Hızlı Sıralama Algoritması (Quick Sort)

- Yağız Zorlu - 2311502270
- Muhammed Nebi Altın - 2311502278
- Danyel Can - 2211502033
- Gürhan Kaya - 231502277

Sıralama Algoritmaları Nedir?

Sıralama algoritmaları, verileri düzenlemek ve belirli bir kriter doğrultusunda sıralamak için kullanılırlar. Bu sıralama işlemi genellikle veri analizi, veri tabanları, grafik işleme ve çeşitli bilgisayar bilimi uygulamalarında önemlidir. Sıralama algoritmalarının işlevleri şunlardır:

-Bir dizi veriyi belirli bir kriter doğrultusunda artan veya azalan sıraya dizmek, verilerin daha düzenli ve erişilebilir olmasını sağlar.

-Verilerin sıralı olması, arama, ekleme ve silme gibi işlemleri daha verimli hale getirir.

-Sıralı veriler, belirli bir değere erişmek veya belirli bir aralıktaki verilere hızlıca ulaşmak için daha uygun bir yapı sunar.

-Veri sıralaması, birçok algoritma ve uygulama için performansı artırır.

Quick Sort

Quicksort algoritması, basit ve hızlı bir sıralama algoritmasıdır. 1959 yılında Tony Hoare tarafından geliştirilmiştir. Mergesort gibi parçala ve fethet (divide and conquer) prensibiyle çalışmaktadır. Quicksort, veri setini parçalara ayırarak ve her bir parçayı sıralayarak işlem yapar. Bu parçalar daha sonra birleştirilir. Bu algoritmanın uygulanma şekli şöyledir:

Temel Çalısma Prensibi

- Dizide bir pivot elemanı seçilir.(Genellikle en sağdaki eleman seçilir)
- Bu eleman bir nevi referans noktası olarak kabul edilir.
- Bu elemanın soluna kendinden küçük değerdeki elemanlar, sağına ise kendinden büyük değerdeki elemanlar sıralanır. Bunun adı "Partition" dır.
- Solda ve sağda oluşan dizilere de pivot elemanlar seçilip aynı işlem uygulanmaya devam eder.
- Bu şekilde sıralanırken bölünen diziler en son birleştirilir ve ortaya sıralanmış dizi çıkar.(Recursive)

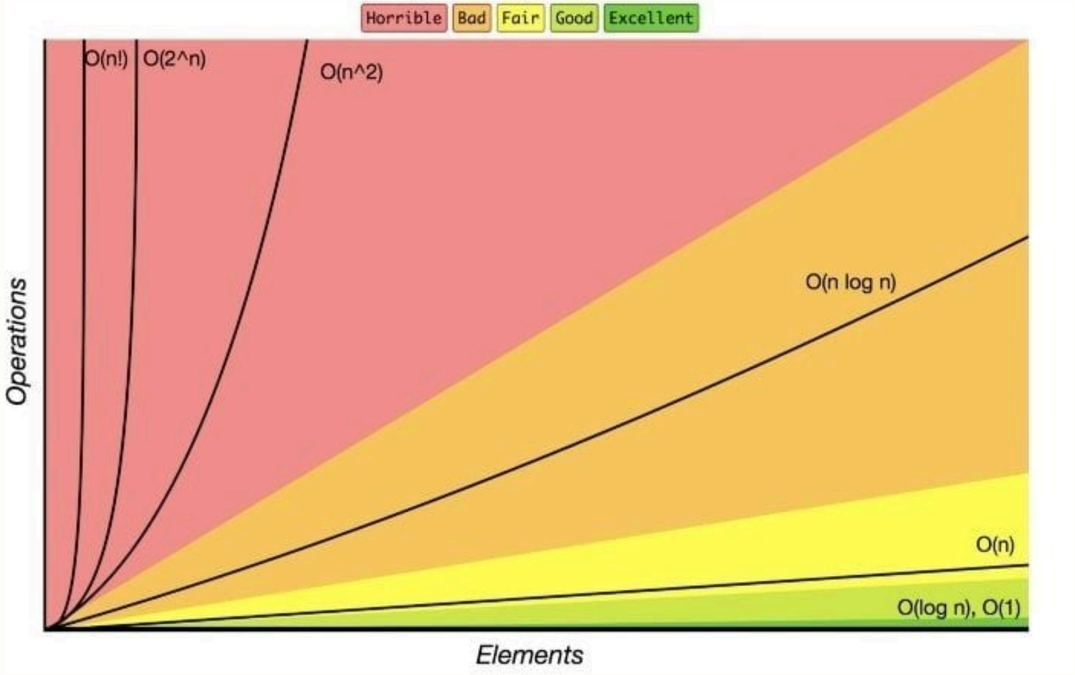
Uygulama Alanları ve Kullanım Senaryoları

- Bilgisayar Bilimleri ve Veri Yapıları
- Uygulama Geliştirme (Verileri sıralamada ve ulaşmaya kolaylık sağlamada)
- Büyük Veri İşleme (Veri madenciliği)
- Oyun Geliştirme (Oyun içi sıralama, puanlama, veri sıralayıp saklama)
- Dizinleme ve Arama Algoritmaları (Hızlı Arama Ağaçları)

QuickSort Zaman Karmaşıklığı

Zaman Karmaşıklığı: En iyi senaryoda O(n logn) olarak çalışır. Her adımda seçilen pivot diziyi neredeyse eşit parçalara böler. Bu durumda, algoritma dengeli bölmeler oluşturur ve verimli bir şekilde sıralama yapar.

En kötü durum senaryosu O(N^2) durumudur. Bu durum, her adımda seçilen pivotun sonucunda son derece dengesiz bölmelere neden olduğunda ortaya çıkar.



QuickSort Alan Karmaşıklığı

Quicksort, elemanları mevcut olarak kullanılan dizide sıralayan yerinde bir sıralama algoritmasıdır. Algoritma ek bir bellek kullanmadığı için sabit O(1) bellek karmaşıklığına sahiptir. Ancak, özyinelemeli çağrılar için yığına bellek ayrılması gerektiği için ortalama bellek karmaşıklığı ortalama durumda O(log n) olur. Ancak, kötü bir pivot seçimi yapıldığında en kötü durumda O(n) seviyesine yükselebilir.

QuickSort Avantajları

- Hızlı çalışma süresine sahiptir.
- En kötü durum senaryoları az görülür. Pivot seçimiyle en aza indirilebilir. İç içe (rekürsif) yapıya sahiptir
- Yerinde bir sıralama algoritmasıdır. Bu da ekstra depolama gerektirmemesini ve minimum hafıza gereksinimi ile çalışmasını sağlar.
- Çalışması için az miktarda bellek gereklidir. Bu yüzden ek yükü düşüktür. Diziler için kullanıldığında iyi referans konumuna sahip olduğundan önbellek dostu bir sıralama algoritmasıdır.

QuickSort Dezavantajları

- En kötü zaman karmaşıklığı O(N^2) dir. Bu da büyük veri kümelerinde düşük performansa neden olabilir.
- Sıralanacak öğelerin orijinal sıralamalarını korumaz. Eğer aynı değere sahip iki öğe varsa, bu öğelerin sıralandıktan sonraki pozisyonları, orijinal dizideki pozisyona bağlı kalmaz. Bu durum sıralama işlemi sonucunda aynı değere sahip öğelerin orijinal sıralamalarını korumaz. Dolayısıyla kararlı bir sıralama algoritması değildir.
- Aynı elemanların farklı konumlarda olması istenmeyen sonuçlar ortaya çıkarabilir. Aynı elemanların sıralanmasında kararsızlığa neden olabilir.
- Çok hızlı fakat nispeten karmaşık bir algoritmadır. Bu nedenle küçük veri grupları için pratik değildir.
- Bu algoritma işlem sırasında çok fazla rastgele erişim gerektirir.Bağlantılı listelerde bu maliyetlidir çünkü tek tek dolaşma gerektirebilir ve çok zaman alabilir.Bu yüzden Quicksort genellikle dizilerde kullanılır.

Pivot Seçimi Stratejileri

1. Son Elemanı Seçme (Last Element Pivot Selection)

- Yaygın olarak kullanılır.
- Pivot en son eleman seçilir.
- Sıralı dizilerde performansı düşüktür. **O(n^2)**
- Ortalama performansı. **O(n log n)**
- Bellek kullanımı **O(log n)** ya da **O(n)** dir.

2. İlk Elemanı Seçme (First Element Pivot Selection)

- Pivot ilk eleman seçilir.
- Sıralı dizilerde performansı düşüktür. **O(n^2)**
- Ortalama performansı. **O(n log n)**
- Bellek kullanımı **O(log n)** ya da **O(n)** dir.

3. Rastgele Eleman Seçme (Random Element Pivot Selection)

- Pivot rastgele seçilir.
- Sıralı olsa bile en kötü durum engeller.
- Ortalama performansı. **O(n log n)**
- Bellek kullanımı **O(log n)** ya da **O(n)** dir.

Pivot Seçimi Stratejileri

4. Üçlü Medyan Pivot Seçimi (Median-of-Three Pivot Selection)

- Pivot baştaki, sondaki ve ortadaki değerin ortada olanı seçilir.
- Pivotun bu şekilde seçilmesi worst case oluşmasını engeller.
- Genel durumlarda dengelidir.
- Ortalama performansı. **O(n log n)**
- Bellek kullanımı **O(log n)** ya da **O(n)** dir.

5. Ortanca Eleman Pivot Seçimi (Median Pivot Selection)

- Diziden elemanlar seçilir ve ortada olan sayı pivot seçilr
- 'Üçlü Medyan pivot seçiminden' farkı pivot için üçten fazla karşılaştırma yapmasıdır.
- Daha büyük veri setlerinde performansı daha yüksektir
- Ortalama performansı. **O(n log n)**
- Bellek kullanımı **O(log n)** ya da **O(n)** dir.

İyileştirmeler ve Optimizasyonlar

1. Pivot seçme tipini değiştirerek daha optimize bir algoritma yazılabilir.

2. Daha küçük alt dizlere inildikçe quick sorttan daha hızlı olan algoritmalar kullanılabilir. (ör: Insertion Sort)

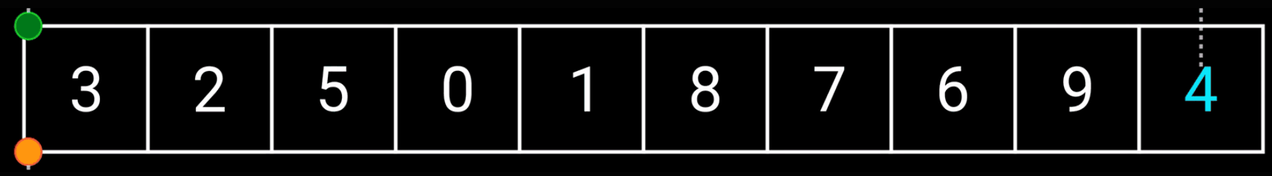
3. Çift yönlü pivot kullanımı büyük veri setlerinde daha etkili olabilir.

Quick Sort Sözde Kod

```
QuickSort (A, low, high)

   if (low < high)
      pivotIndex = Partition(A, low, high)
      QuickSort (A,        low          , pivotIndex - 1)
      QuickSort (A, pivotIndex + 1 , high)

Partition (A, low, high)
   pivot = A[high]
   i = low - 1
   for (j = low; j < high; j++)
      if (A[j] <= pivot){
         i ++
         Swap(A[i], A[j])
         }
   Swap(A[i + 1], A[high])
   return i + 1
```



Parçalama kısmına gerekli bilgileri gönderme

Sol tarafı recurisve şekilde sıralar

Sag tarafı recurisve şekilde sıralar

Pivot dizinin en sonundaki seçilir

İki adet index sayacı tanımlanır i ve j.
j pivottan büyük ise artmaya devam eder.
Değilse i indexi 1 artar. Ardından i ve j yer değiştirir

Dizinin sonuna gelindiğinde (for döngüsü bittiğinde) pivot ve i indexinde bulunan değerler yer değiştirir.

Quick Sort Animasyon

Quick Sort Animasyon linki :

https://www.youtube.com/watch?v=WprjBK0p6rw&t=38s

Quick Sort Java Kodu 1

```java
import java.util.Arrays;

public class QuicksortAlgorithm {
    public static void QuickSort(int[] A, int start, int end){  3 usages
        if (end <= start) return;
        int pivot = partition(A, start, end);
        QuickSort(A, start, pivot - 1);
        QuickSort(A, pivot + 1, end);
    }
    private static int partition(int[] A, int start, int end) {  1 usage
        int i = start - 1;
        int pivot = A[end];
        for (int j = start; j < end; j++){
            if (A[j] < pivot){
                i++;
                int tmp = A[j];
                A[j] = A[i];
                A[i] = tmp;
            }
        }
        i++;
        int tmp = A[i];
        A[i] = A[end];
        A[end] = tmp;
        return i;
    }
    public static void main(String[] args) {
        int[] arr = {12, 5, 7, 3, 8, 10, 1, 15, 6};
        System.out.println("Dizinin İlk Hali : " + Arrays.toString(arr));
        QuicksortAlgorithm.QuickSort(arr, 0, arr.length - 1);
        System.out.println("Dizinin Son Hali: " + Arrays.toString(arr));
    }
}
```

Çıktı

```
Dizinin İlk Hali : [12, 5, 7, 3, 8, 10, 1, 15, 6]
Dizinin Son Hali: [1, 3, 5, 6, 7, 8, 10, 12, 15]

Process finished with exit code 0
```

Quick Sort Java Kodu 2

Partition

```java
public class QuickSort1 {

    public static int partition(int[] array, int start, int end) {  1 usage
        int pivot = array[start]; // Pivotu seç (örneğin, ilk elemanı seçebilirsiniz)

        int leftIndex = start + 1;
        int rightIndex = end;

        while (leftIndex <= rightIndex) {
            while (leftIndex <= rightIndex && array[leftIndex] <= pivot) {
                leftIndex++;
            }
            while (leftIndex <= rightIndex && array[rightIndex] > pivot) {
                rightIndex--;
            }
            if (leftIndex < rightIndex) {

                // Değerlerin yerlerini değiştir
                int temp = array[leftIndex];
                array[leftIndex] = array[rightIndex];
                array[rightIndex] = temp;
            }
        }
        // Pivotun yeni konumunu belirle
        int temp = array[start];
        array[start] = array[rightIndex];
        array[rightIndex] = temp;
        // Pivotun yeni konumunu geri döndür
        return rightIndex;
    }
```

Quick Sort ve Main

```java
    public static void quickSort(int[] array, int start, int end) {  3 usages
        if (start >= end)
            return;                                    // Tek eleman ya da boş dizi zaten sıralıdır
        int pivotIndex = partition(array, start, end); // Pivot elemanını seç ve diziyi pivota göre bölecek indeksi bul
        quickSort(array, start, pivotIndex - 1); // Sol tarafa olan alt diziyi sırala (pivot dahil değil)
        quickSort(array, pivotIndex + 1, end); // Sağ tarafa olan alt diziyi sırala (pivot dahil değil)
    }
    public static void main(String[] args) {
        int[] array = {5, 2, 9, 1, 5, 6};
        quickSort(array, 0, array.length - 1);
        for (int i = 0; i < array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

Quick Sort Çıktı

```
1 2 5 5 6 9

Process finished with exit code 0
```

Sonuç

Hızlı sıralama algoritması, veri yapıları ve algoritmalarında önemli bir yere sahiptir. **Böl ve fethet** stratejisiyle etkili bir şekilde çalışır ve genellikle yüksek performans sergiler. Ancak, pivot seçimi ve optimizasyon yöntemleri dikkatle ele alınmalıdır.

Dinlediğiniz için Teşekkürler

Kaynaklar

https://en.wikipedia.org/wiki/Quicksort
https://stackoverflow.com/
https://www.quora.com/
https://www.geeksforgeeks.org
https://www.javatpoint.com
https://www.mshowto.org
https://medium.com
https://www.yazilimkoyu.org