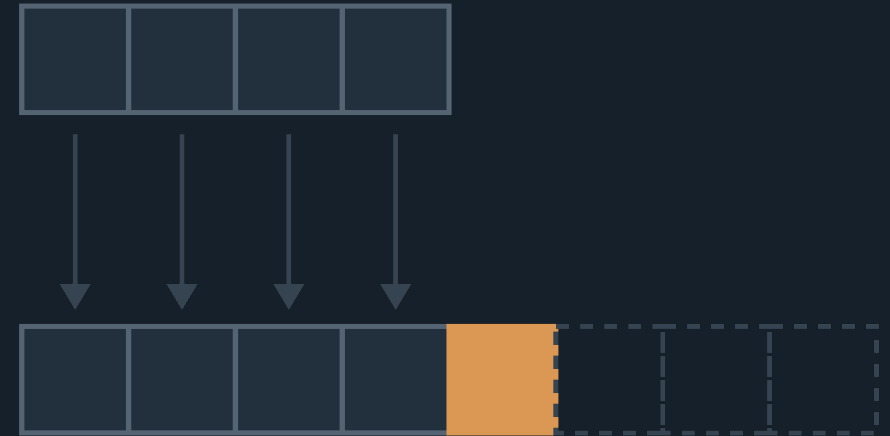


VERİ YAPILARI · 2. HAFTA

Diziler ve Python Listesi

array



GEÇEN HAFTADAN ÜÇ SORU

Bu fonksiyonların Big O değeri ne?

1

```
def ilk_bes(liste):  
    for i in range(5):  
        print(liste[i])
```

3

```
def ortak_var_mi(liste1, liste2):  
    for sayi in liste1:  
        if sayi in liste2:  
            return True  
    return False
```

2

```
def en_kucuk_ve_en_buyuk(liste):  
    en_kucuk = liste[0]  
    for sayi in liste:  
        if sayi < en_kucuk:  
            en_kucuk = sayi  
    en_buyuk = liste[0]  
    for sayi in liste:  
        if sayi > en_buyuk:  
            en_buyuk = sayi  
    return en_kucuk, en_buyuk
```

3. soruda iki listenin de n elemanlı olduğunu düşünün.

Dizi nedir? **array**

Birçok değeri tek bir isim altında, sıra halinde tutan veri yapısı. Python'da dizi olarak liste (*list*) kullanılır.

dizi olmadan

40 ayrı değişken

not1 = 85

not2 = 72

not3 = 90

...

not40 = 64

dizi ile

Tek bir dizi

notlar = [85, 72, 90, 64]

85	72	90	64
0	1	2	3

Kutudaki her değer bir eleman (*element*), altındaki sayı sıra numarası (*index*). Sayma 0'dan başlar.

Bir dizide dört iş yaparız **okumak** **eklemek** **silmek** **aramak**

BUGÜNÜN ANA SORUSU

Sona eklemek neden **hızlı**, başa eklemek neden **yavaş**?

Önce ikisinin ne olduğuna ve neden yapıldığına bakacağız. Sonra hız farkının sebebini bulacağız.

Sona eklemek **append**

85	72	90	64	77
0	1	2	3	4

```
notlar = [85, 72, 90, 64]
notlar.append(77)
# [85, 72, 90, 64, 77]
```

Var olan elemanların yeri değişmez. Yeni eleman sıranın sonuna geçer.

Ne zaman yaparız?

Geliş sırasını korumak istediğimizde

1. Yeni kayıt olan öğrenci yoklama listesinin sonuna yazılır
2. Yeni gelen mesaj sohbetin sonuna eklenir
3. Sensörden gelen her ölçüm listenin sonuna eklenir
4. Sepete atılan ürün sepetin sonuna gelir

Başa eklemek `insert(0, x)`

99	85	72	90	64
0	1	2	3	4
yeni	önceden 0	önceden 1	önceden 2	önceden 3

```
notlar = [85, 72, 90, 64]
notlar.insert(0, 99)
# [99, 85, 72, 90, 64]
```

Yeni eleman 0 numaraya geçti. Diğer herkesin sıra numarası bir arttı.

Ne zaman yaparız?

En yenin en önde durmasını istediğimizde

1. Haber sitesinde en yeni haber en üstte durur
2. Posta kutusunda en son gelen ileti en üsttedir
3. Tarayıcı geçmişinde son ziyaret edilen sayfa en baştadır
4. Acil hasta bekleme sırasının başına alınır

`insert(k, x)` istenen yere de ekler. Buna araya ekleme (*insertion*) denir.

Çıkarmak da iki türlü

notlar.pop()

Sondan çıkarmak

Son elemanı çıkarır ve geri verir.

Geri al tuşu en son yapılan işlemi geri alır.

notlar.pop(0)

Baştan çıkarmak

İlk elemanı çıkarır ve geri verir.

Banka sırasında ilk gelen müşteriye ilk hizmet verilir.

Bu iki fikir 4. ve 5. haftanın konusu, yığın (*stack*) ve kuyruk (*queue*).

```
notlar.append(x)
```

sona ekler

```
notlar.insert(0, x)
```

başa ekler

BİR TAHMİN

**İkisi de tek satır, ikisi de tek eleman ekliyor.
Hangisi daha hızlı?**

Sınıftaki uzun bir sıra

Sona oturmak

geç kalan öğrenci boş sandalyeye oturur

0

kişi yerinden kalkar

Başa oturmak

1 numaralı sandalyeye oturmak isterse

40

kişi birer sandalye sağa kayar

1

2

3

4

...

38

39

40

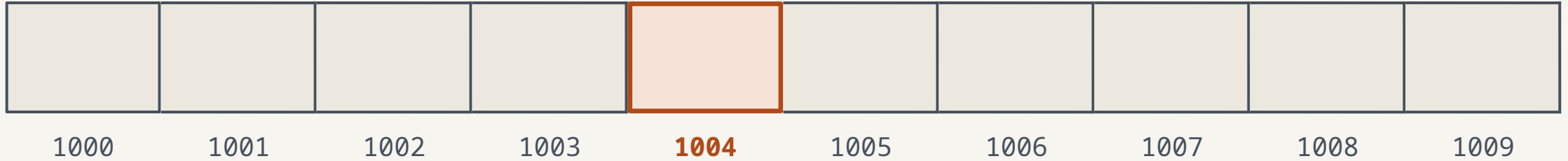
boş

boş

boş

Bellek numaralı kutulardan oluşur

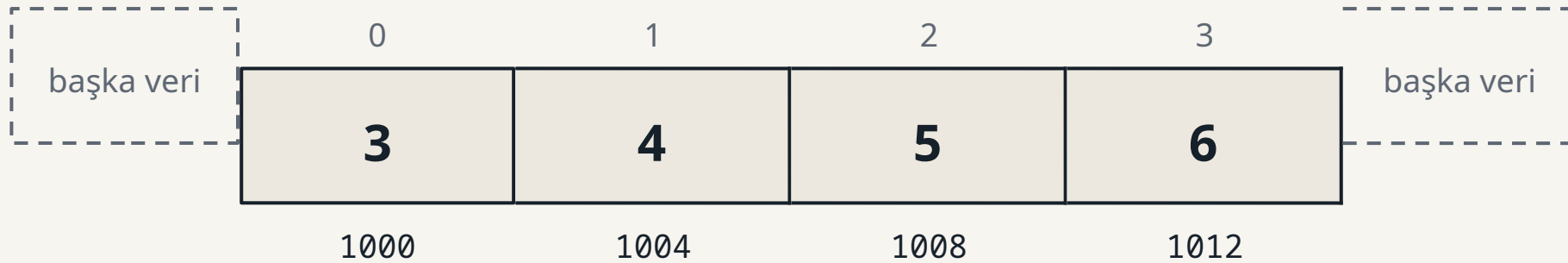
Her kutunun bir numarası var. Bu numaraya adres (*address*) denir. Her kutu bir bayt (*byte*) tutar.



Bilgisayar adresini bildiği kutuya **doğrudan** gider. Aradaki kutulara bakmaz.

Dizi bellekte nasıl durur?

Elemanlar bellekte yan yana, aralarında boşluk olmadan durur. Her eleman aynı boyda yer kaplar.



Bitişik

contiguous, aralarında boşluk yok

Aynı boyda

her eleman 4 bayt

Başlangıcı belli

dizi 1000 numaralı adreste başlıyor

$O(1)$ Sıra numarasıyla erişim *index access*

Adres hesabı. i numaralı elemanın adresi, başlangıç adresi artı i çarpı eleman boyudur.

```
def adres_hesapla(baslangic, boy, i):  
    return baslangic + i * boy  
adres_hesapla(1000, 4, 3)  
adres_hesapla(1000, 4, 500_000)
```

Başlangıç adresi, eleman boyu ve sıra numarası

Bir çarpma, bir toplama. Dizinin boyuna bakmaz

Sonuç 1012, yani 3 numaralı elemanın adresi

Sonuç 2.001.000. Dizi 1 milyon elemanlı olsa da hesap aynı

Veri 2 katına çıkınca ➡ iş aynı kalır

Turuncu satır Big O'nun kaynağı

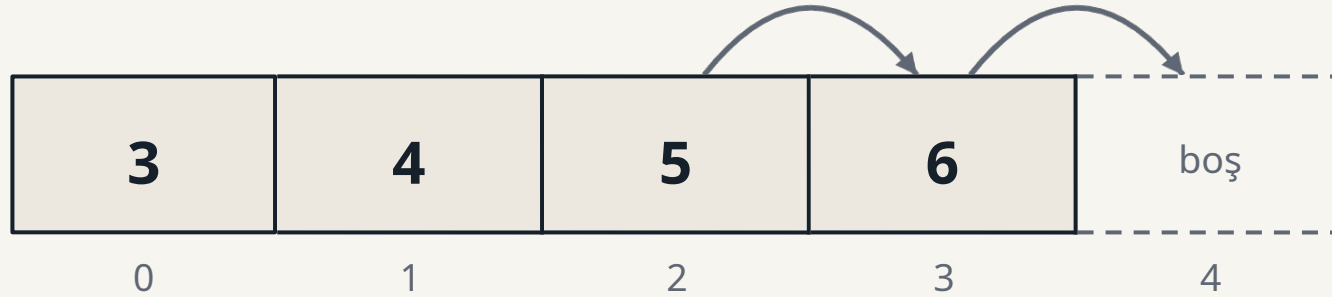
Araya eklemek için arkadakiler kayar

0 değerini 2 numaralı yere ekliyoruz. Kaydırma sondan başlar, yoksa bir eleman diğerinin üstüne yazılır.

2. kayma

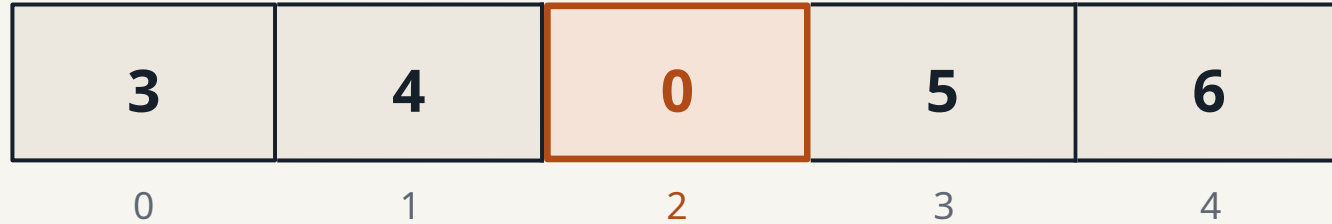
1. kayma

Önce



2 eleman kaydı

Sonra



yeni eleman

$O(n)$ Araya ekleme *insertion*

Günlük hayattan. Sıranın başına oturmak isteyen öğrenci için herkes bir sandalye kayar.

```
def araya_ekle(kutular, boyut, konum, x):  
    for i in range(boyut, konum, -1):  
        kutular[i] = kutular[i - 1]  
    kutular[konum] = x  
    return boyut + 1
```

x değerini konum numaralı yere ekler

Sondan konuma kadar geriye sayar, boyut eksi konum tur

Her elemanı bir sağa kaydırır

Açılan boşluğa yeni değeri yazar

Dizide artık bir eleman fazla var

Veri 2 katına çıkınca ➡ iş 2 katına çıkar

Turuncu satır Big O'nun kaynağı

ANA SORUNUN CEVABI

Kim kayıyor?

İşlem	Python	Yer değiştiren	Big O
Sona ekleme	liste.append(x)	hiçbiri	O(1)
Sondan çıkarma	liste.pop()	hiçbiri	O(1)
Başa ekleme	liste.insert(0, x)	n eleman	O(n)
Baştan çıkarma	liste.pop(0)	n eksi 1 eleman	O(n)
Ortaya ekleme	liste.insert(k, x)	n eksi k eleman	O(n)

Başa eklemek herkesi kaydırır, sona eklemek kimseyi.

Sıra numarası mı, değer mi?

liste[5]

5 numaralı eleman ne?

Adres hesabıyla doğrudan gider

$O(1)$

5 in liste

5 değeri listede var mı?

Elemanlara tek tek bakmak zorunda

$O(n)$

Dolap numarasını biliyorsan doğrudan gidersin. Sadece içindekini biliyorsan dolapları tek tek açarsın.

Sabit boyutlu dizi *static array*

Boyutu baştan bellidir ve değişmez. Dolunca yeni eleman eklenemez.



İyi yanı

Ne kadar bellek kullanacağı baştan belli. Taşınma yok.

Kötü yanı

Boyutu önceden bilmek gerekir. Büyük açarsan bellek boşa gider, küçük açarsan yetmez.

Dinamik dizi *dynamic array*

İçinde sabit boyutlu bir dizi taşır. Dolunca daha büyüğüne taşınır. Dışarıdan bakınca sınırsız büyüyen bir dizi gibi görünür.

boyut 5 · size, len(liste)



kapasite 8 · capacity

Sona eklemek hızlı, çünkü sonda **boş kutular** var.

Dizi dolunca dört adım

eski dizi, kapasite 4, dolu



yeni dizi, kapasite 8, sonda 3 boş kutu

1. Daha büyük yeni bir dizi aç, genelde iki katı
2. Eski elemanları tek tek kopyala
3. Yeni elemanı sona yaz
4. Eski diziyi bırak

Taşınma olan ekleme bütün elemanları kopyalar. Sonraki 3 ekleme ise hiç kopyalama yapmaz.

Dinamik dizi kodla

```
class DinamikDizi
```

```
def ekle(self, deger):  
    if self.boyut == self.kapasite:  
        self.buyut(2 * self.kapasite)  
    self.kutular[self.boyut] = deger  
    self.boyut += 1  
def buyut(self, yeni_kapasite):  
    yeni_kutular = [None] * yeni_kapasite  
    for i in range(self.boyut):  
        yeni_kutular[i] = self.kutular[i]  
    self.kutular = yeni_kutular  
    self.kapasite = yeni_kapasite
```

Sona bir eleman ekler. self dizinin kendisi
Bütün kutular dolu mu?
Doluyorsa iki katı büyüklüğe taşın
İlk boş kutuya yaz
Bir eleman arttı
Daha büyük kutulara taşınma
Yeni ve boş kutular
Bütün elemanları tek tek kopyalar, n tur
Eski kutudan yeni kutuya
Artık yeni kutular kullanılıyor
Kapasite güncellenir

Taşınma olan ekleme ➡ **O(n)**

Turuncu satır Big O'nun kaynağı

Bir kutu mu, iki katı mı?

Boş diziye 1.000 eleman ekliyoruz. Kapasite dolunca nasıl büyütürsek toplam kaç eleman kopyalanır?

her dolduğunda

1 kutu ekle

499.500

kopyalama, $O(n^2)$

her dolduğunda

100 kutu ekle

4.510

kopyalama, yine $O(n^2)$

her dolduğunda

2 katına çıkar

1.023

kopyalama, toplam $O(n)$

1 milyon elemanda fark daha da açılıyor. Bir kutu ekleyerek yaklaşık 500 milyar, iki katına çıkararak yaklaşık 1 milyon kopyalama.

AMORTİZE MALİYET · AMORTIZED COST

Her ekleme 3 jeton öder

1 jeton

elemanı kutusuna yazmaya gider

2 jeton

kumbaraya atılır

Taşınma

kumbaradaki jetonlarla ödenir

Kapasite 4'ten 8'e çıktıktan sonra 4 ekleme kumbaraya 8 jeton koyar. Dizi 8'de dolunca 8 elemanı kopyalamak tam 8 jeton tutar.

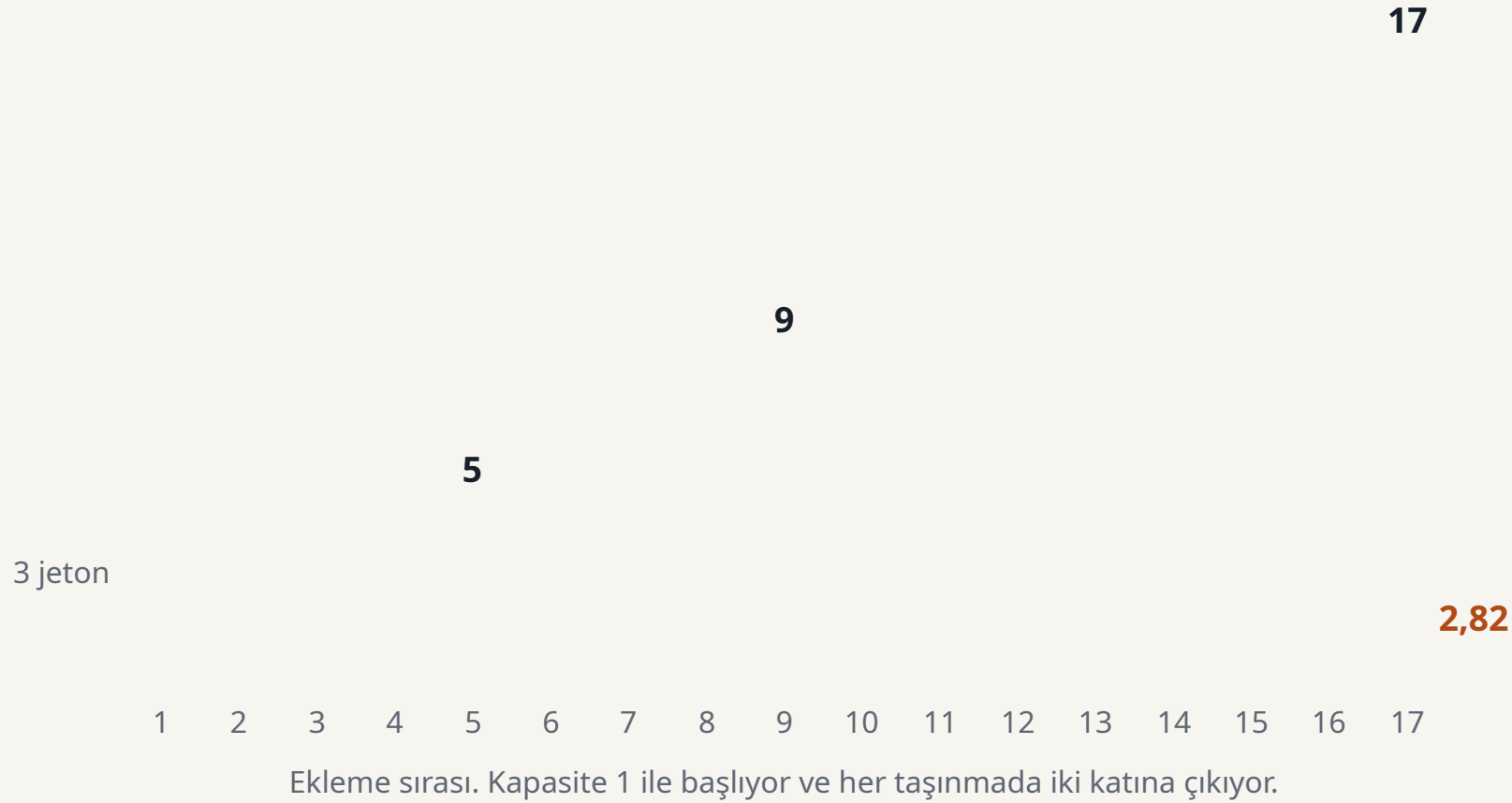
Sona ekleme

$O(1)$

amortize

Tek bir ekleme bazen $O(n)$ sürer, ortalaması sabittir.

Tek tek pahalı, ortalamada ucuz

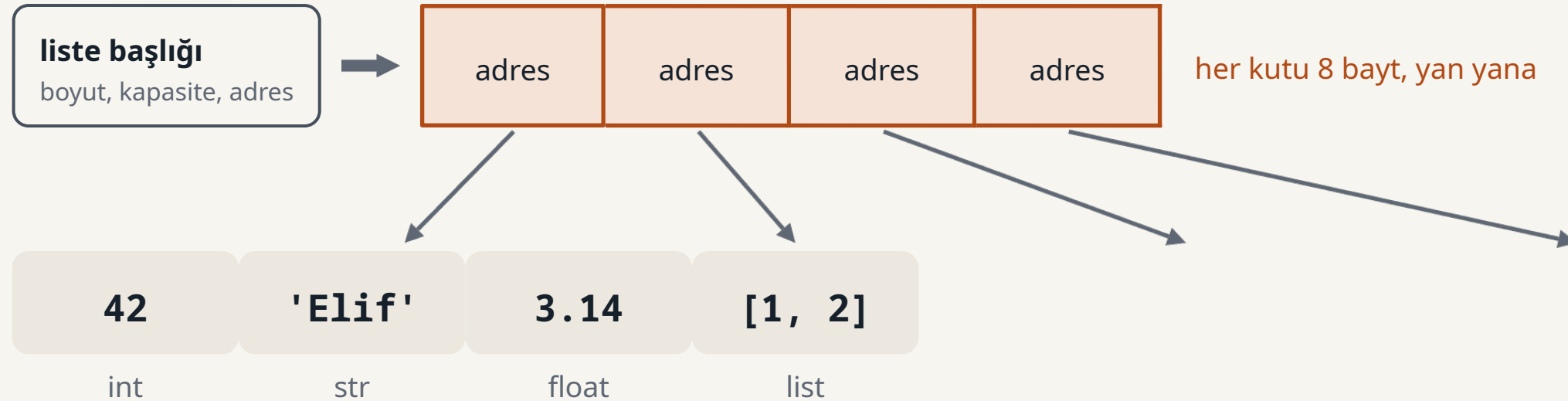


17 ekleme sonunda
toplam **48** jeton
ortalama **2,82**
jeton

Gri çubuk bir eklemenin
maliyeti. Turuncu çizgi o
ana kadarki ortalama.
Kesikli çizgi 3 jeton.

Python listesi arka planda

Bir dinamik dizi. Ama kutularda nesnelerin kendisi değil, adresleri yani referansları (*reference*) durur.



Kutular yine aynı boyda, adres hesabı yine çalışır ve **`liste[i]` yine $O(1)$** . Nesneler bellekte herhangi bir yerde durabilir.

Python listesinde ne ucuz, ne pahalı?

İşlem	Örnek	Maliyet
Sıra numarasıyla okuma	<code>liste[i]</code>	$O(1)$
Sona ekleme	<code>liste.append(x)</code>	$O(1)$ amortize
Sondan çıkarma	<code>liste.pop()</code>	$O(1)$
Başa ekleme	<code>liste.insert(0, x)</code>	$O(n)$
Baştan çıkarma	<code>liste.pop(0)</code>	$O(n)$
Değer arama	<code>x in liste</code>	$O(n)$
Dilim alma, k elemanlı	<code>liste[a:b]</code>	$O(k)$

Sona dokunan işler ucuz, başa ve ortaya dokunan işler pahalı.

DERSİN TEK CÜMLESİ

Dizide sıra numarasıyla okumak ve sona eklemek ucuz, başa ve araya eklemek pahalı.

liste[i]

O(1), adres hesabı

append(x)

O(1) amortize

insert(0, x)

O(n), herkes kayar

pop(0)

O(n), herkes kayar

YAKLAŞIK 22 DAKİKA

Şimdi kodla pekiştirelim



01_ders.ipynb

Projeksiyonda birlikte çalıştırıyoruz. Her ölçümden önce siz tahmin ediyorsunuz. n 100.000 olunca başa ekleme ne kadar sürer?



Kağıt üzerinde alıştırma

İkişerli çalışın. Sonraki iki slayttaki yedi fonksiyonun Big O değerini 7 dakikada tahmin edin, sonra birlikte oylayacağız.

Yanınızda kağıt ve kalem olsun. Erken bitirenler için ek görev de var.

Tahmin alıştırması

Soru 1 ile 4. n listedeki eleman sayısı. Tahmininizi kağıda yazın.

Soru 1

```
def son_elemani_degistir(liste, deger):  
    liste[-1] = deger  
liste[-1] listenin son elemanıdır
```

Soru 2

```
def basa_ekle(liste, deger):  
    liste.insert(0, deger)  
insert(0, x) x'i listenin başına ekler
```

Soru 3

```
def kopyala(liste):  
    yeni = []  
    for x in liste:  
        yeni.append(x)  
    return yeni  
append, x'i listenin sonuna ekler
```

Soru 4

```
def ilk_yarisi(liste):  
    return liste[:len(liste) // 2]  
liste[:k] listenin ilk k elemanını verir, // tam sayı bölmesi
```

Tahmin alıştırması

Soru 5 ile 7. n listedeki eleman sayısı. Tahmininizi kağıda yazın.

Soru 5

```
def sondan_bosalt(liste):  
    while len(liste) > 0:  
        liste.pop()
```

pop() listenin son elemanını çıkarır

Soru 6

```
def bastan_bosalt(liste):  
    while len(liste) > 0:  
        liste.pop(0)
```

pop(0) listenin ilk elemanını çıkarır

Soru 7

```
def ters_cevir(liste):  
    sonuc = []  
    for x in liste:  
        sonuc.insert(0, x)  
    return sonuc
```

insert(0, x) x'i listenin başına ekler

Ek görev

7. fonksiyonun $O(n)$ çalışan bir halini kağıda yazın.

İpucu olarak insert(0, x) yerine append kullanmanın bir yolunu arayın. En iyi cevabı projeksiyonda birlikte çalıştıracağız.

GELECEK HAFTA

Bağlı liste (*linked list*)

Başa eklerken kimseyi
kaydırmamak **olmaz mı?**