**Veri Yapıları ve Algoritmalar**

Veri yapıları ile algoritmalar konusu birbirleri ile yakın biçimde ilişkilidir. Bu nedenle bu iki konu genellikle birlikte ele alınmaktadır.

**Algoritma Nedir?**

Bir problemi çözüme götüren adımlar topluluğuna algoritma (algorithm) denilmektedir. Sözcük etimolojik bakımdan cebirin kurucusu El-Harizmi'nin isminden türetilmiştir. Algoritma problemi kesin çözüme götürür. Ancak bazı problemlerin kesin çözümü çok uzun zaman alabilmektedir. Bunlar için kısa sürede iyi bir çözüm ile yetinilebilir. İşte problemin kesin çözümünü bulmayan ama iyi bir çözüm vaat eden adımlar topluluğuna "sezgisel yöntemler (heuristic)" denilmektedir.

Algoritmalar nasıl ifade edilirler? Bunun için çok formel bir yöntem yoktur. Bazen akış diyagramları, sahte kodlar (pseudo codes) kullanılabilir. Hemen her zaman sözel bir anlatım buna eşlik eder. Algoritma anlatmak için özel diller tasarlanmış olsa da bunlar yaygın bir kullanım bulamamıştır. Algoritmayı anlatmak için en çok kullanılan yöntem popüler bir dille algoritmayı gerçekleştirmektir. Eskiden bu amaçla C yaygın kullanılırdı. Şimdi C'nin yanı sıra C++, Java, C#, Python gibi diller de bu amaçla kullanılmaktadır.

Pekiyi algoritma dilden bağımsız mıdır? Algoritmaların çoğu dilden bağımsız olarak ifade edilebilir. Ancak ince noktalara gelindiğinde algoritmalar da dile bağımlı hale gelebilmektedir. Örneğin bir dilde belli işi daha kolay yapan deyimler olabilir. Bu durumda algoritma o deyimlerle gerçekleştirilebilir. Ya da örneğin fonksiyonel dillerde algoritmalar özyineleme içerecek biçimde tasarlanmaktadır.

Bazı problemlerin çözümü için tek bir yol bulunuyor olabilir. Fakat bazıları için birden fazla algoritma söz konusu olabilir. Bu durumda bu algoritmaları kıyaslamak isteyebiliriz. Kıyaslamak için ise ölçütler gerekir. Algoritmaları kıyaslamak için iki önemli ölçüt vardır: Hız ve kaynak kullanımı. Ancak baskın ölçüt hızdır. Bu nedenle algoritmaları karşılaştırmak denildiğinde default olarak hız ölçütüne göre karşılaştırma anlaşılır. Algoritmaların masaya yatırılıp karşılaştırılması sürecine "algoritma analizi (analyisis of algorithms)" denilmektedir.

**Algoritma Analizi**

İki algoritmanın hızı nasıl ölçülebilir? İlk akla gelen yöntem simülasyon yöntemidir. Bu yöntemde algoritmalar çalıştırılıp aldığı zamanlar ölçülür. Tabii algoritmaların çoğu birtakım girdiler üzerinde işlem yapmaktadır (örneğin bir dizi üzerinde). Bu durumda o girdilerin dağılımı da önemlidir. Eğer biz simülasyonu bir kez yaparsak yanlış sonuçlar elde edebiliriz. Rastgele girdiler üzerinde çok sayıda denemenin bir ortalaması ancak bir fikir verebilir.

Simülasyon yöntemi matematiksel bir yöntem değildir. Bu nedenle bize önemli bir bilgi vermez. Algoritmaları cebirsel olarak ölçmek daha sağlam bir zemin oluşturabilmektedir. Algoritmaları analiz ederken kullanılabilecek sağlam yöntemlerden biri algoritmadaki çözüm için toplamda kaç işlemin gerektiğine bakmaktır. (Tabii tüm işlemlerin makine zamanları aynı değildir fakat bu durum ihmal edilebilir.) Örneğin iki sort algoritmasını karşılaştıracak olalım. İkisinde de sort işlemi için kaç işlem gerektiğini hesaplamaya çalışabiliriz. Tabii algoritmaların girdi parametreleri vardır. Sort işleminde dizinin uzunluğu olan N bir girdi parametresidir. Algoritma için gereken işlemlerin sayısı bu N ile ilgili olacaktır.

Algoritmadaki işlemlerin sayısı bulunabilir mi? Ya algoritmada karışık if deyimleri varsa ne olacaktır? Biz akışın nasıl olacağını giridiyi bilmeden anlayamayız. Pekiyi bu durumda işlemlerin sayısı nasıl hesaplanacaktır? Örneğin aşağıdaki en büyük sayıyı bulma algorimasına bakalım:

int [] a = new int[N]{...};

int max, i;

max = a[0];

for (i = 1; i < N; +i)

if (max < a[i])

max = a[i];

Burada toplam işlemin sayısı max = a[i] işlemine bağlıdır. Pekiyi bu işlemden kaç tane olacağını diziyi görmeden bilebilir miyiz? Yanıt tabii ki hayır. İşte algoritma analizinde işlemlerin sayısını hesaplarken üç durum dikkate alınabilmektedir:

1) Ortalama durum (average case condition)

2) En kötü durum (worst case condition)

3) En iyi durum (best case condition)

En kötü durum olabileceklerin en kötüsüdür. Yukarıdaki örnekte en kötü durumda max = a[i] işlemi N - 1 kez yapılır. Ortalama durum tüm olasılıkların ortalamasını temsil eder. Yukarıdaki örnekte ortalama olarak max = a[i] işlemi (N - 1) / 2 kez yapılmaktadır. En iyi durum da olabileceklerin en iyisini temsil eder. Yukarı örnekte en iyi durumda max = a[i] işlemi hiç yapılmaz.

Tabii analizde en iyi durumun çok yararı yoktur. Çünkü aşırı iyimserlik programlamada çok değerli değildir. En kötü durum senaryosu önemlidir. Çünkü bazı durumlarda en kötü duruma bile hazırlıklı olmak gerekebilir. Şüphesiz en önemli durum ortalama durumdur. Ortalama durum algoritmanın karakterini en iyi yansıtan durumdur. O halde bu üç durumun önemi şöyle sıralanabilir: Ortalama durum, en kötü durum, en iyi durum. Genellikle algoritmalar ele alınırken hem ortalama durum hem de en kötü durum analizi yapılır.

Algoritmaların işlem sayısına ilişkin analizler de sanıldığı kadar kolay değildir. Çünkü özellikle ortalama durumu hesaplamak bazen çok karmaşık olabilmektedir. Algoritmaları pratik bakımdan kıyaslamak için asimtotik notasyonlardan faydalanılmaktadır. Bunların en yaygın kullanılanı Big O notasyonudur.

**Big O Notasyonu**

Big O notasyonu algoritmaları karmaşıklıklarına göre kategorilere (sınıflara) ayırır. Kategoriler arasında da iyilik kötülük ilişkisi vardır. Yani Big O notasyonunda karmaşıklıktaki algoritmaları aynı kategoriye (sınıfa) içerisine yerleştirilir. Bu kategoriler da birbirlerine göre iyi ya da kötü olabilmektedir. Aynı kategoride olan algoritmalar aralarında farklılıklar olsa bile aynı düzeyde iyi kabul edilmektedir.

Big O notasyonu kaba bir kıyaslama biçimidir. Eğer algoritmalar ayrıntılı bir biçimde kıyaslanacaksa yukarıda belirtilen işlem sayıları hesaplanmalıdır. Big O notasyonunda belli karakterdeki algoritmalar aynı kategoridedir. Adeta bunlar arasında fark yokmuş gibi davranılır. Eğer algoritma birden fazla kategoriye giriyorsa en kötü kategori onun gerçek kategorisini belirtir. İyiden kötüye doğru Big O kategorileri şöyledir:

**O(1):** Bu kategoriye "sabit karmaşıklık" da denilmektedir. Eğer algoritmada hiç döngü yoksa her şey tekil işlemlerle yapılıyorsa algoritma O(1) karmaşıklıktadır. (Örneğin üçgenin alanının bulunması gibi). Tekil işlemlerin sayısı önemli değildir.

**O(log N):** Bu karmaşıklıkta N uzunluğunda bir girdi kümesi (örneğin bir dizi) vardır. Algoritmada açık ya da özyinelemeli olarak bir döngü bulunur. Ancak bu döngü N kadar değil log N kadar dönmektedir. (log default olarak 2 tabanına ilişkin düşünülmelidir.). Örneğin "ikili arama (binary search)" algoritması O(log N) karmaşıklıktadır. (Tabi yukarıda da belirtildiği gibi başka tekil işlemler de algoritmada bulunabilir.) Bu karmaşıklık kategorisine "logaritmik karmaşıklık" da denilmektedir.

**O(N):** Bu kategoriye "doğrusal karmaşıklık" da denilmektedir. Bu tür algoritmalarda iç içe olmayan tekil birden fazla döngü bulunabilir. Girdi uzunluğu N ise bu döngüler N ile bağlantılı olarak (tipik olarak N defa ya da N / 2 defa ya da N - 5 defa vs.) dönerler. Fakat dönüş logaritmik değildir. N ile oransal olabilir (Örneğin 2N, 5N, N/2 gibi). Örneğin en büyük sayıyı bulma algoritması, doğrusal arama işlemi, bir dizinin arasına bir elemanın eklenmesi işlemi doğrusal karmaşıklıktadır.

**O(N \* log N):** Bu tür algoritmalarda iç içe iki döngü vardır. Ancak döngülerden biri N ile orantılı biçimde dönerken diğeri log N kadar döner. Örneğin "quick sort" algoritması O(N log N) karmaşıklıktadır.

**O(N2):** Bu kategoriye "karesel karmaşıklık" da denilmektedir. Bu tür algoritmalarda iç içe iki döngü vardır (tabii başka tekil döngüler de olabilir.) Döngülerin ikisi de N ile orantılı biçimde dönerler. Örneğin "kabarcık sıralaması (bubble sort)", "seçerek sıralama (selection sort)" algoritmaları karesel karmaşıklıktadır.

**O(N3):** Bu kategoriye "küpsel karmaşıklık" da denilmektedir. Burada iç içe N ile orantılı 3 döngü vardır. Örneğin matris çarpımı böyle bir döngüsel yapı gerektirmektedir.

**O(NK):** Bu kategoriye K'sal karmaşıklık denir. İç içe k tane N ile orantılı döngü bulunmaktadır.

Yukarıdaki tüm kategorilere "polinomsal karmaşıklık" da denilmektedir. Polinomsal karmaşıklıktaki problemlerin bugünkü bilgisayarlarla makul zamanda çözülmesi mümkündür.

**O(KN):** Bu kategoriye üstel karmaşıklık (exponential complexity) de denilmektedir. Bu kategorideki algoritmalardaki işlem sayısı N'e göre çok hızlı artmaktadır. Bugünkü bilgisayarlarla bile üstel karmaşıklığa sahip algoritmaların kesin çözümleri çok fazla zaman alabilir. Bu nedenle bunların bilgisayar çözümleri mümkün olmayabilir. Bu tür algoritmalarda sezgisel yöntemler (heuristic) önem kazanmaktadır. Örneğin N elemanlı bir kümenin alt kümelerinin sayısı 2N tanedir. Bir kümenin tüm alt kümelerine bakarak işlem yapan algoritmalar (böyle çok algoritma vardır) üstel karmaşıklıktadır.

**O(N!):** Bu karmaşıklığa faktöryel karmaşıklığı denilmektedir. Bu en kötü karmaşıklık grubudur. Örneğin gezgin satıcı probleminde satıcı bir merkezden (düğümden) çıkarak tüm şehirleri (düğümleri) dolaşıp tekrar başlangıç yerine geri gelir. Amaç en kısa turu atacak çözümün bulunmasıdır. Bu problemde N tane düğüm varsa (N - 1)! / 2 tane rotayı hesaplamak gerekir. O halde gezgin satıcı problemi faktöryel karmaşıklıkta bir algoritmadır. İş sıralama çizelgeleme (sequencing and scheduling) problemleri de genel olarak bu karmaşıklıktadır.

Üstel ve faktöryel karmaşıklıklara "polinomsal olmayan (Non-Polynomial ya da kısaca NP)" karmaşıklıkta problem denilmektedir. İşte bu tür problemler hala üzerinde en çok çalışılan ve özel yöntem bulunmaya çalışılan problemlerdir.

İşte algoritmaları karşılaştırırken onların Big O kategorileri belirlenir. Sonra hangi kategori değerinden daha iyiyse o algoritmanın kategorik olarak diğerinden daha iyi olduğu söylenir. Tabii bu yöntemde aynı kategoriye sahip algoritmalar sanki aynı etkinlikteymiş gibi değerlendirilmektedir. Big O gibi asimtotik notasyonlar hızlı bir biçimde genel bir fikir edinmek için kullanılmaktadır. Algoritmanın ayrıntılı analizinde "ortalama" ve "en kötü" işlem zamanları belirtilmelidir. Örneğin bir algoritma iç içe iki döngü içeriyorsa ve otuz kadar tekil döngü içeriyorsa bu algoritma karesel karmaşıklıktadır. Kategorik olarak bu algoritmanın iç içe iki döngü içeren ve on tekil döngü içeren algoritmadan bir farkı yoktur. Fakat ayrıntılı analizde bunun diğerinden daha iyi olduğu sonucu çıkartılabilir.

Aslında algoritmalar dünyasında çoğu kez bir algoritmanın mutlak iyi olduğunu söyleyemeyiz. Algoritmanın karmaşıklık kategorisi diğerinden yüksek olsa bile ya da ortalama ve en kötü olasılıktaki işlem sayısı diğerinden kötü olsa bile bazı durumlarda diğerinden hızlı çalışabilmektedir. Örneğin "quick sort" algoritması kategorik olarak O(N log N) karmaşıklıktadır. Bubble sort algoritması ise O(N2) karmaşıklatadır. Fakat elemanların çoğu sıralı az sayıda elemanın sırasının bozuk olduğu dizilimlerde "bubble sort" algoritması "quick sort" algoritmasından hızlı çalışabilmektedir. O halde biz sırasının çok az bozuk olduğunu bildiğimiz olguları sıraya dizmek istiyorsak burada "bubble sort" algoritmasını tercih edebiliriz. O halde aslında karmaşıklığı kötü olan algoritmaların bazı özel durumlarda daha etkin olabilmektedir. Biz de sistemi değerlendirip ona göre hangi algoritmanın kullanılacağına karar veririz.

**Algoritmaların Sınıflandırılması**

Algoritmalar çeşitli ölçütlere göre sınıflandırılabilmektedir. Tipik kullanılan sınıflandırma ölçütleri şunlardır:

**Gerçekleştirim (Implementation) Biçimlerine Göre Sınıflandırma:** Bu sınıflandırmada algoritmalar onların kodlarının nasıl yazıldığına göre sınıflandırılmaktadır. Tipik alt sınıflar şunlardır:

- Özyinelemeli Algoritmalar

- Mantıksal Algoritmalar

- Seri Algoritmalar

- Paralel ya da Dağıtık (Distributed) Algoritmalar

- Deterministik ya da Deterministik Olmayan (Probablistic, Stochastic) Algoritmalar

- Quantum Bilgisayarlarına Yönelik Algoritmalar

**Tasarım Biçimine Göre Sınıflandırma:** Bu sınıflandırma algoritmaları genel tasarımına göre sınıflandırmaktadır. Tipik alt sınıflar şunlardır:

- Böl ve Ele Geçir (Divide and Conquer) Algoritmaları

- Rastgele İşlemlerin Uygulandığı Algoritmalar

- Karmaşıklığın Azaltılması İle Çözüm Bulunmaya Çalışılan Algoritmalar

- Sürekli İyileştirme Yapılarak Geri Dönüşlü (Backtracking) Algoritmalar

**Optimizasyon Tekniğine Göre Sınıflandırma:** Bu sınıflandırma genellikle algoritmaların bir en iyi değer bulma çabasına göre yapılmaktadır. Tipik alt sınıfları şunlardır:

- Doğrusal Programlama

- Dinamik Programlama

- Greedy Algoritmaları

- Sezgisel (Heuristic) Yöntemler

**Karmaşıklığa Göre Sınıflandırma:** Bu sınıflandırmada algoritmanın karmaşıklığına bakılır.Zaten bu sınıflşandırma yukarıda ele alınmıştı.

- O(1) Karmaşıklıktaki Algoritmalar

- O(log N) Karmaşıklıktaki Algoritmalar

- O(N) Karmaşıklıktaki Algoritmalar

- O(N log N) Karmaşıklıktaki Algoritmalar

- O(N2) Karmaşıklıktaki Algoritmalar

- O(N3) Karmaşıklıktaki Algoritmalar

- O(Nk) Karmaşıklıktaki Algoritmalar

- Üstel Karmaşıklıktaki Algoritmalar (O(aN))

- Faktöryel Karmaşıklıktaki Algoritmalar (O(N!))

Donald Knuth "The Art Of Computing Programming" kitap serisinde algoritmaları ayrı kitaplar biçiminde şöyle sınıflandırmıştır:

- Temel Algoritmalar (Fundamental Algorithms): Bunlar günlük programalamada karşılaşılan temel algoritmalardır.

- Yarı Nümerik Algoritmalar (Seminumerical Algorithms): Bunlar nümerik tarafı da olan ama programalama dünyasında karşılaşılaşılabilen algoritmalardır.

- Arama ve Sıraya Dizme Algoritmaları (Sorting and Searching Algorithms): Bunlar arama ve sıraya dizme işlemlerine ilişkin algoritmalardır.

- Graf Algoritmaları (Graph Algorithms): Graflar üzerinde dolaşım ve optimizasyon içeren algortimalardır.

- Optimizasyon Algoritmaları (Optimization Algoritms): Matematiksel en iyi değerleri bulmaya çalışan algoritmalardır.

- Nümerik Algoritmalar (Numerical Algoritms): Sayısal olarak kök bulma, denklem çözme, türev, integral gibi nümerik işlemleri konu edinen algoritmalardır.