

МИНОБРНАУКИ РОССИИ

федеральное государственное бюджетное образовательное учреждение высшего образования

«Санкт-Петербургский государственный технологический институт (технический университет)»

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| УГНС | | 09.00.00 | Информатика и вычислительная техника | | |
| Направление подготовки | | 09.03.03 | Прикладная информатика | | |
| Направленность (профиль) | |  | Прикладная информатика в химии | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | I | | | Группа | 415 |

Отчёт по контрольной работе № 3

Вариант №4

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 415 |  |  |  | Шарашидзе Никита Леванович |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Федин Алексей Константинович |
|  |  | (дата, подпись) |  | Корниенко Иван Григорьевич |

Санкт-Петербург

2022

Содержание

[1 Постановка задачи 3](#_Toc99067513)

[2 Исходные данные 3](#_Toc99067514)

[3 Особые ситуации 3](#_Toc99067515)

[4 Математические методы и алгоритмы решения задач 3](#_Toc99067516)

[5 Форматы представления данных 3](#_Toc99067517)

[6 Структура программы 4](#_Toc99067518)

[7 Блок-схемы алгоритмов 4](#_Toc99067519)

[8 Результаты работы программы 5](#_Toc99067522)

[9 Описание хода выполнения контрольной работы 8](#_Toc99067523)

[10 Исходный текст программы 8](#_Toc99067524)

## 1 Постановка задачи

Необходимо составить программу для сортировки массива данных методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировки. Вывести на экран неупорядоченный (один раз) и упорядоченные (для каждого из методов) массивы данных. Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки.

Реализовать абстрактный базовый класс ISort, содержащий чистыйвиртуальный метод Sort и необходимые счетчики, от которого наследовать подклассы для реализации сортировок.

Каждая нечётная строка сортируется по возрастанию абсолютных величин, каждый чётный столбец матрицы сортируется по возрастанию.

## 2 Исходные данные

## Неупорядоченная матрица из N строк и M столбцов задаётся и заполняется один раз (с клавиатуры, из файла или случайными числами), далее она используется для каждого из методов сортировки.

## 3 Особые ситуации

Необходимо рассмотреть следующие особые ситуации:

– Данные, введённые пользователем, содержат что-либо помимо чисел.

– Файл, из которого пользователь попытается извлечь данные о матрице, пуст или содержит некорректные значения.

– Пользователь введёт отрицательное или нулевое количество строк/столбцов.

– Пользователь попытается создать файл для сохранения данных с уже занятым именем.

– Пользователь выберет несуществующий пункт меню.

## 4 Математические методы и алгоритмы решения задач

Согласно постановке задачи, для составления программы использованы пять различных способов сортировки данных: пузырьковый, отбор, вставки, Шелла и быстрая сортировка.

Алгоритм сортировки пузырьковым методом состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются N – 1 раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива.

Первый шаг алгоритма сортировки методом отбора заключается в поиске минимального элемента массива (изначально за минимальный элемент принимается первый элемент массива). Вторым шагом производится обмен этого элемента с первой неотсортированной позицией (обмен не нужен, если минимальный элемент уже находится на данной позиции). Далее сортируются остальные элементы, исключая уже отсортированные.

На каждом шаге алгоритма сортировки вставками выбирается один из элементов массива и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока все элементы не будут на своих местах.

На первом шаге алгоритма сортировки Шелла два элемента, расположенных друг от друга на расстоянии d, равном N\2, сравниваются между собой, и в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние d сокращается в два раза. Постепенно расстояние между элементами уменьшается, и на d=1 проход по массиву происходит в последний раз.

На первом шаге алгоритма быстрой сортировки выбирается опорный элемент массива – средний элемент массива. На втором шаге сравниваются все остальные элементы массива с опорным и переставляются таким образом, чтобы разбить массив на два непрерывных отрезка, следующих друг за другом: «элементы меньше опорного» и «большие». Далее для отрезков «меньших» и «больших» значений выполняется рекурсивно та же последовательность операций, если длина отрезка больше единицы.

## 5 Форматы представления данных

Таблица 1 – Переменные, используемые в программе

|  |  |  |
| --- | --- | --- |
| **Имя** | **Тип** | **Описание** |
| comparison\_counter | int | Количество сравнений в матрице после сортировки |
| swap\_counter | int | Количество перестановок в матрице после сортировки |
| myFile | ofstream | Файл, в который записываются/их которого считываются данные |
| number\_of\_lines | int | Количество строк в матрице |
| number\_of\_colums | int | Количество столбцов в матрице |
| matrix | vector<vector<int>> | Матрица |

Таблица 2 – Классы, используемые в программе

|  |  |
| --- | --- |
| **Имя класса** | **Описание** |
| BubbleSort | Для реализации пузырьковой сортировки |
| ColumSum | Для получения суммы цифр чисел в колонке |

Продолжение таблицы 2.

|  |  |
| --- | --- |
| InputCheck | Проверяет ввод пользователя на корректность значений |
| InsertionSort | Для реализации сортировки вставками |
| ISort | Абстрактный базовый класс для всех сортировок |
| QuickSort | Для реализации быстрой сортировки |
| SelectionSort | Для реализации отбором |
| ShellSort | Для реализации сортировки методом Шелла |
| SortedMatrix | Хранит отсортированную матрицу |

## 6 Структура программы

Для оптимизации работы с кодом было принято решение разделить программу на модули.

Таблица 3 – Модули, составляющие программу

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| Main.cpp | Основной модуль |
| BubbleSort.cpp | Хранит класс пузырьковой сортировки |
| GetColum.cpp | Для получения столбцов матрицы |
| GetFileName.cpp | Позволяет вводить данные из файла и проверять файл |
| InitialMatrixFileOutput.cpp | Модуль для сохранения матрицы в файл |
| InputCheck.cpp | Поверка данных, введённых пользователем |
| Check.cpp | Проверка введённого пункта меню |
| InsertionSort.cpp | Сортировка вставками |
| ISort.cpp | Базовый класс всех сортировок |
| MatrixInput.cpp | Модуль для ввода матрицы |
| QuickSort.cpp | Быстрая сортировка |
| SaveData.cpp | Модуль для сохранения данных в файл |
| SelectionSort.cpp | Сортировка выбором |
| ShellSort.cpp | Сортировка методом Шелла |
| SortedMatrix.cpp | Модуль для отсортированной матрицы |
| UnitTest.cpp | Модуль для юнит-тестов |

## Таблица 4 – Функции, составляющие Check.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| CheckMenu | Проверяет пункт меню, введённый пользователем |

## Таблица 5 – Функции, составляющие GetFileName.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| FileInput | Получение данных из введённого файла |
| FileOutput | Запись матрицы в файл |
| GetFileName | Получает имя файла |

## Таблица 6 – Функции, составляющие GetColum.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| GetColum | Хранение столбцов матрицы |
| SetColum | Получение чисел в столбцах матрицы |

## Таблица 7 – Функции, составляющие InitialMatrixFileOutput.cpp

|  |  |
| --- | --- |
| **Имя** | **Описание** |
| InitialMatrixFileOutput | Позволяет сохранить изначальную матрицу |

## Таблица 8 – Функции, составляющие MatrixInput.cpp

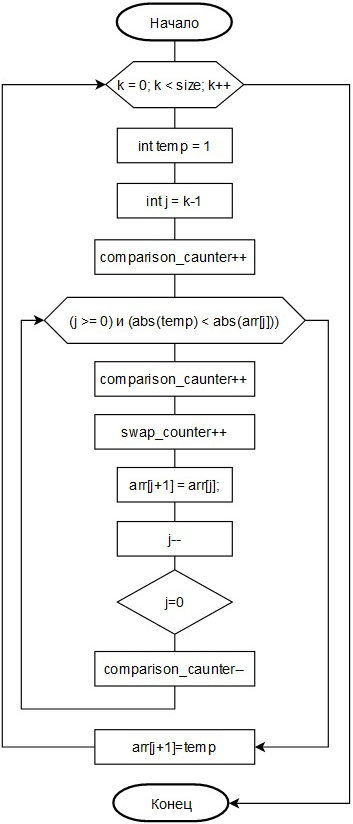
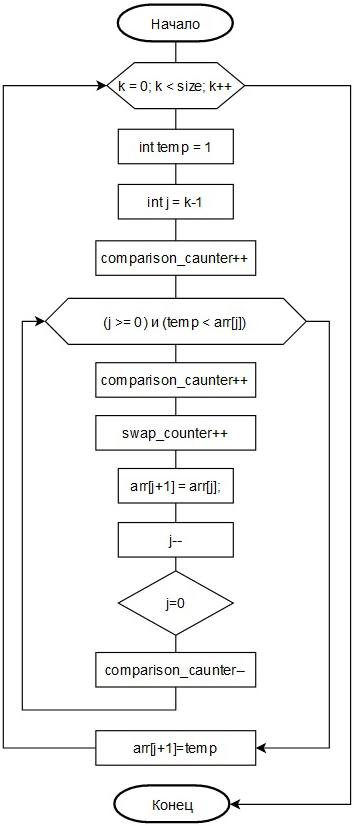
|  |  |
| --- | --- |
| **Имя** | **Описание** |
| MatrixInput | Для выбора пользователем способа вода матрицы |
| ConsoleInput | Для ввода матрицы вручную |
| FileInput | Для ввода матрицы из файла |
| RandomInput | Для заполнения матрицы случайными числами |

## Таблица 9 – Функции, составляющие SaveData.cpp

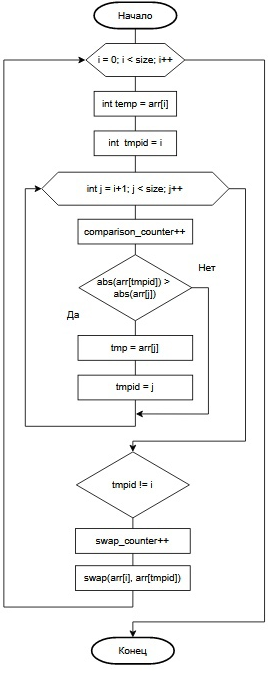
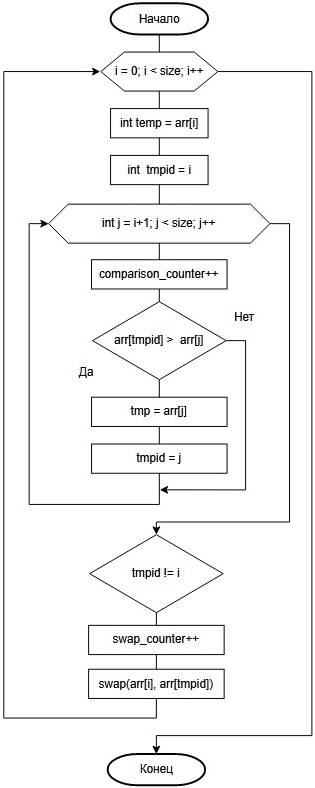
|  |  |
| --- | --- |
| **Имя** | **Описание** |
| SaveData | Позволяет сохранить отсортированную матрицу |

## https://sun9-70.userapi.com/impg/Cpo7IfdN-YiKgJoOV7If-gIdLuYG9jkOVF5oXQ/MgnTDETTXsk.jpg?size=376x876&quality=96&sign=d90314613646e116eb4728710021158b&type=albumhttps://sun9-25.userapi.com/impg/eiwRqMMEoyadndKfD5A-Ja62ROdKj0p2SNNs6g/qW3EIAkNSf8.jpg?size=352x860&quality=96&sign=6e209bd8ffe6b6d4ba11dd4607e48379&type=album7 Блок-схемы алгоритмов

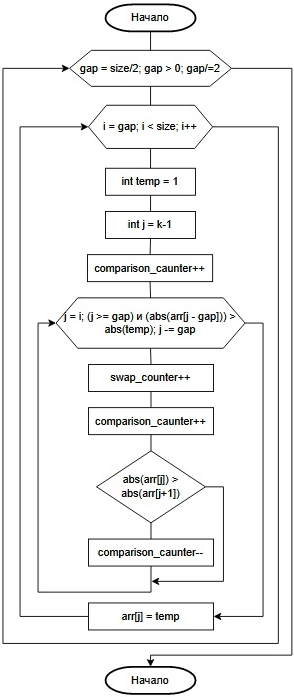
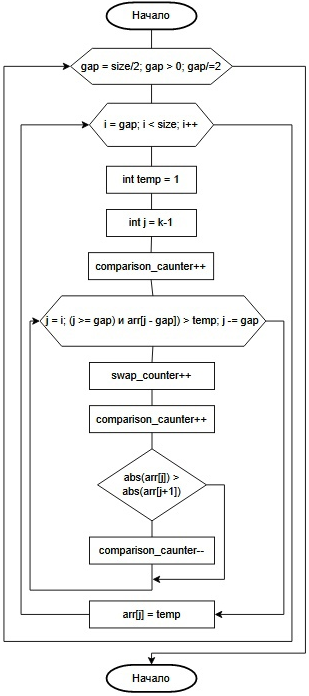
Рисунки 1 и 2 – Блок-схемы алгоритма сортировки пузырьковым методом



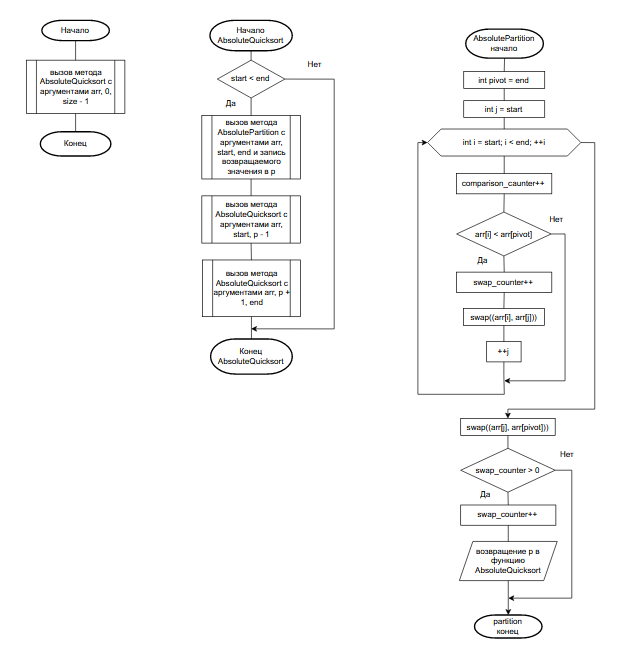
Рисунки 3 и 4 – Блок-схемы алгоритма сортировки методом выбора

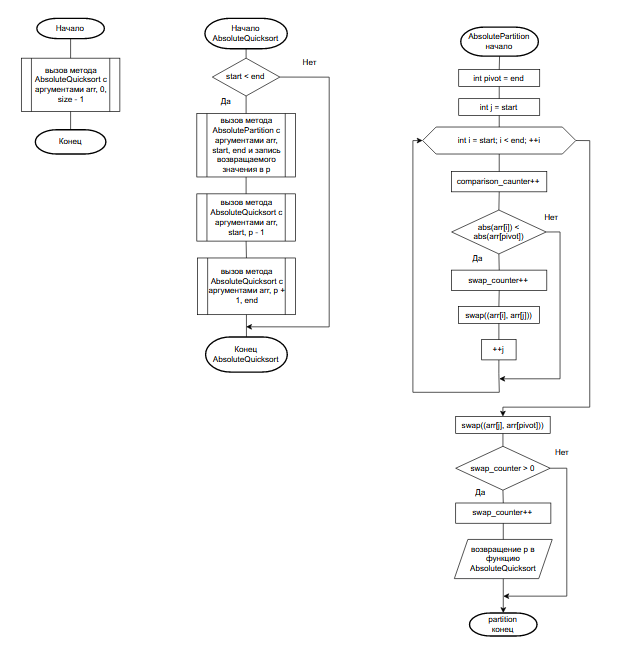


Рисунки 5 и 6 – Блок-схемы алгоритма сортировка методом вставки



Рисунки 7 и 8 – Блок-схемы алгоритма сортировки методом Шелла





Рисунки 9 и 10 – Блок-схемы алгоритма быстрого метода сортировки

## 8 Результаты работы программы

В результате работы программы создаётся неотсортированная матрица, введённая пользователем, и пять матриц, каждая отсортированная одним из 5 способов. Также выводится таблица эффективности каждой из сортировок. Далее пользователь может сохранить все матрицы в файл, и отдельно сохранить в файл отсортированную матрицу с таблицей эффективности каждой сортировки.

На экранных копиях работы программы показаны как корректный ввод данных, так и не корректный.

Экранные копии работы программы:

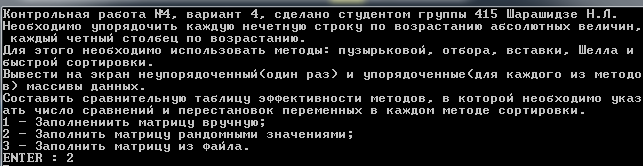
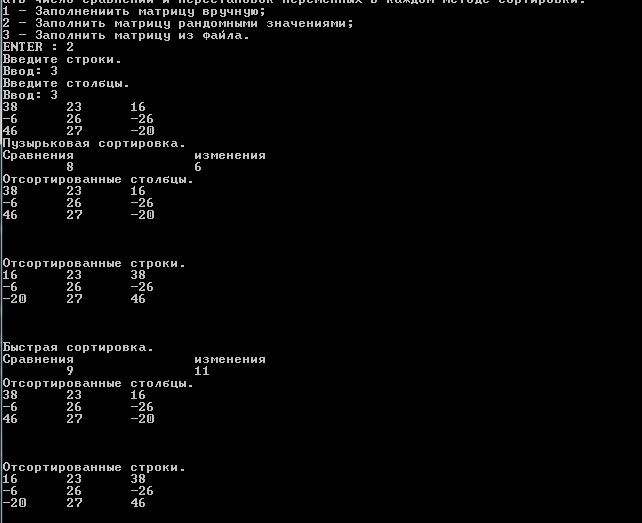


Рисунок 11 – Экранная копия меню программы



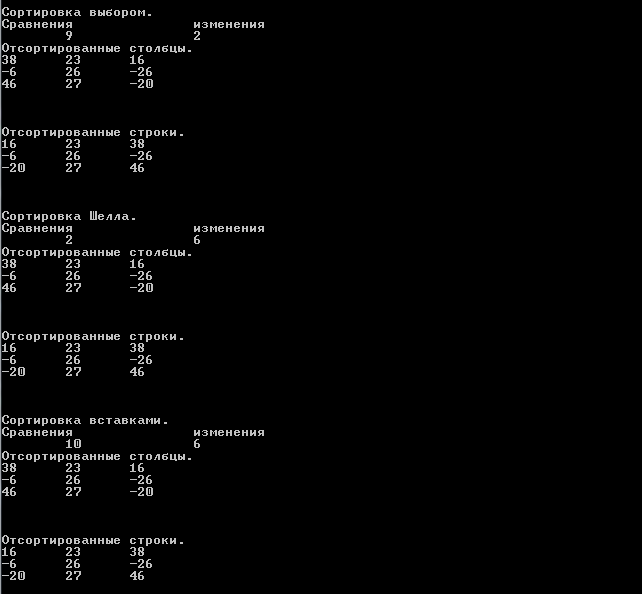


Рисунок 12 – Экранная копия ввода матрицы случайными числами и вывода отсортированной матрицы всеми способами сортировок

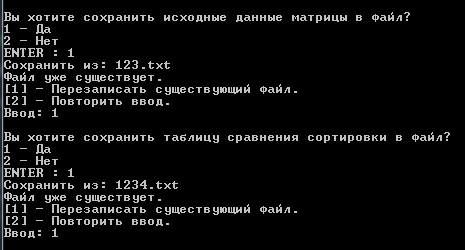
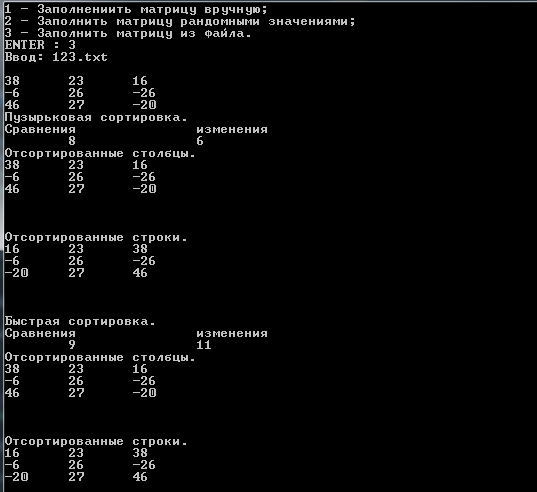


Рисунок 13 – Экранная копия сохранения полученной матрицы и таблицы эффективности в файл



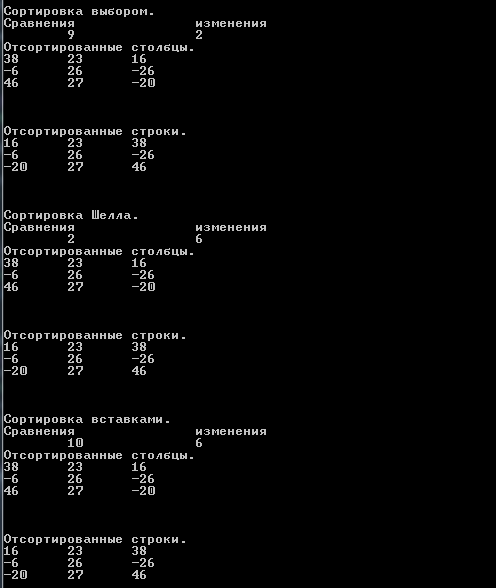


Рисунок 14 – Экранная копия способоа ввода матрицы из файла и вывода отсортированной матрицы всеми способами сортировок

# 9 Описание хода выполнения контрольной работы

1) В ходе контрольной работы было создано решение (Solution) в

интегрированной среде разработки Microsoft Visual Studio C++ 2017. В нём был создан проект.

2) Были нарисованы блок-схемы для понимания и наглядности алгоритма всех сортировок.

3) Было принято решение разделить код программы на модули так,

чтобы в каждом модуле содержался один класс и 1-3 функции, выполняющие одну задачу.

4) При создании системы проверки файловых данных возникли

проблемы с проверкой файла на недопустимое содержание (например, пустой файл при попытке ввода данных из него) или на недопустимое имя. Для решения проблемы была изучена и впоследствии подключена библиотека filesystem, а также изменена версия С++ с 14 года на 17 год.

## 10 Исходный текст программы

**[ Начало Main.cpp--- ]**

#include <windows.h>

#include "ISort.h"

#include "BubbleSort.h"

#include "InsertionSort.h"

#include "QuickSort.h"

#include "ShellSort.h"

#include "SelectionSort.h"

#include "MatrixInput.h"

#include "InitialMatrixFileOutput.h"

#include "SaveData.h"

enum class Answer

{

Yes = 1,

No

};

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

setlocale(LC\_ALL, "Russian");

int const sorts\_count = 5;

std::cout << "Контрольная работа №4, вариант 4, сделано студентом группы 415 Шарашидзе Н.Л." << std::endl

<< "Необходимо упорядочить каждую нечетную строку по возрастанию абсолютных величин, каждый четный столбец по возрастанию." << std::endl

<< "Для этого необходимо использовать методы: пузырьковой, отбора, вставки, Шелла и быстрой сортировки." << std::endl

<< "Вывести на экран неупорядоченный(один раз) и упорядоченные(для каждого из методов) массивы данных." << std::endl

<< "Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки." << std::endl;

while (true)

{

std::vector <std::shared\_ptr<ISort>> sorts\_case;

std::vector <std::shared\_ptr<SortedMatrix>> sorts\_matrix;

std::vector <std::vector<int>> matrix;

sorts\_case.emplace\_back(std::make\_shared<BubbleSort>());

sorts\_case.emplace\_back(std::make\_shared<QuickSort>());

sorts\_case.emplace\_back(std::make\_shared<SelectionSort>());

sorts\_case.emplace\_back(std::make\_shared<ShellSort>());

sorts\_case.emplace\_back(std::make\_shared<InsertionSort>());

InputMode mode = static\_cast<InputMode> (MatrixInput(matrix));

for (int i = 0; i < static\_cast<int>(matrix.size()); i++)

{

for (int j = 0; j < static\_cast<int>(matrix[0].size()); j++)

{

std::cout << matrix[i][j] << "\t";

}

std::cout << std::endl;

}

for (int i = 0; i < sorts\_count; i++)

{

sorts\_matrix.emplace\_back(std::make\_shared<SortedMatrix>(matrix, sorts\_case[i]));

sorts\_case[i]->PrintCounters();

sorts\_matrix[i]->Show();

}

if (mode == InputMode::console\_by\_hand || mode == InputMode::console\_random)

{

std::cout << "Вы хотите сохранить исходные данные матрицы в файл?" << std::endl;

std::cout << "1 - Да\n2 - Нет" << std::endl;

Answer initial\_choice = static\_cast<Answer> (InputCheck::CheckMenu(2));

if (initial\_choice == Answer::Yes)

InitialMatrixFileOutput(matrix);

}

std::cout << "Вы хотите сохранить таблицу сравнения сортировки в файл?" << std::endl;

std::cout << "1 - Да\n2 - Нет" << std::endl;

Answer data\_choice = static\_cast<Answer> (InputCheck::CheckMenu(2));

if(data\_choice == Answer::Yes)

SaveData(sorts\_case, sorts\_matrix);

std::cout << "Продолжить?" << std::endl;

std::cout << "1 - Да\n2 - Нет" << std::endl;

Answer continue\_choice = static\_cast<Answer> (InputCheck::CheckMenu(2));

if (continue\_choice == Answer::No) return 0;

system("cls");

}

}

**[---Конец Main.cpp]**

**[ Начало BubbleSort.h--- ]**

#pragma once

#include "ISort.h"

class BubbleSort : public ISort

{

public:

void PrintCounters() override;

void Sort(std::vector<int>& arr, int size) override;

void AbsoluteSort(std::vector<int>& arr, int size) override;

};

**[---Конец BubbleSort.h]**

**[ Начало BubbleSort.cpp--- ]**

#include "BubbleSort.h"

void BubbleSort::PrintCounters()

{

std::cout << "Пузырьковая сортировка." << std::endl;

ISort::PrintCounters();

}

void BubbleSort::AbsoluteSort(std::vector<int>& arr, int size)

{

bool end = true;

for (int i = 0; i < size - 1; i++)

{

for (int j = 0; j < size - i - 1; j++)

{

comparison\_counter++;

if (abs(arr[j]) > abs(arr[j + 1]))

{

swap\_counter++;

std::swap(arr[j], arr[j + 1]);

end = false;

}

}

if (end)

return;

}

}

void BubbleSort::Sort(std::vector<int>& arr, int size)

{

bool end = true;

for (int i = 0; i < size - 1; i++)

{

for (int j = 0; j < size - i - 1; j++)

{

comparison\_counter++;

if (arr[j] > arr[j + 1])

{

swap\_counter++;

std::swap(arr[j], arr[j + 1]);

end = false;

}

}

if (end)

return;

}

}

**[---Конец BubbleSort.cpp]**

**[ Начало Check.h--- ]**

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include <cstdlib>

int CheckMenu(int items);

**[---Конец Check.h]**

**[ Начало Check.cpp--- ]**

#include "Check.h"

int CheckMenu(int items)

{

std::string value\_input;

int value;

while (true)

{

std::cout << "Ввод: ";

std::getline(std::cin, value\_input);

std::cout << std::endl;

try

{

value = stoi(value\_input);

if (value <= items && value >= 1)

return value;

else

throw value;

}

catch (std::exception&)

{

std::cout << "Не существующий пункт меню. Повторите ввод." << std::endl;

}

catch (int ex\_value)

{

std::cout << ex\_value << "Не существующий пункт меню. Повторите ввод." << std::endl;

}

}

}

**[---Конец Check.cpp]**

**[ Начало GetColum.h--- ]**

#pragma once

#include <iostream>

#include <vector>

std::vector<int> GetColum(std::vector<std::vector<int>> matrix, int colum, int lines);

void SetColum(std::vector<std::vector<int>>& matrix, std::vector<int> colum, int column\_number, int lines);

**[---Конец GetColum.h]**

**[ Начало GetColum.cpp --- ]**

#include "GetColum.h"

std::vector<int> GetColum(std::vector<std::vector<int>> matrix, int colum, int lines)

{

std::vector<int> tempColum;

for (int i = 0; i < lines; i++)

{

tempColum.push\_back(matrix[i][colum]);

}

return tempColum;

}

void SetColum(std::vector<std::vector<int>>& matrix, std::vector<int> colum, int columNumber, int lines)

{

for (int i = 0; i < lines; i++)

{

matrix[i][columNumber] = colum[i];

}

}

**[---Конец GetColum.cpp]**

**[ Начало GetFileName.h--- ]**

#pragma once

#include <iostream>

#include <fstream>

#include <string>

#include <experimental/filesystem>

#include <filesystem>

#include "Check.h"

#define WIN\_NAME\_SIZE 22

using namespace std::experimental::filesystem;

enum

{

output = 0,

input = 1

};

std::string GetFileName(int stream\_type);

**[---Конец GetFileName.h]**

**[ Начало GetFileName.cpp--- ]**

#include "GetFileName.h"

std::string FileInput()

{

while (true)

{

std::string file\_path;

std::cout << "Ввод: ";

getline(std::cin, file\_path);

if (file\_path.find(".txt") >= std::string::npos)

{

std::cout << "Не текстовый файл. Повторите ввод." << std::endl;

return false;

}

if (!std::ifstream(file\_path)) {

std::cout << "Файл не существует. Повторите ввод." << std::endl;

continue;

}

std::error\_code ec;

if (!is\_regular\_file(file\_path, ec))

{

std::cout << "Адрес содержит недопустимые значения. Повторите ввод." << std::endl;

continue;

}

std::ifstream myFile(file\_path);

if (!myFile) {

std::cout << "Загрузка запрещена. Повторите ввод." << std::endl;

myFile.close();

continue;

}

myFile.close();

return file\_path;

}

}

std::string FileOutput()

{

const int get\_name = 2;

while (true)

{

std::string file\_path;

std::cout << "Сохранить из: ";

getline(std::cin, file\_path);

if (file\_path.find(".txt") >= std::string::npos)

{

std::cout << "Не текстовый файл. Повторите ввод." << std::endl;

return false;

}

if (std::ifstream(file\_path))

{

std::cout << "Файл уже существует." << std::endl;

std::cout << "[1] - Перезаписать существующий файл." << std::endl;

std::cout << "[2] - Повторить ввод." << std::endl;

int tryAnotherFile = CheckMenu(2);

if (tryAnotherFile == get\_name)

{

continue;

}

}

std::ofstream myFile(file\_path, std::ofstream::app);

std::error\_code ec;

if (!is\_regular\_file(file\_path, ec))

{

std::cout << "Адрес содержит недопустимые значения. Повторите ввод." << std::endl;

continue;

}

if (!myFile)

{

std::cout << "Запись запрещена. Повторите ввод." << std::endl;

myFile.close();

continue;

}

myFile.close();

return file\_path;

}

}

std::string GetFileName(int stream\_type)

{

while (true) {

std::string file\_path;

switch (stream\_type)

{

case input:

return file\_path = FileInput();

case output:

return file\_path = FileOutput();

}

}

}

**[---Конец GetFileName.cpp]**

**[ Начало InitialMatrixFileOutput.h--- ]**

#pragma once

#include "SortedMatrix.h"

#include <fstream>

#include "InputCheck.h"

#include "GetFileName.h"

void InitialMatrixFileOutput(std::vector<std::vector<int>> matrix);

**[---Конец InitialMatrixFileOutput.h]**

**[ Начало InitialMatrixFileOutput.cpp --- ]**

#include "InitialMatrixFileOutput.h"

void InitialMatrixFileOutput(std::vector<std::vector<int>> matrix)

{

const int this\_file = 1;

std::string name;

std::ofstream file;

file.exceptions(std::ofstream::badbit | std::ofstream::failbit);

while (true)

{

try

{

name = GetFileName(output);

file.open(name);

}

catch (const std::exception&)

{

std::cout << "Ошибка доступа." << std::endl;

continue;

}

break;

}

int number\_of\_lines = static\_cast<int> (matrix.size());

int number\_of\_colums = static\_cast<int>(matrix[0].size());

file << number\_of\_lines << std::endl;

file << number\_of\_colums << std::endl;

for (int i = 0; i < number\_of\_lines; i++)

{

for (int j = 0; j < number\_of\_colums; j++)

{

file << matrix[i][j] << std::endl;

}

}

file.close();

}

**[---Конец InitialMatrixFileOutput.cpp]**

**[ Начало InputCheck.h--- ]**

#pragma once

#include <iostream>

#include <string>

#include <fstream>

#include <cstdlib>

class InputCheck

{

private:

static std::string value\_input;

static int value;

public:

static int CheckIntValue();

static int CheckIntValuePlus();

static int CheckMenu(int items);

static int CheckLineInt(std::ifstream& file);

static int CheckLineUnsInt(std::ifstream& file);

};

**[---Конец InputCheck.h]**

**[ Начало InputCheck.cpp--- ]**

#include "InputCheck.h"

int InputCheck::value = 0;

std::string InputCheck::value\_input = "";

int InputCheck::CheckIntValue()

{

while (true)

{

std::cout << "Ввод: ";

std::getline(std::cin, value\_input);

try

{

value = stoi(value\_input);

return value;

}

catch (std::exception&)

{

std::cout << "Ошибка. Попробуйте снова." << std::endl;

}

}

}

int InputCheck::CheckIntValuePlus()

{

while (true)

{

std::cout << "Ввод: ";

std::getline(std::cin, value\_input);

try

{

value = stoi(value\_input);

if (value <= 0)

throw std::exception("NegativeInt");

else

return value;

}

catch (std::exception&)

{

std::cout << "Ошибка. Попробуйте снова." << std::endl;

}

}

}

int InputCheck::CheckMenu(int items)

{

while (true)

{

std::cout << "ENTER : ";

std::getline(std::cin, value\_input);

try

{

value = stoi(value\_input);

if (value <= items && value >= 1)

return value;

else

throw value;

}

catch (std::exception&)

{

std::cout << "Пункт меню не существует. Повторите ввод." << std::endl;

}

catch (int ex\_value)

{

std::cout << ex\_value << "Пункт меню не существует. Повторите ввод." << std::endl;

}

}

}

int InputCheck::CheckLineInt(std::ifstream& file)

{

std::string temp\_v = "";

int temp\_i = 0;

try

{

std::getline(file, temp\_v);

}

catch (const std::exception&)

{

std::cout << "Недостаточно данных в файле или неправильная форма заполнения файла." << std::endl;

throw std::exception("Wrong data");

}

try

{

temp\_i = stoi(temp\_v);

return temp\_i;

}

catch (const std::exception&)

{

std::cout << "Некорректные данные." << std::endl;

throw std::exception("Wrong data");

}

}

int InputCheck::CheckLineUnsInt(std::ifstream& file)

{

std::string temp\_v = "";

int temp\_i = 0;

try

{

std::getline(file, temp\_v);

}

catch (const std::exception&)

{

std::cout << "Недостаточно данных в файле или неправильная форма заполнения файла. " << std::endl;

throw std::exception("Wrong data");

}

try

{

temp\_i = stoi(temp\_v);

if (temp\_i < 0)

{

throw std::exception("Wrong data");

}

else

return temp\_i;

}

catch (const std::exception&)

{

std::cout << "Некорректные данные." << std::endl;

throw std::exception("Wrong data");

}

}

**[---Конец InputCheck.cpp]**

**[ Начало InsertionSort.h--- ]**

#pragma once

#include "ISort.h"

class InsertionSort : public ISort

{

public:

void PrintCounters() override;

void Sort(std::vector<int>& arr, int size) override;

void AbsoluteSort(std::vector<int>& arr, int size) override;

};

**[---Конец InsertionSort.h]**

**[ Начало InsertionSort.cpp--- ]**

#include "InsertionSort.h"

void InsertionSort::PrintCounters()

{

std::cout << "Сортировка вставками." << std::endl;

ISort::PrintCounters();

}

void InsertionSort::AbsoluteSort(std::vector<int>& arr, int size)

{

for (int k = 1; k < size; k++)

{

int temp = arr[k];

int j = k - 1;

comparison\_counter++;

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

{

comparison\_counter++;

swap\_counter++;

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

j--;

if(j==0)

comparison\_counter--;

}

arr[j + 1] = temp;

}

}

void InsertionSort::Sort(std::vector<int>& arr, int size)

{

for (int k = 1; k < size; k++)

{

int temp = arr[k];

int j = k - 1;

comparison\_counter++;

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

{

comparison\_counter++;

swap\_counter++;

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

j--;

if (j == 0)

comparison\_counter--;

}

arr[j + 1] = temp;

}

}

**[---Конец InsertionSort.cpp]**

**[ Начало ISort.h--- ]**

#pragma once

#include <string>

#include <iostream>

#include <vector>

#include "Names.h"

class ISort

{

protected:

int comparison\_counter = 0;

int swap\_counter = 0;

public:

int GetComprasionCounter();

int GetSwapCountrt();

virtual void PrintCounters();

virtual void AbsoluteSort(std::vector<int>& arr, int size) = 0;

virtual void Sort(std::vector<int>& arr, int size) = 0;

virtual ~ISort() = default;

};

**[---Конец ISort.h]**

**[ Начало ISort.cpp--- ]**

#include "ISort.h"

#include "Names.h"

int ISort::GetComprasionCounter()

{

return comparison\_counter;

}

int ISort::GetSwapCountrt()

{

return swap\_counter;

}

void ISort::PrintCounters()

{

std::cout << "Сравнения" << "\t\t" << "изменения" << std::endl;

std::cout << "\t" << comparison\_counter << "\t\t" << swap\_counter << std::endl;

}

**[---Конец ISort.cpp]**

**[ Начало MatrixInput.h--- ]**

#pragma once

#include "SortedMatrix.h"

#include "InputCheck.h"

#include "GetFileName.h"

int MatrixInput(std::vector<std::vector<int>>& matrix);

std::vector<std::vector<int>> ConsoleInput();

std::vector<std::vector<int>> FileInput();

std::vector<std::vector<int>> RandomInput();

**[---Конец MatrixInput.h]**

**[ Начало MatrixInput.cpp--- ]**

#include "MatrixInput.h"

#include <time.h>

int MatrixInput(std::vector<std::vector<int>>& matrix)

{

std::cout << "1 - Заполнениить матрицу вручную;" << std::endl

<< "2 - Заполнить матрицу рандомными значениями;" << std::endl

<< "3 - Заполнить матрицу из файла." << std::endl;

InputMode mode = static\_cast<InputMode>(InputCheck::CheckMenu(3));

switch (mode)

{

case InputMode::console\_by\_hand:

matrix = ConsoleInput();

break;

case InputMode::console\_random:

matrix = RandomInput();

break;

case InputMode::io\_file:

matrix = FileInput();

break;

}

return static\_cast<int>(mode);

}

std::vector<std::vector<int>> ConsoleInput()

{

std::cout << "Введите количество строк." << std::endl;;

int lines = InputCheck::CheckIntValuePlus();

std::cout << "Введите количество столбцов." << std::endl;;

int colums = InputCheck::CheckIntValuePlus();

std::vector<std::vector<int>> matrix;

for (int i = 0; i < lines; i++)

{

matrix.push\_back(std::vector<int>{});

for (int j = 0; j < colums; j++)

{

std::cout << "Ввод [" << i + 1 << "] [" << j + 1 << "] ." << std::endl;

matrix[i].push\_back(InputCheck::CheckIntValue());

}

}

return matrix;

}

std::vector<std::vector<int>> FileInput()

{

std::string name = "";

std::ifstream file;

std::vector<std::vector<int>> matrix;

file.exceptions(std::ifstream::badbit | std::ifstream::failbit);

while (true)

{

try

{

name = GetFileName(input);

file.open(name);

}

catch (const std::exception&)

{

std::cout << "Ошибка доступа." << std::endl;

continue;

}

try

{

int lines = (InputCheck::CheckLineUnsInt(file));

int colums = (InputCheck::CheckLineUnsInt(file));

for (int i = 0; i < lines; i++)

{

matrix.push\_back(std::vector<int>{});

for (int j = 0; j < colums; j++)

{

matrix[i].push\_back( InputCheck::CheckLineInt(file));

}

}

}

catch (std::exception&)

{

file.close();

continue;

}

std::cin.clear();

std::cin.ignore();

file.close();

return matrix;

}

}

std::vector<std::vector<int>> RandomInput()

{

srand(static\_cast<unsigned>(time(nullptr)));

std::cout << "Введите строки." << std::endl;;

int lines = InputCheck::CheckIntValuePlus();

std::cout << "Введите столбцы." << std::endl;;

int colums = InputCheck::CheckIntValuePlus();

std::vector<std::vector<int>> matrix;

for (int i = 0; i < lines; i++)

{

matrix.push\_back(std::vector<int>{});

for (int j = 0; j < colums; j++)

{

matrix[i].push\_back(std::rand() % 101 - 50);

}

}

return matrix;

}

**[---Конец MatrixInput.cpp]**

**[ Начало QuickSort.h--- ]**

#pragma once

#include "ISort.h"

class QuickSort : public ISort

{

public:

void PrintCounters() override;

void Sort(std::vector<int>& arr, int size) override;

void AbsoluteSort(std::vector<int>& arr, int size) override;

void Quicksort(std::vector<int>& arr, int start, int end);

void AbsoluteQuicksort(std::vector<int>& arr, int start, int end);

int Partition(std::vector<int>& arr, int start, int end);

int AbsolutePartition(std::vector<int>& arr, int start, int end);

};

**[---Конец QuickSort.h]**

**[ Начало QuickSort.cpp--- ]**

#include "QuickSort.h"

void QuickSort::PrintCounters()

{

std::cout << "Быстрая сортировка." << std::endl;

ISort::PrintCounters();

}

void QuickSort::Sort(std::vector<int>& arr, int size)

{

Quicksort(arr, 0, size - 1);

}

void QuickSort::Quicksort(std::vector<int>& arr, int start, int end)

{

if (start < end) {

int p = Partition(arr, start, end);

Quicksort(arr, start, p - 1);

Quicksort(arr, p + 1, end);

}

}

void QuickSort::AbsoluteSort(std::vector<int>& arr, int size)

{

AbsoluteQuicksort(arr, 0, size - 1);

}

void QuickSort::AbsoluteQuicksort(std::vector<int>& arr, int start, int end)

{

if (start < end) {

int p = AbsolutePartition(arr, start, end);

AbsoluteQuicksort(arr, start, p - 1);

AbsoluteQuicksort(arr, p + 1, end);

}

}

int QuickSort::AbsolutePartition(std::vector<int>& arr, int start, int end)

{

int pivot = end;

int j = start;

for (int i = start; i < end; ++i) {

comparison\_counter++;

if (abs(arr[i]) < abs(arr[pivot])) {

swap\_counter++;

std::swap(arr[i], arr[j]);

++j;

}

}

std::swap(arr[j], arr[pivot]);

if (swap\_counter > 0)

swap\_counter++;

return j;

}

int QuickSort::Partition(std::vector<int>& arr, int start, int end)

{

int pivot = end;

int j = start;

for (int i = start; i < end; ++i) {

comparison\_counter++;

if (arr[i] < arr[pivot]) {

swap\_counter++;

std::swap(arr[i], arr[j]);

++j;

}

}

std::swap(arr[j], arr[pivot]);

if (swap\_counter > 0)

swap\_counter++;

return j;

}

**[---Конец QuickSort.cpp]**

**[ Начало SaveData.h--- ]**

#pragma once

#include <vector>

#include <memory>

#include <fstream>

#include "InputCheck.h"

#include "SortedMatrix.h"

#include "GetFileName.h"

#include "ISort.h"

void SaveData(std::vector<std::shared\_ptr<ISort>>& sorts\_case, std::vector < std::shared\_ptr<SortedMatrix>> sorts\_matrix);

**[---Конец SaveData.h]**

**[ Начало SaveData.cpp--- ]**

#include "SaveData.h"

void SaveData(std::vector<std::shared\_ptr<ISort>>& sorts\_case, std::vector < std::shared\_ptr<SortedMatrix>> sorts\_matrix)

{

const int this\_file = 1;

std::string name;

std::ofstream file;

file.exceptions(std::ofstream::badbit | std::ofstream::failbit);

while (true)

{

try

{

name = GetFileName(output);

file.open(name);

}

catch (const std::exception&)

{

std::cout << "Ошибка доступа." << std::endl;

continue;

}

break;

}

std::vector<std::vector<int>> matrix\_colums = sorts\_matrix[0]->GetMatrixColums();

std::vector<std::vector<int>> matrix\_lines = sorts\_matrix[0]->GetMatrixLines();

file<< std::string(15,' ') << "Сравнение" << "\t" << "Изменение" << std::endl;

for (int i = 0; i < sorts\_case.size(); i++)

{

std::string temp\_name = typeid(\*sorts\_case[i]).name();

temp\_name.erase(0, 6);

file << temp\_name << std::string(15 - temp\_name.size(),' ') << sorts\_case[i]->GetComprasionCounter() << "\t\t\t" << sorts\_case[i]->GetSwapCountrt() << std::endl;;

}

file << "\nОтсортированные четные столбцы." << std::endl;

for (int i = 0; i < sorts\_matrix[0]->GetNumberOflines(); i++)

{

for (int j = 0; j < sorts\_matrix[0]->GetNumberOfColums(); j++)

{

file << matrix\_colums[i][j] << "\t";

}

file << std::endl;

}

file << "Отсортированные нечетные строки." << std::endl;

for (int i = 0; i < sorts\_matrix[0]->GetNumberOflines(); i++)

{

for (int j = 0; j < sorts\_matrix[0]->GetNumberOfColums(); j++)

{

file << matrix\_colums[i][j] << "\t";

}

file << std::endl;

}

file.close();

}

**[---Конец SaveData.cpp]**

**[ Начало SelectionSort.h--- ]**

#pragma once

#include "ISort.h"

class SelectionSort : public ISort

{

public:

void PrintCounters() override;

void Sort(std::vector<int>& arr, int size) override;

void AbsoluteSort(std::vector<int>& arr, int size) override;

};

**[---Конец SelectionSort.h]**

**[ Начало SelectionSort.cpp--- ]**

#include "SelectionSort.h"

void SelectionSort::PrintCounters()

{

std::cout << "Сортировка выбором." << std::endl;

ISort::PrintCounters();

}

void SelectionSort::AbsoluteSort(std::vector<int>& arr, int size)

{

for (int i = 0; i < size - 1; i++)

{

int tmp = arr[i];

int tmpid = i;

for (int j = i + 1; j < size; j++)

{

comparison\_counter++;

if (abs(arr[tmpid]) > abs(arr[j]))

{

tmp = arr[j];

tmpid = j;

}

}

if (tmpid != i)

{

swap\_counter++;

std::swap(arr[i], arr[tmpid]);

}

}

}

void SelectionSort::Sort(std::vector<int>& arr, int size)

{

for (int i = 0; i < size - 1; i++)

{

int tmp = arr[i];

int tmpid = i;

for (int j = i + 1; j < size; j++)

{

comparison\_counter++;

if ((arr[tmpid]) > arr[j])

{

tmp = arr[j];

tmpid = j;

}

}

if (tmpid != i)

{

swap\_counter++;

std::swap(arr[i], arr[tmpid]);

}

}

}

**[---Конец SelectionSort.cpp]**

**[ Начало ShellSort.h--- ]**

#pragma once

#include "ISort.h"

class ShellSort : public ISort

{

public:

void PrintCounters() override;

void Sort(std::vector<int>& arr, int size) override;

void AbsoluteSort(std::vector<int>& arr, int size) override;

};

**[---Конец ShellSort.h]**

**[ Начало ShellSort.cpp--- ]**

#include "ShellSort.h"

void ShellSort::PrintCounters()

{

std::cout << "Сортировка Шелла." << std::endl;

ISort::PrintCounters();

}

void ShellSort::AbsoluteSort(std::vector<int>& arr, int size)

{

for (int gap = size/2; gap > 0; gap/=2)

for (int i = gap; i < size; i++)

{

int temp = arr[i];

int j;

comparison\_counter++;

for (j = i; j >= gap && abs(arr[j - gap]) > abs(temp); j -= gap)

{

swap\_counter++;

arr[j] = arr[j - gap];

if (j == gap)

comparison\_counter--;

}

arr[j] = temp;

}

}

void ShellSort::Sort(std::vector<int>& arr, int size)

{

for (int gap = size / 2; gap > 0; gap /= 2)

for (int i = gap; i < size; i++)

{

int temp = arr[i];

int j;

comparison\_counter++;

for (j = i; j >= gap && (arr[j - gap]) > temp; j -= gap)

{

swap\_counter++;

arr[j] = arr[j - gap];

if (j == gap)

comparison\_counter--;

}

arr[j] = temp;

}

}

**[---Конец ShellSort.cpp]**

**[ Начало SortedMatrix.h--- ]**

#pragma once

#include <iostream>

#include <vector>

#include "ISort.h"

#include "GetColum.h"

class SortedMatrix

{

private:

int number\_of\_lines;

int number\_of\_colums;

std::vector<std::vector<int>> matrix\_colums;

std::vector<std::vector<int>> matrix\_lines;

std::vector<std::vector<int>> SortLines(std::vector<std::vector<int>> matrix, std::shared\_ptr<ISort> sort);

std::vector<std::vector<int>> SortClumns(std::vector<std::vector<int>> matrix, std::shared\_ptr<ISort> sort);

public:

SortedMatrix( std::vector<std::vector <int>> matrix\_value,std::shared\_ptr<ISort> sort)

{

number\_of\_lines = static\_cast<int>(matrix\_value.size());

number\_of\_colums = static\_cast<int>(matrix\_value[0].size());

matrix\_colums = SortLines(matrix\_value, sort);

matrix\_lines = SortClumns(matrix\_value, sort);

}

int GetNumberOflines();

int GetNumberOfColums();

std::vector<std::vector<int>> GetMatrixColums();

std::vector<std::vector<int>> GetMatrixLines();

void Show();

};

**[---Конец SortedMatrix.h]**

**[ Начало SortedMatrix.cpp--- ]**

#include "SortedMatrix.h"

std::vector<std::vector<int>> SortedMatrix::SortLines(std::vector<std::vector<int>> matrix, std::shared\_ptr<ISort> sort)

{

std::vector<int> temp\_column;

for (int i = 1; i < number\_of\_colums; i += 2)

{

temp\_column = GetColum(matrix, i, number\_of\_lines);

sort -> Sort(temp\_column, number\_of\_lines);

SetColum(matrix, temp\_column, i, number\_of\_lines);

}

return matrix;

}

std::vector<std::vector<int>> SortedMatrix::SortClumns(std::vector<std::vector<int>> matrix, std::shared\_ptr<ISort> sort)

{

for (int i = 0; i < number\_of\_lines; i += 2)

{

sort -> AbsoluteSort(matrix[i], number\_of\_colums);

}

return matrix;

}

int SortedMatrix::GetNumberOflines()

{

return number\_of\_lines;

}

int SortedMatrix::GetNumberOfColums()

{

return number\_of\_colums;

}

std::vector<std::vector<int>> SortedMatrix::GetMatrixColums()

{

return matrix\_colums;

}

std::vector<std::vector<int>> SortedMatrix::GetMatrixLines()

{

return matrix\_lines;

}

void SortedMatrix::Show()

{

std::cout << "Отсортированные столбцы." << std::endl;

for (int i = 0; i < number\_of\_lines; i++)

{

for (int j = 0; j < number\_of\_colums; j++)

{

std::cout << matrix\_colums[i][j] << "\t";

}

std::cout << std::endl;

}

std::cout << "\n\n" << std::endl;

std::cout << "Отсортированные строки." << std::endl;

for (int i = 0; i < number\_of\_lines; i++)

{

for (int j = 0; j < number\_of\_colums; j++)

{

std::cout << matrix\_lines[i][j] << "\t";

}

std::cout << std::endl;

}

std::cout << "\n\n" << std::endl;

}

**[---Конец SortedMatrix.cpp]**

**[ Начало UnitTest.cpp--- ]**

#include "pch.h"

#include "CppUnitTest.h"

#include "../Project/ISort.h"

#include "../Project/ISort.cpp"

#include "../Project/BubbleSort.h"

#include "../Project/BubbleSort.cpp"

#include "../Project/InsertionSort.h"

#include "../Project/InsertionSort.cpp"

#include "../Project/SelectionSort.h"

#include "../Project/SelectionSort.cpp"

#include "../Project/ShellSort.h"

#include "../Project/ShellSort.cpp"

#include "../Project/QuickSort.h"

#include "../Project/QuickSort.cpp"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace UnitTest

{

TEST\_CLASS(UnitTest)

{

public:

private:

BubbleSort bubleSort;

InsertionSort insertionSort;

SelectionSort selectionSort;

ShellSort shellSort;

QuickSort quickSort;

std::vector <int> result = { 1,2,3,4,5 };

std::vector <int> start = { 5,4,1,2,3 };

std::vector <int> result\_abs = { -1,2,-3,4,-5 };

std::vector <int> start\_abs = { -5,4,-1,2,-3 };

public:

TEST\_METHOD(BubleTest)

{

std::vector <int> temp = start;

bubleSort.Sort(temp, static\_cast<int>(temp.size()));

Assert::IsTrue(temp == result);

std::vector <int> temp\_abs = start\_abs;

bubleSort.AbsoluteSort(temp\_abs, static\_cast<int>(temp\_abs.size()));

Assert::IsTrue(temp\_abs == result\_abs);

}

TEST\_METHOD(InsertionTest)

{

std::vector <int> temp = start;

insertionSort.Sort(temp, static\_cast<int>(temp.size()));

Assert::IsTrue(temp == result);

std::vector <int> temp\_abs = start\_abs;

insertionSort.AbsoluteSort(temp\_abs, static\_cast<int>(temp\_abs.size()));

Assert::IsTrue(temp\_abs == result\_abs);

}

TEST\_METHOD(SelectionTest)

{

std::vector <int> temp = start;

selectionSort.Sort(temp, static\_cast<int>(temp.size()));

Assert::IsTrue(temp == result);

std::vector <int> temp\_abs = start\_abs;

selectionSort.AbsoluteSort(temp\_abs, static\_cast<int>(temp\_abs.size()));

Assert::IsTrue(temp\_abs == result\_abs);

}

TEST\_METHOD(ShellTest)

{

std::vector <int> temp = start;

shellSort.Sort(temp, static\_cast<int>(temp.size()));

Assert::IsTrue(temp == result);

std::vector <int> temp\_abs = start\_abs;

shellSort.AbsoluteSort(temp\_abs, static\_cast<int>(temp\_abs.size()));

Assert::IsTrue(temp\_abs == result\_abs);

}

TEST\_METHOD(QuickTest)

{

std::vector <int> temp = start;

quickSort.Sort(temp, static\_cast<int>(temp.size()));

Assert::IsTrue(temp == result);

std::vector <int> temp\_abs = start\_abs;

quickSort.AbsoluteSort(temp\_abs, static\_cast<int>(temp\_abs.size()));

Assert::IsTrue(temp\_abs == result\_abs);

}

};

};

**[---Конец UnitTest.cpp]**