

**T.C. Fırat Üniversitesi**

**Bilgisayar Mühendisliği Bölümü**

**BMÜ316-Algoritma Analizi Dersi**

**Zaman Karmaşıklığı Analizi Ödevi Raporu**

**Ders Sorumlusu:** Prof. Dr. Mehmet KARAKÖSE

**Öğrenci İsim:** Mert İNCİDELEN

**Öğrenci No.:** 170260101

**Ödev Tanımı**

Ödevde en az 8 farklı algoritma tasarım stratejisine göre yazılmış 8 ayrı algoritmanın sözde kodu ile birlikte algoritmaların karmaşıklıklarının saptanması ve derleyici üzerinde çalıştırılarak en iyi, ortalama ve en kötü senaryolar için geçen sürelerin analizinin yapılması beklenmektedir. Raporda farklı algoritma tasarım stratejileri için farklı algoritmalar örneklendirilmiş, zaman karmaşıklıkları belirtilerek Java dilinde derleyici üzerinde çalıştırarak zaman tüketimleri analiz edilmiştir. Derleyici üzerinde analizi yapılan kodlar raporun sonundaki ekte sunulmuştur.

**1. Böl ve Yönet (Divide and Conquer) Algoritma Stratejisi**

Böl ve yönet stratejisi, problemi daha küçük parçalara bölerek hızlı bir şekilde çözüme ulaşabilmeyi sağlar. Bu strateji için aşağıda ikili arama algoritması örneklenmiştir.

ikiliArama(dizi[], bas, son, aranan)

**while** bas ≤ son **do**

orta = (bas + son) / 2

**if** dizi[orta] == aranan **then**

**return** orta

**else if** dizi[orta] < aranan **then**

bas = orta + 1

**else**

son = orta - 1

**end if**

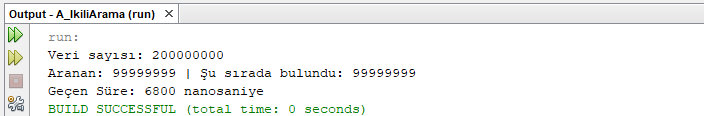
**end while**

**Zaman Karmaşıklığı: O(log n)**

İkili arama algoritmasında, döngünün her bir adımında sıralı dizinin ortasındaki değeri kontrol ederek arama işlemini gerçekleştirmekte ve her adımdan sonra odaklanılması gereken veri sayısı yarıya inmektedir. Dolayısıyla n elemanlı sıralı dizi için zaman karmaşıklığının **O(log n)** olması beklenir. Algoritma Java dilinde bilgisayar üzerinde yürütülerek en iyi, ortalama ve en kötü durumda zaman tüketimi analiz edilmiştir. Boyutu 200000000 olan diziye for döngüsüyle 0-199999999 arası sayılar sıralı bir şekilde yerleştirilmiştir.

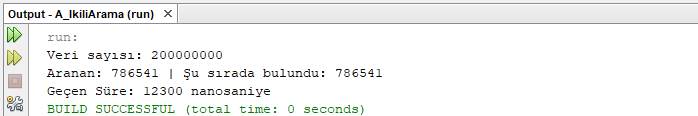
**En İyi Durum: O(1)**

Aranan verinin, veri kümesinin ortasındaki veri olması durumu ikili arama algoritması için en iyi durum senaryosudur. 0-199999999 arası sıralı sayıların bulunduğu dizinin ortasındaki 99999999 değerinin aranması ile gerçekleşen zaman tüketiminin derleyici üzerindeki görünümü aşağıdaki gibidir.

****

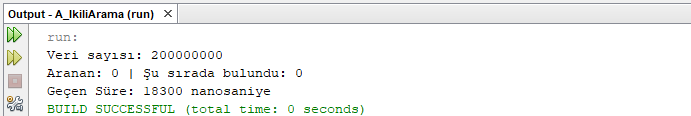
**Ortalama Durum: O(log n)**

Aranan verinin, veri kümesinin herhangi bir yerinde bulunması durumudur. 0-199999999 arası sıralı sayıların bulunduğu dizide 786541 değerinin aranması ile gerçekleşen zaman tüketiminin derleyici üzerindeki görünümü aşağıdaki gibidir.



**En Kötü Durum:** **O(log n)**

Aranan verinin dizin başında ve sonunda bulunması durumu en kötü durum senaryolarıdır. 0-199999999 arası sıralı sayıların bulunduğu dizide 0 değerinin aranması ile gerçekleşen zaman tüketiminin derleyici üzerindeki görünümü aşağıdaki gibidir.



Zaman tüketimi analizinde görülebileceği üzere büyük sıralı veri kümeleri için arama zaman tüketimi minimum düzeyde kalmaktadır. 200000000 veri için en kötü durumda arama işlemi 28 adımda tamamlanmaktadır.

**2. Kaba Kuvvet (Brute Force) Algoritma Stratejisi**

Kaba kuvvet yaklaşımıyla tasarlanan algoritmalar çözüme ulaşmak için tek tek her yolu deneyerek işlemleri gerçekleştirirler. Bu strateji için aşağıda kabarcık sıralama algoritması örneklendirilmiştir. Algoritma, sırasız bir dizinin sıralanabilmesi için diziyi baştan sona dizi boyutu kadar tarayarak sıralama işlemini gerçekleştirir.

kabarcikSiralama(dizi[], diziUzunluk)

**for** i = 0, 1, 2, … , diziUzunluk-1 **do**

**for** j = 0, 1, 2, … , diziUzunluk-1 **do**

**if** dizi[j] > dizi[j+1] **then**

temp=dizi[j]

dizi[j] = dizi[j+1]

dizi[j+1] = temp

**end if**

**end for**

**end for**

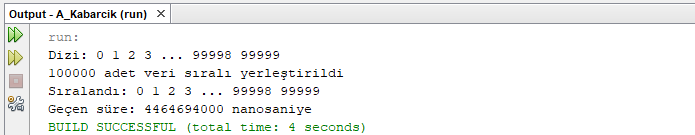
**Zaman Karmaşıklığı: O(n²)**

Kabarcık sıralama algoritmasında dizi boyutu n kadar, n kez tarama işlemi gerçekleştirildiğinden zaman karmaşıklığı **O(n²)** ‘dir. Veri sayısı arttıkça zaman tüketimi de karesel olarak artacaktır.

Algoritma Java dilinde bilgisayar üzerinde yürütülerek en iyi, ortalama ve en kötü durumda zaman tüketimi analiz edilmiştir. Veri sayısı arttıkça zaman tüketimi görülür biçimde karesel olarak artmaktadır. Bu nedenle 100000 adet elemanlı dizi üzerinde zaman tüketim analizi yapılmıştır. Boyutu 100000 olan diziye for döngüsüyle 0-99999 arası sayılar sıralı ve tersten sıralı bir şekilde yerleştirilmiştir. Ortalama durum analizi için ise sayılar dizinin bir başına ve bir sonuna eklenerek dağınık bir şekilde yerleştirilmiştir.

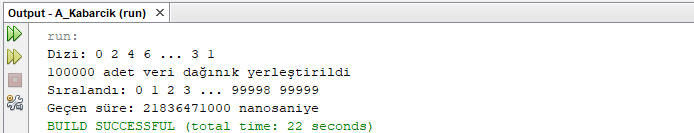
**En İyi Durum: O(n)**

Dizinin sıralı bir şekilde bulunması durumu en iyi durum senaryosudur. Bu durumda dizi bir kez n defa taranacaktır. 0-99999 arası sayıların sıralı bulunduğu dizi için zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



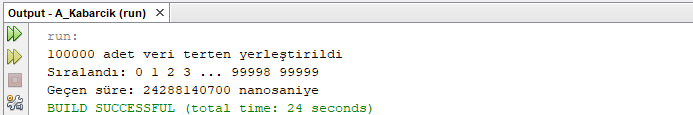
**Ortalama Durum: O(n²)**

Dizide verilerin dağınık bir biçimde bulunma durumu ortalama durum senaryosudur. 0-99999 arası sayılar sırasıyla dizinin başına ve sonuna eklenerek elde edilen dağınık dizinin sıralanması işleminin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**En Kötü Durum: O(n²)**

Dizide verilerin tersten sıralı bir biçimde bulunma durumu en kötü durum senaryosudur. 0-99999 arası sayıların tersten sıralı bir şekilde yerleştirilerek elde edilen sırasız dizinin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**3. Rekürsif Algoritma Stratejisi**

Rekürsif algoritmalar, kendini tekrarlayarak işlemleri gerçekleştirip çözüme kavuşturan algoritmalardır. Rekürsif algoritma örneği için aşağıda n. sıradaki Fibonacci Sayısını hesaplayan rekürsif algoritmanın sözde kodu yer almaktadır.

fibonacci(n)

**if** n == 0 **or** n == 1 **then**

**return** n

**else**

**return** fibonacci(n-1) + fibonacci(n-2)

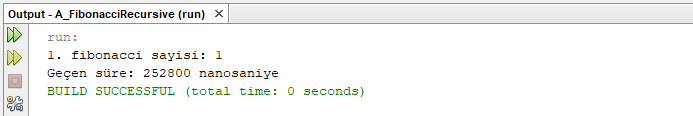
**end if**

**Zaman Karmaşıklığı: O(2n)**

Bu algoritmada n. sayının hesaplanabilmesi için n-1. ve n-2. sayıların hesaplanması ve bu her iki sayı için de iki yeni hesaplamanın yapılarak bu şekilde hesaplamanın devam etmesi dolayısıyla işlemlerin 2n defa tekrarlanması gerekmektedir. Buna göre bu algoritmanın zaman karmaşıklığı **O(2n)**‘dir. n değeri arttıkça hesaplama süresi 2 tabanında üssel artmaktadır.

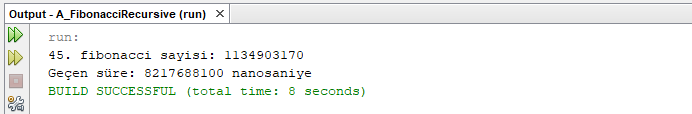
**En İyi Durum: O(1)**

1. sıradaki fibonacci sayısının hesaplanması işlemi en iyi durum senaryosudur. 1. Sıradaki fibonacci sayısının hesaplanması işleminin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



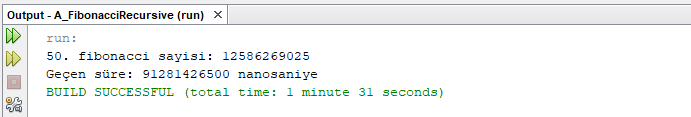
**Ortalama Durum: O(2n)**

n değeri arttıkça hesaplama işlemi için gereken süre artmaktadır. n değeri 45 seçilerek zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**En Kötü Durum:** **O(2n)**

Aşağıda yer alan derleyici üzerindeki görüntüde n değeri önceki değere göre 5 arttırılmış, buna rağmen zaman tüketiminde büyük artış gözlemlenmiştir.



Anlaşılacağı üzere yüksek n değerlerine karşılık fibonacci sayılarının hesaplanabilmesi için çok uzun zamana ihtiyaç duyulacaktır. Bir sonraki başlıkta fibonacci sayılarının hesaplanması işlemi dinamik programlama stratejisi ile incelenmiştir.

**4. Dinamik Programlama Stratejisi**

Dinamik programlama stratejisinde, problemin alt problemlere bölünmesi ve bu alt problemlerin bir kez çözülerek çözümün sonraki tekrarda kullanılması yaklaşımı görülür. Aşağıda dinamik programlama stratejisi ile n. sıradaki Fibonacci Sayısını hesaplayan algoritmanın sözde kodu yer almaktadır.

fibonacci(n)

**new** fib[]

fib[0] = 1

fib[1] = 1

**for** i = 2, 3, 4, … , n **do**

fib[i] = fib[i - 1] + fib[i - 2]

**end for**

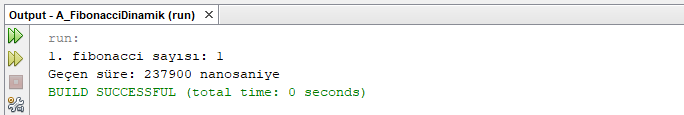
**return** fib[n-1]

**Zaman Karmaşıklığı: O(n)**

Bu algoritmada n. fibonacci sayısının hesaplanabilmesi için 1’den n’e kadar fibonacci değerleri hesaplanıp kaydedilecek, dolayısıyla n kez bu işlemler tekrarlanacaktır. Buna göre bu algoritmanın zaman karmaşıklığı **O(n)** ‘dir. n değeri arttıkça hesaplama için gereken zaman doğrusal olarak artacaktır. Algoritma Java dilinde bilgisayar üzerinde yürütülerek en iyi, ortalama ve en kötü durum senaryoları için n değeri seçilerek zaman tüketimi analiz edilmiştir.

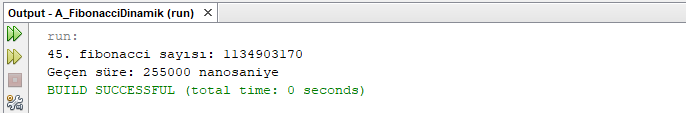
**En İyi Durum: O(1)**

1. sıradaki fibonacci sayısının hesaplanması işlemi en iyi durum senaryosudur. Bu işlemin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



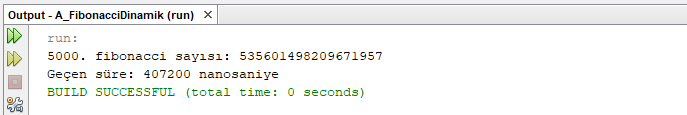
**Ortalama Durum:** **O(n)**

Bir önceki rekürsif fibonacci algoritmasında n=45 değeri için gereken zaman tüketimi bu algoritma için derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**En Kötü Durum: O(n)**

n değeri arttıkça zaman tüketimi doğrusal olarak artacaktır. Burada n değerinin rekürsif algoritmada örneklenen değerlerden oldukça yüksek olmasına rağmen kısa sürede hesaplama işlemi sonuçlanmıştır.



**5. İteratif Algoritma Stratejisi**

İteratif algoritma stratejisi, bir problemi çözüme kavuşturmak için gerekli adımları ayrı ayrı gerçekleştirmek yerine tekrarlayan işlemleri bir döngü ile adım adım çözümlendirme yaklaşımıdır. Aşağıda n sayısının faktöriyel hesabını gerçekleştiren sözde kod yer almaktadır.

faktoriyel(n)

a=1

**for** i = 1, 2, 3, … , n+1 **do**

a = a \* i

**end for**

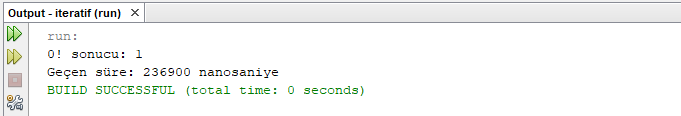
**return** a

**Zaman Karmaşıklığı: O(n)**

Bu algoritmada n değerinin faktöriyel hesabı için döngü n defa gerçekleşecektir. Dolayısıyla bu algoritmanın karmaşıklığı **O(n)**’dir. n değeri arttıkça işlemler için geçen zaman doğrusal olarak artar.

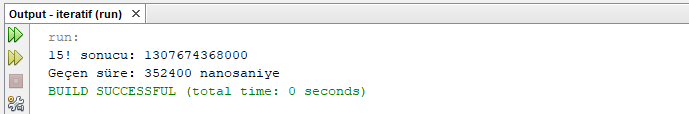
**En İyi Durum: O(1)**

n=0 değeri, hesaplama için en iyi durumdur. Bu durumda gerçekleştirilen işlemin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



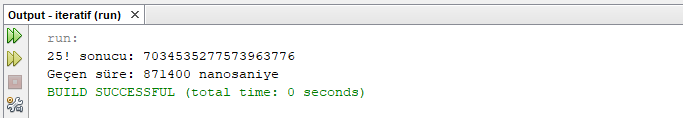
**Ortalama Durum: O(n)**

n değeri arttıkça hesaplama işlemi için gereken süre doğrusal olarak artmaktadır. n değeri 15 seçilerek zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**En Kötü Durum:** **O(n)**

n değeri yüksek değerler seçildiğinde tüketilen zaman doğrusal olarak artacaktır. n=25 değeri için tüketilen zamanın derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**6. Azalt ve Yönet (Decrease and Conquer) Stratejisi**

Azalt ve yönet stratejisi, problemin veri boyutunu azaltarak çözüme kavuşturmaya yönelik bir stratejidir. Bu strateji için aşağıda eklemeli sıralama algoritması örneklenmiştir.

eklemeliSiralama(dizi[], diziUzunluk)

**for** i = 1, 2, 3, … , diziUzunluk **do**

temp = dizi[i]

j = i

**while** j > 0 **and** dizi[j - 1] > temp **do**

dizi[j] = dizi[j - 1]

j = j - 1

**end while**

dizi[j] = temp

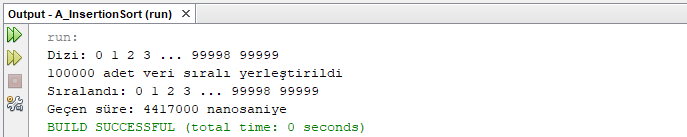
**end for**

**Zaman Karmaşıklığı: O(n²)**

Eklemeli sıralama algoritması n elemanlı bir dizide 2. veri için 1 kez, 3. Veri için 2 kez ve her veri için aynı şekilde devam ederek işlemler gerçekleştirilir. Dolayısıyla algoritmanın zaman karmaşıklığı **O(n²)** ‘dir. Veri sayısı arttıkça zaman tüketimi de karesel olarak artacaktır.

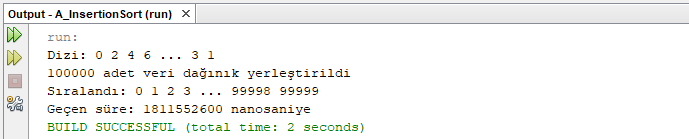
**En İyi Durum: O(n)**

Dizinin sıralı bir şekilde bulunması durumu en iyi durum senaryosudur. Bu durumda dizi bir kez n-1 defa taranacaktır. 0-99999 arası sayıların sıralı bulunduğu dizi için zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**Ortalama Durum: O(n²)**

Dizide verilerin dağınık bir biçimde bulunma durumu ortalama durum senaryosudur. 0-99999 arası sayılar sırasıyla dizinin başına ve sonuna eklenerek elde edilen dağınık dizinin sıralanması işleminin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.



**En Kötü Durum: O(n²)**

Dizide verilerin tersten sıralı bir biçimde bulunma durumu en kötü durum senaryosudur. 0-99999 arası sayıların tersten sıralı bir şekilde yerleştirilerek elde edilen sırasız dizinin zaman tüketiminin derleyici üzerindeki görüntüsü aşağıdaki gibidir.

