Algoritma Analizi

Ders 7 – Bölüm 2: Lineer Zamanlı Sıralama

Doç. Dr. Mehmet Dinçer Erbaş
Bolu Abant İzzet Baysal Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

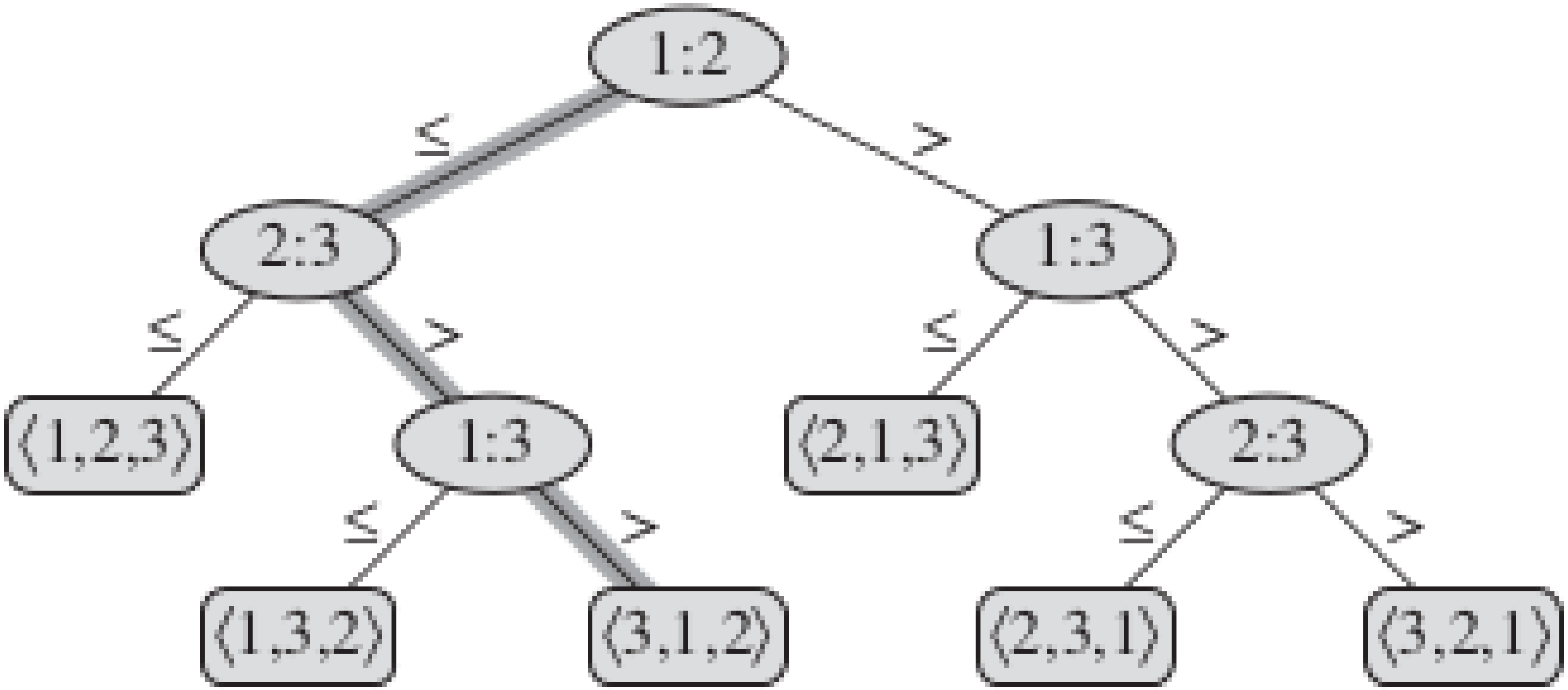
Lineer zamanlı sıralama

- Bu aşamaya kadar birçok farklı sıralama algoritması gördük.
- Bu algoritmaların bazıları (merge sort ve heap sort) O(n lg n) en kötü durumda sıralayabiliyor.
- Quick sort ortalamada aynı değere ulaşabiliyor.
- Ayrıca belirtilen bu algoritmalar için belli bir dizi oluşturarak, algoritmaların Ω(n lg n) sürede çalışmalarını sağlayabiliriz.
- Bu algoritmaların ortak bir özelliği var:
  - Sıralama yaparken verilen girdideki elemanları karşılaştırıyorlar.
  - Bu sebeple bu algoritmalara karşılaştırmalı sıralama algoritmaları diyoruz.
- Bu bölümde öncelikle karşılaştıran sıralama algoritmaları için Ω(n lg n) olduğunu göstereceğiz.
- Daha sonra verilen girdinin belli özelliklerinden yararlanarak karşılaştırma yapmadan sıralayan algoritmalar göreceğiz.
  - Yukarıda belirtilen alt sınır bu algoritmalar için geçerli olmayacak.

Karşılaştırmalı sıralama

- Karşılaştırma algoritmaları girdi olarak verilen dizilerin elemanları arasında karşılaştırma yapar.
- Verilen dizi $<a_1, a_2, \ldots, a_n>$ ise, herhangi iki eleman, a_i, a_j için
  - Aşağıdaki durumlardan birini test ederiz
    - $a_i < a_j$, $a_i \leq a_j$, $a_i = a_j$, $a_i \geq a_j$ veya $a_i > a_j$
  - Bu sayede elemanların sıralamasını belirleriz.
- Karşılaştırmalı sıralama algoritmalarını çalışma mantığı karar ağaçları ile incelenebilir.
  - Bir sonraki slaytta insertion sort algoritmasının üç elemanlı bir diziyi sıralarken yaptığı işlemler gösteriliyor.

Karşılaştırmalı sıralama



Karşılaştırmalı sıralama

- Algoritma her türlü sıralama ile sonuçlanabilir. Bu sebeple n! permutasyonun her biri ağacın kökünden yapraklarına hareket sonucu oluşabilir.
- Ağacın kökünden herhangi bir yaprağına olabilecek en uzun yol en kötü durumda kaç tane karşılaştırma yapılacağını gösteriyor.
- Buna bağlı olarak en kötü durumda yapılacak karşılaştırma sayısı ağacın yüksekliğine eşittir.
- Ağacın yüksekliği için bir alt sınır bulursak bu algoritmanın en kötü çalışma süresi için bir alt sınır olacaktır.

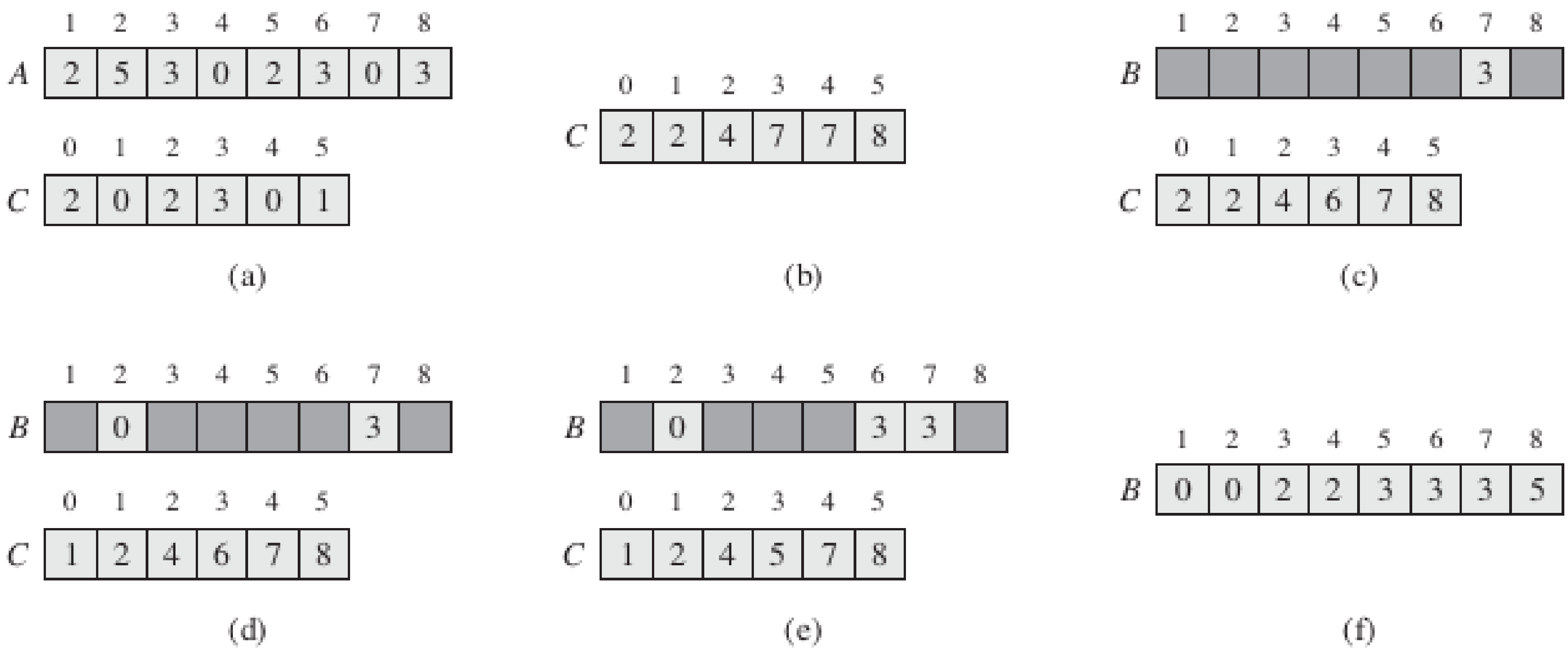
Karşılaştırmalı sıralama

- Teori: Karşılaştırmalı sıralama algoritmaları en kötü durumda $\Omega(n \lg n)$ sürede çalışır.
  - İspat için karar ağacının yüksekliğini hesaplayacağız.
  - Ağacın yüksekliği h, yaprak sayısı l olsun.
  - Her permütasyon en azından bir kere yapraklarda oluşması gerekir, bu sebeple $n! \leq l$.
  - Yüksekliği h olan bir ikili ağaç en fazla 2^h yaprağa sahip olabilir.
  - $n! \leq l \leq 2^h$
  - $h \geq \lg(n!)$
  - $= \Omega(n \lg n)$
- Heapsort ve mergesort algoritmaları optimal sıralama algoritmalarıdır.
  - $O(n \lg n)$ ve $\Omega(n \lg n)$ oldukları için bu durum gözlemlenebilir.

Sayarak Sıralama

- Sayarak Sıralama (İng: Counting Sort) 0 ile k tam sayısı arasındaki n elemanı sıralamak için kullanılır. $k = O(n)$ olduğunda Sayarak Sıralama $\Theta(n)$ sürede sıralama yapar.
- Sayarak Sıralama şu şekilde çalışır:
  - Dizideki her bir x elemanı için, x sayısından küçük olan elemanların sayılarını hesaplar.
    - Bu bilgi kullanılarak x sayısı sonuç dizinde doğru pozisyona yerleştirilebilir.
    - Örneğin x sayısından küçük 5 eleman var ise x sayısını 6 numaralı pozisyona yerleştirebiliriz.

Sayarak Sıralama



Sayarak Sıralama

```
COUNTING-SORT(A, B, k)
let C[0..k] be a new array
for i = 0 to k
    C[i] = 0
for j = 1 to A.length
    C[A[j]] = C[A[j]] + 1
// C[i] now contains the number of elements equal to i.
for i = 1 to k
    C[i] = C[i] + C[i − 1]
// C[i] now contains the number of elements less than or equal to i.
for j = A.length downto 1
    B[C[A[j]]] = A[j]
    C[A[j]] = C[A[j]] − 1
```

Sayarak Sıralama

- Sayarak Sıralama algoritmasının analizi
  - 1-2 satırlarındaki for döngüsü $\Theta(k)$ süre alıyor.

    4-5 satırlarındaki for döngüsü $\Theta(n)$ süre alıyor.
  - 7-8 satırlarındaki for döngüsü $\Theta(k)$ süre alıyor.
  - 10-12 satırlarındaki for döngüsü $\Theta(n)$ süre alıyor.
  - Sonuç olarak algoritma $\Theta(n + k)$ süre alıyor.
  - $k = O(n)$ olduğunda algoritma $\Theta(n)$ sürede çalışır.
- Sayarak Sıralama karşılaştırmalı sıralama algoritmalarının $\Omega(n \lg n)$ sınırını geçebiliyor.
- Elemanlar arası karşılaştırma yapılmıyor.
- Bunun yerine elemanların değerleri kullanılarak yeni bir dizi içerisinde indeksleme yapılıyor.

Sayarak Sıralama

- Sayarak Sıralama‘un bir diğer önemli özelliği de stabil olmasıdır.
  - Yani aynı değere sahip iki eleman dizide hangi sırayla bulunuyorsa sıralandıklarında aynı sırayla listelenirler.
- Normalde stabil olma özelliği anahtar değerler ile ek bazı değerlerin taşınması durumunda yararlıdır.
- Ayrıca Sayarak Sıralama stabil olduğu için bu algoritma birazdan göreceğimiz Tabana Göre Sıralama algoritmasının içerisinde kullanılabilir.

Tabana Göre Sıralama

- Tabana Göre Sıralama (İng: Radix Sort) algoritması sıralanacak d basamağa ayırır.
- En önemsiz basamaktan başlanarak bütün basamaklara göre sayılar sıralanır.
- Bu sebeple sayılar d basamaktan oluşuyorsa toplam d kere sıralama yapılır.
- Algoritmanın doğru çalışabilmesi için sıralama yapılırken kullanılan alt operasyonun stabil olması gerekmektedir.

Tabana Göre Sıralama

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

Tabana Göre Sıralama

```
RADIX-SORT(A, d)
for i = 1 to d
    use a stable sort to sort array A on digit i
```

Tabana Göre Sıralama

- Tabana Göre Sıralama algoritmasının analizi
  - Eğer her basamak 0 ile k – 1 arası değer alıyor ve k sayısı büyük değilse Sayarak Sıralama kullanarak basamaklar sıralanabilir.
  - Bu durumda her basamak O(k) sürede sıralanacaktır.
  - Toplam sıralanacak eleman sayısı n ise bir basamak için sıralama O(n+k) süre alır.
  - Basamak sayısı d ise toplam süre O(d(n+k)) olarak hesaplanır.
  - n tane b adet basamağa sahip sayılar verildiğinde, herhangi $r \leq b$ pozitif tam sayı değeri için Tabana Göre Sıralama bu sayıları $\Theta((b/r)(n+2^r))$ sürede sıralar.
  - Verilen n ve b değerleri için yukarıdaki ifadeyi en küçük yapan $r \leq b$ değeri seçmeliyiz.

Tabana Göre Sıralama

- Tabana Göre Sıralama algoritmasının analizi
  - b < $\lfloor lg\, n \rfloor$ ise herhangi r ≤ b değeri için $(n+2^r) = \Theta(n)$'dir. Öyle ise r=b seçerek $(b/b)(n+2^b) = \Theta(n)$ çalışma süresine erişebiliriz.
  - b ≥ $\lceil lg\, n \rceil$ ise r = $\lfloor lg\, n \rfloor$ seçerek belli bir sabit dahilinde en iyi çalışma süresine erişebiliriz.
  - Sonuç olarak girdi olarak verilen sayılar belli özelliklere sahipse Tabana Göre Sıralama Θ(n) çalışma süresine erişebilir.