Министерство науки и Высшего образования

Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Курский государственный университет»

Факультет физики, математики, информатики

Кафедра программного обеспечения и администрирования информационных систем

Курсовой проект

по дисциплине

структуры и алгоритмы компьютерной обработки данных

на тему: ПРОГРАММНАЯ РЕАЛИЗАЦИЯ СРАВНЕНИЯ   
АЛГОРИТМОВ СОРТИРОВКИ ДАННЫХ

Обучающегося 2 курса

очной формы обучения

направления подготовки

02.03.03 Математическое обеспечение и администрирование информационных систем

Направленность (профиль) Проектирование информационных систем и баз данных

Ольшевского Даниила Вадимовича

Руководитель:

профессор кафедры ПОиАИС Кудинов Виталий Алексеевич

Допустить к защите:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ / \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_\_\_г.

Курск, 2023

Содержание

[1. Задание 3](#_Toc112685589)

[1.1 Формулировка условия задачи 3](#_Toc112685590)

[1.2 Краткие теоретические сведения об объекте исследования 3](#_Toc112685591)

[2 Проектирование и разработка приложения 5](#_Toc112685604)

[2.1 Описание структуры приложения 5](#_Toc112685606)

[2.2 Используемые алгоритмы 5](#_Toc112685607)

[3 Функциональное описание структур данных и алгоритмов 12](#_Toc112685613)

[3.1 Разработка структуры данных 12](#_Toc112685615)

[3.2 Разработка модели поведения объекта 13](#_Toc112685616)

[3.3 Защита приложения от ошибок пользователя 14](#_Toc112685616)

[4 Анализ проекта и тестирование 16](#_Toc112685613)

[4.1 Тестирование приложения 16](#_Toc112685615)

[4.2 Выводы 22](#_Toc112685616)

[Заключение 22](#_Toc112685617)

[Список использованных источников 23](#_Toc112685618)

[Приложение А](#_Toc112685621) [Текст программы 24](#_Toc112685622)

# **Задание**

## Формулировка условия задачи

Реализовать программу для сравнения алгоритмов сортировки данных, в которой возможно генерировать массив случайных чисел из произвольного количества элементов с возможностью задания типа числовых данных (вещественных или целочисленных), а также отображение содержимого массива до и после сортировки. Реализовать построение графиков зависимости времени выполнения сортировки от размера массива и количества произведенных перестановок, в зависимости от размера массива. Также, после произведенных сортировок массива данных различными алгоритмами, результаты должны сохраняться в отчет. Предоставить возможность пользователю наглядно отследить работу алгоритма сортировки с помощью демонстрации динамики алгоритма в виде анимации.

## Краткие теоретические сведения об объекте исследования

Одной из основных сложностей при создании программ для сортировки данных является выбор подходящего алгоритма. Существует множество различных алгоритмов сортировки, каждый из которых имеет свои преимущества и недостатки в зависимости от типа данных и объема информации.

Некоторые из наиболее распространенных алгоритмов сортировки включают в себя:

1. Сортировка пузырьком: это один из самых простых алгоритмов сортировки, который работает путем сравнения двух элементов и перестановки их местами, если они находятся в неправильном порядке;

2. Сортировка вставками: этот алгоритм работает путем вставки каждого элемента на свое место в уже отсортированную последовательность;

3. Сортировка выбором: этот алгоритм работает путем выбора наименьшего элемента из списка и перемещения его на первую позицию, затем выбора следующего наименьшего элемента из оставшейся части списка и перемещения его на вторую позицию и т.д.;

4. Быстрая сортировка: это один из самых эффективных алгоритмов сортировки, который работает путем разбиения списка на две части, сортировки каждой из них отдельно и объединения их вместе.

Теоретические основы программирования сортировки данных включают в себя понимание того, какие алгоритмы работают лучше для разных типов данных, какие методы оптимизации могут быть использованы для ускорения процесса сортировки и как реализовать алгоритмы с использованием различных языков программирования и структур данных.

Важно также учитывать, что время выполнения алгоритма сортировки зависит от объема данных, которые нужно отсортировать. Чем больше данные, тем больше времени потребуется на сортировку. Поэтому при разработке программ для сортировки данных необходимо учитывать возможность оптимизации алгоритмов и использование параллельных вычислений для ускорения процесса.

Для реализации возможности задания числового типа пользователем при генерации числовых массивов, для их дальнейшей сортировки одним из алгоритмов, было принято решение использовать шаблонные типы данных для хранения данных.

# **2 Структурное описание разработки**



## Описание структуры приложения

Приложение состоит из следующих модулей:

MyForm.cpp, MyForm.h – модуль содержащий реализацию основных функций приложения, заголовочный файл и файл формы Main.

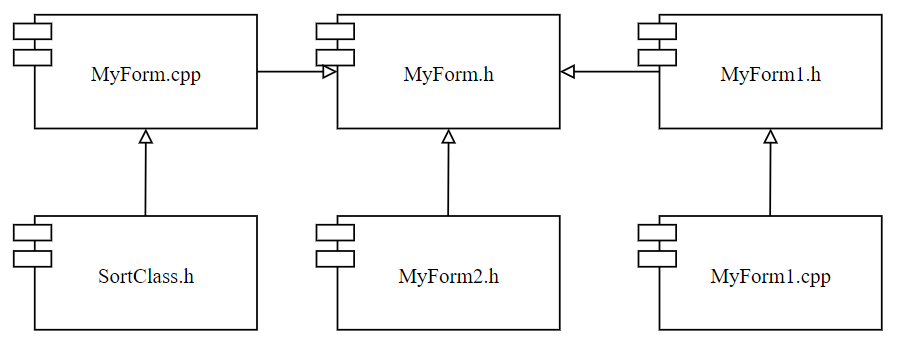
MyForm1.cpp, MyForm1.h – модуль содержащий реализацию динамического отображения процесса сортировки в виде анимации.

MyForm2.h – модуль содержащий справочную информацию.

SortClass.h – файл содержащий каталог алгоритмов сортировки.

resource.h – файл проекта среды разработки Visual Studio 2019.

Диаграмма компонентов приложения представлена на рисунке 12.



## Рисунок 1 – Диаграмма компонентов

## Используемые алгоритмы

Основные алгоритмы, обеспечивающие стабильную работу и функциональность программы, реализуют функции, записанные в модуле SortClass.h. В зависимости от необходимого пользователю алгоритма сортировки происходит переход к одному из производных классов, описанных в 2.1, содержащих в себе переопределение абстрактных методов my\_sort1 и my\_sort2 из класса *BaseSort*.

SortClass::BaseSort<int>\* p;

p = new SortClass::Buble\_sort<int>(10);

p->my\_sort1(p->v, re = 0);

Выполняет пузырьковую сортировку числового массива.

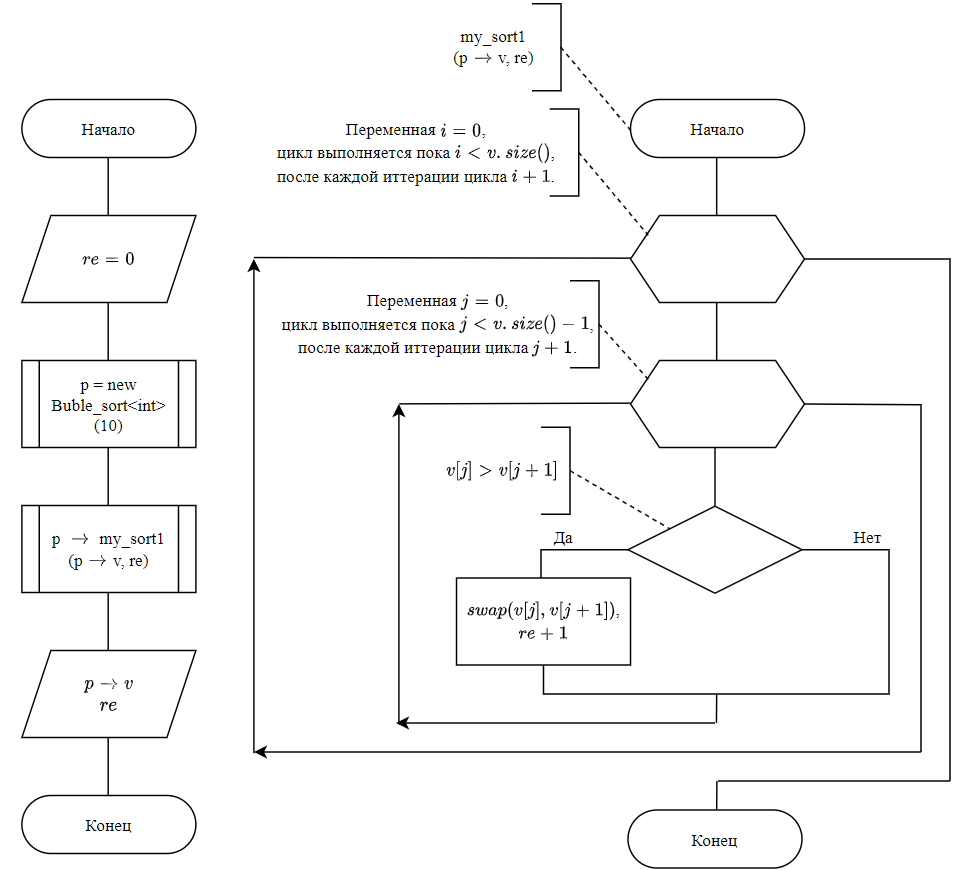


Рисунок 2 – Алгоритм метода пузырьковой сортировки

(производный класс Buble\_sort)

SortClass::BaseSort<int>\* p;

p = new SortClass:: Choice\_sort<int>(10);

p->my\_sort1(p->v, re = 0);

Выполняет сортировку числового массива выбором.

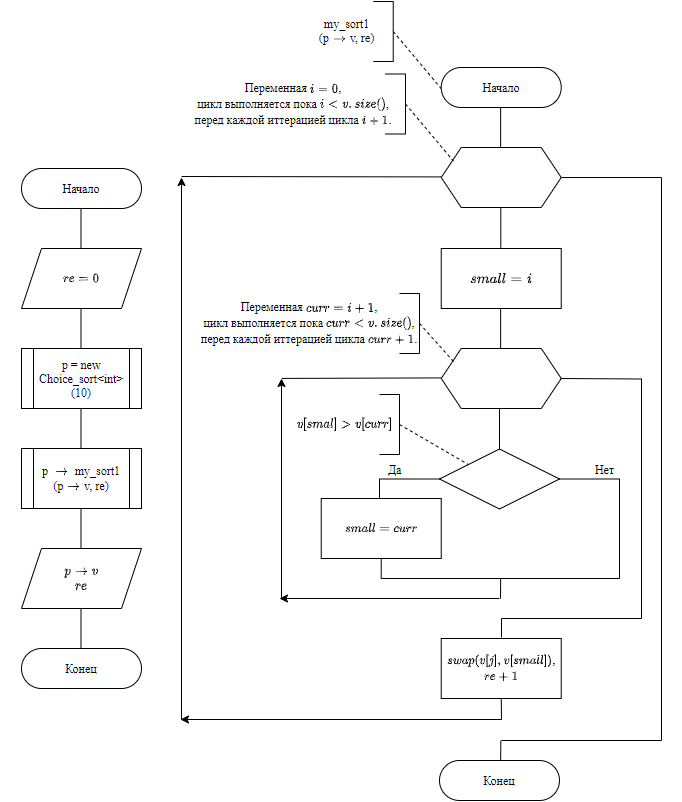


Рисунок 3 – Алгоритм метода сортировки выбором

(производный класс Choice\_sort)

SortClass::BaseSort<int>\* p;

p = new SortClass:: Paste\_sort<int>(10);

p->my\_sort1(p->v, re = 0);

Выполняет сортировку числового массива вставками.

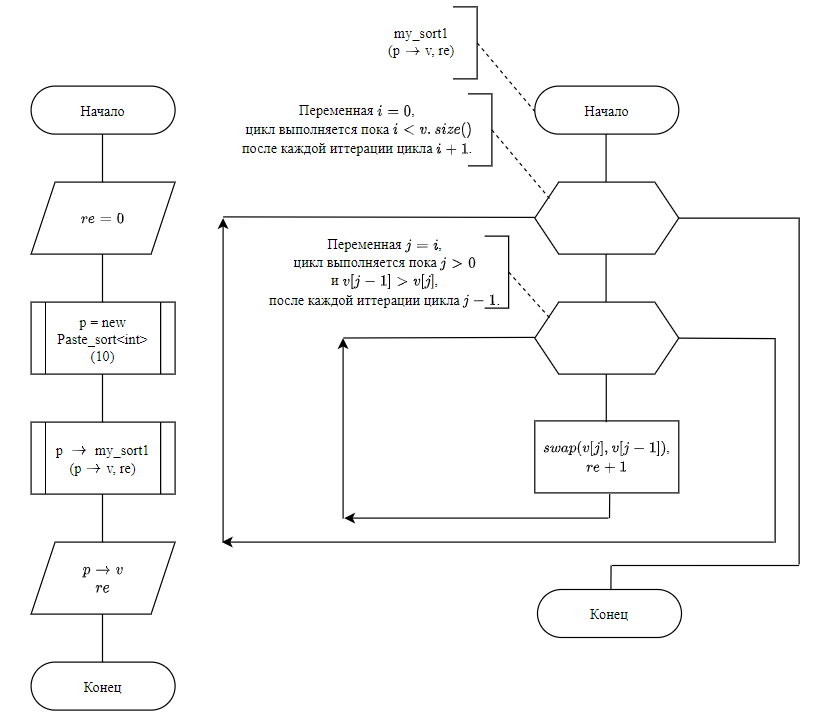


Рисунок 4 – Алгоритм метода сортировки вставками

(производный класс Paste\_sort)

SortClass::BaseSort<int>\* p;

p = new SortClass:: Even\_odd\_sort<int>(10);

p->my\_sort1(p->v, re = 0);

Выполняет чёт-нечет сортировку числового массива.

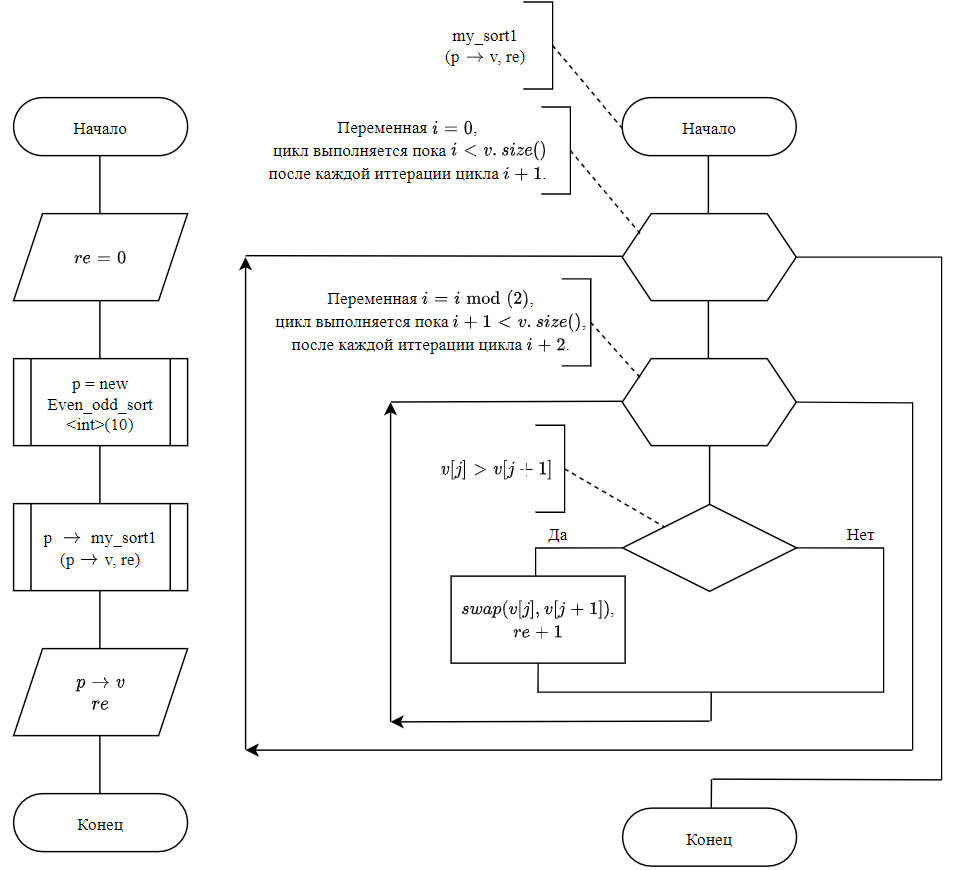


Рисунок 5 – Алгоритм метода сортировки чёт-нечет

(производный класс Even\_odd\_sort)

SortClass::BaseSort<int>\* p;

p = new SortClass:: Shaker \_sort<int>(10);

p->my\_sort1(p->v, re = 0);

Выполняет шейкерную сортировку числового массива.

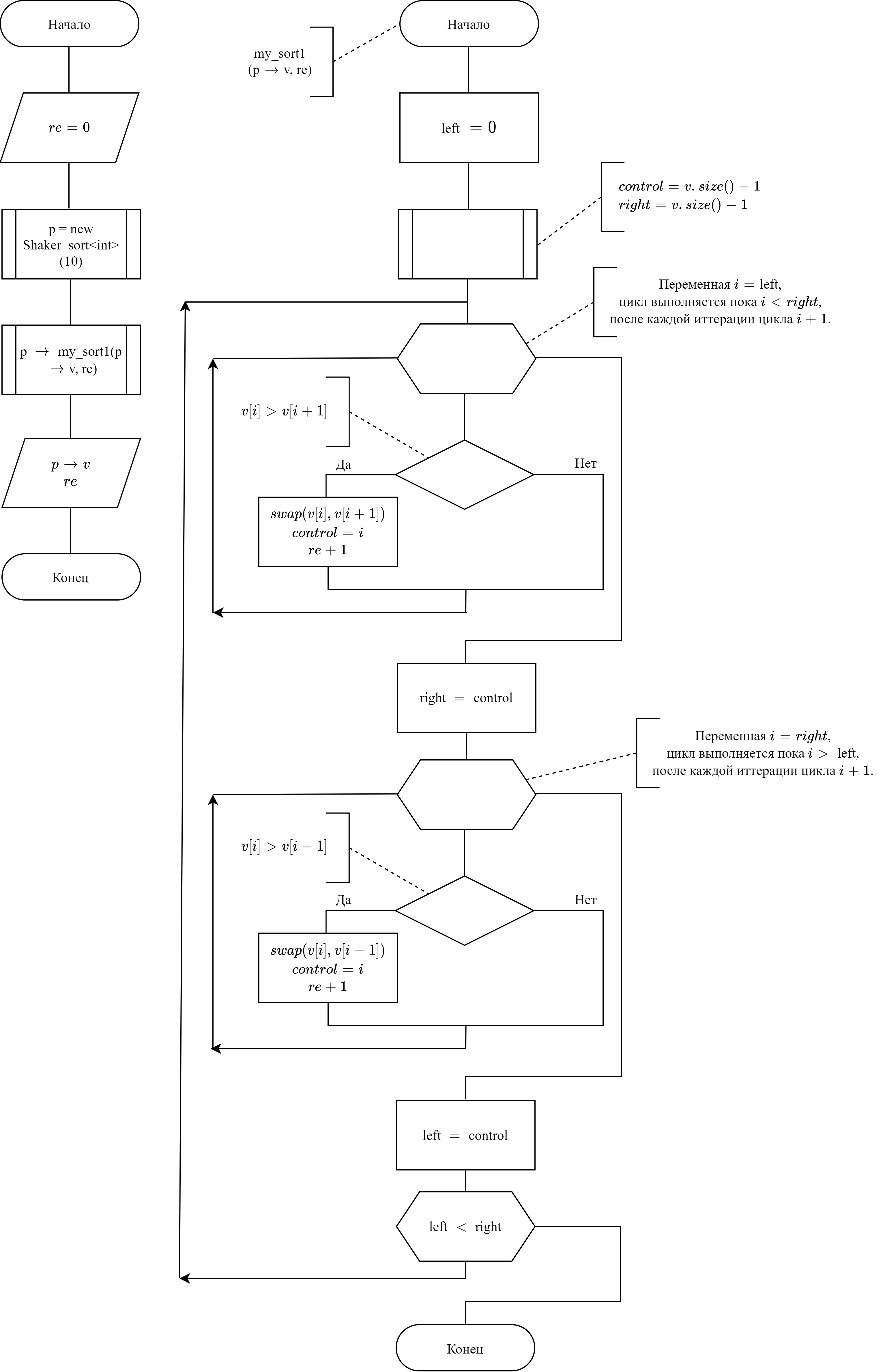


Рисунок 6 – Алгоритм метода шейкерной сортировки

(производный класс Shaker\_sort)

SortClass::BaseSort<int>\* p;

p = new SortClass::Shell\_sort<int>(10);

p->my\_sort1(p->v, re = 0);

Выполняет сортировку числового массива алгоритмом Шелла.

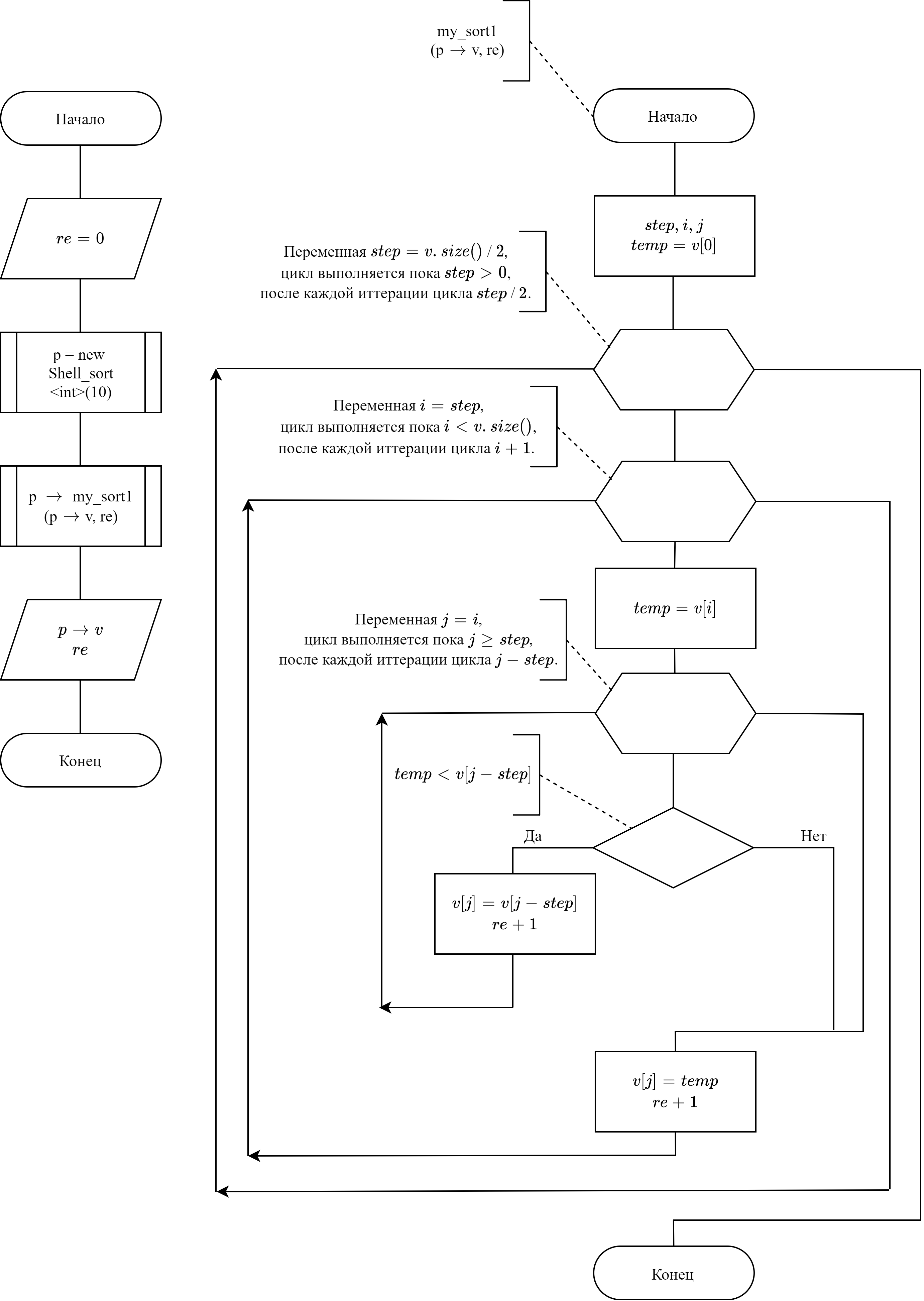


Рисунок 7 – Алгоритм метода сортировки Шелла

(производный класс Shell \_sort)

# 3 Функциональное описание структур данных и алгоритмов

## 3.1 Разработка структуры данных

Программный объект представляет собой набор алгоритмов сортировки данных, к которым программа будет обращаться в процессе работы, а также числовой массив, заполняющийся случайными числами. Для его хранения был выбран базовый класс сортировки – *BaseSort*, содержащий в себе необходимое поле v – числовой массив, а также абстрактные методы, описанные в производных классах. [[5]](#_Список_использованных_источников).

*class* BaseSort

    {

*public:*

*//Конструктор(ы)*

        BaseSort(*const* BaseSort<T>*&*);

        BaseSort(*size\_t*);

        BaseSort() : BaseSort(default\_size) {}

        ~BaseSort() {};

Необходимые конструкторы и деконструктор объекта.

*//Метод(ы)*

*virtual* *void* my\_sort1(std::vector<T>*&* *v*, *size\_t&* *re*) = 0;

*virtual* *void* my\_sort2(std::vector<T>*&* *v*, *int* *first*, *int* *last*, *size\_t&* *re*) = 0;

*void* exception() { if (true) throw MyException(); };

Абстрактные методы сортировки, при вызове которых происходит обращение к производным классам сортировок.

*//Поля*

        std::vector <T> v;

Массив чисел, который будет отсортирован в ходе работы программы.

*//Класс исключений*

*class* MyException {};

Локальный класс исключений.

*private:*

*const* *size\_t* default\_size = 0;

Размер числового массива по умолчанию.

    };

Для обращения к необходимому алгоритму сортировки используются производные классы от *BaseSort:* *Buble\_sort, Choice\_sort, Paste\_sort, Even\_odd\_sort, Shaker\_sort, Shell\_sort.* В данных производных классах отсутствуют поля для хранения данных, а также в них содержатся различные алгоритмы сортировки, к которым идет обращение при вызове абстрактного метода сортировки my\_sort1 или my\_sort2 из класса *BaseSort.*

## 3.2 Разработка модели поведения объекта

Приложение работает с объектами класса *BaseSort* и его полями.

*virtual* *void* my\_sort1(std::vector<T>*&* *v*, *size\_t&* *re*) = 0;

Выполняет переход к необходимому алгоритму сортировки.

*virtual* *void* my\_sort2(std::vector<T>*&* *v*, *int* *first*, *int* *last*, *size\_t&* *re*) = 0;

Выполняет переход к необходимому алгоритму сортировки.

*void* exception() { if (true) throw MyException(); };

Осуществляет вызов исключения.

## 3.3 Защита приложения от ошибок пользователя

С целью предотвращения некорректной работы приложения в результате действий пользователя были использованы следующие средства:

1. При попытке удаления из таблицы единственной строки, пользователю отображается сообщение об ошибке.

Пример,

if (dataGridView1->Rows->Count == 1)

MessageBox::Show(this, "Нельзя удалить единственную строчку!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

else

dataGridView1->Rows->RemoveAt(dataGridView1-> CurrentRow->Index);

1. При попытке сохранения пустого файла пользователем, отображается сообщение об ошибке.

Пример,

try {

if (saveFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK && saveFileDialog1->FileName->Length > 0) {

IO::File::WriteAllText(saveFileDialog1->FileName, L"");

IO::File::AppendAllText(saveFileDialog1->FileName, "Тест № " + System::Convert::ToString(test) + Environment::NewLine + Environment::NewLine);

for (int i = 0; i < dataGridView1->Rows[i]->Cells->Count; i++) {

IO::File::AppendAllText(saveFileDialog1->FileName,

" Метод сортировки: " + dataGridView1->Rows[i]->Cells[0]->Value->ToString() + Environment::NewLine

+ " Количество элементов: " + dataGridView1->Rows[i]->Cells[1]->Value->ToString() + Environment::NewLine

+ " Время выполнения сортировки: " + dataGridView1->Rows[i]->Cells[2]->Value->ToString() + Environment::NewLine

+ " Количество перестановок: " + dataGridView1->Rows[i]->Cells[3]->Value->ToString() + Environment::NewLine + Environment::NewLine);

}

MessageBox::Show(this, "Отчет сохранен", "Готово", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

catch (Exception^ e) {

MessageBox::Show(this, "Файл пустой", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

1. Если пользователь не укажет размерность генерируемого числового массива, отобразится соответствующее сообщение об ошибке. Генерация числового массива при этом не происходит.

Пример,

try {

p = new SortClass::Buble\_sort<int>(System:: Convert::ToInt32(textBox2->Text));

if (p->v.size() <= 100) {

textBox1->Text = nullptr;

for (size\_t i = 0; i < p->v.size(); i++) {

textBox1->Text = textBox1->Text + System::Convert::ToString(p->v[i]) + " ";

}

MessageBox::Show(this, "Массив успешно создан", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

textBox1->Text = nullptr;

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

catch (Exception^ e) {

textBox1->Text = nullptr;

MessageBox::Show(this, "Некоректные данные", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

Приложение отслеживает возможные ошибки ввода данных, предотвращает совершение ошибок пользователя при выполнении действий над данными. Таким образом, можно считать, что приложение имеет достаточный уровень защиты от ошибок пользователя.

# 4 Описание работы программы на конкретных примерах



## Тестирование приложения

Проверка функционирования приложения в нормальных условиях при обычных значениях входных данных представлена на рисунках 8 –10.

На рисунке 8 изображён запуск приложения, генерация числового массива с целочисленным типом данных, на рисунке 9 – выполняется сортировка массива различными алгоритмами, а также построение графиков зависимости на основе полученных данных, на рисунке 10 – запуск визуализации работы алгоритма сортировки в виде анимации.

Проверка в экстремальных условиях при граничных значениях входных данных представлена на рисунках 11 – 13.

На рисунке 11 изображена сортировка массива с большим количеством элементов, на рисунке 12 – большое количество элементов для динамического отображения работы алгоритма, на рисунке 13 – не монотонные данные при строении графиков зависимостей.

Проверка в исключительных ситуациях при недопустимых значениях входных данных представлена на рисунках 14 – 16.

На рисунках 14 – 15 изображена попытка генерации числового массива на основе неверных введенных данных. На рисунке 16 изображена попытка заполнить массив, для динамического отображения работы алгоритма сортировки, некорректными данными.

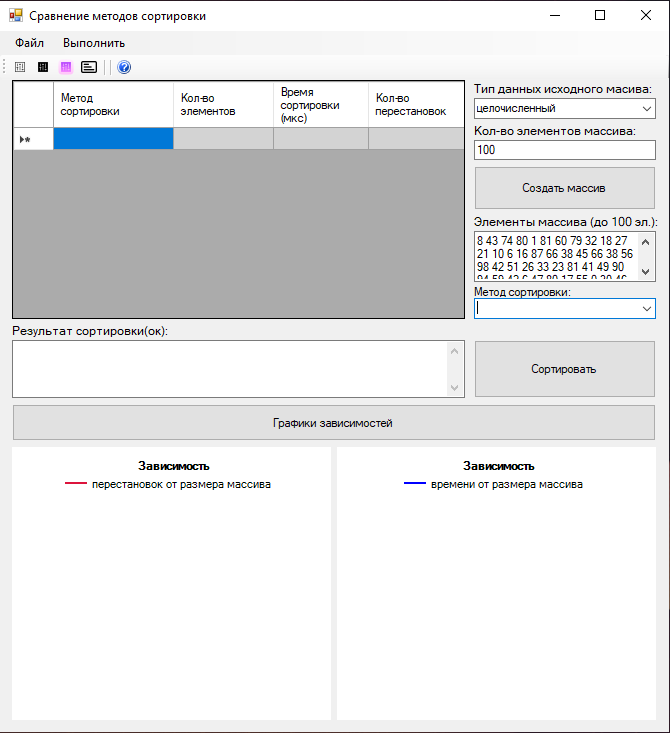


Рисунок 8 – Тест в нормальных условиях

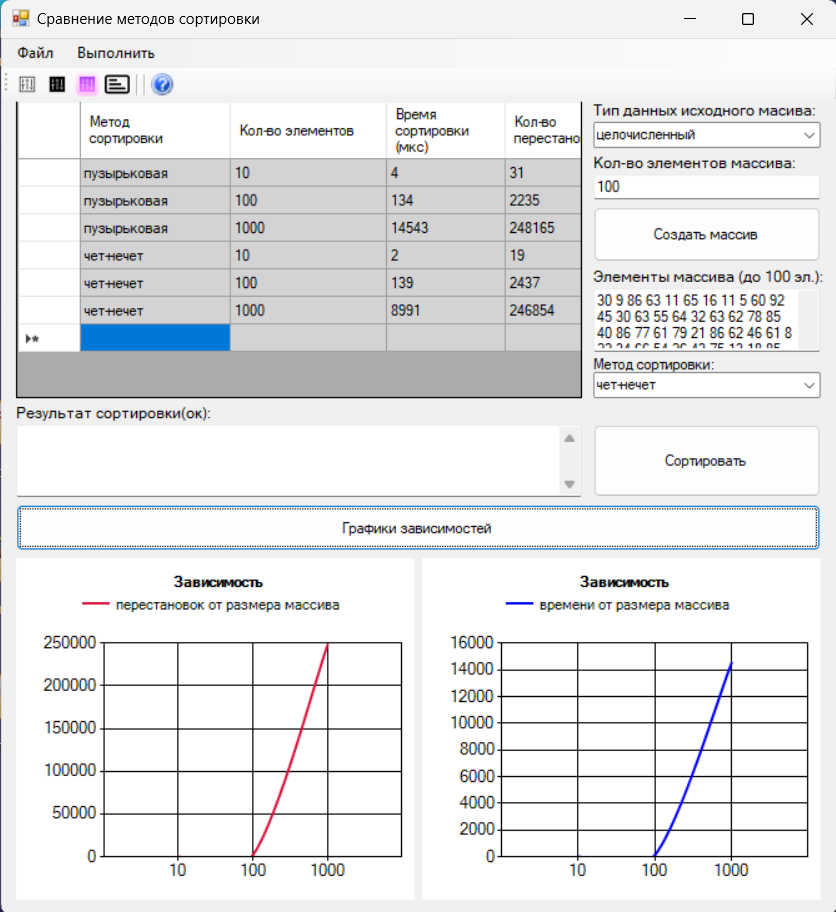


Рисунок 9 – Тест в нормальных условиях

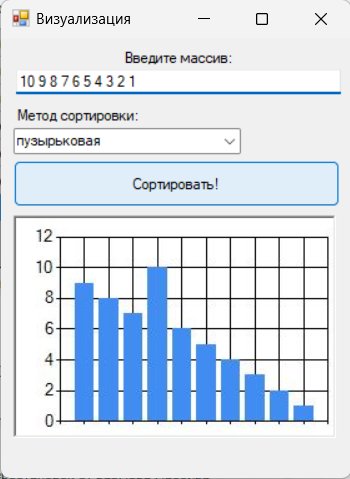


Рисунок 10 – Тест в нормальных условиях

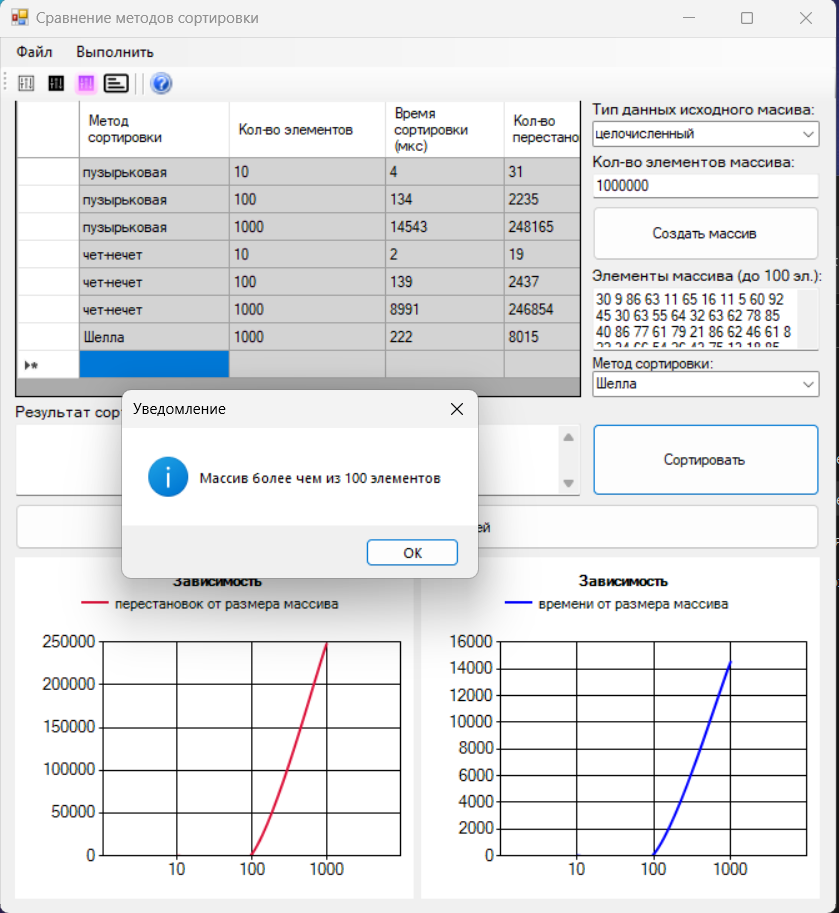


Рисунок 11 – Тест в экстремальных условиях

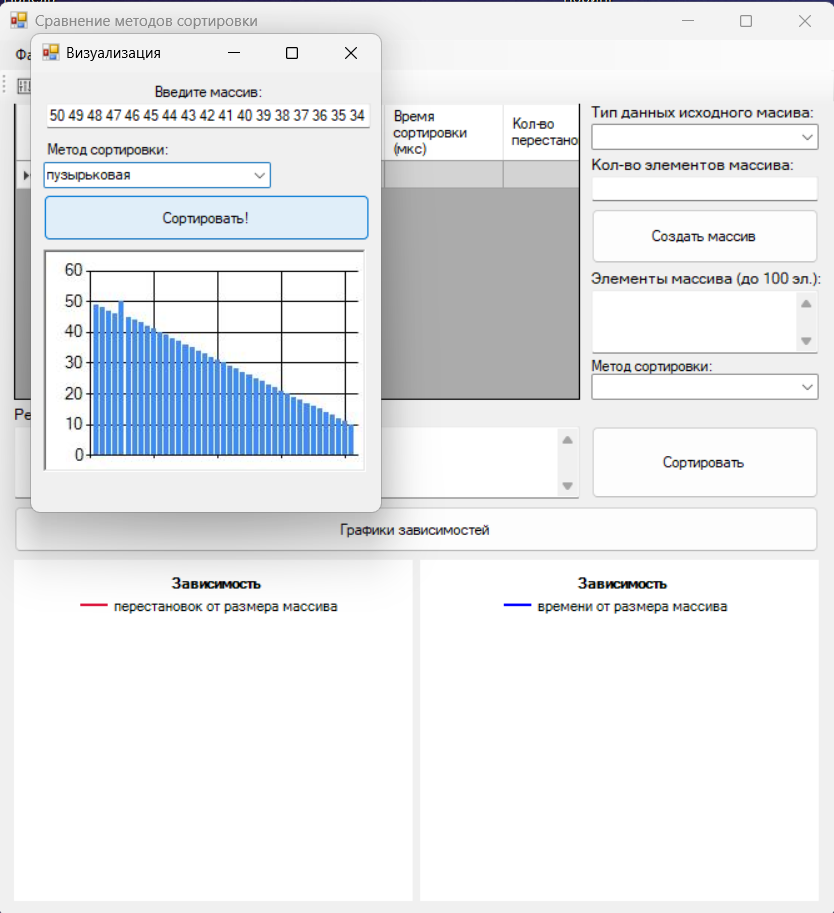


Рисунок 12 – Тест в экстремальных условиях

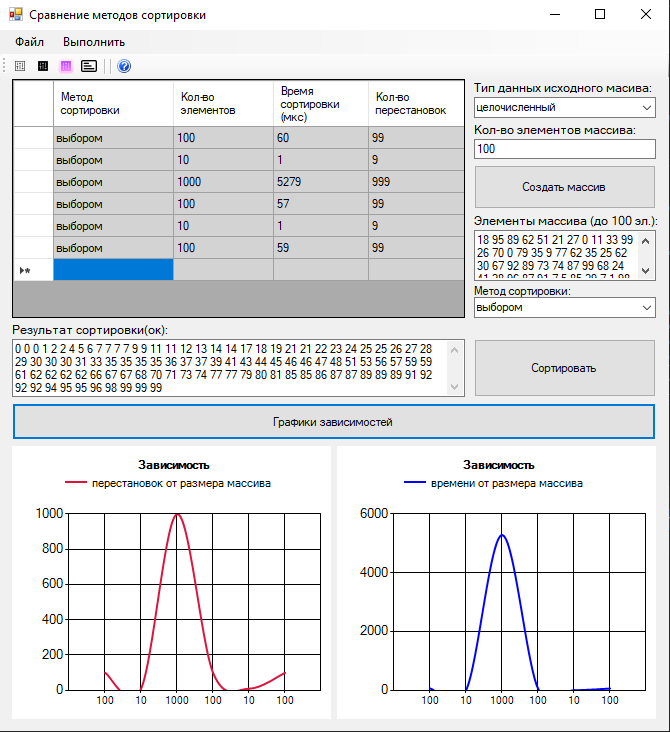


Рисунок 13 – Тест в экстремальных условиях

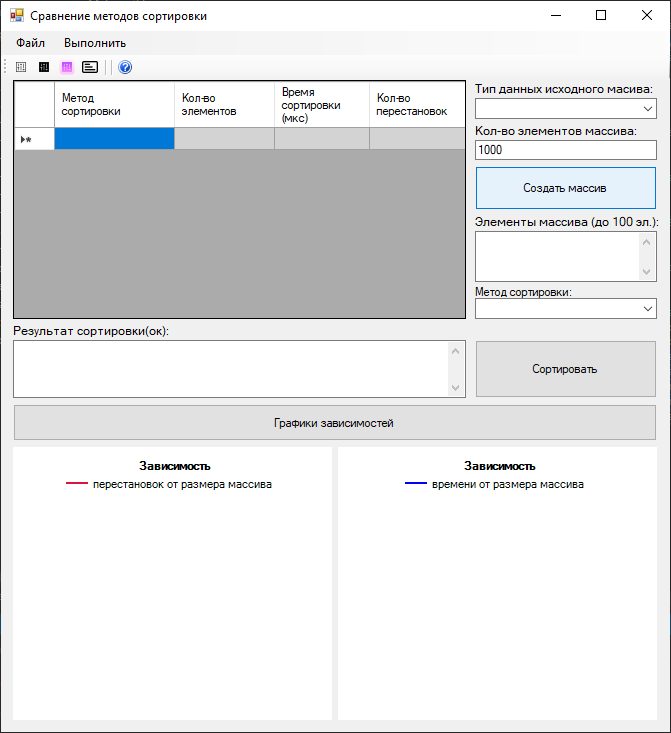


Рисунок 14 – Тест в исключительных условиях

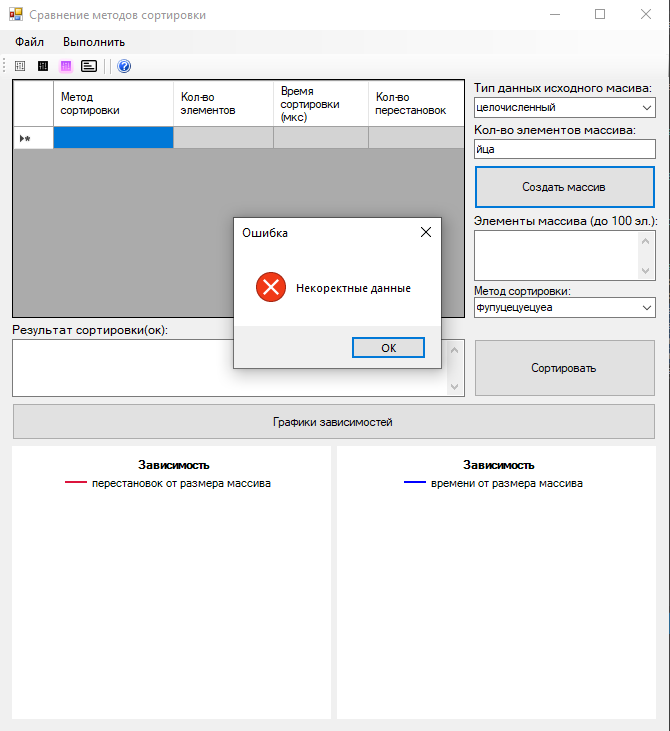


Рисунок 15 – Тест в исключительных условиях

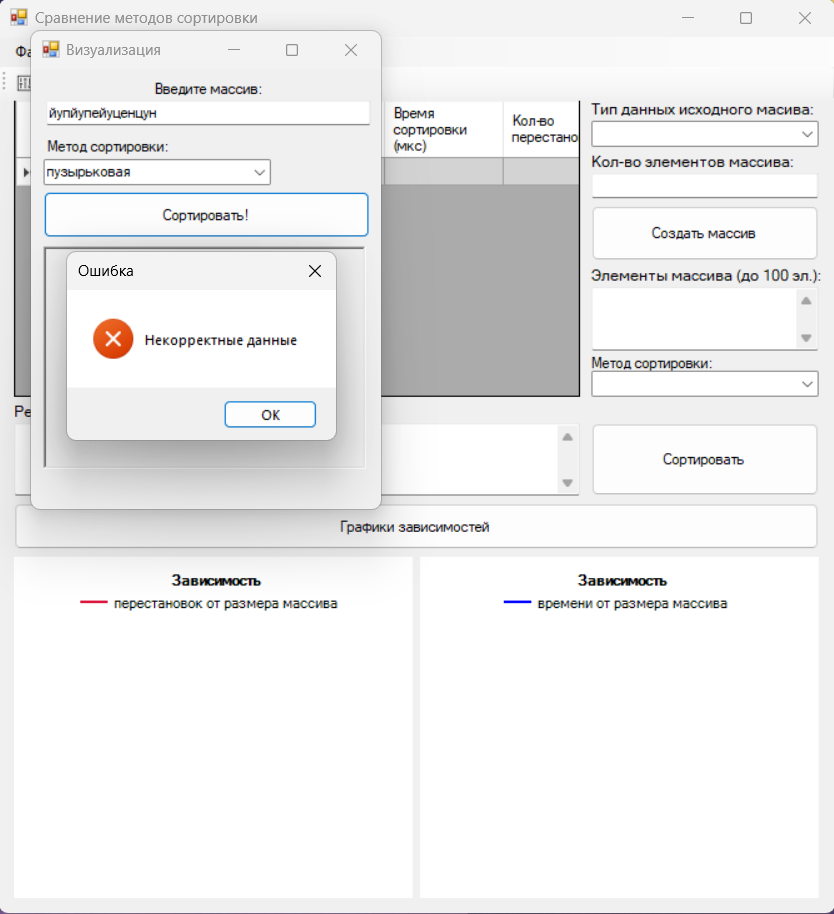


Рисунок 16 – Тест в исключительных условиях

## Выводы

Временные характеристики выполнения алгоритмов зависят исключительно от комплектующих используемого компьютера. Программа приводит затраченное время на работу алгоритма, а также предоставляет информацию о количестве совершенных перестановок в числовом массиве в процессе работы.

Программа протестирована в исключительных и экспериментальных условиях, из чего можно сделать вывод о ее корректной работе, даже в случае подачи некорректных данных на ввод.

# Заключение

В процессе разработки программного обеспечения для решения конкретной задачи была изучена специфическая литература по теме проекта: грамотное и рациональное распределение задач, документация по среде разработки Visual Studio 2022, документация по языку C++. Был разработан алгоритм генерации числовых массивов требуемого числового типа, его сортировки различными алгоритмами, интерфейс приложения, разработаны классы для обеспечения необходимого функционала программы. Так же были изучены библиотеки языка, работающие с потоками файлов, с растровой графикой, вводом и выводом информации.

Программа может быть использована пользователем, для проведения сравнительного анализа алгоритмов сортировки, а также для наглядного понимания работы алгоритма, на основе динамического отображения в виде анимации.

Усовершенствовать данное приложения можно с помощью улучшения интерфейса, добавлением большего числа алгоритмов для динамической демонстрации.

# Список использованных источников

1. "ГОСТ 7.32-2017. Межгосударственный стандарт. Система стандартов по информации, библиотечному и издательскому делу. Отчет о научно-исследовательской работе. Структура и правила оформления" (введен в действие Приказом Росстандарта от 24.10.2017 N 1494-ст)
2. Сидорина, Т.Л. Самоучитель Microsoft Visual Studio C++ и MFC. – СПб.: БХВ-Петербург, 2009. – 848 с.
3. Седжвик, Р. Алгоритмы на С++.: Пер. с англ. – М.: ООО «И.Д. Вильямс», 2016. – 1056 с.
4. Павловская, Т.А. C/C++. Программирование на языке высокого уровня. —СПб.: Питер, 2003. – 461с: ил.
5. Лафоре, Р. Объектно-ориентированное программирование в C++. Классика Computer Scince. 4-е изд. – СПб.: Питер, 2022. – 928 с.
6. Справка по C++ – std::vector [Сайт]. URL: https://ru.cppreference.com/ w/cpp/container/vector (дата обращения 15.11.2022)
7. Справка по C++ – Chart класс [Сайт]. URL: https://learn.microsoft.com/ru/dotnet/api/system.windows.forms.datavisualization.charting.chart?view=netframework-4.8.1&viewFallbackFrom=net-6.0 (дата обращения 19.04.2023)
8. Вирт, Н. Алгоритмы и структуры данных. – М.: ДМК Пресс, 2016 – 272 с.

# Приложение А

# Текст программы

# Модуль SortClass.h:

Макаров

Петров

КП.020303.213.20.20

#include <vector>

#include <algorithm>

#include <time.h>

#include <iostream>

namespace SortClass

{

template<typename T>

class BaseSort

{

public:

//Конструктор(ы)

BaseSort(const BaseSort<T>&);

BaseSort(size\_t);

BaseSort() : BaseSort(default\_size) {}

~BaseSort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) = 0;

virtual void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) = 0;

void exception() { if (true) throw MyException(); };

//Поля

std::vector <T> v;

//Класс исключений

class MyException {};

private:

const size\_t default\_size = 0;

};

//Конструктор(ы) (BaseSort)

template<typename T>

BaseSort<T>::BaseSort(const BaseSort<T>& ob)

{

this->v = v;

}

template<>

BaseSort<int>::BaseSort(size\_t size) : v(size)

{

srand(time(NULL));

for (size\_t i = 0; i < size; i++) {

v[i] = rand() % 100;

}

}

template<>

BaseSort<double>::BaseSort(size\_t size) : v(size)

{

srand(time(NULL));

for (size\_t i = 0; i < size; i++) {

v[i] = (double)rand() / 1000.7;

}

}

/// НАСЛЕДНИКИ (СОРТИРОВКИ) ///

template<typename T>

class Buble\_sort : public BaseSort<T>

{

public:

//Конструктор(ы)

using BaseSort<T>::BaseSort;

~Buble\_sort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) override {

re = 0;

for (size\_t i = 0; i < v.size(); i++) {

for (size\_t j = 0; j < v.size() - 1; j++) {

if (v[j] > v[j + 1]) {

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

re++;

}

}

}

}

void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) override {}

};

template<typename T>

class Choice\_sort : public BaseSort<T>

{

public:

//Конструктор(ы)

using BaseSort<T>::BaseSort;

~Choice\_sort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) override {

re = 0;

for (size\_t i = 0; i < v.size() - 1; ++i)

{

size\_t smal = i;

for (size\_t curr = i + 1; curr < v.size(); ++curr)

{

if (v[curr] < v[smal])

smal = curr;

}

std::swap(v[i], v[smal]);

re++;

}

}

void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) override {}

};

template<typename T>

class Paste\_sort : public BaseSort<T>

{

public:

//Конструктор(ы)

using BaseSort<T>::BaseSort;

~Paste\_sort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) override {

re = 0;

for (size\_t i = 1; i < v.size(); i++) {

for (size\_t j = i; j > 0 && v[j - 1] > v[j]; j--) {

std::swap(v[j], v[j - 1]);

re++;

}

}

}

void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) override {}

};

template<typename T>

class Even\_odd\_sort : public BaseSort<T>

{

public:

//Конструктор(ы)

using BaseSort<T>::BaseSort;

~Even\_odd\_sort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) override {

re = 0;

for (size\_t i = 0; i < v.size(); i++) {

// (i % 2) ? 0 : 1 возвращает 1, если i четное, 0, если i не четное

for (size\_t i = (i % 2) ? 0 : 1; i + 1 < v.size(); i += 2) {

if (v[i] > v[i + 1]) {

std::swap(v[i], v[i + 1]);

re++;

}

}

}

}

void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) override {}

};

template<typename T>

class Shaker\_sort : public BaseSort<T>

{

public:

//Конструктор(ы)

using BaseSort<T>::BaseSort;

~Shaker\_sort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) override {

re = 0;

int control = v.size() - 1;

int left = 0;

int right = v.size() - 1;

do {

for (int i = left; i < right; i++) {

if (v[i] > v[i + 1]) {

std::swap(v[i], v[i + 1]);

control = i;

re++;

}

}

right = control;

for (int i = right; i > left; i--) {

if (v[i] < v[i - 1]) {

std::swap(v[i], v[i - 1]);

control = i;

re++;

}

}

left = control;

} while (left < right);

}

void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) override {}

};

template<typename T>

class Shell\_sort : public BaseSort<T>

{

public:

//Конструктор(ы)

using BaseSort<T>::BaseSort;

~Shell\_sort() {};

//Метод(ы)

virtual void my\_sort1(std::vector<T>& v, size\_t& re) override {

re = 0;

size\_t step, i, j;

auto temp = v[0];

for (step = v.size() / 2; step > 0; step /= 2) {

re++;

for (i = step; i < v.size(); i++)

{

re++;

temp = v[i];

for (j = i; j >= step; j -= step)

{

if (temp < v[j - step]) {

v[j] = v[j - step];

}

else

break;

}

v[j] = temp;

}

}

}

void my\_sort2(std::vector<T>& v, int first, int last, size\_t& re) override {}

};

}

Модуль MyForm.cpp:

#include <ctime>

#include "MyForm.h"

#include "MyForm1.h"

#include "MyForm2.h"

#include <Windows.h>

#include <vector>

#include <algorithm>

#include <string>

#include <fstream>

#include <chrono>

#include <msclr\marshal\_cppstd.h>

#include <exception>

#include <numeric>

#ifndef SORTCLASS\_H

#define SORTCLASS\_H

#include "SortClass.h"

#endif

size\_t re1, re2, re3, re4, re5, re6, re7, re8, re9, re10, re11, re12, re13;

SortClass::BaseSort<int>\* p;

SortClass::BaseSort<double>\* p2;

using namespace KR2;

[STAThreadAttribute]

int WINAPI WinMain(HINSTANCE, HINSTANCE, LPSTR, int) {

Application::EnableVisualStyles();

Application::SetCompatibleTextRenderingDefault(false);

Application::Run(gcnew MyForm);

return 0;

}

System::Void KR2::MyForm::dataGridView1\_CellContentClick(System::Object^ sender, System::Windows::Forms::DataGridViewCellEventArgs^ e) {

return System::Void();

}

System::Void KR2::MyForm::deleteEntryToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

if (dataGridView1->Rows->Count == 1)

MessageBox::Show(this, "Нельзя удалить единственную строчку!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

else

dataGridView1->Rows->RemoveAt(dataGridView1->CurrentRow->Index);

}

System::Void KR2::MyForm::contextMenuStrip1\_Opening(System::Object^ sender, System::ComponentModel::CancelEventArgs^ e) {

return System::Void();

}

System::Void KR2::MyForm::saveAsToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

SaveFileDialog^ saveFileDialog1 = gcnew SaveFileDialog;

saveFileDialog1->Filter = "txt files (\*.txt)|\*.txt|All files (\*.\*)|\*.\*";

test++;

try {

if (saveFileDialog1->ShowDialog() == System::Windows::Forms::DialogResult::OK && saveFileDialog1->FileName->Length > 0) {

IO::File::WriteAllText(saveFileDialog1->FileName, L"");

IO::File::AppendAllText(saveFileDialog1->FileName, "Тест № " + System::Convert::ToString(test) + Environment::NewLine + Environment::NewLine);

for (int i = 0; i < dataGridView1->Rows[i]->Cells->Count; i++) {

IO::File::AppendAllText(saveFileDialog1->FileName,

" Метод сортировки: " + dataGridView1->Rows[i]->Cells[0]->Value->ToString() + Environment::NewLine

+ " Количество элементов: " + dataGridView1->Rows[i]->Cells[1]->Value->ToString() + Environment::NewLine

+ " Время выполнения сортировки: " + dataGridView1->Rows[i]->Cells[2]->Value->ToString() + Environment::NewLine

+ " Количество перестановок: " + dataGridView1->Rows[i]->Cells[3]->Value->ToString() + Environment::NewLine + Environment::NewLine);

}

MessageBox::Show(this, "Отчет сохранен", "Готово", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

catch (Exception^ e) {

MessageBox::Show(this, "Файл пустой", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (std::out\_of\_range) {

MessageBox::Show(this, "Индекс вне диапозона!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

System::Void KR2::MyForm::button1\_Click(System::Object^ sender, System::EventArgs^ e) {

try {

if (comboBox1->Text == "целочисленный") {

p = new SortClass::Buble\_sort<int>(System::Convert::ToInt32(textBox2->Text));

if (p->v.size() <= 100) {

textBox1->Text = nullptr;

for (size\_t i = 0; i < p->v.size(); i++) {

textBox1->Text = textBox1->Text + System::Convert::ToString(p->v[i]) + " ";

}

MessageBox::Show(this, "Массив успешно создан", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

textBox1->Text = nullptr;

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

if (comboBox1->Text == "вещественный") {

p2 = new SortClass::Buble\_sort<double>(System::Convert::ToInt32(textBox2->Text));

if (p2->v.size() <= 100) {

textBox1->Text = nullptr;

for (size\_t i = 0; i < p2->v.size(); i++) {

textBox1->Text = textBox1->Text + System::Convert::ToString(p2->v[i]) + " ";

}

MessageBox::Show(this, "Массив успешно создан", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

textBox1->Text = nullptr;

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

catch (Exception^ e) {

textBox1->Text = nullptr;

MessageBox::Show(this, "Некоректные данные", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

System::Void KR2::MyForm::toolStripButton1\_Click(System::Object^ sender, System::EventArgs^ e) {

p = new SortClass::Buble\_sort<int>(10);

if (p->v.size() <= 100) {

textBox1->Text = nullptr;

for (size\_t i = 0; i < p->v.size(); i++) {

textBox1->Text = textBox1->Text + System::Convert::ToString(p->v[i]) + " ";

}

comboBox1->Text = "целочисленный";

textBox2->Text = System::Convert::ToString(10);

MessageBox::Show(this, "Массив успешно создан", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

textBox1->Text = nullptr;

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

System::Void KR2::MyForm::toolStripButton2\_Click(System::Object^ sender, System::EventArgs^ e) {

p = new SortClass::Buble\_sort<int>(100);

if (p->v.size() <= 100) {

textBox1->Text = nullptr;

for (size\_t i = 0; i < p->v.size(); i++) {

textBox1->Text = textBox1->Text + System::Convert::ToString(p->v[i]) + " ";

}

comboBox1->Text = "целочисленный";

textBox2->Text = System::Convert::ToString(100);

MessageBox::Show(this, "Массив успешно создан", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

textBox1->Text = nullptr;

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

System::Void KR2::MyForm::toolStripButton3\_Click(System::Object^ sender, System::EventArgs^ e) {

p = new SortClass::Buble\_sort<int>(1000);

if (p->v.size() <= 100) {

textBox1->Text = nullptr;

for (size\_t i = 0; i < p->v.size(); i++) {

textBox1->Text = textBox1->Text + System::Convert::ToString(p->v[i]) + " ";

}

comboBox1->Text = "целочисленный";

textBox2->Text = System::Convert::ToString(1000);

MessageBox::Show(this, "Массив успешно создан", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

System::Void KR2::MyForm::toolStripButton4\_Click(System::Object^ sender, System::EventArgs^ e) {

Visual\_menu::MyForm1^ visual = gcnew Visual\_menu::MyForm1();

visual->Show();

}

System::Void KR2::MyForm::справкаToolStripButton\_Click(System::Object^ sender, System::EventArgs^ e) {

reference::MyForm2^ visual = gcnew reference::MyForm2();

visual->Show();

}

System::Void KR2::MyForm::button3\_Click(System::Object^ sender, System::EventArgs^ e) {

double quantity = System::Convert::ToDouble(textBox2->Text);

try {

using std::chrono::microseconds;

if (comboBox2->Text == "пузырьковая") {

if (comboBox1->Text == "целочисленный") {

SortClass::BaseSort<int>\* p3;

p3 = new SortClass::Buble\_sort<int>(0);

p3->v = p->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re1 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re1 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "пузырьковая";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re1);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

if (comboBox1->Text == "вещественный") {

SortClass::BaseSort<double>\* p3;

p3 = new SortClass::Buble\_sort<double>(0);

p3->v = p2->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re1 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p2->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p2->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re1 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "пузырьковая";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re1);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

if (comboBox2->Text == "выбором") {

if (comboBox1->Text == "целочисленный") {

SortClass::BaseSort<int>\* p3;

p3 = new SortClass::Choice\_sort<int>(0);

p3->v = p->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re2 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re2 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "выбором";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re2);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

if (comboBox1->Text == "вещественный") {

SortClass::BaseSort<double>\* p3;

p3 = new SortClass::Choice\_sort<double>(0);

p3->v = p2->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re2 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p2->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p2->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re2 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "выбором";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re2);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

if (comboBox2->Text == "вставками") {

if (comboBox1->Text == "целочисленный") {

SortClass::BaseSort<int>\* p3;

p3 = new SortClass::Paste\_sort<int>(0);

p3->v = p->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re3 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re3 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "вставками";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re3);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

if (comboBox1->Text == "вещественный") {

SortClass::BaseSort<double>\* p3;

p3 = new SortClass::Paste\_sort<double>(0);

p3->v = p2->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re3 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p2->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p2->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re3 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "вставками";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re3);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

if (comboBox2->Text == "чет-нечет") {

if (comboBox1->Text == "целочисленный") {

SortClass::BaseSort<int>\* p3;

p3 = new SortClass::Even\_odd\_sort<int>(0);

p3->v = p->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re4 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re4 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "чет-нечет";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re4);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

if (comboBox1->Text == "вещественный") {

SortClass::BaseSort<double>\* p3;

p3 = new SortClass::Even\_odd\_sort<double>(0);

p3->v = p2->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re4 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p2->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p2->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re4 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "чет-нечет";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re4);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

if (comboBox2->Text == "шейкерная") {

if (comboBox1->Text == "целочисленный") {

SortClass::BaseSort<int>\* p3;

p3 = new SortClass::Shaker\_sort<int>(0);

p3->v = p->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re5 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re5 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "шейкерная";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re5);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

if (comboBox1->Text == "вещественный") {

SortClass::BaseSort<double>\* p3;

p3 = new SortClass::Shaker\_sort<double>(0);

p3->v = p2->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re5 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p2->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p2->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re5 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "шейкерная";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re5);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

if (comboBox2->Text == "Шелла") {

if (comboBox1->Text == "целочисленный") {

SortClass::BaseSort<int>\* p3;

p3 = new SortClass::Shell\_sort<int>(0);

p3->v = p->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re10 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re10 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "Шелла";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re10);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

if (comboBox1->Text == "вещественный") {

SortClass::BaseSort<double>\* p3;

p3 = new SortClass::Shell\_sort<double>(0);

p3->v = p2->v;

auto fail\_start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re10 = 0);//

auto fail\_end = std::chrono::system\_clock::now();//

auto fail\_res = std::chrono::duration\_cast<std::chrono::microseconds>(fail\_end - fail\_start).count();//

p3->v = p2->v;

if (!std::equal(p3->v.begin(), p3->v.end(), p2->v.begin())) throw;

auto start = std::chrono::system\_clock::now();

p3->my\_sort1(p3->v, re10 = 0);

auto end = std::chrono::system\_clock::now();

dataGridView1->Rows->Add();

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[0]->Value = "Шелла";

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[1]->Value = System::Convert::ToString(p3->v.size());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[2]->Value = System::Convert::ToString(std::chrono::duration\_cast<std::chrono::microseconds>(end - start).count());

dataGridView1->Rows[dataGridView1->Rows->Count - 2]->Cells[3]->Value = System::Convert::ToString(re10);

textBox3->Text = nullptr;

if (p3->v.size() <= 100) {

for (size\_t i = 0; i < p3->v.size(); i++) {

textBox3->Text = textBox3->Text + System::Convert::ToString(p3->v[i]) + " ";

}

}

else

MessageBox::Show(this, "Массив более чем из 100 элементов", "Уведомление", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

}

}

}

catch (SortClass::BaseSort<int>::MyException)

{

MessageBox::Show(this, "Контрольная сумма неверна.", "Алгоритм не стабилен!", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (Exception^ e) {

MessageBox::Show(this, "Некоректные данные!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (std::bad\_alloc) {

MessageBox::Show(this, "Ошибка распределения памяти!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

catch (std::out\_of\_range) {

MessageBox::Show(this, "Индекс вне диапозона!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

System::Void KR2::MyForm::button4\_Click(System::Object^ sender, System::EventArgs^ e) {

try

{

if (dataGridView1->Rows->Count < 3) {

SortClass::BaseSort<int>\* excep;

excep->exception();

}

else

{

double x, y;

String^ str\_x, ^ str\_y;

this->chart1->Series[0]->Points->Clear();

this->chart2->Series[0]->Points->Clear();

//перестановки

for (int i = 0; i < dataGridView1->Rows->Count - 1; i++) {

if (dataGridView1->Rows[i]->Cells[0]->Value->ToString() == dataGridView1->Rows[0]->Cells[0]->Value->ToString()) {

str\_x = dataGridView1->Rows[i]->Cells[1]->Value->ToString();

str\_y = dataGridView1->Rows[i]->Cells[3]->Value->ToString();

x = System::Convert::ToDouble(str\_x);

y = System::Convert::ToDouble(str\_y);

this->chart1->Series[0]->Points->AddXY(x, y);

}

}

//время

for (int i = 0; i < dataGridView1->Rows->Count - 1; i++) {

if (dataGridView1->Rows[i]->Cells[0]->Value->ToString() == dataGridView1->Rows[0]->Cells[0]->Value->ToString()) {

str\_x = dataGridView1->Rows[i]->Cells[1]->Value->ToString();

str\_y = dataGridView1->Rows[i]->Cells[2]->Value->ToString();

x = System::Convert::ToDouble(str\_x);

y = System::Convert::ToDouble(str\_y);

this->chart2->Series[0]->Points->AddXY(x, y);

}

}

}

}

catch (SortClass::BaseSort<int>::MyException)

{

MessageBox::Show(this, "Недостаточно данных", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

System::Void KR2::MyForm::openFileDialog1\_FileOk(System::Object^ sender, System::ComponentModel::CancelEventArgs^ e) {

return System::Void();

}

System::Void KR2::MyForm::saveFileDialog1\_FileOk(System::Object^ sender, System::ComponentModel::CancelEventArgs^ e) {

return System::Void();

}

System::Void KR2::MyForm::editToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

dataGridView1->Rows[dataGridView1->CurrentRow->Index]->Cells[dataGridView1->CurrentCell->ColumnIndex]->ReadOnly = false;

}

System::Void KR2::MyForm::findToolStripMenuItem\_Click(System::Object^ sender, System::EventArgs^ e) {

if (dataGridView1->Rows[dataGridView1->CurrentRow->Index]->Cells[dataGridView1->CurrentCell->ColumnIndex]->Value == nullptr)

MessageBox::Show(this, "Ячейка пуста!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

else {

for (int i = 0; i < dataGridView1->Rows->Count; i++)

for (int j = 0; j < dataGridView1->Rows[i]->Cells->Count; j++)

dataGridView1->Rows[i]->Cells[j]->Style->BackColor = BackColor.LightGray;

String^ find\_str = dataGridView1->Rows[dataGridView1->CurrentRow->Index]->Cells[dataGridView1->CurrentCell->ColumnIndex]->Value->ToString();

int count = 0;

for (int i = 0; i < dataGridView1->Rows->Count; i++)

if (dataGridView1->Rows[i]->Cells[dataGridView1->CurrentCell->ColumnIndex]->Value->ToString() == find\_str) {

dataGridView1->Rows[i]->Cells[dataGridView1->CurrentCell->ColumnIndex]->Style->BackColor = BackColor.Green;

count++;

}

if (count > 1) {

String^ answ = "Найдено похожих элементов: " + count.ToString();

MessageBox::Show(answ, "Результат", MessageBoxButtons::OK, MessageBoxIcon::Information);

}

else {

for (int i = 0; i < dataGridView1->Rows->Count; i++)

for (int j = 0; j < dataGridView1->Rows[i]->Cells->Count; j++)

dataGridView1->Rows[i]->Cells[j]->Style->BackColor = BackColor.LightGray;

MessageBox::Show(this, "Похожих элементов не обнаружено!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

}

}

}

System::Void KR2::MyForm::toolStripMenuItem2\_Click(System::Object^ sender, System::EventArgs^ e) {

if (dataGridView1->Rows->Count == 1)

MessageBox::Show(this, "Нельзя удалить единственную строчку!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

else

dataGridView1->Rows->RemoveAt(dataGridView1->CurrentRow->Index);

}

System::Void KR2::MyForm::toolStripMenuItem1\_Click(System::Object^ sender, System::EventArgs^ e) {

if (dataGridView1->Rows->Count == 1)

MessageBox::Show(this, "Нельзя удалить единственную строчку!", "Ошибка", MessageBoxButtons::OK, MessageBoxIcon::Error);

else

for (int i = dataGridView1->Rows->Count - 2; i >= 0; i--) {

dataGridView1->Rows->RemoveAt(dataGridView1->Rows[i]->Index);

}

}

System::Void KR2::MyForm::MyForm\_KeyPress(System::Object^ sender, System::Windows::Forms::KeyPressEventArgs^ e) {

if (e->KeyChar == 's') {

button3->PerformClick();

}

if (e->KeyChar == 'g') {

button4->PerformClick();

}

if (e->KeyChar == 'v') {

Visual\_menu::MyForm1^ visual = gcnew Visual\_menu::MyForm1();

visual->Show();

}

}