Computer Graphics (UCS505)

Project on

Sorting Algorithm Visualizer

Submitted By

Sudansh Rana	102217005
Saksham Dhiman	102217026
Avneesh Jarangal	102217029

3CS1

B.E. Third Year – COE

Submitted To:

Ms. Kudart Aulakh

Computer Science and Engineering Department

Thapar Institute of Engineering and Technology

Patiala – 147001

Table of Contents

1. Introduction To Project

1. Introduction to Project

This project visualizes various sorting algorithms using the OpenGL library. The goal is to help understand how sorting algorithms operate internally by showing real-time graphical transformations of data elements represented as vertical bars.

The program executes the following:

- Initializes the OpenGL graphics window.
- Generates an array of random numbers represented as bars of variable heights.
- Executes different sorting algorithms such as Quick Sort, Merge Sort, Selection Sort, and Bubble Sort.
- Visually shows the comparisons and swaps by updating the bars in real time.
- Displays the name of the algorithm currently running and the number of comparisons made.

2. Computer Graphics Concepts Used

- OpenGL: Used for rendering 2D graphics and drawing primitive shapes like rectangles.
- Transformation: Applied to dynamically update bar positions and heights during sorting.
- Modelling: Bars are modelled as rectangles with variable heights.
- Coloring: Different RGB colors are used to distinguish the text and bar visuals.
- Rendering: glFlush() is used to update the visual output immediately.
- Text Display: Uses rasterized bitmap fonts for displaying algorithm names and comparisons.

3. User Defined Functions

- **init():** Sets the background color and initializes the OpenGL environment.
- **displayText(x, y, text):** Displays a string at the specified screen coordinates.
- **drawbar(arr):** Clears the screen and redraws all bars based on the current array, updating in real-time.
- **Quicksort::**quicksort() / partition() / findpivot(): Implements quick sort and includes steps to count comparisons and show real-time updates.
- **Selection Sort: selectionsort():** Performs selection sort with comparison count tracking.
- **Bubble Sort: bubblesort():** Implements bubble sort with step-by-step visualization.
- **Merge Sort: mergesort() / merge():** Implements merge sort and visualizes merging steps.
- **display():** Calls each sorting algorithm one after another on the same dataset with delays in between.

2. Code

```
#include <GL/glut.h>
#include <windows.h>
#include <iostream>
#include <vector>
#include <string>
#include <ctime>

#define width 500
#define height 500

using namespace std;

string comparisontext = "No. of Comparisons";
string sort_name;
int comparison_count;
vector<int> arr(50); // Reduced from 500 to 50 for faster visualization

// Function for displaying text
void displayText(float x, float y, string stringToDisplay)
{
    glRasterPos2f(x, y);
    for (char c : stringToDisplay)
    {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, c);
    }
}

// Display Initialization
void init()
{
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
}

// Function to draw bars
void drawbar(vector<int> arr)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0, 0.0, 0.0);
    displayText(190, 470, sort_name);
    displayText(150, 430, "No. of Comparisons = " + to_string(comparison_count));
    glColor3f(1.0, 0.0, 0.0);
    int barWidth = width / arr.size();
    int x = 0;
    for (int i = 0; i < arr.size(); i++)
    {
        glRecti(x, 0, x + barWidth - 1, arr[i]);
        x += barWidth;
    }
    glFlush();
    Sleep(10); // Reduced delay for smoother animation
}

// Quicksort structure
struct Quicksort
{
    int findpivot(vector<int>&, int, int, int);
    int partition(vector<int>&, int, int);
    void quicksort(vector<int>&, int, int);
};
```

```
int Quicksort::findpivot(vector<int>& sample, int left, int mid, int right)
{
    comparison_count++;
    drawbar(sample);
    if ((sample[left] >= sample[mid] && sample[left] <= sample[right]) ||
        (sample[left] <= sample[mid] && sample[left] >= sample[right]))
        return left;

    comparison_count++;
    drawbar(sample);
    if ((sample[mid] >= sample[left] && sample[mid] <= sample[right]) ||
        (sample[mid] <= sample[left] && sample[mid] >= sample[right]))
        return mid;

    comparison_count++;
    drawbar(sample);
    return right;
}

int Quicksort::partition(vector<int>& sample, int left, int right)
{
    int p = findpivot(sample, left, (left + right) / 2, right);
    swap(sample[left], sample[p]);
    drawbar(sample);
    p = sample[left];
    int i = left + 1;
    int j = left + 1;
    for (; i <= right; i++)
    {
        comparison_count++;
        if (sample[i] < p)
        {
            swap(sample[i], sample[j]);
            j++;
        }
        drawbar(sample);
    }
    swap(sample[left], sample[j - 1]);
    drawbar(sample);
    return j - 1;
}

void Quicksort::quicksort(vector<int>& sample, int left, int right)
{
    if (right > left)
    {
        int p = partition(sample, left, right);
        quicksort(sample, left, p - 1);
        quicksort(sample, p + 1, right);
    }
}

// Selection Sort
struct SelectionSort
{
    void selectionsort(vector<int>&);
};

void SelectionSort::selectionsort(vector<int>& sample)
{
    drawbar(sample);
```

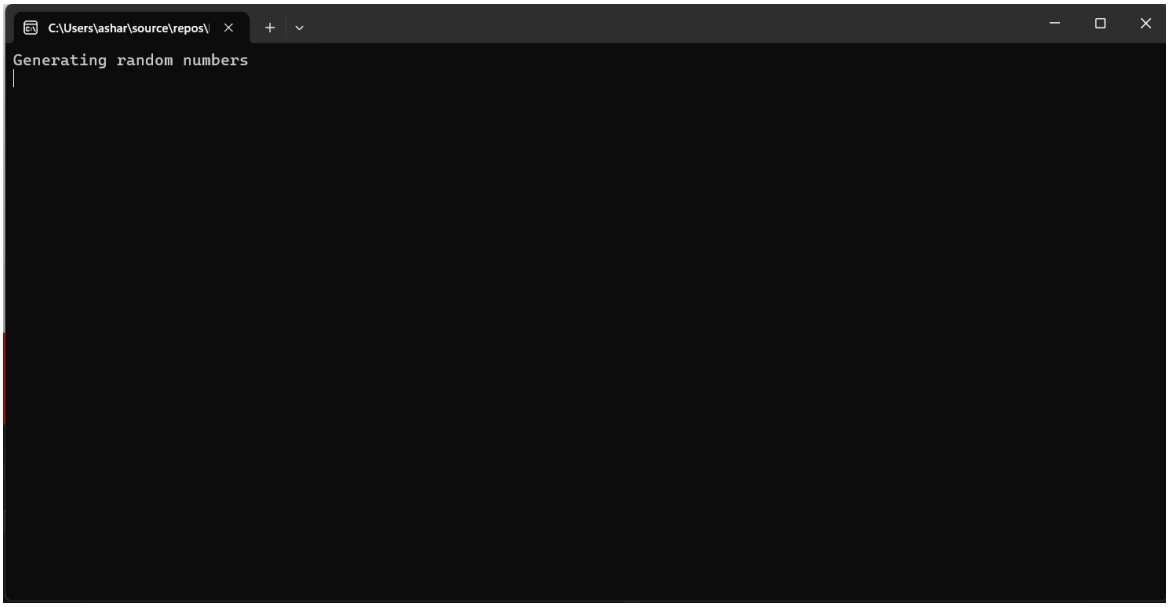
```
    int n = sample.size();
    for (int i = 0; i < n - 1; i++)
    {
        int min_idx = i;
        for (int j = i + 1; j < n; j++)
        {
            if (sample[j] < sample[min_idx])
                min_idx = j;
            comparison_count++;
            drawbar(sample);
        }
        swap(sample[min_idx], sample[i]);
        drawbar(sample);
    }
    drawbar(sample);
}

// Bubble Sort
struct BubbleSort
{
    void bubblesort(vector<int>&);
};

void BubbleSort::bubblesort(vector<int>& sample)
{
    drawbar(sample);
    int n = sample.size();
    for (int i = 0; i < n - 1; i++)
    {
        bool swapped = false;
        for (int j = 0; j < n - i - 1; j++)
        {
            if (sample[j] > sample[j + 1])
            {
                swap(sample[j], sample[j + 1]);
                swapped = true;
            }
            comparison_count++;
            drawbar(sample);
        }
        if (!swapped)
            break;
    }
    drawbar(sample);
}

// Merge Sort
struct MergeSort
{
    void merge(vector<int>&, int, int, int);
    void mergesort(vector<int>&, int, int);
};

void MergeSort::merge(vector<int>& sample, int l, int m, int r)
{
    drawbar(sample);
    int n1 = m - l + 1;
    int n2 = r - m;

    vector<int> l_sample(n1), r_sample(n2);

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

```
        l_sample[i] = sample[l + i];
    for (int j = 0; j < n2; j++)
        r_sample[j] = sample[m + 1 + j];

    int i = 0, j = 0, k = l;
    while (i < n1 && j < n2)
    {
        if (l_sample[i] <= r_sample[j])
        {
            sample[k++] = l_sample[i++];
        }
        else
        {
            sample[k++] = r_sample[j++];
        }
        comparison_count++;
        drawbar(sample);
    }

    while (i < n1)
    {
        sample[k++] = l_sample[i++];
        comparison_count++;
        drawbar(sample);
    }

    while (j < n2)
    {
        sample[k++] = r_sample[j++];
        comparison_count++;
        drawbar(sample);
    }
    drawbar(sample);
}

void MergeSort::mergesort(vector<int>& sample, int l, int r)
{
    if (l < r)
    {
        int m = l + (r - l) / 2;
        mergesort(sample, l, m);
        mergesort(sample, m + 1, r);
        merge(sample, l, m, r);
    }
}

// Display function
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    vector<int> temp;

    temp = arr;
    Quicksort my_qsort;
    comparison_count = 0;
    sort_name = "Quick Sort";
    my_qsort.quicksort(temp, 0, temp.size() - 1);
    Sleep(2000);

    temp = arr;
    SelectionSort my_selsort;
    comparison_count = 0;
```

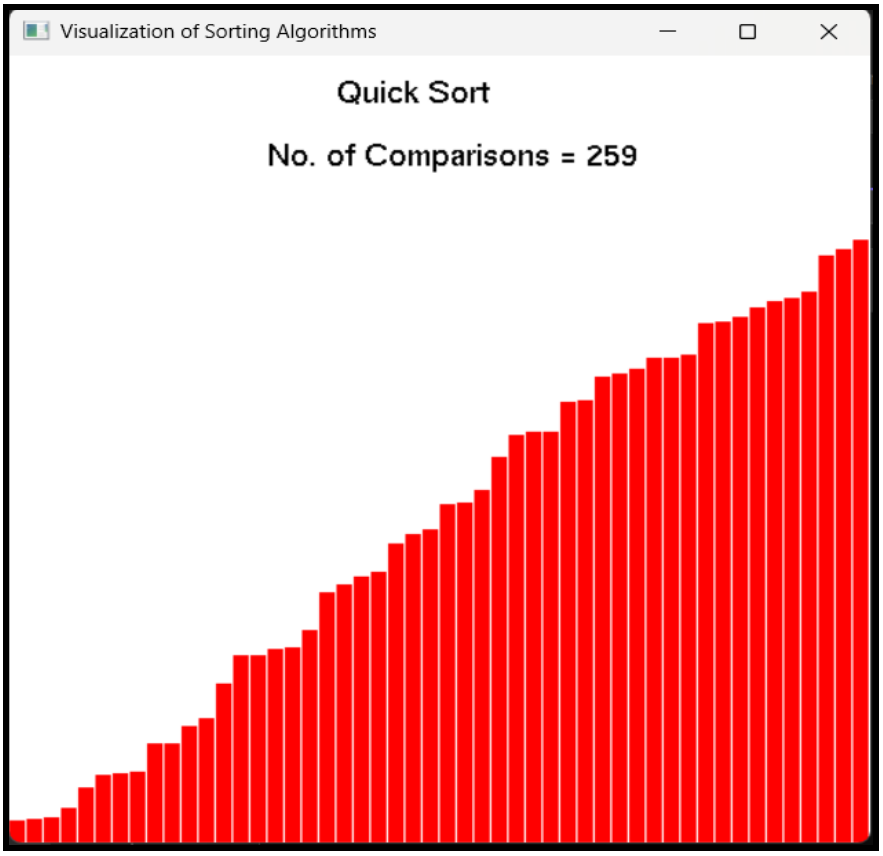
```
    sort_name = "Selection Sort";
    my_selsort.selectionsort(temp);
    Sleep(2000);

    temp = arr;
    BubbleSort my_bubsort;
    comparison_count = 0;
    sort_name = "Bubble Sort";
    my_bubsort.bubblesort(temp);
    Sleep(2000);

    temp = arr;
    MergeSort my_mergesort;
    comparison_count = 0;
    sort_name = "Merge Sort";
    my_mergesort.mergesort(temp, 0, temp.size() - 1);
    Sleep(2000);
}

// Main Function
int main(int argc, char** argv)
{
    srand(time(NULL));
    cout << "Generating random numbers\n";

    for (int i = 0; i < arr.size(); i++)
        arr[i] = rand() % 400;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
    glutInitWindowSize(width, height);
    glutCreateWindow("Visualization of Sorting Algorithms");
    glutDisplayFunc(display);
    init();
    gluOrtho2D(0, width, 0, height);
    glutMainLoop();
    return 0;
}
```

Output/Screenshots

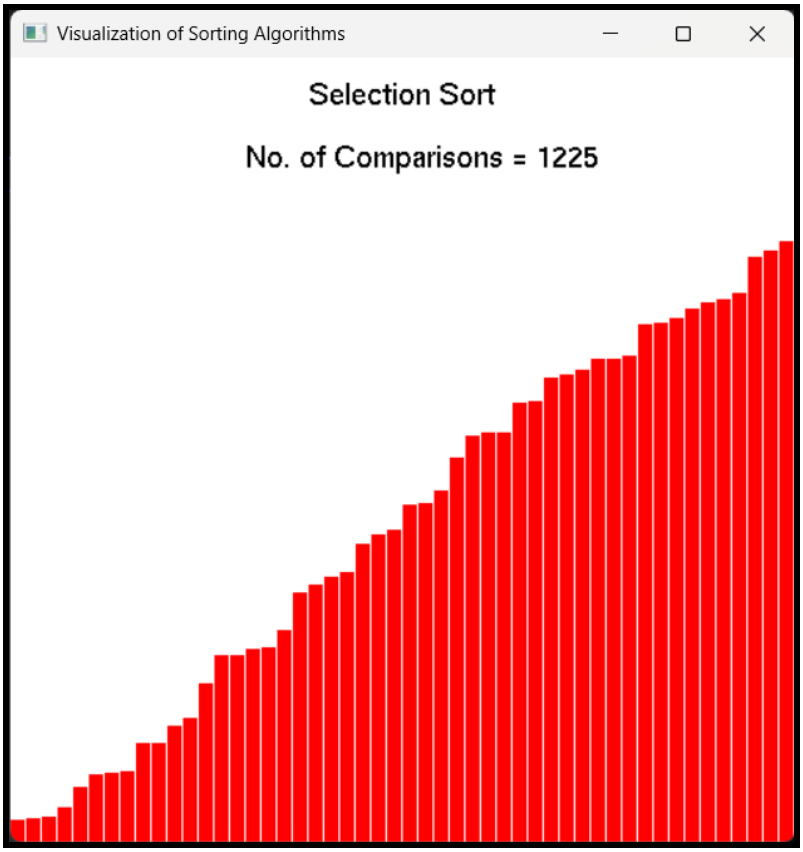
Generates Random Numbers:



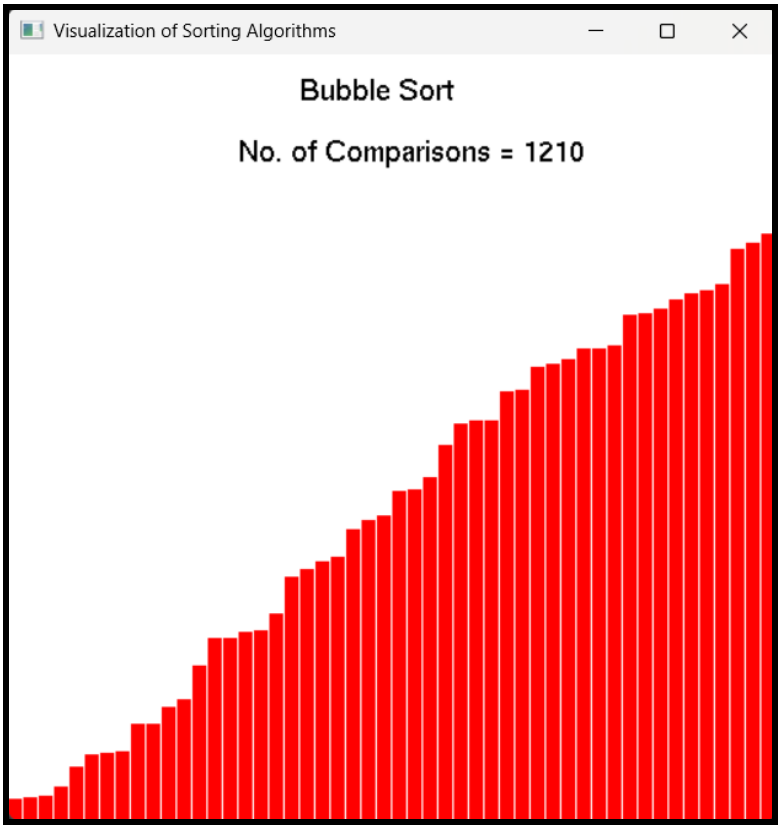
Quick Sort:



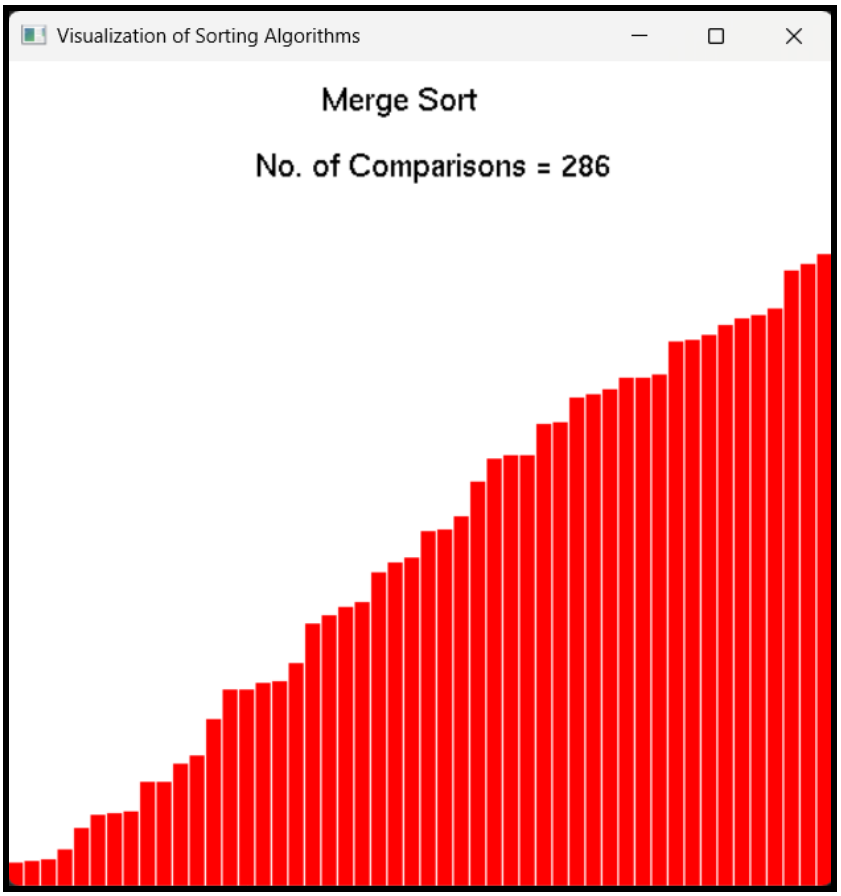
Selection Sort :


Visualization of Sorting Algorithms
Selection Sort
No. of Comparisons = 1225

Bubble Sort:


Visualization of Sorting Algorithms
Bubble Sort
No. of Comparisons = 1210

Merge Sort:


Visualization of Sorting Algorithms
Merge Sort
No. of Comparisons = 286