Veri Yapıları ve Algoritmalar

1.1 Veri Yapısı

Verinin bilgisayar belleğinde saklanma şekli ve organizasyonuna veri yapısı denir. Veri tipleri, verinin türüne göre farklılık gösterir. İşaretli olmayan bir pozitif sayı doğrudan ikili yapıda tutulurken, işaretli sayı ikinin tümleyeni şeklinde tutulur.

Veriye farklı ve hızlı erişim ihtiyacı, verinin bilgisayar belleğinde tutulması için farklı organizasyonlar oluşturulması oluşturulmasını doğurmuştur. Örneğin belirli bir miktardaki tam sayı bilgisayar belleğinde sırasız, sıralı (küçükten büyüğe gibi) veya bir ağaç yapısında tutulabilir. Dikkat edilirse burada tam sayı verilerin tutulması için üç farklı organize düşünülmüştür.

Verinin bellekte kapladığı alan, erişim şekli ve hızı kavramları organizasyonun yapısının farklılaşmasını, uygun organizasyon tasarımı yapılmasını gerekli kılar. Verilere özel olarak, belirli veriler ayrı ayrı ulaşma ihtiyacı yoksa verilerin sıralı bir biçimde tutulmasına ihtiyaç yoktur. Fakat eğer bir arama yapılacaksa bu durumda verilerin sıralı olması bir avantaj sağlayacaktır. Eğer verilerin eklenmesi ve çıkarılması fazlaca yapılıyorsa ve sınır belirli değilse bu durumda ağaç yapısının kullanılması ayrıca bir kolaylık getirecektir.

Verilerin bilgisayar belleğinde tutulması için organizasyon tasarımında belirli temel düşünceler yer alır. Bellekte fazla yer kaplamayacak şekilde en uygun yapıda tutulması gereklidir. Ayrıca verilerin oluşturulması, eklenmesi, çıkarılması ve ulaşım şekli konusunda kolay ve etkin algoritmalar sunması gereklidir.

Verilerin bilgisayar belleğinde tutulması için yapılacak tasarımda amaca göre farklılık gösterebilir. Bellek boyutunun artmaması öncelik ise hızdan, hız ve esneklik söz konusu ise bellekten feragat edilebilir. Hangi organizasyon yapısının kullanılacağı tamamen yapılacak uygulamaya bağlıdır. Doğrudan ve kesin olarak cevap verilemez.

1.2 Algoritma

Bir algoritma, bir işi yapmak veya bir problemi çözmek için gerekli sıralı adımlar veya komutlar dizisi olarak tanımlanabilir. Algoritma işlenmemiş bir giriş verisine karşılık belirli bir çıkış bilgisi oluştururlar. Bir algoritmayı oluşturan komutlar şu kurallar çerçevesinde olmalıdır:

- Yazılan komut herkes tarafından aynı şekilde değerlendirilmeli ve tek bir anlama gelmelidir.
- Komut yapılabilir, uygulanabilir olmalıdır.
- Çalıştırılan komut sayısı sınırlı olmalı ve sonlanmalıdır.

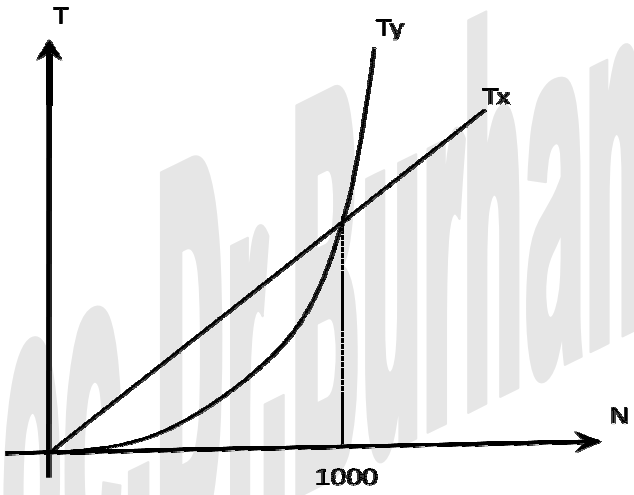
Bir algoritma oluşturulurken,'Tasarım', 'Doğruluğu Teyit', 'Analiz', 'Uygulama' ve 'Test' aşamalarından geçer.

1.3 Algoritma Analizi

Algoritma analizi, algoritma oluşturulduktan sonra özelliklerinin incelenmesidir. Bu aşamada; algoritmanın bellekte ne kadar yer kapladığı ve ne kadar sürede tamamlandığı belirlenmeye çalışılır. Bu belirlenen özellikler algoritmanın ne kadar iyi olduğu sorusunun cevabını vermeye çalışır. Ne kadar

iyi olduğunun cevabının verilebilmesi içinde varsa benzer algoritmalar ile karşılaştırmasının yapılması gereklidir.

Örneğin elimizde X ve Y gibi iki algoritma olduğunu ve analizlerinin yapıldığını kabul edelim. N algoritmanın işleyeceği veri sayısı olmak üzere veri sayısına bağlı olarak algoritmaların harcadıkları zaman $T_x(N)=1000N$ ve $T_y(N)=N^2$ hesaplanmış olsun. Harcadıkları zamanın grafiği şekil 1'deki gibi olur.



Şekil 1.1. X ve Y algoritmalarının harcadıkları zaman.

X algoritmasına ait zaman harcamasının doğrusal bir şekilde arttığı görülürken Y algoritmasına ait zaman harcamasının karesel olarak arttığı görülecektir. Grafikler incelendiğinde X algoritmasının daha iyi olduğu sonucuna varılacaktır. Çünkü girdi sayısı 1000'den küçük olması durumunda Y algoritması X algoritmasından daha zaman harcasa da, genel olarak bakılırsa girdi sayısı 1000'i geçince N arttıkça Y algoritması X algoritmasına göre çok daha fazla zaman harcayacaktır. Sonuç olarak X algoritmasının daha iyi olduğu söylenebilecektir.

Akla şu gelebilir, algoritmanın koşturulduğu bilgisayarın donanımı, kullanılan işletim sistemin yapısı ve kullanılan programlama dilinin özellikleri karşılaştırma açısından ne kadar önemlidir. İşte bu tür ayrıntılardan kurtulup sadece algoritmanın çalışma mantığına odaklanarak değerlendirme yapmak için çeşitli notasyonlar geliştirilmiştir. Bu notasyonlar algoritmaların analizinin yapılmasını kolaylaştırmıştır.

1.4 O Notasyonu

Algoritma analizi için, O (büyük O), Omaga ve Teta gibi notasyonlar mevcuttur. Bu notasyonlardan Omega notasyonu en iyi durum için sonuç verirken Teta notasyonu ortalama harcanan zamanı ifade etmek için kullanılır. O (büyük O) notasyonu ise algoritmanın en kötü durumunu ifade etmeye çalışır. O notasyonunda her hangi bir algoritmanın çalışma süresi *T(N)=O(f(N))* olarak ifade edilir. Buradaki eşitlik gerçek bir eşitlik olmayıp bir gösterim, notasyon şeklidir. *O* bir fonksiyon değildir.

O notasyonu tanım olarak şöyledir;

Her durumda $T(N) \leq cf(N)$ ve $\forall\ n \geq n_0$ şartlarını sağlayan pozitif, sabit c ve n_0 değeri varsa;

T(N)= O(f(N)) ifadesi doğrudur.

T(N)= O(f(N)) ifadesi cümle olarak şöyle ifade edilebilir: " T(N) fonksiyonunun N değişkenine göre artış hızı, f(N) fonksiyonun artış hızından fazla değildir." Bu da bir üst sınır belirtme ifadesidir.

Daha önce verilen X algoritması O notasyonu olarak nasıl ifade edilebileceğini araştıralım. X algoritmasının süresinin $T_X(N)=O(N)$ olarak gösterilip gösterilemeyeceğini inceleyelim. Yukarıdaki tanıma göre 1000N=O(N) olabilmesi için;

$1000N \leq cN$ ve $\forall\, n \geq n_0$ şartını sağlayan bir c ve n_0 sabitleri bulmak gereklidir. Eğer c=3000 ve $n_0=1$ alınırsa şartın sağlanacağı bellidir.

Yine daha önce verilen Y algoritması O notasyonu olarak ifade edersek; $T_B(N)=O(N^2)$ 'dir. Yine tanıma göre $N^2=O(N^2)$ olabilmesi için;

$N^2 \leq cN^2$ ve $\forall\, n \geq n_0$ şartını sağlayan bir c ve n_0 sabitleri bulmak gereklidir. Eğer c=1 ve $n_0=1$ alınırsa şart sağlanacaktır.

Örnek:

$T(N)=9N^2+3 \leq cN = O(N)$ ifadesi doğru mudur?

O notasyonuna göre; $T(N)=9N^2+3 \leq cn$ ve $\forall\, n \geq n_0$ şartını sağlayan bir c ve n_0 sabitleri bulmak gereklidir. Her iki taraf N sayısına bölünürse; $9N+5/n \leq cN$ elde edilir.

Buna göre eşitliğin sağlanabilmesi için N sayısı değiştirilirse C sayısı da değiştirilmelidir. Sabit bir c ve n_0 çifti bulmak mümkün olmadığından notasyon gösterimi doğru değildir.

Örnek:

$T(N)=7N^2+5 \leq cN^2 = O(N^2)$ ifadesi doğru mudur?

O notasyonuna göre; $T(N)=9N^2+3 \leq cN^2$ ve $\forall\, n \geq n_0$ şartını sağlayan bir c ve n_0 sabitleri bulmak gereklidir. Eğer c=50 ve $n_0=1$ alınırsa şart sağlanacaktır. Tek bir çift olması şartın doğru olması için yeterlidir.

Genelde algoritmalar karşılaştırılırken, algoritmaların en fazla harcayacakları zaman üzerinde durulur. Bu nedenle O notasyonu önemlidir. İki algoritma karşılaştırılırken zaman mertebesinden bahsedilir. Mertebesi büyük olan daha yavaş olacağı açıktır. Algoritmanın çalıştırıldığı bilgisayarlar arası fark bir katsayı olarak düşünülürse, $O(9N^2)$ yerine $O(N^2)$ kullanılmasında bir sakınca olmayacaktır. Diğer bir deyişle sabitler ihmal edilebilirler.

1.5 Analiz

Algoritma analizi, bir algoritmanın çalışma süresinin hesaplamak için yapılan çalışma olarak tanımlanabilir.

Örnek olarak n=1'den n=10'a kadar olan sayıların toplanması algoritmasını ele alalım. Sonuç çalışma zamanı Tablo 1'de verilmiştir.

	Birim Zaman	Sıklık	Toplam
sum=0	1	1	1
i=1	1	1	1
while i≤10	1	11	11
sum=sum+1	2	10	20
i=i+1	2	10	20
		Genel	53

Tablo 1.1. n=1'den n=10'a kadar olan sayıların toplanması algoritmasının çalışma zamanı.

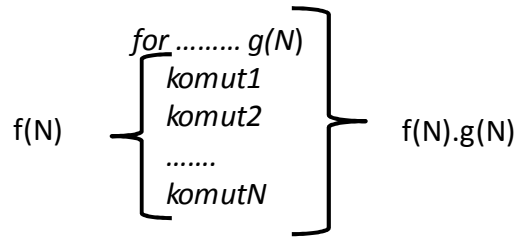
Bu örnek genelleştirilip n=1'den n=N'e kadar olan sayılar için yapılırsa çalışma zamanı Tablo 2'deki gibi olur.

	Birim Zaman	Sıklık	Toplam
sum=0	1	1	1
i=1	1	1	1
while i≤N	1	N+1	N+1
sum=sum+1	2	N	2N
i=i+1	2	N	2N
		Genel	5N+3

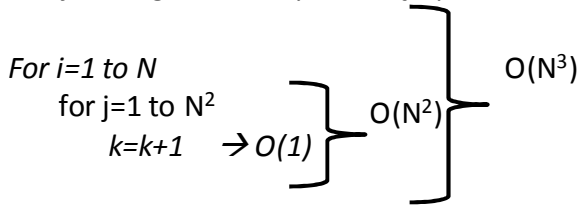
Tablo 1.2. n=1'den n=N'a kadar olan sayıların toplanması algoritmasının çalışma zamanı.

1.6 Algoritma Analizinde Temel Kurallar

Döngüler (For /While/Do..While):
Bir *for* döngüsü yürütme zamanı en çok; *for* döngüsünün içindeki deyim sürelerinin yineleme sayısı kadar miktarına test süresinin ilavesi kadardır.

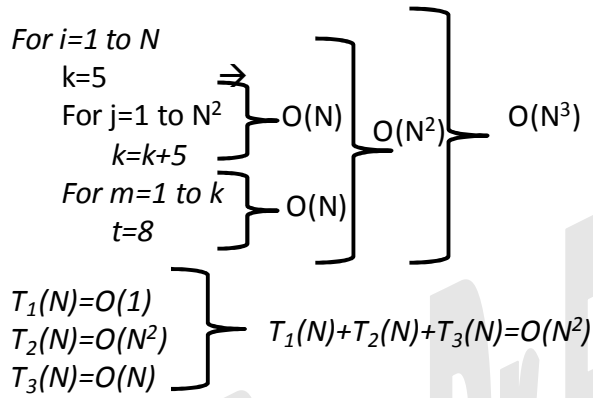


İç içe Döngüler:
İç içe döngülerde grubun içindeki deyimin toplam yürütme zamanı, deyimlerin yürütme sürelerinin bütün *for* döngülerinin boyutlarını çarpımı kadardır. Analiz içten dışa doğru yapılmalıdır.



Ardışık Deyimler:
Ardışık deyimlerin toplam yürütme zamanı, her bir deyimin yürütme zamanının toplanması ile bulunur.

Yani;
$T_1(N)=O(f(N))$ ve $T_2(N)=O(g(N))$ ise ; *$T_1(N)+ T_2(N)=max(O(f(N)) +O(g(N)))$*'dir.
Görüleceği gibi $N+N^2$ yerine N^2 alınmalıdır.



Şart Deyimleri (If/Else):
Toplam zaman, şartın sağlanıp sağlanmamasına bağlı olarak en büyük harcama zamanına karşılaştırması için harcanan zaman ilave edilerek bulunur.

Örnek olarak eşitliğe dayalı ikili arama (binary search) algoritmasının analizini yapalım. Burada Max kullanılacak N adet eleman sayısını ifade etmektedir. İlk döngüde N adet eleman var iken, 2.de N/2, 3.'de N/4 ve k.'de $N/2_k$ =1 ve son döngüde 0 adet vardır.

```
int binarysearch(array, target, Max)
    low=0, high=Max-1;                →O(1)
    while (low≤high)
          mid=(low+high)/2;           →O(1)
          if (array(mid) < target)    →O(1)
             low=mid+1
          else
             high=mid-1;
          if (array[mid]=target)      →O(1)
             return mid;
    return -1; // Bulunamadı          →O(1)
```