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

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]. Şekil-1'de Bubble Sorta ait bilgiler şematize edilmiştir.

Genel Bilgiler	
Sınıf:	Sıralama algoritması
Veri yapısı:	Dizi
Zaman karmaşıklığı:	$O(n^2)$
Alan karmaşıklığı:	toplamda $O(n)$, ek alan $O(1)$
En iyi:	Yok

Şekil-1 Bubble Sorta Ait Genel Özellikler

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]. Şekil-2'de Quick Sorta ait genel özellikler kısaca ifade özetlenmiştir.

Genel Bilgiler	
Sınıf:	Sıralama algoritması
Veri yapısı:	Değişken
Zaman karmaşıklığı:	Ortalama $O(n \log n)$
Alan karmaşıklığı:	Uygulamaya göre değişken
En iyi:	Ara sıra

Şekil-2 Quick Sorta Ait Genel Özelliklerin Şematize Edilmesi

2.3 Selection Sort

Seçmeli Sıralama, bilgisayar bilimlerinde kullanılan karmaşıklığı bir sıralama algoritmasıdır. Karmaşıklığı {O}(n^2) olduğu için büyük listeler üzerinde kullanıldığında verim sağlamaz ve genel olarak benzeri olan eklemeli sıralamadan daha başarısızdır. Seçmeli sıralama yalın olduğu ve bazı durumlarda daha karmaşık olan algoritmalardan daha iyi sonuç verdiği için tercih edilebilir. [3]

Algoritma aşağıdaki gibi çalışır:

- ✓ Listedeki en küçük değerli öğeyi bul.
- ✓ İlk konumdaki öğeyle bulunan en küçük değerli öğenin yerini değiştir.
- ✓ Yukarıdaki adımları listenin ilk elemanından sonrası için (ikinci elemandan başlayarak) yinele.

Şekil-3'te Selection Sort algoritması için Sözde kod, Şekil-4'te ise Genel bilgilerin özetlendiğinde şematize edilmiş yapı teşkil edilmiştir.

```
for i ← 0 to n-2 do
    min ← i
    for j ← (i + 1) to n-1 do
        if A[j] < A[min]
            min ← j
    swap A[i] and A[min]
```

Şekil-4 Selection Sort Algoritması İçin Sözde Kodun Belirtilmesi

Genel Bilgiler	
Sınıf:	Sıralama algoritması
Veri yapısı:	Dizi
Zaman karmaşıklığı:	$O(n^2)$
Alan karmaşıklığı:	toplamda $O(n)$, ek alan $O(1)$
En iyi:	Genellikle değil

Şekil-5 Selection Sorta Ait Genel Özelliklerin Şematize Edilmesi

2.4 Counting Sort

Verinin hafızada sıralı tutulması için geliştirilen sıralama algoritmalarından (sorting algorithms) bir tanesidir. Basitçe sıralanacak olan dizideki her sayının kaç tane olduğunu farklı bir dizide sayar. Daha sonra bu sayıların bulunduğu dizinin üzerinde bir işlemle sıralanmış olan diziyi elde eder. [5]

Sıralanmak istenen verimiz:

```
5,7,2,9,6,1,3,7
```

olsun. Bu verilerin bir oluşumun (composition) belirleyici alanları olduğunu düşünebiliriz. Yani örneğin vatandaşlık numarası veya öğrenci numarası gibi. Dolayısıyla örneğin öğrencilerin numaralarına göre sıralanması durumunda kullanılabilir.[5]

Bu dizi üzerinden bir kere geçilerek aşağıdaki sayma dizisi elde edilir:

Dizi indisi:	0	1	2	3	4	5	6	7	8	9
Değeri (sayma):	0	1	1	1	0	1	1	2	0	1

Yukarıdaki tabloda da gösterildiği üzere dizide bulunan en büyük eleman sayısı kadar eleman içeren bir sayma dizisi oluşturulmuş ve bu dizinin her elemanına, sıralanmak istenen dizideki her elemanın sayısı girilmiştir.

Bu sayma işleminden sonra sıralı dizinin üretilmesi yapılabilir. Bu işlem de dizinin üzerinden tek bir geçişle her eleman için kaç tekrar olduğu yazılarak yapılabilir. buna göre örneğin sıralanmış dizide 0 hiç olmayacak 1'den 1 tane, 2'den 1 tane olacak şeklinde devam eder ve sonuç: [5]

1,2,3,5,6,7,7,9

şeklinde elde edilir.

Bu sıralama algoritmasının JAVA dilindeki kodu aşağıda verilmiştir: (Şekil-6)

```
public static void countingsort(int[]A, int[]B, int k)
{
   int C[]=new int[k]; // sayma dizisi oluşturuluyor
   int i;
   int j;

   for(i=0; i<k; i++)
   {
      C[i]=0;
   }
   for(j=0; j<A.length; j++)
   {
      C[A[j]]=C[A[j]]+1 ;
   }
   for(i=1; i<k; i++)
   {
      C[i]=C[i]+C[i-1];
   }
   for(j=0; j<A.length; j++)
   {
      B[C[A[j]]]=A[j];
      C[A[j]]=C[A[j]]-1;
   }
}
```

Şekil-6 Counting Sort İçin JAVA Kodu Örneği

Yukarıdaki fonksiyon bir adet sıralanacak dizi, bir adet sıralanmış hali geri döndürecek atıf ile çağırma (call by reference ile) boş dizi ve dizideki en büyük sayının değerini alır. Sonuç ikinci parametre olan boş diziye döner.

Bu sıralama algoritmasının karmaşıklığı (complexity) hesaplarnısa. Dizideki her elemanın üzerinden bir kere geçilerek sayıları hesaplanır. Bu geçiş n elemanlı dizi için n zaman alır. Ayrıca dizideki en büyük elemanlı sayı kadar (bu örnekte k diyelim) büyük olan ikinci bir sayma dizisinin üzerinden de bir kere geçilir bu işlem de k zaman alır. Dolayısıyla toplam zaman n+k kadardır. Bu durumda zaman karmaşıklığı O(n) olur.

Hafıza karmaşıklığına baklırsa (memory complexity) hafızada mevcut verilerin saklandığı bir dizi (yukarıdaki örnek kodda A dizisi). Sonuçların saklandığı ikinci bir dizi (yukarıdaki örnekte B dizisi) ve her elemanın kaçar tane olduğunun durduğu bir dizi (yukarıdaki örnekte C dizisi) tutulmaktadır. Bu durumda A ve B dizileri n, C dizisi ise k boyutundadır ve toplam hafıza ihtiyacı 2n+k kadardır. [5]

2.5 Insertion Sort

Eklemeli Sıralama (İngilizcesi: Insertion Sort), bilgisayar bilimlerinde kullanılan ve sıralı diziyi her adımda öğe oluşturan bir sıralama algoritmasıdır. Büyük dizilerle çalışıldığında hızlı sıralama, birleştirmeli sıralama ve yığın sıralaması gibi daha gelişmiş sıralama algoritmalarından daha verimsiz çalışır. Ancak buna karşın bazı artıları da vardır:

- ✓ Uygulaması kolaydır.
- ✓ Küçük Veri kümeleri üzerinde kullanıldığında verimlidir.
- ✓ Çoğunluğu zaten sıralanmış olan diziler üzerinde kullanıldığında verimlidir.
- ✓ Karmaşıklığı O}(n^2) olan seçmeli sıralama ve kabarcık sıralaması gibi çoğu yalın sıralama algoritmalarından daha verimlidir.
- ✓ Kararlı bir sıralama algoritmasıdır (değeri eşit olan öğelerin asıl listedeki göreceli konumlarını değiştirmez)
- ✓ Sıralanacak diziyi yerinde sıralar, ek bir bellek alanı gerektirmez.
- ✓ Sıralanacak dizinin hepsinin algoritmanın girdisi olmasına gerek yoktur. Dizi parça parça da alınabilir ve sıralama işlemi sırasında diziye yeni veriler eklenebilir.

İnsanlar herhangi bir şeyi (örneğin, iskambil kartları) sıralarken, çoğu durumda eklemeli sıralamaya benzer bir yöntem kullanırlar [4]. Şekil-7'de Insertion Sort için Sözde kod verilmiştir.

```
insertionSort(array A)
    for i = 1 to length[A-1] do
        value = A[i]
        j = i-1
        while j >= 0 and A[j] > value
            A[j + 1] = A[j]
            j = j-1
        A[j+1] = value
```

Şekil-7 Insertion Sort Algoritmasına Ait Sözde Kod

2.6 Merge Sort

Konumuz olan Merge Sort güçlü bir sıralama algoritmasıdır. Birleşmeli Sıralama (Merge Sort), bilgisayar bilimlerinde {O}(n~\log(n)) derecesinde karmaşıklığa sahip bir sıralama

algoritmasıdır. Girdi olarak aldığı diziyi en küçük hale gelene kadar ikili gruplara böler ve karşılaştırma yöntemi kullanarak diziyi sıralar.[6]

Algoritmanın çalışması kavramsal olarak şöyledir:

- ✓ Sıralı olmayan listeyi ortadan eşit olarak iki alt listeye ayırır.
- ✓ Alt listeleri kendi içinde sıralar.
- ✓ Sıralı iki alt listeyi tek bir sıralı liste olacak şekilde birleştirir.

Bu algoritma John von Neumann tarafından 1945 yılında bulunmuştur. Sözde kod formatındaki bir algoritma örneği aşağıdaki gibidir. (Şekil-8)

```
function mergesort(m)
   var list left, right
   if length(m) ≤ 1
       return m
   else
       middle = length(m) / 2
       for each x in m up to middle
           add x to left
       for each x in m after middle
           add x to right
       left = mergesort(left)
       right = mergesort(right)
       result = merge(left, right)
       return result
```

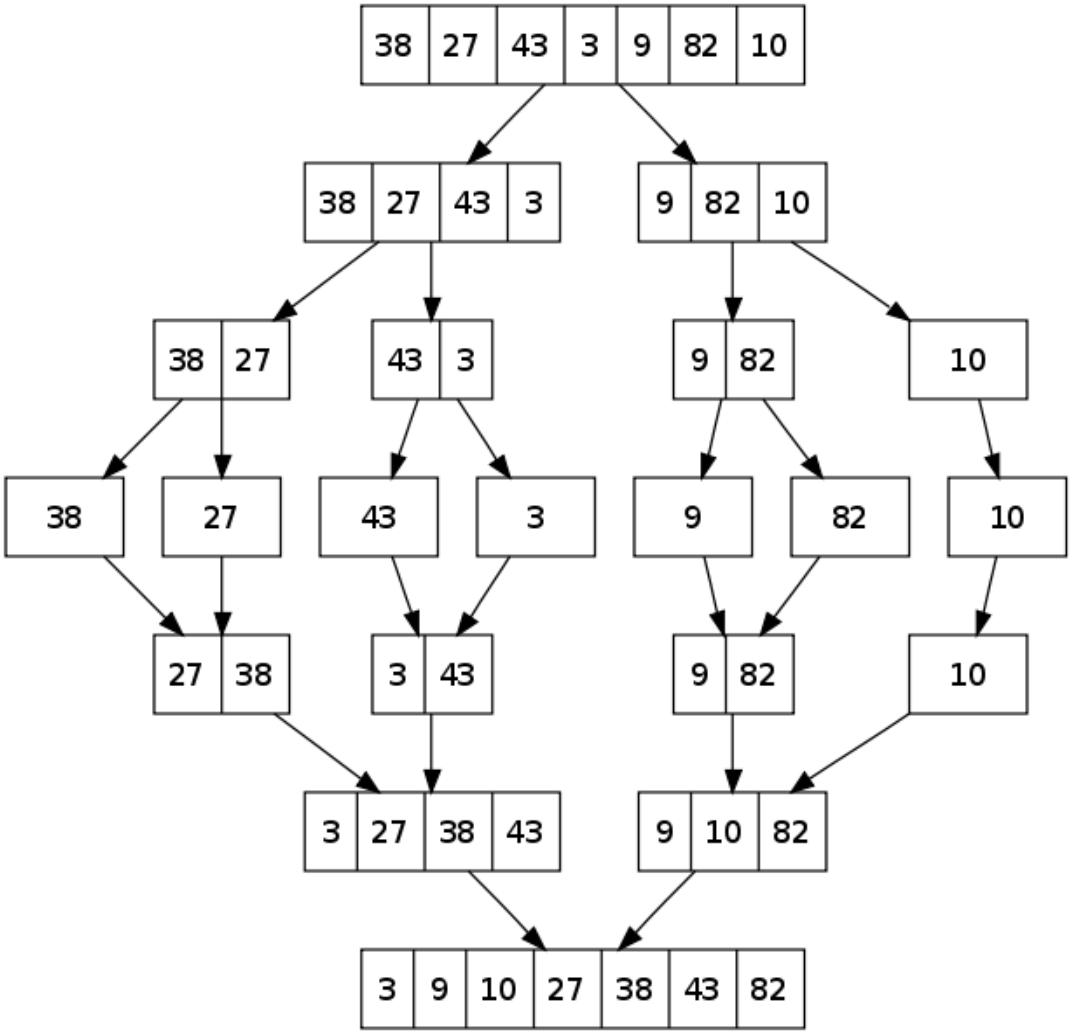
Şekil-8 Merge Sort İçin Sözde Kod Yazımının Belirlenmesi

Yukarıda kullanılan merge( ) fonksiyonunun değişik şekilleri olabilir. Bunlardan en basiti şöyledir. (Şekil-9)

```
function merge(left,right)
   var list result
   while length(left) > 0 and length(right) > 0
       if first(left) ≤ first(right)
           append first(left) to result
           left = rest(left)
       else
           append first(right) to result
           right = rest(right)
   if length(left) > 0
       append left to result
   if length(right) > 0
       append right to result
   return result
```

Şekil-9 Merge Sort İçin Sözde Kod Yazımının Belirlenmesi (Merge() fonksiyonu)

Merge algoritmasına ait örnek aşağıda belirtilmiştir. (Şekil-10)



Şekil-10 (Merge Algoritması İle Sıralama Örneği) [7]

Görüldüğü gibi Merge diziyi önce sürekli 2'ye bölen sonra bu parçaları sıralayan sonra ise birleştiren rekürsif ve hızlı bir algoritmadır. Ödevimizin konusu olması bakımından kod ve algoritmik akış Bölüm-3 ve Bölüm-4 te detaylı olarak incelenmiştir.

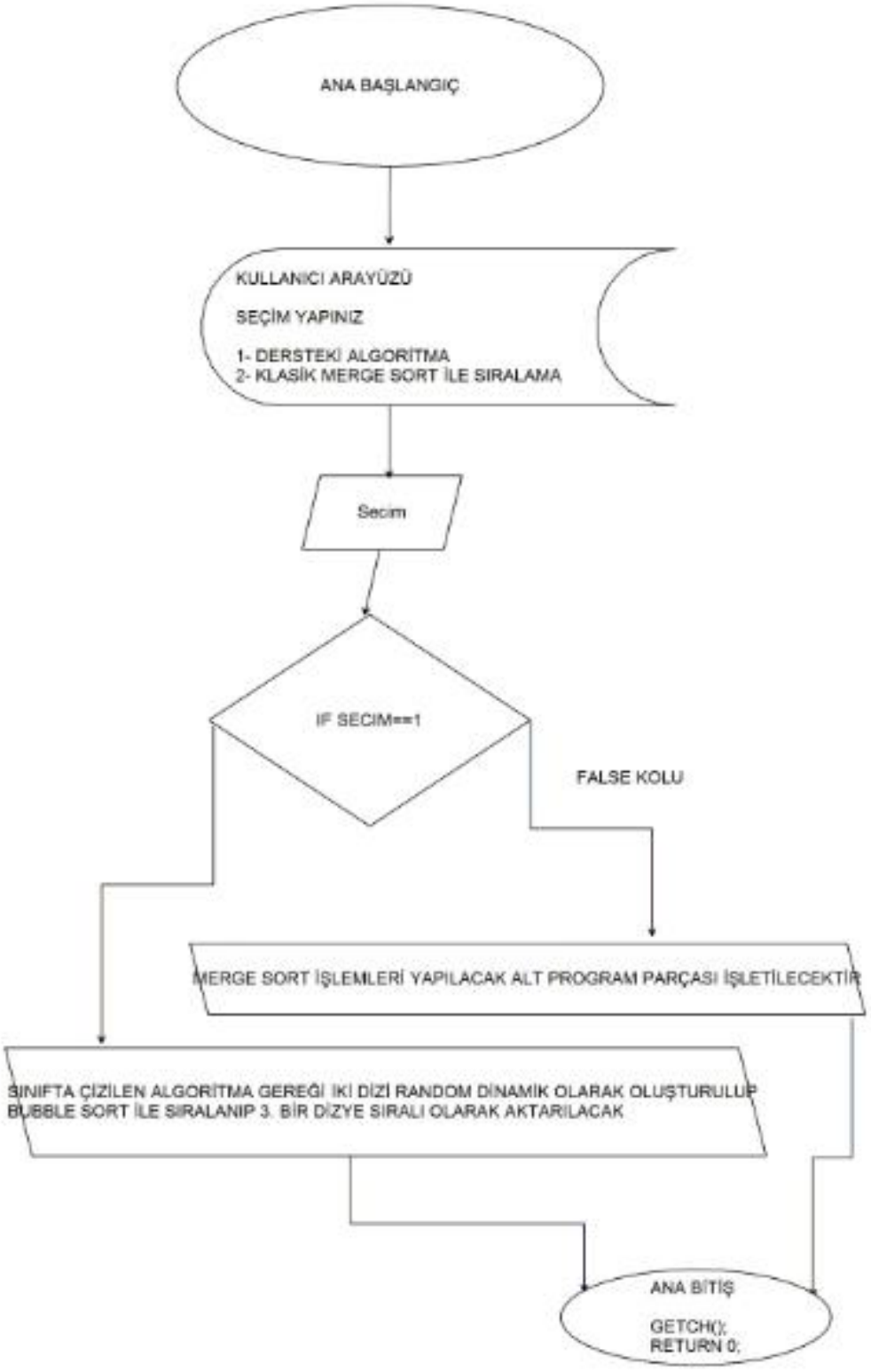
3. Ödevin gerçekleştirilmesi

Bu bölümde öncelikle kodu yazılacak yapıya ait algoritmik akış Diagram Designer adlı programda çizilerek raporda sunulacaktır. Ardından bu algoritmaya ait kod C programlama dilinde kodlanacak, derlenecek ve sonuçlara göre algoritma analizi ve yorumu sunulacaktır.

3.1 Kod için algoritmanın çizilmesi

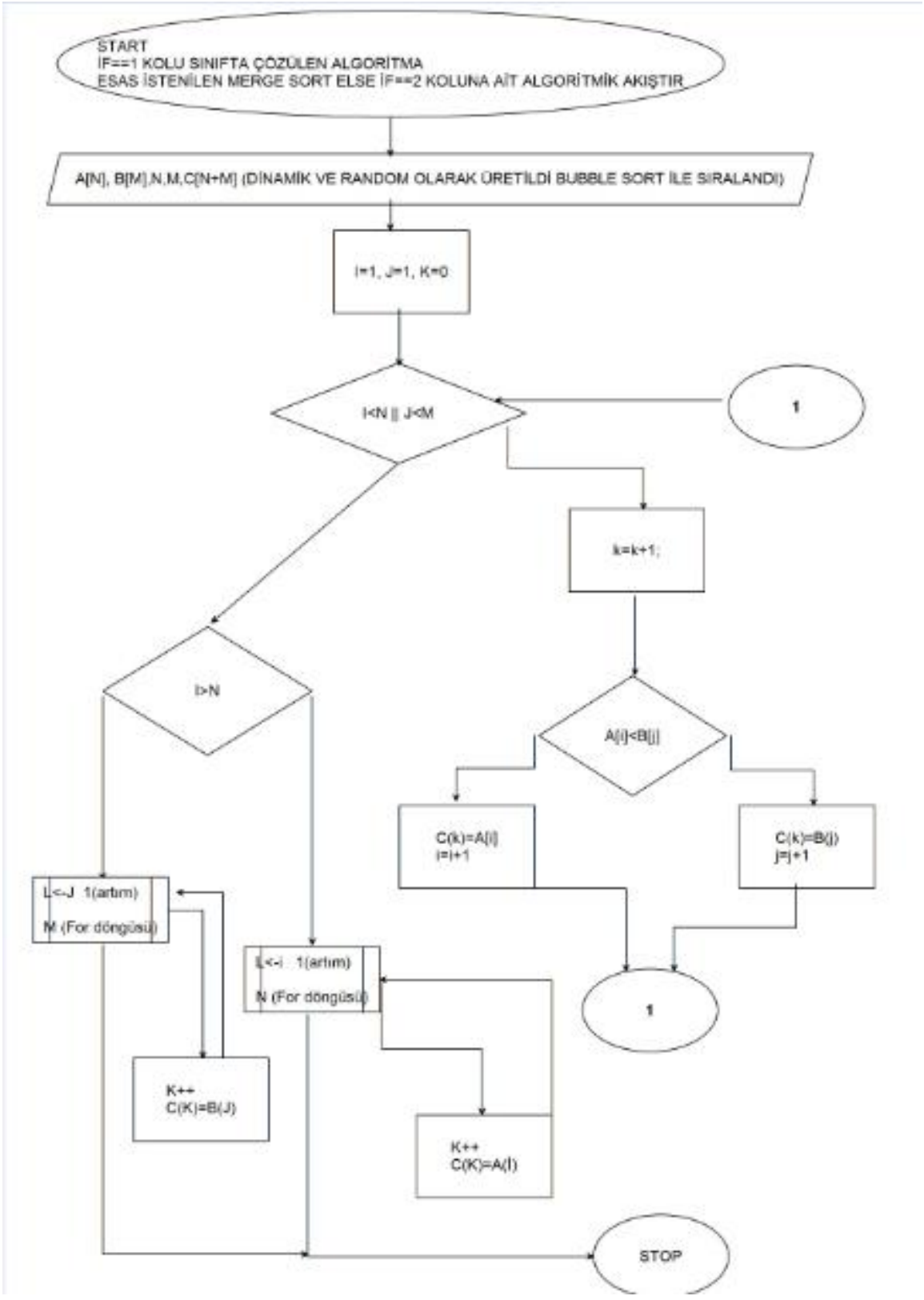
Algoritma için iki durum düşünülmüştür. İlk olarak kullanıcı arayüzü üzerinden 2 alternatif kullanıcıya sunulacaktır. Bu alternatiflerden 1'i derste yapıldığı gibi iki random dizi üretilerek bu dinamik iki diziyi bubble sort ile sıralamak ve akabinde sırası bozulmadan 3. Bir dizide birleştirmektir.

Şekil-11 de ana start ve programın kullanıcıya gelen arayüzü teşkil edilmiştir.



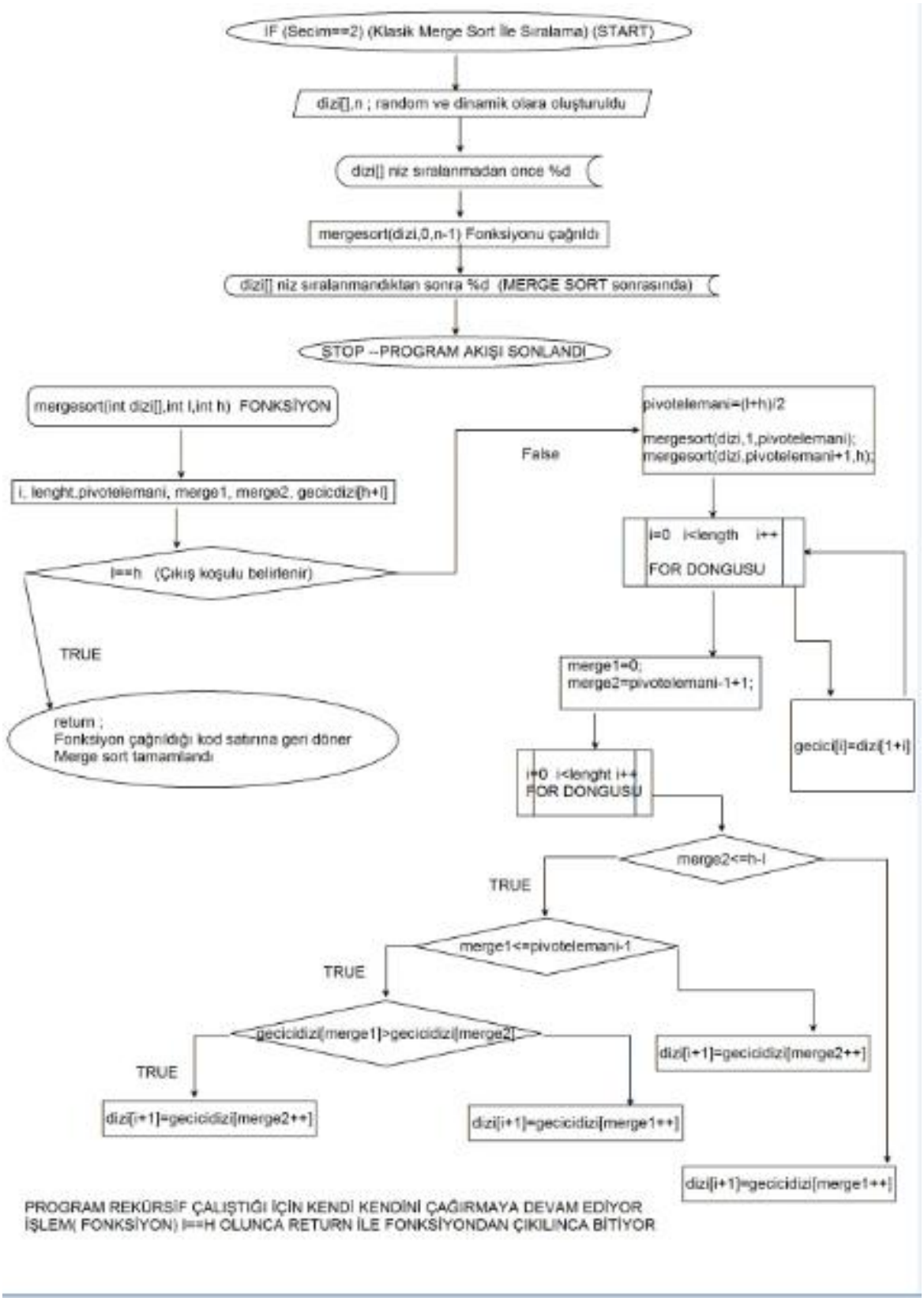
Şekil-11 Programın Ana Başlangıç ve Bitişine Ait Algoritmik Akış

Şekil-12 de kullanıcının 1'i seçmesi durumuna karşılık olan dersteki algoritmik örneğimize ait algoritmik akış gözlemlenmektedir.



Şekil-12 Kullanıcının 1' basması ve dersteki algoritmik akışın oluşturulması

Şekil-13 de kullanıcının 2'i seçmesi durumuna karşılık olan klasik bir Merge Sort ile random dizinin sıralanmasına ait algoritmik akış incelenmiştir.



Şekil 13- Merge Sort Algoritmasının İnclenmesi

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 islediğimiz 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)
   {
```

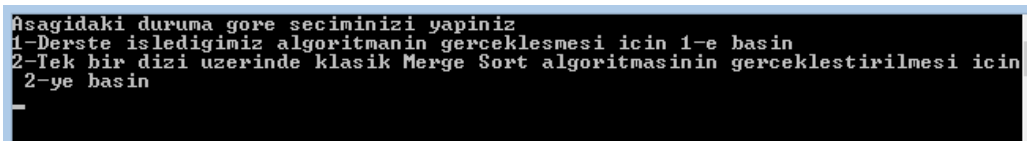
```
   if(gecicidizi[merge1] > gecicidizi[merge2])
    dizi[i + l] = gecicidizi[merge2++];

   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

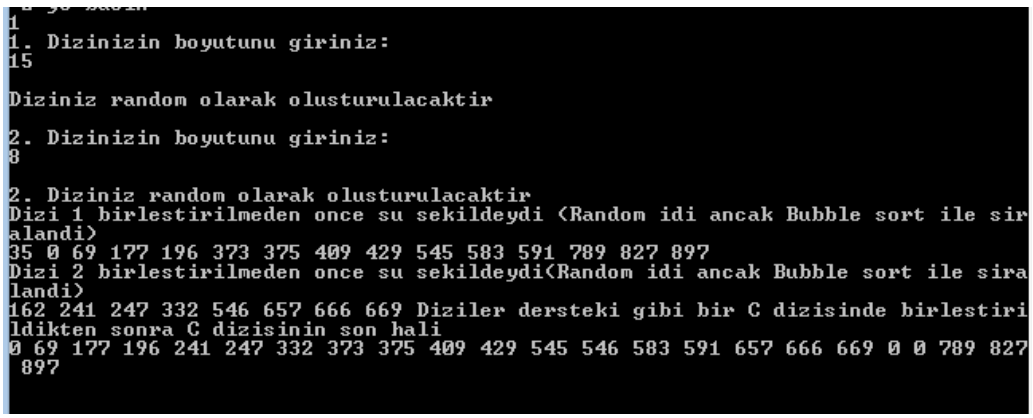
Program başladığında bir arayüz kullanıcıyı yönlendirmektedir. Şekil-14 deki ekran çıktısı bu arayüzü göstermektedir.



Şekil-14 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-15 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-15 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-16 deki gibi olacaktır.

```
Asagidaki duruma gore seciminizi yapiniz
1-Derste islediğimiz algoritmanin gerceklesmesi icin 1-e basin
2-Tek bir dizi uzerinde klasik Merge Sort algoritmasinin gerceklestirilmesi icin
 2-ye basin
1
1. Dizinizin boyutunu giriniz:
15

Diziniz random olarak olusturulacaktir

2. Dizinizin boyutunu giriniz:
22

2. Diziniz random olarak olusturulacaktir
Dizi 1 birlestirilmeden once su sekildeydi (Random idi ancak Bubble sort ile sir
alandi)
53 0 141 173 238 398 539 602 606 673 684 699 824 881 919
Dizi 2 birlestirilmeden once su sekildeydi(Random idi ancak Bubble sort ile sira
landi)
66 104 108 199 204 205 247 309 342 460 467 530 595 633 661 690 832 848 850 870 9
09 973 Diziler dersteki gibi bir C dizisinde birlestirildikten sonra C dizisinin
 son hali
0 104 108 141 173 199 204 205 238 247 309 342 398 460 467 530 539 595 602 606 63
3 661 673 684 690 699 824 832 848 850 870 881 909 919 947 0 0 973
```

Şekil-16 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-17 'de 100 eleman seçilerek işlem gerçekleştirilmiştir. Şekil-18 de eleman sayısı 300 olarak yeniden değerlendirme yapılmıştır.

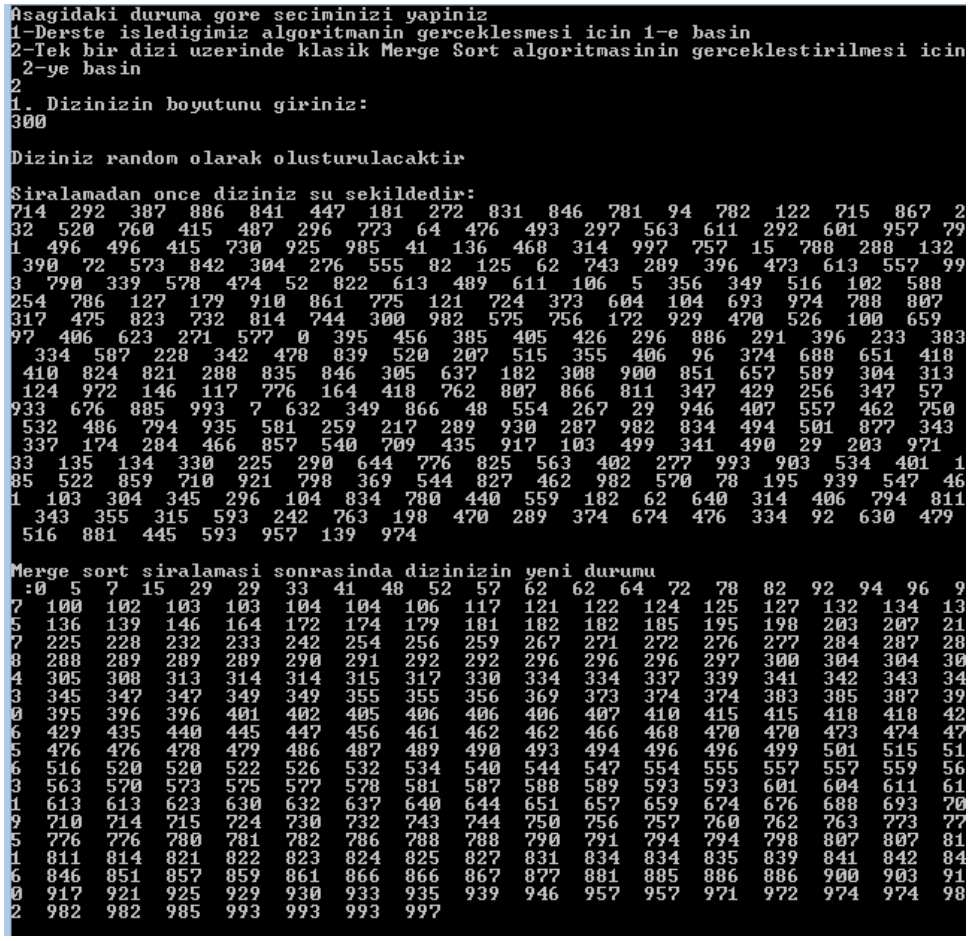
```
 2-ye basin
2
1. Dizinizin boyutunu giriniz:
100

Diziniz random olarak olusturulacaktir

Siralamadan once diziniz su sekildedir:
225  451  557  294  219  330  927  389  186  317  690  777  999  405  401  553
459  994  152  70  128  108  524  672  424  961  927  522  572  46  426  394  23
2  529  275  522  813  982  106  482  95  409  591  82  944  868  406  338  796
 601  581  901  167  425  477  837  708  62  623  144  152  535  337  740  654
794  529  319  723  7  834  722  29  17  861  158  184  392  625  48  641  134
726  81  814  183  870  953  629  20  141  733  229  417  98  510  267  182  167
  477

Merge sort siralamasi sonrasinda dizinizin yeni durumu
 :7  17  20  29  46  48  62  70  81  82  95  98  106  108  128  134  141  144  1
52  152  158  167  167  182  183  184  186  219  225  229  232  267  275  294  3
17  319  330  337  338  389  392  394  401  405  406  409  417  424  425  426  4
51  459  477  477  482  510  522  522  524  529  529  535  553  557  572  581  5
91  601  623  625  629  641  654  672  690  708  722  723  726  733  740  777  7
94  796  813  814  834  837  861  868  870  901  927  927  944  953  961  982  9
94  999
```

Şekil-17 Kullanıcının 2'ye Basması Sonucu Klasik Bir Merge Sort Algoritmasıyla Random Dizinin Sıralanması (100 Eleman İle-Random Maks:1000)



Şekil-18 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.

Yine ödevin bir sonucu olarak karmaşıklığı n^2 olan algoritmaların zaman olarak çok zaman aldığı düşünülerek gerçek zamanlı bilgisayar uygulamalarında veya yüksek sayıda veri ile uğraştığımız işlem gücü sınırlı donanımlarda bu tip algoritmalar yerine daha hızlı cevap verebilecek sorting algoritmalarının var olduğunu bilmek ve uygulayabilmek olmuştur.

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)

[3] http://tr.wikipedia.org/wiki/Se%C3%A7meli_s%C4%B1ralama (İnternet kaynağı, Erişim Tarihi: 22.09.2014)

[4] http://tr.wikipedia.org/wiki/Eklemeli_s%C4%B1ralama (İnternet kaynağı, Erişim Tarihi: 22.09.2014)

[5] http://bilgisayarkavramlari.sadievrenseker.com/2008/08/09/sayarak-siralama-counting-sort/(İnternet kaynağı, Erişim Tarihi: 22.09.2014)

[6] http://tr.wikipedia.org/wiki/Birle%C5%9Ftirmeli_s%C4%B1ralama /(İnternet kaynağı, Erişim Tarihi: 22.09.2014)

[7] www.barisariburnu.com (İnternet kaynağı, Erişim Tarihi: 22.09.2014) (Merge Sort Örnek Resim İçin)

NOT: Genelde kaynak olarak Wikipedia seçilmiş olmasının sebebi tek bir kaynaktan bilgiye erişerek sorting algoritmaları arasındaki farkların daha keskin ve net olarak kavranmasını sağlamaktır. Ödevin genel amacı sorting algoritmalarının çeşitlerini ve temel farklılarını anlamlayarak gözlemleyebilmek olmuştur.