BİL3014

Algoritma Analizi

Dr. Öğr. Üyesi Emre DELİBAŞ

Yürütme Zamanı ve Asimptotik Analiz

Algoritma Analizinde Yaklaşımlar

Teorik Analiz

- Zaman verimliliği, girdi boyutunun bir fonksiyonu olarak temel işlemin tekrar sayısı belirlenerek analiz edilir.

Kaba Analiz

- Bir giriş verisi seçilir
- Zaman birimi veya yürütülen temel işlem adımlarının sayısı seçilir.
- Deneysel verilerle analiz yapılır.

Yürütme Zamanı

- Algoritmanın işlevini yerine getirebilmesi için kabul edilen işlemlerden kaç adet yürütülmesi gerektiğini gösteren matematiksel bir ifadedir.
- İşlem olarak karşılaştırma sayısı, çevrim sayısı, aritmetik işlem sayısı kabul edilir
- T(n) hesaplanırken hangi işleme göre hesaplandığı bildirilmelidir

Problemler Temel İşlemler

Problem	Giriş verisi	Temel işlemler
n elemanlı bir dizide eleman arama	Dizinin eleman sayısı. n	Elemanları karşılaştırma
İki matrisin çarpımı	Matris boyutu veya toplam eleman sayısı	İki sayının çarpımı
İnteger bir sayının asal olup olmadığının kontrolü	N sayısının dijit sayısı (ikili gösterilim)	Bölme
Graf problemi	Düğüm / Kenar sayısı	Düğüm ve kenarların gezilmesi

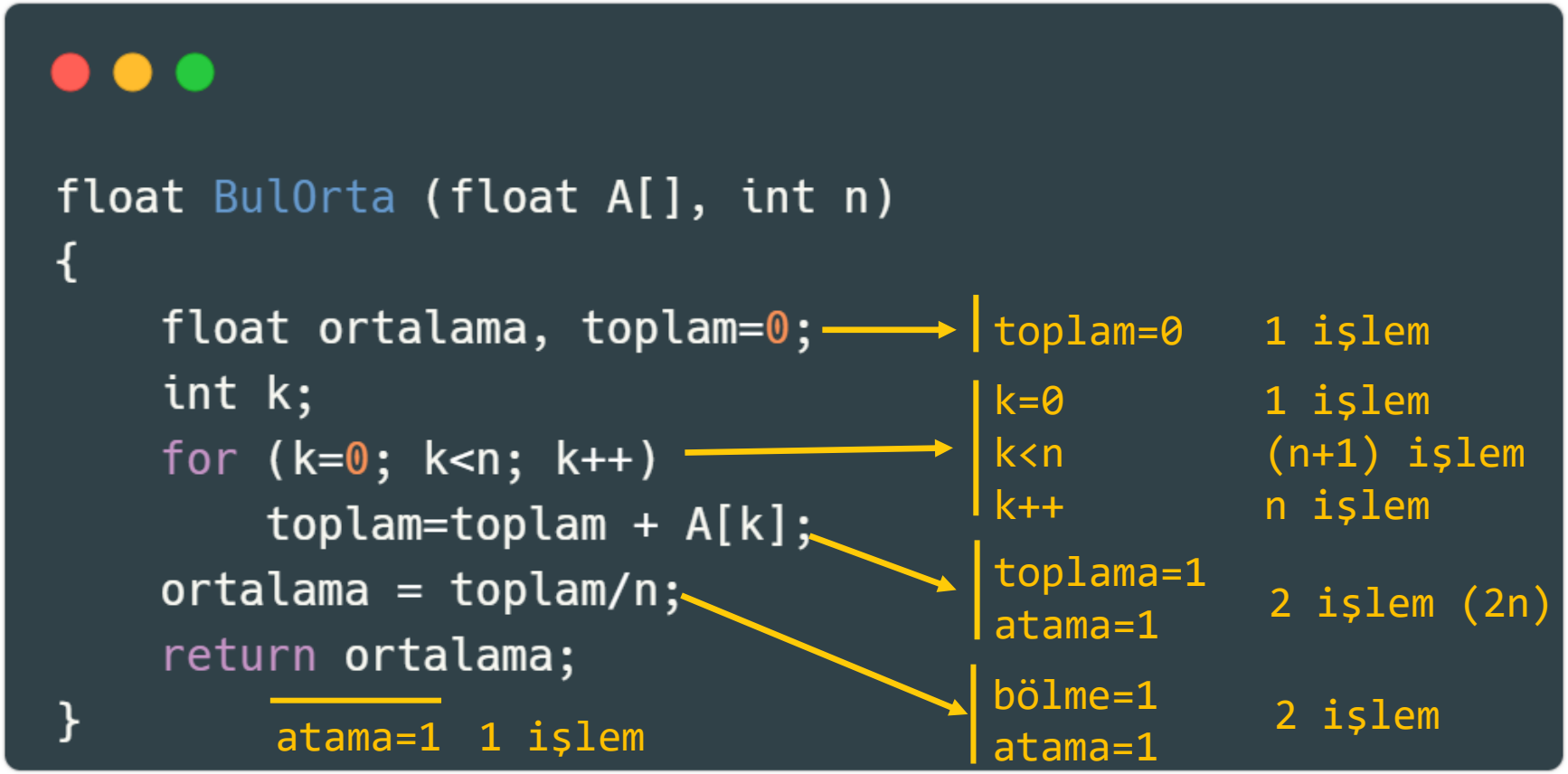
Yürütme Zamanı — *T(n)*

- $T(n) = 2n^2 – 2n + 5$ olabilir.
- n, ifadenin bağımsız değişkenidir ve temel işlem sayısına bağlıdır.

$T(n) = 1$	sabit
$T(n) = \log n$	logaritmik
$T(n) = n$	lineer
$T(n) = n^2$	karesel

Yürütme Zamanı

Örnek

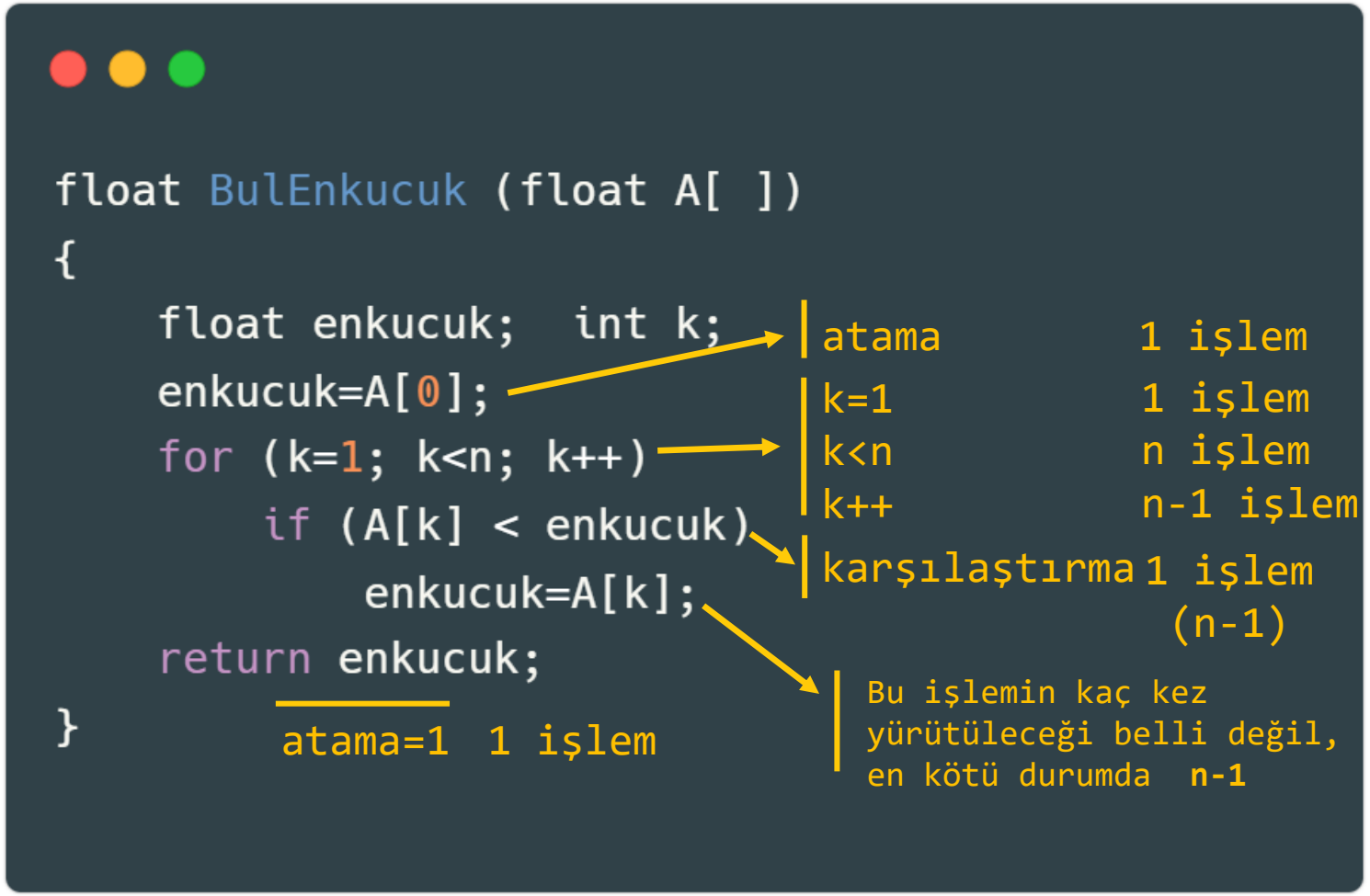


$T(n)=1+1+(n+1)+n+2n+2+1 = 4n+6$

$T(n)=4n+6$

Yürütme Zamanı

Örnek



$T(n)=1+1+n+(n-1)+(n-1)+(n-1)+1$

$T(n)=4n$

Yürütme Zamanı

Örnek

```
public static int[] selectionsort(int[] A, int n)
{
    int tmp;
    int min;

    for (int i = 0; i < n-1 ; i++) I
    {
        min = i; II
        for (int j=i ; j < n ; j++) III
        {
            if (A[j] < A[min]){ IV
                min = j; V
            }
        }
        tmp = A[i];
        A[i] = A[min];   VI
        A[min] = tmp;
    }
    return A;  VII
}
```

	İşlem	Tekrar	Toplam
I	1,1,1	1,n,n-1	2n
II	1	n-1	n-1
III	1,1,1	n-1, (n-1)n/2+1, (n-1)n/2	n^2-1
IV	1	(n-1)n/2	(n-1)n/2
V	1	(n-1)n/2	(n-1)n/2
VI	1,1,1	1,1,1	3
VII	1	1	1
$2n^2+2n+2$			

Yürütme Zamanı

Örnek

```
for (i=0; i<n; i++)                  → 1+(n+1) +n = 2n + 2
    for (j=0; j<m; j++)              → (1+(m+1) + m) * n = (2m + 2)n
        C[i][j] = A[i][j] + B[i][j]; → (2 * m * n)
```

4mn + 4n + 2

m ve n eşit alırsak $T(n) = 4n^2 + 4n + 2$

Karmaşıklık

- Algoritmaları karşılaştırabilmek için bir algoritmanın zorluk derecesi ölçümüne “Computational Complexity” denir.
- Computational complexity bir algoritmanın gerçekleştirilmesi için gereken maliyeti veya çabayı ifade eder. Maliyet veya çaba zaman (time) ve kullanılan alan (space) ile ifade edilir.
- Karmaşıklıktaki amaç, gerçek zaman ve bellek büyüklüğü bilgisi değil, veri kümesi büyüdüğünde maliyet bilgisinin değişimidir.

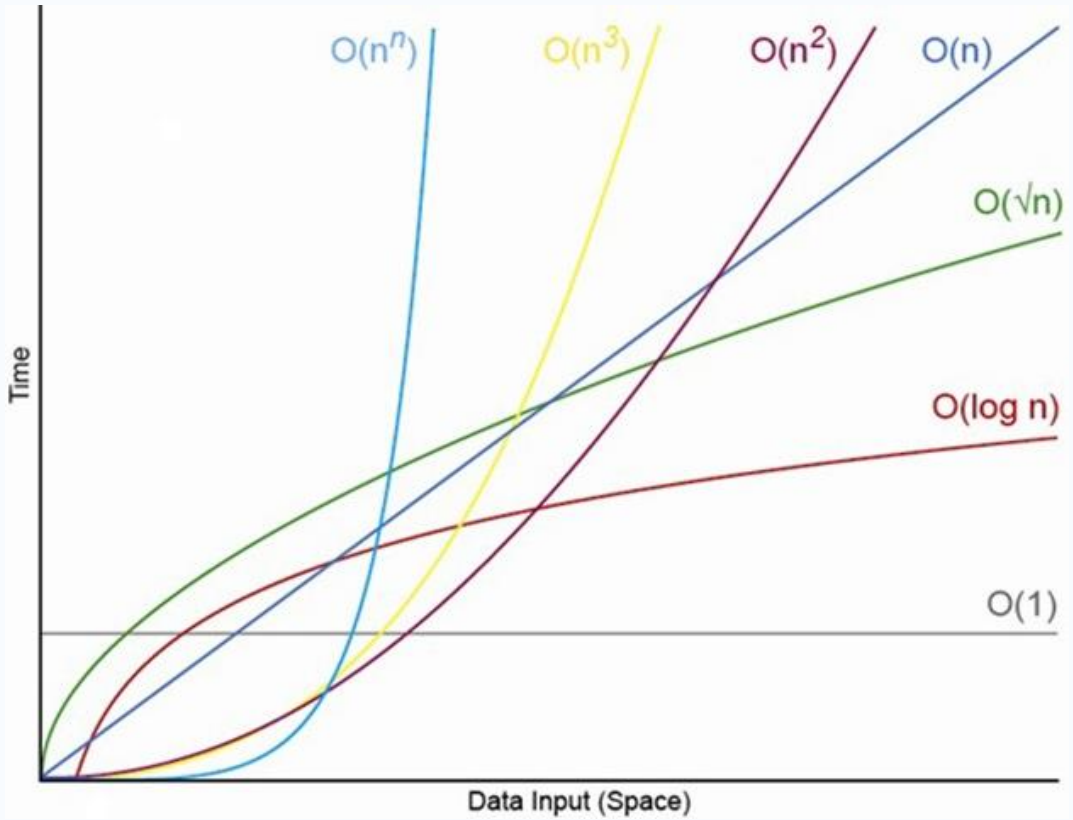
Asimptotik Karmaşıklık

- Algoritmalarda t (süre) ve n (giriş boyutu) arasındaki ilişki çoğu zaman çok karmaşıktır.
- Fonksiyon içerisindeki önemsiz kısımlar ve katsayılar atılarak basitleştirilir ve gerçek fonksiyona göre yaklaşık bir değer bulunur.
- Elde edilen bu yeni etkinlik ölçümüne “Asymptotic Complexity” denir.
- Genellikle girişin büyümesine bağlı olarak fonksiyonun büyümesinde en büyük etkiye sahip olan parametre alınır.

Asimptotik Karmaşıklık

- Bilgisayar biliminde, girdinin boyutu büyüdükçe, bir algoritmanın bir sorunu ne kadar hızlı çözebileceğini anlamak isteriz.
  - Aynı problemi çözmek için iki farklı algoritmanın verimliliğini karşılaştırabiliriz
  - Girdi büyüdükçe belirli bir algoritmayı kullanmanın pratik olup olmadığını da belirleyebiliriz

Asimptotik Karmaşıklık



n	$\log_2 n$	n	$n \log_2 n$	n^2	n^3	2^n	$n!$
10	3.3	10^1	$3.3 \cdot 10^1$	10^2	10^3	10^3	$3.6 \cdot 10^6$
10^2	6.6	10^2	$6.6 \cdot 10^2$	10^4	10^6	$1.3 \cdot 10^{30}$	$9.3 \cdot 10^{157}$
10^3	10	10^3	$1.0 \cdot 10^4$	10^6	10^9		
10^4	13	10^4	$1.3 \cdot 10^5$	10^8	10^{12}		
10^5	17	10^5	$1.7 \cdot 10^6$	10^{10}	10^{15}		
10^6	20	10^6	$2.0 \cdot 10^7$	10^{12}	10^{18}		

Asimptotik Analiz

- Amaç: detaylardan kurtularak çalışma süresi analizini basitleştirmek
- Sayılar için “rounding” işlemi: $1{,}000{,}001 \approx 1{,}000{,}000$
- Fonksiyonlar için “rounding” işlemi: $3n^2 \approx n^2$
- Niteliğini belirlemek (Capturing the essence): belirlenen limit içerisinde girişin boyutuna göre algoritmanın çalışma süresinin nasıl arttığının bulunması

Asimptotik Analiz Türleri

- **Worst case (en kötü):** Algoritma çalışmasının en fazla sürede gerçekleştiği analiz türüdür.
- En kötü durum, çalışmazamanında bir üst sınırdır ve o algoritma için verilendurumdan *“daha uzun sürmeyeceği”* garantisi verir.
- Bazı algoritmalar için en kötü durum oldukça sık rastlanır.
- Arama algoritmasında, aranan öğe genellikle dizide olmaz, dolayısıyla döngü N kez çalışır.

Asimptotik Analiz Türleri

- **Best case (en iyi):** Algoritmanın en kısa sürede ve en az adımda çalıştığı giriş durumu olan analiz türüdür. Çalışma zamanında bir alt sınırdır.
- **Average case (ortalama):** Algoritmanın ortalama sürede ve ortalama adımda çalıştığı giriş durumu olan analiz türüdür.

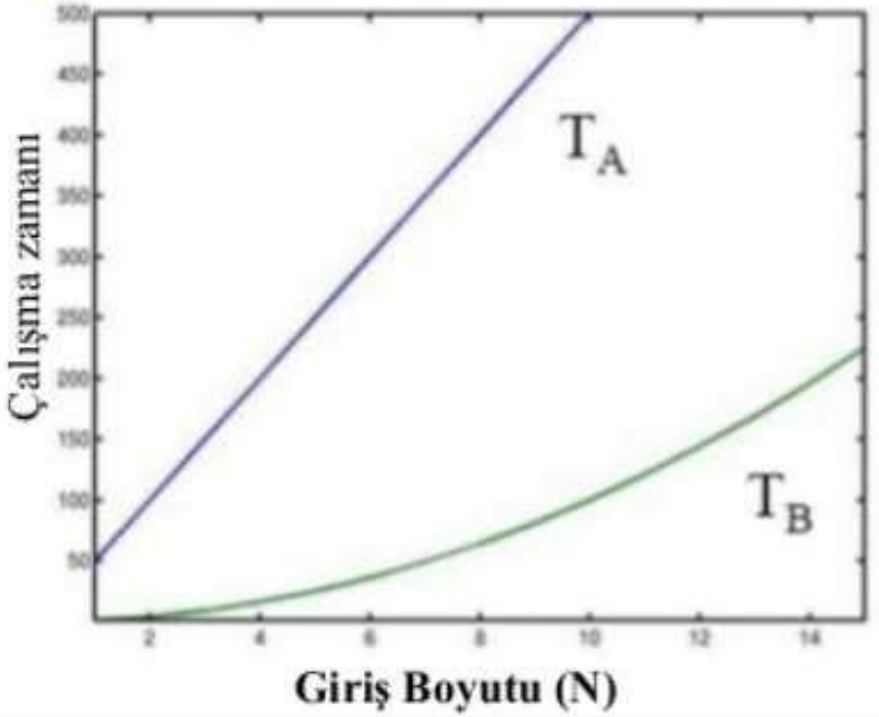
Asimptotik Analiz Türleri

- Bir algoritmanın genelde EN KÖTÜ durumdaki çalışma zamanına bakılır. Neden?
  - En kötü durum çalışma zamanında bir üst sınırdır ve o algoritma için verilen durumdan daha uzun sürmeyeceği garantisi verir.
  - Ortalama çalışma zamanı genellikle en kötü çalışma zamanı kadardır. Arama algoritması için hem ortalama hem de en kötü çalışma zamanı *doğrusal fonksiyondur*.

Asimptotik Analiz Türleri

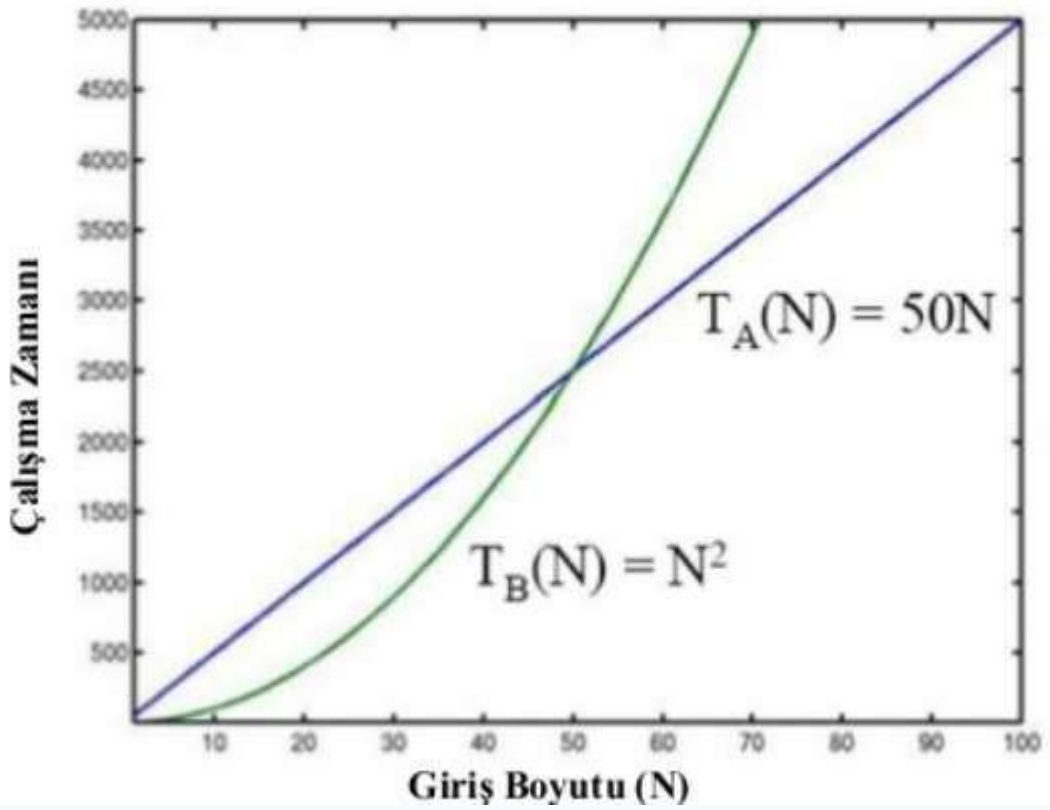
- Bir problemi çözmek için **A ve B** şeklinde iki algoritma verildiğini düşünelim.
- Giriş boyutu N için aşağıda A ve B algoritmalarının çalışma zamanı **T_A ve T_B fonksiyonları** verilmiştir.

Hangi algoritmayı seçersiniz?



Asimptotik Analiz Türleri

- N büyüdüğü zaman A ve B nin çalışma zamanı:



Şimdi hangi algoritmayı seçersiniz?

Asimptotik Notasyon

- ***Asimptotik notasyon***, eleman sayısı n'nin sonsuza gitmesi durumunda algoritmanın, benzer işi yapan algoritmalarla karşılaştırmak için kullanilir.
- Eleman sayısının küçük olduğu durumlar mümkün olabilir fakat bu birçok uygulama için gecerli degildir.
- Verilen iki algoritmanin çalışma zamanı T1(N) ve T2(N) fonksiyonları şeklinde gosterilir.
- Hangisinin daha iyi olduğunu belirlemek için bir yol belirlememiz gerekiyor.
  - Big-O (Big 0): Asimptotik üst sınır
  - Big Ω (Big Omega): Asimptotik alt sınır
  - Big θ (Big Teta): Asimptotik alt ve üst sınır

f(x), bir algoritmanın fonksiyon şeklindeki gösterimi ise karmaşıklık O(f(x)), Ω(f(x)), ... şeklinde gösterilir.

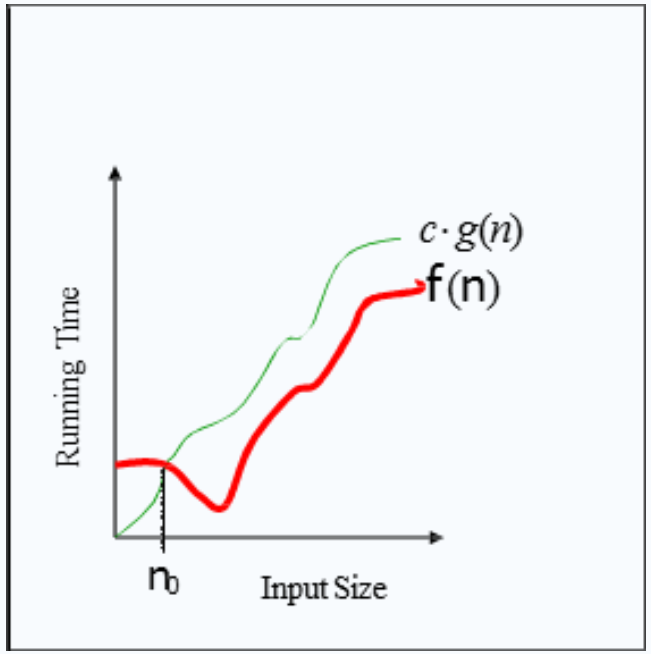
Big-O ifadesi

Asymptotic upper bound

- **Tanım: *f*** ve ***g***, tamsayı kümesinden veya reel sayı kümesinden reel sayılara tanımlanmış olsun.

$$\mathbb{Z}+ \rightarrow \mathbb{R}$$

- $f(n) = O(g(n))$,eğer sabit bir C ve n_0 değerleri için $f(n) \leq c\ g(n)$ bütün $n > n_0$ değerleri için doğruysa
- $f(n)\ ve\ g(n)$ pozitif değere sahip fonksiyonlardır.



Big-O ifadesi

Asymptotic upper bound

Show that $f(x) = x^2 + 2x + 1$ is $O(x^2)$

Since when $x > 1$, $x < x^2$ and $1 < x^2$

$$0 \leq x^2 + 2x + 1 \leq x^2 + 2x^2 + x^2 = 4x^2$$

Can take $C = 4$ and $k = 1$ as witnesses to show that

$f(x)$ is $O(x^2)$

Alternatively, when $x > 2$, we have $2x \leq x^2$ and $1 < x^2$.

Hence, $0 \leq x^2 + 2x + 1 \leq x^2 + x^2 + x^2 = 3x^2$ when $x > 2$.

- Can take $C = 3$ and $k = 2$ as witnesses instead.

Big-O ifadesi

Asymptotic upper bound

- O-ifadesi için genellikle en basit formül kullanılır.

Örnek:

$3n^2+2n+5 = O(n^2)$

- Aşağıdaki örneklerde doğrudur ancak genellikle kullanılmazlar.

$3n^2+2n+5 = O(3n^2+2n+5)$

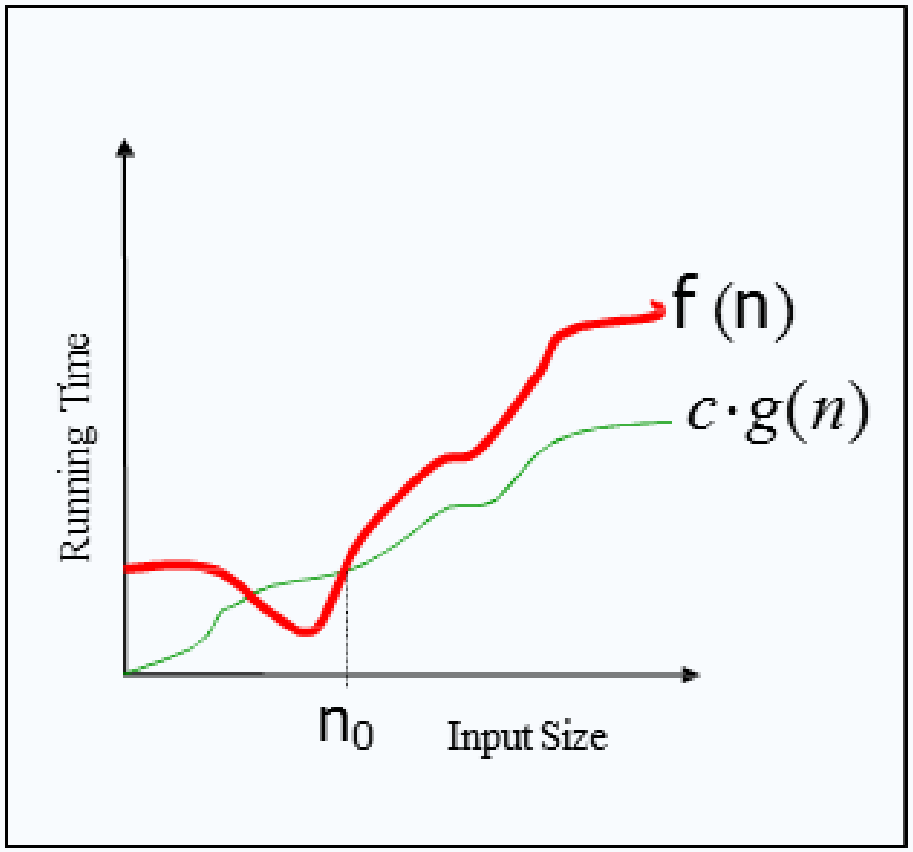
$3n^2+2n+5 = O(n^2+n)$

$3n^2+2n+5 = O(3n^2)$

Big-Ω ifadesi

Asymptotic lower bound

- *f(n) = Ω(g(n))*, eğer sabit bir *c* ve n_0 değeri için $c.g(n) \leq f(n)$ bütün $n \geq n_0$ değerleri için doğruysa
- Best-case çalışma süresi veya lower bound tanımlamasında kullanılır.



Big-Ω ifadesi

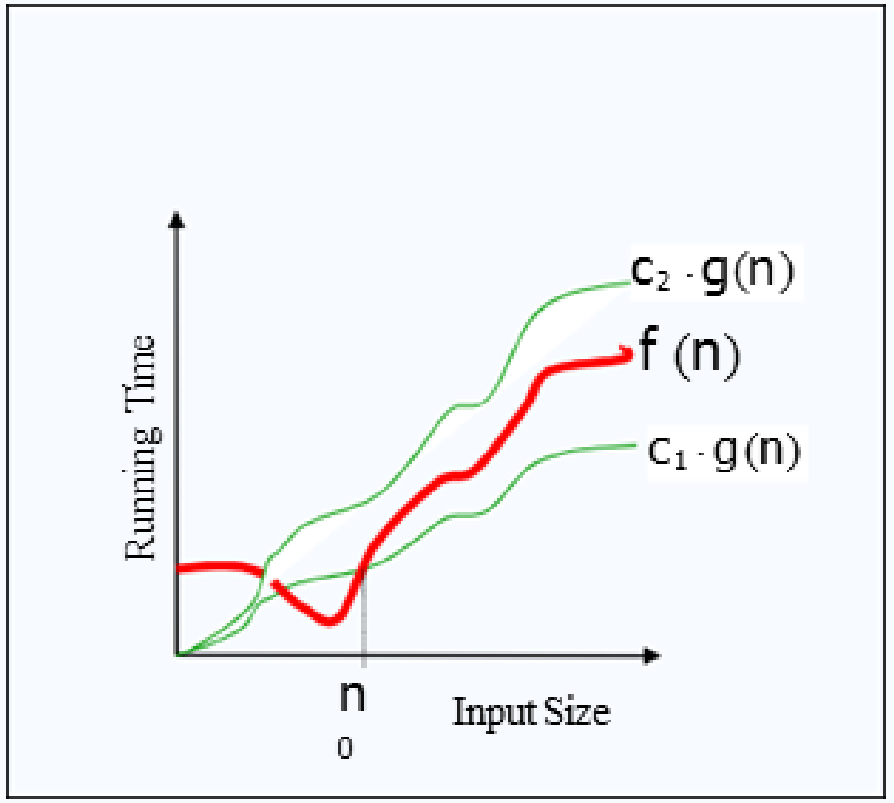
Asymptotic lower bound

- **Basit kural:** Küçük dereceden terimler ve sabitler atılır.
- *50 n log n* ifadesi *O(n log n)* şeklinde gösterilir.
- *7n - 3* ifadesi *O(n)*
- $8n^2 \log n + 5n2 + n$ ifadesi $O(n^2 \log n)$ şeklinde ifade edilir.

Big-θ ifadesi

Asymptotic tight bound

- $f(n) = \theta(g(n))$, eğer sabit c_1, c_2 , ve n_0 , değerleri için
- $c_1\, g(n) \leq f(n) \leq c_2\, g(n)$ bütün $n \geq n_0$ değerleri için
- doğruysa.



Asiptotik Analiz

- Birçok algoritma birden fazla alt programdan oluşabilir.

$f1 \rightarrow O(n^c)$
$f2 \rightarrow O(n^d)$
$1<c<d$ ise
$\max(O(n^c), O(n^d)) \rightarrow O(n^d)$

$f1 \rightarrow O(\log n)$
$f2 \rightarrow O(n)$
$\max(O(\log n), O(n)) \rightarrow O(n)$

$f1 \rightarrow O(2^n)$
$f2 \rightarrow O(n)$
$\max(O(2^n), O(n)) \rightarrow O(2^n)$

$f1 \rightarrow O(n^2)$
$f2 \rightarrow O(n^2)$
$\rightarrow O(n^2)$