Изображение выглядит как цепь, медальон

Автоматически созданное описание

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 | | 09.03.01 | Информатика и вычислительная техника | | |
| Направленность (профиль) | |  | Системы автоматизированного проектирования | | |
| Форма обучения | |  | очная | | |
|  | |  |  | | |
| Факультет | |  | Информационных технологий и управления | | |
| Кафедра | |  | Систем автоматизированного проектирования и управления | | |
| Учебная дисциплина | |  | Программирование | | |
| Курс | I | | | Группа | 414 |

Отчёт по контрольной работе № 3

Вариант № 10

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Исполнитель: |  |  |  |  |
| обучающийся группы 414 |  |  |  | Кондиляброва  Вероника Данииловна |
|  |  | (дата, подпись) |  |  |
|  |  |  |  |  |
| Проверили: |  |  |  | Корниенко Иван Григорьевич |
|  |  | (дата, подпись) |  | Федин Алексей Константинович |

Содержание

[1 Постановка задачи 3](#_Toc114324151)

[2 Исходные данные 3](#_Toc114324152)

[3 Особые ситуации 3](#_Toc114324153)

[4 Математические методы и алгоритмы решения задач 3](#_Toc114324154)

[5 Форматы представления данных 5](#_Toc114324155)

[6 Структура программы 8](#_Toc114324156)

[7 Описание хода выполнения 8](#_Toc114324157)

[8 Блок-схема алгоритма решения задачи 9](#_Toc114324158)

[9 Результаты работы программы 16](#_Toc114324159)

[10 Исходный код полученного программного решения 25](#_Toc114324160)

## 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. Неупорядоченная матрица из N строк и M столбцов задается и заполняется один раз (с клавиатуры, из файла или случайными числами), далее она используется для каждого из методов сортировки. Реализовать абстрактный базовый класс ISort, содержащий чистый виртуальный метод Sort и необходимые счетчики, от которого наследовать подклассы для реализации сортировок.

Упорядочить каждый столбец матрицы по возрастанию.

## 2 Исходные данные

В качестве исходных данных программа использует количество строк N и столбцов M, ячейки матрицы. Ввод с клавиатуры, из файла или заполнение случайными числами.

## 3 Особые ситуации

1 Пользователь может ввести количество столбцов или строк меньше одного, устранено с помощью функции GetInt, осуществляющей проверку ввода;

2 Отсутствие ожидаемых программой файлов устранено методом open. Если программа не находит файл с введенным пользователем именем, этот метод сам его создает.

## 4 Математические методы и алгоритмы решения задач

Алгоритмы для сортировки

Сортировка пузырьковым методом с помощью двойного цикла for:

1. Прохождение по строчкам каждого столбика;
2. Сравнивание между собой пар соседних ячеек одного столбика;
3. Если при сравнении оказывается, что значение ячейки i больше, чем значение ячейки i+1, то мы меняем значения этих ячеек местами;
4. Переходим к другому столбику

Сортировка методом отбора:

* 1. Прохождение по столбику;
  2. Поиск минимального элемента;
  3. Перемещение минимального элемента в отсортированную часть;
  4. Переход к следующему столбику

Сортировка методом вставки:

1. Прохождение по столбику;
2. Сравнение каждого элемента столбика с элементами из отсортированной части;
3. Замена сравнивающихся элементов, если столбик не отсортирован
4. Переход к следующему столбику

Сортировка Шелла:

1. Прохождение по столбику;
2. Сравнение элементов, находящихся на расстоянии N, N-1, N-2 и так далее, пока расстояние не станет равным единице;
3. Обмен элементов на расстоянии;
4. Переход к следующему столбику

Быстрая сортировка:

Алгоритм работы пользовательского интерфейса

1. Цикл while, отвечающий за выход из программы;
2. Оператор множественного выбора switch – case осуществляет остальные действия в работе с пользователем

## 5 Форматы представления данных

Таблица 1 – Переменные

|  |  |  |
| --- | --- | --- |
| Имя | Тип | Описание |
| matrix | vector <vector<int>> | Матрица чисел |
| user\_choice | int | Пользовательский выбор в главном меню |
| exit | int | Выход из меню |
| n | int | Количество строк |
| m | int | Количество столбцов |

Базовый класс CISort

Таблица 2 – Поля

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Тип |
| N | protected | int |
| M | protected | int |
| matrix | protected | vector <vector<int>> |
| Count\_comparison | protected | int |
| Count\_permutation | protected | int |

Таблица 3 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| Sort | public | Виртуальный метод для сортировок |
| Show | public | Передает каждое поле |
| GetCountComparisons | public | Вывод количества сравнений |
| GetCountPermutations | public | Вывод количества перестановок |

Таблица 4 – Конструкторы

|  |  |  |
| --- | --- | --- |
| Имя | Принимаемые значения | Описание |
| CISort | - | По умолчанию |
| CISort | vector<vector<int>> Matrix, int N, int M | Копирования |

Производный класс CBubbleSort

Таблица 5 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| Sort | public | Виртуальный метод для сортировок |

Таблица 6 – Конструкторы

|  |  |  |
| --- | --- | --- |
| Имя | Принимаемые значения | Описание |
| CBubbleSort | - | По умолчанию |
| CBubbleSort | vector<vector<int>> Matrix, int N, int M | Копирования |

Производный класс CSelectionSort

Таблица 7 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| Sort | public | Виртуальный метод для сортировок |

Таблица 8 – Конструкторы:

|  |  |  |
| --- | --- | --- |
| Имя | Принимаемые значения | Описание |
| CSelectionSort | - | По умолчанию |
| CSelectionSort | vector<vector<int>> Matrix, int N, int M | Копирования |

Производный класс CInsertionSort

Таблица 9 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| Sort | public | Виртуальный метод для сортировок |

Таблица 10 – Конструкторы

|  |  |  |
| --- | --- | --- |
| Имя | Принимаемые значения | Описание |
| CInsertionSort | - | По умолчанию |
| CInsertionSort | vector<vector<int>> Matrix, int N, int M | Копирования |

Производный класс CShellSort

Таблица 11 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| Sort | public | Виртуальный метод для сортировок |

Таблица 12 – Конструкторы

|  |  |  |
| --- | --- | --- |
| Имя | Принимаемые значения | Описание |
| CShellSort | - | По умолчанию |
| CShellSort | vector<vector<int>> Matrix, int N, int M | Копирования |

Производный класс CQuickSort

Таблица 13 – Методы

|  |  |  |
| --- | --- | --- |
| Имя | Модификатор | Описание |
| Sort | public | Виртуальный метод для сортировок |

Таблица 14 – Конструкторы:

|  |  |  |
| --- | --- | --- |
| Имя | Принимаемые значения | Описание |
| CQuickSort | - | По умолчанию |
| CQuickSort | vector<vector<int>> Matrix, int N, int M | Копирования |

## 6 Структура программы

Таблица 15 – Структура программы

|  |  |  |
| --- | --- | --- |
| Модуль | Функция | Описание |
| Correct\_input | GetInt | Проверка ввода количества строк и столбцов |
|  | GetInputMatrix | Проверка ввода выбора меню для ввода матрицы |
| Function | MannualInputMatrix | Ввод матрицы с клавиатуры |
|  | RandomInputMatrix | Заполнение матрицы случайными числами |
|  | OfFileMatrix | Заполнение матрицы из файла |
|  | LookMatrix | Просмотр матрицы |
| Menu | MenuMain | Осуществляет работу пользовательского интерфейса |
| Main | main | Вызывает функцию MenuMain, с которой начинается выполнение программы |
| Test | Test | Тестирует работу программы |

## 7 Описание хода выполнения

При выполнении контрольной работы я:

* Узнала про виртуальные функции;
* Работала с абстрактным классом;
* Изучала работу с двумерным вектором;
* Училась заполнять матрицу случайными числами;

Решенные возникшие проблемы:

* Заполнить матрицу случайными числами – функция rand, функция srand, функция time;
* Создание двумерного вектора – во внешнем цикле for создаем вектор temp, во внутреннем заполняем его, во внешнем добавляем вектор temp в вектор Matrix;

Непонятные моменты:

* Быстрая сортировка рекурсией;

## 8 Блок-схема алгоритма решения задачи

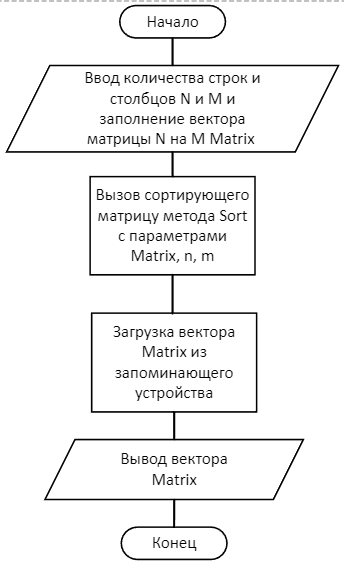


Рисунок 1 – Блок-схема функции алгоритма решения

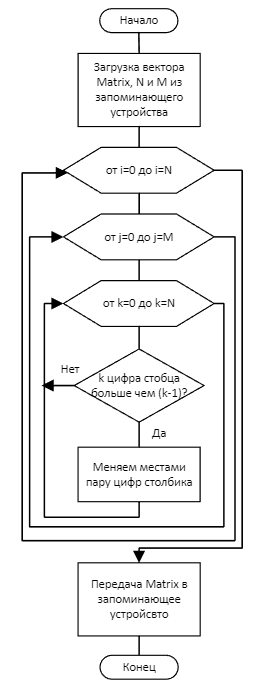


Рисунок 2 – Блок-схема сортировки пузырьком

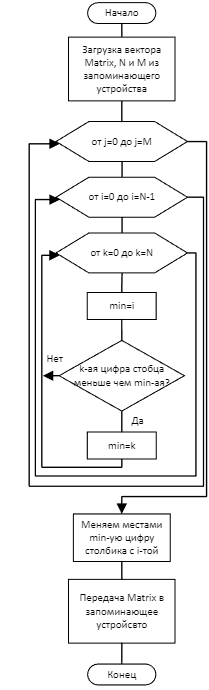


Рисунок 3 – Блок-схема сортировки отбором

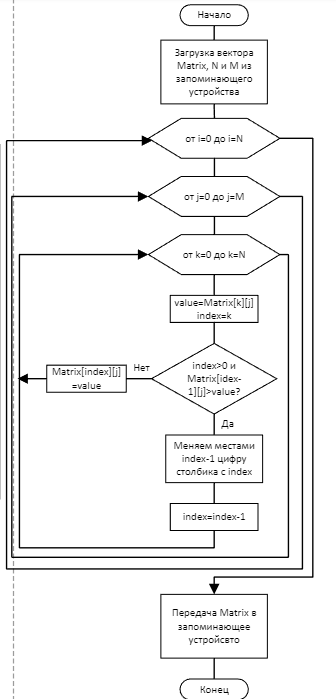


Рисунок 4 – Блок-схема сортировки вставки

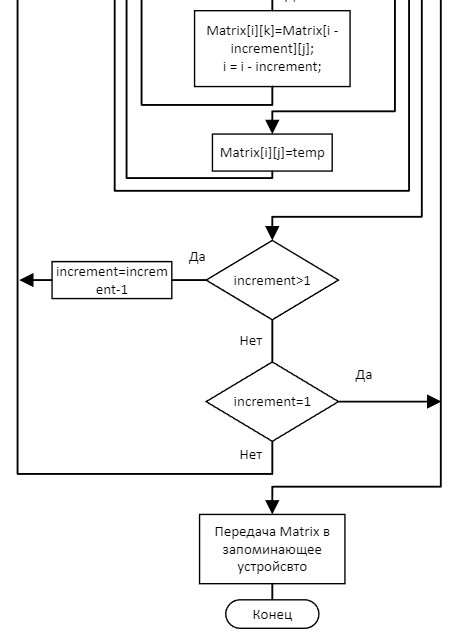
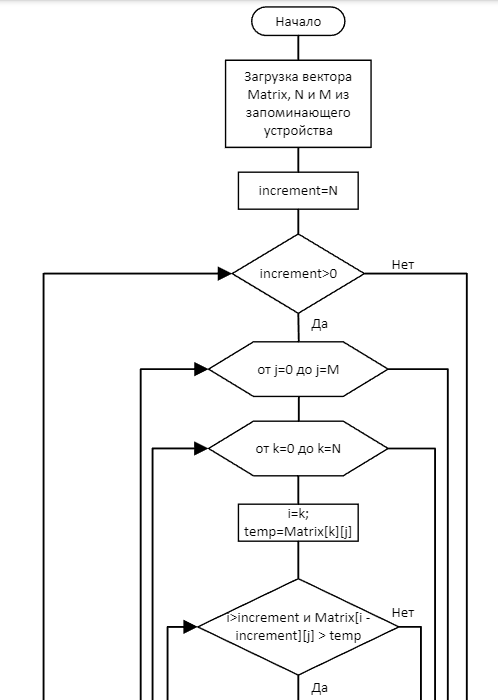


Рисунок 5 – Блок-схема сортировки Шелла

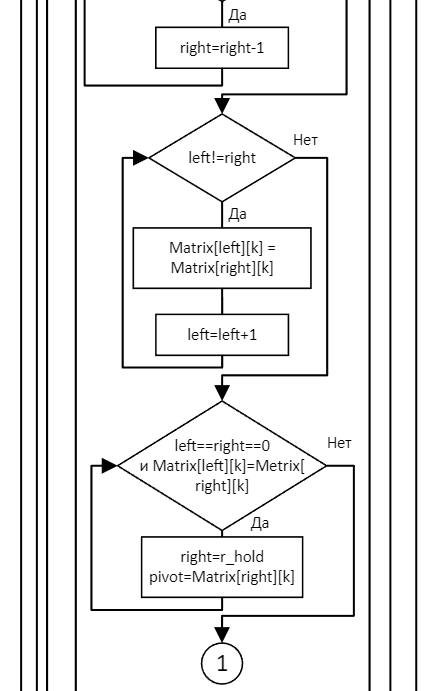
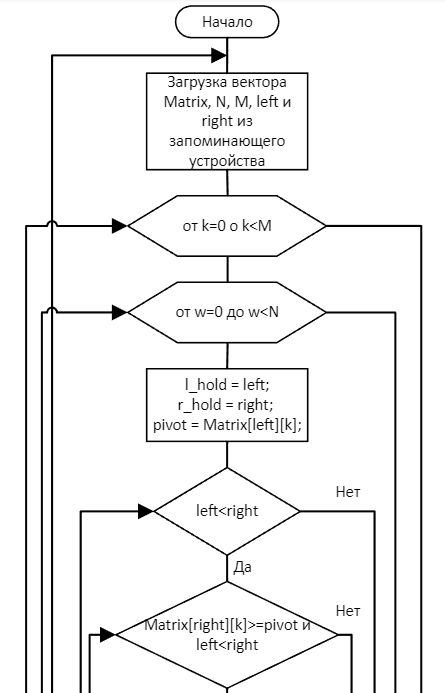


Рисунок 6 – Блок-схема быстрой сортировки

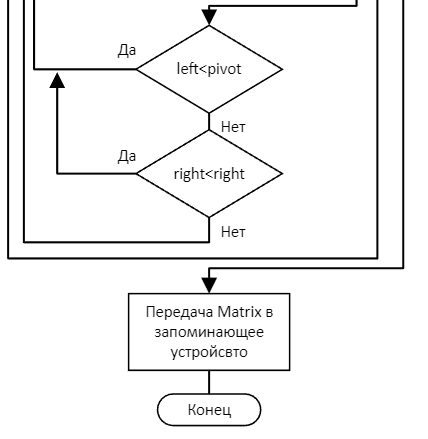
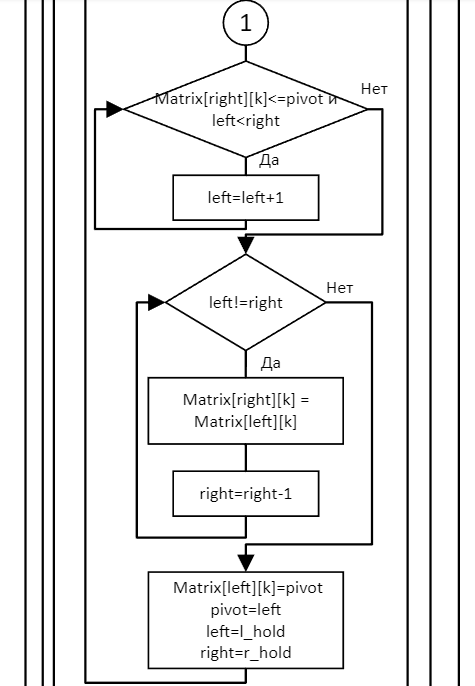


Рисунок 7 – Блок-схема быстрой сортировки

## 9 Результаты работы программы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 8 – Скриншот успешного теста

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 9 – Скриншот непройденного теста

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 10 – Скриншот пузырьковой сортировки

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 11 – Скриншот сортировки отбором

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 12 – Скриншот сортировки вставкой

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 13 – Скриншот сортировки Шелла

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 14 – Скриншот быстрой сортировки

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 15 – Скриншот проверки на ввод цифр, букв

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 16 – Скриншот проверки ввода матрицы

Изображение выглядит как текст

Автоматически созданное описание

Рисунок 17 – Скриншот матрицы из файла

## 10 Исходный код полученного программного решения

//Class.h

#pragma once

#include<vector>

#include "Menu.h"

#include <iostream>

using namespace std;

class CISort {

protected:

int N{};

int M{};

vector<vector<int>> matrix;

int Count\_comparison{};

int Count\_permutation{};

public:

CISort(vector<vector<int>> Matrix, int N, int M);

CISort();

virtual ~CISort();

virtual vector<vector<int>> Sort(vector<vector<int>> Matrix, int n, int m);

void Show(vector<vector<int>> Matrix, int N, int M);

int GetCountComparisons() {

return Count\_comparison;

}

int GetCountPermutations() {

return Count\_permutation;

}

};

class CBubbleSort : public CISort {

public:

vector<vector<int>> Sort(vector<vector<int>> Matrix, int N, int M) override;

CBubbleSort(vector<vector<int>> Matrix, int N, int M);

~CBubbleSort();

};

class CSelectionSort : public CISort {

public:

vector<vector<int>> Sort(vector<vector<int>> Matrix, int N, int M) override;

CSelectionSort(vector<vector<int>> Matrix, int N, int M);

~CSelectionSort();

};

class CInsertionSort : public CISort {

public:

vector<vector<int>> Sort(vector<vector<int>> Matrix, int N, int M) override;

CInsertionSort(vector<vector<int>> Matrix, int N, int M);

~CInsertionSort();

};

class CShellSort : public CISort {

public:

vector<vector<int>> Sort(vector<vector<int>> Matrix, int N, int M) override;

CShellSort(vector<vector<int>> Matrix, int N, int M);

~CShellSort();

};

class CQuickSort : public CISort {

public:

CQuickSort(vector<vector<int>> Matrix, int N, int M);

vector<vector<int>> Sort(vector<vector<int>> Matrix, int N, int M) override;

vector<int> RecurtionSort(vector<int>& Column, int left, int right);

~CQuickSort();

};

//Class.cpp

#include"ParentClass.h"

CISort :: ~CISort() {};

vector<vector<int>> CISort::Sort(vector<vector<int>> Matrix, int n, int m) { n; m; return Matrix; };

CISort::CISort(vector<vector<int>> Matrix, int n, int m) {

this->matrix = Matrix;

this->N = n;

this->M = m;

}

CISort::CISort() {};

void CISort::Show(vector<vector<int>> Matrix, int n, int m) {

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

for (int j = 0; j < m; j++) {

cout << Matrix[i][j]<<"\t";

}

cout << endl;

}

}

CBubbleSort::CBubbleSort(vector<vector<int>> Matrix, int n, int m) {

this->matrix = Matrix;

this->N = n;

this->M = m;

}

vector<vector<int>> CBubbleSort:: Sort(vector<vector<int>> Matrix, int n, int m) {

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

for (int j = 0; j < m; j++) {

for (int k = 1; k < n; k++) {

Count\_comparison += 1;

if (Matrix[k][j] < Matrix[k - 1][j]) {

int temp = Matrix[k - 1][j];

Matrix[k - 1][j] = Matrix[k][j];

Matrix[k][j] = temp;

Count\_permutation += 1;

}

}

}

}

return Matrix;

}

CBubbleSort::~CBubbleSort() {}

CSelectionSort::CSelectionSort(vector<vector<int>> Matrix, int n, int m) {

this->matrix = Matrix;

this->N = n;

this->M = m;

}

vector<vector<int>> CSelectionSort::Sort(vector<vector<int>> Matrix, int n, int m) {

for (int j = 0; j < m; j++) {

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

int min = i;

for (int k = i + 1; k < n; k++) {

Count\_comparison += 1;

if (Matrix[k][j] < Matrix[min][j]) {

min = k;

}

}

int temp = Matrix[i][j];

Matrix[i][j] = Matrix[min][j];

Matrix[min][j] = temp;

Count\_permutation += 1;

}

}

return Matrix;

}

CSelectionSort::~CSelectionSort() {}

CInsertionSort::CInsertionSort(vector<vector<int>> Matrix, int n, int m) {

this->matrix = Matrix;

this->N = n;

this->M = m;

}

vector<vector<int>> CInsertionSort:: Sort(vector<vector<int>> Matrix, int n, int m) {

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

for (int j = 0; j < m; j++) {

for (int k = 1; k < n; k++) {

int value = Matrix[k][j];

int index=k;

Count\_comparison += 1;

while (index > 0 && Matrix[index - 1][j] > value) {

Matrix[index][j] = Matrix[index - 1][j];

index--;

}

Matrix[index][j] = value;

Count\_permutation+= 1;

}

}

}

return Matrix;

}

CInsertionSort::~CInsertionSort() {}

CShellSort::CShellSort(vector<vector<int>> Matrix, int n, int m) {

this->matrix = Matrix;

this->N = n;

this->M = m;

}

vector<vector<int>> CShellSort::Sort(vector<vector<int>> Matrix, int n, int m) {

int increment = n;

while (increment > 0) {

for (int j = 0; j < m; j++) {

for (int k = 0; k < n; k++) {

Count\_comparison += 1;

int i = k;

int temp = Matrix[k][j];

if (i < increment) {

Count\_comparison -= 1;

}

while ((i >= increment) && (Matrix[i - increment][j] > temp)) {

Matrix[i][j] = Matrix[i - increment][j];

i = i - increment;

}

Matrix[i][j] = temp;

Count\_permutation += 1;

}

}

if (increment > 1) {

increment = increment-1;

}

else {

if (increment == 1) {

break;

}

}

}

return Matrix;

}

CShellSort::~CShellSort() {}

CQuickSort::CQuickSort(vector<vector<int>> Matrix, int n, int m) {

this->matrix = Matrix;

this->N = n;

this->M = m;

}

vector<vector<int>> CQuickSort::Sort(vector<vector<int>> Matrix, int n, int m) {

int Left = 0;

int count\_temp = 0;

int Right = n - 1;

vector<int> Column;

vector<vector<int>> temp;

for (int f = 0; f < m; f++) {

for (int k = 0; k < n; k++) {

Column.push\_back(Matrix[k][f]);

}

temp.push\_back(RecurtionSort(Column, Left, Right));

count\_temp++;

for (int i = count\_temp-1; i < count\_temp; i++) {

for (int j = 0; j < n; j++) {

Matrix[j][i] = temp[i][j];

}

}

Column.clear();

}

return Matrix;

}

vector<int> CQuickSort::RecurtionSort(vector<int>& Column, int left, int right) {

int i = left, j = right;

int tmp{};

int Middle = Column[(left + right) / 2];

while (i <= j) {

while (Column[i] < Middle) {

i++;

}

while (Column[j] > Middle) {

j--;

}

Count\_comparison += 1;

if (i <= j) {

tmp = Column[i];

Column[i] = Column[j];

Column[j] = tmp;

Count\_permutation += 1;

i++;

j--;

}

}

if (left < j)

RecurtionSort(Column, left, j);

if (i < right)

RecurtionSort(Column, i, right);

return Column;

}

CQuickSort::~CQuickSort() {}

//Correct\_ipnput.h

#pragma once

#include <iostream>

#include<string>

using namespace std;

int GetInt();

int GetInputMatrix();

//CorrectInput.cpp

#include <iostream>

#include "CorrectInput.h"

using namespace std;

int GetInt() {

int input;

cin >> input;

while (!(input >= 1)) {

cout << "Error! Try again." << endl;

cin.clear();

cin.sync();

cin.ignore(INT\_MAX, '\n');

cin >> input;

}

return input;

}

int GetInputMatrix() {

int input;

cin >> input;

while (!(input >= 1 && input<=3)) {

cout << "Error! Try again." << endl;

cin.clear();

cin.sync();

cin.ignore(INT\_MAX, '\n');

cin >> input;

}

return input;

}

//Function.h

#pragma once

#include "ParentClass.h"

#include "CorrectInput.h"

#include<vector>

#include<ctime>

vector<vector<int>> MannualInputMatrix(vector<vector<int>> Matrix, int N, int M);

vector<vector<int>> RandomInputMatrix(vector<vector<int>> Matrix, int N, int M);

vector<vector<int>> OfFileMatrix(vector<vector<int>> Matrix, int N, int M);

void LookMatrix(vector<vector<int>> Matrix, int N, int M);

vector<vector<int>> CopySortMatrix(vector<vector<int>> Matrix, int N, int M);

vector<vector<int>> SortMatrix;

//Menu.h

#pragma once

#include <iostream>

#include<vector>

#include"Function.h"

enum {

Test\_matrix = 1,

Look\_matrix,

Look\_sort,

Look\_table,

Exit,

};

void MainMenu();

//Menu.cpp

#include "Menu.h"

#include"Test.h"

void MainMenu() {

int n{};

int m{};

vector<vector<int>> Matrix;

cout << "Manual input - 1\nRandom input - 2\nInput of file - 3\nChoose: ";

int choice = GetInputMatrix();

cout << "Enter number of rows: ";

n = GetInt();

cout << "Enter number of columns: ";

m = GetInt();

if (choice == 1) {

Matrix=MannualInputMatrix(Matrix, n, m);

}

if (choice == 2) {

Matrix=RandomInputMatrix(Matrix, n, m);

}

if (choice == 3) {

Matrix = OfFileMatrix(Matrix, n, m);

}

int exit{};

while (exit == 0) {

cout << "Test - 1\nLook on matrix - 2\nLook sorting - 3\nCheck comparative table of the effectiveness of methods - 4\nExit - 5\nChoose: ";

int user\_choice{};

user\_choice = GetInt();

switch (user\_choice) {

case(Test\_matrix): {

Test();

}break;

case(Look\_matrix): {

LookMatrix(Matrix, n, m);

}break;

case(Look\_sort): {

cout << "Bubble sorting" << endl;

CBubbleSort BubbleMatrix(Matrix, n, m);

LookMatrix(BubbleMatrix.Sort(Matrix, n, m), n, m);

cout << "Selection sorting" << endl;

CSelectionSort SelectionMatrix(Matrix, n, m);

LookMatrix(SelectionMatrix.Sort(Matrix, n, m), n, m);

cout << "Insertion sorting" << endl;

CInsertionSort InsertionMatrix(Matrix, n, m);

LookMatrix(InsertionMatrix.Sort(Matrix, n, m), n, m);

cout << "Shell sorting" << endl;

CShellSort ShellMatrix(Matrix, n, m);

LookMatrix(ShellMatrix.Sort(Matrix, n, m), n, m);

cout << "Quick sorting" << endl;

CQuickSort QuickMatrix(Matrix, n, m);

LookMatrix(QuickMatrix.Sort(Matrix, n, m), n, m);

}break;

case(Look\_table): {

CBubbleSort BubbleMatrix(Matrix, n, m);

BubbleMatrix.Sort(Matrix, n, m);

CSelectionSort SelectionMatrix(Matrix, n, m);

SelectionMatrix.Sort(Matrix, n, m);

CInsertionSort InsertionMatrix(Matrix, n, m);

InsertionMatrix.Sort(Matrix, n, m);

CShellSort ShellMatrix(Matrix, n, m);

ShellMatrix.Sort(Matrix, n, m);

CQuickSort QuickMatrix(Matrix, n, m);

QuickMatrix.Sort(Matrix, n, m);

cout << "Name sorting\tCount comparisons\tCount permutations" << endl << endl;

cout << "Bubble\t\t\t" << BubbleMatrix.GetCountComparisons();

cout << "\t\t\t" << BubbleMatrix.GetCountPermutations() << endl << endl;

cout << "Selection\t\t" << SelectionMatrix.GetCountComparisons();

cout << "\t\t\t" << SelectionMatrix.GetCountPermutations() << endl << endl;

cout << "Insertion\t\t" << InsertionMatrix.GetCountComparisons();

cout << "\t\t\t" << InsertionMatrix.GetCountPermutations() << endl << endl;

cout << "Shell\t\t\t" << ShellMatrix.GetCountComparisons();

cout << "\t\t\t" << ShellMatrix.GetCountPermutations() << endl << endl;

cout << "Quick\t\t\t" << QuickMatrix.GetCountComparisons();

cout << "\t\t\t" << QuickMatrix.GetCountPermutations() << endl << endl;

}break;

case(Exit): {

exit = true;

}break;

default: {

cout << "Error. Try again." << endl;

}break;

}

}

}

#include"Test.h"

const int N = 3;

const int M = 4;

vector<vector<int>> Orig\_matrix{ {5, 2, 7, 4}, {1, 3, 6, 10}, {3, 1, 2, 5} };

vector<vector<int>> Matrix\_result{ {1, 1, 2, 4}, {3, 2, 6, 5}, {5, 3, 7, 10} };

bool TestBubble(vector<vector<int>> orig\_matrix, vector<vector<int>> Result\_matrix, int n, int m) {

vector<vector<int>> Matrix\_of\_work;

CBubbleSort BubbleMatrix(orig\_matrix, n, m);

Matrix\_of\_work = CopySortMatrix(BubbleMatrix.Sort(orig\_matrix, n, m), n, m);

bool testing = true;

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

for (int j = 0; j < m; j++) {

if (Matrix\_of\_work[i][j] != Result\_matrix[i][j]) {

testing = false;

}

}

}

if (testing == false) {

cout << endl << "Bubble sorting failed." << endl << endl;

cout << "The result of the work:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Matrix\_of\_work[i][j] << "\t";

}

cout << endl;

}

cout << endl << "Expected result:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Result\_matrix[i][j] << "\t";

}

cout << endl;

}

}

return testing;

}

bool TestSelect(vector<vector<int>> orig\_matrix, vector<vector<int>> Result\_matrix, int n, int m) {

vector<vector<int>> Matrix\_of\_work;

CSelectionSort SelectionMatrix(orig\_matrix, n, m);

Matrix\_of\_work = CopySortMatrix(SelectionMatrix.Sort(orig\_matrix, n, m), n, m);

bool testing = true;

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

for (int j = 0; j < m; j++) {

if (Matrix\_of\_work[i][j] != Result\_matrix[i][j]) {

testing = false;

}

}

}

if (testing == false) {

cout << endl << "Selection sorting failed." << endl << endl;

cout << "The result of the work:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Matrix\_of\_work[i][j] << "\t";

}

cout << endl;

}

cout << endl << "Expected result:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Result\_matrix[i][j] << "\t";

}

cout << endl;

}

}

return testing;

}

bool TestInsert(vector<vector<int>> orig\_matrix, vector<vector<int>> Result\_matrix, int n, int m) {

vector<vector<int>> Matrix\_of\_work;

CInsertionSort InsertionMatrix(orig\_matrix, n, m);

Matrix\_of\_work = CopySortMatrix(InsertionMatrix.Sort(orig\_matrix, n, m), n, m);

bool testing = true;

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

for (int j = 0; j < m; j++) {

if (Matrix\_of\_work[i][j] != Result\_matrix[i][j]) {

testing = false;

}

}

}

if (testing == false) {

cout << endl << "Insertion sorting failed." << endl << endl;

cout << "The result of the work:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Matrix\_of\_work[i][j] << "\t";

}

cout << endl;

}

cout << endl << "Expected result:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Result\_matrix[i][j] << "\t";

}

cout << endl;

}

}

return testing;

}

bool TestShell(vector<vector<int>> orig\_matrix, vector<vector<int>> Result\_matrix, int n, int m) {

vector<vector<int>> Matrix\_of\_work;

CShellSort ShellMatrix(orig\_matrix, n, m);

Matrix\_of\_work = CopySortMatrix(ShellMatrix.Sort(orig\_matrix, n, m), n, m);

bool testing = true;

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

for (int j = 0; j < m; j++) {

if (Matrix\_of\_work[i][j] != Result\_matrix[i][j]) {

testing = false;

}

}

}

if (testing == false) {

cout << endl << "Shell sorting failed." << endl << endl;

cout << "The result of the work:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Matrix\_of\_work[i][j] << "\t";

}

cout << endl;

}

cout << endl << "Expected result:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Result\_matrix[i][j] << "\t";

}

cout << endl;

}

}

return testing;

}

bool TestQuick(vector<vector<int>> orig\_matrix, vector<vector<int>> Result\_matrix, int n, int m) {

vector<vector<int>> Matrix\_of\_work;

CQuickSort QuickMatrix(orig\_matrix, n, m);

Matrix\_of\_work = CopySortMatrix(QuickMatrix.Sort(orig\_matrix, n, m), n, m);

bool testing = true;

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

for (int j = 0; j < m; j++) {

if (Matrix\_of\_work[i][j] != Result\_matrix[i][j]) {

testing = false;

}

}

}

if (testing == false) {

cout << endl << "Quick sorting failed." << endl << endl;

cout << "The result of the work:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Matrix\_of\_work[i][j] << "\t";

}

cout << endl;

}

cout << endl << "Expected result:" << endl;

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

for (int j = 0; j < m; j++) {

cout << Result\_matrix[i][j] << "\t";

}

cout << endl;

}

}

return testing;

}

void Test() {

if (TestBubble(Orig\_matrix, Matrix\_result, N, M) == true && TestSelect(Orig\_matrix, Matrix\_result, N, M) == true && TestInsert(Orig\_matrix, Matrix\_result, N, M) == true && TestShell(Orig\_matrix, Matrix\_result, N, M) == true && TestQuick(Orig\_matrix, Matrix\_result, N, M) == true) {

cout << endl << "Test was successful." << endl << endl;

}

}

#pragma once

#include"Menu.h"

bool TestBubble(vector<vector<int>> Matrix, vector<vector<int>> Result\_matrix, int N, int M);

bool TestSelect(vector<vector<int>> Matrix, vector<vector<int>> Result\_matrix, int N, int M);

bool TestInsert(vector<vector<int>> Matrix, vector<vector<int>> Result\_matrix, int N, int M);

bool TestShell(vector<vector<int>> Matrix, vector<vector<int>> Result\_matrix, int N, int M);

bool TestQuick(vector<vector<int>> Matrix, vector<vector<int>> Result\_matrix, int N, int M);

void Test();

//Main.cpp

#include"Menu.h"

int main(void) {

cout << "Kondilyabrova Veronika Daniilovna\nGroup 414" << endl;

cout << "Control work 3\nVariant 10\nArrange each column of the matrix in ascending order.\n" << endl;

MainMenu();

return 0;

}