

# ALGORİTMA ANALİZİ

## 1-INSERT SORT

```
7 static int[] InsertSort(int[] dizi)
8 {
9     Boolean eklendi;
10    int deger, uzunluk = dizi.Length; ;
11    for (int i = 0; i < uzunluk; i++)
12    {
13        deger = dizi[i];
14        eklendi = false;
15        for (int j = i-1; j >= 0 && !eklendi;)
16        {
17            if (deger < dizi[j])
18            {
19                dizi[j + 1] = dizi[j];
20                j--;
21                dizi[j + 1] = deger;
22            }
23            else
24                eklendi = true;
25        }
26    }
27    return dizi;
28 }
```

### Algoritma Tasarım Stratejisi : Azalt ve Yönet

**En Kötü Durum( O ) :  $n * n$**

Bu algoritmada en kötü durum dizinin elemanın her iterasyonda yer değiştirme sayısının en yüksek olduğu durumdur. O halde;

$$T(n) = T(n) + T(n * n)$$

**En İyi Durum (  $\Omega$  ) :  $n$**

Bu algoritmada en iyi durum dizi elemanının zaten sıralı olmasıdır. Bu durumda algoritma her durum için n kadar işlenir. Yani ;

$$T(n) = T(n) + T(n)$$

Ortalama Durum (  $\Theta$  ):  $d + n$

$d$  yer değiştirme sayısıdır ve  $n^2$  cinsindendir.

Analiz :

```
261 static void Main(string[] args) {
262
263     long onceki_zaman;
264     int[] dizi = DiziUret(5000, false);
265
266     //En iyi durum için sıralı bir dizi oluşturulup algoritma test edildi
267     onceki_zaman = GetMiliseconds();
268     InsertSort(dizi);
269     Console.WriteLine("En iyi durum için süre harcanan süre: "
270         + (GetMiliseconds() - onceki_zaman) + " ms");
271
272     //Ortalama durum için dizi tersten sıralanmış şekilde algoritmada test edildi
273     dizi = DiziTersCevir(dizi);
274     onceki_zaman = GetMiliseconds();
275     InsertSort(dizi);
276     Console.WriteLine("Ortalama bir durum için harcanan süre: "
277         + (GetMiliseconds() - onceki_zaman) + " ms");
278
279     //En kötü durum için rastgele dizi oluşturulup algoritma test edildi
280     dizi = DiziUret(50000, true);
281     onceki_zaman = GetMiliseconds();
282     InsertSort(dizi);
283     Console.WriteLine("En kötü durum için süre harcanan süre: "
284         + (GetMiliseconds() - onceki_zaman) + " ms");
285
286     Console.ReadKey();
287 }
288 }
```

C:\Program Files\dotnet\dotnet.exe

En iyi durum için süre harcanan süre: 335 ms  
Ortalama bir durum için harcanan süre: 107942 ms  
En kötü durum için süre harcanan süre: 4443587 ms

## 2-BOUBLE SORT

```
30 static int[] BoubleSort(int[] dizi)
31 {
32     int deger, uzunluk = dizi.Length;
33     for (int i = 0; i < uzunluk; i++)
34     {
35         for (int j = uzunluk - 1; j >= 1; j--)
36         {
37             if (dizi[j - 1] > dizi[j])
38             {
39                 deger = dizi[j];
40                 dizi[j] = dizi[j - 1];
41                 dizi[j - 1] = deger;
42             }
43         }
44     }
45     return dizi;
46 }
47
```

### Algoritma Tasarım Stratejisi : Kaba kuvvet

**En Kötü Durum( O ) :  $n * n$**

Bu algoritmada en kötü durum dizinin tam anlamıyla sırasız olması durumudur. O halde;

$$T(n) = T(n) + T(n * n)$$

**En İyi Durum (  $\Omega$  ) :  $n$**

Bu algoritmada en iyi durumun dizinin sıralı olması durumudur. Bu durumda algoritma her durum için  $n$  kadar işlenir. Yani ;

$$T(n) = T(n) + T(n)$$

**Ortalama Durum (  $\Theta$  ):  $n * n$**

## Analiz :

```
261 static void Main(string[] args) {
262
263     long onceki_zaman;
264     int[] dizi = DiziUret(5000, false);
265
266     //En iyi durum için sıralı bir dizi oluşturulup algoritma test edildi
267     onceki_zaman = GetMiliseconds();
268     BoubleSort(dizi);
269     Console.WriteLine("En iyi durum için süre harcanan süre: "
270         +(GetMiliseconds() - onceki_zaman) + " ms");
271
272     //Ortalama durum için dizi tersten sıralanmış şekilde algoritmada test edildi
273     dizi = DiziTersCevir(dizi);
274     onceki_zaman = GetMiliseconds();
275     BoubleSort(dizi);
276     Console.WriteLine("Ortalama bir durum için harcanan süre: "
277         + (GetMiliseconds() - onceki_zaman) + " ms");
278
279     //En kötü durum için rastgele dizi oluşturulup algoritma test edildi
280     dizi = DiziUret(50000, true);
281     onceki_zaman = GetMiliseconds();
282     BoubleSort(dizi);
283     Console.WriteLine("En kötü durum için süre harcanan süre: "
284         + (GetMiliseconds() - onceki_zaman) + " ms");
285
286     Console.ReadKey();
287 }
288 }
```

C:\Program Files\dotnet\dotnet.exe

En iyi durum için süre harcanan süre: 116272 ms  
Ortalama bir durum için harcanan süre: 153495 ms  
En kötü durum için süre harcanan süre: 15067142 ms

## 3-FAKTORYEL

```
48 static int Faktoryel(int N)
49 {
50     if (N <= 1)
51     {
52         return 1;
53     }
54     else
55         return N * Faktoryel(N - 1);
56 }
```

// 0 |  $\Omega$   
// n | 1  
// n-1 | 0

### Algoritma Tasarım Stratejisi : Rekürsif

En Kötü Durum( O ) : **n**

En İyi Durum (  $\Omega$  ) : **1**

Ortalama Durum (  $\Theta$  ) : **n**

$$T(n) = T(n) + T(n-1)$$

### Analiz :

```
269 //En iyi durum için faktoryel hesabı 1
270 önceki_zaman = GetMilliseconds();
271 Faktoryel(1);
272 Console.WriteLine("En iyi durum için süre harcanan süre: "
273     + (GetMilliseconds() - önceki_zaman) + " ms");
274
275 //Ortalama durum için faktoryel hesabı 250
276 önceki_zaman = GetMilliseconds();
277 Faktoryel(250);
278 Console.WriteLine("Ortalama bir durum için harcanan süre: "
279     + (GetMilliseconds() - önceki_zaman) + " ms");
280
281 //En kötü durum için faktöryel hesabı 500
282 önceki_zaman = GetMilliseconds();
283 Faktoryel(500);
284 Console.WriteLine("En kötü durum için süre harcanan süre: "
285     + (GetMilliseconds() - önceki_zaman) + " ms");
286
287 Console.ReadKey();
288 //Ortalama ve Kötü durumda Notasyon N'e bağlı olduğu için değer N'e bağlı değişir
289 }
290 }
291
```

C:\Program Files\dotnet\dotnet.exe

En iyi durum için süre harcanan süre: 900 ms  
Ortalama bir durum için harcanan süre: 10500 ms  
En kötü durum için süre harcanan süre: 17300 ms

## 4-BINARY SEARCH

```
57 static int BinarySearch(int[] sayilar, int arananSayi)
58 {
59     // 0 | Ω
60     int baslangic = 0, bitis = sayilar.Length - 1, orta;
61     while (baslangic <= bitis) // n logn | 1
62     {
63         orta = (baslangic + bitis) / 2;
64         if (sayilar[orta] > arananSayi)
65         {
66             bitis = orta - 1;
67         }
68         else if (sayilar[orta] < arananSayi)
69         {
70             baslangic = orta + 1;
71         }
72         else
73         {
74             return orta;
75         }
76     }
77     return -1;
}
```

### Algoritma Tasarım Stratejisi : Böl - Yönet

**En Kötü Durum( O ) :  $n \log n$**

Bu algorithma en kötü durum aranan değerinin dizinin son iterasyonda bulunması durumudur. Algoritma diziyi sürekli bölerek arama işlemi gerçekleştirilir. O halde;

$$T(n) = T(n) + T(n \log n)$$

**En İyi Durum ( Ω ) : 1**

En iyi durum aranan değerin direk bulunmasıdır. İşlem tek iterasyonda gerçekleşir.

**Ortalama Durum ( Θ ) :  $n \log n$**

## Analiz :

```
262 long onceki_zaman;
263 // 500.000'den geriye doğru sıralı bir dizi
264 int[] dizi = DiziUret(500000, false);
265
266 //En iyi durum ilk iterasyonda ulaşılacak elemanı aramak
267 onceki_zaman = GetMilliseconds();
268 BinarySearch(dizi, 500000);
269 Console.WriteLine("En iyi durum için süre harcanan süre: "
270 + (GetMilliseconds() - onceki_zaman) + " ms");
271
272 //Ortalama rastgele bir elemanı aramak
273 onceki_zaman = GetMilliseconds();
274 BinarySearch(dizi, 7);
275 Console.WriteLine("Ortalama bir durum için harcanan süre: "
276 + (GetMilliseconds() - onceki_zaman) + " ms");
277
278 //En kötü durum için aranan eleman ya en son iterasyonda bulunacak yada hiç bulunmayacak
279 onceki_zaman = GetMilliseconds();
280 BinarySearch(dizi, 500001);
281 Console.WriteLine("En kötü durum için süre harcanan süre: "
282 + (GetMilliseconds() - onceki_zaman) + " ms");
283
284 Console.ReadKey();
285
286 }
287
288 }
289
```

C:\Program Files\dotnet\dotnet.exe

En iyi durum için süre harcanan süre: 900 ms  
Ortalama bir durum için harcanan süre: 1200 ms  
En kötü durum için süre harcanan süre: 2100 ms

## 5-FİBONACCI

```
78 public int fibonacci_dinamik_programlama(int kacinci)
79 {
80     int[] fibonacci_sayilari = new int[kacinci]; // 0 | Ω
81     for (int i = 0; i <= kacinci; i++) // n | n
82     {
83         if (i == 0 || i == 1)
84         {
85             fibonacci_sayilari[0] = 1; fibonacci_sayilari[1] = 1;
86         }
87         else
88         {
89             fibonacci_sayilari[i] = fibonacci_sayilari[i - 1] + fibonacci_sayilari[i - 2];
90         }
91     }
92     return fibonacci_sayilari[kacinci];
93 }
94
```

## Algoritma Tasarım Stratejisi : Dinamik Programlama

En Kötü Durum( O ) : **n**

En İyi Durum ( Ω ) : **n**


Ortalama Durum ( Θ ) : **n**

Bu algorithmada normalde recursif ile  $2^n$  karmaşıklığına sahip programı dinamik stratejisi ile  $n$  karmaşıklığına indirgenir

$$T(n) = T(n)$$

## Analiz :

```
270  /* Fibonacci için algoritma karmaşıklığı yalnızca N' e bağlı olduğu için ;  
271     değişimi gözlemlemek adına farklı n değerleri test edildi */  
272  önceki_zaman = GetMilliseconds();  
273  fibonacci_dinamik_programlama(50);  
274  Console.WriteLine("50 için süre harcanan süre: "  
275      + (GetMilliseconds() - önceki_zaman) + " ms");  
276  
277  //Ortalama durum için rastgele oluşturulmuş dizi test edildi  
278  önceki_zaman = GetMilliseconds();  
279  fibonacci_dinamik_programlama(500);  
280  Console.WriteLine("500 için harcanan süre: "  
281      + (GetMilliseconds() - önceki_zaman) + " ms");  
282  
283  //En kötü durum için tersten sıralanmış dizi test edildi  
284  önceki_zaman = GetMilliseconds();  
285  fibonacci_dinamik_programlama(5000);  
286  Console.WriteLine("5000 için süre harcanan süre: "  
287      + (GetMilliseconds() - önceki_zaman) + " ms");  
288  
289  Console.ReadKey();  
290  } // Görüldüğü gibi harcanan süre N' e bağlı olarak arttı  
291  }  
292  }  
293
```



```
C:\Program Files\dotnet\dotnet.exe  
50 için süre harcanan süre: 2200 ms  
500 için harcanan süre: 10400 ms  
5000 için süre harcanan süre: 61600 ms
```



## 6-COUNTING SORT

```
96 static int[] CountingSort(int[] dizi)
97 {
98     int uzunluk = dizi.Length, deger;
99     int[] say = new int[uzunluk];
100     for (int i = 0; i < uzunluk; i++)
101     {
102         deger = dizi[i];
103         say[deger]++;
104     }
105
106     for (int i = 1; i < uzunluk; i++)
107     {
108         say[i] = say[i] + say[i - 1];
109     }
110
111     int[] sirali = new int[uzunluk];
112
113     for (int i = uzunluk - 1; i >= 0; i--)
114     {
115         deger = dizi[i];
116         int position = say[deger] - 1;
117         sirali[position] = deger;
118         say[deger]--;
119     }
120     return sirali;
121 }
```

### Algoritma Tasarım Stratejisi : Sayarak

En Kötü Durum(  $O$  ) :  $n$

En İyi Durum (  $\Omega$  ) :  $n$

Ortalama Durum (  $\Theta$  ) :  $n$

Bu algoritma tüm durumlar için dizi boyutu(k) oranında işlem gerçekleşir

$$T(n) = T(n) + T(n) + T(n)$$

## Analiz :

```
263     long onceki_zaman;
264     // 500.000'den geriye doğru sıralı ve rastgele sıralı diziler
265     int[] sirali_dizi = DiziUret(50000, false);
266     int[] rastgele_dizi = DiziUret(50000, true);
267     int[] ters_sirali = DiziTersCevir(sirali_dizi);
268
269
270
271     //En iyi durum için sıralı dizi test edildi
272     onceki_zaman = GetMiliseconds();
273     CountingSort(sirali_dizi);
274     Console.WriteLine("En iyi durum için süre harcanan süre: "
275         + (GetMiliseconds() - onceki_zaman) + " ms");
276
277     //Ortalama durum için rastgele oluşturulmuş dizi test edildi
278     onceki_zaman = GetMiliseconds();
279     CountingSort(rastgele_dizi);
280     Console.WriteLine("Ortalama bir durum için harcanan süre: "
281         + (GetMiliseconds() - onceki_zaman) + " ms");
282
283     //En kötü durum için tersten sıralanmış dizi test edildi
284     onceki_zaman = GetMiliseconds();
285     CountingSort(ters_sirali);
286     Console.WriteLine("En kötü durum için süre harcanan süre: "
287         + (GetMiliseconds() - onceki_zaman) + " ms");
288
289     Console.ReadKey();
290 } // Görüldüğü gibi harcanan süre dizinin sıralı, sırasız olması durumunda fark etmedi
291
```

C:\Program Files\dotnet\dotnet.exe

En iyi durum için süre harcanan süre: 1399700 ms  
Ortalama bir durum için harcanan süre: 1458900 ms  
En kötü durum için süre harcanan süre: 1389201 ms

## 7-BOZO SORT

```
121 static int[] BozoSort(int[] dizi)
122 {
123     int slot1 = 0;
124     int slot2 = 0;
125     int ara;
126     Random rand = new Random();
127     while (!Siralimi(dizi))
128     {
129         slot1 = rand.Next(0, dizi.Length);
130         slot2 = rand.Next(0, dizi.Length);
131
132         ara = dizi[slot1];
133         dizi[slot1] = dizi[slot2];
134         dizi[slot2] = ara;
135     }
136     return dizi;
137 }
138 1 başvuru
139 static bool Siralimi(int[] dizi)
140 {
141     for (int i = 0; i < dizi.Length - 1; i++)
142     {
143         if (dizi[i] > dizi[i + 1])
144         {
145             return false;
146         }
147     }
148     return true;
149 }
```

Algoritma Tasarım Stratejisi : Rastgele

En Kötü Durum( O ) :  $\infty$

En İyi Durum (  $\Omega$  ) : 1

Ortalama Durum (  $\Theta$  ):  $n!$

## Analiz :

```
270 // 10 elamanlı sıralanmış bir dizi için yapılan test
271 önceki_zaman = GetMilliseconds();
272 BozoSort(sirali_dizi);
273 Console.WriteLine("En iyi durum için harcanan süre: "
274     + (GetMilliseconds() - önceki_zaman) + " ms");
275
276 // 10 elamanlı sırasız bir dizi için yapılan test
277 önceki_zaman = GetMilliseconds();
278 BozoSort(ters_sirali);
279 Console.WriteLine("Ortalama durum için harcanan süre: "
280     + (GetMilliseconds() - önceki_zaman) + " ms");
281
282 Console.ReadKey();
283 }
284 }
285 }
286
```

C:\Program Files\dotnet\dotnet.exe

En iyi durum için harcanan süre: 3800 ms  
Ortalama durum için harcanan süre: 43622800 ms

Bu algoritma sıralama işlemini tamamıyla rastgele gerçekleştirdiği için en kötü durum dizin şanssız bir şekilde **hiçbir şekilde sıralanmaması** durumudur. Ortalama durumda bile 10 elamanlı dizi için yaklaşık 44 snyde işlemi gerçekleştirirken, **dizi eleman sayısı arttıkça** sıralanması bir okadar zorlaşacaktır.

## 8-HEAP SORT

```
150 static void HeapSort(int[] dizi)
151 {
152     int uzunluk = dizi.Length;
153     for (int i = uzunluk / 2 - 1; i >= 0; i--)
154         heapify(dizi, uzunluk, i);
155     for (int i = uzunluk - 1; i >= 0; i--)
156     {
157         int temp = dizi[0];
158         dizi[0] = dizi[i];
159         dizi[i] = temp;
160         heapify(dizi, i, 0);
161     }
162 }
163 // 3 başvuru
164 static void heapify(int[] dizi, int n, int i)
165 {
166     int enbuyuk = i;
167     int sol = 2 * i + 1;
168     int sag = 2 * i + 2;
169     if (sol < n && dizi[sol] > dizi[enbuyuk])
170         enbuyuk = sol;
171     if (sag < n && dizi[sag] > dizi[enbuyuk])
172         enbuyuk = sag;
173     if (enbuyuk != i)
174     {
175         int gecici = dizi[i];
176         dizi[i] = dizi[enbuyuk];
177         dizi[enbuyuk] = gecici;
178         heapify(dizi, n, enbuyuk);
179     }
180 }
```

**Algoritma Tasarım Stratejisi : Dönüştür ve Fethet**

**En Kötü Durum( O ) :  $n \log n$**

**En İyi Durum (  $\Omega$  ) :  $n \log n$**

**Ortalama Durum (  $\Theta$  ) :  $n \log n$**

$$T(n) = T(n) + T(n \log n) + T(n) + T(n \log n)$$

## Analiz :

```
261 static void Main(string[] args) {
262
263     long onceki_zaman;
264     // 100 sıralı,sırasız, ve ters sıralı diziler oluşturuldu
265     int[] sirali_dizi = DiziUret(1000, false);
266     int[] ters_sirali = DiziTersCevir(sirali_dizi);
267     int[] rastgele_sirali = DiziUret(1000, false);
268
269     onceki_zaman = GetMilliseconds();
270     HeapSort(sirali_dizi);
271     Console.WriteLine("En iyi durum için harcanan süre: "
272         + (GetMilliseconds() - onceki_zaman) + " ms");
273
274     onceki_zaman = GetMilliseconds();
275     HeapSort(ters_sirali);
276     Console.WriteLine("Ortalama durum için harcanan süre: "
277         + (GetMilliseconds() - onceki_zaman) + " ms");
278
279     onceki_zaman = GetMilliseconds();
280     HeapSort(rastgele_sirali);
281     Console.WriteLine("En kötü durum için harcanan süre: "
282         + (GetMilliseconds() - onceki_zaman) + " ms");
283
284     Console.ReadKey();
285 }
286
287 }
288
```

Seç C:\Program Files\dotnet\dotnet.exe

En iyi durum için harcanan süre: 380100 ms  
Ortalama durum için harcanan süre: 364000 ms  
En kötü durum için harcanan süre: 357001 ms

Görüldüğü gibi sıralama karmaşıklığı dizinin, sıralı veya sırasız olma durumuna bakmaksızın harcanan süre yalnızca dizi boyutuna bağlı kaldı. Süredeki farklılık bilgisayarın o anki duruma göre değişiklik gösterdi. Ortalama tüketilen süre yaklaşık olarak aynı.

**Fırat AKTAY**