

Bölüm 9: Çevrimiçi Algoritmalar

Algoritmalar

Çevrimiçi ve çevrimdışı

Rekabetçi analiz

Kayak kiralama

Sayfalama: FIFO, LRU, OPT

Liste güncelleme

Veri akışı algoritmaları

Rezervuar örnekleme

Artımlı öğrenme

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

GİRİŞ

Öğrenme Çıktıları

Model

Çevrimiçi (online) ve çevrimdışı (offline) problem modelini ayırt etmek; geri alınamaz karar kavramını açıklamak.

Rekabetçi analiz

Rekabet oranını (competitive ratio) tanımlamak; kayak kiralama problemi için $2 - 1/B$ oranını kanıtlamak.

Sayfalama ve listeler

FIFO, LRU ve OPT ile sayfa hatalarını adım adım hesaplamak; LRU'nun k -rekabetçi, MTF'nin 2-rekabetçi olduğunu yorumlamak.

Veri akışları

Hareketli ortalama, çevrimiçi ortanca ve rezervuar örneklemeyi Java ile yazmak; öğ başına zaman ve belleği analiz etmek.

GİRİŞ

Bilgi Olsun Yeter

- Bazen kararlar **tüm bilgiye sahip olunmadan** alınır.
- **Çevrimiçi algoritmalar** (online algorithms):
 - bilginin **adım adım** geldiği durumlarda
 - o ana kadarki bilgiyle **en iyi kararı** vermeye çalışır.
- Verilen karar **geri alınamaz**: sonradan gelen bilgi geçmiş kararı değiştirmez.

Satranç oyuncusu rakibin sonraki hamlesini bilmeden hamle yapar; hamle bir kez yapıldıktan sonra geri alınamaz.



GİRİŞ

Çevrimiçi Algoritmalar

- Bir **labirente** yolculuk yapmaya benzer.
 - Labirentin tamamı görünmez; yalnızca önünüzdeki yolu görürsünüz.
- Her adımda **sınırlı bilgi** ile en iyi yol seçilmeye çalışılır.
- Bilgi akışının **sürekli** olmadığı durumlarda **hızlı karar** vermeyi sağlar.
- Örnekler:
 - Trafik sıkışıklığında hangi yol kullanılacak?
 - Satranç oynarken bir sonraki hamle ne olacak?
 - Anlık trafik yoğunluğuna göre ışıkların ayarlanması.



GİRİŞ

Çevrimiçi ve Çevrimdışı Problem Modeli

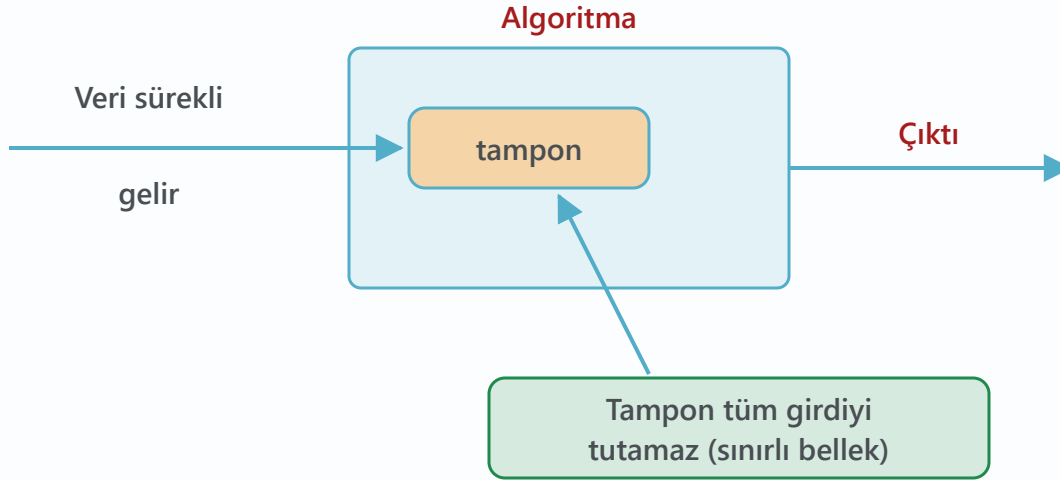
Girdi bir **istek dizisidir**: $\sigma = \sigma_1, \sigma_2, \dots, \sigma_n$. Fark, algoritmanın bu diziyi **ne zaman** gördüğündedir.

	Çevrimdışı (offline)	Çevrimiçi (online)
Girdi	Tamamı baştan bilinir	Parça parça gelir; σ_t gelince σ_{t+1} bilinmez
Karar	Tüm girdiye bakılarak verilir	σ_t geldiği anda verilmeli, geri alınamaz
Örnek	Sıralama, Dijkstra, sırt çantası (DP)	Önbellek yönetimi, kayak kiralama, borsa
En iyi çözüm	OPT hesaplanabilir	OPT ancak geriye dönüp bakınca bilinir
Ölçüt	Zaman / bellek karmaşıklığı	Rekabet oranı : ALG / OPT

Dikkat: "Çevrimiçi" burada *internet* anlamına gelmez; **girdinin zaman içinde açığa çıkması** demektir. Eklemeli sıralama (insertion sort) çevrimiçi çalışabilir, seçimli sıralama (selection sort) çalışamaz.

GİRİŞ

Çevrimiçi Algoritma Modeli (Online Algorithms)



- Veri **sürekli** gelir; algoritma her parçayı geldiğinde işler ve hemen çıktı üretir.
- Bellek (tampon, buffer) **tüm girdiyi tutamaz**: geçmiş ancak özet olarak saklanabilir (toplam, heap, örnek...).
- Bu yüzden her öge için **az iş** (çoğunlukla $O(1)$ ya da $O(\log n)$) ve **az bellek** hedeflenir.

GİRİŞ

Uygulama Alanları

Borsa

Anlık fiyata göre al-sat kararı; gelecek fiyat bilinmez.

Önbellek

İşlemci, işletim sistemi, web: hangi sayfa bellekte kalsın?

Yönlendirme

Bağlantıları sürekli değişen ağda en kısa yol.

Kaynak tahsisi

Bant genişliği ve CPU zamanının dinamik paylaşımı.

Zamanlama

İşler geldikçe makinelere atanır (iş çizelgeleme).

Reklam ve öneri

Kullanıcı geldiğinde hangi reklam / ürün gösterilsin?

Makine öğrenmesi

Model her yeni örnekle güncellenir (artımlı öğrenme).

Güvenlik

Dolandırıcılık ve DDoS saldırılarını akış üzerinde yakalama.

Otonom araçlar

Algılayıcı verisi geldikçe gerçek zamanlı karar.

Ortak nokta: ortam **dinamik**, karar **gerçek zamanlı**, geleceğe dair bilgi **eksik**.

Avantajlar ve Dezavantajlar

Avantajlar

- + **Dinamik** ortamlarda çalışır; girdinin bitmesini beklemez.
- + **Gerçek zamanlı** karar verir; gecikme düşüktür.
- + Az bellekle çok büyük, hatta **sonsuz** akışları işleyebilir.
- + Rekabet oranı ile **en kötü durum garantisi** verilebilir.

Dezavantajlar

- İlk adımlarda **yanlış kararlar** alınabilir ve sonraki adımları etkiler.
- Satrançta yanlış bir hamle oyunu kaybettirebilir.
- Her zaman **en iyi global çözümü** bulmayı garanti etmez.
- Başlangıçtaki girdiye duyarlıdır; tasarımı ve analizi daha karmaşıktır.

Çevrimiçi Algoritma Tasarım Yaklaşımları

Açgözlü (greedy)

Geleni o an en iyi görünen biçimde karşılar; küresel etkiyi düşünmez. Örnek: istek geldiğinde en yakın boş sunucuya atama.

Özetleme

Geçmişi tümüyle saklamak yerine küçük bir özet tutar: toplam, iki heap, örneklem. Örnek: hareketli ortalama, çevrimiçi ortanca.

Uyarlanır (adaptive)

Hatalarından öğrenir, davranışını zamanla değiştirir. Örnek: LRU, öne taşı (MTF), artımlı öğrenme.

Deterministik

Aynı girdiye hep aynı kararı verir; kötü niyetli bir girdi dizisi (düşman) onu kolayca tuzağa düşürebilir.

Rastgele (randomized)

Kararlarına rastlantı katar; düşman ne yapacağını önceden bilemez. Örnek: rezervuar örnekleme, rastgele işaretleme.

Rekabetçi Analiz (Competitive Analysis)

- Çevrimiçi algoritma ALG, her girdide **her şeyi önceden bilen** en iyi çevrimdışı algoritma OPT ile karşılaştırılır.
- ALG, **c-rekabetçidir** (c-competitive), eğer her istek dizisi σ için

$$\text{ALG}(\sigma) \leq c \cdot \text{OPT}(\sigma) + \alpha$$

koşulu sağlanır (α : σ 'dan bağımsız sabit; $\alpha = 0$ ise **kesin** rekabetçi).

- En küçük böyle c değerine **rekabet oranı** denir: $c = \sup_{\sigma} \text{ALG}(\sigma) / \text{OPT}(\sigma)$.
- Oran ne kadar **1'e yakınsa**, gelecek bilgisinin eksikliği o kadar az zarar verir.

Kötümser bakış

Analiz **en kötü girdi** üzerinedir: ortalama ya da olasılık varsayımı yoktur.

Maliyet problemi

Burada maliyet en aza indirilir ($c \geq 1$). Kazanç problemlerinde oran OPT / ALG olarak tanımlanır.

Düşman (Adversary) Modeli

Rekabet oranını **alttan sınırlamak** için algoritmaya karşı oynayan bir düşman düşünülür: girdiyi o üretir ve OPT'yi o çalıştırır.

Deterministik ALG

Düşman algoritmanın kodunu bilir; her adımda ALG'nin **en çok zarar göreceği** isteği üretir.

Habersiz düşman

(oblivious) Rastgele algoritmanın yazı-tura sonuçlarını göremez; diziyi baştan seçer. Beklenen ALG maliyeti ölçülür.

Uyarlanır düşman

(adaptive) ALG'nin rastgele kararlarını görerek sonraki isteği seçer; rastgeleliğin faydası azalır.

Kanıt kalıbı: "Hiçbir deterministik algoritma c'den iyi olamaz" demek için, her ALG'ye karşı $ALG(\sigma) / OPT(\sigma) \geq c$ yapan bir σ dizisi kurulur. Sonraki örneklerde (kayak kiralama, sayfalama) bu kalıp kullanılacak.

Kayak Kiralama Problemi (Ski Rental)

- Kayak takımı **günlüğü 1 TL'ye kiralanabilir** ya da **B TL'ye satın alınabilir**.
- Kaç gün kayak yapılacağı (d) **bilinmez**: her sabah "bugün de kayacak mıyım?" sorusu yanıtlanır. Hava bozarsa sezon biter.
- Her gün karar: **bugün de kirala** mı, yoksa **satın al** mı?
- Geleceği bilen OPT kolaydır: **$\text{OPT}(d) = \min(d, B)$** .

Hep kirala

$d \rightarrow \infty$ iken maliyet d, $\text{OPT} = B$: oran **sınırsız**.

İlk gün satın al

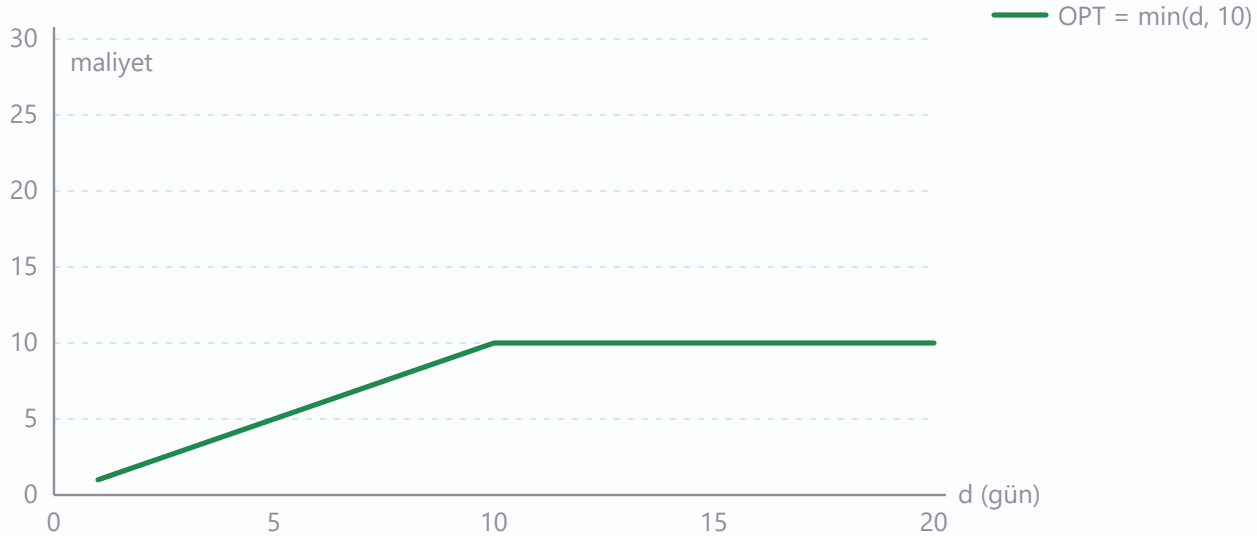
$d = 1$ iken maliyet B, $\text{OPT} = 1$: oran **B**.

Başa baş (break-even)

$B - 1$ gün kirala, B. gün satın al: oran **$2 - 1/B$** .

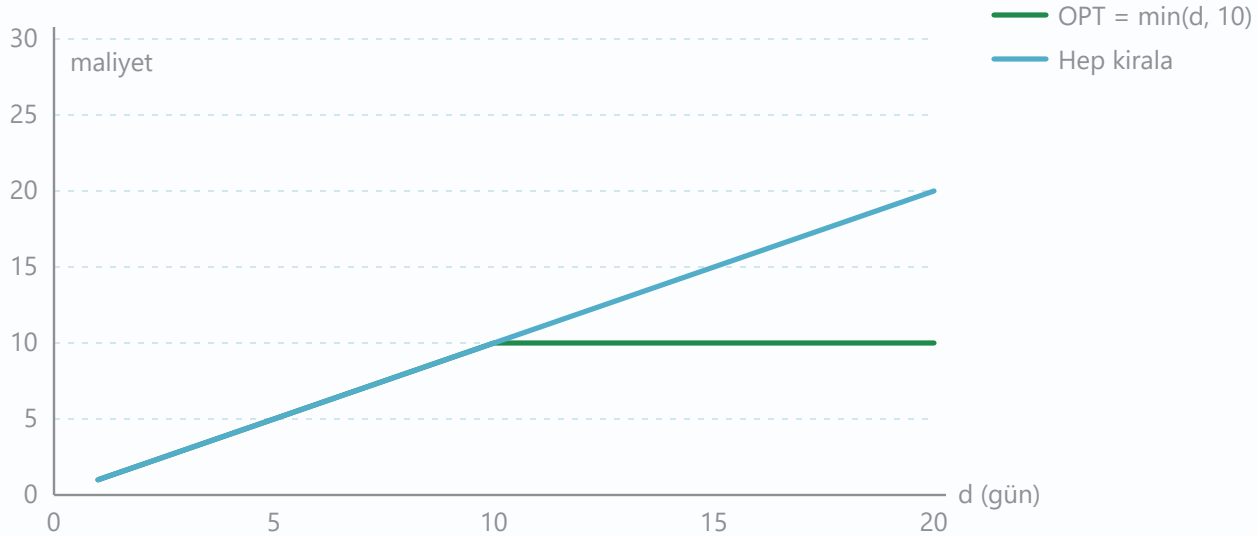
Aynı yapı: bulut sunucusunu saatlik mi kiralayalım, yıllık mı alalım; TCP bağlantısını açık mı turalım, kapatalım mı.

Stratejilerin Maliyeti (B = 10)



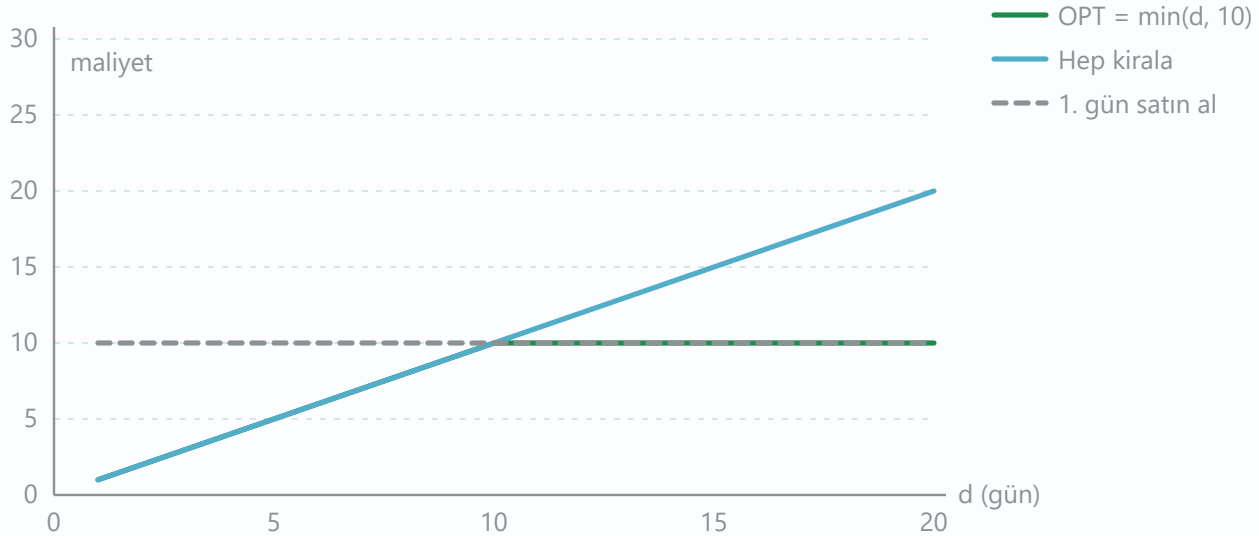
Geleceği bilen **OPT**: $d < 10$ ise hep kiralar (maliyet d), $d \geq 10$ ise ilk gün satın alır (maliyet 10).

Stratejilerin Maliyeti (B = 10)



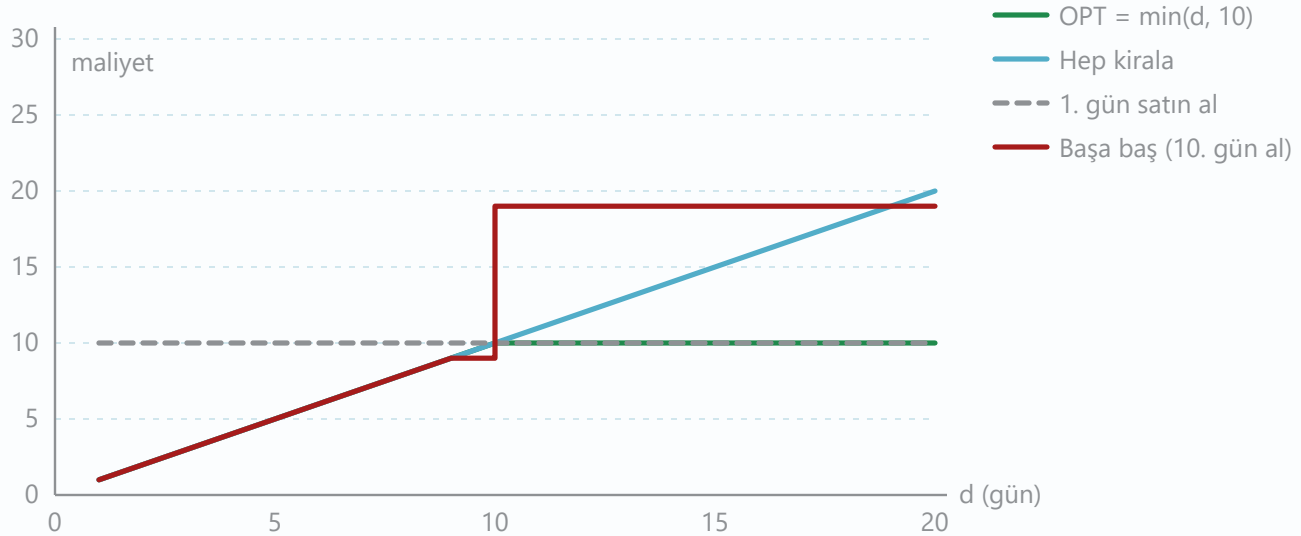
Hep kirala: maliyet d. d büyüdükçe OPT = 10 sabitken maliyet sınırsız artar → rekabetçi değil.

Stratejilerin Maliyeti (B = 10)



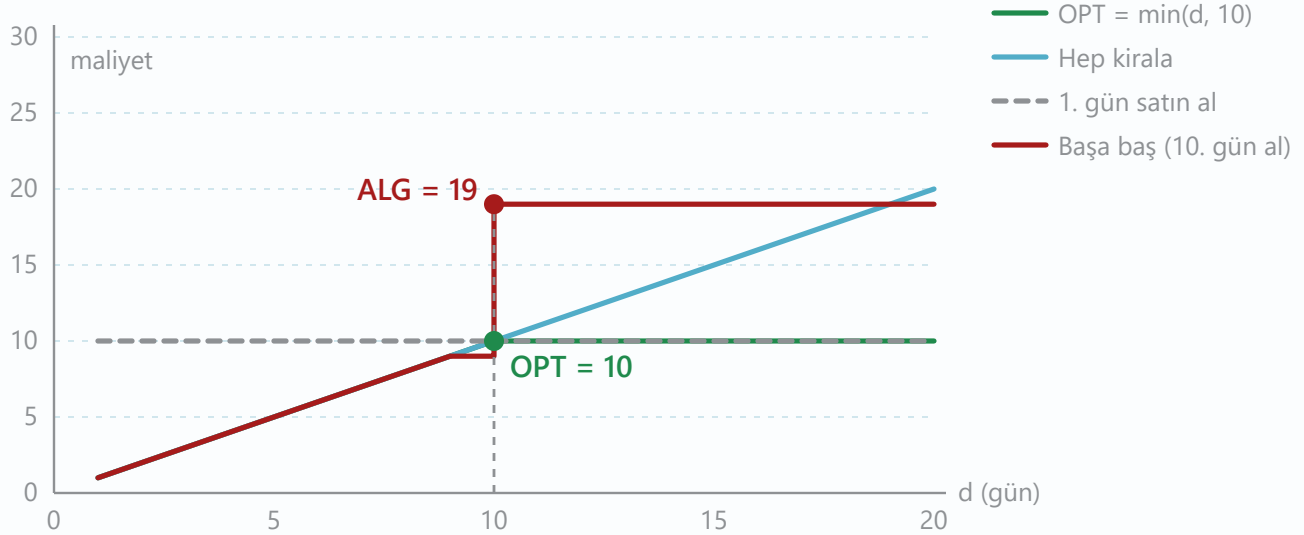
İlk gün satın al: maliyet hep 10. Ama $d = 1$ ise $OPT = 1 \rightarrow$ oran $10 = B$. Kötü.

Stratejilerin Maliyeti (B = 10)



Başa baş: 9 gün kirala, 10. gün satın al. $d < 10$ iken OPT ile aynı; $d \geq 10$ iken maliyet $9 + 10 = 19$.

Stratejilerin Maliyeti (B = 10)



En kötü durum $d = 10$: $ALG = 19$, $OPT = 10 \rightarrow$ oran **$1,9 = 2 - 1/B$** . Hiçbir d bundan kötü değildir.

Neden Daha İyisi Yok? Alt Sınır

Her deterministik strateji tek bir sayı ile tanımlanır: **k** = satın alma günü.

- **Düşman**: satın aldığınız günden sonra hava bozar, yani $d = k$.
- $ALG = (k - 1) \text{ kira} + B$, $OPT = \min(k, B)$.
- $k < B$: oran $(k - 1 + B) / k \geq (2B - 1) / B$.
- $k \geq B$: oran $(k - 1 + B) / B \geq (2B - 1) / B$.
- En iyisi **k = B**: oran **$2 - 1/B$** .

Rastgele seçilen k ile beklenen oran **$e / (e - 1)$** $\approx 1,58$ 'e iner (Karlin vd., 1994).

k	kira	ALG	OPT	oran
1	0	10	1	10
2	1	11	2	5,5
5	4	14	5	2,8
9	8	18	9	2
10	9	19	10	1,9
11	10	20	10	2
15	14	24	10	2,4
20	19	29	10	2,9

$B = 10$; $d = k$ (düşmanın seçimi). Tablo, en kötü oranın $k = B$ 'de en küçük olduğunu gösterir.

Deneyin: Kayak Kiralama

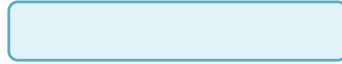
B  10 satın alma günü k  10 gerçek gün d  12

çevrimiçi (ALG)



19 TL

geleceği bilen (OPT)



10 TL

9 gün kira + 10 TL satın alma. Oran $ALG / OPT = 1,90$ · bu k için en kötü oran ($d = k$) = **1,90** · en iyi deterministik: $2 - 1/B = 1,90$

Strateji: $k - 1$ gün kiralama, k . gün hâlâ kayıyorsan satın al. En kötü oran, düşmanın $d = k$ seçtiği durumdur.

Sayfalama Problemi (Paging)



- Hızlı bellek (önbellek, cache) **k sayfa** tutar; yavaş bellek sınırsızdır.
- İstenen sayfa önbellekteyse **isabet** (hit, maliyet 0); değilse **sayfa hatası** (page fault, maliyet 1): sayfa yüklenir.
- Önbellek doluysa **hangi sayfa çıkarılacak?** Gelecek istekler bilinmez: tipik bir çevrimiçi problem.
- Amaç: toplam **sayfa hatası sayısını** en aza indirmek.

Sayfa Çıkarma Politikaları

Politika	Hangi sayfayı çıkarır?	Çevrimiçi?	Rekabet oranı
FIFO	Önbelleğe en önce yüklenen sayfayı (kuyruk)	evet	k
LRU	En uzun süredir kullanılmayan sayfayı (least recently used)	evet	k
FWF	Hata olunca ve önbellek doluysa hepsini boşaltır (flush when full)	evet	k
LFU	En az sıklıkla kullanılan sayfayı (least frequently used)	evet	sınırsız
LIFO	En son yüklenen sayfayı	evet	sınırsız
OPT (MIN)	Bir sonraki kullanımı en uzak gelecekte olanı (Belady, 1966)	hayır	1

OPT (longest forward distance) geleceği bilmeyi gerektirir; uygulanamaz ama **karşılaştırma ölçütü** olarak kullanılır. Kanıtlanabilir biçimde en az sayfa hatasını verir.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7												
Ç2													
Ç3													
Sonuç	H												

FIFO · $k = 3$ · sayfa hatası: 1

İstek 1: sayfa **7** yok → **sayfa hatası**; boş çerçeve Ç1'e yüklenir. Toplam hata: 1.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7											
Ç2		0											
Ç3													
Sonuç	H	H											

FIFO · $k = 3$ · sayfa hatası: 2

İstek 2: sayfa **0** yok → **sayfa hatası**; boş çerçeve Ç2'e yüklenir. Toplam hata: 2.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7										
Ç2		0	0										
Ç3			1										
Sonuç	H	H	H										

FIFO · $k = 3$ · sayfa hatası: 3

İstek 3: sayfa **1** yok → **sayfa hatası**; boş çerçeve Ç3'e yüklenir. Toplam hata: 3.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2									
Ç2		0	0	0									
Ç3			1	1									
Sonuç	H	H	H	H									

FIFO · $k = 3$ · sayfa hatası: 4

İstek 4: sayfa **2** yok → **sayfa hatası**; en önce yüklenen **7** çıkarılır. Toplam hata: 4.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2								
Ç2		0	0	0	0								
Ç3			1	1	1								
Sonuç	H	H	H	H	✓								

FIFO · $k = 3$ · sayfa hatası: 4

İstek 5: sayfa **0** önbellekte → **isabet**. FIFO sırası **değişmez**. Toplam hata: 4.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2							
Ç2		0	0	0	0	3							
Ç3			1	1	1	1							
Sonuç	H	H	H	H	✓	H							

FIFO · $k = 3$ · sayfa hatası: 5

İstek 6: sayfa **3** yok → **sayfa hatası**; en önce yüklenen **0** çıkarılır. Toplam hata: 5.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2						
Ç2		0	0	0	0	3	3						
Ç3			1	1	1	1	0						
Sonuç	H	H	H	H	✓	H	H						

FIFO · $k = 3$ · sayfa hatası: 6

İstek 7: sayfa **0** yok → **sayfa hatası**; en önce yüklenen **1** çıkarılır. Toplam hata: 6.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4					
Ç2		0	0	0	0	3	3	3					
Ç3			1	1	1	1	0	0					
Sonuç	H	H	H	H	✓	H	H	H					

FIFO · $k = 3$ · sayfa hatası: 7

İstek 8: sayfa **4** yok → **sayfa hatası**; en önce yüklenen **2** çıkarılır. Toplam hata: 7.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4				
Ç2		0	0	0	0	3	3	3	2				
Ç3			1	1	1	1	0	0	0				
Sonuç	H	H	H	H	✓	H	H	H	H				

FIFO · $k = 3$ · sayfa hatası: 8

İstek 9: sayfa **2** yok → **sayfa hatası**; en önce yüklenen **3** çıkarılır. Toplam hata: 8.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4			
Ç2		0	0	0	0	3	3	3	2	2			
Ç3			1	1	1	1	0	0	0	3			
Sonuç	H	H	H	H	✓	H	H	H	H	H			

FIFO · $k = 3$ · sayfa hatası: 9

İstek 10: sayfa **3** yok → **sayfa hatası**; en önce yüklenen **0** çıkarılır. Toplam hata: 9.

FIFO Adım Adım

FIFO: ön bellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0		
Ç2		0	0	0	0	3	3	3	2	2	2		
Ç3			1	1	1	1	0	0	0	3	3		
Sonuç	H	H	H	H	✓	H	H	H	H	H	H		

FIFO · $k = 3$ · sayfa hatası: 10

İstek 11: sayfa **0** yok → **sayfa hatası**; en önce yüklenen **4** çıkarılır. Toplam hata: 10.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0	0	
Ç2		0	0	0	0	3	3	3	2	2	2	2	
Ç3			1	1	1	1	0	0	0	3	3	3	
Sonuç	H	H	H	H	✓	H	H	H	H	H	H	✓	

FIFO · $k = 3$ · sayfa hatası: 10

İstek 12: sayfa **3** önbellekte → **isabet**. FIFO sırası **değişmez**. Toplam hata: 10.

FIFO Adım Adım

FIFO: önbellek bir **kuyruk** gibi davranır; en önce gelen ilk çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0	0	0
Ç2		0	0	0	0	3	3	3	2	2	2	2	2
Ç3			1	1	1	1	0	0	0	3	3	3	3
Sonuç	H	H	H	H	✓	H	H	H	H	H	H	✓	✓

FIFO · $k = 3$ · sayfa hatası: 10

İstek 13: sayfa **2** önbellekte → **isabet**. FIFO sırası **değişmez**. Toplam hata: 10.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7												
Ç2													
Ç3													
Sonuç	H												

LRU · $k = 3$ · sayfa hatası: 1

İstek 1: sayfa **7** yok → **sayfa hatası**; boş çerçeve Ç1'e yüklenir. Toplam hata: 1.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7											
Ç2		0											
Ç3													
Sonuç	H	H											

LRU · $k = 3$ · sayfa hatası: 2

İstek 2: sayfa **0** yok → **sayfa hatası**; boş çerçeve Ç2'e yüklenir. Toplam hata: 2.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7										
Ç2		0	0										
Ç3			1										
Sonuç	H	H	H										

LRU · $k = 3$ · sayfa hatası: 3

İstek 3: sayfa **1** yok → **sayfa hatası**; boş çerçeve Ç3'e yüklenir. Toplam hata: 3.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2									
Ç2		0	0	0									
Ç3			1	1									
Sonuç	H	H	H	H									

LRU · $k = 3$ · sayfa hatası: 4

İstek 4: sayfa **2** yok → **sayfa hatası**; en uzun süredir kullanılmayan **7** çıkarılır. Toplam hata:
4.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2								
Ç2		0	0	0	0								
Ç3			1	1	1								
Sonuç	H	H	H	H	✓								

LRU · $k = 3$ · sayfa hatası: 4

İstek 5: sayfa **0** önbellekte → **isabet**. 0 artık en yeni kullanılan sayfa. Toplam hata: 4.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2							
Ç2		0	0	0	0	0							
Ç3			1	1	1	3							
Sonuç	H	H	H	H	✓	H							

LRU · $k = 3$ · sayfa hatası: 5

İstek 6: sayfa **3** yok → **sayfa hatası**; en uzun süredir kullanılmayan **1** çıkarılır. Toplam hata: 5.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2						
Ç2		0	0	0	0	0	0						
Ç3			1	1	1	3	3						
Sonuç	H	H	H	H	$\checkmark$	H	$\checkmark$						

LRU · $k = 3$ · sayfa hatası: 5

İstek 7: sayfa **0** önbellekte → **isabet**. 0 artık en yeni kullanılan sayfa. Toplam hata: 5.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4					
Ç2		0	0	0	0	0	0	0					
Ç3			1	1	1	3	3	3					
Sonuç	H	H	H	H	✓	H	✓	H					

LRU · $k = 3$ · sayfa hatası: 6

İstek 8: sayfa **4** yok → **sayfa hatası**; en uzun süredir kullanılmayan **2** çıkarılır. Toplam hata: 6.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4				
Ç2		0	0	0	0	0	0	0	0				
Ç3			1	1	1	3	3	3	2				
Sonuç	H	H	H	H	$\checkmark$	H	$\checkmark$	H	H				

LRU · $k = 3$ · sayfa hatası: 7

İstek 9: sayfa **2** yok → **sayfa hatası**; en uzun süredir kullanılmayan **3** çıkarılır. Toplam hata:

7.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4			
Ç2		0	0	0	0	0	0	0	0	3			
Ç3			1	1	1	3	3	3	2	2			
Sonuç	H	H	H	H	✓	H	✓	H	H	H			

LRU · $k = 3$ · sayfa hatası: 8

İstek 10: sayfa **3** yok → **sayfa hatası**; en uzun süredir kullanılmayan **0** çıkarılır. Toplam hata: 8.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0		
Ç2		0	0	0	0	0	0	0	0	3	3		
Ç3			1	1	1	3	3	3	2	2	2		
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H		

LRU · $k = 3$ · sayfa hatası: 9

İstek 11: sayfa **0** yok → **sayfa hatası**; en uzun süredir kullanılmayan **4** çıkarılır. Toplam hata: 9.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0	0	
Ç2		0	0	0	0	0	0	0	0	3	3	3	
Ç3			1	1	1	3	3	3	2	2	2	2	
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H	✓	

LRU · $k = 3$ · sayfa hatası: 9

İstek 12: sayfa **3** önbellekte → **isabet**. 3 artık en yeni kullanılan sayfa. Toplam hata: 9.

LRU Adım Adım

LRU: her sayfanın **son kullanım zamanı** izlenir; en eskisi çıkar. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0	0	0
Ç2		0	0	0	0	0	0	0	0	3	3	3	3
Ç3			1	1	1	3	3	3	2	2	2	2	2
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H	✓	✓

LRU · $k = 3$ · sayfa hatası: 9

İstek 13: sayfa **2** önbellekte → **isabet**. 2 artık en yeni kullanılan sayfa. Toplam hata: 9.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7												
Ç2													
Ç3													
Sonuç	H												

OPT · $k = 3$ · sayfa hatası: 1

İstek 1: sayfa **7** yok → **sayfa hatası**; boş çerçeve Ç1'e yüklenir. Toplam hata: 1.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7											
Ç2		0											
Ç3													
Sonuç	H	H											

OPT · $k = 3$ · sayfa hatası: 2

İstek 2: sayfa **0** yok → **sayfa hatası**; boş çerçeve Ç2'e yüklenir. Toplam hata: 2.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2
0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7										
Ç2		0	0										
Ç3			1										
Sonuç	H	H	H										

OPT · $k = 3$ · sayfa hatası: 3

İstek 3: sayfa **1** yok → **sayfa hatası**; boş çerçeve Ç3'e yüklenir. Toplam hata: 3.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2									
Ç2		0	0	0									
Ç3			1	1									
Sonuç	H	H	H	H									

OPT · $k = 3$ · sayfa hatası: 4

İstek 4: sayfa **2** yok → **sayfa hatası**; **7** bir daha hiç istenmiyor → çıkarılır. Toplam hata: 4.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2								
Ç2		0	0	0	0								
Ç3			1	1	1								
Sonuç	H	H	H	H	✓								

OPT · $k = 3$ · sayfa hatası: 4

İstek 5: sayfa **0** önbellekte → **isabet**. Toplam hata: 4.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2							
Ç2		0	0	0	0	0							
Ç3			1	1	1	3							
Sonuç	H	H	H	H	✓	H							

OPT · $k = 3$ · sayfa hatası: 5

İstek 6: sayfa **3** yok → **sayfa hatası**; **1** bir daha hiç istenmiyor → çıkarılır. Toplam hata: 5.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2						
Ç2		0	0	0	0	0	0						
Ç3			1	1	1	3	3						
Sonuç	H	H	H	H	✓	H	✓						

OPT · $k = 3$ · sayfa hatası: 5

İstek 7: sayfa **0** önbellekte → **isabet**. Toplam hata: 5.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	2					
Ç2		0	0	0	0	0	0	4					
Ç3			1	1	1	3	3	3					
Sonuç	H	H	H	H	$\checkmark$	H	$\checkmark$	H					

OPT · $k = 3$ · sayfa hatası: 6

İstek 8: sayfa **4** yok → **sayfa hatası**; bir sonraki kullanımı en uzak olan **0** (11. istek) çıkarılır.

Toplam hata: 6.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	2	2				
Ç2		0	0	0	0	0	0	4	4				
Ç3			1	1	1	3	3	3	3				
Sonuç	H	H	H	H	✓	H	✓	H	✓				

OPT · $k = 3$ · sayfa hatası: 6

İstek 9: sayfa **2** önbellekte → **isabet**. Toplam hata: 6.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	2	2	2			
Ç2		0	0	0	0	0	0	4	4	4			
Ç3			1	1	1	3	3	3	3	3			
Sonuç	H	H	H	H	✓	H	✓	H	✓	✓			

OPT · $k = 3$ · sayfa hatası: 6

İstek 10: sayfa **3** önbellekte → **isabet**. Toplam hata: 6.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	2	2	2	2		
Ç2		0	0	0	0	0	0	4	4	4	0		
Ç3			1	1	1	3	3	3	3	3	3		
Sonuç	H	H	H	H	$\checkmark$	H	$\checkmark$	H	$\checkmark$	$\checkmark$	H		

OPT · $k = 3$ · sayfa hatası: 7

İstek 11: sayfa **0** yok → **sayfa hatası**; **4** bir daha hiç istenmiyor → çıkarılır. Toplam hata: 7.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, **✓**: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	2	2	2	2	2	
Ç2		0	0	0	0	0	0	4	4	4	0	0	
Ç3			1	1	1	3	3	3	3	3	3	3	
Sonuç	H	H	H	H	✓	H	✓	H	✓	✓	H	✓	

OPT · $k = 3$ · sayfa hatası: 7

İstek 12: sayfa **3** önbellekte → **isabet**. Toplam hata: 7.

OPT Adım Adım

OPT: geleceği bilir; bir sonraki kullanımı **en uzak** olan sayfayı çıkarır. İstek dizisi 7 0 1 2 0 3 0 4 2 3 0 3 0 4 2 3 0 3 2, $k = 3$. **H**: hata, $\checkmark$: isabet.

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	2	2	2	2	2	2
Ç2		0	0	0	0	0	0	4	4	4	0	0	0
Ç3			1	1	1	3	3	3	3	3	3	3	3
Sonuç	H	H	H	H	$\checkmark$	H	$\checkmark$	H	$\checkmark$	$\checkmark$	H	$\checkmark$	$\checkmark$

OPT · $k = 3$ · sayfa hatası: 7

İstek 13: sayfa **2** önbellekte → **isabet**. Toplam hata: 7.

Üç Politikanın Karşılaştırması

Politika	Sayfa hatası	İsabet	ALG / OPT	Gerekli bilgi
FIFO	10	3	1,43	Yükleme sırası (kuyruk)
LRU	9	4	1,29	Son kullanım zamanı
OPT	7	6	1	Tüm gelecek istekler

- İlk $k = 3$ hata **kaçınılmazdır** (zorunlu / soğuk hatalar, compulsory misses): her politika öder.
- LRU, **yerellik** (locality of reference) olan dizilerde iyidir: yakın geçmişte kullanılan sayfa yakında yine kullanılır. Burada 0 sayfası sık istendiği için FIFO onu gereksiz yere çıkarır.
- Gerçek sistemlerde LRU'ya yakın, ucuz yaklaşımlar kullanılır: **saat algoritması** (clock / ikinci şans).

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
▶   var onb = new LinkedList<Integer>();  
▶   int hata = 0;  
4   for (int sayfa : istekler) {  
5       if (onb.remove((Integer) sayfa)) {  
6           onb.addLast(sayfa); // en yeni yap  
7       } else {  
8           hata++;  
9           if (onb.size() == k)  
10              onb.removeFirst(); // en eskiyi at  
11              onb.addLast(sayfa);  
12          }  
13      }  
14      return hata;  
15 }
```

istek:

1

2

3

1

4

2

onb:

en eski

en yeni

Önbellek boş, **hata = 0**. Liste başı en eski, sonu en yeni kullanılan sayfa.

Adım 1 / 19

DEĞİŞKENLER

k = 3

sayfa = -

hata = 0

onb = []

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11                onb.addLast(sayfa);  
12            }  
13        }  
14        return hata;  
15    }
```

istek:



onb:



en eski

en yeni

Sayfa 1 listede yok: **remove** false → else dalı.

Adım 2 / 19

DEĞİŞKENLER

k = 3

sayfa = 1

hata = 0

onb = []

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             ▶ hata++;  
9             ▶ if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11                 onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

hata = 1 . Önbellekte yer var (boyut $0 < 3$)
→ çıkarma yok.

Adım 3 / 19

DEĞİŞKENLER

k = 3

sayfa = 1

hata = 1

onb = []

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11         }  
12     }  
13 }  
14 return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 1 sona eklenir.

Adım 4 / 19

DEĞİŞKENLER

k = 3

sayfa = 1

hata = 1

onb = [1]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11            onb.addLast(sayfa);  
12        }  
13    }  
14    return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 2 listede yok: **remove** false → else dalı.

Adım 5 / 19

DEĞİŞKENLER

k = 3

sayfa = 2

hata = 1

onb = [1]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             ▶ hata++;  
9             ▶ if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11                 onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

hata = 2 . Önbellekte yer var (boyut 1 < 3)
→ çıkarma yok.

Adım 6 / 19

DEĞİŞKENLER

k = 3

sayfa = 2

hata = 2

onb = [1]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11             ▶ onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 2 sona eklenir.

Adım 7 / 19

DEĞİŞKENLER

k = 3

sayfa = 2

hata = 2

onb = [1, 2]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11            onb.addLast(sayfa);  
12        }  
13    }  
14    return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 3 listede yok: **remove** false → else dalı.

Adım 8 / 19

DEĞİŞKENLER

k = 3

sayfa = 3

hata = 2

onb = [1, 2]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             ▶ hata++;  
9             ▶ if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11                 onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

hata = 3. Önbellekte yer var (boyut 2 < 3)
→ çıkarma yok.

Adım 9 / 19

DEĞİŞKENLER

k = 3

sayfa = 3

hata = 3

onb = [1, 2]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11             ▶ onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 3 sona eklenir.

Adım 10 / 19

DEĞİŞKENLER

k = 3

sayfa = 3

hata = 3

onb = [1, 2, 3]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11            onb.addLast(sayfa);  
12        }  
13    }  
14    return hata;  
15 }
```

istek: 1 2 3 1 4 2

onb: 2 3
en eski en yeni

Sayfa 1 listede: **remove** true döner ve onu listeden çıkarır → **isabet**.

Adım 11 / 19

DEĞİŞKENLER

k = 3 sayfa = 1 hata = 3

onb = [2, 3]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             ▶ onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11                 onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 1 sona eklenir: artık **en yeni** kullanılan.

Adım 12 / 19

DEĞİŞKENLER

k = 3

sayfa = 1

hata = 3

onb = [2, 3, 1]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11                onb.addLast(sayfa);  
12            }  
13        }  
14        return hata;  
15    }
```

istek:



onb:



en eski

en yeni

Sayfa 4 listede yok: **remove** false → else dalı.

Adım 13 / 19

DEĞİŞKENLER

k = 3

sayfa = 4

hata = 3

onb = [2, 3, 1]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             ▶ hata++;  
9             ▶ if (onb.size() == k)  
10                ▶ onb.removeFirst(); // en eskiyi at  
11             onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



çıkan: 2

hata = 4 . Önbellek dolu → baştaki (en eski kullanılan) **2** çıkarılır.

Adım 14 / 19

DEĞİŞKENLER

k = 3

sayfa = 4

hata = 4

onb = [3, 1]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11         }  
12     }  
13 }  
14 return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 4 sona eklenir.

Adım 15 / 19

DEĞİŞKENLER

k = 3

sayfa = 4

hata = 4

onb = [3, 1, 4]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11            onb.addLast(sayfa);  
12        }  
13    }  
14    return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 2 listede yok: **remove** false → else dalı.

Adım 16 / 19

DEĞİŞKENLER

k = 3

sayfa = 2

hata = 4

onb = [3, 1, 4]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             ▶ hata++;  
9             ▶ if (onb.size() == k)  
10                ▶ onb.removeFirst(); // en eskiyi at  
11             onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

çıkan: 3

hata = 5. Önbellek dolu → baştaki (en eski kullanılan) **3** çıkarılır.

Adım 17 / 19

DEĞİŞKENLER

k = 3

sayfa = 2

hata = 5

onb = [1, 4]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                onb.removeFirst(); // en eskiyi at  
11         }  
12     }  
13 }  
14 return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Sayfa 2 sona eklenir.

Adım 18 / 19

DEĞİŞKENLER

k = 3

sayfa = 2

hata = 5

onb = [1, 4, 2]

LRU: Kodu Adım Adım İzle

```
1 int lruHata(int[] istekler, int k) {  
2     var onb = new LinkedList<Integer>();  
3     int hata = 0;  
4     for (int sayfa : istekler) {  
5         if (onb.remove((Integer) sayfa)) {  
6             onb.addLast(sayfa); // en yeni yap  
7         } else {  
8             hata++;  
9             if (onb.size() == k)  
10                 onb.removeFirst(); // en eskiyi at  
11             onb.addLast(sayfa);  
12         }  
13     }  
14     return hata;  
15 }
```

istek:



onb:



en eski

en yeni

Dizi bitti: 6 istekte **5 sayfa hatası**, 1 isabet.

remove listeyi taradığı için her istek $O(k)$;
HashMap + çift bağlı liste ile $O(1)$ olur.

Adım 19 / 19

DEĞİŞKENLER

k = 3

sayfa = -

hata = 5

onb = [1, 4, 2]

LRU Önbellegi: O(1) Uygulama

```
class LruOnbellek<A, D>
    extends LinkedHashMap<A, D> {
    private final int k;

    LruOnbellek(int k) {
        super(16, 0.75f, true); // erişim sırası
        this.k = k;
    }

    @Override
    protected boolean removeEldestEntry(
        Map.Entry<A, D> enEski) {
        return size() > k;      // k aşılinca at
    }
}
```

- `LinkedHashMap` = hash tablosu + çift bağlı liste.
- `true` (accessOrder): her `get` / `put` ögeyi listenin **sonuna** taşır.
- `removeEldestEntry` : boyut k'yı aşınca **baştaki** (en eski kullanılan) silinir.
- Bul, taşı, sil: hepsi **O(1)**; bellek O(k).

FIFO için aynı sınıf `false` (insertion order) ile kurulur: erişim sırayı değiştirmez.

Teorem: Deterministik Alt Sınır k

- $k + 1$ farklı sayfa yeter. Düşman her adımda ALG'nin önbelleğinde **olmayan** tek sayfayı ister → ALG **her istekte** hata yapar.
- OPT (en uzağı çıkar) bir hata yaptığında, çıkardığı sayfa en az $k - 1$ istek boyunca istenmez → OPT en çok **k istekte bir** hata yapar.
- Sonuç: ALG / OPT → **k** (Sleator ve Tarjan, 1985).

Örnekte ($k = 3$, LRU'ya karşı): LRU 12 hata, OPT 6 hata; ilk 3 zorunlu hata dışarıda tutulursa 9'a 3.

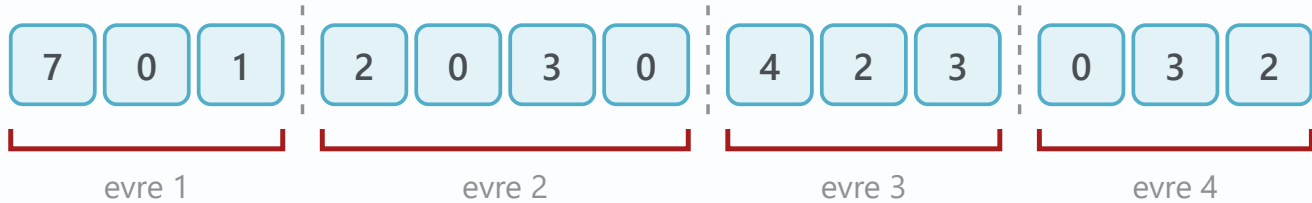
İstek	1	2	3	4	1	2	3	4	1	2	3	4
Ç1	1	1	1	4	4	4	3	3	3	2	2	2
Ç2		2	2	2	1	1	1	4	4	4	3	3
Ç3			3	3	3	2	2	2	1	1	1	4
Sonuç	H	H	H	H	H	H	H	H	H	H	H	H

LRU · $k = 3$ · sayfa hatası: 12

İstek	1	2	3	4	1	2	3	4	1	2	3	4
Ç1	1	1	1	1	1	1	1	1	1	2	2	2
Ç2		2	2	2	2	2	3	3	3	3	3	3
Ç3			3	4	4	4	4	4	4	4	4	4
Sonuç	H	H	H	H	✓	✓	H	✓	✓	H	✓	✓

OPT · $k = 3$ · sayfa hatası: 6

Teorem: LRU (ve FIFO) k-Rekabetçidir



- Diziyi **evrelere** (phase) böl: her evre, **k farklı sayfa** içeren en uzun parça; $(k + 1)$. farklı sayfa yeni evreyi başlatır.
- **LRU, bir evrede en çok k hata yapar:** evrede istenen bir sayfa, evre bitene kadar çıkarılmaz (ondan daha eski kullanılmış en az bir sayfa hep vardır). Her farklı sayfa en çok bir kez hata verir.
- **OPT, ardışık iki evre başına en az 1 hata yapar:** bir evrenin 2. isteğinden sonraki evrenin 1. isteğine kadar $k + 1$ farklı sayfa istenir; k çerçeveye sığmaz.
- m evre için $\text{LRU} \leq k \cdot m$, $\text{OPT} \geq m - 1 \rightarrow \text{LRU} \leq k \cdot \text{OPT} + k$.

Örnek dizide 4 evre: $\text{LRU} = 9 \leq 3 \cdot 4$, $\text{OPT} = 7 \geq 3$. Teori en kötü durumu sınırlar; gerçek dizilerde yerellik sayesinde LRU, OPT'ye çok daha yakındır.

Belady Anomalisi: Daha Çok Bellek, Daha Çok Hata

İstek	1	2	3	4	1	2	5	1	2	3	4	5
Ç1	1	1	1	4	4	4	5	5	5	5	5	5
Ç2		2	2	2	1	1	1	1	1	3	3	3
Ç3			3	3	3	2	2	2	2	2	4	4
Sonuç	H	H	H	H	H	H	H	✓	✓	H	H	✓

FIFO · k = 3 · sayfa hatası: 9

İstek	1	2	3	4	1	2	5	1	2	3	4	5
Ç1	1	1	1	1	1	1	5	5	5	5	4	4
Ç2		2	2	2	2	2	2	1	1	1	1	5
Ç3			3	3	3	3	3	3	2	2	2	2
Ç4				4	4	4	4	4	4	3	3	3
Sonuç	H	H	H	H	✓	✓	H	H	H	H	H	H

FIFO · k = 4 · sayfa hatası: 10

- Dizi: 1 2 3 4 1 2 5 1 2 3 4 5.
- FIFO, k = 3: **9 hata**; k = 4: **10 hata**.
Önbellek büyüdü, hata arttı!
- LRU aynı dizide: k = 3 → 10, k = 4 → 8.
LRU ve OPT bir **yığın algoritmasıdır** (stack algorithm): k çerçevedeki sayfalar, k + 1 çerçevedekilerin alt kümesidir → anomali olmaz.

Rastgele Sayfalama: İşaretleme Algoritması

İşaretleme (Marking)

1. Evre başında tüm sayfaların işaretini kaldır.
2. İstenen sayfayı **işaretle** (isabet ya da hata).
3. Hata olursa **işaretsiz** sayfalardan birini **rastgele** çıkar.
4. Tüm sayfalar işaretliyse yeni evre başlar (adım 1).

Sonuçlar

- Habersiz düşmana karşı beklenen oran **$2H_k$** (Fiat vd., 1991).
- $H_k = 1 + 1/2 + \dots + 1/k \approx \ln k$. $k = 8$ için $H_8 \approx 2,72 \rightarrow$ oran $\approx 5,4$ (deterministikte 8).
- Hiçbir rastgele algoritma **H_k** 'dan iyi olamaz.
- LRU da bir işaretleme algoritmasıdır (en eski işaretsiz seçer).

Rastgelelik, düşmanın "ALG'nin önbelleğinde olmayan sayfayı iste" stratejisini bozar: hangi sayfanın çıktığını bilemez.

Deneyin: Sayfalama Politikaları

istekler

7 0 1 2 0 3 0 4 2 3 0 3 2

k

3

FIFO

LRU

OPT

Belady örneği

İstek	7	0	1	2	0	3	0	4	2	3	0	3	2
Ç1	7	7	7	2	2	2	2	4	4	4	0	0	0
Ç2		0	0	0	0	0	0	0	0	3	3	3	3
Ç3			1	1	1	3	3	3	2	2	2	2	2
Sonuç	H	H	H	H	✓	H	✓	H	H	H	H	✓	✓

LRU gösteriliyor · k = 3 · sayfa hatası: FIFO **10**, LRU **9**, OPT **7**

0–9 arası rakamlar, boşlukla ayrılmış (en çok 16 istek). Turuncu hücre: yeni yüklenen sayfa · yeşil: isabet. Enter = yeniden çalıştır.

Liste Güncelleme Problemi (List Update)

- n öğeli bir **bağlı liste**; i. konumdaki öğeye erişmenin maliyeti **i**'dir (baştan taranır).
- Erişilen öğe, erişimden sonra **ücretsiz** olarak öne doğru taşınabilir.
- İstekler çevrimiçi gelir: hangi öğenin sık isteneceği bilinmez.

Öne taşı (MTF)

Move-to-front: erişilen öğe listenin **başına** alınır.

Yer değiştir

Transpose: erişilen öğe **bir önceki** ile yer değiştirir.

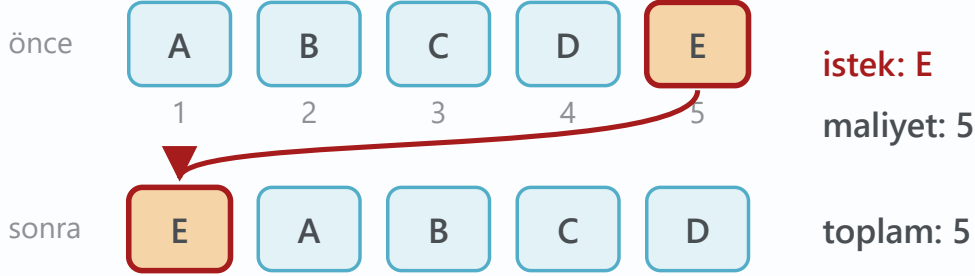
Sıklık sayacı

Frequency count: öğeler erişim sayısına göre **azalan** sırada tutulur.

Uygulamalar: kendini düzenleyen listeler (self-organizing lists), hash tablosu kovalarındaki zincirler, **MTF dönüşümü** ile veri sıkıştırma (bzip2).

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



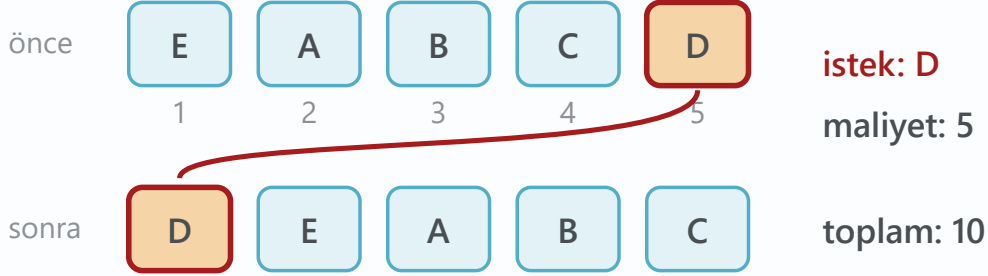
İstek **E**: 5. konumda → maliyet 5; **baş**a taşınır. Toplam: 5.

Adım 1 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	21
Taşıma yok	5	4	5	1	4	5	24
Yer değiştir	5	5	5	1	5	5	26

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



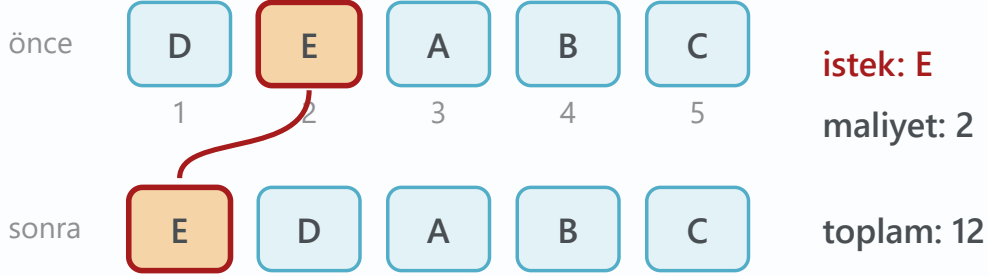
İstek **D**: 5. konumda → maliyet 5; **başta taşınır**. Toplam: 10.

Adım 2 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	21
Taşıma yok	5	4	5	1	4	5	24
Yer değiştir	5	5	5	1	5	5	26

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



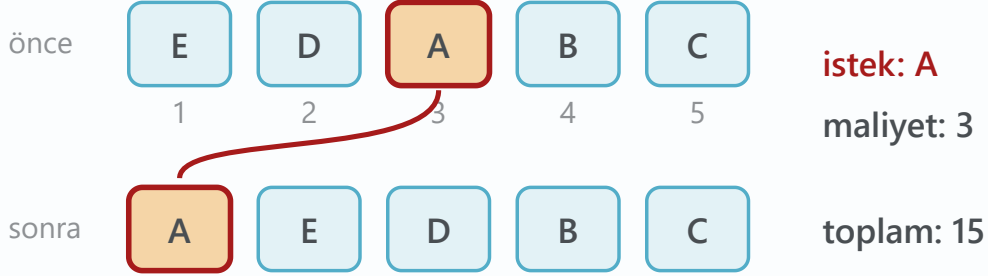
İstek **E**: 2. konumda → maliyet 2; **başına taşınır**. Toplam: 12.

Adım 3 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	21
Taşıma yok	5	4	5	1	4	5	24
Yer değiştir	5	5	5	1	5	5	26

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



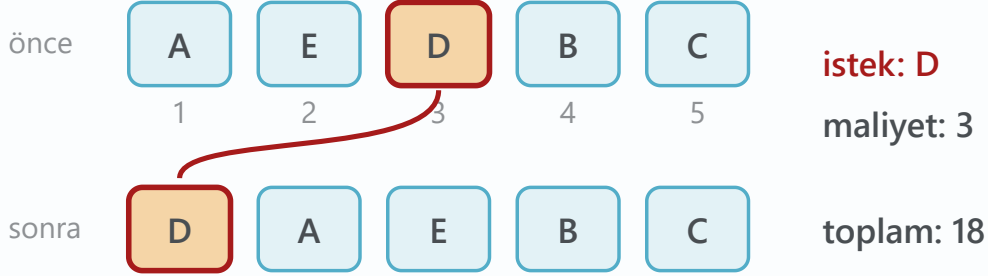
İstek **A**: 3. konumda → maliyet 3; **başta taşınır**. Toplam: 15.

Adım 4 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	21
Taşıma yok	5	4	5	1	4	5	24
Yer değiştir	5	5	5	1	5	5	26

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



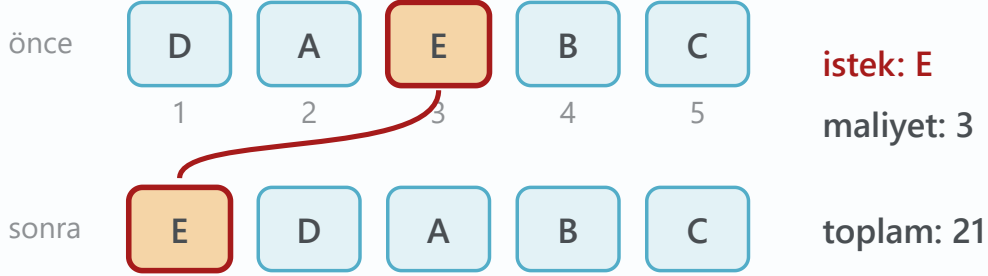
İstek **D**: 3. konumda → maliyet 3; **baş**a taşınır. Toplam: 18.

Adım 5 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	21
Taşıma yok	5	4	5	1	4	5	24
Yer değiştir	5	5	5	1	5	5	26

Öne Taşı (MTF) Adım Adım

Başlangıç listesi A B C D E; istekler **E D E A D E**.



İstek **E**: 3. konumda → maliyet 3; **başta taşınır**. Toplam: 21. Aynı dizide hiç taşımayan liste 24, transpoz (transpose) 26 öder.

Adım 6 / 6

Strateji	E	D	E	A	D	E	Toplam
Öne taşı (MTF)	5	5	2	3	3	3	21
Taşıma yok	5	4	5	1	4	5	24
Yer değiştir	5	5	5	1	5	5	26

Teorem: MTF 2-Rekabetçidir

- **Sleator ve Tarjan (1985):** her istek dizisi σ için $\text{MTF}(\sigma) \leq 2 \cdot \text{OPT}(\sigma)$ (yaklaşık; ücretli takaslar sayılmazsa).
- Kanıt fikri: **potansiyel fonksiyonu** Φ = MTF listesi ile OPT listesi arasındaki **ters sıralı çift** (inversion) sayısı. Her erişimde MTF'nin gerçek maliyeti + $\Delta\Phi \leq 2 \cdot (\text{OPT maliyeti})$ – 1. Bu, amortize analizdir.

Yer değiştir rekabetçi değil

- Son iki öge sırayla istenir: $x_n, x_{n-1}, x_n, \dots$
Her erişim yerini değiştirir ama hep n . ve $(n-1)$. konumda kalır $\rightarrow$ istek başına $\approx n$.
MTF ise ilk ikisinden sonra istek başına 2 öder.

Sıklık sayacı da değil

- Önce x_1 n kez, x_2 $n - 1$ kez, ... istenir;
sonra ters sırada. Sayaç eski sırayı uzun süre korur; maliyet $\Theta(n)$ kat kötüleşebilir.

Veri Akışı (Data Stream) Modeli

- Öğeler $x_1, x_2, \dots$ tek tek gelir; akış çok uzun ya da **sonsuz** olabilir.
- Her öğeye **bir kez** bakılır (tek geçiş); geçmiş tümüyle saklanamaz.
- Her an bir sorunun cevabı hazır olmalı: "şu ana kadarki ortalama / ortanca / bir örneklem nedir?"

Problem	Saklanan özet	Öğeye başına	Bellek
Hareketli ortalama (son k)	Kuyruk + toplam	$O(1)$	$O(k)$
Çevrimiçi ortanca	Max-heap + min-heap	$O(\log n)$	$O(n)$
Rezervuar örnekleme	k öğelik örneklem	$O(1)$	$O(k)$
Akışta örüntü arama	Son m karakter + kayan hash	$O(1)$	$O(m)$
Çalışan ortalama	n ve μ	$O(1)$	$O(1)$

Hareketli Ortalama (Moving Average)

- Akıştaki **son k öğenin** ortalaması: sensör verisini yumuşatma, borsada 50 günlük ortalama.
- Saf yöntem her seferinde k öğeyi toplar: **$O(k)$** .
- Akıllı yöntem: **kayan pencere** (sliding window). Toplamı tut; yeni öğe girerken ekle, en eski çıkarken çıkar: **$O(1)$** .

$$\text{toplamlam} \leftarrow \text{toplamlam} - x_{t-k} + x_t$$
$$\text{ortalama} = \text{toplamlam} / \min(t, k)$$

```
class HareketliOrtalama {  
    private final int k;  
    private final Queue<Integer> pencere  
        = new LinkedList<>();  
    private int toplam = 0;  
  
    HareketliOrtalama(int k) { this.k = k; }  
  
    double sonraki(int deger) {  
        if (pencere.size() == k)  
            toplam -= pencere.poll();  
        pencere.offer(deger);  
        toplam += deger;  
        return toplam / (double) pencere.size();  
    }  
}
```

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4         pencere.offer(deger);  
5         toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

toplam = 0

Yeni değer 4. Pencerede 0 öge var (< 3) → çıkarma yok.

Adım 1 / 17

DEĞİŞKENLER

k = 3

deger = 4

toplam = 0

pencere = []

ÇIKTI

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

4

toplam = 4

4 kuyruğa eklenir, toplama katılır: toplam = 4.

Adım 2 / 17

DEĞİŞKENLER

k = 3

deger = 4

toplam = 4

pencere = [4]

ÇIKTI



Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

4

toplam = 4

Ortalama = $4 / 1 = 4$.

Adım 3 / 17

DEĞİŞKENLER

k = 3

deger = 4

toplam = 4

pencere = [4]

ÇIKTI

4.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4         pencere.offer(deger);  
5         toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

4

toplam = 4

Yeni değer 8. Pencerede 1 öğe var (< 3) → çıkarma yok.

Adım 4 / 17

DEĞİŞKENLER

k = 3

deger = 8

toplam = 4

pencere = [4]

ÇIKTI

4.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2   if (pencere.size() == k)  
3     toplam -= pencere.poll(); // en eski  
4   ▶ pencere.offer(deger);  
5   ▶ toplam += deger;  
6   return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

4

8

toplam = 12

8 kuyruğa eklenir, toplama katılır: toplam = 12.

Adım 5 / 17

DEĞİŞKENLER

k = 3

deger = 8

toplam = 12

pencere = [4, 8]

ÇIKTI

4.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

4

8

toplam = 12

Ortalama = $12 / 2 = 6$.

Adım 6 / 17

DEĞİŞKENLER

k = 3

deger = 8

toplam = 12

pencere = [4, 8]

ÇIKTI

4.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4         pencere.offer(deger);  
5         toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4 8 6 2 10

pencere:

4 8

toplam = 12

Yeni değer 6. Pencerede 2 öğe var (< 3) → çıkarma yok.

Adım 7 / 17

DEĞİŞKENLER

k = 3

deger = 6

toplam = 12

pencere = [4, 8]

ÇIKTI

4.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4 8 6 2 10

pencere:

4 8 6

toplam = 18

6 kuyruğa eklenir, toplama katılır: toplam = 18.

Adım 8 / 17

DEĞİŞKENLER

k = 3

deger = 6

toplam = 18

pencere = [4, 8, 6]

ÇIKTI

4.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

4

8

6

toplam = 18

Ortalama = $18 / 3 = 6$.

Adım 9 / 17

DEĞİŞKENLER

k = 3

deger = 6

toplam = 18

pencere = [4, 8, 6]

ÇIKTI

4.00 6.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4 8 6 2 10

pencere:

4 8 6

toplam = 18

Yeni değer 2. Pencere dolu (boyut 3 = k) → en eski çıkarılacak.

Adım 10 / 17

DEĞİŞKENLER

k = 3

deger = 2

toplam = 18

pencere = [4, 8, 6]

ÇIKTI

4.00 6.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4 8 6 2 10

pencere:

8 6

çıkan: 4

toplam = 14

En eski öge 4 kuyruktan alınır, toplamdan çıkarılır: toplam = 14.

Adım 11 / 17

DEĞİŞKENLER

k = 3

deger = 2

toplam = 14

pencere = [8, 6]

ÇIKTI

4.00 6.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2   if (pencere.size() == k)  
3     toplam -= pencere.poll(); // en eski  
4   ▶ pencere.offer(deger);  
5   ▶ toplam += deger;  
6   return toplam / (double) pencere.size();  
7 }
```

akış:

4 8 6 2 10

pencere:

8 6 2

toplam = 16

2 kuyruğa eklenir, toplama katılır: toplam = 16.

Adım 12 / 17

DEĞİŞKENLER

k = 3

deger = 2

toplam = 16

pencere = [8, 6, 2]

ÇIKTI

4.00 6.00 6.00

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

8

6

2

toplam = 16

Ortalama = $16 / 3 = 5,33$.

Adım 13 / 17

DEĞİŞKENLER

k = 3

deger = 2

toplam = 16

pencere = [8, 6, 2]

ÇIKTI

4.00 6.00 6.00 5.33

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4         pencere.offer(deger);  
5         toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

8

6

2

toplam = 16

Yeni değer 10. Pencere dolu (boyut 3 = k) → en eski çıkarılacak.

Adım 14 / 17

DEĞİŞKENLER

k = 3

deger = 10

toplam = 16

pencere = [8, 6, 2]

ÇIKTI

4.00 6.00 6.00 5.33

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

6

2

çıkan: 8

toplam = 8

En eski öge **8** kuyruktan alınır, toplamdan çıkarılır: toplam = 8.

Adım 15 / 17

DEĞİŞKENLER

k = 3

deger = 10

toplam = 8

pencere = [6, 2]

ÇIKTI

4.00 6.00 6.00 5.33

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

6

2

10

toplam = 18

10 kuyruğa eklenir, toplama katılır: toplam = 18.

Adım 16 / 17

DEĞİŞKENLER

k = 3

deger = 10

toplam = 18

pencere = [6, 2, 10]

ÇIKTI

4.00 6.00 6.00 5.33

Hareketli Ortalama: Kodu Adım Adım İzle

```
1 double sonraki(int deger) {  
2     if (pencere.size() == k)  
3         toplam -= pencere.poll(); // en eski  
4     pencere.offer(deger);  
5     toplam += deger;  
6     return toplam / (double) pencere.size();  
7 }
```

akış:

4

8

6

2

10

pencere:

6

2

10

toplam = 18

Ortalama = $18 / 3 = 6$.

Adım 17 / 17

DEĞİŞKENLER

k = 3

deger = 10

toplam = 18

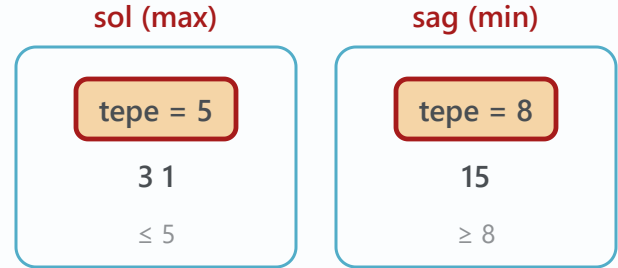
pencere = [6, 2, 10]

ÇIKTI

4.00 6.00 6.00 5.33 6.00

Çevrimiçi Ortanca (Running Median)

- Her yeni öğeden sonra **o ana kadarki ortanca** (median) istenir.
- Her seferinde sıralamak: $O(n \log n)$. Sıralı diziye eklemek: $O(n)$.
- İki heap**: küçük yarı bir **max-heap**'te (sol), büyük yarı bir **min-heap**'te (sag).
- Değişmezler: sol'daki her öğe $\leq$ sag'daki her öğe; $|sol| = |sag|$ ya da $|sol| = |sag| + 1$.
- Ortanca heap tepelerinden **$O(1)$** 'de okunur; ekleme **$O(\log n)$** .



ortanca = 5 (5 öğe)

Sayı adedi tekse ortanca sol'un tepesi; çiftse iki tepenin ortalamasıdır. Java'da max-heap: `new PriorityQueue<((a, b) → b - a)`.

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
  ► if (sol.isEmpty() || x <= sol.peek())  
  ►   sol.offer(x);          // küçük yarı  
4  else  
5    sag.offer(x);          // büyük yarı  
6  if (sol.size() > sag.size() + 1)  
7    sag.offer(sol.poll());  
8  else if (sag.size() > sol.size())  
9    sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5

15

1

3

8

7

sol
(max)

5

sag
(min)

boş

x = 5: sol boş → **sol** heap'ine eklenir.

Adım 1 / 18

DEĞİŞKENLER

x = 5

sol = [5]

sag = []

ortalanca = -

ÇIKTI

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   ▶ if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   ▶ else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5

15

1

3

8

7

sol
(max)

5

sag
(min)

boş

Boyutlar 1 ve 0: denge tamam, taşıma yok.

Adım 2 / 18

DEĞİŞKENLER

x = 5

sol = [5]

sag = []

ortalanca = -

ÇIKTI

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   ▶ if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   ▶ return sol.peek();  
15 }
```

akış:

5

15

1

3

8

7

sol
(max)

5

sag
(min)

boş

ortanca = 5

Öge sayısı tek: ortanca = sol tepesi = 5.

Adım 3 / 18

DEĞİŞKENLER

x = 5

sol = [5]

sag = []

ortanca = 5

ÇIKTI

5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
  ► if (sol.isEmpty() || x <= sol.peek())  
3    sol.offer(x);           // küçük yarı  
  ► else  
  ►    sag.offer(x);         // büyük yarı  
6  if (sol.size() > sag.size() + 1)  
7    sag.offer(sol.poll());  
8  else if (sag.size() > sol.size())  
9    sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12  if (sol.size() == sag.size())  
13    return (sol.peek() + sag.peek()) / 2.0;  
14  return sol.peek();  
15 }
```

akış:

5

15

1

3

8

7

sol
(max)

5

sag
(min)

15

$x = 15$: $15 > 5 \rightarrow$ **sag** heap'ine eklenir.

Adım 4 / 18

DEĞİŞKENLER

 $x = 15$

sol = [5]

sag = [15]

ortalanca = -

ÇIKTI

5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   ▶ if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   ▶ else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5

15

1

3

8

7

sol
(max)

5

sag
(min)

15

Boyutlar 1 ve 1: denge tamam, taşıma yok.

Adım 5 / 18

DEĞİŞKENLER

x = 15

sol = [5]

sag = [15]

ortanca = -

ÇIKTI

5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   if (sol.size() == sag.size())  
13     ▶ return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5

sag
(min)

15

ortanca = 10Öge sayısı çift: ortanca = $(5 + 15) / 2 = 10$.

Adım 6 / 18

DEĞİŞKENLER

x = 15

sol = [5]

sag = [15]

ortanca = 10

ÇIKTI

5.0 10.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
  ► if (sol.isEmpty() || x <= sol.peek())  
  ►   sol.offer(x);           // küçük yarı  
4  else  
5    sag.offer(x);           // büyük yarı  
6  if (sol.size() > sag.size() + 1)  
7    sag.offer(sol.poll());  
8  else if (sag.size() > sol.size())  
9    sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 1

sag
(min)

15

 $x = 1: 1 \leq 5 \rightarrow$ sol heap'ine eklenir.

Adım 7 / 18

DEĞİŞKENLER

x = 1

sol = [5, 1]

sag = [15]

ortanca = -

ÇIKTI

5.0 10.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   ▶ if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   ▶ else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 1

sag
(min)

15

Boyutlar 2 ve 1: denge tamam, taşıma yok.

Adım 8 / 18

DEĞİŞKENLER

x = 1

sol = [5, 1]

sag = [15]

ortanca = -

ÇIKTI

5.0 10.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   ▶ if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   ▶ return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 1

sag
(min)

15

ortanca = 5

Öge sayısı tek: ortanca = sol tepesi = 5.

Adım 9 / 18

DEĞİŞKENLER

x = 1

sol = [5, 1]

sag = [15]

ortanca = 5

ÇIKTI

5.0 10.0 5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
  ▶ if (sol.isEmpty() || x <= sol.peek())  
  ▶   sol.offer(x);           // küçük yarı  
4  else  
5    sag.offer(x);           // büyük yarı  
6  if (sol.size() > sag.size() + 1)  
7    sag.offer(sol.poll());  
8  else if (sag.size() > sol.size())  
9    sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12  if (sol.size() == sag.size())  
13    return (sol.peek() + sag.peek()) / 2.0;  
14  return sol.peek();  
15 }
```

akış:

sol
(max)sag
(min)

$x = 3: 3 \leq 5 \rightarrow$ **sol** heap'ine eklenir.

Adım 10 / 18

DEĞİŞKENLER

x = 3

sol = [5, 3, 1]

sag = [15]

ortalanca = -

ÇIKTI

5.0 10.0 5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   ▶ if (sol.size() > sag.size() + 1)  
7     ▶ sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

3 1

sag
(min)

5 15

Denge bozuldu ($|sol| = 3$, $|sag| = 1$): sol tepesi **5** sag heap'ine taşınır.

Adım 11 / 18

DEĞİŞKENLER

x = 3

sol = [3, 1]

sag = [5, 15]

ortalanca = -

ÇIKTI

5.0 10.0 5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   if (sol.size() == sag.size())  
13     ▶ return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

3 1

sag
(min)

5 15

ortanca = 4Öge sayısı çift: $\text{ortanca} = (3 + 5) / 2 = 4$.

Adım 12 / 18

DEĞİŞKENLER

x = 3

sol = [3, 1]

sag = [5, 15]

ortanca = 4

ÇIKTI

5.0 10.0 5.0 4.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
  ► if (sol.isEmpty() || x <= sol.peek())  
3    sol.offer(x);          // küçük yarı  
  ► else  
  ►    sag.offer(x);        // büyük yarı  
6  if (sol.size() > sag.size() + 1)  
7    sag.offer(sol.poll());  
8  else if (sag.size() > sol.size())  
9    sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12  if (sol.size() == sag.size())  
13    return (sol.peek() + sag.peek()) / 2.0;  
14  return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

3 1

sag
(min)

5 8 15

$x = 8: 8 > 3 \rightarrow$ **sag** heap'ine eklenir.

Adım 13 / 18

DEĞİŞKENLER

x = 8

sol = [3, 1]

sag = [5, 8, 15]

ortalanca = -

ÇIKTI

5.0 10.0 5.0 4.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   ▶ if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   ▶ else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 3 1

sag
(min)

8 15

Denge bozuldu ($|sol| = 2$, $|sag| = 3$): sag tepesi **5** sol heap'ine taşınır.

Adım 14 / 18

DEĞİŞKENLER

x = 8

sol = [5, 3, 1]

sag = [8, 15]

ortalanca = -

ÇIKTI

5.0 10.0 5.0 4.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortanca() {  
12   ▶ if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   ▶ return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 3 1

sag
(min)

8 15

ortanca = 5

Öge sayısı tek: ortanca = sol tepesi = 5.

Adım 15 / 18

DEĞİŞKENLER

x = 8

sol = [5, 3, 1]

sag = [8, 15]

ortanca = 5

ÇIKTI

5.0 10.0 5.0 4.0 5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
  ► if (sol.isEmpty() || x <= sol.peek())  
3   sol.offer(x);           // küçük yarı  
  ► else  
  ►   sag.offer(x);         // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 3 1

sag
(min)

7 8 15

$x = 7: 7 > 5 \rightarrow$ **sag** heap'ine eklenir.

Adım 16 / 18

DEĞİŞKENLER

x = 7

sol = [5, 3, 1]

sag = [7, 8, 15]

ortalanca = -

ÇIKTI

5.0 10.0 5.0 4.0 5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   ▶ if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   ▶ else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 3 1

sag
(min)

7 8 15

Boyutlar 3 ve 3: denge tamam, taşıma yok.

Adım 17 / 18

DEĞİŞKENLER

x = 7

sol = [5, 3, 1]

sag = [7, 8, 15]

ortalanca = -

ÇIKTI

5.0 10.0 5.0 4.0 5.0

Çevrimiçi Ortanca: Kodu Adım Adım İzle

```
1 void ekle(int x) {  
2   if (sol.isEmpty() || x <= sol.peek())  
3     sol.offer(x);          // küçük yarı  
4   else  
5     sag.offer(x);          // büyük yarı  
6   if (sol.size() > sag.size() + 1)  
7     sag.offer(sol.poll());  
8   else if (sag.size() > sol.size())  
9     sol.offer(sag.poll());  
10 }  
11 double ortalanca() {  
12   if (sol.size() == sag.size())  
13     ▶ return (sol.peek() + sag.peek()) / 2.0;  
14   return sol.peek();  
15 }
```

akış:

5 15 1 3 8 7

sol
(max)

5 3 1

sag
(min)

7 8 15

ortalanca = 6

Öge sayısı çift: ortalanca = $(5 + 7) / 2 = 6$.

Adım 18 / 18

DEĞİŞKENLER

x = 7

sol = [5, 3, 1]

sag = [7, 8, 15]

ortalanca = 6

ÇIKTI

5.0 10.0 5.0 4.0 5.0 6.0

Rezervuar Örnekleme (Reservoir Sampling)

- Uzunluğu **bilinmeyen** bir akıştan, her öğenin **eşit olasılıkla** seçildiği k ögeli bir örneklem tutmak.
- **Algoritma R** (Vitter, 1985): ilk k öğeyi rezervuara koy. Sonra 0 tabanlı i . öge için ($i \geq k$) $j = \text{rastgele}[0, i]$ seç; $j < k$ ise $\text{rezervuar}[j] \leftarrow x_i$.
- i . öge $k / (i + 1)$ olasılıkla girer; girerse rastgele bir yerdeki öğeyi çıkarır.

Kanıt (n öge sonunda her öge k / n olasılıkla rezervuardadır): x_i girer: $k/(i+1)$. Sonraki her $t > i$ adımında çıkarılmama olasılığı $1 - (k/(t+1)) \cdot (1/k) = t/(t+1)$. Çarpım:

$$k/(i+1) \cdot (i+1)/(i+2) \cdot \dots \cdot (n-1)/n = k / n$$

İlk k öge için de aynı: girme olasılığı 1, sonraki çarpım $k/(k+1) \cdot \dots \cdot (n-1)/n = k/n$. Bellek $O(k)$, öge başına $O(1)$.

Rezervuar Örneklemesi: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
    ▶ List<Integer> r = new ArrayList<>();  
    ▶ for (int i = 0; i < k; i++) r.add(akis[i]);  
4    for (int i = k; i < akis.length; i++) {  
5        int j = rastgele.nextInt(i + 1); // 0..i  
6        if (j < k)  
7            r.set(j, akis[i]);  
8    }  
9    return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

10	20	30
0	1	2

İlk $k = 3$ öge (10, 20, 30) doğrudan rezervuara konur.

Adım 1 / 8

DEĞİŞKENLER

$k = 3$

$i = -$

$j = -$

olasılık = -

$r = [10, 20, 30]$

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

10	20	30
0	1	2

i = 3 (değer 40): j = rastgele[0, 3] → j = 1 çıktı
(bu izlemede sabitlendi).

Adım 2 / 8

DEĞİŞKENLER

k = 3

i = 3

j = 1

olasılık = 3/4

r = [10, 20, 30]

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

10	40	30
0	1	2

$j = 1 < 3 \rightarrow r[1] = 40$ (20 çıkar). 40'in girme olasılığı $3/4$.

Adım 3 / 8

DEĞİŞKENLER

$k = 3$

$i = 3$

$j = 1$

olasılık = $3/4$

$r = [10, 40, 30]$

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

10	40	30
0	1	2

i = 4 (değer 50): j = rastgele[0, 4] → j = 4 çıktı
(bu izlemede sabitlendi).

Adım 4 / 8

DEĞİŞKENLER

k = 3

i = 4

j = 4

olasılık = 3/5

r = [10, 40, 30]

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

10	40	30
0	1	2

$j = 4 \geq 3 \rightarrow 50$ rezervuara **girmez**; rezervuar değişmez.

Adım 5 / 8

DEĞİŞKENLER

$k = 3$

$i = 4$

$j = 4$

olasılık = 3/5

$r = [10, 40, 30]$

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

10	40	30
0	1	2

i = 5 (değer 60): j = rastgele[0, 5] → j = 0 çıktı
(bu izlemede sabitlendi).

Adım 6 / 8

DEĞİŞKENLER

k = 3

i = 5

j = 0

olasılık = 3/6

r = [10, 40, 30]

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```

akis:

10	20	30	40	50	60
0	1	2	3	4	5

r:

60	40	30
0	1	2

$j = 0 < 3 \rightarrow r[0] = 60$ (10 çıkar). 60'in girme olasılığı $3/6$.

Adım 7 / 8

DEĞİŞKENLER

$k = 3$

$i = 5$

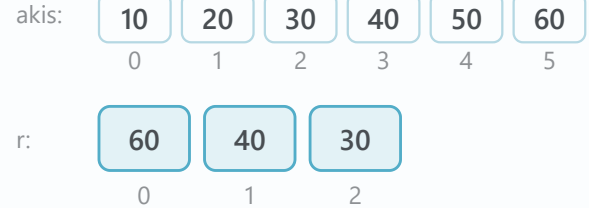
$j = 0$

olasılık = $3/6$

$r = [60, 40, 30]$

Rezervuar Örnekleme: Kodu Adım Adım İzle

```
1 List<Integer> rezervuar(int[] akis, int k) {  
2     List<Integer> r = new ArrayList<>();  
3     for (int i = 0; i < k; i++) r.add(akis[i]);  
4     for (int i = k; i < akis.length; i++) {  
5         int j = rastgele.nextInt(i + 1); // 0..i  
6         if (j < k)  
7             r.set(j, akis[i]);  
8     }  
9     return r;  
10 }
```



Akış bitti: örneklem [60, 40, 30]. Başka j değerleriyle başka sonuç çıkar; her öge $3/6 = 1/2$ olasılıkla seçilir.

Adım 8 / 8

DEĞİŞKENLER

k = 3

i = 6

j = -

olasılık = -

r = [60, 40, 30]

Akışta Örüntü Arama: Kayan Hash

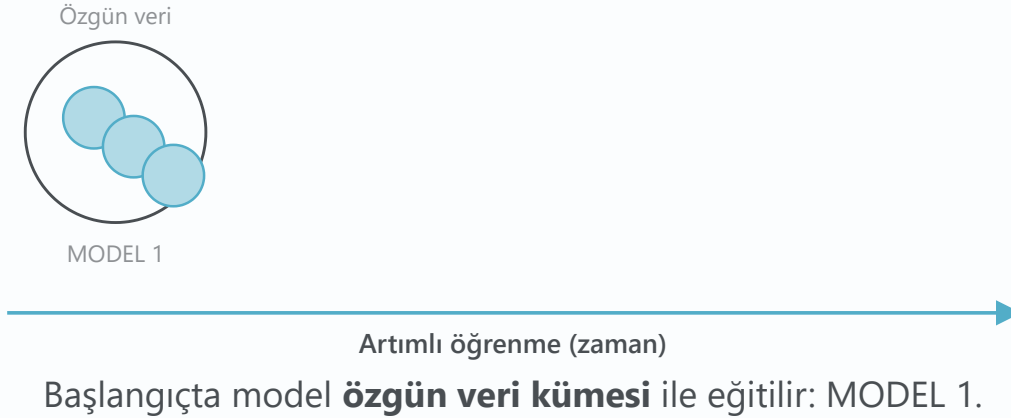
Akış karakter karakter gelir; örüntü "AB" ($m = 2$), $h = c_0 \cdot 101 + c_1$ ($A = 65$, $B = 66$, $C = 67$, $D = 68$). $h("AB") = 6631$. Yeni karakter gelince hash **O(1)**'de güncellenir:

$$h_{\text{yeni}} = (h_{\text{eski}} - c_{\text{çıkan}} \cdot 101^{m-1}) \cdot 101 + c_{\text{giren}}$$

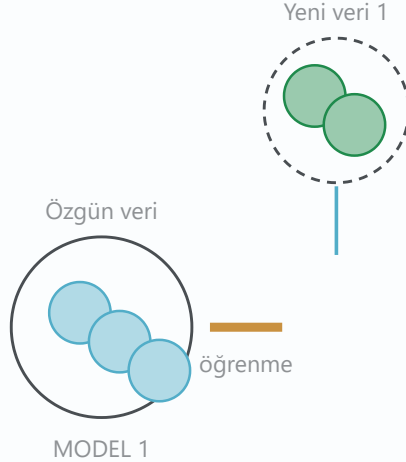
konum	pencere	hesap	hash	sonuç
0	CA	$67 \cdot 101 + 65$	6832	—
1	AB	$(6832 - 67 \cdot 101) \cdot 101 + 66$	6631	eşleşme
2	BD	$(6631 - 65 \cdot 101) \cdot 101 + 68$	6734	—
3	DA	$(6734 - 66 \cdot 101) \cdot 101 + 65$	6933	—
4	AB	$(6933 - 68 \cdot 101) \cdot 101 + 66$	6631	eşleşme

Akış "CABDAB": örüntü 1. ve 4. konumda. Hash eşitse karakterler de karşılaştırılır (çakışma olabilir). Hash'i her pencerede baştan hesaplamak $O(m)$ sürer; kayan güncelleme ile $O(1)$ 'e iner.

Artımlı Öğrenme (Incremental Learning)



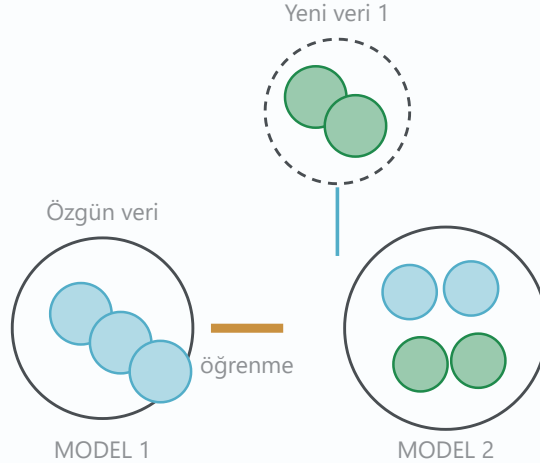
Artımlı Öğrenme (Incremental Learning)



Artımlı öğrenme (zaman)

Yeni veri 1 gelir. Tüm veriyle baştan eğitmek yerine model yalnızca yeni veriyle **güncellenir**.

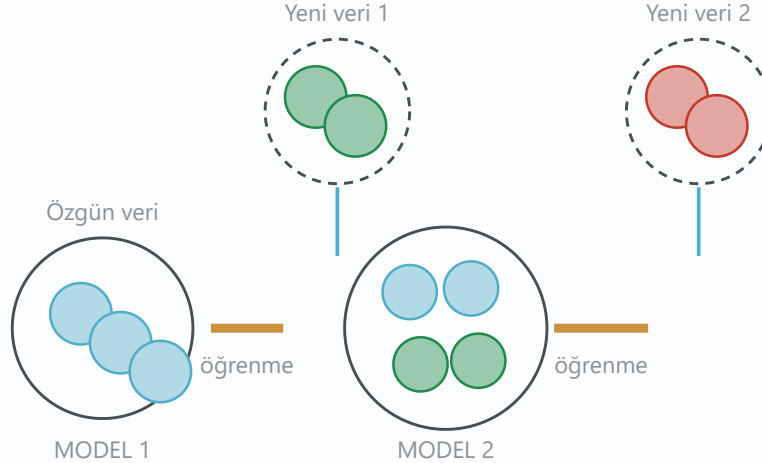
Artımlı Öğrenme (Incremental Learning)



Artımlı öğrenme (zaman)

MODEL 2 hem eski hem yeni verideki bilgiyi temsil eder; eski veri **yeniden işlenmez**.

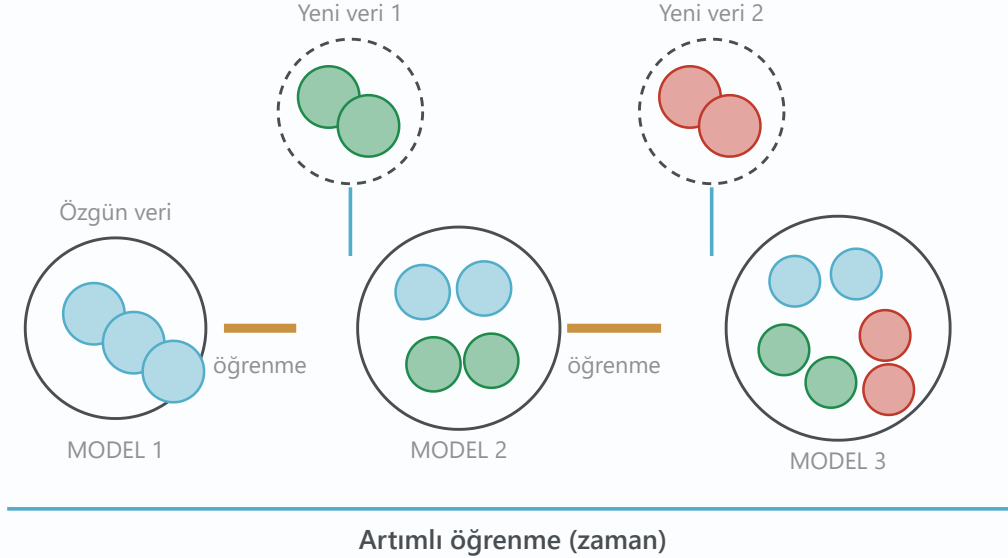
Artımlı Öğrenme (Incremental Learning)



Artımlı öğrenme (zaman)

Yeni veri 2 gelir; aynı güncelleme tekrarlanır.

Artımlı Öğrenme (Incremental Learning)



MODEL 3 üç veri kümesinin bilgisini taşır. Model hep **güncel**, maliyet yalnızca yeni veri kadardır.

Artımlı Güncelleme ve Zorlukları

Çalışan ortalama

Ortalama yeni öğeyle güncellenir; geçmiş saklamaya gerek yoktur:

$$\mu_n = \mu_{n-1} + (x_n - \mu_{n-1}) / n$$

Akış 4, 8, 6: $\mu = 4 \rightarrow 4 + (8 - 4)/2 = 6 \rightarrow 6 + (6 - 6)/3 = 6$.

Çevrimiçi gradyan inişi

$$w \leftarrow w - \eta \cdot \nabla \ell(w; x_t, y_t)$$

Her örnek bir kez kullanılır; η : öğrenme oranı.

Zorluklar

- **Felaket boyutunda unutma** (catastrophic forgetting): yeni veriye uyum sağlarken eski bilgi kaybolur.
- **Kavram kayması** (concept drift): verinin dağılımı zamanla değişir; eski veri yanıltıcı olabilir.
- **Sıra etkisi**: veri hangi sırayla gelirse model ona göre şekillenir.
- **Gürültü**: tek bir aykırı örnek modeli bozabilir; küçük η ve kayan pencere yardımcı olur.

ÖZET

Problemler ve Sonuçlar

Problem	Çevrimiçi algoritma	Sonuç	Not
Kayak kiralama	$B - 1$ gün kiral, sonra al	$2 - 1/B$	deterministikte en iyi; rastgele $e/(e-1)$
Sayfalama	LRU, FIFO, FWF	k	deterministikte en iyi
Sayfalama	LFU, LIFO	sınırsız	rekabetçi değil
Sayfalama (rastgele)	İşaretleme	$2H_k$	alt sınır H_k
Liste güncelleme	Öne taşı (MTF)	2	transpoz ve sıklık sayacı rekabetçi değil
Hareketli ortalama	Kuyruk + toplam	$O(1)$	bellek $O(k)$
Çevrimiçi ortanca	İki heap	$O(\log n)$	ortanca okuma $O(1)$
Rezervuar örnekleme	Algoritma R	$O(1)$	her öge k/n olasılıkla

Üst beş satırdaki sayılar rekabet oranı; alttakiler öge başına zamandır.

ÖZET

Özet

Konu	Hatırlanacaklar
Model	girdi parça parça gelir; karar hemen verilir ve geri alınamaz
Rekabet oranı	$ALG(\sigma) \leq c \cdot OPT(\sigma) + \alpha$, her σ için; en kötü durum analizi, düşman modeli
Kayak kiralama	$OPT = \min(d, B)$; başa baş stratejisi $2 - 1/B$; daha iyi deterministik strateji yok
Sayfalama	FIFO, LRU k-rekabetçi; OPT en uzak geleceği çıkarır; FIFO'da Belady anomalisi
Liste güncelleme	maliyet = konum; MTF 2-rekabetçi (potansiyel fonksiyonu: ters çiftler)
Veri akışları	tek geçiş, az bellek: kayan pencere, iki heap, rezervuar, kayan hash
Artımlı öğrenme	model yeni veriyle güncellenir; unutma ve kavram kayması zorlukları

ÖZET

Sık Yapılan Hatalar

"Çevrimiçi = internet" sanmak

Çevrimiçi algoritma, girdinin **zamanla açığa çıktığı** modeldir; ağ bağlantısıyla ilgisi yoktur.

Oranı ortalama durumla karıştırmak

Rekabet oranı **en kötü** girdiye göredir; "genelde iyi çalışır" bir rekabet oranı değildir.

FIFO ile LRU'yu karıştırmak

FIFO'da isabet sırayı **değiştirmez**; LRU'da isabet sayfayı en yeni yapar.

Zorunlu hataları unutmak

İlk k farklı sayfa her politikada hata verir; OPT bile 0 hata yapamaz.

Rezervuarda yanlış aralık

$j = \text{rastgele}[0, i]$ olmalı (i dahil, 0 tabanlı).
`nextInt(i)` yazmak olasılıkları bozar.

Ortancada dengeyi unutmak

Yalnızca eklemek yetmez; boyut farkı 1'i aşınca tepe öbür heap'e taşınmalıdır.

ÖZET

Örnek Problemler

1. İstek dizisi **1 2 3 4 1 2 5 1 2 3 4 5**, $k = 3$ için OPT kaç sayfa hatası yapar?
2. Kayak kiralamada $B = 20$. Başa baş stratejisinin oranı nedir? 5. gün satın alan stratejinin en kötü oranı nedir?
3. A B C D listesinde MTF ile **D C D A** isteklerinin toplam maliyeti nedir?
4. Akış 2, 9, 4, 7 için çevrimiçi ortancaları sırasıyla yazın.
5. Rezervuar örneklemede $k = 2$, akışın 0 tabanlı 9. ögesi rezervuara hangi olasılıkla girer? Akış 10 öğeyle biterse bu öge sonda hangi olasılıkla rezervuardadır?

► Cevapları göster

ÖZET

Alıştırmalar

1. FIFO sayfalamayı bir `Queue` ve bir `HashSet` ile $O(1)$ zamanda yazın.
2. OPT'yi (Belady) Java ile yazın. Zaman karmaşıklığı nedir?
3. Kayak kiralamada "B/2. gün satın al" stratejisinin oranını bulun.
4. Çevrimiçi ortancaya `sil(x)` eklemek neden zordur?
5. $k = 1$ rezervuarda 3 ögeli akışın her ögesinin $1/3$ olasılıkla seçildiğini gösterin.
6. Son k ögenin **en büyüğü** amortize $O(1)$ zamanda bulun.

► Cevaplar / ipuçları

Kaynaklar

- A. Borodin ve R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, 1998.
- D. D. Sleator ve R. E. Tarjan, "Amortized Efficiency of List Update and Paging Rules", *Communications of the ACM* 28(2), 1985.
- L. A. Belady, "A Study of Replacement Algorithms for a Virtual-Storage Computer", *IBM Systems Journal* 5(2), 1966.
- A. Fiat, R. M. Karp, M. Luby, L. A. McGeoch, D. D. Sleator ve N. E. Young, "Competitive Paging Algorithms", *Journal of Algorithms* 12(4), 1991.
- J. S. Vitter, "Random Sampling with a Reservoir", *ACM Transactions on Mathematical Software* 11(1), 1985.
- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, 2022 (Bölüm 27: Online Algorithms).

Son

Bölüm 9: Çevrimiçi Algoritmalar

Sonraki bölüm: Bölüm 10: Paralel Algoritmalar · [Terim sözlüğü](#)