BÖLÜM 4

ALGORİTMALAR

- Bir **algoritma** aşağıdaki özelliklere sahip komutların sonlu bir kümesidir.
  - **Kesinlik :** Algoritma adımları kesin olarak tespit edilmelidir.
  - **Bir teklik:** Her bir adımın yürütülmesinde sonuçlar bir tek olarak tanımlı olmalı ve sadece girdilere bağlı olmalıdır.
  - **Sonluluk**: Algoritma sonlu sayıda komutların yürütülmesinden sonra durmalıdır.
  - **Girdi:** Algoritma girdi almalıdır
  - **Çıktı**: Algoritma çıktı üretmelidir.
  - **Genelilik:** Algoritma, girdilerin bir kümesine uygulanabilir olmalıdır.

Örnek

Girdiler: a, b ve c

Herhangi üç a, b ve c değeri. Bu haliyle algoritma geneldir.

Bu algoritma verilen üç sayıdan en büyüğünü bulur

Girdi: *a, b* ve *c* tam sayıları

Çıktı: a, *b* ve *c* 'nin en büyüğü olarak *large* sayısı

```
1.    max3(a,b,c) {
2.      large=a
3.        if b>large then        // if b is largen than large, update large
4.            large=b
5.        if c>large then        // if c is largen than large, update large
6.            large=c
7.    return large
8.    }
```

Sonlu sayıda adım

kesinlik

çıktı

Bu algoritma $s_1, s_2, \ldots, s_n$ dizisi içinden en büyük sayıyı bulur.

Girdi: s,n

Çıktı: *large*

```
find_large(s,n)  {
    large =s1
    for i=2 to n
        if (si >large)      //a larger value was found
            large =si
    return (large )
}
```

Bu algoritma bir t metni içinde p desenini arar. Eğer p deseni t metni içinde varsa bulunduğu ilk yerin damgasını döndürür. Eğer yoksa geriye 0 döndürür.

Girdi: p (1 'den m 'e kadar), m, t (1 'den n 'e kadar), n

Çıktı: i

```
text_search(p,m,t,n) {
        for i=1 to n-m+1 {
            j=1
            // burada i değeri p ile karşılaştırılacak olan t metni
            //içindeki alt karakter dizgesinin ilk karakterinin damgasıdır.
            // j değeri p 'deki damgadır.

            // burada while döngüsü t_i...t_{i+m-1} ile p_1...p_m 'leri karşılaştırır.
            while (t_{i+j-1} ==p_j) {
                    j=j+1
                    if(j>m)
                            return i
            }       // while
        }           // for
        return 0
}
```

"010001" metni içinde "001" metnini arayalım

j=1 ↓ 001 010001 ↑ i=1 (1)	j=2 ↓ (x) 001 010001 ↑ i=1 (2)	j=1 ↓ (x) 001 010001 ↑ i=2 (3)
j=1 ↓ 001 010001 ↑ i=3 (4)	j=2 ↓ 001 010001 ↑ i=3 (5)	j=3 ↓ (X) 001 010001 ↑ i=3 (6)
j=1 ↓ 001 010001 ↑ i=4 (7)	j=2 ↓ 001 010001 ↑ i=4 (8)	j=3 ↓ 001 010001 ↑ i=4 (9)

Ekleme Sıralaması

Bu algoritma $s_1, s_2, \ldots, s_n$ dizisini artan sırada sıralar

Girdi: s, n

Çıktı: s (sıralı)

```
insertion_sort(s,n) {
    for i=2 to n {
        val = s_i
        j=i-1
        // if val<s_j , move s_j right to make room for s_i
        while (j≥1 ∧ val<s_j ) {
            s_{j+1}=s_j
            j=j-1
        }          // while
        s_{j+1} =val  // insert val
    }              // for
}
```

8	13	20		27

16 değeri için

8	13	20		27

8	13		20	27

8	13	16	20	27

Karma

Bu algoritma

$a_1,\ldots,a_n$

dizisinin değerlerini aralarında karar.

Girdi: a,n

Çıktı: a (karılmış)

```
shuffle (a,n) {
        for i=1 to n-1
                swap( a_i,a_rand(i,n))
}
```

17	9	5	23	21

i=1 ve j=rand(1,5)=3 olmak üzere a_i ile a_j yer değişsin

5	9	17	23	21

i=2 ve j=rand(2,5)=5 olmak üzere a_i ile a_j yer değişsin

5	21	17	23	9

i=3 ve j=rand(3,5)=3 olduğunda yer değiştirme yok

i=4 ve j=rand(4,5)=5 olmak üzere a_i ile a_j yer değişsin

5	21	17	9	23

Time to execute an algorithm if one step takes 1 microsecond to execute[†]

Number of Steps to Termination for Input of Size n	Time to Execute if $n =$				
	3	***6***	***9***	***12***	
1	10^{-6} sec	10^{-6} sec	10^{-6} sec	10^{-6} sec	
$\lg \lg n$	10^{-6} sec	10^{-6} sec	2×10^{-6} sec	2×10^{-6} sec	
$\lg n$	2×10^{-6} sec	3×10^{-6} sec	3×10^{-6} sec	4×10^{-6} sec	
n	3×10^{-6} sec	6×10^{-6} sec	9×10^{-6} sec	10^{-5} sec	
$n \lg n$	5×10^{-6} sec	2×10^{-5} sec	3×10^{-5} sec	4×10^{-5} sec	
n^2	9×10^{-6} sec	4×10^{-5} sec	8×10^{-5} sec	10^{-4} sec	
n^3	3×10^{-5} sec	2×10^{-4} sec	7×10^{-4} sec	2×10^{-3} sec	
2^n	8×10^{-6} sec	6×10^{-5} sec	5×10^{-4} sec	4×10^{-3} sec	
	50	***100***	***1000***	***10^5***	***10^6***
1	10^{-6} sec	10^{-6} sec	10^{-6} sec	10^{-6} sec	10^{-6} sec
$\lg \lg n$	2×10^{-6} sec	3×10^{-6} sec	3×10^{-6} sec	4×10^{-6} sec	4×10^{-6} sec
$\lg n$	6×10^{-6} sec	7×10^{-6} sec	10^{-5} sec	2×10^{-5} sec	2×10^{-5} sec
n	5×10^{-5} sec	10^{-4} sec	10^{-3} sec	0.1 sec	1 sec
$n \lg n$	3×10^{-4} sec	7×10^{-4} sec	10^{-2} sec	2 sec	20 sec
n^2	3×10^{-3} sec	0.01 sec	1 sec	3 hr	12 days
n^3	0.13 sec	1 sec	16.7 min	32 yr	31,710 yr
2^n	36 yr	4×10^{16} yr	3×10^{287} yr	3×10^{30089} yr	3×10^{301016} yr

[†] $\lg n$ denotes $\log_2 n$ (the logarithm of n to base 2).

- Boyu *n* olan tüm girdilerle programı yürütmek için gerekli olan minimum zamanın **en iyi çalışma zamanı** (best-case time) denir.
- Boyu *n* olan tüm girdilerle yürütme için gerekli olan maksimum zamana **en kötü çalışma zamanı** (worst-case time) denir.
- Boyu *n* olan girdilerin sonlu kümesi üzerinden olan **ortalama çalışma zamanı** (average-case time).

n	*t(n)*	$60n^2$
10	6051	6000
100	600,51	600.000
1000	60,005,001	60,000,000
10000	6,000,050,001	6,000,000,000

- **Tanım 4.3.2:** {1,2,3,...} değer bölgesine sahip iki fonksiyon *f* ve *g* olsun. Eğer *n* pozitif bir tamsayı olmak üzere

  $$|f(n)| \leq C_1|g(n)|$$

  olacak şekilde bir pozitif C_1 sayısı varsa *f(n) en fazla g(n) mertebedendir* denir ve

  $$f(n)=O(g(n))$$

  şeklinde yazılır. Buna *f* için **büyük oh gösterimi** denir.

- Eğer
  $|f(n)| \geq C_2|g(n)|$
  olacak şekilde bir pozitif C_2 sayısı varsa *f(n) en az g(n) mertebedendir* denir ve
  $f(n)=\Omega(g(n))$
  şeklinde yazılır. Buna *f* için **omega gösterimi** denir.
- Eğer $f(n)=O(g(n))$ ve $f(n)=\Omega(g(n))$ ise, bu durumda *f(n) fonksiyonu g(n) mertebedendir* denir ve bu
  $f(n)=\Theta(g(n))$
  şeklinde yazılır. Buna *f* için **theta gösterimi** denir.

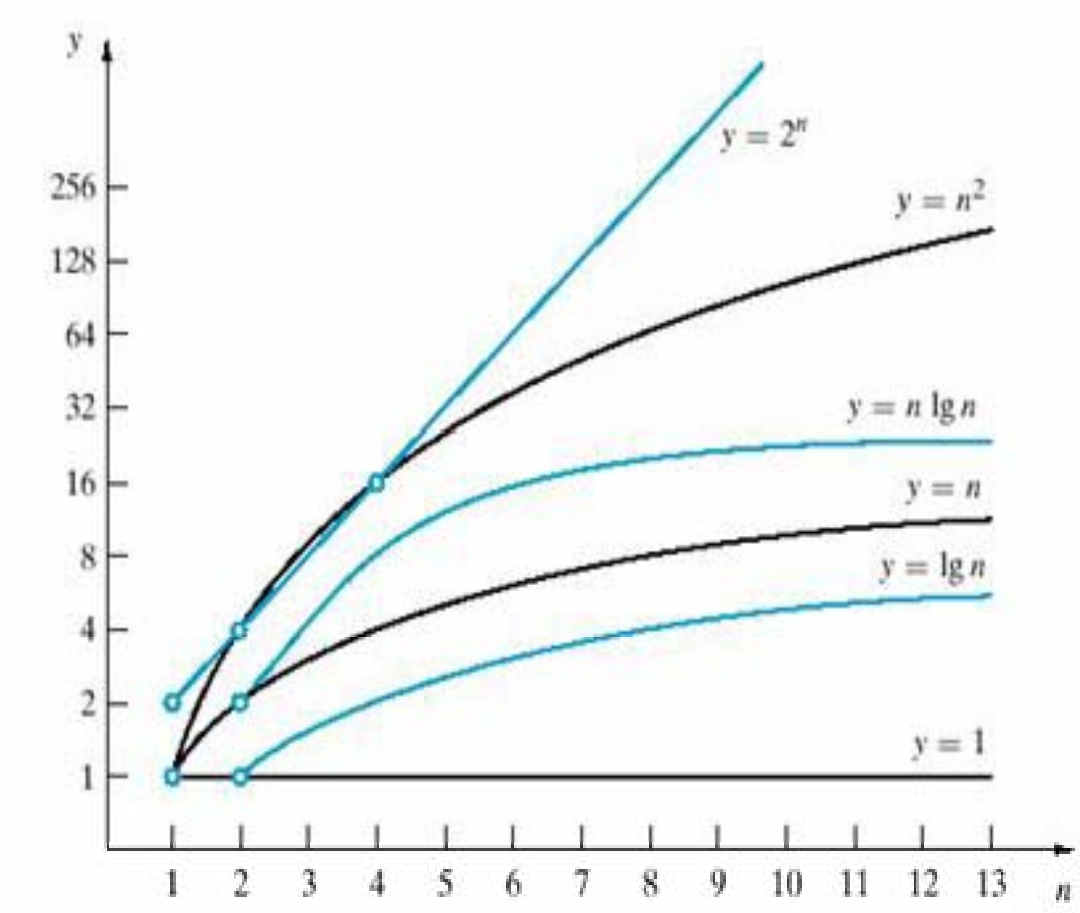
- **Toerem 4.3.4:** *k*. dereceden *n* değişkeninin bir polinomu, her bir a_i negatif olmamak üzere

  $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0$

  ile verilsin. Bu durumda

  $a_k n^k + a_{k-1} n^{k-1} + \ldots + a_1 n + a_0 = \theta(n^k)$

  olur.


y
256
128
64
32
16
8
4
2
1
y = 2^n
y = n^2
y = n lg n
y = n
y = lg n
y = 1
1 2 3 4 5 6 7 8 9 10 11 12 13
n

- **Örnek 4.3.6:** $n \geq 1$ için
  $1+2+\ldots+n$
  toplamı.
- **Örnek 4.3.7:** k pozitif bir tamsayı olmak üzere $n \geq 1$ için
  $1^k+2^k+\ldots+n^k$
  toplamı
- **Örnek 4.3.8:** Benzer argümanlarla
  $\lg n! = \Theta(n \lg n)$
  olduğunu gösterelim.

- **Örnek 4.3.9:** Eğer $f(n)=\Theta(g(n))$ ve $g(n)=\Theta(h(n)$ ise, bu durumda $f(n)=\Theta(h(n))$ olur.
- **Örnek 4.3.10:** $x := x+1$ deyiminin yürütülmesi için n değişkeninin bir fonksiyonu olan bir theta gösterimi bulalım.

```
for i = to n
  for j =1 to i
      x = x+1
```

- **Örnek 4.3.11** : $x=x+1$ deyiminin yürütülmesi için n değişkeninin bir fonksiyonu olan bir başka theta gösterimi bulalım.

```
i=n
while (i ≥ 1) {
   x=x+1
   i=⌊i/2⌋
}
```

- **Örnek 4.3.12:** $x=x+1$ deyiminin yürütülmesi için n değişkeninin bir fonksiyonu olan bir başka theta gösterimi bulalım.

```
j = n
while (j≥1) {
  for i = 1 to j
    x = x+1
    j = ⌊j/2⌋
}
```

Sırasız Diziyi Taramak

Bu algoritma

$$s_1, s_2, \ldots, s_n$$

dizisi ve bir *key* değeri verildiğinde *key* değerinin dizi içindeki yerini döndürür. Eğer *key* değeri bulunamazsa 0 döndürülür.

Girdi: $s_1, s_2, \ldots, s_n$, n ve *key*

Çıktı: *key* değerinin damgası ya da 0

```
1       linear_search(s,n,key) {
2           for i=1 to n
3            if(key==s_i)
4                   return i
5         return 0
6       }
```

Matris Çarpımı

Bu algoritma nxn ebatlarındaki iki A ve B matrisinin çarpımını hesaplar

Girdi: A, B, n

Çıktı: *C*

```
matrix_product(A,B,n) {
        for i=1 to n
          for j=1 to n {
                C_ij=0
                for k=1 to n
                        C_ij=C_ij+A_ik*B_kj
        }
        return C
}
```

ÖZYİNELİ ALGORİTMALAR

Algoritma 4.4.1

Bu özyineli algoritma *n*! değerini hesaplar.

Girdi: 0 'dan büyük ya da eşit bir *n* tamsayısı

Çıktı: *n*

```
1.  factorial(n) {
2.      if (n==0)
3.          return 1
4.      return n*factorial(n-1)
5.  }
```

- **Teorem 4.4.2:** Algoritma 4.4.1 'ın çıktısı *n!* $(n \geq o)$ değeridir.

- **Örnek 4.4.3:** Bir robot 1 metre ya da 2 metre uzunluğunda adım atabilmektedir. Robotun *n* metreyi yürüyebilmesi için gerekli olan yolların sayısını hesaplayan algoritma.

Uzaklık	Adımların Dizisi	Yürüyüş çeşidi sayısı
1	1	1
2	1,1 ya da 2	2
3	1,1,1 ya da 1,2 ya da 2,1	3
4	1,1,1,1 ya da 1,1,2 ya da 1,2,1 ya da 2,1,1 ya da 2,2	5

Bu algoritma

$$walk(n)=\begin{cases}1, & n=1\\ 2, & n=2\\ walk(n-1)+walk(n-1), & n>2\end{cases}$$

ile tanımlanan fonksiyonu hesaplar.

Girdi: n

Çıktı: walk(n)

```
walk(n) {
        if (n==1 ∨ n==2)
                return n
        return walk(n-1) + walk(n-2)
}
```