Bölüm 1: Giriş

Algoritmalar

Algoritma nedir? | Algoritma türleri | Zaman karmaşıklığı | Büyüme tablosu

Asallık testi | Matris çarpımı | Permütasyon ve Fibonacci | Öklid algoritması

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

© Sercan Külcü. Tüm hakları saklıdır. · Sunumun etkileşimli HTML sürümü: https://sercankulcu.github.io/files/algorithms/slides/Bolum_01_Giris.html

GİRİŞ

Öğrenme Çıktıları

Algoritma

Algoritmanın tanımını ve özelliklerini (girdi, çıktı, açıklık, sonluluk, etkinlik) açıklamak; günlük bir işi algoritma olarak yazmak.

Analiz

Çalışma süresini girdi boyutu n'nin fonksiyonu T(n) olarak ifade etmek; büyüme tablosundan n büyürken sürenin nasıl değiştiğini okumak.

Karmaşıklık örnekleri

Kısa kodların işlem sayısını sayarak karmaşıklık sınıfını (1, n, $\sqrt{n}$, n^3, $n!$, 2^n) bulmak; belleklemenin üstel süreyi nasıl düşürdüğünü görmek.

Öklid algoritması

EBOB hesabını adım adım izlemek; algoritmanın doğruluğunu ve $O(\log \min(a, b))$ süresini gerekçelendirmek.

ALGORİTMA KAVRAMI

Algoritma Nedir?

- Bir problemi çözmek veya bir görevi yerine getirmek için izlenen **adım adım yönergeler dizisidir**.
- Girdi verileri alınır, işlemler yapılır, **çıktı** elde edilir.
- Her adım **belirli, açık ve anlaşılır** olmalı; algoritma **sınırlı sayıda adımda** sonlanmalıdır.

Girdi (input)
Sıfır veya daha fazla değer dışarıdan alınır.

Çıktı (output)
En az bir sonuç üretilir; sonuç girdiyle ilişkilidir.

Açıklık
Her adımın tek bir anlamı vardır; yoruma yer bırakmaz.

Sonluluk
Her girdi için sonlu sayıda adımdan sonra durur.

Etkinlik
Her adım temel ve gerçekten yapılabilir bir işlemdir.

Doğruluk
Her geçerli girdi için doğru çıktıyı verir.

ALGORİTMA KAVRAMI

Günlük Hayattan: Çay Hazırlama Algoritması

Girdi: su, çay yaprakları, şeker (isteğe bağlı), süt (isteğe bağlı)
Çıktı: çay

1. Suyu kaynat.
2. Kaynamış suyu demliğe koy.
3. Demliğin içine çay yapraklarını ekle.
4. Çayın demlenmesini bekle.
5. Demlenen çayı fincana süzerek koy.
6. İsteğe bağlı olarak şeker ve süt ekle.

Sıra önemlidir

Adım 2 ile 3'ün yeri değişebilir, ama 1 ile 5'in yeri değişemez: algoritma bir **sıralı** adımlar listesidir.

Açık mı?

"Demlenmesini bekle" ne kadar? Bir bilgisayar için adım şöyle olmalıdır: **10 dakika bekle**.

Karar

Adım 6 bir **koşul** içerir: "şeker isteniyorsa ekle". Algoritmalar karar noktaları da içerir.

Ortalama Bulma: Algoritmadan Koda

Algoritma

Girdi: Sayı1, Sayı2, Sayı3 · Çıktı: Ortalama

1. Toplam = Sayı1 + Sayı2 + Sayı3
2. Ortalama = Toplam / 3
3. Ortalama değerini ekrana yazdır.

Python

```python
def ortalama_hesapla(sayi1, sayi2, sayi3):
    ortalama = (sayi1 + sayi2 + sayi3) / 3
    return ortalama

sayi1 = float(input("Birinci sayıyı girin: "))
sayi2 = float(input("İkinci sayıyı girin: "))
sayi3 = float(input("Üçüncü sayıyı girin: "))
ort = ortalama_hesapla(sayi1, sayi2, sayi3)
print("Girilen sayıların ortalaması:", ort)
```

Java

```java
public static void main(String[] args) {
  Scanner giris = new Scanner(System.in);
  System.out.println("Birinci sayıyı girin: ");
  double sayi1 = giris.nextDouble();
  System.out.println("İkinci sayıyı girin: ");
  double sayi2 = giris.nextDouble();
  System.out.println("Üçüncü sayıyı girin: ");
  double sayi3 = giris.nextDouble();
  double ortalama = (sayi1 + sayi2 + sayi3) / 3;
  System.out.println("Sayıların ortalaması: "
                     + ortalama);
}
```

Aynı algoritma farklı dillerde yazılabilir: algoritma **dilden bağımsızdır**, program onun bir gerçeklemesidir.

ALGORİTMA KAVRAMI

Algoritmalar Nerede? Başlıca Türler

Sıralama
Öğeleri bir düzene koyar: eklemeli sıralama, birleştirme sıralaması, hızlı sıralama.

Arama
Bir öğeyi bulur: doğrusal ve ikili arama, hash.

Çizge
En kısa yol (Dijkstra), minimum kapsayan ağaç, maksimum akış (Ford–Fulkerson).

Dizgi ve sıkıştırma
Metin arama, trie; dosya boyutunu küçültme: Huffman, gzip.

Tasarım teknikleri
Böl ve fethet (birleştirme sıralaması, ikili arama, hızlı üs alma), dinamik programlama, açgözlü, geri izleme, dal sınır.

Rastgele ve yaklaşık
Aynı girdide farklı yol izleyebilen **rastgele** algoritmalar; zor problemlere yaklaşık çözüm.

Deterministik algoritma aynı girdi için her zaman aynı adımları izler ve aynı çıktıyı verir (ör. en büyük öğeyi bulma). **Rastgele** (randomized) algoritma zar atar: hızlı sıralamada pivotu rastgele seçmek gibi.

ALGORİTMA ANALİZİ

Zaman Karmaşıklığı

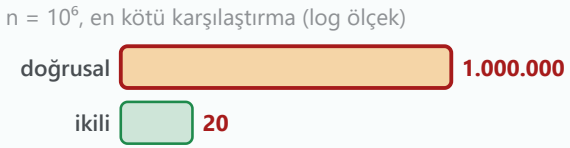
- **Algoritma analizi**, işlemci zamanı ve bellek gibi **kaynakların** kullanımını değerlendirir; aynı problemi çözen algoritmalar arasında seçim yapmayı sağlar.
- Çalışma süresi **girdi boyutu n** ile ilişkilendirilir: **T(n)**. Saniye değil, **temel işlem sayısı** sayılır: saniye bilgisayara ve dile göre değişir, işlem sayısı değişmez.

Örnek: n öğeli dizinin toplamı, satır satır sayım

```
int toplam = 0;                    // 1 kez
for (int i = 0; i < n; i++)        // n+1 karşılaştırma
  toplam += dizi[i];               // n kez
return toplam;                     // 1 kez
```

Neden önemli?

10^6 öğeli sıralı bir dizide arama: doğrusal arama en kötü **1.000.000**, ikili arama en fazla **20** karşılaştırma yapar.



T(n) = 1 + (n + 1) + n + 1 = **2n + 3**. Sabitler makineye bağlıdır; önemli olan T(n)'nin **n ile doğrusal** büyümesidir → Θ(n).

ALGORİTMA ANALİZİ

Büyüme Tablosu: n Büyürken

f(n)	n = 10	n = 100	n = 1000	$n = 10^6$	süre ($n = 10^6$)
1	1	1	1	1	1 ns
$\log_2 n$	3,3	6,6	10,0	19,9	19,9 ns
$\sqrt{n}$	3,2	10	31,6	1.000	1 µs
n	10	100	1.000	1.000.000	1 ms
$n \log_2 n$	33,2	664,4	9.966	$2{,}0 \cdot 10^7$	19,9 ms
n^2	100	10.000	1.000.000	$1{,}0 \cdot 10^{12}$	16,7 dk
n^3	1.000	1.000.000	$1{,}0 \cdot 10^9$	$1{,}0 \cdot 10^{18}$	≈ 32 yıl
2^n	1.024	$1{,}3 \cdot 10^{30}$	$\approx 1{,}1 \cdot 10^{301}$	$\approx 10^{301030}$	evrenin yaşından çok
n!	3.628.800	$9{,}3 \cdot 10^{157}$	$\approx 10^{2567}$	$\approx 10^{5565708}$	evrenin yaşından çok

Son sütun: saniyede 10^9 işlem yapan bir bilgisayarda $n = 10^6$ için süre. Daha hızlı bir bilgisayar sabiti küçültür; **daha iyi bir algoritma büyüme hızını** değiştirir. n^2 yerine $n \log n$, 10^6 öğede dakikaları milisaniyelere indirir.

KARMAŞIKLIK ÖRNEKLERİ

Sabit ve Doğrusal Zaman: O(1), O(n)

İlk öğeyi yazdırma T(n) = O(1)

```
void ilkElemaniYazdir(int[] dizi) {
  if (dizi.length > 0)
    System.out.println("İlk eleman: "
                       + dizi[0]);
  else
    System.out.println("Dizi boş");
}
```

Dizi 10 öğeli de olsa 10 milyon öğeli de olsa **tek** karşılaştırma ve tek erişim yapılır: süre n'den bağımsızdır.

En büyük öğeyi bulma T(n) = O(n)

```
Integer enBuyukBul(int[] dizi) {
  if (dizi.length == 0)
    return null;              // dizi boşsa
  int enBuyuk = dizi[0];
  for (int eleman : dizi)
    if (eleman > enBuyuk)
      enBuyuk = eleman;
  return enBuyuk;
}
```

Her öğe **bir kez** incelenir: n karşılaştırma. Ek bellek O(1). Örnek: [5, 8, 12, 4] → 12.



KARMAŞIKLIK ÖRNEKLERİ

Asal Sayı Kontrolü: O(√n)

```java
boolean asalMi(int n) {
  if (n <= 1)
    return false;
  else if (n == 2)
    return true;
  else if (n % 2 == 0)
    return false;
  int kokN = (int) Math.sqrt(n) + 1;
  for (int i = 3; i < kokN; i += 2)
    if (n % i == 0)
      return false;
  return true;
}
```

Neden √n'ye kadar?

n = a · b ise a ve b'den biri **√n'den küçük ya da eşittir**. √n'ye kadar bölen yoksa n asaldır.

Kaç bölme?

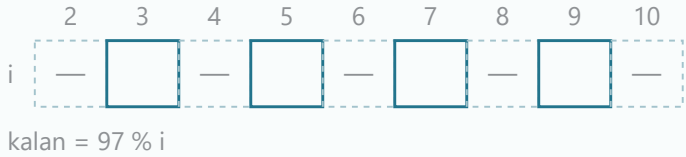
Çift sayılar baştan elenir; yalnızca tek i denenir: en fazla ≈ √n / 2 bölme → **T(n) = O(√n)**.

n = 97: i = 3, 5, 7, 9 → **4** bölme (2'den 96'ya denense 95).
n = 1.000.000.007: **15.810** bölme.

Not: burada girdi boyutu n sayısının **değeridir**. Sayının bit uzunluğu b ise $\sqrt{n} = 2^{b/2}$: bu yöntem bit sayısına göre üstel büyür.

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
9    for (int i = 3; i < kokN; i += 2)
10     if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

n = 97: 1'den büyük, 2'ye eşit değil ve tek. İlk üç kontrolün hiçbiri sonuç döndürmez; 2'nin katları baştan elenmiş olur.

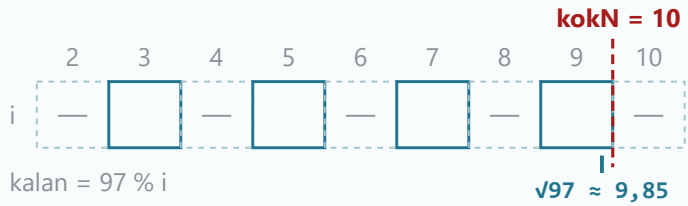
Adım 1 / 8

DEĞİŞKENLER

`n = 97` `kokN = –` `i = –`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
▶    int kokN = (int) Math.sqrt(n) + 1;
9    for (int i = 3; i < kokN; i += 2)
10     if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

kokN = (int) √97 + 1 = 9 + 1 = **10**. Yalnızca 10'dan küçük tek sayılar denenecek: 3, 5, 7, 9.

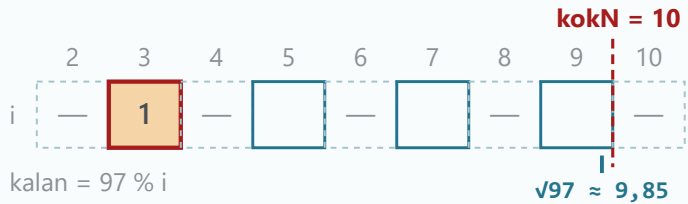
Adım 2 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = –`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
▶    for (int i = 3; i < kokN; i += 2)
▶      if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

i = 3: 97 % 3 = **1** ≠ 0 → 3, 97'yi bölmez. `i += 2`.

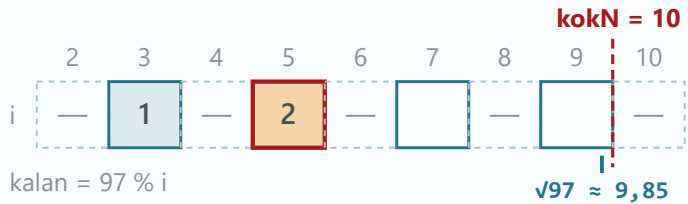
Adım 3 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = 3` `n % i = 1`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
▶    for (int i = 3; i < kokN; i += 2)
▶      if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

i = 5: 97 % 5 = **2** ≠ 0 → 5, 97'yi bölmez. `i += 2`.

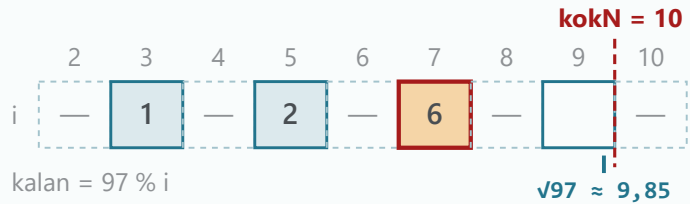
Adım 4 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = 5` `n % i = 2`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
▶    for (int i = 3; i < kokN; i += 2)
▶      if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

i = 7: 97 % 7 = **6** ≠ 0 → 7, 97'yi bölmez. `i += 2`.

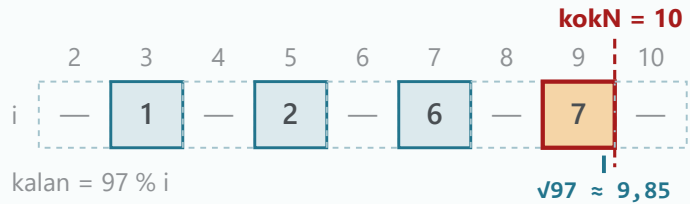
Adım 5 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = 7` `n % i = 6`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
▶    for (int i = 3; i < kokN; i += 2)
▶      if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

i = 9: 97 % 9 = **7** ≠ 0 → 9, 97'yi bölmez. `i += 2`. Sıradaki aday i = 11.

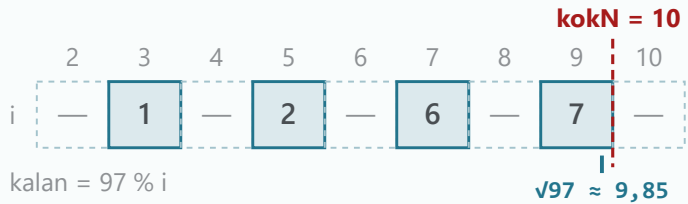
Adım 6 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = 9` `n % i = 7`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
▶    for (int i = 3; i < kokN; i += 2)
10     if (n % i == 0)
11       return false;
12   return true;
13 }
```



çift i'ler (gri) hiç denenmez

i = 11 < 10 sağlanmaz, döngü biter. √97'ye kadar hiçbir tek sayı 97'yi bölmedi.

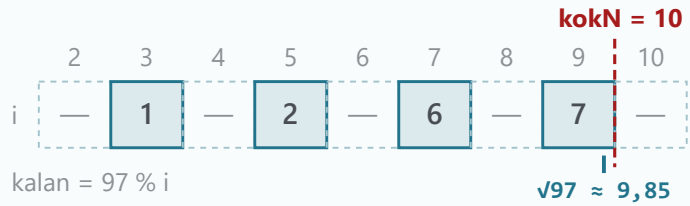
Adım 7 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = 11` `n % i = –`

asalMi(97): Adım Adım

```
1  boolean asalMi(int n) {
2    if (n <= 1)
3      return false;
4    else if (n == 2)
5      return true;
6    else if (n % 2 == 0)
7      return false;
8    int kokN = (int) Math.sqrt(n) + 1;
9    for (int i = 3; i < kokN; i += 2)
10     if (n % i == 0)
11       return false;
▶    return true;
13 }
```



çift i'ler (gri) hiç denenmez

`return true` : 97 asaldır. Yalnızca **4 bölme** yetti; 2'den 96'ya kadar her sayı denenseydi 95 bölme gerekirdi.

Adım 8 / 8

DEĞİŞKENLER

`n = 97` `kokN = 10` `i = 11` `sonuç = true`

Kübik Zaman: Matris Çarpımı, $\Theta(n^3)$

```
int[][] matrisCarp(int[][] A, int[][] B) {
  int n = A.length, m = B[0].length, p = B.length;
  int[][] sonuc = new int[n][m];
  for (int i = 0; i < n; i++)
    for (int j = 0; j < m; j++) {
      int eleman = 0;
      for (int k = 0; k < p; k++)
        eleman += A[i][k] * B[k][j];
      sonuc[i][j] = eleman;
    }
  return sonuc;
}
```

Girdi: A = $[a_{ij}]$, B = $[b_{ij}]$ (n × n). **Çıktı:** C = $[c_{ij}]$ = A · B, i, j = 1, 2, …, n.

Tanım

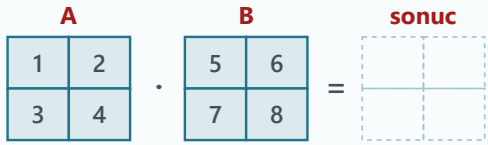
$$c_{ij} = \Sigma_{k=1..n}\, a_{ik} \cdot b_{kj}$$

1	2	·	5	6	=	19	22
3	4		7	8		43	50

c_{11} = 1·5 + 2·7 = 19. Her c_{ij} için n çarpma, n^2 öğe → **n^3 çarpma**: T(n) = $\Theta(n^3)$.

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
▶    int n = A.length, m = B[0].length, p = B.length;
▶    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
6        int eleman = 0;
7        for (int k = 0; k < p; k++)
8          eleman += A[i][k] * B[k][j];
9        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



A ve B 2 × 2: n = m = p = 2. `sonuc` 2 × 2 sıfır matrisi olarak oluşturulur.

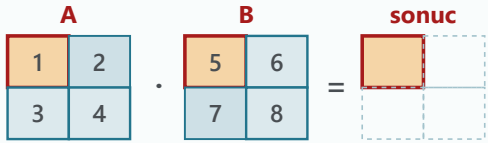
Adım 1 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = –` `j = –` `k = –` `eleman = –` `çarpma = 0`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
▶        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
9        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 1·5 = 5

i = 0, j = 0: `eleman = 0`. k = 0: A[0][0] · B[0][0] = 1 · 5 = 5 eklenir.

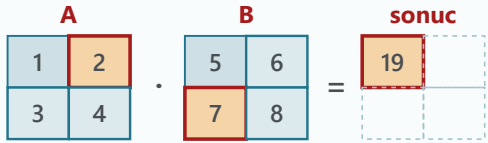
Adım 2 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 0` `j = 0` `k = 0` `eleman = 5` `çarpma = 1`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
6        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
▶        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 5 + 2·7 = 19

k = 1: A[0][1] · B[1][0] = 2 · 7 → eleman = 5 + 14 = **19**. İç döngü biter: `sonuc[0][0] = 19`.

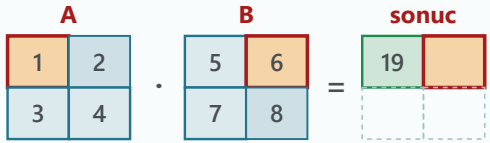
Adım 3 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 0` `j = 0` `k = 1` `eleman = 19` `çarpma = 2`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
▶        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
9        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 1·6 = 6

i = 0, j = 1: `eleman = 0`. k = 0: A[0][0] · B[0][1] = 1 · 6 = 6 eklenir.

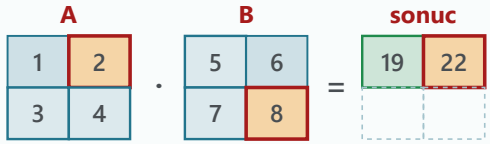
Adım 4 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 0` `j = 1` `k = 0` `eleman = 6` `çarpma = 3`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
6        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
▶        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 6 + 2·8 = 22

k = 1: A[0][1] · B[1][1] = 2 · 8 → eleman = 6 + 16 = **22**. İç döngü biter: `sonuc[0][1] = 22`.

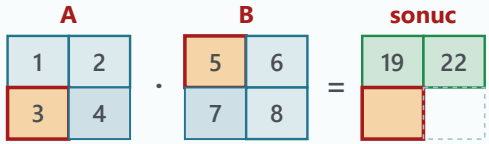
Adım 5 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 0` `j = 1` `k = 1` `eleman = 22` `çarpma = 4`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
       int eleman = 0;
       for (int k = 0; k < p; k++)
         eleman += A[i][k] * B[k][j];
9      sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 3·5 = 15

i = 1, j = 0: `eleman = 0`. k = 0: A[1][0] · B[0][0] = 3 · 5 = 15 eklenir.

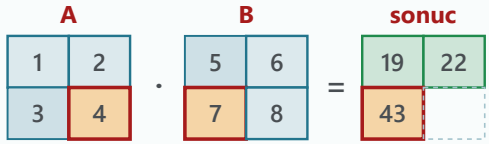
Adım 6 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 1` `j = 0` `k = 0` `eleman = 15` `çarpma = 5`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
6        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
▶        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 15 + 4·7 = 43

k = 1: A[1][1] · B[1][0] = 4 · 7 → eleman = 15 + 28 = **43**. İç döngü biter: `sonuc[1][0] = 43` .

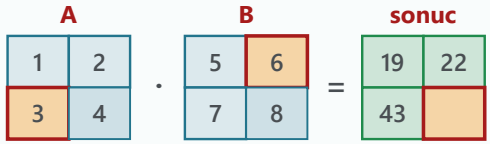
Adım 7 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 1` `j = 0` `k = 1` `eleman = 43` `çarpma = 6`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
▶        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
9        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 3·6 = 18

i = 1, j = 1: `eleman = 0`. k = 0: A[1][0] · B[0][1] = 3 · 6 = 18 eklenir.

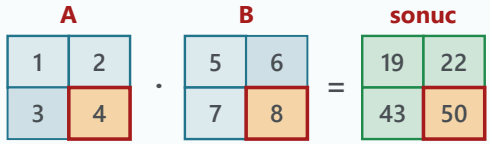
Adım 8 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 1` `j = 1` `k = 0` `eleman = 18` `çarpma = 7`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
6        int eleman = 0;
▶        for (int k = 0; k < p; k++)
▶          eleman += A[i][k] * B[k][j];
▶        sonuc[i][j] = eleman;
10     }
11   return sonuc;
12 }
```



eleman = 18 + 4·8 = 50

k = 1: A[1][1] · B[1][1] = 4 · 8 → eleman = 18 + 32 = **50**. İç döngü biter: `sonuc[1][1] = 50` .

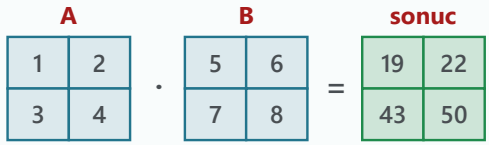
Adım 9 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 1` `j = 1` `k = 1` `eleman = 50` `çarpma = 8`

matrisCarp: 2 × 2 Matrisler Adım Adım

```
1  int[][] matrisCarp(int[][] A, int[][] B) {
2    int n = A.length, m = B[0].length, p = B.length;
3    int[][] sonuc = new int[n][m];
4    for (int i = 0; i < n; i++)
5      for (int j = 0; j < m; j++) {
6        int eleman = 0;
7        for (int k = 0; k < p; k++)
8          eleman += A[i][k] * B[k][j];
9        sonuc[i][j] = eleman;
10     }
▶    return sonuc;
12 }
```



sonuc = [[19, 22], [43, 50]] döner. **8 = 2^3 çarpma**: n^2 öğenin her biri için n çarpma → n^3.

Adım 10 / 10

DEĞİŞKENLER

`n = 2` `m = 2` `p = 2` `i = 2` `j = –` `k = –` `eleman = –` `çarpma = 8`

KARMAŞIKLIK ÖRNEKLERİ

Faktöriyel Zaman: Tüm Permütasyonları Üretme

```java
void permutasyon(int[] dizi, int k) {
  if (k == dizi.length) {          // bir permütasyon hazır
    System.out.println(Arrays.toString(dizi));
    return;
  }
  for (int i = k; i < dizi.length; i++) {
    takas(dizi, k, i);             // i. elemanı k. konuma al
    permutasyon(dizi, k + 1);      // geri kalanları diz
    takas(dizi, k, i);             // eski hâline getir
  }
}
```

[1, 2, 3] → 123, 132, 213, 231, 321, 312. İlk konum için n, ikinci için n−1, … seçenek: **n · (n−1) · … · 1 = n!** permütasyon.

n	n!
3	6
5	120
10	3.628.800
15	$1{,}3 \cdot 10^{12}$
20	$2{,}4 \cdot 10^{18}$

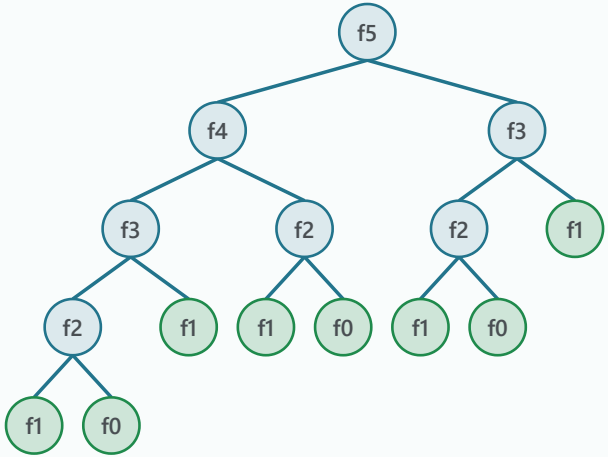
Her permütasyonu yazmak n adım sürer: toplam **Θ(n · n!)**, kısaca "O(n!) sınıfı". n = 20 bile pratikte imkânsızdır.

Üstel Zaman: Fibonacci Serisi

```
long fib(int n) {           // saf özyineleme
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}

long[] bellek = new long[N + 1]; // 0: yok
long fibBellek(int n) {     // bellekleme
  if (n <= 1) return n;
  if (bellek[n] == 0)
    bellek[n] = fibBellek(n - 1) + fibBellek(n - 2);
  return bellek[n];
}
```

F(0) = 0, F(1) = 1, F(n) = F(n−1) + F(n−2).
fib: her çağrı iki çağrı doğurur → **O(2^n)** (kesin olarak Θ($1{,}618^n$)); fib(30) 2.692.537 çağrı yapar.
fibBellek: her F(k) bir kez hesaplanır, 2n − 1 çağrı → **O(n)**; n = 30 için 59 çağrı.

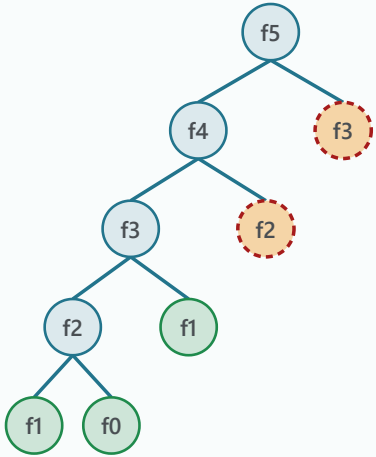


Saf özyineleme: **15 çağrı**. f3 iki kez, f2 üç kez, f1 beş kez hesaplanır; ağaç her düzeyde yaklaşık ikiye katlanır → üstel büyüme.

Üstel Zaman: Fibonacci Serisi

```
long fib(int n) {           // saf özyineleme
  if (n <= 1) return n;
  return fib(n - 1) + fib(n - 2);
}

long[] bellek = new long[N + 1]; // 0: yok
long fibBellek(int n) {     // bellekleme
  if (n <= 1) return n;
  if (bellek[n] == 0)
    bellek[n] = fibBellek(n - 1) + fibBellek(n - 2);
  return bellek[n];
}
```



F(0) = 0, F(1) = 1, F(n) = F(n−1) + F(n−2).
fib: her çağrı iki çağrı doğurur → **O(2^n)** (kesin olarak Θ($1{,}618^n$)); fib(30) 2.692.537 çağrı yapar.
fibBellek: her F(k) bir kez hesaplanır, 2n − 1 çağrı → **O(n)**; n = 30 için 59 çağrı.

Belleklemeli: **9 çağrı**. Turuncu kesikli düğümler (f2, f3) bellek dizisinden **hazır** döner, alt ağaçları hiç açılmaz → O(n).

Öklid Algoritması (EBOB)

- **En büyük ortak böleni** (EBOB, gcd) bulur.
- Bilinen en eski algoritmalardan biridir (Öklid, MÖ ~300); en basit EBOB yöntemi.
- Temel fikir: **ebob(a, b) = ebob(b, a mod b)**, ebob(a, 0) = a.

1. **Adım 1:** büyük sayıyı küçüğe böl, kalanı al.
2. **Adım 2:** kalan sıfır olana kadar (bölen, kalan) çiftiyle tekrarla.
3. **Adım 3:** kalan sıfır olduğunda son **bölen** EBOB'dur.

```
int ebob(int a, int b) {
  if (b == 0)
    return a;
  return ebob(b, a % b);
}
```

Neden doğru?

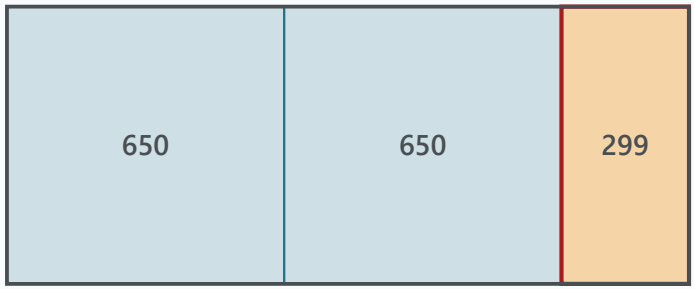
a = q·b + r ise a ile b'yi bölen her sayı r = a − q·b'yi de böler; b ile r'yi bölen her sayı a'yı da böler. Ortak bölenler aynıdır.

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
▶   if (b == 0)
3     return a;
▶   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

`ebob(1599, 650)`



1599 × 650 dikdörtgen

1599 = 2 · 650 + **299**. 1599×650 dikdörtgenden 2 kare kesilir, kalan şerit 650×299 → `ebob(650, 299)`.

Adım 1 / 7

DEĞİŞKENLER

`a = 1599` `b = 650` `a%b = 299`

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
▶   if (b == 0)
3     return a;
▶   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

- **ebob(650, 299)**
- ebob(1599, 650)



650 × 299 dikdörtgen

650 = 2 · 299 + **52**. 650×299 dikdörtgenden 2 kare kesilir, kalan şerit 299×52 → `ebob(299, 52)`.

Adım 2 / 7

DEĞİŞKENLER

`a = 650` `b = 299` `a%b = 52`

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
▶   if (b == 0)
3     return a;
▶   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

- **ebob(299, 52)**
- ebob(650, 299)
- ebob(1599, 650)



299 × 52 dikdörtgen

299 = 5 · 52 + **39**. 299×52 dikdörtgenden 5 kare kesilir, kalan şerit 52×39 → `ebob(52, 39)`.

Adım 3 / 7

DEĞİŞKENLER

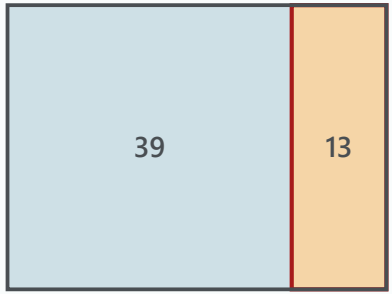
`a = 299` `b = 52` `a%b = 39`

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
▶   if (b == 0)
3     return a;
▶   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

- **ebob(52, 39)**
- ebob(299, 52)
- ebob(650, 299)
- ebob(1599, 650)



52 × 39 dikdörtgen

52 = 1 · 39 + **13**. 52×39 dikdörtgenden 1 kare kesilir, kalan şerit 39×13 → `ebob(39, 13)`.

Adım 4 / 7

DEĞİŞKENLER

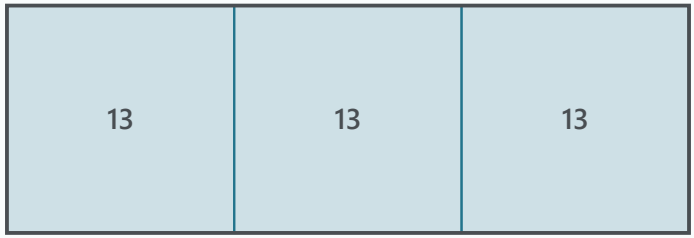
`a = 52` `b = 39` `a%b = 13`

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
▶   if (b == 0)
3     return a;
▶   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

- **ebob(39, 13)**
- ebob(52, 39)
- ebob(299, 52)
- ebob(650, 299)
- ebob(1599, 650)



39 × 13 dikdörtgen

39 = 3 · 13 + **0**. 39×13 dikdörtgenden 3 kare kesilir, kalan şerit 13×0 → `ebob(13, 0)`.

Adım 5 / 7

DEĞİŞKENLER

`a = 39` `b = 13` `a%b = 0`

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
▶   if (b == 0)
▶     return a;
4   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

- **ebob(13, 0)**
- ebob(39, 13)
- ebob(52, 39)
- ebob(299, 52)
- ebob(650, 299)
- ebob(1599, 650)

ebob(13, 0) = **13**

b == 0 → **return 13**. 13, 1599 ile 650'yi tam bölen en büyük sayıdır.

Adım 6 / 7

DEĞİŞKENLER

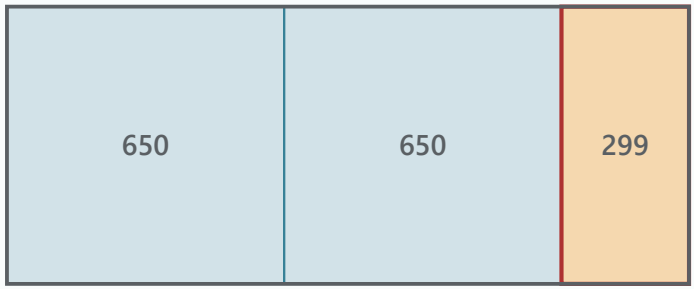
a = **13** b = **0**

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {
2   if (b == 0)
3     return a;
▶   return ebob(b, a % b);
5 }
```

ÇAĞRI YIĞINI

`ebob(1599, 650)`



1599 × 650 dikdörtgen

Dönüş değeri 13 tüm çağrılardan aynen geri döner: **ebob(1599, 650) = 13**. 5 bölme yetti. 13 × 13'lük kareler 1599 × 650'yi tam döşer.

Adım 7 / 7

DEĞİŞKENLER

`a = 1599` `b = 650` `sonuç = 13`

Öklid Algoritması: Neden Hızlı?

- Verimli ve hızlı çalışır: **her iki adımda** büyük sayı en az **yarıya** iner (a mod b < a/2).
- Bu yüzden adım sayısı en fazla ≈ 2 $\log_2$ min(a, b): **O(log min(a, b))**.
- **En kötü senaryo** ardışık Fibonacci sayılarıyla oluşur: her bölümde q = 1 çıkar, sayılar en yavaş küçülür (Lamé teoremi).
- Karşılaştırma: 1'den min(a, b)'ye kadar tüm bölenleri denemek O(min(a, b)) sürer.

En kötü örnek: ebob(55, 34)

```
55 = 1 · 34 + 21
34 = 1 · 21 + 13
21 = 1 · 13 + 8
13 = 1 · 8 + 5
8 = 1 · 5 + 3
5 = 1 · 3 + 2
3 = 1 · 2 + 1
2 = 2 · 1 + 0
```

8 bölme, EBOB = 1. ebob(1599, 650) ise yalnızca 5 bölme yapar.

Özet

Konu	Hatırlanacaklar
Algoritma	girdi → açık, sonlu, etkin adımlar → çıktı; dilden bağımsızdır
Analiz	T(n): girdi boyutuna göre temel işlem sayısı; zaman ve bellek
Büyüme	$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < n!$; daha iyi algoritma büyüme hızını değiştirir
Örnekler	ilk öğe $O(1)$, en büyük öğe $O(n)$, asallık $O(\sqrt{n})$, matris çarpımı $\Theta(n^3)$, permütasyonlar $O(n!)$, saf Fibonacci $O(2^n)$
Bellekleme	aynı alt problemi saklamak Fibonacci'yi $O(2^n)$'den $O(n)$'ye indirir
Öklid	ebob(a, b) = ebob(b, a mod b); $O(\log \min(a, b))$; en kötü: ardışık Fibonacci

ÖZET

Etkileşimli Simülatörler

Algoritma Karmaşıklığı Analizi

O(log n), O(n), O(n log n), O(n^2) ve O(n^3) örnek JavaScript fonksiyonları ya da yeni bir fonksiyon; beş farklı girdi boyutunda tarayıcıda ölçülen sürelerin grafikte karşılaştırması.

ÖZET

Sık Yapılan Hatalar

Süreyi saniyeyle ölçmek

Saniye makineye bağlıdır; analizde n'ye bağlı işlem sayısı ve büyüme hızı kullanılır.

Belleklemeyi unutmak

Saf özyinelemeli Fibonacci $O(2^n)$'dir; aynı alt problemi saklamak onu O(n)'ye indirir.

√n sınırını eksik kurmak

`i < (int) Math.sqrt(n)` yazılırsa √n'nin kendisi denenmez: 9, 25, 49 gibi kare sayılar asal çıkar.

Girdi boyutunu karıştırmak

Asallık testinde n sayının değeridir; b bitlik bir sayı için $\sqrt{n} = 2^{b/2}$, yani bit sayısına göre üstel süre.

Matris çarpımını n^2 sanmak

Sonuçta n^2 öğe vardır, ama her öğe n çarpma ister: toplam n^3 çarpma.

Takası geri almamak

Permütasyon üretiminde ikinci `takas` unutulursa dizi bozulur: [1, 2, 3] için aynı permütasyonlar tekrarlanır, bazıları hiç üretilmez.

ÖZET

Örnek Problemler

1. `asalMi(221)` çağrısında döngüde hangi i değerleri denenir, kaç bölme yapılır, sonuç nedir?
2. $T(n) = 5n^2 + 100n \log_2 n + 7$ için Θ sınıfını yazın. $n = 1000$'de hangi terim baskındır?
3. İki 3×3 matris `matrisCarp` ile çarpılırken kaç çarpma yapılır? $n = 100$ için?
4. `permutasyon([1, 2, 3, 4], 0)` kaç satır yazdırır? İlk iki satır nedir?
5. ebob(252, 105)'i Öklid algoritmasıyla bulun.

▶ **Cevapları göster**

ÖZET

Alıştırmalar

1. Bir dizide verilen bir sayının kaç kez geçtiğini bulan algoritmayı yazın; zaman ve alan karmaşıklığını verin.
2. İç içe döngüler: `for i in 0..n-1: for j in i..n-1: işlem` kaç kez çalışır? Sınıfı nedir?
3. `enBuyukBul`'u dizinin hem en küçük hem en büyük öğesini döndürecek biçimde değiştirin. Kaç karşılaştırma yapar?
4. Öklid algoritmasını döngüyle (özyinelemesiz) yazın.
5. Saniyede 10^9 işlemle 1 saniyede çözülebilecek en büyük n: f(n) = n^2 ve f(n) = 2^n için bulun.
6. `while (n > 1) n = n / 2;` döngüsü kaç kez döner?

▶ **Cevaplar / ipuçları**

Kaynaklar

- Cormen, Leiserson, Rivest ve Stein, *Introduction to Algorithms*, 4. baskı, MIT Press, 2022 (Bölüm 1–3).
- Sedgewick ve Wayne, *Algorithms*, 4. baskı, Addison-Wesley, 2011 (Bölüm 1.4: Analysis of Algorithms).
- D. E. Knuth, *The Art of Computer Programming, Vol. 1: Fundamental Algorithms*, 3. baskı, 1997 (algoritmanın özellikleri; Öklid).
- J. Kleinberg ve É. Tardos, *Algorithm Design*, Pearson, 2005 (Bölüm 2: Basics of Algorithm Analysis).

Son

Bölüm 1: Giriş

Sonraki bölüm: Bölüm 2: Algoritma Karmaşıklığı · Terim sözlüğü