**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“Jnana Sangama”, Belagavi-590018, Karnataka**

****

**BANGALORE INSTITUTE OF TECHNOLOGY**

**K. R. Road, V. V. Puram, Bengaluru-560 004**



**DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING**

**Computer Graphics Laboratory With Mini Project Report-17CSL68**

**on**

**“SORTING ALGORITHM SIMULATION”**

**Submitted By**

**1BI18CS118 RAJ KAMAL**

**for the academic year 2021-22**

Under the guidance of

**Prof. Bhanushree K J** **Prof. N. Thanuja** Assistant Professor Assistant Professor

**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**

**“Jnana Sangama”, Belagavi-590018, Karnataka**

**BANGALORE INSTITUTE OF TECHNOLOGY**

**K. R. Road, V. V. Puram, Bengaluru-560 004**

****

**Department of Computer Science & Engineering**

***Certificate***

This is to certify that the implementation of **Computer Graphics Laboratory With Mini Project (17CSL68)** entitled **“SORTING ALGORITHM SIMULATION”** has been successfully completed by

**1BI18CS118 RAJ KAMAL**

of VI semester B.E. for the partial fulfillment of the requirements for the Bachelor's degree in **Computer Science & Engineering** of the **Visvesvaraya Technological University** during the academic year **2021-2022**.

**Lab In charges:**

|  |  |  |
| --- | --- | --- |
| **Prof. N.Thanuja** | **Prof. Bhanushree K. J.** | **Dr. Asha T.** |
| Assistant Professor | Assistant Professor | Professor and Head |
| Dept. of CSE, BIT | Dept. of CSE, BIT | Dept. of CSE, BIT |

Examiners: 1) 2)

**ACKNOWLEDGEMENT**

The knowledge & satisfaction that accompany the successful completion of any task would be incomplete without mention of people who made it possible, whose guidance and encouragement crowned my effort with success. I would like to thank all and acknowledge the help I have received to carry out this Mini Project.

I would like to convey my sincere thanks to **Dr. M. U. Aswath**, Principal, BIT and **Dr. Asha T.**, HOD, Department of CS&E, BIT for being kind enough to provide the necessary support to carry out the mini project.

I am most humbled to mention the enthusiastic influence provided by the lab in-charges **Prof. N. Thanuja** and **Prof. Bhanushree K. J.,** on the project for their ideas, time to time suggestions for being a constant guide and co-operation showed during the venture and making this project a great success.

I would also take this opportunity to thank my friends and family for their constant support and help. I'm very much pleasured to express my sincere gratitude to the friendly co-operation showed by all the **staff members** of Computer Science Department, BIT.

**RAJ KAMAL**

**1BI18CS118**

**Table of contents**

1. Introduction………………………………………………………… ……. 1
   1. Computer Graphics…………………………………………………… 1
   2. Application of Computer Graphics…………………………………... 1
   3. OpenGL……………………………………………………………… 3
   4. Problem Statement…………………………………………………… 4
   5. Objective of The Project …….……………………………………… 4
   6. Organization of The Project..………………………………………… 5
2. System Specification …………………………………………………….. 6
   1. Hardware Requirements……………………………………………….. 6
   2. Software Requirements……………………………………………… 6
3. Design…………………………………………………………………….. 7

3.1 Flow Diagram………………………………………………………… 7

3.2 Description of Flow Diagram………………………………………… 8

1. Implementation…………………………………………………………… 9
   1. Built In Functions…………………………………………………….. 9
   2. User Defined Functions With Modules………………………………. 13
   3. Sorce code……………………………………………………………. 14
2. Snapshots………………………………………………………………….. 27
3. Conclusion………………………………………………………………… 33

Future Enhancement………………………………………………………. 33

Bibliography……………………………………………………………….. 34

**Chapter -1**

**INTRODUCTION**

**1.1** **Computer Graphics**

Computer graphics is an art of drawing pictures, lines, charts, using computers with the help of programming. Computer graphics is made up of number of pixels. Pixel is the smallest graphical picture or unit represented on the computer screen. Basically, there are 2 types of computer graphics namely,

Interactive Computer Graphics involves a two-way communication between computer and user. The observer is given some control over the image by providing him with an input device. This helps him to signal his request to the computer.

Non-Interactive Computer Graphics otherwise known as passive computer graphics it is the computer graphics in which user does not have any kind of control over the image. Image is merely the product of static stored program and will work according to the instructions given in the program linearly. The image is totally under the control of program instructions not under the user. Example: screen savers.

**1.2 Applications of Computer Graphics**

Scientific Visualization

Scientific visualization is a branch of science, concerned with the visualization of three-dimensional phenomena, such as architectural, meteorological, medical, biological systems.

Graphic Design

The term graphic design can refer to a number of artistic and professional disciplines which focus on visual communication and presentation

Computer-aided Design

Computer-aided design (CAD) is the use of computer technology for the design of objects, real or virtual. The design of geometric models for object shapes, in particular, is often called computer-aided geometric design (CAGD). The manufacturing process is tied in to the computer description of the designed objects so that the fabrication of a product can be automated using methods that are referred to as CAM, computer-aided manufacturing.

Web Design

Web design is the skill of designing presentations of content usually hypertext or hypermedia that is delivered to an end-user through the World Wide Web, by way of a Web browser.

Digital Art

Digital art most commonly refers to art created on a computer in digital form.

Video Games

A video game is an electronic game that involves interaction with a user interface to generate visual feedback on a raster display device.

Virtual Reality

Virtual reality (VR) is a technology which allows a user to interact with a computer simulated environment. The simulated environment can be similar to the real world. This allows the designer to explore various positions of an object. Animations in virtual reality environments are used to train heavy equipment operators or to analyse the effectiveness of various cabin configurations and control placements.

Computer Simulation

A computer simulation, a computer model or a computational model is a computer program, or network of computers, that attempts to simulate an abstract model of a particular system.

Education and Training

Computer simulations have become a useful part of mathematical modelling of many natural systems in physics, chemistry and biology, human systems in economics, psychology, and social science and in the process of engineering new technology, to gain insight into the operation of those systems, or to observe their behaviour. Most simulators provide screens for visual display of the external environment with multiple panels is mounted in front of the simulator.

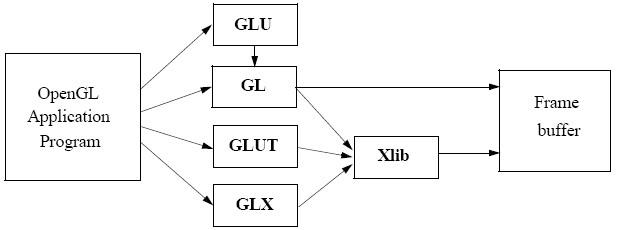
Image Processing

The modification or interpretation of existing pictures such as photographs and TV scans, is called image processing. In computer graphics, a computer is used to create a picture. Image processing techniques, on the other hand, are used to improve picture quality, analyse images, or recognize visual patterns for robotics applications

**1.3 OpenGL**

OpenGL has become a widely accepted standard for developing graphics applications. Most of our applications will be designed to access OpenGL directly through functions in the three libraries. Functions in main GL libraries have names that begin with the letters gl and are stored in a library usually referred to as GL.

The second is the OpenGL Utility Library (GLU). This library uses only GL functions but contains code for creating common objects and simplifying viewing. All function in GLU can be created from the core GL library. The GLU library is available in all OpenGL implementations. Functions in the GLU library starts with the letters glu.

The third is the OpenGL Utility Toolkit (GLUT). It provides the minimum functionality that should be formulated in modern windowing systems.

**Figure 1.1 Basic block diagram of OpenGL**

**1.4 Problem Statement**

“Simulation to different sorting algorithm using OpenGL framework,graphic primitives, textures and keyboard/mouse interactions.”

**1.5 Objectives of the Project**

* To show the Simulation of different sorting algorithm which will help student to visualize how different sorting takes place
* To determine how much time each sorting algorithm takes to complete sorting.
* To show the implementation of the OpenGL transformation functions.
* To show the user and programme interaction using input devices.

**1.**6 **Organisation of the Project**

The project was organised in a systematic way. First we analysed what are the basic features to be included in the project to make it acceptable. As it is a graphics oriented project, we made the sketches prior so as to have an idea like how our output must look like. After all these, the source code was formulated as a paper work.As the project keep on growing we tried adding more features and functions into it.initially project showed simulation in only ascending order, after some time we included the function to simulate in either ascending or descending order.To make project more graphic oriented we added 3d view in the project. All the required software were downloaded. Finally, the successful implementation of the project.

**Chapter -2**

**SYSTEM SPECIFICATION**

**2.1 Hardware Requirements**

* Main Processor : PENTIUM III
* Processor Speed: 800 MHz
* RAM Size : 128 MB DDR
* Keyboard : Standard qwerty serial or PS/2 keyboard
* Mouse : Standard serial or PS/2 mouse
* Compatibility : AT/T Compatible
* Cache memory : 256 KB
* Diskette drive : 1,44MB,3.5 inches

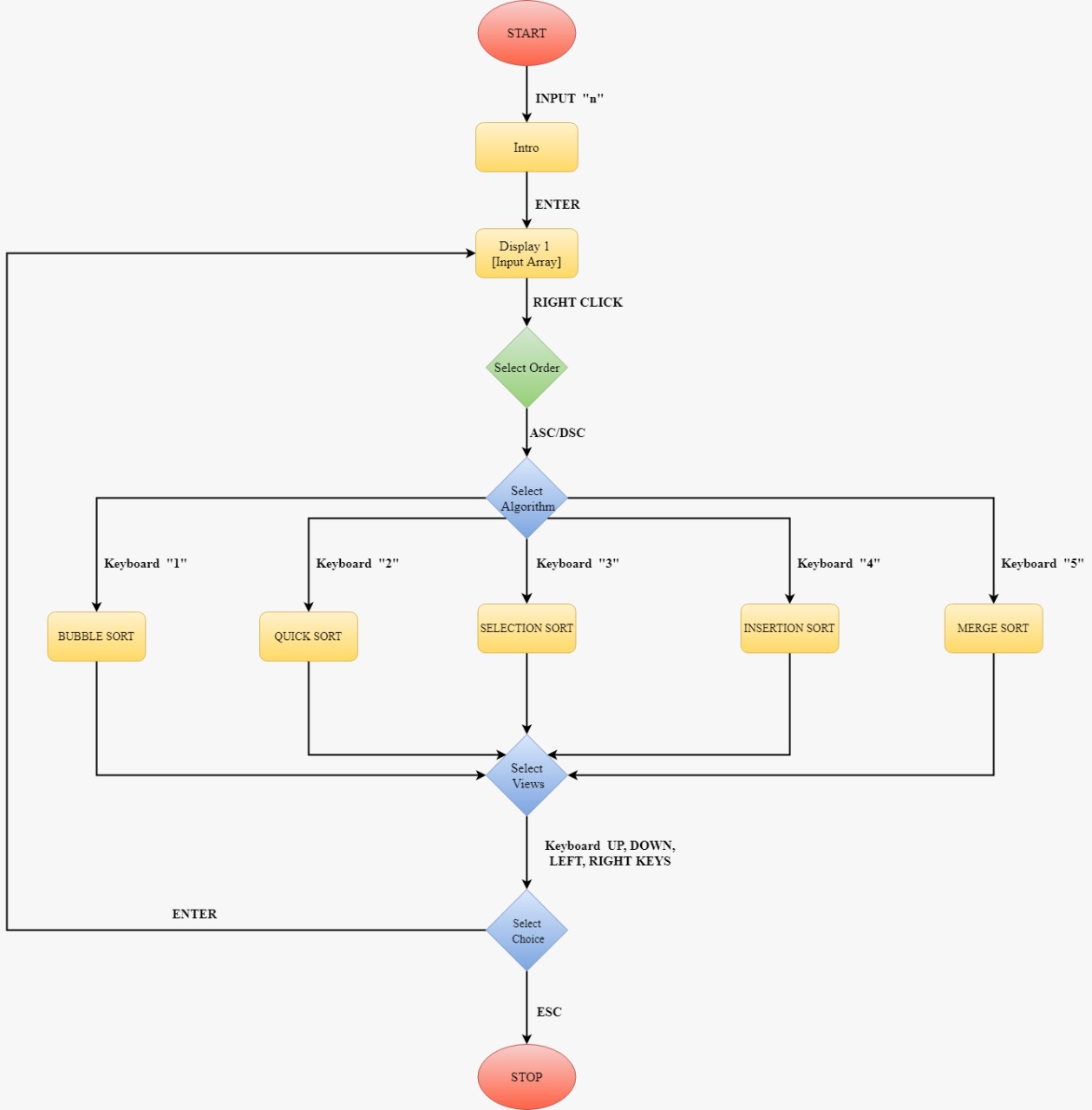
**2.2 Software Requirements**

* Operating System: Windows 10 or Linux (Fedora) or macOS
* Hypervisor used : Docker
* Compiler used : g++
* Language used : C++ language
* Editor : Visual Studio Code
* Toolkit : GLUT Toolkit

**Chapter -3**

**DESIGN**

**3.1 Flow Diagram**



**Figure 3.1 : Flow diagram of scene change**

**3.2 Description of Flow Diagram**

The description of the flow diagram is as follows:

**Step 1:** Start

**Step 2:** The user is presented with intro and presses ‘Enter’ to proceed to next scene

**Step 3:** User can select order of sorting by right clicking.

**Step 4:** The user then selects the type of algorithm for which simulation is to occur.

If user selects 1. Then bubble sort simulation occurs

If user selects 2. The quick sort simulation occurs.

If user selects 3. Then selection sort simulation occurs.

If user selects 4. Then insertion sort simulation occurs

If user selects 5. Then Merge sort simulation occurs..

**Step 5:**During the simulation of sorting algorithm the user can roate the arrangement to see different views.Up,Down.Left and Right button are pressed to change views..

**Step 6:** User then presses 0 to again randomize the array

**Step 7:**.It keeps looping and start again from step 3.

**Step 8:** Stop

**Chapter -4**

**IMPLEMENTATION**

**4.1 Built in Functions**

**1. glutInit()**

glutInit is used to initialize the GLUT library.  
Usage: void glutInit (int \*argc, char \*\*argv);  
Description: glutInit will initialize the GLUT library and negotiate a session with the window system.

**2. glutInitDisplayMode()**

glutInitDisplayMode sets the initial display mode.  
Usage: void glutInitDisplayMode (unsigned int mode);  
Mode-Display mode, normally the bitwise OR-ing GLUT display mode bit masks. Description: 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. glutCreateWindow()**

glutCreateWindow creates a top-level window.

Usage: intglutCreateWindow (char \*name); Name-ASCII character string for use as window name

Description: glutCreateWindow creates a top-level window. The name will be provided to the window system as the window’s name. The intent is that the window system will label the window with the name. Implicitly, the current window is set to the newly created window.

Each created window has a unique associated OpenGL context.

**4. glutDisplayFunc()**

glutDisplayFunc sets the display callback for the current window.  
Usage: void glutDisplayFunc (void(\*func)(void));  
Func: The new display callback function.  
Description: 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. Before the callback, the current window is set to the window needing to be redisplayed and the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback.

**5. glutMainLoop()**

glutMainLoop enters the GLUT event processing loop.

Usage: void glutMainLoop(void);

Description: 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 callbacks that have been registered.

**6. glMatrixMode()**

The two most important matrices are the model-view and projection matrix. At many times, the state includes values for both of these matrices, which are initially set to identity matrices. There is only a single set of functions that can be applied to any type of matrix. Select the matrix to which the operations apply by first set in the matrix mode, a variable that is set to one type of matrix and is also part of the state.

**7. glTranslate(GLfloat X, GLfloat Y, GLfloat Z)**

glTranslate produces a translation by x y z. If the matrix mode is either GL\_MODEL\_VIEW or GL\_PROJECTION, all objects drawn after a call to glTranslate are translated.

**8. glRotatef(GLdouble angle, GLdouble X, GLdouble Y, GLdouble Z)**

glRotatef produces a rotation of angle degrees around the vector x y z. If the matrix mode is either GL\_MODEL\_VIEW or GL\_PROJECTION, all objects drawn after glRotatef is called are rotated. Use glPushMatrix() and glPopmatrix() to save and restore the unrotated coordinate system.

**9. glPushMatrix()**

There is a stack of matrices for each of the matrix mode. In GL\_MODELVIEW mode, the stack depth is atleast 32. In other modes, GL\_COLOR, GL\_PROJECTION, and GL\_TEXTURE, the depth is atleast 2. The current matrix in any mode is the matrix on the top of the stack for that mode.

**10. glPopMatrix()**

glPopMatrix pops the current matrix stack, replacing the current matrix with the one below it on the stack. Initially, each of the stack contains one matrix, an identity matrix. It is an error to push a full matrix stack or pop a matrix stack that contains only a single matrix. In either case, the error flag is set and no other change is made to GL state.

**11. glutSwapBuffers()**

**Usage:** void glutSwapBuffers(void);

**Description:** Performs a buffer swap on the layer in use for the current window. Specifically, glutSwapBuffers promotes the contents of the front buffer. The contents of the back buffer then become undefined.

**12. glPointSize(GLfloat size)**

glPointSize specifies the rasterized diameter of points. This value will be used rasterize points. Otherwise, the value written to the shading language built-in variable gl-PointSize will be used. The point size specified by glPointSize is always returned when GL\_POINT\_SIZE is queried.

**13. glutKeyboardFunc()**

**Usage:** void glutKeyboardFunc(void(\*func)(unsigned char key, int x, int y)

**Func:** The new keyboard callback function

**Description:** 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.

**14****. glutCreateMenu(void (\*)(int)):**

Creates an empty menu. The argument is the callback function, which is of the form void myfunc(int value), where *value* holds the index of the menu item which was selected.

**15. glutAddMenuEntry(char \*name, int value):**

Adds a menu entry to the bottom of the current menu. The character string *name* is the text to be displayed in the menu entry, and the integer *value* is passed to your callback procedure to identify the selected item.

**16. glutAttachMenu(int button):**

Attaches the current menu to the specified mouse button, which is either GLUT\_LEFT\_BUTTON, GLUT\_MIDDLE\_BUTTON, or GLUT\_RIGHT\_BUTTON.

## 17.glutSpecialFunc()

  glutSpecialFunc sets the special keyboard callback for the current window.

**Usage**: void glutSpecialFunc(void (\*func)(int key, int x, int y));

**Func**: The new entry callback function.

**Description**: glutSpecialFunc sets the special keyboard callback for the current window. The special keyboard callback is triggered when keyboard function or directional keys are pressed. The key callback parameter is a GLUT\_KEY\_\* constant for the special key pressed. The x and y callback parameters indicate the mouse in window relative coordinates when the key was pressed. When a new window is created, no special callback is initially registered and special key strokes in the window are ignored. Passing NULL to glutSpecialFunc disables the generation of special callbacks.

**18.glLoadIdentity(void)**

glLoadIdentity replaces the current matrix with the identity matrix. It is semantically equivalent to calling glLoadMatrix with the identity matrix.

**19. void glutInitWindowPosition(int x, int y);**

glutInitWindowPosition and glutInitWindowSize set the initial window position and size respectively.

**Usage**:void glutInitWindowSize(int width, int height);

**Width:**Width in pixels.

**Height:**Height in pixels.

**X**:Window X location in pixels.

**Y**:Window Y location in pixels.

**Description**:Windows created by glutCreateWindow will be requested to be created with the current initial window position and size.

**4.2 User Defined Functions**

**1. void myinit()**

Used to set Matrix Mode and clipping coordinates and set background color.

**2. void randomize()**

Used to randomize the values present in array to be sorted.

**3. void cube()**

Used to create a Cuboid for each of the array element.Each cuboid has 6 faces, so internally cube() uses 6 polygon function to draw each face and each face has for vertex which are indicated by glvertex3i() function.

**4. void mergeSort(int arr[], int l, int r)**

This function is used to implement merge sort algorithm. Merge sort is based on divide and conquer .Here we divide the array to be sorted into 2 parts and then apply merge sort on those divided parts and after that we merge the two sorted array to get the final sorted array.

**5. void merge(int arr[], int l, int m, int r)**

This function is a part of merge sort algorithm .It is used to merge two sorted array to get final sorted array.Main heart of merge sort algorithm is merge procedure.

**6. void insertionSort (int arr[], int n)**

This function is used to implement insertion sort algorithm. To sort an array of size n in ascending order:   
1: Iterate from arr[1] to arr[n] over the array.   
2: Compare the current element (key) to its predecessor.   
3: If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element

**7. void selectionSort(int a[], int n)**

Used to implement selection sort algorithm. The selection sort algorithm sorts an array by repeatedly finding the minimum element (considering ascending order) from unsorted part and putting it at the beginning. The algorithm maintains two subarrays in a given array.  
1) The subarray which is already sorted.   
2) Remaining subarray which is unsorted

**8. void swap(int a[], int n, int low, int high)**

Used to swap two values that is sent to the function..

**9. void quicksort(int a[], int n, int low, int high)**

Used to implement quick sort algorithm. QuickSort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. There are many different versions of quickSort that pick pivot in different ways.

1. Always pick first element as pivot.
2. Always pick last element as pivot (implemented below)
3. Pick a random element as pivot.
4. Pick median as pivot.

**10. extern int partition(int a[], int n, int low, int high)**

The key process in quickSort is partition(). Target of partitions is, given an array and an element x of array as pivot, put x at its correct position in sorted array and put all smaller elements (smaller than x) before x, and put all greater elements (greater than x) after x.

**11. void delay(int d)**

Delay function is used to add delay time to the start time of a sorting algorithm.It act as a timer for sorting algorithms.

**12. void bsort(int a[], int n)**

Bsort function is used to implement bubble sort algorithm. Bubble Sort is the simplest sorting algorithm that works by repeatedly swapping the adjacent elements if they are in wrong order. In computer graphics it is popular for its capability to detect a very small error (like swap of just two elements) in almost-sorted arrays and fix it with just linear complexity (2n)

**13. void displayScene2()**

This function is used to display the front page of the project.It uses texture mapping to display the image on the screen .

**14. void loadBackground()**

This function is used to setup the background for the front screen to display.

**15. void display()**

This function is used to display the array list in form of rectangular columsn.Its the most important function of our project.Everytime the valuses in the array are changed either due to call to randomize() func or due to sorting algorithm then display function is called to redraw the array in form of rectangular columns.

**16. void drawtext(int x, int y, int z, char \*s)**

This function is used to draw the text onto the screen.

**17. void scale(int i)**

Function used to scale the rectangular column as per the value present in the array.

**18. void translate(int i)**

Function used find the postion for each of the array element in the displayed arrangement of colmns..

**19. void multiply(char x)**

Function used in translate() and scale() function to multiply the as per the index of the array elemnt and as per the value at that index.

**20. void keys(unsigned char key, int x, int y)**

Function used to change the value of type variable which determines the type of sorting algorithm to visalize.the value of type variable is changed as per the button pressed on keyboard.

**21. void SpecialInput(int key, int x, int y)**

Function used to rotate the arrangement of array elemnts which are represented in form of cuboids. Either in x direction or y direction depending on button pressed.

**22. void menu( int n)**

Function to display menu and process menu events.Used to change the value of order variable .ord variable determines the sorting order of the array either ascending or descending.

**4.3 Source code**

**main.c**

#include <GL/glut.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

#define STB\_IMAGE\_IMPLEMENTATION

#include"stb\_image.h"

unsigned int bg2;

void displayScene2();

void loadBackground();

GLint osq[3][4] = {{0, 1, 1, 0}, {0, 0, 1, 1}, {1, 1, 1, 1}};

GLint nsq[3][4] = {{0, 1, 1, 0}, {0, 0, 1, 1}, {1, 1, 1, 1}};

GLint tsq[3][4] = {{0, 1, 1, 0}, {0, 0, 1, 1}, {1, 1, 1, 1}};

GLint scalem[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

GLint transm[3][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

GLint \*a;

GLint n;

char type = 'r';

char order = 'a';

int font = 0;

static GLfloat theta[] = {0.0, 0.0, 0.0};

static GLint axis = 0;

clock\_t start, finish;

void display();

void multiply(char);

void scale(int);

void translate(int);

void randomize();

void cube();

void drawtext(int, int, int, char \*);

void bsort(int[], int);

void delay(int);

void quickSort(int[], int, int, int);

int partition(int[], int, int, int);

void msgtext(int, int, int, char \*);

void swap(int \*, int \*);

void displayScene2()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glEnable(GL\_TEXTURE\_2D);

glEnable(GL\_TEXTURE\_2D);

glColor3f(1, 1, 1);

glBindTexture(GL\_TEXTURE\_2D, bg2);

glBegin(GL\_QUADS);

glVertex3f(-0.30\*n, 0.825\*1.8\*n, 0);

glTexCoord2f(0, 0);

glVertex3f(1.280\*1.8\*n, 0.825\*1.8\*n, 0);

glTexCoord2f(0, 1);

glVertex3f(1.280\*1.8\*n,-0.5\*n ,0);

glTexCoord2f(1, 1);

glVertex3f(-0.30\*n, -0.5\*n, 0);

glTexCoord2f(1, 0);

glEnd();

glFlush();

glDisable(GL\_TEXTURE\_2D);

glutSwapBuffers();

}

void loadBackground(void)

{

glGenTextures(1, &bg2);

glBindTexture(GL\_TEXTURE\_2D, bg2);

glPixelStorei(GL\_UNPACK\_ALIGNMENT,1);

// set the bg1 wrapping/filtering options (on the currently bound bg1 object)

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T, GL\_REPEAT);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER, GL\_LINEAR);

glTexParameterf(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

// load and generate the bg1

int width, height, nrChannels;

unsigned char \*data = stbi\_load("glass1.jpg", &width, &height, &nrChannels, STBI\_rgb\_alpha);

if (data)

{

glTexImage2D(GL\_TEXTURE\_2D, 0, GL\_RGBA, width, height, 0, GL\_RGBA, GL\_UNSIGNED\_BYTE, data);

//glGenerateMipmap(GL\_TEXTURE\_2D);

}

else

{

}

stbi\_image\_free(data);

}

void display()

{

int i;

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

switch (type)

{

case 'r':

drawtext(0, n, 0, "Array to be sorted");

break;

case 'q':

drawtext(0, n, 0, "Quick Sort | Execution time in ms=");

break;

case 'b':

drawtext(0, n, 0, "Bubble Sort | Execution time in ms=");

break;

case 's':

drawtext(0, n, 0, "Selection Sort | Execution time in ms=");

break;

case 'i':

drawtext(0, n, 0, "Insertion Sort | Execution time in ms=");

break;

case 'm':

drawtext(0, n, 0, "Merge Sort | Execution time in ms=");

break;

}

for (i = 0; i < n; i++)

{

scale(a[i]);

translate(i);

nsq[0][0] = 0 + 2 \* i;

nsq[1][0] = 0;

nsq[0][1] = 1 + 2 \* i;

nsq[1][1] = 0;

cube();

}

glFlush();

glutSwapBuffers();

}

void drawtext(int x, int y, int z, char \*s)

{

int i, j;

char t[10];

char \*u = malloc(strlen(s) + 20);

for (i = 0; i < strlen(u); i++)

u[i] = ' ';

if (type != 'r')

{

sprintf(t, "%d", (int)((clock() - start) / 1000));

strcat(u, s);

strcat(u, t);

glColor3f(1.0, 1.0, 1.0);

glRasterPos3i(x, y, z);

for (i = 0; u[i] != '\0'; i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, u[i]);

}

else

{

glColor3f(1.0, 1.0, 1.0);

glRasterPos3i(x, y, z);

for (i = 0; s[i] != '\0'; i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, s[i]);

char m[150];

sprintf(m, "%s", " 1. Bubble | 2. Quick | 3. Selection | 4. Insertion | 5. Merge");

glRasterPos3i(x, y + 4, z);

for (i = 0; m[i] != '\0'; i++)

glutBitmapCharacter(GLUT\_BITMAP\_HELVETICA\_18, m[i]);

}

}

void bsort(int a[], int n)

{

int i, j, x, y, temp;

for (i = 0; i < n; i++)

for (j = i + 1; j < n; j++)

{

if (order == 'a')

{

if (a[i] > a[j])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

display();

}

}

if (order == 'd')

{

if (a[i] < a[j])

{

temp = a[i];

a[i] = a[j];

a[j] = temp;

display();

}

}

delay(2000);

}

}

void delay(int d)

{

clock\_t start = clock();

while (clock() < start + d\*10)

{

}

}

void quickSort(int a[], int n, int low, int high)

{

if (low < high)

{

int pi = partition(a, n, low, high);

quickSort(a, n, low, pi - 1);

quickSort(a, n, pi + 1, high);

}

}

extern int partition(int a[], int n, int low, int high)

{

int pivot = a[high];

int i = (low - 1), j;

for (j = low; j <= high - 1; j++)

{

if (order == 'a')

{

if (a[j] <= pivot)

{

i++;

swap(&a[i], &a[j]);

display();

delay(2000);

}

}

if (order == 'd')

{

if (a[j] >= pivot)

{

i++;

swap(&a[i], &a[j]);

display();

delay(2000);

}

}

}

swap(&a[i + 1], &a[high]);

display();

delay(3000);

return (i + 1);

}

void swap(int \*x, int \*y)

{

int temp = \*x;

\*x = \*y;

\*y = temp;

}

void selectionSort(int a[], int n)

{

int i, j, min\_idx;

for (i = 0; i < n - 1; i++)

{

min\_idx = i;

for (j = i + 1; j < n; j++)

{

if (order == 'a')

{

if (a[j] < a[min\_idx])

min\_idx = j;

}

else

{

if (a[j] > a[min\_idx])

min\_idx = j;

}

}

swap(&a[min\_idx], &a[i]);

delay(10000);

display();

}

}

void insertionSort(int arr[], int n)

{

int i, key, j;

for (i = 1; i < n; i++)

{

key = arr[i];

j = i - 1;

if (order == 'a')

{

while (j >= 0 && arr[j] > key)

{

arr[j + 1] = arr[j];

j = j - 1;

display();

delay(1000);

}

}

else

{

while (j >= 0 && arr[j] < key)

{

arr[j + 1] = arr[j];

j = j - 1;

display();

delay(1000);

}

}

arr[j + 1] = key;

}

}

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1 + j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (order == 'a')

{

if (L[i] <= R[j])

{

arr[k] = L[i];

display();

delay(1000);

i++;

}

else

{

arr[k] = R[j];

display();

delay(1000);

j++;

}

}

else

{

if (L[i] >= R[j])

{

arr[k] = L[i];

display();

delay(1000);

i++;

}

else

{

arr[k] = R[j];

display();

delay(1000);

j++;

}

}

k++;

}

while (i < n1)

{

arr[k] = L[i];

display();

delay(1000);

i++;

k++;

}

while (j < n2)

{

arr[k] = R[j];

display();

delay(1000);

j++;

k++;

}

}

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

int m = l + (r - l) / 2;

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

merge(arr, l, m, r);

}

}

void cube()

{

glColor3f(0.0, 0.8, 0.8);

glBegin(GL\_POLYGON);

glVertex3i(nsq[0][0], nsq[1][0], 0);

glVertex3i(nsq[0][1], nsq[1][1], 0);

glVertex3i(nsq[0][2], nsq[1][2], 0);

glVertex3i(nsq[0][3], nsq[1][3], 0);

glEnd();

glColor3f(0.0, 1.0, 0.0);

glBegin(GL\_POLYGON);

glVertex3i(nsq[0][0], nsq[1][0], 0);

glVertex3i(nsq[0][3], nsq[1][3], 0);

glVertex3i(nsq[0][3], nsq[1][3], 1);

glVertex3i(nsq[0][0], nsq[1][0], 1);

glEnd();

glColor3f(0.0, 0.0, 1.0);

glBegin(GL\_POLYGON);

glVertex3i(nsq[0][1], nsq[1][1], 0);

glVertex3i(nsq[0][2], nsq[1][2], 0);

glVertex3i(nsq[0][2], nsq[1][2], 1);

glVertex3i(nsq[0][1], nsq[1][1], 1);

glEnd();

glColor3f(1.0, 1.0, 0.0);

glBegin(GL\_POLYGON);

glVertex3i(nsq[0][0], nsq[1][0], 1);

glVertex3i(nsq[0][1], nsq[1][1], 1);

glVertex3i(nsq[0][2], nsq[1][2], 1);

glVertex3i(nsq[0][3], nsq[1][3], 1);

glEnd();

glColor3f(1.0, 0.0, 0.5);

glBegin(GL\_POLYGON);

glVertex3i(nsq[0][2], nsq[1][2], 0);

glVertex3i(nsq[0][3], nsq[1][3], 0);

glVertex3i(nsq[0][3], nsq[1][3], 1);

glVertex3i(nsq[0][2], nsq[1][2], 1);

glEnd();

glColor3f(1.0, 0.0, 1.0);

glBegin(GL\_POLYGON);

glVertex3i(nsq[0][0], nsq[1][0], 0);

glVertex3i(nsq[0][1], nsq[1][1], 0);

glVertex3i(nsq[0][1], nsq[1][1], 1);

glVertex3i(nsq[0][0], nsq[1][0], 1);

glEnd();

}

void multiply(char x)

{

int i, j, k;

if (x == 's')

{

for (i = 0; i < 3; i++)

{

for (j = 0; j < 4; j++)

{

nsq[i][j] = 0;

for (k = 0; k < 4; k++)

nsq[i][j] += scalem[i][k] \* osq[k][j];

}

}

for (i = 0; i < 3; i++)

for (j = 0; j < 4; j++)

tsq[i][j] = nsq[i][j];

}

else

{

for (i = 0; i < 3; i++)

{

for (j = 0; j < 4; j++)

{

nsq[i][j] = 0;

for (k = 0; k < 4; k++)

nsq[i][j] += transm[i][k] \* tsq[k][j];

}

}

}

}

void scale(int i)

{

scalem[1][1] = i + 1;

multiply('s');

}

void translate(int i)

{

transm[0][2] = 2 \* i;

multiply('t');

}

void keys(unsigned char key, int x, int y)

{

if (key == 13)

{

type = 'r';

randomize(a, n);

}

if (key == '0')

{

type = 'r';

randomize(a, n);

}

if (key == '1')

{

type = 'b';

start = clock();

bsort(a, n);

}

if (key == '2')

{

type = 'q';

start = clock();

quickSort(a, n, 0, n - 1);

}

if (key == '3')

{

type = 's';

start = clock();

selectionSort(a, n);

display();

}

if (key == '4')

{

type = 'i';

start = clock();

insertionSort(a, n);

display();

}

if (key == '5')

{

type = 'm';

start = clock();

mergeSort(a, 0, n);

display();

}

}

void SpecialInput(int key, int x, int y)

{

if (key == GLUT\_KEY\_UP)

{

axis = 1;

theta[axis] += 0.5;

if (theta[axis] >= 360.0)

theta[axis] -= 360.0;

glRotatef(theta[0], 1.0, 0.0, 0.0);

display();

}

if (key == GLUT\_KEY\_DOWN)

{

axis = 1;

theta[axis] += 0.5;

if (theta[axis] >= 360.0)

theta[axis] -= 360.0;

glRotatef(-theta[0], 1.0, 0.0, 0.0);

display();

}

if (key == GLUT\_KEY\_LEFT)

{

axis = 0;

theta[axis] += 0.3;

if (theta[axis] >= 360.0)

theta[axis] -= 360.0;

glRotatef(theta[1], 0.0, 1.0, 0.0);

display();

}

if (key == GLUT\_KEY\_RIGHT)

{

axis = 0;

theta[axis] += 0.3;

if (theta[axis] >= 360.0)

theta[axis] -= 360.0;

glRotatef(-theta[1], 0.0, 1.0, 0.0);

display();

}

}

void myReshape(int w, int h)

{

glViewport(0, 0, w, h);

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

if (w <= h)

glOrtho(-2.0, 2.0, -2.0 \* (GLfloat)h / (GLfloat)w, 2.0 \* (GLfloat)h / (GLfloat)w, -10.0, 10.0);

else

glOrtho(-2.0 \* (GLfloat)h / (GLfloat)w, 2.0 \* (GLfloat)h / (GLfloat)w, -2.0, 2.0, -10.0, 10.0);

glMatrixMode(GL\_MODELVIEW);

}

void randomize()

{

srand(time(NULL));

int i;

for (i = n - 1; i > 0; i--)

{

int j = rand() % (i + 1);

swap(&a[i], &a[j]);

}

display();

}

void myInit()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.58, 0.58, 0.58, 0.0);

glMatrixMode(GL\_MODELVIEW);

glLoadIdentity();

glOrtho(0, 2 \* n, 0, n, 0, n);

}

void menu(int n)

{

switch (n)

{

case 1:

order = 'a';

display();

break;

case 2:

order = 'd';

display();

break;

}

}

/\*

void timer(int value)

{

glutPostRedisplay();

}

\*/

int main(int argc, char \*\*argv)

{

printf("Enter the number of elements you'd like to see: ");

scanf("%d", &n);

int i;

a = malloc((n + 1) \* sizeof(int));

for (i = 0; i < n; i++)

a[i] = i;

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGBA | GLUT\_DEPTH);

glutInitWindowSize(1280, 720);

glutInitWindowPosition(0, 0);

glutCreateWindow("Sorting Algorithm");

glutDisplayFunc(displayScene2);

loadBackground();

//glutTimerFunc(500, timer, 0);

glutCreateMenu(menu);

glutAddMenuEntry("Ascending", 1);

glutAddMenuEntry("Descending", 2);

glutAttachMenu(GLUT\_RIGHT\_BUTTON);

myInit();

randomize();

glutReshapeFunc(myReshape);

glutKeyboardFunc(keys);

glutSpecialFunc(SpecialInput);

glEnable(GL\_DEPTH\_TEST);

glutMainLoop();

return 0;

//glutDisplayFunc(display);

//glutReshapeFunc(reshape);

//texture = LoadTexture();

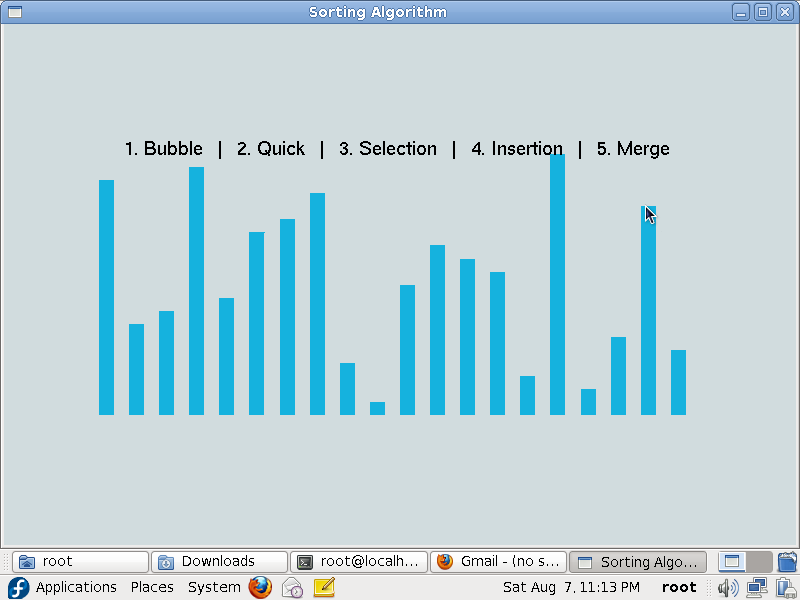
}

**Chapter -5**

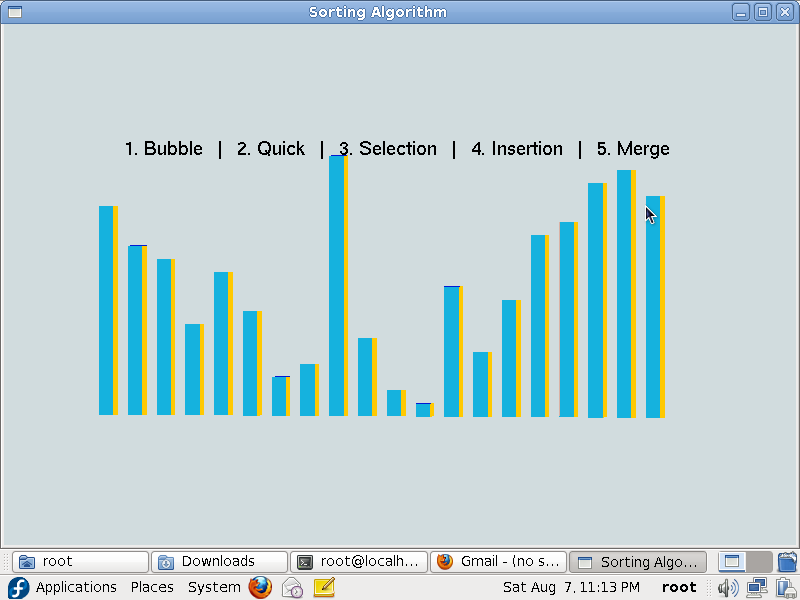
**SNAPSHOTS**



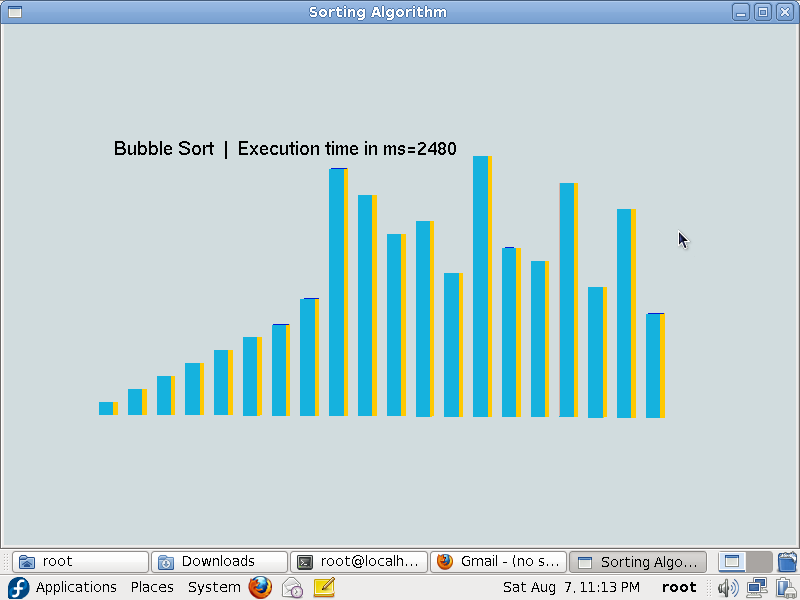
**Figure 5.1 : Intro scene**



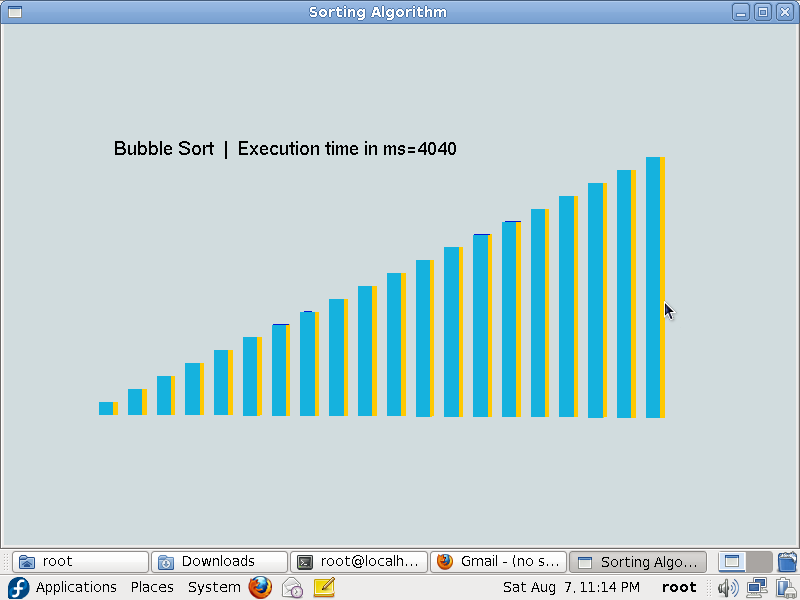
**Figure 5.2 : Randomized Array**



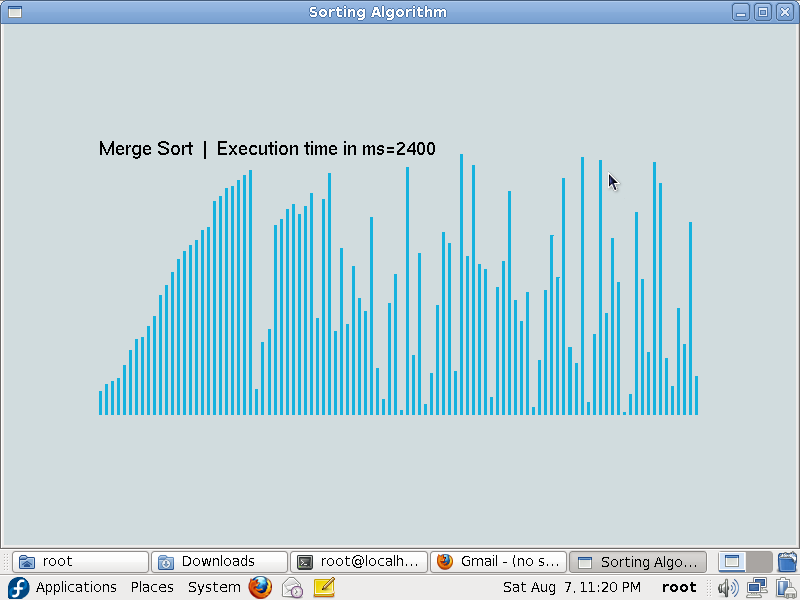
**Figure 5.3: Randomized array in 3d view**

****

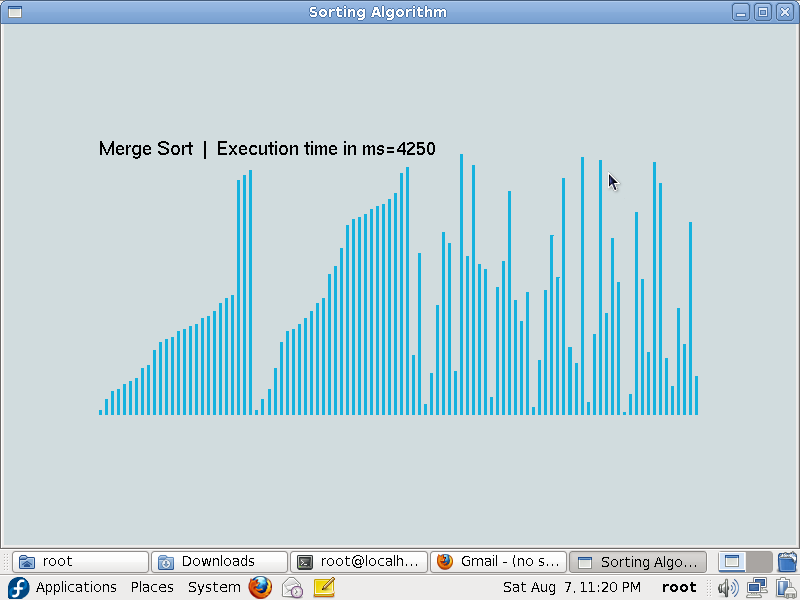
**Figure 5.4: Simulation of bubble sort**

****

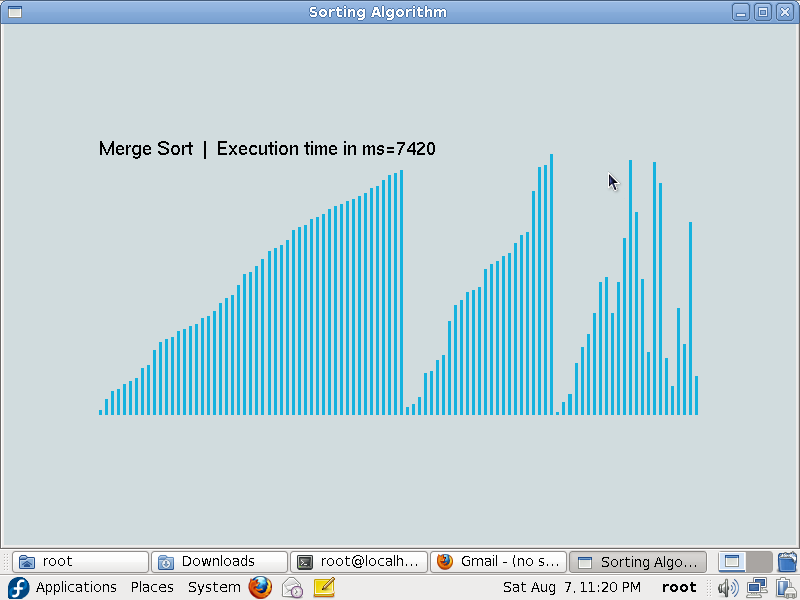
**Figure 5.5: Sorted Array**

****

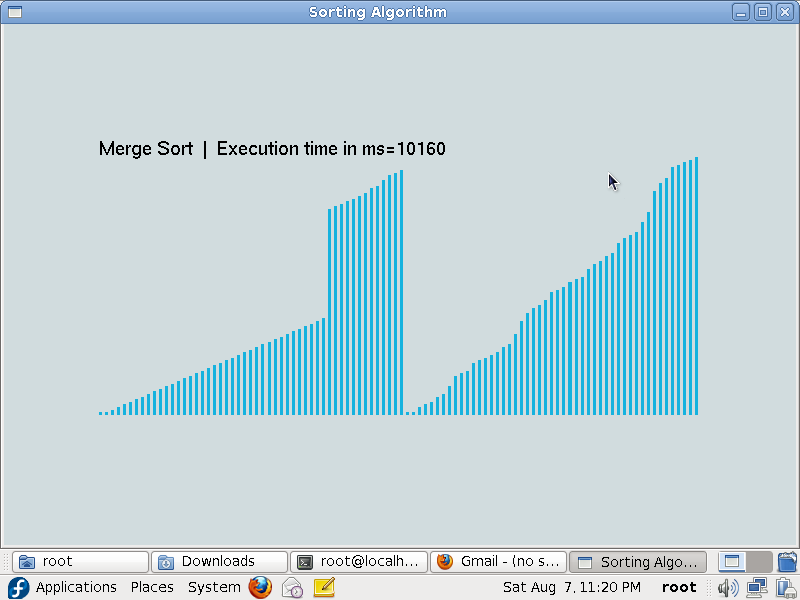
**Figure 5.6: Merge sort of array of 100 elemnts**

****

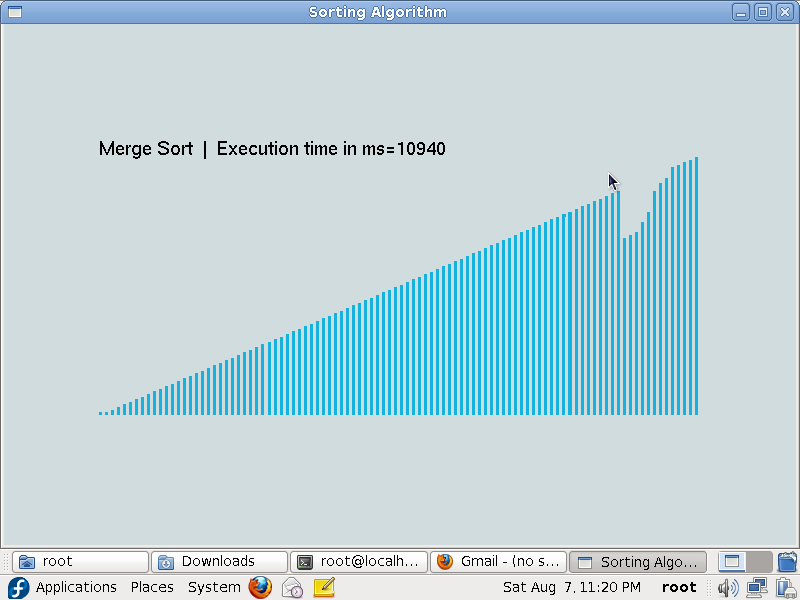
**Figure 5.7: merge sort 2**

****

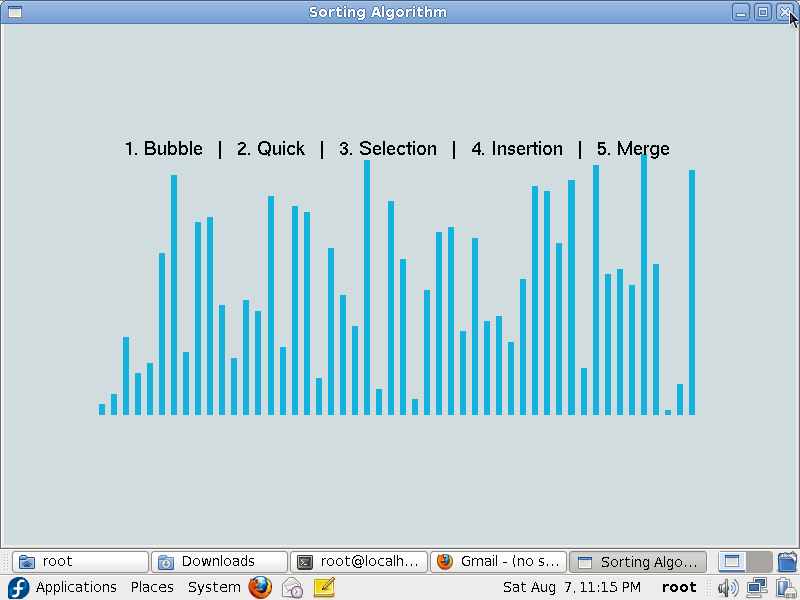
**Figure 5.8: Merge sort 3**

****

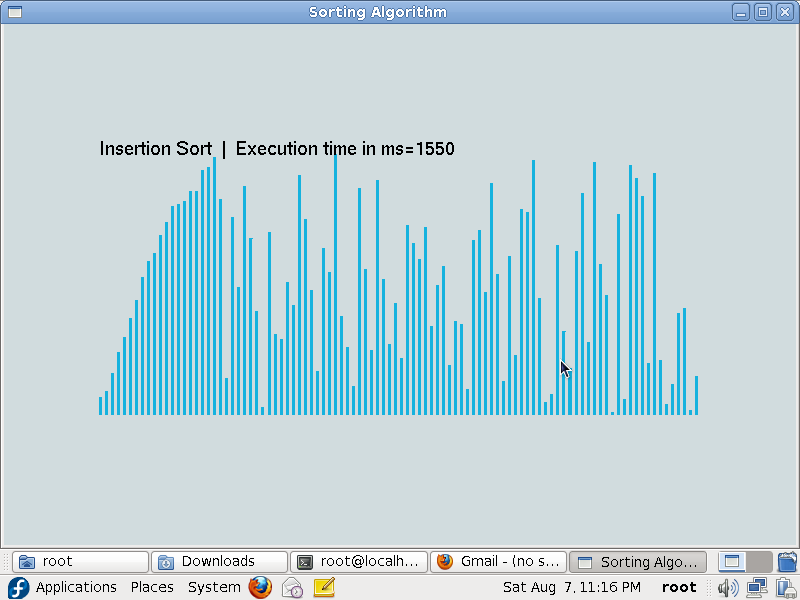
**Figure 5.9: merge sort 4**

****

**Figure 5.10: Merge sort 5**

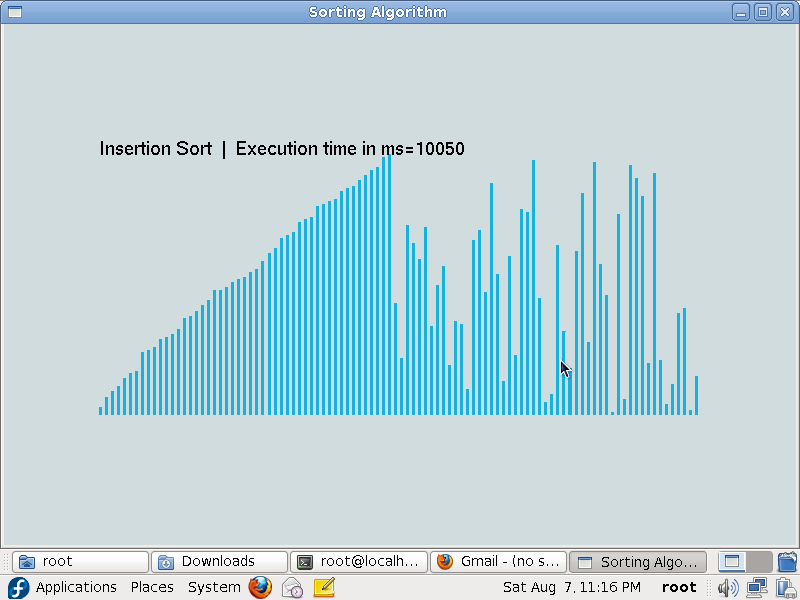
****

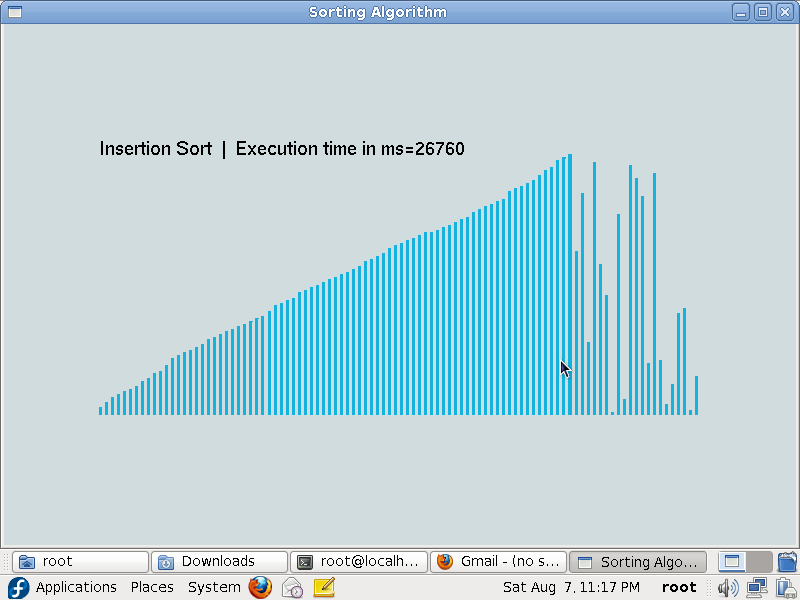
**Figure 5.11: randomized array of 100 elements**

****

**Figure 5.12: Moral Scene**

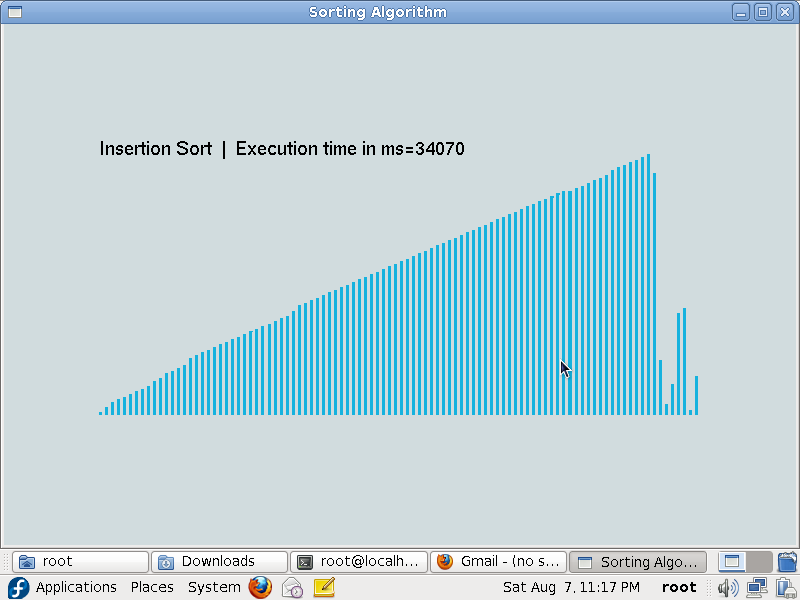
**Figure 5.12: Insertion sort1**

****

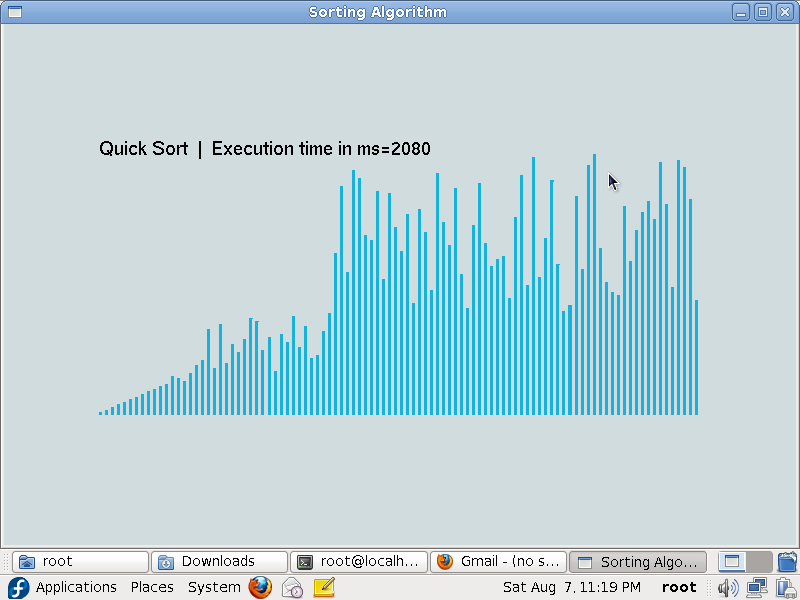
****

**Figure 5.13: Insertion sort 2**

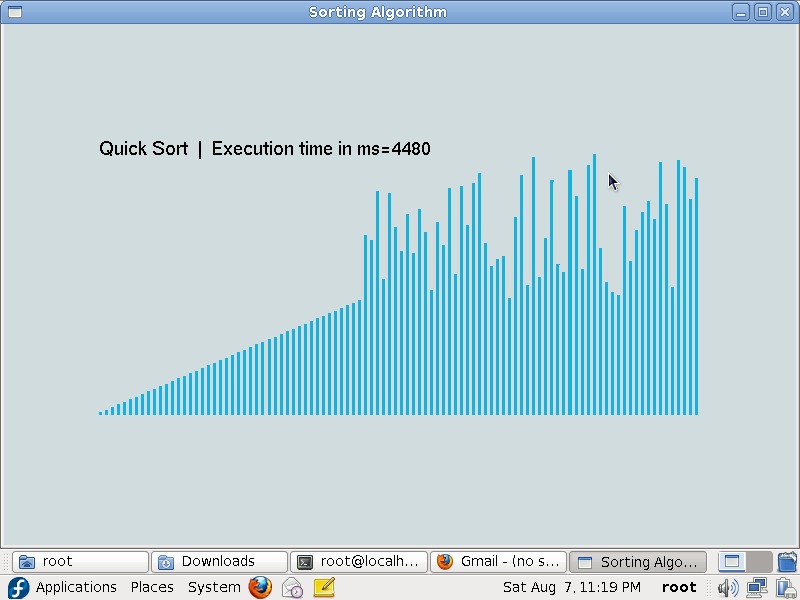
**Figure 5.14: Insertion sort 3**

****

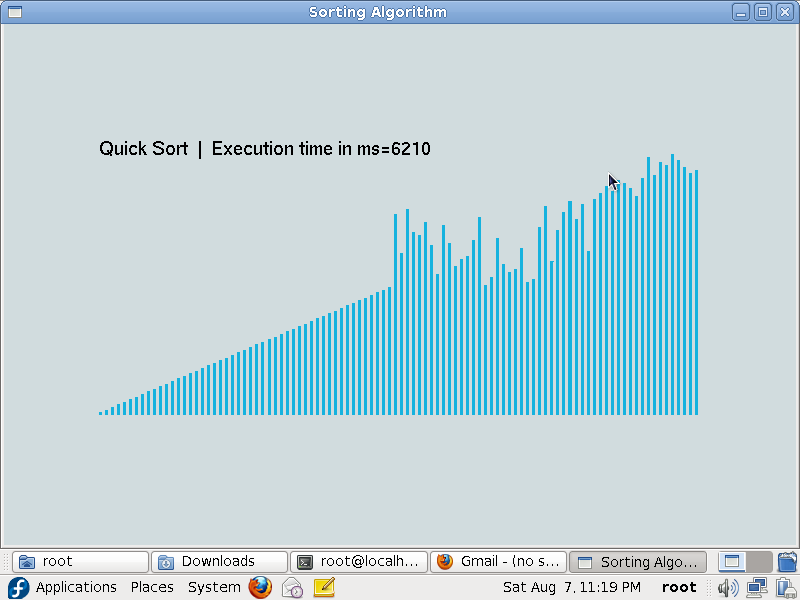
**Figure 5.15: Insertion sort 4**

****

**Figure 5.16: Quick sort 1**

****

**Figure 5.17: Quick sort 2**

****

**Figure 5.18: Quick sort 3**

**Chapter -6**

**CONCLUSION**

“Simulation of Sorting algorithm” project was done in a perspective of understanding the openGL software toolkit and to visualize how different sorting algorithms work. Through this project , we have acquired a much deeper knowledge of the openGL constraints and computer graphics as a whole.

We thus would like to emphasize the importance of this project to many other perspectives of Technical, mathematical, graphical and software concepts which we were unaware of.

**6.1 Future Enhancements**

* In future the same project can be enhanced in such a way that we can interact more with the project. Also the project can be implemented in 3D Space.
* A vast amount of future work can be possible by following investigations and strategies.
* More features can be included and can be modified in a more versatile way.

**BIBLIOGRAPHY**

**Reference Books**

[1] Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version, 3rd/4th Edition, Pearson Education, 2011

[2] Edward Angel: Interactive Computer Graphics- A Top Down approach with OpenGL, 5th Edition, Pearson Education

[3] James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer Graphics with OpenGL, Pearson Education

[4] Macro Cantu: Mastering Delphi

**Websites**

[1] https://www.opengl.org/  
[2] https://learnopengl.com/