# Algoritmaların Karşılaştırılması

Doç. Dr. Aybars UĞUR

# Giriş

Bir programın performansı genel olarak programın işletimi için gerekli olan bilgisayar zamanı ve belleğidir. Bir programın zaman karmaşıklığı (time complexity) programın işletim süresidir. Bir programın yer karmaşıklığı (space complexity) programın işletildiği sürece gerekli olan yer miktarıdır. Bir problemin çözümünde, kullanılabilecek olan algoritmalardan en etkin olanı seçilmelidir. En kısa sürede çözüme ulaşan veya en az işlem yapan algoritma tercih edilmelidir. Burada bilgisayarın yaptığı iş önemlidir. Bazı durumlarda da en az bellek harcayan algoritmanın tercih edilmesi gerekebilir. Ayrıca, programcının yaptığı iş açısından veya algoritmaların anlaşılırlıkları bakımından da algoritmalar karşılaştırılabilir. Daha kısa sürede biten bir algoritma yazmak için daha çok kod yazmak veya daha çok bellek kullanmak gerekebilir (trade-off).

# Algoritmaların Karşılaştırılması - I

- Rakip algoritmaları yaptıkları iş açısından karşılaştırmak için her algoritmaya uygulanabilecek somut ölçüler tanımlanmalıdır. Aynı işi yapan algoritmalardan daha az işlemde sonuca ulaşanın (hızlı olanın) belirlenmesi yani daha genel olarak algoritma analizi teorik bilgisayar bilimlerinin önemli bir alanıdır.

- Yazılımcılar, iki farklı algoritmanın yaptıkları işi nasıl ölçüp karşılaştırırlar? İlk çözüm algoritmaları bir programlama dilinde kodlayıp her iki programı da çalıştırarak işletim sürelerini karşılaştırmaktır. İşletim süresi kısa olan daha iyi bir algoritma denilebilir mi? Bu yöntemde işletim süreleri belirli bir bilgisayara özeldir. Dolayısı ile işletim süresi de bu bilgisayara bağlıdır. Daha genel bir ölçüm yapabilmek için olası tüm bilgisayarlar üzerinde algoritmanın çalıştırılması gerekir.

# Algoritmaların Karşılaştırılması - II

- İkinci çözüm, işletilen komut ve deyimlerin sayısını bulmaktır. Fakat bu ölçüm kullanılan programlama diline göre ve programcıların stiline göre değişim gösterir. Bunun yerine algoritmadaki kritik geçişlerin sayısı hesaplanabilir. Her tekrar için sabit bir iş yapılıyor ve sabit bir süre geçiyorsa, bu ölçü anlamlı hale gelir.

- Buradan, algoritmanın temelinde yatan bir işlemi ayırarak, bu işlemin kaç kere tekrarlandığını bulma düşüncesi doğmuştur. Örnek olarak bir tamsayı dizisindeki tüm elemanların toplamını hesaplama işleminde gerekli olan iş miktarını ölçmek için tamsayı toplama işlemlerinin sayısı bulunabilir. 100 elemanlı bir dizideki elemanların toplamını bulmak için 99 toplama işlemi yapmak gerekir. n elemanlı bir listedeki elemanların toplamını bulmak için n-1 toplama işlemi yapmak gerekir diye genelleştirme yapabiliriz. Böylece algoritmaları karşılaştırırken belirli bir dizi boyutu ile sınırlı kalınmaz.

# Algoritmaların Karşılaştırılması -III

- İki gerçel matrisin çarpımında kullanılan algoritmaların karşılaştırılması istendiğinde, matris çarpımı için gereken gerçel sayı çarpma ve toplama işlemlerinin karışımı bir ölçü olacaktır. Bu örnekten ilginç bir sonuca ulaşılır: Bazı işlemlerin ağırlığı diğerlerine göre fazladır. Birçok bilgisayarda bilgisayar zamanı cinsinden gerçel sayı çarpımı gerçel sayı toplamından çok daha uzun sürer. Dolayısı ile tüm matris çarpımı düşünüldüğünde toplama işlemlerinin etkinlik üzerindeki etkisi az olacağından ihmal edilebilirler. Sadece çarpma işlemlerinin sayısı dikkate alınabilir. Algoritma analizinde genelde algoritmada egemen olan bir işlem bulunur ve bu diğerlerini gürültü (noise) düzeyine indirger.

# Algoritmalarda Karmaşıklık ve Zaman Karmaşıklığı Analizi

**İşletim Zamanı (Running Time)**

İşletim zamanını girdi boyutunun bir fonksiyonu olarak ele almak tüm geçerli girdileri tek değere indirir. Bu da değişik algoritmaları karşılaştırmayı kolaylaştırır. En yaygın karmaşıklık ölçüleri "Worst –Case Running Time" (en kötü durum işletim süresi) ve "Average-Case Running Time" (ortalama durum işletim süresi)'dir.

# İşletim Zamanı

**Worst-Case Running Time :**

Bu işletim süresi, her girdi boyutundaki herhangi bir girdi için en uzun işletim süresini tanımlar. Örnek olarak bir programın en kötü ihtimalle ne kadar süreceğinin tahmin edilmesi istenen bir durumdur. n elemanlı bir listede sıralı arama en kötü ihtimalle (aranan bulunamazsa) n karşılaştırma gerektirecektir. Yani worst-case running time (işletim zamanı) T(n) = n'dir. Tüm problemlerde sadece en kötü girdi dikkate alındığı için worst-case running time değerini hesaplamak göreceli olarak kolaydır.

**Average-Case Running Time :**

Bu işletim süresi, her girdi boyutundaki tüm girdilerin ortalamasıdır. n elemanın her birinin aranma olasılığının eşit olduğu varsayıldığında ve liste dışından bir eleman aranmayacağı varsayıldığında ortalama işletim süresi (n+1)/2'dir. İkinci varsayım kaldırıldığında ortalama işletim süresi [(n+1)/2,n] aralığındadır (aranan elemanların listede olma eğilimine bağlı olarak). Ortalama durum analizi basit varsayımlar yapıldığında bile zordur ve varsayımlar da gerçek performansın iyi tahminlenememesine neden olabilir.

# Asimptotik Analiz

Algoritmaların karşılaştırılmasında asimptotik etkinlikleri de dikkate alınabilir. Girdi boyutu sonsuza yaklaşırken işletim süresinin artışı. Asimptotik gösterimin elemanı olan 4 önemli gösterim vardır : O-notation, o-notation, Ω-notation, θ-notation. Burada sadece O gösterimi üzerinde durulacaktır. O gösterimi, fonksiyonların artış oranının üst sınırını belirler. O(f(n)), f(n) fonksiyonundan daha hızlı artmayan fonksiyonlar kümesini gösterir.

**Big-O Gösterimi (notasyonu)**

n elemanlı bir listedeki elemanların toplamını bulmak için n-1 toplama işlemi yapmak gerekir diye genelleştirme yapmıştık. Yapılan işi, girdi boyutunun bir fonksiyonu olarak ele almış olduk. Bu fonksiyon yaklaşımını matematiksel gösterim kullanarak ifade edebiliriz : Big-O (O harfi, 0 sayısı değil) gösterimi veya büyüklük derecesi (order of magnitude). Büyüklük derecesini problemin boyutuna bağlı olarak fonksiyonda en hızlı artış gösteren terim belirler.

# Big-O Örneği

Örnek olarak :

$f(n) = n^4 + 100n^2 + 10n + 50 = O(n^4)$

fonksiyonunda n'in derecesi $n^4$'tür yani n'in büyük değerleri için fonksiyonu en fazla n4 etkiler. Peki daha düşük dereceli deyimlere ne olmaktadır? n'in çok büyük değerleri için $n^4$, $100n^2$'den, 10n'den ve 50'den çok büyük olacağından daha düşük dereceli terimler dikkate alınmayabilir. Bu diğer terimlerin, işlem süresini etkilemedikleri anlamına gelmez; bu yaklaşım yapıldığında n'in çok büyük değerlerinde önem taşımadıkları anlamına gelir.

n, problemin boyutudur. Yığıt, liste, kuyruk, ağaç gibi veri yapılarında eleman sayılarıdır. n elemanlı bir dizi gibi ...

# Bir Listenin Bir Dosyaya Yazılması

Bir listedeki tüm elemanların dosyaya yazılması için ne kadar iş yapılır :
Cevap, listedeki eleman sayısına bağlıdır.

**Algoritma**

OPEN (Rewrite) the file

WHILE more elements in list DO

Print the next element

İşlemi yapmak için geçen süre :

(n*(Bir elemanın dosyaya yazılması için geçen süre))+dosyanın açılması sırasında geçen süre

Algoritma O(n)'dir (Algoritmanın zaman karmaşıklığı O(n)'dir) . Çünkü, n tane işlem + sadece dosya açılması işlemi vardır. Yüzlerce elemanın dosyaya kaydedildiği düşünülürse, dosya açılması sırasında geçen süre miktarı rahatlıkla ihmal edilebilir. Ama az sayıda eleman varsa dosya açılması sırasında geçen süre miktarı önem taşıyabilir ve toplam süreye katılımı daha fazla olur.

# Benchmark Testi

Bir algoritmanın büyüklük derecesi, bilgisayarda işletildiğinde sonucun ne kadar sürede alınacağını belirtmez. Bazen de bu tür bir bilgiye gereksinim duyulur. Örnek olarak bir kelime işlemcinin 50 sayfalık bir yazı üzerinde yazım denetimi yapma süresinin birkaç saniye düzeyinden fazla olmaması istenir. Böyle bir bilgi istendiğinde, Big-O analizi yerine diğer ölçümler kullanılmalıdır. Program değişik yöntemlere göre kodlanır ve karşılaştırma yapılır. Programın çalıştırılmasından önce ve sonra bilgisayarın saati kaydedilir. İki saat arasındaki fark alınarak geçen süre bulunur. Bu tür bir "Benchmark" testi, işlemlerin belirli bir bilgisayarda belirli bir işlemci ve belirli kaynaklar kullanılarak ne kadar sürdüğünü gösterir.

# Big-O ve Program Boyutu

Bilgisayarın yaptığı işin programın boyutu ile, örnek olarak satır sayısı ile ilgili olması gerekmez. N elemanlı bir diziyi 0'layan iki program da O(n) olduğu halde kaynak kodlarının satır sayıları oldukça farklıdır :

Program 1 :

```
dizi[0] = 0;
dizi[1] = 0;
dizi[2] = 0;
dizi[3] = 0;
.....
dizi[n-1] = 0;
```

Program 2 :

```
for(int i=0; i<n; ++i)
                dizi[i] = 0;
```

# N’e kadar olan sayıların toplamında Big-O

1’den n’e kadar olan sayıların toplamını hesaplayan iki kısa programı düşünelim :

Program 1 :

```
toplam = 0;
for(int i=0; i<n; ++i)
        toplam = toplam + i;
```

Program 2 :

```
toplam = n * (n+1) / 2;
```

Program 1, O(n)’dir. n=50 olursa programın çalışması sırasında n=5 için harcanan sürenin yaklaşık 10 katı süre harcanacaktır. Program 2 ise O(1)’dir. n=1 de olsa n=50’de olsa program aynı sürede biter.

# Artış Oranı Fonksiyonları

**O(1) :** Sabit zaman

Örnek : n elemanlı bir dizinin i. elemanına bir değer atanması O(1)'dir. Çünkü bir elemana indisinden doğrudan erişilmektedir.

**O(n) :** Doğrusal zaman

Örnek : n elemanlı bir dizinin tüm elemanlarının ekrana yazdırılması O(n)'dir.

Örnek : sıralı olmayan bir dizideki (listedeki) elemanlardan birinin aranması O(n)'dir (en kötü durumda da, ortalama durumda da).

**$O(\log_2 n)$ :** O(1)'den fazla O(n)'den azdır.

Örnek : Sıralı bir listenin elemanları içinde ikili arama (binary search) uygulanarak belirli bir değerin aranması $O(\log_2 n)$'dir.

**$O(n^2)$ :** İkinci dereceli zaman

Örnek : Basit sıralama algoritmalarının birçoğu (selection sort gibi) $O(n^2)$'dir.

**$O(n \log_2 n)$ :** Bazı hızlı sıralama algoritmaları $O(n \log_2 n)$'dir.

**$O(n^3)$ :** Kübik zaman

Örnek : Üç boyutlu bir tamsayı tablosundaki her elemanın değerini artıran algoritma.

**$O(2^n)$ :** Üstel zaman, çok büyük değerlere ulaşır.

# Örnek süre hesaplaması

- Bir programın işletimi $n^3$ adım sürüyorsa, ve n=1000 ise, program $1000^3$ adım sürecek demektir. Yani 1 000 000 000 (bir milyar) adım.

- Kullanılan bilgisayar saniyede 1 000 000 000 adımı gerçekleştirebilecek kadar hızlı ise bu program tam 1 saniye sürecektir.

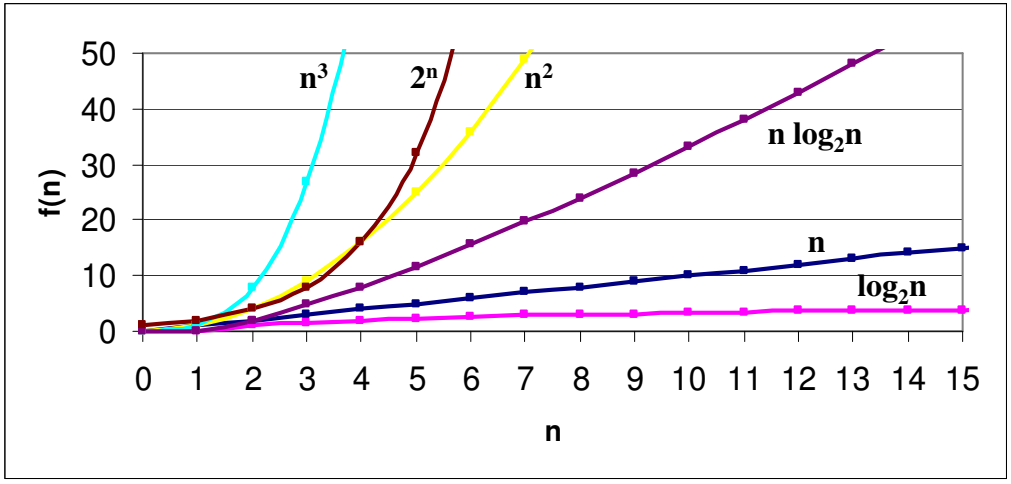
# Artış Oranı Fonksiyonları

| Fonksiyon | İsim |
|---|---|
| 1 | constant |
| log n | logarithmic |
| n | linear |
| n log n | n log n |
| $n^2$ | quadratic |
| $n^3$ | cubic |
| $2^n$ | exponential |
| n! | factorial |

Yaygın Artış Oranı Fonksiyonları

| log n | n | n log n | $n^2$ | $n^3$ | $2^n$ |
|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 2 |
| 1 | 2 | 2 | 4 | 8 | 4 |
| 2 | 4 | 8 | 16 | 84 | 16 |
| 3 | 8 | 24 | 64 | 512 | 256 |
| 4 | 16 | 64 | 256 | 4096 | 65536 |
| 5 | 32 | 160 | 1024 | 32768 | 4294967296 |

Artış Oranı Fonksiyonlarının Aldıkları Değerler




Artış Oranı Fonksiyonlarının Grafikleri : n, f(n)