VISVESVARAYA TECHNOLOGICAL UNIVERSITY

JNANA SANGAMA, BELAGAVI – 590014

A VI th Semester Mini-Project Report on

“MERGE SORT SIMULATION”

Submitted in partial fulfillment of the requirements for the award of degree of

BACHELOR OF ENGINEERING
IN
COMPUTER SCIENCE AND ENGINEERING

Submitted by:

Prathamesh Chougule	**2AG19CS047**
Shrinath Korajkar	**2AG19CS073**

Under the Guidance of
Prof. Priyanka Pujari
Assistant Professor, Dept. of CSE,
AITM, Belagavi.

ANGADI INSTITUTE OF TECHNOLOGY & MANAGEMENT
BELAGAVI-590009
2021-2022

ANGADI INSTITUTE OF TECHNOLOGY & MANAGEMENT

BELAGAVI -590009

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is to certify that Project entitled **“Merge Sort Simulation”** is work carried out by Prathamesh Chougule (2AG19CS047) and Shrinath Korajkar (2AG19CS073) in partial fulfillment of the requirements for the award of the degree of **Bachelor of Computer Science & Engineering under Visvesvaraya Technological University, Belagavi** during the year 2021-2022. It is certified that all the correction/suggestion indicated for internal assessment have been incorporated in the report. The Computer Graphics 6th Semester Mini-Project report has been approved as it satisfies the academic requirements in respect of mini-project work prescribed for the Bachelor of Engineering degree.

Signature of the Guide	Signature of the HOD	Signature of the Principal
Prof. Priyanka Pujari	Prof. Sagar Birje	Dr. Anand Deshpande
Assistant Professor,	Assistant Professor and head	Principal and Director,
Dept. of CSE, AITM	Dept. of CSE, AITM	AITM, Belagavi

Name of the Examiner:	**Signature with date:**
1.________________________	____________________
2.________________________	____________________

DECLARATION

We **Prathamesh Chougule (2AG19CS047), Shrinath Korajkar (2AG19CS073),** studying in the 6th semester of Bachelor of Engineering in Computer Science and Engineering at Angadi Institute of Technology and Management, Belagavi, hereby declare that this mini project work entitled **"Merge Sort Simulation"** which is being submitted by us in the partial fulfillment for the award of the degree of Bachelor of Engineering in Computer Science and Engineering from Visvesvaraya Technological University, Belagavi is an authentic record of us carried out during the academic year 2021-2022 under the guidance of **Prof. Priyanka Pujari,** Assistant Professor Department of Computer Science and Engineering, Angadi Institute of Technology and Management, Belagavi.

We further undertake that the matter embodied in the dissertation has not been submitted previously for the award of any degree or diploma by us to any other university or institution.

Place: Belagavi	**Prathamesh Chougule**
Date:	**Shrinath Korajkar**

ACKNOWLEDGEMENT

It is our proud privilege and duty to acknowledge the kind of help and guidance received from several people in preparation of this report. It would not have been possible to prepare this report in this form without their valuable help, cooperation and guidance.

First and foremost, we wish to record our sincere gratitude to **Management of Angadi Institute of Technology and Management, Belagavi** and to our beloved Principal, **Dr. Anand Deshpande,** Angadi Institute of Technology and Management, Belagavi for his constant support and encouragement in preparation of this report and for making available library and laboratory facilities needed to prepare this report.

Our sincere thanks to our HOD, **Prof. Sagar Birje,** Department of Computer Science and Engineering, Angadi Institute of Technology and Management, Belagavi for his valuable suggestions and guidance throughout the period of this report.

We express our sincere gratitude to our guide **Prof. Priyanka Pujari,** Assistant Professor Department of Computer Science and Engineering, Angadi Institute of Technology and Management, Belagavi for guiding us in investigations for this seminar and in carrying out experimental work. Our numerous discussions with her were extremely helpful. I hold her in esteem for guidance, encouragement and inspiration received from her.

The mini-project on **“Merge Sort Simulation”** was very helpful to us in giving the necessary background information and inspiration in choosing this topic for the project.

Last but not the least, we wish to thank our **parents** for financing our studies in this college as well as for constantly encouraging us to learn engineering. Their personal sacrifice in providing this opportunity to learn engineering is gratefully acknowledged.

Prathamesh Chougule

Shrinath Korajkar

ABSTRACT

The purpose of the project entitled as "MERGE SORT SIMULATION" is to develop an external algorithm and based on divide and conquer strategy. In this sorting: The elements are split into two sub-arrays (n/2) again and again until only one element is left. Merge sort uses additional storage for sorting the auxiliary array. Merge sort uses three arrays where two are used for storing each half, and the third external one is used to store the final sorted list by merging the other two and each array is then sorted recursively. At last, all sub-arrays are merged to make it 'n' element size of the array.

TABLE OF CONTENTS

Chapter 1

INTRODUCTION

1.1 Introduction to Computer Graphics

Computer Graphics is concerned with all the aspects of producing pictures or images using a computer. The field began humbly almost fifty years ago with the display of few lines on a cathode ray tube (CRT), Now we can create images with computer that are indistinguishable from photographs of real objects.

The development of Computer Graphics has been driven by both the needs of the user community and by the advances in hardware and software. The applications of Computer Graphics are many and varied. However, we can divide them into 7 major areas –

1. Display of Information.
2. Design.
3. Simulation and Animation.
4. User Interfaces.
5. Graphs and Charts, CAD, Data Visualizations.
6. Image Processing, Education and Training, Entertainment etc**.**

1.2 Introduction to OpenGL

It is a software interface to graphics hardware. OpenGL is an industry standard portable 3-D Computer Graphics API. OpenGL is a premiere environment for developing portable interactive 2D or 3D applications.

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex three-dimensional scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games**;** here it competes with Direct3D on Microsoft Windows platforms (see Direct3D vs. OpenGL). OpenGL is managed by the non-profit technology consortium, the Khronos Group.

Functions in the main GL library have names that begin with the letters gl and are stored in a library usually referred to as GL. The second is the Open GL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing.

To interface with the window system and to get input from external devices into our programs, we need at least one more library. For each major window system there is a system-specific library that provides the "glut" between the window system and open GL. For the X window system, this library is called as GLX, for Windows, it is gl, and for the Macintosh, it is agl. Rather than using a different library for each system, we use a readily available library called the Open GL Utility Toolkit (GLUT), which provides the minimum functionality that should be expected in modern windowing system.

1.2.1 OpenGL for a developer

1. OpenGL is not a PL but it contains pre-defined functionality.
2. Provides functions to set or get or change the state variables.
3. Provides functions to render scene onto a buffer which can then be shown in a window.
4. Platform to create simple objects and animate them using various functions.
5. Provides functions to develop more artistic options for a certain approach.
6. Provides functions to explore multiple graphics related concepts.
7. Platform Independent
8. Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

Chapter 2

METHODOLOGY AND INPUT KEYS

2.1 Program Definition

Program to implement using interactive devices like Keyboard functions and user-friendly Mouse function like Left button, Right button and Middle button. It also includes many library functions like attribute, primitive, input, transformation, viewing and control functions.

2.2 Description

A Merge Sort Simulation is a visual representation of the working of merge sort which is stable, not in place algorithm that sort's an array in O (n log n) Time Complexity and N Space Complexity. It is a divide and conquer strategy-based algorithm which divide the array in subparts and then combined in a sorted manner. We can think of it as a recursive algorithm that continuously splits the array in half until it cannot be further divided. We split array into halves and recursively invoke the merge sort on each of the halves, then finally when both halves are sorted the merge operation is applied.

The Merge Sort Simulation which performs all these functions has been implemented using OpenGL functions and contains the Menu options and the Keyboard interface. It has the following features and performs the following functions:

1. OpenGL based Merge Sort simulation which automatically displays various steps involved in sorting using merge sort.
2. The program takes input array from user and automatically sort it using merge sort. The user can give both positive and negative integers but only two-digit numbers.
3. While sorting the array each state of array is preserved after every step and displayed visually to the user.
4. The user can start and stop simulation at any instance using mouse and keyboard interface, and can also refer the rules provided in rules window.
5. The project is implemented on C++ platform with the help of OpenGL in-built functions. Care is taken to provide an easy-to-use mouse and keyboard interface involving an icon-based interaction.

2.3 Mouse Interface

Using mouse, it provides the menu that consists following options:

1. Quit
2. Home
3. Rules
4. Start Simulation
5. Stop Simulation

2.4 Keyboard Interface

Using keyboard, we can change the color of the boat or ship by pressing the corresponding keys:

1. The key ‘esc’, ‘q’ and ‘Q’ is used to Quit or Exit.
2. The key ‘n’ and ‘N’ is used to go to Next page.
3. The key ‘s’ and ‘S’ is used to Start the simulation.
4. The key ‘h’ and ‘H’ is used to display Help menu.

Chapter 3

REQUIREMENT SPECIFICATIONS

3.1 Hardware Requirements

- Processor: Pentium 3 or higher processor.
- Speed: 1.1 GHz
- RAM: 128MB or more.
- Hard Disk: 2GB (approx.).
- Display: VGA Color Monitor.
- Keyboard: Standard Keyboard
- Mouse: PS/2 mouse

3.2 Software Requirements

- Operating System: Windows 98SE/2000/XP/Vista/UBUNTU.
- IDE: Code Blocks.
- Compiler: C/C++ (gcc).
- Graphics Tool: OpenGL.
- Other: Notepad/Text Editor.

Chapter 4

ADVANTAGES AND DISADVANTAGES

4.1 Advantages of OpenGL

1. It is by nature, not restricted to a single operating system.
2. Offers quite a bit of low-level control for graphics.
3. Supports a lot of programming languages.
4. (Opinionated) People tend to say that it's easier to learn/get started with than other low-level 3D graphics-based APIs.
5. Pretty stable.
6. Is extensible i.e., new hardware features are exposed quickly.
7. Graphics based software tend to use OpenGL for their 3D back end.
8. It has lower CPU overhead for drawing calls and state changes than the other API's.
9. It is portable, cross-platform API i.e., it can be used on various platforms.
10. Support sophisticated features like shadows, fog, textures, lighting effects etc.

4.2 Disadvantages of OpenGL

1. It's only a graphics API, meaning that it does not handle anything more than graphics. Audio, controls, logic, etc must be programmed in manually.
2. Tends to run slightly slower than say DirectX because of OpenGL's robustness (and relative easiness), but of course, that's arguable. There is no one true benchmark of everything.
3. low level freedom to do anything, minimal memory management.
4. The OpenGL primitives do not retain the geometric shapes drawn; they simply write over pixels in its raster display.
5. If you want to delete the line, rescale, or do other re-editing, the application program has to send additional commands to do it. Compare with other API's where you can easily grab objects and move them about the screen.

Chapter 5

OPENGL FUNCTIONS

5.1 GLUT Functions

1. **glutInit()**
   glutInit will initialize the GLUT library and negotiate a session with the window system.

2. **glutInitDisplayMode()**
   The initial display mode is used when creating top-level windows, sub windows, and overlays to determine the OpenGL display mode for the to-be-created window or overlay.

3. **glutInitWindowSize()**
   The intent of the initial window position and size values is to provide a suggestion to the window system for a window's initial size and position.

4. **glutCreateWindow()**
   glutCreateWindow creates a top-level window. The name will be provided to the window system as the window's name.

5. **glutDisplayFunc()**
   glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called.

6. **glutmainLoop()**
   glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any call backs that have been registered.

7. **glutPostRedisplay()**
   glutPostRedisplay requests that the display call back be executed after the current call back returns.

8. **glutSwapBuffers()**
   When we have two buffers front and back, the front buffer will always be displayed where as the back buffer in which we draw. Hence the rendering is put into the display callback, to update the back buffer. When the rendering is done glutSwapBuffers is executed and the results will be displayed.

9. **glutKeyboardFunc()**
   glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed. When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored.

10. **glutIdleFunc()**
    glutIdleFunc sets the global idle callback to be func so a GLUT program can perform background processing tasks or continuous animation when window system events are not being received.

5.2 GL Functions

1. **glEnable()**
   voidglEnable(GLenum cap);
   glEnable and glDisable enable and disable various capabilities. Use glIsEnabled or glGet to determine the current setting of any capability. The initial value for each capability with the exception of GL_DITHER is GL_TRUE.

2. **glDepthFunc()**
   voidglDepthFunc(GLenumfunc);
   glDepthFunc specifies the function used to compare each incoming pixel depth value with the depth value present in the depth buffer. The comparison is performed only if depth testing is enabled.

3. **GL_PROJECTION()**
   Applies subsequent matrix operations to the projection matrix stack.

4. **glLightfv()**
   voidglLightfv(GLenum light, GLenumpname, GLfloatparam);
   glLight sets the values of individual light source parameters.
   light names the light and is a symbolic name of the form GL_LIGHT i, where i ranges from 0 to the value of GL_MAX_LIGHTS-1. pname specifies one of ten light source parameters, again by symbolic name. params is either a single value or a pointer to an array that contains the new values.

5. **glPushMatrix()**
   glPushMatrix pushes the current matrix stack down by one, duplicating the current matrix. That is, after a glPushMatrix call, the matrix on top of the stack is identical to the one below it.

6. **glClearColor()**
   voidglClearColor(GLclampfr,GLclampfg,GLclampf b, GLclamp a);
   sets the present RGBA clear colour used when clearing the colour buffer. Variable of type GLclampf are floating-point numbers between 0.0 and 1.0.

7. **glViewport()**
   voidglViewport(int x, inty,GLsizei width, GLsizei height);
   specifies a width*height viewport in pixels whose lower-left corner is at (x,y) measured from the origin of the window.

8. **glFlush()**
   voidglFlush();
   forces any buffered openGL commands to execute.

9. **glBegin()**
   This function specifies the type of the primitive that the vertices define. Every glBegin() ends with a corresponding glEnd(), which ends the list of vertices. The different primitives which we can use are GL_POLYGON, GL_LINE_STRIP, GL_POINTS, GL_LINE_STRIP, GL_LINE_LOOP etc.

10. **glTranslatef()**
    It alters the current graphic image by post-multiplication. The matrix representing the image is post-multiplie

Chapter 6

IMPLEMENTATION

The Merge Sort Simulation can be implemented using some of the OpenGL inbuilt functions along with some user defined functions. The inbuilt OpenGL functions that are used mentioned under the FUNCTIONS USED category. The user defined functions are mentioned under USER DEFINED FUNCTIONS category.

6.1 Functions Used

1. **Void glColor3f (float red, float green, float blue)**
   This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. The 'f' gives the data type float. The arguments are in the order RGB (Red, Green and Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

2. **Void glClearColor(int red, int green, int blue, int alpha)**
   This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

3. **Void glutKeyboardFunc()**
   Where func () is the new keyboard callback function. glutKeyboardFunc sets the keyboard callback for the *current window*. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The x and y callback parameters indicate the mouse location in window for relative coordinates when the key gets pressed. Prototype is as given below:

   *Void glutKeyboardFunc (void (*func) (unsigned char key, int x, int y));*

   When a new window is created, no keyboard callback is initially registered, and the ASCII keystrokes that are within the output window are ignored. Passing NULL to glutKeyboardFunc disables the generation of keyboard callbacks.

4. **Void GLflush()**
   Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. *GLflush ()* empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

5. **Void glMatrixMode(GLenum mode)**
   Where "mode" specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted are:
   GL_MODELVIEW, GL_PROJECTION and GL_TEXTURE
   The initial value is *GL_MODELVIEW*.
   The function *GlMatrixMode* sets the current matrix mode. *Mode* can assume one of these values:
   GL_MODELVIEW : Applies matrix operations to the model view matrix stack.
   GL_PROJECTION: Applies matrix operations to the projection matrix stack.

6. **void viewport(GLint x, GLint y, GLsizei width, GLsizei height)**
   Here, (x, y) specifies the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0).
   Width, height: Specifies the width and height of the viewport. When a GL context is first attached to a surface (e.g. window), width and height are set to the dimensions of that surface.
   Viewport specifies the affine transformation of *x* and *y* from normalized device coordinates to window coordinates. Let (x_{nd}, y_{nd}) be normalized device coordinates. Then the window coordinates (x_w, y_w) are computed as follows:
   $x_w =(x_{nd} +1 \ {}^{width}/_2 + x$
   $y_w = (y_{nd} + 1) \ {}^{height}/_2 + y$
   Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, we call the Glgetinteger with argument *GL_MAX_VIEWPORT_DIMS.*

7. **void glutInit (int *argc, char **argv)**
   GlutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options. GlutInit also processes command line options, but the specific options parse are window system dependent.

8. **Void glutReshapeFunc (void (*func) (int width, int height))**
   GlutReshapeFunc sets the reshape callback for the *current window*. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width and height parameters of the callback specify the new window size in pixels. Before the callback, the *current window* is set to the window that has been reshaped.
   If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply

9. **glOrtho ()**
   Syntax:
   void glOrtho (GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
   The function defines an orthographic viewing volume with all parameters measured from the center of the projection plane.

10. **void glutMainLoop(void)**
    GlutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never stop.

11. **glutPostRedisplay()**
    GlutPostRedisplay, glutPostWindowRedisplay — marks the current or specified window as needing to be redisplayed.

6.2 User Defined Functions

1. **void mergeSortAlgo(int arr[], int l, int r)**
   This function is the main function which sorts the array using merge sort. It takes user array as input, sort the array and save the state of array after each step in another array that is used for simulation.

2. **void printText(int x, int y, float r, float g, float b, void *font, char *str)**
   This function is used to print the text on window. It takes the position from where the text is to be printed, the color of the text, font of the text and the text as string input. It uses inbuilt function glutBitmapCharacter() to print text.

3. **void displayArray(int startDigit, int endDigit)**
   This function is used to display the array which is saved while sorting in simulation layers. It takes the number of digits to display and displays them at given position with specified space between them. It displays the array boxes with the numbers in them by converting the numbers into characters.

4. **void drawLeftArrow(int arrowStartx, int arrowStarty, int arrowEndx, int arrowEndy, int tilt)**
   This function is used to draw the left arrow on particular location with particular direction. It takes the start and end points of the array with the tilt or rotation angle as input and display arrow in that particular position.

5. **void drawRightArrow(int arrowStartx, int arrowStarty, int arrowEndx, int arrowEndy, int tilt)**
   This function is used to draw the right arrow on particular location with particular direction. It takes the start and end points of the array with the tilt or rotation angle as input and display arrow in that particular position.

6. **void layerDisplay()**

   This function is used to display all the layers one by one in simulation. It calls layer functions that create layers by translating the array into particular direction and calling the displayArray() to draw array.

7. **void callTimer(int start)**

   This function is used to maintain a timer that is used to display the merge sort steps or layers in the simulation at a particular interval. It uses glutTimerFunc() to create a timer.

8. **void createMenu()**

   This function creates a menu that consists of following options:

   a) START SIMULATION.
   b) STOP SIMULATION.
   c) HOME.
   d) RULES
   e) QUIT.

   Hence it provides the mouse interface to user.

9. **void keyBoard(unsigned char key, int x, int y)**

   This function is used to provide the keyboard interface to the user. Using this function we can travel between the windows, start and stop the simulation, exit from the window. It is also used to take user input array.

10. **void mergeSimulation()**

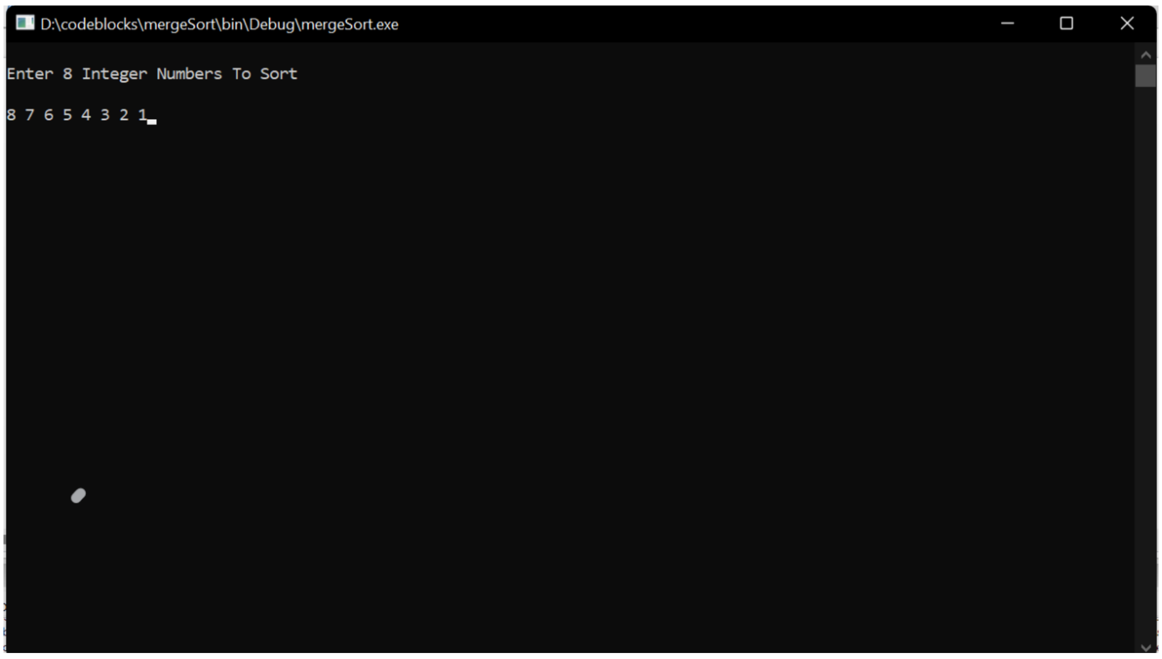
    It is the callback function used in glutDisplayFunc() that will call all other functions to create a display and simulation window.

11. **int main(int argc, char**argv)**

    Here we call all the function we defined previously in the program and this function contains other functions that create an output window on the monitor and help in redisplaying the pixels in loop.

Chapter 7

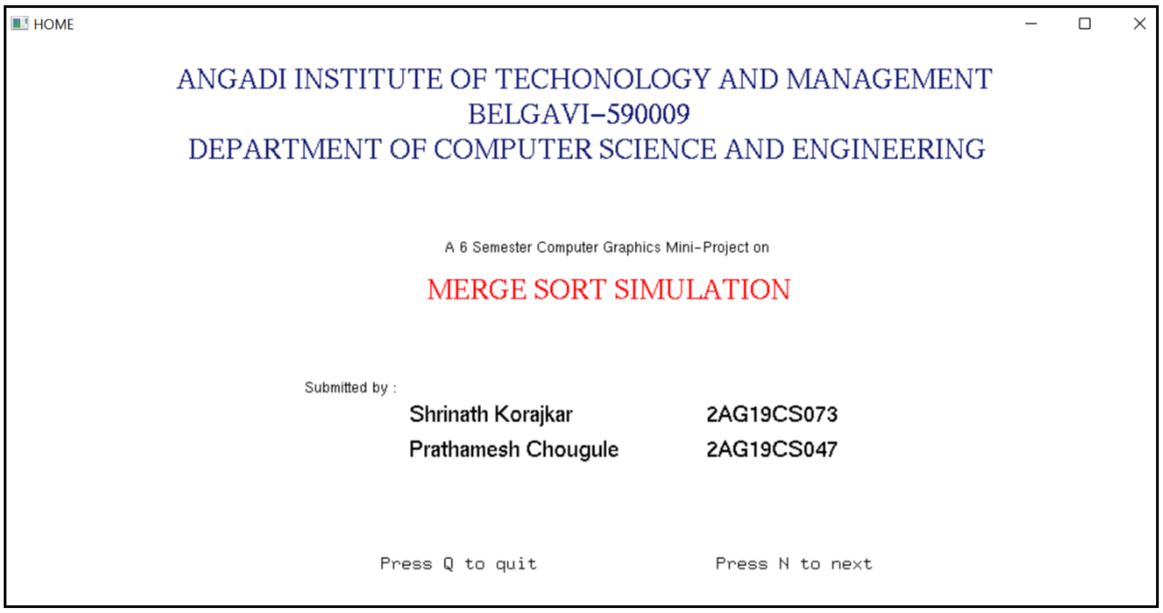
SNAPSHOTS



```
D:\codeblocks\mergeSort\bin\Debug\mergeSort.exe

Enter 8 Integer Numbers To Sort
8 7 6 5 4 3 2 1
```

Fig: 7.1: Initial User Input Array



HOME

ANGADI INSTITUTE OF TECHONOLOGY AND MANAGEMENT
BELGAVI–590009
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

A 6 Semester Computer Graphics Mini-Project on

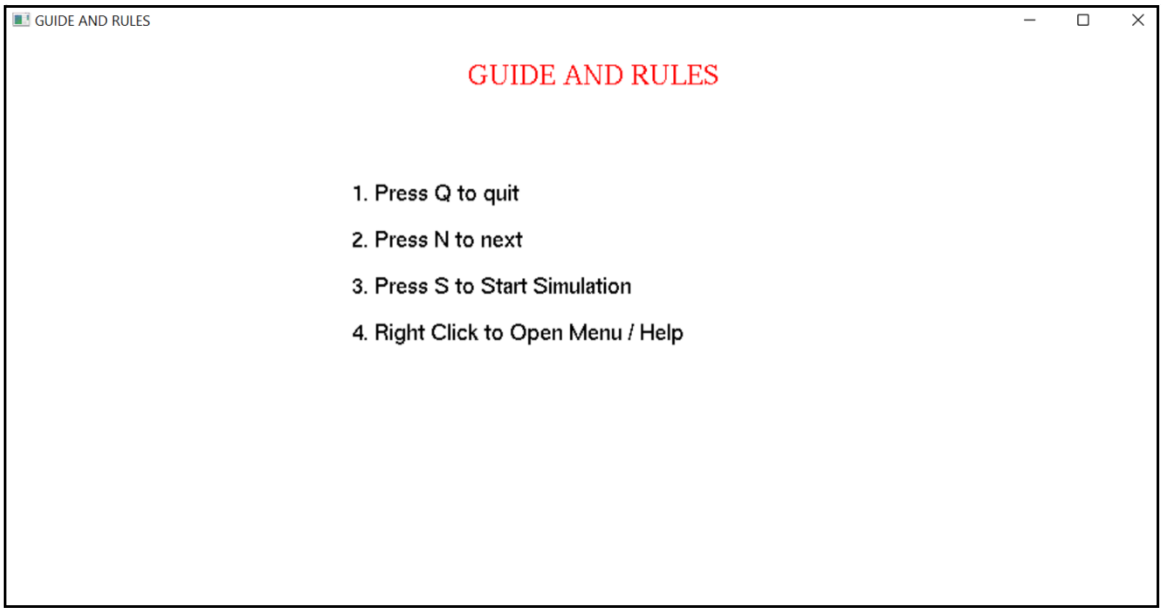
MERGE SORT SIMULATION

Submitted by :

Shrinath Korajkar	2AG19CS073
Prathamesh Chougule	2AG19CS047

Press Q to quit   Press N to next

Fig: 7.2: Front Page Window

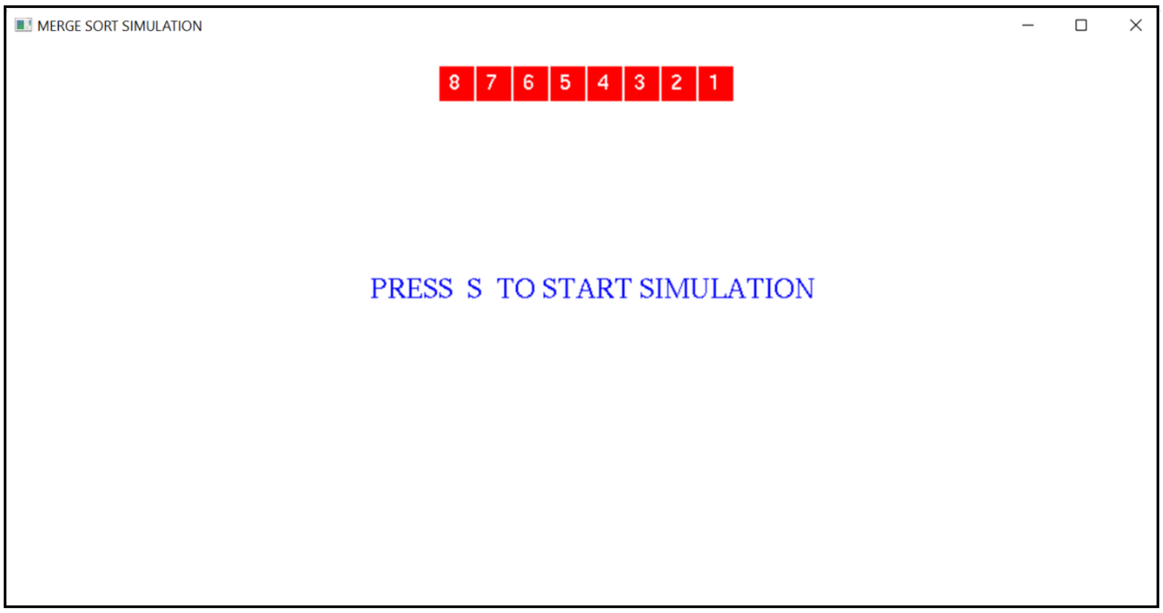


GUIDE AND RULES

GUIDE AND RULES

1. Press Q to quit
2. Press N to next
3. Press S to Start Simulation
4. Right Click to Open Menu / Help

Fig: 7.3: Guide and Rules Window



MERGE SORT SIMULATION

PRESS S TO START SIMULATION

Fig: 7.4: Main Simulation Window

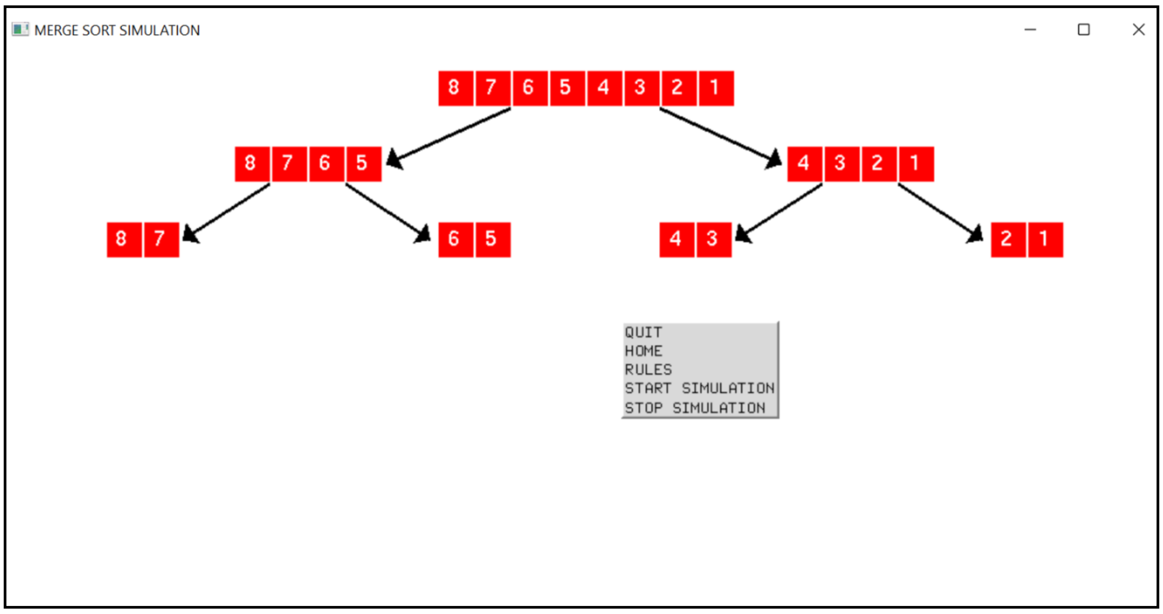


Fig: 7. 5: Simulation with Menu

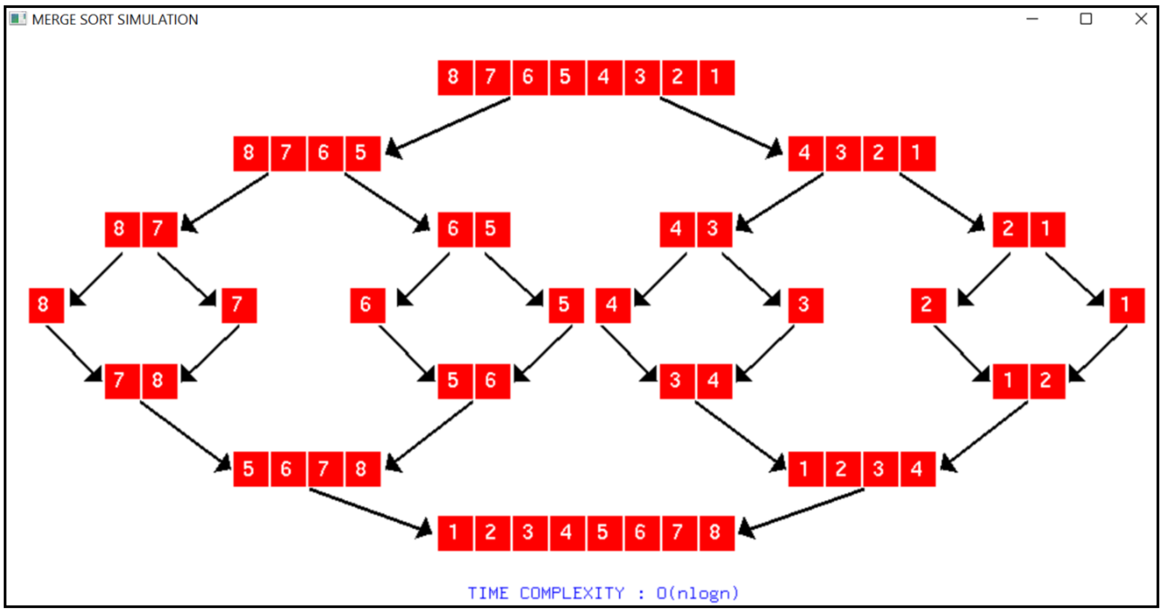


Fig: 7. 6: Complete Simulation of Merge Sort

Chapter 8

CONCLUSION

This project creates a 2-D/VIRTUAL view of Merge Sort. Here, we have created a simulation consisting of the merge sort which take user input and sort it using merge sort algorithm, then display various steps involved in sorting the given array. It is helpful for the visual understanding of the working of merge sort. It is easy to understand also we have provided mouse interface to start and stop simulation and to exit the window. We have also included the keyboard input function to traverse between the windows.

Further Enhancements:

The following are some of the features that can be included in the revised versions of this code are:

1. Generalize it for variable input length.
2. Include good animation for displaying the layers.
3. Support for advanced 3D representation of the entire scenario.
4. Support for transparency of layers and originality.
5. Make the code responsive.

REFERENCES

Book References:

- Ariponnammal, S. and Natarajan, S. (1994) ‘Transport Phenomena of Sm Sel-X Asx’, Pramana – Journal of Physics Vol.42, No.1, pp.421-425.
- Barnard, R.W. and Kellogg, C. (1980) ‘Applications of Convolution Operators to Problems in Univalent Function Theory’, Michigan Mach, J., Vol.27, pp.81–94.
- Shin, K.G. and McKay, N.D. (1984) ‘Open Loop Minimum Time Control of Mechanical Manipulations and its Applications’, Proc.Amer.Contr.Conf., San Diego, CA, pp. 1231-1236.

Web References:

- www.opengl.org
- www.google.com
- www.sourcecode.com
- www.pearsoned.co.in
- www.wikipedia.org