KAHRAMANMARAŞ SÜTÇÜ İMAM ÜNİVERSİTESİ

MÜHENDİSLİK MİMARLIK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2019-2020 BAHAR DÖNEMİ

ALGORİTMA ANALİZİ VE TASARIMI DERSİ

VİZE PROJE RAPORU

SIRALI, TERSDEN SIRALI VE FARKLI RANDOM DİZİLER İLE

INSERTION SORT VE MERGE SORT

İÇİN KARMAŞIKLIĞIN HESAPLANMASI VE KARŞILAŞTIRILMASI

Hazırlayan

EBRU ŞİMŞEK

18110131003 (ebrusimsek201@gmail.com)

Ders Öğretim Üyesi

Dr.Öğr.Üyesi Zeynep Banu Özger (zeynepozger@ksu.edu.tr)

Proje Hakkında

Bu çalışmada verilen proje konularında konu 1 seçilmiştir. Projede C kodu ile 5 dizi hazırlamamız, ve bunları seçilen 2 sıralama algoritması ile sıralayıp, karmaşıklığın hesaplanması ve karşılaştırılması istenmektedir. Bu sebeple algoritmalardan ,en az biri böl yönet algoritması olmalı şartına dikkat edilerek Insertion sort ve Merge sort algoritma olarak seçilmiştir. Sık kullanılan algoritmaların karmaşıklıklarını görebilmek için bu algoritmalar seçilmiştir. .

1. Proje için öncelikle istenen diziler hazırlanmıştır.
   İstenen dizi özellikleri
- 3 tanesi random değerlerden oluşan dizi.(100, 500 ve 1000 elemanlı)
- 1000 elemanlı sıralı dizi
- 1000 elemanlı tersten sıralı dizi

2. Sonrasında sıralama yapabilmek için sıralama algoritmaları içeren fonksiyonlar yazılmıştır.
3. Karmaşıklık hesaplama fonksiyonları yazılmıştır.
4. Sonrasında Menü oluşturulmuştur.
5. Fonksiyonlar menu içerisinde çağırılmıştır.
6. Algoritmaların yürütme zamanlarını hesaplamak için clock() fonksiyonu kullanılmıştır.

Yorumlarken sıralı dizi en iyi durum olacağından best case, tersden sıralı durum en kötü durum olacağından worst case durumları incelenmiştir.Sıralı ve tersden sıralı diziler karmaşıklığa sıralanmışlığın etkisini göstermektedir. Random diziler rastgele sıralanmış elemanlar içerdiğinden ortalama durum (average case) belirtmektedir. Boyutları farklı olduğundan karmaşıklığa boyut artmasının etkisini göstermektedir.

ZAMAN KARMAŞIKLIĞI

Zaman karmaşıklığı, sıralama problemlerinde en verimli algoritma ile sıralamayı yapmak için algoritmalarda izlenen adımların, dizi boyutu parametresi ile bir fonksiyonla ifade edilmesidir. Örneğin n boyutlu bir dizi verildiğinde n adımda sıralama yapılıyorsa, n zaman karmaşıklığına sahiptir.

Algoritmanın kaç adımda hesaplanacağı, algoritmanın çalıştırıldığı makineye veya kodlandığı dile de bağlıdır. Ama bunlardan bağımsız olarak genelleme yapabilmek için Big O (best case), Big Ω (worst case), Big θ (average case) gibi asimptotik fonksiyonlar yardımıyla hesaplamalar yapılır.

Bir algoritmanın çalışma süresine running time denir. T(n) fonksiyonu ile gösterilir

T(n)'in büyüme oranı algoritmanın hesaplama karmaşıklığıdır. Bu büyüme oranına erki eden bazı faktörler vardır. Eleman sayısı, zaten sıralı olma yada olmama durumu gibi. T(n) üyüme oranı hesaplanırken Big O (best case), Big Ω (worst case), Big θ (average case) gibi asimptotik fonksiyonlar kullanılır.

Gerçek yürütme zamanı=

$$T(n)+C(n)$$

T(n): algoritmadan gelen maliyet,

C(n):İşlemci iş zamanı)

İşlemci iş zamanı bilinemeyeceğinden göz ardı edilir, karmaşıklık T(n) üzerinden değerlendirilir. Bu nedenle T(n) hiçbir zaman gerçek yürütme zamanını vermez. Bu sebeple yaklaşık değerler bulmak için üst ve alt sınırlar belirlenir. Yani karmaşıklık, algoritmanın çalışma süresini

temsil eder. Böylece verimlilik yorumlanabilir. Verimli algoritma mümkün olduğu kadar düşük büyüme oranına yani düşük karmaşıklığa sahip algoritmadır..

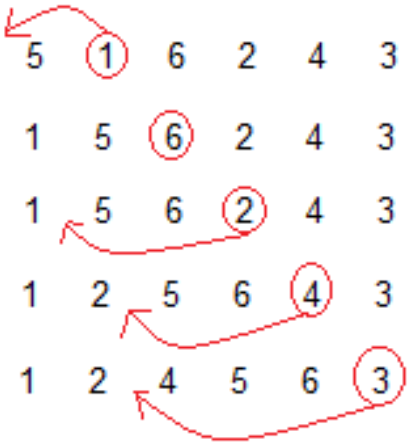
SIRALAMA ALGORİTMALARI

Sıralama artan veya azalan sırada düzenleme işlemidir.Karşılaştırma tabanlı yada doğrusal zamanlı sıralama algoritmaları vardır. Bu çalışmada karşılaştırma tabanlı olan iki sıralama algoritması incelenmiştir.

Sıralama algoritmalarda, N elemanlı bir dizi için **N-1** defa işlem tekrarlanıp karşılaştırma yapılarak dizi sıralanır. Örneğin 100 eleman için 99 kez karşılaştırma yapılır.

INSERTION SORT

Karşılaştırma tabanlı sıralama algoritmasıdır.Insertion sort da sıralı ve sırasız olarak iki dizi varmış gibi farz edilir. Sıralı dizi ilk başta boş kümedir. Sırasız dizide ise verilen sıralanacak dizi bulunur.Her adımda sırasız diziden sırasıyla bir eleman alınıp karşılaştırılarak sıralı dizide uygun yere yerleştirilir.



Insertion sort da karşılaştırma ve yer değiştirme adımları vardır. Elemanlar zaten doğru sırada ise yer değiştirmeye gerek kalmaz. Sadece karşılaştırma yapılır. Elemanların karışıklığı arttıkça yer değiştirme miktarıda artar.

Maliyet hesabı

Best case durumunda zaten sıralı dizi gelir. Sıralı dizi için sadece karşılaştırma yapılır.Yer değiştirme olmaz. Bu sebeple karmaşıklığı O(n) dir.

1.adım (n-1) karşılaştırma +0 yer değiştirme=(n-1)

2.adım (n-2) karşılaştırma +0 yer değiştirme=(n-2)

…

(n-1). adım 1 karşılaştırma + 0 yer değiştirme=1

Karmaşıklık **O(n)**

Worst case durumunda ters sıralanmış dizi gelir.Karşılaştırma ve yer değiştirme olur. Bu sebeple karmaşıklığı $O(n^2)$ olur.

Average case durumunda rastgele sıralı bir dizi gelir. Karşılaştırma ve yer değiştirme olur. Bu sebeple karmaşıklığı $O(n^2)$ olur.

1.adım (n-1) karşılaştırma +(n-1) yer değiştirme=2(n-1)

2.adım (n-2) karşılaştırma +(n-2) yer değiştirme=2(n-2)

…

(n-1). adım 1 karşılaştırma + 1 yer değiştirme=2

Karmaşıklık $O(n^2-n)=\mathbf{O(n^2)}$

```
void insertionSort(int arr[], int n) {
int i, key, j;
for (i = 1; i < n; i++)   {  // ----------------------------------→O(n)
        key = arr[i]; //-------------------------------------→O(n-1)
        j = i - 1;     // -------------------------------------→O(n-1)
        while (j >= 0 && arr[j] > key) { // --------------→O(n*(n+1)/2)

                arr[j + 1] = arr[j];  // ----------------------→O((n-1)*n/2)
                j = j - 1;  // ---------------------------------→O((n-1)*n/2)
}
        arr[j + 1] = key; //----------------------------------→O(n-1)
} }
```

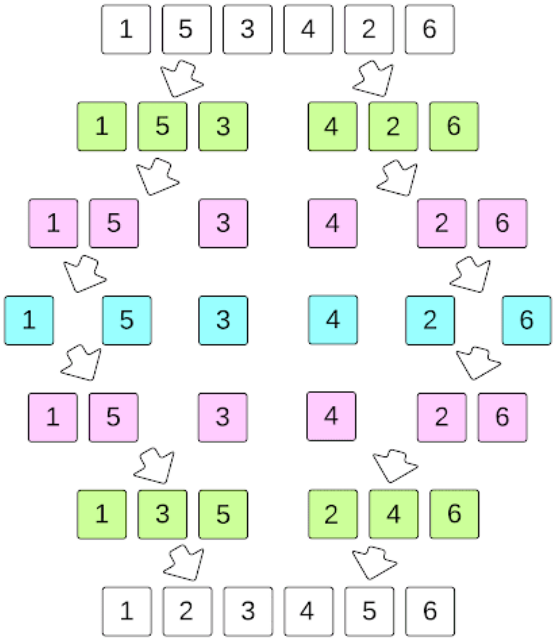
+

Toplam $n+(n-1)+(n-1)+(n^2+n)/2)+ (n^2-n)/2)+ (n^2-n)/2) +(n-1)=(3n^2+7n-6)/2$

Karmaşıklık $O((3n^2+7n-6)/2)=\mathbf{O(n^2)}$

MERGE SORT

Karşılaştırma tabanlı sıralama algoritmasıdır. Böl Yönet sistemi ile recursive çalışır. Öncelikle sıralanacak dizi iki parçaya bölünür. Fonksiyon kendisini sağ ve sol alt kümeler için 2 defa çağırır.Çağırma işlemi alt dizide 2 eleman kalana kadar devam eder.' Eleman kaldığında önce bunları kendi aralarında karşılaştırarak sıralar. Daha sonra bu alt dizileri büyüklüklerini göz önüne alarak birleştirir ve final sıralanmış dizi oluşur.

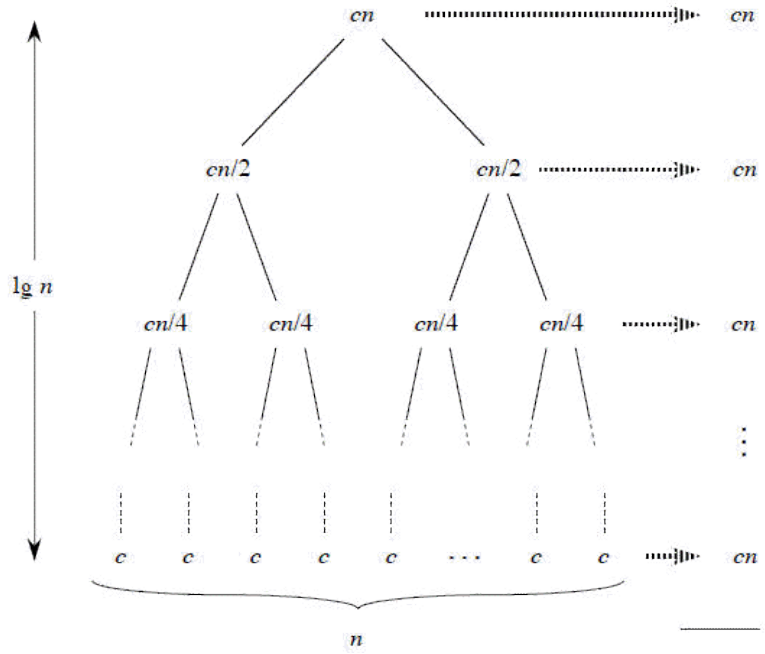


Maliyet Hesabı

T(n)=T(n/2)+ T(n/2)+ cn

T(n)=T(n/2)(sağ bölmeden gelen maliyet)+T(n/2)(sol bölmeden gelen maliyet)+cn(n eleman birleştirme)

Fonksiyon kendisini her alt dalda tekrar çağırdığı için boyut önce yarıya sonra onun yarısına düşer en son 2 eleman kalana kadar bu böyle devam eder. Bu sebeple maliyetde boyut ile birlikde alt dal için yeniden hesaplanır.Seviye seviye alt dallara ayrıldığı düşünülürse her seviyede maliyet cn dir.



Merge sort da karşılaştırma ve yer değiştirme adımları tüm elemanlara uygulanır. O yüzden maliyet hesabında ayrı ayrı hesaplanmaz.

1.adım sağ ve sol toplam (karşılaştırma+yer değiştirme)= cn

2.adım sağ ve sol toplam (karşılaştırma+yer değiştirme)= c(n/2)+c(n/2)=c(n)

…

i. adım sağ ve sol toplam (karşılaştırma+yer değiştirme)= c(n/i)+ c(n/i)+… + c(n/i)= c(n)

toplam level sayısı (1+i) toplam maliyet=cn*(1+i) idi

2^i=n i=logn old level sayısı (1+logn) alınırsa;

toplam maliyet=cn*(1+logn) olur.

toplam maliyet=c*(n+nlogn)

Karmaşıklık O(nlogn) bulunur.

```
void mergeSort(int arr[], int l, int r) {
  if (l < r) {
        int m = l+(r-l)/2;     //--------→1
        mergeSort(arr, l, m);  // ----------→O(n/2)
```

sağ dal kendini çağırır O(n/4)+O(n/4) onlarda kendini çağırır O(n/8)+O(n/8)+ O(n/8)+O(n/8) ……..son 2 eleman kalana kadar çağırır böylece her seviyeden O(n/2) gelir.

```
        mergeSort(arr, m+1, r); //--------→O(n/2)
```

sol dal kendini çağırır O(n/4)+O(n/4) onlarda kendini çağırır O(n/8)+O(n/8)+ O(n/8)+O(n/8) ……..son 2 eleman kalana kadar çağırır. böylece her seviyeden O(n/2) gelir.

Sağdan ve soldan gelenler toplanırsa her seviyede maliyet O(n) olur.

i level varsa 2^i tane alt dala ayrılmaktadır.Maliyetde $n/2^i$ ye bölünmektedir. Sistem recursive çalışır. toplam level sayısı (1+i) .

```
        merge(arr, l, m, r);       // ----------→O(n) ayrılan alt dalların birleştirilmesi i. seviyede geri birleştirilip sıralı dizi elde edilir.
        }
}
```

\+

Toplam cn*(1+i)

2^i=n i=logn old level sayısı (1+logn) alınırsa;

Toplam cn*(1+logn)

Toplam c*(n+nlogn)

Karmaşıklık O(n+nlogn)=**O(nlogn)**

UYGULAMA BULGULARI

SIRALI DİZİ

Best case yani en iyi durum olup sıralamanın en kolay olduğu 1000 sıralı elemana sahiptir.

Insertion sort

```
Running Time=97.000000

Insertion sort karmasikligi 1000.000000
```

Merge sort

```
Running Time=1532.000000

Merge sort karmasikligi 9965.784180
```

TERSTEN SIRALI DİZİ

Worst case yani en kötü durum olup sıralamanın en zor olduğu 1000 karışık elemana sahiptir.

Insertion sort

```
Running Time=1830.000000

Insertion sort karmasikligi 1000000.000000
```

Merge Sort

```
Running Time=1358.000000

Merge sort karmasikligi 9965.784180
```

100 ELEMANLI RANDOM DİZİ

Average Case yani ortalama durup ortalama karışıklıkta 100 elemana sahiptir.

Insertion sort

```
Running Time=11.000000

Insertion sort karmasikligi 10000.000000
```

Merge sort

```
Running Time=12.000000

Merge sort karmasikligi 664.385620
```

500 ELEMANLI RANDOM DİZİ

Average Case yani ortalama durup ortalama karışıklıkta 500 elemana sahiptir.

Insertion sort

```
Running Time=643.000000

Insertion sort karmasikligi 250000.000000
```

Merge sort

```
Running Time=661.000000

Merge sort karmasikligi 4482.892090
```

1000 ELEMANLI RANDOM DİZİ

Average Case yani ortalama durup ortalama karışıklıkta 1000 elemana sahiptir.

Insertion sort

```
5 985 986 986 991 993 995 995 995 995 996 996 997

Running Time=2177.000000

Insertion sort karmasikligi 1000000.000000
```

Merge sort

```
5 985 986 986 991 993 995 995 995 995 996

Running Time=1909.000000

Merge sort karmasikligi 9965.784180
```

Running Time ile Karmaşıklık Analizi

Algoritmayı yürütme zamanlarına bakacak olursak;

	Özellik	Eleman sayısı	INSERTION SORT RUNNING TİME	MERGE SORT RUNNING TİME
Best case	SIRALI	1000	97 ms (0.097s)	1532 ms (1.532 s)
Worst case	TERS SIRALI	1000	1830 (0,183 s)	1358 ms (1.358 s)
Average case	RANDOM	100	11 ms (0.011 s)	12 ms (0.012 s)
Average case	RANDOM	500	643 ms (0.643 s)	661 ms (0.661 s)
Average case	RANDOM	1000	2177 ms (2.177 s)	1909 ms (1.909 s)

Insertion sort, SIRALI durumda TERS SIRALI dan daha hızlı işlem görmüştür. RANDOM durumlarda boyut arttıkça süre uzamıştır.

Merge sort, SIRALI VE TERS SIRALI durumlarda hız olarak çok fark yaratmamıştır. RANDOM durumda boyut arttıkça süre uzamıştır.

Anlık yürütme zamanını etkileyen birtakım faktörler vardır. Yürütme zamanındaki büyüme oranını gösterebilmek için eklenmiştir. Bu çalışmada 2:53 GHz Intel Core i5-M460 işlemci ve 3GB RAM bellege sahip sistem üzerinde Dev C++ Version 5.11 ˘programı kullanılmıştır.

Genellemeler ile Karmaşıklık Analizi

	Özellik	Eleman sayısı	Formül	INSERTİON SORT karmaşıklığı	Formül	MERGE SORT karmaşıklığı
Best case	SIRALI	1000	$\Omega(n)$	1 000	$\Omega(nlogn)$	9 965
Worst case	TERS SIRALI	1000	$O(n^2)$	1 000 000	O(nlogn)	9 965
Average case	RANDOM	100	$\theta(n^2)$	10 000	θ(nlogn)	664
Average case	RANDOM	500	$\theta(n^2)$	250 000	θ(nlogn)	4 482
Average case	RANDOM	1000	$\theta(n^2)$	1 000 000	θ(nlogn)	9 965

SIRALI dizi insertion sort karmaşıklığı Ω(n) dir. Algoritmanın çalışma süresindeki artış lineerdir. Çalışma zamanındaki artış, yani karmaşıklık, input büyüklüğündeki artış ile doğru orantılıdır.

SIRALI dizi, TERSDEN SIRALI dizi ve RANDOM sıralı dizi merge sort karmaşıklığı Ω(nlogn), O(nlogn), θ(nlogn) dir. Problem alt programlara ayrılarak çözülüp sonra çözümler geri birleştiriliyor olmasından dolayı çalışma zamanındaki artış, yani karmaşıklık, input büyüklüğündeki artış ile logaritmasının çarpımı kadar oranla artar.

TERSDEN SIRALI dizi ve RANDOM SIRALI dizi insertion sort karmaşıklığı $O(n^2)$, $\theta(n^2)$dir. Yani aynıdır. Algoritmanın çalışma süresi quadratic (ikinci dereceden)dir. Her biri n defa çalışan iç içe 2 döngünün çalışma süresini verir.Bu sebeple çalışma zamanındaki artış, yani karmaşıklık, input büyüklüğündeki artışın karesi kadar oranla artar

Karşılaştırma yapıldığında

SIRALI diziler best case durumu olup, Insertion Sort yerine yerleştirmesine gerek kalmadığı için Merge Sort dan daha iyi çalışmaktadır. Dizinin sıralı olması Merge Sort karmaşıklığını etkilememektedir.

TERS SIRALI DİZİLER worst case durumu olup Merge Sort daha iyi çalışmaktadır.

100 ELEMANLI RANDOM DİZİLERDE mükemmel bir fark olmamakla birlikte Merge Sort daha iyi çalışmaktadır.

500 ELEMANLI RANDOM DİZİLERDE Merge Sort daha iyi çalışmaktadır.

1000 ELEMANLI RANDOM DİZİLERDE Merge Sort daha iyi çalışmaktadır.

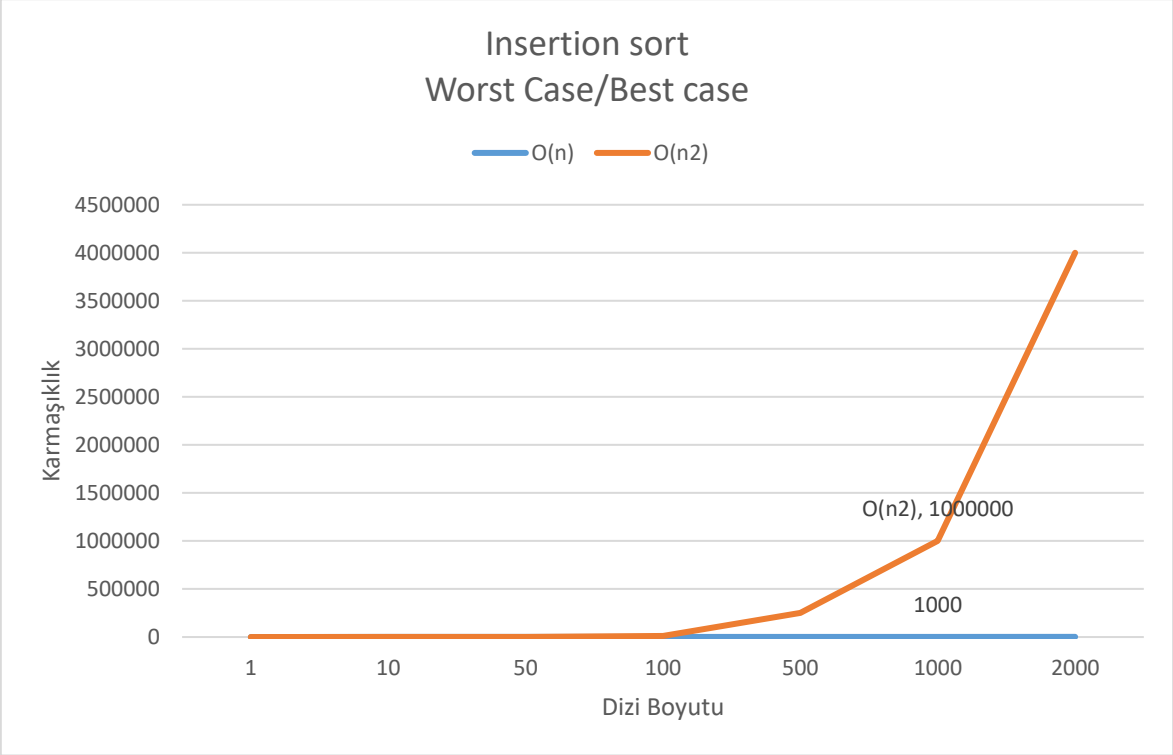
Insertion sort için eleman sayısı arttıkça karmaşıklığı oldukça artmıştır, bu da küçük boyutlu dizilerde daha iyi çalıştığını göstermektedir.

Merge Sort için eleman sayısı arttıkça karmaşıklığı biraz artmıştır, bu da boyutun çok mükemmel bir fark yaratmadığını göstermektedir.

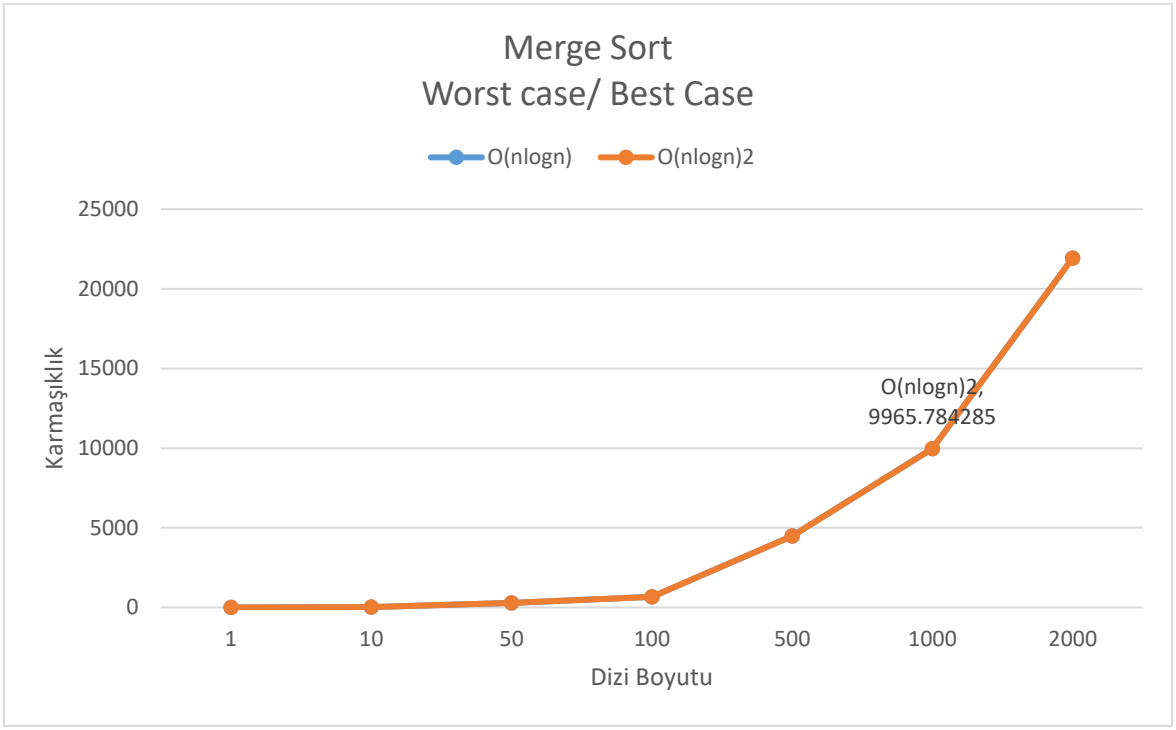
Hem insertion hem merge sort için **ters sıralı ve random dizi** karmaşıklığının aynı olduğu görülmektedir.

SIRALANMIŞ OLMANIN KARMAŞIKLIĞA ETKİSİ

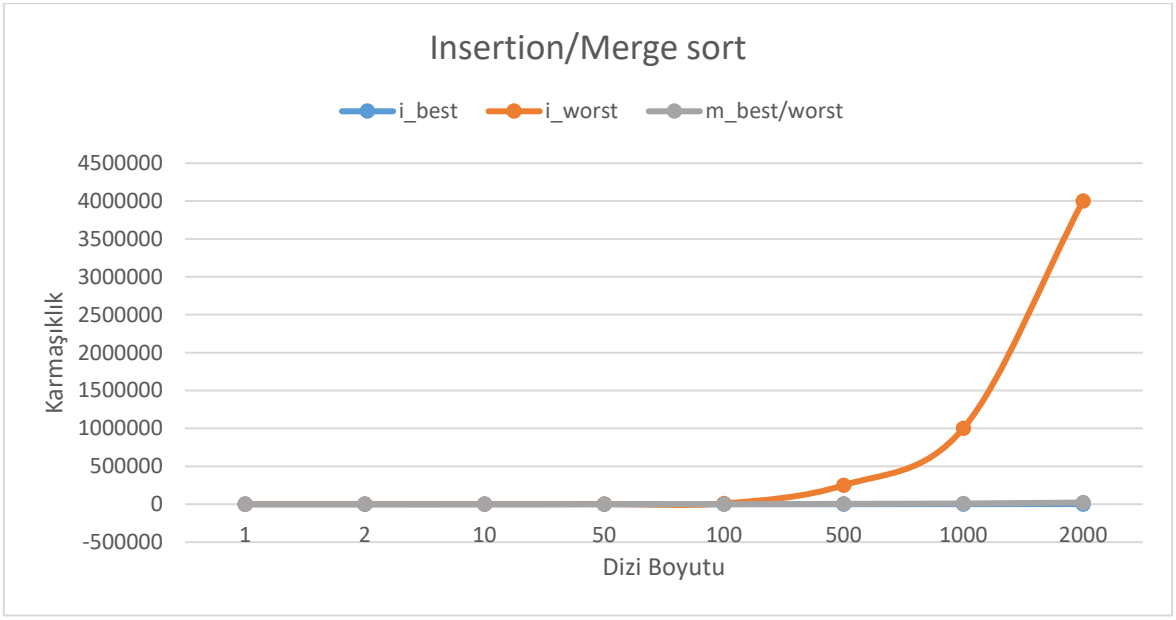
Sıralanmış olmak dizideki elemanların karışıklığıyla alakalıdır.



Insertion sort için best case $\Omega(n)$, worst case $O(n^2)$ bazı dizi boyutlarında hesaplanarak grafiği çizilmiştir. Yukardaki grafiktende anlaşılacağı gibi best case en alt seviyede lineer fonksiyon oluştururken, worst case logaritmik artmıştır. Sıralanmış olmak karmaşıklığı azaltırken, ne kadar sıra bozuk olursa karmaşıklık artmaktadır. 1000 elemanlı dizi grafikte belirtilmiştir. Sıralanmış dizide sıralanmış elemana algoritmanın bazı adımları uygulanmamaktadır.



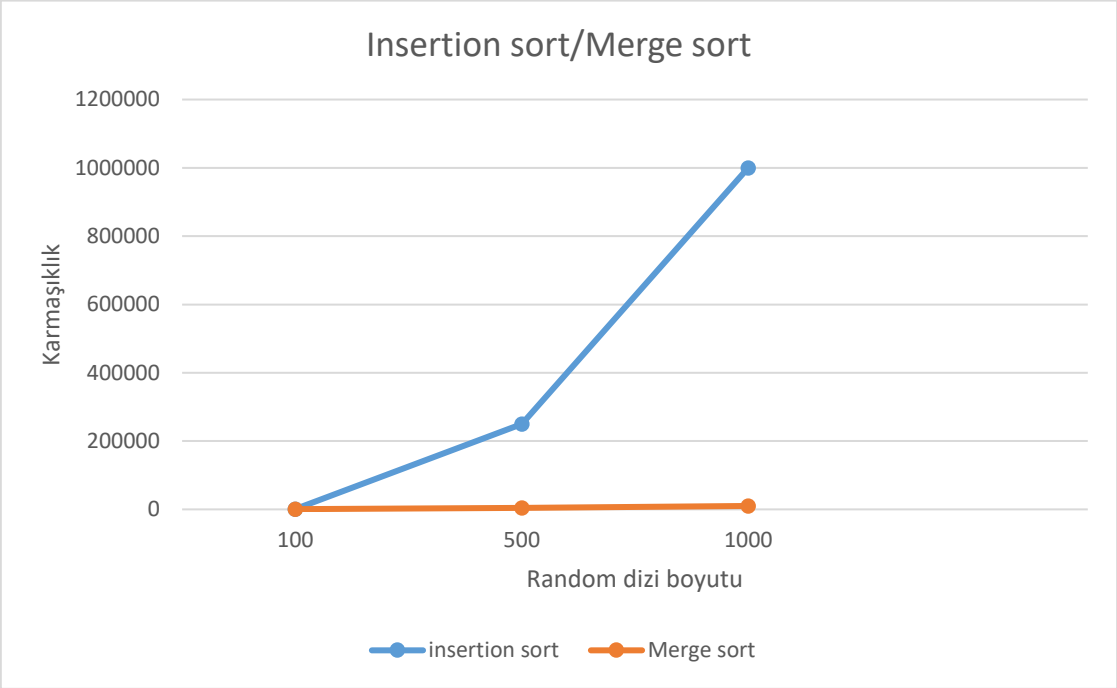
Merge sort için best case Ω(nlogn), worst case O(nlogn) bazı dizi boyutlarında hesaplanarak grafiği çizilmiştir. Yukardaki grafiktende anlaşılacağı gibi iki durumdada fonksiyonlar üst üste binmiştir. 1000 elemanlı dizi grafikte belirtilmiştir. Sıralanmış olmak yada olmamak merge sortta karmaşıklığı etkilememektedir. Algoritma tüm adımları her elemana uygulamaktadır.



Sıralı durumlara göre insertion sort, merge sort karşılaştırıldığında worst case durumunda insertion sort daha karmaşıktır. Best case durumunda ise yukarıdaki grafiktende anlaşılacağı gibi belirgin fark olmamakla birlikte(fonksiyonlar neredeyse üst üste) merge sort biraz daha karmaşıktır. Ama sıra bozuldukça merge sort karmaşıklığı değişmezken insertion sort çok fazla karmaşıklaşmaktadır.

BOYUT ARTMASININ KARMAŞIKLIĞA ETKİSİ

100, 500 ve 1000 boyutlu diziler için average case insertion ve merge sorta göre hesaplanıp grafiği çizilmiştir. Grafikten anlaşılacağı gibi boyut artımı merge sort da önemli bir değişiklik oluşturmazken, insertion sort da karmaşıklık ciddi oranda artmıştır.



Sonuç

Insertion sort sıralı ve küçük boyutlu dizilerde daha verimli iken, Merge sort büyük boyutlu dizilerde daha verimlidir.Sıralama Insertion sort da elemanlara uygulanan algoritma adımlarını azaltırken, merge sort da değiştirmemektedir.

Kaynaklar

1. Ders notları
2. Insertion sort

https://www.studytonight.com/data-structures/insertion-sorting

3. Merge sort

https://www.studytonight.com/data-structures/merge-sort

4. Sıralama karşılaştırması

https://medium.com/@halisak_/s%C4%B1ralama-algoritmalar%C4%B1n%C4%B1n-kar%C5%9F%C4%B1la%C5%9Ft%C4%B1r%C4%B1lmas%C4%B1-dae2a88de6b

5. Sıralama algoritmaları

https://www.toptal.com/developers/sorting-algorithms

6. Running Time

https://www.geeksforgeeks.org/how-to-measure-time-taken-by-a-program-in-c/

7. Sıralama algoritmaları için karmaşıklık formülleri

https://www.geeksforgeeks.org/time-complexities-of-all-sorting-algorithms/