**ABSTRACT**

Quick Sort is a Divide and Conquer algorithm. It picks an element as pivot and partitions the given array around the picked pivot. Quick Sort is favoured in comparison to other methods in terms of speed and economy of storage. It’s popular due to its high efficiency. The project involves graphically depicting the Quick Sort method of sorting. Objects are used to graphically portray the working of the Quick Sort algorithm.

OpenGL is used in the making of the project. It is preferred in terms of ease of use, familiarity with its functionality and stable interface.

**INTRODUCTION**

Quicksort is a sorting algorithm whose worst-case running time is http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/bound.gif(*n*2) on an input array of *n* numbers. In spite of this slow worst-case running time, Quicksort is often the best practical choice for sorting because it is remarkably efficient on the average: its expected running time is http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/bound.gif(*n* lg *n*), and the constant factors hidden in the http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/bound.gif(*n* lg *n*) notation are quite small. It also has the advantage of sorting in place, and it works well even in virtual memory environments.

**Divide:**The array *A*[*p . . r*] is partitioned (rearranged) into two nonempty subarrays *A*[*p . . q*] and *A*[*q* + 1 . . *r*] such that each element of *A*[*p . . q*] is less than or equal to each element of *A*[*q* + 1 . . *r*]. The index *q* is computed as part of this partitioning procedure.

**Conquer:**The two subarrays *A*[*p . . q*] and *A*[*q* + 1 . . *r*] are sorted by recursive calls to quicksort.

**Combine:**Since the subarrays are sorted in place, no work is needed to combine them: the entire array *A*[*p . . r*] is now sorted.

The following procedure implements Quicksort.

QUICKSORT(*A,p,r*)

1 **if** *p* < *r*

2 **then** *q* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif PARTITION(*A,p,r*)

3 QUICKSORT(*A,p,q*)

4 QUICKSORT(*A,q* + 1,*r*)

To sort an entire array *A*, the initial call is QUICKSORT(*A*, 1, *length*[*A*]).

**Partitioning the array**

The key to the algorithm is the PARTITION procedure, which rearranges the subarray *A*[*p . . r*] in place.

PARTITION(*A,p,r*)

1 *x* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *A*[*p*]

2 *i* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *p* - 1

3 *j* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *r* + 1

4 **while** TRUE

5 **do repeat** *j* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *j* - 1

6 **until** *A*[*j*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/lteq12.gif *x*

*7* **repeat** *i* http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/arrlt12.gif *i* + 1

8 **until** *A*[*i*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/gteq.gif *x*

9 **if** *i* < *j*

10 **then** exchange *A*[*i*] http://staff.ustc.edu.cn/~csli/graduate/algorithms/images/dblarr12.gif *A*[*j*]

11 **else return** *j*

**IMPLEMENTATION**

Implementation was done using OpenGL.

**Special Functions:**

**void drawstring(GLfloat x,GLfloat y,char \*string)**

**{**

**char \*c;**

**glRasterPos2f(x,y);**

**for(c=string;\*c!='\0';c++)**

**{**

**glColor3f(1,0,0);**

**glutBitmapCharacter(currentfont,\*c);**

**}**

**}**

**// Used for Printing the characters on the window.**

**CODE**

#include<windows.h>

#include<GL/glut.h>

#include<stdio.h>

#include<string.h>

int y=400,list[100],n;

int l,m,o,r;

int z;

void drawstring(GLfloat x,GLfloat y,char \*string);

void \*currentfont;

int low=0,high=n-1, posx = 10, posy = 800;

void quickSort(int arr[], int low, int high);

void \*font = GLUT\_BITMAP\_HELVETICA\_18;

void drawstring(GLfloat x,GLfloat y,char \*string)

{

char \*c;

glRasterPos2f(x,y);

for(c=string;\*c!='\0';c++)

{

glColor3f(1,0,0);

glutBitmapCharacter(currentfont,\*c);

}

}

void setFont(void \*font)

{

currentfont=font;

}

void delay()

{

int i,j;

for(i=0;i<32000;i++)

for(j=0;j<15000;j++);

}

void output(float x,float y,int\* pr)

{

int i;

glRasterPos2f(x,y);

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

{

x=posx+i\*50;

y=posy-10;

glColor3f(1,1,0);

glBegin(GL\_POLYGON);

glVertex2f(x,y+40);

glVertex2f(x+30,y+40);

glVertex2f(x+30,y);

glVertex2f(x,y);

glEnd();

glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,pr[i]+'0');

glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,' ');

/\* glutBitmapCharacter(font,' ');

glutBitmapCharacter(font,' ');\*/

}

}

void out(int x,int y,int pivot)

{

x=posx+460;

y=posy-10;

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(x,y+40);

glVertex2f(x+30,y+40);

glVertex2f(x+30,y);

glVertex2f(x,y);

glEnd();

glColor3f(0,0,0);

x=posx+460;

y=posy;

glRasterPos2f(x+10,y);

glutBitmapCharacter(font,pivot+'0');

}

void outlow(int x,int y,int l)

{

x=posx+500;

y=posy-10;

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2f(x,y+40);

glVertex2f(x+30,y+40);

glVertex2f(x+30,y);

glVertex2f(x,y);

glEnd();

glColor3f(0,0,0);

x=posx+500;

y=posy;

glRasterPos2f(x+10,y);

glutBitmapCharacter(font,l+'0');

}

void outhigh(int x,int y,int m)

{

x=posx+540;

y=posy-10;

glColor3f(0,1,0);

glBegin(GL\_POLYGON);

glVertex2f(x,y+40);

glVertex2f(x+30,y+40);

glVertex2f(x+30,y);

glVertex2f(x,y);

glEnd();

glColor3f(0,0,0);

x=posx+540;

y=posy;

glRasterPos2f(x+10,y);

glutBitmapCharacter(font,m+'0');

}

void init(void)

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glClearColor(0.184, 0.310, 0.310,0);

glLineWidth(1.5);

glColor3f(0.65,0.65,0.65);

gluOrtho2D(0.0,1500.0,0.0,1000.0);

setFont(GLUT\_BITMAP\_TIMES\_ROMAN\_24);

}

void swap(int\* a, int\* b)

{

int t = \*a;

\*a = \*b;

\*b = t;

}

int partition (int arr[], int low, int high)

{

int pivot = arr[high];

int i = (low - 1);

glColor3f(1.0,0.0,0.0);

output(posx, posy, list);

delay();

glFlush();

posy-=50;

if(posy==100)

{

posx=600;

posy=700;

}

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

{

if (arr[j] <= pivot)

{

i++;

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

//glutBitmapCharacter(font,arr[i]+'0');

//glutBitmapCharacter(font,arr[j]+'0');

glColor3f(0,0,1);

output(posx, posy, list);

glColor3f(1,0,0);

out(posx, posy, pivot);

outlow(posx, posy, l);

outhigh(posx, posy, m);

delay();

glFlush();

posy-=50;

if(posy==100)

{

posx=600;

posy=700;

}

}

// glutPostRedisplay();

}

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

//glutBitmapCharacter(font,arr[i+1]+'0');

//glutBitmapCharacter(font,arr[high]+'0');

glColor3f(1,0,0);

output(posx, posy, list);

glColor3f(1,0,0);

out(posx, posy, pivot);

outlow(posx, posy, l);

outhigh(posx, posy, m);

delay();

glFlush();

posy-=50;

if(posy==100)

{

glColor3f(1,0,0);

glBegin(GL\_LINE);

glVertex2f(500,900);

glVertex2f(500,100);

glEnd();

posx=600;

posy=700;

}

return (i + 1);

}

void quickSort(int arr[], int low, int high)

{

o=low;

l=arr[o];

r=high;

m=arr[r];

// m=high;

if (low < high)

{

int pi = partition(arr, low, high);

print\_from\_to(arr,low,high);

quickSort(arr, low, pi - 1);

quickSort(arr, pi + 1, high);

}

}

void display()

{

glClear(GL\_COLOR\_BUFFER\_BIT);

glColor3f(0,1,1);

drawstring(400,950,"QUICKSORT");

glColor3f(1,0,0);

glBegin(GL\_POLYGON);

glVertex2f(40,900);

glVertex2f(80,900);

glVertex2f(80,860);

glVertex2f(40,860);

glEnd();

drawstring(100,865,"PIVOT");

glColor3f(1,1,1);

glBegin(GL\_POLYGON);

glVertex2f(300,900);

glVertex2f(340,900);

glVertex2f(340,860);

glVertex2f(300,860);

glEnd();

drawstring(350,865,"LOW");

glColor3f(0,1,0);

glBegin(GL\_POLYGON);

glVertex2f(540,900);

glVertex2f(580,900);

glVertex2f(580,860);

glVertex2f(540,860);

glEnd();

drawstring(600,865,"HIGH");

glColor3f(0,0,0);

quickSort(list,0,n-1);

//glColor3f(0,1,1);

//drawstring(posx,posy-40,"!!!!!SORTED!!!!!");

glColor3f(0,1,1);

drawstring(posx,posy-40,"!!!!!SORTED!!!!!");

glColor3f(1,0,0);

output(posx, posy, list);

glFlush();

}

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

{

printf("enter the value of n \n");

scanf("%d",&n);

int i=0;

printf("enter the values \n");

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

{

scanf("%d",&list[i]);

}

glutInit(&argc,argv);

glutInitDisplayMode(GLUT\_RGB|GLUT\_SINGLE);

glutInitWindowSize(1500,1000);

glutInitWindowPosition(0,0);

glutCreateWindow("project");

glutDisplayFunc(display);

printf("\n\n\n\n");

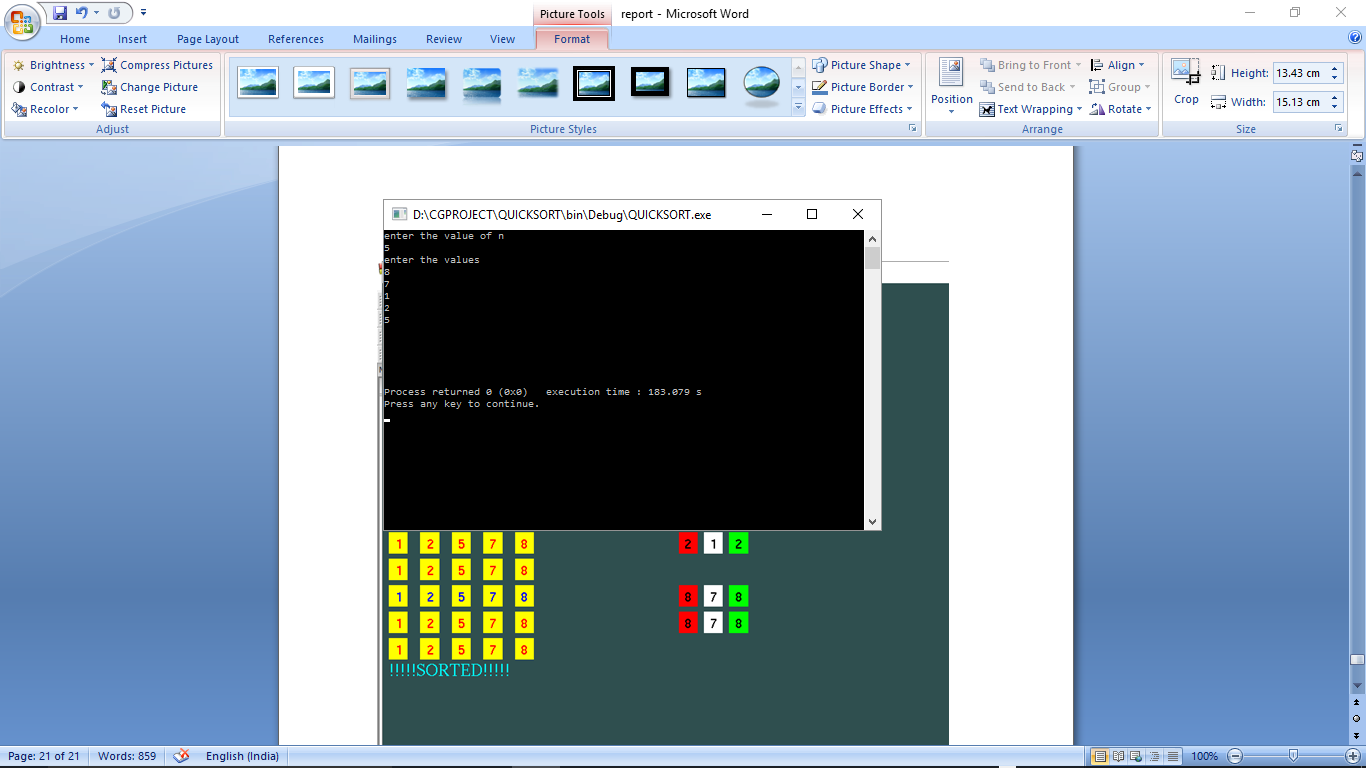
init();

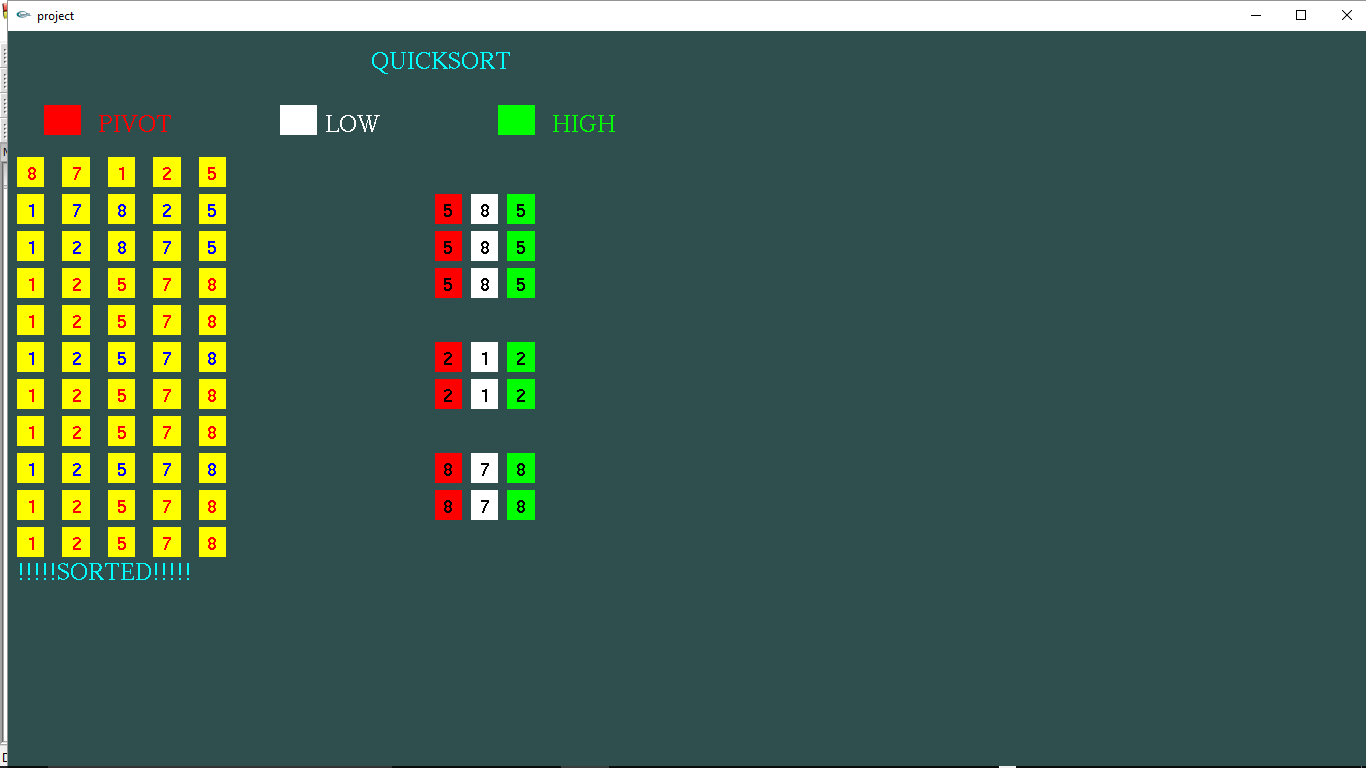
glutMainLoop();

return 0;

}

**RESULT**

****

****