ALGORİTMA ANALİZİ

1

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DERS ANA HATLARI

SINAVLAR

- Ara Sınav (Klasik sınav, %40)
- Final Sınavı (Klasik sınav, %60)

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

2

DERS ANA HATLARI

KAYNAKLAR

- Michael T. Goodrich, Roberto Tamassia, David M. Mount, Data Structures and Algorithms in C++,
- Michael T. Goodrich, Roberto Tamassia, Algorithm Design and Applications, www.it-ebooks.info, Wiley.
- Olcay Taner Yıldız, C&&Java ile Veri Yapılarına Giriş, Boğaziçi Üniversitesi Yayınevi.
- Rıfat Çölkesen, Veri Yapıları ve Algoritmalar, Papatya Yayıncılık.
- Sefer Kurnaz, Veri Yapıları ve Algoritma Temelleri, Papatya Yayıncılık.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMANIN VE VERİ YAPILARININ ÖNEMİ NEDİR?

- Problem çözülürken;
  - Doğru bir şekilde adım adım sıralayabilmek,
  - Uygun bir veri modeli belirlemek gerekmektedir.
- Programlama dillerinin bize sağladığı temel veri modelleri, pek çok gereksinim çerçevesinde ortaya atılmış ileri düzey veri modeli veya kendimizin geliştireceği yeni bir veri modeli ile problemi bilgisayarlarda çözecek yöntemleri geliştirebilmeyi hedefler. Bir veri yapısı, verinin organizasyonu ve veriye erişim sistematiği yoludur.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA NEDİR?

Algoritma bir problemin çözümünde izlenecek yol anlamına gelir. Bir başlangıç durumundan başlar ve açıkça belirlenmiş bir son durumunda sonlanan, sonlu işlemler (adımlar) kümesidir. Algoritmalar bilgisayarlar tarafından işletilebilir. Programlama dillerinin temeli algoritmaya dayanmaktadır ve dilden bağımsız.

- Her adım son derece belirleyici olmalıdır. Hiç bir şey şansa bağlı olmamalıdır.
- Belirli bir sayıda adım sonunda algoritma sonlanmalıdır.
- Algoritmalar karşılaşılabilecek tüm ihtimalleri ele alabilecek kadar genel olmalıdır.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA NASIL HAZIRLANIR?

Bir problemin çözümü hazırlanırken 3 temel bileşenimiz vardır. Bunlar;

- Değişkenler
- Algoritma
- Akış Diyagramı

Bununla ilgili küçük bir örnek verelim: Örneğin klavyeden girilen iki sayının toplamını bulan ve sonucu ekrana yazdıran programın algoritması ve akış diyagramı istenseydi.

- Değişkenler

  birinci sayı: x
  ikinci sayı :y
  iki sayının toplamı:toplam
- Algoritma

  Adım 1:Başla
  Adım 2:Birinci sayıyı oku ve x değişkenine aktar.
  Adım 3:ikinci sayıyı oku ve y değişkenine aktar.
  Adım 4:x ve y sayılarını topla sonucu toplam değişkenine aktar.
  Adım 5:Toplam değerini ekrana yazdır.
  Adım 6:Dur
- Akış diyagramı (http://www.dahiweb.com/algoritma-nedir/)

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA ANALİZİ

Her hangi bir algoritmanın ne kadar hızlı çalıştığını veya ne kadar sürede çalıştığını o algoritmayı analiz ederek yapabiliriz. Peki, algoritma analizi nedir? Algoritma analizi denince akla iki önemli kavram gelir bunlar alan ve zaman karmaşıklığıdır.

- Alan karmaşıklığı yazdığınız algoritma bellekten ne kadar yer kullanıyor,
- zaman karmaşıklığı ise yazdığınız algoritmanın çalışma süresini ifade eder.

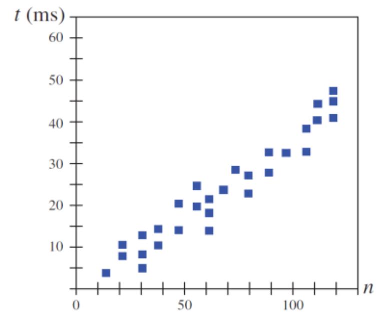
ALGORİTMA ANALİZİ

Algoritma analizine neden ihtiyaç duyarız;

- çünkü yazdığımız algoritmanın performansını bilmek isteriz,
- farklı algoritmalarla karşılaştırmak isteriz
- ve daha iyisi mümkün mü sorusuna ancak analiz yaparak cevap verebiliriz.

DENEYSEL ÖRNEKLEM

Bir deneysel çalışmada, ***n*** giriş sayısını göstermekte ve ***t*** işlem zamanın değişimini göstermektedir. Bu veri kümesinden en iyi giriş verisini istatistiksel olarak tanımlaya çalışalım. Veri kümesinden o şekilde seçilmelidir ki en iyi sembolize etsin.



Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DENEYSEL ÖRNEKLEM

Deneysel çalışmanın en iyi çalışma zamanını seçerken aşağıdaki sınırlamaları dikkate almalıyız;

- Deneyler sadece test girişlerinin sınırlı bir kümeye uygulanmalıdır, dolayısı ile deneye dahil olmayan girdiler çalıma zamanını için ayrılırlar (ve bu girişler önemli olabilir).
- Eğer deneyler aynı yazılım ve donanımda yapılmazsa, iki algoritmanın deneysel çalışma zamanın karşılaştırmak zor olur.
- Deneysel olarak çalışma süresi incelemek için bir algoritmayı yürütmek ve tamamı ile uygulamak zorundayız.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DENEYSEL ÖRNEKLEM

Algoritmaların çalışma zamanın genel bir analiz için;

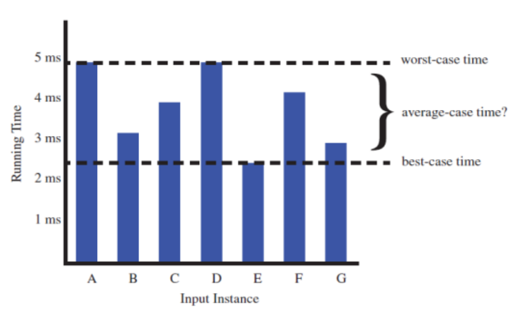
- Olası bütün giriş durumlarını dikkate alın.
- Yazılım ve donanımdan bağımsız olarak iki algoritmayı değerlendirin.
- Gerçek uygulama yada deneyler çalıştırılmadan algoritmanın yüksek seviyesi tanımlanarak çalışılmalıdır.

Bu yöntem ile giriş boyutu ***n*** nin fonksiyonu olarak algoritmanın çalışma zamanı ***f(n)*** fonksiyonu ile tanımlanır.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

TEMEL İŞLEMLER

Çalışma zamanı için girişlerin boyutu önemlidir. Aynı boyutlar için ortalama çalışma zamanı kolay belirlenir. Farklı boyutlar için ise en iyi ve en kötü zaman arasındadır.



Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA ANALİZİ

Bir algoritma çalışmasını bitirene kadar geçen süre yürütme zamanı olarak adlandırılır. Ve algoritmada genelde eleman sayısı n olarak gösterilir ve yürütme zamanı da T(n) ile ifade edilir. Algoritmadaki eleman sayısı çok fazla olduğunda yürütme zamanı, zaman karmaşıklığı olarak adlandırılır. Ve derecesi asimptotik notasyon ile verilir. Dereceyi belirlemek için; O(o), Θ(o) veya Ω(o) gibi notasyonlar kullanılmaktadır.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA ANALİZİ

Algoritmanın işlevini yerine getirmesi için kullandığı bellek miktarına alan maliyeti denir. Örneğin n elemanlı bir dizi, elemanlarının her biri 4 byte ise bu dizi için alan maliyeti bağıntısı;

$$T(n) = 4n$$

Aynı şekilde alan karmaşıklığı da eleman sayısı çok büyük olduğu zaman alan maliyetini ifade eden asimptotik ifadedir. Zaman karmaşıklığında kullanılan notasyonlar burada da kullanılır.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA ANALİZİ

Big Oh Notasyonu O(n)

Paul Bachman tarafından tanıtılmıştır. Zaman karmaşıklığında üst sınırı gösterir. Örneğin n^3 + n^2+3n gibi bir ifadeyi O(n^3) olarak ifade ederiz.

Big Omega Notasyonu

Big Oh notasyonunun tam tersidir. Zaman karmaşıklında alt sınırı gösterir. Yani Omega ile ölçülen değerden daha hızlı bir değer elde etmeniz mümkün değildir.

Big Theta Notasyonu

Bu notasyon big Oh notasyonu ile big Omega notasyonu arasında ortalama bir karmaşıklığı ifade eder.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

ALGORİTMA ANALİZİ

```
i= 1; // C1 zamanda bir kez yapılacak
sum = 0; // C2 zamanda bir kez yapılacak
while (i <= n) { // C3 zamanda n+1 kez yapılacak; çünkü
// i değişkeninin eşit olma durumunu arıyoruz
i = i + 1; // C4 zamanda n kez yapılacak
sum = sum + i; // C5 zamanda n kez yapılacak
}
```

C1+C2+C3*(n+1)+C4*n+C5*n zamanda çalışacak, yani polinomsal yazarak, (C3+C4+C5)***n**+(C1+C2+C3) şeklinde ***n***'e bağlı bir denklem elde ederiz.

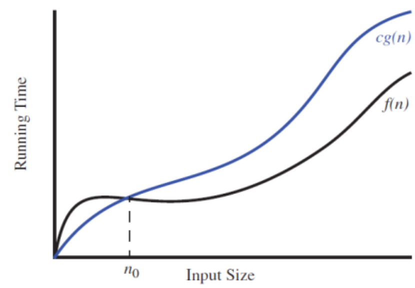
Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

BİG OH NOTASYONU

$f(n) \leq c * g(n)$

ise c>1 ve $n \geq n_0$ için bir fonksiyon tanımlanabilir. $f(n)$ fonksiyonu $g(n)$ fonksiyonun Big Oh denir.

Önceki örnekte tanımlama fonksiyonu için C*g(x) fonksiyonu şimdi (C3+C4+C5)*n şeklinde dönüşmüş ve g(x)=n, dolayısıyla O(n) karmaşıklığa sahip bir kodumuz var.



Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DİZİDEKİ SAYILARIN TOPLAMI

```
int Topla(int A[], int N)
{
  int toplam = 0;

  for  (i=0; i < N; i++){
    toplam += A[i];
  }

  return toplam;
}
```

Bu fonksiyonun çalışma zamanı ne kadardır?

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DİZİDEKİ SAYILARIN TOPLAMI

İşlem Sayısı

```
int Topla(int A[], int N)  ---> 1
{
  int topla = 0;           ---> 1

  for (i=0; i < N; i++){   ---> N
    topla += A[i];         ---> N
  }

  return topla;            ---> 1
}
                              --------
```

Toplam: 1 +1+ N + N + 1 = 2N +3

Çalışma zamanı: T(N) = 2N+3

Dr. Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DİZİDE BİR ELEMANIN ARANMASI

İşlem sayısı

```
int Arama(int A[], int N,
               int sayi) {          ---> 1
  int i = 0;                        ---> 1

  while (i < N){                    ---> N+1
    if (A[i] == sayi) break;        ---> N
    i++;                            ---> N
  }

  if (i < N) return i;              ---> }
  else return -1;                   ---> } 1
}
                                      ---------
```

Toplam: 1+N+1+N+N+1+1 = 3N+4

Dr. Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

DİZİDE BİR ELEMANIN ARANMASI

- En iyi çalışma zamanı nedir?
  - Döngü sadece bir kez çalıştı=>T(n) = 3n+4=3*1+4=7
- Ortalama (beklenen) çalışma zamanı nedir?
  - Döngü N/2 kez çalıştı =>T(n)=3*n/2+4 = 1.5n+4
- En kötü çalışma zamanı nedir?
  - Döngü N kez çalıştı =>T(n) = 3n+4

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

İÇ İÇE DÖNGÜLER

```
for (i=1; i<=N; i++){
    for (j=1; j<=N; j++){
        printf("Foo\n");
    } //bitti-içteki for
} //bitti-dıştaki for
```

- Prinf fonksiyonu kaç kez çalıştırıldı?
  - Veya Foo yazısı ekrana kaç kez yazılır?

$$T(N) = \sum_{i=1}^{N}\sum_{i=1}^{N} N = N + N * N + N * N = 2N^2 + N$$

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Matris Çarpimi

```
/* İki boyutlu dizi A, B, C.     Hesapla C = A*B    */
for (i=0; i<N; i++) {......................N
   for (j=0; j<N; j++) {..................N*N
      C[i][j] = 0; .........................N*N
      for (int k=0; k<N; k++){.........N*N*N
         C[i][j] += A[i][k]*B[k][j]; .......N*N*N
      } //bitti-en içteki for
   } //bitti-içteki for
} //bitti-dıştaki for
```

$$T(N) = \sum_{i=0}^{N-1}\sum_{j=0}^{N-1}(1+\sum_{k=0}^{N-1}1) = 2N^3 + 2N^2 + N$$

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

İkili Arama

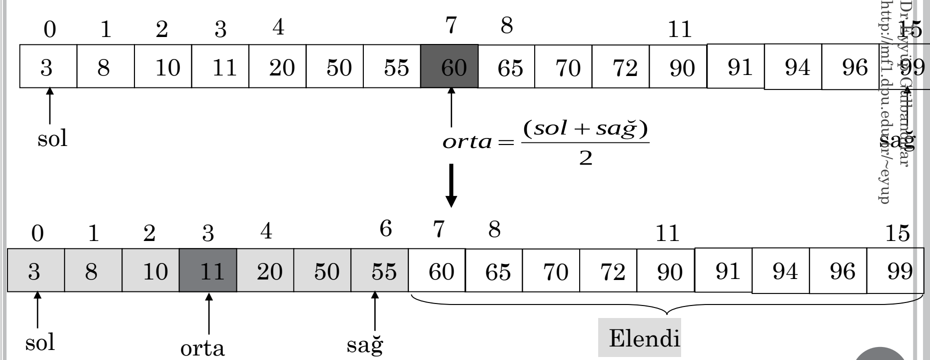
- Problem: Sıralı bir dizi veriliyor ve bir sayıyı arıyorsunuz.
  - Doğrusal arama– T(n) = 3n+2 (En kötü durum)
  - Daha iyisi yapılabilir mi?
  - Ö.g. Aşağıdaki sıralı dizide 55 sayısını arayalım

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
3	8	10	11	20	50	55	60	65	70	72	90	91	94	96	99

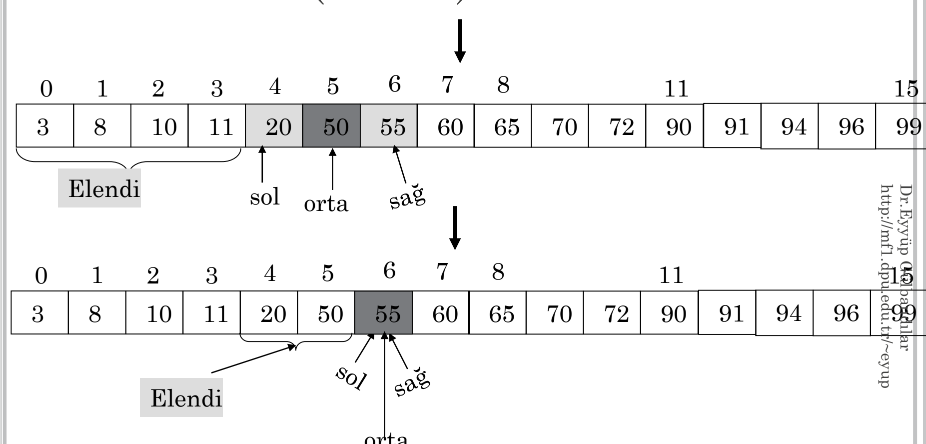
Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

İKİLİ ARAMA

- Dizi sıralanmış olduğundan, dizinin ortasında bulunan sayı ile aranan sayıyı karşılaştırarak arama boyutunu yarıya düşürülür ve bu şekilde devam edilir.
- Örnek: 55'i arayalım

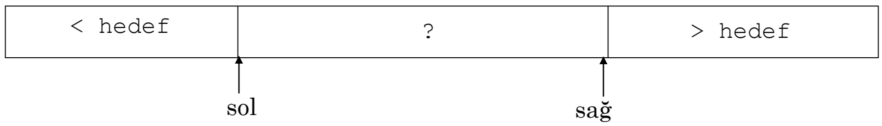


İKİLİ ARAMA (DEVAM)



- 55'i bulduk→ Başarılı arama
- 57'yi aradığımızda, bir sonraki işlemde başarısız bir şekilde sonlanacak.

İKİLİ ARAMA (DEVAM)



- Hedefi ararken herhangi bir aşamada, arama alanımızı "sağ" ile "sol" arasındaki alana kısıtlamış oluyoruz.
- "sol" 'un solunda kalan alan hedeften küçüktür ve bu alan arama alanından çıkarılır.
- "sağ" ın sagında kalan alan hedeften büyüktür ve bu alan arama alanından çıkarılır.

İKİLİ ARAMA - ALGORİTMA

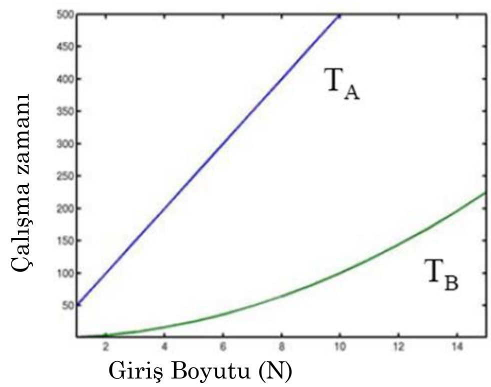
```
// Aranan sayının indeksini döndürür aranan sayı bulunamazsa -1
   döndürür.
int ikiliArama(int A[], int N, int sayi){
   int sol = 0,      int sag = N-1;
   while (sol <= sag){
         int orta = (sol+sag)/2;             // Test edilecek sayının indeksi
         if (A[orta] == sayi) return orta;   // Aranan sayı bulundu. İndeksi döndür
         else if (sayi < A[orta]) sag = orta – 1;      // Sağ tarafı ele
         else sol = orta+1;                            // Sol tarafı ele
   } //bitti-while
   return –1;   // Aranan sayı bulunamadı
} //bitti-ikiliArama
```

En kötü çalışma zamanı: $T(n) = 3 + 5*\log_2 N$????

Asİmptotİk Notasyon

- 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.

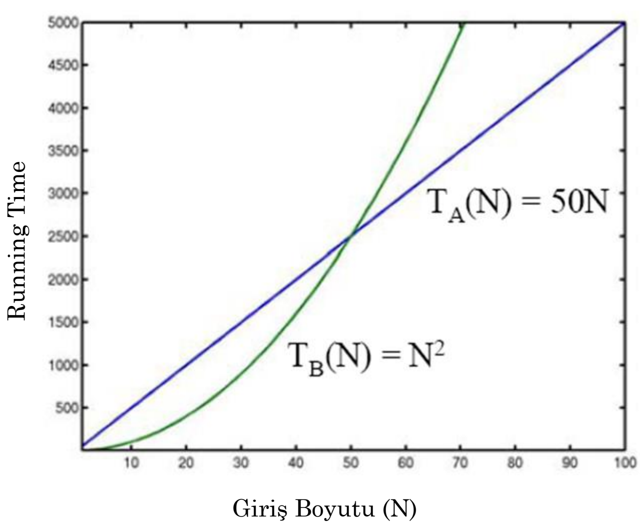
Hangi algoritma seçersiniz?



Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Asİmptotİk Notasyon

- N büyüdüğü zaman A ve B nin çalışma zamanı:



Şimdi hangi algoritmayı seçersiniz?

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

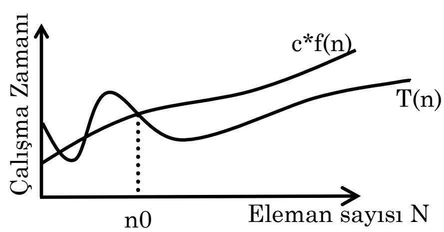
Asimptotik Notasyon

- Genel olarak, asimptotik notasyon, eleman sayısı n'nin sonsuza gitmesi durumunda algoritmanın, benzer işi yapan algoritmalarla karşılaştırmak için kullanılır.
- Eleman sayısının küçük olduğu durumlar pratikte mümkün olabilir fakat bu birçok uygulama için geçerli değildir.
- Verilen iki algoritmanın çalışma zamanını T1(N) ve T2(N) fonksiyonları şeklinde gösteriyoruz. Fakat hangisinin daha iyi olduğunu belirlemek için bir yol belirlememiz gerekiyor. (asimptotik olarak daha küçük gibi)
  - Asimptotik notasyonlar
  - Büyük-Oh, Ω, Θ notasyonları

Dr.Eyyüp Gülbandlar
http://mf1.dpu.edu.tr/~eyup

Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır

- T(n) = O(f(n))
  - c ve n_0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. n >= n_0 ifadesini sağlayan tüm değerler için T(n) <= c*f(n) dir.



- Örnek: T(n) = 50n → O(n). Neden?
  - c=50, n_0=1 seçersek. n>=1 için 50n <= 50n olur.
  - Başka uyan sayılarda mevcuttur.

Dr.Eyyüp Gülbandlar
http://mf1.dpu.edu.tr/~eyup

BÜYÜK-OH(BİG-OH) NOTASYONU: ASİMPTOTİK ÜST SINIR

- $T(n) = O(f(n))$
  - c ve n_0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. n >= n_0 ifadesini sağlayan tüm değerler için T(n) <= c*f(n) dir.
- Örnek: T(n) = 2n+5 ise O(n) Nedir?
  - n>=n_0 şartını sağlayan tüm sayılar için; T(n) = 2n+5 <= c*n
  - n>=1 için 2n+5 <= 2n+5n <= 7n
    - c = 7, n_0 = 1
  - n>=5 şartını sağlayan tüm sayılar için 2n+5 <= 3n
    - c = 3, n_0=5
  - Diğer c ve n_0 değerleri de bulunabilir.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

33

BÜYÜK-OH(BİG-OH) NOTASYONU: ASİMPTOTİK ÜST SINIR

- $T(n) = O(f(n))$
  - c ve n_0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. n >= n_0 ifadesini sağlayan tüm değerler için T(n) <= c*f(n) dir.
- Örnek: T(n) = 2n+5 ise $O(n^2)$ Nedir?
  - n>=n_0 şartını sağlayan tüm sayılar içi T(n) = 2n+5 <= c*n^2 şartını sağlayan c ve n0 değerlerini arıyoruz.
  - n>=4 için 2n+5 <= 1*n^2
    - c = 1, no = 4
  - n>=3 için 2n+5 <= 2*n^2
    - c = 2, no = 3
  - Diğer c ve n_0 değerleri de bulunabilir.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

34

Büyük-Oh(Big-Oh) Notasyonu: Asimptotik Üst Sınır

- $T(n) = O(f(n))$
  - c ve n_0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. $n >= n_0$ ifadesini sağlayan tüm değerler için $T(n) <= c*f(n)$ dir.
- Örnek: $T(n) = n(n+1)/2 \rightarrow O(?)$
  - $T(n) = n^2/2 + n/2 \rightarrow O(N^2)$. Neden?
  - $n >= 1$ iken $n^2/2 + n/2 <= n^2/2 + n^2/2 <= n^2$
  - Böylece, $T(n)=n*(n+1)/2 <= 1* n^2$ for all $n >=1$
    - c=1, no=1
  - Not: $T(n)$ ayrıca $O(n^3)$ tür.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

35

Karşılaşılan Genel Fonksiyonlar

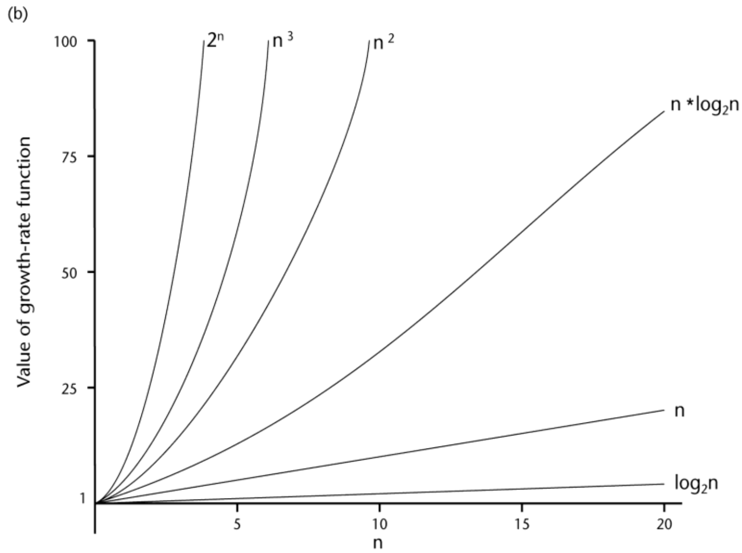
Maliyet artar

İsim	Büyük-Oh	Yorum
Sabit	O(1)	Yenilmez!
Log log	O(loglogN)	Tahminsel arama
Logaritmik	O(logN)	İyi hazırlanmış arama algoritmalarının tipik zamanı
Doğrusal	O(N)	Hızlı bir algoritmadır. N tane veriyi girmek için gereken zaman.
N logN	O(NlogN)	Çoğu sıralama algoritması
Karesel	$O(N^2)$	Veri miktarı az olduğu zamanlarda uygun (N<1000)
Kübik	$O(N^3)$	Veri miktarı az olduğu zamanlarda uygun (N<1000)
Üssel	$O(2^N)$	Veri miktarı çok az olduğunda uygun (n<=20)

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

36

KARŞILAŞILAN GENEL FONKSİYONLAR (DEVAM)



Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

37

MAKSİMUM ALT DİZİ TOPLAMI

- Tanım: Verilen bir tamsayı listesi içerisinde/dizisinde ***elemanları komşu olmak şartıyla*** hangi (bitişik) alt dizi en yüksek toplamı verir?
- Örneğin:
  - { -2,11,-4,13,-5,2 } → Cevap=20
  - { 1,2,-5,4,7,-2 } → Cevap=11
  - { 1,5,-3,4,-2,1 } → Cevap=7
- Bu problemi çözen çok sayıda algoritma vardır.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

38

Kaba Kuvvet Algoritması

```
public static int maxAltDiziT( int[] a){
    int maxTop = 0;
    for(int i=0; i<a.length; i++)
        for(int j=i; j<a.length; j++){
            int top=0;
            for(int k=i; k<=j; k++)
                top += a[k];
            if(top > maxTop){
                maxTop = top;
                int bas = i;     // alt dizinin başlangıcı
                int son = j;     // alt dizinin sonu
            }
        }
    return maxTop;
}
```

Bu algoritmanın karmaşıklığı nedir?

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Kaba Kuvvet Algoritması

```
public static int maxAltDiziT( int[] a){
    int maxTop = 0; .........................................1
    for(int i=0; i<a.length; i++) ...........................N
        for(int j=i; j<a.length; j++){.......................N*N
            int top=0; ......................................N*N
            for(int k=i; k<=j; k++) .........................N*N*N
                top += a[k]; ................................N*N*N
            if(top > maxTop){................................N*N*N
                maxTop = top; ...............................N*N*N
                int bas = i; // alt dizinin başlangıcı.N*N*N
                int son = j;     // alt dizinin sonu .......N*N*N
            }
        }
    return maxTop; ..........................................1
}
```

$6n^3+2n^2+n+2 \rightarrow O(n^3)$

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Geliştirilmiş Algoritma

```
public static int maxAltDiziT(int[] a) {
    int maxTop = 0;
    for (int i = 0; i < a.length; i++) {
        int top = 0;
        for (int j = i; j <= a.length; j++) {
            top += a[j];
            if (top > maxTop) {
                maxTop = top;
                int bas = i;    // alt dizinin başlangıcı
                int son = j;    // alt dizinin sonu
            }
        }
    }
    return maxTop;
}
```

Bu algoritmanın karmaşıklığı nedir?

Geliştirilmiş Algoritma

```
public static int maxAltDiziT(int[] a) {
    int maxTop = 0; .............................................1
    for (int i = 0; i < a.length; i++) {.........................N
        int top = 0; ............................................N
        for (int j = i; j <= a.length; j++) {..................N*N
            top += a[j]; .......................................N*N
            if (top > maxTop) {.................................N*N
                maxTop = top; ..................................N*N
                int bas = i; // alt dizinin başlangıcı...N*N
                int son = j; // alt dizinin sonu.........N*N
            }
        }
    }
    return maxTop; ..............................................1
}
```

$6n^2+2n+2 \rightarrow O(n^2)$

Doğrusal Algoritma

```
public static int maxAltDiziT(int[] a) {
    int maxTop = 0;
    int top = 0;
    for (int i=0, j=0; j<=a.length; j++) {
        top += a[j];
        if (top > maxTop) {
            maxTop = top;
            int bas = i;     // alt dizinin başlangıcı
            int son = j;     // alt dizinin sonu
        } else if (top<0){
            i = j + 1;
            top = 0;
        }
    }
    return maxTop;
}
```

Bu algoritmanın karmaşıklığı nedir?

$9n+3 \rightarrow O(n)$

Daha iyisi yapılabilir mi?

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Maksimum Alt Dizi Toplamı Çalışma Süresi

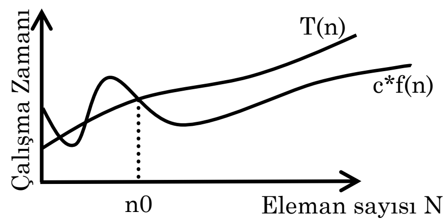
Çeşitli *Maksimum Alt Dizi Toplamı* algoritmaları için çalışma süreleri aşağıda verilmiştir. (saniye cinsinden)

N	O(N³)	O(N²)	O(N log N)	O(N)
10	0,000001	0,000000	0,000001	0,000000
100	0,000288	0,000019	0,000014	0,000005
1 000	0,223111	0,001630	0,000154	0,000053
10 000	218	0,133064	0,001630	0,000533
100 000	NA	13,17	0,017467	0,005571
1 000 000	NA	NA	0,185363	0,056338

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Ω NOTASYONU: ASİMPTOTİK ALT SINIR

- T(n) = Ω (f(n))
  - c ve n_0 şeklinde pozitif sabitlerimiz olduğunu düşünelim. n >= n_0 ifadesini sağlayan tüm değerler için T(n) >= c*f(n) dir.



Örnek: T(n) = 2n + 5 → Ω(n). Neden?

2n+5 >= 2n, tüm n >= 1 için

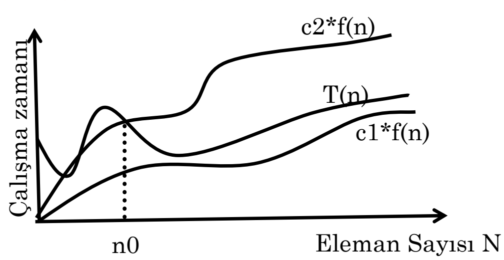
T(n) = 5*n² - 3*n → Ω(n²). Neden?

5*n² - 3*n >= 4*n², tüm n >= 4 için

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

Θ NOTASYONU: ASİMPTOTİK ALT VE ÜST SINIR

- T(n) = Θ (f(n))
  - c1,c2 ve n_0 şeklinde pozitif sabitlerimiz olduğunu düşünelim n >= n_0 ifadesini sağlayan tüm değerler için c1*f(n) <= T(n) <= c2*f(n) dir.



Örnek: T(n) = 2n + 5 → Θ(n). Neden?

2n <= 2n+5 <= 3n, tüm n >= 5 için

T(n) = 5*n² - 3*n → Θ(n²). Neden?

4*n² <= 5*n² - 3*n <= 5*n², tüm n >= 4 için

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

BÜYÜK-OH, THETA, OMEGA

İpucu:

- O(f(N)) düşünürsek f(N) ile "eşit veya küçük"
  - Üstten sınır: f(N) ile "yavaş veya aynı hızda büyür"
- Ω(f(N)) düşünürsek f(N) ile "eşit veya büyük"
  - Alttan sınır: f(N) ile "aynı hızda veya hızlı büyür"
- Θ(f(N)) düşünürsek f(N) ile "eşit"
  - Alttan ve Üsten sınır : büyüme oranları eşit
- (*N'nin büyük olduğu ve sabiterin elendiği durumlarda*)

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup

SIKÇA YAPILAN HATALAR

- Karmaşıklığı bulmak için sadece döngüleri saymakla yetinmeyin.
  - 2 içi içe döngünün 1 den N^2 kadar döndüğünü düşünürsek karmaşıklık $O(N^4)$ olur.
- $O(2N^2)$ veya $O(N^2+N)$ gibi ifadeler kullanmayın.
  - Sadece baskın terim kullanılır.
  - Öndeki sabitler kaldırılır.
- İç içe döngüler karmaşıklığı direk etkilerken art arda gelen döngüler karmaşıklığı etkilemez.

Dr.Eyyüp Gülbandılar
http://mf1.dpu.edu.tr/~eyup