Veri Yapıları

Sıralama ve Arama Algoritmaları

Seçmeli Sırlama Algoritması

Sıralama ve arama tekniklerinden pek çok program tarafından yararlanılmaktadır.

Sıralama genelde arama işlemlerinin hızlı bir şekilde gerçekleşmesi için kullanılır.

Not:Fibonacci sayı dizisi ile ayçiçeğinin çekirdekleri aynı şekilde sıralanır.

Seçmeli Sırlama Algoritması

```
#include <iostream.h>
#define n 10
main(){
int dizi[n]={100,-250,400,125,550,900,689,450,347,700};
          int enkucuk;

for (int i = 0; i < dizi.Length - 1; i++) {
        enkucuk = i;
        for (int j = i + 1; j < dizi.Length; j++)
            if (dizi[j] < dizi[enkucuk])
                enkucuk = j;

        if (enkucuk != i) {
            yedek = dizi[i];
            dizi[i] = dizi[enkucuk];
            dizi[enkucuk] = yedek;
        }
 }
```


Kabarcık Sıralama (Bubble Sort)

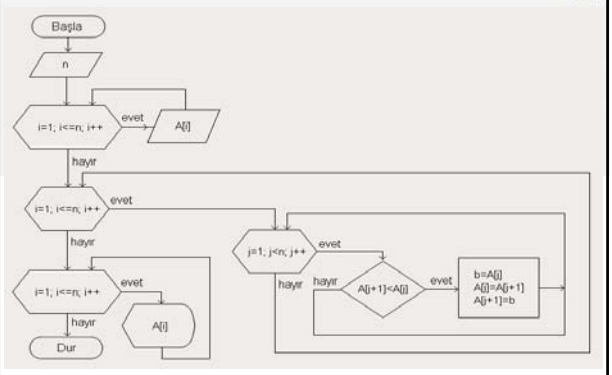
Sıralanacak veri kümesinde elemanların ardışık olarak birbirleri ile karşılaştırılması ile gerçekleştirilir.

Karşılaştırmalar ikişerli olarak yapılır ve büyük değer bir sonraki indise geçer. (1. ve 2. , 2. ve 3……)

En kötü durum performansı (worst-case performance) O(n2) 'dir.
En iyi durum performansı (best-case performance) O(n) 'dir.

En kötü durum performansında N<1000 olduğu sürece hız anlamında sorun yaşatmayacaktır.

Kabarcık Sıralama (Bubble Sort)



Kabarcık Sıralama

```
#include <iostream.h>
#define n 10
main(){
int a[n]={100,-250,400,125,550,900,689,450,347,700};
int   j,k,gecici;

            for(k=0; k<n-1; k++)
                for(j=0; j<n-1; j++)
                 if( a[j]<a[j+1] ){
                          gecici = a[j];10:
                                a[j] = a[j+1];
                   a[j+1] = gecici;
                 }


system("pause");
}
```

Eklemeli sıralama (insertion sort)

eleman kümesinin 2. elemanından başlayarak kendinden önceki elemanlarla karşılaştırma yapar.

Karşılaştırma yaptığı elemanlar kendinden büyükse bu elemanlar, küme içerisinde sağa doğru kaydırılır.

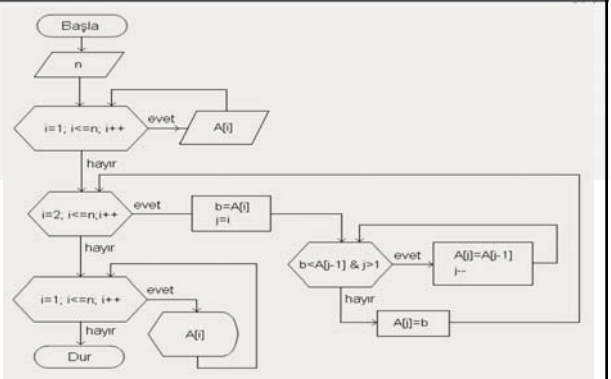
Ve seçili eleman uygun yere yerleştirilir.

Elemanların kaydırılması ile kaybedilen süre algoritmanın dezavantajıdır.

En kötü durum performansı (worst-case performance) O(n2) 'dir.
En iyi durum performansı (best-case performance) O(n) 'dir.

(N<1000) için sorun yaşamayacağımızı görüyoruz.

Eklemeli sıralama (insertion sort)



Eklemeli sıralama (insertion sort)

```
#include <iostream.h>
#define n 10
main() {
int   A[n]={100,-250,400,125,550,900,689,450,347,700};
int  i, j,k,gecici;
for (i=1; i<n; i++){
        j=i;
        gecici =A[j];
        while (j>0 && A[j-1]>gecici){
                A[j]=A[j-1];
                j--;
        }
        A[j]= gecici;
}

system("pause");
}
```

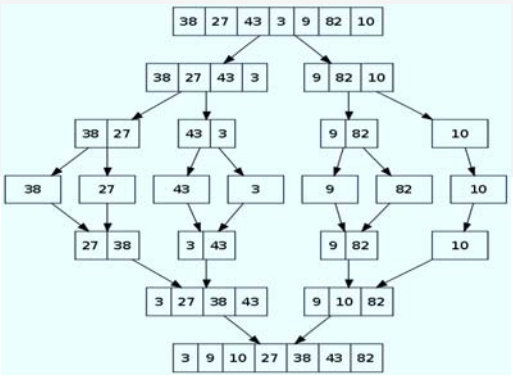

Birleştirmeli sıralama (Marge sort)

Merge Sort Algoritması, John von Neumann tarafından 1945 yılında yayınlanmıştır. Türkçeleştirildiğinde "Birleştirmeli Sıralama" anlamı çıkmaktadır.

Eleman kümesini (array/dizi) en küçük duruma getirene kadar sürekli ikili parçalara ayırır.
Küçük parçalara ayırdığı bu parçaları kendi aralarında sıralayarak bütünü elde eder ve sıralamayı yapmış olur.
Recursive fonksiyon kullandığımızda kolay çözebiliriz.

Durum performansı O(n) ile O(n2) arasındadır.
En kötü durum performansı O(n log n)'dir.
En iyi durum performansı O(n)'dir.
Ortalama durum performansı O(n log n) kabul edilir

Birleştirmeli sıralama (Marge sort)



Birleştirmeli sıralama (Marge sort)

```
int[] Sirala(int[] Dizi){
if (Dizi.Length < 2)
{
  return Dizi;
}

int Uzunluk = Dizi.Length / 2;
int KalanUzunluk = Dizi.Length - Uzunluk;
int deger1 = 0, deger2 = 0, deger3 = 0, sinir = 0;
int[] EkDizi = new int[Uzunluk];
int[] YedekDizi = new int[KalanUzunluk];

for (int i = 0; i < Uzunluk; i++)
{
  EkDizi[i] = Dizi[i];
}

for (int k = Uzunluk; k < Dizi.Length && sinir < KalanUzunluk; k++,
sinir++)
{
  YedekDizi[sinir] = Dizi[k];
}

int[] siraliEkDizi = Sirala(EkDizi);
int[] siraliYedekDizi = Sirala(YedekDizi);

while (deger1 < Uzunluk && deger2 < KalanUzunluk)
{
  if (siraliEkDizi[deger1]  siraliYedekDizi[deger2])
  {
    Dizi[deger3] = siraliYedekDizi[deger2];
    deger2++;
    deger3++;
  }
}

while (deger1 < Uzunluk)
{
  Dizi[deger3] = siraliEkDizi[deger1];
  deger3++;
  deger1++;
}

while (deger2 < KalanUzunluk)
{
  Dizi[deger3] = siraliYedekDizi[deger2];
  deger3++;
  deger2++;
}

return Dizi;
}

static void Main(string[] args)
{
  int[] Dizi = new int[] { 6, 5, 3, 1, 8, 7, 2, 4 };
  Dizi = Sirala(Dizi);
}
```

Heap Sort

Ağaçlar konusu ile beraber anlatılacak

Hızlı Sıralama (Quick Sort)

Quick Sort, eleman kümesi içerisinde seçilen pivot sayının diğer sayılarla karşılaştırılarak büyüklük ve küçüklük durumuna göre pivot sayının sağ ve sol tarafına yerleştirilmesi sonucu oluşan sıralama algoritmasıdır. Quick sort algoritmasını Türkçe kaynaklarda 'Hızlı Sıralama Algoritması' olarak görebilirsiniz.

Quick sort algoritması, sürekli olarak aynı mantık ile eleman kümesinden pivot sayı seçip karşılaştırma yaptığı için recursive fonksiyon ile çözüme kavuşturulması daha uygundur.

En kötü durum performansı (worst case performance) O(n2)'dir

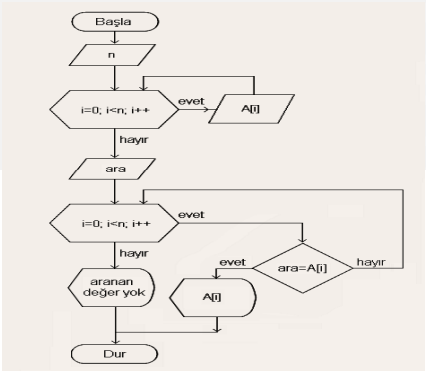

```
static void Main(string[] args)
{
int[] dizi = new int[] { 6, 5, 3, 1, 8, 7, 2, 4 };
siralamayap(dizi, 0, dizi.Length - 1);
}

public static int parca(int[] dizi, int sol, int
sag)
{
int i = sol, j = sag;
int gecici;
int pivot = dizi[(sol + sag) / 2];

while (i <= j)
{
while (dizi[i] < pivot) i++;
while (dizi[j] > pivot) j--;
if (i <= j)
{
gecici = dizi[i];
dizi[i] = dizi[j];
dizi[j] = gecici;
i++;
j--;
}
};
return i;
}

public static void siralamayap(int[] dizi, int
sol, int sag)
{
int index = parca(dizi, sol, sag);
if (sol < index - 1) siralamayap(dizi, sol,
index - 1);
if (index < sag) siralamayap(dizi, index,
sag);
}
```


Doğrusal Arama (Linear Search)



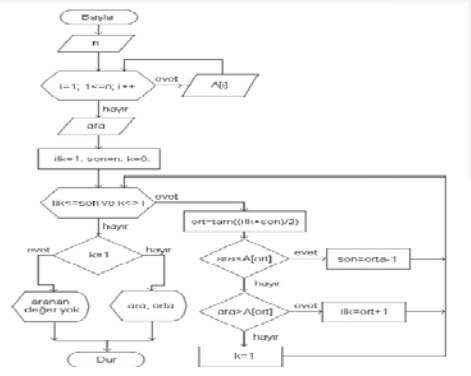
Doğrusal Arama (Linear Search)

```
#include <iostream.h>
#define n 10
main(){
      int n=10,ara;
      bool bulundu=false;
      int A[10]={100,-250,400,125,550,900,689,450,347,700};

      cout<<"aranan deger: ";
      cin>>ara;

      for(int i=0;i<n;i++){
        if(ara==A[i]){
          cout<<"deger: "<< A[i]<< "sirasi: "<< i+1;
          bulundu=true;
          break;
        }
      }
      if(!bulundu) cout<<"aranan deger yok";
      system("pause");
}
```


İkili (Binary) Arama Algoritması



İkili (Binary) Arama Algoritması

```
main(){
      int ilk=0, son=n-1,ort,ara;
      int A[10]={-250,100,125,347,400,450,550,689,700,900};
      cout<<"aranan deger: "; cin>>ara;
      bool bulundu=false;
      while(ilk<=son && bulundu==false){
         ort=(ilk+son)/2;
         if(ara<A[ort])
            son=ort-1;
         else if(ara>A[ort])
            ilk=ort+1;
         else
            bulundu=true;
      }
      if(bulundu)
            cout<<"deger:"<<A[ort]<<"sirasi: "<<ort+1;
      else
            cout<<"aranan deger bulunamadı";
      system("pause");
}
```


İyi çalışmalar…