

SIRALAMA (SORT) VE ARAMA(SEARCH) ALGORİTMALARI

Düzensiz olarak verilmiş dizi elemanlarını küçükten büyüğe veya büyükten küçüğe doğru sıralamak, bir düzene koymak için çeşitli sıralama algoritmaları geliştirilmiştir. Sıralanan bilgilerde arama işlemleri de kolaylaşmaktadır. En çok kullanılan sıralama algoritmaları aşağıdaki algoritmalarlardır:

- 1.Bubble sort (kabarcık sıralama)
- 2.Shell sort (ikili bölerek sıralama)
- 3.Selection sort(seçerek sıralama)
- 4.Quick sort (hızlı sıralama)

1. BUBBLE SORT ALGORİTMASI

Dizide yer alan her eleman, sırayla kendisinden sonra gelen elemanla karşılaştırılır ve gerekiyorsa yer değiştirilir. Sıralama işlemi yer değiştirme olduğu sürece devam eder. Herhangi bir sıralama yönteminin etkinliğini analiz ederken kaç kez karşılaştırma ve yer değiştirme (takas) yapıldığına bakılır. Kabarcık sıralamada karşılaştırma sayısı daima aynıdır. Verilen dizi sıralı olsa bile kullanılan iki **for** döngüsü belirtilen sayıda tekrar yapar. Bu da yöntemin $(n^2 - n)/2$ karşılaştırma yapacağı anlamına gelir. Buradaki n sıralanacak eleman sayısıdır. Bu formüle dıştaki döngünün **n-1** kez, içteki döngünün de ortalama **n/2** kez çalıştığı düşünülerek varılmıştır. Kabarcık sıralamanın bir n^2 algoritması olduğu söylenmektedir. Çünkü çalışma süresi, sıraladığı eleman sayısının karesiyle orantılıdır. Dolayısıyla dizi elemanlarının sayısının fazla olması durumunda işlem zamanı fazlaşan bir algoritmadır. Sırası tamamen bozuk bir veri yapısı için, yani en kötü durum için bu algoritmanın Büyük-O Notasyonu (Big-O Notation) **$O(n^2)$** dir.

$$(n-1) + (n-2) + \dots + 2 + 1 = n*(n-1)/2 = O(n^2)$$

Özel bir durum olarak, eğer verinin belli kısmı sıralı ise, n adet verinin c tanesi sıralı değil ise algoritmanın karmaşıklığı $O(cn)$ olmaktadır.

Örnek 1.1: Buble Sort ile küçükten büyüğe random elemanlara sahip dizi sıralama C++ kodu

```
#include <iostream>

using namespace std;

//----- bubbleSort -----
//
// Amaç      : Verilen değerlerin, küçükten büyüğe sıraya koymak
//
// Giriş parametresi : Değerlerin yazılı olduğu vektör
//
// Dönüş değeri : void
//
// Fonksiyonda sıralama işlemi, Bubble Sort algoritması ile
// gerçekleştirilmiştir.
//
//-----
```

EEM 227 Veri Yapıları - 4.Hafta Ders Notları

```
void bubbleSort ( int dizi[],int size)
{
    int temp;
    // for her dönüşünde, sıraya sokulmamış elemanların ilkinden başlayarak
    // her elemanı sağdaki komşusu ile karşılaştırıyor. Eğer komşusundan
    // daha büyük bir değere sahipse, onunla yer değiştiriliyor.
    for (int i = size; i > 2; i --)
    {
        // Buradaki i, sıraya sokulmamış eleman sayısını gösteriyor, yani
        // (i-1). terime kadar olan elemanlar henüz sıralı değiller.
        // Sıraya girmemiş elemanların herbiri, sağındaki komşusu ile
        // karşılaştırılıyor. Eğer değeri komşusundan büyükse, onunla yer
        // değiştiriliyor. Sıraya sokulmamış elemanların değerice en büyük
        // olanı, (i-1). terime kadar tüm elemanlar ile yer değiştireceği
        // için, sıralı elemanların başına kadar ulaşır. Daha önceden
        // sıraya girmiş elemanların her biri zaten bu elemndan daha
        // büyük bir değere sahip olduğu için, bu eleman da artık sıralıdır.
        for (int j = 0; j < i - 1; j ++)
        {
            // Komşusu ile yer değiştiriyor. (swap)
            if (dizi[j] > dizi[j + 1])
            {
                temp = dizi[j + 1];
                dizi[j + 1] = dizi[j];
                dizi[j] = temp;
            }
        }
    }
}

//----- main -----

int main ()
{
    int d[20];

    // 20 elemanlı, rasgele bir katar oluşturuluyor.
    for (int i = 0; i < 20; i ++)
        d[i]=(rand () % 1000);
    bubbleSort (d,20);

    for (int i = 0; i < 20; i ++)
        cout<<d[i]<<" ";
    cout<<endl;
    system("pause");
    return 0;
}
```

Örnek 1.2 : Buble Sort C++ kodu (Vektor Template Kullanılarak)

```
#include <iostream>
#include <vector>
using namespace std;

template <class T>
void bubbleSort (vector<T>& dizi)
{
    for (int i = dizi.size (); i > 2; i --)
    {
        for (int j = 0; j < i - 1; j ++)
        {
            if (dizi[j] > dizi[j + 1])
            {
                T temp = dizi[j + 1];
                dizi[j + 1] = dizi[j];
                dizi[j] = temp;
            }
        }
    }
}

//----- main -----

int main ()
{
    vector<int> D;

    // 20 elemanlı, rasgele bir katar oluşturuluyor.
    for (int i = 0; i < 20; i ++) D.push_back (rand () % 10);
    bubbleSort (D);
    return 0;
}
```

2.SELECTION SORT ALGORİTMASI

Seçerek sıralamada, eğer dizi küçükten büyüğe sıralanacaksa, ilk eleman alınır ve geri kalanlar arasında en küçük değerli eleman aranır eğer bulunursa ilk elemanla yer değiştirilir. Daha sonra kalan n-1 elemandan en küçük olan bulunur ve ikinci elemanla yer değiştirilir. İşlem bu şekilde devam eder. Bu yöntem de $(n^2 - n)/2$ sayıda karşılaştırma gerektirir. Kabarcık sıralamaya göre karşılaştırma sayısı aynı olsa bile ortalama yer değiştirme sayısı daha azdır.

$$\begin{aligned} T(n) &= 2*(n-1) + 2*(n-2) + 2*(n-3) + \dots + 2*(n-(n-2)) + 2*(n-(n-1)) \\ &= 2*((n-1) + (n-2) + (n-3) + \dots + 2 + 1) \text{ (or } 2*(\text{sum of first } n-1 \text{ ints}) \\ &= 2*((n-1)*n)/2 = n^2 - n \end{aligned}$$

Dolayısıyla bu algoritmanın karmaşıklığı, $O(n^2)$ dir.

Örnek 2.1: Selection Sort Algoritması ile küçükten büyüğe dizi sıralama

```
#include<stdio.h>
/* selection sort ile diziyi kucukten buyuye dogru siralar */
int main(void)
{
    int d[10];
    int i,j,n,min,yer;
    printf(" dizinin eleman sayisini < 10 giriniz \n");
    scanf("%d",&n);
    printf(" dizi elemanlarini giriniz \n");
    for (i=0;i<n;i++)
        scanf("%d",&d[i]);
    for(i=0;i<n-1;i++)
    {
        yer=i;
        min=d[i];
        for(j=i+1;j<n;j++)
            if(d[j]<min)
            {
                yer=j;
                min=d[j];
            }
        d[yer]=d[i];
        d[i]=min;
    }
    printf(" sirali dizi \n\n");

    for(i=0;i<n;i++)
        printf("%4d",d[i]);

    return 0;
}
```

Örnek 2.2: Selection Sort (Vektör Template)

```
#include <iostream>
#include <vector>
using namespace std;

//----- selectionSort -----
//
// Amaç      : Verilen değerlerin, küçükten büyüğe sıraya koymak
//
// Giriş parametresi : Değerlerin yazılı olduğu vektör
//
// Dönüş değeri : void
//
// Fonksiyonda sıralama işlemi, Selection Sort algoritması ile
// gerçekleştirilmiştir.
//
//-----
template <class T>
void selectionSort (vector<T>& dizi)
{
    int minIndex;

    // for her dönüşünde, sıraya sokulmamış elemanlar içerisinde en
    // küçük olanını buluyor ve bu değerın vektör üzerindeki yerini
    // minIndex isimli değişkende saklıyor.
```

EEM 227 Veri Yapıları - 4.Hafta Ders Notları

```
for (int i = 0; i < dizi.size () - 1; i ++)  
{  
    minIndex = i;  
  
    // vektörün sıraya sokulmamış kısmında (i+1'den itibaren),  
    // en küçük değerli eleman bulunuyor. İlk olarak, sıralanmamış  
    // kısımdaki ilk elemanın en küçük olduğu varsayılıyor. Bu  
    // elemandan itibaren diğer elemanlara bakılıyor. Eğer daha  
    // küçük bir değer ile karşılaşılsa, artık bu elemanın yeri  
    // minIndex isimli değişkene atanıyor. Böylece, sıralanmamış  
    // kısımdaki en küçük eleman bulunmuş olunuyor.  
    for (int j = i + 1; j < dizi.size (); j ++)  
        if (dizi [minIndex] > dizi [j]) minIndex = j;  
  
    // swap işlemi  
    // bulunan bu yeni eleman ile sıraya sokulmamış ilk eleman  
    // yer değiştiriliyor. Zaten ikiside aynı eleman ise, bu  
    // değişime gerek yoktur.  
    if (minIndex != i)  
    {  
        T temp = dizi[i];  
        dizi[i] = dizi[minIndex];  
        dizi[minIndex] = temp;  
    }  
}  
  
//----- main -----  
  
int main ()  
{  
    vector<int> D;  
  
    // 20 elemanlı, rasgele bir katar oluşturuluyor.  
    for (int i = 0; i < 20; i ++)  
        D.push_back (rand () % 10);  
  
    selectionSort (D);  
  
    return 0;  
}
```

3. QUICK SORT ALGORİTMASI

Geliştirilmiş sıralama algoritmalarının en iyisi olarak bilinen bir algoritmadır. Genel işleyişi, karşılaştırma değeri (comparand) adı verilen bir değer seçilir ve sonra dizi ikiye ayrılır. Karşılaştırma değerinden büyük ya da eşit olanlar bir tarafta, küçük olanlar diğer tarafta yerleştirilir. Bu işlem geriye kalan her bölme üzerinde tekrarlanır. Karşılaştırma değeri ya rasgele seçilir ya da dizideki az sayıda elemanın değerinin ortalaması alınır. Quick sort algoritması için ortalama karşılaştırma sayısı $n \log n$ ve ortalama yer değiştirme sayısı $n / 6 \log n$ olarak verilebilir. Yöntemde dikkat edilmesi gereken husus karşılaştırma sayısının iyi seçilmesidir. Her bölüm için karşılaştırılan değer en büyük değer ise yöntem yavaşlayacaktır. Verilerin yapısına göre uygun bir sıralama algoritmasının seçilmesi isabetli olacaktır. Quick sort algoritması en etkin algoritma olmasına rağmen az sayıda elemana sahip dizilerde tekrarlanan çağrılarından ötürü randımanlı olmayabilir. Bu tip dizilerde (eleman sayısı 100 veya daha az) Bubble sort veya selection sort daha iyi sonuç verecektir.

Örnek 3.1: Quick Sort ile katar sıralama

```
#include<iomanip>
/*karakter dizisini quick sort ile siralar */
#include<stdio.h>
#include<string.h>
void quick_str(char dizi[][10],int say);
void qs_string(char dizi[][10],int sol,int sag);
char str[][10]={"tulay","pelin","aysu","can","beril","ogul"};
int main(void)
{

    int i;
    quick_str(str,sizeof(str));
    printf("siralanmis dizi\n");
    for (i=0;i<sizeof(str);i++)
        printf("%s\n",str[i]);
    system("pause");
    return 0;

}

void quick_str(char dizi[][10],int say)
{
    qs_string(dizi,0,say-1);
}

void qs_string(char dizi[][10],int sol,int sag)
{
    int i,j;
    char *x;
    char yedek[10];
    i=sol;j=sag;
    x=dizi[(sol+sag)/2];
    do {
        while((strcmp(dizi[i],x)< 0) && (i< sag))i++;
        while((strcmp(dizi[j],x)> 0) && (j > sol))j--;
        if(i <= j)
        {
            strcpy( yedek,dizi[i]);
            strcpy( dizi[i],dizi[j]);
            strcpy( dizi[j],yedek);
            i++; j--;
        }
    }while(i <= j);
    if(sol < j) qs_string(dizi,sol,j);
    if(i < sag) qs_string(dizi,i,sag);
}
```

Örnek 3.2: Vektor Template Quick Sort

```
#include <iostream>
#include <vector>
using namespace std;

//----- quickSort -----
//
// Amaç      : Verilen değerleri, küçükten büyüğe doğru sıraya koymak
//
// GirSonş parametresi : Değerlerin yazılı olduğu vektör
//
//-----

template <class T>
void quickSort (vector<T>& dizi)
{
    quickSort (dizi, 0, dizi.size () - 1);
}

//----- quickSort -----
//
// Giriş parametreleri : Değerlerin yazılı olduğu vektör. Bu vektörün
//                        hangi aralığı ile ilgilendiğimizi gösteren
//                        sol ve sag isimli iki değişken. Bu aralık
//                        içerisinde bir pivot seçilip, bu aralık da
//                        iki parçaya bölünecek.
//
//-----

template <class T>
void quickSort (vector<T>& dizi, int sol, int sag)
{
    int x = sol;
    int y = sag;
    T pivot = dizi[x]; // Verilen aralığın ilk elemanı pivot olarak
    seçiliyor

    // Pivottan büyük elemanların sağ tarafa, pivottan küçük
    // elemanların ise sol tarafa alınması sağlanıyor. En sonda
    // ise pivot, oluşan yeni iki gurubun ortasına yerleşecek.
    while (x < y)
    {
        // Sağda pivottan küçük ilk elemanı bul
        while ((dizi[y] >= pivot) && (x < y)) y --;

        // Bu elemanı sola al
        if (x != y)
        {
            dizi[x] = dizi[y];
            x ++;
        }

        // Solda pivottan büyük ilk elemanı bul
        while ((dizi[x] <= pivot) && (x < y)) x ++;

        // bu elemanı sağa al
        if (x != y)
        {
            dizi[y] = dizi[x];

```

EEM 227 Veri Yapıları - 4.Hafta Ders Notları

```
        y --;
    }
}

dizi[x] = pivot; // pivot buraya taşındı

int pivotIndis = x; // pivotun yeni yeri

// Pivottan küçük elemanlar [sol,pivotIndis-1] aralığına yerleşti.
// Pivottan büyük elemanlar [pivotIndis+1,sag] aralığına yerleşti.
// Her iki aralık değeri için, eğer aralığın boyu birden büyükse,
// quickSort fonksiyonu özyinelemeli olarak yeniden çağırılıyor

if (sol + 1 < pivotIndis)
    quickSort (dizi, sol, pivotIndis - 1); // pivotun sol tarafı

if (sag - 1 > pivotIndis)
    quickSort (dizi, pivotIndis+1, sag); // pivotun sağ tarafı
}

//----- main -----

int main()
{
    vector<int> D;

    // 100 elemanlı, rasgele bir katar oluşturuluyor.
    for (unsigned int i = 0; i < 100 ; i ++) D.push_back (rand ()%10);

    quickSort (D);

    return 0;
}
```

4.SHELL SORT ALGORİTMASI

Bu sıralama, algoritmayı geliştiren D.L.Shell'in adını almasına karşın, bu isimle anılması işleyiş şeklinin üst üste yığılmış deniz kabuklarına(shell) benzemesindendir. n adet elemanı sıralamak için yöntemin çalışma süresinin $n^{1.2}$ ile orantılı olduğu söylenebilir. Bu durumda shell sort algoritmasının n -kare algoritmalarına göre iyileştirilmiş bir algoritma olduğu görülmektedir.

ARAMA ALGORİTMALARI

Sıralanmamış bir dizide bilgi aramak iki şekilde mümkündür. Ya ilk elemandan başlayarak eşiti bulunduğu sona eren ya da dizinin sonuna gelindiğinde aramayı bitiren bir sıralı arama (sequential search) yapılır. Sıralı bir dizi kullanılıyorsa ikili arama (binary search) algoritması ile daha kısa zamanda sonuç alınabilmektedir.

SIRALI ARAMA

Eleman sayısı bilinen bir dizide istenilen bilgi bulununcaya kadar arama yapan basit bir algoritmadır.

```
# include <stdio.h>
# include <stdlib.h>
/* sirali arama yapar */
int sirali_ara (int a[],int n)
{
    int sonuc,x,j;
    printf("aranacak sayiyi Giriniz:\n");
    scanf("%d",&x);
    for(j=0;j<n;j++){
        if(a[j]==x){
            sonuc=j;
            break;
        }
        else sonuc=-1;
    }
    return(sonuc);
}
int main (void)
{
    int n,i,dizi[10],x,bul;
    printf("dizinin eleman sayisini giriniz < 10:\n");
    scanf("%d",&n);
    printf("dizi elemanlarini Giriniz:\n");
    for (i=0;i<n;i++)
        scanf("%d",&dizi[i]);
    bul=sirali_ara(dizi,n);
    if(bul== -1)printf(" aranan sayi dizide yer almiyor\n");
    else
        printf(" aranan sayi dizinin %d . elemanidir",bul);
    return 0;
}
```

İKİLİ ARAMA

Bu yöntemde önce **sıralı dizide** tam ortadaki eleman alınır aranan sayı ile karşılaştırılır. Aranan sayı ortadaki elemandan büyükse, tekrar ikinci yarının orta elemanı bulunarak karşılaştırma yapılır. İşlem aranan sayı bulununcaya kadar devam eder. İkili aramada en kötü durumda karşılaştırma sayısı $\log_2 n$ dir.

```
#include <stdio.h>
/* binary search algoritmasi ile sirali dizide arama yapar */
int main (void)
{
    int a[10],i,bas,son,x,orta,bul,n;
    printf(" dizinin eleman sayisini < 10 giriniz \n");
    scanf("%d",&n);
    printf("Diziyi sirali olarak giriniz\n");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf("\n aranacak sayiyi giriniz\n");
    scanf("%d",&x);
    bas=0;son=n;
    bul=0;
```

```
while(bas <= son && bul==0){
    orta=(bas+son)/2;
    if(x==a[orta]) bul=1 ;
    else
        if(x>a[orta]) bas=orta+1;
        else
            son=orta-1;
    }
    if(x==a[orta]) printf("aranan sayı%3d dizideki indisi:%3d",x,orta);
    else printf("bulunamadı");
scanf("%d",&x); // ekranı durdurma
return 0;
}
```

MERGE(BİRLEŞTİRME)ALGORİTMASI

Girilen iki sıralı diziyi birleştiren bir algoritmadır.

```
# include <stdio.h>
/* merge algoritmasi ile iki sirali diziyi birlestirir */
// Not Her iki girilen dizide kendi içinde sıralı olmalıdır...

int main (void)
{
    int a[10],b[10],d[20];
    int n,m,i,j,l;
    printf("ilk dizinin boyutu < 10 ?\n");
    scanf("%d",&n);
    printf("ikinci dizinin boyutu <10 ?\n");
    scanf("%d",&m);
    printf(" birinci dizinin elemanlari\n ");
    for(i=0;i<n;i++)
        scanf("%d",&a[i]);
    printf(" ikinci dizinin elemanlari\n ");
    for(i=0;i<m;i++)
        scanf("%d",&b[i]);
    a[n]=b[m-1];
    b[m]=a[n-1];
    i=0;
    j=0;
    for(l=0;l<n+m;l++)
        if(a[i]<=b[j]){
            d[l]=a[i];
            i++;
        }
        else
        { d[l]=b[j];
          j++;
        }
    printf(" birlestirilmis dizi \n\n" );
    for(i=0;i<n+m;i++)
        printf("%5d",d[i]);
    scanf("%d",&n); // ekranı durdurmak için
    return 0; }
```