Ayrık Matematik
(Ayrık İşlemsel Yapılar)

Fırat İsmailoğlu, PhD

Hafta 6:
Algoritma Analizi
(Algoritmik Karmaşıklık)

Hafta 6
Plan

1. Asimptotlar
2. Buyuk O Notasyonu
3. Buyuk Omege Notasyonu
4. Buyuk Theta Notasyonu

Giriş

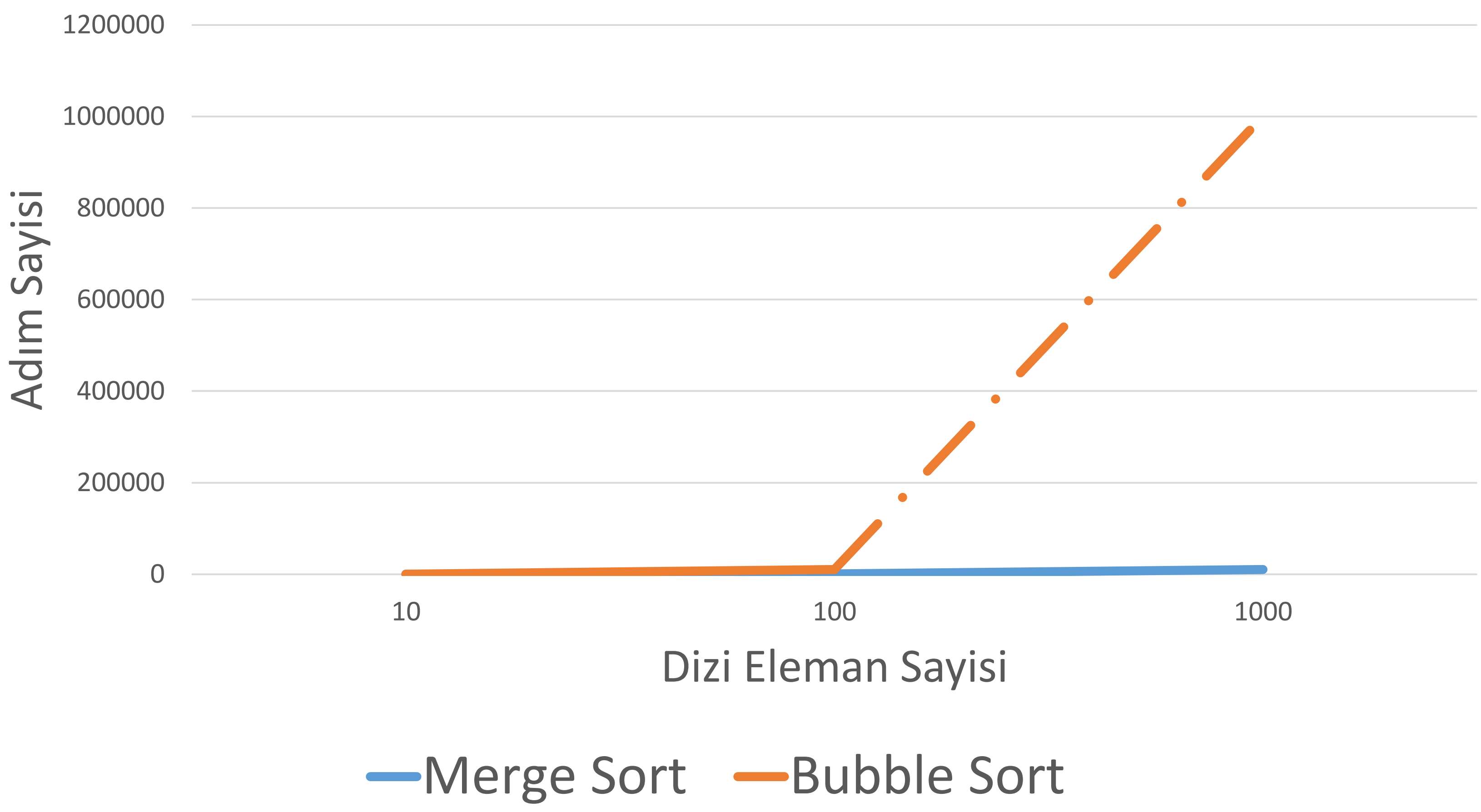
Bu derste algoritmalarin hizlarini kiyaslayacagiz. Bunun icin bir matematik konusu olan asimptotlardan yararlanacagiz.

Bir problemi çözebilmek kadar o problemi hizli cozebilmek de cok onemlidir. Hizli bir sekilde web aramasi yapabilmek, hizli bir sekilde en kisa yolu bulabilmek, hizli bir sekilde siralama yapabilmek onemlidir. Bunun icin daha hizli algoritmalara ihtiyac duyariz.

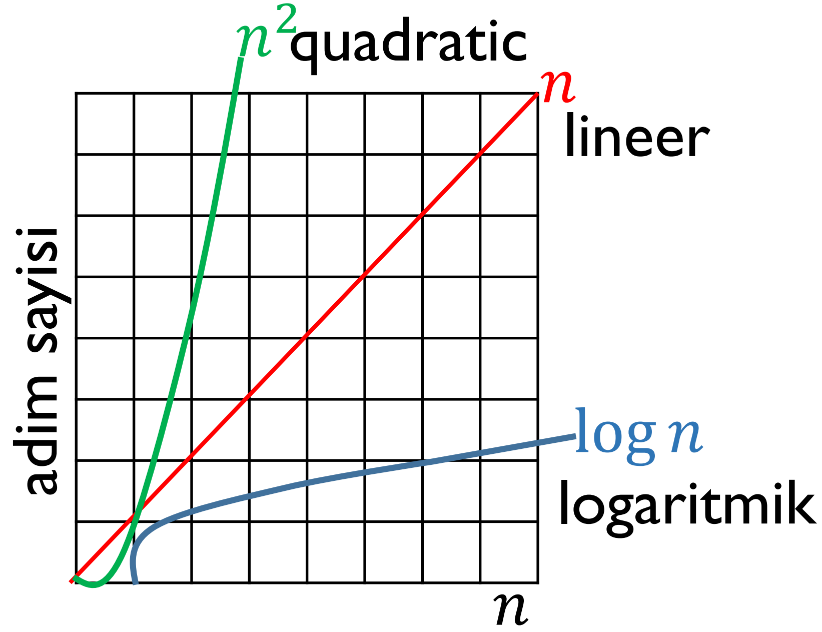
Algoritmalarin hizini ise *asimptotik analiz* ile tahmin ederiz. Burada ana fikir, bir algoritmada girdi buyuklugunu (input size) artirdigimizda algoritmanin ne kadar yavaşlayacagina bakmaktir. Yavaşlamayi ölçmek için ise algoritmanin görevini tamamlamak için ihtiyac duyacagi toplam adim sayisi göz önüne alinir.

Not: İki algoritma karsilastirilirken buyuk girdilerdeki (big inputs) perfomanslarina dikkat edilir. Bunu algoritmalarin uzun dönem davranişlarini inceleme olarak da görebiliriz.

ör. Aşagida, siralama algoritmalari olan merge sort ve bubble sort icin girdi buyuklugunu yani dizi (array) eleman sayisini artirdigimizda verilein diziyi siralamak için gereken operasyon (adim) sayisi verilmistir.



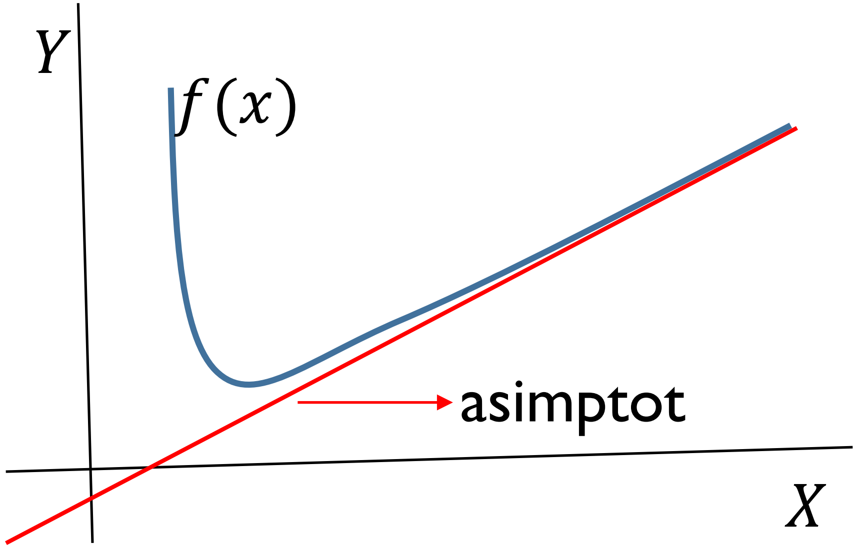
ör. Diyelimki üç tane siralama algoritmamiz var. n uzunlugundaki bir diziyi birinci algoritma $\log n$, ikinci algoritma n, ucuncu algoritma da n^2 adimda siraliyor. Bu uc siralama algoritmasindan hangisi en idealdir?

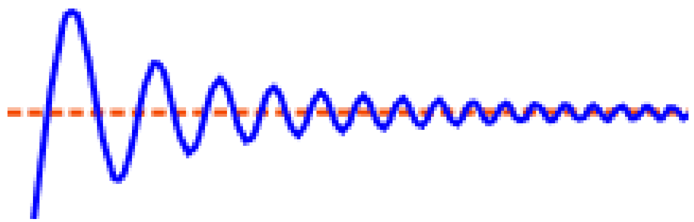


Buradan açikca goruluyorki algoritmalardan en ideali $\log n$ buyume oranina sahip olandir. Cok yuksek n degerleri icin dahi (cok uzun dizilerde dahi) siralamayi tamamlamak icin gereken adim sayisi dusuktur; bu da algoritmanin hizli olmasi anlamina gelir.

Asimptotlar (Sonuşmaz)

Asimptot bir eğrinin yaklaştigi dogrudur. Eğri ile asimptot arasindaki fark sonsuzda 0 olur.





Bir $f(x)$ fonksiyonun asimptotik davranışı x çok büyüdükçe $f(x)$'in hangi dogruya (hangi asimptota) yaklaştiği ile alakalidir. Böylece asimptotik analiz ile fonksiyonun çok buyuk girdilerde ne gibi değerler alacagini öngörmüş oluruz.

Algoritma analizinde de zaten amacimiz algoritma girdisi çok buyudukce algoritmanin ne kadar hizli calisacagini gormekti. Algoritmanin calisma zamanini algoritma girdisinin bir fonksiyonu olarak dusunursek asimptotlari algoritmalari kiyaslarken kullanabiliriz.

Asimptotik Notasyon

Bir fonksiyonun nasil buyudugunu (buyume davranisini) tanimlarken bazi kavramlardan faydalanacagiz. Bunlar büyük O, büyük Ω ve büyük Θ kavramlaridir.

1. Büyük O (Big O) ($\leq$) (Oh) (Üst Sınır)

Bir f fonksiyonu bir g fonksiyonundan daha hizli büyümez (g, f icin bir ust sinirdir)

($f \leq g$) seklindeki durumlari büyük O notasyonunu kullanarak ifade ederiz.

Bu durumu $f(n) = O\big(g(n)\big)$ ile gosteririz.

Formal olarak,

Eğer $f(n) = O\big(g(n)\big)$ ise $c > 0$ ve $n_0 \geq 0$ gibi iki sabit (değişmez) vardir oyleki
$\forall\, n \geq n_0$: $f(n) \leq c \cdot g(n)$.

Son satiri su sekilde yorumlayabiliriz: Bir n_0 noktasindan sonraki her nokta icin (yani f ve g yeterince buyudugunde) f fonksiyonu $\text{sabit} \times g$ fonksiyonundan kucuk kalacaktir.

Not: Burada f ve g fonksiyonunu $f\colon \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ ve $g\colon \mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$ olarak dusunuyoruz. Yani kartezyen duzlemin sag ust kosesinde tanimlidirlar. Bunun nedeni, bir algoritmanin buyume hizini olcerken fonksiyondaki X ekseni girdi buyuklugunu, Y ekseni adim sayisini (bazen zamani) gosterecektir, ve bu degerler her zaman pozitif kabul edileceklerdir.

ör. $f_1(n) = n,\ f_2(n) = 2n,\ \ f_3(n) = 10000n,\ \ f_4(n) = 6,\ \ f_5(n) = n + 8$ fonksiyonlari $O(n)$'e eşittir. Çünkü bu fonksiyonlarin her biri icin bir $c > 0$ ve bir $n_0 > 0$ sabitleri bulunur oyleki $\forall\, n \geq n_0\colon\ f(n) \leq c \cdot n.$

Örnegin $f_3(n) = 10000n$ için c sabitini 10001 alirsak $10000n \leq 10001 \cdot n$ olur.

ör. $f(n) = 3n^2 + 4n - 2$ fonksiyonu $O(n^2)$ dir.

Bunu gösterebilmek icin $c > 0$ ve $n_0 \geq 0$ sabitleri bulacagiz oyleki

$\forall\, n \geq n_0\colon\ 3n^2 + 4n - 2 \leq c \cdot n^2$

$n \geq 1$ için $3n^2 + 4n - 2 \leq 3n^2 + 4n^2 + 2n^2 = 7 \cdot n^2.$

Yani $n_0 = 1$ ve $c = 7$ secilerek $f(n) = O(n^2)$ saglanir.

Not: $f(n) = O(g(n))$ iken $g(n)$, $f(n)$için (herhangi) bir üst sınırdır ve tek değildir. Yani bir f fonksiyonu için birçok -hatta sonsuz- ust sınır bulunabilir.

Örnegin $f(n) = 4n$ fonksiyonu $O(n)$ dir, ayni zamanda $O(n^2)$'dir, $O(n^3)$'dür...

($\exists\, c > 0$ ve $\exists\, n_0 \geq 0 : \forall\, n \geq n_0: 4n \leq c \cdot n^2$)

Fakat genel yaklasimimiz ust sinirlarin en kucugunu almaktir. $f(n) = 4n$ fonksiyonu $O(n)$ dir diyecegiz.

ör. $f(n) = n^3$ fonksiyonunun $O(n^2)$ *olmadigini* gosteriniz.

$f(n)$ fonksiyonu $O(n^2)$ olsaydi:

$\exists\, c > 0$, $\exists\, n_0 \geq 0 : \forall\, n \geq n_0: n^3 \leq c \cdot n^2$ olurdu.

$f(n)$'nin $O(n^2)$ olmadigini gostermek icin yukardaki ifadenin tersinin dogru oldugunu gosterecegiz.

Bu ifadenin tersini De Morgan kurali ile bulacagiz:

$\forall\, c > 0$,$\forall\, n_0 \geq 0 : \exists\, n \geq n_0: n^3 > c \cdot n^2$

(Dikkat tersini alirken $\forall$ (her) vardir ($\exists$); vardir her oldu!)

$\forall\, c > 0\ ,\forall\, n_0 \geq 0 : \exists\, n \geq n_0 :\ n^3 > c \cdot n^2$

matematiksel ifadesinin okunuşu:

c'nin ve n_0'in her pozitif degeri icin n_0'dan buyuk oyle bir n degeri vardirki $n^3 > c \cdot n^2$ dir.

Eğer n'yi $n = \max(n_0, c+1)$ seklinde secersek $n > c$ olur (neden?)

$n > c$ esitsizliginde her iki tarafi positif n^2 ile carparsak aradigimiz $n^3 > c \cdot n^2$ esitsizligini elde ederiz.

(Not burada kanit yaparken c ve n_0 degerleri icin herhangi bir varsayimda bulunmadik, jenerik dusunduk. Buda buldugumuz esitsiziligin her c ve her n_0 icin gecerli oldugu anlamina gelirki zaten bunu kanitlamak istiyorduk)

Polinomlar

Bir (yada daha fazla degiskenin) katsayilarla carpilmis kuvvetlerinin toplami olan matematiksel ifadedir. Bir polinomun genel formu su sekildedir:

$$a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0 = \sum_{i=0}^{n} a_i x^i \quad (a_i \in \mathbb{R}, i = 1, \dots, n)$$

Polinomun Derecesi: Bir polinomun derecesi, o polinomdaki degiskenin en yuksek kuvvetidir.

ör. $p(x) = -x^6 + 4x^2 - 2$ polinomunun derecesi 6'dir.

Polinomlarin büyümelerinin en belirgin ozelligi, büyemelerinin dereceleri tarafindan karar verilmesidir. Yani $a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ polinomu x^n asimptotik olarak gibi davranir.

Teorem: $p(x) = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$ polinomu $O(x^n)$ dir.

Sabit-Logaritmik – Polinom – Eksponansiyel-Faktoriyel Fonksiyonlarinin Buyume Davranislar

en yavaş

sabit fonksiyon ← K

logaritmik ← $\log n$

lineer ← n

$n \cdot \log n$

polinom/üssel ← n^2

n'in daha büyük kuvvetleri

eksponensiyel ← 2^n

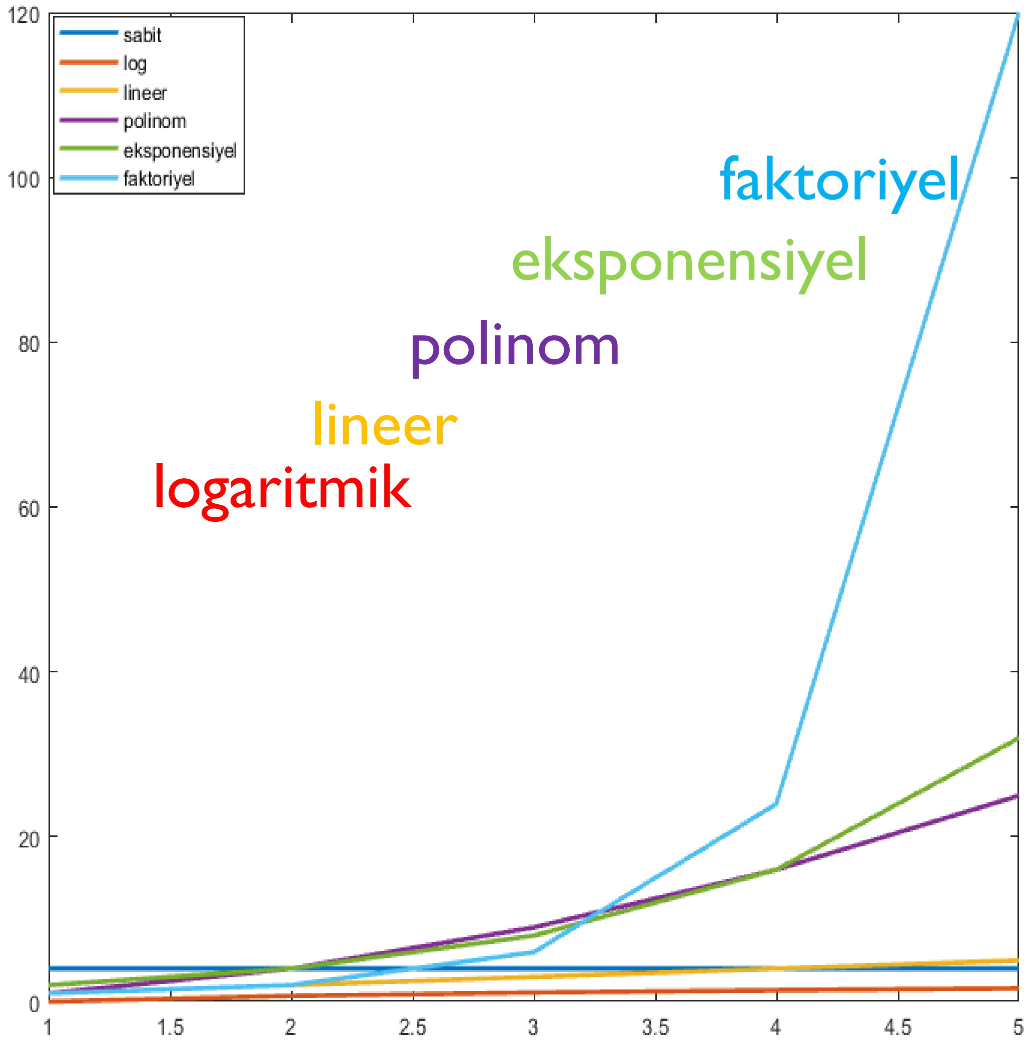
3^n

kuvveti n olan daha büyük sabitler

faktoriyel ← $n!$

n^n

en hızlı



Büyük O nun Bazı Özellikleri

1. Toplam Kurali

Eger $f_1(n) = O(g_1(n))$ ve $f_2(n) = O(g_2(n))$ ise $f_1(n) + f_2(n) = O\left(max(g_1(n), g_2(n))\right)$

Toplamlar en hizli terim kadar hizli buyurler. Toplamdaki en hizli terim toplamin ne kadar hizli olacagini belirler.

ör. $f_1(n) = 3n^2$ ve $f_2(n) = +4n^6$ olsun. Bu halde $f_1(n) = O(n^2)$ ve $f_2(n) = O(n^6)$ dir.

Kurala gore $f_1(n) + f_2(n) = 3n^2 + 4n^6$ fonksiyonu $O(max(n^6, n^2)) = O(n^6)$.

(Buldugumuz sonuc polinomun buyume oranini polinomun derecesi belirler savini destekler)

Kanit:

$f_1(n) = O(g_1(n))$ ise $\exists\, c_1 > 0, \exists\, n_1 \geq 0 : \forall\, n \geq n_1 : f_1(n) \leq c_1 \cdot g_1(n)$

$f_2(n) = O(g_2(n))$ ise $\exists\, c_2 > 0, \exists\, n_2 \geq 0 : \forall\, n \geq n_1 : f_2(n) \leq c_2 \cdot g_2(n)$

Su halde

$f_1(n) + f_2(n) \leq c_1 \cdot g_1(n) + c_2 \cdot g_2(n)$

$a, b \in \mathbb{R}$ icin $a \leq max(a, b)$ ve $b \leq max(a, b)$

Yukaridaki bilgiyi kullanirsak

$f_1(n) + f_2(n) \leq c_1 \cdot g_1(n) + c_2 \cdot g_2(n) \leq max(c_1, c_2) \cdot g_1(n) + max(c_1, c_2) \cdot g_2(n) \leq$

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

$f_1(n) + f_2(n) \leq 2max(c_1, c_2) \cdot max\big(g_1(n), g_2(n)\big)$

$2max(c_1, c_2)$'ye c' gibi yeni bir sabit dersek, sonuc olarak:

$f_1(n) + f_2(n) \leq c' \cdot max\big(g_1(n), g_2(n)\big)$ olur.

Buradan $f_1(n) + f_2(n) = O\Big(max\big(g_1(n), g_2(n)\big)\Big)$ olur. □

2. Pozitif Sabitin Önemsizligi

$f(n) = O\big(g(n)\big)$ ise her $k > 0$ sabiti icin $k \cdot f(n) = O\big(g(n)\big)$.

Yani bir fonksiyonun buyume oranini incelerken sabitlerin bir onemi yoktur, dikkate almayiz.

Kanit:

$f(n) = O(g(n))$ ise $\exists\, c > 0$, $\exists\, n_0 \geq 0$: $\forall\, n \geq n_0$: $f(n) \leq c \cdot g(n)$

Esitsizligin her iki tarafini pozitif k ile carparsak

$$k \cdot f(n) \leq k \cdot c \cdot g(n)$$

$k \cdot c > 0$ sabitine c' gibi yeni bir sabit dersek

$k \cdot f(n) \leq c' \cdot g(n)$ olur. Buradan $k \cdot f(n) = O(g(n))$ olur.

ör. $50n = O(n), 500000000n = O(n), 0.000000005n = O(n)$

Teorem: Logaritmik fonksiyonun tabani fonksiyonun buyume oranina etki etmez (yani logaritma tabaninin asimtotik olarak bir onemi yoktur). (tum logaritmik fonksiyonlar ayni oranda büyürler). Her $t > 1$ tabani icin $f(x) = log_t x$ fonksiyonu $O(logx)$'dir.

Kanit: $f(x) = log_t x = \frac{logx}{logt}$ seklinde yazabiliriz. Su halde $f(x) = \frac{1}{logt} \cdot logx$ olur.

$\frac{1}{logt}$ 'den buyuk bir $c > 0$ sabiti bulunabilir; ornegin $c = 2$.

$f(x) = \frac{1}{logt} \cdot logx \leq c \cdot logx$ olup $log_t x = O(logx)$ olur.

3. Geçişkenlik

Eğer $f(n) = O(g(n))$ ve $g(n) = O(h(n))$ ise $f(n) = O(h(n))$ olur.

Eger f fonksiyonu en fazla g fonksiyonu kadar hizli buyurse; g fonksiyonuda en fazla h fonksiyonu kadar hizli buyurse f fonksiyonu en fazla h fonksiyonu kadar hizli buyur diyebiliriz.

Kanit:

$f(n) = O(g(n))$ ise bir $c_1 > 0$ icin $f(n) \leq c_1 \cdot g(n)$

$g(n) = O(h(n))$ ise bir $c_2 > 0$ icin $g(n) \leq c_2 \cdot h(n)$

Su halde $f(n) \leq c_1 \cdot g(n) \leq c_1 \cdot c_2 \cdot h(n)$ olur. Boylece $f(n) = O(h(n))$ olur.

2. Büyük Ω ($\geq$) (Omega)(Alt Sınır)

Büyük Ω notasyonu, büyük O notasyonunun zıttıdir ve alt sınırı gösterir.

$f(n) = \Omega(g(n))$ durumunda, f fonksiyonu g fonksiyonundan daha hizli büyür.

(f'nin buyume orani g'nin buyume oranindan büyüktür)

Formal gösterim:

Eğer $f(n) = \Omega\big(g(n)\big)$ ise $d > 0$ ve $n_0 \geq 0$ gibi iki sabit (değişmez) vardir oyleki
$\forall\, n \geq n_0$: $f(n) \geq d \cdot g(n)$.

Teorem: $f(n) = O\big(g(n)\big) \Leftrightarrow g(n) = \Omega\big(f(n)\big)$

Kanit: $\Rightarrow$: $f(n) = O\big(g(n)\big) \Rightarrow \exists\, c > 0\,, \exists\, n_0 \geq 0 : \forall\, n \geq n_0 : f(n) \leq c \cdot g(n)$

$$\Rightarrow g(n) \geq \frac{1}{c} \cdot f(n) \Rightarrow g(n) = \Omega\big(f(n)\big)$$

$\Leftarrow$: $g(n) = \Omega\big(f(n)\big) \Rightarrow \exists\, d > 0\,, \exists\, n_0 \geq 0 : \forall\, n \geq n_0 : g(n) \geq d \cdot f(n)$

$$\Rightarrow f(n) \leq \frac{1}{d} \cdot g(n) \;\Rightarrow f(n) = O\big(g(n)\big)$$

3. Büyük (Θ) (Theta)(Eşitlik)

Eğer bir f fonskiyonu bir g fonskiyonu ile ayni oranda buyurse bunu $f(n) = \Theta(g(n))$ ile gösteririz. Bu durumda f'in buyume orani g'nin buyume oranina esittir deriz.

$f(n) = \Theta(g(n))$ durumunda $f(n) = O(g(n))$ ve $f(n) = \Omega(g(n))$ olur.

$$\boxed{f(n) = \Theta(g(n)) \Leftrightarrow f(n) = O(g(n)) \text{ ve } f(n) = \Omega(g(n))}$$

Not: Yukarida gösterilen özellik, reel sayilardan bildigimiz bir ozelligin fonksiyon buyumelerine yansimasidir:

$a, b \in \mathbb{R}$ için

$a \leq b$ ve $b \leq a \Rightarrow a = b$

ör. $f(n) = n^2$ fonskiyonu $\Theta(n^2)$ dir. Çunku bu fonksiyon hem $O(n^2)$ dir hemde $\Omega(n^2)$ dir.

Büyük O Notasyonun Yanlış Kullanimi

Bu cümle anlamsizdir:

Ali, Mehmet'ten daha zengindir; çünkü Ali'nin en fazla 1 milyonu var, Mehmet'in en fazla 100 lirasi var.

Belki Ali'nin 50 lirasi var (hatta hic yok) ama Mehmet'in 90 lirasi var?

Ayni sekilde:

f fonksiyonunu g ye tercih ederim; çunku $f(n) = O(n^2)$ dir ve $g(n) = O(n^3)$ dir.

çikarimi da yanlistir. Çunku belki $f(n) = n^2$ dir ve $g(n) = n$ dir.

Burada hatirlamaiz gereken şey <u>O'nun bir üst sınır olduğudur</u>. Ve üst sınır ile kıyaslama yapamayiz!!

Eger bir kiyaslama yapacaksak O yerine Θ'yi kullanmaliyiz.

($f(n) = \Theta(n^2)$ ve $g(n) = \Theta(n^3)$ ise f, g'den iyidir diyebiliriz.)

Asimptotik Analizin Algoritmalarda Kullanimi

En başta da belirttigimiz gibi biz daha hizli algoritmalarla ilgileniyoruz. Algoritmanin görevini tamamlamak için ihtiyac duyacagi toplam adim sayisi bize algoritmanin hizi hakkinda bilgi verir. Bu adim sayisi ne kadar fazla ise algoritma o kadar yavastır.

Not: Programin basladigi zaman ile bittigi zaman arasindaki fark kullandigimiz algoritmanin hizi ile ilgili saglikli bilgi vermez. Gecen zaman kullandigimiz bilgisayara bağlı olur; böylece zamani kullanarak yaptigimiz tahmin bilgisayardan bilgisayara degisir.

ör. Bir A dizisinde bir x degerinin olup olmadigina karar veren şöyle bir algoritmamiz olsun.

```
linearSearch(A,x):
 n=length(A)
 for i=1:n{
     if A[i]=x{
         return True}
 return False}
```

→ Bu algoritma basitce dizinin tum elemanlarini gezer aranan x'i bulmasi durumunda True'ya döner ve program sonlanir; bulamazsa False'a döner ve program sonlanir.

$f(n)$, `linearSearch` algoritmasinin n uzunlugundaki bir dizinin içinde bir x degerinin olup olmadigina karar vermesi icin takip etmesi gereken adim sayisi olsun. (yani $f(n)$, kisaca algortimanin n buyuklugundeki girdi icin hızı)

En kötu durumda aranan x dizinin icinde degildir, bu halde algoritmadaki `for` dongusu n defa calisir. Algoritmanin tamamladigi adim sayisi en fazla n olabilir. n, $f(n)$ icin bir ust sinirdir. $f(n) = O(n)$ dir ($f(n) \leq n$).

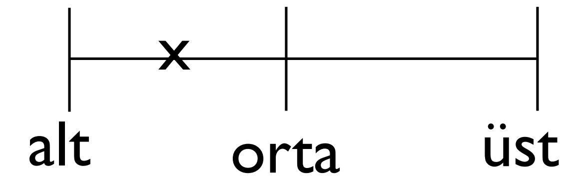
Yine en kötu durumda , algoritma n defa arama yapar; adim sayisi $f(n) = \mathrm{n}$ olur. O halde $f(n) \geq n$ yazabiliriz. Buradan $f(n) = \Omega(n)$ olur.

Sonuc olarak $f(n) = \Theta(n)$ dir.

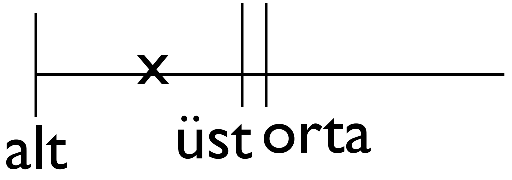
Şimdi daha hizli bir arama algoritmasi uzerine çalisalım.

`binarySearch` sıralanmış bir diziyi alır ve `linearSearch`'e göre daha hizli çalışır.

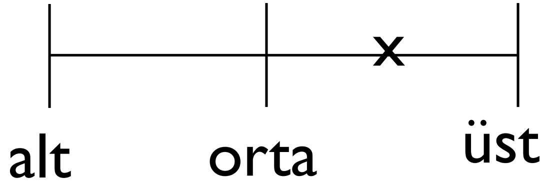
```
binarySearch(A,x)
n=length(A)
alt=1, ust=n
while alt≤ ust{
      orta=⌊(alt+ust)/2⌋ //tam sayi cikmazsa asagi yuvarliyoruz.
      if A[orta]=x{
         return True}
      else if A[orta]>x{  //ust'u asagi cek (kucult)
         ust=orta-1 }
      else // orta deger x'in altindaysa
         alt=orta +1    // alt'i yukari cek (buyult)
return False
```



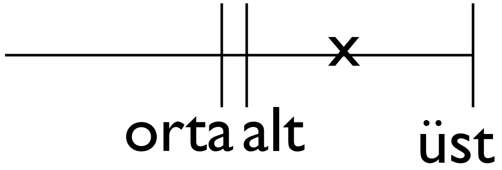
burada A[orta]>x. üst’ü aşagi çekeriz:ortanin bir soluna



üst=orta-1



A[orta]<x. alt’i yukari cekeriz: orta’nin bir sagina



alt=orta+1

while’in 1.iterasyonunda eger x orta eleman degilse dizideki elemanlarin yarisi atilir.

Yani 1.iterasyon sonucunda $\frac{n}{2}$ tane eleman kalir.

2.iterasyon sonucunda $\frac{n}{2^2}$ tane eleman kalir. ($\frac{n}{2}$’nin yarisi)

3.iterasyon sonucunda $\frac{n}{2^3}$ tane eleman kalir.

Yarilanmalar en fazla 1 tane eleman kalincaya kadar devam eder (en kötü durum).

i iterasyon sonunda 1 tane eleman kalsin: $\frac{n}{2^i} = 1$ ise $2^i = n$. Buradan $i = log_2 n$ olur.

Sonuc olarak while dongusu en fazla $log_2 n$ defa calisir. Şu halde $f(n) = O(log_2 n)$ olur.