Insertion Sort ve Mergesort Algoritmalarının Java Programlama Dili Kullanılarak Gerçekleştirimi ve Karşılaştırılması

Ömer Faruk Alaca
91150000557 – omerfarukalaca@gmail.com
Ege Üniversitesi – Uluslararası Bilgisayar Enstitüsü
Nisan, 2016

ÖZET

Sıralama problemleri yazılım dünyasında verimli kaynak kullanımı ve çözüme hızlı ulaşma parametreleri açısından üzerinde çalışmalar yapılan önemli bir konudur. Bu çalışmalar sonucunda ortaya atılan algoritmalardan ikisi de *"insertion sort"* ve *"mergesort"* algoritmalarıdır. Bu çalışma ile java dili kullanılarak Eclipse geliştirme ortamında bu iki algoritma gerçekleştirilecektir. Sonuç olarak kullanıcının istediği uzunlukta, rastgele doğal sayılarla doldurulmuş dizileri, yukarda bahsedilen iki algoritmayla sıralayan ve sonucunda sıralama sürelerini hesaplayan bir program elde edilmiştir.

1. GİRİŞ

Çalışmada yer alan "insertion sort" ve "mergesort" algoritmaları sıralama problemi çözüm olarak geliştirilen bir çok algoritmadan ikisidir. Insertion sort algoritması *"Bubble Sort"* sıralam algoritmasının iyileştirilmiş biçimidir. Zaman karmaşası $O(n^2)$'dir.**[1]** Sıralı diziyi her adımda öge öge oluşturan bir algoritmadır. *"Insertion Sort"* algoritması, düzensiz dizi elemanlarını tek tek ele alarak her birini dizinin sıralanmış kısmındaki uygun yerine yerleştirme esasına dayanır. Genelllikle günlük hayatımızda farketmeden kullandığımız bir çözüm yöntemidir. Küçük boyutlu dizileri sıralamada hızlı olsa da çok sayıda veriyi sıralarken Insertion Sort diğer sıralama yöntemlerine göre çok yavaş kalmaktadır.**[2]**

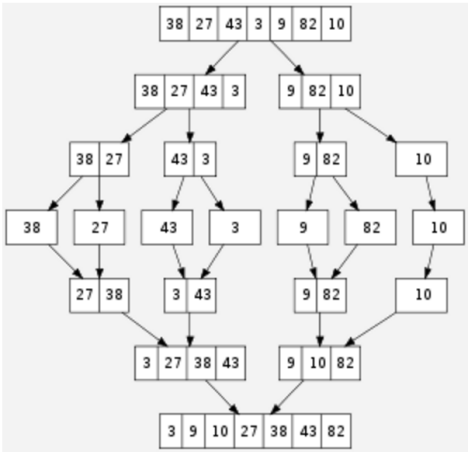
Insertion Sort algoritması başlangıçtan itibaren Şekil 1'de de görüldüğü gibi solunu sıralı hale getirerek dizinin sağına doğru ilerleme mantığı taşır.

Veriler	25	57	48	37	12	92	86	33
Tekrar 1	25	57	48	37	12	92	86	33
Tekrar 2	25	48	57	37	12	92	86	33
Tekrar 3	25	37	48	57	12	92	86	33
Tekrar 4	12	25	37	48	57	92	86	33
Tekrar 5	12	25	37	48	57	92	86	33
Tekrar 6	12	25	37	48	57	86	92	33
Tekrar 7	12	25	33	37	48	57	86	92

Şekil 1 Insertion Sort [4]

Merge Sort algoritması ise diziyi ardışık olarak en küçük alt dizilerine kadar bölen sonra da onları sıraya koyarak bireştiren özyineli bir

algoritmadır. Bölme işlemi en büyük alt dizi en çok iki öğeli olana kadar sürer. Sonra *"Merge (Birleştirme)"* işlemiyle altdiziler ikişer ikişer bölünüş sırasıyla sıralı olarak bir üst dizide bireşir. Süreç sonunda en üstte sıralı diziye ulaşılır.**[3]** Zaman karmaşıklığı O(nlogn)'dir.



Şekil 2 Merge Sort[3]

Şekil 2'de de görüldüğü gibi öncelikle sıralı olmayan dizi ortadan olacak şekilde alt iki diziye ayrılır. Ardından bu ayırma işlemi alt diziler en çok iki elemanlı olana kadar devam eder.Sonrasında altdizileri kendi içinde sıralayarak birleştirir ve bütün diziye sıralı olarak ulaşılır.

Yukarda ayrıntılan algoritmalar bizim çalışmamızda temel alınan sıralama algoritmalarıdırAlgoritmaları gerçekleşteceğimiz java uygulaması ile bu iki algoritmanın 300 bin, 500 bin, 1 milyon ve 10 milyon gibi uzunluklardaki dizileri sıralarken ortaya koydukları performanslar ölçülecektir. Ayrıca kullanıcının da dizi uzunluğu girerek sıralayabilmesi sağlanacaktır.

Çalışmanın bundan sonraki bölümlerinde öncelile ortaya konacak programın yazılımsal analizi 2. bölümde yapılacaktır. 3. bölümde nesneye yönelik yazılım prensipleri ile gerçekleştirilecek yazılımın tasarımı ortaya konacaktır. 4. bölümde ise gerçekleştirim aşamasına dair ayrıntılandırma yapılacaktır. Ardından 5. bölümde çalışmanın önemli bir çıktısı olan iki algoritmanın karşılaştırılması çalışması yapılacak, son bölüm olan 6. bölümde ise çalışmadan çıkarılan sonuçlar tartışılacaktır.

2. ANALİZ

Gerçekleştirmek istediğimiz yazılım bir sıralama algoritmasını uygulama yoluyla analiz etme amacı taşımaktadır. Gerçekleştirim sınırlarımız her ne kadar 2 adet sıralama algoritması ile sınırlanmış olsa da yazılımın doğasında aranan tekrar kullanılabilirlik ve bakım sonucu gelecekte üzerine yeni sıralama algoritmalarının eklenebildiği bir yazılım düşünülmelidir. Çalışmanın ana odağı ise verilen algoritmalar için sıralama süresi hesaplayabilen bir yazılımın ortaya çıkarılmasıdır.

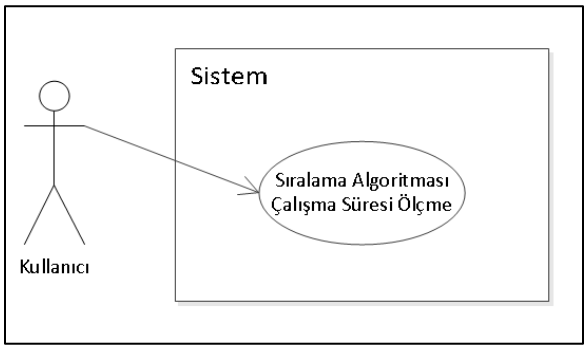
Analiz aşamasında ilk olarak bu yazılımı kullanacak aktörlere ve kullanım senaryolarına yakından bakalım.

Yazılım sonucunda beklenti bir sıralama algoritmasına verilen dizinin sıralanması ve bu sıralama süresince geçen sürenin hesaplanmasıdır. Hedefi göz önüne aldığımızda sistemimizde sadece kullanıcı aktörü bulunmaktadır ve tek bir kullanım senaryosu mevcuttur. Bu durum gelecekte ekleme/çıkarma yapılabilecek bir yazılım için dikkatli olarak sınırlamaların yapılmasını zorunlu kılar.

Aktör : Kullanıcı
Amaç: Bir diziyi sıralamak ve sıralama süresini elde etmek.

Sistemimizde bu çalışma için 2 adet algoritma bulunacaktır ve gelecekte ekleme yapıldığını düşünürsek kullanıcı aynı diziyi birden fazla algortima ile sıralayıp sürelerini görmek isteyebilir. Ayrıca diziler sabit uzunlukta olmayacaktır. Daha kullanılabilir ve kullanıcının

amacına uygun bir yazılım için dizi uzunluğu kullanıcıdan alınacaktır.



Şekil 3 : Aktör ve Kullanım Senaryosu

Kullanım Senaryosu

Sıralama Algoritması Çalışma Süresi Ölçme

1. Kullanıcı programı açarak sistemi aktif hale getirir.
2. Sistem kullanıcıdan sıralamak istediği dizinin uzunluğunu ister.
3. Kullanıcı dizi uzunluğunu sisteme girer.
4. Sistem verilen uzunlukta bir dizi oluşturur ve diziyi rastgele doğal sayılarla doldurur.
5. Sistem kullanıcıdan sıralama için kullanmak istediği algoritmaları seçmesini ister.
6. Kullanıcı dizinin sıralanmasını istediği algoritmaları belirtir.
7. Sistem belirtilen algoritmalarla diziyi sıralar.
8. Sistem algoritmaların diziyi sıralama süresini kullanıcıya gösterir.
9. Kullanıcı sistemden çıkar.

Bu senaryo kullanıcının hatalı giriş yapması durumunda sistemin aynı veriyi tekrar istemesi şeklinde dallanabilir. Bu durumlar haricinde sistem kaynaklı aksamalar da akışı bozabilecek diger durumdur.

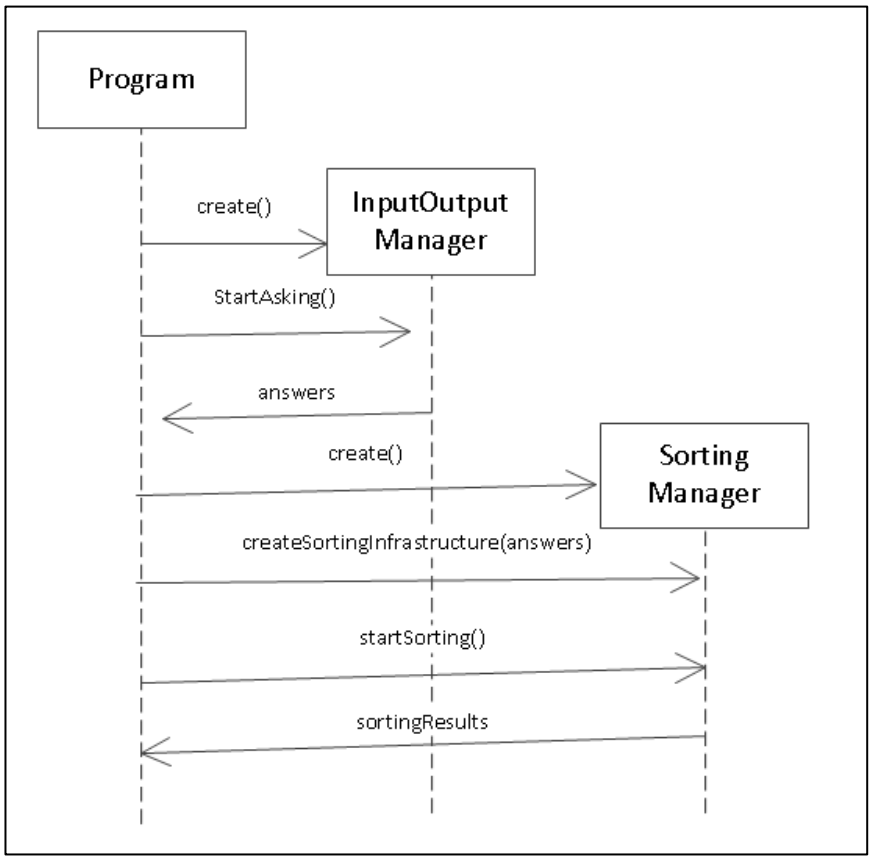
Analiz aşamasında yaptığımız çalışma ile sistemin büyük resmini görmüş ve genel amacı kesinleştirmiş olduk. Böylece elimizde tek bir kullanım senaryosu ve aktör bulunmakta. Tasarım aşamasında da bu aktörü ve kullanım senaryosunu ayrıntılandıracağız.

3. TASARIM

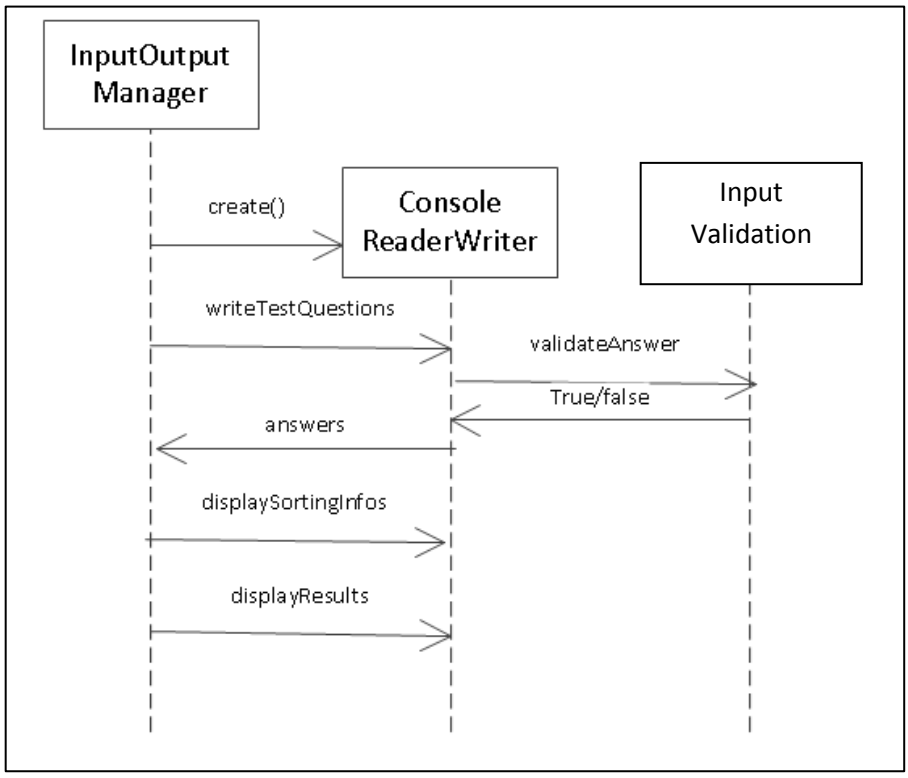
Analiz aşamasında elde ettiğimiz sonuç neticesinde elimizde bir adet kullanım senaryosu bulunmakta bu doğrultuda bu doğrultuda bu senaryoya uygun sınıf yapısını oluşturuyoruz. Burada dikkat edeceğimiz iki nokta var. Büyük sisteme baktığımızda hedef işin fazla parçalı olmamasından dolayı iki temek sınıf gözümü çarpıyor. Birincisi kullanıcı ile sistemin haberleşmesini sağlayacak olan sınıfı "InputOutput Manager" ve sıralama işleminden sorumlu olan "Sorting Manager" sınıfıdır. Sistemin akışını daha iyi anlamak için öncelikle program ile bu iki sınıfın etkileşimini gösteren sequence diyagramı inceleyelim.

Şekil 4'te gördüğümüz sequence diyagramda programın başlatılmasıyla birlikte öncelikle "InputOutput Manager" yaratılıyor. Bu sınıf kullanıcı ile programı bağlayan sınıf olarak tasarlanmıştır. Yaratıldıktan sonra tetiklenen sınıf kullanıcıya yapılacak sıralama ile ilgili sorular soruyor ve cevapları tekrar program sınıfına dönüyor. İkinci aşamada sıralama işlemlerini yürütecek olan Sorting Manager sınıfı yaratılıyor ve cevaplarla beraber sıralama ortamını oluşturması için tetikleniyor. Böylece bu sınıf diziyi oluşturup rastgele elemanlarla doldurarak istenen algoritmalarla sıralanacak şekilde hazır hale getiriyor. Sonraki aşamada ise program tarafından tekrar tetiklenen Sorting Manager sınıfı sıralama işlemlerini yaparak sonuçları program sınıfına dönüyor.

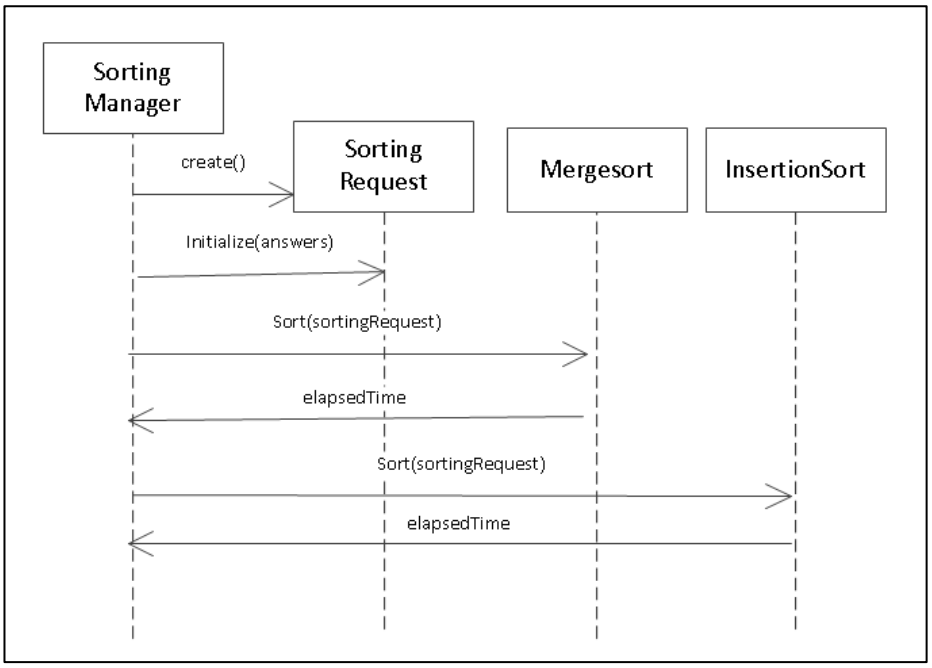
Burada dikkat edilmesi gereken nokta bu sınıfların birbirlerini bilme durumlarının düzenlenmesidir. Örneğin sıralama işlemini sorting manager sınıfı tek başına yapmamaktadır. Rahat anlaşılması için sorting manager'ın iletişim kurduğu ve işlemleri gerçekleştirmelerini sağladığı diger sınıflar şekilde gösterilmemiştir. Aynı durum InputOutput Manager sınıfı için de geçerlidir.



Şekil 4



Şekil 5



Şekil 6

Yukarda bahsedilen sadeleştirme sonucu kaybolan diger önemli sınıfların ana sınıflarımızla ilişkileri de Şekil 5 ve Şekil 6'da gösterilmiştir. Bu diyagramlarda da karmaşa yaratmamak için bütün olarak gösterilmemiş önemli sınıflar sembolize edilmiştir.

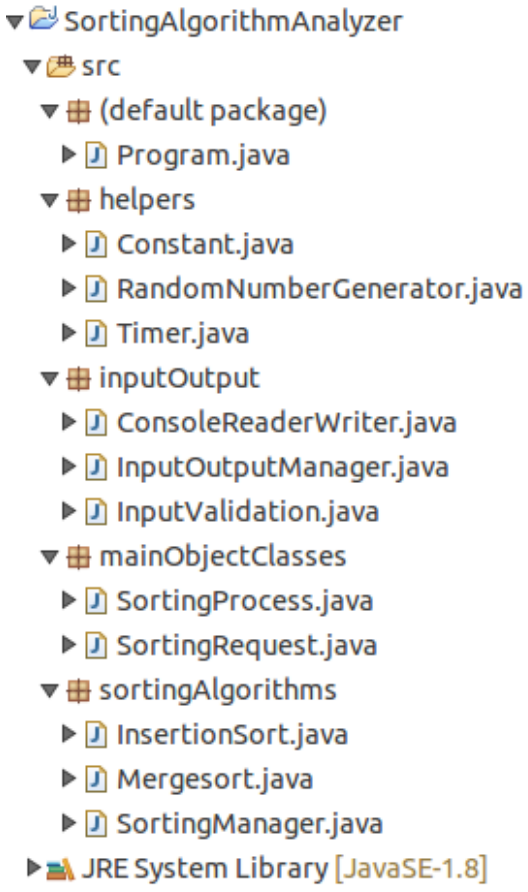
4. GERÇEKLEŞTİRİM

Gerçekleştirim aşaması analiz ve tasarımda detayları belirlenen yazılım sisteminin java programlama dili ile eclipse ortamında gerçekleştirilmesi ile tamamlanmıştır.

İşletim Sistemi: Ubuntu 14.04
Programlama Dili: Java
IDE : Eclipse Mars

Gerçekleştirim aşamasında Şekil 7 'de görülen sınıflar package'lar içerisinde gruplanarak gerçekleştirilmiştir.

(Default package) : İçerisinde sadece program.java sınıfı bulunur. Program.java sistemin main sınıfını temsil eder.



Şekil 7

helpers: İçerisinde sistemin ana sınıfları olmayan yardımcı sınıfları barındırır. Bu sınıflar hedefe ulaşırken kullanılan küçük işleri tamamlar.

inputOutput : Kullanıcı ile yazılımın iletişimini sağlayan pakettir. Bu paket gerçekleşen sıralama işi ile ilgili bilgi içermez. Sadece verien datayı yazıp, aldığı datayı aktarır.

mainObjectClasses: Bu sınıflar sıralama işlemi için temel nesneleri barınıdırır. Bu nesneler Sıralama isteği ve sıralama işlemleridir. Sıralama isteği kullanıcının sistemi kullanarak sıralama yapmak istediği anda yaratılan ve içesinde dizi uzunluğu, dizi, kullanıcının istediği sıralama algoritmaları gibi bilgileri tutar. Sıralama işlemleri ise kullanıcının istediği algoritmaların nesneleridir. Örneğin kullanıcı mergesort ile sıralama yapılsın dediğinde yaratılır ve içerisine sıralanacak dizi'nin kopyası oluşturulur. Ayrıca içerisinde algoritmanın başlangıç bitiş zamanları toplam sürelerin nanosaniye, milisaniye cinsinden değerleri gibi o sıralama algoritmasına ait bilgileri saklar.Doğrudan sıralama isteğine bağlı nesnelerdir. Sonuç olarak harhangi bir sıralama hakkında bütün bilgiler bu iki nesne içerisindedir.

5. TEST

Gerçekletirilen program üzerinde mergesort ve insertionSort algoritmalarının tamamlan süreleri Tablo 1'deki değerler gibi gerçekleşmiştir.

	Mergesort	InsertionSort
300 Bin	58 ms	42292 ms
500 Bin	114 ms	117716 ms
1 Milyon	165 ms	472812 ms
10 Milyon	1799 ms	-

Tablo 1

Testler 300 bin, 500 bin, 1 milyon ve 10 milyon uzunluklu dizilerde gerçekleştirilmiştir. Testin daha verimli olabilmesi için aynı diziler iki algoritmaya da uygulanmıştır. Test öncesi beklentimiz zaman karmaşıklığı daha iyi olan mergesortun daha hızlı bir sıralama gerçekleştirmesiydi. Test sonuçları da bu durumu doğrulamıştır. 10 milyon uzunluklu dizide insertion sort işlemi ideal sürelerde tamamlayamadığı için sonucu görülememiştir. Bunun harici diğer dizi uzunluklarında iki algoritma da sonuca ulaşmıştır.

Test sonuçlarına ait ekran görüntüleri raporun sonuna eklenmiştir. **[EK 2]**

6. SONUÇ

Bu çalışma sonucunda sıralama algoritmalarından mergesort ve insertionSort algoritmalarının çalışma mantıkları kavranmış, java dilinde gerçekleştirimleri yapılmıştır. Ayrıca büyük ölçekli dizilerde insertion sort algoritmasının verimli çalışamadığı, mergesortun ise iyi sonuçlar verdiği gözlemlenmiştir.

KAYNAKÇA

[1] Baskent Universitesi Ders Notları
http://www.baskent.edu.tr/~tkaracay

[2] ITU Ders Notları; Insertion Sort
http://bidb.itu.edu.tr/seyirdefteri/blog/2013/09/08/insertion-sort-algoritmas%C4%B1

[3] ITU Ders Notları; Merge Sort
http://bidb.itu.edu.tr/seyirdefteri/blog/2013/09/08/merge-sort-(bile%C5%9Ftirme-s%C4%B1ralamas%C4%B1)-algoritmas%C4%B1

[4] Ege Univ. Prof. Dr. Aybars Uğur
http://yzgrafik.ege.edu.tr/~ugur/09_10_Fall/DS/11_SORTING.pd

EK 1: SIRALAMA KAYNAK KODLARI

```
 # INSERTION SORT PSEUDOCODE

 INSERTIONSORT(A,n)   >A[0..n]
 for j <- 1 to n
         do key <- A[j]
            i <- j-1
            while i>0 and A[i]>key
                  do A[i+1] <- A[i]
                     i <- i-1
            A[i+1] = key
```

```
# MERGESORT PSEUDOCODE

func mergesort( var a as array )
     if ( n == 1 ) return a

     var l1 as array = a[0] ... a[n/2]
     var l2 as array = a[n/2+1] ... a[n]

     l1 = mergesort( l1 )
     l2 = mergesort( l2 )

     return merge( l1, l2 )
end func

func merge( var a as array, var b as array )
     var c as array

     while ( a and b have elements )
          if ( a[0] > b[0] )
               add b[0] to the end of c
               remove b[0] from b
          else
               add a[0] to the end of c
               remove a[0] from a
     while ( a has elements )
          add a[0] to the end of c
          remove a[0] from a
     while ( b has elements )
          add b[0] to the end of c
          remove b[0] from b
     return c
end func
```

EK 2: TEST SONUÇLARI

```
################################################################################
SIRALAMA BİLGİLERİ;
Sıralama Zamanı         : Thu Apr 14 01:34:29 EEST 2016
Dizi Uzunluğu           : 300000
Algoritmalar            : Mergesort - Insertion Sort -

Mergesort               : 58.0 milisaniye       =       58683829 nanosaniye
Insertion Sort          : 42292.0 milisaniye    =       42292032568 nanosaniye

################################################################################


################################################################################
SIRALAMA BİLGİLERİ;
Sıralama Zamanı         : Thu Apr 14 01:36:43 EEST 2016
Dizi Uzunluğu           : 500000
Algoritmalar            : Mergesort - Insertion Sort -

Mergesort               : 114.0 milisaniye      =       114322879 nanosaniye
Insertion Sort          : 117716.0 milisaniye   =       117716512979 nanosaniye

################################################################################


################################################################################
SIRALAMA BİLGİLERİ;
Sıralama Zamanı         : Thu Apr 14 01:40:07 EEST 2016
Dizi Uzunluğu           : 1000000
Algoritmalar            : Mergesort - Insertion Sort -

Mergesort               : 165.0 milisaniye      =       165382721 nanosaniye
Insertion Sort          : 472812.0 milisaniye   =       472812404204 nanosaniye

################################################################################

################################################################################
SIRALAMA BİLGİLERİ;
Sıralama Zamanı         : Thu Apr 14 01:49:36 EEST 2016
Dizi Uzunluğu           : 10000000
Algoritmalar            : Mergesort -

Mergesort               : 1799.0 milisaniye     =       1799152804 nanosaniye

################################################################################
```