Sıralama Algoritmaları

Sıralama Algoritmaları Analiz Özet

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

Alt Sınırları Sıralama Doğrusal-Zaman (linear time) Sıralaması

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 **Ω(n lgn)** 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.

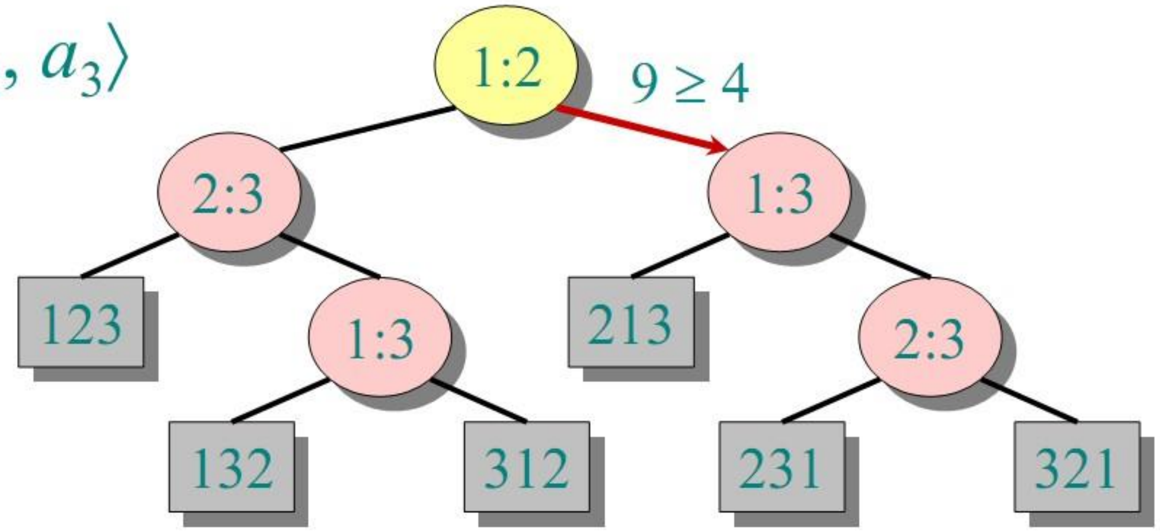
Sıralama İçin Alt Sınırlar

- n sayıda elemanı karşılaştırarak sıralamak için **en kötü durumda** **Ω(n lgn)** 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 $a_i \leq a_j, a_i \geq a_j, a_i >$ değerlendirmemiz yeterli olacaktır.
- **n** tane sayı **n!** düzende karşımıza gelebilir.

Karar-ağacı örneği

- Alt sınır belirlemememizde <u>karar ağaçlarından faydalanabiliriz.</u>

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



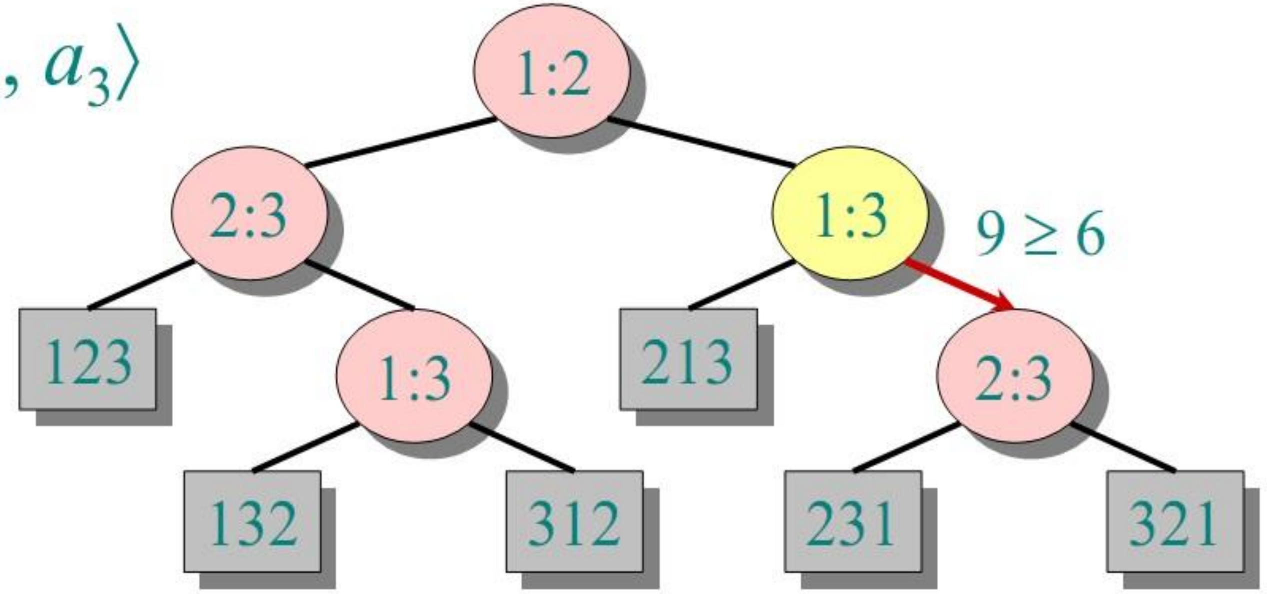
* Araya yerleştirme algoritmasına uyan bir karar ağacı örneği

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.

Karar-ağacı örneği

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

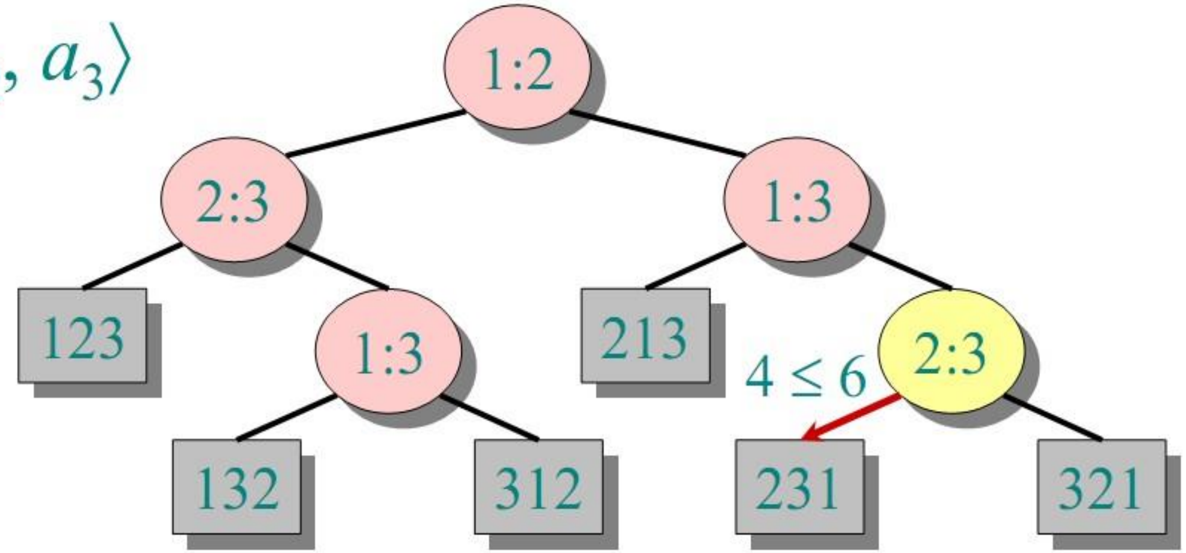


Her iç boğumun etiketlenmesi $i{:}j$; $i, j \in \{1, 2, \ldots, n\}$ için:

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

Karar-ağacı örneği

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

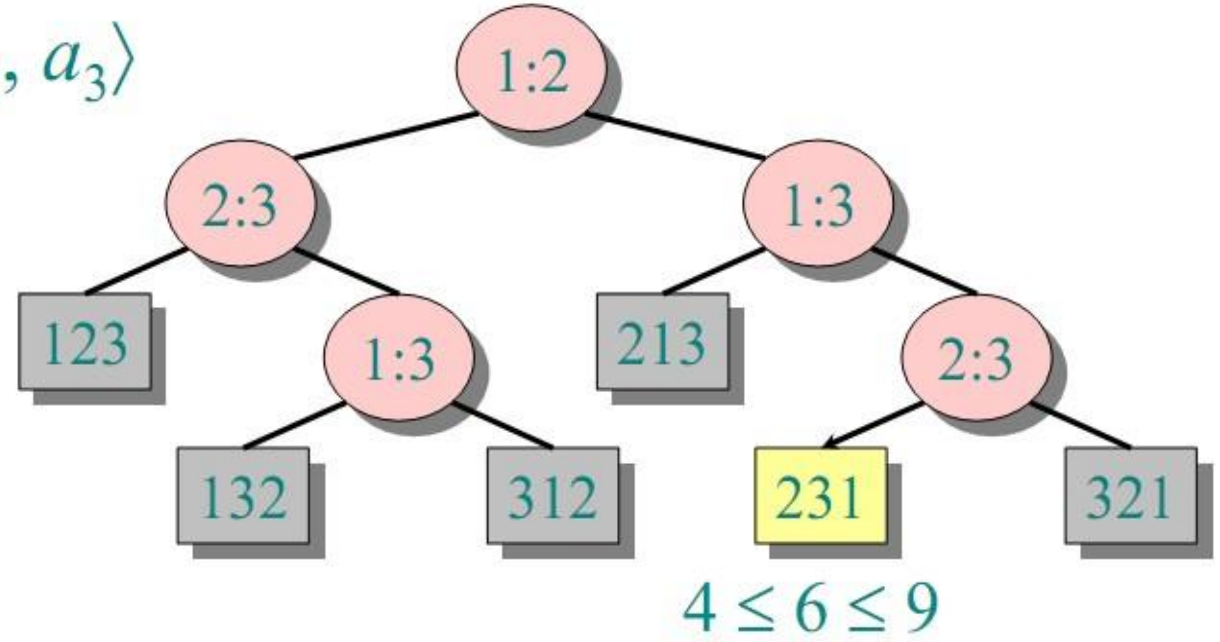


Her iç boğumun etiketlenmesi $i{:}j$; $i, j \in \{1, 2, \ldots, n\}$ için.

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

Karar-ağacı örneği

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 $a_{\pi(1)} \le a_{\pi(2)} \le \cdots \le a_{\pi(n)}$ 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, $n! \leq 2^h$.

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

Karar-ağacı sıralamasında alt sınır

- **Doğal sonuç:**

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

Doğrusal zamanda sıralama

❖ Sayma sıralaması (Counting Sort):

Elemanlar arası karşılaştırma yok.

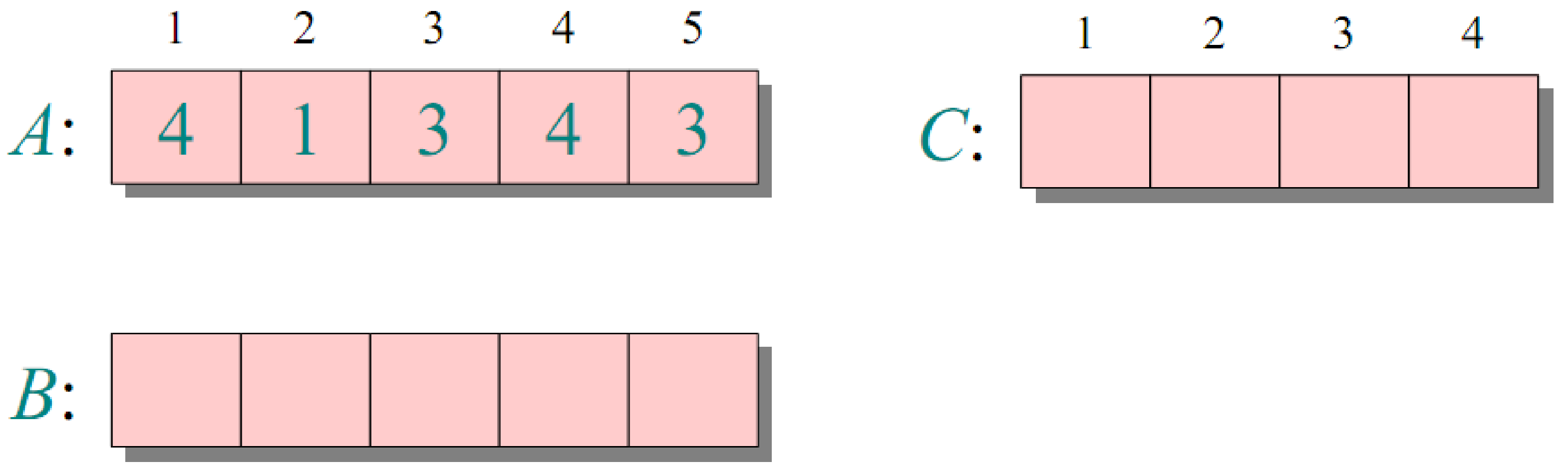
Giriş: A[1 . . n], burada A[j]$\in${1, 2, …, k} .

- **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] .

Sayma/Sayarak Sıralama

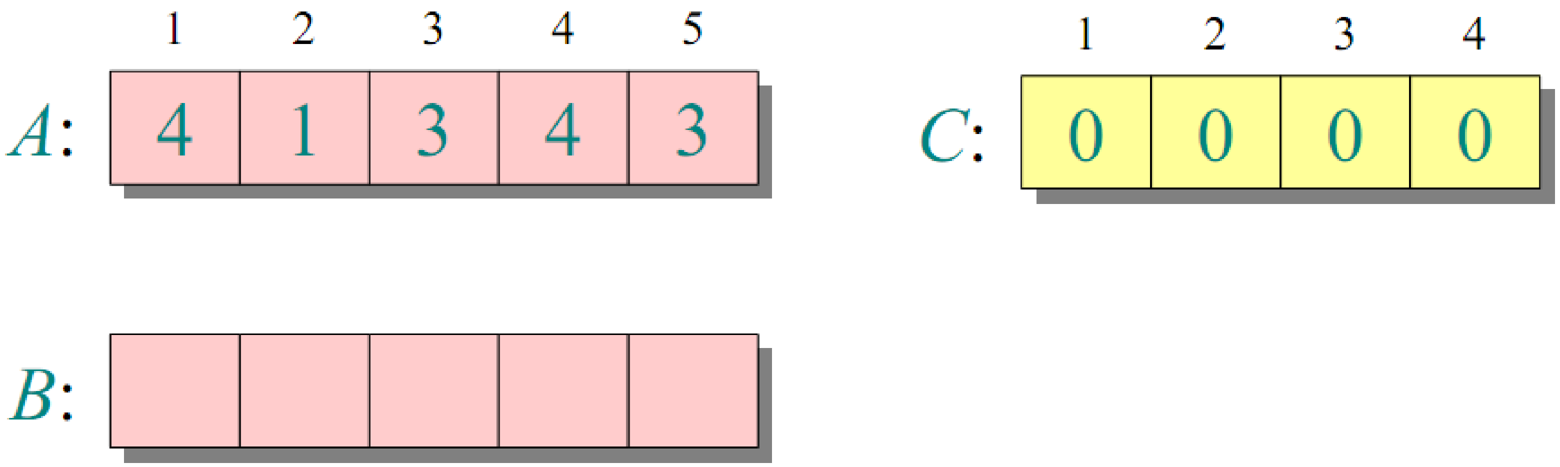


* Yedek depolama alanı

- Dizi girişi 1 ile 4 arasındadır. O zaman **k=4** olur.

Bir başka ifade ile dizinin en büyük elemanı bulunur.

Sayma/Sayarak Sıralama



Dizinin elemanları ilk etapta sıfırlanır.

Döngü 1

for $i \leftarrow 1$ **to** k
    do $C[i] \leftarrow 0$

Sayma/Sayarak Sıralama

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	0	0	0	1

B:					

İlk elemandan başlanarak her bir elemandan kaç tane var sayılır.

Döngü 2

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

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

Sayma/Sayarak Sıralama

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	0	1

B:

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

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

Döngü 2

Sayma/Sayarak Sıralama

A:

1	2	3	4	5
4	1	3	4	3

C:

1	2	3	4
1	0	1	1

B:

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

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

Sayma/Sayarak Sıralama

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	1	2

B:					

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

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

Döngü 2

Sayma/Sayarak Sıralama

	1	2	3	4	5
A:	4	1	3	4	3

	1	2	3	4
C:	1	0	2	2

B:					

Döngü 2

for $j \leftarrow 1$ **to** n
  do $C[A[j]] \leftarrow C[A[j]] + 1 \quad \triangleright\ C[i] = |\{\text{key} = i\}|$

Sayma/Sayarak Sıralama

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	2	2

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

do $C[i] \leftarrow C[i] + C[i-1]$ $\quad \triangleright\ C[i] = |\{\text{key} \leq i\}|$ **Döngü 3**

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

Sayma/Sayarak Sıralama

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	2

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

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

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.

Sayma/Sayarak Sıralama

	1	2	3	4	5
A:	4	1	3	4	3
B:					

	1	2	3	4
C:	1	0	2	2
C':	1	1	3	5

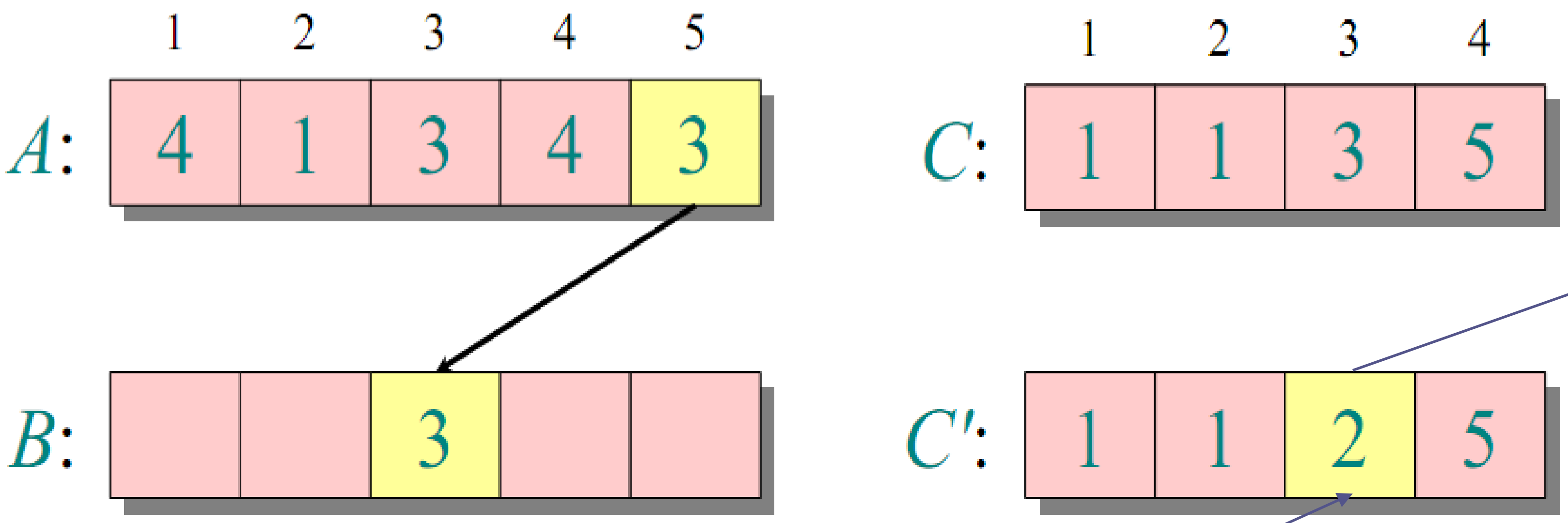
for $i \leftarrow 2$ **to** k

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

Döngü 3

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

Sayma/Sayarak Sıralama



*** 3’ten küçük ve eşit 2 tane eleman var.**
Dolayısıyla bu sayı 3. indekste yer alacaktır.

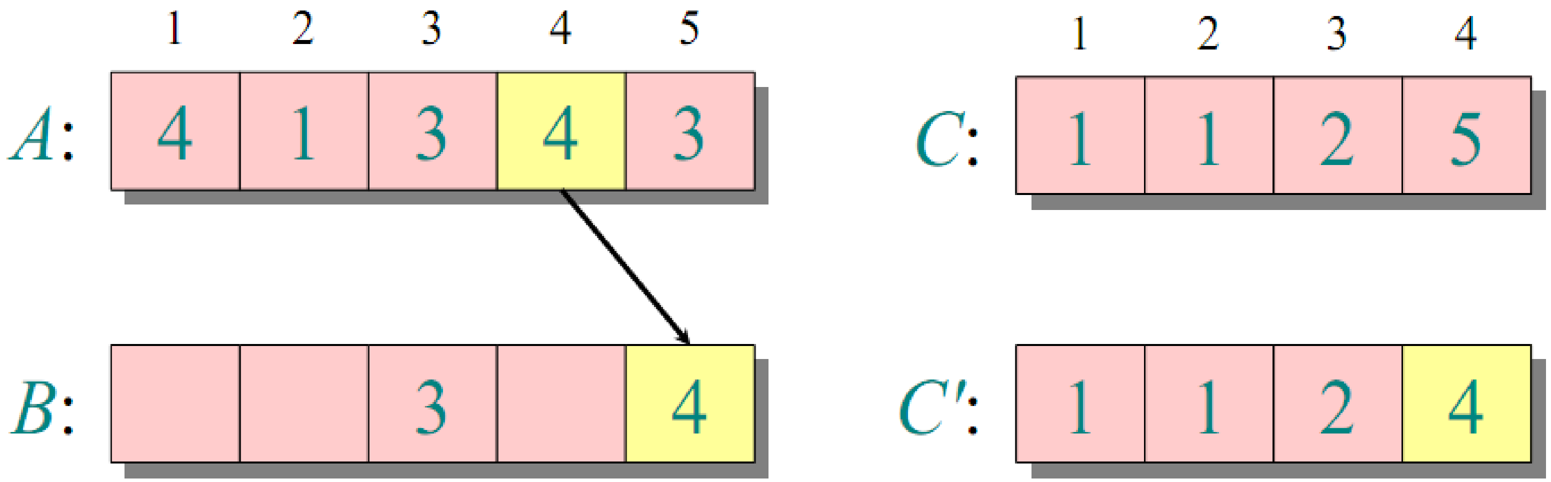
3

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

Döngü 4

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

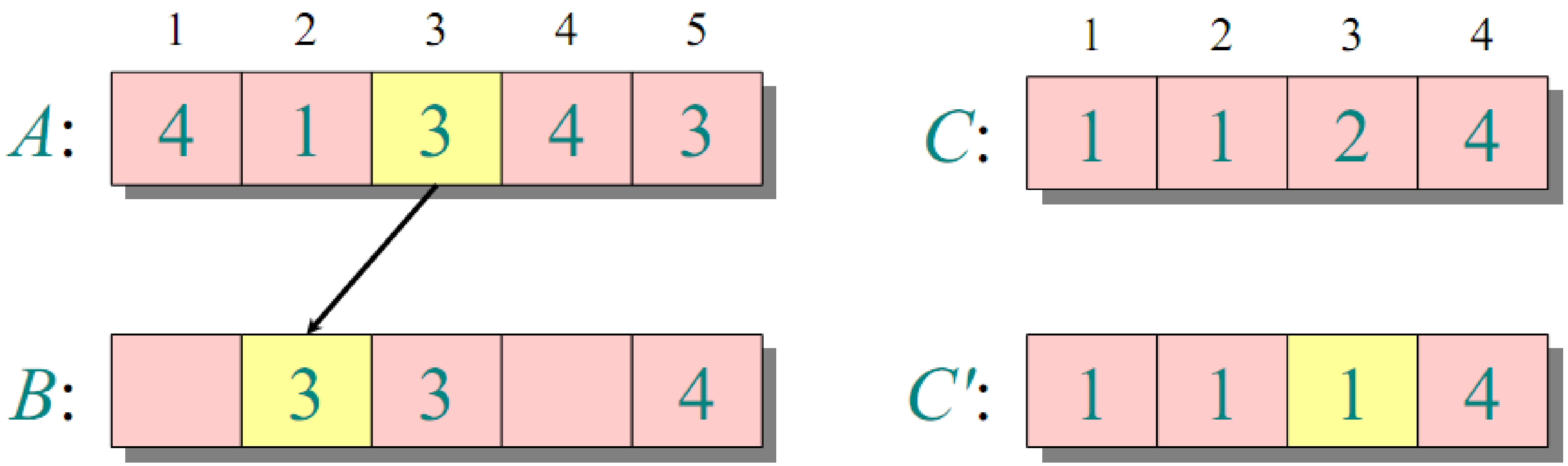
Sayma/Sayarak Sıralama



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

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

Sayma/Sayarak Sıralama



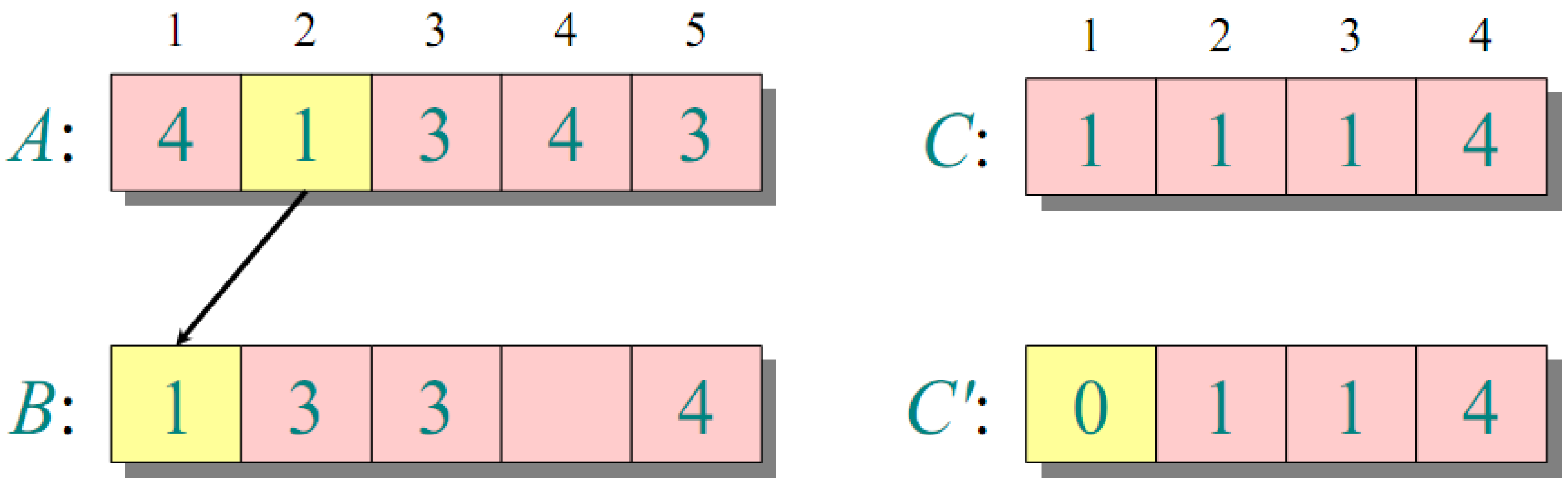
for $j \leftarrow n$ **down to** 1

  do $B[C[A[j]]] \leftarrow \mathrm{A}[j]$

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

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

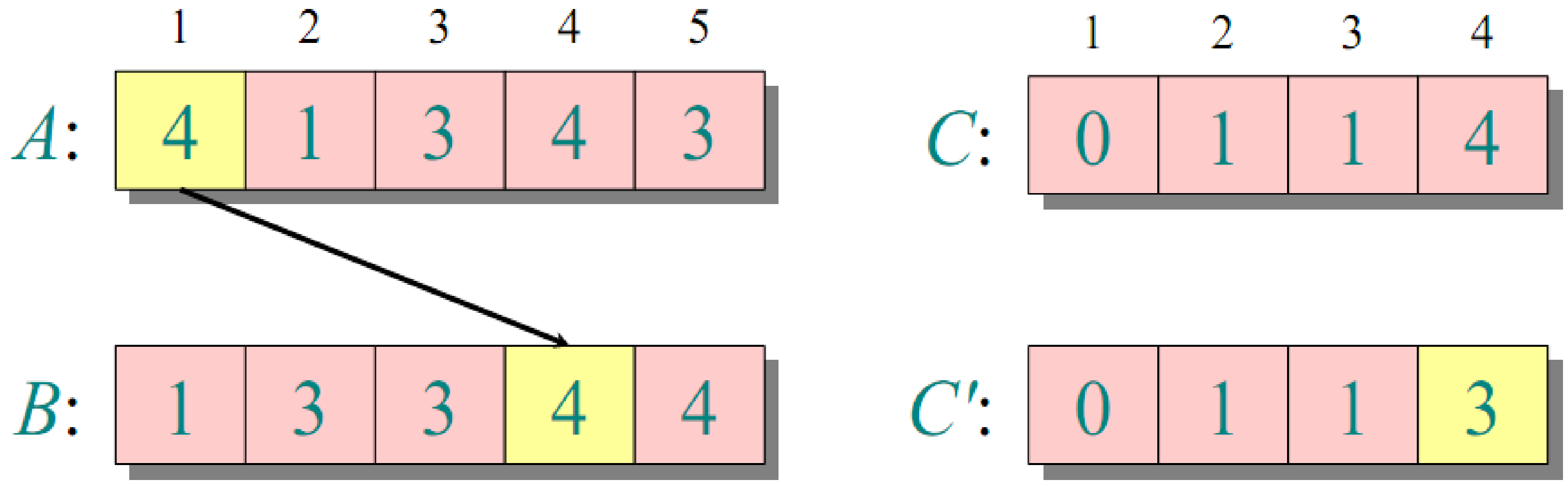
Sayma/Sayarak Sıralama



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.

Sayma/Sayarak Sıralama



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

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

Çözümleme

Not: Bu döngüler iç içe değildir.

$$\Theta(k)\quad\left\{\begin{array}{l} \textbf{for } i \leftarrow 1 \textbf{ to } k \\ \qquad \textbf{do } C[i] \leftarrow 0 \end{array}\right.$$

$$\Theta(n)\quad\left\{\begin{array}{l} \textbf{for } j \leftarrow 1 \textbf{ to } n \\ \qquad \textbf{do } C[A[j]] \leftarrow C[A[j]] + 1 \end{array}\right.$$

$$\Theta(k)\quad\left\{\begin{array}{l} \textbf{for } i \leftarrow 2 \textbf{ to } k \\ \qquad \textbf{do } C[i] \leftarrow C[i] + C[i-1] \end{array}\right.$$

$$\Theta(n)\quad\left\{\begin{array}{l} \textbf{for } j \leftarrow n \textbf{ down to } 1 \\ \qquad \textbf{do } B[C[A[j]]] \leftarrow \mathrm{A}[j] \\ \qquad\qquad C[A[j]] \leftarrow C[A[j]] - 1 \end{array}\right.$$

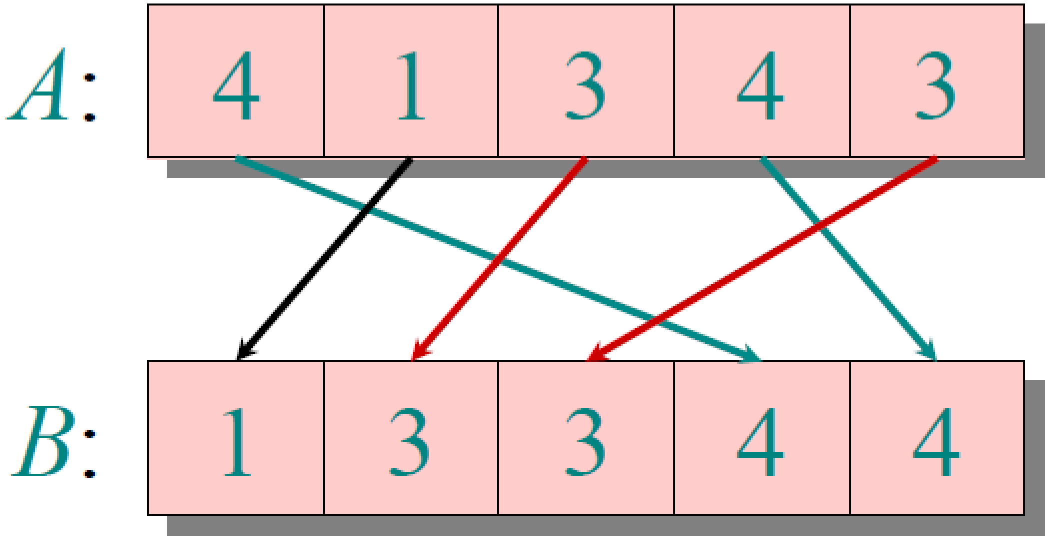
$$\Theta(n + k)$$

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.**

Taban (Radix) sıralaması

- **En az öneme sahip** basamaktan *başlayarak* sıralama yapar.
- **Birden fazla anahtara** göre **sıralama gerektiğinde kolaylıkla kullanılabilir.**
- Örneğin **tarihe göre sıralamada** *yıl*, *ay*, *gün*'e göre sıralama yapılır. Tarih sıralamasında önce **gün**, **sonra ay** ve **yıl**' a göre kolayca sıralanır.

Taban (Radix) sıralaması

- **Basamak basamak sıralama.**
- **Kötü fikir:** sıralamaya önceli en önemli basamaktan başlamak.
- **İyi fikir:** Sıralamaya **en önemsiz basamaktan** başlamak ve **ek kararlı** sıralama uygulamak.

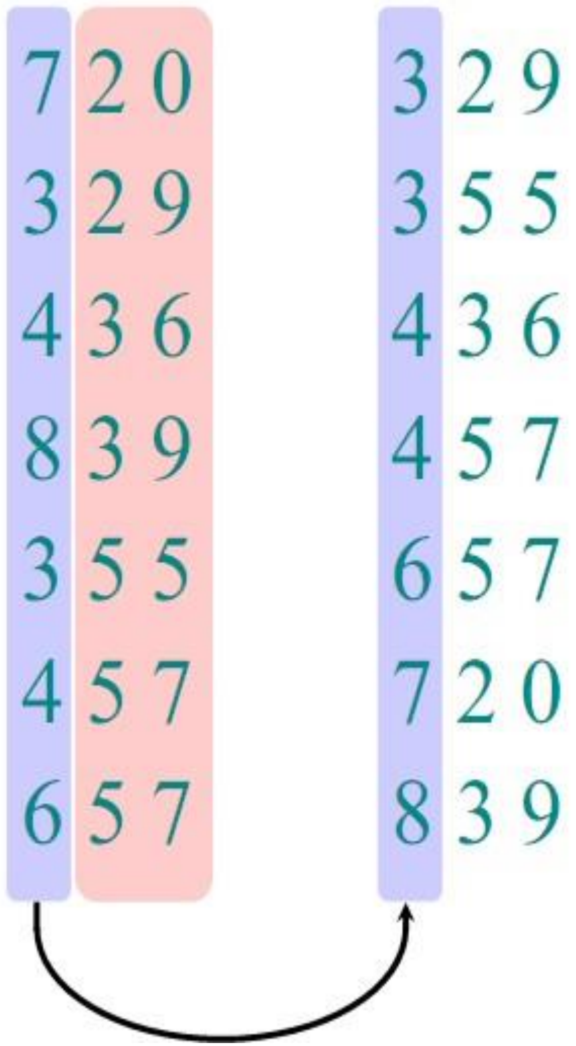
Taban (Radix) sıralaması -örnek

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

Taban (Radix) sıralaması – örnek 2

Basamak konumunda tümevarım

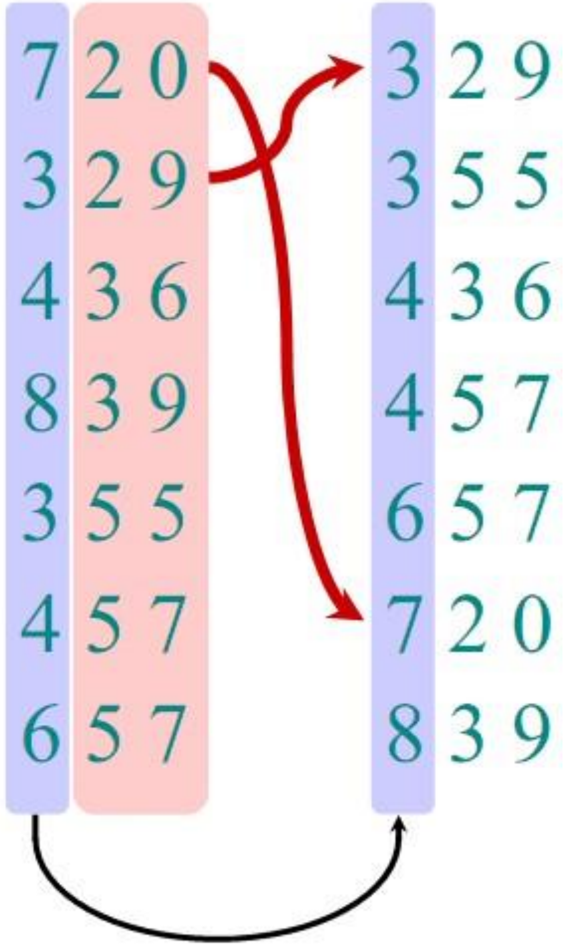
- Sayıların düşük düzeyli $t-1$ basamaklarına göre sıralandığını varsayın.
- t basamağında sıralama yapın.



Taban (Radix) sıralaması – örnek 2

Basamak konumunda tümevarım

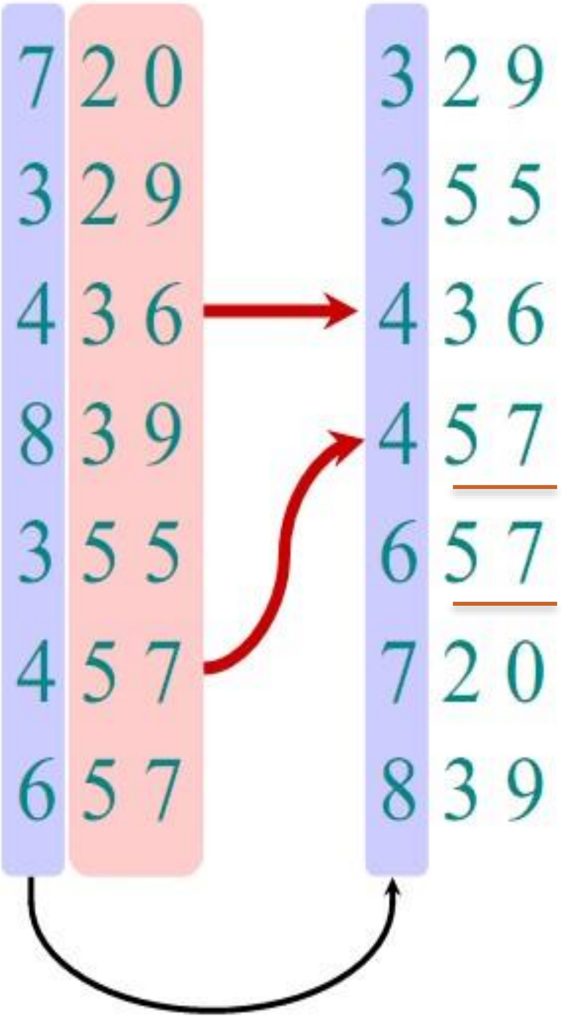
- Sayıların düşük düzeyli *t –1* basamaklarına göre sıralandığını varsayın.
- *t* basamağında sıralama yapın.
  - *t* basamağında farklı olan iki sayı doğru sıralanmış.



Taban (Radix) sıralaması – örnek 2

Basamak konumunda tümevarım

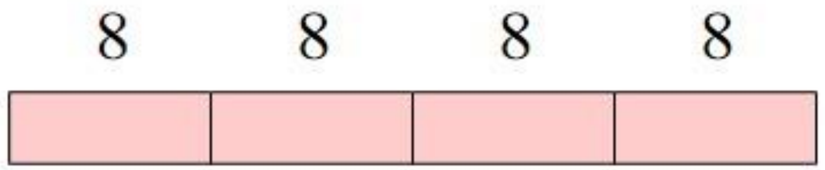
- Sayıların düşük düzeyli *t* –*1* basamaklarına göre sıralandığını varsayın.
- *t* basamağında sıralama yapın.
  - *t* basamağında farklı olan iki sayı doğru sıralanmış.
  - *t* basamağındaki iki eşit sayının girişteki sıraları muhafaza edilmiş ⇒ doğru sıra.



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?

Taban sıralamasının çözümlemesi

Hatırla: Sayma sıralaması $\Theta(n + k)$ süresini alır; (0 ile $k - 1$ aralığında n sayıyı sıralamak için).

Her b-bitlik sözcük r-bitlik parçalara ayrılırsa, sayma sıralamasının her geçişi $\Theta(n + 2^r)$ süre alır. Bu durumda b/r geçiş olduğundan, elimizde:

$$T(n,b) = \Theta\left(\frac{b}{r}\left(n + 2^r\right)\right)$$ olur.

r' yi, $T(n,b)$' yi en aza düşürecek gibi seçin:

- r'yi arttırmak daha az geçiş demektir, ama $r >> \lg n$ olduğundan, süre üstel olarak artar.

r' yi seçmek

(r’ye göre)

$T(n,b)$'yi türevini alıp 0' a eşitleyerek en aza düşürün.

Veyahut da , istemediğimiz değer $2^r >> n$ olduğundan, bu sınırlamaya bağlı kalarak r'yi olabildiğince büyük seçmenin asimptotik bir sakıncası olmadığını gözleyin.

$r = \lg n$ seçimi $T(n,b) = \Theta(bn/\lg n)$ anlamına gelir.

- 0 ile $n^d - 1$ aralığındaki sayılarla $b = d \lg n$ 'yi

elde ederiz. $\Rightarrow$ taban sıralaması $\Theta(d\,n)$ süresini alır.

Not: Değer aralığı 0 … 2^b -1= 0 … n^d kabul edip her iki tarasın logaritması alınır. **b=dlogn** olur. Burada d **<u>basamak sayısıdır.</u>**

Sonuçlar

Pratikte taban sıralaması büyük girişler için hızlıdır; aynı zamanda kod yazması ve bakımı kolaydır.

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.

Pratikte radix sort yani taban sıralaması, <u>sayılarınız gerçekten küçük değilse</u> çok hızlı bir algoritma <u>değildir</u>.

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.

Kova Sıralama (BucketSort)

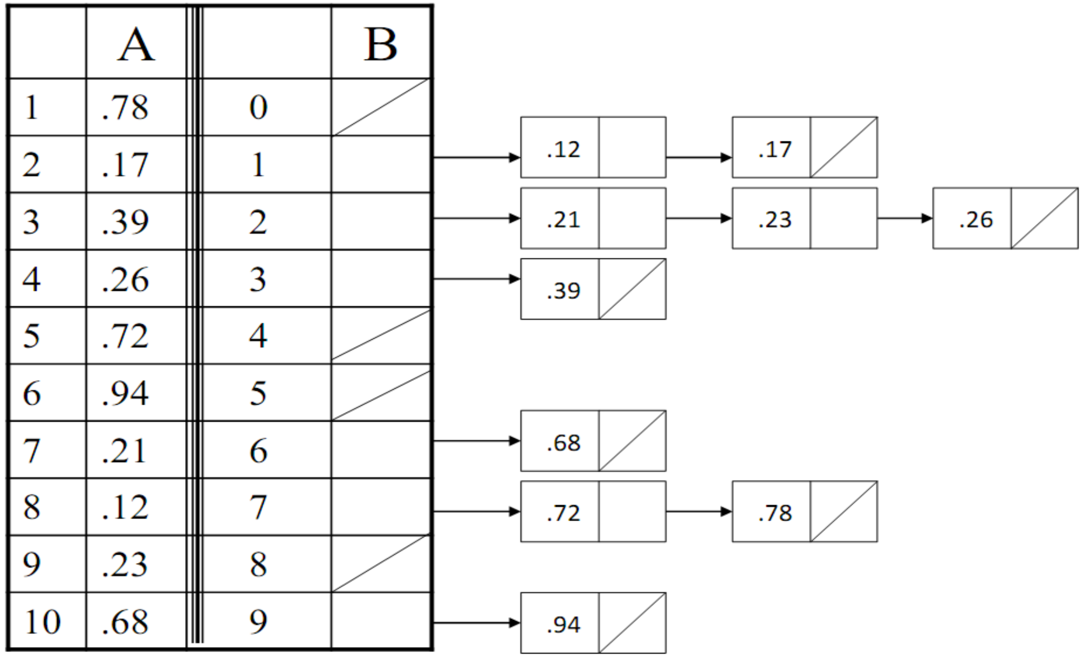
Kova sıralaması aşağıdaki biçimde çalışır:

- Başlangıçta boş olan bir "kovalar" dizisi oluştur.
- Asıl dizinin üzerinden geçerek her öğeyi ilgili aralığa denk gelen kovaya at.
- Boş olmayan bütün kovaları sırala.
- Boş olmayan kovalardaki bütün öğeleri yeniden diziye al.

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 <u>**girişi bu aralıklara dağıtır.**</u>
- Aralıklardaki değerleri **insert sort (araya sokma)** ile **sıralar.**
- Aralıkları bir biri ardına <u>**ekleyerek sıralanmış diziyi elde eder.**</u>

Kova/Sepet Sıralama (Bucket Sort)




```
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[⌊n A[i]⌋]
7  for i = 0 to n − 1
8          sort list B[i] with insertion sort
9  concatenate the lists B[0], B[1], ..., 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. Sıralanan çıktı, B[0],B[1], ...B[9] listelerinin sırasına göre bir birleştirmeden oluşur.

Kova/Sepet Sıralama (Bucket Sort)

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

BUCKET-SORT(A)

O(n)	$n = A.length$
O(n)	let $B[0\,..\,n-1]$ be a new array
O(n)	**for** $i = 0$ **to** $n-1$
O(n)	make $B[i]$ an empty list
O(n)	**for** $i = 1$ **to** n
O(n)	insert $A[i]$ into list $B[\lfloor nA[i] \rfloor]$
O(n)	**for** $i = 0$ **to** $n-1$
$O(n_i^2)$	sort list $B[i]$ with insertion sort
O(n)	concatenate the lists $B[0], B[1], \ldots, B[n-1]$ together in order

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).**

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