Bursa Uludağ Üniversitesi

Bilgisayar Mühendisliği Bölümü

Algoritma Analizi

Proje Ödevi Rapor 1

Sorting Algorithms : Quick Sort - Insertion Sort

Samet YILMAZ- 032090094

Ece JİLTA- 032090105

Hatice Feyza Benal- 032090082

İÇİNDEKİLER

GİRİŞ

Sıralama problemi, araştırmacılar tarafından bilgisayar biliminin tüm dalları için temel bir görev olarak görülmektedir. Tamamen sıralı bir kümeden gelen bellekteki ardışık öğelerin bir girdi koleksiyonunun istenen bir sırayla yeniden düzenlenmesinden oluşur, 1sayılar veya dizeler gibi. Sıralamanın en büyük önemi, istediğimiz sırada bir sıramız varsa, bir öğeyi aramak gibi diğer önemli işlemleri daha verimli bir şekilde uygulayabilmemiz ve böylece gerçek zamanlı sorgulara hızlı bir şekilde yanıt verebilmemizdir. Arama, verileri sıraladıktan sonra sıklıkla ortaya çıkan sıralama sorununa çok yakın bir görevdir. Örneğin, bir web arama motoru, verileri düzenli bir şekilde düzenlenmedikçe saniyeler içinde yanıt veremez. Bilimsel araştırmanın çeşitli alanlarında yeni önerilerin tasarımı için verimli sıralamanın gerekli olduğu birçok uygulama bulabiliriz.

Genel amaçlı algoritmaların çoğunda temel amaç, hesaplama süresini en aza indiren bir problemi çözmektir. Sıralama sorununun durumu da farklı değil. Sıralama algoritmalarının en önemli amaçlarından biri, iyi bir ampirik performans elde etmeyi umarak, problem için optimale yakın bir teorik zaman sunmaktır. Ancak büyük veriler bizi sadece CPU zamanında değil, bellek ve güç tüketimi gibi diğer önemli kaynakların kullanımında da verimli olmaya zorluyor. İyi haber şu ki, bir algoritma mükemmel işleme süresi sunuyorsa ve bellekteki verileri işlemede verimliyse, yürütülmesi için daha düşük enerji maliyetine sahip olacaktır. Aynı doğrultuda, geleneksel sorunları çözen klasik algoritmalar, büyük ölçekli veri işleme problemiyle başa çıkmak için zorlanmaktadır. Bu nedenle, on yıllardır çok hızlı olmasıyla öne çıkan bir algoritma, büyük verileri işlememiz gerektiğinde mutlaka uygun bir yöntem değildir.

Sıralama, bilgisayar bilimlerinde onlarca yıldır üzerinde çalışılan temel problemlerden biridir. Neredeyse tüm teknoloji alanları, görüntü işleme, multimedya, bilimsel hesaplama, ağ oluşturma gibi sıralama işlemleri gerektirir. Daha büyük. Tüm algoritmalar, öğe sayısı arttıkça sıralama yapmak için daha fazla zaman alırken, bazıları diğerlerinden daha yavaştır. Gerçek zamanlı işlemde hesaplama ve sıralamadaki doğal algoritmik farklılığı nedeniyle, sıralama programlarını bugün genel amaçlı bilgisayarlarda yürütmek verimli değildir. Ayrıca, yazılım algoritmasının yüksek hıza ulaşması mümkün değildir. Bu nedenle, sıralama işlemi için ALU (Aritmetik Mantık Birimi) içinde özel bir donanım bloğu uygulamak doğal olacaktır.**[1]**

VLSI tasarım alanının hızla gelişmesiyle, böyle bir sıralama modülü, tek bir çip üzerinde FPGA (Alan Programlanabilir Kapı Dizisi) kullanılarak uygulanabilir. Tasarımcının, kapı seviyesi paralelliği ile uygulama tasarımını kolayca değiştirebildiği, yeniden yapılandırılabilir bir kapı dizisi türüdür. Bir depolama aygıtından çok sayıda veri bir ALU'ya ilerler ve sıralanan sonuç depolama aygıtına geri yüklenir. Bu, veri kümesinin sıralanmasının bir sıralayıcıdan ve sıralayıcıya sıralı aktarım gerektirdiğini

belirtir. FPGA'da çok sayıda bellek mevcuttur ve az sayıda pin ile erişilebilir. Böylece, FPGA uygulaması paralellik ile daha iyi bir hız performansı sağlar. Tüm sıralama algoritmalarından bazıları az miktarda veri için verimli, bazıları ise çok sayıda veri için etkilidir. Seçim sıralama ve ekleme sıralama, az sayıda veri için birleştirme sıralama veya hızlı sıralama gibi karmaşık sıralama algoritmalarından daha hızlı olan en yaygın ve temel sıralama algoritmalarıdır. Bu çalışmada Verilog HDL dili kullanılarak seçmeli sıralama ve eklemeli sıralama algoritması uygulanacaktır. Tüm uygulama için RTL diyagramı verilecek ve ilgili zamanlama diyagramı en kötü durum için incelenecektir. Bazı operasyon parametreleri incelenecektir. Karşılaştırmalı analiz ve sentez için rapor ve zamanlama özet raporu tablolaştırılacak ve parametrelerin analizine dayalı olarak karşılaştırma sonucu verilecektir. **[2]**

Sorting Algorithms

Birçok farklı sıralama algoritması mevcuttur.

- Insertion Sort
- Quick-sort
- Bucket-sort
- Radix-sort
- Merge-sort (recap)
- Selection Sort
- Heap-sort (next lecture)
- Timsort (next lecture)

Sort	Computational Complexity	Memory
Bubble	O(n)	O(1)
Selection	O(n)	O(1)
Insertion	O(n)	O(1)
Shell	O(n log2 n)	O(1)
Quick	O(n log n)	O(log n)
Merge	O(n log n)	O(n)
Heap	O(n log n)	O(n)
Clump	O(n log n)	O(log n)

[11]

İnsertion Sort

Eklemeli sıralamanın, neredeyse sıralanmış dizilerde diğer O(n 2) sıralama algoritmalarından daha iyi performans gösterdiği bilinmektedir. Bu yazıda , genel durumlarda diğer tüm O(n 2) sıralama algoritmalarını geride bırakan, sıralama için iki bölümlü bir ekleme algoritması geliştiriyoruz . Farklı sıralama algoritmaları tarafından alınan toplam zaman grafikleri, algoritmamızın mevcut diğer benzer algoritmalara göre üstünlüğünü doğrulamaktadır.

Bu yazıda ilk olarak, veri setini iki eşit bölüme ayırdığımız bir sıralama algoritması geliştirdik, öyle ki bir bölümdeki tüm öğeler diğer bölümlerin tüm öğelerinden daha küçük olacak ve her iki bölüm de sıralanacak. Gereken yineleme sayısını yarıya indirerek algoritmanın zaman karmaşıklığını daha da azaltıyoruz.

Esas olarak iki tür karşılaştırma tabanlı sıralama algoritması vardır (i)O (n2)ve (ii)O (n günlüğün). Genel olarakO (n2)sıralama algoritmaları daha yavaş çalışırO (n günlüğün)algoritmalar, ancak yine de önemi göz ardı edilemez. önemiO (n2)algoritmalar ile birlikte kullanılabilmeleridir.O (n günlüğün)algoritmalar ve son algoritma, uygun bir durumu eşleştirmek için uygulanabilir, örneğin büyük bir diziyi daha küçük dizilere bölmek, bu dizileri sıralamakO (n2)sıralama algoritması ve ardından tamamen sıralanmış bir dizi üretmek için bunları birleşti rin. **[3]**

Arama işlemlerine genellikle bilgisayar veri işlemede ihtiyaç duyulur. Daha verimli arama yönteminin kullanılabilmesi için genellikle verilerin anahtar kelimelere göre sıralı bir düzende işlenmesine ihtiyaç duyulmaktadır. Bu nedenle, sıralama işlemi bilgisayar programcılığındaki temel işlemlerden biridir.

DoğrudanInsertion Sort Algoritması

Doğrudan ekleme sıralama algoritması, basit bir dahili ekleme sıralama yöntemidir.

Algoritma Tasarım Fikirleri

Doğrudan eklemeli sıralama algoritmasının temel tasarım fikri aşağıdaki gibidir:

Sıralanacak tabloyu önce sıralı kısım ve sırasız kısım olmak üzere ikiye ayırın; Ardından, sıralanmamış parçanın baş kaydını, kademeli olarak sıralı bir tablo oluşturacak şekilde sıralı parçaya yerleştirin.

Algoritma yürütmesindeki somut adımlar aşağıdaki gibidir:

İlk sıralama gezisinde, sıralı kısım, sıralanacak tablonun yalnızca ilk kaydından oluşur ve sırasız kısım geri kalanından oluşur, bu nedenle, başı konumlandırmak için ilk kayıt anahtar sözcüğü ile ikincisi arasında yalnızca bir karşılaştırmaya ihtiyaç duyar.

SONUÇ:

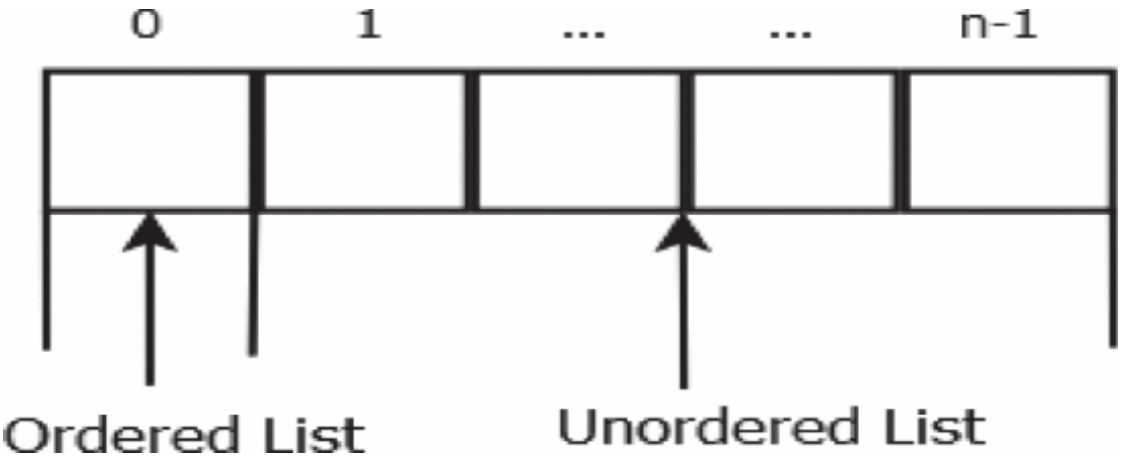
2 elemanlı eklemeli sıralama algoritması, geleneksel eklemeli sıralama algoritmasına dayanan geliştirilmiş bir algoritma tasarım fikridir.

2 elemanlı eklemeli sıralama algoritması, doğrudan eklemeli sıralama algoritmasıyla aynı ortalama zaman karmaşıklığına ve alan karmaşıklığına sahiptir, ancak orijinalinden daha verimlidir ve yeni algoritma, orijinal olandan bir tane daha fazla yardımcı değişken alanı sunar. Ayrıca yeni algoritma, algoritmaların basitliği açısından orijinalinden daha karmaşıktır.

Doğrudan eklemeli sıralama algoritması, tanımlanan RecordType türüne (burada n, sıralanacak kayıt sayısıdır) dayalı olarak yukarıda C'de açıklanabilir

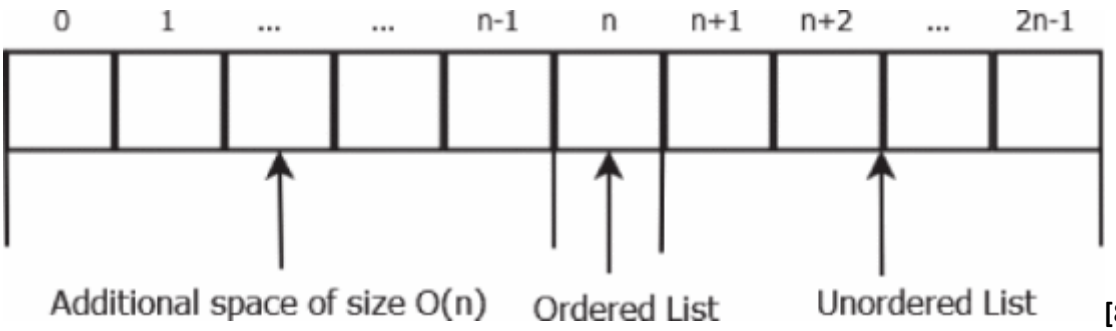
Bu algoritma, C'deki dizi alt indisinin başlangıç sayısı olan sıfırın özelliklerini kullanır,r [0]değişkenlerin taşınmasında ara geçici değişkendir, bu nedenle daha önce açıklanan algoritmanın başlangıç koşulu, sıralanacak kayıtların1 ~ nalt simge biri mleri. **[5]**

Bu makalede Insertion Sort'un uyarlanabilirliğini arttırmaya yönelik çalışmalardan bahsetmektedir.



Ekleme Sıralama, sıralanacak eleman listesini sıralı ve sırasız olmak üzere 2 parçaya böler. Sıralı, listenin ilk öğesini içerirken, sırasız kısım, sıralanacak listenin kalan öğelerini içerir. Şeki1 sıralanacaklistenin ilk durumunu gösterir. Liste, Şekil l' de gösterildiği gibi bölündükten sonra, sırasız listedeki her eleman, listenin tüm elemanları sıralı kısımda olana kadar listenin sıralı kısmına eklenir.

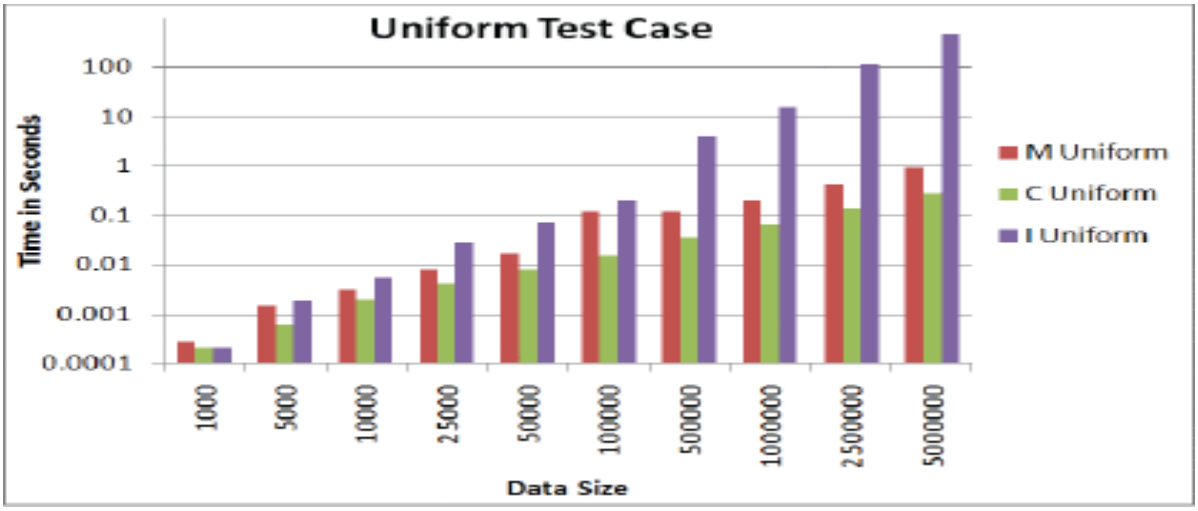
Adaptive Insertion Sort, AIS, eklemeli sıralamada olduğu gibi listeyi bölümlere ayırır ve geleneksel eklemeli sıralamada olduğu gibi sırasız listedeki öğeyi sıralı listeye ekler. Bununla birlikte, AIS, sıralı listedeki sırasız listeden elemanın konumunu bulmak için ikili aramayı kullanır. **[8]**



[8]

Bu çalışmada yazarlar, bölmeler üzerinde birleştirme, sayma ve yerleştirme sıralamasını ayrı ayrı kullanmışlar ve sonuçlar birbirleriyle karşılaştırılmıştır. Algoritmaları test etmek için sıralama kriteri kullanılmışlardır. Algoritmaların zamandan ve mekândan tasarruf etme analizini yapmışlardır.

Bu araştırmada birleştirme, ekleme ve sayma sıralama, paketler için yerel sıralama olarak kullanıldığı tespit edilmiştir.



M = Birleştirme Sıralama (merge sort)

C = Sayım Sıralaması (count sort)

I = Ekleme Sıralaması (insertion sort)

Bunun sonucunda sıralama algoritmalarının kendi içinde kıyaslanmasıyla birlikte insertion sort'un zaman açısından daha verimsiz olduğuu test edilmiştir **[9]**

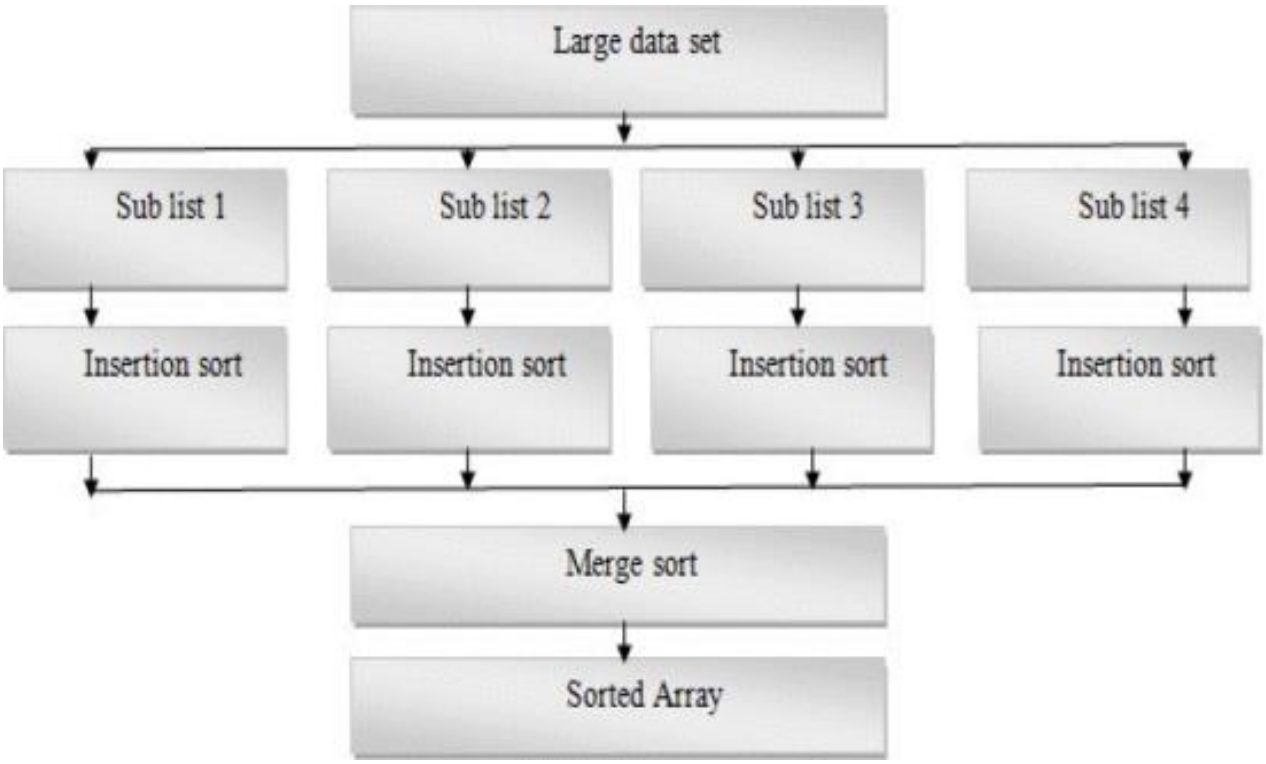
Bu çalışmada sıralamanın bilgisayar bilimi alanında yaygın bir sorun olduğundan bahsetmişlerdir. Sıralama algoritmalarının performansı, kararlılık, uyarlanabilirlik, zaman karmaşıklığı, uzay karmaşıklığı olmak üzere dört parametreye dayalı olarak analiz edilmiş ve tüm sıralama algoritmalarının karmaşıklığı, çoğunlukla sıralanacak toplam öğe sayısının genel bir işlevi olarak çalışma süresini ölçtüğünü belirtmişlerdir. Her algoritmanın olumlu ve olumsuz etkileri vardır. Sıralama algoritmaları alanında, çok basit, bellekte fazla yer gerektirmeyen ve az sayıda girdi için daha hızlı olan algoritmalar vardır. Bununla birlikte, girdilerin sayısını artırarak bu algoritmalar daha fazla zaman almaya ve yavaşlamaya başlar. Burada odaklanılan ana algoritmalar Insertion, Qucik ve Binary Search olmuştur.

Algoritmalarımızın en büyük avantajı, mevcut algoritmaların en kötü durumunda en iyi performansı vermeleri ve eleman sayısına bakılmaksızın tüm sayıları oldukça düşük sayıda karşılaştırma ile çok verimli bir şekilde sıralamalarıdır. Bu kombinasyon algoritmaları, hesaplama performanslarını artırmak için çok çekirdekli işlemcilere uyacak şekilde tasarlanmıştır.

Bu çalışmanın sonucunda elde ettikleri veriler, farklı donanım, işletim sistemi kullanımı ile sıralama algoritmalarının verimliliğinin değiştiğini ve yürütme zamanının sıralama performansı üzerinde önemli bir etkisi olduğunu tespit etmişlerdir.

Bu deneylerde, ubuntu ortamındaki en iyi sıralama algoritması eklemeli sıralama (ınsertion sort), en kötü sıralama algoritması ise hızlı sıralama (quick sort) idi.

Eklemeli sıralamanın (Insertion Sort) ana avantajı basitliğidir. Az sayıda elemanla uğraşırken iyi bir performans gösterir. Ancak eklemeli sıralamanın ana dezavantajı, dizi ters sırada sıralandığında iyi performans göstermemesidir.

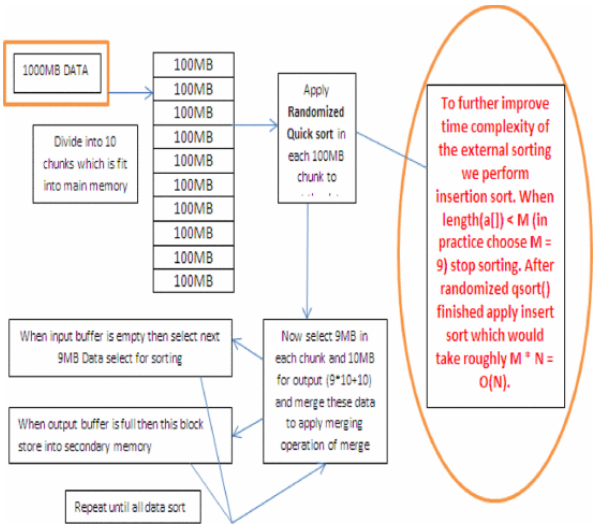


Hızlı sıralama (Quick Sort), böl ve yönet ilkesine göre çalışır. İlk olarak, seçilen bir pivot elemana dayalı olarak eleman listesini iki alt listeye böler. Birinci alt listedeki tüm öğeler, pivottan daha küçük olacak şekilde düzenlenirken, ikinci alt listedeki tüm öğeler, pivottan daha büyük olacak şekilde düzenlenir. Aynı bölümleme ve düzenleme işlemi, tüm eleman listesi sıralanıncaya kadar, ortaya çıkan alt listelerde sürekli ve tekrar tekrar gerçekleştirilir. Hızlı sıralama, en iyi sıralama algoritmalarından biri olarak kabul edilir. Bunun nedeni, çok sayıda öğeyle iyi başa çıkma yeteneğinden dolayı verimlilik açısından önemli avantajıdır. Yerinde sıralandığı için ek depolamaya da gerek yoktur. Ancak hızlı sıralamanın en kötü durumu şudur: **[10]**

- Dizi zaten aynı sırada sıralanmış.
- Dizi zaten ters sırada sıralanmış.
- Tüm elemanlar aynı

[10]

Insertion Sort, Küçük veri kümeleri için analiz edilirken bağımsız ekleme sıralama çalıştırmak, birçok küçük sıralamayı başlatmak ve durdurmak için çok zaman kazandırır, ancak büyük segment sınırları boyunca anahtarları karşılaştırma maliyetini azaltır. Bu sonuçta rastgele hızlı sıralamadan daha az bir zaman karmaşıklığı ile sonuçlanır.



Bu makale, büyük bilgi kayıtları için harici sıralamayı teşvik etmek için yeni bir yaklaşım önermektedir. Insertion Sort ve Harici Sıralamada Random Quick Sort'un dahil edilmesiyle, büyük veri kümelerini sıralamanın sonucunu yaklaşık %20'ye çıkarıyoruz. **[12]**

Quick Sort

Hızlı sıralama algoritması, yüksek verimliliği, hızlı hızı, bilimsel yapısı nedeniyle veri işleme sistemlerinde yaygın olarak kullanılmaktadır. Bu nedenle, hızlı sıralama algoritmasının zaman karmaşıklığına dayalı kapsamlı bir çalışma büyük önem taşımaktadır. Özellikle zaman karmaşıklığı açısından, hızlı sıralama algoritması ile diğer algoritmanın karşılaştırılması özellikle önemlidir. Bu makale, hızlı sıralama algoritmasının zaman karmaşıklığından bahsetmekte ve hızlı sıralamayı diğer sıralama algoritmalarıyla ilişkilendirmek için kurulan fonksiyonun birinci dereceden türevini analiz ederek geliştirilmiş kabarcık sıralama ile hızlı sıralama arasında bir karşılaştırma yapmaktadır.

```
void QuickSort(int L[], int first, int last){
      if(first<last){
          int split=part(L,first,last);
          QuickSort (L,first,split-1);
          QuickSort (L,split+1,last);
       }
}
```

Yukarıdaki programdan, hızlı sıralamanın ana fikrinin bölme işlemi olduğunu görebiliriz. Hızlı sıralamanın diğer sıralama yöntemlerinden daha hızlı çalışmasının dahili nedenlerinden biri olan bölümleme programını çalıştırırken öğeler dizide büyük ölçüde hareket eder.

İlk olarak, bölme programını çalıştırmak için ilk ve son olarak adlandırılan iki işaretçi ayarlanmalıdır. İki işaretçinin başlangıç değerleri, sırasıyla sıralanmayı bekleyen listenin üst sınırı ve alt sınırıdır, yani, LP=ilk, RP=son. Listede n eleman varsa, o zaman ilk=0, son=n-1. Listedeki ilk elemanı pivot olarak seçiyoruz, yani pivot= L[first]. Bölümleme programı, pivottan daha küçük bir öğe bulana kadar tüm listeyi RP adlı işaretçiden sola taradı, ardından öğeyi LP adlı işaretçinin gösterdiği konuma getirin ve LR işaretçisini bir konum sağa kaydırmasını sağlayın. . Bundan sonra, bölümleme programı, pivottan daha küçük olmayan bir öğe bulana kadar listenin geri kalanını LP adlı işaretçiden sağa doğru taradı, daha sonra öğeyi RP adlı işaretçinin gösterdiği konuma getirin ve RP işaretçisini bir konum sola kaydırmasını sağlayın. Aslında, bölme programı yukarıdaki işlem boyunca bir döngüsünü tamamlar.

Hızlı Sıralama Performansını Geliştirmek İçin Bir Analiz

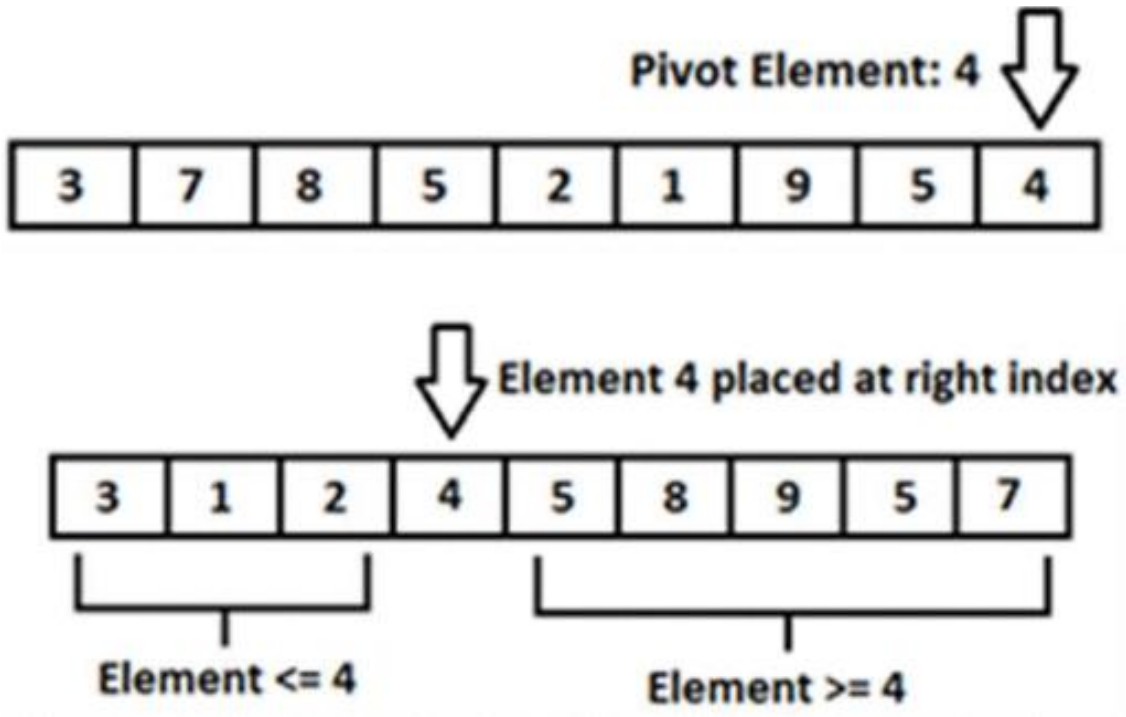
Bölme, hızlı sıralamada zaman karmaşıklığı ile ağırlaşır. Bu nedenle, pivot olarak uygun bir elemanı seçmek hızlı sıralama için çok önemlidir. Uygun eleman, listenin ilk, orta ve son konumunda bulunan elemanların ortalamasına en yakın olmalıdır. Seçtiğimiz uygun eleman V olsun, o zaman hızlı sıralama, bölme işlemleri yapılmadan önce V ile listenin ilk konumundaki eleman arasında değer değiş tokuş etmelidir.

Hızlı sıralama, algoritmanın uygulanmasına ve analizine yardımcı olan özyinelemeli algoritma yöntemini benimser. Bununla birlikte, yöntem, zaman karmaşıklığını ve uzay karmaşıklığını arttırmak için algoritmanın büyük bir yığın gerektirmesini sağlar. Hızlı sıralama algoritmasının aşağı itilen yığın prosedürlerinin sıklığını azaltabilmesi için özyinelemeli algoritmalar yönteminin yerini almak için program içinde bilimsel olarak yığınlar oluşturabiliriz. **[6]**

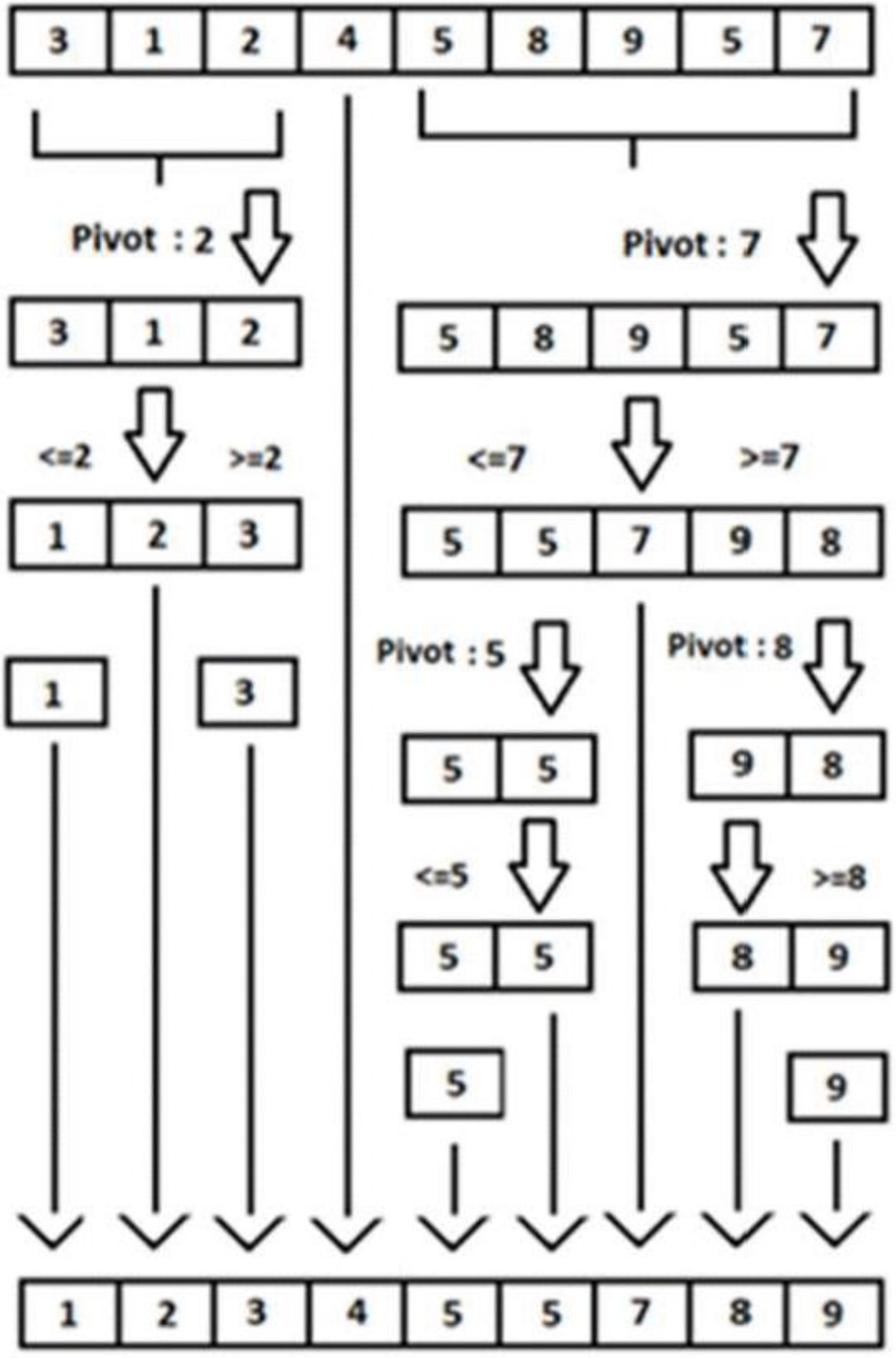
Zamanın bir noktasında en hızlı algoritmaydı. Ancak bazen polinom zaman karmaşıklığı verebilir. Bu algoritmada önemli olan tek şey Pivot Eleman seçimidir.

Quicksort algoritmasının fikri, bir pivot eleman seçmek ve tüm elemanları, pivottan daha küçük olan elemanlar pivotun sol tarafına, pivottan daha büyük olan elemanlar ise sağ tarafına yerleştirilecek şekilde düzenlemektir . Daha sonra aynı işlem pivotun sol tarafında bulunan elemanlar ve pivotun sağ tarafında bulunan elemanlar üzerinde gerçekleştirilir.

Dizinin [3, 7, 8, 5, 2, 1, 9, 5,4] olduğunu varsayalım. Pivot olarak herhangi bir öğeyi seçebilirsiniz. Burada son eleman pivot eleman olarak seçilir.



Daha Hızlı Sıralama Algoritması: Hızlı Sıralamanın zaman karmaşıklığını Doğrusal Logaritmik'e yükseltme:



Hızlı sıralama ile ilgili sorun, pivot olarak minimum veya maksimum öğe seçildiğinde ortaya çıkar. Böyle bir durumda, bu algoritmanın zaman karmaşıklığı polinom olacaktır.

Analysis on 2-element insertion sort algorithm (ieee makale)

2 elemanlı eklemeli sıralama algoritması, geleneksel eklemeli sıralama algoritmasına dayanan geliştirilmiş bir algoritma tasarım fikridir.

2 elemanlı eklemeli sıralama algoritması, doğrudan eklemeli sıralama algoritmasıyla aynı ortalama zaman karmaşıklığına ve alan karmaşıklığına sahiptir, ancak orijinalinden daha verimlidir ve yeni algoritma, orijinal olandan bir tane daha fazla yardımcı değişken alanı sunar. Ayrıca yeni algoritma, algoritmaların basitliği açısından orijinalinden daha karmaşıktır. **[7]**

Küçük öğe listesi için Ekleme sıralama neden Hızlı sıralamadan daha iyidir?

Big-O Notasyonu, asimptotik davranış olarak da bilinen n büyük olduğunda sınırlayıcı davranışı tanımlar. Bu bir yaklaşımdır. (Bkz. http://en.wikipedia.org/wiki/Big_O_notation)

Hızlı Sıralama, özyinelemeli işlev çağrılarından fazladan ek yüke sahip olduğundan, eklemeli sıralama küçük n için daha hızlıdır. Eklemeli sıralama da Hızlı sıralamadan daha kararlıdır ve daha az bellek gerektirir. **[13]**

KAYNAKÇA

- https://www.sciencedirect.com/science/article/pii/S1877750322002253 [1]
- https://ieeexplore.ieee.org/document/9036675 [2]
- https://dsacl3-2019.github.io/slides/dsa3_2019_3_sorts_insert_quick.pdf [3]
- https://ieeexplore.ieee.org/document/5234731 [4]
- https://ieeexplore.ieee.org/document/5541165 [5]
- https://ieeexplore.ieee.org/document/6115063 [6]
- https://ieeexplore.ieee.org/document/9357725 [7]
- .
- https://ieeexplore.ieee.org/document/7943114 [9]
- https://ieeexplore.ieee.org/document/8472956 [10]
- https://ieeexplore.ieee.org/document/5489222 [11]
- https://ieeexplore.ieee.org/document/6954224 [12]
- https://stackoverflow.com/questions/8101546/why-is-insertion-sort-better-than-quick-sort-for-small-list-of-elements [13]