Algoritma Analizi

Giriş

- Algoritma tasarlamadan *önce bazı teknikleri bilmek gerekir* ve **bu tekniklerden sonra ancak algoritma tasarlanır.**
- **Algoritma analizi**

1. Performans (başarım)
2. Kaynak kullanımı

üzerinde odaklanır.

Giriş

- Programcılıkta başarımdan daha önemli neler vardır?

- modülerlik
- doğruluk
- bakım kolaylığı
- işlevsellik
- sağlamlık
- kullanıcı dostluğu
- programcı zamanı
- basitlik
- genişletilebilirlik
- güvenilirlik

* Başarım (performans) olmazsa hızdan bahsedilemez ve hız olmayınca da diğerler durumlar gölgede kalacaktır.

Örnek: Sıralama Problemleri

- Algoritma analizi için bir dizinin elemanlarını **artan sıraya** göre <u>sıralayan problemi ele alalım</u>.

  Girdi: dizi $\langle a_1, a_2, \ldots, a_n \rangle$ sayıları.

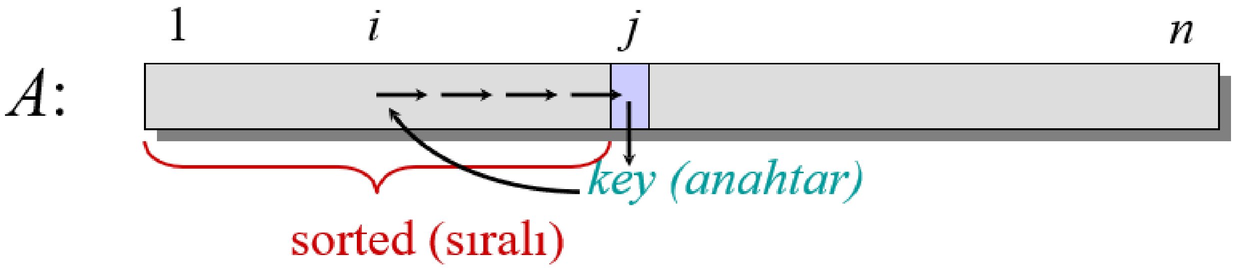
  Çıktı: permütasyon $\langle a'_1, a'_2, \ldots, a'_n \rangle$
  öyle ki $a'_1 \le a'_2 \le \cdots \le a'_n$.

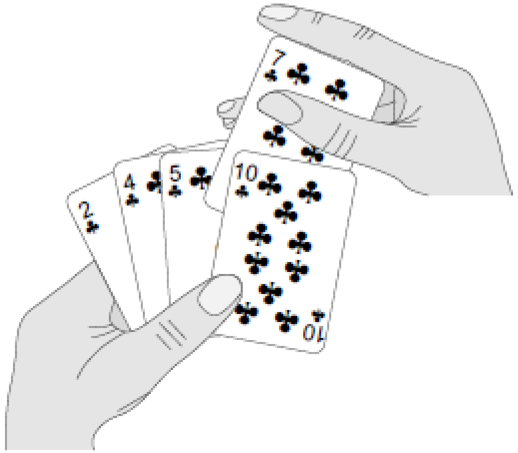
  Örnek:

  Girdi: 8 2 4 9 3 6

  Çıktı: 2 3 4 6 8 9

Araya yerleştirme sıralaması (Insertion sort)





8 2 4 9 3 6

8 2 4 9 3 6

2 8 4 9 3 6

Araya yerleştirme sıralaması (Insertion sort)

2 8 4 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

2 4 8 9 3 6

Araya yerleştirme sıralaması (Insertion sort)

2 4 8 9 3 6

2 3 4 8 9 6

2 3 4 8 9 6

2 3 4 6 8 9

Araya Yerleştirme Algoritmasının Sözde Kodu

- Sözde (Kaba) kod, bir algoritmayı daha anlaşılır hale getirmek ve mümkün olduğunca daha kısa bir şekilde sunmayı sağlar.

‘pseudocode”

(sözdekod)

INSERTION-SORT (A, n) $\triangleleft A[1 \,.\,.\, n]$

for $j \leftarrow 2$ **to** n

do $key \leftarrow A[j]$

$i \leftarrow j - 1$

while $i > 0$ and $A[i] > key$

do $A[i+1] \leftarrow A[i]$

$i \leftarrow i - 1$

$A[i+1] = key$ *(anahtar)*

Koşma süresi (Running time)

1. Girişe bağımlıdır:

Önceden sıralanmış bir diziyi sıralamak daha kolaydır.

*Örneğin **araya yerleştirme algoritması**, sıralı bir dizide daha az iş yapar ve etkilidir. Fakat tersten sıralı olan bir dizide çok iş yapar.*

2. Girişin boyutuna bağımlıdır.

Kısa dizileri sıralamak uzun dizilere oranla daha kolaydır.

*Genellikle, koşma süresinde üst sınırları ararız, çünkü herkes garantiden hoşlanır.

Çözümleme (Analiz) Türleri

- **En kötü durum (Worst-case):** (genellikle)
  - $T(n) =$ ***n*** boyutlu bir girişte algoritmanın maksimum süresi
- **Ortalama durum:** (bazen)
  - $T(n) = n$ boyutlu her girişte algoritmanın beklenen süresi.
  - Girişlerin istatistiksel dağılımı için varsayım gerekli.
- **En iyi durum:** (gerçek dışı)
  - Bir giriş yapısında hızlı çalışan yavaş bir algoritma ile hile yapmak.

Makineden-bağımsız zaman

- ***Araya yerleştirme sıralamasının en kötü zamanı nedir?***
  - Bilgisayarın hızına bağlıdır:
    - *bağıl (relative) zaman (aynı makinede),*
    - *mutlak (absolute) zaman (farklı makinelerde).*
- **BÜYÜK FİKİR (BIG IDEA):**
  - Makineye bağımlı sabitleri görmezden gel.
  - $n \longrightarrow \infty$ ' a yaklaştıkça, $T(n)$'nin ***büyümesi***ne bakılır.

" Asimptotik Çözümleme"

Θ (Theta) simgelemi (notation)

- ***Matematik:***

  - $\Theta(g(n)) = \{ f(n)$: Öyle c_1, c_2, n_0 pozitif sabit sayıları vardır ki tüm $n \geq n0\}$ için $0 \leq c_1\, g(n) \leq \boldsymbol{f(n)} \leq c_2\, g(n)$.

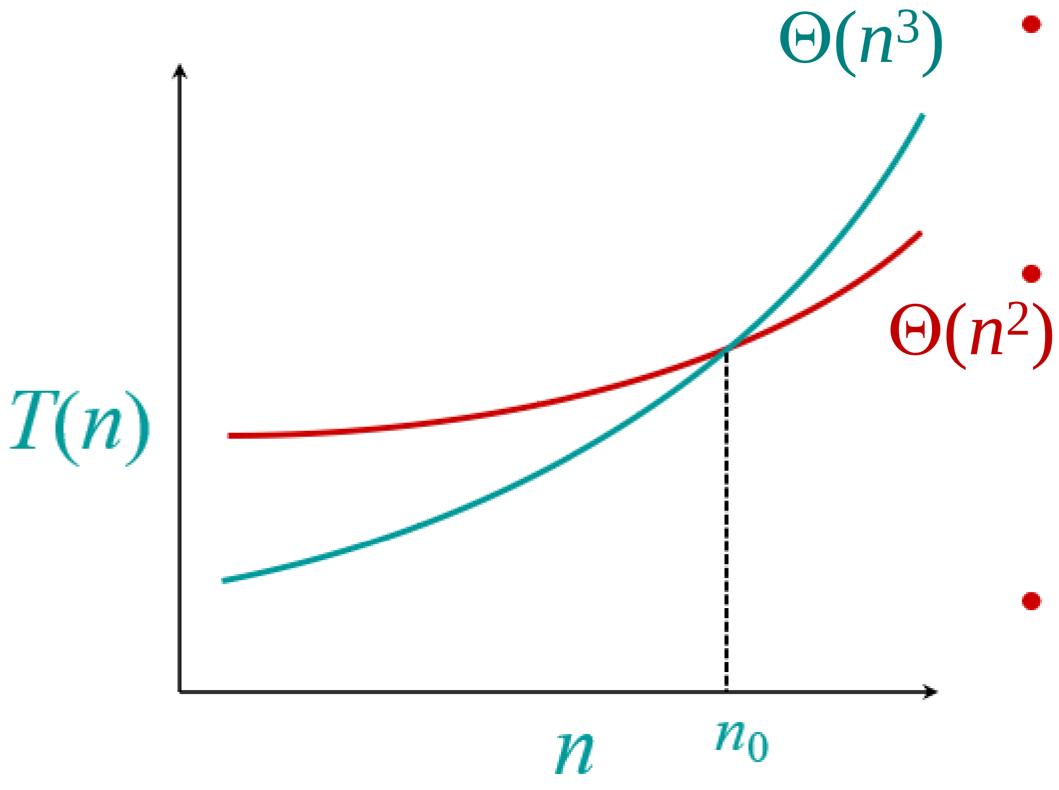
- ***Mühendislik:***

  - Düşük değerli terimleri at; ön sabitleri ihmal et.
  - Örnek: $3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3)$

* Bu notasyon **ortalama çalışma zamanını** matematiksel olarak gösteriminde kullanılmaktadır.

Asimptotik başarım

n yeterince büyürse, $\Theta(n^2)$ algoritması bir $\Theta(n^3)$ algoritmasından her zaman daha hızlıdır.



- Öte yandan asimptotik açıdan yavaş algoritmaları ihmal etmemeliyiz.
- Gerçek dünyada tasarımın mühendislik hedefleriyle dikkatle dengelenmesi gereklidir.
- Asimptotik çözümleme düşüncemizi yapılandırmada önemli bir araçtır.

Örnek: Araya yerleştirme sıralaması çözümlemesi

- **Her bir satır için maliyet ve adet sütunlarını yazarız.**

Insertion-Sort(A)	*cost*	*times*
for $j = 2$ **to** $A.length$	c_1	n
$key = A[j]$	c_2	$n-1$
// Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
$i = j - 1$	c_4	$n-1$
while $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
$A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
$i = i - 1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
$A[i+1] = key$	c_8	$n-1$

Örnek: Araya yerleştirme sıralaması çözümlemesi

- **Çalışma zamanı :** ***T(n)***’yi hesaplamak için maliyetlerin ve adet sütunlarının çarpımına eşittir.

$$
\begin{aligned}
T(n) \;&=\; c_1 n + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) \\
&\quad + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)\,.
\end{aligned}
$$

Örnek: Araya yerleştirme sıralaması çözümlemesi

- ***En iyi durumda (Best-case) çalışma zamanı:*** ***Dizi sıralı bir şekilde geldiğini varsayarız.***

5. Satırda ***A[i]≤ key*** *olur*. Böylece ***j=2,3,…n için*** $\boldsymbol{t_j=1}$ olur.

$$
\begin{aligned}
T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8)\,.
\end{aligned}
$$

- Bu çalışma zamanını *a* ve *b* sabitler ($\boldsymbol{c_i}$ maliyetlerine bağlı) olmak üzere ***an+b*** olarak ifade edebiliriz.
- ***Bu ise n ye göre doğrusal (linear) bir fonksiyondur.***

Örnek: Araya yerleştirme sıralaması çözümlemesi

- ***En kötü durumda (Worst case) çalışma zamanı:*** ***Dizi ters bir şekilde sıralı olduğunu varsayırız.***

Bu durumda her bir A[j] elemanını sıralı A[1,…,j-1] dizisinin her elemanı ile karşılaştırmalıyız. Böylece j=2,3,…,n için $t_j=j$ olur.

- Araya yerleştirme sıralama algoritmasının toplam zamanın hesaplanması bir **aritmetik seri hesaplaması** olarak görülebilir.

Hatırlatma:

$$\sum_{k=1}^{n} k = 1 + 2 + \cdots + n$$

$$\sum_{k=1}^{n} k = \frac{1}{2}n(n+1) = \Theta(n^2) .$$

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1$$

5. Satır (While döngüsü)

and

$$\sum_{j=2}^{n} (j-1) = \frac{n(n-1)}{2}$$

6. ve 7. Satır (While döngüsü içi)

Örnek: Araya yerleştirme sıralaması çözümlemesi

❑En Kötü Çalışma Zamanı :

$$\begin{aligned} T(n) &= c_1 n + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8)\ . \end{aligned}$$

- En kötü çalışma zamanı $\boldsymbol{c_i}$ maliyetine bağlı olan **a, b** ve **c** sabitleri için $\boldsymbol{an^2 + bn + c}$ *olarak ifade edebiliriz.*
- *Bu durumda **çalışma zamanı n′**ye bağlı* **kuadratik fonksiyon (ikinci dereceden)** olur.

$$T(n) = \sum_{j=2}^{n} \Theta(j) = \Theta(n^2)$$

Örnek: Araya yerleştirme sıralaması çözümlemesi

- ***Ortalama durumda (Avarage-case) çalışma zamanı:***

Varsayalım ki rastgele **n** *tane sayıyı seçilir.*

Ortalama olarak dizi elemanlarının yarısı A[j]'den küçük yarısıda A[j]'den büyüktür.

Bu yüzden ortalama olarak ***A[1,…,j-1]*** *alt dizisinin* ***yarısını kontrol*** **ederiz. Böylece yaklaşık olarak t_j=j/2 olur.**

- Ortalama durum, en kötü durumdaki gibi girdi boyutunun kuadratik bir fonksiyonudur.

$$T(n) = \sum_{j=2}^{n} \Theta(j/2) = \Theta(n^2)$$

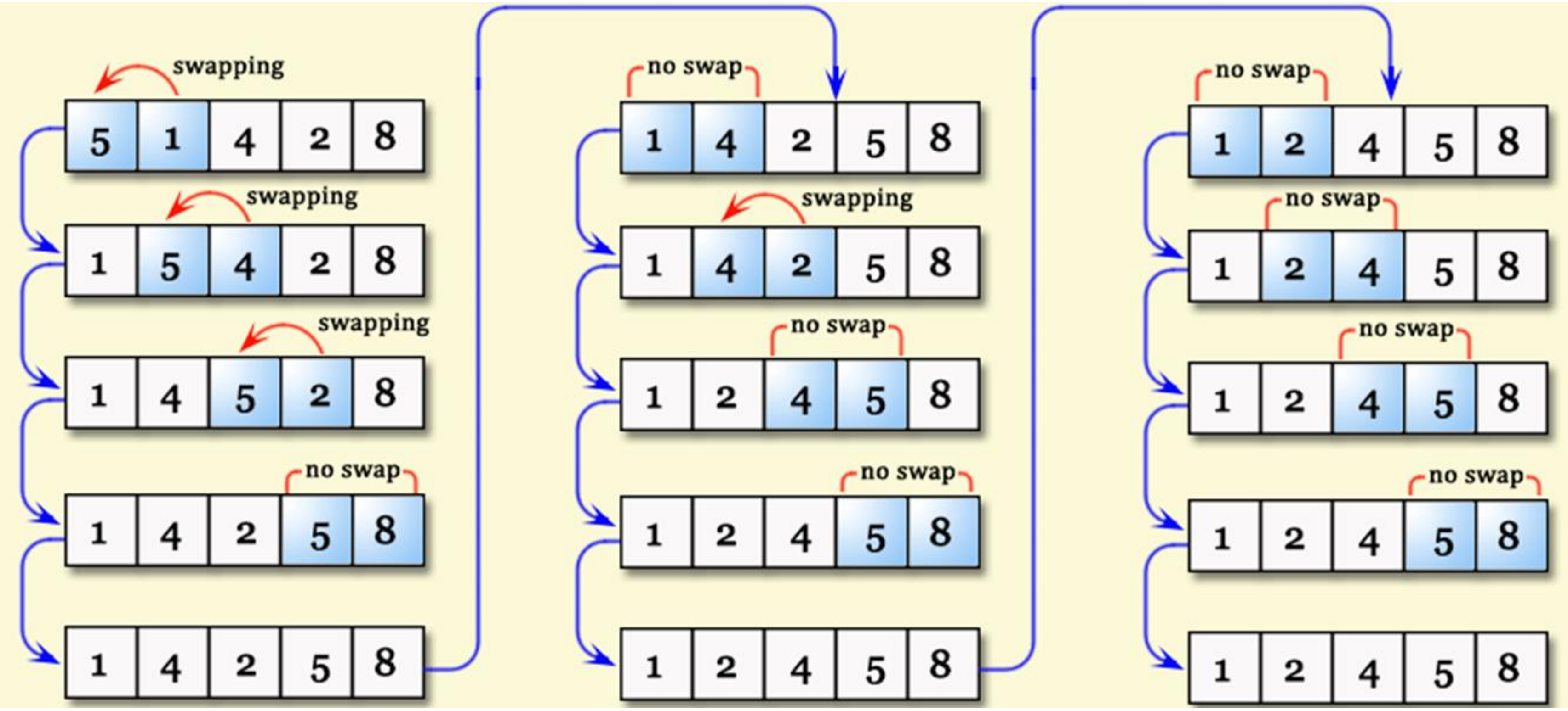
Örnek: Araya yerleştirme sıralaması çözümlemesi

- **Ortalama durum analizinin kapsamı sınırlıdır.** Çünkü ***özel bir problem için 'ortalama'*** girdinin **ne olacağı anlaşılmayabilir.**
- Genellikle bir verilen boyutta tüm girdilerin **eşit olasılıkta olduğu kabul edilir.**
- Fakat pratikte beklenen çalışma zamanını sağlamak için **rassal algoritmalar kullanılır.**

Örnek: Araya yerleştirme sıralaması çözümlemesi

- **En kötü durum analizi** ile hesaplanan çalışma zamanı **bize çalışma zamanı hakkında <u>üst sınırı verir.</u>**
- Bazı algoritmalar için genellikle en kötü durum meydana gelir.
  - Örneğin **bir veritabanında bir veri parçası aratıldığında <u>verinin bulunmama durumu ile karşılaşılabilir.</u>**

Örnek: Kabarcık (Buble) Sıralama Algoritması



Örnek: Kabarcık (Buble) Sıralama Algoritması

BUBBLESORT(A)	Cost	Times
for $i = 1$ **to** $A.length - 1$	c1	n
for $j = A.length$ **downto** $i + 1$	c2	$\sum_{j=1}^{n} t_j$
if $A[j] < A[j-1]$	c3	$\sum_{j=1}^{n} t_j$
exchange $A[j]$ with $A[j-1]$	c4	$\sum_{j=1}^{n} t_j$

```
BUBLESORT(B)
Data: Input array A[]
Result: Sorted A[]
int i, j, k;
N = length(A);
for j = 1 to N do
    for i = 0 to N-1 do
        if A[i] > A[i+1] then
            temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
        end
    end
end
```

T(n)=c1*n+$c_2 \sum_{j=1}^{n} t_j + c_3 \sum_{j=1}^{n} t_j + c_4 \sum_{j=1}^{n} t_j$

T(n) =c1*n+c2*n(n+1)/2+ c3*n(n+1)/2+ c4*n(n+1)/2

T(n)= $\theta(n^2)$

Örnek: Kabarcık (Buble) Sıralama Algoritması

- Bir dizi verildiğinde N, ilk yineleme (N - 1)karşılaştırmalar yapar .
- İkinci yineleme (N - 2)karşılaştırmalar gerçekleştirir.
- Bu şekilde, toplam karşılaştırma sayısı şöyle olacaktır:

$$(N-1)+(N-2)+(N-3)+\ldots\ldots+3+2+1=\frac{N(N-1)}{2}=\mathcal{O}(N^2)$$

Örnek: Kabarcık (Buble) Sıralama Algoritması

- Bu nedenle, içinde **ortalama durumda** , zaman karmaşıklığı tür olurdu **T(n)=** $\theta(n^2)$ olur.
- Eğer dizi sıralanmış bir şekilde verilmiş olursa **en iyi durumda** yine benzer kıyaslama yapmak durumundayız ve **T(n)=** $\theta(n^2)$ olur.
- Eğer dizi ters sıralanmış bir şekilde verilmiş olursa **T(n)=** $\theta(n^2)$ olur.

Örnek: Kabarcık (Buble) Sıralama Algoritması İyileştirme

```
Improved Bubble Sort
Data: Input array A[]
Result: Sorted A[]
int i, j, k;
indicator = 1;
N = length(A);
for j = 1; j ≤ (N-1) ∧ indicator == 1; j++ do
    indicator = 0;
    for i = 1 to N-1; i++ do
        if A[i] > A[i+1] then
            temp = A[i];
            A[i] = A[i+1];
            A[i+1] = temp;
            indicator = 1;
        end
    end
end
```

- *Indicator* adında bir değişken ile *değiştirilen öğelerin kaydını tutmak* için yeni bir değişken ekledik.
- Ayrıca bu yineleme sırasında bu dizinin *sıralanmış olup olmadığını gösterebilir.*
- Bu durumda **ortalama** ve **en kötü durumda** yine **T(n)=** $\boldsymbol{\theta(n^2)}$ olacaktır.
- **Eğer *dizi sıralanmış*** ise yani **en iyi durumda T(n)= θ(n)** olacaktır.

Örnek: Birleştirme (Merge) Sıralama Algoritması

- Birleştirme sıralama algoritması, **böl ve yönet (fethet)** yaklaşımını takip etmektedir.
- **Böl:** Aynı problemi *daha küçük alt problemlere ayır.*
- **Yönet (Fethet):** *Alt problemler özyineleme ile çözülür.* Eğer alt problem boyutları yeteri kadar küçükse daha kolay bir şekilde çözülürler.
- **Birleştir: Alt problemlerin çözümleri orijinal problemin çözümü için birleştirilir.**

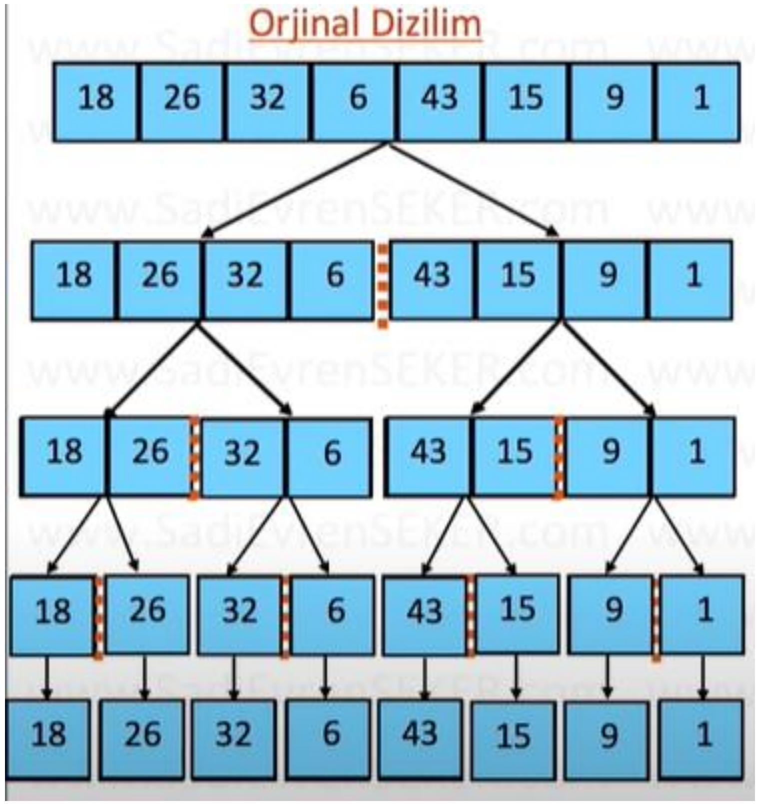
Örnek: Birleştirme (Merge) Sıralama Algoritması

1. Eğer $n = 1$ ise, işlem bitti.
2. $A[\,1\,.\,.\lceil n/2\rceil\,]$ ve $A[\,\lceil n/2\rceil+1\,.\,.\,n\,]$'yi özyinelemeli sırala.
3. 2 sıralanmış listeyi "***Birleştir***".

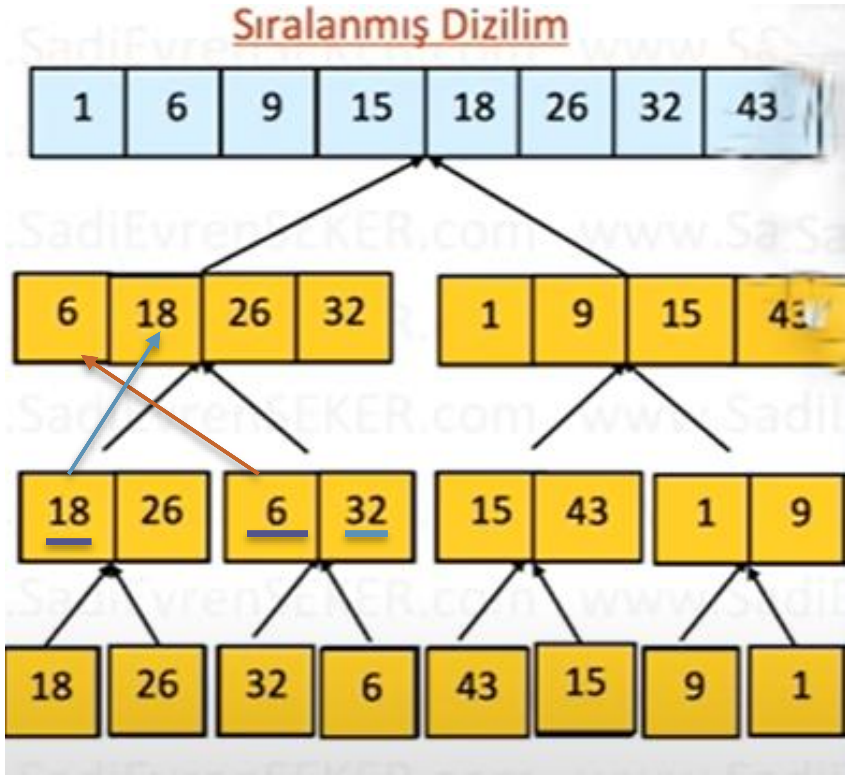
Anahtar altyordam: **Birleştirme**

Örnek: Birleştirme (Merge) Sıralama Algoritması

Böl (Divide)



Fethet (Conquer) ve Birleştir



Örnek: Birleştirme (Merge) Sıralama Algoritması

	$T(n)$	**BİRLEŞTİRME-SIRALAMASI** $A[1 \,..\, n]$
	$\Theta(1)$	1. Eğer $n = 1$'se, bitir.
Suistimal	$2T(n/2)$	2. Yinelemeli olarak $A[\,1 \,..\, \lceil n/2 \rceil\,]$ ve $A[\,\lceil n/2 \rceil{+}1 \,..\, n\,]$'yi sırala.
	$\Theta(n)$	3. ***İki*** sıralı listeyi ***"Birleştir"***

Özensizlik: $T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor)$ olması gerekir, ama asimptotik açıdan bu önemli değildir. Yani sonuca etki etmeyecektir.

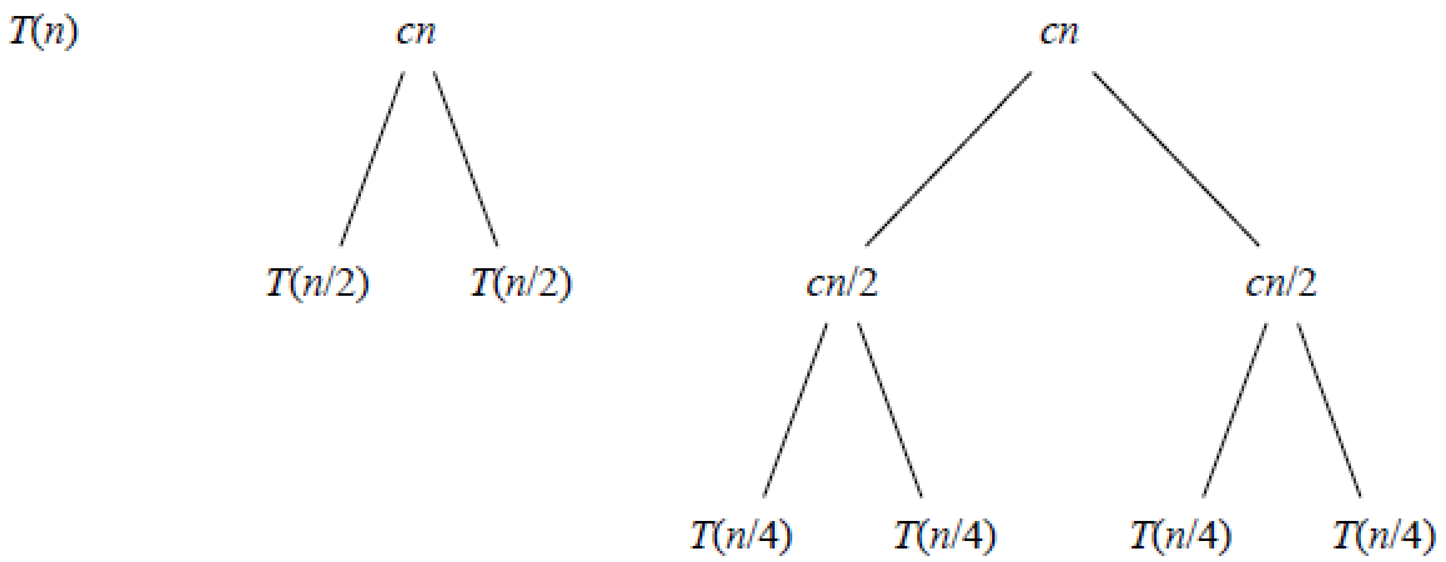
Örnek: Birleştirme (Merge) Sıralama Algoritması

$$T(n) = \begin{cases} \Theta(1) \text{ eğer } n = 1 \text{ ise;} \\ 2T(n/2) + \Theta(n) \text{ eğer } n > 1 \text{ ise.} \end{cases}$$

- Genellikle n'nin küçük değerleri için taban durumu (base case) olan $T(n) = \Theta(1)$ 'i hesaplara katmayacağız; ama bunu sadece yinelemenin asimptotik çözümünü etkilemiyorsa yapacağız.

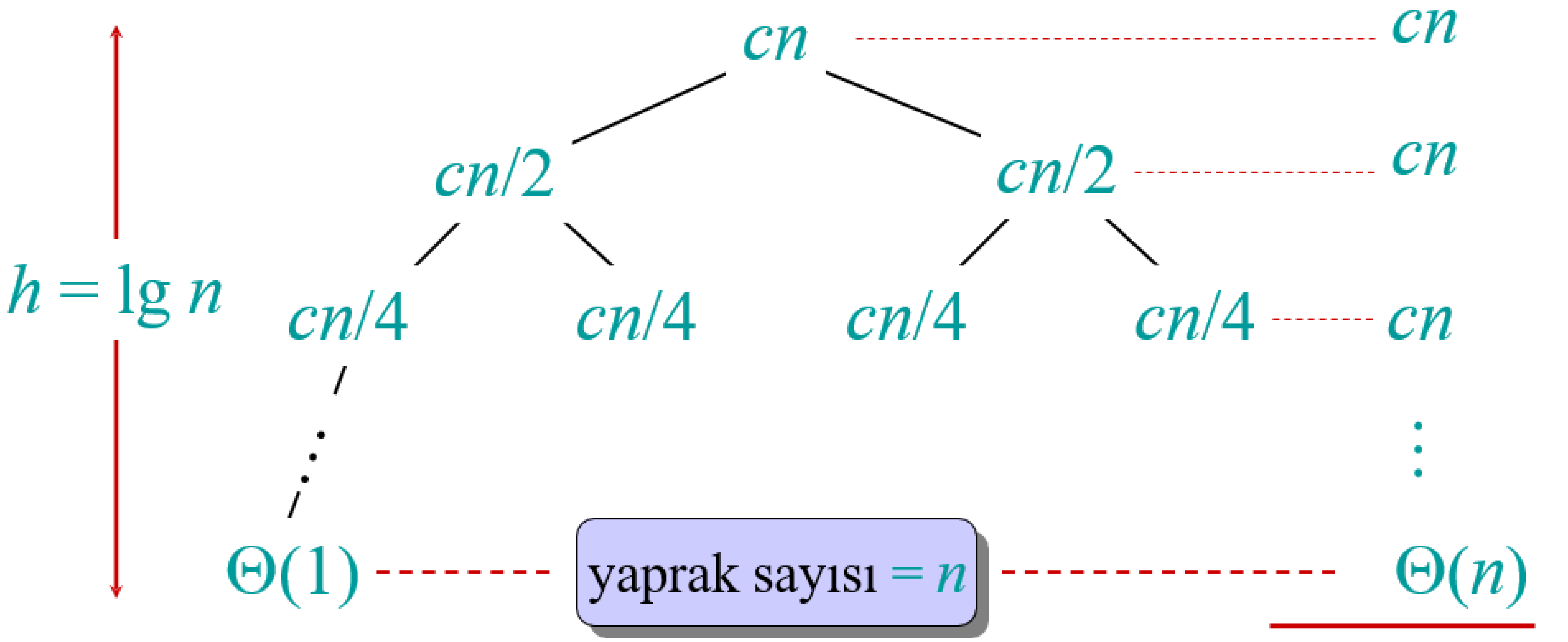
Yineleme ağacı

- İşlemi daha basit ifade etmek için Θ(n) yerine c*n diyebiliriz.



Yineleme ağacı

$T(n) = 2T(n/2) + cn$'i çözün; burada $c > 0$ bir sabittir.



Toplam $= \Theta(n \lg n)$

Sonuç

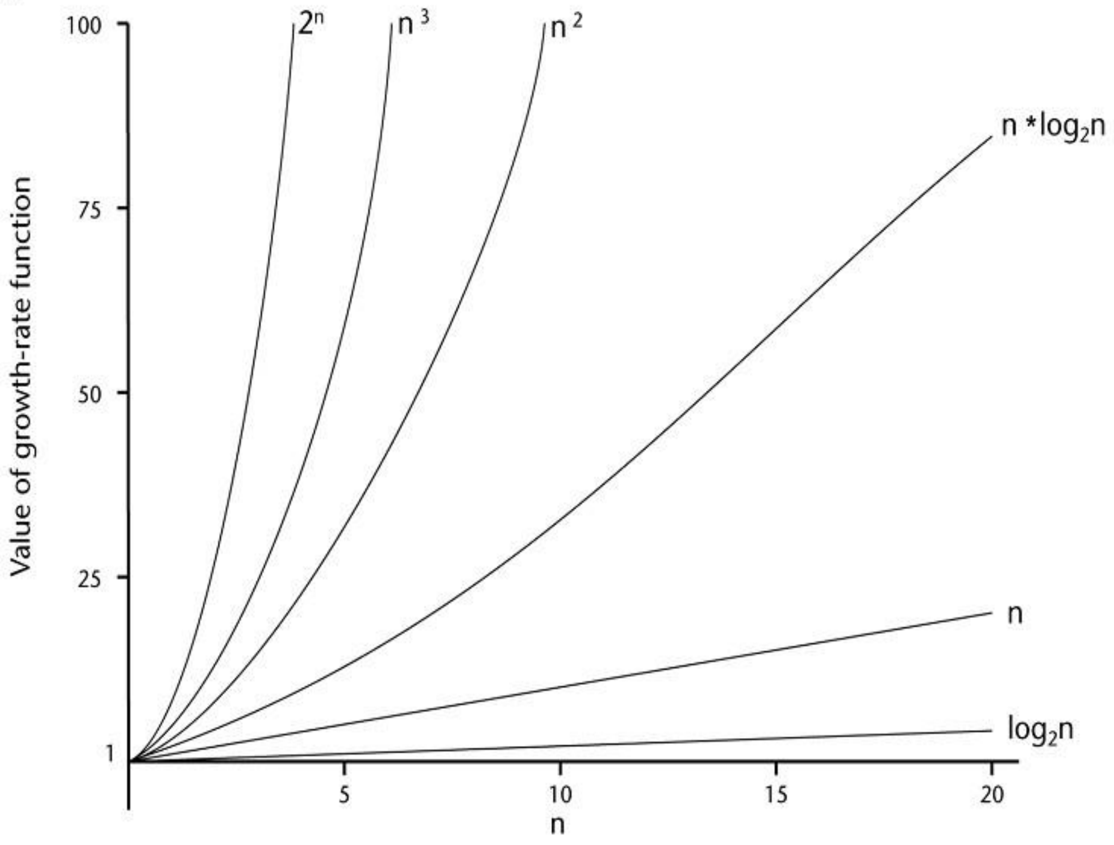
- $\Theta(n \lg n)$, $\Theta(n^2)$'dan daha yavaş büyür.

- En kötü durumda, birleştirme sıralaması asimptotik olarak araya yerleştirme sıralamasından daha iyidir.
- Pratikte, birleştirme sıralaması araya yerleştirme sıralamasını $n > 30$ değerlerinde geçer.

Algoritmanın Büyüme Oranları

- **Büyüme oranlarına bakarak iki algoritmanın verimliliğini karşılaştırabiliriz.**
- *Algoritma tasarımcılarının amacı*, çalışma zaman fonksiyonu olan f(n) nin mümkün olduğu kadar <u>düşük büyüme oranı sahip bir algoritma olmasıdır.</u>

Büyüme oranı fonksiyonlarının karşılaştırılması



- **Örnek** : Verilen zaman karmaşıklığı ile algoritmalar için yürütme zamanı

n	f(n)=n	f(n)=nlogn	f(n)=n^2	f(n)=2^n
20	0.02 µs	0.086 µs	0.4 µs	1 ms
10^6	1 µs	19.93 ms	16.7 dk	31.7 yıl
10^9	1s	29.9 s	31.7 yıl	!!! yüzyıllar

Algoritmanın Büyüme Oranları

1	Algoritmadaki emirlerin **icra sayısı**, **kesin olarak tespit ediliyorsa**; bu algoritmanın koşma zamanı sabit değerler ile gösterilir.
log*n*	**n’nin büyüyen değerine** karşın algoritma **çok az değer ile yavaşlıyorsa** algoritmanın logaritmik bir karakteristiği olduğu söylenebilir. Bu karakteristik problemi alt problemlere bölerek çözen algoritmalarda görülür.
n	Giriş değerlerine bağlı olarak çalışma zamanını **lineer olarak değiştiği** algoritmalar bu sınıfa girer. Bu durum algoritmada **beklenen en iyi karakteristiktir.**
nlogn	Bu tarz büyüme oranına sahip algoritma **problemi, alt problemlere bölüp**, oluşan alt problemleri **bağımsız olarak çözen ve son aşamada bu sonuçları birleştiren** algoritmalarda görülür.
n^2	**İç içe döngüler yardımı** ile verileri **ikişerli olarak ele alıp** , tüm verileri inceleyen algoritmalarda görülür. İşlenecek veri uzunluğu fazla olmadığı zamanlarda kullanılır.
n^3	**İç içe döngüler yardımı** ile verileri **üçerli olarak ele alıp** , tüm verileri inceleyen algoritmalarda görülür. (örn: matris çarpımı) İşlenecek veri uzunluğu fazla olmadığı zamanlarda kullanılır.
2^n	Bu tarz algoritmada **mümkün tüm durumların denenmesi** söz konusu olmaktadır.

Asimptotik simgelem

- Bir algoritmanın orantılı zaman gereksinimi **büyüme oranı (veya büyüme hızı)**olarak bilinir.
- **T(n)** nin büyüme oranı, algoritmanın **hesaplama karmaşıklığı**dır.
- Karmaşıklığı belirtmek için **asimtotik notasyon(simgelem) ifadeleri kullanılmaktadır.**
- Genel olarak, az sayıda parametreler için karmaşıklıkla ilgilenilmez; eleman sayısı **n**'nin sonsuza gitmesi durumunda T(n) büyümesine bakılır.

Asimptotik Notasyon (Simgelem)

1. O-simgelemi (üst sınırlar):

Tüm n ≥ n_0 değerleri için sabitler c> 0, n_0 > 0 ise
0 ≤ f(n) ≤ cg(n) durumunda
f(n)=O(g(n)) yazabiliriz.

ÖRNEK: $2n^2 = O(n^3)$ $(c = 1,\ n_0 = 2)$

fonksiyonlar, değerler değil

komik, "tek yönlü" eşitlik

Notasyonlarda eşitlik"=" gösterimi

- **A=B ise B = A anlamında olabilir.**
- Fakat, $f(n) = O(g(n))$, $O(g(n)) = f(n)$ anlamına gelmez.

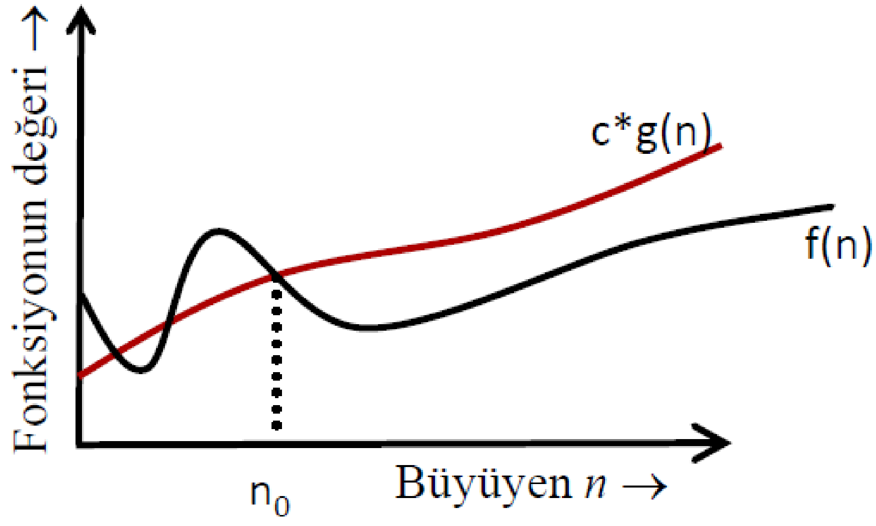
Burada tek eşitlik söz konusudur.

- **Burada "=", üyelik işlemi (ϵ) olarak tercih edilmiştir.**
- $f(n) = O(g(n)) \rightarrow f(n) \in O(g(n))$ dir
- $O(g(n))$ bir küme anlamına gelir.
- $f(n) = O(g(n)) \rightarrow O(g(n)) =\{ f(n) \}$ gösterimi doğrudur.

Asimptotik simgelem

1. O-simgelemi (üst sınırlar):

- *Bu notasyon **en kötü durumdaki zamanın** belirlenmesinde kullanılabilir.*



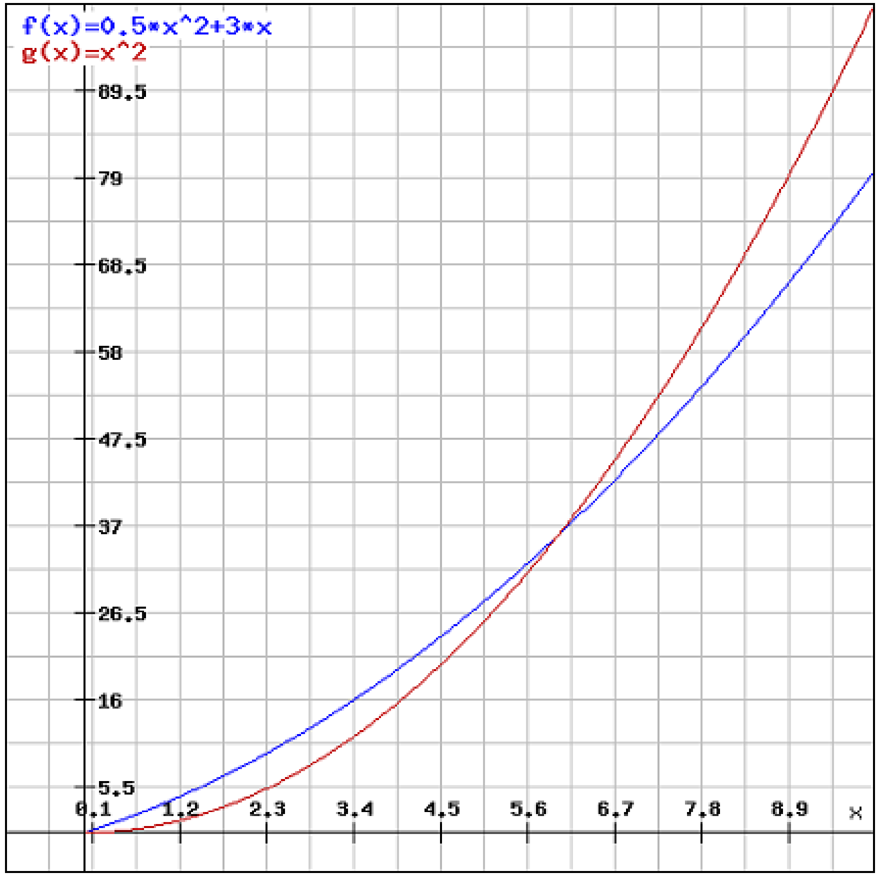
Notasyon	İsim
O(1)	Sabit
O(logn)	Logaritmik
O([logn]^c)	Polilogaritmik
O(n^2)	Karesel
O(n^c), c>1	Polinomial veya cebirsel
O(c^n), c>1	Üssel veya geometrik
O(n!)	Faktöriyel veya kombinatöryel
O(n^n)	Geokombinatör

Asimptotik simgelem

1. O-simgelemi (üst sınırlar):

- **Örnek:** *$(1/2)n^2 + 3n$ için üst sınırın $O(n^2)$ olduğunu gösteriniz.*
  - *$c=1$ için*
  - *$(1/2)n^2 + 3n \le n^2$*
  - *$3n \le 1/2n^2$*
  - *$6 \le n$,*
  - *$n_0=6$*

Çözüm kümesini sağlayan kaç tane $\mathbf{n_0}$ ve **c** çifti olduğu önemli değildir. Tek bir çift olması notasyonun doğruluğu için yeterlidir.



Asimptotik simgelem

2. Ω(Omega) simgelemi (alt sınırlar):

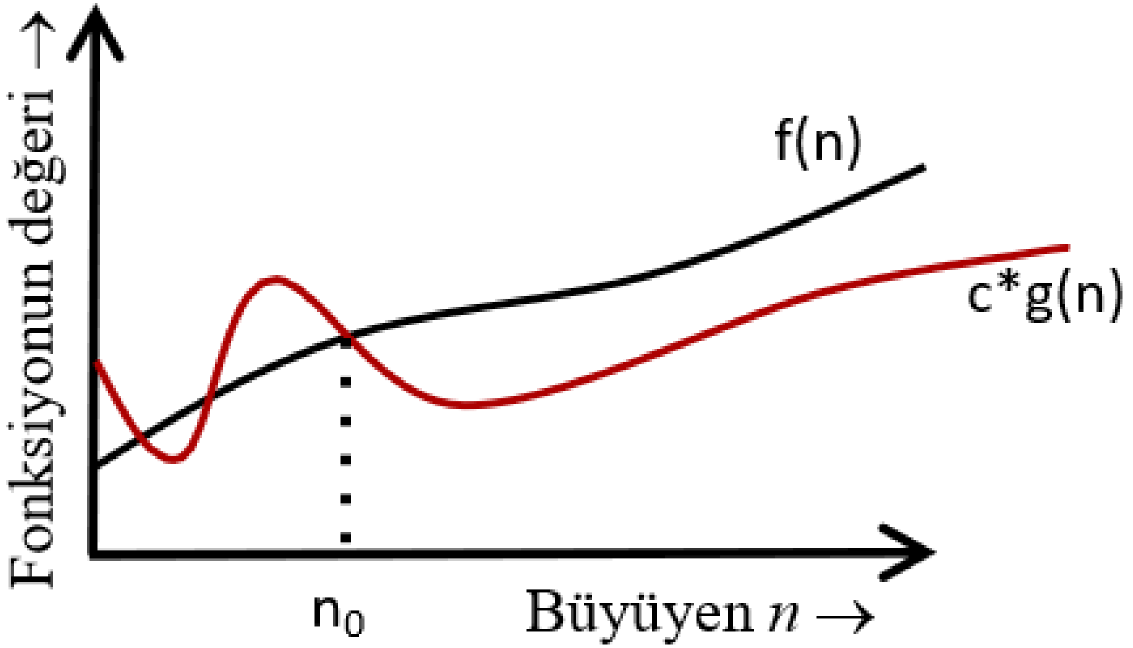
- Bu notasyon **algoritmaların <u>en iyi durumdaki</u> çalışma zamanını** <u>ifade etmek için kullanılır.</u>
- **Giriş değerleri analiz edildikten sonra** algoritmanın *O notasyonu* ***Ω*** ile ifade edilir.

$$\Omega(g(n)) = \{ f(n) : \text{tüm } n \geq n\_0 \text{ değerlerinde } c > 0,\ n\_0 > 0 \text{ ise, } 0 \leq cg(n) \leq f(n) \}$$

Asimptotik simgelem

2. Ω(Omega) simgelemi (alt sınırlar):

- Her durumda **$f(n) \geq c\, g(n)$** ve **$n \geq n_0$** koşullarını sağlayan pozitif, sabit **c** ve **n_0** değerleri bulunabiliyorsa **$f(n)=\Omega(g(n))$** dir.



Asimptotik simgelem

2. Ω(Omega) simgelemi (alt sınırlar)- Örnek

- $2n + 5 \in \Omega(n)$ olduğunu gösteriniz
  - $n_0 \geq 0$, $2n+5 \geq n$, olduğundan sonuç elde etmek için $c=1$ ve $n_0 = 0$ *(eşitliği sağlayan değer)* alabiliriz.

- $5*n^2 - 3*n = \Omega(n^2)$ olduğunu gösteriniz.
  - $5*n^2 - 3*n \geq n^2$, $c=1$, $n_0 =0$ değerleri için sağlar.

Asimptotik simgelem

3. θ(Theta) simgelemi :

- **Ortalama çalışma zamanı algoritma karakteristiğine** ve çözülen probleme göre değişecektir.
- **f(n) = θ(g(n))** olması için tüm **n≥k** değerinde

$0 \le c_1 g(n) \le f(n) \le c_2 g(n)$ olacak şekilde

pozitif c_1, c_1 ve k sabitleri tanımlı olması gerekir.

*Bu sabitler, **fonksiyona bağlı** fakat *g*'den **bağımsızdır.**

$$\Theta(g(n)) = O(g(n)) \cap \Omega(g(n))$$

Asimptotik simgelem

4. o simgelemi (sıkı olmayan üst sınırlar):

- **Örneğin**
  2 $* \, n^2$**=O(**n^2) asimtotik olarak sıkı bir üst sınırdır.
- Fakat **2*n = O(**n^2) sıkı bir üst sınır değildir. Yani sonsuz değer alır.
- **O ile o notasyonlarının tanımları birbirine benzerdir.**

Temel fark:

- **Her durumda** $c_1.g(n) \le f(n) \le c_2.g(n)$ ve $\mathbf{n \ge n_0}$ koşullarını sağlayan pozitif, sabit $\mathbf{c_1, c_2}$ **ve** $\mathbf{n_0}$ değerleri bulunabiliyorsa $f(n) = o(g(n))$ ifadesi doğrudur.
- **Her durumda olmuyorsa O** notasyonu geçerli olmaktadır.

o notasyonu matematiksel tanımı

$$\lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Asimptotik simgelem

4. o simgelemi (sıkı olmayan üst sınırlar)- Örnek

- $f(n) = 2n + 5 \in o(n)$.
  - $2n \leq 2n+5 \leq 3n$, tüm $n \geq 5$ için

- $f(n) = 5*n^2 - 3*n \in o(n^2)$.
  - $4*n^2 \leq 5*n^2 - 3*n \leq 5*n^2$, tüm $n \geq 4$ için

Asimptotik simgelem

4. ω simgelemi (sıkı olmayan alt sınırlar):

- ***Ω**(Omega) ve **ω*** simgelemesi arasındaki ilişkide **O** ve **o** arasındaki benzerliğe sahiptir.
- ω sıkı olmayan alt sınırı ifade eder.
- Örneğin $n^2/2 = \omega(n)$, fakat $n^2/2 \neq \omega(n^2)$ bu durum aşağıdaki gibi bir ifade doğurur.

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$$

Yukarıdaki ifadeye göre **n sonsuza giderken**
<u>$f(n)'$nin $g(n)'$e göre **çok büyük olduğunu** gösterir.</u>

o-notasyonu ve ω-notasyonu

- **O**-notasyonu ve **Ω**-notasyonu **≤** ve **≥** gibidir.
- **o**-notasyonu ve **ω**-notasyonu **<** ve **>** gibidir.
- **o**-notasyonunda **üst sınıra**, **ω** notasyonununda ise **alt sınıra eşitlik yoktur.** Bundan dolayı üst ve alt sınırları sıkı bir asimptotik notasyon değildir.
- Öncekinden tek farklılığı, **c katsayısı ve bir n_0 değeri var demek yerine**, *her c katsayısı için* ***başka bir n_0*** *olacağını kabul etmek.*

Asimptotik Notasyonların Karşılaştırılması

Geçişlilik (Transitivity):

- **$f(n) = \Theta(g(n))$** ve $g(n)$ **$= \Theta(h(n))$** ise **$f(n)$** $=$ **$\Theta(h(n))$,**
- $f(n) = O(g(n))$ ve $g(n) = O(h(n))$ ise $f(n) = O(h(n))$,
- $f(n) = \Omega(g(n))$ ve $g(n) = \Omega(h(n))$ ise $f(n) = \Omega(h(n))$,
- $f(n) = o(g(n))$ ve $g(n) = o(h(n))$ ise $f(n) = o(h(n))$,
- $f(n) = \omega(g(n))$ ve $g(n) = \omega(h(n))$ ise $f(n) = \omega(h(n))$.
- Dönüşlülük veya yansıma(Reflexivity):
- $f(n)= \Theta(f(n))$,
- $f(n)= O(f(n))$,
- $f(n)= \Omega(f(n))$.

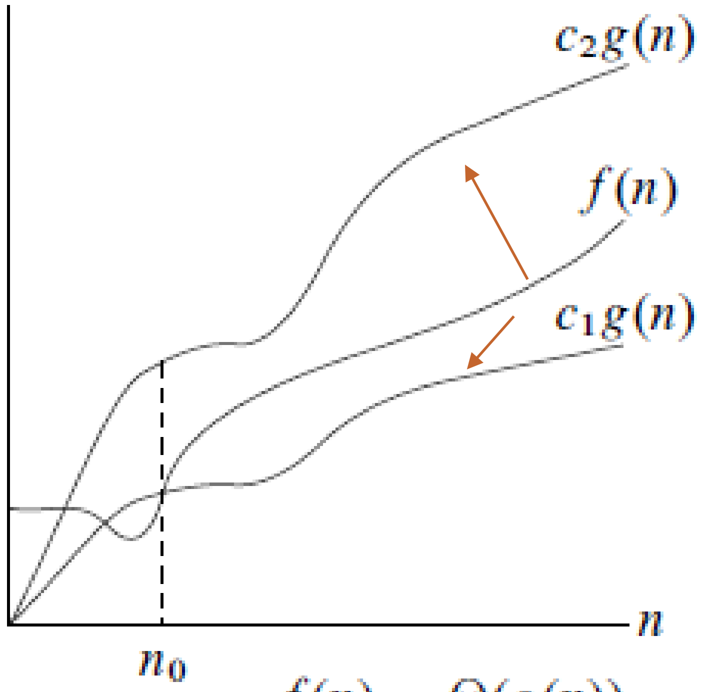
Asimptotik Notasyonların Karşılaştırılması

- Simetri(Symmetry):
  - **$g(n) = \Theta(f(n))$** olduğu durumda, **$f(n) = \Theta(g(n))$**
- Transpose (Ters Simetri):
  - **$g(n) = \Omega(f(n))$** olduğu durumda, **$f(n) = O(g(n))$,**
  - $g(n) = \omega(f(n))$ olduğu durumda, $f(n) = o(g(n))$.

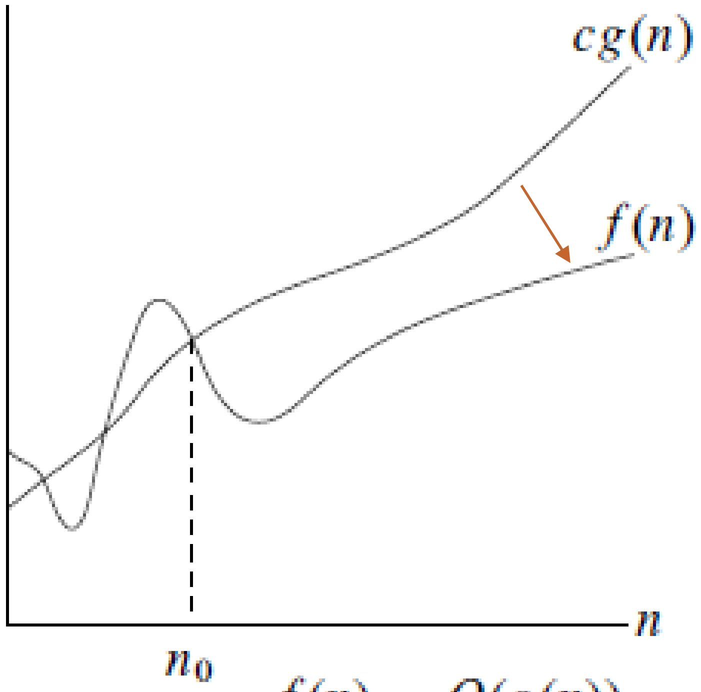
Not: Eğer f*(n)=o(g(n))* ise *f(n)*'in *g(n)*'den <u>*asimptotik küçük*</u>

eğer *f(n)= ω(g(n))* ise *f(n)*'in *g(n)*'den <u>*asimptotik büyük*</u> olduğunu söyleyebiliriz.

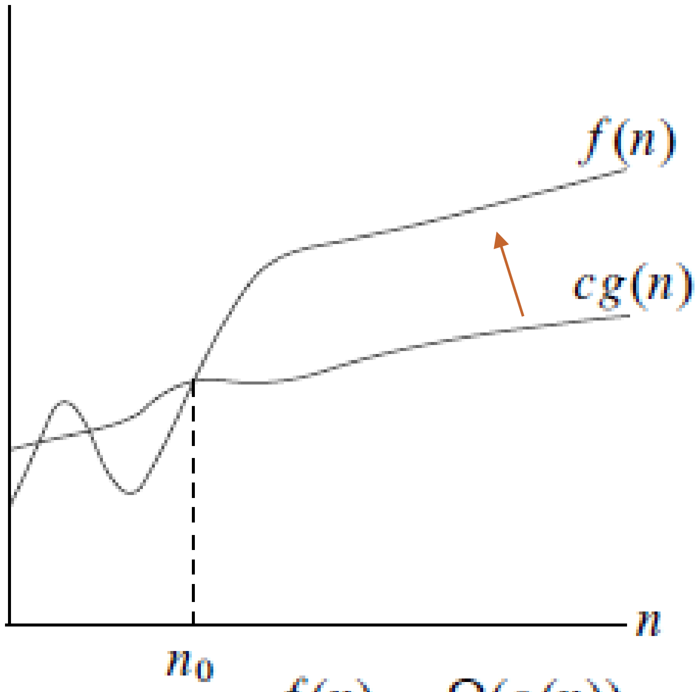
Asimptotik Notasyonların Karşılaştırılması



$f(n) = \Theta(g(n))$



$f(n) = O(g(n))$



$f(n) = \Omega(g(n))$

Kaynakça

- Algoritmalar : Prof. Dr. Vasif NABİYEV, Seçkin Yayıncılık
- Algoritmalara Giriş : Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, Palme YAYINCILIK
- Algoritmalar : Robert Sedgewick , Kevin Wayne, Nobel Akademik Yayıncılık
- M.Ali Akcayol, Gazi Üniversitesi, Algoritma Analizi Ders Notları
- Doç. Dr. Erkan TANYILDIZI, Fırat Üniversitesi, Algoritma Analizi Ders Notları