SIRALAMA ALGORİTMALARI

Sıralama Algoritmaları Genel Karşılaştırma

Adı	Ortalama	En Kötü	Bellek	Kararlı mı?	Yöntem
Kabarcık Sıralaması	—	$O(n^2)$	O(1)	Evet	Değiştirme
Kokteyl Sıralaması	—	$O(n^2)$	O(1)	Evet	Değiştirme
Tarak Sıralaması	$O(n \log n)$	$O(n \log n)$	O(1)	Hayır	Değiştirme
Cüce Sıralaması	—	$O(n^2)$	O(1)	Evet	Değiştirme
Seçmeli Sıralama	$O(n^2)$	$O(n^2)$	O(1)	Hayır	Seçme
Eklemeli Sıralama	$O(n + d)$	$O(n^2)$	O(1)	Evet	Ekleme
Kabuk Sıralaması	—	$O(n \log^2 n)$	O(1)	Hayır	Ekleme
Ağaç Sıralaması	$O(n \log n)$	$O(n \log n)$	$O(n)$	Evet	Ekleme
Kütüphane Sıralaması	$O(n \log n)$	$O(n^2)$	$O(n)$	Evet	Ekleme
Birleştirmeli Sıralama	$O(n \log n)$	$O(n \log n)$	$O(n)$	Evet	Birleştirme
Yerinde Birleştirmeli Sıralama	$O(n \log n)$	$O(n \log n)$	O(1)	Evet	Birleştirme
Yığın Sıralaması	$O(n \log n)$	$O(n \log n)$	O(1)	Hayır	Seçme
Rahat Sıralama	—	$O(n \log n)$	O(1)	Hayır	Seçme
Hızlı Sıralama	$O(n \log n)$	$O(n^2)$	$O(\log n)$	Hayır	Bölümlendirme
İçgözlemle Sıralama	$O(n \log n)$	$O(n \log n)$	$O(\log n)$	Hayır	Melez
Sabır Sıralaması	—	$O(n^2)$	$O(n)$	Hayır	Ekleme
İplik Sıralaması	$O(n \log n)$	$O(n^2)$	$O(n)$	Evet	Seçme

1. Ne kadar hızlı sıralayabiliriz?

- ***n* adet sayı *O(n lgn)*** zamanda sıralayan çeşitli algoritmalar vardır.
  - **Birleştirmeli** ve **bellek yığın** (heap) sıralama bu üst sınıra **en kötü durumda ulaşır**.
  - **Hızlı (quick) sıralama** ise **ortalama zamanda** ulaşır.
- Bu algoritmaların her biri için $\Omega(n \lg n)$ sürede çalışmasına sebep olan ***n* girdi sayıda bir sıra üretebiliriz.**
- Bu algoritmalarda ***belirledikleri sıralı düzen*** **sadece girdi elemanları arasındaki karşılaştırmaya dayanır.**
- Bu algoritmalar **karşılaştırmalı sıralama algoritmalardır.**

2. Sıralama İçin Alt Sınırlar

❖***n* sayıda elemanı karşılaştırarak sıralamak** için **en kötü durumda** $\boldsymbol{\Omega(n \lg n)}$ karşılaştırma yapılmasını kanıtlamalıyız.

- Karşılaştırma sıralamasında $\langle a_1, a_2, \ldots, a_n \rangle$ girdi dizisi hakkında düzenli bir bilgi edinmek için karşılaştırmalar kullanılır.
- Verilen a_i ve a_j elemanları göreceli bir düzene karar vermek için $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$ veya $a_i > a_j$ testlerinden birinden geçirilir.

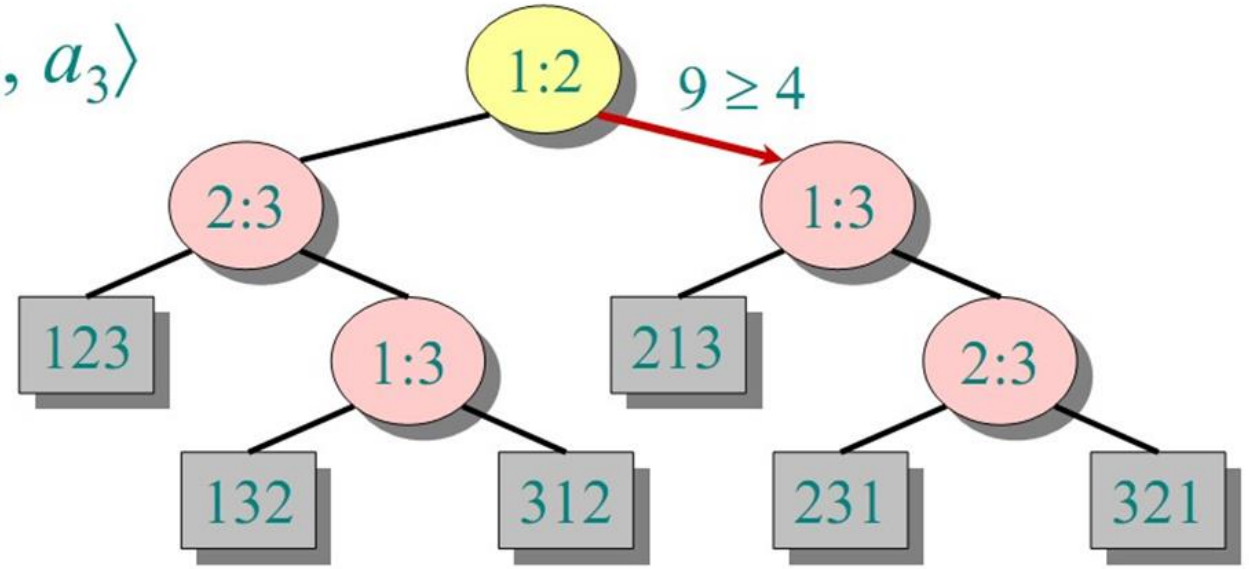
  Burada = olma veya > , < olma durumları yerine $a_i \leq a_j$, $a_i \geq a_j$ değerlendirmemiz yeterli olacaktır.

- **n** tane sayı **n!** düzende karşımıza gelebilir.

Alt Sınırı Nasıl Belirleriz?

❖Alt sınır belirlememizde **karar ağaçlarından faydalanabiliriz.**

Sırala $\langle a_1, a_2, a_3 \rangle$ $= \langle 9, 4, 6 \rangle$:

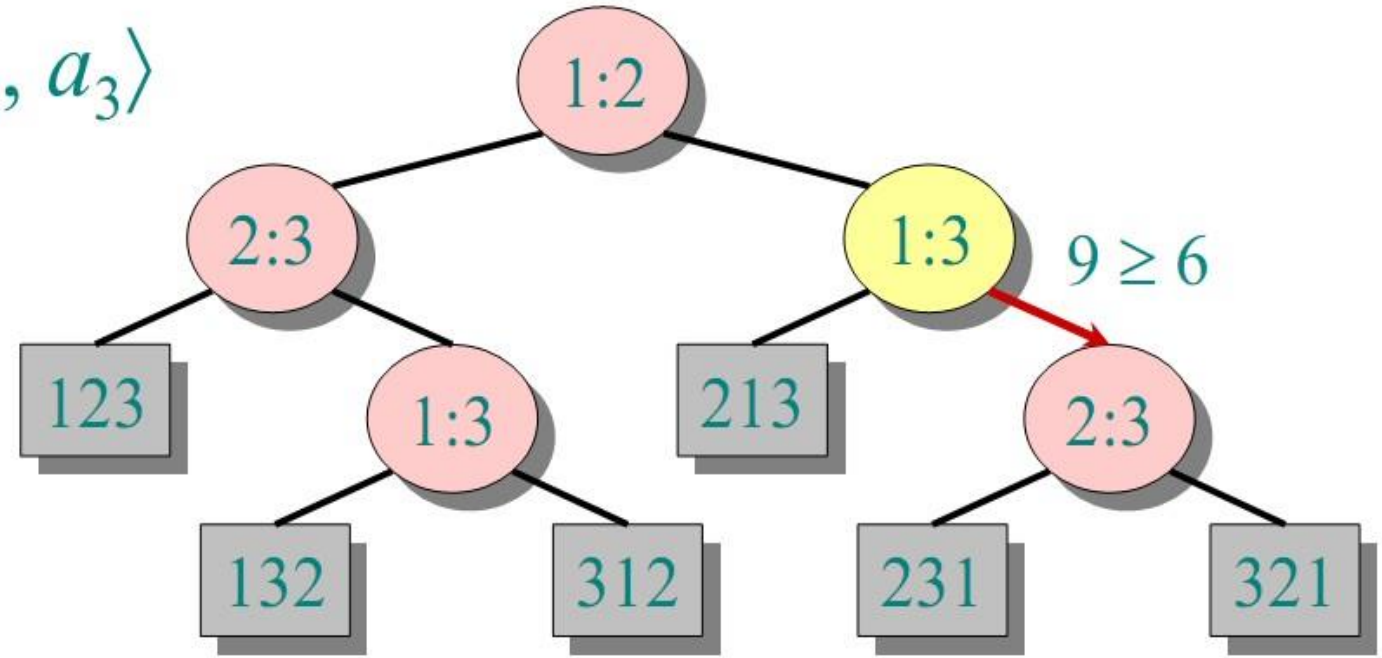


* Araya yerleştirme algoritmasına uyan bir karar ağacı örneği

Her iç boğumun etiketlenmesi $i{:}j$; $i, j \in \{1, 2, \ldots, n\}$ için.

- Sol alt-ağaç $a_i \leq a_j$ ise, ardarda karşılaştırmaları gösterir.
- Sağ alt-ağaç $a_i \geq a_j$ ise, ardarda karşılaştırmaları gösterir.

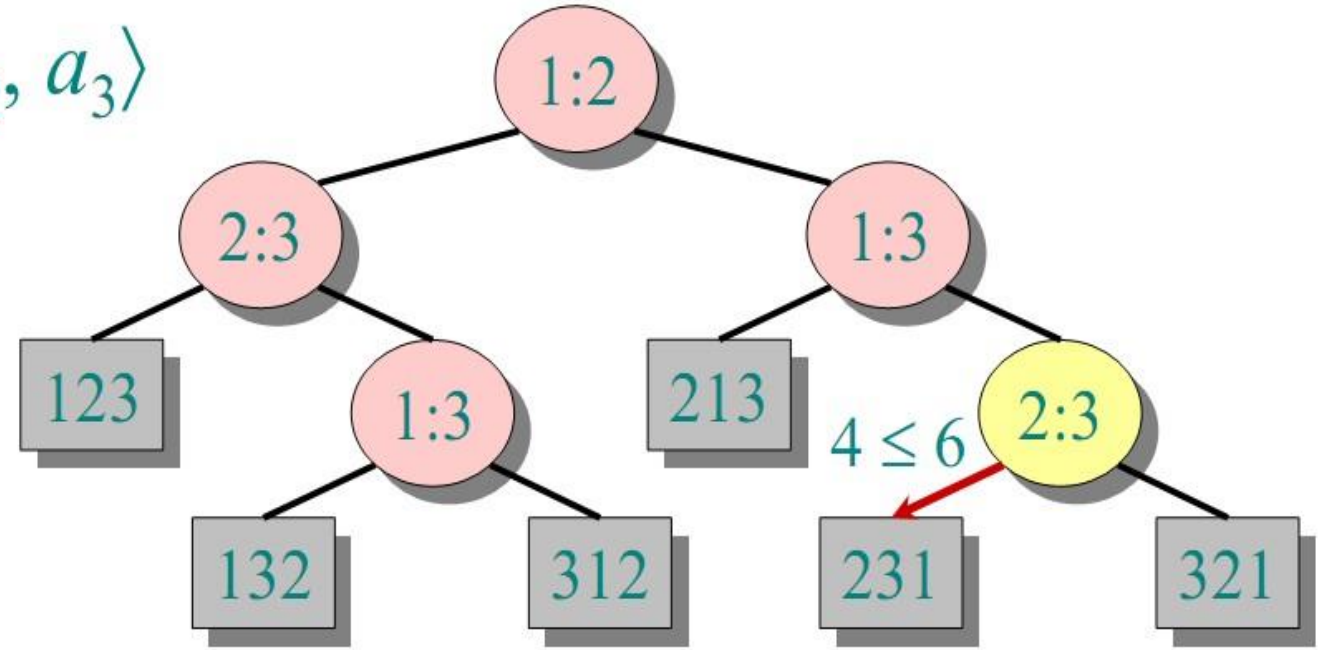
Sırala $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



Her iç boğumun etiketlenmesi $i{:}j$; $i, j \in \{1, 2, \ldots, n\}$ için:

- Sol alt-ağaç $a_i \leq a_j$ ise, ardarda karşılaştırmaları gösterir.
- Sağ alt-ağaç $a_i \geq a_j$ ise, ardarda karşılaştırmaları gösterir.

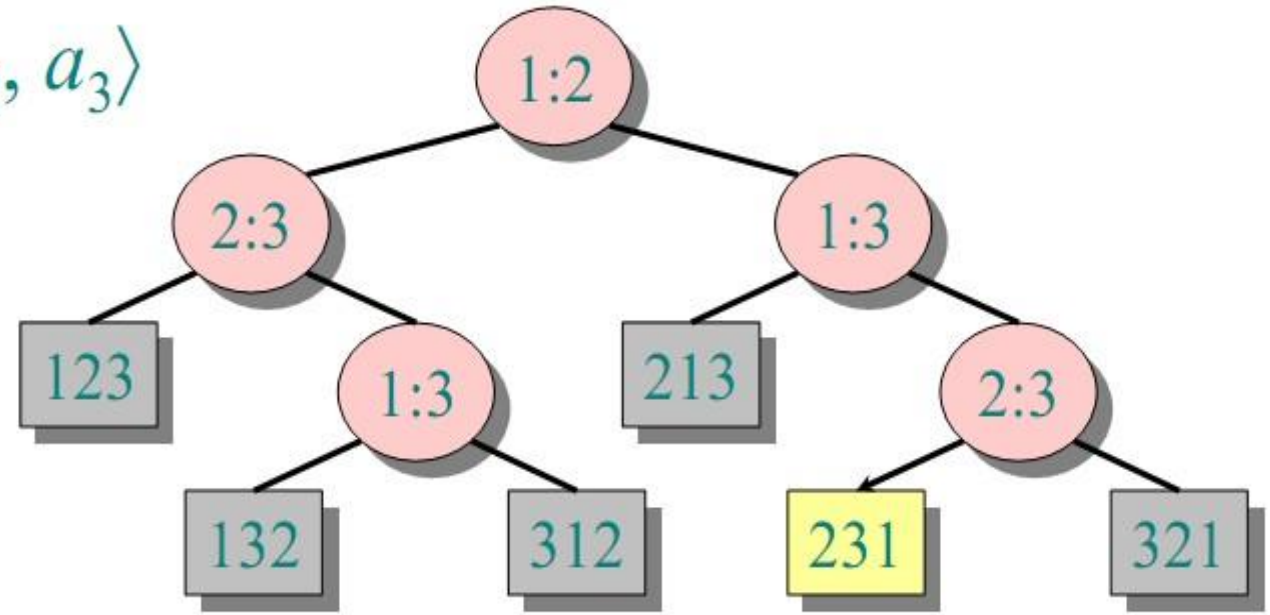
Sırala $\langle a_1, a_2, a_3 \rangle$
$= \langle 9, 4, 6 \rangle$:



Her iç boğumun etiketlenmesi $i{:}j$; $i, j \in \{1, 2, \ldots, n\}$ için.

- Sol alt-ağaç $a_i \leq a_j$ ise, ardarda karşılaştırmaları gösterir.
- Sağ alt-ağaç $a_i \geq a_j$ ise, ardarda karşılaştırmaları gösterir.

Sırala $\langle a_1, a_2, a_3 \rangle$ $= \langle\ 9, 4, 6\ \rangle$:



Her yaprakta $\langle \pi(1), \pi(2), \ldots, \pi(n) \rangle$ permütasyonu vardır bu $a_{\pi(1)} \leq a_{\pi(2)} \leq \cdots \leq a_{\pi(n)}$ sıralamasının tamamlanmış olduğunu gösterir.

Karar-Ağacı Modeli

- *Bir karar ağacı her karşılaştırma sıralaması uygulanmasını modelleyebilir:*
  - *Her **n** giriş boyutu için bir ağaç.*
  - *Algoritmayı **iki elemanı karşılaştırdığında** bölünüyormuş gibi görün.*
  - *Ağaç tüm olası **komut izlerindeki karşılaştırmalar içerir.***
  - ***Algoritmanın çalışma zamanı = Takip edilen yolun uzunluğu.***
  - ***En kötü-durum çalışma zamanı = Ağacın boyu.***

Karar-ağacı sıralamasında alt sınır

Teorem. n elemanı sıralayabilen bir karar-ağacının yüksekliği (boyu) $\Omega(n \lg n)$ olmalıdır.

Kanıtlama. Ağacın $\geq n!$ yaprağı olmalıdır, çünkü ortada $n!$ olası permütasyon vardır. Boyu h olan bir ikili ağacın $\leq 2^h$ yaprağı olur. Böylece, $n! \leq 2^h$.

$$
\begin{aligned}
\therefore\ h &\geq \lg(n!) && (\lg \text{ monoton artışlı}) \\
&\geq \lg\left((n/e)^n\right) && (\text{Stirling'in formülü}) \\
&= n \lg n - n \lg e \\
&= \Omega(n \lg n)\,. \qquad \square
\end{aligned}
$$

Karar-ağacı sıralamasında alt sınır

❖**Doğal sonuç:**

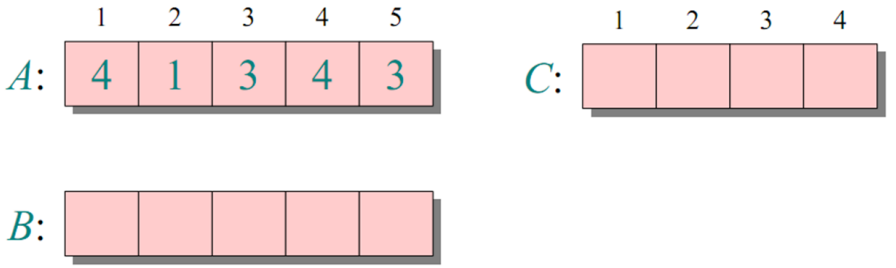
Yığın sıralaması ve ***birleştirme sıralaması*** *asimptotik* olarak **en iyi karşılaştırma sıralaması** algoritmalarıdır.

2. Doğrusal zamanda sıralama

2.1. Sayma sıralaması (Counting Sort):

❖Elemanlar arası karşılaştırma yoktur.

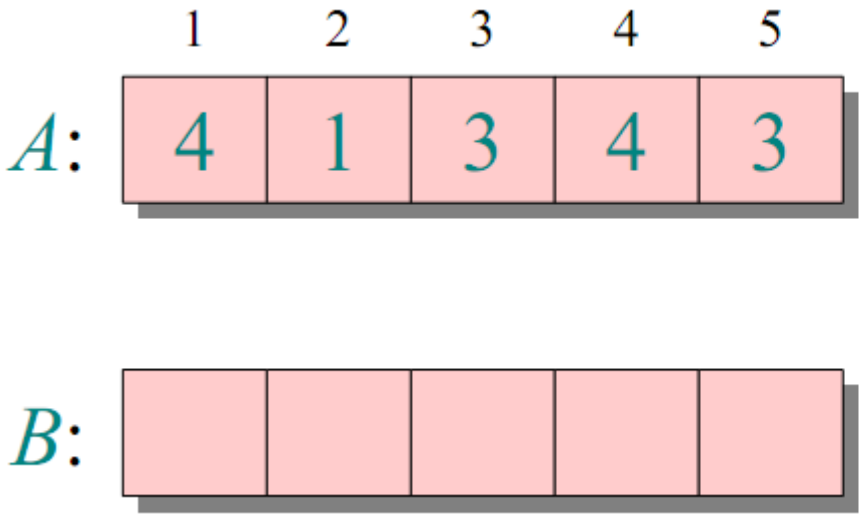
Giriş: **A[1 . . n],** burada A[j]∈{1, 2, …, **k**} .



k, *küçük* ise iyi bir algoritma olur.
k, büyük ise *çok kötü bir algoritma olur.*

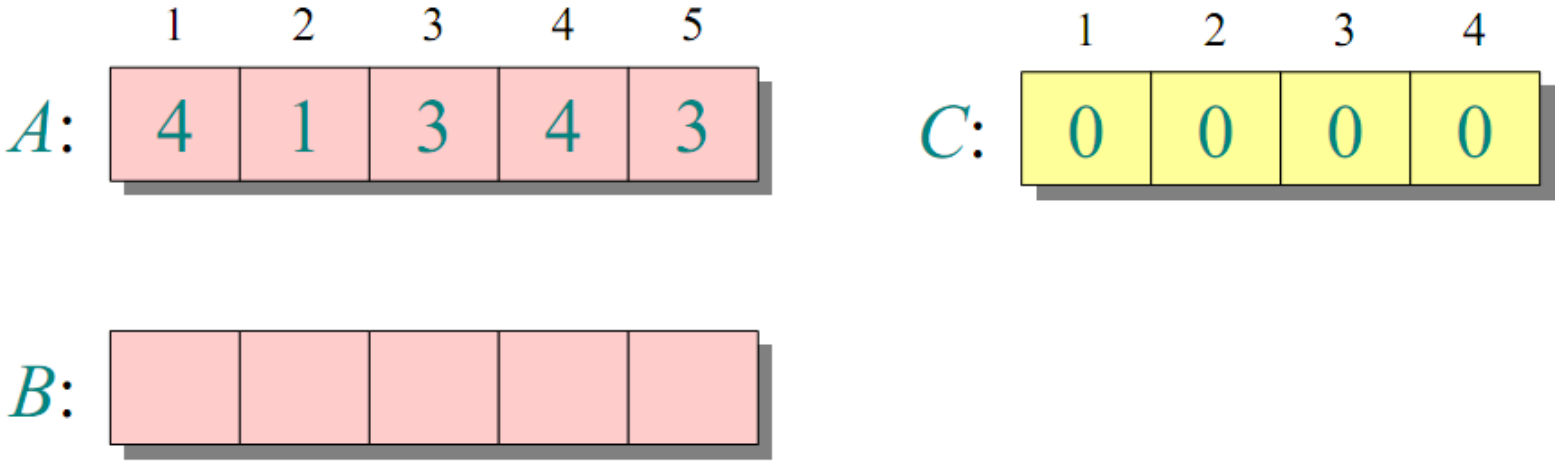
Çıkış: B[1 . . n], sıralı.

Yedek depolama: C[1 . . k] .



* Yedek depolama alanı

- Dizi girişi 1 ile 4 arasındadır.
- O zaman **k=4** olur.
- Bir başka ifade ile dizinin en büyük elemanı bulunur.

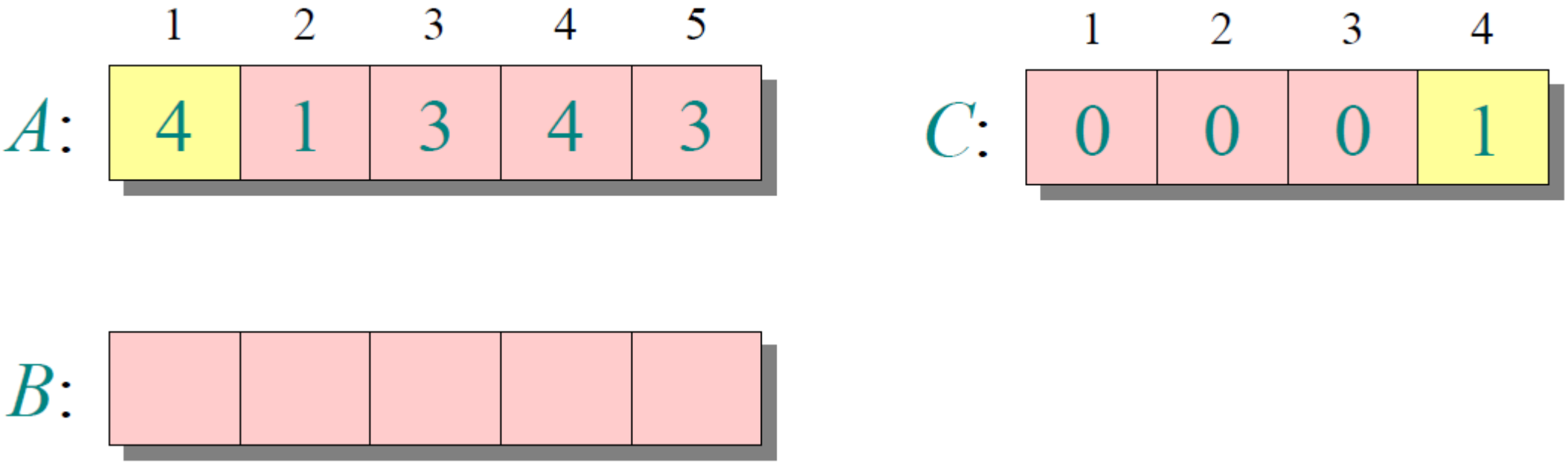


Dizinin elemanları ilk etapta sıfırlanır.

Döngü 1

for $i \leftarrow 1$ **to** k

do $C[i] \leftarrow 0$



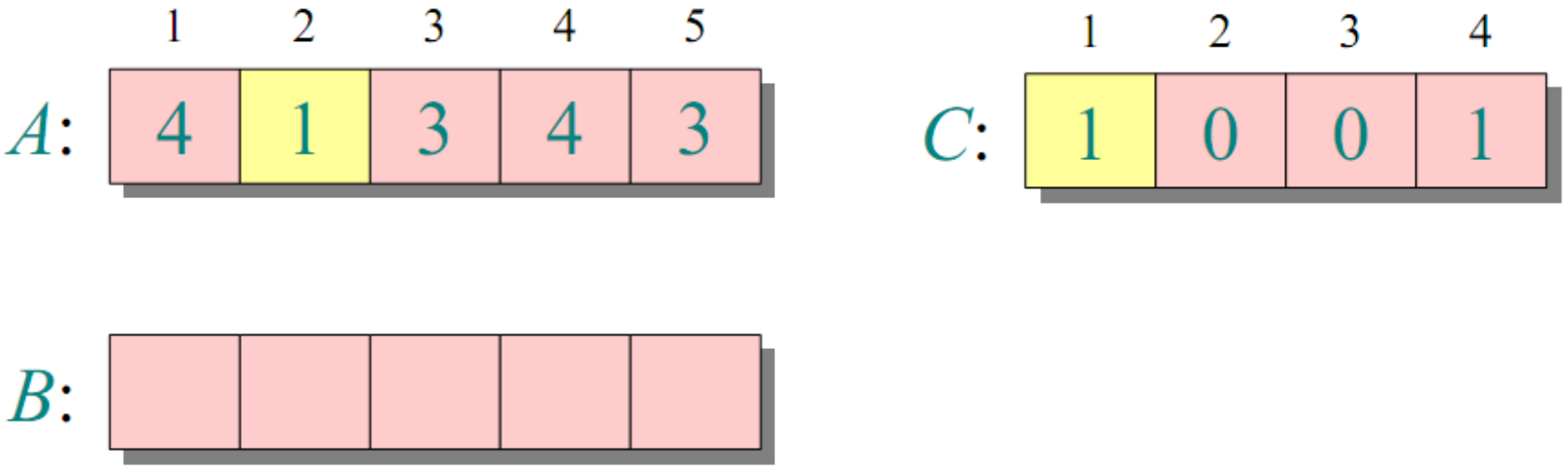
❖ **İlk elemandan başlanarak her bir elemandan kaç tane var sayılır.**

Döngü 2

for $j \leftarrow 1$ **to** n

do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$

Örnek: j=1 — 4 — 4 — => C[4]=0+1=1

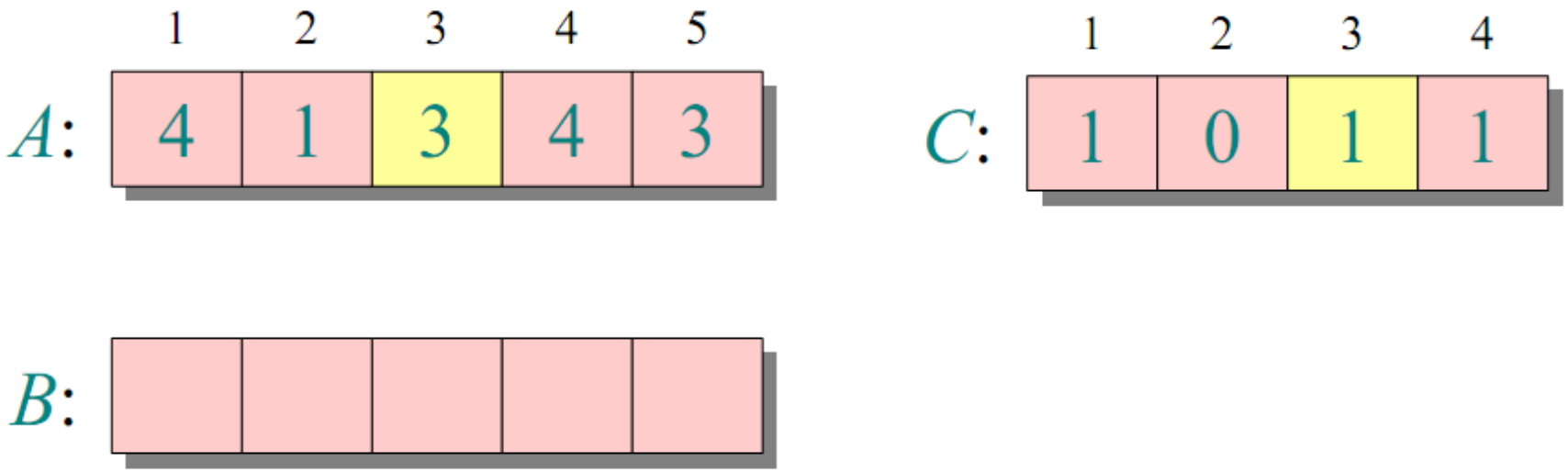


for $j \leftarrow 1$ **to** n

do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$

Döngü 2

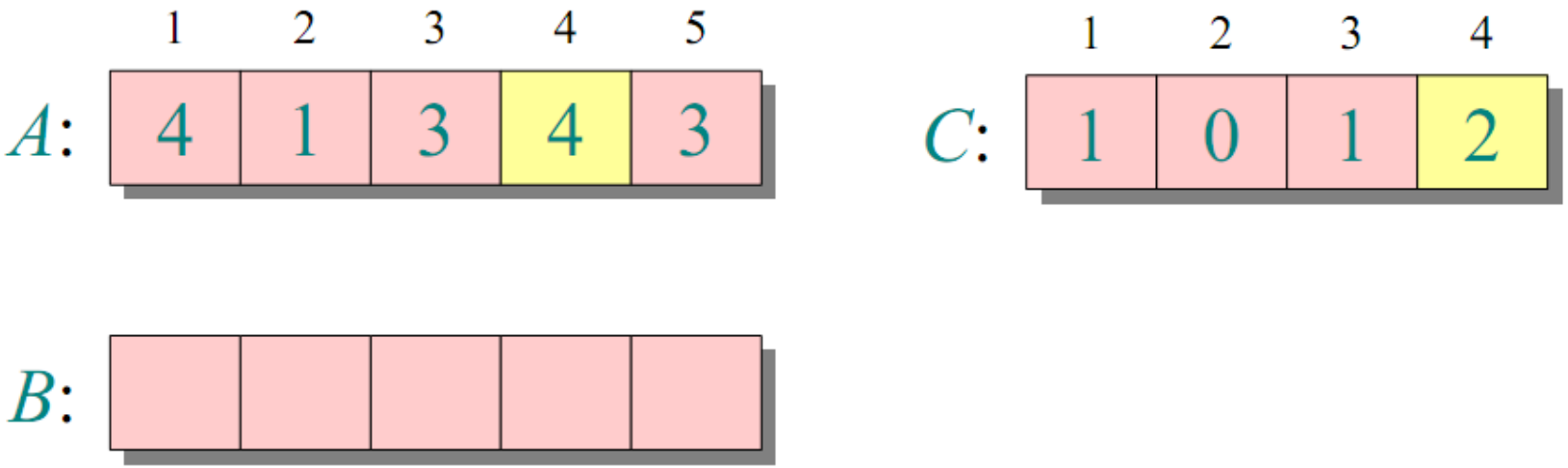
Örnek: j=2 1 1 => C[1]=0+1=1



for $j \leftarrow 1$ **to** n

do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$

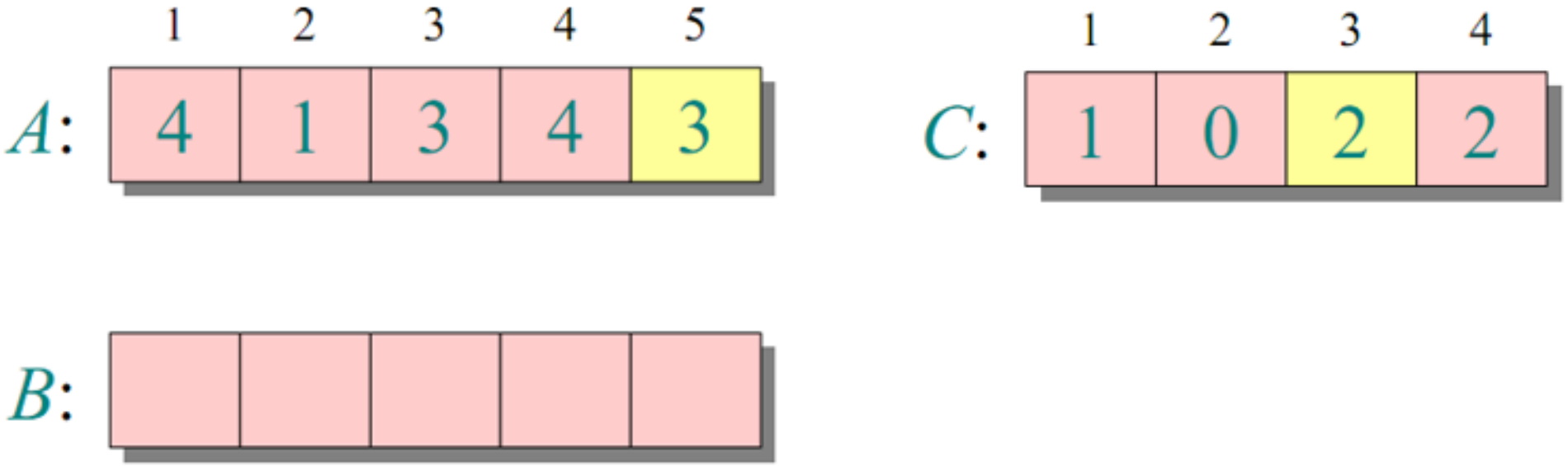
Döngü 2



for $j \leftarrow 1$ **to** n

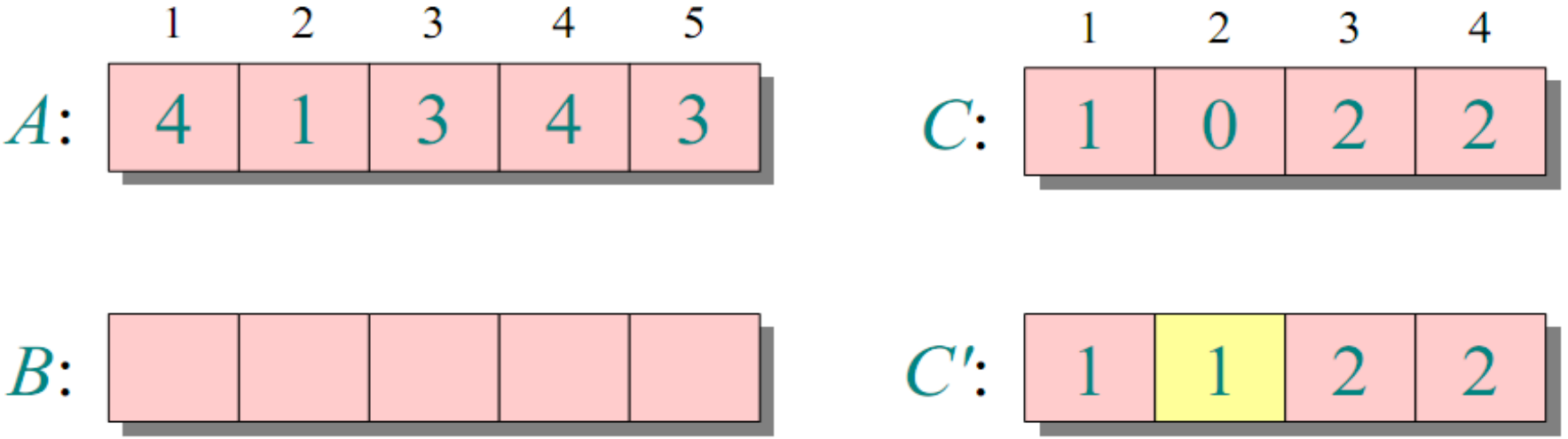
do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$

Döngü 2



for $j \leftarrow 1$ **to** n

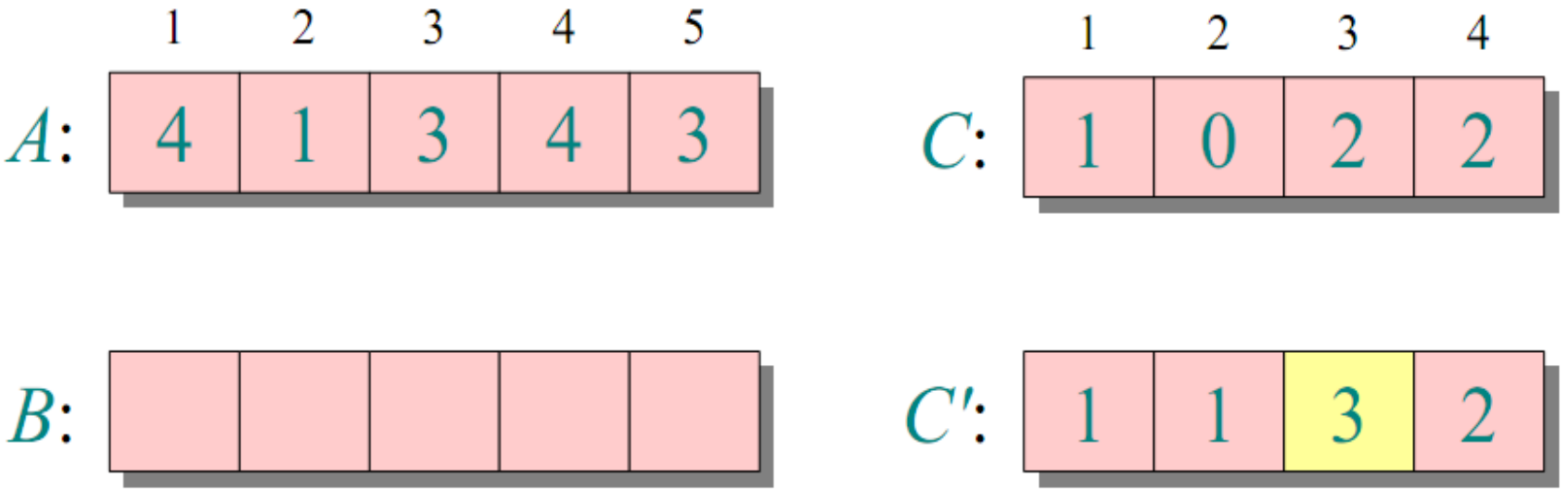
do $C[A[j]] \leftarrow C[A[j]] + 1$ ▷ $C[i] = |\{\text{key} = i\}|$ **Döngü 2**



for $i \leftarrow 2$ **to** k
  do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \le i\}|$

Döngü 3

- Bir elemanın değeri bir önceki elemanın değeriyle **toplanır** ve **elemana** yazılır.
- Bir başka ifadeyle *C[i],* i den küçük veya eşit olan elamanların sayısını içerir.

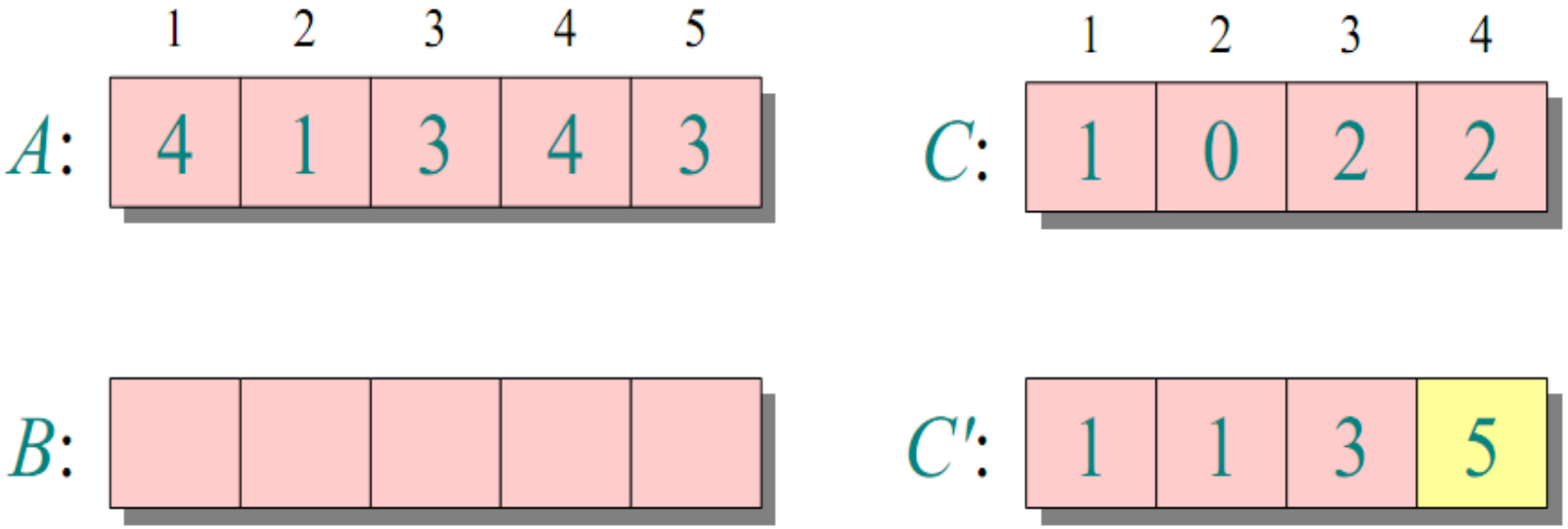


for $i \leftarrow 2$ **to** k

do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \leq i\}|$

Döngü 3

- Bir elemanın değeri bir önceki elemanın değeriyle **toplanır** ve **elemana** yazılır.
- Bir başka ifadeyle *C[i],* i den küçük veya eşit olan elamanların sayısını içerir.



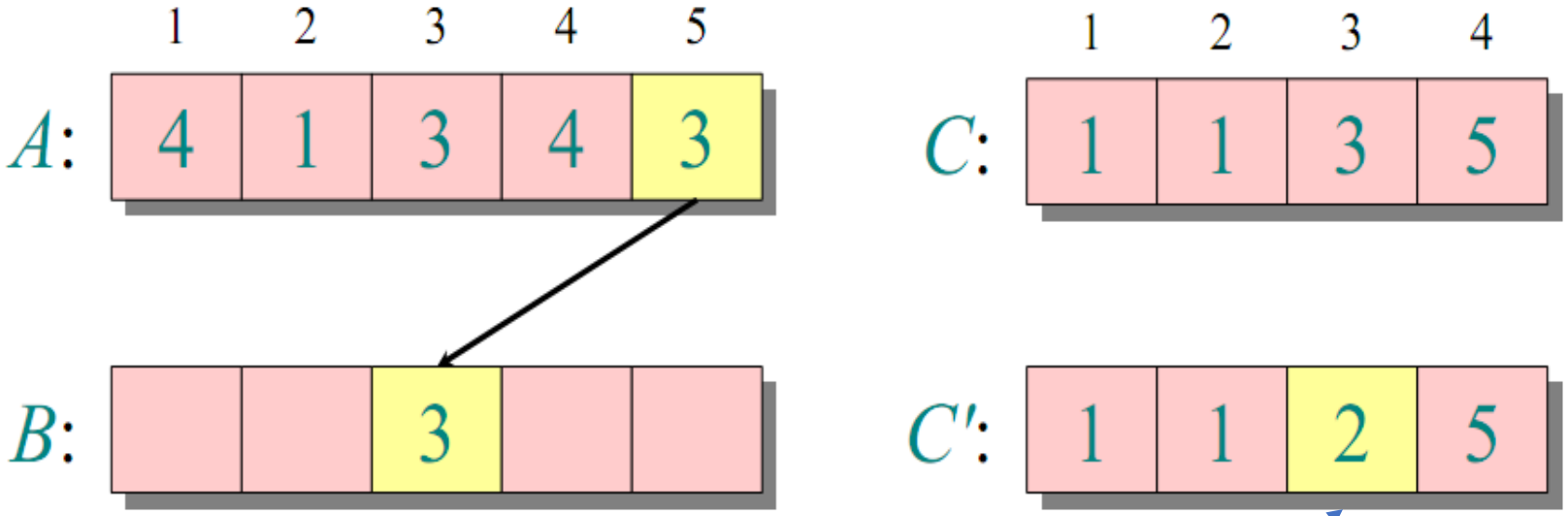
for $i \leftarrow 2$ **to** k

do $C[i] \leftarrow C[i] + C[i-1]$ ▷ $C[i] = |\{\text{key} \le i\}|$

Döngü 3

- Bir elemanın değeri bir önceki elemanın değeriyle **toplanır** ve **elemana** yazılır.
- Bir başka ifadeyle *C[i],* i den küçük veya eşit olan elamanların sayısını içerir.

- **A dizisinin sondan** itibaren elemanı seçilir.
- **A[j] elemanını çıktı dizisi B**'de doğru pozisyona yerleştirir.



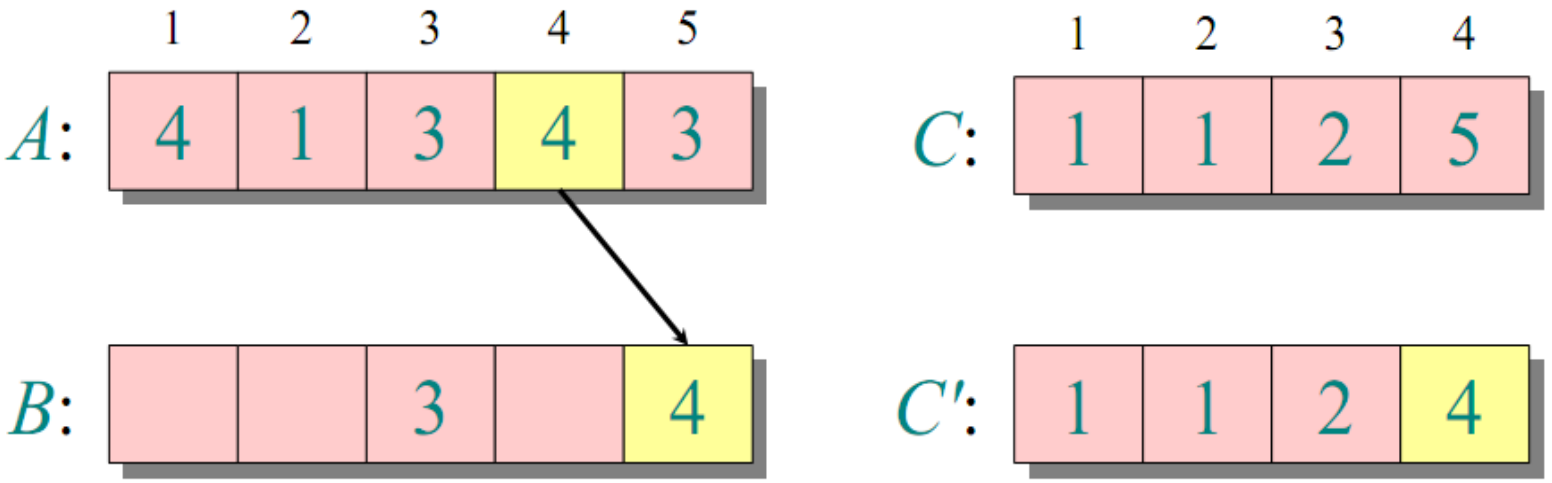
Döngü 4

3

for $j \leftarrow n$ **down to** 1
  do $B[C[A[j]]] \leftarrow A[j]$
    $C[A[j]] \leftarrow C[A[j]] - 1$

Örnek: B[C[**A[5]**]]=B[C[3]]=B[3]= **A[5]**
=>B[3]=3

Örnek: C[**A[5]**]=C[**A[5]**]-1=>C[3]=C[3]-1=>C[3]=2



for $j \leftarrow n$ **down to** 1

do $B[C[A[j]]] \leftarrow A[j]$

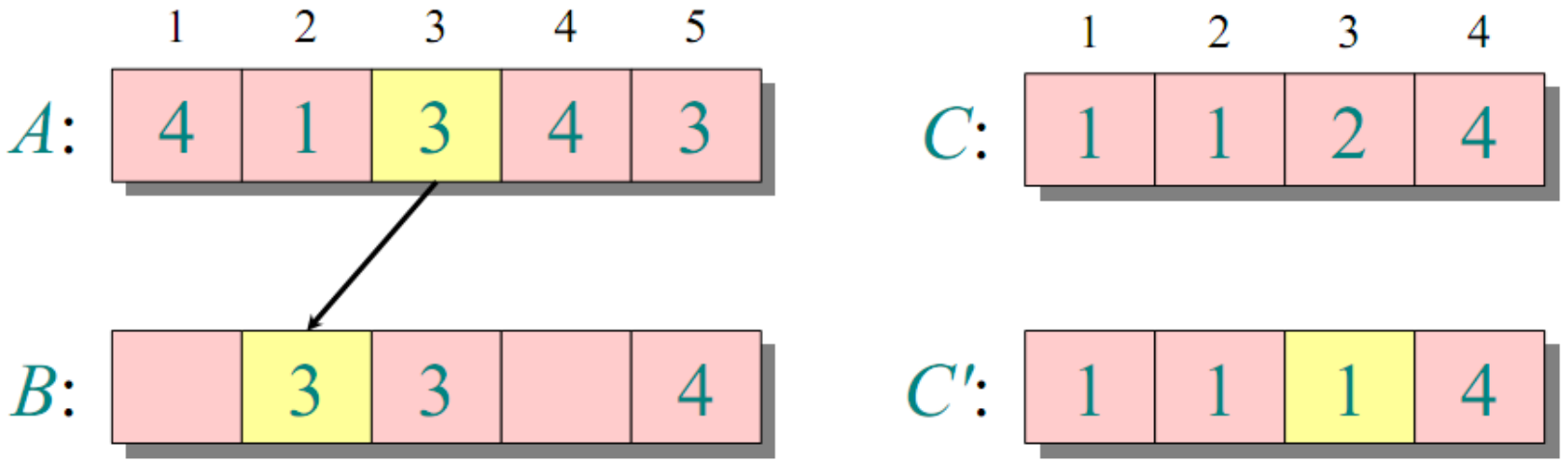
$C[A[j]] \leftarrow C[A[j]] - 1$

Not: Burada kısaca şu şekilde düşünülebilir.

Örneğin A[4]'teki yani 4'ü B'de atayacağım yerin indisi C'de tutulur.

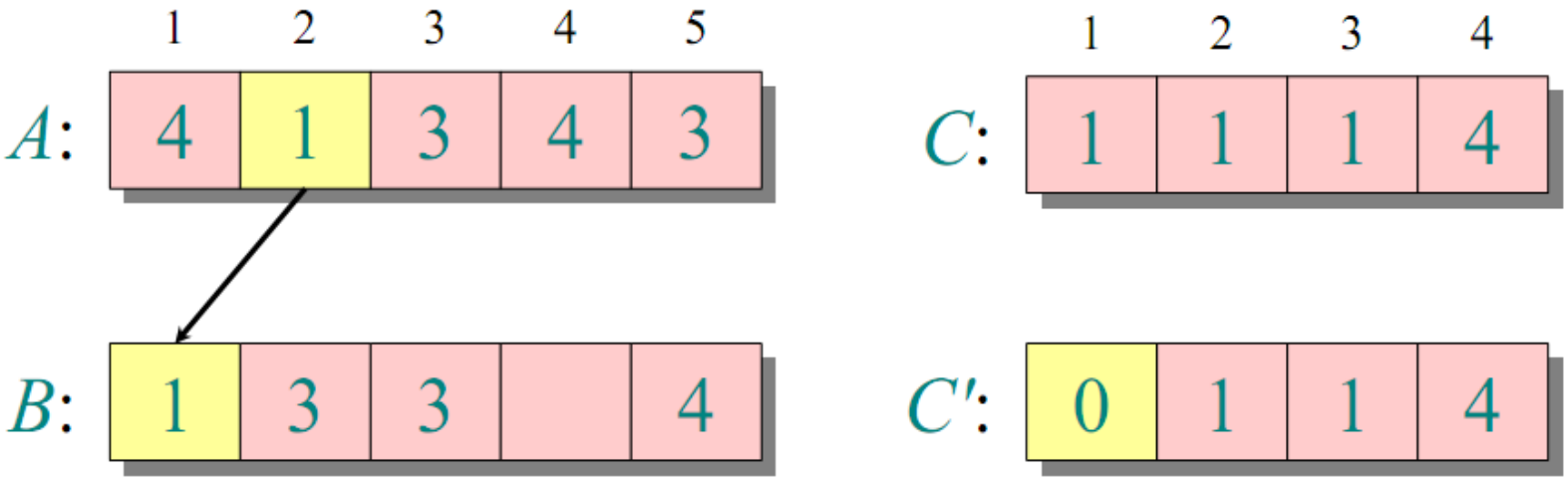
Buda 5. sırada olduğunu gösterir.

C' ise bu sıranın bir azaldığını gösterir.



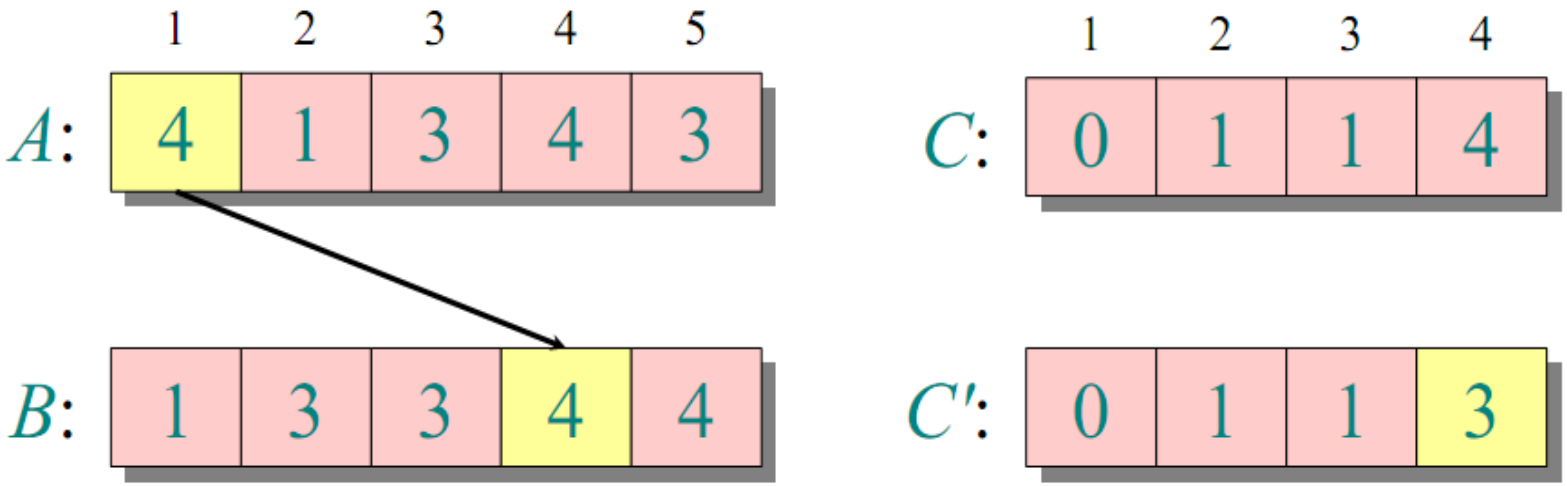
for $j \leftarrow n$ **down to** 1
do $B[C[A[j]]] \leftarrow A[j]$
$C[A[j]] \leftarrow C[A[j]] - 1$

- A dizisinin **sondan birer birer** azaltılarak elemanı seçilir.
- A[j] **elemanını çıktı dizisi B**'de doğru pozisyona yerleştirir.



for $j \leftarrow n$ **down to** 1
do $B[C[A[j]]] \leftarrow A[j]$
$C[A[j]] \leftarrow C[A[j]] - 1$

- A dizisinin **sondan** birer birer azaltılarak elemanı seçilir.
- A[j] **elemanını çıktı dizisi B**'de doğru pozisyona yerleştirir.

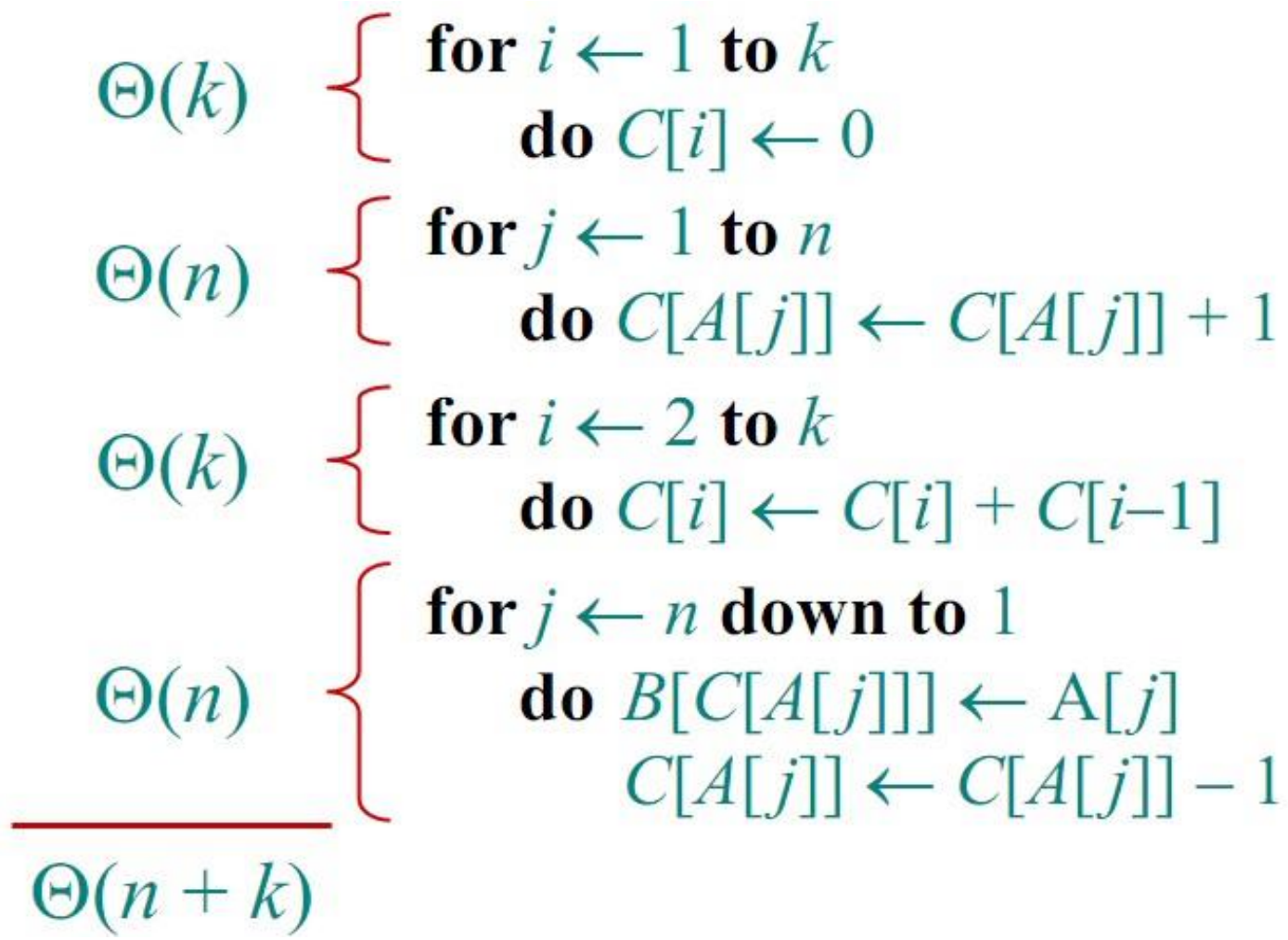


for $j \leftarrow n$ **down to** 1

do $B[C[A[j]]] \leftarrow A[j]$

$C[A[j]] \leftarrow C[A[j]] - 1$

Çözümleme



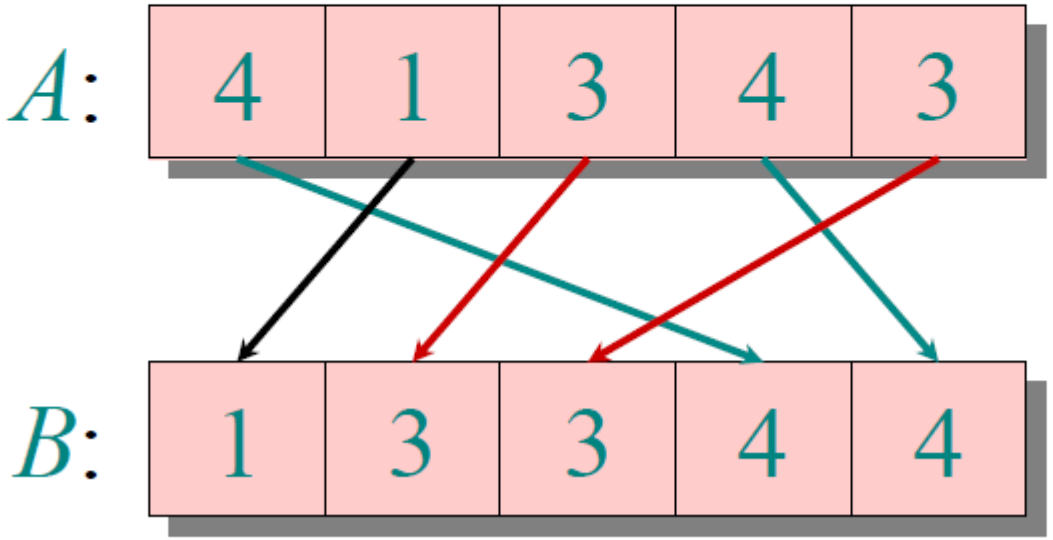
Not: Bu döngüler iç içe değildir.

Çalışma Zamanı

- **$k = O(n)$** ise, sayma sıralaması **$\Theta(n)$** süresi alır.
  Eğer **$k=n^2$** veya **$k=2^n$** çok kötü bir algoritma olur.
- **k** tamsayı olmalı.
- Ama sıralamalar **$\Omega(n \lg n)$** süresi alıyordu! (karar ağacı)
- Hata nerede?
- **Yanıt:**
- Karşılaştırma sıralaması **$\Omega(n \lg n)$** süre alır.
- **Sayma sıralaması bir karşılaştırma sıralaması değildir.**
- Aslında elemanlar arasında bir tane bile karşılaştırma yapılmaz!

Sayma/Sayarak Sıralama

Sayma sıralaması ***kararlı*** bir sıralamadır: eşit eşit elemanlar arasındaki düzeni korur.



Sayma/Sayarak Sıralamanın Avantaj ve Dezavantajları

- **Avantajları:**
  - **n** ve **k** da doğrusaldır (lineer).
  - Kararlı yapıdadır.
  - Kolay uygulanır.
- **Dezavantajları:**
  - **Yerinde sıralama yapmaz.**
  - Ekstra depolama alanına ihtiyaç duyar.
  - Sayıların küçük tam sayı olduğu varsayılır.
  - Byte ise ek dizinin boyutu en fazla 2^8= 256 olur fakat sayılar int ise yani 32 bit lik sayılar ise 2^{32}= 4.2 milyar sayı eder oda yaklaşık **16 Gb yer tutar.**

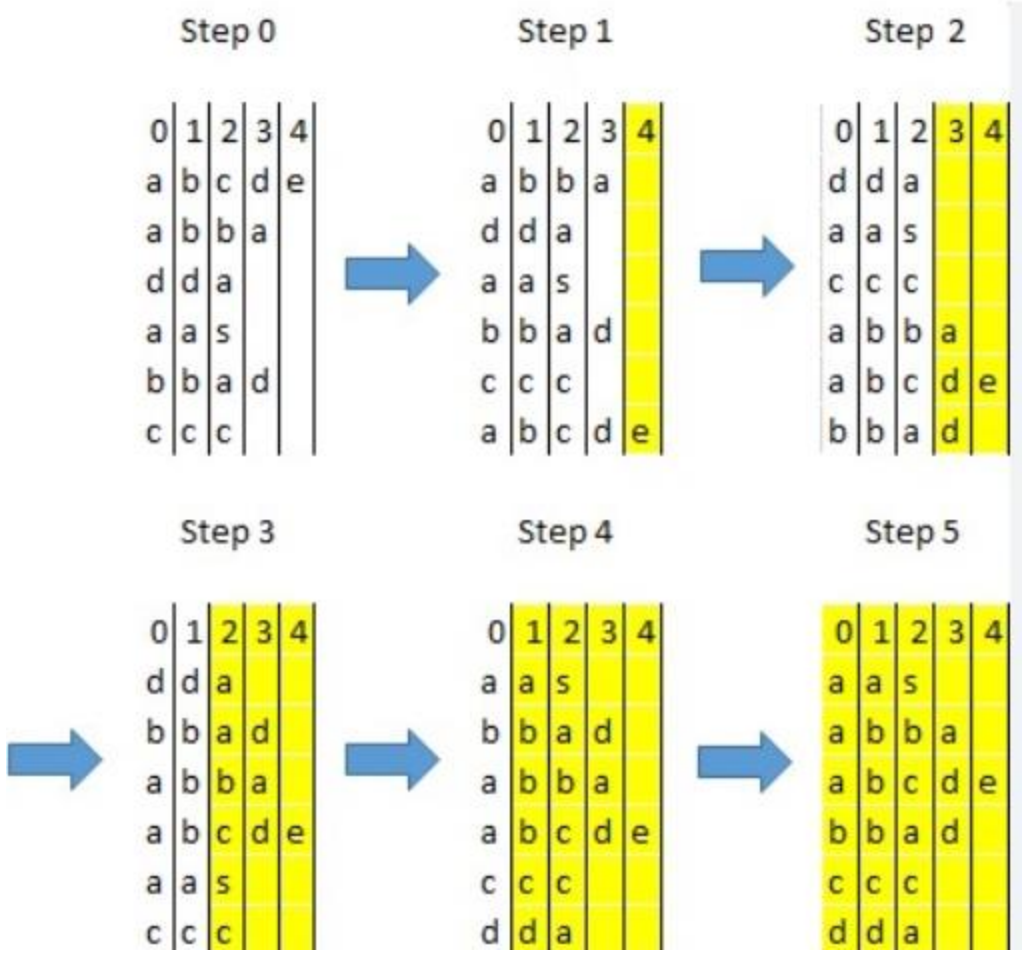
2.2. Taban (Radix) sıralaması

- **Radix Sort,** sayıları basamaklarının üzerinde işlem yaparak sıralayan doğrusal sıralama algoritmalarından biridir.
- **Birden fazla anahtara** göre **sıralama gerektiğinde** **kolaylıkla kullanılabilir.**
- **Örnekler:**

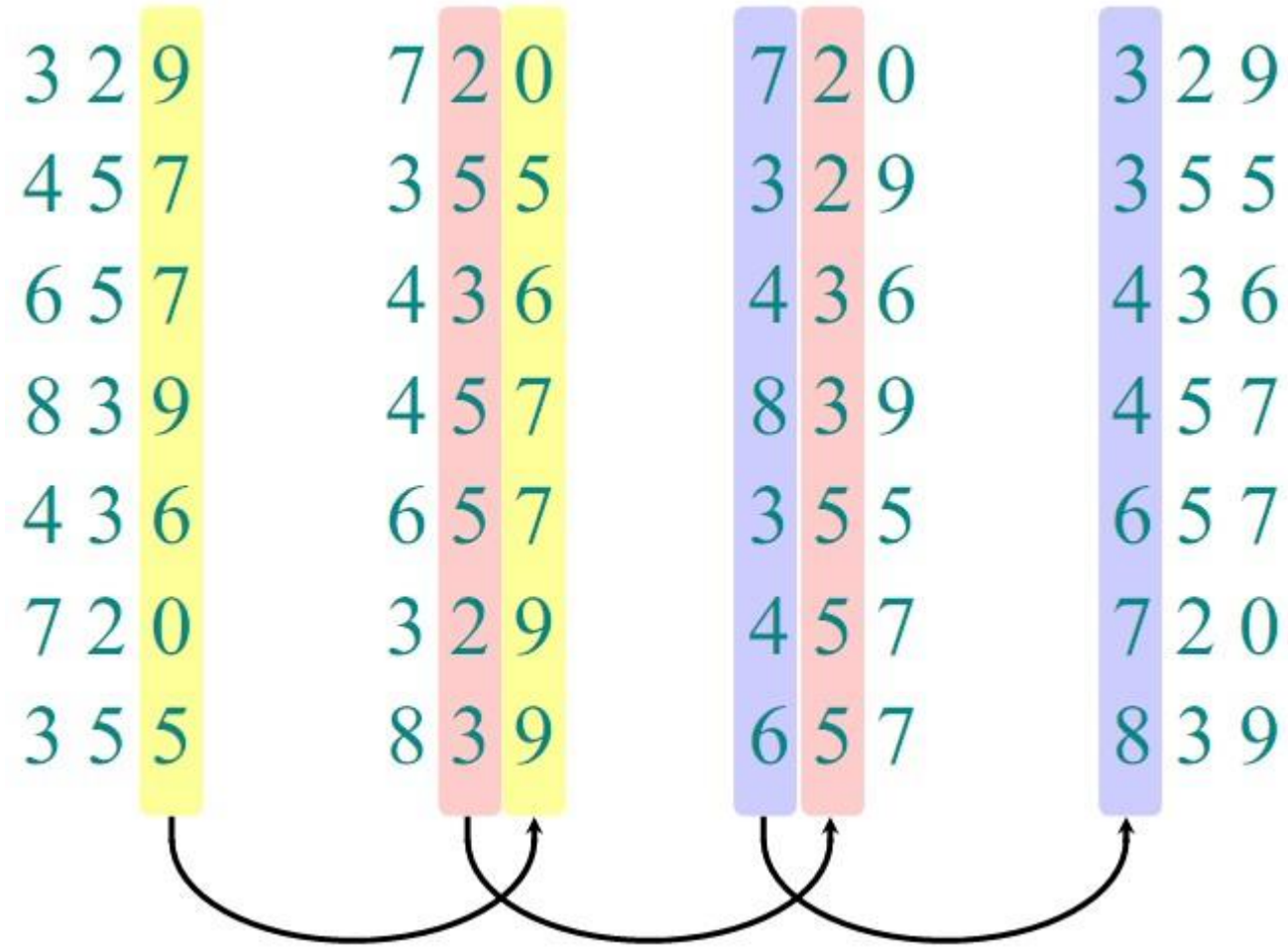
- **Tarihe göre sıralamada** *yıl*, *ay*, *gün*'e göre sıralama yapılır. Tarih sıralamasında önce **gün**, **sonra ay** ve **yıl**' a göre kolayca sıralanır.
- Telefon numaralarının (+1-xxx-yyy-zzzz) sıralanması için kullanılabilir.
- Ondalıklı sayıların sıralanması
- Sözlüksel sıralamalarda da kullanılır.

2.2. Taban (Radix) sıralaması

- **Basamak basamak sıralama.**
- **<u>Kötü fikir:</u>** Sıralamaya en önemli basamaktan başlamak.
- **<u>İyi fikir:</u>** Sıralamaya **en önemsiz basamaktan** başlamak ve **ek kararlı** sıralama uygulamak.



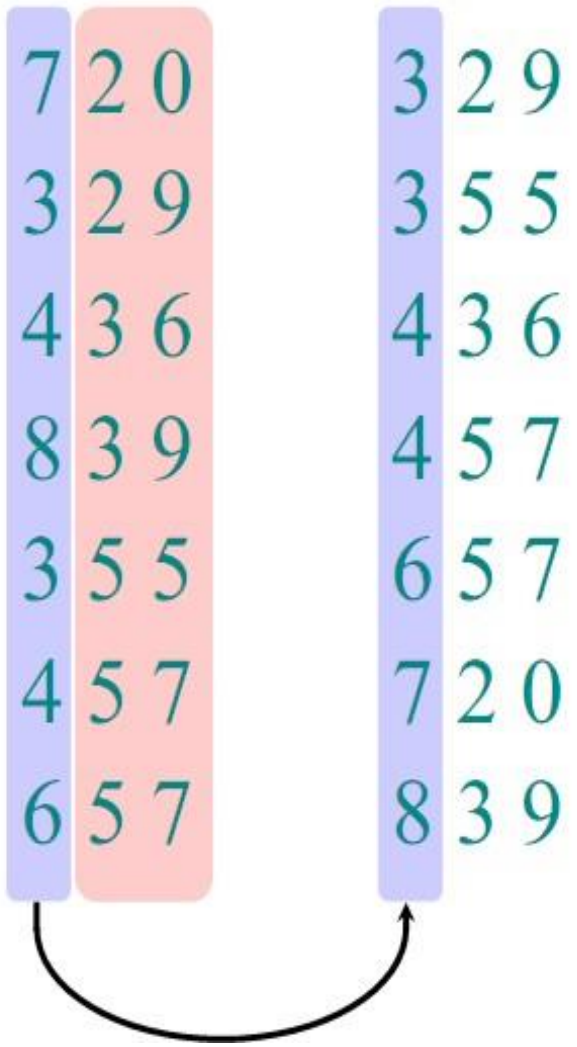
Taban (Radix) sıralaması -örnek



Taban (Radix) sıralaması – örnek 2

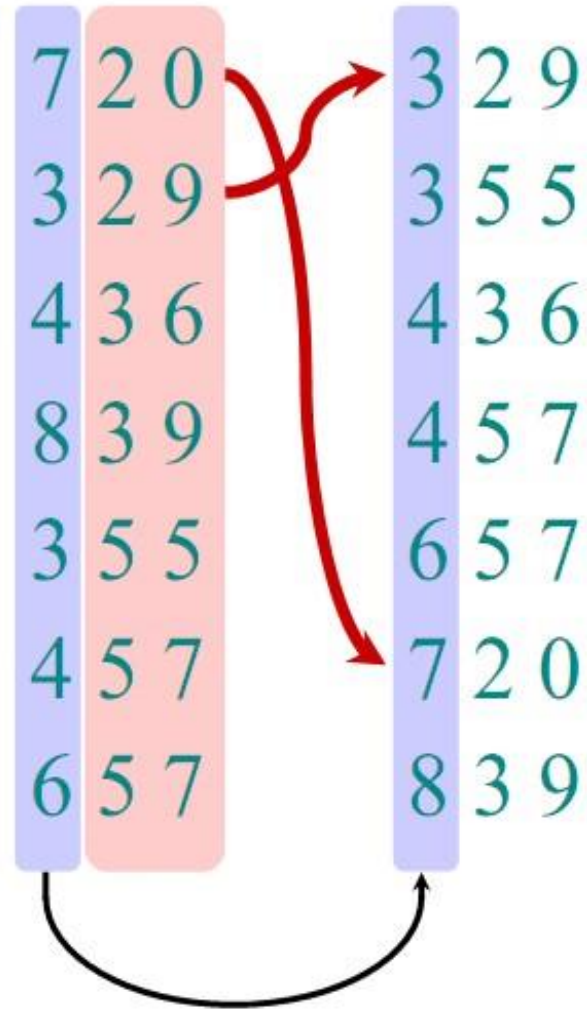
Basamak konumunda tümevarım

- Sayıların düşük düzeyli $t-1$ basamaklarına göre sıralandığını varsayın.
- t basamağında sıralama yapın.



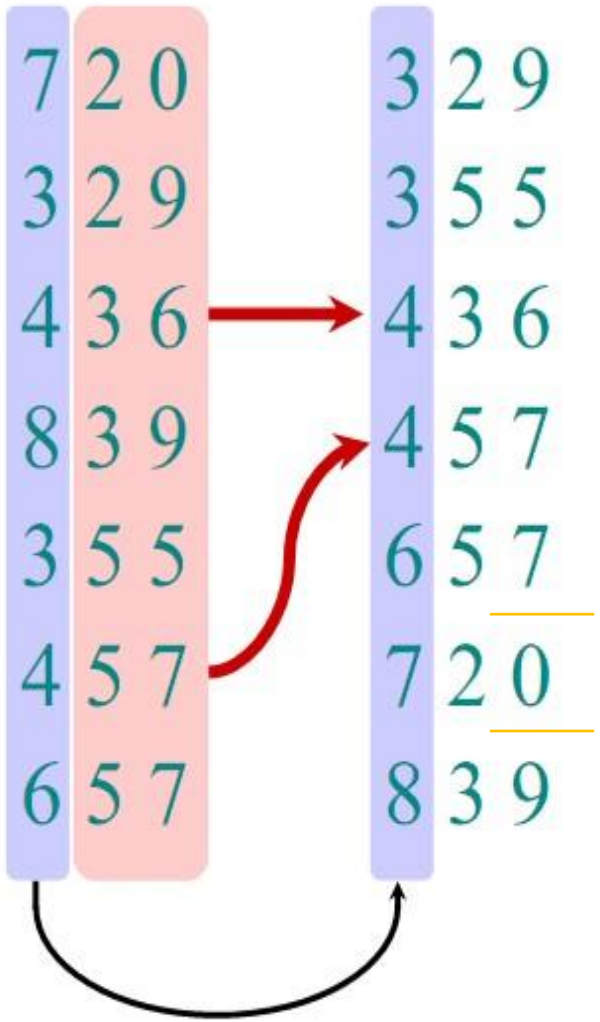
Basamak konumunda tümevarım

- Sayıların düşük düzeyli *t* –*1* basamaklarına göre sıralandığını varsayın.
- *t* basamağında sıralama yapın.
  - *t* basamağında farklı olan iki sayı doğru sıralanmış.



Basamak konumunda tümevarım

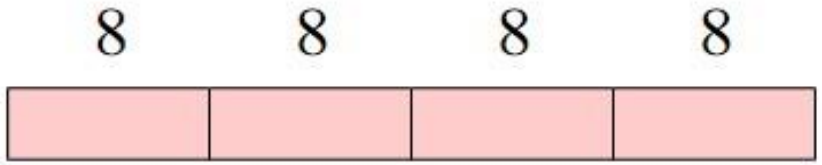
- Sayıların düşük düzeyli $t-1$ basamaklarına göre sıralandığını varsayın.
- t basamağında sıralama yapın.
  - t basamağında farklı olan iki sayı doğru sıralanmış.
  - t basamağındaki iki eşit sayının girişteki sıraları muhafaza edilmiş $\Rightarrow$ doğru sıra.



Taban sıralamasının çözümlemesi

- Sayma sıralamasını ek kararlı sıralama varsayın.
- Herbiri b bit olan n bilgiişlem sözcüğünü sıralayın.
- Her sözcüğün basamak yapısı b/r taban-2^r olarak görülebilir.

Örnek: 32-bit sözcük



$r = 8 \Rightarrow b/r = 4$ ise, taban-2^8 basamak durumunda sıralama 4 geçiş yapar; veya $r = 16 \Rightarrow b/r = 2$ ise, taban-2^{16} basamakta 2 geçiş yapar.

Kaç geçiş yapmalıyız?

Radixsort'un çalışma süresi

❖**Kaç geçiş var?**

❖**Geçiş başına ne kadar iş?**

Örnek (32-bitlik sayılar için):

- En çok 3 geçiş (≥ 2000 sayının sıralanmasında).
- Birleştirme sıralaması /çabuk sıralama $\lceil \lg 2000 \rceil$ en az 11 geçiş yaparlar.

Dezavantajı: Çabuk sıralamanın aksine, taban sıralamasının yer referansları zayıftır ve bu nedenle ince ayarlı bir çabuk sıralama, dik bellek sıradüzeni olan günümüz işlemcilerinde daha iyi çalışır.

* Kısacası bellek gereksinimi, kullanımı çabuk sıralamadan kötüdür.

❖**Toplam zaman?**

❖**Sonuç olarak**

K büyükse gerçekten doğrusal değildir.

❖**Uygulamada RadixSort yalnızca çok sayıda öğe ve nispeten küçük anahtarlar içeren problemler için iyidir.**

- Taban sıralamanın hesaplama karmaşıklığı aşağıdaki gibi formüle edilebilir.

T(n) = O(L*(N+b))

- L, en büyük anahtardaki basamak sayısı veya en fazla uzunluğa sahip dizinin uzunluğudur.
- N sıralanacak öğelerin sayısıdır.
- b sayı sisteminin tabanıdır.

	5th pass	4th pass	3rd pass	2nd pass	1st pass
String 1	z	i	p	p	y
String 2	z	a	p		
String 3	a	n	t	s	
String 4	f	l	a	p	s

- Bu örnekte karmaşıklık, kısaca O(N*L) şeklindedir.

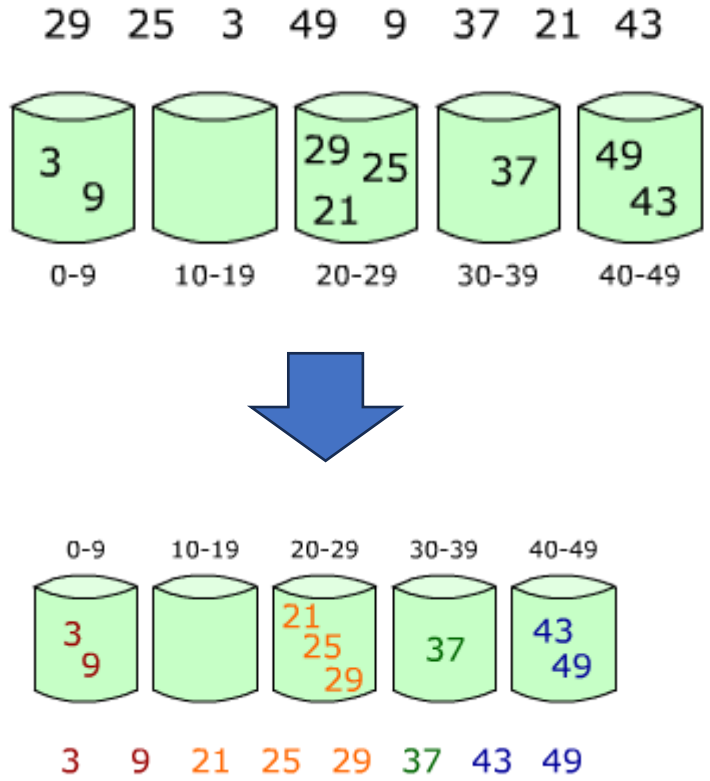
2.3. Kova Sıralama (BucketSort)

- **Kova Sıralaması** (ya da **sepet sıralaması**), sıralanacak bir diziyi **parçalara ayırarak** sınırlı sayıdaki *kovalara* (ya da *sepetlere*) atan bir sıralama algoritmasıdır.
- Ayrışma işleminin ardından her kova kendi içinde **ya farklı bir algoritma kullanılarak** ya da kova sıralamasını özyinelemeli olarak çağırarak sıralanır.

2.3. Kova Sıralama (BucketSort)

Kova sıralaması aşağıdaki biçimde çalışır:

1. Başlangıçta boş olan bir "kovalar" dizisi (belirli aralıklara bölerek) oluştur.
2. Asıl dizinin üzerinden geçerek her öğeyi ilgili aralığa denk gelen kovaya at.
3. Boş olmayan bütün kovaları sırala.
4. Boş olmayan kovalardaki bütün öğeleri yeniden diziye al.

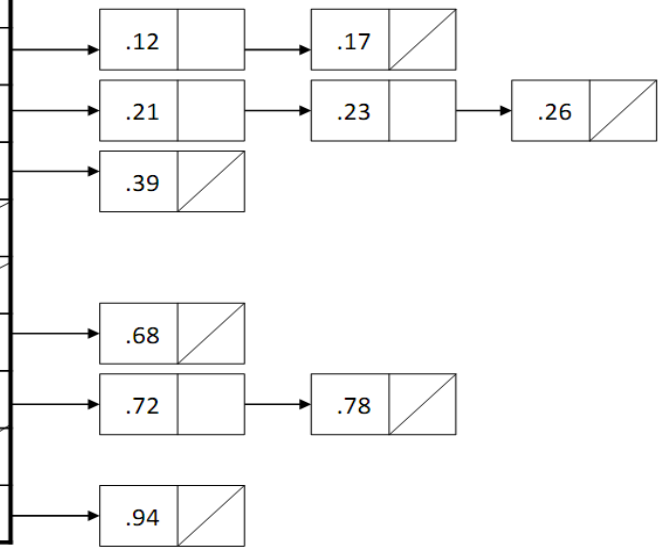


2.3. Kova Sıralama (BucketSort)

- Kova sıralaması **doğrusal zamanda çalışır.**
- Girişin düzgün dağılımlı olduğu kabul edilir.
- Random olarak [0,1) aralığında oluşturulmuş giriş bilgileri olduğu kabul edilir.
- Temel olarak [0, 1) aralığını **n** eşit **alt aralığa böler** ve **girişi bu aralıklara dağıtır.**
- Aralıklardaki değerleri **insert sort (araya sokma)** ile **sıralar.**
- Aralıkları bir biri ardına **ekleyerek sıralanmış diziyi elde eder.**

Örnek

	A		B
1	.78	0	
2	.17	1	
3	.39	2	
4	.26	3	
5	.72	4	
6	.94	5	
7	.21	6	
8	.12	7	
9	.23	8	
10	.68	9	



BUCKET-SORT(A)

1 $n = A.length$
2 let $B[0..n-1]$ be a new array
3 **for** $i = 0$ **to** $n-1$
4 make $B[i]$ an empty list
5 **for** $i = 1$ **to** n
6 insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
7 **for** $i = 0$ **to** $n-1$
8 sort list $B[i]$ with insertion sort
9 concatenate the lists $B[0], B[1], \ldots, B[n-1]$ together in order

Yukarıdaki şekilde n = 10 için BUCKET-SIRALAMA işlemi.

(a) Giriş dizisi A[1..10] **(b)** Algoritmanın 8. satırından sonra sıralanmış listelerin (kovalar) B[0..9] dizisi. Kova ***i***, yarı açık aralıktaki **[i/10, i+1/10)** değerleri tutar.

(b) Sıralanan çıktı, B[0],B[1], ...B[9] listelerinin sırasına göre bir birleştirmeden oluşur.

Karmaşıklık

```
                BUCKET-SORT(A)
O(n)    1   n = A.length
O(n)    2   let B[0..n − 1] be a new array
O(n)    3   for i = 0 to n − 1
O(n)    4       make B[i] an empty list
O(n)    5   for i = 1 to n
O(n)    6       insert A[i] into list B[⌊n A[i]⌋]
O(n)    7   for i = 0 to n − 1
O(n_i^2) 8      sort list B[i] with insertion sort
O(n)    9   concatenate the lists B[0], B[1], ..., B[n − 1] together in order
```

En kötü durum performansı	$O(n^2)$
Ortalama durum performansı	$O(n+k)$

Her i kovasının aynı $E(n_i^2)$ değerine sahip olması şaşırtıcı değildir.

A girdi dizisindeki **her bir değerin** herhangi bir kovaya düşme olasılığı **eşit derecededir (1/n).**

UYGULAMALAR

1. **1000 ile 100000 arasında sayılardan oluşan 100 elemanlı bir sayı dizisi oluşturunuz**.

- Bu diziyi **Radix sort** algoritmasında en önemli ve en önemsiz basamaktan başlanarak iki farklı yaklaşım ile sıralayınız.
- Aradaki farklar neler?
- Her zaman en önemsiz sayı basamağından başlamak avantaj sağlar mı?

2. Yukarıda oluşturulan 100 elemanlı sayı dizisini **counting sort** algoritması ile sıralayınız.

3. Yukarıda oluşturulan 100 elemanlı sayı dizisini bucket sort algoritması ile sıralayınız.

4. Yukarıda oluşturulan 100 elemanlı sayı dizisini sıralamak için en etkili algoritma hangisidir?

5. 10 kişinin adı ve soyadı ve de bu kişilerin doğum tarihini klavyeden girdikten sonra en genç olandan en yaşlı olana doğru bu kişileri listeleyen programı yazınız.

Kaynakça

- Algoritmalar : Prof. Dr. Vasif NABİYEV, Seçkin Yayıncılık
- Algoritmalara Giriş : Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Palme YAYINCILIK
- Algoritmalar : Robert Sedgewick , Kevin Wayne, Nobel Akademik Yayıncılık
- M.Ali Akcayol, Gazi Üniversitesi, Algoritma Analizi Ders Notları
- Doç. Dr. Erkan TANYILDIZI, Fırat Üniversitesi, Algoritma Analizi Ders Notları
- http://www.bilgisayarkavramlari.com