Bölüm 2: Sıralama Algoritmaları

Algoritmalar

Kararlılık ve yerinde sıralama | Kabarcık | Eklemeli | Seçimli | Shell | Birleştirme | Hızlı | Heap sort | Saymalı ve basamak | Timsort | Karşılaştırma

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Kavramlar

Sıralamayı; zaman, ek bellek, **kararlılık**, **yerinde** çalışma ve karşılaştırmalı / karşılaştırmasız olma ölçütleriyle değerlendirmek.

Basit algoritmalar

Kabarcık, eklemeli, seçimli ve Shell sıralamasını bir dizi üzerinde adım adım izlemek ve Java ile yazmak.

Böl ve fethet, heap

Birleştirme sıralamasının, hızlı sıralamanın ve heap sort'un çalışmasını açıklamak; O(n log n) ve en kötü durum davranışlarını gerekçelendirmek.

Doğrusal zaman ve seçim

Karşılaştırma alt sınırını, saymalı ve basamak sıralamasını açıklamak; bir problem için uygun algoritmayı seçmek.

GİRİŞ

Sıralama Algoritmaları

- Alışveriş listelerinden sosyal medya gönderilerine kadar dijital dünyada sürekli **veri** ile karşılaşırız.
- Verinin etkin kullanılabilmesi için **düzenli** tutulması gerekir.
- **Sıralama algoritmaları** (sorting algorithms) bir listedeki öğeleri belirli bir **ölçüte** göre düzenler:
  - sayısal değer,
  - alfabetik sıra,
  - tarih …
- Sıra **artan** (küçükten büyüğe) ya da **azalan** olabilir.
- Biçimsel tanım: girdi $\langle a_1, a_2, \ldots, a_n \rangle$; çıktı, aynı sayıların $a'_1 \leq a'_2 \leq \ldots \leq a'_n$ koşulunu sağlayan bir **permütasyonu**.



Sıralı veride **ikili arama** O(log n), tekrar bulma, birleştirme, ortanca ve en küçük/en büyük k öğe gibi işlemler çok kolaylaşır.

GİRİŞ

Sıralama Algoritmalarının Çeşitleri

Farklı sıralama algoritmaları farklı çalışma prensiplerine sahiptir:

Algoritma	Temel fikir
Kabarcık (Bubble Sort)	Komşu öğeleri karşılaştırır, yanlış sıradaysa yer değiştirir.
Eklemeli (Insertion Sort)	Sıradaki öğeyi sıralı kısımda doğru konuma yerleştirir.
Seçimli (Selection Sort)	En küçük öğeyi bulup başa yerleştirir, sonra kalanları sıralar.
Shell (Shell Sort)	Uzak öğelerle başlayan, aralığı küçülen eklemeli sıralama.
Birleştirme (Merge Sort)	Diziyi yarıya böler, sıralar, sonra birleştirir.
Hızlı (Quick Sort)	Pivot etrafında parçalar, iki parçayı özyinelemeli sıralar.
Heap (Heap Sort)	Max-heap kurar, en büyüğü tekrar tekrar sona taşır.
Saymalı / Basamak (Counting / Radix)	Karşılaştırma yapmaz; anahtarları sayar ya da basamak basamak dağıtır.
Timsort	Eklemeli ve birleştirme sıralamasının karması; Java ve Python'un nesne sıralaması.

GİRİŞ

Bir Sıralama Algoritmasını Değerlendirmek

Zaman

En iyi, ortalama ve en kötü durumda karşılaştırma ve takas (taşıma) sayısı: $\Theta(n^2)$, $\Theta(n \log n)$, $\Theta(n + k)$ ...

Ek bellek

Girdinin dışında gereken alan. O(1) ek bellekle çalışan algoritma **yerinde** (in-place) sıralar.

Kararlılık

Eşit anahtarlı öğelerin **göreli sırası korunuyorsa** algoritma kararlıdır (stable).

Karşılaştırmalı mı?

Yalnızca "a ≤ b?" sorusuyla çalışan algoritmalar en kötü durumda $\Omega(n \log n)$ karşılaştırma yapar. Saymalı ve basamak sıralaması bu sınırın dışındadır.

Uyarlanabilirlik

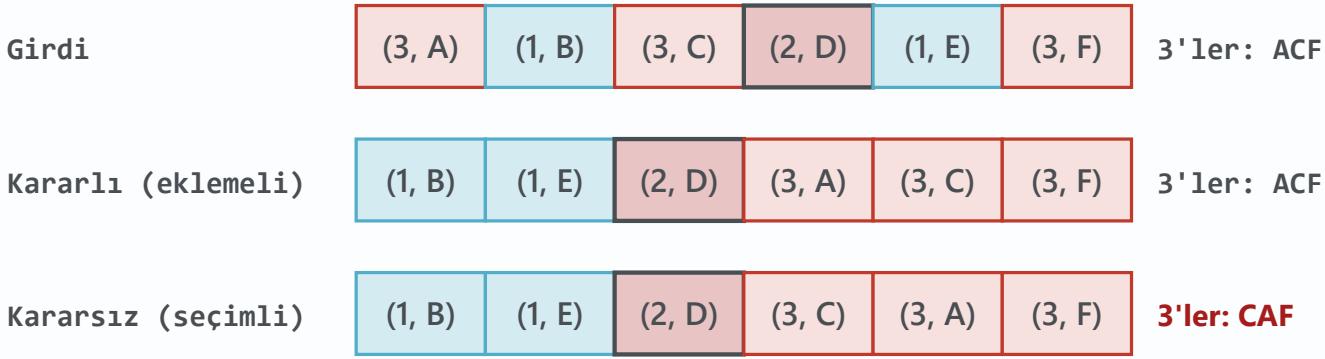
Neredeyse sıralı girdide daha hızlı çalışan algoritma **uyarlanabilirdir** (adaptive): eklemeli, bayraklı kabarcık, Timsort.

Çevrim içi

Veri geldikçe sıralayabilen algoritma **çevrim içidir** (online): eklemeli sıralama.

GİRİŞ

Kararlılık (Stability): Örnek



Kayıtlar yalnızca **sayıya** (anahtar) göre sıralanır; harf, kaydın kimliğidir. Kararlı algoritmada anahtarı 3 olanlar girdideki gibi **A, C, F** sırasında kalır; seçimli sıralamanın uzun mesafeli takasları bu sırayı **C, A, F** yapar. (Bu girdi, simülatör laboratuvarındaki örnektir.)

Kararlılık, **çok anahtarlı sıralamada** önemlidir: öğrenciler önce ada, sonra (kararlı bir algoritmayla) bölüme göre sıralanırsa her bölümün içi alfabetik kalır. Basamak sıralaması da bu yüzden kararlı bir ara adım ister.

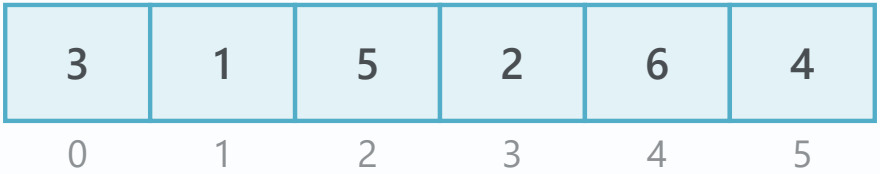
Kabarcık Sıralaması (Bubble Sort)

- Her turda dizideki **büyük öğeler** dizinin sonuna doğru kaydırılır; su içindeki kabarcık gibi yükselir.
- **Komşu** öğeler ikili olarak karşılaştırılır.
- Sol öğe büyükse yerleri değiştirilir (**swap**).
- i. tur bittiğinde en büyük i öğe **son yerlerindedir**; bir sonraki tur bir öğe kısa olur.
- Bir turda hiç takas olmazsa dizi sıralıdır → **erken çıkış**.



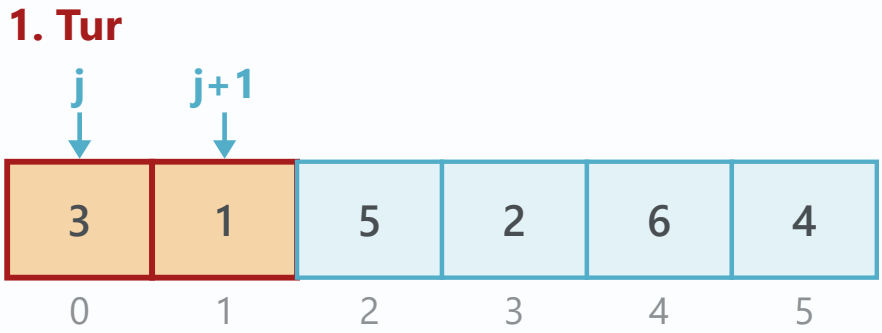
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

Başlangıç



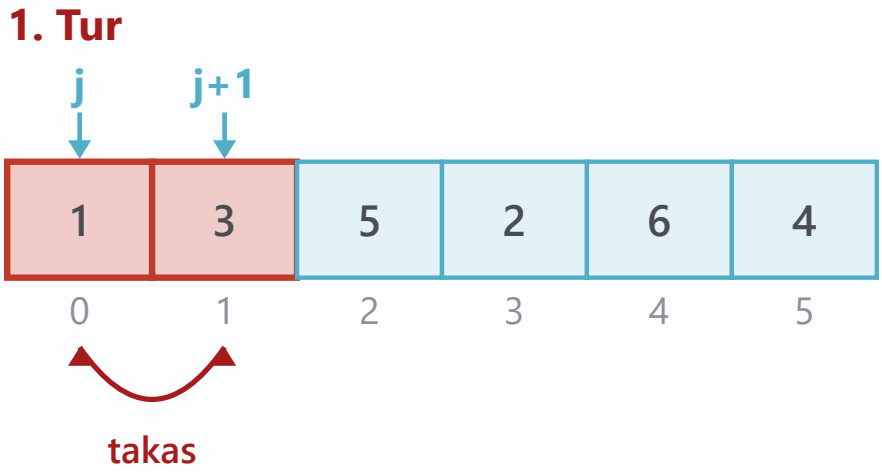
Dizi: [3, 1, 5, 2, 6, 4]. Her turda komşu çiftler soldan sağa karşılaştırılır; büyük olan sağa kayar.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]



1. tur, j = 0: 3 > 1? **Evet** → yer değiştirilecek.

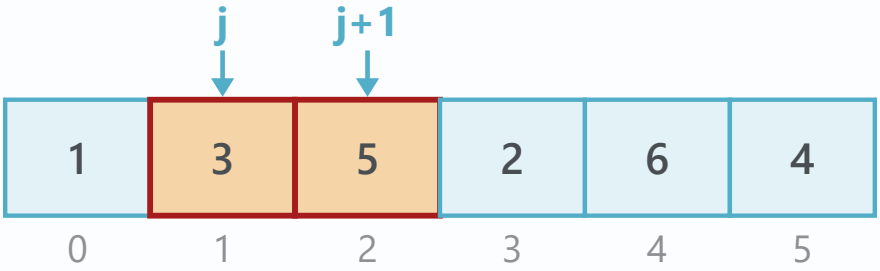
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]



Takas: dizi artık [1, 3, 5, 2, 6, 4].

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

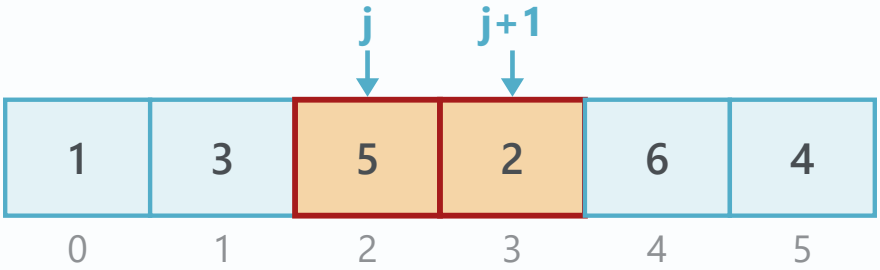
1. Tur



1. tur, j = 1: 3 > 5? Hayır → yerinde kalır.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

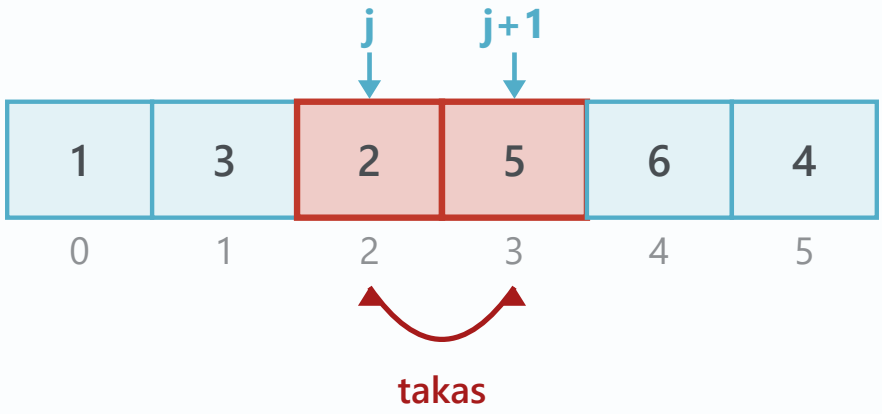
1. Tur



1. tur, j = 2: 5 > 2? **Evet** → yer değiştirilecek.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

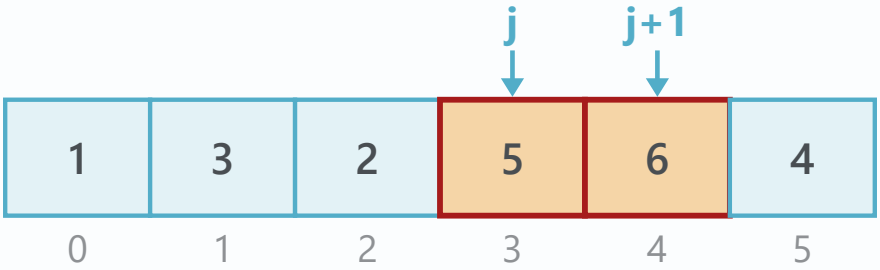
1. Tur



Takas: dizi artık [1, 3, 2, 5, 6, 4].

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

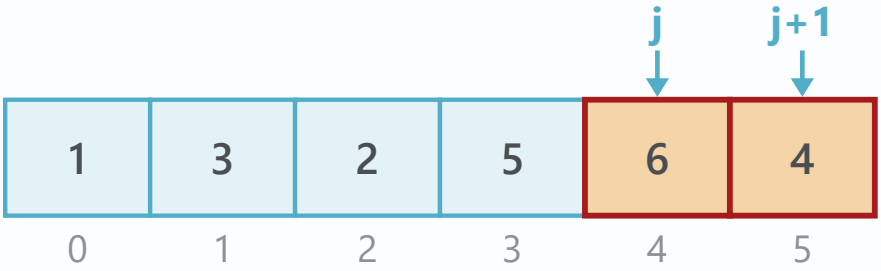
1. Tur



1. tur, j = 3: 5 > 6? Hayır → yerinde kalır.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

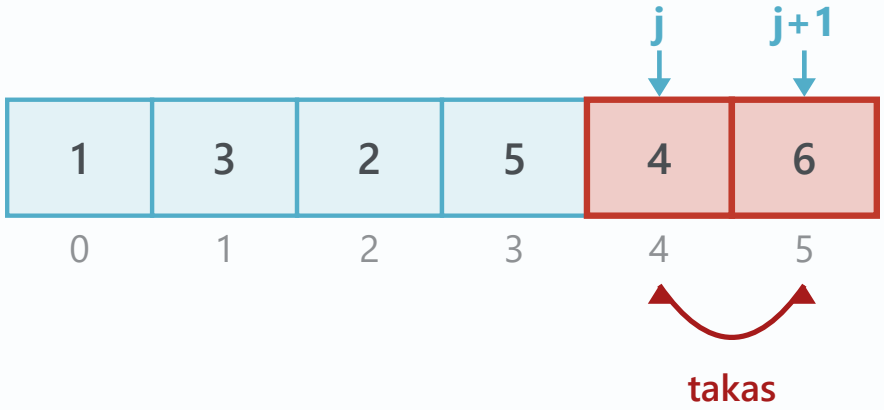
1. Tur



1. tur, j = 4: 6 > 4? **Evet** → yer değiştirilecek.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

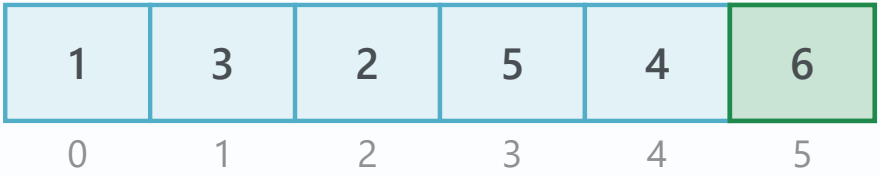
1. Tur



Takas: dizi artık [1, 3, 2, 5, 4, 6].

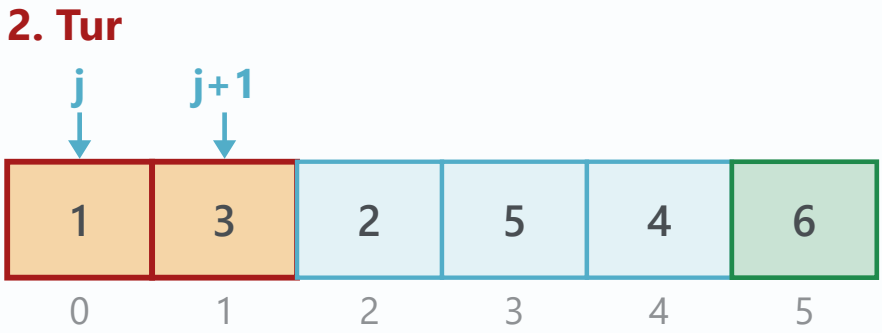
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

1. Tur sonu



1. tur bitti: en büyük kalan öğe **6** son yerine (indis 5) yerleşti.

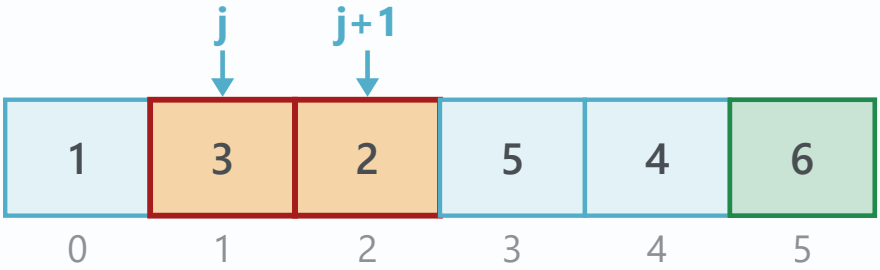
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]



2. tur, j = 0: 1 > 3? Hayır → yerinde kalır.

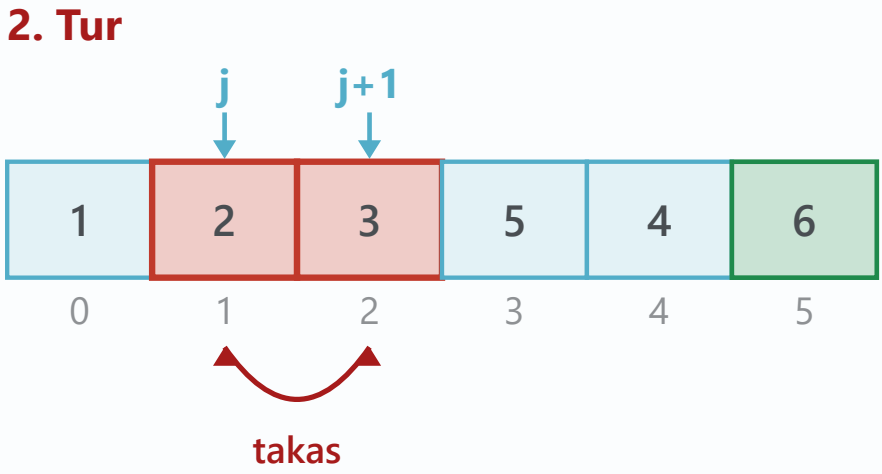
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

2. Tur



2. tur, j = 1: 3 > 2? **Evet** → yer değiştirilecek.

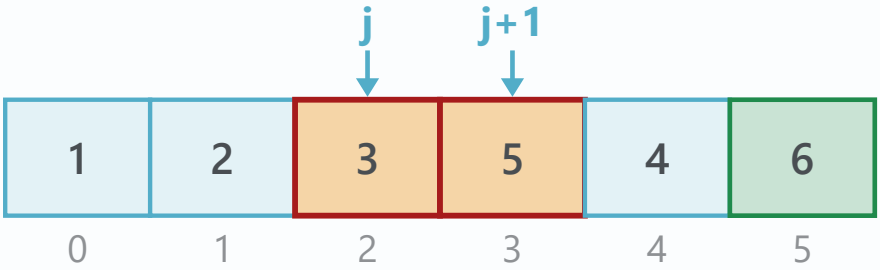
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]



Takas: dizi artık [1, 2, 3, 5, 4, 6].

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

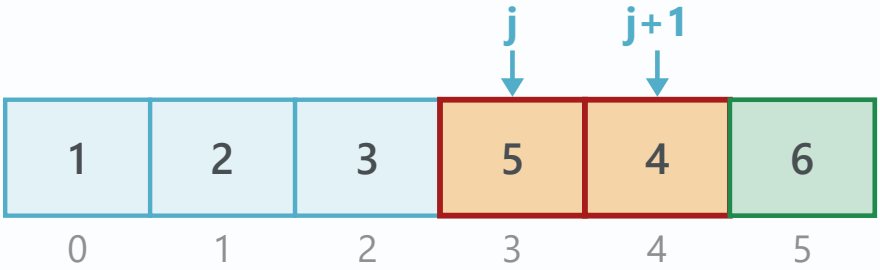
2. Tur



2. tur, j = 2: 3 > 5? Hayır → yerinde kalır.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

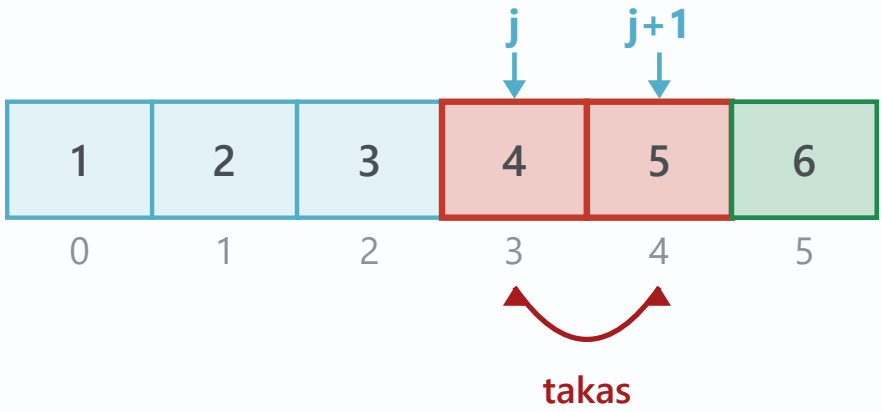
2. Tur



2. tur, j = 3: 5 > 4? **Evet** → yer değiştirilecek.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

2. Tur



Takas: dizi artık [1, 2, 3, 4, 5, 6].

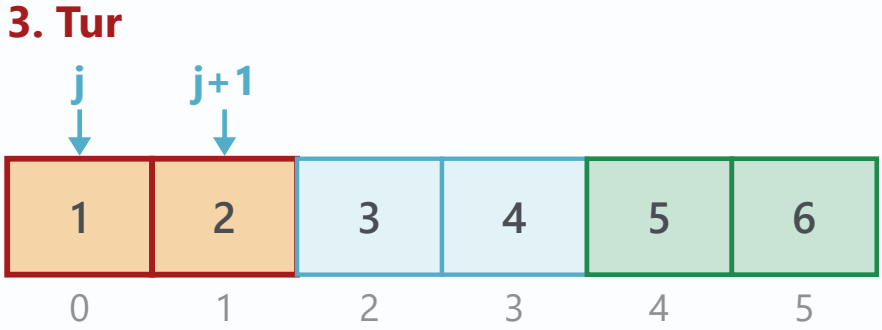
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

2. Tur sonu



2. tur bitti: en büyük kalan öğe **5** son yerine (indis 4) yerleşti.

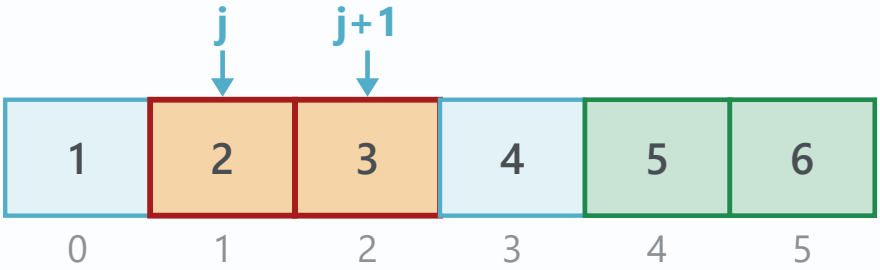
Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]



3. tur, j = 0: 1 > 2? Hayır → yerinde kalır.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

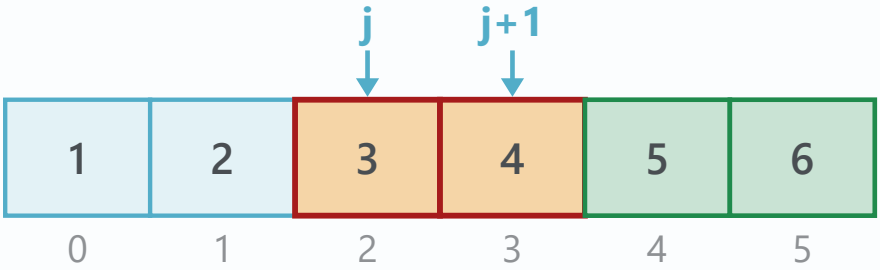
3. Tur



3. tur, j = 1: 2 > 3? Hayır → yerinde kalır.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

3. Tur



3. tur, j = 2: 3 > 4? Hayır → yerinde kalır.

Kabarcık Sıralaması Adım Adım: [3, 1, 5, 2, 6, 4]

3. Tur sonu



3. turda hiç takas olmadı → dizi **sıralı**; algoritma erken durur. Sonuç: [1, 2, 3, 4, 5, 6].

Kabarcık Sıralaması: Java Kodu

```java
boolean yerDegistiMi;
for(int i = 0; i < n - 1; i++) {
  yerDegistiMi = false;
  for(int j = 0; j < n - 1 - i; j++) {
    if(dizi[j] > dizi[j+1]) {
      int gecici = dizi[j];
      dizi[j] = dizi[j+1];
      dizi[j+1] = gecici;
      yerDegistiMi = true;
    }
  }
  if(yerDegistiMi == false) break;
}
```

- **Dış döngü** (i): en fazla n − 1 tur.
- **İç döngü** (j): sıralanmamış kısımdaki komşu çiftler; sondaki i öğe zaten yerinde olduğu için `j < n − 1 − i`.
- Takas `gecici` değişkeniyle üç atamada yapılır.
- `yerDegistiMi` bayrağı: tur boyunca hiç değiştirme yoksa dizi sıralıdır → `break`.

Koşul `>` olduğu (`≥` değil) için eşit öğeler asla yer değiştirmez: kabarcık sıralaması **kararlıdır**.

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
▶ boolean yerDegistiMi;
2 for(int i = 0; i < n - 1; i++) {
3   yerDegistiMi = false;
4   for(int j = 0; j < n - 1 - i; j++) {
5     if(dizi[j] > dizi[j+1]) {
6       int gecici = dizi[j];
7       dizi[j] = dizi[j+1];
8       dizi[j+1] = gecici;
9       yerDegistiMi = true;
10    }
11  }
12  if(yerDegistiMi == false) break;
13 }
```

dizi[]	5	1	9	2	10
	0	1	2	3	4

n = 5. `yerDegistiMi` bayrağı tanımlanır.

Adım 1 / 19

DEĞİŞKENLER

`n = 5` `i = –` `j = –` `yerDegistiMi = –`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
▶ for(int i = 0; i < n - 1; i++) {
▶   yerDegistiMi = false;
4   for(int j = 0; j < n - 1 - i; j++) {
5     if(dizi[j] > dizi[j+1]) {
6       int gecici = dizi[j];
7       dizi[j] = dizi[j+1];
8       dizi[j+1] = gecici;
9       yerDegistiMi = true;
10    }
11  }
12  if(yerDegistiMi == false) break;
13 }
```

dizi[]	5	1	9	2	10
	0	1	2	3	4

Dış döngü i = 0: bayrak `false` yapılır.

Adım 2 / 19

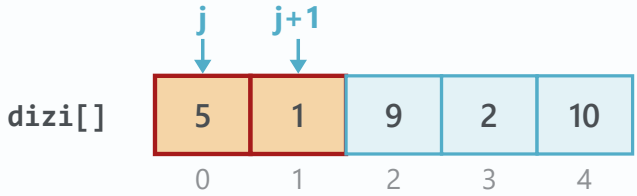
DEĞİŞKENLER

n = **5** | i = **0** | j = **–**

yerDegistiMi = **false**

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 0: dizi[0] = 5 > dizi[1] = 1? **Evet**.

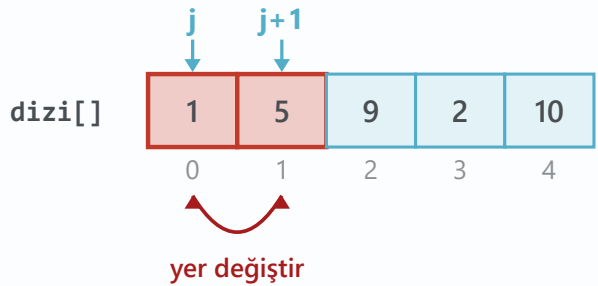
Adım 3 / 19

DEĞİŞKENLER

`n = 5` `i = 0` `j = 0` `yerDegistiMi = false`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
4    for(int j = 0; j < n - 1 - i; j++) {
5      if(dizi[j] > dizi[j+1]) {
▶        int gecici = dizi[j];
▶        dizi[j] = dizi[j+1];
▶        dizi[j+1] = gecici;
▶        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



`gecici` = 5 ile yer değiştirilir; bayrak `true`.

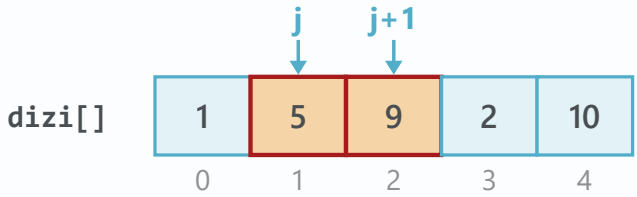
Adım 4 / 19

DEĞİŞKENLER

`n = 5` `i = 0` `j = 0` `gecici = 5` `yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 1: dizi[1] = 5 > dizi[2] = 9? Hayır.

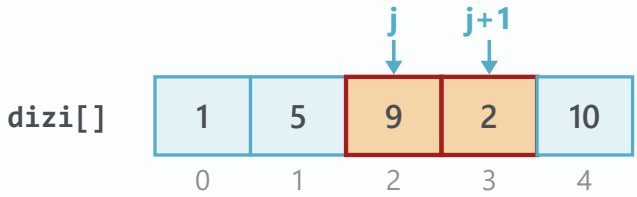
Adım 5 / 19

DEĞİŞKENLER

`n = 5` `i = 0` `j = 1` `yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 2: dizi[2] = 9 > dizi[3] = 2? **Evet**.

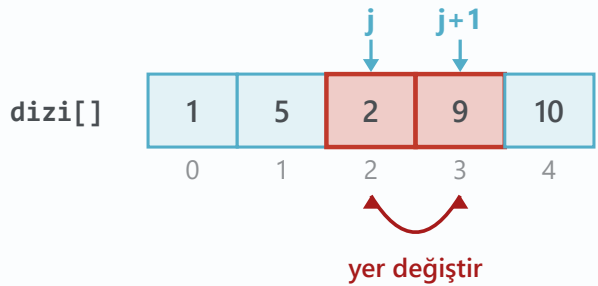
Adım 6 / 19

DEĞİŞKENLER

`n = 5` `i = 0` `j = 2` `yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
4    for(int j = 0; j < n - 1 - i; j++) {
5      if(dizi[j] > dizi[j+1]) {
         int gecici = dizi[j];
         dizi[j] = dizi[j+1];
         dizi[j+1] = gecici;
         yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



`gecici` = 9 ile yer değiştirilir; bayrak `true`.

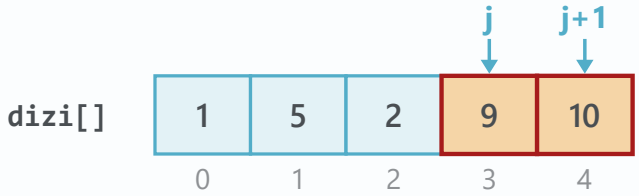
Adım 7 / 19

DEĞİŞKENLER

`n = 5` `i = 0` `j = 2` `gecici = 9` `yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 3: dizi[3] = 9 > dizi[4] = 10? Hayır.

Adım 8 / 19

DEĞİŞKENLER

`n = 5` `i = 0` `j = 3` `yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
4    for(int j = 0; j < n - 1 - i; j++) {
5      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
▶    if(yerDegistiMi == false) break;
13 }
```

dizi[]	1	5	2	9	10
	0	1	2	3	4

Tur sonu: 10 yerine oturdu. Bayrak true → devam.

Adım 9 / 19

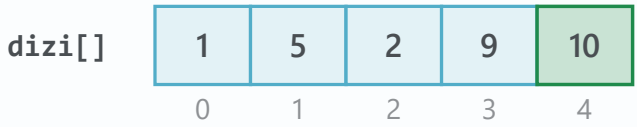
DEĞİŞKENLER

`n = 5` `i = 0` `j = –`

`yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
▶ for(int i = 0; i < n - 1; i++) {
▶   yerDegistiMi = false;
4   for(int j = 0; j < n - 1 - i; j++) {
5     if(dizi[j] > dizi[j+1]) {
6       int gecici = dizi[j];
7       dizi[j] = dizi[j+1];
8       dizi[j+1] = gecici;
9       yerDegistiMi = true;
10    }
11  }
12  if(yerDegistiMi == false) break;
13 }
```



Dış döngü i = 1: bayrak `false` yapılır.

Adım 10 / 19

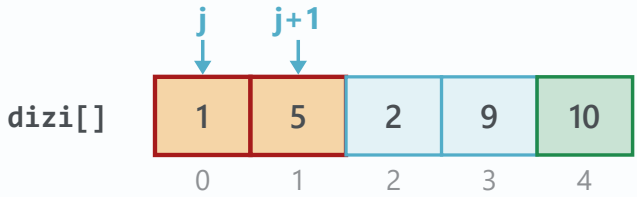
DEĞİŞKENLER

`n = 5` `i = 1` `j = –`

`yerDegistiMi = false`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 0: dizi[0] = 1 > dizi[1] = 5? Hayır.

Adım 11 / 19

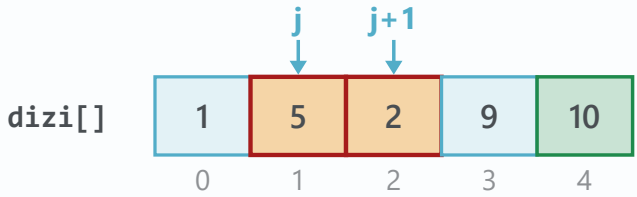
DEĞİŞKENLER

`n = 5` `i = 1` `j = 0`

`yerDegistiMi = false`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 1: dizi[1] = 5 > dizi[2] = 2? **Evet**.

Adım 12 / 19

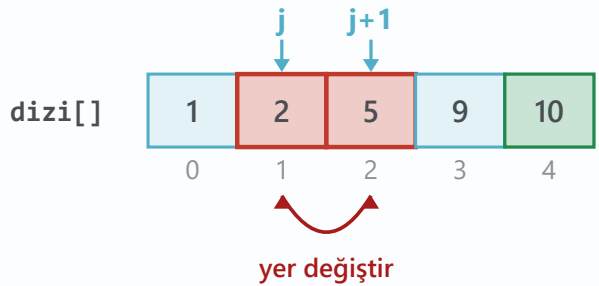
DEĞİŞKENLER

`n = 5` `i = 1` `j = 1`

`yerDegistiMi = false`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
4    for(int j = 0; j < n - 1 - i; j++) {
5      if(dizi[j] > dizi[j+1]) {
         int gecici = dizi[j];
         dizi[j] = dizi[j+1];
         dizi[j+1] = gecici;
         yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



`gecici` = 5 ile yer değiştirilir; bayrak `true`.

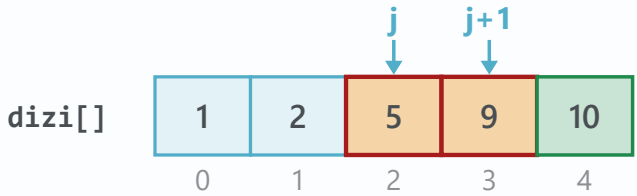
Adım 13 / 19

DEĞİŞKENLER

`n = 5` `i = 1` `j = 1` `gecici = 5` `yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 2: dizi[2] = 5 > dizi[3] = 9? Hayır.

Adım 14 / 19

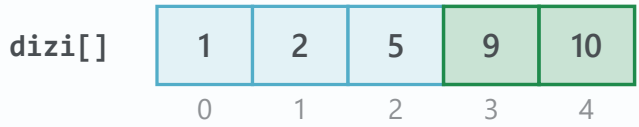
DEĞİŞKENLER

`n = 5` `i = 1` `j = 2`

`yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
4    for(int j = 0; j < n - 1 - i; j++) {
5      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
▶    if(yerDegistiMi == false) break;
13 }
```



Tur sonu: 9 yerine oturdu. Bayrak true → devam.

Adım 15 / 19

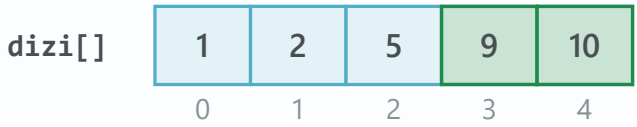
DEĞİŞKENLER

`n = 5` `i = 1` `j = –`

`yerDegistiMi = true`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
▶ for(int i = 0; i < n - 1; i++) {
▶   yerDegistiMi = false;
4   for(int j = 0; j < n - 1 - i; j++) {
5     if(dizi[j] > dizi[j+1]) {
6       int gecici = dizi[j];
7       dizi[j] = dizi[j+1];
8       dizi[j+1] = gecici;
9       yerDegistiMi = true;
10    }
11  }
12  if(yerDegistiMi == false) break;
13 }
```



Dış döngü i = 2: bayrak `false` yapılır.

Adım 16 / 19

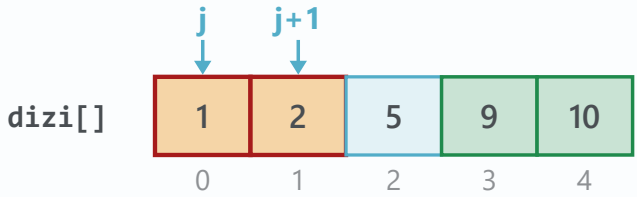
DEĞİŞKENLER

n = **5** | i = **2** | j = **–**

yerDegistiMi = **false**

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 0: dizi[0] = 1 > dizi[1] = 2? Hayır.

Adım 17 / 19

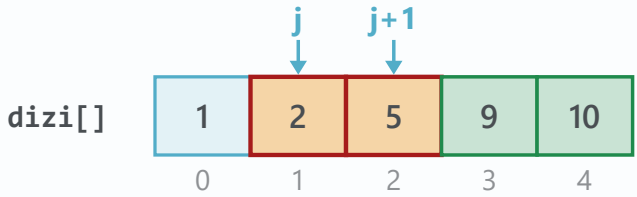
DEĞİŞKENLER

`n = 5` `i = 2` `j = 0`

`yerDegistiMi = false`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
▶    for(int j = 0; j < n - 1 - i; j++) {
▶      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
12   if(yerDegistiMi == false) break;
13 }
```



j = 1: dizi[1] = 2 > dizi[2] = 5? Hayır.

Adım 18 / 19

DEĞİŞKENLER

`n = 5` `i = 2` `j = 1`

`yerDegistiMi = false`

Kabarcık Sıralaması: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1  boolean yerDegistiMi;
2  for(int i = 0; i < n - 1; i++) {
3    yerDegistiMi = false;
4    for(int j = 0; j < n - 1 - i; j++) {
5      if(dizi[j] > dizi[j+1]) {
6        int gecici = dizi[j];
7        dizi[j] = dizi[j+1];
8        dizi[j+1] = gecici;
9        yerDegistiMi = true;
10     }
11   }
▶    if(yerDegistiMi == false) break;
13 }
```

dizi[]	1	2	5	9	10
	0	1	2	3	4

Bu turda takas yok: `break`. Dizi sıralı: [1, 2, 5, 9, 10].

Adım 19 / 19

DEĞİŞKENLER

`n = 5` `i = 2` `j = –`

`yerDegistiMi = false`

Kabarcık Sıralaması: Analiz

Durum	Girdi	Karşılaştırma	Takas	Zaman
En iyi	sıralı	$n - 1$ (tek tur, bayrakla)	0	$\Theta(n)$
Ortalama	rastgele	$\approx n^2/2$	$\approx n^2/4$	$\Theta(n^2)$
En kötü	ters sıralı	$n(n - 1)/2$	$n(n - 1)/2$	$\Theta(n^2)$

Artıları

+ Anlaşılması ve yazılması çok kolay.
+ Yerinde ($O(1)$ ek bellek) ve kararlı.
+ Bayrakla sıralı girdiyi $\Theta(n)$'de tanır.

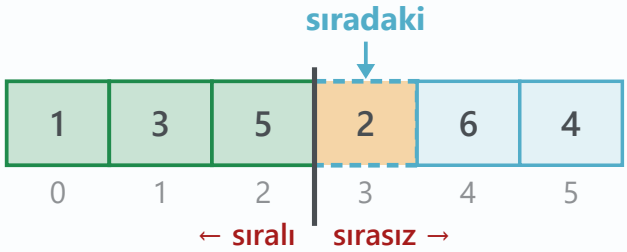
Eksileri

– $\Theta(n^2)$: büyük dizilerde çok yavaş.
– Çok fazla takas yapar (her ters çift = 1 takas).
– Pratikte eklemeli sıralamadan hep yavaştır.

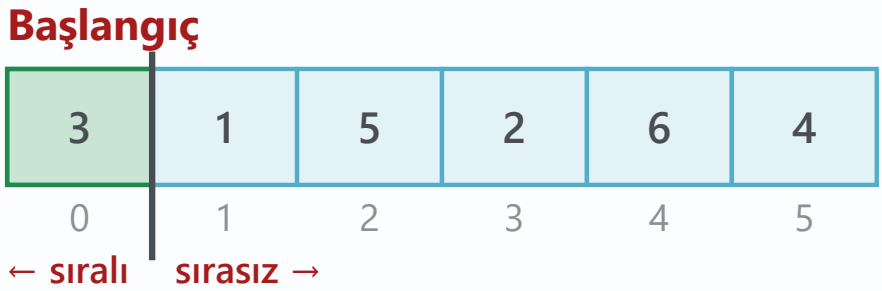
Ters çift (inversion): $i < j$ iken $dizi[i] > dizi[j]$. Kabarcık ve eklemeli sıralama her adımda tam olarak bir ters çifti düzeltir; ters sıralı dizide $n(n - 1)/2$ ters çift vardır.

Eklemeli Sıralama (Insertion Sort)

- Eldeki **oyun kartlarını** sıralamaya benzer: her yeni kart, eldeki sıralı kartların arasında doğru yere yerleştirilir.
- Dizi iki parçaya ayrılır: **sıralı kısım** (solda) ve **sırasız kısım**.
- Sırasız kısmın ilk öğesi alınır, sıralı kısımda **doğru konumuna** yerleştirilir.
- Bunun için ondan büyük öğelerin bir sağa **kaydırılması** gerekir.
- **Değişmez** (invariant): her dış adımın başında `dizi[0..i−1]` sıralıdır.

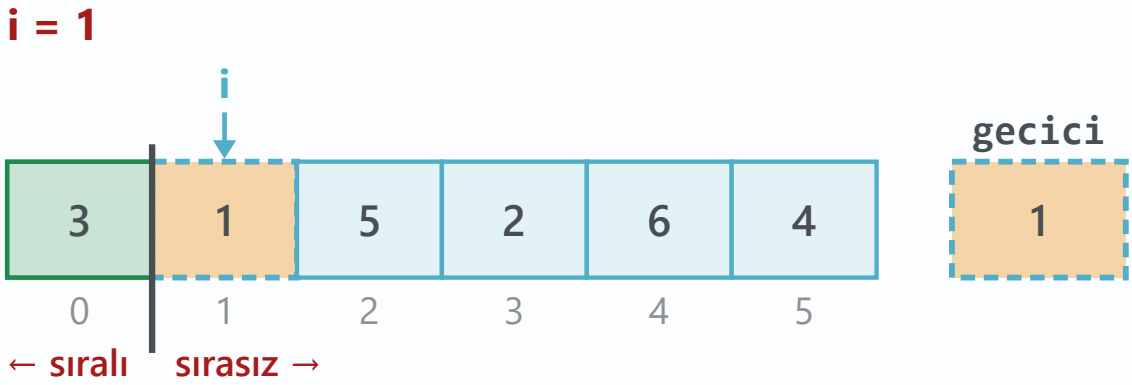


Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



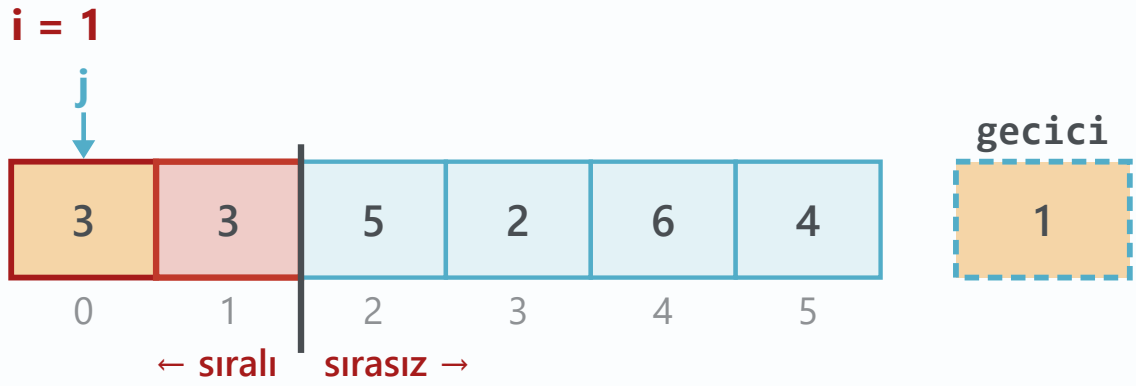
Dizi [3, 1, 5, 2, 6, 4]. İlk öğe tek başına **sıralı kısımdır**; geri kalanı sırasız.

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



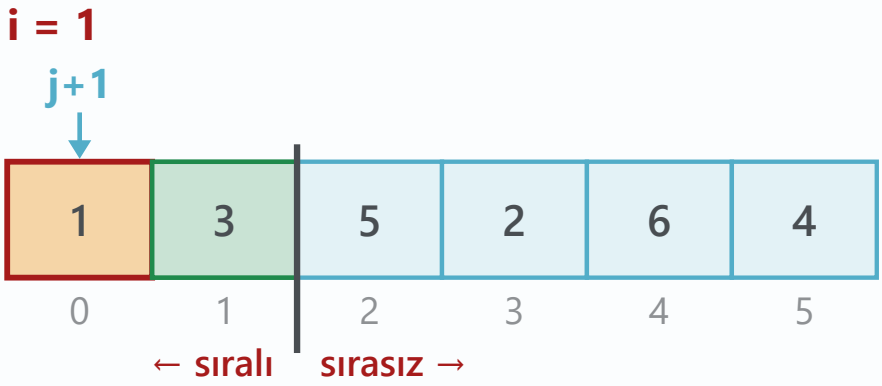
Sırasız kısmın ilk öğesi alınır: `gecici` = 1.

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



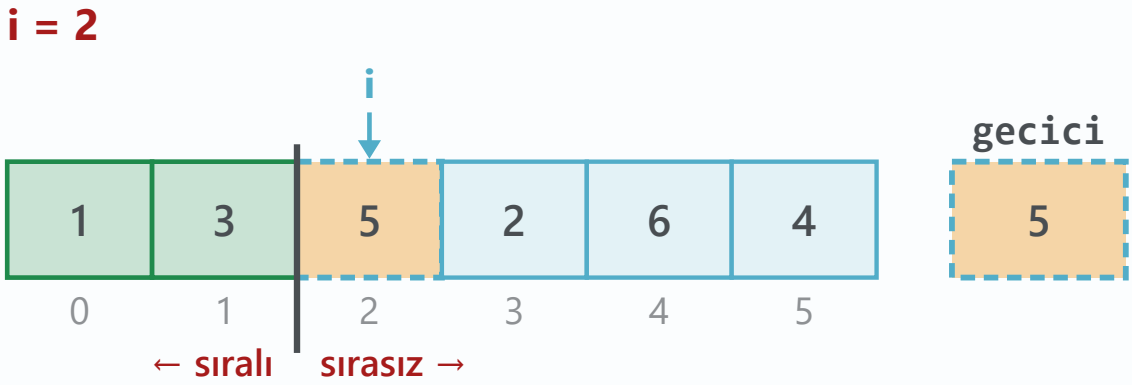
3 > 1 → 3 bir sağa **kaydırılır** (indis 0 → 1).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



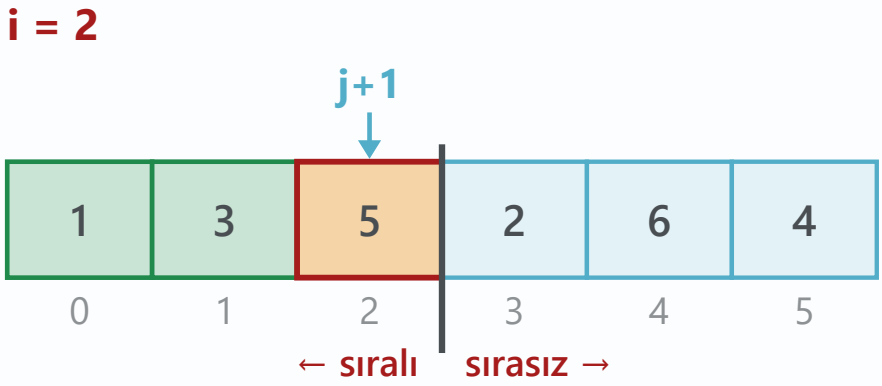
Dizinin başına gelindi → 1 indis 0'a **yerleştirilir** (1 kaydırma).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



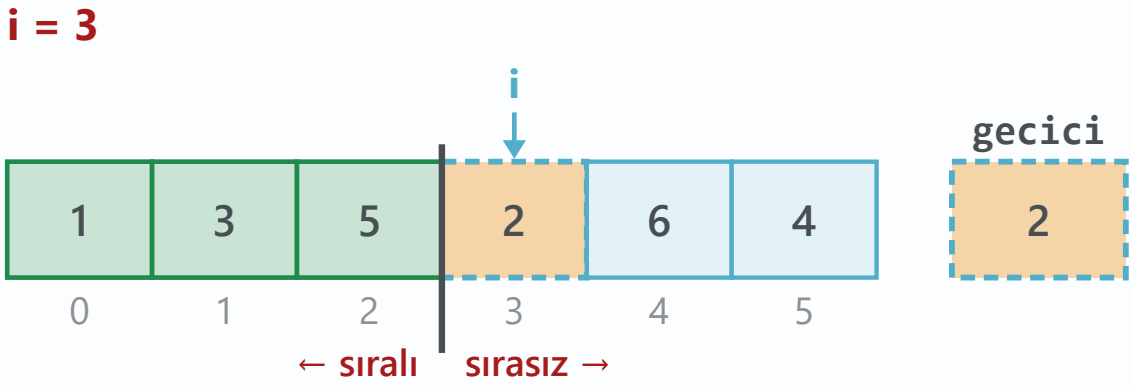
Sırasız kısmın ilk öğesi alınır: `gecici` = 5.

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



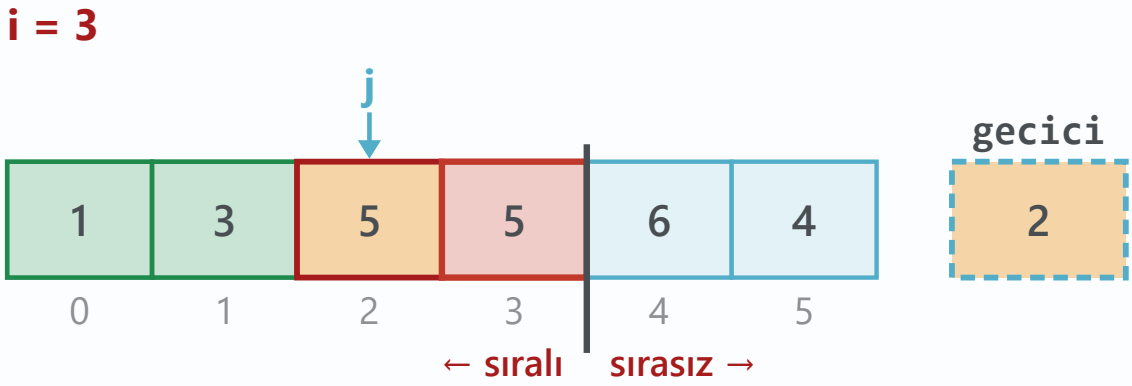
3 ≤ 5 → 5 indis 2'ye **yerleştirilir** (kaydırma gerekmedi).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



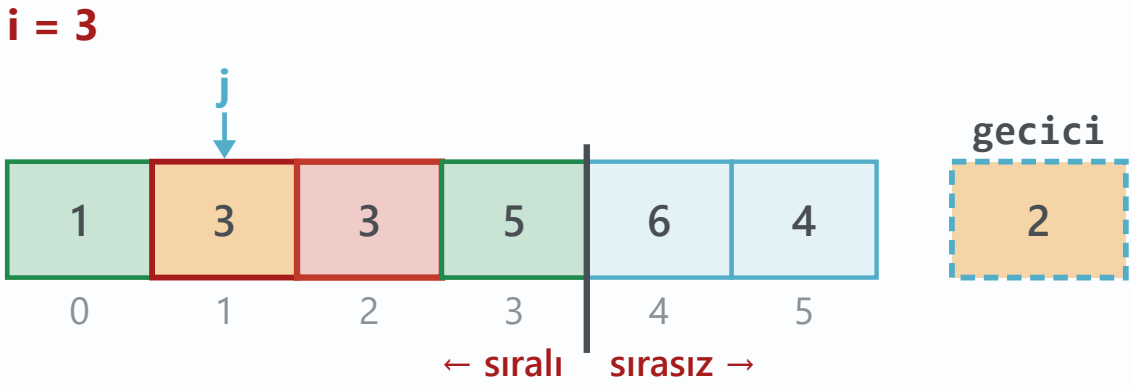
Sırasız kısmın ilk öğesi alınır: `gecici` = 2.

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



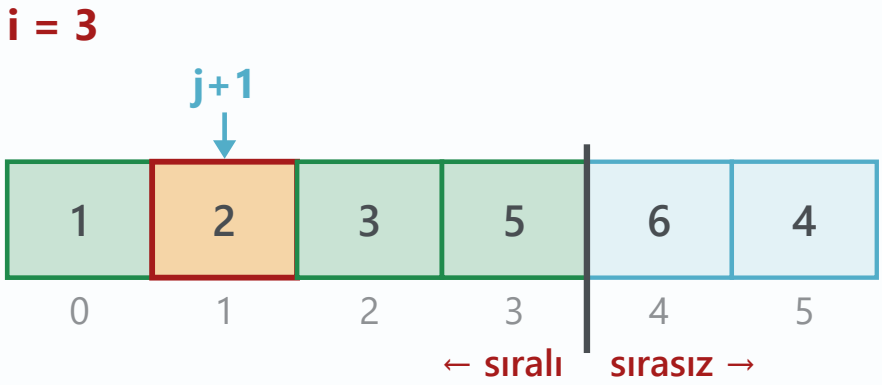
5 > 2 → 5 bir sağa **kaydırılır** (indis 2 → 3).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



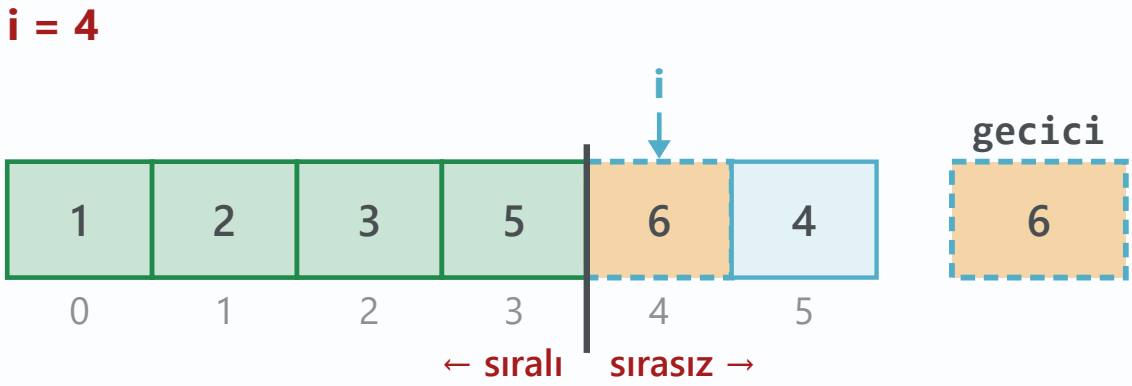
3 > 2 → 3 bir sağa **kaydırılır** (indis 1 → 2).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



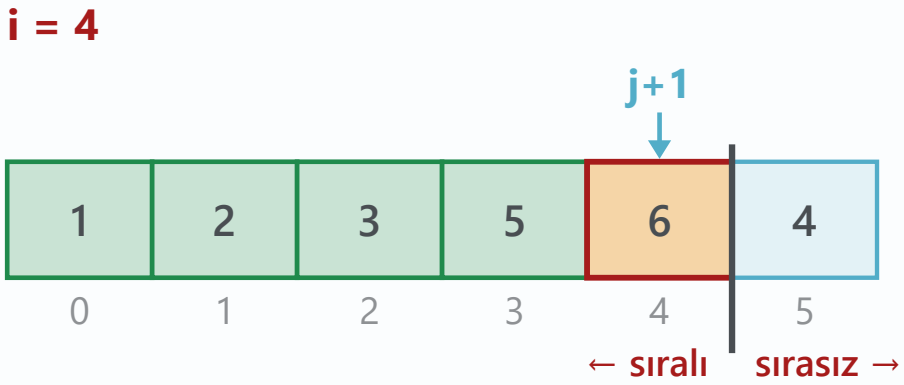
1 ≤ 2 → 2 indis 1'e **yerleştirilir** (2 kaydırma).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



Sırasız kısmın ilk öğesi alınır: `gecici` = 6.

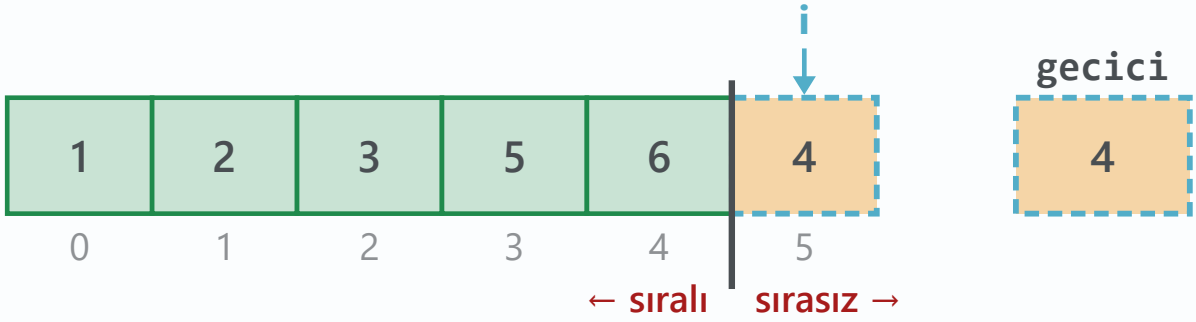
Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



5 ≤ 6 → 6 indis 4'e **yerleştirilir** (kaydırma gerekmedi).

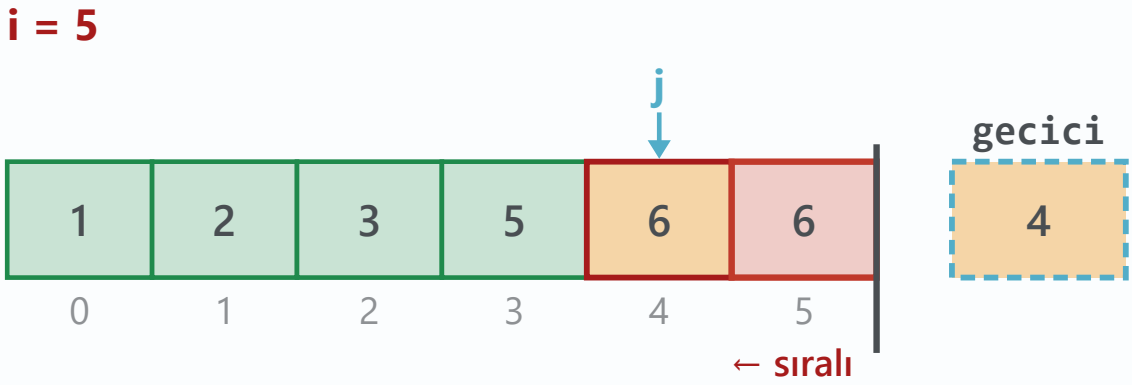
Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

i = 5



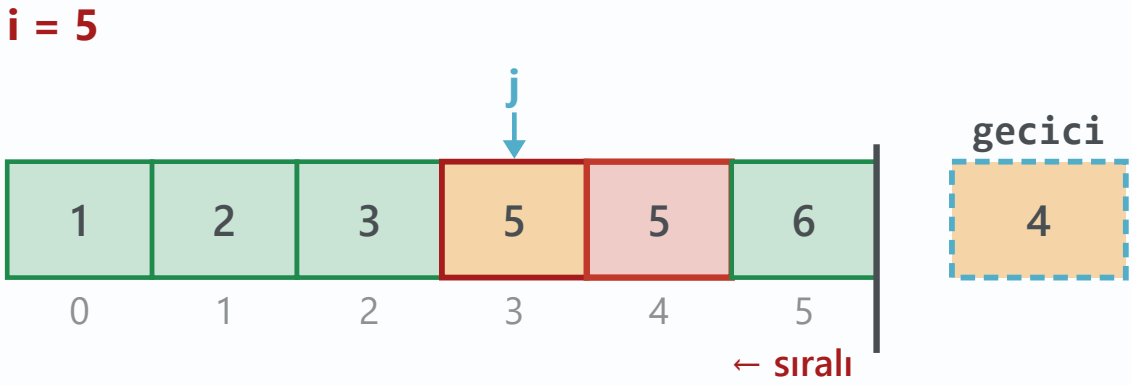
Sırasız kısmın ilk öğesi alınır: `gecici` = 4.

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



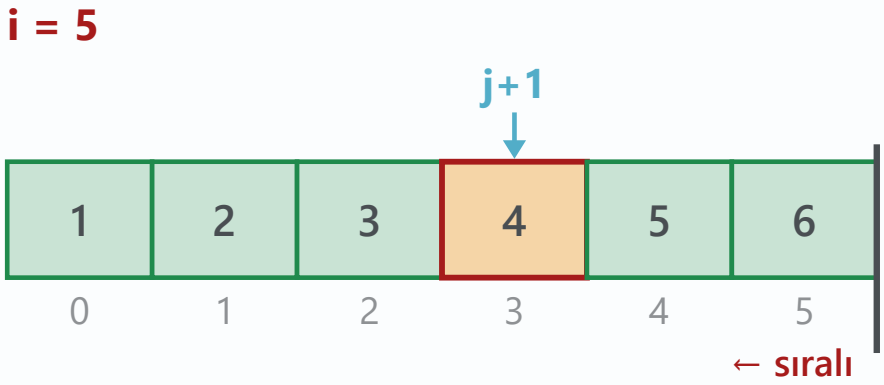
6 > 4 → 6 bir sağa **kaydırılır** (indis 4 → 5).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



5 > 4 → 5 bir sağa **kaydırılır** (indis 3 → 4).

Eklemeli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



3 ≤ 4 → 4 indis 3'e **yerleştirilir** (2 kaydırma). Dizi sıralı: [1, 2, 3, 4, 5, 6].

Eklemeli Sıralama: Java Kodu

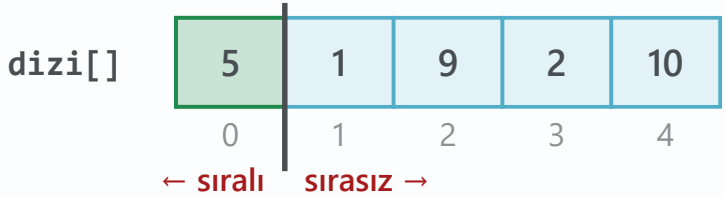
```
for(int i = 1; i < n; i++) {
  int gecici = dizi[i];
  int j = i - 1;
  while(j >= 0 && dizi[j] > gecici) {
    dizi[j+1] = dizi[j];
    j = j - 1;
  }
  dizi[j+1] = gecici;
}
```

- i = 1'den başlanır: tek öğeli `dizi[0..0]` zaten sıralıdır.
- `gecici` : yerleştirilecek öğe (anahtar). Kaydırmalar onun hücresinin üzerine yazacağı için önce saklanır.
- `while` : soldaki öğe büyük olduğu sürece bir sağa kaydırılır.
- Döngü bitince boşalan yer `j + 1` 'dir.

Takas (swap) değil **kaydırma** yapılır: her adımda 3 yerine 1 atama. Bu yüzden eklemeli sıralama kabarcıktan belirgin biçimde hızlıdır.

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



n = 5. Sıralı kısım: yalnızca dizi[0].

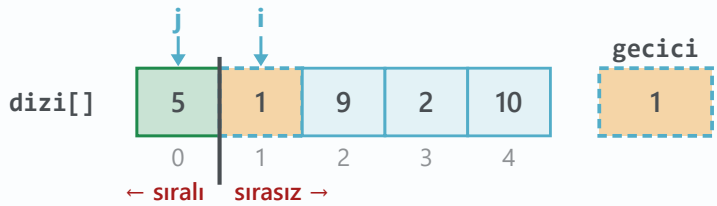
Adım 1 / 20

DEĞİŞKENLER

n = 5 | i = – | j = – | gecici = –

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
for(int i = 1; i < n; i++) {
  int gecici = dizi[i];
  int j = i - 1;
  while(j >= 0 && dizi[j] > gecici) {
    dizi[j+1] = dizi[j];
    j = j - 1;
  }
  dizi[j+1] = gecici;
}
```



i = 1: `gecici` = dizi[1] = 1, j = 0.

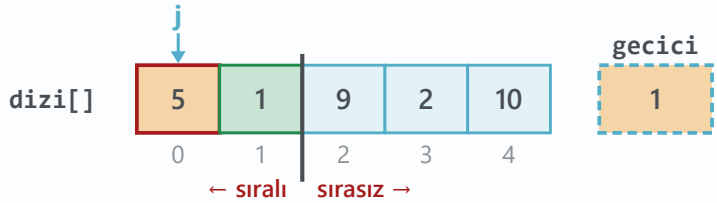
Adım 2 / 20

DEĞİŞKENLER

`n = 5` `i = 1` `j = 0` `gecici = 1`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



j = 0 ≥ 0 ve dizi[0] = 5 > 1 → döngüye gir.

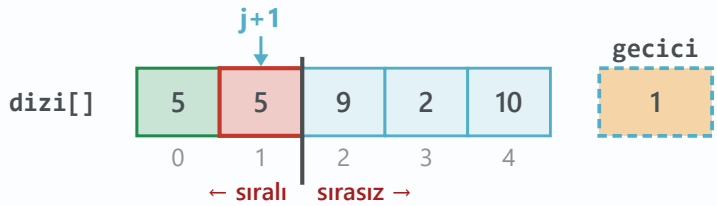
Adım 3 / 20

DEĞİŞKENLER

n = 5 | i = 1 | j = 0 | gecici = 1

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
      dizi[j+1] = dizi[j];
      j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



dizi[1] = dizi[0] = 5 (sağa kaydır), j = −1.

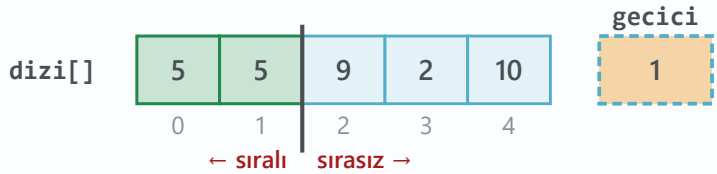
Adım 4 / 20

DEĞİŞKENLER

n = 5 | i = 1 | j = −1 | gecici = 1

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



j = −1: döngü biter.

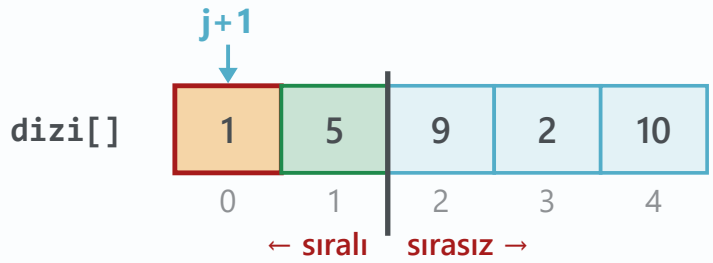
Adım 5 / 20

DEĞİŞKENLER

`n = 5` `i = 1` `j = -1` `gecici = 1`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
▶   dizi[j+1] = gecici;
9 }
```



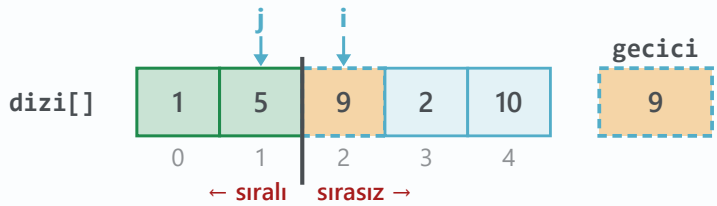
dizi[0] = gecici = 1. Sıralı kısım: [1, 5].

DEĞİŞKENLER

`n = 5` `i = 1` `j = -1` `gecici = 1`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
for(int i = 1; i < n; i++) {
  int gecici = dizi[i];
  int j = i - 1;
  while(j >= 0 && dizi[j] > gecici) {
    dizi[j+1] = dizi[j];
    j = j - 1;
  }
  dizi[j+1] = gecici;
}
```



i = 2: `gecici` = dizi[2] = 9, j = 1.

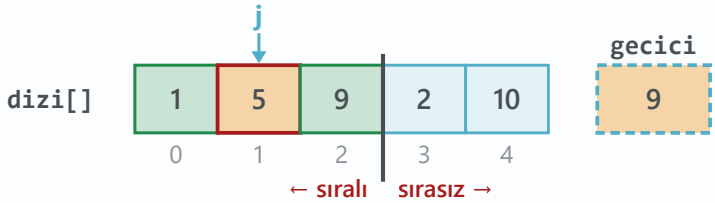
Adım 7 / 20

DEĞİŞKENLER

`n = 5` `i = 2` `j = 1` `gecici = 9`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



dizi[1] = 5 ≤ 9: döngü biter.

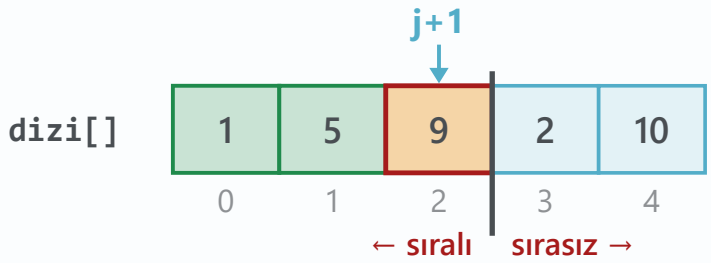
Adım 8 / 20

DEĞİŞKENLER

`n = 5` `i = 2` `j = 1` `gecici = 9`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
▶   dizi[j+1] = gecici;
9 }
```



dizi[2] = gecici = 9. Sıralı kısım: [1, 5, 9].

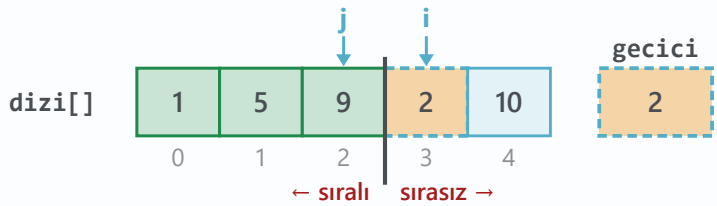
Adım 9 / 20

DEĞİŞKENLER

`n = 5` `i = 2` `j = 1` `gecici = 9`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
for(int i = 1; i < n; i++) {
  int gecici = dizi[i];
  int j = i - 1;
  while(j >= 0 && dizi[j] > gecici) {
    dizi[j+1] = dizi[j];
    j = j - 1;
  }
  dizi[j+1] = gecici;
}
```



i = 3: `gecici` = dizi[3] = 2, j = 2.

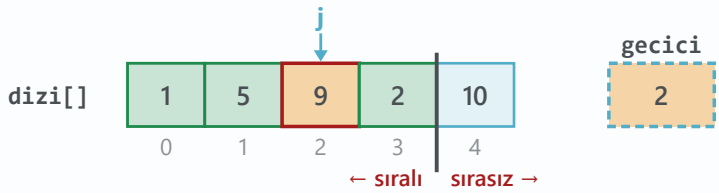
Adım 10 / 20

DEĞİŞKENLER

n = 5 | i = 3 | j = 2 | gecici = 2

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



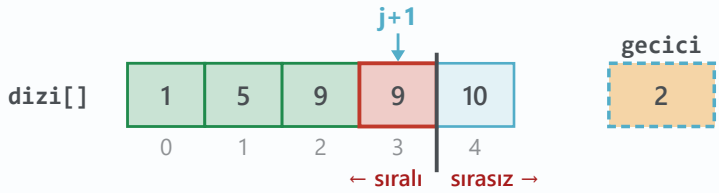
j = 2 ≥ 0 ve dizi[2] = 9 > 2 → döngüye gir.

DEĞİŞKENLER

`n = 5` `i = 3` `j = 2` `gecici = 2`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
      dizi[j+1] = dizi[j];
      j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



dizi[3] = dizi[2] = 9 (sağa kaydır), j = 1.

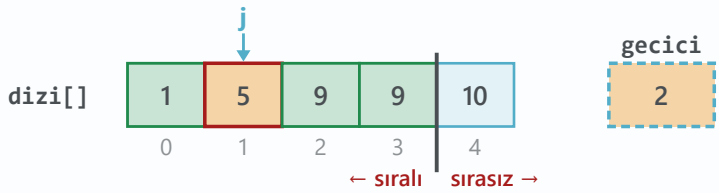
Adım 12 / 20

DEĞİŞKENLER

n = 5 | i = 3 | j = 1 | gecici = 2

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



j = 1 ≥ 0 ve dizi[1] = 5 > 2 → döngüye gir.

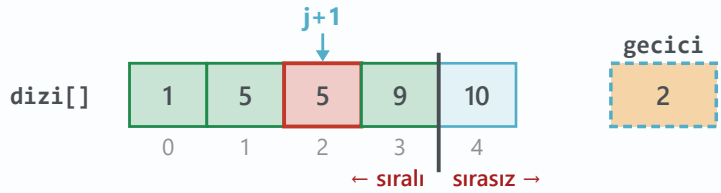
Adım 13 / 20

DEĞİŞKENLER

`n = 5` `i = 3` `j = 1` `gecici = 2`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
      dizi[j+1] = dizi[j];
      j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



dizi[2] = dizi[1] = 5 (sağa kaydır), j = 0.

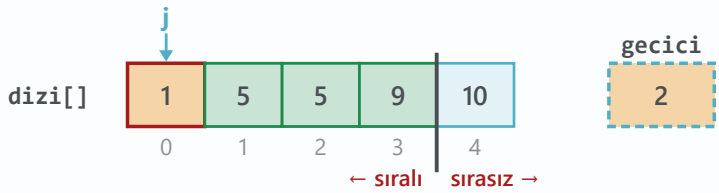
Adım 14 / 20

DEĞİŞKENLER

n = 5 | i = 3 | j = 0 | gecici = 2

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



dizi[0] = 1 ≤ 2: döngü biter.

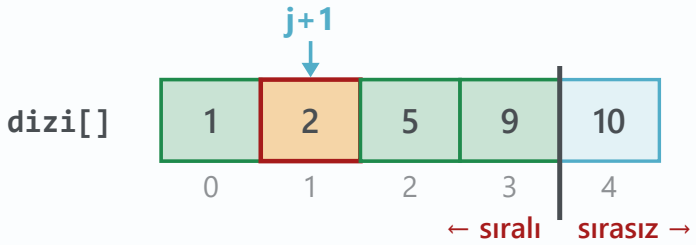
Adım 15 / 20

DEĞİŞKENLER

`n = 5` `i = 3` `j = 0` `gecici = 2`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
▶   dizi[j+1] = gecici;
9 }
```



dizi[1] = gecici = 2. Sıralı kısım: [1, 2, 5, 9].

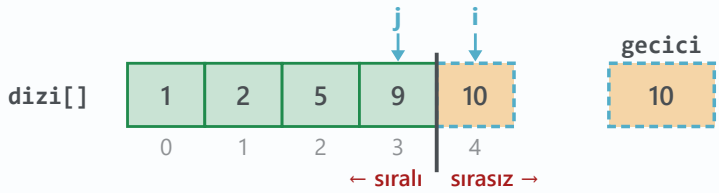
Adım 16 / 20

DEĞİŞKENLER

`n = 5` `i = 3` `j = 0` `gecici = 2`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
for(int i = 1; i < n; i++) {
  int gecici = dizi[i];
  int j = i - 1;
  while(j >= 0 && dizi[j] > gecici) {
    dizi[j+1] = dizi[j];
    j = j - 1;
  }
  dizi[j+1] = gecici;
}
```



i = 4: `gecici` = dizi[4] = 10, j = 3.

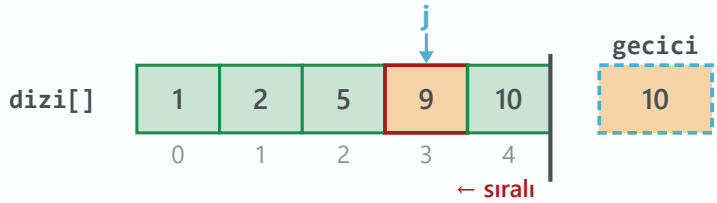
Adım 17 / 20

DEĞİŞKENLER

n = 5 | i = 4 | j = 3 | gecici = 10

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
▶   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
8   dizi[j+1] = gecici;
9 }
```



dizi[3] = 9 ≤ 10: döngü biter.

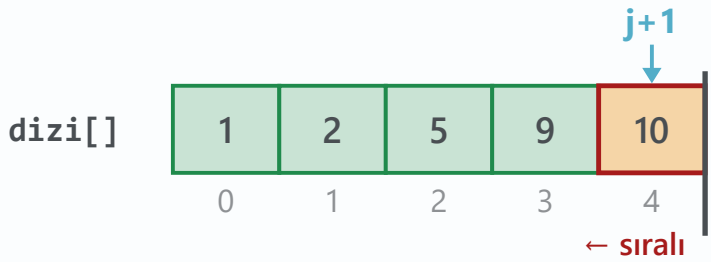
Adım 18 / 20

DEĞİŞKENLER

`n = 5` `i = 4` `j = 3` `gecici = 10`

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
1 for(int i = 1; i < n; i++) {
2   int gecici = dizi[i];
3   int j = i - 1;
4   while(j >= 0 && dizi[j] > gecici) {
5     dizi[j+1] = dizi[j];
6     j = j - 1;
7   }
▶   dizi[j+1] = gecici;
9 }
```



dizi[4] = gecici = 10. Sıralı kısım: [1, 2, 5, 9, 10].

Adım 19 / 20

DEĞİŞKENLER

n = 5 | i = 4 | j = 3 | gecici = 10

Eklemeli Sıralama: Kodu İzle, dizi = [5, 1, 9, 2, 10]

```
for(int i = 1; i < n; i++) {
  int gecici = dizi[i];
  int j = i - 1;
  while(j >= 0 && dizi[j] > gecici) {
    dizi[j+1] = dizi[j];
    j = j - 1;
  }
  dizi[j+1] = gecici;
}
```

dizi[]

1	2	5	9	10
0	1	2	3	4

i = 5 < n değil → döngü biter. Sonuç: [1, 2, 5, 9, 10].

Adım 20 / 20

DEĞİŞKENLER

`n = 5` `i = 5` `j = –` `gecici = –`

Eklemeli Sıralama: Analiz

Durum	Girdi	Karşılaştırma	Kaydırma	Zaman
En iyi	sıralı	$n - 1$	0	$\Theta(n)$
Ortalama	rastgele	$\approx n^2/4$	$\approx n^2/4$	$\Theta(n^2)$
En kötü	ters sıralı	$n(n-1)/2$	$n(n-1)/2$	$\Theta(n^2)$

Artıları

+ Kararlı, yerinde, çevrim içi.
+ Uyarlanabilir: süre $\Theta(n + \text{ters çift sayısı})$.
+ Küçük dizilerde ($n \lesssim 20$–50) en hızlılardan.

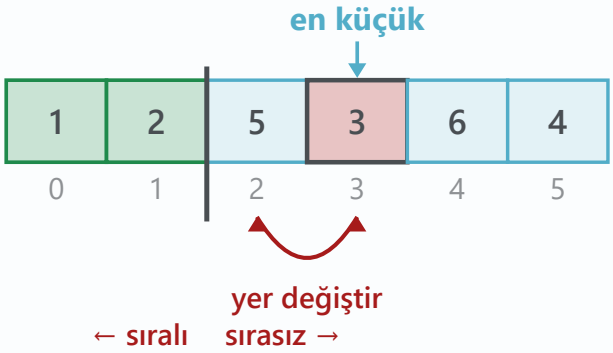
Eksileri

– Rastgele büyük dizilerde $\Theta(n^2)$.
– Bir öğe tek adımda yalnızca bir hücre ilerler; uzaktaki öğe çok kaydırma ister (→ Shell sıralaması).

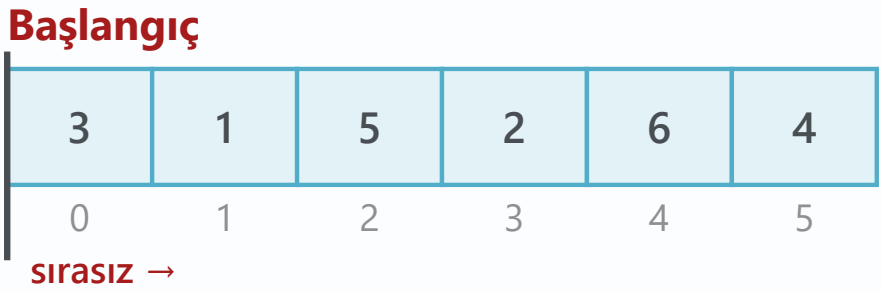
Tüm girdiler için yalnızca $O(n^2)$ ve $\Omega(n)$ yazılabilir; tek bir Θ yoktur. Timsort ve birçok kütüphane, hızlı sıralamada ve birleştirme sıralamasında küçük alt dizileri eklemeli sıralamaya bırakır.

Seçimli Sıralama (Selection Sort)

- Dizi iki bölüme ayrılır: **sıralı** ve **sırasız**.
- Her adımda:
  - Sırasız bölümden **en küçük** öğe bulunur.
  - Sırasız bölümün **başındaki** öğe ile yer değiştirilir.
  - Takasın ardından sıralı bölümün parçası olur.
- Toplam n − 1 tur; her tur **en fazla bir** takas.

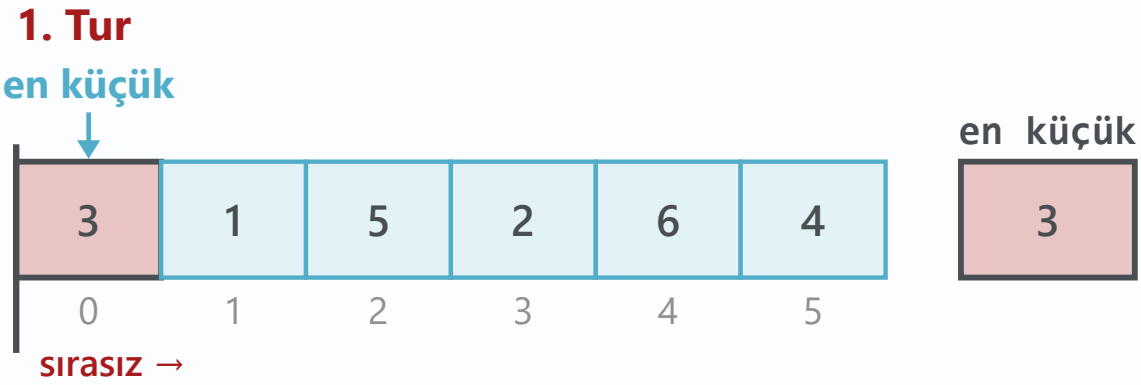


Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



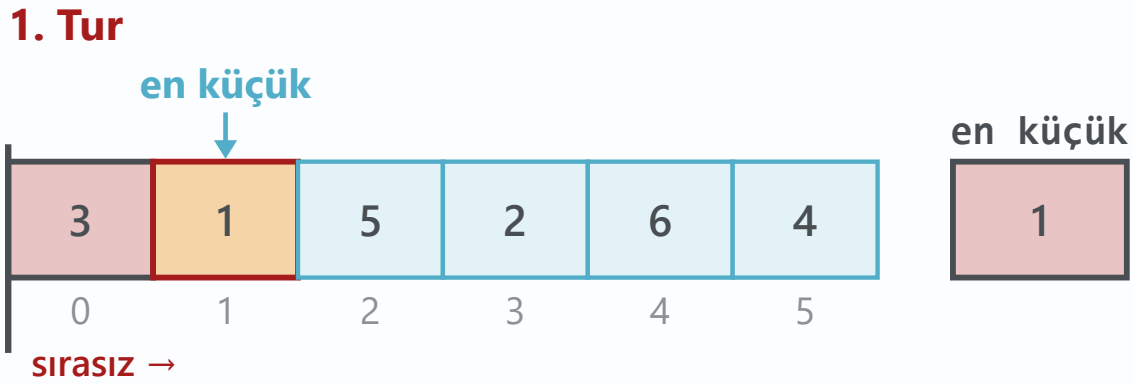
Dizi [3, 1, 5, 2, 6, 4], uzunluk = 6. Başta sıralı kısım boştur.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



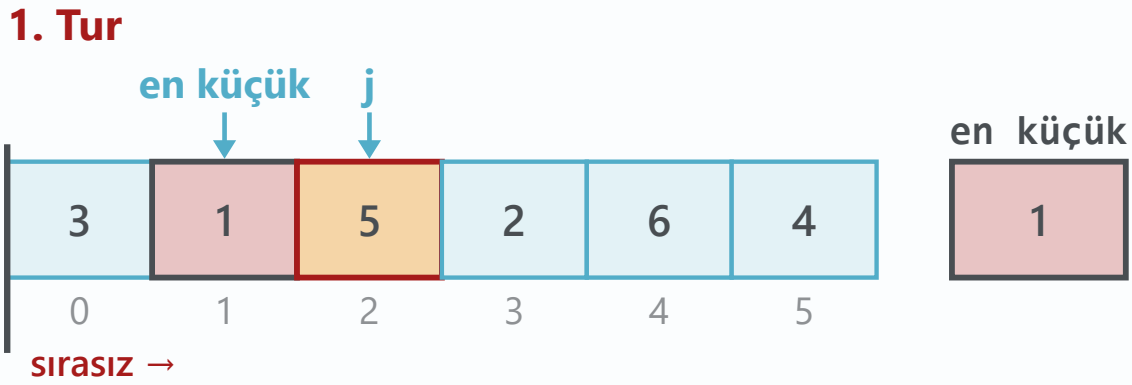
1. tur: sırasız kısmın ilk öğesi geçici en küçük kabul edilir: en küçük = 3.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



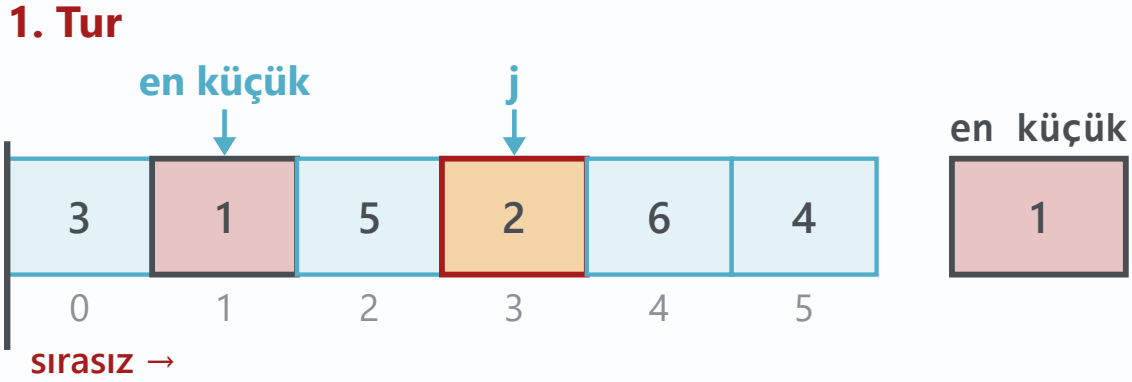
1 < en küçük (3)? **Evet** → en küçük = 1.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



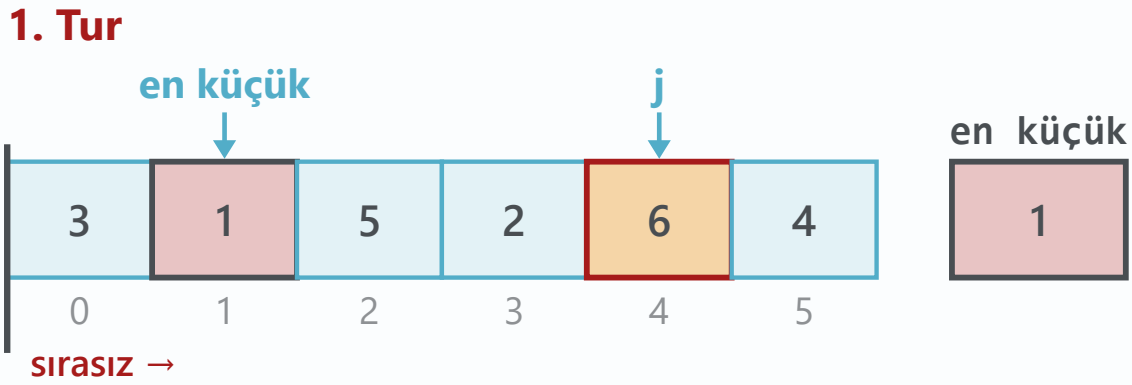
5 < en küçük (1)? Hayır.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



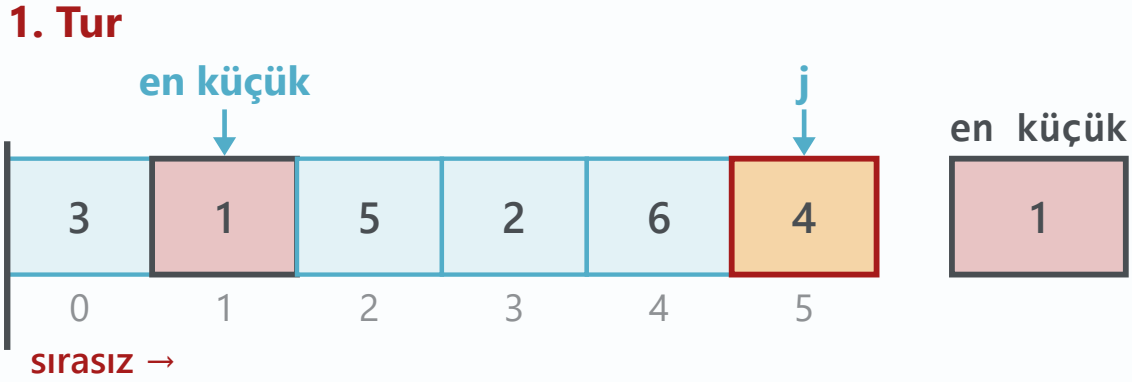
2 < en küçük (1)? Hayır.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



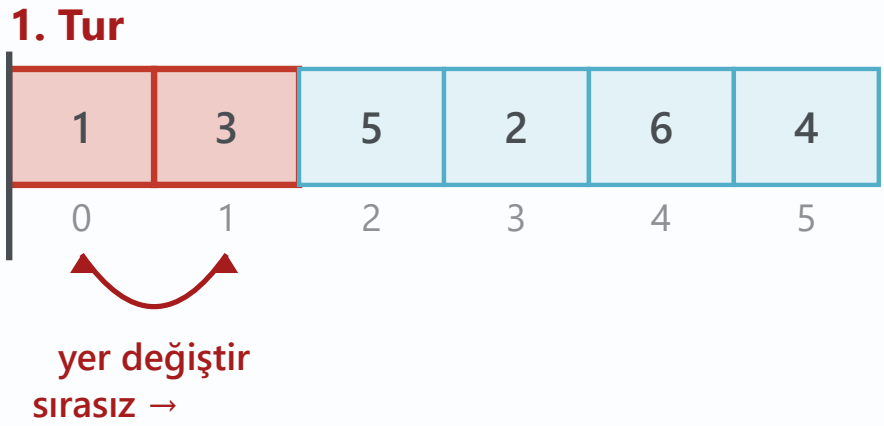
6 < en küçük (1)? Hayır.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



4 < en küçük (1)? Hayır.

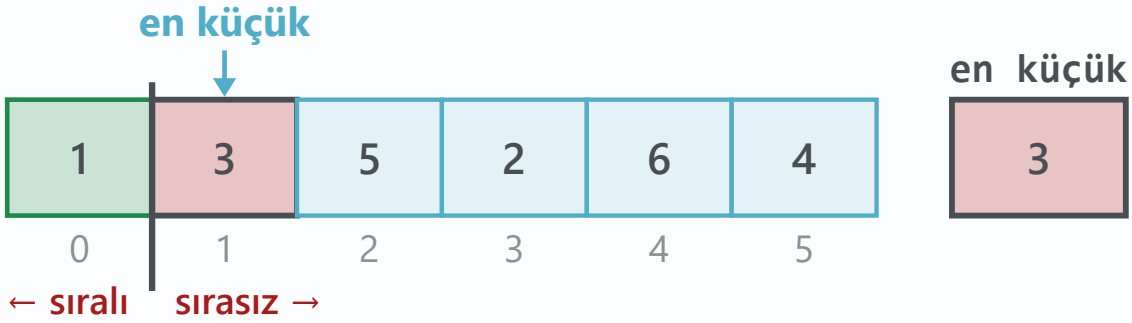
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



En küçük (1) sırasız kısmın başıyla yer değiştirir → [1, 3, 5, 2, 6, 4]; sıralı kısım bir büyür.

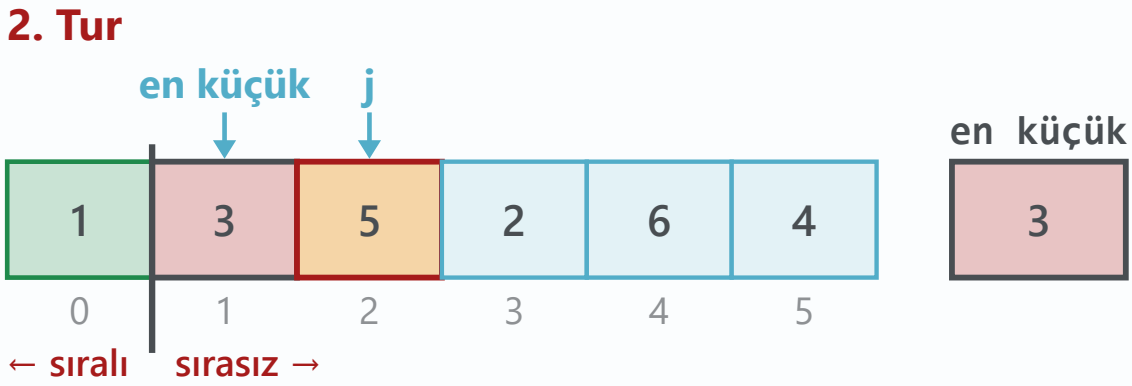
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

2. Tur



2. tur: sırasız kısmın ilk öğesi geçici en küçük kabul edilir: en küçük = 3.

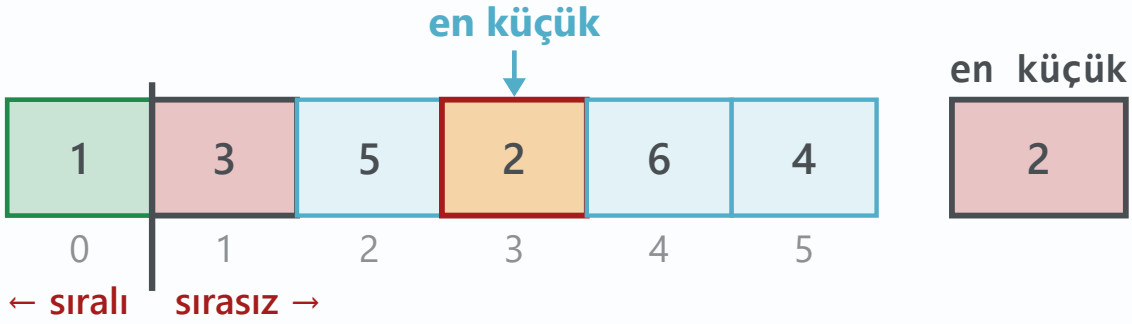
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



5 < en küçük (3)? Hayır.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

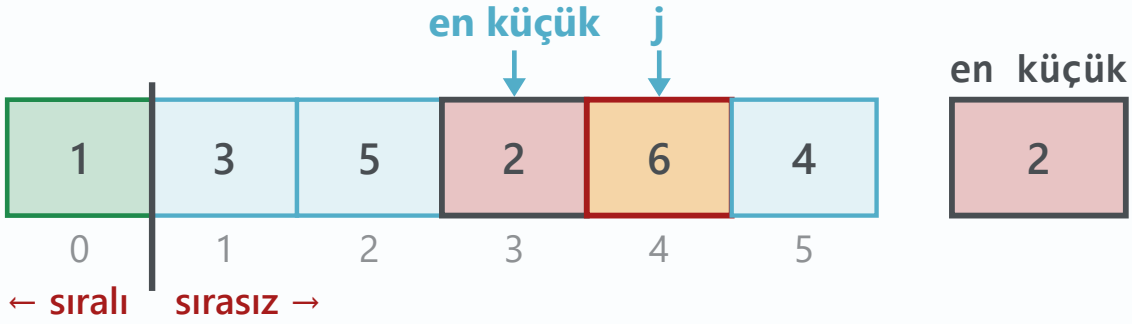
2. Tur



2 < en küçük (3)? **Evet** → en küçük = 2.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

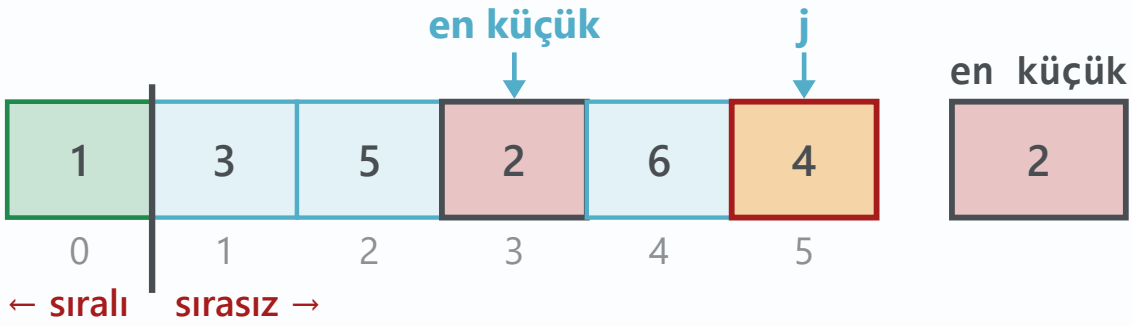
2. Tur



6 < en küçük (2)? Hayır.

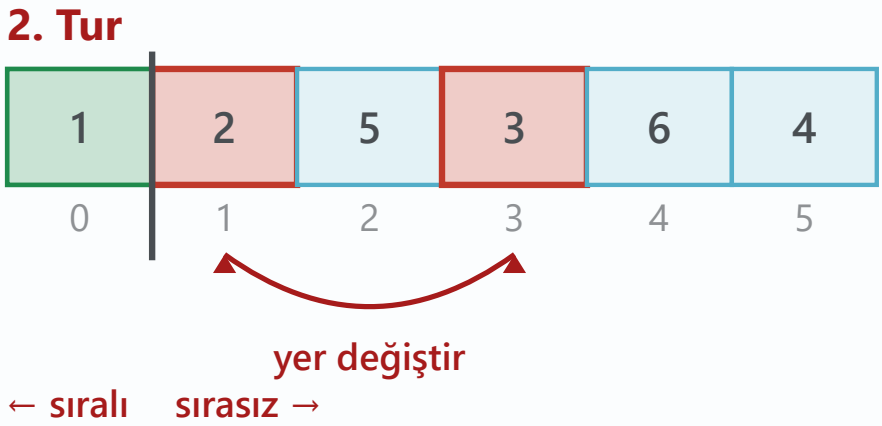
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

2. Tur



4 < en küçük (2)? Hayır.

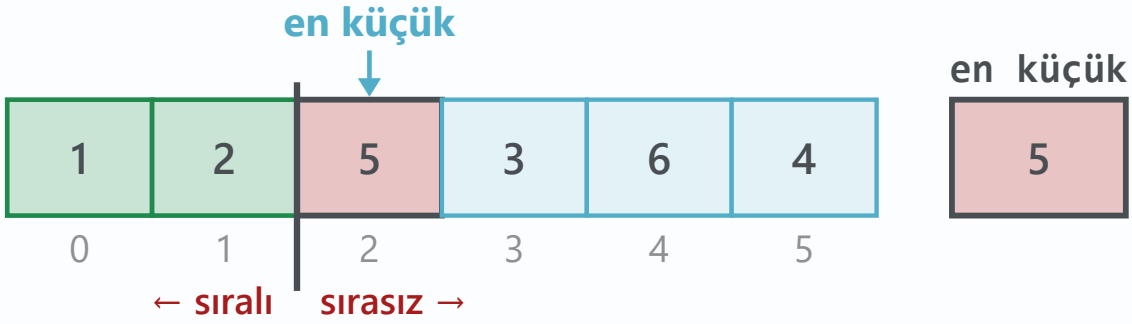
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



En küçük (2) sırasız kısmın başıyla yer değiştirir → [1, 2, 5, 3, 6, 4]; sıralı kısım bir büyür.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

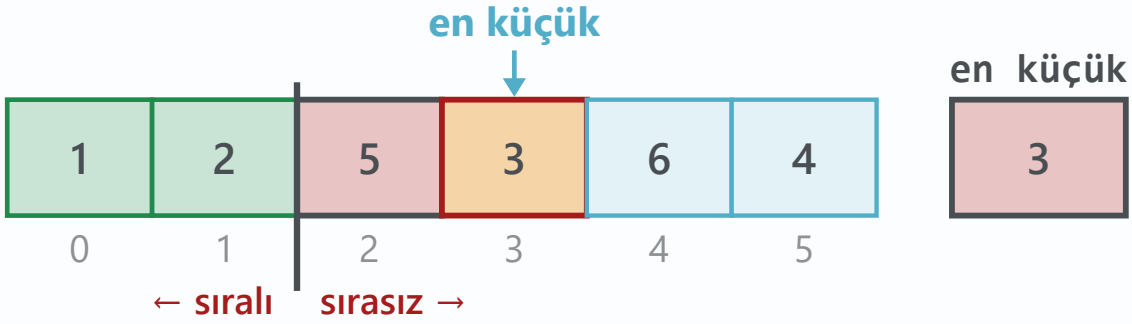
3. Tur



3. tur: sırasız kısmın ilk öğesi geçici en küçük kabul edilir: en küçük = 5.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

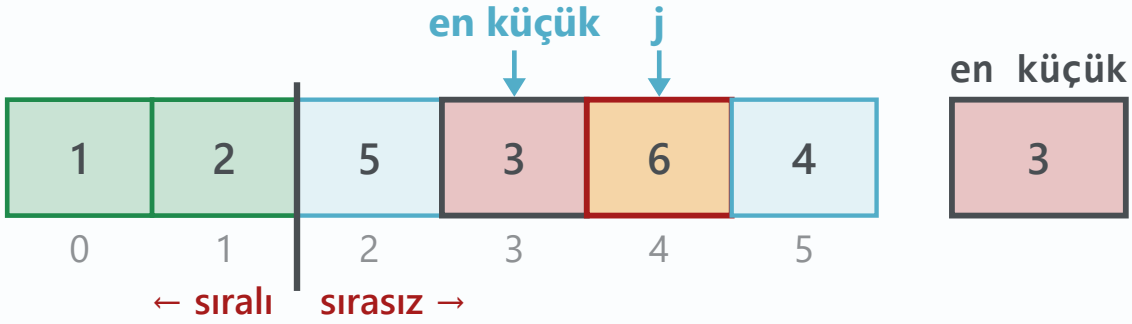
3. Tur



3 < en küçük (5)? **Evet** → en küçük = 3.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

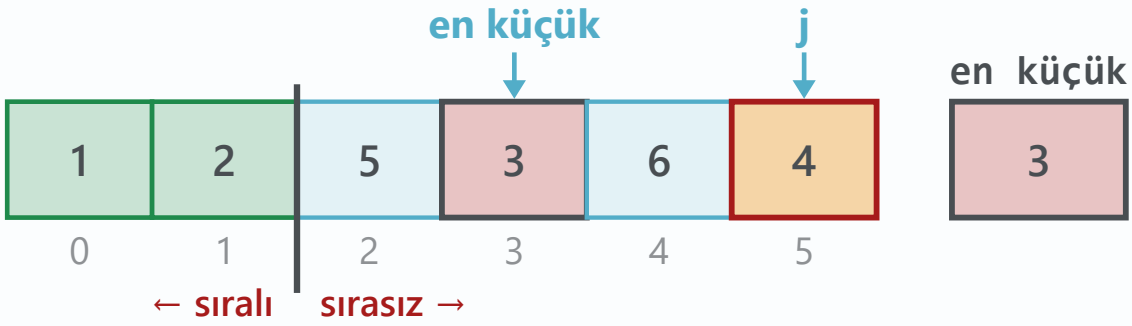
3. Tur



6 < en küçük (3)? Hayır.

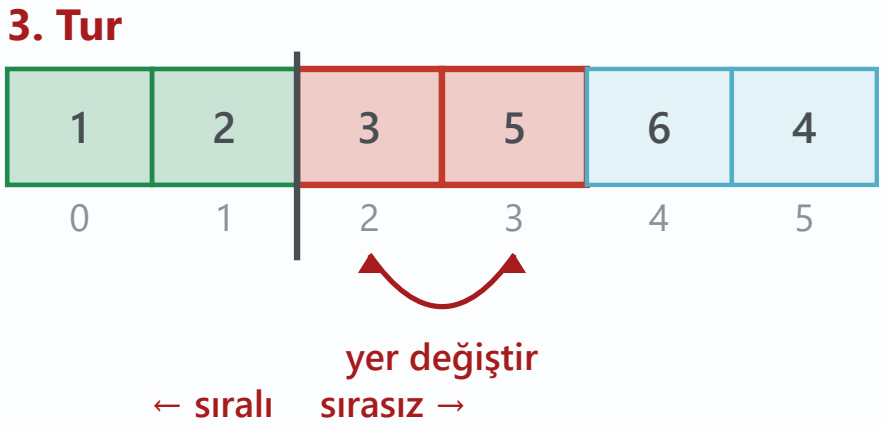
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

3. Tur



4 < en küçük (3)? Hayır.

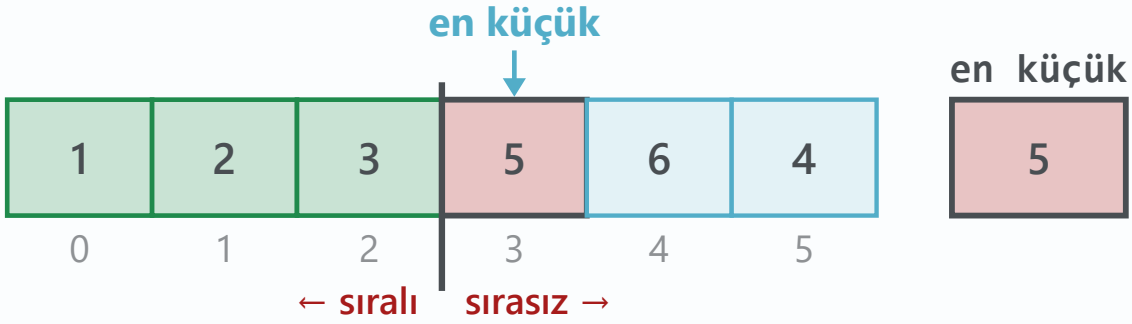
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



En küçük (3) sırasız kısmın başıyla yer değiştirir → [1, 2, 3, 5, 6, 4]; sıralı kısım bir büyür.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

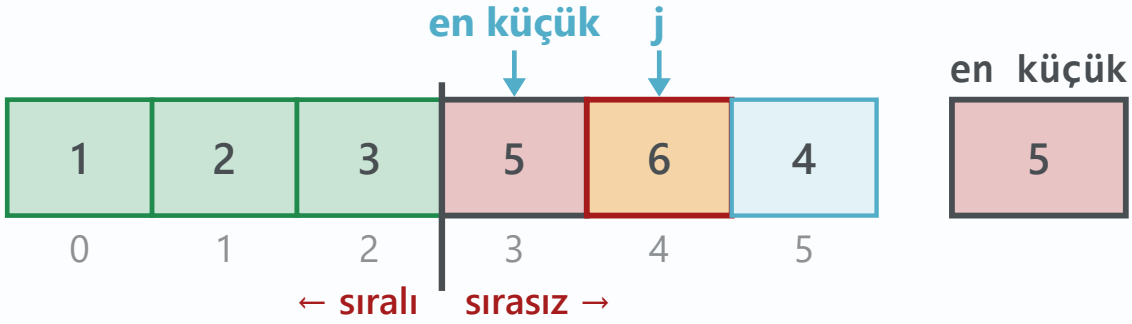
4. Tur



4. tur: sırasız kısmın ilk öğesi geçici en küçük kabul edilir: en küçük = 5.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

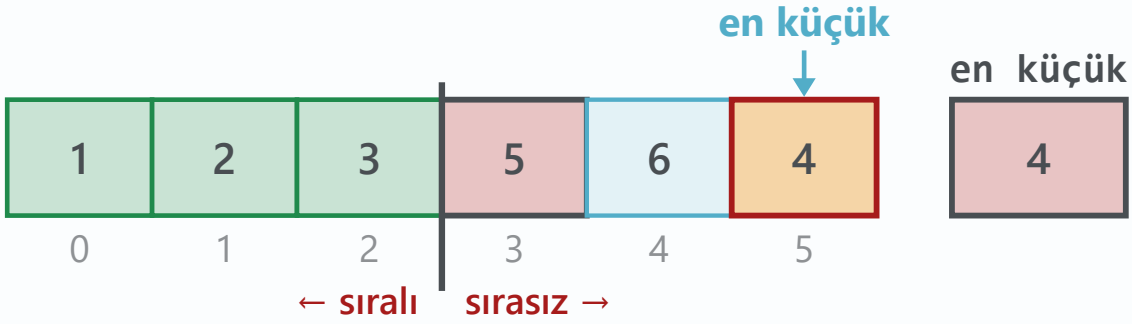
4. Tur



6 < en küçük (5)? Hayır.

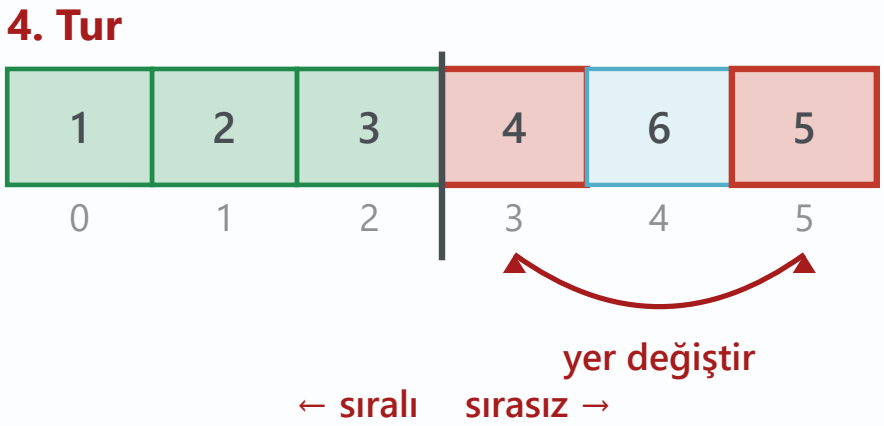
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

4. Tur



4 < en küçük (5)? **Evet** → en küçük = 4.

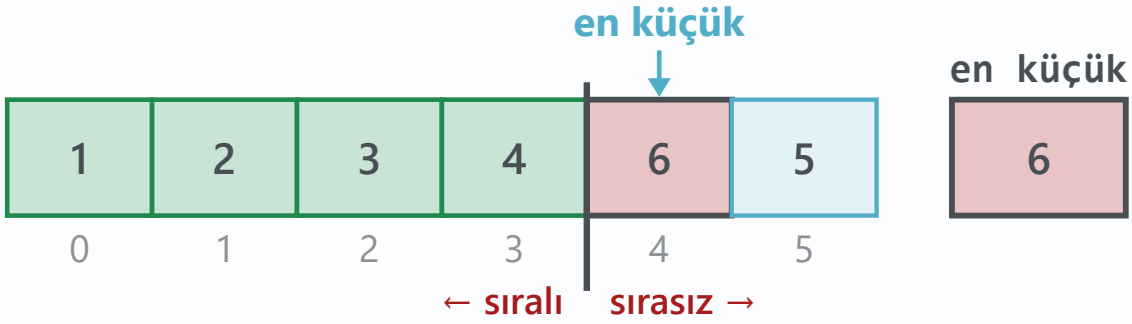
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



En küçük (4) sırasız kısmın başıyla yer değiştirir → [1, 2, 3, 4, 6, 5]; sıralı kısım bir büyür.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

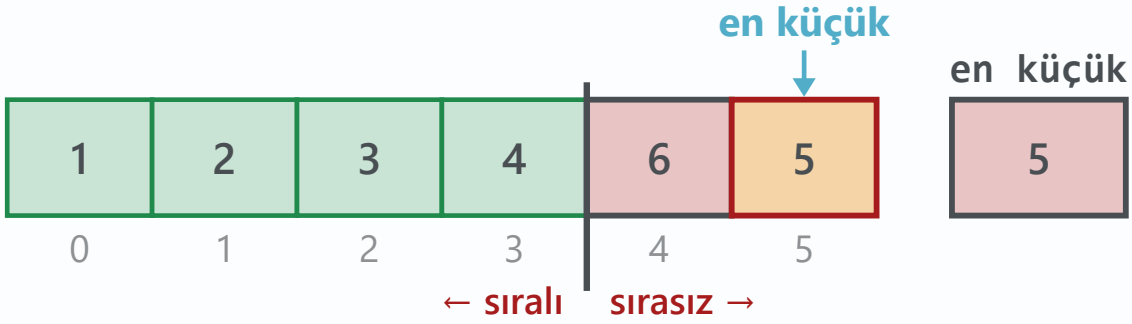
5. Tur



5. tur: sırasız kısmın ilk öğesi geçici en küçük kabul edilir: en küçük = 6.

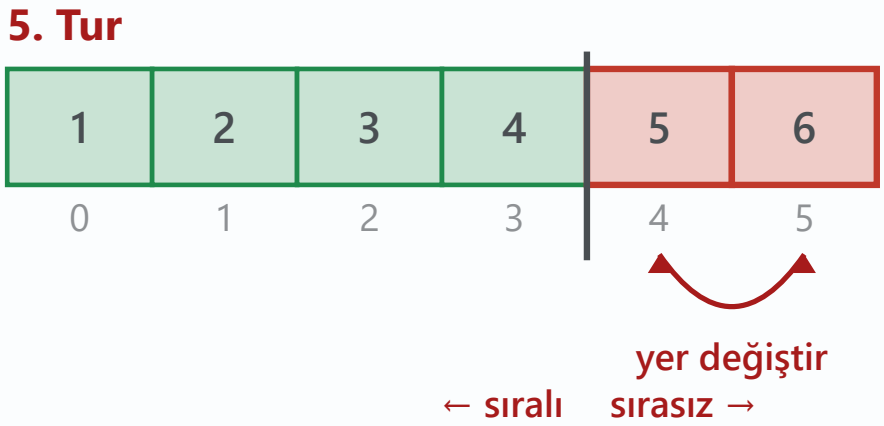
Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

5. Tur



5 < en küçük (6)? **Evet** → en küçük = 5.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]



En küçük (5) sırasız kısmın başıyla yer değiştirir → [1, 2, 3, 4, 5, 6]; sıralı kısım bir büyür.

Seçimli Sıralama Adım Adım: [3, 1, 5, 2, 6, 4]

Sonuç



5 tur sonunda son öğe kendiliğinden yerindedir. Sıralı dizi: [1, 2, 3, 4, 5, 6].

Seçimli Sıralama: Java Kodu

```java
public void sirala(int[] dizi) {
  int n = dizi.length;
  for(int i = 0; i < n - 1; i++) {
    int enKucuk = i;
    for(int j = i + 1; j < n; j++) {
      if(dizi[j] < dizi[enKucuk]) {
        enKucuk = j;
      }
    }
    int gecici = dizi[enKucuk];
    dizi[enKucuk] = dizi[i];
    dizi[i] = gecici;
  }
}
```

- `enKucuk` en küçük öğenin **indisini** tutar (değerini değil): sonunda yer değiştirmek için indis gerekir.
- İç döngü sırasız kısmın tamamını tarar; girdi sıralı olsa bile tarama kısalmaz.
- Tur başına bir takas: toplam en fazla **n − 1**.

Yazma işleminin pahalı olduğu ortamlarda (ör. flash bellek) az sayıda takas seçimli sıralamanın tek gerçek avantajıdır.

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
▶    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



n = dizi.length = 5.

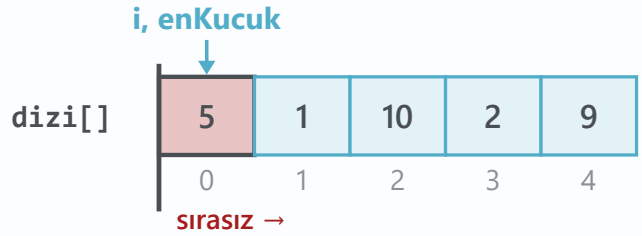
Adım 1 / 24

DEĞİŞKENLER

`n = 5` `i = –` `enKucuk = –` `j = –`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
▶    for(int i = 0; i < n - 1; i++) {
▶      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



i = 0: enKucuk = 0 (dizi[enKucuk] = 5).

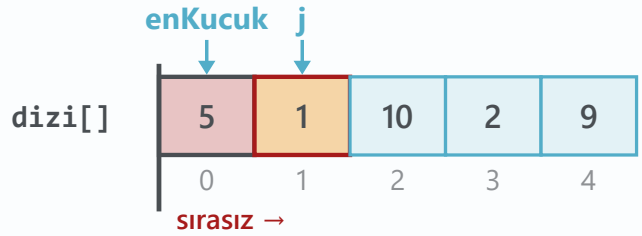
Adım 2 / 24

DEĞİŞKENLER

`n = 5` `i = 0` `enKucuk = 0` `j = –`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 1: dizi[1] = 1 < dizi[enKucuk] = 5? **Evet**.

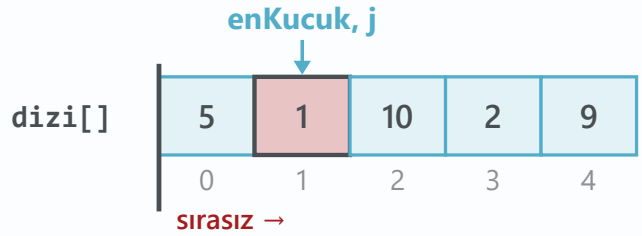
Adım 3 / 24

DEĞİŞKENLER

`n = 5` `i = 0` `enKucuk = 0` `j = 1`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
▶          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



enKucuk = 1 (yeni en küçük: 1).

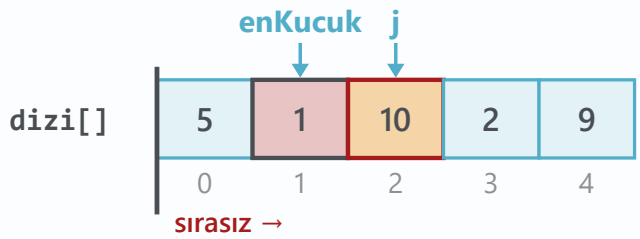
Adım 4 / 24

DEĞİŞKENLER

n = 5 | i = 0 | enKucuk = 1 | j = 1

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 2: dizi[2] = 10 < dizi[enKucuk] = 1? Hayır.

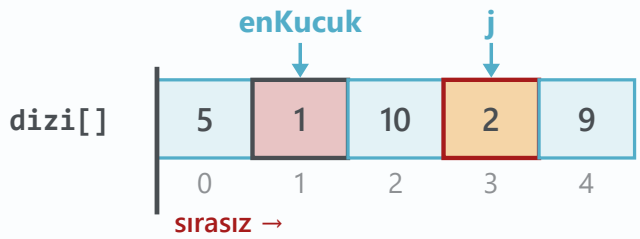
Adım 5 / 24

DEĞİŞKENLER

`n = 5` `i = 0` `enKucuk = 1` `j = 2`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 3: dizi[3] = 2 < dizi[enKucuk] = 1? Hayır.

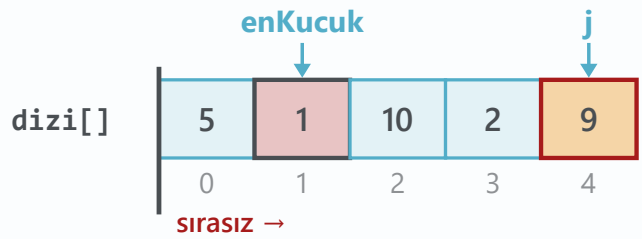
Adım 6 / 24

DEĞİŞKENLER

`n = 5` `i = 0` `enKucuk = 1` `j = 3`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 4: dizi[4] = 9 < dizi[enKucuk] = 1? Hayır.

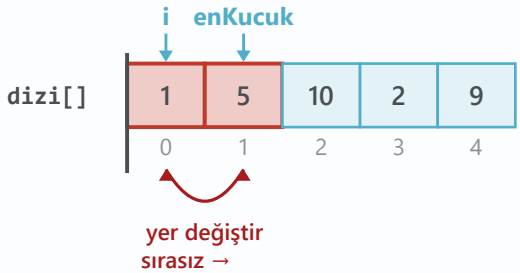
Adım 7 / 24

DEĞİŞKENLER

`n = 5` `i = 0` `enKucuk = 1` `j = 4`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
▶      int gecici = dizi[enKucuk];
▶      dizi[enKucuk] = dizi[i];
▶      dizi[i] = gecici;
13   }
14 }
```



j = 5 → iç döngü biter. `gecici` = 1; dizi[1] ↔ dizi[0] → [1, 5, 10, 2, 9].

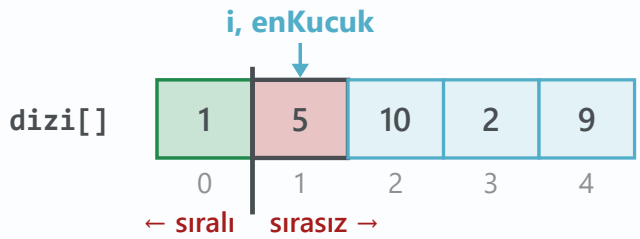
Adım 8 / 24

DEĞİŞKENLER

`n = 5` `i = 0` `enKucuk = 1` `j = 5` `gecici = 1`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
▶    for(int i = 0; i < n - 1; i++) {
▶      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



i = 1: enKucuk = 1 (dizi[enKucuk] = 5).

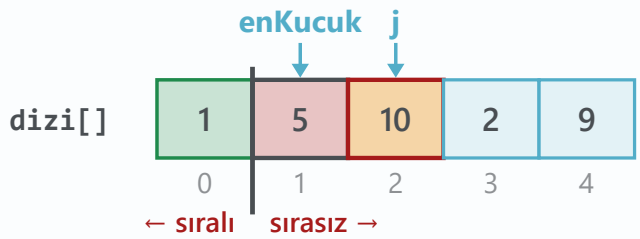
Adım 9 / 24

DEĞİŞKENLER

n = 5 | i = 1 | enKucuk = 1 | j = –

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 2: dizi[2] = 10 < dizi[enKucuk] = 5? Hayır.

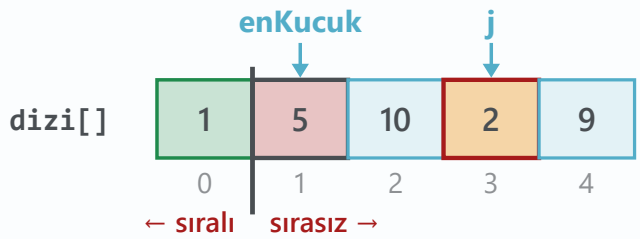
Adım 10 / 24

DEĞİŞKENLER

n = 5 | i = 1 | enKucuk = 1 | j = 2

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 3: dizi[3] = 2 < dizi[enKucuk] = 5? **Evet**.

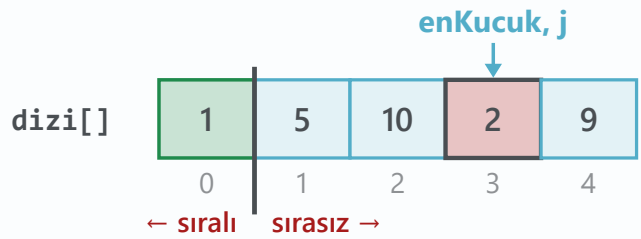
Adım 11 / 24

DEĞİŞKENLER

`n = 5` `i = 1` `enKucuk = 1` `j = 3`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
▶          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



enKucuk = 3 (yeni en küçük: 2).

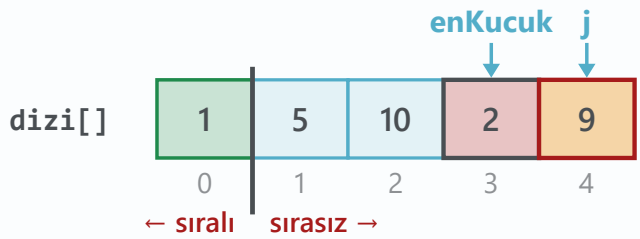
Adım 12 / 24

DEĞİŞKENLER

`n = 5` `i = 1` `enKucuk = 3` `j = 3`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 4: dizi[4] = 9 < dizi[enKucuk] = 2? Hayır.

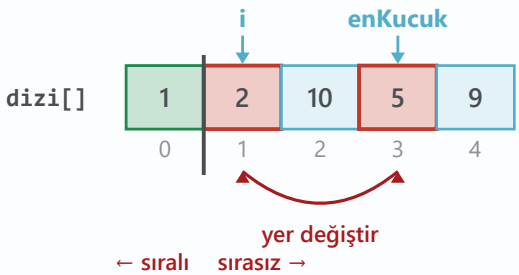
Adım 13 / 24

DEĞİŞKENLER

n = 5 | i = 1 | enKucuk = 3 | j = 4

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
▶      int gecici = dizi[enKucuk];
▶      dizi[enKucuk] = dizi[i];
▶      dizi[i] = gecici;
13   }
14 }
```



j = 5 → iç döngü biter. `gecici` = 2; dizi[3] ↔ dizi[1] → [1, 2, 10, 5, 9].

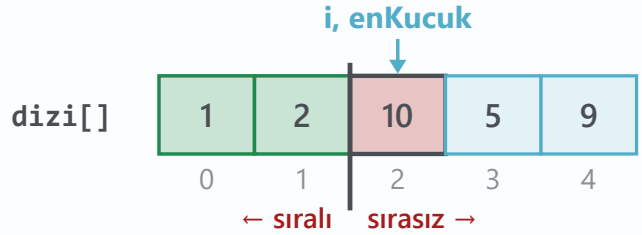
Adım 14 / 24

DEĞİŞKENLER

`n = 5` `i = 1` `enKucuk = 3` `j = 5` `gecici = 2`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
▶    for(int i = 0; i < n - 1; i++) {
▶      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



i = 2: enKucuk = 2 (dizi[enKucuk] = 10).

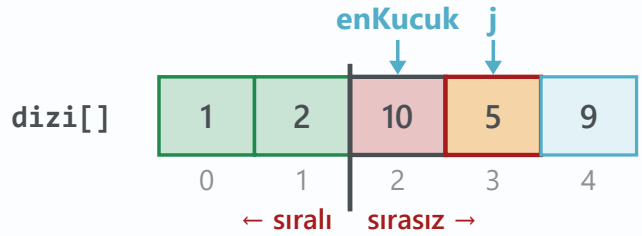
Adım 15 / 24

DEĞİŞKENLER

n = 5 | i = 2 | enKucuk = 2 | j = –

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 3: dizi[3] = 5 < dizi[enKucuk] = 10? **Evet**.

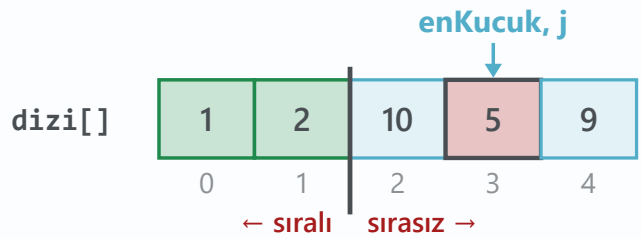
Adım 16 / 24

DEĞİŞKENLER

`n = 5` `i = 2` `enKucuk = 2` `j = 3`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
▶          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



enKucuk = 3 (yeni en küçük: 5).

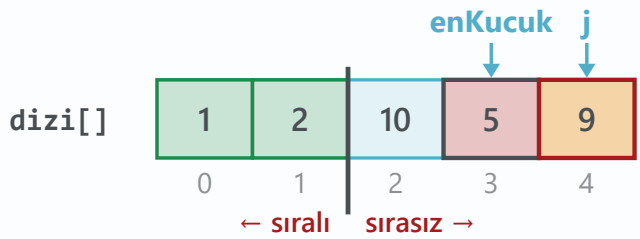
Adım 17 / 24

DEĞİŞKENLER

n = 5 | i = 2 | enKucuk = 3 | j = 3

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 4: dizi[4] = 9 < dizi[enKucuk] = 5? Hayır.

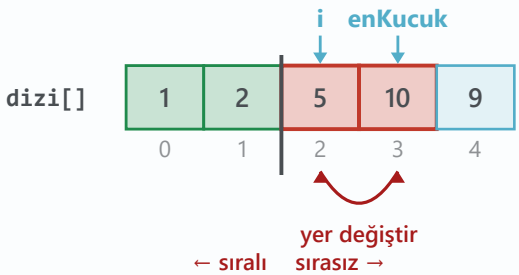
Adım 18 / 24

DEĞİŞKENLER

n = 5 | i = 2 | enKucuk = 3 | j = 4

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
▶      int gecici = dizi[enKucuk];
▶      dizi[enKucuk] = dizi[i];
▶      dizi[i] = gecici;
13   }
14 }
```



j = 5 → iç döngü biter. `gecici` = 5; dizi[3] ↔ dizi[2] → [1, 2, 5, 10, 9].

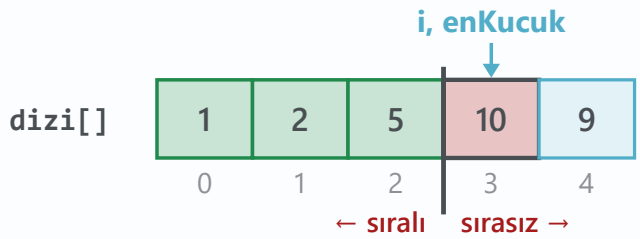
Adım 19 / 24

DEĞİŞKENLER

`n = 5` `i = 2` `enKucuk = 3` `j = 5` `gecici = 5`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
▶    for(int i = 0; i < n - 1; i++) {
▶      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



i = 3: enKucuk = 3 (dizi[enKucuk] = 10).

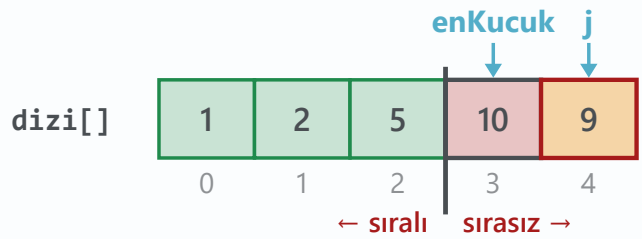
Adım 20 / 24

DEĞİŞKENLER

n = 5 | i = 3 | enKucuk = 3 | j = –

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
▶      for(int j = i + 1; j < n; j++) {
▶        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



j = 4: dizi[4] = 9 < dizi[enKucuk] = 10? **Evet**.

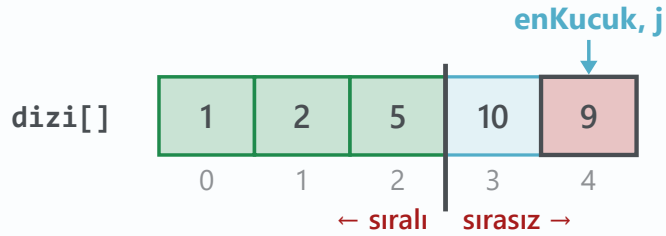
Adım 21 / 24

DEĞİŞKENLER

`n = 5` `i = 3` `enKucuk = 3` `j = 4`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
▶          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



enKucuk = 4 (yeni en küçük: 9).

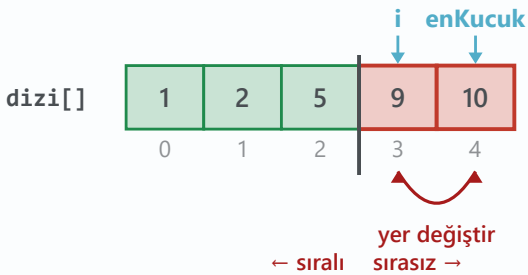
Adım 22 / 24

DEĞİŞKENLER

`n = 5` `i = 3` `enKucuk = 4` `j = 4`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
3    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
▶      int gecici = dizi[enKucuk];
▶      dizi[enKucuk] = dizi[i];
▶      dizi[i] = gecici;
13   }
14 }
```



j = 5 → iç döngü biter. `gecici` = 9; dizi[4] ↔ dizi[3] → [1, 2, 5, 9, 10].

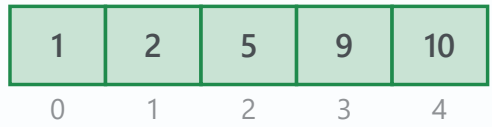
Adım 23 / 24

DEĞİŞKENLER

`n = 5` `i = 3` `enKucuk = 4` `j = 5` `gecici = 9`

Seçimli Sıralama: Kodu İzle, dizi = [5, 1, 10, 2, 9]

```
1  public void sirala(int[] dizi) {
2    int n = dizi.length;
▶    for(int i = 0; i < n - 1; i++) {
4      int enKucuk = i;
5      for(int j = i + 1; j < n; j++) {
6        if(dizi[j] < dizi[enKucuk]) {
7          enKucuk = j;
8        }
9      }
10     int gecici = dizi[enKucuk];
11     dizi[enKucuk] = dizi[i];
12     dizi[i] = gecici;
13   }
14 }
```



i = 4 < n − 1 değil → biter. Sonuç: [1, 2, 5, 9, 10].

Adım 24 / 24

DEĞİŞKENLER

n = 5 | i = 4 | enKucuk = – | j = –

Seçimli Sıralama: Analiz

Ölçü	Değer
Karşılaştırma	(n − 1) + (n − 2) + … + 1 = **n(n − 1)/2**, *her* girdide
Takas	en fazla n − 1
Zaman	en iyi = ortalama = en kötü = **Θ(n^2)**
Ek bellek	O(1), yerinde
Kararlı mı?	**Hayır** (uzun mesafeli takas)



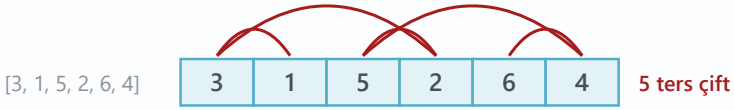
En küçük (1), baştaki 2_a ile yer değiştirir: 2_a, 2_β'nin **arkasına** geçer.

Uyarlanabilir değildir: sıralı bir diziye de n(n − 1)/2 karşılaştırma harcar.

Kabarcık, Eklemeli, Seçimli: Aynı Girdide

Hücreler: **karşılaştırma / takas (ya da kaydırma)** sayısı, n = 6. Değerler kodlar çalıştırılarak sayılmıştır.

Algoritma	Taşıma türü	[3, 1, 5, 2, 6, 4]	sıralı 1…6	ters 6…1
Kabarcık (bayraklı)	takas	12 / 5	5 / 0	15 / 15
Eklemeli	kaydırma	9 / 5	5 / 0	15 / 15
Seçimli	takas	15 / 5	15 / 0	15 / 3



- Kabarcık ve eklemeli sıralamada taşıma sayısı = **ters çift** sayısı ([3, 1, 5, 2, 6, 4] için 5); bir takas 3, bir kaydırma 1 atamadır.
- Seçimli sıralama girdiden bağımsız olarak hep n(n − 1)/2 = 15 karşılaştırma yapar ama en az takası o yapar.
- Sıralı girdide bayraklı kabarcık ve eklemeli yalnızca n − 1 = 5 karşılaştırma yapar (uyarlanabilir).

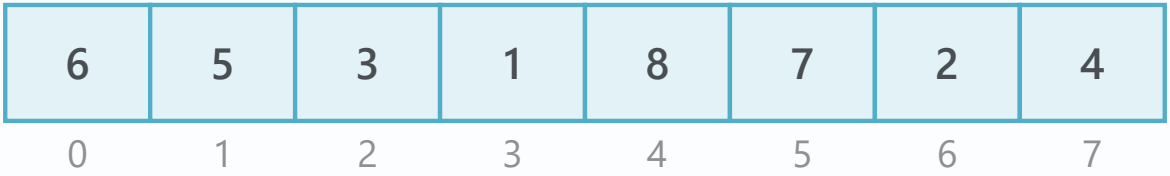
Shell Sıralaması (Shell Sort)

- D. Shell (1959): eklemeli sıralamanın **genelleştirilmesi**.
- Önce birbirinden **aralık** (gap) uzaklıktaki öğeler eklemeli sıralanır; uzaktaki küçük öğeler tek hamlede büyük mesafe atlar.
- Aralık küçültülür (n/2, n/4, …); son tur **aralık = 1**, yani sıradan eklemeli sıralama — ama dizi artık neredeyse sıralıdır.
- Yerinde, **kararsız**; süre aralık dizisine bağlı: n/2 aralıklarıyla en kötü $\Theta(n^2)$, Hibbard $(2^k - 1)$ ile $O(n^{1,5})$.

```
void kabukSirala(int[] dizi) {
  int n = dizi.length;
  for (int aralik = n / 2; aralik > 0; aralik /= 2)
    for (int i = aralik; i < n; i++) {
      int gecici = dizi[i], j;
      for (j = i; j >= aralik && dizi[j - aralik] > gecici; j -= aralik)
        dizi[j] = dizi[j - aralik];
      dizi[j] = gecici;
    }
}
```

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

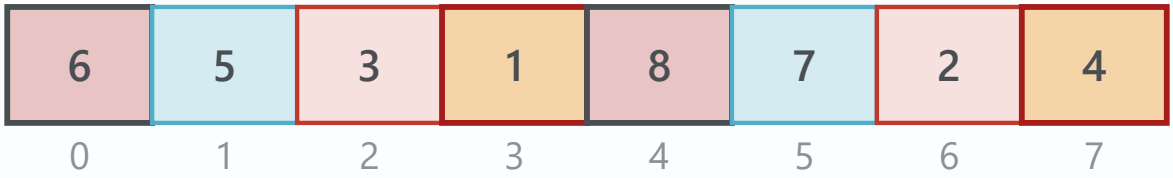
Başlangıç



Dizi [6, 5, 3, 1, 8, 7, 2, 4], n = 8. Aralık dizisi: n/2 = 4, 2, … , 1.

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

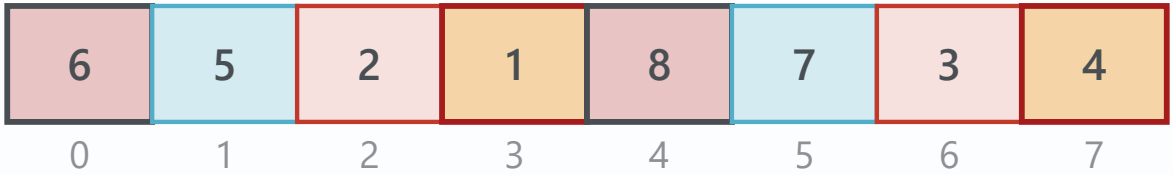
aralik = 4: gruplar



aralik = 4: aralarında 4 indis olan öğeler aynı grupta (aynı renk): [6, 8], [5, 7], [3, 2], [1, 4].

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

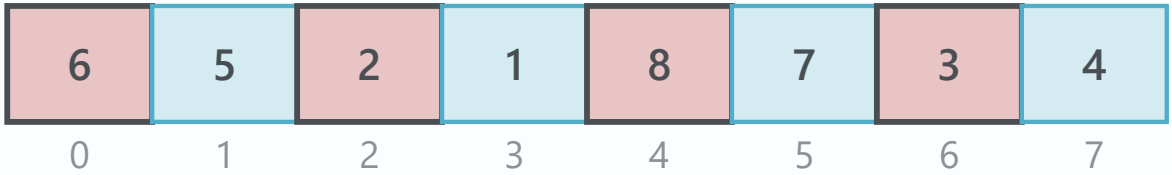
aralik = 4: sonra



Her grup eklemeli sıralamayla sıralandı (1 kaydırma): [6, 8], [5, 7], [2, 3], [1, 4] → [6, 5, 2, 1, 8, 7, 3, 4].

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

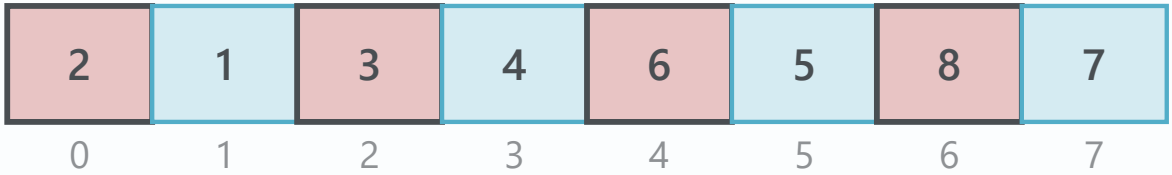
aralik = 2: gruplar



aralik = 2: aralarında 2 indis olan öğeler aynı grupta (aynı renk): [6, 2, 8, 3], [5, 1, 7, 4].

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

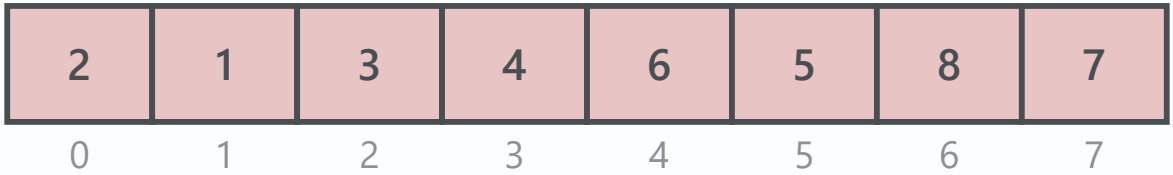
aralik = 2: sonra



Her grup eklemeli sıralamayla sıralandı (6 kaydırma): [2, 3, 6, 8], [1, 4, 5, 7] → [2, 1, 3, 4, 6, 5, 8, 7].

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

aralik = 1: gruplar



aralik = 1: aralarında 1 indis olan öğeler aynı grupta (aynı renk): [2, 1, 3, 4, 6, 5, 8, 7].

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

aralik = 1: sonra



aralik = 1 sıradan eklemeli sıralamadır; dizi neredeyse sıralı olduğu için yalnızca 3 kaydırma gerekti. Sonuç: [1, 2, 3, 4, 5, 6, 7, 8].

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
▶    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
4      for (int i = aralik; i < n; i++) {
5        int gecici = dizi[i], j;
6        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
8        dizi[j] = gecici;
9      }
10 }
```



aralik = 3: aynı işaretli öğeler bir grup

dizi = [5, 2, 6, 1, 4, 3], n = 6. İlk aralık n / 2 = 3: birbirinden 3 uzaktaki öğeler (0, 3), (1, 4), (2, 5) ayrı gruplar olarak eklemeli sıralanacak.

Adım 1 / 10

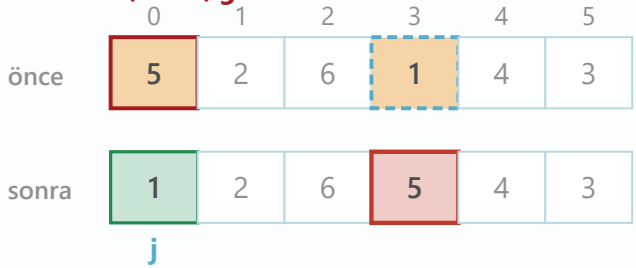
DEĞİŞKENLER

`n = 6` `aralik = –` `i = –` `gecici = –` `j = –`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
4      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
▶          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 3, i = 3, gecici = 1



i = 3: gecici = 1. dizi[0] = 5 > 1 → 3 indisine kayar. j = 0'da durulur; 1 oraya yazılır.

Adım 2 / 10

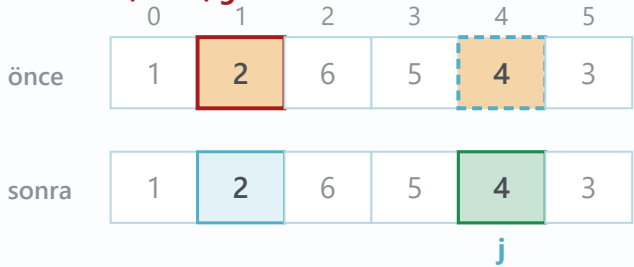
DEĞİŞKENLER

`n = 6` `aralik = 3` `i = 3` `gecici = 1` `j = 0`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
▶      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 3, i = 4, gecici = 4



i = 4: gecici = 4. dizi[1] = 2 ≤ 4: kaydırma yok, 4 yerinde kalır.

Adım 3 / 10

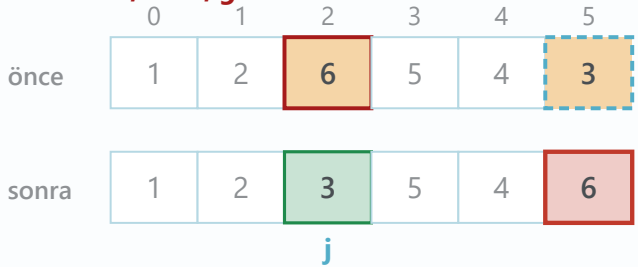
DEĞİŞKENLER

`n = 6` `aralik = 3` `i = 4` `gecici = 4` `j = 4`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
4      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
▶          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 3, i = 5, gecici = 3



i = 5: gecici = 3. dizi[2] = 6 > 3 → 5 indisine kayar. j = 2'de durulur; 3 oraya yazılır.

Adım 4 / 10

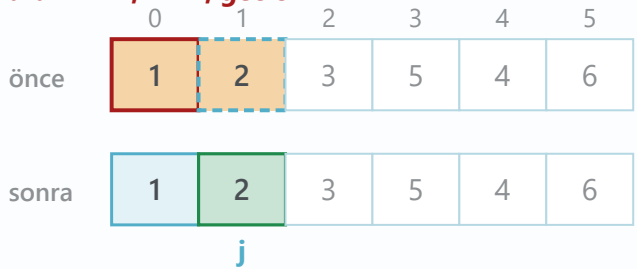
DEĞİŞKENLER

`n = 6` `aralik = 3` `i = 5` `gecici = 3` `j = 2`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
▶      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 1, i = 1, gecici = 2



aralik = 3 / 2 = 1: artık sıradan eklemeli sıralama. i = 1: gecici = 2. dizi[0] = 1 ≤ 2: kaydırma yok, 2 yerinde kalır.

Adım 5 / 10

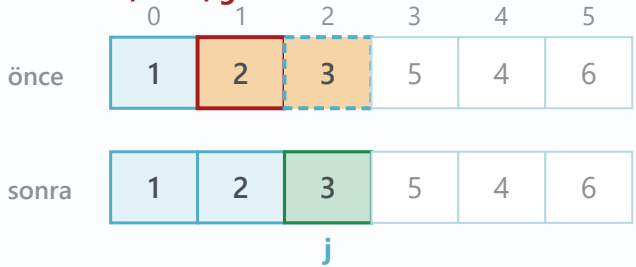
DEĞİŞKENLER

`n = 6` `aralik = 1` `i = 1` `gecici = 2` `j = 1`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
▶      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 1, i = 2, gecici = 3



i = 2: gecici = 3. dizi[1] = 2 ≤ 3: kaydırma yok, 3 yerinde kalır.

Adım 6 / 10

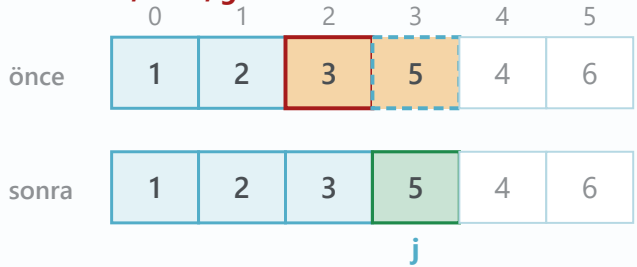
DEĞİŞKENLER

`n = 6` `aralik = 1` `i = 2` `gecici = 3` `j = 2`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
▶      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 1, i = 3, gecici = 5



i = 3: gecici = 5. dizi[2] = 3 ≤ 5: kaydırma yok, 5 yerinde kalır.

Adım 7 / 10

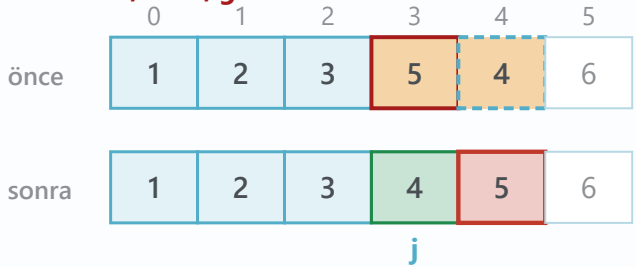
DEĞİŞKENLER

`n = 6` `aralik = 1` `i = 3` `gecici = 5` `j = 3`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
4      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
▶          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 1, i = 4, gecici = 4



i = 4: gecici = 4. dizi[3] = 5 > 4 → 4 indisine kayar. dizi[2] = 3 ≤ 4, durulur; 4 oraya yazılır.

Adım 8 / 10

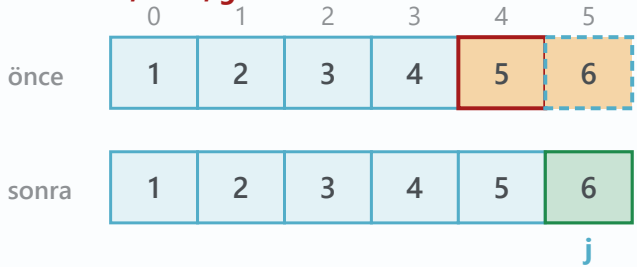
DEĞİŞKENLER

`n = 6` `aralik = 1` `i = 4` `gecici = 4` `j = 3`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
3    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
▶      for (int i = aralik; i < n; i++) {
▶        int gecici = dizi[i], j;
▶        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
▶        dizi[j] = gecici;
9      }
10 }
```

aralik = 1, i = 5, gecici = 6



i = 5: gecici = 6. dizi[4] = 5 ≤ 6: kaydırma yok, 6 yerinde kalır.

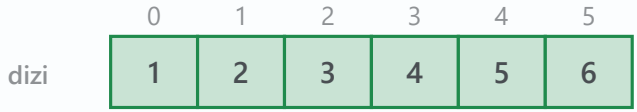
Adım 9 / 10

DEĞİŞKENLER

`n = 6` `aralik = 1` `i = 5` `gecici = 6` `j = 5`

Shell Sıralaması: Kodu İzle, dizi = [5, 2, 6, 1, 4, 3]

```
1  void kabukSirala(int[] dizi) {
2    int n = dizi.length;
▶    for (int aralik = n / 2; aralik > 0; aralik
   /= 2)
4      for (int i = aralik; i < n; i++) {
5        int gecici = dizi[i], j;
6        for (j = i; j >= aralik && dizi[j -
   aralik] > gecici; j -= aralik)
7          dizi[j] = dizi[j - aralik];
8        dizi[j] = gecici;
9      }
10 }
```



aralik = 1 / 2 = 0: döngü biter, dizi sıralı. Toplam yalnızca **3 kaydırma** yapıldı; aralık 3 turu küçük öğeleri tek hamlede öne taşıdı.

Adım 10 / 10

DEĞİŞKENLER

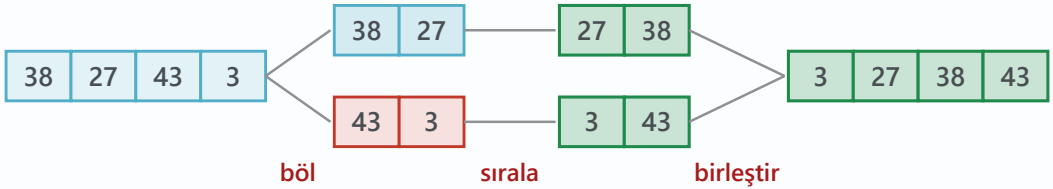
`n = 6` `aralik = 0` `i = –` `gecici = –` `j = –`

Birleştirme Sıralaması (Merge Sort)

- **Böl ve fethet** (divide and conquer) yaklaşımını kullanır.
- 1945'te **John von Neumann** tarafından geliştirilmiştir.
- En kötü durumda bile **O(n log n)**.
- **Kararlıdır**; birleştirme için **ek bellek** (O(n)) gerekir.

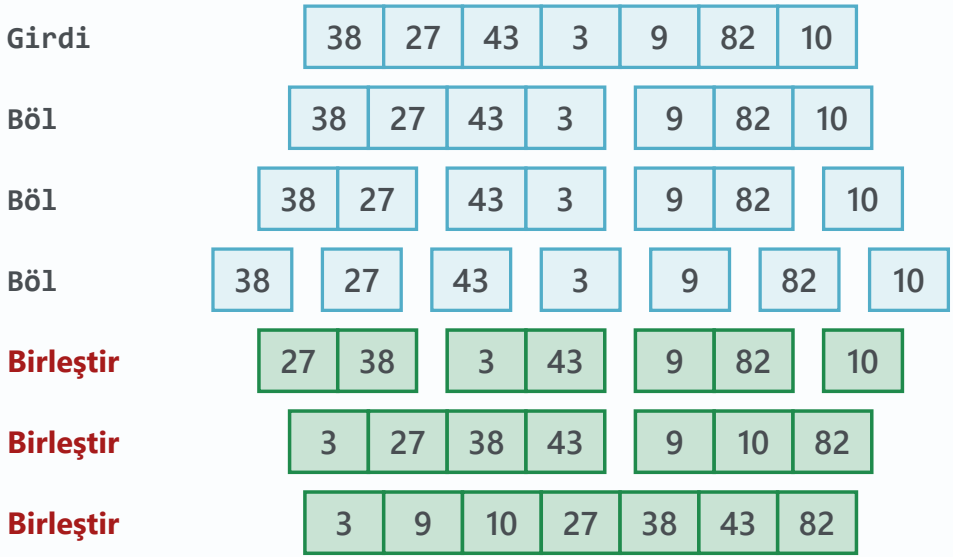
Nasıl çalışır?

1. Böl (divide): liste iki alt listeye bölünür; alt listeler tek öğe kalana kadar tekrar tekrar bölünür.
2. Fethet (conquer): her yarı aynı yöntemle (özyinelemeli) sıralanır.
3. Birleştir (combine): sıralı alt listeler, sıralı tek bir liste oluşana kadar birleştirilir.



Tek öğeli liste tanım gereği sıralıdır; bu, özyinelemenin **taban durumudur**. Asıl iş birleştirme adımında yapılır.

Örnek: [38, 27, 43, 3, 9, 82, 10]



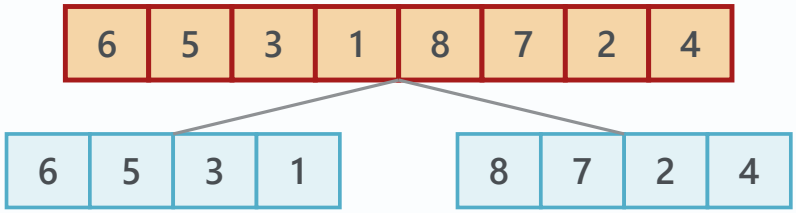
Yedi öğe: orta = [(sol + sağ)/2] olduğu için sol yarı bir öğe fazla alır ([38, 27, 43, 3] ve [9, 82, 10]).

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



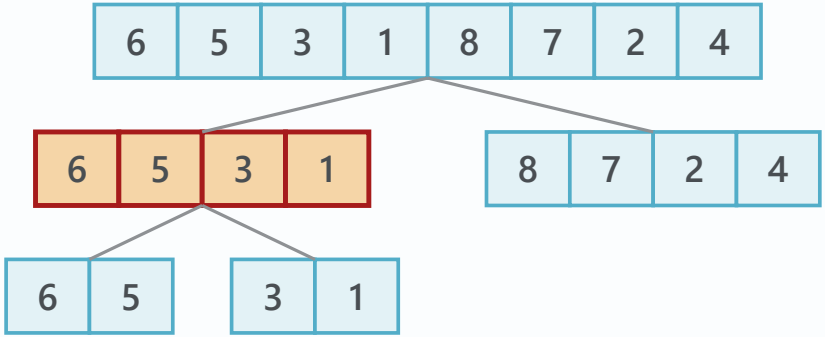
Dizi [6, 5, 3, 1, 8, 7, 2, 4]: önce tek öğe kalana kadar **bölünür**, sonra sıralı parçalar **birleştirilir**.

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



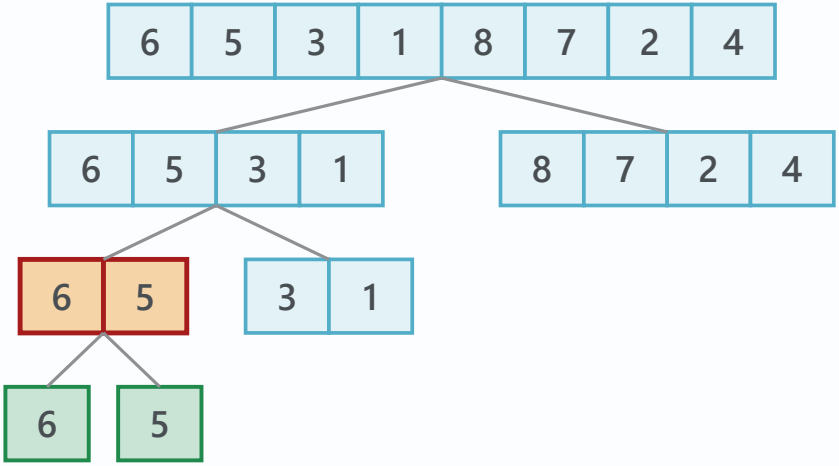
Böl: [6, 5, 3, 1, 8, 7, 2, 4] → [6, 5, 3, 1] ve [8, 7, 2, 4].

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



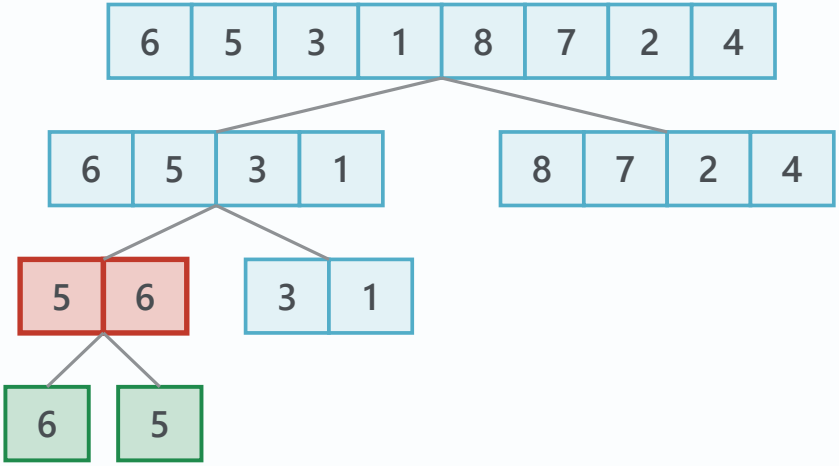
Böl: [6, 5, 3, 1] → [6, 5] ve [3, 1].

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



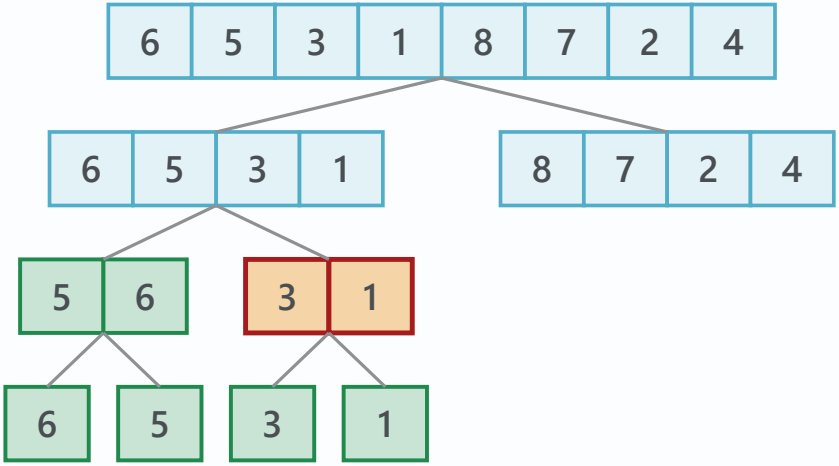
Böl: [6, 5] → [6] ve [5]. Tek öğeli parçalar zaten sıralıdır.

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



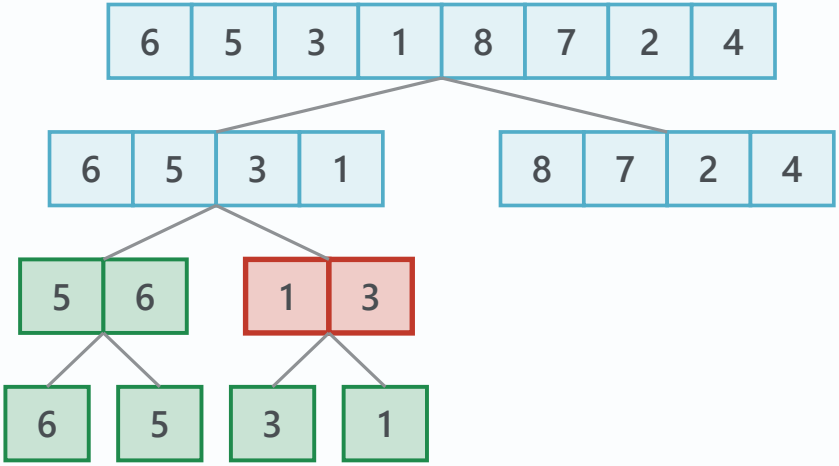
Birleştir: [6] + [5] → **[5, 6]**.

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



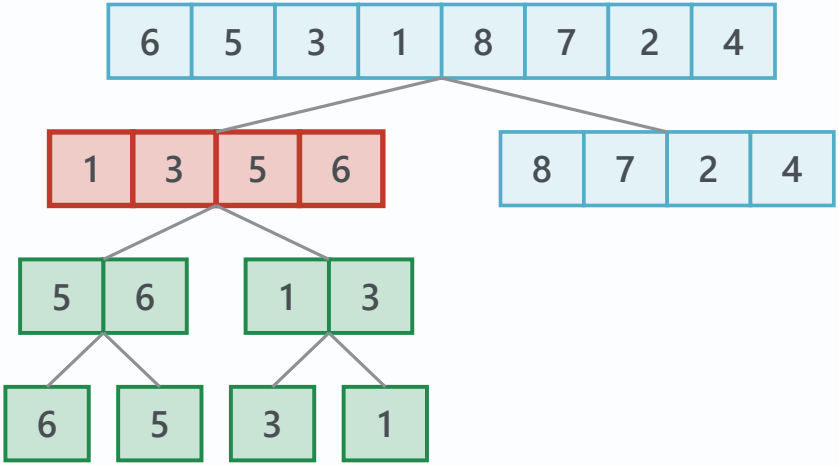
Böl: [3, 1] → [3] ve [1]. Tek öğeli parçalar zaten sıralıdır.

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



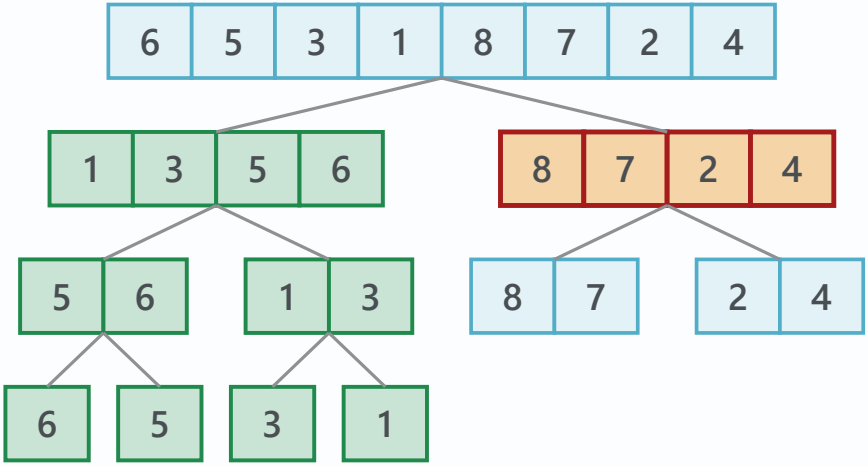
Birleştir: [3] + [1] → **[1, 3]**.

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



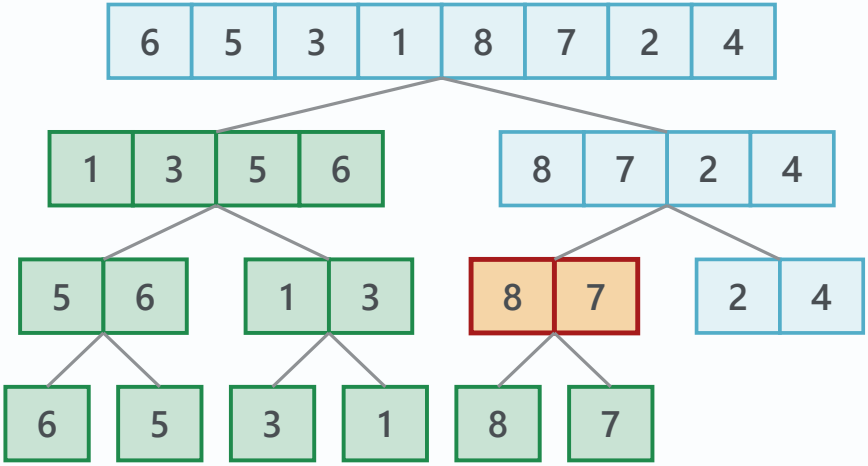
Birleştir: [5, 6] + [1, 3] → **[1, 3, 5, 6]**.

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



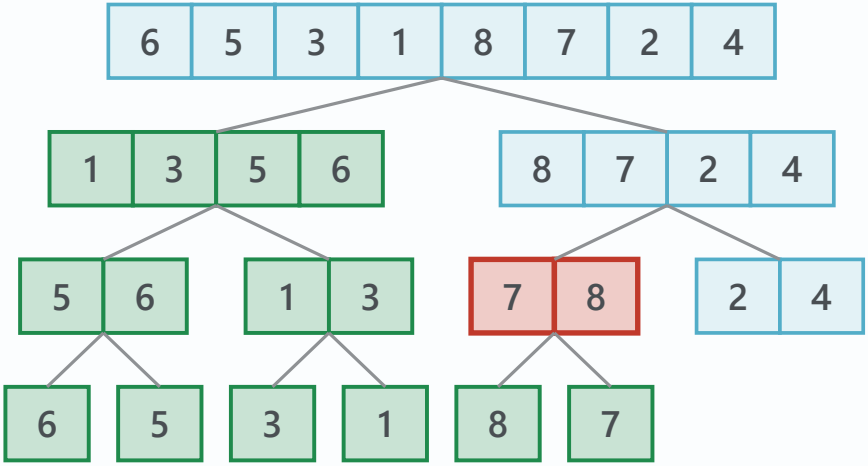
Böl: [8, 7, 2, 4] → [8, 7] ve [2, 4].

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



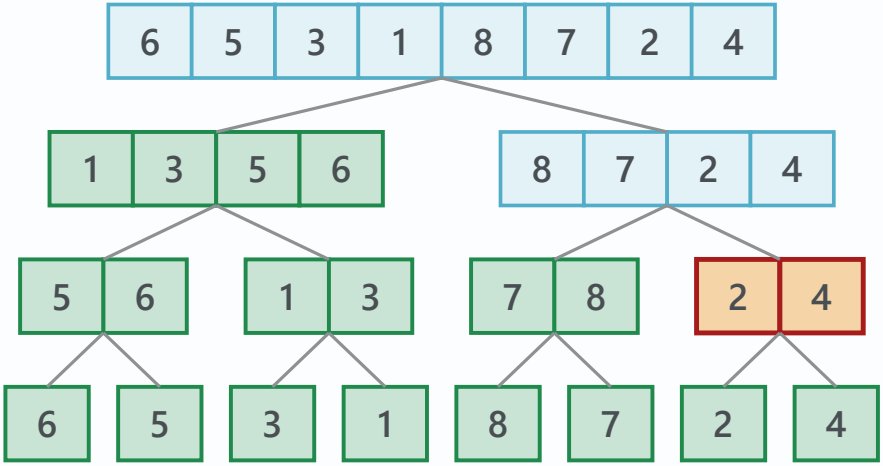
Böl: [8, 7] → [8] ve [7]. Tek öğeli parçalar zaten sıralıdır.

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



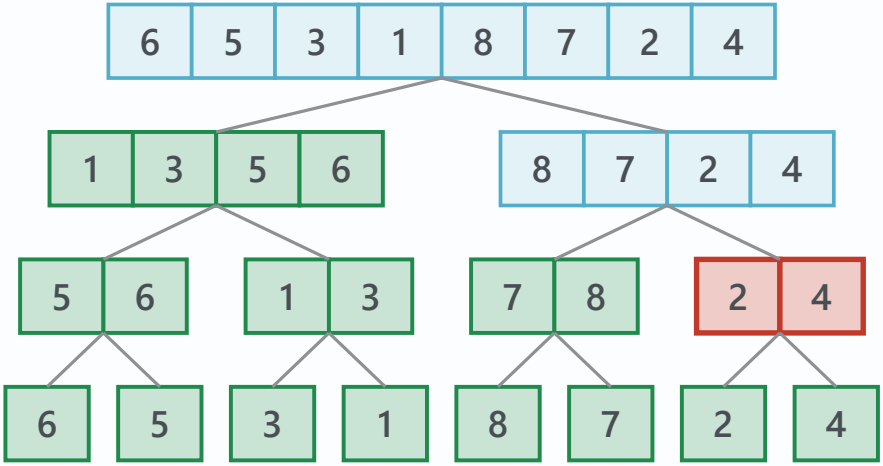
Birleştir: [8] + [7] → **[7, 8]**.

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



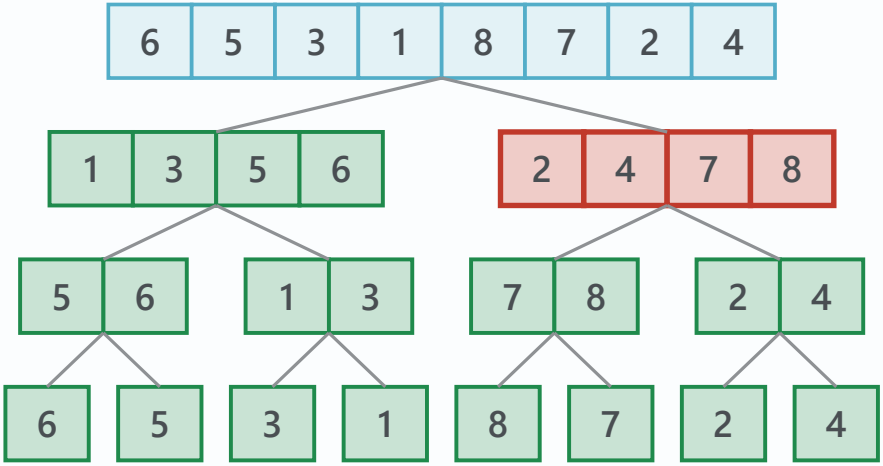
Böl: [2, 4] → [2] ve [4]. Tek öğeli parçalar zaten sıralıdır.

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



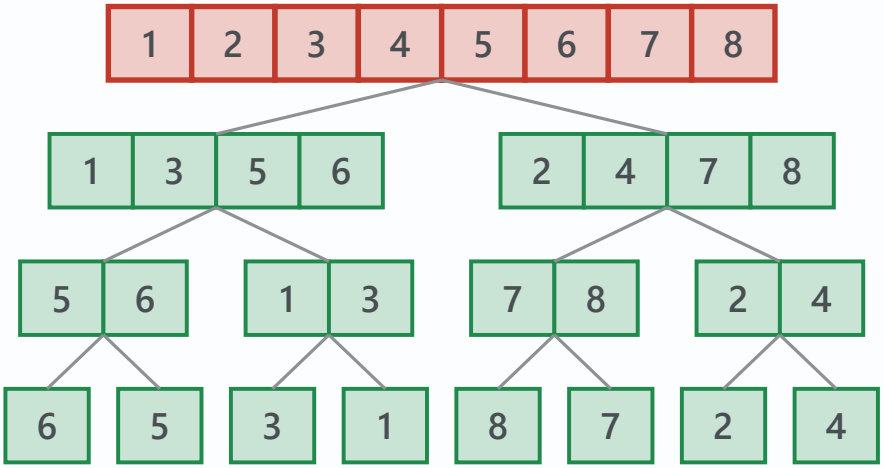
Birleştir: [2] + [4] → **[2, 4]**.

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



Birleştir: [7, 8] + [2, 4] → **[2, 4, 7, 8]**.

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



Birleştir: [1, 3, 5, 6] + [2, 4, 7, 8] → **[1, 2, 3, 4, 5, 6, 7, 8]**. Dizi sıralandı.

Merge Sort: Java Kodu

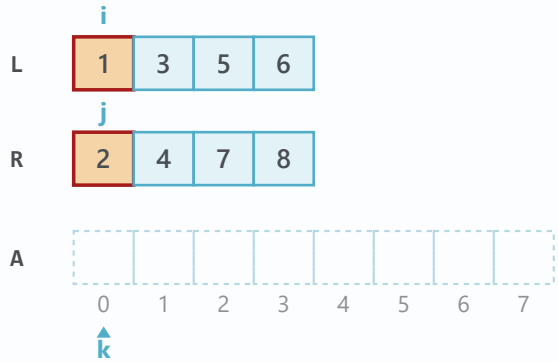
```java
void birlestirmeliSirala(int[] dizi, int sol, int sag) {
  if (sol < sag) {
    int orta = sol + (sag - sol) / 2;
    birlestirmeliSirala(dizi, sol, orta);
    birlestirmeliSirala(dizi, orta + 1, sag);
    birlestir(dizi, sol, orta, sag);
  }
}
```

```java
void birlestir(int[] A, int p, int q, int r) {
  int[] L = Arrays.copyOfRange(A, p, q + 1);
  int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
  int i = 0, j = 0, k = p;
  while (i < L.length && j < R.length) {
    if (L[i] <= R[j]) A[k++] = L[i++];
    else              A[k++] = R[j++];
  }
  while (i < L.length) A[k++] = L[i++];
  while (j < R.length) A[k++] = R[j++];
}
```

`sol + (sag - sol) / 2` yazımı, `(sol + sag) / 2` toplamındaki int taşmasını önler.

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
▶    int[] L = Arrays.copyOfRange(A, p, q + 1);
▶    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
▶    int i = 0, j = 0, k = p;
5    while (i < L.length && j < R.length) {
6      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



L = A[p..q] = [1, 3, 5, 6] ve R = A[q+1..r] = [2, 4, 7, 8] kopyalanır (ikisi de sıralı). i = j = 0, k = p = 0.

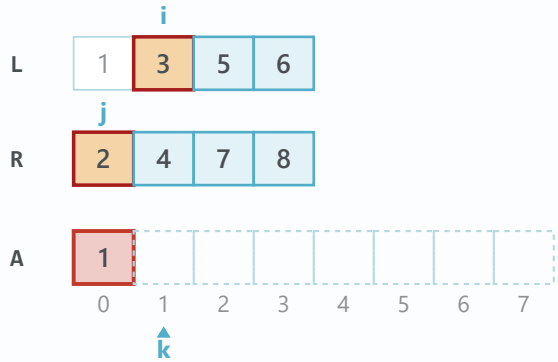
Adım 1 / 9

DEĞİŞKENLER

`i = 0` `j = 0` `k = 0`

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
▶    while (i < L.length && j < R.length) {
▶      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



L[0] = 1 ≤ R[0] = 2 → A[0] = 1; i++, k++.

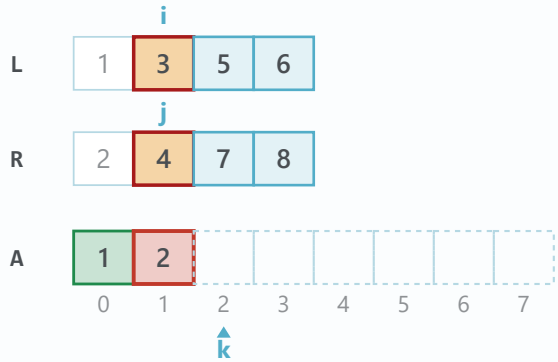
Adım 2 / 9

DEĞİŞKENLER

`i = 1` `j = 0` `k = 1`

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
▶    while (i < L.length && j < R.length) {
6      if (L[i] <= R[j]) A[k++] = L[i++];
▶      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



L[1] = 3 > R[0] = 2 → A[1] = 2; j++, k++.

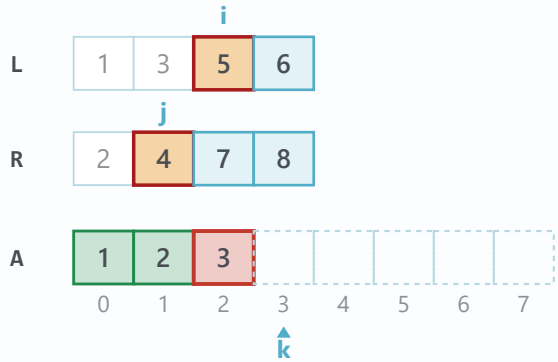
Adım 3 / 9

DEĞİŞKENLER

`i = 1` `j = 1` `k = 2`

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
▶    while (i < L.length && j < R.length) {
▶      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



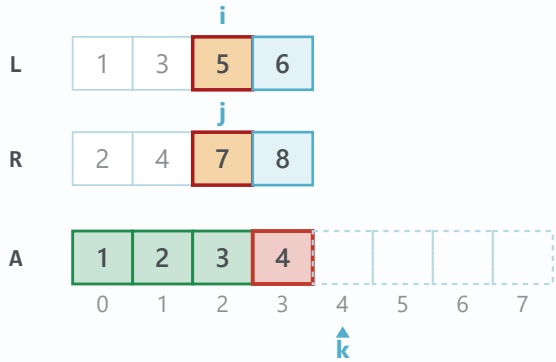
L[1] = 3 ≤ R[1] = 4 → A[2] = 3; i++, k++.

DEĞİŞKENLER

i = 2 | j = 1 | k = 3

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
▶    while (i < L.length && j < R.length) {
6      if (L[i] <= R[j]) A[k++] = L[i++];
▶      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



L[2] = 5 > R[1] = 4 → A[3] = 4; j++, k++.

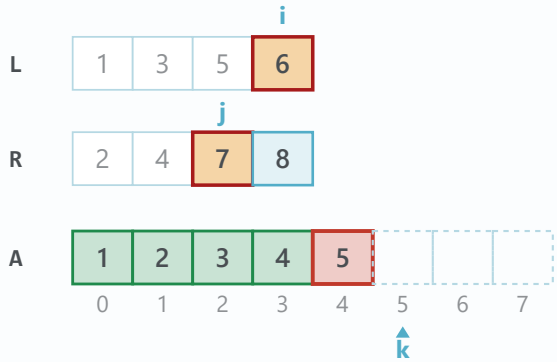
Adım 5 / 9

DEĞİŞKENLER

i = 2 j = 2 k = 4

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
▶    while (i < L.length && j < R.length) {
▶      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



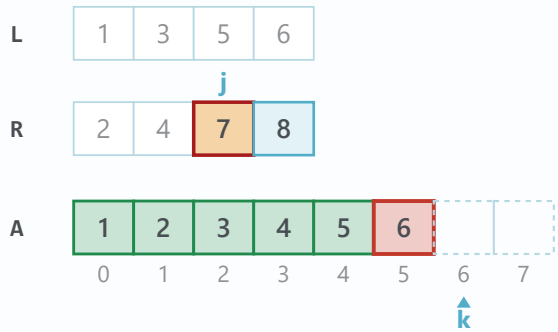
L[2] = 5 ≤ R[2] = 7 → A[4] = 5; i++, k++.

DEĞİŞKENLER

i = 3 | j = 2 | k = 5

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
▶    while (i < L.length && j < R.length) {
▶      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
10   while (j < R.length) A[k++] = R[j++];
11 }
```



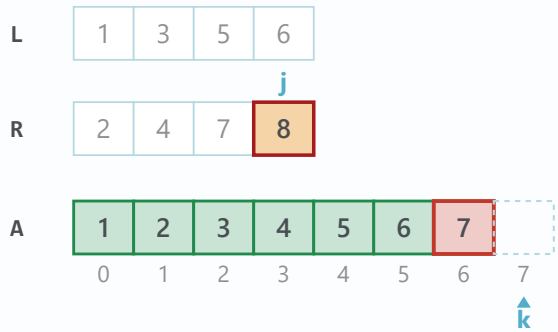
L[3] = 6 ≤ R[2] = 7 → A[5] = 6; i++, k++.

DEĞİŞKENLER

i = 4 j = 2 k = 6

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
5    while (i < L.length && j < R.length) {
6      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
▶    while (j < R.length) A[k++] = R[j++];
11 }
```



L bitti (i = 4); kalan R öğeleri sırayla kopyalanır: A[6] = 7.

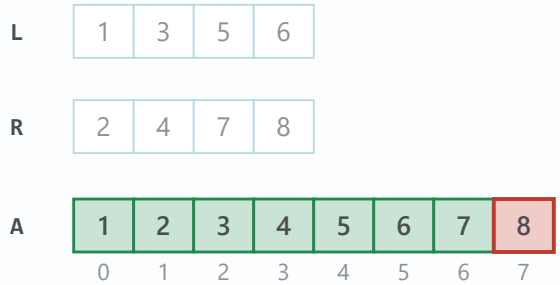
Adım 8 / 9

DEĞİŞKENLER

`i = 4` `j = 3` `k = 7`

birlestir(): İki Sıralı Diziyi Birleştir

```
1  void birlestir(int[] A, int p, int q, int r) {
2    int[] L = Arrays.copyOfRange(A, p, q + 1);
3    int[] R = Arrays.copyOfRange(A, q + 1, r + 1);
4    int i = 0, j = 0, k = p;
5    while (i < L.length && j < R.length) {
6      if (L[i] <= R[j]) A[k++] = L[i++];
7      else              A[k++] = R[j++];
8    }
9    while (i < L.length) A[k++] = L[i++];
▶    while (j < R.length) A[k++] = R[j++];
11 }
```



A[7] = 8. Birleştirme bitti: [1, 2, 3, 4, 5, 6, 7, 8] (8 öğe, en fazla 7 karşılaştırma).

Adım 9 / 9

DEĞİŞKENLER

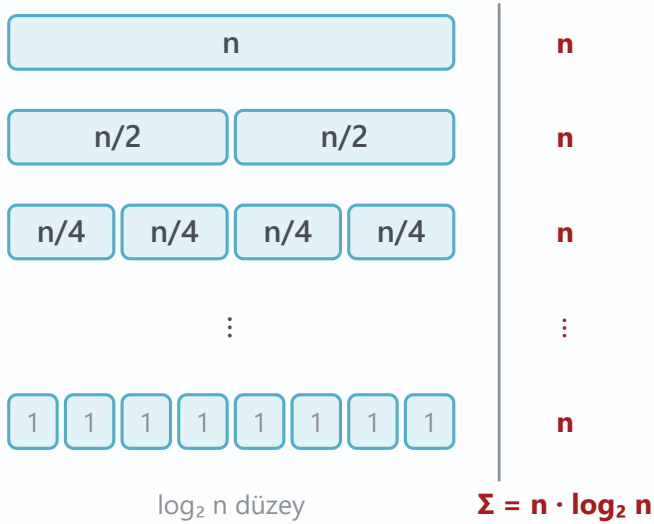
`i = 4` `j = 4` `k = 8`

Merge Sort: Analiz

Yineleme bağıntısı (recurrence):

$$T(n) = 2\,T(n/2) + \Theta(n)$$

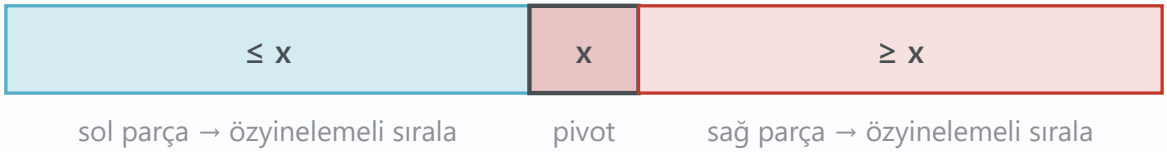
- Her düzeyde toplam birleştirme işi **$\Theta(n)$**.
- Düzey sayısı **$\log_2 n$** (n'yi 1'e kadar yarılamak).
- Toplam: Θ(n log n) — **en iyi = ortalama = en kötü**.
- Ek bellek Θ(n) (yardımcı diziler) + Θ(log n) özyineleme yığını.
- `L[i] <= R[j]` : eşitlikte soldan alınır → **kararlı**.



Ana teoremle: a = 2, b = 2, $f(n) = \Theta(n) = \Theta(n^{\log_2 2})$ → 2. durum → $T(n) = \Theta(n \log n)$.

Hızlı Sıralama (Quick Sort)

- C. A. R. Hoare (1959). Yine **böl ve fethet**, ama işin çoğu **bölmede** yapılır.
- Bir **pivot** öğe seçilir.
- Dizi pivot etrafında iki parçaya bölünür (**bölümleme**, partitioning): solda ≤ x, sağda ≥ x.
- İki alt parça özyinelemeli olarak sıralanır. Birleştirme adımına **gerek yoktur**.



Bölümlemeden sonra pivot **kesin yerindedir**: sıralı dizide de tam o indiste duracaktır.

Bölümleme (Partitioning) Prosedürü

Lomuto: pivot = son öğe

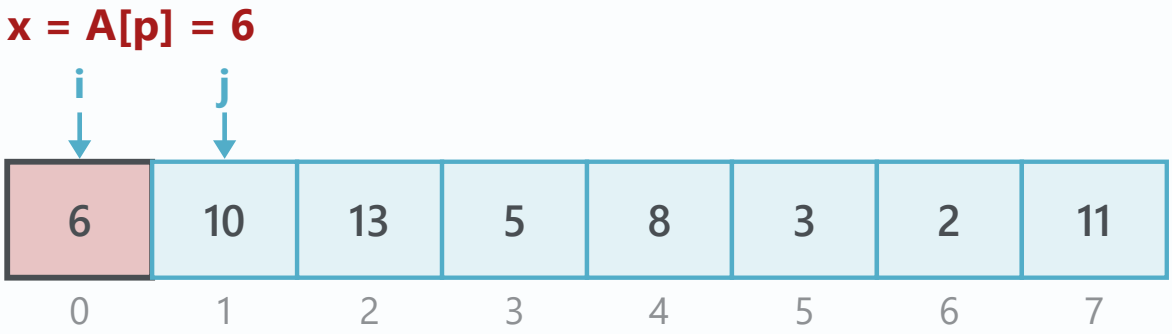
```
parcala(A, p, q):
  pivot = A[q]          // pivot seç
  i = p - 1             // ≤ pivot bölgesinin sonu
  for j = p to q - 1:
    if A[j] <= pivot:
      i = i + 1
      takas(A[i], A[j])
  takas(A[i + 1], A[q])  // pivotu yerleştir
  return i + 1           // pivotun indisi
```

Aynı fikir: pivot = ilk öğe

```
PARCALA(A, p, q):
  x = A[p]              // pivot
  i = p
  for j = p + 1 to q:
    if A[j] <= x:
      i = i + 1
      takas(A[i], A[j])
  takas(A[p], A[i])
  return i
```

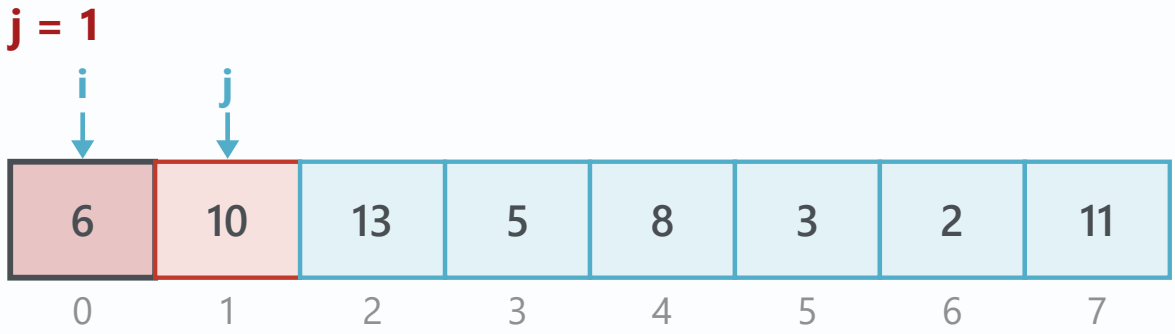
Değişmez (invariant): A[p..i] ≤ pivot, A[i+1..j−1] > pivot, A[j..] henüz incelenmedi. Tek geçiş: **Θ(n)** zaman, O(1) ek bellek.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



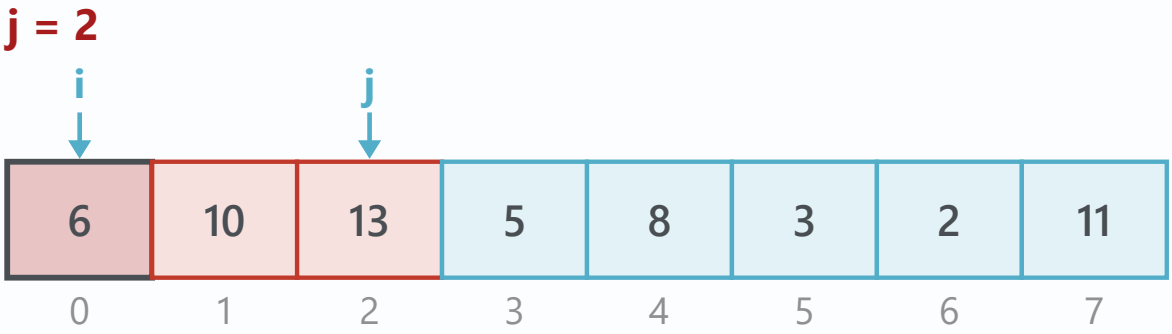
Pivot x = A[p] = 6 (sarı). i = p; j, p + 1'den başlar. Mavi: ≤ x, kırmızı: > x, beyaz: henüz bakılmadı.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



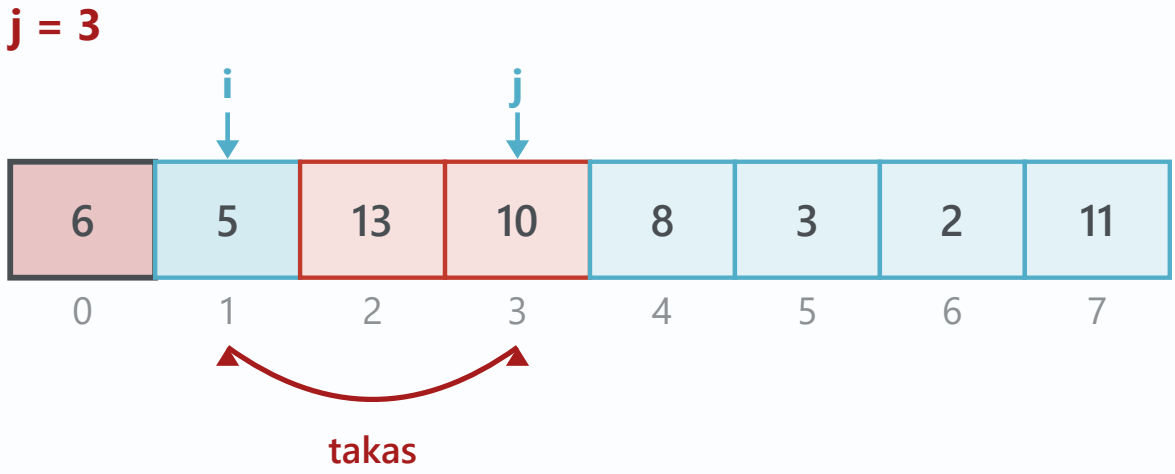
A[j] = 10 > 6 → yalnızca j ilerler; kırmızı bölge büyür.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



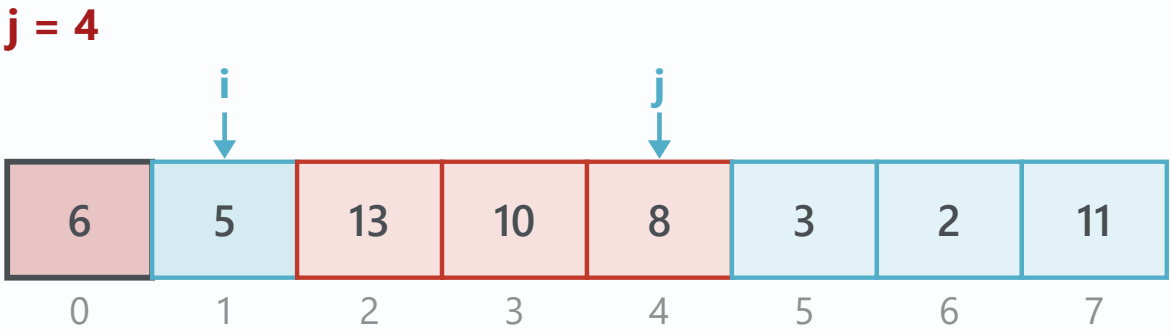
A[j] = 13 > 6 → yalnızca j ilerler; kırmızı bölge büyür.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



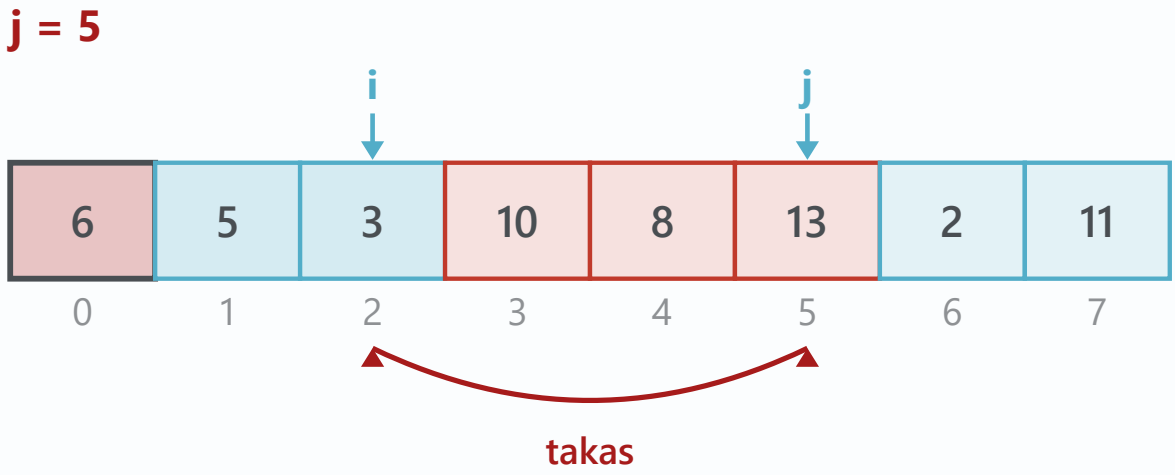
A[j] = 5 ≤ 6 → i = 1; A[i] ↔ A[j]: [6, 5, 13, 10, 8, 3, 2, 11]. Mavi bölge bir büyüdü.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



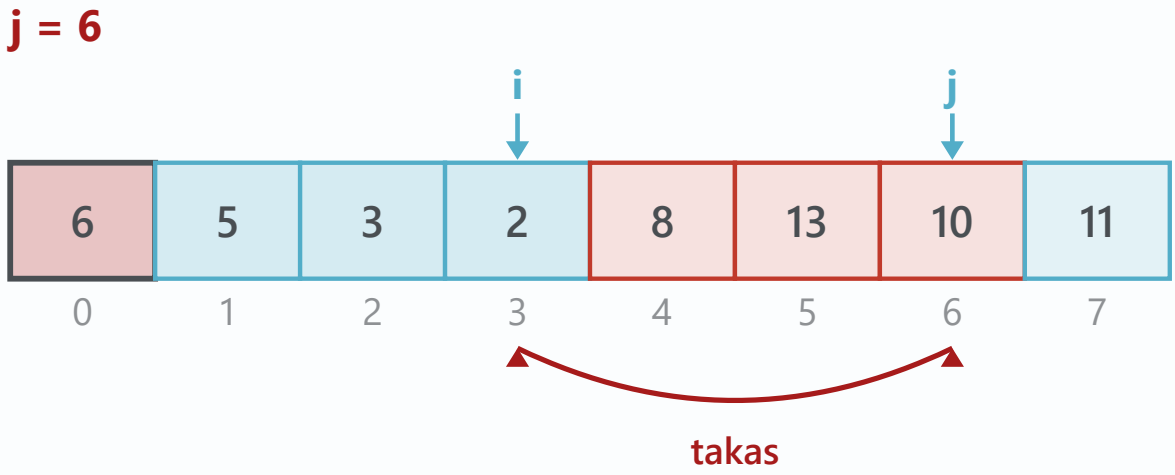
A[j] = 8 > 6 → yalnızca j ilerler; kırmızı bölge büyür.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



A[j] = 3 ≤ 6 → i = 2; A[i] ↔ A[j]: [6, 5, 3, 10, 8, 13, 2, 11]. Mavi bölge bir büyüdü.

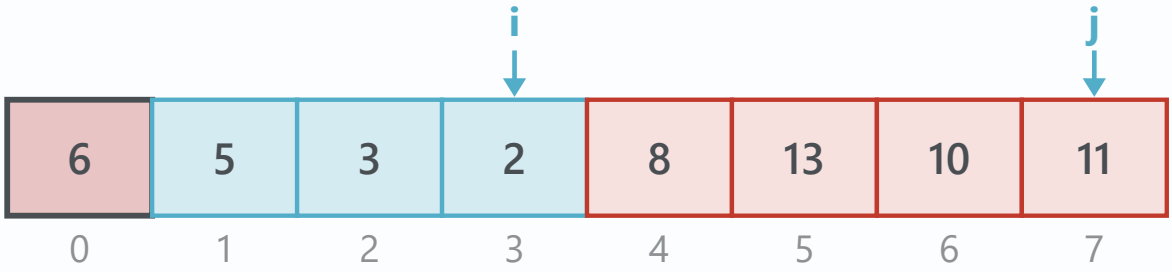
Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



A[j] = 2 ≤ 6 → i = 3; A[i] ↔ A[j]: [6, 5, 3, 2, 8, 13, 10, 11]. Mavi bölge bir büyüdü.

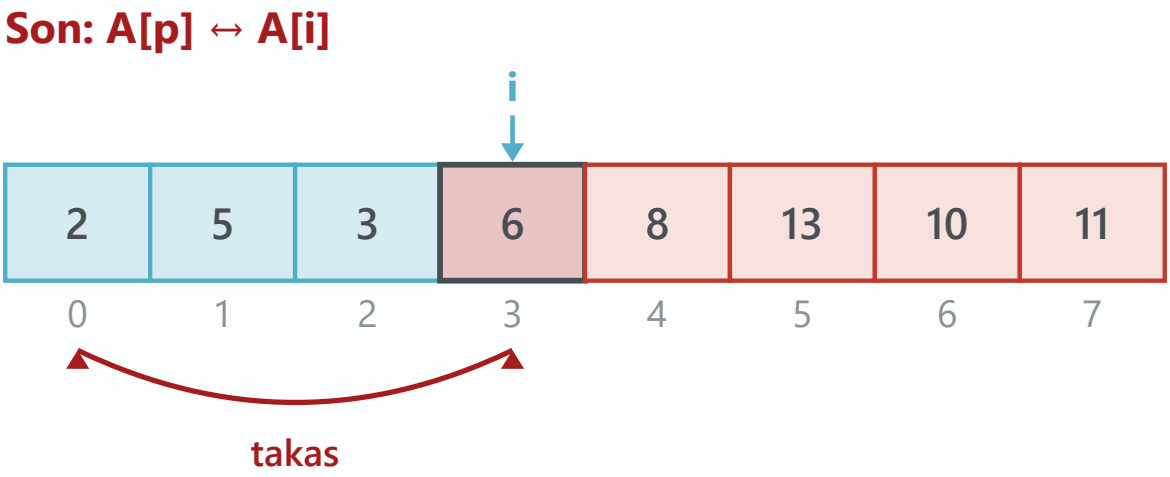
Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]

j = 7



A[j] = 11 > 6 → yalnızca j ilerler; kırmızı bölge büyür.

Parçalara Ayırma: [6, 10, 13, 5, 8, 3, 2, 11]



Döngü bitti. Pivot, mavi bölgenin son öğesiyle yer değiştirir: [2, 5, 3, 6, 8, 13, 10, 11]. Pivot 6 artık **kesin yerinde** (indis 3); solu ≤ 6, sağı ≥ 6.

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
▶    int pivot = dizi[ust];
▶    int i = alt - 1;
4    for (int j = alt; j < ust; j++) {
5      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```

dizi[]	2	8	7	1	3	5	6	4
	0	1	2	3	4	5	6	7

pivot = dizi[ust] = 4; i = alt − 1 = −1 (≤ pivot bölgesi boş).

Adım 1 / 13

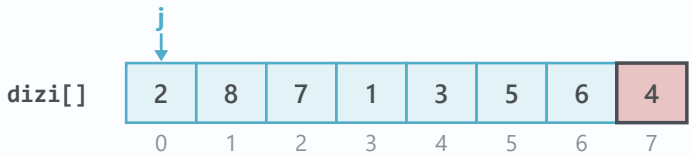
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = -1` `j = –`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 0: dizi[0] = 2 ≤ 4? **Evet**.

Adım 2 / 13

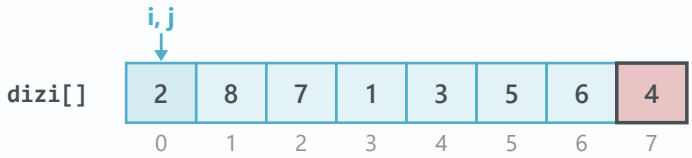
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = -1` `j = 0`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
4    for (int j = alt; j < ust; j++) {
5      if (dizi[j] <= pivot) {
▶        i++;
▶        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



i = 0; takas(dizi, 0, 0) → [2, 8, 7, 1, 3, 5, 6, 4] (kendisiyle).

Adım 3 / 13

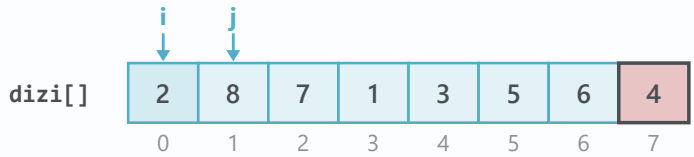
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 0` `j = 0`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 1: dizi[1] = 8 ≤ 4? Hayır → j ilerler.

Adım 4 / 13

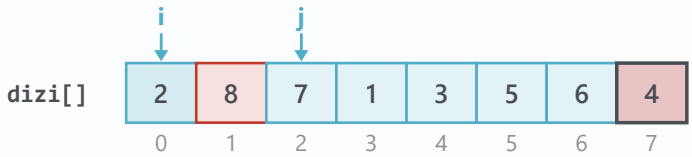
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 0` `j = 1`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 2: dizi[2] = 7 ≤ 4? Hayır → j ilerler.

Adım 5 / 13

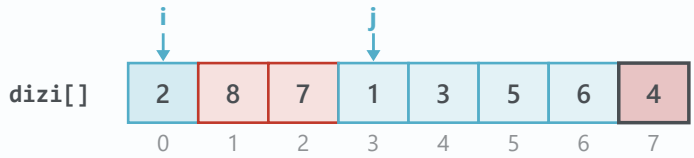
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 0` `j = 2`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 3: dizi[3] = 1 ≤ 4? **Evet**.

Adım 6 / 13

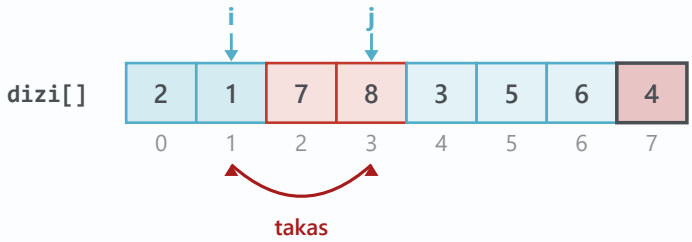
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 0` `j = 3`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
4    for (int j = alt; j < ust; j++) {
5      if (dizi[j] <= pivot) {
▶        i++;
▶        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



i = 1; takas(dizi, 1, 3) → [2, 1, 7, 8, 3, 5, 6, 4].

Adım 7 / 13

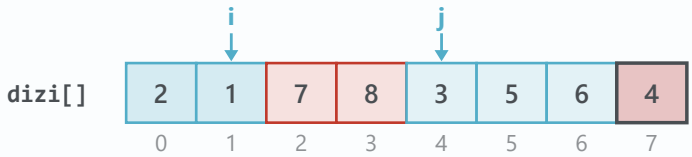
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 1` `j = 3`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 4: dizi[4] = 3 ≤ 4? **Evet**.

Adım 8 / 13

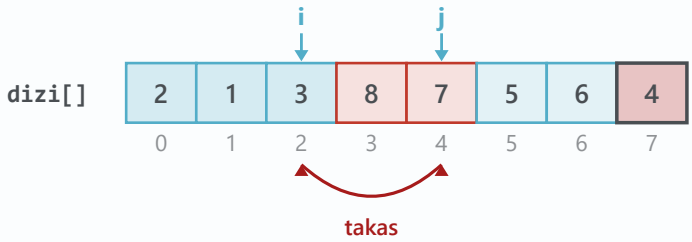
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 1` `j = 4`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
4    for (int j = alt; j < ust; j++) {
5      if (dizi[j] <= pivot) {
▶        i++;
▶        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



i = 2; takas(dizi, 2, 4) → [2, 1, 3, 8, 7, 5, 6, 4].

Adım 9 / 13

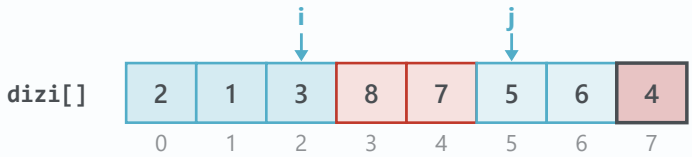
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 2` `j = 4`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 5: dizi[5] = 5 ≤ 4? Hayır → j ilerler.

Adım 10 / 13

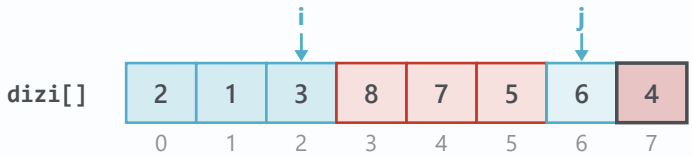
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 2` `j = 5`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
▶    for (int j = alt; j < ust; j++) {
▶      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



j = 6: dizi[6] = 6 ≤ 4? Hayır → j ilerler.

Adım 11 / 13

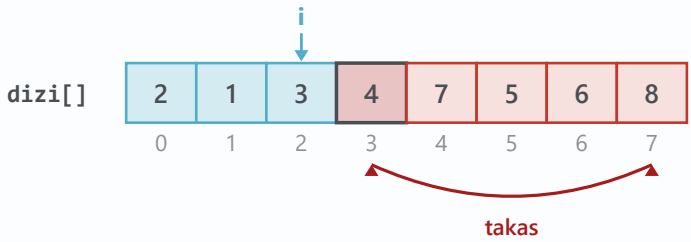
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 2` `j = 6`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
4    for (int j = alt; j < ust; j++) {
5      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
▶    takas(dizi, i + 1, ust);
11   return i + 1;
12 }
```



Döngü bitti. takas(dizi, i+1 = 3, ust = 7) → [2, 1, 3, 4, 7, 5, 6, 8].

Adım 12 / 13

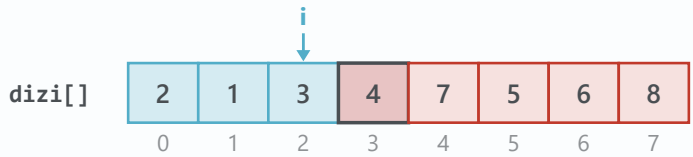
DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 2` `j = 7`

ÇIKTI

parcala(): Kodu İzle, dizi = [2, 8, 7, 1, 3, 5, 6, 4]

```
1  int parcala(int[] dizi, int alt, int ust)
   {
2    int pivot = dizi[ust];
3    int i = alt - 1;
4    for (int j = alt; j < ust; j++) {
5      if (dizi[j] <= pivot) {
6        i++;
7        takas(dizi, i, j);
8      }
9    }
10   takas(dizi, i + 1, ust);
▶    return i + 1;
12 }
```



`return 3` : pivot 4 kesin yerinde; sol [0..2] ≤ 4, sağ [4..7] > 4.

Adım 13 / 13

DEĞİŞKENLER

`alt = 0` `ust = 7` `pivot = 4` `i = 2` `j = 7`

ÇIKTI

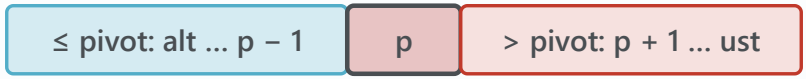
```
parcala → 3
```

QuickSort: Özyinelemeli Yapı

```
HIZLI-SIRALA(A, p, r)
  if p < r
    q = PARCALA(A, p, r)
    HIZLI-SIRALA(A, p, q – 1)
    HIZLI-SIRALA(A, q + 1, r)

İlk çağrı: HIZLI-SIRALA(A, 1, n)
```

```
void hizliSirala(int[] dizi, int alt, int ust) {
  if (alt < ust) {
    int p = parcala(dizi, alt, ust);
    hizliSirala(dizi, alt, p - 1);
    hizliSirala(dizi, p + 1, ust);
  }
}
```



hizliSirala(dizi, alt, p – 1) yerinde **hizliSirala(dizi, p + 1, ust)**

- Pivot (indis q) iki özyinelemeli çağrının **hiçbirine** dahil edilmez: yerine oturmuştur.
- Taban durumu: 0 ya da 1 öğeli alt dizi (p ≥ r) zaten sıralıdır.
- Java sürümü 0-tabanlıdır: ilk çağrı `hizliSirala(dizi, 0, n – 1)`.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

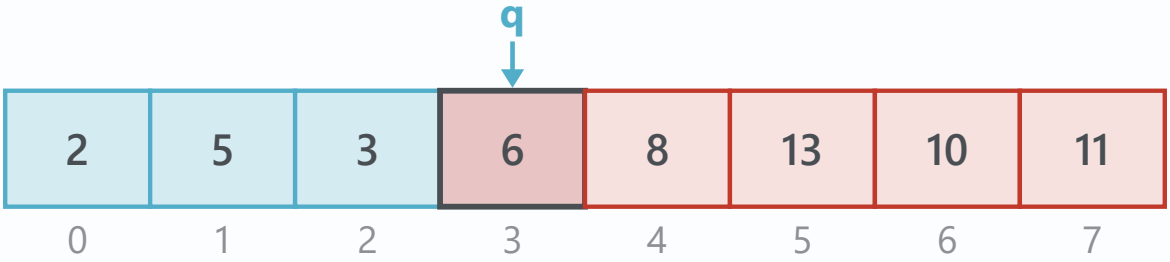
HIZLI-SIRALA(A, 0, 7)



Dizi [6, 10, 13, 5, 8, 3, 2, 11]. Her çağrı alt diziyi pivot = ilk öğe ile parçalar, sonra iki tarafı özyinelemeli sıralar.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

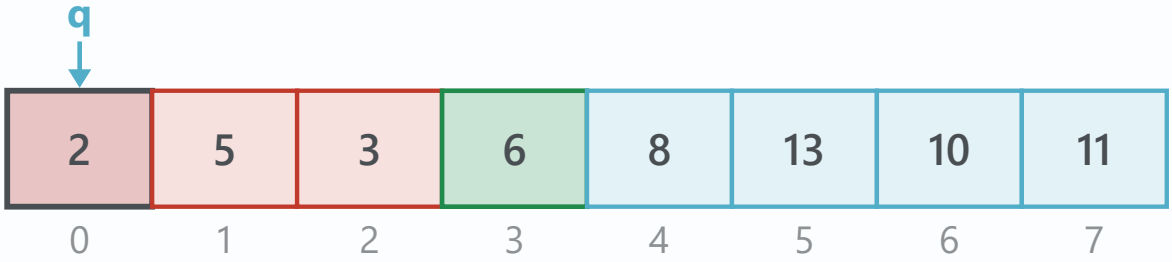
HIZLI-SIRALA(A, 0, 7)



Derinlik 0: [6, 10, 13, 5, 8, 3, 2, 11] pivot 6 ile bölümlenir → pivot indis 3'e oturur; sonra [0..2] ve [4..7] sıralanır.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

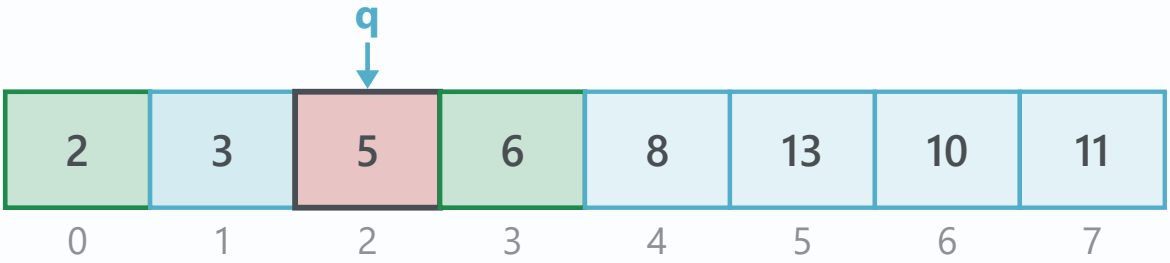
HIZLI-SIRALA(A, 0, 2)



Derinlik 1: [2, 5, 3] pivot 2 ile bölümlenir → pivot indis 0'a oturur; sonra [0..-1] ve [1..2] sıralanır.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

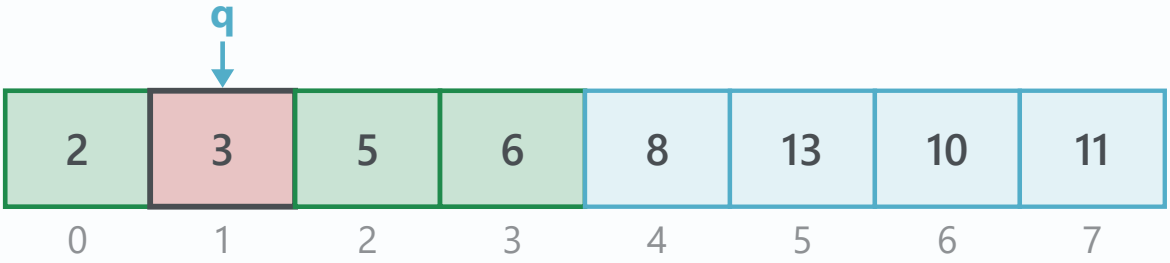
HIZLI-SIRALA(A, 1, 2)



Derinlik 2: [5, 3] pivot 5 ile bölümlenir → pivot indis 2'ye oturur; sonra [1..1] ve [3..2] sıralanır.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

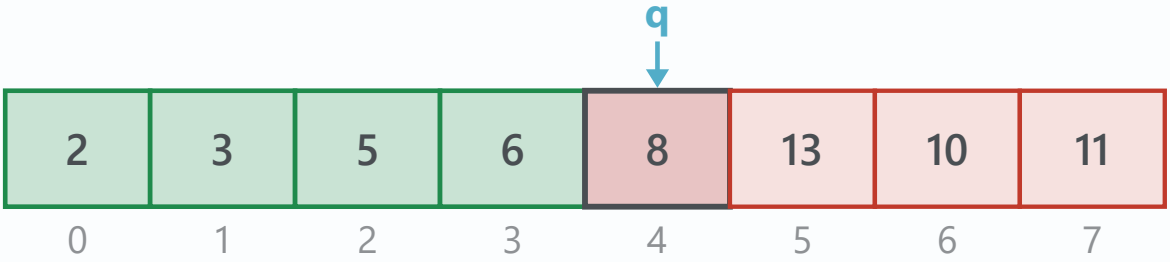
HIZLI-SIRALA(A, 1, 1)



Derinlik 3: [1..1] tek öğeli (3) → zaten sıralı (p < r değil).

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

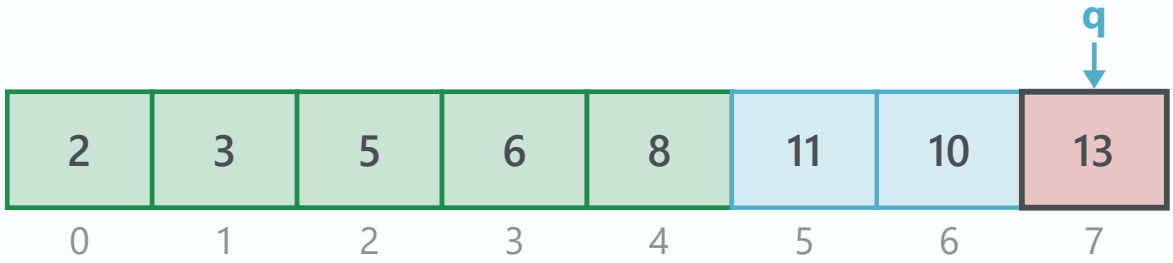
HIZLI-SIRALA(A, 4, 7)



Derinlik 1: [8, 13, 10, 11] pivot 8 ile bölümlenir → pivot indis 4'e oturur; sonra [4..3] ve [5..7] sıralanır.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

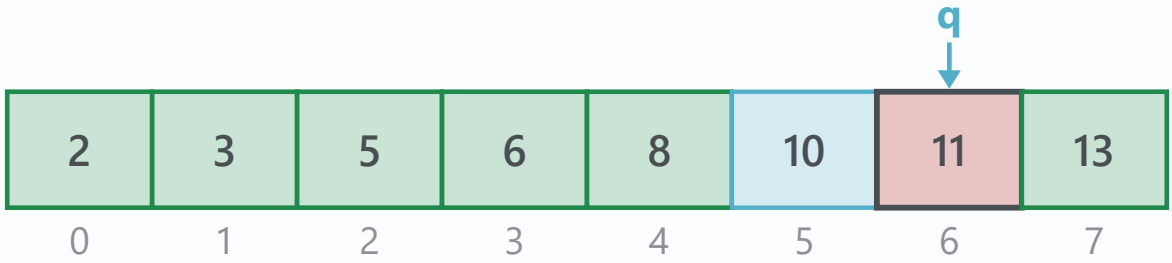
HIZLI-SIRALA(A, 5, 7)



Derinlik 2: [13, 10, 11] pivot 13 ile bölümlenir → pivot indis 7'ye oturur; sonra [5..6] ve [8..7] sıralanır.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

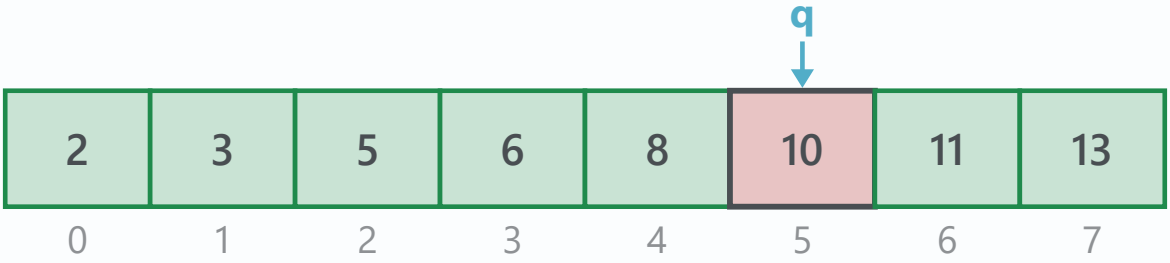
HIZLI-SIRALA(A, 5, 6)



Derinlik 3: [11, 10] pivot 11 ile bölümlenir → pivot indis 6'ya oturur; sonra [5..5] ve [7..6] sıralanır.

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

HIZLI-SIRALA(A, 5, 5)



Derinlik 4: [5..5] tek öğeli (10) → zaten sıralı (p < r değil).

QuickSort Adım Adım: [6, 10, 13, 5, 8, 3, 2, 11]

Sonuç



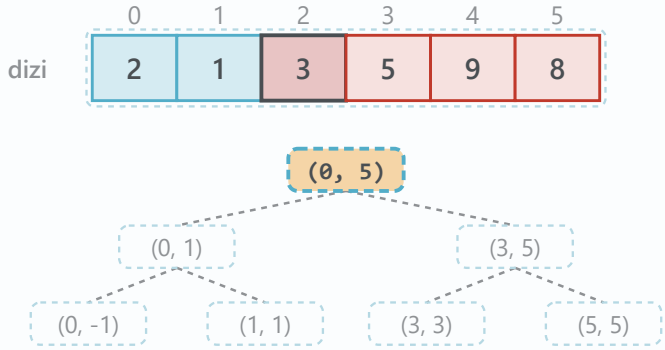
Tüm çağrılar bitti; birleştirme adımı yok, dizi yerinde sıralandı: [2, 3, 5, 6, 8, 10, 11, 13].

hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
▶     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

`hizliSirala(0, 5)`



`hizliSirala(0, 5)` : alt < ust. `parcala` pivot = 3 (son öğe) ile böler: [2, 1] solda, [5, 9, 8] sağda; pivot **p = 2** indisine oturur. Sonra satır 4: sol parça.

Adım 1 / 8

DEĞİŞKENLER

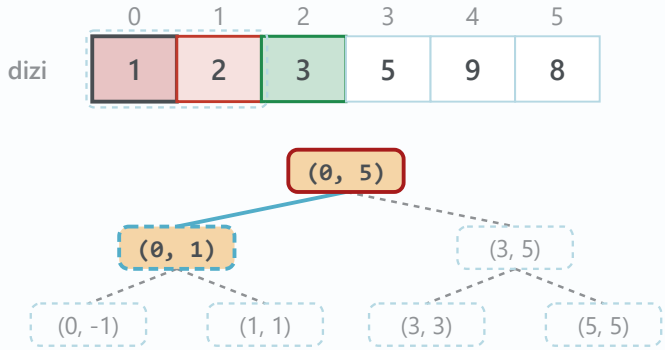
`alt = 0` `ust = 5` `p = 2`

hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
▶     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

- **hizliSirala(0, 1)**
- hizliSirala(0, 5)



`hizliSirala(0, 1)` : alt < ust. `parcala` pivot = 1 (son öğe) ile böler: [] solda, [2] sağda; pivot **p = 0** indisine oturur.

Adım 2 / 8

DEĞİŞKENLER

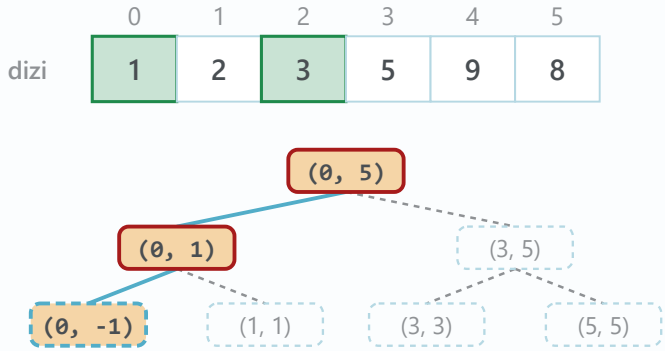
`alt = 0` `ust = 1` `p = 0`

hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
3     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

- **hizliSirala(0, -1)**
- hizliSirala(0, 1)
- hizliSirala(0, 5)



`hizliSirala(0, -1)` : alt > ust, boş parça → hemen döner (taban durum).

Adım 3 / 8

DEĞİŞKENLER

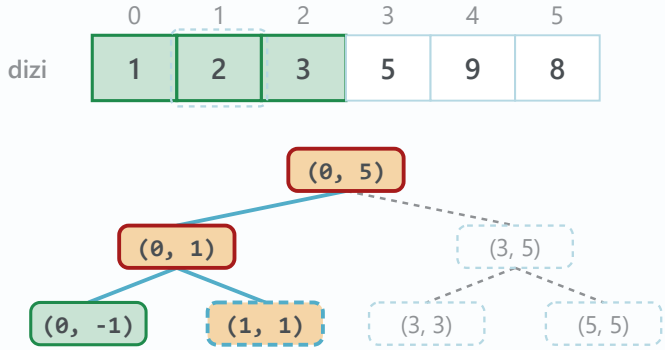
`alt = 0` `ust = -1` `p = –`

hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
3     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

- **hizliSirala(1, 1)**
- hizliSirala(0, 1)
- hizliSirala(0, 5)



`hizliSirala(1, 1)` : tek öğeli parça [2] zaten sıralı → döner. (0, 1) çağrısı biter; satır 5: (3, 5).

Adım 4 / 8

DEĞİŞKENLER

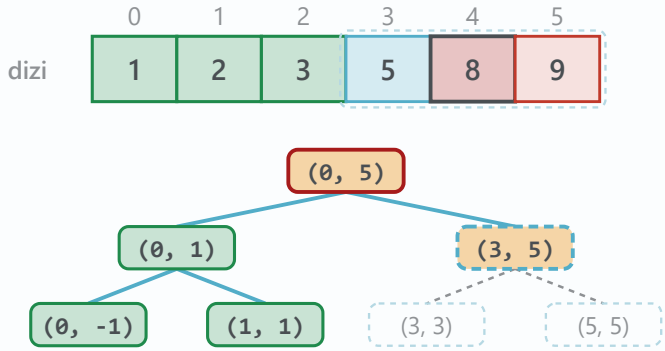
`alt = 1` `ust = 1` `p = –`

hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
▶     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

- **hizliSirala(3, 5)**
- hizliSirala(0, 5)



`hizliSirala(3, 5)` : alt < ust. `parcala` pivot = 8 (son öğe) ile böler: [5] solda, [9] sağda; pivot **p = 4** indisine oturur.

Adım 5 / 8

DEĞİŞKENLER

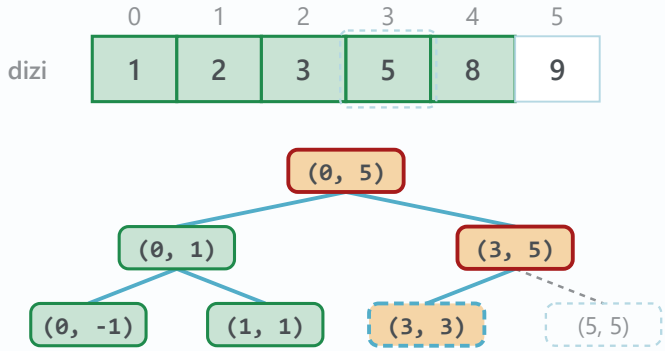
`alt = 3` `ust = 5` `p = 4`

hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
3     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

- **hizliSirala(3, 3)**
- hizliSirala(3, 5)
- hizliSirala(0, 5)



`hizliSirala(3, 3)` : tek öğeli parça [5] zaten sıralı → döner.

Adım 6 / 8

DEĞİŞKENLER

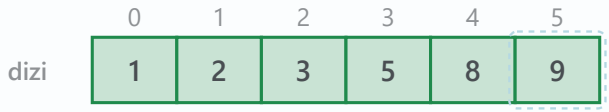
`alt = 3` `ust = 3` `p = –`

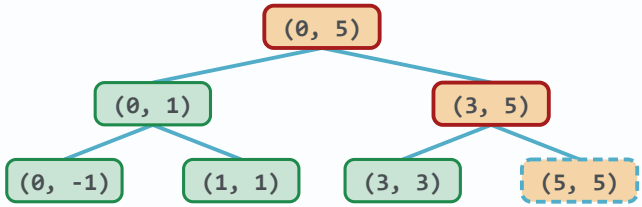
hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
▶   if (alt < ust) {
3     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
5     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

- **hizliSirala(5, 5)**
- hizliSirala(3, 5)
- hizliSirala(0, 5)





`hizliSirala(5, 5)` : tek öğeli parça [9] zaten sıralı → döner.

Adım 7 / 8

DEĞİŞKENLER

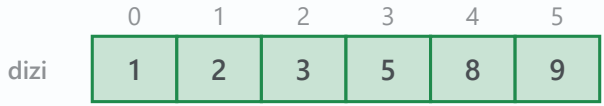
`alt = 5` `ust = 5` `p = –`

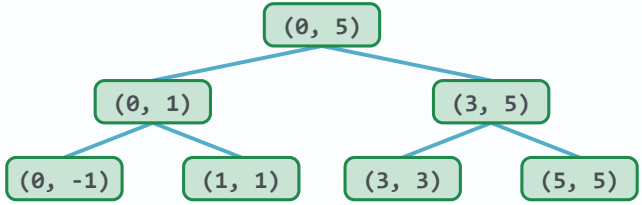
hizliSirala: Çağrıları İzle, dizi = [5, 2, 8, 1, 9, 3]

```
1 void hizliSirala(int[] dizi, int alt, int
  ust) {
2   if (alt < ust) {
3     int p = parcala(dizi, alt, ust);
4     hizliSirala(dizi, alt, p - 1);
▶     hizliSirala(dizi, p + 1, ust);
6   }
7 }
```

ÇAĞRI YIĞINI

`hizliSirala(0, 5)`





Tüm çağrılar döndü: dizi = [1, 2, 3, 5, 8, 9]. 3 `parcala` ve 4 taban çağrısı; pivotlar hiçbir alt çağrıya girmedi, her biri kendi yerine oturdu.

Adım 8 / 8

DEĞİŞKENLER

`alt = 0` `ust = 5` `p = 2`

HIZLI SIRALAMA

QuickSort: Analiz ve Pivot Seçimi

Durum	Ne zaman?	Yineleme	Zaman
En iyi	pivot her seferinde ortanca	$T(n) = 2T(n/2) + \Theta(n)$	$\Theta(n \log n)$
Ortalama	rastgele girdi / rastgele pivot	beklenen $\approx 1{,}39\, n \log_2 n$ karşılaştırma	$\Theta(n \log n)$
En kötü	pivot hep en küçük/en büyük (ör. *sıralı* dizi + son öğe pivot)	$T(n) = T(n-1) + \Theta(n)$	$\Theta(n^2)$

Rastgele pivot
Pivotu rastgele seçmek kötü girdiyi olanaksız kılar; beklenen süre her girdide $\Theta(n \log n)$.

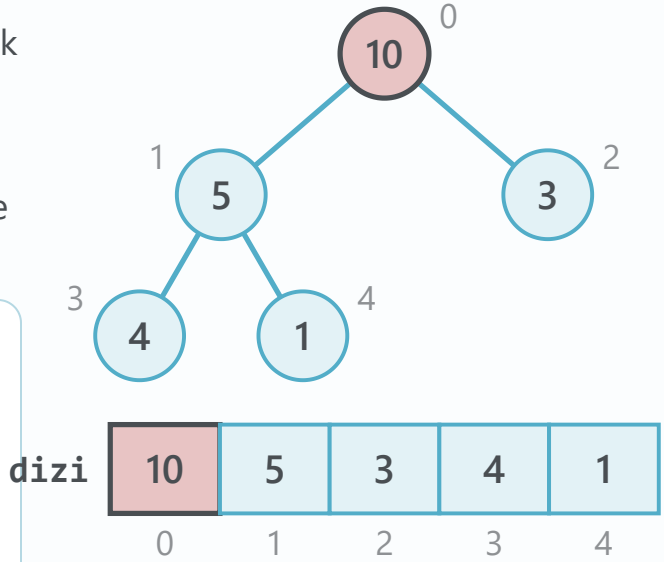
Üçün ortancası
İlk, orta ve son öğenin ortancası pivot olur; sıralı girdide en kötü durumu önler.

Küçük parçalar
$n \lesssim 16$ olunca eklemeli sıralamaya geçmek ve küçük tarafa önce inmek (yığın $O(\log n)$) pratikte hızlandırır.

Yerinde ($O(\log n)$ ortalama yığın), **kararsız**. Önbellek dostu olduğu için pratikte genellikle en hızlı karşılaştırmalı sıralamadır (Java `Arrays.sort(int[])` : çift pivotlu quicksort).

Heap Sort

- **Heap** veri yapısını kullanarak sıralar; ortalama ve en kötü durumda **O(n log n)**.
- **Max-heap**: her düğüm çocuklarından büyük ya da eşittir → en büyük öğe **köktedir**.
- Ağaç ayrı bir yapı değildir; **dizinin kendisidir**: dizi[i]'nin çocukları `2i + 1` ve `2i + 2`, ebeveyni `(i − 1) / 2`.



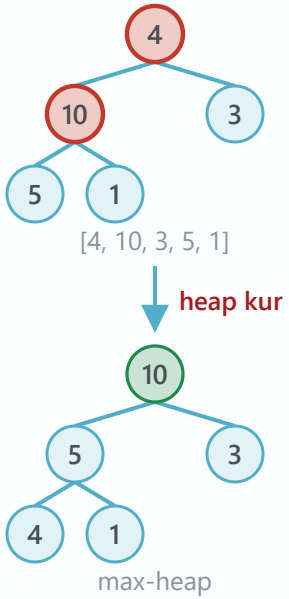
İşleyiş

1. Dizi max-heap yapısına dönüştürülür.
2. En büyük öğe (kök) dizinin sonuna yerleştirilir.
3. Kalan dizi tekrar max-heap yapılır (kök aşağı süzülür).
4. Tüm öğeler sıralanana kadar tekrarlanır.

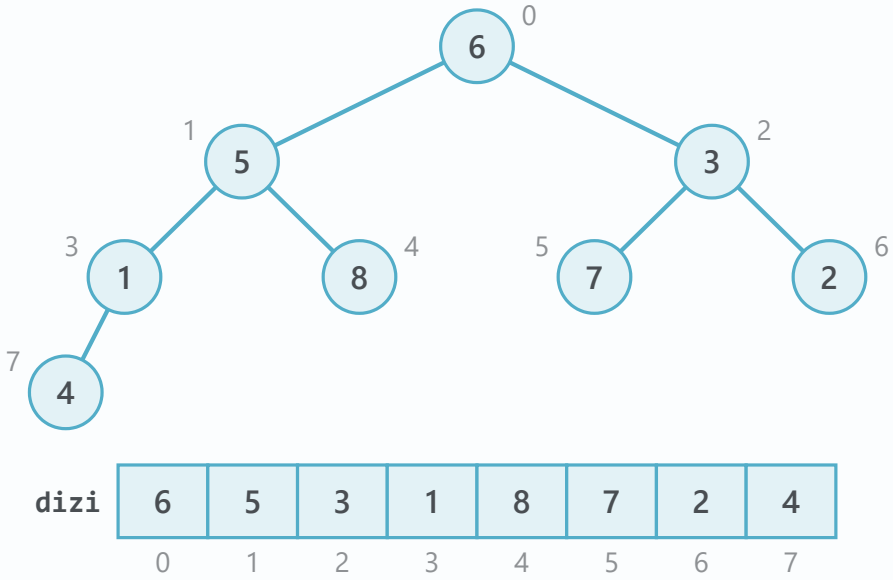
HEAP SORT

Örnek: [4, 10, 3, 5, 1]

Adım	Dizi (heap | sıralı)	Açıklama
Dizi	`[4, 10, 3, 5, 1]`	
Max-heap oluştur	`[10, 5, 3, 4, 1]`	kök = en büyük
Kök ↔ dizi[4]	`[1, 5, 3, 4] | [10]`	10 sıralı bölgeye
Yeniden max-heap yap	`[5, 4, 3, 1] | [10]`	kök aşağı süzüldü
Kök ↔ dizi[3]	`[1, 4, 3] | [5, 10]`	5 sıralı bölgeye
Yeniden max-heap yap	`[4, 1, 3] | [5, 10]`	kök aşağı süzüldü
Kök ↔ dizi[2]	`[3, 1] | [4, 5, 10]`	4 sıralı bölgeye
Yeniden max-heap yap	`[3, 1] | [4, 5, 10]`	kök aşağı süzüldü
Kök ↔ dizi[1]	`[1] | [3, 4, 5, 10]`	3 sıralı bölgeye
Sonuç	`[1, 3, 4, 5, 10]`	

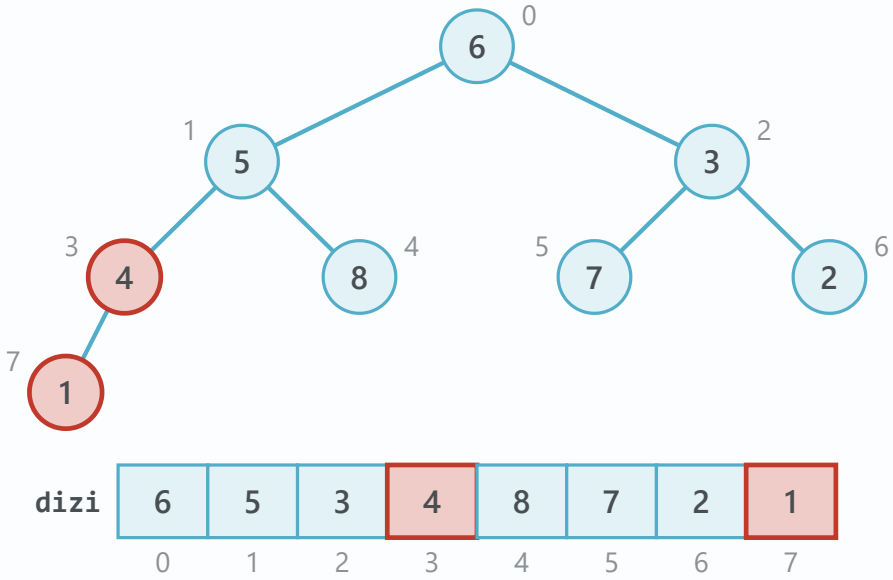


1. Aşama: Max-Heap Kurma — [6, 5, 3, 1, 8, 7, 2, 4]



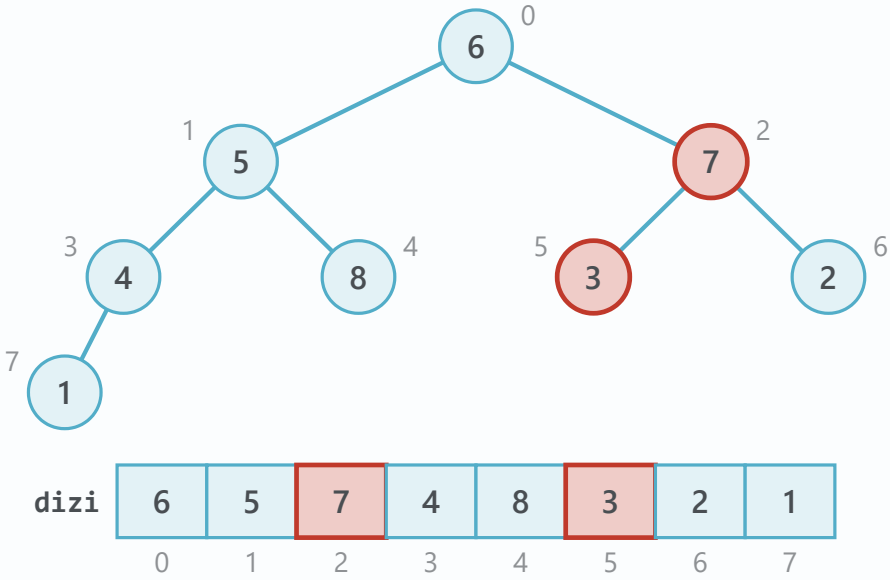
Dizi [6, 5, 3, 1, 8, 7, 2, 4] tam ikili ağaç olarak düşünülür: dizi[i]'nin çocukları 2i+1 ve 2i+2. Henüz heap değil.

1. Aşama: Max-Heap Kurma — [6, 5, 3, 1, 8, 7, 2, 4]



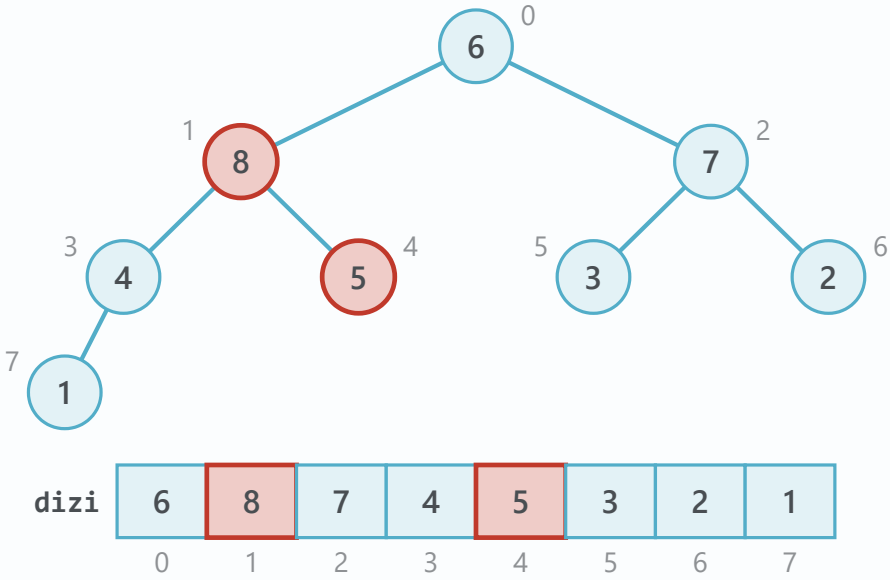
yiginlastir(i = 3): 1 çocuklarından küçük → yer değiştir (1 ↔ 4). Dizi: [6, 5, 3, 4, 8, 7, 2, 1].

1. Aşama: Max-Heap Kurma — [6, 5, 3, 1, 8, 7, 2, 4]



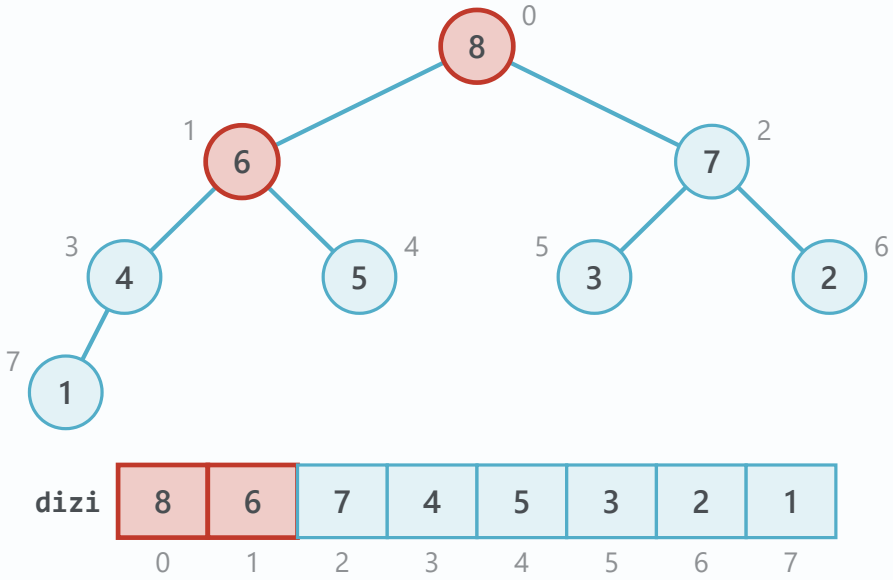
yiginlastir(i = 2): 3 çocuklarından küçük → yer değiştir (3 ↔ 7). Dizi: [6, 5, 7, 4, 8, 3, 2, 1].

1. Aşama: Max-Heap Kurma — [6, 5, 3, 1, 8, 7, 2, 4]



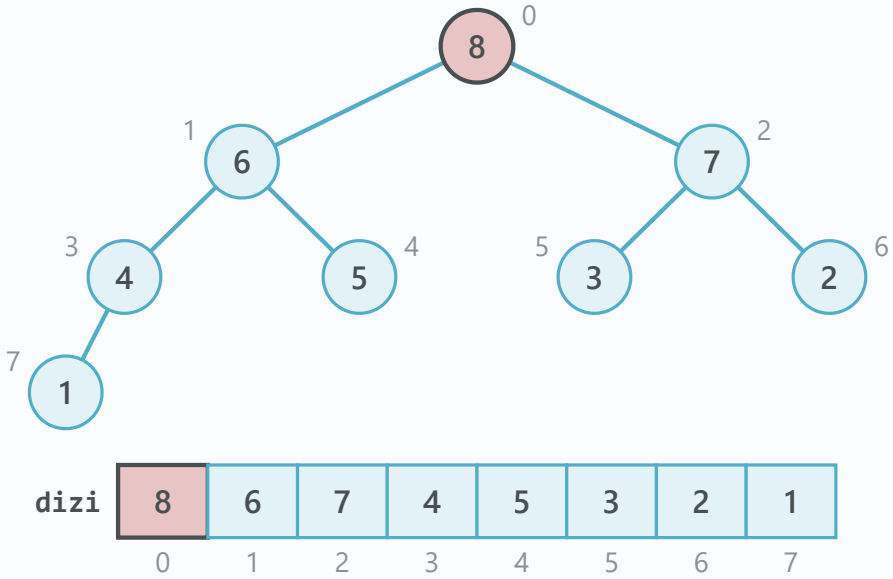
yiginlastir(i = 1): 5 çocuklarından küçük → yer değiştir (5 ↔ 8). Dizi: [6, 8, 7, 4, 5, 3, 2, 1].

1. Aşama: Max-Heap Kurma — [6, 5, 3, 1, 8, 7, 2, 4]



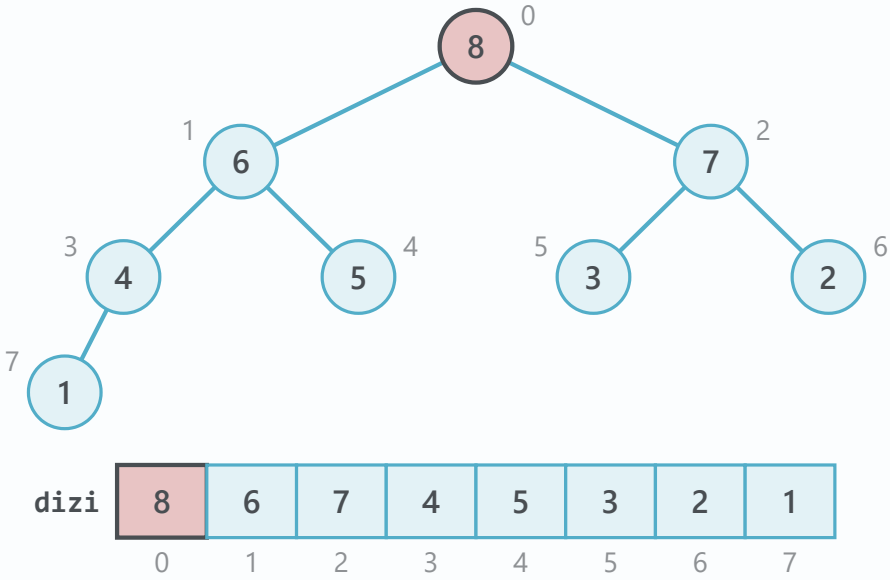
yiginlastir(i = 0): 6 çocuklarından küçük → yer değiştir (6 ↔ 8). Dizi: [8, 6, 7, 4, 5, 3, 2, 1].

1. Aşama: Max-Heap Kurma — [6, 5, 3, 1, 8, 7, 2, 4]



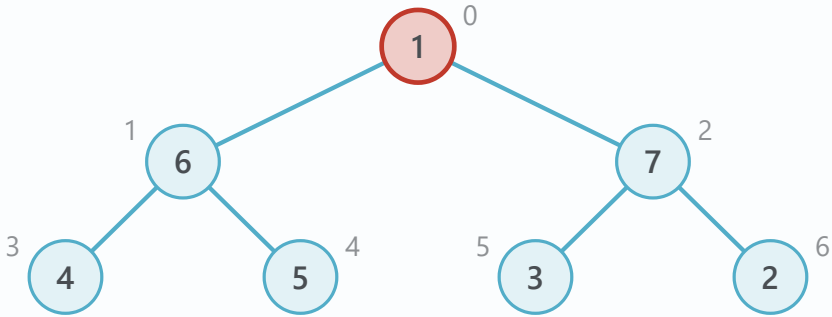
Yaprak olmayan tüm düğümler (i = 3 … 0) işlendi: **max-heap** [8, 6, 7, 4, 5, 3, 2, 1]. En büyük öğe (8) kökte.

2. Aşama: Kökü Sona Taşı, Süz



Max-heap: [8, 6, 7, 4, 5, 3, 2, 1]. Her turda kök (en büyük) heap'in son öğesiyle yer değiştirir ve heap bir küçülür.

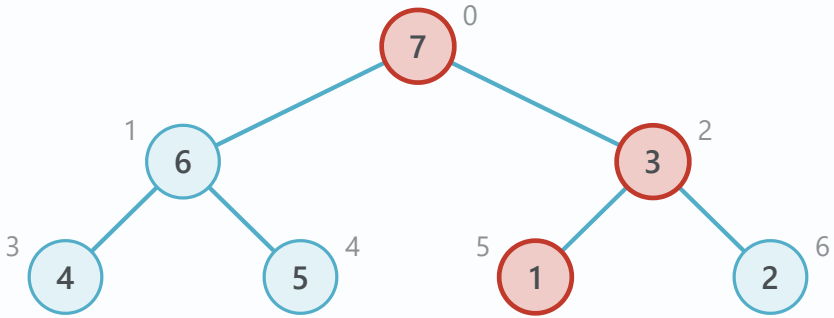
2. Aşama: Kökü Sona Taşı, Süz





Kök 8 ↔ dizi[7] = 1: **8** sıralı bölgeye (yeşil) geçti. Heap boyutu 7.

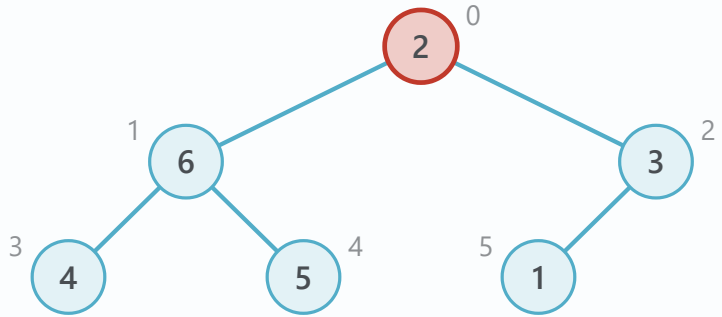
2. Aşama: Kökü Sona Taşı, Süz





yiginlastir(0) ile kök aşağı süzülür (1 ↔ 7; 1 ↔ 3) → heap: [7, 6, 3, 4, 5, 1, 2].

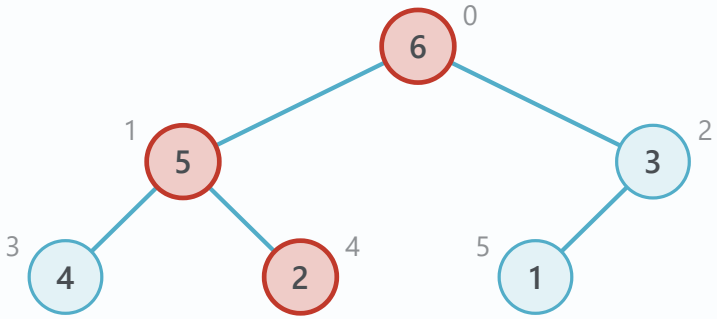
2. Aşama: Kökü Sona Taşı, Süz

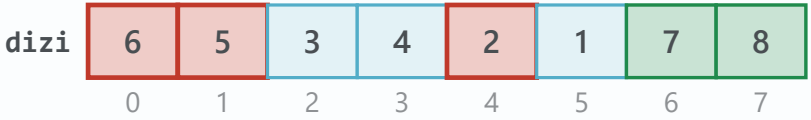




Kök 7 ↔ dizi[6] = 2: **7** sıralı bölgeye (yeşil) geçti. Heap boyutu 6.

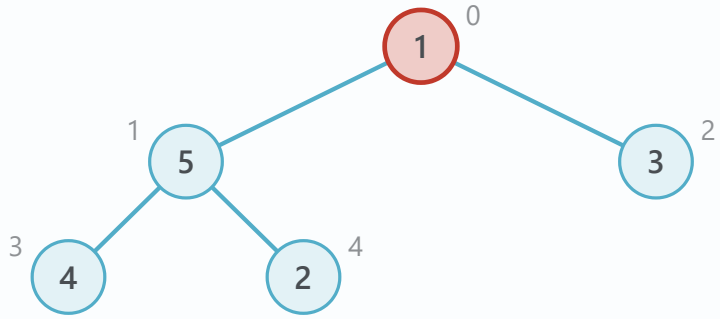
2. Aşama: Kökü Sona Taşı, Süz





yiginlastir(0) ile kök aşağı süzülür (2 ↔ 6; 2 ↔ 5) → heap: [6, 5, 3, 4, 2, 1].

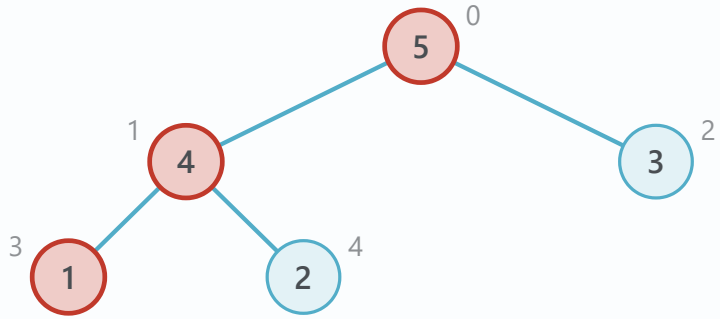
2. Aşama: Kökü Sona Taşı, Süz

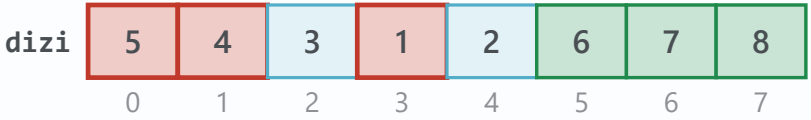




Kök 6 ↔ dizi[5] = 1: **6** sıralı bölgeye (yeşil) geçti. Heap boyutu 5.

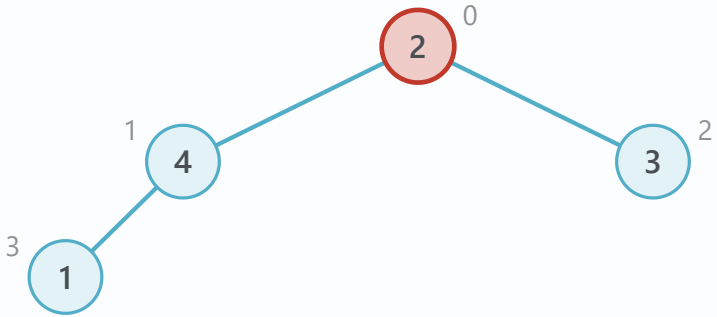
2. Aşama: Kökü Sona Taşı, Süz

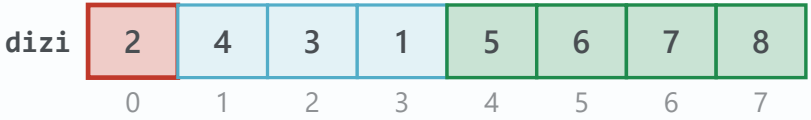




yiginlastir(0) ile kök aşağı süzülür (1 ↔ 5; 1 ↔ 4) → heap: [5, 4, 3, 1, 2].

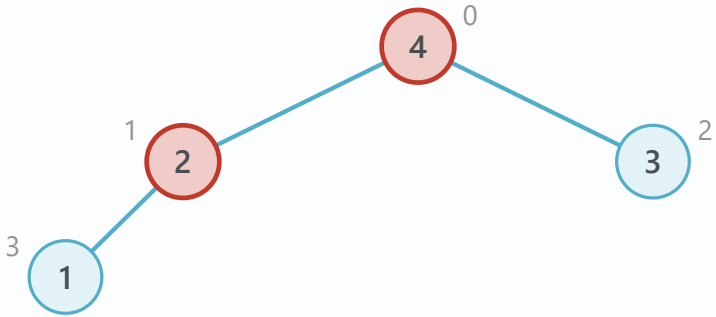
2. Aşama: Kökü Sona Taşı, Süz

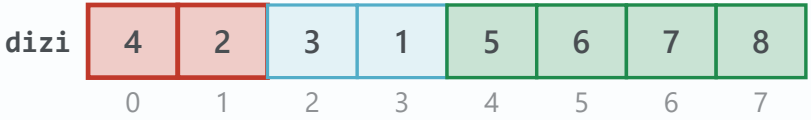




Kök 5 ↔ dizi[4] = 2: **5** sıralı bölgeye (yeşil) geçti. Heap boyutu 4.

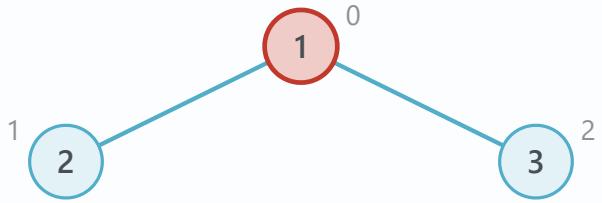
2. Aşama: Kökü Sona Taşı, Süz





yiginlastir(0) ile kök aşağı süzülür (2 ↔ 4) → heap: [4, 2, 3, 1].

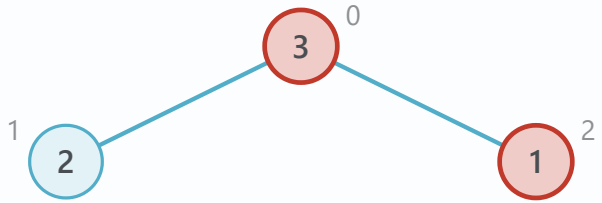
2. Aşama: Kökü Sona Taşı, Süz





Kök 4 ↔ dizi[3] = 1: **4** sıralı bölgeye (yeşil) geçti. Heap boyutu 3.

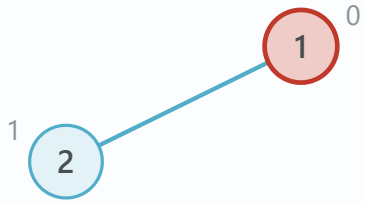
2. Aşama: Kökü Sona Taşı, Süz





yiginlastir(0) ile kök aşağı süzülür (1 ↔ 3) → heap: [3, 2, 1].

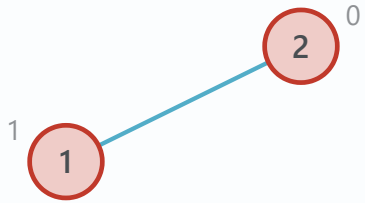
2. Aşama: Kökü Sona Taşı, Süz





Kök 3 ↔ dizi[2] = 1: **3** sıralı bölgeye (yeşil) geçti. Heap boyutu 2.

2. Aşama: Kökü Sona Taşı, Süz





yiginlastir(0) ile kök aşağı süzülür (1 ↔ 2) → heap: [2, 1].

2. Aşama: Kökü Sona Taşı, Süz





Kök 2 ↔ dizi[1] = 1: **2** sıralı bölgeye (yeşil) geçti. Heap boyutu 1.

2. Aşama: Kökü Sona Taşı, Süz





Heap'te tek öğe kaldı; o da yerinde. Sıralı dizi: [1, 2, 3, 4, 5, 6, 7, 8].

Heap Sort: Java Kodu ve Analiz

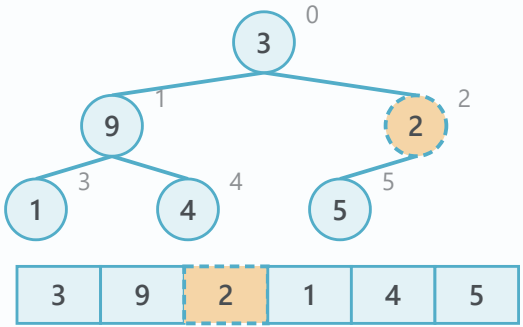
```java
void yiginSirala(int[] dizi) {
  int n = dizi.length;
  for (int i = n / 2 - 1; i >= 0; i--)   // 1) yığın kur
    yiginlastir(dizi, n, i);
  for (int i = n - 1; i > 0; i--) {      // 2) sırala
    takas(dizi, 0, i);                   // en büyüğü sona
    yiginlastir(dizi, i, 0);             // kökü süz
  }
}
void yiginlastir(int[] dizi, int n, int i) {
  int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
  if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
  if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
  if (enBuyuk != i) {
    takas(dizi, i, enBuyuk);
    yiginlastir(dizi, n, enBuyuk);
  }
}
```

- **Heap kurma**: yapraklar ($i \geq n/2$) zaten tek öğeli heap'tir; $i = n/2 - 1$'den köke doğru süzülür. Toplam **O(n)** (çoğu düğüm alçaktadır).
- **Sıralama**: $n - 1$ kez kök sona alınır ve O(log n)'de süzülür → **O(n log n)**.
- En kötü durumda bile O(n log n), O(1) ek bellek (yerinde).
- **Kararsız**: kök–son takası eşit anahtarların sırasını bozabilir.

Heap, öğeleri tek tek ekleyerek (yukarı süzme) de kurulabilir; bu da geçerli bir max-heap verir ama O(n log n) sürer. Buradaki aşağıdan yukarı kurma (Floyd yöntemi) O(n)'dir.

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
▶ for (int i = n / 2 - 1; i >= 0; i--)
▶   yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
5    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6    if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
7    if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
8    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



dizi = [3, 9, 2, 1, 4, 5]. Yapraklar (indis 3, 4, 5) zaten tek öğeli heap'tir. Kurma i = n / 2 − 1 = **2**'den köke doğru ilerler.

Adım 1 / 13

DEĞİŞKENLER

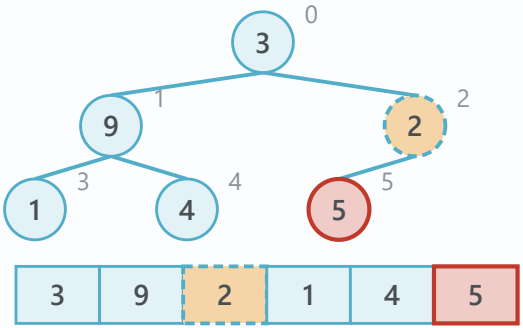
`n = 6` `i = 2` `sol = –` `sag = –` `enBuyuk = –`

ÇAĞRI YIĞINI

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
▶    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
▶    if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
▶    if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
8    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



i = 2 (2): sol = 5 (5) > 2 → enBuyuk = sol; sag yok.

Adım 2 / 13

DEĞİŞKENLER

`n = 6` `i = 2` `sol = 5` `sag = 6` `enBuyuk = 5`

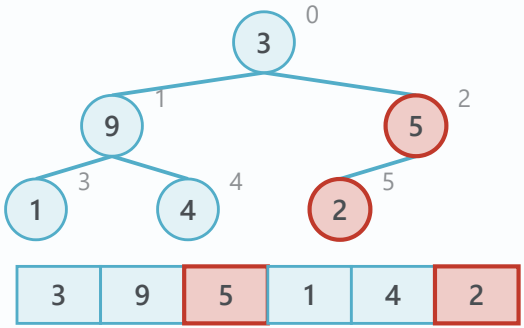
ÇAĞRI YIĞINI

`yiginlastir(2)`

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
5    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6    if (sol < n && dizi[sol] > dizi[enBuyuk])
       enBuyuk = sol;
7    if (sag < n && dizi[sag] > dizi[enBuyuk])
       enBuyuk = sag;
▶    if (enBuyuk != i) {
▶      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



enBuyuk ≠ i: `takas(dizi, 2, 5)` → 5 yukarı çıkar, 2 aşağı iner; süzme 5 numaralı düğümde sürer (satır 10).

Adım 3 / 13

DEĞİŞKENLER

`n = 6` `i = 2` `sol = 5` `sag = 6` `enBuyuk = 5`

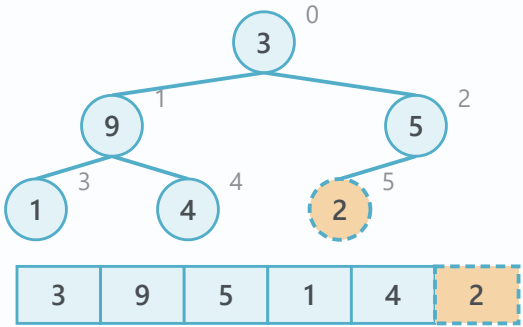
ÇAĞRI YIĞINI

`yiginlastir(2)`

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
     int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
     if (sol < n && dizi[sol] > dizi[enBuyuk])
   enBuyuk = sol;
     if (sag < n && dizi[sag] > dizi[enBuyuk])
   enBuyuk = sag;
     if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



i = 5 (2): çocuğu yok (sol = 11 ≥ n). enBuyuk = i, süzme biter.

Adım 4 / 13

DEĞİŞKENLER

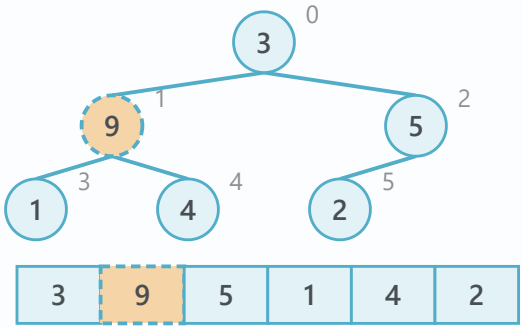
n = 6 | i = 5 | sol = 11 | sag = 12 | enBuyuk = 5

ÇAĞRI YIĞINI

- yiginlastir(5)
- yiginlastir(2)
- yiginSirala

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
▶ for (int i = n / 2 - 1; i >= 0; i--)
▶   yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
5    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6    if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
7    if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
8    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



Döngü i = 1: `yiginlastir(dizi, 6, 1)`.

Adım 5 / 13

DEĞİŞKENLER

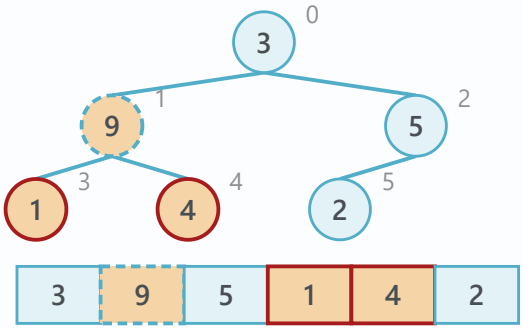
`n = 6` `i = 1` `sol = –` `sag = –` `enBuyuk = –`

ÇAĞRI YIĞINI

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
▶    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
▶    if (sol < n && dizi[sol] > dizi[enBuyuk])
       enBuyuk = sol;
▶    if (sag < n && dizi[sag] > dizi[enBuyuk])
       enBuyuk = sag;
▶    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



i = 1 (9): sol = 3 (1) > 9 değil; sag = 4 (4) > 9 değil. Değişiklik yok: zaten çocuklarından büyük.

Adım 6 / 13

DEĞİŞKENLER

`n = 6` `i = 1` `sol = 3` `sag = 4` `enBuyuk = 1`

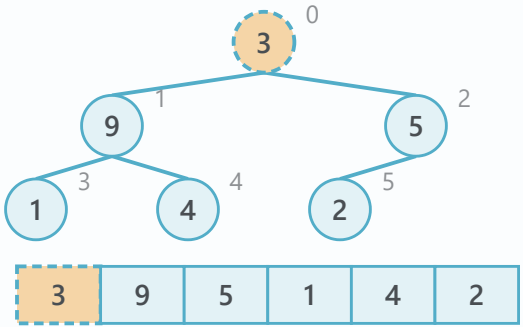
ÇAĞRI YIĞINI

`yiginlastir(1)`

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
▶ for (int i = n / 2 - 1; i >= 0; i--)
▶   yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
5    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6    if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
7    if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
8    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



Döngü i = 0: `yiginlastir(dizi, 6, 0)`.

Adım 7 / 13

DEĞİŞKENLER

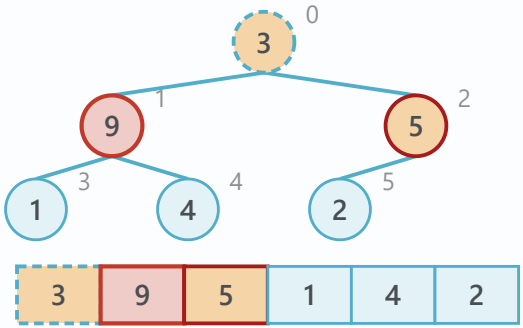
`n = 6` `i = 0` `sol = –` `sag = –` `enBuyuk = –`

ÇAĞRI YIĞINI

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
▶    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
▶    if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
▶    if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
8    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



i = 0 (3): sol = 1 (9) > 3 → enBuyuk = sol; sag = 2 (5) > 9 değil.

Adım 8 / 13

DEĞİŞKENLER

`n = 6` `i = 0` `sol = 1` `sag = 2` `enBuyuk = 1`

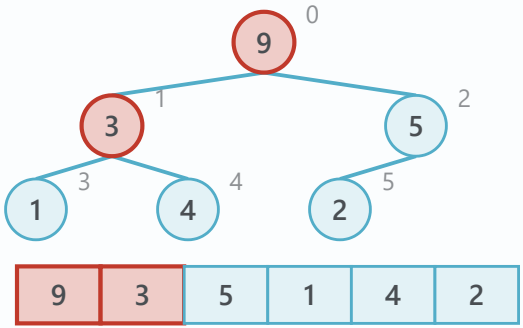
ÇAĞRI YIĞINI

`yiginlastir(0)`

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
5    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6    if (sol < n && dizi[sol] > dizi[enBuyuk])
       enBuyuk = sol;
7    if (sag < n && dizi[sag] > dizi[enBuyuk])
       enBuyuk = sag;
▶    if (enBuyuk != i) {
▶      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



enBuyuk ≠ i: `takas(dizi, 0, 1)` → 9 yukarı çıkar, 3 aşağı iner; süzme 1 numaralı düğümde sürer (satır 10).

Adım 9 / 13

DEĞİŞKENLER

`n = 6` `i = 0` `sol = 1` `sag = 2` `enBuyuk = 1`

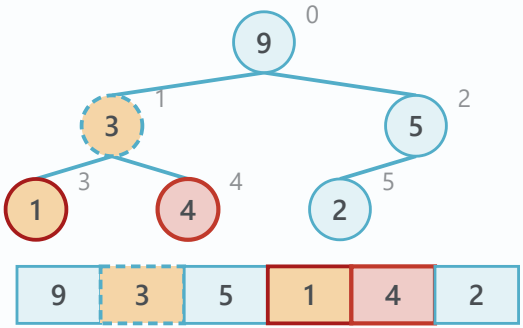
ÇAĞRI YIĞINI

`yiginlastir(0)`

`yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
▶    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
▶    if (sol < n && dizi[sol] > dizi[enBuyuk])
       enBuyuk = sol;
▶    if (sag < n && dizi[sag] > dizi[enBuyuk])
       enBuyuk = sag;
8    if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



i = 1 (3): sol = 3 (1) > 3 değil; sag = 4 (4) > 3 → enBuyuk = sag.

Adım 10 / 13

DEĞİŞKENLER

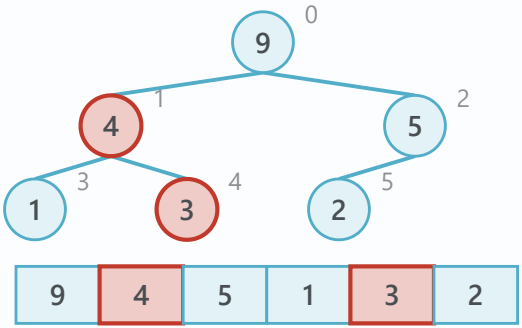
`n = 6` `i = 1` `sol = 3` `sag = 4` `enBuyuk = 4`

ÇAĞRI YIĞINI

- `yiginlastir(1)`
- `yiginlastir(0)`
- `yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
5    int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6    if (sol < n && dizi[sol] > dizi[enBuyuk])
       enBuyuk = sol;
7    if (sag < n && dizi[sag] > dizi[enBuyuk])
       enBuyuk = sag;
▶    if (enBuyuk != i) {
▶      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



enBuyuk ≠ i: `takas(dizi, 1, 4)` → 4 yukarı çıkar, 3 aşağı iner; süzme 4 numaralı düğümde sürer (satır 10).

Adım 11 / 13

DEĞİŞKENLER

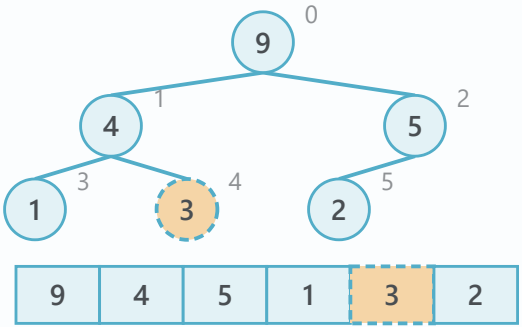
`n = 6` `i = 1` `sol = 3` `sag = 4` `enBuyuk = 4`

ÇAĞRI YIĞINI

- **`yiginlastir(1)`**
- `yiginlastir(0)`
- `yiginSirala`

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
1  for (int i = n / 2 - 1; i >= 0; i--)
2    yiginlastir(dizi, n, i);
3
4  void yiginlastir(int[] dizi, int n, int i) {
     int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
     if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
     if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
     if (enBuyuk != i) {
9      takas(dizi, i, enBuyuk);
10     yiginlastir(dizi, n, enBuyuk);
11   }
12 }
```



i = 4 (3): çocuğu yok (sol = 9 ≥ n). enBuyuk = i, süzme biter.

Adım 12 / 13

DEĞİŞKENLER

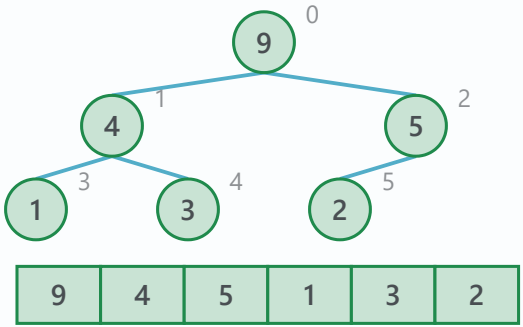
n = 6 | i = 4 | sol = 9 | sag = 10 | enBuyuk = 4

ÇAĞRI YIĞINI

- yiginlastir(4)
- yiginlastir(1)
- yiginlastir(0)
- yiginSirala

Max-Heap Kurma: Kodu İzle, dizi = [3, 9, 2, 1, 4, 5]

```
▶ for (int i = n / 2 - 1; i >= 0; i--)
2   yiginlastir(dizi, n, i);
3
4 void yiginlastir(int[] dizi, int n, int i) {
5   int enBuyuk = i, sol = 2 * i + 1, sag = 2 * i + 2;
6   if (sol < n && dizi[sol] > dizi[enBuyuk]) enBuyuk = sol;
7   if (sag < n && dizi[sag] > dizi[enBuyuk]) enBuyuk = sag;
8   if (enBuyuk != i) {
9     takas(dizi, i, enBuyuk);
10    yiginlastir(dizi, n, enBuyuk);
11  }
12 }
```



i = −1: döngü biter. **[9, 4, 5, 1, 3, 2]** bir max-heap: her ebeveyn çocuklarından büyük, en büyük öğe 9 kökte.

Adım 13 / 13

DEĞİŞKENLER

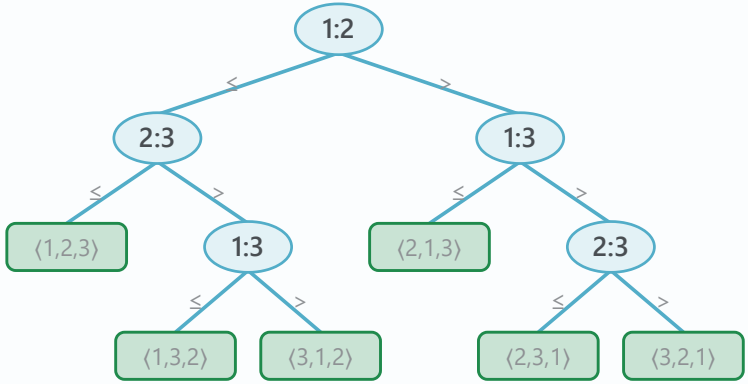
`n = 6` `i = -1` `sol = –` `sag = –` `enBuyuk = –`

ÇAĞRI YIĞINI

`yiginSirala`

Karşılaştırmalı Sıralamanın Alt Sınırı

- Yalnızca "$a_i \le a_j$?" sorusu soran her algoritma bir **karar ağacı** (decision tree) ile gösterilebilir.
- Her yaprak bir sıralamadır (permütasyon); doğru çalışmak için **n!** yaprak gerekir.
- Yüksekliği h olan ikili ağacın en fazla 2^h yaprağı vardır: $2^h \ge n! \Rightarrow h \ge \log_2(n!) =$ **$\Omega(n \log n)$**.
- Sonuç: birleştirme sıralaması ve heap sort **asimptotik olarak en iyidir**.
- Daha hızlı olmak için karşılaştırma dışında bilgi gerekir: anahtarlar küçük tam sayılar → **saymalı, basamak**.



3! = 6 yaprak → yükseklik ≥ ⌈log₂ 6⌉ = 3

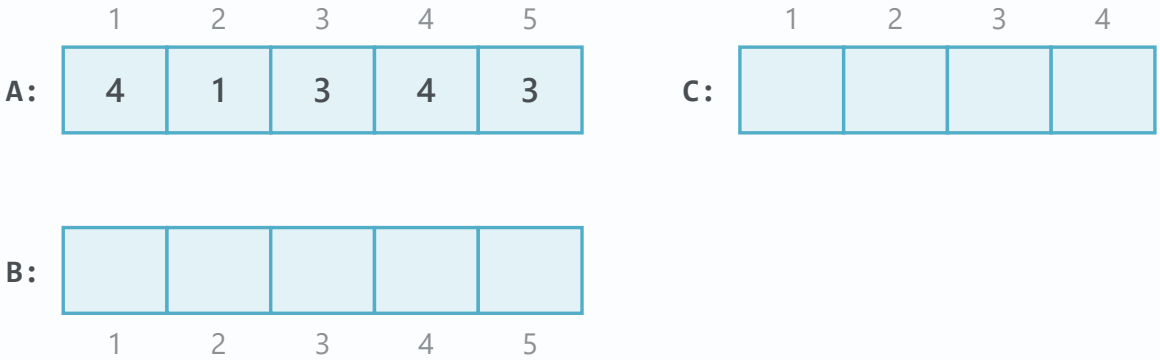
Saymalı Sıralama (Counting Sort)

- Öğeler birbiriyle **karşılaştırılmaz**.
- **Girdi**: A[1 … n], A[j] ∈ {1, 2, …, k}.
- **Çıktı**: B[1 … n], sıralı.
- **Geçici**: C[1 … k] sayaçları.

Fikir: anahtarı ≤ i olan öğe sayısı biliniyorsa, i anahtarlı son öğe tam o konuma yazılır.

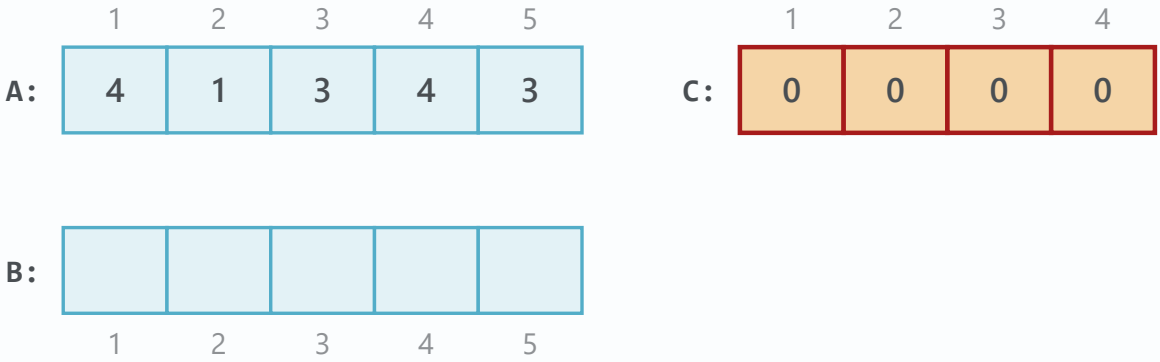
```
for i = 1 to k
  C[i] = 0
for j = 1 to n
  C[A[j]] = C[A[j]] + 1   // C[i] = |{anahtar = i}|
for i = 2 to k
  C[i] = C[i] + C[i-1]    // C[i] = |{anahtar ≤ i}|
for j = n downto 1
  B[C[A[j]]] = A[j]
  C[A[j]] = C[A[j]] - 1
```

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



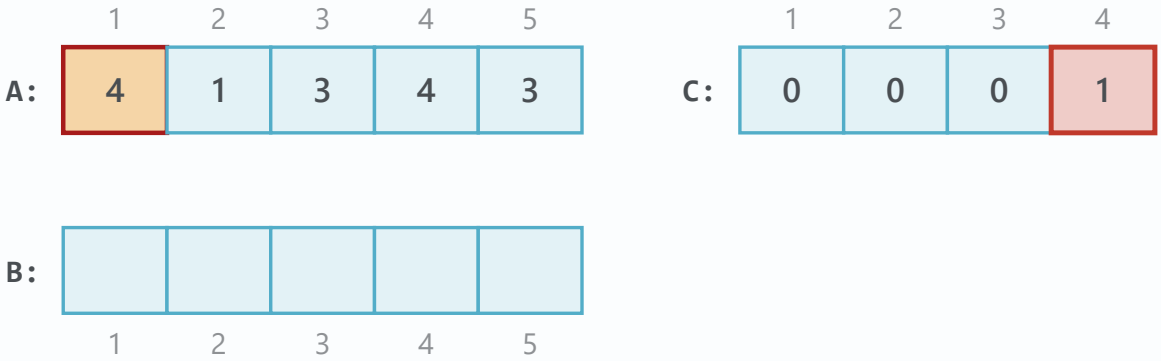
Girdi A = [4, 1, 3, 4, 3], n = 5, anahtarlar {1, …, 4}. C: sayaç dizisi (k = 4), B: çıktı.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



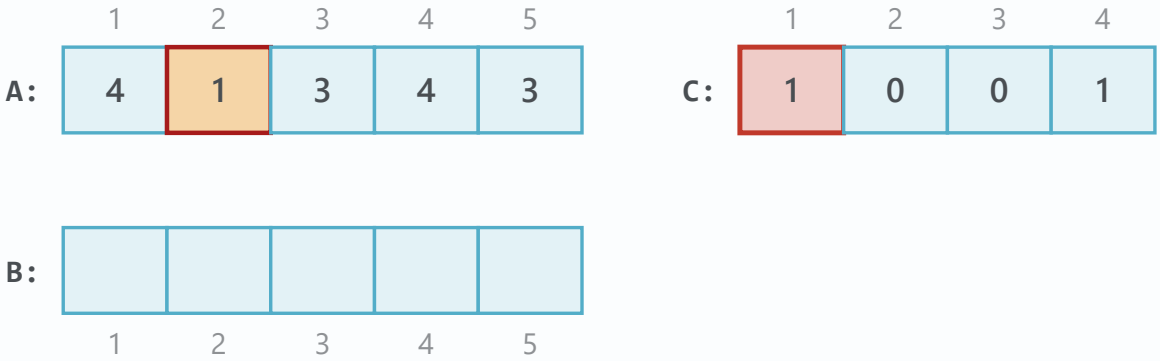
Döngü 1: `for i = 1 to k: C[i] = 0` — tüm sayaçlar sıfırlanır. Θ(k).

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



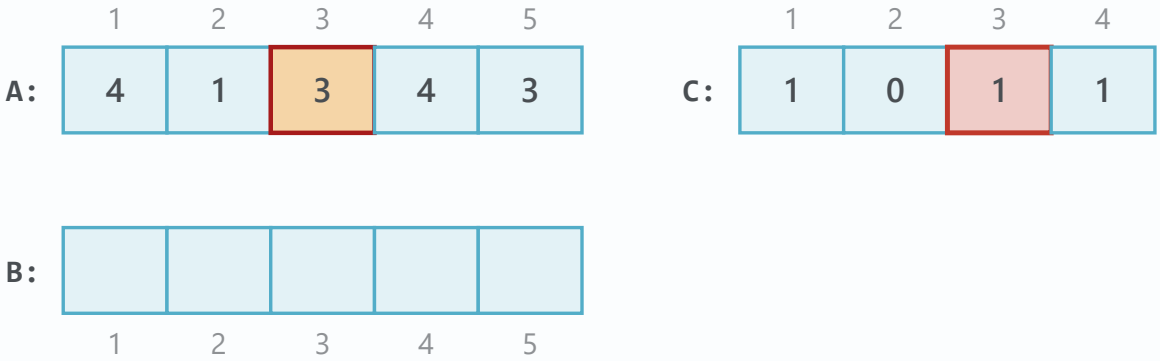
Döngü 2, j = 1: A[1] = 4 → C[4]++ = 1. (C[i] = i anahtarının adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



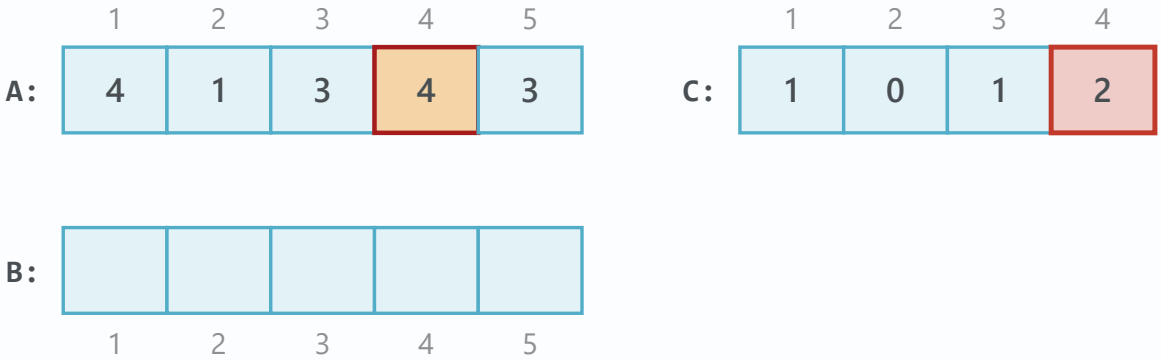
Döngü 2, j = 2: A[2] = 1 → C[1]++ = 1. (C[i] = i anahtarının adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



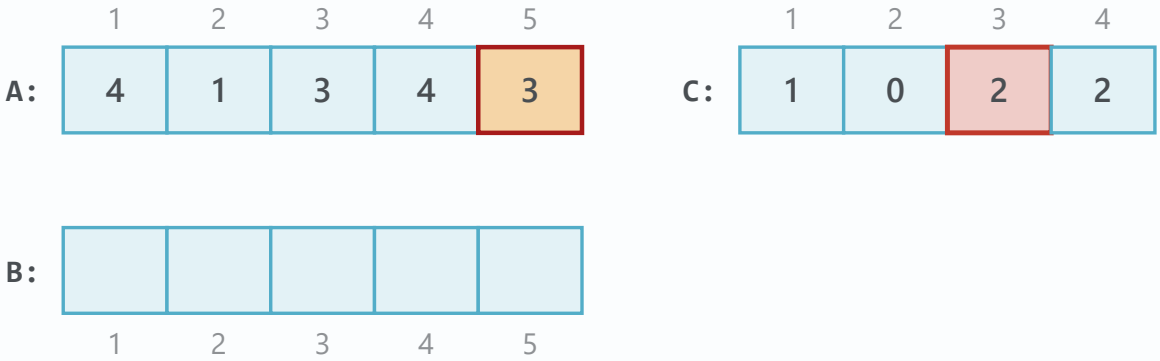
Döngü 2, j = 3: A[3] = 3 → C[3]++ = 1. (C[i] = i anahtarının adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



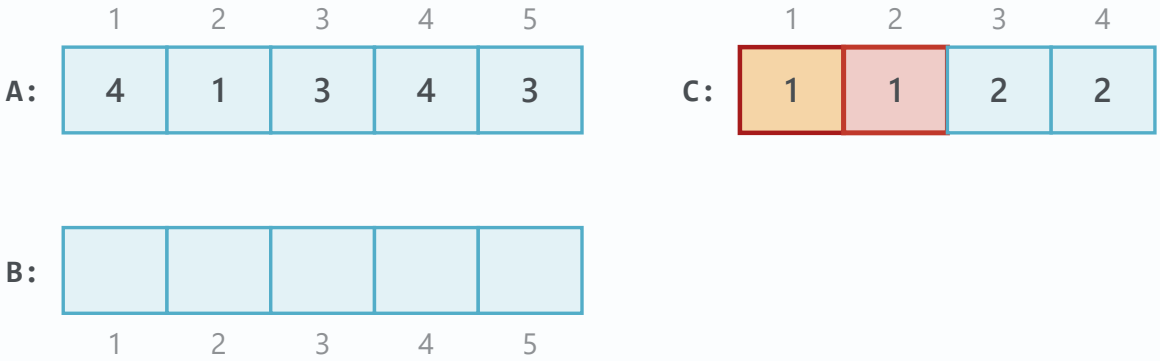
Döngü 2, j = 4: A[4] = 4 → C[4]++ = 2. (C[i] = i anahtarının adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



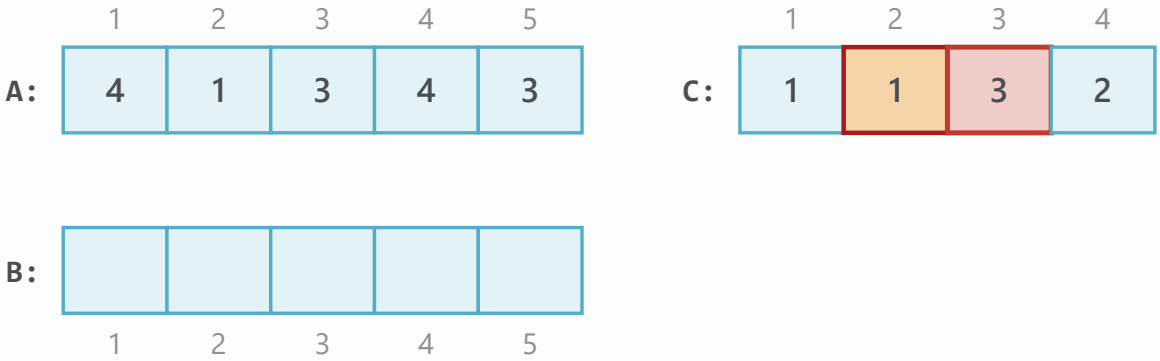
Döngü 2, j = 5: A[5] = 3 → C[3]++ = 2. (C[i] = i anahtarının adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



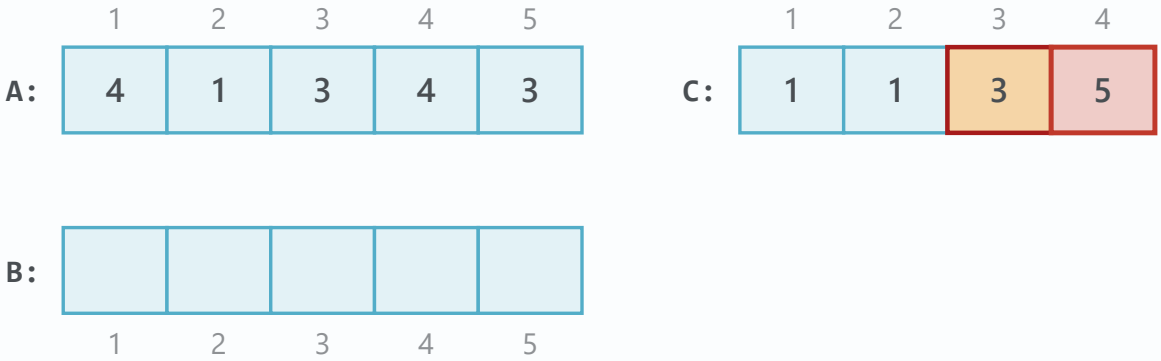
Döngü 3, i = 2: C[2] = C[2] + C[1] = 1. (C[i] = anahtarı ≤ i olanların adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



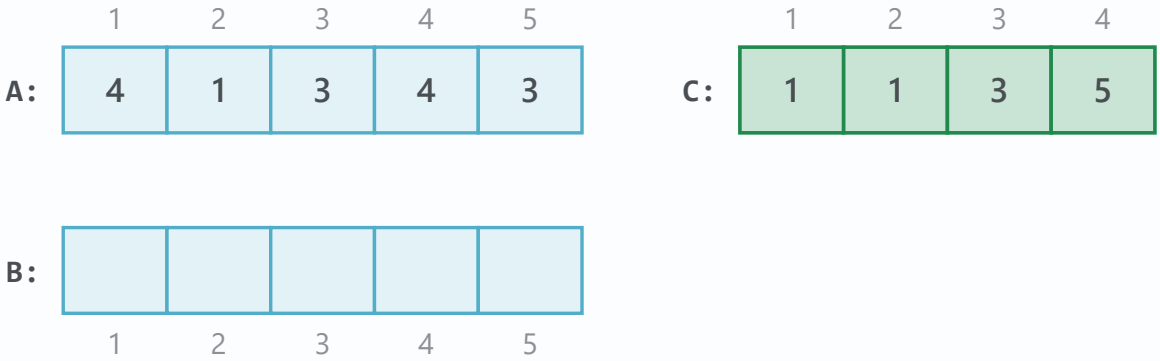
Döngü 3, i = 3: C[3] = C[3] + C[2] = 3. (C[i] = anahtarı ≤ i olanların adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



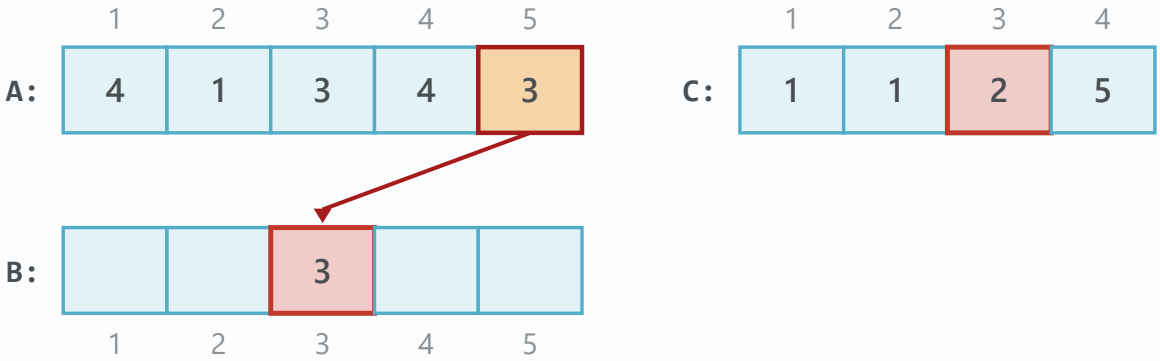
Döngü 3, i = 4: C[4] = C[4] + C[3] = 5. (C[i] = anahtarı ≤ i olanların adedi)

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



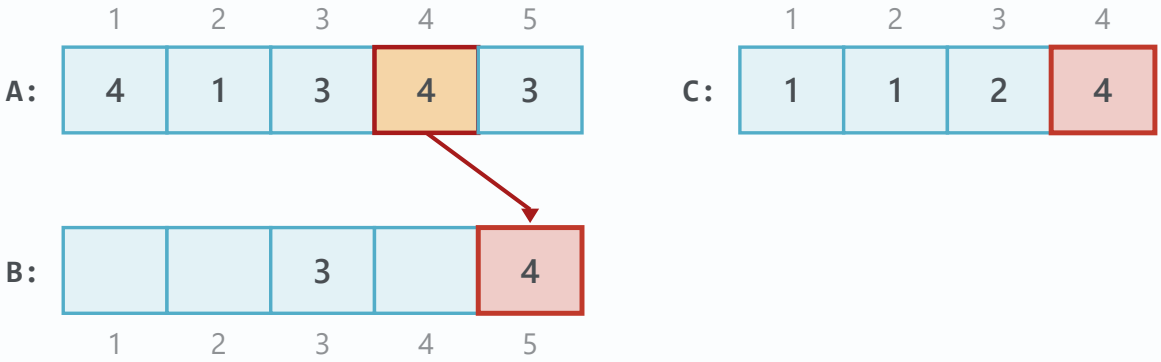
Önek toplamı hazır: C = [1, 1, 3, 5]. C[i], i anahtarlı son öğenin B'deki konumudur.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



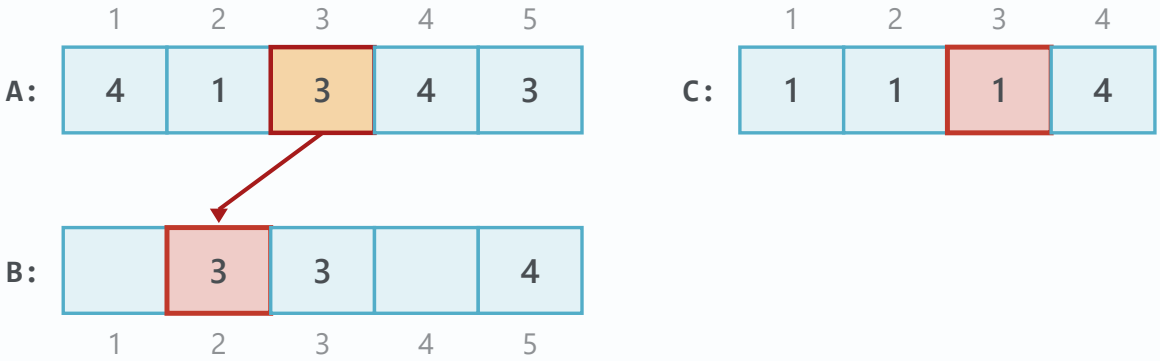
Döngü 4, j = 5: A[5] = 3 → B[C[3] = 3] = 3; C[3]−− = 2.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



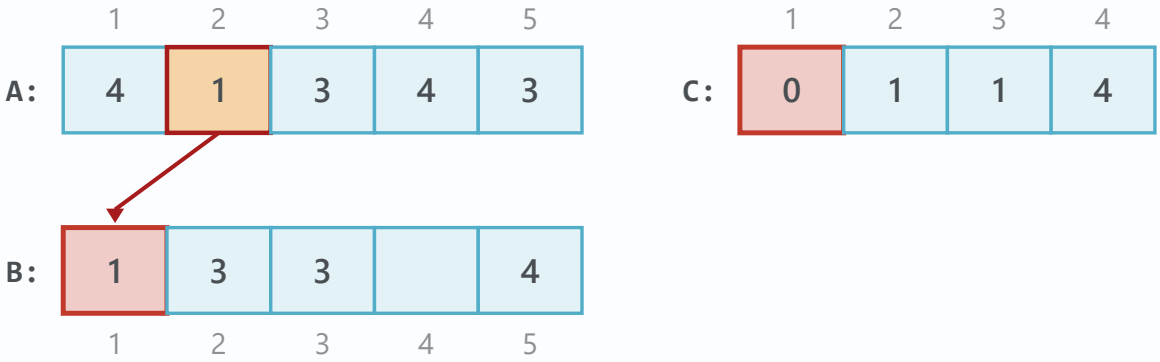
Döngü 4, j = 4: A[4] = 4 → B[C[4] = 5] = 4; C[4]−− = 4.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



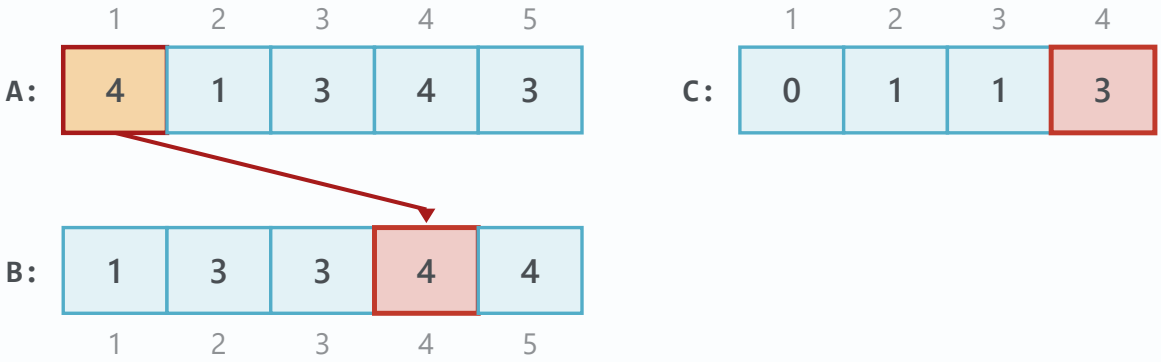
Döngü 4, j = 3: A[3] = 3 → B[C[3] = 2] = 3; C[3]−− = 1.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



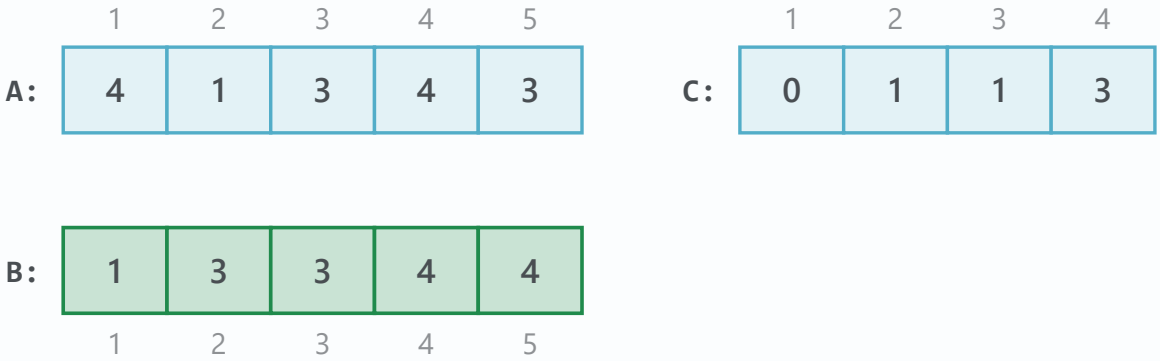
Döngü 4, j = 2: A[2] = 1 → B[C[1] = 1] = 1; C[1]−− = 0.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



Döngü 4, j = 1: A[1] = 4 → B[C[4] = 4] = 4; C[4]−− = 3.

Saymalı Sıralama Adım Adım: A = [4, 1, 3, 4, 3]



Sonuç B = [1, 3, 3, 4, 4]. Hiç karşılaştırma yapılmadı; sondan başa gezildiği için eşit anahtarların sırası korunur (**kararlı**).

Saymalı Sıralama: Karmaşıklık ve Java

Döngü	İş	Süre
1	C[1..k] = 0	Θ(k)
2	sayma	Θ(n)
3	önek toplamı	Θ(k)
4	B'ye yerleştir	Θ(n)
Toplam		**Θ(n + k)**

```java
int[] saymaliSirala(int[] dizi) {
  int maks = enBuyuk(dizi);
  int[] sayac = new int[maks + 1];
  for (int x : dizi) sayac[x]++;
  for (int i = 1; i <= maks; i++) sayac[i] += sayac[i - 1];
  int[] cikti = new int[dizi.length];
  for (int i = dizi.length - 1; i >= 0; i--) {
    cikti[sayac[dizi[i]] - 1] = dizi[i];
    sayac[dizi[i]]--;
  }
  return cikti;
}
```

- k = O(n) ise **Θ(n)**: Ω(n log n) sınırını aşar, çünkü karşılaştırma yapmaz.
- Ek bellek Θ(n + k); k çok büyükse (ör. 32 bitlik tam sayılar) pratik değildir. Java sürümü 0-tabanlıdır ve 0 … maks aralığını kullanır.
- Son döngünün **sondan başa** gezmesi kararlılığı sağlar → basamak sıralamasının temeli.

Basamak Sıralaması (Radix Sort)

- Tam sayıları **basamak basamak** sıralar; en anlamsız basamaktan (birler) başlar: **LSD**.
- Her geçişte o basamağa göre **kararlı** bir sıralama (saymalı, k = 10) yapılır.
- Kararlılık şarttır: onlar basamağına göre sıralarken birler sırası korunmalıdır.
- d basamak, taban b: **Θ(d · (n + b))**. d sabitse Θ(n).
- Ek bellek Θ(n + b); yerinde değildir.

```
int[] basamakSirala(int[] dizi) {
  int maks = enBuyuk(dizi);
  for (int us = 1; maks / us > 0; us *= 10)
    dizi = basamagaGoreSay(dizi, us);
  return dizi;
}
int[] basamagaGoreSay(int[] dizi, int us) {
  int n = dizi.length;
  int[] cikti = new int[n], sayac = new int[10];
  for (int x : dizi) sayac[(x / us) % 10]++;
  for (int i = 1; i < 10; i++) sayac[i] += sayac[i - 1];
  for (int i = n - 1; i >= 0; i--) {
    int b = (dizi[i] / us) % 10;
    cikti[--sayac[b]] = dizi[i];
  }
  return cikti;
}
```

Basamak Sıralaması Adım Adım

Girdi

170	45	75	90	802	24	2	66

Girdi [170, 45, 75, 90, 802, 24, 2, 66]. En büyük sayı 802: 3 basamak → 3 geçiş (en anlamsız basamaktan başlayarak, LSD).

Basamak Sıralaması Adım Adım

us = 1: birler basamağı

17**0**	4**5**	7**5**	9**0**	80**2**	2**4**	**2**	6**6**
0	5	5	0	2	4	2	6

Geçiş 1: her sayının birler basamağı (altta) anahtar olarak kullanılır; eksik basamak 0 sayılır.

Basamak Sıralaması Adım Adım

birler basamağına göre sıralı

17**0**	9**0**	80**2**	**2**	2**4**	4**5**	7**5**	6**6**
0	0	2	2	4	5	5	6

Kararlı saymalı sıralama sonrası: [170, 90, 802, 2, 24, 45, 75, 66]. Eşit basamaklılar önceki sıralarını korudu.

Basamak Sıralaması Adım Adım

us = 10: onlar basamağı

170	90	802	2	24	45	75	66
7	9	0	0	2	4	7	6

Geçiş 2: her sayının onlar basamağı (altta) anahtar olarak kullanılır; eksik basamak 0 sayılır.

Basamak Sıralaması Adım Adım

onlar basamağına göre sıralı

802	2	24	45	66	170	75	90
0	0	2	4	6	7	7	9

Kararlı saymalı sıralama sonrası: [802, 2, 24, 45, 66, 170, 75, 90]. Eşit basamaklılar önceki sıralarını korudu.

Basamak Sıralaması Adım Adım

us = 100: yüzler basamağı

802	2	24	45	66	**1**70	75	90
8	0	0	0	0	1	0	0

Geçiş 3: her sayının yüzler basamağı (altta) anahtar olarak kullanılır; eksik basamak 0 sayılır.

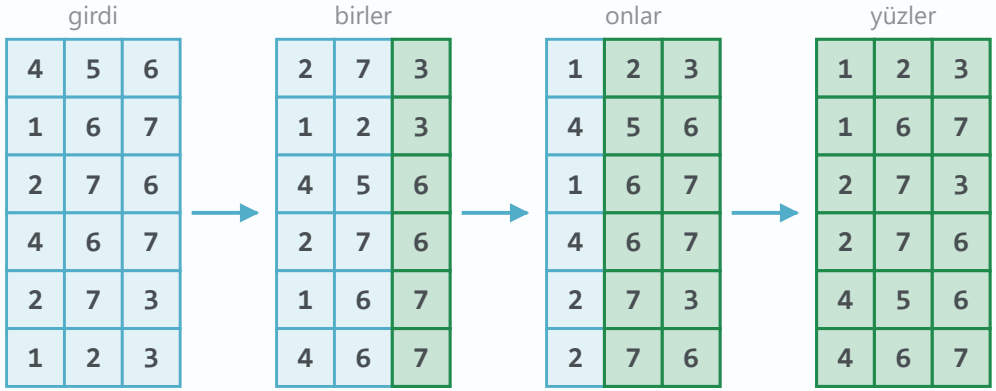
Basamak Sıralaması Adım Adım

yüzler basamağına göre sıralı

2	24	45	66	75	90	**1**70	**8**02
0	0	0	0	0	0	1	8

Kararlı saymalı sıralama sonrası: [2, 24, 45, 66, 75, 90, 170, 802]. Tüm basamaklar işlendi → **sıralı**.

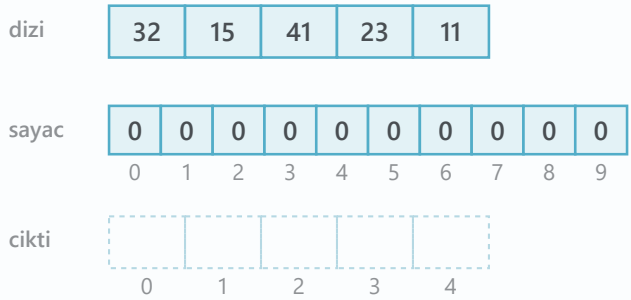
Basamak Sıralaması: Kararlılık Neden Gerekli?



- Yeşil sütunlar, o ana kadar **sıralı hâle gelen son ekler**dir: k geçişten sonra dizi, son k basamağa göre sıralıdır.
- 2. geçişte 167 ile 467'nin onlar basamağı eşittir (6); 1. geçişten gelen sıra (7 = 7, giriş sırası) korunur.
- Alt sıralama **kararsız** olsaydı, örneğin 273 ile 276 son geçişte yer değiştirebilir ve sonuç bozulurdu.

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
▶    int n = dizi.length;
▶    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



us = 1: **birler** basamağına göre sıralanacak. `sayac[0..9]` sıfır, `cikti` boş.

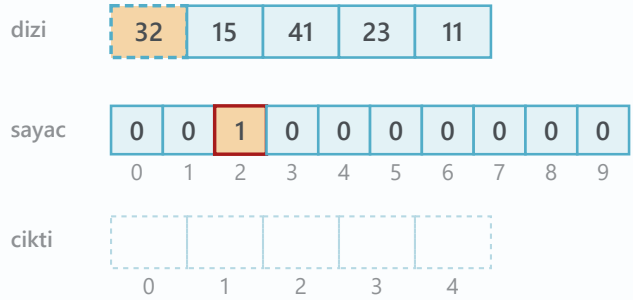
Adım 1 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **–** | b = **–**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
▶    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



x = 32: basamak (32 / 1) % 10 = **2** → `sayac[2]++` = 1.

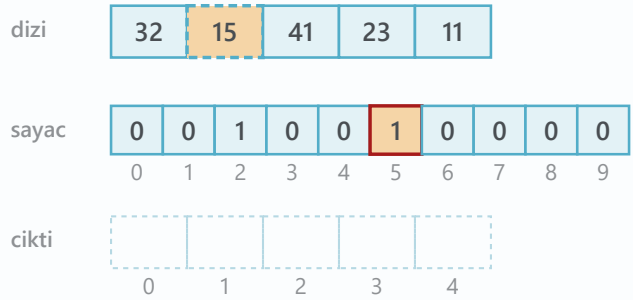
Adım 2 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **32** | b = **2**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
▶    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



x = 15: basamak (15 / 1) % 10 = **5** → `sayac[5]++` = 1.

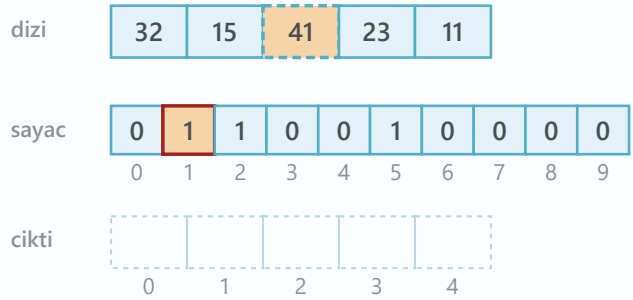
Adım 3 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **15** | b = **5**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
▶    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



x = 41: basamak (41 / 1) % 10 = **1** → `sayac[1]++` = 1.

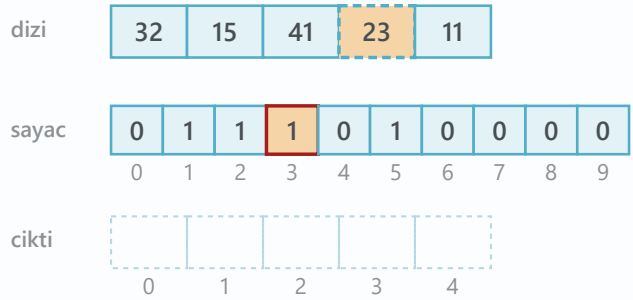
Adım 4 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **41** | b = **1**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
▶    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



x = 23: basamak (23 / 1) % 10 = **3** → `sayac[3]++` = 1.

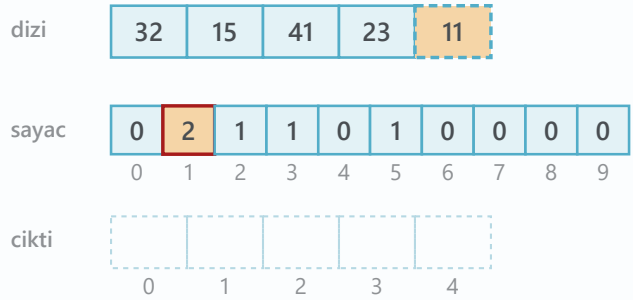
Adım 5 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **23** | b = **3**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
▶    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



x = 11: basamak (11 / 1) % 10 = **1** → `sayac[1]++` = 2.

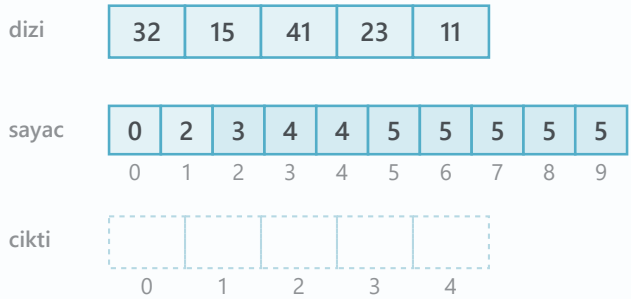
Adım 6 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **11** | b = **1**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
▶    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



Önek toplamı: `sayac[b]` artık basamağı ≤ b olan öğe sayısıdır; ör. sayac[3] = 4. Bu sayı, o basamaklı son öğenin `cikti` 'daki konumunun bir fazlasıdır.

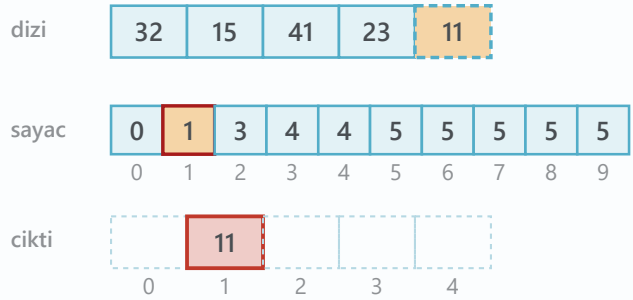
Adım 7 / 13

DEĞİŞKENLER

us = **1** | n = **5** | x = **–** | b = **–**

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
▶    for (int i = n - 1; i >= 0; i--) {
▶      int b = (dizi[i] / us) % 10;
▶      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



i = 4: x = 11, b = 1. `--sayac[1]` = 1 → `cikti[1] = 11` .

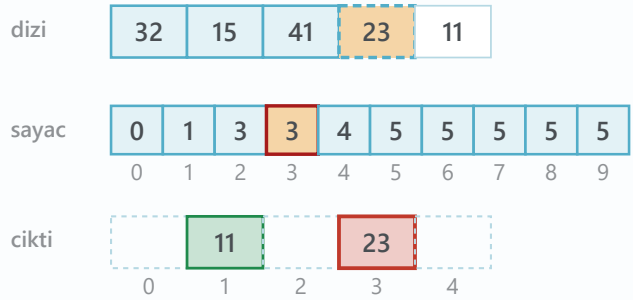
Adım 8 / 13

DEĞİŞKENLER

`us = 1` `n = 5` `i = 4` `x = 11` `b = 1`

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
▶    for (int i = n - 1; i >= 0; i--) {
▶      int b = (dizi[i] / us) % 10;
▶      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



i = 3: x = 23, b = 3. `--sayac[3]` = 3 → `cikti[3] = 23` .

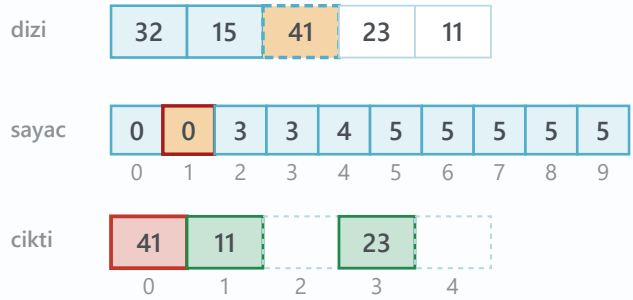
Adım 9 / 13

DEĞİŞKENLER

`us = 1` `n = 5` `i = 3` `x = 23` `b = 3`

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
▶    for (int i = n - 1; i >= 0; i--) {
▶      int b = (dizi[i] / us) % 10;
▶      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



i = 2: x = 41, b = 1. `--sayac[1]` = 0 → `cikti[0] = 41`. 41 ile 11'in basamağı aynı; sağdan sola gezildiği için 41 (girdide önce) 11'in soluna düştü: **kararlı**.

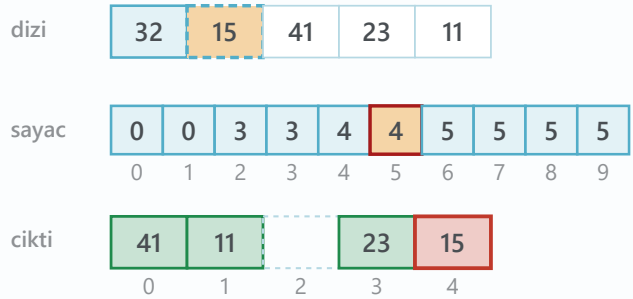
Adım 10 / 13

DEĞİŞKENLER

`us = 1` `n = 5` `i = 2` `x = 41` `b = 1`

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
▶    for (int i = n - 1; i >= 0; i--) {
▶      int b = (dizi[i] / us) % 10;
▶      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



i = 1: x = 15, b = 5. `--sayac[5]` = 4 → `cikti[4] = 15` .

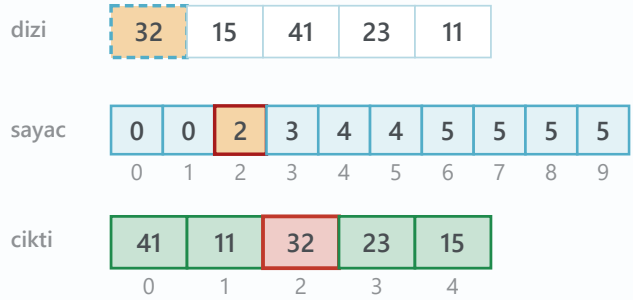
Adım 11 / 13

DEĞİŞKENLER

us = 1 | n = 5 | i = 1 | x = 15 | b = 5

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
▶    for (int i = n - 1; i >= 0; i--) {
▶      int b = (dizi[i] / us) % 10;
▶      cikti[--sayac[b]] = dizi[i];
9    }
10   return cikti;
11 }
```



i = 0: x = 32, b = 2. `--sayac[2]` = 2 → `cikti[2] = 32` .

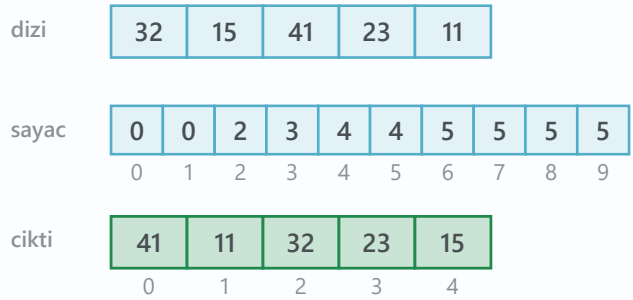
Adım 12 / 13

DEĞİŞKENLER

`us = 1` `n = 5` `i = 0` `x = 32` `b = 2`

basamagaGoreSay(dizi, 1): dizi = [32, 15, 41, 23, 11]

```
1  int[] basamagaGoreSay(int[] dizi, int us) {
2    int n = dizi.length;
3    int[] cikti = new int[n], sayac = new
   int[10];
4    for (int x : dizi) sayac[(x / us) % 10]++;
5    for (int i = 1; i < 10; i++) sayac[i] +=
   sayac[i - 1];
6    for (int i = n - 1; i >= 0; i--) {
7      int b = (dizi[i] / us) % 10;
8      cikti[--sayac[b]] = dizi[i];
9    }
▶    return cikti;
11 }
```



`cikti` = [41, 11, 32, 23, 15]: birler basamağına göre sıralı. Sonraki geçiş us = 10 ile onlar basamağına bakar; kararlılık sayesinde birler sırası korunur.

Adım 13 / 13

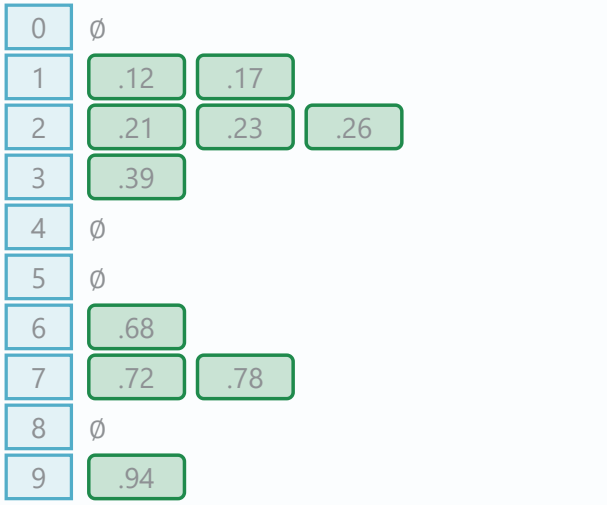
DEĞİŞKENLER

us = **1** | n = **5** | x = **–** | b = **–**

Kova Sıralaması (Bucket Sort)

- Girdi [0, 1) aralığına **düzgün dağılmış** sayılar olsun.
- n kova açılır; x değeri `⌊n · x⌋` numaralı kovaya atılır.
- Her kova (eklemeli sıralamayla) sıralanır, kovalar sırayla birleştirilir.
- Beklenen süre **Θ(n)**; tüm öğeler tek kovaya düşerse en kötü $\Theta(n^2)$.
- Simülatördeki onuncu algoritmadır; dağılım hakkında **varsayım** yapar.

A = .78, .17, .39, .26, .72, .94, .21, .12, .23, .68



Birleştir: .12, .17, .21, .23, .26, .39, .68, .72, .78, .94

TIMSORT

Timsort: Hibrit Sıralama

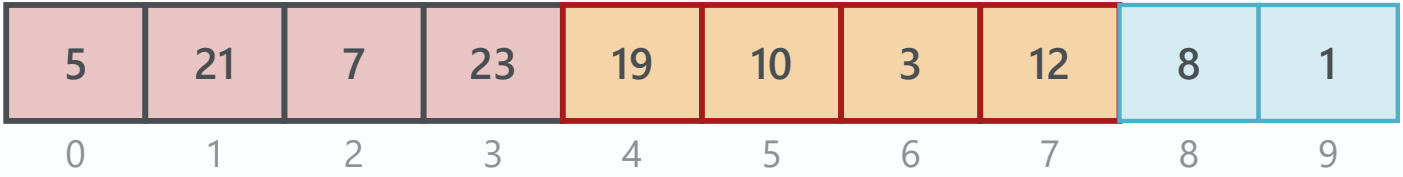
- Tim Peters (2002, Python). Java'da nesne dizileri (`Arrays.sort(Object[])` , `List.sort`) ve Python'da `sorted` Timsort kullanır.
- Gerçek veride sık görülen **sıralı parçaları** (run) kullanır: kısa run'lar **eklemeli** sıralamayla uzatılır, run'lar **birleştirme** sıralamayla birleştirilir.
- Kararlı; en kötü O(n log n), sıralı girdide **Θ(n)**; O(n) ek bellek.

```
static final int MIN_PARCA = 4;
void timSirala(int[] dizi) {
  int n = dizi.length;
  for (int i = 0; i < n; i += MIN_PARCA)            // 1) run'lar
    eklemeliSirala(dizi, i, Math.min(i + MIN_PARCA - 1, n - 1));
  for (int boyut = MIN_PARCA; boyut < n; boyut *= 2)  // 2) birleştir
    for (int sol = 0; sol < n; sol += 2 * boyut) {
      int orta = sol + boyut - 1;
      int sag = Math.min(sol + 2 * boyut - 1, n - 1);
      if (orta < sag) birlestir(dizi, sol, orta, sag);
    }
}
```

Buradaki sürüm sabit uzunluklu run'larla basitleştirilmiştir (gerçek Timsort'ta minrun 32–64, doğal run tespiti ve "galloping" vardır).

Timsort Adım Adım (MIN_PARCA = 4)

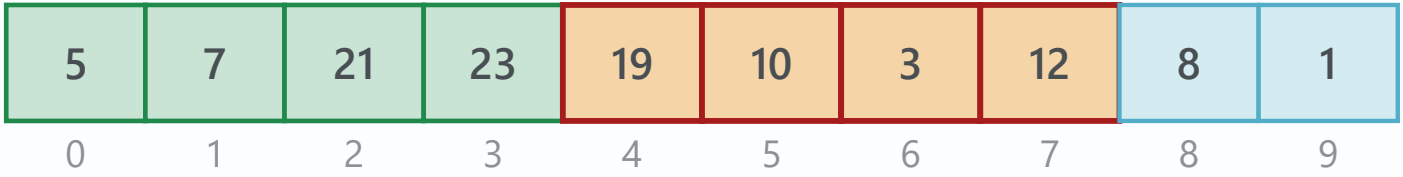
Girdi, MIN_PARCA = 4



Dizi [5, 21, 7, 23, 19, 10, 3, 12, 8, 1], n = 10. Dizi 4'erli parçalara (run) ayrılır (renkler).

Timsort Adım Adım (MIN_PARCA = 4)

eklemeliSirala(0, 3)



Run [0..3] eklemeli sıralama ile sıralandı: [5, 7, 21, 23].

Timsort Adım Adım (MIN_PARCA = 4)

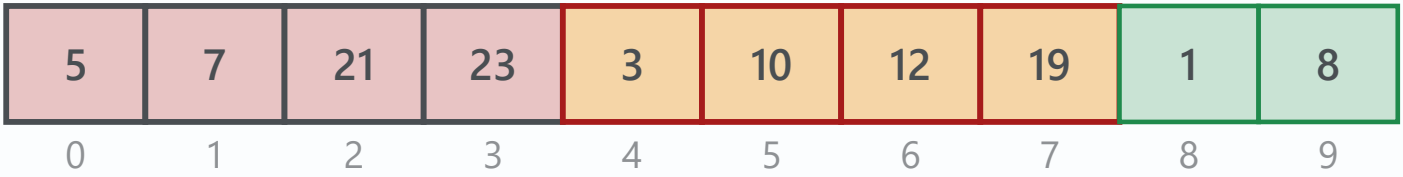
eklemeliSirala(4, 7)



Run [4..7] eklemeli sıralama ile sıralandı: [3, 10, 12, 19].

Timsort Adım Adım (MIN_PARCA = 4)

eklemeliSirala(8, 9)



Run [8..9] eklemeli sıralama ile sıralandı: [1, 8].

Timsort Adım Adım (MIN_PARCA = 4)

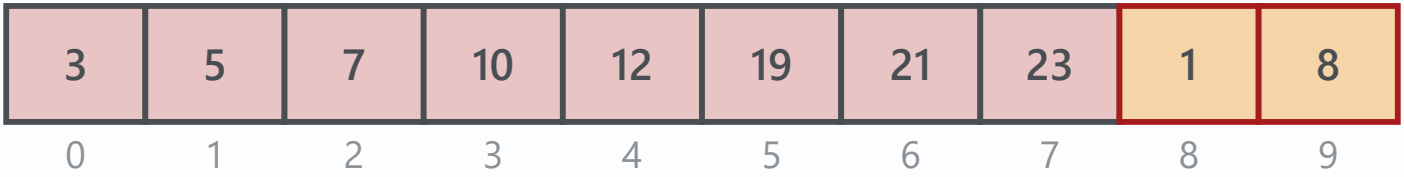
boyut = 4: birlestir(0, 3, 7)



boyut = 4: [5, 7, 21, 23] + [3, 10, 12, 19] birleştirildi → [3, 5, 7, 10, 12, 19, 21, 23].

Timsort Adım Adım (MIN_PARCA = 4)

boyut = 4: [8..9] tek başına



boyut = 4: [8..9] parçasının birleşeceği eşi yok (orta = 11 ≥ sag = 9) → aynen kalır.

Timsort Adım Adım (MIN_PARCA = 4)

boyut = 8: birlestir(0, 7, 9)



boyut = 8: [3, 5, 7, 10, 12, 19, 21, 23] + [1, 8] birleştirildi → [1, 3, 5, 7, 8, 10, 12, 19, 21, 23]. Dizi sıralandı.

KARŞILAŞTIRMA

Sıralama Algoritmalarının Karşılaştırılması

Algoritma	En iyi	Ortalama	En kötü	Ek bellek	Kararlı	Yerinde
Kabarcık (bayraklı)	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$	Evet	Evet
Eklemeli	$\Theta(n)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$	Evet	Evet
Seçimli	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	$O(1)$	Hayır	Evet
Shell (n/2 aralık)	$\Theta(n \log n)$	aralığa bağlı	$\Theta(n^2)$	$O(1)$	Hayır	Evet
Birleştirme	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n)$	Evet	Hayır
Hızlı	$\Theta(n \log n)$	$\Theta(n \log n)$	$\Theta(n^2)$	$O(\log n)$*	Hayır	Evet
Heap sort	$O(n \log n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$O(1)$	Hayır	Evet
Saymalı	$\Theta(n + k)$	$\Theta(n + k)$	$\Theta(n + k)$	$\Theta(n + k)$	Evet	Hayır
Basamak (LSD)	$\Theta(d(n + b))$	$\Theta(d(n + b))$	$\Theta(d(n + b))$	$\Theta(n + b)$	Evet	Hayır
Kova	$\Theta(n)$	$\Theta(n)$†	$\Theta(n^2)$	$\Theta(n)$	Evet	Hayır
Timsort	$\Theta(n)$	$\Theta(n \log n)$	$\Theta(n \log n)$	$O(n)$	Evet	Hayır

* özyineleme yığını (küçük parçaya önce inilirse). † düzgün dağılımda beklenen. k: anahtar aralığı, d: basamak sayısı, b: taban. Timsort'un $\Theta(n)$ en iyi durumu doğal run tespitiyle gerçek sürüm içindir.

KARŞILAŞTIRMA

Kaç Karşılaştırma? n = 1000

Algoritma	Rastgele	Sıralı	Ters sıralı
Kabarcık (bayraklı)	497.960	999	499.500
Eklemeli	252.205	999	499.500
Seçimli	499.500	499.500	499.500
Shell (n/2)	14.723	8.006	11.716
Birleştirme	8.693	5.044	4.932
Hızlı (son öğe pivot)	11.384	499.500	499.500
Heap sort	16.791	17.583	15.965

- Öğe karşılaştırma sayıları; bu bölümdeki kodlar n = 1000 için çalıştırılarak sayıldı (rastgele dizi sabit tohumla üretildi).
- Karşılaştırma için: $n^2 / 2 \approx 500.000$, $n \log_2 n \approx 9.966$. Sıralı girdide son öğe pivotlu hızlı sıralama en kötü durumuna düşer.

KARŞILAŞTIRMA

Hangi Durumda Hangi Algoritma?

Küçük ya da neredeyse sıralı

Eklemeli sıralama: az ek yük, uyarlanabilir, kararlı.

Genel amaçlı, hız

Hızlı sıralama (rastgele / üçün ortancası pivot); önbellek dostu, yerinde.

Kararlılık gerekli

Birleştirme sıralaması ya da Timsort (kütüphanelerin nesne sıralaması).

En kötü durum garantisi, az bellek

Heap sort: O(n log n) ve O(1) ek bellek (introsort'un yedeği).

Küçük tam sayı anahtarlar

Saymalı ya da basamak sıralaması: Θ(n + k), Θ(d(n + b)).

Bağlı liste, dış bellek

Birleştirme sıralaması: rastgele erişim gerektirmez; disk üzerindeki büyük dosyalar parça parça birleştirilir.

Kabarcık ve seçimli sıralama öğretim amaçlıdır; seçimli yalnızca yazmanın çok pahalı olduğu durumlarda anlamlıdır.

Özet

Konu	Hatırlanacaklar
Ölçütler	zaman (en iyi/ortalama/en kötü), ek bellek, kararlılık, yerinde, uyarlanabilirlik
Kabarcık	komşu takas; tur sonunda en büyük sonda; bayrakla en iyi $\Theta(n)$, aksi $\Theta(n^2)$; kararlı
Eklemeli	anahtarı sakla, büyükleri kaydır, yerleştir; $\Theta(n + \text{ters çift})$; küçük/neredeyse sıralı veride en iyi
Seçimli	en küçüğü bul, başa al; her zaman $n(n-1)/2$ karşılaştırma, $\leq n-1$ takas; kararsız
Shell	azalan aralıklarla eklemeli sıralama; yerinde, kararsız
Birleştirme	böl–sırala–birleştir; $T(n) = 2T(n/2) + \Theta(n) = \Theta(n \log n)$; $\Theta(n)$ ek bellek; kararlı
Hızlı	pivot etrafında bölümle; ortalama $\Theta(n \log n)$, en kötü $\Theta(n^2)$; yerinde, kararsız
Heap sort	$O(n)$ max-heap kur, $n-1$ kez kökü sona al ve süz; $O(n \log n)$, $O(1)$ ek bellek
Doğrusal	karşılaştırmalı alt sınır $\Omega(n \log n)$; saymalı $\Theta(n + k)$, basamak $\Theta(d(n + b))$, kova beklenen $\Theta(n)$

ÖZET

Etkileşimli Simülatörler

Sıralama Algoritmaları

On algoritma (kabarcık, eklemeli, seçimli, birleştirme, hızlı, saymalı, basamak, kova, heap, Shell) çubuk grafik üzerinde adım adım; karşılaştırılan ve sıralanan öğeler, işlem günlüğü ve karmaşıklık tablosu.

Sıralama Yarışı

Kabarcık, eklemeli, seçimli ve hızlı sıralama aynı rastgele dizi (8–40 öğe) üzerinde yan yana yarışır; karşılaştırma ve takas sayıları karşılaştırılır.

Sıralama Özellikleri Laboratuvarı

Eşit anahtarlı kayıtlar (3, A), (1, B), (3, C) … eklemeli ya da seçimli sıralamayla sıralanır; göreli sıranın korunup korunmadığı (kararlılık) görülür.

ÖZET

Sık Yapılan Hatalar

Dizi sınırını aşmak

Kabarcıkta iç döngü `j < n - 1 - i` olmalı; `j < n` yazmak `dizi[j+1]` ile taşar.

Anahtarı saklamamak

Eklemeli sıralamada `gecici` önceden alınmazsa ilk kaydırma onu ezer.

En küçüğü değer olarak tutmak

Seçimli sıralamada indis tutulmazsa takas yapılamaz.

Kararlılığı bozmak

Birleştirmede `<=` yerine `<`, saymalıda baştan sona gezmek eşit anahtarların sırasını bozar.

Pivotu özyinelemeye katmak

`hizliSirala(dizi, alt, p)` yazmak pivotu tekrar işler; iki öğe kalınca sonsuz özyineleme olabilir.

"Quicksort her zaman n log n"

Sıralı girdide sabit pivot $\Theta(n^2)$ ve n derinlikli özyineleme (StackOverflowError) verir.

ÖZET

Örnek Problemler

1. [5, 2, 4, 6, 1, 3] dizisine bayraklı kabarcık sıralaması uygulanıyor. Her tur sonundaki diziyi yazın.
2. Aynı diziyi eklemeli sıralama kaç karşılaştırma ve kaç kaydırma ile sıralar? Seçimli sıralama kaç takas yapar?
3. `parcala` (Lomuto, pivot = son öğe) [2, 8, 7, 1, 3, 5, 6, 4] üzerinde çalışınca dizi ve dönüş değeri ne olur?
4. [4, 10, 3, 5, 1] dizisinden aşağıdan yukarı kurulan max-heap nedir?
5. A = [2, 5, 3, 0, 2, 3, 0, 3] (anahtarlar 0 … 5) için saymalı sıralamanın 3. döngüden sonraki C dizisi nedir?
6. Birleştirme sıralaması 8 öğeyi en fazla kaç karşılaştırmayla sıralar?

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. n = 6 öğeli ters sıralı dizide kabarcık sıralaması kaç karşılaştırma ve kaç takas yapar?
2. Seçimli sıralamanın kararsız olduğunu gösteren üç öğeli bir örnek verin.
3. Eklemeli sıralamada doğru konumu **ikili arama** ile bulursak zaman karmaşıklığı ne olur?
4. Hızlı sıralamayı (son öğe pivot) zaten sıralı 1000 öğeli diziyle çağırınca ne olur? Nasıl düzeltilir?
5. [329, 457, 657, 839, 436, 720, 355] dizisini LSD basamak sıralaması ile sıralayın; her geçişten sonraki diziyi yazın.
6. Bir dizideki **ters çift** sayısını O(n log n)'de bulan bir yöntem önerin.

▶ **Cevaplar / ipuçları**

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, MIT Press, 2022 (Bölüm 2, 6, 7, 8).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, Addison-Wesley, 2011 (Bölüm 2: Sorting).
- D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, 2. baskı, 1998.
- E. D. Demaine ve C. E. Leiserson, MIT 6.046J *Introduction to Algorithms* ders notları, 2005 (bölümleme ve saymalı sıralama örnekleri).
- C. A. R. Hoare, "Quicksort", *The Computer Journal* 5(1), 1962; D. L. Shell, "A High-Speed Sorting Procedure", *CACM* 2(7), 1959.
- T. Peters, "listsort.txt" (Timsort tasarım notu), CPython, 2002.

Son

Bölüm 2: Sıralama Algoritmaları

Sonraki bölüm: Bölüm 3: Arama Algoritmaları · Terim sözlüğü