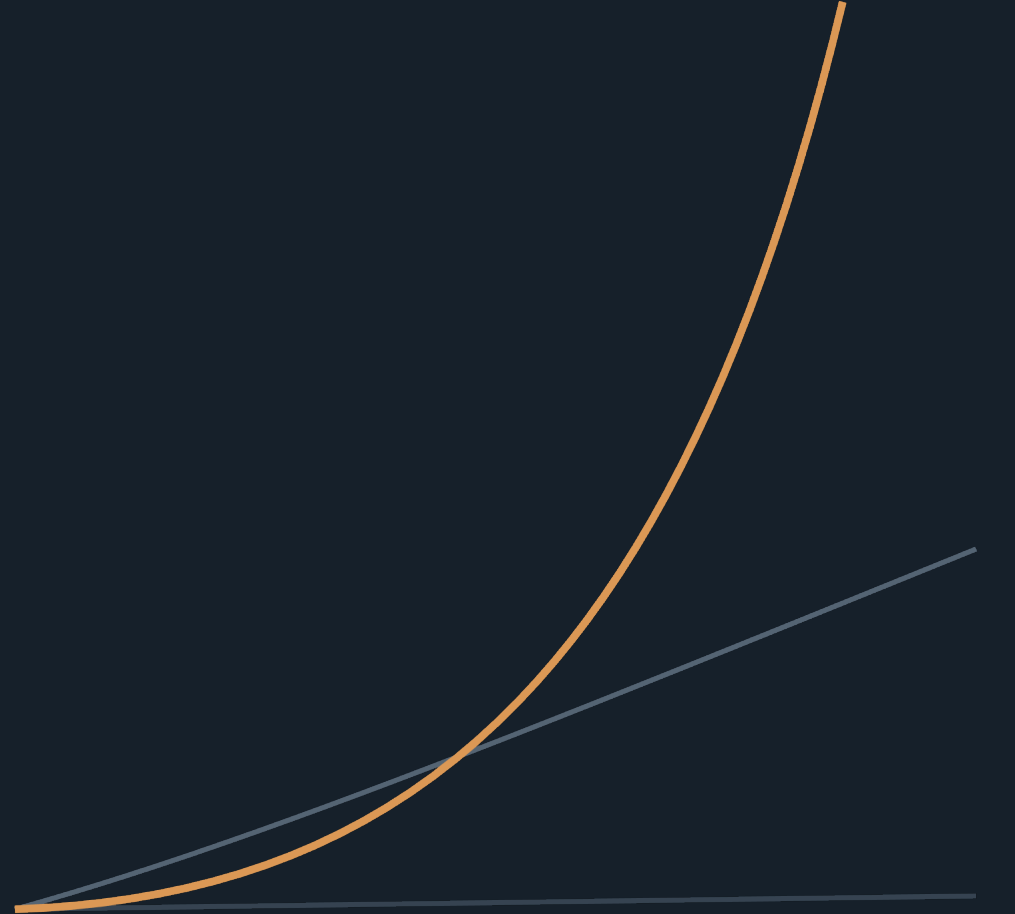


VERİ YAPILARI · 1. HAFTA

Big O Notasyonu

Veri büyüdükçe iş nasıl büyür?



1000

sınav kâğıdı, öğrenci numarasına göre
sıralı

BİR TAHMİN

**Birinizin kâğıdını
arıyorum. En kötü
ihtimalle kaç kâğıda
bakarım?**

Aynı iş için iki yol

Tek tek bakmak

doğrusal arama, *linear search*

1000

bakış, kâğıt en sondaysa ya da hiç yoksa

Ortadan bölmek

ikili arama, *binary search*

10

bakış, her seferinde kalanın yarısını atarak

1000

500

250

125

63

32

16

8

4

2

1

Peki 1 milyon kâğıt olsaydı?

1.000.000

bakış, tek tek bakarak

20

bakış, ortadan bölerek

Bugün bu farkın adını koyacağız. **Adı Big O.**

Veri tipi ve veri yapısı

VERİ TİPİ · DATA TYPE

Tek bir değer

int

float

str

bool

42

3.14

"Elif"

True

VERİ YAPISI · DATA STRUCTURE

Birçok değer ve aralarındaki ilişki

list

dict

set

tuple

notlar = [85, 72, 90, 64]

Veri yapıları bir programlama diline bağlı değildir. Python bu fikirleri göstermek için kullandığımız araç.

Veri yapılarının iki ailesi

DÖNEMİN İLK YARISI

Doğrusal

Linear. Elemanlar tek bir sıra halinde.

Dizi

array

yan yana kutular

Bağlı liste

linked list

her eleman sonrakini gösterir

Yığın

stack

üst üste tabaklar

Kuyruk

queue

banka sırası

DÖNEMİN İKİNCİ YARISI

Doğrusal olmayan

Non-linear. Elemanlar dallanır, ağ kurar.

Ağaç

tree

klasör yapısı gibi

Heap

heap

en büyük hep tepede

Graf

graph

harita, sosyal ağ

Trie

prefix tree

otomatik tamamlama

Hash tablosu (*hash table*), yani Python'daki dict ve set, iki aileye de tam oturmaz. Ona ayrı bir hafta ayıracağız.

Hangi işte hangi yapıyı seçeceğiz?

Yapıları karşılaştırmak için ortak bir dile ihtiyacımız var.

Big O

Hız nasıl ölçülür?



Eski dizüstü bilgisayar

4 saniye



Yeni masaüstü bilgisayar

1 saniye

Aynı kod, iki farklı süre. O yüzden **adımları sayarız.**

Adım dediğimiz şey temel işlem (*basic operation*). Karşılaştırma, atama ya da toplama gibi.

Veri büyüdükçe iş nasıl büyüyor?

n

girdi boyutu, *input size*

n = 40

40 kişilik sınıf

n = 1.000.000

1 milyon satırlık dosya

Karmaşıklık burada maliyet demek

Kodun okunmasının zor ya da kolay olmasıyla ilgisi yok. Algoritmanın ne kadar kaynak harcadığını anlatır.

ZAMAN KARMAŞIKLIĞI

time complexity

Kaç adım atıyor?

Bugünün konusu

BELLEK KARMAŞIKLIĞI

space complexity

Ne kadar ek yer kullanıyor?

Kısaca değineceğiz

Kavram, Hartmanis ve Stearns'ün 1965 tarihli makalesiyle yerleşti. İkisi bu çalışmayla 1993 Turing Ödülü'nü aldı.

Big O nedir? Big O notation

Veri büyüdükçe işin **en fazla** hangi hızla büyüdüğünü anlatan bir gösterim.

En fazla diyoruz, çünkü Big O bir üst sınırdır (*upper bound*).

SORULACAK TEK SORU

Veri 2 katına çıkarsa iş ne olur?

ADI NEREDEN GELİYOR

O harfi Almanca *Ordnung* yani mertebe kelimesinden gelir. Gösterimi 1894'te matematikçi Paul Bachmann önerdi.

$O(1)$ **Sabit zaman** *constant time*

Günlük hayattan. Sınıf listesinde 5. sıradaki öğrenci kim? Liste ne kadar uzun olursa olsun doğrudan o satıra gideriz.

```
def ilk_eleman(liste):  
    return liste[0]  
liste = [7, 3, 9, 1, 5]  
ilk_eleman(liste)
```

Bir fonksiyon (function) tanımlıyoruz

İlk elemanı doğrudan alır, listenin boyuna bakmaz

Beş elemanlı bir liste (list)

Sonuç 7. Sıra numarası (index) 0'dan başlar

Veri 2 katına çıkınca ➡ iş aynı kalır

Turuncu satır Big O'nun kaynağı

$O(n)$ Doğrusal zaman *linear time*

Günlük hayattan. Yoklama listesinde Elif var mı? Liste sırasız. En kötü durumda (worst case) Elif listede yoktur ve bütün isimlere bakarız.

```
def ara(liste, aranan):  
    for eleman in liste:  
        if eleman == aranan:  
            return True  
    return False
```

İki girdi alır, liste ve aradığımız değer

Döngü (loop) her elemana bir kez uğrar, n tur

Bu eleman aradığımız mı? == eşit mi demek

Bulduk, True (doğru) döndürüp hemen çıkarız

Döngü bitti ve bulamadık, False (yanlış)

Veri 2 katına çıkınca ➡ iş 2 katına çıkar

Turuncu satır Big O'nun kaynağı

$O(n^2)$ **Karesel zaman** *quadratic time*

Günlük hayattan. Sınıfta aynı gün doğmuş iki kişi var mı? Döngünün içinde ikinci bir döngü var, buna iç içe döngü (nested loop) denir.

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

Aynı değerden iki tane var mı?

Dış döngü (outer loop), her öğrenci için bir tur

İç döngü (inner loop), ondan sonraki herkes

İki doğum günü aynı mı?

Aynısını bulduk

Bütün ikilileri denedik, tekrar yok

Veri 2 katına çıkınca ➡ **iş 4 katına çıkar**

Turuncu satırlar Big O'nun kaynağı

Aynı soru, farklı yapı

Herkes tahtadaki takvime kendi gününü işaretliyor. Python'da takvimin yerini küme (set) alıyor.

```
def tekrar_var_mi_hizli(liste):  
    gorulenler = set()  
    for eleman in liste:  
        if eleman in gorulenler:  
            return True  
        gorulenler.add(eleman)  
    return False
```

Aynı soru, bu sefer takvim fikriyle
Boş bir küme (set). Takvimimiz bu
Tek döngü, her öğrenci bir kez geliyor, n tur
Bu gün işaretli mi? Kümede ortalama tek adım
İşaretliyse aynı gün doğan iki kişi var
Değilse günü işaretle, yani kümeye ekle
Herkes geldi, tekrar yok

Önceki çözüm

$O(n^2)$

Bu çözüm

$O(n)$

Bedeli ek bellek, *time-space tradeoff*

$O(\log n)$ **Logaritmik zaman** *logarithmic time*

İkili arama (binary search). Sıralı listede, sol ve sağ aralığın uçları.

```
while sol <= sag:
    orta = (sol + sag) // 2
    if liste[orta] == aranan:
        return True
    elif liste[orta] < aranan:
        sol = orta + 1
    else:
        sag = orta - 1
```

Aralık boşalmadığı sürece dönen döngü

Aralığın ortası. // tam sayı bölmesi

Ortadaki kâğıt aradığımız mı?

Bulduk

Ortadaki küçükse aranan sağ yarıda

Sol yarıyı tamamen atıyoruz

Değilse aranan sol yarıda

Sağ yarıyı tamamen atıyoruz

Veri 2 katına çıkınca  **iş sadece 1 adım artar**

Formülsüz logaritma logarithm

$\log n$, n sayısını kaç kere ikiye bölersek 1'e ineceğimizi söyler.

1.000 kâğıt

10

bölme

1 milyon kâğıt

20

bölme

1 milyar kâğıt

30

bölme

Dünyadaki herkes

33

bölme, 8 milyardan fazla
kişi

Veri iki katına çıkınca sadece **bir bölme daha** gerekir.

İki sınıf daha

$O(n \log n)$

linearithmic time

İyi sıralama algoritmaları

Python'daki `sorted()` bu sınıfta. $O(n)$ 'den biraz yavaş, $O(n^2)$ 'den çok daha hızlı. Sıralama konusunda göreceğiz.

$O(2^n)$

üstel zaman, exponential time

Bütün olasılıkları denemek

Bir sınıftan kurulabilecek bütün grupları listelemek. 60 kişilik sınıfta, saniyede 1 milyar işlem yapan bir bilgisayarla 36 yıl sürer.

SIRA SİZDE

Hızlıdan yavaşa doğru sıralayın

$O(n^2)$

$O(1)$

$O(n \log n)$

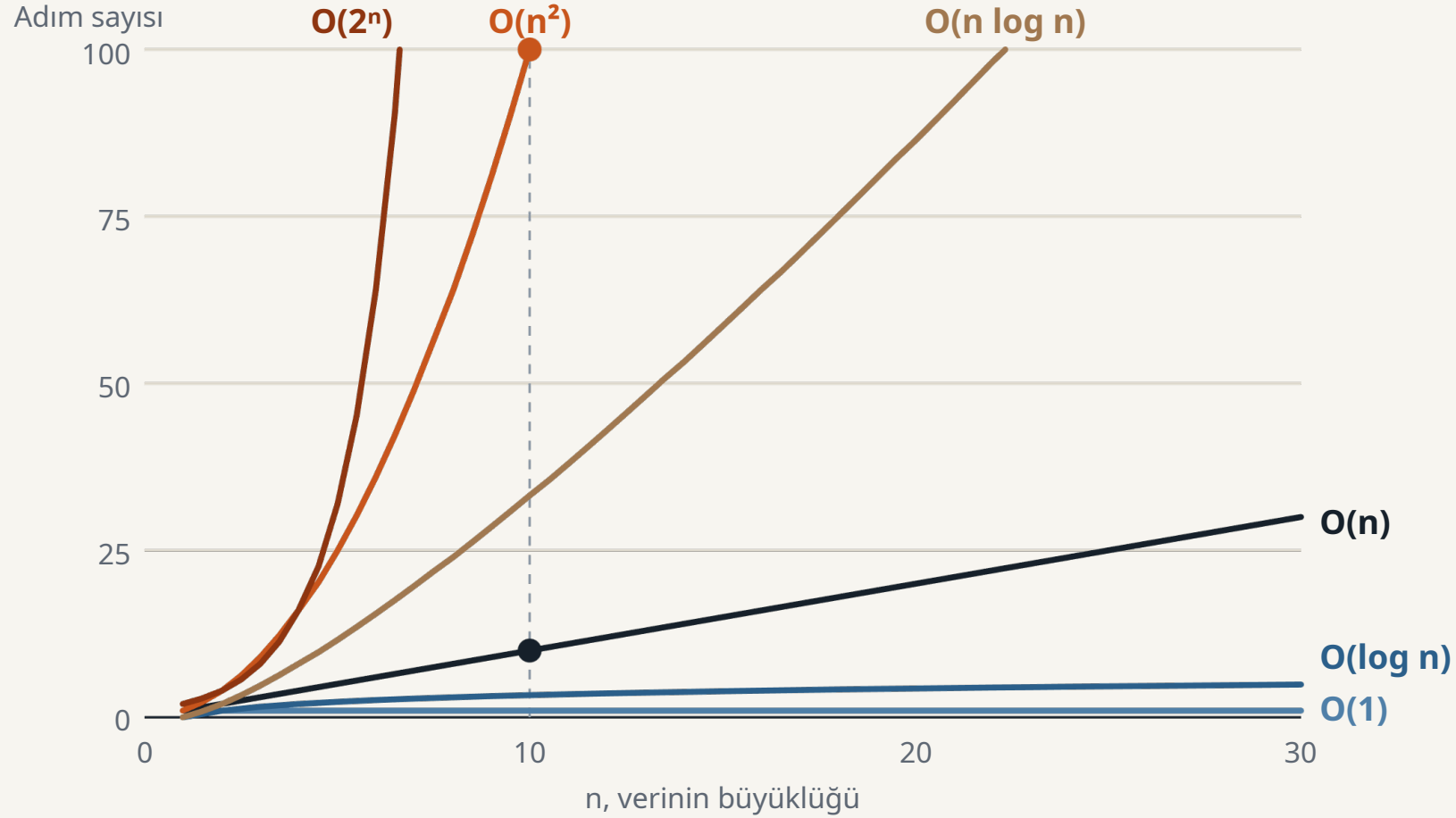
$O(2^n)$

$O(\log n)$

$O(n)$

Bir dakika düşünün, sonra yanınızdakiyle karşılaştırın.

Eğriler çok çabuk ayrılıyor



n = 10 iken

$O(n)$ 10 adım

$O(n^2)$ 100 adım

$O(2^n)$ 1.024 adım

n büyüyünce aradaki uçurum

n	$O(\log n)$	$O(n)$	$O(n^2)$
10	4	10	100
1.000	10	1.000	1.000.000
1.000.000	20	1.000.000	1.000.000.000.000

$O(n)$ yöntem

saniyenin binde biri

$O(n^2)$ yöntem

yaklaşık 17 dakika

n 1 milyon iken, saniyede 1 milyar işlem yapan bir bilgisayarda

Koda bakarak Big O bulmak

Kodda ne görüyoruz	Big O
Döngü yok	$O(1)$
Listeyi baştan sona gezen tek döngü (loop)	$O(n)$
İç içe iki döngü (nested loops)	$O(n^2)$
Arka arkaya iki döngü	$O(n)$, toplanır
Her turda yarıya inen döngü	$O(\log n)$
Hep 5 kez dönen döngü, range(5)	$O(1)$

Döngünün varlığına değil, **kaç kez döndüğüne** bakarız.

İki sadeleştirme kuralı

KURAL 1

Sabitler atılır

drop the constants

$$2n \rightarrow O(n)$$

$n/2$ ya da $100n$ de $O(n)$. Veri iki katına çıkınca hepsinde iş iki katına çıkıyor.

KURAL 2

Küçük terimler atılır

drop the non-dominant terms

$$n^2 + n \rightarrow O(n^2)$$

n bin iken işin yüzde 99,9'u n^2 kısmından geliyor. Geri kalanı sonucu değiştirmiyor.

Gizli döngü

Bir değerin listede olup olmadığına bakmaya üyelik kontrolü (*membership test*) denir. Tek satırdır ama her zaman tek adım değildir.

```
x in liste  
# O(n), listeyi baştan sona tarar  
  
x in kume  
# O(1), ortalamada
```

Listede 100 arama

0,5 saniye

Kümede 100 arama

0,0001 saniye

1 milyon elemanda, yuvarlanmış süreler. Fark binlerce kat.

DERSİN TEK CÜMLESİ

Big O, veri büyüdükçe işin nasıl büyüdüğünü anlatır.

Veri 2 katına çıkınca

$O(1)$

iş aynı kalır

$O(\log n)$

1 adım artar

$O(n)$

2 katına çıkar

$O(n^2)$

4 katına çıkar

YAKLAŞIK 25 DAKİKA

Şimdi kodla pekiştirelim



01_ders.ipynb

Hız farkını birlikte çalıştırıyoruz. $n = 8000$ olunca süre ne olur? Önce tahmin, sonra çalıştırma.



02_tahmin_et.ipynb

İkişerli çalışın. Yedi fonksiyonun Big O değerini 7 dakikada tahmin edin, sonra birlikte oylayacağız.

Erken bitirenler için ek görev. 7. fonksiyonu set kullanarak hızlandırın.

GELECEK HAFTA

Diziler (*arrays*) ve Python listesi

Listenin sonuna eklemek çok hızlı,
başına eklemek yavaş. **Neden?**