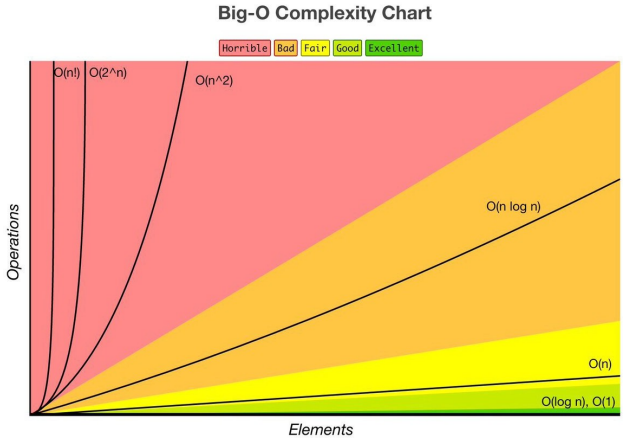
PROGRAMLAMA ÖDEVI 1

Konu Algoritma Karmaşıklık

Kurs Eğitmenleri: Doç Dr. Erkut Erdem, Doç DrDr. Hacer Yalım Kele¸s, Doç . Adnan Özsoy
TA'lar: Sı meyye Meryem Taşyı rek*, Hikmet Can Do˘gancı
Programlama Dili: Java (OpenJDK 11)
Son Teslim Tarihi: **21.03.2025 Cuma (23:59:59)**



1

Algoritmaların analizi, çözüm . farklı yöntemlerinin verimliliğini analiz etmek için araçlar sağlayan bilgisayar bilimi alanıdırBir algoritmanın verimliliği şu parametrelere ; bağlıdıri) ne kadar zaman, ii) bellek alanı, iii) disk alanı gerektirdiği. Algoritmaların temel olarak performansı tahmin etmek ve için geliştirilen algoritmaları karşılaştırmak için kullanılır. analizi aynı görev Ayrıca performans için garantiler sağlar ve teorik temelin anlaşılmasına .yardımcı olur

Bir algoritmanın çalışma süresinin tam bir analizi aşağıdaki adımları içerir:

- Algoritmayı tamamen .uygulayın
- Her bir temel işlem .için gereken süreyi belirleyin
- Temel işlemlerin .yürütülme sıklığını tanımlamak için kullanılabilecek bilinmeyen miktarları belirleyebilme
- Programın .girdisi için gerçekçi bir model geliştirin
- Modellenen girdiyi varsayarak bilinmeyen büyüklükleri analiz edin.
- Her bir işlem ve ardından tüm ürünleri toplayarak .için süreyi frekansla çarparak toplam çalışma süresini hesaplayın

Bu deneyde, farklı sıralama algoritmalarını analiz edecek ve boyutları değişen bir dizi girdi üzerinde çalışma sürelerini karşılaştıracaksınız.

2 Arka Plan ve Problem

Verimli sıralama, girdi verilerinin sıralanmasını gerektiren diğer algoritmaların (arama ve birleştirme algoritmaları gibi) verimliliğini optimize etmek için önemlidir. Bir sıralama algoritmasının verimliliği, farklı boyutlardaki veri kümelerini ve sıralanacak veri kümesi örneklerinin diğer özelliklerini sıralamak için uygulanarak gözlemlenebilir. Bu ödevde, verilen sıralama algoritmalarını iki kritere göre sınıflandıracaksınız:

- **Hesaplama (Zaman) Karmaşıklığı**: Veri kümesinin boyutu açısından en iyi, en kötü ve ortalama durum davranışının belirlenmesi. Tablo 1, bazı iyi bilinen sıralama algoritmalarının hesaplama karmaşıklığının karşılaştırmasını göstermektedir.

  Tablo 1: Bazı iyi bilinen algoritmaların .hesaplama karmaşıklığı karşılaştırması

Algoritma	En İyi Durum	Ortalama Vaka	En Kötü Durum
Seçim Sıralaması	$\Omega(n^{(2)})$	$\Theta(n^2)$	$O(n^2)$
Kabarcık Sıralama	$\Omega(n)$	$\Theta(n^2)$	$O(n^2)$
Yığın Sıralama	$\Omega(n \log)$	$\Theta(n \log n)$	$O(n \log n)$
Hızlı Sıralama	$\Omega(n \log)$	$\Theta(n \log n)$	$O(n^2)$
Sıralamayı Birleştir	$\Omega(n \log)$	$\Theta(n \log n)$	$O(n \log n)$
Radix Sıralama	$\Omega(nk)$	$\Theta(nk)$	$O(nk)$

- **Yardımcı Bellek (Alan) Karmaşıklığı**: Bazı sıralama algoritmaları takas . kullanılarak "yerinde" gerçekleştirilirYerinde sıralama, sıralanan öğeler için kullanılan bellek dışında belleğe yalnızca $O(1)$ yardımcı ihtiyaç duyar. Öte yandan, bazı algoritmalar ihtiyaç duyabilirsıralama işlemleri için yardımcı belleğe . $O(\log n)$ veya $O(n)$ Tablo 2, aynı iyi bilinen sıralama algoritmalarının yardımcı alan karmaşıklığı karşılaştırmasını göstermektedir.

  Tablo 2: Bazı iyi bilinen algoritmaların .yardımcı uzay karmaşıklığı karşılaştırması

Algoritma	Yardımcı Alan Karmaşıklık
Kabarcık Sıralama	$O(1)$
Ekleme Sıralama	$O(1)$
Yığın Sıralama	$O(1)$
Hızlı Sıralama	$O(\log n)$
Sıralamayı Birleştir	$O(n)$
Radix Sıralama	$O(k+ n)$

Zaman karmaşıklığı analizi, bir çözümün maliyetine . toplam hakim olması muhtemel olan algoritmaların verimliliğindeki büyük farklılıklara odaklanırAşağıda verilen örneğe bakınız:

Kod	Birim Maliyet	Times
i=1;	c1	1
toplam= 0;	c2	1
while (i≤ n) {	c3	n+ 1
j=1;	c4	n
while (j≤ n) {	c5	n - (n+ 1)
toplam= toplam+ i;	c6	n - n
j= j+ 1;	c7	n - n
}		
i= i +1;	c8	n
}		

Verilen algoritmanın toplam maliyeti c1+ c2+ (n+ 1) - c3+ n n n n n - - - - - - c4+(+ 1) c5+c6+n - c7+ n - $c8$'dir. Bu algoritma için gereken çalışma süresi, orantılıdır büyüme oranı olarak belirlenen n^2 0ileve genellikle $O(n^2)$ .olarak gösterilir

3 Atama

Bu ödevin temel amacı, algoritma uygulamalarının çalışma süreleri ile teorik asimptotik karmaşıklıkları . arasındaki ilişkiyi göstermektirSözde kod olarak verilen verilen sıralama algoritmalarını uygulamanız ve ampirik verilerin ilgili asimptotik büyüme fonksiyonlarını . takip ettiğini göstermek için veri kümeleri üzerinde bir dizi deney yapmanız beklenmektedirBunu , yapmak içinçalışma süresi ölçümlerinizdeki gürültüyü nasıl azaltacağınızı düşünmeniz asimptotik karmaşıklıkları analiz .göstermek ve etmek için sonuçları çizmeniz gerekecektir

3.1 Sıralama Algoritmaları

- **Karşılaştırma tabanlı sıralama algoritmaları**: Karşılaştırma tabanlı sıralama algoritmalarında, öğeler nihai sıralanmış çıktıdaki . sıralarını belirlemek için karşılaştırılırTüm karşılaştırma tabanlı sıralama algoritmalarının karmaşıklık alt sınırı Ω(n log)'dir. Aşağıdaki karşılaştırma tabanlı sıralama algoritmalarını uygulayacaksınız:
  - Alg'. de verilen tarak sıralaması1
  - Alg. 2'de verilen ekleme sıralaması
  - Alg'. de verilen çalkalayıcı sıralaması3
  - Alg'de . verilen kabuk sıralaması4
- **Karşılaştırma tabanlı olmayan sıralama algoritmaları**: Bazı algoritmalar, genellikle yapıdan veya önceden tanımlanmış varsayımlardan yararlanarak doğrudan öğe karşılaştırmaları olmadan . verileri sıralarKarşılaştırma tabanlı olmayan sıralama algoritmaları $O(n)$ zaman karmaşıklığında . çalışabilirAşağıdaki karşılaştırma tabanlı olmayan sıralama algoritmasını uygulayacaksınız:
  - Alg'. de verilen radix sıralaması5

Algoritma 1 Tarak Sıralama

```
1: prosedür COMBSORT(A: dizi)
2:     boşluk← uzunluk (A)
3:     küçültmek← 1.3
4:     sıralanmış← false
5:     while sorted= false do
6:         gap← max(1, floor(gap / shrink))
7:         sıralanmış← (boşluk== 1)
8:         for i← 0 to length(A) - gap do
9:             eğer ise A[i]> A[i+gap] o zaman
10:                swap(A[i], A[i+gap])
11:                sıralanmış← false
12:            end if
13:        için son
14:    end while
15: prosedürü sonlandır
```

Algoritma 2 Ekleme Sıralaması

```
1: prosedür INSERTION-SORT(A: dizi)
2:     j için← 1, . . . , uzunluk(A) için do
3:         anahtar← A[j]
4:         i← j - 1
5:         while i> 0 and A[i]> key do
6:             A[i+ 1]← A[i]
7:             i← i - 1
8:         end while
9:         A[i + 1]← anahtar
10:    için son
11: prosedürü sonlandır
```

Algoritma 3 Çalkalayıcı Sıralama

```
1: prosedür SHAKERSORT(A: dizi)
2:     değiştirildi← true
3:     takas edilirken do
4:         değiştirildi← false
5:         for i← 0 to length(A)-2 do
6:             eğer A[i]] > A[i+1ise
7:                 swap(A[i], A[i+1])
8:                 değiştirildi← true
9:             end if
10:        için son
11:        takas edilmemişse o zaman
12:            Mola
13:        end if
14:        değiştirildi← false
15:        for i← length(A)-2 to 0 do
16:            eğer A[i]] > A[i+1ise
17:                swap(A[i], A[i+1])
18:                değiştirildi← true
19:            end if
20:        için son
21:    end while
22: prosedürü sonlandır
```

Algoritma 4 Kabuk Sıralama

```
1: prosedür SHELLSORT(A: dizi)
2:     n← uzunluk(A)
3:     boşluk← n / 2
4:     while gap> 0 do
5:         for i← gap to n-1 do
6:             temp← A[i]
7:             j← i
8:             while j≥  gap and A[j - gap]> temp do
9:                 A[j]← A[j - boşluk]
10:                j j ←- gap
11:            end while
12:            A[j]← temp
13:        için son
14:        gap← gap / 2
15:    end while
16: prosedürü sonlandır
```

Algoritma 5 Radix Sıralama

```
1: prosedür RADIXSORT(A: dizi, d: basamak )sayısı
2:     for pos← 1 to d do                          ▷ Her basamak konumunu işleyin
3:         A← COUNTINGSORT(A, pos)
4:     için son
5:     A'yı döndür
6: prosedürü sonlandır
7: prosedür COUNTINGSORT(A: dizi, pos: basamak konumu)
8:     count← 10 sıfırdan oluşan dizi              ▷ Ondalık basamakların (0-9)
varsayılması
9:     A ile aynı uzunlukta← dizisinin çıktısı
10:    boyut← uzunluk (A)
11:    i için← 1, . . . , boyut için do
12:        digit getDigit(A[i], ← pos)
13:        count[digit]← count[digit]+ 1
14:    için son
15:    i için← 2, . . . , 10 için yap
16:        count[i]← count[ count[i]i+− 1]
17:    için son
18:    i için← boyut, . . . , 1 do
19:        digit getDigit(A[i], ← pos)
20:        count[digit]← count[digit] 1−
21:        output[count[digit]] ← A[i]
22:    için son
23:    dönüş çıktısı
24: prosedürü sonlandır
```

3.2 Seti

Verilen sıralama algoritmalarını, FlowMeter aracılığıyla gerçek bir ağ izinden . oluşturulan bir bir bir test test ağının büyük miktarda kaydedilmiş trafik akışını içeren gerçek veri kümesinin kısaltılmış sürümü üzerinde Bu veri kümesi (TrafficFlowDataset.csv)250.000'den fazla içerir., belirli süre boyunca ve alıcılar bir göndericiler arasında çift yönlü olarak gönderilen iletişim paketi yakalamasını

Verilen sıralama algoritmalarının farklı veri boyutları üzerindeki performansının karşılaştırmalı bir analizini yapabilmek için, veri kümesinin birkaç küçük bölümünü yani dikkate alacaksınız,dosyanın başından başlayarak ve 250000 . 500, 1000, 2000, 4000, 8000, 16000, , 32000, 64000128000 boyutlarındaki alt kümelerini Kayıtlarıgöre sıralayacaksınız , türünde olan) *int* 3verilen özelliğine *üncü* sütunda (**Akış SüresiAkışSüresi**(bkz. Şekil 1).

Flow ID	Timestamp	Flow Duration
192.168.1.101-67.212.184.66-2156-80-6	13.06.2010 06:01	2328040
192.168.1.101-67.212.184.66-2159-80-6	13.06.2010 06:01	2328006
192.168.2.106-192.168.2.113-3709-139-6	13.06.2010 06:01	7917
192.168.5.122-64.12.90.98-59707-25-6	13.06.2010 06:01	113992
192.168.5.122-64.12.90.98-59707-25-6	13.06.2010 06:01	3120
192.168.5.122-64.12.90.66-37678-25-6	13.06.2010 06:01	121910
192.168.5.122-64.12.90.66-37678-25-6	13.06.2010 06:01	4073
192.168.5.122-64.12.90.97-56782-25-6	13.06.2010 06:01	128308
192.168.5.122-64.12.90.97-56782-25-6	13.06.2010 06:01	2449
192.168.5.122-205.188.59.193-54493-25-6	13.06.2010 06:01	110814
192.168.5.122-205.188.59.193-54493-25-6	13.06.2010 06:01	2391
192.168.5.122-205.188.155.110-59130-25-6	13.06.2010 06:01	178255
192.168.5.122-205.188.155.110-59130-25-6	13.06.2010 06:01	2955
192.168.1.101-67.212.184.66-2159-80-6	13.06.2010 06:01	9624620

Şekil 1: Yalnızca .Akış Süresi sütunundaki verilerin sıralanması

3.3 Deneyler ve Analiz

Atama Adımları Özetlendi:

- Verilen algoritmaları Java.'da uygulayın
- Verilen veri kümesi .üzerinde deneyler gerçekleştirin
  - Farklı girdi boyutları .ile testler
  - Rastgele, sıralanmış ve tersine sıralanmış girdiler .üzerinde testler
- Sonuç tablolarını doldurun ve sonuçları .çizin
- Bulguları .tartışın

Verilen algoritmaları Java'da uyguladıktan sonra, deney bir dizi gerçekleştirmeniz, sonuçları raporlamanız, göstermeniz ve ve analiz etmeniz bulgularınızı tartışmanız gerekir.

3.3.1 Sıralama ile Deneyler

Verilen deney , setindeverilen sıralama algoritmalarını farklı girdi türleri ve boyutları .üzerinde çalıştıracaksınız

Verilen Veriler Rastgele Üzerinde Deneyler İlk deney setinde, algoritmaları farklı girdi boyutlarına sahip verilen rastgele veri kümeleri üzerinde test edeceksiniz. **Her bir deneyi 10 kez çalıştırarak ve kaydedilen çalışma sürelerinin ortalamasını alarak her çalışma** her bir girdi boyutu için . bir algoritmanın süresini **ortalama** ölçmeniz beklenmektedirBir sonraki deney . seti için sıralanmış girdi verilerini kaydettiğinizden emin olunLütfen yalnızca . sıralama işleminin çalışma süresini ölçtüğünüze dikkat edin$n$ boyutundaki her bir test vakası için girdi sayılarını elde etmek için, verilen veri kümesinin satırını ilk alın.n

Tablo 3'.teki ilk beş boş satırı doldurarak bulgularınızı milisaniye (ms) cinsinden bildirin

Tablo 3: Farklı girdi boyutları için gerçekleştirilen çalışma süresi testlerinin sonuçları (cinsindenms).

	Giriş Boyutu n									
Algoritma	**500**	**1000**	**2000**	**4000**	**8000**	**16000**	**32000**	**64000**	**128000**	**250000**
	Rastgele Giriş Verileri Zamanlama Sonuçları ms cinsinden									
Tarak sıralama										
Ekleme sıralaması										
Çalkalayıcı sıralaması										
Kabuk sıralama										
Radix sıralama										
	Sıralanmış Giriş Verileri Zamanlama Sonuçları ms cinsinden									
Tarak sıralama										
Ekleme sıralaması										
Çalkalayıcı sıralaması										
Kabuk sıralama										
Radix sıralama										
	Tersine Sıralanmış Giriş Verileri ms cinsinden Zamanlama Sonuçları									
Tarak sıralama										
Ekleme sıralaması										
Çalkalayıcı sıralaması										
Kabuk sıralama										
Radix sıralama										

Sıralanmış Veriler Üzerinde Deneyler Bu ikinci deney , , setindealgoritmalarınızı baştan çalıştırmalısınızancak bu sefer önceki deneylerde . elde ettiğiniz zaten sıralanmış giriş verileri üzerinde**Bu adımda da 10 deneme üzerinden aynı ortalama alma kuralı uygulanmalıdır**. Tablo satırı3', sıralanmış girdi verileri için ölçülen yeni çalışma süreleriyle .teki sonraki beş boş doldurun

Tersine Sıralanmış Veriler Üzerinde Deneyler Bu üçüncü deney , setindealgoritmalarınızı tersine sıralanmış giriş verileri . üzerinde çalıştırmalısınızÖnceki deneylerde elde ettiğiniz sıralanmış giriş verilerini ve önce bunları tersine çevirmelisiniz (veya okumalısınızzaten kullanmalı sıralanmış diziyi sondan). Lütfen **sadece sıralama işleminin çalışma süresini ölçtüğünüzden** emin olun. **Bu adımda da 10 deneme üzerinden aynı ortalama alma kuralı uygulanmalıdır**. Tablo 3', tersine sıralanmış giriş verileri için ölçülen yeni çalışma süreleriyle .teki son beş boş satırıdoldurun

3.3.2 Karmaşıklık Analizi ve Sonuç

Tüm tamamladıktan sonra, elde edilen sonuçları birleşik açısından analiz testleriverilen algoritmaların . ve yardımcı uzay karmaşıklığı etmelisinizİlk olarak, Tablo 4 ve i 5'tamamlayın ve cevaplarınızı kısaca gerekçelendirin. Bu algoritmaların genel uzay karmaşıklığı ile ilgilenmediğimiziilgilendiğimizi , sadece sıralama işlemlerini gerçekleştirirken .kullandıkları ek bellek alanı ile unutmayın Lütfen yardımcı uzay karmaşıklığı cevaplarını elde etmek için verilen sözde kodlardan hangi satırları kullandığınızı belirtin.

Tablo 4: Verilen algoritmaların .hesaplama karmaşıklığı karşılaştırması

Algoritma	En İyi Durum	Ortalama Vaka	En Kötü Durum
Tarak sıralama	Ω()	Θ()	*O*()
Ekleme sıralaması	Ω()	Θ()	*O*()
Çalkalayıcı sıralaması	Ω()	Θ()	*O*()
Kabuk sıralama	Ω()	Θ()	*O*()
Radix sıralama	Ω()	Θ()	*O*()

Tablo 5: Verilen algoritmaların .yardımcı uzay karmaşıklığı

Algoritma	Yardımcı Alan Karmaşıklık
Tarak sıralama	*O*()
Ekleme sıralaması	*O*(
O()) Çalkalayıcı sıralaması	
Kabuk sıralama	*O*()
Radix sıralama	*O*()

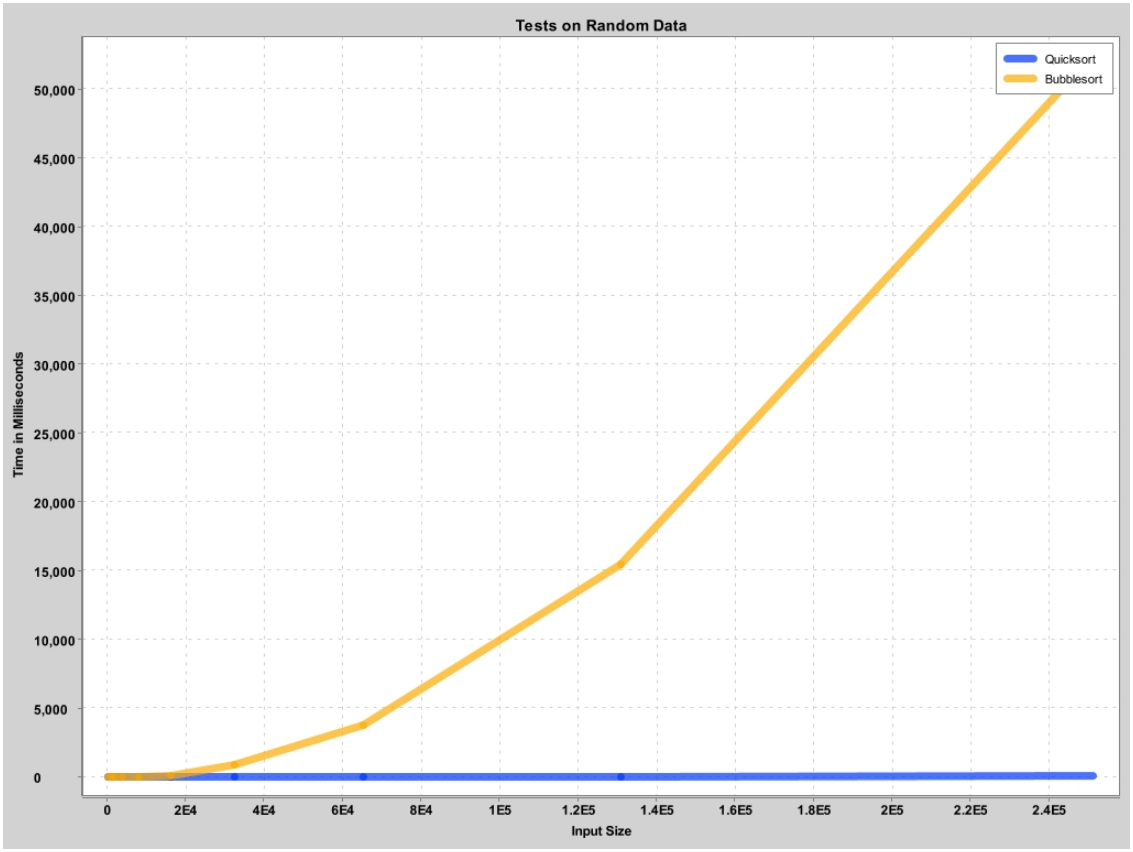
Deneylerden elde edilen sonuçları 3.3.1'de çizin. **Her bir sıralama algoritması için rastgele, sıralanmış ve tersine sıralanmış verilerdeki . performansı gösteren ayrı grafikler elde etmelisiniz**Tablo 3, grafikle her bir birbirleriyle girdi koşulu . altında tüm sıralama algoritmalarını karşılaştıran üç sonuçlanmalıdırTüm grafikler için X *ekseni* girdi boyutunu (girdi örneklerinin sayısı *n*), ise *Y* ekseni çalışma süresini ms . cinsinden temsil etmelidirSonuçlarınızı .nasıl göstermeniz gerektiğine dair bir örnek görmek için Şekil '2ye bakınız

Örnek grafik iki farklı algoritmayı içermekte ve ortalama olarak daha hızlı bir algoritmanın performansının bazı en kötü durum senaryolarında nasıl önemli ölçüde düşebileceğini göstermektedir. Çizimleriniz, şekilde .benzer sunulan tüm algoritmaların sonuçlarını içermelidir

Çizim işlemi, kütüphaneleri . hazır *Java* kullanılarak programlı bir şekilde gerçekleştirilmelidir herhangi bir kütüphaneden yararlanabilirsiniz. Açık kaynak kodlu ve kullanımı oldukça kolay olan . **XChart** kütüphanesini de kullanabilirsiniz**XChart** kütüphanesini kullanmak için öncelikle aşağıdaki linkten indirme butonunu kullanarak *.zip* dosyasını temin etmelisiniz `https://knowm.org/open-source/xchart/` . Ardından, dosyayı ve ayıklamalı **xchart-3.8.1.jar**'ı projenize ekleyin; yani, sınıf yolunuza dahil edin. Yazarlar tarafından sağlanan örnek kodu kontrol edebilirsiniz: şu bağlantıdan `https://knowm.org/open-source/xchart/ xchart-example-code/`.

3.3.3 Sonuçlar Analiz ve

Tablo 4'e ve 3.3.2'de elde edilen elde edilen grafiklere atıfta bulunarak sonuçları kısaca tartışınız. Aşağıdaki soruları yanıtlayınız:



Şekil 2: İki örnek sıralama algoritması için rastgele giriş verileri üzerinde değişen giriş boyutları için çalışma süresi sonuçlarını gösteren örnek bir grafik.

- Sıralanacak girdi verileri ?açısından verilen algoritmalar için iyi, ortalama ve en kötü durumlar nelerdir
- Elde edilen sonuçlar (çalışma algoritma uygulamalarınızın) süreleriteorik asimptotik karmaşıklıklarıyla ?eşleşiyor mu
- Shaker Sort ve Comb Sort, Bubble Sort'a kıyasla performansı artırıyor mu? Eğer öyleyse, yaklaşımları bu iyileşmeye nasıl katkıda bulunuyor?
- Shell Sort 'daha büyük veri kümeleri için tan Insertion Sortdaha iyi performans gösterir mi? Hangi koşullar Ekleme Sıralaması hala iyi performans gösterir?altında
- Radix Sort'un karşılaştırmaya dayalı olmayan bir sıralama algoritması olduğu göz önüne alındığında, büyük sayısal aralıkları nasıl verimli bir şekilde işler?

Notlandırma

- Algoritmaların : uygulanması%20
- Deneylerin yapılması ve sonuçların raporlanması (verilen sonuç tablolarının doldurulması) ve yorum yapılması: 25%

- Verilen iki hesaplama ve yardımcı uzay karmaşıklığı tablosunu tamamlamak ve yorum yapmak: 15%
- Sonuçların : çizilmesi%20
- Sonuçların analizi ve tartışma: %20

Nelere Yer Verilmeli

Bu **Programlama Ödevi Rapor Şablonunu** kullanmanız ve raporlarınızı 'te oluşturmanız LATEXönerilir. Bunun için platformunu Overleaf kullanmanızı öneririz. Bu zorunlu değildir, ancak raporunuzun gerekli tüm bölümlere ve bilgilere sahip olduğundan emin olun (PDF görmek için buraya tıklayınörneğini).

Raporunuz aşağıdakileri içermelidir:

1. Kısa bir sorun açıklaması .ekleyin
2. Verilen sıralama algoritmalarına karşılık gelen Java kodlarınızı ekleyin.
3. Değişen girdi boyutları için verilen rastgele, sıralanmış ve tersine sıralanmış veriler üzerinde oluşturulan tüm deney setlerine karşılık gelen tüm çalışma süresi sonuç tablolarını ekleyin. Beş algoritmanın tamamı test edilmelidir.
4. Verilen algoritmaların teorik hesaplama ve yardımcı alan karmaşıklıklarını gösteren iki tamamlanmış tabloyu, cevaplarınızın kısa bir gerekçesi ile birlikte ekleyin.
5. Adım 3'.ten elde edilen sonuçların en az dört grafiğini ekleyin
6. Verilen soruları .yanıtlayarak elde edilen sonuçları kısaca tartışınız
7. Raporu , hazırlarkençevrimiçi veya başka herhangi bir teorik kaynak kullanabilirsiniz, ancak referansları raporunuza eklediğinizden emin olun. **İnternetten herhangi bir hazır kodu kopyalamayın, çünkü başka birinin de aynı şeyi yapma ihtimali çok yüksektir ve tanımasanız bile hile yapmaktan yakalanabilirsiniz!**

Önemli

- Son başvuru tarihini : kaçırmayın**21.03.2025 Cuma (23:59:59)**.
- Ödeve not verilene .kadar tüm çalışmalarınızı kaydedin
- Gönderdiğiniz ödev çözümü sizin özgün, bireysel çalışmanız olmalıdır. Kopya kopya .veya benzer ödevlerin her ikisi de olarak değerlendirilecektir
- Sorularınızı Piazza üzerinden sorabilirsiniz (`https://piazza.com/hacettepe.edu.tr/spring2025/bbm204`) ve Piazza'da tartışılan her şeyden haberdar olmanız gerekmektedir
- Çalışmanızı adresi üzerinden göndereceksinizaşağıda verilen dosya hiyerarşisi ile :`https://submit.cs.hacettepe.edu.tr/`

```
b< studentID> .zip
└─ src.zip DOSYA<>
   ├─ Main.java DOSYA<>
   └─ *.java DOSYA<>
└─ report.pdf DOSYA<>
```

- Main metodunu içeren main sınıfının adı Mainolmalıdır**.java** . **XChart** kütüphanesini . kullanmanın yararlı bir örneğini içeren **bu başlangıç kodunu** kullanabilirsinizAna sınıf ve diğer tüm sınıflar doğrudan (alt klasör olmadan) zip adlı bir zip dosyasına yerleştirilmelidir**src.**.
- Bu dosya hiyerarşisi gönderilmeden önce sıkıştırılmalıdır (değil.rar , .sadece zip dosyaları desteklenir).

Akademik Dürüstlük

Ödevler üzerindeki tüm çalışmalar **bireysel yapılmalıdır**.. Verilen ödevleri sınıf arkadaşlarınızla , tartışmanız teşvik edilmektedirancak bu tartışmalar soyut bir şekilde yürütülmelidirYani, belirli bir bir sorunun (gerçek kodda veya sözde kodda) belirli çözümüyle ilgili tartışmalara **tolerans gösterilmeyecektir**. Kısacası, başkasının teslim birçalışmasını (dahil olmak üzereinternette) bulunan çalışmalar **da**tamamen veya kısmen kendi çalışmanız olarak **etmeniz akademik dürüstlüğün ihlali** olarak değerlendirilecektir. Önceki koşulun çevrimiçi/AI kaynaklar için de geçerli olduğunu lütfen unutmayın. Bu kaynaklardan sorumlu bir şekilde faydalanın kodu kaçının, uygulamanız . istenenüretmektenBizim de bu tür araçlara olduğunu ve bu tür durumları tespit etmeyi kolaylaştırdığını .erişimimiz unutmayın

Sunumlar benzerlik kontrolüne . tabi tutulacaktırBenzerlik kontrolünden geçemeyen alt görevler notlandırılmayacak ve akademik dürüstlük ihlali vakası olarak etik kurula bildirilecek, bu da ilgili öğrencilerin .uzaklaştırılmasına neden olabilecektir

Referanslar

[1] "Sıralama algoritması." Wikipedia, https://en.wikipedia.org/wiki/Sorting_algorithm, Son Erişim: 10/02/2022.

[2] N. Faujdar ve S. P. Ghrera, "Analysis and Testing of Sorting Algorithms on a Standard Dataset," 2015 Fifth International Conference on Communication Systems and Network Technologies, 2015, s. 962-967.

[3] G. Batista, "Big O," Towards Data Science, 5 Kasım. 2018, https://towardsdatascience.com/big-o-d13a8b1068c8 , Son Erişim: 10/03/2024.