

Visualisering af algoritmer til sortering

Jacob Aae Mikkelsen

Vejleder: Joan Boyar

8. juni 2007

Indhold

1 Indledning	2
2 Krav og design	3
Kravspecifikation	3
Sorteringsalgoritmerne	4
3 Implementation	7
GUI klassen	7
SourceCodeCanvas klassen	7
ExplainArea klassen	8
Entry klassen	8
DrawingRegion klassen	8
TimedExplanation klassen	8
FilledArrays klassen	8
Algorithm klassen	8
Sorteringsalgoritme klasserne	8
Kommentarer til koden	9
4 Forslag til opgaver	10
5 Test	11
6 Konklusion	12
A Kildekode	14
Main.java	14
GUI.java	15
SourceCodeCanvas.java	27
DrawingRegion.java	28
ExplainArea.java	30
TimedExplanation.java	32
Entry.java	34
FilledArrays.java	35
Algorithm.java	37
InsertionSort.java	38
SelectionSort.java	42
QuickSort.java	46
MergeSort.java	51
RandomizedQuickSort.java	57

1 Indledning

Denne rapport beskriver design, implementation og test af en applikation der kan bruges til at visualisere 5 sorteringsalgoritmer, samt give oplysninger om deres effektivitet.

Projektet er udarbejdet af Jacob Aae Mikkelsen med kyndig vejledning af Joan Boyar på IMADA, Syddansk Universitet, forår 2007. Projektet skulle fylde 1,5 ECTS point, og dette vurderes at være være opfyldt.

Motivation for at lave projektet kom efter faget “Introduktion til datalogi” hvor applets på diverse hjemmesider blev brugt til noget lignende, men uden sikkerhed for at disse hjemmesider stadig eksisterede fra år til år. Med produktet af dette projekt, er det problem forhåbenligt løst.

I afsnit 2 beskrives de krav der var til opgaven, sorteringsalgoritmerne er angivet i pseudokode og det er beskrevet hvor der tælles sammenligninger og kopier i algoritmerne.

Afsnit 3 beskriver de enkelte klasser i implementationen, og de dele af implementationen der er interessant.

I afsnit 4 er stillet opgaver som eksempler på hvad man kan bruge programmet til, og sluttligt opsummeres rapportens dele i afsnit 6.

Pseudokoden til algoritmerne i afsnit 2 er typografisk opsat som i [1], ved brug af `clrscode.sty` af Thomas Cormen. Applikationen er lavet på engelsk, men til målgruppen ’nystartede datalogistuderende’ mener jeg ikke at dette er et problem.

2 Krav og design

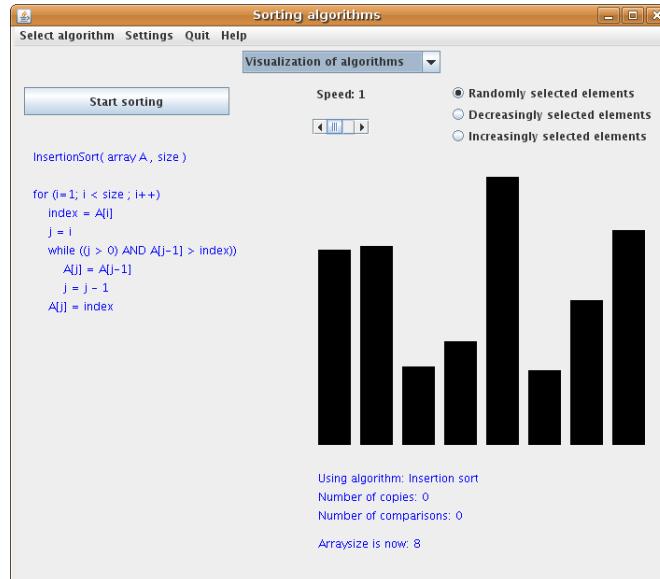
Kravspecifikation

Der var følgende krav til projektet:

- Insertion sort, selection sort, merge sort og quick sort skulle implementeres.
- Mulighed for at tage tid på algoritmerne og mulighed for at se dem i brug visuelt.
- Mulighed for at følge kildekoden og arrayet mens algoritmen arbejder.
- Input skulle kunne vælges mellem sorteret, baglæns sorteret eller randomiseret.
- Kunne måle antal sammenligninger og antal gange elementerne bliver kopieret.
- Mulighed for at vælge arrays af forskellige størrelser

Designet skulle gerne være let at forstå og bruge, så koncentrationen kan bruges på at forstå algoritmerne, og tænke over problemstillinger med valget mellem forskellige sorteringsalgoritmer. Der er derfor forsøgt at holde funktionaliteten simpel.

Valg af algoritme og arraystørrelse foregår ved hjælp af menuen, mens start/-stop, hastighed og den oprindelige arrangement af elementer i arrayet kan vælges via knapper på den grafiske brugerflade. Figur 1 viser et billede af programmet, som det ser ud fra start.



Figur 1: Screenshot af programmet fra start.

Der er anvendt en klasse til at afsnittet med søgerne, der repræsenterer elementerne i arrayet (**DrawingRegion**), en klasse til afsnittet med kildekoden

(`SourceCodeCanvas`), og en klasse med det formål at vise informationerne nederst (`ExplanationArea`). Der ud over er der en klasse der holder sammen på det hele (`GUI`), mere herom i section 3.

Sorteringsalgoritmerne

Algoritmerne der er implementeret findes i flere forskellige udgaver, og er derfor gengivet i pseudokode form her, også med angivelse af hvor der tælles sammenligninger og hvor der tælles kopieringer.

InsertionSort

I `Insertion sort` algoritmen tælles kopier i linie 2,5 og 7, og sammenligninger i linie 4. For sammenligningerne, tælles de op inden i `while` løkken, og udenfor hvis `while` løkken fejler på $j > 0$.

```
INSERTIONSORT( $A, size$ )
1 for  $j \leftarrow 1$  to  $size - 1$ 
2     do  $index \leftarrow A[i]$ 
3          $j \leftarrow i$ 
4         while  $j > 0$  and  $A[j - 1] > index$ 
5             do  $A[j] \leftarrow A[j - 1]$ 
6              $j \leftarrow j - 1$ 
7      $A[j] \leftarrow index$ 
```

SelectionSort

I `Selection sort` algoritmen tælles sammenligningerne i linie 4 og kopier i linie 6. Kopier tælles op med tre, da `Swap` funktionen laver tre kopier for at bytte om på to elementer.

```
SELECTIONSORT( $A, size$ )
1 for  $i \leftarrow 0$  to  $size - 1$ 
2     do  $min \leftarrow i$ 
3         for  $j \leftarrow i + 1$  to  $size$ 
4             do if  $A[j] < A[min]$ 
5                 then  $min \leftarrow j$ 
6             EXCHANGE( $A[i], A[min]$ )
```

QuickSort

I `Quicksort` algoritmen sker en stor del af arbejdet, også alle sammenligninger og kopier, i hjælpefunktionen `Partition`.

```
QUICKSORT( $A, p, r$ )
1 if  $p < r$ 
2     then  $q \leftarrow \text{PARTITION}(A, p, r)$ 
3         QUICKSORT( $A, p, q - 1$ )
4         QUICKSORT( $A, q + 1, r$ )
```

I **Partition** tælles sammenligninger i linie 5, og kopier i linie 1,7 og 8. I linie 7 og 8, der benytter **Swap**, tælles op med tre.

```
PARTITION( $A, p, r$ )
1    $x \leftarrow A[r]$ 
2    $i \leftarrow p - 1$ 
3   for  $j \leftarrow p$  to  $r$ 
4     do
5       if  $A[j] \leq x$ 
6         then  $i \leftarrow i + 1$ 
7           EXCHANGE( $A[i], A[j]$ )
8   EXCHANGE( $A[i + 1], A[r]$ )
9   return  $i + 1$ 
```

Randomiseret QuickSort

Den eneste forskel fra Randomiseret QuickSort til almindelig Quicksort er at der som det første i **Partition** byttes om på $A[r]$ og et tilfældigt valgt element mellem p og r . Her tælles dette for tre kopier, da der er tilsvarende til en **Swap** funktion.

MergeSort

Mergesort algoritmen er rekursiv med **Merge** som hjælpefunktion, og det er **Merge** der laver den tunge del af arbejdet.

```
MERGESORT( $A, lo, hi$ )
1   if  $lo < hi$ 
2     then  $m \leftarrow (lo + hi)/2$ 
3       MERGESORT( $A, lo, m$ )
4       MERGESORT( $A, m + 1, hi$ )
5       MERGE( $A, lo, m, hi$ )
```

I **Merge** tælles sammenligningerne op i linie 7, mens kopier tælles i linie 2,8,10 og 14. Denne version af **Merge** er valgt, da det er lettere at se hvor hvilke elementer kommer fra, når der er et helt midlertidigt array under det array der skal sorteres. Jeg foretrak denne frem for andre der har mindre overhead rent pladsmæssigt fordi jeg tror den er lettere at forstå hvis det er første gang man ser på sorteringsalgoritmer.

```
MERGE( $A, lo, m, hi$ )
1  for  $i \leftarrow lo$  to  $hi$ 
2      do  $temp[i] \leftarrow A[i]$ 
3       $i \leftarrow lo$ 
4       $j \leftarrow m + 1$ 
5       $k \leftarrow lo$ 
6      while  $i \leq m$  and  $j \leq hi$ 
7          do if  $temp[i] \leq temp[j]$ 
8              then  $A[k] \leftarrow temp[i]$ 
9                   $i \leftarrow i + 1$ 
10             else  $A[k] \leftarrow temp[j]$ 
11                  $j \leftarrow j + 1$ 
12              $k \leftarrow k + 1$ 
13     while  $i \leq m$ 
14         do  $A[k] \leftarrow temp[i]$ 
15              $i \leftarrow i + 1$ 
16              $k \leftarrow k + 1$ 
```

3 Implementation

Implementationen beskrives i dette afsnit ved at gennemgå de enkelte klasser i programmet samt deres funktion.

GUI klassen

GUI klassen har ansvaret for at samle alle de grafiske elementer, samt at være samlende klasse for de indre klasser der har til opgave at være 'listener' på et menuelement, en knap eller lignende. Den er i sig selv en nedarvning fra JFrame, så det er nemt at sætte den overordnede størrelse, titel og lignende. Den implementerer følgende klasser:

ItemListener Gør at den kan skifte mellem de to layout muligheder: Den visuelle og den tidtagende.

Runnable Med denne kan den køre som en tråd, så det er muligt at lytte på knapperne selv om der er en algoritme der arbejder.

ActionListener Således kan den lytte til de inputs der kommer fra knapper og menuen.

AdjustmentListener Denne er den type **listener** der bruges ved hastigheds knappen.

Bortset fra Runnable, gøres det ved at definere dem som indre klasser. Jeg mener det er en nænere og mere overskuelig måde at gøre det på end at implementere dem som anonyme indre klasser ved de menuelementer de bliver brugt ved.

I GUI klassen er alle tekststrenge samlet som globale variable i toppen af klassen, hvilket gør en eventuel oversættelse af programmet lettere. Algoritmenavnene og layoutmulighederne er lavet **public**, men er til gengæld også **final**, og benyttes af flere af de andre klasser.

Selve funktionaliteten for at køre en algoritme visuelt ser således ud:

```
while(Sorteringen ikke er færdig)
    Opdater den grafiske brugerflade med de nye værdier
    Bed sorteringsalgoritmen om at lave et skridt
    Vent selv i den tid 'speed' variablen angiver
```

Dette er implementeret i **run()** metoden, der implicit bliver kaldt når en tråd oprettes, og dette sker i **startAction()** metoden.

De fire næste klasser hører alle til det visuelle layout format.

SourceCodeCanvas klassen

Klassen **SourceCodeCanvas** 'extends' fra **Canvas** tilhørende AWT biblioteket (den gamle del af JAVA's brugergrænseflade bibliotek). Dette gør det meget simpelt at arbejde med. Klassen har to variable, et linienummer, der bestemmer hvilken linie der skal være rød, og en **ArrayList**, med kildekoden liggende som tekststrenge. Når et objekt fra klassen skal vises eller opdateres på skærmen, bliver **paint** metoden kaldt, med objektets tilhørende **Graphics** objekt. På dette kan der direkte skrives teksten med en angivet farve.

ExplainArea klassen

ExplainArea er implementeret på samme måde som SourceCodeCanvas. Der er blot en spidsfindighed, da antallet af kopier og sammenligninger er af typen double. Typen double har et komma efterfulgt af et nul, hvis de konateneres med en streng, metoden `formatDouble(double d)` fjerner decimalen så længe vi holder os indenfor størrelsen af en integer.

Entry klassen

I den visuelle del af layoutet, er det simple integers der sorteres, men knyttet med dem er information om farven på den viste søjle, og dette er pakket ned i Entry klassen

DrawingRegion klassen

Her i DrawingRegion klassen er igen brugt 'extends' fra Canvas. Den har mulighed for at vise ét eller to arrays, efter behov, dette bestemmes ved at kalde `setIncludeTemp` metoden med en boolean. Bredden af søgerne beregnes ud fra antallet, og søgerne tegnes med Graphics klassens metode `fillRect`.

TimedExplanation klassen

Det der adskiller TimedExplanation fra ExplainArea er primært tekstens indhold og placering. Der er dog sat en større skrifttype med fed.

FilledArrays klassen

FilledArrays klassen indeholder kun to statiske metoder, begge til at lave arrays med en angivne sortering (randomiseret, stigende eller faldende). Den ene metode leverer en ArrayList med Entry elementer, og den anden leverer et almindeligt array med int's.

Algorithm klassen

Algorithm klassen er et interface, der trænger de klasser der implementerer den til at have de rette metoder og i særdeleshed sikrer at alle algoritmer kan oplettes som tråde, da den `extends Runnable`. Det er dette interface der gør det relativt let at tilføje nye algoritmer til programmet, da de metoder der benyttes af de andre klasser er specificeret her.

Sorteringsalgoritme klasserne

De fem algoritmer InsertionSort, SelectionSort, QuickSort, MergeSort og RandomizedQuickSort er som klasser meget ens, og implementerer alle interface't et Algorithm. Da tidtagningen af algoritmerne i forvejen afhænger af operativsystemets afsatte resourcer til JAVA VM, mente jeg det var relevant at der ikke var andet arbejde i sorteringsalgoritmen til dette formål. Alle sorteringsalgoritmerne er derfor implementeret i en 'ren' udgave til tidtagning, og en udgave

der for hver linie pseudokode pauser og opdaterer forklaringer, kildekode linien, kopier og sammenligninger. Til dette formål benyttes `setStatusLineAndWait(int lineno, String statusString)`, der udfører tre opgaver:

1. Opdaterer linienummeret
2. Opdaterer den beskrivende tekst
3. Venter indtil der bliver meldt klar til at tage et nyt skridt.

`Mergesort` er den eneste af algoritmerne der ikke er inplace sortering, og det er derfor at metoden `getTempList()` er implementeret, de fire andre klasser returnerer blot `null`, men for at interfacet skulle være ens for algoritmerne var dette nødvendigt.

Kommentarer til koden

Der er i koden benyttet de 'Coding Guidelines' som er beskrevet i af Michael Kölking i kurset SW08 som findes i [2].

I sorterings algoritmernes viosuelle del er klassen `ArrayList` benyttet, dette kunne med fordel være ændret til et almindeligt array, da vi kender antal elementer fra start. Med JAVA's implementation tror jeg overhead er begrænset, og jeg har valgt ikke at lave denne omskrivning.

Da Quicksort og Randomiseret quicksort minder meget om hinanden, kunne de være slået sammen til én klasse med en parameter. Jeg har valgt ikke at gøre dette, da det giver en bedre mulighed for at lave en anden implementation af randomiseret quicksort senere på denne måde. Jeg mener også at det letter læseligheden af koden at det er to forskellige klasser.

4 Forslag til opgaver

Her er nogle forslag til opgaver der kan stilles i forbindelse med programmet. Opgaverne retter sig imod studerende der skal starte med at arbejde med algoritmer, for at give en intuitiv forståelse for at problemer kan have forskellige algoritmiske løsninger, der adskiller sig kraftigt fra hinanden alt efter problemets input.

- Prøv at kør insertion sort algoritmen med randomiseret input og arraystørrelse 8. Beskriv hvordan algoritmen virker. Prøv at kør algoritmen et par gange, hvorfor får du ikke det samme antal hver gang? Gør det samme med selection sort. Hvad er forskellen mellem de to algoritmer? Prøv at sammenligne med sorteret input og baglæns sorteret input. Hvad bemærker du?
- Skift til tidtagning, og vælg arraystørrelse 50.000. Prøv at kør Insertion sort og selection sort med sorteret data og randomiseret data. Er der forskel? Hvilken algoritme vil du vælge hvis du skulle prøve med 500.000 elementer?
- Gå tilbage til visuel sortering og vælg Quicksort med arraystørrelse 16 og randomiseret data. Hvordan virker algoritmen? Det røde element i partition kaldes pivot element, og skal helst dele arrayet i to lige store dele. Hvis dette sker kører algoritmen i $\Theta(n \log n)$. Prøv at kør algoritmen på sorteret data. Hvad sker der? Hvordan kan dette problem løses? Skift evt. til randomiseret quicksort for at se et forslag.
- Prøv at vælg Merge sort algoritmen med randomiseret data og arraystørrelsen 16. Sæt eventuelt hastigheden op og se hvordan algoritmen virker. Hvad adskiller Merge sort fra de andre algoritmer?
- Skift til tidstagning, og prøv de fem algoritmer med input størrelse 50.000 på først randomiseret input, og bagefter på sorteret input. Noter tiden i hvert tilfælde. Hvilke algoritmer er effektive til hvilke opgaver? Hvilken algoritme ville du vælge til at sortere 500.000 elementer? Hvilke ville du ikke vælge?

5 Test

Med arraystørrelse 8 er der lavet test på alle algoritmerne med baglæns sorteret data og sorteret.

	Visuelt		Tidtaget	
	Kopier	Sammenligninger	kopier	sammenligninger
Insertion sort	42	28	42	28
Selection sort	21	28	21	28
Quick sort	64	28	64	28
Merge sort	48	12	48	12
Rand. quick sort	62	16	56	16

Figur 2: Baglæns sorteret data med arraystørrelse 8

	Visuelt		Tidtaget	
	Kopier	Sammenligninger	kopier	sammenligninger
Insertion sort	14	7	14	7
Selection sort	21	28	21	28
Quick sort	112	28	112	28
Merge sort	36	12	36	12
Rand. quick sort	50	14	47	14

Figur 3: Sorteret data med arraystørrelse 8

Vi ved at Selection sort bruger et fast antal sammenligninger[3] ud fra formlen:

$$\sum_{i=2}^n (i - 1) = \frac{1}{2}n^2 - \frac{1}{2}n$$

Dette stemmer overens med applikationens, for $n = 8$ fåes 24 i både visuel og tidstagnings displayet, og for $n = 100$ fåes 4950. Dette sammenholdt med dataerne i forhold til hinanden og kendskabet til algoritmerne indikerer at programmet virker korrekt.

6 Konklusion

Programmet der er udviklet i projektet opfylder kravene der er stillet, endda med en ekstra algoritme. Den grafiske brugerflade er let at anvende, og jeg tror at programmet kan hjælpe med forståelsen for algoritmers forskellighed, og sorteringsalgoritmernes virkemåde.

Det har været en udfordring at programmere den grafiske del, og at få interaktionen mellem trådene til at fungere. Der har også være et par 'A-ha' oplevelser imellem i implementationen af de forskellige sorteringsalgoritmer, der findes i mange vidt forskellige udgaver. Slutteligt var det en udfordring at skrive pseudokoden i latex for algoritmerne.

Jeg synes generelt at projektet er vellykket, og håber at programmet finder anvendelse i forbindelse med præsentationen af algoritmer i 'Introduktion til Datalogi' men måske også for matematikkere og mat.øk-studerende i 'Algoritmer og datastrukturer'.

Der er mulighed for at udvide programmet med flere algoritmer, uden at dette er en stor opgave. Til gengæld har jeg valgt ikke at implementere en pause-knap, da dette ville være en større omskrivning af både den grafiske del, men også kontrol-flowet i algoritmerne. Jeg har vurderet at den kunne undværes, og ladet den ligge som en mulig udvidelse til senere.

Programmet kan hentes fra:

<http://www.imada.sdu.dk/~kok04/sorting.jar>

Kildekoden som zip-fil kan hentes på:

<http://www.imada.sdu.dk/~kok04/Sorting.zip>

Litteratur

- [1] Cormen, Leiserson, Rivest og Stein *Introduction to Algorithms* 2nd ed. The MIT Press.
- [2] Barnes og Kölbing *Objects First with JAVA using BlueJ* 2nd. ed. Pearson Educational Limited.
- [3] Joan Boyar, *Ugeseddel 6 fra DM501, E06*
- [4] <http://linux.wku.edu/~lamonml/algor/sort/>

A Kildekode

Main.java

```
1 public class Main {  
2     public Main()  
3     {  
4         new GUI();  
5     }  
6  
7     /**  
8      * @param args  
9      */  
10    public static void main( String[] args )  
11    {  
12        new Main();  
13    }  
14 }  
15 }
```

GUI.java

```

1 import java.awt.BorderLayout;
2 import java.awt.CardLayout;
3 import java.awt.Container;
4 import java.awt.Dimension;
5 import java.awt.FlowLayout;
6 import java.awt.GridLayout;
7 import java.awt.event.ActionEvent;
8 import java.awt.event.ActionListener;
9 import java.awt.event.AdjustmentEvent;
10 import java.awt.event.AdjustmentListener;
11 import java.awt.event.ItemEvent;
12 import java.awt.event.ItemListener;
13 import javax.swingBoxLayout;
14 import javax.swing.ButtonGroup;
15 import javax.swing.JButton;
16 import javax.swing.JComboBox;
17 import javax.swing.JFrame;
18 import javax.swing.JLabel;
19 import javax.swing.JMenu;
20 import javax.swing.JMenuBar;
21 import javax.swing.JMenuItem;
22 import javax.swing.JOptionPane;
23 import javax.swing.JPanel;
24 import javax.swing.JRadioButton;
25 import javax.swing.JScrollBar;
26
27 public class GUI extends JFrame implements AdjustmentListener,
   ActionListener, Runnable, ItemListener
28 {
29     public final static String VISUAL = "Visualization_of_
       algorithms";
30     public final static String TIMED = "Timetaking_of_the_
       algorithms";
31     public final static String INSERTIONSORT = "Insertion_sort";
32     public final static String SELECTIONSORT = "Selection_sort";
33     public final static String QUICKSORT = "Quick_sort";
34     public final static String MERGESORT = "Merge_sort";
35     public final static String RANDQUICKSORT = "Randomized_quick_
       sort";
36
37     private final static String START = "Start_sorting";
38     private final static String STOP = "Stop_sorting";
39     private final static String RESET = "Reset";
40     private final static String TITLE = "Sorting_algorithms";
41     private final static String SELECT = "Select_algorithm";
42     private final static String SETTINGS = "Settings";
43     private final static String ARRSIZE = "Choose_arraysize";
44     private final static String QUIT = "Quit";
45     private final static String HELP = "Help";
46     private final static String ABOUT = "About_Sorting_algorithms";
47     private final static String SPEED = "Speed:";
48     private final static String RANDOM = "Randomly_selected_
       elements";
49     private final static String INCREASING = "Increasingly_selected_
       elements";

```

```

50     private final static String DECREASING = "Decreasingly selected
      _elements";
51     private final static String INPUT = "Input the size of the_
      array.\n" +
52                                         "In visual mode select_
      array size between 1 and
      25.\n" +
53                                         "in timed mode between 1_
      and 500.000";
54     private final static String PLEASE = "Please write a number_
      using only digits";
55     private final static String PLEASE2 = "Please select a number_
      between 1 and 25 when using visual display.";
56     private final static String PLEASE3 = "Please select a number_
      between 1 and 500000 when using timed display.";
57     private final static String ERROR = "Error";
58     private final static String INFO = "This application was_
      developed by Jacob Aae Mikkelsen\n" +
59                                         "with competent guidance by
      Joan Boyar on IMADA --
      \n" +
60                                         "University of Southern
      Denmark, SDU -- spring -
      2007.\n\n" +
61                                         "Please feel free to use_
      this application if it
      is of any help\n\n" +
62                                         "Any comments, please_
      contact_
      kokken@grydeske.dk";
63
64     private static final long serialVersionUID = 1L;
65     private DrawingRegion canvas;
66     private ExplainArea explanation;
67     private Algorithm thisAlg;
68     private int thisAlgType;
69     private JButton runButton;
70     private JButton runButton2;
71     private JScrollBar speedScrollBar;
72     private int speed;
73     private int arraySize;
74     private JLabel speedLabel;
75     private Thread guiThread;
76     private Thread alg;
77     private int startConfig;
78     private SourceCodeCanvas scc;
79     private JPanel cards;
80     private String mode;
81     private TimedExplanation timedExpArea;
82
83     public GUI()
84     {
85         mode = VISUAL;
86         startConfig = 0;
87         arraySize = 8;
88         thisAlgType = 1;
89

```

```

90         this . setTitle (TITLE) ;
91         this . setSize (new Dimension (800 ,600)) ;
92         addMenu () ;
93
94         JPanel comboBoxPane = new JPanel () ;
95         String comboBoxItems [] = { VISUAL, TIMED } ;
96         JComboBox cb = new JComboBox (comboBoxItems) ;
97         cb . setEditable (false) ;
98         cb . addItemListener (this ) ;
99         comboBoxPane . add (cb) ;
100
101        //Create the displays .
102        JPanel visualDisplay = addVisualDisplay () ;
103        JPanel timedDisplay = addTimedDisplay () ;
104
105        //Create the panel that contains the "cards " .
106        cards = new JPanel (new CardLayout ()) ;
107        cards . add (visualDisplay , VISUAL) ;
108        cards . add (timedDisplay , TIMED) ;
109
110        Container pane = this . getContentPane () ;
111        pane . add (comboBoxPane , BorderLayout . PAGE _START) ;
112        pane . add (cards , BorderLayout . CENTER) ;
113
114        initialize () ;
115        update () ;
116        setLocationRelativeTo (null) ;
117        setVisible (true) ;
118    }
119
120    /**
121     * Adds the menubar to the frame
122     */
123    private void addMenu ()
124    {
125        JMenuBar menuBar = new JMenuBar () ;
126        this . setJMenuBar (menuBar) ;
127
128        JMenu algChooser = new JMenu (SELECT) ;
129        menuBar . add (algChooser) ;
130        JMenuItem insertionSort = new JMenuItem (INSERTIONSORT) ;
131        insertionSort . setActionCommand (INSERTIONSORT) ;
132        insertionSort . addActionListener (new algChooserListener ()) ;
133        algChooser . add (insertionSort) ;
134
135        JMenuItem selectionSort = new JMenuItem (SELECTIONSORT) ;
136        selectionSort . setActionCommand (SELECTIONSORT) ;
137        selectionSort . addActionListener (new algChooserListener ()) ;
138        algChooser . add (selectionSort) ;
139
140        JMenuItem quickSort = new JMenuItem (QUICKSORT) ;
141        quickSort . setActionCommand (QUICKSORT) ;
142        quickSort . addActionListener (new algChooserListener ()) ;
143        algChooser . add (quickSort) ;
144
145        JMenuItem mergeSort = new JMenuItem (MERGESORT) ;
146        mergeSort . setActionCommand (MERGESORT) ;

```

```

147     mergeSort.addActionListener(new algChooserListener()) ;
148     algChooser.add(mergeSort) ;
149
150     JMenuItem randQuickSort = new JMenuItem(RANDQUICKSORT) ;
151     randQuickSort.setActionCommand(RANDQUICKSORT) ;
152     randQuickSort.addActionListener(new algChooserListener()) ;
153     algChooser.add(randQuickSort) ;
154
155     JMenu sizeChooser = new JMenu(SETTINGS) ;
156
157     menubar.add(sizeChooser) ;
158     JMenuItem size = new JMenuItem(ARRAYSIZE) ;
159     size.addActionListener(new sizeChooserListener()) ;
160     sizeChooser.add(size) ;
161
162     JMenu quitmenu = new JMenu(QUIT) ;
163     menubar.add(quitmenu) ;
164     JMenuItem quit = new JMenuItem(QUIT) ;
165     quit.addActionListener(new QuitActionListener()) ;
166     quitmenu.add(quit) ;
167
168     JMenu helpmenu = new JMenu(HELP) ;
169     menubar.add(helpmenu) ;
170     JMenuItem about = new JMenuItem(ABOUT) ;
171     about.addActionListener(new HelpActionListener()) ;
172     helpmenu.add(about) ;
173 }
174
175 private JPanel addVisualDisplay()
176 {
177     JPanel visualDisplay =new JPanel() ;
178     Container displayPane = new Container() ;
179     displayPane.setLayout(new BorderLayout()) ;
180
181     //North
182     Container northContainer = new Container() ;
183     northContainer.setSize(new Dimension(800,80)) ;
184     GridLayout gl = new GridLayout(1,3) ;
185     northContainer.setLayout(gl) ;
186
187     //Run button
188     Container buttonContainer = new Container() ;
189     buttonContainer.setSize(new Dimension(220,80)) ;
190     FlowLayout fl = new FlowLayout() ;
191     fl.setAlignment(FlowLayout.CENTER) ;
192     buttonContainer.setLayout(fl) ;
193     runButton = new JButton(START) ;
194     runButton.setPreferredSize(new Dimension(220,30)) ;
195     runButton.addActionListener(new runActionListener()) ;
196     buttonContainer.add(runButton) ;
197
198     //Speed control
199     Container speedContainer = new Container() ;
200     speedContainer.setSize(new Dimension(300,800)) ;
201     speedContainer.setLayout(new GridLayout(2,1)) ;
202
203     Container speedContainerTop = new Container() ;

```

```

204         speedContainerTop.setLayout(f1);
205         speed = 1820;
206         speedLabel = new JLabel(SPEED+1);
207         speedContainerTop.add(speedLabel);
208
209         Container speedContainerBot = new Container();
210         speedContainerBot.setLayout(f1);
211
212         speedScrollBar = new JScrollBar(JScrollBar.HORIZONTAL, 1,
213                                         2, 1, 12);
213         speedScrollBar.addAdjustmentListener(this);
214         speedScrollBar.setSize(new Dimension(200, 20));
215         speedContainerBot.add(speedScrollBar);
216
217         speedContainer.add(speedContainerTop);
218         speedContainer.add(speedContainerBot);
219
220         northContainer.add(buttonContainer);
221         northContainer.add(speedContainer);
222         northContainer.add(createRadioButtons());
223         //Center
224         Container centerContainer = new Container();
225         centerContainer.setLayout(new BoxLayout(centerContainer,
226                                         BoxLayout.Y_AXIS));
226         canvas = new DrawingRegion();
227         canvas.setSize(new Dimension(300, 300));
228         explanation = new ExplainArea("");
229         explanation.setSize(new Dimension(300, 100));
230         centerContainer.add(canvas);
231         centerContainer.add(explanation);
232
233         //West
234         Container westContainer = new Container();
235         westContainer.setLayout(f1);
236         scc = new SourceCodeCanvas();
237         scc.setSize(new Dimension(300, 450));
238         westContainer.add(scc);
239
240         //Add all major components
241         displayPane.add(northContainer, BorderLayout.NORTH);
242         displayPane.add(westContainer, BorderLayout.WEST);
243         displayPane.add(centerContainer, BorderLayout.CENTER);
244
245         visualDisplay.add(displayPane);
246         return visualDisplay;
247     }
248
249     /**
250      * Display for the timed part of the application
251      */
252     private JPanel addTimedDisplay()
253     {
254         JPanel timedDisplay = new JPanel();
255         Container displayPane = new Container();
256         displayPane.setLayout(new BorderLayout());
257
258         //North

```

```

259         Container northContainer = new Container();
260         northContainer.setSize(new Dimension(800,80));
261         GridLayout gl = new GridLayout(1,2);
262         northContainer.setLayout(gl);
263
264         //Run button
265         Container buttonContainer = new Container();
266         buttonContainer.setSize(new Dimension(220,80));
267         FlowLayout fl = new FlowLayout();
268         fl.setAlignment(FlowLayout.CENTER);
269         buttonContainer.setLayout(fl);
270         runButton2 = new JButton(START);
271         runButton2.setPreferredSize(new Dimension(220,30));
272         runButton2.addActionListener(new runActionListener());
273         buttonContainer.add(runButton2);
274
275         northContainer.add(buttonContainer);
276         northContainer.add(createRadioButtons());
277
278         //Center
279         Container centerContainer = new Container();
280         timedExpArea = new TimedExplanation();
281         timedExpArea.setSize(new Dimension(800,400));
282         centerContainer.add(timedExpArea);
283
284         displayPane.add(northContainer, BorderLayout.NORTH);
285         displayPane.add(centerContainer, BorderLayout.CENTER);
286         timedDisplay.add(displayPane);
287         return timedDisplay;
288     }
289
290     private Container createRadioButtons()
291     {
292         //Create the radio buttons.
293         JRadioButton button1 = new JRadioButton(RANDOM);
294         button1.setActionCommand("0");
295         button1.setSelected(true);
296
297         JRadioButton button2 = new JRadioButton(DECREASING);
298         button2.setActionCommand("1");
299
300         JRadioButton button3 = new JRadioButton(INCREASING);
301         button3.setActionCommand("2");
302
303         //Group the radio buttons.
304         ButtonGroup group = new ButtonGroup();
305         group.add(button1);
306         group.add(button2);
307         group.add(button3);
308
309         //Register a listener for the radio buttons.
310         button1.addActionListener(this);
311         button2.addActionListener(this);
312         button3.addActionListener(this);
313
314         Container radioButtonContainer = new Container();
315         radioButtonContainer.setSize(new Dimension(100,80));

```

```

316
317     radioButtonContainer.setLayout(new
318         BoxLayout(radioButtonContainer, BoxLayout.Y_AXIS));
319     radioButtonContainer.add(button1);
320     radioButtonContainer.add(button2);
321     radioButtonContainer.add(button3);
322
323     return radioButtonContainer;
324 }
325
326 public void update()
327 {
328     if(mode == VISUAL) {
329         updateVisualView();
330     } else { //Timed sort
331         updateTimedView();
332     }
333     doLayout();
334     pack();
335 }
336
337 private void initialize()
338 {
339     stopAction();
340     switch(thisAlgType) {
341     case 1:
342         thisAlg = new InsertionSort(arraySize, startConfig,
343             mode);
344         canvas.setIncludeTemp(false);
345         break;
346     case 2:
347         thisAlg = new SelectionSort(arraySize, startConfig,
348             mode);
349         canvas.setIncludeTemp(false);
350         break;
351     case 3:
352         thisAlg = new QuickSort(arraySize, startConfig, mode);
353         canvas.setIncludeTemp(false);
354         break;
355     case 4:
356         thisAlg = new MergeSort(arraySize, startConfig, mode);
357         canvas.setIncludeTemp(true);
358         canvas.setTempArrayList(null);
359         break;
360     case 5:
361         thisAlg = new RandomizedQuickSort(arraySize,
362             startConfig, mode);
363         canvas.setIncludeTemp(false);
364         break;
365     }
366     explanation.setAlgorithm(thisAlg.getName());
367     explanation.setText("");
368     if(mode == VISUAL) {
369         runButton.setText(START);
370     } else {
371         runButton2.setText(START);
372     }

```

```

369         update();
370     }
371
372     private void updateVisualView()
373     {
374         if(canvas.isDisplayable())
375             canvas.setArrayList(thisAlg.getList());
376             canvas.invalidate();
377             canvas.update(canvas.getGraphics());
378     }
379     if(explanation.isDisplayable())
380         explanation.setCopies(thisAlg.getCopies());
381         explanation.setComparisons(thisAlg.getComparisons());
382         explanation.setArraySize(arraySize);
383         explanation.invalidate();
384         explanation.update(explanation.getGraphics());
385     }
386     if(scc.isDisplayable())
387         scc.setSourceCode(thisAlg.getCode());
388         scc.invalidate();
389         scc.update(scc.getGraphics());
390     }
391 }
392
393     private void updateTimedView()
394     {
395         if(timedExpArea.isDisplayable())
396             timedExpArea.setAlgorithm(thisAlg.getName());
397             timedExpArea.setArraySize(arraySize);
398             timedExpArea.setComparisons(thisAlg.getComparisons());
399             timedExpArea.setCopies(thisAlg.getCopies());
400             timedExpArea.setTime(thisAlg.getTime());
401             timedExpArea.setConfiguration(getConfiguration());
402             timedExpArea.update(timedExpArea.getGraphics());
403     }
404 }
405
406 /**
407 * Creates a new thread, so interaction still is possible with
408 * the GUI
409 */
410 public void startAction()
411 {
412     guiThread = new Thread(this);
413     guiThread.start();
414 }
415 /**
416 * Kills the GUI thread, and stops action of the selected
417 * sorting algorithm
418 */
419 public void stopAction()
420 {
421     scc.setHighlightLine(-1);
422     if(thisAlg != null)
423         thisAlg.setStop();
424 }
```

```

424         try { Thread.sleep(200); } catch (InterruptedException e1) {}  

425         guiThread = null;  

426     }  

427  

428     /**  

429      * Starts thread action  

430      */  

431     public void run()  

432     {  

433         thisAlg.setMode(mode);  

434         alg = new Thread(thisAlg);  

435         alg.start();  

436         if(mode == VISUAL)  

437             scc.setSourceCode(thisAlg.getCode());  

438         while(!thisAlg.finishedSorting()) {  

439             explanation.setText(thisAlg.getText());  

440             scc.setHighlightLine(thisAlg.getLine());  

441             explanation.setCopies(thisAlg.getCopies());  

442             explanation.setComparisons(  

443                 thisAlg.getComparisons());  

444             canvas.setArrayList(thisAlg.getList());  

445             if(thisAlg.getTempList() != null) {  

446                 canvas.setTempArrayList(thisAlg.getTempList());  

447             }  

448             canvas.invalidate();  

449             update();  

450             thisAlg.setReadyForNext();  

451             try { Thread.sleep(speed); } catch  

452                 (InterruptedException e1) {}  

453         }  

454         runButton.setText(RESET);  

455         explanation.setText(thisAlg.getText());  

456         explanation.setCopies(thisAlg.getCopies());  

457         explanation.setComparisons(thisAlg.getComparisons());  

458         explanation.setArraySize(arraySize);  

459     }  

460     if(mode == TIMED) {  

461         updateTimedView();  

462         while(!thisAlg.finishedSorting()) {  

463             try { Thread.sleep(100); } catch  

464                 (InterruptedException e1) {}  

465         }  

466         runButton2.setText(RESET);  

467         updateTimedView();  

468     }  

469     /**  

470      * Method used to change between timed and visual display  

471      */  

472     public void itemStateChanged(ItemEvent e) {  

473         stopAction();  

474         CardLayout cl = (CardLayout)(cards.getLayout());  

475         if(VISUAL.equals((String)e.getItem())) {  

476             mode = VISUAL;  

477         } else {  


```

```

478         mode = TIMED;
479     }
480     initialize();
481     cl.show(cards, (String)e.getItem());
482 }
483
484 private String getConfiguration()
485 {
486     switch(startConfig) {
487     case 0:
488         return RANDOM;
489     case 1:
490         return DECREASING;
491     case 2:
492         return INCREASING;
493     }
494     return "";
495 }
496
497 /**
498 * Sets the speed
499 */
500 public void adjustmentValueChanged(AdjustmentEvent e)
501 {
502     speed = 2000 - 180*e.getValue();
503     speedLabel.setText(SPEED +e.getValue());
504 }
505
506 /**
507 * Sets the initial array as random, decreasing or increasing
508 */
509 public void actionPerformed(ActionEvent e)
510 {
511     startConfig = Integer.parseInt(e.getActionCommand());
512     initialize();
513 }
514
515 /*
516 * From this point on, only inner classes handling action
517 * listenings from the GUI
518 */
519 class runActionListener implements ActionListener
520 {
521     public void actionPerformed(ActionEvent e)
522     {
523         if(mode == VISUAL) {
524             if(e.getActionCommand().equals(START)) {
525                 runButton.setText(STOP);
526                 update();
527                 startAction();
528             }
529             if(e.getActionCommand().equals(STOP)) {
530                 runButton.setText(START);
531                 stopAction();
532                 update();
533                 initialize();
534             }
535         }
536     }
537 }

```



```

590             thisAlgType = 4;
591         }
592         if( e.getActionCommand().equals(RANDQUICKSORT) ) {
593             thisAlgType = 5;
594         }
595         initialize();
596     }
597 }
598
599 /**
600 * Inner class handling input of arraysize (including wrong
601 * input)
602 */
603 class sizeChooserListener implements ActionListener
604 {
605     public void actionPerformed(ActionEvent e)
606     {
607         String inputValue = JOptionPane.showInputDialog(INPUT);
608         try {
609             int size = Integer.parseInt(inputValue);
610             if(mode.equals(VISUAL)) {
611                 if( 1 <= size && size <= 25) {
612                     arraySize = size;
613                     initialize();
614                 } else {
615                     JOptionPane.showMessageDialog(new JFrame(),
616                         PLEASE2, ERROR, 1);
617                 }
618             if(mode.equals(TIMED)) {
619                 if( 1 <= size && size <= 500000) {
620                     arraySize = size;
621                     initialize();
622                 } else {
623                     JOptionPane.showMessageDialog(new JFrame(),
624                         PLEASE3, ERROR, 1);
625                 }
626             }
627             catch (Exception ex) {
628                 JOptionPane.showMessageDialog(new JFrame(), PLEASE,
629                     ERROR, 1);
630             }
631     }

```

SourceCodeCanvas.java

```
1 import java.awt.Canvas;
2 import java.awt.Color;
3 import java.awt.Graphics;
4 import java.util.ArrayList;
5
6 public class SourceCodeCanvas extends Canvas
7 {
8     private ArrayList<String> sourceCode;
9     private static final long serialVersionUID = 1L;
10    private int highlightLine;
11
12    public SourceCodeCanvas()
13    {
14        highlightLine = -1;
15        sourceCode = new ArrayList<String>();
16    }
17
18    public void setSourceCode(ArrayList<String> sourceCode)
19    {
20        this.sourceCode = sourceCode;
21    }
22
23    public void setHighlightLine(int line)
24    {
25        highlightLine = line;
26    }
27
28    public void paint(Graphics g)
29    {
30        g.setColor(Color.BLUE);
31        for(int i = 0; i < sourceCode.size(); i++) {
32            if(i == highlightLine) {
33                g.setColor(Color.RED);
34            }
35            g.drawString(sourceCode.get(i), 10, 10+20*i);
36            if(i == highlightLine) {
37                g.setColor(Color.BLUE);
38            }
39        }
40    }
41 }
```

DrawingRegion.java

```

1 import java.awt.Canvas;
2 import java.awt.Color;
3 import java.awt.Graphics;
4 import java.util.ArrayList;
5
6 class DrawingRegion extends Canvas
7 {
8     private static final long serialVersionUID = 1L;
9     private ArrayList<Entry> toSort;
10    private ArrayList<Entry> temp;
11    private boolean includeTemp;
12
13    /*
14     * Note, constructor handled by superclass
15     */
16
17    public void setArrayList (ArrayList<Entry> a)
18    {
19        toSort = a;
20    }
21
22    public void setTempArrayList (ArrayList<Entry> a)
23    {
24        temp = a;
25    }
26
27    public void paint (Graphics g)
28    {
29        //Calculate the width of the bars
30        if (toSort != null && toSort.size() != 0 && toSort.size() <
31            26) {
32            if (includeTemp) { // Mergesort
33                int width = Math.max(1, (this.getWidth() - 20) /
34                    toSort.size());
35                for (int i = 0; i < toSort.size(); i++) {
36                    //Paint the bars of the normal array
37                    g.setColor (toSort.get (i).getColor ());
38                    g.fillRect (10 + width * i, (this.getHeight () - 10) / 2
39                                - toSort.get (i).getValue () / 2, width - 10,
40                                toSort.get (i).getValue () / 2);
41                    // The temp array
42                    if (temp != null) {
43                        g.setColor (temp.get (i).getColor ());
44                        g.fillRect (10 + width * i,
45                                (this.getHeight () - 10) -
46                                temp.get (i).getValue () / 2, width - 10,
47                                temp.get (i).getValue () / 2);
48                    }
49                    g.setColor (Color.BLACK);
50                }
51            } else {
52                int width = Math.max(1, (this.getWidth() - 20) /
53                    toSort.size());
54                //Paint the bars
55                for (int i = 0; i < toSort.size(); i++) {
56

```

```
48                     g . setColor ( toSort . get ( i ) . getColor ( ) );
49                     g . fillRect ( 10 + width * i , ( this . getHeight ( ) - 10 )
50                               - toSort . get ( i ) . getValue ( ) , width - 10 ,
51                               toSort . get ( i ) . getValue ( ) );
52                 }
53             }
54         } else {
55             if ( toSort == null ) {
56                 g . drawString ( "Select an algorithm" , 20 , 50 );
57             } else {
58                 g . drawString ( "Array to large to display" , 20 , 50 );
59                 g . drawString ( "Select array size between 1 and 25" ,
60                               20 , 70 );
61             }
62         }
63     }
64     public void setIncludeTemp ( boolean includeTemp )
65     {
66         this . includeTemp = includeTemp ;
67     }
```

ExplainArea.java

```

1 import java.awt.Canvas;
2 import java.awt.Color;
3 import java.awt.Graphics;
4
5 public class ExplainArea extends Canvas
6 {
7     private static final long serialVersionUID = 1L;
8     private String text = "\u25aa";
9     private String algorithm = "";
10    private double copies;
11    private double comparisons;
12    private int arraySize;
13
14    public void setCopies(double i)
15    {
16        copies = i;
17    }
18
19    public void setComparisons(double i)
20    {
21        comparisons = i;
22    }
23
24    public ExplainArea()
25    {
26        text = "\u25aa";
27        algorithm = "";
28    }
29
30    public ExplainArea(String s)
31    {
32        text = s;
33    }
34
35    public void setText(String s)
36    {
37        text = s;
38    }
39
40    public void setAlgorithm(String s)
41    {
42        algorithm = s;
43    }
44
45    public void paint(Graphics g)
46    {
47        g.setColor(Color.BLUE);
48        g.drawString(text, 10, 10);
49        g.drawString("Using " + algorithm + " algorithm", 10, 30);
50        g.drawString("Number of copies: " + formatDouble(copies) ,
51                    10, 50);
52        g.drawString("Number of comparisons: " +
53                    formatDouble(comparisons) , 10, 70);
54        g.drawString("Arraysize is now: " + arraySize, 10, 100);
55    }

```

```
54
55     public void setArraySize(int arraySize) {
56         this.arraySize = arraySize;
57     }
58
59     private String formatDouble(double d)
60     {
61         if(d < Integer.MAX_VALUE) {
62             int i = (int) d;
63             return ""+i;
64         }
65         return ""+d;
66     }
67 }
```

TimedExplanation.java

```

1 import java.awt.Canvas;
2 import java.awt.Color;
3 import java.awt.Font;
4 import java.awt.Graphics;
5
6 public class TimedExplanation extends Canvas
7 {
8     private static final long serialVersionUID = 1L;
9     private String algorithm;
10    private int arraySize;
11    private double copies;
12    private double comparisons;
13    private String time;
14    private String configuration = "";
15
16    public void paint(Graphics g)
17    {
18        g.setFont(new Font("SansSerif", Font.BOLD, 14));
19        g.setColor(Color.BLUE);
20        g.drawString("Using algorithm: " + algorithm, 10, 30);
21        g.drawString("With array size: " + arraySize, 10, 60);
22        g.setColor(Color.BLACK);
23        g.drawString("Results", 10, 140);
24        g.setColor(Color.BLUE);
25        if(comparisons != 0)
26            g.drawString("Time: " + time + " milliseconds", 10,
27                         180);
28        g.drawString("Number of copies: " +
29                    formatDouble(copies), 10, 210);
30        g.drawString("Number of comparisons: " +
31                    formatDouble(comparisons), 10, 240);
32        g.drawString("Initially: " + configuration, 10, 280);
33    } else {
34        g.drawString("Not ready yet", 10, 180);
35    }
36
37    public void setAlgorithm(String algorithm)
38    {
39        this.algorithm = algorithm;
40    }
41
42    public void setArraySize(int arraySize)
43    {
44        this.arraySize = arraySize;
45    }
46
47    public void setComparisons(double comparisons)
48    {
49        this.comparisons = comparisons;
50    }
51
52    public void setConfiguration(String configuration)
53    {
54        this.configuration = configuration;
55    }

```

```
53      }
54
55  public void setCopies(double copy)
56  {
57      this.copies = copy;
58  }
59
60  public void setTime(String time)
61  {
62      this.time = time;
63  }
64
65  private String formatDouble(double d)
66  {
67      if(d < Integer.MAX_VALUE) {
68          int i = (int) d;
69          return ""+i;
70      }
71      return ""+d;
72  }
73 }
```

Entry.java

```
1 import java.awt.Color;
2
3 public class Entry
4 {
5     private int value;
6     private Color color;
7
8     public Entry(int value, Color color)
9     {
10         this.value = value;
11         this.color = color;
12     }
13
14     public Color getColor()
15     {
16         return color;
17     }
18
19     public void setColor(Color color)
20     {
21         this.color = color;
22     }
23
24     public int getValue()
25     {
26         return value;
27     }
28 }
```

FilledArrays.java

```

1 import java.awt.Color;
2 import java.util.ArrayList;
3 import java.util.Random;
4
5 public class FilledArrays
6 {
7     public static ArrayList<Entry> getArrayList(int entries, int
8         ordered)
9     {
10        ArrayList<Entry> list = new ArrayList<Entry>();
11        int heighth = 300;
12        switch (ordered) {
13            case 0:
14                Random r = new Random();
15                for (int i = 0; i < entries; i++) {
16                    list.add(i, new
17                        Entry(2+r.nextInt(heighth-2), Color.BLACK));
18                }
19                break;
20            case 1:
21                for (int i = 0; i < entries; i++) {
22                    list.add(i, new Entry(heighth - (2 + heighth /
23                        entries)*i, Color.BLACK));
24                }
25                break;
26            case 2:
27                for (int i = 0; i < entries; i++) {
28                    list.add(i, new Entry(2+((heighth-2) /
29                        entries)*i, Color.BLACK));
30                }
31        }
32        return list;
33    }
34    public static int[] getArray(int entries, int ordered)
35    {
36        int[] timedList = new int[entries];
37        switch (ordered) {
38            case 0:
39                Random r = new Random();
40                for (int i = 0; i < entries; i++) {
41                    timedList[i] = r.nextInt(Integer.MAX_VALUE);
42                }
43                break;
44            case 1:
45                for (int i = 0; i < entries; i++) {
46                    timedList[i] = Integer.MAX_VALUE -
47                        (Integer.MAX_VALUE / entries)*i;
48                }
49                break;
50            case 2:
51                for (int i = 0; i < entries; i++) {
52                    timedList[i] = (Integer.MAX_VALUE / entries)*i;
53                }
54        }
55    }
56}
```

```
51         break;
52     }
53     return timedList;
54 }
55 }
```

Algorithm.java

```
1 import java.util.ArrayList ;
2
3 public interface Algorithm extends Runnable
4 {
5     public void setStop();
6
7     public double getCopies();
8
9     public double getComparisons();
10
11    public boolean finishedSorting();
12
13    public void setReadyForNext();
14
15    public String getText();
16
17    public void setMode(String mode);
18
19    public String getName();
20
21    public ArrayList<String> getCode();
22
23    public ArrayList<Entry> getList();
24
25    public ArrayList<Entry> getTempList();
26
27    public int getLine();
28
29    public String getTime();
30 }
```

InsertionSort.java

```

1 import java.awt.Color;
2 import java.util.ArrayList;
3
4 public class InsertionSort implements Algorithm
5 {
6     private ArrayList<Entry> list;
7     private int[] timedList;
8     private ArrayList<String> code;
9     private double copies;
10    private double comparisons;
11    private boolean finishedSorting;
12    private boolean readyForNext;
13    private String status;
14    private int line;
15    private String mode;
16    private String time;
17
18    /**
19     * 0 = random order
20     * 1 = sorted backwards
21     * 2 = sorted normally
22     */
23    public InsertionSort(int entries, int ordered, String mode)
24    {
25        this.mode = mode;
26        if(mode.equals(GUI.VISUAL)) {
27            list = FilledArrays.getArrayList(entries, ordered);
28            code = new ArrayList<String>();
29            addCode();
30            line = -1;
31            readyForNext = true;
32            status = "";
33        }
34        if(mode.equals(GUI.TIMED)) {
35            timedList = FilledArrays.getArray(entries, ordered);
36        }
37        copies = 0;
38        comparisons = 0;
39        finishedSorting = false;
40    }
41
42    private void myWait()
43    {
44        readyForNext = false;
45        while(!readyForNext) {try {Thread.sleep(100);} catch
46            (InterruptedException e) {}};
47    }
48    private void setStatusLineAndWait(int lineno, String
49                                     statusString)
50    {
51        status = statusString;
52        line = lineno;
53        myWait();
54    }

```

```

54
55     public void run()
56     {
57         if( mode.equals(GUI.VISUAL) ) {
58             Entry dummy = new Entry(0, Color.GRAY);
59             int i, j;
60             Entry index;
61             list.get(0).setColor(Color.GREEN);
62             for (i=1; i < list.size(); i++) {
63                 if(finishedSorting) { break; }
64                 setStatusLineAndWait(2, "i is now " + i);
65
66                 list.get(i).setColor(Color.RED);
67                 index = list.get(i);
68                 copies++;
69                 setStatusLineAndWait(3, "index is set to the next "
70                                     + "unsorted element" );
71
72                 list.get(i).setColor(Color.BLACK);
73                 list.set(i, dummy);
74
75                 j = i;
76                 setStatusLineAndWait(4, "Sets j = i");
77
78                 setStatusLineAndWait(5, "Comparing elements (j-1) "
79                                     + "and index");
80                 while ((j > 0) && (list.get(j-1).getValue() >
81                                     index.getValue())) {
82                     if(finishedSorting) { break; }
83                     comparisons++;
84                     copies++;
85                     list.set(j, list.get(j-1));
86                     list.set(j-1, dummy);
87                     setStatusLineAndWait(6, "Now moving item " +
88                                     (j-1) + " to " + j);
89
90                     j = j - 1;
91                     setStatusLineAndWait(7, "Updating j");
92                 }
93                 if(j > 0) { // The comparison that failed the
94                             // while loop;
95                     comparisons++;
96                 }
97                 index.setColor(Color.GREEN);
98                 list.set(j, index);
99                 copies++;
100                list.get(j).setColor(Color.GREEN);
101                setStatusLineAndWait(8, "Inserting " + i + " at "
102                                     + position + j);
103            }
104            setStatusLineAndWait(-1, "Finished sorting");
105            setStop();
106        }
107        else { //Timed sort
108            long starttime = System.currentTimeMillis();
109        }
110    }
111}

```

```

105         int i, j, index;
106         for (i=1; i < timedList.length ; i++){
107             if(finishedSorting) {
108                 break;
109             }
110             index = timedList [ i ];
111             copies++;
112             j = i ;
113
114             while ((j > 0) && (timedList [ j-1 ] > index)) {
115                 assert (comparisons >= 0) :comparisons;
116                 assert (copies >= 0) :copies;
117                 comparisons++;
118                 copies++;
119                 timedList [ j ] = timedList [ j-1 ];
120                 j = j - 1;
121             }
122             if(j>0) {
123                 comparisons++; //The comparison that failed
124                     the while loop
125             }
126             timedList [ j ] = index ;
127             copies++;
128         }
129     }
130
131     long endtime = System.currentTimeMillis();
132     time = "" +(endtime - starttime);
133     setStop();
134     myWait();
135 }
136
137     public void setReadyForNext ()
138     {
139         readyForNext = true;
140     }
141
142     private void addCode()
143     {
144         code.add("InsertionSort (array A, size )"); // 0
145         code.add(""); // 1
146         code.add("for (i=1; i < size ; i++)"); // 2
147         code.add("index=A[ i ]"); // 3
148         code.add("j=i"); // 4
149         code.add("while ((j>0) AND A[ j-1 ]> index )"); // 5
150         code.add("A[ j ]=A[ j-1 ]"); // 6
151         code.add("j=j-1"); // 7
152         code.add("A[ j ]=index"); // 8
153     }
154
155     public boolean finishedSorting()
156     {
157         return finishedSorting;
158     }
159
160     public ArrayList<Entry> getList ()
161     {
162         return list;

```

```
161     }
162
163     public ArrayList<String> getCode()
164     {
165         return code;
166     }
167
168     public String getText()
169     {
170         return status;
171     }
172
173     public double getComparisons()
174     {
175         return comparisons;
176     }
177
178     public double getCopies()
179     {
180         return copies;
181     }
182
183     public void setStop()
184     {
185         finishedSorting = true;
186     }
187
188     public int getLine()
189     {
190         return line;
191     }
192     public String getName() {
193         return GUI.INSERTIONSORT;
194     }
195     public void setMode( String mode)
196     {
197         this.mode = mode;
198     }
199
200     public String getTime() {
201         return time;
202     }
203
204     public ArrayList<Entry> getTempList()
205     {
206         return null;
207     }
208 }
```

SelectionSort.java

```

1 import java.awt.Color;
2 import java.util.ArrayList;
3
4 public class SelectionSort implements Algorithm
5 {
6     private ArrayList<Entry> list;
7     private int[] timedList;
8     private ArrayList<String> code;
9     private double copies;
10    private double comparisons;
11    private boolean finishedSorting;
12    private boolean readyForNext;
13    private String status;
14    private int line;
15    private String mode;
16    private String time;
17
18    public SelectionSort(int entries, int ordered, String mode)
19    {
20        if(mode.equals(GUI.VISUAL)) {
21            list = FilledArrays.getArrayList(entries, ordered);
22            code = new ArrayList<String>();
23            addCode();
24            line = -1;
25            readyForNext = true;
26            status = "";
27        }
28        if(mode.equals(GUI.TIMED)) {
29            timedList = FilledArrays.getArray(entries, ordered);
30        }
31        copies = 0;
32        comparisons = 0;
33        finishedSorting = false;
34    }
35
36    private void myWait()
37    {
38        readyForNext = false;
39        while(!readyForNext) {try {Thread.sleep(100);} catch
40            (InterruptedException e) {}};
41    }
42
43    private void setStatusLineAndWait(int lineno, String
44                                     statusString)
45    {
46        status = statusString;
47        line = lineno;
48        myWait();
49    }
50
51    public void run()
52    {
53        if(mode.equals(GUI.VISUAL)) {
54            int i, j, min;
55            Entry temp;

```

```

54         setStatusLineAndWait (0, "Starting algorithm");
55         for (i = 0; i < list.size()-1; i++) {
56             if(finishedSorting) { break; }
57             list.get(i).setColor(Color.RED);
58             setStatusLineAndWait (2, "i is now: "+i);
59
60             min = i;
61             setStatusLineAndWait (3, "min=i");
62
63             for (j = i+1; j < list.size(); j++) {
64                 setStatusLineAndWait (4, "j is now: "+j);
65
66                 comparisons++;
67                 setStatusLineAndWait (5, "Comparing element j and min");
68                 if (list.get(j).getValue() < list.get(min).getValue()) {
69                     if(min != i) {
70                         list.get(min).setColor(Color.BLACK);
71                     }
72                     list.get(j).setColor(Color.BLUE);
73                     min = j;
74                     setStatusLineAndWait (6, "min is set to j");
75                 }
76
77                 temp = list.get(i);
78                 list.set(i, list.get(min));
79                 list.set(min, temp);
80                 copies+=3;
81                 list.get(min).setColor(Color.BLACK);
82                 list.get(i).setColor(Color.GREEN);
83                 setStatusLineAndWait (7, "Exchanging items min and i");
84             }
85             list.get(list.size()-1).setColor(Color.GREEN);
86             setStatusLineAndWait (-1, "Sorting has finished");
87             finishedSorting = true;
88         } else { //Timed sort
89             long starttime = System.currentTimeMillis();
90             int i,j,min,temp;
91             for(i = 0; i < timedList.length-1; i++) {
92                 min = i;
93                 for(j = i+1; j < timedList.length; j++) {
94                     comparisons++;
95                     if(timedList[j] < timedList[min]) {
96                         min = j;
97                     }
98                 }
99                 copies+=3;
100                temp = timedList[i];
101                timedList[i] = timedList[min];
102                timedList[min] = temp;
103            }
104            long endtime = System.currentTimeMillis();
105            time = "" +(endtime - starttime);
106            setStop();
107            myWait();

```

```
108         }
109     }
110
111     private void addCode()
112     {
113         code.add("SelectionSort (array A, size )"); // 0
114         code.add(""); // 1
115         code.add("for ( i=0; i< size -1; i++)"); // 2
116         code.add("min=i"); // 3
117         code.add("for ( j=i+1; j< size ; j++)"); // 4
118         code.add("if (A[ j]<A[ min])"); // 5
119         code.add("min=j"); // 6
120         code.add("swap(A[ i], A[ min])"); // 7
121     }
122
123     public boolean finishedSorting()
124     {
125         return finishedSorting;
126     }
127
128     public ArrayList<Entry> getList ()
129     {
130         return list ;
131     }
132
133     public ArrayList<String> getCode()
134     {
135         return code;
136     }
137
138     public String getText()
139     {
140         return status;
141     }
142
143     public double getComparisons()
144     {
145         return comparisons;
146     }
147
148     public double getCopies()
149     {
150         return copies;
151     }
152
153     public void setStop ()
154     {
155         finishedSorting = true;
156     }
157
158     public int getLine ()
159     {
160         return line;
161     }
162
163     public void setReadyForNext ()
164     {
```

```
165         readyForNext = true;
166     }
167
168     public String getName()
169     {
170         return GUI.SELECTIONSORT;
171     }
172
173     public void setMode( String mode)
174     {
175         this.mode = mode;
176     }
177
178     public String getTime()
179     {
180         return time;
181     }
182
183     public ArrayList<Entry> getTempList()
184     {
185         return null;
186     }
187 }
```

QuickSort.java

```

1 import java.awt.Color;
2 import java.util.ArrayList;
3
4 public class QuickSort implements Algorithm
5 {
6     private ArrayList<Entry> list;
7     private int[] timedList;
8     private ArrayList<String> code;
9     private double copies;
10    private double comparisons;
11    private boolean finishedSorting;
12    private boolean readyForNext;
13    private String status;
14    private int line;
15    private String mode;
16    private String time;
17    private int ordered;
18
19    public QuickSort(int entries, int ordered, String mode)
20    {
21        this.ordered = ordered;
22        if(mode.equals(GUI.VISUAL)) {
23            list = FilledArrays.getArrayList(entries, ordered);
24            code = new ArrayList<String>();
25            addCode();
26            line = -1;
27            readyForNext = true;
28            status = "";
29
30        }
31        if(mode.equals(GUI.TIMED)) {
32            timedList = FilledArrays.getArray(entries, ordered);
33        }
34        copies = 0;
35        comparisons = 0;
36        finishedSorting = false;
37    }
38
39    private void myWait()
40    {
41        readyForNext = false;
42        while(!readyForNext) {try {Thread.sleep(100);} catch
43            (InterruptedException e) {}};
44    }
45
46    private void setStatusLineAndWait(int lineno, String
47        statusString)
48    {
49        status = statusString;
50        line = lineno;
51        myWait();
52    }
53    public void run()
54    {

```

```

54         if (mode. equals(GUI.VISUAL)) {
55             setStatusLineAndWait(0, "Starts_the_algorithm");
56             visualquicksort(0, list.size() -1);
57             setStatusLineAndWait(-1, "Finished_sorting");
58         }
59         if(mode. equals(GUI.TIMED)) {
60             if(ordered != 0 && timedList.length > 7500 ) {
61                 // Creates stack overflow!
62                 time = "STACK_OVERFLOW";
63                 copies = 1;
64                 comparisons = 1;
65             } else {
66                 long starttime = System.currentTimeMillis();
67                 timedQuicksort(0, timedList.length -1);
68                 long endtime = System.currentTimeMillis();
69                 time = "" +(endtime - starttime);
70             }
71         }
72         setStop();
73         myWait();
74     }
75
76     private void visualquicksort(int p , int r)
77     {
78         setStatusLineAndWait(1, "_evaluates_(p<_r)");
79         if(p < r) {
80             setStatusLineAndWait(2, "Calls_partition_method");
81             int q = visualPartition(p, r);
82             setStatusLineAndWait(3, "Makes_recursive_call");
83             visualquicksort(p, q-1);
84             setStatusLineAndWait(4, "Makes_recursive_call");
85             visualquicksort(q+1, r);
86         }
87     }
88
89     private int visualPartition(int p , int r)
90     {
91         for(int m = 0; m < list.size(); m++) {
92             if(m >= p && m <= r) {
93                 list.get(m).setColor(Color.BLUE);
94             }
95         }
96         setStatusLineAndWait(6, "Partition_method");
97
98         list.get(r).setColor(Color.RED);
99         copies++;
100        setStatusLineAndWait(7, "x=A[r]");
101        Entry x = list.get(r);
102        setStatusLineAndWait(8, "i=p-1");
103        int i = p-1;
104
105        setStatusLineAndWait(9, "for_loop");
106        for(int j = p; j < r; j++) {
107            comparisons++;
108            setStatusLineAndWait(10, "Comparing A[j] and x");
109            if(list.get(j).getValue() <= x.getValue()) {
110                setStatusLineAndWait(11, "Increment_i");

```

```

111             i = i+1;
112             copies+=3;
113             list .get( i ) .setColor ( Color .LIGHT_GRAY );
114             list .get( j ) .setColor ( Color .LIGHT_GRAY );
115             setStatusLineAndWait ( 12 , "Exchanging " + i + " and "
116                                     + j );
116             Entry temp = list .get( i );
117             list .set( i , list .get( j ) );
118             list .set( j , temp );
119             list .get( i ) .setColor ( Color .BLUE );
120             list .get( j ) .setColor ( Color .BLUE );
121
122         }
123     }
124     copies+=3;
125     list .get( i+1 ) .setColor ( Color .LIGHT_GRAY );
126     list .get( r ) .setColor ( Color .LIGHT_GRAY );
127     setStatusLineAndWait ( 13 , "Exchanging " + i+1 + " + ( i+1 ) + " and "
128                           + r );
128     Entry temp = list .get( i+1 );
129     list .set( i+1 , list .get( r ) );
130     list .set( r , temp );
131     list .get( i+1 ) .setColor ( Color .BLUE );
132     list .get( r ) .setColor ( Color .BLUE );
133
134
135     setStatusLineAndWait ( 14 , "Returns " + i+1 );
136     for ( Entry s : list ) {
137         s .setColor ( Color .BLACK );
138     }
139     return i+1;
140 }
141
142 private void timedQuicksort ( int p , int r )
143 {
144     if ( p < r ) {
145         int q = timedPartition ( p , r );
146         timedQuicksort ( p , q-1 );
147         timedQuicksort ( q+1 , r );
148     }
149 }
150
151 private int timedPartition ( int p , int r )
152 {
153     int x = timedList [ r ];
154     copies++;
155     int i = p-1;
156     for ( int j = p ; j < r ; j++ ) {
157         comparisons++;
158         if ( timedList [ j ] <= x ) {
159             i = i + 1;
160             copies+=3;
161             int temp = timedList [ i ];
162             timedList [ i ] = timedList [ j ];
163             timedList [ j ] = temp;
164         }
165     }

```

```

166         copies+=3;
167         int temp = timedList [ i+1];
168         timedList [ i+1] = timedList [ r ];
169         timedList [ r ] = temp;
170         return i+1;
171     }
172
173     private void addCode()
174     {
175         code.add("QuickSort( array A, p, r )"); // 0
176         code.add(" if (p < r)"); // 1
177         code.add("     q = Partition( A, p, r )"); // 2
178         code.add("     QuickSort( A, p, q - 1 )"); // 3
179         code.add("     QuickSort( A, q + 1, r )"); // 4
180         code.add(" "); // 5
181         code.add("     Partition( array A, p, r )"); // 6
182         code.add("     x = A[ r ]"); // 7
183         code.add("     i = p - 1"); // 8
184         code.add("     for ( j = p; j < r; j++ )"); // 9
185         code.add("         if (A[ j ] <= x )"); // 10
186         code.add("             i = i + 1"); // 11
187         code.add("         Exchange( A[ i ], A[ j ] )"); // 12
188         code.add("     Exchange( A[ i + 1 ], A[ r ] )"); // 13
189         code.add("     return i + 1"); // 14
190     }
191
192     public boolean finishedSorting()
193     {
194         return finishedSorting;
195     }
196
197     public ArrayList<Entry> getList()
198     {
199         return list;
200     }
201
202     public ArrayList<String> getCode()
203     {
204         return code;
205     }
206
207     public String getText()
208     {
209         return status;
210     }
211
212     public double getComparisons()
213     {
214         return comparisons;
215     }
216
217     public double getCopies()
218     {
219         return copies;
220     }
221
222     public void setStop()

```

```
223     {
224         finishedSorting = true;
225     }
226
227     public int getLine()
228     {
229         return line;
230     }
231
232     public void setReadyForNext()
233     {
234         readyForNext = true;
235     }
236
237     public String getName()
238     {
239         return GUI.QUICKSORT;
240     }
241
242     public void setMode( String mode)
243     {
244         this.mode = mode;
245     }
246
247     public String getTime()
248     {
249         return time;
250     }
251
252     public ArrayList<Entry> getTempList()
253     {
254         return null;
255     }
256 }
```

MergeSort.java

```

1 import java.awt.Color;
2 import java.util.ArrayList;
3
4 public class MergeSort implements Algorithm
5 {
6     private ArrayList<Entry> list;
7     private ArrayList<Entry> temp;
8     private int[] timedList;
9     private int[] timedTemp;
10    private ArrayList<String> code;
11    private double copies;
12    private double comparisons;
13    private boolean finishedSorting;
14    private boolean readyForNext;
15    private String status;
16    private int line;
17    private String mode;
18    private String time;
19    private Entry empty;
20
21    public MergeSort(int entries, int ordered, String mode)
22    {
23        if(mode.equals(GUI.VISUAL)) {
24            list = FilledArrays.getArrayList(entries, ordered);
25            temp = new ArrayList<Entry>();
26            code = new ArrayList<String>();
27            addCode();
28            line = -1;
29            readyForNext = true;
30            status = "";
31
32            empty = new Entry(0, Color.BLACK);
33            for(int i = 0; i < entries; i++) {
34                temp.add(empty);
35            }
36        }
37        if(mode.equals(GUI.TIMED)) {
38            timedList = FilledArrays.getArray(entries, ordered);
39            timedTemp = new int[entries];
40
41        }
42        copies = 0;
43        comparisons = 0;
44        finishedSorting = false;
45    }
46
47    private void myWait()
48    {
49        readyForNext = false;
50        while(!readyForNext) {try {Thread.sleep(100);} catch
51            (InterruptedException e) {}};
52    }
53    private void setStatusLineAndWait(int lineno, String
54        statusString)

```

```

54      {
55          status = statusString ;
56          line = lineno ;
57          myWait() ;
58      }
59
60      public void run()
61      {
62          if(mode.equals(GUI.VISUAL)) {
63              visualMergeSort(0, list.size()-1);
64              setStatusLineAndWait(-1, "Finished sorting");
65          } else { //Timed sort
66              long starttime = System.currentTimeMillis();
67              timedMergeSort(0, timedList.length-1);
68              long endtime = System.currentTimeMillis();
69              time = "" +(endtime - starttime);
70          }
71          setStop();
72          myWait();
73      }
74
75      private void visualMergeSort(int lo , int hi)
76      {
77          setStatusLineAndWait(0, "Mergesort called with lo : " +lo +
78                                " .. hi : " + hi);
79          setStatusLineAndWait(1, "Evaluating if statement");
80          if(lo<hi) {
81              setStatusLineAndWait(2, "Calculating middle of array");
82              int m = (lo+hi)/2;
83              setStatusLineAndWait(3, "Recursive call with first half
84                                of array");
85              visualMergeSort(lo , m);
86              setStatusLineAndWait(4, "Recursive call with second
87                                half of array");
88              visualMergeSort(m+1 , hi);
89              setStatusLineAndWait(5, "Merging the two parts
90                                together , using Merge");
91              visualMerge(lo , m , hi);
92          }
93      }
94
95      void visualMerge(int lo , int m, int hi)
96      {
97          setStatusLineAndWait(0, "Merge called with lo : " +lo + " .. m : "
98                                " + m + " .. hi : " + hi);
99          int i,j,k;
100
101         //Copy to temporary array
102         for(i = lo ; i <=hi; i++) {
103             setStatusLineAndWait(8, "for loop to move elements");
104             copies++;
105             temp.set(i, new Entry(list.get(i).getValue(),
106                                   Color.LIGHT_GRAY));
107             setStatusLineAndWait(9, "Moving element to temporary
108                                array");
109         }
110         i= lo ;

```

```

104         j = m+1;
105         k= lo ;
106         setStatusLineAndWait (10 , "Initializing _variables");
107         while(i<=m && j<=hi) {
108             comparisons++;
109             setStatusLineAndWait (12 , "if _statement _evaluated");
110             if(temp .get (i) .getValue() <= temp .get (j) .getValue()) {
111                 temp .get (i) .setColor (Color .RED);
112                 list .set (k, new Entry (temp .get (i) .getValue(),
113                                         Color .GREEN));
114                 copies++;
115                 setStatusLineAndWait (13 , "Copying _temp[i] _to _A[k]");
116                 list .get (k) .setColor (Color .BLACK);
117                 temp .get (i) .setColor (Color .LIGHT_GRAY);
118                 setStatusLineAndWait (14 , "Incrementing _i");
119                 i=i+1;
120             } else {
121                 setStatusLineAndWait (15 , "evaluating _else _part");
122                 temp .get (j) .setColor (Color .RED);
123                 list .set (k, new Entry (temp .get (j) .getValue(),
124                                         Color .GREEN));
125                 copies++;
126                 setStatusLineAndWait (16 , "Copying _temp[j] _to _A[k]");
127                 list .get (k) .setColor (Color .BLACK);
128                 temp .get (j) .setColor (Color .LIGHT_GRAY);
129                 setStatusLineAndWait (17 , "Incrementing _j");
130                 j=j+1;
131             }
132             k=k+1;
133         }
134         setStatusLineAndWait (19 , "While _loop , i=" + i + " , m=" + m);
135         while(i<=m) {
136             temp .get (i) .setColor (Color .RED);
137             list .set (k, new Entry (temp .get (i) .getValue(),
138                                         Color .GREEN));
139             copies++;
140             setStatusLineAndWait (20 , "Copying _temp[i] _to _A[k]");
141             list .get (k) .setColor (Color .BLACK);
142             temp .get (i) .setColor (Color .LIGHT_GRAY);
143             setStatusLineAndWait (21 , "Incrementing _i _and _k");
144             k=k+1;
145             i=i+1;
146         }
147         emptyTempArray();
148     }
149     private void emptyTempArray()
150     {
151         for( int i = 0; i < temp .size () ; i++) {
152             temp .set (i , empty );
153         }
154     }
155     private void timedMergeSort( int lo , int hi)

```

```

156      {
157          if(lo<hi) {
158              int m = (lo+hi)/2;
159              timedMergeSort( lo , m );
160              timedMergeSort( m+1 , hi );
161              timedMerge( lo , m , hi );
162          }
163      }
164
165      void timedMerge( int lo , int m, int hi)
166      {
167          int i,j,k;
168          //Copy to temporary array
169          for(i = lo ; i <=hi; i++) {
170              copies++;
171              timedTemp[ i ] = timedList [ i ];
172          }
173          i= lo ;
174          j = m+1;
175          k= lo ;
176          while(i<=m && j<=hi) {
177              comparisons++;
178              if(timedTemp[ i ] <= timedTemp[ j ]) {
179                  copies++;
180                  timedList [ k ] = timedTemp[ i ];
181                  i=i+1;
182              } else {
183                  copies++;
184                  timedList [ k ] = timedTemp[ j ];
185                  j=j+1;
186              }
187              k=k+1;
188          }
189          while(i<=m) {
190              copies++;
191              timedList [ k ] = timedTemp[ i ];
192              k=k+1;
193              i=i+1;
194          }
195      }
196
197      private void addCode()
198      {
199          code.add("MergeSort( A , int lo , int hi )"); // 0
200          code.add(" if ( lo<hi )"); // 1
201          code.add("     int m=(lo+hi)/2 ;"); // 2
202          code.add("     MergeSort( A , lo , m );"); // 3
203          code.add("     MergeSort( A , m+1 , hi );"); // 4
204          code.add("     Merge( A , lo , m , hi );"); // 5
205          code.add(" "); // 6

```

```

206         code.add("Merge( array A, int lo, int m, int hi)");      // 7
207         code.add("    for (int i=lo; i<=hi; i++)");           // 8
208         code.add("        temp[ i ] =A[ i ];");                  // 9
209         code.add("        int i=lo; int j=m+1; int k=lo;");       // 10
210         code.add("        while (i<=m&&j<=hi)");            // 11
211         code.add("            if (temp[ i ] <= temp[ j ])"); // 12
212         code.add("                A[ k ] = temp[ i ];");      // 13
213         code.add("                i=i+1;");                   // 14
214         code.add("            else");                      // 15
215         code.add("                A[ k ] = temp[ j ];");      // 16
216         code.add("                j=j+1;");                   // 17
217         code.add("                k=k+1;");                   // 18
218         code.add("            while ( i<=m )");            // 19
219         code.add("                A[ k ] = temp[ i ];");      // 20
220         code.add("                k=k+1; i=i+1;");          // 21
221     }
222
223     public boolean finishedSorting()
224     {
225         return finishedSorting;
226     }
227
228     public ArrayList<Entry> getList()
229     {
230         return list;
231     }
232
233     public ArrayList<Entry> getTempList()
234     {
235         return temp;
236     }
237
238     public ArrayList<String> getCode()
239     {
240         return code;
241     }
242
243     public String getText()
244     {
245         return status;
246     }
247

```

```
248     public double getComparisons()
249     {
250         return comparisons;
251     }
252
253     public double getCopies()
254     {
255         return copies;
256     }
257
258     public void setStop()
259     {
260         finishedSorting = true;
261     }
262
263     public int getLine()
264     {
265         return line;
266     }
267
268     public void setReadyForNext()
269     {
270         readyForNext = true;
271     }
272
273     public String getName()
274     {
275         return GUI.MERGESORT;
276     }
277
278     public void setMode( String mode)
279     {
280         this.mode = mode;
281     }
282
283     public String getTime()
284     {
285         return time;
286     }
287 }
```

RandomizedQuickSort.java

```

1 import java.awt.Color;
2 import java.util.ArrayList;
3 import java.util.Random;
4
5 public class RandomizedQuickSort implements Algorithm
6 {
7     private ArrayList<Entry> list;
8     private int[] timedList;
9     private ArrayList<String> code;
10    private double copies;
11    private double comparisons;
12    private boolean finishedSorting;
13    private boolean readyForNext;
14    private String status;
15    private int line;
16    private String mode;
17    private String time;
18    private Random rand;
19
20    public RandomizedQuickSort(int entries, int ordered, String
21                               mode)
22    {
23        rand = new Random();
24        if(mode.equals(GUI.VISUAL)) {
25            list = FilledArrays.getArrayList(entries, ordered);
26            code = new ArrayList<String>();
27            addCode();
28            line = -1;
29            readyForNext = true;
30            status = "";
31        }
32        if(mode.equals(GUI.TIMED)) {
33            timedList = FilledArrays.getArray(entries, ordered);
34        }
35        copies = 0;
36        comparisons = 0;
37        finishedSorting = false;
38    }
39
40    private void myWait()
41    {
42        readyForNext = false;
43        while(!readyForNext) {try {Thread.sleep(100);} catch
44            (InterruptedException e) {}};
45    }
46
47    private void setStatusLineAndWait(int lineno, String
48                                     statusString)
49    {
50        status = statusString;
51        line = lineno;
52        myWait();
53    }

```

```

53  public void run()
54  {
55      if(mode.equals(GUI.VISUAL)) {
56          setStatusLineAndWait(0, "Starts the algorithm");
57          visualquicksort(0, list.size()-1);
58          setStatusLineAndWait(-1, "Finished sorting");
59      }
60      if(mode.equals(GUI.TIMED)) {
61          long starttime = System.currentTimeMillis();
62          timedQuicksort(0, timedList.length-1);
63          long endtime = System.currentTimeMillis();
64          time = "" +(endtime - starttime);
65      }
66      setStop();
67      myWait();
68  }
69
70  private void visualquicksort(int p, int r)
71  {
72      setStatusLineAndWait(1, "evaluates (p < r)");
73      if(p < r) {
74          setStatusLineAndWait(2, "Calls partition method");
75          int q = visualPartition(p, r);
76          setStatusLineAndWait(3, "Makes recursive call");
77          visualquicksort(p, q-1);
78          setStatusLineAndWait(4, "Makes recursive call");
79          visualquicksort(q+1, r);
80      }
81  }
82
83  private int visualPartition(int p, int r)
84  {
85      for(int m = 0; m < list.size(); m++) {
86          if(m >= p && m <= r) {
87              list.get(m).setColor(Color.BLUE);
88          }
89      }
90      setStatusLineAndWait(5, "Partition method");
91
92      int randomElementNumber = rand.nextInt(r-p)+p;
93      Entry t = list.get(randomElementNumber);
94      t.setColor(Color.CYAN);
95      setStatusLineAndWait(6, "Selecting element at random");
96
97      list.get(r).setColor(Color.CYAN);
98      setStatusLineAndWait(6, "Exchanging this with last element");
99      copies+=3;
100     list.set(randomElementNumber, list.get(r));
101     list.set(r, t);
102
103     list.get(r).setColor(Color.BLUE);
104     list.get(randomElementNumber).setColor(Color.BLUE);
105     setStatusLineAndWait(6, "Now last element is selected at random");
106
107

```

```

108         list . get ( r ) . setColor ( Color .RED) ;
109         copies++ ;
110         setStatusLineAndWait ( 7 , "x = A[ r ]" ) ;
111         Entry x = list . get ( r ) ;
112         setStatusLineAndWait ( 8 , "i = p-1" ) ;
113         int i = p-1 ;
114
115         setStatusLineAndWait ( 9 , "for loop" ) ;
116         for ( int j = p ; j < r ; j ++ ) {
117             comparisons++ ;
118             setStatusLineAndWait ( 10 , "Comparing A[ j ] and x" ) ;
119             if ( list . get ( j ) . getValue ( ) <= x . getValue ( ) ) {
120                 setStatusLineAndWait ( 11 , "Increment i" ) ;
121                 i = i +1 ;
122                 copies+=3 ;
123                 list . get ( i ) . setColor ( Color .LIGHT_GRAY) ;
124                 list . get ( j ) . setColor ( Color .LIGHT_GRAY) ;
125                 setStatusLineAndWait ( 12 , "Exchanging i : "+i+" and "
126                                         j : "+j ) ;
127                 Entry temp = list . get ( i ) ;
128                 list . set ( i , list . get ( j ) ) ;
129                 list . set ( j , temp ) ;
130                 list . get ( i ) . setColor ( Color .BLUE) ;
131                 list . get ( j ) . setColor ( Color .BLUE) ;
132             }
133             copies+=3 ;
134             list . get ( i +1 ) . setColor ( Color .LIGHT_GRAY) ;
135             list . get ( r ) . setColor ( Color .LIGHT_GRAY) ;
136             setStatusLineAndWait ( 13 , "Exchanging i+1 : "+( i +1)+" and "
137                                         r : "+r ) ;
138             Entry temp = list . get ( i +1 ) ;
139             list . set ( i +1 , list . get ( r ) ) ;
140             list . set ( r , temp ) ;
141             list . get ( i +1 ) . setColor ( Color .BLUE) ;
142             list . get ( r ) . setColor ( Color .BLUE) ;
143
144
145             setStatusLineAndWait ( 14 , "Returns i+1" ) ;
146             for ( Entry s : list ) {
147                 s . setColor ( Color .BLACK) ;
148             }
149             return i +1 ;
150         }
151
152         private void timedQuicksort ( int p , int r )
153     {
154             if ( p < r ) {
155                 int q = timedPartition ( p , r ) ;
156                 timedQuicksort ( p , q -1 ) ;
157                 timedQuicksort ( q +1 , r ) ;
158             }
159         }
160
161         private int timedPartition ( int p , int r )
162     {

```

```

163     int randomElementNumber = rand.nextInt(r-p)+p;
164     int t = timedList[randomElementNumber];
165     timedList[randomElementNumber] = timedList[r];
166     timedList[r] = t;
167     copies+=3;
168     int x = timedList[r];
169     copies++;
170     int i = p-1;
171
172     for(int j = p; j < r ; j++) {
173         comparisons++;
174         if(timedList[j] <= x) {
175             i = i + 1;
176             copies+=3;
177             int temp = timedList[i];
178             timedList[i] = timedList[j];
179             timedList[j] = temp;
180         }
181     }
182     copies+=3;
183     int temp = timedList[i+1];
184     timedList[i+1] = timedList[r];
185     timedList[r] = temp;
186     return i+1;
187 }
188
189 private void addCode()
190 {
191     code.add("QuickSort(_array_A_,_p_,_r_)" ); // 0
192     code.add(" _ _ _ if_(p_<_r_)" ); // 1
193     code.add(" _ _ _ _ q_=Partition(A_,_p_,_r_)" ); // 2
194     code.add(" _ _ _ _ QuickSort(A_,_p_,_q_-1_)" ); // 3
195     code.add(" _ _ _ _ QuickSort(A_,_q_+_1_,_r_)" ); // 4
196     code.add(" Partition(_array_A_,_p_,_r_)" ); // 5
197     code.add(" _ _ _ Exchange(A[_r_],_random_element)" ); // 6
198     code.add(" _ _ _ x=_A[_r_]" ); // 7
199     code.add(" _ _ _ i=_p_-1" ); // 8
200     code.add(" _ _ _ for(j=_p_+_1_+_j<_r_+_j++)" ); // 9
201     code.add(" _ _ _ _ if(A[_j_]<=_x)" ); // 10
202     code.add(" _ _ _ _ _ i=_i+_1" ); // 11
203     code.add(" _ _ _ _ _ Exchange(A[_i_],_A[_j_])" ); // 12
204     code.add(" Exchange(A[_i_+_1_],_A[_r_])" ); // 13
205     code.add(" return_i+1" ); // 14
206 }
207
208 public boolean finishedSorting()
209 {
210     return finishedSorting;
211 }
212
213 public ArrayList<Entry> getList()
214 {
215     return list;
216 }
217
218 public ArrayList<String> getCode()
219 {

```

```
220         return code;
221     }
222
223     public String getText()
224     {
225         return status;
226     }
227
228     public double getComparisons()
229     {
230         return comparisons;
231     }
232
233     public double getCopies()
234     {
235         return copies;
236     }
237
238     public void setStop()
239     {
240         finishedSorting = true;
241     }
242
243     public int getLine()
244     {
245         return line;
246     }
247
248     public void setReadyForNext()
249     {
250         readyForNext = true;
251     }
252
253     public String getName()
254     {
255         return GUI.RANDQUICKSORT;
256     }
257
258     public void setMode(String mode)
259     {
260         this.mode = mode;
261     }
262
263     public String getTime()
264     {
265         return time;
266     }
267
268     public ArrayList<Entry> getTempList()
269     {
270         return null;
271     }
272 }
```