

Bölüm 1: Giriş

Algoritmalar

Algoritma nedir?

Algoritma analizi

O , Ω ve Θ gösterimleri

Karmaşıklık sınıfları

Eklemeli sıralama

Birleştirme sıralaması

İkili arama ve üs alma

Euclid algoritması

Sercan KÜLCÜ · Giresun Üniversitesi · Bilgisayar Mühendisliği

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; en kötü, ortalama ve en iyi durumu ayırt etmek.

Gösterimler

O , Ω ve Θ gösterimlerinin üst, alt ve sıkı sınır anlamını kavramak; bir koddan karmaşıklık sınıfını (1 , $\log n$, n , $n \log n$, n^2 , n^3 , 2^n , $n!$) okumak.

Temel algoritmalar

Eklemeli sıralama, birleştirme, ikili arama, hızlı üs alma ve Euclid algoritmasını adım adım izlemek ve özyineleme bağıntılarını kurmak.

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.

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

```
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

```
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 gerçekleştirmidir.

Algoritmalar Nerede? Başlıca Türler

Sıralama

Öğeleri bir düzene koyar: eklemeli sıralama, birleştirme sıralaması, hızlı sıralama. (Bölüm 2)

Arama

Bir öğeyi bulur: doğrusal ve ikili arama, hash. (Bölüm 3)

Çizge

En kısa yol (Dijkstra), minimum kapsayan ağaç, maksimum akış (Ford–Fulkerson). (Bölüm 4)

Dizgi ve sıkıştırma

Metin arama, trie; dosya boyutunu küçültme: Huffman, gzip. (Bölüm 5)

Tasarım teknikleri

Böl ve fethet, dinamik programlama, açgözlü, geri izleme, dal sınır. (Bölüm 6, 11–14)

Rastgele ve yaklaşık

Aynı girdide farklı yol izleyebilen **rastgele** algoritmalar; zor problemlere yaklaşık çözüm. (Bölüm 7–8)

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.

Algoritma Analizi

- Bir algoritmanın **performansını** anlamak ve değerlendirmek için kullanılır.
- İşlemci ve bellek gibi **kaynakların** verimli kullanılmasını sağlar.
- **Zaman analizi:** algoritmanın çalışma süresini ölçer; genellikle büyük O gösterimiyle yazılır.
- **Bellek analizi:** algoritmanın ek bellek kullanımını (bellek karmaşıklığı) değerlendirir.
- Aynı problemi çözen algoritmalar arasında **seçim** yapmamızı sağlar.

Neden önemli?

10^6 ögeli sıralı bir dizide arama:

doğrusal arama → en kötü **1.000.000** karşılaştırma

ikili arama → en fazla **20** karşılaştırma

Bellek örnekleri

En büyük öğeyi bulma: **$O(1)$** ek bellek.

Birleştirme sıralaması: **$O(n)$** yardımcı dizi.

Özyineleme: çağrı yığını kadar bellek.

Zaman Karmaşıklığı

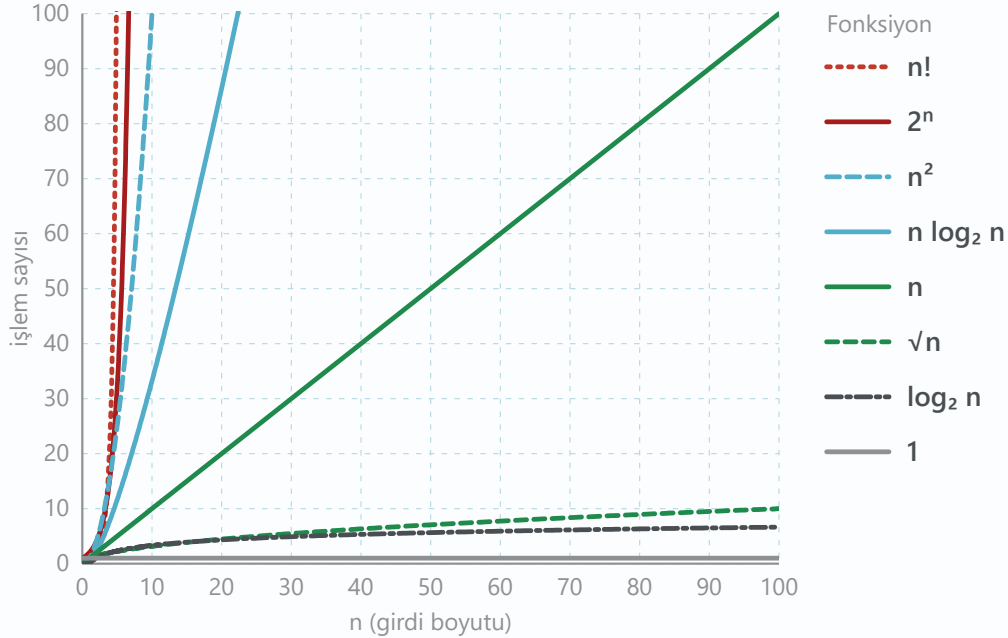
- Bir algoritmanın çalışma süresini **girdi boyutu n** ile ilişkilendiren kavramdır: **T(n)**.
- Saniye değil, **temel işlem sayısı** sayılır: saniye bilgisayara ve dile göre değişir, işlem sayısı değişmez.
- Veri büyüdükçe performansın **nasıl değiştiğini** inceler; bir algoritmanın iyi ya da kötü olduğunu belirlemede kritik faktördür.

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
```

$T(n) = 1 + (n + 1) + n + 1 = 2n + 3$. Sabitler makineye bağlıdır; önemli olan $T(n)$ 'nin **n ile doğrusal** büyümesidir $\rightarrow \Theta(n)$.

Fonksiyonların Büyüme Hızları



$n = 10$ 'da bile $n!$ (3,6 milyon) ve 2^n (1024) grafiğin dışına çıkar; $\log_2 n$ ve $\sqrt{n}$ ise $n = 100$ 'de hâlâ 10'un altındadır.

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
√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 · 10 ⁷	19,9 ms
n ²	100	10.000	1.000.000	1,0 · 10 ¹²	16,7 dk
2 ⁿ	1.024	1,3 · 10 ³⁰	≈ 1,1 · 10 ³⁰¹	≈ 10 ³⁰¹⁰³⁰	evrenin yaşından çok
n!	3.628.800	9,3 · 10 ¹⁵⁷	≈ 10 ²⁵⁶⁷	≈ 10 ⁵⁵⁶⁵⁷⁰⁸	evrenin yaşından çok

Son sütun: saniyede 10⁹ temel işlem yapan bir bilgisayarda n = 10⁶ için tahmini süre. 2ⁿ ve n! için n = 100 bile (≈ 10³⁰ ve 10¹⁵⁸ işlem) pratikte çözülemez.

Daha hızlı bir bilgisayar sabiti küçültür; **daha iyi bir algoritma büyüme hızını** değiştirir. n² yerine n log n, 10⁶ ögede dakikaları milisaniyelere indirir.

Deneyin: Büyüme Hesaplayıcı

n = 1000

10

100

 10^4 10^6

n × 2

f(n)	işlem sayısı	10^9 işlem/s ile süre	bağıl
1	1	1 ns	
$\log_2 n$	10	10 ns	
$\sqrt{n}$	31,6	31,6 ns	
n	1.000	1 μ s	
$n \log_2 n$	9.965,8	10 μ s	
n^2	1.000.000	1 ms	
n^3	$1 \cdot 10^9$	1 s	
2^n	$1,1 \cdot 10^{301}$	$\approx 10^{284}$ yıl	
$n!$	$4 \cdot 10^{2567}$	$\approx 10^{2551}$ yıl	

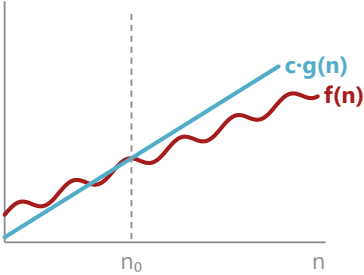
n = **1.000** · çubuk: işlem sayısının basamak sayısı (log ölçek) · kırmızı: 1 saatten uzun

"n × 2" düğmesiyle girdiyi ikiye katlayın: n sütunu 2 katına, n^2 4 katına, n^3 8 katına çıkar; $\log_2 n$ yalnızca 1 artar, 2^n ise karesini alır.

Asimptotik Gösterimler: O , Ω , Θ

Büyük O : üst sınır

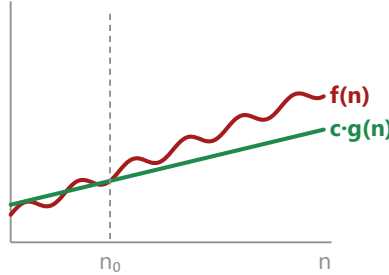
$f(n) = O(g(n))$: $n \geq n_0$ için $f(n) \leq c \cdot g(n)$. "En fazla bu kadar hızlı büyür."



Süre 100 s ile sınırlıysa daha kısa sürebilir, ama **daha uzun** süremez.

Omega Ω : alt sınır

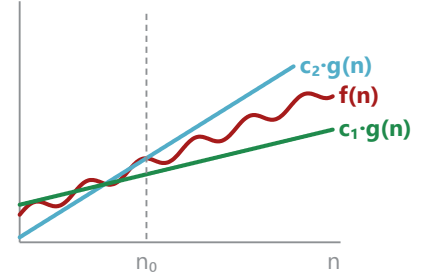
$f(n) = \Omega(g(n))$: $n \geq n_0$ için $f(n) \geq c \cdot g(n)$. "En az bu kadar hızlı büyür."



Alt sınır 100 s ise daha uzun sürebilir, ama **daha kısa** süremez.

Theta Θ : sıkı sınır

$f(n) = \Theta(g(n))$: hem $O(g(n))$ hem $\Omega(g(n))$. "Tam olarak bu hızda büyür."



Doğrusal aramanın en kötü durumu: $c_1 \cdot n \leq T(n) \leq c_2 \cdot n \rightarrow \Theta(n)$.

Dikkat: O , Ω , Θ **sınır** türleridir; en kötü / en iyi **durum** ile aynı şey değildir. Uygulamada O çoğunlukla en kötü durum süresi için kullanılır. Ayrıntılı tanım ve özellikler: Bölüm 1.2.

Doğrusal Arama (Linear Search)

```
int dogrusalAra(int[] dizi, int aranan) {  
    for (int i = 0; i < dizi.length; i++) {  
        if (dizi[i] == aranan) {  
            return i; // bulundu!  
        }  
    }  
    return -1; // bulunamadı  
}
```

Zaman karmaşıklığı

En kötü durumda (hedef yok ya da sonda) döngü n kez döner:

$$T(n) = c_1 \cdot n + c_2$$

c_1 ve c_2 bilgisayarın hızı, programlama dili gibi etkenlere bağlıdır.

$\Theta(n)$

Süre, girdi boyutu n ile **orantılı** büyür: yeterince büyük n için bazı $k_1, k_2 > 0$ sabitleriyle

$$k_1 \cdot n \leq T(n) \leq k_2 \cdot n.$$

En iyi durum: hedef ilk öge $\rightarrow$ 1 karşılaştırma, $\Theta(1)$.

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 da olsa 10 milyon öğeli da 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.

Logaritmik Zaman: İkili Arama, $O(\log n)$

```
int ikiliArama(int[] dizi, int aranan) {  
    int alt = 0, ust = dizi.length - 1;  
    while (alt <= ust) {  
        int orta = (alt + ust) / 2;  
        int tahmin = dizi[orta];  
        if (tahmin == aranan)  
            return orta;    // bulundu  
        else if (tahmin < aranan)  
            alt = orta + 1;  
        else  
            ust = orta - 1;  
    }  
    return -1;    // bulunamadı  
}
```

Dizi **sıralı** olmalıdır. Her adımda aranan aralık **yarıya** iner: $n \rightarrow n/2 \rightarrow n/4 \rightarrow \dots \rightarrow 1$.

n	en fazla adım
8	4
1.000	10
1.000.000	20
$1,0 \cdot 10^9$	30

En fazla $\lfloor \log_2 n \rfloor + 1$ adım $\rightarrow \mathbf{T(n) = O(\log n)}$. Adım adım örnek: Böl ve Fethet bölümünde.

Asal Sayı Kontrolü: $O(\sqrt{n})$

```
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 $\sqrt{n}$ 'e kadar?

$n = a \cdot b$ ise a ve b 'den biri $\sqrt{n}$ 'den **küçük ya da eşittir**. $\sqrt{n}$ 'e kadar bölen yoksa n asaldır.

Kaç bölme?

Çift sayılar baştan elenir; yalnızca tek i denenir: en fazla $\approx \sqrt{n} / 2$ bölme $\rightarrow \mathbf{T(n)}$
 $= \mathbf{O(\sqrt{n})}$.

$n = 97$: $i = 3, 5, 7, 9 \rightarrow \mathbf{4}$ 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.

$n \log n$ Zaman: Çabuk (Hızlı) Sıralama

```
List<Integer> hizliSiralama(List<Integer> liste) {  
    if (liste.size() <= 1)  
        return liste;  
    int pivot = liste.get(0);  
    List<Integer> kucukler = new ArrayList<>();  
    List<Integer> buyukler = new ArrayList<>();  
    for (int x : liste.subList(1, liste.size()))  
        if (x <= pivot) kucukler.add(x);  
        else buyukler.add(x);  
    List<Integer> sonuc = hizliSiralama(kucukler);  
    sonuc.add(pivot);  
    sonuc.addAll(hizliSiralama(buyukler));  
    return sonuc;  
}
```

İlk öge **pivot**; küçük-eşitler solda, büyükler sağda özyinelemeli sıralanır ve birleştirilir.

Ortalama: $O(n \log n)$

Pivot diziyi yaklaşık ikiye bölerse $\log_2 n$ düzey olur, her düzeyde toplam n karşılaştırma yapılır.

En kötü: $O(n^2)$

Dizi zaten sıralıysa ilk öge hep en küçüktür; bölünme $n-1 / 0$ olur: $(n-1) + (n-2) + \dots + 1 = n(n-1)/2$.

Hızlı sıralamanın $O(n \log n)$ süresi **ortalama** durumdur; rastgele pivot seçimi en kötü durumu olası olmaktan çıkarır (Bölüm 2).

Karesel Zaman: Kabarcık Sıralaması, $O(n^2)$

```
void kabarcikSiralama(int[] dizi) {  
    int n = dizi.length;  
    for (int i = 0; i < n; i++)  
        for (int j = 0; j < n - i - 1; j++)  
            if (dizi[j] > dizi[j + 1]) {  
                int gecici = dizi[j];    // takas  
                dizi[j] = dizi[j + 1];  
                dizi[j + 1] = gecici;  
            }  
}
```

Komşu öğeler karşılaştırılır, sırasızsa takas edilir. Her dış turda en büyük öğe sona "kabarcık gibi" çıkar.

Sayım

i. turda iç döngü $n - i - 1$ kez döner:
 $(n-1) + (n-2) + \dots + 1 + 0 = \mathbf{n(n-1)/2}$
karşılaştırma.

$n = 1000 \rightarrow 499.500$ karşılaştırma; n iki katına çıkınca süre $\approx$ **4 katına** çıkar.

$T(n) = O(n^2)$.

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: $A = [a_{ij}]$, $B = [b_{ij}]$ ($n \times n$). **Çıktı:** $C = [c_{ij}] = A \cdot B$, $i, j = 1, 2, \dots, n$.

Tanım

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

1	2	5	6	19	22
3	4	7	8	43	50

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

Faktöriyel Zaman: Tüm Permütasyonları Üretme

```
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  
    }  
}
```

$[1, 2, 3] \rightarrow 123, 132, 213, 231, 321, 312$. İlk konum için n , ikinci için $n-1$, ... seçenek: $n \cdot (n-1) \cdot \dots \cdot 1 = n!$ 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 $\Theta(n \cdot n!)$, 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) {        // ezberleme
    if (n <= 1) return n;
    if (bellek[n] == 0)
        bellek[n] = fibBellek(n - 1) + fibBellek(n - 2);
    return bellek[n];
}
```

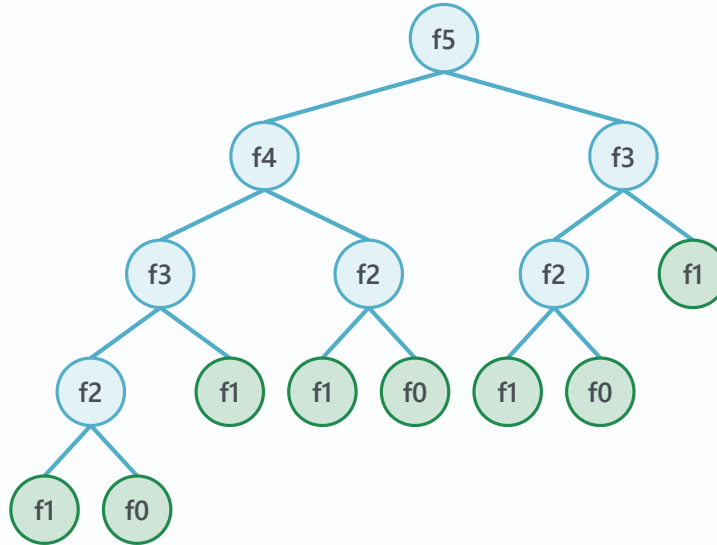
$F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2)$.

n	fib çağrısı	fibBellek
5	15	9
10	177	19
20	21.891	39
30	2.692.537	59

fib: her çağrı iki çağrı doğurur → **$O(2^n)$**
(kesin olarak $\Theta(1,618^n)$).

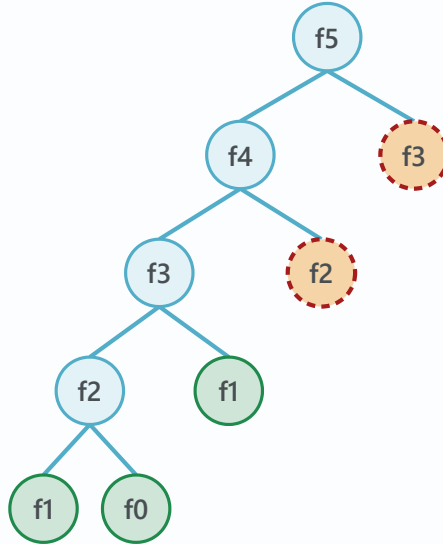
fibBellek: her $F(k)$ bir kez hesaplanır,
 $2n - 1$ çağrı → **$O(n)$** .

Fibonacci: Çağrı Ağacı fib(5)



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

Fibonacci: Çağrı Ağacı fib(5)



Ezberlemeli: **9 çağrı**. Turuncu kesikli düğümler (f2, f3) bellek dizisinden **hazır** döner, alt ağaçları hiç açılmaz $\rightarrow O(n)$.

KARMAŞIKLIK ÖRNEKLERİ

Karmaşıklık Sınıfları: Örneklerle

Sınıf	Adı	Örnek	n iki katına çıkınca süre
$O(1)$	sabit	ilk öğeyi yazdırma, dizi erişimi	değişmez
$O(\log n)$	logaritmik	ikili arama, hızlı üs alma, Euclid	+ 1 adım
$O(\sqrt{n})$	karekök	asal sayı kontrolü (deneme bölmesi)	$\approx 1,41$ katı
$O(n)$	doğrusal	doğrusal arama, en büyük öge, birleştirme	2 katı
$O(n \log n)$	log-doğrusal	birleştirme sıralaması, hızlı sıralama (ortalama)	2 katından biraz fazla
$O(n^2)$	karesel	kabarcık, eklemeli sıralama	4 katı
$O(n^3)$	kübik	klasik matris çarpımı	8 katı
$O(2^n)$	üstel	saf özyinelemeli Fibonacci, alt kümeler	karesi alınır
$O(n!)$	faktöriyel	tüm permütasyonlar	çok daha hızlı patlar

Yukarıdan aşağıya büyüme hızı artar: $1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < n!$. Polinom sınıflar (n^k) genelde "verimli", üstel olanlar "verimsiz" sayılır.

Sıralama Problemi

- **Girdi:** n sayıdan oluşan dizi $\langle a_1, a_2, \dots, a_n \rangle$.
- **Çıktı:** aynı sayıların bir permütasyonu $\langle a'_1, a'_2, \dots, a'_n \rangle$ öyle ki $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

Girdi:

8	2	4	9	3	6
---	---	---	---	---	---

Çıktı:

2	3	4	6	8	9
---	---	---	---	---	---

Neden sıralama?

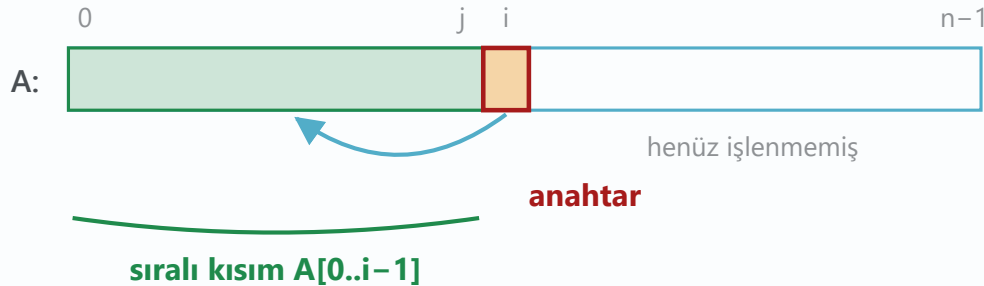
Arama (ikili arama sıralı dizi ister), tekrar bulma, sıralı rapor, birleştirme gibi pek çok işlemin ön adımıdır.

Örnek girdi

Bu bölüm boyunca aynı diziyi kullanacağız: **8 2 4 9 3 6**. Bölüm 2'de pek çok sıralama algoritması karşılaştırılır.

Ekleme Sıralama (Insertion Sort)

- Diziyi sıralamak için **iç içe iki döngü** kullanır.
- **Dış döngü** dizinin her ögesini bir kez geçer: $A[i]$ **anahtar** değişkenine alınır.
- **İç döngü** anahtardan büyük öğeleri birer sağa kaydırır ve anahtarı uygun konuma yerleştirir.
- İskambil kâğıtlarını elde sıralamak gibi: her yeni kâğıt, sıralı eldeki yerine sokulur.



Değişmez (invariant): her dış adımın başında $A[0..i-1]$ sıralıdır. Zaman karmaşıklığı (en kötü) $O(n^2)$.

Eklemeli Sıralama: Kod

```
public void Sirala(int[] A) {  
    for (int i = 1; i < A.length; ++i) {  
        int anahtar = A[i];  
        int j = i - 1;  
        // A[0..i-1] alt dizisi sıralı.  
        // anahtar uygun konuma yerleştirilir.  
        while (j >= 0 && A[j] > anahtar) {  
            A[j + 1] = A[j];  
            j = j - 1;  
        }  
        A[j + 1] = anahtar;  
    }  
}
```

Dış döngü: $n - 1$ adım

$i = 1, 2, \dots, n-1$. Her adımda bir öge sıralı kısma eklenir.

İç döngü: en fazla i adım

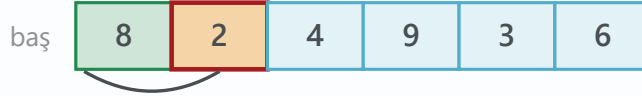
Anahtar, sıralı kısımdaki tüm öğelerden küçükse (dizi tersten sıralıysa) i öge kaydırılır.

Toplam (en kötü)

$1 + 2 + \dots + (n-1) = n(n-1)/2 \rightarrow \mathbf{O(n^2)}$.
Ek bellek $O(1)$: yerinde (in-place) sıralar.

EKLEMELİ SIRALAMA

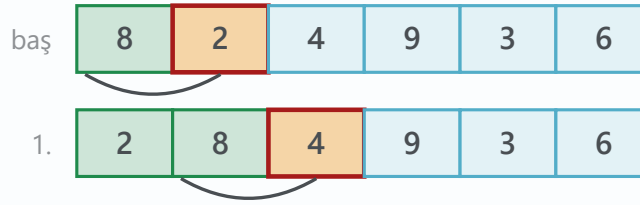
Örnek: 8 2 4 9 3 6



Başlangıç: 8 tek başına sıralıdır (yeşil). Anahtar **2**: 8'den küçük → 8 sağa kayar, 2 başa girer.

EKLEMELİ SIRALAMA

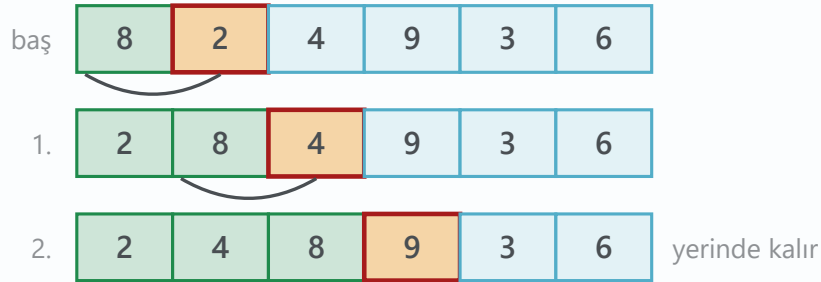
Örnek: 8 2 4 9 3 6



1. ekleme sonrası: 2 8 4 9 3 6. Sıradaki anahtar **4**: kendinden büyük 1 öğe sağa kayar, 1. konuma girer.

EKLEMELİ SIRALAMA

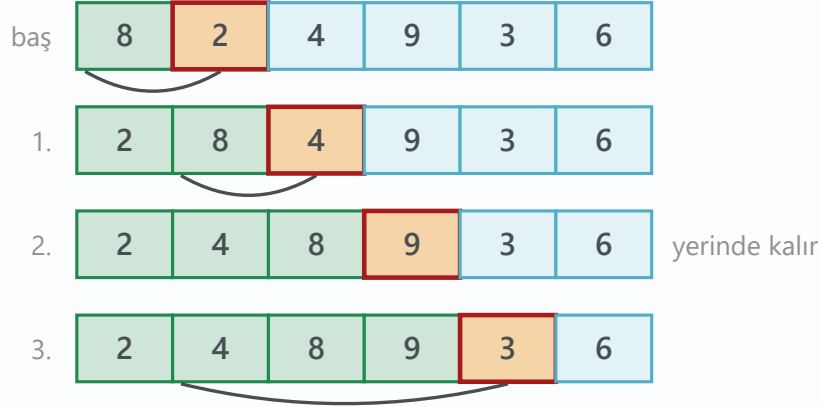
Örnek: 8 2 4 9 3 6



2. ekleme sonrası: 2 4 8 9 3 6. Sıradaki anahtar **9**: solundaki 8'ten büyük, yerinde kalır.

EKLEMELİ SIRALAMA

Örnek: 8 2 4 9 3 6



3. ekleme sonrası: 2 4 8 9 3 6. Sıradaki anahtar **3**: kendinden büyük 3 öğe sağa kayar, 1. konuma girer.

EKLEMELİ SIRALAMA

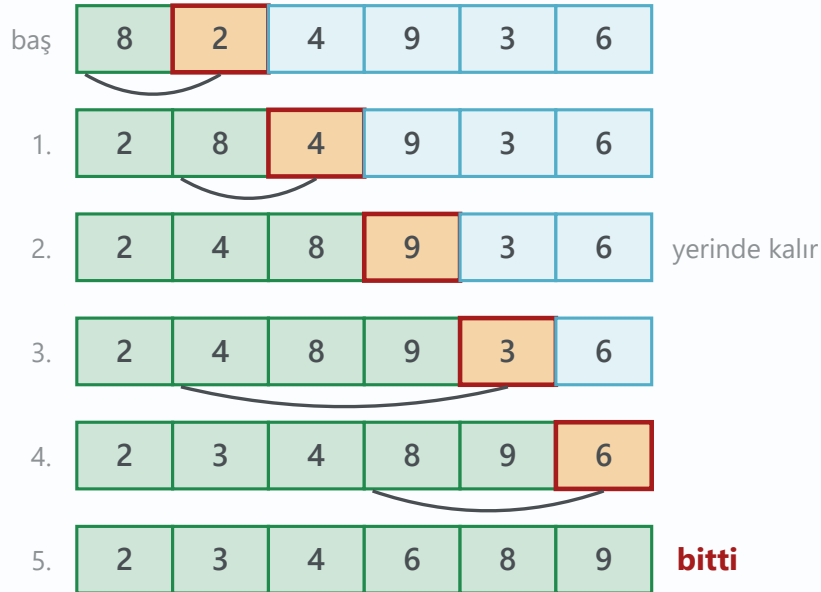
Örnek: 8 2 4 9 3 6



4. ekleme sonrası: 2 3 4 8 9 6. Sıradaki anahtar **6**: kendinden büyük 2 öğe sağa kayar, 3. konuma girer.

EKLEMELİ SIRALAMA

Örnek: 8 2 4 9 3 6



Son öge da yerine girdi: **2 3 4 6 8 9** — sıralama bitti ($n - 1 = 5$ dış adım).

Sirala(A): Kodu Adım Adım İzle

```
► public void Sirala(int[] A) {  
2   for (int i = 1; i < A.length; ++i) {  
3       int anahtar = A[i];  
4       int j = i - 1;  
5       // A[0..i-1] sıralı  
6       // anahtar yerine girer  
7       while (j >= 0 && A[j] > anahtar) {  
8           A[j + 1] = A[j];  
9           j = j - 1;  
10      }  
11      A[j + 1] = anahtar;  
12  }  
13 }
```

0	1	2	3	4	5
8	2	4	9	3	6

Dizi: 8 2 4 9 3 6. A[0] = 8 tek başına sıralı kabul edilir;
dış döngü i = 1'den başlar.

Adım 1 / 19

DEĞİŞKENLER

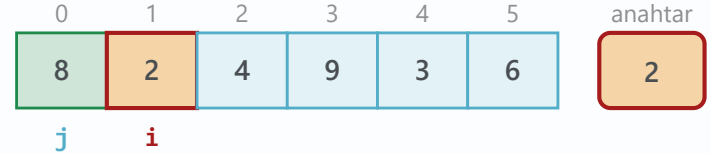
i = -

anahtar = -

j = -

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         ▶ int anahtar = A[i];  
4         ▶ int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



i = 1: anahtar = A[1] = 2, j = 0. Sıralı kısım: 8.

Adım 2 / 19

DEĞİŞKENLER

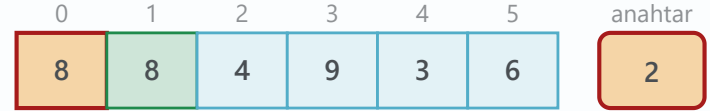
i = 1

anahtar = 2

j = 0

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1; j=-1  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[0] = 8 > 2 \rightarrow A[1] = A[0]$ (sağa kaydır), $j = -1$.

Adım 3 / 19

DEĞİŞKENLER

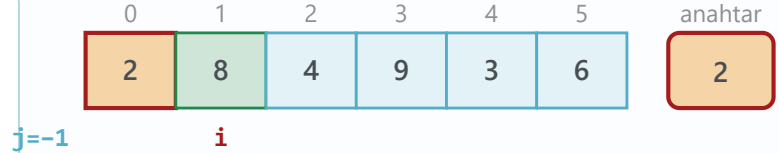
$i = 1$

anahtar = 2

$j = -1$

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



Koşul yanlış: $j = -1$ (başa gelindi). $A[0] = \text{anahtar} = 2$. $A[0..1]$ sıralı: 2 8.

Adım 4 / 19

DEĞİŞKENLER

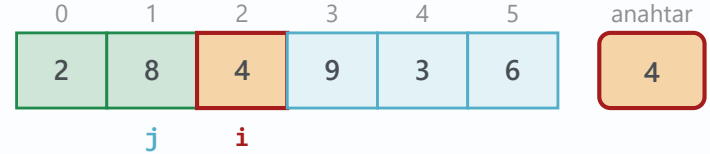
$i = 1$

$\text{anahtar} = 2$

$j = -1$

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2   for (int i = 1; i < A.length; ++i) {  
3     ▶ int anahtar = A[i];  
4     ▶ int j = i - 1;  
5     // A[0..i-1] sıralı  
6     // anahtar yerine girer  
7     while (j >= 0 && A[j] > anahtar) {  
8       A[j + 1] = A[j];  
9       j = j - 1;  
10    }  
11    A[j + 1] = anahtar;  
12  }  
13 }
```



i = 2: anahtar = A[2] = 4, j = 1. Sıralı kısım: 2 8.

Adım 5 / 19

DEĞİŞKENLER

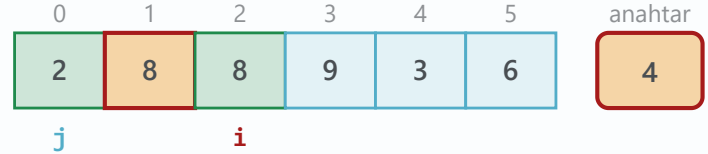
i = 2

anahtar = 4

j = 1

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[1] = 8 > 4 \rightarrow A[2] = A[1]$ (sağa kaydır), $j = 0$.

Adım 6 / 19

DEĞİŞKENLER

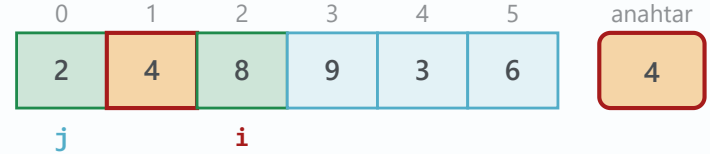
i = 2

anahtar = 4

j = 0

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



Koşul yanlış: $A[0] = 2 \leq 4$. $A[1] = \text{anahtar} = 4$.
 $A[0..2]$ sıralı: 2 4 8.

Adım 7 / 19

DEĞİŞKENLER

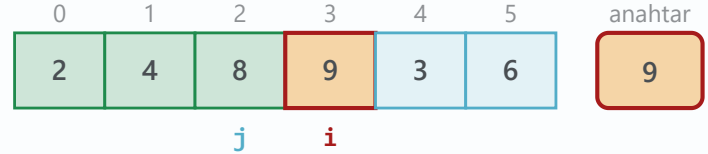
i = 2

anahtar = 4

j = 0

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2   for (int i = 1; i < A.length; ++i) {  
3     ▶ int anahtar = A[i];  
4     ▶ int j = i - 1;  
5     // A[0..i-1] sıralı  
6     // anahtar yerine girer  
7     while (j >= 0 && A[j] > anahtar) {  
8       A[j + 1] = A[j];  
9       j = j - 1;  
10    }  
11    A[j + 1] = anahtar;  
12  }  
13 }
```



i = 3: anahtar = A[3] = 9, j = 2. Sıralı kısım: 2 4 8.

Adım 8 / 19

DEĞİŞKENLER

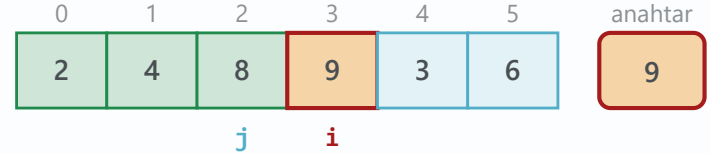
i = 3

anahtar = 9

j = 2

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



Koşul yanlış: $A[2] = 8 \leq 9$. $A[3] = \text{anahtar} = 9$.
 $A[0..3]$ sıralı: 2 4 8 9.

Adım 9 / 19

DEĞİŞKENLER

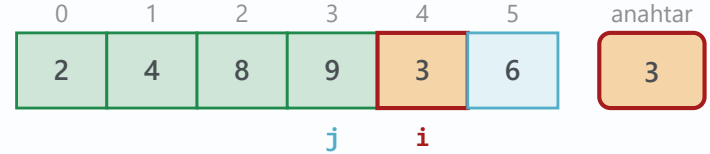
i = 3

anahtar = 9

j = 2

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2   for (int i = 1; i < A.length; ++i) {  
3     ▶ int anahtar = A[i];  
4     ▶ int j = i - 1;  
5     // A[0..i-1] sıralı  
6     // anahtar yerine girer  
7     while (j >= 0 && A[j] > anahtar) {  
8       A[j + 1] = A[j];  
9       j = j - 1;  
10    }  
11    A[j + 1] = anahtar;  
12  }  
13 }
```



i = 4: anahtar = A[4] = 3, j = 3. Sıralı kısım: 2 4 8 9.

Adım 10 / 19

DEĞİŞKENLER

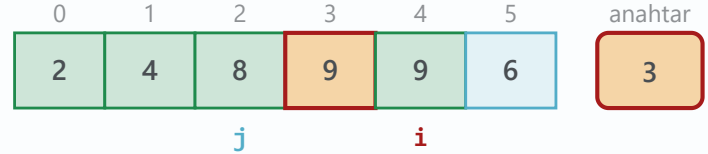
i = 4

anahtar = 3

j = 3

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[3] = 9 > 3 \rightarrow A[4] = A[3]$ (sağa kaydır), $j = 2$.

Adım 11 / 19

DEĞİŞKENLER

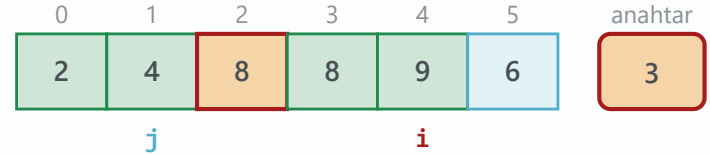
i = 4

anahtar = 3

j = 2

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[2] = 8 > 3 \rightarrow A[3] = A[2]$ (sağa kaydır), $j = 1$.

Adım 12 / 19

DEĞİŞKENLER

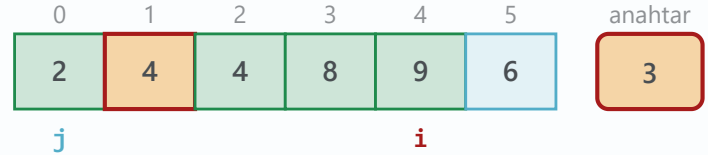
i = 4

anahtar = 3

j = 1

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[1] = 4 > 3 \rightarrow A[2] = A[1]$ (sağa kaydır), $j = 0$.

Adım 13 / 19

DEĞİŞKENLER

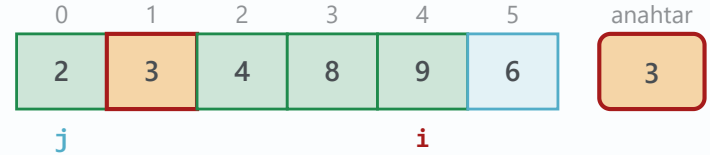
$i = 4$

$anahtar = 3$

$j = 0$

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



Koşul yanlış: $A[0] = 2 \leq 3$. $A[1] = \text{anahtar} = 3$.
A[0..4] sıralı: 2 3 4 8 9.

Adım 14 / 19

DEĞİŞKENLER

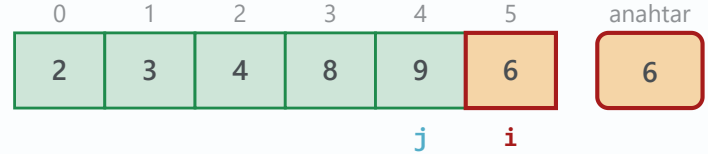
$i = 4$

$\text{anahtar} = 3$

$j = 0$

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         ▶ int anahtar = A[i];  
4         ▶ int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



i = 5: anahtar = A[5] = 6, j = 4. Sıralı kısım: 2 3 4 8 9.

Adım 15 / 19

DEĞİŞKENLER

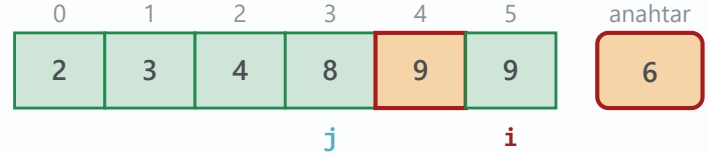
i = 5

anahtar = 6

j = 4

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[4] = 9 > 6 \rightarrow A[5] = A[4]$ (sağa kaydır), $j = 3$.

Adım 16 / 19

DEĞİŞKENLER

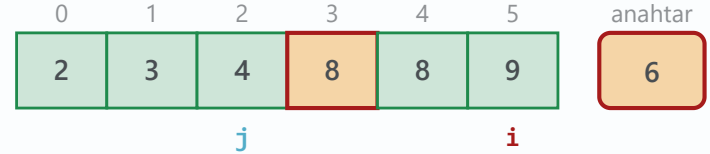
i = 5

anahtar = 6

j = 3

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             ▶ A[j + 1] = A[j];  
9             ▶ j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



$A[3] = 8 > 6 \rightarrow A[4] = A[3]$ (sağa kaydır), $j = 2$.

Adım 17 / 19

DEĞİŞKENLER

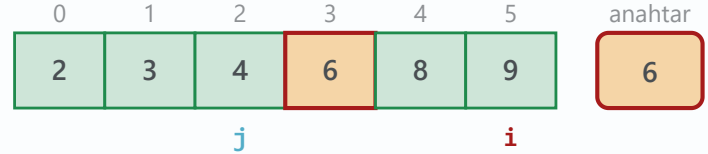
i = 5

anahtar = 6

j = 2

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```



Koşul yanlış: $A[2] = 4 \leq 6$. $A[3] = \text{anahtar} = 6$.
 $A[0..5]$ sıralı: 2 3 4 6 8 9.

Adım 18 / 19

DEĞİŞKENLER

i = 5

anahtar = 6

j = 2

Sirala(A): Kodu Adım Adım İzle

```
1 public void Sirala(int[] A) {  
2     for (int i = 1; i < A.length; ++i) {  
3         int anahtar = A[i];  
4         int j = i - 1;  
5         // A[0..i-1] sıralı  
6         // anahtar yerine girer  
7         while (j >= 0 && A[j] > anahtar) {  
8             A[j + 1] = A[j];  
9             j = j - 1;  
10        }  
11        A[j + 1] = anahtar;  
12    }  
13 }
```

0	1	2	3	4	5
2	3	4	6	8	9

$i = 6 = A.length \rightarrow$ döngü biter. Sonuç: **2 3 4 6 8 9**.
Toplam 11 karşılaştırma, 7 kaydırma.

Adım 19 / 19

DEĞİŞKENLER

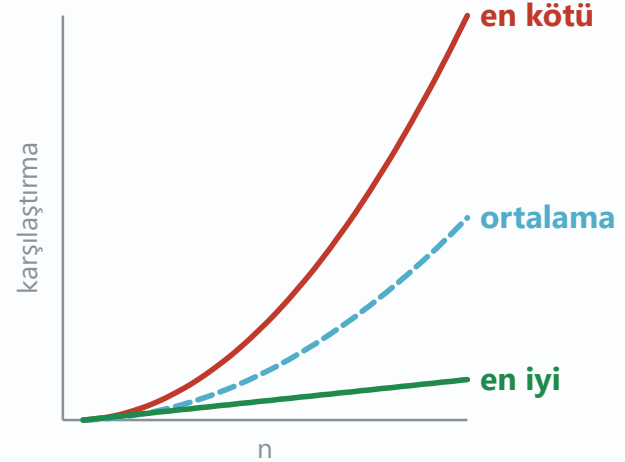
i = 6

anahtar = -

j = -

Çalışma Süresi ve Analiz Türleri

- **Girdi durumuna bağlı:** dizi zaten sıralıysa sıralamak kolaydır.
- **Girdi boyutuna bağlı:** az öğeli dizi kısa sürede sıralanır $\rightarrow T(n)$.
- **En kötü durum** (genellikle): n boyutlu herhangi bir girdi için **maksimum** süre. Herkes garanti ister.
- **Ortalama durum** (bazen): tüm girdiler üzerinden **beklenen** süre; girdi dağılımı hakkında varsayım gerekir.
- **En iyi durum** (yanıltıcı): özel, "hileli" bir girdide geçen süre; tek başına bir şey kanıtlamaz.



Eklemeli sıralama: **sıralı** girdi $\rightarrow n-1$ karşılaştırma; **tersten sıralı** girdi $\rightarrow n(n-1)/2$.
Örnek dizimizde: 11 karşılaştırma.

Dikkat: Durum ve Sınır Farklı Kavramlardır

Durum (case)

Hangi **girdiye** bakıyoruz? En kötü, ortalama, en iyi girdi. Her durumun kendi süre fonksiyonu vardır.

Sınır (bound)

Bir fonksiyonu nasıl **özetliyoruz**? O : üst sınır, Ω : alt sınır, Θ : ikisi birden. Her duruma uygulanabilir.

Eklemeli sıralama	Sıkı sınır	Yorum
En iyi durum (sıralı girdi)	$\Theta(n)$	$n-1$ karşılaştırma
En kötü durum (ters sıralı girdi)	$\Theta(n^2)$	$n(n-1)/2$ karşılaştırma
Tüm girdiler için	$O(n^2)$ ve $\Omega(n)$	tek bir Θ yazılamaz

Dikkat: "En kötü $O(n \log n)$, en iyi $\Omega(n)$ ise çalışma süresi $\Theta(n \log n)$ " çıkarımı yanlıştır. Θ ancak alt ve üst sınır **aynı** fonksiyon olduğunda yazılır; ör. birleştirme sıralaması her girdide $\Theta(n \log n)$ 'dir.

Makineden Bağımsız Çalışma Süresi

- Eklemeli sıralamanın en kötü durumu $O(n^2)$: girdi tersten sıralıysa gerçekleşir.
- Bilgisayarın hızına göre süre değişir; **asimptotik analiz** büyüme oranına odaklanır, makineye bağlı **sabit çarpanlar** göz ardı edilir.
- $n \rightarrow \infty$ giderken $T(n)$ 'nin büyüme oranına bakılır: yalnızca **en önemli** (üssü en büyük) terim değerlendirilir.
- $T(n) = 3n^2 + 2n + 5 = O(n^2)$

n	$3n^2+2n+5$	$3n^2$	oran
10	325	300	1,0833
100	30.205	30.000	1,0068
1.000	3.002.005	3.000.000	1,0007

n büyüdükçe düşük dereceli terimlerin payı sıfıra yaklaşır. 3 katsayısı ise makineye bağlıdır: hızlı bilgisayarda 1, yavaşta 10 olabilir → atılır.

Θ Gösterimi: Tanım ve Uygulama

$\Theta(g(n)) = \{ f(n) : \text{tüm } n \geq n_0 \text{ değerleri için } 0 \leq c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n) \text{ şartını sağlayan pozitif } c_1, c_2 \text{ ve } n_0 \text{ sabitleri vardır} \}$

Mühendislik kuralı:

1. $T(n) = 3n^3 + 90n^2 - 5n + 6046$
2. Düşük kuvvetli terimleri sil: $3n^3$
3. Katsayıları kaldır: **$T(n) = \Theta(n^3)$**

Tanımla doğrulama

$c_1 = 3$: $3n^3 \leq T(n)$, çünkü $90n^2 - 5n + 6046 > 0$ (her $n \geq 1$).

$c_2 = 4$: $T(n) \leq 4n^3 \Leftrightarrow 90n^2 - 5n + 6046 \leq n^3$,
bu da **$n \geq 91$** için doğrudur.

$\Rightarrow c_1 = 3, c_2 = 4, n_0 = 91$ ile $T(n) = \Theta(n^3)$.

O ve Ω 'nın biçimsel tanımları, limit yöntemi ve özellikleri Bölüm 1.2: Karmaşıklık'ta ele alınır.

Böl ve Fethet (Divide and Conquer)

1. Böl (Divide)

Problemi aynı türden, daha küçük alt problemlere ayır.

2. Fethet (Conquer)

Alt problemleri özyinelemeli çöz; yeterince küçükse doğrudan çöz.

3. Birleştir (Combine)

Alt çözümleri birleştirerek asıl problemin çözümünü elde et.

Algoritma	Böl	Fethet	Birleştir	T(n)
Birleştirme sıralaması	ortadan ikiye	2 yarıyı sırala	birleştirme: $\Theta(n)$	$2T(n/2) + \Theta(n)$
İkili arama	ortayı kontrol et	1 yarıda ara	önemsiz: $\Theta(1)$	$T(n/2) + \Theta(1)$
Üs alma a^n	n'yi ikiye böl	$a^{\lfloor n/2 \rfloor}$ hesapla	kare al: $\Theta(1)$	$T(n/2) + \Theta(1)$

Böyle özyinelemeli tanımlanan süre fonksiyonlarına **özyineleme bağıntısı** (recurrence) denir. Çözüm yöntemleri (özyineleme ağacı, ana teorem) Bölüm 1.2'de.

Birleştirme Sıralaması (Merge Sort)

Adım	Maliyet
BirleştirSırala A[1 .. n]	T(n)
1. $n = 1$ ise işlem tamam.	
2. liste1 = BirleştirSırala A[1 .. $\lfloor n/2 \rfloor$]	
3. liste2 = BirleştirSırala A[$\lfloor n/2 \rfloor + 1$.. n]	
4. İki sıralı listeyi birleştir.	

Birleştirme sıralaması (Merge Sort): n öğeli diziyi sıralamanın süresi **T(n)** olsun.

Birleştirme Sıralaması (Merge Sort)

Adım	Maliyet
BirleştirSırala A[1 .. n]	$T(n)$
1. $n = 1$ ise işlem tamam.	$\Theta(1)$
2. liste1 = BirleştirSırala A[1 .. $\lfloor n/2 \rfloor$]	
3. liste2 = BirleştirSırala A[$\lfloor n/2 \rfloor + 1$.. n]	
4. İki sıralı listeyi birleştir.	

$n = 1$ ise dizi zaten sıralıdır: sabit süre, **$\Theta(1)$** .

Birleştirme Sıralaması (Merge Sort)

Adım	Maliyet
BirleştirSırala A[1 .. n]	$T(n)$
1. $n = 1$ ise işlem tamam.	$\Theta(1)$
2. liste1 = BirleştirSırala A[1 .. $\lfloor n/2 \rfloor$]	$T(n/2)$
3. liste2 = BirleştirSırala A[$\lfloor n/2 \rfloor + 1$.. n]	
4. İki sıralı listeyi birleştir.	

Sol yarıyı sıralamak, aynı problemin $n/2$ boyutlu hâlidir: **$T(n/2)$** .

Birleştirme Sıralaması (Merge Sort)

Adım	Maliyet
BirleştirSırala A[1 .. n]	$T(n)$
1. $n = 1$ ise işlem tamam.	$\Theta(1)$
2. liste1 = BirleştirSırala A[1 .. $\lfloor n/2 \rfloor$]	$T(n/2)$
3. liste2 = BirleştirSırala A[$\lfloor n/2 \rfloor + 1$.. n]	$T(n/2)$
4. İki sıralı listeyi birleştir.	

Sağ yarı da **$T(n/2)$** sürer.

Birleştirme Sıralaması (Merge Sort)

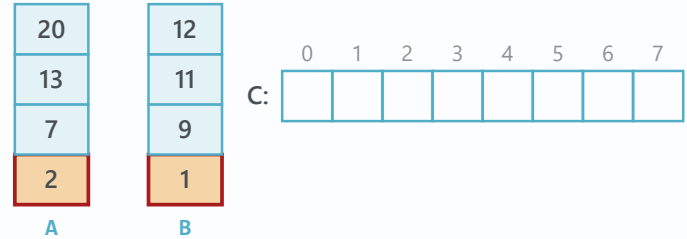
Adım	Maliyet
BirleştirSırala A[1 .. n]	$T(n)$
1. $n = 1$ ise işlem tamam.	$\Theta(1)$
2. liste1 = BirleştirSırala A[1 .. $\lfloor n/2 \rfloor$]	$T(n/2)$
3. liste2 = BirleştirSırala A[$\lfloor n/2 \rfloor + 1$.. n]	$T(n/2)$
4. İki sıralı listeyi birleştir.	$\Theta(n)$

$$T(n) = \Theta(1), n = 1 \text{ ise}; \quad T(n) = 2T(n/2) + \Theta(n), n > 1 \text{ ise.}$$

İki sıralı listeyi birleştirmek her öğeyi bir kez yazar: **$\Theta(n)$** . Böylece $T(n) = 2T(n/2) + \Theta(n)$.

Birleştirme İşlemi (Merge): Adım Adım

```
► int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3   if (A[i] <= B[j])
4     C[k++] = A[i++];
5   else
6     C[k++] = B[j++];
7 }
8 while (i < A.length)
9   C[k++] = A[i++];
10 while (j < B.length)
11   C[k++] = B[j++];
```



İki sıralı liste: A = 2 7 13 20, B = 1 9 11 12 (en küçükler altta). Yalnızca iki listenin **en önündeki** (sarı) öğeler karşılaştırılır.

Adım 1 / 9

DEĞİŞKENLER

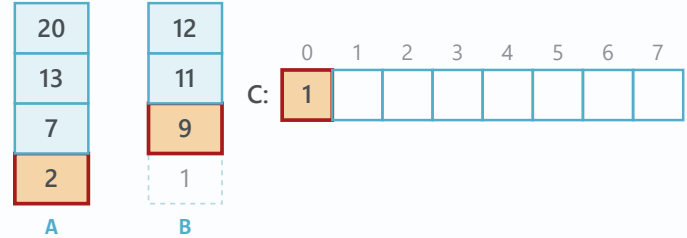
i = 0

j = 0

k = 0

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



$A[0] = 2 > B[0] = 1 \rightarrow 1$, C'ye yazılır; j ilerler.

Adım 2 / 9

DEĞİŞKENLER

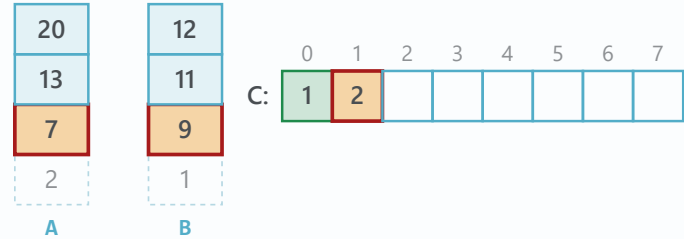
i = 0

j = 1

k = 1

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



$A[0] = 2 \leq B[1] = 9 \rightarrow 2$, C'ye yazılır; i ilerler.

Adım 3 / 9

DEĞİŞKENLER

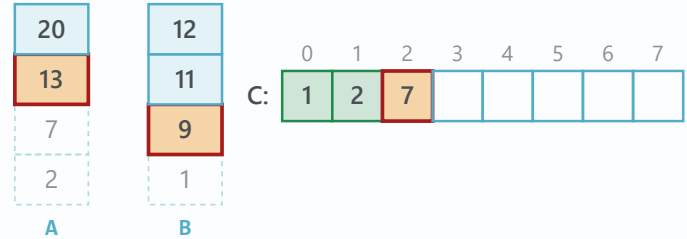
i = 1

j = 1

k = 2

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



$A[1] = 7 \leq B[1] = 9 \rightarrow 7$, C'ye yazılır; i ilerler.

Adım 4 / 9

DEĞİŞKENLER

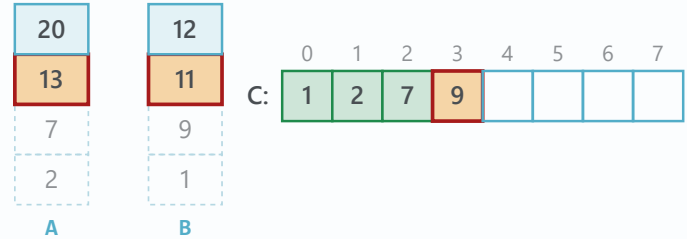
i = 2

j = 1

k = 3

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



$A[2] = 13 > B[1] = 9 \rightarrow 9$, C'ye yazılır; j ilerler.

Adım 5 / 9

DEĞİŞKENLER

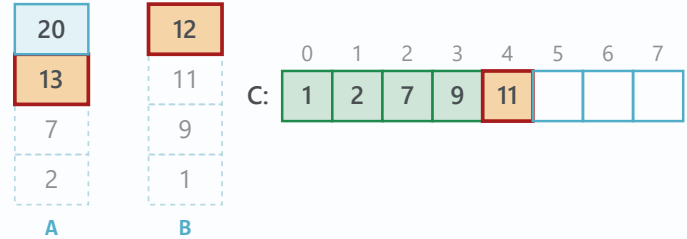
i = 2

j = 2

k = 4

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



$A[2] = 13 > B[2] = 11 \rightarrow 11$, C'ye yazılır; j ilerler.

Adım 6 / 9

DEĞİŞKENLER

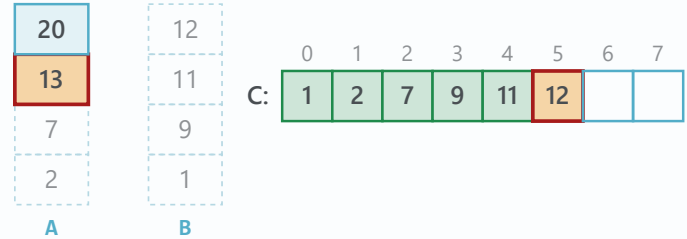
i = 2

j = 3

k = 5

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



$A[2] = 13 > B[3] = 12 \rightarrow 12$, C'ye yazılır; j ilerler.

Adım 7 / 9

DEĞİŞKENLER

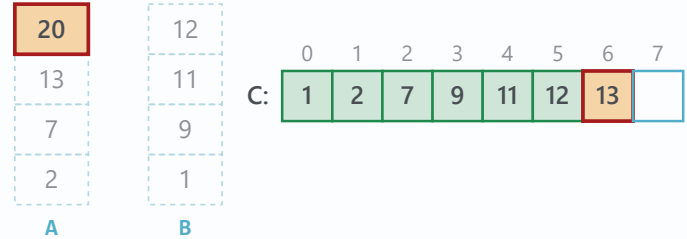
i = 2

j = 4

k = 6

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
8 while (i < A.length)
9     C[k++] = A[i++];
10 while (j < B.length)
11     C[k++] = B[j++];
```



B bitti: A'da kalan 13 doğrudan C'ye kopyalanır (karşılaştırma yok).

Adım 8 / 9

DEĞİŞKENLER

i = 3

j = 4

k = 7

Birleştirme İşlemi (Merge): Adım Adım

```
1 int i = 0, j = 0, k = 0;
2 while (i < A.length && j < B.length) {
3     if (A[i] <= B[j])
4         C[k++] = A[i++];
5     else
6         C[k++] = B[j++];
7 }
▶ while (i < A.length)
▶   C[k++] = A[i++];
10 while (j < B.length)
11   C[k++] = B[j++];
```

20
13
7
2

A

12
11
9
1

B

	0	1	2	3	4	5	6	7
C:	1	2	7	9	11	12	13	20

B bitti: A'da kalan 20 doğrudan C'ye kopyalanır (karşılaştırma yok). Sonuç: **1 2 7 9 11 12 13 20**. Her öge bir kez yazıldı → $\Theta(n)$.

Adım 9 / 9

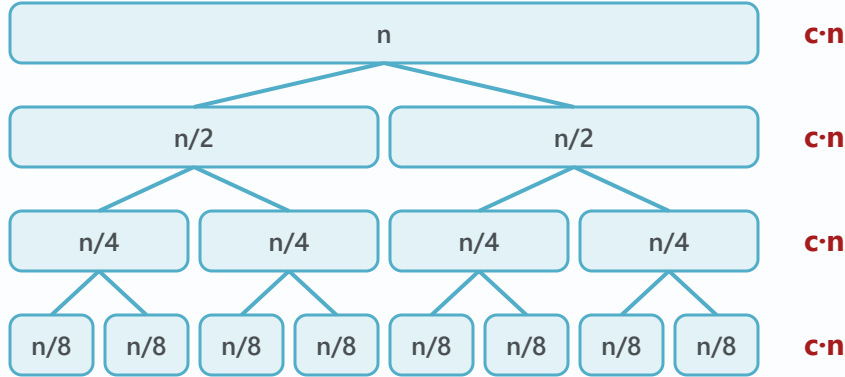
DEĞİŞKENLER

i = 4

j = 4

k = 8

Birleştirme Sıralaması: Neden $\Theta(n \log n)$?



$$\log_2 n + 1 \text{ düzey} \times c \cdot n \text{ iş} \Rightarrow T(n) = \Theta(n \log n)$$

- n öğeli listeyi birleştirmek $\Theta(n)$ sürer (yukarıdaki örnekte 8 öğe, 8 yazma, 6 karşılaştırma).
- Her düzeyde alt problemlerin toplam boyutu n 'dir $\rightarrow$ her düzeyin toplam birleştirme işi $c \cdot n$.
- Boyut her düzeyde yarıya iner: 1'e inmek için $\log_2 n$ bölme gerekir.
- Eklemeli sıralamanın $\Theta(n^2)$ en kötü durumuna karşı, birleştirme sıralaması **her girdide** $\Theta(n \log n)$ 'dir; bedeli $O(n)$ ek bellektir.

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {  
▶ 2     int alt = 0, ust = dizi.length - 1;  
3     while (alt <= ust) {  
4         int orta = (alt + ust) / 2;  
5         int tahmin = dizi[orta];  
6         if (tahmin == aranan)  
7             return orta;    // bulundu  
8         else if (tahmin < aranan)  
9             alt = orta + 1;  
10        else  
11            ust = orta - 1;  
12    }  
13    return -1;                // bulunamadı  
14 }
```

0	1	2	3	4	5	6	
3	5	7	8	9	12	15	hedef: 9
alt						ust	

Örnek: **9'u bul**. Dizi sıralı; aralık tüm dizi: alt = 0, ust = 6.

Adım 1 / 7

DEĞİŞKENLER

alt = 0

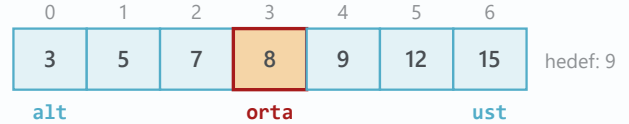
ust = 6

orta = -

tahmin = -

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {  
2   int alt = 0, ust = dizi.length - 1;  
3   while (alt <= ust) {  
4     int orta = (alt + ust) / 2;  
5     int tahmin = dizi[orta];  
6     if (tahmin == aranan)  
7       return orta;    // bulundu  
8     else if (tahmin < aranan)  
9       alt = orta + 1;  
10    else  
11      ust = orta - 1;  
12  }  
13  return -1;          // bulunamadı  
14 }
```



$orta = (0 + 6) / 2 = 3$, tahmin = dizi[3] = 8.

Adım 2 / 7

DEĞİŞKENLER

alt = 0

ust = 6

orta = 3

tahmin = 8

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {  
2     int alt = 0, ust = dizi.length - 1;  
3     while (alt <= ust) {  
4         int orta = (alt + ust) / 2;  
5         int tahmin = dizi[orta];  
6         if (tahmin == aranan)  
7             return orta;    // bulundu  
8         else if (tahmin < aranan)  
9             alt = orta + 1;  
10        else  
11            ust = orta - 1;  
12    }  
13    return -1;    // bulunamadı  
14 }
```



$8 < 9 \rightarrow$ hedef sağ yarıda: **alt = 4**. Sol yarı elendi.

Adım 3 / 7

DEĞİŞKENLER

alt = 4

ust = 6

orta = 3

tahmin = 8

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {  
2   int alt = 0, ust = dizi.length - 1;  
3   while (alt <= ust) {  
4     int orta = (alt + ust) / 2;  
5     int tahmin = dizi[orta];  
6     if (tahmin == aranan)  
7       return orta;    // bulundu  
8     else if (tahmin < aranan)  
9       alt = orta + 1;  
10    else  
11      ust = orta - 1;  
12  }  
13  return -1;          // bulunamadı  
14 }
```



$orta = (4 + 6) / 2 = 5$, tahmin = dizi[5] = 12 .

Adım 4 / 7

DEĞİŞKENLER

alt = 4

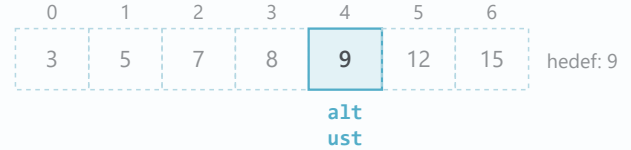
ust = 6

orta = 5

tahmin = 12

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {
2   int alt = 0, ust = dizi.length - 1;
3   while (alt <= ust) {
4     int orta = (alt + ust) / 2;
5     int tahmin = dizi[orta];
6     if (tahmin == aranan)
7       return orta;    // bulundu
8     else if (tahmin < aranan)
9       alt = orta + 1;
10    else
11      ust = orta - 1;
12  }
13  return -1;          // bulunamadı
14 }
```



12 > 9 → hedef sol yarıda: **ust = 4** . Sağ taraf elendi.

Adım 5 / 7

DEĞİŞKENLER

alt = 4

ust = 4

orta = 5

tahmin = 12

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {  
2   int alt = 0, ust = dizi.length - 1;  
3   while (alt <= ust) {  
4     int orta = (alt + ust) / 2;  
5     int tahmin = dizi[orta];  
6     if (tahmin == aranan)  
7       return orta;    // bulundu  
8     else if (tahmin < aranan)  
9       alt = orta + 1;  
10    else  
11      ust = orta - 1;  
12  }  
13  return -1;          // bulunamadı  
14 }
```

0	1	2	3	4	5	6	
3	5	7	8	9	12	15	hedef: 9

alt
ust
orta

$orta = (4 + 4) / 2 = 4$, tahmin =
dizi[4] = 9 .

Adım 6 / 7

DEĞİŞKENLER

alt = 4

ust = 4

orta = 4

tahmin = 9

İkili Arama: 9'u Bul

```
1 int ikiliArama(int[] dizi, int aranan) {  
2     int alt = 0, ust = dizi.length - 1;  
3     while (alt <= ust) {  
4         int orta = (alt + ust) / 2;  
5         int tahmin = dizi[orta];  
6         ▶ if (tahmin == aranan)  
7             ▶ return orta;    // bulundu  
8     else if (tahmin < aranan)  
9         alt = orta + 1;  
10    else  
11        ust = orta - 1;  
12    }  
13    return -1;    // bulunamadı  
14 }
```



alt
ust
orta

tahmin == aranan → **return 4**. Yalnızca 3 karşılaştırma ($\lfloor \log_2 7 \rfloor + 1 = 3$).

Adım 7 / 7

DEĞİŞKENLER

alt = 4

ust = 4

orta = 4

tahmin = 9

İkili Arama: Böl ve Fethet Analizi

İkili arama (Binary Search): sıralı listede bir öğeyi arar	Maliyet
Böl (Divide): orta öğeyi kontrol et.	$\Theta(1)$
Fethet (Conquer): seçilen alt diziyi özyinelemeli olarak ara.	$T(n/2)$
Birleştir (Combine): önemsiz (trivial); alt dizinin cevabı asıl cevaptır.	$\Theta(1)$

$$T(n) = 1 \cdot T(n/2) + \Theta(1)$$

alt problem sayısı

alt problem büyüklüğü

bölme + birleştirme

Açılım: $T(n) = T(n/2) + c = T(n/4) + 2c = \dots = T(1) + c \cdot \log_2 n \Rightarrow T(n) = \Theta(\log n)$

Bir Sayının Üssünü Alma: a^n

Problem: a^n 'i hesapla ($n \in \mathbb{N}$). **Doğal algoritma:** $a^n = a \cdot a \cdot \dots \cdot a \rightarrow \Theta(n)$ çarpma.

Böl ve fethet:

$$a^n = a^{n/2} \cdot a^{n/2} \quad (n \text{ çift ise})$$
$$a^n = a^{(n-1)/2} \cdot a^{(n-1)/2} \cdot a \quad (n \text{ tek ise})$$

```
long us(long a, int n) {  
    if (n == 0) return 1;  
    long yarim = us(a, n / 2);           // a^(n/2), tek kez!  
    if (n % 2 == 0)  
        return yarim * yarim;           // n çift  
    else  
        return a * yarim * yarim;       // n tek  
}
```

n	doğal	böl ve fethet
10	10	6
62	62	11
1.000	1.000	16
1.000.000	1.000.000	27

çarpma sayıları (yukarıdaki kodla).

Örn. 2^{10} : $n = 10 \rightarrow 5 \rightarrow 2 \rightarrow 1 \rightarrow 0$.

$$T(n) = T(n/2) + \Theta(1)$$

$$\Rightarrow T(n) = \Theta(\lg n)$$

Euclid Algoritması (EBOB)

- **En büyük ortak böleni** (EBOB, gcd) bulur.
 - Bilinen en eski algoritmalarından biridir (Öklid, MÖ ~300); en basit EBOB yöntemi.
 - Temel fikir: **$\text{ebob}(a, b) = \text{ebob}(b, a \bmod b)$** , $\text{ebob}(a, 0) = a$.
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?

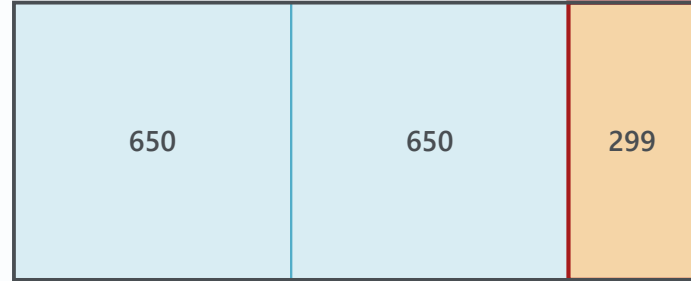
$a = q \cdot b + r$ ise a ile b 'yi bölen her sayı $r = a - q \cdot b$ 'yi de böler; b ile r 'yi bölen her sayı a 'yı da böler. Ortak bölenler aynıdır.

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)



$1599 = 2 \cdot 650 + 299$. 1599×650 dikdörtgendenden 2 kare kesilir, kalan şerit 650×299 → **ebob(650, 299)**.

Adım 1 / 7

DEĞİŞKENLER

a = 1599

b = 650

a%b = 299

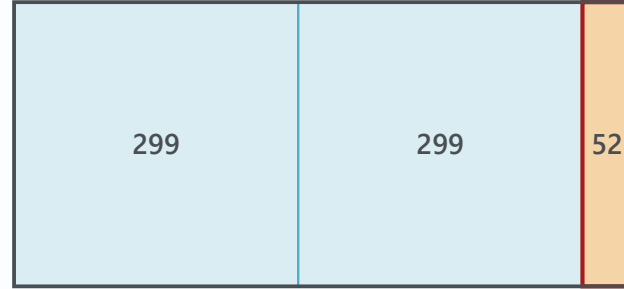
ebob(1599, 650): Adım Adım

```
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)



$650 = 2 \cdot 299 + 52$. 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

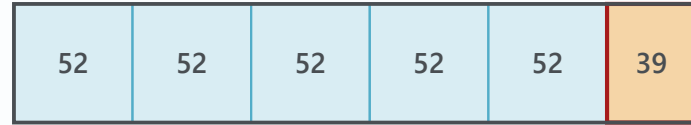
```
1 int ebob(int a, int b) {
  ► if (b == 0)
    3 return a;
  ► return ebob(b, a % b);
  5 }
```

ÇAĞRI YİĞİNİ

ebob(299, 52)

ebob(650, 299)

ebob(1599, 650)



299 × 52 dikdörtgen

$299 = 5 \cdot 52 + 39$. 299×52 dikdörtgenden 5 kare kesilir, kalan şerit $52 \times 39 \rightarrow$ 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 }
```

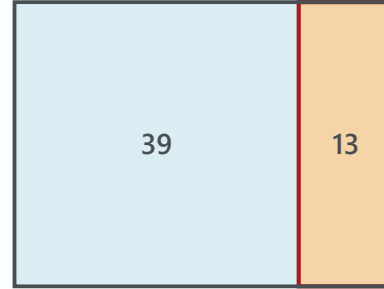
ÇAĞRI YIĞINI

ebob(52, 39)

ebob(299, 52)

ebob(650, 299)

ebob(1599, 650)



52 × 39 dikdörtgen

$52 = 1 \cdot 39 + 13$. 52×39 dikdörtgenden 1 kare kesilir, kalan şerit $39 \times 13 \rightarrow$ `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

$39 = 3 \cdot 13 + 0$. 39×13 dikdörtgenden 3 kare kesilir, kalan şerit $13 \times 0 \rightarrow$ 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 }
```

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

ÇAĞRI YIĞINI

ebob(13, 0)

ebob(39, 13)

ebob(52, 39)

ebob(299, 52)

ebob(650, 299)

ebob(1599, 650)

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

Adım 6 / 7

DEĞİŞKENLER

a = 13

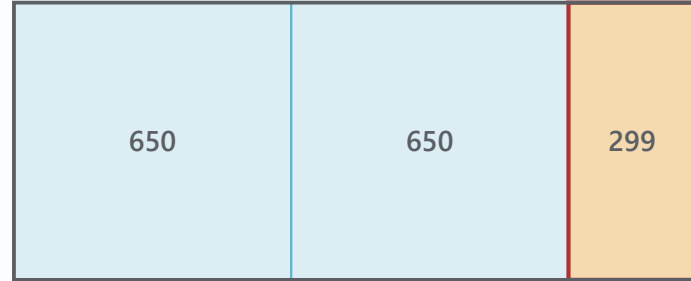
b = 0

ebob(1599, 650): Adım Adım

```
1 int ebob(int a, int b) {  
2   if (b == 0)  
3     return a;  
4   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

Euclid Algoritması: Neden Hızlı?

- Verimli ve hızlı çalışır: **her iki adımda** büyük sayı en az **yarıya** iner ($a \bmod b < a/2$).
- Bu yüzden adım sayısı en fazla $\approx 2 \log_2 \min(a, b)$: **$O(\log \min(a, b))$** .
- **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)

$$55 = 1 \cdot 34 + 21$$

$$34 = 1 \cdot 21 + 13$$

$$21 = 1 \cdot 13 + 8$$

$$13 = 1 \cdot 8 + 5$$

$$8 = 1 \cdot 5 + 3$$

$$5 = 1 \cdot 3 + 2$$

$$3 = 1 \cdot 2 + 1$$

$$2 = 2 \cdot 1 + 0$$

8 bölme, EBOB = 1. Karşılaştırın: ebob(1599, 650) yalnızca 5 bölme.

ÖZET

Özet

Konu	Hatırlanacaklar
Algoritma	girdi → açık, sonlu, etkin adımlar → çıktı; dilden bağımsızdır
Analiz	$T(n)$: girdi boyutuna göre temel işlem sayısı; zaman ve bellek
Durumlar	en kötü (garanti), ortalama (beklenen), en iyi (yanıltıcı)
Gösterimler	O üst, Ω alt, Θ sıkı sınır; sabitler ve düşük terimler atılır
Sınıflar	$1 < \log n < \sqrt{n} < n < n \log n < n^2 < n^3 < 2^n < n!$
Eklemeli sıralama	en iyi $\Theta(n)$, en kötü $\Theta(n^2)$, yerinde, $O(1)$ ek bellek
Böl ve fethet	birleştirme sıralaması $2T(n/2) + \Theta(n) = \Theta(n \log n)$; ikili arama ve üs alma $T(n/2) + \Theta(1) = \Theta(\log n)$
Euclid	$\text{ebob}(a, b) = \text{ebob}(b, a \bmod b)$; $O(\log \min(a, b))$; en kötü: ardışık Fibonacci

Etkileşimli Simülatörler

Algoritma Karmaşıklığı Analizi

$O(\log n)$, $O(n)$, $O(n \log n)$, $O(n^2)$ ve $O(n^3)$ örnek JavaScript fonksiyonlarından birini seçin ya da kendi fonksiyonunuzu yazın; beş farklı girdi boyutunda tarayıcıda çalıştırıp ölçülen süreleri grafikte karşılaştırın.

Bu bölümde: Büyüme Hesaplayıcı

"Deneyin: Büyüme" slaytında bir n değeri girin; 1'den $n!$ 'e kadar her fonksiyonun işlem sayısını ve saniyede 10^9 işlem yapan bir bilgisayardaki süresini görün.

ÖZET

Sık Yapılan Hatalar

O = en kötü, Ω = en iyi sanmak

O, Ω , Θ sınır türleridir; her durum (en kötü, en iyi) için ayrı ayrı yazılabilir.

Yanlış Θ çıkarımı

En kötü $O(n^2)$, en iyi $\Omega(n)$ ise tüm girdiler için tek bir Θ yoktur. Θ için iki sınır aynı olmalı.

Süreyi saniyeyle ölçmek

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

Ezberlemeyi unutmak

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

Ortalama = garanti sanmak

Hızlı sıralama ortalamada $O(n \log n)$, ama sıralı girdide ilk öge pivotla $O(n^2)$ 'dir.

Sıralanmamış diziye ikili arama

İkili arama yalnızca sıralı dizide doğrudur; aksi hâlde var olan öge "yok" bulunabilir.

ÖZET

Örnek Problemler

1. Eklemeli sıralama [5, 4, 3, 2, 1] ve [1, 2, 3, 4, 5] dizilerinde kaç karşılaştırma ve kaç kaydırma yapar?
2. $T(n) = 5n^2 + 100n \log_2 n + 7$ için Θ sınıfını yazın. $n = 1000$ 'de hangi terim baskındır?
3. [2, 4, 7, 10, 13, 18, 21, 25, 30] dizisinde ikili arama ile 21 ve 5'i arayın; incelenen orta öğeleri yazın.
4. $\text{ebob}(252, 105)$ 'i Euclid algoritmasıyla bulun.
5. 2^{13} yukarıdaki `us` fonksiyonuyla hesaplanırken hangi üsler için çağrı yapılır, kaç çarpma gerekir?

► Cevapları göster

ÖZET

Alıştırmalar

1. Bir dizide verilen bir sayının kaç kez geçtiğini bulan algoritmayı yazın; zaman ve bellek 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. Eklemeli sıralamayı azalan sırada sıralayacak şekilde değiştirin. Hangi satır değişir?
4. Euclid 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; Euclid).
- J. Kleinberg ve É. Tardos, *Algorithm Design*, Pearson, 2005 (Bölüm 2: Basics of Algorithm Analysis).

Son

Bölüm 1: Giriş

Sonraki bölüm: Bölüm 1.2: Algoritma Karmaşıklığı · [Terim sözlüğü](#)