C# Dilinde Random Sınıfı

Rastgele sayı üretmek için kullanılan bir sınıftır. Sayı üretebilmek için Random sınıfından bir nesne türetilmesi gerekir. Random sınıfına ait Next() metodu ile sayılar üretilmeye başlar. Next metodunun üç farklı kullanımı vardır.

1) Next metodu aşağıdaki gibi kullanılırsa negatif olmayan rastgele sayılar üretir.

```
Random s = new Random();
Console.Write(s.Next());
```

2) Parantez içerisindeki sayı üst sınır olmak üzere *(üst sınır hariç)* negatif olmayan sayı üretir.

```
Random s = new Random();
Console.Write(s.Next(15));
```

3) Parantez içerisinde alt ve üst sınır parametre olarak girilir. Alt sınır dahil, üst sınır hariç olacak şekilde negatif olmayan sayılar üretir.

```
Random s = new Random();
Console.Write(s.Next(5, 15));
```

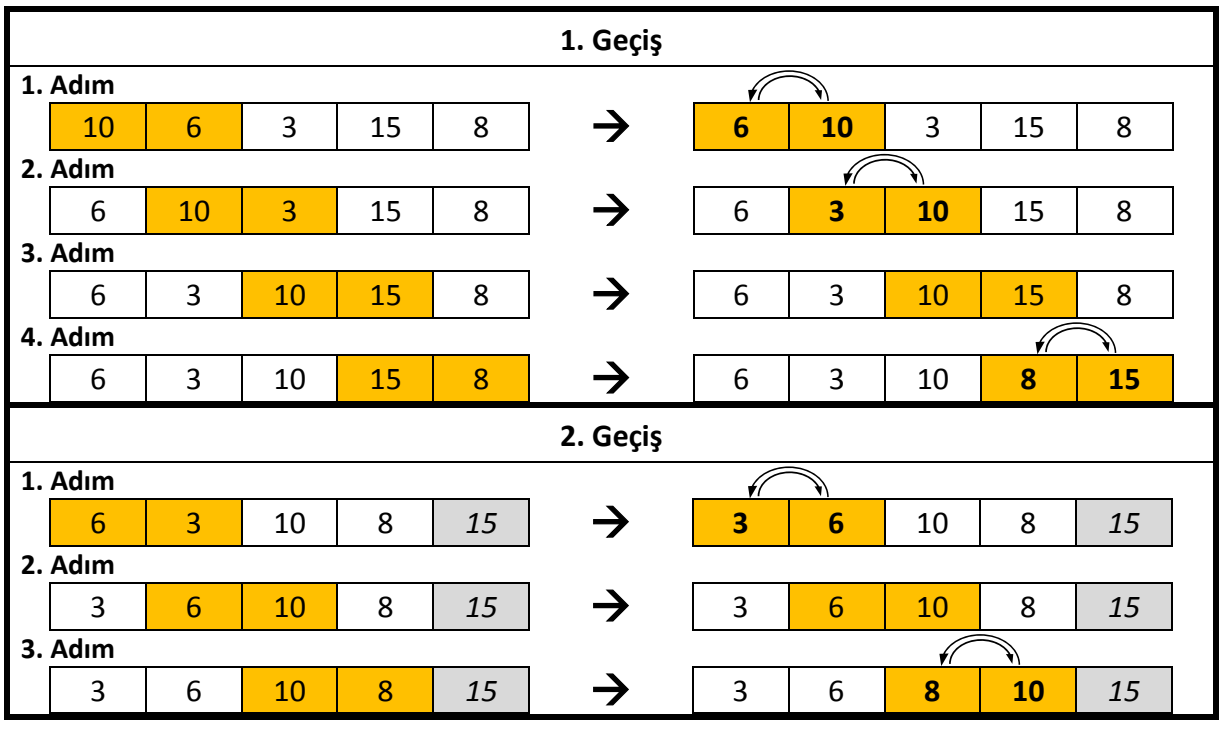
Nesne Tabanlı Programlama I – 01.12.2017 *(Vize Sonrası İlk Ders)*

C# Dilinde Sıralama Algoritmaları

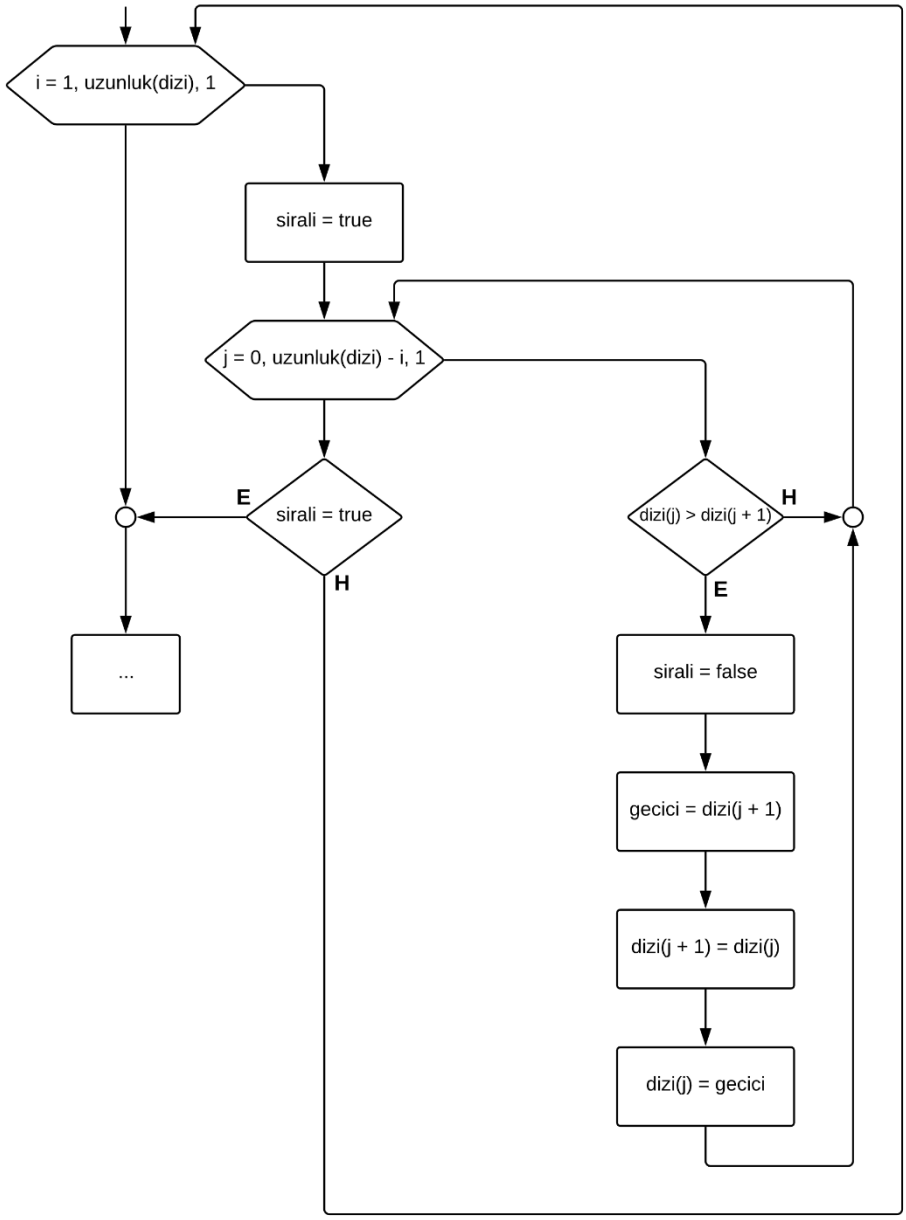
1) Kabarcık Sıralama (Bubble Sort) Algoritması

Her defasında listenin iki elemanını karşılaştırarak gerekliyse yer değişikliği yapılır. Bu işlem hiçbir yer değişikliği yapılmadığı ana kadar devam eder. Her geçişte en büyük sayı listenin en sağına aktarılmış olur. Kabarcık sıralaması algoritmasında yapılan işlem listeyi küçükten büyüğe doğru sıralamaktır.

Örnek: Liste = { 10, 6, 3, 15, 8 } listesini kabarcık sıralaması algoritmasını kullanarak çözünüz.



Algoritmanın çalışma prensibi dikkatle incelendiğinde ikinci geçişin sonunda sıralamanın gerçekleştiği görülmektedir. Kodlama yapılırken sıralamanın gerçekleşip gerçekleşmediğini kontrol ederek gereksiz adımların önüne geçebiliriz. Kod bloğundaki kabarcık sıralama metodunda sıralı isimli değişken bu amaçla kullanılmıştır. Listenin küçükten büyüğe sıralanması için uygulanan kabarcık sıralama metodu aşağıda akış diyagramı çizilerek gösterilmiştir.



Yukarıdaki akış diyagramının kodlamasını C# dilinde yazalım;

```
static void Main(string[] args)
{
   int [] sayilar = { 10, 6, 3, 15, 8 };
   diziYaz(sayilar);
   kabarcikSirala(sayilar);
   diziYaz(sayilar);
   Console.ReadKey();
}

public static void kabarcikSirala(int [] dizi)
{
   int gecici;
   bool sirali;

   for (int i = 1; i < dizi.Length; i++)
   {
      sirali = true;
      for (int j = 0; j < dizi.Length - i; j++)
      {
```

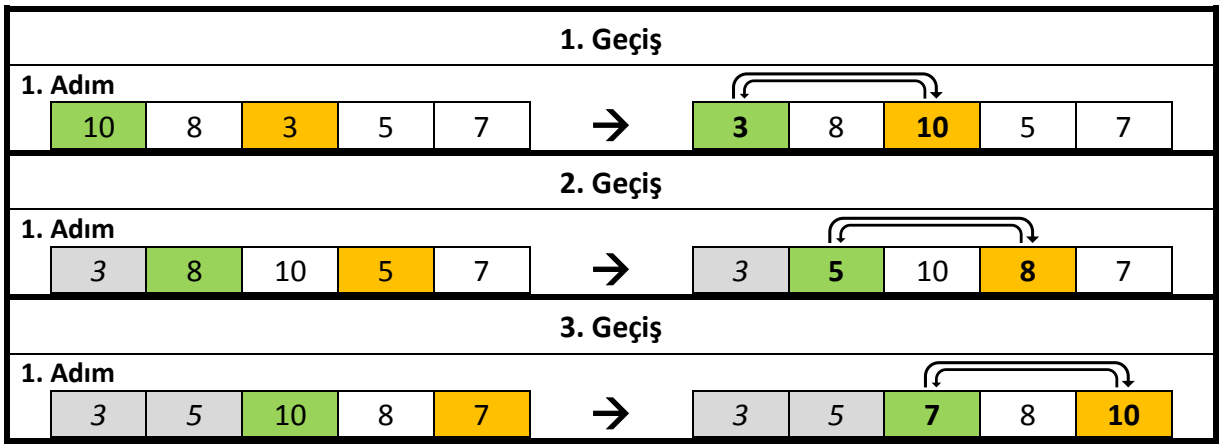
```
            if (dizi[j] > dizi[j + 1])
            {
               sirali = false;
               gecici = dizi[j + 1];
               dizi[j + 1] = dizi[j];
               dizi[j] = gecici;
            }
         }
         if (sirali)
            break;
      }
}

public static void diziYaz(int [] dizi)
{
   for (int i = 0; i <= dizi.Length - 1; i++)
      Console.Write(dizi[i] + " ");
   Console.WriteLine();
}
```

2) Seçmeli Sıralama (Selection Sort) Algoritması

Seçmeli sıralama algoritması listeden sırasıyla en küçük elemanın seçilerek yerine koyulması esasına dayanır. Bu algoritma başlangıçta listenin ilk elemanını en küçük değer olarak kabul eder.

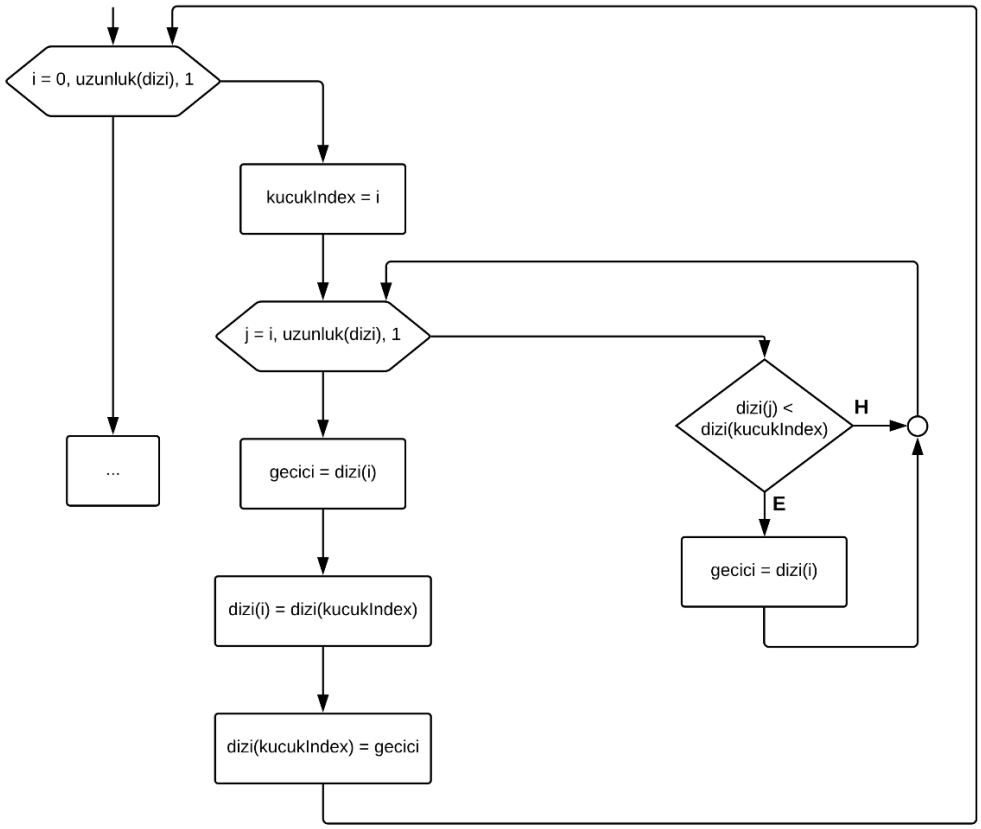
Örnek: Liste = { 10, 8, 3, 5, 7 } listesini seçmeli sıralama algoritmasını kullanarak çözünüz.



4. Geçiş										
1. Adım										
	3	*5*	*7*	8	10	→	*3*	*5*	*7*	8	10

4. geçişte görüldüğü üzere yer değişikliğine gerek duyulmamıştır.

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki yaptığımız akış diyagramını bir de C# dilinde yazalım;

```
public static void secmeliSirala(int [] dizi)
{
    int gecici, kucukIndex, i, j;
    for (i = 0; i < dizi.Length; i++)
    {
       kucukIndex = i;
       for (j = i; j < dizi.Length; j++)
       {
          if (dizi[j] < dizi[kucukIndex])
             kucukIndex = j;
       }
       gecici = dizi[i];
       dizi[i] = dizi[kucukIndex];
       dizi[kucukIndex] = gecici;
    }
}

public static void diziYaz(int [] dizi)
{
   for (int i = 0; i < dizi.Length; i++)
      Console.Write(dizi[i] + " ");
   Console.WriteLine();
}
```

```
static void Main(string[] args)
{
   int [] sayilar = { 10, 8, 3, 5, 7 };
   diziYaz(sayilar);
   secmeliSirala(sayilar);
   diziYaz(sayilar);
   Console.ReadKey();
}
```

3) Eklemeli Sıralama (Insertion Sort) Algoritması

Basit bir algoritma mantığı olan eklemeli sıralama algoritması verimsiz bir sıralama yöntemidir. Çalışma mantığı olarak listenin ikinci elemanı ile ilk elemanı birinci adımda karşılaştırılır. Gerekli ise yer değişikliği yapılır. Bir sonraki adımda ise listenin üçüncü elemanı ile kendinden önceki elemanlar tek tek karşılaştırılır ve gerekli ise yer değişikliği yapılır. Bu işlem son elemana kadar aynen tekrar edilir. Eklemeli sıralama algoritması büyük veritabanlarında tercih edilen bir yaklaşım değildir.

Örnek: Liste = { 2, 5, 5, 8, 10, 7, 4, 12, 9 } listesini eklemeli sıralama algoritmasını kullanarak çözünüz.

1. Adım

2	5	5	8	10	7	4	12	9

2. Adım

2	5	5	8	10	7	4	12	9

3. Adım

2	5	5	8	10	7	4	12	9

4. Adım

2	5	5	8	10	7	4	12	9

5. Adım

2	5	5	8	10	7	4	12	9
2	5	5	7	8	10	4	12	9

6. Adım

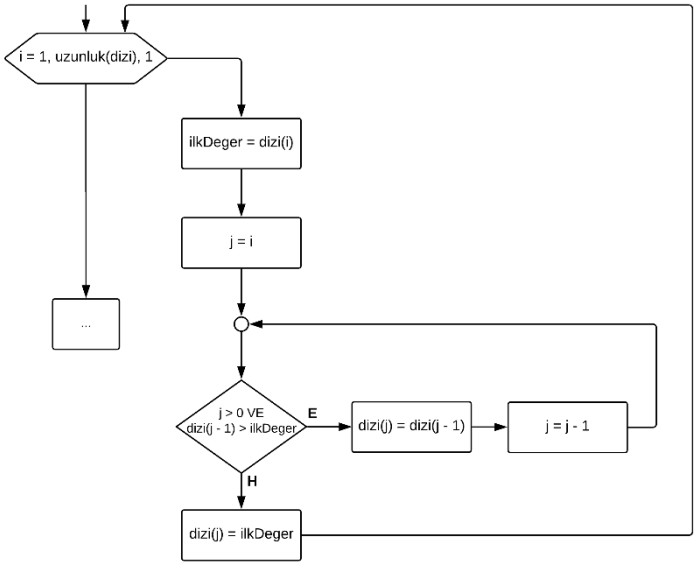
2	5	5	7	8	10	4	12	9
2	4	5	5	7	8	10	12	9

7. Adım

2	4	5	5	7	8	10	12	9

8. Adım

2	4	5	5	7	8	10	12	9
2	4	5	5	7	8	9	10	12



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir;

```
public static void eklemeliSiralama(int[] dizi)
{
   int i, j, ilkDeger;
   for (i = 1; i < dizi.Length; i++)
   {
      ilkDeger = dizi[i];
      j = i;
      while (j > 0 && dizi[j - 1] > ilkDeger)
         {
            dizi[j] = dizi[j - 1];
            j--;
         }
         dizi[j] = ilkDeger;
   }
}

public static void diziYaz(int[] dizi)
{
   for (int i = 0; i < dizi.Length; i++)
      Console.Write(dizi[i] + " ");
   Console.WriteLine();
}
```

```
static void Main(string[] args)
{
   int[] sayilar = {2,5,5,8,10,7,4,12,9};
   diziYaz(sayilar);
   eklemeliSiralama(sayilar);
   diziYaz(sayilar);
   Console.ReadKey();
}
```

4) Kabuk Sıralama (Shell Sort) Algoritması

Bu algoritmada karşılaştırma işlemi belirlenmiş olan atlama sırasına göre daha uzaktaki elemanların karşılaştırılması esasına dayanır. Başlangıçta atlama sayısı eleman sayısının yarısı alınarak hesaplanır. Atlama sayısı her çevrimde *(her geçişte)* bir önceki atlama sayısının yarısı olacak şekilde hesaplanır. Son geçişte atlama sayısı daima 1'dir. Çok sık kullanılan bir algoritmadır.

1. Adım

Atlama sayısı: 9 / 2 = **4**

6	9	5	7	3	4	2	1	8	→	3	4	2	1	6	9	5	7	8

2. Adım

Atlama sayısı: 4 / 2 = **2**

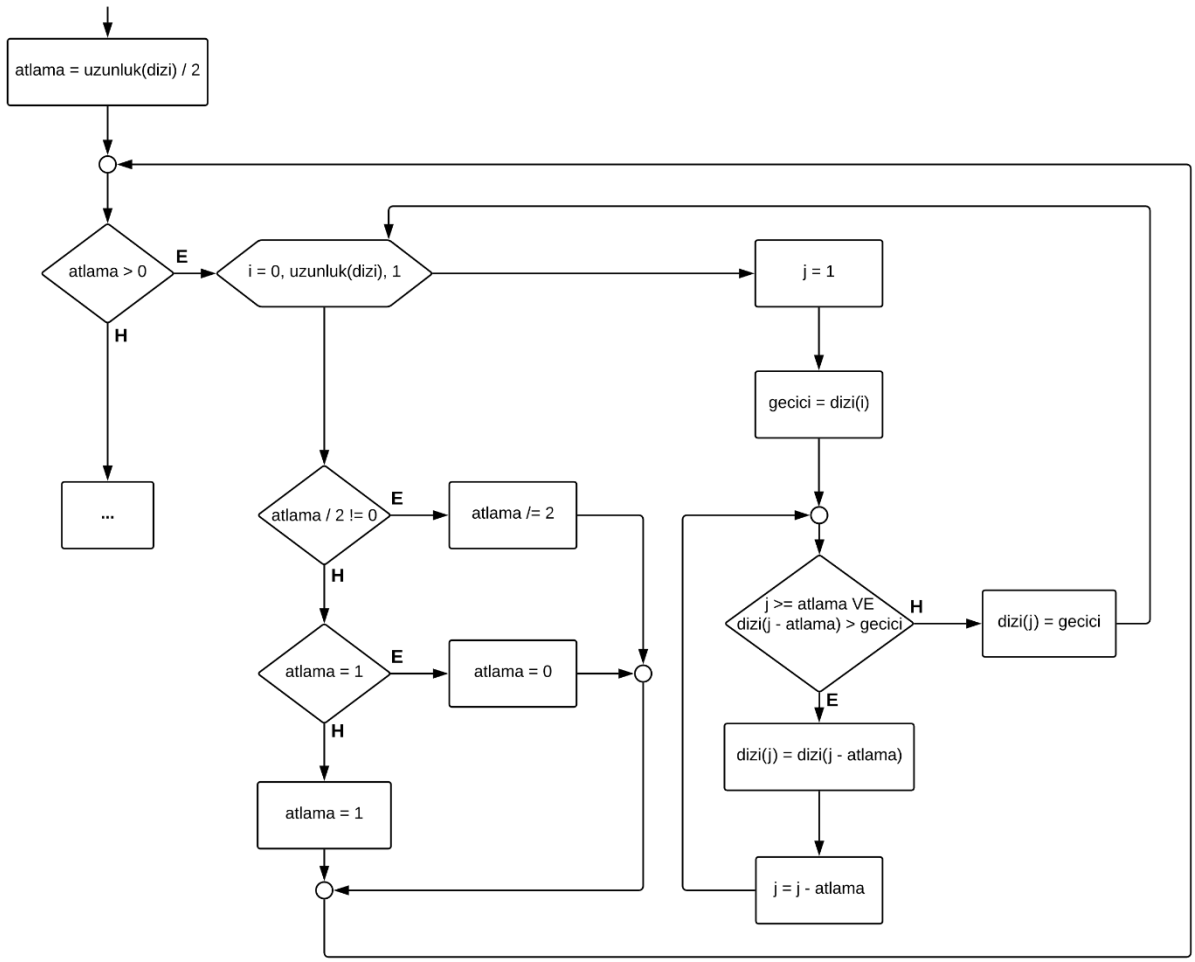
3	4	2	1	6	9	5	7	8	→	2	1	3	4	5	7	6	9	8

3. Adım

Atlama sayısı: 2 / 2 = **1**

2	1	3	4	5	7	6	9	8	→	1	2	3	4	5	6	7	8	9

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir:

```
class Program
{
    static void Main(string[] args)
    {
        int [] sayilar = { 6, 9, 5, 7, 3, 4, 2, 1, 8 };
        diziYaz(sayilar);
        kabukSiralama(sayilar);
        diziYaz(sayilar);
        Console.ReadKey();
    }

    public static void kabukSiralama(int [] dizi)
    {
        int i, j, atlama, gecici;
        atlama = dizi.Length / 2;
        while (atlama > 0)
        {
            for (i = 0; i < dizi.Length; i++)
            {
                j = i;
                gecici = dizi[i];
                while (j >= atlama && dizi[j - atlama] > gecici)
                {
                    dizi[j] = dizi[j - atlama];
                    j = j - atlama;
                }
                dizi[j] = gecici;
            }
            if (atlama / 2 != 0)
                atlama /= 2;
            else if (atlama == 1)
                atlama = 0;
            else
                atlama = 1;
        }
    }

    public static void diziYaz(int [] dizi)
    {
        for (int i = 0; i < dizi.Length; i++)
            Console.Write(dizi[i] + " ");
        Console.WriteLine();
    }
}
```

C# Dilinde Arama Algoritmaları

Listelerden herhangi bir elemanın aranması için kullanılan algoritmalardır. Listeler üzerinde arama yapabilmek için kullanılan üç arama algoritması bulunur. Arama işlemi index numarasına göre yapılır.

1. Doğrusal Arama (Lineer Search) Algoritması

Listedeki tüm elemanlar sırasıyla taranır. Aranan eleman bulunduğunda algoritma sonlanır. Eleman sayısı fazla olan listelerde bu algoritmayı kullanmak performans açısından verimsizdir. Listedeki tüm elemanlara sırasıyla bakıldığından listenin sıralı olması gerekli değildir.

1. Adım

Aranan sayı: **3**

5	1	3	7	6	2

→ 5 ≠ 3 olduğu için sonraki elemana geçilir.

2. Adım

Aranan sayı: **3**

5	**1**	3	7	6	2

→ 1 ≠ 3 olduğu için sonraki elemana geçilir.

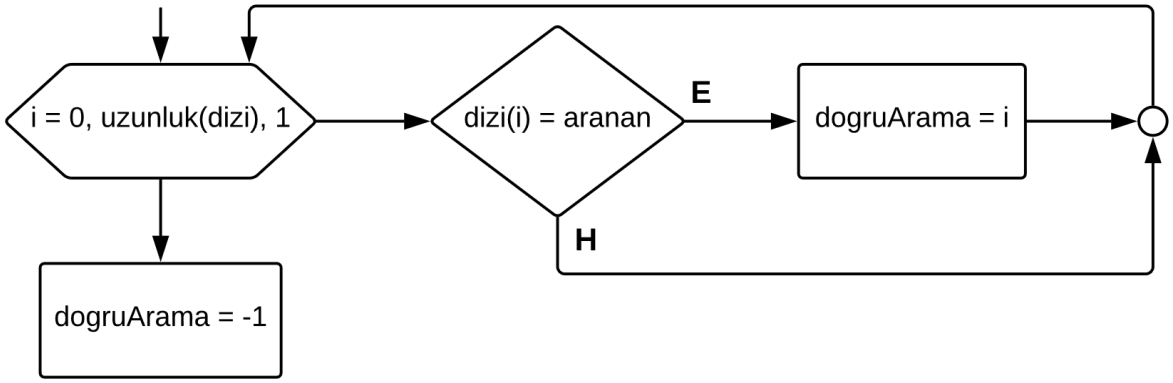
3. Adım

Aranan sayı: **3**

5	*1*	**3**	7	6	2

→ 3 = 3 olduğu için algoritma sonlanır.

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir.

```
class Program
{
    static void Main(string[] args)
    {
        int [] liste = { 5, 1, 3, 7, 6, 2 };
        int aranan = 3;
        int indis = dogrusalArama(liste, aranan);

        if (indis == -1)
            Console.Write("Eleman bulunamadı!");
        else
            Console.Write("Elemanın indisi: " + indis);

        Console.ReadKey();
    }

    public static int dogrusalArama (int [] dizi, int aranan)
    {
        for (int i = 0; i < dizi.Length; i++)
        {
            if (dizi[i] == aranan)
                return i;
        }
        return -1;
    }
}
```

Nesne Tabanlı Programlama I – 15.12.2017

2. İkili Arama (Binary Search) Algoritması

İkili arama algoritması sıralı listeler üzerinde çalışmaktadır. Aranan eleman dizinin ortasındaki eleman ile karşılaştırılır. Ortadaki eleman ile aranan eleman eşitse algoritma sonlandırılır.

a. Aranan eleman, ortadaki elemandan küçükse listenin sol tarafı dikkate alınır.
b. Aranan eleman, ortadaki elemandan büyükse listenin sağ tarafı dikkate alınır.

Bu işlemler eleman bulunana kadar bu işlem devam eder.

Örnek: Liste = { 2, 5, 6, 8, 10, 15, 17, 18, 20, 25, 30 } listesi ve aranan sayımız 25 olsun.[2]

1. Geçiş

Aranan sayı: **25**

Elemanlar	2	5	6	8	10	**15**	17	18	20	**25**	30
Index No.	0	1	2	3	4	5	6	7	8	9	10

$\text{orta} = \frac{0 + 10}{2} = \mathbf{5}$ → 15 ≠ 25 ve 25 > 15 olduğu için listenin sağ tarafı dikkate alınır.

2. Geçiş

Aranan sayı: **25**

Elemanlar	17	18	**20**	**25**	30
Index No.	0	1	2	3	4

$\text{orta} = \frac{0 + 4}{2} = \mathbf{2}$ → 20 ≠ 25 ve 25 > 20 olduğu için listenin sağ tarafı dikkate alınır.

3. Geçiş

Aranan sayı: **25**

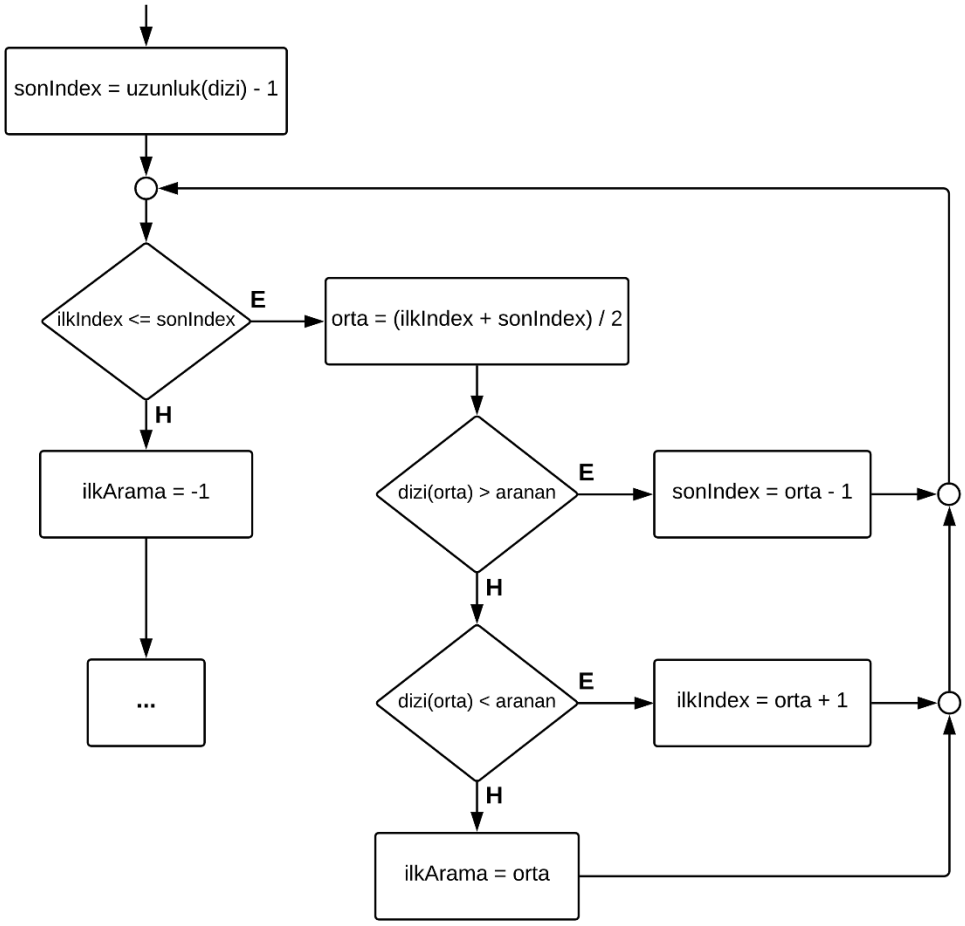
Elemanlar	**25**	30
Index No.	0	1

$\text{orta} = \frac{0 + 1}{2} = 0{,}5 \cong \mathbf{0}$ → 25 = 25 olduğu için algoritma sonlanır.

Not: Üçüncü geçişte görüldüğü üzere değer 0,5 çıkmasına rağmen sonucu sıfır olarak kabul ettik. Bunun nedeni hem tek hem de çift sayıda elemana sahip dizilerde doğru sonuca ulaşabilmek içindir. Yani, eğer ki index numarası bölümündeki orta adı verdiğimiz sonuç virgüllü çıkarsa bir alt değere yuvarlıyoruz.

Yukarıda bahsettiğimiz işi bir de bir sonraki sayfada akış diyagramına dökelim.

[2] İkili sıralamada ortanca değeri bulmak için $\text{orta} = \frac{\text{İlk terim + Son terim}}{2}$ formülü kullanılır.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir.

```
class Program
{
    public static int ikiliArama(int [] dizi, int aranan)
    {
        int ilkIndex = 0;
        int sonIndex = dizi.Length - 1;
        int orta;
        while (ilkIndex <= sonIndex)
        {
            orta = (ilkIndex + sonIndex) / 2;
            if (dizi[orta] > aranan)
                sonIndex = orta - 1;
            else if (dizi[orta] < aranan)
                ilkIndex = orta + 1;
            else
                return orta;
        }
        return -1;
    }
    static void Main(string[] args)
    {
        int [] liste = { 2, 5, 6, 8, 10, 15, 17, 18, 20, 25, 30 };
        int aranan = 25;
        int index = ikiliArama(liste, aranan);
        if (index == -1)
            Console.WriteLine("Aranan değer bulunamadı!");
        else
            Console.WriteLine("Aranan elemanın index değeri: " + index);
        Console.ReadKey();
    }
}
```

3. Ara Değer Arama (Interpolation Search) Algoritması

Sıralı listelerde arama yapan bir algoritmadır. İkili arama algoritması ile aynı mantıkta çalışır. Tek farkı orta eleman değerinin aşağıdaki formüle göre hesaplanmasıdır.

$$\text{orta} = \text{ilkIndex} + \frac{\big((\text{aranan} - \text{dizi}[\text{ilkIndex}]) * (\text{sonIndex} - \text{ilkIndex})\big)}{(\text{dizi}[\text{sonIndex}] - \text{dizi}[\text{ilkIndex}])}$$

Örnek: Liste = {2, 3, 5, 8, 11, 15, 18, 22, 25, 30} listesi ve aranan değerimiz 22 olsun.

1. Geçiş

Aranan sayı: **22**

Elemanlar	2	3	5	8	11	15	**18**	22	25	30
Index No.	0	1	2	3	4	5	6	7	8	9

$\text{orta} = 0 + \frac{((22-2)*(9-0))}{(30-2)} = \frac{180}{28} = 6{,}42 \cong \mathbf{6}$ → 18 ≠ 22 ve 22 > 18 olduğu için listenin sağ tarafı dikkate alınır.

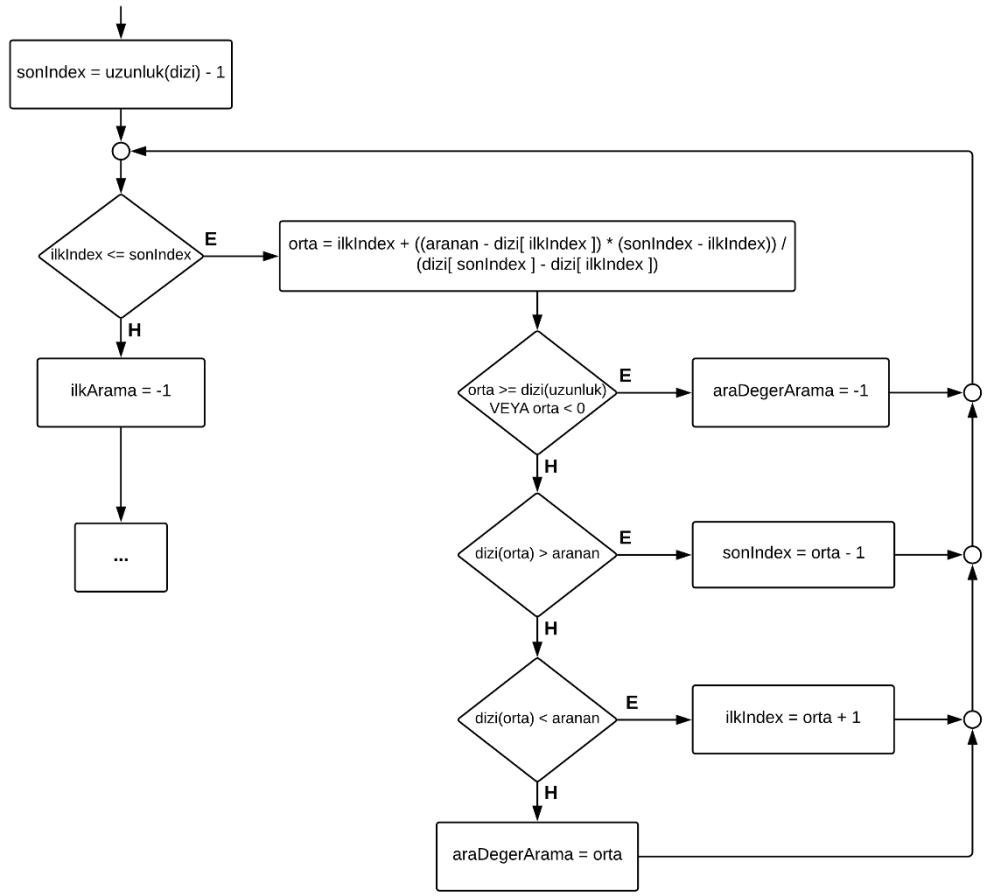
2. Geçiş

Aranan sayı: **22**

Elemanlar	**22**	25	30
Index No.	7	8	9

$\text{orta} = 7 + \frac{((22-22)*(9-7))}{(30-22)} = 7 + \frac{0}{8} = \mathbf{7}$ → 22 = 22 olduğu için algoritma sonlanır.

Yukarıda bahsettiğimiz işi bir de akış diyagramına dökelim.



Yukarıdaki akış diyagramının C# dilindeki yazımı aşağıdaki gibidir.

```
class Program
{
    public static int aradaDegerArama(int [] dizi, int aranan)
    {
        int ilkIndex = 0;
        int sonIndex = dizi.Length - 1;
        int orta;
        while (ilkIndex <= sonIndex)
        {
            orta = ilkIndex + ((aranan - dizi[ilkIndex]) * (sonIndex - ilkIndex)) /
(dizi[sonIndex] - dizi[ilkIndex]);
            if (orta >= dizi.Length || orta < 0)
                return -1;
            else if (dizi[orta] > aranan)
                sonIndex = orta - 1;
            else if (dizi[orta] < aranan)
                ilkIndex = orta + 1;
            else
                return orta;
        }
        return -1;
    }
    static void Main(string[] args)
    {
        int [] liste = { 2, 3, 5, 8, 11, 15, 18, 22, 25, 30 };
        int aranan = 22;
        int index = aradaDegerArama(liste, aranan);
        if (index == -1)
            Console.WriteLine("Aranan değer bulunamadı!");
        else
            Console.WriteLine("Aranan elemanın index değeri: " + index);
        Console.ReadKey();
    }
}
```

C# Dilinde Dosyalama İşlemleri

Dosyalama işlemleri klasör ve dosya işlemleri olmak üzere ikiye ayrılır. Bu işlemler yapılırken programın başına "`using`" anahtar sözcüğünden sonra "`System.IO`" ön işlemcisi eklenmelidir. Klasör işlemlerinde sıklıkla "`Directory`" sınıfı, dosya işlemlerinde ise "`File`" sınıfı tercih edilir. Her iki sınıfta da kullanılan metoda bağlı olarak verilen parametre değeri ve bu parametrenin oluşturacağı yol daima string şeklinde ve (`@"..."`) olarak verilmelidir.

1. Klasör İşlemleri

a. Klasör Oluşturmak

Directory sınıfını kullanarak klasör oluşturmak için "CreateDirectory" komutunu kullanırız.

```
Directory.CreateDirectory(@"C:\Ornek"); // Ornek adında bir klasör oluşturur.
```

b. Klasör Silmek

Directory sınıfını kullanarak klasör silmek için "Delete" komutunu kullanırız.

```
Directory.Delete(@"C:\Ornek"); // Ornek adındaki klasörü siler.
```

Yukarıdaki komut satırında klasörün bulunamaması veya klasörün içi dolu olma durumunda hata ile karşılaşılabilir. Bu durumda "false" komutu eklenerek silme işlemi iptal edilir.

```
Directory.Delete(@"C:\Ornek", false);
// Ornek klasörünün içerisi boş olmadığı için işlem iptal edilir.
```