2.Hafta
Asimptotik Analiz (Notasyonlar)

O, Ω, ve Θ Notasyonu

Algoritmanın Büyüme Oranları

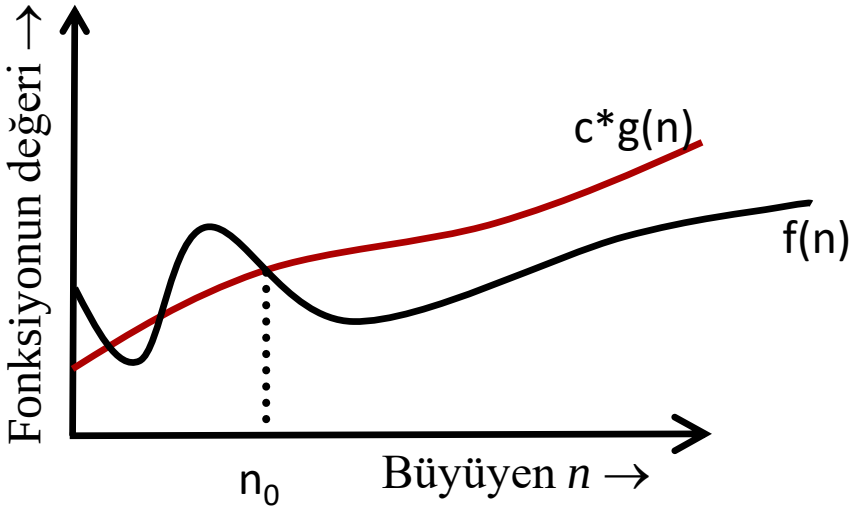
- Bir algoritmanın orantılı zaman gereksinimi **büyüme oranı (veya büyüme hızı)** olarak bilinir.
  - **T(n)** nin büyüme oranı, algoritmanın **hesaplama karmaşıklığı**dır.
- **Hesaplama karmaşıklığı** belirli bir uygulamadan bağımsız olarak **n** ile değişen **T(n)'** nin çalışma zamanını daha doğru bir şekilde bulmayı sağlar.
  - Genel olarak, az sayıda parametreler için karmaşıklıkla ilgilenilmez; eleman sayısı **n**'nin sonsuza gitmesi durumunda T(n) büyümesine bakılır.
  - Karmaşıklığı belirtmek için asimtotik notasyon (simgelem) ifadeleri kullanılmaktadır.

Asimptotik simgelem (notasyon)

O-simgelemi (üst sınırlar)

Tüm $n \geq n\_0$ değerleri için sabitler $c > 0$, $n\_0 > 0$ ise
$0 \leq f(n) \leq cg(n)$ durumunda
$f(n) = O(g(n))$ yazabiliriz.

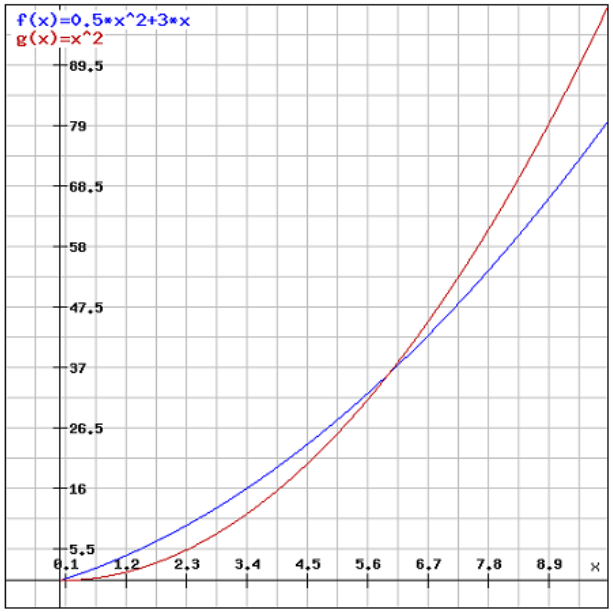
- ***f(n)*** *ve* ***g(n)*** verilen iki çalışma zamanı fonksiyonudur.
- ***f(n)*** $\leq$ ***c.g(n)*** ve ***n*** $\geq$ $\boldsymbol{n_0}$ *koşullarını sağlayan* ***c*** *ve* $\boldsymbol{n_0}$ *değerleri varsa* ***f(n)*** *zaman karmaşıklığı* ***O(g(n))*** *dir.*
- *Başka bir deyişle, n sayısı yeteri kadar büyük olduğunda, f(n), g(n) ile aynı büyüklüktedir.*
- *O-notasyonu* sabit bir katsayı içinde bir fonksiyon için üst sınırı verir



O-simgelemi (üst sınırlar)

- **Örnek:** $2n^2 = O(n^3)$ *için c,* n_0 *değerlerini bulunuz?*
- $0 \leq f(n) \leq c.g(n)$, $0 \leq 2n^2 \leq cn^3$
- $c=1$ için $n_0 = 2$, *şartı sağlar.*
- **Örnek:** $(1/2)n^2 + 3n$ *için üst sınırın* $O(n^2)$ *olduğunu gösteriniz.*
- $c=1$ *için*
- $(1/2)n^2 + 3n \leq n^2$
- $3n \leq 1/2n^2$
- $6 \leq n$,
- $n_0=6$

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



Algoritmanın Büyüme Oranları

- Büyüme oranlarına bakarak iki algoritmanın verimliliğini karşılaştırabiliriz.
  - ***n'*** nin yeterince büyük değerleri için düşük büyüme oranına sahip algoritma her zaman daha hızlıdır.
  - Örneğin; $f(n)=n^2+3n+5$ ifadesinin büyüme oranı $O(n^2)$ dir.
- Algoritma tasarımcılarının amacı, çalışma zaman fonksiyonu olan f(n) nin mümkün olduğu kadar düşük büyüme oranı sahip bir algoritma olmasıdır.

O-notasyonu

- **Örnek**
- $3n^2+2n+5 = 0(n^2)$ olduğunu gösteriniz
- $10\ n^2 = 3n^2 + 2n^2 + 5n^2$
- $10n^2 \geq 3n^2 + 2n + 5\ ,\ n \geq 1$
- $c = 10,\ n_0 = 1$

Ortak Büyüme Oranları

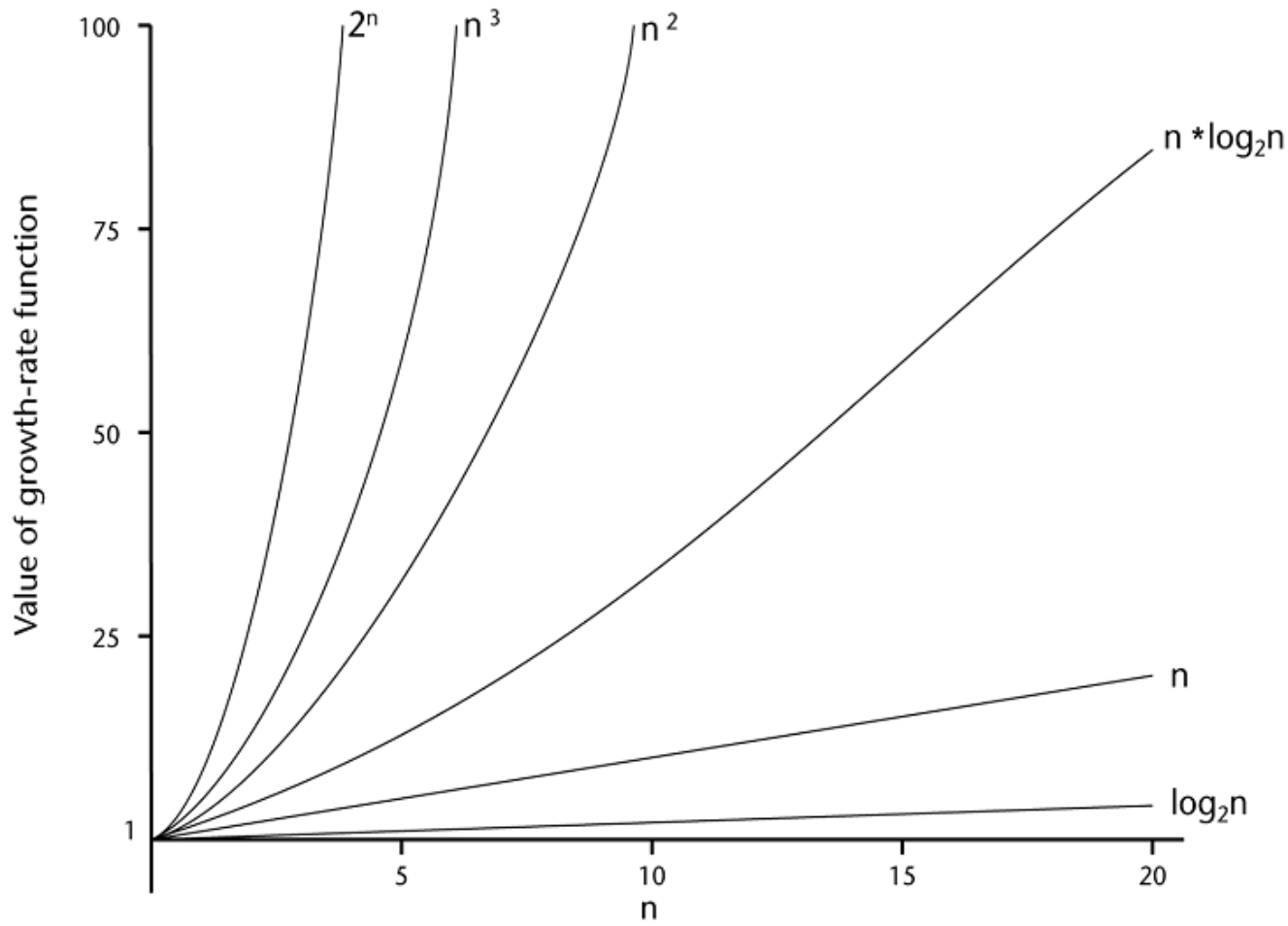
Maliyet artar

Fonksiyon	Büyüme oranı ismi
c	Sabit, komut bir veya birkaç kez çalıştırılır. Yenilmez!
log n	Logaritmik, Büyük bir problem, her bir adımda sabit kesirler tarafından orijinal problemin daha küçük parçalara ayrılması ile çözülür. İyi hazırlanmış arama algoritmalarının tipik zamanı
$\log^2 n$	Karasel logaritmik
n	Doğrusal, Küçük problemlerde her bir eleman için yapılır. Hızlı bir algoritmadır. N tane veriyi girmek için gereken zaman.
n log n	Doğrusal çarpanlı logaritmik. Çoğu sıralama algoritması
n^2	Karasel. Veri miktarı az olduğu zamanlarda uygun (n<1000)
n^3	Kübik. Veri miktarı az olduğu zamanlarda uygun (n<1000)
2^n	İki tabanında üssel. Veri miktarı çok az olduğunda uygun (n<=20)
10^n	On tabanında üssel
n!	Faktöriyel
n^n	n tabanında üstel (çoğu ilginç problem bu kategoride)

Büyüme oranı fonksiyonlarının karşılaştırılması

Function	n = 10	100	1,000	10,000	100,000	1,000,000
1	1	1	1	1	1	1
$\log_2 n$	3	6	9	13	16	19
n	10	10^2	10^3	10^4	10^5	10^6
$n * \log_2 n$	30	664	9,965	10^5	10^6	10^7
n^2	10^2	10^4	10^6	10^8	10^{10}	10^{12}
n^3	10^3	10^6	10^9	10^{12}	10^{15}	10^{18}
2^n	10^3	10^{30}	10^{301}	$10^{3,010}$	$10^{30,103}$	$10^{301,030}$

Büyüme oranı fonksiyonlarının karşılaştırılması



Büyüme oranı fonksiyonlarının karşılaştırılması

- **Verimli algoritmaları geliştirmenin önemi:**

Dizi boyutu	Sıralı arama	İkili (Binary) arama
128	128	8
1,048,576	1,048,576	21

- 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

Büyüme oranı fonksiyonlarının özellikleri

- **1-** Bir algoritmanın büyüme oranı fonksiyonunda düşük dereceli terimler, sabitler ve katsayılar ihmal edilebilir.
  - $O(n^3+4n^2+3n) \rightarrow O(n^3)$
  - $O(8n^4) \rightarrow O(n^4)$

- **2-** Algoritmanın büyüme fonksiyonlarını birleştirebiliriz.
  - $O(f(n))+O(g(n)) = O(f(n)+g(n))$
  - $O(n^3)+O(4n^2) \rightarrow O(n^3+4n^2) \rightarrow O(n^3)$
  - Çarpma içinde benzer kurallara sahiptir.

Büyüme oranı analizi ile ilgili problemler

- **1-** Daha küçük büyüme oranına sahip bir algoritma yeterince büyük olmayan belirli **n** değerleri için daha hızlı büyüme oranına sahip algoritmadan hızlı çalışmaz.

- **2-** Aynı büyüme oranına sahip algoritmalar çalışma zamanı fonksiyonlarındaki sabitlerden dolayı farklı çalışma zamanlarına sahip olabilirler. Ama iki algoritmanın da kırılma noktası aynı n değerine sahiptir.

Notasyonlarda eşitlik"=" gösterimi

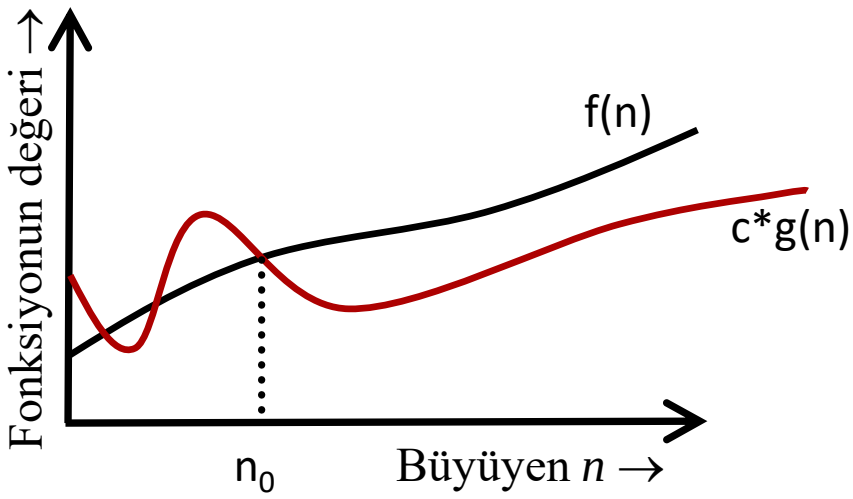
- A=B ise B = A anlamında değil mi?
- Fakat, f(n) = O(g(n)) , O(g(n)) = f(n) anlamına gelmez. Burada tek eşitlik söz konusudur.
- Burada "=", üyelik işlemi (ϵ) olarak tercih edilmiştir.
  - f(n) = O(g(n))→ f(n) ϵ O(g(n)) dir
  - O(g(n)) bir küme anlamına gelir.
  - f(n) = O(g(n)) → O(g(n)) ={ f(n) gösterimi doğrudur.

Diğer Asimptotik Notasyonlar

Ω-simgelemi (alt sınırlar)

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

- Her durumda **f(n) ≥ c g(n)** ve **n ≥ n_0** koşullarını sağlayan pozitif, sabit **c** ve **n_0** değerleri bulunabiliyorsa **f(n)=Ω(g(n))** dir.



Ω notasyonu-Örnek

- $2n + 5 \in \Omega(n)$ olduğunu gösteriniz
  - $\mathbf{n_0 \geq 0}$, $2n+5 \geq n$, olduğundan sonuç elde etmek için $c=1$ ve $n_0 = 0$ 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

Examples

- $5n^2 = \Omega(n)$

  $\exists\ c, n_0$ such that: $0 \le cn \le 5n^2 \Rightarrow cn \le 5n^2 \Rightarrow c = 1$ and $n_0 = 1$

- $100n + 5 \ne \Omega(n^2)$

  $\exists\ c, n_0$ such that: $0 \le cn^2 \le 100n + 5$

  $100n + 5 \le 100n + 5n\ (\forall\ n \ge 1) = 105n$

  $cn^2 \le 105n \Rightarrow n(cn - 105) \le 0$

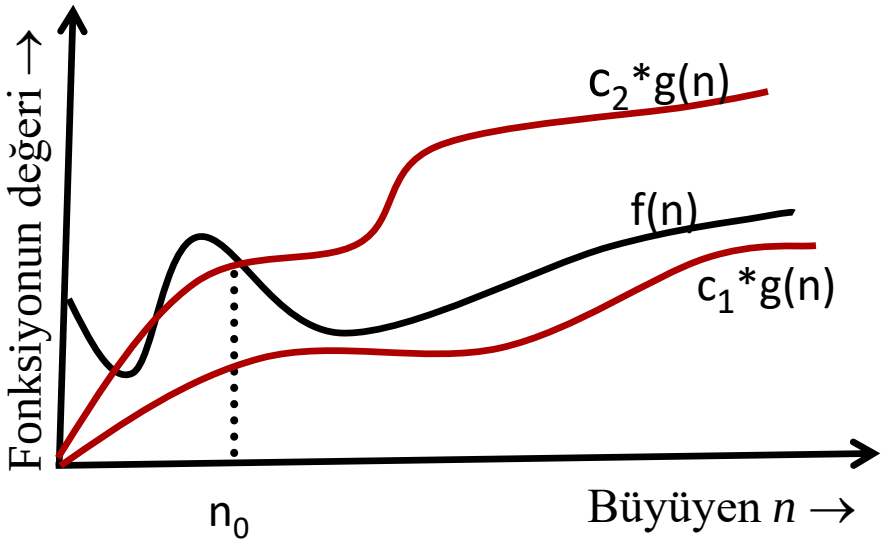
  Since n is positive $\Rightarrow cn - 105 \le 0 \Rightarrow n \le 105/c$

  $\Rightarrow$ contradiction: n cannot be smaller than a constant

- $n = \Omega(2n)$, $n^3 = \Omega(n^2)$, $n = \Omega(\log n)$

Θ Notasyonu – Sıkı Sınırlar

- Her durumda $c_1.g(n) \leq f(n) \leq c_2.g(n)$ ve $n \geq n_0$ koşullarını sağlayan pozitif, sabit c_1,c_2 ve n_0 değerleri bulunabiliyorsa $f(n)= \Theta(g(n))$ ifadesi doğrudur.



max n_0 alınır

$2n+5 \geq 2n$ $\quad n_0 \geq 1$

$2n+5 \leq 3n$ $\quad n_0 \geq 5$

Θ notasyonu- Örnek

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

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

Examples

- $n^2/2 - n/2 = \Theta(n^2)$
  - $\frac{1}{2} n^2 - \frac{1}{2} n \leq \frac{1}{2} n^2 \; \forall n \geq 0 \quad \Rightarrow \quad c_2 = \frac{1}{2}$
  - $\frac{1}{2} n^2 - \frac{1}{2} n \geq \frac{1}{2} n^2 - \frac{1}{2} n * \frac{1}{2} n \; (\forall n \geq 2) = \frac{1}{4} n^2 \Rightarrow \quad c_1 = \frac{1}{4}$

- $n \neq \Theta(n^2)$: $c_1 \; n^2 \leq n \leq c_2 \; n^2 \Rightarrow$ only holds for: $n \leq 1/c_1$
- $6n^3 \neq \Theta(n^2)$: $c_1 \; n^2 \leq 6n^3 \leq c_2 \; n^2 \Rightarrow$ only holds for:

  $n \leq c_2 / 6$

n değerini keyfi olarak belirlemek imkansızdır. Çünkü c_2 sabittir.

Another example

- Prove that $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
  - **Determine c_1, c_2 and n_0 such that**

$$c_1 n^2 \le \frac{1}{2}n^2 - 3n \le c_2 n^2$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \le c_2$$

$$\frac{1}{2} - \frac{3}{n} \le c_2 \rightarrow n \ge 1, c_2 \ge \frac{1}{2}$$

$$c_1 \le \frac{1}{2} - \frac{3}{n} \rightarrow n \ge 7, c_1 \le \frac{1}{14}$$

For any polynomial $p(n) = \sum_{i=0}^{d} a_i n^i$
$p(n) = \Theta(n^d)$

- c_1=1/14, c_2=1/2, n_0=7

O, Ω ve Θ notasyonları arasındaki ilişkiler

- Eğer g(n) = Ω(f(n)) ise→ f(n)=O(g(n))
- Eğer f(n)= O(g(n)) ve f(n)= Ω(g(n)) ise →f(n)= Θ(g(n))
- Θ(g(n)) = O(g(n)) ∩ Ω (g(n))

.

Ortalama Durum Kötü Durum Iyi Durum

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

O, Ω ve Θ notasyonları arasındaki ilişkiler

- **n**'nin büyük olduğu ve sabitlerin elendiği durumlarda.
- O(g(n)) düşünürsek f(n) ile "eşit veya büyük"
  - Üstten sınır: f(n) ile "aynı hızda veya hızlı büyür"
- Ω(g(n)) düşünürsek f(n) ile "eşit veya küçük"
  - Alttan sınır: f(n) ile "aynı hızda veya yavaş büyür"
- Θ(g(n)) düşünürsek f(n) ile "eşit"
  - Alttan ve Üsten sınır : büyüme oranları eşit
- Her bağıntı için alt ve üst sınırlar aynı zamanda belirlenemiyorsa veya bu sınırlardan sadece biri belirlenebiliyorsa, o sınıra göre notasyon gösterimi yapılır.

Θ Notasyonunun özellikleri

- $f(n) = \Theta(f(n))$, yansıtma (reflexivity) özelliği
- $g(n) = \Theta(f(n))$ olduğu durumda $f(n) = \Theta(g(n))$ dir. simetri (symmetry) özelliği
- Eğer $f(n)= \Theta (g(n))$ ve $g(n)= \Theta (h(n))$ ise $f(n)= \Theta(h(n))$ geçişme (transitivity) özeliği
- $c> 0$ olduğu her hangi bir durum için, bu fonksiyon $c.f(n) = \Theta(f(n))$ dir.
- Eğer $f_1= \Theta(g_1(n))$ ve $f_2(n)=\Theta (g_2(n))$ ise $(f_1+ f_2)(n)= \Theta (\max\{ g_1(n),g_2(n)\})$
- Eğer $f_1= \Theta(g_1(n))$ ve $f_2(n)=\Theta(g_2(n))$ ise $(f_1.f_2)(n)= \Theta((g_1.g_2)(n))$
- Simetri özelliği dışındaki diğer özellikler **O** ve Ω notasyonlarında da vardır.

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.

o-notasyonu

- **o(g(n)) = { f(n) : tüm n ≥ n_0 değerlerinde**
  c>0 sabiti için n_0 sabiti varsa
  0 ≤ f(n) < cg(n).}

- Çalışma sürelerinin karşılaştırılması için kullanılır. Eğer f(n)= o(g(n)), ise g (n), f(n) fonksiyonundan daha ağırlıklıdır. Sıkı bir üst sınır vermez.

  - $\lim_{n \to \infty} f(n)/g(n) = 0$
  - **o**-notasyonunda **n** sonsuza gittiğinde f(n) fonksiyonu g(n) fonksiyonu karşısında önemini kaybeder.

o-notasyonu

- Örnek: $n^2/2 \in o(n^3)$,
  - $\lim_{n \to \infty} (n^2/2)/ n^3 = \lim_{n \to \infty} 1/2n = 0$
- Örnek: $2n = o(n^2)$, fakat $2n^2 \neq o(n^2)$
- Örnek: $2n^2 = o(n^3)$ ($n_0 = 2/c$)
- **Önerme**: $f(n) \in o(g(n)) \Rightarrow O(f(n)) \subset O(g(n))$

ω-notasyonu

- **$\omega(g(n)) = \{ f(n)$: tüm $n \geq n_0$ değerlerinde**
  $c>0$ sabiti için n_0 sabiti varsa
  $0 \leq c.g(n) < f(n)$ }
- Çalışma sürelerinin karşılaştırılması için kullanılır. Eğer $f(n)=\omega(g(n))$, ise g (n), f(n) fonksiyonundan daha ağırlıklıdır. Sıkı bir alt sınır vermez.
  - $\lim_{n \to \infty} f(n)/g(n) = \infty$, eğer limit varsa
- Örnek: $n^2/2 \in \omega(n)$,
  - $\lim_{n \to \infty} (n^2/2)/ n = \infty$, fakat $n^2/2 \neq \omega(n^2)$
- Örnek: $n^{1/2} = \omega(\log n)$, ($n_0 = 1+1/c$)

Asimptotik Notasyonlar

- $f(n) = O(g(n)) \cong f \leq g$
- $f(n) = \Omega(g(n)) \cong f \geq g$
- $f(n) = \Theta(g(n)) \cong f = g$
- $f(n) = o(g(n)) \cong f < g$
- $f(n) = \omega(g(n)) \cong f > g$

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

En iyi (Best), En kötü (Worst), Ortalama(Average) Durum Analizi

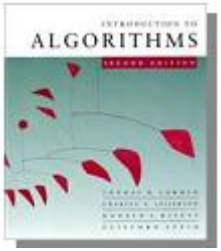
- **En iyi durum (best case):** Bir algoritma için, çalışma zamanı, maliyet veya karmaşıklık hesaplamalarında en iyi sonucun elde edildiği duruma “en iyi durum” denir. Bir giriş yapısında hızlı çalışan yavaş bir algoritma ile hile yapmak. (gerçek dışı) Ör: Bütün elemanların sıralı olduğu durum.

- **En kötü durum (worst case):** Tüm olumsuz koşulların oluşması durumunda algoritmanın çözüm üretmesi için gerekli maksimum çalışma zamanıdır. (genellikle). Ör: Bütün elemanlar ters sıralı.

- **Ortalama durum (avarage case):** Giriş parametrelerin en iyi ve en kötü durum arasında gelmesi ile ortaya çıkan durumda harcanan zamandır. Fakat bu her zaman ortalama durumu vermeyebilir. (bazen) Ör: Elemanların yaklaşık yarısı kendi sırasındadır.

En iyi (Best), En kötü (Worst), Ortalama(Average) Durum Analizi

- **Dizi arama**
  - Worst case = O(n) Average case = O(n)
- **Quick sort**
  - Worst case = $O(n^2)$ Average case = O(n log n)
- **Merge Sort, Heap Sort**
  - Worst case = O(n log n) Average case = O(n log n)
- **Bubble sort**
  - Worst case = $O(n^2)$ Average case = $O(n^2)$
- **Binary Search Tree:** Bir elaman için arama
  - Worst case = O(log n) Average case = O(log n)

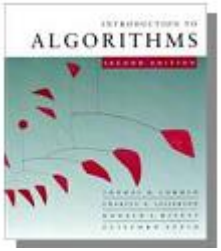


Diziler (Arrays)

- Diziler, tipleri homojen olan birden fazla elemandan oluşan veri grubudur. Diziler tek boyutlu, iki boyutlu, üç boyutlu, vb. şeklinde tanımlanabilirler.

- Diziler genelde diğer veri yapılarında kullanılan bir veri yapısıdır. Diziler **statik veriler** olarak da isimlendirilebilirler.
- Örnek: Tek boyutlu A dizisi

0	1	2	3	4	5	6	7	8	9
6	7	12	23	46	78	12	5	8	2



Diziler-Doğrusal Arama (Search)

- Statik ve sıralanmamış veri dizisi üzerinde arama algoritması (Lineer arama).

```
    Lineer Arama(n,x)
1.  i←1, Veri_Var←0
2.  (i<=n) ve ~ (Veri_Var) olduğu sürece devam et
3.      eğer Dizi[i]=x ise
4.          Veri_Var←1
5.      i←i+1
```

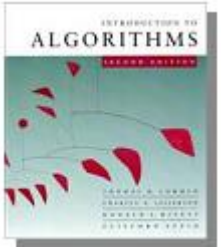
- **x** , dizi içinde aranacak veri ve **n** dizinin boyutudur.
- **~** simgesinin anlamı önüne geldiği mantıksal ifadenin değilini almaktır.

Diziler-Arama (Serach) Analizi

- **Başarısız Arama : O(n)**
- **Başarılı Arama:**
  - **Best-Case: x** dizinin ilk elemanı ise: **O(1)**
  - **Worst-Case: x** dizinin son elamanı ise: **O(n)**
  - **Average-Case:** Listedeki her bir sayıyı bir kez aradığımızı düşünelim. Anahtar karşılaştırmalarının sayısı 1,2,……,n ise : **O(n)**

$$T(n) = \begin{cases} \Theta(1) & \text{en iyi durum} \\ \Theta(n) & \text{en kötü durum} \\ \Theta(n) & \text{ortalama durumu} \end{cases}$$

$$\frac{\sum_{i=1}^{n} i}{n} = \frac{(n^2+n)/2}{n}$$

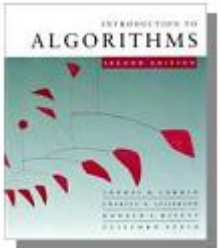


Diziler-Arama (Serach) Analizi

- Lineer arama algoritması, herhangi bir dizi üzerinde yapılan bir aramadır.

- Dizinin elemanları sıralı olması veya olmaması herhangi bir anlam ifade etmez. Bu aramada aranacak eleman dizinin bütün elemanları ile karşılaştırılır ve bu işleme, aranan eleman dizide bulununcaya kadar veya dizide olmadığı kesinlik kazanıncaya kadar devam edilir.

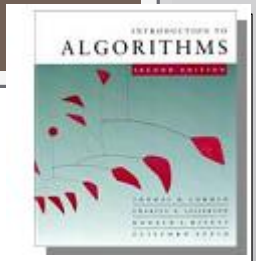
- Eğer dizinin elemanları sıralı ise, bu durumda mertebesi daha iyi olan bir algoritma kullanılabilmektedir.
  - **İkili Arama (Binary Search)**



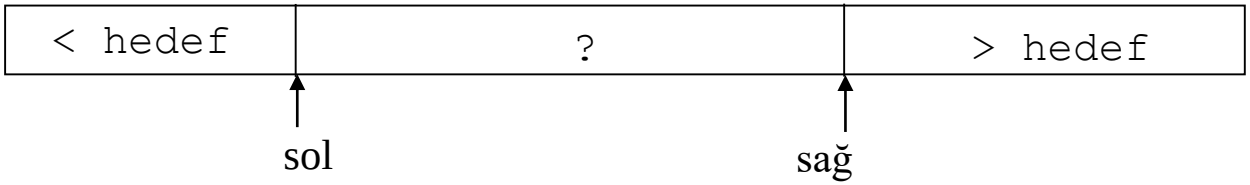
İkili Arama (Binary Search)

- İkili aramada dizi sıralanmış olduğundan, dizinin ortasında bulunan sayı ile aranan sayıyı karşılaştırarak arama boyutunu yarıya düşürülür ve bu şekilde devam edilir.

- Bu algoritmanın temel mantığı aranacak elemanı dizinin ortasındaki eleman ile karşılaştırıp, aranacak eleman eğer bu elemana eşitse, amaca ulaşılmıştır. Eğer eşit değilse, bu durumda aranacak eleman dizinin hangi parçası içinde olabileceği kararı verilir. Bu sayede arama boyutu yarıya düşürülür ve bu şekilde devam edilir.



İkili Arama (Binary Search)

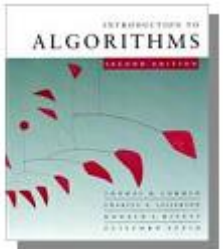


- Statik ve sıralı veri dizisi üzerinde ikili arama algoritması.

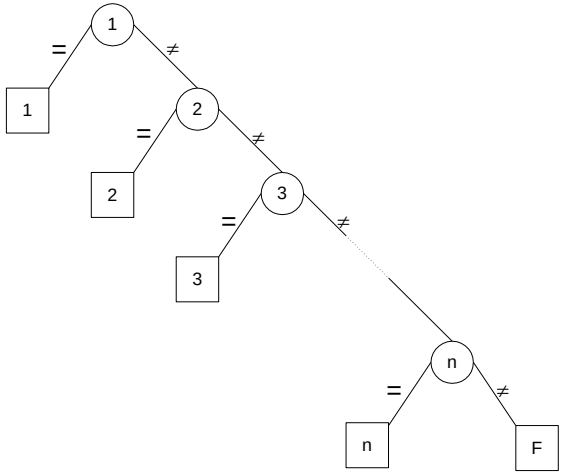
İkili Arama(Dizi,n,x,bulundu, yeri)

1. **Yerel değişkenler**
   alt, ust, orta : integer;
2. **Ust ← n, alt ← 1, bulundu ← 0**
3. **(bulundu = 0) ve (ust ≥ alt) olduğu sürece devam et**
4. **orta ← ⌊ (alt + ust) / 2 ⌋**
5. **eğer x = Dizi[orta] ise**
6. **bulundu ← 1**
7. **değil ve eğer x < $Dizi_2$ [orta] ise**
8. **ust ← orta-1**
9. **değilse**
10. **alt ← orta+1**
11. **yeri ← orta**

İkili Arama (Binary Search)

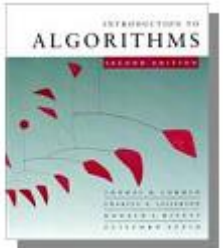


- İkili aramanın mantığı veya işleyiş şekli ağaç şeklinde gösterilebilir ve bu ağaca **karşılaştırma ağacı** veya **karar ağacı** denir. **Karşılaştırma ağacı**, bir algoritmanın yapmış olduğu karşılaştırmaların hepsinin temsil edildiği bir ağaçtır.
- Örneğin, statik veri yapıları üzerinde Lineer arama algoritmasının karşılaştırma ağacı;

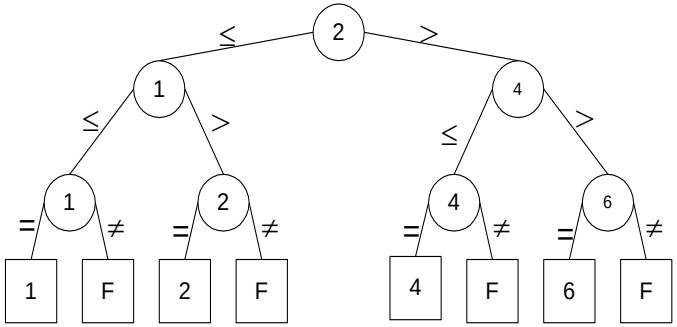


Aranacak bir elemanın dizinin içinde olup olmadığını test etmek amacıyla dizinin her elemanı ile karşılaştırma yapıldığından ağaç tek dal üzerine büyümektedir ve ağacın bir tarafı hiç yok iken, diğer tarafı çok büyüyebilir. Buradan da rahatlıkla görülebileceği gibi Lineer arama algoritması iyi bir algoritma değildir.

İkili Arama (Binary Search)

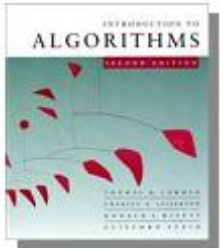


- Statik veri yapıları üzerinde yapılan ikili arama algoritması için karşılaştırma ağacı;



- Görülen karşılaştırma ağacı dengeli bir ağaçtır. Bazı durumlarda karşılaştırma ağacının bütün yaprakları aynı seviyede olmayabilir.

İkili Arama (Binary Search)

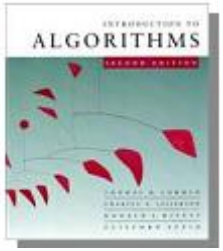


- Bu algoritmanın analizini yapmak için ilk olarak $n=2^k$ kabulü yapılsın.

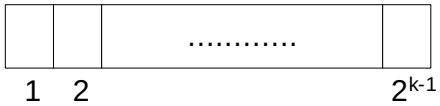
- $\frac{n}{2}, \frac{n}{2^2}, \frac{n}{2^3}, \dots, \frac{n}{2^k} = 1$, $n=2^k$

- Bu kabulün yapılması genelliliği bozmayacaktır. Algoritma çalıştığında ilk bakılacak eleman 2^{k-1} endeksli eleman olacaktır. Eğer eşitlik varsa, amaca ulaşılmıştır. Eşitlik yoksa geriye kalan ve her birinin boyutu 2^{k-1} olan dizi parçalarının hangisinde aranan elemanın olacağına karar verilir.

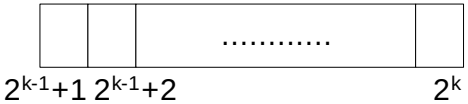
İkili Arama (Binary Search)



- Eğer x değişkeninin değeri Dizi[2^{k-1}] elemanın değerinden küçükse,

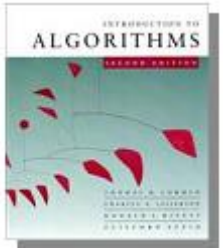


- x elemanı bu parçanın içinde olabilir, diğer parçanın içinde olması mümkün değildir. Eğer x değişkeninin değeri Dizi[2^{k-1}] elemanın değerinden büyükse,



- x elemanı bu parçanın içinde olabilir, diğer parçanın içinde olması olasılığı sıfırdır. Bu şekilde geriye kalan parçanın içindeki eleman sayısı 2’ nin kuvveti kadar eleman kalacaktır. Bundan dolayı ortadaki elemanı bulmak için taban veya tavan fonksiyonuna ihtiyaç duyulmayacaktır. Dizi üzerinde yapılacak bölme sayısı logn olacaktır.

İkili Arama (Binary Search)

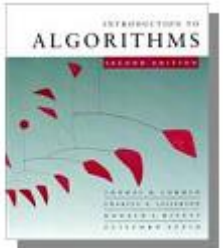


- Bir dizi için başarılı arama sayısı n ve 1 tanede başarısız arama sayısı yapılabilir. Toplam n+1 tane arama yapılabilir.
- Bu algoritmada en iyi durum aramasında döngü kısmı bir sefer çalışır ve en kötü durumda döngü kısmı lgn+1 sefer çalışır.
- Ortalama arama zamanına bakılacak olursa, toplam arama sayısı ile arama adım sayısı çarpılır ve n değerine bölünür.
- Asimptotik notasyonda sabit katsayılar ihmal edildiğinden dolayı, ikili arama algoritmasının mertebesi T(n),

$$T(n) = \begin{cases} \Theta(1) & \text{en iyi durum} \\ \Theta(\lg n) & \text{en kötü durum} \\ \Theta(\lg n) & \text{ortalama durum} \end{cases}$$

$$\frac{\sum_{i=1}^{n} \log n}{n} = \frac{n\log n}{n} = \log n$$

İkili Arama (Binary Search)



- Bu algoritmanın da en kötü durumu ile ortalama durumun asimptotik davranışları aynıdır. Fakat en kötü durum ve ortalama durum mertebeleri (çalışma zamanı) logaritmik olduğundan, lineer aramaya göre çok iyi olan bir algoritmadır.

- Eğer bir dizi içindeki veriler sıralı ise, her zaman ikili arama algoritmasını kullanmak sistemin performansını arttırır.

Binary Arama Analizi

- **Başarısız Arama :** O(n)
- **Başarılı Arama:**
- **Best-Case:** O(1)
- **Worst-Case:** O(log n)
- **Average-Case:** O(log n)

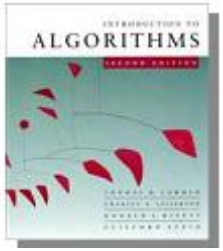
```
int ikiliArama(int A[], int N, int sayi)
{   sol = 0;
    sag = N-1;
    while (sol <= sag){
            int orta = (sol+sag)/2;
            if (A[orta] == sayi) return orta;
            else if (sayi < A[orta]) sag = orta – 1;
            else sol = orta+1;
          }
    return –1;
}
```

$$\sum_{i=1}^{n} i2^i = 1 \cdot 2 + 2 \cdot 2^2 + \cdots + n2^n = (n-1)2^{n+1} + 2$$

$$\frac{1 \times 1 + 2 \times 2 + 3 \times 4 + \ldots + \log n 2^{\log n - 1}}{n} = \frac{1}{n} \sum_{i=1}^{\log n} i2^{i-1}$$

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^{\log n} i2^{i-1} &= \frac{(\log n - 1)2^{\log n} + 1}{n} \\ &= \frac{n(\log n - 1) + 1}{n} \\ &\approx \log n - 1 \end{aligned}$$

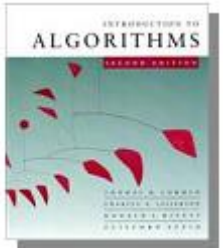
İkili Arama (Binary Search)



- İkili arama algoritmasının performansını ölçmek için, bu algoritmanın T(n) zaman bağıntısının tekrarlı bağıntısı (özyinelemeli bağıntı) elde edilebilir. İkili arama algoritmasının T(n) bağıntısı
- $T(n)=T(\lfloor n/2 \rfloor)+\Theta(1)$

  şeklinde olur. Bu tekrarlı bağıntının çözümü iteratif (iterasyon) yapılırsa (taban fonksiyonunu ihmal ederek),
  - $T(n)=T(n/2)+\Theta(1)$
  - $=\Theta(1)+(\Theta(1)+T(n/2^2))$
  - $=\Theta(1)+(\Theta(1)+(\Theta(1)+T(n/2^3)))$
  - $=(\lg n-1)\,\Theta(1)+T(n/2^{\lg n})$
  - $=(\lg n-1)\Theta(1)+T(1)$
- elde edilir.

İkili Arama (Binary Search)



- İkili arama için T(1)=Θ(1) olur, bundan dolayı
- T(n) = (lgn-1)Θ(1)+Θ(1)
- = (lgn)Θ(1)
- = Θ(lgn) olur.
- Master yöntemi kullanılırsa, f(n)= Θ(1) ve a=1, b=2 ,
- $n^{\log_b a} = n^{\log_2 1} = n^0 = 1$ olduğundan olur. Bundan dolayı f(n)= $\Theta\left(n^{\log_b a}\right)$ olur. Master yönteminde bu şart sağlandığında
- T(n) = $\Theta\left(n^{\log_b a} \lg n\right)$
- = $\Theta(n^0 \lg n)$
- = Θ(lgn)
- elde edilir.
- Bu yöntemler ilerleyen bölümde detaylı olarak ele alınacaktır.

3.Hafta Algoritmaların Analizi

Araya Yerleştirme Sırlaması
(Insert Sort)
Birleştirme Sıralaması (Merge Sort)
Yinelemeler

Sorular

- 1. f(n) ve g(n) asimptotik negatif olmayan fonksiyonlar olsunlar. max(f(n),g(n))= Θ(f(n)+g(n)) olduğunu gösteriniz.
- 2. b>0 olmak üzere reel a ve b sabitleri için $(n+a)^b=\Theta(n^b)$ olduğunu gösteriniz.
- 3. $f(n)=2^{n+1}$ ve $g(n)=2^n$ ise f(n)=O(g(n)) midir? $f(n)=3^{2n}$ olduğu durumda f(n)=O(n)) midir?
- 4. Bir algoritmanın en iyi çalışma performansında çalışma zamanı Ω(g(n)) ve en kötü çalışma durumunda çalışma zamanı O(g(n)) ise, algoritmanın mertebesi f(n)=Θ(g(n)) olduğunu gösteriniz.
- 5. Aşağıdaki kümelerden hangisi veya hangileri boş kümedir?
- $\omega(f(n))\cap o(f(n))$
- $\Omega(f(n))\cap O(f(n))$

Sorular

- 6. Asimptotik notasyonların tanımı bir parametre için verilmiştir. Eğer algoritmada birden fazla parametre varsa, bu durumda tanımlama şu şekilde yapılabilir. f(m,n) ve g(m,n) asimptotik negatif olmayan fonksiyonlar olsunlar. Bu durumda O-notasyonun tanımı
- $O(g(m,n))=\{ f(m,n)$: c, n_0 ve m_0 pozitif sabitler olmak üzere $0 \leq f(m,n) \leq cg(m,n)$, $n \geq n_0$ ve $m \geq m_0\}$.
- Benzer tanımlamaları Θ, Ω, ϖ ve o için yapınız.
- 7. $\lg(n!)=\Theta(n\lg n)$ ve $n!=o(n^n)$ olduğunu gösteriniz.
- 8 . Eğer bir algoritmanın çalışma ortalama zaman bir k>0 sabiti için $T(n)=O(n^k)$ ise $T(n)=n^{O(1)}$ olduğunu gösteriniz. Tersinin de doğruluğunu gösteriniz.
- 9. $\lceil \lg n \rceil!$ fonksiyonu asimptotik olarak sınırlı mıdır? n>0 için $\lfloor \lg n \rfloor!$ fonksiyonu polinom olarak sınırlı mıdır?

Sorular

- 10. P(n), derecesi d olan bir polinom olsun. k>0 bir sabit olmak üzere $P(n)=O(n^k)$, $P(n)=\Theta(n^k)$, $P(n)=\Omega(n^k)$, $P(n)=\omega(n^d)$ ve $P(n)=o(n^d)$ durumları için d ve k arasındaki ilişkileri gösteriniz.
- 11. f(n) ve g(n) asimptotik pozitif fonksiyonlar olsunlar. Aşağıdakilerin doğruluğunu veya yanlışlığını gösteriniz.
  - $f(n)=O(g(n))\rightarrow g(n)=O(f(n))$
  - $f(n)+g(n)=\Theta(\min(f(n),g(n)))$
  - $f(n)=O(g(n))\rightarrow 2^{f(n)}=O(2^{g(n)})$
  - $f(n)=O((f(n))^2)$
  - $f(n)=O(g(n))\rightarrow g(n)=\Omega(f(n))$
  - $f(n)=\Theta(f(n/2))$
  - $f(n)+o(f(n))=\Theta(f(n))$

Sorular

- 12. f(n) ve g(n) asimptotik pozitif fonksiyonlar ve yeterince büyük bütün n' ler için $f(n) \geq 1$ ve $\lg(g(n))>0$ olsun. $f(n)=O(g(n))$ ise $\lg(f(n))=O(\lg(g(n)))$ olduğunu gösteriniz.
- 13. Eğer $f(n)=O(F(n))$ ve $g(n)=O(G(n))$ ise

$$\frac{f(n)}{g(n)} = O\left(\frac{F(n)}{G(n)}\right)$$

olur. Bu iddianın doğru ya da yanlış olduğunu gösteriniz.

- 14. Aşağıda iddiaların doğru ya da yanlış olduğunu gösteriniz.
- Eğer $f(n)=O(g(n))$ ve $g(n)=O(h(n))$ ise, $f(n)=O(h(n))$ olur.
- Eğer $f(n)=\Theta(g(n))$ ve $g(n)=\Theta(h(n))$ ise, $f(n)=\Theta(h(n))$ olur.

Ek-

Algoritmaların doğruluğu

- Bir algoritma geçerli olan herhangi bir giriş için sonlanmakta ve istenen bir sonucu üretmekte ise doğrudur.

- Algoritmaların doğruluğunun kontrolünde pratik teknikler kullanılır.

Döngü Değişmezleri (Loop Invaritants)

- **Değişmezler (Invariants)-** Herhangi bir zamanda ulaşıldıklarında veya işlem yapıldıklarında doğru oldukları varsayılır (algoritma çalışma sırasında tekrarlı gerçekleşen işlemler, Örnek: döngülerde)
- Döngü değişmezleri için üç şeyi göstermek zorunludur:
  - **Başlatma(Initialization)** - ilk iterasyondan önce doğrudur
  - **Koruma(Maintenance)** - bir iterasyondan önce doğruysa bir sonraki iterasyondan öncede doğruluğunu korur
  - **Sonlandırma (Termination)** – döngünün değişmezleri bitirmesi algoritmanın doğruluğunu gösterir

Örnek: Binary Search (1)

- null değeri döndüğünde q değerinin A dizisinde olmadığından emin olmak istiyoruz.
- **Değişmez** : her **while** döngüsünün başlangıcında, $A[i] < q$ bütün $i \in [1 \ldots left-1]$ ve $A[i] > q$ bütün $i \in [right+1 \ldots n]$
- **Başlatma**: *left=1, right=n* olarak seçilir ve *left*' in solunda ve *right'* ın sağında hiçbir elaman kalmaz.

```
left ← 1
right←n
do
    j← ⌊ (left + right ) / 2 ⌋
   if A[j]=q then return j
   else if A[j]>q then right ←j-1
   else left= j+1
while left<=right
return null
```

Örnek: Binary Search (2)

- **Koruma** :Eğer *A[j]>q* ise *A[i]>q* olur her *i* $\in$*[j…n]*, çünkü dizi sıralıdır. Algoritma *right* değişkenine *j-1* değerini atar.
- **Sonlandırma**: *left>right* olduğunda döngü bitirilir. *q* değeri *left* solundaki A'nın tüm değerlerinden büyüktür ve *right'* ın sağındaki A'nın tüm değerlerinden küçüktür. Bu A'nın tüm elemanlarından *q* değerinin küçük yada büyük olduğunu gösterir.

```
left ← 1
right←n
do
    j← ⌊ (left + right ) / 2 ⌋
   if A[j]=q then return j
   else if A[j]>q then right ←j-1
   else left= j+1
while left<=right
return null
```

```
for  j ← 2 to length[A]
do   key ← A[j]
     i ← j –1
     while i > 0 and A[i] > key
     do   A[i+1] ← A[i]
          i ← i –1
     done
     A[i+1]  ← key
done
```

Örnek: Insert Sort

- **Değişmez:** her for döngüsü *başlangıcında, A[1…j-1]* dizisi sıralı olarak *A[1…j-1]* aralığındaki elemanlardan ibarettir.
- **Başlatma:** *j=2,* olarak alınır. Çünkü A[1] zaten sıralıdır.
- **Koruma** : İçteki while döngü elemanlar arasında *A[j-1], A[j-2], …, A[j-k]* şeklinde sırasını değiştirmeden bir önceki elemana gider. Daha sonra *A[j]* elemanı *k.* Pozisyona yerleştirilir, böylece *A[k-1]<A[k]<A[k+1]* olur.
  A[1...j-1] sıralı *+ A [j]* → *A[1...j]* sıralı
- **Sonlandırma**: *j=n+1* olduğunda döngü bitirilir. Böylece *A[1...n]* orijinal olarak alınmış olan *A[1...n]* elemanla aynıdır ancak sıralanmış şekildedir.