

Bölüm 8: Rastgele Algoritmalar

Algoritmalar

Rastgelelik ve beklenen süre

Las Vegas ve Monte Carlo

Fisher–Yates karıştırma

BogoSort

Rastgele QuickSort

Monte Carlo ile π

Miller–Rabin

Karger min-kesim

GİRİŞ

Öğrenme Çıktıları

Kavram

Rastgele algoritmanın tanımını, beklenen çalışma süresini ve Las Vegas / Monte Carlo ayrımını açıklamak.

Karıştırma ve BogoSort

Fisher–Yates karıştırmasını yazmak, saf karıştırmanın neden yanlı olduğunu göstermek; BogoSort'un beklenen $\Theta(n \cdot n!)$ süresini hesaplamak.

Rastgele QuickSort

Rastgele pivotlu parçalamayı adım adım izlemek; beklenen $O(n \log n)$ ve en kötü $O(n^2)$ süreyi gerekçelendirmek.

Monte Carlo

π tahmini, Miller–Rabin asalılık testi ve Karger min-kesim algoritmasında hata olasılığını hesaplamak ve tekrarla küçültmek.

GİRİŞ

Şanslı Algoritmalar

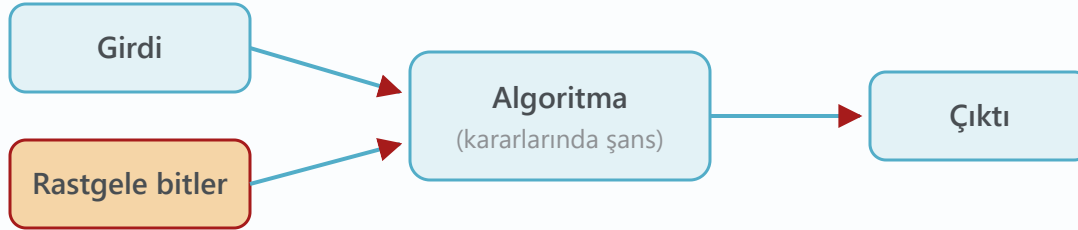
- Bazen problem çözmede **şans oyunlarına** benzer teknikler kullanılır.
- Algoritma, karar verme sürecinde **rastgelelikten** (şans) yararlanır: yazı-tura atar, zar atar, rastgele bir öge seçer.
- Örnek oyun: iki zar atılır; toplam **7 ya da 11** ise oyuncu kazanır.



kazanma: 7 ya da 11

$P(7) = 6/36$, $P(11) = 2/36 \rightarrow P(\text{kazanma}) = 8/36 = 2/9 \approx 0,22$. Kazanana kadar beklenen atış sayısı = $1 / (2/9) = 4,5$.

Rastgele Algoritmalar



- **Rastgele algoritma** (randomized algorithm): girdiye ek olarak **rastgele sayılar** kullanarak çalışır.
- Aynı girdiyle iki çalıştırma **farklı adımlar** izleyebilir: çalışma süresi (ve bazen çıktı) her çalıştırmada değişebilir.
- Süre artık bir **rastgele değişkendir: beklenen süre** $E[T(n)]$ incelenir. Beklenen değer algoritmanın *kendi* yazı-turaları üzerinden alınır, girdi üzerinden değil.

GİRİŞ

Neden Rastgelelik?

Kötü girdiye karşı koruma

Deterministik QuickSort sıralı dizide $O(n^2)$ olur. Rastgele pivotta **hiçbir girdi** sürekli kötü değildir: kötü durum yalnızca şanssız yazı-turlarla oluşur.

Basitlik

Rastgele algoritma çoğu zaman en iyi deterministik algoritmadan çok daha kısa ve anlaşılırdır (Karger min-kesim, rastgele QuickSelect).

Hız

Miller–Rabin, yüzlerce basamaklı sayıların asallığını milisaniyeler içinde, çok küçük bir hata olasılığıyla belirler.

Örnekleme ve simetri kırma

Monte Carlo benzetimleri, rastgele örneklem ve dağıtık sistemlerde lider seçimi gibi işler rastgelelik olmadan zordur.

Bedel: süre ya da doğruluk artık **olasılıksal** bir garantidir; iyi bir rastgele sayı üretici gerekir.

GİRİŞ

Olasılık Hatırlatması

Kavram	Tanım / kural	Örnek
Beklenen değer	$E[X] = \sum x \cdot P(X = x)$	bir zarın beklenen değeri 3,5
Doğrusallık	$E[X + Y] = E[X] + E[Y]$ (bağımsızlık gerekmez)	iki zarın toplamı: 7
Gösterge değişkeni	$X_A = 1$ (A olursa), $0 \rightarrow E[X_A] = P(A)$	QuickSort'ta "i ile j karşılaştırıldı mı?"
Geometrik dağılım	başarı olasılığı p olan denemeler: ilk başarıya kadar beklenen deneme $1/p$	BogoSort: $p = 1/n! \rightarrow n!$ karıştırma
Bağımsız tekrar	her biri p ile başarısız k bağımsız deneme: hepsi başarısız p^k	Miller–Rabin: $(1/4)^k$
Markov eşitsizliği	$X \geq 0$ ise $P(X \geq c \cdot E[X]) \leq 1/c$	Las Vegas $\rightarrow$ Monte Carlo

Las Vegas ve Monte Carlo

	Las Vegas	Monte Carlo
Doğruluk	Her zaman doğru sonuç	Küçük bir olasılıkla yanlış sonuç
Süre	Rastgele; beklenen süre analiz edilir	Sınırlı (deterministik üst sınır)
Rastgele olan	ne kadar süreceği	sonucun doğru olup olmadığı
Örnekler	Rastgele QuickSort, QuickSelect, BogoSort, Fisher–Yates	Miller–Rabin, Karger min-kesim, π tahmini

Tek yönlü hata

Miller–Rabin "bileşik" derse **kesin** doğrudur; yalnızca "asal" yanıtı yanlış olabilir.

İki yönlü hata

π tahmini gibi sayısal benzetimler her iki yönde de sapabilir; hata örnek sayısıyla küçülür.

Hata Olasılığını Küçültme

Tekrarla güçlendirme (amplification)

Tek turda yanılma olasılığı en çok q ise, k **bağımsız** tur sonra yanılma $\leq q^k$.

- Miller–Rabin: $q \leq 1/4 \rightarrow k = 20$ turda $\leq 4^{-20} \approx 9 \cdot 10^{-13}$.
- Karger: başarı $\geq 2/(n(n-1))$; $n(n-1)/2 \cdot \ln n$ tekrarla başarısızlık $\leq 1/n$.

Türler arası dönüşüm

- **Las Vegas** $\rightarrow$ **Monte Carlo**: algoritmayı $2 \cdot E[T]$ adım sonra durdur. Markov: $P(T \geq 2 E[T]) \leq 1/2 \rightarrow$ en çok $1/2$ olasılıkla "yanıt yok".
- **Monte Carlo** $\rightarrow$ **Las Vegas**: yanıt hızlıca **doğrulanabiliyorsa** (ör. bulunan çarpanlar çarpılır), doğru çıkana kadar tekrarlar. Beklenen tekrar = $1/p$.

KARIŞTIRMA

Örnek: Karıştırma

- Sıralı bir listenin öğelerini **karıştırmak** (shuffle) için rastgelelik kullanılır: kart destesi, rastgele test verisi, şarkı listesi.
- **Doğruluk:** her öğenin her konumda **eşit olasılıkla** bulunması; daha güçlüsü, $n!$ permütasyonun her birinin olasılığı **$1/n!$** olmalı.
- **Çalışma zamanı:** Fisher–Yates her karıştırmada tam $n - 1$ takas yapar: **$\Theta(n)$** . Değişen, sürenin değil *sonucun* kendisidir.
- Java'da hazır:
`Collections.shuffle(liste)` (Fisher–Yates).

n	n! permütasyon
3	6
4	24
10	3 628 800
52	$\approx 8,07 \cdot 10^{67}$

Fisher–Yates (Knuth) Karıştırması

```
► static void karistir(int[] a) {  
2   for (int i = a.length - 1; i > 0; i--) {  
3       int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4       int t = a[i]; a[i] = a[j]; a[j] = t;  
5   }  
6 }
```



Başlangıç: `a = [1, 2, 3, 4, 5]`. Sonra başlayarak her konuma, henüz yerleşmemiş öğelerden biri rastgele seçilir.

Adım 1 / 14

DEĞİŞKENLER

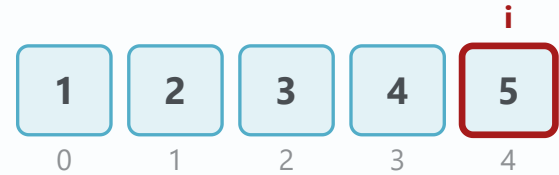
`a = [1, 2, 3, 4, 5]`

`i = -`

`j = -`

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



$i = 4$: $a[0..4]$ arasındaki 5 öğeden biri $a[4]$ konumuna gelecek.

Adım 2 / 14

DEĞİŞKENLER

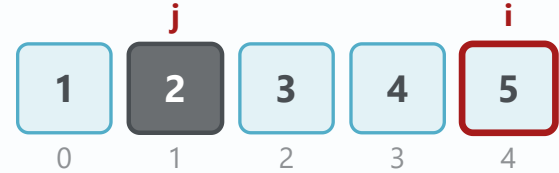
$a = [1, 2, 3, 4, 5]$

$i = 4$

$j = -$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2   for (int i = a.length - 1; i > 0; i--) {  
3     ▶ int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4     int t = a[i]; a[i] = a[j]; a[j] = t;  
5   }  
6 }
```



zar.nextInt(5) → j = 1 (olasılık 1/5).

Adım 3 / 14

DEĞİŞKENLER

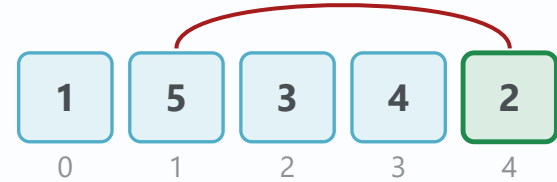
a = [1, 2, 3, 4, 5]

i = 4

j = 1

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



$a[4] \leftrightarrow a[1]$ takas $\rightarrow$ [1, 5, 3, 4, 2]. $a[4] = 2$ artık **kesinleşti** (yeşil).

Adım 4 / 14

DEĞİŞKENLER

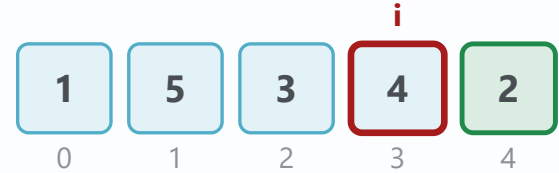
$a = [1, 5, 3, 4, 2]$

$i = 4$

$j = 1$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); // 0 ≤ j ≤ i  
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



$i = 3$: $a[0..3]$ arasındaki 4 öğeden biri $a[3]$ konumuna gelecek.

Adım 5 / 14

DEĞİŞKENLER

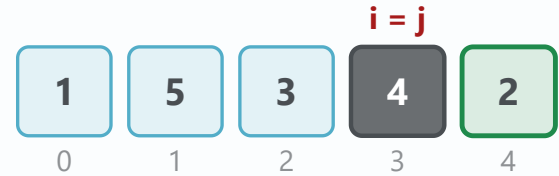
$a = [1, 5, 3, 4, 2]$

$i = 3$

$j = -$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2   for (int i = a.length - 1; i > 0; i--) {  
3     ▶ int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4     int t = a[i]; a[i] = a[j]; a[j] = t;  
5   }  
6 }
```



zar.nextInt(4) → j = 3 (j = i: öge yerinde kalır).

Adım 6 / 14

DEĞİŞKENLER

a = [1, 5, 3, 4, 2]

i = 3

j = 3

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
▶     int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



Kendisiyle takas: dizi değişmez; $a[3] = 4$ kesinleşti.

Adım 7 / 14

DEĞİŞKENLER

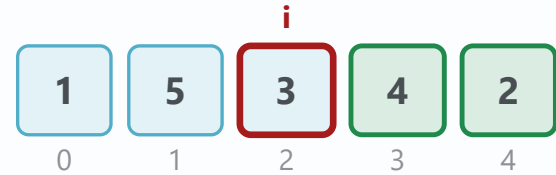
$a = [1, 5, 3, 4, 2]$

$i = 3$

$j = 3$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



$i = 2$: $a[0..2]$ arasındaki 3 öğeden biri $a[2]$ konumuna gelecek.

Adım 8 / 14

DEĞİŞKENLER

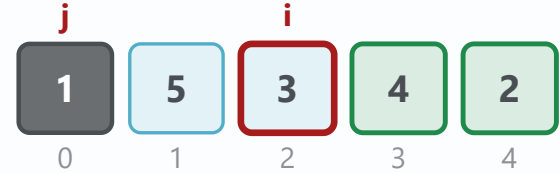
$a = [1, 5, 3, 4, 2]$

$i = 2$

$j = -$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2   for (int i = a.length - 1; i > 0; i--) {  
3     ▶ int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4     int t = a[i]; a[i] = a[j]; a[j] = t;  
5   }  
6 }
```



zar.nextInt(3) → $j = 0$ (olasılık $1/3$).

Adım 9 / 14

DEĞİŞKENLER

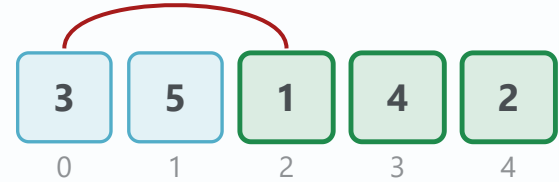
a = [1, 5, 3, 4, 2]

i = 2

j = 0

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2   for (int i = a.length - 1; i > 0; i--) {  
3     int j = zar.nextInt(i + 1); // 0 ≤ j ≤ i  
4     int t = a[i]; a[i] = a[j]; a[j] = t;  
5   }  
6 }
```



$a[2] \leftrightarrow a[0]$ takas $\rightarrow$ [3, 5, 1, 4, 2]. $a[2] = 1$ artık **kesinleşti** (yeşil).

Adım 10 / 14

DEĞİŞKENLER

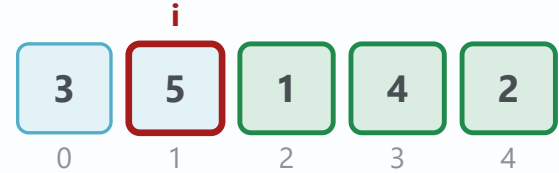
$a = [3, 5, 1, 4, 2]$

$i = 2$

$j = 0$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



$i = 1$: $a[0..1]$ arasındaki 2 öğeden biri $a[1]$ konumuna gelecek.

Adım 11 / 14

DEĞİŞKENLER

$a = [3, 5, 1, 4, 2]$

$i = 1$

$j = -$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2   for (int i = a.length - 1; i > 0; i--) {  
3     ▶ int j = zar.nextInt(i + 1); // 0 ≤ j ≤ i  
4     int t = a[i]; a[i] = a[j]; a[j] = t;  
5   }  
6 }
```



zar.nextInt(2) → j = 1 (j = i: öge
yerinde kalır).

Adım 12 / 14

DEĞİŞKENLER

a = [3, 5, 1, 4, 2]

i = 1

j = 1

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); //  $0 \leq j \leq i$   
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



Kendisiyle takas: dizi değişmez; $a[1] = 5$ kesinleşti.

Adım 13 / 14

DEĞİŞKENLER

$a = [3, 5, 1, 4, 2]$

$i = 1$

$j = 1$

Fisher–Yates (Knuth) Karıştırması

```
1 static void karistir(int[] a) {  
2     for (int i = a.length - 1; i > 0; i--) {  
3         int j = zar.nextInt(i + 1); // 0 ≤ j ≤ i  
4         int t = a[i]; a[i] = a[j]; a[j] = t;  
5     }  
6 }
```



Döngü bitti: [3, 5, 1, 4, 2]. Toplam olası çekiliş sayısı $5 \cdot 4 \cdot 3 \cdot 2 = 120 = 5!$: her permütasyon **tam bir kez** → düzgün dağılım.

Adım 14 / 14

DEĞİŞKENLER

a = [3, 5, 1, 4, 2]

i = 0

j = -

KARIŞTIRMA

Saf Karıştırma Yanlıdır

```
// YANLI: her i için 0..n-1 arasından seç
for (int i = 0; i < n; i++) {
    int j = zar.nextInt(n);
    takas(a, i, j);
}
```

- n çekilişin her biri n değer alır → n^n eşit olasılıklı yol.
- $n = 3$: $3^3 = 27$ yol, $3! = 6$ permütasyon. 27, 6'ya bölünmez → permütasyonlar **eşit olasılıklı olamaz**.
- Sık görülen bu kalıbın doğrusu Fisher–Yates'tir:
`nextInt(i + 1)`.

[1, 2, 3] dizisinin 27 çekiliş dizisi tek tek sayıldı (Fisher–Yates'te $3 \cdot 2 = 6$ yol, her permütasyon 1 kez).

sonuç	yol	olasılık	olması gereken
123	4/27	0,148	$1/6 \approx 0,167$
132	5/27	0,185	$1/6 \approx 0,167$
213	5/27	0,185	$1/6 \approx 0,167$
231	5/27	0,185	$1/6 \approx 0,167$
312	4/27	0,148	$1/6 \approx 0,167$
321	4/27	0,148	$1/6 \approx 0,167$

BogoSort

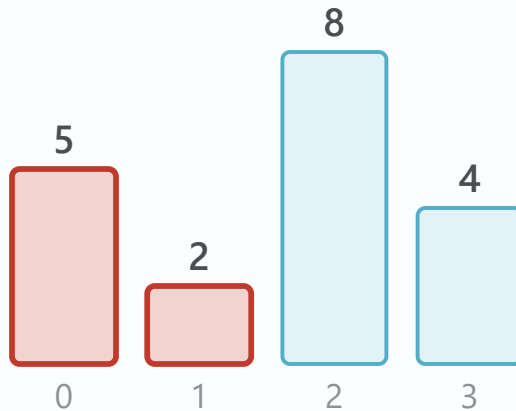
- Sıralama algoritmaları arasında **en kötü performans**a sahip olanlardan biridir ("aptal sıralama", permutation sort).
 - Temel mantık: liste sıralı olana kadar **rastgele karıştır**.
 - Eğlenceli ve öğretici bir **Las Vegas** algoritmasıdır; gerçek dünyada kullanılmaz.
1. Liste sıralı mı? **Evet** → işlemi tamamla.
 2. **Hayır** → listeyi rastgele karıştır, adım 1'e dön.

```
static void bogoSiralama(int[] a) {  
    while (!siraliMi(a))  
        karistir(a);           // Fisher-Yates  
}  
  
static boolean siraliMi(int[] a) {  
    for (int i = 0; i + 1 < a.length; i++)  
        if (a[i] > a[i + 1]) return false;  
    return true;  
}
```

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

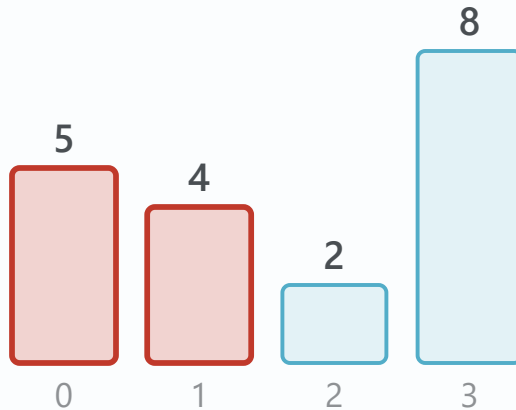
sütun: konum · satır: sıra

Başlangıç **[5, 2, 8, 4]** : $a[0] = 5 > a[1] = 2 \rightarrow$ **sıralı değil**, karıştırmaya başla.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

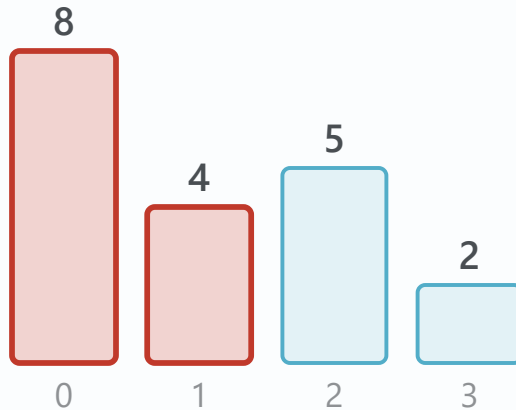
sütun: konum · satır: sıra

Karıştırma 1: [5, 4, 2, 8] → $a[0] = 5 > a[1] = 4$, sıralı değil.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

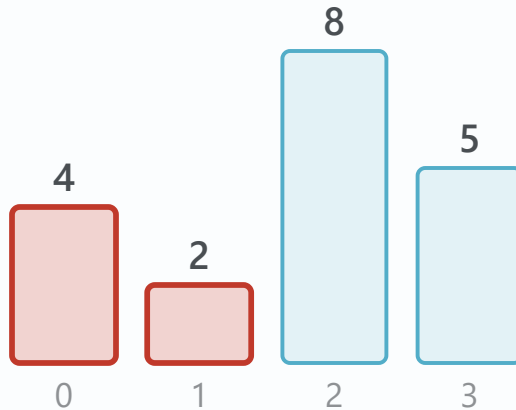
sütun: konum · satır: sıra

Karıştırma 2: **[8, 4, 5, 2]** → $a[0] = 8 > a[1] = 4$, sıralı değil.

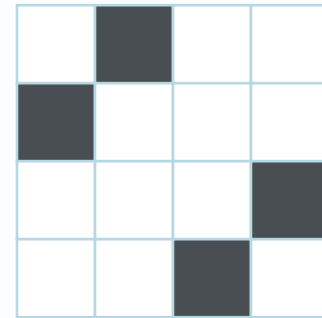
BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi



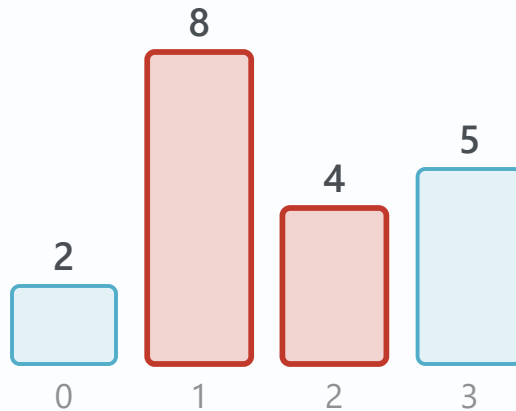
sütun: konum · satır: sıra

Karıştırma 3: **[4, 2, 8, 5]** → $a[0] = 4 > a[1] = 2$, sıralı değil.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

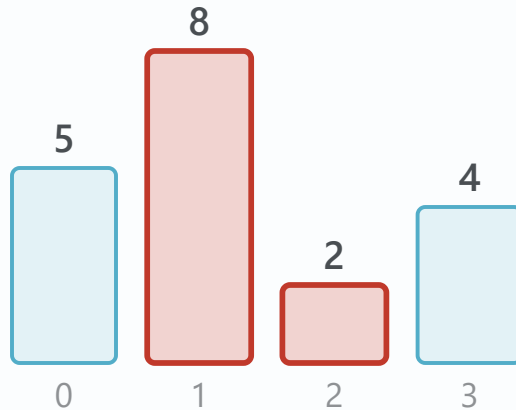
sütun: konum · satır: sıra

Karıştırma 4: `[2, 8, 4, 5]` → $a[1] = 8 > a[2] = 4$, sıralı değil.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

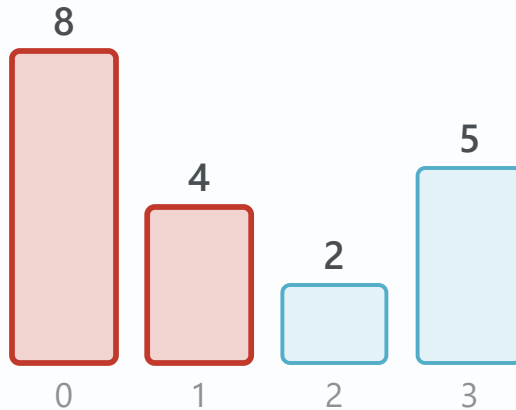
sütun: konum · satır: sıra

Karıştırma 5: `[5, 8, 2, 4]` → $a[1] = 8 > a[2] = 2$, sıralı değil.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

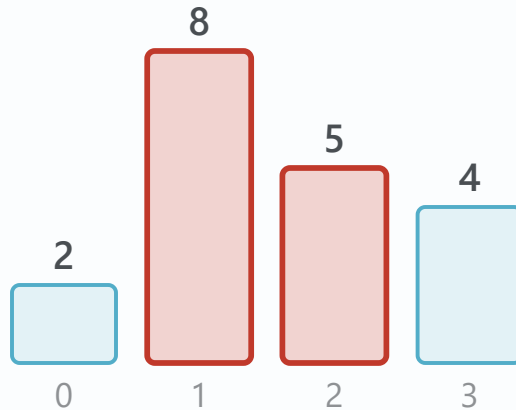
sütun: konum · satır: sıra

Karıştırma 6: `[8, 4, 2, 5]` → $a[0] = 8 > a[1] = 4$, sıralı değil.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



permütasyon matrisi

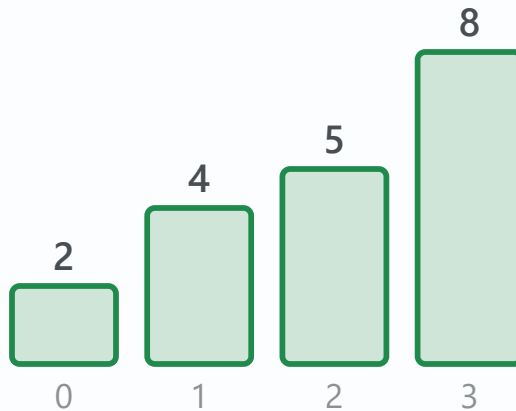
sütun: konum · satır: sıra

Karıştırma 7: `[2, 8, 5, 4]` → $a[1] = 8 > a[2] = 5$, sıralı değil.

BOGOSORT

BogoSort Örneği: [5, 2, 8, 4]

[5, 2, 8, 4] listesi; her karıştırma Fisher–Yates ile (tohumlu bir çalıştırma). Kırmızı: ilk ters sıralı komşu çift.



Karıştırma 8: [2, 4, 5, 8] → **sıralı!** Algoritma durur. Bu çalıştırmada 8 karıştırma gerekti; beklenen değer $4! = 24$.

BogoSort Analizi

- n farklı öge: rastgele bir karıştırmanın sıralı çıkma olasılığı $p = 1/n!$.
- Denemeler bağımsız → geometrik dağılım: beklenen karıştırma sayısı $1/p = n!$.
- Her tur: karıştırma $O(n)$ + kontrol $O(n)$ → **beklenen süre $O(n \cdot n!)$** . Bu, $O((n + 1)!)$ ile aynı büyüklüktedir, çünkü $(n + 1)! = (n + 1) \cdot n!$.
- **En iyi durum:** dizi zaten sıralı → tek kontrol, $O(n)$.
- **En kötü durum: sınırsız.** Şanssız karışırtmalar sonsuza dek sürebilir (olasılığı 0'a gider). $O((n + 1)!)$ yalnızca *beklenen* süre için bir sınırdır, en kötü durum için değil.

n	beklenen karıştırma n!	süre (10^9 karıştırma/s)
5	120	0,12 μ s
10	3 628 800	3,6 ms
12	479 001 600	0,48 s
13	6 227 020 800	6,2 s
15	1 307 674 368 000	$\approx$ 22 dakika
20	2 432 902 008 176 640 000	$\approx$ 77 yıl

QuickSort Algoritması

- **Böl ve fethet** (divide and conquer) ile çalışan, yerinde (in-place) hızlı bir sıralama algoritmasıdır (Hoare, 1962).
- Temel mantık:
 1. Bir **pivot** öge seçilir.
 2. **Parçalama** (partition): pivottan küçükler soluna, büyükler sağına yerleştirilir; pivot **kalıcı yerine** oturur.
 3. Her iki parça için aynı işlem **özyinelemeli** uygulanır.
 4. Birleştirme adımı **gerekmez**: parçalar zaten yerinde sıralanmıştır.

Rastgele QuickSort

Pivot, alt dizinin öğeleri arasından **düzgün rastgele** seçilir. Algoritma her zaman doğru sonuç verir (**Las Vegas**); yalnızca süresi pivot seçimlerine bağlıdır.

Pivot Seçimi ve Etkisi

Pivot stratejisi	Kötü girdi	Beklenen / tipik	En kötü
İlk ya da son öge	sıralı / ters sıralı dizi (çok yaygın!)	$O(n \log n)^*$	$O(n^2)$
Ortanca-üç (median-of-3)	özel hazırlanmış "killer" diziler	$O(n \log n)^*$	$O(n^2)$
Rastgele öge	yok: kötülük yalnızca şanssızlıktan	$O(n \log n)$, her girdi için	$O(n^2)$
Gerçek ortanca (BFPRT)	yok	$O(n \log n)$	$O(n \log n)$

- * Deterministik stratejilerde $O(n \log n)$ yalnızca girdi rastgele sıralıysa geçerlidir (**ortalama durum**). Rastgele pivotta ise beklenti algoritmanın kendi seçimleri üzerindendir (**beklenen süre**).
- İyi pivot seçimi **beklenen** süreyi iyileştirir, en kötü durumu ortadan kaldırmaz: rastgele pivotta da $O(n^2)$ mümkündür, ama olasılığı çok küçüktür.
- BFPRT'nin sabitleri büyüktür; uygulamada rastgele ya da ortanca-üç pivot tercih edilir.

QuickSort Adım Adım: [7, 2, 1, 6, 8, 5, 3, 4]

[7, 2, 1, 6, 8, 5, 3, 4]

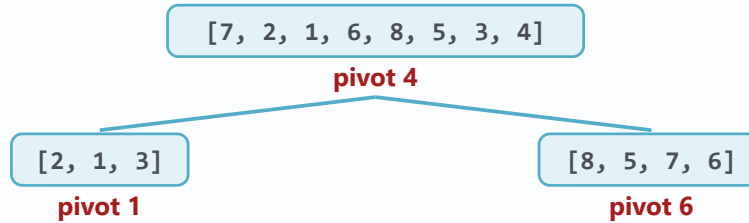
pivot 4

Kök: [7, 2, 1, 6, 8, 5, 3, 4] , pivot **4** seçildi.

Adım 1 / 4

Parça içi sıralar Lomuto parçalamasına göreidir. Pivot, kalıcı yerine oturduğu için parçaların **hiçbirine** dahil edilmez.

QuickSort Adım Adım: [7, 2, 1, 6, 8, 5, 3, 4]

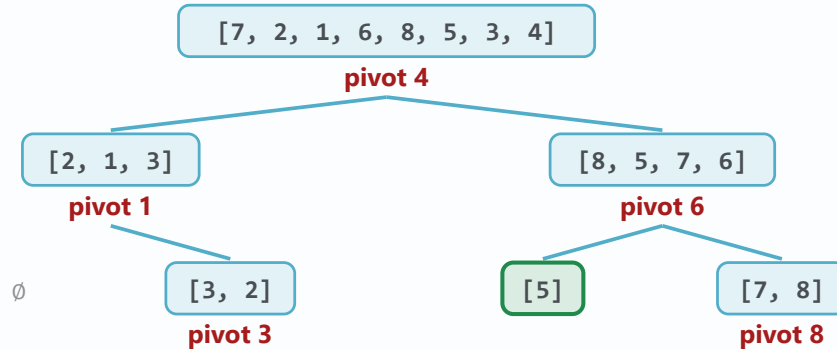


Parçalama sonucu: sol **[2, 1, 3]** (≤ 4), sağ **[8, 5, 7, 6]** (> 4). Pivot 4 iki parçaya da **girmez**, indis 3'te kalıcı yerindedir. Sol parçada pivot 1, sağda 6 seçiliyor.

Adım 2 / 4

Parça içi sıralar Lomuto parçalamasına göre. Pivot, kalıcı yerine oturduğu için parçaların **hiçbirine** dahil edilmez.

QuickSort Adım Adım: [7, 2, 1, 6, 8, 5, 3, 4]

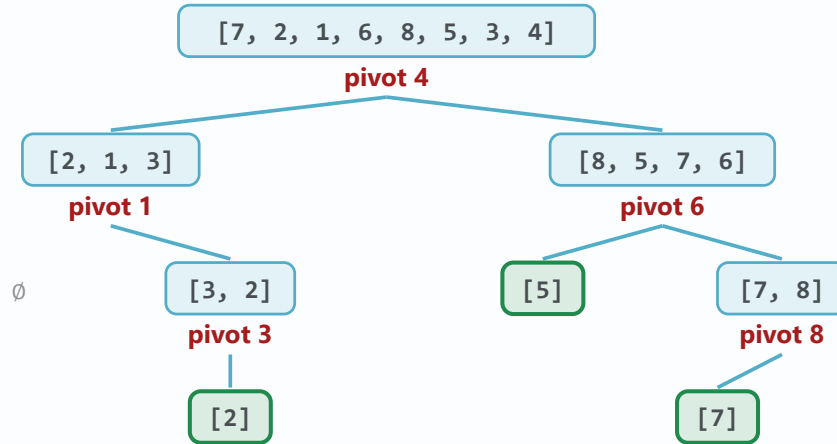


Pivot 1: sol parça **boş** ($\emptyset$), sağ **[3, 2]**. Pivot 6: sol **[5]**, sağ **[7, 8]**. Tek öğeli parçalar zaten sıralıdır (yeşil).

Adım 3 / 4

Parça içi sıralar Lomuto parçalamasına göre sıralıdır. Pivot, kalıcı yerine oturduğu için parçaların **hiçbirine** dahil edilmez.

QuickSort Adım Adım: [7, 2, 1, 6, 8, 5, 3, 4]



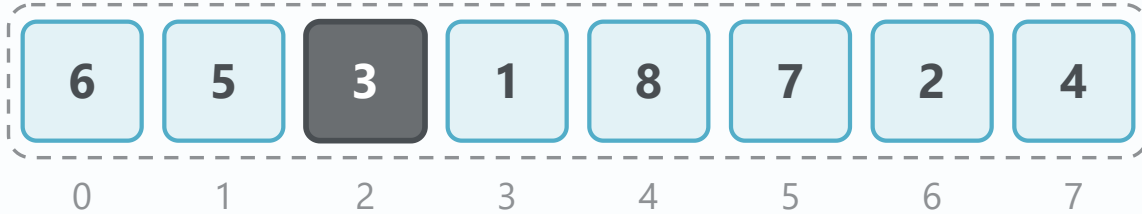
Son düzey: [3, 2] pivot 3 → [2] ; [7, 8] pivot 8 → [7] . Yapraklar birleştirilmeden
yerinde sıralı dizi oluşur: [1, 2, 3, 4, 5, 6, 7, 8] .

Adım 4 / 4

Parça içi sıralar Lomuto parçalamasına göre. Pivot, kalıcı yerine oturduğu için parçaların **hiçbirine** dahil edilmez.

Rastgele Pivotlu QuickSort Çalıştırması

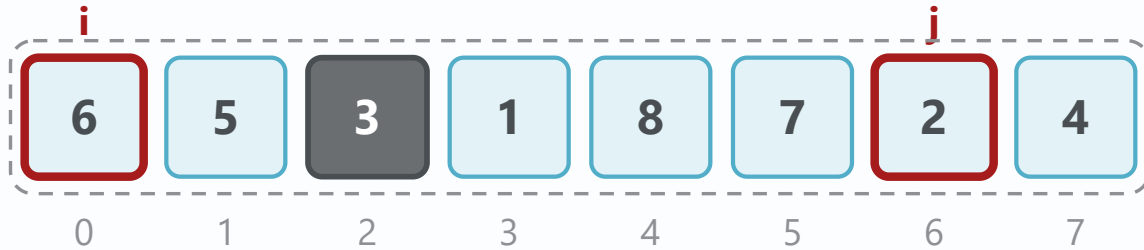
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Alt dizi $a[0..7] = [6, 5, 3, 1, 8, 7, 2, 4]$: pivot **rastgele** seçildi → **3** (indis 2, siyah kutu).

Rastgele Pivotlu QuickSort Çalıştırması

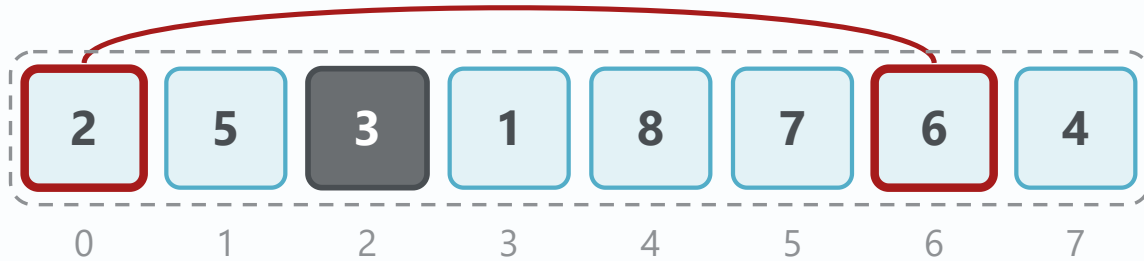
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



i soldan ilerler, 3'ten küçükleri atlar ve $a[0] = 6$ 'da durur; j sağdan ilerler, 3'ten büyükleri atlar ve $a[6] = 2$ 'de durur. $i < j \rightarrow$ takas edilecek.

Rastgele Pivotlu QuickSort Çalıştırması

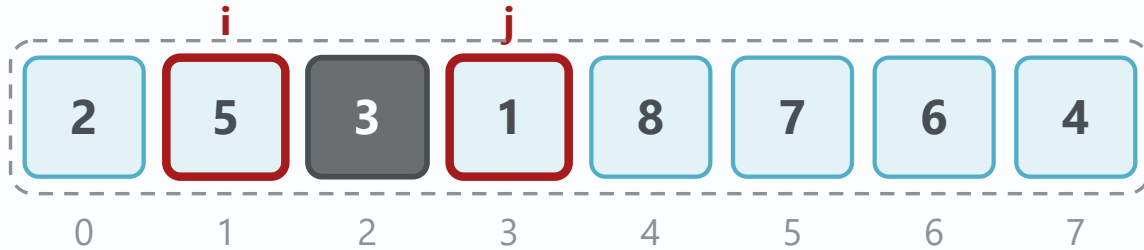
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



$a[0] \leftrightarrow a[6]$ takas $\rightarrow$ [2, 5, 3, 1, 8, 7, 6, 4] .

Rastgele Pivotlu QuickSort Çalıştırması

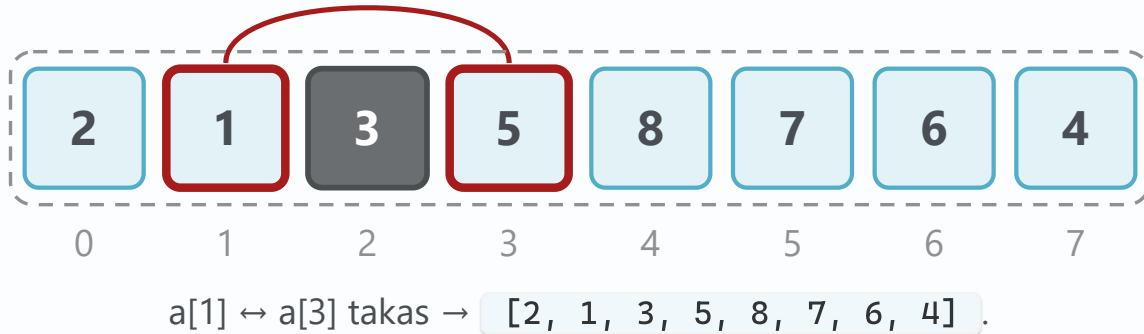
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



i soldan ilerler, 3'ten küçükleri atlar ve $a[1] = 5$ 'te durur; j sağdan ilerler, 3'ten büyükleri atlar ve $a[3] = 1$ 'de durur. $i < j \rightarrow$ takas edilecek.

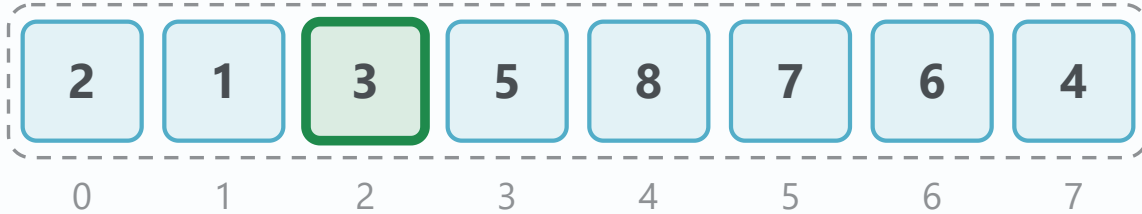
Rastgele Pivotlu QuickSort Çalıştırması

[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Rastgele Pivotlu QuickSort Çalıştırması

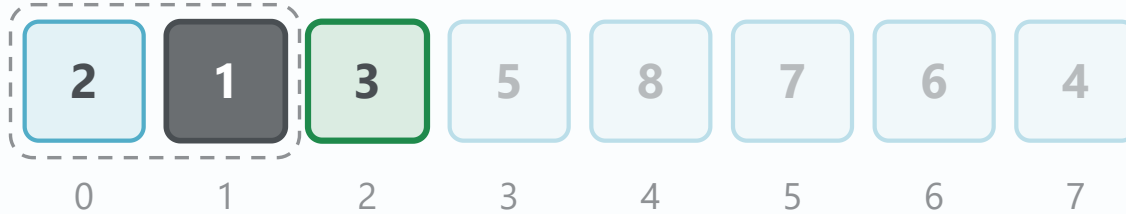
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



İşaretçiler karşılaştı ($i \geq j$): pivot **3** kalıcı yerinde (indis 2). Solunda yalnızca küçükler, sağında yalnızca büyükler var.

Rastgele Pivotlu QuickSort Çalıştırması

[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Alt dizi $a[0..1] = [2, 1]$: pivot **rastgele** seçildi → **1** (indis 1, siyah kutu).

Rastgele Pivotlu QuickSort Çalıştırması

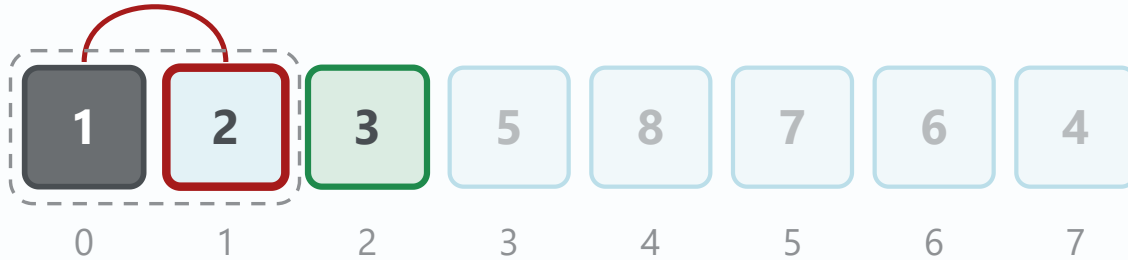
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



i soldan ilerler, 1'den küçükleri atlar ve $a[0] = 2$ 'de durur; j sağdan ilerler, 1'den büyükleri atlar ve $a[1] = 1$ 'de durur. $i < j \rightarrow$ takas edilecek.

Rastgele Pivotlu QuickSort Çalıştırması

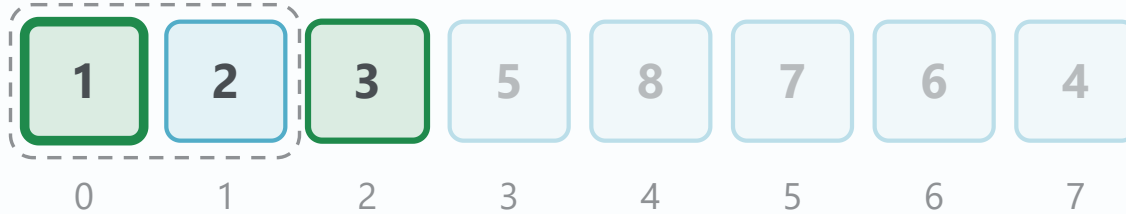
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



$a[0] \leftrightarrow a[1]$ takas $\rightarrow$ [1, 2, 3, 5, 8, 7, 6, 4]. Pivotun kendisi de yer değiştirebilir; konumu izlenir.

Rastgele Pivotlu QuickSort Çalıştırması

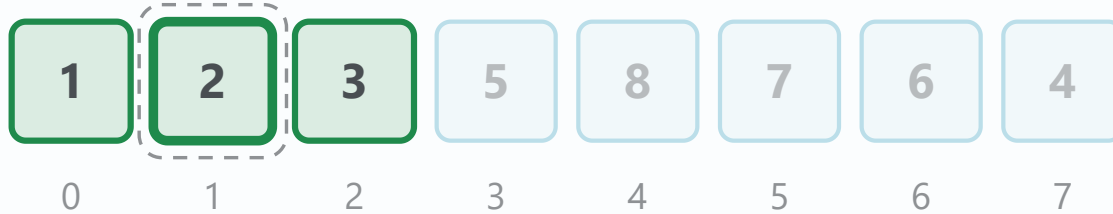
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



İşaretçiler karşılaştı ($i \geq j$): pivot **1** kalıcı yerinde (indis 0). Solunda yalnızca küçükler, sağında yalnızca büyükler var.

Rastgele Pivotlu QuickSort Çalıştırması

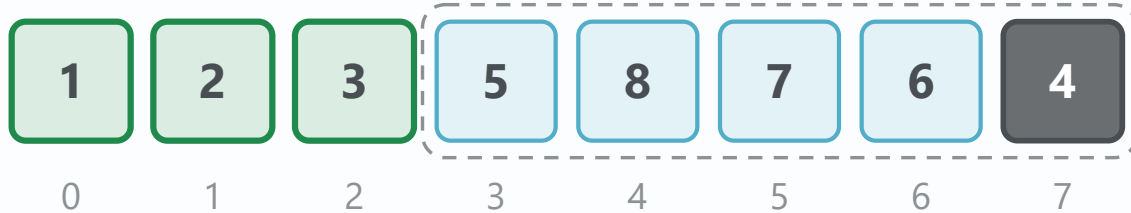
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Tek öğeli alt dizi $a[1] = 2$ zaten sıralı: kesinleşti.

Rastgele Pivotlu QuickSort Çalıştırması

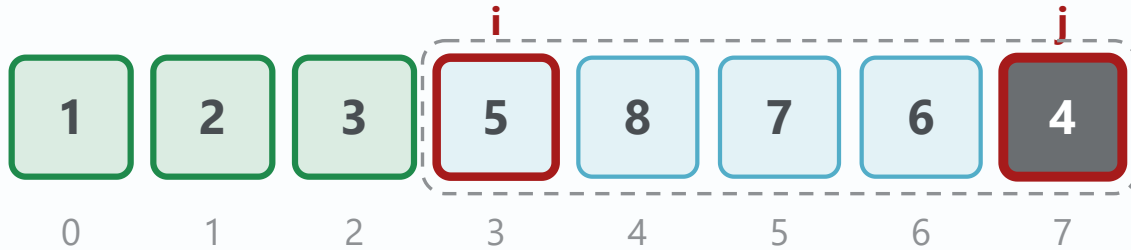
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Alt dizi $a[3..7] = [5, 8, 7, 6, 4]$: pivot **rastgele** seçildi → **4** (indis 7, siyah kutu).

Rastgele Pivotlu QuickSort Çalıştırması

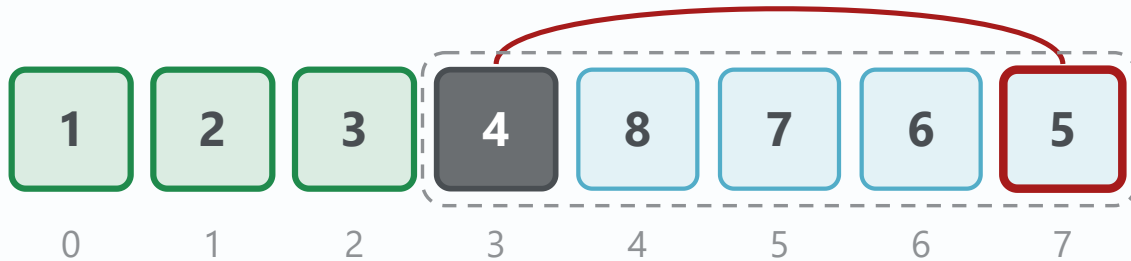
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



i soldan ilerler, 4'ten küçükleri atlar ve $a[3] = 5$ 'te durur; j sağdan ilerler, 4'ten büyükleri atlar ve $a[7] = 4$ 'te durur. $i < j \rightarrow$ takas edilecek.

Rastgele Pivotlu QuickSort Çalıştırması

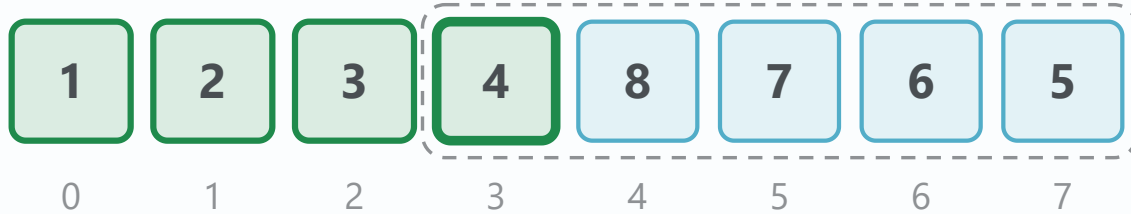
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



$a[3] \leftrightarrow a[7]$ takas $\rightarrow$ [1, 2, 3, 4, 8, 7, 6, 5]. Pivotun kendisi de yer değiştirebilir; konumu izlenir.

Rastgele Pivotlu QuickSort Çalıştırması

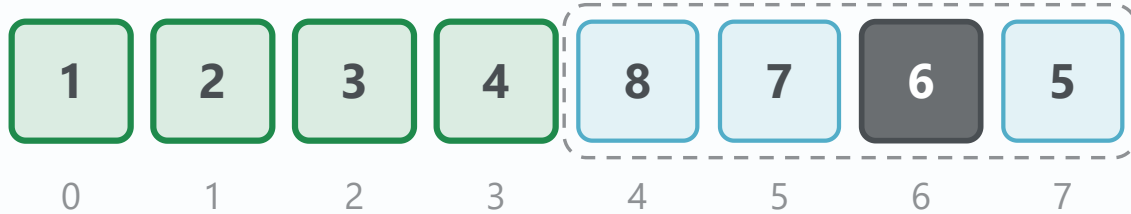
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



İşaretçiler karşılaştı ($i \geq j$): pivot **4** kalıcı yerinde (indis 3). Solunda yalnızca küçükler, sağında yalnızca büyükler var.

Rastgele Pivotlu QuickSort Çalıştırması

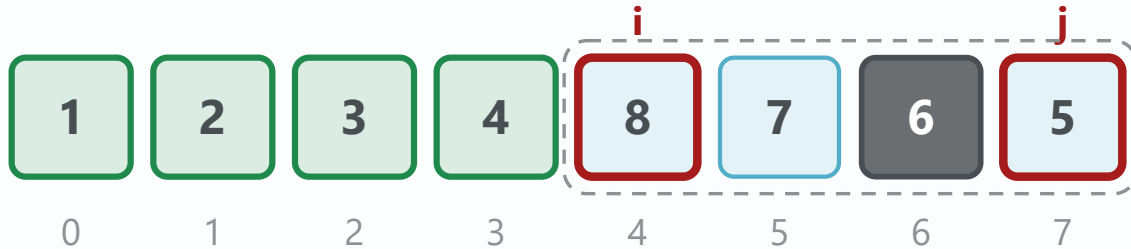
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Alt dizi $a[4..7] = [8, 7, 6, 5]$: pivot **rastgele** seçildi → **6** (indis 6, siyah kutu).

Rastgele Pivotlu QuickSort Çalıştırması

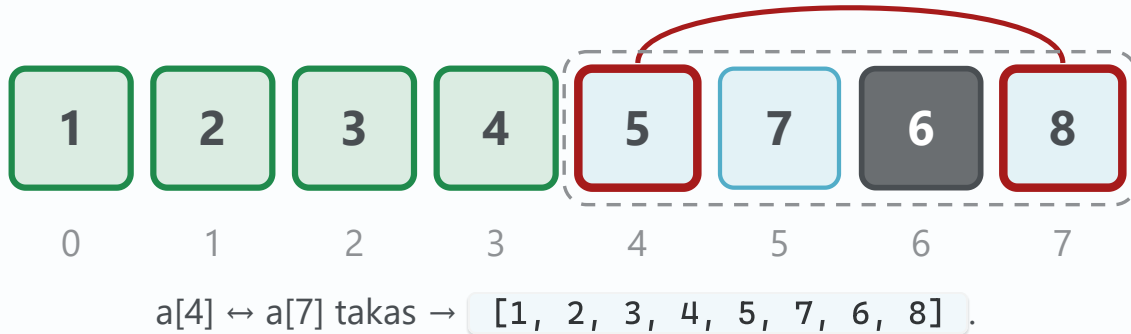
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



i soldan ilerler, 6'dan küçükleri atlar ve $a[4] = 8$ 'de durur; j sağdan ilerler, 6'dan büyükleri atlar ve $a[7] = 5$ 'te durur. $i < j \rightarrow$ takas edilecek.

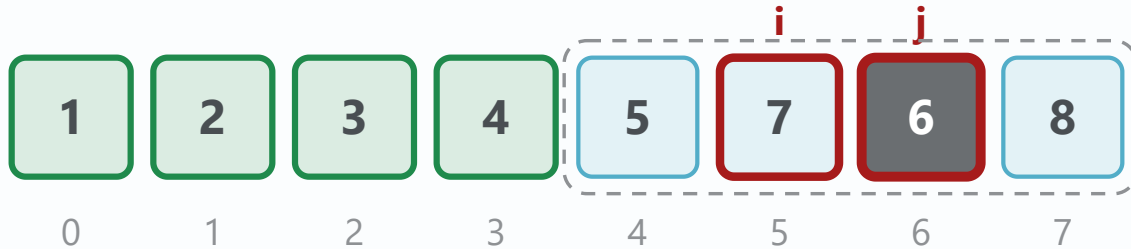
Rastgele Pivotlu QuickSort Çalıştırması

[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Rastgele Pivotlu QuickSort Çalıştırması

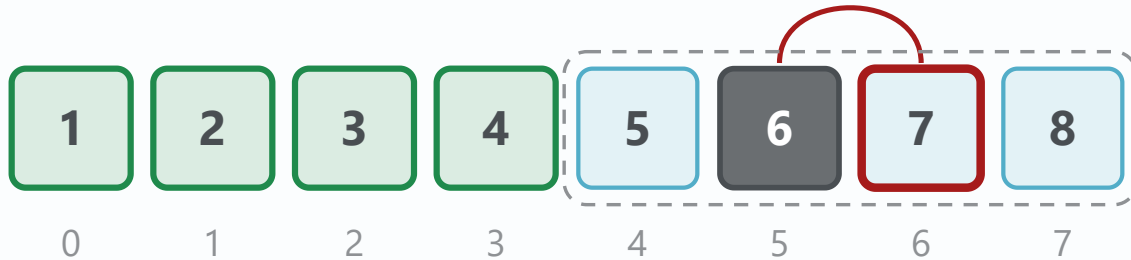
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



i soldan ilerler, 6'dan küçükleri atlar ve $a[5] = 7$ 'de durur; j sağdan ilerler, 6'dan büyükleri atlar ve $a[6] = 6$ 'da durur. $i < j \rightarrow$ takas edilecek.

Rastgele Pivotlu QuickSort Çalıştırması

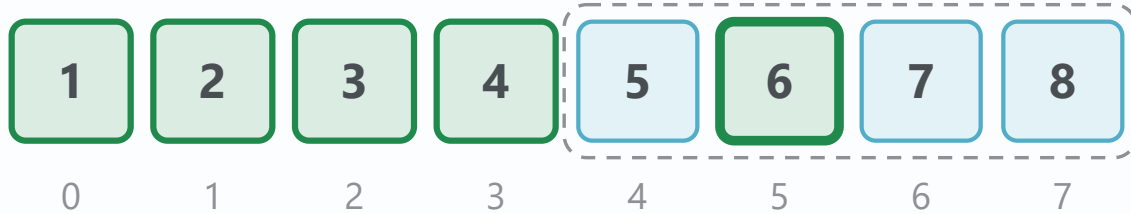
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



$a[5] \leftrightarrow a[6]$ takas $\rightarrow$ [1, 2, 3, 4, 5, 6, 7, 8]. Pivotun kendisi de yer değiştirebilir; konumu izlenir.

Rastgele Pivotlu QuickSort Çalıştırması

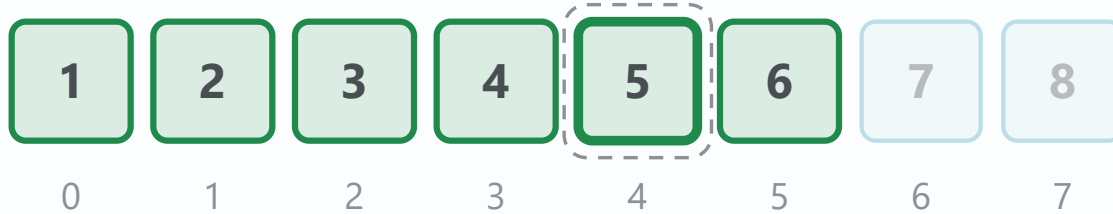
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



İşaretçiler karşılaştı ($i \geq j$): pivot **6** kalıcı yerinde (indis 5). Solunda yalnızca küçükler, sağında yalnızca büyükler var.

Rastgele Pivotlu QuickSort Çalıştırması

[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Tek öğeli alt dizi $a[4] = 5$ zaten sıralı: kesinleşti.

Rastgele Pivotlu QuickSort Çalıştırması

[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Alt dizi $a[6..7] = [7, 8]$: pivot **rastgele** seçildi → **8** (indis 7, siyah kutu).

Rastgele Pivotlu QuickSort Çalıştırması

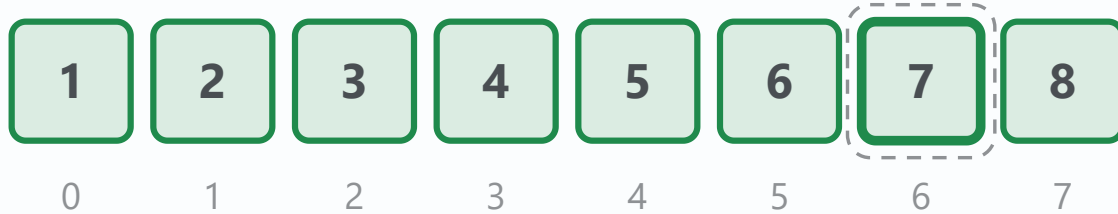
[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Pivot 8: i, 7'yi atlayıp 8'de durur; j zaten 8'de. $i \geq j \rightarrow$ takas yok, **8** kalıcı yerinde (indis 7).

Rastgele Pivotlu QuickSort Çalıştırması

[6, 5, 3, 1, 8, 7, 2, 4]; pivotlar rastgele seçildi: 3, 1, 4, 6, 8. Siyah: pivot · kırmızı çerçeve: i ve j işaretçileri · yeşil: kesinleşmiş · soluk: etkin alt dizinin dışı.



Tek öğeli alt dizi $a[6] = 7$ zaten sıralı: kesinleşti. **Dizi sıralandı:** [1, 2, 3, 4, 5, 6, 7, 8].

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }
    
```



zar.nextInt(8) → r = 2 : pivot a[2] = 3.

Adım 1 / 14

DEĞİŞKENLER

r = 2 pivot = - i = - k = -

a = [6, 5, 3, 1, 8, 7, 2, 4]

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);          // pivot yerine
9     return k;
10 }

```



Pivot sona taşınır: $a[2] \leftrightarrow a[7] \rightarrow [6, 5, 4, 1, 8, 7, 2, 3]$.

Adım 2 / 14

DEĞİŞKENLER

$r = 2$

$\text{pivot} = -$

$i = -$

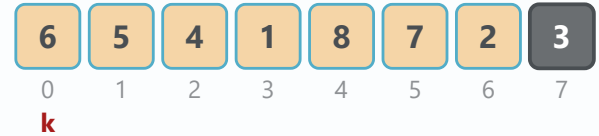
$k = -$

$a = [6, 5, 4, 1, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     ▶ int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }
    
```



pivot = 3 , **k = 0** : k, " $\leq$ pivot" bölgesinin bittiği yer.

Adım 3 / 14

DEĞİŞKENLER

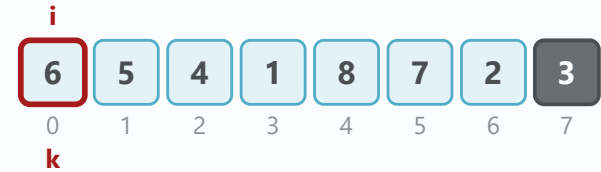
r = 2 **pivot = 3** **i = -** **k = 0**
a = [6, 5, 4, 1, 8, 7, 2, 3]

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8             takas(a, k, ust);    // pivot yerine
9     return k;
10 }

```



$a[0] = 6 \leq 3$ mü? Hayır $\rightarrow "> \text{pivot}"$ bölgesinde kalır, i ilerler.

Adım 4 / 14

DEĞİŞKENLER

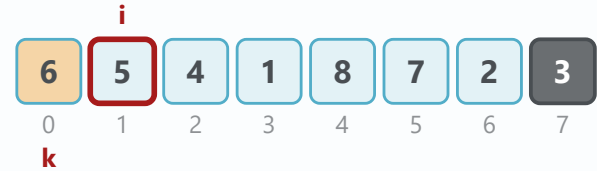
$r = 2$ $\text{pivot} = 3$ $i = 0$ $k = 0$
 $a = [6, 5, 4, 1, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8             takas(a, k, ust);    // pivot yerine
9     return k;
10 }

```



$a[1] = 5 \leq 3$ mü? Hayır $\rightarrow "> \text{pivot}"$ bölgesinde kalır, i ilerler.

Adım 5 / 14

DEĞİŞKENLER

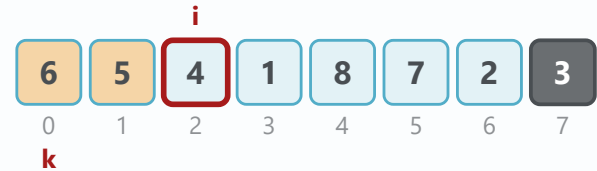
$r = 2$ $\text{pivot} = 3$ $i = 1$ $k = 0$
 $a = [6, 5, 4, 1, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8             takas(a, k, ust);    // pivot yerine
9     return k;
10 }

```



$a[2] = 4 \leq 3$ mü? Hayır $\rightarrow "> \text{pivot}"$ bölgesinde kalır, i ilerler.

Adım 6 / 14

DEĞİŞKENLER

$r = 2$ $\text{pivot} = 3$ $i = 2$ $k = 0$

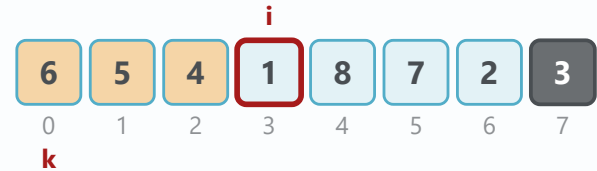
$a = [6, 5, 4, 1, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8             takas(a, k, ust);    // pivot yerine
9     return k;
10 }

```



$a[3] = 1 \leq 3$ mü? **Evet** → k konumuna taşınacak.

Adım 7 / 14

DEĞİŞKENLER

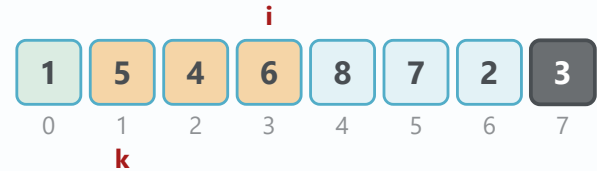
$r = 2$ $\text{pivot} = 3$ $i = 3$ $k = 0$
 $a = [6, 5, 4, 1, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }

```



$a[3] \leftrightarrow a[7]$ takas, $k = 4$. Yeşil-gri: ≤ 3 bölgesi, açık: > 3 bölgesi.

Adım 8 / 14

DEĞİŞKENLER

$r = 2$ $\text{pivot} = 3$ $i = 3$ $k = 4$

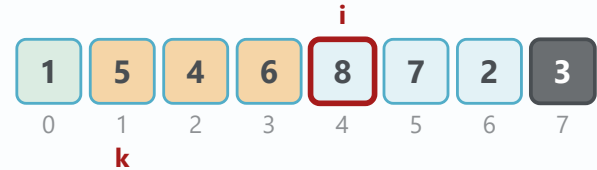
$a = [1, 5, 4, 6, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }

```



$a[4] = 8 \leq 3$ mü? Hayır $\rightarrow "> \text{pivot}"$ bölgesinde kalır, i ilerler.

Adım 9 / 14

DEĞİŞKENLER

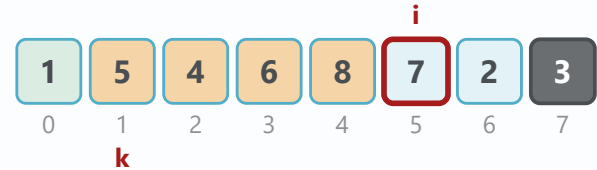
$r = 2$ $\text{pivot} = 3$ $i = 4$ $k = 1$
 $a = [1, 5, 4, 6, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8             takas(a, k, ust);    // pivot yerine
9     return k;
10 }

```



$a[5] = 7 \leq 3$ mü? Hayır $\rightarrow "> \text{pivot}"$ bölgesinde kalır, i ilerler.

Adım 10 / 14

DEĞİŞKENLER

$r = 2$ $\text{pivot} = 3$ $i = 5$ $k = 1$

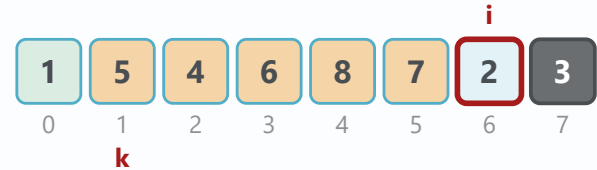
$a = [1, 5, 4, 6, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }

```



$a[6] = 2 \leq 3$ mü? **Evet** → k konumuna taşınacak.

Adım 11 / 14

DEĞİŞKENLER

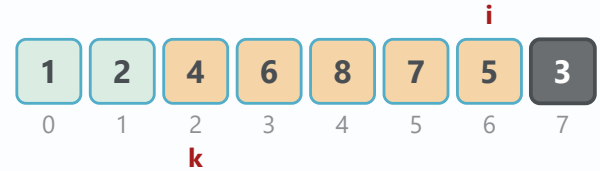
$r = 2$ $\text{pivot} = 3$ $i = 6$ $k = 1$
 $a = [1, 5, 4, 6, 8, 7, 2, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }

```



$a[6] \leftrightarrow a[1]$ takas, $k = 2$. Yeşil-gri: ≤ 3 bölgesi, açık: > 3 bölgesi.

Adım 12 / 14

DEĞİŞKENLER

$r = 2$ $\text{pivot} = 3$ $i = 6$ $k = 2$

$a = [1, 2, 4, 6, 8, 7, 5, 3]$

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }

```



Döngü bitti. Pivot, k = 2 konumuna takas edilir → [1, 2, 3, 6, 8, 7, 5, 4] .

Adım 13 / 14

DEĞİŞKENLER

r = 2

pivot = 3

i = -

k = 2

a = [1, 2, 3, 6, 8, 7, 5, 4]

Rastgele Pivotlu Parçalama (Lomuto)

```

1 static int bol(int[] a, int alt, int ust) {
2     int r = alt + zar.nextInt(ust - alt + 1);
3     takas(a, r, ust);           // pivot sona
4     int pivot = a[ust], k = alt;
5     for (int i = alt; i < ust; i++)
6         if (a[i] <= pivot)
7             takas(a, i, k++);    // küçükleri sola
8     takas(a, k, ust);           // pivot yerine
9     return k;
10 }

```



return 2 : 3 kalıcı yerinde. Sonra
 hızlıSırala(a, 0, 1) ve
 hızlıSırala(a, 3, 7) çağrılır.

Adım 14 / 14

DEĞİŞKENLER

r = 2 pivot = 3 i = - k = 2

a = [1, 2, 3, 6, 8, 7, 5, 4]

Rastgele QuickSort: Kod

```
static Random zar = new Random();

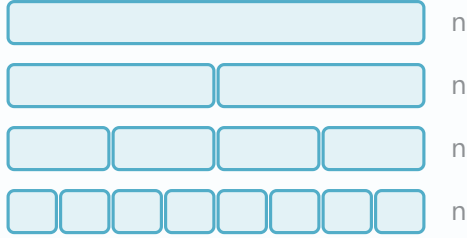
static void hizliSiralama(int[] a, int alt, int ust) {
    if (alt < ust) {
        int p = bol(a, alt, ust); // rastgele
        hizliSiralama(a, alt, p - 1);
        hizliSiralama(a, p + 1, ust);
    }
}
```

- **bol** (önceki slayt) rastgele bir indis seçer, pivotu sona taşır ve Lomuto parçalaması uygular.
- Tek bir **Random** nesnesi kullanılır; her çağrıda **new Random()** oluşturmak gereksiz ve kötü uygulamadır.
- Ek bellek: özyineleme yığını; beklenen derinlik $O(\log n)$. Küçük parçaya önce inilip büyük parça döngüye çevrilirse en kötü durumda da $O(\log n)$.
- Tekrarlı anahtarlar çoksa üç yollu parçalama ($<$, $=$, $>$) kullanılır.

RASTGELE QUICKSORT

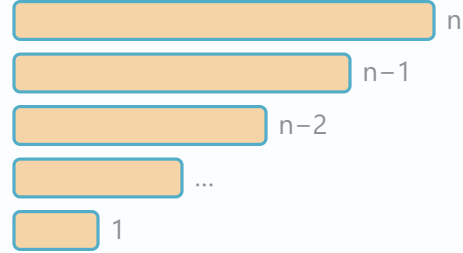
QuickSort Analizi

En iyi: dengeli bölme



$\log_2 n$ düzey $\times n$ iş = $O(n \log n)$

En kötü: hep en küçük/büyük pivot



n düzey: $n + (n-1) + \dots + 1 = O(n^2)$

Durum	Özyineleme bağıntısı	Süre
En iyi	$T(n) = 2T(n/2) + \Theta(n)$	$\Theta(n \log n)$
Beklenen (rastgele pivot)	$T(n) = (1/n) \sum_q [T(q) + T(n-1-q)] + \Theta(n)$	$\Theta(n \log n)$
En kötü	$T(n) = T(n-1) + \Theta(n)$	$\Theta(n^2)$

Rastgele pivotta en kötü durumun olasılığı: her adımda en küçük ya da en büyük seçilmeli $\rightarrow 2^{n-1}/n!$ ($n = 20$ için $\approx 2 \cdot 10^{-13}$).

Beklenen Karşılaştırma

- Öğeleri sıralı hâlleriyle adlandır: $z_1 < z_2 < \dots < z_n$. Gösterge $X_{ij} = 1 \Leftrightarrow z_i$ ile z_j karşılaştırılır.
- İki öge yalnızca biri **pivotken** karşılaştırılır. $\{z_i, \dots, z_j\}$ kümesinden **ilk** seçilen pivot arada bir öge olursa z_i ile z_j ayrı parçalara düşer ve hiç karşılaşmaz.
- Bu kümeden ilk pivot düzgün rastgeledir $\rightarrow P(X_{ij} = 1) = 2 / (j - i + 1)$.
- Doğrusallık: $E[X] = \sum_{i < j} 2 / (j - i + 1) < 2n H_n \approx 1,39 n \log_2 n$.
- Kesin değer: $E[X] = 2(n + 1)H_n - 4n$.

n	rastgele pivot (beklenen)	sıralı girdi, ilk öge pivot
10	24,4	45
100	648	4 950
1 000	10 986	499 500
10^6	$2,48 \cdot 10^7$	$5,0 \cdot 10^{11}$

Sıralama Algoritmaları Karşılaştırması

Algoritma	En iyi	Beklenen / ort.	En kötü	Ek bellek	Kararlı
BubbleSort	$O(n)$	$O(n^2)$	$O(n^2)$	$O(1)$	evet
MergeSort	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	evet
Rastgele QuickSort	$O(n \log n)$	$O(n \log n)$	$O(n^2)$	$O(\log n)$	hayır
BogoSort	$O(n)$	$\Theta(n \cdot n!)$	sınırsız	$O(1)$	hayır

- QuickSort, MergeSort'tan **daha fazla takas** yapabilir; ancak yerinde çalıştığı ve ön belleği iyi kullandığı için genellikle **daha hızlıdır**.
- BubbleSort gibi basit $O(n^2)$ algoritmalardan büyük n için çok daha hızlıdır.
- Java'nın `Arrays.sort(int[])` yöntemi ilkel türler için çift pivotlu QuickSort kullanır.

Rastgele Seçme (QuickSelect)

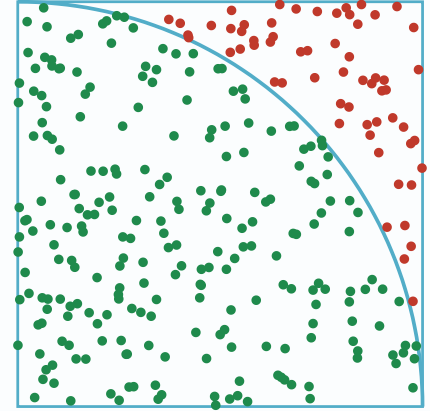
```
static int sec(int[] a, int alt, int ust, int s) {  
    if (alt == ust) return a[alt];  
    int p = bol(a, alt, ust);  
    if (s == p) return a[p];  
    if (s < p) return sec(a, alt, p - 1, s);  
    return sec(a, p + 1, ust, s);  
}
```

- Amaç: dizinin **s. en küçük** ögesini (ör. ortanca) sıralamadan bulmak.
- Aynı rastgele parçalama kullanılır, ama yalnızca **tek tarafa** inilir.
- Beklenen süre **$O(n)$** (ortanca için $\approx 3,39 n$ karşılaştırma); en kötü $O(n^2)$.
- Örnek: [6, 5, 3, 1, 8, 7, 2, 4] için $s = 2$ (0 tabanlı). Pivot 3 ile parçalama $p = 2$ verir (bkz. parçalama izlemesi) → ilk denemede yanıt **3**.

Monte Carlo ile π

- Birim kareye düzgün rastgele noktalar at. Çeyrek dairenin alanı $\pi/4$, karenin alanı 1 $\rightarrow P(\text{içeride}) = \pi/4$, yani $\pi \approx 4 \cdot \text{iç} / n$.
- Hata $\approx 1/\sqrt{n}$ ile azalır: bir basamak daha doğruluk için **100 kat** nokta gerekir.

```
static double piTahmin(int n) {  
    int ic = 0;  
    for (int t = 0; t < n; t++) {  
        double x = zar.nextDouble(), y = zar.nextDouble();  
        if (x * x + y * y ≤ 1) ic++;  
    }  
    return 4.0 * ic / n;  
}
```



300 nokta: 239 içeride $\rightarrow \pi \approx 4 \cdot 239/300 = \mathbf{3,187}$.

Deneyin: Monte Carlo π

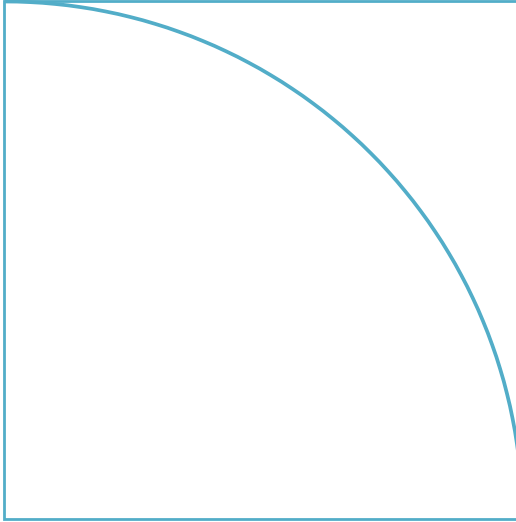
+ 10 nokta

+ 100

+ 1000

+ 10000

sıfırla



nokta sayısı n	0
içeride	0
π tahmini	—
hata $ \hat{\pi} - \pi $	—
beklenen hata $\approx 1,64/\sqrt{n}$	—

Nokta eklemek için bir düğmeye basın.

Yeşil: çeyrek dairenin içinde · kırmızı: dışında. Çizimde ilk 3000 nokta gösterilir, sayım tüm noktalarla sürer. $1,64 = 4 \cdot \sqrt{(p(1 - p))}$, $p = \pi/4$.

Miller–Rabin Asallık Testi

- **Fermat'nın küçük teoremi:** n asal ve $1 < a < n$ ise $a^{n-1} \equiv 1 \pmod{n}$. Bu eşitlik bozulursa n kesin bileşiktir; ama bazı bileşikler (Carmichael sayıları) her taban için eşitliği sağlar.
- **Karekök özelliği:** n asal ise $x^2 \equiv 1 \pmod{n}$ yalnızca $x \equiv \pm 1$ için sağlanır.
- $n - 1 = 2^s \cdot d$ (d tek) yazılır; rastgele a için $a^d, a^{2d}, \dots, a^{2^{s-1}d} \pmod{n}$ dizisine bakılır.
- Dizi 1 ile başlamıyor ve $n - 1$ hiç görünmüyorsa a, n 'nin **bileşik** olduğunun **tanığıdır** (witness). Aksi hâlde n bu turu "geçer".
- Bileşik n için tabanların en çok **$1/4$** 'ü yalancıdır $\rightarrow k$ bağımsız turda hata $\leq 4^{-k}$. Her tur $O(\log n)$ modüler çarpma: toplam $O(k \log^3 n)$ bit işlemi.

Tek yönlü hata: "bileşik" yanıtı **kesindir**; "asal" yanıtı yalnızca **muhtemelen** doğrudur.

Miller–Rabin: Kod

```
static boolean asalMi(long n, int k) {
    if (n < 4) return n == 2 || n == 3;
    if (n % 2 == 0) return false;
    long d = n - 1; int s = 0;      //  $n-1 = 2^s \cdot d$ 
    while (d % 2 == 0) { d /= 2; s++; }
    for (int tur = 0; tur < k; tur++) {
        long a = 2 + zar.nextLong(n - 3); // [2, n-2]
        long x = usMod(a, d, n);          //  $a^d \bmod n$ 
        if (x == 1 || x == n - 1) continue;
        boolean tanik = true;
        for (int r = 1; r < s && tanik; r++) {
            x = x * x % n;
            if (x == n - 1) tanik = false;
        }
        if (!tanik) return false;          // kesin bileşik
    }
    return true;                          // muhtemelen asal
}
```

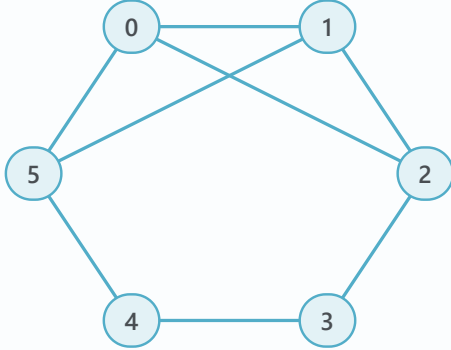
- `usMod` : hızlı modüler üs alma (kare al-çarp), $O(\log d)$ çarpma.
- Taban a , $[2, n - 2]$ aralığından düzgün seçilir.
- `long` ile `x * x` yalnızca $n < 3 \cdot 10^9$ için taşmaz; daha büyük n için `BigInteger.modPow` ya da `isProbablePrime`.

Miller–Rabin Örnekleri

n	n - 1	a	$a^d, a^{2^d}, \dots \pmod n$	sonuç
61	$2^2 \cdot 15$	2	11, 60	geçti (asal olabilir)
221	$2^2 \cdot 55$	174	47, 220	geçti (asal olabilir)
221	$2^2 \cdot 55$	137	188, 205	bileşik (a tanık)
561	$2^4 \cdot 35$	2	263, 166, 67, 1	bileşik (a tanık)

- **61** asaldır: dizi $n - 1 = 60$ 'a ulaşır.
- **221 = 13 · 17**: $a = 174$ bir **güçlü yalancıdır** (dizi 220'ye ulaşır); $a = 137$ ise tanıktır. Bu yüzden birden çok tur gerekir.
- **561 = 3 · 11 · 17** bir Carmichael sayısıdır: $2^{560} \equiv 1$, yani Fermat testi aldanır. Miller–Rabin ise $67^2 \equiv 1$ olduğunu görür: 67, 1'in ± 1 dışında bir karekökü → **bileşik**.

Karger Min-Kesim Algoritması

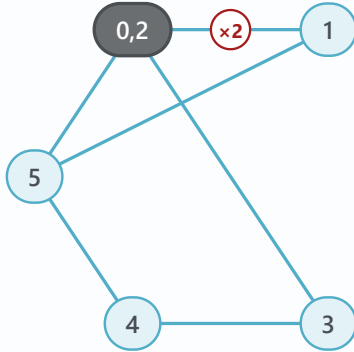


Başlangıç: 6 düğüm, 8 kenar. Minimum kesim 2'dir (ör. {3, 4} | geri kalanı). Koyu düğüm: son büzmede oluşan süper düğüm.

Adım 1 / 5

- **Problem:** çizgeyi iki parçaya ayıran en az kenarlı kesim (global min-cut).
- **Algoritma:** 2 düğüm kalana dek düzgün rastgele bir kenarı büz; kalan kenar sayısı bir kesimdir.
- Bir çalıştırma minimum kesimi en az $2/(n(n-1))$ olasılıkla bulur ($n = 6: \geq 1/15$). Bu çizgede 200 000 benzetimde gözlenen $\approx 0,46$.
- Çıktı yanlış (daha büyük kesim) olabilir → **Monte Carlo**. Ör. ilk kenarlar (1,2), (4,5), (3,4), (0,1) seçilirse {0,1,2} | {3,4,5} kesimi, 3 kenar.
- $n(n-1)/2 \cdot \ln n$ tekrar → başarısızlık $\leq 1/n$; toplam $O(n^4 \log n)$ (basit uygulama).

Karger Min-Kesim Algoritması

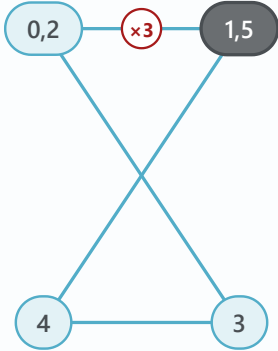


Rastgele kenar (0, 2) **büzüldü**: uçları tek süper düğümde birleşti, aradaki kenarlar (öz-döngüler) silindi, paralel kenarlar ($\times k$) korunur. Kalan: 7 kenar.

Adım 2 / 5

- **Problem:** çizgeyi iki parçaya ayıran en az kenarlı kesim (global min-cut).
- **Algoritma:** 2 düğüm kalana dek düzgün rastgele bir kenarı büz; kalan kenar sayısı bir kesimdir.
- Bir çalıştırma minimum kesimi en az **$2/(n(n-1))$** olasılıkla bulur ($n = 6: \geq 1/15$). Bu çizgede 200 000 benzetimde gözlenen $\approx 0,46$.
- Çıktı yanlış (daha büyük kesim) olabilir → **Monte Carlo**. Ör. ilk kenarlar (1,2), (4,5), (3,4), (0,1) seçilirse $\{0,1,2\} \mid \{3,4,5\}$ kesimi, 3 kenar.
- $n(n-1)/2 \cdot \ln n$ tekrar → başarısızlık $\leq 1/n$; toplam $O(n^4 \log n)$ (basit uygulama).

Karger Min-Kesim Algoritması

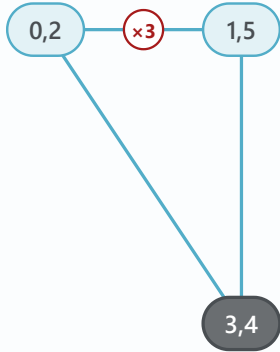


Rastgele kenar (1, 5) **büzüldü**: uçları tek süper düğümde birleşti, aradaki kenarlar (öz-döngüler) silindi, paralel kenarlar ($\times k$) korunur. Kalan: 6 kenar.

Adım 3 / 5

- **Problem:** çizgeyi iki parçaya ayıran en az kenarlı kesim (global min-cut).
- **Algoritma:** 2 düğüm kalana dek düzgün rastgele bir kenarı büz; kalan kenar sayısı bir kesimdir.
- Bir çalıştırma minimum kesimi en az $2/(n(n-1))$ olasılıkla bulur ($n = 6: \geq 1/15$). Bu çizgede 200 000 benzetimde gözlenen $\approx 0,46$.
- Çıktı yanlış (daha büyük kesim) olabilir → **Monte Carlo**. Ör. ilk kenarlar (1,2), (4,5), (3,4), (0,1) seçilirse $\{0,1,2\} \mid \{3,4,5\}$ kesimi, 3 kenar.
- $n(n-1)/2 \cdot \ln n$ tekrar → başarısızlık $\leq 1/n$; toplam $O(n^4 \log n)$ (basit uygulama).

Karger Min-Kesim Algoritması

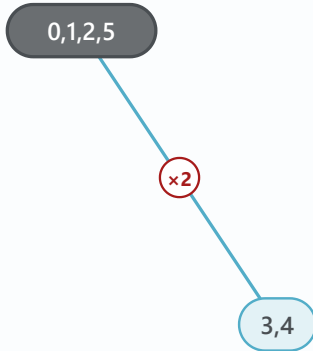


Rastgele kenar (3, 4) **büzüldü**: uçları tek süper düğümde birleşti, aradaki kenarlar (öz-döngüler) silindi, paralel kenarlar ($\times k$) korunur. Kalan: 5 kenar.

Adım 4 / 5

- **Problem:** çizgeyi iki parçaya ayıran en az kenarlı kesim (global min-cut).
- **Algoritma:** 2 düğüm kalana dek düzgün rastgele bir kenarı büz; kalan kenar sayısı bir kesimdir.
- Bir çalıştırma minimum kesimi en az $2/(n(n-1))$ olasılıkla bulur ($n = 6: \geq 1/15$). Bu çizgede 200 000 benzetimde gözlenen $\approx 0,46$.
- Çıktı yanlış (daha büyük kesim) olabilir → **Monte Carlo**. Ör. ilk kenarlar (1,2), (4,5), (3,4), (0,1) seçilirse $\{0,1,2\} \mid \{3,4,5\}$ kesimi, 3 kenar.
- $n(n-1)/2 \cdot \ln n$ tekrar → başarısızlık $\leq 1/n$; toplam $O(n^4 \log n)$ (basit uygulama).

Karger Min-Kesim Algoritması



Rastgele kenar (0, 1) **büzüldü**: uçları tek süper düğümde birleşti, aradaki kenarlar (öz-döngüler) silindi, paralel kenarlar ($\times k$) korunur. Kalan: 2 kenar. İki süper düğüm kaldı
→ bulunan kesim = **2** (minimum, başarılı çalıştırma).

Adım 5 / 5

- **Problem:** çizgeyi iki parçaya ayıran en az kenarlı kesim (global min-cut).
- **Algoritma:** 2 düğüm kalana dek düzgün rastgele bir kenarı büz; kalan kenar sayısı bir kesimdir.
- Bir çalıştırma minimum kesimi en az **$2/(n(n-1))$** olasılıkla bulur ($n = 6: \geq 1/15$). Bu çizgede 200 000 benzetimde gözlenen $\approx 0,46$.
- Çıktı yanlış (daha büyük kesim) olabilir → **Monte Carlo**. Ör. ilk kenarlar (1,2), (4,5), (3,4), (0,1) seçilirse $\{0,1,2\} \mid \{3,4,5\}$ kesimi, 3 kenar.
- $n(n-1)/2 \cdot \ln n$ tekrar → başarısızlık $\leq 1/n$; toplam $O(n^4 \log n)$ (basit uygulama).

Diğer Uygulamalar

Çizge boyama

Farklı boyama düzenlerini rastgele kararlarla dener; komşu düğümlere aynı renk vermeden iyi boyamalar arar.

Rastgele DFS

Komşuları rastgele sırayla gezerek bağlantılılığı ve kapsayan ağaçları bulur (ör. labirent üretimi).

Eşleştirme

Rastgele artımlı algoritmalar kenarları rastgele sırayla ekleyerek maksimum eşleşmeye yaklaşır.

Yaklaşık sayma

Count-Min Sketch, rastgele hash fonksiyonlarıyla büyük veri akışında frekansları denetimli hatayla tahmin eder.

Matris işlemleri

Rastgele matris yaklaşımı düşük ranklı yaklaşımlar üretir: boyut indirgeme, veri sıkıştırma.

Hesaplamalı geometri

Rastgele artımlı dışbükey örtü, en yakın nokta çifti, Delaunay üçgenleme: beklenen $O(n \log n)$.

Rastgele zamanlama (iş atama), benzetimli tavlama ve genetik algoritmalar da rastgeleliği arama için kullanır (sonraki slayt).

Rastgele Arama Sezgiselleri

Benzetimli tavlama (Simulated Annealing)

- Rastgele bir komşu çözüm üret; daha iyiye kabul et.
- Daha kötüyse $e^{-\Delta/T}$ olasılıkla yine kabul et: yerel minimumdan kaçış.
- Sıcaklık T her adımda azalır (ör. $T_0 = 100$, soğuma oranı 0,003).

Genetik algoritma

- Rastgele bir çözüm **popülasyonu** ile başla (ör. 100 birey).
- Uygunluğa göre seç, **çaprazla** (rastgele kesim noktası), küçük olasılıkla **mutasyon** uygula (0,05).
- Kuşaklar boyunca iyi çözümler çoğalır.

Bu yöntemler **sezgiseldir**: optimuma yakınsama garantisi yoktur ve sonuç başlangıç çözümüne duyarlı olabilir. NP-zor problemlerde iyi çözümleri makul sürede bulmak için kullanılır.

Sözde Rastgele Sayılar

- Bilgisayar gerçek rastgelelik yerine **sözde rastgele** (pseudo-random) sayı üretir: deterministik ama rastgele görünen bir dizi.
- **Doğrusal eşlik üretici** (LCG): $X_{k+1} = (a \cdot X_k + c) \bmod m$. `java.util.Random` : $a = 25214903917$, $c = 11$, $m = 2^{48}$.
- Aynı **tohum** (seed) aynı diziye verir: `new Random(42)` ile çalıştırmalar tekrarlanabilir (hata ayıklama, deney).
- Kriptografi için `SecureRandom` ; çok iş parçacıklı kodda `ThreadLocalRandom` .

Rastgele aralıklar

- `zar.nextInt(n)` → $0 \dots n - 1$
- `alt + zar.nextInt(ust - alt + 1)` → $alt \dots ust$
- `zar.nextDouble()` → $[0, 1)$
- `zar.nextInt(6) + 1` → zar atışı: $1 \dots 6$

ÖZET

Özet

Konu	Hatırlanacaklar
Rastgele algoritma	girdi + rastgele bitler; süre/çıktı rastgele değişken; beklenti algoritmanın seçimleri üzerinden
Las Vegas	her zaman doğru, süre rastgele: QuickSort, QuickSelect, BogoSort
Monte Carlo	süre sınırlı, küçük hata olasılığı; tekrarlar q^k 'ya iner
Fisher-Yates	<code>j = nextInt(i + 1)</code> , $\Theta(n)$, $n!$ sonuç eşit olasılıklı; saf karıştırma yanlı
BogoSort	$p = 1/n!$, beklenen $\Theta(n \cdot n!)$, en kötü sınırsız
Rastgele QuickSort	beklenen $\approx 1,39 n \log_2 n$ karşılaştırma, her girdi için $O(n \log n)$; en kötü $O(n^2)$
Miller-Rabin	$n - 1 = 2^s d$; hata $\leq 4^{-k}$; "bileşik" kesin
Karger	rastgele kenar büzme; başarı $\geq 2/(n(n-1))$; çok kez tekrarlar
π tahmini	$4 \cdot i_{\text{ç}} / n$; hata $\propto 1/\sqrt{n}$

Etkileşimli Simülatörler

Sayı Kuramı Algoritmaları

Öklid ve genişletilmiş Öklid, modüler aritmetik, hızlı modüler üs alma, asalılık testi, modüler ters, Fermat'ın küçük teoremi, Çin kalan teoremi ve Eratosthenes eleği (İngilizce). Oyun alanındaki **$a^b \bmod m$** düğmesiyle Miller–Rabin turlarındaki $a^d \bmod n$ değerlerini hesaplayın, **Is a Prime?** (deneme bölmesi) ile sonucu doğrulayın.

Bu bölümde: Monte Carlo π

"Deneyin: Monte Carlo π " slaytında birim kareye nokta atarak π tahmininin n büyüdükçe nasıl yakınsadığını ve hatanın yaklaşık $1/\sqrt{n}$ ile küçüldüğünü gözleyin.

ÖZET

Sık Yapılan Hatalar

Saf karıştırma

Her i için `nextInt(n)` ile takas yanlıdır (n^n yol, $n!$ permütasyon). Doğrusu `nextInt(i + 1)`.

Ortalama ile beklenen süreyi karıştırmak

Ortalama durum girdi dağılımı varsayar; beklenen süre her girdi için algoritmanın yazı-turaları üzerinden hesaplanır.

Taşma

`int` ile `x * x % n`, $n > 46\,340$ için taşar.
`long`, `Math.multiplyHigh` ya da `BigInteger.modPow` kullanın.

"Rastgele pivot en kötüyü yok eder"

Hayır: en kötü durum hâlâ $O(n^2)$. Değişen, hiçbir girdinin sürekli kötü olmaması ve kötü durumun olasılığının çok küçük olmasıdır.

Monte Carlo yanıtını kesin saymak

Miller–Rabin'de "asal" yanıtı yalnızca **muhtemelen** asaldır; tur sayısı k hata sınırını belirler.

Her çağrıda new Random()

Gereksiz nesne üretir ve tohumlar ilişkili olabilir. Tek bir `Random` paylaşın; Karger'de kenarı **düzgün** seçin (önce düğüm, sonra komşu seçmek düzgün değildir).

ÖZET

Örnek Problemler

1. Fisher–Yates `[1, 2, 3, 4]` dizisine uygulanıyor; $i = 3, 2, 1$ için çekilen j değerleri 1, 2, 0. Sonuç nedir?
2. 5 farklı öğeli bir dizide Bogosort'un ilk karıştırmada bitme olasılığı ve beklenen karıştırma sayısı nedir?
3. Rastgele QuickSort'ta en küçük ve en büyük öge hangi olasılıkla karşılaştırılır? Sıralı hâlde komşu iki öge?
4. Miller–Rabin: $n = 25$ için $a = 7$ ve $a = 2$ tabanlarıyla testi uygulayın.
5. Monte Carlo ile 1000 noktanın 787'si çeyrek dairenin içine düştü. π tahmini nedir?

► Cevapları göster

ÖZET

Alıştırmalar

1. `[3, 8, 2, 5, 1, 4, 7, 6]` dizisinde `bol` çağrılıyor ve `r = 5` (pivot 4) çekiliyor. Parçalama sonrası dizi ve dönüş değeri nedir?
2. Sıralı 8 öğeli bir dizide ilk öğeyi pivot seçen QuickSort kaç karşılaştırma yapar?
3. Miller–Rabin'de hata olasılığını 10^{-6} 'nın altına indirmek için en az kaç tur gerekir?
4. Karger'i 6 düğümlü bir çizgede (başarı $\geq 1/15$) kaç kez tekrarlıyorsanız başarısızlık olasılığı $1/6$ 'nın altına iner?
5. Başarı olasılığı $1/3$ olan ve sonucu doğrulanabilen bir Monte Carlo algoritmasını Las Vegas'a çevirin. Beklenen tekrar sayısı nedir?

► Cevapları göster

Kaynaklar

- R. Motwani ve P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995.
- M. Mitzenmacher ve E. Upfal, *Probability and Computing*, 2. baskı, Cambridge University Press, 2017.
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 5, 7, 9, 31).
- C. A. R. Hoare, "Quicksort", *The Computer Journal* 5(1), 1962.
- M. O. Rabin, "Probabilistic Algorithm for Testing Primality", *Journal of Number Theory* 12(1), 1980.
- D. R. Karger, "Global Min-cuts in RNC, and Other Ramifications of a Simple Min-Cut Algorithm", *SODA*, 1993.
- D. E. Knuth, *The Art of Computer Programming*, Cilt 2, 3. baskı, 1997 (Bölüm 3.4.2: karıştırma).

Son

Bölüm 8: Rastgele Algoritmalar

Sonraki bölüm: Bölüm 9: Çevrimiçi Algoritmalar · [Terim sözlüğü](#)