Bölüm 12: Özyineleme

Bilgisayar ve Programlamaya Giriş

Taban durum · Çağrı · Yığın · Verimlilik · Bellekleme

Sercan KÜLCÜ · Giresun Üniversitesi

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/computer/slides/Bolum_12_Ozyineleme.html

GİRİŞ

Öğrenme Çıktıları

Yapı

Taban durum ve özyinelemeli durumu belirlemek.

İzleme

Çağrı yığınını adım adım takip etmek.

Karşılaştırma

Özyineleme ile döngüyü değerlendirmek.

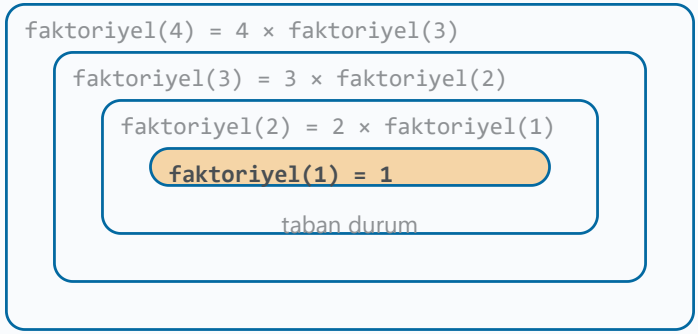
İyileştirme

Tekrarlı hesaplamayı bellekleme (memoization) ile azaltmak.

TEMELLER

Özyineleme Nedir?

- Bir metodun problemi **daha küçük bir örneğine** indirip kendisini çağırmasıdır.
- Her çağrı kendi parametre ve yerel değişkenleriyle ayrı bir **yığın çerçevesi** oluşturur.
- Problem her çağrıda **taban duruma** yaklaşmalıdır; yoksa çağrılar sonlanmaz.



İki Zorunlu Parça

Taban durum

Sonucun doğrudan bilindiği, yeni çağrı yapılmayan durum (ör. n ≤ 1 → 1).

Özyinelemeli durum

Problemi daha küçük örneğe dönüştüren çağrı (ör. n × f(n − 1)).



ÖRNEKLER

Özyinelemeli Faktöriyel

```java
static long faktoriyel(int n) {
    if (n < 0)
        throw new IllegalArgumentException();
    if (n <= 1) return 1;
    return n * faktoriyel(n - 1);
}
```

Tanım

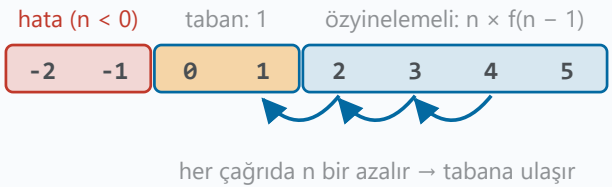
n! = n × (n − 1)!, 0! = 1! = 1

Taban durum

n ≤ 1 → 1; her çağrıda n bir azalır.

Negatif n

Faktöriyel tanımsız → hata fırlatılır.



Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

main, `faktoriyel(4)` 'ü çağırır; sonucu bekler.

Adım 1 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
5     return n * faktoriyel(n - 1);
6 }
▶ System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

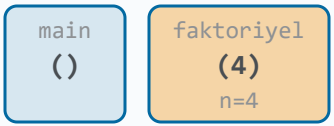
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

Yeni çerçeve: **n = 4**. **4 < 0** → **false**, **4 <= 1** → **false**.

Adım 2 / 12

```
1 static long faktoriyel(int n) {
▶     if (n < 0)
3         throw new IllegalArgumentException();
▶     if (n <= 1) return 1;
5     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

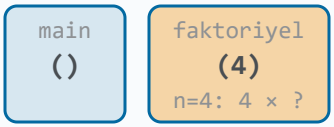
`n = 4`

ÇAĞRI YIĞINI

`faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`4 * faktoriyel(3)` : çarpma için önce alt sonuç gerekli → bu çerçeve **bekler**, yeni çağrı yapılır.

Adım 3 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
▶     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

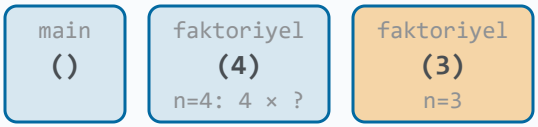
`n = 4`

ÇAĞRI YIĞINI

`faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

Yeni çerçeve: **n = 3**. **3 < 0** → **false**, **3 <= 1** → **false**.


```
1 static long faktoriyel(int n) {
▶     if (n < 0)
3         throw new IllegalArgumentException();
▶     if (n <= 1) return 1;
5     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

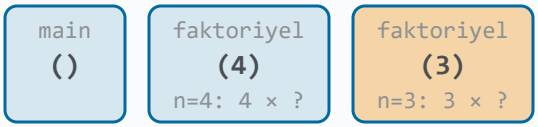
`n = 3`

ÇAĞRI YIĞINI

`faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`3 * faktoriyel(2)` : çarpma için önce alt sonuç gerekli → bu çerçeve **bekler**, yeni çağrı yapılır.

Adım 5 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
▶     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

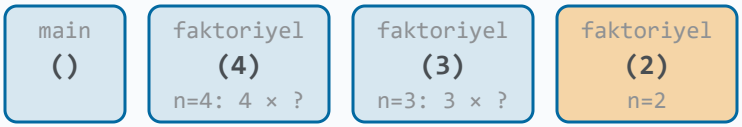
`n = 3`

ÇAĞRI YIĞINI

`faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

Yeni çerçeve: **n = 2**. **2 < 0** → **false**, **2 <= 1** → **false**.

Adım 6 / 12

```
1 static long faktoriyel(int n) {
▶     if (n < 0)
3         throw new IllegalArgumentException();
▶     if (n <= 1) return 1;
5     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

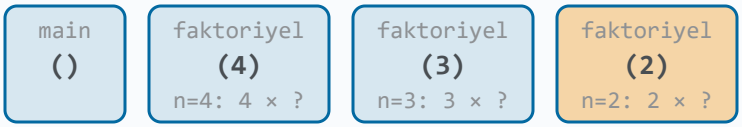
`n = 2`

ÇAĞRI YIĞINI

`faktoriyel(2)` `faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`2 * faktoriyel(1)` : çarpma için önce alt sonuç gerekli → bu çerçeve **bekler**, yeni çağrı yapılır.

Adım 7 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
▶     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

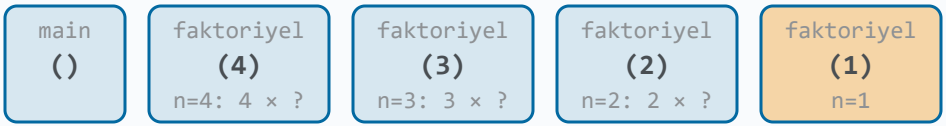
`n = 2`

ÇAĞRI YIĞINI

`faktoriyel(2)` `faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

Yeni çerçeve: **n = 1**. **1 < 0** → **false**, **1 <= 1** → **true**: **taban durum**, return 1.

Adım 8 / 12

```
1 static long faktoriyel(int n) {
▶     if (n < 0)
3         throw new IllegalArgumentException();
▶     if (n <= 1) return 1;
5     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

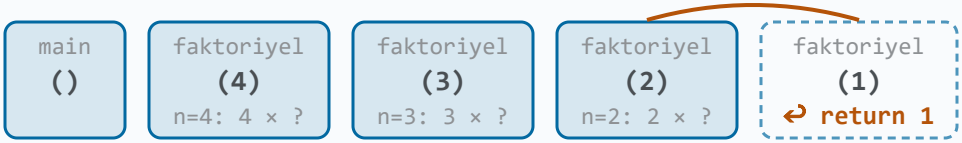
`n = 1`

ÇAĞRI YIĞINI

`faktoriyel(1)` `faktoriyel(2)` `faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`faktoriyel(1)` **1** döndürdü; çerçevesi silindi. Şimdi **2 × 1 = 2**.

Adım 9 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
▶     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

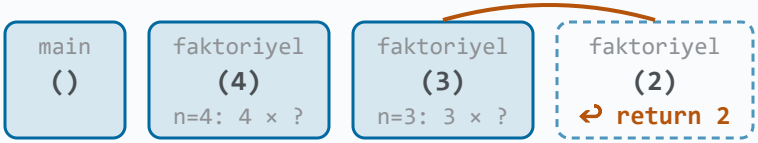
`n = 2`

ÇAĞRI YIĞINI

`faktoriyel(2)` `faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`faktoriyel(2)` **2** döndürdü; çerçevesi silindi. Şimdi **3 × 2 = 6**.

Adım 10 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
▶     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

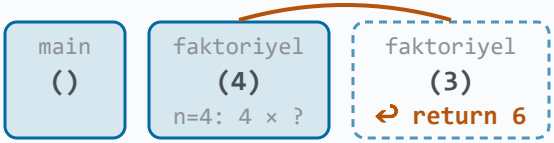
`n = 3`

ÇAĞRI YIĞINI

`faktoriyel(3)` `faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`faktoriyel(3)` **6** döndürdü; çerçevesi silindi. Şimdi **4 × 6 = 24**.

Adım 11 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
▶     return n * faktoriyel(n - 1);
6 }
7 System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

`n = 4`

ÇAĞRI YIĞINI

`faktoriyel(4)` `main()`

ÇIKTI

Kod İzleme: `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

En üstteki çerçeve de kapandı; main **24** yazdırır. En fazla 5 çerçeve açıktı → ek bellek O(n).

Adım 12 / 12

```
1 static long faktoriyel(int n) {
2     if (n < 0)
3         throw new IllegalArgumentException();
4     if (n <= 1) return 1;
5     return n * faktoriyel(n - 1);
6 }
▶ System.out.println(faktoriyel(4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

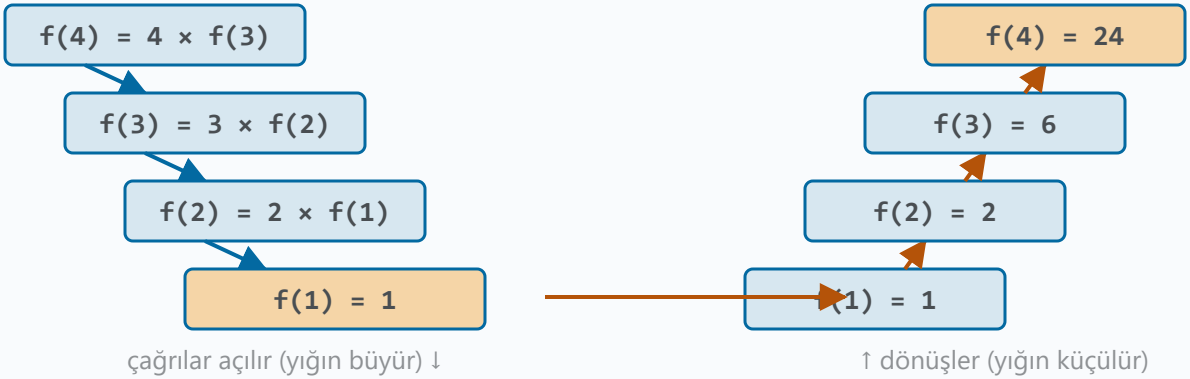
`main()`

ÇIKTI

```
24
```

ÇAĞRI YIĞINI

faktoriyel(4): Açılım ve Dönüş



- Çağrılar taban duruma ulaşıncaya kadar yığında açılır; hiçbir çarpma henüz yapılmamıştır.
- f(1) = 1 döndükten sonra bekleyen çarpımlar **ters sırada** tamamlanır: 2, 6, 24.

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

Tanım: toplam(n) = n + toplam(n − 1), toplam(0) = 0. main, `toplam(3)` 'ü çağırır.

Adım 1 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
3     return n + toplam(n - 1);
4 }
▶ System.out.println(toplam(3));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

3 <= 0 → false.

Adım 2 / 12

```
1 static int toplam(int n) {
▶       if (n <= 0) return 0;
3       return n + toplam(n - 1);
4   }
5   System.out.println(toplam(3));
```

DEĞİŞKENLER

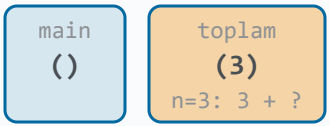
`n = 3`

ÇAĞRI YIĞINI

`toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

`3 + toplam(2)` : toplama bekler, yeni çağrı.

Adım 3 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
▶     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

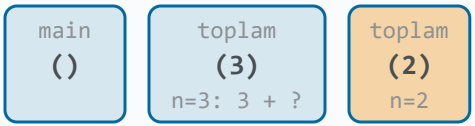
`n = 3`

ÇAĞRI YIĞINI

`toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

2 <= 0 → false.

Adım 4 / 12

```
1 static int toplam(int n) {
▶     if (n <= 0) return 0;
3     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

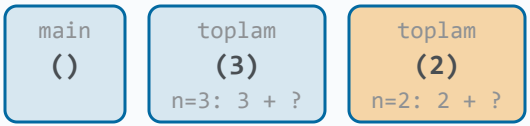
`n = 2`

ÇAĞRI YIĞINI

`toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

`2 + toplam(1)` : toplama bekler, yeni çağrı.

Adım 5 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
▶     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

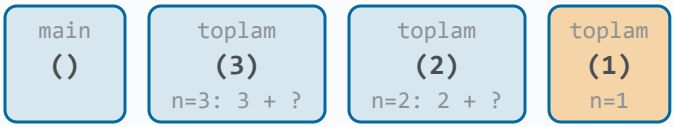
`n = 2`

ÇAĞRI YIĞINI

`toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

1 <= 0 → false.

Adım 6 / 12

```
1 static int toplam(int n) {
▶     if (n <= 0) return 0;
3     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

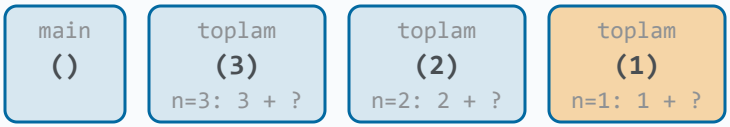
`n = 1`

ÇAĞRI YIĞINI

`toplam(1)` `toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

`1 + toplam(0)` : toplama bekler, yeni çağrı.

Adım 7 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
▶     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

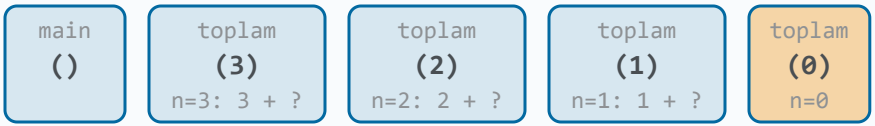
`n = 1`

ÇAĞRI YIĞINI

`toplam(1)` `toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

0 <= 0 → **true**: taban durum, **return 0**.

Adım 8 / 12

```
1 static int toplam(int n) {
▶     if (n <= 0) return 0;
3     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

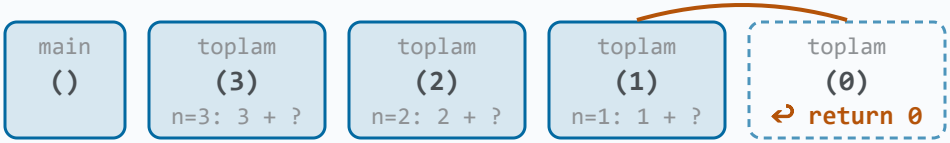
`n = 0`

ÇAĞRI YIĞINI

`toplam(0)` `toplam(1)` `toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

`toplam(0)` = 0 döndü → **1 + 0 = 1**.

Adım 9 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
▶     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

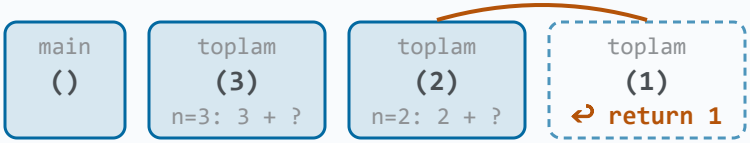
`n = 1`

ÇAĞRI YIĞINI

`toplam(1)` `toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

`toplam(1)` = 1 döndü → **2 + 1 = 3**.

Adım 10 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
▶     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

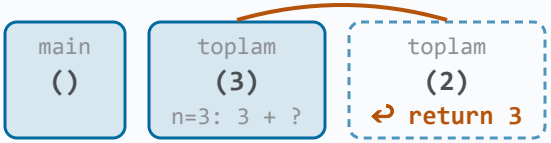
`n = 2`

ÇAĞRI YIĞINI

`toplam(2)` `toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

`toplam(2)` = 3 döndü → **3 + 3 = 6**.

Adım 11 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
▶     return n + toplam(n - 1);
4 }
5 System.out.println(toplam(3));
```

DEĞİŞKENLER

`n = 3`

ÇAĞRI YIĞINI

`toplam(3)` `main()`

ÇIKTI

1'den n'ye Toplam: `toplam(3)`



main (yığının dibi) → yığın bu yönde büyür → üst

6 = 3 + 2 + 1 + 0 yazdırılır.

Adım 12 / 12

```
1 static int toplam(int n) {
2     if (n <= 0) return 0;
3     return n + toplam(n - 1);
4 }
▶ System.out.println(toplam(3));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
6
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

main, `rakamToplami(472)` 'yi çağırır.

Adım 1 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
▶  System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

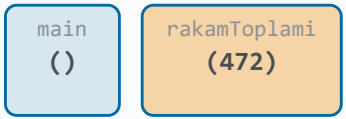
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

Negatif sayılar için mutlak değer alınır: **m = 472** (long'a çevirmek en küçük int'te taşmayı önler).

Adım 2 / 12

```
1  static int rakamToplami(int n) {
▶      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

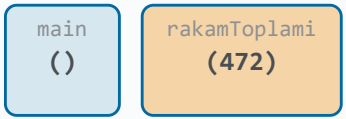
`n = 472` `m = 472`

ÇAĞRI YIĞINI

`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

Asıl işi yapan yardımcı metot çağrılır.

Adım 3 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
▶      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 472` `m = 472`

ÇAĞRI YIĞINI

`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

472 < 10 → **false**: son rakam 2, kalan 47.

Adım 4 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
▶      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 472`

ÇAĞRI YIĞINI

`rakamToplamiPozitif(472)`
`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

`472 % 10 = 2` bekletilir; `472 / 10 = 47` ile yeni çağrı.

Adım 5 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
▶      return (int) (n % 10)
▶          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 472`

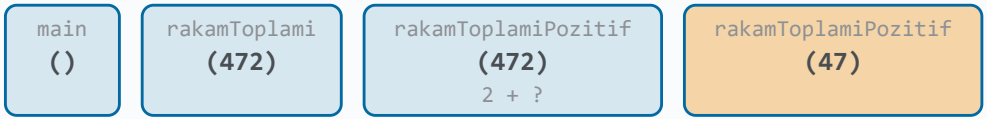
ÇAĞRI YIĞINI

`rakamToplamiPozitif(472)`

`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

47 < 10 → **false**: son rakam 7, kalan 4.

Adım 6 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
▶      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

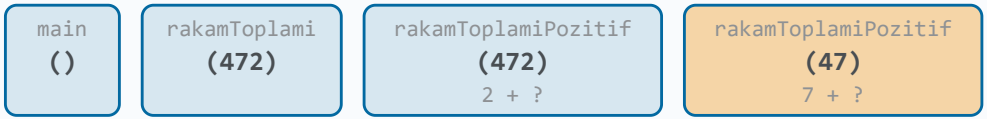
`n = 47`

ÇAĞRI YIĞINI

`rakamToplamiPozitif(47)`
`rakamToplamiPozitif(472)`
`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

`47 % 10 = 7` bekletilir; `47 / 10 = 4` ile yeni çağrı.

Adım 7 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
▶      return (int) (n % 10)
▶          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 47`

ÇAĞRI YIĞINI

`rakamToplamiPozitif(47)`
`rakamToplamiPozitif(472)`
`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

4 < 10 → true: tek basamak, **return 4**.

Adım 8 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
▶      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 4`

ÇAĞRI YIĞINI

`rakamToplamiPozitif(4)`
`rakamToplamiPozitif(47)`
`rakamToplamiPozitif(472)`
`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

Alt çağrı 4 döndü → **7 + 4 = 11**.

Adım 9 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
▶      return (int) (n % 10)
▶          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

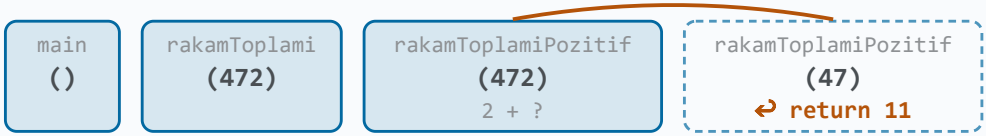
`n = 47`

ÇAĞRI YIĞINI

`rakamToplamiPozitif(47)`
`rakamToplamiPozitif(472)`
`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

Alt çağrı 11 döndü → **2 + 11 = 13**.

Adım 10 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
▶      return (int) (n % 10)
▶          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 472`

ÇAĞRI YIĞINI

`rakamToplamiPozitif(472)`
`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

Yardımcı metot **13** döndürdü; aynen döndürülür.

Adım 11 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
▶      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
10 System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

`n = 472` `m = 472`

ÇAĞRI YIĞINI

`rakamToplami(472)` `main()`

ÇIKTI

Rakamlar Toplamı: `rakamToplami(472)`



main (yığının dibi) → yığın bu yönde büyür → üst

4 + 7 + 2 = **13** yazdırılır.

Adım 12 / 12

```
1  static int rakamToplami(int n) {
2      long m = Math.abs((long) n);
3      return rakamToplamiPozitif(m);
4  }
5  static int rakamToplamiPozitif(long n) {
6      if (n < 10) return (int) n;
7      return (int) (n % 10)
8          + rakamToplamiPozitif(n / 10);
9  }
▶  System.out.println(rakamToplami(472));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
13
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

main, `palindrom("kayak", 0, 4)` 'ü çağırır.

Adım 1 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
▶ System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

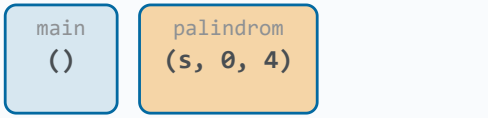
—

ÇAĞRI YIĞINI

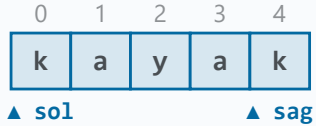
`main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



0 >= 4 → false.

Adım 2 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
▶     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

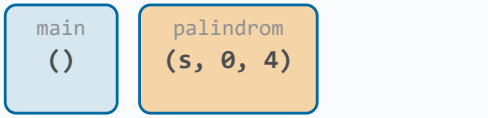
`s = "kayak"` `sol = 0` `sag = 4`

ÇAĞRI YIĞINI

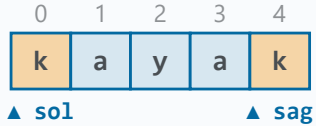
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



`'k' != 'k'` → **false**: uçlar eşit, içe doğru devam.

Adım 3 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
▶     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

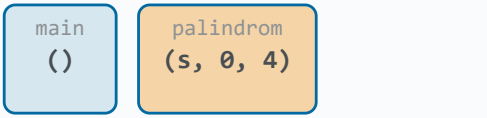
`s = "kayak"` `sol = 0` `sag = 4`

ÇAĞRI YIĞINI

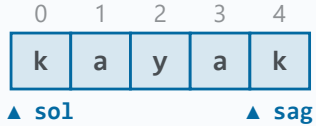
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



İki uç atılır: `palindrom(s, 1, 3)` çağrılır.

Adım 4 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
▶     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

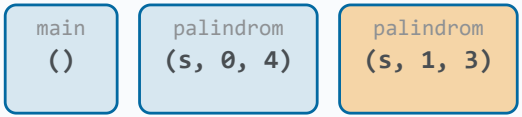
`s = "kayak"` `sol = 0` `sag = 4`

ÇAĞRI YIĞINI

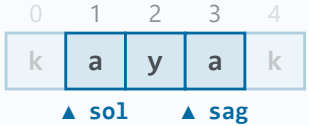
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



1 >= 3 → false.

Adım 5 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
▶     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

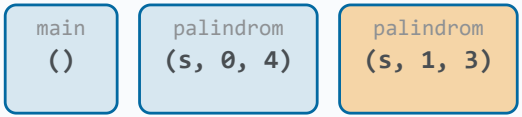
`s = "kayak"` `sol = 1` `sag = 3`

ÇAĞRI YIĞINI

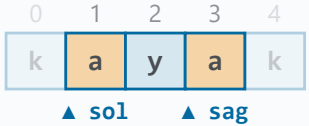
`palindrom(s, 1, 3)`
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



`'a' != 'a'` → **false**: uçlar eşit, içe doğru devam.

Adım 6 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

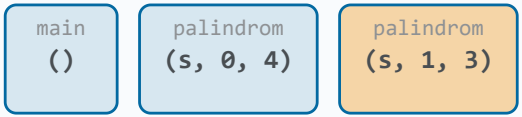
`s = "kayak"` `sol = 1` `sag = 3`

ÇAĞRI YIĞINI

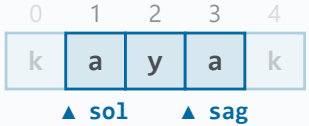
`palindrom(s, 1, 3)`
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



İki uç atılır: `palindrom(s, 2, 2)` çağrılır.

Adım 7 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
▶     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

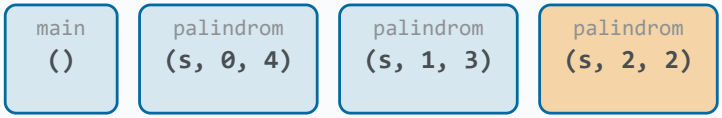
`s = "kayak"` `sol = 1` `sag = 3`

ÇAĞRI YIĞINI

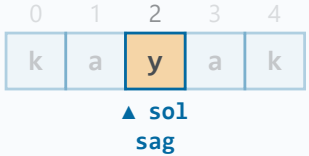
`palindrom(s, 1, 3)`
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



2 >= 2 → **true**: ortada tek karakter kaldı → **return true**.

Adım 8 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
▶     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

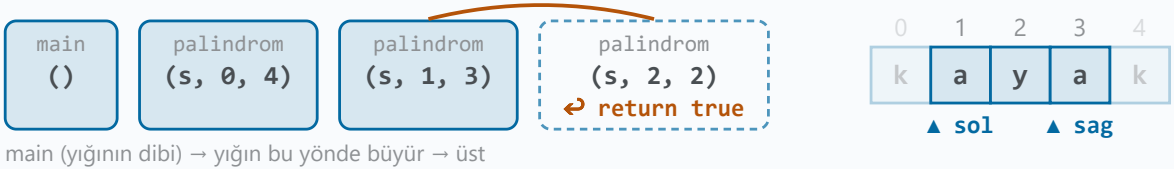
`s = "kayak"` `sol = 2` `sag = 2`

ÇAĞRI YIĞINI

`palindrom(s, 2, 2)`
`palindrom(s, 1, 3)`
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



Alt çağrı **true** döndürdü; bu çerçeve de aynı değeri döndürür.

Adım 9 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
▶     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

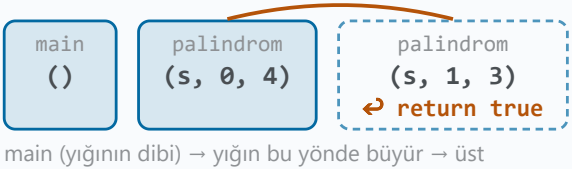
`s = "kayak"` `sol = 1` `sag = 3`

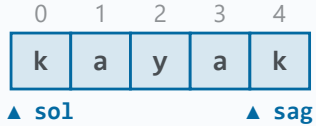
ÇAĞRI YIĞINI

`palindrom(s, 1, 3)`
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`





Alt çağrı **true** döndürdü; bu çerçeve de aynı değeri döndürür.

Adım 10 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
▶     return palindrom(s, sol + 1, sag - 1);
7 }
8 System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

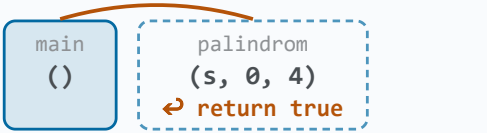
`s = "kayak"` `sol = 0` `sag = 4`

ÇAĞRI YIĞINI

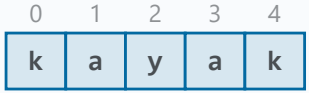
`palindrom(s, 0, 4)` `main()`

ÇIKTI

Palindrom: `palindrom("kayak", 0, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst



Sonuç main'e ulaştı: **true**. "kayak" tersten de aynı okunur.

Adım 11 / 11

```
1 static boolean palindrom(String s,
2                          int sol, int sag) {
3     if (sol >= sag) return true;
4     if (s.charAt(sol) != s.charAt(sag))
5         return false;
6     return palindrom(s, sol + 1, sag - 1);
7 }
▶ System.out.println(palindrom("kayak", 0, 4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
true
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

main, `ebob(48, 18)` 'i çağırır.

Adım 1 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
3     return ebob(b, a % b);
4 }
▶ System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

18 == 0 → false.

Adım 2 / 12

```
1 static int ebob(int a, int b) {
▶     if (b == 0) return Math.abs(a);
3     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

`a = 48` `b = 18`

ÇAĞRI YIĞINI

`ebob(48, 18)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

`48 % 18 = 12` → `ebob(18, 12)` çağrılır (sayılar küçülüyor).

Adım 3 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
▶     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

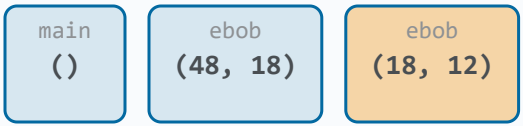
`a = 48` `b = 18`

ÇAĞRI YIĞINI

`ebob(48, 18)` `main()`

ÇIKTI

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

12 == 0 → false.

Adım 4 / 12

```
1 static int ebob(int a, int b) {
▶     if (b == 0) return Math.abs(a);
3     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

`a = 18` `b = 12`

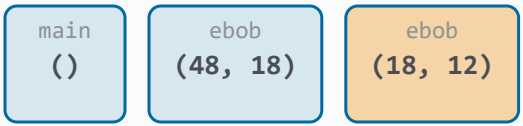
ÇAĞRI YIĞINI

`ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

`18 % 12 = 6` → `ebob(12, 6)` çağrılır (sayılar küçülüyor).

Adım 5 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
▶     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

`a = 18` `b = 12`

ÇAĞRI YIĞINI

`ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

6 == 0 → false.

Adım 6 / 12

```
1 static int ebob(int a, int b) {
▶     if (b == 0) return Math.abs(a);
3     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

`a = 12` `b = 6`

ÇAĞRI YIĞINI

`ebob(12, 6)` `ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

`12 % 6 = 0` → `ebob(6, 0)` çağrılır (sayılar küçülüyor).

Adım 7 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
▶     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

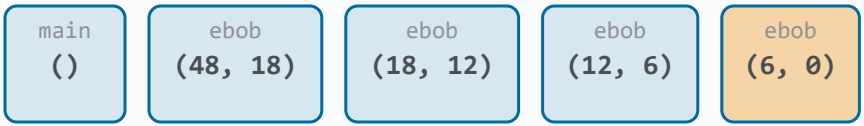
`a = 12` `b = 6`

ÇAĞRI YIĞINI

`ebob(12, 6)` `ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

0 == 0 → **true**: taban durum, **return 6**.

Adım 8 / 12

```
1 static int ebob(int a, int b) {
▶     if (b == 0) return Math.abs(a);
3     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

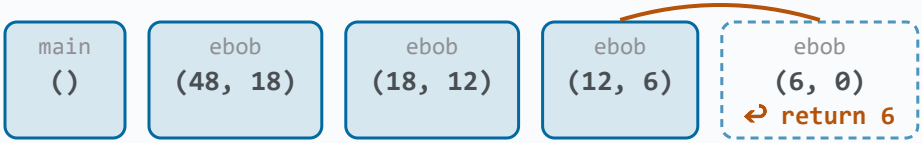
`a = 6` `b = 0`

ÇAĞRI YIĞINI

`ebob(6, 0)` `ebob(12, 6)` `ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

Alt çağrı **6** döndürdü; aynen döndürülür.

Adım 9 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
▶     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

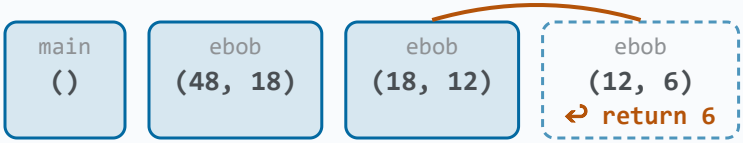
`a = 12` `b = 6`

ÇAĞRI YIĞINI

`ebob(12, 6)` `ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

Alt çağrı **6** döndürdü; aynen döndürülür.

Adım 10 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
▶     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

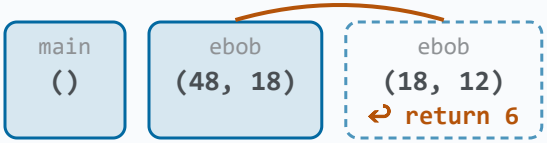
`a = 18` `b = 12`

ÇAĞRI YIĞINI

`ebob(18, 12)` `ebob(48, 18)` `main()`

ÇIKTI

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

Alt çağrı **6** döndürdü; aynen döndürülür.

Adım 11 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
▶     return ebob(b, a % b);
4 }
5 System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

`a = 48` `b = 18`

ÇAĞRI YIĞINI

`ebob(48, 18)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Öklid ile EBOB: `ebob(48, 18)`



main (yığının dibi) → yığın bu yönde büyür → üst

EBOB(48, 18) = **6**: kalanlar küçülerek 0'a ulaştı; son sıfır olmayan kalan EBOB'dur.

Adım 12 / 12

```
1 static int ebob(int a, int b) {
2     if (b == 0) return Math.abs(a);
3     return ebob(b, a % b);
4 }
▶ System.out.println(ebob(48, 18));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
6
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 0

3 disk A'dan C'ye taşınacak; B yardımcı. Kural: büyük disk küçüğün üstüne konamaz.

Adım 1 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
5   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
▶ hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

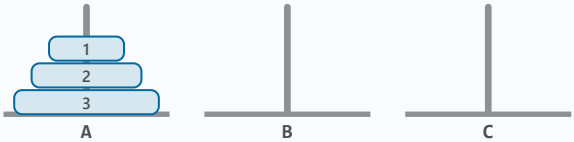
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 0

`hanoi(3, A, C, B)` : önce üstteki 2 disk A → B (yardımcıya) taşınmalı.

Adım 2 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
▶   hanoi(n - 1, kaynak, ara, hedef);
5   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

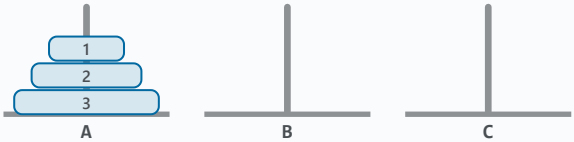
`n = 3` `kaynak = A` `hedef = C` `ara = B`

ÇAĞRI YIĞINI

`hanoi(3,A,C,B)` `main()`

ÇIKTI

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 0

`hanoi(2, A, B, C)` : önce üstteki 1 disk A → C (yardımcıya) taşınmalı.

Adım 3 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
▶   hanoi(n - 1, kaynak, ara, hedef);
5   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

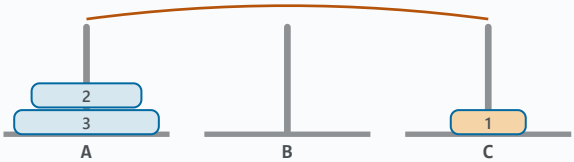
`n = 2` `kaynak = A` `hedef = B` `ara = C`

ÇAĞRI YIĞINI

`hanoi(2,A,B,C)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 1

Hamle 1: disk **1** A → C. (hanoi(0) çağrıları hemen döner)

Adım 4 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 1` `kaynak = A` `hedef = C` `ara = B`

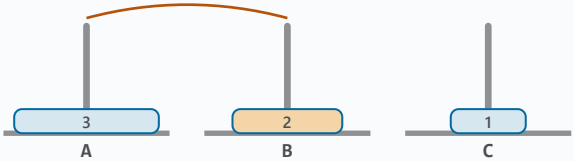
ÇAĞRI YIĞINI

`hanoi(1,A,C,B)` `hanoi(2,A,B,C)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 2

Hamle 2: disk **2** A → B.

Adım 5 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 2` `kaynak = A` `hedef = B` `ara = C`

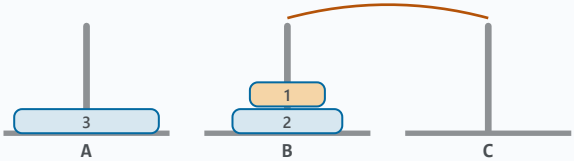
ÇAĞRI YIĞINI

`hanoi(2,A,B,C)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 3

Hamle 3: disk **1** C → B.

Adım 6 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 1` `kaynak = C` `hedef = B` `ara = A`

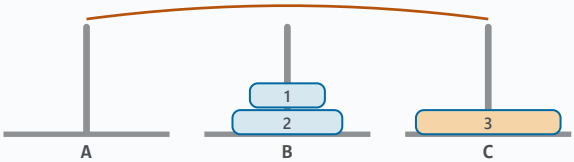
ÇAĞRI YIĞINI

`hanoi(1,C,B,A)` `hanoi(2,A,B,C)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B C->B
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 4

Hamle 4: disk **3** A → C.

Adım 7 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 3` `kaynak = A` `hedef = C` `ara = B`

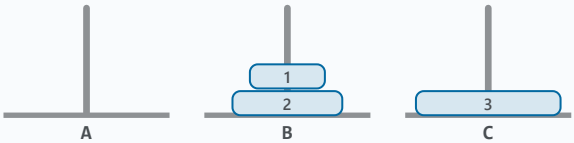
ÇAĞRI YIĞINI

`hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B C->B A->C
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 4

`hanoi(2, B, C, A)` : önce üstteki 1 disk B → A (yardımcıya) taşınmalı.

Adım 8 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
▶   hanoi(n - 1, kaynak, ara, hedef);
5   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 2` `kaynak = B` `hedef = C` `ara = A`

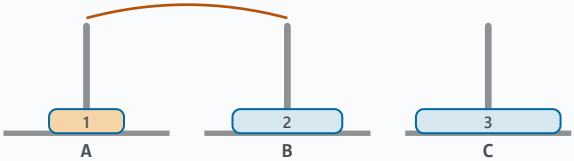
ÇAĞRI YIĞINI

`hanoi(2,B,C,A)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B C->B A->C
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 5

Hamle 5: disk **1** B → A.

Adım 9 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 1` `kaynak = B` `hedef = A` `ara = C`

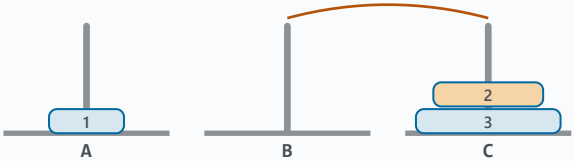
ÇAĞRI YIĞINI

`hanoi(1,B,A,C)` `hanoi(2,B,C,A)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B C->B A->C B->A
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 6

Hamle 6: disk **2** B → C.

Adım 10 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 2` `kaynak = B` `hedef = C` `ara = A`

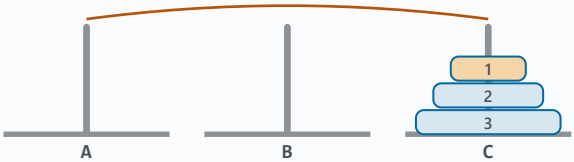
ÇAĞRI YIĞINI

`hanoi(2,B,C,A)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B C->B A->C B->A B->C
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 7

Hamle 7: disk **1** A → C.

Adım 11 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
▶   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
8 hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

`n = 1` `kaynak = A` `hedef = C` `ara = B`

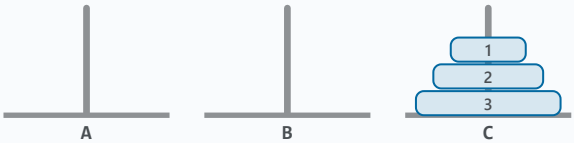
ÇAĞRI YIĞINI

`hanoi(1,A,C,B)` `hanoi(2,B,C,A)` `hanoi(3,A,C,B)` `main()`

ÇIKTI

```
A->C A->B C->B A->C B->A B->C A->C
```

Hanoi Kuleleri: `hanoi(3, 'A', 'C', 'B')`



hamle: 7

Bitti: **7 = 2^3 − 1** hamle. n disk için 2^n − 1 hamle gerekir (üstel).

Adım 12 / 12

```
1 static void hanoi(int n, char kaynak,
2                   char hedef, char ara) {
3   if (n == 0) return;
4   hanoi(n - 1, kaynak, ara, hedef);
5   System.out.print(kaynak + "->" + hedef + " ");
6   hanoi(n - 1, ara, hedef, kaynak);
7 }
▶ hanoi(3, 'A', 'C', 'B');
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
A->C A->B C->B A->C B->A B->C A->C
```

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

$2^4 = 2 \times 2^3$ tanımı: us(taban, n) = taban × us(taban, n − 1), us(taban, 0) = 1.

Adım 1 / 11

```
1 static long us(long taban, int n) {
2   if (n == 0) return 1;
3   return taban * us(taban, n - 1);
4 }
▶ System.out.println(us(2, 4));
```

DEĞİŞKENLER

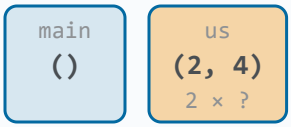
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

4 == 0 → **false**: 2 × us(2, 3) için yeni çağrı.


```
1 static long us(long taban, int n) {
▶   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

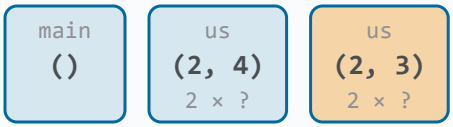
`taban = 2` `n = 4`

ÇAĞRI YIĞINI

`us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

3 == 0 → **false**: 2 × us(2, 2) için yeni çağrı.

Adım 3 / 11

```
1 static long us(long taban, int n) {
▶   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

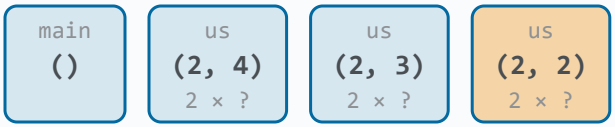
`taban = 2` `n = 3`

ÇAĞRI YIĞINI

`us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

2 == 0 → **false**: 2 × us(2, 1) için yeni çağrı.

Adım 4 / 11

```
1 static long us(long taban, int n) {
▶   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

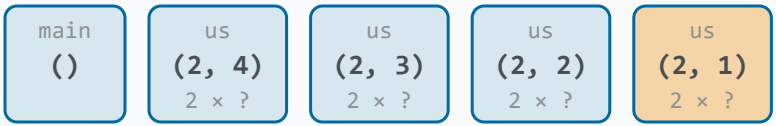
`taban = 2` `n = 2`

ÇAĞRI YIĞINI

`us(2, 2)` `us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

1 == 0 → false: 2 × us(2, 0) için yeni çağrı.

Adım 5 / 11

```
1 static long us(long taban, int n) {
▶   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

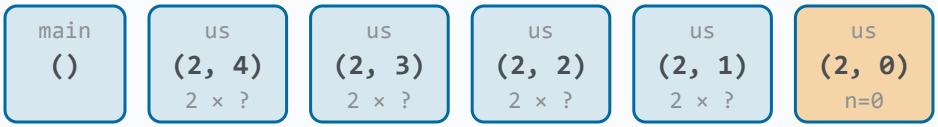
`taban = 2` `n = 1`

ÇAĞRI YIĞINI

`us(2, 1)` `us(2, 2)` `us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

0 == 0 → **true**: taban durum, return 1.

Adım 6 / 11

```
1 static long us(long taban, int n) {
▶   if (n == 0) return 1;
3   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

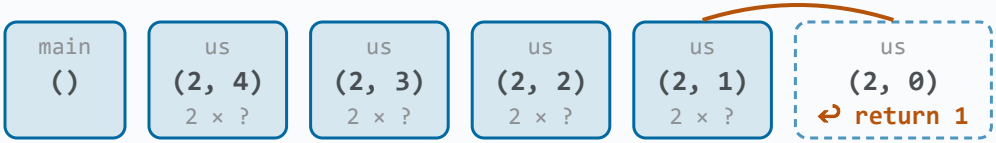
`taban = 2` `n = 0`

ÇAĞRI YIĞINI

`us(2, 0)` `us(2, 1)` `us(2, 2)` `us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

us(2, 0) = 1 döndü → **2 × 1 = 2**.

Adım 7 / 11

```
1 static long us(long taban, int n) {
2   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

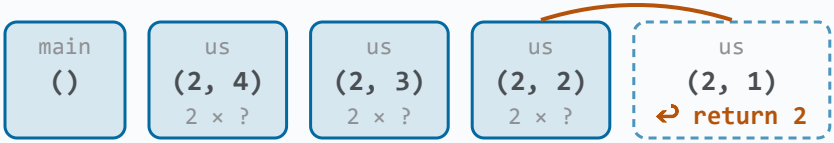
`taban = 2` `n = 1`

ÇAĞRI YIĞINI

`us(2, 1)` `us(2, 2)` `us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

us(2, 1) = 2 döndü → **2 × 2 = 4**.

Adım 8 / 11

```
1 static long us(long taban, int n) {
2   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

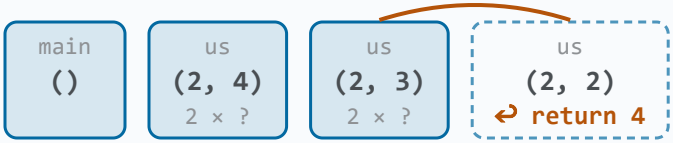
`taban = 2` `n = 2`

ÇAĞRI YIĞINI

`us(2, 2)` `us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

us(2, 2) = 4 döndü → **2 × 4 = 8**.

Adım 9 / 11

```
1 static long us(long taban, int n) {
2   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

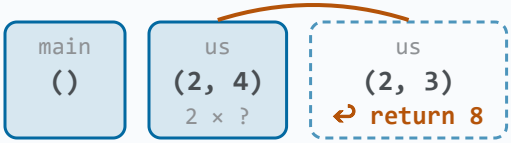
`taban = 2` `n = 3`

ÇAĞRI YIĞINI

`us(2, 3)` `us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

us(2, 3) = 8 döndü → **2 × 8 = 16**.

Adım 10 / 11

```
1 static long us(long taban, int n) {
2   if (n == 0) return 1;
▶   return taban * us(taban, n - 1);
4 }
5 System.out.println(us(2, 4));
```

DEĞİŞKENLER

`taban = 2` `n = 4`

ÇAĞRI YIĞINI

`us(2, 4)` `main()`

ÇIKTI

Özyinelemeli Üs: `us(2, 4)`



main (yığının dibi) → yığın bu yönde büyür → üst

16: n + 1 = 5 çağrı → O(n).

Adım 11 / 11

```
1 static long us(long taban, int n) {
2   if (n == 0) return 1;
3   return taban * us(taban, n - 1);
4 }
▶ System.out.println(us(2, 4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
16
```

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

Fikir: $2^{10} = (2^5)^2$, $2^5 = (2^2)^2 \times 2$. Üs her çağrıda **yarıya** iner.

Adım 1 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
▶ System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

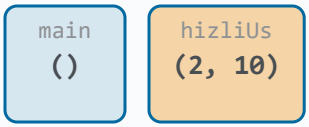
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

us = 10 ≠ 0 → önce `hizliUs(2, 5)` (10 / 2 = 5).


```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
▶   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

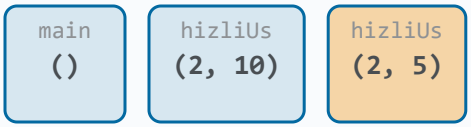
`taban = 2` `us = 10`

ÇAĞRI YIĞINI

`hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

us = 5 ≠ 0 → önce `hizliUs(2, 2)` (5 / 2 = 2).

Adım 3 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
▶   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

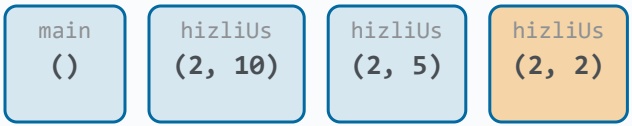
`taban = 2` `us = 5`

ÇAĞRI YIĞINI

`hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

us = 2 ≠ 0 → önce `hizliUs(2, 1)` (2 / 2 = 1).

Adım 4 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
▶   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

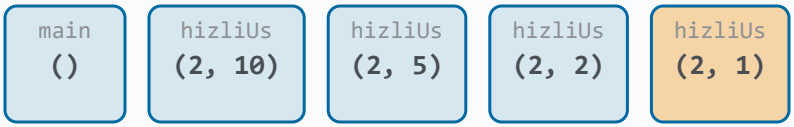
`taban = 2` `us = 2`

ÇAĞRI YIĞINI

`hizliUs(2, 2)` `hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

us = 1 ≠ 0 → önce `hizliUs(2, 0)` (1 / 2 = 0).

Adım 5 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
▶   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

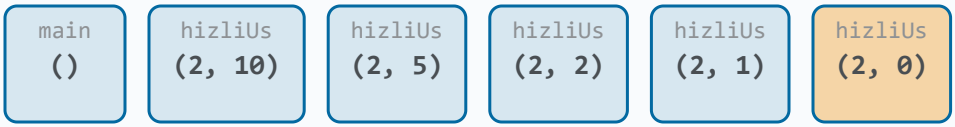
`taban = 2` `us = 1`

ÇAĞRI YIĞINI

`hizliUs(2, 1)` `hizliUs(2, 2)` `hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

us == 0: return 1.

Adım 6 / 11

```
1 static long hizliUs(long taban, int us) {
▶   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

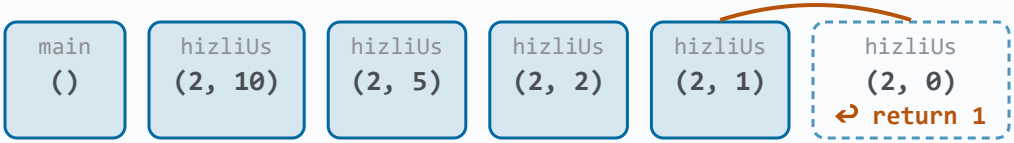
`taban = 2` `us = 0`

ÇAĞRI YIĞINI

`hizliUs(2, 0)` `hizliUs(2, 1)` `hizliUs(2, 2)` `hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

yarim = 1; 1 tek → **1 × 1 × 2 = 2**.

Adım 7 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
▶   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

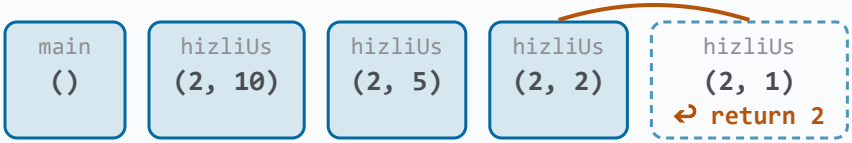
`taban = 2` `us = 1` `yarim = 1`

ÇAĞRI YIĞINI

`hizliUs(2, 1)` `hizliUs(2, 2)` `hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

yarim = 2; 2 çift → **2 × 2 = 4**.

Adım 8 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
▶   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

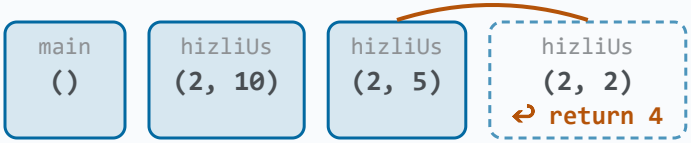
`taban = 2` `us = 2` `yarim = 2`

ÇAĞRI YIĞINI

`hizliUs(2, 2)` `hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

yarim = 4; 5 tek → **4 × 4 × 2 = 32**.

Adım 9 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
▶   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

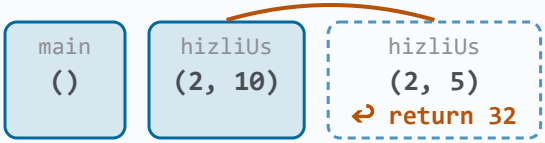
`taban = 2` `us = 5` `yarim = 4`

ÇAĞRI YIĞINI

`hizliUs(2, 5)` `hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

yarim = 32; 10 çift → **32 × 32 = 1024**.

Adım 10 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
▶   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
7 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

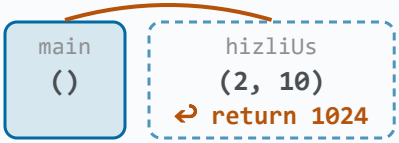
`taban = 2` `us = 10` `yarim = 32`

ÇAĞRI YIĞINI

`hizliUs(2, 10)` `main()`

ÇIKTI

Hızlı Üs: `hizliUs(2, 10)`



main (yığının dibi) → yığın bu yönde büyür → üst

1024, yalnızca 5 çağrıyla (düz yöntem 11 çağrı yapardı) → **O(log n)**.

Adım 11 / 11

```
1 static long hizliUs(long taban, int us) {
2   if (us == 0) return 1;
3   long yarim = hizliUs(taban, us / 2);
4   if (us % 2 == 0) return yarim * yarim;
5   return yarim * yarim * taban;
6 }
▶ System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
1024
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



Döngü yerine özyineleme: her çağrı aralığın yarısıyla kendini çağırır.

Adım 1 / 8

```
1 static int ara(int[] a, int x,
2                int lo, int hi) {
3   if (lo > hi) return -1;
4   int mid = (lo + hi) / 2;
5   if (a[mid] == x) return mid;
6   if (a[mid] < x) return ara(a, x, mid + 1, hi);
7   return ara(a, x, lo, mid - 1);
8 }
9 // a = {2, 5, 8, 12, 16, 23, 38, 56}
▶ System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

x = 38

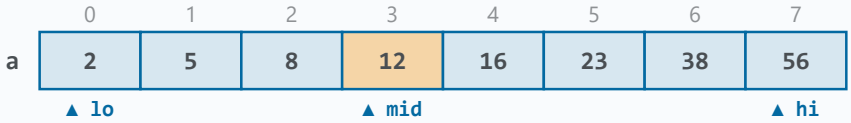
ÇAĞRI YIĞINI

main()

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



0 <= 7; **mid = 3**, a[mid] = 12.

Adım 2 / 8

```
1  static int ara(int[] a, int x,
2                 int lo, int hi) {
▶    if (lo > hi) return -1;
▶    int mid = (lo + hi) / 2;
5    if (a[mid] == x) return mid;
6    if (a[mid] < x) return ara(a, x, mid + 1, hi);
7    return ara(a, x, lo, mid - 1);
8  }
9  // a = {2, 5, 8, 12, 16, 23, 38, 56}
10 System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

`x = 38` `lo = 0` `hi = 7` `mid = 3`

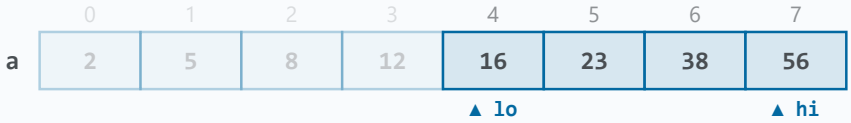
ÇAĞRI YIĞINI

`ara(a, 38, 0, 7)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



12 < 38 → sağ yarı: `ara(a, 38, 4, 7)`.

Adım 3 / 8

```
1  static int ara(int[] a, int x,
2                 int lo, int hi) {
3    if (lo > hi) return -1;
4    int mid = (lo + hi) / 2;
5    if (a[mid] == x) return mid;
▶    if (a[mid] < x) return ara(a, x, mid + 1, hi);
7    return ara(a, x, lo, mid - 1);
8  }
9  // a = {2, 5, 8, 12, 16, 23, 38, 56}
10 System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

`x = 38` `lo = 0` `hi = 7` `mid = 3`

ÇAĞRI YIĞINI

`ara(a, 38, 0, 7)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



4 <= 7; **mid = 5**, a[mid] = 23.

Adım 4 / 8

```
1  static int ara(int[] a, int x,
2                 int lo, int hi) {
3    if (lo > hi) return -1;
4    int mid = (lo + hi) / 2;
5    if (a[mid] == x) return mid;
6    if (a[mid] < x) return ara(a, x, mid + 1, hi);
7    return ara(a, x, lo, mid - 1);
8  }
9  // a = {2, 5, 8, 12, 16, 23, 38, 56}
10 System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

`x = 38` `lo = 4` `hi = 7` `mid = 5`

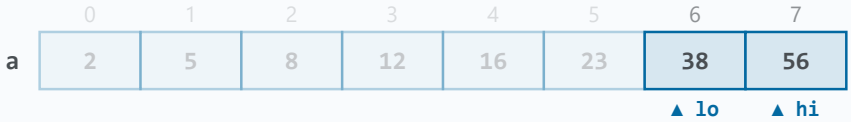
ÇAĞRI YIĞINI

`ara(a, 38, 4, 7)` `ara(a, 38, 0, 7)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



23 < 38 → sağ yarı: `ara(a, 38, 6, 7)`.

Adım 5 / 8

```
1  static int ara(int[] a, int x,
2                 int lo, int hi) {
3    if (lo > hi) return -1;
4    int mid = (lo + hi) / 2;
5    if (a[mid] == x) return mid;
▶    if (a[mid] < x) return ara(a, x, mid + 1, hi);
7    return ara(a, x, lo, mid - 1);
8  }
9  // a = {2, 5, 8, 12, 16, 23, 38, 56}
10 System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

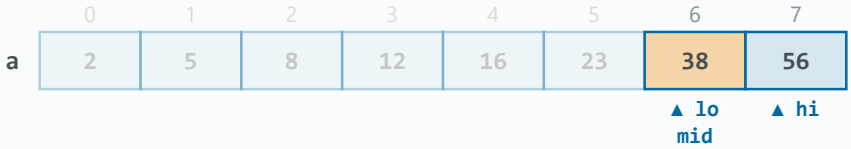
`x = 38` `lo = 4` `hi = 7` `mid = 5`

ÇAĞRI YIĞINI

`ara(a, 38, 4, 7)` `ara(a, 38, 0, 7)` `main()`

ÇIKTI

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



6 <= 7; **mid = 6**, a[mid] = 38.

Adım 6 / 8

```
1  static int ara(int[] a, int x,
2                 int lo, int hi) {
▶    if (lo > hi) return -1;
▶    int mid = (lo + hi) / 2;
5    if (a[mid] == x) return mid;
6    if (a[mid] < x) return ara(a, x, mid + 1, hi);
7    return ara(a, x, lo, mid - 1);
8  }
9  // a = {2, 5, 8, 12, 16, 23, 38, 56}
10 System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

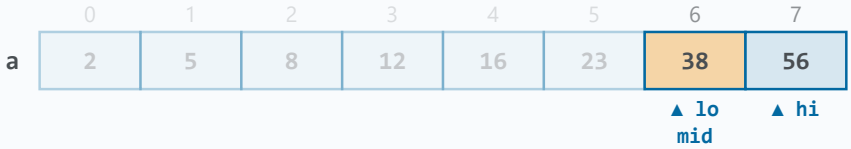
`x = 38` `lo = 6` `hi = 7` `mid = 6`

ÇAĞRI YIĞINI

`ara(a, 38, 6, 7)` `ara(a, 38, 4, 7)` `ara(a, 38, 0, 7)` `main()`

ÇIKTI

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



38 == 38 → return 6 (taban durum).

Adım 7 / 8

```
1  static int ara(int[] a, int x,
2                 int lo, int hi) {
3    if (lo > hi) return -1;
4    int mid = (lo + hi) / 2;
▶    if (a[mid] == x) return mid;
6    if (a[mid] < x) return ara(a, x, mid + 1, hi);
7    return ara(a, x, lo, mid - 1);
8  }
9  // a = {2, 5, 8, 12, 16, 23, 38, 56}
10 System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

`x = 38` `lo = 6` `hi = 7` `mid = 6`

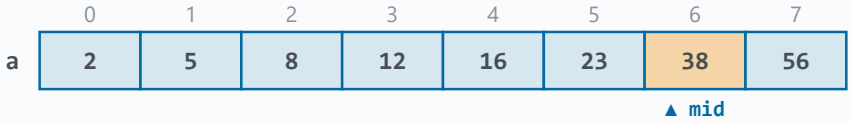
ÇAĞRI YIĞINI

`ara(a, 38, 6, 7)` `ara(a, 38, 4, 7)` `ara(a, 38, 0, 7)` `main()`

ÇIKTI

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Özyinelemeli İkili Arama: `ara(a, 38, 0, 7)`



Sonuç çerçevelerden aynen geri geçer: **6**.

Adım 8 / 8

```
1 static int ara(int[] a, int x,
2                int lo, int hi) {
3   if (lo > hi) return -1;
4   int mid = (lo + hi) / 2;
5   if (a[mid] == x) return mid;
6   if (a[mid] < x) return ara(a, x, mid + 1, hi);
7   return ara(a, x, lo, mid - 1);
8 }
9 // a = {2, 5, 8, 12, 16, 23, 38, 56}
▶ System.out.println(ara(a, 38, 0, 7));
```

DEĞİŞKENLER

`x = 38`

ÇAĞRI YIĞINI

`main()`

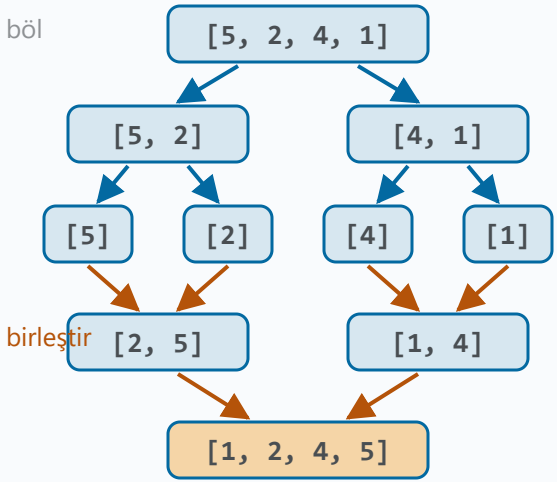
ÇIKTI

```
6
```

ÖRNEKLER

Böl ve Fethet: Birleştirme Sıralaması

```java
static void sirala(int[] a, int lo, int hi) {
    if (lo >= hi) return;
    int mid = (lo + hi) / 2;
    sirala(a, lo, mid);
    sirala(a, mid + 1, hi);
    int[] t = new int[hi - lo + 1];
    int i = lo, j = mid + 1, k = 0;
    while (i <= mid && j <= hi)
        t[k++] = a[i] <= a[j] ? a[i++] : a[j++];
    while (i <= mid) t[k++] = a[i++];
    while (j <= hi) t[k++] = a[j++];
    System.arraycopy(t, 0, a, lo, t.length);
}
```



gri ok: böl · renkli ok: birleştir

Her düzeyde toplam n öğe birleştirilir, düzey sayısı $\log_2 n$: zaman O(n log n), ek bellek O(n).

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

ters(s) = ters(ilk karakter hariç kalan) + ilk karakter.

Adım 1 / 7

```
1 static String ters(String s) {
2   if (s.length() <= 1) return s;
3   return ters(s.substring(1)) + s.charAt(0);
4 }
▶ System.out.println(ters("abc"));
```

DEĞİŞKENLER

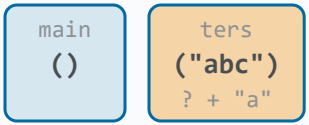
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

`ters("bc")` çağrılır; 'a' sona eklenmek üzere bekler.

Adım 2 / 7

```
1 static String ters(String s) {
2   if (s.length() <= 1) return s;
▶   return ters(s.substring(1)) + s.charAt(0);
4 }
5 System.out.println(ters("abc"));
```

DEĞİŞKENLER

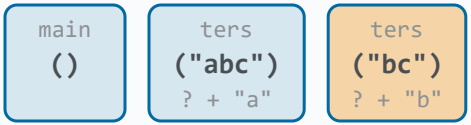
`s = "abc"`

ÇAĞRI YIĞINI

`ters("abc")` `main()`

ÇIKTI

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

`ters("c")` çağrılır; 'b' sona eklenmek üzere bekler.

Adım 3 / 7

```
1 static String ters(String s) {
2   if (s.length() <= 1) return s;
▶   return ters(s.substring(1)) + s.charAt(0);
4 }
5 System.out.println(ters("abc"));
```

DEĞİŞKENLER

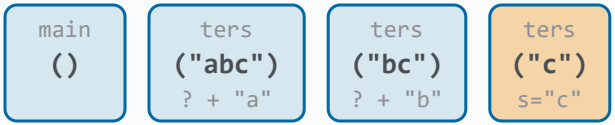
`s = "bc"`

ÇAĞRI YIĞINI

`ters("bc")` `ters("abc")` `main()`

ÇIKTI

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

Uzunluk 1 ≤ 1 → taban durum, "c" döner.

Adım 4 / 7

```
1 static String ters(String s) {
▶   if (s.length() <= 1) return s;
3   return ters(s.substring(1)) + s.charAt(0);
4 }
5 System.out.println(ters("abc"));
```

DEĞİŞKENLER

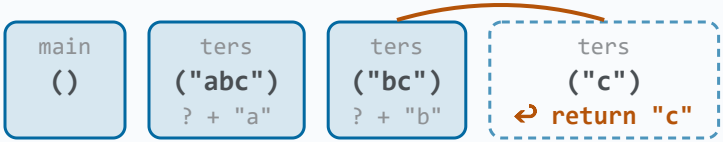
`s = "c"`

ÇAĞRI YIĞINI

`ters("c")` `ters("bc")` `ters("abc")` `main()`

ÇIKTI

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

"c" + 'b' = **"cb"**.

Adım 5 / 7

```
1 static String ters(String s) {
2   if (s.length() <= 1) return s;
▶   return ters(s.substring(1)) + s.charAt(0);
4 }
5 System.out.println(ters("abc"));
```

DEĞİŞKENLER

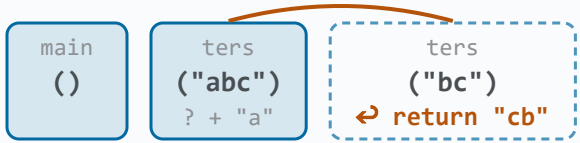
`s = "bc"`

ÇAĞRI YIĞINI

`ters("bc")` `ters("abc")` `main()`

ÇIKTI

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

"cb" + 'a' = **"cba"**.

Adım 6 / 7

```
1 static String ters(String s) {
2   if (s.length() <= 1) return s;
▶   return ters(s.substring(1)) + s.charAt(0);
4 }
5 System.out.println(ters("abc"));
```

DEĞİŞKENLER

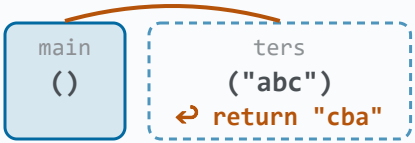
`s = "abc"`

ÇAĞRI YIĞINI

`ters("abc")` `main()`

ÇIKTI

Dizgiyi Özyinelemeli Ters Çevirme



main (yığının dibi) → yığın bu yönde büyür → üst

cba yazdırılır. Her çağrı `substring` ile yeni bir alt dizgi üretir.

Adım 7 / 7

```
1 static String ters(String s) {
2   if (s.length() <= 1) return s;
3   return ters(s.substring(1)) + s.charAt(0);
4 }
▶ System.out.println(ters("abc"));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
cba
```

ÖRNEKLER · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

main, boş bir liste ile `altKume(a, 0, sec)` 'yi çağırır. Her öğe için iki seçim var: dahil et / hariç tut.

Adım 1 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
▶ altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

i = 0: dizi bitmedi (0 ≠ 2). Önce **1 dahil** dalı.

Adım 2 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 0` `sec = []`

ÇAĞRI YIĞINI

`altKume(0)` `main()`

ÇIKTI

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

`sec.add(1)` : liste artık [1].

Adım 3 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
▶      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 0` `sec = [1]`

ÇAĞRI YIĞINI

`altKume(0)` `main()`

ÇIKTI

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

Yeni çerçeve **i = 1**: 2 için karar; önce dahil et.


```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 1` `sec = [1]`

ÇAĞRI YIĞINI

`altKume(1)` `altKume(0)` `main()`

ÇIKTI

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

`sec.add(2)` : liste [1, 2].

Adım 5 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
▶      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 1` `sec = [1, 2]`

ÇAĞRI YIĞINI

`altKume(1)` `altKume(0)` `main()`

ÇIKTI

Kod İzleme: Alt Küme Üretimi



i = 2 = a.length: taban durum. Alt küme yazdırılır: **[1, 2]**.

Adım 6 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
▶          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 2` `sec = [1, 2]`

ÇAĞRI YIĞINI

`altKume(2)` `altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

Geri dönüş: `sec.remove(...)` 2'yi çıkarır (**geri izleme**) → [1].

Adım 7 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
▶      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 1` `sec = [1]`

ÇAĞRI YIĞINI

`altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

2'yi **hariç tut** dalı yaprağa ulaşır: **[1]**.


```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
▶          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 2` `sec = [1]`

ÇAĞRI YIĞINI

`altKume(2)` `altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

altKume(i=1) bitti; `remove` 1'i çıkarır → [].


```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
▶      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 0` `sec = []`

ÇAĞRI YIĞINI

`altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

1'i **hariç tut** dalı: yeni çerçeve i = 1.

Adım 10 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 1` `sec = []`

ÇAĞRI YIĞINI

`altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

`sec.add(2)` : liste [2].

Adım 11 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
▶      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 1` `sec = [2]`

ÇAĞRI YIĞINI

`altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

Yaprak: **[2]** yazdırılır.

Adım 12 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
▶          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 2` `sec = [2]`

ÇAĞRI YIĞINI

`altKume(2)` `altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
[2]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

`remove` 2'yi çıkarır → [].

Adım 13 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
▶      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 1` `sec = []`

ÇAĞRI YIĞINI

`altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
[2]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

Son yaprak: boş küme **[]**.

Adım 14 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
▶      if (i == a.length) {
▶          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
12 altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

`i = 2` `sec = []`

ÇAĞRI YIĞINI

`altKume(2)` `altKume(1)` `altKume(0)` `main()`

ÇIKTI

```
[1, 2]
[1]
[2]
[]
```

Kod İzleme: Alt Küme Üretimi



main (yığının dibi) → yığın bu yönde büyür → üst

Tüm çerçeveler kapandı. 2 öğe → 2^2 = **4 alt küme**; çıktı sırası: [1, 2], [1], [2], [].

Adım 15 / 15

```
1  static void altKume(int[] a, int i,
2                      List<Integer> sec) {
3      if (i == a.length) {
4          System.out.println(sec);
5          return;
6      }
7      sec.add(a[i]);
8      altKume(a, i + 1, sec);
9      sec.remove(sec.size() - 1);
10     altKume(a, i + 1, sec);
11 }
▶ altKume(new int[]{1, 2}, 0, new ArrayList<>());
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

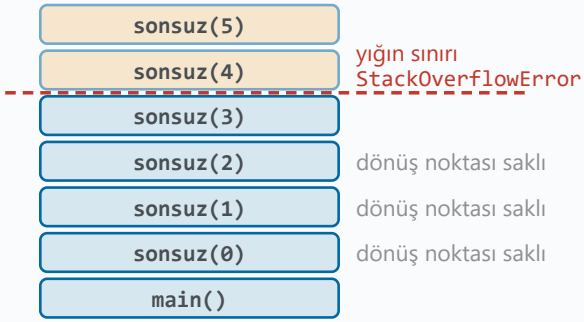
`main()`

ÇIKTI

```
[1, 2]
[1]
[2]
[]
```

Çağrı Yığını

- Yeni çağrı yapıldığında mevcut metodun **dönüş noktası** ve yerel değişkenleri çerçevesinde saklanır.
- Taban durumdan sonra çerçeveler **ters sırada** (son açılan ilk) kapanır.
- Taban durum yoksa (sonsuz özyineleme) ya da çok derin özyinelemede yığın dolar ve yığın taşması olur: `StackOverflowError`.



ÇAĞRI YIĞINI · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: `StackOverflowError`



main (yığının dibi) → yığın bu yönde büyür → üst

main, `sonsuz(0)` 'ı çağırır.

Adım 1 / 6

```
1 static void sonsuz(int n) {
2     sonsuz(n + 1);
3 }
▶ sonsuz(0);
```

DEĞİŞKENLER

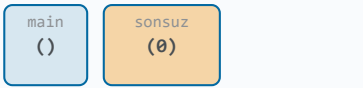
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `StackOverflowError`



main (yığının dibi) → yığın bu yönde büyür → üst

İlk çerçeve. Metotta **taban durum yok**: hiçbir koşul çağrıyı durdurmuyor, hemen `sonsuz(1)` çağrılır.

Adım 2 / 6

```
1 static void sonsuz(int n) {
▶     sonsuz(n + 1);
3 }
4 sonsuz(0);
```

DEĞİŞKENLER

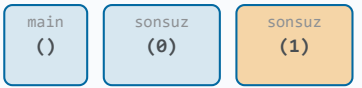
`n = 0`

ÇAĞRI YIĞINI

`sonsuz(0)` `main()`

ÇIKTI

Kod İzleme: `StackOverflowError`



main (yığının dibi) → yığın bu yönde büyür → üst

İkinci çerçeve; önceki çerçeve hâlâ yığında.

Adım 3 / 6

```
1 static void sonsuz(int n) {
▶     sonsuz(n + 1);
3 }
4 sonsuz(0);
```

DEĞİŞKENLER

`n = 1`

ÇAĞRI YIĞINI

`sonsuz(1)` `sonsuz(0)` `main()`

ÇIKTI

Kod İzleme: `StackOverflowError`



main (yığının dibi) → yığın bu yönde büyür → üst

Üçüncü çerçeve. Hiçbir çağrı dönmediği için hiçbir çerçeve silinmez.

Adım 4 / 6

```
1 static void sonsuz(int n) {
▶     sonsuz(n + 1);
3 }
4 sonsuz(0);
```

DEĞİŞKENLER

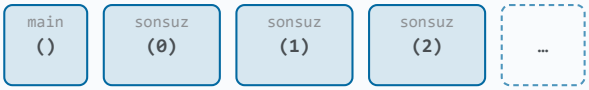
`n = 2`

ÇAĞRI YIĞINI

`sonsuz(2)` `sonsuz(1)` `sonsuz(0)` `main()`

ÇIKTI

Kod İzleme: `StackOverflowError`



main (yığının dibi) → yığın bu yönde büyür → üst

Bu böyle sürer; yığın belleği **sınırlıdır** (varsayılan boyutta genellikle on binlerce çağrı).

Adım 5 / 6

```
1 static void sonsuz(int n) {
▶     sonsuz(n + 1);
3 }
4 sonsuz(0);
```

DEĞİŞKENLER

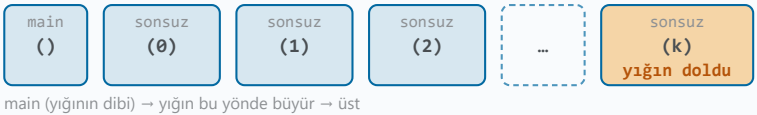
`n = …`

ÇAĞRI YIĞINI

`…` `sonsuz(2)` `sonsuz(1)` `sonsuz(0)` `main()`

ÇIKTI

Kod İzleme: `StackOverflowError`



main (yığının dibi) → yığın bu yönde büyür → üst

Yığın doldu: JVM `StackOverflowError` fırlatır, program biter; hata metni **stderr**'e yazılır.

Adım 6 / 6

```
1 static void sonsuz(int n) {
▶     sonsuz(n + 1);
3 }
4 sonsuz(0);
```

DEĞİŞKENLER

`n = …`

ÇAĞRI YIĞINI

`sonsuz(k)` `…` `sonsuz(2)` `sonsuz(1)` `sonsuz(0)` `main()`

ÇIKTI

```
Exception in thread "main"
java.lang.StackOverflowError
    at Main.sonsuz(Main.java:2)
    at Main.sonsuz(Main.java:2)
    ...
```

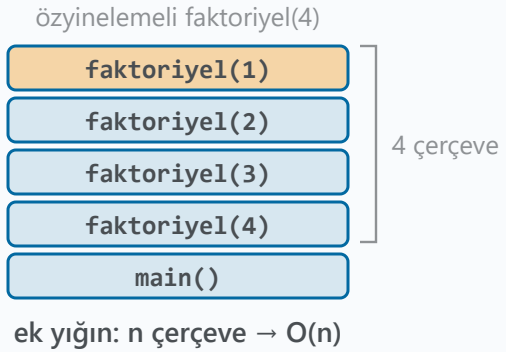
Özyineleme mi Döngü mü?

- Kendine benzeyen problemlerde (ağaçlar, böl-parçala) özyineleme tanıma yakın ve açıktır.
- Basit doğrusal tekrarlarda döngü tek çerçevede çalışır, daha az yığın belleği kullanır.
- Seçim yalnızca kod kısalığına değil; doğruluk, okunabilirlik ve kaynak kullanımına dayanmalıdır.

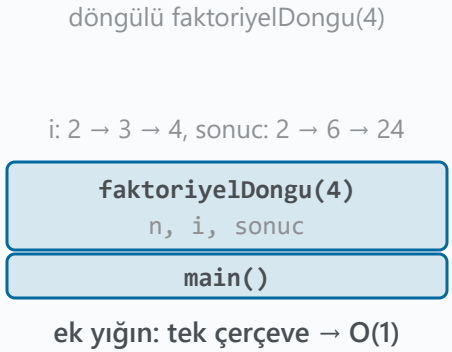


Faktöriyel Karşılaştırması

Özellik	Özyinelemeli	Döngülü
Zaman karmaşıklığı	O(n)	O(n)
Ek yığın	O(n)	O(1)
Akış	Tanıma yakın	Açık sayaç
Derinlik riski	Var	Yok



ek yığın: n çerçeve → O(n)



ek yığın: tek çerçeve → O(1)

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

main, `faktoriyelDongu(4)` 'ü çağırır.

Adım 1 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
3     for (int i = 2; i <= n; i++)
4         sonuc *= i;
5     return sonuc;
6 }
▶ System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

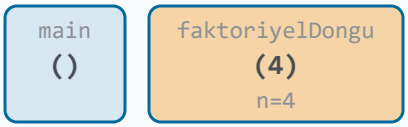
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

Tek çerçeve açılır; **sonuc = 1**.


```
1 static long faktoriyelDongu(int n) {
▶       long sonuc = 1;
3       for (int i = 2; i <= n; i++)
4           sonuc *= i;
5       return sonuc;
6   }
7   System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

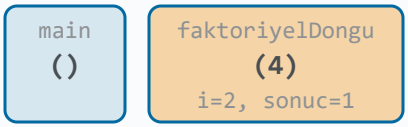
`n = 4` `sonuc = 1`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

2 <= 4 → true

Adım 3 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
▶     for (int i = 2; i <= n; i++)
4         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

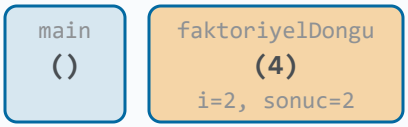
`n = 4` `sonuc = 1` `i = 2`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`sonuc *= 2` → 1 × 2 = **2**. Yeni çerçeve açılmadı.

Adım 4 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
3     for (int i = 2; i <= n; i++)
▶         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

`n = 4` `sonuc = 2` `i = 2`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`

main
()

faktoriyelDongu
(4)
i=3, sonuc=2

main (yığının dibi) → yığın bu yönde büyür → üst

`i++` → **3 <= 4 → true**

Adım 5 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
▶     for (int i = 2; i <= n; i++)
4         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

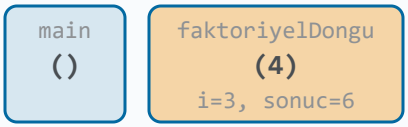
`n = 4` `sonuc = 2` `i = 3`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`sonuc *= 3` → 2 × 3 = **6**. Yeni çerçeve açılmadı.

Adım 6 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
3     for (int i = 2; i <= n; i++)
▶         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

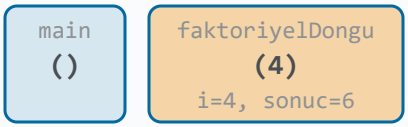
`n = 4` `sonuc = 6` `i = 3`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`i++` → **4 <= 4 → true**

Adım 7 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
▶     for (int i = 2; i <= n; i++)
4         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

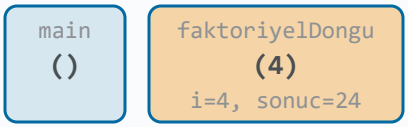
`n = 4` `sonuc = 6` `i = 4`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`sonuc *= 4` → 6 × 4 = **24**. Yeni çerçeve açılmadı.


```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
3     for (int i = 2; i <= n; i++)
▶         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

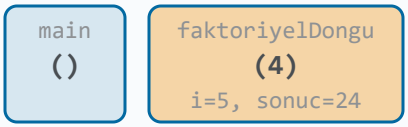
`n = 4` `sonuc = 24` `i = 4`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

`i++` → **5 <= 4** → **false**: döngü biter.


```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
▶     for (int i = 2; i <= n; i++)
4         sonuc *= i;
5     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

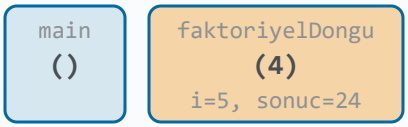
`n = 4` `sonuc = 24` `i = 5`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

return 24.

Adım 10 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
3     for (int i = 2; i <= n; i++)
4         sonuc *= i;
▶     return sonuc;
6 }
7 System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

`n = 4` `sonuc = 24`

ÇAĞRI YIĞINI

`faktoriyelDongu(4)` `main()`

ÇIKTI

Karşılaştırma: Döngülü `faktoriyel(4)`



main (yığının dibi) → yığın bu yönde büyür → üst

24 yazdırılır. Yığın hiç 2 çerçeveyi aşmadı → ek yığın O(1).

Adım 11 / 11

```
1 static long faktoriyelDongu(int n) {
2     long sonuc = 1;
3     for (int i = 2; i <= n; i++)
4         sonuc *= i;
5     return sonuc;
6 }
▶ System.out.println(faktoriyelDongu(4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

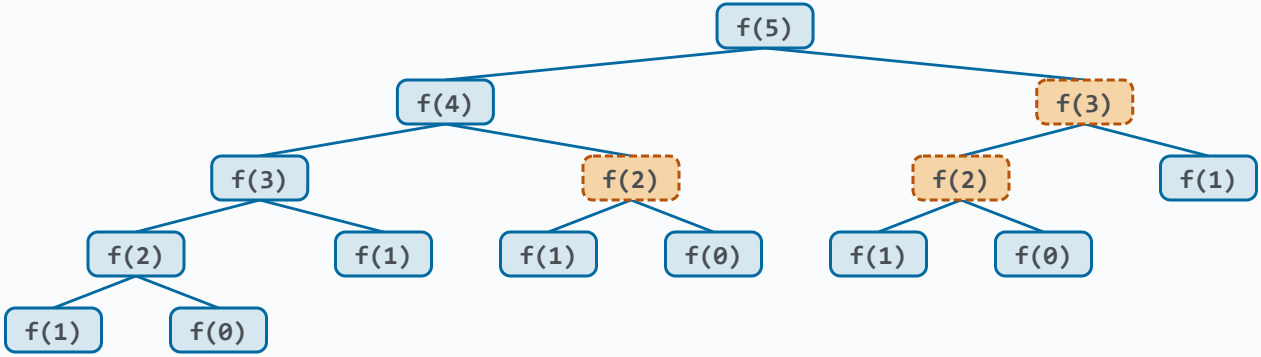
`main()`

ÇIKTI

```
24
```

VERİMLİLİK

Fibonacci Sorunu: `fib(5)` Özyineleme Ağacı



- fib(5) için **15** çağrı yapılır: f(3) **2** kez, f(2) **3** kez, f(1) 5 kez hesaplanır (kesikli turuncu = tekrar).
- fib(n − 1) ve fib(n − 2) alt ağaçları büyük ölçüde örtüşür; çağrı sayısı n ile yaklaşık **üstel** büyür.

Kod İzleme: `fib(4)`

main, `fib(4)` 'ü çağırır.

Adım 1 / 15

```
1 static long fib(int n) {
2     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
▶ System.out.println(fib(4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `fib(4)`

f(4)

`fib(4)` çağrıldı: **4 <= 1** → **false**.

Adım 2 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

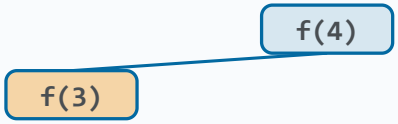
n = 4

ÇAĞRI YIĞINI

fib(4) main()

ÇIKTI

Kod İzleme: `fib(4)`



`fib(3)` çağrıldı: **3 <= 1 → false**.

Adım 3 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

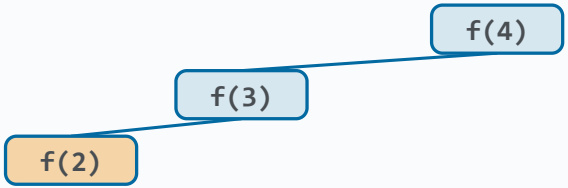
`n = 3`

ÇAĞRI YIĞINI

`fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(2)` çağrıldı: **2 <= 1 → false**.

Adım 4 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

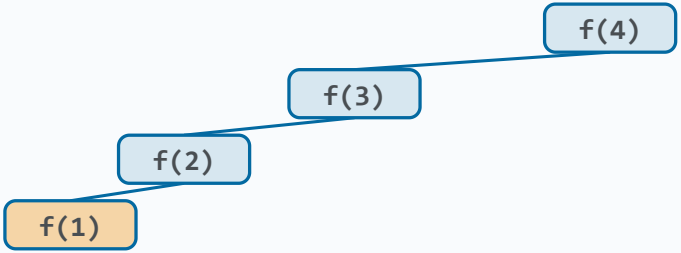
`n = 2`

ÇAĞRI YIĞINI

`fib(2)` `fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(1)` çağrıldı: **1 <= 1 → true**, **return 1**.

Adım 5 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

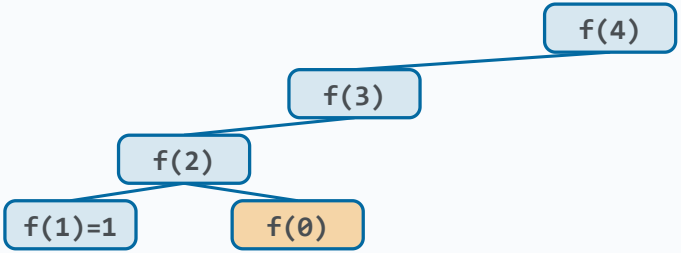
`n = 1`

ÇAĞRI YIĞINI

`fib(1)` `fib(2)` `fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(0)` çağrıldı: **0 <= 1 → true**, **return 0**.

Adım 6 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

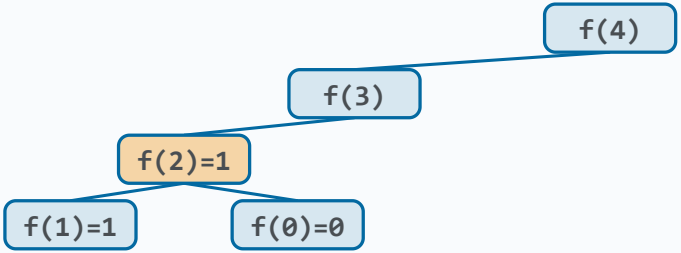
`n = 0`

ÇAĞRI YIĞINI

`fib(0)` `fib(2)` `fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



İki alt sonuç hazır: **1 + 0 = 1**, return.

Adım 7 / 15

```
1 static long fib(int n) {
2     if (n <= 1) return n;
▶     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

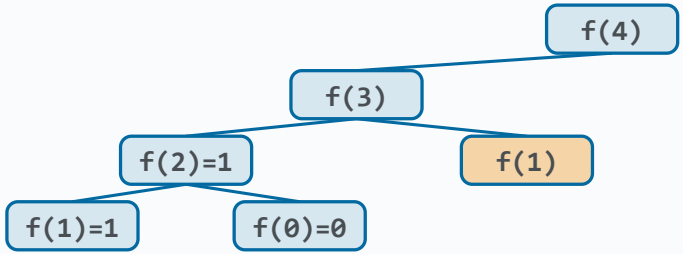
`n = 2`

ÇAĞRI YIĞINI

`fib(2)` `fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(1)` çağrıldı: **1 <= 1 → true**, **return 1**.

Adım 8 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

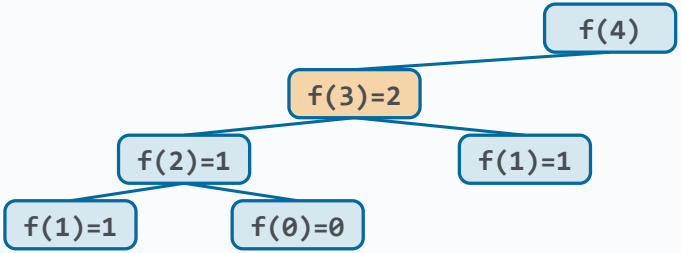
`n = 1`

ÇAĞRI YIĞINI

`fib(1)` `fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



İki alt sonuç hazır: **1 + 1 = 2**, return.

Adım 9 / 15

```
1 static long fib(int n) {
2     if (n <= 1) return n;
▶     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

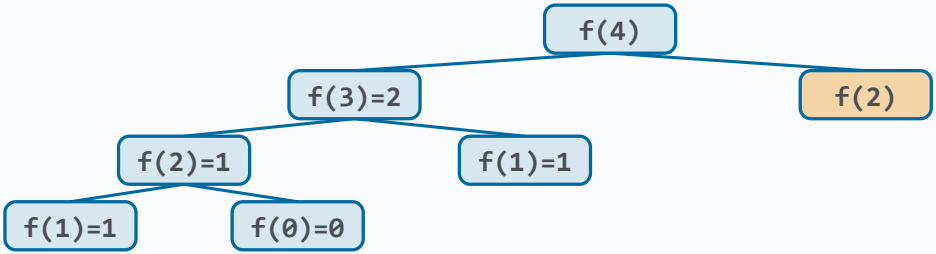
`n = 3`

ÇAĞRI YIĞINI

`fib(3)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(2)` çağrıldı: **2 <= 1 → false**. **Tekrar:** f(2) yeniden hesaplanıyor!

Adım 10 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

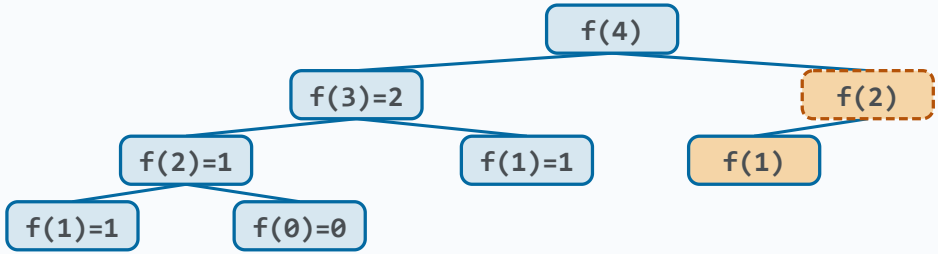
`n = 2`

ÇAĞRI YIĞINI

`fib(2)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(1)` çağrıldı: **1 <= 1 → true**, **return 1**.

Adım 11 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

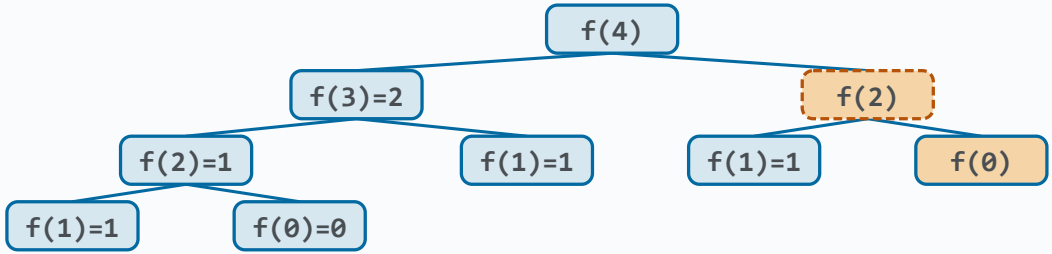
`n = 1`

ÇAĞRI YIĞINI

`fib(1)` `fib(2)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



`fib(0)` çağrıldı: **0 <= 1 → true**, **return 0**.

Adım 12 / 15

```
1 static long fib(int n) {
▶     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

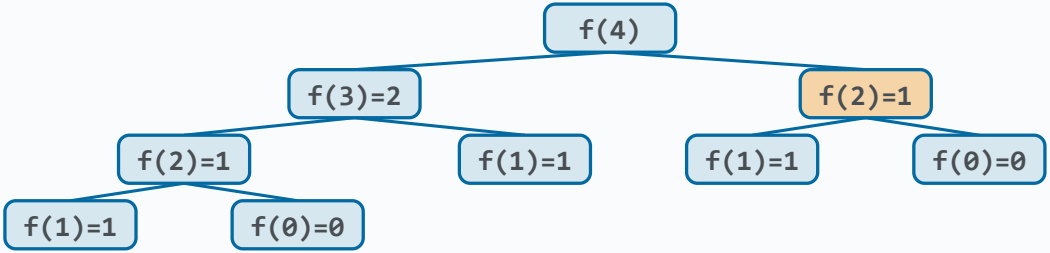
`n = 0`

ÇAĞRI YIĞINI

`fib(0)` `fib(2)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



İki alt sonuç hazır: **1 + 0 = 1**, return.

Adım 13 / 15

```
1 static long fib(int n) {
2     if (n <= 1) return n;
▶     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

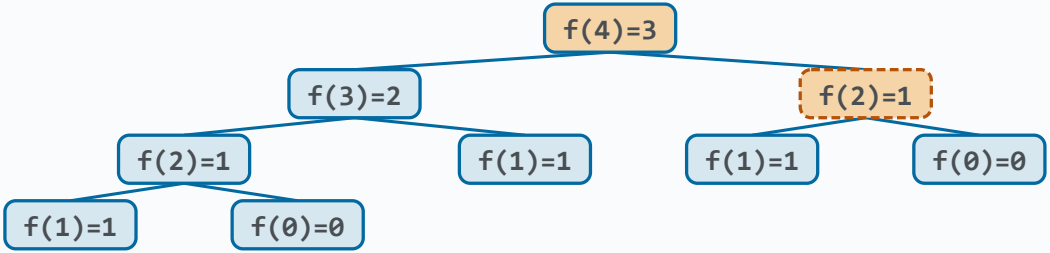
`n = 2`

ÇAĞRI YIĞINI

`fib(2)` `fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



İki alt sonuç hazır: **2 + 1 = 3**, return.

Adım 14 / 15

```
1 static long fib(int n) {
2     if (n <= 1) return n;
▶     return fib(n - 1) + fib(n - 2);
4 }
5 System.out.println(fib(4));
```

DEĞİŞKENLER

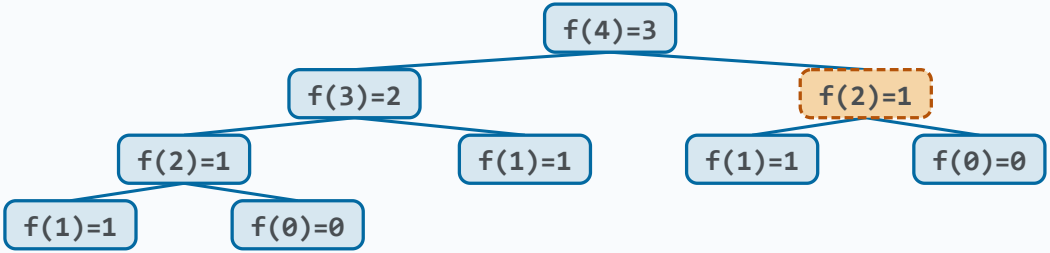
`n = 4`

ÇAĞRI YIĞINI

`fib(4)` `main()`

ÇIKTI

Kod İzleme: `fib(4)`



3 yazdırılır. Küçük bir n için bile 9 çağrı yapıldı.

Adım 15 / 15

```
1 static long fib(int n) {
2     if (n <= 1) return n;
3     return fib(n - 1) + fib(n - 2);
4 }
▶ System.out.println(fib(4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

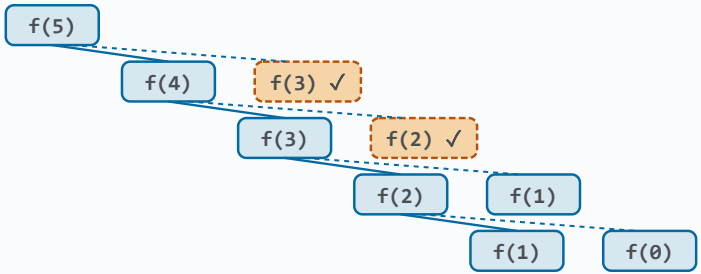
ÇIKTI

```
3
```

İYİLEŞTİRME

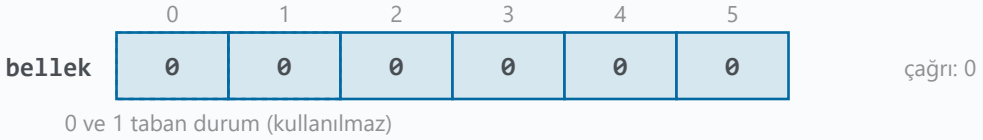
Bellekleme (Memoization)

- Hesaplanan alt problem sonuçları bir dizide (önbellek) saklanır.
- Aynı girdi tekrar gelince özyineleme ağacı yeniden açılmaz, **kayıtlı değer** döner.
- fib(5) için 15 çağrı yerine 9 çağrı; büyük n'de fark çok büyür.
- Zaman kazanımı karşılığında O(n) ek bellek kullanılır.



✓ = bellekten okundu, alt ağaç açılmadı

Kod İzleme: Belleklemeli `fib(5)`



6 hücreli bellek dizisi; **0** = henüz hesaplanmadı.

Adım 1 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
▶ long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(5)` : **bellek[5] == 0** → henüz yok, hesaplanacak.

Adım 2 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
▶     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 5`

`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(4)` : **bellek[4] == 0** → henüz yok, hesaplanacak.

Adım 3 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
▶     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 4`

`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(3)` : **bellek[3] == 0** → henüz yok, hesaplanacak.

Adım 4 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
▶     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 3`

`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(2)` : **bellek[2] == 0** → henüz yok, hesaplanacak.

Adım 5 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
▶     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 2`

`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(2)` `fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



0 ve 1 taban durum (kullanılmaz)

`fib(1)` : **1 <= 1** → **true**, return 1.

Adım 6 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 1`

`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(1)` `fib(2)` `fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(0)` : **0 <= 1** → **true**, return 0.

Adım 7 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 0`

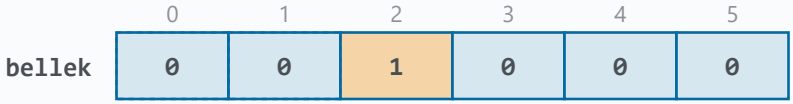
`bellek = [0, 0, 0, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(0)` `fib(2)` `fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



bellek[2] = 1 + 0 = 1 kaydedilir ve döndürülür.

Adım 8 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
▶     bellek[n] = fib(n - 1, bellek)
▶               + fib(n - 2, bellek);
▶     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 2`

`bellek = [0, 0, 1, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(2)` `fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(1)` : **1 <= 1** → **true**, return 1.

Adım 9 / 15

```
1 static long fib(int n, long[] bellek) {
▶     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 1`

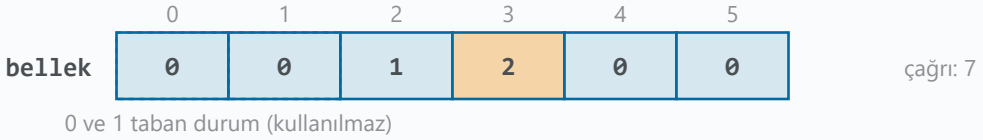
`bellek = [0, 0, 1, 0, 0, 0]`

ÇAĞRI YIĞINI

`fib(1)` `fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



bellek[3] = 1 + 1 = 2 kaydedilir ve döndürülür.

Adım 10 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
▶     bellek[n] = fib(n - 1, bellek)
▶               + fib(n - 2, bellek);
▶     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 3`

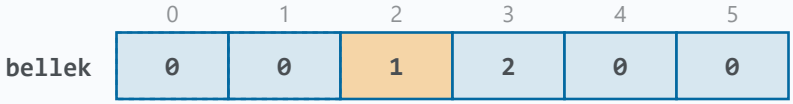
`bellek = [0, 0, 1, 2, 0, 0]`

ÇAĞRI YIĞINI

`fib(3)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(2)` : **bellek[2] = 1 != 0** → hesaplamadan **return 1**.

Adım 11 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
▶     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 2`

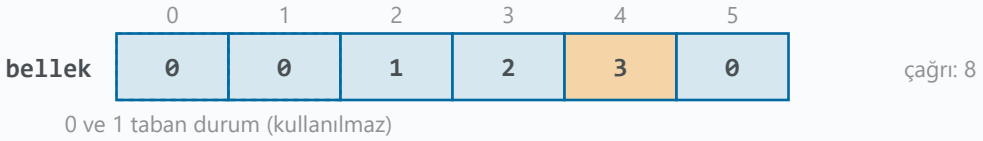
`bellek = [0, 0, 1, 2, 0, 0]`

ÇAĞRI YIĞINI

`fib(2)` `fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



bellek[4] = 2 + 1 = 3 kaydedilir ve döndürülür.

Adım 12 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
▶     bellek[n] = fib(n - 1, bellek)
▶               + fib(n - 2, bellek);
▶     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 4`

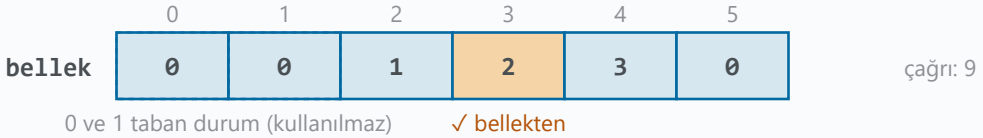
`bellek = [0, 0, 1, 2, 3, 0]`

ÇAĞRI YIĞINI

`fib(4)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



`fib(3)` : **bellek[3] = 2 != 0** → hesaplamadan **return 2**.

Adım 13 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
▶     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 3`

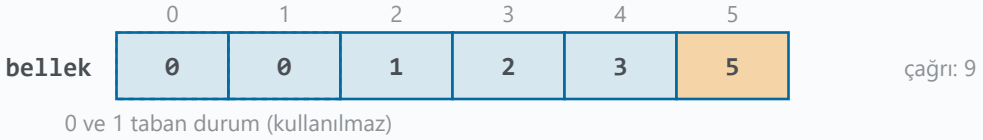
`bellek = [0, 0, 1, 2, 3, 0]`

ÇAĞRI YIĞINI

`fib(3)` `fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



bellek[5] = 3 + 2 = 5 kaydedilir ve döndürülür.

Adım 14 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
▶     bellek[n] = fib(n - 1, bellek)
▶               + fib(n - 2, bellek);
▶     return bellek[n];
7 }
8 long[] bellek = new long[6];
9 System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`n = 5`

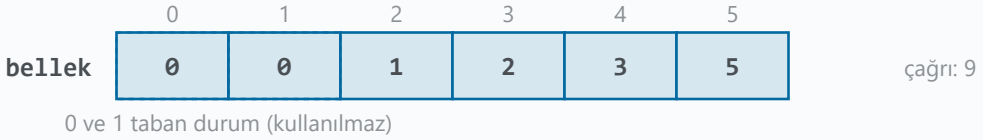
`bellek = [0, 0, 1, 2, 3, 5]`

ÇAĞRI YIĞINI

`fib(5)` `main()`

ÇIKTI

Kod İzleme: Belleklemeli `fib(5)`



5 yazdırılır. Toplam 9 çağrı (belleksiz 15) → zaman O(n).

Adım 15 / 15

```
1 static long fib(int n, long[] bellek) {
2     if (n <= 1) return n;
3     if (bellek[n] != 0) return bellek[n];
4     bellek[n] = fib(n - 1, bellek)
5               + fib(n - 2, bellek);
6     return bellek[n];
7 }
8 long[] bellek = new long[6];
▶ System.out.println(fib(5, bellek));
```

DEĞİŞKENLER

`bellek = [0, 0, 1, 2, 3, 5]`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

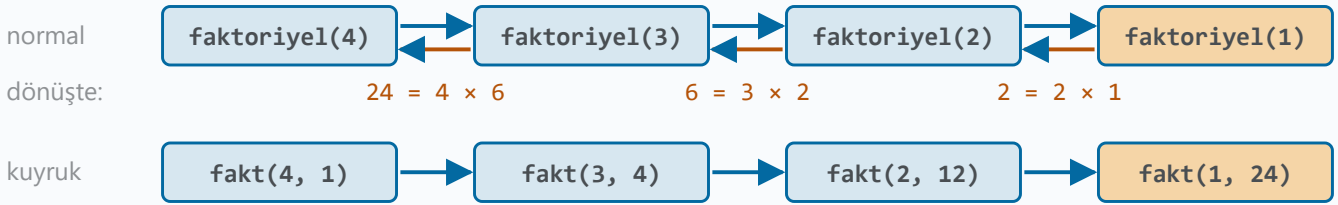
```
5
```

İYİLEŞTİRME

Kuyruk Özyinelemesi

- Özyinelemeli çağrı metodun **son işlemidir**; dönüşte yapılacak iş kalmaz.
- Ara sonuç bir parametrede (`birikim`) taşınır.
- Bazı diller bu çağrıları yığını büyütmeden optimize eder.
- Java dili/JVM kuyruk çağrısı optimizasyonunu **garanti etmez**: çerçeveler yine birikir.

```java
static long fakt(int n, long birikim) {
    if (n <= 1) return birikim;
    return fakt(n - 1, n * birikim);
}
// fakt(4, 1) → fakt(3, 4)
//   → fakt(2, 12) → fakt(1, 24) → 24
```



taban durum 24'ü döndürür; dönüşte çarpma kalmaz, 24 aynen geri iletilir

Kod İzleme: `fakt(4, 1)`



main, fakt(4, 1)'i çağırır; birikim başlangıçta 1

main (yığının dibi) → yığın bu yönde büyür → üst

main, `fakt(4, 1)` 'i çağırır: ikinci parametre **birikim** başlangıçta 1.

Adım 1 / 6

```
1 static long fakt(int n, long birikim) {
2     if (n <= 1) return birikim;
3     return fakt(n - 1, n * birikim);
4 }
▶ System.out.println(fakt(4, 1)); // 24
```

DEĞİŞKENLER

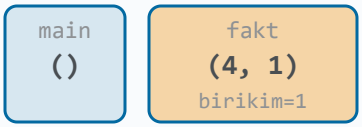
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `fakt(4, 1)`



main (yığının dibi) → yığın bu yönde büyür → üst

Çerçeve açıldı. **4 <= 1 → false**. Son işlem `fakt(3, 4 * 1)` yani **fakt(3, 4)** çağrısı; dönüşte yapılacak çarpma yok.

Adım 2 / 6

```
1 static long fakt(int n, long birikim) {
▶     if (n <= 1) return birikim;
▶     return fakt(n - 1, n * birikim);
4 }
5 System.out.println(fakt(4, 1)); // 24
```

DEĞİŞKENLER

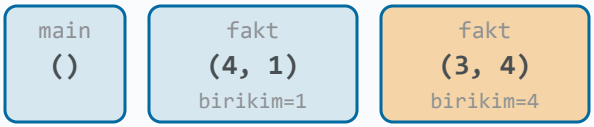
`n = 4` `birikim = 1`

ÇAĞRI YIĞINI

`fakt(4, 1)` `main()`

ÇIKTI

Kod İzleme: `fakt(4, 1)`



main (yığının dibi) → yığın bu yönde büyür → üst

Çarpma çağrıdan **önce** yapıldı: sıradaki çağrı **fakt(2, 12)**. Eski çerçevenin bekleyecek işi yok; ama Java onu **silmez**.

Adım 3 / 6

```
1 static long fakt(int n, long birikim) {
▶     if (n <= 1) return birikim;
▶     return fakt(n - 1, n * birikim);
4 }
5 System.out.println(fakt(4, 1)); // 24
```

DEĞİŞKENLER

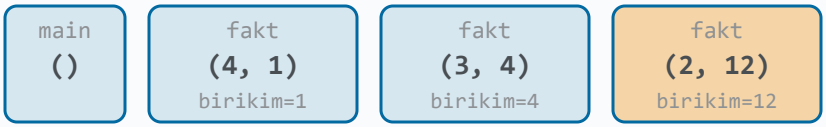
`n = 3` `birikim = 4`

ÇAĞRI YIĞINI

`fakt(3, 4)` `fakt(4, 1)` `main()`

ÇIKTI

Kod İzleme: `fakt(4, 1)`



main (yığının dibi) → yığın bu yönde büyür → üst

Sıradaki çağrı **fakt(1, 24)**. Yığın yine her çağrıda **bir çerçeve** büyüyor.

Adım 4 / 6

```
1 static long fakt(int n, long birikim) {
▶     if (n <= 1) return birikim;
▶     return fakt(n - 1, n * birikim);
4 }
5 System.out.println(fakt(4, 1)); // 24
```

DEĞİŞKENLER

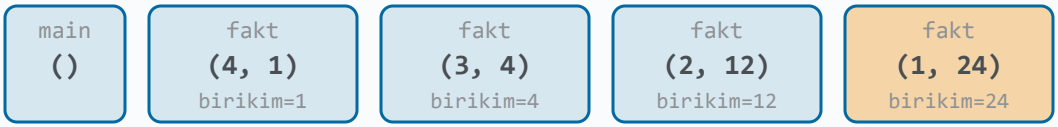
`n = 2` `birikim = 12`

ÇAĞRI YIĞINI

`fakt(2, 12)` `fakt(3, 4)` `fakt(4, 1)` `main()`

ÇIKTI

Kod İzleme: `fakt(4, 1)`



main (yığının dibi) → yığın bu yönde büyür → üst

1 <= 1 → **true**: taban durum, `return birikim` → **24**. Kuyruk özyinelemesi olsa da yığında **5 çerçeve** (main + 4) birikti: Java'da kuyruk çağrısı optimizasyonu yok.

Adım 5 / 6

```
1 static long fakt(int n, long birikim) {
▶     if (n <= 1) return birikim;
3     return fakt(n - 1, n * birikim);
4 }
5 System.out.println(fakt(4, 1)); // 24
```

DEĞİŞKENLER

`n = 1` `birikim = 24`

ÇAĞRI YIĞINI

`fakt(1, 24)` `fakt(2, 12)` `fakt(3, 4)` `fakt(4, 1)` `main()`

ÇIKTI

Kod İzleme: `fakt(4, 1)`



24 çerçevelerden geçerek main'e döner; çarpma yapılmadı, yalnızca sonuç taşındı. Bellek yine **O(n)**.

Adım 6 / 6

```
1 static long fakt(int n, long birikim) {
2     if (n <= 1) return birikim;
3     return fakt(n - 1, n * birikim);
4 }
▶ System.out.println(fakt(4, 1)); // 24
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
24
```

DOĞRULAMA

Taşma ve Uç Durumlar

long taşması

`faktoriyel(20)` → `2432902008176640000` (doğru).
`faktoriyel(21)` → `-4249290049419214848`
Hata fırlatılmaz; değer sessizce sarar. `long` en çok 9223372036854775807 tutar.

Negatif üs

`us(2, -1)`: taban durum `n == 0` hiç sağlanmaz; n −1, −2, … diye iner → **StackOverflowError**. Önlem: `n < 0` ise `IllegalArgumentException` fırlatılır.

Fibonacci: özyinelemeli ≈ O(2ⁿ), döngülü O(n)

n	fib(n) özyinelemeli: çağrı sayısı	fib(n) döngülü: tur sayısı
10	177	10
20	21891	20
30	2692537	30

Sayılar, her çağrıda sayaç artırılarak elde edildi; süre ölçümü değildir.

Hata Ayıklama

- Taban duruma her girdi için ulaşıldığı gösterilir.
- Problemin her çağrıda hangi ölçüye göre küçüldüğü belirlenir.
- Küçük bir girdi için çağrı ve dönüş sırası tabloya dökülür (sağda `toplam(2)`).
- Özyineleme derinliği ve tekrarlanan alt problemler değerlendirilir.

Derinlik	Olay	Değer
0	`toplam(2)` çağrıldı	
1	`toplam(1)` çağrıldı	
2	`toplam(0)` çağrıldı	
2	`toplam(0)` döndü	0
1	`toplam(1)` döndü	1
0	`toplam(2)` döndü	3

ÖZET

Bölüm Özeti

Yapı

Taban durum + problemi küçülten özyinelemeli durum.

Çağrı yığını

Her çağrı bir çerçeve açar; taban durumdan sonra çerçeveler ters sırada kapanır.

Maliyet

Doğrusal özyineleme O(n) yığın kullanır; döngü çoğu zaman O(1) ile aynı işi yapar.

Bellekleme

Tekrarlanan alt problem sonuçlarını saklamak fib'i üstelden O(n)'e indirir.

ÖZET

Kontrol Soruları

1. Taban durum olmazsa ne olur?
2. Çağrı yığını neden ters sırada çözülür?
3. Özyinelemeli faktöriyel ne kadar ek yığın kullanır?
4. Doğrudan Fibonacci neden yavaştır?
5. Java'da kuyruk özyinelemesi neden yığın güvenliği sağlamaz?

▶ **Cevaplar**

ÖZET

Alıştırmalar

1. Bir dizinin öğeleri toplamını özyinelemeli yazın: `toplam(a, i)` = a[i] + toplam(a, i + 1).
2. Bir sayının basamak sayısını özyinelemeli bulun (`basamak(472)` = 3).
3. Onluk sayıyı özyinelemeyle ikiliye çeviren `ikili(n)` metodunu yazın (13 → "1101").
4. Hanoi'de n = 4 için kaç hamle gerekir? Programla doğrulayın.
5. `kombinasyon(n, k) = kombinasyon(n − 1, k − 1) + kombinasyon(n − 1, k)` için taban durumları yazın; belleklemeyle hızlandırın.
6. Bir dizgideki bir karakterin kaç kez geçtiğini özyinelemeli sayın.

ÖZET

Kaynaklar

- Roberts, Thinking Recursively with Java, Wiley, 2006.
- Sedgewick ve Wayne, Computer Science: An Interdisciplinary Approach, Addison-Wesley, 2016: **2.3 Recursion**.
- Cormen, Leiserson, Rivest ve Stein, Introduction to Algorithms, 4. baskı, MIT Press, 2022.
- Liang, Introduction to Java Programming and Data Structures, 13. baskı, Pearson, 2024.
- Oracle, Java SE 21 API: **StackOverflowError**.
- Downey ve Mayfield, Think Java, 2. baskı, O'Reilly, 2019 (**ücretsiz çevrim içi sürüm**).

Son

Özyineleme

Çalışma soruları · Simülatör: Özyineleme Görselleştirici · Terim sözlüğü