# A Java Based Tool to Monitor Execution Time of Different Sorting Algorithms


Submitted By

**Nusrat Chowdhury**

**Id. 2009-3-60-002**

Supervised By

**Dr. Shamim Akhter**

Assistant Professor,

Dept. of Computer Science and Engineering


**A Project Submitted in Partial Fulfillment of the Requirements for the Degree of Bachelors of Science in Computer Science and Engineering**



**Department of Computer Science and Engineering**
**East West University, Dhaka**


**Jan, 2016**

# Abstract

In computer science, each sorting algorithm is better in some situation and has its own advantages. For example, the insertion sort is preferable to the quick sort for small files and for almost-sorted files. To measure the performance of each sorting algorithm, the most important factor is runtime that a specific sort uses to execute a data. Because the fastest algorithm is the best algorithm, it pays to know which is the sorting algorithm fastest.

In this study, we will determine the efficiency of the various sorting algorithms according to the time and number of swaps by using randomized trials. The build environment will be built using the Java language. The research will discuss and implement several sorting algorithms such as bubble sort, selection sort, insertion sort and will also include complexity sort such as quick sort, cocktail sort, and merge sort. I will represent these algorithms as a way to sort an array or integers and run random trails of length. The research will provide the number of swaps and the runtime of each sorting algorithm. To investigate, I create a package called “sorting” which contains two classes. First is called “sorting Algorithms” which contains all sorting algorithms that can be called from any other classes. Another is “sorting Test” which is the class that we will use to test these sorting algorithms. The “sorting Test” class, will provide the amount of swaps of each sorting algorithm and the runtime (in millisecond) to execute a sort. In the experiment we will measure the runtime in millisecond because it can show the different of each algorithm better than using second. Each algorithm takes short time to execute a sort. It takes less than a second for a big size of array (n=50,000).

# Declaration

We hereby declare that, this project was done under CSE499 and has not been submitted elsewhere for requirement of any degree or diploma or for any purpose

Signature of the students

---

**Nusrat Chowdhury**

2009-3-60-002

Department of Computer Science and Engineering
East West University

# Letter of Acceptance

I hereby declare that this thesis is from the student's own work and best effort of mine, and all other source of information used have been acknowledge. This thesis has been submitted with my approval.

**Board of Examiners**

---

**Dr. Shamim Akhter**
Assistant Professor
Department of Computer Science and Engineering
East West University

---

**Dr. Shamim H Ripon**
Associate Professor and Chairperson
Department of Computer Science and Engineering
East West University

# Acknowledgement

Firstly, my most heartfelt gratitude goes to my beloved parents for their endless support, continuous inspiration, great contribution and perfect guidance from the beginning to end.

I owe my thankfulness to my supervisor Dr. Shamim Ak hter for his skilled, utmost direction, encouragement and care to prepare myself.

My sincere gratefulness for the faculty of Computer Science and Engineering whose friendly attitude and enthusiastic support that has given me for four years.

I am very grateful for the motivation and stimulation from my good friends and seniors.

I also thank the researchers for their works that help me to learn and implement Time analysis of different sorting algorithm.

# Table of Contents

# Chapter 1

## 1.1 Introduction

From time to time people ask the ageless question: Which sorting algorithm is the fastest? This question doesn't have an easy or unambiguous answer, however. The speed of sorting can depend quite heavily on the environment where the sorting is done, the type of items that are sorted and the distribution of these items.

For example, sorting a database which is so big that cannot fit into memory all at once is quite different from sorting an array of 100 integers. Not only will the implementation of the algorithm be quite different, naturally, but it may even be that the same algorithm which is fast in one case is slow in the other. Also sorting an array may be different from sorting a linked list, for example.

In this study I will only concentrate on sorting items in an array in memory using comparison sorting (because that's the only sorting method that can be easily implemented for any item type, as long as they can be compared with the less-than operator).

## 1.2 Objectives

The objective of my work is able to:

- construct a simple user interface using appropriate Java libraries
- evaluate user interfaces using appropriate techniques
- describe the prototyping cycle
- describe the use of empirical testing in the evaluation of design alternatives
- describe and apply cognitive walkthrough for the evaluation of designs
- produce usability requirements for an application context
- describe the facilities offered by current window systems
- implement a simple user interface in Java

It describes the architecture and component functionality of a User Interface class library and produce design rationales for designs in the HCI literature and for your own designs.

Even if it is just a Java application (i.e. solely consists of Java classes), JPF can be viewed as a Java Virtual Machine (JVM) in itself. The consequence is that (*.class) class files, and even the same files at times, are processed in two different ways in a JVM running JPF

- as ordinary Java classes managed and executed by the host JVM (standard Java™ library classes, JPF implementation classes)
- as "modeled" classes managed and processed (verified) by JPF

Class lookup in both layers is based on the CLASSPATH environment variable / command line parameter, but this should not obfuscate the fact that we have to clearly distinguish between these two modes. In particular, JPF (i.e. the "Model" layer) has its own class and object model, which is completely different and incompatible to the (hidden) class and object models of the underlying host JVM executing JPF

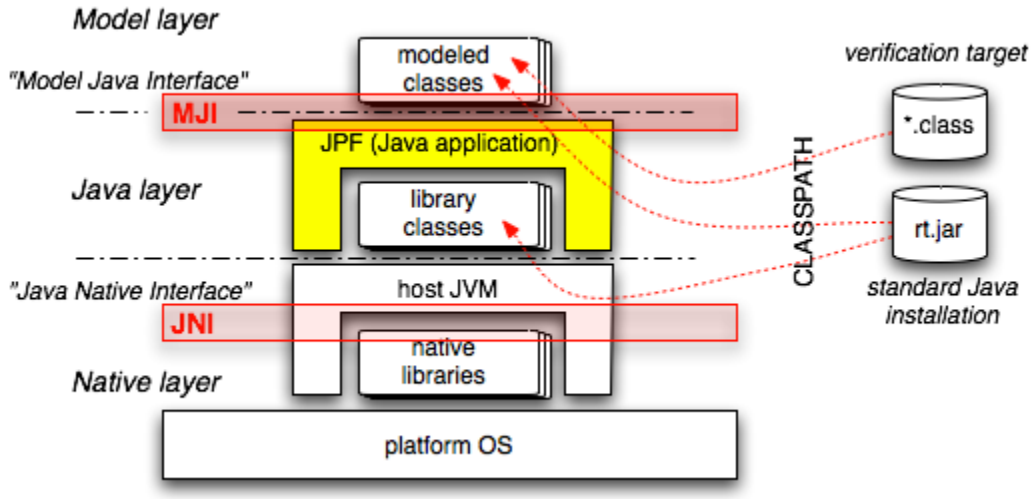


Figure. 1: Java Virtual Machine

## 1.3 Motivation

Human computer interaction is arguably the most important topic to be studied as part of any computing science course. Here are some of the reasons why it is important to study this topic:

- Unless we understand the needs of our users then there is little prospect that we will be able to support their tasks. This is a non-trivial problem. Users may not be able to tell you what they would like their system to do. If they have never used a computer, they may have unrealistic expectations. Even if they are familiar with computer systems then it may be difficult to look beyond the applications that they already use. For example, try to imagine what the successor to the Macintosh's operating system or Windows98 might look like. It is difficult to understand what the user is doing even with their present systems. For instance, it is crazy to ask someone what they do in their working day. Most people have thousands of tasks that vary over time - it's hard to know where to begin. One way round this is to watch people and record the activities that any new system must support. However, people will alter their behavior when they know that somebody is watching them use a system. This phenomenon has become known as the Hawthorne effect after a 1939 study of car workers in which output increased just because people were studying their production techniques. Human computer interaction addresses these problems by providing analytical techniques that can be used to identify users' real world activities so that designers are better prepared to support those tasks when they build computer systems.

## 1.4 Safety

People make mistakes when they use computer systems. They inadvertently delete files. They ignore warnings and fail to read help files. They type the wrong input when asked to provide information to their systems. Such "errors" are to be expected. Whilst they may have only a minimal impact upon most office systems, they have more serious implications as computer systems are integrated into process control applications. Many recent aviation accidents that were blamed upon pilot error were originally caused by a well-known HCI "mode confusion" problem. This occurred when pilots thought that they were being asked to provide one set of figures but the system was, in fact, expecting another set of figures. Human computer interaction provides a range of evaluation techniques that can be used to detect situations in which such "errors" are likely to occur.

## 1.5 Equality and Accessibility

Computers provide their users with access to vast amounts of information. The techniques that we have devised for people to interact with these devices also prevents many users from accessing this information.Screen-readers can convert textual displays into spoken output but these applications will not work for the user interfaces that dominate today's mass market applications.

# Chapter 2

## 2.1 Background study

In order to test the speed of the different sorting algorithms I made a C++ program which runs each algorithm several times for randomly-generated arrays.

The test was run in a Windows Machine With JVM version 7

The rand() function of glibc was used for random number generation. This should be a rather high-quality random number generator.

Four different array sizes were used: 100, 5000, 100000 and 1 million items (the last one used only on the integer array tests). Random numbers between 0 and 10 times the array size were generated to create the array contents, except for the high-repetition test, in which numbers between 0 and 1/100 times the array size were generated (which means that each item is repeated in average 100 times).

Four different random number distributions were used:

1. Completely random.
2. Almost sorted: 90% of the items are in increasing order, but 10% of randomly-chosen items are random.
3. Almost reversed: Like above, but the sorted items are in reverse order.
4. The array is already sorted, except for the last 256 items which are random. (This case was used to test which sorting algorithm would be best for this kind of data container, where items are kept sorted and new items are added to the end, and the entire container sorted after the amount of items at the end grows too large.)

Four different test cases were run:

1. Items are 32-bit integers. These are both very fast to compare and copy.
2. Also 32-bit integers, but with a high number of repetitions. Each value in the array repeats approximately 100 times in average.
3. Items are C++ strings with identical beginnings. Strings of 50 characters (with only the last 8 characters differing) were used for the test. This tests the case where copying is fast but comparison is slow (copying is fast because the strings in gcc use copy-on-write).
4. Items are arrays of integers. Arrays of 50 integers (ie. 200 bytes) were used. Only the first integer was used for the comparison. This tests the case where comparison is fast but copying is slow.

More detailed info for these test cases is given in their individual pages.

I tried to implement the program so that it first counts how much time is spent generating the data to be sorted, and then this time is subtracted from the total time (before dividing it by the

number of loops). While it's not possible to do this in a very exact way, I'm confident that the results are close enough to reality.

Each test case with each sorting algorithm was run several times, every time with different random data (about 100-10000 times depending on the size of the array). This was done to average out individual worst cases.

Most Sorting Algorithm work by comparing the being sorted. It may be desirable to sort

large chunk of data (for instance, a struct containing a name and address) based on only a portion of that data. The piece of data actually used to determine the sorted order is called the key.

Sorting algorithms are usually judged by their efficiency. In this case, efficiency refers to the algorithmic efficiency as the size of the input grows large and is generally based on the number of elements to sort. Most of the algorithms in use have an algorithmic efficiency of either O(n^2) or O(n*log(n)). A few special case algorithms can sort certain data sets faster than O(n*log(n)). These algorithms are not based on comparing the items being sorted and rely on tricks. It has been shown that no key-comparison algorithm can perform better than O(n*log(n)).

Many algorithms that have the same efficiency do not have the same speed on the same input. First, algorithms must be judged based on their average case, best case, and worst case efficiency. Some algorithms, such as quick sort, perform exceptionally well for some inputs, but horribly for others. Other algorithms, such as merge sort, are unaffected by the order of input data. Even a modified version of bubble sort can finish in O(n) for the most favorable inputs.

A second factor is the "constant term". As Big-O notation abstracts away many of the details of a process, it is quite useful for looking at the big picture. But one thing that gets dropped out is the constant in front of the expression: for instance, O(c*n) is just O(n). In the real world, the constant, c, will vary across different algorithms.

A second criterion for judging algorithms is their space requirement. Some algorithms never require extra space, whereas some are most easily understood when implemented with extra space. Space requirements may even depend on the data structure used (merge sort on arrays versus merge sort on linked lists, for instance).

A third criterion is stability. Most simple sorts do just this, but some sorts, do not.

# Chapter 3

## 3.1 Proposed Model

Interfaces are used to encode similarities which the classes of various types share, but do not necessarily constitute a class relationship. For instance, a human and a parrot can both whistle; however, it would not make sense to represent Humans and Parrots as subclasses of a Whistler class. Rather they would most likely be subclasses of an Animal class (likely with intermediate classes), but both would implement the Whistler interface.

Another use of interfaces is being able to use an object without knowing its type of class, but rather only that it implements a certain interface. For instance, if one were annoyed by a whistling noise, one may not know whether it is a human or a parrot, because all that could be determined is that a whistler is whistling. The call whistler.whistle() will call the implemented method whistle of object whistler no matter what class it has, provided it implements Whistler. In a more practical example, a sorting algorithm may expect an object of type Comparable. Thus, without knowing the specific type, it knows that objects of that type can somehow be sorted.

# Chapter 4

## 4.1 Implementation

To implement this project the software tools that used here are open source , that mean it is free to download , which reduce the cost to buy software for implementing project .For design the project HTML, Jawing and Java are used for design.

# Chapter 5

## 5.1 Program Code Algorithm

```
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
packagealgo;

importjava.awt.Graphics2D;
importjava.awt.HeadlessException;
importjava.awt.RenderingHints;
importjava.awt.event.ActionEvent;
importjava.awt.event.ActionListener;
importjava.awt.image.BufferedImage;
importjava.io.File;
importjava.io.FileInputStream;
importjava.io.FileNotFoundException;
importjava.io.IOException;
importjava.io.PrintWriter;
importjava.io.UnsupportedEncodingException;
importjava.util.Random;
importjava.util.logging.Level;
importjava.util.logging.Logger;
importjavax.imageio.ImageIO;
importjavax.swing.ImageIcon;
importjavax.swing.JComboBox;
importjavax.swing.JFrame;
importjavax.swing.JOptionPane;
importjavax.swing.WindowConstants;
importjavax.swing.event.DocumentEvent;
importjavax.swing.event.DocumentListener;

importorg.jfree.chart.ChartFactory;
importorg.jfree.chart.ChartPanel;
importorg.jfree.chart.JFreeChart;
importorg.jfree.chart.plot.PlotOrientation;
importorg.jfree.data.category.CategoryDataset;
importorg.jfree.data.category.DefaultCategoryDataset;
importorg.jfree.ui.RefineryUtilities;

/**
 *
 * @author Nusrat
 */
classPlotBarextendsJFrame{

publicPlotBar(String applicationTitle, String chartTitle, String btn)throwsFileNotFoundException,IOException{
super(applicationTitle);
JFreeChartbarChart=ChartFactory.createBarChart(
chartTitle,
"Algorithms",
```

```
"Execution Time(ms)",
createDataset(btn),
PlotOrientation.VERTICAL,
true,true,false);
this.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
ChartPanelchartPanel=newChartPanel(barChart);
chartPanel.setPreferredSize(newjava.awt.Dimension(550,420));
setContentPane(chartPanel);
}

doublefileReadTime(String fileName)throwsFileNotFoundException,IOException{
      File file=new File(fileName);
FileInputStreamfis=newFileInputStream(file);
byte[] data =newbyte[(int)file.length()];
fis.read(data);
fis.close();
      String str=newString(data,"UTF-8");
returnDouble.parseDouble((str.isEmpty())?"0.0":str);
}

privateCategoryDatasetcreateDataset(String btn)throwsFileNotFoundException,IOException{
final String BinarySearchMergeSort="Binary";
final String InterpolationSearchMerge="Interpolation";
final String NormalMergeSort="Normal";
final String TanSort="TAN";
finalDefaultCategoryDataset dataset =newDefaultCategoryDataset();

if(btn.equals("COMPARE")){
dataset.addValue(fileReadTime("time_binary_search_merge_sort.txt")*1000,BinarySearchMergeSort,BinarySearch
MergeSort);
dataset.addValue(fileReadTime("time_interpolation_search_merge_sort.txt")*1000,InterpolationSearchMerge,Interp
olationSearchMerge);
dataset.addValue(fileReadTime("time_normal_merge_sort.txt")*1000,NormalMergeSort,NormalMergeSort);
dataset.addValue(fileReadTime("time_tan_sor2.txt")*1000,TanSort,TanSort);
}elseif(btn.equals("PLOT")){
switch(Algorithm.comboAlgo.getSelectedIndex()){
/// For Heap.jpg image
case0:
JOptionPane.showMessageDialog(null,"Select Algorithm To Plot Time","Algorithm
Selection",JOptionPane.INFORMATION_MESSAGE);
break;
/// For merge.jpg
case1:
dataset.addValue(fileReadTime("time_binary_search_merge_sort.txt")*1000,BinarySearchMergeSort,BinarySearch
MergeSort);
break;
/// For quick.jpg
case2:
dataset.addValue(fileReadTime("time_interpolation_search_merge_sort.txt")*1000,InterpolationSearchMerge,Interp
olationSearchMerge);
case3:
dataset.addValue(fileReadTime("time_normal_merge_sort.txt")*1000,NormalMergeSort,NormalMergeSort);
break;

case4:
dataset.addValue(fileReadTime("time_tan_sor2.txt")*1000,TanSort,TanSort);
```

```
break;
default:
break;
}
}

return dataset;
}
}

publicclass Algorithm extendsjavax.swing.JFrameimplementsActionListener{

JComboBoxcb;

/**
    * Creates new form Algorithm
    *
    * @throws java.io.IOException
    */
publicAlgorithm()throwsIOException{
setTitle("Algorithm Simulation");
initComponents();
BufferedImagewPic= ImageIO.read(ClassLoader.getSystemResource("algo/res/bubbles.jpg"));
imageLabel.setIcon(newImageIcon(wPic));
// Listen for changes in the text
datasetGen.getDocument().addDocumentListener(newDocumentListener(){
publicvoidchangedUpdate(DocumentEvent e){
warn();
}

publicvoidremoveUpdate(DocumentEvent e){
warn();
}

publicvoidinsertUpdate(DocumentEvent e){
warn();
}

publicvoidwarn(){
try{
if(Integer.parseInt(datasetGen.getText())<=0||Integer.parseInt(datasetGen.getText())>50000){
JOptionPane.showMessageDialog(null,
"Error: Please enter number 1 to 50000","Error Massage",
JOptionPane.ERROR_MESSAGE);
}
}catch(NumberFormatException|HeadlessException e){
}

}

});
}

/**
    * This method is called from within the constructor to initialize the form.
    * WARNING: Do NOT modify this code. The content of this method is always
```

```
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
// <editor-fold defaultstate="collapsed" desc="Generated Code">//GEN-BEGIN:initComponents
privatevoidinitComponents(){

comboAlgo=newjavax.swing.JComboBox();
buttonOk=newjavax.swing.JButton();
imageLabel=newjavax.swing.JLabel();
labelExecutionTime=newjavax.swing.JLabel();
labelExecutionOutput=newjavax.swing.JLabel();
labelExecution=newjavax.swing.JLabel();
labelExecutionTimeElapsed=newjavax.swing.JLabel();
btnPlot=newjavax.swing.JButton();
datasetGen=newjavax.swing.JTextField();
btnGenerate=newjavax.swing.JButton();
btnCompare=newjavax.swing.JButton();

        setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setPreferredSize(newjava.awt.Dimension(550,420));

comboAlgo.setModel(newjavax.swing.DefaultComboBoxModel(new String[]{"Select Algorithm","Binary Search
Merge Sort","Interpolation Search Merge Sort","Normal Merge Sort","Tan Sort"}));
comboAlgo.addActionListener(newjava.awt.event.ActionListener(){
publicvoidactionPerformed(java.awt.event.ActionEventevt){
comboAlgoActionPerformed(evt);
}
});

buttonOk.setText("Run");
buttonOk.addActionListener(newjava.awt.event.ActionListener(){
publicvoidactionPerformed(java.awt.event.ActionEventevt){
buttonOkActionPerformed(evt);
}
});

labelExecutionTime.setText("Time Elapsed :");
labelExecutionTime.setToolTipText("");

labelExecution.setText("Execution :");

btnPlot.setText("Plot");
btnPlot.addActionListener(newjava.awt.event.ActionListener(){
publicvoidactionPerformed(java.awt.event.ActionEventevt){
btnPlotActionPerformed(evt);
}
});

btnGenerate.setText("Generate Dataset");
btnGenerate.addActionListener(newjava.awt.event.ActionListener(){
publicvoidactionPerformed(java.awt.event.ActionEventevt){
btnGenerateActionPerformed(evt);
}
});

btnCompare.setText("Compare");
```

```
btnCompare.addActionListener(newjava.awt.event.ActionListener(){
publicvoidactionPerformed(java.awt.event.ActionEventevt){
btnCompareActionPerformed(evt);
}
});

javax.swing.GroupLayout layout =newjavax.swing.GroupLayout(getContentPane());
getContentPane().setLayout(layout);
layout.setHorizontalGroup(
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addGap(30,30,30)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addComponent(labelExecution,javax.swing.GroupLayout.PREFERRED_SIZE,61,javax.swing.GroupLayout.PREF
ERRED_SIZE)
.addGap(41,41,41)
.addComponent(labelExecutionOutput,javax.swing.GroupLayout.DEFAULT_SIZE,javax.swing.GroupLayout.DEF
AULT_SIZE,Short.MAX_VALUE))
.addGroup(layout.createSequentialGroup()
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,false)
.addGroup(layout.createSequentialGroup()
.addComponent(labelExecutionTime,javax.swing.GroupLayout.PREFERRED_SIZE,97,javax.swing.GroupLayout.P
REFERRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(labelExecutionTimeElapsed,javax.swing.GroupLayout.PREFERRED_SIZE,159,javax.swing.Group
Layout.PREFERRED_SIZE))
.addGroup(layout.createSequentialGroup()
.addComponent(comboAlgo,javax.swing.GroupLayout.PREFERRED_SIZE,219,javax.swing.GroupLayout.PREFE
RRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
.addComponent(datasetGen,javax.swing.GroupLayout.PREFERRED_SIZE,104,javax.swing.GroupLayout.PREFER
RED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(btnGenerate,javax.swing.GroupLayout.DEFAULT_SIZE,javax.swing.GroupLayout.DEFAULT_SI
ZE,Short.MAX_VALUE))
.addGroup(layout.createSequentialGroup()
.addComponent(imageLabel,javax.swing.GroupLayout.PREFERRED_SIZE,416,javax.swing.GroupLayout.PREFE
RRED_SIZE)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addComponent(btnCompare)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,false)
.addComponent(buttonOk,javax.swing.GroupLayout.DEFAULT_SIZE,75,Short.MAX_VALUE)
.addComponent(btnPlot,javax.swing.GroupLayout.DEFAULT_SIZE,javax.swing.GroupLayout.DEFAULT_SIZE,S
hort.MAX_VALUE)))))
.addContainerGap(12,Short.MAX_VALUE))))
);
layout.setVerticalGroup(
layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addGap(18,18,18)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
.addGroup(layout.createSequentialGroup()
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
.addComponent(comboAlgo,javax.swing.GroupLayout.PREFERRED_SIZE,javax.swing.GroupLayout.DEFAULT_
```

```
SIZE,javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(datasetGen,javax.swing.GroupLayout.PREFERRED_SIZE,javax.swing.GroupLayout.DEFAULT_
SIZE,javax.swing.GroupLayout.PREFERRED_SIZE)
.addComponent(btnGenerate))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
.addComponent(imageLabel,javax.swing.GroupLayout.PREFERRED_SIZE,250,javax.swing.GroupLayout.PREFE
RRED_SIZE))
.addGroup(layout.createSequentialGroup()
.addGap(37,37,37)
.addComponent(buttonOk)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
.addComponent(btnPlot)
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
.addComponent(btnCompare)))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,javax.swing.GroupLayout.DEFAULT
_SIZE,Short.MAX_VALUE)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,false)
.addComponent(labelExecution,javax.swing.GroupLayout.DEFAULT_SIZE,29,Short.MAX_VALUE)
.addComponent(labelExecutionOutput,javax.swing.GroupLayout.DEFAULT_SIZE,javax.swing.GroupLayout.DEF
AULT_SIZE,Short.MAX_VALUE))
.addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
.addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,false)
.addComponent(labelExecutionTime,javax.swing.GroupLayout.DEFAULT_SIZE,29,Short.MAX_VALUE)
.addComponent(labelExecutionTimeElapsed,javax.swing.GroupLayout.DEFAULT_SIZE,javax.swing.GroupLayout
.DEFAULT_SIZE,Short.MAX_VALUE))
.addContainerGap())
);

pack();
}// </editor-fold>//GEN-END:initComponents

privatevoidcomboAlgoActionPerformed(java.awt.event.ActionEventevt){//GEN-
FIRST:event_comboAlgoActionPerformed
// TODO add your handling code here:
//       cb = (JComboBox) evt.getSource();
//       String comboSelected = (String) cb.getSelectedItem();
//       System.out.println(comboSelected);
/// For bubbles.jpg image
switch(comboAlgo.getSelectedIndex()){
/// For Heap.jpg image
case0:
try{
BufferedImagewPic= ImageIO.read(ClassLoader.getSystemResource("algo/res/bubbles.jpg"));
//JLabelwIcon = new JLabel(new ImageIcon(original));

labelExecutionOutput.setText("");

imageLabel.setIcon(newImageIcon(wPic));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
break;
/// For merge.jpg
case1:
try{
```

```
BufferedImage original = ImageIO.read(ClassLoader.getSystemResource("algo/res/heap.jpg"));
//JLabelwIcon = new JLabel(new ImageIcon(original));

labelExecutionOutput.setText("");

//double widthFactor = .4;
//double heightFactor = .4;
// imageLabel.setIcon(new ImageIcon(bufferResize(original, widthFactor, heightFactor)));
imageLabel.setIcon(newImageIcon(original));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
break;
/// For quick.jpg
case2:
try{
BufferedImage original = ImageIO.read(ClassLoader.getSystemResource("algo/res/merge.jpg"));

//double widthFactor = .3;
//double heightFactor = .38;
labelExecutionOutput.setText("");

// imageLabel.setIcon(new ImageIcon(bufferResize(original, widthFactor, heightFactor)));
imageLabel.setIcon(newImageIcon(original));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
break;
case3:
try{
BufferedImage original = ImageIO.read(ClassLoader.getSystemResource("algo/res/quick.png"));
doublewidthFactor=.8;
doubleheightFactor=.9;
labelExecutionOutput.setText("");
imageLabel.setIcon(newImageIcon(bufferResize(original,widthFactor,heightFactor)));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
break;

case4:

try{
BufferedImage original = ImageIO.read(ClassLoader.getSystemResource("algo/res/chess.png"));
//double widthFactor = .5;
//double heightFactor = .5;
labelExecutionOutput.setText("");
//imageLabel.setIcon(new ImageIcon(bufferResize(original, widthFactor, heightFactor)));
imageLabel.setIcon(newImageIcon(original));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
```

```
break;
default:
break;
}

}//GEN-LAST:event_comboAlgoActionPerformed

privateBufferedImagebufferResize(BufferedImage original,doublewidthFactor,doubleheightFactor){

// original image width & height
int w, h;
      w =original.getHeight();
      h =original.getWidth();
//       System.out.println(original.getHeight());
//       System.out.println(original.getWidth());

// new width & height calculated by multiplying factor
intnewWidth=new Double(original.getWidth()*widthFactor).intValue();
intnewHeight=new Double(original.getWidth()*heightFactor).intValue();

// new resized image
BufferedImagebuffResized=newBufferedImage(newWidth,newHeight,original.getType());

      Graphics2D g =buffResized.createGraphics();

g.setRenderingHint(RenderingHints.KEY_INTERPOLATION,RenderingHints.VALUE_INTERPOLATION_BILI
NEAR);
g.drawImage(original,0,0,newWidth,newHeight,0,0, w, h,null);
g.dispose();

returnbuffResized;
}


privatevoidbuttonOkActionPerformed(java.awt.event.ActionEventevt){//GEN-
FIRST:event_buttonOkActionPerformed
long time;
/// For bubbles.jpg image
switch(comboAlgo.getSelectedIndex()){
/// For Bubble.jpg image
case0:
JOptionPane.showMessageDialog(null,"Select                                                    Algorithm","Algorithm
Selection",JOptionPane.INFORMATION_MESSAGE);

break;
/// For merge.jpg
case1:

labelExecutionOutput.setText("Binary Search Merge Sort");
try{
// New Code Added
longstartTime=System.currentTimeMillis();
System.out.println(comboAlgo.getSelectedItem().toString());
                Runtime rt=Runtime.getRuntime();
                Process pr=rt.exec(new String[]{"cmd.exe",
"/c",
```

```
"start",
"Binary_search_merge_sort.exe"
});
// New Code Added
                time =(System.currentTimeMillis()-startTime);
System.out.println("\n Time Elapsed Binary Merge sort: "
+ time +" ms");

labelExecutionTimeElapsed.setText(String.valueOf(time +" ms"));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}

break;
/// For quick.jpg
case2:

labelExecutionOutput.setText("Interpolation Merge Sort");
try{
// New Code Added
longstartTime=System.currentTimeMillis();
System.out.println(comboAlgo.getSelectedItem().toString());
                Runtime rt=Runtime.getRuntime();
                Process pr=rt.exec(new String[]{"cmd.exe",
"/c",
"start",
"interpolation_search_merge_sort.exe"
});
// New Code Added
                time =System.currentTimeMillis()-startTime;
System.out.println("\n Time Elapsed Interpolation Merge Sort: "
+ time +" ms");
labelExecutionTimeElapsed.setText(String.valueOf(time +" ms"));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}

break;
case3:

labelExecutionOutput.setText("Normal Merge Sort");
try{
// New Code Added
longstartTime=System.currentTimeMillis();
System.out.println(comboAlgo.getSelectedItem().toString());
                Runtime rt=Runtime.getRuntime();
                Process pr=rt.exec(new String[]{"cmd.exe",
"/c",
"start",
"normal_merge_sort.exe"
});
// New Code Added
                time =System.currentTimeMillis()-startTime;
System.out.println("\n Time Elapsed Normal Merge Sort: "
```

```
+ time +" ms");
labelExecutionTimeElapsed.setText(String.valueOf(time +" ms"));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}

break;

case4:

labelExecutionOutput.setText("TAN Sort");
try{
// New Code Added
longstartTime=System.currentTimeMillis();
System.out.println(comboAlgo.getSelectedItem().toString());
                Runtime rt=Runtime.getRuntime();
                Process pr=rt.exec(new String[]{"cmd.exe",
"/c",
"start",
"TAN_SOR2.exe"
});
// New Code Added
                time =System.currentTimeMillis()-startTime;
System.out.println("\n Time Elapsed TAN Sort: "
+(System.currentTimeMillis()-startTime)+" ms");
labelExecutionTimeElapsed.setText(String.valueOf(time +" ms"));

}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}

break;
default:
break;
}

}//GEN-LAST:event_buttonOkActionPerformed

privatevoidbtnPlotActionPerformed(java.awt.event.ActionEventevt){//GEN-FIRST:event_btnPlotActionPerformed
try{
if(Algorithm.comboAlgo.getSelectedIndex()==0){
JOptionPane.showMessageDialog(null,"Select Algorithm To Plot Time","Algorithm
Selection",JOptionPane.INFORMATION_MESSAGE);
}else{
PlotBar chart =newPlotBar("Algorithm Execution","Time Chart of Algorithm","PLOT");
chart.pack();
RefineryUtilities.centerFrameOnScreen(chart);
chart.setVisible(true);
}
}catch(FileNotFoundException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
}//GEN-LAST:event_btnPlotActionPerformed
```

```
privatevoidbtnGenerateActionPerformed(java.awt.event.ActionEventevt){//GEN-
FIRST:event_btnGenerateActionPerformed
        Random randomGenerate=newRandom();

if(!datasetGen.getText().isEmpty()){
            String range =datasetGen.getText();
            Integer n =Integer.parseInt(range);
if(n >50000){
                n =50000;
}
try{
PrintWriter pw =newPrintWriter("data.txt","UTF-8");
for(inti=0;i< n;i++){
pw.println(randomGenerate.nextInt(50000));
}
pw.close();
JOptionPane.showMessageDialog(null,"Dataset                                                    Generated","Data
Generation",JOptionPane.INFORMATION_MESSAGE);
}catch(FileNotFoundException|UnsupportedEncodingException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
}else{
JOptionPane.showMessageDialog(null,"Please                    Input                    Range","Range
Input",JOptionPane.INFORMATION_MESSAGE);
}

}//GEN-LAST:event_btnGenerateActionPerformed

privatevoidbtnCompareActionPerformed(java.awt.event.ActionEventevt){//GEN-
FIRST:event_btnCompareActionPerformed
try{
PlotBar chart =newPlotBar("Algorithm Execution Plot","Comparison of Sorting Algorithms","COMPARE");
chart.pack();
RefineryUtilities.centerFrameOnScreen(chart);
chart.setVisible(true);
}catch(FileNotFoundException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
}//GEN-LAST:event_btnCompareActionPerformed

/**
    * @paramargs the command line arguments
    */
publicstaticvoidmain(String args[]){
/* Set the Nimbus look and feel */
//<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
/* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
        * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
        */
try{
for(javax.swing.UIManager.LookAndFeelInfo info :javax.swing.UIManager.getInstalledLookAndFeels()){
if("Nimbus".equals(info.getName())){
javax.swing.UIManager.setLookAndFeel(info.getClassName());
```

```
break;
}
}
}catch(ClassNotFoundException ex){
java.util.logging.Logger.getLogger(Algorithm.class.getName()).log(java.util.logging.Level.SEVERE,null, ex);
}catch(InstantiationException ex){
java.util.logging.Logger.getLogger(Algorithm.class.getName()).log(java.util.logging.Level.SEVERE,null, ex);
}catch(IllegalAccessException ex){
java.util.logging.Logger.getLogger(Algorithm.class.getName()).log(java.util.logging.Level.SEVERE,null, ex);
}catch(javax.swing.UnsupportedLookAndFeelException ex){
java.util.logging.Logger.getLogger(Algorithm.class.getName()).log(java.util.logging.Level.SEVERE,null, ex);
}
//</editor-fold>

/* Create and display the form */
java.awt.EventQueue.invokeLater(new Runnable(){
publicvoidrun(){
try{
new Algorithm().setVisible(true);
}catch(IOException ex){
Logger.getLogger(Algorithm.class.getName()).log(Level.SEVERE,null, ex);
}
}
});

}

// Variables declaration - do not modify//GEN-BEGIN:variables
privatejavax.swing.JButtonbtnCompare;
privatejavax.swing.JButtonbtnGenerate;
privatejavax.swing.JButtonbtnPlot;
privatejavax.swing.JButtonbuttonOk;
publicstaticjavax.swing.JComboBoxcomboAlgo;
privatejavax.swing.JTextFielddatasetGen;
privatejavax.swing.JLabelimageLabel;
privatejavax.swing.JLabellabelExecution;
privatejavax.swing.JLabellabelExecutionOutput;
privatejavax.swing.JLabellabelExecutionTime;
privatejavax.swing.JLabellabelExecutionTimeElapsed;
// End of variables declaration//GEN-END:variables


   @Override
publicvoidactionPerformed(ActionEvent e){
//throw new UnsupportedOperationException("Not supported yet."); //To change body of generated methods, choose Tools | Templates.
}
}
```

# Chapter 6

## 6.1 Conclusion and Future Works

Actually, I Want to make a bridge between C++ and Java Programs. This Work is an Introduction of This Bridge. In Future this program can be used as a module of Large Industrial Project's as a Module.

## References

1. http://www.dcs.gla.ac.uk/~johnson/teaching/hci-java/course.html

2. Savitch, Walter and Carrano, Frank. Arrays. Java: Introduction to Problem Solving and Programming (5th Edition). Prentice Hall, 9780136072256.

3. http://docs.oracle.com/javase/tutorial/uiswing/

4. http://zetcode.com/tutorials/javaswingtutorial/

5. http://www.tutorialspoint.com/java/

6. http://www.cprogramming.com/tutorial.html