Bölüm 7: Yığınlar

Veri Yapıları

LIFO ve temel işlemler | Dizi ile yığın | Bağlı liste ile yığın | Parantez kontrolü | Sonraki büyük öğe | Hanoi kuleleri | Stock span | Infix → postfix / prefix | Postfix değerlendirme

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

LIFO modeli

Yığının son giren ilk çıkar davranışını açıklamak.

Gerçekleme

Yığını dizi ve bağlı liste kullanarak gerçekleştirmek.

Uygulamalar

Yığınları ifade çözümleme ve çağrı yönetiminde kullanmak.

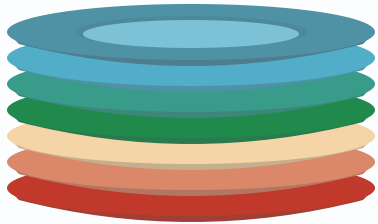
Maliyet analizi

Temel yığın işlemlerinin zaman ve alan maliyetlerini değerlendirmek.

YIĞIN ADT

Yığın (Stack)

- Ekleme ve çıkarma yalnızca bir uçtan, yığının **üstünden (top)** yapılır.
- Üst üste konmuş tabaklar ya da kutular gibi düşünülebilir.
- En alttaki öğeye ulaşmak için üstündeki öğeler tek tek çıkarılmalıdır.
- **LIFO** (Last-In, First-Out · son giren ilk çıkar) ya da eşdeğeri **FILO** (ilk giren son çıkar) mantığıyla çalışır.



Yeni tabak en üste konur, alınan tabak da en üstten alınır.

Geri al (undo)
Son yapılan işlem ilk geri alınır.

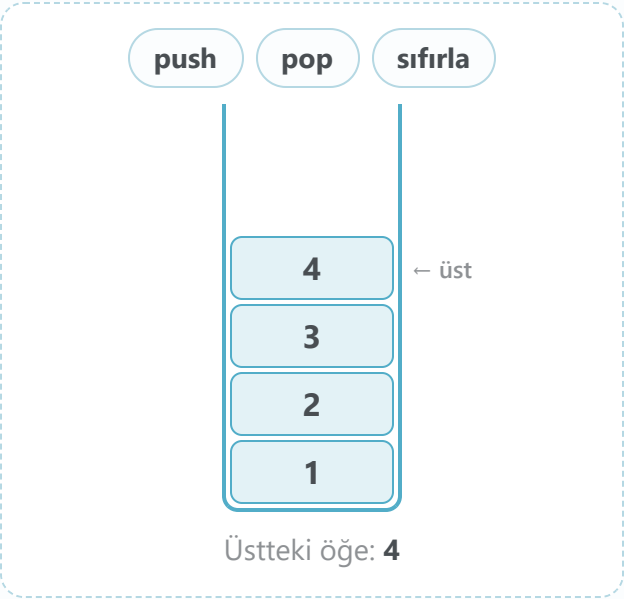
Tarayıcı “geri”
En son ziyaret edilen sayfaya dönülür.

Fonksiyon çağrıları
En son çağrılan fonksiyon ilk biter.

Temel İşlemler

İşlem	Açıklama
`push(x)`	Üste yeni öğe x ekler.
`pop()`	Üstteki öğeyi çıkarır ve döndürür.
`top()`	Üstteki öğeyi **çıkarmadan** döndürür (Java: `peek()`).
`isEmpty()`	Yığın boş mu?
`size()`	Öğe sayısını döndürür.

Kullanıcı yalnızca üstteki öğeyi görür; ortadaki öğelere doğrudan erişim **yoktur**.



push ve pop Adım Adım

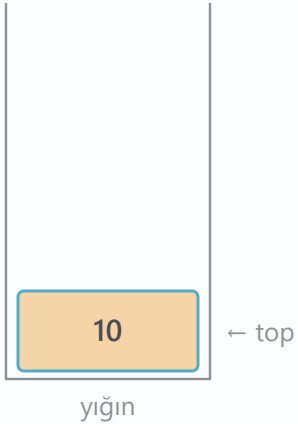
İşlem	Dönen değer	size



Başlangıçta yığın boş: **size = 0**, üstte öğe yok.

push ve pop Adım Adım

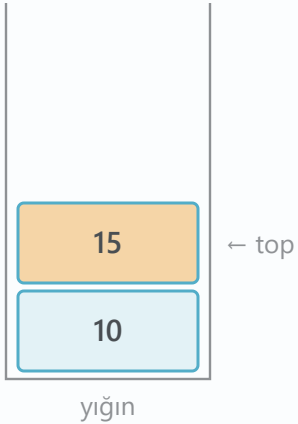
İşlem	Dönen değer	size
`push(10)`	–	1



push(10): 10 boş yığına eklenir; üst (top) artık 10.

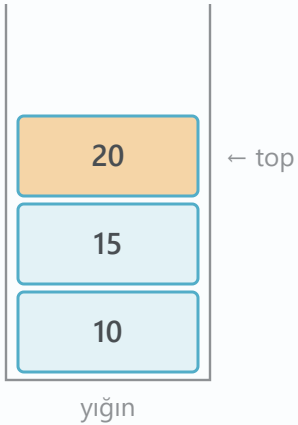
push ve pop Adım Adım

İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2



push(15): 15, 10'un üstüne konur.

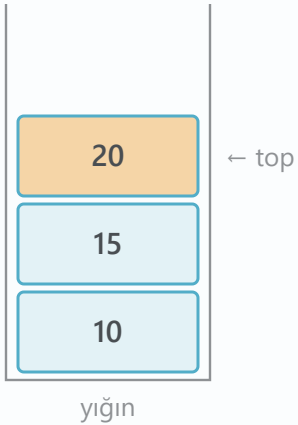
push ve pop Adım Adım



İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2
`push(20)`	–	3

push(20): 20 en üste konur; **size = 3**.

push ve pop Adım Adım



İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2
`push(20)`	–	3
`top()`	20	3

top(): üstteki 20 okunur ama **çıkarılmaz**.

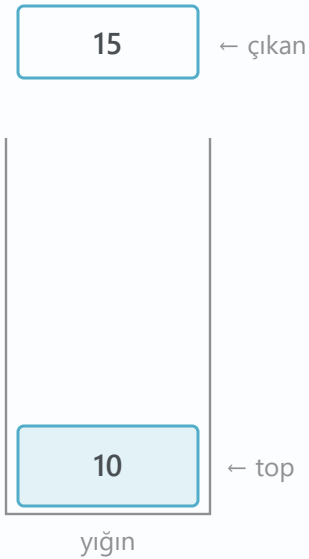
push ve pop Adım Adım



İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2
`push(20)`	–	3
`top()`	20	3
`pop()`	20	2

pop(): son giren 20 ilk çıkar (LIFO).

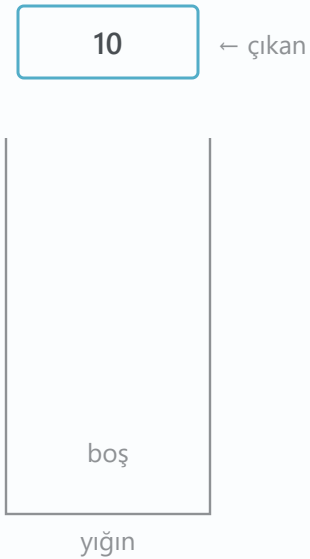
push ve pop Adım Adım



İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2
`push(20)`	–	3
`top()`	20	3
`pop()`	20	2
`pop()`	15	1

pop(): 15 çıkar; üst artık 10.

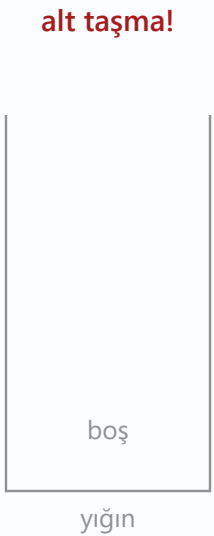
push ve pop Adım Adım



İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2
`push(20)`	–	3
`top()`	20	3
`pop()`	20	2
`pop()`	15	1
`pop()`	10	0

pop(): 10 çıkar; yığın yeniden boş.

push ve pop Adım Adım



İşlem	Dönen değer	size
`push(10)`	–	1
`push(15)`	–	2
`push(20)`	–	3
`top()`	20	3
`pop()`	20	2
`pop()`	15	1
`pop()`	10	0
`pop()`	**hata!**	0

Boş yığında **pop()** → **alt taşma (underflow)** hatası.

Yığın ADT'sinin Java Arayüzü

```
public interface Stack<E> {
  int size();
  boolean isEmpty();
  void push(E e);
  E top();     // boşsa null
  E pop();     // boşsa istisna
}
```

- **ADT = ne?** Arayüz yalnızca işlemleri ve anlamlarını söyler; verinin nasıl saklandığını söylemez.
- **Gerçekleme = nasıl?** `DiziYigin<E> implements Stack<E>` ya da bağlı liste tabanlı bir sınıf.
- Kod arayüze göre yazılırsa gerçekleme, çağıran kod değişmeden **değiştirilebilir**.

Boş yığında	`pop()`	`top()` / `peek()`
Kendi `Stack<E>`'imiz (yukarıdaki)	istisna fırlatır	`null` döner
`java.util.Stack`	`EmptyStackException`	`EmptyStackException`
`java.util.ArrayDeque`	`NoSuchElementException`	`null` döner

Aynı soyut işlem kütüphaneye göre farklı "boş" davranışı gösterir: önce `isEmpty()` ile kontrol edin. Kendi arayüzümüz `java.util.Stack` ile aynı adı taşır; ikisini aynı dosyada `import` etmeyin.

İşlemlerin Algoritmaları (Dizi ile)

Yığın bir dizide tutulur; `top` üstteki öğenin indisidir ve boş yığında **top = −1**.

push(x)

```
başla
  eğer yığın doluysa    // top = kap−1
    hata ver: taşma (overflow)
  değilse
    top değerini 1 artır
    yığın[top] ← x
son
```

pop()

```
başla
  eğer yığın boşsa      // top = −1
    hata ver: alt taşma (underflow)
  değilse
    x ← yığın[top]; top 1 azalt
    x değerini döndür
son
```

top()

```
başla
  eğer yığın boşsa null döndür
  değilse yığın[top] döndür
son
```

isEmpty() · size()

```
isEmpty: top = −1 ise true, değilse false
size:    top + 1 döndür
```

Zaman ve Alan Karmaşıklığı

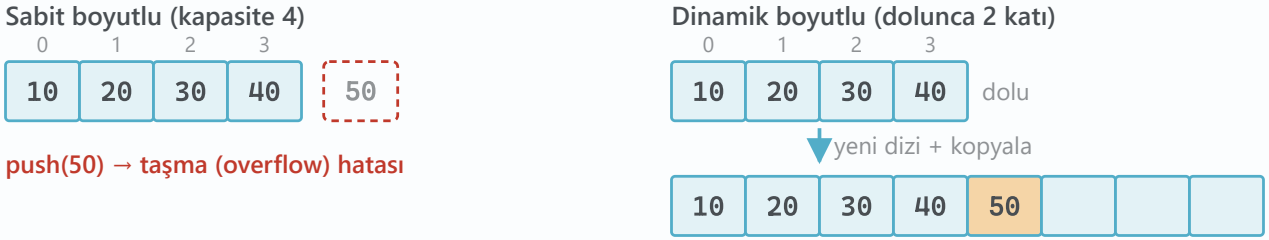
İşlem	Zaman	Neden?
`push`	O(1)	yalnızca üst değişir
`pop`	O(1)	yalnızca üst değişir
`top`	O(1)	üst doğrudan bilinir
`isEmpty`, `size`	O(1)	sayaç / indis tutulur
bir öğeyi arama	O(n)	öğeler tek tek çıkarılmalı

- Temel işlemlerin süresi yığındaki öğe sayısından **bağımsızdır**.
- n öğeli yığın **O(n)** bellek kullanır; işlemler ek bellek gerektirmez (O(1)).
- Dinamik dizide kapasite dolunca dizi büyütülür: push **amortize O(1)**.

Yığın, erişimi kısıtlayarak hız kazanır: “yalnızca üst” kuralı sayesinde hiçbir temel işlem öğeleri kaydırmaz ya da taramaz.

GERÇEKLEME

Yığın Türleri



Sabit Boyutlu Yığın (Fixed Size Stack)

- Kapasitesi baştan belirlenir; büyüyemez, küçülemez.
- Dolu yığına ekleme → **taşma** (overflow).
- Boş yığından çıkarma → **alt taşma** (underflow).
- Arka planda sabit boyutlu bir **dizi** kullanır.

Dinamik Boyutlu Yığın (Dynamic Size Stack)

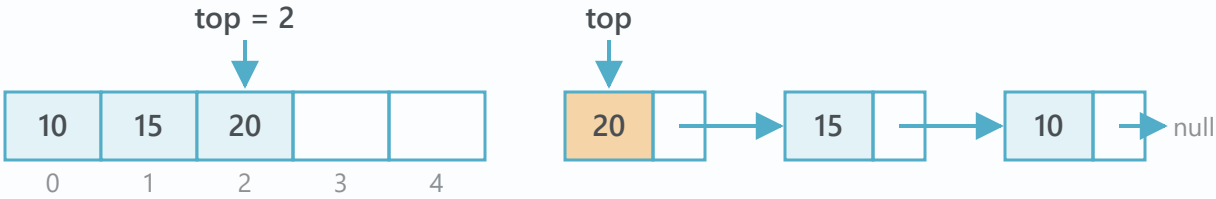
- Çalışma sırasında büyüyüp küçülebilir.
- Dolunca kapasitesini otomatik artırır (ör. iki katına çıkarır).
- Öğe sayısı kapasiteye göre çok azalınca küçülebilir.
- **Dinamik dizi** ya da **bağlı liste** ile gerçekleştirilir.

Boş yığından çıkarma (alt taşma) her iki türde de hatadır; taşma yalnızca sabit boyutlu yığında görülür.

GERÇEKLEME

Yığın Gerçekleme Yöntemleri

Yığın ADT'si bir **dizi** (array) ya da **bağlı liste** (linked list) ile gerçekleştirilebilir.



dizi: push → top++ , pop → top−−

bağlı liste: push/pop listenin başında

Dizi Tabanlı

`push` : top indisi artırılır, öğe `dizi[top]` 'a yazılır.
`pop` : `dizi[top]` alınır, sonra top azaltılır.

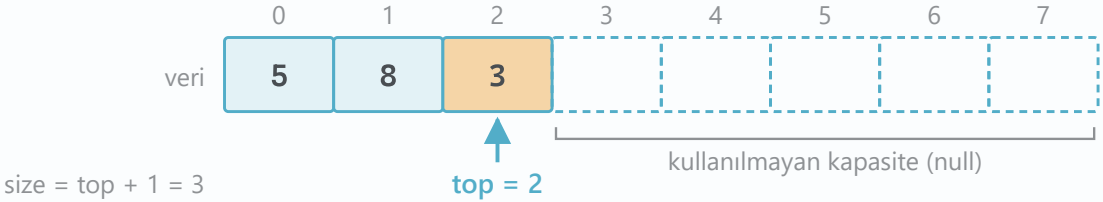
Bağlı Liste Tabanlı

`push` : yeni düğüm listenin **başına** eklenir.
`pop` : baştaki düğüm çıkarılır, değeri döndürülür.

GERÇEKLEME

Dizi Tabanlı Yığın

Bu yaklaşıma **sınırlı yığın** (bounded stack) da denir: kapasite oluşturulurken belirlenir ve değişmez.



Avantajları

- \+ Basit ve kolay uygulanır.
- \+ Düğüm başına referans tutulmaz → bellek tasarrufu.
- \+ Bellek kullanımı sıkı biçimde kontrol edilir; belleği doldurmaya yönelik saldırılara karşı daha güvenlidir.
- \+ Öğeler ardışık → önbellek (cache) dostu.

Dikkat edilecekler

- – Eklemeden önce yığının dolu olmadığı kontrol edilmelidir.
- – Kapasite az seçilirse taşma, çok seçilirse boşa bellek.
- – Nesne tutan dizide çıkarılan hücreye `null` atanmalıdır; yoksa nesne bellekte gereksiz yere kalır.

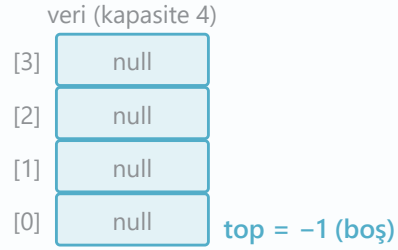
GERÇEKLEME

Dizi Tabanlı Yığın: Java

```java
public class DiziYigin<E> implements Stack<E> {
    private E[] veri;
    private int top = -1;                    // -1 → boş yığın
    public DiziYigin(int kapasite) { veri = (E[]) new Object[kapasite]; }
    public boolean isEmpty() { return top == -1; }
    public int size()        { return top + 1; }
    public E top()           { return isEmpty() ? null : veri[top]; }
    public void push(E x) {
        if (top == veri.length - 1) throw new IllegalStateException("Taşma");
        veri[++top] = x;
    }
    public E pop() {
        if (isEmpty()) throw new IllegalStateException("Alt taşma");
        E x = veri[top];
        veri[top--] = null;                  // referansı bırak
        return x;
    }
}
```

Jenerik (generic) dizi doğrudan oluşturulamadığı için `Object[]` dönüştürülür (derleyici "unchecked" uyarısı verir). Tüm işlemler O(1).

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)
y.push(15)
y.push(20)
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

`new DiziYigin<>(4)` : 4 hücrelik dizi, hepsi **null**; boş yığında **top = −1**.

Adım 1 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

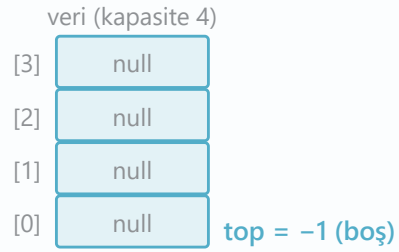
`top = −1`

`veri = [null, null, null, null]`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)
y.push(15)
y.push(20)
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

push(10): dolu mu? top = -1, veri.length − 1 = 3 → hayır, devam.

Adım 2 / 18

```
1  public void push(E x) {
▶    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = -1`

`veri = [null, null, null, null]`

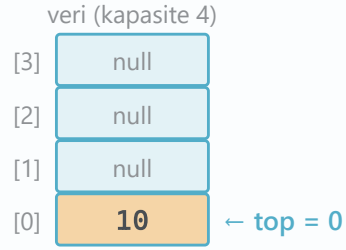
`x = 10`

ÇAĞRI YIĞINI

`y.push(10)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)    ✓
y.push(15)
y.push(20)
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

`veri[++top] = x` : önce top -1 → **0**, sonra veri[0] = 10. Tek atama → O(1).

Adım 3 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
▶    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 0`

`veri = [10, null, null, null]`

`x = 10`

ÇAĞRI YIĞINI

`y.push(10)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)     ✓
y.push(15)
y.push(20)
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

push(15): dolu mu? top = 0, veri.length − 1 = 3 → hayır, devam.

Adım 4 / 18

```
1  public void push(E x) {
▶    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

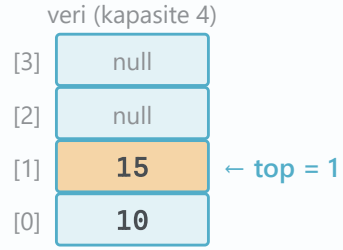
DEĞİŞKENLER

- `top = 0`
- `veri = [10, null, null, null]`
- `x = 15`

ÇAĞRI YIĞINI

- `y.push(15)`
- `main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

`veri[++top] = x` : önce top 0 → **1**, sonra veri[1] = 15. Tek atama → O(1).

Adım 5 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
▶    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 1`

`veri = [10, 15, null, null]`

`x = 15`

ÇAĞRI YIĞINI

`y.push(15)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

push(20): dolu mu? top = 1, veri.length − 1 = 3 → hayır, devam.

Adım 6 / 18

```
1  public void push(E x) {
▶    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 1`

`veri = [10, 15, null, null]`

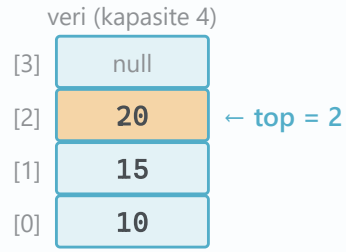
`x = 20`

ÇAĞRI YIĞINI

`y.push(20)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

`veri[++top] = x` : önce top 1 → **2**, sonra veri[2] = 20. Tek atama → O(1).

Adım 7 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
▶    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 2`

`veri = [10, 15, 20, null]`

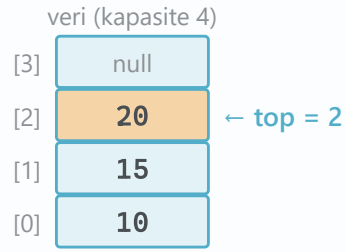
`x = 20`

ÇAĞRI YIĞINI

`y.push(20)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

top(): yığın boş değil → **veri[2] = 20** döner. Dizi ve top **değişmez**.

Adım 8 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
▶ public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 2`

`veri = [10, 15, 20, null]`

ÇAĞRI YIĞINI

`y.top()`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

pop(): isEmpty()? top = 2 ≠ −1 → boş değil, devam.

Adım 9 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
▶    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 2`

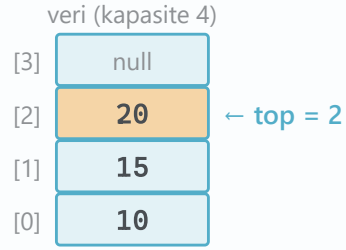
`veri = [10, 15, 20, null]`

ÇAĞRI YIĞINI

`y.pop()`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

`E x = veri[top]` → x = 20 (üstteki öğe saklanır).

Adım 10 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
▶   E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 2`

`veri = [10, 15, 20, null]`

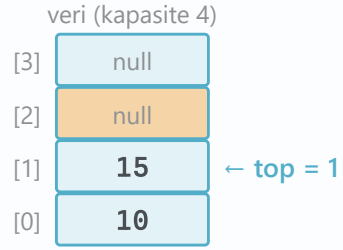
`x = 20`

ÇAĞRI YIĞINI

`y.pop()`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()
y.push(25)
y.push(30)
y.push(40)
```

`veri[top--] = null` : hücre boşaltılır (nesne çöp toplayıcıya bırakılır), top → **1**.

Adım 11 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
▶    veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 1`

`veri = [10, 15, null, null]`

`x = 20`

ÇAĞRI YIĞINI

`y.pop()`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()         → 20
y.push(25)
y.push(30)
y.push(40)
```

return 20: son giren ilk çıktı (LIFO).

Adım 12 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
▶    return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

- `top = 1`
- `veri = [10, 15, null, null]`
- `x = 20`

ÇAĞRI YIĞINI

- `y.pop()`
- `main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()         → 20
y.push(25)
y.push(30)
y.push(40)
```

push(25): dolu mu? top = 1, veri.length − 1 = 3 → hayır, devam.

Adım 13 / 18

```
1  public void push(E x) {
▶    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

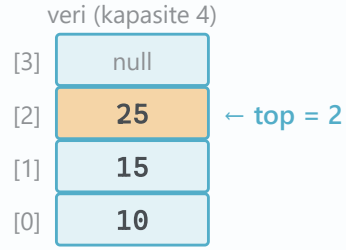
DEĞİŞKENLER

- `top = 1`
- `veri = [10, 15, null, null]`
- `x = 25`

ÇAĞRI YIĞINI

- `y.push(25)`
- `main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()         → 20
y.push(25)      ✓
y.push(30)
y.push(40)
```

`veri[++top] = x` : önce top 1 → **2**, sonra veri[2] = 25. Tek atama → O(1).

Adım 14 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
▶    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 2`

`veri = [10, 15, 25, null]`

`x = 25`

ÇAĞRI YIĞINI

`y.push(25)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()         → 20
y.push(25)      ✓
y.push(30)
y.push(40)
```

push(30): dolu mu? top = 2, veri.length − 1 = 3 → hayır, devam.

Adım 15 / 18

```
1  public void push(E x) {
▶    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 2`

`veri = [10, 15, 25, null]`

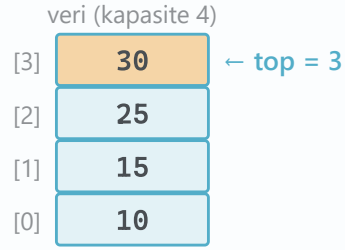
`x = 30`

ÇAĞRI YIĞINI

`y.push(30)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()         → 20
y.push(25)      ✓
y.push(30)      ✓
y.push(40)
```

`veri[++top] = x` : önce top 2 → **3**, sonra veri[3] = 30. Tek atama → O(1).

Adım 16 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
▶    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

`top = 3`

`veri = [10, 15, 25, 30]`

`x = 30`

ÇAĞRI YIĞINI

`y.push(30)`

`main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)      ✓
y.push(15)      ✓
y.push(20)      ✓
y.top()         → 20
y.pop()         → 20
y.push(25)      ✓
y.push(30)      ✓
y.push(40)
```

push(40): dolu mu? top = 3, veri.length − 1 = 3 → **evet, dolu!**

Adım 17 / 18

```
1  public void push(E x) {
▶    if (top == veri.length - 1)
3      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

- `top = 3`
- `veri = [10, 15, 25, 30]`
- `x = 40`

ÇAĞRI YIĞINI

- `y.push(40)`
- `main()`

Kod İzleme: Dizi Tabanlı Yığın



main() içindeki çağrılar

```
y.push(10)     ✓
y.push(15)     ✓
y.push(20)     ✓
y.top()        → 20
y.pop()        → 20
y.push(25)     ✓
y.push(30)     ✓
y.push(40)     ✗ Taşma
```

Dizi dolu: **IllegalStateException("Taşma")** fırlatılır. Sabit kapasiteli yığında **taşma** budur; dinamik dizi burada diziyi büyütürdü.

Adım 18 / 18

```
1  public void push(E x) {
2    if (top == veri.length - 1)
▶      throw new IllegalStateException("Taşma");
4    veri[++top] = x;
5  }
6  public E pop() {
7    if (isEmpty())
8      throw new IllegalStateException("Alt taşma");
9    E x = veri[top];
10   veri[top--] = null;          // referansı bırak
11   return x;
12 }
13 public E top() { return isEmpty() ? null : veri[top]; }
```

DEĞİŞKENLER

- `top = 3`
- `veri = [10, 15, 25, 30]`
- `x = 40`

ÇAĞRI YIĞINI

- `y.push(40)`
- `main()`

GERÇEKLEME

Dinamik Dizi: Dolunca İki Katına Büyütme

```
public void push(E x) {
  if (top == veri.length - 1)
    buyut(2 * veri.length);
  veri[++top] = x;
}
private void buyut(int yeniKap) {
  E[] yeni = (E[]) new Object[yeniKap];
  for (int i = 0; i <= top; i++)
    yeni[i] = veri[i];
  veri = yeni;
}
```

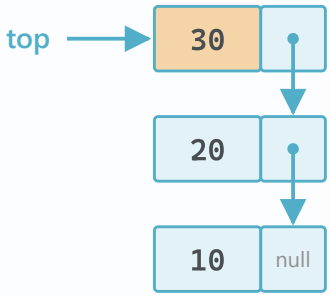
- Dizi dolunca iki katı kapasiteli yeni dizi ayrılır ve öğeler kopyalanır: büyütme yapan push **O(n)** sürer.
- Büyütmeler seyrektir: n push'ta kopyalar 2 + 4 + 8 + … < 2n, toplam maliyet < 3n → push **amortize O(1)**.
- Her dolumda sabit sayıda (+c) hücre eklenirse toplam $O(n^2)$, push başına O(n) olur; kapasite **oranla** büyütülmelidir.

Amortize maliyet tek bir işlemin değil, uzun bir işlem dizisinin ortalamasıdır.

GERÇEKLEME

Bağlı Liste Tabanlı Yığın

- Öğeler birbirine bağlı düğümlerden oluşan bir zincirde tutulur → esnek boyut.
- Boyut, çalışma zamanında ihtiyaca göre değişir; **taşma olmaz** (bellek bitene kadar).
- Her düğümde bir referans (sonraki) tutulduğu için ek bellek kullanır.
- Rastgele erişim yoktur, ama üstteki öğeye (listenin başı) erişim hızlı ve basittir.
- Ekleme ve çıkarma listenin başında yapıldığı için O(1)'dir.



push / pop yalnızca baştaki (top) düğümde

```
class Dugum {
    int veri;
    Dugum sonraki;
    Dugum(int veri) { this.veri = veri; }   // sonraki = null
}
```

Neden başa ekliyoruz? Tek yönlü listede sondan silmek için sondan bir önceki düğümü bulmak gerekir → O(n).

Kod İzleme: Bağlı Liste ile `push`

top = null

Başlangıç: **top = null**, uzunluk = 0 → boş yığın.

düğüm = [veri | sonraki]

Adım 1 / 10

```
Dugum top;          // yığının üstü
int uzunluk;

public void push(int veri) {
  Dugum gecici = new Dugum(veri);
  gecici.sonraki = top;
  top = gecici;
  uzunluk++;
}
```

DEĞİŞKENLER

`top = null` `uzunluk = 0`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: Bağlı Liste ile `push`



push(10): `new Dugum(10)` oluşturulur; **gecici** onu gösterir, sonraki alanı varsayılan olarak null.

düğüm = [veri | sonraki]

Adım 2 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
▶   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
7   top = gecici;
8   uzunluk++;
9 }
```

DEĞİŞKENLER

`veri = 10` `gecici = düğüm(10)` `top = null` `uzunluk = 0`

ÇAĞRI YIĞINI

`push(10)`

`main()`

Kod İzleme: Bağlı Liste ile `push`



`gecici.sonraki = top` : top null olduğu için sonraki yine **null** (yeni düğüm tek başına).

düğüm = [veri | sonraki]

Adım 3 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
▶   gecici.sonraki = top;
7   top = gecici;
8   uzunluk++;
9 }
```

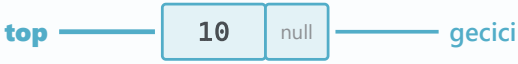
DEĞİŞKENLER

`veri = 10` `gecici = düğüm(10)` `gecici.sonraki = null` `top = null` `uzunluk = 0`

ÇAĞRI YIĞINI

- `push(10)`
- `main()`

Kod İzleme: Bağlı Liste ile `push`



`top = gecici` : yığının üstü artık yeni düğüm.

düğüm = [veri | sonraki]

Adım 4 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
▶   top = gecici;
8   uzunluk++;
9 }
```

DEĞİŞKENLER

`veri = 10` `gecici = düğüm(10)` `gecici.sonraki = null` `top = düğüm(10)` `uzunluk = 0`

ÇAĞRI YIĞINI

`push(10)`

`main()`

Kod İzleme: Bağlı Liste ile `push`



`uzunluk++` → 1. Metot biter; yerel **gecici** kaybolur, düğüm top üzerinden erişilir.

düğüm = [veri | sonraki]

Adım 5 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
7   top = gecici;
▶   uzunluk++;
9 }
```

DEĞİŞKENLER

`veri = 10` `gecici = düğüm(10)`

`gecici.sonraki = null` `top = düğüm(10)`

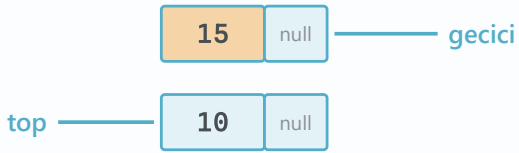
`uzunluk = 1`

ÇAĞRI YIĞINI

`push(10)`

`main()`

Kod İzleme: Bağlı Liste ile `push`



düğüm = [veri | sonraki]

push(15): yeni düğüm 15 oluşturulur. Henüz listeye **bağlı değil**; top hâlâ 10'u gösterir.

Adım 6 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
▶   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
7   top = gecici;
8   uzunluk++;
9 }
```

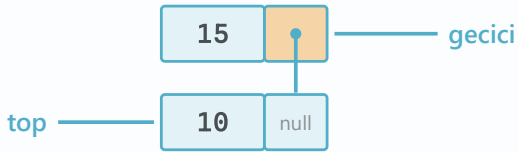
DEĞİŞKENLER

`veri = 15` `gecici = düğüm(15)` `top = düğüm(10)` `uzunluk = 1`

ÇAĞRI YIĞINI

- `push(15)`
- `main()`

Kod İzleme: Bağlı Liste ile `push`



düğüm = [veri | sonraki]

`gecici.sonraki = top` : 15, eski üst olan 10'a bağlanır. Önce bağlarız ki eski liste kaybolmasın.

Adım 7 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
▶  gecici.sonraki = top;
7   top = gecici;
8   uzunluk++;
9 }
```

DEĞİŞKENLER

`veri = 15` `gecici = düğüm(15)`

`gecici.sonraki = düğüm(10)` `top = düğüm(10)`

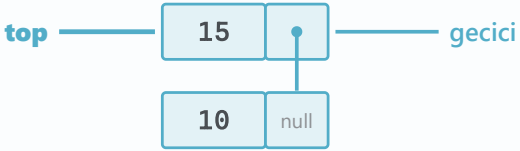
`uzunluk = 1`

ÇAĞRI YIĞINI

`push(15)`

`main()`

Kod İzleme: Bağlı Liste ile `push`



düğüm = [veri | sonraki]

`top = gecici` : yeni üst 15. Listenin **başına** ekleme, kaydırma yok → O(1).

Adım 8 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
▶   top = gecici;
8   uzunluk++;
9 }
```

DEĞİŞKENLER

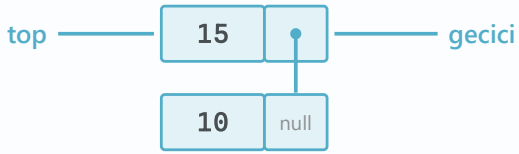
`veri = 15` `gecici = düğüm(15)` `gecici.sonraki = düğüm(10)` `top = düğüm(15)` `uzunluk = 1`

ÇAĞRI YIĞINI

`push(15)`

`main()`

Kod İzleme: Bağlı Liste ile `push`



`uzunluk++` → 2.

düğüm = [veri | sonraki]

Adım 9 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
7   top = gecici;
▶   uzunluk++;
9 }
```

DEĞİŞKENLER

`veri = 15` `gecici = düğüm(15)`

`gecici.sonraki = düğüm(10)` `top = düğüm(15)`

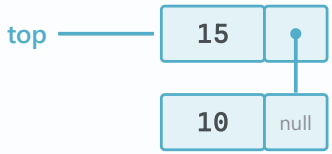
`uzunluk = 2`

ÇAĞRI YIĞINI

`push(15)`

`main()`

Kod İzleme: Bağlı Liste ile `push`



push biter. Yığın (üstten alta): **15** → **10**. Son eklenen en üstte.

düğüm = [veri | sonraki]

Adım 10 / 10

```
1 Dugum top;          // yığının üstü
2 int uzunluk;
3
4 public void push(int veri) {
5   Dugum gecici = new Dugum(veri);
6   gecici.sonraki = top;
7   top = gecici;
8   uzunluk++;
▶ }
```

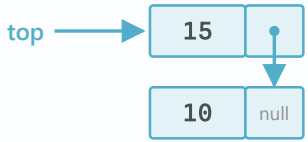
DEĞİŞKENLER

`top = düğüm(15)` `uzunluk = 2`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: Bağlı Liste ile `pop`



Yığında 15 ve 10 var. Döngü pop()'u **3 kez** çağıracak.

düğüm = [veri | sonraki]

Adım 1 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
▶ for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

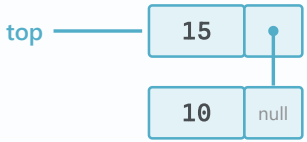
`k = 0` `top = düğüm(15)` `uzunluk = 2`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: Bağlı Liste ile `pop`



pop(): önce boşluk kontrolü; top null değil → devam.

düğüm = [veri | sonraki]

Adım 2 / 15

```
1  public int pop() {
▶    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 0` `top = düğüm(15)` `uzunluk = 2`

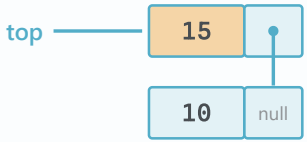
ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

Kod İzleme: Bağlı Liste ile `pop`



düğüm = [veri | sonraki]

`sonuc = top.veri` → **15** saklanır (düğüm kopmadan önce).

Adım 3 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
▶    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

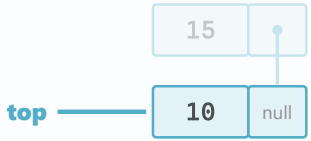
`k = 0` `top = düğüm(15)` `sonuc = 15` `uzunluk = 2`

ÇAĞRI YIĞINI

- `pop()`
- `main()`

ÇIKTI

Kod İzleme: Bağlı Liste ile `pop`



düğüm = [veri | sonraki]

`top = top.sonraki` : top bir aşağı iner → **düğüm(10)**. Eski üst düğüme artık referans yok.

Adım 4 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
▶    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 0` `top = düğüm(10)` `sonuc = 15` `uzunluk = 2`

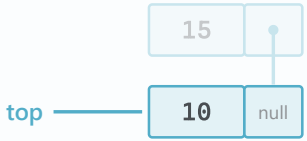
ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

Kod İzleme: Bağlı Liste ile `pop`



`uzunluk--` → 1.

düğüm = [veri | sonraki]

Adım 5 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
▶    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 0` `top = düğüm(10)` `sonuc = 15` `uzunluk = 1`

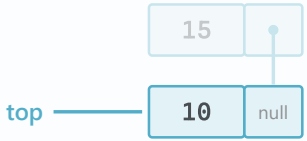
ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

Kod İzleme: Bağlı Liste ile `pop`



düğüm = [veri | sonraki]

return 15. Erişilemeyen düğümü çöp toplayıcı temizler.

Adım 6 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
▶    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 0` `top = düğüm(10)` `sonuc = 15` `uzunluk = 1`

ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

Kod İzleme: Bağlı Liste ile `pop`



main, dönen **15** değerini yazdırır.

düğüm = [veri | sonraki]

Adım 7 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
▶    System.out.println(pop());
```

DEĞİŞKENLER

`k = 0` `top = düğüm(10)` `uzunluk = 1`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
15
```

Kod İzleme: Bağlı Liste ile `pop`



pop(): önce boşluk kontrolü; top null değil → devam.

düğüm = [veri | sonraki]

Adım 8 / 15

```
1  public int pop() {
▶    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 1` `top = düğüm(10)` `uzunluk = 1`

ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

```
15
```

Kod İzleme: Bağlı Liste ile `pop`



`sonuc = top.veri` → **10** saklanır (düğüm kopmadan önce).

düğüm = [veri | sonraki]

Adım 9 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
▶    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 1` `top = düğüm(10)` `sonuc = 10` `uzunluk = 1`

ÇAĞRI YIĞINI

- `pop()`
- `main()`

ÇIKTI

```
15
```

GERÇEKLEME · KODU ADIM ADIM ÇALIŞTIR

Kod İzleme: Bağlı Liste ile `pop`



`top = top.sonraki` : top bir aşağı iner → **null**. Eski üst düğüme artık referans yok.

düğüm = [veri | sonraki]

Adım 10 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
▶    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 1` `top = null` `sonuc = 10` `uzunluk = 1`

ÇAĞRI YIĞINI

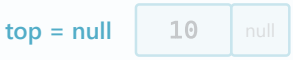
`pop()`

`main()`

ÇIKTI

```
15
```

Kod İzleme: Bağlı Liste ile `pop`



`uzunluk--` → 0.

düğüm = [veri | sonraki]

Adım 11 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
▶    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 1` `top = null` `sonuc = 10` `uzunluk = 0`

ÇAĞRI YIĞINI

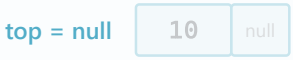
`pop()`

`main()`

ÇIKTI

```
15
```

Kod İzleme: Bağlı Liste ile `pop`



return 10. Erişilemeyen düğümü çöp toplayıcı temizler.

düğüm = [veri | sonraki]

Adım 12 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
▶    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 1` `top = null` `sonuc = 10` `uzunluk = 0`

ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

```
15
```

Kod İzleme: Bağlı Liste ile `pop`

top = null

main, dönen **10** değerini yazdırır.

düğüm = [veri | sonraki]

Adım 13 / 15

```
1  public int pop() {
2    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
▶    System.out.println(pop());
```

DEĞİŞKENLER

`k = 1` `top = null` `uzunluk = 0`

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
15
10
```

Kod İzleme: Bağlı Liste ile `pop`

top = null

pop(): **top == null** → yığın boş!

düğüm = [veri | sonraki]

Adım 14 / 15

```
1  public int pop() {
▶    if (top == null)
3      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 2` `top = null` `uzunluk = 0`

ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

```
15
10
```

Kod İzleme: Bağlı Liste ile `pop`

top = null

Alt taşma istisnası; program durur. Kontrol olmasaydı `top.veri` NullPointerException verirdi.

düğüm = [veri | sonraki]

Adım 15 / 15

```
1  public int pop() {
2    if (top == null)
▶      throw new RuntimeException("boş");
4    int sonuc = top.veri;
5    top = top.sonraki;
6    uzunluk--;
7    return sonuc;
8  }
9  // main içinde (yığın: 15 → 10):
10 for (int k = 0; k < 3; k++)
11   System.out.println(pop());
```

DEĞİŞKENLER

`k = 2` `top = null` `uzunluk = 0`

ÇAĞRI YIĞINI

`pop()`

`main()`

ÇIKTI

```
15
10
RuntimeException: boş
```

GERÇEKLEME

Dizi mi, Bağlı Liste mi?

Ölçüt	Dizi tabanlı	Bağlı liste tabanlı
Boyut	sabit kapasite (dinamik dizide büyütülür)	esnek, çalışma zamanında değişir
push / pop	O(1) (büyütmede amortize O(1))	O(1)
Taşma	taşma mümkün	yalnızca bellek biterse
Öğe başına bellek	yalnızca veri	veri + sonraki referansı
Boş yer	kullanılmayan kapasite	yok
Java	`ArrayDeque`, `Stack` (Vector tabanlı)	`LinkedList` (Deque olarak)

Java'da yeni kodda `Deque<Integer> y = new ArrayDeque<>();` ile `push` / `pop` / `peek` önerilir; eski `Stack` sınıfı senkronize olduğu için daha yavaştır.

YIĞIN UYGULAMALARI

Yığınların Kullanıldığı Yerler

İfadeler
Infix, prefix, postfix dönüşümü ve değerlendirmesi; parantez kontrolü.

Geri al / yinele
Metin düzenleyicilerde undo ve redo işlemleri.

Tarayıcı geçmişi
“Geri” ve “ileri” düğmeleri iki yığınla yönetilir.

Çağrı yığını
Fonksiyon çağrıları, yerel değişkenler ve dönüş adresleri (call stack).

Geri izleme
Labirentte yol bulma, n-vezir, Şövalye Turu (backtracking).

Algoritmalar
Hanoi kuleleri, ağaç gezinme, stock span, histogram problemleri.

Çizge algoritmaları
Derinlik öncelikli arama, topolojik sıralama, güçlü bağlantılı bileşenler.

Dizgi ters çevirme
Karakterler sırayla eklenir, ters sırayla çıkar.

Sonraki büyük öğe
Her öğe için sağındaki ilk büyük değer.

YIĞIN UYGULAMALARI

Tarayıcıda Geri ve İleri: İki Yığın



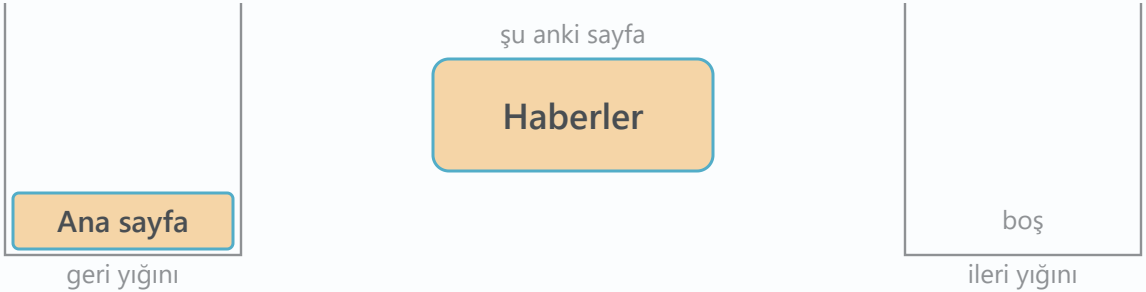
Tarayıcı açıldı: şu anki sayfa **Ana sayfa**; iki yığın da boş.

Adım 1 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

Tarayıcıda Geri ve İleri: İki Yığın

`ziyaret("Haberler")`



ziyaret(Haberler): eski sayfa geri yığınına **push**, yeni sayfa açılır.

Adım 2 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

Tarayıcıda Geri ve İleri: İki Yığın



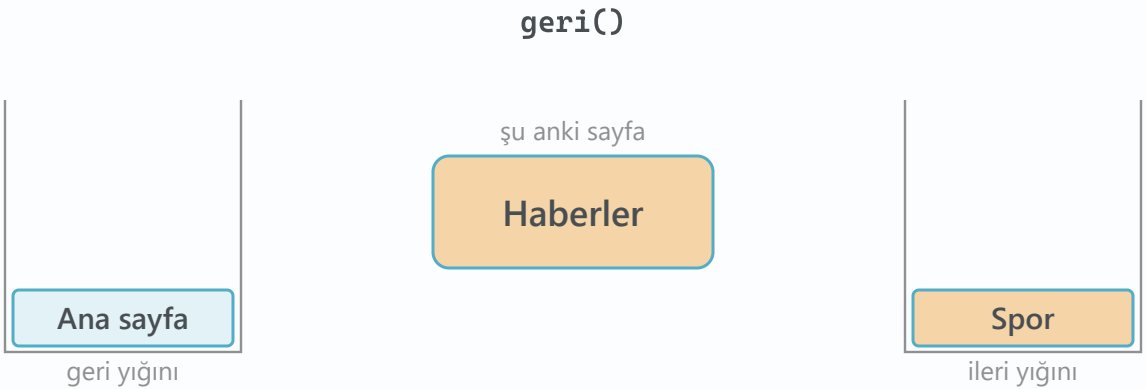
ziyaret(Spor): eski sayfa geri yığınına **push**, yeni sayfa açılır.

Adım 3 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

YIĞIN UYGULAMALARI

Tarayıcıda Geri ve İleri: İki Yığın



geri(): şu anki sayfa ileri yığınına push; geri yığınından **pop** → Haberler.

Adım 4 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

YIĞIN UYGULAMALARI

Tarayıcıda Geri ve İleri: İki Yığın

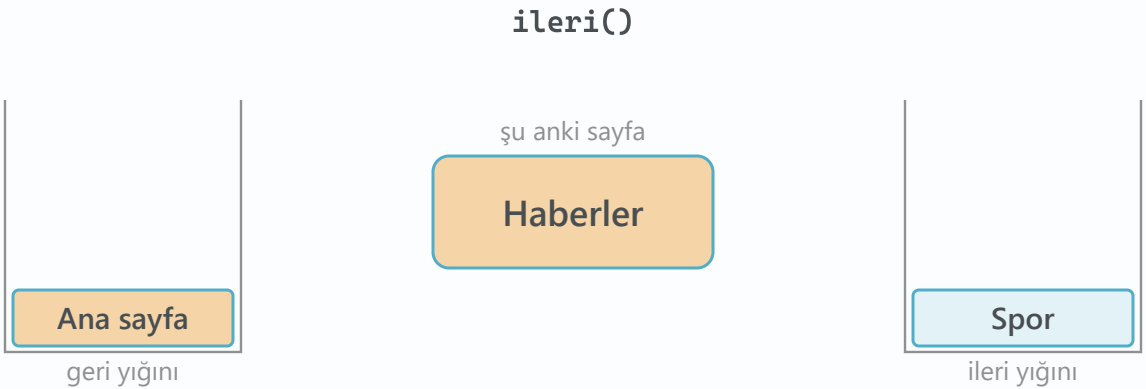


geri(): şu anki sayfa ileri yığınına push; geri yığınından **pop** → Ana sayfa.

Adım 5 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

Tarayıcıda Geri ve İleri: İki Yığın



ileri(): şu anki sayfa geri yığınına push; ileri yığınından pop → Haberler.

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

YIĞIN UYGULAMALARI

Tarayıcıda Geri ve İleri: İki Yığın



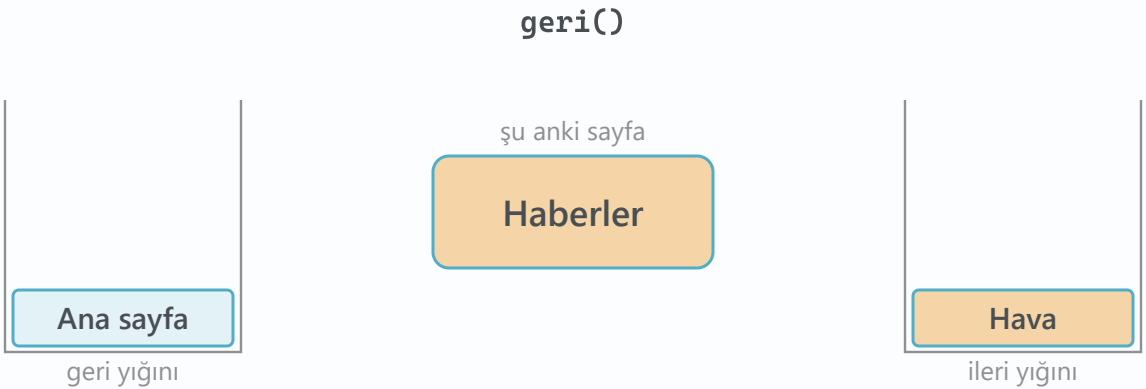
ziyaret(Hava): eski sayfa geri yığınına **push**, yeni sayfa açılır. Yeni bir yola girildiği için **ileri yığını boşaltılır**.

Adım 7 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

YIĞIN UYGULAMALARI

Tarayıcıda Geri ve İleri: İki Yığın

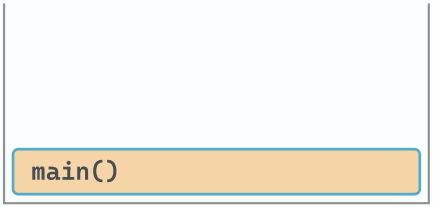


geri(): şu anki sayfa ileri yığınına push; geri yığınından **pop** → Haberler.

Adım 8 / 8

Kural: **ziyaret** → geri'ye push, ileri'yi temizle · **geri** → ileri'ye push, geri'den pop · **ileri** → geri'ye push, ileri'den pop. Her işlem O(1). Metin düzenleyicideki **geri al / yinele** (undo/redo) aynı düzeni kullanır.

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

main() çalışıyor

çerçeve = parametreler
+ yerel değişkenler
+ dönüş adresi

main çalışıyor; `fakt(3)` çağrılacak. Her çağrı yığına yeni bir **yığın çerçevesi** (stack frame) iter.

Adım 1 / 10

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

DEĞİŞKENLER

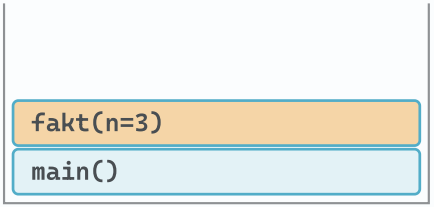
—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

push: fakt(3)

yığın derinliği = 1

`fakt(3)` çerçevesi yığına **push** edildi. `n <= 1` ? 3 <= 1 → **false**.

Adım 2 / 10

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

DEĞİŞKENLER

`n = 3`

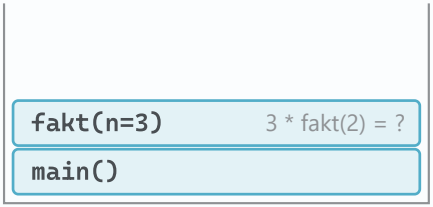
ÇAĞRI YIĞINI

`fakt(n=3)`

`main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

fakt(3) askıda

önce fakt(2) bitmeli

`3 * fakt(2)` hesaplanamaz: önce fakt(2) bitmeli. Bu çerçeve **askıda** bekler.

Adım 3 / 10

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

DEĞİŞKENLER

`n = 3`

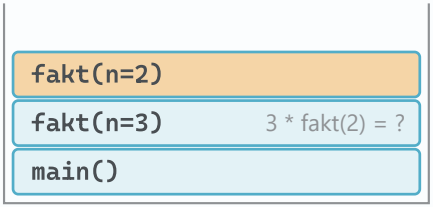
ÇAĞRI YIĞINI

`fakt(n=3)`

`main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

push: fakt(2)

yığın derinliği = 2

`fakt(2)` çerçevesi yığına **push** edildi. `n <= 1` ? 2 <= 1 → **false**.

Adım 4 / 10

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

DEĞİŞKENLER

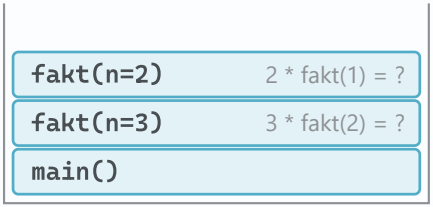
`n = 2`

ÇAĞRI YIĞINI

- `fakt(n=2)`
- `fakt(n=3)`
- `main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

fakt(2) askıda

önce fakt(1) bitmeli

`2 * fakt(1)` hesaplanamaz: önce fakt(1) bitmeli. Bu çerçeve **askıda** bekler.

Adım 5 / 10

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

DEĞİŞKENLER

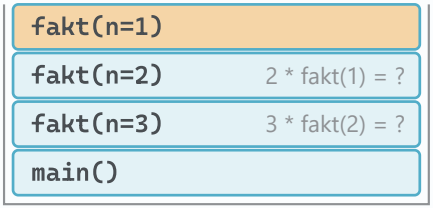
`n = 2`

ÇAĞRI YIĞINI

- `fakt(n=2)`
- `fakt(n=3)`
- `main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

push: fakt(1)

yığın derinliği = 3

`fakt(1)` çerçevesi yığına **push** edildi. `n <= 1` ? 1 <= 1 → **true**.

Adım 6 / 10

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

DEĞİŞKENLER

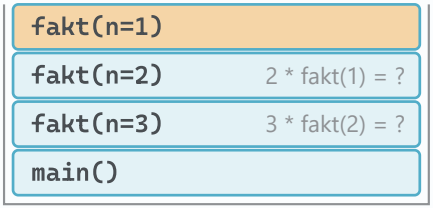
`n = 1`

ÇAĞRI YIĞINI

- `fakt(n=1)`
- `fakt(n=2)`
- `fakt(n=3)`
- `main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

taban durum

return 1

Taban durum: **return 1**. Artık yığın **çözülmeye** (unwinding) başlar.

Adım 7 / 10

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

DEĞİŞKENLER

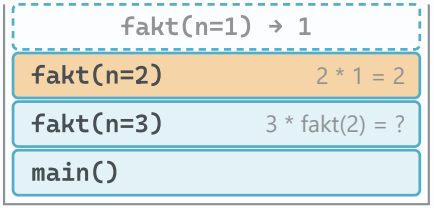
`n = 1`

ÇAĞRI YIĞINI

- `fakt(n=1)`
- `fakt(n=2)`
- `fakt(n=3)`
- `main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

pop: fakt(1) → 1

2 * 1 = 2

fakt(1) **pop** edildi, 1 döndü.
fakt(2) devam eder: 2 * 1 = **2**.

Adım 8 / 10

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

DEĞİŞKENLER

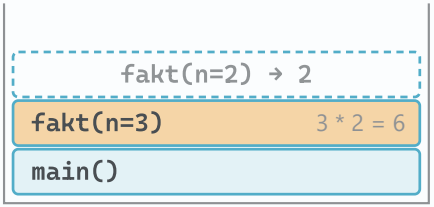
`n = 2` `fakt(1) = 1`

ÇAĞRI YIĞINI

- `fakt(n=2)`
- `fakt(n=3)`
- `main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

pop: fakt(2) → 2

3 * 2 = 6

fakt(2) **pop** edildi, 2 döndü.
fakt(3) devam eder: 3 * 2 = **6**.

Adım 9 / 10

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

DEĞİŞKENLER

`n = 3` `fakt(2) = 2`

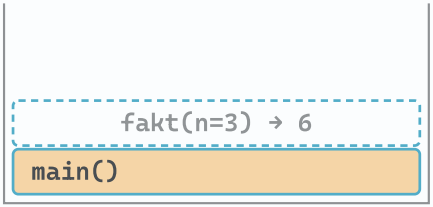
ÇAĞRI YIĞINI

`fakt(n=3)`

`main()`

ÇIKTI

Çağrı Yığını: `fakt(3)` Adım Adım



çağrı yığını (call stack)

pop: fakt(3) → 6

println(6)

fakt(3) = **6** yazdırılır. Derinlik n → **O(n)** bellek; çok derin özyineleme `StackOverflowError` verir.

Adım 10 / 10

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

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

```
6
```

Dizgi Tersine Çevirme: Adım Adım

`str = "ABCD"`

main, `tersCevir("ABCD")` çağırır.

Adım 1 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
▶  System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım

`str = "ABCD"`



Boş bir karakter yığını oluşturulur.

Adım 2 / 13

```
1  String tersCevir(String str) {
▶      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"` `yigin = []`

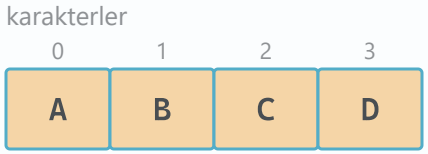
ÇAĞRI YIĞINI

`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





Dizgi karakter dizisine kopyalanır (üzerine yazacağız).

Adım 3 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
▶      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

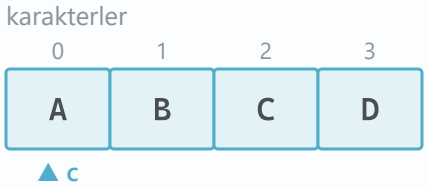
- `str = "ABCD"`
- `karakterler = [A, B, C, D]`
- `yigin = []`

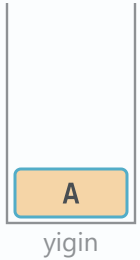
ÇAĞRI YIĞINI

- `tersCevir("ABCD")`
- `main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





1. döngü: c = 'A' → **push('A')**.

Adım 4 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
▶          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [A, B, C, D]` `c = A`

`yigin = [A]`

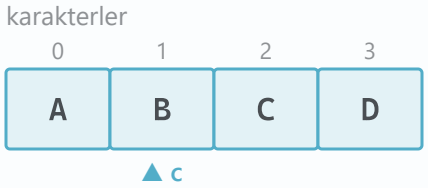
ÇAĞRI YIĞINI

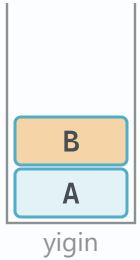
`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





1. döngü: c = 'B' → **push('B')**.

Adım 5 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
▶          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [A, B, C, D]` `c = B`

`yigin = [A, B]`

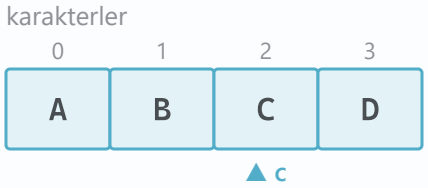
ÇAĞRI YIĞINI

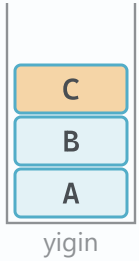
`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





1. döngü: c = 'C' → **push('C')**.

Adım 6 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
▶          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [A, B, C, D]` `c = C`

`yigin = [A, B, C]`

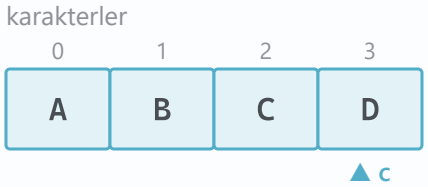
ÇAĞRI YIĞINI

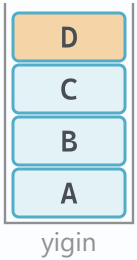
`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





1. döngü: c = 'D' → **push('D')**. Tüm karakterler yığında; en üstte son karakter.

Adım 7 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
▶          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [A, B, C, D]` `c = D`

`yigin = [A, B, C, D]`

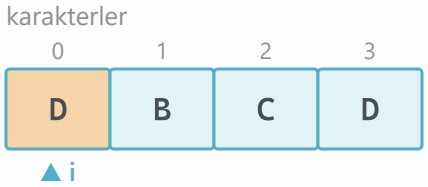
ÇAĞRI YIĞINI

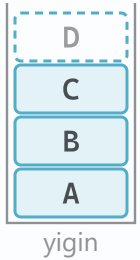
`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





2. döngü, i = 0: **pop()** → 'D', karakterler[0] = 'D'. Çıkış sırası girişin **tersi**.

Adım 8 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
▶          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [D, B, C, D]` `i = 0`

`yigin = [A, B, C]`

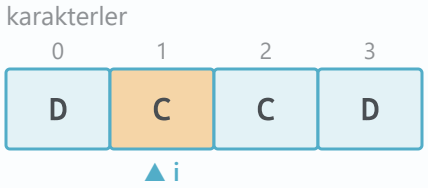
ÇAĞRI YIĞINI

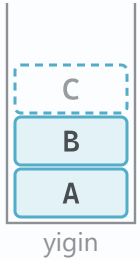
`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





2. döngü, i = 1: **pop()** → 'C', karakterler[1] = 'C'. Çıkış sırası girişin **tersi**.

Adım 9 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
▶          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [D, C, C, D]` `i = 1`

`yigin = [A, B]`

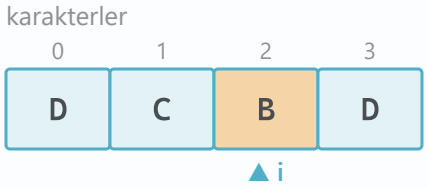
ÇAĞRI YIĞINI

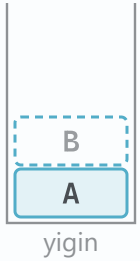
`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





2. döngü, i = 2: **pop()** → 'B', karakterler[2] = 'B'. Çıkış sırası girişin **tersi**.

Adım 10 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
▶          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [D, C, B, D]` `i = 2`

`yigin = [A]`

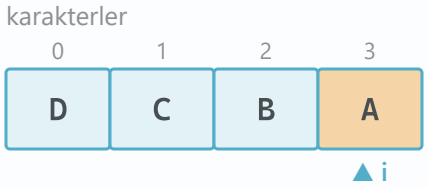
ÇAĞRI YIĞINI

`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





2. döngü, i = 3: **pop()** → 'A', karakterler[3] = 'A'. Çıkış sırası girişin **tersi**.

Adım 11 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
▶          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

`str = "ABCD"`

`karakterler = [D, C, B, A]` `i = 3`

`yigin = []`

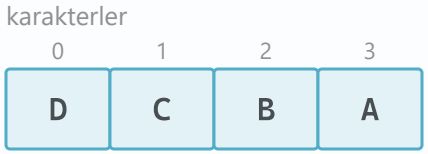
ÇAĞRI YIĞINI

`tersCevir("ABCD")`

`main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





return "DCBA". Zaman ve ek bellek **O(n)**.

Adım 12 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
▶      return new String(karakterler);
9  }
10 // main içinde:
11 System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

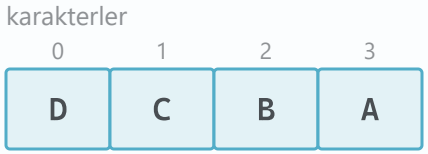
- `str = "ABCD"`
- `karakterler = [D, C, B, A]`
- `yigin = []`

ÇAĞRI YIĞINI

- `tersCevir("ABCD")`
- `main()`

ÇIKTI

Dizgi Tersine Çevirme: Adım Adım





main sonucu yazdırır: **DCBA**.

Adım 13 / 13

```
1  String tersCevir(String str) {
2      Deque<Character> yigin = new ArrayDeque<>
   ();
3      char[] karakterler = str.toCharArray();
4      for (char c : karakterler)
5          yigin.push(c);
6      for (int i = 0; i < str.length(); i++)
7          karakterler[i] = yigin.pop();
8      return new String(karakterler);
9  }
10 // main içinde:
▶  System.out.println(tersCevir("ABCD"));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

DCBA

YIĞIN UYGULAMALARI

Parantez Kontrolü

S dizisi yalnızca `( ) { } [ ]` karakterlerinden oluşsun. S, ancak ve ancak şu iki koşul sağlanırsa **geçerlidir**:

1. her açık parantez **aynı tip** bir parantezle kapatılmışsa ve
2. parantezler **doğru sırada** kapatılmışsa (en son açılan ilk kapanır → LIFO!).

Algoritma: eşleşmemiş açık parantezler yığında tutulur. S baştan sona gezilir:

- Açık parantez → yığına **push**.
- Kapalı parantez → yığın boşsa **geçersiz**; değilse **pop**, çıkan aynı tipte değilse **geçersiz**.
- S bitince yığın **boşsa geçerli**, değilse geçersiz (kapanmamış parantez var).

`{[()]}`	`([)]`	`(()`	`())`
geçerli	sıra yanlış	kapanmamış	fazla kapanış

Parantez Kontrolü: Java

```java
boolean gecerliMi(String s) {
  Deque<Character> yigin = new ArrayDeque<>();
  for (char c : s.toCharArray()) {
    if (c == '(' || c == '[' || c == '{') {
      yigin.push(c);
    } else {
      if (yigin.isEmpty()) return false;
      char acik = yigin.pop();
      if (!eslesir(acik, c)) return false;
    }
  }
  return yigin.isEmpty();
}
```

Yardımcı metot

```java
boolean eslesir(char a, char k) {
  return (a == '(' && k == ')')
      || (a == '[' && k == ']')
      || (a == '{' && k == '}');
}
```

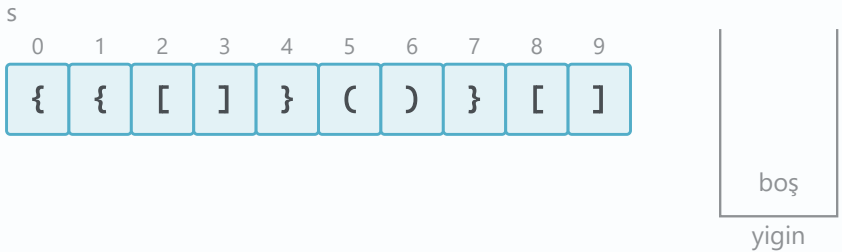
Boş yığın kontrolü

Olmasaydı `"())"` ya da `"]"` girdisinde boş yığından pop → istisna.

Karmaşıklık

Zaman **O(n)**, en kötü durumda yığın **O(n)** bellek.

Kod İzleme: `gecerliMi("{{[]}()}[]")`



Eşleşmemiş açık parantezleri tutacak boş yığın oluşturulur.

Adım 1 / 17

```
1  boolean gecerliMi(String s) {
▶    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

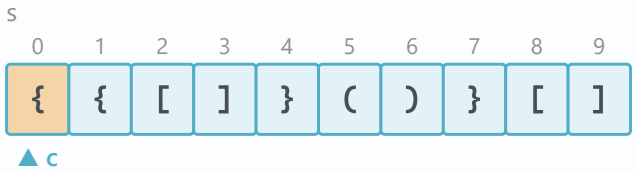
`s = "{{[]}()}[]"` `yigin = []`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



'{' açık parantez → **push**. Kapanışını bekleyecek.

Adım 2 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '{'`

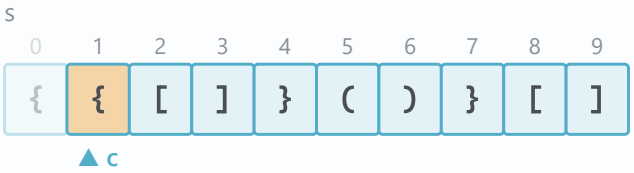
`yigin = ['{']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



'{' açık parantez → **push**. Kapanışını bekleyecek.

Adım 3 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '{'`

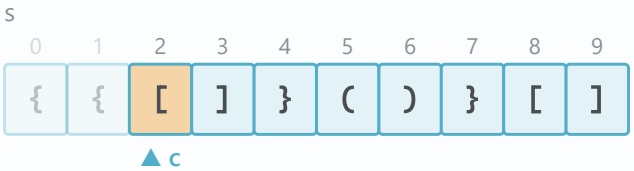
`yigin = ['{', '{']`

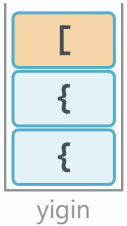
ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`





'[' açık parantez → **push**. Kapanışını bekleyecek.

Adım 4 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '['`

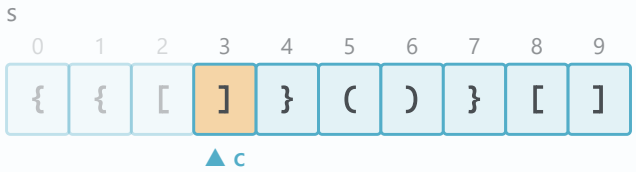
`yigin = ['{', '{', '[']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



']' kapalı parantez: yığın boş değil → **pop**, en son açılan '[' çıkar.

Adım 5 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
▶        if (yigin.isEmpty()) return false;
▶        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = ']'`

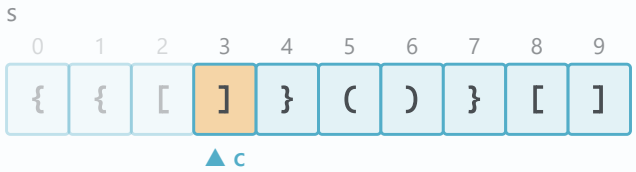
`acik = '['` `yigin = ['{', '{']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



eslesir('[', ']') → **true**: aynı tip, doğru sıra. Devam.

Adım 6 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
▶        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = ']'`

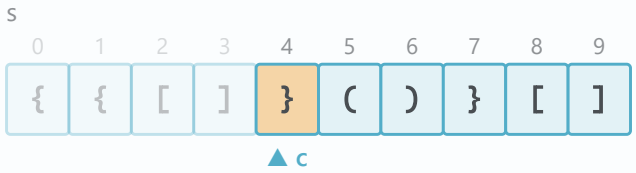
`acik = '['` `yigin = ['{', '{']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



'}' kapalı parantez: yığın boş değil → **pop**, en son açılan '{' çıkar.

Adım 7 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
▶        if (yigin.isEmpty()) return false;
▶        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '}'`

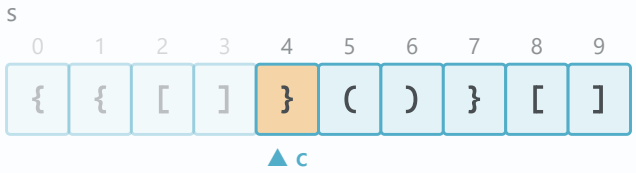
`acik = '{'` `yigin = ['{']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



eslesir('{', '}') → **true**: aynı tip, doğru sıra. Devam.

Adım 8 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
▶        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '}'`

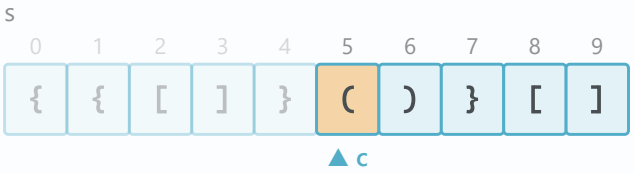
`acik = '{'` `yigin = ['{']`

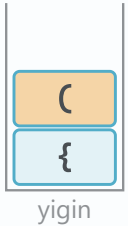
ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`





'(' açık parantez → **push**. Kapanışını bekleyecek.

Adım 9 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '('`

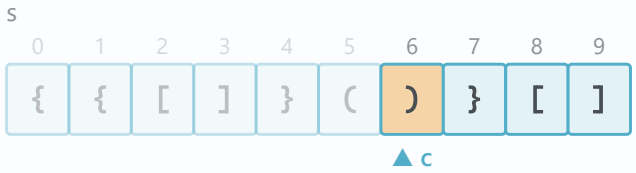
`yigin = ['{', '(']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



')' kapalı parantez: yığın boş değil → **pop**, en son açılan '(' çıkar.

Adım 10 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
▶        if (yigin.isEmpty()) return false;
▶        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

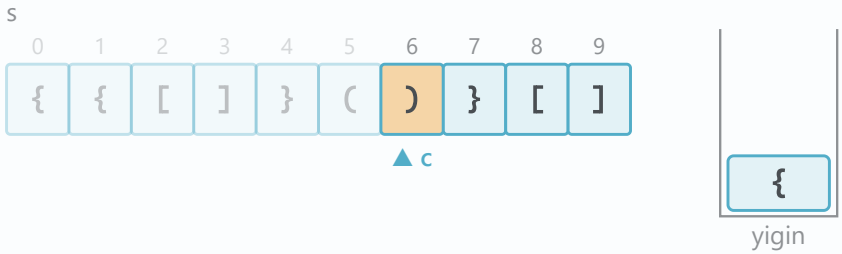
`s = "{{[]}()}[]"` `c = ')'` `acik = '('` `yigin = ['{']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



eslesir('(', ')') → **true**: aynı tip, doğru sıra. Devam.

Adım 11 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
▶        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = ')'`

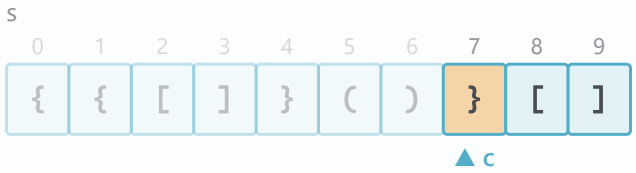
`acik = '('` `yigin = ['{']`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



'}' kapalı parantez: yığın boş değil → **pop**, en son açılan '{' çıkar.

Adım 12 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
▶        if (yigin.isEmpty()) return false;
▶        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

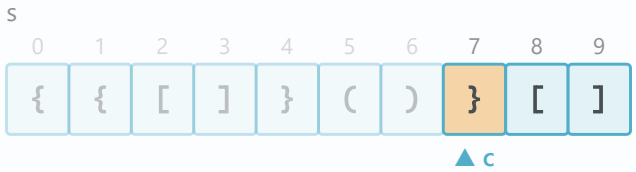
`s = "{{[]}()}[]"` `c = '}'` `acik = '{'` `yigin = []`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



eslesir('{', '}') → **true**: aynı tip, doğru sıra. Devam.

Adım 13 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
▶        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

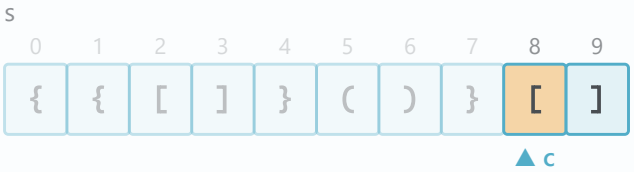
DEĞİŞKENLER

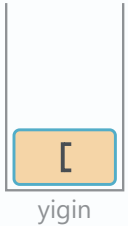
`s = "{{[]}()}[]"` `c = '}'`
`acik = '{'` `yigin = []`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`
`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`





'[' açık parantez → **push**. Kapanışını bekleyecek.

Adım 14 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = '['` `yigin = ['[']`

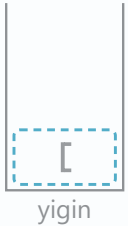
ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`





']' kapalı parantez: yığın boş değil → **pop**, en son açılan '[' çıkar.

Adım 15 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
▶        if (yigin.isEmpty()) return false;
▶        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

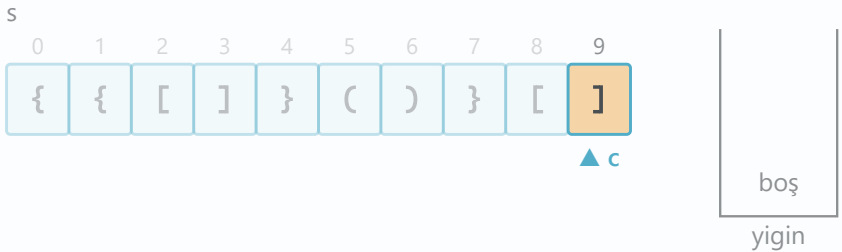
`s = "{{[]}()}[]"` `c = ']'` `acik = '['` `yigin = []`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



eslesir('[', ']') → **true**: aynı tip, doğru sıra. Devam.

Adım 16 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
▶        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "{{[]}()}[]"` `c = ']'`

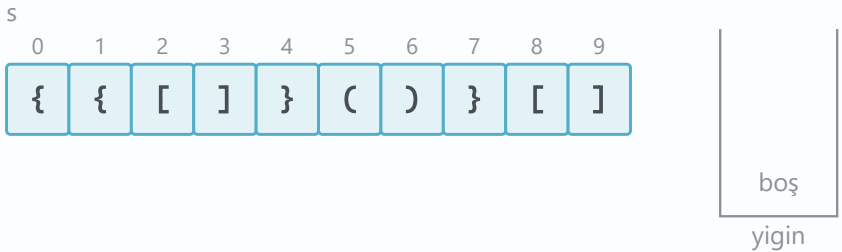
`acik = '['` `yigin = []`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("{{[]}()}[]")`



Dizgi bitti, yığın **boş** → **return true**: dizgi geçerli. Her karakter bir kez işlendi → O(n).

Adım 17 / 17

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
▶    return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

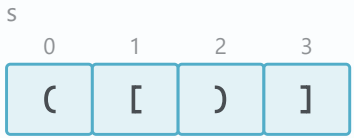
`s = "{{[]}()}[]"` `yigin = []`

ÇAĞRI YIĞINI

`gecerliMi("{{[]}()}[]")`

`main()`

Kod İzleme: `gecerliMi("([])")`





Eşleşmemiş açık parantezleri tutacak boş yığın oluşturulur.

Adım 1 / 5

```
1  boolean gecerliMi(String s) {
▶    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

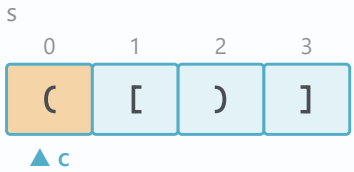
`s = "([])"` `yigin = []`

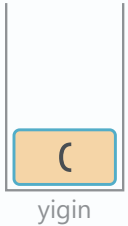
ÇAĞRI YIĞINI

`gecerliMi("([])")`

`main()`

Kod İzleme: `gecerliMi("([])")`





'(' açık parantez → **push**. Kapanışını bekleyecek.

Adım 2 / 5

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

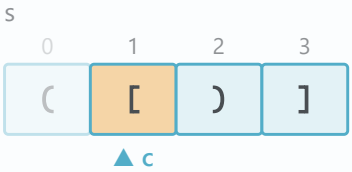
`s = "([])"` `c = '('` `yigin = ['(']`

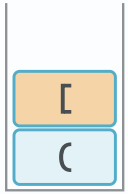
ÇAĞRI YIĞINI

`gecerliMi("([])")`

`main()`

Kod İzleme: `gecerliMi("([])")`





'[' açık parantez → **push**.
Kapanışını bekleyecek.

Adım 3 / 5

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
▶      if (c == '(' || c == '[' || c == '{') {
▶        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "([])"` `c = '['`

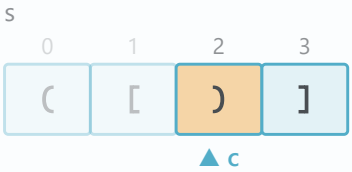
`yigin = ['(', '[']`

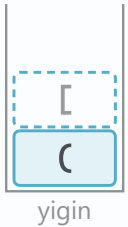
ÇAĞRI YIĞINI

`gecerliMi("([])")`

`main()`

Kod İzleme: `gecerliMi("([])")`





')' kapalı parantez: yığın boş değil → **pop**, en son açılan '[' çıkar.

Adım 4 / 5

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
▶        if (yigin.isEmpty()) return false;
▶        char acik = yigin.pop();
9        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

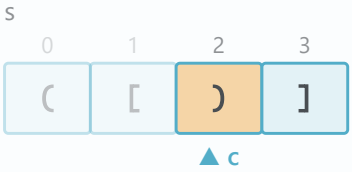
`s = "([])"` `c = ')'` `acik = '['` `yigin = ['(']`

ÇAĞRI YIĞINI

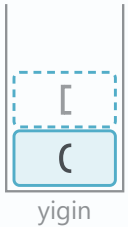
`gecerliMi("([])")`

`main()`

Kod İzleme: `gecerliMi("([])")`



eşleşmedi!



eslesir('[', ')') → **false**: ')' kapanıyor ama en son açılan '['. **return false**. Benzer biçimde `(()` sonda yığın dolu kaldığı, `())` ise boş yığında kapanış geldiği için geçersizdir.

Adım 5 / 5

```
1  boolean gecerliMi(String s) {
2    Deque<Character> yigin = new ArrayDeque<>();
3    for (char c : s.toCharArray()) {
4      if (c == '(' || c == '[' || c == '{') {
5        yigin.push(c);
6      } else {
7        if (yigin.isEmpty()) return false;
8        char acik = yigin.pop();
▶        if (!eslesir(acik, c)) return false;
10     }
11   }
12   return yigin.isEmpty();
13 }
```

DEĞİŞKENLER

`s = "([])"` `c = ')'` `acik = '['` `yigin = ['(']`

ÇAĞRI YIĞINI

`gecerliMi("([])")`

`main()`

Sonraki Büyük Öğe (Next Greater Element)

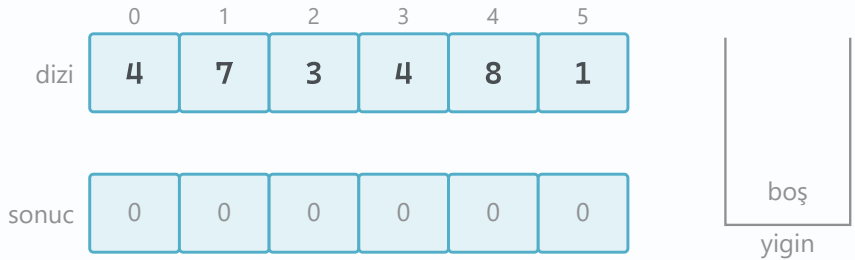
Dizideki her öğe için **sağında kalan, kendisinden büyük ilk** öğeyi bul; yoksa −1.

dizi	4	7	3	4	8	1
sonuc	7	8	4	8	−1	−1

- Kaba kuvvet: her öğe için sağını tara → $O(n^2)$.
- Yığınla: dizi **sağdan sola** gezilir; yığında yalnızca "aday" büyük değerler kalır → **O(n)**.
- Yığın, alttan üste doğru **azalan** sırada kalır (monoton yığın).

```
int[] sonrakiBuyuk(int[] dizi) {
    int[] sonuc = new int[dizi.length];
    Deque<Integer> yigin = new ArrayDeque<>();
    for (int i = dizi.length-1; i >= 0; i--) {
        while (!yigin.isEmpty() &&
               yigin.peek() <= dizi[i])
            yigin.pop();
        sonuc[i] = yigin.isEmpty()
                 ? -1 : yigin.peek();
        yigin.push(dizi[i]);
    }
    return sonuc;
}
```

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



sonuc (hepsi 0) ve boş yığın oluşturulur. Dizi **sağdan sola** gezilecek.

Adım 1 / 17

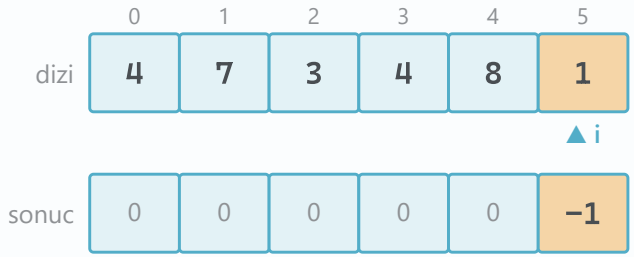
```
1  int[] sonrakiBuyuk(int[] dizi) {
▶      int[] sonuc = new int[dizi.length];
▶      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

DEĞİŞKENLER

`sonuc = [0, 0, 0, 0, 0, 0]`

`yigin = []`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`





i = 5 (dizi[i] = 1): yığın boş → **−1**.

Adım 2 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
▶          sonuc[i] = yigin.isEmpty()
▶                  ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

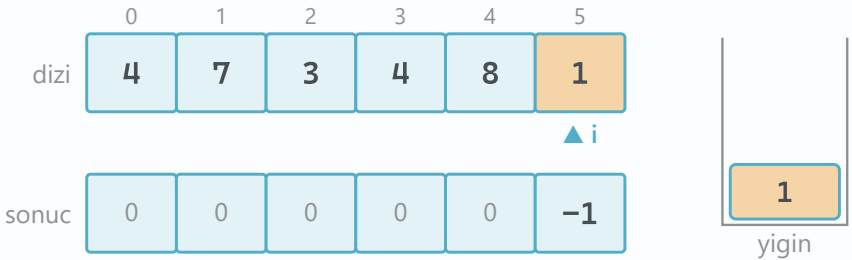
DEĞİŞKENLER

`i = 5` `dizi[i] = 1`

`sonuc = [0, 0, 0, 0, 0, −1]`

`yigin = []`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



push(1): soldaki öğeler için yeni aday. Yığın alttan üste azalan kalır.

Adım 3 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
▶          yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

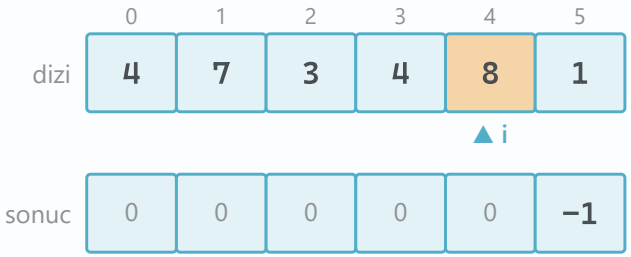
DEĞİŞKENLER

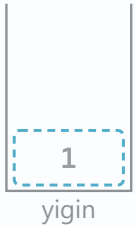
`i = 5` `dizi[i] = 1`

`sonuc = [0, 0, 0, 0, 0, -1]`

`yigin = [1]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`





i = 4: üstteki 1 ≤ 8 → **pop**. 1, 8'in solundakiler için asla cevap olamaz (arada daha büyük 8 var).

Adım 4 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
▶             yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

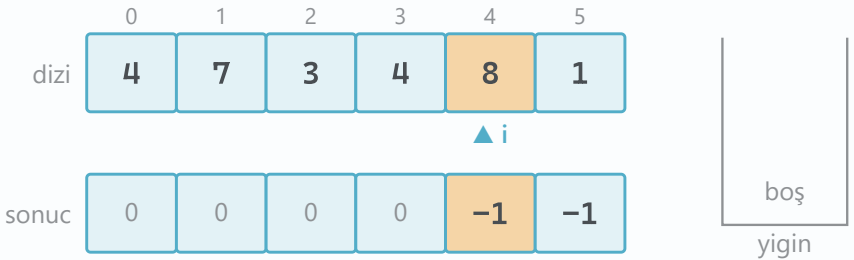
DEĞİŞKENLER

`i = 4` `dizi[i] = 8`

`sonuc = [0, 0, 0, 0, 0, -1]`

`yigin = []`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



i = 4 (dizi[i] = 8): yığın boş → **−1**.

Adım 5 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
▶          sonuc[i] = yigin.isEmpty()
▶                  ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

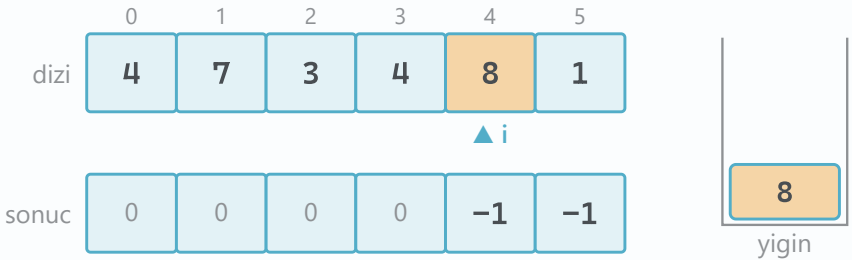
DEĞİŞKENLER

`i = 4` `dizi[i] = 8`

`sonuc = [0, 0, 0, 0, −1, −1]`

`yigin = []`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



push(8): soldaki öğeler için yeni aday. Yığın alttan üste azalan kalır.

Adım 6 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
▶          yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

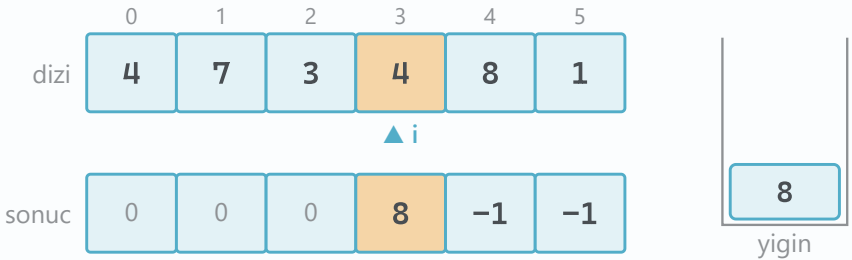
DEĞİŞKENLER

`i = 4` `dizi[i] = 8`

`sonuc = [0, 0, 0, 0, -1, -1]`

`yigin = [8]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



i = 3 (dizi[i] = 4): üstteki 8 > 4 → **sonuc[3] = 8**.

Adım 7 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
▶          sonuc[i] = yigin.isEmpty()
▶                  ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

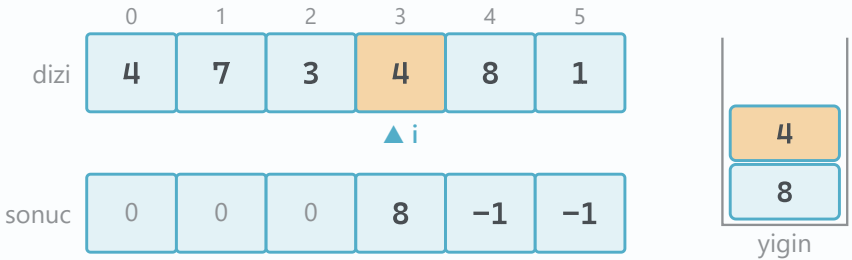
DEĞİŞKENLER

`i = 3` `dizi[i] = 4`

`sonuc = [0, 0, 0, 8, -1, -1]`

`yigin = [8]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



push(4): soldaki öğeler için yeni aday. Yığın alttan üste azalan kalır.

Adım 8 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
▶          yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

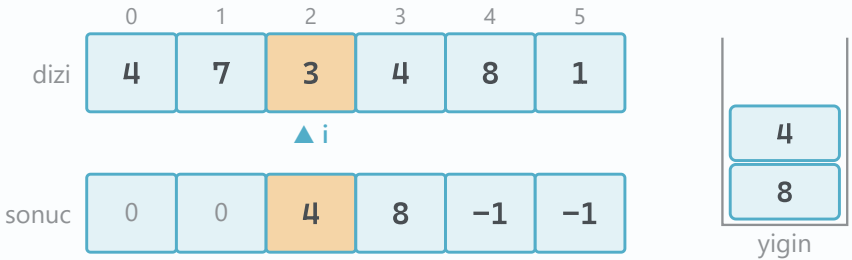
DEĞİŞKENLER

`i = 3` `dizi[i] = 4`

`sonuc = [0, 0, 0, 8, -1, -1]`

`yigin = [8, 4]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



i = 2 (dizi[i] = 3): üstteki 4 > 3 → **sonuc[2] = 4**.

Adım 9 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
▶          sonuc[i] = yigin.isEmpty()
▶                  ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

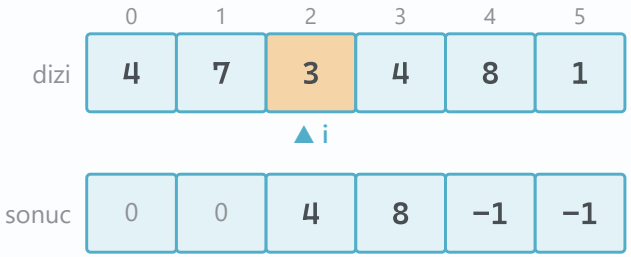
DEĞİŞKENLER

`i = 2` `dizi[i] = 3`

`sonuc = [0, 0, 4, 8, −1, −1]`

`yigin = [8, 4]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`





push(3): soldaki öğeler için yeni aday. Yığın alttan üste azalan kalır.

Adım 10 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
▶          yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

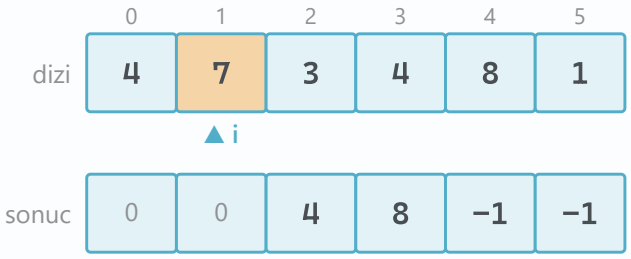
DEĞİŞKENLER

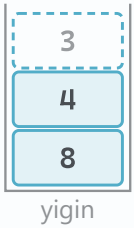
`i = 2` `dizi[i] = 3`

`sonuc = [0, 0, 4, 8, -1, -1]`

`yigin = [8, 4, 3]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`





i = 1: üstteki 3 ≤ 7 → **pop**. 3, 7'nin solundakiler için asla cevap olamaz (arada daha büyük 7 var).

Adım 11 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
▶             yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

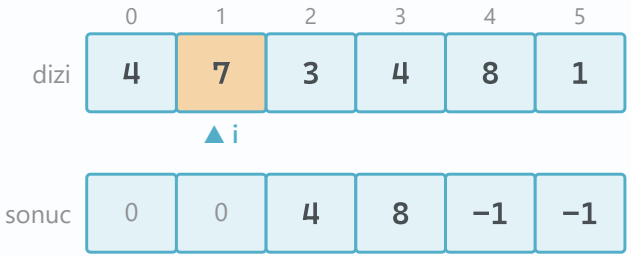
DEĞİŞKENLER

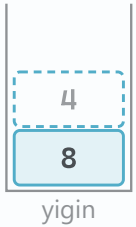
`i = 1` `dizi[i] = 7`

`sonuc = [0, 0, 4, 8, -1, -1]`

`yigin = [8, 4]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`





i = 1: üstteki 4 ≤ 7 → **pop**. 4, 7'nin solundakiler için asla cevap olamaz (arada daha büyük 7 var).

Adım 12 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
▶              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

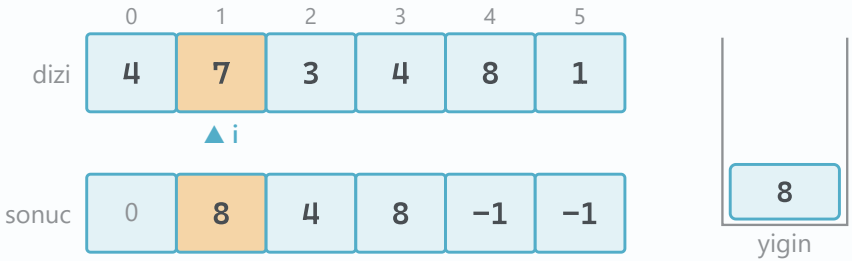
DEĞİŞKENLER

`i = 1` `dizi[i] = 7`

`sonuc = [0, 0, 4, 8, -1, -1]`

`yigin = [8]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



i = 1 (dizi[i] = 7): üstteki 8 > 7 → **sonuc[1] = 8**.

Adım 13 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
▶          sonuc[i] = yigin.isEmpty()
▶                  ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

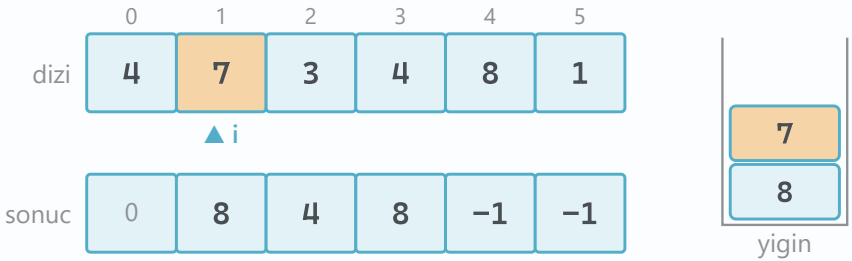
DEĞİŞKENLER

`i = 1` `dizi[i] = 7`

`sonuc = [0, 8, 4, 8, −1, −1]`

`yigin = [8]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



push(7): soldaki öğeler için yeni aday. Yığın alttan üste azalan kalır.

Adım 14 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
▶          yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

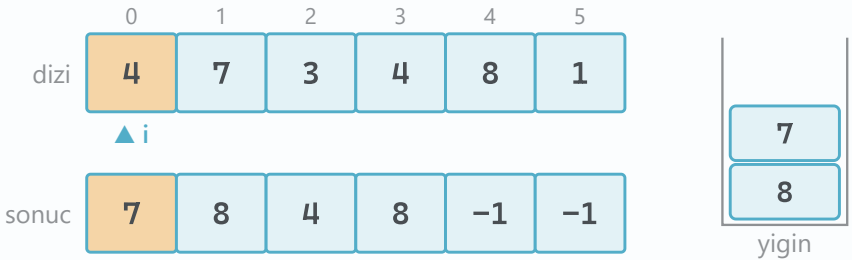
DEĞİŞKENLER

`i = 1` `dizi[i] = 7`

`sonuc = [0, 8, 4, 8, −1, −1]`

`yigin = [8, 7]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



i = 0 (dizi[i] = 4): üstteki 7 > 4 → **sonuc[0] = 7**.

Adım 15 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
▶          sonuc[i] = yigin.isEmpty()
▶                  ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

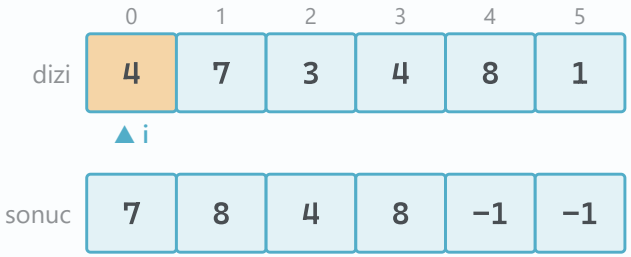
DEĞİŞKENLER

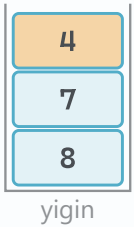
`i = 0` `dizi[i] = 4`

`sonuc = [7, 8, 4, 8, −1, −1]`

`yigin = [8, 7]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`





push(4): soldaki öğeler için yeni aday. Yığın alttan üste azalan kalır.

Adım 16 / 17

```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
▶          yigin.push(dizi[i]);
11     }
12     return sonuc;
13 }
```

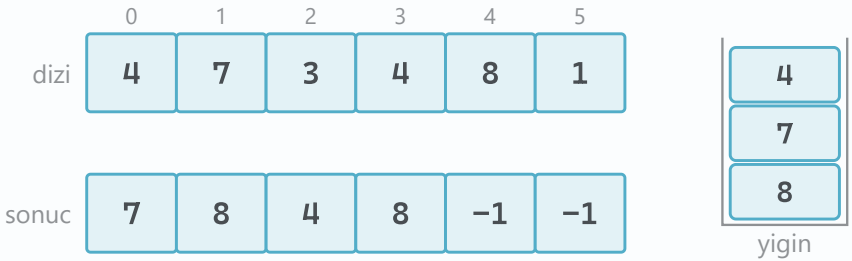
DEĞİŞKENLER

`i = 0` `dizi[i] = 4`

`sonuc = [7, 8, 4, 8, −1, −1]`

`yigin = [8, 7, 4]`

Kod İzleme: `sonrakiBuyuk({4, 7, 3, 4, 8, 1})`



return [7, 8, 4, 8, -1, -1]. Her öğe en fazla bir kez push ve bir kez pop edilir → **O(n)**.

Adım 17 / 17

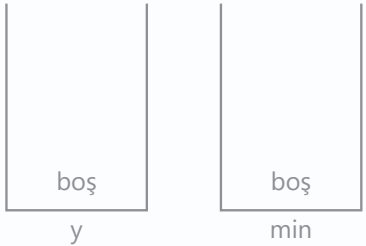
```
1  int[] sonrakiBuyuk(int[] dizi) {
2      int[] sonuc = new int[dizi.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      for (int i = dizi.length-1; i >= 0; i--) {
5          while (!yigin.isEmpty() &&
6                 yigin.peek() <= dizi[i])
7              yigin.pop();
8          sonuc[i] = yigin.isEmpty()
9                   ? -1 : yigin.peek();
10         yigin.push(dizi[i]);
11     }
▶      return sonuc;
13 }
```

DEĞİŞKENLER

`sonuc = [7, 8, 4, 8, −1, −1]`

`yigin = [8, 7, 4]`

Kod İzleme: `getMin()` O(1) Olan Yığın



`push(5)`	`pop()`
`push(3)`	`getMin()`
`push(7)`	`pop()`
`push(3)`	`pop()`
`getMin()`	`getMin()`

min'in üstü = o anki en küçük

İki yığın: **y** tüm öğeleri, **min** ise her an en küçüğü üstte tutar. Amaç: `getMin()` O(1).

Adım 1 / 15

```
▶ Deque<Integer> y = new ArrayDeque<>();
▶ Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

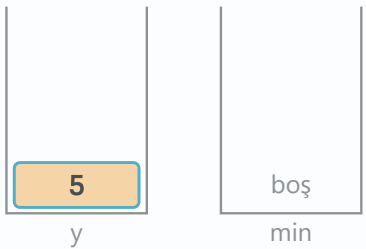
DEĞİŞKENLER

`y = []` `min = []`

ÇAĞRI YIĞINI

`main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5)	pop()
push(3)	getMin()
push(7)	pop()
push(3)	pop()
getMin()	getMin()

min'in üstü = o anki en küçük

push(5): `y.push(5)` .

Adım 2 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
▶    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

DEĞİŞKENLER

`y = [5]` `min = []` `x = 5`

ÇAĞRI YIĞINI

- `push(5)`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5)	pop()
push(3)	getMin()
push(7)	pop()
push(3)	pop()
getMin()	getMin()

min'in üstü = o anki en küçük

min boş → `min.push(5)` : yeni (ya da eşit) en küçük.

Adım 3 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
▶    if (min.isEmpty() || x <= min.peek())
▶      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

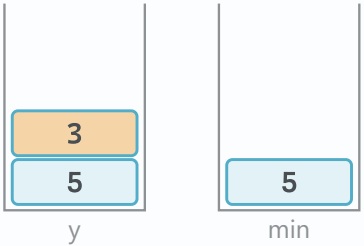
DEĞİŞKENLER

`y = [5]` `min = [5]` `x = 5`

ÇAĞRI YIĞINI

- `push(5)`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5)	pop()
push(3)	getMin()
push(7)	pop()
push(3)	pop()
getMin()	getMin()

min'in üstü = o anki en küçük

push(3): `y.push(3)` .

Adım 4 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
▶    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

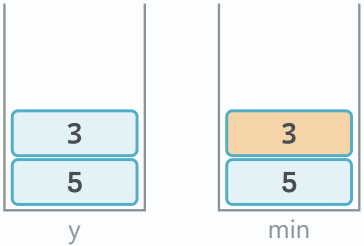
DEĞİŞKENLER

`y = [5, 3]` `min = [5]` `x = 3`

ÇAĞRI YIĞINI

- `push(3)`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5)	pop()
push(3)	getMin()
push(7)	pop()
push(3)	pop()
getMin()	getMin()

min'in üstü = o anki en küçük

3 <= 5 → true → `min.push(3)` : yeni (ya da eşit) en küçük.

Adım 5 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
▶    if (min.isEmpty() || x <= min.peek())
▶      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

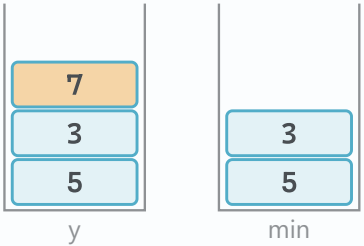
DEĞİŞKENLER

`y = [5, 3]` `min = [5, 3]` `x = 3`

ÇAĞRI YIĞINI

- `push(3)`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5) push(3) **push(7)** push(3) getMin()

pop() getMin() pop() pop() getMin()

min'in üstü = o anki en küçük

push(7): `y.push(7)`.

Adım 6 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
▶    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

DEĞİŞKENLER

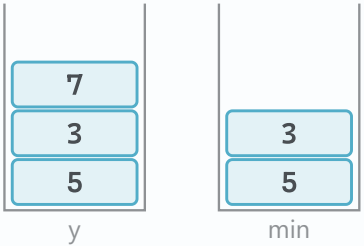
`y = [5, 3, 7]` `min = [5, 3]` `x = 7`

ÇAĞRI YIĞINI

`push(7)`

`main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5) push(3) **push(7)** push(3) getMin()

pop() getMin() pop() pop() getMin()

min'in üstü = o anki en küçük

7 <= 3 → false: 7 en küçük değil, min yığını **değişmez**.

Adım 7 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
▶    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

DEĞİŞKENLER

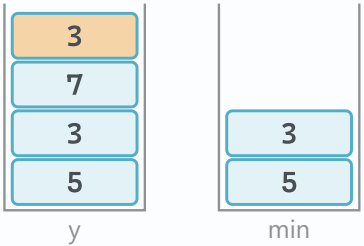
`y = [5, 3, 7]` `min = [5, 3]` `x = 7`

ÇAĞRI YIĞINI

`push(7)`

`main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5)	pop()
push(3)	getMin()
push(7)	pop()
push(3)	pop()
getMin()	getMin()

min'in üstü = o anki en küçük

push(3): `y.push(3)` .


```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
▶    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

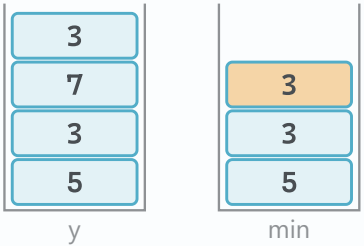
DEĞİŞKENLER

`y = [5, 3, 7, 3]` `min = [5, 3]` `x = 3`

ÇAĞRI YIĞINI

- `push(3)`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5) push(3) push(7) **push(3)** getMin()

pop() getMin() pop() pop() getMin()

min'in üstü = o anki en küçük

3 <= 3 → true → `min.push(3)` : yeni (ya da eşit) en küçük.

Adım 9 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
▶    if (min.isEmpty() || x <= min.peek())
▶      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

DEĞİŞKENLER

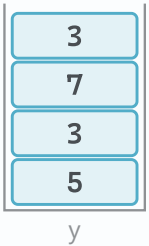
`y = [5, 3, 7, 3]` `min = [5, 3, 3]` `x = 3`

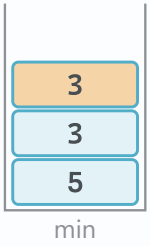
ÇAĞRI YIĞINI

push(3)

main()

Kod İzleme: `getMin()` O(1) Olan Yığın





`push(5)`	`pop()`
`push(3)`	`getMin()`
`push(7)`	`pop()`
`push(3)`	`pop()`
`getMin()` → 3	`getMin()`

min'in üstü = o anki en küçük

getMin(): `min.peek()` = **3**.
Arama yok → O(1).

Adım 10 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
▶ int getMin() { return min.peek(); }
```

DEĞİŞKENLER

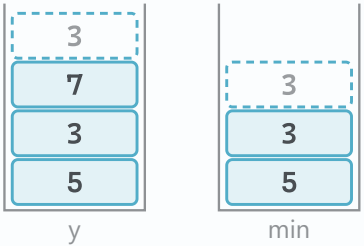
`y = [5, 3, 7, 3]` `min = [5, 3, 3]`

ÇAĞRI YIĞINI

`getMin()`

`main()`

Kod İzleme: `getMin()` O(1) Olan Yığın




```
push(5)
push(3)
push(7)
push(3)
getMin()  → 3
```

min'in üstü = o anki en küçük

```
pop()     → 3
getMin()
pop()
pop()
getMin()
```

pop(): y'den **3** çıktı. 3 == min.peek()? **evet** → min'den de pop; yeni en küçük 3.

Adım 11 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
▶    int x = y.pop();
▶    if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

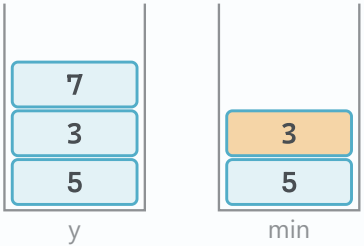
DEĞİŞKENLER

`y = [5, 3, 7]` `min = [5, 3]` `x = 3`

ÇAĞRI YIĞINI

- `pop()`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın




```
push(5)
push(3)
push(7)
push(3)
getMin()  → 3
```

```
pop()     → 3
getMin()  → 3
pop()
pop()
getMin()
```

min'in üstü = o anki en küçük

getMin(): `min.peek()` = **3**. Arama yok → O(1).

Adım 12 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
▶ int getMin() { return min.peek(); }
```

DEĞİŞKENLER

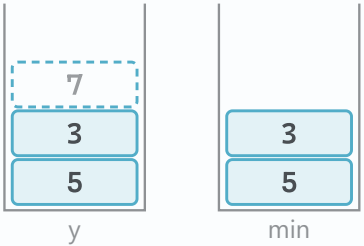
`y = [5, 3, 7]` `min = [5, 3]`

ÇAĞRI YIĞINI

`getMin()`

`main()`

Kod İzleme: `getMin()` O(1) Olan Yığın




```
push(5)          pop()      → 3
push(3)          getMin()   → 3
push(7)          pop()      → 7
push(3)          pop()
getMin()  → 3    getMin()
```

min'in üstü = o anki en küçük

pop(): y'den **7** çıktı. 7 == min.peek()? hayır → min değişmez.

Adım 13 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
▶    int x = y.pop();
▶    if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

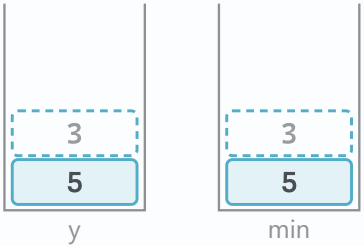
DEĞİŞKENLER

`y = [5, 3]` `min = [5, 3]` `x = 7`

ÇAĞRI YIĞINI

- `pop()`
- `main()`

Kod İzleme: `getMin()` O(1) Olan Yığın




```
push(5)          pop()      → 3
push(3)          getMin()   → 3
push(7)          pop()      → 7
push(3)          pop()      → 3
getMin()  → 3    getMin()
```

min'in üstü = o anki en küçük

pop(): y'den **3** çıktı. 3 == min.peek()? **evet** → min'den de pop; yeni en küçük 5.

Adım 14 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
▶    int x = y.pop();
▶    if (x == min.peek()) min.pop();
11   return x;
12 }
13 int getMin() { return min.peek(); }
```

DEĞİŞKENLER

`y = [5]` `min = [5]` `x = 3`

ÇAĞRI YIĞINI

`pop()`

`main()`

Kod İzleme: `getMin()` O(1) Olan Yığın



push(5)	pop() → 3
push(3)	getMin() → 3
push(7)	pop() → 7
push(3)	pop() → 3
getMin() → 3	getMin() → 5

min'in üstü = o anki en küçük

getMin(): `min.peek()` = **5**. Arama yok → O(1). `<=` sayesinde ikinci 3 çıkınca en küçük hâlâ 3 kalmıştı.

Adım 15 / 15

```
1  Deque<Integer> y = new ArrayDeque<>();
2  Deque<Integer> min = new ArrayDeque<>();
3  void push(int x) {
4    y.push(x);
5    if (min.isEmpty() || x <= min.peek())
6      min.push(x);
7  }
8  int pop() {
9    int x = y.pop();
10   if (x == min.peek()) min.pop();
11   return x;
12 }
▶ int getMin() { return min.peek(); }
```

DEĞİŞKENLER

`y = [5]` `min = [5]`

ÇAĞRI YIĞINI

`getMin()`

`main()`

Hanoi Kuleleri: Yığın ile Özyinelemesiz Çözüm

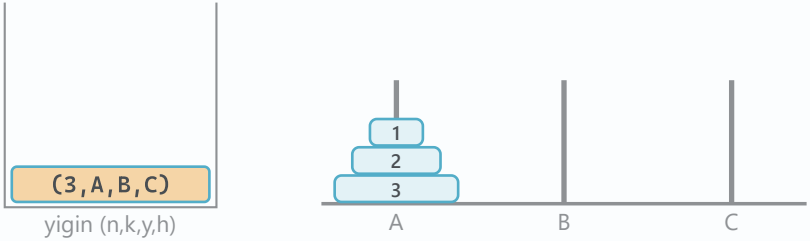
Kural: her hamlede en üstteki tek disk taşınır; büyük disk küçüğün üstüne konamaz.

```
// Gorev: n diski k(aynak)'tan h(edef)'e, y(ardımcı) ile taşı
record Gorev(int n, char k, char y, char h) {}

void hanoi(int n) {
  Deque<Gorev> yigin = new ArrayDeque<>();
  yigin.push(new Gorev(n, 'A', 'B', 'C'));
  while (!yigin.isEmpty()) {
    Gorev g = yigin.pop();
    if (g.n() == 1)
      System.out.println(g.k() + " → " + g.h());
    else {                          // ters sırayla
      yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
      yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
      yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
    }
  }
}
```

Özyinelemeli çözümdeki çağrı yığınının yerini **açık bir görev yığını** alır. n > 1 ise üç alt görev **ters sırayla** eklenir: **1.** n−1 diski kaynaktan yardımcıya, **2.** 1 diski kaynaktan hedefe, **3.** n−1 diski yardımcıdan hedefe; böylece 1. görev en üstte olur.

Kod İzleme: `hanoi(3)` (yığınla)



Başlangıç görevi **(3, A, B, C)**: “3 diski A'dan C'ye, B yardımcı ile taşı”.

Adım 1 / 11

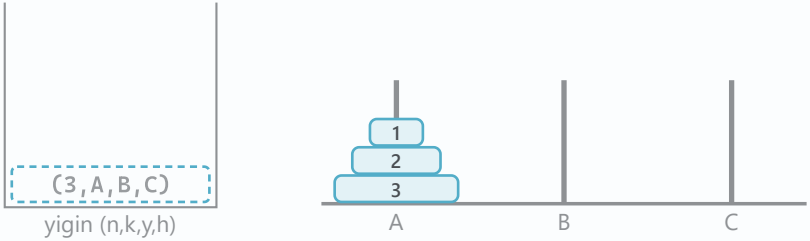
```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
▶    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`yigin.size() = 1`

ÇIKTI

Kod İzleme: `hanoi(3)` (yığınla)



Adım 2 / 11

pop → g = (3,A,B,C): şimdi bu görev yapılacak.

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

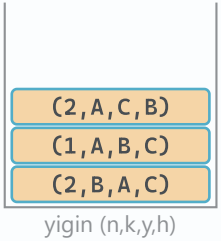
DEĞİŞKENLER

`g = (3,A,B,C)`

`yigin.size() = 0`

ÇIKTI

Kod İzleme: `hanoi(3)` (yığınla)





Adım 3 / 11

n = 3 > 1 → üç alt görev **ters sırayla** push edilir; ilk yapılacak (2, A, C, B) en üstte kalır (LIFO).

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
▶        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
▶        yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
▶        yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

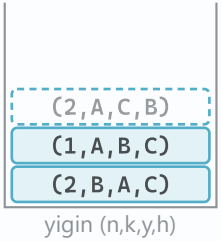
DEĞİŞKENLER

`g = (3,A,B,C)`

`yigin.size() = 3`

ÇIKTI

Kod İzleme: `hanoi(3)` (yığınla)





pop → g = (2,A,C,B): şimdi bu görev yapılacak.

Adım 4 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

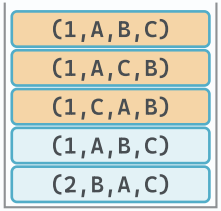
DEĞİŞKENLER

`g = (2,A,C,B)`

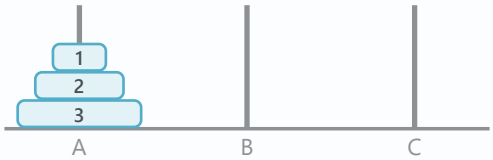
`yigin.size() = 2`

ÇIKTI

Kod İzleme: `hanoi(3)` (yığınla)



yigin (n,k,y,h)



n = 2 > 1 → üç alt görev **ters sırayla** push edilir; ilk yapılacak (1, A, B, C) en üstte kalır (LIFO).

Adım 5 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
▶        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
▶        yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
▶        yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

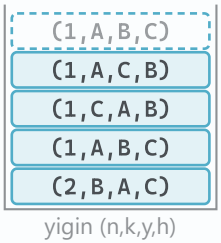
DEĞİŞKENLER

`g = (2,A,C,B)`

`yigin.size() = 5`

ÇIKTI

Kod İzleme: `hanoi(3)` (yığınla)





pop → g = (1,A,B,C): şimdi bu görev yapılacak.

Adım 6 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

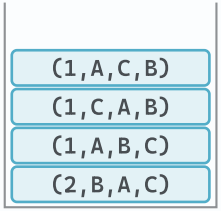
DEĞİŞKENLER

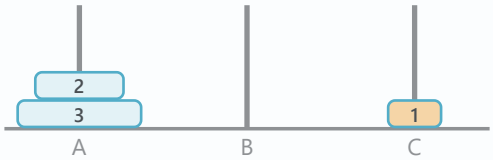
`g = (1,A,B,C)`

`yigin.size() = 4`

ÇIKTI

Kod İzleme: `hanoi(3)` (yığınla)





n = 1 → doğrudan taşı: disk 1, **A** → **C** (hamle 1).

Adım 7 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

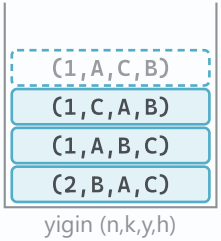
`g = (1,A,B,C)`

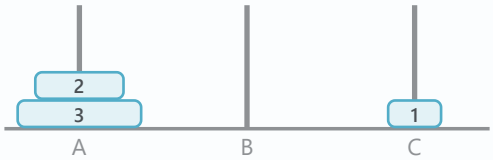
`yigin.size() = 4`

ÇIKTI

```
A → C
```

Kod İzleme: `hanoi(3)` (yığınla)





pop → g = (1,A,C,B): şimdi bu görev yapılacak.

Adım 8 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

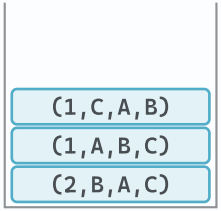
`g = (1,A,C,B)`

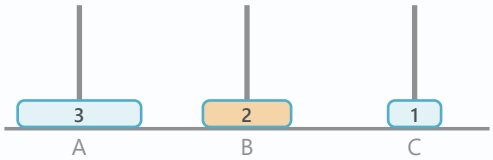
`yigin.size() = 3`

ÇIKTI

```
A → C
```

Kod İzleme: `hanoi(3)` (yığınla)





n = 1 → doğrudan taşı: disk 2, **A** → **B** (hamle 2).

Adım 9 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

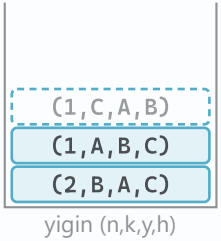
`g = (1,A,C,B)`

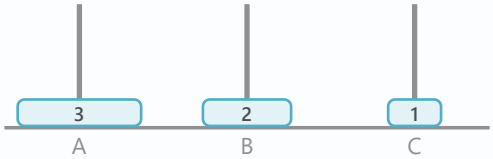
`yigin.size() = 3`

ÇIKTI

```
A → C
A → B
```

Kod İzleme: `hanoi(3)` (yığınla)





pop → g = (1,C,A,B): şimdi bu görev yapılacak.

Adım 10 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

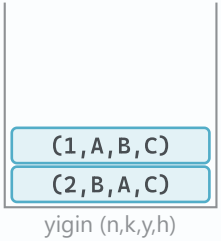
`g = (1,C,A,B)`

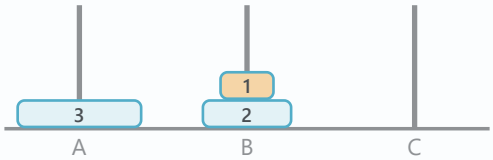
`yigin.size() = 2`

ÇIKTI

```
A → C
A → B
```

Kod İzleme: `hanoi(3)` (yığınla)





Adım 11 / 11

n = 1 → doğrudan taşı: disk 1, **C** → **B** (hamle 3).

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

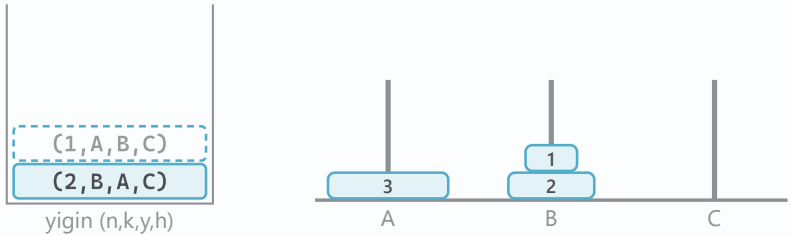
`g = (1,C,A,B)`

`yigin.size() = 2`

ÇIKTI

```
A → C
A → B
C → B
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



Adım 1 / 11

pop → g = (1,A,B,C): şimdi bu görev yapılacak.

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`g = (1,A,B,C)`

`yigin.size() = 1`

ÇIKTI

```
A → C
A → B
C → B
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



Adım 2 / 11

n = 1 → doğrudan taşı: disk 3, **A** → **C** (hamle 4).

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`g = (1,A,B,C)`

`yigin.size() = 1`

ÇIKTI

```
A → C
A → B
C → B
A → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



Adım 3 / 11

pop → g = (2,B,A,C): şimdi bu görev yapılacak.

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`g = (2,B,A,C)`

`yigin.size() = 0`

ÇIKTI

```
A → C
A → B
C → B
A → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



n = 2 > 1 → üç alt görev **ters sırayla** push edilir; ilk yapılacak (1, B, C, A) en üstte kalır (LIFO).

Adım 4 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
▶        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
▶        yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
▶        yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`g = (2,B,A,C)`

`yigin.size() = 3`

ÇIKTI

```
A → C
A → B
C → B
A → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



Adım 5 / 11

pop → g = (1,B,C,A): şimdi bu görev yapılacak.

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

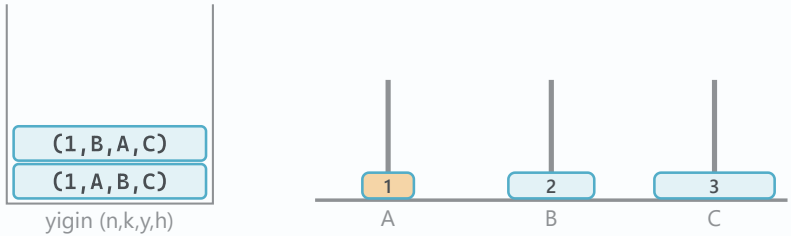
`g = (1,B,C,A)`

`yigin.size() = 2`

ÇIKTI

```
A → C
A → B
C → B
A → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



Adım 6 / 11

n = 1 → doğrudan taşı: disk 1, **B** → **A** (hamle 5).

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

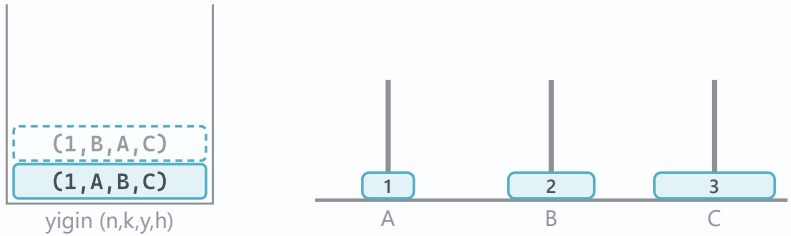
`g = (1,B,C,A)`

`yigin.size() = 2`

ÇIKTI

```
A → C
A → B
C → B
A → C
B → A
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



pop → g = (1,B,A,C): şimdi bu görev yapılacak.

Adım 7 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

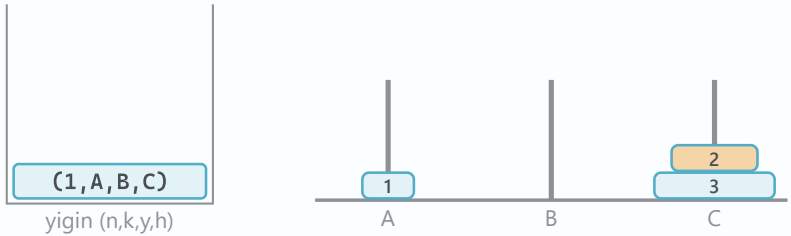
`g = (1,B,A,C)`

`yigin.size() = 1`

ÇIKTI

```
A → C
A → B
C → B
A → C
B → A
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



n = 1 → doğrudan taşı: disk 2, **B** → **C** (hamle 6).

Adım 8 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

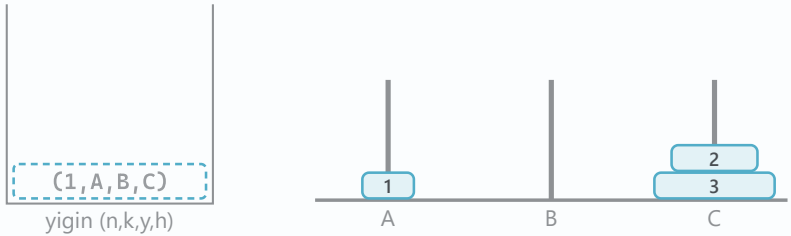
`g = (1,B,A,C)`

`yigin.size() = 1`

ÇIKTI

```
A → C
A → B
C → B
A → C
B → A
B → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



pop → g = (1,A,B,C): şimdi bu görev yapılacak.

Adım 9 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
▶      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`g = (1,A,B,C)`

`yigin.size() = 0`

ÇIKTI

```
A → C
A → B
C → B
A → C
B → A
B → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



n = 1 → doğrudan taşı: disk 1, **A** → **C** (hamle 7).

Adım 10 / 11

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
4    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
▶        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

`g = (1,A,B,C)`

`yigin.size() = 0`

ÇIKTI

```
A → C
A → B
C → B
A → C
B → A
B → C
A → C
```

Kod İzleme: `hanoi(3)` (yığınla, devam)



Adım 11 / 11

Yığın boş → döngü biter. **7 = 2^3 − 1** hamle; tüm diskler C'de.

```
1  void hanoi(int n) {
2    Deque<Gorev> yigin = new ArrayDeque<>();
3    yigin.push(new Gorev(n, 'A', 'B', 'C'));
▶    while (!yigin.isEmpty()) {
5      Gorev g = yigin.pop();
6      if (g.n() == 1)
7        System.out.println(g.k() + " → " + g.h());
8      else {                          // ters sırayla
9        yigin.push(new Gorev(g.n()-1, g.y(), g.k(), g.h()));
10       yigin.push(new Gorev(1, g.k(), g.y(), g.h()));
11       yigin.push(new Gorev(g.n()-1, g.k(), g.h(), g.y()));
12     }
13   }
14 }
```

DEĞİŞKENLER

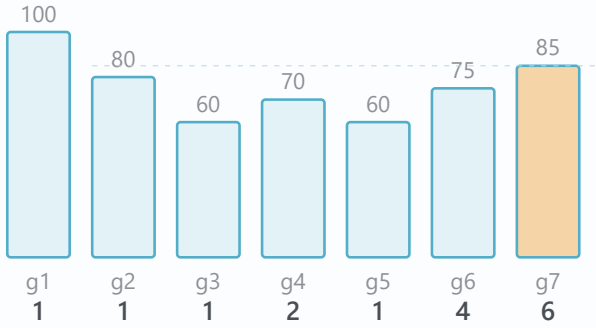
`yigin.size() = 0`

ÇIKTI

```
A → C
A → B
C → B
A → C
B → A
B → C
A → C
```

Stock Span (Hisse Senedi Sıçrama) Problemi

- Günlük kapanış fiyatları bir dizide verilir.
- Bir günün **span** değeri: o gün dahil, geriye doğru fiyatı bugünkü fiyattan **küçük ya da eşit** olan ardışık günlerin sayısı.
- Finansal veri analizinde bir fiyatın "kaç gündür aşılmadığını" gösterir.



Gün	Fiyat	Span	Neden?
1	100	1	ilk gün
4	70	2	60 ≤ 70, 80 > 70
6	75	4	60, 70, 60 ≤ 75; 80 > 75
7	85	6	75, 60, 70, 60, 80 ≤ 85

Fiyatlar [100, 80, 60, 70, 60, 75, 85] → span [1, 1, 1, 2, 1, 4, 6]. Çubukların altındaki kalın sayılar span değerleridir.

STOCK SPAN

Stock Span: Yığın ile O(n) Çözüm

```
int[] span(int[] fiyatlar) {
    int[] sicrama = new int[fiyatlar.length];
    Deque<Integer> yigin = new ArrayDeque<>();
    yigin.push(0); sicrama[0] = 1;
    for (int i = 1; i < fiyatlar.length; i++) {
        while (!yigin.isEmpty() &&
            fiyatlar[i] >= fiyatlar[yigin.peek()])
            yigin.pop();
        sicrama[i] = yigin.isEmpty() ? i + 1
                                : i - yigin.peek();
        yigin.push(i);
    }
    return sicrama;
}
```

Yığında ne var?

Henüz aşılmamış günlerin **indisleri** (fiyatlar alttan üste azalır).

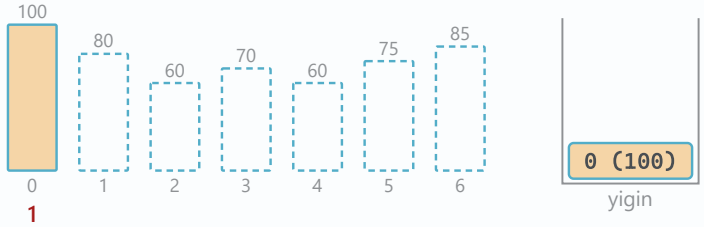
Span

Üst = önceki ilk **daha yüksek** gün: span = i − üst; yığın boşsa i + 1.

Karmaşıklık

Her indis bir kez eklenir, en çok bir kez çıkar → **O(n)**.

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



İlk gün: indis 0 yığına, **sicrama[0] = 1** (öncesinde gün yok).

Adım 1 / 19

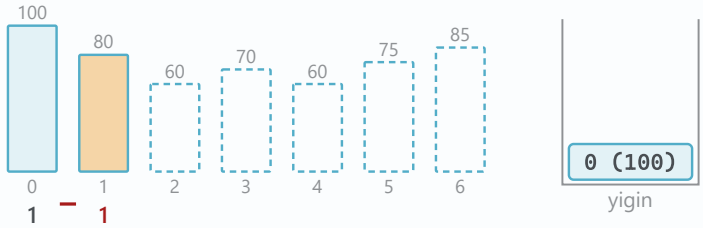
```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
▶      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

DEĞİŞKENLER

`sicrama = [1, 0, 0, 0, 0, 0, 0]`

`yigin = [0]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 1 (fiyat 80): üstteki 0. gün (100) daha yüksek → span = 1 − 0 = **1**.

Adım 2 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
▶          sicrama[i] = yigin.isEmpty() ? i + 1
▶                                       : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

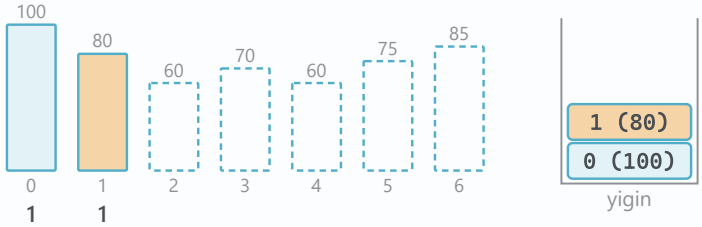
DEĞİŞKENLER

`i = 1` `fiyatlar[i] = 80`

`sicrama = [1, 1, 0, 0, 0, 0, 0]`

`yigin = [0]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



push(1): sonraki günler için olası sınır.

Adım 3 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
▶          yigin.push(i);
12     }
13     return sicrama;
14 }
```

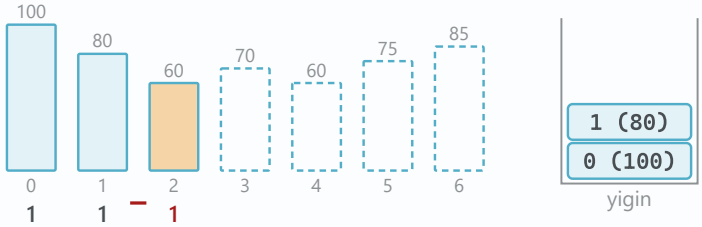
DEĞİŞKENLER

`i = 1` `fiyatlar[i] = 80`

`sicrama = [1, 1, 0, 0, 0, 0, 0]`

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 2 (fiyat 60): üstteki 1. gün (80) daha yüksek → span = 2 − 1 = **1**.

Adım 4 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
▶          sicrama[i] = yigin.isEmpty() ? i + 1
▶                                       : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

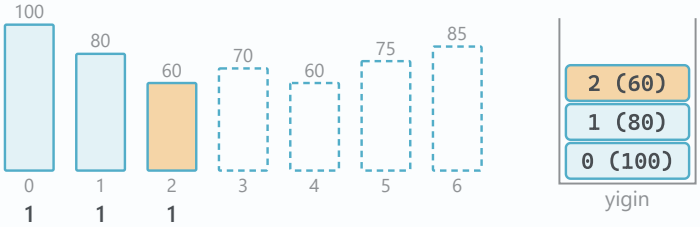
DEĞİŞKENLER

`i = 2` `fiyatlar[i] = 60`

`sicrama = [1, 1, 1, 0, 0, 0, 0]`

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



push(2): sonraki günler için olası sınır.

Adım 5 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
▶          yigin.push(i);
12     }
13     return sicrama;
14 }
```

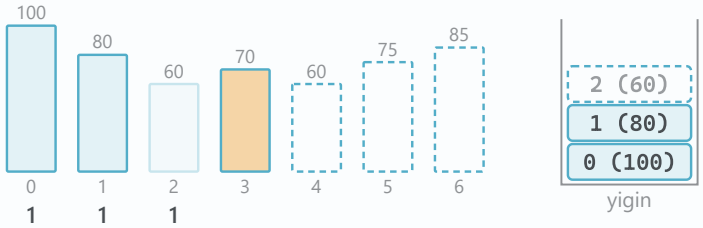
DEĞİŞKENLER

`i = 2` `fiyatlar[i] = 60`

`sicrama = [1, 1, 1, 0, 0, 0, 0]`

`yigin = [0, 1, 2]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 3: 70 ≥ fiyatlar[2] = 60 → **pop**. 2. gün artık kimsenin sınırı olamaz.

Adım 6 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
▶              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

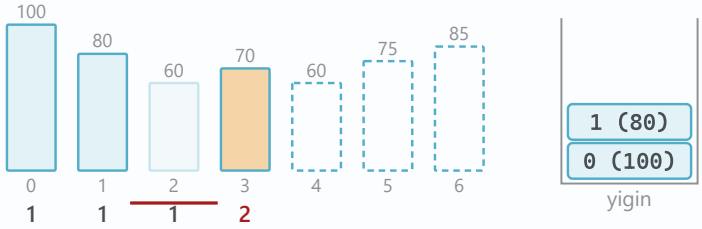
DEĞİŞKENLER

`i = 3` `fiyatlar[i] = 70`

`sicrama = [1, 1, 1, 0, 0, 0, 0]`

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 3 (fiyat 70): üstteki 1. gün (80) daha yüksek → span = 3 − 1 = **2**.

Adım 7 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
▶          sicrama[i] = yigin.isEmpty() ? i + 1
▶                                       : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

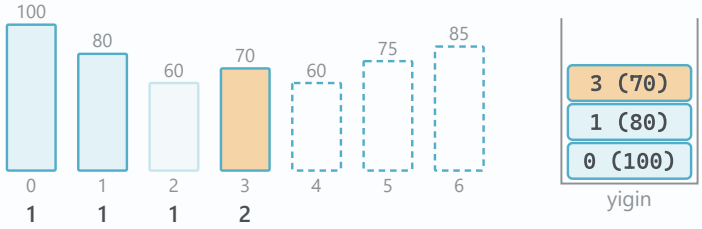
DEĞİŞKENLER

`i = 3` `fiyatlar[i] = 70`

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

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



push(3): sonraki günler için olası sınır.

Adım 8 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
▶          yigin.push(i);
12     }
13     return sicrama;
14 }
```

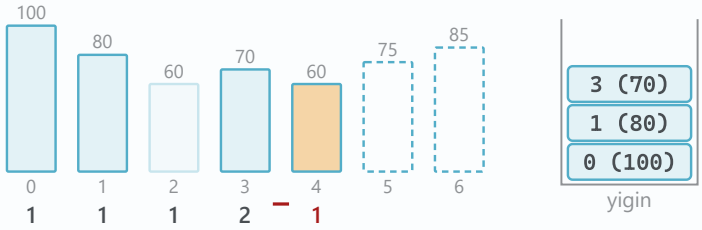
DEĞİŞKENLER

`i = 3` `fiyatlar[i] = 70`

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

`yigin = [0, 1, 3]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 4 (fiyat 60): üstteki 3. gün (70) daha yüksek → span = 4 − 3 = **1**.

Adım 9 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
▶          sicrama[i] = yigin.isEmpty() ? i + 1
▶                                       : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

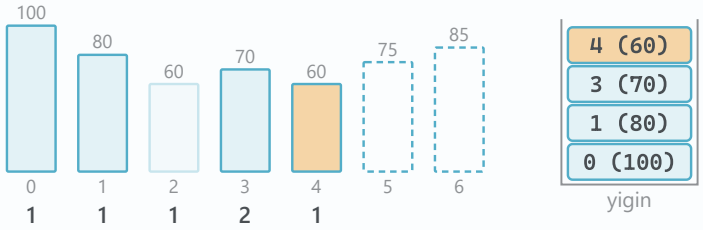
DEĞİŞKENLER

`i = 4` `fiyatlar[i] = 60`

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

`yigin = [0, 1, 3]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



push(4): sonraki günler için olası sınır.

Adım 10 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
▶          yigin.push(i);
12     }
13     return sicrama;
14 }
```

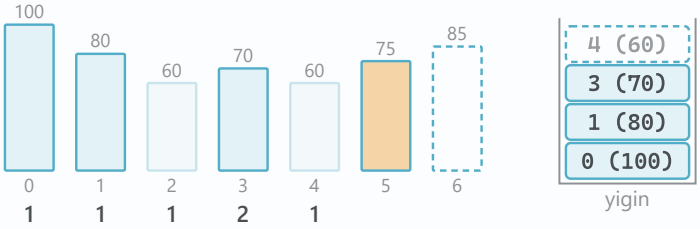
DEĞİŞKENLER

`i = 4` `fiyatlar[i] = 60`

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

`yigin = [0, 1, 3, 4]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 5: 75 ≥ fiyatlar[4] = 60 → **pop**. 4. gün artık kimsenin sınırı olamaz.

Adım 11 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
▶             yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

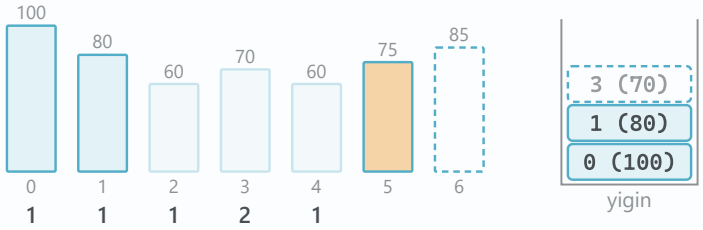
DEĞİŞKENLER

`i = 5` `fiyatlar[i] = 75`

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

`yigin = [0, 1, 3]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 5: 75 ≥ fiyatlar[3] = 70 → **pop**. 3. gün artık kimsenin sınırı olamaz.

Adım 12 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
▶              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

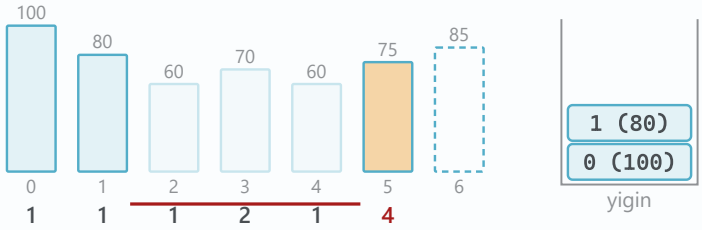
DEĞİŞKENLER

`i = 5` `fiyatlar[i] = 75`

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

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 5 (fiyat 75): üstteki 1. gün (80) daha yüksek → span = 5 − 1 = **4**.

Adım 13 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
▶          sicrama[i] = yigin.isEmpty() ? i + 1
▶                                       : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

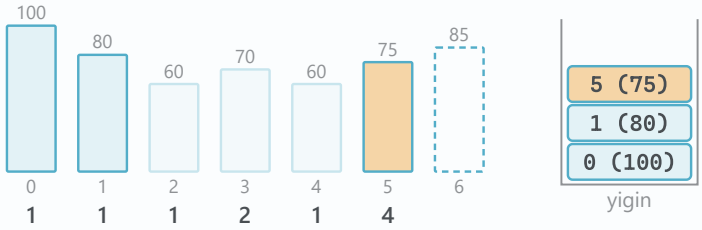
DEĞİŞKENLER

`i = 5` `fiyatlar[i] = 75`

`sicrama = [1, 1, 1, 2, 1, 4, 0]`

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



push(5): sonraki günler için olası sınır.

Adım 14 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
▶          yigin.push(i);
12     }
13     return sicrama;
14 }
```

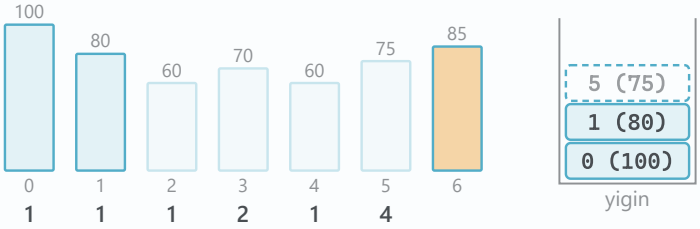
DEĞİŞKENLER

`i = 5` `fiyatlar[i] = 75`

`sicrama = [1, 1, 1, 2, 1, 4, 0]`

`yigin = [0, 1, 5]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 6: 85 ≥ fiyatlar[5] = 75 → **pop**. 5. gün artık kimsenin sınırı olamaz.

Adım 15 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
▶             yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

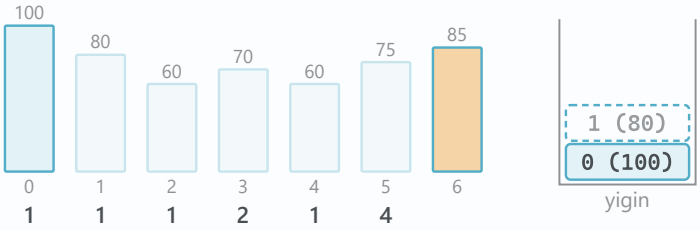
DEĞİŞKENLER

`i = 6` `fiyatlar[i] = 85`

`sicrama = [1, 1, 1, 2, 1, 4, 0]`

`yigin = [0, 1]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 6: 85 ≥ fiyatlar[1] = 80 → **pop**. 1. gün artık kimsenin sınırı olamaz.

Adım 16 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
▶             yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

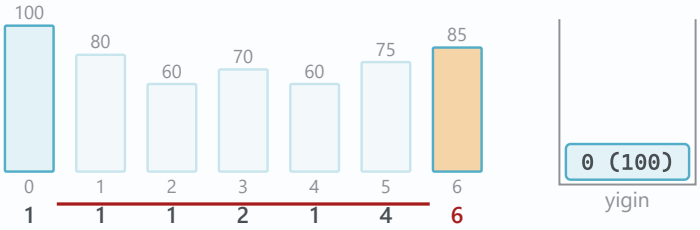
DEĞİŞKENLER

`i = 6` `fiyatlar[i] = 85`

`sicrama = [1, 1, 1, 2, 1, 4, 0]`

`yigin = [0]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



i = 6 (fiyat 85): üstteki 0. gün (100) daha yüksek → span = 6 − 0 = **6**.

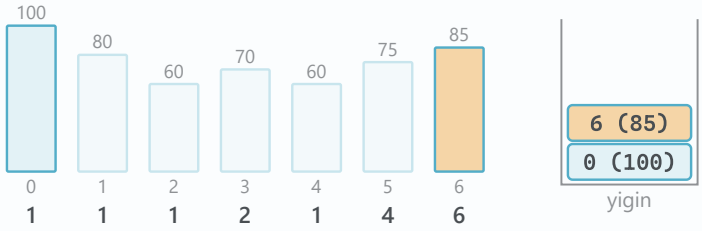
Adım 17 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
▶          sicrama[i] = yigin.isEmpty() ? i + 1
▶                                       : i - yigin.peek();
11         yigin.push(i);
12     }
13     return sicrama;
14 }
```

DEĞİŞKENLER

- `i = 6`
- `fiyatlar[i] = 85`
- `sicrama = [1, 1, 1, 2, 1, 4, 6]`
- `yigin = [0]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



push(6): sonraki günler için olası sınır.

Adım 18 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
▶          yigin.push(i);
12     }
13     return sicrama;
14 }
```

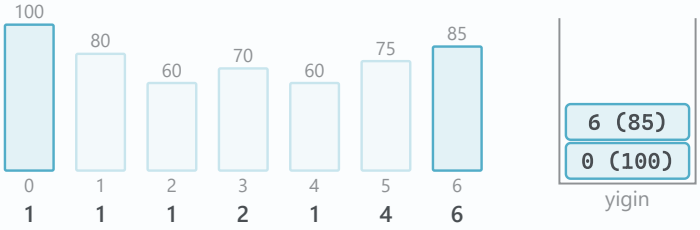
DEĞİŞKENLER

`i = 6` `fiyatlar[i] = 85`

`sicrama = [1, 1, 1, 2, 1, 4, 6]`

`yigin = [0, 6]`

Kod İzleme: `span({100, 80, 60, 70, 60, 75, 85})`



return [1, 1, 1, 2, 1, 4, 6]. Her indis bir kez eklenir, en çok bir kez çıkar → **O(n)**.

Adım 19 / 19

```
1  int[] span(int[] fiyatlar) {
2      int[] sicrama = new int[fiyatlar.length];
3      Deque<Integer> yigin = new ArrayDeque<>();
4      yigin.push(0); sicrama[0] = 1;
5      for (int i = 1; i < fiyatlar.length; i++) {
6          while (!yigin.isEmpty() &&
7              fiyatlar[i] >= fiyatlar[yigin.peek()])
8              yigin.pop();
9          sicrama[i] = yigin.isEmpty() ? i + 1
10                                      : i - yigin.peek();
11         yigin.push(i);
12     }
▶      return sicrama;
14 }
```

DEĞİŞKENLER

`sicrama = [1, 1, 1, 2, 1, 4, 6]`

`yigin = [0, 6]`

İFADE GÖSTERİMLERİ

Infix, Prefix ve Postfix Gösterimleri

Gösterim	İşlecin yeri	A * (B + C) / D
Ara ek (infix)	işlenenlerin arasında	`A * ( B + C ) / D`
Önek (prefix) · Polonya gösterimi	işlenenlerden önce	`/ * A + B C D`
Sonek (postfix) · ters Polonya (RPN)	işlenenlerden sonra	`A B C + * D /`

- Prefix ve postfixte **parantez ve öncelik kuralı gerekmez**: ifade soldan sağa (ya da sağdan sola) tek geçişte, yığınla değerlendirilir.
- Derleyiciler ve yorumlayıcılar ifadeleri çoğunlukla postfix biçimine (ör. yığın tabanlı bayt koduna) çevirir; hesap makineleri de RPN kullanabilir.
- Sembolik cebir ve ifade ağaçları üzerinde çalışırken de kullanılır.

Dikkat: prefix `* A / + B C D` yanlıştır; bu `A * ((B + C) / D)` demektir. Soldan sağa işlem sırasına uygun doğru prefix: **`/ * A + B C D`**.

İFADE GÖSTERİMLERİ

Öncelik, Birleşme ve Elle Dönüşüm

İşleç	Öncelik	Birleşme
`^` (üs)	3	sağdan sola
`* /`	2	soldan sağa
`+ -`	1	soldan sağa

Aynı öncelikte soldan sağa: `A - B + C` = `(A - B) + C` → postfix `A B - C +`.

Elle dönüşüm: tam parantezleme

1. Her işlemi parantezle: `((A * (B + C)) / D)`
2. **Postfix**: her işleci kendi parantezinin **sağına** taşı → `A B C + * D /`
3. **Prefix**: her işleci kendi parantezinin **soluna** taşı → `/ * A + B C D`

İşlenenlerin (A, B, C, D) sırası üç gösterimde de **aynı kalır**; yalnızca işleçlerin yeri değişir. Bu, dönüşümü kontrol etmenin hızlı bir yoludur.

İFADE GÖSTERİMLERİ

Infix → Postfix: Algoritma

Girdi soldan sağa okunur; işleçler ve açık parantezler bir yığında bekletilir.

Okunan simge	Yapılacak işlem
İşlenen (A, B, 5 …)	Doğrudan çıktıya yaz.
`(`	Yığına koy.
`)`	`(` görülene kadar yığından çıkarıp çıktıya yaz; `(`'ı at.
İşleç op	Üstte `(` olmadığı ve üstteki işlecin önceliği **≥** op olduğu sürece çıkarıp çıktıya yaz; sonra op'u yığına koy.
Girdi bitti	Yığında kalan tüm işleçleri çıkarıp çıktıya yaz.

"≥" kuralı soldan sağa birleşmeyi sağlar. Sağdan birleşen `^` için yalnızca önceliği **>** olanlar çıkarılır. Zaman O(n).

Infix → Postfix: Java

```java
String infixToPostfix(String ifade) {
  StringBuilder cikti = new StringBuilder();
  Deque<Character> yigin = new ArrayDeque<>();
  for (char c : ifade.toCharArray()) {
    if (Character.isLetterOrDigit(c)) cikti.append(c);
    else if (c == '(') yigin.push(c);
    else if (c == ')') {
      while (yigin.peek() != '(')
        cikti.append(yigin.pop());
      yigin.pop();                    // '(' atılır
    } else {                          // işleç
      while (!yigin.isEmpty() &&
          oncelik(yigin.peek()) >= oncelik(c))
        cikti.append(yigin.pop());
      yigin.push(c);
    }
  }
  while (!yigin.isEmpty()) cikti.append(yigin.pop());
  return cikti.toString();
}
```

```java
int oncelik(char op) {
  switch (op) {
    case '+': case '-': return 1;
    case '*': case '/': return 2;
    default: return 0;   // '('
  }
}
```

Varsayımlar

Tek karakterli işlenenler, yalnızca + − * / (^ yok), boşluksuz ve geçerli ifade.

Neden (→ 0?

İşleç geldiğinde `(` hiçbir zaman çıkarılmaz; yalnızca `)` onu atar.

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D

cikti:



Girdi soldan sağa okunacak. İşlenenler çıktıya, işleçler ve '(' yığına gider.

Adım 1 / 13

```
▶ for (char c : ifade.toCharArray()) {
2   if (Character.isLetterOrDigit(c)) cikti.append(c);
3   else if (c == '(') yigin.push(c);
4   else if (c == ')') {
5     while (yigin.peek() != '(')
6       cikti.append(yigin.pop());
7     yigin.pop();                  // '(' atılır
8   } else {                        // işleç
9     while (!yigin.isEmpty() &&
10        oncelik(yigin.peek()) >= oncelik(c))
11      cikti.append(yigin.pop());
12    yigin.push(c);
13  }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = ""` `yigin = []`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
▲ c								

cikti: **A**



A işlenen → doğrudan çıktıya.

Adım 2 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'A'` `cikti = "A"` `yigin = []`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
	▲ c							

cikti: **A**



*: yığın boş → **push** *.

Adım 3 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = '*'` `cikti = "A"` `yigin = [*]`

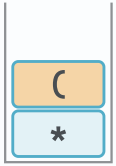
Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
		▲ c						

cikti: **A**

yigin



(→ her zaman yığına. Kapanana kadar içindeki işleçleri korur.

Adım 4 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
▶    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = '('` `cikti = "A"` `yigin = [*, (]`

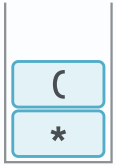
Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
			▲ c					

cikti: **A B**

yigin



B işlenen → doğrudan çıktıya.

Adım 5 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'B'` `cikti = "AB"`

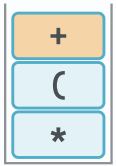
`yigin = [*, (]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
				▲ c				

cikti: **A B**



yigin

+: üstte '(' var → **push +**.

Adım 6 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = '+'` `cikti = "AB"` `yigin = [*, (, +]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
					▲ c			

cikti: **A B C**

yigin

+
(
*

C işlenen → doğrudan çıktıya.

Adım 7 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'C'` `cikti = "ABC"`

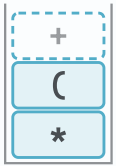
`yigin = [*, (, +]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
						▲ c		

cikti: **A B C +**



yigin

): '(' görülene kadar pop → **+** çıktıya.

Adım 8 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
▶      while (yigin.peek() != '(')
▶        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = ')'` `cikti = "ABC+"`

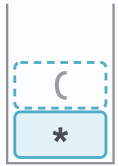
`yigin = [*, (]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
						▲ c		

cikti: **A B C +**



yigin

'(' yığından atılır; parantezler çıktıya **yazılmaz**.

Adım 9 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
▶      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = ')'` `cikti = "ABC+"`

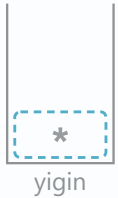
`yigin = [*]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
							▲ c	

cikti: **A B C + ***



/: üstteki * önceliği (2) = / önceliği (2) → * çıktıya.

Adım 10 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
▶      while (!yigin.isEmpty() &&
▶          oncelik(yigin.peek()) >= oncelik(c))
▶        cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

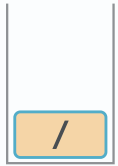
`c = '/'` `cikti = "ABC+*"` `yigin = []`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
							▲ c	

cikti: **A B C + ***



yigin

/: yığın boş → **push /**.

Adım 11 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = '/'` `cikti = "ABC+*"`

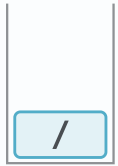
`yigin = [/]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D
								▲ c

cikti: **A B C + * D**



yigin

D işlenen → doğrudan çıktıya.

Adım 12 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'D'` `cikti = "ABC+*D"`

`yigin = [/]`

Kod İzleme: `infixToPostfix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D

cikti: **A B C + * D /**



Girdi bitti: kalan işleç **/** çıktıya.
Sonuç: **A B C + * D /**

Adım 13 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
▶ while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = "ABC+*D/"` `yigin = []`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E

cikti:



Girdi soldan sağa okunacak. İşlenenler çıktıya, işleçler ve '(' yığına gider.

Adım 1 / 14

```
▶ for (char c : ifade.toCharArray()) {
2   if (Character.isLetterOrDigit(c)) cikti.append(c);
3   else if (c == '(') yigin.push(c);
4   else if (c == ')') {
5     while (yigin.peek() != '(')
6       cikti.append(yigin.pop());
7     yigin.pop();                  // '(' atılır
8   } else {                        // işleç
9     while (!yigin.isEmpty() &&
10        oncelik(yigin.peek()) >= oncelik(c))
11      cikti.append(yigin.pop());
12    yigin.push(c);
13  }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = ""` `yigin = []`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E

▲ c

cikti: **A**



A işlenen → doğrudan çıktıya.

Adım 2 / 14

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

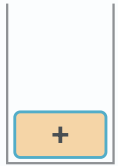
`c = 'A'` `cikti = "A"` `yigin = []`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
	▲ c							

cikti: **A**



yigin

+: yığın boş → **push** +.

Adım 3 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶      yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

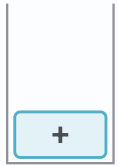
`c = '+'` `cikti = "A"` `yigin = [+]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
		▲ c						

cikti: A B



yigin

B işlenen → doğrudan çıktıya.

Adım 4 / 14

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

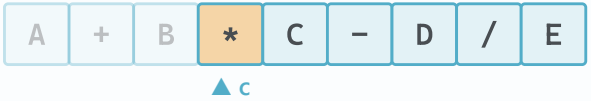
DEĞİŞKENLER

`c = 'B'` `cikti = "AB"`

`yigin = [+]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade



cikti: **A B**



yigin

*: üstteki + önceliği (1) < 2 → **push** *.

Adım 5 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

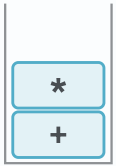
`c = '*'` `cikti = "AB"` `yigin = [+, *]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
				▲ c				

cikti: **A B C**



yigin

C işlenen → doğrudan çıktıya.

Adım 6 / 14

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'C'` `cikti = "ABC"`

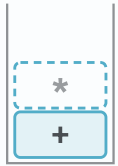
`yigin = [+, *]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	−	D	/	E
					▲ c			

cikti: **A B C ***



yigin

-: üstteki * önceliği (2) > - önceliği (1) → * çıktıya.

Adım 7 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
▶      while (!yigin.isEmpty() &&
▶          oncelik(yigin.peek()) >= oncelik(c))
▶        cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = '-'` `cikti = "ABC*"` `yigin = [+]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

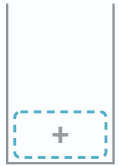
ifade

0	1	2	3	4	5	6	7	8
A	+	B	*	C	−	D	/	E

▲ c

çıktı: **A B C * +**

yigin



-: üstteki + önceliği (1) = - önceliği (1) → + çıktıya.

Adım 8 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
▶      while (!yigin.isEmpty() &&
▶          oncelik(yigin.peek()) >= oncelik(c))
▶        cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

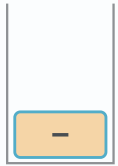
`c = '-'` `cikti = "ABC*+"` `yigin = []`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
					▲ c			

cikti: **A B C * +**



-: yığın boş → **push -**.

Adım 9 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶      yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = '-'` `cikti = "ABC*+"`

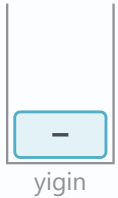
`yigin = [-]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
						▲ c		

cikti: **A B C * + D**



yigin

D işlenen → doğrudan çıktıya.

Adım 10 / 14

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'D'` `cikti = "ABC*+D"`

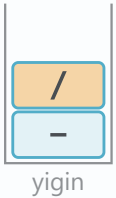
`yigin = [-]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
							▲ c	

cikti: **A B C * + D**



yigin

/: üstteki - önceliği (1) < 2 → **push /**.

Adım 11 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

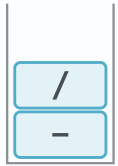
`c = '/'` `cikti = "ABC*+D"` `yigin = [-, /]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E
								▲ c

cikti: **A B C * + D E**



yigin

E işlenen → doğrudan çıktıya.

Adım 12 / 14

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'E'` `cikti = "ABC*+DE"`

`yigin = [-, /]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E

cikti: **A B C * + D E /**



Girdi bitti: kalan işleç **/** çıktıya.

Adım 13 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
▶ while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

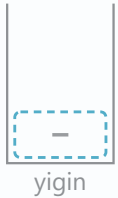
`cikti = "ABC*+DE/"` `yigin = [-]`

Kod İzleme: `infixToPostfix("A+B*C-D/E")`

ifade

A	+	B	*	C	-	D	/	E

cikti: **A B C * + D E / -**



yigin

Girdi bitti: kalan işleç **-** çıktıya.
Sonuç: **A B C * + D E / −** = (A + B·C) − D/E

Adım 14 / 14

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) >= oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
▶ while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = "ABC*+DE/-"` `yigin = []`

Postfix İfadeyi Değerlendirme

- Soldan sağa oku.
- **Sayı** → yığına push.
- **İşleç** → iki öğe pop: önce **sağ**, sonra **sol** işlenen; sonucu push.
- Sonda yığında tek öğe kalır: sonuç. Parantez ve öncelik yok → O(n).

```
int hesapla(int sol, char op, int sag) {
  switch (op) {
    case '+': return sol + sag;
    case '-': return sol - sag;
    case '*': return sol * sag;
    default:  return sol / sag;
  }
}
```

```
int postfixDegerlendir(String ifade) {
  Deque<Integer> yigin = new ArrayDeque<>();
  for (char c : ifade.toCharArray()) {
    if (Character.isDigit(c)) {
      yigin.push(c - '0');
    } else {
      int sag = yigin.pop();
      int sol = yigin.pop();
      yigin.push(hesapla(sol, c, sag));
    }
  }
  return yigin.pop();
}
```

Varsayım: tek basamaklı sayılar, boşluksuz ve geçerli ifade.

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade

4	1	2	+	*	3	/



A=4, B=1, C=2, D=3 için **A B C + * D /** → "412+*3/". Boş işlenen yığını oluşturulur.

Adım 1 / 12

```
1  int postfixDegerlendir(String ifade) {
▶    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

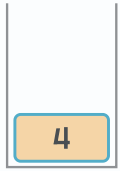
`yigin = []`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade

4	1	2	+	*	3	/

▲ c



4 rakam → `c - '0'` = 4 yığına.

Adım 2 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
▶      if (Character.isDigit(c)) {
▶        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '4'` `yigin = [4]`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade

4	1	2	+	*	3	/

▲ c

yigin

1
4

1 rakam → `c - '0'` = 1 yığına.

Adım 3 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
▶      if (Character.isDigit(c)) {
▶        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '1'` `yigin = [4, 1]`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade





2 rakam → `c - '0'` = 2 yığına.

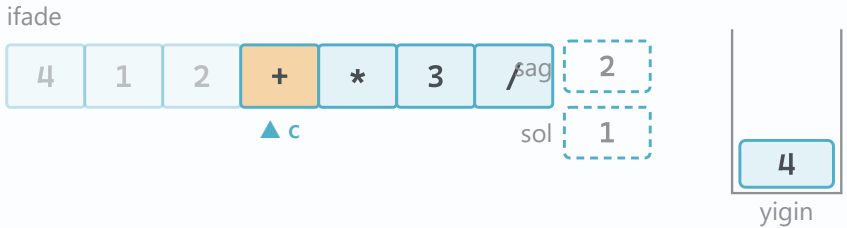
Adım 4 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
▶      if (Character.isDigit(c)) {
▶        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '2'` `yigin = [4, 1, 2]`

Kod İzleme: `postfixDegerlendir("412+*3/")`



+ işleç: önce **sag = 2**, sonra **sol = 1** çıkar (sıra önemli: − ve / için).

Adım 5 / 12

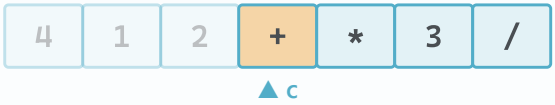
```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
▶        int sag = yigin.pop();
▶        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '+'` `sag = 2` `sol = 1` `yigin = [4]`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade



1 + 2 = 3



hesapla(1, '+', 2) = **3** → sonuç yığına geri konur.

Adım 6 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
▶        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '+'` `sag = 2` `sol = 1`

`yigin = [4, 3]`

Kod İzleme: `postfixDegerlendir("412+*3/")`



* işleç: önce **sag = 3**, sonra **sol = 4** çıkar (sıra önemli: − ve / için).

Adım 7 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
▶        int sag = yigin.pop();
▶        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '*'` `sag = 3` `sol = 4` `yigin = []`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade

4	1	2	+	*	3	/

▲ c

4 * 3 = 12

yigin

12

hesapla(4, '*', 3) = **12** → sonuç yığına geri konur.

Adım 8 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
▶        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '*'` `sag = 3` `sol = 4` `yigin = [12]`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade

4	1	2	+	*	3	/
					▲ c	



3 rakam → `c - '0'` = 3 yığına.

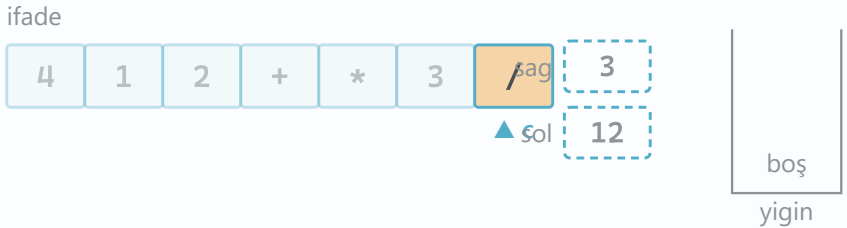
Adım 9 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
▶      if (Character.isDigit(c)) {
▶        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '3'` `yigin = [12, 3]`

Kod İzleme: `postfixDegerlendir("412+*3/")`



/ işleç: önce **sag = 3**, sonra **sol = 12** çıkar (sıra önemli: − ve / için).

Adım 10 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
▶        int sag = yigin.pop();
▶        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '/'` `sag = 3` `sol = 12` `yigin = []`

Kod İzleme: `postfixDegerlendir("412+*3/")`

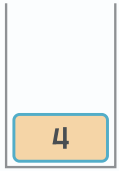
ifade

4	1	2	+	*	3	/

▲ c

12 / 3 = 4

yigin



hesapla(12, '/', 3) = **4** → sonuç yığına geri konur.

Adım 11 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
▶        yigin.push(hesapla(sol, c, sag));
10     }
11   }
12   return yigin.pop();
13 }
```

DEĞİŞKENLER

`c = '/'` `sag = 3` `sol = 12` `yigin = [4]`

Kod İzleme: `postfixDegerlendir("412+*3/")`

ifade

4	1	2	+	*	3	/

sonuç = 4



İfade bitti; yığındaki tek değer sonuçtur: **return 4**. Parantez ve öncelik gerekmedi → O(n).

Adım 12 / 12

```
1  int postfixDegerlendir(String ifade) {
2    Deque<Integer> yigin = new ArrayDeque<>
   ();
3    for (char c : ifade.toCharArray()) {
4      if (Character.isDigit(c)) {
5        yigin.push(c - '0');
6      } else {
7        int sag = yigin.pop();
8        int sol = yigin.pop();
9        yigin.push(hesapla(sol, c, sag));
10     }
11   }
▶    return yigin.pop();
13 }
```

DEĞİŞKENLER

`yigin = []`

Infix → Prefix Dönüşümü

1. Infix ifadeyi **ters çevir**.
2. Parantezleri **değiştir**: `(` ↔ `)`.
3. Postfix algoritmasını uygula; ama eşit öncelikte **çıkarma**, yalnızca önceliği `>` olanları çıkar.
4. Çıktıyı **ters çevir**.

Adım	A * (B + C) / D
1–2. Ters çevir, parantezleri değiştir	`D / ( C + B ) * A`
3. Postfix (eşitte çıkarmadan)	`D C B + A * /`
4. Ters çevir → **prefix**	`/ * A + B C D`

3. adımda `*` geldiğinde üstte eşit öncelikli `/` vardır; `/` çıkarılmaz. Böylece ters çevirince soldan birleşme korunur: önce `A * (B + C)`, sonra `/ D`.

Örnek

`A + B * C - D / E` → `- + A * B C / D E`

Prefix değerlendirme

Postfix gibi, ama ifade **sağdan sola** okunur; ilk çıkan **sol** işlenendir.

Kod İzleme: `infixToPrefix("A*(B+C)/D")`

ifade

A	*	(	B	+	C	)	/	D

main, `infixToPrefix("A*(B+C)/D")` çağırır.

Adım 1 / 6

```
1  String ters(String s) {
2    return new StringBuilder(s).reverse().toString();
3  }
4  String infixToPrefix(String ifade) {
5    String t = ters(ifade);          // 1. ters çevir
6    t = parantezCevir(t);            // 2. ( ↔ )
7    String p = prefixIcinPostfix(t); // 3. eşitte pop yok
8    return ters(p);                  // 4. ters çevir
9  }
10 // main içinde:
▶  System.out.println(infixToPrefix("A*(B+C)/D"));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

ÇIKTI

Kod İzleme: `infixToPrefix("A*(B+C)/D")`

ifade	A	*	(	B	+	C	)	/	D
t = ters(ifade)	D	/	)	C	+	B	(	*	A

1. Dizgi **ters çevrilir**. Parantezler de ters döndü: ')' solda, '(' sağda.

Adım 2 / 6

```
1  String ters(String s) {
2    return new StringBuilder(s).reverse().toString();
3  }
4  String infixToPrefix(String ifade) {
▶    String t = ters(ifade);          // 1. ters çevir
6    t = parantezCevir(t);            // 2. ( ↔ )
7    String p = prefixIcinPostfix(t); // 3. eşitte pop yok
8    return ters(p);                  // 4. ters çevir
9  }
10 // main içinde:
11 System.out.println(infixToPrefix("A*(B+C)/D"));
```

DEĞİŞKENLER

`t = "D/)C+B(*A"`

ÇAĞRI YIĞINI

`infixToPrefix("A*(B+C)/D")`

`main()`

ÇIKTI

Kod İzleme: `infixToPrefix("A*(B+C)/D")`

ifade	A	*	(	B	+	C	)	/	D
t = ters(ifade)	D	/	)	C	+	B	(	*	A
t = parantezCevir(t)	D	/	(	C	+	B	)	*	A

2. **(↔)** değiştirilir; artık geçerli bir infix ifade: **D / (C + B) * A**.

Adım 3 / 6

```
1  String ters(String s) {
2    return new StringBuilder(s).reverse().toString();
3  }
4  String infixToPrefix(String ifade) {
5    String t = ters(ifade);          // 1. ters çevir
▶    t = parantezCevir(t);            // 2. ( ↔ )
7    String p = prefixIcinPostfix(t); // 3. eşitte pop yok
8    return ters(p);                  // 4. ters çevir
9  }
10 // main içinde:
11 System.out.println(infixToPrefix("A*(B+C)/D"));
```

DEĞİŞKENLER

`t = "D/(C+B)*A"`

ÇAĞRI YIĞINI

`infixToPrefix("A*(B+C)/D")`

`main()`

ÇIKTI

Kod İzleme: `infixToPrefix("A*(B+C)/D")`

ifade	A	*	(	B	+	C	)	/	D
t = ters(ifade)	D	/	)	C	+	B	(	*	A
t = parantezCevir(t)	D	/	(	C	+	B	)	*	A
p (eşitte pop yok)	D	C	B	+	A	*	/		

3. Postfix algoritması, ama eşit öncelikte **pop yapmadan** → p = **D C B + A * /**.

Adım 4 / 6

```
1  String ters(String s) {
2    return new StringBuilder(s).reverse().toString();
3  }
4  String infixToPrefix(String ifade) {
5    String t = ters(ifade);          // 1. ters çevir
6    t = parantezCevir(t);            // 2. ( ↔ )
▶    String p = prefixIcinPostfix(t); // 3. eşitte pop yok
8    return ters(p);                  // 4. ters çevir
9  }
10 // main içinde:
11 System.out.println(infixToPrefix("A*(B+C)/D"));
```

DEĞİŞKENLER

`t = "D/(C+B)*A"`

`p = "DCB+A*/"`

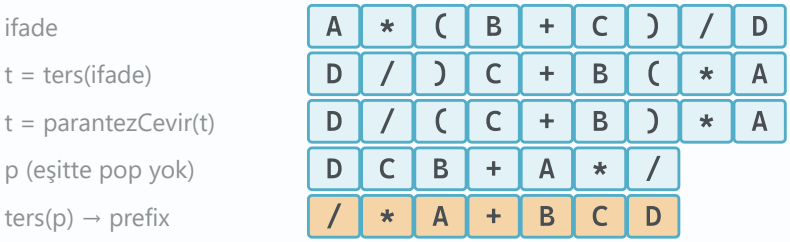
ÇAĞRI YIĞINI

`infixToPrefix("A*(B+C)/D")`

`main()`

ÇIKTI

Kod İzleme: `infixToPrefix("A*(B+C)/D")`



Adım 5 / 6

4. p ters çevrilir → **/ * A + B C D** döner.

```
1  String ters(String s) {
2    return new StringBuilder(s).reverse().toString();
3  }
4  String infixToPrefix(String ifade) {
5    String t = ters(ifade);          // 1. ters çevir
6    t = parantezCevir(t);            // 2. ( ↔ )
7    String p = prefixIcinPostfix(t); // 3. eşitte pop yok
▶    return ters(p);                  // 4. ters çevir
9  }
10 // main içinde:
11 System.out.println(infixToPrefix("A*(B+C)/D"));
```

DEĞİŞKENLER

`t = "D/(C+B)*A"`

`p = "DCB+A*/"`

ÇAĞRI YIĞINI

`infixToPrefix("A*(B+C)/D")`

`main()`

ÇIKTI

Kod İzleme: `infixToPrefix("A*(B+C)/D")`

ifade	A	*	(	B	+	C	)	/	D
t = ters(ifade)	D	/	)	C	+	B	(	*	A
t = parantezCevir(t)	D	/	(	C	+	B	)	*	A
p (eşitte pop yok)	D	C	B	+	A	*	/		
ters(p) → prefix	/	*	A	+	B	C	D		

Adım 6 / 6

main sonucu yazdırır. Kontrol: işlenenler A, B, C, D sırası korunmuş.

```
1  String ters(String s) {
2    return new StringBuilder(s).reverse().toString();
3  }
4  String infixToPrefix(String ifade) {
5    String t = ters(ifade);          // 1. ters çevir
6    t = parantezCevir(t);            // 2. ( ↔ )
7    String p = prefixIcinPostfix(t); // 3. eşitte pop yok
8    return ters(p);                  // 4. ters çevir
9  }
10 // main içinde:
▶  System.out.println(infixToPrefix("A*(B+C)/D"));
```

DEĞİŞKENLER

—

ÇAĞRI YIĞINI

`main()`

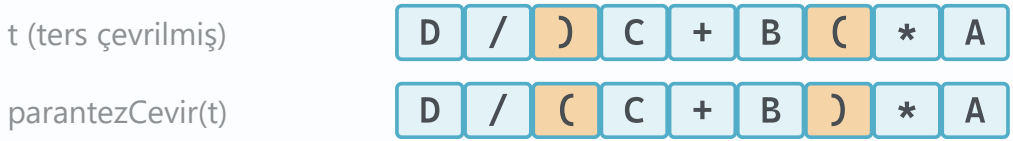
ÇIKTI

```
/*A+BCD
```

2. Adım: `parantezCevir`

Ters çevirme parantezlerin yönünü de bozar. Bu metot her `(` yerine `)`, her `)` yerine `(` yazar; diğer karakterler değişmez.

```
String parantezCevir(String s) {
  StringBuilder sb = new StringBuilder();
  for (char c : s.toCharArray()) {
    if (c == '(') sb.append(')');
    else if (c == ')') sb.append('(');
    else sb.append(c);
  }
  return sb.toString();
}
```



Dizgi bir kez taranır: O(n) süre, O(n) ek bellek.

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A

cikti:



Girdi soldan sağa okunacak. İşlenenler çıktıya, işleçler ve '(' yığına gider.

Adım 1 / 13

```
▶ for (char c : ifade.toCharArray()) {
2   if (Character.isLetterOrDigit(c)) cikti.append(c);
3   else if (c == '(') yigin.push(c);
4   else if (c == ')') {
5     while (yigin.peek() != '(')
6       cikti.append(yigin.pop());
7     yigin.pop();                  // '(' atılır
8   } else {                        // işleç
9     while (!yigin.isEmpty() &&
10        oncelik(yigin.peek()) > oncelik(c))
11      cikti.append(yigin.pop());
12    yigin.push(c);
13  }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = ""` `yigin = []`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A
▲ c								

cikti: **D**



D işlenen → doğrudan çıktıya.

Adım 2 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'D'` `cikti = "D"`

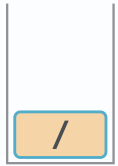
`yigin = []`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A
	▲ c							

cikti: **D**



yigin

/: yığın boş → **push /**.

Adım 3 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

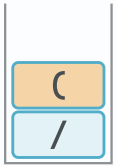
`c = '/'` `cikti = "D"` `yigin = [/]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A
		▲ c						

çıktı: **D**



yigin

(→ her zaman yığına. Kapanana kadar içindeki işleçleri korur.

Adım 4 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
▶    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

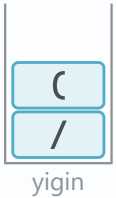
`c = '('` `cikti = "D"` `yigin = [/, (]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)



cikti: **D C**



C işlenen → doğrudan çıktıya.

Adım 5 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

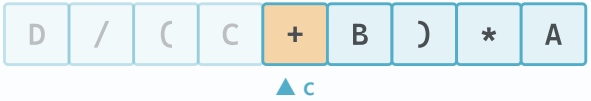
DEĞİŞKENLER

`c = 'C'` `cikti = "DC"`

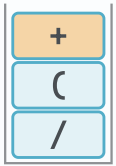
`yigin = [/, (]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)



cikti: **D C**



yigin

+: üstte '(' var → **push +**.

Adım 6 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

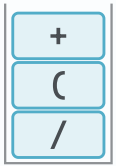
`c = '+'` `cikti = "DC"` `yigin = [/, (, +]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A
			▲ c					

cikti: D C B



yigin

B işlenen → doğrudan çıktıya.

Adım 7 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = 'B'` `cikti = "DCB"`

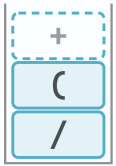
`yigin = [/, (, +]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)



cikti: **D C B +**



yigin

): '(' görülene kadar pop → **+** çıktıya.

Adım 8 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
▶      while (yigin.peek() != '(')
▶        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

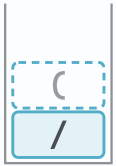
`c = ')'` `cikti = "DCB+"` `yigin = [/, (]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)



cikti: **D C B +**



yigin

'(' yığından atılır; parantezler çıktıya **yazılmaz**.

Adım 9 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
▶      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`c = ')'` `cikti = "DCB+"`

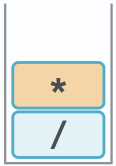
`yigin = [/]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A
			▲ c					

cikti: **D C B +**



yigin

*: üstteki / önceliği (2) = 2, eşitte çıkarmıyoruz → **push** *.

Adım 10 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
▶    yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

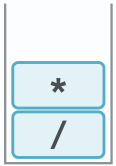
`c = '*'` `cikti = "DCB+"` `yigin = [/, *]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A
								▲ c

cikti: **D C B + A**



yigin

A işlenen → doğrudan çıktıya.

Adım 11 / 13

```
1  for (char c : ifade.toCharArray()) {
▶    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
15 while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

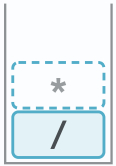
DEĞİŞKENLER

`c = 'A'` `cikti = "DCB+A"` `yigin = [/, *]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A



yigin

Girdi bitti: kalan işleç * çıktıya.

cikti: **D C B + A ***

Adım 12 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
▶ while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = "DCB+A*"` `yigin = [/]`

Kod İzleme: `prefixIcinPostfix("D/(C+B)*A")`

t (ters çevrilmiş, parantezler değişmiş)

D	/	(	C	+	B	)	*	A

cikti: **D C B + A * /**



yigin

Girdi bitti: kalan işleç **/** çıktıya. p = **D C B + A * /** → ters: **/ * A + B C D**

Adım 13 / 13

```
1  for (char c : ifade.toCharArray()) {
2    if (Character.isLetterOrDigit(c)) cikti.append(c);
3    else if (c == '(') yigin.push(c);
4    else if (c == ')') {
5      while (yigin.peek() != '(')
6        cikti.append(yigin.pop());
7      yigin.pop();                    // '(' atılır
8    } else {                          // işleç
9      while (!yigin.isEmpty() &&
10         oncelik(yigin.peek()) > oncelik(c))
11       cikti.append(yigin.pop());
12     yigin.push(c);
13   }
14 }
▶ while (!yigin.isEmpty()) cikti.append(yigin.pop());
```

DEĞİŞKENLER

`cikti = "DCB+A*/"` `yigin = []`

ÖZET

Sık Yapılan Hatalar

Boş yığında pop / peek

Önce `isEmpty()`; yoksa istisna ya da `null` (kütüphaneye göre). Parantez kontrolünde `"())"` tam bu hatayı tetikler.

top ile size karışıklığı

Dizi yığınında `top` son öğenin indisidir, `size = top + 1`. Dolu koşulu `top == kapasite - 1`.

Çıkarılan hücreyi boşaltmamak

Nesne dizisinde `veri[top--] = null` yapılmazsa nesne çöp toplayıcıdan kaçar (bellek sızıntısı, loitering).

Postfix'te işlenen sırası

İlk pop **sağ**, ikinci pop **sol** işlenendir: `5 2 -` = 5 − 2 = 3, 2 − 5 değil.

Yineleme sırası

`ArrayDeque` for-each ile **üstten alta**, eski `Stack` ise **alttan üste** gezinir; çıktılar ters görünür.

Sonsuz özyineleme

Taban durum eksikse çerçeveler birikir ve `StackOverflowError` oluşur (çağrı yığını taşar).

Laboratuvar: Parantez Kontrolü ve Postfix

Parantez kontrolü: {{[]}()}[]

Girdi (yalnız ( ) [ ] { } karakterleri) {{[]}()}[]

KOD · **adım 1 / 38**

```
1 boolean gecerliMi(String s) {
2   Deque<Character> yigin = new ArrayDeq…
3   for (char c : s.toCharArray()) {
4     if (c == '(' || c == '[' || c == '{…
5       yigin.push(c);
6     } else {
7       if (yigin.isEmpty()) return false;
8       char acik = yigin.pop();
9       if (!eslesir(acik, c)) return fal…
10    }
11  }
12  return yigin.isEmpty();
13 }
```

s { { [ ] } ( ) } [ ]

yigin boş

yigin.size() = 0

DÖNÜŞ DEĞERİ –

gecerliMi("{{[]}()}[]") çağrılır: 10 karakter soldan sağa taranacak.

ÖZET

Özet: Yığınla Çözülen Problemler

Problem	Yığında tutulan	Zaman
push / pop / top / isEmpty / size	—	O(1)
Dizgi ters çevirme	karakterler	O(n)
Parantez kontrolü	eşleşmemiş açık parantezler	O(n)
Sonraki büyük öğe	sağdaki aday değerler (azalan)	O(n)
Stock span	henüz aşılmamış günlerin indisleri	O(n)
Hanoi kuleleri	alt görevler (n, kaynak, yard., hedef)	$O(2^n)$
Infix → postfix / prefix	işleçler ve `(`	O(n)
Postfix değerlendirme	ara sonuçlar	O(n)
Min yığın (`getMin`)	her an en küçük değer	O(1)
Geri / ileri gezinme	ziyaret edilen sayfalar (iki yığın)	O(1)

Ortak desen: “en son görülen, henüz sonuçlanmamış öğe” ile ilgileniyorsak yığın kullanırız (LIFO).

ÖZET

Etkileşimli Simülatörler

Yığın Simülatörü

push, pop, peek işlemleri; dizi ve bağlı liste gösterimi.

Stock Span Simülatörü

Fiyat dizisi işlenirken yığının nasıl değiştiği.

Infix → Postfix Dönüştürücü

İşleç yığını ve çıktı, simge simge.

Infix → Prefix Dönüştürücü

Ters çevirme yöntemi, adım adım.

ÖZET

Örnek Problemler

1. Boş yığına sırayla push(5), push(3), pop(), push(8), push(1), pop(), pop(), push(4) uygulanıyor. pop()'ların döndürdüğü değerler ve son yığın nedir?
2. `{[()()]}` ve `{[(])}` dizilerinden hangisi geçerlidir? Geçersiz olanda hata hangi karakterde fark edilir?
3. `(A + B) * (C - D) / E` ifadesini postfix ve prefix biçimine çevirin.
4. `5 1 2 + 4 * + 3 -` postfix ifadesinin değeri nedir?
5. Fiyatlar [10, 4, 5, 90, 120, 80] için span dizisini bulun.
6. {2, 5, 3, 1, 4} dizisi için sonraki büyük öğe dizisini bulun.

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. 1, 2, 3, 4 sırayla push ediliyor, araya pop'lar serpiştirilebiliyor. `2 4 3 1` ve `4 1 2 3` çıkış sıraları mümkün mü?
2. Tek dizide iki yığın (biri baştan, biri sondan büyür): taşma koşulu nedir?
3. Tek bir yardımcı yığınla bir yığını sıralayın. Karmaşıklık?
4. Bir kelimenin palindrom olduğunu yığınla kontrol edin.
5. `<b><i>…</i></b>` gibi HTML etiketlerinin iç içeliği nasıl denetlenir?
6. Dizi dolunca 2 katı yerine **+10** büyütülseydi n push'un toplam maliyeti?

▶ **İpuçları / cevaplar**

Kaynaklar

- Goodrich, Tamassia ve Goldwasser, *Data Structures and Algorithms in Java*, 6. baskı, 2014.
- Lafore, *Data Structures and Algorithms in Java*, 2. baskı, 2002.
- Oracle, Java SE 17 `Deque` ve `ArrayDeque` API belgeleri.

Son

Bölüm 7: Yığınlar

Sonraki sunum: Bölüm 8: Kuyruklar ve Deque · Çalışma soruları · Simülatörler: Yığın Simülatörü, Stock Span Simülatörü, Infix → Prefix Dönüştürücü, Infix → Postfix Dönüştürücü · Terim sözlüğü