COMPUTER GRAPHICS PROJECT

DELHI TECHNOLOGICAL UNIVERSITY

PROJECT NAME:
SORTING ALGORITHM VISUALIZER

NAME:R.Prashant
Rollnumber:2k18/EC/124
SECTION:B3(G1)

Submitted To:Ms.Chingmuankim Naulak

INDEX

Introduction

Sorting is any process of arranging items systematically, and has two common, yet distinct meanings:

1. ordering: arranging items in a sequence ordered by some criterion;
2. categorizing: grouping items with similar properties.

In computer science, arranging in an ordered sequence is called "sorting". Sorting is a common operation in many applications, and efficient algorithms to perform it have been developed.

The most common uses of sorted sequences are:

- making lookup or search efficient;
- making merging of sequences efficient.
- enable processing of data in a defined order.

The opposite of sorting, rearranging a sequence of items in a random or meaningless order, is called shuffling.

For sorting, either a weak order, "should not come after", can be specified, or a strict weak order, "should come before" (specifying one defines also the other, the two are the complement of the inverse of each other, see operations on binary relations). For the sorting to be unique, these two are restricted to a total order and a strict total order, respectively.

Sorting n-tuples (depending on context also called e.g. records consisting of fields) can be done based on one or more of its components. More generally objects can be sorted based on a property. Such a component or property is called a sort key.

For example, the items are books, the sort key is the title, subject or author, and the order is alphabetical.

A new sort key can be created from two or more sort keys by lexicographical order. The first is then called the primary sort key, the second the secondary sort key, etc.

For example, addresses could be sorted using the city as primary sort key, and the street as secondary sort key.

If the sort key values are totally ordered, the sort key defines a weak order of the items: items with the same sort key are equivalent with respect to sorting. See also stable sorting. If different items have different sort key values then this defines a unique order of the items.

This project tries to explore the world of sorting and the various algorithms used with the help of a sorting visualizer which has been implemented on C++ using Open Glut and Open GL.

There are different types of sorting algorithms.

The project tries to explore the following sorting algorithms:

- **Bubble Sort:**

The **bubble sort** is also known as the *ripple* sort. The bubble sort is probably the first, reasonably complex module that any beginning programmer has to write. It is a very simple construct which introduces the student to the fundamentals of how sorting works.

A bubble sort makes use of an array and some sort of "swapping" mechanism. Most programming languages have a built-in function to swap elements of an array. Even if a swapping function does not exist, only a couple of extra lines of code are required to store one array element in a temporary field in order to swap a second element into its place. Then the first element is moved out of the temporary field and back into the array at the second element's position.

Here is a simple example of how a bubble sort works: Suppose you have a row of children's toy blocks with letters on them. They are in random order and you wish to arrange them in alphabetical order from left to right.

Step 1. Begin with the first block. In this case, the letter **G**. (Fig. 1.)

Fig. 1

Step 2. Look at the block just to the right of it.

Step 3. If the block to the right should come before the block on the left, swap them so that they are in order (Fig. 2.)

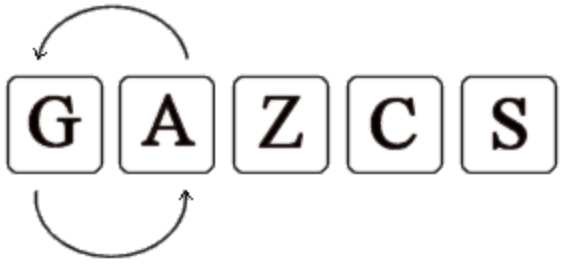


Fig. 2

If you were doing this by hand, you might just pick the blocks to be moved with one in each hand and cross your arms to swap them. Or you might move the first one out of its position temporarily, move the second one in its place, then move the first one to the now empty position (this is the difference between having a single function to do the swap, or writing some code to do it).

Step 4. Compare the next block in line with the first, and repeat step 3. Do this until you run out of blocks. Then begin step one again with the second block. (Fig. 3,4,5,6)

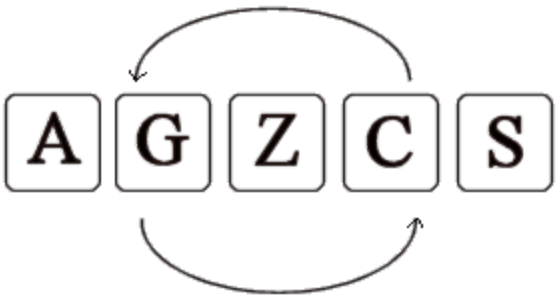


Fig. 3 - Pass #2

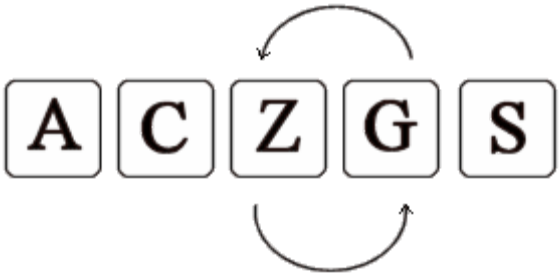


Fig. 4 - Pass #3

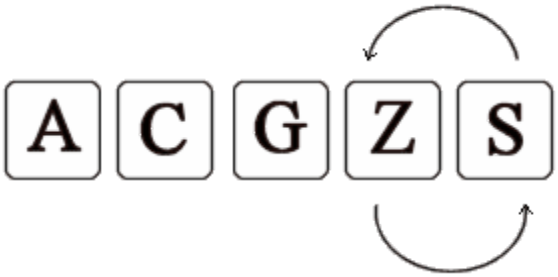


Fig. 5 - Pass #4

Fig. 6 - Completed Sort

- **Selection Sort**

In computer science, **selection sort** is an in-place comparison sorting algorithm. It has an O(n^2) time complexity, which makes it inefficient on large lists, and generally performs worse than the similar insertion sort. Selection sort is noted for its simplicity and has performance advantages over more complicated algorithms in certain situations, particularly where auxiliary memory is limited.

The algorithm divides the input list into two parts: a sorted sublist of items which is built up from left to right at the front (left) of the list and a sublist of the remaining unsorted items that occupy the rest of the list. Initially, the sorted sublist is empty and the unsorted sublist is the entire input list. The algorithm proceeds by finding the smallest (or largest, depending on sorting order) element in the unsorted sublist, exchanging (swapping) it with the leftmost unsorted element (putting it in sorted order), and moving the sublist boundaries one element to the right.

The time efficiency of selection sort is quadratic, so there are a number of sorting techniques which have better time complexity than selection sort. One thing which distinguishes selection sort from other sorting algorithms is that it makes the minimum possible number of swaps, $n - 1$ in the worst case.

Here is an example of this sort algorithm sorting five elements:

Sorted sublist	Unsorted sublist	Least element in unsorted list

()	(11, 25, 12, 22, 64)	11
(11)	(25, 12, 22, 64)	12
(11, 12)	(25, 22, 64)	22
(11, 12, 22)	(25, 64)	25
(11, 12, 22, 25)	(64)	64
(11, 12, 22, 25, 64)	()	

- **Insertion Sort:**

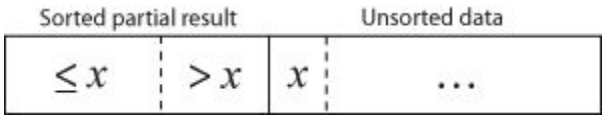
Insertion sort is a simple sorting algorithm that builds the final sorted array (or list) one item at a time. It is much less efficient on large lists than more advanced algorithms such as quicksort, heapsort, or merge sort. However, insertion sort provides several advantages:

- Simple implementation: Jon Bentley shows a three-line C version, and a five-line optimized version[1]
- Efficient for (quite) small data sets, much like other quadratic sorting algorithms
- More efficient in practice than most other simple quadratic (i.e., O(n^2)) algorithms such as selection sort or bubble sort
- Adaptive, i.e., efficient for data sets that are already substantially sorted: the time complexity is O(kn) when each element in the input is no more than k places away from its sorted position
- Stable; i.e., does not change the relative order of elements with equal keys
- In-place; i.e., only requires a constant amount O(1) of additional memory space
- Online; i.e., can sort a list as it receives it

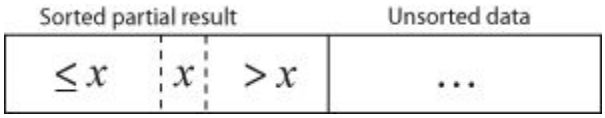
Insertion sort iterates, consuming one input element each repetition, and grows a sorted output list. At each iteration, insertion sort removes one element from the input data, finds the location it belongs within the sorted list, and inserts it there. It repeats until no input elements remain.

Sorting is typically done in-place, by iterating up the array, growing the sorted list behind it. At each array-position, it checks the value there against the largest value in the sorted list (which happens to be next to it, in the previous array-position checked). If larger, it leaves the element in place and moves to the next. If smaller, it finds the correct position within the sorted list, shifts all the larger values up to make a space, and inserts into that correct position.

The resulting array after k iterations has the property where the first $k + 1$ entries are sorted ("+1" because the first entry is skipped). In each iteration the first remaining entry of the input is removed, and inserted into the result at the correct position, thus extending the result:



becomes



with each element greater than x copied to the right as it is compared against x.

The most common variant of insertion sort, which operates on arrays, can be described as follows:

1. Suppose there exists a function called *Insert* designed to insert a value into a sorted sequence at the beginning of an array. It operates by beginning at the end of the sequence and shifting each element one place to the right until a suitable position is found for the new element. The function has the side effect of overwriting the value stored immediately after the sorted sequence in the array.
2. To perform an insertion sort, begin at the left-most element of the array and invoke *Insert* to insert each element encountered into its correct position. The ordered sequence into which the element is inserted is stored at the beginning of the array in the set of indices already examined. Each insertion overwrites a single value: the value being inserted.

Technology:

The project has implemented the visualization of the above three sorting algorithms .This has been made possible due to the presence of open source libraries in C++ namely OPENGL and Glut.

This project has been implemented in C++ with the help of the libraries like OpenGL and GLUT.Since it is difficult for students to visualize the sorting algorithms while studying about them this project tries to exploit the shortcomings by implementing the sorting visualizer in C++ which helps to visualize these sorting algorithms easily.

In short:

Technology used:

1.C++

2.OPENGL

3.GLUT

Methodology

Whenever we study about any project it becomes imperative to know about its methodology.Hence this section tries to explain the complete workflow of the project .

The code has been divided into various sections with the help of suitable functions.
1.GLUTINIT Function:

The glutinit function uses two arguments.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.

2.GlutinitDisplayMode Function:
glutInitDisplayMode sets the *initial display mode*.
Specifically talking glut single or glut rgb is used
1)GLUT_RGB
An alias for GLUT_RGBA.

2)GLUT_SINGLE
Bit mask to select a single buffered window. This is the default if neither GLUT_DOUBLE or GLUT_SINGLE are specified.

C)GlutInitWindowSize function

This function allows you to request initial dimensions for future windows.

There is a callback function to inform you of the new window shape (whether initially opened, changed by your glutReshapeWindow() request, or changed directly by the user).

D)GlutInitWindowPosition function

glutInitWindowPosition and glutInitWindowSize set the *initial window position* and *size* respectively.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. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

E)GlutCreateWindow function

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.

F)Initialize

This is a user defined function that has several other Glut functions required for proper functioning of this code.It basically initializes all the important functions and parameters that we might need in the program

G)glutDiisplayFunc function

This function is used to display the output on the screen

H)glutKeyboardFunc function

This function is used to take the commands from the user like s for sorting and r for randomizing the input and so on.

I)glutTimerFunction
This function is used to set the time for which the program needs to run on an average.

H)Sorting Functions:

The sorting algorithms of Selection sort ,Bubble sort ,Ripple sort and Selection Sort have been additionally implemented in this program.

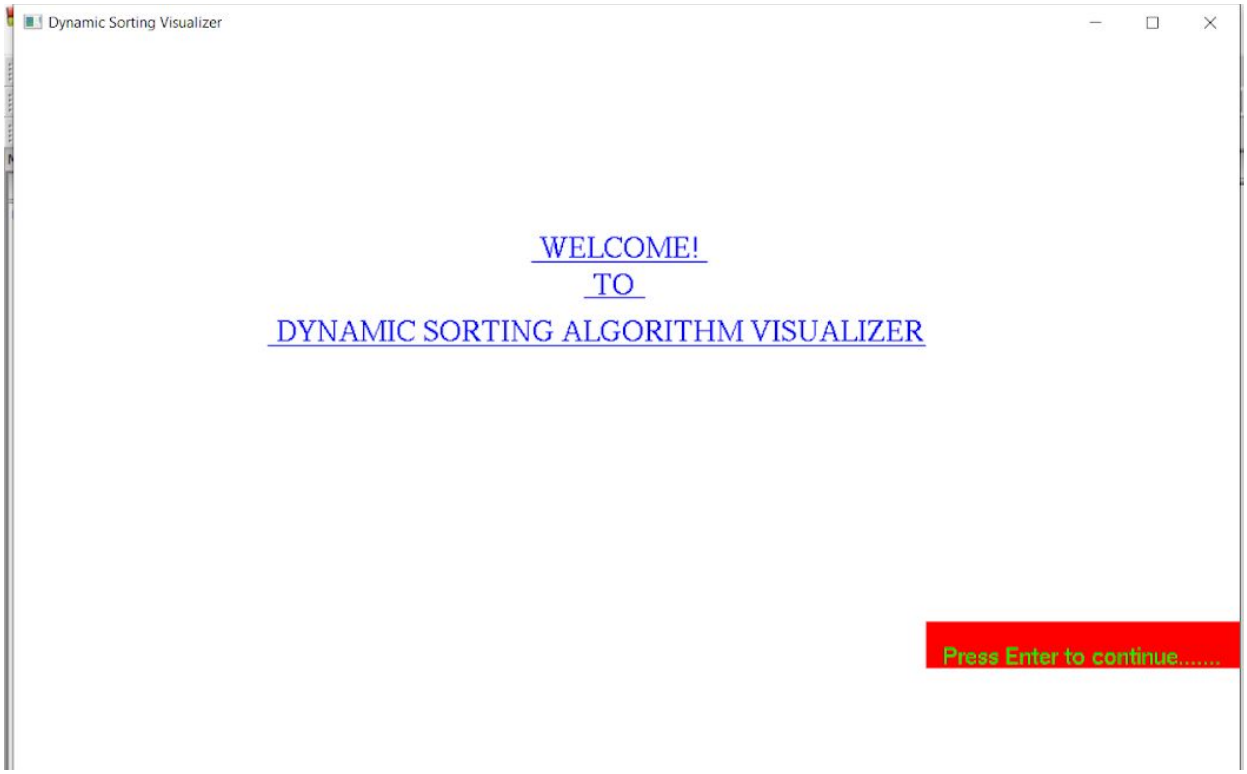
The program takes a random input in the form of a bar graph and then sorts it on the basis of user defined sorting algorithms and shows the sorting in a visualized form.This helps the user to comparatively study the sorting algorithms.

DESIGN AND IMPLEMENTATION

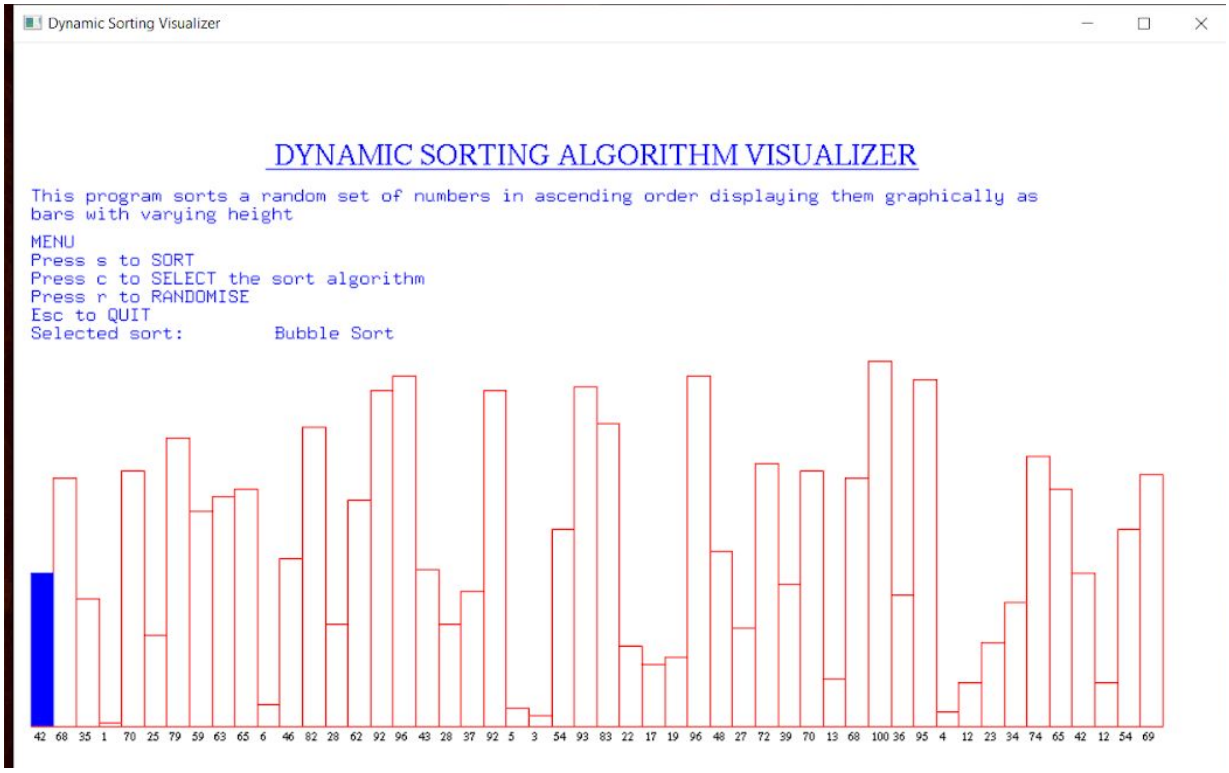
The design and implementation part of this project were carried out in C++ using Open Gl and Glut libraries which are part of C++.

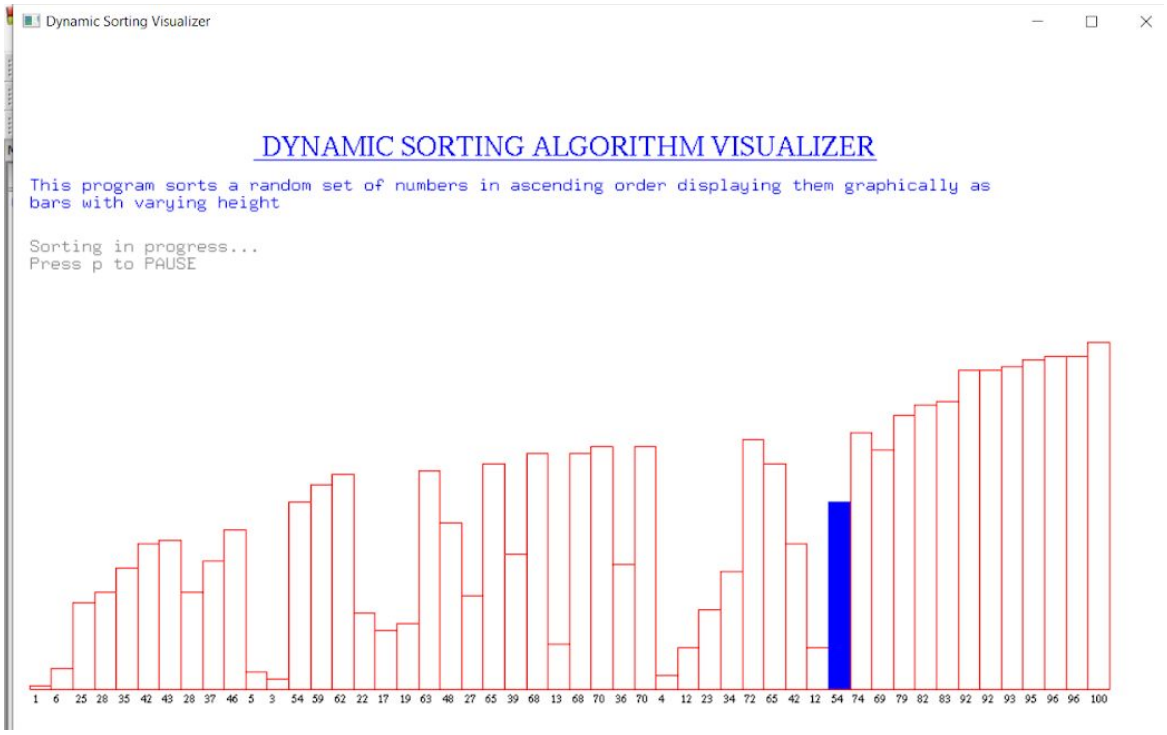
To Randomize the numbers use ‘r’
To Sort the numbers use ‘s’
To Select the sorting algorithm use ‘c’

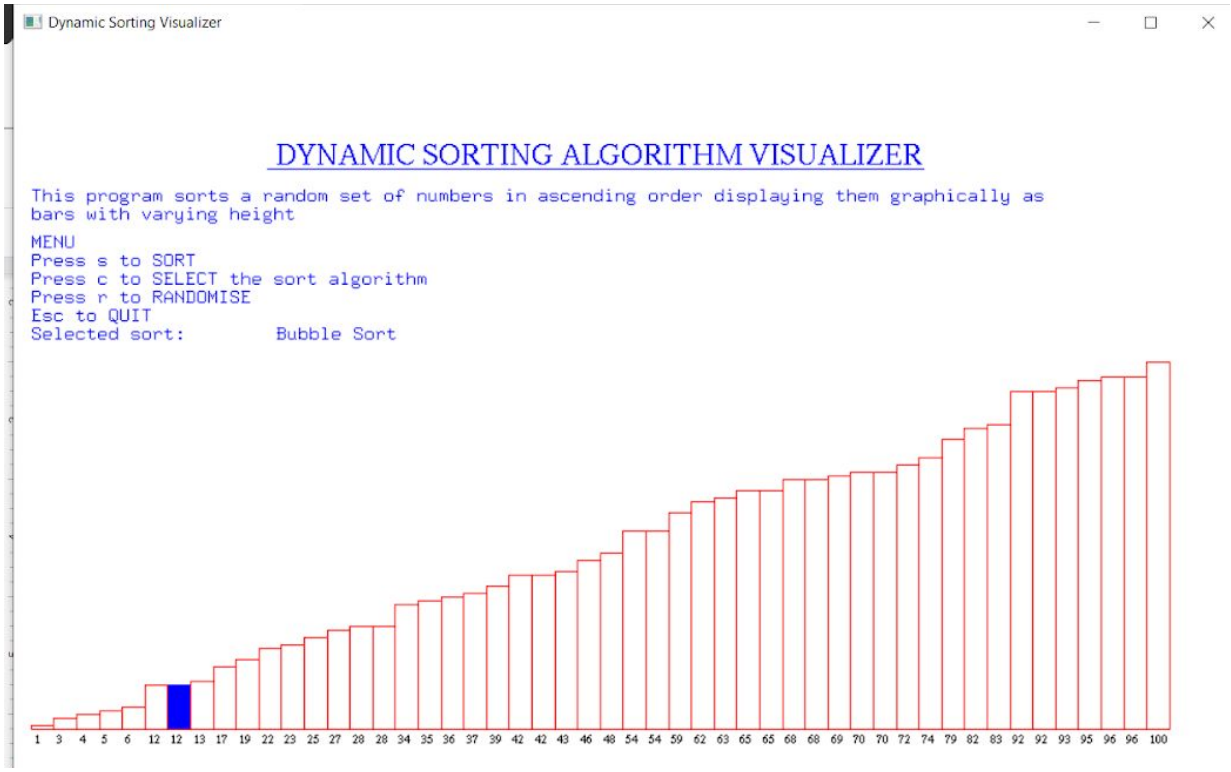
The following are few outputs of the code:



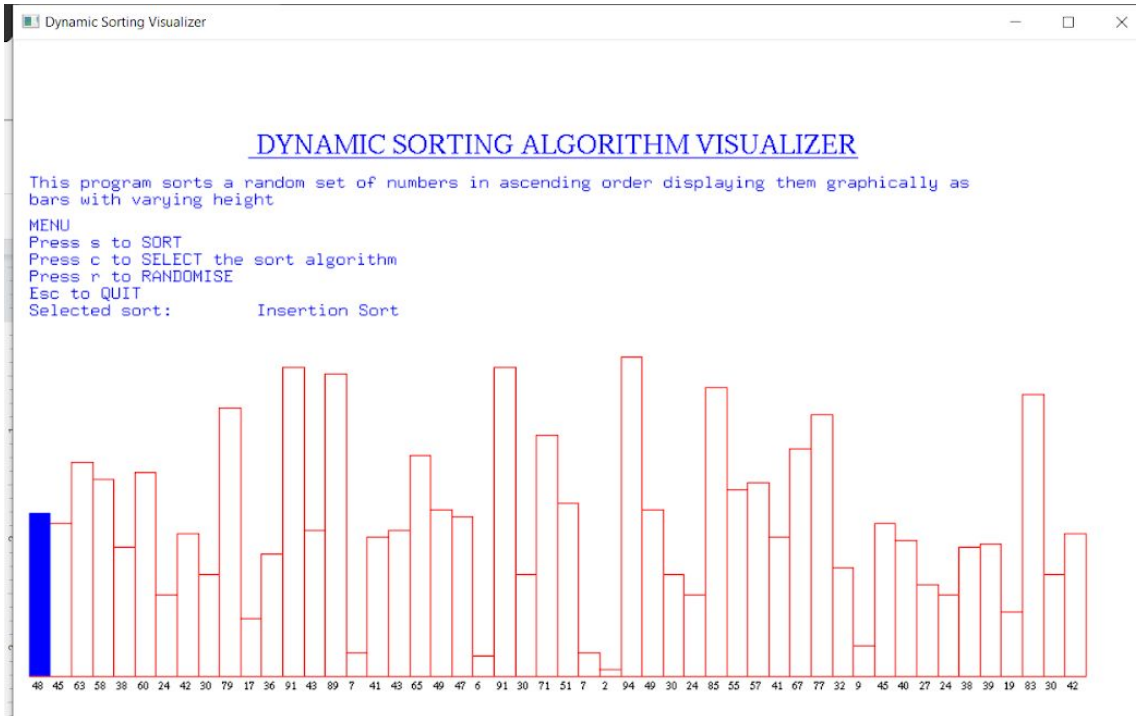
A)BUBBLE SORT

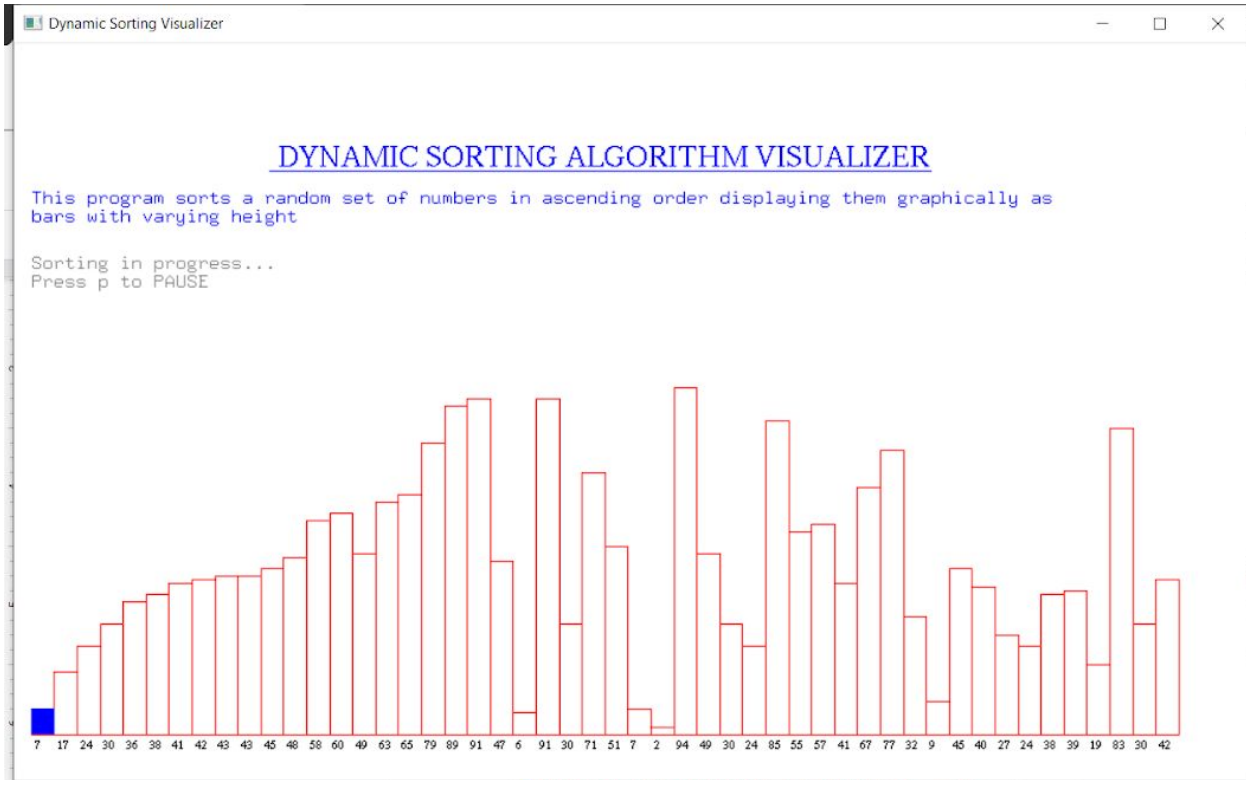


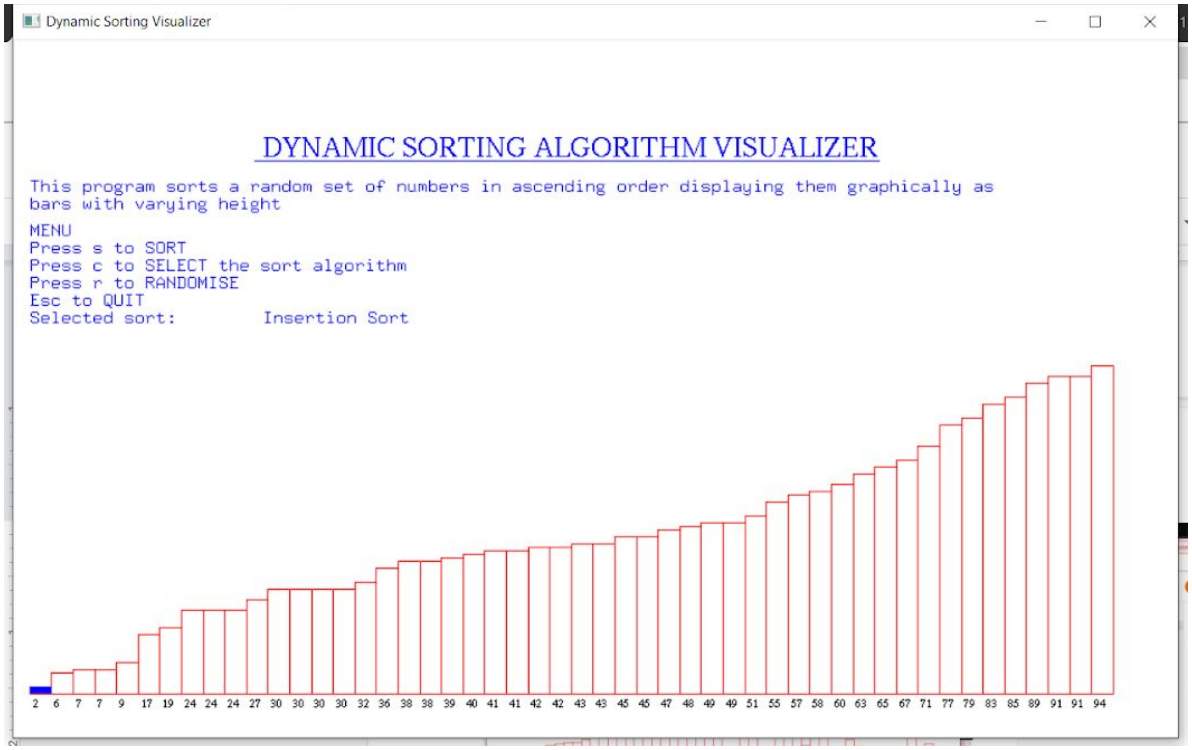




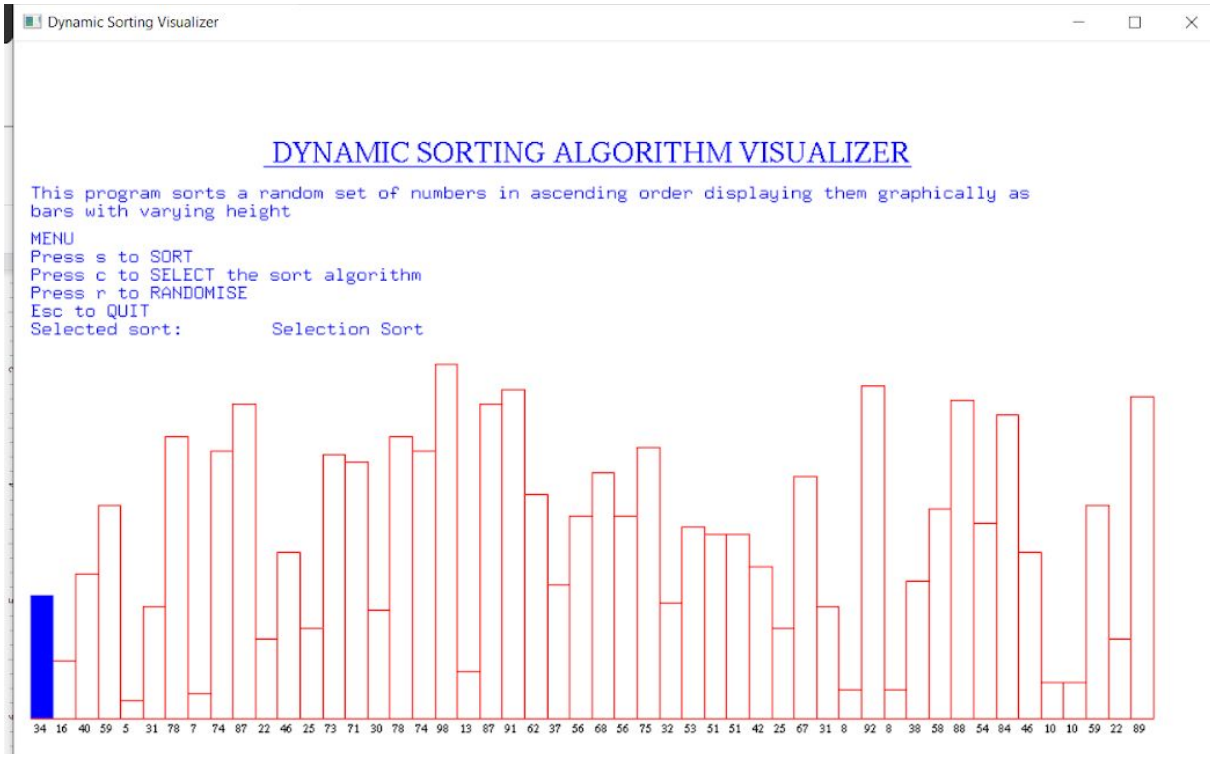
B)INSERTION SORT

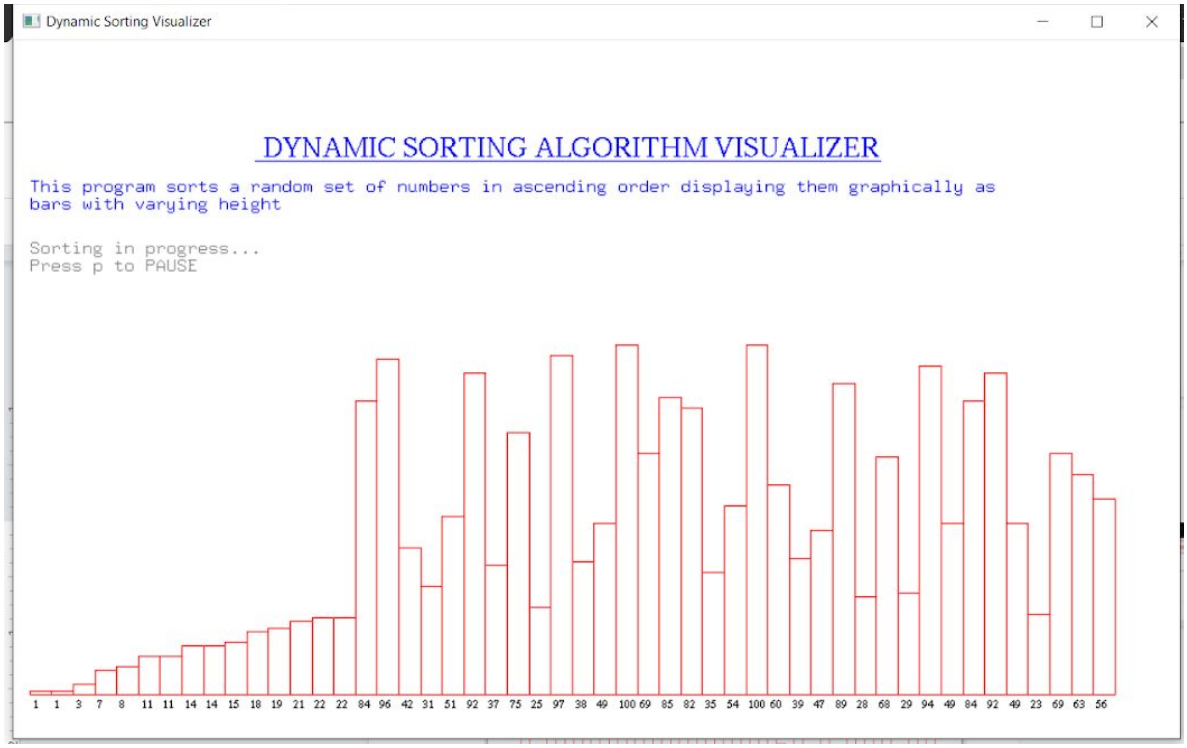

Dynamic Sorting Visualizer
DYNAMIC SORTING ALGORITHM VISUALIZER
This program sorts a random set of numbers in ascending order displaying them graphically as bars with varying height
MENU
Press s to SORT
Press c to SELECT the sort algorithm
Press r to RANDOMISE
Esc to QUIT
Selected sort: Insertion Sort
48 45 63 58 38 60 24 42 30 79 17 36 91 43 89 7 41 43 65 49 47 6 91 30 71 51 7 2 94 49 30 24 85 55 57 41 67 77 32 9 45 40 27 24 38 39 19 83 30 42

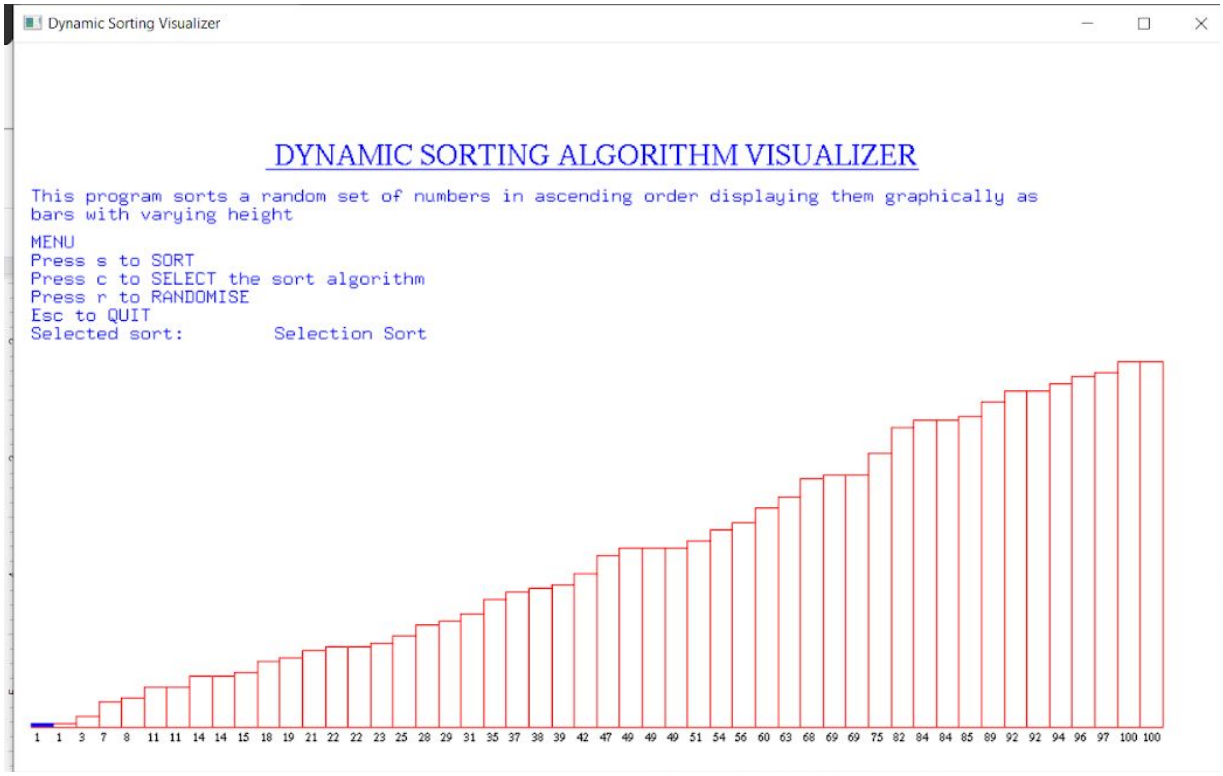

Dynamic Sorting Visualizer
DYNAMIC SORTING ALGORITHM VISUALIZER
This program sorts a random set of numbers in ascending order displaying them graphically as bars with varying height
Sorting in progress...
Press p to PAUSE
7 17 24 30 36 38 41 42 43 43 45 48 58 60 49 63 65 79 89 91 47 6 91 30 71 51 7 2 94 49 30 24 85 55 57 41 67 77 32 9 45 40 27 24 38 39 19 83 30 42



C)Selection Sort




Dynamic Sorting Visualizer
DYNAMIC SORTING ALGORITHM VISUALIZER
This program sorts a random set of numbers in ascending order displaying them graphically as bars with varying height
Sorting in progress...
Press p to PAUSE
1 1 3 7 8 11 11 14 14 15 18 19 21 22 22 84 96 42 31 51 92 37 75 25 97 38 49 100 69 85 82 35 54 100 60 39 47 89 28 68 29 94 49 84 92 49 23 69 63 56


Dynamic Sorting Visualizer
DYNAMIC SORTING ALGORITHM VISUALIZER
This program sorts a random set of numbers in ascending order displaying them graphically as bars with varying height
MENU
Press s to SORT
Press c to SELECT the sort algorithm
Press r to RANDOMISE
Esc to QUIT
Selected sort: Selection Sort
1 1 3 7 8 11 11 14 14 15 18 19 21 22 22 23 25 28 29 31 35 37 38 39 42 47 49 49 49 51 54 56 60 63 68 69 69 75 82 84 84 85 89 92 92 94 96 97 100 100

The code for the program has been attached herewith:

```
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<GL\glut.h>
#define SORT_NO 4       // Number of sorting algorithms
#define MAX 50          // Number of values in the array
#define SPEED 700       // Speed of sorting, must be greater than MAX always
int a[MAX];             // Array
int swapflag=0;         // Flag to check if swapping has occured
int i=0,j=0;        // To iterate through the array
int flag=0;             // For Insertion Sort
int dirflag=0;      // For Ripple Sort, to change direction at the ends
int count=1;        // For Ripple Sort, to keep count of how many are sorted at the
end
int k=0;                // To Switch from Welcome screen to Main Screen
int sorting=0;          // 1 if Sorted
char *sort_string[]={"Bubble Sort","Selection Sort","Insertion Sort","Ripple Sort"};
int sort_count=0;   // To cycle through the string

// Function to display text on screen char by char
void bitmap_output(int x, int y, char *string, void *font)
{
        int len, i;

        glRasterPos2f(x, y);
        len = (int) strlen(string);
        for (i = 0; i < len; i++) {
                glutBitmapCharacter(font, string[i]);
        }
}

// Function to integer to string
void int_str(int rad,char r[])
{
        itoa(rad,r,10);
}
```

```
void display_text()
{
	glColor3f(0,0,1);
	bitmap_output(150, 665, "DYNAMIC SORTING ALGORITHM
VISUALIZER",GLUT_BITMAP_TIMES_ROMAN_24);
	glBegin(GL_LINE_LOOP);
		glVertex2f(145, 660);
		glVertex2f(520, 660);
	glEnd();

	// other text small font
	bitmap_output(10, 625, "This program sorts a random set of numbers in
ascending order displaying them graphically as ",GLUT_BITMAP_9_BY_15);
	bitmap_output(10, 605, "bars with varying
height",GLUT_BITMAP_9_BY_15);


	if (sorting == 0)	// if not sorting display menu
	{
		bitmap_output(10, 575, "MENU",GLUT_BITMAP_9_BY_15);
		bitmap_output(10, 555, "Press s to
SORT",GLUT_BITMAP_9_BY_15);
		bitmap_output(10, 535, "Press c to SELECT the sort
algorithm",GLUT_BITMAP_9_BY_15);
		bitmap_output(10, 515, "Press r to
RANDOMISE",GLUT_BITMAP_9_BY_15);
		bitmap_output(10, 495, "Esc to QUIT",GLUT_BITMAP_9_BY_15);
		bitmap_output(10, 475, "Selected sort: ",GLUT_BITMAP_9_BY_15);
		bitmap_output(150, 475,
*(sort_string+sort_count),GLUT_BITMAP_9_BY_15);
	}
	else if (sorting == 1)		// while sorting
	{
		glColor3f(0.5,0.5,0.5);
		bitmap_output(10, 555, "Sorting in
progress...",GLUT_BITMAP_9_BY_15);
```

```
                bitmap_output(10, 535, "Press p to
PAUSE",GLUT_BITMAP_9_BY_15);
                glColor3f(0.0,0.0,0.0);
        }
}

void front()
{
        glColor3f(0.0,0.0,1.0);
        bitmap_output(300, 565,
"WELCOME!",GLUT_BITMAP_TIMES_ROMAN_24);
        glBegin(GL_LINE_LOOP);
                glVertex2f(295, 560);
                glVertex2f(395, 560);
        glEnd();
        bitmap_output(330, 525, "TO",GLUT_BITMAP_TIMES_ROMAN_24);
        glBegin(GL_LINE_LOOP);
                glVertex2f(325, 521);
                glVertex2f(360, 521);
        glEnd();

        bitmap_output(150, 475, "DYNAMIC SORTING ALGORITHM
VISUALIZER",GLUT_BITMAP_TIMES_ROMAN_24);
        glBegin(GL_LINE_LOOP);
                glVertex2f(145, 470);
                glVertex2f(520, 470);
        glEnd();
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_QUADS);

glVertex2f(520,120.0);glVertex2f(520,170);glVertex2f(796,170);glVertex2f(796,12
0.0);
        glEnd();
        glColor3f(0.0,1.0,0.0);
        bitmap_output(530, 125, "Press Enter to
continue.....",GLUT_BITMAP_HELVETICA_18);
}
```

```
void Initialize() {
	int temp1;

	// Reset the array
	for(temp1=0;temp1<MAX;temp1++){
		a[temp1]=rand()%100+1;
		printf("%d ",a[temp1]);
	}

	// Reset all values
	i=j=0;
	dirflag=0;
	count=1;
	flag=0;

	glClearColor(1,1,1,1);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(0, 699,0,799);
}

// Return 1 if not sorted
int notsorted(){
	int q;
	for(q=0;q<MAX-1;q++)
	{
		if(a[q]>a[q+1])
			return 1;
	}
	return 0;
}

// Main function for display
void display()
{
	int ix,temp;
```

```
	glClear(GL_COLOR_BUFFER_BIT);

	if(k==0)
		front();
	else{
		display_text();
		char text[10];

		for(ix=0;ix<MAX;ix++)
		{
			glColor3f(1,0,0);
			glBegin(GL_LINE_LOOP);
				glVertex2f(10+(700/(MAX+1))*ix,50);
				glVertex2f(10+(700/(MAX+1))*(ix+1),50);
				glVertex2f(10+(700/(MAX+1))*(ix+1),50+a[ix]*4);
				glVertex2f(10+(700/(MAX+1))*ix,50+a[ix]*4);
			glEnd();

			int_str(a[ix],text);
			//printf("\n%s",text);
			glColor3f(0,0,0);
			bitmap_output(12+(700/(MAX+1))*ix, 35,
text,GLUT_BITMAP_TIMES_ROMAN_10);
		}

		if(swapflag || sorting==0)
		{
			glColor3f(0,0,1);
			glBegin(GL_POLYGON);
				glVertex2f(10+(700/(MAX+1))*j,50);
				glVertex2f(10+(700/(MAX+1))*(j+1),50);
				glVertex2f(10+(700/(MAX+1))*(j+1),50+a[j]*4);
				glVertex2f(10+(700/(MAX+1))*j,50+a[j]*4);
			glEnd();
			swapflag=0;
		}
	}
```

```
        glFlush();
}

// Insertion Sort
void insertionsort()
{
        int temp;

        while(i<MAX)
        {
                if(flag==0){j=i; flag=1;}
                while(j<MAX-1)
                {
                        if(a[j]>a[j+1])
                        {
                                swapflag=1;
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;

                                goto A;
                        }
                        j++;
                        if(j==MAX-1){flag=0;}
                        printf("swap %d and %d\n",a[j],a[j+1]);
                }
                i++;
        }
        sorting=0;
        A:
        i=j=0;
}

// Selection Sort
void selectionsort()
{
        int temp,j,min,pos;
```

```
while(notsorted())
{

        while(i<MAX-1)
        {
                min=a[i+1];
                pos=i+1;
                if(i!=MAX-1)
                {
                        for(j=i+2;j<MAX;j++)
                        {
                                if(min>a[j])
                                {
                                        min=a[j];
                                        pos=j;
                                }
                        }
                }
                printf("\ni=%d min=%d at %d",i,min,pos);
                printf("\nchecking %d and %d",min,a[i]);
                if(min<a[i])
                {

                        //j=pos;
                        printf("\nswapping %d and %d",min,a[i]);
                        temp=a[pos];
                        a[pos]=a[i];
                        a[i]=temp;
                        goto A;
                }
                i++;
        }
}
sorting=0;
i=j=0;
A:  printf("");
```

```
}

//Bubble Sort
void bubblesort()
{
	int temp;
	while(notsorted())
	{
		while(j<MAX-1)
		{
			if(a[j]>a[j+1])
			{
				swapflag=1;
				temp=a[j];
				a[j]=a[j+1];
				a[j+1]=temp;

				goto A;
			}
			j++;
			if(j==MAX-1) j=0;
			printf("swap %d and %d\n",a[j],a[j+1]);
		}
	}
	sorting=0;
	A: printf("");
}

//Ripple Sort
void ripplesort()
{
	int temp;
	while(notsorted() && sorting)
	{
		if(dirflag==0)
		{
			while(j<MAX-1)
```

```
                {
                        if(a[j]>a[j+1])
                        {
                                swapflag=1;
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;

                                goto A;
                        }
                        j++;
                        if(j==MAX-1) {count++; j=MAX-count; dirflag=1-dirflag;}
                        printf("j=%d forward swap %d and %d\n",j,a[j],a[j+1]);
                }
        }
        else
        {
                while(j>=0)
                {
                        if(a[j]>a[j+1])
                        {
                                swapflag=1;
                                temp=a[j];
                                a[j]=a[j+1];
                                a[j+1]=temp;

                                goto A;
                        }
                        j--;
                        if(j==0){ dirflag=1-dirflag;}
                        printf("j=%d backward swap %d and %d\n",j,a[j],a[j+1]);
                }
        }
    }
    sorting=0;
    A: printf("");
}
```

```
// Timer Function, takes care of sort selection
void makedelay(int)
{
	if(sorting)
	{
		switch(sort_count)
		{
			case 0:	bubblesort();		break;
			case 1:	selectionsort();	break;
			case 2: insertionsort();	break;
			case 3: ripplesort();		break;
		}
	}
	glutPostRedisplay();
	glutTimerFunc(SPEED/MAX,makedelay,1);
}


// Keyboard Function
void keyboard (unsigned char key, int x, int y)
{
	if(key==13)
		k=1;
	if (k==1 && sorting!=1)
	{
		switch (key)
		{
			case 27	:	exit (0); // 27 is the ascii code for the ESC
key
			case 's' :	sorting = 1; break;
			case 'r' :	Initialize(); break;
			case 'c' :	sort_count=(sort_count+1)%SORT_NO;
break;
		}
	}
	if(k==1 && sorting==1)
```

```
                if(key=='p')   sorting=0;
}

// Main Function
int main(int argc,char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(1000,600);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Dynamic Sorting Visualizer ");
        Initialize();
        glutDisplayFunc(display);
        glutKeyboardFunc(keyboard);
        glutTimerFunc(1000,makedelay,1);
        glutMainLoop();
        return 0;
}
```

References:

a.Computer Graphics -Rajiv Chopra
b.Computer Graphics-Nabuhiko Mukhai
c.Computer Graphics-Jamees.D.Foley
d.Computer Graphics-Donald.D.Hearn