**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ**

**РОССИЙСКОЙ ФЕДЕРАЦИИ**

**ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ**

**Кафедра «Математическое обеспечение и применение ЭВМ»**

Курсовая работа

по дисциплине «Структуры и алгоритмы обработки данных»

на тему «Сравнение и анализ алгоритмов сортировки двухсвязных списков. Эффективные алгоритмы сортировки»

ПГУ 09.03.04 - 03КР212.10 ПЗ

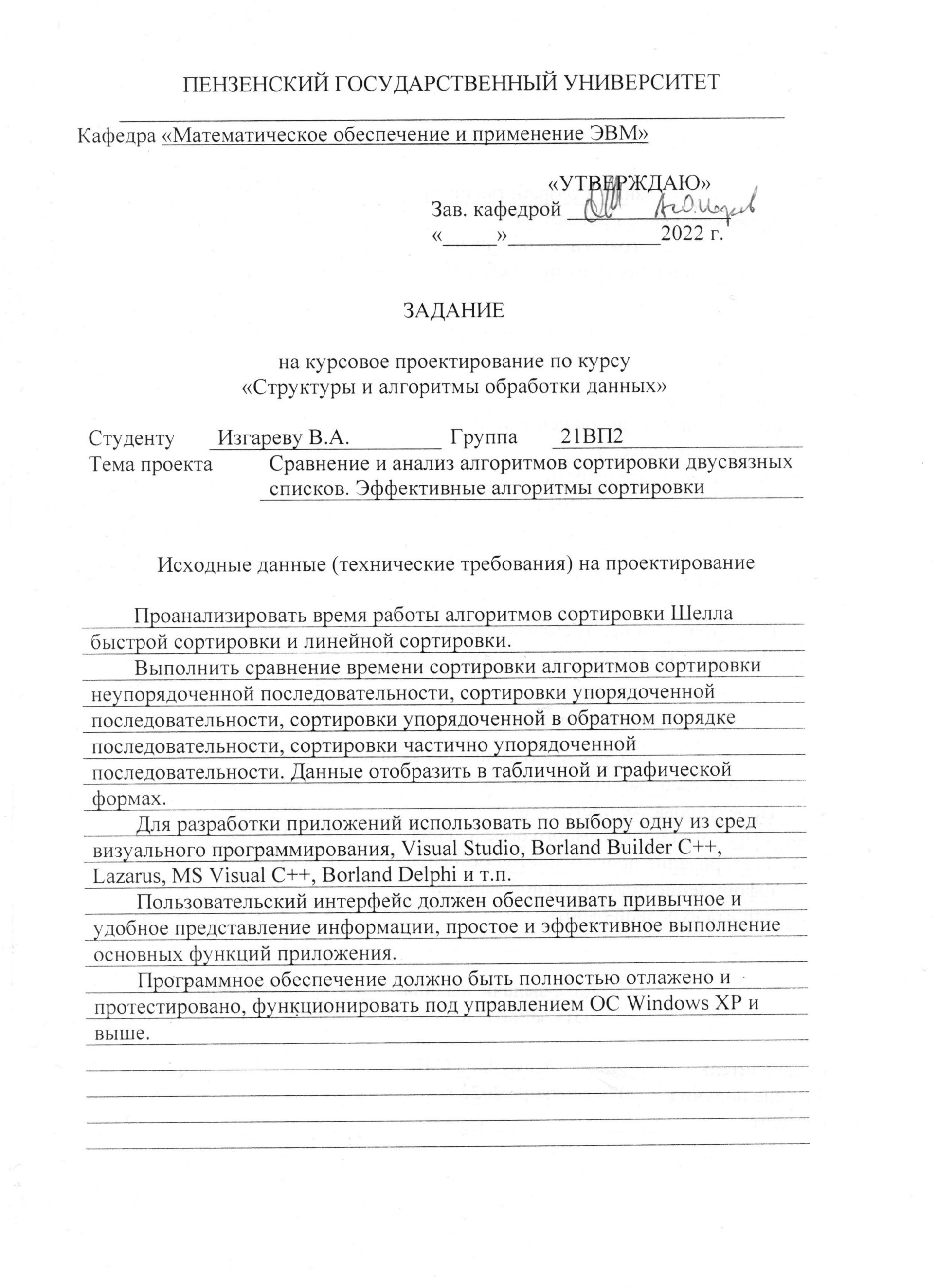
Направление подготовки – 09.03.04 Программная инженерия

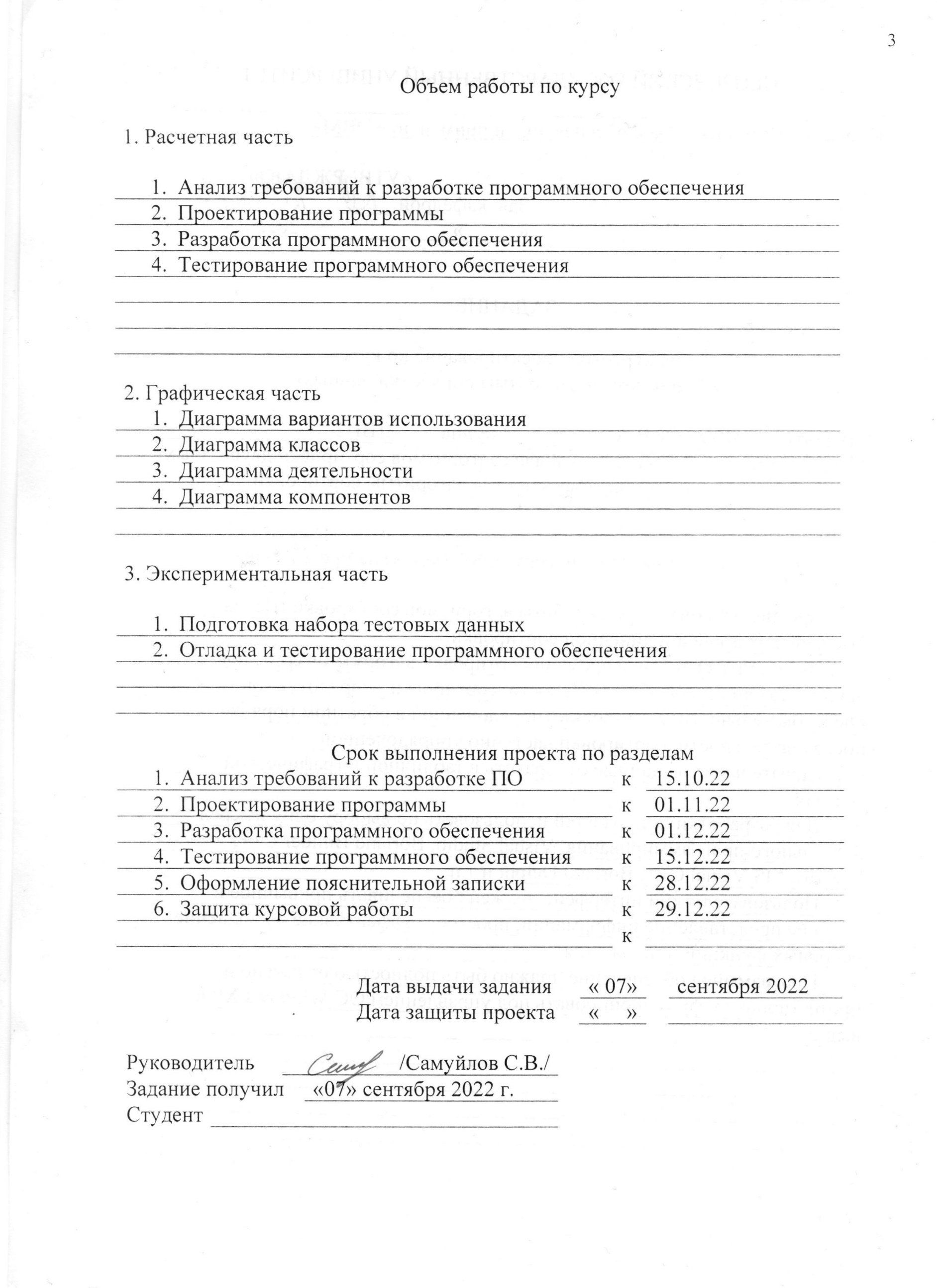
Профиль подготовки – Программная инженерия

|  |  |  |
| --- | --- | --- |
| Выполнил студент |  | Изгарев В.А. |
| Группа |  | 21ВП2 |
|  |  |  |
| Руководитель |  |  |
| к.т.н., доцент |  | Самуйлов С.В. |

|  |  |
| --- | --- |
| Работа защищена с оценкой |  |
| Преподаватели |  |
|  |
| Дата защиты |  |

2022





**Реферат**

Пояснительная записка содержит 47 листа 17 рисунков, 4 таблицы, 11 использованных источников, 1 приложение.

СОРТИРОВКА, СОРТИРОВКА ШЕЛЛА, БЫСТРАЯ СОРТИРОВКА, ЛИНЕЙНАЯ СОРТИРОВКА, RUP, ФУНКЦИОНАЛЬНОЕ ТЕСТИРОВАНИЕ, ДВУСВЯЗНЫЙ СПИСОК, СТРУКТУРА ДАННЫХ, ХУДШЕЕ ВРЕМЯ, ВЫДАЮЩИЙСЯ РЕЗУЛЬТАТ.

Объектом исследования являются эффективные алгоритмы сортировок для двусвязного списка.

Целью курсовой работы является разработка приложения для сравнения и анализа алгоритмов сортировок для двусвязного списка.

Разработка проводилась на языке программирования С# в среде разработки Visual Studio 2022.

Осуществлено функциональное тестирование разработанного программного обеспечения, которое показало корректность его работы.

Содержание

[Введение 6](#_Toc122205338)

[1. Постановка задачи и анализ предметной области 8](#_Toc122205339)

[1.1 Основные понятия и определения 8](#_Toc122205340)

[1.2 Постановка задачи на разработку программы 9](#_Toc122205341)

[1.3 Анализ требований 9](#_Toc122205342)

[1.3.1 Требования к интерфейсу пользователя 9](#_Toc122205343)

[1.3.2 Требования к структуре данных 10](#_Toc122205344)

[1.3.3 Требования к программным средствам 11](#_Toc122205345)

[1.4 Технология разработки программного обеспечения 13](#_Toc122205346)

[2. Проектирование программы 15](#_Toc122205347)

[2.1. Модель интерфейса 15](#_Toc122205348)

[2.2. Проектирование структур данных 16](#_Toc122205349)

[2.3. Структура программного обеспечения 19](#_Toc122205350)

[3. Реализация программы 21](#_Toc122205351)

[3.1. Кодирование 21](#_Toc122205352)

[3.2. Диаграмма компонентов 22](#_Toc122205353)

[4. Тестирование 24](#_Toc122205354)

[4.1. Виды тестирования программных средств 24](#_Toc122205355)

[4.2. Функциональное тестирование программы 27](#_Toc122205356)

[5. Анализ результатов 33](#_Toc122205357)

[Заключение 36](#_Toc122205358)

[Список использованных источников 37](#_Toc122205359)

[Приложение А. 38](#_Toc122205360)

# Введение

В настоящие время с каждой секундой растет большое количество информации, которую необходимо обрабатывать. Есть множество различных алгоритмов сортировки, которые активно используются в обработке различной информации, чтобы в дальнейшем облегчить поиск.

Алгоритм сортировки – это алгоритм для упорядочивания элементов в какой-либо структуре данных[1]. Мы знаем, что одну и ту же задачу можно решить с помощью разных алгоритмов, и каждый раз изменение алгоритма приводит к новым, более или менее эффективным решениям задачи. Основными требованиями к эффективности алгоритмов сортировки является, прежде всего, эффективность по времени и экономное использование памяти.

Цель разрабатываемой программы является сравнение времени работы эффективных алгоритмов, таких как сортировка Шелла, быстрая сортировка, линейная сортировка в такой структуре данных, как двусвязный список.

Программа будет выводит время работы всех сортировок в табличный и в графический вид, что позволяет нам провести анализ полученных результатов.

Алгоритмы сортировки будут сортировать следующие типы последовательности двусвязного списка:

* Неупорядоченный
* Упорядоченный
* Упорядоченный в обратном порядке
* Частично упорядоченный

Для достижения цели курсовой работы были выделены следующие подзадачи:

1. Проанализировать поставленные требования;
2. Разработать интерфейс программы;
3. Разработать функционал для сравнения алгоритмов;
4. Реализовать отображение результатов в табличной и графической формах;
5. Протестировать разработанную программу.
6. Провести анализ результатов

При создании программы был использован такой язык программирования, как C# и среда разработки под названием «Visual Studio 2022».

# Постановка задачи и анализ предметной области

## Основные понятия и определения

**Сортировка** – это процесс расстановки элементов в некотором порядке[2].

**Сортировка Шелла** – алгоритм сортировки, являющийся усовершенствованным вариантом сортировки вставками[3]. Идея сортировки методом Шелла состоит в том, чтобы сортировать элементы, отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1[4]. Худшее время – O(n2). Лучшее время – О(n log2(n)). Среднее время зависит от выбранных шагов.

**Быстрая сортировка** – является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена, известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы. Общая идея алгоритма состоит в следующем: выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: «элементы меньшие опорного», «равные» и «большие». Для отрезков «меньших» и «больших» значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы. Лучшее время — O(nlog n).  
Худшее время — O(n2)[5].

**Линейная сортировка** — алгоритм сортировки, в котором используется диапазон чисел сортируемого массива (списка) для подсчёта совпадающих элементов[6]. Главная идея алгоритма — посчитать, сколько раз встречается каждый элемент в массиве, а потом заполнить исходный массив результатами этого подсчёта. Для этого нам нужен вспомогательный массив, где мы будем хранить результаты подсчётах[7]. Время выполнения алгоритма — O(n + k), где k - диапазон неотрицательных значений ключа.

**Структура данных** –программная единица, позволяющая хранить и обрабатывать однотипные и/или логически связанные данные. Для добавления, поиска, изменения и удаления данных структура данных предоставляет некоторый набор функций, составляющих её интерфейс[8].

## Постановка задачи на разработку программы

Проанализировать время работы алгоритмов сортировки Шелла, быстрой сортировки и линейной сортировки.

Выполнить сравнение времени сортировки алгоритмов сортировки неупорядоченной последовательности, сортировки упорядоченной последовательности, сортировки упорядоченной в обратном порядке последовательности, сортировки частично упорядоченной последовательности. Данные отобразить в табличной и графической формах.

Для разработки приложении решено использовать Visual Studio.

Пользовательский интерфейс должен обеспечивать привычное и удобное представление информации, простое и эффективное выполнение основных функций приложения.

Программное обеспечение должно быть полностью отлажено и протестировано, функционировать под управление ОС Windows 10.

## Анализ требований

### Требования к интерфейсу пользователя

Интерфейс пользователя – интерфейс, обеспечивающий передачу информации между пользователем-человеком и программно-аппаратными компонентами компьютерной системы.

Приложение должно предоставлять пользователю удобный интерфейс пользователя, позволяющий вводить размерность двусвязного списка, выбирать степень упорядоченности, а именно неупорядоченный, упорядоченный, упорядоченный в обратном порядке, частично упорядоченный с возможностью указать процент упорядоченности двусвязного списка. Сравнивать между собой алгоритмы сортировки Шелла, быстрой сортировки, линейной сортировки. Выводить результаты выполнения программы в графический и табличный вид.

### Требования к структуре данных

В данной курсовой работе используется структура данных, которая называется двусвязный список. Двусвязный список – связная структура данных, состоящая из набора последовательно связанных записей, называемых узлами[9]. Каждый узел содержит всего три поля, два из которых являются ссылками и указывают на предыдущий и последующий элементы в последовательности узлов, а третье поле является информационным. (Рисунок 1).

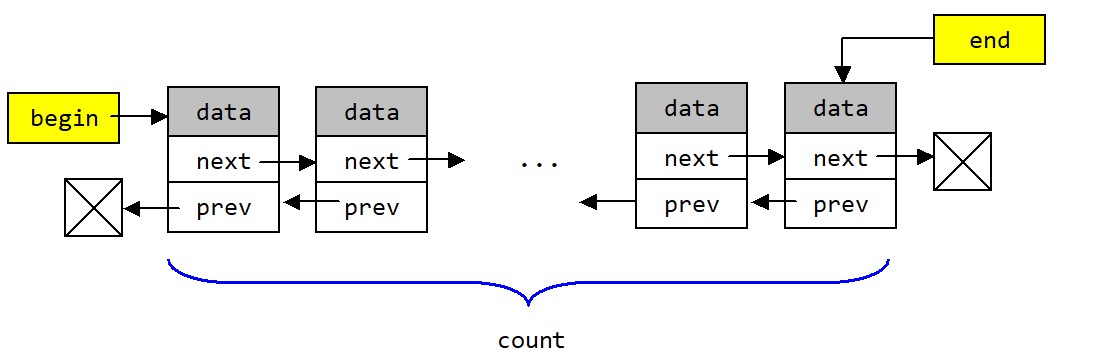


Рисунок 1 – Логическое представление двусвязного списка

Основные операции над двусвязным списком:

* Вставка, то есть добавление элемента в начало списка.
* Удаление элемента из начала списка.
* Вставка последним, то есть добавление элемента в конец списка.
* Удаление последнего, то есть удаление элемента из конца списка.
* Вставка после, то есть добавление элемента после какого-то элемента списка.
* Удаление элемента из списка по заданному ключу.
* Поиск элемента по заданному ключу.
* Отображение вперед полного списка в прямом порядке.
* Отображение назад полного списка в обратном порядке.

### Требования к программным средствам

Программа должна формировать 4 типа двусвязного списка: неупорядоченный, упорядоченный, упорядоченной в обратном порядке, частично упорядоченный. Выполнять 3 сортировки двусвязного списка: сортировку Шелла, быструю сортировку, линейную сортировку. Выводить результат выполнения в интерфейс пользователя. Кроме того, от программы требуется сравнить результаты алгоритмов сортировки в виде графической и табличной формы.

Диаграмма вариантов использования (ДВИ)–диаграмма, отражающая отношения между актерами и прецедентами и являющаяся составной частью модели прецедентов, позволяющей описать систему на концептуальном уровне.

На основе вышеперечисленного функционала была сформирована диаграмма вариантов использования[10] (Рисунок 2).

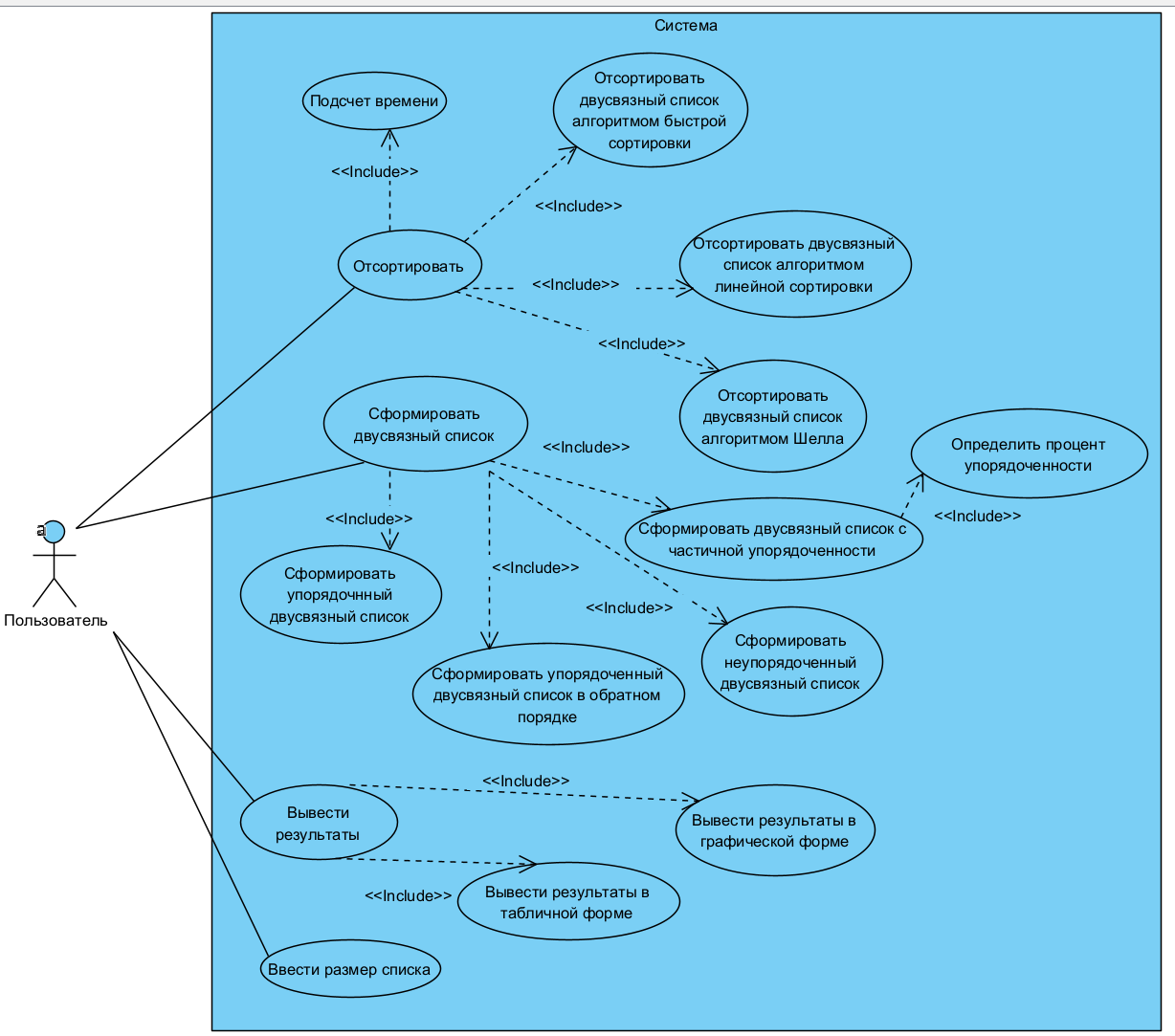


Рисунок 2 – Диаграмма вариантов использования

Описания сценариев вариантов использования, а именно «Отсортировать двусвязный список алгоритмом быстрой сортировки», «Сформировать неупорядоченный двусвязный список», «Ввести размер списка» приведены в таблицах 1 – 3[10].

Таблица 1 – Сценарий варианта использования «Отсортировать двусвязный список алгоритмом быстрой сортировки»

|  |
| --- |
| **Наименование: отсортировать двусвязный список алгоритмом быстрой сортировки** |
| **Краткое описание:** система сортирует двусвязный список алгоритмом быстрой сортировки |
| **Действующие лица:** пользователь |
| **Предусловие:** двусвязный список сформирован |
| **Основной поток:**   1. Система сортирует двусвязный список алгоритмом быстрой сортировки |
| **Постусловие:** двусвязный список отсортирован алгоритмом быстрой сортировки |

Таблица 2 – Сценарий варианта использования «Сформировать неупорядоченный двусвязный список»

|  |
| --- |
| **Наименование: сформировать неупорядоченный двусвязный список** |
| **Краткое описание:** система формирует неупорядоченный двусвязный список |
| **Действующие лица:** пользователь |
| **Предусловие: -** |
| **Основной поток:**   1. В систему передается размерность двусвязного списка, которую указал пользователь 2. Система формирует неупорядоченный двусвязный список |
| **Постусловие:** неупорядоченный двусвязный список создан |

Таблица 3– Сценарий варианта использования «Ввести размер списка»

|  |
| --- |
| **Наименование: ввести размер списка** |
| **Краткое описание:** система запрашивает у пользователя размер двусвязного списка |
| **Действующие лица:** пользователь |
| **Предусловие: -** |
| **Основной поток:**   1. Пользователь вводит размерность двусвязного списка |
| **Постусловие: -** |

## Технология разработки программного обеспечения

**RAD** (Rapid Application Development) – концепция организации технологического процесса разработки программных продуктов, ориентированная на максимально быстрое получение результата в условиях сильных ограничений по срокам и бюджету и нечётко определённых требований к продукту. Эффект ускорения разработки достигается путём использования соответствующих технических средств и непрерывного, параллельного с ходом разработки, уточнения требований и оценки текущих результатов с привлечением заказчика. RAD создана в конце 1980-х как альтернатива более ранним каскадной и итеративной моделям. С конца XX века RAD получила широкое распространение.

Принципы RAD технологии направлены на обеспечение трёх основных её преимуществ – высокой скорости разработки, низкой стоимости и высокого качества. Достигнуть высокого качества программного продукта весьма непросто и одна из главных причин возникающих трудностей заключается в том, что разработчик и заказчик видят предмет разработки (ПО) по-разному.

**RUP** (Rational Unified Process) – методология разработки программного обеспечения, созданная компанией Rational Software. RUP использует итеративную модель разработки. В конце каждой итерации (в идеале продолжающейся от 2 до 6 недель) проектная команда должна достичь запланированных на данную итерацию целей, создать или доработать проектные артефакты и получить промежуточную, но функциональную версию конечного продукта. Итеративная разработка позволяет быстро реагировать на меняющиеся требования, обнаруживать и устранять риски на ранних стадиях проекта, а также эффективно контролировать качество создаваемого продукта. Первые идеи итеративной модели разработки были заложены в «спиральной модели».

В нашей курсовой работе используется технология RUP, потому что в ней были выделены следующие преимущества:

* Итеративный подход приспособлен к меняющимся требованиям
* Своевременная интеграция
* Риски обнаруживаются и устраняются на ранних итерациях
* Облегчается повторное использование
* Дефекты можно найти и исправить за несколько итераций

# Проектирование программы

## Модель интерфейса

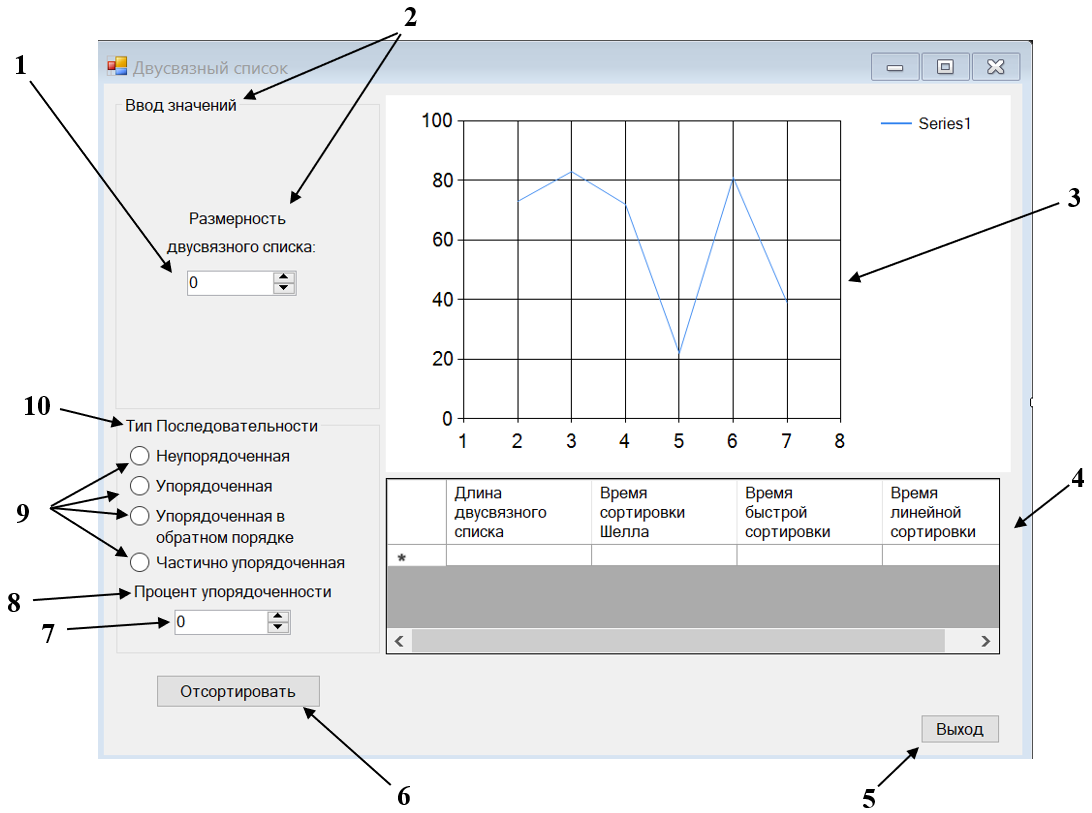
После анализа требований интерфейса пользователя, была сформирована следующая модель интерфейса(рисунок 3).

Рисунок 3 – Интерфейс программы

GroupBox(2) – используется, для отделения ввода пользователя от других элементов интерфейса. В нем расположен Label(2), который указывает на то, что пользователь может задавать размерность списка в NumericUpDown(1).

Chart(3) – позволяет выводить время работы сортировок в графическом виде, что позволяет их сравнить между собой.

DataGridView(4) – таблица, которая имеет 4 столбца, а именно длина двусвязного списка и время работы всех сортировок, указанных в задании. Строки будут добавлять по мере количества выполненных сортировок в программе.

Button(5) – кнопка, которая позволяет пользователю закрыть программу.

Button(6) – кнопка, которая позволяет запустить сортировки двусвязного списка и по их окончании выводит время работы в табличном и графическом виде.

GroupBox(10) – контейнер, содержащий компоненты, которые позволяют выбрать тип последовательности списка. В нем находятся такие компоненты, как RadioButton(9), с помощью которых пользователь выбирает соответствующий тип последовательности и Label(8), который указывает, на то, что пользователь может, с помощью NumericUpDown(7), задать процент упорядоченности, для частично упорядоченного двусвязного списка.

## Проектирование структур данных

В данной курсовой работе, чтобы реализовать такую структура данных, как двусвязный список на языке С#, мы используем класс LinkedList.

const int n = 10000;

LinkedList<int> double\_list = new LinkedList<int>();

LinkedList<int> double\_list\_sort = new LinkedList<int>();

Random rnd = new Random();

int user\_size, percent;

В результате этой строки, мы имеем созданные двусвязные списки или объекты от класса LinkedList. N – константное значение, количества элементов в double\_list, double\_list – исходный двусвязный список, которому мы будем задавать тип последовательности в соответствии с выбором пользователя и из которого будем копировать значения в double\_list\_sort, double\_list\_sort – двусвязный список, в котором будут работать поочередно три алгоритма сортировки, а именно сортировка Шелла, быстрая сортировка, линейная сортировка, rnd – генератор случайных чисел, user\_size – размер двусвязного списка, задаваемый пользователем, percent – процент упорядоченности, который задает пользователь.

Далее для двусвязного списка нужно реализовать 4 вида последовательности, указанных в задании, а именно неупорядоченная, упорядоченная, упорядоченная в обратном порядке, частично упорядоченная.

Код неупорядоченной последовательности:

user\_size = (int)numericUpDown1.Value;

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

{

double\_list.AddLast(rnd.Next(1, user\_size));

}

В этом типе последовательности мы берем константу N, и формируем цикл, в котором двусвязный список будет заполнять случайными числами в интервале от 0 до значения переменной user\_size, которую указывает пользователь.

Код упорядоченной последовательности:

user\_size = (int)numericUpDown1.Value;

double\_list.AddLast(rnd.Next(user\_size));

for (int i = 1; i < N; i++)

{

double\_list.AddLast(double\_list.ElementAt(i - 1) + rnd.Next(1, 5));

}

Для формирования упорядоченности, нам нужно проинициализировать самый первый элемент случайным числом до цикла, что мы и делаем, во второй строчке. Далее, чтобы заполнить двусвязный список, и он был упорядоченным, нужно в цикле достать значение предыдущего элемента и прибавить к нему случайное число в интервале от 1 до 5.

Код упорядоченной последовательности в обратном порядке:

user\_size = (int)numericUpDown1.Value;

double\_list.AddLast(user\_size);

for (int i = 1; i < N; i++)

{

double\_list.AddLast(double\_list.ElementAt(i - 1) - rnd.Next(1, 5));

}

В этой последовательности мы также инициализируем первый элемент случайным числом от 0 до переменной user\_size, которое указал пользователем. И далее по аналогии с предыдущем типом последовательности, мы достаем предыдущий элемент, но только не прибавляем к нему, а отнимаем у него случайное число в интервале от 1 до 5.

Код частичной упорядоченной последовательности:

percent = (int)numericUpDown2.Value;

user\_size = (int)numericUpDown1.Value;

int count\_yp = (N \* percent) / 100;

double\_list.AddLast(rnd.Next(0, user\_size));

for (int i = 1; i < count\_yp; i++)

{

double\_list.AddLast(double\_list.ElementAt(i - 1) + rnd.Next(1, 10));

}

for (int i = count\_yp; i < N; i++)

{

double\_list.AddLast(rnd.Next(0, user\_size));

}

В данной последовательности мы берем процент и константный размер N двусвязного списка, умножаем их и делим полученный результат на 100. Таким образом, мы получаем количество элементов, которые должны быть упорядочены. С помощью первого цикла, мы заполняем двусвязный список упорядоченно до последнего элемента, который мы получили в ходе ранних вычислений(count\_yp = (N \* percent) / 100), а далее, оставшиеся место в двусвязном списке, с помощью второго цикла, заполняем как неупорядоченный.

## Структура программного обеспечения

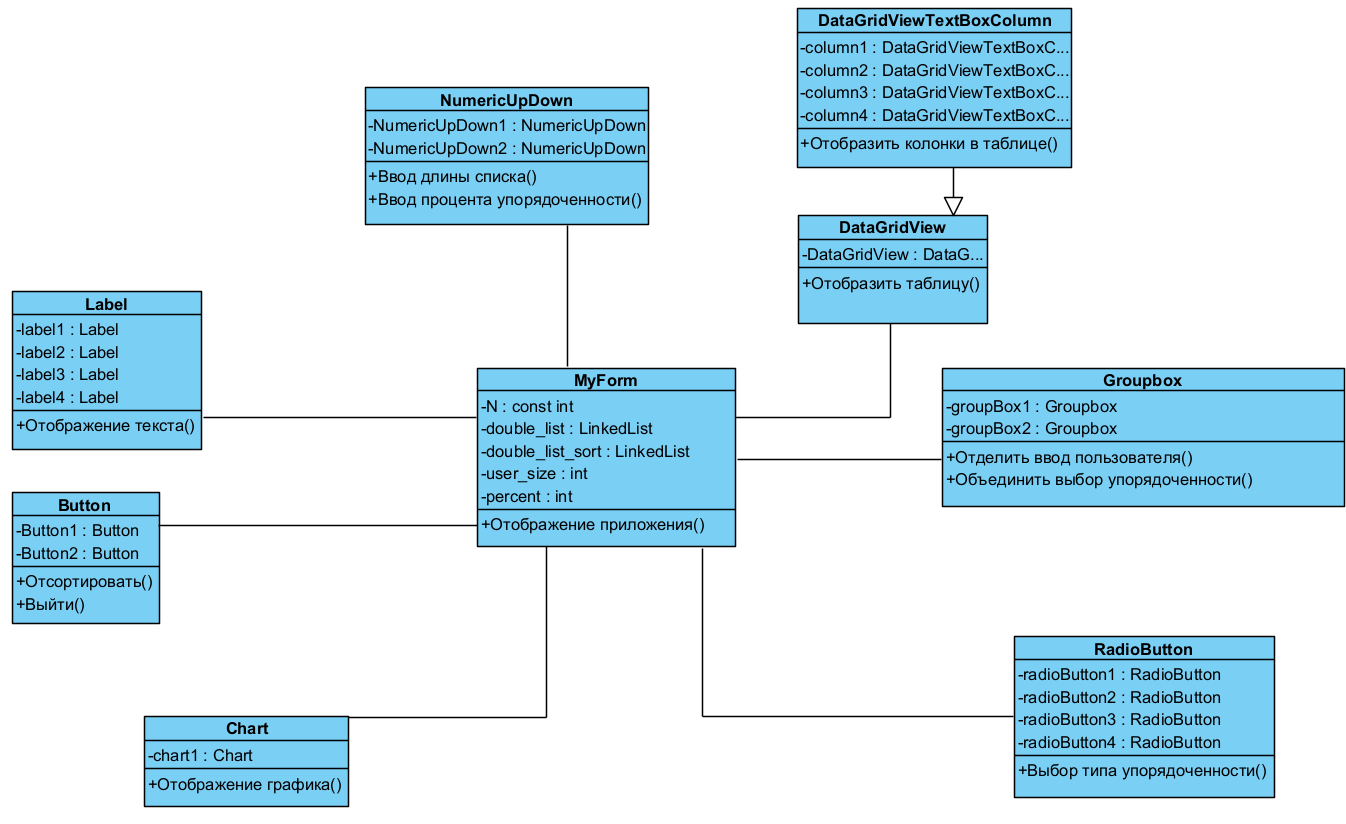
В ходе выполнения курсовой работы была сформирована диаграмма классов на рисунке 4[10].

Рисунок 4 – Диаграмма классов

Основной класс – форма MyForm. В ней содержатся атрибуты, такие как, три двусвязных списка, их размерность и процент частичной упорядоченности. Методом является отображение приложения. Она имеет ассоциации почти со всеми остальными классами.

Класс Label используется для отображения текстовой информации.

Класс Button предназначен для выхода из программы и запуска сортировок. Имеет такие атрибуты как button1 и button2. Методами являются отсортировать двусвязные списки и выйти из программы.

Класс Chart нужен для отображения времени работы сортировок в графическом виде. Имеет атрибут chart1 и единственный метод, который позволяет отобразить график в программе.

Класс RadioButton позволяет выбрать пользователю один тип последовательности из четырех. Имеет четыре атрибута, где каждый атрибут соответствует своему типу последовательности и единственный метод, позволяющий выбрать тип последовательности двусвязного списка.

Класс GroupBox предназначен для объединения компонентов и отделения их от основной части интерфейса. Имеет два атрибута и имеет два метода, где первый метод предназначен для отделения ввода пользователя от основной части интерфейса, второй метод используется для группирования выбора упорядоченности.

Класс DataGridView отображает время работы сортировок в табличном виде. Имеет один атрибут и метод, который вывод таблицу в программу. Данный класс является наследником от класса DataGrigViewTextBoxColumb, у которого имеется 4 атрибута – колонки в таблице и один метод, который позволяет отобразить колонки в таблице.

Класс NumericUpDown используется для того, чтобы пользователь мог задавать значения. Класс имеет два атрибута и два метода, позволяющие задавать пользователю размерность двусвязного списка и процент частичной упорядоченности.

# Реализация программы

## Кодирование

В ходе выполнения курсовой работы было разработано приложение «Сравнение и анализ алгоритмов сортировки двухсвязных списков. Эффективные алгоритмы сортировки», которое позволяет проанализировать время работы алгоритмов сортировки, а также наглядно отобразить время в графическом и табличном видах. Код программы приведён в **приложении А**.

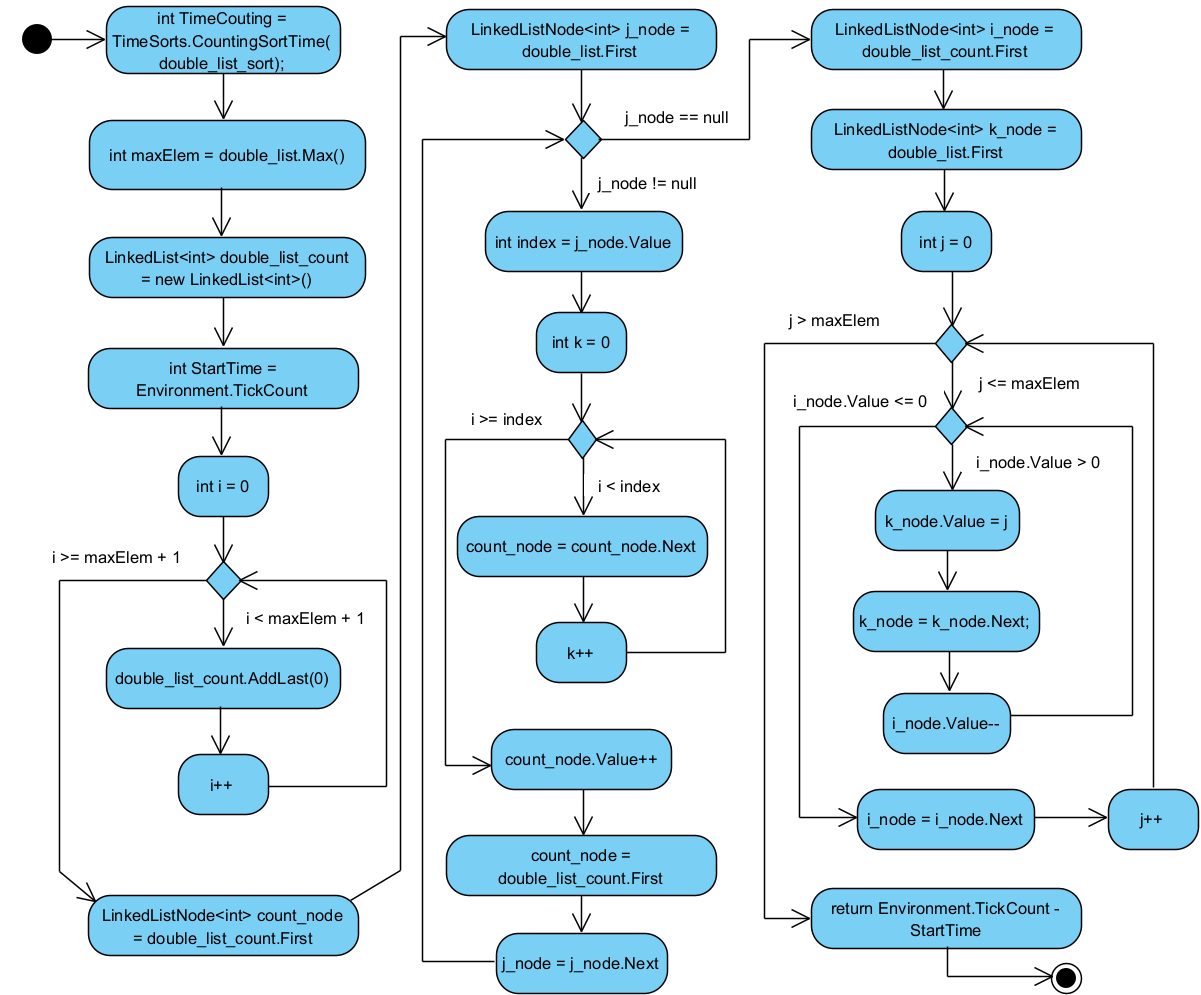
Для алгоритма сортировки подсчетом была разработана диаграмма деятельности (рисунок 5)[10].

Рисунок 5 – Диаграмма деятельности «Алгоритм сортировки подсчетом»

На диаграмме используются следующие переменные:

* TimeCouting – получает время работы сортировки, с помощью функции, класса TimeSorts, CountingSortTime, в которую передается двусвязный список.
* maxElem – максимальный элемент в двусвязном списке.
* double\_list\_count – дополнительный список, в который будет записываться количество вхождений каждого элемента двусвязного списка.
* StartTime – зафиксирует время начала работы сортировки.
* i – переменная-счетчик, для первого цикла.
* count\_node – узел дополнительного списка для второго цикла.
* j\_node – узел сортируемого списка для второго цикла.
* index – переменная, в которую записывается значения узла j\_node.
* k – переменная-счетчик, для второго цикла.
* i\_node - узел дополнительного списка для третьего цикла.
* k\_node - узел сортируемого списка для третьего цикла.
* j – переменная-счетчик для третьего цикла.

## Диаграмма компонентов

В процессе выполнения курсовой работы была составлена диаграмма компонентов, которая отображает разбиение программной системы на структурные компоненты и связи между компонентами (рисунок 6).

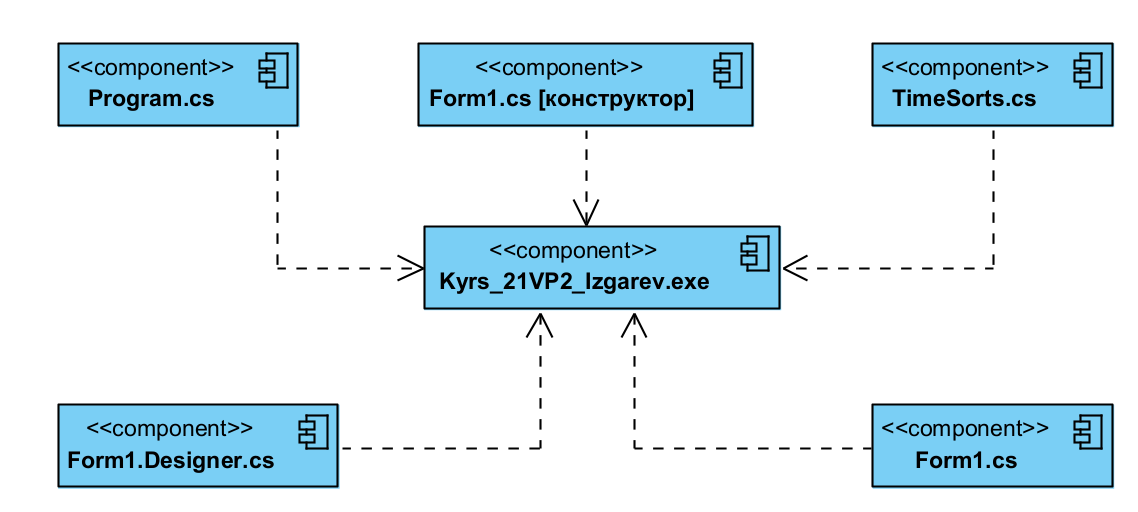
****

Рисунок 6 – Диаграмма компонентов

Kyrs\_21VP2\_Izgarev.exe – исполняемый файл

TimeSorts.cs – файл, который содержит алгоритмы трёх сортировок, а именно, сортировка Шелла, быстрая сортировка, сортировка подсчетом.

Form1.cs – файл класса окон.

Form1.cs [Конструктор] – файл конструктора, в котором происходит создание интерфейса.

Form1.Designer.cs – файл конструктора, в котором элементы формы инициализируются.

Program.cs – файл основного приложения.

# Тестирование

## Виды тестирования программных средств

Существуют разные виды тестирования, которые помогают выявлять ошибки в программном обеспечении. Ниже описаны наиболее известные методы тестирования:

* Функциональное тестирование

Функциональное тестирование является одним из ключевых видов тестирования, задача которого – установить соответствие разработанного программного обеспечения (ПО) исходным функциональным требованиям компании клиента. То есть проведение функционального тестирования позволяет проверить способность информационной системы в определенных условиях решать задачи, нужные пользователям.

В зависимости от степени доступа к коду системы можно выделить два типа функциональных испытаний:

1. Тестирование black box (черный ящик) – проведение функционального тестирования без доступа к коду системы,
2. Тестирование white box (белый ящик) – функциональное тестирование с доступом к коду системы.

Основные преимущества:

1. Функциональное тестирование ПО полностью имитирует фактическое использование системы.
2. Позволяет своевременно выявить системные ошибки ПО и, тем самым, избежать множества проблем при работе с ним в дальнейшем.
3. Экономия за счет исправления ошибок на более раннем этапе жизненного цикла ПО.

* Системное тестирование

Системное тестирование предназначено для тестирования готового ПО в том состоянии, в котором оно будет внедряться в опытно-промышленную эксплуатацию.

Системное тестирование позволяет обнаружить такие дефекты как выявление отсутствующего функционала в системе, некорректная работа функций системы, возникновение ошибок при использовании специфических тестовых данных или их комбинации, ошибки взаимодействия с другими системами. Основные преимущества:

1. Сокращение количества дефектов в опытно-промышленной эксплуатации;
2. Возможность использования тестовых сценариев в качестве обучающих материалов для будущих пользователей системы;
3. Выявление ошибок настройки стенда, что облегчает работу администраторов АС при установке системы в опытно-промышленную эксплуатацию.

* Тестирование производительности

Тестирование производительности - тестирование, которое проводится с целью определения, как быстро работает вычислительная система или её часть под определённой нагрузкой. Также может служить для проверки и подтверждения других атрибутов качества системы, таких как масштабируемость, надёжность и потребление ресурсов.

* Регрессионное тестирование

Регрессионное тестирование позволяет проверить корректность дополнений и удостовериться в том, что программа после изменений продолжает соответствовать установленным требованиям и успешно взаимодействует с другими системами.

Основные преимущества:

1. При регулярном проведении регрессионного тестирования — значительное сокращение количества дефектов в системе к моменту релиза.
2. Исключение деградации качества системы при росте функциональности.
3. Уменьшение вероятности критических ошибок в опытно-промышленной эксплуатации.

* Модульное тестирование

Модульное тестирование - тестирование, которое позволяет проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки.

Основные преимущества:

1. Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно.
2. Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом.
3. Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера.

* Тестирование безопасности

Тестирование безопасности проводится с целью оценки устойчивости системы к противоправным действиям: хакерским атакам, проникновению вирусам, попыткам доступа к корпоративной информации.

Основные преимущества:

1. Тестирование безопасности снижает вероятность несанкционированного доступа к системе, краж информации и потерь данных;
2. Клиенты получают объективную оценку уровня защищенности систем.

При анализе вышеперечисленных видов тестирования, было решено выбрать для курсовой работы такой вид, как функциональное тестирование. Полная имитация использования, а также краткость жизненного цикла разработанного приложения, позволяют избрать именно функциональное тестирование[11].

## Функциональное тестирование программы

В курсовой работе было выполнено функциональное тестирование разработанного программного обеспечения. Результаты тестирования приведены в таблице 4.

Таблица 4 – Результаты тестирования.

|  |  |  |
| --- | --- | --- |
| **Состав теста** | **Ожидаемый результат** | **Наблюдаемый результат** |
| Проверка на работу приложения без указания размерности двусвязного списка | Программа выведет ошибку в виде сообщения на экране пользователя | На экране вывелась ошибка: «Вы не ввели размерность двусвязного списка!» (Рисунок 7) |
| Проверка на работу приложения без выбора типа упорядоченности двусвязного списка | Программа выведет ошибку в виде сообщения на экране пользователя | На экране вывелась ошибка: «Выберите один из четырёх типов упорядоченности!» (Рисунок 8) |

Продолжение таблицы 4

|  |  |  |
| --- | --- | --- |
| Сравнение сортировок для неупорядоченного двусвязного списка (n=9000) | Должна быть выполнена сортировка тремя способами, выведены результаты в табличном и графическом виде | Сортировка выполнена, результаты получены и выведены. Тест пройден (Рисунок 9) |
| Сравнение сортировок для упорядоченного двусвязного списка (n=13000) | Должна быть выполнена сортировка тремя способами, выведены результаты в табличном и графическом виде | Сортировка выполнена, результаты получены и выведены. Тест пройден (Рисунок 10) |
| Сравнение сортировок для упорядоченного двусвязного списка (n=7000) | Должна быть выполнена сортировка тремя способами, выведены результаты в табличном и графическом виде | Сортировка выполнена, результаты получены и выведены. Тест пройден (Рисунок 11) |
| Сравнение сортировок для частично упорядоченного двусвязного списка (n=17000, 40% упорядоченности) | Должна быть выполнена сортировка тремя способами, выведены результаты в табличном и графическом виде | Сортировка выполнена, результаты получены и выведены. Тест пройден (Рисунок 12) |
| Сравнение сортировок для частично упорядоченного двусвязного списка (n=14000, 70% упорядоченности) | Должна быть выполнена сортировка тремя способами, выведены результаты в табличном и графическом виде | Сортировка выполнена, результаты получены и выведены. Тест пройден (Рисунок 13) |

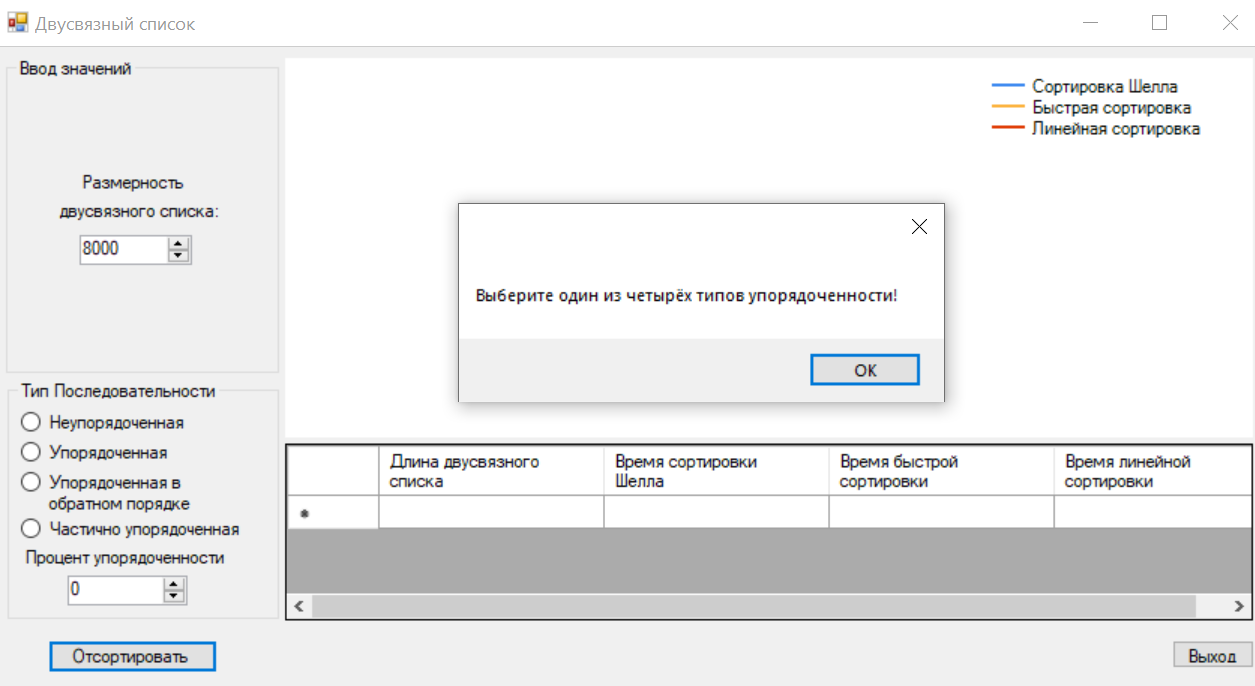
На рисунках 7-13 приведены скриншоты, которые показывают результаты работы программы в процессе тестирования. 

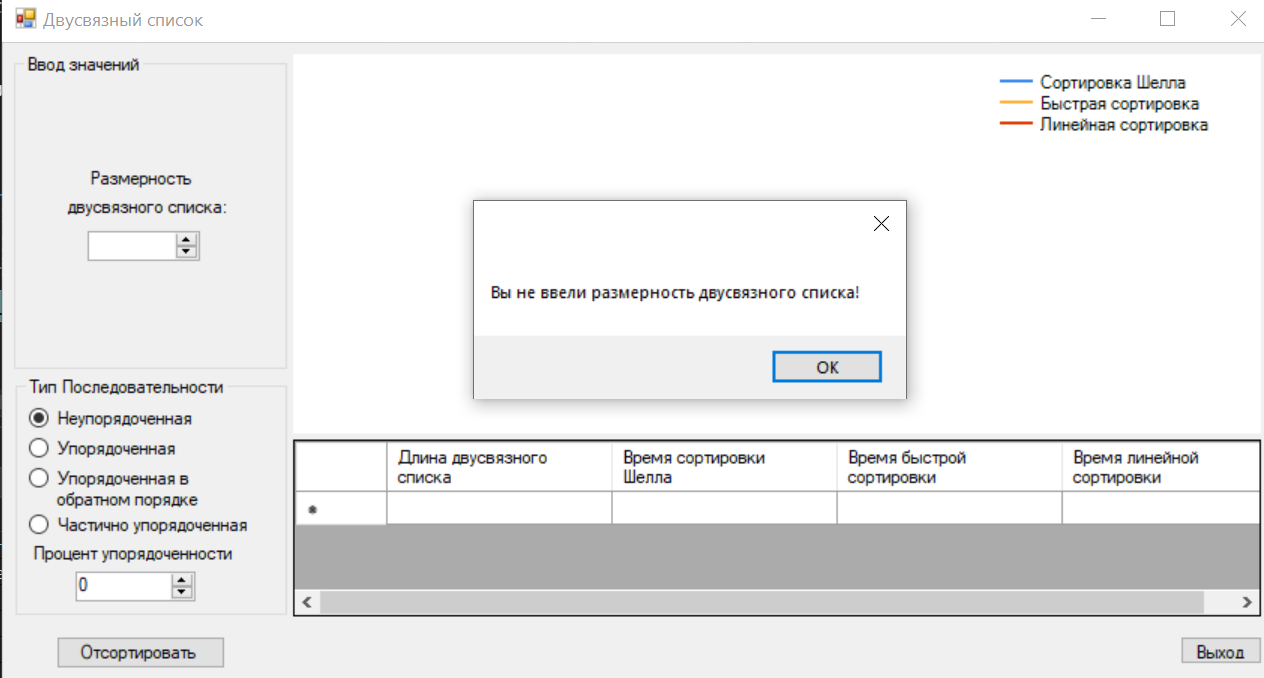
Рисунок 7 – Проверка на работы приложения без указания размерности двусвязного списка

Рисунок 8 – Проверка на работы приложения без выбора типа упорядоченности двусвязного списка

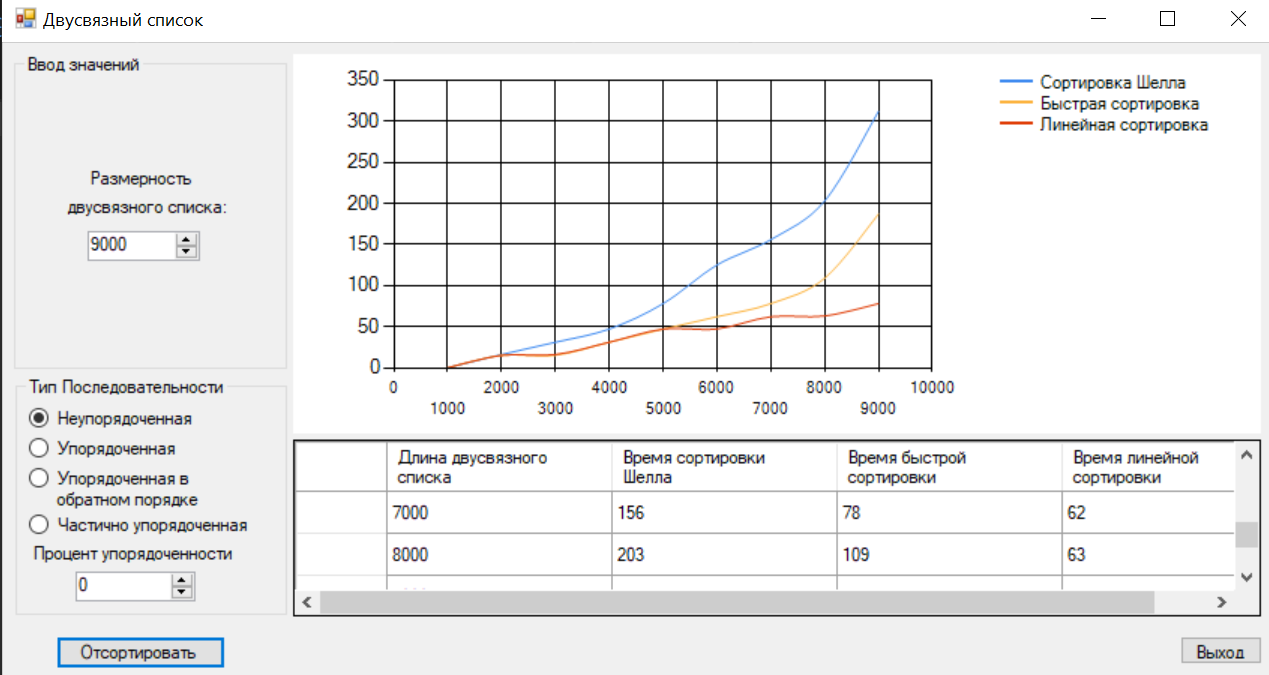


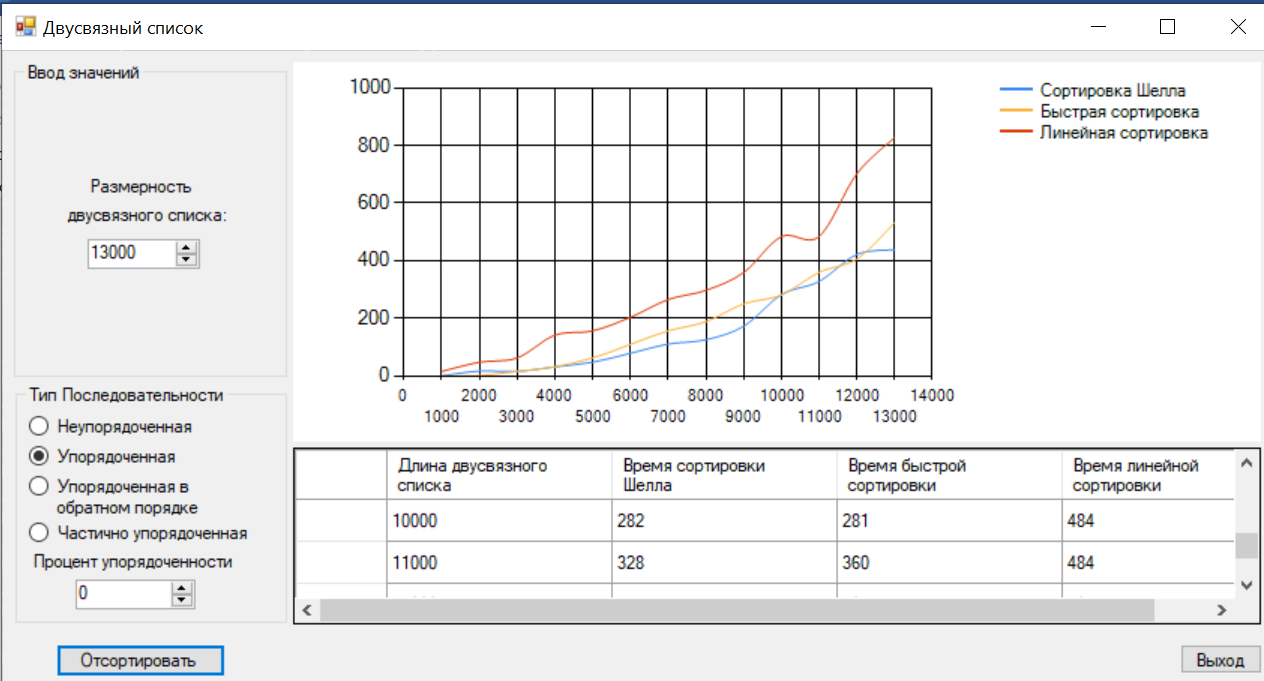
Рисунок 9 – Тестирование программы для неупорядоченного двусвязного списка

Рисунок 10 – Тестирование программы для упорядоченного двусвязного списка

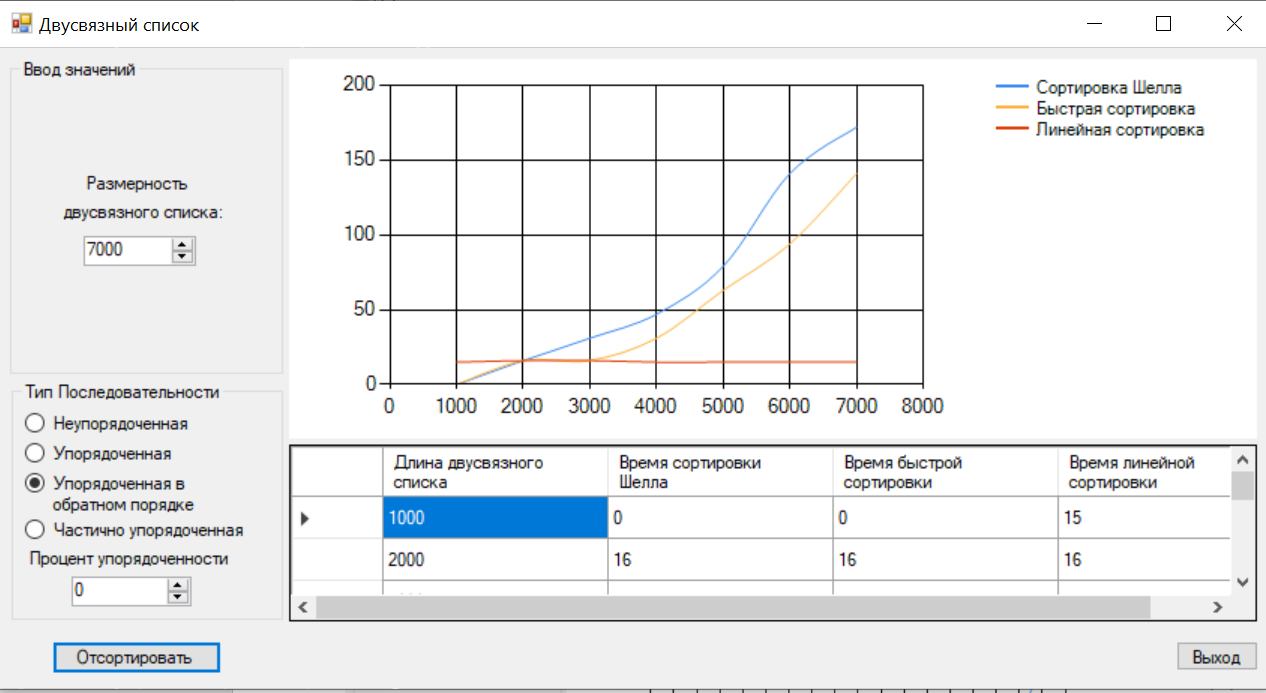


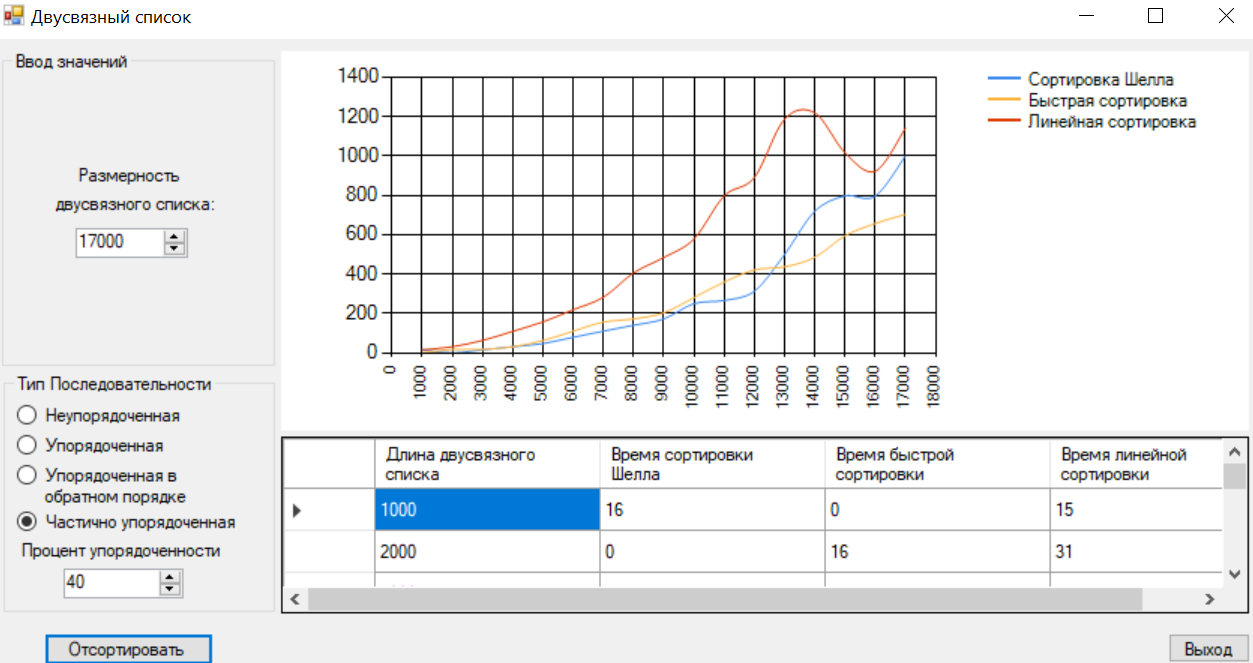
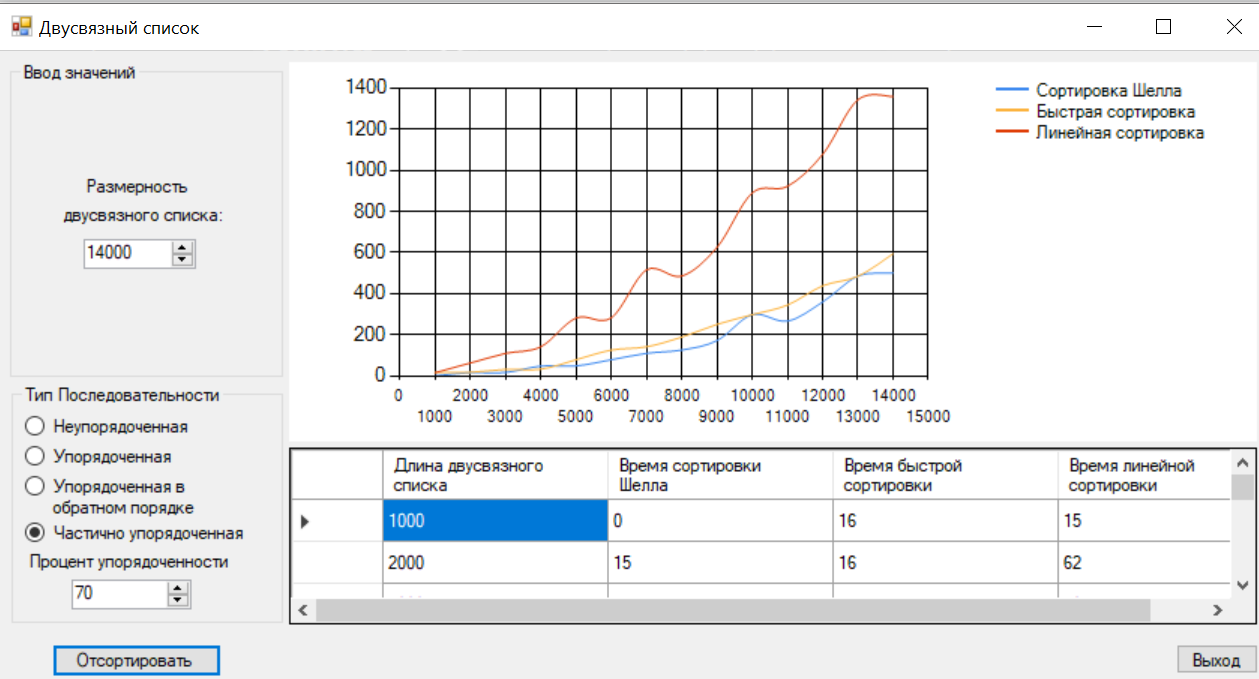
Рисунок 11 – Тестирование программы для упорядоченного в обратном порядке двусвязного списка 

Рисунок 12 – Тестирование программы для частично упорядоченного(40%) двусвязного списка

Рисунок 13 – Тестирование программы для частично упорядоченного(70%) двусвязного списка

В ходе выполнения тестирования несовпадения ожидаемого и наблюдаемого результата не выявлены. Следовательно, можно сделать вывод, что программа работает корректно.

# Анализ результатов

В ходе выполнения курсовой работы были проанализированы такие сортировки, как сортировка Шелла, быстрая сортировка и линейная сортировка.

В приведенном графике (рисунок 14) указано количество сортируемых элементов неупорядоченного двусвязного списка, за определенное время, для каждой из трех сортировок, где размерность равна 15000. На графике мы видим, что самая эффективная сортировка – линейная сортировка, самая худшая сортировка – сортировка Шелла.

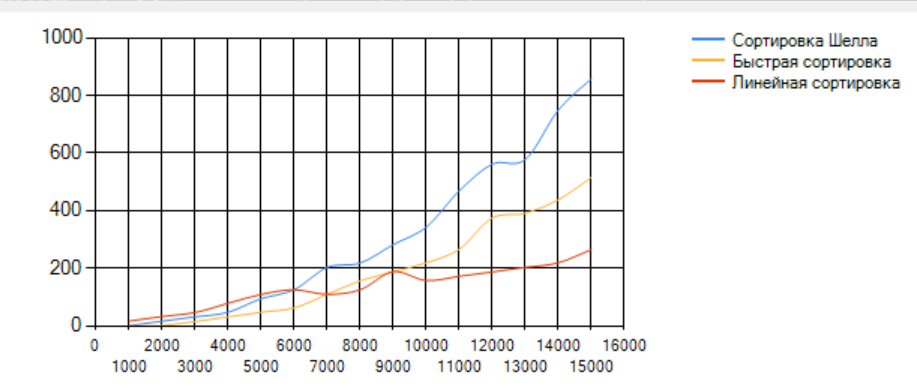


Рисунок 14– График неупорядоченного двусвязного списка.

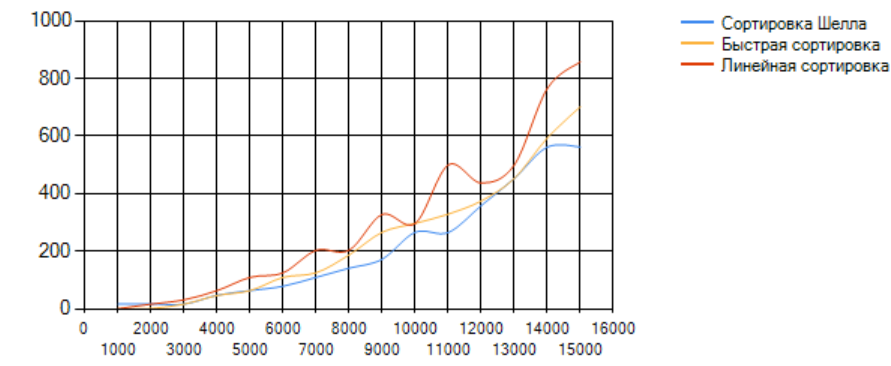
Далее рассмотри следующий график (рисунок 15) упорядоченного двусвязного списка, размерностью 15000. На данном графике мы видим противоположную ситуацию предыдущему графику. Самая лучшая сортировка – сортировка Шелла, худшая сортировка – линейная сортировка.

Рисунок 15 – График упорядоченного двусвязного списка

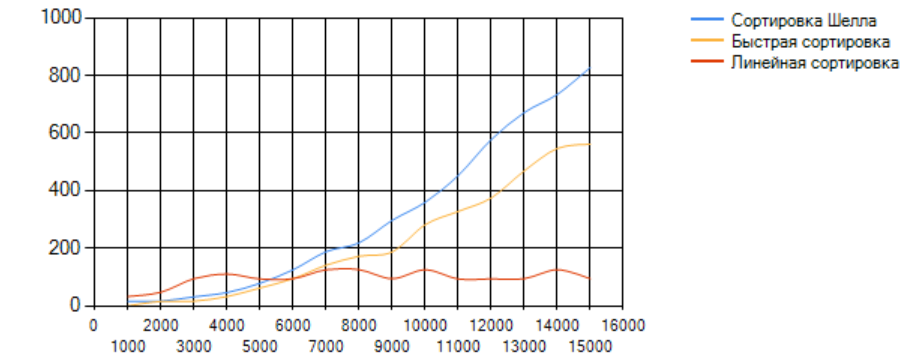
Следующий график (рисунок 16) сформирован из упорядоченного в обратном порядке двусвязного списка, с размерность 15000. На этом графике мы можем отметить то, что лучше всего в этом типе последовательности справляется линейная сортировка. Самый худший результат – сортировка Шелла. 

Рисунок 16 – График упорядоченного в обратном порядке двусвязного списка

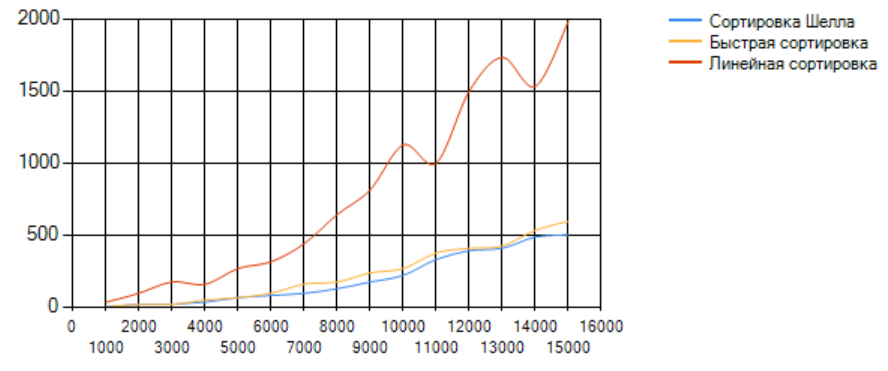
На последнем графике (рисунок 17), который был сформирован из частично упорядоченного двусвязного списка(процент упорядоченности равен 50%), мы видим, что самый выдающийся результат у сортировки Шелла, почти такой же результат у быстрой сортировки. Худшее время – линейная сортировка. 

Рисунок 17 – График частично упорядоченного двусвязного списка(50%)

Таким образом, анализируя данные результаты различных видов последовательности двусвязного списка, можно сделать вывод о том, что нельзя выбрать одну их трех сортировок в двусвязном списке, которая будет работать всегда быстрее других, независимо от типа последовательности. Сортировка Шелла отлично себя показывает в упорядоченном и в частично упорядоченном типе последовательности. Линейная сортировка работает быстрее всех в неупорядоченном и в упорядоченном в обратном порядке двусвязном списке. Самой стабильной сортировкой, которая показывала на протяжении всех четырех графиков средний результат, является быстрая сортировка.

# Заключение

В данной курсовой работе была поставлена такая задача, как проанализировать время работы алгоритмов трех сортировок, а именно сортировка Шелла, быстрая сортировка, линейная сортировка. Далее было необходимо сравнить результаты сортировок в неупорядоченной, упорядоченной, упорядоченной в обратном порядке, частично упорядоченной типах последовательности двусвязного списка.

Для реализации поставленной задачи были выполнены следующие шаги:

* Проанализированы требования
* Спроектированы структура данных и модель интерфейса
* Реализована программа с учетом поставленных требований
* Проведено функциональное тестирование
* Проанализированы результаты работ сортировок.

С помощью сравнения времени работ сортировок, были сделаны выводы о том, что в каких ситуациях и какую сортировку следует использовать для достижения быстрого результата в такой структуре данных, как двусвязной список.

# Список использованных источников

1. Википедия – <https://ru.wikipedia.org/wiki/Алгоритм_сортировки>
2. И.А. Казакова, С.В. Самуйлов «Структуры данных. Учебное пособие» Пенза, Издательство ПГУ, 2011 г.
3. Википедия – [https://ru.wikipedia.org/wiki/Сортировка\_Шелла](http://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%A8%D0%B5%D0%BB%D0%BB%D0%B0)
4. <https://vscode.ru/prog-lessons/sortirovka-shella.html>
5. Википедия –[https://ru.wikipedia.org/wiki/Быстрая\_сортировка](https://ru.wikipedia.org/wiki/%D0%91%D1%8B%D1%81%D1%82%D1%80%D0%B0%D1%8F_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0)
6. Википедия – [https://ru.wikipedia.org/wiki/Сортировка\_подсчетом](https://ru.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%BA%D0%B0_%D0%BF%D0%BE%D0%B4%D1%81%D1%87%D1%91%D1%82%D0%BE%D0%BC)
7. <https://thecode.media/counting-sort/>
8. Википедия – <https://ru.wikipedia.org/wiki/Структура_данных>
9. <https://ru.wikibrief.org/wiki/Doubly_linked_list>
10. Джим Арлоу. UML2 и Унифицированный процесс. Практический объектно-ориентированный анализ и проектирование/Джим Арлоу, Айла Нейштадт. – Санкт-Петербург, Издательство Символ-Плюс, 2007. – 624
11. <https://daglab.ru/funkcionalnoe-testirovanie-programmnogo-obespechenija/>

# Приложение А.

**КОД ПРОГРАММЫ**

**(обязательное)**

**Файл Program.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace Kyrs\_21VP2\_Izgarev

{

internal static class Program

{

/// <summary>

/// Главная точка входа для приложения.

/// </summary>

[STAThread]

static void Main()

{

Application.EnableVisualStyles();

Application.SetCompatibleTextRenderingDefault(false);

Application.Run(new Form1());

}

}

}

**Файл TimeSorts.cs**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Runtime.CompilerServices;

using System.Text;

using System.Threading.Tasks;

namespace Kyrs\_21VP2\_Izgarev

{

public class TimeSorts

{

public static int ShellSortingTime(LinkedList<int> double\_list)

{

int StartTime = Environment.TickCount;

LinkedListNode<int> j\_node;

int step = double\_list.Count / 2;

while (step > 0)

{

for (LinkedListNode<int> i\_node = double\_list.First; i\_node != null; i\_node = i\_node.Next)

{

j\_node = i\_node;

LinkedListNode<int> j\_step = j\_node;

for (int k = 0; k < step && j\_step != null; k++)

{

j\_step = j\_step.Next;

}

while ((j\_node != null && j\_step != null) && (j\_node.Value > j\_step.Value))

{

int tmp = j\_node.Value;

j\_node.Value = j\_step.Value;

j\_step.Value = tmp;

for (int k = 0; k < step && j\_node != null; k++)

{

j\_step = j\_step.Previous;

j\_node = j\_node.Previous;

}

}

}

step = step / 2;

}

return Environment.TickCount - StartTime;

}

public static void QuickSort(LinkedList<int> double\_list, int leftIndex, int rightIndex)

{

int i = leftIndex;

int j = rightIndex;

LinkedListNode<int> i\_node = double\_list.First;

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

{

i\_node = i\_node.Next;

}

LinkedListNode<int> j\_node = double\_list.Last;

for (int k = double\_list.Count - 1; k > j; k--)

{

j\_node = j\_node.Previous;

}

int pivot = i\_node.Value;

while (i <= j)

{

while (i\_node.Value < pivot)

{

i++;

i\_node = i\_node.Next;

}

while (j\_node.Value > pivot)

{

j--;

j\_node = j\_node.Previous;

}

if (i <= j)

{

int temp = i\_node.Value;

i\_node.Value = j\_node.Value;

j\_node.Value = temp;

i++;

j--;

i\_node = i\_node.Next;

j\_node = j\_node.Previous;

}

}

if (leftIndex < j)

QuickSort(double\_list, leftIndex, j);

if (i < rightIndex)

QuickSort(double\_list, i, rightIndex);

}

/\* Из-за рекурсивности данного алгоритма невозможно в основной функции сортировки

сделать подсчет времени. Для этого было решено сделать дополнительный метод,

в котором вычисляется стартовое время \*/

public static int QuickSortTime(LinkedList<int> double\_list, int low, int high)

{

int StartTime = Environment.TickCount;

QuickSort(double\_list, low, high);

return Environment.TickCount - StartTime;

}

public static int CountingSortTime(LinkedList<int> double\_list)

{

int maxElem = double\_list.Max();

LinkedList<int> double\_list\_count = new LinkedList<int>();

int StartTime = Environment.TickCount;

for (int i = 0; i < maxElem + 1; i++)

{

double\_list\_count.AddLast(0);

}

LinkedListNode<int> count\_node = double\_list\_count.First;

for (LinkedListNode<int> j\_node = double\_list.First; j\_node != null; j\_node = j\_node.Next)

{

int index = j\_node.Value;

for (int k = 0; k < index; k++)

{

count\_node = count\_node.Next;

}

count\_node.Value++;

count\_node = double\_list\_count.First;

}

LinkedListNode<int> i\_node = double\_list\_count.First;

LinkedListNode<int> k\_node = double\_list.First;

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

{

while (i\_node.Value > 0)

{

k\_node.Value = j;

k\_node = k\_node.Next;

i\_node.Value--;

}

i\_node = i\_node.Next;

}

return Environment.TickCount - StartTime;

}

}

}

**Файл Form1.cs**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Diagnostics;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Windows.Forms.DataVisualization.Charting;

using static System.Windows.Forms.VisualStyles.VisualStyleElement.Button;

namespace Kyrs\_21VP2\_Izgarev

{

public partial class Form1 : Form

{

const int N = 30000;

LinkedList<int> double\_list = new LinkedList<int>();

LinkedList<int> double\_list\_sort = new LinkedList<int>();

Random rnd = new Random();

int user\_size, percent;

public Form1()

{

InitializeComponent();

}

public void chart1\_Click(object sender, EventArgs e)

{

}

private void label1\_Click(object sender, EventArgs e)

{

}

private void radioButton1\_CheckedChanged(object sender, EventArgs e)

{

}

private void panel1\_Paint(object sender, PaintEventArgs e)

{

}

private void groupBox1\_Enter(object sender, EventArgs e)

{

}

private void label3\_Click(object sender, EventArgs e)

{

}

private void Form1\_Load(object sender, EventArgs e)

{

}

private void groupBox2\_Enter(object sender, EventArgs e)

{

}

private void radioButton6\_CheckedChanged(object sender, EventArgs e)

{

check\_user\_size(user\_size);

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

{

double\_list.AddLast(rnd.Next(1, user\_size));

}

}

private void radioButton5\_CheckedChanged(object sender, EventArgs e)

{

check\_user\_size(user\_size);

double\_list.AddLast(rnd.Next(user\_size));

for (int i = 1; i < N; i++)

{

double\_list.AddLast(double\_list.ElementAt(i - 1) + rnd.Next(1, 5));

}

}

private void button1\_Click(object sender, EventArgs e)

{

if(!radioButton1.Checked && !radioButton5.Checked && !radioButton6.Checked && !radioButton4.Checked)

{

MessageBox.Show("Выберите один из четырёх типов упорядоченности!");

Application.Restart();

}

ClearGraphic();

CleatTable();

const int step = 1000;

int step\_grapchic = user\_size / 1000;

int step\_elem = step;

if (user\_size % 1000 != 0)

{

step\_elem = (user\_size % 1000) + step;

}

for (int i = 0; i < step\_grapchic; i++)

{

update\_list(double\_list, double\_list\_sort, step\_elem);

int TimeShell = TimeSorts.ShellSortingTime(double\_list\_sort);

update\_list(double\_list, double\_list\_sort, step\_elem);

int TimeQuick = TimeSorts.QuickSortTime(double\_list\_sort, 0, double\_list\_sort.Count - 1);

update\_list(double\_list, double\_list\_sort, step\_elem);

int TimeCouting = TimeSorts.CountingSortTime(double\_list\_sort);

Graphic(TimeShell, TimeQuick, TimeCouting, step\_elem);

Table(TimeShell, TimeQuick, TimeCouting, step\_elem);

step\_elem += step;

}

}

private void update\_list(LinkedList<int> double\_list, LinkedList<int> double\_list\_sort, int new\_size)

{

double\_list\_sort.Clear();

for (int i = 0; i < new\_size; i++)

{

double\_list\_sort.AddLast(double\_list.ElementAt(i));

}

}

private void Graphic(int first\_sort, int second\_sort, int third\_sort, int number)

{

this.chart1.Series[0].Points.AddXY(number, first\_sort);

this.chart1.Series[1].Points.AddXY(number, second\_sort);

this.chart1.Series[2].Points.AddXY(number, third\_sort);

}

private void ClearGraphic()

{

for(int i = 0; i < 3; i++)

{

this.chart1.Series[i].Points.Clear();

}

this.chart1.Update();

}

private void Table(int first\_sort, int second\_sort, int third\_sort, int number)

{

dataGridView1.Rows.Add(number, first\_sort, second\_sort, third\_sort);

}

private void CleatTable()

{

dataGridView1.Rows.Clear();

}

private void numericUpDown1\_ValueChanged(object sender, EventArgs e)

{

user\_size = (int)numericUpDown1.Value;

}

private void label2\_Click(object sender, EventArgs e)

{

}

private void radioButton4\_CheckedChanged(object sender, EventArgs e)

{

check\_user\_size(user\_size);

double\_list.AddLast(user\_size);

for (int i = 1; i < N; i++)

{

double\_list.AddLast(double\_list.ElementAt(i - 1) - rnd.Next(1, 5));

}

}

private void radioButton1\_CheckedChanged\_1(object sender, EventArgs e)

{

check\_user\_size(user\_size);

percent = (int)numericUpDown2.Value;

int count\_yp = (N \* percent) / 100;

double\_list.AddLast(rnd.Next(0, user\_size));

for (int i = 1; i < count\_yp; i++)

{

double\_list.AddLast(double\_list.ElementAt(i - 1) + rnd.Next(1, 10));

}

for (int i = count\_yp; i < N; i++)

{

double\_list.AddLast(rnd.Next(0, user\_size));

}

}

public void check\_user\_size(int size)

{

if (size == 0)

{

MessageBox.Show("Вы не ввели размерность двусвязного списка!");

Application.Restart();

}

}

private void numericUpDown2\_ValueChanged(object sender, EventArgs e)

{

}

private void dataGridView1\_CellContentClick(object sender, DataGridViewCellEventArgs e)

{

}

private void groupBox3\_Enter(object sender, EventArgs e)

{

}

}

}