Algoritma Analizi

Ders 7 – Bölüm 1: Sıralama Algoritmaları

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

Hızlı sıralama

- Hızlı sıralama algoritması en kötü çalışma süresi $O(n^2)$ olan bir sıralama algoritmasıdır.
- En kötü çalışma süresi yavaş olmasına rağmen, birçok uygulamada en verimli sıralama algoritması olarak kullanılır. Bunun nedeni:
  - Ortalamaya bakıldığında, ortalama çalışma süresi $\Theta(n \lg n)$ olarak hesaplanır.
  - $\Theta(n \lg n)$ çalışma süresindeki gizli sabitler oldukça küçüktür.
  - Ayrıca Yığın sıralama ve insertion sort için olduğu gibi yerinde sıralama yapar.
- Bu bölümde öncelikle Hızlı sıralama algoritmasını tanımlayacağız.
- Daha sonra algoritmanın çalışma süresini analiz edeceğiz.

Hızlı sıralama

- Hızlı sıralama, merge sort gibi böl ve yönet yaklaşımı ile çalışır.
- Yapılan işlemler, sıralanmamış A[1..r] dizisi için şunlardır:
  - Böl: Verilen A[p..r] dizisini şu şekide parçala (veya yeniden düzenle)
    - Verilen dizi iki farklı altdiziye ayrılır. Bunlar A[p..q-1] ve A[q+1..r].
    - A[p..q-1] altdizisindeki elemanların değeri A[q]‘den azdır veya eşittir.
    - A[q+1..r] altdisizindeki elemanların değeri A[q]‘den fazladır veya eşittir.
  - Fethet: Oluşturulan iki altdiziyi yinelemeli Hızlı sıralama çağrısı ile sırala.
  - Birleştir: Altdiziler yerinde sıralandığı için son birleşme işlemine gerek yoktur. Dizinin tamamı sıralanmıştır.

Hızlı sıralama

- Hızlı sıralama algoritması aşağıdaki şekilde uygulanabilir:

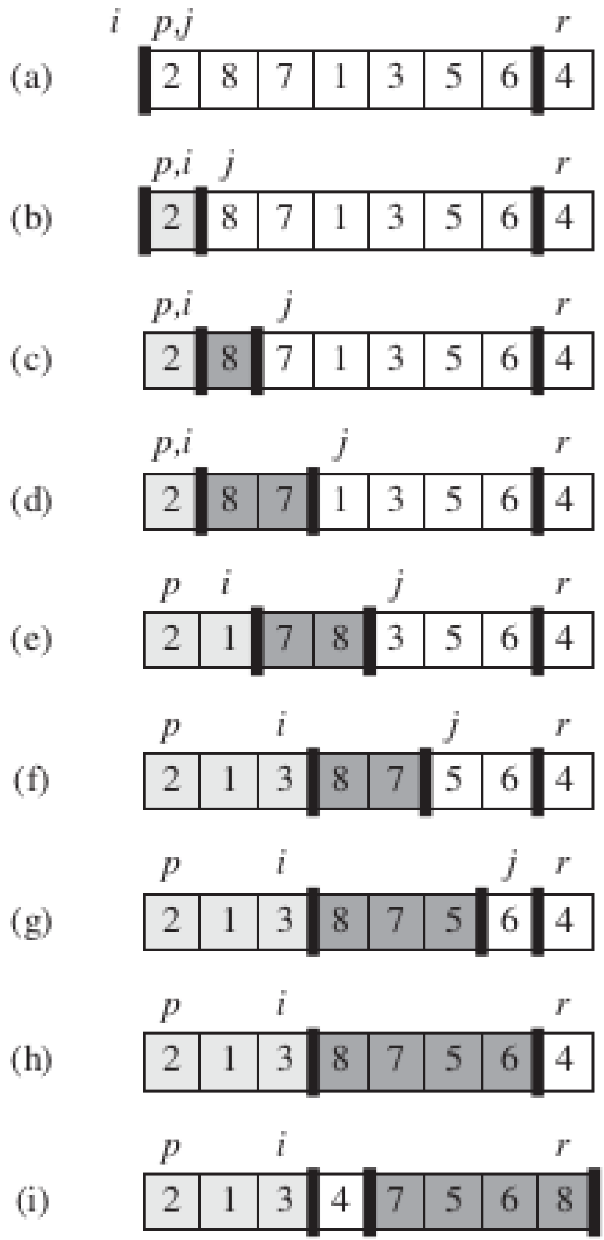
```
QUICKSORT(A, p, r)
    if p < r
        q = PARTITION(A, p, r)
        QUICKSORT(A, p, q - 1)
        QUICKSORT(A, q + 1, r)
```

Hızlı sıralama

- Verilen diziyi parçalama işlemi şu şekilde yapılır:

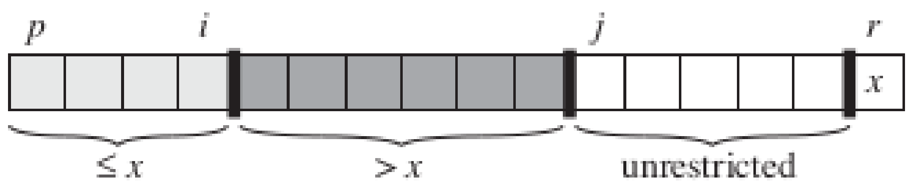
```
PARTITION(A, p, r)
x = A[r]
i = p − 1
for j = p to r − 1
    if A[j] ≤ x
        i = i + 1
        exchange A[i] with A[j]
exchange A[i + 1] with A[r]
return i + 1
```

Hızlı sıralama



Hızlı sıralama

- Döngü sabiti:
  - 3-6 satırlarındaki döngünün her çalışmasından önce, her k indeks değeri için aşağıdaki durumlardan biri söz konusudur:
    - Eğer p ≤ k ≤ i ise A[k] ≤ x
    - Eğer i+1 ≤ k ≤ j – 1 ise A[k] > x
    - Eğer k = r ise, A[k] = x



Hızlı sıralama

- Döngü sabiti:
  - Başlangıç: Döngünün ilk çalışmasından önce i = p – 1 ve j = p. Bu durumda p ile i arasında ve i + 1 ile j – 1 arasında bir değer bulunmuyor. Öyleyse döngü sabiti doğru.
  - Sürdürme: Bir sonraki şekilde görüldüğü üzere her döngü çalışmasında iki farklı durum söz konusudur:
    - A[j] > x ise, j değeri bir artırılır. Bu durumda A[j – 1] için 2 numaralı durum söz konusudur. Diğer indekslerin durumu etkilenmemiştir.
    - A[j] ≤ x ise, i değeri bir artırılır, A[i] ile A[j]‘nin yeri değiştirilir ve j değeri bir artırılır. Sonuç olarak yer değiştirmeden dolayı A[i]<x ise 1 numaralı durum söz konusudur. Ayrıca A[j – 1] > x durumu devam etmektedir çünkü değiştirdiğimiz değer döngü sabitine göre x değerinden büyüktür.

Hızlı sıralama

- Döngü sabiti:
  - Sonlanma: Döngü sonlandığında j = r. Bu durumda dizideki her eleman döngü sabitinde belirtilen üç durumdan birindedir. Böylece diziyi üç ayrı parçaya ayırdık: değeri x‘den küçük veya x‘e eşit olanlar, değeri x‘den büyük olanlar ve değeri x olan eleman.
- Algortmanın son iki satırı pivot olarak kullanılan elemanı doğru yerine yerleştirmektedir. Böylece parçalama işleminde belirtilen dizi oluşturulmuştur.
- Verilen A[p..r] dizisi için PARTITION fonksiyonu $\Theta(n)$ sürede çalışmaktadır, şöyle ki $n = r – p + 1$.

Hızlı sıralama

- Hızlı sıralama algoritmasının performans analizi
  - Hızlı sıralama algoritmasının çalışma hızı parçalama fonksiyonun oluşturduğu parçaların dengeli veya dengesiz olmasına bağlıdır.
    - Oluşan parçalar benzer büyüklükte ise algoritma asimtotik olarak merge sort kadar hızlı çalışabilmektedir.
    - Oluşan parçalar dengesiz şekilde dağılmış ise algoritma asimtotik olarak insertion sort kadar yavaş çalışabilmektedir.
  - Ön kötü parçalama durumu
    - Bu durumda parçalama mevcut problemi n – 1 elemanlı bir altproblem ve 0 elemanlı bir altprobleme ayırmaktadır.
    - Bu durumun her yinelemeli çağrıda oluştuğunu düşünelim
      - Parçalama $\Theta(n)$ kadar zaman almakta.
      - $T(n) = T(n-1) + \Theta(0) + \Theta(n)$

        $= T(n-1) + \Theta(n)$

        $= \Theta(n^2)$
    - Bu durum ne zaman oluşur?

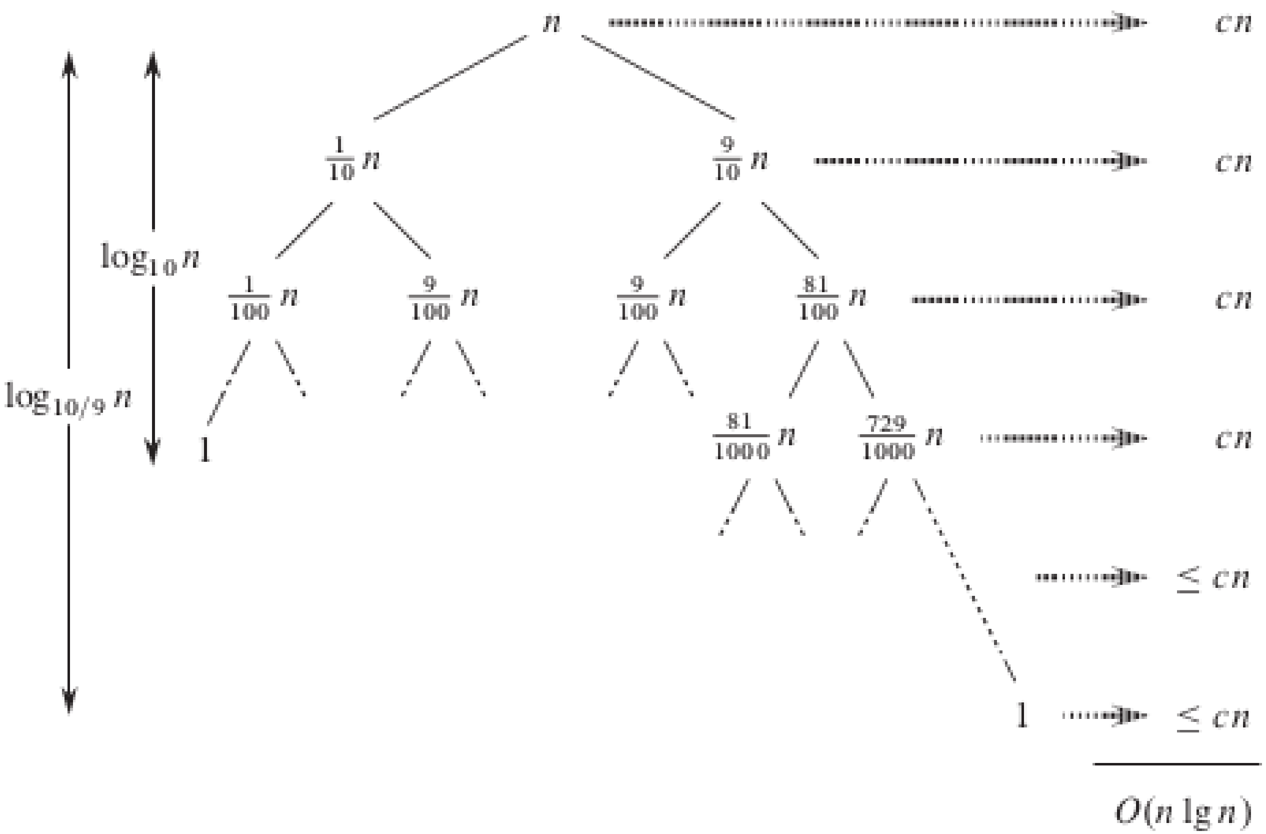
Hızlı sıralama

- Hızlı sıralama algoritmasının performans analizi:
  - En iyi çalışma zamanı:
    - En iyi durumda parçalıyıcı problemi yaklaşık olarak iki eşit parçaya böler
      - Parçalardan biri $\lfloor n/2 \rfloor$ diğeri ise $\lceil n/2 \rceil - 1$ büyüklüğünde olacaktır.
      - Bu durumda her iki problemin büyüklüğüde n/2 ile sınırlıdır.
    - $T(n) \leq 2T(n/2) + \Theta(n)$
    - Master metodu kullanırsak
      - $O(n \lg n)$

Hızlı sıralama

- Hızlı sıralama algoritmasının performans analizi
  - Algoritmanın ortalama çalışma süresi en iyi çalışma süresine oldukça yakındır.
  - Bunu şu örnek üzerinden görebiliriz:
    - Diyelim ki parçalama fonksiyonu her sefer problemi 9-1 olarak ayırsın.
    - $T(n) \leq T(9n/10) + T(n/10) + cn$

Hızlı sıralama



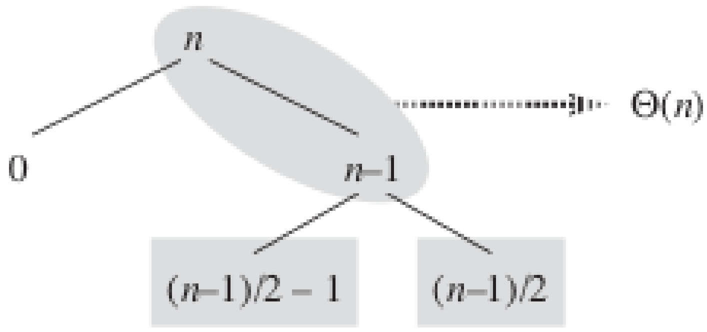
Hızlı sıralama

- Hızlı sıralama algoritmasının perfomans analizi
  - Örnek: Diyelim ki parçalama fonksiyonu her sefer problemi 9-1 olarak ayırsın.
  - $T(n) \leq T(9n/10) + T(n/10) + cn$
    - Sınır durumu $\log_{10} n = \Theta(\lg n)$ derinlikte oluşuncaya kadar ağacın her seviyesinde cn kadar zaman harcanıyor.
    - Bu durumdan sonra her seviye en fazla cn kadar zaman harcanıyor.
    - $\log_{10/9} n = \Theta(\lg n)$ derinlikte bütün yinelemeli çağrılar sonlanıyor.
      - Bu durumda Hızlı sıralama çalışma süresi $O(n \lg n)$.
  - Öyleyse 9‘a 1 gibi dengesiz bir ayırma durumunda dahi Hızlı sıralama $O(n \lg n)$ sürede çalışıyor.
  - 99‘a 1 de ayırsak aynı durum söz konusu.
  - Sabit oranda ayrım yapıldığında ağacın boyu $O(\lg n)$ ve her seviyede harcanan zaman $O(n)$ oluyor.
  - Sonuç olarak her seviyede sabit oranda ayrım yapıldığında Hızlı Sıralama çalışma süres $\Theta(n \lg n)$.

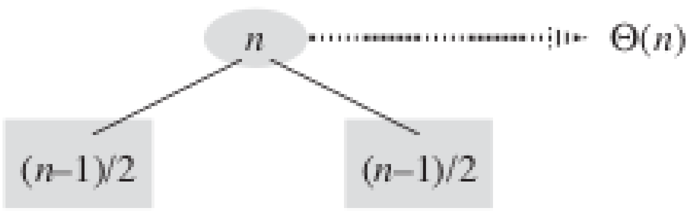
Hızlı Sıralama

- Hızlı Sıralama algoritmasının perfomans analizi
  - Görüldüğü üzere Hızlı Sıralama‘un çalışma süresi parçalama işlemi sırasında pivot olarak seçilen elemanın sıralamasına göre değişim gösteriyor.
  - Her türlü sıralamanın eşit olasılıkla ortaya çıkabiliceğini varsayarsak:
    - Bu durumda her parçalamada farklı oranlarda alt parçalar oluşacaktır.
      - Bazı parçalamalar benzer büyüklükte iki parça oluştururken bazı parçalamalarda dengesiz büyüklükte parçalar oluşacaktır.
    - Farklı parçalama tiplerinin Hızlı Sıralama algoritmasının çalışma süresine etkisini anlayabilmek için aşağıdaki gibi bir basitleştirme yapalım:
      - Aşamalarda birbirini takip eder şekilde kötü (dengesiz) ve iyi (dengeli) parçalama yapılıyor.
      - İyi parçalama tam dengeli iki parça oluştururken, kötü parçalama en dengesiz şekilde ayırıyor.

Hızlı Sıralama



(a)



(b)

Hızlı Sıralama

- Hızlı Sıralama algoritmasının performans analizi
  - Önceki slaytta görülen durumda sonuç olarak dengeli iki parça oluşuyor.
  - Öyleyse Hızlı Sıralama iyi ve kötü ayrımlar yaptığında halen $\Theta(n \lg n)$ sürede çalışıyor.
    - Ancak şekilde görülen kötü ayrımlar nedeniyle daha yüksek gizli sabit değere sahip oluyor ve biraz daha fazla süre alıyor.

Hızlı Sıralama

- Rastgele hale geitirilmiş Hızlı Sıralama (Randomized Quick Sort)
  - Hızlı Sıralama önceden neredeyse sıralanmış bir diziyi sıralarken, pivot seçimi nedeniyle yavaş çalışabilir.
    - Pivot olarak hep en sondaki eleman seçilir ve bu eleman genellikle dizinin en yüksek değere sahip elemanı olursa, parçalama işlemi sonucu dengesiz iki parça oluşur.
  - Bu tür durumları engellemek için pivot elemanı rastgele seçilebilir.
    - Böylece devamlı en yüksek değere sahip elemanı seçme ihtimali düşer ve görece dengeli parçalama işlemi yapılabilir.

Hızlı Sıralama

- Rastgele hale geitirilmiş Hızlı Sıralama (Randomized Quick Sort)

```
RANDOMIZED-PARTITION(A, p, r)
    i = RANDOM(p, r)
    exchange A[r] with A[i]
    return PARTITION(A, p, r)
```

The new quicksort calls RANDOMIZED-PARTITION in place of PARTITION:

```
RANDOMIZED-QUICKSORT(A, p, r)
    if p < r
        q = RANDOMIZED-PARTITION(A, p, r)
        RANDOMIZED-QUICKSORT(A, p, q - 1)
        RANDOMIZED-QUICKSORT(A, q + 1, r)
```