

T.C. İSTANBUL AYDIN ÜNİVERSİTESİ

-BİLGİSAYAR MÜHENDİSLİĞİ YÜKSEK LİSANS-

-BYL563 İleri Algoritma Analizi-

**BAZI SIRALAMA**

**ALGORİTMALARININ**

**KARŞILAŞTIRMALI**

**ANALİZİ**

**Hazırlayan:** Veysi ERTEKİN (YL1213.010020)

İçindekiler

[A. Problemin Tanımı 3](#_Toc370974282)

[B. Sıralama Algortimaları 3](#_Toc370974283)

[1. Bouble sort: 3](#_Toc370974284)

[2. Insertion sort: 4](#_Toc370974285)

[3. Selection Sort: 4](#_Toc370974286)

[4. Merge Sort: 5](#_Toc370974287)

[5. Quick Sort: 6](#_Toc370974288)

[6. Heap Sort: 7](#_Toc370974289)

[C. Algoritmaların Analizi 8](#_Toc370974290)

[D. Ekler 10](#_Toc370974291)

[1. TestCase.java: 10](#_Toc370974292)

[2. Main.java: 11](#_Toc370974293)

[3. BubbleSort.java: 11](#_Toc370974294)

[4. HeapSort.java: 11](#_Toc370974295)

[5. InsertionSort.java: 12](#_Toc370974296)

[6. MergeSort.java: 12](#_Toc370974297)

[7. QuickSort.java: 13](#_Toc370974298)

[8. SelectionSort.java: 14](#_Toc370974299)

[9. AAlgorithm.java (Abstract Class): 14](#_Toc370974300)

[E. Kaynakça 15](#_Toc370974301)

# Problemin Tanımı

Herhangi bir sayıdaki aynı tip verilerin sınırlı bellek ve işlem gücü ile belirli bir sıraya göre dizilmesinin sağlanmasıdır. Burada önemli olan en optimum bellek/performans ikilisini verecek bir algoritma(yöntem)nın elde edilmesidir.

Sıralama algoritmaları bazı kriterlere göre sınıflandırılabilir:

* ***Hesaplama karmaşıklığı:*** oluşturulmuş olan algoritmanın yaptığı işlem sayısının genel bir yapı ile ifade edilmesidir. Temel üç grup ölçek kullanılır. Bunlar **en iyi(best)**, **ortalama(average)** ve **en kötü(worst)** durum olarak belirtilir. Örnek verilecek olursa **Big O** belirteci algoritma karmaşıklığının üst sınırını belirtmek adına oluşturulmuştur. İşlem yoğunluğu ***zamanın işleyişiyle*** paralel olduğundan(ne kadar çok işlem yapılırsa o kadar uzun süre geçer) algoritmanın işleyiş süresini de etkiler.
* ***Yer değiştirmelerin Karmaşıklığı:*** içerisinde *ek* bellek kullanmayan**(in place)** algoritmalarda kullanılan karşılaştırılabilmesi içinönemli bir ölçüttür.
* ***Bellek Kullanımı:*** çalışırken ek bellek ihtiyacı duyan algoritmalarda kullanılabilecek bir ölçüttür.
* ***Rekürsiflik:*** iç içe kendi kendini çağıran algoritmalarda kullanılan bir ölçüttür. Burada en önemli kriter **stack** dediğimiz maximum iç içe çağrım kapasitesine dikkat edilmesi ve bu kapasitenin kullanılma sıklığıdır.
* ***Durağanlık(stability):*** algoritmanın uygulanması sırasında sıralanmış bir verinin tekrar sıralamaya tabi tutulup tutulmadığını belirten ölçüttür.

# Sıralama Algortimaları

## Bouble sort:

“Kabarcık sıralaması”nda sırasıyla ikililer şeklinde karşılaştırma yapılır, belirlenen sıralama ölçütüne bağlı olarak verilerin hangisinin önde veya arkada olacağının bulunması ve eğer verilerin yanlış bir sırada olmasının tespit edilmesi durumunda yerlerinin birbiriyle değiştirilmesine (**swapping**) dayanır.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| En iyi | Ortalama | En kötü | Bellek | Stabil mi? |
| *n.log(n)* | ***n.log(n)*** | ***n^2*** | ***log(n)~n*** | ***Evet*** |

(Genel algoritma yapısı)

|  |
| --- |
| **function** boubleSort**(**A**:** list **of** items**)**  **for** **(**i**=**1**;** i**<**A**.**lenght**-**1**;** i**++)**  **for** **(**j**=**A**.**lenght**;** j**>**i**;** j**--)**  **if** A**[**j**]** **<** A**[**j**-**1**]**  swap**(**j**,** j**-**1**,** A**)** // A[j] ile A[j-1]’in yerlerini değiştir. |

## Insertion sort:

Her bir değerin sırayla alınıp kendinden küçük ve kendinden büyük değerlerin arasına alınmasıyla gerçekleştirilir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| En iyi | Ortalama | En kötü | Bellek | Stabil mi? |
| *n* | ***n^2*** | ***n^2*** | ***1*** | ***Evet*** |

(Genel algoritma yapısı)

|  |
| --- |
| **function** insertionSort**(**a**:** list **of** items**)**  **for** **(**i**=**1**;** i**<**a**.**lenght**;** i**++)** // her bir elemanı en az bir kere işle  int temp **=** a**[**i**];** // seçilen elemanı tamponla  int j**;**  **for** **(**j**=**i**-**1**;** j**>=**0 **&&** temp**<** a**[**j**];** j**--)** // en büyükten başlayarak kendinden büyükleri bir öne al  a**[**j**+**1**]** **=** a**[**j**]**  a**[**j**+**1**]** **=** temp**;** // en son öne aldığımızın bir altına o anki elemanı koy |

## Selection Sort:

İlk olarak elimizdeki bütün elemanlardan en küçük olanı bulunur ve ilk sıraya konur. İkinci olarak geriye kalanların en küçüğü bulunur ve ikinci sıraya konur. Bu şekilde elimizde bir eleman kalana kadar küçük elemanlar bulunarak dizilirler.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| En iyi | Ortalama | En kötü | Bellek | Stabil mi? |
| *n^2* | ***n^2*** | ***n^2*** | ***1*** | ***Hayır*** |

(Genel algoritma yapısı)

|  |
| --- |
| **function** selectionSort**(**a**:** list **of** items**)**  int i**,**j**;**  int min**;**  **for** **(**j **=** 0**;** j **<** n**-**1**;** j**++)**  min **=** j**;**  **for** **(** i **=** j**+**1**;** i **<** n**;** i**++)**  **if** **(**a**[**i**]** **<** a**[**min**])**  min **=** i**;**  **if** **(** min !**=** j **)** // Eğer en küçük sayı bulunmuşsa sıraya ekle  swap**(**a**[**j**],** a**[**min**]);** |

## Merge Sort:

Elimizdeki veri dizisinin logaritmik olarak yarı yarıya bölünüp sonrasında ikişerli gruplar halinde sıralanarak birleştirilmesi ardındanda birleştirilen grupların kendi aralarında bütün gruplar birleşene kadar seviye seviye sıralanarak birleştirilmesinden oluşur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| En iyi | Ortalama | En kötü | Bellek | Stabil mi? |
| *n.log(n)* | ***n.log(n)*** | ***n.log(n)*** | ***n*** | ***Evet*** |

(Genel algoritma yapısı)

|  |
| --- |
| **function** merge\_sort**(**list m**) // bu kısım elimizdeki liste parçasını önce bülüp sonrasında**  **// sonrasında bölünmüş parçaları “merge” methodu ile**  **// sıralanarak birleştirilmesini sağlıyor.**  **if** length**(**m**)** **<=** 1  return m  **var** list left**,** right  **var** integer middle **=** length**(**m**)** **/** 2  **for** each x **in** m before middle  add x **to** left  **for** each x **in** m after **or** equal middle  add x **to** right  left **=** merge\_sort**(**left**)**  right **=** merge\_sort**(**right**)**  return merge**(**left**,** right**)**  **function** merge**(**left**,** right**)**  **var** list result  **while** length**(**left**)** **>** 0 **or** length**(**right**)** **>** 0  **if** length**(**left**)** **>** 0 **and** length**(**right**)** **>** 0  **if** first**(**left**)** **<=** first**(**right**)**  append first**(**left**)** **to** result  left **=** rest**(**left**)**  **else**  append first**(**right**)** **to** result  right **=** rest**(**right**)**  **else** **if** length**(**left**)** **>** 0  append first**(**left**)** **to** result  left **=** rest**(**left**)**  **else** **if** length**(**right**)** **>** 0  append first**(**right**)** **to** result  right **=** rest**(**right**)**  **end** **while**  return result |

## Quick Sort:

Şu ana kadar bilinen en gözde ve hızlı algoritmadır. Uygulama adımları şu şekilde sıralanabilir:

* Diziden herhangi bir eleman al(pivot eleman).
* Pivot elemanından küçük olanları bir diziye, büyükleri bir diziye topla.
* Bu alt dizilerden yukarıdaki gibi pivot elemanlar seçip aynı işlemi uygula. İç içe en küçük parçalara ulaşana kadar bu yöntemi sürdür.
* Oluşan dizicikleri birleştir.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| En iyi | Ortalama | En kötü | Bellek | Stabil mi? |
| *n.log(n)* | ***n.log(n)*** | ***n^2*** | ***log(n)~n*** | ***Evet*** |

(Genel algoritma yapısı)

|  |
| --- |
| **function** quicksort**(**int low**,** int high**,** list numbers**)**  int i **=** low**,** j **=** high**;**  int pivot **=** numbers**[**low **+** **(**high**-**low**)/**2**];**  **while** **(**i **<=** j**)**  **while** **(**numbers**[**i**]** **<** pivot**)**  i**++;**  **while** **(**numbers**[**j**]** **>** pivot**)**  j**--;**  **if(**i **<=** j**)**  swap**(**i**,** j**,** numbers**);**  i**++;**  j**--;**  **if** **(**low **<** j**)**  quicksort**(**low**,** j**);**  **if** **(**i **<** high**)**  quicksort**(**i**,** high**);**  **function** swap**(**int i**,** int j**,** list numbers**) // yerlerini değiştir**  int temp **=** numbers**[**i**];**  numbers**[**i**]** **=** numbers**[**j**];**  numbers**[**j**]** **=** temp**;** |

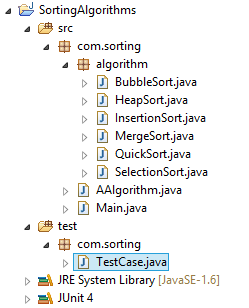
## Heap Sort:

Verilerden bir heap tree oluşturulur. Sonrasında sırayla kökteki elemanlar çekilir. Böylece datamız sıralanmış olunur.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| En iyi | Ortalama | En kötü | Bellek | Stabil mi? |
| *n.log(n)* | ***n.log(n)*** | ***n.log(n)*** | ***1*** | ***Hayır*** |

|  |
| --- |
| **private** Integer**[]** **array;**  **private** int largest**;**  **private** int sol**;**  **private** int n**;**  **private** int sag**;**  **function** heapSort**(**Integer**[]** **array)**  this**.array** **=** **array;**  heapOlustur**();**  **for** **(**int i **=** n**;** i **>** 0**;** i**--)**  swap**(**0**,** i**);**  n **=** n **-** 1**;**  enBuyuk**(**0**);**    return **array;**  **function** heapOlustur**()**  n **=** **array.**length **-** 1**;**  **for** **(**int i **=** n **/** 2**;** i **>=** 0**;** i**--)**  enBuyuk**(**i**);**  **function** swap**(**int i**,** int j**)**  int t **=** **array[**i**];**  **array[**i**]** **=** **array[**j**];**  **array[**j**]** **=** t**;**  **function** enBuyuk**(**int i**)**  sol **=** 2 **\*** i**;**  sag **=** 2 **\*** i **+** 1**;**    **if** **(**sol **<=** n **&&** **array[**sol**]** **>** **array[**i**])**  largest **=** sol**;**  **else**  largest **=** i**;**  **if** **(**sag **<=** n **&&** **array[**sag**]** **>** **array[**largest**])**  largest **=** sag**;**  **if** **(**largest !**=** i**)**  swap**(**i**,** largest**);**  enBuyuk**(**largest**);** |

# Algoritmaların Analizi



Algoritmalar Java programlama dilinde soldaki resimde gözüken hiyerarşide yazıldı.(Kaynak kodlarını eklerde bulabilirsiniz)

JUNIT tabanlı **TestCase.java**’da algoritmalara hız testi uygulanacak biçimde her biri için bir test hazırlandı.

“**Main.java**” da ise yukarıda belirtilen TestCase sırasıyla 1, 10, 100, 1000, 5000, 10000, 50000, 100000, 1000000 eleman sayılarında eleman içeren diziler için test çalıştırıldı ve bir dosyaya kaydedildi.

“**Worst case”** için her bir dizi tersten sıralı olarak algoritmalara uygulandı.

Test sonuçları aşağıdaki gibidir:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Eleman Sayısı** | **Bubble Sort** | **Heap Sort** | **Insertion Sort** | **Merge Sort** | **Quick Sort** | **Selection Sort** |
| 1 | 0,026684 | 0,020116 | 0,008621 | 0,013958 | 0,023811 | 0,014368 |
| 10 | 0,027505 | 0,027095 | 0,015189 | 0,020937 | 0,011084 | 0,0234 |
| 100 | 0,871964 | 0,249602 | 0,199928 | 0,491814 | 0,049674 | 1,066555 |
| 1000 | 13,764225 | 3,048179 | 7,818941 | 1,836708 | 0,601836 | 15,507743 |
| 5000 | 69 | 1 | 17 | 0,621131 | 0,126854 | 112 |
| 10000 | 340,056982 | 1,86914 | 69,510813 | 0,971312 | 0,24057 | 279,965762 |
| 50000 | 8863 | 17 | 1582 | 5 | 1 | 5467 |
| 100000 | 52531,72734 | 34,779628 | 6019,242738 | 16,726193 | 3,329803 | 62097,13327 |
| 1000000 | 5769351,34 | 368,493521 | 907838,0811 | 385,541159 | 45,005501 | 4525921,923 |

Grafiksel çıktı:

Yatay grafiksel çıktı:

***Sonuç:*** Grafiklerden de anlaşılacağı üzere yüksek elemanlı dizilerde hız açısından en hızlıdan en yavaşa olan sıralama aşağıdaki gibi tespit edilmiştir.

**Quick Sort**, **Merge Sort**, **Heap Sort**, **Insertion Sort**, **Selection Sort**, **Bouble Sort**

# Ekler

## 1. TestCase.java:

|  |
| --- |
| package com**.**sorting**;**  **import** java**.**util**.**HashMap**;**  **import** java**.**util**.**Map**;**  **import** org**.**junit**.**After**;**  **import** org**.**junit**.**Before**;**  **import** org**.**junit**.**Rule**;**  **import** org**.**junit**.**Test**;**  **import** org**.**junit**.**rules**.**TestName**;**  public class TestCase **{**  AAlgorithm algorithm**;**  @Rule  public TestName name **=** **new** TestName**();**  Integer**[]** numbers**;**  public static int SIZE **=** 0**;**  public static Map**<**String**,** Double**>** map **=** **new** HashMap**<**String**,** Double**>();**  long start**;**  @Before  public void setUp**()** **throws** Exception **{**  algorithm **=** createObject**(**Class**.**forName**(**"com.sorting.algorithm." **+** name**.**getMethodName**()));**  numbers **=** generateArray**(**SIZE**);**  start **=** System**.**nanoTime**();**  **}**  @After  public void tearDown**()** **throws** Exception **{**  double duration **=** **(**System**.**nanoTime**()** **-** start**)** **/** 1000000D**;**  map**.**put**(**name**.**getMethodName**(),** duration**);**  **}**  @Test  public void BubbleSort**()** **{**  algorithm**.**sort**(**numbers**);**  **}**  @Test  public void HeapSort**()** **{**  algorithm**.**sort**(**numbers**);**  **}**  @Test  public void InsertionSort**()** **{**  algorithm**.**sort**(**numbers**);**  **}**  @Test  public void MergeSort**()** **{**  algorithm**.**sort**(**numbers**);**  **}**  @Test  public void QuickSort**()** **{**  algorithm**.**sort**(**numbers**);**  **}**  @Test  public void SelectionSort**()** **{**  algorithm**.**sort**(**numbers**);**  **}**  @SuppressWarnings**(**"rawtypes"**)**  public static AAlgorithm createObject**(**Class klass**)** **{**  **try** **{**  **return** **(**AAlgorithm**)** klass**.**newInstance**();**  **}**  **catch** **(**Exception e**)** **{**  e**.**printStackTrace**();**  **return** **null;**  **}**  **}**  public static Integer**[]** generateArray**(**Integer size**)** **{**  Integer**[]** array **=** **new** Integer**[**size**];**  **for** **(**int i **=** size **-** 1**,** j **=** 0**;** i **>=** 0**;** i**--,** j**++)** **{**  array**[**j**]** **=** i**;**  **}**  **return** array**;**  **}**  **}** |

## 2. Main.java:

|  |
| --- |
| package com**.**sorting**;**  **import** java**.**io**.**File**;**  **import** java**.**io**.**FileNotFoundException**;**  **import** java**.**io**.**PrintWriter**;**  **import** org**.**junit**.**runner**.**JUnitCore**;**  public class Main **{**  public static void main**(**String**[]** args**)** **throws** FileNotFoundException **{**  PrintWriter print **=** **new** PrintWriter**(**System**.**getProperty**(**"user.home"**)** **+** File**.**separator **+** "Desktop" **+** File**.**separator **+** "x.csv"**);**  TestCase**.**SIZE **=** 1**;**  String line **=** ""**;**  int repeat **=** 7**;**  **for** **(**int i **=** 0**;** i **<** repeat**;** i**++)** **{**  JUnitCore**.**runClasses**(**TestCase**.**class**);**  print**.**print**(**line **+** TestCase**.**SIZE **+** "\t" **+** TestCase**.**map**.**get**(**"BubbleSort"**)** **+** "\t" **+** TestCase**.**map**.**get**(**"HeapSort"**)** **+** "\t" **+** TestCase**.**map**.**get**(**"InsertionSort"**)** **+** "\t"  **+** TestCase**.**map**.**get**(**"MergeSort"**)** **+** "\t" **+** TestCase**.**map**.**get**(**"QuickSort"**)** **+** "\t" **+** TestCase**.**map**.**get**(**"SelectionSort"**));**  line **=** "\n"**;**  TestCase**.**map**.**clear**();**  TestCase**.**SIZE **\*=** 10**;**  **}**  print**.**close**();**  System**.**out**.**println**();**  **}**  **}** |

## 3. BubbleSort.java:

|  |
| --- |
| package com**.**sorting**.**algorithm**;**  **import** com**.**sorting**.**AAlgorithm**;**  public class BubbleSort **extends** AAlgorithm **{**  @Override  public Integer**[]** sort**(**Integer**[]** array**)** **{**  int length **=** array**.**length**;**  **for** **(**int i **=** 0**;** i **<** length**;** i**++)** **{**  **for** **(**int j **=** length **-** 1**;** j **>** i**;** j**--)** **{**  **if** **(**array**[**j**]** **<** array**[**j **-** 1**])** **{**  swap**(**j**,** j **-** 1**,** array**);**  **}**  **}**  **}**  **return** array**;**  **}**  **}** |

## 4. HeapSort.java:

|  |
| --- |
| package com**.**sorting**.**algorithm**;**  **import** com**.**sorting**.**AAlgorithm**;**  public class HeapSort **extends** AAlgorithm **{**  private Integer**[]** array**;**  private int largest**;**  private int left**;**  private int n**;**  private int right**;**  public void buildheap**()** **{**  n **=** array**.**length **-** 1**;**  **for** **(**int i **=** n **/** 2**;** i **>=** 0**;** i**--)** **{**  maxheap**(**i**);**  **}**  **}**  public void exchange**(**int i**,** int j**)** **{**  int t **=** array**[**i**];**  array**[**i**]** **=** array**[**j**];**  array**[**j**]** **=** t**;**  **}**  public void maxheap**(**int i**)** **{**  left **=** 2 **\*** i**;**  right **=** 2 **\*** i **+** 1**;**  **if** **(**left **<=** n **&&** array**[**left**]** **>** array**[**i**])** **{**  largest **=** left**;**  **}**  **else** **{**  largest **=** i**;**  **}**  **if** **(**right **<=** n **&&** array**[**right**]** **>** array**[**largest**])** **{**  largest **=** right**;**  **}**  **if** **(**largest **!=** i**)** **{**  exchange**(**i**,** largest**);**  maxheap**(**largest**);**  **}**  **}**  @Override  public Integer**[]** sort**(**Integer**[]** array**)** **{**  **this.**array **=** array**;**  buildheap**();**  **for** **(**int i **=** n**;** i **>** 0**;** i**--)** **{**  exchange**(**0**,** i**);**  n **=** n **-** 1**;**  maxheap**(**0**);**  **}**  **return** array**;**  **}**  **}** |

## 5. InsertionSort.java:

|  |
| --- |
| package com**.**sorting**.**algorithm**;**  **import** com**.**sorting**.**AAlgorithm**;**  public class InsertionSort **extends** AAlgorithm **{**  @Override  public Integer**[]** sort**(**Integer**[]** array**)** **{**  **for** **(**int i **=** 1**;** i **<** array**.**length**;** i**++)** **{**  int temp **=** array**[**i**];**  int j**;**  **for** **(**j **=** i **-** 1**;** j **>=** 0 **&&** temp **<** array**[**j**];** j**--)** **{**  array**[**j **+** 1**]** **=** array**[**j**];**  **}**  array**[**j **+** 1**]** **=** temp**;**  **}**  **return** array**;**  **}**  **}** |

## 6. MergeSort.java:

|  |
| --- |
| package com**.**sorting**.**algorithm**;**  **import** com**.**sorting**.**AAlgorithm**;**  public class MergeSort **extends** AAlgorithm **{**  private Integer**[]** numbers**;**  private int**[]** helper**;**  private int number**;**  @Override  public Integer**[]** sort**(**Integer**[]** values**)** **{**  **this.**numbers **=** values**;**  number **=** values**.**length**;**  **this.**helper **=** **new** int**[**number**];**  mergesort**(**0**,** number **-** 1**);**  **return** values**;**  **}**  private void mergesort**(**int low**,** int high**)** **{**  **if** **(**low **<** high**)** **{**  int middle **=** low **+** **(**high **-** low**)** **/** 2**;**  mergesort**(**low**,** middle**);**  mergesort**(**middle **+** 1**,** high**);**  merge**(**low**,** middle**,** high**);**  **}**  **}**  private void merge**(**int low**,** int middle**,** int high**)** **{**  **for** **(**int i **=** low**;** i **<=** high**;** i**++)** **{**  helper**[**i**]** **=** numbers**[**i**];**  **}**  int i **=** low**;**  int j **=** middle **+** 1**;**  int k **=** low**;**  **while** **(**i **<=** middle **&&** j **<=** high**)** **{**  **if** **(**helper**[**i**]** **<=** helper**[**j**])** **{**  numbers**[**k**]** **=** helper**[**i**];**  i**++;**  **}**  **else** **{**  numbers**[**k**]** **=** helper**[**j**];**  j**++;**  **}**  k**++;**  **}**  **while** **(**i **<=** middle**)** **{**  numbers**[**k**]** **=** helper**[**i**];**  k**++;**  i**++;**  **}**  **}**  **}** |

## 7. QuickSort.java:

|  |
| --- |
| package com**.**sorting**.**algorithm**;**  **import** com**.**sorting**.**AAlgorithm**;**  public class QuickSort **extends** AAlgorithm **{**  private Integer**[]** numbers**;**  private Integer number**;**  @Override  public Integer**[]** sort**(**Integer**[]** values**)** **{**  **if** **(**values **==** **null** **||** values**.**length **==** 0**)** **{**  **return** values**;**  **}**  **this.**numbers **=** values**;**  number **=** values**.**length**;**  quicksort**(**0**,** number **-** 1**);**  **return** values**;**  **}**  private void quicksort**(**int low**,** int high**)** **{**  int i **=** low**,** j **=** high**;**  int pivot **=** numbers**[**low **+** **(**high **-** low**)** **/** 2**];**  **while** **(**i **<=** j**)** **{**  **while** **(**numbers**[**i**]** **<** pivot**)** **{**  i**++;**  **}**  **while** **(**numbers**[**j**]** **>** pivot**)** **{**  j**--;**  **}**  **if** **(**i **<=** j**)** **{**  swap**(**i**,** j**,** numbers**);**  i**++;**  j**--;**  **}**  **}**  **if** **(**low **<** j**)**  quicksort**(**low**,** j**);**  **if** **(**i **<** high**)**  quicksort**(**i**,** high**);**  **}**  **}** |

## 8. SelectionSort.java:

|  |
| --- |
| package com**.**sorting**.**algorithm**;**  **import** com**.**sorting**.**AAlgorithm**;**  public class SelectionSort **extends** AAlgorithm **{**  @Override  public Integer**[]** sort**(**Integer**[]** array**)** **{**  Integer i**,** j**;**  Integer min**;**  Integer n **=** array**.**length**;**  **for** **(**j **=** 0**;** j **<** n **-** 1**;** j**++)** **{**  min **=** j**;**  **for** **(**i **=** j **+** 1**;** i **<** n**;** i**++)** **{**  **if** **(**array**[**i**]** **<** array**[**min**])** **{**  min **=** i**;**  **}**  **}**  **if** **(**min **!=** j**)** **{**  swap**(**j**,** min**,** array**);**  **}**  **}**  **return** array**;**  **}**  **}** |

## AAlgorithm.java (Abstract Class):

|  |
| --- |
| package com**.**sorting**;**  public abstract class AAlgorithm **{**  public abstract Integer**[]** sort**(**Integer**[]** array**);**  public Integer**[]** swap**(**int i**,** int j**,** Integer**[]** array**)** **{**  int temp **=** array**[**i**];**  array**[**i**]** **=** array**[**j**];**  array**[**j**]** **=** temp**;**  **return** array**;**  **}**  **}** |

# Kaynakça

1. <http://en.wikipedia.org/wiki/Sorting_algorithm>
2. <http://www.vogella.com/articles/JavaAlgorithmsQuicksort/article.html>
3. <http://www.vogella.com/articles/JavaAlgorithmsMergesort/article.html>
4. <http://en.wikipedia.org/wiki/Bubble_sort>
5. <http://www.cprogramming.com/tutorial/computersciencetheory/sorting1.html>
6. <http://www.youtube.com/watch?v=EeQ8pwjQxTM>
7. <http://en.wikipedia.org/wiki/Merge_sort>
8. <http://en.wikipedia.org/wiki/Quicksort>
9. <http://www.cs.auckland.ac.nz/~jmor159/PLDS210/qsort.html>
10. <http://www.sorting-algorithms.com/quick-sort>
11. <http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Sorting/quickSort.htm>
12. <http://algs4.cs.princeton.edu/23quicksort/>
13. <http://en.wikipedia.org/wiki/Heapsort>
14. <http://www.youtube.com/watch?v=6NB0GHY11Iw>