



UFUK ÜNİVERSİTESİ · BİLGİSAYAR MÜHENDİSLİĞİ · GÜZ DÖNEMİ

MAT201 · 5. Hafta

86 Milyon Kişi Arasında Birini Kaç Adımda Bulursunuz?

Algoritmalar: Arama, Sıralama, Büyük O Gösterimi ve Karmaşıklık

Dr. Alaettin Uçan

Rosen, Bölüm 3.1–3.3



5. HAFTA · BUGÜNÜN AKIŞI



Bugün beş bölümde ilerleyeceğiz

1

Algoritma nedir?

Çay demlemekten not ortalamasına: bir tarifi algoritma yapan nedir?

2

Arama

86 milyon kayıt içinde bir kişi, uzun bir metin içinde bir sözcük

3

Sıralama ve ağgözlü algoritmalar

2,5 milyon YKS adayını sıraya dizmek, para üstünü en az bozuk parayla vermek

4

Büyük O gösterimi

Algoritmaları saniyeyle değil, büyüme hızıyla karşılaştırmak

5

Karmaşıklık ve sınırlar

43 gün süren bir kontrol, 81 ilin turu ve hiç yazılamayacak bir program

Sorularınızı istediğiniz an Teams sohbetine yazabilirsiniz.

AÇILIŞ



TÜİK'e göre Türkiye'nin nüfusu 31 Aralık 2025 itibarıyla 86 092 168.

Herkesin kaydının, kimlik numarasına göre sıralanmış tek bir listede durduğunu düşünelim.

Kaynak: TÜİK, Adrese Dayalı Nüfus Kayıt Sistemi 2025 sonuçları

3

AÇILIŞ



SORU

Bu listede aradığınız kişiyi en fazla kaç adımda bulursunuz?

Her adımda listedeki tek bir kaydın numarasına bakabiliyorsunuz. Tahmininizi sohbete yazın: bin mi, bir milyon mu, 86 milyon mu?

4



1. BÖLÜM

Algoritma Nedir?

Bir işi, makinenin anlayacağı kesinlikte tarif etmek



1. BÖLÜM · ALGORİTMA NEDİR?



... iyi tanımlanmış kuralların ve işlemlerin adım adım uygulanmasıyla bir sorunun giderilmesi veya sonuca en hızlı biçimde ulaşılması işlemi

TDK sözlüğünde “algoritma” sözcüğünün tanımı.



Algoritma sözcüğü, 9. yüzyılda Bağdat'ta yaşamış matematikçi Harezmi'nin adından gelir.

Harezmi'nin Hint rakamlarıyla hesap yapmayı anlatan kitabı Latinceye çevrilirken yazarın adı "Algoritmi" diye yazıldı. TDK maddesindeki "Harezmi yolu" ifadesi buradan geliyor.

Kaynak: TDK Güncel Türkçe Sözlük, "algoritma" maddesi

7



Algoritma, bir problemin çözümü için izlenen adım adım işlemlerin tanımıdır.

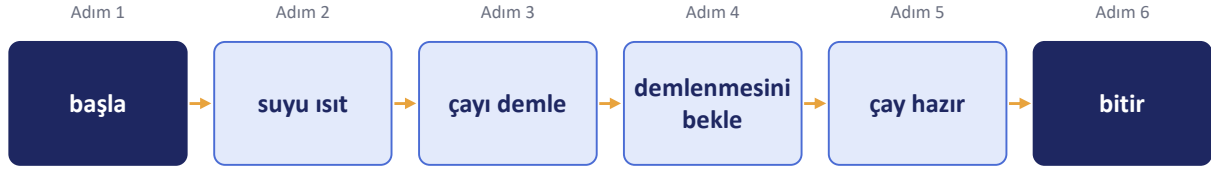
Lisedeki Bilgisayar Bilimi dersinde verilen tanım.

Kaynak: Ataköy Lisesi, 9. sınıf Bilgisayar Bilimi çalışma kâğıdı

8



Lisedeki örnek: çay demleme algoritması



Girdi: su ve kuru çay. Çıktı: demlenmiş çay. Aradaki dört adım işi yapıyor.



SORU

Bu tarifi bir makineye versek çay demleyebilir mi?

“Suyu ısıt”: kaç dereceye kadar? “Çayı demle”: kaç kaşık çay, ne kadar su? “Demlenmesini bekle”: kaç dakika?



Rosen'ın ilk algoritma örneği: sonlu bir tam sayı dizisindeki en büyük sayıyı bulmak.

Girdi bir sayı dizisi, çıktı tek bir sayı. Basit görünen bu problem, bir algoritmada bulunması gereken bütün özellikleri gösteriyor.

Kaynak: Rosen, Bölüm 3.1, Örnek 1

11



Rosen'ın sözde koduyla

```
Rosen, Algoritma 1
procedure max( $a_1, a_2, \dots, a_n$ : integers)
  max :=  $a_1$ 
  for i := 2 to n
    if max <  $a_i$  then max :=  $a_i$ 
  return max    {max is the largest element}
```

Sözde kod

Hiçbir programlama diline bağlı olmadan adımları açıkça yazan gösterim. Rosen bütün algoritmaları böyle yazar.

:=

Atama: sağdaki değer soldaki değişkene yazılır. Python'daki = işaretinin karşılığıdır.

Fikir

O ana kadar görülen en büyük sayıyı aklında tut; daha büyüğü gelirse onunla değiştir.

Kaynak: Rosen, Bölüm 3.1, Algoritma 1

12



Aynı algoritma Türkiye verisiyle: 81 il arasından nüfusu en kalabalık olanı bulmak.

TÜİK her ilin nüfusunu ayrı ayrı yayımlıyor. Elimizdeki liste plaka sırasında olsun; nüfusa göre sıralı değil.

Kaynak: TÜİK, Adrese Dayalı Nüfus Kayıt Sistemi 2025 sonuçları

13



En büyüğü bulmak

```
max.py
def largest(populations):
    best = populations[0]
    for p in populations[1:]:
        if p > best:           # one comparison
            best = p          # new record
    return best
```

Fikir

Şimdiye kadar gördüğün en büyük değeri aklında tut; daha büyüğü gelirse onu tut.

Karşılaştırma sayısı

İlk il dışındaki her il bir kez karşılaştırılır: 81 il için 80 karşılaştırma.

n eleman için $n - 1$ karşılaştırma.

Kaynak: Rosen, Bölüm 3.1

14



SORU

En büyüğü bulmak için 80'den az karşılaştırma yetebilir mi?

Yöntem fark etmez: herhangi bir algoritmayı düşünebilirsiniz. Cevabınızı ve gerekçenizi sohbete yazın.

15



CEVAP

Hayır. Her karşılaştırma en fazla bir ili “en kalabalık değil” diye eler.

En kalabalık il dışındaki 80 ilin her birinin en az bir kez kaybetmesi gerekir. Bu yüzden 80 karşılaştırmanın altına inilemez.

16



Rosen, en büyüğü bulma algoritmasının bir algoritmada aranan yedi özelliğin hepsini taşıdığını gösteriyor.

Bu yedi özelliğe tek tek bakalım ve her birinde lisedeki çay tarifini de aynı ölçüyle sınavalım.



1. Girdi

En büyüğü bulma algoritması

Girdi, $a_1, a_2, \dots, a_n$ tam sayı dizisidir.

Sağlanıyor

Çay tarifi

Girdi, su ve kuru çaydır.

Sağlanıyor

Algoritmanın işe başlarken aldığı değerler.



2. Çıktı

En büyüğü bulma algoritması

Çıktı, dizinin en büyük elemanıdır.

Sağlanıyor

Çay tarifi

Çıktı, demlenmiş çaydır.

Sağlanıyor

Algoritmanın sonunda ürettiği sonuç.



3. Belirlilik

En büyüğü bulma algoritması

Adımlar yalnızca atama ve karşılaştırmadan oluşur; her adımda ne yapılacağı bellidir.

Sağlanıyor

Çay tarifi

“Suyu ısıt” adımı kaç dereceye kadar ısıtılacağını söylemiyor.

Sağlanmıyor

Her adım kesin ve tek anlamlı olarak tanımlanmış olmalı.



4. Doğruluk

En büyüğü bulma algoritması

Her adımdan sonra max, o ana kadar görülen en büyük sayıdır; dizi bitince de dizinin en büyüğüdür.

Sağlanıyor

Çay tarifi

Ne kadar çay, ne kadar su konacağı belli olmadığı için iyi bir çay çıkacağı garanti değil.

Sağlanmıyor

Algoritma her geçerli girdi için doğru sonucu üretmeli.



5. Sonluluk

En büyüğü bulma algoritması

Döngü $n - 1$ kez döner ve algoritma durur.

Sağlanıyor

Çay tarifi

“Demlenmesini bekle” adımının ne zaman biteceği belli değil.

Sağlanmıyor

Algoritma her girdi için sonlu sayıda adımdan sonra durmalı.



6. Etkinlik

En büyüğü bulma algoritması

Bir atama ya da bir karşılaştırma sonlu sürede yapılır.

Sağlanıyor

Çay tarifi

Suyu ısıtmak, çayı demlemek ve beklemek gerçekten yapılabilen işlerdir.

Sağlanıyor

Her adım sonlu sürede ve gerçekten yapılabilmeli.



7. Genellik

En büyüğü bulma algoritması

Uzunluğu ne olursa olsun her sonlu tam sayı dizisinde çalışır.

Sağlanıyor

Çay tarifi

Her demlikte ve her çay için uygulanabilir.

Sağlanıyor

Algoritma tek bir girdi için değil, aynı türdeki bütün girdiler için çalışmalı.



Çay tarifi belirlilik, doğruluk ve sonlulukta takılıyor; Rosen'ın algoritması ise yedi özelliğin hepsini sağlıyor.

Bir tarifi algoritma yapan, onu izleyen kişinin hiçbir şeyi tahmin etmek zorunda kalmamasıdır.

Kaynak: Rosen, Bölüm 3.1, Örnek 2

25



Yönetmeliğimizde de bir algoritma var: yarıyıl not ortalamasının hesaplanması.

26



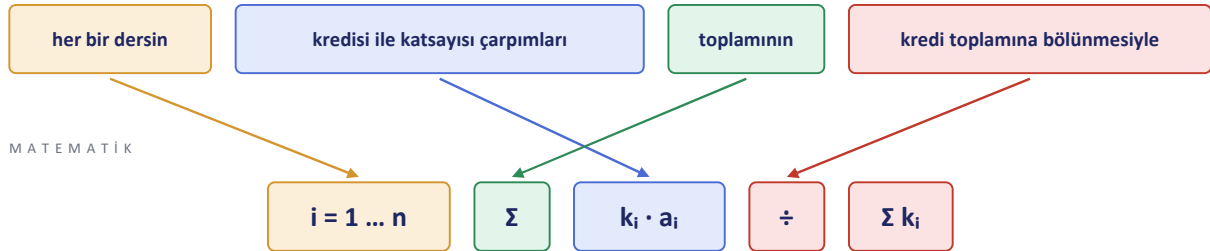
... o yarıyıl da aldığı her bir dersin kredisi ile başarı puan kat sayısı çarpımları toplamının o yarıyıl aldığı kredi toplamına bölünmesiyle bulunur.

Yarıyıl not ortalamasının (YNO) tanımı.



Türkçeden matematiğe

TÜRKÇE



MATEMATİK

$$YNO = (k_1 a_1 + k_2 a_2 + \dots + k_n a_n) \div (k_1 + k_2 + \dots + k_n)$$

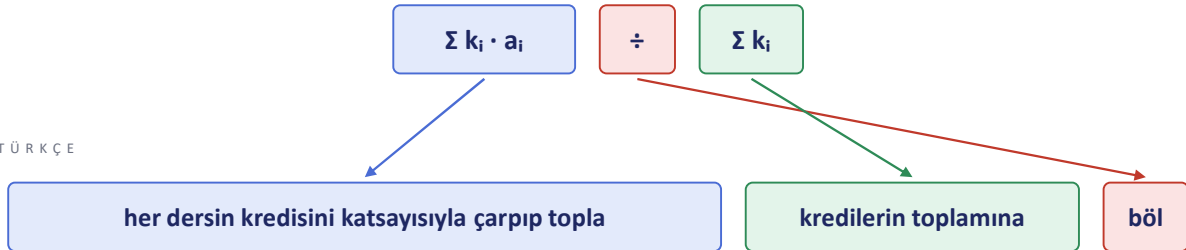
k_i : i. dersin kredisi. a_i : i. dersin başarı katsayısı. n : o yarıyıl alınan ders sayısı.



Formülü geri okuyalım

MATEMATİK

TÜRKÇE



“Her dersin kredisini katsayısıyla çarpıp topla, kredilerin toplamına böl.”

29



Aynı kural Python'da

```
gpa.py
def semester_gpa(courses):
    weighted, credits = 0, 0
    for credit, coef in courses:
        weighted += credit * coef
        credits += credit
    return weighted / credits
```

Girdi

Derslerin (kredi, katsayı) ikilileri.

Çıktı

Yarıyıl not ortalaması.

Döngü

Her ders için bir kez döner: n ders, n tur.

30



Bir algoritmanın hızını saniyeyle değil, yaptığı işlem sayısı ile ölçeriz.

Süre, kullandığınız bilgisayara göre değişir; işlem sayısı ise yalnızca girdinin büyüklüğüne bağlıdır.

31



YNO algoritması kaç işlem yapar?

Her ders için: 1 çarpma + 2 toplama

Döngünün bir turu



n ders için: $3n$ işlem

Döngü n kez döner



Sonda 1 bölme: toplam $3n + 1$ işlem

return satırı

Ders sayısı iki katına çıkarsa işlem sayısı da yaklaşık iki katına çıkar.

32



2. BÖLÜM

Arama

86 milyon kayıt içinde bir kişi, uzun bir metin içinde bir sözcük



2. BÖLÜM · ARAMA



Önce kayıtların karışık sırada olduğunu düşünelim.

Baştan başlar, her kaydı tek tek aradığımız numarayla karşılaştırırız. Bu yönteme doğrusal arama denir.



Doğrusal arama

```
search.py
def linear_search(ids, x):
    for i, value in enumerate(ids):
        if value == x:          # one comparison
            return i
    return -1                  # not found
```

Ne zaman biter?

Aranan bulununca ya da liste bitince.

En kötü durum

Aranan en sonda ya da hiç yok: her kayda bakılır.



Rosen'ın sözde koduyla doğrusal arama

```
Rosen, Algoritma 2
procedure linear search(x: integer,
    a1, a2, ..., an: distinct integers)
i := 1
while (i ≤ n and x ≠ ai)
    i := i + 1
if i ≤ n then location := i
else location := 0
return location    {0 if x is not found}
```

Rosen'da

Dizi a_i 'den başlar. x bulunamazsa sonuç 0'dır.

Python'da

Liste 0. indeksten başladığı için biz bulunamayan değer için -1 döndürdük.



Aradığımız kişi listenin sonundaysa ya da listede hiç yoksa 86 092 168 karşılaştırma yaparız.

En kötü durumda karşılaştırma sayısı kayıt sayısına eşittir: $f(n) = n$.



Türkçeden matematiğe

TÜRKÇE

En kötü durumda

her kayda

bir kez bakılır

MATEMATİK

 $f(n)$

=

 n $\cdot 1$

Doğrusal aramanın en kötü durumdaki karşılaştırma sayısı: $f(n) = n$



Kayıtlar sıralıysa çok daha hızlı bir yol vardır.

Basılı bir sözlükte kelime ararken ilk sayfadan başlamayız. Ortalardan bir sayfa açar, aradığımız kelimenin önde mi arkada mı kaldığına bakarız.



İkili arama: ortadaki kayda bak. Aradığın numara küçükse sol yarıda, büyükse sağ yarıda aramaya devam et.

Her karşılaştırma, geriye kalan kayıtların yarısını aramanın dışına atar.



Küçük bir örnek: sıralı plaka kodları arasında 35'i arayalım

	Adana	Ankara	Antalya	Bursa	İstanbul	İzmir	Konya	
1. adım	01	06	07	16	34	35	42	Ortada 16 var. $35 > 16$: sağ yarıya geç.
2. adım	01	06	07	16	34	35	42	Kalan üçün ortası 35. Bulundu.

Yedi kayıt, iki adım. Doğrusal arama aynı işi altı adımda yapardı.

Kaynak: Rosen, Bölüm 3.1

41



İkili arama

```
search.py
def binary_search(ids, x):
    lo, hi = 0, len(ids) - 1
    while lo <= hi:
        mid = (lo + hi) // 2
        if ids[mid] == x:
            return mid
        elif ids[mid] < x:
            lo = mid + 1 # drop left half
        else:
            hi = mid - 1 # drop right half
    return -1
```

lo, hi

Hâlâ aranan aralığın iki ucu.

mid

Aralığın ortası; // tam sayı bölmesi (geçen haftanın taban fonksiyonu).

Döngü koşulu

Aralık boşalınca arama biter: kayıt yok.

Kaynak: Rosen, Bölüm 3.1

42



Rosen'ın örneği: 16 terimli sıralı bir listede 19'u arayalım.

Liste: 1, 2, 3, 5, 6, 7, 8, 10, 12, 13, 15, 16, 18, 19, 20, 22. Her adımda listeyi ikiye böler ve 19'un hangi yarıda kaldığına bakarız.



Rosen'ın örneği adım adım: 19 nerede?

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
19 > 10: sağ yarı	1	2	3	5	6	7	8	10	12	13	15	16	18	19	20	22
19 > 16: sağ yarı	1	2	3	5	6	7	8	10	12	13	15	16	18	19	20	22
19 > 19 değil: sol yarı	1	2	3	5	6	7	8	10	12	13	15	16	18	19	20	22
19 > 18: sağ yarı	1	2	3	5	6	7	8	10	12	13	15	16	18	19	20	22
19 = 19: bulundu	1	2	3	5	6	7	8	10	12	13	15	16	18	19	20	22

Turuncu kutu, x'in karşılaştırıldığı orta terim. 19, listenin 14. terimi.



Rosen'ın sözde koduyla ikili arama

Rosen, Algoritma 3

```
procedure binary search(x: integer,
    a1, a2, ..., an: increasing integers)
i := 1    {left endpoint}
j := n    {right endpoint}
while i < j
    m := [(i + j)/2]
    if x > am then i := m + 1
    else j := m
if x = ai then location := i
else location := 0
return location
```

||

Geçen haftanın taban fonksiyonu: ortadaki terimin yeri.

Python sürümünden farkı

Rosen döngüde yalnızca $x > a_m$ sorusunu sorar; eşitliğe en sonda bir kez bakar.

Kaynak: Rosen, Bölüm 3.1, Algoritma 3

45



Türkçeden matematiğe

TÜRKÇE

Her adımda

aranacak kayıt sayısı

yarıya iner

MATEMATİK

k adım sonra kalan:

n

$/ 2^k$

k adım sonra en fazla $n / 2^k$ kayıt kalır.

Kaynak: Rosen, Bölüm 3.3

46



86 092 168 kayıta ikili arama

Adım	Aranan aralıktaki kayıt sayısı
1	86 092 168
2	43 046 084
3	21 523 042
10	168 148
20	164
25	5
26	2
27	1

Kaynak: TÜİK, Adrese Dayalı Nüfus Kayıt Sistemi 2025 sonuçları

47



Kaç adım gerekir?

$$n / 2^k = 1$$

Tek kayıt kalana kadar



$$2^k = n$$

İki tarafı 2^k ile çarpalım



$$k = \log_2 n$$

Logaritmanın tanımı



$$\log_2 86\,092\,168 \approx 26,4 \rightarrow 27 \text{ adım}$$

Yukarı yuvarlayalım: $\lceil 26,4 \rceil = 27$

86 milyon kayıt arasında en fazla 27 adım. Açılıştaki sorunun cevabı bu.

Kaynak: Rosen, Bölüm 3.3

48



SORU

Nüfus 1000 katına çıksaydı ikili arama kaç adım uzardı?

İpucu: $2^{10} = 1024$.



CEVAP

Yalnızca 10 adım: 27 yerine 37.

Girdi 1000 katına çıktı, adım sayısı 10 arttı. Doğrusal aramada ise karşılaştırma sayısı da 1000 katına çıkardı. Logaritmik büyümenin gücü budur.



İkili aramanın bir bedeli var: kayıtların önceden sıralanmış olması gerekir.

Sıralama bir kez yapılır, arama ise milyonlarca kez. Veritabanlarının kayıtları dizinlerde sıralı tutmasının nedeni bu.



Python'da ikili arama hazır geliyor

```
search.py
import bisect

ids.sort()
i = bisect.bisect_left(ids, x)
found = i < len(ids) and ids[i] == x
```

once
every search

bisect modülü, sıralı bir listede ikili aramayı yapar.



Bir metinde sözcük aramak da bir arama problemidir.

Tarayıcıda Ctrl+F'ye bastığınızda ya da yönetmelikte “devamsızlık” sözcüğünü aradığınızda çalışan algoritma budur.



Geçen haftaki NVİ cümlesinde “ver” sözcüğünü arayalım

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	
Metin	b	i	r	_	d	e	f	a	_	v	e	r	i	l	i	r	
s = 0	b	i	r	_	d	e	f	a	_	v	e	r	i	l	i	r	$b \neq v$
s = 5	b	i	r	_	d	e	f	a	_	v	e	r	i	l	i	r	$e \neq v$
s = 9	b	i	r	_	d	e	f	a	_	v	e	r	i	l	i	r	Bulundu

Sözcüğü her konuma (kaydırmaya) tek tek yerleştirip harf harf karşılaştırıyoruz.



Saf metin arama

```
match.py
def find_all(text, word):
    n, m = len(text), len(word)
    hits = []
    for s in range(n - m + 1):      # every shift
        if text[s:s + m] == word:  # up to m checks
            hits.append(s)
    return hits
```

Kaydırma sayısı

 $n - m + 1$

Her kaydırmada

En fazla m harf karşılaştırması.

En kötü durum

$(n - m + 1) \cdot m$ karşılaştırma: $O(nm)$.



SORU

Hangi metin ve sözcükte saf metin arama en çok zorlanır?

Her kaydırmada m harfin hepsinin karşılaştırılması gereken bir örnek bulun.



CEVAP

“aaaa...aaa” metninde “aab” sözcüğünü aramak.

Her kaydırmada ilk iki harf tutar, ancak üçüncü harfte ayrılır. Her kaydırma m karşılaştırmaya mal olur ve sözcük hiç bulunmaz.



Sıralama ve Açıgözlü Algoritmalar

2,5 milyon adayı sıraya dizmek, para üstünü en az bozuk parayla vermek





ÖSYM'ye göre 2025 YKS'ye 2 560 649 aday başvurdu.

Her adayın puanına göre bir sıralaması var. O sıralamayı üreten de bir algoritma.

Kaynak: ÖSYM, "YKS'ye 2,5 milyondan fazla aday katılacak" duyurusu (16.06.2025)

59



Kabarcık sıralaması: yan yana duran iki elemanı karşılaştır, sıraları yanlışsa yerlerini değiştir.

Listenin sonuna kadar böyle ilerle ve bunu turlar halinde tekrarla. Her turda kalanların en büyüğü, suyun yüzüne çıkan bir kabarcık gibi sona yerleşir.

Kaynak: Rosen, Bölüm 3.1

60



Rosen'ın örneği: 3, 2, 4, 1, 5 kabarcık sıralamasıyla

Başlangıç	3	2	4	1	5	
1. tur	2	3	1	4	5	3 ile 2, sonra 4 ile 1 yer değiştirdi
2. tur	2	1	3	4	5	3 ile 1 yer değiştirdi
3. tur	1	2	3	4	5	2 ile 1 yer değiştirdi
4. tur	1	2	3	4	5	Yer değiştirme yok; liste sıralı

Her tur, kalanların en büyüğünü sona yerleştirir. Lacivert kutular yerine oturmuş elemanlar.

Kaynak: Rosen, Bölüm 3.1, Örnek 4

61



Aynı yöntem, beş ilin plaka kodlarıyla

Başlangıç	34	06	35	01	16	İstanbul, Ankara, İzmir, Adana, Bursa
1. tur	06	34	01	16	35	4 karşılaştırma; 35 sona yerleşti
2. tur	06	01	16	34	35	3 karşılaştırma; 34 yerine oturdu
3. tur	01	06	16	34	35	2 karşılaştırma
4. tur	01	06	16	34	35	1 karşılaştırma; liste sıralı

Toplam $4 + 3 + 2 + 1 = 10$ karşılaştırma. Lacivert kutular yerine oturmuş elemanlar.

Kaynak: Rosen, Bölüm 3.1

62



Kabarcık sıralaması Python'da

```
sort.py
def bubble_sort(a):
    n = len(a)
    for i in range(n - 1):          # passes
        for j in range(n - 1 - i): # unsorted part
            if a[j] > a[j + 1]:
                a[j], a[j + 1] = a[j + 1], a[j]
    return a
```

Dış döngü

$n - 1$ tur.

İç döngü

Her turda bir eleman yerine oturduğu için bir kısılır.

Takas

Python'da iki değeri tek satırda yer değiştirmek.



Kabarcık sıralaması kaç karşılaştırma yapar?

$$(n - 1) + (n - 2) + \dots + 2 + 1$$

Turların karşılaştırma sayıları



$$n(n - 1) / 2$$

Ardışık sayıların toplamı



$$n = 5 \text{ için } 5 \cdot 4 / 2 = 10$$

Plaka örneğindeki sayı

Karşılaştırma sayısı n^2 ile büyür. Az sonra 2,5 milyon aday için bunun ne demek olduğunu göreceğiz.



Araya yerleştirme sıralaması: elemanları sırayla al, her birini önündeki sıralı kısımda doğru yere yerleştir.

Sınav kâğıtlarını öğrenci numarasına göre dizerken yaptığınız şey budur: yeni gelen kâğıdı destede doğru yere sokarsınız.

Kaynak: Rosen, Bölüm 3.1

65



Rosen'ın örneği: aynı listeyi araya yerleştirmeyeyle dizelim

Başlangıç

3	2	4	1	5
---	---	---	---	---

İlk eleman tek başına sıralı

2'yi yerleştir

2	3	4	1	5
---	---	---	---	---

$2 < 3$: başa geçti

4'ü yerleştir

2	3	4	1	5
---	---	---	---	---

$4 > 2$ ve $4 > 3$: yerinde kaldı

1'i yerleştir

1	2	3	4	5
---	---	---	---	---

$1 < 2$: başa geçti

5'i yerleştir

1	2	3	4	5
---	---	---	---	---

1, 2, 3 ve 4'ten büyük: sonda kaldı

Rosen yeni elemanın yerini sıralı kısmın başından başlayarak arar. Turuncu kutu yeni yerleşen eleman.

Kaynak: Rosen, Bölüm 3.1, Örnek 5

66



Aynı yöntem, beş plaka koduyla

Başlangıç	34	06	35	01	16	İlk eleman tek başına sıralı
06'yı yerleştir	06	34	35	01	16	34'ün önüne geçti
35'i yerleştir	06	34	35	01	16	Yerinde kaldı
01'i yerleştir	01	06	34	35	16	En başa kadar gitti
16'yı yerleştir	01	06	16	34	35	06 ile 34 arasına girdi

Lacivert kısım her adımda sıralı. Turuncu kutu yeni yerleşen eleman.

Kaynak: Rosen, Bölüm 3.1

67



Araya yerleştirme Python'da

```
sort.py
def insertion_sort(a):
    for i in range(1, len(a)):
        x = a[i]
        j = i - 1
        while j >= 0 and a[j] > x:
            a[j + 1] = a[j]
            j -= 1
        a[j + 1] = x
    return a
```

En kötü durum

Liste ters sıralıysa her eleman en başa kadar gider: $n(n - 1)/2$ karşılaştırma.

En iyi durum

Liste zaten sıralıysa her eleman tek karşılaştırmayla yerinde kalır: $n - 1$.

Kaynak: Rosen, Bölüm 3.1

68



SORU

Kabarcık sıralaması 2 560 649 adayı kaç dakikada sıralar?

Saniyede bir milyar karşılaştırma yapan bir bilgisayar düşünün. Tahmininizi sohbete yazın.

69



YKS adaylarını sıralamak: $n = 2\,560\,649$

Yöntem	Karşılaştırma sayısı	Saniyede 10^9 işlemle
Kabarcık sıralaması	$n(n-1)/2 \approx 3,3 \cdot 10^{12}$	yaklaşık 55 dakika
$n \log_2 n$ 'lik bir sıralama	$\approx 5,5 \cdot 10^7$	yaklaşık 0,05 saniye

Python'daki sorted() de $n \log n$ zamanda çalışan bir yöntem kullanır.

Kaynak: ÖSYM, "YKS'ye 2,5 milyondan fazla aday katılacak" duyurusu (16.06.2025)

70



Market kasasında para üstü verirken farkında olmadan bir algoritma çalıştırırsınız.

Önce verebileceğiniz en büyük parayı verirsiniz, kalan tutar için aynısını tekrarlırsınız. Her adımda o an en iyi görüneni seçen bu yaklaşıma açgözlü (greedy) algoritma denir.



Tedavüldeki madeni paralar



Para üstünü kuruş cinsinden düşünelim: 100, 50, 25, 10, 5 ve 1.



Para üstü 1,85 TL: açğözlü yöntem adım adım

185 kuruş → 1 TL ver

Kalan 85 kuruş



85 kuruş → 50 kuruş ver

Kalan 35 kuruş



35 kuruş → 25 kuruş ver

Kalan 10 kuruş



10 kuruş → 10 kuruş ver

Kalan 0

4 madeni para. Bu tutar daha az parayla verilemez.



Açğözlü para üstü

```
change.py
def make_change(amount, coins=(100, 50, 25, 10, 5, 1)):
    given = []
    for c in coins:
        # largest coin first
        while amount >= c:
            amount -= c
            given.append(c)
    return given

make_change(185)
# [100, 50, 25, 10]
```

Her adımda kalan tutarı aşmayan en büyük parayı seçiyor.



Rosen'ın örneği: 67 sent para üstü, 25, 10, 5 ve 1 sentliklerle

67 sent → 25 sentlik ver

Kalan 42 sent



42 sent → 25 sentlik ver

Kalan 17 sent



17 sent → 10 sentlik ver

Kalan 7 sent



7 sent → 5 sentlik ver

Kalan 2 sent



2 sent → iki tane 1 sentlik ver

Kalan 0

Toplam 6 bozuk para: 25 + 25 + 10 + 5 + 1 + 1.

Kaynak: Rosen, Bölüm 3.1, Örnek 6

75



SORU

Açgözlü yöntem her zaman en az sayıda madeni parayı mı verir?

Tahmininizi sohbete yazın. Bir karşı örnek bulabilir misiniz?

76



CEVAP

Bugünkü madeni paralarla evet.

1 ile 1000 kuruş arasındaki her tutarı bilgisayarla tek tek denemek, açgözlü yöntemin her seferinde en az parayı verdiğini gösteriyor.



Rosen, açgözlü yöntemin 25, 10, 5 ve 1 sentliklerle her zaman en az sayıda bozuk parayı verdiğini ispatlıyor.

İspatın anahtarı şu gözlem: en az parayla verilen bir para üstünde en fazla iki 10'luk, en fazla bir 5'lik ve en fazla dört 1'lik bulunur; iki 5'lik ile bir 10'luk de aynı anda bulunamaz.



Ama kasada 5 kuruşluk kalmamışsa? Para üstü 30 kuruş

Açğözlü yöntem

Önce en büyüğü: 25 kuruş
Kalan 5 kuruş: 10'luk sığmaz
1 + 1 + 1 + 1 + 1 kuruş

6 madeni para

En iyi çözüm

25'liğı hiç kullanma
10 + 10 + 10 kuruş

3 madeni para

Rosen da aynı örneğı veriyor: 5 sentlik olmazsa 30 sent, açğözlü yöntemle 6 bozuk paraya çıkar.



Açğözlü algoritma hızlıdır ama her zaman en iyi sonucu garanti etmez.

Bir para sisteminde en iyi sonucu verdiği ayrıca ispatlanmalıdır. 5'lik paranın yokluğu, ABD paraları için yapılan ispatı bile geçersiz kılıyor.



Rosen'ın ikinci açgözlü örneği: bir salona olabildiğince çok konuşma yerleştirmek.

Her konuşmanın başlangıç ve bitiş saati belli. Aynı anda iki konuşma yapılamaz; ama biri biterken öteki başlayabilir.

Kaynak: Rosen, Bölüm 3.1, Örnek 7

81



SORU

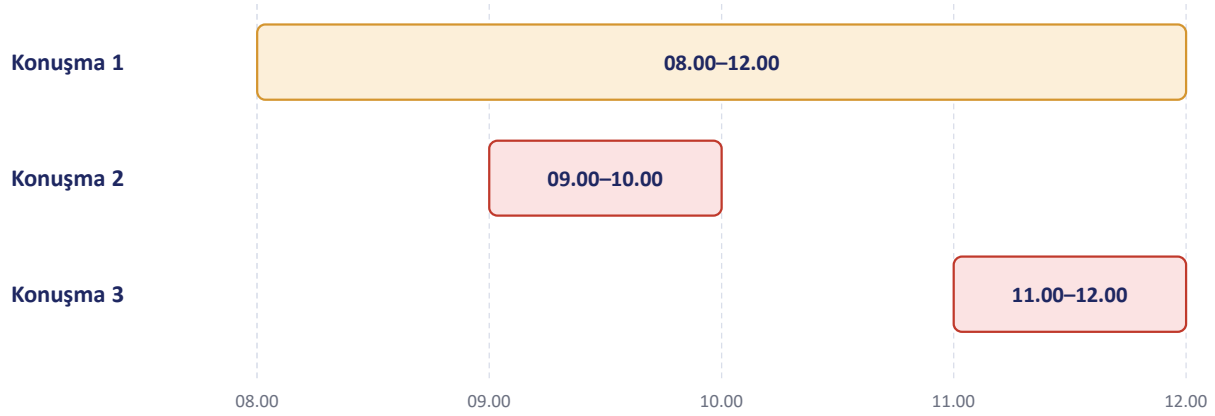
Her adımda hangi konuşmayı seçmeliyiz?

En erken başlayanı mı, en kısa süreni mi, yoksa en erken biteni mi? Tahmininizi sohbete yazın.

82



Ölçüt 1: en erken başlayanı seç



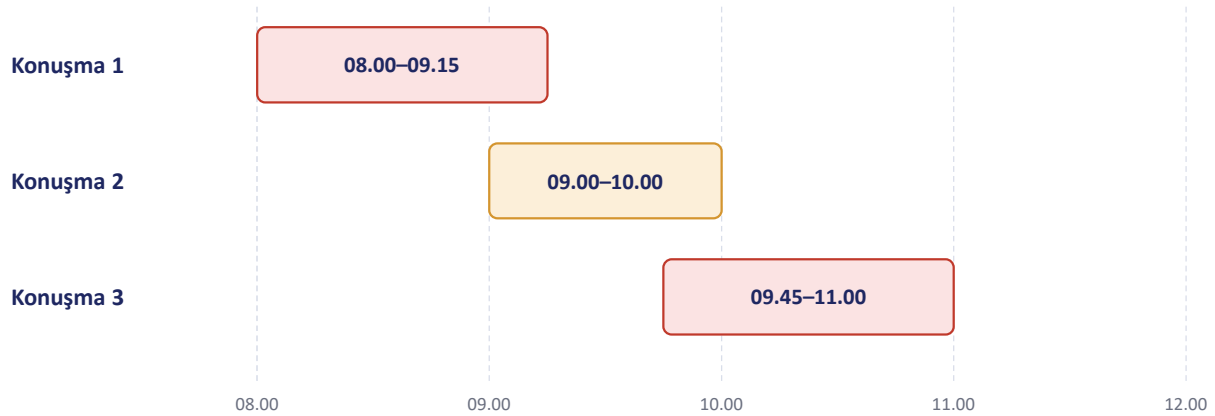
Önce Konuşma 1 seçilir; öteki ikisi onunla çakıştığı için salona tek konuşma sığar. Oysa 2 ile 3 birlikte yapılabilirdi.

Kaynak: Rosen, Bölüm 3.1, Örnek 7

83



Ölçüt 2: en kısa süreni seç



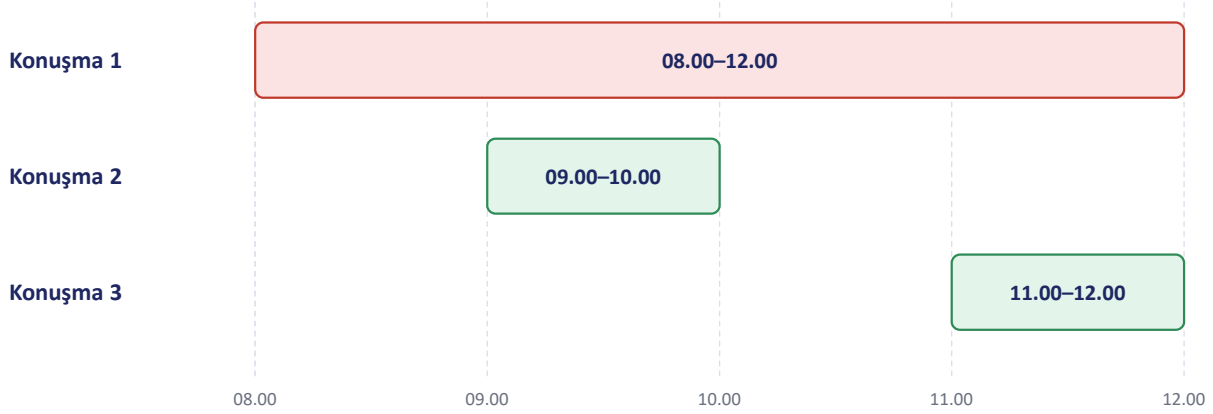
En kısa olan Konuşma 2 seçilir; öteki ikisi onunla çakışır ve yine tek konuşma kalır. Oysa 1 ile 3 çakışmıyor.

Kaynak: Rosen, Bölüm 3.1, Örnek 7

84



Ölçüt 3: en erken biteni seç



İlk örneğe dönelim: önce 10.00'da biten Konuşma 2, sonra onunla çakışmayan Konuşma 3 seçilir. İki konuşma, yani en iyisi.

Kaynak: Rosen, Bölüm 3.1, Örnek 7

85



En erken biteni seçen açgözlü algoritma her zaman en çok konuşmayı yerleştirir.

Rosen bunu 5. bölümde tümevarımla ispatlıyor. Açgözlü bir algorithmada asıl iş, her adımda neye göre seçim yapılacağını doğru belirlemektir.

Kaynak: Rosen, Bölüm 3.1, Örnek 7

86



4. BÖLÜM

Büyük O Gösterimi

Ayrıntıları bir kenara bırakıp büyüme hızına bakmak



4. BÖLÜM · BÜYÜK O



$3n + 1$ işlem mi, $3n + 5$ işlem mi? n büyüdükçe bu ayrıntının önemi kalmaz.

Bizi asıl ilgilendiren, işlem sayısının n ile birlikte nasıl büyüdüğü: doğrusal mı, karesel mi, logaritmik mi?



Büyük O gösterimi, bir fonksiyonun büyüme hızına üst sınır koyar.

“ $f(x)$, $O(g(x))$ 'tir” demek şu demektir: bir noktadan sonra f , g 'nin sabit bir katını hiç aşmaz.



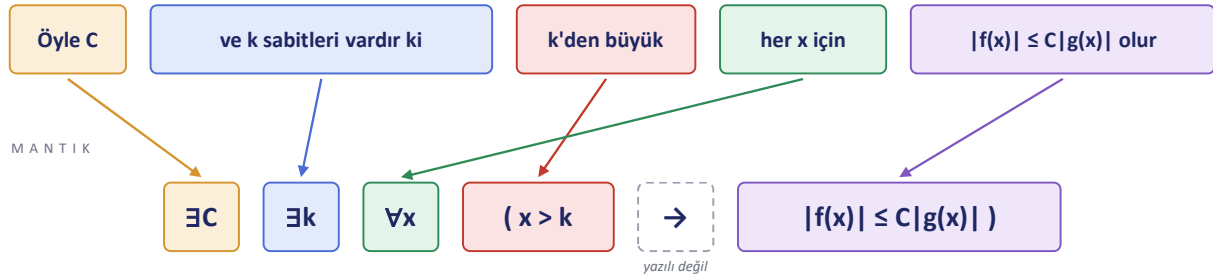
Tanım: Öyle C ve k sabitleri vardır ki, k 'den büyük her x için $|f(x)| \leq C \cdot |g(x)|$ olur.

Bu durumda $f(x) = O(g(x))$ yazar, “ f , büyük O g 'dir” diye okuruz.



Türkçeden mantığa

TÜRKÇE



$$\exists C \exists k \forall x (x > k \rightarrow |f(x)| \leq C|g(x)|)$$

3. haftanın kuralı: “her” ile kurulan cümlede koşul $\rightarrow$ ile bağlanır. Burada $\rightarrow$ açıkça yazılmamış.

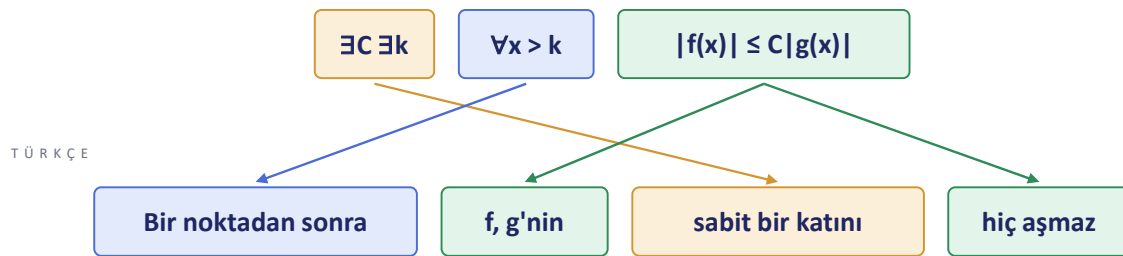
Kaynak: Rosen, Bölüm 3.2

91



Formülü geri okuyalım

MANTIK



“Bir noktadan sonra f, g 'nin sabit bir katını hiç aşmaz.”

92



Niceleyicilerin sırası önemli: önce C ve k seçilir, sonra her x için eşitsizlik sağlanmalı.

3. haftadaki “her derde deva bir ilaç” örneğini hatırlayın: $\exists \forall$. C ve k, x'e göre değişmez.



Rosen'ın ilk örneği: $x^2 + 2x + 1 = O(x^2)$

$$x^2 + 2x + 1$$

$x > 1$ olsun



$$\leq x^2 + 2x^2 + x^2$$

$x > 1$ için $x < x^2$ ve $1 < x^2$



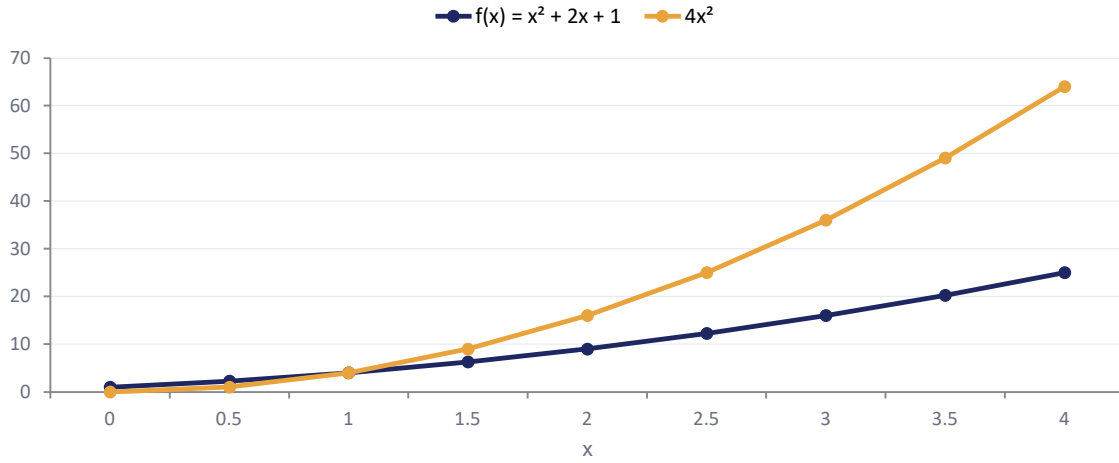
$$= 4x^2$$

Toplayalım

C = 4 ve k = 1 tanıklardır: $x^2 + 2x + 1 = O(x^2)$.



Grafikte görelim: $x = 1$ 'den sonra f hep $4x^2$ 'nin altında



$x = 1$ 'de ikisi de 4. Bu noktadan sonra $4x^2$ daha hızlı artıyor; $k = 1$ buradan geliyor.

Kaynak: Rosen, Bölüm 3.2, Örnek 1

95



C ve k sayılarına tanık denir. Tanıklar tek değildir.

Rosen aynı örnekte ikinci bir çift de veriyor: $x > 2$ için $2x \leq x^2$ ve $1 \leq x^2$ olduğundan $x^2 + 2x + 1 \leq 3x^2$. Yani $C = 3$ ve $k = 2$ de iş görür.

Kaynak: Rosen, Bölüm 3.2, Örnek 1

96



Ters yönde de doğru: $x^2 = O(x^2 + 2x + 1)$.

$x > 1$ için $x^2 < x^2 + 2x + 1$; tanıklar $C = 1$ ve $k = 1$. İki fonksiyon birbirinin büyük O'su olduğunda aynı mertebeden büyür. Bunun adını birazdan koyacağız: Θ .



Rosen'ın ikinci örneği: $7x^2 = O(x^3)$

$x > 7$ olsun

$k = 7$



$7x^2 < x \cdot x^2 = x^3$

$7 < x$ olduğundan



$7x^2 < 1 \cdot x^3$

$C = 1$

$C = 1$ ve $k = 7$ tanıklardır. Başka bir çift: $x > 1$ için $7x^2 < 7x^3$, yani $C = 7$ ve $k = 1$.



YNO algoritması: $3n + 1 = O(n)$

$$3n + 1$$

$n > 1$ olsun



$$\leq 3n + n$$

$n > 1$ için $1 \leq n$



$$= 4n$$

Toplayalım

Tanımlar $C = 4$ ve $k = 1$. Not ortalaması algoritması $O(n)$ zamanda çalışır.



SORU

Peki bir fonksiyonun $O(g)$ olmadığını nasıl gösteririz?

Tanımın değilini alırız. 3. haftanın kuralı: değil işareti niceleyicilerin önünden geçerken $\exists$ ile $\forall$ yer değiştirir.



Büyük O tanımının negeli

$$\neg \exists C \exists k \forall x (x > k \rightarrow |f(x)| \leq C|g(x)|)$$

Başlangıç

$$\forall C \forall k \exists x \neg (x > k \rightarrow |f(x)| \leq C|g(x)|)$$

Değilleme kuralı, üç kez

$$\forall C \forall k \exists x (x > k \wedge |f(x)| > C|g(x)|)$$

 $\neg(p \rightarrow q) \equiv p \wedge \neg q$

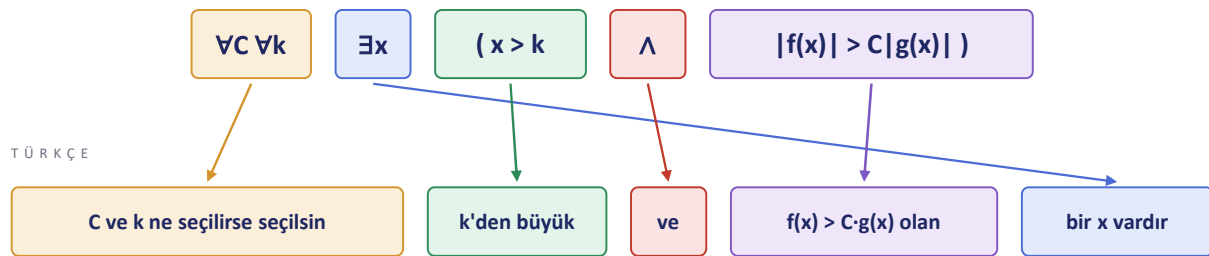
Bu son satır sağlanıyorsa f, O(g) değildir.



Mantıktan Türkçeye

MANTIK

TÜRKÇE



"C ve k ne seçilirse seçilsin, k'den büyük ve f(x) > C·g(x) olan bir x vardır."



Rosen'ın örneği: n^2 , $O(n)$ değildir

$$n^2 \leq Cn \text{ olsun (her } n > k \text{ için)}$$

Tanıkların var olduğunu varsayalım



$$n \leq C$$

İki tarafı n 'ye bölelim ($n > 0$)



$$n = \max(k, C) + 1 \text{ seçelim}$$

Bu n , k 'den büyük ama C 'den de büyük

Çelişki: hiçbir C ve k çifti yetmez. n^2 , $O(n)$ değildir.



Rosen'ın örneği: x^3 , $O(7x^2)$ değildir

$$x^3 > C \cdot 7x^2$$

Göstermemiz gereken eşitsizlik



$$x > 7C$$

İki tarafı x^2 'ye bölelim ($x > 0$)



$$x = \max(k, 7C) + 1 \text{ seçelim}$$

Hem $x > k$ hem $x > 7C$

Her C ve k için uygun bir x bulduk. x^3 , $O(7x^2)$ değildir; oysa $7x^2 = O(x^3)$ idi.



Bir polinomun büyüme hızını en yüksek dereceli terimi belirler.

$f(x) = a_n x^n + \dots + a_1 x + a_0$ ise $f(x) = O(x^n)$. Tanıklar: $C = |a_n| + \dots + |a_1| + |a_0|$ ve $k = 1$.



Rosen'ın ilk örneğindeki $C = 4$ de bu teoremden gelir: $x^2 + 2x + 1$ için $1 + 2 + 1 = 4$.

Teorem, her polinom için tanıkları hazır veriyor: katsayıların mutlak değerlerini toplamak yeterli.



Rosen'ın örneği: $1 + 2 + \dots + n = O(n^2)$

$$1 + 2 + \dots + n$$

İlk n pozitif tam sayı



$$\leq n + n + \dots + n$$

Her terim en fazla n



$$= n^2$$

n tane n

$C = 1$ ve $k = 1$. Kabarcık sıralamasının karşılaştırma sayısı da bu türden bir toplamdı.



Faktöriyel çok hızlı büyür: $20! = 2\,432\,902\,008\,176\,640\,000$.

Yalnızca 20 sayıyı çarptık ve 19 basamaklı bir sayı elde ettik. 81 ilin turunda karşımıza çıkacak $80!$ ise 119 basamaklı.



Rosen'ın örneği: $n! = O(n^n)$

$$n! = 1 \cdot 2 \cdot 3 \cdot \dots \cdot n$$

n tane çarpan



$$\leq n \cdot n \cdot n \cdot \dots \cdot n$$

Her çarpan en fazla n



$$= n^n$$

Tanıklar $C = 1, k = 1$

$$n! = O(n^n).$$



İki tarafın logaritmasını alırsak: $\log n! \leq \log n^n = n \log n$.

Yani $\log n! = O(n \log n)$. Bu tahmin, karşılaştırmayla sıralamanın neden $n \log n$ 'den hızlı olamayacağı gösterilirken karşımıza çıkar.



Rosen'ın örneği: $n = O(2^n)$ ve $\log n = O(n)$

Her pozitif n için $n < 2^n$

Rosen bunu 5. bölümde ispatlıyor



$n = O(2^n)$

$C = 1, k = 1$



$\log n < n$

İki tarafın 2 tabanında logaritması

$\log n = O(n)$: logaritma, doğrusal fonksiyondan yavaş büyür.



Büyük O gösteriminde logaritmanın tabanı önemsizdir.

$\log_{10} n = \log_2 n \div \log_2 10 \approx 0,3 \cdot \log_2 n$. İki logaritma arasında yalnızca sabit bir çarpan var. Bu yüzden tabanı yazmadan $O(\log n)$ deriz.



Sık karşılaşılan büyüme hızları

Gösterim	Adı	n = 10	n = 1000	n = 86 092 168
$O(1)$	Sabit	1	1	1
$O(\log n)$	Logaritmik	4	10	27
$O(n)$	Doğrusal	10	1 000	86 092 168
$O(n \log n)$	Doğrusal-logaritmik	33	9 966	$\approx 2,3 \cdot 10^9$
$O(n^2)$	Karesel	100	10^6	$\approx 7,4 \cdot 10^{15}$
$O(2^n)$	Üstel	1 024	$\approx 10^{301}$	26 milyon basamaklı
$O(n!)$	Faktöriyel	3 628 800	$\approx 4 \cdot 10^{2567}$	hesaplanamayacak kadar büyük

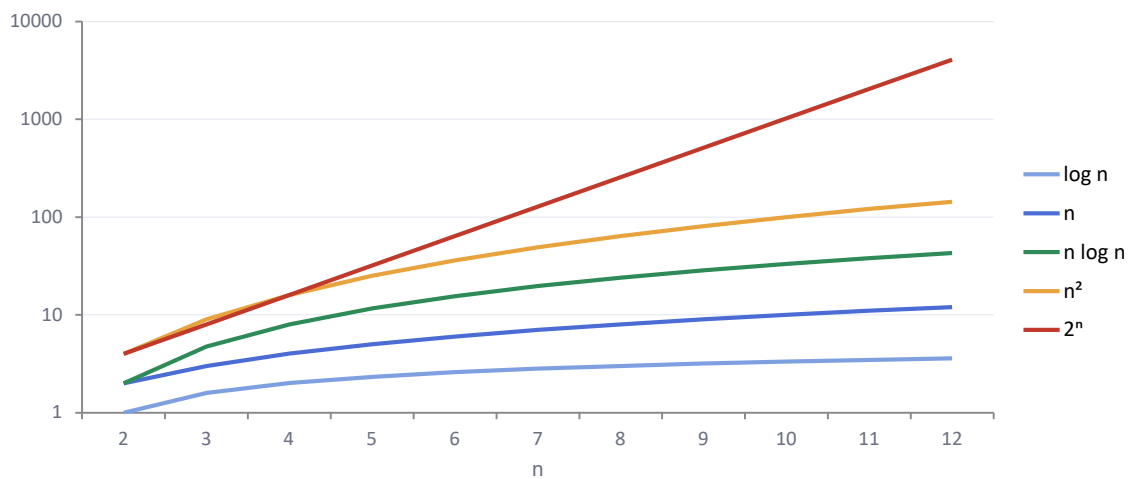
Logaritmalar 2 tabanında; $\log n$ sütunu ikili aramanın adım sayısı olarak yukarı yuvarlandı.

Kaynak: Rosen, Bölüm 3.2

113



Büyüme hızlarını yan yana görelim



Dikey eksen logaritmik: her çizgi 10 katlık bir sıçrama. 2^n , $n = 12$ 'de bile ötekilerden çok uzakta.

Kaynak: Rosen, Bölüm 3.2

114



Saniyede 10^9 işlemle, $n = 86\,092\,168$ için süreler

Büyüme hızı	İşlem sayısı	Süre
$O(\log n)$	27	anında
$O(n)$	86 092 168	yaklaşık 0,09 saniye
$O(n \log n)$	$\approx 2,3 \cdot 10^9$	yaklaşık 2,3 saniye
$O(n^2)$	$\approx 7,4 \cdot 10^{15}$	yaklaşık 86 gün

Daha hızlı bir bilgisayar süreyi birkaç kat kısaltır; daha iyi bir algoritma ise milyonlarca kat.

115



Bu altı fonksiyonu yavaştan hızlıya sıralayın

SORU

$8\sqrt{n}$	$(\log n)^2$	$2n \log n$	$n!$	$(1,1)^n$	n^2
-------------	--------------	-------------	------	-----------	-------

Dizideki her fonksiyon, kendisinden sonra gelenin büyük O'su olmalı.

116



Cevap: yavaştan hızlıya

CEVAP



Logaritmanın her kuvveti, n 'nin her pozitif kuvvetinden yavaş büyür. Tabanı 1'e ne kadar yakın olursa olsun $(1,1)^n$ sonunda her polinomu geçer; $n!$ ise her üstel fonksiyonu.



Toplama kuralı: art arda gelen iki işin toplamında hızlı büyüyen kazanır.

$f_1 = O(g_1)$ ve $f_2 = O(g_2)$ ise $f_1 + f_2 = O(\max(|g_1|, |g_2|))$. Örneğin $3n^2 + 5n + 7 = O(n^2)$.



Çarpma kuralı: iç içe iki işin büyüklükleri çarpılır.

$f_1 = O(g_1)$ ve $f_2 = O(g_2)$ ise $f_1 \cdot f_2 = O(g_1 \cdot g_2)$. Örneğin n kez dönen bir döngünün içindeki $\log n$ 'lik iş, toplamda $O(n \log n)$ eder.



Rosen'ın örneği: $f(n) = 3n \log(n!) + (n^2 + 3) \log n$

$$\log(n!) = O(n \log n) \text{ ve } 3n = O(n)$$

Az önceki örnekten



$$3n \log(n!) = O(n^2 \log n)$$

Çarpma kuralı



$$(n^2 + 3) \log n = O(n^2 \log n)$$

$n > 2$ için $n^2 + 3 < 2n^2$



$$f(n) = O(n^2 \log n)$$

Toplama kuralı

İki terim aynı mertebede: $f(n) = O(n^2 \log n)$.



Rosen'ın örneği: $f(x) = (x + 1) \log(x^2 + 1) + 3x^2$

$$x + 1 = O(x)$$

$x > 1$ için $x + 1 < 2x$



$$\log(x^2 + 1) \leq \log(2x^2) \leq 3 \log x$$

$x > 2$ için



$$(x + 1) \log(x^2 + 1) = O(x \log x)$$

Çarpma kuralı



$$f(x) = O(\max(x \log x, x^2))$$

Toplama kuralı

$x > 1$ için $x \log x \leq x^2$ olduğundan $f(x) = O(x^2)$.



SORU

$f(n) = 4n^2 + 10n \log n + 500$ hangi büyüme sınıfında?

Toplama kuralını kullanın.



CEVAP

$O(n^2)$.

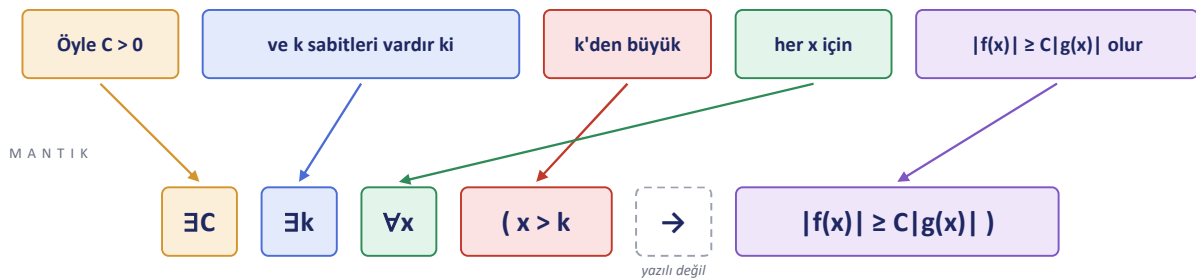
Üç terimden en hızlı büyüyen $4n^2$. $10n \log n$ ve 500 bir noktadan sonra onun yanında önemsiz kalır; 4 katsayısını da büyük O siler.

123



Alt sınır: büyük Ω

TÜRKÇE



$$f(x) = \Omega(g(x)): \exists C \exists k \forall x (x > k \rightarrow |f(x)| \geq C|g(x)|)$$

Büyük O tanımıyla arasındaki tek fark eşitsizliğin yönü.

124



Rosen'ın örneği: $8x^3 + 5x^2 + 7 = \Omega(x^3)$

$$8x^3 + 5x^2 + 7 \geq 8x^3$$

Her pozitif x için



$$|f(x)| \geq 8 \cdot |x^3|$$

$C = 8, k = 0$

$8x^3 + 5x^2 + 7 = \Omega(x^3)$. Bu, $x^3 = O(8x^3 + 5x^2 + 7)$ demekle aynı şeydir.



$f(x) = \Theta(g(x))$: f hem $O(g(x))$ hem de $\Omega(g(x))$ 'tir.

Bu durumda f ile g aynı mertebededir: biri ötekinin hem üst hem alt sınırıdır.



Rosen'ın örneği: $1 + 2 + \dots + n = \Theta(n^2)$

$$1 + 2 + \dots + n \geq \lceil n/2 \rceil + \dots + (n-1) + n$$

Baştaki terimleri atalım

$$\geq \lceil n/2 \rceil + \lceil n/2 \rceil + \dots + \lceil n/2 \rceil$$

Kalan her terim en az $\lceil n/2 \rceil$

$$\geq (n/2) \cdot (n/2) = n^2/4$$

En az $n/2$ terim kaldı

Toplam $\Omega(n^2)$; daha önce $O(n^2)$ olduğunu da görmüştük. Demek ki $\Theta(n^2)$.



Rosen'ın örneği: $3x^2 + 8x \log x = \Theta(x^2)$

$$3x^2 + 8x \log x \leq 3x^2 + 8x^2$$

$x > 1$ için $\log x < x$

$$= 11x^2$$

$C = 11, k = 1: O(x^2)$

$$3x^2 + 8x \log x \geq x^2$$

$x > 1$ için $8x \log x > 0: \Omega(x^2)$

Hem $O(x^2)$ hem $\Omega(x^2)$: $3x^2 + 8x \log x = \Theta(x^2)$.



$3n + 1 = O(n^2)$ de doğrudur, ama $\Theta(n)$ kadar bilgi vermez.

Büyük O yalnızca üst sınırdır; daha gevşek bir üst sınır da doğrudur. Bir algoritmanın mertebesini tam söylemek için Θ kullanırız.



Derecesi n olan bir polinom, x^n ile aynı mertebededir: $a_n \neq 0$ ise $a_n x^n + \dots + a_0 = \Theta(x^n)$.

Rosen'in örnekleri: $3x^8 + 10x^7 + 221x^2 + 1444$ polinomu $\Theta(x^8)$, $x^{19} - 18x^4 - 10112$ polinomu ise $\Theta(x^{19})$ mertebesindedir.



5. BÖLÜM

Karmaşıklık ve Sınırlar

43 gün süren bir kontrol, 81 ilin turu ve hiç yazılamayacak bir program



5. BÖLÜM · KARMAŞIKLIK



Bir algoritmanın zaman karmaşıklığı, girdinin büyüklüğüne göre yaptığı işlem sayısıdır.

Rosen işlem olarak çoğunlukla karşılaştırmaları sayar; arama ve sıralamada temel işlem budur.



En kötü durum analizi, belirli büyüklükteki her girdide en fazla kaç işlem yapılacağını söyler.

Ortalama durum analizi ise bütün girdiler üzerinden ortalama işlem sayısını verir; hesaplaması genellikle daha zordur.



Algoritmanın ne kadar bellek kullandığı da ölçülebilir; buna alan karmaşıklığı denir.

Bu derste zamana odaklanıyoruz. Yine de unutmayın: kümeyle yapılan hızlı tekrar kontrolü, bütün numaraları bellekte tutar.



Rosen'ın örneği: en büyüğü bulmak kaç karşılaştırma ister?

Her $i = 2, \dots, n$ için 2 karşılaştırma

$i \leq n$ ve $max < a_i$



Döngüden çıkarken 1 karşılaştırma

$i = n + 1$ olduğunda



$2(n - 1) + 1 = 2n - 1$

Toplam

Sayı girdinin sırasına bağlı değil: her zaman $2n - 1$, yani $\Theta(n)$. Döngü kontrollerini saymasaydık $n - 1$ bulurduk; mertebe değişmez.



Rosen'ın örneği: doğrusal aramanın en kötü durumu

Her turda 2 karşılaştırma

$i \leq n$ ve $x \neq a_i$



$x = a_i$ ise $2i + 1$ karşılaştırma

i tur ve döngü dışında 1



x listede yoksa $2n + 2$ karşılaştırma

n tur, çıkış ve sondaki if

En kötü durum $2n + 2$ karşılaştırma: $\Theta(n)$.



Rosen'ın örneği: ikili aramanın en kötü durumu

$n = 2^k$ terimli bir liste alalım

Hesabı kolaylaştırmak için



Her aşamada 2 karşılaştırma

$i < j$ ve $x > a_m$



k aşamada liste tek terime iner

$2^k \rightarrow 2^{k-1} \rightarrow \dots \rightarrow 1$



Sonda 2 karşılaştırma daha: $2k + 2$

$i < j$ ve $x = a_i$

$k = \log n$ olduğundan en fazla $2 \log n + 2$ karşılaştırma: $\Theta(\log n)$.



Doğrusal aramada aranan kayıt listede rastgele bir yerdeyse, ortalama $(n + 1) / 2$ eleman karşılaştırılır.

Bu da $O(n)$: ortalama durum en kötü durumun yaklaşık yarısıdır, ama büyüme hızı aynıdır.



Rosen'ın örneği: doğrusal aramanın ortalama durumu

$x, i.$ terimse $2i + 1$ karşılaştırma

Döngü kontrolleri dahil



Ortalama = $(3 + 5 + \dots + (2n + 1)) / n$

x her yerde eşit olasılıkla



= $(2(1 + 2 + \dots + n) + n) / n$

Terimleri ayırılım



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

$1 + \dots + n = n(n + 1)/2$

Ortalama $n + 2$ karşılaştırma: yine $\Theta(n)$.



Kabarcık sıralaması, liste erkenden sıralansa bile bütün turları tamamlar: her zaman $(n - 1)n / 2$ karşılaştırma yapar.

Bu yüzden en kötü durumu da en iyi durumu da $\Theta(n^2)$ mertebesinde.



Rosen'ın araya yerleştirme sürümü en kötü durumda $n(n + 1)/2 - 1$ karşılaştırma yapar.

Rosen yeni elemanın yerini sıralı kısmın başından, bizim Python sürümümüz ise sonundan arar. Sayılar biraz farklı çıkar ama ikisi de $\Theta(n^2)$.

Kaynak: Rosen, Bölüm 3.3, Örnek 6

141



İki $n \times n$ matrisi tanımdaki gibi çarpmak n^3 çarpma ve $n^2(n - 1)$ toplama gerektirir.

Sonuç matrisinin n^2 elemanının her biri n çarpma ve $n - 1$ toplamayla bulunur: $\Theta(n^3)$. Matrisleri ilerleyen haftalarda ayrıntılı göreceğiz.

Kaynak: Rosen, Bölüm 3.3, Örnek 7

142



Döngülere bakarak karmaşıklığı tahmin etmek

```
loops.py
for i in range(n):           # O(n)
    work()

for i in range(n):           # O(n^2)
    for j in range(n):
        work()

i = n
while i > 1:                  # O(log n)
    i = i // 2
    work()
```

Tek döngü

n tur döner: $O(n)$.

İç içe döngü

$n \cdot n$ tur döner: $O(n^2)$.

Yarıya inen döngü

İkili aramadaki gibi: $O(\log n)$.

143



Bu kod kaç kez work() çağırır?

```
quiz.py
for i in range(n):
    j = 1
    while j < n:
        j = j * 2
        work()
```

Cevabınızı büyük O ile sohbete yazın.

144



CEVAP

$O(n \log n)$.

Dış döngü n kez döner. İç döngüde j her adımda iki katına çıkar ve yaklaşık $\log_2 n$ turda n 'yi geçer. Çarpma kuralı: $n \cdot \log n$.

145



Geçen hafta NVİ'nin cümlesini görmüştük: “Türkiye Cumhuriyeti Kimlik Numarası kişiye bir defa verilir ve değiştirilemez.”

Kayıtlarda aynı numaranın iki kişide geçmediğini nasıl kontrol ederiz?

146



İlk akla gelen çözüm: bütün çiftleri tek tek karşılaştırmak

duplicates.py

```
def has_duplicates(ids):
    n = len(ids)
    for i in range(n):
        for j in range(i + 1, n):
            if ids[i] == ids[j]:
                return True
    return False
```

Fikir

Her numarayı kendisinden sonra gelen bütün numaralarla karşılaştır.

Tanıdık geldi mi?

İç döngü her turda bir kısalıyor: kabarcık sıralamasıyla aynı sayma.

147



Kaç karşılaştırma yapar?

$$(n - 1) + (n - 2) + \dots + 1$$

İç döngünün tur sayıları



$$n(n - 1) / 2$$

Kabarcık sıralamasındaki toplam



$$\approx 3,7 \cdot 10^{15}$$

$n = 86\,092\,168$ için

Saniyede bir milyar karşılaştırmayla yaklaşık 43 gün. Karmaşıklık $O(n^2)$.

148



Daha iyi iki çözüm

```
def has_duplicates(ids):  
    # sort first: equal numbers become neighbours  
    s = sorted(ids) # O(n log n)  
    return any(a == b for a, b in zip(s, s[1:])) # O(n)  
  
def has_duplicates_set(ids):  
    # use a set  
    return len(set(ids)) != len(ids) # O(n)
```

Sıralamayla birkaç saniye, kümeyle bir saniyenin altında.

149



**Aynı problemi çözen iki algoritmadan biri 43 gün,
öteki birkaç saniye sürebilir.**

150



Gizli döngülere dikkat

```
hidden_loop.py
# looks like a single loop ...
for x in new_ids:
    if x in old_ids:          # ... but 'in' on a list is a loop too
        report(x)

old_set = set(old_ids)      # build once
for x in new_ids:
    if x in old_set:         # O(1) on average
        report(x)
```

Listede "in" aslında doğrusal aramadır. İlk kod $O(n \cdot m)$, ikincisi $O(n + m)$.

151



Karmaşıklık sınıfları ve bugünkü örnekler

Karmaşıklık	Adı	Bugün gördüğümüz örnek
$\Theta(1)$	Sabit	Listenin ilk elemanına bakmak
$\Theta(\log n)$	Logaritmik	İkili arama
$\Theta(n)$	Doğrusal	Doğrusal arama, en büyüğü bulma, YNO
$\Theta(n \log n)$	Doğrusal-logaritmik	Python'daki sorted()
$\Theta(n^b)$	Polinom	Kabarcık sıralaması, çiftleri karşılaştırma ($b = 2$)
$\Theta(b^n), b > 1$	Üstel	n önermenin doğruluk tablosu (2^n satır)
$\Theta(n!)$	Faktöriyel	81 ilin bütün turlarını denemek

152



Kaba kuvvet (brute force): problemin tanımından yola çıkıp bütün olasılıkları tek tek denemek.

Çiftleri tek tek karşılaştıran tekrar kontrolü, kabarcık sıralaması ve para üstü için yaptığımız 1000 tutarlık deneme bu türden. Yazması kolaydır; çoğu zaman da yavaştır.



Rosen'ın kaba kuvvet örneği: düzlemdeki n nokta arasından birbirine en yakın ikisini bulmak.

Bütün nokta çiftlerinin uzaklığı hesaplanır ve en küçüğü seçilir: $n(n - 1)/2$ uzaklık, yani $O(n^2)$.



81 il merkezinden birbirine en yakın ikisini bu yöntemle bulmak $81 \cdot 80 / 2 = 3240$ uzaklık hesabı ister.

Bu sayı küçük. Ama nokta sayısı bin katına çıkarsa hesap sayısı yaklaşık bir milyon katına çıkar.



SORU

81 ilin her birine bir kez uğrayıp başladığı yere dönen en kısa rotayı bulmak istiyoruz.

Kaba kuvvetle: bütün rotaları tek tek dene, en kisasını seç. Kaç rota var?



81 ilin turunda kaç rota var?

Başlangıç ilini sabitleyelim

Tur bir döngü; nereden başladığı önemsiz



Kalan 80 il $80!$ farklı sırayla dizilir

$80 \cdot 79 \cdot \dots \cdot 1$



Her rota iki yönde aynı: $80! / 2$

Ankara'dan başlayıp saat yönünde ya da tersine



$\approx 3,6 \cdot 10^{118}$ rota

Saniyede bir milyar rotayla $\approx 10^{102}$ yıl

Evrenin yaşı yaklaşık $1,4 \cdot 10^{10}$ yıl. Kaba kuvvet burada umutsuz.

157



2. haftadaki doğruluk tabloları da üstel büyür: n önerme, 2^n satır.

20 önermeyle 1 048 576 satır. 60 önermeyle, saniyede bir milyar satır kontrol eden bir bilgisayarda yaklaşık 36 yıl.

158



Bir formülü doğru yapan bir değer ataması var mı? Buna sağlanabilirlik problemi denir.

Bir atama verilirse formülün doğru olup olmadığını hemen kontrol ederiz. Ama atamayı bulmak için bilinen bütün genel yöntemler en kötü durumda üstel sürer.



Çözümü hızla kontrol edilebilen her problem, hızla çözülebilir mi?

Bu, $P = NP$ sorusudur. Bilgisayar biliminin en ünlü açık problemidir; çözene Clay Matematik Enstitüsü bir milyon dolar ödül veriyor.



Rosen'ın üçlü ayrımı

Çözülebilir

Polinom zamanda çalışan bir algoritması var.

İkili arama, sıralama, tekrar kontrolü.

Pratikte çözülemez

Bilinen bütün algoritmalar en kötü durumda üstel ya da daha yavaş.

81 ilin en kısa turu, sağlanabilirlik.

Çözülemez

Bu problemi çözen bir algoritma hiç yok; yazılamaz.

Durma problemi.

Son kutudaki problemi biraz daha yakından inceleyelim.



3. haftayı şu cümleyle kapatmıştık: Her soruyu cevaplayan bir algoritma yazmak her zaman mümkün değildir.

Şimdi bunun en ünlü örneğine bakalım: durma problemi.



Durma problemi: verilen bir program, verilen bir girdiyle çalıştırıldığında eninde sonunda durur mu?

Derleyiciniz “bu döngü hiç bitmeyecek” uyarısını her durumda doğru verebilseydi bu soruyu çözmüş olurdu.



Böyle bir fonksiyon yazılabildiğini varsayalım

```

halting.py
def halts(program, data):
    ... # assume: always answers correctly

def paradox(program):
    if halts(program, program):
        while True:           # loop forever
            pass
    else:
        return                # stop at once
  
```

halts

Program o girdiyle durursa True, durmazsa False.

paradox

halts ne derse tersini yapar.

Soru

paradox(paradox) durur mu?



paradox kendi kendine verilirse

paradox(paradox) duruyorsa

halts(paradox, paradox) True döner



... o zaman sonsuz döngüye girer: durmaz

Çelişki



paradox(paradox) durmuyorsa

halts False döner



... o zaman hemen return eder: durur

Yine çelişki

Her iki durumda da çelişki çıktı. Demek ki halts fonksiyonu yazılamaz.



Durma problemini çözen bir algoritma yoktur.

Bunu 1936'da Alan Turing gösterdi. Bu bir donanım sınırı değil: bilgisayar ne kadar hızlı olursa olsun böyle bir program yazılamaz.



ARA SINAVA HAZIRLIK

Kendinizi Sınavın

Yedi soru; her birinin ardından cevabı geliyor



KENDİNİZİ SINAYIN



SORU 1

2025 YKS'ye başvuran 2 560 649 aday, aday numarasına göre sıralı bir listede. İkili arama bir adayı en fazla kaç adımda bulur?

Kaynak: ÖSYM, "YKS'ye 2,5 milyondan fazla aday katılacak" duyurusu (16.06.2025)

168

KENDİNİZİ SINAYIN



Cevap 1: 22 adım

$$2^{21} = 2\,097\,152$$

Aday sayısından küçük



$$2^{22} = 4\,194\,304$$

Aday sayısından büyük



$$21 < \log_2 2\,560\,649 < 22$$

Yani $\log_2 n \approx 21,3$

$\lceil \log_2 2\,560\,649 \rceil = 22$. Doğrusal arama en kötü durumda 2 560 649 adım atardı.

Kaynak: ÖSYM, "YKS'ye 2,5 milyondan fazla aday katılacak" duyurusu (16.06.2025)

169

KENDİNİZİ SINAYIN



Hangileri doğru?

SORU 2

(a) $2n^2 + 100n = O(n^2)$

(b) $2n^2 + 100n = O(n^3)$

(c) $2n^2 + 100n = \Theta(n^3)$

(d) $n \log n = O(n)$

Her biri için "doğru" ya da "yanlış" deyin ve gerekçenizi bir cümleyle yazın.

170

KENDİNİZİ SINAYIN



Cevap 2

İfade	Doğru mu?	Neden
(a) $2n^2 + 100n = O(n^2)$	Doğru	En yüksek dereceli terim n^2 ; $C = 102$, $k = 1$
(b) $2n^2 + 100n = O(n^3)$	Doğru	O üst sınırdır; daha gevşek bir sınır da doğrudur
(c) $2n^2 + 100n = \Theta(n^3)$	Yanlış	$\Omega(n^3)$ değil: n^3 'ün sabit bir katına yetişemez
(d) $n \log n = O(n)$	Yanlış	$\log n$ sınırsız büyür; hiçbir sabit C yetmez

171

KENDİNİZİ SINAYIN



SORU 3

$7n + 3 = O(n)$ olduğunu gösterin. Tanıkları açıkça yazın.

172

KENDİNİZİ SINAYIN



Cevap 3

$$7n + 3$$

 $n > 1$ olsun


$$\leq 7n + 3n$$

 $n > 1$ için $3 \leq 3n$


$$= 10n$$

Toplayalım

$C = 10$ ve $k = 1$ tanıklardır. $C = 8$ ve $k = 3$ gibi başka tanıklar da olur.

173

KENDİNİZİ SINAYIN



Bu fonksiyonları yavaştan hızlıya doğru sıralayın

SORU 4

$$2^n$$

$$n^2$$

$$1000$$

$$\log n$$

$$n!$$

$$n \log n$$

$$n$$

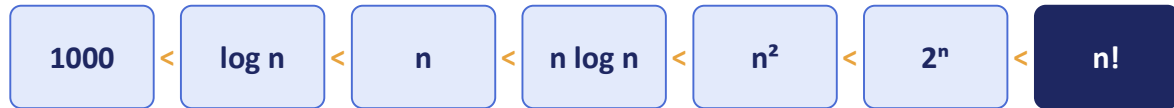
İpucu: $n = 1\,000\,000$ için her birinin yaklaşık değerini düşünün.

174

KENDİNİZİ SINAYIN



Cevap 4



1000 bir sabittir: $O(1)$. Sayı büyük görünse de n ile hiç büyümmez, bu yüzden en başta yer alır.

Kaynak: Rosen, Bölüm 3.2

175

KENDİNİZİ SINAYIN



Soru 5: Bu kodun karmaşıklığı nedir?

```
queries.py
import bisect

ids.sort()
for x in queries:
    i = bisect.bisect_left(ids, x)
    print(i < len(ids) and ids[i] == x)
```

n ids
m queries

n kimlik numarası ve m sorgu için büyük O ile yazın.

176

KENDİNİZİ SINAYIN



CEVAP 5

$O(n \log n + m \log n)$.

Sıralama bir kez yapılır: $O(n \log n)$. Her sorgu bir ikili aramadır: $O(\log n)$, m sorgu için $O(m \log n)$. Art arda gelen iki iş toplanır.

177

KENDİNİZİ SINAYIN



SORU 6

Kasada 5 kuruşluk yok. Para üstü 40 kuruş: açgözlü yöntem kaç madeni para verir, en az kaç parayla verilebilir?

Kaynak: Türkiye Cumhuriyet Merkez Bankası, tedavüldeki madeni paralar

178

KENDİNİZİ SINAYIN



Cevap 6

Açgözlü yöntem

25 kuruş, kalan 15
10 kuruş, kalan 5
1 + 1 + 1 + 1 + 1 kuruş

7 madeni para

En iyi çözüm

10 + 10 + 10 + 10 kuruş

4 madeni para

Açgözlü yöntem her adımda en büyüğünü seçer; bu sefer ilk seçim yanlış çıktı.

179

KENDİNİZİ SINAYIN



SORU 7

“Listedeki sayıların en büyüğünü bul: listeye bak ve en büyük olanı seç.” Bu tarif bir algoritmanın hangi özelliğini sağlamıyor?

Kaynak: Rosen, Bölüm 3.1

180

KENDİNİZİ SINAYIN



CEVAP 7

Belirliliği. “En büyük olanı seç” adımı nasıl yapılacağını söylemiyor.

Asıl işi bu adım yapıyor ama nasıl yapılacağı tarif edilmemiş. Bugün gördüğümüz algoritma ise adımı açıkça yazıyor: her sayıyı şimdiye kadarki en büyükle karşılaştır.

Kaynak: Rosen, Bölüm 3.1

181



ÖZET

Bugün Neler Öğrendik?

Yedi cümlede bugünkü ders ve açılıştaki sorunun cevabı





Algoritma, bir problemi çözmek için kesin olarak tanımlanmış, sonlu sayıda adımdan oluşan bir yöntemdir.

Çay demleme tarifi bir bilgisayar için yeterince kesin değil; yönetmelikteki YNO kuralı ise öyle.



Algoritmaların hızını saniyeyle değil, girdinin büyüklüğüne göre yapılan işlem sayısıyla ölçeriz.



Doğrusal arama en kötü durumda n , ikili arama yaklaşık $\log_2 n$ karşılaştırma yapar.

İkili aramanın şartı, listenin sıralı olması. Saf metin arama $O(nm)$ sürer.



Kabarcık sıralaması ve araya yerleştirme $O(n^2)$ sürer. Açgözlü yöntem hızlıdır ama en iyi her zaman garanti etmez.



$f(x) = O(g(x))$: bir noktadan sonra f , g 'nin sabit bir katını hiç aşmaz.

$\exists C \exists k \forall x (x > k \rightarrow |f(x)| \leq C|g(x)|)$. Alt sınır için Ω , ikisi birden için Θ .



Toplamda en hızlı büyüyen terim kalır; tek döngü $O(n)$, iç içe iki döngü $O(n^2)$, yarıya inen döngü $O(\log n)$.



Doğru algoritmayı seçmek, daha hızlı bir bilgisayar almaktan çok daha etkilidir.

Bazı problemler için bilinen en iyi yöntemler üstel sürer; durma problemi gibi bazılarının ise hiç algoritması yoktur.



86 milyon kişi arasında birini kaç adımda bulursunuz?

Doğrusal arama: 86 092 168 adım

Liste sıralı değilse



İkili arama: $\lceil \log_2 86\,092\,168 \rceil = 27$ adım

Liste sıralıysa

En fazla 27 adım. Liste iki katına çıksa 28 adım.



Bu haftanın konularını nereden çalışabilirsiniz?

Ders kitabı

Rosen, Discrete Mathematics and Its Applications: Bölüm 3.1 (algoritmalar, arama, sıralama, ağgözlü algoritmalar, durma problemi), 3.2 (fonksiyonların büyümesi) ve 3.3 (algoritmaların karmaşıklığı).

Türkçe kaynaklar

TDK sözlüğünde “algoritma” maddesi, lise Bilgisayar Bilimi çalışma kâğıdı, Ufuk Üniversitesi Sınav Yönetmeliği (Madde 42), TÜİK nüfus verileri, ÖSYM 2025-YKS duyurusu ve TCMB madeni para bilgileri.

Görselleştirme

Visualgo sitesinde arama ve sıralama algoritmalarını adım adım izleyebilirsiniz (visualgo.net).

Gelecek hafta: Sayma

Bugün karşımıza çıkan 2^n satır ve $80!/2$ rota gibi sayıları nasıl hesaplayacağımızı göreceğiz: çarpma ve toplama kuralları, permütasyon ve kombinasyon.

Sorularınızı Teams sohbetinden ya da kanaldan iletebilirsiniz.