Bölüm 3: Diziler

Veri Yapıları

Dizi kavramı | Java'da diziler | Stack ve heap | Dizi algoritmaları | Çok boyutlu diziler | Karmaşıklık | Sıralama | Sezar şifresi

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Dizi modeli

Dizilerin bellekteki yerleşimini ve indisleme mantığını açıklamak.

Java kullanımı

Tek ve çok boyutlu dizileri doğru biçimde tanımlayıp kullanmak.

Algoritmalar

Arama, ekleme, silme ve sıralama işlemlerini diziler üzerinde uygulamak.

Maliyet analizi

Dizi işlemlerinin zaman ve alan maliyetlerini karşılaştırmak.

TEMEL KAVRAMLAR

Diziler (Arrays)

- **Aynı türden** birden çok veriyi tek bir isim altında, düzenli olarak saklar.
- Boyutu dizi oluşturulurken belirlenir ve **sonradan değiştirilemez**.
- Öğeler bellekte **ardışık** (contiguous) yuvalarda, yan yana tutulur.
- Öğelerin değerleri her zaman güncellenebilir; "ekleme/silme" ise öğeleri kaydırmak demektir.
- Matris, resim (pikseller) ve ses (örnekler) gibi verileri temsil etmek için kullanılır.

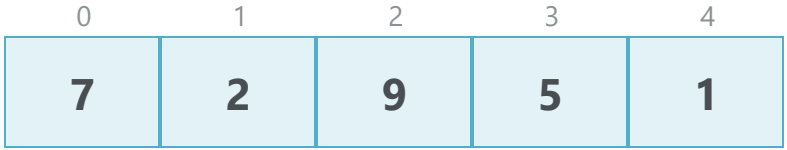


dizi.length = 5 · ilk indis 0 · son indis 4

TEMEL KAVRAMLAR

Dizi Kullanımı: İndisle Erişim

- Dizi öğelerine **indis** (index) numarasıyla erişilir: `diziAdi[indis]`.
- İlk öğenin indisi **0**, son öğeninki **length − 1**'dir.
- Öğeler ardışık ve eşit boyutlu olduğundan her öğenin adresi tek bir çarpma ve toplamayla hesaplanır.



Bir hücreye tıkla: üst sıra indis, alt sıra değer.

```
int[] dizi = {7, 2, 9, 5, 1};
int ilkOge = dizi[0];      // 7
int sonOge = dizi[dizi.length - 1];      // 1
```

TEMEL KAVRAMLAR

Dizi Türleri

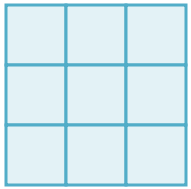
Tek boyutlu



Öğeler tek bir satırda saklanır.

```
int[] a = new int[5];
```

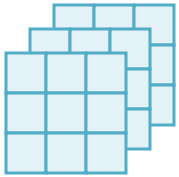
İki boyutlu



Öğeler satır ve sütunlarda saklanır (matris).

```
int[][] m = new int[3][3];
```

Çok boyutlu



Daha karmaşık düzenler: ör. zaman × satır × sütun.

```
int[][][] k = new int[3][3][3];
```

TEMEL KAVRAMLAR

Diziler: Avantajlar ve Dezavantajlar

Avantajları

+ İndisle doğrudan, hızlı erişim: **O(1)**.
+ Verileri gruplar ve eklenme sırasını korur.
+ Hem ilkel tipleri (`int` , `char` ...) hem nesneleri tutabilir.
+ Ardışık bellek: ek referans yükü yok, önbellek (cache) dostu.

Dezavantajları

– Sabit boyutludur; dolunca büyütülemez, yenisi oluşturulup kopyalanır.
– Araya ekleme ve silme öğe kaydırmayı gerektirir: **O(n)**.
– Kullanılmayan kapasite boşa gider.
– Dizinin sıralama, arama gibi metotları yoktur; bunlar için `java.util.Arrays` kullanılır.

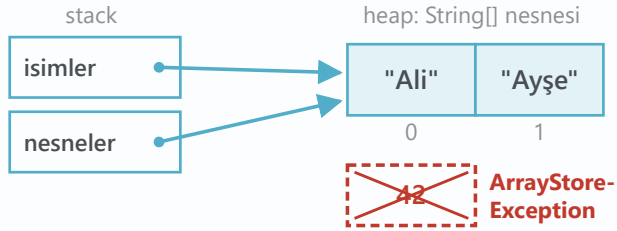
Java dizileri **kovaryanttır** (covariant): bu hem esneklik hem de bir tür güvenliği açığı getirir.

TEMEL KAVRAMLAR

Dizi Kovaryansı ve ArrayStoreException

- `String`, `Object`'in alt türü olduğundan `String[]` da `Object[]`'in alt türü sayılır.
- Böylece `Object[]` alan tek bir metot her referans dizisiyle çalışır.
- Bedeli: yanlış türde atama **derleme zamanında değil**, çalışma zamanında yakalanır.

```
String[] isimler = {"Ali", "Ayşe"};
Object[] nesneler = isimler; // derlenir
nesneler[0] = 42;    // derlenir, ama
                     // çalışırken istisna!
```



Esneklik

Genel amaçlı metotlar yazılabilir.

```
static void yazdir(Object[] a)
```

Karşılaştırma

Jenerik (generic) listeler kovaryant değildir; hata derleme sırasında yakalanır.

```
List<Object> l = new ArrayList<String>
(); // hata
```

Dizi Tanımlama

```
VeriTipi[] diziAdi = new VeriTipi[diziBoyutu];
```

VeriTipi
Dizide saklanacak verilerin türü.
`int, double, String…`

diziAdi
Diziyi gösteren referans değişkenin adı.
`sayilar`

diziBoyutu
Öğe sayısı; çalışma zamanında hesaplanabilir.
`5, n, 2 * n`

```
int[] sayilar = new int[5];
// → [0, 0, 0, 0, 0]
```

`new` ile oluşturulan dizinin her öğesi, türünün **varsayılan değerini** alır.

Tür	Varsayılan değer
`int`, `long`…	0
`double`	0.0
`boolean`	false
`char`	'\u0000'
nesne (`String`…)	null

JAVA'DA DİZİLER

Erişim, İlklendirme ve `length`

İlklendirme
İlklendirici (initializer) listesi başlangıç değerlerini atar; boyut yazılmaz, **öğe sayısı** boyutu belirler.

Erişim
İndisler 0'dan başlar. Okuma ve yazma aynı sözdizimiyle yapılır.

length
Dizinin öğe sayısını verir. Metot değil **alan**dır: parantez yok.

```
int[] sayilar = {10, 20, 30, 40, 50};   // 5 ilklendirici → boyut 5
int ilkOge    = sayilar[0];            // 10
int ikinciOge = sayilar[1];            // 20
sayilar[4] = 55;                         // son öğe güncellendi
int diziBoyutu   = sayilar.length;     // 5

sayilar = new int[] {1, 2, 3};          // sonradan atamada new int[] gerekir
int[] hatali = new int[3] {1, 2, 3};   // derleme hatası: boyut + liste
```

Kod İzleme: Dizide Okuma ve Yazma



İlklendirici listesinde **5 değer** var → heap'te 5 öğeli bir `int[]` oluşur; `sayilar` yalnızca onun **referansını** tutar.

Adım 1 / 6

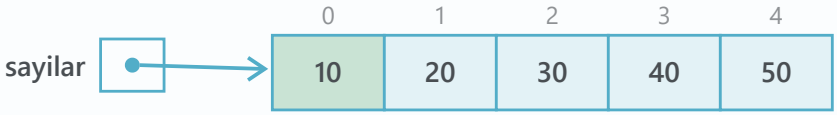
```
▶ int[] sayilar = {10, 20, 30, 40, 50};
2 int ilkOge    = sayilar[0];
3 int ikinciOge = sayilar[1];
4 sayilar[4] = 55;
5 int diziBoyutu   = sayilar.length;
6 sayilar = new int[] {1, 2, 3};
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

Son satır (`new int[3] {1, 2, 3}`) derlenmez.

Kod İzleme: Dizide Okuma ve Yazma



Okuma: `sayilar[0]` ilk hücre. Adres = başlangıç + 0 × 4 → tek adımda, O(1).

Adım 2 / 6

```
1 int[] sayilar = {10, 20, 30, 40, 50};
▶ int ilkOge    = sayilar[0];
3 int ikinciOge = sayilar[1];
4 sayilar[4] = 55;
5 int diziBoyutu   = sayilar.length;
6 sayilar = new int[] {1, 2, 3};
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

`ilkOge = 10`

Son satır (`new int[3] {1, 2, 3}`) derlenmez.

Kod İzleme: Dizide Okuma ve Yazma



İndis 1 = **ikinci** öğe (indisler 0'dan başlar).

Adım 3 / 6

```
1 int[] sayilar = {10, 20, 30, 40, 50};
2 int ilkOge    = sayilar[0];
▶ int ikinciOge = sayilar[1];
4 sayilar[4] = 55;
5 int diziBoyutu   = sayilar.length;
6 sayilar = new int[] {1, 2, 3};
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

`ilkOge = 10` `ikinciOge = 20`

Son satır (`new int[3] {1, 2, 3}`) derlenmez.

Kod İzleme: Dizide Okuma ve Yazma



Yazma: aynı sözdizimi, atamanın solunda. Son hücre (indis 4 = length − 1) 50 → **55**.

Adım 4 / 6

```
1 int[] sayilar = {10, 20, 30, 40, 50};
2 int ilkOge    = sayilar[0];
3 int ikinciOge = sayilar[1];
▶ sayilar[4] = 55;
5 int diziBoyutu   = sayilar.length;
6 sayilar = new int[] {1, 2, 3};
```

DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 55]`

`ilkOge = 10` `ikinciOge = 20`

Son satır (`new int[3] {1, 2, 3}`) derlenmez.

Kod İzleme: Dizide Okuma ve Yazma



`length` bir **alan**dır (parantez yok) → 5. Boyut, dizi oluşturulurken sabitlendi.

Adım 5 / 6

```
1 int[] sayilar = {10, 20, 30, 40, 50};
2 int ilkOge    = sayilar[0];
3 int ikinciOge = sayilar[1];
4 sayilar[4] = 55;
▶ int diziBoyutu   = sayilar.length;
6 sayilar = new int[] {1, 2, 3};
```

DEĞİŞKENLER

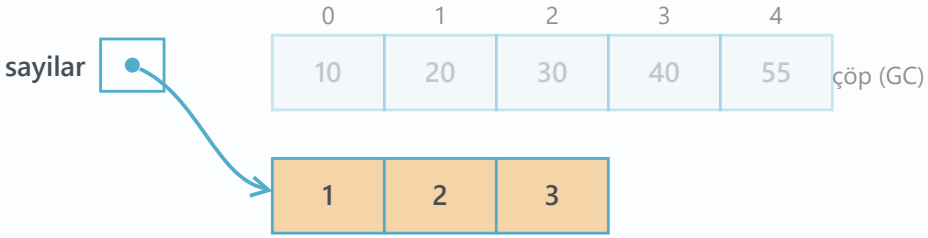
`sayilar = [10, 20, 30, 40, 55]`

`ilkOge = 10` `ikinciOge = 20`

`diziBoyutu = 5`

Son satır (`new int[3] {1, 2, 3}`) derlenmez.

Kod İzleme: Dizide Okuma ve Yazma



Sonradan atamada `new int[]` gerekir: **yeni** 3 öğeli dizi oluşur, `sayilar` onu gösterir. Eski dizi değişmedi; referansı kalmadığı için GC toplayabilir. `diziBoyutu` hâlâ 5: o bir kopyaydı.

Adım 6 / 6

```
1 int[] sayilar = {10, 20, 30, 40, 50};
2 int ilkOge    = sayilar[0];
3 int ikinciOge = sayilar[1];
4 sayilar[4] = 55;
5 int diziBoyutu   = sayilar.length;
▶ sayilar = new int[] {1, 2, 3};
```

DEĞİŞKENLER

`sayilar = [1, 2, 3]`

`ilkOge = 10` `ikinciOge = 20`

`diziBoyutu = 5`

Son satır (`new int[3] {1, 2, 3}`) derlenmez.

Örnek Kod: Tanımla, Doldur, Yazdır

```java
public class DiziOrnegi {
    public static void main(String[] args) {
        int[] sayilar = new int[5];            // 5 öğeli dizi
        for (int i = 0; i < sayilar.length; i++)
            sayilar[i] = (i + 1) * 10;             // 10, 20, 30, 40, 50

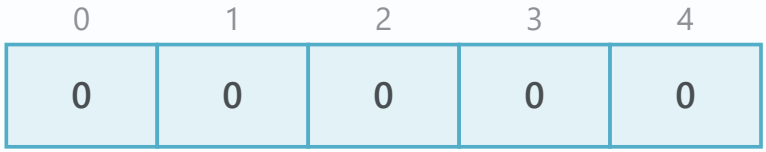
        System.out.println("Dizinin birinci öğesi: " + sayilar[0]);
        System.out.println("Dizinin üçüncü öğesi: " + sayilar[2]);

        System.out.print("Dizi Öğeleri: ");
        for (int x : sayilar)                      // for-each: indis gerekmiyorsa
            System.out.print(x + " ");
    }
}
```

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri: 10 20 30 40 50
```

Beş ayrı `sayilar[k] = …` ataması yerine döngü kullanmak, boyut değişse de doğru çalışır. Döngü sınırı her zaman `i < sayilar.length`.

Kod İzleme: Doldur ve Yazdır



`new int[5]` : 5 hücre ayrılır ve hepsi `int` 'in varsayılan değeri **0** ile başlar.

Adım 1 / 15

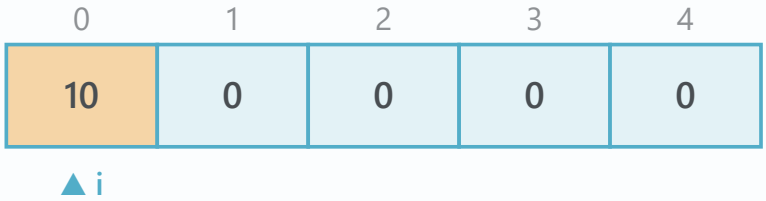
```
▶ int[] sayilar = new int[5];
2 for (int i = 0; i < sayilar.length; i++)
3     sayilar[i] = (i + 1) * 10;
4 System.out.println("Dizinin birinci öğesi: "
5         + sayilar[0]);
6 System.out.println("Dizinin üçüncü öğesi: "
7         + sayilar[2]);
8 System.out.print("Dizi Öğeleri: ");
9 for (int x : sayilar)
10    System.out.print(x + " ");
```

DEĞİŞKENLER

`sayilar = [0, 0, 0, 0, 0]`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



i = 0 < 5 → **sayilar[0] = (0 + 1) × 10 = 10**.

Adım 2 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
▶      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

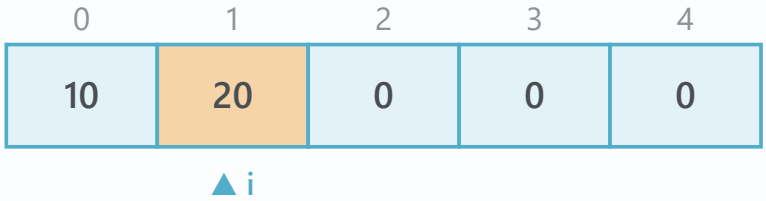
DEĞİŞKENLER

`sayilar = [10, 0, 0, 0, 0]`

`i = 0`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



i = 1 < 5 → **sayilar[1] = (1 + 1) × 10 = 20**.

Adım 3 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
▶      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

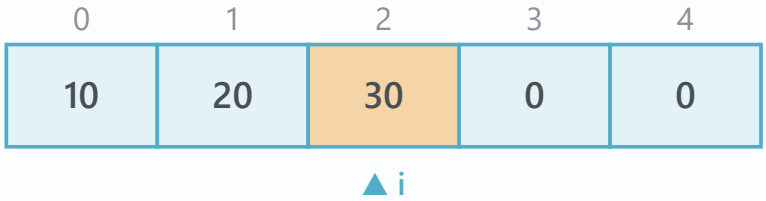
DEĞİŞKENLER

`sayilar = [10, 20, 0, 0, 0]`

`i = 1`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



i = 2 < 5 → **sayilar[2] = (2 + 1) × 10 = 30**.

Adım 4 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
▶      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

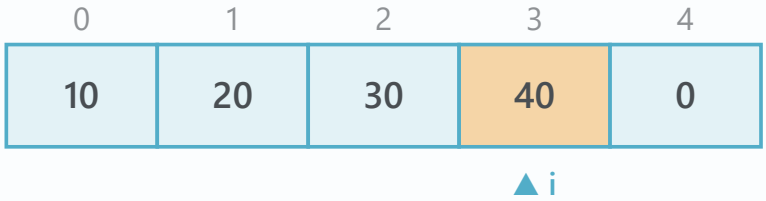
DEĞİŞKENLER

`sayilar = [10, 20, 30, 0, 0]`

`i = 2`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



i = 3 < 5 → **sayilar[3] = (3 + 1) × 10 = 40**.

Adım 5 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
▶      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

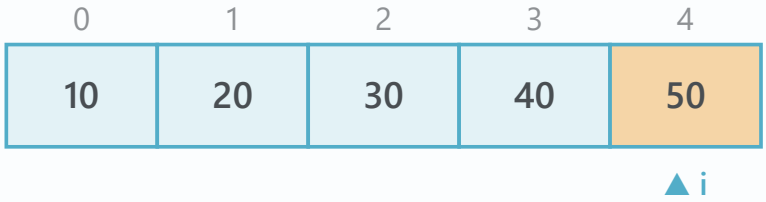
DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 0]`

`i = 3`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



i = 4 < 5 → **sayilar[4] = (4 + 1) × 10 = 50**.

Adım 6 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
▶      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

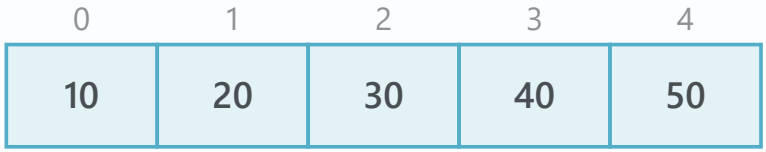
DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

`i = 4`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



i = 5 için **5 < 5** yanlış → döngü biter. Sınır `sayilar.length` olduğundan boyut değişse de kod doğru kalır.

Adım 7 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
3      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

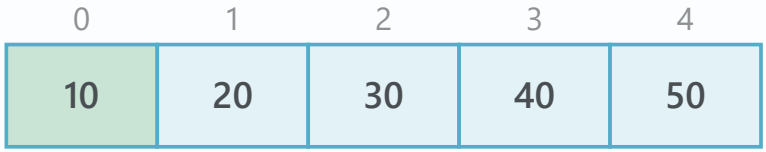
DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

`i = 5`

ÇIKTI

Kod İzleme: Doldur ve Yazdır



İndis 0 okunur → 10; `println` satır sonu ekler.

Adım 8 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
3      sayilar[i] = (i + 1) * 10;
▶  System.out.println("Dizinin birinci öğesi: "
▶          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

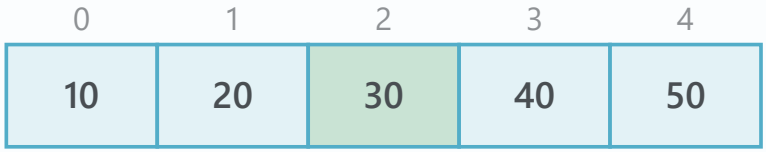
DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

ÇIKTI

```
Dizinin birinci öğesi: 10
```

Kod İzleme: Doldur ve Yazdır



Üçüncü öğe **indis 2**'dedir → 30.

Adım 9 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
3      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
▶  System.out.println("Dizinin üçüncü öğesi: "
▶          + sayilar[2]);
8  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

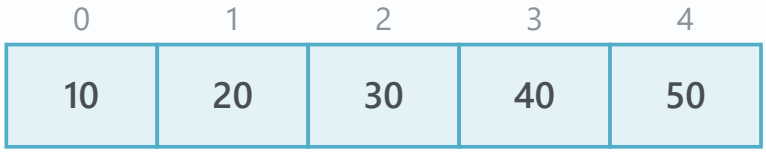
DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
```

Kod İzleme: Doldur ve Yazdır



`print` satır sonu eklemez; öğeler aynı satıra yazılacak.

Adım 10 / 15

```
1  int[] sayilar = new int[5];
2  for (int i = 0; i < sayilar.length; i++)
3      sayilar[i] = (i + 1) * 10;
4  System.out.println("Dizinin birinci öğesi: "
5          + sayilar[0]);
6  System.out.println("Dizinin üçüncü öğesi: "
7          + sayilar[2]);
▶  System.out.print("Dizi Öğeleri: ");
9  for (int x : sayilar)
10     System.out.print(x + " ");
```

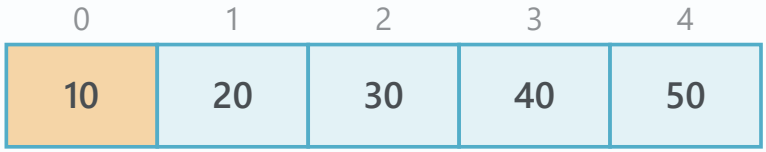
DEĞİŞKENLER

`sayilar = [10, 20, 30, 40, 50]`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri:
```

Kod İzleme: Doldur ve Yazdır



for-each: **x = 10** (indis 0). İndis değişkeni gerekmiyorsa daha okunaklı.

Adım 11 / 15

```
1 int[] sayilar = new int[5];
2 for (int i = 0; i < sayilar.length; i++)
3     sayilar[i] = (i + 1) * 10;
4 System.out.println("Dizinin birinci öğesi: "
5         + sayilar[0]);
6 System.out.println("Dizinin üçüncü öğesi: "
7         + sayilar[2]);
8 System.out.print("Dizi Öğeleri: ");
▶ for (int x : sayilar)
▶     System.out.print(x + " ");
```

DEĞİŞKENLER

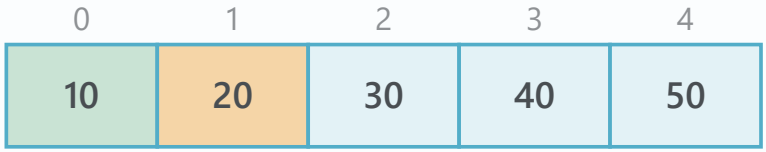
`sayilar = [10, 20, 30, 40, 50]`

`x = 10`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri: 10
```

Kod İzleme: Doldur ve Yazdır



x = 20 yazdırılır.

Adım 12 / 15

```
1 int[] sayilar = new int[5];
2 for (int i = 0; i < sayilar.length; i++)
3     sayilar[i] = (i + 1) * 10;
4 System.out.println("Dizinin birinci öğesi: "
5         + sayilar[0]);
6 System.out.println("Dizinin üçüncü öğesi: "
7         + sayilar[2]);
8 System.out.print("Dizi Öğeleri: ");
▶ for (int x : sayilar)
▶     System.out.print(x + " ");
```

DEĞİŞKENLER

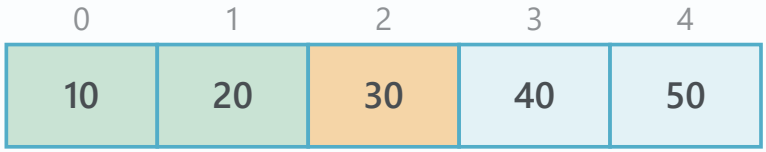
`sayilar = [10, 20, 30, 40, 50]`

`x = 20`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri: 10 20
```

Kod İzleme: Doldur ve Yazdır



x = 30 yazdırılır.

Adım 13 / 15

```
1 int[] sayilar = new int[5];
2 for (int i = 0; i < sayilar.length; i++)
3     sayilar[i] = (i + 1) * 10;
4 System.out.println("Dizinin birinci öğesi: "
5         + sayilar[0]);
6 System.out.println("Dizinin üçüncü öğesi: "
7         + sayilar[2]);
8 System.out.print("Dizi Öğeleri: ");
▶ for (int x : sayilar)
▶     System.out.print(x + " ");
```

DEĞİŞKENLER

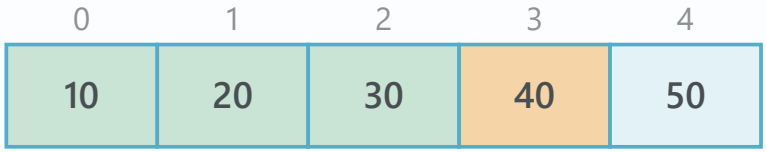
`sayilar = [10, 20, 30, 40, 50]`

`x = 30`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri: 10 20 30
```

Kod İzleme: Doldur ve Yazdır



x = 40 yazdırılır.

Adım 14 / 15

```
1 int[] sayilar = new int[5];
2 for (int i = 0; i < sayilar.length; i++)
3     sayilar[i] = (i + 1) * 10;
4 System.out.println("Dizinin birinci öğesi: "
5         + sayilar[0]);
6 System.out.println("Dizinin üçüncü öğesi: "
7         + sayilar[2]);
8 System.out.print("Dizi Öğeleri: ");
▶ for (int x : sayilar)
▶     System.out.print(x + " ");
```

DEĞİŞKENLER

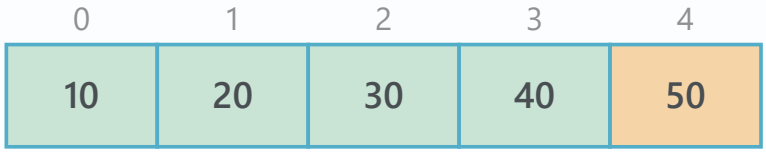
`sayilar = [10, 20, 30, 40, 50]`

`x = 40`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri: 10 20 30 40
```

Kod İzleme: Doldur ve Yazdır



x = 50 yazdırılır. Tüm öğeler gezildi → döngü biter.

Adım 15 / 15

```
1 int[] sayilar = new int[5];
2 for (int i = 0; i < sayilar.length; i++)
3     sayilar[i] = (i + 1) * 10;
4 System.out.println("Dizinin birinci öğesi: "
5         + sayilar[0]);
6 System.out.println("Dizinin üçüncü öğesi: "
7         + sayilar[2]);
8 System.out.print("Dizi Öğeleri: ");
▶ for (int x : sayilar)
▶     System.out.print(x + " ");
```

DEĞİŞKENLER

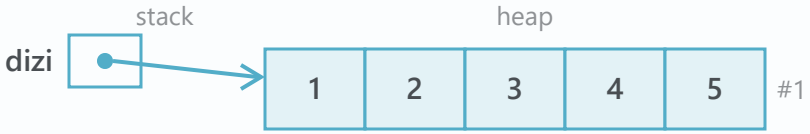
`sayilar = [10, 20, 30, 40, 50]`

`x = 50`

ÇIKTI

```
Dizinin birinci öğesi: 10
Dizinin üçüncü öğesi: 30
Dizi Öğeleri: 10 20 30 40 50
```

Dizi Kopyalama: Referans mı, Kopya mı?



Dizi bir **nesnedir**: heap'te oluşur (#1). Stack'teki `dizi` değişkeni yalnızca onun adresini tutar.

Adım 1 / 4

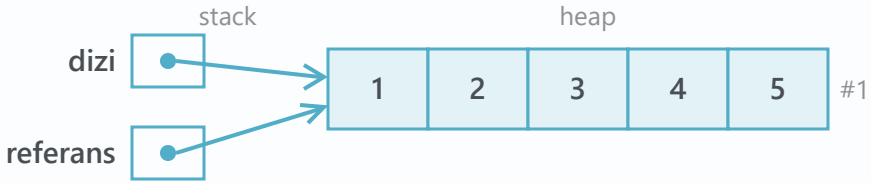
```
▶ int[] dizi = {1, 2, 3, 4, 5};
2 int[] referans = dizi;   // aynı nesne
3 int[] kopya =
4     Arrays.copyOf(dizi, dizi.length);
5 referans[0] = 99;
```

DEĞİŞKENLER

`dizi = →#1`

`#1 = [1, 2, 3, 4, 5]`

Dizi Kopyalama: Referans mı, Kopya mı?



`=` yalnızca **adresi** kopyalar: iki değişken **aynı** diziyi (#1) gösterir. Yeni dizi oluşmadı.

Adım 2 / 4

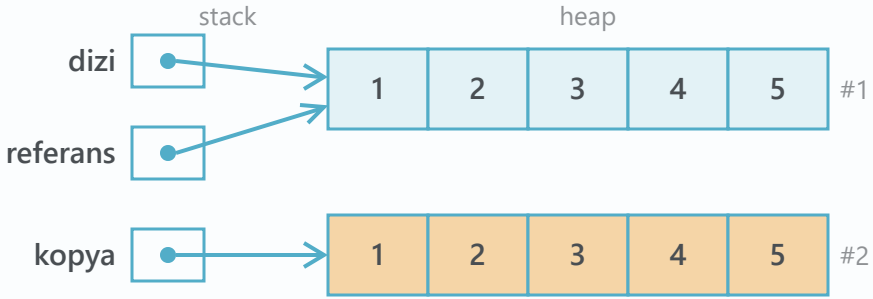
```
1 int[] dizi = {1, 2, 3, 4, 5};
▶ int[] referans = dizi;   // aynı nesne
3 int[] kopya =
4     Arrays.copyOf(dizi, dizi.length);
5 referans[0] = 99;
```

DEĞİŞKENLER

`dizi = →#1` `referans = →#1`

`#1 = [1, 2, 3, 4, 5]`

Dizi Kopyalama: Referans mı, Kopya mı?



`Arrays.copyOf` heap'te **yeni** bir dizi (#2) oluşturur ve öğeleri tek tek kopyalar → O(n).
(`dizi.clone()` ve `System.arraycopy` da öğeleri kopyalar.)

Adım 3 / 4

```
1 int[] dizi = {1, 2, 3, 4, 5};
2 int[] referans = dizi;   // aynı nesne
▶ int[] kopya =
▶     Arrays.copyOf(dizi, dizi.length);
5 referans[0] = 99;
```

DEĞİŞKENLER

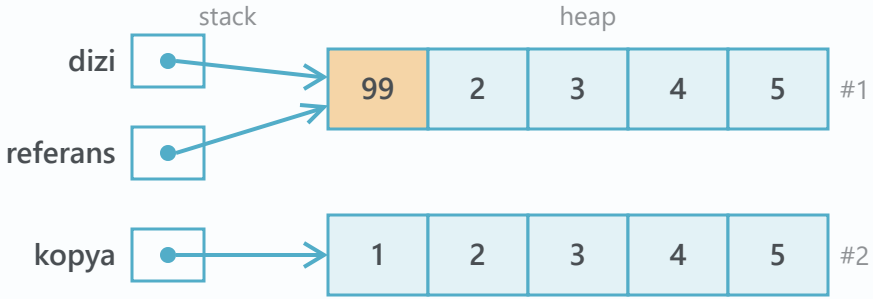
`dizi = →#1` `referans = →#1`

`kopya = →#2`

`#1 = [1, 2, 3, 4, 5]`

`#2 = [1, 2, 3, 4, 5]`

Dizi Kopyalama: Referans mı, Kopya mı?



`referans[0] = 99` #1'i değiştirir → `dizi[0]` da **99** olur. `kopya[0]` ayrı nesnede, hâlâ **1**.

Adım 4 / 4

```
1 int[] dizi = {1, 2, 3, 4, 5};
2 int[] referans = dizi;   // aynı nesne
3 int[] kopya =
4     Arrays.copyOf(dizi, dizi.length);
▶ referans[0] = 99;
```

DEĞİŞKENLER

`dizi = →#1` `referans = →#1`

`kopya = →#2`

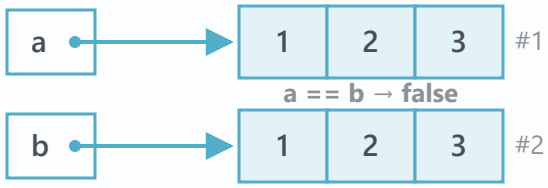
`#1 = [99, 2, 3, 4, 5]`

`#2 = [1, 2, 3, 4, 5]`

Hazır Dizi İşlemleri: `java.util.Arrays`

Metot	Ne yapar?	Maliyet
`Arrays.toString(a)`	`"[1, 2, 3]"` biçiminde metin üretir	O(n)
`Arrays.equals(a, b)`	Uzunluklar ve öğeler tek tek eşit mi?	O(n)
`Arrays.fill(a, v)`	Tüm hücrelere `v` yazar	O(n)
`Arrays.copyOf(a, n)`	Yeni dizi; `n > a.length` ise fazlası varsayılan değer	O(n)
`Arrays.sort(a)`	Artan sıraya dizer (yerinde)	O(n log n)
`Arrays.binarySearch(a, x)`	**Sıralı** dizide ikili arama; yoksa negatif sonuç	O(log n)

```
int[] a = {1, 2, 3};
int[] b = {1, 2, 3};
a == b              // false: ayrı nesne
Arrays.equals(a, b) // true: aynı içerik
println(a)          // [I@1b6d3586
```



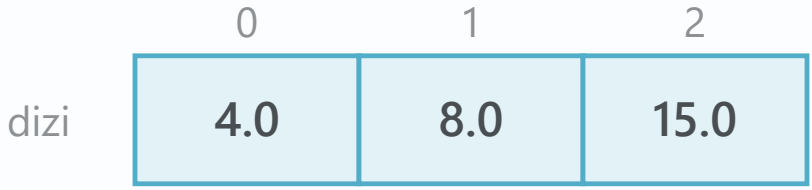
Varargs (Değişken Sayıda Argüman)

- `tip... ad` yazımı, metodun istediği sayıda argüman almasını sağlar.
- Metodun içinde parametre sıradan bir **dizidir**.
- Yalnızca **son** parametre varargs olabilir; metot başına bir tane.
- Hazır bir dizi de doğrudan geçirilebilir.

```
static double ortalama(double... dizi) {
    if (dizi.length == 0)
        return Double.NaN;  // tanımsız
    var toplam = 0.0;
    for (var oge : dizi)
        toplam += oge;
    return toplam / dizi.length;
}
```

Çağrı	dizi.length	Sonuç
`ortalama(4, 8, 15)`	3	9.0
`ortalama(7.5)`	1	7.5
`ortalama()`	0	NaN
`ortalama(new double[] {1, 2})`	2	1.5

Kod İzleme: `ortalama(4, 8, 15)`



Çağrı `ortalama(4, 8, 15)` : derleyici argümanları bir **double[]** dizisine koyar (int → double genişletilir).

Adım 1 / 10

```
▶ static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
6         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

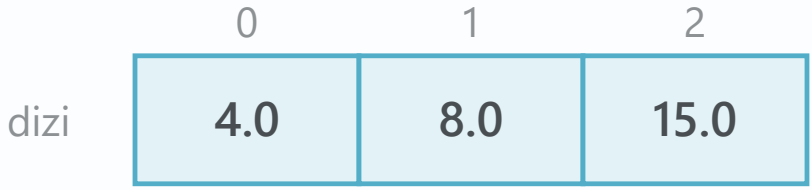
`dizi = [4.0, 8.0, 15.0]`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



`dizi.length` = 3 ≠ 0 → boş dizi durumu değil, devam.

Adım 2 / 10

```
1 static double ortalama(double... dizi) {
▶     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
6         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

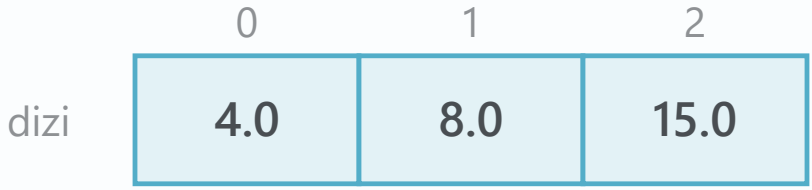
`dizi = [4.0, 8.0, 15.0]`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



`var` : türü sağdaki değerden çıkarılır → `double` .

Adım 3 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
6         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

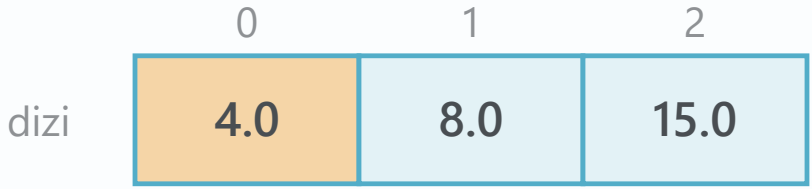
`toplam = 0.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



for-each: **öğe = 4.0** (dizideki 1. öğe).

Adım 4 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
▶     for (var oge : dizi)
6         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

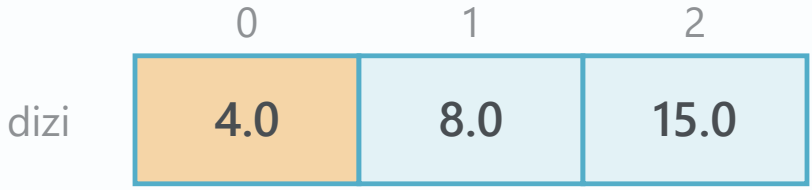
`toplam = 0.0` `oge = 4.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



toplam = 0.0 + 4.0 = **4.0**.

Adım 5 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
▶         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

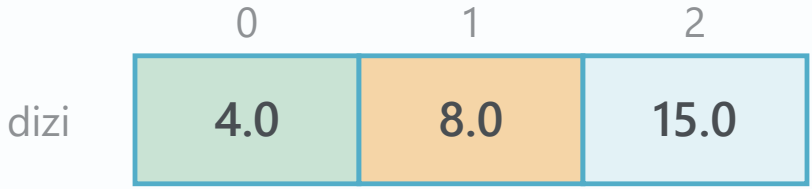
`toplam = 4.0` `oge = 4.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



for-each: **öğe = 8.0** (dizideki 2. öğe).

Adım 6 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
▶     for (var oge : dizi)
6         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

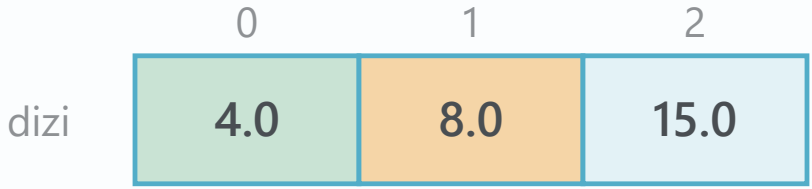
`toplam = 4.0` `oge = 8.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



toplam = 4.0 + 8.0 = **12.0**.

Adım 7 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
▶         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

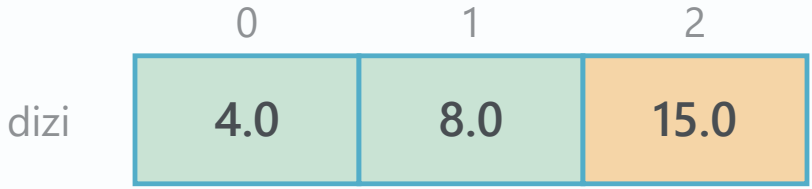
`toplam = 12.0` `oge = 8.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



for-each: **öğe = 15.0** (dizideki 3. öğe).

Adım 8 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
▶     for (var oge : dizi)
6         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

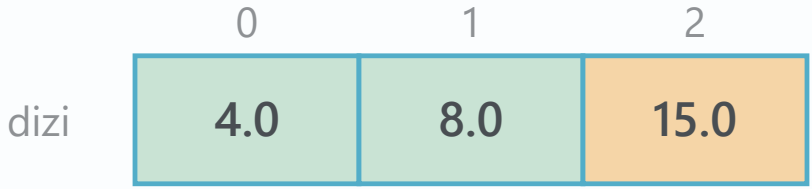
`toplam = 12.0` `oge = 15.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



toplam = 12.0 + 15.0 = **27.0**.

Adım 9 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
▶         toplam += oge;
7     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

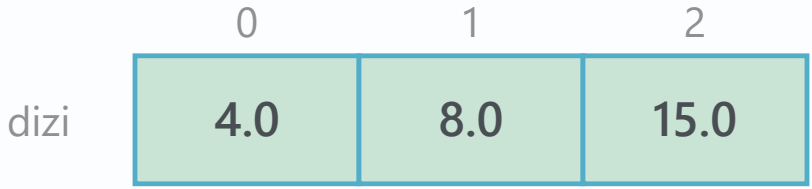
`toplam = 27.0` `oge = 15.0`

ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Kod İzleme: `ortalama(4, 8, 15)`



Döngü bitti. **return 27.0 / 3 = 9.0**. `ortalama()` çağrılsaydı length 0 olur, 3. satırda `NaN` dönerdi.

Adım 10 / 10

```
1 static double ortalama(double... dizi) {
2     if (dizi.length == 0)
3         return Double.NaN;  // tanımsız
4     var toplam = 0.0;
5     for (var oge : dizi)
6         toplam += oge;
▶     return toplam / dizi.length;
8 }
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0]`

`toplam = 27.0`

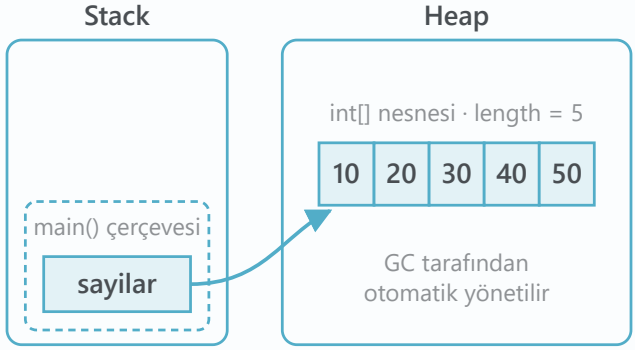
ÇAĞRI YIĞINI

`ortalama(4, 8, 15)`

`main()`

Bellekte Diziler: Stack ve Heap

- **Stack (yığın belleği)**: yerel değişkenler ve metot çağrıları. Metot bitince çerçevesi otomatik silinir.
- **Heap**: `new` ile oluşturulan tüm nesneler. Java'da **her dizi** heap'tedir; stack'te yalnızca referansı durur.
- Heap'i **çöp toplayıcı** (garbage collector, GC) yönetir: hiçbir referansın göstermediği nesneleri kendisi serbest bırakır.



	Stack	Heap
Ne tutulur?	yerel değişkenler, referanslar, çağrı çerçeveleri	nesneler, diziler
Ömür	metot bitene kadar	erişilebilir olduğu sürece
Yönetim	otomatik (çağrı / dönüş)	otomatik (GC)

JAVA'DA DİZİLER

"Statik" ve "Dinamik" Diziler: C ve Java

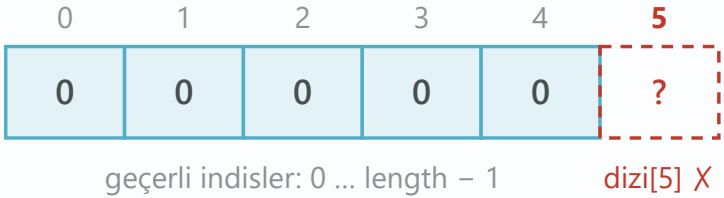
	C / C++	Java
Stack'te dizi	`int a[10];` — boyut derleme zamanında sabit, fonksiyon bitince yok olur	Yok: tüm diziler heap'te
Heap'te dizi	`malloc` / `new int[n]`	`new int[n]` (ilklendirici listesi de heap'te)
Boyut	heap'te çalışma zamanında belirlenir	çalışma zamanında belirlenir, sonra sabit
Serbest bırakma	programcı: `free` / `delete[]`	GC otomatik; `dizi = null` yalnızca referansı koparır

```
int[] dinamikDizi = new int[n];   // heap'te n öğelik yer ayrılır
dinamikDizi = null;               // başka referans yoksa GC toplayabilir
```

`System.gc()` yalnızca bir **istektir**, toplamayı garanti etmez; Java'da `delete` yoktur. "Dinamik dizi" adı, dolunca kendiliğinden büyüyen dizi (`ArrayList`) için de kullanılır.

İndis Sınır Dışı Hatası

- Geçerli aralığın dışındaki bir indise erişilirse Java `ArrayIndexOutOfBoundsException` fırlatır (`IndexOutOfBoundsException`'ın alt sınıfı).
- Yakalanmazsa program sonlanır. Java her erişimi denetler; C'de bu denetim yoktur ve sessizce başka belleğe yazılır.




```
int[] dizi = new int[5];
dizi[5] = 1;  // Index 5 out of
              // bounds for length 5
dizi[-1] = 1; // aynı hata
```

```
if (indis >= 0
        && indis < dizi.length) {
    int deger = dizi[indis];
} else {
    System.out.println("Geçersiz");
}
```

Diziyi Kullanıcıdan Okuma

Ortalama, ortanca ve mod programları aynı giriş kalıbıyla başlar; ortak kodu bir yardımcı metotta toplayalım (mod için `int[]` sürümü aynıdır):

```
import java.util.Scanner;

static double[] diziOku(Scanner scanner) {
    System.out.print("Dizi boyutunu girin: ");
    int boyut = scanner.nextInt();
    double[] dizi = new double[boyut];  // boyut çalışma zamanında
    for (int i = 0; i < boyut; i++) {
        System.out.print((i + 1) + ". öğeyi girin: "); // 1'den say
        dizi[i] = scanner.nextDouble();
    }
    return dizi;
}
```

Ortalama
Toplam / öğe sayısı
`O(n)`

Ortanca (median)
Sıralı dizinin ortası
`O(n log n)`

Mod
En sık tekrar eden değer
`O(n log n)`

Kod İzleme: `diziOku(sc)`

henüz dizi yok (boyut bilinmiyor)

Kullanıcıdan önce boyut istenir. `print` satır sonu eklemez; imleç aynı satırda bekler.

Adım 1 / 12

```
1  static double[] diziOku(Scanner scanner) {
▶    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

—

ÇIKTI

```
Dizi boyutunu girin:
```

Kod İzleme: `diziOku(sc)`

henüz dizi yok (boyut bilinmiyor)

Kullanıcı **4** yazar; `nextInt()` okur. (Konsolda kullanıcının yazdığı da görünür.)

Adım 2 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
▶    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

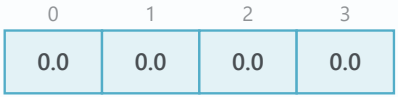
DEĞİŞKENLER

`boyut = 4`

ÇIKTI

```
Dizi boyutunu girin: 4
```

Kod İzleme: `diziOku(sc)`



Boyut **çalışma zamanında** belli oldu → `new double[4]`, hücreler 0.0.

Adım 3 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
▶    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

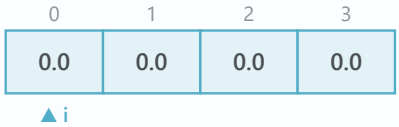
`boyut = 4`

`dizi = [0.0, 0.0, 0.0, 0.0]`

ÇIKTI

```
Dizi boyutunu girin: 4
```

Kod İzleme: `diziOku(sc)`



i = 0, ama kullanıcıya **1.** öğe diye sorulur (insanlar 1'den sayar).

Adım 4 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
▶      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

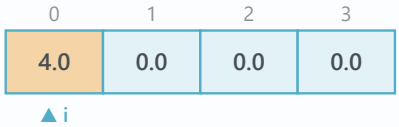
`dizi = [0.0, 0.0, 0.0, 0.0]`

`i = 0`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin:
```

Kod İzleme: `diziOku(sc)`



Kullanıcı 4 yazar → **dizi[0] = 4.0**.

Adım 5 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
▶      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

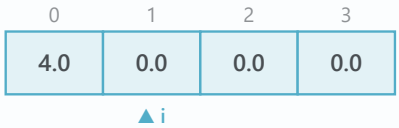
`dizi = [4.0, 0.0, 0.0, 0.0]`

`i = 0`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
```

Kod İzleme: `diziOku(sc)`



i = 1, ama kullanıcıya **2.** öğe diye sorulur (insanlar 1'den sayar).

Adım 6 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
▶      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

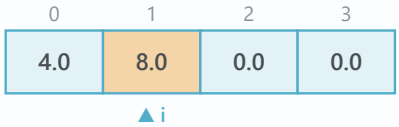
`dizi = [4.0, 0.0, 0.0, 0.0]`

`i = 1`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin:
```

Kod İzleme: `diziOku(sc)`



Kullanıcı 8 yazar → **dizi[1] = 8.0**.

Adım 7 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
▶      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

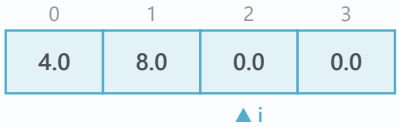
`dizi = [4.0, 8.0, 0.0, 0.0]`

`i = 1`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin: 8
```

Kod İzleme: `diziOku(sc)`



i = 2, ama kullanıcıya **3.** öğe diye sorulur (insanlar 1'den sayar).

Adım 8 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
▶      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

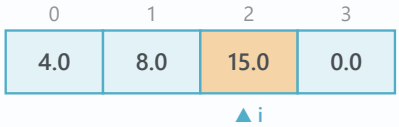
`dizi = [4.0, 8.0, 0.0, 0.0]`

`i = 2`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin: 8
3. öğeyi girin:
```

Kod İzleme: `diziOku(sc)`



Kullanıcı 15 yazar → **dizi[2] = 15.0**.

Adım 9 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
▶      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

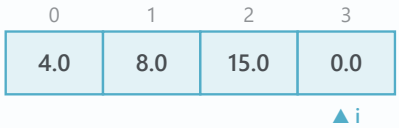
`dizi = [4.0, 8.0, 15.0, 0.0]`

`i = 2`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin: 8
3. öğeyi girin: 15
```

Kod İzleme: `diziOku(sc)`



i = 3, ama kullanıcıya **4.** öğe diye sorulur (insanlar 1'den sayar).

Adım 10 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
▶      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

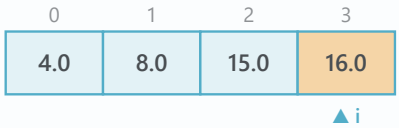
`dizi = [4.0, 8.0, 15.0, 0.0]`

`i = 3`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin: 8
3. öğeyi girin: 15
4. öğeyi girin:
```

Kod İzleme: `diziOku(sc)`



Kullanıcı 16 yazar → **dizi[3] = 16.0**. i = 4 olunca döngü biter.

Adım 11 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
▶      dizi[i] = scanner.nextDouble();
8    }
9    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

`dizi = [4.0, 8.0, 15.0, 16.0]`

`i = 3`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin: 8
3. öğeyi girin: 15
4. öğeyi girin: 16
```

Kod İzleme: `diziOku(sc)`



Dizinin **referansı** döndürülür; dizi heap'te olduğundan metot bitince de yaşar.

Adım 12 / 12

```
1  static double[] diziOku(Scanner scanner) {
2    System.out.print("Dizi boyutunu girin: ");
3    int boyut = scanner.nextInt();
4    double[] dizi = new double[boyut];
5    for (int i = 0; i < boyut; i++) {
6      System.out.print((i + 1) + ". öğeyi girin: ");
7      dizi[i] = scanner.nextDouble();
8    }
▶    return dizi;
10 }
```

DEĞİŞKENLER

`boyut = 4`

`dizi = [4.0, 8.0, 15.0, 16.0]`

ÇIKTI

```
Dizi boyutunu girin: 4
1. öğeyi girin: 4
2. öğeyi girin: 8
3. öğeyi girin: 15
4. öğeyi girin: 16
```

Ortalama Hesaplama

```java
public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    double[] dizi = diziOku(sc);
    double toplam = 0;
    for (int i = 0; i < dizi.length; i++)
        toplam += dizi[i];
    double ortalama = toplam / dizi.length;
    System.out.println("Dizi: "
        + Arrays.toString(dizi));
    System.out.println("Ortalama: "
        + ortalama);
    sc.close();
}
```

Örnek

dizi = {4, 8, 15, 16, 23}
toplam = 66
ortalama = 66 / 5 =
13.2

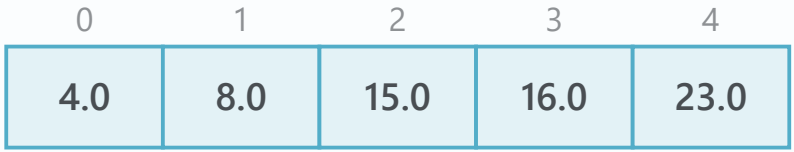
Maliyet

Her öğe bir kez toplanır → **O(n)** zaman, O(1) ek bellek.

Sınır durumu

Boyut 0 ise 0.0 / 0 = `NaN`; kullanıcıya uyarı vermek daha iyi.

Kod İzleme: Ortalama Hesaplama



Bu kez kullanıcı 5 değer girmiş olsun: `diziOku` {4, 8, 15, 16, 23} dizisini döndürür.

Adım 1 / 11

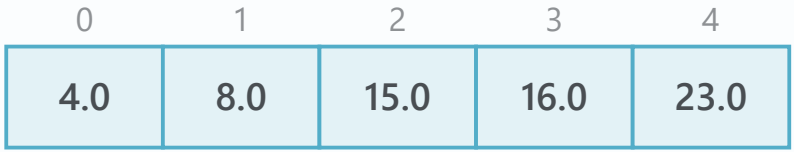
```
▶ double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
4     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

DEĞİŞKENLER

dizi =	[4.0, 8.0, 15.0, 16.0, 23.0]

ÇIKTI

Kod İzleme: Ortalama Hesaplama



Toplayıcı değişken 0 ile başlar (int 0 → double 0.0).

Adım 2 / 11

```
1 double[] dizi = diziOku(sc);
▶ double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
4     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

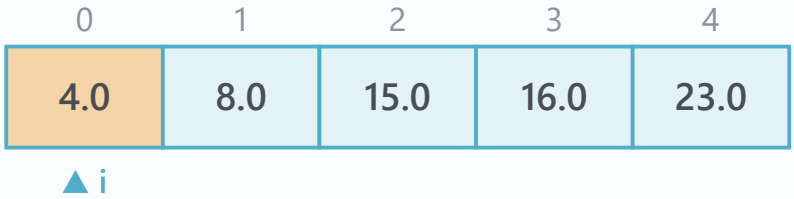
DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 0.0`

ÇIKTI

Kod İzleme: Ortalama Hesaplama



toplam += dizi[0] → 0.0 + 4.0 = **4.0**.

Adım 3 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
▶     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

DEĞİŞKENLER

dizi = [4.0, 8.0, 15.0, 16.0, 23.0]

toplam = 4.0 | i = 0

ÇIKTI

Kod İzleme: Ortalama Hesaplama



toplam += dizi[1] → 4.0 + 8.0 = **12.0**.

Adım 4 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
▶     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

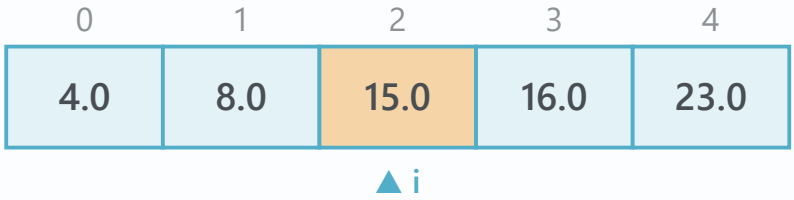
DEĞİŞKENLER

dizi = [4.0, 8.0, 15.0, 16.0, 23.0]

toplam = 12.0 | i = 1

ÇIKTI

Kod İzleme: Ortalama Hesaplama



toplam += dizi[2] → 12.0 + 15.0 = **27.0**.

Adım 5 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
▶     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 27.0` `i = 2`

ÇIKTI

Kod İzleme: Ortalama Hesaplama



toplam += dizi[3] → 27.0 + 16.0 = **43.0**.

Adım 6 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
▶     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

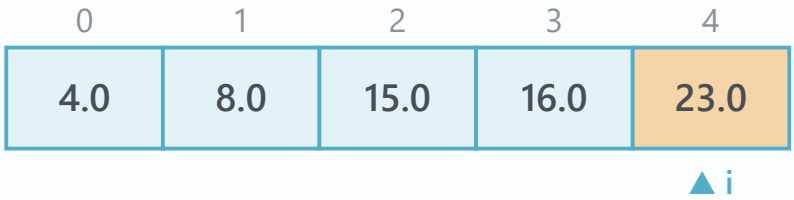
DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 43.0` `i = 3`

ÇIKTI

Kod İzleme: Ortalama Hesaplama



toplam += dizi[4] → 43.0 + 23.0 = **66.0**.

Adım 7 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
▶     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

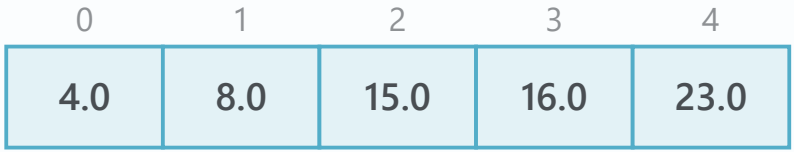
DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 66.0` `i = 4`

ÇIKTI

Kod İzleme: Ortalama Hesaplama



i = 5, 5 < 5 yanlış → döngü biter. Her öğe **bir kez** toplandı → O(n).

Adım 8 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
▶ for (int i = 0; i < dizi.length; i++)
4     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

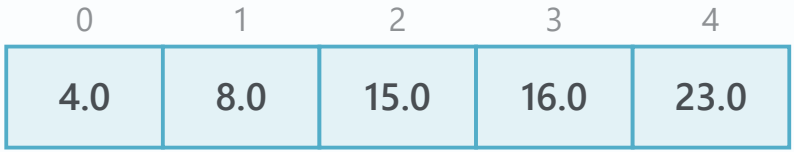
DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 66.0` `i = 5`

ÇIKTI

Kod İzleme: Ortalama Hesaplama



ortalama = 66.0 / 5 = **13.2**. Boyut 0 olsaydı 0.0 / 0 = NaN çıkardı.

Adım 9 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
4     toplam += dizi[i];
▶ double ortalama = toplam / dizi.length;
6 System.out.println("Dizi: "
7     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

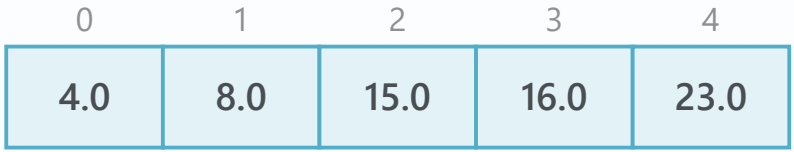
DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 66.0` `ortalama = 13.2`

ÇIKTI

Kod İzleme: Ortalama Hesaplama



`Arrays.toString` diziyi `[a, b, …]` biçiminde yazıya çevirir. (`println(dizi)` yalnızca tür@adres yazardı.)

Adım 10 / 11

```
1 double[] dizi = diziOku(sc);
2 double toplam = 0;
3 for (int i = 0; i < dizi.length; i++)
4     toplam += dizi[i];
5 double ortalama = toplam / dizi.length;
▶ System.out.println("Dizi: "
▶     + Arrays.toString(dizi));
8 System.out.println("Ortalama: "
9     + ortalama);
```

DEĞİŞKENLER

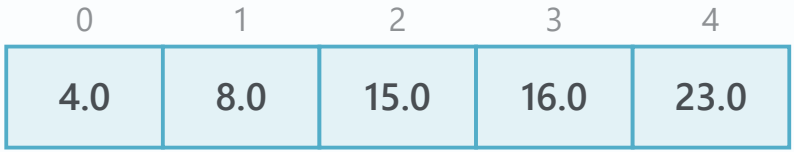
`dizi` = **[4.0, 8.0, 15.0, 16.0, 23.0]**

`toplam` = **66.0** `ortalama` = **13.2**

ÇIKTI

```
Dizi: [4.0, 8.0, 15.0, 16.0, 23.0]
```

Kod İzleme: Ortalama Hesaplama



Sonuç yazdırılır. Toplam O(n) zaman, O(1) ek bellek.

Adım 11 / 11

```
1  double[] dizi = diziOku(sc);
2  double toplam = 0;
3  for (int i = 0; i < dizi.length; i++)
4      toplam += dizi[i];
5  double ortalama = toplam / dizi.length;
6  System.out.println("Dizi: "
7      + Arrays.toString(dizi));
▶  System.out.println("Ortalama: "
▶      + ortalama);
```

DEĞİŞKENLER

`dizi = [4.0, 8.0, 15.0, 16.0, 23.0]`

`toplam = 66.0` `ortalama = 13.2`

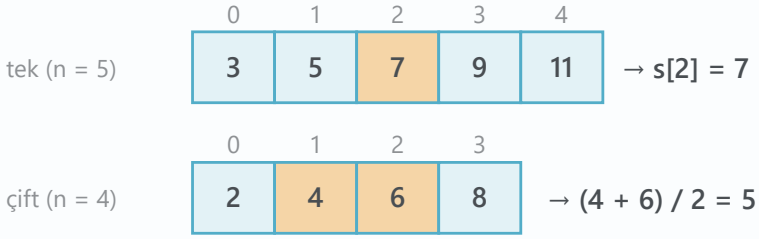
ÇIKTI

```
Dizi: [4.0, 8.0, 15.0, 16.0, 23.0]
Ortalama: 13.2
```

Ortanca (Median) Hesaplama

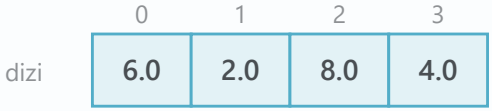
```java
static double ortanca(double[] dizi) {
    double[] s = dizi.clone(); // kopya
    Arrays.sort(s);            // O(n log n)
    int boyut = s.length;
    double ortanca;
    if (boyut % 2 == 0) {      // çift
        int o1 = boyut/2 - 1, o2 = boyut/2;
        ortanca = (s[o1] + s[o2]) / 2;
    } else {                   // tek
        ortanca = s[boyut / 2];
    }
    return ortanca;
}
```

- Ortanca, **sıralı** dizinin ortasındaki değerdir.
- Tek boyutta tam ortadaki, çift boyutta ortadaki iki değerin ortalaması alınır.
- Ortalamanın aksine uç değerlerden (ör. 1000) etkilenmez.



Kod İzleme: `ortanca()`

```
▶ static double ortanca(double[] dizi) {
2     double[] s = dizi.clone(); // kopya
3     Arrays.sort(s);              // O(n log n)
4     int boyut = s.length;
5     double ortanca;
6     if (boyut % 2 == 0) {        // çift
7         int o1 = boyut/2 - 1, o2 = boyut/2;
8         ortanca = (s[o1] + s[o2]) / 2;
9     } else {                     // tek
10        ortanca = s[boyut / 2];
11    }
12    return ortanca;
13 }
```



1. çağrı (çift boyut): `ortanca` 4 öğeli bir diziyle çağrılır.

Adım 1 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

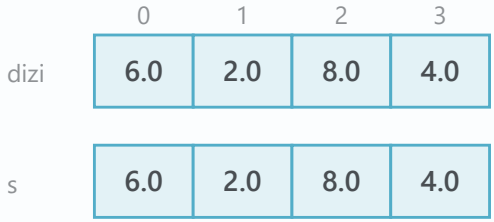
ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
▶      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```



Önce **kopya**: çağıranın dizisinin sırası bozulmasın.

Adım 2 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [6.0, 2.0, 8.0, 4.0]`

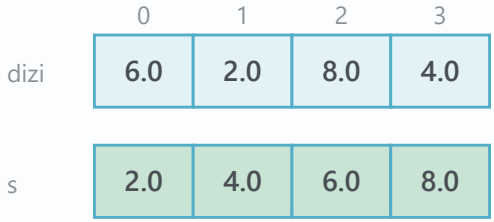
ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
▶      Arrays.sort(s);            // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {      // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                   // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```



Kopya sıralanır; orijinal `dizi` değişmedi.

Adım 3 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [2.0, 4.0, 6.0, 8.0]`

ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
▶      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3
dizi	6.0	2.0	8.0	4.0
s	2.0	4.0	6.0	8.0

boyut = 4.

Adım 4 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [2.0, 4.0, 6.0, 8.0]` `boyut = 4`

ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
▶      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3
dizi	6.0	2.0	8.0	4.0
s	2.0	4.0	6.0	8.0

4 % 2 == 0 → **çift**: ortada tek bir öğe yok, iki öğe var.

Adım 5 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [2.0, 4.0, 6.0, 8.0]` `boyut = 4`

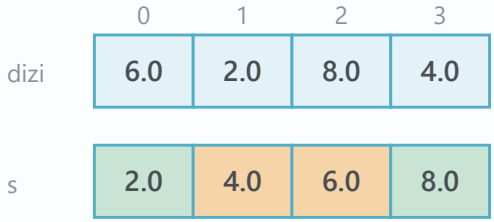
ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
▶          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```



Ortadaki iki indis: o1 = 1, o2 = 2.

Adım 6 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [2.0, 4.0, 6.0, 8.0]` `boyut = 4`

`o1 = 1` `o2 = 2`

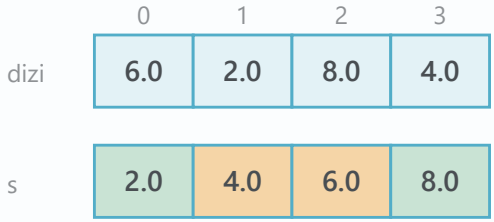
ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
▶          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```



ortanca = (4.0 + 6.0) / 2 = **5.0**.

Adım 7 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [2.0, 4.0, 6.0, 8.0]` `boyut = 4`

`o1 = 1` `o2 = 2` `ortanca = 5.0`

ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
▶      return ortanca;
13 }
```

	0	1	2	3
dizi	6.0	2.0	8.0	4.0
s	2.0	4.0	6.0	8.0

return 5.0.

Adım 8 / 15

DEĞİŞKENLER

`dizi = [6.0, 2.0, 8.0, 4.0]`

`s = [2.0, 4.0, 6.0, 8.0]` `boyut = 4`

`o1 = 1` `o2 = 2` `ortanca = 5.0`

ÇAĞRI YIĞINI

`ortanca({6, 2, 8, 4})`

`main()`

Kod İzleme: `ortanca()`

```
▶ static double ortanca(double[] dizi) {
2     double[] s = dizi.clone(); // kopya
3     Arrays.sort(s);              // O(n log n)
4     int boyut = s.length;
5     double ortanca;
6     if (boyut % 2 == 0) {        // çift
7         int o1 = boyut/2 - 1, o2 = boyut/2;
8         ortanca = (s[o1] + s[o2]) / 2;
9     } else {                     // tek
10        ortanca = s[boyut / 2];
11    }
12    return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0

2. çağrı (tek boyut): `ortanca` 5 öğeli bir diziyle çağrılır.

Adım 9 / 15

DEĞİŞKENLER

`dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`

ÇAĞRI YIĞINI

`ortanca({9, 3, 11, 7, 5})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
▶      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0
s	9.0	3.0	11.0	7.0	5.0

Önce **kopya**: çağıranın dizisinin sırası bozulmasın.

Adım 10 / 15

DEĞİŞKENLER

`dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`

`s = [9.0, 3.0, 11.0, 7.0, 5.0]`

ÇAĞRI YIĞINI

`ortanca({9, 3, 11, 7, 5})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
▶      Arrays.sort(s);            // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {      // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                   // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0
s	3.0	5.0	7.0	9.0	11.0

Kopya sıralanır; orijinal `dizi` değişmedi.

Adım 11 / 15

DEĞİŞKENLER

`dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`

`s = [3.0, 5.0, 7.0, 9.0, 11.0]`

ÇAĞRI YIĞINI

`ortanca({9, 3, 11, 7, 5})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
▶      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0
s	3.0	5.0	7.0	9.0	11.0

boyut = 5.

Adım 12 / 15

DEĞİŞKENLER

`dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`

`s = [3.0, 5.0, 7.0, 9.0, 11.0]`

`boyut = 5`

ÇAĞRI YIĞINI

`ortanca({9, 3, 11, 7, 5})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);             // O(n log n)
4      int boyut = s.length;
5      double ortanca;
▶      if (boyut % 2 == 0) {       // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                    // tek
10         ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0
s	3.0	5.0	7.0	9.0	11.0

5 % 2 = 1 → **tek**: else dalına geçilir.

Adım 13 / 15

DEĞİŞKENLER

- `dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`
- `s = [3.0, 5.0, 7.0, 9.0, 11.0]`
- `boyut = 5`

ÇAĞRI YIĞINI

- `ortanca({9, 3, 11, 7, 5})`
- `main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
▶          ortanca = s[boyut / 2];
11     }
12     return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0
s	3.0	5.0	7.0	9.0	11.0

Tam orta: indis 5 / 2 = 2 → **7.0**.

Adım 14 / 15

DEĞİŞKENLER

`dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`

`s = [3.0, 5.0, 7.0, 9.0, 11.0]`

`boyut = 5` `ortanca = 7.0`

ÇAĞRI YIĞINI

`ortanca({9, 3, 11, 7, 5})`

`main()`

Kod İzleme: `ortanca()`

```
1  static double ortanca(double[] dizi) {
2      double[] s = dizi.clone(); // kopya
3      Arrays.sort(s);              // O(n log n)
4      int boyut = s.length;
5      double ortanca;
6      if (boyut % 2 == 0) {        // çift
7          int o1 = boyut/2 - 1, o2 = boyut/2;
8          ortanca = (s[o1] + s[o2]) / 2;
9      } else {                     // tek
10         ortanca = s[boyut / 2];
11     }
▶      return ortanca;
13 }
```

	0	1	2	3	4
dizi	9.0	3.0	11.0	7.0	5.0
s	3.0	5.0	7.0	9.0	11.0

return 7.0. Sıralama baskın → O(n log n).

Adım 15 / 15

DEĞİŞKENLER

`dizi = [9.0, 3.0, 11.0, 7.0, 5.0]`

`s = [3.0, 5.0, 7.0, 9.0, 11.0]`

`boyut = 5` `ortanca = 7.0`

ÇAĞRI YIĞINI

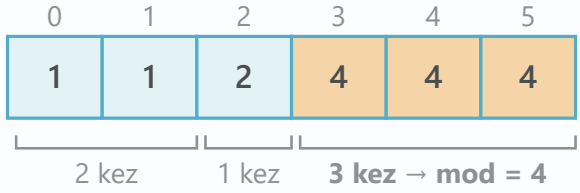
`ortanca({9, 3, 11, 7, 5})`

`main()`

Mod (En Sık Tekrar Eden Değer)

```
static int mod(int[] dizi) { // boş değil
    int[] s = dizi.clone();
    Arrays.sort(s);  // eşitler yan yana
    int enCok = s[0], enCokSayi = 1;
    int mevcutSayi = 1;
    for (int i = 1; i < s.length; i++) {
        if (s[i] == s[i - 1]) mevcutSayi++;
        else mevcutSayi = 1;   // yeni değer
        if (mevcutSayi > enCokSayi) {
            enCokSayi = mevcutSayi;
            enCok = s[i];
        }
    }
    return enCok;
}
```

- Sıralamadan sonra eşit değerler **ardışık** olur; tek geçişte en uzun tekrar dizisi bulunur.
- Eşitlikte ilk (en küçük) değer kalır, çünkü karşılaştırma `>` .
- Maliyet: sıralama O(n log n) + tarama O(n).



Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
▶ static int mod(int[] dizi) { // boş değil
2     int[] s = dizi.clone();
3     Arrays.sort(s);  // eşitler yan yana
4     int enCok = s[0], enCokSayi = 1;
5     int mevcutSayi = 1;
6     for (int i = 1; i < s.length; i++) {
7         if (s[i] == s[i - 1]) mevcutSayi++;
8         else mevcutSayi = 1;   // yeni değer
9         if (mevcutSayi > enCokSayi) {
10            enCokSayi = mevcutSayi;
11            enCok = s[i];
12        }
13    }
14    return enCok;
15 }
```



Girdi: {4, 1, 4, 2, 1, 4}. Eşit değerler dağınık.

Adım 1 / 14

DEĞİŞKENLER

`dizi = [4, 1, 4, 2, 1, 4]`

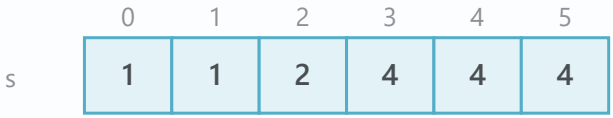
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
▶      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



Kopya sıralanınca eşit değerler **yan yana** gelir; artık tek geçiş yeter.

Adım 2 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]`

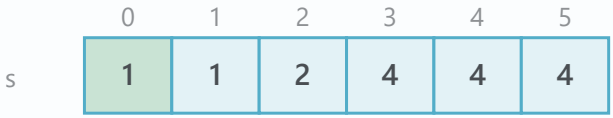
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
▶      int enCok = s[0], enCokSayi = 1;
▶      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



Başlangıç: ilk değer (1) şimdilik en çok görülen, 1 kez.

Adım 3 / 14

DEĞİŞKENLER

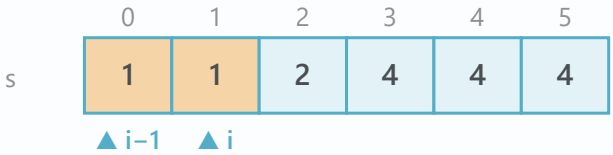
`s = [1, 1, 2, 4, 4, 4]` `enCok = 1` `enCokSayi = 1` `mevcutSayi = 1`

ÇAĞRI YIĞINI

- `mod(dizi)`
- `main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
▶          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



s[1] = 1 == s[0] → aynı değer sürüyor, **mevcutSayi = 2**.

Adım 4 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 1`

`enCok = 1` `enCokSayi = 1`

`mevcutSayi = 2`

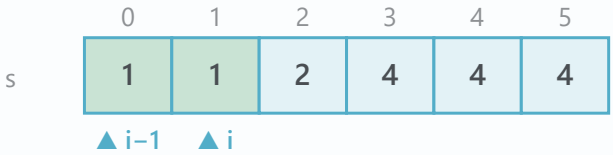
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
▶          if (mevcutSayi > enCokSayi) {
▶              enCokSayi = mevcutSayi;
▶              enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



2 > önceki rekor → **yeni rekor**: enCok = 1, 2 kez.

Adım 5 / 14

DEĞİŞKENLER

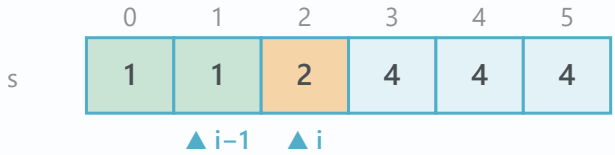
`s = [1, 1, 2, 4, 4, 4]` `i = 1` `enCok = 1` `enCokSayi = 2` `mevcutSayi = 2`

ÇAĞRI YIĞINI

- `mod(dizi)`
- `main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
▶          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



s[2] = 2 ≠ 1 → **yeni değer**, sayaç 1'e döner.

Adım 6 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 2`

`enCok = 1` `enCokSayi = 2`

`mevcutSayi = 1`

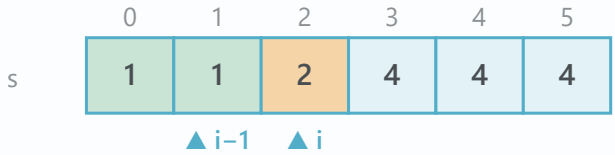
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
▶          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



1 > 2 yanlış → rekor değişmez.

Adım 7 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 2`

`enCok = 1` `enCokSayi = 2`

`mevcutSayi = 1`

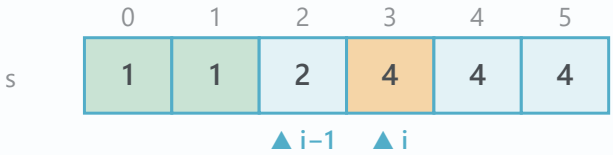
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
▶          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



s[3] = 4 ≠ 2 → **yeni değer**, sayaç 1'e döner.

Adım 8 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 3`

`enCok = 1` `enCokSayi = 2`

`mevcutSayi = 1`

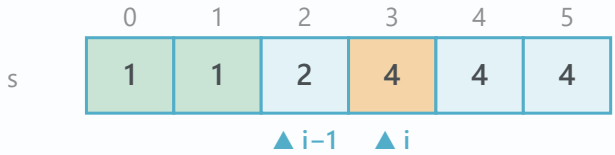
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
▶          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



1 > 2 yanlış → rekor değişmez.

Adım 9 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 3`

`enCok = 1` `enCokSayi = 2`

`mevcutSayi = 1`

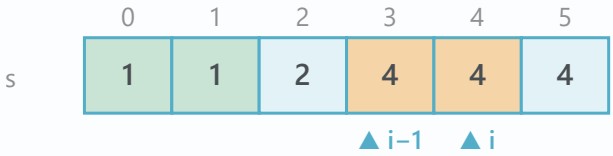
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
▶          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



s[4] = 4 == s[3] → aynı değer sürüyor, **mevcutSayi = 2**.

Adım 10 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 4`

`enCok = 1` `enCokSayi = 2`

`mevcutSayi = 2`

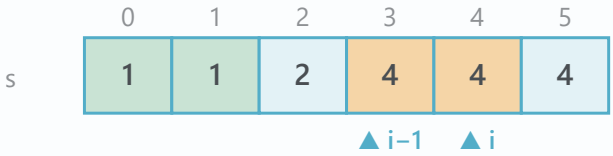
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
▶          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



2 > 2 yanlış → rekor değişmez. (Eşitlikte ilk değer kalır.)

Adım 11 / 14

DEĞİŞKENLER

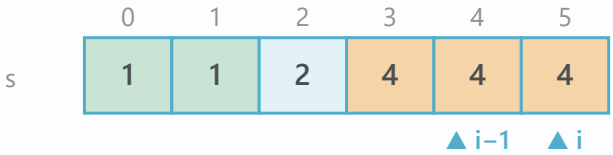
`s = [1, 1, 2, 4, 4, 4]` `i = 4` `enCok = 1` `enCokSayi = 2` `mevcutSayi = 2`

ÇAĞRI YIĞINI

- **`mod(dizi)`**
- `main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
▶          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



s[5] = 4 == s[4] → aynı değer sürüyor, **mevcutSayi = 3**.

Adım 12 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 5`

`enCok = 1` `enCokSayi = 2`

`mevcutSayi = 3`

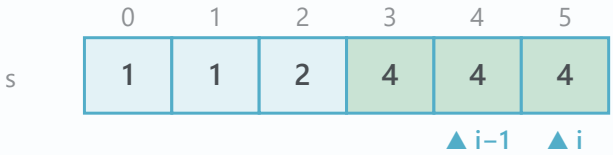
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
▶          if (mevcutSayi > enCokSayi) {
▶              enCokSayi = mevcutSayi;
▶              enCok = s[i];
12         }
13     }
14     return enCok;
15 }
```



3 > önceki rekor → **yeni rekor**: enCok = 4, 3 kez.

Adım 13 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `i = 5`

`enCok = 4` `enCokSayi = 3`

`mevcutSayi = 3`

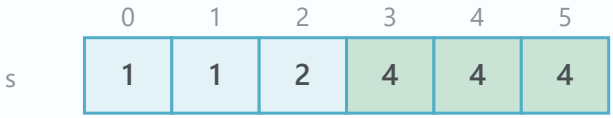
ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Kod İzleme: `mod({4, 1, 4, 2, 1, 4})`

```
1  static int mod(int[] dizi) { // boş değil
2      int[] s = dizi.clone();
3      Arrays.sort(s);  // eşitler yan yana
4      int enCok = s[0], enCokSayi = 1;
5      int mevcutSayi = 1;
6      for (int i = 1; i < s.length; i++) {
7          if (s[i] == s[i - 1]) mevcutSayi++;
8          else mevcutSayi = 1;   // yeni değer
9          if (mevcutSayi > enCokSayi) {
10             enCokSayi = mevcutSayi;
11             enCok = s[i];
12         }
13     }
▶      return enCok;
15 }
```



return 4 (3 kez). Sıralama O(n log n) + tarama O(n).

Adım 14 / 14

DEĞİŞKENLER

`s = [1, 1, 2, 4, 4, 4]` `enCok = 4` `enCokSayi = 3` `mevcutSayi = 3`

ÇAĞRI YIĞINI

`mod(dizi)`

`main()`

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
▶       int baslangic = 0;
3       int bitis = dizi.length - 1;
4       while (baslangic < bitis) {
5           // uçtaki iki öğeyi takas et
6           int gecici = dizi[baslangic];
7           dizi[baslangic] = dizi[bitis];
8           dizi[bitis] = gecici;
9           baslangic++;   // içe doğru ilerle
10          bitis--;
11      }
12 }
```

0	1	2	3	4	5
3	8	1	9	4	6

▲ bas

gecici = —

Sol işaretçi ilk öğede.

Adım 1 / 18

DEĞİŞKENLER

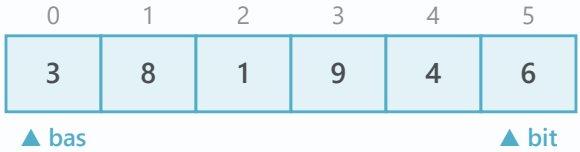
`dizi = [3, 8, 1, 9, 4, 6]`

`baslangic = 0`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
▶      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = —

Sağ işaretçi son öğede (length − 1 = 5).

Adım 2 / 18

DEĞİŞKENLER

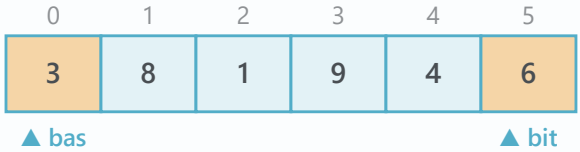
`dizi = [3, 8, 1, 9, 4, 6]`

`baslangic = 0` `bitis = 5`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n − 1 − i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
▶      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = —

0 < 5 → takas edilecek bir çift daha var.

Adım 3 / 18

DEĞİŞKENLER

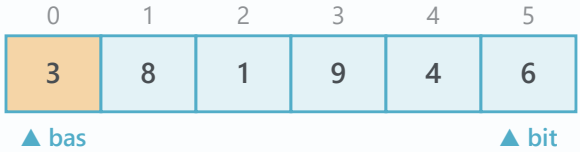
`dizi = [3, 8, 1, 9, 4, 6]`

`baslangic = 0` `bitis = 5`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
▶          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 3

gecici = dizi[0] = 3: soldaki değer kaybolmasın diye saklanır.

Adım 4 / 18

DEĞİŞKENLER

`dizi = [3, 8, 1, 9, 4, 6]`

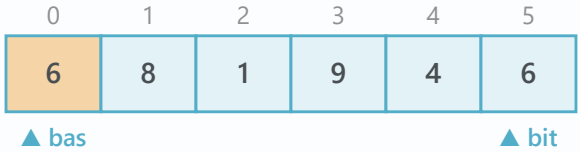
`baslangic = 0` `bitis = 5`

`gecici = 3`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n − 1 − i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
▶          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 3

dizi[0] = dizi[5] → **6**. (Eski 3 artık yalnızca gecici'de.)

Adım 5 / 18

DEĞİŞKENLER

`dizi = [6, 8, 1, 9, 4, 6]`

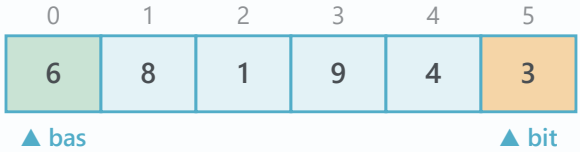
`baslangic = 0` `bitis = 5`

`gecici = 3`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
▶          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 3

dizi[5] = gecici → **3**. Çift yer değiştirdi.

Adım 6 / 18

DEĞİŞKENLER

`dizi = [6, 8, 1, 9, 4, 3]`

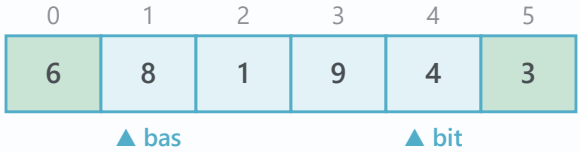
`baslangic = 0` `bitis = 5`

`gecici = 3`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n − 1 − i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
▶          baslangic++;   // içe doğru ilerle
▶          bitis--;
11     }
12 }
```



gecici = 3

İşaretçiler içe doğru: baslangic = 1, bitis = 4.

Adım 7 / 18

DEĞİŞKENLER

`dizi = [6, 8, 1, 9, 4, 3]`

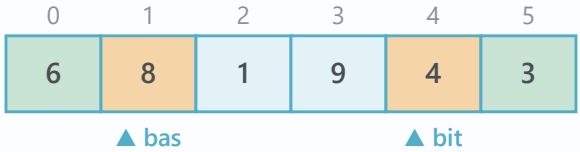
`baslangic = 1` `bitis = 4`

`gecici = 3`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n − 1 − i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
▶      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 3

1 < 4 → takas edilecek bir çift daha var.

Adım 8 / 18

DEĞİŞKENLER

`dizi = [6, 8, 1, 9, 4, 3]`

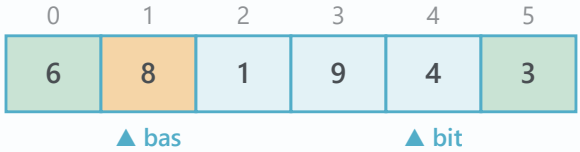
`baslangic = 1` `bitis = 4`

`gecici = 3`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
▶          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 8

gecici = dizi[1] = 8: soldaki değer kaybolmasın diye saklanır.

Adım 9 / 18

DEĞİŞKENLER

`dizi = [6, 8, 1, 9, 4, 3]`

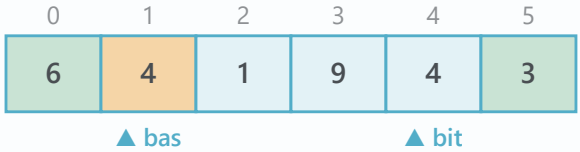
`baslangic = 1` `bitis = 4`

`gecici = 8`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n − 1 − i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
▶          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 8

dizi[1] = dizi[4] → **4**. (Eski 8 artık yalnızca gecici'de.)

Adım 10 / 18

DEĞİŞKENLER

`dizi = [6, 4, 1, 9, 4, 3]`

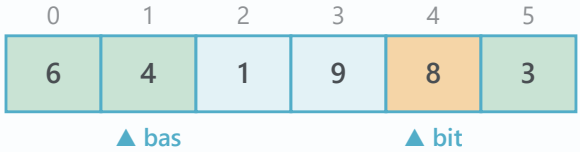
`baslangic = 1` `bitis = 4`

`gecici = 8`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
▶          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 8

dizi[4] = gecici → **8**. Çift yer değiştirdi.

Adım 11 / 18

DEĞİŞKENLER

`dizi = [6, 4, 1, 9, 8, 3]`

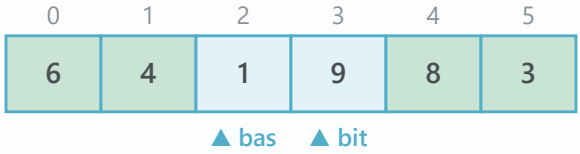
`baslangic = 1` `bitis = 4`

`gecici = 8`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
▶          baslangic++;   // içe doğru ilerle
▶          bitis--;
11     }
12 }
```



gecici = 8

İşaretçiler içe doğru: baslangic = 2, bitis = 3.

Adım 12 / 18

DEĞİŞKENLER

`dizi = [6, 4, 1, 9, 8, 3]`

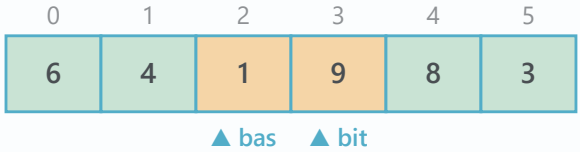
`baslangic = 2` `bitis = 3`

`gecici = 8`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
▶      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 8

2 < 3 → takas edilecek bir çift daha var.

Adım 13 / 18

DEĞİŞKENLER

`dizi = [6, 4, 1, 9, 8, 3]`

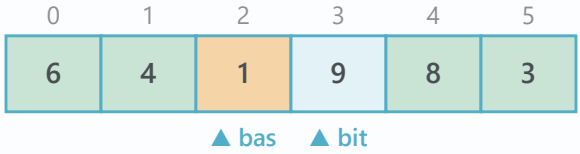
`baslangic = 2` `bitis = 3`

`gecici = 8`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 1

gecici = dizi[2] = 1: soldaki değer kaybolmasın diye saklanır.

Adım 14 / 18

DEĞİŞKENLER

`dizi = [6, 4, 1, 9, 8, 3]`

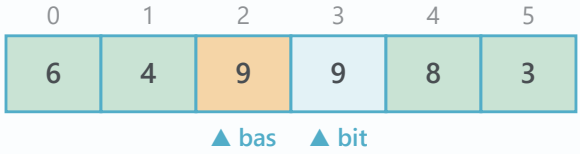
`baslangic = 2` `bitis = 3`

`gecici = 1`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
▶          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 1

dizi[2] = dizi[3] → **9**. (Eski 1 artık yalnızca gecici'de.)

Adım 15 / 18

DEĞİŞKENLER

`dizi = [6, 4, 9, 9, 8, 3]`

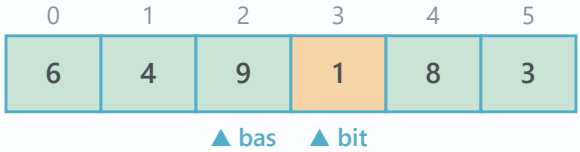
`baslangic = 2` `bitis = 3`

`gecici = 1`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
▶          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 1

dizi[3] = gecici → **1**. Çift yer değiştirdi.

Adım 16 / 18

DEĞİŞKENLER

`dizi = [6, 4, 9, 1, 8, 3]`

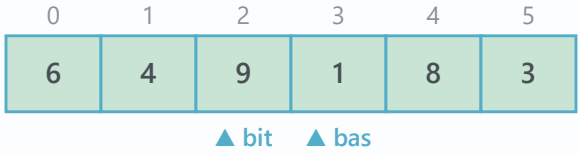
`baslangic = 2` `bitis = 3`

`gecici = 1`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n − 1 − i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
4      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
▶          baslangic++;   // içe doğru ilerle
▶          bitis--;
11     }
12 }
```



gecici = 1

İşaretçiler içe doğru: baslangic = 3, bitis = 2.

Adım 17 / 18

DEĞİŞKENLER

`dizi = [6, 4, 9, 1, 8, 3]`

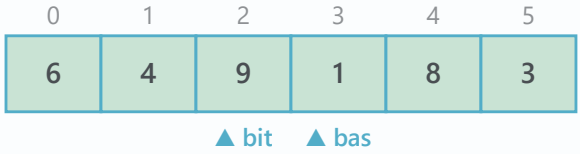
`baslangic = 3` `bitis = 2`

`gecici = 1`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Diziyi Yerinde Tersine Çevirme

```
1  static void tersineCevir(int[] dizi) {
2      int baslangic = 0;
3      int bitis = dizi.length - 1;
▶      while (baslangic < bitis) {
5          // uçtaki iki öğeyi takas et
6          int gecici = dizi[baslangic];
7          dizi[baslangic] = dizi[bitis];
8          dizi[bitis] = gecici;
9          baslangic++;   // içe doğru ilerle
10         bitis--;
11     }
12 }
```



gecici = 1

3 < 2 yanlış → işaretçiler ortada birbirini geçti, döngü biter. ⌊n/2⌋ = 3 takas → O(n) zaman, O(1) ek bellek.

Adım 18 / 18

DEĞİŞKENLER

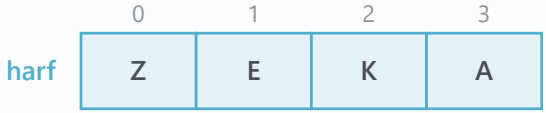
`dizi = [6, 4, 9, 1, 8, 3]`

`baslangic = 3` `bitis = 2`

`gecici = 1`

Alternatif: yeni diziye sondan başa kopyalamak (`yeni[i] = dizi[n - 1 - i]`) da O(n) zamandır, ama O(n) ek bellek ister. Öğe sayısı tekse ortadaki öğe yerinde kalır.

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



`toCharArray()` metnin harflerini yeni bir **char[]** dizisine kopyalar. `String` değiştirilemez; dizi ise yerinde değiştirilebilir.

Adım 1 / 10

```
1 static String sifrele(String mesaj, int k) {
▶     char[] harf = mesaj.toCharArray();
3     for (int i = 0; i < harf.length; i++) {
4         int sira = harf[i] - 'A';        // 0..25 (yalnız
  A–Z)
5         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`

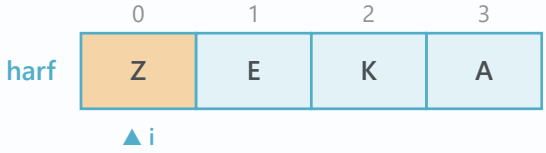
`harf = [Z, E, K, A]`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`

`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'Z' − 'A' = 25

`i = 0` : `'Z' − 'A'` harfin alfabedeki sırasını verir: **25** (char değerleri tam sayıdır).

Adım 2 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
▶     for (int i = 0; i < harf.length; i++) {
▶         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
5         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`

`harf = [Z, E, K, A]` `i = 0`

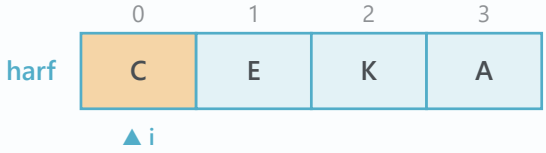
`sira = 25`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`

`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sıra = 'Z' − 'A' = 25 → (25 + 3) % 26 = 2 → 'C'

`harf[0] = 'C'` : sıra 3 kaydırılır, yeniden harfe çevrilir. Sona taşan sıra `% 26` ile başa sarar (Z → C).

Adım 3 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
3     for (int i = 0; i < harf.length; i++) {
4         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
▶         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

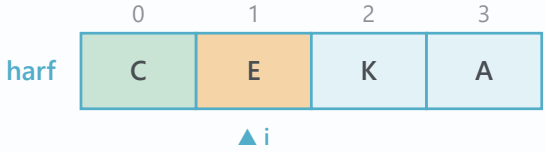
DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`
`harf = [C, E, K, A]` `i = 0`
`sira = 25`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`
`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'E' − 'A' = 4

`i = 1` : `'E' - 'A'` harfin alfabedeki sırasını verir: **4** (char değerleri tam sayıdır).

Adım 4 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
▶     for (int i = 0; i < harf.length; i++) {
▶         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
5         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`

`harf = [C, E, K, A]` `i = 1`

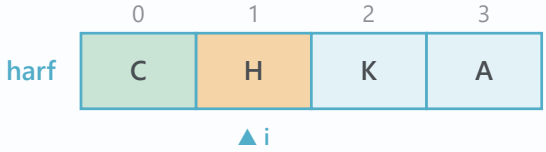
`sira = 4`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`

`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'E' − 'A' = 4 → (4 + 3) % 26 = 7 → 'H'

`harf[1] = 'H'` : sıra 3 kaydırılır, yeniden harfe çevrilir.

Adım 5 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
3     for (int i = 0; i < harf.length; i++) {
4         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
▶           harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

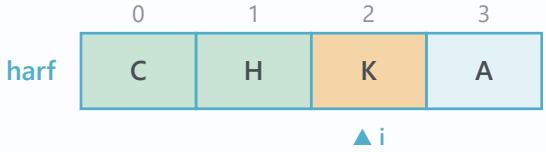
DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`
`harf = [C, H, K, A]` `i = 1`
`sira = 4`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`
`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'K' − 'A' = 10

`i = 2` : `'K' − 'A'` harfin alfabedeki sırasını verir: **10** (char değerleri tam sayıdır).

Adım 6 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
▶     for (int i = 0; i < harf.length; i++) {
▶         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
5         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`

`harf = [C, H, K, A]` `i = 2`

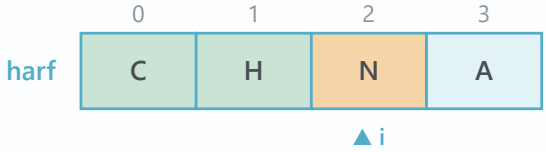
`sira = 10`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`

`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'K' − 'A' = 10 → (10 + 3) % 26 = 13 → 'N'

`harf[2] = 'N'` : sıra 3 kaydırılır, yeniden harfe çevrilir.

Adım 7 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
3     for (int i = 0; i < harf.length; i++) {
4         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
▶           harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

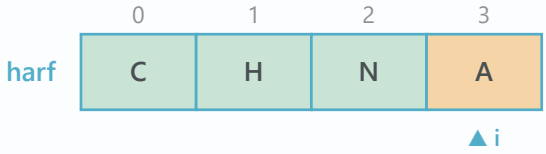
DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`
`harf = [C, H, N, A]` `i = 2`
`sira = 10`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`
`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'A' − 'A' = 0

`i = 3` : `'A' - 'A'` harfin alfabedeki sırasını verir: **0** (char değerleri tam sayıdır).

Adım 8 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
▶     for (int i = 0; i < harf.length; i++) {
▶         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
5         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`

`harf = [C, H, N, A]` `i = 3`

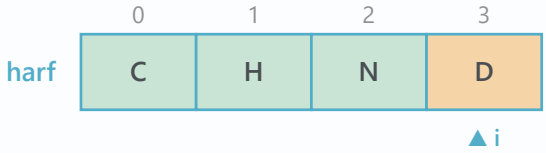
`sira = 0`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`

`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



sira = 'A' − 'A' = 0 → (0 + 3) % 26 = 3 → 'D'

`harf[3] = 'D'` : sıra 3 kaydırılır, yeniden harfe çevrilir.

Adım 9 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
3     for (int i = 0; i < harf.length; i++) {
4         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
▶         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
7     return new String(harf);
8 }
```

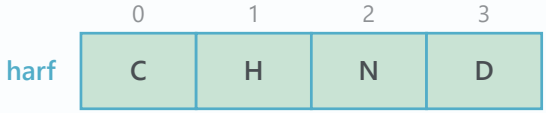
DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`
`harf = [C, H, N, D]` `i = 3`
`sira = 0`

ÇAĞRI YIĞINI

`sifrele("ZEKA", 3)`
`main()`

Uygulama: Sezar Şifresi `sifrele("ZEKA", 3)`



`new String(harf)` → **"CHND"**. Çözmek için aynı metot `k = 26 − 3 = 23` ile çağrılır. Her harf bir kez işlenir: **O(n)**.

Adım 10 / 10

```
1 static String sifrele(String mesaj, int k) {
2     char[] harf = mesaj.toCharArray();
3     for (int i = 0; i < harf.length; i++) {
4         int sira = harf[i] - 'A';        // 0..25 (yalnız A–Z)
5         harf[i] = (char) ('A' + (sira + k) % 26);
6     }
▶     return new String(harf);
8 }
```

DEĞİŞKENLER

`mesaj = "ZEKA"` `k = 3`

`harf = [C, H, N, D]`

ÇAĞRI YIĞINI

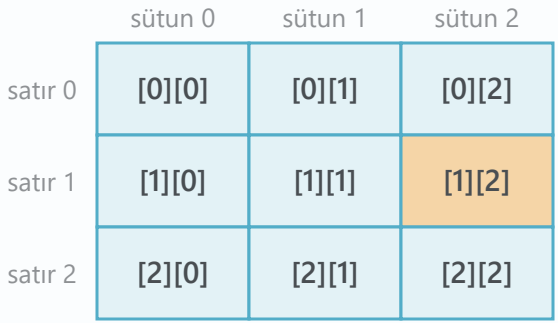
`sifrele("ZEKA", 3)`

`main()`

İki Boyutlu Diziler (Matrisler)

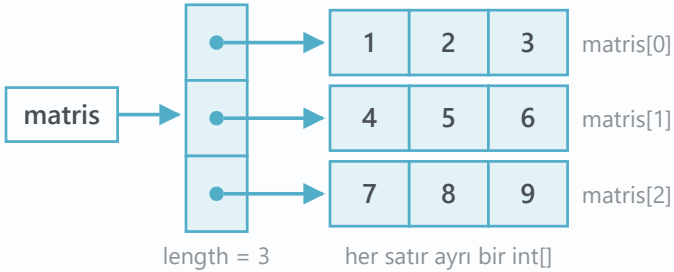
- Satır ve sütunlardan oluşan tablo benzeri bir yapıdır.
- Satranç tahtası, resim (piksel ızgarası), tablo gibi iki boyutlu veriler için kullanışlıdır.
- Bir öğeye iki indisle erişilir: `matris[satır][sütun]`.

```
int[][] matris = new int[3][3];
matris[1][2] = 7;   // 2. satır, 3. sütun
```



Java'da Matris = Dizilerin Dizisi

```
int[][] matris = {
    {1, 2, 3},
    {4, 5, 6},
    {7, 8, 9}
};
matris.length      // 3 (satır sayısı)
matris[0].length   // 3 (sütun sayısı)
```



- İlklendiricide boyut yazılmaz: satır sayısı dış listedeki, sütun sayısı iç listelerdeki öğe sayısından çıkar.
- Her satır ayrı bir dizi olduğundan satırların boyu farklı olabilir (**düzensiz dizi**, jagged array): `int[][] ucgen = new int[3][]; ucgen[0] = new int[1];` ...

Düzensiz Dizi: Pascal Üçgeni



`new int[n][]` : yalnızca 4 **referanslık** dış dizi oluşur; satır dizileri henüz yok (`null`).

Adım 1 / 12

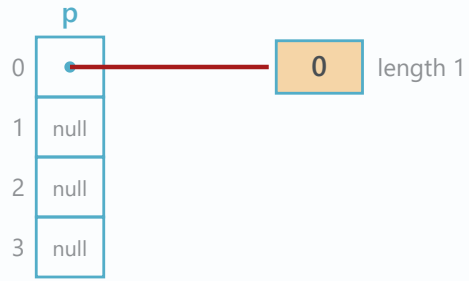
```
▶ int n = 4;
▶ int[][] p = new int[n][];          // satırlar yok
  for (int i = 0; i < n; i++) {
      p[i] = new int[i + 1];         // i + 1 uzunluk
      p[i][0] = p[i][i] = 1;
      for (int j = 1; j < i; j++)
          p[i][j] = p[i-1][j-1] + p[i-1][j];
  }
```

DEĞİŞKENLER

n = 4

p = [null, null, null, null]

Düzensiz Dizi: Pascal Üçgeni



`p[0] = new int[1]` : 0. satır **1** uzunlukta ayrı bir dizi; hücreler 0 ile başlar.

Adım 2 / 12

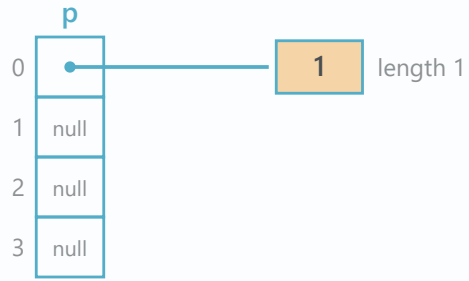
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
▶     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 0`

`p = [[0], null, null, null]`

Düzensiz Dizi: Pascal Üçgeni



`p[0][0] = 1` : tek hücreli satırda baş ve son aynı hücre.

Adım 3 / 12

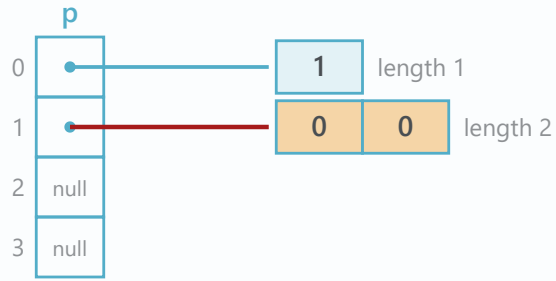
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
▶     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 0`

`p = [[1], null, null, null]`

Düzensiz Dizi: Pascal Üçgeni



`p[1] = new int[2]` : 1. satır **2** uzunlukta ayrı bir dizi; satırların boyu farklı → düzensiz (jagged) dizi.

Adım 4 / 12

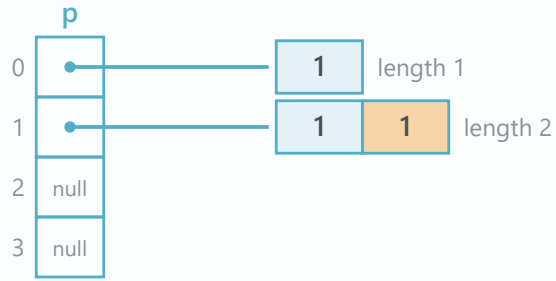
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
▶     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 1`

`p = [[1], [0, 0], null, null]`

Düzensiz Dizi: Pascal Üçgeni



`p[1][0] = p[1][1] = 1` : satırın iki ucu 1 (zincirleme atama sağdan sola).

Adım 5 / 12

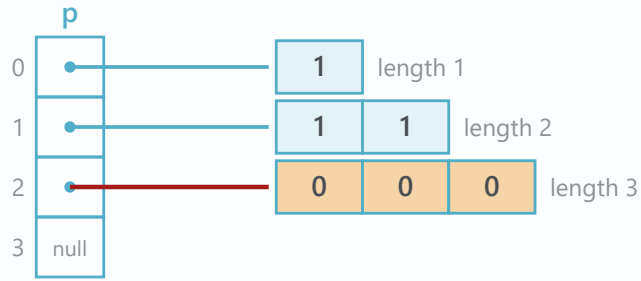
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
▶     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 1`

`p = [[1], [1, 1], null, null]`

Düzensiz Dizi: Pascal Üçgeni



`p[2] = new int[3]` : 3 hücrelik yeni satır.

Adım 6 / 12

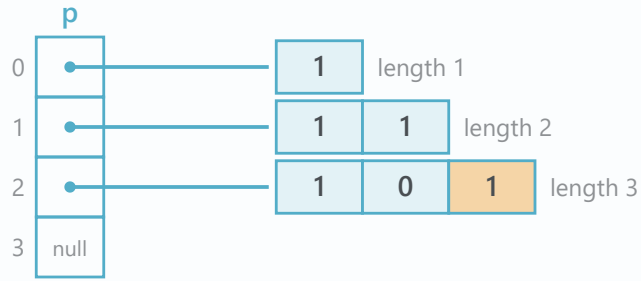
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
▶     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 2`

`p = [[1], [1, 1], [0, 0, 0], null]`

Düzensiz Dizi: Pascal Üçgeni



`p[2][0] = p[2][2] = 1` : satırın iki ucu 1 (zincirleme atama sağdan sola).

Adım 7 / 12

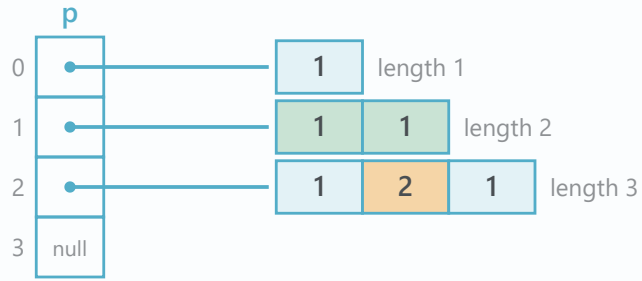
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
▶     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 2`

`p = [[1], [1, 1], [1, 0, 1], null]`

Düzensiz Dizi: Pascal Üçgeni



`p[2][1] = p[1][0] + p[1][1]` = 1 + 1 = **2**: üstteki iki komşunun toplamı.

Adım 8 / 12

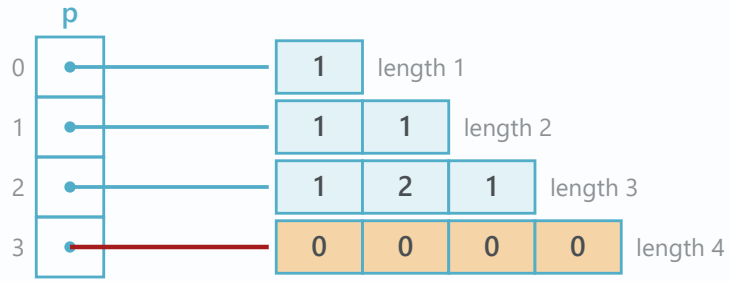
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
▶         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 2` `j = 1`

`p = [[1], [1, 1], [1, 2, 1], null]`

Düzensiz Dizi: Pascal Üçgeni



`p[3] = new int[4]` : 4 hücrelik yeni satır.

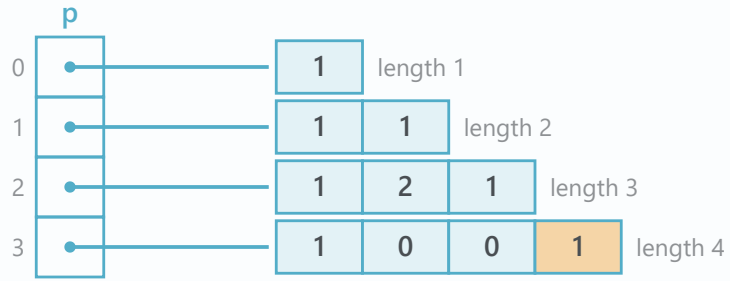

```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
▶     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 3`

`p = [[1], [1, 1], [1, 2, 1], [0, 0, 0, 0]]`

Düzensiz Dizi: Pascal Üçgeni



`p[3][0] = p[3][3] = 1` : satırın iki ucu 1 (zincirleme atama sağdan sola).

Adım 10 / 12

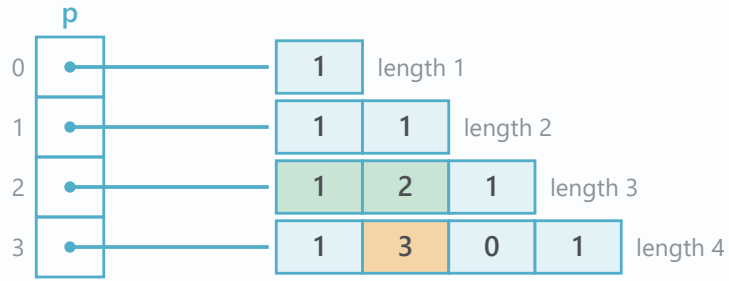
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
▶     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
7         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 3`

`p = [[1], [1, 1], [1, 2, 1], [1, 0, 0, 1]]`

Düzensiz Dizi: Pascal Üçgeni



`p[3][1] = p[2][0] + p[2][1]` = 1 + 2 = **3**: üstteki iki komşunun toplamı.

Adım 11 / 12

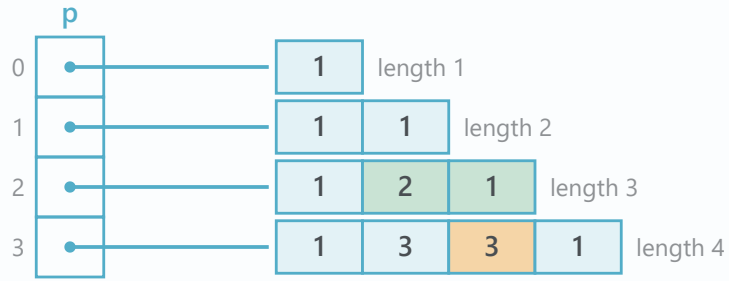
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
▶         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 3` `j = 1`

`p = [[1], [1, 1], [1, 2, 1], [1, 3, 0, 1]]`

Düzensiz Dizi: Pascal Üçgeni



`p[3][2] = p[2][1] + p[2][2]` = 2 + 1 = **3**: üstteki iki komşunun toplamı. Pascal üçgeni tamam: toplam 1+2+3+4 = 10 hücre (4×4 = 16 değil).

Adım 12 / 12

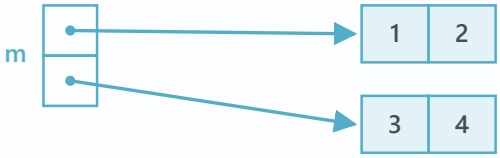
```
1 int n = 4;
2 int[][] p = new int[n][];          // satırlar yok
3 for (int i = 0; i < n; i++) {
4     p[i] = new int[i + 1];         // i + 1 uzunluk
5     p[i][0] = p[i][i] = 1;
6     for (int j = 1; j < i; j++)
▶         p[i][j] = p[i-1][j-1] + p[i-1][j];
8 }
```

DEĞİŞKENLER

`n = 4` `i = 3` `j = 2`

`p = [[1], [1, 1], [1, 2, 1], [1, 3, 3, 1]]`

2B Dizide Sığ ve Derin Kopya



2×2 matris: dış dizi `m` iki satır dizisini gösterir (dizilerin dizisi).

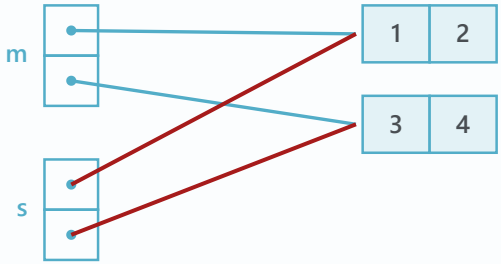
Adım 1 / 8

```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
6 for (int i = 0; i < 2; i++)
7     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

`m = [[1, 2], [3, 4]]`

2B Dizide Sığ ve Derin Kopya



`m.clone()` yalnızca **dış** diziyi kopyalar: `s` yeni bir dış dizi, ama içindeki referanslar **aynı** satırları gösterir → **sığ kopya** (shallow copy).

Adım 2 / 8

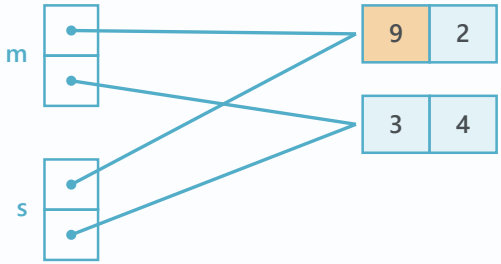
```
1 int[][] m = { {1, 2}, {3, 4} };
▶ int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
6 for (int i = 0; i < 2; i++)
7     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

`m = [[1, 2], [3, 4]]`

`s = [[1, 2], [3, 4]]`

2B Dizide Sığ ve Derin Kopya



`s[0][0] = 9` ortak satırı değiştirir: `m[0][0]` de **9** olur. Sık yapılan hata!

Adım 3 / 8

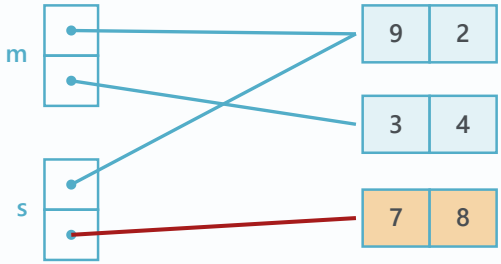
```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
▶ s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
6 for (int i = 0; i < 2; i++)
7     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

`m = [[9, 2], [3, 4]]`

`s = [[9, 2], [3, 4]]`

2B Dizide Sığ ve Derin Kopya



`s[1] = new int[] {7, 8}` ise `s` 'nin kendi dış dizisindeki referansı değiştirir; `m[1]` hâlâ {3, 4}.

Adım 4 / 8

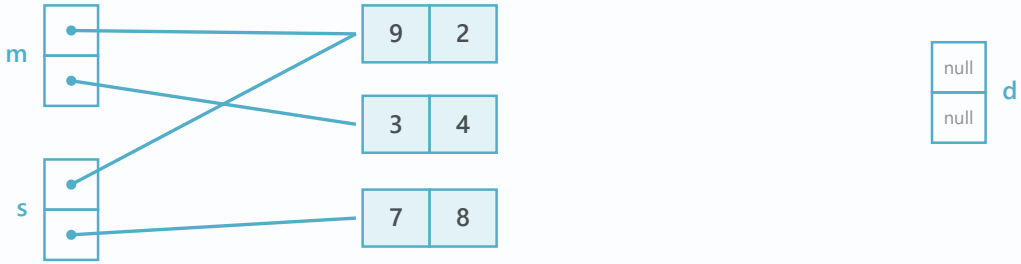
```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
▶ s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
6 for (int i = 0; i < 2; i++)
7     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

`m = [[9, 2], [3, 4]]`

`s = [[9, 2], [7, 8]]`

2B Dizide Sığ ve Derin Kopya



Derin kopya (deep copy) için önce `d` dış dizisi oluşturulur; satırları henüz yok.

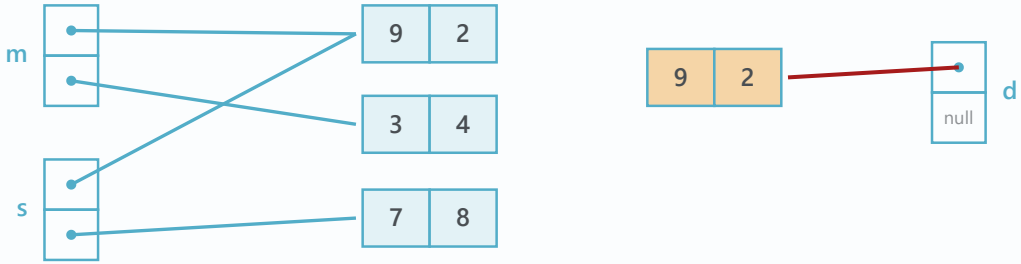
Adım 5 / 8

```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
▶ int[][] d = new int[2][];       // derin kopya:
6 for (int i = 0; i < 2; i++)
7     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

- `m = [[9, 2], [3, 4]]`
- `s = [[9, 2], [7, 8]]`
- `d = [null, null]`

2B Dizide Sığ ve Derin Kopya



`d[0] = m[0].clone()` : 0. satırın **öğeleri** yeni bir diziye kopyalanır.

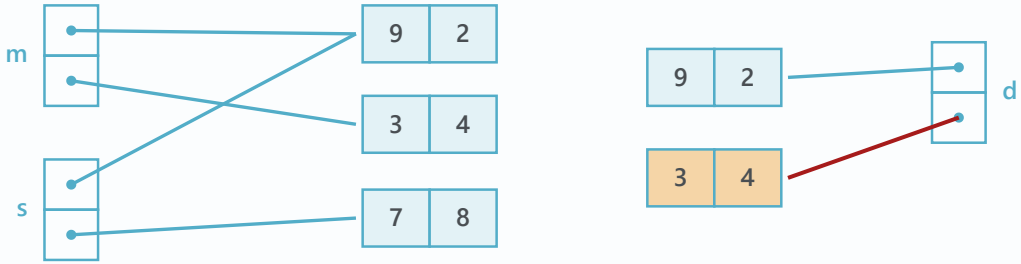
Adım 6 / 8

```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
▶ for (int i = 0; i < 2; i++)
▶     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

- `m = [[9, 2], [3, 4]]`
- `s = [[9, 2], [7, 8]]`
- `d = [[9, 2], null]`
- `i = 0`

2B Dizide Sığ ve Derin Kopya



`d[1] = m[1].clone()` : 1. satırın **öğeleri** yeni bir diziye kopyalanır.

Adım 7 / 8

```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
▶ for (int i = 0; i < 2; i++)
▶     d[i] = m[i].clone();        // satırlar da
8 d[0][1] = 5;                    // m etkilenmez
```

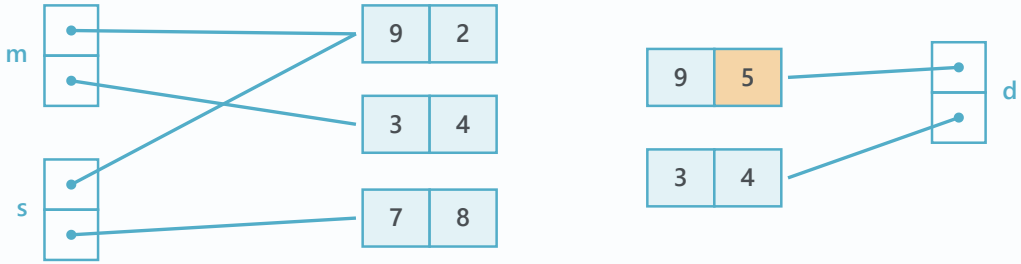
DEĞİŞKENLER

`m = [[9, 2], [3, 4]]`

`s = [[9, 2], [7, 8]]`

`d = [[9, 2], [3, 4]]` `i = 1`

2B Dizide Sığ ve Derin Kopya



`d[0][1] = 5` yalnızca kopyayı değiştirir; `m[0][1]` hâlâ 2. Derin kopya O(satır × sütun) sürer.

Adım 8 / 8

```
1 int[][] m = { {1, 2}, {3, 4} };
2 int[][] s = m.clone();          // sığ kopya
3 s[0][0] = 9;                    // m[0][0] de 9!
4 s[1] = new int[] {7, 8};        // yalnız s değişir
5 int[][] d = new int[2][];       // derin kopya:
6 for (int i = 0; i < 2; i++)
7     d[i] = m[i].clone();        // satırlar da
▶ d[0][1] = 5;                    // m etkilenmez
```

DEĞİŞKENLER

- `m = [[9, 2], [3, 4]]`
- `s = [[9, 2], [7, 8]]`
- `d = [[9, 5], [3, 4]]`

Örnek: Matrisi Doldur ve Yazdır

```java
public class CokBoyutluDiziOrnegi {
  public static void main(String[] a) {
    int[][] matris = new int[3][3];
    // ilk değer atama: 1…9
    for (int i = 0; i < 3; i++)      // satır
      for (int j = 0; j < 3; j++)    // sütun
        matris[i][j] = i * 3 + j + 1;

    System.out.println("Matrisin İçeriği:");
    for (int i = 0; i < 3; i++) {
      for (int j = 0; j < 3; j++)
        System.out.print(matris[i][j] + " ");
      System.out.println();   // yeni satır
    }
  }
}
```

```
Matrisin İçeriği:
1 2 3
4 5 6
7 8 9
```

İç içe döngü

Dış döngü satırları, iç döngü sütunları gezer. 3 × 3 = 9 atama.

`i * 3 + j + 1`

Sabit 3 yerine `matris.length` ve `matris[i].length` yazmak, her boyutta çalışan kod verir.

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	0	0	0
i=1	0	0	0
i=2	0	0	0

3 satırlık dış dizi + 3 ayrı satır dizisi oluşur; tüm hücreler 0.

Adım 1 / 11

DEĞİŞKENLER

`matris = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	0	0
i=1	0	0	0
i=2	0	0	0

i = 0, j = 0 → **matris[0][0] = 0·3 + 0 + 1 = 1**. Dış döngü satırı, iç döngü sütunu seçer.

Adım 2 / 11

DEĞİŞKENLER

`matris = [[1, 0, 0], [0, 0, 0], [0, 0, 0]]`

`i = 0` `j = 0`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	0
i=1	0	0	0
i=2	0	0	0

i = 0, j = 1 → **matris[0][1] = 0·3 + 1 + 1 = 2**.

Adım 3 / 11

DEĞİŞKENLER

`matris = [[1, 2, 0], [0, 0, 0], [0, 0, 0]]`

`i = 0` `j = 1`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	0	0	0
i=2	0	0	0

i = 0, j = 2 → **matris[0][2] = 0·3 + 2 + 1 = 3**.

Adım 4 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [0, 0, 0], [0, 0, 0]]`

`i = 0` `j = 2`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	0	0
i=2	0	0	0

i = 1, j = 0 → **matris[1][0] = 1·3 + 0 + 1 = 4**. İç döngü bitti, dış döngü sonraki satıra geçti; j yeniden 0.

Adım 5 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 0, 0], [0, 0, 0]]`

`i = 1` `j = 0`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	0
i=2	0	0	0

i = 1, j = 1 → **matris[1][1] = 1·3 + 1 + 1 = 5**.

Adım 6 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 0], [0, 0, 0]]`

`i = 1` `j = 1`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	0	0	0

i = 1, j = 2 → **matris[1][2] = 1·3 + 2 + 1 = 6**.

Adım 7 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [0, 0, 0]]`

`i = 1` `j = 2`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	0	0

i = 2, j = 0 → **matris[2][0] = 2·3 + 0 + 1 = 7**. İç döngü bitti, dış döngü sonraki satıra geçti; j yeniden 0.

Adım 8 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 0, 0]]`

`i = 2` `j = 0`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	0

i = 2, j = 1 → **matris[2][1] = 2·3 + 1 + 1 = 8**.

Adım 9 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 0]]`

`i = 2` `j = 1`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
▶      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

i = 2, j = 2 → **matris[2][2] = 2·3 + 2 + 1 = 9**.

Adım 10 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 2` `j = 2`

Kod İzleme: Matrisi Doldur

```
1  int[][] matris = new int[3][3];
▶  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

i = 3 → dış döngü biter. İç gövde 3 × 3 = **9 kez** çalıştı.

Adım 11 / 11

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 3`

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
▶ System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Başlık yazılır; matris artık dolu.

Adım 1 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

ÇIKTI

```
Matrisin İçeriği:
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Satır 0: j = 0'dan başlar → 1.

Adım 2 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 0` `j = 0`

ÇIKTI

```
Matrisin İçeriği:
1
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

matris[0][1] = 2 aynı satıra yazılır (`print`).

Adım 3 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 0` `j = 1`

ÇIKTI

```
Matrisin İçeriği:
1 2
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

matris[0][2] = 3 aynı satıra yazılır (`print`).

Adım 4 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 0` `j = 2`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
▶    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Satır 0 bitti → `println()` alt satıra geçer.

Adım 5 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 0`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Satır 1: j = 0'dan başlar → 4.

Adım 6 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 1` `j = 0`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

matris[1][1] = 5 aynı satıra yazılır (`print`).

Adım 7 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 1` `j = 1`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4 5
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

matris[1][2] = 6 aynı satıra yazılır (`print`).

Adım 8 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 1` `j = 2`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4 5 6
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
▶    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Satır 1 bitti → `println()` alt satıra geçer.

Adım 9 / 13

DEĞİŞKENLER

`matris` = `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i` = **1**

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4 5 6
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Satır 2: j = 0'dan başlar → 7.

Adım 10 / 13

DEĞİŞKENLER

`matris` = `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 2` `j = 0`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4 5 6
7
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

matris[2][1] = 8 aynı satıra yazılır (`print`).

Adım 11 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 2` `j = 1`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4 5 6
7 8
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
▶      System.out.print(matris[i][j] + " ");
9    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

matris[2][2] = 9 aynı satıra yazılır (`print`).

Adım 12 / 13

DEĞİŞKENLER

`matris = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i = 2` `j = 2`

ÇIKTI

```
Matrisin İçeriği:
1 2 3
4 5 6
7 8 9
```

Kod İzleme: Matrisi Yazdır (devam)

```
1  int[][] matris = new int[3][3];
2  for (int i = 0; i < 3; i++)      // satır
3    for (int j = 0; j < 3; j++)    // sütun
4      matris[i][j] = i * 3 + j + 1;
5  System.out.println("Matrisin İçeriği:");
6  for (int i = 0; i < 3; i++) {
7    for (int j = 0; j < 3; j++)
8      System.out.print(matris[i][j] + " ");
▶    System.out.println();   // yeni satır
10 }
```

	j=0	j=1	j=2
i=0	1	2	3
i=1	4	5	6
i=2	7	8	9

Satır 2 bitti → `println()` alt satıra geçer. Tüm matris yazıldı.

Adım 13 / 13

DEĞİŞKENLER

`matris` = `[[1, 2, 3], [4, 5, 6], [7, 8, 9]]`

`i` = **2**

ÇIKTI

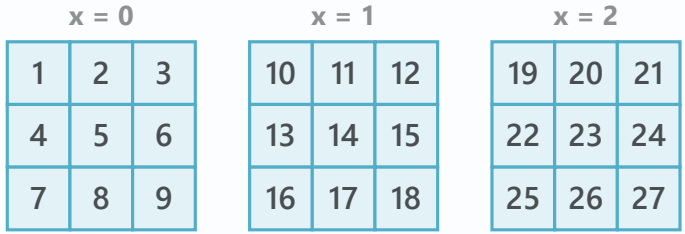
```
Matrisin İçeriği:
1 2 3
4 5 6
7 8 9
```

Üç Boyutlu Diziler

- Java'da üç ya da daha çok boyutlu diziler kullanılabilir (ör. zaman × satır × sütun, RGB katmanları).
- Her boyut bir iç içe döngü demektir: 3 × 3 × 3 dizide 27 öğe.

```
int[][][] ucBoyutluDizi =
        new int[3][3][3];
for (int x = 0; x < 3; x++)
  for (int y = 0; y < 3; y++)
    for (int z = 0; z < 3; z++)
      ucBoyutluDizi[x][y][z] =
          x * 9 + y * 3 + z + 1;

// yazdırma: aynı üç döngü;
// z döngüsünden sonra println (y),
// y döngüsünden sonra println (x)
```



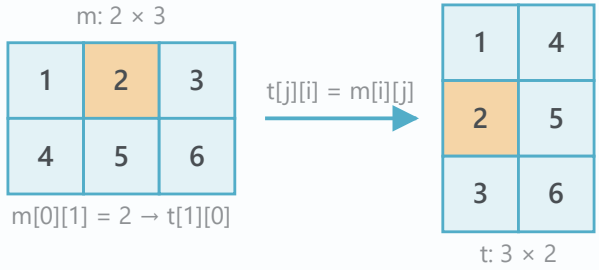
her katman bir 3×3 matris · satır y, sütun z

Değişken adında Türkçe karakter (`üçBoyutluDizi`) Java'da geçerlidir, ancak taşınabilirlik için ASCII adlar tercih edilir.

Matris Transpozu

Satırlar sütuna, sütunlar satıra dönüşür: m×n matrisin transpozu n×m'dir.

```
static int[][] transpoz(int[][] m) {
    int satir = m.length;
    int sutun = m[0].length;
    int[][] t = new int[sutun][satir];
    for (int i = 0; i < satir; i++)
        for (int j = 0; j < sutun; j++)
            t[j][i] = m[i][j];
    return t;
}
static void yazdirMatris(int[][] m) {
    for (int[] s : m) {
        for (int x : s)
            System.out.print(x + " ");
        System.out.println();
    }
}
```



Maliyet: her öğe bir kez kopyalanır → O(m·n).

Yardımcı metotlar `static` olmalı: `main` statiktir ve nesne oluşturmadan yalnızca statik metotları çağırabilir.

Kod İzleme: `transpoz(m)`

m: 2 × 3

1	2	3
4	5	6

m: 2 satır, 3 sütun.

Adım 1 / 11

```
▶ static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
7             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`

m: 2 × 3

1	2	3
4	5	6

`m.length` = **satır** sayısı (dış dizinin boyu).

Adım 2 / 11

```
1 static int[][] transpoz(int[][] m) {
▶       int satir = m.length;
3       int sutun = m[0].length;
4       int[][] t = new int[sutun][satir];
5       for (int i = 0; i < satir; i++)
6           for (int j = 0; j < sutun; j++)
7               t[j][i] = m[i][j];
8       return t;
9   }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`

m: 2 × 3

1	2	3
4	5	6

`m[0].length` = ilk satırın boyu = **sütun** sayısı.

Adım 3 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
▶     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
7             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`

m: 2 × 3

1	2	3
4	5	6

t: 3 × 2

0	0
0	0
0	0

Boyutlar yer değiştirir: **3 × 2**'lik yeni matris (hepsi 0).

Adım 4 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
▶     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
7             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3`

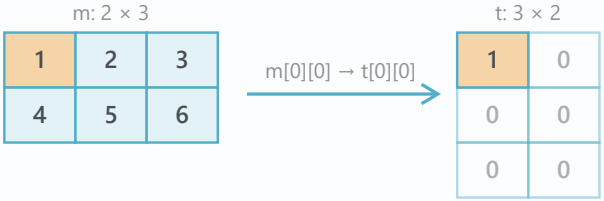
`t = [[0, 0], [0, 0], [0, 0]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`



t[0][0] = m[0][0] = 1: satır ↔ sütun.

Adım 5 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
▶             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3` `i = 0`

`j = 0`

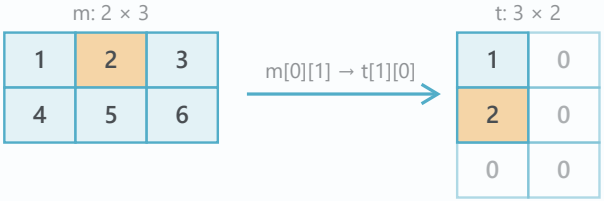
`t = [[1, 0], [0, 0], [0, 0]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`



t[1][0] = m[0][1] = 2: satır ↔ sütun.

Adım 6 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
▶             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3` `i = 0`

`j = 1`

`t = [[1, 0], [2, 0], [0, 0]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`



t[2][0] = m[0][2] = 3: satır ↔ sütun. m'nin 0. satırı t'nin 0. sütunu olur.

Adım 7 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
▶             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3` `i = 0`

`j = 2`

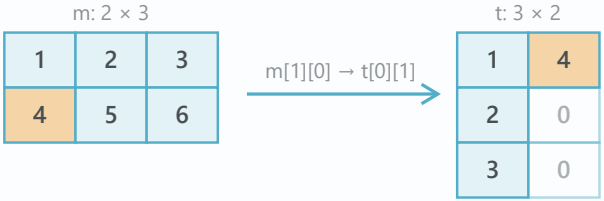
`t = [[1, 0], [2, 0], [3, 0]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`



t[0][1] = m[1][0] = 4: satır ↔ sütun.

Adım 8 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
▶             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3` `i = 1`

`j = 0`

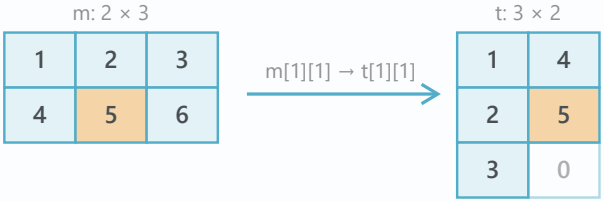
`t = [[1, 4], [2, 0], [3, 0]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`



t[1][1] = m[1][1] = 5: satır ↔ sütun.

Adım 9 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
▶             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3` `i = 1`

`j = 1`

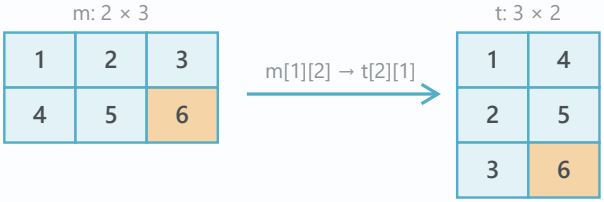
`t = [[1, 4], [2, 5], [3, 0]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`



t[2][1] = m[1][2] = 6: satır ↔ sütun.

Adım 10 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
▶             t[j][i] = m[i][j];
8     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3` `i = 1`

`j = 2`

`t = [[1, 4], [2, 5], [3, 6]]`

ÇAĞRI YIĞINI

`transpoz(m)`

`main()`

Kod İzleme: `transpoz(m)`

m: 2 × 3

1	2	3
4	5	6

t: 3 × 2

1	4
2	5
3	6

return t. 2 × 3 = 6 kopyalama → O(m·n).

Adım 11 / 11

```
1 static int[][] transpoz(int[][] m) {
2     int satir = m.length;
3     int sutun = m[0].length;
4     int[][] t = new int[sutun][satir];
5     for (int i = 0; i < satir; i++)
6         for (int j = 0; j < sutun; j++)
7             t[j][i] = m[i][j];
▶     return t;
9 }
```

DEĞİŞKENLER

`m = [[1, 2, 3], [4, 5, 6]]`

`satir = 2` `sutun = 3`

`t = [[1, 4], [2, 5], [3, 6]]`

ÇAĞRI YIĞINI

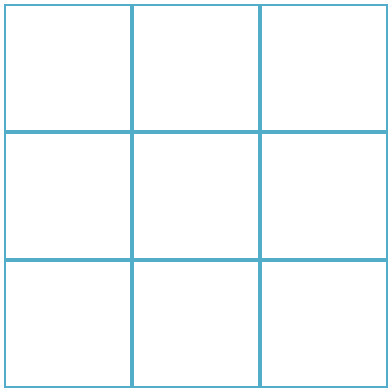
`transpoz(m)`

`main()`

Uygulama: Tic Tac Toe (X-O Oyunu)

- 3×3'lük bir oyun tahtası: `char[][] tahta`.
- İki oyuncu (X ve O) sırayla hamle yapar.
- Kendi işaretini aynı **satır**, **sütun** ya da **çaprazda** üçleyen kazanır.
- Tahtada boş yer kalmazsa oyun berabere biter.

Her hamle yalnızca bir dizi ataması:
`tahta[satir][sutun] = oyuncu`



Sıra **X** oyuncusunda. Bir hücreye tıkla.

yeni oyun

Tic Tac Toe: Ana Döngü

```
public class TicTacToe {   // import java.util.Scanner;
    static Scanner scanner = new Scanner(System.in); // tek Scanner

    public static void main(String[] args) {
        char[][] tahta = new char[3][3];   // 3x3'lük oyun tahtası
        char oyuncu = 'X';                   // ilk oyuncu X
        boolean oyunDevamEdiyor = true;
        tahtaDoldur(tahta);                  // tüm hücreler ' '
        while (oyunDevamEdiyor) {
            tahtayiGoster(tahta);
            hamleYap(tahta, oyuncu);
            oyunDevamEdiyor = oyunDevamEdiyorMu(tahta, oyuncu);
            oyuncu = (oyuncu == 'X') ? 'O' : 'X';   // sıra değişir
        }
        tahtayiGoster(tahta);
    }
```

`new char[3][3]` hücreleri `'\u0000'` ile başlar; boş hücreyi `' '` ile göstermek için tahta önce doldurulur. Tüm yardımcı metotlar `static`.

Tic Tac Toe: Tahta ve Hamle

```
static void tahtaDoldur(char[][] t) {
  for (int i = 0; i < 3; i++)
    for (int j = 0; j < 3; j++)
      t[i][j] = ' ';
}
static void tahtayiGoster(char[][] t) {
  System.out.println("-------------");
  for (int i = 0; i < 3; i++) {
    System.out.print("| ");
    for (int j = 0; j < 3; j++)
      System.out.print(t[i][j]+" | ");
    System.out.println("\n-------------");
  }
}
```

```
static void hamleYap(char[][] t,
                     char oyuncu) {
  int satir, sutun;
  do {
    System.out.print("Sıra " + oyuncu
      + ": satır ve sütun (1-3): ");
    satir = scanner.nextInt() - 1;
    sutun = scanner.nextInt() - 1;
  } while (satir < 0 || satir > 2
        || sutun < 0 || sutun > 2
        || t[satir][sutun] != ' ');
  t[satir][sutun] = oyuncu;
}
// kullanıcı 1-3 girer, dizi 0-2;
// geçersiz/dolu hücre → tekrar sor
```

Tic Tac Toe: Kazanma ve Beraberlik

8 kazanma koşulu (3 satır + 3 sütun + 2 çapraz) bir döngü (satır, sütun) ve tek bir return (çaprazlar) ile denetlenir.

```
static boolean kazandi(char[][] t, char o) {
    for (int i = 0; i < 3; i++)
        if ((t[i][0] == o && t[i][1] == o && t[i][2] == o)     // satır i
         || (t[0][i] == o && t[1][i] == o && t[2][i] == o))    // sütun i
            return true;
    return (t[0][0] == o && t[1][1] == o && t[2][2] == o)      // ana çapraz
        || (t[0][2] == o && t[1][1] == o && t[2][0] == o);     // ters çapraz
}
static boolean oyunDevamEdiyorMu(char[][] t, char o) {
    if (kazandi(t, o)) {
        System.out.println("Tebrikler, " + o + " oyuncusu kazandı!");
        return false;
    }
    for (char[] satir : t)
        for (char c : satir)
            if (c == ' ') return true;          // boş hücre var → devam
    System.out.println("Oyun berabere bitti.");
    return false;
}
```

Kod İzleme: `kazandi(t, 'X')`

```
▶ static boolean kazandi(char[][] t, char o) {
2   for (int i = 0; i < 3; i++)
3     if ((t[i][0] == o && t[i][1] == o
4          && t[i][2] == o)   // satır i
5      || (t[0][i] == o && t[1][i] == o
6          && t[2][i] == o))  // sütun i
7       return true;
8   return (t[0][0] == o && t[1][1] == o
9          && t[2][2] == o)   // ana çapraz
10      || (t[0][2] == o && t[1][1] == o
11         && t[2][0] == o);  // ters çapraz
12 }
```

	0	1	2
0	X	O	
1		X	O
2			X

Son hamleyi X yaptı; X kazandı mı? 8 çizgi denetlenecek.

Adım 1 / 9

DEĞİŞKENLER

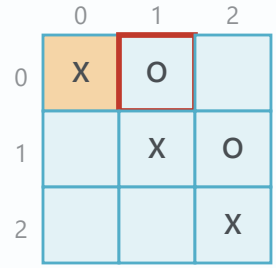
`o = 'X'`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
▶      if ((t[i][0] == o && t[i][1] == o
▶           && t[i][2] == o)   // satır i
5       || (t[0][i] == o && t[1][i] == o
6           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10      || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



i = 0, **satır 0**: t[0][1] = 'O' ≠ 'X' → yanlış. `&&` kısa devre: kalan 1 hücreye bakılmaz.

Adım 2 / 9

DEĞİŞKENLER

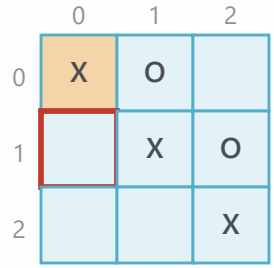
`o = 'X'` `i = 0`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
3      if ((t[i][0] == o && t[i][1] == o
4           && t[i][2] == o)   // satır i
▶       || (t[0][i] == o && t[1][i] == o
▶           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10       || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



i = 0, **sütun 0**: t[1][0] = boş ≠ 'X' → yanlış.
`&&` kısa devre: kalan 1 hücreye bakılmaz.

Adım 3 / 9

DEĞİŞKENLER

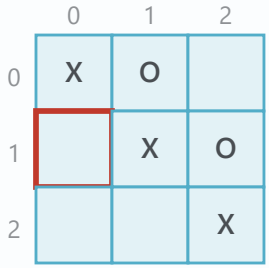
`o = 'X'` `i = 0`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
▶      if ((t[i][0] == o && t[i][1] == o
▶           && t[i][2] == o)   // satır i
5       || (t[0][i] == o && t[1][i] == o
6           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10      || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



i = 1, **satır 1**: t[1][0] = boş ≠ 'X' → yanlış. `&&` kısa devre: kalan 2 hücreye bakılmaz.

Adım 4 / 9

DEĞİŞKENLER

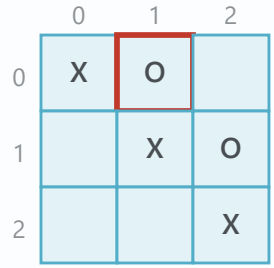
`o = 'X'` `i = 1`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
3      if ((t[i][0] == o && t[i][1] == o
4           && t[i][2] == o)   // satır i
▶       || (t[0][i] == o && t[1][i] == o
▶           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10       || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



i = 1, **sütun 1**: t[0][1] = 'O' ≠ 'X' → yanlış. `&&` kısa devre: kalan 2 hücreye bakılmaz.

Adım 5 / 9

DEĞİŞKENLER

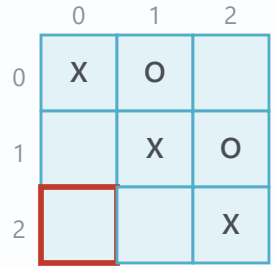
`o = 'X'` `i = 1`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
▶      if ((t[i][0] == o && t[i][1] == o
▶           && t[i][2] == o)   // satır i
5       || (t[0][i] == o && t[1][i] == o
6           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10      || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



i = 2, **satır 2**: t[2][0] = boş ≠ 'X' → yanlış. `&&` kısa devre: kalan 2 hücreye bakılmaz.

Adım 6 / 9

DEĞİŞKENLER

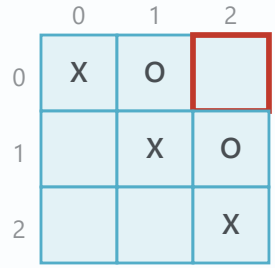
`o = 'X'` `i = 2`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
3      if ((t[i][0] == o && t[i][1] == o
4           && t[i][2] == o)   // satır i
▶       || (t[0][i] == o && t[1][i] == o
▶           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10       || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



i = 2, **sütun 2**: t[0][2] = boş ≠ 'X' → yanlış.
`&&` kısa devre: kalan 2 hücreye bakılmaz.

Adım 7 / 9

DEĞİŞKENLER

`o = 'X'` `i = 2`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
▶    for (int i = 0; i < 3; i++)
3      if ((t[i][0] == o && t[i][1] == o
4           && t[i][2] == o)   // satır i
5       || (t[0][i] == o && t[1][i] == o
6           && t[2][i] == o))  // sütun i
7        return true;
8    return (t[0][0] == o && t[1][1] == o
9           && t[2][2] == o)   // ana çapraz
10       || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```

	0	1	2
0	X	O	
1		X	O
2			X

i = 3 → döngü biter: hiçbir satır ve sütun X ile dolu değil.

Adım 8 / 9

DEĞİŞKENLER

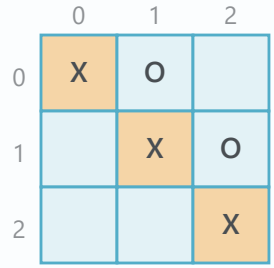
`o = 'X'` `i = 3`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

Kod İzleme: `kazandi(t, 'X')`

```
1  static boolean kazandi(char[][] t, char o) {
2    for (int i = 0; i < 3; i++)
3      if ((t[i][0] == o && t[i][1] == o
4           && t[i][2] == o)   // satır i
5       || (t[0][i] == o && t[1][i] == o
6           && t[2][i] == o))  // sütun i
7        return true;
▶    return (t[0][0] == o && t[1][1] == o
▶         && t[2][2] == o)   // ana çapraz
10      || (t[0][2] == o && t[1][1] == o
11          && t[2][0] == o);  // ters çapraz
12 }
```



Ana çapraz: t[0][0], t[1][1], t[2][2] hepsi 'X' → `||` sağ tarafa hiç bakılmaz, **return true**.

Adım 9 / 9

DEĞİŞKENLER

`o = 'X'`

ÇAĞRI YIĞINI

- `kazandi(t, 'X')`
- `oyunDevamEdiyorMu(t, 'X')`
- `main()`

KARMAŞIKLIK

Dizi İşlemlerinin Maliyeti

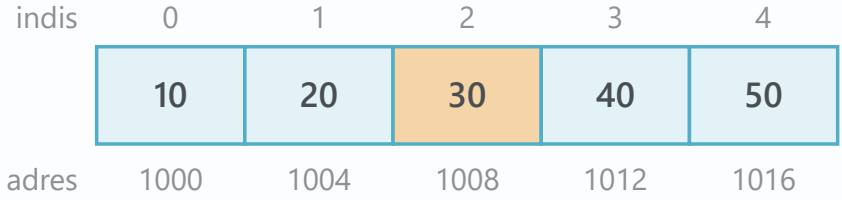
Algoritma karmaşıklığı, çalışma süresinin ve bellek kullanımının girdi boyutu **n**'ye göre nasıl büyüdüğünü gösterir.

İşlem	Zaman	Neden?
İndisle erişim / güncelleme `a[i]`	**O(1)**	adres doğrudan hesaplanır
Değer arama (sırasız dizi)	**O(n)**	her öğeye bakmak gerekebilir
Değer arama (sıralı dizi, ikili arama)	**O(log n)**	her adımda aralık yarıya iner
Sona ekleme / sondan silme (yer varsa)	**O(1)**	kaydırma yok
Araya ekleme / aradan silme	**O(n)**	sonraki öğeler kaydırılır
Sıralama (iyi bir algoritma)	**O(n log n)**	ör. birleştirme sıralaması
Kopyalama / büyütme	**O(n)**	tüm öğeler yeni diziye taşınır

KARMAŞIKLIK

Dizi Öğesine Erişim: O(1)

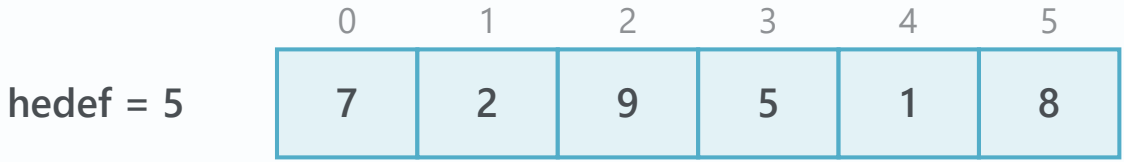
- İşlem süresi dizinin boyutundan bağımsızdır: 5 öğeli dizide de 5 milyon öğelide de aynı.
- Çünkü her öğenin adresi hesaplanabilir: `adres = başlangıç + i × öğe boyutu`.



adres(dizi[2]) = 1000 + 2 × 4 = 1008

```
int[] dizi = {10, 20, 30, 40, 50};
int oge = dizi[2];      // 30 — tek adımda, O(1)
```

Dizide Arama: Doğrusal Arama, O(n)



1. çağrı: **5** aranıyor. Dizi sırasız → baştan sona bakmaktan başka yol yok.

Adım 1 / 14

```
▶ static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
3     for (int oge : dizi)
4         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

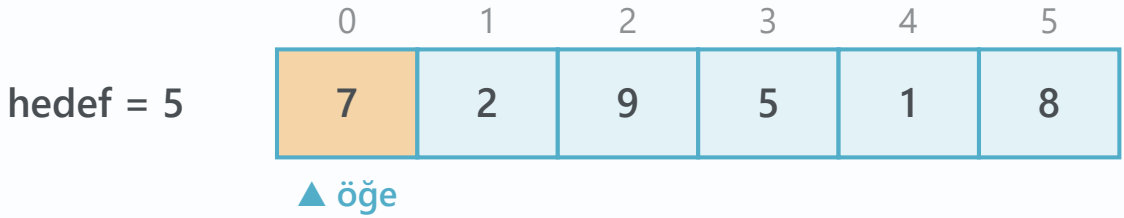
hedef = **5**

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 5)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



7 ≠ 5 → sonraki öğe.

Adım 2 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

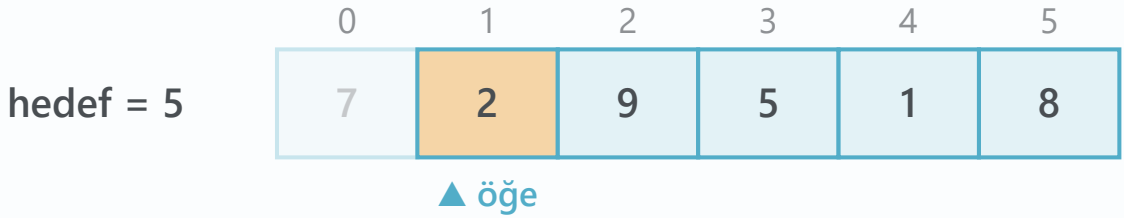
`hedef = 5` `oge = 7` `karşılaştırma = 1`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 5)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



2 ≠ 5 → sonraki öğe.

Adım 3 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

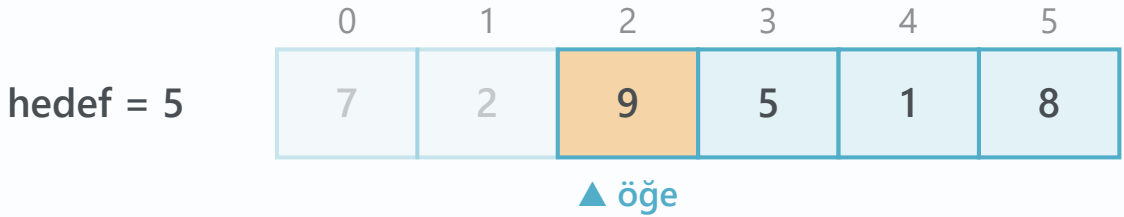
`hedef = 5` `oge = 2` `karşılaştırma = 2`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 5)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



9 ≠ 5 → sonraki öğe.

Adım 4 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

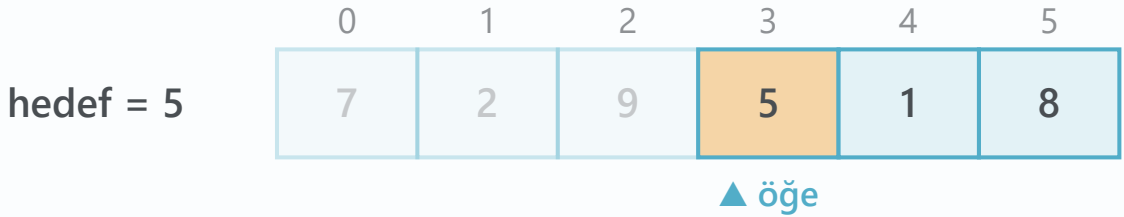
`hedef = 5` `oge = 9` `karşılaştırma = 3`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 5)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



5 == 5 → **eşit**.

Adım 5 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

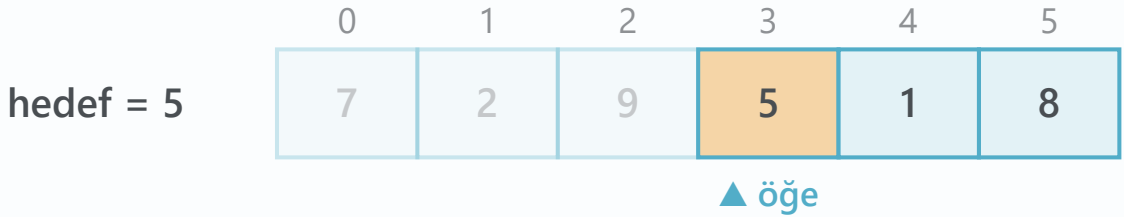
`hedef = 5` `oge = 5` `karşılaştırma = 4`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 5)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



return true: 4 karşılaştırmada bulundu; kalan öğelere bakılmaz. Hedef baştaysa 1 karşılaştırma → en iyi durum O(1).

Adım 6 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
3     for (int oge : dizi)
4         if (oge == hedef)
▶             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

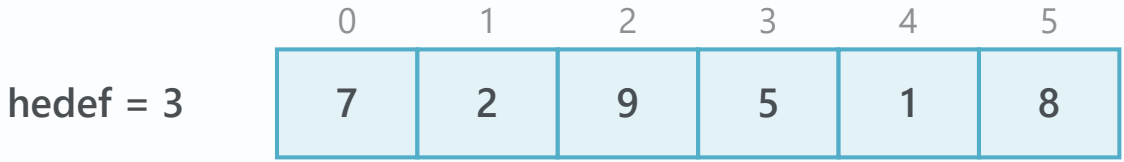
`hedef = 5` `oge = 5` `karşılaştırma = 4`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 5)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



2. çağrı: **3** aranıyor. Dizi sırasız → baştan sona bakmaktan başka yol yok.

Adım 7 / 14

```
static boolean dogrusalArama(int[] dizi,
                             int hedef) {
    for (int oge : dizi)
        if (oge == hedef)
            return true; // bulundu
    return false;
}
```

DEĞİŞKENLER

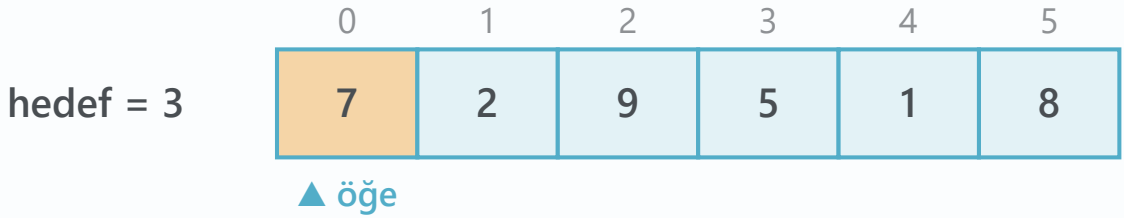
`hedef = 3`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



7 ≠ 3 → sonraki öğe.

Adım 8 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

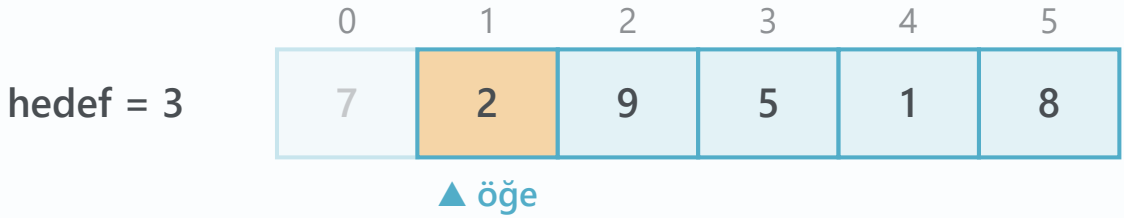
`hedef = 3` `oge = 7` `karşılaştırma = 1`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



2 ≠ 3 → sonraki öğe.

Adım 9 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

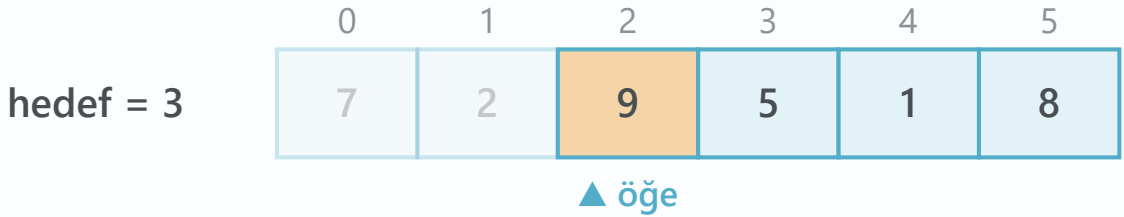
`hedef = 3` `oge = 2` `karşılaştırma = 2`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



9 ≠ 3 → sonraki öğe.

Adım 10 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

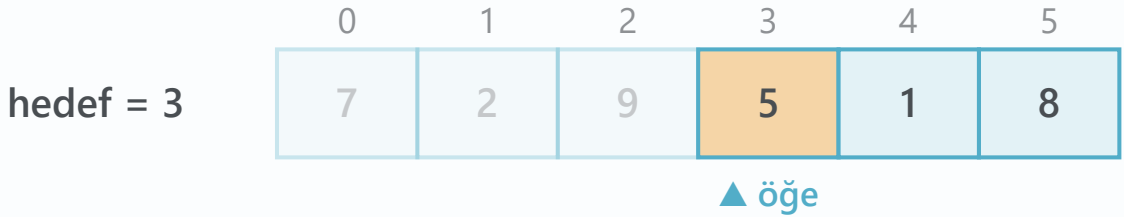
`hedef = 3` `oge = 9` `karşılaştırma = 3`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



5 ≠ 3 → sonraki öğe.

Adım 11 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

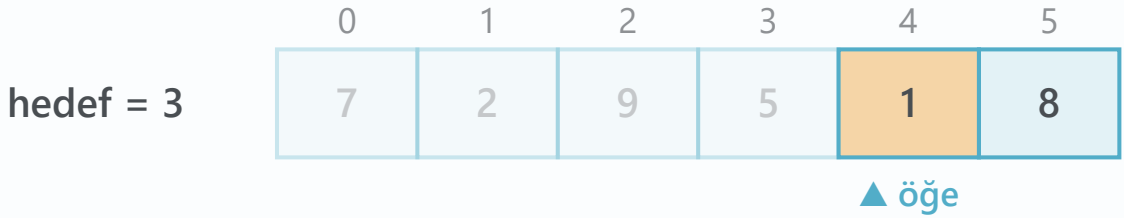
`hedef = 3` `oge = 5` `karşılaştırma = 4`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



1 ≠ 3 → sonraki öğe.

Adım 12 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶       for (int oge : dizi)
▶           if (oge == hedef)
5               return true; // bulundu
6       return false;
7   }
```

DEĞİŞKENLER

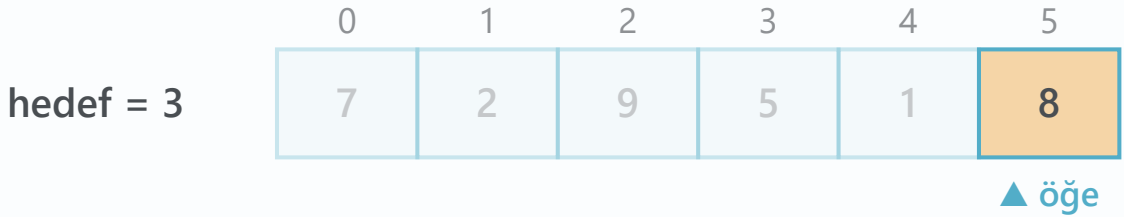
`hedef = 3` `oge = 1` `karşılaştırma = 5`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)



8 ≠ 3 → sonraki öğe.

Adım 13 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
▶     for (int oge : dizi)
▶         if (oge == hedef)
5             return true; // bulundu
6     return false;
7 }
```

DEĞİŞKENLER

`hedef = 3` `oge = 8`

`karşılaştırma = 6`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

Dizide Arama: Doğrusal Arama, O(n)

hedef = 3

0	1	2	3	4	5
7	2	9	5	1	8

Dizi bitti, **return false**: n = 6 öğenin hepsine bakıldı → en kötü durum **O(n)**.

Adım 14 / 14

```
1 static boolean dogrusalArama(int[] dizi,
2                              int hedef) {
3     for (int oge : dizi)
4         if (oge == hedef)
5             return true; // bulundu
▶     return false;
7 }
```

DEĞİŞKENLER

`hedef = 3` `karşılaştırma = 6`

ÇAĞRI YIĞINI

`dogrusalArama(dizi, 3)`

`main()`

KARMAŞIKLIK

İki Dizide Ortak Öğe Bulma: O(n·m)

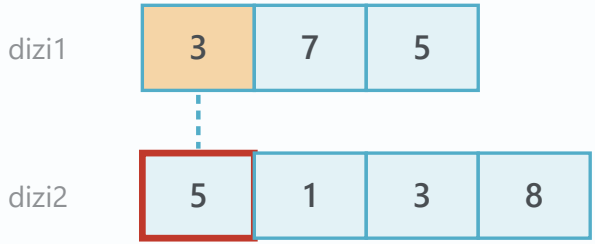
```
static void eslestir(int[] dizi1, int[] dizi2) {
    for (int oge1 : dizi1)          // n kez
        for (int oge2 : dizi2)      // m kez
            if (oge1 == oge2)
                System.out.println(
                    "Eşleşen öğe: " + oge1);
}
```

- Her `dizi1` öğesi, `dizi2` 'nin her öğesiyle karşılaştırılır: n·m karşılaştırma.
- İki dizi de n öğeliyse **O(n²)**: büyük veri için verimsiz.
- Daha iyisi: birini sıralayıp ikili arama O(n log n) ya da hash kümesi, ortalama O(n).



3 × 4 = 12 karşılaştırma, 2 eşleşme

Kod İzleme: `eslestir()`



karşılaştırma: 1

Dış döngü: **oge1 = 3**; iç döngü dizi2'yi baştan tarar. 3 ≠ 5.

Adım 1 / 15

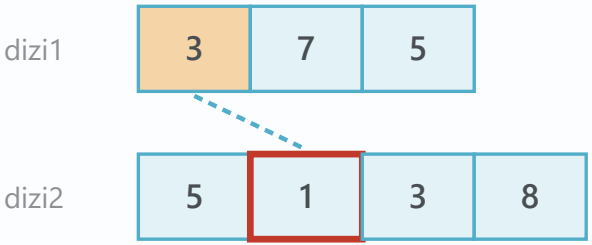
```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

`oge1 = 3` `oge2 = 5` `karşılaştırma = 1`

ÇIKTI

Kod İzleme: `eslestir()`



karşılaştırma: 2

3 ≠ 1.

Adım 2 / 15

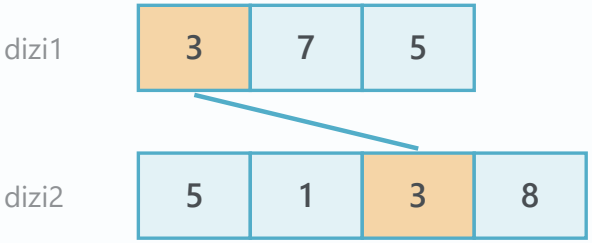
```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

`oge1 = 3` `oge2 = 1` `karşılaştırma = 2`

ÇIKTI

Kod İzleme: `eslestir()`



karşılaştırma: 3

3 == 3 → **eşleşme**.

Adım 3 / 15

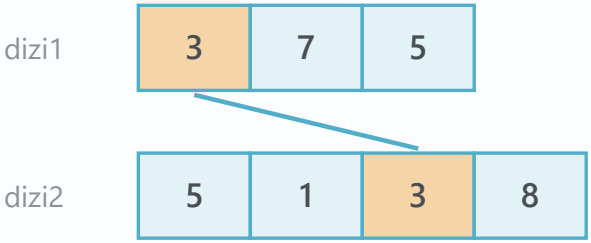
```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

`oge1 = 3` `oge2 = 3` `karşılaştırma = 3`

ÇIKTI

Kod İzleme: `eslestir()`



karşılaştırma: 3

Eşleşme yazdırılır; iç döngü yine de **sonuna kadar** devam eder.

Adım 4 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
3         for (int oge2 : dizi2)      // m kez
4             if (oge1 == oge2)
▶                 System.out.println(
▶                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

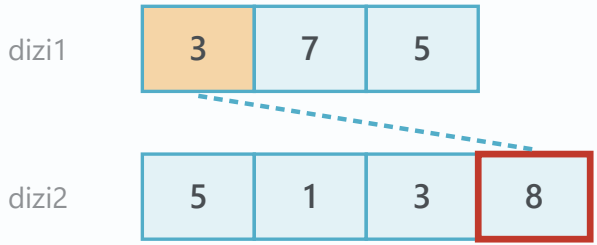
`oge1 = 3` `oge2 = 3`

`karşılaştırma = 3`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 4

3 ≠ 8.

Adım 5 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

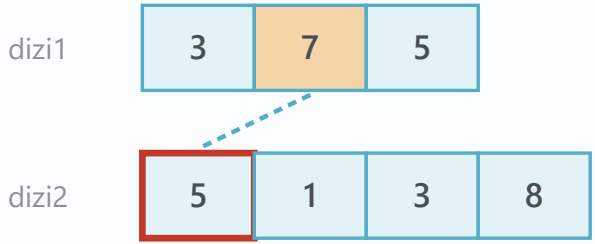
DEĞİŞKENLER

`oge1 = 3` `oge2 = 8` `karşılaştırma = 4`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 5

Dış döngü: **oge1 = 7**; iç döngü dizi2'yi baştan tarar. 7 ≠ 5.

Adım 6 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

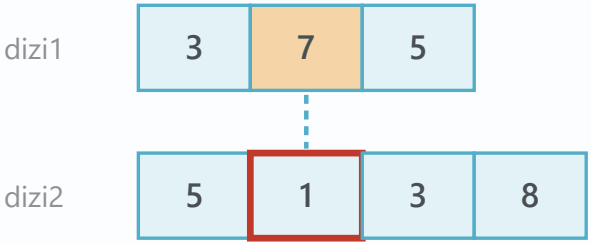
DEĞİŞKENLER

`oge1 = 7` `oge2 = 5` `karşılaştırma = 5`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 6

7 ≠ 1.

Adım 7 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

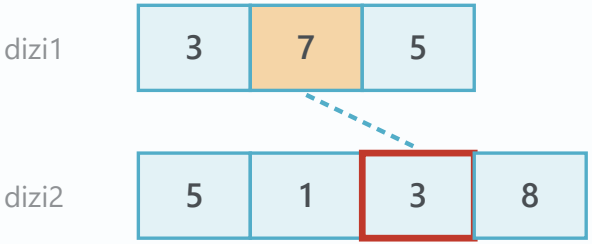
DEĞİŞKENLER

`oge1 = 7` `oge2 = 1` `karşılaştırma = 6`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 7

7 ≠ 3.

Adım 8 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

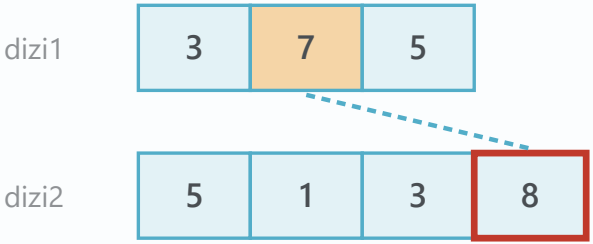
DEĞİŞKENLER

`oge1 = 7` `oge2 = 3` `karşılaştırma = 7`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 8

7 ≠ 8.

Adım 9 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

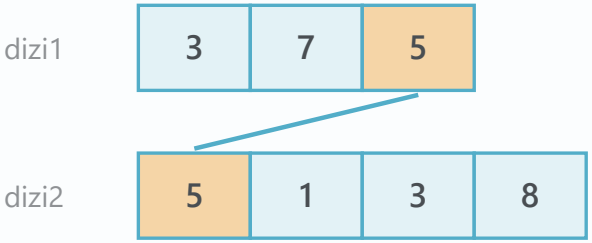
DEĞİŞKENLER

`oge1 = 7` `oge2 = 8` `karşılaştırma = 8`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 9

5 == 5 → **eşleşme**.

Adım 10 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

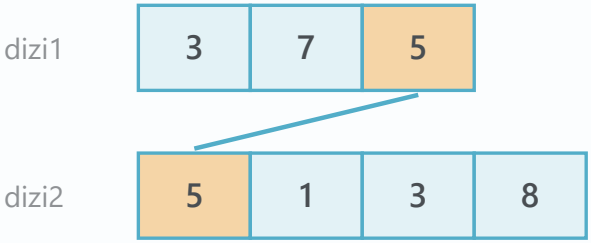
`oge1 = 5` `oge2 = 5`

`karşılaştırma = 9`

ÇIKTI

```
Eşleşen öğe: 3
```

Kod İzleme: `eslestir()`



karşılaştırma: 9

Eşleşme yazdırılır; iç döngü yine de **sonuna kadar** devam eder.

Adım 11 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
3         for (int oge2 : dizi2)      // m kez
4             if (oge1 == oge2)
▶                 System.out.println(
▶                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

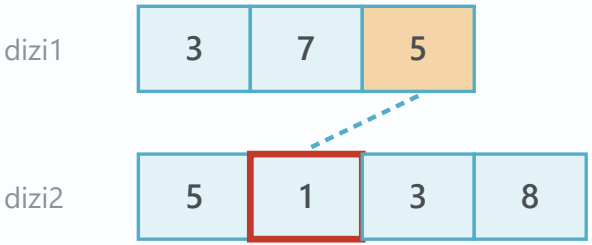
`oge1 = 5` `oge2 = 5`

`karşılaştırma = 9`

ÇIKTI

```
Eşleşen öğe: 3
Eşleşen öğe: 5
```

Kod İzleme: `eslestir()`



karşılaştırma: 10

5 ≠ 1.

Adım 12 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

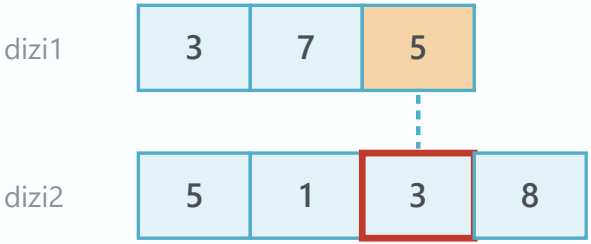
`oge1 = 5` `oge2 = 1`

`karşılaştırma = 10`

ÇIKTI

```
Eşleşen öğe: 3
Eşleşen öğe: 5
```

Kod İzleme: `eslestir()`



karşılaştırma: 11

5 ≠ 3.

Adım 13 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

`oge1 = 5` `oge2 = 3`

`karşılaştırma = 11`

ÇIKTI

```
Eşleşen öğe: 3
Eşleşen öğe: 5
```

Kod İzleme: `eslestir()`



karşılaştırma: 12

5 ≠ 8.

Adım 14 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
▶         for (int oge2 : dizi2)      // m kez
▶             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
7 }
```

DEĞİŞKENLER

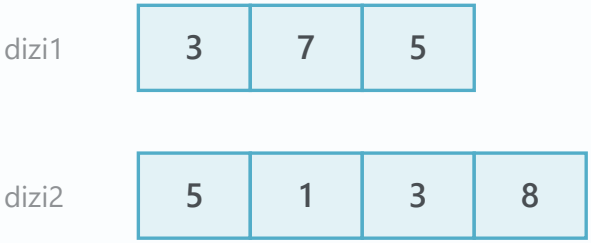
`oge1 = 5` `oge2 = 8`

`karşılaştırma = 12`

ÇIKTI

```
Eşleşen öğe: 3
Eşleşen öğe: 5
```

Kod İzleme: `eslestir()`



karşılaştırma: 12

Bitti: 3 × 4 = **12 karşılaştırma** = n·m. İki dizi de n öğeliyse O(n^2).

Adım 15 / 15

```
1 static void eslestir(int[] dizi1, int[] dizi2) {
2     for (int oge1 : dizi1)          // n kez
3         for (int oge2 : dizi2)      // m kez
4             if (oge1 == oge2)
5                 System.out.println(
6                     "Eşleşen öğe: " + oge1);
▶ }
```

DEĞİŞKENLER

`karşılaştırma = 12`

ÇIKTI

```
Eşleşen öğe: 3
Eşleşen öğe: 5
```

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Araya Ekleme: Sağa Kaydırma, O(n)



Kapasite 6, dolu 5 hücre. **4**'ü indis 1'e eklemek için önce yer açılmalı.

Adım 1 / 8

```
▶ static int ekle(int[] a, int boyut,
2               int indis, int deger) {
3   // ön koşul: boyut < a.length
4   for (int k = boyut; k > indis; k--)
5     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 1, 0]` `boyut = 5`

`indis = 1` `deger = 4`

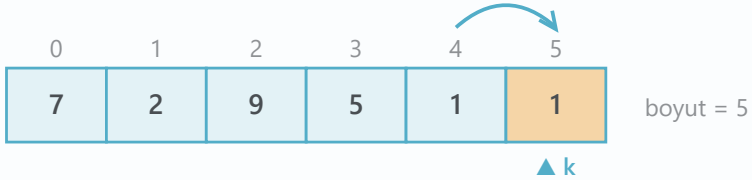
ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

Araya Ekleme: Sağa Kaydırma, O(n)



k = 5: **a[5] = a[4] = 1** (bir sağa kopyalandı). Sağdan başlanır: soldan başlasaydık 2, 9'un üzerine yazardı.

Adım 2 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
▶   for (int k = boyut; k > indis; k--)
▶     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 1, 1]` `boyut = 5` `indis = 1` `deger = 4` `k = 5`

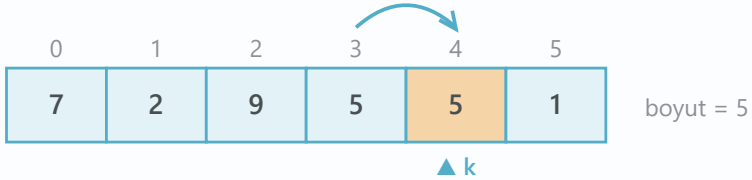
ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

Araya Ekleme: Sağa Kaydırma, O(n)



k = 4: **a[4] = a[3] = 5** (bir sağa kopyalandı).

Adım 3 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
▶   for (int k = boyut; k > indis; k--)
▶     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 5, 1]` `boyut = 5` `indis = 1` `deger = 4` `k = 4`

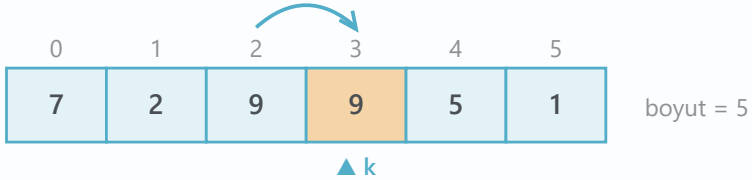
ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

Araya Ekleme: Sağa Kaydırma, O(n)



k = 3: **a[3] = a[2] = 9** (bir sağa kopyalandı).

Adım 4 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
▶   for (int k = boyut; k > indis; k--)
▶     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 9, 5, 1]` `boyut = 5` `indis = 1` `deger = 4` `k = 3`

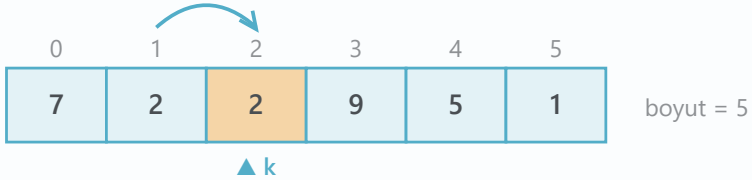
ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

Araya Ekleme: Sağa Kaydırma, O(n)



k = 2: **a[2] = a[1] = 2** (bir sağa kopyalandı).

Adım 5 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
▶   for (int k = boyut; k > indis; k--)
▶     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 2, 2, 9, 5, 1]` `boyut = 5` `indis = 1` `deger = 4` `k = 2`

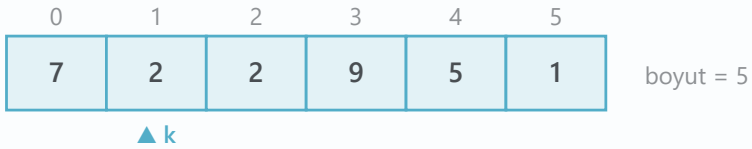
ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

Araya Ekleme: Sağa Kaydırma, O(n)



k = 1, 1 > 1 yanlış → döngü biter. 4 kaydırma yapıldı; a[1] ve a[2] şu an aynı (2).

Adım 6 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
▶  for (int k = boyut; k > indis; k--)
5     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 2, 2, 9, 5, 1]` `boyut = 5` `indis = 1` `deger = 4` `k = 1`

ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

Araya Ekleme: Sağa Kaydırma, O(n)



Açılan yere yazılır: **a[1] = 4**.

Adım 7 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
4   for (int k = boyut; k > indis; k--)
5     a[k] = a[k - 1];   // sağa kaydır
▶   a[indis] = deger;
7   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 4, 2, 9, 5, 1]` `boyut = 5` `indis = 1` `deger = 4`

ÇAĞRI YIĞINI

`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Araya Ekleme: Sağa Kaydırma, O(n)



return 6. Başa eklemek n, ortaya ≈ n/2 kaydırma → **O(n)**; sona eklemek (yer varsa) O(1).

Adım 8 / 8

```
1 static int ekle(int[] a, int boyut,
2              int indis, int deger) {
3   // ön koşul: boyut < a.length
4   for (int k = boyut; k > indis; k--)
5     a[k] = a[k - 1];   // sağa kaydır
6   a[indis] = deger;
▶   return boyut + 1;   // yeni boyut
8 }
```

DEĞİŞKENLER

`a = [7, 4, 2, 9, 5, 1]` `boyut = 6` `indis = 1` `deger = 4`

ÇAĞRI YIĞINI

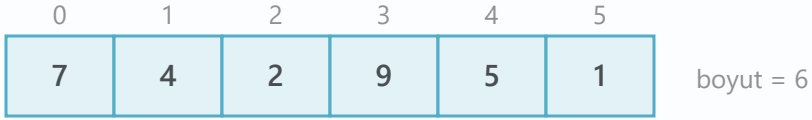
`ekle(a, 5, 1, 4)`

`main()`

Dizi doluysa (boyut == a.length) önce daha büyük bir dizi oluşturulup kopyalanır: O(n).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Aradan Silme: Sola Kaydırma, O(n)



Önceki eklemeden sonra: **a[1] = 4** silinecek. Boşluk kalmaması için sağdakiler sola kayar.

Adım 1 / 8

```
▶ static int sil(int[] a, int boyut, int indis) {
2   for (int k = indis; k < boyut - 1; k++)
3     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

a = [7, 4, 2, 9, 5, 1]

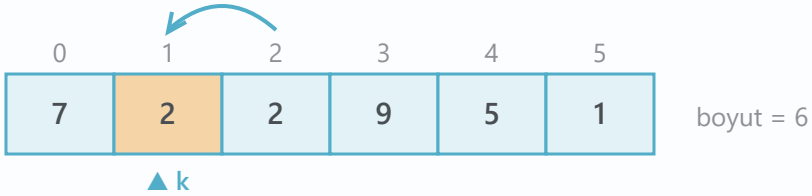
boyut = 6 | indis = 1

ÇAĞRI YIĞINI

sil(a, 6, 1)

main()

Aradan Silme: Sola Kaydırma, O(n)



k = 1: **a[1] = a[2] = 2**. Bu kez **soldan** başlanır: silinen değerin üzerine yazılır.

Adım 2 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
▶   for (int k = indis; k < boyut - 1; k++)
▶     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

`a = [7, 2, 2, 9, 5, 1]`

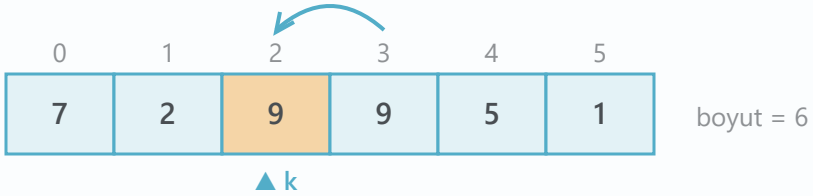
`boyut = 6` `indis = 1` `k = 1`

ÇAĞRI YIĞINI

`sil(a, 6, 1)`

`main()`

Aradan Silme: Sola Kaydırma, O(n)



k = 2: **a[2] = a[3] = 9**.

Adım 3 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
▶   for (int k = indis; k < boyut - 1; k++)
▶     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

a = [7, 2, 9, 9, 5, 1]

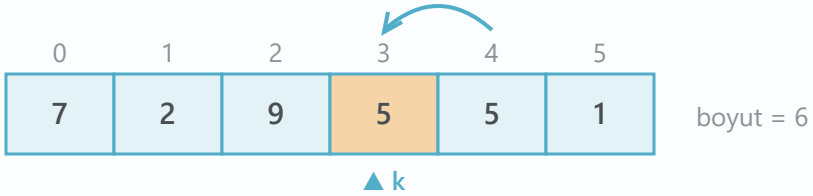
boyut = 6 | indis = 1 | k = 2

ÇAĞRI YIĞINI

sil(a, 6, 1)

main()

Aradan Silme: Sola Kaydırma, O(n)



k = 3: **a[3] = a[4] = 5**.

Adım 4 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
▶   for (int k = indis; k < boyut - 1; k++)
▶     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 5, 1]`

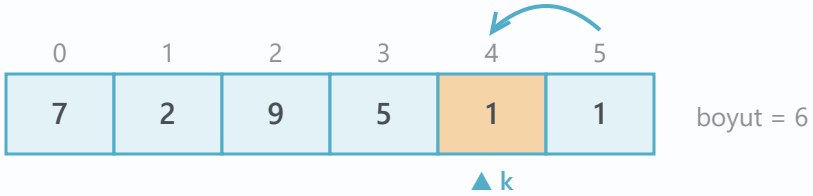
`boyut = 6` `indis = 1` `k = 3`

ÇAĞRI YIĞINI

`sil(a, 6, 1)`

`main()`

Aradan Silme: Sola Kaydırma, O(n)



k = 4: **a[4] = a[5] = 1**.

Adım 5 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
▶   for (int k = indis; k < boyut - 1; k++)
▶     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

a = [7, 2, 9, 5, 1, 1]

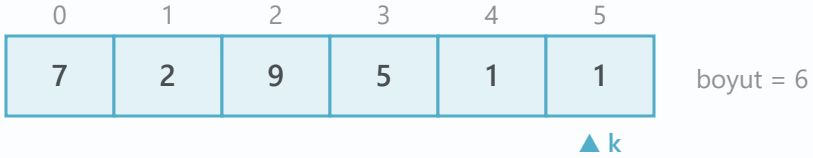
boyut = 6 | indis = 1 | k = 4

ÇAĞRI YIĞINI

sil(a, 6, 1)

main()

Aradan Silme: Sola Kaydırma, O(n)



k = 5 < 5 yanlış → döngü biter. 4 kaydırma.

Adım 6 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
▶   for (int k = indis; k < boyut - 1; k++)
3     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 1, 1]`

`boyut = 6` `indis = 1` `k = 5`

ÇAĞRI YIĞINI

`sil(a, 6, 1)`

`main()`

Aradan Silme: Sola Kaydırma, O(n)



Son hücrede eski değerin kopyası kalmıştı; 0 yazılır (nesne dizilerinde `null`: GC için önemli).

Adım 7 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
2   for (int k = indis; k < boyut - 1; k++)
3     a[k] = a[k + 1];    // sola kaydır
▶   a[boyut - 1] = 0;     // eski son hücreyi temizle
5   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 1, 0]`

`boyut = 6` `indis = 1`

ÇAĞRI YIĞINI

`sil(a, 6, 1)`

`main()`

Aradan Silme: Sola Kaydırma, O(n)



return 5. Baştan silme n − 1 kaydırma → **O(n)**; sondan silme O(1).

Adım 8 / 8

```
1 static int sil(int[] a, int boyut, int indis) {
2   for (int k = indis; k < boyut - 1; k++)
3     a[k] = a[k + 1];    // sola kaydır
4   a[boyut - 1] = 0;     // eski son hücreyi temizle
▶   return boyut - 1;     // yeni boyut
6 }
```

DEĞİŞKENLER

`a = [7, 2, 9, 5, 1, 0]`

`boyut = 5` `indis = 1`

ÇAĞRI YIĞINI

`sil(a, 6, 1)`

`main()`

KARMAŞIKLIK

Dizi Sıralama: Zaman Karmaşıklığı

Sıralamanın maliyeti kullanılan algoritmaya bağlıdır:

Algoritma	En iyi	Ortalama	En kötü	Ek bellek
Kabarcık sıralaması (Bubble Sort)	O(n)*	$O(n^2)$	$O(n^2)$	O(1)
Seçimli sıralama (Selection Sort)	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Eklemeli sıralama (Insertion Sort)	O(n)	$O(n^2)$	$O(n^2)$	O(1)
Hızlı sıralama (Quick Sort)	O(n log n)	O(n log n)	**$O(n^2)$**	O(log n)**
Birleştirme sıralaması (Merge Sort)	O(n log n)	O(n log n)	O(n log n)	O(n)

* Takas olmayan turda duran sürümde. Hızlı sıralamanın en kötü durumu kötü pivot seçiminde (ör. sıralı girdi + ilk öğe pivot) ortaya çıkar. ** Hızlı sıralamanın ek belleği ortalamada O(log n), en kötü durumda O(n)'dir (özyineleme derinliği).

`Arrays.sort` : ilkel diziler için çift pivotlu hızlı sıralama, nesne dizileri için kararlı (stable) TimSort kullanır.

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
▶ int[] dizi = {64, 25, 12, 22, 11};
▶ int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



Sırasız 5 öğeli dizi. Her tur, sıralanmamış kısmın en küçüğünü o kısmın başına getirecek.

Adım 1 / 19

DEĞİŞKENLER

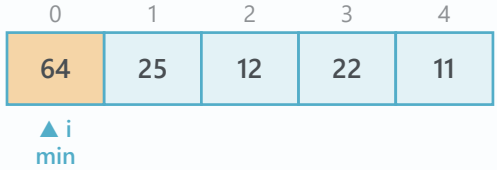
`dizi = [64, 25, 12, 22, 11]` `n = 5`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
▶      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



1. tur: sıralanmamış kısım indis 0'dan başlar; şimdilik en küçük dizi[0] = 64.

Adım 2 / 19

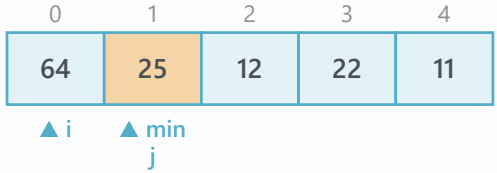
DEĞİŞKENLER

`dizi = [64, 25, 12, 22, 11]` `i = 0` `minIndis = 0` `karş = 0`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
▶              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



25 < 64 → **minIndis = 1**.

Adım 3 / 19

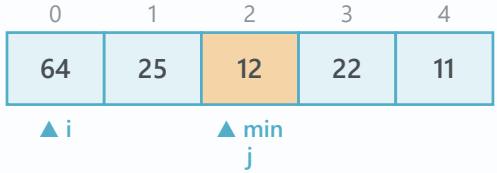
DEĞİŞKENLER

`dizi = [64, 25, 12, 22, 11]` `i = 0` `j = 1` `minIndis = 1` `karş = 1`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
▶              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



12 < 25 → **minIndis = 2**.

Adım 4 / 19

DEĞİŞKENLER

`dizi = [64, 25, 12, 22, 11]` `i = 0` `j = 2` `minIndis = 2` `karş = 2`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



22 < 12 yanlış → değişmez.

Adım 5 / 19

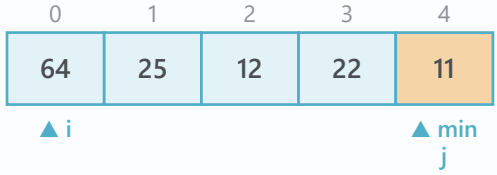
DEĞİŞKENLER

`dizi = [64, 25, 12, 22, 11]` `i = 0` `j = 3` `minIndis = 2` `karş = 3`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
▶              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



11 < 12 → **minIndis = 4**.

Adım 6 / 19

DEĞİŞKENLER

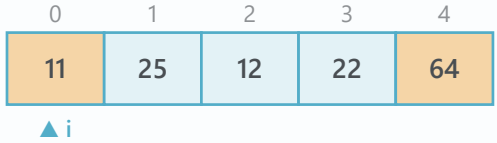
`dizi = [64, 25, 12, 22, 11]` `i = 0` `j = 4` `minIndis = 4` `karş = 4`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
▶      int gecici = dizi[minIndis];
▶      dizi[minIndis] = dizi[i];
▶      dizi[i] = gecici;
12 }
```



Takas: dizi[0] ↔ dizi[4] (64 ↔ 11). İndis 0 artık kesin yerinde.

Adım 7 / 19

DEĞİŞKENLER

`dizi = [11, 25, 12, 22, 64]` `i = 0` `minIndis = 4` `karş = 4`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
▶      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



2. tur: sıralanmamış kısım indis 1'den başlar; şimdilik en küçük dizi[1] = 25.

Adım 8 / 19

DEĞİŞKENLER

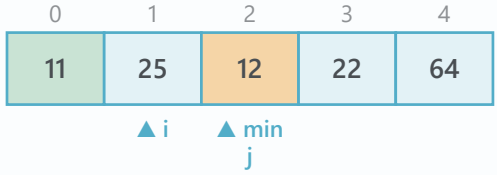
`dizi = [11, 25, 12, 22, 64]` `i = 1` `minIndis = 1` `karş = 4`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
▶              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



12 < 25 → **minIndis = 2**.

Adım 9 / 19

DEĞİŞKENLER

`dizi = [11, 25, 12, 22, 64]` `i = 1` `j = 2` `minIndis = 2` `karş = 5`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



22 < 12 yanlış → değişmez.

Adım 10 / 19

DEĞİŞKENLER

`dizi = [11, 25, 12, 22, 64]` `i = 1` `j = 3` `minIndis = 2` `karş = 6`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



64 < 12 yanlış → değişmez.

Adım 11 / 19

DEĞİŞKENLER

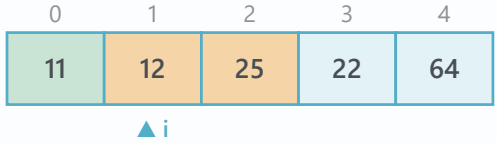
`dizi = [11, 25, 12, 22, 64]` `i = 1` `j = 4` `minIndis = 2` `karş = 7`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
▶      int gecici = dizi[minIndis];
▶      dizi[minIndis] = dizi[i];
▶      dizi[i] = gecici;
12 }
```



Takas: dizi[1] ↔ dizi[2] (25 ↔ 12). İndis 1 artık kesin yerinde.

Adım 12 / 19

DEĞİŞKENLER

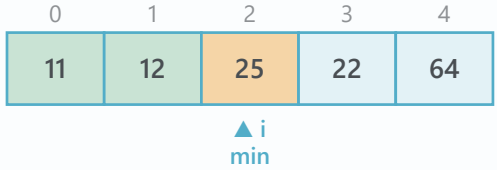
`dizi = [11, 12, 25, 22, 64]` `i = 1` `minIndis = 2` `karş = 7`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
▶      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



3. tur: sıralanmamış kısım indis 2'den başlar; şimdilik en küçük dizi[2] = 25.

Adım 13 / 19

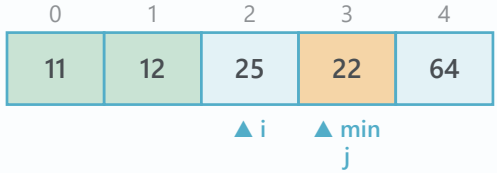
DEĞİŞKENLER

`dizi = [11, 12, 25, 22, 64]` `i = 2` `minIndis = 2` `karş = 7`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
▶              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



22 < 25 → **minIndis = 3**.

Adım 14 / 19

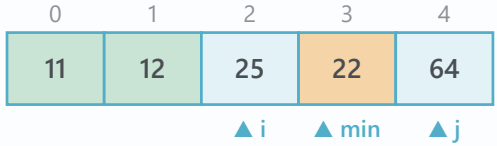
DEĞİŞKENLER

`dizi = [11, 12, 25, 22, 64]` `i = 2` `j = 3` `minIndis = 3` `karş = 8`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



64 < 22 yanlış → değişmez.

Adım 15 / 19

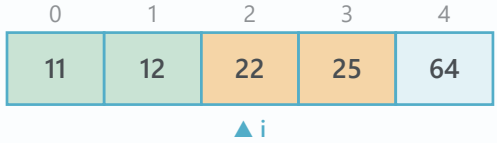
DEĞİŞKENLER

`dizi = [11, 12, 25, 22, 64]` `i = 2` `j = 4` `minIndis = 3` `karş = 9`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
▶      int gecici = dizi[minIndis];
▶      dizi[minIndis] = dizi[i];
▶      dizi[i] = gecici;
12 }
```



Takas: dizi[2] ↔ dizi[3] (25 ↔ 22). İndis 2 artık kesin yerinde.

Adım 16 / 19

DEĞİŞKENLER

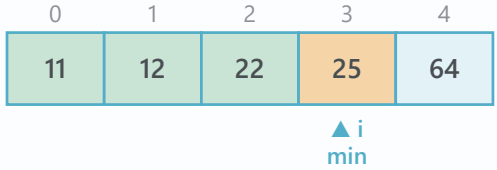
`dizi = [11, 12, 22, 25, 64]` `i = 2` `minIndis = 3` `karş = 9`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
▶      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



4. tur: sıralanmamış kısım indis 3'ten başlar; şimdilik en küçük dizi[3] = 25.

Adım 17 / 19

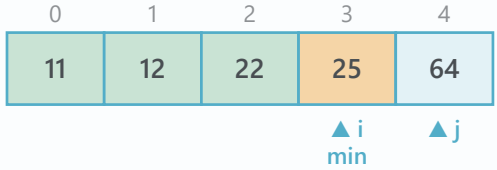
DEĞİŞKENLER

`dizi = [11, 12, 22, 25, 64]` `i = 3` `minIndis = 3` `karş = 9`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
▶          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
9      int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```



64 < 25 yanlış → değişmez.

Adım 18 / 19

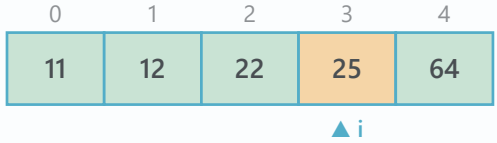
DEĞİŞKENLER

`dizi = [11, 12, 22, 25, 64]` `i = 3` `j = 4` `minIndis = 3` `karş = 10`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

Seçimli Sıralama (Selection Sort)

```
1  int[] dizi = {64, 25, 12, 22, 11};
2  int n = dizi.length;
3  for (int i = 0; i < n - 1; i++) {
4      int minIndis = i;
5      for (int j = i + 1; j < n; j++)
6          if (dizi[j] < dizi[minIndis])
7              minIndis = j;
8      // takas: indislerle, yerinde
▶      int gecici = dizi[minIndis];
▶      dizi[minIndis] = dizi[i];
▶      dizi[i] = gecici;
12 }
```



En küçük zaten yerinde (minIndis = i = 3); takas kendisiyle yapılır, dizi değişmez. Son öğe de kendiliğinden yerinde → **sıralandı**: 4 + 3 + 2 + 1 = 10 = n(n−1)/2 karşılaştırma, O(n²).

Adım 19 / 19

DEĞİŞKENLER

`dizi = [11, 12, 22, 25, 64]` `i = 3` `minIndis = 3` `karş = 10`

Java'da parametreler değerle geçer: `yerDegistir(dizi[minIndis], dizi[i])` yalnızca kopyaları değiştirir. Takas indislerle yapılmalı (ya da `yerDegistir(dizi, i, minIndis)`).

KARMAŞIKLIK · KODU ADIM ADIM ÇALIŞTIR

Eklemeli Sıralama (Insertion Sort)



Sıralı kısım başta yalnız a[0]'dır (tek öğe her zaman sıralıdır). Her tur, sıradaki öğeyi sıralı kısımda doğru yerine **kaydırarak** yerleştirir.

Adım 1 / 16

```
▶ static void eklemeliSirala(int[] a) {
2     for (int k = 1; k < a.length; k++) {
3         int anahtar = a[k];
4         int j = k;
5         while (j > 0 && a[j - 1] > anahtar) {
6             a[j] = a[j - 1];   // sağa kaydır
7             j--;
8         }
9         a[j] = anahtar;        // yerleş
10    }
11 }
```

DEĞİŞKENLER

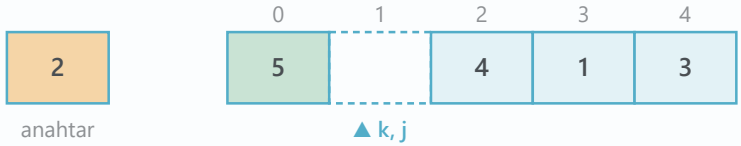
a = [5, 2, 4, 1, 3]

ÇAĞRI YIĞINI

eklemeliSirala(a)

main()

Eklemeli Sıralama (Insertion Sort)



Tur k = 1: `anahtar = a[1] = 2` bir kenara alınır; hücresi artık boşluk sayılır.

Adım 2 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
▶          int anahtar = a[k];
▶          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

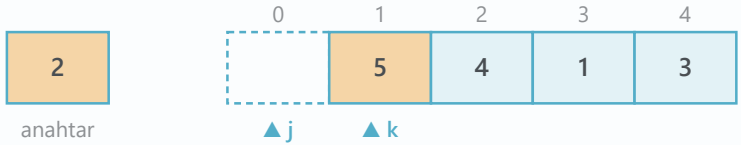
DEĞİŞKENLER

`a = [5, 2, 4, 1, 3]` `k = 1` `anahtar = 2` `j = 1`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[0] = 5 > 2` → 5 bir sağa kayar; boşluk indis 0'a geçer.

Adım 3 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

DEĞİŞKENLER

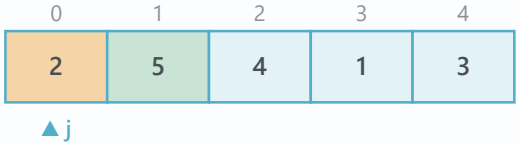
`a = [5, 5, 4, 1, 3]` `k = 1` `anahtar = 2` `j = 0`

ÇAĞRI YIĞINI

`eklemeliSirala(a)`

`main()`

Eklemeli Sıralama (Insertion Sort)



`j == 0` : sıralı kısmın başına gelindi; `a[0] = 2` . İlk 2 öğe artık sıralı.

Adım 4 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
▶          a[j] = anahtar;        // yerleş
10     }
11 }
```

DEĞİŞKENLER

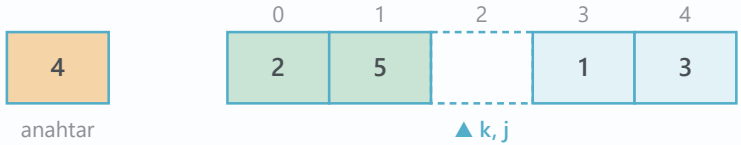
`a = [2, 5, 4, 1, 3]` `k = 1` `anahtar = 2` `j = 0`

ÇAĞRI YIĞINI

`eklemeliSirala(a)`

`main()`

Eklemeli Sıralama (Insertion Sort)



Tur k = 2: `anahtar = a[2] = 4` bir kenara alınır; hücresi artık boşluk sayılır.

Adım 5 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
▶          int anahtar = a[k];
▶          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

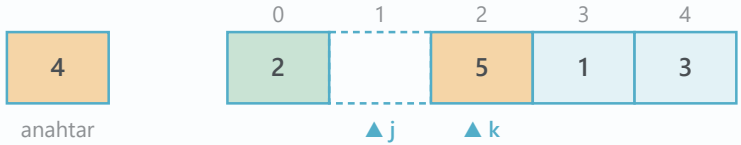
DEĞİŞKENLER

`a = [2, 5, 4, 1, 3]` `k = 2` `anahtar = 4` `j = 2`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[1] = 5 > 4` → 5 bir sağa kayar; boşluk indis 1'e geçer.

Adım 6 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

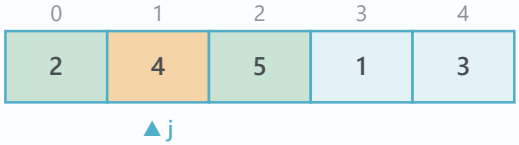
DEĞİŞKENLER

`a = [2, 5, 5, 1, 3]` `k = 2` `anahtar = 4` `j = 1`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[0] = 2 ≤ 4` : durulur; `a[1] = 4` . İlk 3 öğe artık sıralı.

Adım 7 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
▶          a[j] = anahtar;        // yerleş
10     }
11 }
```

DEĞİŞKENLER

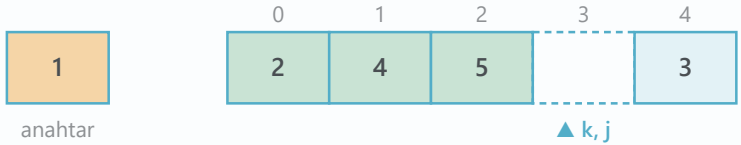
`a = [2, 4, 5, 1, 3]` `k = 2` `anahtar = 4` `j = 1`

ÇAĞRI YIĞINI

`eklemeliSirala(a)`

`main()`

Eklemeli Sıralama (Insertion Sort)



Tur k = 3: `anahtar = a[3] = 1` bir kenara alınır; hücresi artık boşluk sayılır.

Adım 8 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
▶          int anahtar = a[k];
▶          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

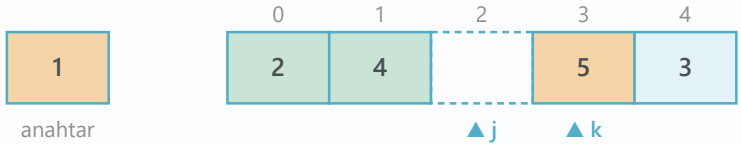
DEĞİŞKENLER

`a = [2, 4, 5, 1, 3]` `k = 3` `anahtar = 1` `j = 3`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[2] = 5 > 1` → 5 bir sağa kayar; boşluk indis 2'ye geçer.

Adım 9 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

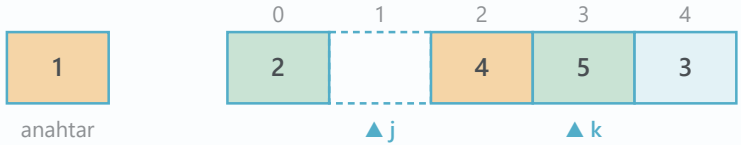
DEĞİŞKENLER

`a = [2, 4, 5, 5, 3]` `k = 3` `anahtar = 1` `j = 2`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[1] = 4 > 1` → 4 bir sağa kayar; boşluk indis 1'e geçer.

Adım 10 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

DEĞİŞKENLER

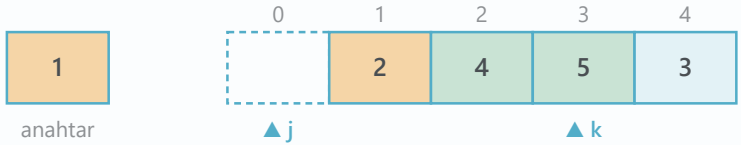
`a = [2, 4, 4, 5, 3]` `k = 3` `anahtar = 1` `j = 1`

ÇAĞRI YIĞINI

`eklemeliSirala(a)`

`main()`

Eklemeli Sıralama (Insertion Sort)



`a[0] = 2 > 1` → 2 bir sağa kayar; boşluk indis 0'a geçer.


```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

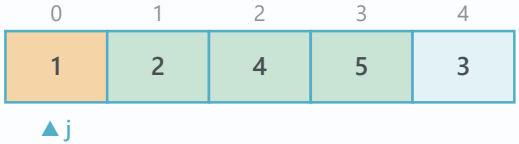
DEĞİŞKENLER

`a = [2, 2, 4, 5, 3]` `k = 3` `anahtar = 1` `j = 0`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`j == 0` : sıralı kısmın başına gelindi; `a[0] = 1` . İlk 4 öğe artık sıralı.

Adım 12 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
▶          a[j] = anahtar;        // yerleş
10     }
11 }
```

DEĞİŞKENLER

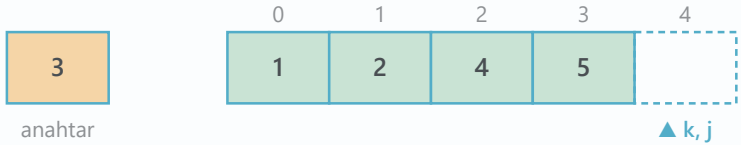
`a = [1, 2, 4, 5, 3]` `k = 3` `anahtar = 1` `j = 0`

ÇAĞRI YIĞINI

`eklemeliSirala(a)`

`main()`

Eklemeli Sıralama (Insertion Sort)



Tur k = 4: `anahtar = a[4] = 3` bir kenara alınır; hücresi artık boşluk sayılır.

Adım 13 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
▶          int anahtar = a[k];
▶          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

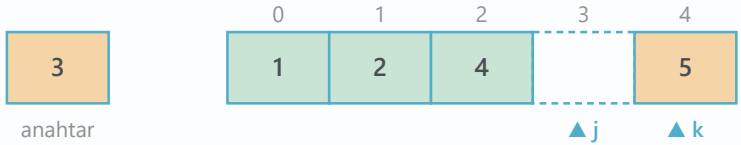
DEĞİŞKENLER

`a = [1, 2, 4, 5, 3]` `k = 4` `anahtar = 3` `j = 4`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[3] = 5 > 3` → 5 bir sağa kayar; boşluk indis 3'e geçer.

Adım 14 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

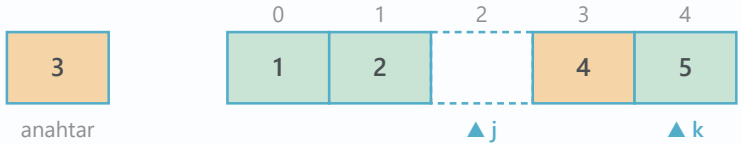
DEĞİŞKENLER

`a = [1, 2, 4, 5, 5]` `k = 4` `anahtar = 3` `j = 3`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[2] = 4 > 3` → 4 bir sağa kayar; boşluk indis 2'ye geçer.

Adım 15 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
▶          while (j > 0 && a[j - 1] > anahtar) {
▶              a[j] = a[j - 1];   // sağa kaydır
▶              j--;
8          }
9          a[j] = anahtar;        // yerleş
10     }
11 }
```

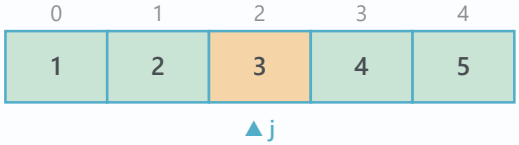
DEĞİŞKENLER

`a = [1, 2, 4, 4, 5]` `k = 4` `anahtar = 3` `j = 2`

ÇAĞRI YIĞINI

- `eklemeliSirala(a)`
- `main()`

Eklemeli Sıralama (Insertion Sort)



`a[1] = 2 ≤ 3` : durulur; `a[2] = 3` . Dizi sıralı. 7 kaydırma = dizideki ters sıralı çift (inversion) sayısı. En kötü $O(n^2)$, zaten sıralı dizide O(n).

Adım 16 / 16

```
1  static void eklemeliSirala(int[] a) {
2      for (int k = 1; k < a.length; k++) {
3          int anahtar = a[k];
4          int j = k;
5          while (j > 0 && a[j - 1] > anahtar) {
6              a[j] = a[j - 1];   // sağa kaydır
7              j--;
8          }
▶          a[j] = anahtar;        // yerleş
10     }
11 }
```

DEĞİŞKENLER

`a = [1, 2, 3, 4, 5]` `k = 4` `anahtar = 3` `j = 2`

ÇAĞRI YIĞINI

`eklemeliSirala(a)`

`main()`

KARMAŞIKLIK

Alan Karmaşıklığı ve Bellek Kullanımı

- Alan karmaşıklığı, bir algoritmanın/veri yapısının bellek ihtiyacının n ile nasıl büyüdüğüdür.
- n öğeli bir dizi **O(n)** alan kaplar; bayt olarak miktar öğe tipine bağlıdır.

Dizi tipi	Öğe başına	100 öğe (yalnızca veri)
`int[]`	4 bayt	400 bayt
`char[]` (UTF-16)	2 bayt	200 bayt
`double[]`, `long[]`	8 bayt	800 bayt
`Object[]` (referans)	4 / 8 bayt	400 / 800 bayt + nesneler

Ek yük

Her dizi nesnesinin ayrıca ~16 baytlık başlığı (tür bilgisi + `length`) vardır.

Referans dizileri

Yalnızca referanslar ardışıktır; gösterilen nesneler heap'te ayrı yer kaplar.

ÖZET

Dizilerle Sık Yapılan Hatalar

Sınır hatası (off-by-one)

`<=` son turda `a[a.length]`'e erişir → istisna.

```
for (i = 0; i <
a.length; i++)
```

`length` / `length()`

Dizide alan, `String`'de metot.

```
a.length · s.length()
```

`=` kopya değildir

`b = a` aynı diziyi gösterir; kopya için `clone` / `copyOf`.

```
int[] b = a.clone();
```

`==` ile içerik

`==` referansları karşılaştırır.

```
Arrays.equals(a, b)
```

Diziyi doğrudan yazdırmak

`println(a)` tür@adres basar (`[I@…`).

```
Arrays.toString(a)
```

Düzensiz dizide null satır

`new int[3][]` sonrası `m[0][0]` → `NullPointerException`.

```
m[i] = new int[k];
```

Ayrıca: sıralanmamış dizide `binarySearch` yanlış sonuç verir; 2B dizide sütun sayısı için her satırda `m[i].length` kullanılmalıdır.

Laboratuvar: Sıralama

Seçimli sıralama: 64, 25, 12, 22, 11

Girdi (virgülle ayrılmış sayılar) 64, 25, 12, 22, 11

KOD · **adım 1 / 38**

```
 1 int[] dizi = {64, 25, 12, 22, 11};
 2 int n = dizi.length;
 3 for (int i = 0; i < n - 1; i++) {
 4     int minIndis = i;
 5     for (int j = i + 1; j < n; j++)
 6         if (dizi[j] < dizi[minIndis])
 7             minIndis = j;
 8     // takas: indislerle, yerinde
 9     int gecici = dizi[minIndis];
10     dizi[minIndis] = dizi[i];
11     dizi[i] = gecici;
12 }
```

dizi | 64 | 25 | 12 | 22 | 11 |

karşılaştırma = 0

ÇIKTI –

dizi 5 öğeyle oluşturulur: {64, 25, 12, 22, 11}.

ÖZET

Özet

Yapı
Aynı türden, sabit boyutlu, ardışık bellek; indis 0 … length − 1.

Java'da
Dizi bir nesnedir, heap'tedir; `=` referansı kopyalar, `Arrays.copyOf` öğeleri.

Çok boyutlu
Dizilerin dizisi; satırlar farklı boyda olabilir. `m[i][j]`.

İşlem	Maliyet	İşlem	Maliyet
Erişim `a[i]`	O(1)	Araya ekleme / silme	O(n)
Doğrusal arama	O(n)	Tersine çevirme	O(n)
Seçimli sıralama	$O(n^2)$	Transpoz (m×n)	O(m·n)

Etkileşimli simülatör: dizide erişim, ekleme, silme ve aramayı adım adım denemek için Dizi Simülatörü.

ÖZET

Örnek Problemler

1. `int[] a = new int[4];` sonrası `a[4]` ifadesi ne olur? `a.length` kaçtır?
2. `int[] b = a; b[0] = 9;` sonrası `a[0]` kaçtır? Neden?
3. n öğeli dizide en büyük öğeyi bulan metodu yazın. Karmaşıklığı nedir?
4. {5, 1, 4, 2, 8} dizisine seçimli sıralamanın ilk iki turundan sonra dizi nasıl görünür?
5. 1000 öğeli `double[]` dizinin veri kısmı kaç bayttır?
6. 3×4 bir matrisin transpozu kaç satır, kaç sütundur? `t[2][1]` hangi öğedir?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `int[] a = {3, 1, 2}; int[] b = a; b = new int[] {9};` sonrası `a[0]` ve `b.length` kaçtır?
2. {7, 3, 5, 1} dizisine eklemeli sıralamada her dış turdan sonra dizi nedir? Kaç kaydırma yapılır?
3. `int[][] m = new int[3][]; m[1] = new int[4];` sonrası `m.length`, `m[1].length`, `m[0]` nedir?
4. Yeri olan sıralı diziye, sırayı bozmadan öğe eklemenin en kötü maliyeti nedir?
5. `sifrele("XYZ", 2)` ne döndürür? Çözmek için `k` kaç olmalı?
6. Diziyi yerinde bir adım sola döndüren (`a[0]` sona gider) metodu yazın. Maliyeti?

▶ **Cevaplar / ipuçları**

Kaynaklar

- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014.
- Lafore, *Data Structures and Algorithms in Java*, 2. baskı, 2002.
- Oracle, Java SE 17 `Arrays` ve `System` API belgeleri.

Son

Bölüm 3: Diziler

Sonraki sunum: Bölüm 4: Özyineleme · Çalışma soruları · Simülatör: Dizi Simülatörü · Terim sözlüğü