Bölüm 4: Özyineleme

Veri Yapıları

Taban durum ve özyinelemeli adım | Çağrı yığını | Doğrusal özyineleme | Hızlı üs alma | Fibonacci ve ikili arama | Hanoi kuleleri | Geri izleme | Yineleme bağıntıları | Kuyruk özyinelemesi | Bağlı listede özyineleme

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Özyinelemeli tasarım

Bir problemi taban durum ve özyinelemeli adımla tanımlayıp Java'da yazmak.

Çağrı yığını

Özyinelemeli bir metodu çerçeve çerçeve izleyip çıktısını ve dönüş değerini bulmak.

Özyineleme türleri

Doğrusal, ikili ve çoklu özyinelemeyi ve geri izlemeyi örnekleriyle ayırt etmek.

Maliyet

Yineleme bağıntısını yazıp çağrı ağacıyla zaman ve alan maliyetini bulmak; gerektiğinde döngüye çevirmek.

ÖZYİNELEME NEDİR

Günlük Hayatta Özyineleme

Uzun bir kuyrukta kaçıncı sırada olduğunuzu öğrenmek istiyorsunuz ama yerinizden ayrılıp baştan sayamıyorsunuz.



Aynı soru, küçük kuyruk

Herkes önündekine aynı soruyu sorar; soru her seferinde bir kişi daha kısa bir kuyruk için sorulur.

Taban durum

En öndeki kişi kimseye sormadan cevap verir: "1". Sorular burada durur.

Geri dönüş

Cevap geri gelirken her kişi kendi payını ekler: 1, 2, 3, 4, 5.

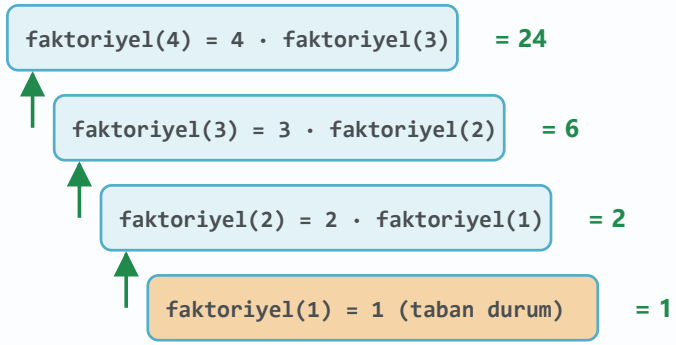
Özyinelemeli bir metot da böyle çalışır: kendini **daha küçük** bir girdiyle çağırır, en küçük girdide cevabı **doğrudan** verir.

ÖZYİNELEME NEDİR

Özyineleme (Recursion)

$$n! = 1 \ (n \le 1) \cdot n! = n \cdot (n - 1)! \ (n > 1)$$

- **Özyineleme**: bir metodun kendini daha küçük bir girdiyle çağırması.
- **Taban durum** (base case): özyinelemesiz, doğrudan çözülür. Olmazsa çağrılar bitmez → `StackOverflowError`.
- **Özyinelemeli adım**: problemi küçültür ve taban duruma yaklaştırır; küçük problemin cevabından büyüğünkini kurar.
- Her çağrı çağrı yığınında yeni bir **çerçeve** açar (kendi n'si).



↓ çağrılar iner · ↑ değerler döner

ÖZYİNELEME NEDİR

Özyinelemeli Düşünmek

1. **Taban durum:** en küçük girdi hangisi ve cevabı ne? Burada özyineleme durur.
2. **Küçültme:** problem, aynı türden daha küçük bir probleme nasıl indirgenir? (n − 1, n / 2, i + 1 …)
3. **Birleştirme:** küçük problemin cevabından büyük problemin cevabı nasıl kurulur?

Küçük problemin doğru çözüldüğü kabul edilir ve yalnızca **tek bir adım** düşünülür. Taban durum doğruysa ve her adım doğruysa tüm zincir doğrudur (tümevarımdaki mantık).

```
void geriSay(int n) {
  if (n == 0) {     // taban durum
    System.out.println("Bitti");
    return;
  }
  System.out.println(n);
  geriSay(n - 1);   // özyinelemeli adım
}
geriSay(3);         // main içinde
```

geriSay(n): n'den 1'e kadar yaz, sonra “Bitti”.

Her özyinelemeli çağrı girdiyi **taban duruma yaklaştırmalıdır**; geriSay'de n her çağrıda 1 azalır ve 0'a ulaşır.

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

main, `geriSay(3)` 'ü çağırır. Yığına yeni bir çerçeve eklenecek.

Adım 1 / 10

```
1  void geriSay(int n) {
2    if (n == 0) {      // taban durum
3      System.out.println("Bitti");
4      return;
5    }
6    System.out.println(n);
7    geriSay(n - 1);    // özyinelemeli adım
8  }
▶ geriSay(3);          // main içinde
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `geriSay(3)`



n = 3 ≠ 0: taban durum değil. **3** yazılır, sonra `geriSay(2)` çağrılır; bu çerçeve bekler.

Adım 2 / 10

```
1 void geriSay(int n) {
▶   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
▶   System.out.println(n);
▶   geriSay(n - 1);    // özyinelemeli adım
8 }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 3`

ÇIKTI

```
3
```

ÇAĞRI YIĞINI

`geriSay(3)`

`main()`

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

n = 2 ≠ 0: taban durum değil. **2** yazılır, sonra `geriSay(1)` çağrılır; bu çerçeve bekler.

Adım 3 / 10

```
1 void geriSay(int n) {
▶   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
▶   System.out.println(n);
▶   geriSay(n - 1);    // özyinelemeli adım
8 }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 2`

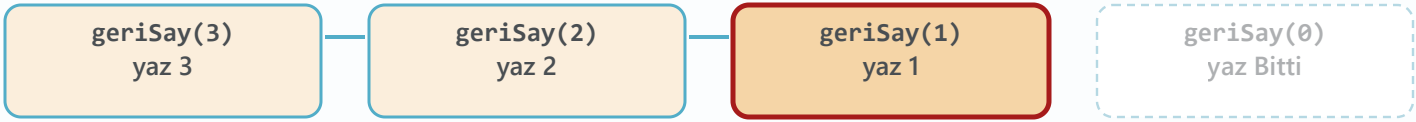
ÇIKTI

```
3
2
```

ÇAĞRI YIĞINI

- `geriSay(2)`
- `geriSay(3)`
- `main()`

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

n = 1 ≠ 0: taban durum değil. **1** yazılır, sonra `geriSay(0)` çağrılır; bu çerçeve bekler.

Adım 4 / 10

```
1 void geriSay(int n) {
    if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
    System.out.println(n);
    geriSay(n - 1);    // özyinelemeli adım
8 }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 1`

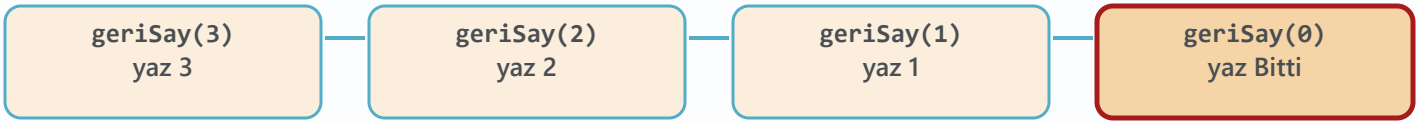
ÇIKTI

```
3
2
1
```

ÇAĞRI YIĞINI

- `geriSay(1)`
- `geriSay(2)`
- `geriSay(3)`
- `main()`

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

n = 0: **taban durum**. “Bitti” yazılır. Yığın en derinde: main + 4 çerçeve.

Adım 5 / 10

```
1 void geriSay(int n) {
▶   if (n == 0) {      // taban durum
▶     System.out.println("Bitti");
4     return;
5   }
6   System.out.println(n);
7   geriSay(n - 1);    // özyinelemeli adım
8 }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 0`

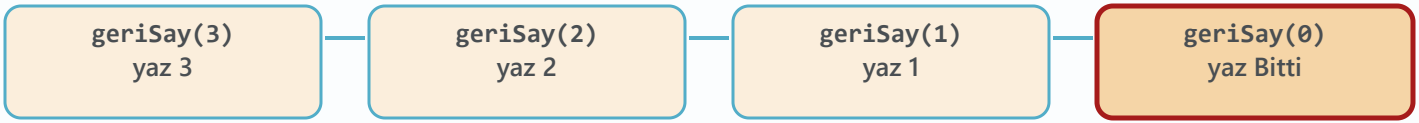
ÇIKTI

```
3
2
1
Bitti
```

ÇAĞRI YIĞINI

- `geriSay(0)`
- `geriSay(1)`
- `geriSay(2)`
- `geriSay(3)`
- `main()`

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

`return` : geriSay(0) biter, çerçevesi yığından çıkar.

Adım 6 / 10

```
1 void geriSay(int n) {
2   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
▶     return;
5   }
6   System.out.println(n);
7   geriSay(n - 1);    // özyinelemeli adım
8 }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 0`

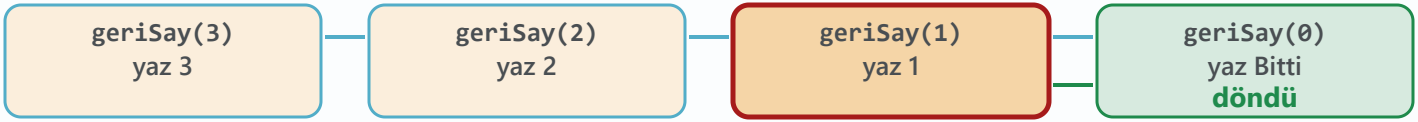
ÇIKTI

```
3
2
1
Bitti
```

ÇAĞRI YIĞINI

- **geriSay(0)**
- geriSay(1)
- geriSay(2)
- geriSay(3)
- main()

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

geriSay(1), satır 7'deki çağrıdan döner. Yapılacak iş kalmadı → o da biter.

Adım 7 / 10

```
1 void geriSay(int n) {
2   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
6   System.out.println(n);
7   geriSay(n - 1);    // özyinelemeli adım
▶ }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 1`

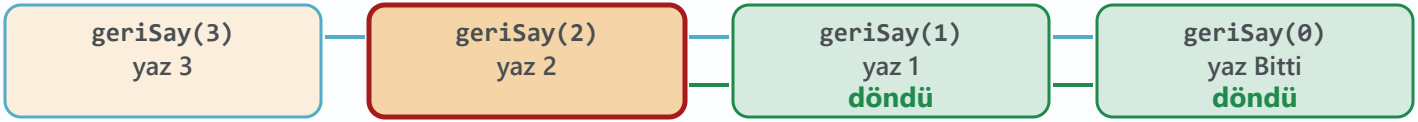
ÇIKTI

```
3
2
1
Bitti
```

ÇAĞRI YIĞINI

- **geriSay(1)**
- geriSay(2)
- geriSay(3)
- main()

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

geriSay(2), satır 7'deki çağrıdan döner. Yapılacak iş kalmadı → o da biter.

Adım 8 / 10

```
1 void geriSay(int n) {
2   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
6   System.out.println(n);
7   geriSay(n - 1);    // özyinelemeli adım
▶ }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 2`

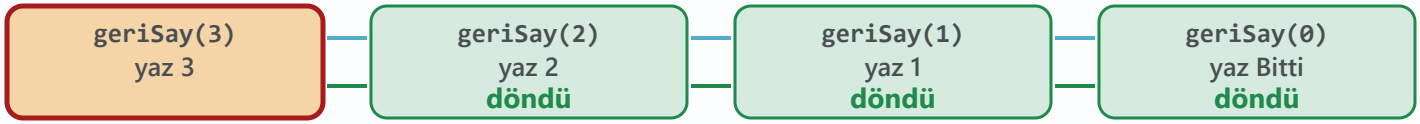
ÇIKTI

```
3
2
1
Bitti
```

ÇAĞRI YIĞINI

- **geriSay(2)**
- geriSay(3)
- main()

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

geriSay(3), satır 7'deki çağrıdan döner. Yapılacak iş kalmadı → o da biter.

Adım 9 / 10

```
1 void geriSay(int n) {
2   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
6   System.out.println(n);
7   geriSay(n - 1);    // özyinelemeli adım
▶ }
9 geriSay(3);          // main içinde
```

DEĞİŞKENLER

`n = 3`

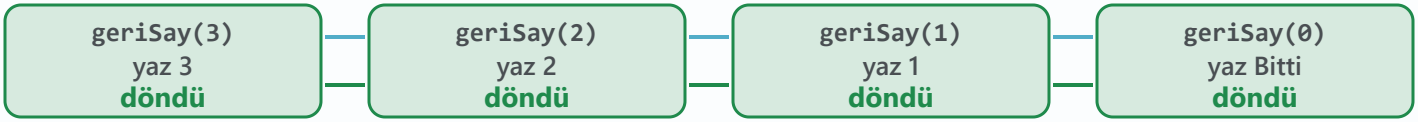
ÇIKTI

```
3
2
1
Bitti
```

ÇAĞRI YIĞINI

- `geriSay(3)`
- `main()`

Kod İzleme: `geriSay(3)`



→ çağrı · ← dönüş

Tüm çerçeveler döndü. 4 çağrı yapıldı (n = 3, 2, 1, 0); çıktı: 3 2 1 Bitti.

Adım 10 / 10

```
1 void geriSay(int n) {
2   if (n == 0) {      // taban durum
3     System.out.println("Bitti");
4     return;
5   }
6   System.out.println(n);
7   geriSay(n - 1);    // özyinelemeli adım
8 }
▶ geriSay(3);          // main içinde
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
3
2
1
Bitti
```

İş Çağrıdan Önce mi, Sonra mı?

```java
void geriSay2(int n) {
  if (n == 0) return;
  System.out.print(n + " ");
  geriSay2(n - 1);      // sonra çağrı
}
```

`geriSay2(3)` çıktısı: **`3 2 1`**
Yazdırma çağrılar **inerken** yapılır.

```java
void ileriSay(int n) {
  if (n == 0) return;
  ileriSay(n - 1);      // önce çağrı
  System.out.print(n + " ");
}
```

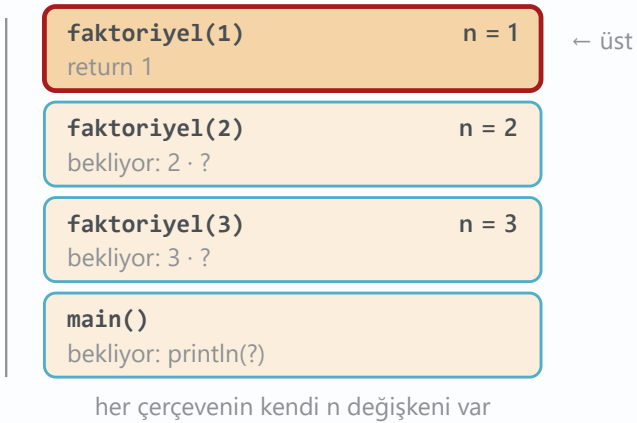
`ileriSay(3)` çıktısı: **`1 2 3`**
Yazdırma çağrılar **dönerken** yapılır.

- `ileriSay(3)`, 3'ü yazmadan önce `ileriSay(2)`'nin bitmesini bekler; o da 2'yi yazmadan önce `ileriSay(1)`'i bekler.
- İlk biten çağrı en son başlayandır (n = 0). Bu yüzden çağrıdan sonra yapılan işler **ters sırayla** yapılır: 1, 2, 3.

Çağrıdan sonra kalan iş, çerçeve yığında beklerken saklanır. Bu iş yığın çözülürken yapılır.

Çağrı Yığını ve Çerçeveler

- Her metot çağrısı için bellekte bir **yığın çerçevesi** (stack frame) ya da kısaca çerçeve açılır: parametreler, yerel değişkenler ve dönüşte devam edilecek yer.
- Çerçeveler **çağrı yığınında** tutulur: çağrı yapılınca üste eklenir (push), metot bitince üstten çıkar (pop). Son başlayan ilk biter: LIFO.
- Yalnızca en üstteki çerçeve çalışır; alttakiler, üstlerindeki çağrının sonucunu bekler.
- Özyinelemede aynı metodun **birden çok çerçevesi** aynı anda yığında bulunur; her birinin n'si ayrıdır.



Özyinelemeli bir metodun bellek maliyeti, yığında aynı anda bulunan çerçeve sayısıyla, yani **en derin çağrı zinciriyle** belirlenir.

Kod İzleme: `faktoriyel(4)` ve çağrı yığını

→ çağrı · ← dönen değer

main, `faktoriyel(4)`'ü çağırır.

Adım 1 / 13

```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
▶ System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını

faktoriyel(4)
n = 4

→ çağrı · ← dönen değer

Yeni çerçeve: **n = 4**. 4 ≤ 1 mi? Hayır.

Adım 2 / 13

```
1 int faktoriyel(int n) {
▶   if (n <= 1)
3     return 1;                // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- faktoriyel(4)
- main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını

faktoriyel(4)
n = 4

→ çağrı · ← dönen değer

4 · faktoriyel(3): çarpım için önce **faktoriyel(3)** bitmeli; bu çerçeve **bekler**.

Adım 3 / 13

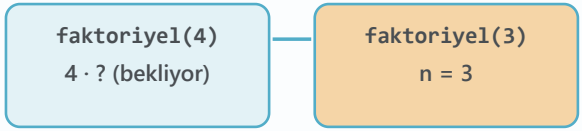
```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                 // taban durum
▶   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- faktoriyel(4)
- main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

Yeni çerçeve: **n = 3**. 3 ≤ 1 mi? Hayır.

Adım 4 / 13

```
1 int faktoriyel(int n) {
▶   if (n <= 1)
3     return 1;                 // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

3 · faktoriyel(2): çarpım için önce **faktoriyel(2)** bitmeli; bu çerçeve **bekler**.

Adım 5 / 13

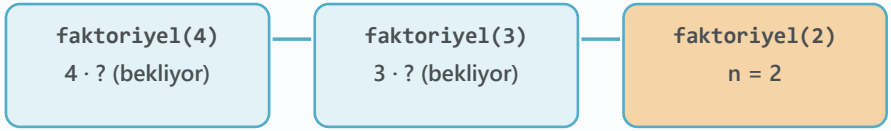
```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                 // taban durum
▶   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- **faktoriyel(3)**
- faktoriyel(4)
- main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

Yeni çerçeve: **n = 2**. 2 ≤ 1 mi? Hayır.

Adım 6 / 13

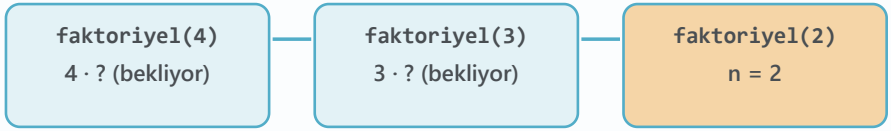
```
1 int faktoriyel(int n) {
▶   if (n <= 1)
3     return 1;                 // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- **faktoriyel(2)**
- faktoriyel(3)
- faktoriyel(4)
- main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

2 · faktoriyel(1): çarpım için önce **faktoriyel(1)** bitmeli; bu çerçeve **bekler**.

Adım 7 / 13

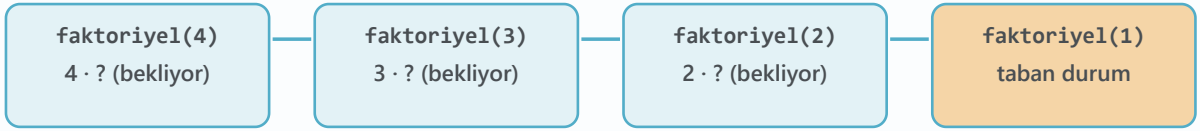
```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                 // taban durum
▶   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- **faktoriyel(2)**
- faktoriyel(3)
- faktoriyel(4)
- main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

Yeni çerçeve: **n = 1**. 1 ≤ 1 mi? **Evet**.

Adım 8 / 13

```
1 int faktoriyel(int n) {
▶   if (n <= 1)
3     return 1;                 // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

Taban durum: return 1. Artık çağrı yok; değerler geri dönmeye başlar.

Adım 9 / 13

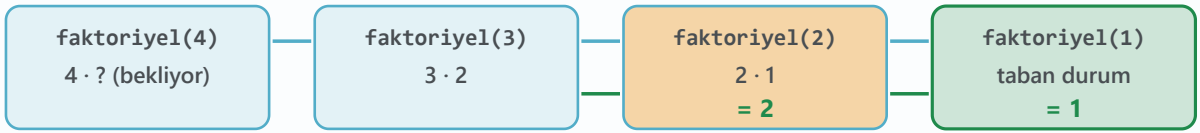
```
1 int faktoriyel(int n) {
2   if (n <= 1)
▶     return 1;                // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- **faktoriyel(1)**
- faktoriyel(2)
- faktoriyel(3)
- faktoriyel(4)
- main()

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

faktoriyel(1) = 1 döndü → **2 · 1 = 2**, return 2.

Adım 10 / 13

```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                 // taban durum
▶   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

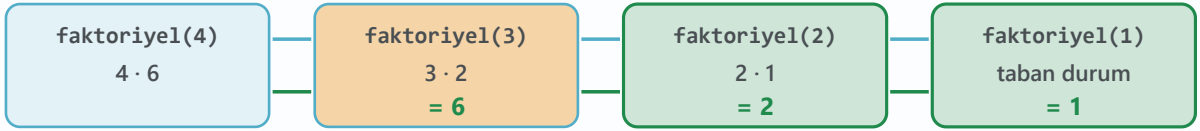
ÇIKTI

[redacted]

ÇAĞRI YIĞINI

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

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

faktoriyel(2) = 2 döndü → **3 · 2 = 6**, return 6.

Adım 11 / 13

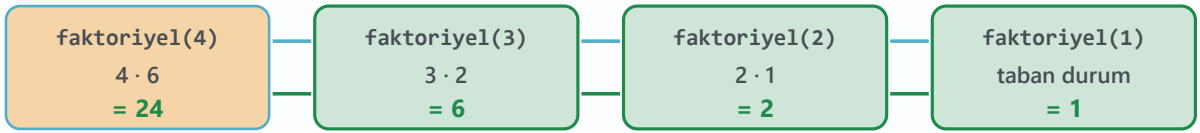
```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                 // taban durum
▶   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

faktoriyel(3) = 6 döndü → **4 · 6 = 24**, return 24.

Adım 12 / 13

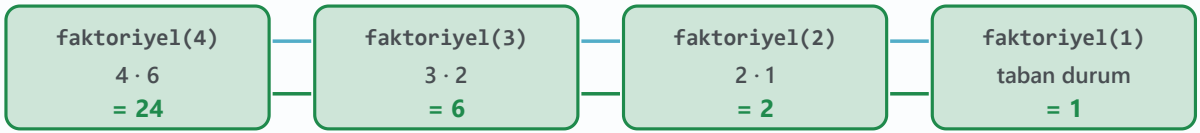
```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                // taban durum
▶   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

ÇIKTI

ÇAĞRI YIĞINI

- `faktoriyel(4)`
- `main()`

Kod İzleme: `faktoriyel(4)` ve çağrı yığını



→ çağrı · ← dönen değer

Ekrana **24**. 4 çağrı, yığın derinliği en fazla 4 + main → zaman ve bellek **O(n)**.

Adım 13 / 13

```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                // taban durum
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
▶ System.out.println(faktoriyel(4));
```

ÇIKTI

```
24
```

ÇAĞRI YIĞINI

`main()`

Yığın Taşması: `StackOverflowError`

```
long topla(int n) {          // 1 + 2 + … + n
  if (n == 0) return 0;
  return n + topla(n - 1);
}
// main içinde:
System.out.println(topla(10_000));    // 50005000
System.out.println(topla(1_000_000)); // hata
```

```
Exception in thread "main"
  java.lang.StackOverflowError
    at Ornek.topla(Ornek.java:3)
    at Ornek.topla(Ornek.java:3)
    …
```

- Çağrı yığını **sınırlı** bir bellek alanıdır. Boyutu JVM'e ve işletim sistemine bağlıdır; `-Xss` seçeneğiyle değiştirilebilir.
- Her çerçeve yer kaplar. Varsayılan ayarlarla derinlik genellikle birkaç bin ile birkaç on bin çağrıyla sınırlıdır.
- `topla(10_000)` sığar; `topla(1_000_000)` taban duruma ulaşmadan yığını doldurur.
- Aynı toplam bir döngüyle ya da n(n + 1)/2 formülüyle sorunsuz hesaplanır.

Taban Durumu Unutmak

Taban durum yok

```
int f(int n) {
  return n * f(n - 1);
}
```

n = 3, 2, 1, 0, −1, −2 … çağrılar hiç durmaz.

Taban duruma ulaşılamıyor

```
if (n == 1) return 1;
return n * f(n - 1);
```

`f(0)` → −1 → −2 …: n hiçbir zaman 1 olmaz. `n <= 1` bu girdiyi de yakalar.

Problem küçülmüyor

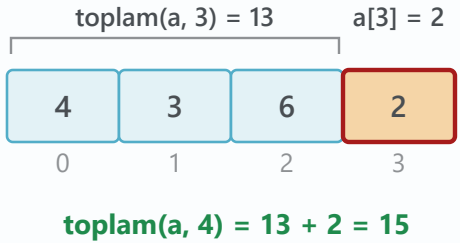
```
if (n <= 1) return 1;
return n * f(n);
```

Aynı girdiyle tekrar çağrı: taban durum var ama ona hiç yaklaşılmıyor.

Üçünün sonucu da aynıdır: yığın dolar ve `StackOverflowError` oluşur. Her olası girdi (0 ve negatif değerler dahil) sonlu sayıda adımda bir taban duruma ulaşmalıdır.

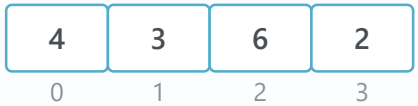
Doğrusal Özyineleme

- Her çağrı **en fazla bir** özyinelemeli çağrı yapar; çağrılar dallanmayan bir **zincir** oluşturur.
- n boyutlu problem, n − 1 (ya da n / 2) boyutlu **tek** bir probleme indirgenir.
- Örnekler: faktoriyel, geriSay, dizinin toplamı, diziyi tersine çevirme, üs alma, bağlı listenin uzunluğu.
- Dizinin toplamı: ilk n öğenin toplamı = ilk n − 1 öğenin toplamı + a[n − 1]. Taban durum: n = 0 (boş parça) → 0.




```
int toplam(int[] a, int n) {   // ilk n öğenin toplamı
  if (n == 0)
    return 0;                  // taban durum
  return toplam(a, n - 1) + a[n - 1];
}
```

Kod İzleme: `toplam(a, 4)`



`toplam(a, 4) = ?`

dönen değerler: —

main, `toplam(a, 4)` 'ü çağırır: dizinin ilk 4 öğesinin toplamı.

Adım 1 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
2   if (n == 0)
3     return 0;                  // taban durum
4   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
▶ System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

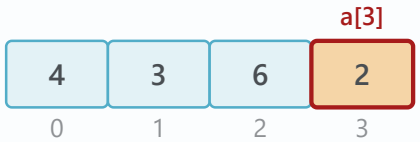
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `toplam(a, 4)`



`toplam(a, 4) = toplam(a, 3) + a[3]`

dönen değerler: —

n = 4 ≠ 0 → önce `toplam(a, 3)` hesaplanmalı; a[3] = 2 toplamaya eklenmeyi bekler.

Adım 2 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
▶   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 4`

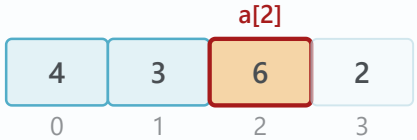
ÇIKTI

ÇAĞRI YIĞINI

`toplam(a,4)`

`main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 3) = toplam(a, 2) + a[2]`

dönen değerler: —

n = 3 ≠ 0 → önce `toplam(a, 2)` hesaplanmalı; a[2] = 6 toplamaya eklenmeyi bekler.

Adım 3 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
▶   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 3`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,3)`**
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 2) = toplam(a, 1) + a[1]`

dönen değerler: —

n = 2 ≠ 0 → önce `toplam(a, 1)` hesaplanmalı; a[1] = 3 toplamaya eklenmeyi bekler.

Adım 4 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
▶   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

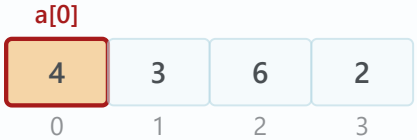
`n = 2`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,2)`**
- `toplam(a,3)`
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 1) = toplam(a, 0) + a[0]`

dönen değerler: —

n = 1 ≠ 0 → önce `toplam(a, 0)` hesaplanmalı; a[0] = 4 toplamaya eklenmeyi bekler.

Adım 5 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
▶   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 1`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,1)`**
- `toplam(a,2)`
- `toplam(a,3)`
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`

4	3	6	2
0	1	2	3

`toplam(a, 0) = 0 (boş parça)`

dönen değerler: 0

n = 0: **taban durum** → 0 döner. Yığın en derinde: main + 5 çerçeve.

Adım 6 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
▶   if (n == 0)
▶     return 0;                  // taban durum
4   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

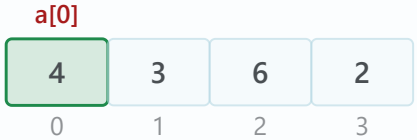
`n = 0`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,0)`**
- `toplam(a,1)`
- `toplam(a,2)`
- `toplam(a,3)`
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 1) = 0 + 4 = 4`

dönen değerler: 0 → 4

toplam(a, 0) 0 döndürdü → 0 + a[0] = 0 + 4 = **4** döner.

Adım 7 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
2   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 1`

`dönen = 0`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,1)`**
- `toplam(a,2)`
- `toplam(a,3)`
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 2) = 4 + 3 = 7`

dönen değerler: 0 → 4 → 7

toplam(a, 1) 4 döndürdü → 4 + a[1] = 4 + 3 = **7** döner.

Adım 8 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
2   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 2`

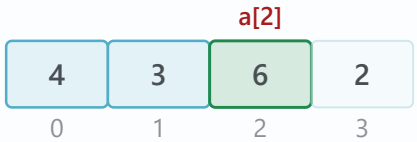
`dönen = 4`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,2)`**
- `toplam(a,3)`
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 3) = 7 + 6 = 13`

dönen değerler: 0 → 4 → 7 → 13

toplam(a, 2) 7 döndürdü → 7 + a[2] = 7 + 6 = **13** döner.

Adım 9 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
2   if (n == 0)
3     return 0;                  // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 3`

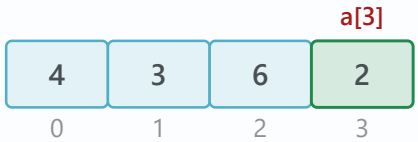
`dönen = 7`

ÇIKTI

ÇAĞRI YIĞINI

- **`toplam(a,3)`**
- `toplam(a,4)`
- `main()`

Kod İzleme: `toplam(a, 4)`



`toplam(a, 4) = 13 + 2 = 15`

dönen değerler: 0 → 4 → 7 → 13 → 15

toplam(a, 3) 13 döndürdü → 13 + a[3] = 13 + 2 = **15** döner.

Adım 10 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
2   if (n == 0)
3     return 0;                    // taban durum
▶   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
8 System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

`n = 4`

`dönen = 13`

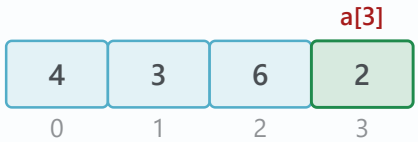
ÇAĞRI YIĞINI

`toplam(a,4)`

`main()`

ÇIKTI

Kod İzleme: `toplam(a, 4)`



`toplam(a, 4) = 13 + 2 = 15`

dönen değerler: 0 → 4 → 7 → 13 → 15

Sonuç **15**. 5 çağrı (n = 4, 3, 2, 1, 0): zaman O(n), en çok 5 çerçeve (+ main): bellek O(n).

Adım 11 / 11

```
1 int toplam(int[] a, int n) {   // ilk n öğenin toplamı
2   if (n == 0)
3     return 0;                  // taban durum
4   return toplam(a, n - 1) + a[n - 1];
5 }
6 // main içinde:
7 int[] a = {4, 3, 6, 2};
▶ System.out.println(toplam(a, 4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

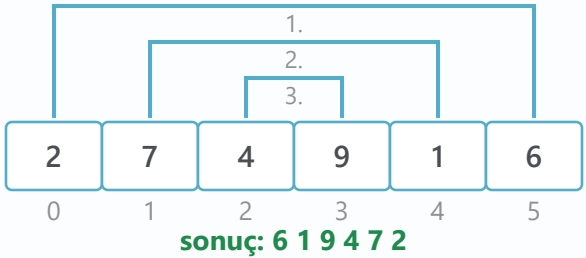
`main()`

ÇIKTI

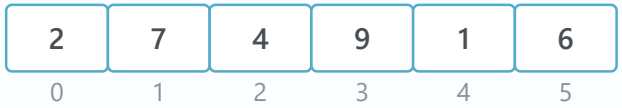
```
15
```

Diziyi Tersine Çevirme

- Fikir: dizinin iki **ucunu** değiştir, aradaki parçayı **aynı yöntemle** tersine çevir.
- Parça `a[i..j]` indisleriyle verilir; ilk çağrı `ters(a, 0, n - 1)`.
- Taban durum: `i >= j`, yani parçada 0 ya da 1 öğe kaldı; yapılacak iş yok.
- Her çağrı parçayı 2 öğe kısaltır: n öğe için ⌊n/2⌋ değiştirme, ⌊n/2⌋ + 1 çağrı → **O(n)**.
- Dizi yerinde değişir; metot bir değer döndürmez (`void`).



Kod İzleme: `ters(a, 0, 5)`



`a = {2, 7, 4, 9, 1, 6}`

main, `ters(a, 0, 5)` 'i çağırır.

Adım 1 / 6

```
1 void ters(int[] a, int i, int j) {
2   if (i >= j)
3     return;                    // taban: 0 ya da 1 öğe
4   int t = a[i]; a[i] = a[j]; a[j] = t;  // uçlar
5   ters(a, i + 1, j - 1);       // içteki parça
6 }
7 // main içinde:
8 int[] a = {2, 7, 4, 9, 1, 6};
▶ ters(a, 0, a.length - 1);
```

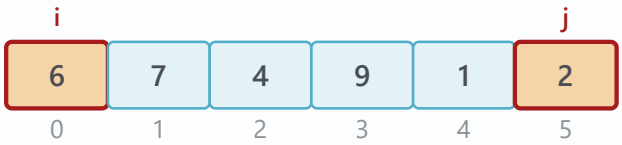
DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `ters(a, 0, 5)`



a[0] ↔ a[5] değişti

i = 0 < j = 5 → a[0] ile a[5] değişir: dizi **6 7 4 9 1 2**. Sonra `ters(a, 1, 4)` çağrılır.

Adım 2 / 6

```
1 void ters(int[] a, int i, int j) {
▶   if (i >= j)
3     return;                    // taban: 0 ya da 1 öğe
▶   int t = a[i]; a[i] = a[j]; a[j] = t;  // uçlar
▶   ters(a, i + 1, j - 1);       // içteki parça
6 }
7 // main içinde:
8 int[] a = {2, 7, 4, 9, 1, 6};
9 ters(a, 0, a.length - 1);
```

DEĞİŞKENLER

`i = 0`

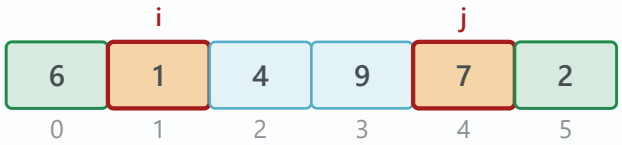
`j = 5`

ÇAĞRI YIĞINI

`ters(0,5)`

`main()`

Kod İzleme: `ters(a, 0, 5)`



`a[1] ⇔ a[4]` değişti

i = 1 < j = 4 → a[1] ile a[4] değişir: dizi **6 1 4 9 7 2**. Sonra `ters(a, 2, 3)` çağrılır.

Adım 3 / 6

```
1 void ters(int[] a, int i, int j) {
▶   if (i >= j)
3     return;                    // taban: 0 ya da 1 öğe
▶   int t = a[i]; a[i] = a[j]; a[j] = t;  // uçlar
▶   ters(a, i + 1, j - 1);       // içteki parça
6 }
7 // main içinde:
8 int[] a = {2, 7, 4, 9, 1, 6};
9 ters(a, 0, a.length - 1);
```

DEĞİŞKENLER

`i = 1`

`j = 4`

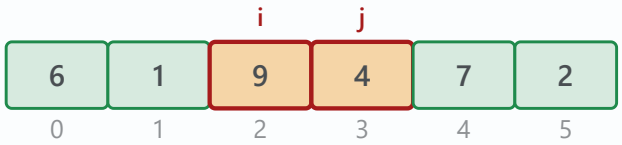
ÇAĞRI YIĞINI

`ters(1,4)`

`ters(0,5)`

`main()`

Kod İzleme: `ters(a, 0, 5)`



a[2] ⇔ a[3] değişti

i = 2 < j = 3 → a[2] ile a[3] değişir: dizi **6 1 9 4 7 2**. Sonra `ters(a, 3, 2)` çağrılır.

Adım 4 / 6

```
1 void ters(int[] a, int i, int j) {
▶   if (i >= j)
3     return;                    // taban: 0 ya da 1 öğe
▶   int t = a[i]; a[i] = a[j]; a[j] = t;  // uçlar
▶   ters(a, i + 1, j - 1);       // içteki parça
6 }
7 // main içinde:
8 int[] a = {2, 7, 4, 9, 1, 6};
9 ters(a, 0, a.length - 1);
```

DEĞİŞKENLER

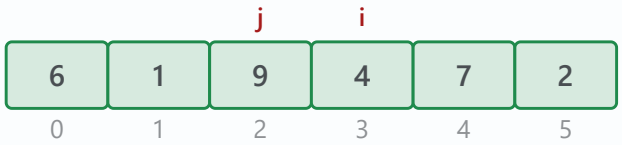
`i = 2`

`j = 3`

ÇAĞRI YIĞINI

- **`ters(2,3)`**
- `ters(1,4)`
- `ters(0,5)`
- `main()`

Kod İzleme: `ters(a, 0, 5)`



`i = 3 ≥ j = 2 → taban durum`

i = 3 ≥ j = 2: **taban durum**, ortada çevrilecek öğe kalmadı. Yığında main + 4 çerçeve.

Adım 5 / 6

```
1 void ters(int[] a, int i, int j) {
▶   if (i >= j)
▶     return;                    // taban: 0 ya da 1 öğe
4   int t = a[i]; a[i] = a[j]; a[j] = t;  // uçlar
5   ters(a, i + 1, j - 1);       // içteki parça
6 }
7 // main içinde:
8 int[] a = {2, 7, 4, 9, 1, 6};
9 ters(a, 0, a.length - 1);
```

DEĞİŞKENLER

- `i = 3`
- `j = 2`

ÇAĞRI YIĞINI

- `ters(3,2)`
- `ters(2,3)`
- `ters(1,4)`
- `ters(0,5)`
- `main()`

Kod İzleme: `ters(a, 0, 5)`



tersine çevrilmiş dizi

Çerçeveler sırayla döner; dönüşte yapılacak iş yok. Sonuç: **6 1 9 4 7 2**. 3 değiştirme, 4 çağrı.

Adım 6 / 6

```
1 void ters(int[] a, int i, int j) {
2   if (i >= j)
3     return;                    // taban: 0 ya da 1 öğe
4   int t = a[i]; a[i] = a[j]; a[j] = t;  // uçlar
5   ters(a, i + 1, j - 1);       // içteki parça
6 }
7 // main içinde:
8 int[] a = {2, 7, 4, 9, 1, 6};
▶ ters(a, 0, a.length - 1);
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

Üs Alma: Yavaş ve Hızlı

Yavaş: $x^n = x \cdot x^{n-1}$, $x^0 = 1$

```
long us(long x, int n) {
  if (n == 0) return 1;
  return x * us(x, n - 1);
}
```

Hızlı: üssü her adımda yarıya indir

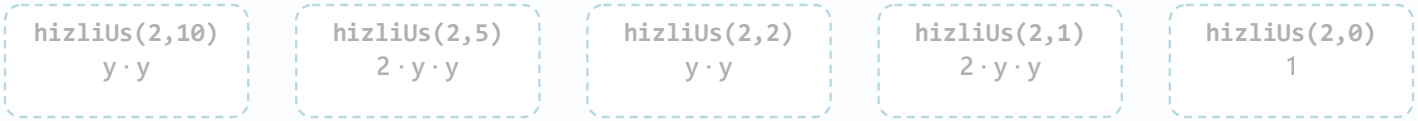
$x^n = (x^{n/2})^2$ (n çift)
$x^n = x \cdot (x^{(n-1)/2})^2$ (n tek)

$2^{10} = (2^5)^2$, $2^5 = 2 \cdot (2^2)^2$,
$2^2 = (2^1)^2$, $2^1 = 2 \cdot (2^0)^2$

n	us: çağrı	hizliUs: çağrı
10	11	5
100	101	8
1.000	1.001	11
10^6	1.000.001	21

Yavaş sürüm n + 1 çağrı yapar: **O(n)**. Hızlı sürümde n her çağrıda yarıya iner: $\lfloor \log_2 n \rfloor + 2$ çağrı, **O(log n)**.

Kod İzleme: `hizliUs(2, 10)`



main, `hizliUs(2, 10)`'u çağırır: 2^{10}.

Adım 1 / 11

```
1 long hizliUs(long x, int n) {
2   if (n == 0) return 1;            // taban durum
3   long y = hizliUs(x, n / 2);      // x^(n/2)
4   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
▶ System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `hizliUs(2, 10)`

`hizliUs(2,10)` y · y | `hizliUs(2,5)` 2 · y · y | `hizliUs(2,2)` y · y | `hizliUs(2,1)` 2 · y · y | `hizliUs(2,0)` 1

→ çağrı · ← dönüş

n = 10 ≠ 0 → önce y = `hizliUs(2, 5)` hesaplanır (10 / 2 = 5).

Adım 2 / 11

```
1 long hizliUs(long x, int n) {
▶   if (n == 0) return 1;            // taban durum
▶   long y = hizliUs(x, n / 2);      // x^(n/2)
4   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
8 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

x = 2

n = 10

ÇIKTI

ÇAĞRI YIĞINI

`hizliUs(2,10)`

`main()`

Kod İzleme: `hizliUs(2, 10)`



n = 5 ≠ 0 → önce y = `hizliUs(2, 2)` hesaplanır (5 / 2 = 2).

Adım 3 / 11

```
1 long hizliUs(long x, int n) {
▶   if (n == 0) return 1;            // taban durum
▶   long y = hizliUs(x, n / 2);      // x^(n/2)
4   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
8 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

`n = 5`

ÇIKTI

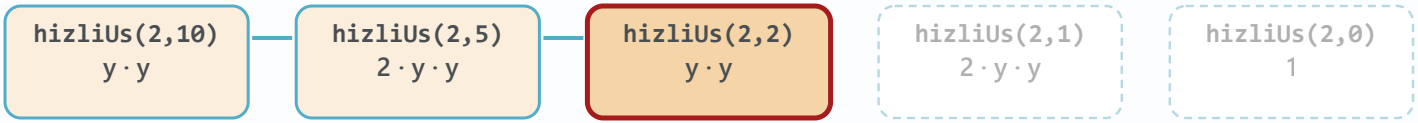
ÇAĞRI YIĞINI

`hizliUs(2,5)`

`hizliUs(2,10)`

`main()`

Kod İzleme: `hizliUs(2, 10)`



→ çağrı · ← dönüş

n = 2 ≠ 0 → önce y = `hizliUs(2, 1)` hesaplanır (2 / 2 = 1).

Adım 4 / 11

```
1 long hizliUs(long x, int n) {
▶   if (n == 0) return 1;            // taban durum
▶   long y = hizliUs(x, n / 2);      // x^(n/2)
4   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
8 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

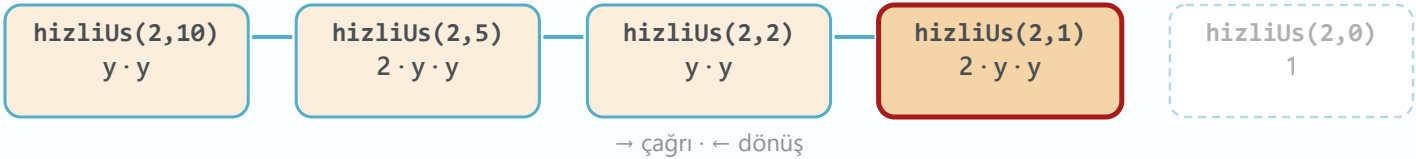
`n = 2`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `hizliUs(2, 10)`



n = 1 ≠ 0 → önce y = `hizliUs(2, 0)` hesaplanır (1 / 2 = 0).

Adım 5 / 11

```
1 long hizliUs(long x, int n) {
▶   if (n == 0) return 1;            // taban durum
▶   long y = hizliUs(x, n / 2);      // x^(n/2)
4   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
8 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

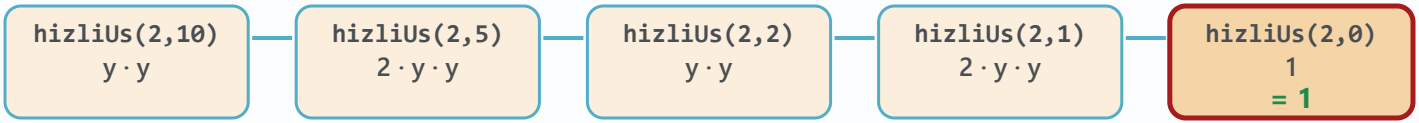
`n = 1`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `hizliUs(2, 10)`



→ çağrı · ← dönüş

n = 0: **taban durum** → 1 döner ($2^0 = 1$).

Adım 6 / 11

```
1 long hizliUs(long x, int n) {
▶   if (n == 0) return 1;            // taban durum
3   long y = hizliUs(x, n / 2);      // x^(n/2)
4   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
8 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

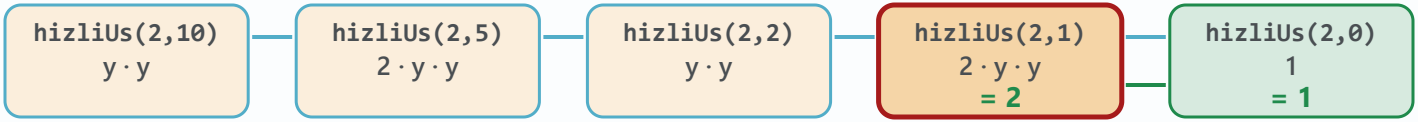
`n = 0`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `hizliUs(2, 10)`



y = 1 geldi. n = 1 tek → 2 · y · y = 2 · 1 · 1 = **2** döner.

Adım 7 / 11

```
1  long hizliUs(long x, int n) {
2    if (n == 0) return 1;            // taban durum
3    long y = hizliUs(x, n / 2);      // x^(n/2)
4    if (n % 2 == 0) return y * y;    // n çift
▶    return x * y * y;                // n tek
6  }
7  // main içinde:
8  System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

`n = 1`

`y = 1`

ÇIKTI

ÇAĞRI YIĞINI

`hizliUs(2,1)`

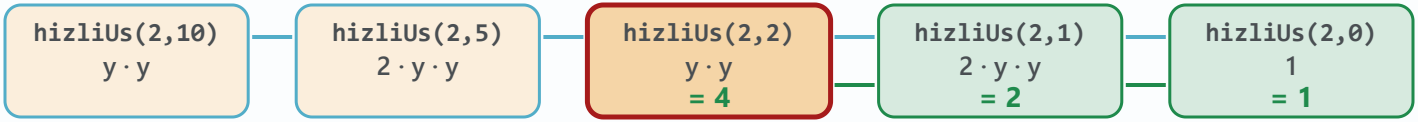
`hizliUs(2,2)`

`hizliUs(2,5)`

`hizliUs(2,10)`

`main()`

Kod İzleme: `hizliUs(2, 10)`



y = 2 geldi. n = 2 çift → y · y = 2 · 2 = **4** döner.

Adım 8 / 11

```
1 long hizliUs(long x, int n) {
2   if (n == 0) return 1;            // taban durum
3   long y = hizliUs(x, n / 2);      // x^(n/2)
▶   if (n % 2 == 0) return y * y;    // n çift
5   return x * y * y;                // n tek
6 }
7 // main içinde:
8 System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

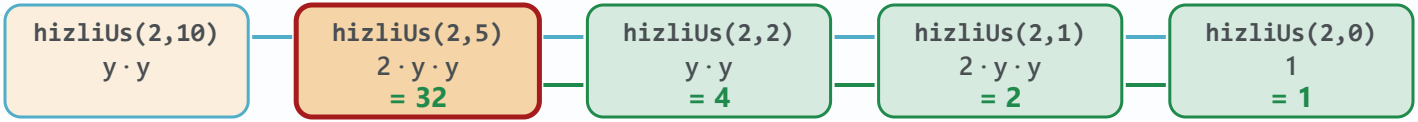
- x = 2
- n = 2
- y = 2

ÇIKTI

ÇAĞRI YIĞINI

- **hizliUs(2,2)**
- hizliUs(2,5)
- hizliUs(2,10)
- main()

Kod İzleme: `hizliUs(2, 10)`



y = 4 geldi. n = 5 tek → 2 · y · y = 2 · 4 · 4 = **32** döner.

Adım 9 / 11

```
1  long hizliUs(long x, int n) {
2    if (n == 0) return 1;            // taban durum
3    long y = hizliUs(x, n / 2);      // x^(n/2)
4    if (n % 2 == 0) return y * y;    // n çift
▶    return x * y * y;                // n tek
6  }
7  // main içinde:
8  System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

`n = 5`

`y = 4`

ÇIKTI

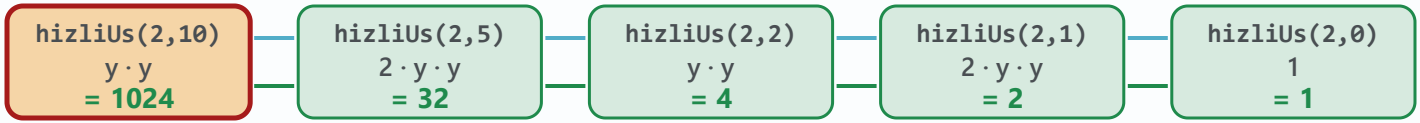
ÇAĞRI YIĞINI

`hizliUs(2,5)`

`hizliUs(2,10)`

`main()`

Kod İzleme: `hizliUs(2, 10)`



y = 32 geldi. n = 10 çift → y · y = 32 · 32 = **1024** döner.

Adım 10 / 11

```
1  long hizliUs(long x, int n) {
2    if (n == 0) return 1;            // taban durum
3    long y = hizliUs(x, n / 2);      // x^(n/2)
▶    if (n % 2 == 0) return y * y;    // n çift
5    return x * y * y;                // n tek
6  }
7  // main içinde:
8  System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

`x = 2`

`n = 10`

`y = 32`

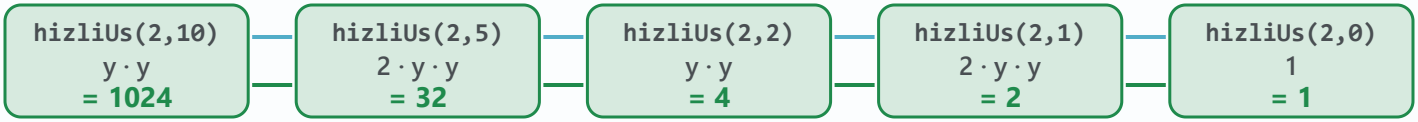
ÇIKTI

ÇAĞRI YIĞINI

`hizliUs(2,10)`

`main()`

Kod İzleme: `hizliUs(2, 10)`



→ çağrı · ← dönüş

Sonuç **1024**. Yalnızca 5 çağrı; yavaş `us(2, 10)` 11 çağrı yapardı.

Adım 11 / 11

```
1  long hizliUs(long x, int n) {
2    if (n == 0) return 1;            // taban durum
3    long y = hizliUs(x, n / 2);      // x^(n/2)
4    if (n % 2 == 0) return y * y;    // n çift
5    return x * y * y;                // n tek
6  }
7  // main içinde:
▶  System.out.println(hizliUs(2, 10));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

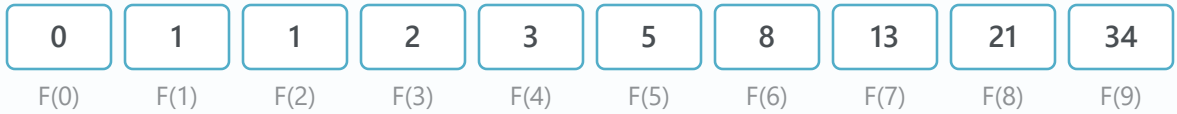
`main()`

ÇIKTI

```
1024
```

İKİLİ ÖZYİNELEME

Fibonacci Sayıları



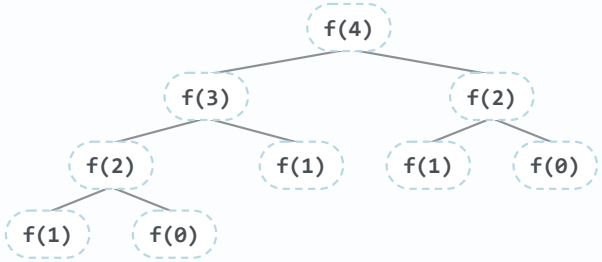

```
int fib(int n) {
  if (n <= 1) return n;     // taban durum
  return fib(n - 1) + fib(n - 2);
}
```

$F(0) = 0, F(1) = 1$

$F(n) = F(n-1) + F(n-2) \quad (n \geq 2)$

- **İkili özyineleme:** bir çağrı **iki** özyinelemeli çağrı yapar.
- Çağrılar artık zincir değil, bir **çağrı ağacı** oluşturur: her düğüm bir çağrı, çocukları onun yaptığı çağrılar.
- Tanım koda doğrudan dökülür; iki taban durum vardır: $n = 0$ ve $n = 1$.
- Ağaçta aynı değer birden çok kez hesaplanır; maliyeti bu belirler.

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

main, `fib(4)`'ü çağırır.

Adım 1 / 15

```
1 int fib(int n) {
2   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
▶ System.out.println(fib(4));
```

DEĞİŞKENLER

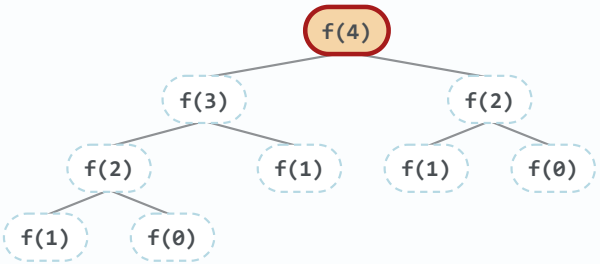
—

ÇIKTI

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(4)` : n > 1 → önce `fib(3)` çağrılır; `fib(2)` sırada bekler.

Adım 2 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

`n = 4`

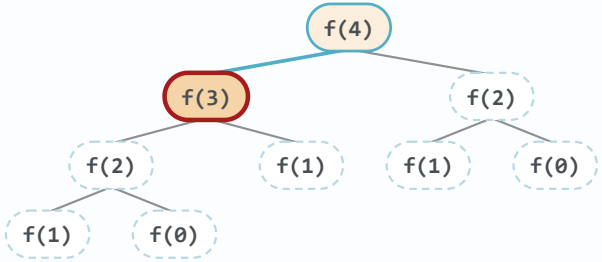
ÇIKTI

ÇAĞRI YIĞINI

`fib(4)`

`main()`

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(3)` : n > 1 → önce `fib(2)` çağrılır; `fib(1)` sırada bekler.

Adım 3 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

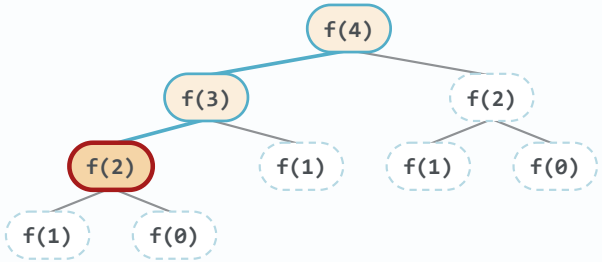
`n = 3`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(2)` : n > 1 → önce `fib(1)` çağrılır; `fib(0)` sırada bekler.

Adım 4 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

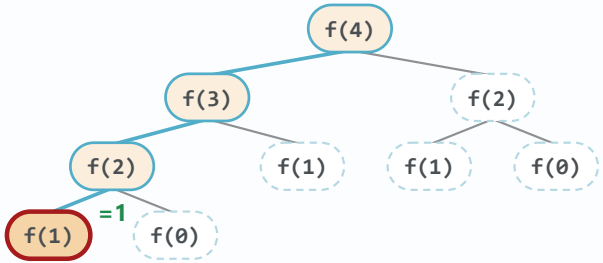
`n = 2`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(1)` : **taban durum** → 1 döner.

Adım 5 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

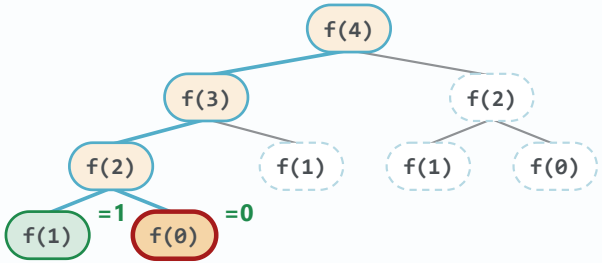
`n = 1`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

İkinci terim: `fib(0)` : **taban durum** → 0 döner.

Adım 6 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

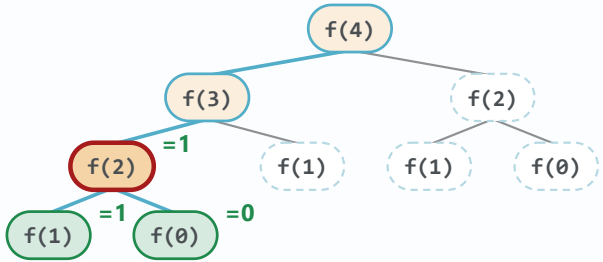
`n = 0`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(2)` = fib(1) + fib(0) = 1 + 0 = **1** döner.

Adım 7 / 15

```
1 int fib(int n) {
2   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

`n = 2`

`fib(1) = 1`

`fib(0) = 0`

ÇIKTI

ÇAĞRI YIĞINI

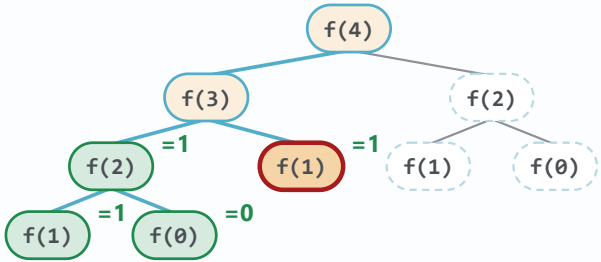
`fib(2)`

`fib(3)`

`fib(4)`

`main()`

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

İkinci terim: `fib(1)` : **taban durum** → 1 döner.

Adım 8 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

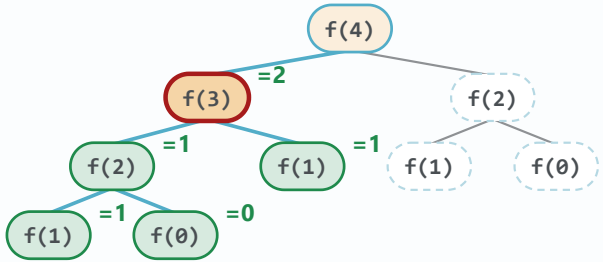
`n = 1`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(3)` = fib(2) + fib(1) = 1 + 1 = **2** döner.

Adım 9 / 15

```
1 int fib(int n) {
2   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

`n = 3`

`fib(2) = 1`

`fib(1) = 1`

ÇAĞRI YIĞINI

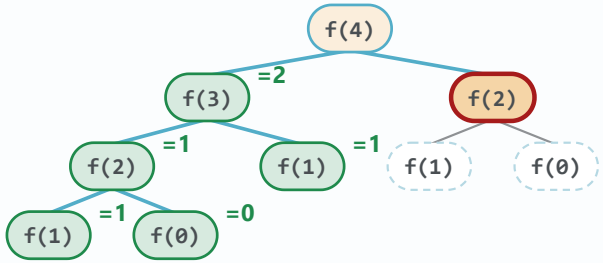
`fib(3)`

`fib(4)`

`main()`

ÇIKTI

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(4)` ikinci terim için `fib(2)`'yi çağırır: fib(2) **ikinci kez** hesaplanacak. Önce fib(1).

Adım 10 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

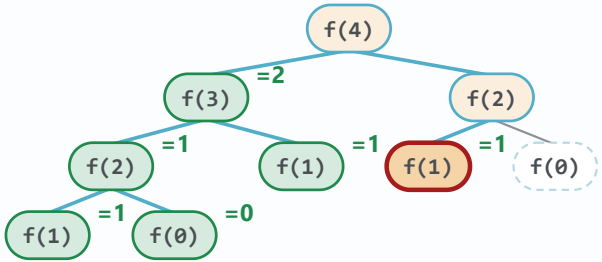
`n = 2`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(1)` : **taban durum** → 1 döner.

Adım 11 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

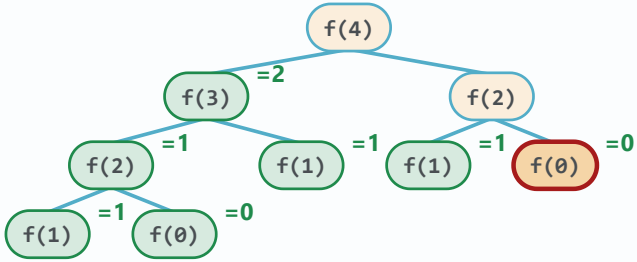
`n = 1`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

İkinci terim: `fib(0)` : **taban durum** → 0 döner.

Adım 12 / 15

```
1 int fib(int n) {
▶   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

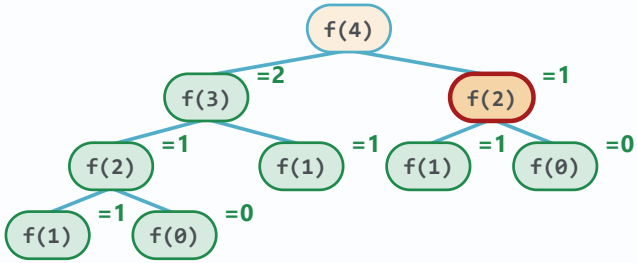
`n = 0`

ÇIKTI

ÇAĞRI YIĞINI

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

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(2)` = fib(1) + fib(0) = 1 + 0 = **1** döner.

Adım 13 / 15

```
1 int fib(int n) {
2   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

`n = 2`

`fib(1) = 1`

`fib(0) = 0`

ÇAĞRI YIĞINI

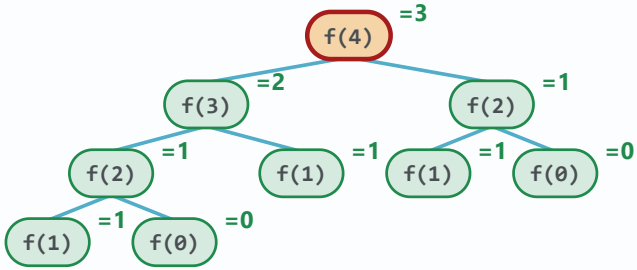
`fib(2)`

`fib(4)`

`main()`

ÇIKTI

Kod İzleme: `fib(4)` çağrı ağacı



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

`fib(4)` = fib(3) + fib(2) = 2 + 1 = **3** döner.

Adım 14 / 15

```
1 int fib(int n) {
2   if (n <= 1) return n;     // taban durum
▶   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
6 System.out.println(fib(4));
```

DEĞİŞKENLER

`n = 4`

`fib(3) = 2`

`fib(2) = 1`

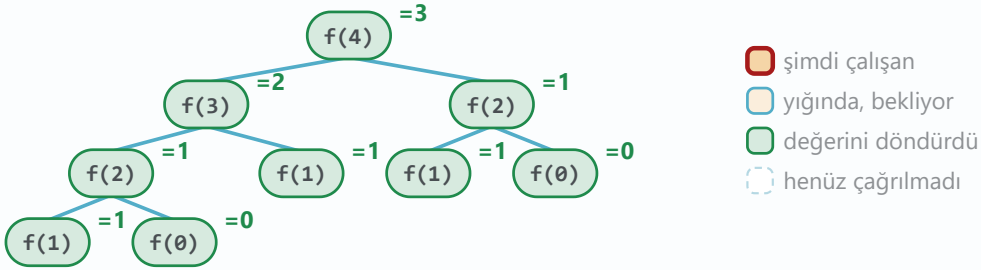
ÇIKTI

ÇAĞRI YIĞINI

`fib(4)`

`main()`

Kod İzleme: `fib(4)` çağrı ağacı



`fib(4)` = **3**. 9 çağrı: fib(2) iki kez, fib(1) üç kez, fib(0) iki kez hesaplandı.

Adım 15 / 15

```
1 int fib(int n) {
2   if (n <= 1) return n;     // taban durum
3   return fib(n - 1) + fib(n - 2);
4 }
5 // main içinde:
▶ System.out.println(fib(4));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

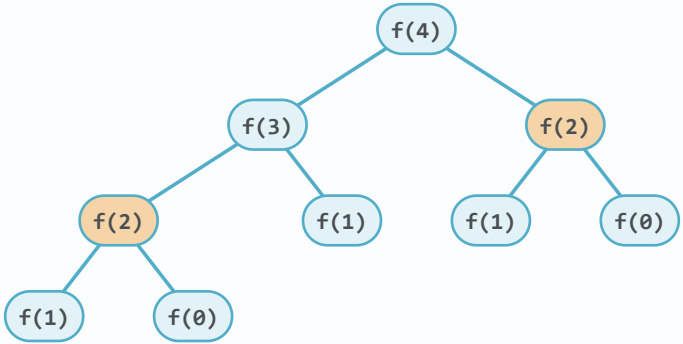
ÇIKTI

```
3
```

Özyinelemenin Maliyeti

```
int fib(int n) {
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}
```

- **faktoriyel**: T(n) = T(n − 1) + c → n çağrı → **O(n)** zaman, **O(n)** yığın belleği.
- **fib**: her çağrı **iki** çağrı yapar; aynı alt problemler tekrar tekrar çözülür.
- fib(4) → 9 çağrı; turuncu **f(2)** iki kez hesaplanıyor. Çağrı sayısı **üstel** büyür (≈ $1{,}6^n$).
- Çözüm: bellekleme ya da döngüyle hesaplama → O(n).



Tekrarlanan Hesaptan Kaçınmak

n	fib(n)	fib çağrı sayısı
5	5	15
10	55	177
20	6.765	21.891
30	832.040	2.692.537
40	102.334.155	331.160.281

Çağrı sayısı = 2 · F(n + 1) − 1: n her 1 arttığında yaklaşık 1,6 katına çıkar.

```
long[] bellek = new long[91];   // 0: hesaplanmadı
long fibBellek(int n) {
  if (n <= 1) return n;
  if (bellek[n] != 0) return bellek[n];
  bellek[n] = fibBellek(n - 1) + fibBellek(n - 2);
  return bellek[n];
}
```

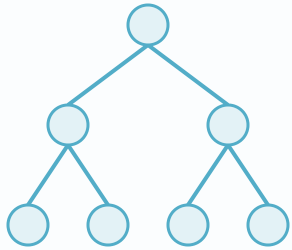
```
long fibDongu(int n) {
  long a = 0, b = 1;              // F(0), F(1)
  for (int i = 0; i < n; i++) {
    long t = a + b;  a = b;  b = t;
  }
  return a;
}
```

Hesaplanan değerler bir dizide saklanırsa her değer **bir kez** hesaplanır: `fibBellek(40)` 79 çağrı yapar. Döngülü sürüm O(n) zaman ve O(1) bellekle çalışır.

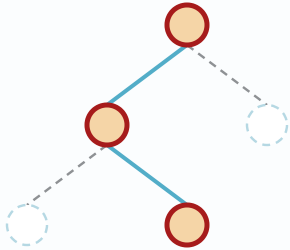
İKİLİ ÖZYİNELEME

Özyinelemeli İkili Arama

- Sıralı dizide x aranır: ortadaki öğeye bakılır; x küçükse sol yarıda, büyükse sağ yarıda **aynı arama** yapılır.
- İki taban durum: aralık boş (`lo > hi`) → −1; ortadaki öğe x → indisi.
- Kodda iki özyinelemeli çağrı yazılıdır ama her çağrı bunlardan **yalnızca birini** yapar. Fibonacci gibi ağaç oluşmaz; çağrılar bir zincirdir.
- Aralık her çağrıda yarıya iner: T(n) = T(n/2) + c → **O(log n)** zaman ve O(log n) yığın belleği.
- n = 10 için en çok 5, n = 10^6 için en çok 21 çağrı.



fib: iki çağrı → ağaç



ikiliAra: biri seçilir → zincir

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



main, 10 öğeli sıralı dizide 23'ü arar: `ikiliAra(a, 23, 0, 9)` .

Adım 1 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;   // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
▶ System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

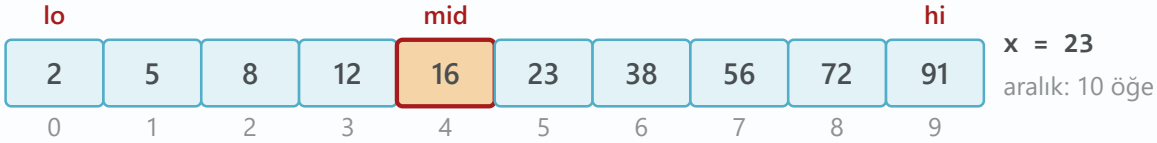
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



lo = 0 ≤ hi = 9: aralık boş değil. mid = (0 + 9) / 2 = **4**, a[4] = 16.

Adım 2 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
▶   if (lo > hi) return -1;       // taban: boş aralık
▶   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;  // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

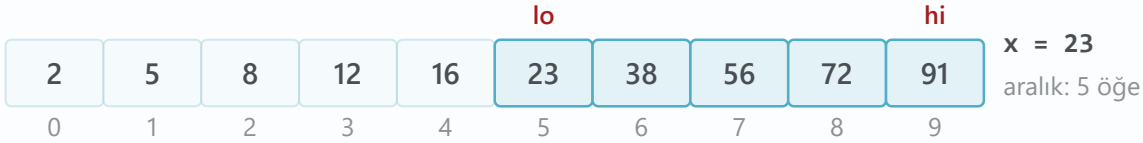
- `x = 23`
- `lo = 0`
- `hi = 9`
- `mid = 4`

ÇIKTI

ÇAĞRI YIĞINI

- `ikiliAra(0,9)`
- `main()`

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



23 > 16 → **sağ yarı**: `ikiliAra(a, 23, 5, 9)`. Sol yarıya hiç bakılmaz.

Adım 3 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
▶   if (a[mid] == x) return mid;  // taban: bulundu
▶   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
▶   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

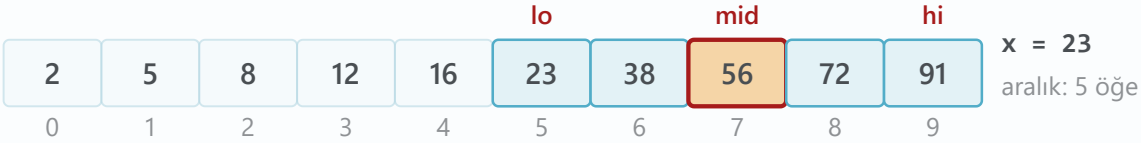
- `x = 23`
- `lo = 0`
- `hi = 9`
- `mid = 4`

ÇAĞRI YIĞINI

- `ikiliAra(0,9)`
- `main()`

ÇIKTI

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



lo = 5 ≤ hi = 9: aralık boş değil. mid = (5 + 9) / 2 = **7**, a[7] = 56.

Adım 4 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
▶   if (lo > hi) return -1;       // taban: boş aralık
▶   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;  // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

- `x = 23`
- `lo = 5`
- `hi = 9`
- `mid = 7`

ÇIKTI

ÇAĞRI YIĞINI

- `ikiliAra(5,9)`
- `ikiliAra(0,9)`
- `main()`

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



23 < 56 → **sol yarı**: `ikiliAra(a, 23, 5, 6)`. Sağ yarıya hiç bakılmaz.

Adım 5 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
▶   if (a[mid] == x) return mid;  // taban: bulundu
▶   if (x < a[mid])
▶     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

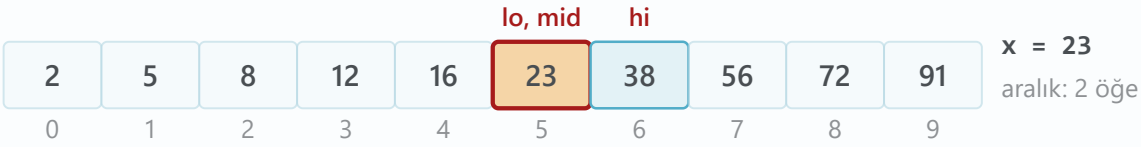
- `x = 23`
- `lo = 5`
- `hi = 9`
- `mid = 7`

ÇIKTI

ÇAĞRI YIĞINI

- `ikiliAra(5,9)`
- `ikiliAra(0,9)`
- `main()`

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



lo = 5 ≤ hi = 6: aralık boş değil. mid = (5 + 6) / 2 = **5**, a[5] = 23.

Adım 6 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
▶   if (lo > hi) return -1;       // taban: boş aralık
▶   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;  // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

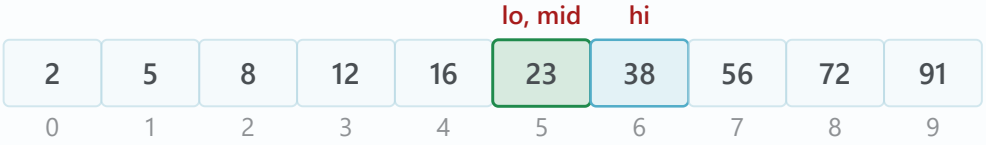
- `x = 23`
- `lo = 5`
- `hi = 6`
- `mid = 5`

ÇIKTI

ÇAĞRI YIĞINI

- `ikiliAra(5,6)`
- `ikiliAra(5,9)`
- `ikiliAra(0,9)`
- `main()`

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



a[5] = 23 = x: **taban durum** (bulundu) → 5 döner.

Adım 7 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
▶   if (a[mid] == x) return mid;   // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

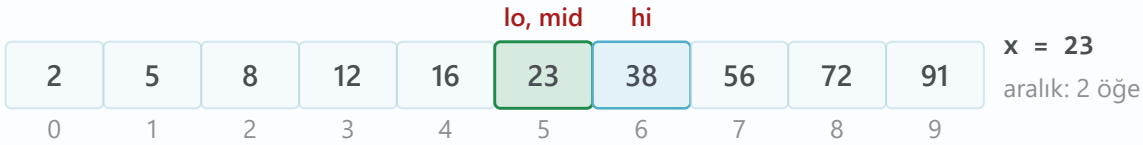
- `x = 23`
- `lo = 5`
- `hi = 6`
- `mid = 5`

ÇIKTI

ÇAĞRI YIĞINI

- `ikiliAra(5,6)`
- `ikiliAra(5,9)`
- `ikiliAra(0,9)`
- `main()`

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



`ikiliAra(a, 23, 5, 9)` aldığı 5'i `return` ile olduğu gibi iletir.

Adım 8 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;   // taban: bulundu
5   if (x < a[mid])
▶     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

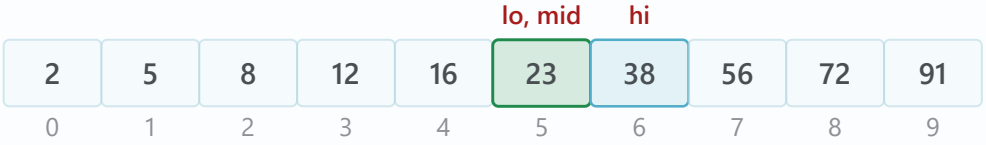
dönen = **5**

ÇIKTI

ÇAĞRI YIĞINI

- **`ikiliAra(5,9)`**
- `ikiliAra(0,9)`
- `main()`

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



`ikiliAra(a, 23, 0, 9)` de 5'i iletir. Dönüşte yeni hesap yapılmaz.

Adım 9 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;   // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
▶   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
9 System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

dönen = **5**

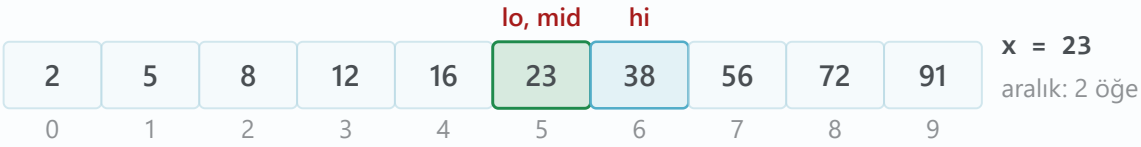
ÇIKTI

ÇAĞRI YIĞINI

ikiliAra(0,9)

main()

Kod İzleme: `ikiliAra(a, 23, 0, 9)`



Sonuç **5**. 3 çağrı; aralık 10 → 5 → 2 öğeye indi.

Adım 10 / 10

```
1 int ikiliAra(int[] a, int x, int lo, int hi) {
2   if (lo > hi) return -1;        // taban: boş aralık
3   int mid = (lo + hi) / 2;
4   if (a[mid] == x) return mid;   // taban: bulundu
5   if (x < a[mid])
6     return ikiliAra(a, x, lo, mid - 1); // sol yarı
7   return ikiliAra(a, x, mid + 1, hi);   // sağ yarı
8 }
▶ System.out.println(ikiliAra(a, 23, 0, 9));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

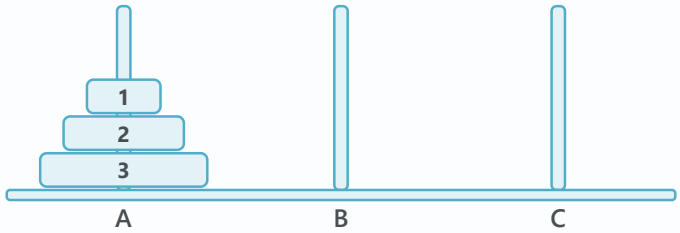
ÇIKTI

```
5
```

Hanoi Kuleleri (Towers of Hanoi)

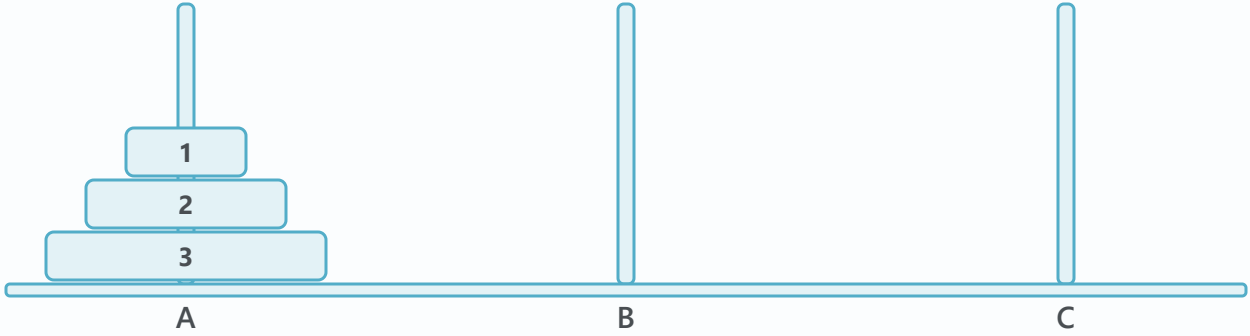
Bilgisayar bilimi ve matematikte klasik bir problem: bir disk yığını, kurallara uyularak kaynak çubuktan hedef çubuğa taşınır.

- Bir hamlede yalnızca **bir disk** taşınır (bir çubuğun en üstündeki).
- Bir disk yalnızca **daha büyük bir diskin** üstüne ya da boş çubuğa konabilir.
- Amaç: yardımcı çubuğu kullanarak tüm diskleri kaynaktan (A) hedefe (C) taşımak.



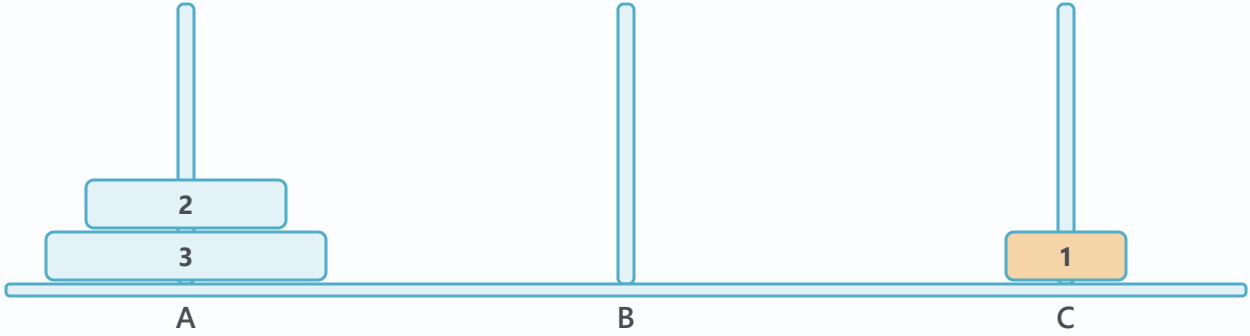
Özyinelemeli fikir: n−1 diski A'dan B'ye taşı → en büyük diski A'dan C'ye taşı → n−1 diski B'den C'ye taşı. Hamle sayısı T(n) = 2T(n−1) + 1 = **$2^n - 1$**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



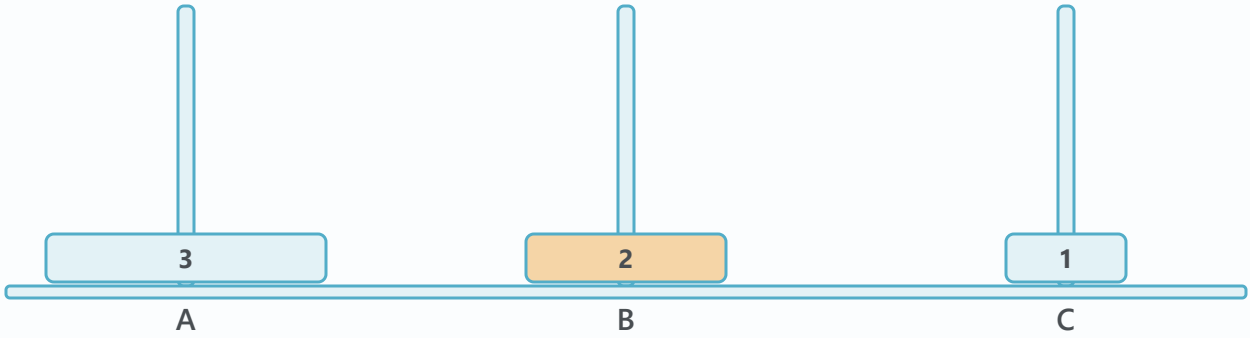
Başlangıç: 3 disk A çubuğunda (büyük altta). Hedef: hepsini C'ye taşımak.

Hanoi Kuleleri: 3 Diskle 7 Hamle



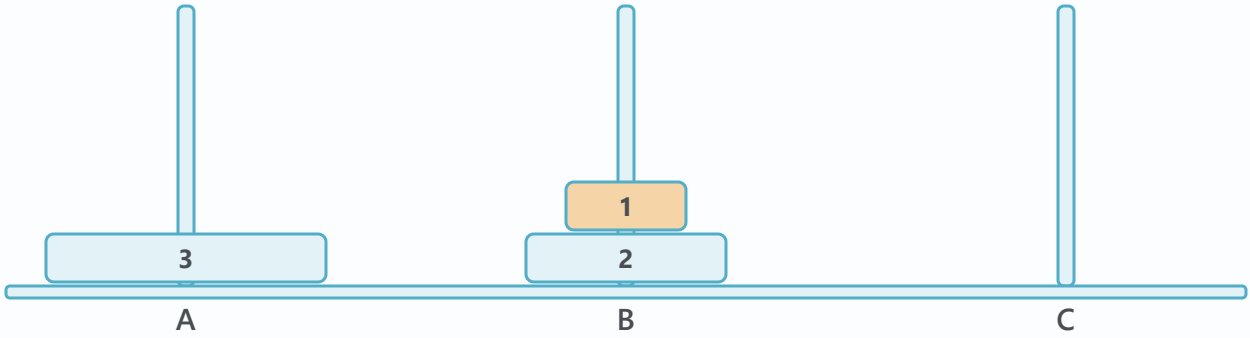
Hamle 1: disk 1, **A** → **C**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



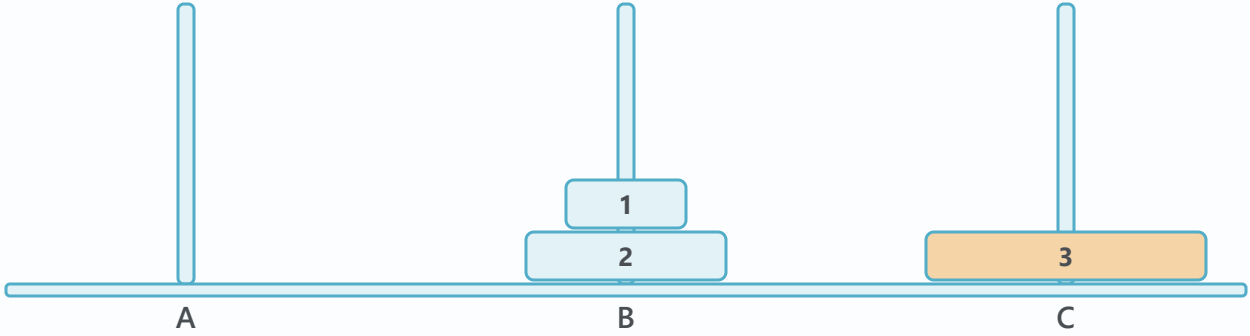
Hamle 2: disk 2, **A** → **B**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



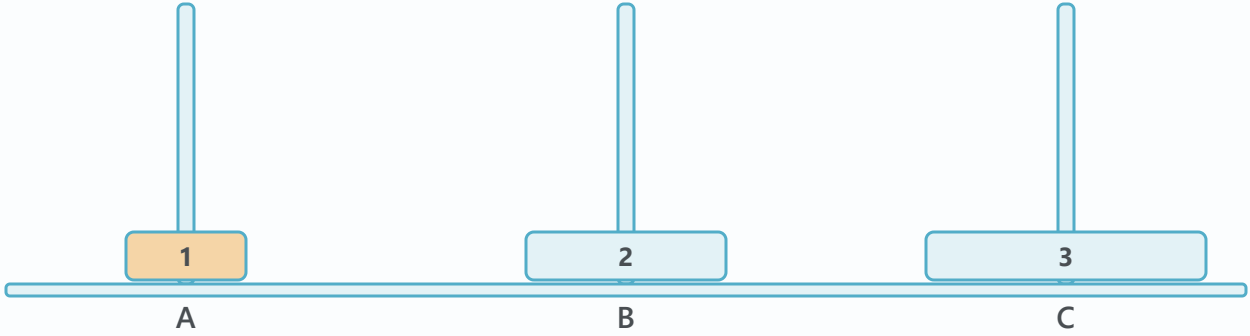
Hamle 3: disk 1, **C** → **B**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



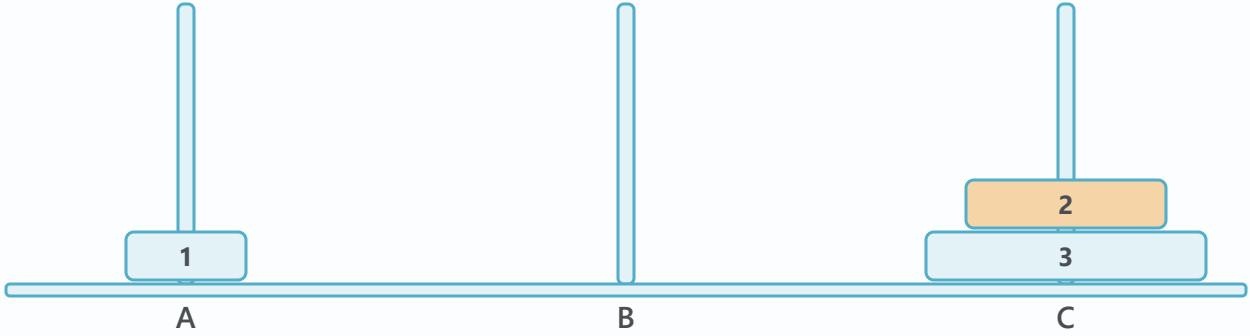
Hamle 4: disk 3, **A** → **C**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



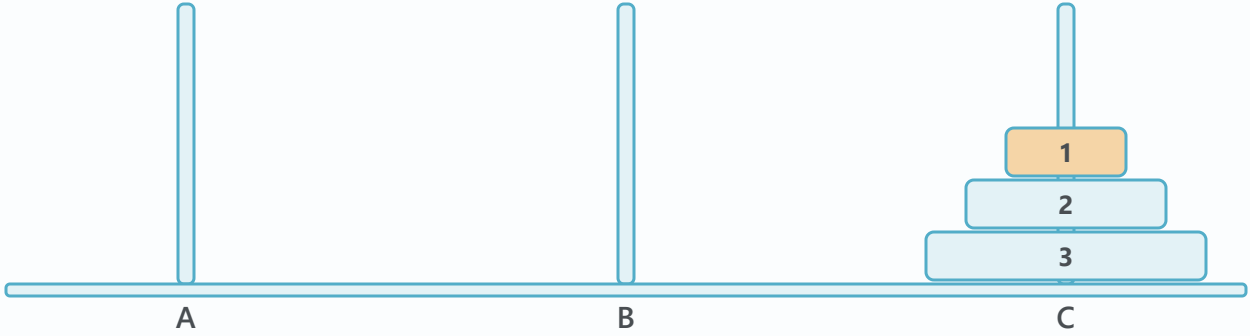
Hamle 5: disk 1, **B** → **A**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



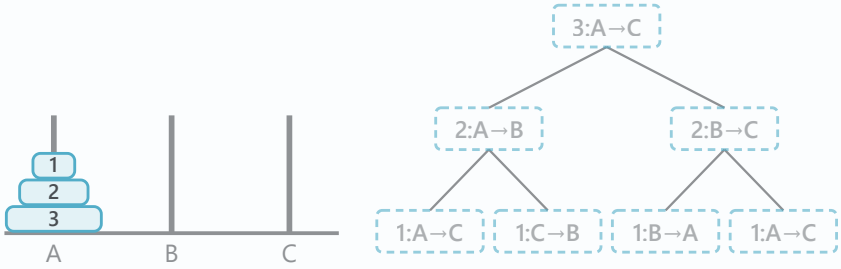
Hamle 6: disk 2, **B** → **C**.

Hanoi Kuleleri: 3 Diskle 7 Hamle



Hamle 7: disk 1, **A** → **C**. Bitti: $2^3 - 1 =$ **7 hamle**.

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



main, `hanoi(3, 'A', 'B', 'C')` çağırır. Ağaçtaki her kutu bir çağrı: **n:kaynak→hedef**.

Adım 1 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
▶  hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

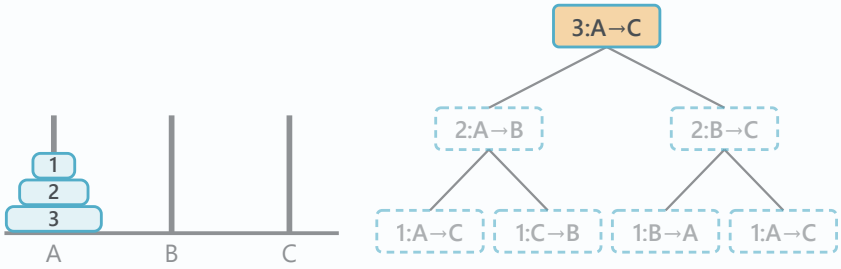
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



hanoi(3,A,B,C): yeni çerçeve açıldı. n = 3 ≠ 1 → önce üstteki 2 disk A çubuğundan B çubuğuna taşınmalı.

Adım 2 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 3` `k = A` `y = B` `h = C`

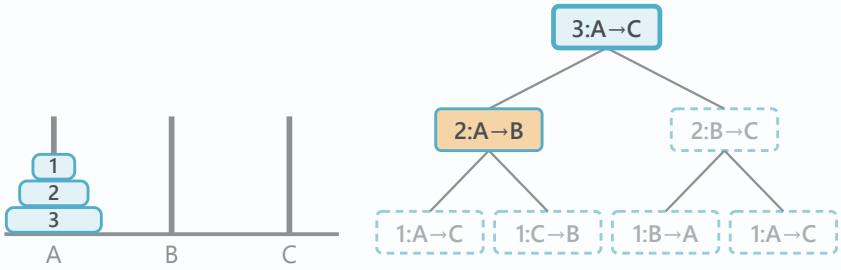
ÇAĞRI YIĞINI

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

`main()`

ÇIKTI

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Satır 6: hanoi(2,A,C,B): yeni çerçeve açıldı. n = 2 ≠ 1 → önce üstteki 1 disk A çubuğundan C çubuğuna taşınmalı.

Adım 3 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

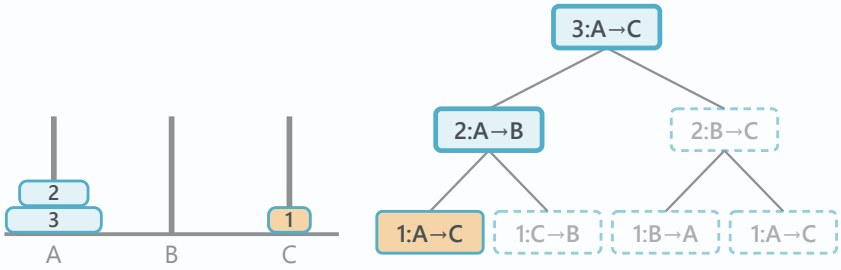
`n = 2` `k = A` `y = C` `h = B`

ÇAĞRI YIĞINI

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

ÇIKTI

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Satır 6: hanoi(1,A,B,C): yeni çerçeve açıldı. **n == 1** → taban durum: disk 1, **A** → **C**.

Adım 4 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
▶      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = A` `y = B` `h = C`

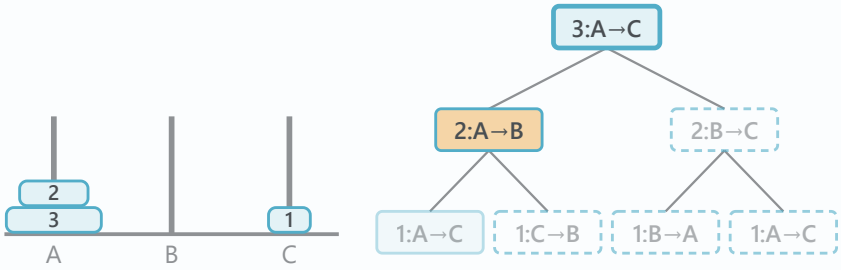
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



return: çerçeve yığından atılır, kontrol çağırana döner.

Adım 5 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
▶      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = A` `y = B` `h = C`

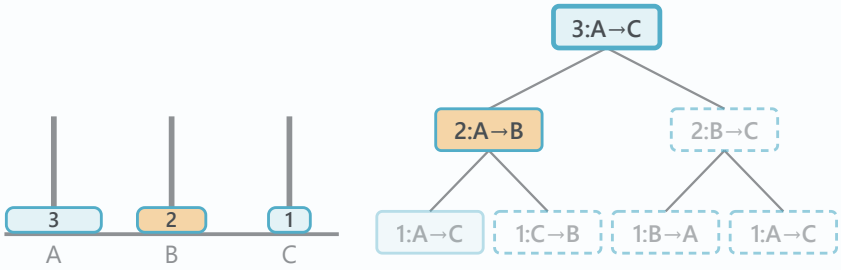
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Sol alt çağrı bitti; A çubuğunda yalnızca disk 2 kaldı → **Disk 2: A → B**.

Adım 6 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
▶    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 2` `k = A` `y = C` `h = B`

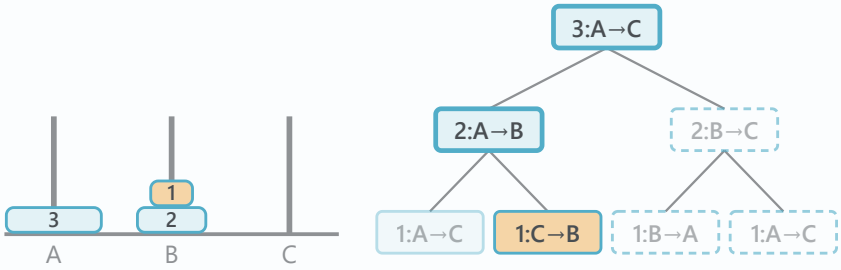
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Satır 8: hanoi(1,C,A,B): yeni çerçeve açıldı. **n == 1** → taban durum: disk 1, **C** → **B**.

Adım 7 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
▶      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = C` `y = A` `h = B`

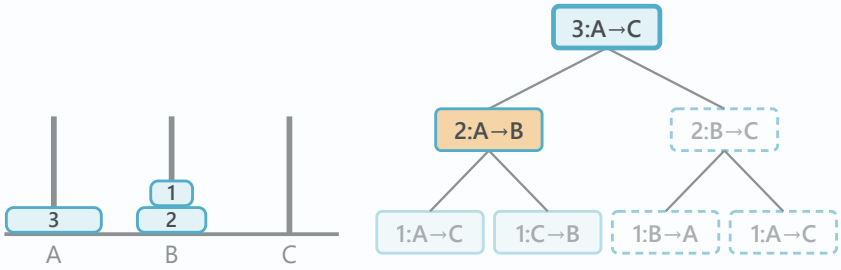
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



return: çerçeve yığından atılır, kontrol çağırana döner.

Adım 8 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
▶      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = C` `y = A` `h = B`

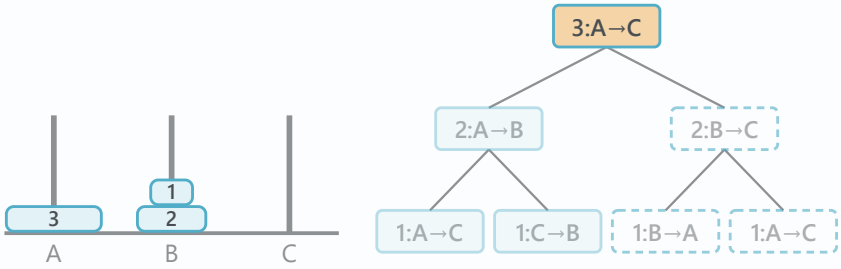
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



hanoi(2,A,C,B) biter; çerçevesi atılır.

Adım 9 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
▶ }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 2` `k = A` `y = C` `h = B`

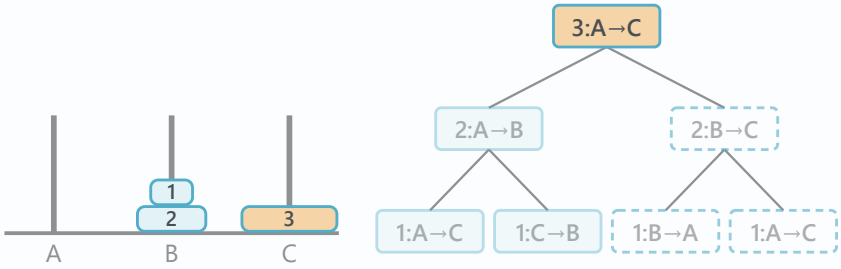
ÇAĞRI YIĞINI

- **hanoi(2,A,C,B)**
- hanoi(3,A,B,C)
- main()

ÇIKTI

```
1: A→C
2: A→B
1: C→B
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Sol alt çağrı bitti; A çubuğunda yalnızca disk 3 kaldı → **Disk 3: A → C**.

Adım 10 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
▶   System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 3` `k = A` `y = B` `h = C`

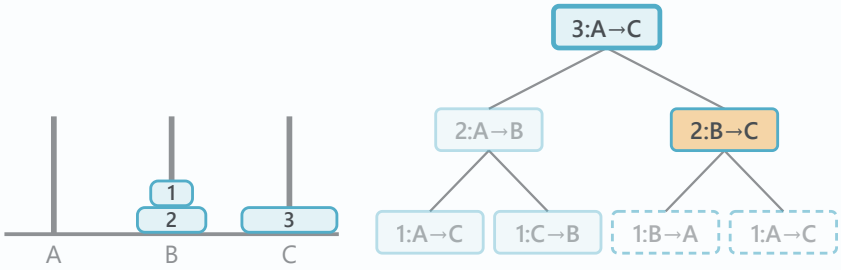
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Satır 8: hanoi(2,B,A,C): yeni çerçeve açıldı. n = 2 ≠ 1 → önce üstteki 1 disk B çubuğundan A çubuğuna taşınmalı.

Adım 11 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 2` `k = B` `y = A` `h = C`

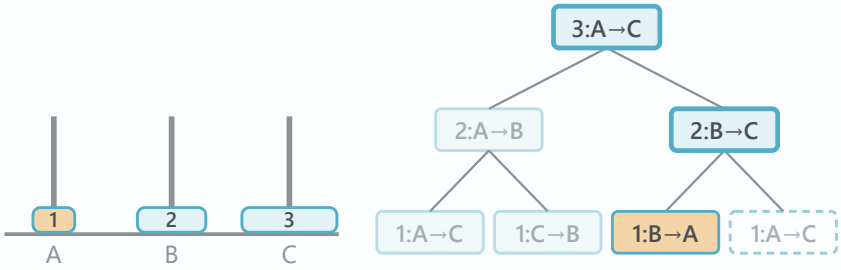
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Satır 6: hanoi(1,B,C,A): yeni çerçeve açıldı. **n == 1** → taban durum: disk 1, **B** → **A**.

Adım 12 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
▶      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = B` `y = C` `h = A`

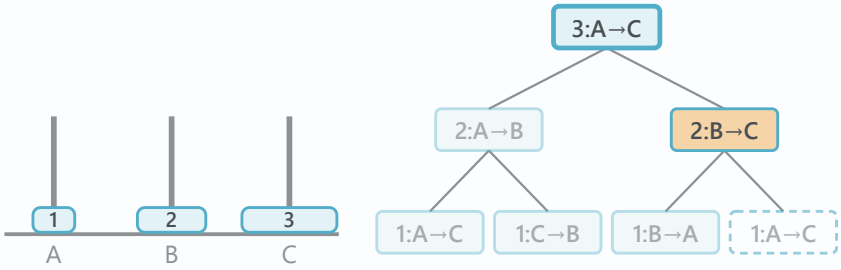
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



return: çerçeve yığından atılır, kontrol çağırana döner.

Adım 13 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
▶      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = B` `y = C` `h = A`

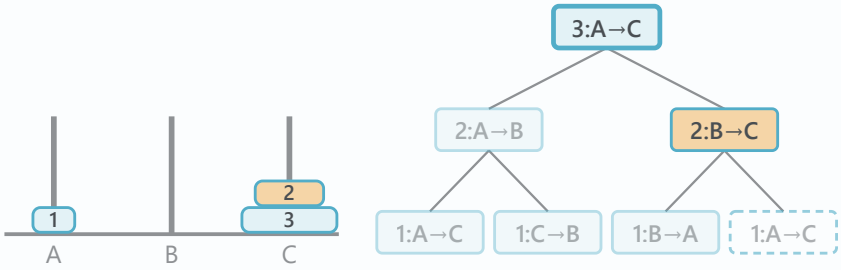
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Sol alt çağrı bitti; B çubuğunda yalnızca disk 2 kaldı → **Disk 2: B → C**.

Adım 14 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
▶    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 2` `k = B` `y = A` `h = C`

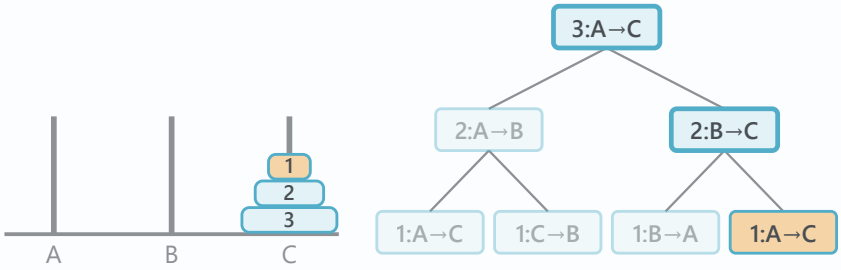
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
2: B→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



Satır 8: hanoi(1,A,B,C): yeni çerçeve açıldı. **n == 1** → taban durum: disk 1, **A** → **C**.

Adım 15 / 19

```
1  void hanoi(int n, char k, char y, char h) {
▶    if (n == 1) {
▶      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = A` `y = B` `h = C`

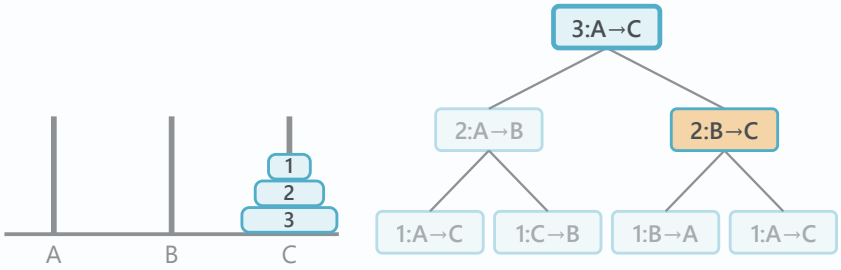
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
2: B→C
1: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



return: çerçeve yığından atılır, kontrol çağırana döner.

Adım 16 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
▶      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 1` `k = A` `y = B` `h = C`

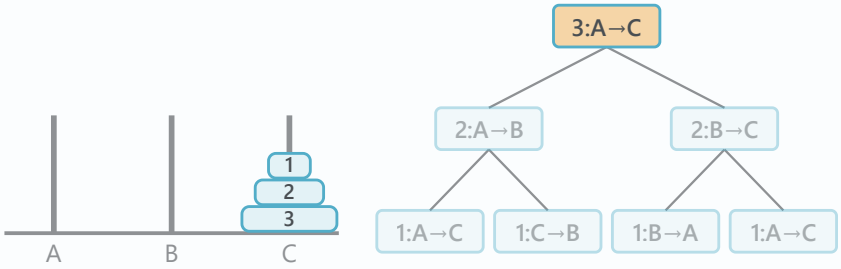
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
2: B→C
1: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



hanoi(2,B,A,C) biter; çerçevesi atılır.

Adım 17 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
▶ }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 2` `k = B` `y = A` `h = C`

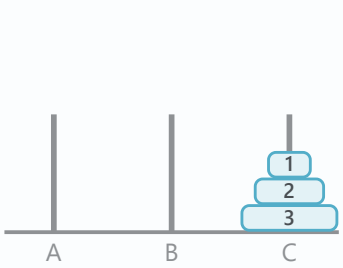
ÇAĞRI YIĞINI

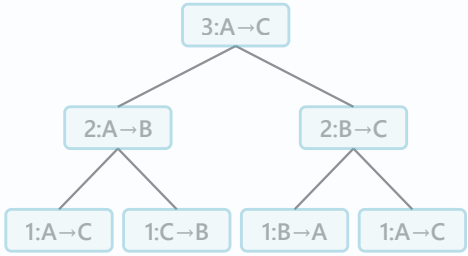
- **`hanoi(2,B,A,C)`**
- `hanoi(3,A,B,C)`
- `main()`

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
2: B→C
1: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`





hanoi(3,A,B,C) biter; çerçevesi atılır. Ağaçtaki 7 çağrı = 7 hamle = $2^3 - 1$.

Adım 18 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
▶ }
10 // main içinde:
11 hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

`n = 3` `k = A` `y = B` `h = C`

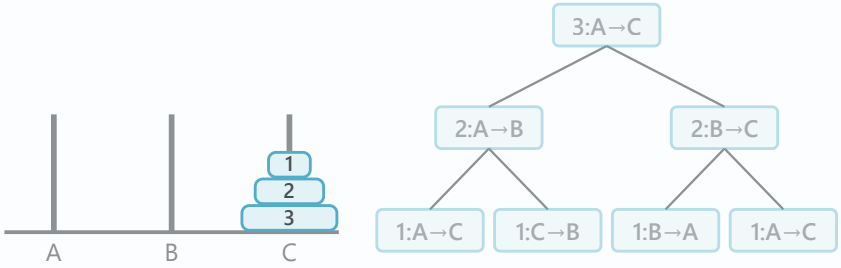
ÇAĞRI YIĞINI

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

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
2: B→C
1: A→C
```

Kod İzleme: Özyinelemeli `hanoi(3, A, B, C)`



main'e dönüldü: tüm diskler C'de. Çağrı yığınının en fazla derinliği n = 3 (+ main) → bellek **O(n)**, zaman **O(2ⁿ)**.

Adım 19 / 19

```
1  void hanoi(int n, char k, char y, char h) {
2    if (n == 1) {
3      System.out.println("1: " + k + "→" + h);
4      return;
5    }
6    hanoi(n - 1, k, h, y);
7    System.out.println(n + ": " + k + "→" + h);
8    hanoi(n - 1, y, k, h);
9  }
10 // main içinde:
▶  hanoi(3, 'A', 'B', 'C');
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

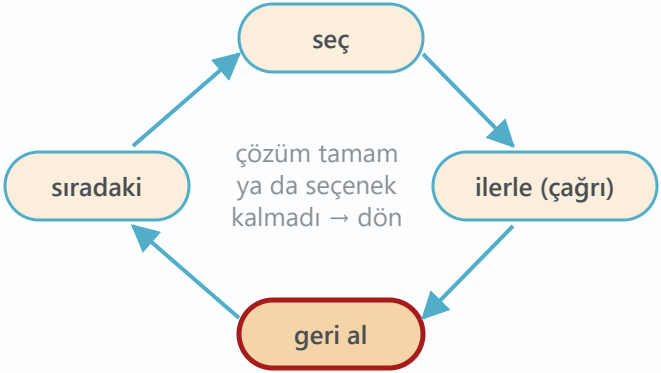
`main()`

ÇIKTI

```
1: A→C
2: A→B
1: C→B
3: A→C
1: B→A
2: B→C
1: A→C
```

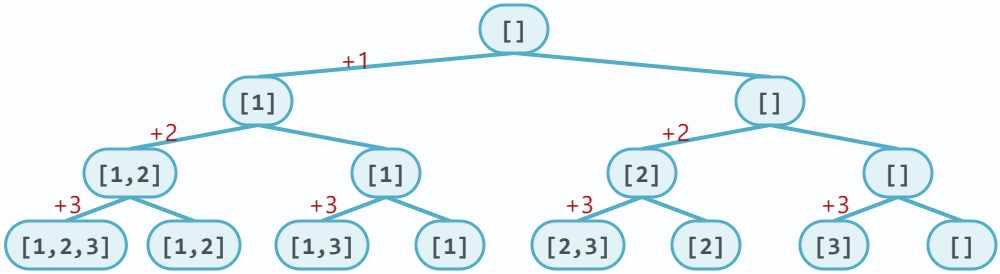
Geri İzleme (Backtracking)

- Çözüm **adım adım** kurulur; her adımda olası seçeneklerden biri denenir.
- Seçimle bir sonraki adıma geçilir: **özyinelemeli çağrı**.
- Çağrı döndüğünde seçim **geri alınır** ve sıradaki seçenek denenir. Böylece her dal aynı başlangıç durumunu görür.
- Bir seçim çözüme götüremeyecekse o dal bırakılır, daha derine inilmez.
- Örnekler: alt kümeleri ya da permütasyonları listeleme, labirentte yol bulma, sudoku, n-vezir.



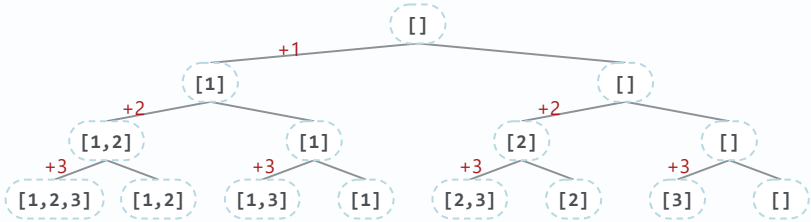
Hanoi'de olduğu gibi her çağrı birden çok özyinelemeli çağrı yapar; çağrı ağacı, denenen tüm seçimlerin ağacıdır (**karar ağacı**).

Alt Kümeleri Listeleme



- Her öğe için iki karar: **al** ya da **alma**. Sol dal öğeyi alır (+1, +2, +3), sağ dal almaz.
- i. düzeyde a[i] için karar verilir; yaprakta (i = n) tüm kararlar bitmiştir → s bir alt kümedir.
- n öğe → 2^n yaprak = 2^n alt küme; çağrı sayısı $2^{n+1} - 1$ (n = 3: 8 alt küme, 15 çağrı).
- Çıktı sırası: `[1, 2, 3] [1, 2] [1, 3] [1] [2, 3] [2] [3] []`

Kod İzleme: `altKumeler` ile {1, 2, 3}



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

main, a = {1, 2, 3} ve boş s ile `altKumeler(a, 0, s)` 'yi çağırır.

Adım 1 / 24

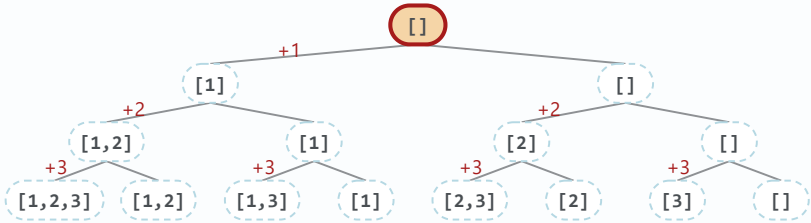
```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {              // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
▶  altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

ÇIKTI

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 0: a[0] = 1 **alınır** (s = [1]), sol dala inilir.

Adım 2 / 24

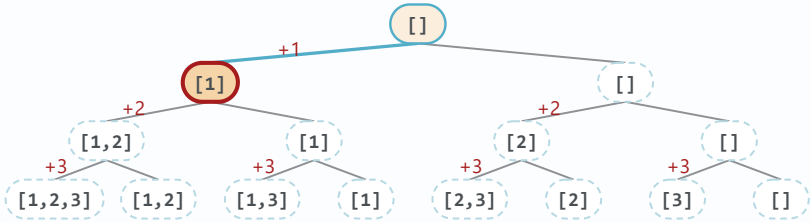
```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {              // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                      // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

ÇIKTI

ÇAĞRI YIĞINI

- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 1: a[1] = 2 **alınır** (s = [1, 2]), sol dala inilir.

Adım 3 / 24

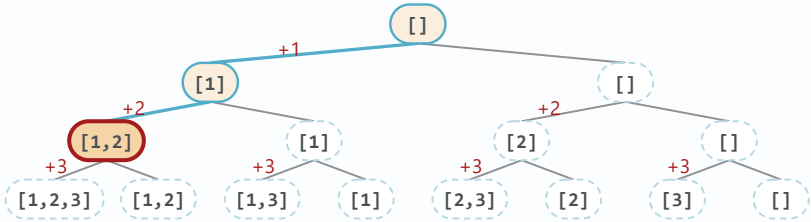
```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                        // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);             // geri al
8    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

ÇIKTI

ÇAĞRI YIĞINI

- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 2: a[2] = 3 **alınır** (s = [1, 2, 3]), sol dala inilir.

Adım 4 / 24

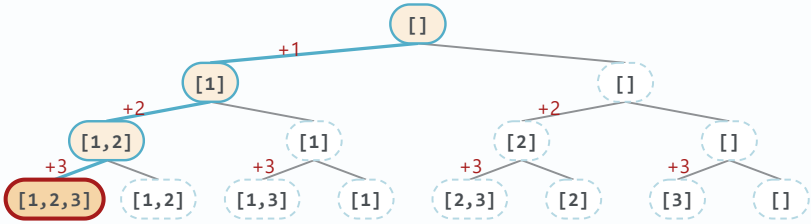
```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {              // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                      // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

ÇIKTI

ÇAĞRI YIĞINI

- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[1, 2, 3]` yazılır.

Adım 5 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

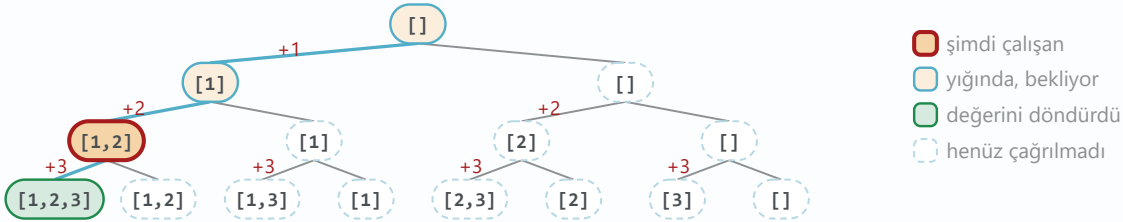
ÇIKTI

```
[1, 2, 3]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 3 **geri alınır** (s = [1, 2]); aynı i ile sağ dala, 3'siz alt kümelere inilir.

Adım 6 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                        // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);             // geri al
▶    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

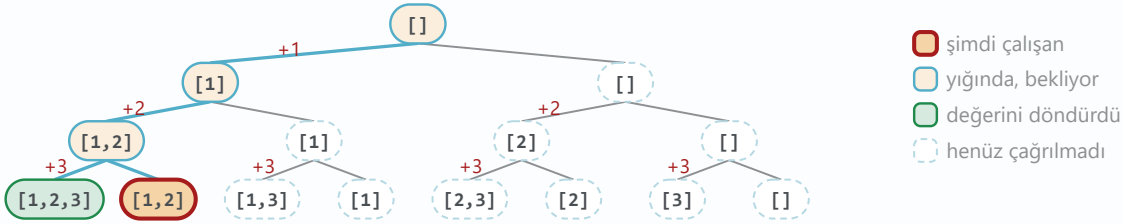
ÇIKTI

```
[1, 2, 3]
```

ÇAĞRI YIĞINI

- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[1, 2]` yazılır.

Adım 7 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

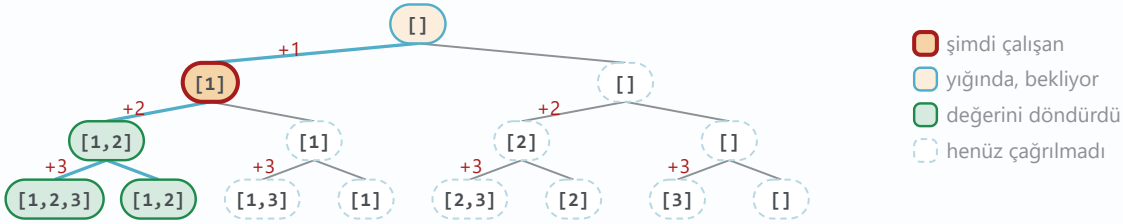
ÇIKTI

```
[1, 2, 3] [1, 2]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 2 **geri alınır** (s = [1]); aynı i ile sağ dala, 2'siz alt kümelere inilir.


```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                        // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);             // geri al
▶    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

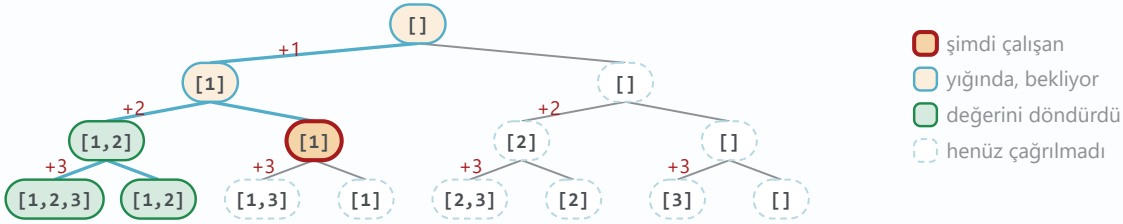
ÇIKTI

```
[1, 2, 3] [1, 2]
```

ÇAĞRI YIĞINI

- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 2: a[2] = 3 **alınır** (s = [1, 3]), sol dala inilir.

Adım 9 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                        // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);             // geri al
8    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

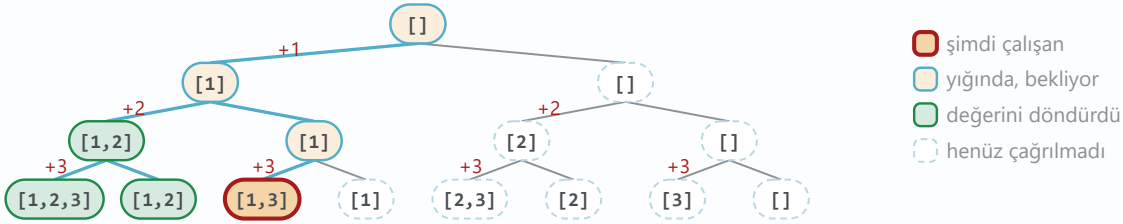
ÇIKTI

```
[1, 2, 3] [1, 2]
```

ÇAĞRI YIĞINI

- **`altKumeler(2)`**
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[1, 3]` yazılır.

Adım 10 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

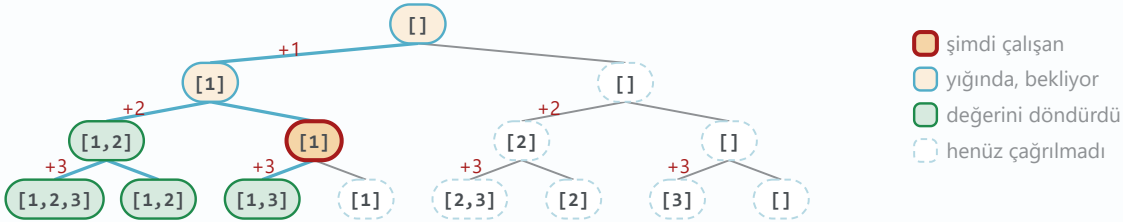
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 3 **geri alınır** (s = [1]); aynı i ile sağ dala, 3'siz alt kümelere inilir.

Adım 11 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                        // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);             // geri al
▶    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

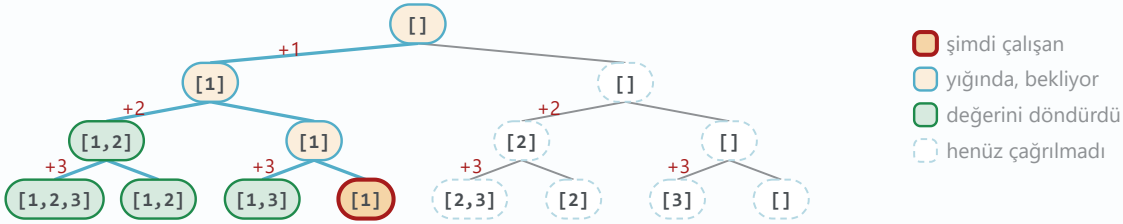
ÇIKTI

```
[1, 2, 3] [1, 2] [1, 3]
```

ÇAĞRI YIĞINI

- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[1]` yazılır.

Adım 12 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

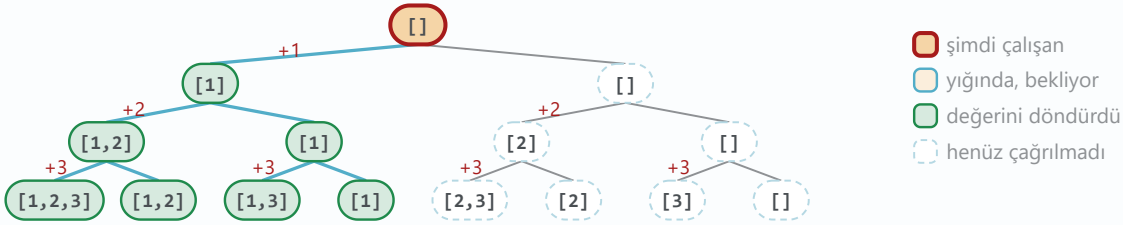
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 1 **geri alınır** (s = []); aynı i ile sağ dala, 1'siz alt kümelere inilir.

Adım 13 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                        // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);             // geri al
▶    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

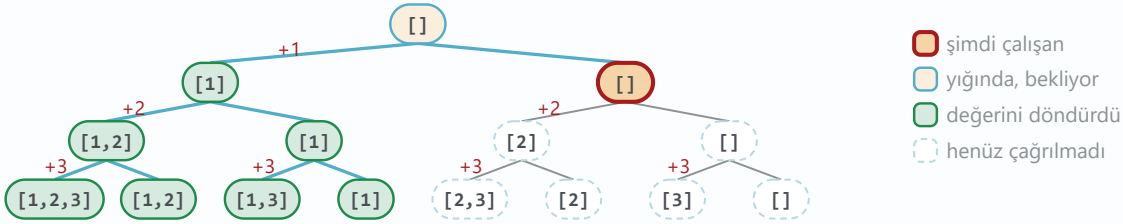
ÇIKTI

```
[1, 2, 3] [1, 2] [1, 3] [1]
```

ÇAĞRI YIĞINI

- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 1: a[1] = 2 **alınır** (s = [2]), sol dala inilir.


```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                        // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);             // geri al
8    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

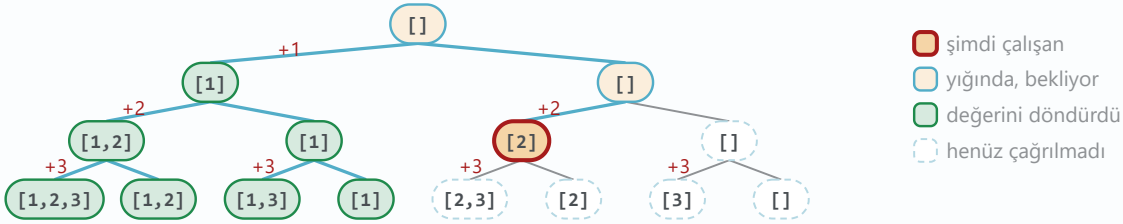
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
```

ÇAĞRI YIĞINI

- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 2: a[2] = 3 **alınır** (s = [2, 3]), sol dala inilir.

Adım 15 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {              // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                      // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

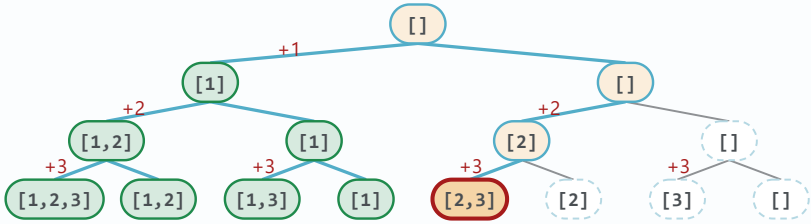
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
```

ÇAĞRI YIĞINI

- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[2, 3]` yazılır.

Adım 16 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

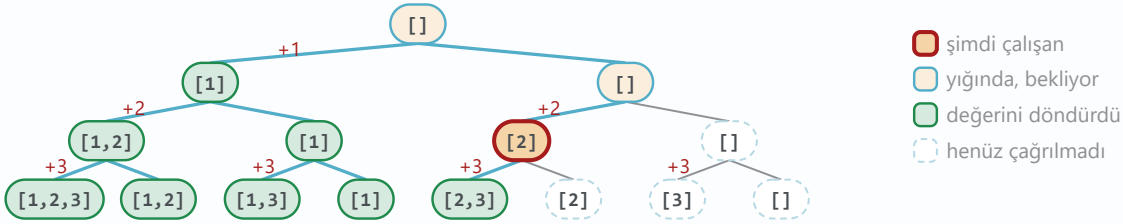
ÇIKTI

```
[1, 2, 3] [1, 2] [1, 3] [1] [2, 3]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 3 **geri alınır** (s = [2]); aynı i ile sağ dala, 3'siz alt kümelere inilir.


```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                        // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);             // geri al
▶    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

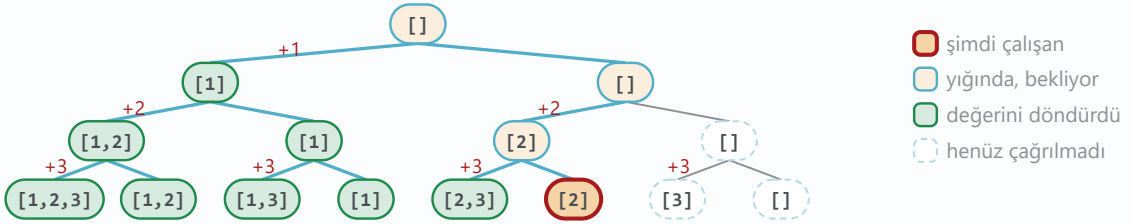
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3]
```

ÇAĞRI YIĞINI

- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[2]` yazılır.

Adım 18 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

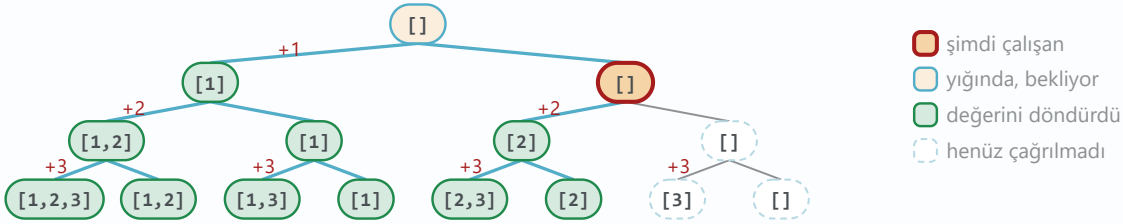
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 2 **geri alınır** (s = []); aynı i ile sağ dala, 2'siz alt kümelere inilir.

Adım 19 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {              // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);           // geri al
▶    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

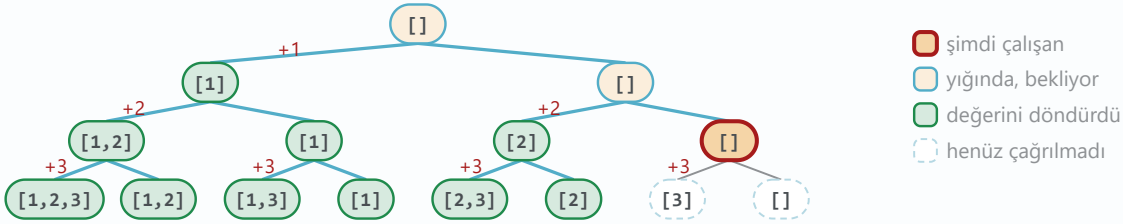
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2]
```

ÇAĞRI YIĞINI

- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



i = 2: a[2] = 3 **alınır** (s = [3]), sol dala inilir.

Adım 20 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
▶    s.add(a[i]);                        // a[i]'yi al
▶    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);             // geri al
8    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

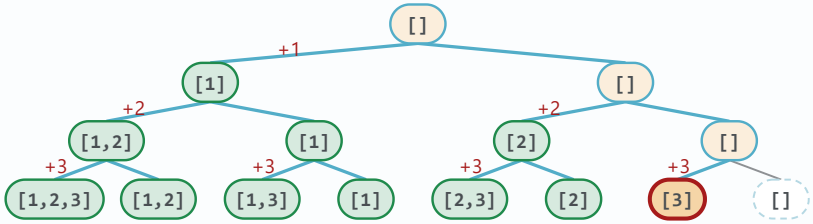
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2]
```

ÇAĞRI YIĞINI

- **`altKumeler(2)`**
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[3]` yazılır.

Adım 21 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

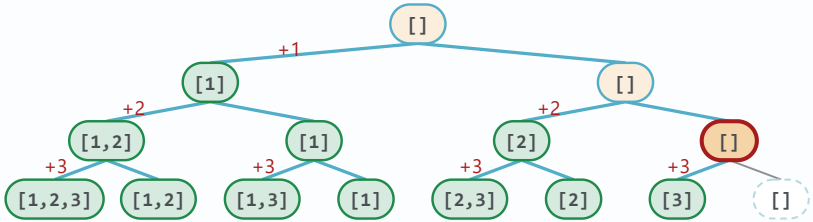
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2] [3]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



Sol daldan dönüldü. 3 **geri alınır** (s = []); aynı i ile sağ dala, 3'siz alt kümelere inilir.

Adım 22 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {                // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                        // a[i]'yi al
6    altKumeler(a, i + 1, s);
▶    s.remove(s.size() - 1);             // geri al
▶    altKumeler(a, i + 1, s);            // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

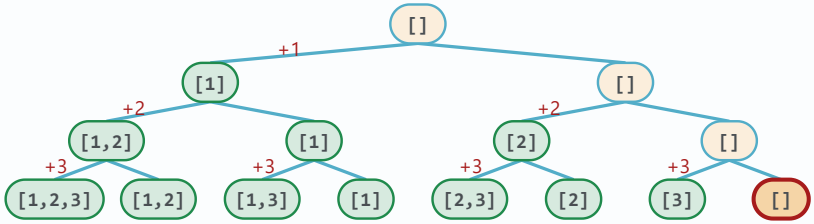
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2] [3]
```

ÇAĞRI YIĞINI

- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

i = 3 = a.length: **taban durum**. Tüm kararlar verildi → `[]` yazılır.

Adım 23 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
▶    if (i == a.length) {              // taban durum
▶      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
11 altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

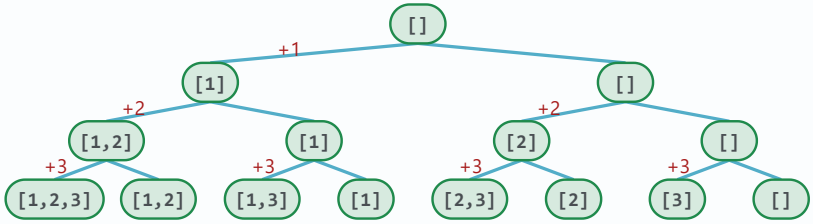
ÇIKTI

```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2] [3]
[]
```

ÇAĞRI YIĞINI

- `altKumeler(3)`
- `altKumeler(2)`
- `altKumeler(1)`
- `altKumeler(0)`
- `main()`

Kod İzleme: `altKumeler` ile {1, 2, 3}



şimdi çalışan
yığında, bekliyor
değerini döndürdü
henüz çağrılmadı

Bitti: **8 alt küme**, 15 çağrı. Yığında aynı anda en çok main + 4 çerçeve vardı.

Adım 24 / 24

```
1  void altKumeler(int[] a, int i, List<Integer> s) {
2    if (i == a.length) {              // taban durum
3      System.out.print(s + " "); return;
4    }
5    s.add(a[i]);                      // a[i]'yi al
6    altKumeler(a, i + 1, s);
7    s.remove(s.size() - 1);           // geri al
8    altKumeler(a, i + 1, s);          // a[i]'siz dene
9  }
10 // main içinde:
▶  altKumeler(new int[]{1, 2, 3}, 0, new ArrayList<>());
```

ÇIKTI

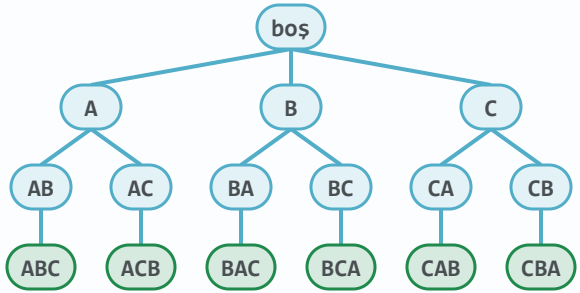
```
[1, 2, 3] [1,
2] [1, 3] [1]
[2, 3] [2] [3]
[]
```

ÇAĞRI YIĞINI

`main()`

Permütasyonları Listeleme

```
void perm(char[] h, boolean[] kul, StringBuilder s) {
  if (s.length() == h.length) {  // taban durum
    System.out.print(s + " "); return;
  }
  for (int k = 0; k < h.length; k++) {
    if (kul[k]) continue;         // harf kullanıldı
    kul[k] = true; s.append(h[k]);   // seç
    perm(h, kul, s);
    kul[k] = false;               // geri al
    s.deleteCharAt(s.length() - 1);
  }
}
```



- Çıktı: `ABC ACB BAC BCA CAB CBA`
- Her düzeyde kalan harflerden biri seçilir: 3 · 2 · 1 = 3! = 6 yaprak.
- Çağrı sayısı 1 + 3 + 6 + 6 = 16.

Bir çağrı, kalan harf sayısı kadar özyinelemeli çağrı yapar: **çoklu özyineleme**. `kul[]` ve `s` tüm çağrılarca paylaşılır; bu yüzden her denemeden sonra seçim geri alınır.

Yineleme Bağıntısı (Recurrence)

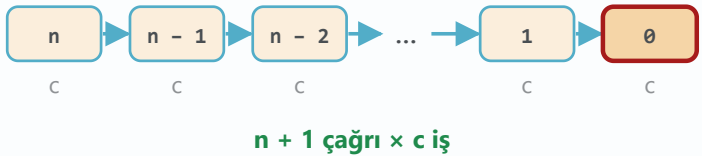
- **T(n)**: n boyutlu girdide metodun yaptığı toplam iş. Özyinelemeli metodun maliyeti, aynı metodun daha küçük girdilerdeki maliyetiyle yazılır: **yineleme bağıntısı**.
- Yazmak için üç soru: taban durumun maliyeti nedir (genellikle sabit c)? Kaç özyinelemeli çağrı yapılıyor ve girdileri ne kadar küçük? Çağrılar dışındaki iş ne kadar?

Metot	Çağrılar	Yineleme bağıntısı	Çözüm
faktoriyel, toplam, geriSay	bir çağrı, n − 1	$T(n) = T(n-1) + c$	$O(n)$
ters	bir çağrı, n − 2	$T(n) = T(n-2) + c$	$O(n)$
ikiliAra, hizliUs	bir çağrı, n / 2	$T(n) = T(n/2) + c$	$O(\log n)$
hanoi	iki çağrı, n − 1	$T(n) = 2T(n-1) + c$	$O(2^n)$
fib	iki çağrı, n − 1 ve n − 2	$T(n) = T(n-1) + T(n-2) + c$	üstel

Bağıntı, **çağrı ağacı** çizilerek çözülür: her düğüm sabit iş yapıyorsa toplam iş ≈ c · (çağrı sayısı).

T(n) = T(n − 1) + c

```
T(n) = T(n − 1) + c
     = T(n − 2) + 2c
     = T(n − 3) + 3c
     …
     = T(0) + n · c
     = c + n · c   →  O(n)
```



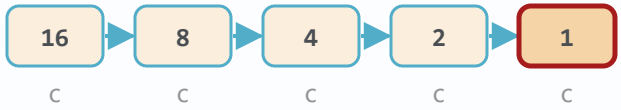
n + 1 çağrı × c iş

- Bağıntı **açılarak** çözülür: her açılışta bir c daha eklenir, n azalır. n adımda taban duruma inilir.
- Çağrı ağacı dallanmayan bir zincirdir: n + 1 çağrı. Zincirin tamamı aynı anda yığındadır → bellek de **O(n)**.
- Örnekler: faktoriyel, toplam, geriSay, yavaş üs alma, bağlı listenin uzunluğu. `ters` için T(n) = T(n − 2) + c: yarı sayıda çağrı, yine O(n).

YİNELEME BAĞINTILARI

$T(n) = T(n/2) + c$

$$
\begin{aligned}
T(n) &= T(n/2) + c \\
&= T(n/4) + 2c \\
&= T(n/8) + 3c \\
&\dots \\
&= T(n/2^k) + k \cdot c
\end{aligned}
$$



$\log_2 16 + 1 = 5$ çağrı

$n/2^k = 1$ olduğunda durulur: $k = \log_2 n$.
$T(n) = T(1) + c \cdot \log_2 n \rightarrow$ **O(log n)**

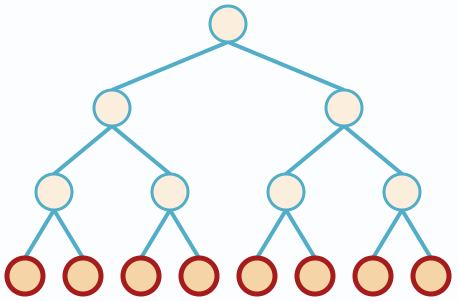
n	16	1.024	1.000.000
yarılama zinciri (çağrı)	5	11	20

Örnekler: ikili arama, hızlı üs alma. Girdi bin kat büyüyünce çağrı sayısı yalnızca yaklaşık 10 artar; zincir derinliği de log n olduğundan bellek O(log n).

YİNELEME BAĞINTILARI

$T(n) = 2T(n - 1) + c$

- Her çağrı n − 1 boyutlu **iki** çağrı yapar: çağrı ağacının her düzeyinde çağrı sayısı ikiye katlanır.
- Düzey k'de 2^k çağrı; düzeyler 0 … n − 1: $1 + 2 + 4 + \ldots + 2^{n-1} = \mathbf{2^n - 1}$ çağrı → $\mathbf{O(2^n)}$.
- Hanoi: tam olarak $2^n - 1$ hamle. n = 20 için 1.048.575; n = 64 için yaklaşık $1{,}8 \cdot 10^{19}$.
- fib: $T(n-2) \le T(n-1)$ olduğundan $T(n) \le 2T(n-1) + c$ → üst sınır $O(2^n)$; gerçek büyüme yaklaşık $1{,}6^n$.

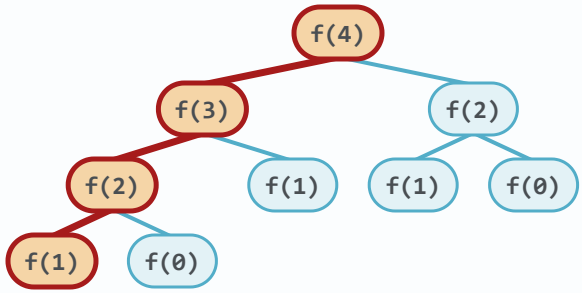


toplam $1 + 2 + 4 + 8 = 15 = 2^4 - 1$

Ağaç üstel büyüse de **derinliği n**'dir: aynı anda yığında en çok n çerçeve bulunur.

Alan Maliyeti: En Derin Çağrı Zinciri

- Bir çağrı, kardeşi başlamadan **biter**. Bu yüzden yığında aynı anda yalnızca kökten bir düğüme giden **tek bir yol** bulunur.
- Bellek = en derin çağrı zinciri × çerçeve boyutu. Toplam çağrı sayısı değil, **derinlik** belirler.
- `fib(4)` : 9 çağrı, ama aynı anda en çok 4 çerçeve (+ main).



Çağrı	Toplam çağrı	En derin zincir	Zaman	Alan
faktoriyel(4)	4	4	O(n)	O(n)
fib(40)	331.160.281	40	üstel	O(n)
hanoi(3, …)	7	3	$O(2^n)$	O(n)
altKumeler, n = 3	15	4	$O(n \cdot 2^n)$	O(n)

Kuyruk Özyinelemesi (Tail Recursion)

- Özyinelemeli çağrı metodun **son işiyse**, yani çağrı döndükten sonra yapılacak bir iş kalmıyorsa, metot **kuyruk özyinelemelidir**.
- Ara sonuç bir **biriktirici** parametreyle (acc) aşağı taşınır; çarpma çağrıdan **önce** yapılır. İlk çağrı: `faktKuyruk(n, 1)`.

Kuyruk özyinelemesi değil

```
int faktoriyel(int n) {
  if (n <= 1) return 1;
  return n * faktoriyel(n - 1);
  // çarpma, çağrı dönünce yapılır
}
```

Kuyruk özyinelemeli

```
int faktKuyruk(int n, int acc) {
  if (n <= 1) return acc;
  return faktKuyruk(n - 1, n * acc);
  // son iş: çağrı (çarpma önce)
}
```

Kuyruk özyinelemeli: geriSay, ters, ikiliAra, faktKuyruk. Değil: faktoriyel, toplam, fib, ileriSay (çağrıdan sonra iş var).

Kod İzleme: `faktKuyruk(4, 1)`

`faktKuyruk(4, 1)`
çağrı: (3, 4)

`faktKuyruk(3, 4)`
çağrı: (2, 12)

`faktKuyruk(2, 12)`
çağrı: (1, 24)

`faktKuyruk(1, 24)`
return acc

→ çağrı · ← dönüş

main, `faktKuyruk(4, 1)`'i çağırır; acc başlangıçta 1.

Adım 1 / 9

```
1 int faktKuyruk(int n, int acc) {
2   if (n <= 1) return acc;    // taban durum
3   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
▶ System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

—

ÇIKTI

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `faktKuyruk(4, 1)`

faktKuyruk(4, 1)
çağrı: (3, 4)

faktKuyruk(3, 4)
çağrı: (2, 12)

faktKuyruk(2, 12)
çağrı: (1, 24)

faktKuyruk(1, 24)
return acc

→ çağrı · ← dönüş

n = 4 > 1 → çarpma **önce**: 4 · 1 = 4. Sonra `faktKuyruk(3, 4)` çağrılır.

Adım 2 / 9

```
1 int faktKuyruk(int n, int acc) {
▶   if (n <= 1) return acc;    // taban durum
▶   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

`n = 4` `acc = 1`

ÇIKTI

ÇAĞRI YIĞINI

`faktKuyruk(4,1)`

`main()`

Kod İzleme: `faktKuyruk(4, 1)`



→ çağrı · ← dönüş

n = 3 > 1 → çarpma **önce**: 3 · 4 = 12. Sonra `faktKuyruk(2, 12)` çağrılır.

Adım 3 / 9

```
1 int faktKuyruk(int n, int acc) {
▶   if (n <= 1) return acc;    // taban durum
▶   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

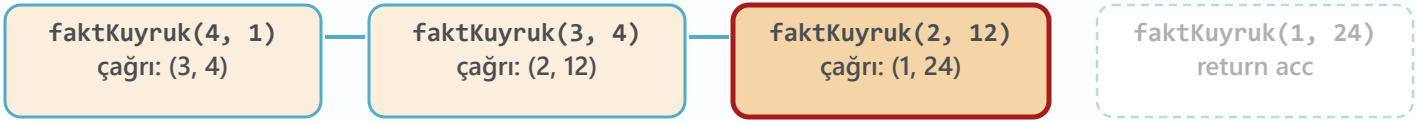
`n = 3` `acc = 4`

ÇIKTI

ÇAĞRI YIĞINI

- `faktKuyruk(3,4)`
- `faktKuyruk(4,1)`
- `main()`

Kod İzleme: `faktKuyruk(4, 1)`



→ çağrı · ← dönüş

n = 2 > 1 → çarpma **önce**: 2 · 12 = 24. Sonra `faktKuyruk(1, 24)` çağrılır.

Adım 4 / 9

```
1 int faktKuyruk(int n, int acc) {
▶   if (n <= 1) return acc;    // taban durum
▶   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

`n = 2`

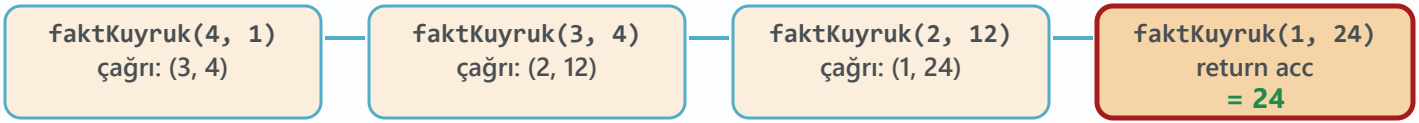
`acc = 12`

ÇIKTI

ÇAĞRI YIĞINI

- `faktKuyruk(2,12)`
- `faktKuyruk(3,4)`
- `faktKuyruk(4,1)`
- `main()`

Kod İzleme: `faktKuyruk(4, 1)`



→ çağrı · ← dönüş

n = 1: **taban durum** → acc = **24** döner. Sonuç burada hazır.

Adım 5 / 9

```
1 int faktKuyruk(int n, int acc) {
▶   if (n <= 1) return acc;    // taban durum
3   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

`n = 1`

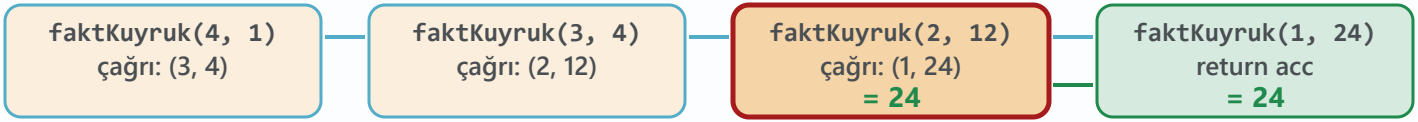
`acc = 24`

ÇIKTI

ÇAĞRI YIĞINI

- `faktKuyruk(1,24)`
- `faktKuyruk(2,12)`
- `faktKuyruk(3,4)`
- `faktKuyruk(4,1)`
- `main()`

Kod İzleme: `faktKuyruk(4, 1)`



faktKuyruk(2, 12), 24'ü **hiç değiştirmeden** iletir: dönüşte yapılacak iş yok.

Adım 6 / 9

```
1 int faktKuyruk(int n, int acc) {
2   if (n <= 1) return acc;    // taban durum
▶   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

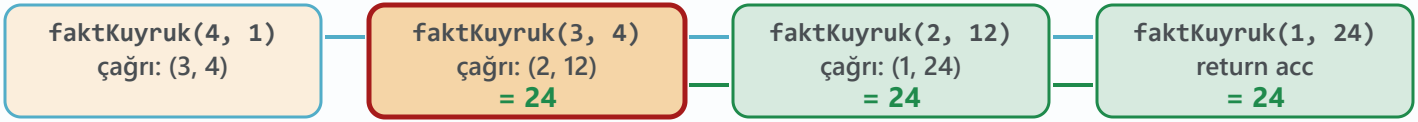
- `n = 2`
- `acc = 12`
- `dönen = 24`

ÇIKTI

ÇAĞRI YIĞINI

- `faktKuyruk(2,12)`
- `faktKuyruk(3,4)`
- `faktKuyruk(4,1)`
- `main()`

Kod İzleme: `faktKuyruk(4, 1)`



→ çağrı · ← dönüş

faktKuyruk(3, 4), 24'ü **hiç değiştirmeden** iletir: dönüşte yapılacak iş yok.

Adım 7 / 9

```
1 int faktKuyruk(int n, int acc) {
2   if (n <= 1) return acc;    // taban durum
▶   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

`n = 3` `acc = 4`

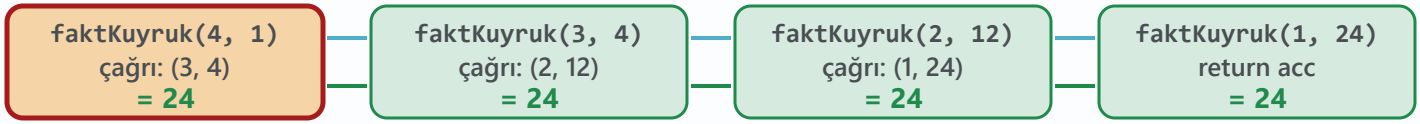
dönen = **24**

ÇIKTI

ÇAĞRI YIĞINI

- **`faktKuyruk(3,4)`**
- `faktKuyruk(4,1)`
- `main()`

Kod İzleme: `faktKuyruk(4, 1)`



→ çağrı · ← dönüş

faktKuyruk(4, 1), 24'ü **hiç değiştirmeden** iletir: dönüşte yapılacak iş yok.

Adım 8 / 9

```
1 int faktKuyruk(int n, int acc) {
2   if (n <= 1) return acc;    // taban durum
▶   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
6 System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

`n = 4` `acc = 1`

dönen = **24**

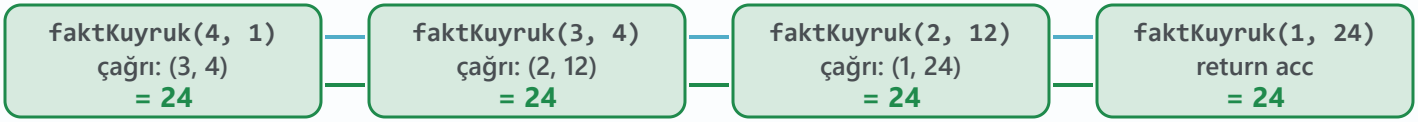
ÇAĞRI YIĞINI

`faktKuyruk(4,1)`

`main()`

ÇIKTI

Kod İzleme: `faktKuyruk(4, 1)`



→ çağrı · ← dönüş

Sonuç **24**. Java'da yine 4 çerçeve açıldı; dönüşte bu çerçevelere ihtiyaç yoktu.

Adım 9 / 9

```
1 int faktKuyruk(int n, int acc) {
2   if (n <= 1) return acc;    // taban durum
3   return faktKuyruk(n - 1, n * acc);  // son iş
4 }
5 // main içinde:
▶ System.out.println(faktKuyruk(4, 1));
```

DEĞİŞKENLER

—

ÇIKTI

```
24
```

ÇAĞRI YIĞINI

`main()`

Kuyruk Özyinelemesini Döngüye Çevirme

```
int faktDongu(int n) {
  int acc = 1;
  while (n > 1) {    // taban durum değilken
    acc = n * acc;   // çağrının yeni
    n = n - 1;       //   parametreleri
  }
  return acc;        // taban: return acc
}
```

- Kuyruk çağrısı, parametreleri yeni değerlerle güncelleyip metodun **başına dönmek** demektir: parametreler değişken, taban durum döngü koşulu olur.
- Çevirme mekaniktir; döngülü ikili arama da `ikiliAra` 'nın böyle çevrilmiş hâlidir.
- Döngü sabit bellekle çalışır: **O(1)**, çerçeve açılmaz.

Java kuyruk çağrısını eniyilemez: derleyici ve JVM `faktKuyruk` için de her çağrıda yeni çerçeve açar; çok derin çağrıda `StackOverflowError` yine oluşur. Scheme gibi bazı diller bu eniyilemeyi garanti eder; Scala ve Kotlin derleyicileri kendini çağıran kuyruk özyinelemesini döngüye çevirebilir.

KUYRUK ÖZYİNELEMESİ

Özyineleme mi, Döngü mü?

Ölçüt	Özyineleme	Döngü
Okunabilirlik	tanıma yakın, kısa	durum değişkenleri elle yönetilir
Bellek	O(derinlik) çağrı yığını	çoğunlukla O(1)
Hız	her çağrının ek maliyeti var	genellikle daha hızlı
Derinlik sınırı	`StackOverflowError` riski	yok

Özyineleme doğal seçim

Ağaç gibi iç içe yapılar, geri izleme (alt küme, permütasyon), problemi yarıya bölen yöntemler, Hanoi. Derinlik küçük (log n ya da n küçük).

Döngü daha iyi

Basit sayma ve toplama; derinliği n'ye ulaşan doğrusal özyineleme (uzun liste, büyük n); tekrarlanan alt problemli ikili özyineleme (fib).

Her özyinelemeli metot, çağrı yığınının işini **açık bir yığınla** yaparak döngüye çevrilebilir.

Bağlı Listede Özyineleme

- Bir bağlı liste ya **boştur** (`null`) ya da bir düğüm ve onu izleyen **daha kısa bir listedir**.
- Tanım özyinelemelidir: düğüm işlenir, kalan liste (`d.sonraki`) aynı metoda verilir.
- Taban durum: `d == null` .




```
class Dugum { int veri; Dugum sonraki; }

int uzunluk(Dugum d) {
  if (d == null) return 0;          // boş liste
  return 1 + uzunluk(d.sonraki);   // 1 + kalan
}
```

```
void yazdir(Dugum d) {
  if (d == null) return;
  System.out.print(d.veri + " ");
  yazdir(d.sonraki);
}
```

Kod İzleme: `uzunluk(bas)`



main, `uzunluk(bas)` 'ı çağırır; d ilk düğümü (7) gösterir.

Adım 1 / 9

```
1 int uzunluk(Dugum d) {
2   if (d == null) return 0;         // boş liste
3   return 1 + uzunluk(d.sonraki);   // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `uzunluk(bas)`



d ≠ null → 1 + `uzunluk(d.sonraki)` : kalan liste (3 ile başlayan) için çağrı; bu çerçeve bekler.

Adım 2 / 9

```
1 int uzunluk(Dugum d) {
▶   if (d == null) return 0;          // boş liste
▶   return 1 + uzunluk(d.sonraki);    // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

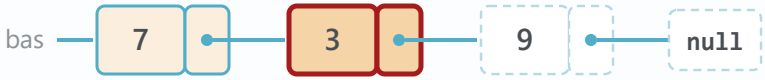
d = düğüm(7)

ÇIKTI

ÇAĞRI YIĞINI

- uzunluk(7)
- main()

Kod İzleme: `uzunluk(bas)`



d ≠ null → 1 + `uzunluk(d.sonraki)` : kalan liste (9 ile başlayan) için çağrı; bu çerçeve bekler.

Adım 3 / 9

```
1 int uzunluk(Dugum d) {
▶   if (d == null) return 0;          // boş liste
▶   return 1 + uzunluk(d.sonraki);    // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

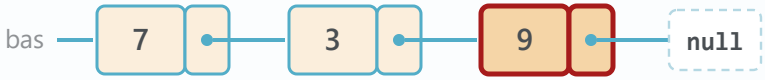
d = düğüm(3)

ÇIKTI

ÇAĞRI YIĞINI

- uzunluk(3)
- uzunluk(7)
- main()

Kod İzleme: `uzunluk(bas)`



d ≠ null → 1 + `uzunluk(d.sonraki)` : d.sonraki = null, boş liste için çağrı.

Adım 4 / 9

```
1 int uzunluk(Dugum d) {
▶   if (d == null) return 0;          // boş liste
▶   return 1 + uzunluk(d.sonraki);   // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

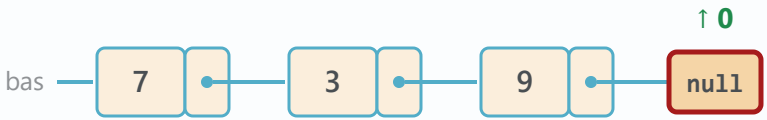
d = düğüm(9)

ÇIKTI

ÇAĞRI YIĞINI

- uzunluk(9)
- uzunluk(3)
- uzunluk(7)
- main()

Kod İzleme: `uzunluk(bas)`



d = null: **taban durum**, boş listenin uzunluğu 0 döner.

Adım 5 / 9

```
1 int uzunluk(Dugum d) {
▶   if (d == null) return 0;          // boş liste
3   return 1 + uzunluk(d.sonraki);   // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

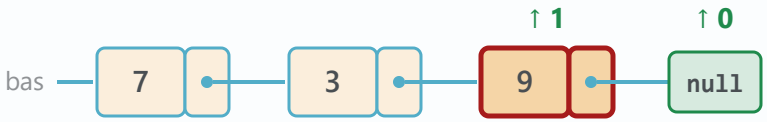
`d = null`

ÇIKTI

ÇAĞRI YIĞINI

- `uzunluk(null)`
- `uzunluk(9)`
- `uzunluk(3)`
- `uzunluk(7)`
- `main()`

Kod İzleme: `uzunluk(bas)`



Kalan listeden 0 geldi → 1 + 0 = **1** döner.

Adım 6 / 9

```
1 int uzunluk(Dugum d) {
2   if (d == null) return 0;          // boş liste
▶   return 1 + uzunluk(d.sonraki);    // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

`d = düğüm(9)`

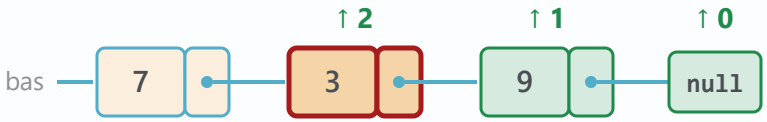
`dönen = 0`

ÇIKTI

ÇAĞRI YIĞINI

- `uzunluk(9)`
- `uzunluk(3)`
- `uzunluk(7)`
- `main()`

Kod İzleme: `uzunluk(bas)`



Kalan listeden 1 geldi → 1 + 1 = **2** döner.

Adım 7 / 9

```
1 int uzunluk(Dugum d) {
2   if (d == null) return 0;          // boş liste
▶   return 1 + uzunluk(d.sonraki);    // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

`d = düğüm(3)`

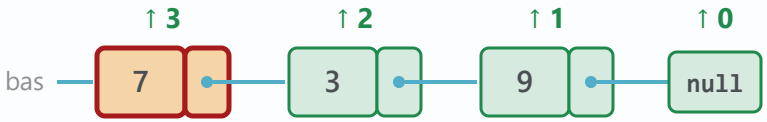
`dönen = 1`

ÇIKTI

ÇAĞRI YIĞINI

- `uzunluk(3)`
- `uzunluk(7)`
- `main()`

Kod İzleme: `uzunluk(bas)`



Kalan listeden 2 geldi → 1 + 2 = **3** döner.

Adım 8 / 9

```
1 int uzunluk(Dugum d) {
2   if (d == null) return 0;          // boş liste
▶   return 1 + uzunluk(d.sonraki);    // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

`d = düğüm(7)`

`dönen = 2`

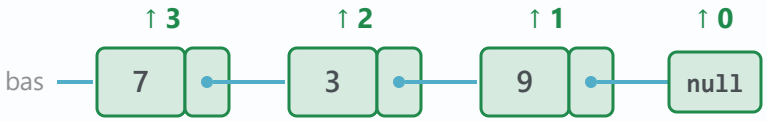
ÇAĞRI YIĞINI

`uzunluk(7)`

`main()`

ÇIKTI

Kod İzleme: `uzunluk(bas)`



Sonuç **3**. 4 çağrı (3 düğüm + null): zaman O(n), yığın O(n).

Adım 9 / 9

```
1 int uzunluk(Dugum d) {
2   if (d == null) return 0;          // boş liste
3   return 1 + uzunluk(d.sonraki);    // 1 + kalan
4 }
5 // main içinde (bas → 7 → 3 → 9 → null):
6 System.out.println(uzunluk(bas));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
3
```

Listeyi Tersten Yazdırma




```java
void yazdir(Dugum d) {
  if (d == null) return;
  System.out.print(d.veri + " ");
  yazdir(d.sonraki);
}
```

Çıktı: `7 3 9` (yazdırma çağrıdan önce)

```java
void tersYazdir(Dugum d) {
  if (d == null) return;
  tersYazdir(d.sonraki);  // önce kalan
  System.out.print(d.veri + " ");
}
```

Çıktı: `9 3 7` (yazdırma çağrıdan sonra)

- Tek yönlü listede geriye bağlantı yoktur; geri dönüşü **çağrı yığını** sağlar: düğümler, yığın çözülürken sondan başa yazılır.
- Bedeli O(n) yığın belleğidir. Çok uzun listelerde döngü ve açık bir yığın daha güvenlidir.

ÖZET

Sık Yapılan Hatalar

Taban durum eksik ya da ulaşılamaz

0 ya da negatif girdi tabanı atlarsa çağrılar bitmez: `StackOverflowError`. `n == 1` yerine `n <= 1` daha güvenlidir.

Problem küçülmüyor

`f(n)` içinden yine `f(n)` ya da `f(n + 1)` çağrılırsa taban duruma yaklaşılmaz.

Dönüş değerini kaybetmek

`ikiliAra`'da sol yarı çağrısının önündeki `return` unutulursa sonuç atılır, akış sağ yarı çağrısına düşer: yanlış cevap, derleyici uyarmaz.

Geri almayı unutmak

Geri izlemede `s.remove(...)` yapılmazsa bir dalın seçimleri sonraki dallara taşar; alt kümeler yanlış çıkar.

Tekrarlanan hesap

`fib` gibi ikili özyinelemede aynı alt problem üstel sayıda çözülür. Bellekleme ya da döngü O(n) verir.

Derin doğrusal özyineleme

Milyon düğümlü listede `uzunluk` yığını taşırır. Java kuyruk çağrısını eniyilemez → döngü.

Laboratuvar: Çağrı Yığını

faktoriyel(4)

Girdi (n, 0–10) 4

KOD · **adım 1 / 13**

```
1 int faktoriyel(int n) {
2   if (n <= 1)
3     return 1;                // taban du…
4   return n * faktoriyel(n - 1);
5 }
6
7 // main içinde:
8 System.out.println(faktoriyel(4));
```

main bekliyor: faktoriyel(4)

derinlik = 0 çağrı sayısı = 0

ÇIKTI –

main, faktoriyel(4) çağrısını yapar; çağrı yığınına yeni bir çerçeve eklenecek.

ÖZET

Özet

Örnek	Özyineleme türü	Zaman	Alan (yığın)
faktoriyel, toplam, ters	doğrusal	O(n)	O(n)
hizliUs, ikiliAra	doğrusal, yarılayan	O(log n)	O(log n)
fib	ikili	üstel ($\approx 1{,}6^n$)	O(n)
hanoi	ikili	$O(2^n)$	O(n)
altKumeler	geri izleme	$O(n \cdot 2^n)$	O(n)
perm	çoklu, geri izleme	$O(n \cdot n!)$	O(n)
faktKuyruk	kuyruk özyinelemesi	O(n)	O(n) (Java'da)
uzunluk, tersYazdir	doğrusal (bağlı liste)	O(n)	O(n)

Özyineleme = taban durum + taban duruma yaklaşan özyinelemeli adım. Zaman çağrı sayısından, alan en derin çağrı zincirinden gelir.

ÖZET

Etkileşimli Simülatörler

Hanoi Kuleleri Simülatörü

1–6 diskle bulmacayı elle çözme; en kısa çözümü adım adım ileri ve geri oynatma ya da otomatik çözdürme. Yapılan hamle sayısı en az hamle sayısıyla ($2^n - 1$) karşılaştırılır: n = 6 için 63.

Her kule bir yığın gibi davranır: yalnızca en üstteki disk alınabilir ve yalnızca daha büyük bir diskin üstüne konabilir.

ÖZET

Örnek Problemler

1. `int f(int n) { if (n <= 0) return 0; return n % 10 + f(n / 10); }` için `f(2026)` kaçtır, kaç çağrı yapılır?
2. `ileriSay(4)` ne yazar? Yığında (main dahil) en çok kaç çerçeve bulunur?
3. `hizliUs(3, 13)` kaç çağrı yapar, sonuç nedir?
4. `fib(6)` kaç çağrı yapar?
5. {1, 2, 3, 4} için `altKumeler` kaç alt küme yazar, kaç çağrı yapar?
6. T(n) = T(n/2) + c ve T(1) = c ise n = 64 için zincirde kaç çağrı vardır?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. `faktoriyel(5)` çalışırken çağrı yığınında (main dahil) en fazla kaç çerçeve olur?
2. 4 diskli Hanoi kaç hamle ister?
3. Dizinin en büyük öğesini döndüren özyinelemeli `enBuyuk(a, n)` için taban durum ve özyinelemeli adım nedir?
4. `palindrom(s, i, j)` uçları karşılaştırıp içe ilerler: "kayak" için kaç çağrı yapılır?
5. `int g(int n) { if (n <= 1) return 1; return g(n - 1) + g(n - 1); }` : g(5) değeri ve çağrı sayısı? Nasıl O(n) yapılır?
6. `faktKuyruk` 'un döngülü sürümünde her turda n ve acc nasıl değişir?

▶ **Cevaplar / ipuçları**

Kaynaklar

- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014.
- Sedgewick ve Wayne, *Computer Science: An Interdisciplinary Approach*, 2016.
- Roberts, *Thinking Recursively with Java*, 2006.
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022.

Son

Bölüm 4: Özyineleme

Sonraki sunum: Bölüm 5: Bağlı Listeler · Çalışma soruları · Simülatör: Hanoi Kuleleri Simülatörü · Terim sözlüğü