

YILDIZ TEKNİK ÜNİVERSİTESİ

ELEKTRİK-ELEKTRONİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ

2014-2015 ÖĞRETİM YILI GÜZ YARIYILI

VERİ YAPILARI VE ALGORİTMALAR

ÖDEV-2

( BLM-2512/ GRUP:1)

**Hazırlanan Anabilim Dalı**

**Bilgisayar Bilimleri Anabilim Dalı**

**Hazırlayan**

Mert Sevil

09013057

Bilgisayar Mühendisliği Lisans Programı

**Öğretim Üyesi**

Prof. Dr. M. Yahya KARSLIGİL

**İSTANBUL, 2014**

**İçindekiler**

1. Ödevin amacı, tanıtımı ve giriş………………………………………………………
2. Sıralama algoritmaları…………………………………...…………………………
   1. Bubble Sort………………………………………………………………….
   2. Quick Sort……………………………………………………………………
   3. Selection Sort…………………………………………………………………
   4. Lineer Sort…………………………………………………………………..
   5. Insertion Sort………………………………………………………………..
   6. Merge Sort……………………………………………………………………….
3. Ödevin gerçekleştirilmesi……………………………………………………………...8
   1. Kod için algoritmanın çizilmesi………………………………………
   2. Kodun yazılması………………………………………………………8,9,10,11
   3. Çıktıların elde edilmesi ve algoritma analizi…………………………..12,13,14
   4. Sonuçların yorumlanması…………………………………………………….14
4. Kaynakça……………………………………………………………………………...14

**1. Ödevin amacı, tanıtımı ve giriş**

**Ödevin amacı:**

* Ödev sıralama algoritmalarını tanımayı hedeflemektedir. Bu kapsamda 1 sıralama algoritması detaylı olarak (Merge Sort), 5 sıralama algoritması ise temel hatlarıyla incelenmiştir.
* Ödev C programla dilinde gerçekleştiği için C programlama dilinde kod geliştirebilme kabiliyetini artırmayı hedefler
* Algoritma çizebilmeyi ve çizilen algoritmayı bir programlama dilinde kodlayabilme kabiliyetinin geliştirilmesi
* Program çıktısı olarak üretilen sonuçların yorumlanabilmesi ve analizinin yapılabilmesi yeteneğinin geliştirilmesi
* Gereksiz döngü ve dizi kullanmadan algoritma tasarlayabilme yeteneğinin geliştirilmesi
* Bir probleme çok yönlü bakabilme kabiliyetinin geliştirilmesi (Bir sorting işleminin 6 açıdan gözlemleyebilmek (6 farklı sort algoritması))
* İleriki algoritma konularına temel teşkil edebilmek

**Ödevin Tanıtımı ve Giriş**

Sıralama algoritmaları günlük yaşamamızdaki pekçok problemin temel konusudur. Bu problemler çoğu zaman bilgisayar desteğiyle çözümlenebilmektedir. Örneğin 70 milyon nüfuslu Türkiye’nin vatandaşlarının TC kimlik numaralarının sıralı olarak kaydedilmesi ve sıralanması bir gerçek hayat problemidir. Bu kadar çok veriyi sıralı olarak tutabilmek ve sıralamak ancak bilgisayarlar ile mümkün olabilmektedir.

Bu açıdan bu önemli gerçek hayat problemlerine konu olabilecek konuda tasarlanacak algoritmalar tasarım yapılan duruma uygun hızlı, efektif ve akılcı algoritmalar olmalıdır. Algoritmalar da genelde iki durumu optimize etmek önemlidir. Bunlar mekan ve zaman problemleridir. Bu açıdan sort işlemenin yapılacağı uygulamaya göre ya yerden ya zamandan tasarruf etmek için en uygun algoritmik akış seçilir ve kodlanır.

Örneğin bellek birimi sınırlı bir gömülü sistem uygulanmasında zaman ikinci planda yer alıyorsa böyle bir durumda ikinci dizi kullanmayan ancak yavaş bir sort algoritması seçilebilir. Buna karşın gerçek zamanlı bir bilgisayar uygulamasında zaman birinci öncelik olduğu için gerekirse ikinci bir dizi kullanılması da göze alınarak karmaşıklığı en az olan algoritma seçilmelidir.

Bu açıdan uygulama tasarlanacak algoritma ile doğrudan ilişkilidir. Ancak en başta sort algoritmalarının neler olduğunu bilmek tasarımdan önceki en temel başlangıç noktası sayılır. Bu ödev temelde bahsi geçen bu durumu kavramak ve sort algoritmaları ile ilgili kapsamlı bir araştırma yapmayı hedefler. Konu sort algoritmaları olarak görünse de temelde ödev düzgün ve efektif çalışan algoritma nasıl çalışır sorusuna yanıt aradığı için ileriki konular için köşe taşı görevi üstlenmektedir.

**2. Sorting Algoritmaları**

Bu bölümde 6 farklı sorting algoritması ele alınmıştır. İlk 5 sorting algoritması kısaca incelenmiş olunup, 6. algoritma olan Merge algoritması konumuz olması vesilesiyle detaylı olarak analiz edilmeye çalışılmıştır.

**2.1 Bubble Sort**

Tasarımı en kolay sıralama algoritması olarak bilinir. Bubble sort Türkçe karşılığıyla kabarcık sıralaması olarak bilinir. Kabarcık Sıralaması, bilgisayar bilimlerinde kullanılan yalın bir sıralama algoritmasıdır. Sıralanacak dizinin üzerinde sürekli ilerlerken her defasında iki öğenin birbiriyle karşılaştırılıp, karşılaştırılan öğelerin yanlış sırada olmaları durumunda yerlerinin değiştirilmesi mantığına dayanır. Algoritma, herhangi bir değişiklik yapılmayıncaya kadar dizinin başına dönerek kendisini yineler. Adına "Kabarcık" sıralaması denmesinin nedeni büyük olan sayıların aynı suyun altındaki bir kabarcık gibi dizinin üstüne doğru ilerlemesidir.[1]

Başlangıçta yer yer değiştirme sıralaması olarak adlandırılan kabarcık sıralaması, dizi içindeki büyük elemanların algoritmanın her adımında dizinin sonuna doğru doğrusal olarak ilerlemesini sağlar. Bu ilerleme, seçmeli sıralama algoritmasındaki dizideki değeri küçük olan elemanların dizinin başında kümelenmesi yöntemine benzer şekilde gerçekleşir.[1]

İçeriği "5 1 4 2 8" olan bir dizi kabarcık sıralaması ile en küçükten en büyüğe doğru aşağıdaki biçimde sıralanır. Her adımda dizinin kalın olarak işaretlenmiş elemanları karşılaştırılan elemanlardır. [1]

Birinci Geçiş:

( **5 1** 4 2 8 ) ->( **1 5** 4 2 8 ) Burada algoritma ilk iki elemanı karşılaştırır ve yerlerini değiştirir.

( 1 **5 4** 2 8 ) -> ( 1 **4 5** 2 8 )

( 1 4 **5 2** 8 ) -> ( 1 4 **2 5** 8 )

( 1 4 2 **5 8** ) -> ( 1 4 2 **5 8** )

Burada elemanlar zaten sıralı olduğu için algoritma yerlerini değiştirmez.

İkinci Geçiş:

( **1 4** 2 5 8 ) ->( **1 4** 2 5 8 )

( 1 **4 2** 5 8 ) -> ( 1 **2 4** 5 8 )

( 1 2 **4 5** 8 ) -> ( 1 2 **4 5** 8 )

( 1 2 4 **5 8** ) -> ( 1 2 4 **5 8** )

Artık dizi sıralıdır ancak algoritma işlemin bittiğini bilmemektedir. Algoritmanın dizinin sıralandığını anlaması için bütün dizinin üzerinden hiçbir değişiklik yapmadan tam bir geçiş yapması gerekir.

Üçüncü Geçiş:

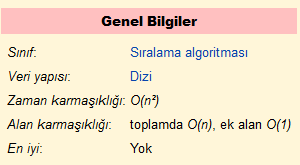
( **1 2** 4 5 8 ) -> ( **1 2** 4 5 8 )

( 1 **2 4** 5 8 ) -> ( 1 **2 4** 5 8 )

( 1 2 **4 5** 8 ) -> ( 1 2 **4 5** 8 )

( 1 2 4 **5 8** ) -> ( 1 2 4 **5 8** )

Sonuç olarak dizi sıralanmıştır ve algoritma sonlanır. [1]



**2.2 Quick Sort**

Hızlı Sıralama (İngilizcesi: Quicksort) günümüzde yaygın olarak kullanılan bir sıralama algoritmasıdır. Hızlı sıralama algoritması n adet sayıyı, ortalama bir durumda, {O}(n~log(n)) karmaşıklığıyla, en kötü durumda ise {O}(n^2) karmaşıklığıyla sıralar. Algoritmanın karmaşıklığı aynı zamanda yapılan karşılaştırma sayısına eşittir. [2]

Hızlı sıralama algoritması, sıralanacak bir sayı dizisini daha küçük iki parçaya ayırıp oluşan bu küçük parçaların kendi içinde sıralanması mantığıyla çalışır.

Algoritmanın adımları aşağıdaki gibidir:

* Sayı dizisinden herhangi bir sayıyı pivot eleman olarak seç.
* Sayı dizisini pivottan küçük olan tüm sayılar pivotun önüne, pivottan büyük olan tüm sayılar pivotun arkasına gelecek biçimde düzenle (pivota eşit olan sayılar her iki yana da geçebilir). Bu bölümlendirme işleminden sonra eleman sıralanmış son dizide olması gerektiği yere gelir. Algoritmanın bu aşamasına bölümlendirme aşaması denir.
* Pivotun sol ve sağ yanında olmak üzere oluşan iki ayrı küçük sayı dizisi, hızlı sıralama algoritması bu küçük parçalar üzerinde yeniden özyineli olarak çağrılarak sıralanır.

Algoritma içinde sayı kalmayan (eleman sayısı sıfır olan) bir alt diziye ulaştığında bu dizinin sıralı olduğunu varsayar. [2]

**2.3 Selection Sort**

**2.4 Lineer Sort**

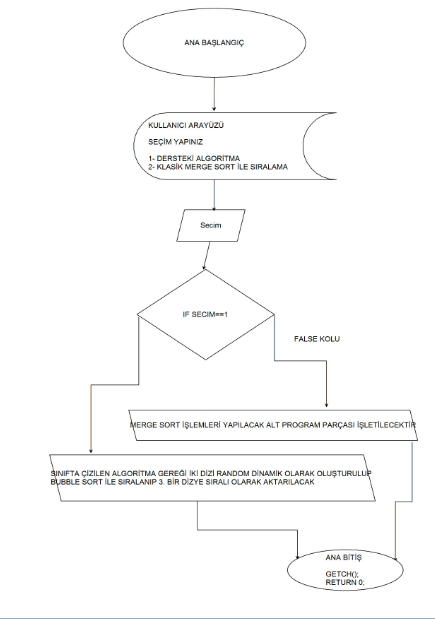
**2.5 Insertion Sort**

**2.6 Merge Sort**

**3.1 Kod için algoritmanın çizilmesi**

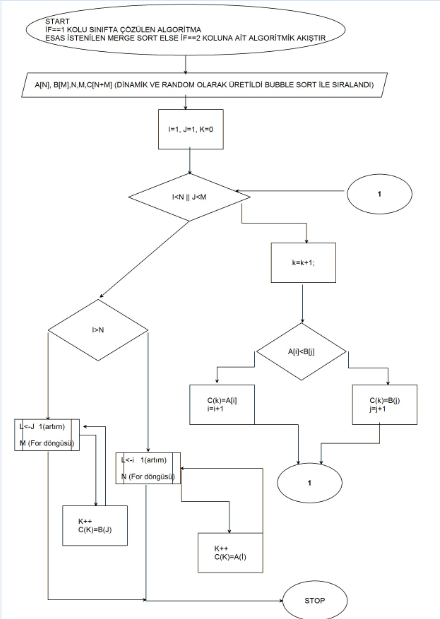
Algoritma için iki durum düşünülmüştür. İlk olarak kullanıcı arayüzü üzerinden 2 alternatif kullanıcıya sunulacaktır. Bu alternatiflerden 1’i derste yapıldığı gibi iki random dizi üretilerek bu dinamik iki diziyi bubble sort ile sıralamak ve akabinde sırası bozulmadan 3. Bir dizide birleştirmektir.

Şekil- de ana start ve programın kullanıcıya gelen arayüzü teşkil edilmiştir.



Şekil- Programın Ana Başlangıç ve Bitişine Ait Algoritmik Akış

Şekil- de kullanıcının 1’i seçmesi durumuna karşılık olan dersteki algoritmik örneğimize ait algoritmik akış gözlemlenmektedir.

****

Şekil- Kullanıcının 1’ basması ve dersteki algoritmik akışın oluşturulması

**3.2 Kodun yazılması**

Kod C programlama dilinde yazılıp DEV C++ derleyicisi kullanılmıştır.

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

void mergesort(int dizi[], int l, int h);

int main(){

int secim;

printf("Asagidaki duruma gore seciminizi yapiniz\n");

printf("1-Derste isledigimiz algoritmanin gerceklesmesi icin 1-e basin\n");

printf("2-Tek bir dizi uzerinde klasik Merge Sort algoritmasinin gerceklestirilmesi icin 2-ye basin\n");

scanf("%d",&secim);

if(secim==1){ ///DERSTEKİ ALGORİTMANIN GERÇEKLENMESİ

srand((unsigned)time(0));

int i,n,j,m;

int k=0,l;

printf("1. Dizinizin boyutunu giriniz:\n");

scanf("%d",&n); //Maksimum eleman sayısı kullanıcıdan alınarak dinamik dizi için yer ayırılacak

int\* dizi; //

dizi=(int \*)malloc((n\*sizeof(int))); //Dinamik olarak dizi oluşturuldu

printf("\nDiziniz random olarak olusturulacaktir \n");

for(i = 0 ; i < n ; i++ )

dizi[i]=rand()%1000; ///Dizinin elemanlari random olarak atanacaktir

printf("\n");

printf("2. Dizinizin boyutunu giriniz:\n");

scanf("%d",&m); //Maksimum eleman sayısı kullanıcıdan alınarak dinamik dizi için yer ayırılacak

int\* dizi2; //

dizi2=(int \*)malloc((m\*sizeof(int))); //Dinamik olarak dizi oluşturuldu

printf("\n2. Diziniz random olarak olusturulacaktir \n");

for(j=0;j<m;j++)

dizi2[j]=rand()%1000;

int gecici;

for (int i = 1; i < m; i++) DERSTEKİ GİBİ SIRALI BİR DİZİ YAPMAK İÇİN BUBBLE SORT

{

for (int j = 0; j < m-i; j++)

{

if (dizi2[j]>dizi2[j+1])

{

gecici = dizi2[j+1];

dizi2[j+1] = dizi2[j];

dizi2[j] = gecici;

}

}

}

for (int i = 1; i < n; i++) // DERSTEKİ GİBİ SIRALI BİR DİZİ YAPMAK İÇİN BUBBLE SORT

{

for (int j = 0; j < n; j++)

{

if (dizi[j]>dizi[j+1])

{

gecici = dizi[j+1];

dizi[j+1] = dizi[j];

dizi[j] = gecici;

}

}

}

i=1;

j=1;

int C[m+n];

while(i<n || j<m){

k++;

if(dizi[i]<dizi2[j]){

C[k]=dizi[i];

i++;

}

else{

if(dizi[i]>dizi2[j]){

C[k]=dizi2[j];

j++;

}

}

}

if(i>n){

for(l=j;l<m;j++){

k++;

C[k]=dizi2[j];

}

}

else{

for(l=i;l<n;i++){

k++;

C[k]=dizi[i];

}

}

printf("Dizi 1 birlestirilmeden once su sekildeydi (Random idi ancak Bubble sort ile siralandi)\n");

for(i=0;i<n;i++)

printf("%d ",dizi[i]);

printf("\n");

printf("Dizi 2 birlestirilmeden once su sekildeydi(Random idi ancak Bubble sort ile siralandi)\n");

for(j=0;j<m;j++)

printf("%d ",dizi2[j]);

printf("Diziler dersteki gibi bir C dizisinde birlestirildikten sonra C dizisinin son hali\n");

for(j=1;j<k+1;j++)

printf("%d ",C[j]);

printf("\n");

}

else if(secim==2){ //KLASIK BİR MERGE SORT İLE BİR DİZİNİN SIRALANMASI

srand((unsigned)time(0));

int i,n,j;

printf("1. Dizinizin boyutunu giriniz:\n");

scanf("%d",&n); //Maksimum eleman sayısı kullanıcıdan alınarak dinamik dizi için yer ayırılacak

int\* dizi; //

dizi=(int \*)malloc((n\*sizeof(int))); //Dinamik olarak dizi oluşturuldu

printf("\nDiziniz random olarak olusturulacaktir \n");

for(i = 0 ; i < n ; i++ )

dizi[i]=rand()%1000; ///Dizinin elemanlari random olarak atanacaktir

printf("\nSiralamadan once diziniz su sekildedir:\n"); ////Siralamadan once dizinin

for(i = 0; i < n; i++)

printf("%d ", dizi[i]);

printf("\n");

mergesort(dizi, 0, n - 1);

printf("\nMerge sort siralamasi sonrasinda dizinizin yeni durumu\n :"); //Siralandiktan sonra dizinin yeni hali

for(i = 0; i < n; i++)

printf("%d ", dizi[i]);

printf("\n");

}

getch();

return 0;

}

void mergesort(int dizi[], int l, int h)

{

int i = 0;

int length = h - l + 1;

int pivotelemani = 0;

int merge1 = 0;

int merge2 = 0;

int gecicidizi[h+1];

if(l==h) // Cikis kosulu olarak l nin h ye esit olmasi durumu belirlenmistir

return;

pivotelemani = (l + h) / 2; //Pivot eleman l ve h nin aritmatik ortalamasini ifade eder

mergesort(dizi, l, pivotelemani);

mergesort(dizi, pivotelemani + 1, h);

for(i = 0; i < length; i++)

gecicidizi[i] = dizi[l + i];

merge1 = 0;

merge2 = pivotelemani-l+1;

for(i=0;i<length;i++)

{

if(merge2<=h-l)

{

if(merge1<= pivotelemani-l)

{

if(gecicidizi[merge1] > gecicidizi[merge2])

dizi[i + l] = gecicidizi[merge2++];

else

dizi[i + l] = gecicidizi[merge1++];

}

else

dizi[i + l] = gecicidizi[merge2++];

}

else

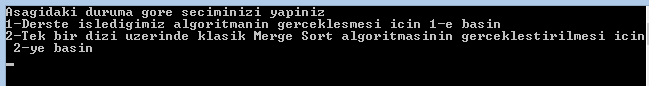
dizi[i + l] = gecicidizi[merge1++];

}

}

**3.3 Çıktıların Elde Edilmesi ve Algoritma Analizi**

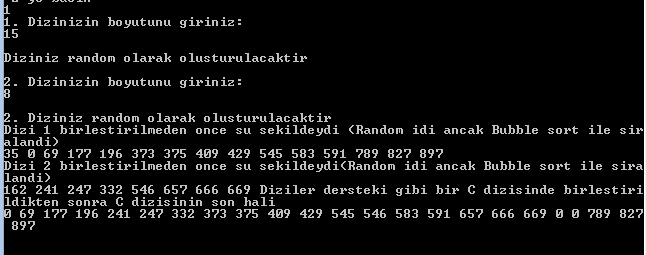
Program başladığında bir arayüz kullanıcıyı yönlendirmektedir. Şekil- deki ekran çıktısı bu arayüzü göstermektedir.



Şekil- Program Başlangıcında Kullanıcıyı Karşılayan Arayüz

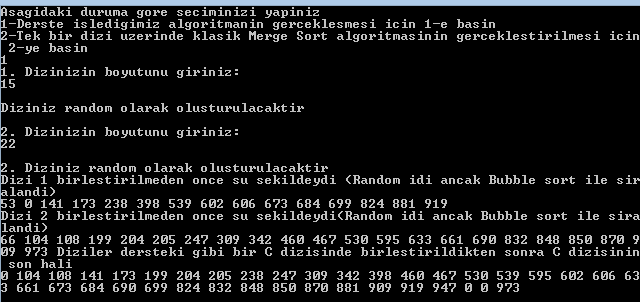
Burada kullanıcının 1’e basması durumunda dersteki algoritmik akış koşturulur. Bu akışa göre öncelikle iki tane random dizi dinamik olarak üretilir. Ardından Bubble Sort a örnek olması açısından diziler bubble sort ile sıralanır. Böylece dinamik ve random olarak derstekine benzer iki farklı dizi elde edilir. Bu diziler bir başka dizide derstekine benzer şekilde sıralı olarak yerleştirilir.

Şekil- de bu durumu özetleyen ekran çıktısı görülmektedir. Dizi 1 için 15 eleman, Dizi 2 için 8 eleman öngörülmüştür. Ve sonuçta dersteki algoritmik akış gerçekleyerek veriler yeni bir C dizisinde sıra ile yerleştirilmiştir. (Ufak bir hata var: C dizisinde ilk iki elemanı göremiyorum sebebini düşündüm ama bulamadım, j yi 0 dan başlatmama rağmen 2. Elemandan başlıyor?)



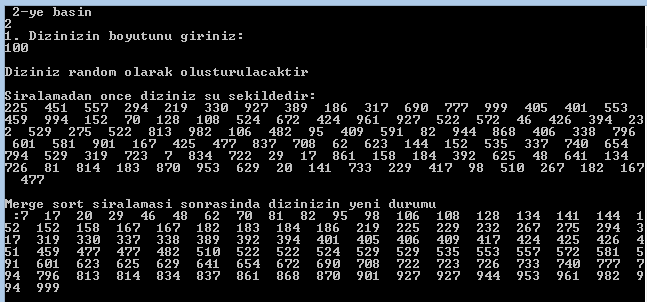
Şekil- Kullanıcının 1’e Basması Halinde Dersteki Algoritmanın Koşturulması

2. bir denemede bu kez ilk dizinin eleman sayısı birinciden az seçilsin ve sırasıyla 15 22 olsun. Bu varsayım altında ekran çıktısı Şekil- deki gibi olacaktır.

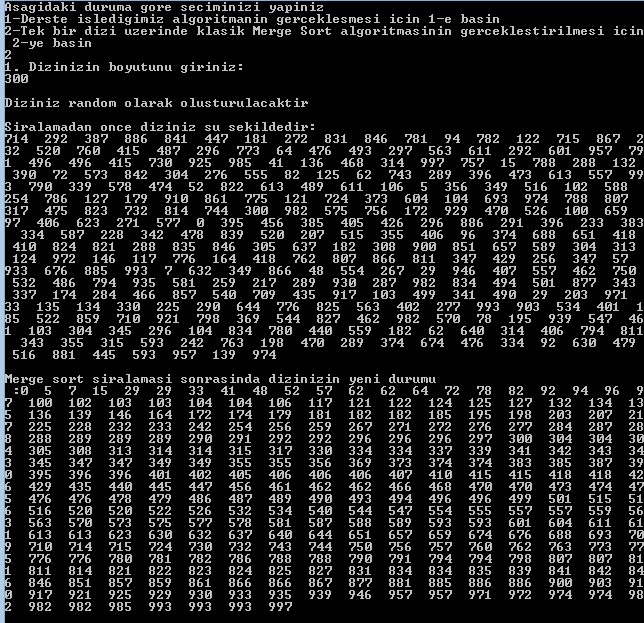


Şekil- Kullanıcının 1’e Basması Halinde Dersteki Algoritmanın Koşturulması

Kullanıcının 2’ye basması durumu esasen bizden ödev olarak istenilen yapıyı teşkil etmektedir. Buna göre bir dizi random ve dinamik olarak oluşturulur. Ardından dizi klasik merge sort algoritmasına göre sıralanır. Bu durum algoritmik akış ve merge sort bölümünde açıkça ifade edilmiştir. Böylece dizi sıralı olarak elde edilir ve ekrana yansıtılır. Buna göre ilgili C kodunun derlenmesi sonucunda elde edilen ekran çıktısı Şekil- de belirtilmiştir. Şekil- ‘de 100 eleman seçilerek işlem gerçekleştirilmiştir. Şekil- de eleman sayısı 300 olarak yeniden değerlendirme yapılmıştır.



Şekil- Kullanıcının 2’ye Basması Sonucu Klasik Bir Merge Sort Algoritmasıyla Random Dizinin Sıralanması (100 Eleman İle-Random Maks:1000)



Şekil- Kullanıcının 2’ye Basması Sonucu Klasik Bir Merge Sort Algoritmasıyla Random Dizinin Sıralanması (300 Eleman İle-Random Maks:1000)

Böylece yazılan kod ve çizilen algoritmik akışa ait çıktılar değerlendirilmiştir. Önemli olduğu düşünülen deneme sonuçları rapora konulmuş, raporda belirtilmeyen pekçok deneme ile sistemin doğru çalıştığı gözlemlenmiştir.

**3.4. Sonuçların yorumlanması**

Sonuçta ödevle sıralama algoritmaları açısından öneme sahip olan Merge Sort algoritması C programlama dili ile gerçekleştirilmiştir. Ödevde sınıfta çözülen algoritmada kodlanıştır. Böylece random olarak bir diziyi sıralamak için en basit sort algoritması olan Bubble Sort algoritması da incelenmiş ve kabaca kodlanmıştır. Sonuçta hız bakımından bir analiz yapılmıştır. Analize göre 100 elemanının üzerindeki değerlerde (2. Dizi olduğu düşünülürse-200 eleman) Bubble sort ile sıralama işlemlerinde uzun saniyeler beklemek gerekliliği ortaya çıkmıştır. Buna karşın 1000 elemanlık bir dizi Merge sort ile milisaniyeler içerisinde sıralanmaktadır. Bu durum hız açısından gözlemlenebilir farkları ortaya çıkarmaktadır. Sonuçta giriş bölümünde de belirtildiği gibi algoritmaların karmaşıklıkları arasında önemli farklar mevcuttur.

**4.Kaynakça**

[1] http://tr.wikipedia.org/wiki/Kabarc%C4%B1k\_s%C4%B1ralamas%C4%B1(İnternet kaynağı, Erişim Tarihi: 22.09.2014)

[2] http://tr.wikipedia.org/wiki/H%C4%B1zl%C4%B1\_s%C4%B1ralama (İnternet kaynağı, Erişim Tarihi: 22.09.2014)