# **Github :** [KLU5200505054/Algoritma\_Analizi\_Odev\_1 (github.com)](https://github.com/KLU5200505054/Algoritma_Analizi_Odev_1)

# **Soru 1 –**

Bir dizideki en büyük elemanı bulmak için kullanılabilecek en mantıklı algoritma biraz değiştirilmiş bir linear searchtir. 0. elementi bir değişkende tutup, dizinin tüm elemanları be değişkenle karşılaştırabiliriz. Böylece big o notasyonuna göre O(N) karmaşıklığında bir algoritmamız olacaktır.

Bir elemanı değil de en büyük elemanı aradığımız için diğer arama algoritmaları bizim için faydalı olmayacaktır.

Ancak, merge sort sıralaması ile diziyi sıraladıktan sonra, son indexteki elemanını alırsak, bu bize yine dizinin en büyük elemanını verecektir. Fakat bu durumda da zaman karmaşıklığı O(NlogN) olacaktır.

Bu nedenle en mantıklı durum kendimize göre uyarladığımız Linear Search algoritmasını kullanmaktır. Ayrıca Linear Search, brute force mantığıyla çalışan bir algoritmadır. Yani tek tek tüm elemanları kontrol etmektedir. Bu durumda, brute force ile aynı çalışma süresi performansını gösterecektir.

Kod üzerinde yaptığımız testlerde de saniyeniin yüz binde biri aralıklarda değişiklikler görünmektedir.

Kodlar github reposunda bulunmaktadır.

# **Soru 2 –**

Bir algoritmanın çalışma süresi, veri boyutuna bağlı olarak doğrusal, karesel, kübik vb şekillerde artış gösterebilir. Bir çok algoritma küçük veri boyutunda yakın çalışma sürelerinde sonuç verecekken, boyut arttıkça zaman karmaşıklığına göre katlanarak seonuçlar değişecektir.

Örneğin;

A algoritması en kötü durumda O(n^3) zaman karmaşıklığına sahipken B algoritmasının O(n) karmaşıklığına sahip olduğunu varsayalım. Buna göre veri boyutu 2 birimken A algoritması 8t zamanında çalışırken B algoritması 2t zamanında çalışır. Aradaki 6t fark görmezden gelinebilir.

Ancak veri boyutu 200 birimken A algoritması 8 milyon t sürede çalışacakken B algoritması ise sadece 200t sürede çalışacaktır.

Bu durumda veri boyutunun çalışma süresine etkisinin, boyuttan ziyade zaman karmaşıklığına göre değişkenlik göstereceğini söyleyebiliriz.

# **Soru 3 –**

Bir algoritmanın en kötü durumda çalışma süresi, alabileceği en büyük inputu aldığında göstereceği performansı ifade eder.  
Bunu hayali olarak sonsuz – 1 olarak ifade edersek, bu büyüklükteki bir veride, algoritmanın çalışma süresinin ne kadar olacağını belirtir.

Bu durum, big o notasyonnu olarak da ifade edilebilir.

Örnek vermek gerekirse, 1 den 5e kadar sıralı bir dizide 1 sayısını arıyorsak, dizinin tüm elemanlarını son elemana kadar kontrol etmemiz gerekir. Her kontrol t sürede gerçekleşirse t sürede algoritma çalışacaktır. Bu bizim için en iyi durumdur. Fakat dizinin ters sıralı olduğunu düşünürsek, tüm elemanları tek tek kontrol etmemiz gerekecek ve 5 kontrol yapacağız. Bu durumda ise algoritma 5t sürede çalışmasını tamamlar. Bu durum, algoritma için en kötü durum demektir. Ayrıca aradığımız elemanın listede olmaması da aynı çalışma süresini verir ve bu da bizim için en kötü durum olarak nitelendirilebilir.

# **Soru 4 –**

Sıralama için en iyi algoritmalar merge ve heap sort algoritmalarıdır. İkisi de en kötü durumda nlogn zaman karmaşıklığında çalışır.

Brute force için ise selection sort kullanacağım. Ancak bubble sort da tercih edilebilir.

Kodlara göre analizde brute force ile merge sort arasında 6 saniyeden fazla uçuk bir çalışma süresi farkı görülmektedir.

Kodlar github reposunda bulunmaktadır.

# **Soru 5 –**

Örnek olarak binary search algoritmasını ele alalım.

Bu algoritma listenin ortasındaki elemanı aranan elemanla karşılaştırır. Sıralı bir listede çalışır. Eğer ortadaki eleman aranan elemana eşitse, arama tamamlanır. Eğer ortadaki eleman aranan elemandan büyükse, liste ikiye bölünür ve sol yarısından arama devam eder. Eğer ortadaki eleman aranandan küçükse, sağ tarafa bakılarak devam eder. Bu işlem, aranan eleman bulunana kadar tekrarlanır veya dizi elemanları tükenene kadar devam eder. İterative veya recursive olarak yazılabilir.

Binary search, asıl performansını veri boyutu büyüdükçe verir. Çünkü big o’ya göre logn karmaşıklığında çalışır. Buradaki logn, 2 tabanındadır. Yani 100 elemanlı bir dizi için 6,67 karşılaştırmada aranan elemanı bulacakken, 100 milyon elemanlı bir dizide ise yalnızca 26,57 karşılaştırmada aranan elemanı bulur.

Analiz edecek olursak,

En kötü durumda, binary search'te her seferinde veri kümesinin yarısı elenir ve bölme işlemi log(n) kez tekrar edilir.

Yani 10 elemanlı bir dizide, önce 5 elemana düşer, sonra 2, ve sonra tek elemana düşer. Bu da 3 kez karşılaştırma demektir.

Log 2 tabanında 10 = 3,…. Sonucu verir. Buna bakarak algoritmanın logaritmik olarak logn karmaşıklığında çalıştığını söyleyebiliriz.