Algorithm Analysis And Design

With Infinity Teams

Eng: Abdulrahman Hamdi

INFINITY TEAM
Since 2020

Table of contents

Eng: Abdulrahman Hamdi

04

Fundamentals of the Analysis of Algorithm Efficiency

Eng: Abdulrahman Hamdi

Algoritma Analizi Çerçevesi

• **Algoritma Analizinde Göz Önünde Bulundurulması Gerekenler Neler?**
– Algoritmanın Doğruluğu (Correctness)
– Zaman Verimliliği (Time Efficiency)
– Bellek Alanı Verimliliği (Space Efficiency)
• Gelişen donanım teknolojisi ile artık çok önemli değil
– Uygunluk, en iyilik (Optimality)

• Girdi Büyüklüğünün Ölçülmesi
• Çalışma Zamanı Ölçü Biriminin Belirlenmesi
• Büyüme Derecesi (Order of Growth)
• En kötü durum, en iyi durum, ortalama durum değerlendirmeleri

Algoritma Verimliliği

Algoritma verimliliği 3 başlık altında incelenir bunlar :

1. Verimlilik / Etkinlik (Efficiency)
2. Basitlik (Simplicity)
3. Genellik (Generality)

Bu üç başlık içerisinde ölçülebilir olan yani algoritmada bir iyileştirme yapıp yapmadığımızdan emin olabileceğimiz tek başlık verimliliktir.

•Bir algoritmanın verimliliğini iki alt başlıkta inceleyebiliriz bunlar :

1. Zaman Verimliliği (Time Efficiency)
2. Hafıza Kullanım Verimliliği (Space Efficiency)

Burada zaman verimliliği, bir algoritmanın ne kadar hızlı olduğuyla ilgilenir. Hafıza kullanım verimliliği ise bir algoritmanın bellekte ne kadar az yer kapladığı ile ilgilenir. Günümüzde artık bilgisayarlardaki depolama kapasiteleri çok yüksek olduğundan hafıza kullanım verimliliği zamanla önemini kaybetmiştir, fakat zaman verimliliği hala çok genel bir konudur. Bizde bu derste zaman verimliliği üzerinde duracağız.

Eng: Abdulrahman Hamdi

Algoritma Verimliliği

Zaman Verimliliğinin Ölçülmesi, Bir algoritma için zaman verimliliği saat, saniye, gün vb. olarak bilinen zaman birimleri ile ölçülmez. Çünkü böyle bir ölçüm, kullanılan işlemciye veya donanıma bağlı olarak değişebilir. Hatta bir algoritmanın aynı bilgisayarda farklı sürelerde çalışması olasıdır (bilgisayarın o anki iş yükü bunda etkili olabilir).

Bunu yerine bir algoritmanın zaman verimliliğini o algoritmanın aldığı girdinin bir fonksiyonu olarak ifade edeceğiz. Burada yaptığımız temel varsayım “algoritmalar büyük girdilerde yavaş, küçük girdilerde hızlı çalışır.” varsayımıdır. Bu şekilde algoritmanın büyük girdilerde ne kadar yavaş çalışabileceği hakkında bir fikrimiz olur.

Eng: Abdulrahman Hamdi

Girdi Büyüklüğünün Ölçülmesi

Girdi Büyüklüğünün Ölçülmesi / İfade Edilmesi, Bir algoritma için girdi büyüklüğünün (input size) nasıl ifade edileceği algoritmadan algoritmaya değişir. Girdi olarak bir dizi alan bir algoritma için girdi büyüklüğü bu dizinin eleman sayısıdır (yani dizi uzunluğu). Girdi olarak bir matris alan bir algoritma için girdi büyüklüğü bu matristeki toplam eleman sayısı yani bu matristeki toplam satır ve sütun sayısının çarpımıdır.

Fakat eğer bir kare matris varsa (yani satır sayısı sütun sayısına eşitse) toplam eleman yerine satır veya sütun sayısı kullanılır. Girdi olarak bir tamsayıyı alan bir algoritma için girdi büyüklüğü genelde bu sayının ikilik sistemdeki basamak sayısıdır öyle ki bir k tamsayısı için bu $\mathbf{b = \lfloor \log_2 k \rfloor + 1}$ İle bulunabilir (burada L ⌋ operatörü floor operatörüdür).

Eng: Abdulrahman Hamdi

Girdi Büyüklüğünün Ölçülmesi

– Tüm algoritmalar için büyük girdiler üzerinde çalışma süresi daha uzundur
• Büyük dizi içinde arama
• Büyük boyutlu matrisleri çarpma
– Algoritmaya girilen veriyi belirten n değerinin belirlenmesi
önemli
• Bazı algoritmalar için kolay
– Arama, sıralama, eleman sayısı bulma
• Bazı algoritmalar için değil
– İki matrisin çarpımı
» Matrislerin derecesi mi? Eleman Sayıları mı?
– Yazım hatası programı
» Karakter Sayısı mı? Kelime Sayısı mı?

Zaman Verimliliği

Tanım:
Bir algoritmanın zaman verimliliği bu algoritmanın temel operasyonunun (basic operation) toplam çalışma sayısıyla ölçülür ve T(n) ile gösterilir (burada n algoritmanın girdi büyüklüğünü temsil etmektedir).

Temel operasyona örnek verilmek istenirse, bir sıralama algoritması için iki sayıyı kıyaslamadır, ilk hafta notlarında gösterilen ebob'u bulan Öklid algoritması için mod almadır. Yani temel operasyon bir algoritmada en çok tekrar eden, algoritmanın toplam çalışma süresi içersinde en çok zaman harcanan operasyondur.

Eng: Abdulrahman Hamdi

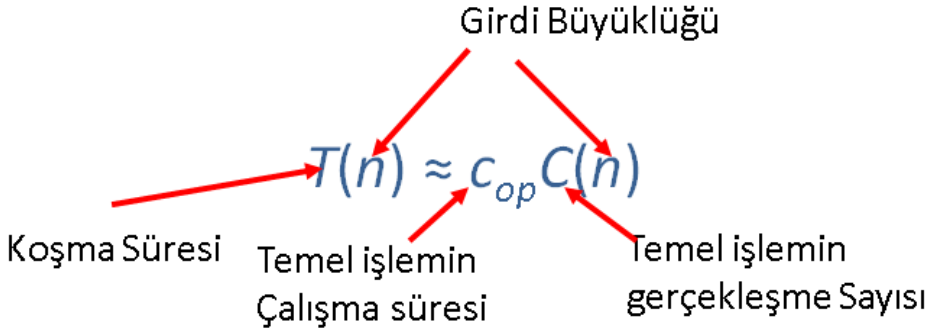
Zaman Verimliliği

Bir Algoritmanın Çalışma Zamanı
– Çalıştığı bilgisayar sisteminin hızı
– Algoritmanın kullanıldığı programın kalitesine
– Makine kodunu üreten derleyici gibi etkenlere bağımlıdır

Bu yüzden dış etkenlere bağımlı olmayan bir ölçüm yolu bulunmalıdır
– Bir yaklaşım: Algoritma içerisindeki tüm işlemlerin kaç kere gerçekleştiğinin sayılması
• Zor ve Gereksiz
– Bir diğer yaklaşım: Algoritma içerisindeki temel işlemin belirlenip, kaç kere gerçekleştiğinin sayılması

Zaman Verimliliği

• Zaman verimliliğinin teorik incelenmesi
– Zaman verimliliği girdi üzerindeki temel işlemin tekrar sayısı üzerinden değerlendirilir
– Temel işlem
• Algoritmanın çalışma süresince en çok gerçekleştirilen işlem



Zaman Verimliliği

• Temel işlemin belirlenmesi
– Genellikle en çok gerçekleşen işlem
– Genellikle en iç döngüde yapılan işlem
• Sıralama algoritmaları için karşılaştırma
• Matematiksel işlemler için genellikle 4 işlem
– / en uzun işlem, sonra *, + ve –

• c_{op}:
– Bir algoritmanın temel işleminin bir bilgisayardaki çalışma süresi
• $C(n)$:
– Temel işlemin gerçekleşme sayısı
• $T(n)$:
– Algoritmanın uygulandığı programın çalışma süresi
• $T(n) \approx c_{op}\ C(n)$
– Bu formülün yaklaşık ve tahmini bir değer verdiği unutulmamalı

Zaman Verimliliği

$$T(n) \approx c_{op}\, C(n)$$

– Bu programı 10 kat hızlı bir bilgisayarda çalıştırsak ne kadar hızlı sonuç alırız?

• Cevap : 10 kez

$C(n) = \frac{1}{2}n(n-1)$ ise girdiyi iki kat büyütürsek çalışma zamanı ne kadar artar?

$$C(n) = \frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \approx \frac{1}{2}n^2$$

$$\frac{T(2n)}{T(n)} \approx \frac{c_{op}C(2n)}{c_{op}C(n)} \approx \frac{\frac{1}{2}(2n)^2}{\frac{1}{2}n^2} = 4.$$

Zaman Verimliliği

Şimdi n satıra ve n sütuna sahip iki kare matrisin toplamını bulan bir algoritma için T(n)'I hesaplayalım.

```
matrisTopla(A[0,…,n-1,0,…,n-1], B[0,…,n-
    1,0,…,n-1])
// Girdi: n satır ve n sütundan oluşan A
    ve B matrisleri
// Çıktı: A ve B matrislerinin toplamı C

    for i = 0'dan n-1'e // n kere tekrar
    eder
    for j = 0'dan n-1'e // n kere tekrar
    eder
        C[i, j] = A[i, j] + B[i, j]

    return C
```

bu algoritmanın temel operasyonu toplamadır ve n satır, n sütun için $n \times n = n^2$ kez toplam yapılır. O halde $T(n) = n^2$ olur.

soru: matrisTopla algoritmasının aldığı matrisler iki kat büyürse yanı yeni alınan matrisler 2n satır ve sütundan oluşursa bu algoritma kaç kat yavaşlar?

cevap: Yeni durum için zaman verimliliği $T(2n) = n^2$ olur. Bu verimlilik bir önceki durumla

Kıyaslanırsa $\frac{T(2n)}{T(n)} = \frac{4n^2}{n^2} = 4$ olup matrisTopla algoritmasının toplam çalışma süresi toplam 4 kat artar.

Eng: Abdulrahman Hamdi

Zaman Verimliliği

Şimdi yine n satır ve n sütuna sahip iki kare matrisin çarpımını dönen bir algoritma yazalım ve T(n)'i hesaplayalım. Bu hesaptan önce şu bilgiye göz atalım.
BİLGİ: n x n boyutundaki A ve B matrislerinin çarpımından oluşan C matrisini i. satırının j. elemanı (C[i,j]) A'nın i. satırı ve B'nin j. sütunun iç çarpımından elde edilir. İki vektörün iç çarpımı da karşılıklı elemanların çarpılıp, bu çarpımların toplanmasıyla oluşan bir sayıdır.

```
matrisCarp(A[0,…,n-1,0,…,n-1], B[0,…,n-
    1,0,…,n-1])
// Girdi: n satır ve n sütundan oluşan A
    ve B matrisleri
// Çıktı: A ve B'nin çarpımı olan C
    matrisi

    for i = 0'dan n-1'e // n kez çalışır
     for j = 0'dan n-1'e // n kez çalışır
         C[i, j] = 0
         for k = 0'dan n-1'e // n kez
    çalışır
                C[i, j] += A[i, k] * B[k, j]

    return C
```

Eng: Abdulrahman Hamdi

Zaman Verimliliği

```
matrisCarp(A[0,…,n-1,0,…,n-1], B[0,…,n-
    1,0,…,n-1])
// Girdi: n satır ve n sütundan oluşan A
    ve B matrisleri
// Çıktı: A ve B'nin çarpımı olan C
    matrisi

    for i = 0'dan n-1'e // n kez çalışır
     for j = 0'dan n-1'e // n kez çalışır
         C[i, j] = 0
         for k = 0'dan n-1'e // n kez
    çalışır
                 C[i, j] += A[i, k] * B[k, j]

    return C
```

Burada $n \times n \times n = n^3$ kez toplama yapılır. O halde $\mathrm{T}(n) = n^3$ 'tür.
Girdi büyüklüğünün iki kata çıkması durumunda $\frac{\mathrm{T}(2n)}{\mathrm{T}(n)} = \frac{8n^3}{n^3} = 8$ olup bu algoritma 8 kat yavaşlar. Yani başka bir deyişle 8 kat fazla bir zamana ihtiyaç duyulur.

Eng: Abdulrahman Hamdi

Zaman Verimliliği

Şimdi aldığı n büyüklüğünde bir dizinin en büyük elemanını dönen bir algoritma yazalım ve T(n)'i hesaplayalım.

```
maxBul(A[0,…,n-1])
// Girdi: n elemanlı A dizisi
// Çıktı: A'nın en büyük
   elemanı

   maxDeger ← -∞
for i = 0'dan n-1'e
    if A[i] > maxDeger
        maxDeger ← A[i]

   return maxDeger
```

Bu algoritmada en çok tekrar eden, toplam çalışma zamanın en büyük bölümünü tüketen iki sayının kıyaslanmasıdır, bu temel operasyondur ve n kez tekrarlanır. O halde maxBul için $T(n)=n'$dir. Bu algoritma için girdi büyüklüğü 3 katına çıkar, 3 kat büyük diziler alırsa, $T(3n)=3n$ olur. Yani zaman verimliliği 3 kat azalır. Önceki iki algoritmanın aksine algoritmanın çalışması için ihtiyaç duyulan zaman üstel olarak değil lineer olarak artar.

Zaman Verimliliği

Soru: Bir algoritmanın zaman verimliliği her zaman yalnızca girdisinin büyüklüğüne mi bağlıdır? Bir algoritmanın ne kadar sürede çalışacağına girdi büyüklüğü haricinde başka faktörler de etki edebilir mi?
Bu sorunun cevabı için aşağıdaki algoritmaya bakalım. Basitçe bu algoritma girdi olarak aldığı n büyüklüğündeki bir dizinin içinde bir K değerini arar, eğer K bu dizinin içinde ise K'nın bulunduğu dizinin indisine döner, eğer K yoksa -1'e döner.

```
sıralıArama(A[0,…,n-1], K)
// Girdi: n elemanlı A dizisi, K
   değeri
// Çıktı: K, A'nın içinde ise K'nın
   bulunduğu indis, değilse -1

   i ← 0 // Arama indisi
while i < n ve A[i] ≠ K // K
   bulunmuyorken ve dizinin içindeyken
    i += 1

   if i < n ise // Bu, K'nın
   bulunduğunu gösterir
    return i
else
    return -1
```

Eng: Abdulrahman Hamdi

Büyüme Derecesi

– Küçük girdi boyutları ile bir algoritmanın etkinliğinin değerlendirilmesi sağlıklı değil

- Algoritmaların analizi için önemli olan çeşitli fonksiyonların değerleri (bazıları yaklaşık)

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}		

En İyi, En Kötü, Ortalama Durum

• **En kötü durum**
– n boyutunda girdi üzerinden en yüksek değer
• **En iyi durum**
– n boyutunda girdi üzerinden en düşük değer
• **Ortalama durum**
– n boyutunda girdi için «ortalama» değer
• Tipik bir girdi için temel işlemin kaç kere olduğu
• En kötü ile en iyinin ortalaması değil
• Temel işlem sayısı olarak bir olasılık dağılımı içerisinden rastgele bir değişken değeri beklenir

- **En Kötü Durum**

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$
//Searches for a given value in a given array by sequential search
//Input: An array $A[0..n-1]$ and a search key K
//Output: The index of the first element in A that matches K
// or -1 if there are no matching elements
$i \leftarrow 0$
while $i < n$ **and** $A[i] \neq K$ **do**
$i \leftarrow i + 1$
if $i < n$ **return** i
else return -1

Eng: Abdulrahman Hamdi

En İyi, En Kötü, Ortalama Durum

• Sıralı arama için
– En kötü durum: Aranan değerin dizide bulunmaması
• N boyutunda girdi için maksimum sayıda karşılaştırma
– $C_{Worst}(n) = n$
• En kötü durum incelemesi bir algoritmanın çalışma zamanı açısından üst sınırını belirler

- **En İyi Durum İncelemesi**

ALGORITHM *SequentialSearch*$(A[0..n-1], K)$
//Searches for a given value in a given array by sequential search
//Input: An array $A[0..n-1]$ and a search key K
//Output: The index of the first element in A that matches K
// or -1 if there are no matching elements
$i \leftarrow 0$
while $i < n$ **and** $A[i] \neq K$ **do**
$i \leftarrow i + 1$
if $i < n$ **return** i
else return -1

En İyi, En Kötü, Ortalama Durum

Bu algoritmanın kaç adımda sonuca ulaşacağı her zaman kesin değildir. Eğer şanslı isek K, dizinin ilk (0.indis) hücresindedir. Bu durumda toplam adım sayısı 1 olur ve $T(n) = 1$ diyebiliriz. Bu en iyi durum senaryosu zaman verimliliğidir (best-case time efficiency). Bazı kaynaklar bu durumda $T(n)$ yerine $C_{best}(n)$ gösterimini kullanırlar.

Yanındaki durumun zıttı olarak K, dizinin son hücresinde olabilir ya da dizide hiç olmayabilir. Bu durumda sıralıArama algoritması maksimum sayıda yani n defa çalışır ve bu durumda $T(n) = n$ olur. Bu en kötü durum zaman verimliliğidir ve bazen $C_{Worst}(n)$ ile gösterilir (worst-case time efficiency)

Eng: Abdulrahman Hamdi

En İyi, En Kötü, Ortalama Durum

- **Ortalama Durum İncelemesi**

Bu şekilde girdinin başlangıç durumuna göre algoritmanın zaman verimliliğinin değiştiği durumlarda genelde $T_{Worst}(n)$ yeni en kötü durum verimliliği dikkate alınır. Böylece başımıza gelebilecek en kötü durumda algoritmanın ne kadar verimli olabileceği hakkında fikrimiz olur.
En iyi durum zaman verimliliği ve en kötü durum zaman verimliliğine ek olarak -bazı durumlardaortalama durum zaman verimliliği (average-case time efficiency) de hesaplanabilir.

Basitçe, bu "ortalama olarak elimizdeki algoritma ne kadar verimlidir" sorusuna bir cevap bulmamızı sağlar.
Ortalama durum zaman verimliliğini hesaplarken olasılık hesabından faydalanırız. Temel olarak $T(n)$ bir (ayrık) rastgele değişkendir, başlangıç durumuna göre değişir. $T(n)$'nin beklenen değeri $E[T(n)]$ bize ortalama durum zaman verimliliğini verir.

Eng: Abdulrahman Hamdi

En İyi, En Kötü, Ortalama Durum

Hatırlarsak, diyelim ki bir X rastgele değişken $x_1, x_2, x_3, \dots, x_k$ değerlerinden birini alsın. Bu durumda X'in beklenen değeri $\mathrm{E}[X] = x_1.P(X = x_1) + x_2.P(X = x_2) + x_3P(X = x_3) + \cdots + x_k\,P(X = x_k)$ ile hesaplanır.
Şimdi sıralıArama algoritması için ortalama durum zaman verimliliğini hesaplayalım.
Varsayalım ki K'nın arama yapılan dizinin içinde olma olasılığı p olsun (bu durumda bu dizinin içinde olmama olasılığı $(1-p)$ olur). Ve yine varsayalım ki K'nın dizinin her bir hücresinde olma olasılığı aynı olsun. Bu durumda K'nın n uzunluğundaki bir dizinin herhangi bir hücresinde olma olasılığı $\frac{\mathrm{p}}{n}$ olur.

K dizisinin ilk hücresinde ise $\mathrm{T}(n) = 1$ olur ve bunun olma olasılığı $\frac{\mathrm{p}}{n}$ 'dir.
K dizisinin ikinci hücresinde ise $\mathrm{T}(n) = 2$ olur ve bunun olma olasılığı $\frac{\mathrm{p}}{n}$ 'dir.
K dizisinin son hücresinde ise $\mathrm{T}(n) = n$ olur ve bunun olma olasılığı $\frac{\mathrm{p}}{n}$ 'dir.
Ya da K dizinde olmaz. Bu durumda dizinin n adet hücresinin tamamı tarandığı için $\mathrm{T}(n)$ yine n olur ve bunun olma olasılığı $(1-p)$ olur.

Eng: Abdulrahman Hamdi

En İyi, En Kötü, Ortalama Durum

$$C_{avg}(n) = [1 \cdot \frac{p}{n} + 2 \cdot \frac{p}{n} + \cdots + i \cdot \frac{p}{n} + \cdots + n \cdot \frac{p}{n}] + n \cdot (1-p)$$
$$= \frac{p}{n}[1 + 2 + \cdots + i + \cdots + n] + n(1-p)$$
$$= \frac{p}{n}\frac{n(n+1)}{2} + n(1-p) = \frac{p(n+1)}{2} + n(1-p).$$

Şimdi bunu biraz inceleyelim. Eğer K bu dizinin içinde ise p=1 olur.

Bu durumda $C_{avg}(n) = \frac{n+1}{2}$ olur. Bu da şu demektir aradığımız değer dizinin içerisinde ise sıralı Arama algoritması ile dizinin yaklaşık olarak yarısını gezeriz ya da yarısından biraz fazla gezeriz.

Eğer aranan değer bu dizinin içinde değilse $p = 0$ olur. Bu durumda $C_{avg}(n) = C_{worst}(n) = n$ olur. Yani algoritma n kez çalışır, dizinin tamamı gezilir.
Sıralı arama için:
– Standart kabule göre
başarılı bir aramanın
olasılığı $p\ (0 \leq p \leq 1)$
– İlk karşılaştırmada
bulma olasılığı her hangi bir i değeri için aynı. $\frac{p}{n}$
– Bulunamama olasılığı
• $n.(1-p)$

Eng: Abdulrahman Hamdi

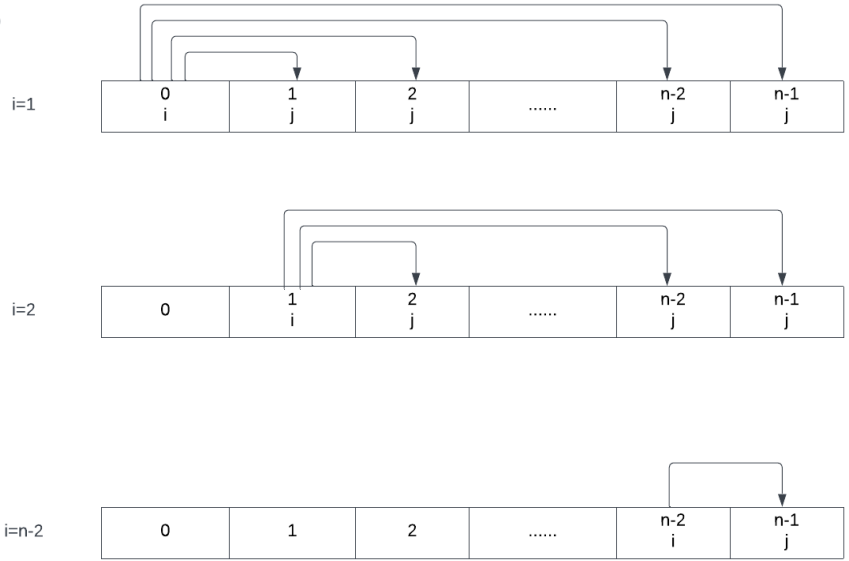
En İyi, En Kötü, Ortalama Durum

soru: Aldığı bir dizide birbirinin aynı iki eleman olup olmadığına karar veren aşağıdaki algoritma için $C_{worst}(n)$ yani en kötü durum zaman verimliliği nedir ?

```
aynıElemanVarMı(A[0,…,n-1])
// Girdi: n elemanlı A dizisi
// Çıktı: Eğer A'da birbirinin
    aynı elemanlar varsa true,
    yoksa false

    for i = 0'dan n-2'ye
     for j = i+1'den n-1'e
         if A[i] == A[j]
             return true

    return false
```



Eng: Abdulrahman Hamdi

En İyi, En Kötü, Ortalama Durum

En kötü durumda birbirinin aynı elemanlar dizinin son iki hücresindedir. Ya da dizide birbirinin hiç aynı eleman yoktur. İki durumda da maksimum sayıda karşılaştırma yapılır ve algoritma en uzun çalışma zamanına sahip olur. Bu durum olduğunda toplam kaç defa kıyaslama olacağını sayalım, bu da bize $C_{worst}(n)$'i verir.

i=0 için j=1, 2, …. , n-1 değerini alır. Toplamda n-1 adet kıyaslama olur.
i=1 için j=2, …. , n-1 değerini alır. Toplamda n-2 adet kıyaslama olur.
i=2 için j=3, 4, …. , n-1 değerini alır. Toplamda n-3 adet kıyaslama olur.
.
.
i=n-2 için j=n-1 değerini alır. 1 adet kıyaslama yapılır.
Tüm bu kıyaslama sayılarını toplarsak:

$$C_{worst}(n) = (n-1) + (n-2) + \cdots + 1 = \frac{(n-1).n}{2}$$

Eng: Abdulrahman Hamdi

Asimptotik Analiz

• **Asimptot:**
Bir fonksiyonun asimptotu, bu fonksiyonunun sonsuzda (yani sonsuz büyüklükte değerler alırken) yaklaştığı doğru ya da eğridir.
Biz de T(n)'nin çok büyük girdiler (büyük n'ler) alması durumunda en fazla ne kadar büyüyebileceğine karar verebilmek için T(n)'nin asimptotlarını bulacağız.

• Büyüme derecesi değerlendirilirken 3 farklı notasyon kullanılır
– **O (Big Oh)**
– **Ω (Big Omega)**
– **Θ (Big Theta)**

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Yukarıdan aşağı ok yönünde büyüme oranları artmaktadır

Not: Eğer n^2'mi yoksa 2^n'mi daha hızlı büyür diye karıştırıyorsanız şöyle düşünebilirsiniz:
n^2'de n yalnızca bir tabandır.
- Ne kadar büyük olursa olsun iki defa çarpım olur: n x n
- 2^n'de ise n tekrar sayısıdır, toplam çarpım sayısıdır; n arttıkça toplam çarpım sayısı artar.

Sınıf	İsim	Açıklamalar
1	**sabit** (*constant*)	En iyi durum verimlilikleri dışında, makul örnekler çok azdır çünkü bir algoritmanın çalışma süresi genellikle giriş boyutu sonsuza ulaştığında sonsuza gider.
log n	**logaritmik** (*logarithmic*)	Genellikle, bir problemin boyutunu her yinelemede sabit bir faktörle azaltmanın bir sonucudur. Not: Bir logaritmik algoritma, tüm girdisini veya sabit bir oranını işleyemez; bunu yapan herhangi bir algoritma en az doğrusal çalışma süresine sahip olacaktır.
n	**doğrusal** (*linear*)	Bir listeyi tarayan algoritmalar (örneğin, sıralı arama) bu sınıfa girer.
n log n	**doğrusal-logaritmik** (*linearithmic*)	Birçok böl ve fethet algoritması, örneğin, merge sort ve quicksort'un ortalama durumu bu kategoriye girer.
n^2	**kuadratik** (*quadratic*)	Genellikle, iç içe geçmiş iki döngü içeren algoritmaların verimliliğini karakterize eder. Temel sıralama algoritmaları ve n × n matrisler üzerindeki belirli işlemler standart örneklerdir.
n^3	**kübik** (*cubic*)	Genellikle, iç içe geçmiş üç döngü içeren algoritmaların verimliliğini karakterize eder. Doğrusal cebirden birkaç önemli algoritma bu sınıfa girer.
2^n	**üstel** (*exponential*)	Bir *n*-elemanlı kümenin tüm alt kümelerini üreten algoritmalar için tipiktir. Genellikle "üstel" terimi, bu ve daha büyük büyüme derecelerini içerecek şekilde daha geniş bir anlamda kullanılır.
n!	**faktöriyel** (*factorial*)	Bir *n*-elemanlı kümenin tüm permütasyonlarını üreten algoritmalar için tipiktir.

Eng: Abdulrahman Hamdi

Asimptotik Analiz

• **Big Oh Notasyonu:**
Basitçe O(g(n)) bir kümedir ve en fazla g(n) kadar büyüyebilen fonksiyonları içerir. (Ayrıca O(g(n)) dediğimize göre O'nun içinde n'nin bir fonksiyonu olacak, yani n'li bir şeyler)
Daha formal tanıma geçersek;
Eğer bir **t(n) ϵ O(g(n))** ise bir n_0 noktasındaki her n değeri için **t(n), c.g(n)**'den küçük kalır yani

t(n)≤ c.g(n)'dir

n_0 ve c burada O'dan büyük herhangi bir sabittir.

• **t(n) ϵ O(g(n))** olduğunu gösterebilmek demek: bir n_0 >0 noktası ve bir **c>0** değeri bulabilmek demektir. Öyle ki bu değerler için **t(n)≤c.g(n) ve** $n \geq n_0$ olur.

Çok Daha Formal:

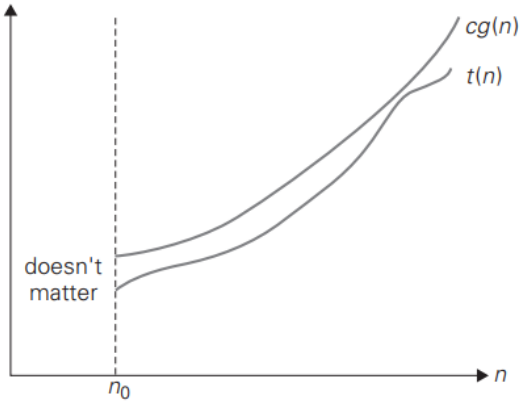
Vardır öyle ki, her

$$t(n) \in O(g(n)) \Leftrightarrow \exists\ n_0>0, \exists\ c>0\ \ni \forall\ n \geq n_0,\ t(n) \leq c.g(n)$$

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Çok Daha Az Formal:
Eğer **t(n) ϵ O(g(n))** olduğunu göstereceksiniz; bir n_0 ve bir c pozitif değerleri bulacaksınız. Öyle ki n_0 'dan büyük her n için **t(n)≤c.g(n)** olacak.



Big-oh notation: $t(n) \in O(g(n))$.

• **Not:** n0 noktası illa burası olmak zorunda değildir. n_0 burası da olabilir. Nasıl olsa bu noktadan sonra da **t(n), c.g(n)** 'den küçük kalmaktadır.

Özet: O(g(n)) :
– Bir g(n) fonksiyonu ile aynı veya daha düşük büyüme derecesine sahip fonksiyonların tümü
• Bir sabit katsayı ve n değeri sonsuza giderken

$$n \in O(n^2), \qquad 100n + 5 \in O(n^2), \qquad \frac{1}{2}n(n-1) \in O(n^2).$$

$$n^3 \notin O(n^2), \qquad 0.00001n^3 \notin O(n^2), \qquad n^4 + n + 1 \notin O(n^2).$$

Eng: Abdulrahman Hamdi

Asimptotik Analiz

100n + 5 ∈ O(n²) için ispat
100n + 5 ≤ 100n + n, n ≥ 5 ise
100n + 5 ≤ 101n
101n ≤ 101 n²
c = 101, n0 = 5 alınabilir
Soru: t(n)=3n²+4n−2 fonksiyonunun O(n²) olduğunu gösteriniz.
Çözüm:
Öyle bir c>0 ve n_0 >0 bulacağız ki, her n≥ n_0 için 3n²+4n−2≤c.n² olacak.
n≥1 için 4n ≤ 4n² ve −2≤2n² olduğundan n_0 = 1
t(n)=3n² +4n−2≤3n² +4n² +2n² =9n² olup
c=9 olur, yani bir c değeri bulunabilir.
O halde t(n) ∈ O(n²)

• **Ayrıca not edin:**
T(n)=3n²+4n−2 ≤ 9n² ≤ 10 n² ≤ 230n² ≤1000n² ≤ ..... 'dir.
Yani bir tek C değeri değil, sonsuz C değeri bulunabilir.
Ama bize bir tanesi yeter.
Soru: Bir önceki soruda t(n)=3n²+4n−2 ∈ O(n²) bulduk. Peki aynı zamanda t(n) ∈ O(n³) müdür?
Çözüm:
t(n)=3n²+4n−2 ≤ 9n² bulduk. Aynı zamanda 9n² ≤ 9n³ olduğundan t(n) ≤ 9n³ olur.
Böylece bir n_0 ve bir c değeri için t(n) ≤ c.g(n) bulmuş oluruz. O halde t(n) ∈ O(n3) diyebiliriz.
Hatta t(n)= 3n² +4n−2 ≤ 9n² ≤ 9n³≤ $9n^4$≤ $9n^5$≤ .....
olduğundan aynı zamanda t(n) ∈ O(n^4),
t(n) ∈ O(n^5)..... hepsi olur.
Fakat biz genelde üst sınırların en küçüğünü kullanırız ve bu yüzden 2. dereceden bir t(n) fonksiyonu için t(n) ∈ O(n²) deriz.

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Soru: t_1(n)=3n, t_2(n)= $\frac{n}{4}$, t_3(n)= 6n+8 , t_4 (n)=10000n fonksiyonlarının hepsinin O(n) kümesine ait olduğunu gösteriniz.

Çözüm:

t_1(n)=3n ≤ 4n

t_2(n)= $\frac{n}{4}$ ≤ 1n

t_3(n)= 6n+8 ≤ 14n altı çizili değerler c değerleridir.

t_4(n)=10000n ≤ 110000n

Eşitsizlikleri her n≥1 için doğru olduğunda bu fonksiyonların hepsi O(n)'e aittir.

Not: n_0 illa her zaman bu şekilde 1 olmak zorunda değildir. Siz isterseniz n_0=2,3.... diyebilirsiniz.

Teorem:

t(n) ϵ O(g(n)) ve g(n) ϵ O(h(n)) ise t(n) ϵ O(h(n))

Kanıt: Bir fonksiyonun üst sınırı için bir üst sınır vardır. Bu fonksiyon bu üst sınırın da altındadır.

t(n) ϵ O(g(n)) olduğundan bir n_0>0 bir c_1>0 için t(n) ≤c_1.g(n)

g(n) ϵ O(h(n)) olduğundan bir n_1>0 bir c_2>0 için g(n) ≤c_2.h(n)

Buradan t(n) ≤c_1.g(n) ≤ $c_1.c_2$ .h(n)

O halde max(n_0, n_1)'den büyük her n için t(n) ≤ $c_1.c_2$.h(n) olur.

O yüzden t(n) ϵ O(h(n)) olur.

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Teorem:
$t_1(n) \in O(g_1(n))$ ve $t_2(n) \in O(g_2(n))$ olsun.
Bu durumda $t_1(n) + t_2(n) \in O(\max(g_1(n), g_2(n))$
Yani toplamlardan oluşan bir fonksiyon için O değeri, bu toplama katılan fonksiyonlar içinde en yüksek O değerine sahip fonksiyonun O değeridir

Örnek:

$t(n) = n^3 - 4n^2 + 3n - 8 + n^5$

$O(n^3)$ $O(n^2)$ $O(n)$ $O(1)$ $O(n^5)$

$t(n) \in O(\max(n^5, n^3, n^2, n, 1))$

$t(n) \in O(n^5)$

Kanıt:

$t_1(n) \in O(g_1(n))$ ise bir $n_0 > 0$ ve bir $c_1 > 0$ vardır:
Öyle ki her $n > n_0$ için $t_1(n) \leq c_1 \cdot g_1(n)$

$t_2(n) \in O(g_2(n))$ ise bir $n_1 > 0$ ve bir $c_2 > 0$ vardır:
Öyle ki her $n \geq n_1$ için $t_2(n) \leq c_2 \cdot g_2(n)$

+

Bunlar taraf tarafa toplanırsa:

$t_1(n) + t_2(n) \leq c_1 \cdot g_1(n) + c_2 \cdot g_2(n)$

$\leq c_1 \cdot \max(g_1(n), g_2(n)) + c_2 \cdot \max(g_1(n), g_2(n))$

$t_1(n) + t_2(n) \leq (c_1 + c_2) \cdot \max(g_1(n), g_2(n))$

$n_0$ve n_1'den büyük olan tercih edilirse ve c=c1+c2 alınırsa $t_1(n)+t_2(n) \leq c.\ \max(g_1(n), g_2(n)$olur.
O halde $t_1(n)+t_2(n) \in O(\max(g_1(n), g_2(n))$ olur.

Eng: Abdulrahman Hamdi

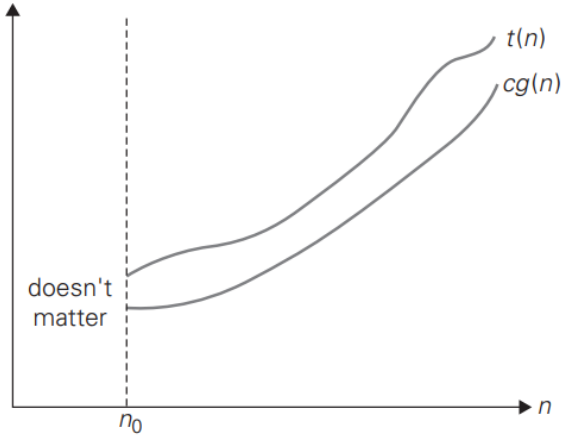
Asimptotik Analiz

Teorem (sabitin önemsizliği):
t(n) ϵ O(g(n)) olsun. Bu durumda her k reel sayısı için k.t(n) ϵ O(g(n)) olur.
Yani bir fonksiyonu sabit bir değerle çarpmak onun büyüme oranını değiştirmez. Daha önce en fazla g(n) kadar büyüyen bir fonksiyon, bir katsayı ile çarpılırsa yine en fazla g(n) kadar büyür.

Kanıt:
t(n) ϵ O(g(n)) ise bir n_0>0 ve c>0 vardır. Öyle ki ∀ n≥n_0 için t(n) ≤c.g(n)
- K negatif ise k.t(n) ≤t(n) ≤c.g(n) olacağından k.t(n) ϵ O(g(n)) olur.
- K pozitif olsun. Bu durumda yukarıdaki eşitsizlik k ile çarpıldığında yönü değişmez. k.t(n) ≤k.c.g(n) olup k.c yeni c değerimiz olur. Buna c diyelim. O halde k.t(n) ≤c.g(n) olur.
- Bu yüzden k.t(n) ϵ O(g(n)) olur.

Asimptotik Analiz

• **Big Omega Notasyonu:**
t(n) ϵ O(g(n))'de g(n), t(n) için bir üst sınırsa,
t(n) ϵ Ω(g(n))'de g(n), t(n) için bir alt sınırdır diyeceğiz. Yani O ve Ω ifadeleri birbirine zıttır.
t(n) ϵ Ω(g(n)) demek t(n) en az g(n) kadar büyük (daha da gidebilir).
Formal olarak:
t(n) ϵ Ω(g(n)) ise bir c>0 ve bir n_0>0 vardır öyle ki bu n0'dan sonraki her n için t(n) ≥c.g(n) olur.
Bir noktadan (n_0)'dan sonra g(n), t(n)'i geçiyor.



Big-omega notation: $t(n) \in \Omega(g(n))$.

n'in n_0 değerini alana kadar ne olduğu ile yani t(n) mi büyük yoksa c.g(n) mi büyük ilgilenmiyoruz.

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Teorem :
t(n) ∈ O(g(n)) ⇔ g(n) ∈ Ω(t(n))
Bu iki ifade birbirine eşittir.
Bir fonksiyon (g(n)) bir fonksiyon (t(n)) için bir üst sınırsa o fonksiyon (t(n)) de o fonksiyona (g(n)) bir alt sınırdır.

Kanıt:
⇒) t(n) ∈ O(g(n)) olsun. Bu durumda bir n_0>0 ve bir c>0 için ve her n≥n_0 için t(n) ≤c.g(n) olur

Her iki taraf pozitif c'ye bölünürse g(n) ≥ $\frac{1}{c}$.t(n) olur.
Bu da g(n) ∈ Ω(t(n)) demektir.
⇐g(n) ∈ Ω(t(n)) olsun. Bu durumda bir n_0>0'dan büyük her n ve bir c>0 için

g(n) ≥c.t(n) olur. Her iki taraf c'ye bölünürse $\frac{1}{c}$. g(n) ≥t(n) olur.

(yani t(n) ≤ $\frac{1}{c}$. g(n)) Bu durumda t(n) ∈ O(g(n)) olur.

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Soru : t(n)= $\frac{n^2}{2} - 1'in\ \Omega(n^2)$ 'nin bir elemanı olduğunu gösterelim.

Çözüm: Bir n_0>0'dan sonraki n değerleri için

$\frac{n^2}{2} - 1 \geq c.n^2$

Olacak şekilde bir c arıyoruz. C= $\frac{1}{4}$ alırsak

(isterseniz $\frac{1}{5}, \frac{1}{6}$,..... alın yeter ki $\frac{1}{2}$ den küçük olsun.)

2'den büyük her n değeri için (bu durumda n_0=2 oluyor)

$\frac{n^2}{2} - 1 \geq \frac{n^2}{4}$ sağlanır

O halde bir n_0=2 ve bir formül değeri bulup, bu değerler için t(n)'i g(n)'den büyük kılabildiğimizden

t(n)= $\frac{n^2}{2} - 1 \in \Omega(n^2)$ diyebiliriz.

Özet: Ω(g(n))

– Bir g(n) fonksiyonu ile aynı veya daha büyük büyüme derecesine sahip fonksiyonların tümü

• Bir sabit katsayı ve n değeri sonsuza giderken

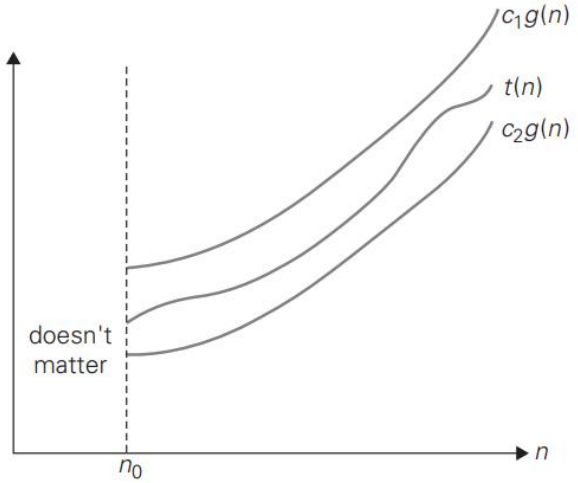
$$n^3 \in \Omega(n^2), \qquad \frac{1}{2}n(n-1) \in \Omega(n^2), \qquad \text{but } 100n + 5 \notin \Omega(n^2).$$

Eng: Abdulrahman Hamdi

Asimptotik Analiz

• **Big Theta Notasyonu:**
Bir g(n) fonksiyonu bir t(n) fonksiyonu için hem bir üst sınır hem de bir alt sınır ise t(n) ϵ Θg(n) deriz.
Bir n_0 >0 ve bir c_1 >0 ve c_2>0 için her n≥ n_0 olduğunda formül oluyorsa c_1.g(n) ≤t(n) ≤ c_2.g(n) oluyorsa t(n) ϵ Θg(n) olur.

c_2g(n) ≤ t (n) ≤ c_1 g(n), n ≥ n_0 ise

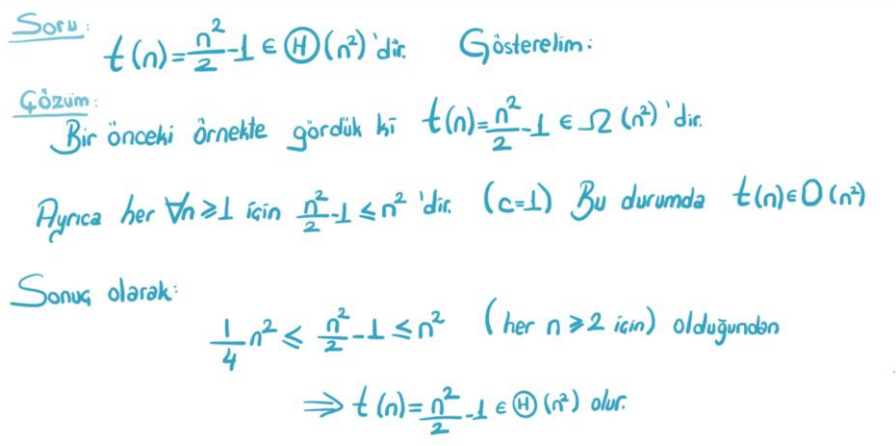


Big-theta notation: $t(n) \in \Theta(g(n))$.

t(n) ϵ Ω(g(n)) ise t(n) ϵ O(g(n)) ve t(n) ϵ Ω(g(n))
Kırmızı rengi g(n), t(n) için bir üst sınır.
Yeşil rengi g(n), t(n) için bir alt sınır.

Eng: Abdulrahman Hamdi

Asimptotik Analiz



Özet: Θ(g(n))

– Bir g(n) fonksiyonu ile aynı büyüme derecesine sahip fonksiyonların tümü

• Bir sabit katsayı ve n değeri sonsuza giderken

– $an^2 + bn + c$, $a > 0$ ise $\Theta(n^2)$ içerisindedir

$$\frac{1}{2}n(n-1) \in \Theta(n^2)$$

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \leq \frac{1}{2}n^2 \quad \text{for all } n \geq 0.$$

$$\frac{1}{2}n(n-1) = \frac{1}{2}n^2 - \frac{1}{2}n \geq \frac{1}{2}n^2 - \frac{1}{2}n\frac{1}{2}n \text{ (for all } n \geq 2) = \frac{1}{4}n^2.$$

$$c_2 = \frac{1}{4},\ c_1 = \frac{1}{2},\ n_0 = 2.$$

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Asimptotik Büyüme Derecesi İçin Bazı Özellikler

- $f(n) \in O(f(n))$
- $f(n) \in O(g(n))$, eğer $g(n) \in \Omega(f(n))$
- Eğer $f(n) \in O(g(n))$ ve $g(n) \in O(h(n))$, ise
  - $f(n) \in O(h(n))$
- If $f_1(n) \in O(g_1(n))$ ve $f_2(n) \in O(g_2(n))$, ise
  - $f_1(n) + f_2(n) \in O(\max\{g_1(n), g_2(n)\})$

Büyüme Derecelerinin Karşılaştırılması

• O, Ω, ve Θ yaklaşımları tanımlarının birbirinden bağımsız olması sebebiyle iki fonksiyonun
büyüme derecelerinin karşılaştırılmasında pek kullanılmaz

• Bu işlem için iki fonksiyonlarının oranının sonsuza giderken limit yaklaşımı daha uygundur

Asimptotik Analiz

Fonksiyonların Büyüme Oranlarının Kıyaslanması

Diyelim ki $g_1(n) \geq g_2(n)$ olsun (bir n_0'dan sonra).
Bir $t_1(n)$ ve $t_2(n)$ fonksiyonları için eğer
$t_1(n) \in O(g_1(n))$ ve $t_2(n) \in O(g_2(n))$
ise $t_1(n)$ fonksiyonu $t_2(n)$'den her zaman daha büyüktür diyemeyiz

$t_1(n)=n \in O(n^3)$ ve $t_2(n)=n2 \in O(n^2)$ ifadeleri doğrudur ama $t_1(n) \geq t_2(n)$ diyemeyiz. t_1, t_2'den daha hızlı büyür diyemeyiz.
Üst sınırlar üzerinden kıyaslama olmaz.

Limit Yöntemi İle Karşılaştırma

Diyelim ki Ahmet'in en fazla 100bin TL'si, Mehmet'in en fazla 10bin TL'si var. Bu durumda Ahmet'in Mehmet'ten daha fazla parası olmak zorunda değildir. Çünkü belki Ahmet'in bin TL'si, Mehmet'in 9bin TL'si vardır. Bu yüzden O kullanarak bir fonksiyonu başka bir fonksiyonla büyüme oranı açısından kıyaslayamayız.
Bunun yerine büyüme oranlarının kıyaslanmasını yaparken "limit" kavramından faydalanacağız

$$\lim_{n\to\infty} \frac{t(n)}{g(n)} = \begin{cases} 0 & t(n) \text{ büyüme derecesi} < g(n) \text{ büyüme derecesi} \\ c > 0 & t(n) \text{ büyüme derecesi} = g(n) \text{ büyüme derecesi} \\ \infty & t(n) \text{ büyüme derecesi} > g(n) \text{ büyüme derecesi} \end{cases}$$

- İlk iki durum : $t(n) \in O(g(n))$,
- *İkinci durum :* $t(n) \in \Theta(g(n))$
- *Son iki durum :* $t(n) \in \Omega(g(n))$,

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Limit Yöntemi İle Karşılaştırma

L'Hôpital Kuralı:

Eğer $\lim_{n\to\infty} f(n) = \lim_{n\to\infty} g(n) = \infty$ ve f',g' türevleri varsa

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

Çok büyük n değerleri için Stirling Formülü

$$n! \approx \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$$

Örnek:

$\frac{1}{2}.n(n-1)$ ile n^2fonksiyonlarının büyüme derecelerini karşılaştırınız

$$\lim_{n\to\infty} \frac{\frac{1}{2}n(n-1)}{n^2} = \frac{1}{2}\lim_{n\to\infty} \frac{n^2-n}{n^2} = \frac{1}{2}\lim_{n\to\infty}(1-\frac{1}{n}) = \frac{1}{2}.$$

Sonuç sabit olduğu için eşit büyüme derecelerine sahipler

$$\frac{1}{2}n(n-1) \in \Theta(n^2)$$

Eng: Abdulrahman Hamdi

Asimptotik Analiz

Örnek:

$\log_2 n$ ve $\sqrt{n}$ fonksiyonlarının büyüme derecelerini karşılaştırınız.

$$\lim_{n\to\infty} \frac{\log_2 n}{\sqrt{n}} = \lim_{n\to\infty} \frac{(\log_2 n)'}{(\sqrt{n})'} = \lim_{n\to\infty} \frac{(\log_2 e)\,\frac{1}{n}}{\frac{1}{2\sqrt{n}}} = 2\log_2 e \lim_{n\to\infty} \frac{1}{\sqrt{n}} = 0.$$

$\log_2 n$ büyüme derecesi daha küçüktür

O halde √n fonksiyonu, $\log_2$n fonksiyonundan daha büyük diyebiliriz.

Başka bir deyişle zaman verimliliği $\log_2$n olan bir algoritmayı zaman verimliliği √n olan bir algoritmaya tercih ederiz

Örnek:

n! ve n^2 fonksiyonlarının büyüme derecelerini karşılaştırınız

$$\lim_{n\to\infty} \frac{n!}{2^n} = \lim_{n\to\infty} \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{2^n} = \lim_{n\to\infty} \sqrt{2\pi n}\frac{n^n}{2^n e^n} = \lim_{n\to\infty} \sqrt{2\pi n}\left(\frac{n}{2e}\right)^n = \infty.$$

n! Daha hızlı büyümektedir.

n! ∈ $\Omega(2^n)$

Soru: $t_1(n)=3n^3-4n^2+5$ fonksiyonu mu yoksa $t_2(n)=2n^2-4$ fonksiyonu mu daha hızlı büyür?

Çözüm:

$$\lim_{n\to\infty} \frac{3n^3-4n^2+5}{2n^2-4} = \lim_{n\to\infty} \frac{9n^2-8n}{8n} = \lim_{n\to\infty} \frac{18n}{8} = \frac{18}{8} \lim_{n\to\infty} n$$

$= \infty$ olup $t_1(n)$ fonksiyonu $t_2(n)$'e göre daha hızlı büyür.

Eng: Abdulrahman Hamdi

Algorithm Analysis And Design

Thanks for listening!

Eng: Abdulrahman Hamdi