Algoritmik Karmaşıklık

Dr. Hakan TEMİZ

Karmaşıklık

- Karmaşıklık belirli bir işlevi gerçekleştirmek için gereken kaynak (işlemci, belek vb.) miktarının bir ölçüsüdür.
- Disk okuma/yazma, bellek kullanımı vb. açısından ölçmek mümkün fakat genellikle işlemci (CPU, GPU vb.) özelinde ele alınır. Bu nedenle, karmaşıklığı belirli bir işlevi gerçekleştirmek için gereken hesaplama veya işlem sayısının bir ölçüsü olarak ele alınır.
- Karmaşıklığın hesaplanmasında genellikle kesin işlem sayısını ölçmek gerekli değildir. Problemin etkilendiği faktörlerin artmasına bağlı olarak yapılacak işin (hesaplama vb.) boyutunun nasıl arttığıdır.

Karmaşıklık

- Algoritma ve veri yapılarının verimliliği hakkında iyimser, ortalama ve kötümser durum şeklinde ve bir dereceye kadar bir öngörü üretmeyi amaçlar.
- Her halükarda, karmaşıklık hesabı, beklenen performansın kesin bir ölçüsü yerine belirli sınırlar ve limitler dahilinde tahminlenmesidir.
- Problemin etmenleri/girdileri arttıkça, işlem sayısı
  - aynı mı kalır?
  - iki katına mı çıkar?
  - girdiler ile doğru orantılı mı artar?
  - üstel olarak mı artar?

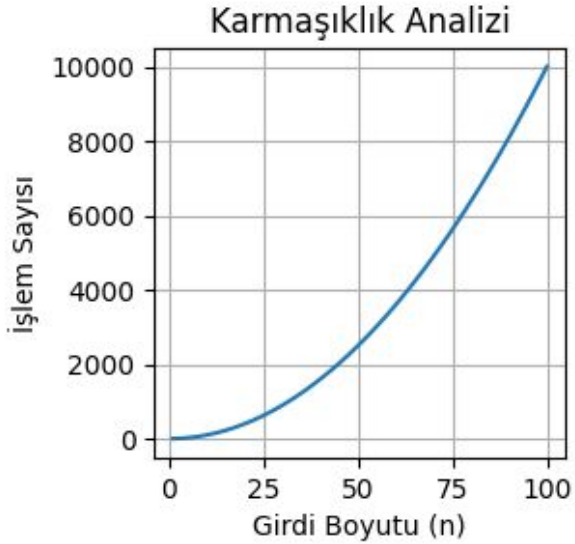
  vb. sorulara cevap aranır.
- Yani, algoritma ne kadar iyi ölçeklenebilmektedir?

Karmaşıklık - Örnek

Geleneksel çarpma işlemi: n basamaklı iki sayının çarpımı n x n ≈ n^2 işlem gerektirir.

```
     1234
     5678
  x______
     9872
    8638
   7404
  6170
_________
  7006652
```

$\approx n^2$ $(4^2=16)$



Girdi boyutu (basamak sayısı) attığında işlem sayısı karesel artmaktadır. Dolayısıyla, bu algoritmanın (geleneksel çarpma) karmaşıklığı üstel (örnekte karesel) artmaktadır. İşlemci üzerindeki iş yükü karesel artacak ve daha yavaş çalışacaktır.

Karmaşıklık - Asimtotik (Asymptotic) Analiz

Bir algoritmanın asimtotik analizi, algoritmanın çalışma süresinin hesaplanmasını ifade eder. Hesaplanmak istenen süre gerçek çalışma süresi değildir. Hesaplama, algoritmanın girdi boyutu açısından ele alınır ve girdi boyutundaki artışla birlikte geçen zamanın nasıl arttığı ölçülür. Algoritmik analizde genel olarak üç tür analiz yapılır:

- En İyi Durum Analizi - Omega (Ω) Notasyonu
- Ortalama Durum Analizi - Theta (Θ) Notasyonu
- En Kötü Durum Analizi - Büyük O Notasyonu

Genellikle, bir algoritmayı analiz etmek için en kötü durum analizi dikkate alınır. Çünkü çalışma süresi için üst sınırın anlaşılmasını sağlar. En iyi durum (iyimser) analizi ise bize alt sınırı sağladığından daha önemsizdir. Diğer taraftan, ortalama durum analizinin hesaplanması genellikle çok zordur. Dolayısıyla, genellikle en iyi ve en kötü durumlar incelenir. İlerleyen bölümlerde En iyi ve Ortalama durum analizleri kısaca verilecek ve En Kötü Durum Analizi detaylı ele alınacaktır.

Karmaşıklık - Asimtotik (Asymptotic) Analiz

Örnek Program:

```
# Bir öğeyi aramak için doğrusal arama programı
def ara(dizi, x):
  for i in range(len(dizi)):
    if dizi [i] == x:
      return i
  return -1

dizi= [2, 5, 4, 1, 9, 3]
x=4
print("x öğesinin dizin (indeks) konumu :" ,ara(dizi, x))
```

Karmaşıklık - En İyi Durum Analizi - Omega (Ω) Notasyonu

Bir F(n) fonksiyonunun Omega notasyonu olan T(n) fonksiyonu şu şekilde tanımlanır:

$T(n)=\Omega(F(n)$ öyle ki: $\forall\, n>=n_0$ için, $0 \leq c(F(n)) \leq T(n)$ koşulunu sağlayan bir n_0 ve c sabiti varsa)

Dolayısıyla, Omega gösterimi, verilen algoritmadan daha az veya ona eşit olan en yüksek büyüme oranı T(n)'yi verir. Diğer bir deyişle, algoritmanın çalışması için gerekecek minimum zamanı (alt sınır çalışma süresi) ifade eder.

Verilen örnek açısından, en iyi durum, aranacak öğenin ilk indekste (ilk karşılaştırmada) bulunmasıdır. Örneğimiz için en iyi durum zaman karmaşıklığı, dizinin ne kadar uzun olduğundan bağımsız olmakla beraber, öğenin ilk sırada olduğu durumdur. Bu nedenle, en iyi durum zaman karmaşıklığı $\Omega(1)$ olur.

```
dizi= [4, 2, 5, 1, 9, 3] # aranan en başta
x=4 # aranan
print("x öğesinin dizin (indeks) konumu :" ,ara(dizi, x))
```

Karmaşıklık - Ortalama Durum Analizi - Theta (Θ) Notasyonu

Bir F(n) fonksiyonunun Θ notasyonu olan T(n) fonksiyonu şu şekilde tanımlanır:

$T(n)=\Theta(F(n))$ öyle ki: $\forall n>=n_0$ için, $0 \leq c_1(F(n)) \leq T(n) \leq c_2(F(n))$ koşulunu sağlayan n0, c_1 ve c_2 sabitleri varsa)

Genellikle belirli bir fonksiyonun hem üst hem de alt sınırlarının aynı olduğu durumlar vardır ve Theta gösteriminin amacı bunun böyle olup olmadığını belirlemektir.

Aranan öğenin dizide bulunabileceği tüm olası durumlar ele alınır ve ardından ortalama çalışma süresi karmaşıklığı hesaplanır. n elemanlı bir dizide, aranacak öğe 0. indekste ise karşılaştırma sayısı 1 olur. Çünkü ilk karşılaştırmada aranan elemana rastlanır. Benzer şekilde, 1,2,3,...(n−1) dizinlerde bulunan öğeler için karşılaştırma sayısı sırasıyla 2,3,...,n olur. Böylece, ortalama zaman karmaşıklığı şu şekilde tanımlanabilir:

Ortalama Durum Karmaşıklığı = (1+ 2+ 3+ … + n) / n = n(n + 1) / 2

```
dizi= [1, 5, 2, 4, 9, 3] # aranan ortalarda
x=4 # aranan
print("x öğesinin dizin (indeks) konumu :" ,ara(dizi, x))
```

Karmaşıklık - En Kötü Durum Analizi - Büyük *O* Notasyonu

- En kötü durum çalışma zamanı karmaşıklığını, yani algoritmanın alacağı maksimum zamanı ölçmeye odaklanır.
- *O* (omicron), karmaşıklıktaki artış miktarının oransal büyüklüğü (order) anlamına gelir.
- Bir $F(n)$ fonksiyonunun Büyük O 'su olan $T(n)$ fonksiyonu şu şekilde tanımlanır:

$T(n)=O(F(n))$ öyle ki: $\forall (n>=n_0)$ için, $T(n)<=c(F(n))$ koşulunu sağlayan bir n_0 ve c sabiti varsa)

$\mathbf{n_0}$ ve **c** yegane değildir; farklı değerleri de çözümü sağlayabilir.

- Burada $T(n)$ fonksiyonu, $F(n)$ 'in sıkı üst sınırını temsil eder.

```
dizi= [2, 5, 3, 1, 9, 4] # aranan sonda, (veya hiç yok)
x=4 # aranan
print("x öğesinin dizin (indeks) konumu :" ,ara(dizi, x))
```

Karmaşıklık - Büyük *O* Notasyonu

- n boyundaki bir problemi çözmek için gereken zaman (adım sayısı) şöyle olsun:

  $\mathbf{T(n) = 2n^2 + 2n + 5}$

- n 'in oldukça büyük değerleri için fonksiyonun nasıl değiştiğine bakalım.
- n büyüdükçe n^2 terimi o kadar hızlı büyüyecektir ki diğer terimlerin büyüme hızı buna kıyasla ihmal edilebilecek kadar düşük kalacaktır. Örneğin,

  n=100 ve **n=1000** için $\mathbf{2n^2}$ terimi **2n** teriminin, sırasıyla **100** ve **1000** **katı** olacaktır. Dolayısıyla ikinci terimin değeri tüm ifadenin değerini belirlemede çoğu durumda ihmal edilebilir bir etkiye sahip olacaktır.

ilk terim ($2n^2$) yerine $\mathbf{n^3}$ olsaydı, **n=100** ve **n=1000** için, ilk terimin ikinci terime göre **oranı** sırasıyla **5 bin** ve **500 bin** olacaktı. Benzer yaklaşımla, ifadedeki sabitin (5) hiç bir önemi olmadığı da açıkça görülmektedir.

Bu fonksiyon için $\mathbf{T(n) \in O(n^2)}$, yani, $\mathbf{n^2}$ dereceden karmaşıklığa sahiptir denir.

Karmaşıklık - Büyük *O* Hesabı

Büyük O gösterimi sabit faktörleri ve düşük dereceli terimleri göz ardı etmemize ve bir fonksiyonun büyümesini etkileyen ana bileşenlere odaklanır.

d mertebeden bir f(n) polinom fonksiyonun artış hızı şu şekilde belirlenir:

$f(n) = a_0 + a_1 n + \ldots + a_d n^d$, ve $a_d > 0 \Rightarrow$ f(n), $\mathbf{O(n^d)}$ 'dir.

Ispat:

$n \geq 1$ için $1 \leq n \leq n^2 \leq \cdots \leq n^d$ 'dir. Böylece,

$a_0 + a_1 n + a_2 n^2 + \cdots + a_d n^d \leq (|a_0| + |a_1| + |a_2| + \cdots + |a_d|) n^d$

$c=|a_0|+|a_1|+|a_2|+\cdots+|a_d|$ ve $n_0=1$ şeklinde tanımlamak suretiyle, f(n) 'in $\mathbf{O(n^d)}$ olduğunu göstermiş oluruz.

Bir polinomdaki en yüksek dereceli terim, o polinomun asimtotik büyüme oranını belirleyen terimdir.

Karmaşıklık - Büyük *O* Hesabı

Büyük O gösterimi sabit faktörleri ve düşük dereceli terimleri göz ardı etmemize ve bir fonksiyonun büyümesini etkileyen ana bileşenlere odaklanır. Aşağıdaki fonksiyonun ana bileşeni $\mathbf{5n^4}$ terimidir ve $\mathbf{O(n^4)}$ 'tür.

$$\mathbf{5n^4 + 3n^3 + 2n^2 + 4n + 1}$$

fonksiyonu için Büyük O hesabını yapalım.

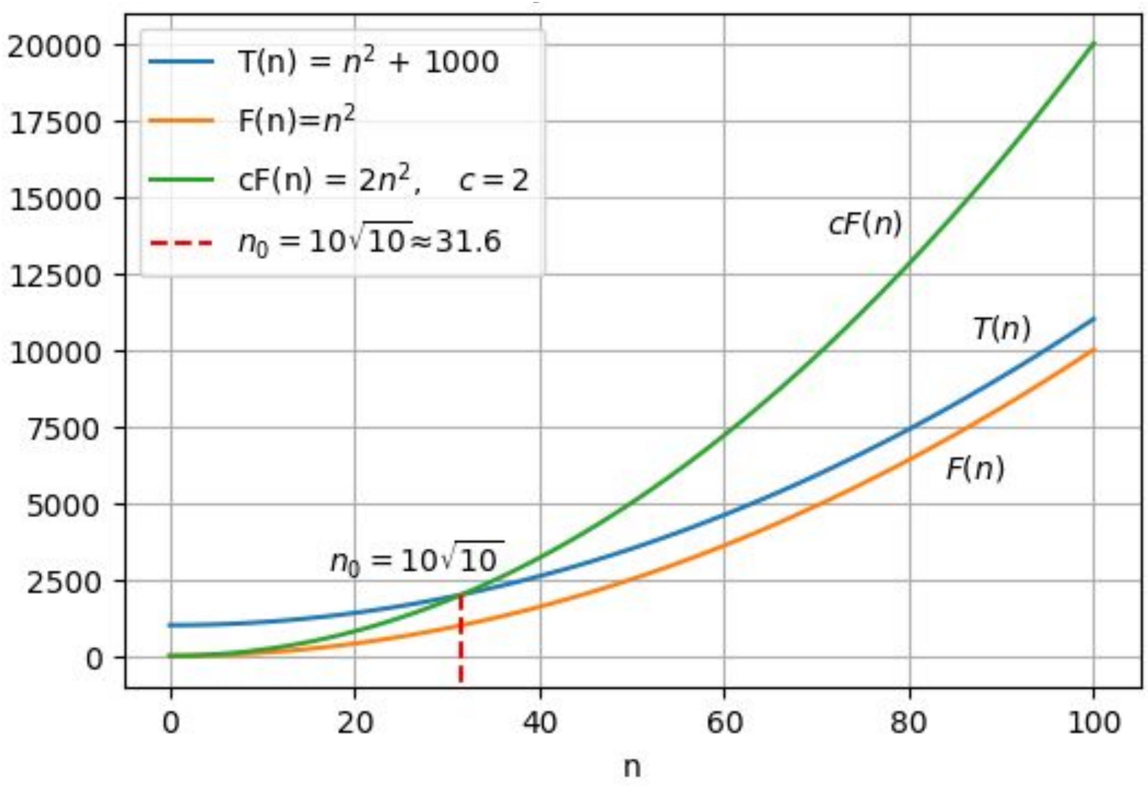
Bulmak istediğimiz fonksiyon, bu ifadeden daha büyük değerler üretebilmesi için mertebesi en azından $\mathbf{n^4}$ formunda olmalıdır. Öyle bir de **c** katsayısına sahip olmalıdır ki, verilen fonksiyonun tüm terimlerinin değerleri toplamından daha büyük bir değer üretmelidir. Bu katsayı için, terimlerin minimum toplamını alabiliriz. Böylece yeni fonksiyonun değerinin daha büyük olmasını sağlarız.

$5n^4 + 3n^3 + 2n^2 + 4n + 1 \leq (5 + 4 + 3 + 2 + 1)n^4 = cn^4$ ve bu durumda,

$n \geq n_0 = 1$ için, $c = 15$ olur.

Karmaşıklık - Büyük *O* Hesabı

Aşağıdaki grafikte, $T(n) = n^2+1000 = O(n^2)$ olduğunu görebiliriz; $C = 2$ ve $n_0 \approx 31.6$ 'dır.



Karmaşıklık - Büyük *O* Hesabı

Soru: $f(n) = 4n + 9$ için, üst sınırını bulunuz?

Çözüm:

$$4n + 9 \leq 5n\ , \quad \forall\ n \geq 9$$

$\therefore\ 4n + 9 = O(n)$ 'dir. $c = 5$ ve $n_0 = 9$ olmak kaydıyla.

Soru: $f(n) = n^2 + 1$ için, üst sınırını bulunuz?

Çözüm:

$$n^2 + 1 \leq 2n^2\ , \quad \forall\ n \geq 1$$

$\therefore\ n^2 + 1 = O(n^2)$ 'dir. $c = 2$ ve $n_0 = 1$ olmak kaydıyla.

Karmaşıklık - Büyük *O* Hesabı

Soru: $f(n) = n^4 + 100n^2 + 50$ için, üst sınırını bulunuz.

Çözüm:

$$n^4 + 100n^2 + 50 \leq 2n^4 \,, \quad \forall\ n \geq 11$$

$\therefore\ n^4 + 100n^2 + 50 = O(n^4)$ 'dir. $c = 2$ ve $n_0 = 11$ olmak kaydıyla.

Soru: $f(n) = 2n^3 - 2n^2$ için, üst sınırını bulunuz.

Çözüm:

$$2n^3 - 2n^2 \leq 2n^3 \,, \quad \forall\ n \geq 1$$

$\therefore\ 2n^3 - 2n^2 = O(n^3)$ 'dir. $c = 2$ ve $n^0 = 1$ olmak kaydıyla.

Karmaşıklık - Büyük *O* Hesabı

Soru: $f(n) = n$ için, üst sınırını bulunuz.

Çözüm:

$$n \leq n \,, \quad \forall \; n \geq 1$$

$\therefore \; n = O(n)$ ′dir. $c = 1$ ve $n_0 = 1$ olmak kaydıyla.

Soru: $f(n) = 500$ için, üst sınırını bulunuz.

Çözüm:

$$500 \leq 500 \,, \quad \forall \; n \geq 1$$

$\therefore \; 500 = O(1)$ ′dir. $c = 1$ ve $n_0 = 1$ olmak kaydıyla.

Karmaşıklık - Büyük *O* Hesabı

Soru: $f(n) = 100n + 10$ için, iki farklı üst sınır bulunuz.

Çözüm 1:

$$100n + 10 \leq 100n + n$$

$$100n + 10 \leq 101n, \qquad \forall\ n \geq 10$$

$\therefore$ $101n$, $c = 101$ ve $n_0 = 10$ olmak kaydıyla, bir üst sınırdır.

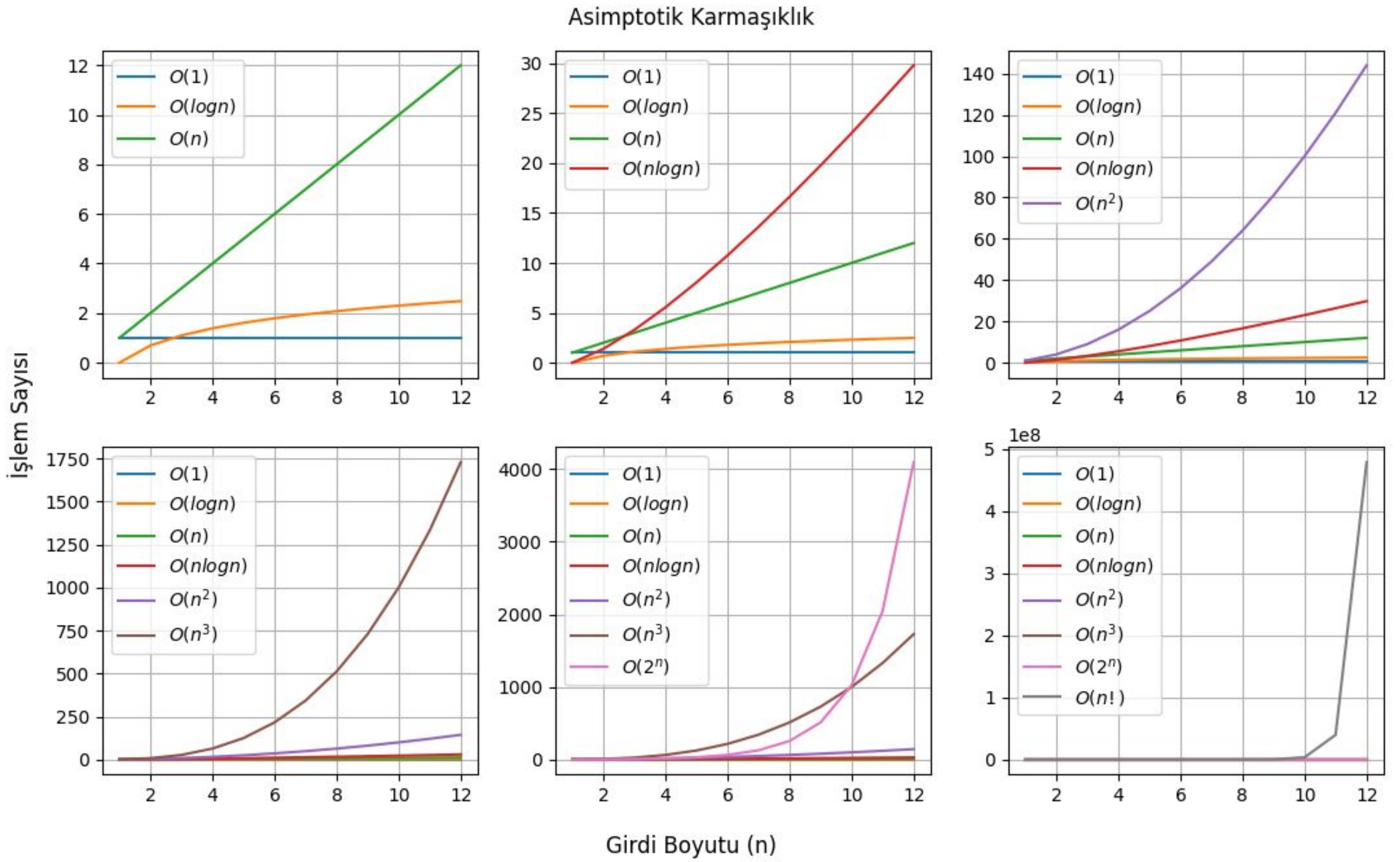
Çözüm 2:

$$100n + 10 \leq 100n + 10n$$

$$100n + 10 \leq 110n\ , \qquad \forall\ n \geq 1$$

$\therefore$ $110n$, $c = 101$ ve $n_0 = 1$ olmak kaydıyla, bir üst sınırdır.

Big-*O* Fonksiyonu	Adı	Açıklama
***O*(1)**	Sabit	**En hızlı.** Verinin büyüklüğünden bağımsız, her zaman aynı miktarda zaman alır. Örnek: Bir dizinin bir öğesini indeks ile aramak.
***O*(log n)**	Logaritmik	**Oldukça hızlı.** Her yinelemede veri miktarı yarıya iner. 100 öğe için cevabı bulmak yaklaşık 7 adım; 1000 öğe için 10 adım ve 1 milyon öğe için yalnızca 20 adım sürer. Büyük miktarda veri için bile süper hızlıdır. Örneğin, ikili arama, birleşme sıralama (merge-sort) veya liste sıralama.
***O*(n)**	Doğrusal	**Hızlı.** 100 öğe varsa 100 birim iş gerekir. Öğe sayısı iki katına çıkarsa 200 birim iş gerekir. Örnek: sıralı arama.
***O*(n log n)**	Doğrusal logaritmik	**Biraz Yavaş.** Doğrusaldan biraz daha kötü ama çok da kötü değil. Örnek: en hızlı genel amaçlı sıralama algoritmaları.
***O*(n^2)**	Karesel	**Oldukça yavaş.** 100 öğelik bir problemde 100^2 =10.000 birim iş gerekir. Öğe sayısının 2 katına çıkması, 2^2=4 kat daha yavaşlamaya neden olur. Örnek: ekleme sıralaması gibi iç içe döngülü algoritmalar.
***O*(n^3)**	Kübik	**Çok yavaş.** 100 öğeli bir problemde 100^3 =1.000.000 birimlik iş gerekir. Giriş boyutunun 2 katına çıkması 2^3=8 kat daha yavaş hale getirir. Örnek: matris çarpımı.
***O*(2^n)**	Üstel	**Aşırı yavaş.** Bu tür bir algoritma istenmez fakat bazen başka seçenek kalmaz. Problemin girişinde sadece ve sadece bir bitlik ilaveçalışma süresini iki katına çıkarır. Örnek: gezici satış elemanı (traveling salesperson) problemi veya Hanoi kuleleri problemi.
***O*(n!)**	Faktöryel	**Kabul edilemez yavaşlıkta.** Kelimenin tam anlamıyla bir milyon yıl sürer.


Asimptotik Karmaşıklık
İşlem Sayısı
Girdi Boyutu (n)
$O(1)$
$O(logn)$
$O(n)$
$O(nlogn)$
$O(n^2)$
$O(n^3)$
$O(2^n)$
$O(n!)$
1e8

Karmaşıklık - Büyük *O* Notasyonu

- Bir fonksiyonun en yüksek dereceli terimi Büyük *O* zaman karmaşıklığını belirler. Örneğin,

  $\mathbf{f(x) = 11n\log_2 n + 45}$ fonksiyonu için zaman karmaşıklığı **O(nlogn)** 'dir.

- Yüksek dereceden bir fonksiyon, kendinden daha az dereceli (karmaşıklığa sahip) tüm fonksiyonları kapsar. Daha somut konuşmak gerekirse, $\mathbf{O(n^2)}$**,** ayrıca **O(n)** , **O(nlogn)** ve benzeri fonksiyonları da içerir.

Karmaşıklık - Algoritmalarda Büyük O Hesabı

- Algoritmanın karmaşıklık boyutunu bulmak için bir dizi temel işlemin toplam çalışma süresini bulmak gerekir. Basit işlemlerin karmaşıklık sınıflarını (fonksiyonlarını) birleştirerek daha karmaşık, birleşik işlemlerin karmaşıklık sınıfı bulunur.
- Birleşik ifadeler analiz edilir, toplam karmaşıklık bulunur. İki karmaşıklık sınıfını toplanarak birleştirilebilir. Bu tür bir durum, iki ardışık işlemin varlığında gerçekleşir.
- Örnek: bir diziye bir öğe ekleme ve ardından listeyi sıralama işlemleri. Bir öğe eklemenin **O(n)** sürede ve sıralama işleminin **O(nlogn)** sürede gerçekleştiği bilgisine dayanarak; toplam zaman karmaşıklığını **O(n+nlogn)** olarak yazılabilir. Yani iki işlev **O(...)** içine alınır ve sadece en yüksek dereceli terimle ilgilenilir. Bu durumda, sadece **O(nlogn)** kalır.
- Bir işlem bir döngüde tekrarlanırsa, karmaşıklık sınıfı işlemin gerçekleştirildiği yineleme sayısıyla çarpılır. Örneğin, zaman karmaşıklığı **O(f(n))** olan bir işlem **O(n)** kez tekrarlanırsa iki karmaşıklık çarpılır:

O(f(n)) * O(n) = O(nf(n))

Karmaşıklık - Algoritmalarda Büyük O Hesabı

Bazı Özellikler

Çarpma:

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

Toplama:

$$O(f(n)) + O(g(n)) = O(\max\{ f(n), g(n) \})$$

Sabitle Çarpma:

$$O(kg(n)) = O(g(n)),\ k \neq 0$$

Sabitle Toplama:

$O(k + g(n)) = O(g(n))$, $g(n) \in O(1)$ değilse; öyleyse $O(1)$ ′dir.

Karmaşıklık - Algoritmalarda Büyük O Hesabı

- Zaman karmaşıklığı $O(n^2)$ olan bir f(...) fonksiyonu bir while döngüsünde n defa yürütülürse,

```
for i in range(n):
  f(...)
```

- döngünün zaman karmaşıklığı $O(n^2)*O(n)=O(n*n^2)=O(n^3)$ olur. Anlaşılacağı üzere, icra edilen operasyonun karmaşıklık derecesi döngünün tekrar sayısı ile çarpılmaktadır.
- Bir döngünün çalışma süresi en fazla döngü içindeki ifadelerin çalışma süresinin yineleme sayısıyla çarpımıdır.

Karmaşıklık - Algoritmalarda Büyük O Hesabı

- Bir iç içe döngü, yani başka bir döngünün içine yerleştirilmiş bir döngü, her iki döngünün de n kez çalıştığını varsayarak, aşağıdaki örnekte gösterildiği gibi **n^2** sürede çalışacaktır.

```
for i in range(0,n):
  for j in range(0,n)
    # yürütülecek ifade
```

- Buradaki her bir ifade bir c sabiti ile ifade edilirse, c*n*n defa yürütülür. Böylece, çalışma zamanı,

$$cnn = cn^2 = O(n^2) \text{ olur.}$$

Karmaşıklık - Algoritmalarda Büyük O Hesabı

- Döngü içerisindeki ardışık ifadeler toplanır ve sonuç döngünün yineleme sayısı ile çarpılır.

```
n=100 # c0 -> 1 kez
# n defa yürüt
for i in range(0,n):
  print(i) # c1 -> n kez
  # n defa yürüt

for i in range(0,n):
  # n defa yürüt
  for j in range(0,n):
    print(j) # c2 -> n^2 kez
```

Kodun karmaşıklığı şu şekilde hesaplanır: $c_0+c_1n+c_2n^2=O(n^2)$

Karmaşıklık - Algoritmalarda Büyük O Hesabı

- Problem boyutu 1/2 oranında azaltılabilirse, karmaşıklık, logaritmik (2 tabanında) boyuta indirgenir.

```
j=1
while j<= n:
  j *= 2
  print(j)
```

- Her yinelemede, j iki katına çıkmaktadır. n=10 için, j , 2, 4, 8 ve 16 değerlerine ulaşır. n 'in değerinin her ikiye katlanışında, yineleme sayısı sadece 1 adet artar. Toplam k adet yineleme için,

$$\log_2(2^k)=\log_2 n$$

$$k\log_2 2=\log_2 n$$

$k=\log(n)$ olur. Dolayısıyla, zaman karmaşıklığı **O(log(n))** olur.

Karmaşıklık - Algoritmalarda Büyük O Hesabı - Örnek

```
# Her j için, Ortalamalar[j] elemanı, dizi[0], ..., dizi[j]
#  arası elemanların ortalamasını verir.
def kismi_ortalama(dizi):
  n = len(dizi) #
  Ortalamalar = [0] * n # n boyutlu liste oluştur
  for j in range(n): #
    toplam = 0 # S[0] + ... + S[j] arası toplamı
    for i in range(j + 1): #
      toplam += dizi[i] #
    Ortalamalar[j] = toplam / (j+1) #
  return Ortalamalar # O(1)
```

Karmaşıklık - Algoritmalarda Büyük O Hesabı - Örnek

```
# Her j için, Ortalamalar[j] elemanı, dizi[0], ..., dizi[j]
#  arası elemanların ortalamasını verir.
def kismi_ortalama(dizi):
  n = len(dizi) # O(1), Python'da dizi uzunluğu tek işlemle alınır.
  Ortalamalar = [0] * n # n boyutlu liste oluştur, O(n)
  for j in range(n): # n defa yinelenir
    toplam = 0 # S[0] + ... + S[j] arası toplamı, O(n)
    for i in range(j + 1): # ortalama n * (n + 1) / 2 kez yinelenir
      toplam += dizi[i] # O(n(n+1)/2)
    Ortalamalar[j] = toplam / (j+1) # O(n)
  return Ortalamalar # O(1)
```

$O(1) + O(n) + O(n) + O(n(n+1)/2) + O(n) + O(1)$

$O(\max\{1 + n + n + \mathbf{n^2/2} + n/2 + n + 1\})$ en yüksek mertebeli terim dikkate alınır.

$O(n^2)$

Karmaşıklık - Algoritmalarda Büyük O Hesabı - Örnek

Aynı iş, farklı yol:

```
def kismi_ortalama_2(dizi):
  n = len(dizi) # O(1), Python'da dizi uzunluğu tek işlemle alınır.
  Ortalamalar = [0] * n # n boyutlu liste oluştur, O(n)
  toplam = 0 # S[0] + ... + S[j] arası toplamı, O(1)
  for j in range(n): # n defa yinelenir
    toplam += dizi[j] # j'nci değere dek toplam, O(n)
    Ortalamalar[j] = toplam / (j + 1) # ortalamayı hesapla O(n)
  return Ortalamalar # O(1)
```

$O(1) + O(n) + O(1) + O(n) + O(n) + O(1)$

$O(\max\{ 1 + n + 1 + n + n + 1 \})$

$O(n)$

Karmaşıklık - Algoritmalarda Büyük O Hesabı

```
# Her j için, Ortalamalar[j] elemanı, dizi[0], ..., dizi[j]
# arası elemanların ortalamasını verir.
def kismi_ortalama_3(dizi):
  n = len(dizi) # O(1), Python'da dizi uzunluğu tek işlemle alınır.
  A = [0] * n # n boyutlu liste oluştur, O(n)
  for j in range(n): # n defa yinelenir
    # Aşağıdaki ifade j'nci konuma kadar elemanların ortalaması.
    # dizi[0:j+1] ifadesi bir alt döngü gibi işlev görür.
    A[j] = sum(dizi[0:j+1]) / (j+1) # O(n(n+1)/2) kez yürütülür.
  return A # O(1)
```

$O(1) + O(n) + O(n(n+1)/2) + O(n) + O(1)$

$O(1 + n + n^2/2 + n/2 + n + 1)$

$O(\max\{1 + n + \mathbf{n^2/2} + n/2 + n + 1\})$

$O(n^2)$