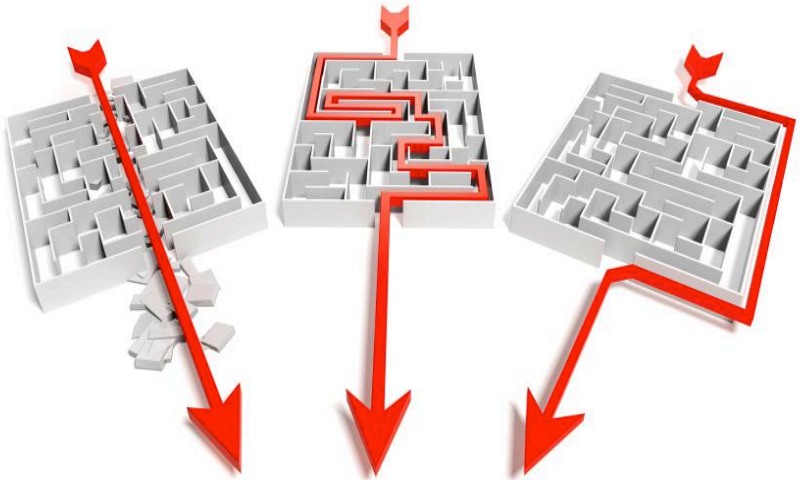
Algoritma Analizi
Big O

Suhap SAHIN
Onur GÖK

Giris



Verimlilik

Karsılastırma

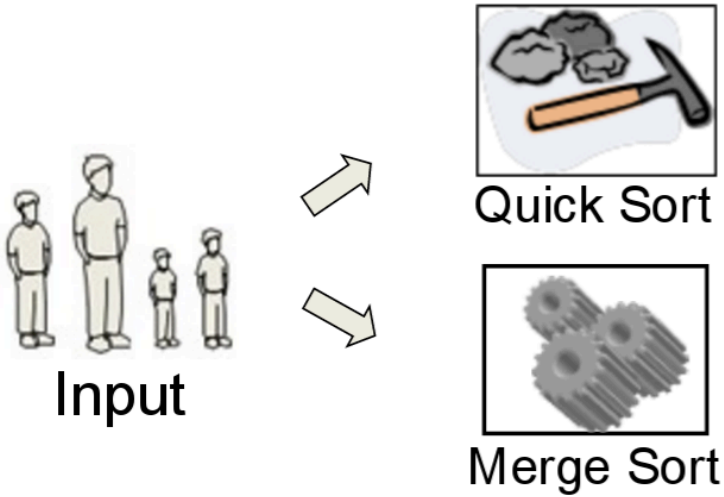
Giris

Input

Hangisi daha iyi?

Hangi kritere göre?

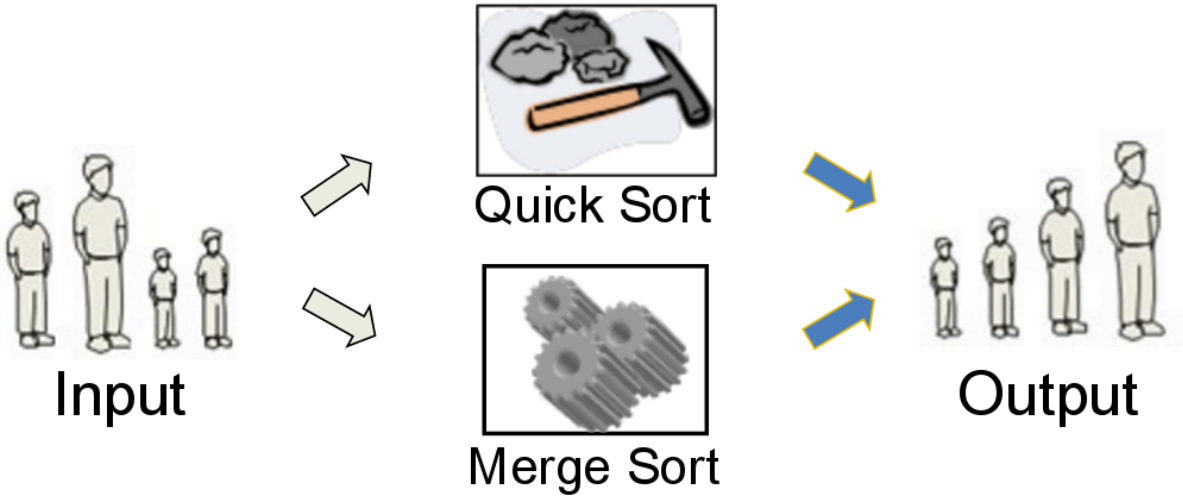
Giris



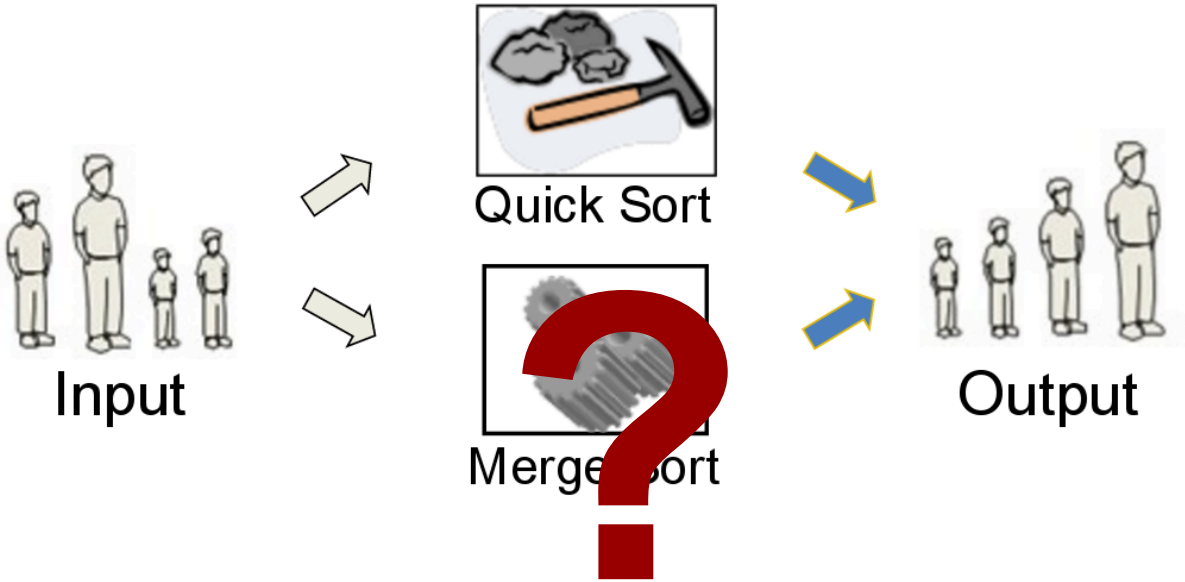
Hangisi daha iyi?

Hangi kritere göre?

Giris

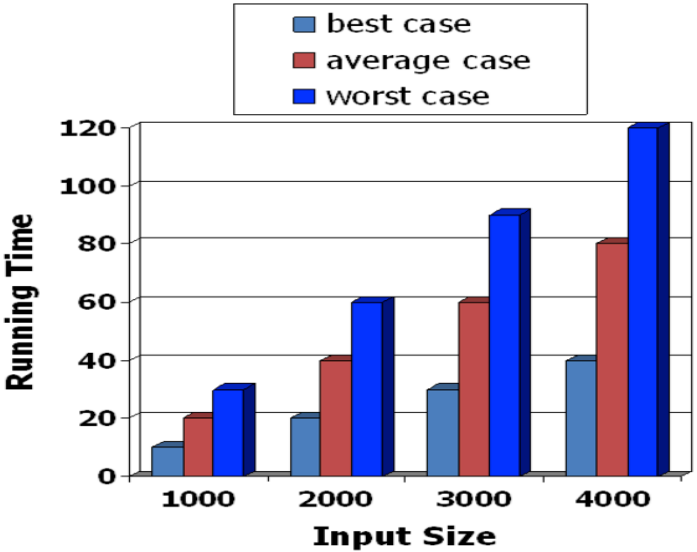


Giris

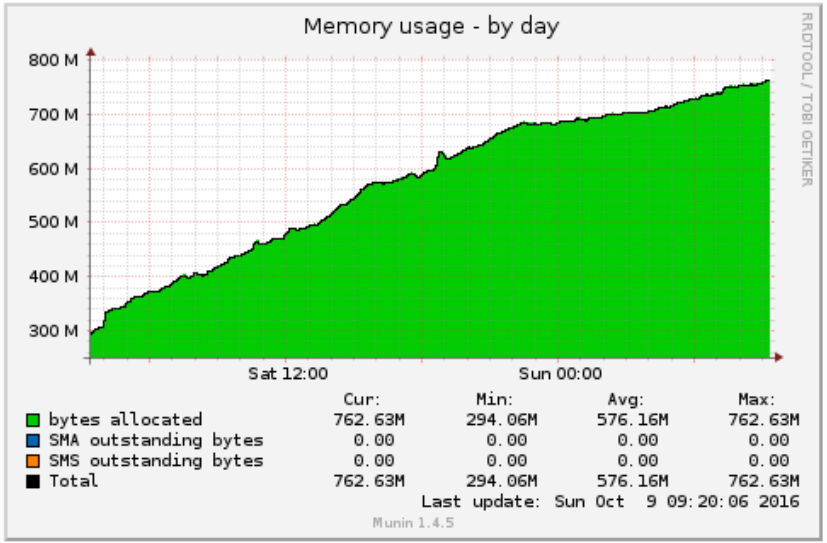


Verimin ölçülmesi

Çalısma Zamanı



Bellek kullanımı



Verimin ölçülmesi

insan maliyeti

Verimin hesaplanması

Çalısma Süresi

Karmasıklık

Parametre sayısı **X** (artım)

Çalısma süresi

Gerekli islem miktarı

Çalısma süresi karsılastırma

- iki farklı bilgisayar
- Aynı bilgisayar farklı zaman
- Kullanılan veri

Verim hesaplama

Veri, bilgisayar ve gerçeklestirime baglı Matematiksel bir analiz

Kıyaslama

Algoritma Analizi

$\sum$	\sum	$\bigcap$	\bigcap	$\bigodot$	\bigodot
$\prod$	\prod	$\bigcup$	\bigcup	$\bigotimes$	\bigotimes
$\coprod$	\coprod	$\bigsqcup$	\bigsqcup	$\bigoplus$	\bigoplus
$\int$	\int	$\bigvee$	\bigvee	$\biguplus$	\biguplus
$\oint$	\oint	$\bigwedge$	\bigwedge		

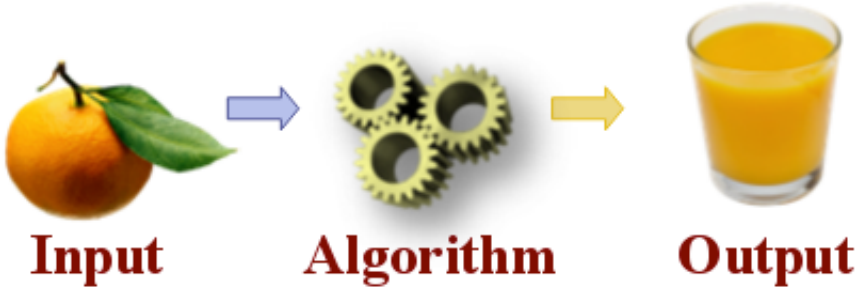
Algoritma Analizi

Matematiksel ifadesi nedir?

Temel hesap birimi **kaç adet** **yapılmalı?**

n

Parametre sayısı



f(n)

Verimlilik

n

T(n) Çalısma süresi

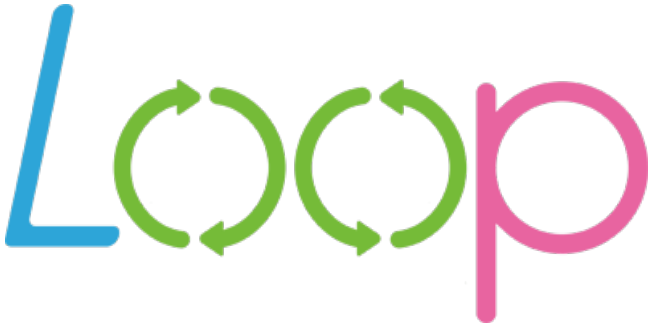
S(n) Bellek gereksinimi

Çalısma Süresi Hesabı

Çalısma Süresi (Tn): Algoritmanın belirli bir isleme (Temel hesap birimi) kaç kez gereksinim duydugunu gösteren bagıntıdır.

Temel hesap birimi:

- Programlama dilindeki deyimler
- Döngü sayısı
- Toplama islem sayısı
- Dosyaya erisim sayısı
- Atama sayısı



ÇSH: Aritmetik Ortalama

```
1  float bulOrta(int A[ ],int n){
2      float ortalama, toplam=0;
3      int k;
4      for (k=0; k<n;k++)
5          toplam +=A[k];  // döngü içinde gerçekleşen işlem
6      ortalama=toplam/n;
7      return ortalama;
8      }
```

ÇSH: Aritmetik Ortalama

	Temel Hesap Birimi	islem	tekrarı	Toplam
1	`float bulOrta(int A[ ],int n){`			
2	`  float ortalama, toplam=0;`			
3	`  int k;`			
4	`  for (k=0; k<n;k++)`	1,1,1	1,(n+1),n	2n+2
5	`  toplam +=A[k];`	1	n	n
6	`  ortalama=toplam/n;`	1	1	1
7	`  return ortalama;`	1	1	1
8	`  }`			

T(n)=3n+4

ÇSH: En Küçük eleman

```
1   int bulEnkucuk(int A[], int n){
2       int enkucuk;
3       int k;
4       enkucuk=A[0];
5       for (k=0; k<n;k++)
6           if(A[k]<enkucuk)
7               enkucuk=A[k];
8       return enkucuk;
9       }
```

ÇSH: En Küçük eleman

	Temel Hesap Birimi	islem	tekrarı	Toplam
1	`int bulEnkucuk(int A[], int n){`			
2	`  int enkucuk;`			
3	`  int k;`			
4	`  enkucuk=A[0];`	1	1	1
5	`  for (k=0; k<n;k++)`	1,1,1	1,n,(n+1)	2n
6	`  if(A[k]<enkucuk)`	1	n-1	n-1
7	`  enkucuk=A[k];`	1	n-1	n-1
8	`  return enkucuk;`	1	1	1
9	`  }`			

T(n)=4n

ÇSH: Matris Toplama

```
1  void toplamMatris(int A[2][2], int B[2][2]){
2      int C[2][2];
3      int i,j;
4      for(i=0;i<2;i++)
5          for(j=0;j<2;j++)
6              C[i][j]=A[i][j]+B[i][j];
7      }
```

ÇSH: Matris Toplama

	Temel Hesap Birimi	islem	tekrarı	Toplam
1	`void toplamMatris(int A[2][2], int B[2][2]){`			
2	`  int C[2][2];`			
3	`  int i,j;`			
4	`  for(i=0;i<2;i++)`	1,1,1	1,n,(n+1)	2n+2
5	`  for(j=0;j<2;j++)`	1,1,1	n*(2m+2)	2nm+2n
6	`  C[i][j]=A[i][j]+B[i][j];`	1	n*m	nm
7	`  }`			

T(n)=3nm+4n+2

ÇSH: Faktöriyel Hesabı

```
1  int faktoriyel(int n){
2      if(n<=1)
3          return 1;
4      else
5          return (n*faktoriyel(n-1));
6  }
```

ÇSH: Faktöriyel Hesabı

	Temel Hesap Birimi	islem	tekrarı	Toplam
1	`int faktoriyel(int n){`			
2	`  if(n<=1)`	**1**	**n**	**n**
3	`  return 1;`	**1**	**1**	**1**
4	`  else`			
5	`  return (n*faktoriyel(n-1));`	**1**	**n-1**	**n-1**
6	`}`			

T(n)=2n

Karmasiklik

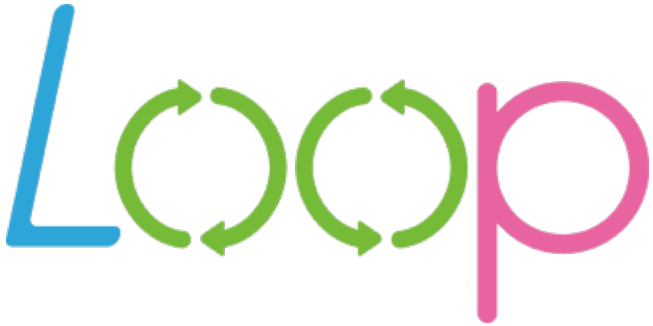
Karmasıklıgı ifade edebilmek için matematiksel ifadeler kullanılmaktadır.

- Küçük-o (small-o)
- Büyük-O (big-o, veya big-oh diye de geçer)
- Teta (Theta Θ, sadece büyük tetadan bahsedebiliz)
- Büyük omega (big-Ω)
- Küçük omega(small-ω)

Lineer Döngü

Asagıdaki kod kaç defa döner:

```
i = 1
loop( i <= 1000 )
        (loop body)
        i = i + 1
end loop
```



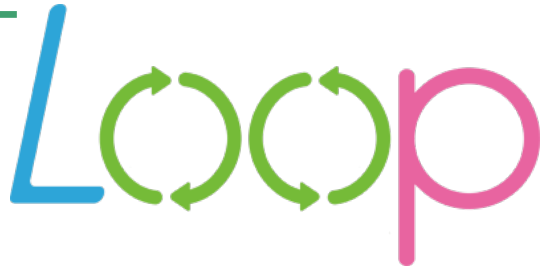
Logaritmik Döngü

Çarpım Döngüsü

```
i = 1
loop( i <= 1000 )
        (loop body)
        i = i * 2
end loop
```

Bölüm Döngüsü

```
i = 1000
loop( i >= 1 )
        (loop body)
        i = i / 2
end loop
```



Logaritmik Döngü

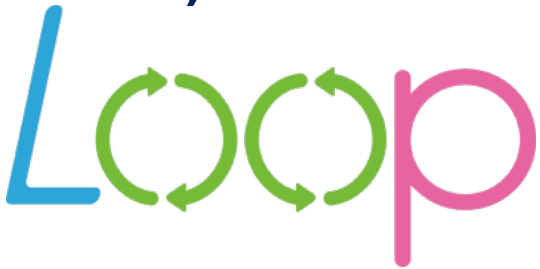
çarpım: $2^{\text{iterasyon}} < 1000$

bölüm: $1000 / 2^{\text{iterasyon}} >= 1$

Çarpım		Bölüm	
iterasyon	i	iterasyon	i
1	1	1	1000
2	2	2	500
3	4	3	250
4	8	4	125
5	16	5	62
6	32	6	31
7	64	7	15
8	128	8	7
9	256	9	3
10	512	10	1
çıkış	1024	çıkış	0

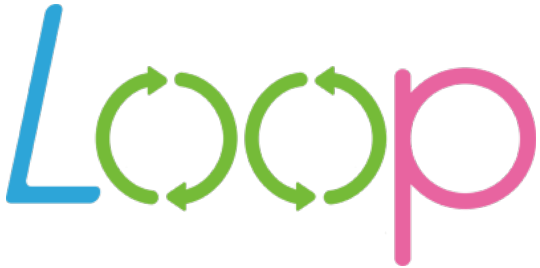
iç içe döngüler

- Karesel (Quadratic)
- Bagımlı Karesel (Dependent Quadratic)
- Lineer Logaritmik (Linear Logarithmic)



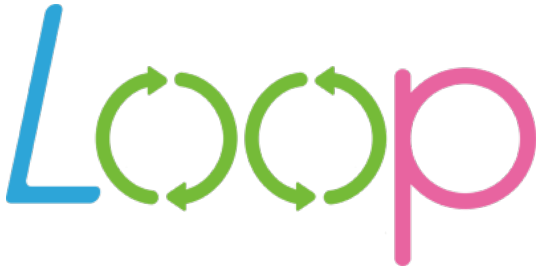
Karesel

```
i = 1
loop( i <= 10 )
    j = 1
    loop( j <= 10 )
        (loop body)
        j = j + 1
    end loop
    i = i + 1
end loop
```



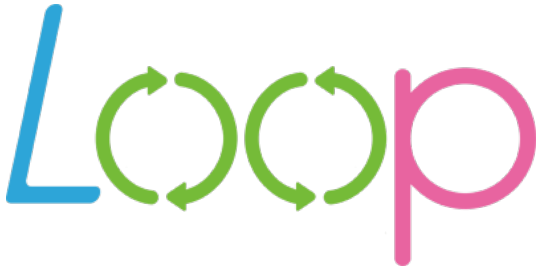
Bagımlı Karesel

```
i = 1
loop( i <= 10 )
    j = 1
    loop( j <= i )
        (loop body)
        j = j + 1
    end loop
    i = i + 1
end loop
```



Lineer Logaritmik

```
i = 1
loop( i <= 10 )
    j = 1
    loop( j <= 10 )
        (loop body)
        j = j * 2
    end loop
    i = i + 1
end loop
```



Big O Gösterimi

f(n) fonksiyonundan O(n) degerini hesaplamak için asagıdaki adımlar gerçeklestirilir:

1) Tüm katsayılar 1 yapılır
2) En büyük terim alınır, digerleri gözardı edilir.

Terimlerin küçükten büyüge sırası asagıdaki gibidir:

sabit log2n n nlog2n n2 n3…nk 2n n!

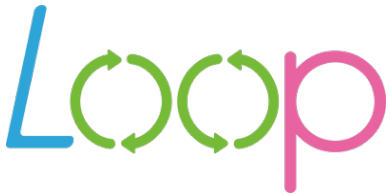
Big O Gösterimi

$$f(n) = \frac{n(n+1)}{2} = \frac{1}{2}n^2 + \frac{1}{2}n$$

$$n^2 + n$$

$$n^2$$

$$O(n^2)$$

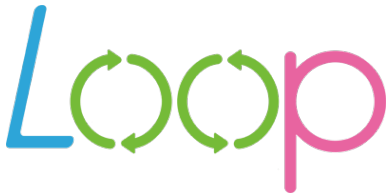


Big O Gösterimi

$$f(n) = 6n^4 \log n + 12n^3 + 2n^2 + n + 128$$

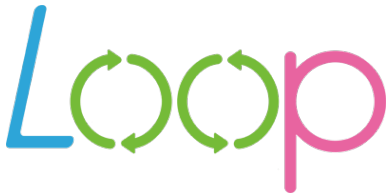
$$f(n) = n^4 \log n + n^3 + n^2 + n + 1$$

$$O(n^4 \log n)$$



Big O Gösterimi

Big O	Degisim sekli
O(1)	Sabit, degismiyor
O(log n)	Logaritmik artıyor
O(n)	Dogrusal artıyor
O(n log n)	Dogrusal çarpanlı logaritmik
O(n2)	Karesel artıyor
O(n3)	Kübik artıyor
O(2n)	iki tabanında üssel artıyor
O(10n)	On tabanında üssel artıyor
O(n!)	Faktöriyel olarak artıyor



Big O Gösterimi

Lineer Döngü

```
i = 1
loop( i <= 1000 )
        (loop body)
        i = i + 1
end loop
```

f(n)=n/2 => f(n)=n => O(n)

Logaritmik Döngü

```
//Çarpım Döngüsü
i = 1
loop( i < 1000 )
        (loop body)
        i = i * 2
end loop
```

$2^{iterasyon}<1000$

```
//Bölüm Döngüsü
i = 1000
loop( i >= 1 )
        (loop body)
        i = i / 2
end loop
```

$2^{i}=n \Rightarrow f(n)=\log_2 n$

$1000/2^{iterasyon}>=1$

O(log n)

Big O Gösterimi

```
// Karesel
i = 1
loop( i <= 10 )
        j = 1
        loop( j <= 10 )

        (loop body)
        j = j + 1
        end loop
        i = i + 1
end loop
```

f(n)=n*n => O(n^2)

```
// Bağımlı Karesel
i = 1
loop( i <= 10 )
        j = 1
        loop( j <= i )

        (loop body)
        j = j + 1
        end loop
        i = i + 1
end loop
```

f(n)=[n*(n+1)]/2 => O(n^2)

```
// Lineer Logaritmik
i = 1
loop( i <= 10 )
        j = 1
        loop( j <= 10 )

        (loop body)
        j = j * 2
        end loop
        i = i + 1
end loop
```

f(n)=n*$\log_2 n$

O(n log n)

En kötü durum

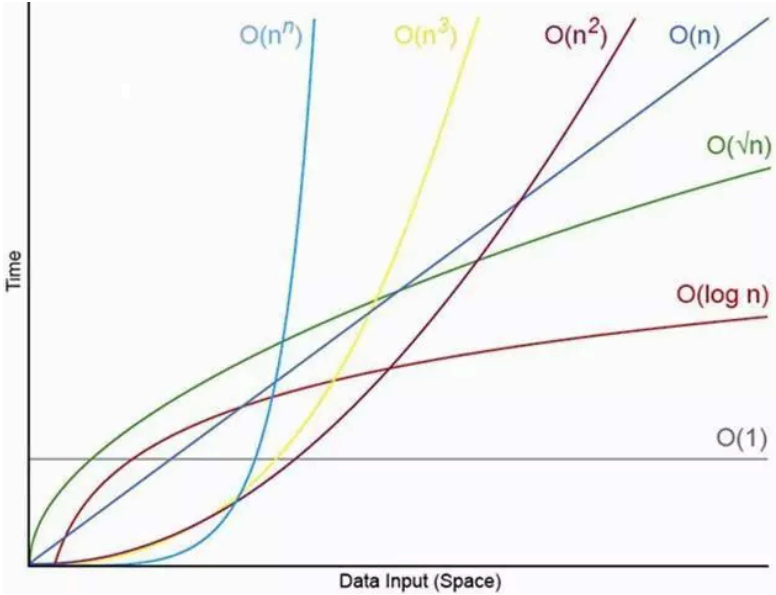
Algoritmalar , aynı sayıda veri için adım sayısı koşula bağlı olarak aynı sayıda dönmeyebilir.

Bu yüzden algoritmaların karmaşıklığını O(n) değerini bulmak için en kötü durumun bulunması gerekebilir. En kötü durum en fazla döngünün olması durumudur.

Arama: aradığınız kayıt listenin başında ise 1 kontrol, listenin sonunda ise listedeki eleman sayısı n ise n kontrol gerektirir.

Büyük O Gösterimi

Algoritmanın büyüme katsayısı



Sorular