Algoritma Analizi (https://birhankarahasan.com/algoritma-analizi-nedir-zaman-karmasikligi-big-o-gosterimi)

- Algoritma analizi, algoritmanın yürütülmesi için gerekli kaynak miktarının belirtilmesidir. Belirli bir problemi çözen herhangi bir algoritmanın ihtiyaç duyduğu kaynaklar için teorik tahminler sağlar. Başka bir ifadeyle, *algoritmanın performansı ve kaynak kullanımı*konusunda yapılan teorik çalışmaların tümüne **algoritma analizi** denir.
- Aslında belli bir makinede çalıştırılabilir *algoritmaların performansı için* dikkat etmemiz gereken kriterler arasında 'programlama dili seçiminin' yanı sıra birçok teknoloji detayı vardır ki bunlar; *derleyici seçimi*, *kodu hangi hızda çalıştırdığımız*, *belleğin ne kadar hızlı olduğu* ve hatta *önbelleğe alma işleminin miktarı* olarak sıralanabilir.
- Peki algoritma analizi en iyi nasıl yapılır? Programlama dillerinden ve donanımlardan bağımsız bir şekilde Algoritma analizi yapılmalıdır. Aksi taktirde en uygun sonuç alınamayabilir.
- Donanımlar veya programlama dilleri farklı cihazlarda aynı performansı vermeyebilir. Örnek verecek olursak, cep telefonları için uygulama tasarladığımızı varsayalım. Bu uygulamanın performansı Apple telefonlar için farklı, Android telefonlar için farklı, arasında donanım farklı olanlar için ayrı olacaktır. Donanım ve diller ile algoritma analizi pek sağlıklı değildir.

  Algoritmanın Performans Kriterleri Nelerdir?
- **Algoritma verimliliğinde** ya da performansında iki kriter vardır. Bunlar *zaman(time)* ve *alan(space)* kavramlarıdır. Amacımız bu kavramlar ışığında aşağıdaki sorulara cevap aramak olacaktır.
- **Zaman (time)**
  - Algoritma ne kadar hızlı faaliyet gösteriyor?
  - Algoritmanın çalışma zamanını ne etkiler?
  - Algoritma için gerekli olan zaman nasıl tahmin edilebilir?
  - Algoritma için gerekli olan zaman nasıl azaltılabilir?

- **Alan (space)**
  - Hangi veri yapısı ne kadar yer kaplar?
  - Ne tür veri yapıları kullanılabilir?
  - Veri yapısı seçimi çalışma zamanını nasıl etkiler?
- Bu noktada *yürütme zamanı*, *zaman karmaşıklığı*, *alan maliyeti* ve *alan karmaşıklığı* gibi kavramlar karşımıza çıkıyor. *Algoritma analizinde* daha çok *yürütme zamanı* ve *zaman karmaşıklığı* üzerinde yoğunlaşacağız ancak diğer kavramlara da bakmakta fayda var.

RAM Modeli

- Genellenebilir bir analiz yapmak için, her algoritmayı aynı bilgisayar ile test ediyor gibi yapacağız.
- Bu hayali makineye RAM (Random Access Machine) diyeceğiz. Ram, algoritmalar arasındaki farkları belirlemek için kullanacağımız bir araç olacak.

  Özellikleri

  1) Her basit işlem (+, -, and, or gibi) 1 birim zaman alır.
  2) Döngüler 1 birim zaman değil, içerisinde kaç defa işlem oluyorsa iterasyon sayısı * işlem sayısı kadar birim zaman alır.
  3) Hafızadan her okuma işlemi 1 birim zaman alır.

Time Complexity

- Algoritmanın verimli olması için belli kurallar vardır.
- Kullanacağımız algoritmayı analiz etmek istiyoruz. Problem aynı olsa da farklı inputlar için algoritmamız farklı performans senaryoları üretebiliyor.
- Diyelim ki bir kelimenin anlamını sözlükte arıyorum. Arama için sayfalara tek tek bakıyorum. Burada algoritmam sayfalara tek tek bakmam, input da aradığım kelime.
- Aradığım kelimenin boyutu aynı olsa da hangi harfle başladığına göre yapacağım işlem sayısı değişebilir. Yani input algoritmamın yapacağı işlem sayısını etkileyebiliyor.
- Bu yüzden analizimizi 3 ana başlık altında yapabiliriz: worst case, avarage case, best case.

1) *Worst Case:* Vereceğimiz inputun algoritmamızı en yavaş (en fazla işlem yapacak) şekilde çalıştırdığı durum. Aradığımız kelimenin "Z" ile başlaması gibi.

   (Her algoritmaya göre farklı. Sözlüğe baştan değil sondan bakmaya başlasaydım "Z" worst case olmazdı.)

2) *Average Case:* Genel olarak beklediğim durum.

3) *Best Case:* Vereceğimiz inputun algoritmamızı en hızlı şekilde çalıştırdığı durum.

- Algoritmamızın çalışmasını en iyi yansıtan average case fakat bu durumu analiz etmek diğerlerine göre çok daha zor. Inputların geldiği dağılımı bilip ona göre analiz etmek gerekiyor.
- Worst case'e göre analiz yaparsak, performansımız için üst sınır çizmiş oluruz. Böylece worst case için bizi tatmin eden bir algoritmamız varsa, average case zaten bundan daha iyi (veya aynı) performans vereceği için o da bizi tatmin edecektir.

Big O Notation

- İki farklı arama yöntemi düşünelim.
- A algoritması tek tek sayfalara bakıyor.
- B algoritması sözlüğün alfabetik sıralanmış olduğundan yararlanarak en başta en ortadaki sayfayı açıyor, eğer bu sayfadaki harfler aradığım kelimeden alfabetik olarak daha ilerideyse sol tarafa aynısını yoksa sağ tarafa aynısını yapıyor.
- Böylece problem her seferinde yarı boyutuna inmiş oluyor.
- Birkaç durum üzerinden konuşalım. Diyelim ki 100 sayfalık bir sözlüğüm var. A algoritması en kötü durumda kaç işlem yapacak? 100 işlem
- B algoritması en kötü durumda kaç işlem yapar? $2^n = 100$
- Bu örneğe bakarak B algoritmasının A algoritmasından daha hızlı olduğunu görebiliyoruz. 100/7 den yaklaşık 15 katı hızında diyebilir miyiz? Peki bu genellenebilir bir şey mi?
- Pek diyemiyoruz. Şöyle düşünelim, sözlüğüm 10.000 elemanlı olsa A algoritması en kötü durumda 10.000 işlem yapar ama B algoritması $2^x = 10.000$ yaklaşık 13 işlem yapar. 10.000/13 = 770 katı hızında gözüküyor.

- Bu yüzden algoritmaların sadece 1 input boyutuna göre karşılaştırmalarına bakıp karar veremeyiz. Genel yapısını bize gösterecek bir analize ihtiyacımız var, burada Big O Notation devreye giriyor.
- Big O Notation algoritmanın ne kadar sürede çalışacağını söylemeyecek. Bize algoritmanın inputun boyutu ile nasıl değişeceğini söyleyecek.
- Sözlük örneğimizde input size'ımıza n dersek, algoritmamızın en kötü durumda n işlem yaptığını söyleyebiliriz. Inputum n boyutunda olunca çalışma süremin de en kötü durumda n olmasını O(n) diye göstereceğim. Aynı şekilde B algoritması için O(logn)
- Big O notation'da yapılacak toplam işlem sayısının input size ile nasıl scale olacağına bakıyoruz. Önemli olan bu fonksiyonun yapısı.
- İşlem sayısı input size ile lineer mi artıyor, karesi ile mi orantılı artıyor, logaritmik mi?
- Karakteristiği önemsediğimiz için 2n işlem yapan algoritmaya da n işlem yapan algoritmaya da O(n) diyoruz, ikiside lineer bir şekilde artıyor, big o notation bakarken katsayılar önemli değil.
- Analizimin sonucu $2n^2+3n+2$ gibi bir şey çıktığını düşünelim. n büyüdükçe, 3n+2 nin etkisi $2n^2$ nin yanında önemsiz kalacak. O yüzden dominant olanı Big O notation olarak yazabiliriz. $O(n^2)$