10. SIRALAMA ALGORİTMALARI

Giriş

Elektronik ortamlarda bulundurulan veriler her geçen gün hızla büyümektedir. Buna paralel olarak bilgisayarların fiziksel bileşenlerinin kapasiteleri ve dolayısıyla performansları artsa da, bu çok büyük veri yığınları içerisinden, üzerinde işlem yapılacak verilerin bulunması, işlenmesi ve tekrar kayıt ortamlarına aktarılmasının maiyeti oldukça yüksektir. Bu maliyetleri azaltmanın bir yolu, elektronik ortamlarda bulundurulacak verilerin kayıtlarının belirli bir düzende (sırada) saklanmasıdır. Sözü edilen bu düzen, veriler sayısal ise küçükten büyüğe veya büyükten küçüğe, alfa sayısal ise A ' dan Z 'ye veya Z'den A'ya olabilir. Sıralı veriler üzerinde, elektronik ortamlarda işlem yapmak çok daha hızlı, çok daha kolay ve dolayasıyla maliyeti de çok daha düşüktür.

Sıralama konusu verilerin bilgisayarlarda işlenmeye başlandığı ilk günden bu güne hep önemli bir konu olmuş ve bunun için araştırmacılar her zaman konu üzerinde yoğun bir şekilde çalışmışlar ve birçok sıralama algoritması önermişlerdir. Bugün yaygın olarak kullanılmakta olan 25-30 sıralama algoritması olduğunu söyleyebiliriz. Bu algoritmaların kullanım alanlarına göre üstün yanları ve eksiklikleri vardır. Biz bu ünitede bu algoritmalardan yaygın olarak kullanılmakta olan Kabarcık Sıralaması (Bubble Sort), Seçerek Sıralama (Selection Sort), Araya Ekleme Sıralaması (Insertion Sort), Kabuk Sıralaması (Shell Sort), Hızlı Sıralama ve Birleştirmeli Sıralama (Merge Sort) algoritmalarını ele alacağız.

Daha önce Algoritmalar ve Programlamaya Giriş dersinde bu algoritmalardan Seçerek Sıralama (Selection Sort), Kabarcık Sıralaması (Bubble Sort), ve Araya Ekleme Sıralaması (Insertion Sort) algoritmaları anlatılmıştı. Veri yapıları dersinin bu son ünitesinde konunun daha iyi pekişmesi açısından yeni ele alınan üç sıralama algoritması yanında yukarıda sayılan bu algoritmalara tekrar yer verilmiş ve daha önce C++ ta

yazılan programların kodları bu ünitede C programlama dilinde verilmiştir.

Yukarıda sayılan veya bu ünitede ele alınmayan algoritmaların bazıları kararlı iken bazıları da kararsızdır. Bir sıralama algoritmasının kararlı olabilmesi, sıralanacak dizinin içinde, değerleri birbirine eşit olan öğelerin, sıralama işleminden sonra da birbirlerine göre olan konumlarını koruması anlamına gelir. Eğer bir sıralama algoritması sıralama işleminin yapılması sırasında, sıralanacak dizinin içinde değerleri birbirine eşit olan öğelerin, sıralama işleminden sonra da birbirlerine göre olan konumlarının korumasını garanti etmiyorsa bu tür sıralama algoritmalarına kararsızdır denir.

Kararlı sıralama algoritmalarına örnek olarak Birleştirmeli Sıralama (Merge Sort), Araya Ekleme Sıralaması (Insertion Sort) , Kabarcık Sıralaması (Bubble Sort), Kokteyl Sıralaması (Coctail Sort), kararsız sıralama algoritmalarına örnek olarak Yığınlama Sıralaması (Heap Sort), Hızlı Sıralama (Ouick Sıralama), Sayarak Sıralama (Counting Sort), Kabuk Sıralama (Shell Sort) gösterilebilir.

Dizinin içinde birbirine eşit değerler içeren öğeler birbirlerinden ayırt edilemiyorsa (örneğin sayılar ya da harfler gibi değerler öğenin kendisini oluşturuyor ise) kararlılık bir sorun değildir. Ancak aşağıda gösterildiği gibi sayı çiftleri veya karakter çiftleri her çiftin virgülden önceki elemanına göre sıralanacağı düşünülürse kararlılık sorunu ortaya çıkar.

Sıralama Algoritmaları İçin önemli İki Kavram:

1. Aratan Sıra (Increasing Order) : Sıralanmış bir dizide önceki elemanı takip eden elemanlar, önceki elemandan büyükse dizi artan sıradadır denir.

2. Azalan Sıra (Decreasing Sort) : Sıralanmış bir dizide önceki elemanı takip eden elemanlar, önceki elemandan küçükse dizi azalan sıradadır denir.

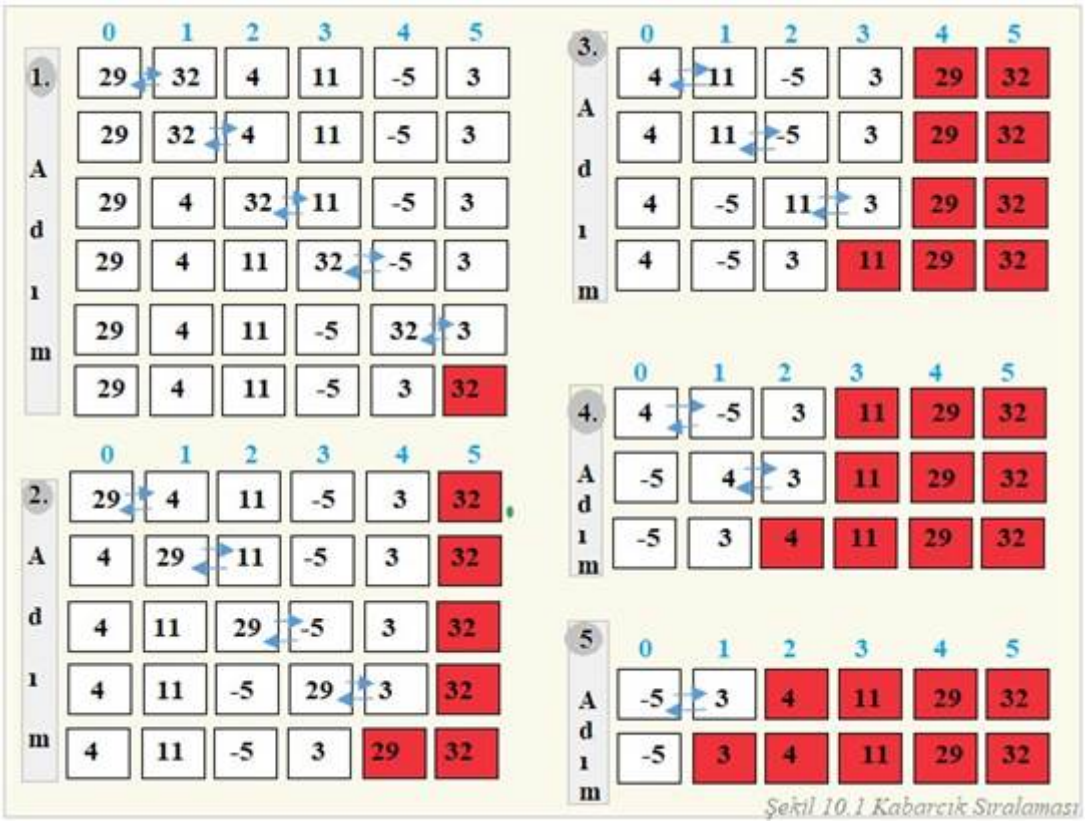
10.1. Kabarcık Sıralaması (Bubble Sort)

Kabarcık Sıralaması (Bubble Sort) basit bir sıralama algoritmasıdır. Bu sıralama algoritması, dizinin her bir bitişik elemanın birbiri ile karşılaştırılması ve elemanlar sıralı değilse elemanların yerlerinin değiştirilmesi esasına dayanan bir sıralama algoritmasıdır.

Kabarcık Sıralaması algoritması ilk önce Yer Değiştirme (Exchange Sort) algoritması olarak isimlendirilmiş, daha sonra dizi içerisindeki büyük elemanların algoritmanın her adımında dizi sonuna kadar doğrusal olarak ilerlemesinden dolayı algoritmaya Kabarcık Sıralaması(Bubble Sort) denmiştir Nabiyev, V. (2016). *Algoritmalar.* Ankara: Seçkin.

Kabarcık sıralaması algoritması ile dizinin küçükten büyüğe sıralanması aşağıdaki şekilde gerçekleşir; Önce dizinin ilk elemanı (indisi sıfır olan eleman) seçilir ve bu eleman kendisinden sonra gelen elemanla (indisi 1 olan eleman) karşılaştırılır. Karşılaştırma sonunda ilk eleman, kendisinden sonra gelen elemandan büyük ise ilk eleman ile kendisinden sonra gelen eleman yer değiştirir. İlk eleman kendisinden sonra gelen elemandan küçük ise yer değiştirme işlemi yapılmaz. Bu işlemden sonra dizinin birinci elemanı dizinin ikinci elemanı ile karşılaştırılır. Karşılaştırma sonucunda birinci eleman ikinci elemandan büyük ise birinci eleman ile ikinci eleman yer değiştirir. Eğer dizinin birinci elemanı dizinin ikinci elemandan daha küçük ise yer değiştirme işlemi yapılmaz. Dizinin son elemanın kendisinden önce gelen eleman ile karşılaştırılıp, gerekirse yer değişikliğinin yapılması sonucunda algoritmanın birinci adımı tamamlanmış olur. Birinci adımın sonunda küçükten büyüğe sıralamada dizinin en büyük elemanı dizinin sonun yerleşir. Kabarcık sıralamasında n elemanlı bir dizinin bütün elemanlarının sıralı hale gelmesi n-1 adımda gerçekleşir. Örneğin 6 (altı) elemanı olan bir dizi 5 (beş) adımda sıralanabilir.

Şekil 10.1’de Kabarcık Sıralaması (Bubble Sort) Algoritmasının 6 (altı) elemanı bulunan bir dizinin sıralaması için gerekli olan adımları gösterilmiştir.



Program 10.1 ‘de C programlama dilinde yazılan Kabarcık Sırlaması Algoritması programı verilmiştir.

Program 10.1Kabarcık Sıralaması

```c
#include <stdio.h>
#include <stdbool.h>
#include <locale.h>
#define MAX 6
int dizi[MAX] = {29,32,4,11,-5,3};
void yazdir() {
   int i;
   printf("[");
   for(i = 0; i < MAX; i++) {
      printf("%d ",dizi[i]);
   }
    printf("]\n");
}
void bubbleSort() {
   int temp;
   int i,j;
   bool degis = false;
   for(i = 0; i < MAX-1; i++) {
       degis = false;
       for(j = 0; j < MAX-1-i; j++) {
        printf("Karşılaştırılan Elemanlar: [ %d, %d ] ", dizi[j],dizi[j+1]);
       if(dizi[j] > dizi[j+1]) {
           temp = dizi[j];
           dizi[j] = dizi[j+1];
           dizi[j+1] = temp;
            degis = true;
           printf(" => yer değiştirdi [%d, %d]  \n",dizi[j],dizi[j+1]);
       } else {
           printf(" => yer değiş
       }
       }
      if(! degis ) {
         break;
      }
      printf("Adım %d#: ",(i+1));
      yazdir();
   }
}
void main() {
   setlocale(LC_ALL,"Turkish");
   printf("Sırasız Dizi: ");
   yazdir();
   printf("\n");
   bubbleSort();
   printf("\nSıralanmış Dizi: ");
   yazdir();
}
```

Ekran Çıktısı

```
Sırasız Dizi: [29 32 4 11 -5 3 ]
Karşılaştırılan Elemanlar: [ 29, 32 ] => yer değiştirmedi
Karşılaştırılan Elemanlar: [ 32, 4 ] => yer değiştirdi [4, 32]
Karşılaştırılan Elemanlar: [ 32, 11 ] => yer değiştirdi [11, 32]
Karşılaştırılan Elemanlar: [ 32, -5 ] => yer değiştirdi [-5, 32]
Karşılaştırılan Elemanlar: [ 32, 3 ] => yer değiştirdi [3, 32]
Adım 1#: [29 4 11 -5 3 32 ]
Karşılaştırılan Elemanlar: [ 29, 4 ] => yer değiştirdi [4, 29]
Karşılaştırılan Elemanlar: [ 29, 11 ] => yer değiştirdi [11, 29]
Karşılaştırılan Elemanlar: [ 29, -5 ] => yer değiştirdi [-5, 29]
Karşılaştırılan Elemanlar: [ 29, 3 ] => yer değiştirdi [3, 29]
Adım 2#: [4 11 -5 3 29 32 ]
Karşılaştırılan Elemanlar: [ 4, 11 ] => yer değiştirmedi
Karşılaştırılan Elemanlar: [ 11, -5 ] => yer değiştirdi [-5, 11]
Karşılaştırılan Elemanlar: [ 11, 3 ] => yer değiştirdi [3, 11]
Adım 3#: [4 -5 3 11 29 32 ]
Karşılaştırılan Elemanlar: [ 4, -5 ] => yer değiştirdi [-5, 4]
Karşılaştırılan Elemanlar: [ 4, 3 ] => yer değiştirdi [3, 4]
Adım 4#: [-5 3 4 11 29 32 ]
Karşılaştırılan Elemanlar: [ -5, 3 ] => yer değiştirmedi
Adım 5#:
Sıralanmış Dizi: [-5 3 4 11 29 32 ]
```

Program 10.1, her C programında olduğu gibi *main* fonksiyonu ile çalışmaya başlamakta ve önce yazdir() isimli yardımcı fonksiyon çalıştırılarak dizinin sıralanmamış hali ekrana yazdırılmaktadır. Sonra 6 elemanlı bir dizi *bubbleSort()* isimli C fonksiyonuna parametre olarak gönderilmekte ve fonksiyon çağırılmaktadır. Bu fonksiyon, iki adet *for* döngüsü içermekte olup ilk döngü Şekil 10.1'deki adımları (iterasyonları) temsil etmektedir. İkinci döngü ise her adımda yapılan karşılaştırma işlemlerini gerçekleştirmektedir. bubbleSort() fonksiyonun

çalışması tamamlandığında dizinin küçükten büyüğe sıralanması da tamamlanmış oluyor. Daha sonra yazdir() fonksiyonu tekrar çağırılarak sıralanmış dizi ekrana yazdırılıyor. Program 10.1 çalıştırıldığında, program listesinin köşesinde verilen ekran çıktısı elde edilir.

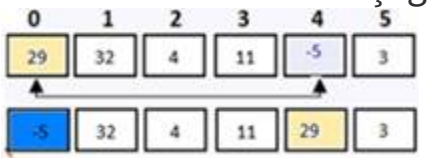
10.2. Seçerek Sıralama (Selection Sort)

Seçerek sıralama algoritması sıralama problemine getirilen basit çözümlerden bir diğeridir. Algoritmanın çalışması sırasında dizinin iki bölüme ayrıldığı varsayılır. Bu bölümlerden ilki sıralanmış dizi, ikincisi ise sıralanmamış dizidir. Başlangıçta sıralanmış dizi boş, sıralanmamış dizi ise, dizinin elemanlarının tamamından oluşur.

Seçerek sıralama algoritmasının çalıştırılmasına, dizinin ilk elemanının seçilmesi ile başlanır. Seçilen dizi elemanı ile bu elemandan sonra gelen dizi elemanları karşılaştırılır ve en küçük eleman bulunur. Bulunan bu en küçük eleman seçilen eleman ile yer değiştirir. Bu işlemin sonucunda dizinin en küçük elemanı belirlenerek dizinin başına yazılmış olur. Dizinin başına yazılmış olan bu eleman sıralanmış dizinin ilk elemanıdır.

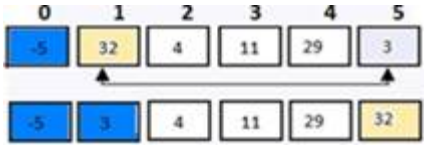
Seçerek sıralama algoritmasının çalışması şekil 10.2'de verilen örnek üzerinde adım adım gösterilmiştir. Şekilden(*1. Adım'a bakınız*) de görülebileceği gibi başlangıçta dizinin **0.** Elemanı olan **29** seçilmiştir. Sonra, seçilen elemandan sonra gelen dizi elemanları içerisinden en küçük olan eleman bulunmuştur. Örnekte bu eleman **-5**'tir. **-5** dizinin dördüncü elemanıdır.

İlk adımda bulunan en küçük sayı **(-5)**, dizinin **0.** Elemanı **29** ile yer değiştirir ve dizinin elemanlarının sıralaması aşağıdaki gibi olur.

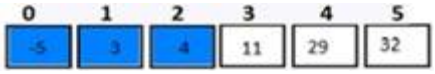


Yukarıdaki işlemden sonra dizinin **1.** Elemanı olan **32** seçilir ve **32'**den sonra gelen dizi elemanları içerisinden en küçük eleman bulunur. Örnekte bu eleman **3**'tür. **3** dizinin **5.** Elemanıdır.

Bu adımda bulunan en küçük sayı (**3**) dizinin 5. Elemanı **32** ile yer değiştirir ve dizinin elemanlarının sıralaması aşağıdaki gibi olur.



Üçüncü adımda dizinin **2.** Elemanı olan **4** seçilir ve **4'**den sonra gelen dizi elemanları içerisinden en küçük eleman araştırılır. Örnekte 4'ten sonra 4'ten küçük eleman yoktur. Dolayısı ile bu adımda dizi elemanları yer değiştirmez. Bu adımdan sonraki dizi elemanlarının sıralı (mavi renkte olan kısım) ve sıralı olmayan kısmı (beyaz renkte olan kısım) aşağıda gösterilmiştir.



Bundan sonraki dördüncü ve beşinci adımlarda da üçüncü adıma benzer şekilde seçilen elemandan sonraki dizi elemanları seçilen elemandan daha küçük olmadığı için dizi elemanları arasında herhangi bir yer değişikliği işlemi yapılmamıştır.

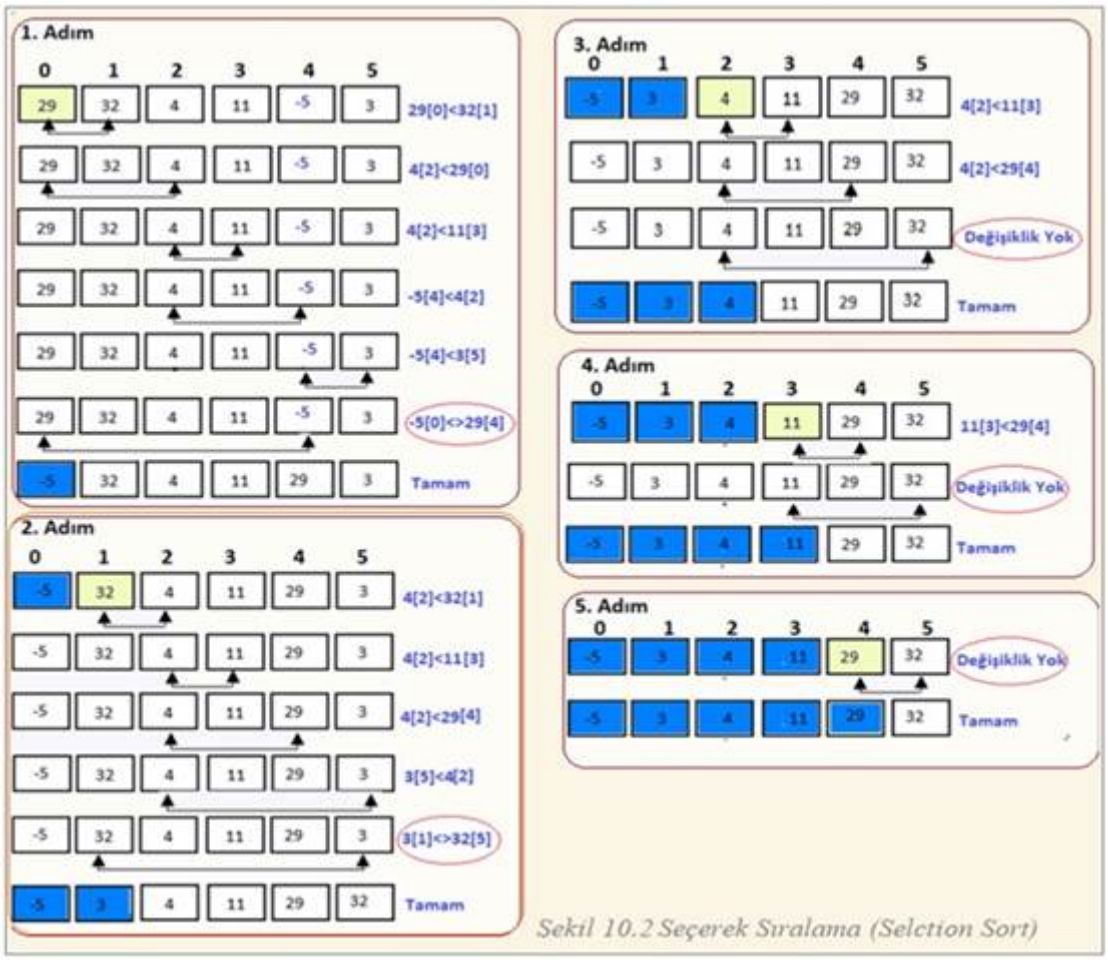
Aşağıda bu adımlar sonunda dizinin durumu gösterilmiştir.

4. Adım sonunda dizi:

5. Adım Sonunda dizi:

Yukarıdaki işlem adımları kontrol edildiğinde Seçerek Sıralama algoritması ile ***n*** elemanlı bir dizinin ***n-1*** adımda sıralanabileceği görülmektedir.

Şekil 10.2'de Seçerek Sıralama (Selection Sort) Algoritmasının çalışma şekli bütün olarak gösterilmiştir.



Şekil 10.2 Seçerek Sıralama (Selction Sort)

Program 10.2 ‘de C programlama dilinde yazılan Seçerek Sırlama Algoritması programı verilmiştir.

Program 10.2 Seçerek Sıralama (Selection Sort

```c
#include <stdbool.h>
#include <locale.h>
#define MAX 6
int dizi[MAX] = {29,32,4,11,-5,3};
void cizgi(int sayac) {
   int i;
   for(i = 0;i < sayac-1;i++) {
      printf("=");
   }
   printf("=\n");
}
void yazdir() {
   int i;
   printf("[");
   for(i = 0;i < MAX;i++) {
      printf("%d ", dizi[i]);
   }
   printf("]\n");
}
void selectionSort() {
   int indexMin,i,j;
    for(i = 0; i < MAX-1; i++) {
      indexMin = i;
      for(j = i+1;j < MAX;j++) {
         if(dizi[j] < dizi[indexMin]) {
            indexMin = j;
         }
      }
      if(indexMin != i) {
 printf("Eleman Değitir:[ %d, %d ]\n",dizi[i],dizi[indexMin]);
      int temp = dizi[indexMin];
      dizi[indexMin] = dizi[i];
      dizi[i] = temp;
      }
      printf("Adım %d#:",(i+1));
      yazdir();
   }
}
void main() {
   setlocale(LC_ALL,"Turkish");
   printf("Sıralı Olmayan Dizi: ");
   yazdir();
   cizgi(50);
   selectionSort();
   printf("Sıralı Dizi        : ");
   yazdir();
   cizgi(50);
}
```

```
Sıralı Olmayan Dizi: [29 32 4 11 -5 3 ]
==================================================
Eleman Değitir: [ 29, -5 ]
Adım 1#:[-5 32 4 11 29 3 ]
Eleman Değitir: [ 32, 3 ]
Adım 2#:[-5 3 4 11 29 32 ]
Adım 3#:[-5 3 4 11 29 32 ]
Adım 4#:[-5 3 4 11 29 32 ]
Adım 5#:[-5 3 4 11 29 32 ]
Sıralı Dizi        : [-5 3 4 11 29 32 ]
==================================================
```

Program 10.2'nin Ekran Çıktısı

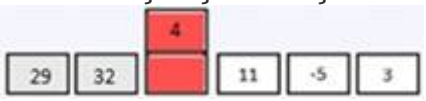
10.3. Araya Ekleme Sıralama (Insertion Sort) Algoritması

Araya ekleme sıralama algoritmasının da sıralama problemine getirilen basit temel yaklaşımlardan biri olduğunu söyleyebiliriz. Algoritmanın çalıştırılmasına dizinin ikinci elemanı seçilerek başlanır. Seçilen ikinci

eleman kendinden önceki elemanlarla karşılaştırılır ve dizinin büyük elemanları sağa kaydırılır. Bu kaydırma sonucunda açılan uygun pozisyona o anda sıralanmakta olan eleman yerleştirilir. Şekil 10.3'te 6 elemanlı sıralı olmayan bir dizi verilmiştir ve bu dizinin elemanlarının yerleştirmeli sıralama yöntemi ile sıralanması adım adım gösterilmiştir.

İlk adımda dizinin 1. elemanı (32) seçilmiş ve kendisinden önce gelen elemanla (29) karşılaştırılmış, 29 sayısı 32'den küçük olduğu için dizi elemanlarının yerlerinde herhangi bir değişiklik yapılmamıştır.

İkinci adımda dizinin 2. Elemanı (4) seçilmiş ve kendisinden önce gelen dizinin 1. ve 0. elemanları ile karşılaştırılmıştır.



Karşılaştırma sonucunda önce dizinin 1. Elemanı olan 32, dizinin ikinci elemanı olan 4'ün yerine, sonra da dizinin 0. Elemanı olan 29, dizinini birinci elemanı olan 32'nin yerine gelmiştir. Seçilmiş olan dizi elemanı bu durumda dizinin 0. elemanı olmuştur. İkinci adımın sonunda dizinin sıralaması aşağıdaki gibi olmuştur.

Bu işlemler aynı kurallarla üçüncü, dördüncü ve beşinci adımlarda sürdürülerek beşinci adımın sonunda dizi küçükten büyüğe sıralanmıştır.

Şekil 10.3'de Araya ekleme sıralama (Insertion Sort) Algoritmasının çalışma şekli bütün olarak gösterilmiştir. Araya ekleme sıralamasında da sıralanacak dizinin **n** elemanı olması durumunda, dizinin tamamen sıralanması için **n-1** adıma ihtiyaç vardır.

Araya ekleme sıralama algoritmasında seçilen dizi elemanının doğru yere yerleştirilmesi için bu elemanın geçici bir değişkende tutulması, kendisinden önce gelen, kendisinden büyük elemanların sağa doğru

kaydırılması ve boşalan yere geçici değişkende tutulan elemanın yerleştirilmesi gerekir.

```c
#include <stdio.h>
#include <locale.h>
 void yazdir(int dizi[], int boyut ){
    int i;
    for (i = 0; i < boyut; i++){
          printf("%d ", dizi[i]);
    }
    printf("\n");
 }
void insertionSort(int dizi[], int boyut)
 {
    int i, j, anahtar;
    for (i = 1; i < boyut; i++){
         anahtar = dizi[i];
         j = i - 1;
        while (j >= 0 && dizi[j] > anahtar) {
            dizi[j + 1] = dizi[j];
            j--;
        }
             dizi[j + 1] = anahtar;
             printf("\n%d. adim: ", i);
             yazdir(dizi, boyut);
    }
    printf("\nSirali : ");
    yazdir(dizi, boyut);
 }
int main(){
  setlocale(LC_ALL,"Turkish");
  int dizi[]={29,32,4,11,-5,3};
  int boyut=sizeof(dizi)/sizeof dizi[0];
  printf("\nSirasiz: ");
  yazdir(dizi, boyut);
  insertionSort(dizi, boyut);
 return 0;
}
```

Program 10.3 Araya Ekleyerek Sıralama

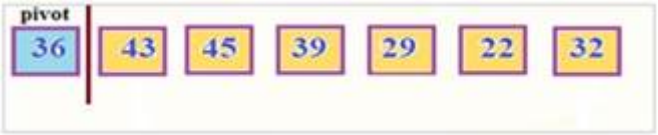
```
Sirasiz: 29 32 4 11 -5 3

1. adim: 29 32 4 11 -5 3

2. adim: 4 29 32 11 -5 3

3. adim: 4 11 29 32 -5 3

4. adim: -5 4 11 29 32 3

5. adim: -5 3 4 11 29 32

Sirali : -5 3 4 11 29 32
```

Program 10.3'ün ekran çıktısı

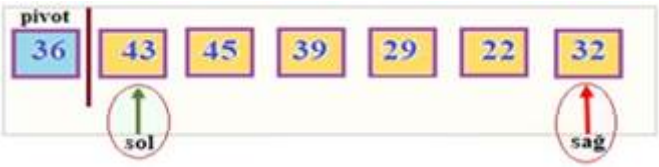
10.4. Hızlı Sıralama (Quick Sort) Algoritması

Hızlı sıralama (Quick Sort) algoritması, oldukça verimli bir sıralama algoritmasıdır ve şu ana kadar ele alınan sıralama algoritmalarından farklı olarak böl ve yönet (divide-and-conquer) yöntemini kullanarak sıralama işlemini gerçekleştirir. Böl ve yönet algoritmaları, problemlerin mümkün olan en küçük alt parçalara ayrıldığı, her bir alt parçanın diğerlerinden bağımsız şekilde çözüldüğü algoritmalardır. Problemin genel çözümü elde edilirken alt parçalara ait çözümler belirli bir sırayla bir araya getirilir.

Hızlı sıralama algoritması çalışmaya başlatılırken önce sıralanacak dizinin elemanlarından biri pivot (dayanak) olarak seçilir. Pivot dizinin ilk elemanı olabileceği gibi, son elemanı ve veya dizinin her hangi bir elemanı da olabilir. Başka bir ifade ile pivot rasgele seçilebilir. 1962 yılında C. A. R. Hoare hızlı sıralama konusunda yazdığı ilk makalede pivot olarak dizinin ilk elemanını seçmiştir. Daha sonra bu konu ile ilgili yazılan makalelerde pivot seçimi birçok tartışma yapılmıştır. Biz de bu ünitede hızlı sıralama konusunu anlatırken pivot olarak dizinin ilk elemanını seçeceğiz.



Hızlı sıralama algoritmasında pivot seçildikten sonra her adımda, dizinin ilk bölümde pivottan küçük elemanlar, ikinci bölümde pivottan büyük elemanlar olacak şekilde dizi iki ayrı kısma ayrılır ve pivot bu iki ayrı bölümün ortasına yerleştirilir. Dizinin ikiye bölünebilmesi için biri ***sol*** diğer ***sağ*** olmak üzere iki pointer kullanılır.



Başlangıçta sol pointer dizinin ilk elemanını, sağ pointer da son elemanını işaret eder ve işaret edilen bu iki eleman karşılaştırılır. Eğer sol pointer tarafından işaret edilen elemanın değeri, sağ pointer ile işaret edilen elemanın değerinden büyük ise bu elemanlar yer değiştirir. Yer değiştirme işleminden sonra sol pointer sağa ve sağ pointer da sola,

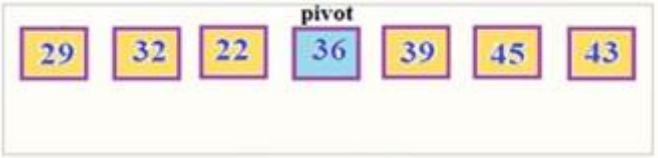
bir sonraki elemanları gösterecek şekilde ilerletilir. Bu işlem karşılaştırılan elemanlardan soldaki eleman sağdaki elemandan büyük olduğu durumda yukarıda açıklandığı şekilde gerçekleşir.

Eğer karşılaştırılan elemanlardan soldaki eleman sağdaki elemandan küçük veya eşit ise işaret edilen elemanlar yer değiştirmez. Böyle bir durumda sağ pointer hareket ettirilmez. Sol pointer bir sonraki elemanı gösterecek şekilde ilerletilir. Pointerların gösterdiği elemanlar tekrar karşılaştırılır.

Sol pointerin işaret ettiği 45 (Kırk Beş) ile sağ pointerin işaret ittiği 22 (Yirmi İki) karşılaştırılır ve bu elemanlar yer değiştirir. Pointerler birer adım ilerletilir.

Sol pointerin işaret ettiği 39 (Otuz Dokuz) ile sağ pointerin işaret ittiği 29 (Yirmi Dokuz) karşılaştırılır ve bu elemanlar yer değiştirir.

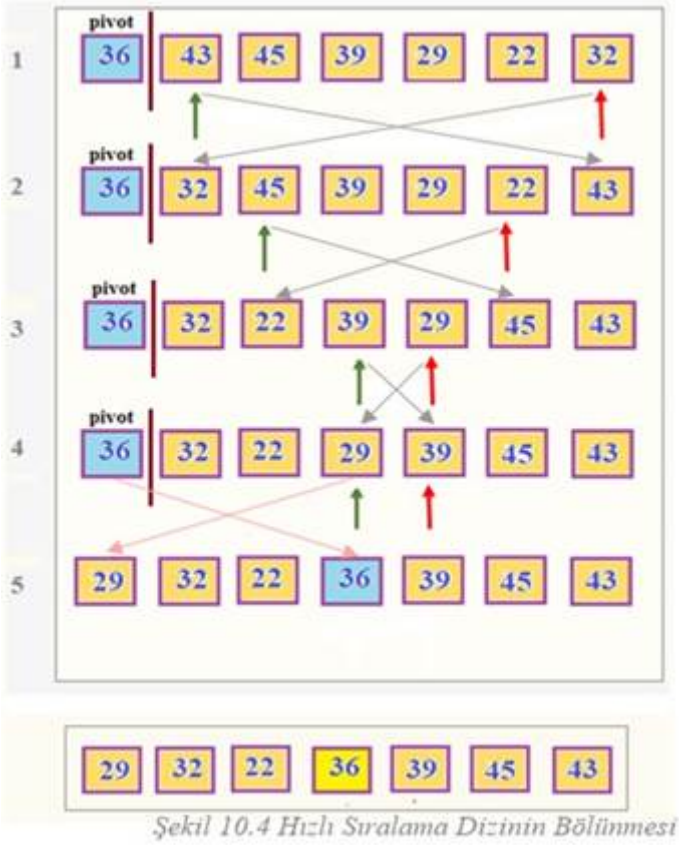
Yukarıdaki adımdan sonra pivot ‘un sağında karşılaştırılacak dizi elemanı kalmamıştır. Bu durumda sol pointer ‘ın işaret ettiği 29 (Yirmi Dokuz) ile pivot yeri değiştirilir.



Bu yer değiştirme işleminden sonra pivot ‘un sağında pivottan küçük olan elemanlardan oluşan bir alt dizi, pivotun solunda ise pivottan büyük elemanlardan oluşan bir alt dizi oluşmuştur. Ayrıca, bu adımla birlikte pivot dizide yeri değişmeyecek bir sıraya yerleşmiştir.

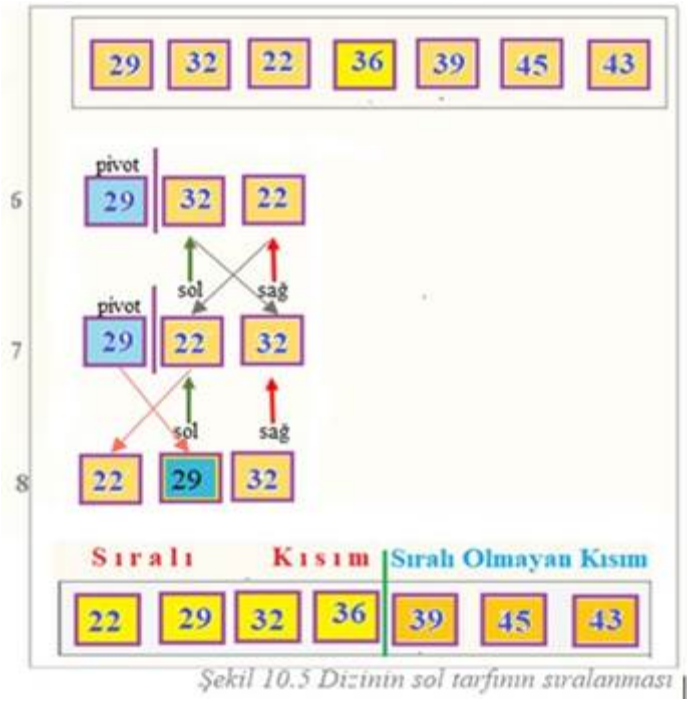
Bundan sonra dizinin sıralanmış olan kısımları konun anlaşılabilmesi için farklı renkte (sarı) gösterilecektir.

Şekil 10.4 ‘te yukarıda açıklanan işlem adımları toplu olarak gösterilmiştir.



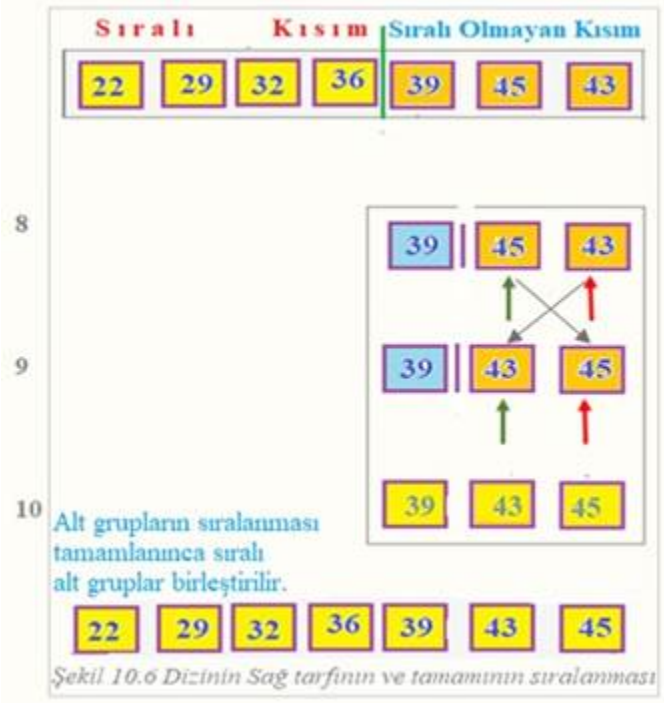
Şekil 10.4 Hızlı Sıralama Dizinin Bölünmesi

Bu aşamadan sonra sol ve sağ alt dizilerin kendi aralarında sıralanması işlemine geçilecektir. Hangi alt dizinin önce sıralanacağının bir önemi yoktur. Fakat biz önce sol alt diziyi ele alacağız. Aslında burada yapılacak olan, yukarıdaki işlem adımlarının aynısının sol alt diziye uygulanmasından ibarettir. Yani, bu defa sol alt dizi için bir pivot belirlenecek, alt dizi üzerine iki pointer yerleştirilecek ve aynı kurallar uygulanarak dizinin elemanları karşılaştırılacaktır. İşlemin sonunda bu alt dizinin de ikiye bölünmesi beklenmektedir. Bu işlemler şekil 10.5 ‘te gösterilmiştir.



Şekil 10.5 Dizinin sol tarfının sıralanması

Şimdi sıra sol alt diziye uygulanan işlemlerin sağ alt diziye de uygulanmasına ve dizinin tamamının sıralama işleminin tamamlanmasına gelmiştir. Şekil 10.6'da dizinin sağ tarafının ikiye bölünmesi ve bu işlemlerin sonucunda dizinin sağ tarafının sıralanması gösterilmiştir.



Şekil 10.6 Dizinin Sağ tarfının ve tamamının sıralanması

Program 10.4 'te hızlı sıralama (Quick Sort) algoritmasının C kodu verilmiştir.

```c
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
void hizliSirala(int *dizi,int ilkEleman,int sonEleman){
    int i,j,pivot,temp;
    pivot=ilkEleman;
    if(sonEleman>ilkEleman){
        pivot=ilkEleman;
        i=ilkEleman;
        j=sonEleman;
        while (i<j){
            while (dizi[i]<=dizi[pivot] && i<sonEleman && j>i){
                i++;
            }
            while (dizi[j]>=dizi[pivot] && j>=ilkEleman && j>=i){
                j--;
            }
            if (j>i){
                temp=dizi[i];
                dizi[i]=dizi[j];
                dizi[j]=temp;
            } }
        temp=dizi[j];
        dizi[j]=dizi[pivot];
        dizi[pivot]=temp;
        hizliSirala(dizi,ilkEleman,j-1);
        hizliSirala(dizi,j+1,sonEleman);
    }
}
int main(){
    setlocale(LC_ALL,"Turkish");
    int *dizi,i,boyut;
    printf("Dizinin Eleman Sayısı : "); scanf("%d",&boyut);
    dizi=(int *)malloc(boyut*sizeof(int));
    for(i=0;i<boyut;i++){
        printf("Dizinin %d. elemanı :",i+1); scanf("%d",&dizi[i]);
    }
    printf("Dizinin Sıralanmadan Önceki Durumu    : ");
    for(i=0;i<boyut;i++){
        printf("%d ",dizi[i]);
    }
    hizliSirala(dizi,0,boyut-1);
    printf("\n\nDizinin Sıralamadan Sonraki Durumu: ");
    for(i=0;i<boyut;i++){
        printf("%d ",dizi[i]);
    }
    return 0;
}
```

Program 10.4 Hızlı Sıralama Algoritması

10.4. Birleştirmeli Sıralama (Merge Sort) Algoritması

Birleştirerek sıralama (Merge Sort) algoritması böl ve yönet (divide-and-conquer) yöntemini kullanarak sıralama yapar. Dolayısıyla doğrusal

(lineer) bir algoritma değildir. Öte yandan birleştirerek sıralama algoritması özyinelemeli (Recursive) bir algoritmadır. Algoritma, bu yönleri ile hızlı sıralama (Quick Sort) algoritmasına benzer.

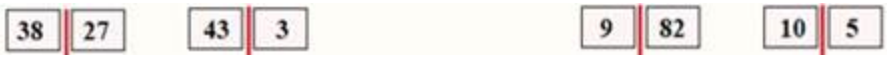
Birleştirmeli sıralama (Merge Sort) algoritması kullanılarak bir dizinin elemanlarını sıralarken ilk önce dizi elemanlarını küçük parçalara ayırmamız gerekmektedir (*sıralanacak dizi birer elemandan oluşan alt diziler elde edilene kadar özyinelemeli olarak ikiye bölünür*). Bu aşama, bölme (divide) aşamasıdır. Daha sonra bu parçalar birleştirilir. Birleştirme sırasında dizi elemanları sıralanır. Aşağıda birleştirmeli sıralama (Merge Sort) algoritması bir örnek üzerinden açıklanacaktır.

Sıralaması yapılacak örnek dizi:

1. Adım: Dizi önce ortadan ikiye bölünür. Eğer dizinin eleman sayısı tek sayı ise alt dizilerden solda kalan (n/2)+1 elemana sahip olur ve sağ tarafta kalan alt dizinin n elemanı olur. Örneğin 5 elemanı olan bir dizinin ikiye bölünmesi gerektiğinde bölme işlemi sonunda alt dizilerin eleman sayısı sırası ile 3 ve 2 olur.

Burada ele alınan örnekte sıralanacak dizinin sekiz elemanı vardır ve bu elemanlar ortadan ikiye bölündüğünde solda ve sağda kalan alt dizilerin eleman sayıları dört olur.

2. Adım: Birinci Adımdaki bölme işlemi sonunda elde edilen diziler tekrar ikiye bölünür.



3. Adım: İkinci adımdaki bölme işlemi sonunda elde edilen diziler tekrar ikiye bölünür. Bu örnekten de görülebileceği gibi sekiz elemanı olan bir diziyi üç adımda en küçük parçalara ayrılmış olur. Bir dizinin eleman sayısı

$$(2^n)$$

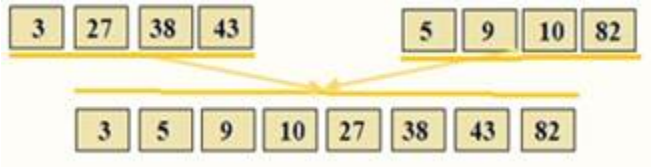
ile ifade edilirse bu dizinin en küçük parçalara ayrılması için gerekli olan adım sayısı n olarak verilir. Örneğin 32 elemanı olan bir dizinin en küçük parçalara ayrılması için gerekli adım sayısı beştir. Çünkü dizinin eleman sayısı olan 32 ikinin kuvveti şeklinde

olarak ifade edilir. Burada n=5 olduğu için bu diziyi en küçük parçalarına ayırmak için gerekli adım sayısı 5 ‘tir. Otuz iki elemanı olan bir dizi tanımlayınız ve bu diziyi bütün adımlarını göstererek en küçük parçalara ayırınız?

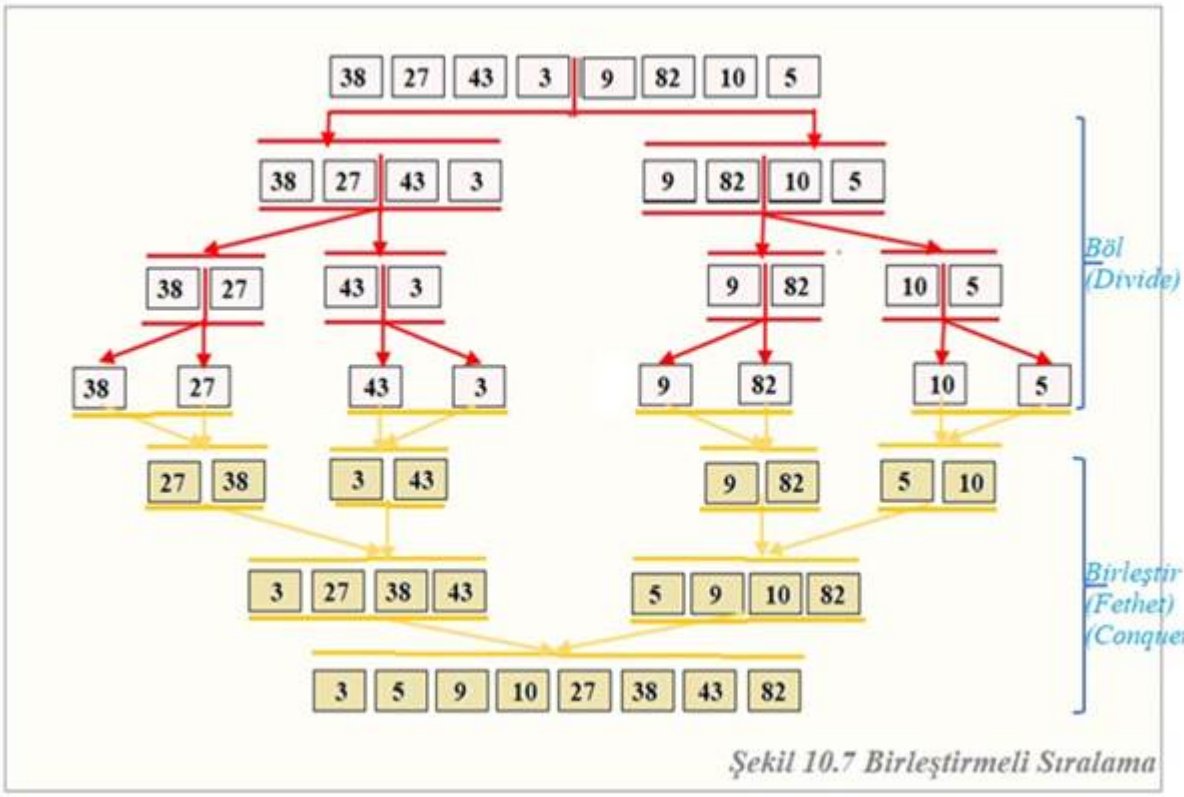
4. Adım: Üçüncü adımda en küçük parçalara ayrılan dizi elemanları ikişerli birleştirilir. Birleştirme yapılırken iki eleman kendi aralarında sıralanır.

5. Adım: Beşinci adımda, dördüncü adımda birleştirilerek oluşturulan ikişer elemanlı alt dizilerden sol baştaki ikisi kendi arasında sıralı olarak ve sonra takip eden iki dizi kendi arasında sıralı olarak birleştirilir ve dörder elemanı olan, kendi içerisinde sıralı iki dizi elde edilir.

6. Adım: Altıncı adımda, beşinci adımda elde edilen dörder elemanlı diziler karşılaştırılarak sekiz elemanlı sıralı bir dizi elde edilir.



Yukarıda açıklanan işlem adımları şekil 10.7 ‘de toplu olarak gösterilmiştir.



Program 10.5 ‘te Birleştirmeli sıralama algoritmasının C kodu verilmiştir.

```c
#include <stdio.h>
#include <locale.h>
void sirala(int dizi[],int ilk,int ort,int son){
    int i, j, k, l, temp[50];
     l = ilk;
     i = ilk;
     j = ort + 1;
    while ((l <= ort) && (j <= son)){
        if (dizi[l] <= dizi[j]){
            temp[i] = dizi[l];
            l++;
        }
        else{
            temp[i] = dizi[j];
            j++;
        }
        i++;
    }
    if (l > ort){
        for (k = j; k <= son; k++)
        {
            temp[i] = dizi[k];
            i++;
        }
    }
    else{
        for (k = l; k <= ort; k++)
        {
            temp[i] = dizi[k];
            i++;
        }
    }
    for (k = ilk; k <= son; k++){
        dizi[k] = temp[k];
    }}
```

①

```c
void birlestir(int dizi[],int ilk,int son){
    int orta;
     if(ilk < son){
        orta = (ilk + son) / 2;
        birlestir(dizi, ilk, orta);
        birlestir(dizi, orta + 1, son);
        sirala(dizi, ilk, orta, son);
    }
}
 int main(){
    int dizi[50];
    int i, boyut;
    setlocale(LC_ALL,"Turkish");
    printf("Dizinin Eleman Sayısı :");
    scanf("%d", &boyut);
    printf("Dizinin Elemanı        :\n");
    for(i = 0; i < boyut; i++){
         scanf("%d", &dizi[i]);
    }
    birlestir(dizi, 0, boyut - 1);
    printf("Birleştirmeli Sıralamadan Sonra:\n");
    for(i = 0;i < boyut; i++){
         printf("%d   ",dizi[i]);
    }
    return 0;
}
```

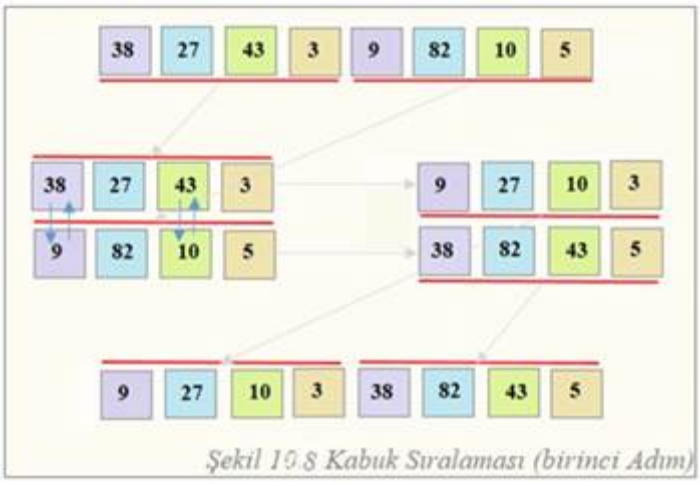
②

Program 10.5 Birleştirmeli Sıralama (Merge Sort)

10.5. Kabuk Sıralama (Shell Sort) Algoritması

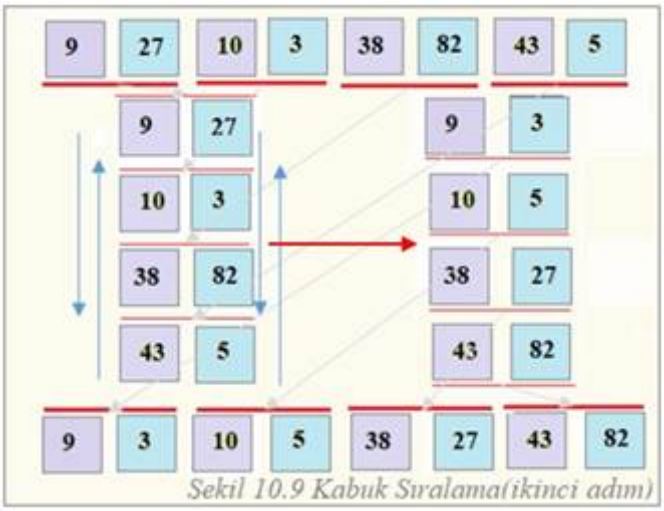
Kabuk sıralaması (Shell Sort) ismini algoritmayı ilk öneren kişiden (Donald Shell) alır. Kabuk sıralaması algoritması aslında başka bir sıralama algoritması (örneğin Kabarcık Sıralaması veya araya ekleme sıralaması vb.) üzerinde çalışan bir sıralama algoritmasıdır. Kabuk sıralaması algoritmasında, sıralanacak dizinin eleman sayısı önce ikiye bölünür (N/2). Sonra oluşan alt dizilerin aynı sıradaki elemanları karşılaştırılır. Sekiz elemanlı bir diziyi örnek olarak ele alacak olursak, diziyi ikiye böldüğümüzde, birinci sıradaki alt dizi dizinin ilk dört elemanından, ikinci alt dizi ise dizinin son dört elemanından oluşur.

Karşılaştırma işlemi için alt diziler alt alta yazılır ve karşılaştırma aynı kolandaki elemanların arasında yapılır. Şekil 10.8'de verilen dizi üzerinden hareket edecek olursak 38 ile 9, 27 ile 82, 43 ile 10 ve 3 ile 5 karşılaştırılır. Karşılaştırma sırasında aynı kolondaki elemanlar bilinen sıralama algoritmalarında biri (örneğin Kabarcık Sıralaması, Araya Yerleştirme Sıralaması vb.) ile sıralanır. Bu sıralama sonunda dizinin ilk dört eleman 9, 27, 10, 3 ve son dört eleman 38, 82, 43, 5 olur. Bkz. Şekil 10.8.



Şekil 10.8 Kabuk Sıralaması (birinci Adım)

Bu işlemin ardından dörder elemanı olan ve kısmen sıralanmış olan gruplar tekrar ikiye bölünür. Bu durumda ikişer elemanlı dört alt dizi oluşur. Bu alt dizilerin ilk elemanları birinci kolonu (9, 10, 38, 43), ikinci elemanları ikinci kolonu (27, 3, 82, 5) oluşturur (Bkz. Şekil 10.9). Kolanlar bir sıralama algoritması kullanılarak belirlenen koşula göre (küçükten büyüğe, büyükten küçüğe) kendi aralarında sıralanır.



Şekil 10.9 Kabuk Sıralama(ikinci adım)

Daha sonra mevcut gruplar tekrar ikiye bölünür. Burada ele alanına örnekte tekrar ikiye bölme işleminden sonra tek kolondan oluşan bir diziye ulaşılır. Kabuk sıralamasında tek kolonluk diziye ulaşıldığında bilenen sıralama algoritmalarından bir kullanılarak dizinin sıralanması tamamlanır. Biz örneğimizde sıralama işlemini Araya ekleyerek sıralama (Insertion Sort) algoritmasını kullanarak tamamladık.

Program 10.6 ‘da kabuk sıralamasına ait C programı verilmiştir.

```c
#include<stdio.h>
#include<locale.h>
void sirala(int a[],int n){
    int k,i,j,temp;
    for(k=n/2;k>0;k/=2) {
        for(i=k;i<n;i+=1){
            temp=a[i];
                for(j=i;j>=k&&a[j-k]>temp;j-=k)
                a[j]=a[j-k];

            a[j]=temp;
        }
    }
}
 int main(){
    setlocale(LC_ALL,"Turkish");
    int a[]={38,27,43,3,9,82,10,5},i,n=8;
    printf("\nSıralama İşleminden Önce Dizi:\n");
    for(i=0;i<n;++i)
        printf("%d ",a[i]);
    sirala(a,n);
    printf("\nSıralama İşleminden Sonra Dizi:\n");
    for(i=0;i<n;++i)
        printf("%d ",a[i]);
     return 0;
}
```

Program 10.6 Kabuk Sıralaması

Bölüm Özeti

Sıralama kavramını açıklayabilecek,

Sıralama, elektronik ortamlarda bulundurulacak verilerin kayıtlarının belirli bir düzende (sırada) saklanmasıdır. Sözü edilen bu düzen, veriler sayısal ise küçükten büyüğe veya büyükten küçüğe, alfa sayısal ise A ’ dan Z ‘ye veya Z’den A’ya olabilir. Bu konu verilerin bilgisayarlarda işlenmeye başlandığı ilk günden bu güne kadar hep önemli bir konu olmuş ve bunun için araştırmacılar konu üzerinde yoğun bir şekilde çalışmışlar ve birçok sıralama algoritması önermişlerdir. Bugün yaygın olarak kullanılmakta olan 25-30 sıralama algoritması olduğunu söyleyebiliriz. Bu algoritmaların kullanım alanlarına göre üstün yanları ve eksiklikleri vardır. Biz bu ünitede bu algoritmalardan yaygın olarak kullanılmakta olan Kabarcık Sıralaması (Bubble Sort), Seçerek Sıralama (Selection Sort), Araya Ekleme Sıralaması (Insertion Sort), Kabuk

Sıralaması (Shell Sort), Hızlı Sıralama ve Birleştirmeli Sıralama (Merge Sort) algoritmalarını ele aldık.

Kabarcık Sıralaması Algoritması ile verileri sıralayabilecek,

Kabarcık sıralaması (Bubble Sort) algoritması, dizinin her bir bitişik elemanın birbiri ile karşılaştırılması ve elemanlar sıralı değilse elemanların yerlerinin değiştirilmesi esasına dayanan bir sıralama algoritmasıdır. Kabarcık Sıralaması algoritması ilk önce Yer Değiştirme (Exchange Sort) algoritması olarak isimlendirilmiş, daha sonra dizi içerisindeki büyük elemanların algoritmanın her adımında dizi sonuna kadar doğrusal olarak ilerlemesinden dolayı algoritmaya Kabarcık Sıralaması(Bubble Sort) denmiştir Nabiyev, V.
(2016). *Algoritmalar.* Ankara: Seçkin.

Seçerek Sıralama Algoritması ile verileri sıralayabilecek,

Seçerek sıralama (Selection Sort) algoritması, sıralama problemine getirilen en basit çözümdür. Algoritmanın çalışması sırasında dizinin iki bölüme ayrıldığı varsayılır. Bu bölümlerden ilki sıralanmış dizi, ikincisi ise sıralanmamış dizidir. Başlangıçta sıralanmış dizi boş, sıralanmamış dizi ise, dizinin elemanlarının tamamından oluşur. Seçerek sıralama algoritmasının çalıştırılmasına, dizinin ilk elemanının seçilmesi ile başlanır. Seçilen dizi elemanı, bu elemandan sonra gelen dizi elemanları karşılaştırılır ve en küçük eleman bulunur. Bulunan bu en küçük eleman seçilen eleman ile yer değiştirir. Bu işlemin sonucunda dizinin en küçük elemanı belirlenerek dizinin başına yazılmış olur. Dizinin başına yazılmış olan bu eleman sıralanmış dizinin ilk elemanıdır. Sonraki adımlara, dizinin sıralanmış en son elemanını takip eden elemanın, dizinin geriye kalan elemanlarının karşılaştırılması ve küçük olan elemanla bu elemanın yer değiştirmesi ile işleme devam edilir.

Araya Ekleyerek Sıralama Algoritması ile verileri sıralayabilecek,

Araya ekleme sıralaması (Insertion Sort) algoritması, sıralama problemine getirilen en temel yaklaşımlardan biridir. Algoritmanın çalıştırılması dizinin sıralanacak elemanının seçilmesi, seçilen elemanın kendinden önceki elemanlarla karşılaştırılması, dizinin büyük

elemanlarının sağa kaydırılması, bu kaydırma sonucunda açılan uygun pozisyona o anda sıralanmakta olan elemanın yerleştirilmesi esasına dayanır.

Hızlı Sıralama Algoritması ile verileri sıralayabilecek,

Hızlı sıralama (Quick Sort) algoritması, oldukça verimli bir sıralama algoritmasıdır. Bu ünitede daha önce ele alınan üç sıralama algoritmasından farklı olarak böl ve yönet (divide-and-conquer) yöntemini kullanarak sıralama işlemini gerçekleştirir. Böl ve yönet algoritmaları, problemlerin mümkün olan en küçük alt parçalara ayrıldığı, her bir alt parçanın diğerlerinden bağımsız şekilde çözüldüğü algoritmalardır. Problemin genel çözümü elde edilirken alt parçalara ait çözümler belirli bir sırayla bir araya getirilir.

Birleştirmeli Sıralama Algoritması ile verileri sıralayabilecek,

Birleştirerek sıralama (Merge Sort) algoritması böl ve yönet (divide-and-conquer) yöntemini kullanarak sıralama yapar. Dolayısıyla doğrusal (lineer) bir algoritma değildir. Öte yandan birleştirerek sıralama algoritması özyinelemeli (Recursive) bir algoritmadır. Algoritma, bu yönleri ile hızlı sıralama (Quick Sort) algoritmasına benzer. Bir dizinin elemanları birleştirmeli sıralama algoritması kullanılarak sıralanırken ilk önce dizi elemanlarının küçük parçalara ayrılması gerekmektedir (*sıralanacak dizi birer elemandan oluşan alt diziler elde edilene kadar özyinelemeli olarak ikiye bölünür*). Bu aşama, bölme (divide) aşamasıdır. Daha sonra bu parçalar birleştirilir. Birleştirme sırasında dizi elemanları sıralanır.

Kabuk Sıralaması Algoritması ile verileri sıralayabilecek

Kabuk sıralaması (Shell Sort) algoritması, ismini algoritmayı ilk öneren kişiden (Donald Shell) almıştır. Bu algoritma aslında başka bir sıralama algoritması (örneğin Kabarcık Sıralaması veya araya ekleme sıralaması vb.) üzerinde çalışan bir sıralama algoritmasıdır. Kabuk sıralaması algoritmasında, sıralanacak dizinin eleman sayısı önce ikiye bölünür (N/2). Sonra oluşan alt dizilerin aynı sıradaki elemanları karşılaştırılır ve bu elemanlar kendi arasında başka bir sıralama algoritması kullanılarak

sıralanır. Dizinin ikiye bölünmesi ve sıralama işlemi, ikiye bölme işlemi sıfır çıkana kadar sürdürülür.

Kaynakça

1. Robert Sedgewick, Kevin Wayne, Algoritmalar, Nobel Yayınevi, 2018

2. Muhammed Mastar, Süha Eriş, C++, KODLAB Yayın Dağıtım Yazılım ve Eğitim Hizmetleri San. ve Tic. Ltd. Şti, 2012.

3. Nejat Yumuşak, M. Fatih Adak, C, C++ ile Veri Yapıları, Seçkin Yayıncılık, 2014.

4. Rıfat Çölkesen, Veri Yapıları ve Algoritmalar, Papatya Yayıncılık, 2002.

5. G. Murat Taşbaşı, İleri C programlama, Altaş Yayıncılık ve Elektronik Tic. Ltd. Şti, 2005.

6. https://tr.wikipedia.org/wiki/S%C4%B1ralama_algoritmas%C4%B1

7.https://www.tutorialspoint.com/data_structures_algorithms/quick_sort_program_in_c.htm