|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **Министерство науки и высшего образования Российской Федерации**  Калужский филиал  федерального государственного бюджетного  образовательного учреждения высшего образования  ***«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»***  ***(КФ МГТУ им. Н.Э. Баумана)*** |

**ФАКУЛЬТЕТ** ***ИУК «Информатика и управление»***

**КАФЕДРА** \_\_***ИУК4 «Программное обеспечение ЭВМ, информационные технологии»***

**ЛАБОРАТОРНАЯ РАБОТА №4**

**«Реализация алгоритма сортировки»**

**ДИСЦИПЛИНА: «Логика и теория алгоритмов»**

|  |  |  |
| --- | --- | --- |
| Выполнил: студент гр. ИУК4-42Б | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Карельский М.К. )  (Подпись) |
| Проверил: | | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ ( Амеличев Г.Э. )  (Подпись) |
| Дата сдачи (защиты):  Результаты сдачи (защиты): | | |
|  | - Балльная оценка:  - Оценка: | |

Калуга, 2022

**Цель:** формирование практических навыков выбора и разработки эффективного алгоритма сортировки с учетом особенностей конкретной задачи и структур обрабатываемых данных.

**Задачи:** реализовать алгоритм сортировки и исследовать его вычислительную сложность.

**Вариант №8**

**Сортировка подсчётом.** Проиллюстрировать работу сортировки на примере цветов. Сгенерировать массив цветов (RGB) и на цветовой палитре продемонстрировать процесс работы сортировки на каждой итерации (программно). На примере HSL модели.

**Псевдокод:**

для i от 1 до размер\_сортируемого N \* c1

счетчик\_цветов[цвета[i]] = счетчик\_цветов[цвета[i]] + 1 N \* c2

для j от 1 до размер\_палитры c3

для i от 1 до счетчик\_цветов[j] N \* c4

результат.добавить\_в\_конец(j) N \* c5

**Сложность:**

O(N \* c1 + N \* c2 + c3 + N \* c4 + N \* c5) = O(N \* (c1 + c2 + c4 + c5) + c3) = = O(N)

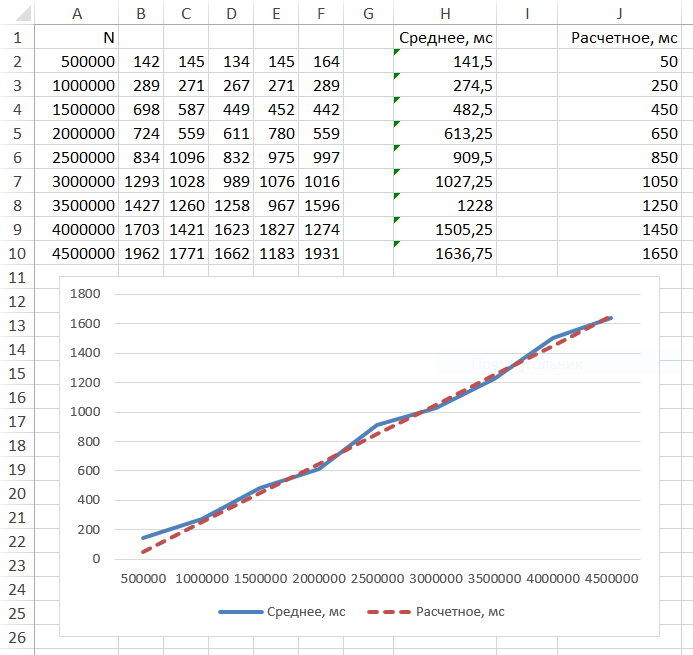


Рис. 1. Время выполнения

**Результат:**

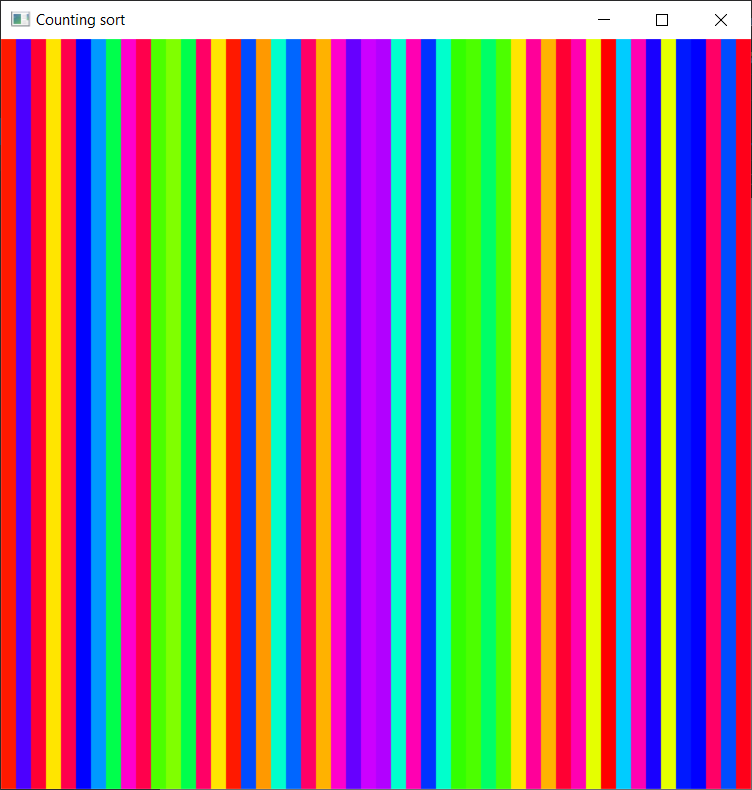
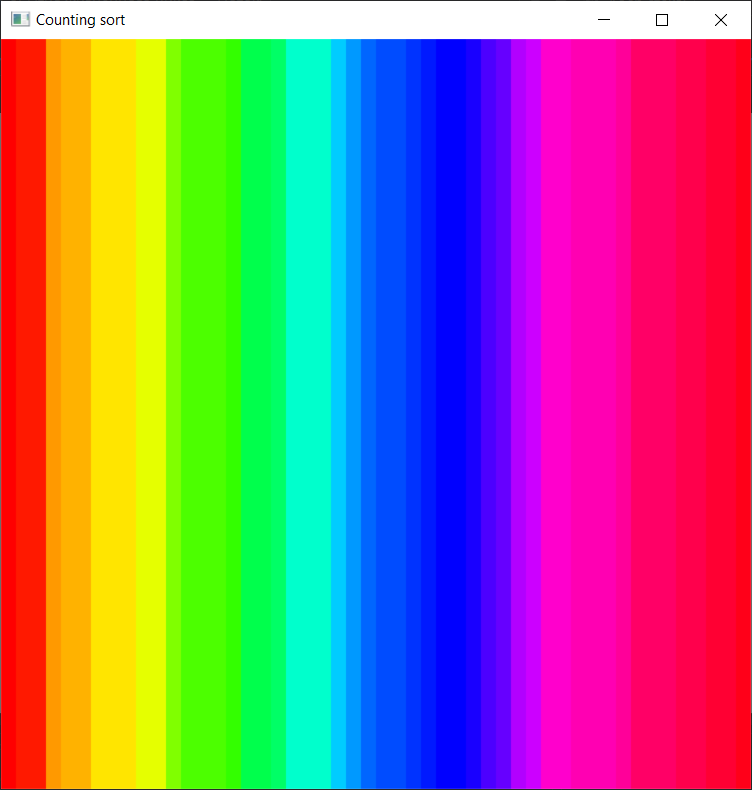
 

Рис. 2. Результат

**Вывод:** в ходе выполнения лабораторной работы были получены практические навыки реализации алгоритма сортировки подсчетом, наглядной демонстрации его работы методами OpenGL.

**ПРИЛОЖЕНИЯ**

**Листинг:**

#include "glew.h"

#include "glut.h"

#include <vector>

#include <time.h>

#include <algorithm>

#include <fstream>

#include <iostream>

struct RGB

{

float red;

float green;

float blue;

};

struct HSL

{

float Hue;

float Saturation;

float Lightness;

};

std::vector<int> colorCodes{};

size\_t numberOfColors = 50;

std::vector<RGB> possibleValues{};

int\* colorCount = nullptr;

size\_t iteration = 0;

std::vector<int> sorted{};

bool showUnsorted = true;

HSL ConvertRGBToHSL(RGB rgb)

{

float cmax = std::max({ rgb.red, rgb.green, rgb.blue });

float cmin = std::min({ rgb.red, rgb.green, rgb.blue });

float delta = cmax - cmin;

float H = 0;

if (delta != 0)

if (cmax == rgb.red && rgb.green >= rgb.blue)

H = 60.0f \* (rgb.green - rgb.blue) / delta;

else if (cmax == rgb.red && rgb.green < rgb.blue)

H = 60.0f \* (rgb.green - rgb.blue) / delta + 360;

else if (cmax == rgb.green)

H = 60.0f \* (rgb.blue - rgb.red) / delta + 120;

else

H = 60.0f \* (rgb.red - rgb.green) / delta + 240;

while (H >= 360)

H -= 360;

while (H < 0)

H += 360;

float L = 0.5 \* (cmax + cmin);

float S = 0;

if (L != 0 && delta != 0)

S = delta / (1 - abs(1 - cmax - cmin));

return { H, S, L };

}

bool CompareRGB(RGB first, RGB second)

{

HSL firstHSL = ConvertRGBToHSL(first);

HSL secondHSL = ConvertRGBToHSL(second);

if (firstHSL.Hue < secondHSL.Hue)

return true;

if (firstHSL.Hue > secondHSL.Hue)

return false;

if (firstHSL.Saturation < secondHSL.Saturation)

return true;

if (firstHSL.Saturation > secondHSL.Saturation)

return false;

return firstHSL.Lightness < secondHSL.Lightness;

}

std::vector<int> CountingSort()

{

int\* colorCountLocal = new int[possibleValues.size()]{};

for (size\_t i{}; i < colorCodes.size(); ++i)

++colorCountLocal[colorCodes[i]];

std::vector<int> result{};

for (size\_t j{}; j < possibleValues.size(); ++j)

for (int i{}; i < colorCountLocal[j]; ++i)

result.push\_back(j);

delete[] colorCountLocal;

return result;

}

void GetColorCount()

{

colorCount = new int[possibleValues.size()]{};

for (size\_t i{}; i < colorCodes.size(); ++i)

++colorCount[colorCodes[i]];

}

void NextIteration()

{

for (int i{}; i < colorCount[iteration]; ++i)

sorted.push\_back(iteration);

++iteration;

}

// Вызывается для рисования сцены

void RenderScene(void)

{

// Очистка окна текущим цветом очистки

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

size\_t size = colorCodes.size();

float step = 2 / (float)size;

for (size\_t i{}; i < size; ++i)

{

float upperEdge = 1.0f;

float lowerEdge = -1.0f;

float leftEdge = -1.0 + i \* step;

float rightEdge = -1.0 + (i + 1) \* step;

if (showUnsorted)

glColor3f(possibleValues[colorCodes[i]].red, possibleValues[colorCodes[i]].green, possibleValues[colorCodes[i]].blue);

else

if (i < sorted.size())

glColor3f(possibleValues[sorted[i]].red, possibleValues[sorted[i]].green, possibleValues[sorted[i]].blue);

else

glColor3f(0.0f, 0.0f, 0.0f);

glBegin(GL\_POLYGON);

glVertex2f(rightEdge, lowerEdge);

glVertex2f(rightEdge, upperEdge);

glVertex2f(leftEdge, upperEdge);

glVertex2f(leftEdge, lowerEdge);

glEnd();

}

showUnsorted = false;

// Отобразить результаты

glutSwapBuffers();

}

// Эта функция выполняет необходимую инициализацию в контексте

// визуализации.

void SetupRC()

{

srand(time(NULL));

for (int i{}; i < 11; ++i)

for (int j{}; j < 11; ++j)

for (int k{}; k < 11; ++k)

{

float red = i \* 1.0f / 10.0f;

float green = j \* 1.0f / 10.0f;

float blue = k \* 1.0f / 10.0f;

float cmax = std::max({ red, green, blue });

float cmin = std::min({ red, green, blue });

float delta = cmax - cmin;

float L = 0.5 \* (cmax + cmin);

float S = 0;

if (L != 0 && delta != 0)

S = delta / (1 - abs(1 - cmax - cmin));

if (L == 0.5 && S == 1)

possibleValues.push\_back({ red, green, blue });

}

std::sort(possibleValues.begin(), possibleValues.end(), CompareRGB);

std::vector<int> times{};

for (size\_t size = 500000; size < 5000000; size += 500000)

{

colorCodes.clear();

numberOfColors = size;

for (size\_t i{}; i < numberOfColors; ++i)

colorCodes.push\_back(rand() % possibleValues.size());

float start = clock();

CountingSort();

float end = clock();

times.push\_back((end - start) / (CLOCKS\_PER\_SEC / 1000));

}

std::ofstream file("WorkTimes.csv");

for (size\_t i{}; i < times.size(); ++i)

{

std::cout << times[i] << "\n";

file << times[i] << '\n';

}

file.close();

GetColorCount();

}

void TimerFunction(int value)

{

if (!showUnsorted && iteration < possibleValues.size())

NextIteration();

// Перерисовка сцены

RenderScene();

// Сброс таймера

glutTimerFunc(500, TimerFunction, 1);

}

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

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB);

glutInitWindowSize(600, 600);

glutCreateWindow("Counting sort");

SetupRC();

glutDisplayFunc(RenderScene);

glutTimerFunc(2000, TimerFunction, 1);

glutMainLoop();

return 0;

}