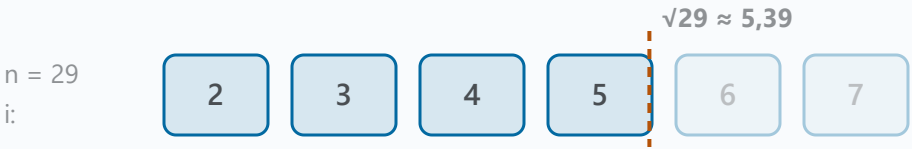
DEĞİŞKENLER

n = 29

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



29 < 2 → **yanlış**; 0 ve 1 değil, devam.

Adım 2 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
6     i = i + 1
7 "asal" yaz
```

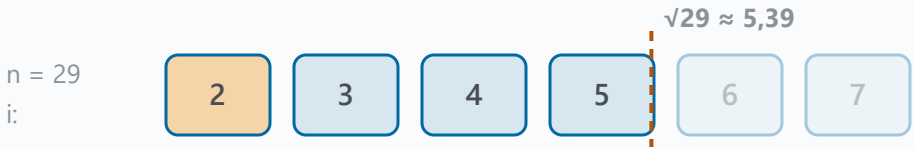
DEĞİŞKENLER

n = 29

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



i = 2: ilk aday bölen.

Adım 3 / 17

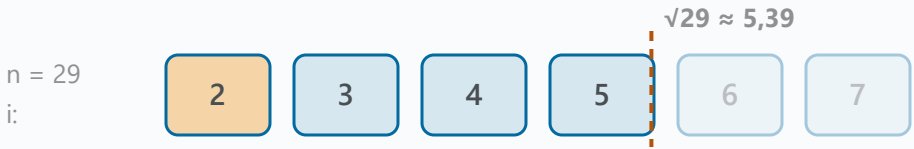
```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
▶ i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

DEĞİŞKENLER

`n = 29` `i = 2`

ÇIKTI

Asal Sayı Adım Adım (n = 29)



Koşul: **2×2 = 4 ≤ 29 → doğru**.

Adım 4 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
▶ i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

DEĞİŞKENLER

`n = 29` `i = 2` `i×i = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



29 % 2 = 1 → sıfır değil, i bölen değil.

Adım 5 / 17

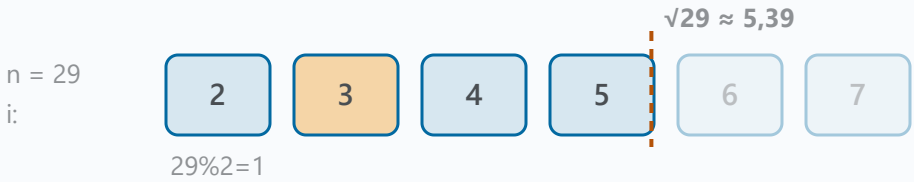
```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
▶     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

DEĞİŞKENLER

`n = 29` `i = 2` `n%i = 1`

ÇIKTI

Asal Sayı Adım Adım (n = 29)



i = 3.

Adım 6 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
▶     i = i + 1
7 "asal" yaz
```

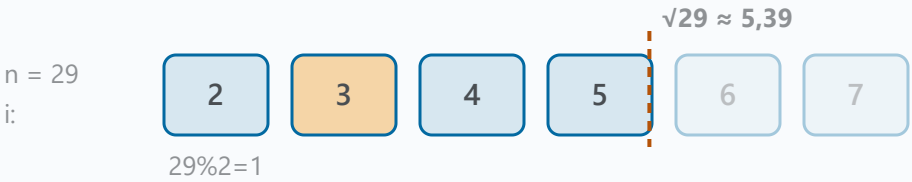
DEĞİŞKENLER

`n = 29` `i = 3`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



Koşul: **3×3 = 9 ≤ 29 → doğru**.

Adım 7 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
▶ i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
6     i = i + 1
7 "asal" yaz
```

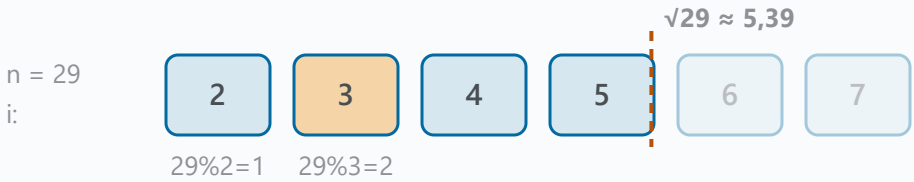
DEĞİŞKENLER

`n = 29` `i = 3` `i×i = 9`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



29 % 3 = 2 → sıfır değil, i bölen değil.

Adım 8 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
▶     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

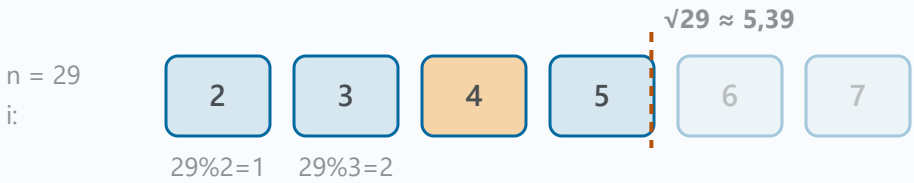
DEĞİŞKENLER

`n = 29` `i = 3` `n%i = 2`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



i = 4.

Adım 9 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
▶     i = i + 1
7 "asal" yaz
```

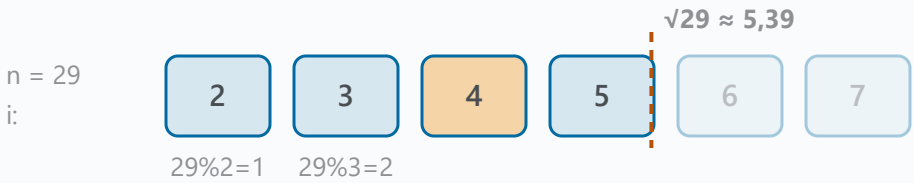
DEĞİŞKENLER

`n = 29` `i = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



Koşul: **4×4 = 16 ≤ 29 → doğru**.

Adım 10 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
▶ i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
6     i = i + 1
7 "asal" yaz
```

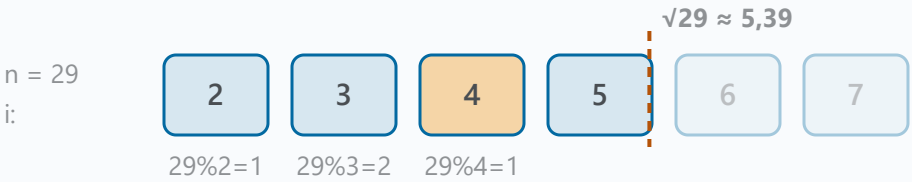
DEĞİŞKENLER

`n = 29` `i = 4` `i×i = 16`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



29 % 4 = 1 → sıfır değil, i bölen değil.

Adım 11 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
▶     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

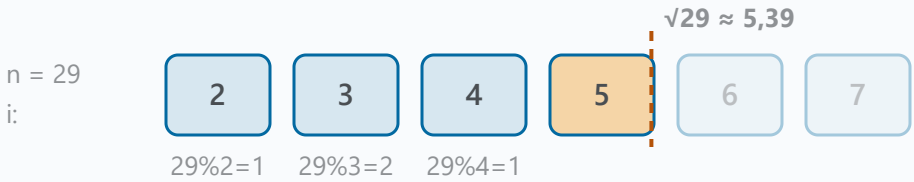
DEĞİŞKENLER

`n = 29` `i = 4` `n%i = 1`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



i = 5.

Adım 12 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
▶     i = i + 1
7 "asal" yaz
```

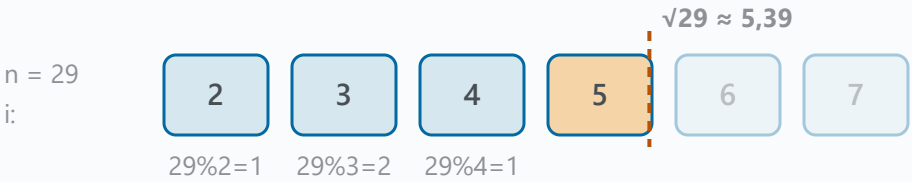
DEĞİŞKENLER

n = 29 i = 5

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



Koşul: **5×5 = 25 ≤ 29 → doğru**.

Adım 13 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
▶ i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
6     i = i + 1
7 "asal" yaz
```

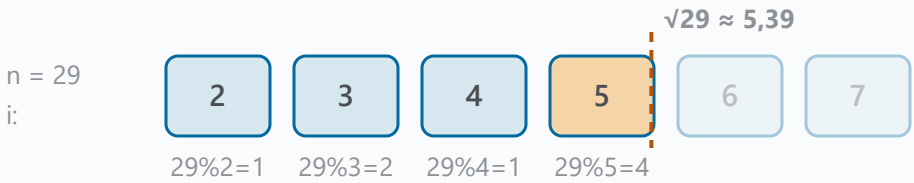
DEĞİŞKENLER

`n = 29` `i = 5` `i×i = 25`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



29 % 5 = 4 → sıfır değil, i bölen değil.

Adım 14 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
▶     eğer n % i == 0 ise "asal değil" yaz ve
  dur
6     i = i + 1
7 "asal" yaz
```

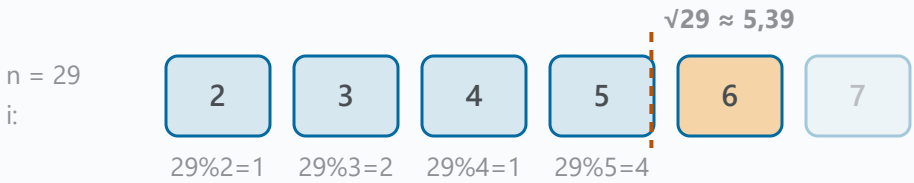
DEĞİŞKENLER

`n = 29` `i = 5` `n%i = 4`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



i = 6.

Adım 15 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
▶     i = i + 1
7 "asal" yaz
```

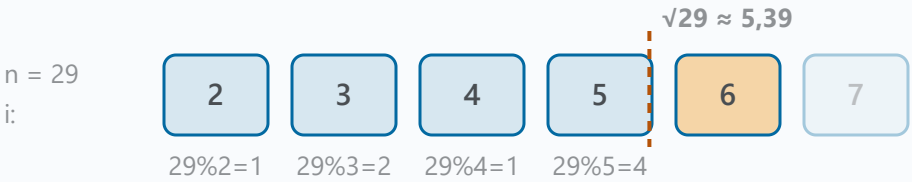
DEĞİŞKENLER

n = 29 i = 6

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



Koşul: **6×6 = 36 ≤ 29** → **yanlış**: i artık √29'u geçti, aramaya gerek yok.

Adım 16 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
▶ i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
6     i = i + 1
7 "asal" yaz
```

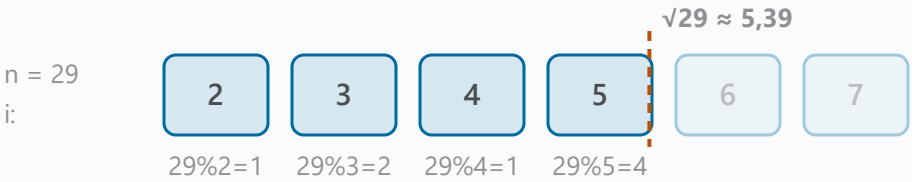
DEĞİŞKENLER

`n = 29` `i = 6` `i×i = 36`

ÇIKTI

ÖRNEKLER · ADIM ADIM İZLEME

Asal Sayı Adım Adım (n = 29)



2, 3, 4, 5 denendi; 6 ve sonrası gereksiz

Bölen bulunmadı → **asal**. Yalnızca 4 bölen denendi (2…28 yerine). n = 21 olsaydı i = 3'te 21 % 3 = 0 → asal değil.

Adım 17 / 17

```
1 oku n
2 eğer n < 2 ise "asal değil" yaz ve dur
3 i = 2
4 i × i ≤ n iken
5     eğer n % i == 0 ise "asal değil" yaz ve dur
6     i = i + 1
▶ "asal" yaz
```

DEĞİŞKENLER

`n = 29` `i = 6`

ÇIKTI

```
asal
```

DOĞRULAMA

Algoritmayı İzleme

- Değişkenlerin her adımdaki değeri bir **izleme tablosuna** yazılır.
- Normal, sınır ve geçersiz girdiler ayrı ayrı denenir.
- Beklenen ve gerçek sonuç ayrıldığında ilk farklı adım bulunur.

Faktöriyel, n = 4 için izleme tablosu (`i`: tur başı değeri, `sonuc`: tur sonu değeri):

An	i	sonuc	i ≤ n
başlangıç	2	1	—
tur 1	2	2	doğru
tur 2	3	6	doğru
tur 3	4	24	doğru
çıkış	5	24	5 ≤ 4 yanlış

Tek bir başarılı test, algoritmanın her girdi için doğru olduğunu kanıtlamaz.

Test Durumları: Faktöriyel Örneği

Tür	Amaç	Girdi	Beklenen
Normal	Yaygın kullanımı doğrulamak	n = 5	120
Sınır	Alt sınırı denemek	n = 0	1 (döngü hiç dönmez)
Geçersiz	Hata davranışını görmek	n = -3	"hata"
Büyük	Taşmayı görmek	n = 21	21! = 51.090.942.171.709.440.000 → 64 bitlik tam sayıya sığmaz



64 bitlik tam sayı (Java `long`) en çok 9.223.372.036.854.775.807 tutar; 20! = 2.432.902.008.176.640.000 sığar. 21! sığmaz. Bu yüzden kısıt 0 ≤ n ≤ 20 olarak yazılır.

ANALİZ

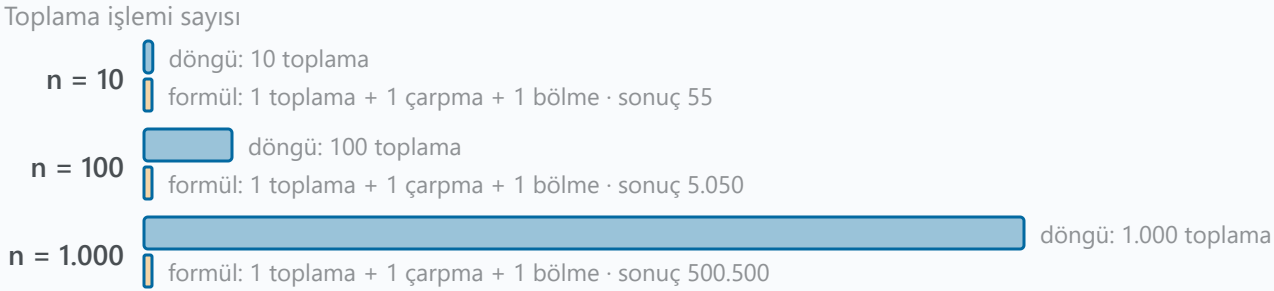
Aynı Problem, İki Algoritma: 1 + 2 + … + n

```
toplam = 0
i = 1
while i <= n
    toplam = toplam + i
    i = i + 1
```

A: döngü, n tur

```
toplam = n * (n + 1) / 2
```

B: Gauss formülü, tek adım



İkisi de **doğru**; ama n büyüdükçe A'nın işi artar, B'ninki sabit kalır: doğruluk kadar **verimlilik** de önemlidir.

ANALİZ

Doğruluk ve Verimlilik

- Doğruluk algoritmanın beklenen sonucu üretmesidir.
- Zaman karmaşıklığı işlem sayısının girdi boyutuyla değişimini inceler; büyük O gösterimiyle yazılır: O(n), O(log n), $O(n^2)$.
- Alan karmaşıklığı kullanılan ek bellek miktarını değerlendirir.
- En hızlı görünen yöntem küçük veride her zaman en anlaşılır yöntem olmayabilir.

n	$\lfloor \log_2 n \rfloor + 1$ (ikili arama)	n (doğrusal arama)	n^2 (basit sıralama)
10	4	10	100
100	7	100	10.000
1.000	10	1.000	1.000.000
1.000.000	20	1.000.000	1.000.000.000.000

ARAMA

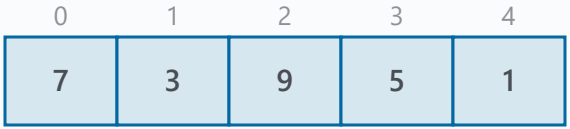
Doğrusal Arama

```java
int ara(int[] dizi, int hedef) {
    for (int i = 0; i < dizi.length; i++) {
        if (dizi[i] == hedef) return i;
    }
    return -1;
}
```

- Öğeler baştan sona karşılaştırılır; verinin sıralı olması gerekmez.
- İlk eşleşmede sonlanır, bulunamazsa -1 döndürür.
- En kötü durumda n öğe incelenir: O(n).

Kod Java ile yazıldı.

Doğrusal Arama Adım Adım (hedef = 5)



`ara(dizi, 5)` çağrıldı. Dizi sıralı değil; baştan tek tek bakılacak.

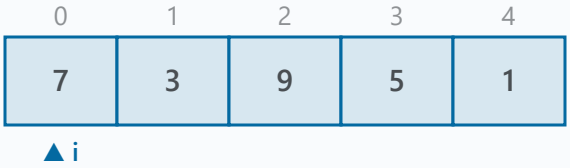
Adım 1 / 9

```
▶ int ara(int[] dizi, int hedef) {
2     for (int i = 0; i < dizi.length; i++) {
3         if (dizi[i] == hedef) return i;
4     }
5     return -1;
6 }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5`

Doğrusal Arama Adım Adım (hedef = 5)



i = 0: 0 < 5 → doğru.

Adım 2 / 9

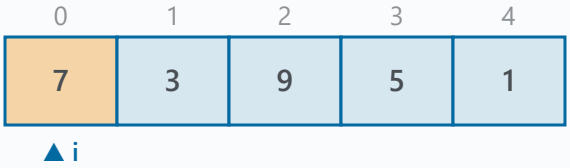
```
1 int ara(int[] dizi, int hedef) {
▶       for (int i = 0; i < dizi.length; i++) {
3           if (dizi[i] == hedef) return i;
4       }
5       return -1;
6   }
```

DEĞİŞKENLER

dizi = [7, 3, 9, 5, 1] hedef = 5

i = 0

Doğrusal Arama Adım Adım (hedef = 5)



dizi[0] = 7 == 5 → **yanlış**.

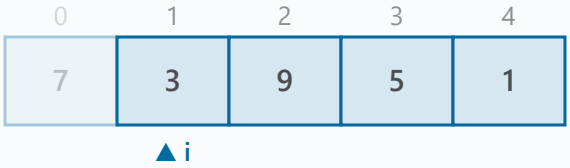
Adım 3 / 9

```
1 int ara(int[] dizi, int hedef) {
2     for (int i = 0; i < dizi.length; i++) {
▶         if (dizi[i] == hedef) return i;
4     }
5     return -1;
6 }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5` `i = 0`

Doğrusal Arama Adım Adım (hedef = 5)



i = 1: 1 < 5 → doğru.

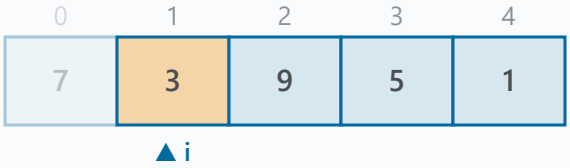
Adım 4 / 9

```
1 int ara(int[] dizi, int hedef) {
▶       for (int i = 0; i < dizi.length; i++) {
3           if (dizi[i] == hedef) return i;
4       }
5       return -1;
6   }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5` `i = 1`

Doğrusal Arama Adım Adım (hedef = 5)



dizi[1] = 3 == 5 → **yanlış**.

Adım 5 / 9

```
1 int ara(int[] dizi, int hedef) {
2     for (int i = 0; i < dizi.length; i++) {
▶         if (dizi[i] == hedef) return i;
4     }
5     return -1;
6 }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5` `i = 1`

Doğrusal Arama Adım Adım (hedef = 5)



i = 2: 2 < 5 → doğru.

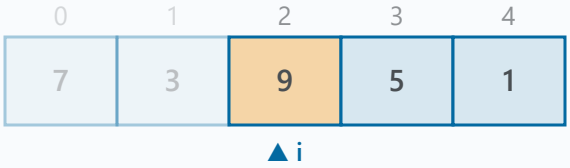
Adım 6 / 9

```
1 int ara(int[] dizi, int hedef) {
▶       for (int i = 0; i < dizi.length; i++) {
3           if (dizi[i] == hedef) return i;
4       }
5       return -1;
6   }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5` `i = 2`

Doğrusal Arama Adım Adım (hedef = 5)



dizi[2] = 9 == 5 → **yanlış**.

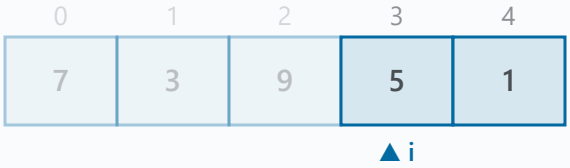
Adım 7 / 9

```
1 int ara(int[] dizi, int hedef) {
2     for (int i = 0; i < dizi.length; i++) {
▶         if (dizi[i] == hedef) return i;
4     }
5     return -1;
6 }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5` `i = 2`

Doğrusal Arama Adım Adım (hedef = 5)



i = 3: 3 < 5 → doğru.

Adım 8 / 9

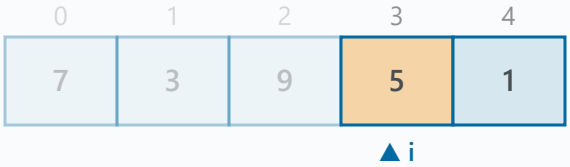
```
1 int ara(int[] dizi, int hedef) {
▶       for (int i = 0; i < dizi.length; i++) {
3           if (dizi[i] == hedef) return i;
4       }
5       return -1;
6   }
```

DEĞİŞKENLER

dizi = [7, 3, 9, 5, 1] hedef = 5

i = 3

Doğrusal Arama Adım Adım (hedef = 5)



dizi[3] = 5 == 5 → **doğru** → **return 3**. 4 karşılaştırma yapıldı; hedef 4 olsaydı 5 karşılaştırmadan sonra -1 dönerdi.

Adım 9 / 9

```
1 int ara(int[] dizi, int hedef) {
2     for (int i = 0; i < dizi.length; i++) {
▶         if (dizi[i] == hedef) return i;
4     }
5     return -1;
6 }
```

DEĞİŞKENLER

`dizi = [7, 3, 9, 5, 1]` `hedef = 5` `i = 3`

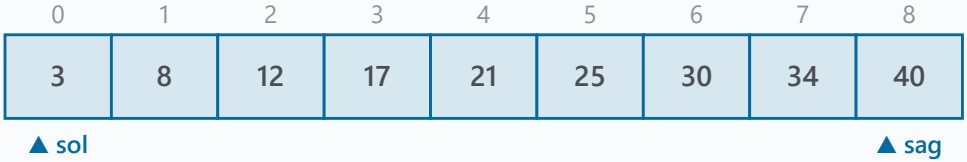
ARAMA

İkili Arama

```java
int ikiliAra(int[] dizi, int hedef) {
    int sol = 0, sag = dizi.length - 1;
    while (sol <= sag) {
        int orta = sol + (sag - sol) / 2;
        if (dizi[orta] == hedef) return orta;
        if (dizi[orta] < hedef) sol = orta + 1;
        else sag = orta - 1;
    }
    return -1;
}
```

- Yalnızca **sıralı** veri üzerinde çalışır; her adımda aralığı yarıya indirir.
- En kötü durumda $\lfloor \log_2 n \rfloor + 1$ tur: O(log n).
- `sol + (sag - sol) / 2`, `(sol + sag) / 2` ile aynı sonucu verir ama çok büyük indislerde taşmaz.
- Tek arama için önce sıralama yapmak toplam maliyeti artırabilir.

İkili Arama Adım Adım (hedef = 34)



Arama aralığı tüm dizi: **sol = 0**, **sag = 8**.

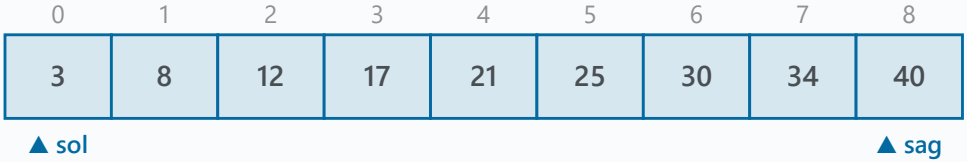
Adım 1 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
▶      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 0` `sag = 8`

İkili Arama Adım Adım (hedef = 34)



0 ≤ 8 → doğru: aralık boş değil, 1. tur.

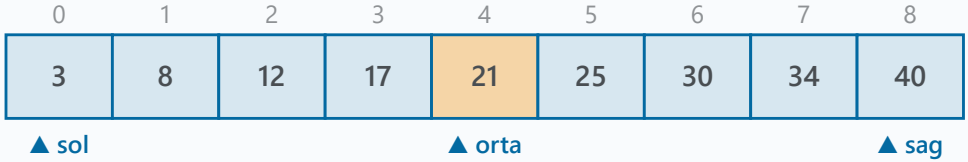
Adım 2 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 0` `sag = 8` `tur = 1`

İkili Arama Adım Adım (hedef = 34)



orta = 0 + (8 − 0) / 2 = 4, dizi[4] = 21.

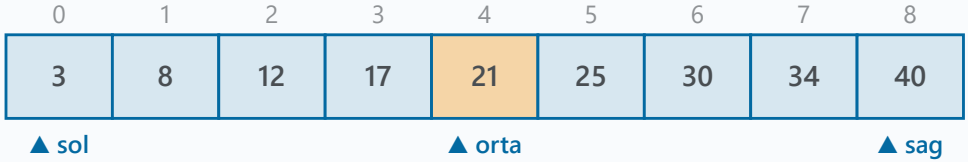
Adım 3 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
▶          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

hedef = 34 | sol = 0 | sag = 8 | orta = 4 | tur = 1

İkili Arama Adım Adım (hedef = 34)



21 == 34 → yanlış.

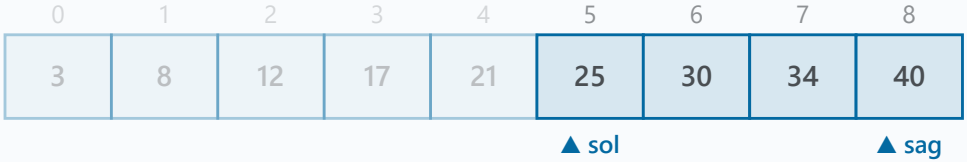
Adım 4 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
▶          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 0` `sag = 8`
`orta = 4` `tur = 1`

İkili Arama Adım Adım (hedef = 34)



21 < 34 → hedef sağ yarıda: **sol = 5**. Sol yarı elendi.

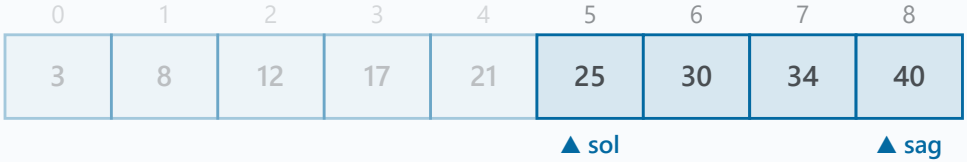
Adım 5 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
▶          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 5` `sag = 8` `orta = 4` `tur = 1`

İkili Arama Adım Adım (hedef = 34)



5 ≤ 8 → doğru: aralık boş değil, 2. tur.

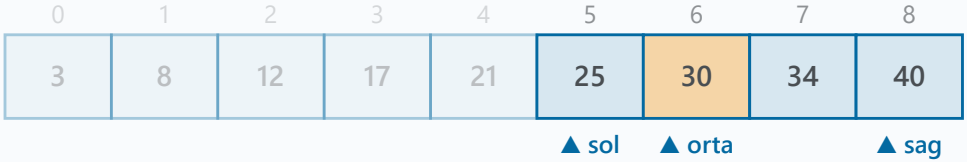
Adım 6 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 5` `sag = 8` `tur = 2`

İkili Arama Adım Adım (hedef = 34)



orta = 5 + (8 − 5) / 2 = 6, dizi[6] = 30.

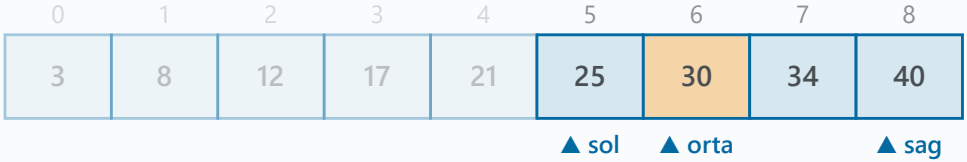
Adım 7 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
▶          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

hedef = 34 | sol = 5 | sag = 8 | orta = 6 | tur = 2

İkili Arama Adım Adım (hedef = 34)



30 == 34 → yanlış.

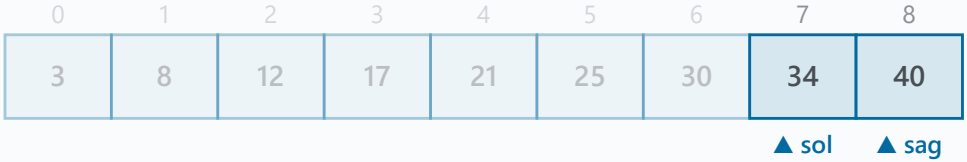
Adım 8 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
▶          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 5` `sag = 8` `orta = 6` `tur = 2`

İkili Arama Adım Adım (hedef = 34)



30 < 34 → hedef sağ yarıda: **sol = 7**. Sol yarı elendi.

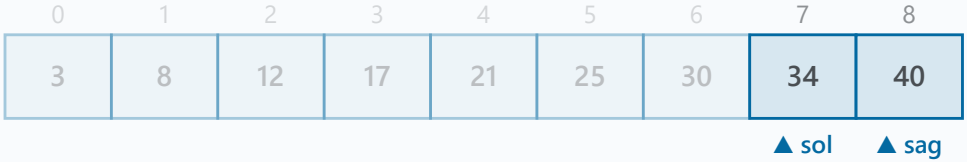
Adım 9 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
▶          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 7` `sag = 8` `orta = 6` `tur = 2`

İkili Arama Adım Adım (hedef = 34)



7 ≤ 8 → doğru: aralık boş değil, 3. tur.

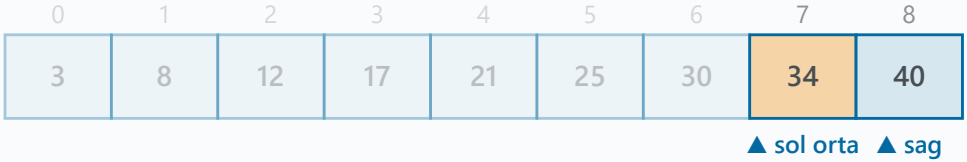
Adım 10 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 7` `sag = 8` `tur = 3`

İkili Arama Adım Adım (hedef = 34)



orta = 7 + (8 − 7) / 2 = 7, dizi[7] = 34.

Adım 11 / 12

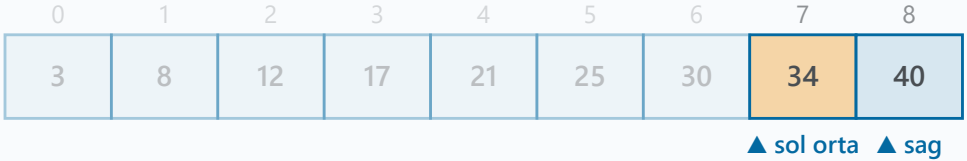
```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
▶          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 7` `sag = 8`

`orta = 7` `tur = 3`

İkili Arama Adım Adım (hedef = 34)



34 == 34 → return 7. 3 turda bulundu; doğrusal arama 8 karşılaştırma yapardı.

Adım 12 / 12

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
▶          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 34` `sol = 7` `sag = 8` `orta = 7` `tur = 3`

ARAMA · KODU ADIM ADIM ÇALIŞTIR

İkili Arama: Hedef Bulunamadı (hedef = 19)



Arama aralığı tüm dizi: **sol = 0**, **sag = 7**. Aranan 19 dizide **yok** (17 ile 21 arasında olurdu).

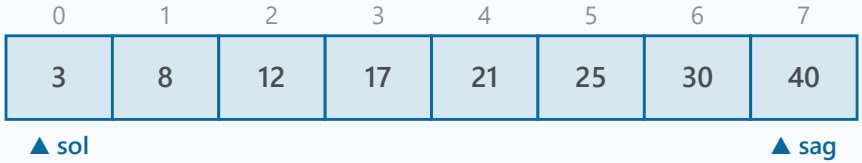
Adım 1 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
▶      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 0` `sag = 7`

İkili Arama: Hedef Bulunamadı (hedef = 19)



0 ≤ 7 → doğru: aralık boş değil, 1. tur.


```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 0` `sag = 7` `tur = 1`

İkili Arama: Hedef Bulunamadı (hedef = 19)



orta = 0 + (7 − 0) / 2 = 3, dizi[3] = 17.


```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
▶          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 0` `sag = 7` `orta = 3` `tur = 1`

İkili Arama: Hedef Bulunamadı (hedef = 19)



17 == 19 → yanlış.

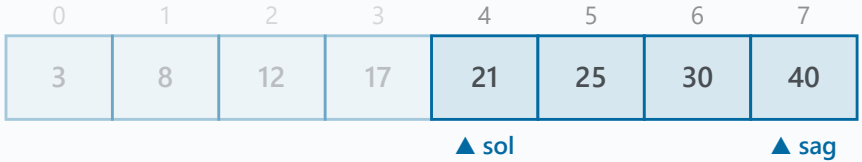
Adım 4 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
▶          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 0` `sag = 7`
`orta = 3` `tur = 1`

İkili Arama: Hedef Bulunamadı (hedef = 19)



17 < 19 → doğru: hedef sağ yarıda olabilir, **sol = 4**. Sol yarı elendi.

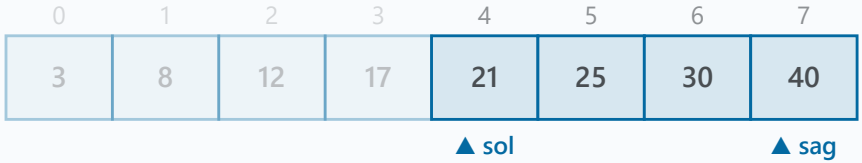
Adım 5 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
▶          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 7` `orta = 3` `tur = 1`

İkili Arama: Hedef Bulunamadı (hedef = 19)



4 ≤ 7 → doğru: aralık boş değil, 2. tur.

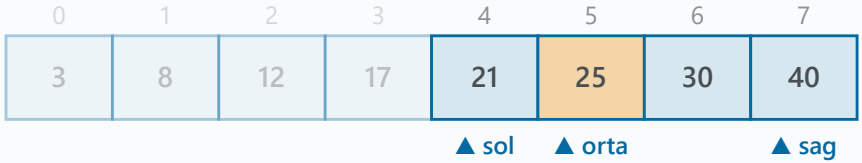
Adım 6 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 7` `tur = 2`

İkili Arama: Hedef Bulunamadı (hedef = 19)



orta = 4 + (7 − 4) / 2 = 5, dizi[5] = 25.

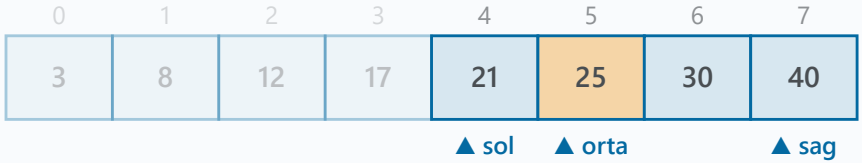
Adım 7 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
▶          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 7` `orta = 5` `tur = 2`

İkili Arama: Hedef Bulunamadı (hedef = 19)



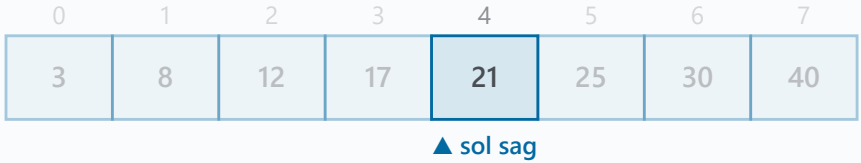
25 == 19 → yanlış.


```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
▶          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 7` `orta = 5` `tur = 2`

İkili Arama: Hedef Bulunamadı (hedef = 19)



25 < 19 → yanlış, **else**: hedef sol yarıda olabilir, **sag = 4**. Sağ yarı elendi.

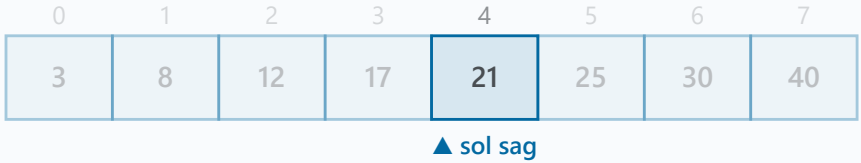
Adım 9 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
▶          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 4` `orta = 5` `tur = 2`

İkili Arama: Hedef Bulunamadı (hedef = 19)



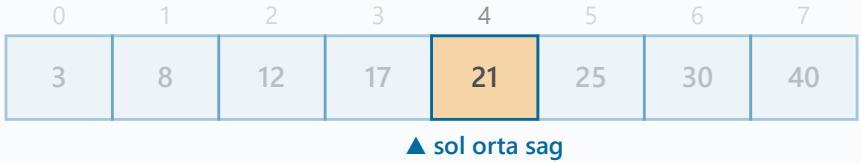
4 ≤ 4 → doğru: aralık boş değil, 3. tur.


```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 4` `tur = 3`

İkili Arama: Hedef Bulunamadı (hedef = 19)



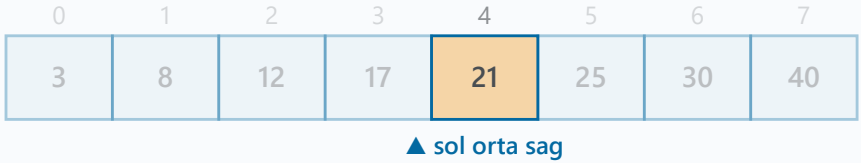
orta = 4 + (4 − 4) / 2 = 4, dizi[4] = 21.


```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
▶          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 4` `orta = 4` `tur = 3`

İkili Arama: Hedef Bulunamadı (hedef = 19)



21 == 19 → yanlış.

Adım 12 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
▶          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 4` `orta = 4` `tur = 3`

İkili Arama: Hedef Bulunamadı (hedef = 19)



21 < 19 → yanlış, **else**: hedef sol yarıda olabilir, **sag = 3**. Sağ yarı elendi.

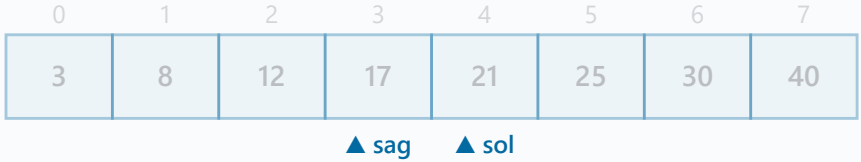
Adım 13 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
▶          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 3` `orta = 4` `tur = 3`

İkili Arama: Hedef Bulunamadı (hedef = 19)



4 ≤ 3 → yanlış: sol, sag'ı geçti; aralık **boş**, döngü biter.

Adım 14 / 15

```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
▶      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
9      return -1;
10 }
```

DEĞİŞKENLER

`hedef = 19` `sol = 4` `sag = 3` `tur = 3`

İkili Arama: Hedef Bulunamadı (hedef = 19)



return -1: hedef bulunamadı. 3 turda aralık boşaldı; 8 öğenin hepsine tek tek bakılmadı.

Adım 15 / 15

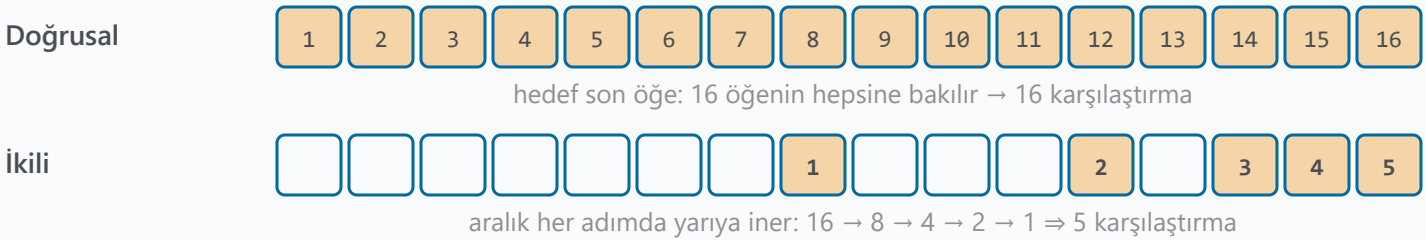
```
1  int ikiliAra(int[] dizi, int hedef) {
2      int sol = 0, sag = dizi.length - 1;
3      while (sol <= sag) {
4          int orta = sol + (sag - sol) / 2;
5          if (dizi[orta] == hedef) return orta;
6          if (dizi[orta] < hedef) sol = orta + 1;
7          else sag = orta - 1;
8      }
▶      return -1;
10 }
```

DEĞİŞKENLER

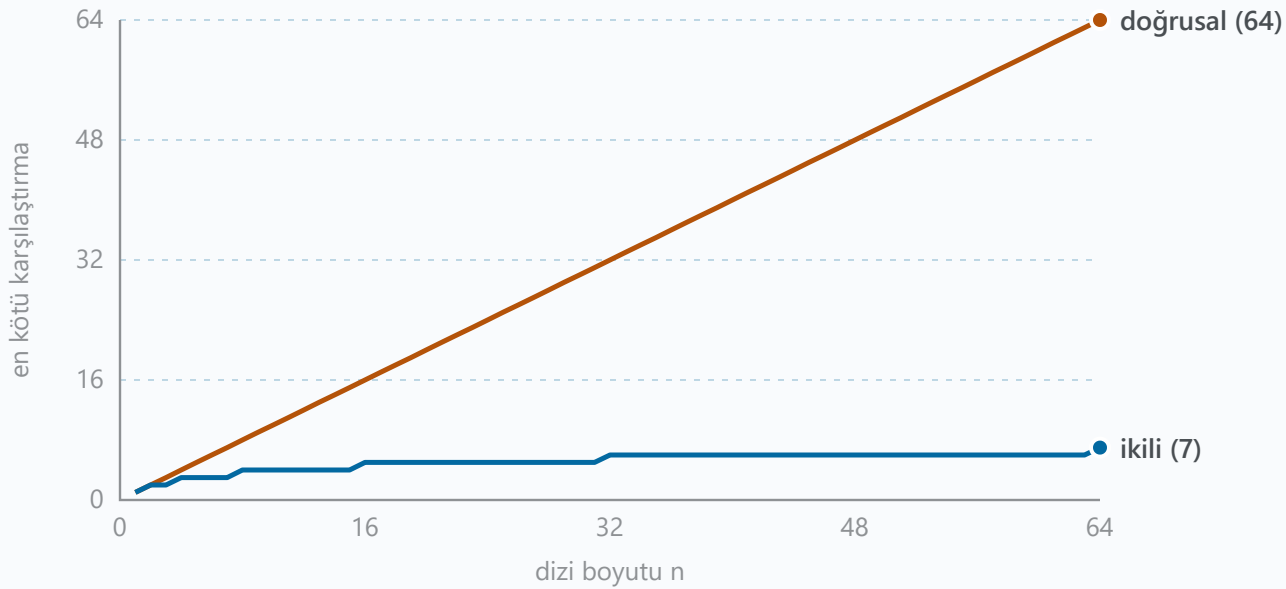
`hedef = 19` `sol = 4` `sag = 3` `tur = 3`

Arama Karşılaştırması

Özellik	Doğrusal	İkili
Sıralı veri	Gerekmez	Gerekir
En kötü zaman	O(n)	O(log n)
Yapı	Dizi veya akış	Rastgele erişimli sıralı yapı
Kullanım	Az veya sırasız veri	Çok sayıda arama



Arama: En Kötü Durumda Kaç Karşılaştırma?



Değerler iki algoritmanın bulunamayan bir hedefle gerçekten çalıştırılmasıyla sayıldı. n = 1.000.000 için doğrusal **1.000.000**, ikili yalnızca **20** tur.

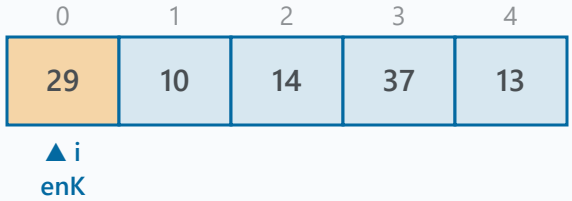
Seçimli Sıralama

- Sıralanmamış bölümde en küçük öğeyi bulur.
- Bulunan öğeyi bölümün ilk konumuyla değiştirir.
- Karşılaştırma sayısı her zaman $n(n-1)/2 \rightarrow O(n^2)$; takas en fazla $n - 1$; ek bellek $O(1)$.

```
i = 0'dan n-2'ye kadar
    enK = i
    j = i+1'den n-1'e kadar
        eğer dizi[j] < dizi[enK] ise enK = j
    dizi[i] ile dizi[enK] yer değiştir
```



Seçimli Sıralama Adım Adım



i = 0: sıralanmamış bölüm 0...4. Aday en küçük: dizi[0] = 29.

Adım 1 / 18

```
▶ i = 0'dan n-2'ye kadar
▶     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [29, 10, 14, 37, 13]` `i = 0` `enK = 0`

Seçimli Sıralama Adım Adım



dizi[1] = 10 < dizi[0] = 29 → **doğru** → **enK = 1**.

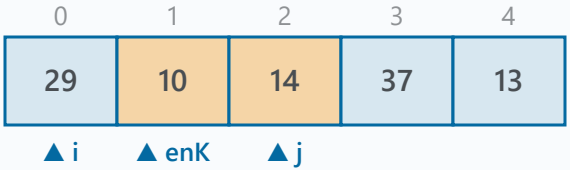
Adım 2 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [29, 10, 14, 37, 13]` `i = 0` `j = 1` `enK = 1` `kars = 1`

Seçimli Sıralama Adım Adım



dizi[2] = 14 < dizi[1] = 10 → **yanlış**.

Adım 3 / 18

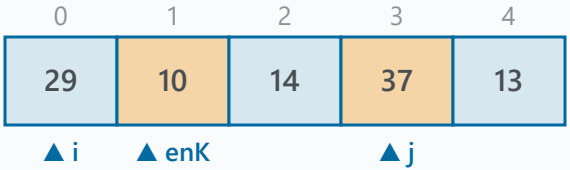
```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [29, 10, 14, 37, 13]` `i = 0`

`j = 2` `enK = 1` `kars = 2`

Seçimli Sıralama Adım Adım



dizi[3] = 37 < dizi[1] = 10 → **yanlış**.

Adım 4 / 18

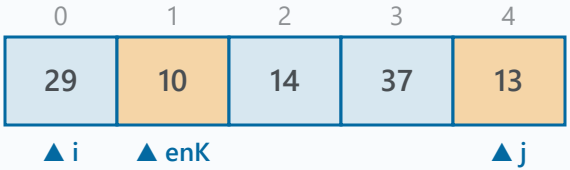
```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [29, 10, 14, 37, 13]` `i = 0`

`j = 3` `enK = 1` `kars = 3`

Seçimli Sıralama Adım Adım



dizi[4] = 13 < dizi[1] = 10 → **yanlış**.

Adım 5 / 18

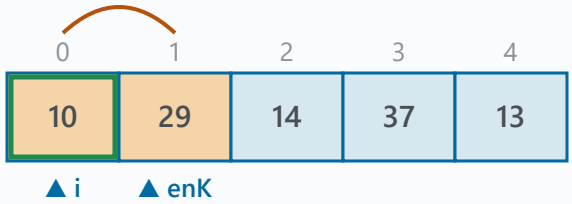
```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [29, 10, 14, 37, 13]` `i = 0`

`j = 4` `enK = 1` `kars = 4`

Seçimli Sıralama Adım Adım



dizi[0] ile dizi[1] yer değiştirir → 10 yerine oturdu.

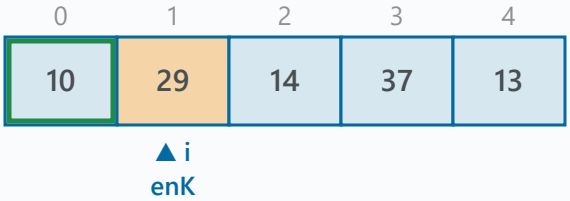
Adım 6 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
▶     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 29, 14, 37, 13]` `i = 0` `enK = 1` `kars = 4`

Seçimli Sıralama Adım Adım



i = 1: sıralanmamış bölüm 1…4. Aday en küçük: dizi[1] = 29.

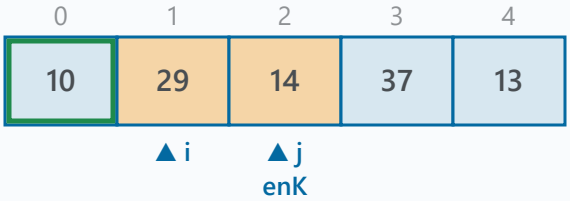
Adım 7 / 18

```
▶ i = 0'dan n-2'ye kadar
▶     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 29, 14, 37, 13]` `i = 1` `enK = 1`

Seçimli Sıralama Adım Adım



dizi[2] = 14 < dizi[1] = 29 → **doğru** → **enK = 2**.

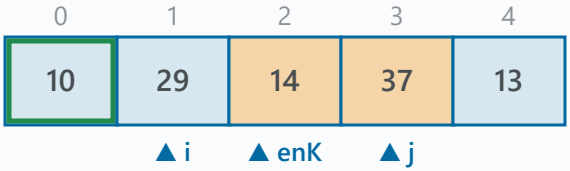
Adım 8 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 29, 14, 37, 13]` `i = 1` `j = 2` `enK = 2` `kars = 5`

Seçimli Sıralama Adım Adım



dizi[3] = 37 < dizi[2] = 14 → **yanlış**.

Adım 9 / 18

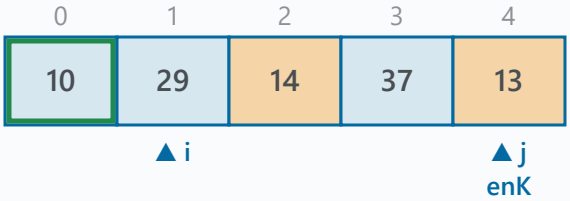
```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 29, 14, 37, 13]` `i = 1`

`j = 3` `enK = 2` `kars = 6`

Seçimli Sıralama Adım Adım



dizi[4] = 13 < dizi[2] = 14 → **doğru** → **enK = 4**.

Adım 10 / 18

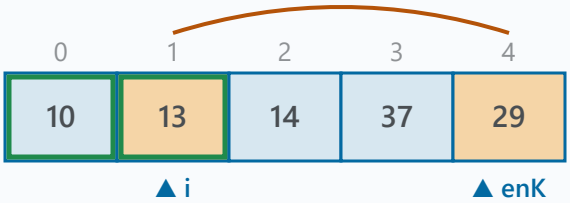
```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 29, 14, 37, 13]` `i = 1`

`j = 4` `enK = 4` `kars = 7`

Seçimli Sıralama Adım Adım



dizi[1] ile dizi[4] yer değiştirir → 13 yerine oturdu.

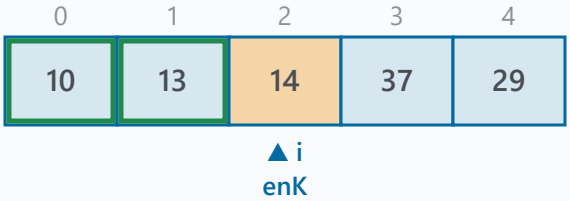
Adım 11 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
▶     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 1` `enK = 4` `kars = 7`

Seçimli Sıralama Adım Adım



i = 2: sıralanmamış bölüm 2...4. Aday en küçük: dizi[2] = 14.

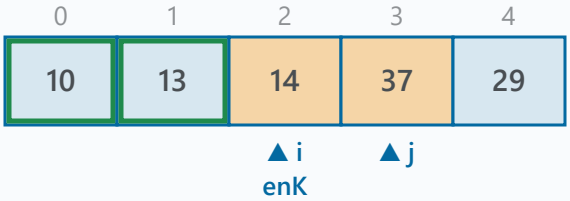
Adım 12 / 18

```
▶ i = 0'dan n-2'ye kadar
▶     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 2` `enK = 2`

Seçimli Sıralama Adım Adım



dizi[3] = 37 < dizi[2] = 14 → **yanlış**.

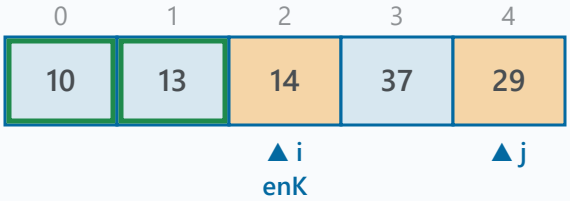
Adım 13 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 2` `j = 3` `enK = 2` `kars = 8`

Seçimli Sıralama Adım Adım



dizi[4] = 29 < dizi[2] = 14 → **yanlış**.

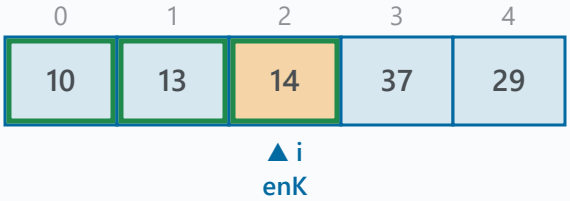
Adım 14 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 2` `j = 4` `enK = 2` `kars = 9`

Seçimli Sıralama Adım Adım



En küçük zaten yerinde (enK = i), değişim etkisiz.

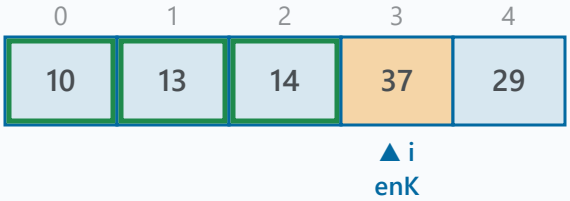
Adım 15 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
▶     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 2` `enK = 2` `kars = 9`

Seçimli Sıralama Adım Adım



i = 3: sıralanmamış bölüm 3...4. Aday en küçük: dizi[3] = 37.

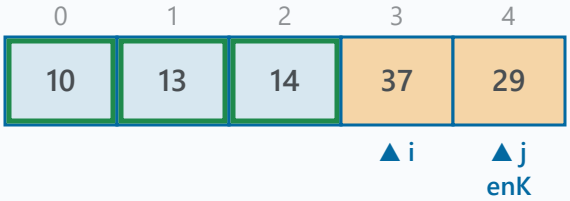
Adım 16 / 18

```
▶ i = 0'dan n-2'ye kadar
▶     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 3` `enK = 3`

Seçimli Sıralama Adım Adım



dizi[4] = 29 < dizi[3] = 37 → **doğru** → **enK = 4**.

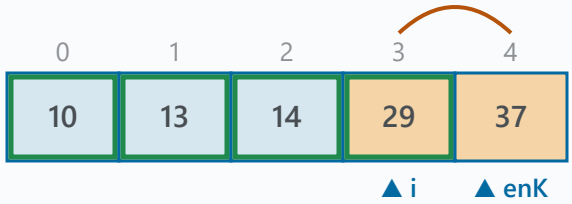
Adım 17 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
▶     j = i+1'den n-1'e kadar
▶         eğer dizi[j] < dizi[enK] ise enK = j
5     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 37, 29]` `i = 3` `j = 4` `enK = 4` `kars = 10`

Seçimli Sıralama Adım Adım



dizi[3] ile dizi[4] yer değiştirir → 29 yerine oturdu. Dizi sıralandı: 10 karşılaştırma = n(n−1)/2.

Adım 18 / 18

```
1 i = 0'dan n-2'ye kadar
2     enK = i
3     j = i+1'den n-1'e kadar
4         eğer dizi[j] < dizi[enK] ise enK = j
▶     dizi[i] ile dizi[enK] yer değiştir
```

DEĞİŞKENLER

`dizi = [10, 13, 14, 29, 37]` `i = 3` `enK = 4` `kars = 10`

SIRALAMA

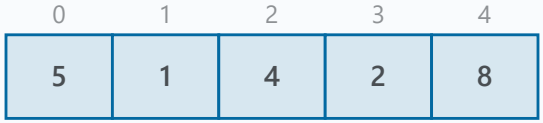
Kabarcık Sıralaması

- Komşu öğeleri karşılaştırır ve yanlış sıradaysa yer değiştirir.
- Her turda kalanların en büyüğü sona taşınır; sonraki tur bir öğe kısa olur.
- Bir turda hiç değişim olmazsa dizi sıralıdır, durulur.
- Genel durumda $O(n^2)$; en iyi durumda (zaten sıralı dizi) tek tur: $O(n)$.

```
tur = 0
tekrarla
    degisti = yanlış
    j = 0'dan n-2-tur'a kadar
        eğer dizi[j] > dizi[j+1] ise
            dizi[j] ile dizi[j+1] yer değiştir
            degisti = doğru
    tur = tur + 1
degisti yanlış olana kadar
```

SIRALAMA · ADIM ADIM İZLEME

Kabarcık Sıralaması Adım Adım (1. geçiş)



tur = 0. Komşu çiftler soldan sağa karşılaştırılacak.

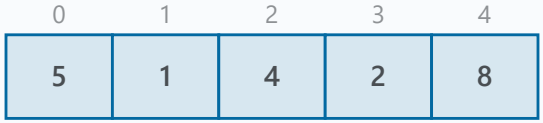
Adım 1 / 11

```
▶ tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [5, 1, 4, 2, 8]` `tur = 0`

Kabarcık Sıralaması Adım Adım (1. geçiş)



1. geçiş: degisti = yanlış; j = 0…3.

Adım 2 / 11

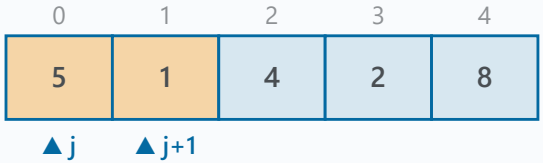
```
1 tur = 0
2 tekrarla
▶     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

dizi = [5, 1, 4, 2, 8] tur = 0

degisti = yanlış

Kabarcık Sıralaması Adım Adım (1. geçiş)



dizi[0] = 5 > dizi[1] = 1 → **doğru** → yer değiştirecek.

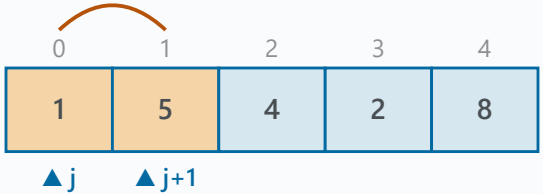
Adım 3 / 11

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [5, 1, 4, 2, 8]` `tur = 0` `j = 0` `degisti = yanlış`

Kabarcık Sıralaması Adım Adım (1. geçiş)



Yer değiştirdi: 5 bir sağa kaydı; **degisti = doğru**.

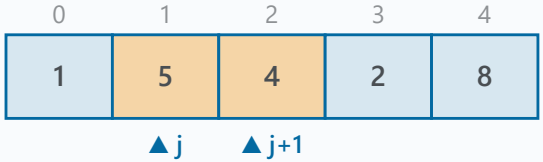
Adım 4 / 11

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
▶             dizi[j] ile dizi[j+1] yer değiştir
▶             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 5, 4, 2, 8]` `tur = 0` `j = 0` `degisti = doğru`

Kabarcık Sıralaması Adım Adım (1. geçiş)



dizi[1] = 5 > dizi[2] = 4 → **doğru** → yer değiştirecek.

Adım 5 / 11

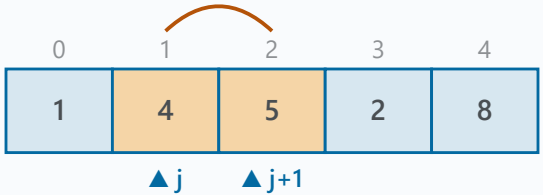
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 5, 4, 2, 8]` `tur = 0` `j = 1`

`degisti = doğru`

Kabarcık Sıralaması Adım Adım (1. geçiş)



Yer değiştirdi: 5 bir sağa kaydı; **degisti = doğru**.

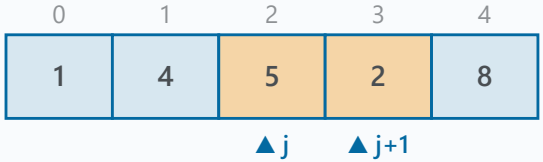
Adım 6 / 11

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
▶             dizi[j] ile dizi[j+1] yer değiştir
▶             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

dizi = [1, 4, 5, 2, 8] tur = 0 j = 1 degisti = doğru

Kabarcık Sıralaması Adım Adım (1. geçiş)



dizi[2] = 5 > dizi[3] = 2 → **doğru** → yer değiştirecek.

Adım 7 / 11

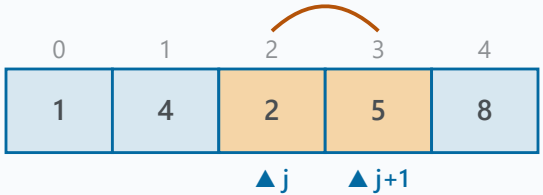
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 5, 2, 8]` `tur = 0` `j = 2`

`degisti = doğru`

Kabarcık Sıralaması Adım Adım (1. geçiş)



Yer değiştirdi: 5 bir sağa kaydı; **degisti = doğru**.

Adım 8 / 11

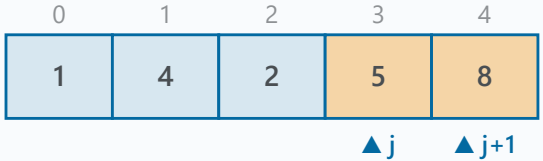
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
▶             dizi[j] ile dizi[j+1] yer değiştir
▶             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 2, 5, 8]` `tur = 0` `j = 2`

`degisti = doğru`

Kabarcık Sıralaması Adım Adım (1. geçiş)



dizi[3] = 5 > dizi[4] = 8 → **yanlış**, sıra doğru.

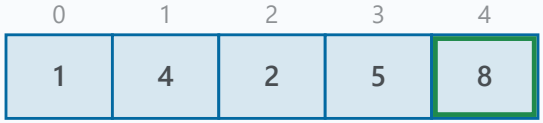
Adım 9 / 11

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 2, 5, 8]` `tur = 0` `j = 3` `degisti = doğru`

Kabarcık Sıralaması Adım Adım (1. geçiş)



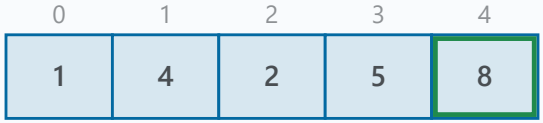
Geçiş bitti: **8** kesin yerinde (yeşil çerçeve). **tur = 1**.


```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
▶     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 2, 5, 8]` `tur = 1` `degisti = doğru`

Kabarcık Sıralaması Adım Adım (1. geçiş)



degisti = doğru → yeni geçiş gerekli.

Adım 11 / 11

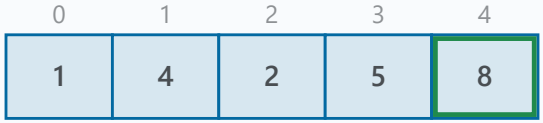
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
▶ degisti yanlış olana kadar
```

DEĞİŞKENLER

dizi = [1, 4, 2, 5, 8] tur = 1

degisti = doğru

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



2. geçiş: degisti = yanlış; j = 0…2.

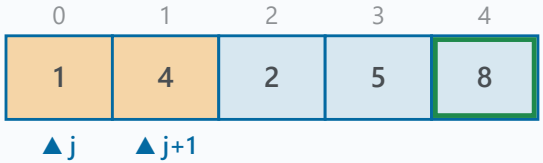
Adım 1 / 7

```
1 tur = 0
2 tekrarla
▶     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 2, 5, 8]` `tur = 1` `degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



dizi[0] = 1 > dizi[1] = 4 → **yanlış**, sıra doğru.

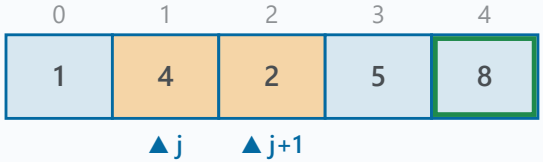
Adım 2 / 7

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 2, 5, 8]` `tur = 1` `j = 0` `degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



dizi[1] = 4 > dizi[2] = 2 → **doğru** → yer değiştirecek.

Adım 3 / 7

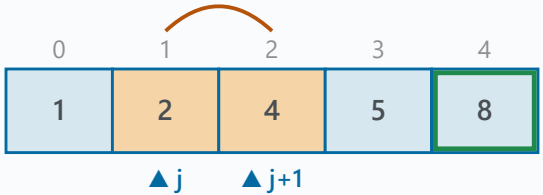
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 4, 2, 5, 8]` `tur = 1` `j = 1`

`degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



Yer değiştirdi: 4 bir sağa kaydı; **degisti = doğru**.

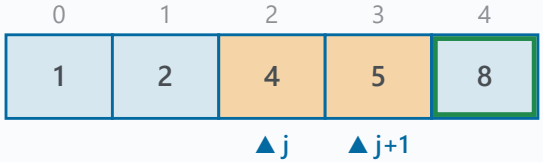
Adım 4 / 7

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
▶             dizi[j] ile dizi[j+1] yer değiştir
▶             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 1` `j = 1` `degisti = doğru`

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



dizi[2] = 4 > dizi[3] = 5 → **yanlış**, sıra doğru.

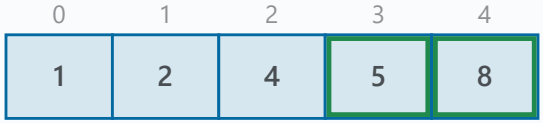
Adım 5 / 7

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 1` `j = 2` `degisti = doğru`

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



Geçiş bitti: **5** kesin yerinde (yeşil çerçeve). **tur = 2**.

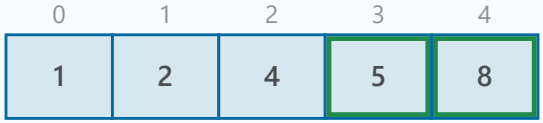
Adım 6 / 7

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
▶     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 2` `degisti = doğru`

Kabarcık Sıralaması Adım Adım (devam: 2. geçiş)



degisti = doğru → yeni geçiş gerekli.

Adım 7 / 7

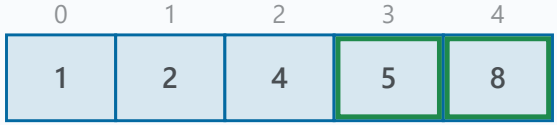
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
▶ degisti yanlış olana kadar
```

DEĞİŞKENLER

dizi = **[1, 2, 4, 5, 8]** tur = **2**

degisti = **doğru**

Kabarcık Sıralaması Adım Adım (devam: 3. geçiş)



3. geçiş: degisti = yanlış; j = 0…1.

Adım 1 / 5

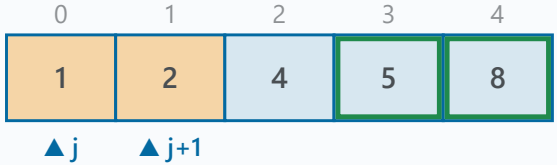
```
1 tur = 0
2 tekrarla
▶     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 2`

`degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 3. geçiş)



dizi[0] = 1 > dizi[1] = 2 → **yanlış**, sıra doğru.

Adım 2 / 5

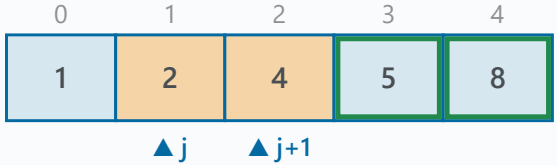
```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 2`

`j = 0` `degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 3. geçiş)



dizi[1] = 2 > dizi[2] = 4 → **yanlış**, sıra doğru.

Adım 3 / 5

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
▶         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 2`

`j = 1` `degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 3. geçiş)



Geçiş bitti: **4** kesin yerinde (yeşil çerçeve). **tur = 3**.

Adım 4 / 5

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
▶     tur = tur + 1
9 degisti yanlış olana kadar
```

DEĞİŞKENLER

`dizi = [1, 2, 4, 5, 8]` `tur = 3`

`degisti = yanlış`

Kabarcık Sıralaması Adım Adım (devam: 3. geçiş)



degisti = yanlış → hiç değişim olmadı, dizi sıralı; **dur**. Toplam 9 karşılaştırma.

Adım 5 / 5

```
1 tur = 0
2 tekrarla
3     degisti = yanlış
4     j = 0'dan n-2-tur'a kadar
5         eğer dizi[j] > dizi[j+1] ise
6             dizi[j] ile dizi[j+1] yer değiştir
7             degisti = doğru
8     tur = tur + 1
▶ degisti yanlış olana kadar
```

DEĞİŞKENLER

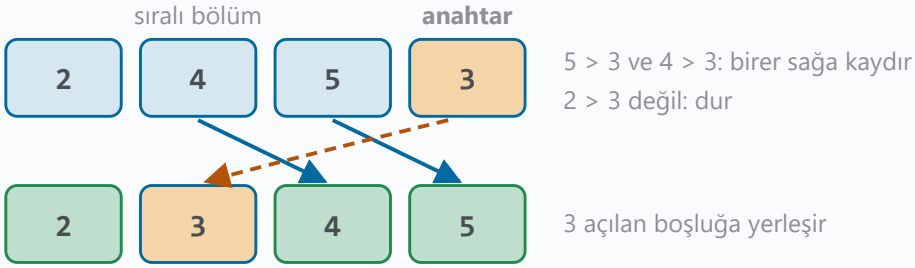
`dizi = [1, 2, 4, 5, 8]` `tur = 3`

`degisti = yanlış`

Eklemeli Sıralama (Insertion Sort)

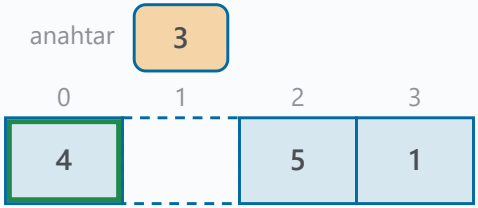
- Elde iskambil kâğıdı dizmeye benzer: soldaki bölüm hep sıralıdır.
- Sıradaki öğe (**anahtar**) alınır; ondan büyükler birer sağa kaydırılır, anahtar açılan boşluğa konur.
- En kötü durumda (ters sıralı) $O(n^2)$; zaten sıralı dizide her anahtar için tek karşılaştırma: O(n). Ek bellek O(1).
- Az sayıda ve neredeyse sıralı veride çok hızlıdır.

```
i = 1'den n-1'e kadar
    anahtar = dizi[i]
    j = i - 1
    j >= 0 ve dizi[j] > anahtar iken
        dizi[j+1] = dizi[j]
        j = j - 1
    dizi[j+1] = anahtar
```



SIRALAMA · ADIM ADIM İZLEME

Eklemeli Sıralama Adım Adım



i = 1: anahtar = dizi[1] = **3** alınır, yeri boşalır. Sol bölüm (0...0) sıralı. j = 0.

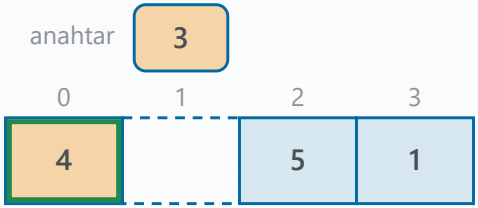
Adım 1 / 17

```
▶ i = 1'den n-1'e kadar
▶     anahtar = dizi[i]
▶     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

dizi = [4, 3, 5, 1] | i = 1 | anahtar = 3 | j = 0

Eklemeli Sıralama Adım Adım



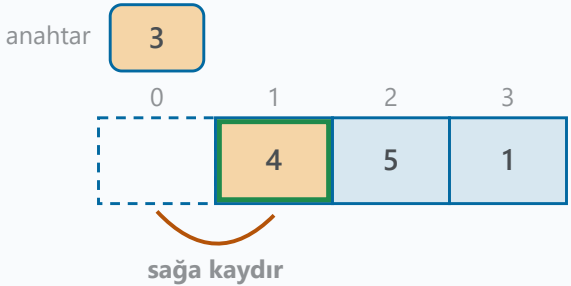
j = 0 ≥ 0 ve dizi[0] = 4 > 3 → **doğru**.


```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [4, _, 5, 1]` `i = 1` `anahtar = 3` `j = 0`

Eklemeli Sıralama Adım Adım



4 bir sağa kaydı; boşluk 0. hücreye geçti. **j = -1**.

Adım 3 / 17

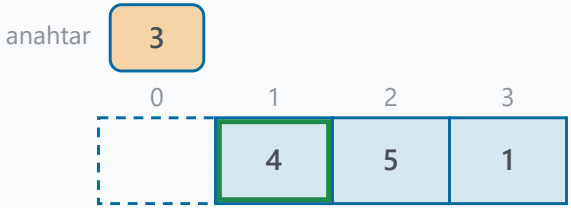
```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
▶         dizi[j+1] = dizi[j]
▶         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [_, 4, 5, 1]` `i = 1`

`anahtar = 3` `j = -1`

Eklemeli Sıralama Adım Adım



j = -1 < 0 → **yanlış** (sol uca gelindi): döngü biter.

Adım 4 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [_, 4, 5, 1]` `i = 1`

`anahtar = 3` `j = -1`

Eklemeli Sıralama Adım Adım



dizi[0] = 3: anahtar boşluğa yerleşti. Artık 0...1 sıralı.

Adım 5 / 17

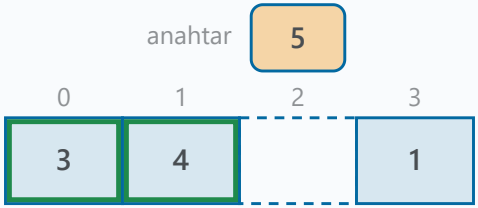
```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
▶     dizi[j+1] = anahtar
```

DEĞİŞKENLER

dizi = [3, 4, 5, 1] i = 1

anahtar = 3 j = -1

Eklemeli Sıralama Adım Adım



i = 2: anahtar = dizi[2] = **5** alınır, yeri boşalır. Sol bölüm (0...1) sıralı. j = 1.

Adım 6 / 17

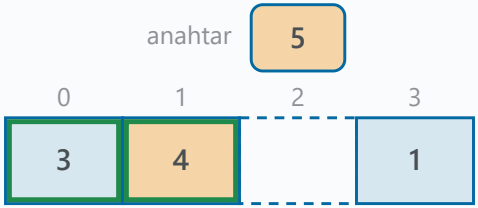
```
▶ i = 1'den n-1'e kadar
▶     anahtar = dizi[i]
▶     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, 4, 5, 1]` `i = 2`

`anahtar = 5` `j = 1`

Eklemeli Sıralama Adım Adım



j = 1 ≥ 0 ve dizi[1] = 4 > 5 → **yanlış**: döngü biter.

Adım 7 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, 4, _, 1]` `i = 2` `anahtar = 5` `j = 1`

Eklemeli Sıralama Adım Adım



dizi[2] = 5: anahtar boşluğa yerleşti. Artık 0...2 sıralı.

Adım 8 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
▶     dizi[j+1] = anahtar
```

DEĞİŞKENLER

dizi = [3, 4, 5, 1] i = 2

anahtar = 5 j = 1

Eklemeli Sıralama Adım Adım



i = 3: anahtar = dizi[3] = **1** alınır, yeri boşalır. Sol bölüm (0…2) sıralı. j = 2.

Adım 9 / 17

```
▶ i = 1'den n-1'e kadar
▶     anahtar = dizi[i]
▶     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, 4, 5, 1]` `i = 3`

`anahtar = 1` `j = 2`

Eklemeli Sıralama Adım Adım



j = 2 ≥ 0 ve dizi[2] = 5 > 1 → **doğru**.

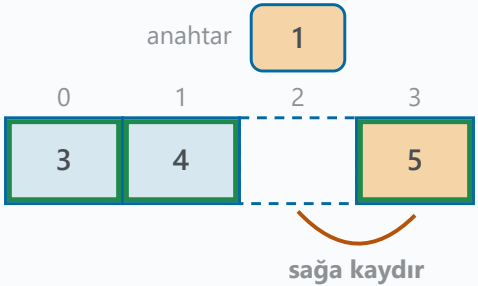
Adım 10 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, 4, 5, _]` `i = 3` `anahtar = 1` `j = 2`

Eklemeli Sıralama Adım Adım



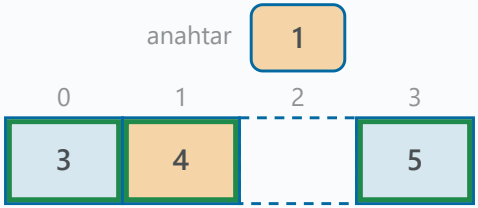
5 bir sağa kaydı; boşluk 2. hücreye geçti. **j = 1**.


```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
▶         dizi[j+1] = dizi[j]
▶         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, 4, _, 5]` `i = 3` `anahtar = 1` `j = 1`

Eklemeli Sıralama Adım Adım



j = 1 ≥ 0 ve dizi[1] = 4 > 1 → **doğru**.

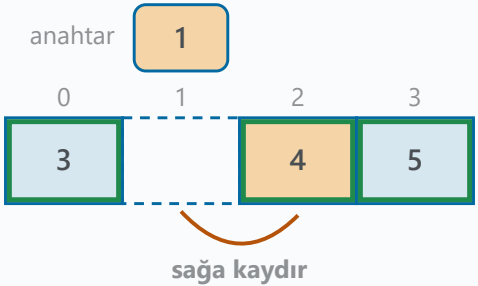
Adım 12 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, 4, _, 5]` `i = 3` `anahtar = 1` `j = 1`

Eklemeli Sıralama Adım Adım



4 bir sağa kaydı; boşluk 1. hücreye geçti. **j = 0**.

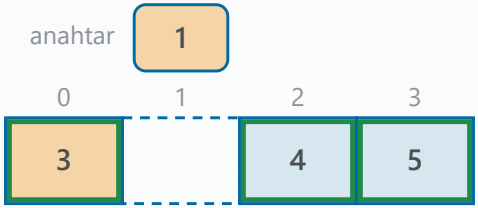

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
▶         dizi[j+1] = dizi[j]
▶         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, _, 4, 5]` `i = 3`

`anahtar = 1` `j = 0`

Eklemeli Sıralama Adım Adım



j = 0 ≥ 0 ve dizi[0] = 3 > 1 → **doğru**.

Adım 14 / 17

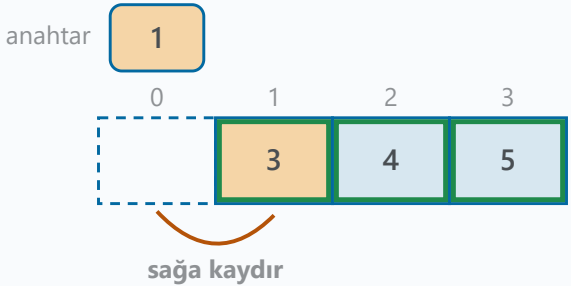
```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [3, _, 4, 5]` `i = 3`

`anahtar = 1` `j = 0`

Eklemeli Sıralama Adım Adım



3 bir sağa kaydı; boşluk 0. hücreye geçti. **j = -1**.

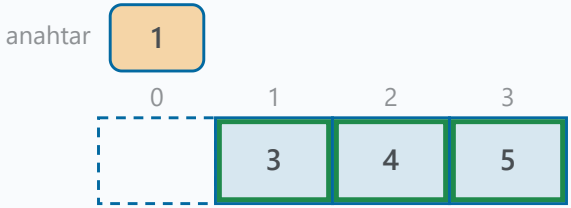

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
▶         dizi[j+1] = dizi[j]
▶         j = j - 1
7     dizi[j+1] = anahtar
```

DEĞİŞKENLER

`dizi = [_, 3, 4, 5]` `i = 3`

`anahtar = 1` `j = -1`

Eklemeli Sıralama Adım Adım



j = -1 < 0 → **yanlış** (sol uca gelindi): döngü biter.

Adım 16 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
▶     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
7     dizi[j+1] = anahtar
```

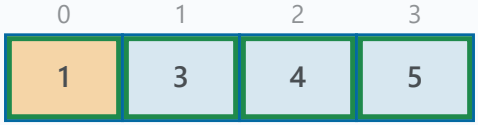
DEĞİŞKENLER

`dizi = [_, 3, 4, 5]` `i = 3`

`anahtar = 1` `j = -1`

SIRALAMA · ADIM ADIM İZLEME

Eklemeli Sıralama Adım Adım



dizi[0] = 1: anahtar boşluğa yerleşti. Artık 0…3 sıralı. Dizi sıralandı; 5 karşılaştırma yapıldı.

Adım 17 / 17

```
1 i = 1'den n-1'e kadar
2     anahtar = dizi[i]
3     j = i - 1
4     j >= 0 ve dizi[j] > anahtar iken
5         dizi[j+1] = dizi[j]
6         j = j - 1
▶     dizi[j+1] = anahtar
```

DEĞİŞKENLER

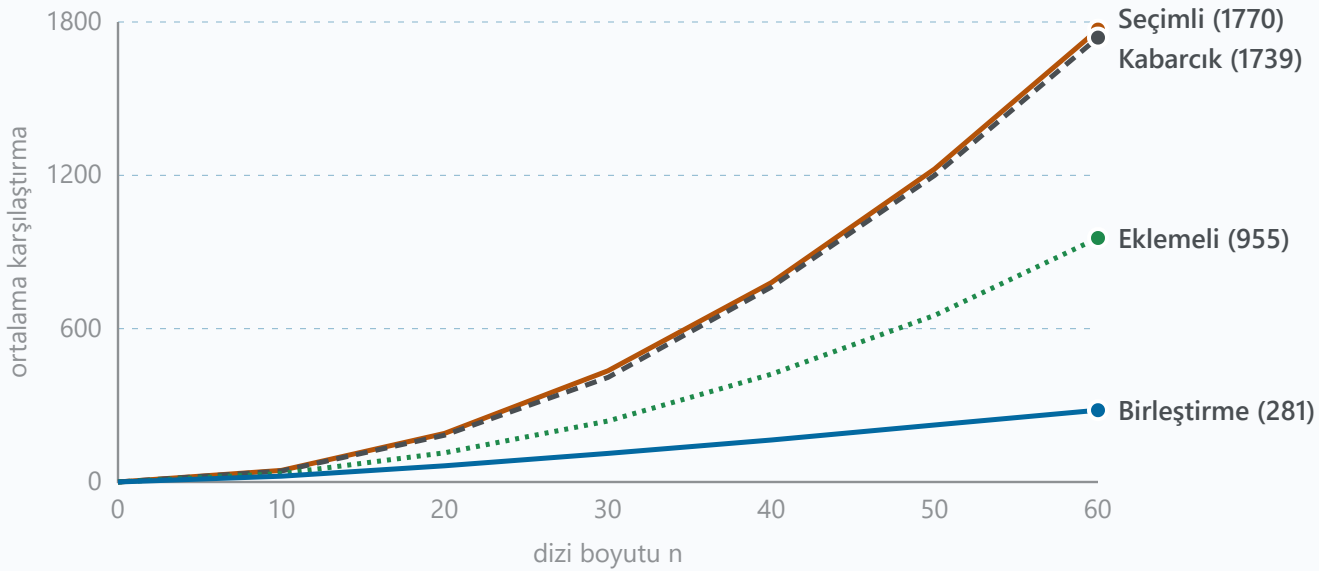
dizi = [1, 3, 4, 5] i = 3

anahtar = 1 j = -1

Sıralama Karşılaştırması

Algoritma	Temel fikir	En kötü	En iyi	Ek alan
Seçimli	En küçüğü seç ve yerleştir	$O(n^2)$	$O(n^2)$	O(1)
Kabarcık	Komşuları değiştir	$O(n^2)$	O(n) (erken durma ile)	O(1)
Eklemeli	Öğeyi sıralı bölüme ekle	$O(n^2)$	O(n)	O(1)
Birleştirme	Böl, sırala, birleştir	O(n log n)	O(n log n)	O(n)

Sıralama: Kaç Karşılaştırma Yapılıyor?



Her n için 20 rastgele dizi dört algoritmayla gerçekten sıralandı; ortalamalar çizildi. n = 60'ta seçimli 1770, birleştirme sıralaması yalnızca 281 karşılaştırma yaptı.

ÖZET

Bölüm Özeti

- Algoritma, girdiyi çıktıya dönüştüren açık ve sonlu adımlar bütünüdür.
- Sözde kod algoritmayı belirli bir dile bağlı kalmadan yazar.
- Elle izleme ve çeşitli test durumları doğruluğu değerlendirmeye yardım eder.
- Doğrusal arama sıralama istemez; ikili arama sıralı veride aralığı yarılar.
- Algoritmalar yalnızca sonuçla değil, zaman ve bellek maliyetiyle de karşılaştırılır.

Kontrol Soruları

1. İyi bir problem tanımında hangi bilgiler bulunur?
2. Sözde kod neden programlama dilinden önce kullanılır?
3. Faktöriyel algoritmasında 0 girdisi nasıl ele alınır?
4. İkili arama neden sıralı veri ister?
5. O(n) ile O(log n) neyi karşılaştırır?
6. Seçimli sıralama ile kabarcık sıralamasının temel farkı nedir?

ÖZET

Alıştırmalar

1. Üç sayıyı (a, b, c) küçükten büyüğe yazan sözde kodu yazın; a = 9, b = 2, c = 5 için izleme tablosu çıkarın.
2. 1'den n'ye kadar olan **çift** sayıların toplamını bulan algoritmayı yazın. n = 0 ve n = 7 için sonucu izleyin.
3. Faktöriyel algoritmasını n = 21 girildiğinde uyarı verecek şekilde değiştirin (kısıt $0 \le n \le 20$).
4. [6, 2, 8, 3] dizisini seçimli, kabarcık ve eklemeli sıralamayla elle sıralayın; her birinde kaç karşılaştırma yapıldığını sayın.
5. [2, 5, 9, 14, 20, 27, 33] dizisinde ikili aramayla 27 ve 4 değerlerini arayın; her turda sol, sag, orta değerlerini yazın.
6. Bir sayının basamaklarının toplamını (örn. 472 → 13) bulan algoritmayı % ve / işlemleriyle yazın.

▶ **İpuçları**

ÖZET

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, Introduction to Algorithms, 4. baskı, MIT Press, 2022.
- Levitin, Introduction to the Design and Analysis of Algorithms, 3. baskı, Pearson, 2012.
- Sedgewick ve Wayne, Algorithms, 4. baskı, Addison-Wesley, 2011 (**kitap sitesi**).
- Sedgewick ve Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **4.1 Performance**, **4.2 Sorting and Searching**.
- Oracle, Java SE 21 API: **java.util.Arrays** (sort, binarySearch).

Son

Algoritmalar

Sonraki sunum: Akış Diyagramları · Çalışma soruları · Simülatörler: Sıralama Algoritmaları Görselleştirici, Akış Diyagramı Yürütücü · Terim sözlüğü