Tayfun ADAKOĞLU

Quicksort Algoritması ve Analizi

İçindekiler

Giriş

İngiliz Bilgisayar bilimcisi, Tony Hoare

Quicksort bir dizideki elemanları sıralamak için kullanılan etkili bir sıralama algoritmasıdır, bazen "partition-exchange sort" algoritması olarak da tabir edilir. Tony Hoare tarafından 1959'da geliştirilmiştir ve 1961'de çalışmasında yayınlanmıştır ve hala günümüzde sıralama için kullanılan bir algoritmadır. Düzgün gerçekleştirildiğinde baş rakipleri "merge sort" ve "heap sort" algoritmalarından 2 veya 3 kat daha hızlı çalışabilir.

Günümüzde en çok kullanılan en hızlı sıralama algoritmasıdır. Ancak füze fırlatma veya hastanelerdeki kalp ritmini ölçen cihazlar gibi tepki süresi önemli yerlerde quick sort yerine merge sort tercih edilir. Merge sort ile quick sort'un hızları hemen hemen aynıdır. Ancak merge sort algoritmasının "response time"' yani tepki süresi daha iyidir, yani merge sort "n" değişkeni giriş sayısı olmak üzere O(*n* log *n*) sürede en kötü durumda cevap verirken quick sort ise en kötü şartlarda, yani verilen giriş verileri bayağı bir sırasız, ise n^2 sürede tepki verir. Bu açıdan bazen yerini merge sort algoritmasına bırakmak zorunda kalabilir. Quick sort hafızayı da diğer algoritmalara göre çok daha az kullanır bu yüzden tercih sebebidir.

Amaç

Bu projenin amacı yüksek performans gerektiren sayısal ve bilimsel hesaplamalarda oldukça etkili çalışan yüksek seviye Julia programlama

dilinin Quicksort algoritması üzerindeki başarımını incelemektir. C# programlama dili ile yazılmış bir Quicksort algoritması ile de karşılaştırma işlemi (benchmarking) ile sonuçları ispatlamaktır.

"Quicksort.jl" Julia derleyicisi kod dosyası
"Quicksort.cs" Visual C# derleyicisi kod dosyası
"Karşılaştırma(Benchmarking).mp4" Video görseli ile iki dilin aynı algoritma üzerinde hız performansının bir karşılaştırılmasının yapılması ve sunulması.

Algoritmanın Tarihi Gelişimi

Quicksort 1959'da Tony Hoare tarafından kendisi Sovyetler Birliği'nde Moskova Devlet Üniversitesi için ziyaretçi öğrenci olduğu zamanlarda geliştirilmiştir. Hoare o dönemde Ulusal Fizik Labaratuvarı için bilgisayarlı çeviri sistemleri üzerine bir peojede çalışıyordu. Rusça-İngilizce çeviri işleminin bir parçası olarak, alfabetik olarak sıralanmış bir sözlük üzerinde çevireceği Rusça cümlelerin sözcüklerini aramaya başlamadan önce bu sözcüklerin sıralanmasının hız açısından daha uygun olacağını düşünmüştür. Ortaya bu fikrini attıktan sonra insertion sort algoritmasını denemiştir ancak bu algoritmayı Tony yavaş bulmuştur daha sonrasında ise kendi geliştiştirdiği Quicksort olarak tabir edeceği algoritmasını kullanmıştır. İngiltere'ye döndüğünde ise kendisinden yeni işinin bir parçası olarak Shellsort algoritması ile kod yazması istenmiştir. Hoare patronuna daha hızlı bir algoritma bildiğini söylemiştir hatta patronu bunun üzerine bir miktar İngiliz altını (sixpence) için kendisiyle daha hızlı bir algoritma olmadığı üzerine bahse girmiştir. Patronu sonunda bahsi kaybettiğini kabul etmiştir. Hoare daha sonra kodları Bilgisayar Derneği (Association for Computing Machinery, bilgisayar bilimleri için en eski kuruluş) için yazdığı bir makalede paylaşmıştır.
Quicksort bütün bu olaylarla vesilesiyle yaygınlık kazanmıştır. Mesela Unix varsayılan sıralama kütüphanesi fonksiyonu olarak Quicksort

kullanır. Java ve C standart kütüphanesi fonksiyonları da quicksort,qsort isimlerini kullanılır.

Algoritma

Quicksort bir böl ve fethet algoritmasıdır. Öncelikle quicksort geniş bir diziyi iki daha küçük alt diziye böler; düşük seviye elemanlar ve yüksek seviye elemanlar olmak üzere. Quicksort daha sonrasında "recursive" bir şekilde alt dizilerin her birini ayrı ayrı sıralar. Her bir alt dizi tekrar bölünür aynı işlemler tekrar yapılır sonuçta bulunan sonuçlar birleştirilir.

Aşamalar:

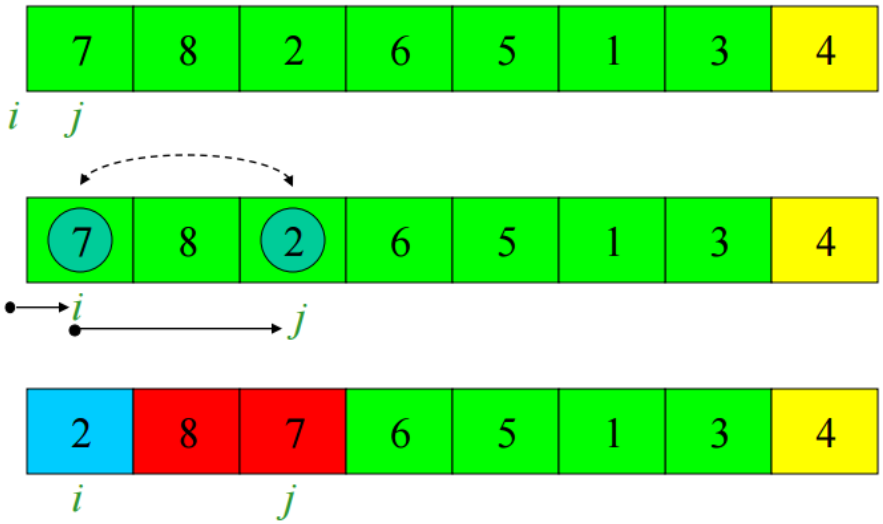
1. **Diziden pivot denecek herhangi bir eleman seç.**
2. **Bölümleme yap, yani dizideki elemanları öyle bir sırala ki pivotun solundakiler pivottan küçük veya eşit sağındakiler ise pivottan eşit veya büyük olsun. Pivota eşit olanlar her iki tarafta da yer alabilir. Sonuçta pivot final pozisyonundadır.**
3. **Yukarıdaki aşamaları "öz yinelemeli olarak" pivottan küçük veya eşit olanlar ve pivottan büyük veya eşit olanlar alt dizileri ayrı ayrı uygula.**

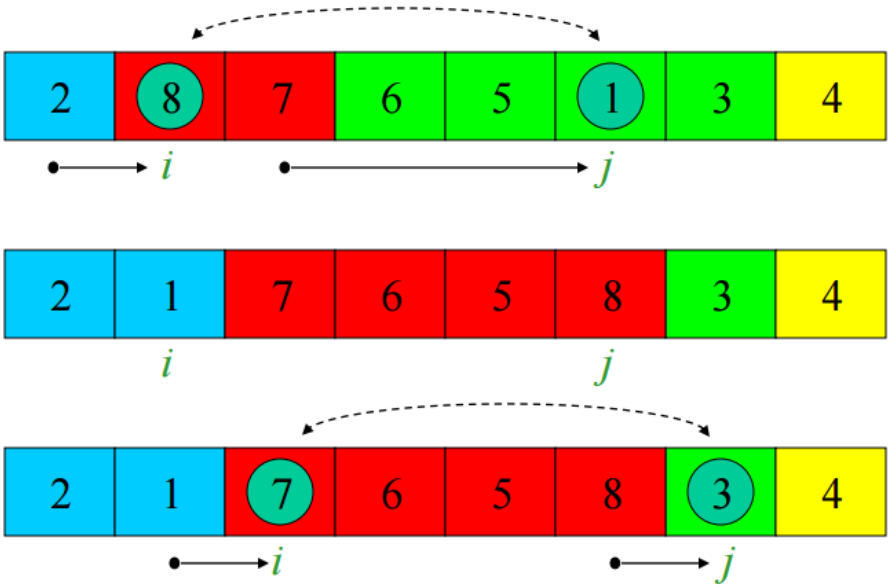
Özyinelemenin temel durumu dizinin bir veya sıfır uzunluğunda olmasıdır, yani hiçbir zaman sıralama yapılmaması durumu. Pivot seçimi ve bölümleme adımları farkı şekillerde yapılabilir, farkı uygulamaların seçilmesi algoritmanın performansını çokça fazla etkilemektedir.

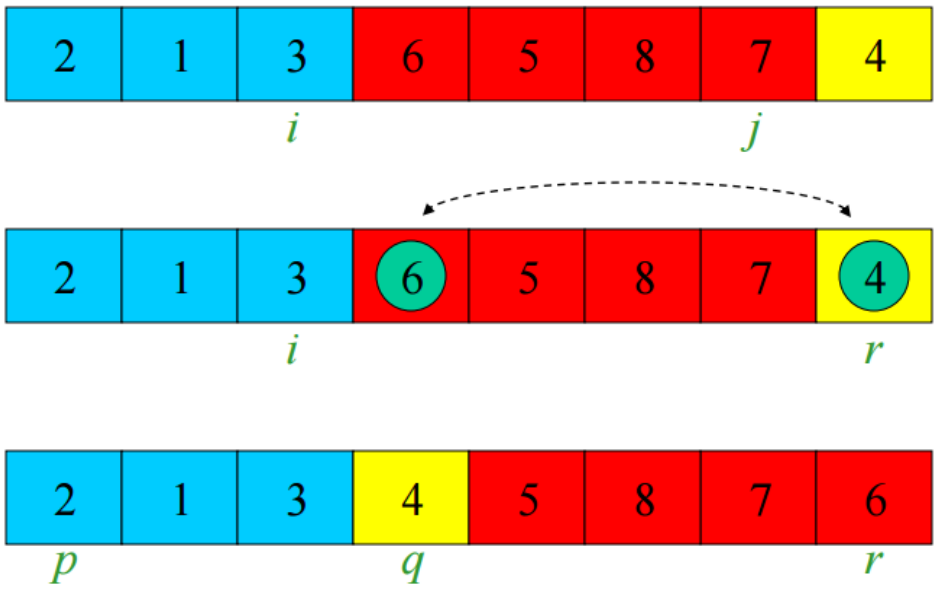
Lomuto partition scheme

Pivot olarak dizinin sonundaki eleman seçilir. İ ve j ilk dizi elemanının indexlerininin değerini başlangıçta alır. Algoritma pivota eşit veya ondan küçük her eleman bulasıya kadar "i" index değeri arttırılır, pivota eşit veya ondan daha büyük bir eleman bulasıya kadar ise "j" değeri arttırılır. Daha sonra i ve j nin gösterdiği elemanlar birbiriyle yer değiştirilir. En sonunda da j dizinin sonuna kadar ulaşır ve pivottan büyük veya eşit değer olarak pivotu seçmiş olur. Daha sonra bu değer i'nin bulduğu eleman ile yer

değiştirilir





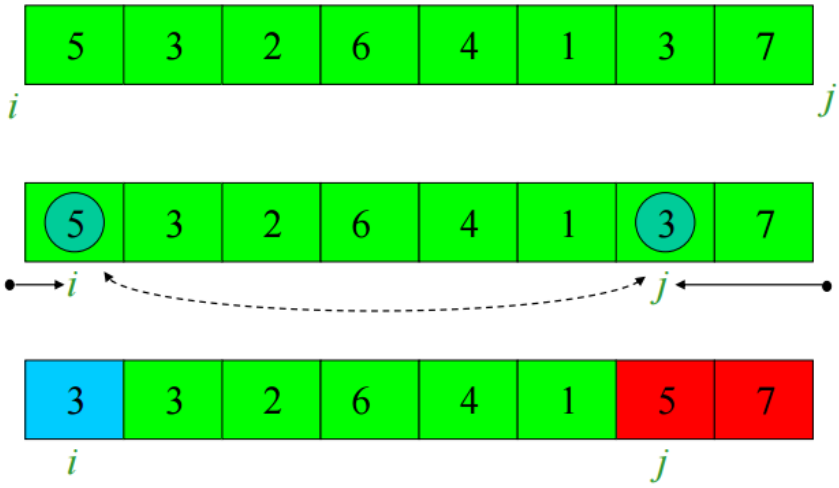


Hoare partition scheme, The Original

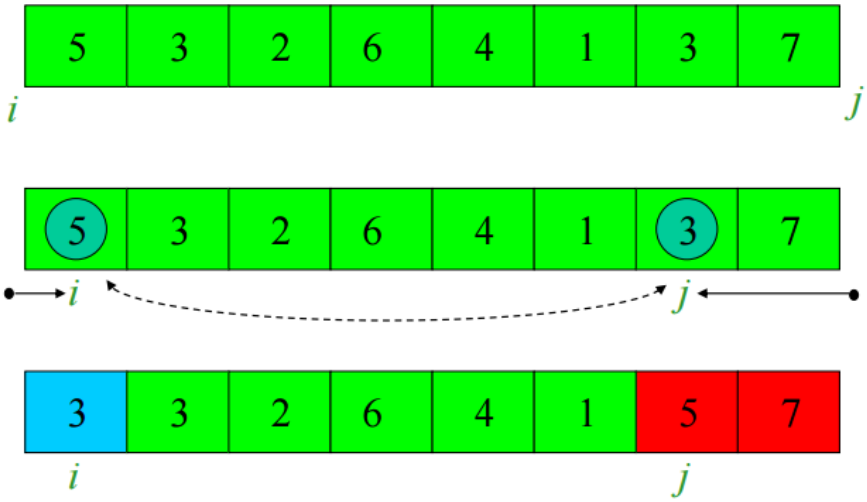
Pivot olarak ilk eleman seçilir. Hoare'nin bölümleme metodunun i ve j indexleri ise dizinin başını ve dizinin sonunu gösterir ve daha sonra birbirine doğru ilerletilir, bir terslik durumu bulunduğunda ,yani öyle ki iki eleman bir tanesi pivottan büyük veya eşit diğeri pivottan küçük veya eşit ,bu elemanlar elemanlar yer değiştirilir. İ ve j indexleri buluştuğunda algoritma durur son indexler pivot olmuş olur yani ortanca eleman. Tam bu noktadan pivot çıkarılıp iki tane ayrı ayrı alt dizi oluşturulur ve her bir alt dizi için işlemler özyinelemeli olarak yapılır sonuçta pivotlar birleştirilir ve sıralama tamamlanmış olur.

Hoare'nin metodu Lomuto'nun bölümleme metodundan daha etkilidir çünkü ortalamada 3 kat daha az eleman değiştirmesi yapar .Ayrıca Lo (lower elements), P (Pivot), Hi (higher elements) indexleri olmak üzere, Lomuto'nun bölümleme metodu [lo..p) and (p..hi] alt dizileri üzerinde çalışır yani pivota bölümeleme yaptıktan sonra diğer alt bölümler içinde pivota ihtiyaç yoktur fakat Hoare'ninkinde ise [lo..p] and (p..hi] olarak ilk bölüme dahil edilir. Ayrıca Hoare'nin bölümleme metodu da Lomuto'nun ki de dizi zaten sıralı ise $O(n^2)$ sürede tüm elemanlara bakarak işlemini tamamlar.

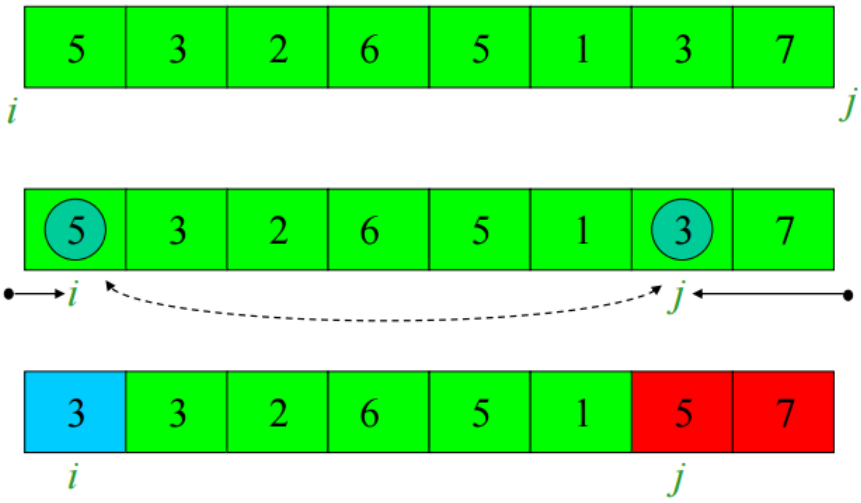
Hoare's Algorithm: Example 1 (pivot = 5)



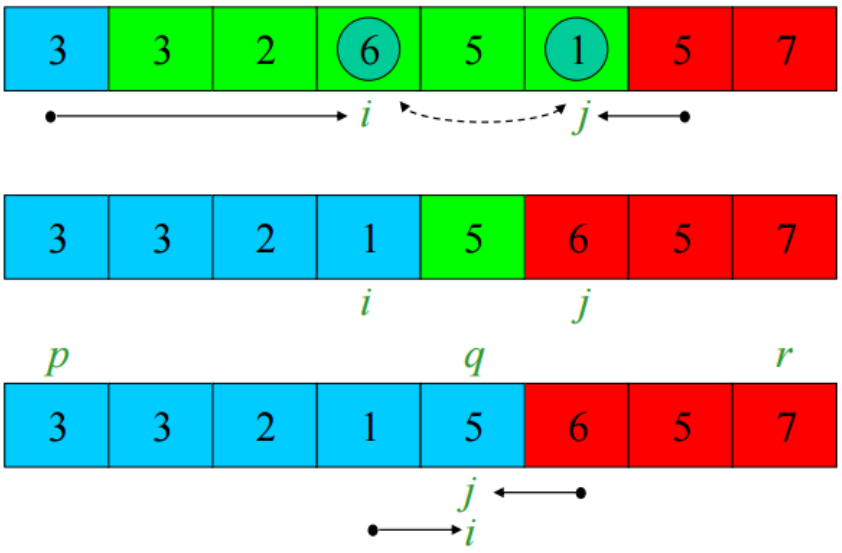
Hoare's Algorithm: Example 1 (pivot = 5)



Hoare's Algorithm: Example 2 (pivot = 5)



Hoare's Algorithm: Example 2 (pivot = 5)



Termination: $i = j = 5$

Hoare’s original quick sort algoritm

Burada lo ve hi, yukadaki tanımlarda anlattığım sırasıyla i ve j’ye tekabül etmektedir. A ise dizimizdir.

```
algorithm quicksort(A, lo, hi) is // sıralama algoritması
  if lo < hi then
    p := partition(A, lo, hi) // ilk bölümlemeyi yapıp pivot’un konumunu bul
    quicksort(A, lo, p) // pivot ve öncesini algoritmaya tabi tutuyor
    quicksort(A, p + 1, hi) // pivot ve sonrasını algoritmaya tabi tutuyor

algorithm partition(A, lo, hi) is // bölümlemeyi yapan algoritma
  pivot := A[lo]
  i := lo - 1
  j := hi + 1
  loop forever
    do:
      i := i + 1
    while A[i] < pivot do

    do:
      j := j - 1
    while A[j] > pivot do

    if i >= j then
      return j

    swap A[i] with A[j]
```

Dizideki eleman sayısı 10'dan küçükse özyinelemeli olmayan insertion sort daha uygundur, çünkü daha az eleman değiştirmesi yapar.

Quicksort böl ve fethet anlaşımıyla sorunu çözdüğü için çok işlemcili sistemlerden yararlanılmayı mümkün kılar. Farklı alt diziler farklı çekirdeklerde sıralanıp dönen sonuçlar ile, işlemler çok hızlı gerçekleştirilebilir.

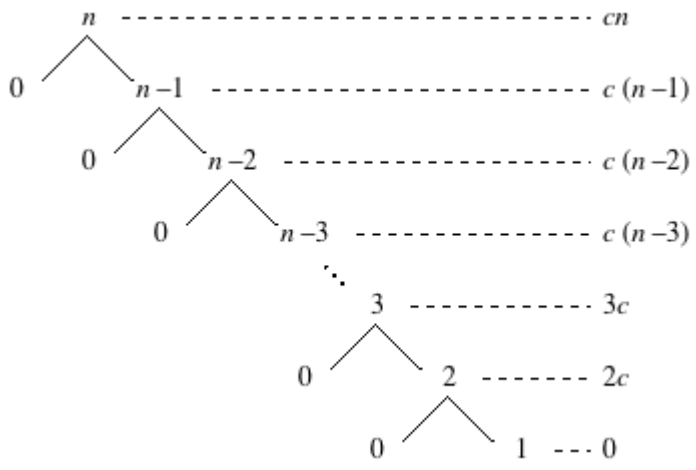
Resmi Analiz

En Kötü Durum Analizi

En dengesiz durum pivotun diziyi boyutları 0 ve n-1 olan iki alt diziye bölmesi halindedir. Yani pivot her seferinde en dizideki en küçük ya da en büyük eleman çıkarsa bu durum gerçekleşir, pivotun solundaki veya sağındaki dizi boş olmak durumunda kalır. İlk dizideki pivotu bulmak için harcadığımız zaman n ile , ikinci dizide n-1 üçüncü dizide n-3.. en sonda elimizde tek eleman kalır o an işlem yapmamız gerek kalmamıştır yani harcanan süre 0'dır. Toplam harcanan süre için hepsini toplarsak n *(n+1)/2 yapar bu da n^2 katsayısı artış oranında zaman alır;

$$\sum_{i=0}^{n}(n-i) = O(n^2)$$

Bu durumda Quicksort $O(n^2)$ zaman karmaşıklığında işlemini en kötü durumda tamamlar.



Ya da formül üzerinden şeklinde de aynı sonucu bulabiliriz; Omega(n) tam olarak n zaman karmaşıklığında (yani n eleman olduğunda karşılaştırma için gereken zamanı ifade eder) işlem süresi gerektiği anlamına gelir. İçindeki değer giriş sayısıdır. Omega(1) bir tane eleman olduğunda yapılması gereken temel işlemin(karşılaştırma) aldığı zamandır. Tek giriş olduğunda dizide karşılaştırmaya gerek kalmadığından gereken zaman 0'dır. Omega(1) = 0, Omega(n)=n, Omega(n-1) = n-1 … şeklinde aşağıdaki böl ve fethet mantığına göre kurulmuş formülden de sonuç bulunabilir.

ANALYSIS OF QUICK SORT
WORST CASE

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

$$T(n) = T(n-2) + \Theta(n-1) + \Theta(n)$$

$$T(n) = T(n-3) + \Theta(n-2) + \Theta(n-1) + \Theta(n)$$

.....

.....

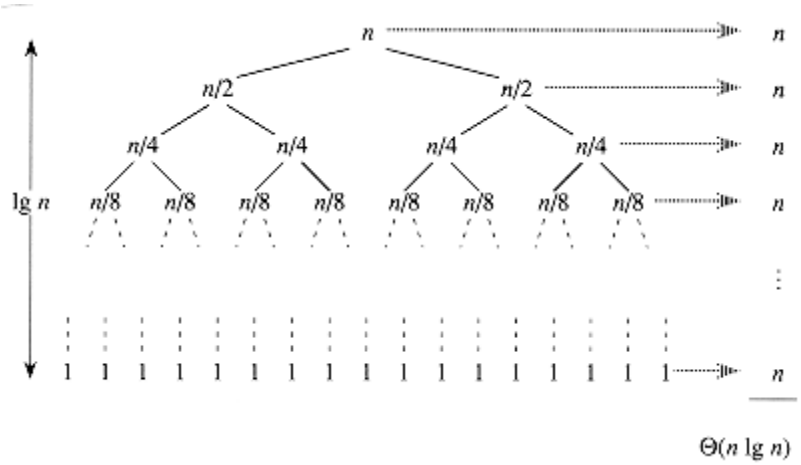
$$T(n) = \Theta(1) + \sum_{k=1}^{n} k = \frac{n(n+1)}{2}$$

$$T(n) = \Theta(n^2)$$

Muhammad Sarwar QAU Islamabad 9

En iyi durum analizi

En dengeli durumda her bölümleme yaptığımızda dizimiz iki neredeyse eşit uzunlukta alt diziye bölünür. Bu her öz yineleme çağrısının toplam boyutun yarısı kadar bir alt diziyi işleyeceği anlamına gelir. Sonuç olarak $\log_2 n$ derinliğinde bir özyineleme çağrı ağacı(call tree) oluşur. Dolayısıyla her seviyedekiler Omega(n) sürede işlemini gerçekleştirir. Sonuç olarak Omega (n * $\log_2 n$) *zaman karmaşıklığında yani n $\log_2 n$ *sürede işlem en iyi durumda tamamlanmış olur.***



$$
\begin{aligned}
T(N) &= 2T(N/2) + cN \\
\frac{T(N)}{N} &= \frac{T(N/2)}{N/2} + c \\
\frac{T(N/2)}{N/2} &= \frac{T(N/4)}{N/4} + c \\
\frac{T(N/4)}{N/4} &= \frac{T(N/8)}{N/8} + c \\
&\vdots \\
\frac{T(2)}{2} &= \frac{T(1)}{1} + c \\
\frac{T(N)}{N} &= \frac{T(1)}{1} + c \log N \\
T(N) &= cN \log N = O(N \log N)
\end{aligned}
$$

Görüldüğü üzere tekerrür ilişkisi matematiksel olarak çözüp ispatlanabilir.
Ancak dizi sol alt dizi sağ dizi ve eşit elemanlar şeklinde üç diziye bölünüp işlem yapılırsa, "three way partition", bu durumda en iyi durum aynı elemanları karşılaştırmaktan kaçınacağı için O(N) olarak bulunur.

Ortalama Durum Analizi

Ortalama durumu tekerrür ilişkisi çözerek yapacağım. Ortalama çalışma süresi(run time) t(n), iki ayrı dizi için gereken özyinelemeli(recursive) çağrıların ve pivot seçimi için ve de iki eşit diziye bölerken geçen zamanın toplamına eşittir. Pivot seçimi ve bölerken geçen zamana f(n) dersek bu Omega(n) = n süresindedir. Çünkü diziyi bir kere dolaşırız bunu rastgele seçmek için. O zaman sol dizi + sağ dizi için gereken

zaman + pivot seçimi için gereken zaman için aşağıdaki bağlantıyı yazarız.

t(n) = t(i) + t(n − i − 1) + n, for n > 1, and t(0) = t(1) = 1,

Burada i bizim sol alt dizimizin uzunluğudur, 0 ile n-1 arasında değerler alır, çünkü pivotun solundaki veya sağındaki dizi 0 ile n-1 arasında değerler alabilir. Toplam dizimizin boyutu n'dir.

Pivotun solundaki dizi + sağındaki dizi için pivotun bulunduğu konuma göre N tane farklı diziliş biçimi vardır. Çünkü pivot N farklı poziyonda yer alabilir. Dolayısıyla bunlardan birini olasılıksak olarak seçme ihtimalim 1/N dir. Aşağıdaki bağıntıyı elde ederiz.

$$t(n) = \frac{1}{N}[\sum_{i=0}^{n-1} t(i) + t(n-i-1)] + n$$

$$\sum_{i=0}^{n-1} t(i) = \sum_{i=0}^{n-1} t(n-i-1)$$

Olduğu için de yukarıdaki bağıntıyı düzenlersek;

$$t(n) = \frac{2}{N}[\sum_{i=0}^{n-1} t(i)] + n$$

bağıntıını elde ederiz. Sonrasında denklemin her iki tarafını n ile çarparsak aşağıdaki denklemi elde ederiz.

$$nt(n) = 2[\sum_{i=0}^{n-1} t(i)] + n^2$$

Sonrasında n yerine n-1 yazarsak;

$$(n-1)t(n-1) = 2[\sum_{i=0}^{n-2} t(i)] + (n-1)^2$$

Bir üstteki denklemden bulduğumuz denklemi çıkartırsak;

$$nt(n) - (n-1)t(n-1) = 2t(n-1) + 2n - 1$$

sonucunu elde ederiz. Sadeleştirme yapacak olursak;

$$nt(n) = (n+1)t(n-1) + 2n$$

Şimdi de denklemi çözülebilir hale getirmek için her iki tarafını n*(n+1) ile bölelim.

$$\frac{t(n)}{n+1} = \frac{t(n-1)}{n} + \frac{2}{n+1}$$

Denklemi çözmek için "back substitution" yani geriye doğru değişkenlerin değerini yerleştirme metodu kullanılırsa;

$$\begin{aligned}
\frac{t(n)}{n+1} &= \frac{t(n-2)}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\
&= \frac{t(n-3)}{n-2} + \frac{2}{n-1} + \frac{2}{n} + \frac{2}{n+1} \\
&= \cdots \\
&= \frac{t(1)}{2} + 2\sum_{i=3}^{n+1} \frac{1}{i} \\
&\approx \frac{1}{2} + 2[\ln(n+1) + \gamma - \frac{3}{2}]
\end{aligned}$$

$\gamma \approx 0.577$

Euler sabitidir. Buradan;

$$\frac{t(n)}{n+1} = O(\log n)$$

ve buradan da;

$$t(n) = O(n \log n).$$

Sonucuna ulaşırız. Görüyoruz ki beklediğimiz gibi ortalama n*log(n) sürede buluyor, bu değer en iyi durum ile en kötü durumun arasında bir değerdir. En iyi durum logn, en kötü durum n^2. Ayrıca ortalama durum en kötü durumun yaklaşık %39'u kadar sürede buluyor.
Bu hızlı ortalama çalışma zamanı (run time) Quick sort algoritmasının diğer sıralama algoritmalarına göre neden baskınlık sağlaması için ayrıca bir nedendir.

Kapanış

Bu çalışma Quicksort algoritması için detaylı bir çalışma örneğidir. Referans olarak Türkçe bilgi eksizliğinden dolayı Wikipedia İngilizce Quicksort makalesi çevrilmiştir. Bilkent Üniversitesi Bilgisayar Mühedisliği bölümü bazı slaytları kullanılmıştır. Quick sort'un average durumu için detaylı matematiksel analiz için http://pami.uwaterloo.ca/~hanan/ Dr. Hanan Hayad'ın makalelerinden çeviri yapılmıştır. Okuduğunuz için teşekkürler.

Tayfun Adakoğlu