

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ  
Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

*Факультет*  
вычислительной техники

*Кафедра*  
МОиПЭВМ

Направление подготовки 09.03.04 «Программная инженерия»

БАКАЛАВРСКАЯ РАБОТА  
на тему  
**ПРОГРАММА ОЦЕНКИ ВРЕМЕННЫХ**  
**ХАРАКТЕРИСТИК АЛГОРИТМОВ**  
**СОРТИРОВКИ**

|                |                 |                                |
|----------------|-----------------|--------------------------------|
| Студент        | _____           | <u>Фрунзе Павел Михайлович</u> |
|                | (подпись, дата) | (ФИО полностью)                |
| Руководитель   | _____           | <u>Макарычев П.П.</u>          |
|                | (подпись, дата) | (фамилия, инициалы)            |
| Нормоконтролёр | _____           | <u>Такташкин Д.В.</u>          |
|                | (подпись, дата) | (фамилия, инициалы)            |

Работа допущена к защите (протокол заседания кафедры от \_\_\_\_\_ № \_\_\_\_\_)

|                     |           |                       |
|---------------------|-----------|-----------------------|
| Заведующий кафедрой | _____     | <u>Макарычев П.П.</u> |
|                     | (подпись) | (фамилия, инициалы)   |

Работа защищена с отметкой \_\_\_\_\_ (протокол заседания ГЭК от \_\_\_\_\_ № \_\_\_\_\_)

|               |           |                     |
|---------------|-----------|---------------------|
| Секретарь ГЭК | _____     | <u>Попова Н.А.</u>  |
|               | (подпись) | (фамилия, инициалы) |

Пенза, 2020

Федеральное государственное бюджетное образовательное учреждение  
высшего образования  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

*Факультет*

Вычислительной техники

*Кафедра*

МО и ПЭВМ

«Утверждаю»  
Заведующий кафедрой  
П.П. Макарычев  
«\_\_» \_\_\_\_\_ 2020 г

**ЗАДАНИЕ  
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ  
БАКАЛАВРА**

1. Студент Фрунзе Павел Михайлович гр. 16ВП1 факультета ФВТ  
<Фамилия, имя, отчество полностью>

2. Тема работы Программа оценки временных характеристик алгоритмов сортировки

Тема утверждена приказом ПГУ № \_\_\_\_\_ от "\_\_\_\_\_" \_\_\_\_\_ 20\_\_\_\_

2. Руководитель работы Макарычев П.П.

3. Задание на работу (назначение разработки, исходные данные и т.п.) Необходимо  
создать программу оценки временных характеристик алгоритмов сортировки массивов.  
Программа должна поддерживать несколько алгоритмов сортировки и типов генерации  
массивов. Оценку временных характеристик осуществлять по методике под названием  
«тест производительности». Измерения должны выполняться с высокой точностью, то  
есть необходимо внедрить систему снижения погрешности. Для ускорения получения  
результатов тестирования внедрить режим многопоточности. Результаты выдавать в виде  
графиков. Внедрить возможность пользовательского расширения программы с помощью  
системы плагинов, которые смогут содержать новые алгоритмы сортировки и генерации.  
Приложение должно быть рассчитано на работу в операционных системах Windows 7 и  
выше.

Созданную программу протестировать.

После успешного тестирования программы применить ее для сравнительного анализа  
временных характеристик двух произвольных алгоритмов сортировки. Полученные  
данные описать и проанализировать.

4. Перечень подлежащих разработке вопросов В рамках работы нужно выполнить  
следующие пункты:

|                                  |
|----------------------------------|
| 1. Планирование разработки       |
| 2. Анализ требований к программе |
| 3. Проектирование программы      |
| 4. Реализация программы          |
| 5. Контроль качества программы   |
| 6. Применение программы          |

5. Календарный график выполнения работы

| №<br>п/п | Наименование этапов работы       | Объем<br>работы | Срок<br>выполнения         | Подпись<br>руководителя |
|----------|----------------------------------|-----------------|----------------------------|-------------------------|
| 1        | Планирование разработки          | 7%              | 7.04.2020 –<br>10.04.2020  |                         |
| 2        | Анализ требований к программе    | 19%             | 13.04.2020 –<br>17.04.2020 |                         |
| 3        | Проектирование программы         | 39%             | 20.04.2020 –<br>28.04.2020 |                         |
| 4        | Реализация программы             | 39%             | 29.04.2020 –<br>11.05.2020 |                         |
| 5        | Контроль качества программы      | 79%             | 12.05.2020 –<br>21.05.2020 |                         |
| 6        | Применение программы             | 86%             | 22.05.2020 –<br>25.05.2020 |                         |
| 7        | Оформление пояснительной записки | 100%            | 26.05.2020 –<br>02.06.2020 |                         |

Дата выдачи задания " 25 " декабря 2019 г.

Руководитель бакалаврской работы \_\_\_\_\_  
(подпись, дата)

Задание к исполнению принял студент \_\_\_\_\_  
(подпись, дата)

Работу к защите допустить  
Декан факультета \_\_\_\_\_

## Реферат

Пояснительная записка содержит 118 листов, 26 рисунков, 15 таблиц, 3 листинга, 32 использованных источника, 2 приложения.

АЛГОРИТМЫ СОРТИРОВКИ, ТЕСТ ПРОИЗВОДИТЕЛЬНОСТИ, ОЦЕНКА ВРЕМЕННЫХ ХАРАКТЕРИСТИК, ПРОГРАММА ОЦЕНКИ ВРЕМЕННЫХ ХАРАКТЕРИСТИК АЛГОРИТМОВ СОРТИРОВКИ, СРАВНИТЕЛЬНЫЙ АНАЛИЗ АЛГОРИТМОВ СОРТИРОВКИ

Объект исследования: программа оценки временных характеристик алгоритмов сортировки.

Цель работы: создание программы оценки временных характеристик алгоритмов сортировки.

Технологии разработки: RUP, C#, ООП.

Результатом работы является программный пакет, состоящий из исполняемого файла, библиотеки-программного интерфейса приложения для расширения с помощью плагинов и семи стандартных плагинов. Анализ продуктов-аналогов выявил, что созданное приложение является наилучшим среди исследованных. Приложение было применено для сравнительного анализа временных характеристик сортировки Шелла и быстрой сортировки.

Созданное приложение может быть применено для исследования алгоритмов сортировки, в том числе в рамках обучения студентов-программистов.

|           |      |                |         |      |                                                                                               |              |      |        |
|-----------|------|----------------|---------|------|-----------------------------------------------------------------------------------------------|--------------|------|--------|
|           |      |                |         |      | ПГУ 09.03.04–8ВР161.26 ПЗ                                                                     |              |      |        |
| Изм.      | Лист | № докум.       | Подпись | Дата | «Программа оценки временных характеристик алгоритмов сортировки»<br><br>Пояснительная записка | Лит.         | Лист | Листов |
| Разраб.   |      | Фрунзе П.М.    |         |      |                                                                                               |              |      |        |
| Провер.   |      | Макарычев П.П. |         |      |                                                                                               |              | 4    | 118    |
|           |      |                |         |      |                                                                                               | Группа 16ВП1 |      |        |
| Н. Контр. |      | Такташкин Д.В. |         |      |                                                                                               |              |      |        |
| Утверд.   |      | Макарычев П.П. |         |      |                                                                                               |              |      |        |

## Содержание

|                                                                                                                |    |
|----------------------------------------------------------------------------------------------------------------|----|
| Введение.....                                                                                                  | 7  |
| 1 Постановка задачи на разработку программы оценки временных характеристик алгоритмов сортировки.....          | 8  |
| 2 Планирование разработки программы оценки временных характеристик алгоритмов сортировки .....                 | 9  |
| 3 Анализ предметной области и требований к программе оценки временных характеристик алгоритмов сортировки..... | 13 |
| 3.1 Анализ предметной области .....                                                                            | 13 |
| 3.2 Анализ программ-аналогов оценки временных характеристик алгоритмов сортировки .....                        | 16 |
| 3.2.1 Анализ приложения «Benchmarking Sorting Algorithms» .....                                                | 16 |
| 3.2.2 Анализ приложения «Sorting benchmark» .....                                                              | 17 |
| 3.2.3 Анализ приложения «Анализ алгоритмов внутренней сортировки» .....                                        | 18 |
| 3.2.4 Сравнительный анализ исследованных программ-аналогов.....                                                | 18 |
| 3.2.5 Стратегии повышения качества разрабатываемой программы .....                                             | 20 |
| 3.3 Анализ функциональных требований.....                                                                      | 21 |
| 3.4 Анализ нефункциональных требований.....                                                                    | 24 |
| 4 Проектирование программы оценки временных характеристик алгоритмов сортировки .....                          | 25 |
| 4.1 Проектирование пользовательского интерфейса .....                                                          | 25 |
| 4.2 Проектирование классов программы.....                                                                      | 27 |
| 4.3 Проектирование процесса исследования алгоритмов сортировки.....                                            | 31 |
| 4.4 Проектирование системы плагинов программы.....                                                             | 35 |
| 5 Реализация программы оценки временных характеристик алгоритмов сортировки .....                              | 36 |

|     |                                                                                                                           |     |
|-----|---------------------------------------------------------------------------------------------------------------------------|-----|
| 5.1 | Реализация исполняемой программы .....                                                                                    | 38  |
| 5.2 | Реализация библиотеки для расширения программы .....                                                                      | 39  |
| 5.3 | Реализация стандартных плагинов.....                                                                                      | 41  |
| 6   | Контроль качества разработанной программы.....                                                                            | 44  |
| 6.1 | Модульное тестирование .....                                                                                              | 44  |
| 6.2 | Структурное тестирование.....                                                                                             | 46  |
| 6.3 | Вычисление метрик кода.....                                                                                               | 49  |
| 7   | Применение разработанной программы для исследования временных характеристик алгоритмов сортировки.....                    | 53  |
|     | Заключение .....                                                                                                          | 55  |
|     | Список использованных источников .....                                                                                    | 56  |
|     | Приложение А Исходный код разработанной программы.....                                                                    | 60  |
|     | Приложение Б Скриншоты исследования временных характеристик алгоритмов сортировки при помощи разработанной программы..... | 111 |

## Введение

На данный момент разработано широкое множество алгоритмов сортировки. Однако, когда возникает необходимость выбора конкретного метода сортировки, заранее не известно, какой из алгоритмов в большей мере соответствует условиям поставленной задачи. Выбор алгоритма сортировки напрямую влияет в том числе на быстродействие ПО и трудозатраты, необходимые для написания кода.

Разрабатываемое приложение позволит количественно сравнить временные характеристики методов сортировки для выбора наиболее эффективного алгоритма в конкретных поставленных условиях.

Целью данной работы является разработка программы оценки временных характеристик алгоритмов сортировки.

Были выделены следующие задачи:

- сформулировать требования к программной системе;
- составить план разработки приложения;
- оценить бюджет разработки программной системы.
- проанализировать поставленные требования;
- на основе требований осуществить проектирование программной системы;
- в соответствие с проведенным проектированием произвести реализацию программной системы;
- осуществить тестирование разработанной программной системы;
- с помощью созданной программной системы получить данные о временных характеристиках некоторых алгоритмов сортировок и произвести их сравнительный анализ.

Для моделирования системы главным образом будет использована методология «UML 2.0» [1].

## **1 Постановка задачи на разработку программы оценки временных характеристик алгоритмов сортировки**

Программа оценки временных характеристик алгоритмов сортировки – это программное обеспечение, позволяющее количественно сравнивать оценки времени выполнения тестируемых алгоритмов сортировки [2]. Такое приложение позволяет обнаружить в рамках поставленной задачи наиболее производительный метод сортировки. Разработка программы этого класса является предметом данной работы. Ниже сформулированы основные требования к разрабатываемой программной системе.

Необходимо создать приложение, которое предоставит возможность измерения времени сортировки массивов, сгенерированных по определённым правилам, с помощью определённых алгоритмов сортировки.

Измерения должны производиться с высокой точностью, для этого необходимо внедрить различные методики снижения погрешности.

Результаты измерений должны выводиться в виде графиков.

Должна быть произведена существенная оптимизация программы в первую очередь в плане быстродействия (в том числе с использованием многопоточности), однако не в ущерб точности измерений.

Для повышения удобства необходимо предоставить пользователю возможность выбирать несколько методов сортировки и генерации одновременно.

Должна быть внедрена система плагинов, с помощью которых пользователи-программисты будут иметь возможность добавлять свои собственные алгоритмы сортировки и генерации массивов, не изменяя существующий код.

Приложение должно быть рассчитано на работу в операционных системах «Windows 7» и выше.

## 2 Планирование разработки программы оценки временных характеристик алгоритмов сортировки

С помощью приложения MS Project [3] было произведено планирование разработки программной системы.

Список запланированных задач представлен на рисунке 1.

| Название задачи                              | Длительность | Начало      | Окончание   | Предш.     | Названия ресурсов                                     |
|----------------------------------------------|--------------|-------------|-------------|------------|-------------------------------------------------------|
| Начало проекта                               | 0 дней       | Пн 13.04.20 | Пн 13.04.20 |            |                                                       |
| ▲ Анализ требований                          | 5 дней       | Пн 13.04.20 | Пт 17.04.20 |            |                                                       |
| Анализ предметной области                    | 1 день       | Пн 13.04.20 | Пн 13.04.20 | 1          | Программист №1; Программист №2                        |
| Анализ функциональных требований             | 2 дней       | Вт 14.04.20 | Ср 15.04.20 | 1          | Программист №1; Программист №2                        |
| Анализ нефункциональных требований           | 2 дней       | Чт 16.04.20 | Пт 17.04.20 | 4          | Программист №1; Программист №2                        |
| Окончание и анализа требований               | 0 дней       | Пт 17.04.20 | Пт 17.04.20 | 4;5        |                                                       |
| ▲ Проектирование программной системы         | 7 дней       | Пн 20.04.20 | Вт 28.04.20 |            |                                                       |
| Проектирование классов                       | 3 дней       | Вт 21.04.20 | Чт 23.04.20 | 6          | Программист №1; Программист №2                        |
| Проектирование процесса исследования         | 1 день       | Пн 20.04.20 | Пн 20.04.20 | 6          | Программист №1; Программист №2                        |
| Проектирование системы плагинов              | 3 дней       | Пт 24.04.20 | Вт 28.04.20 | 8;9        | Программист №1; Программист №2                        |
| Окончание проектирования программной системы | 0 дней       | Вт 28.04.20 | Вт 28.04.20 | 8;9;10     |                                                       |
| ▲ Реализация программной системы             | 9 дней       | Ср 29.04.20 | Пн 11.05.20 |            |                                                       |
| Реализация основной исполняемой программы    | 6 дней       | Ср 29.04.20 | Ср 06.05.20 | 11         | Программист №1[80%]; Программист №2[20%]              |
| Реализация основной библиотеки приложения    | 8 дней       | Ср 29.04.20 | Пт 08.05.20 | 11         | Программист №1[20%]; Программист №2[80%]              |
| Реализация стандартных плагинов              | 1 день       | Пн 11.05.20 | Пн 11.05.20 | 14         | Программист №1[50%]; Программист №2[50%]              |
| Окончание реализации программной системы     | 0 дней       | Пн 11.05.20 | Пн 11.05.20 | 13;14;15   |                                                       |
| ▲ Тестирование                               | 11 дней      | Чт 07.05.20 | Чт 21.05.20 |            |                                                       |
| Тестирование основной программы              | 7 дней       | Чт 07.05.20 | Пт 15.05.20 | 13         | Программист №1[50%]; Тестировщик                      |
| Тестирование основной библиотеки приложения  | 7 дней       | Пн 11.05.20 | Ср 20.05.20 | 14         | Программист №2[50%]; Тестировщик                      |
| Тестирование стандартных плагинов            | 7 дней       | Вт 12.05.20 | Чт 21.05.20 | 15         | Программист №2[25%]; Тестировщик; Программист №1[25%] |
| Окончание тестирования                       | 0 дней       | Чт 21.05.20 | Чт 21.05.20 | 20;18;19   |                                                       |
| Конец проекта                                | 0 дней       | Чт 21.05.20 | Чт 21.05.20 | 6;11;16;21 |                                                       |

Рисунок 1 – Список задач проекта

Задачи распределены по четырем пунктам («анализ требований», «проектирование программной системы», «реализация программной системы», «тестирование»), то есть, сгруппированы в виде дерева суммарных задач и подзадач.

План в виде диаграммы Ганта [4] расположен на рисунке 2.

Для разработки планируется привлечь двух программистов (з/п обоих – 200 рублей в час) и одного тестировщика (з/п – 180 рублей в час), которые

Рисунок 3 – График загрузки ресурса «Программист №1» по дням

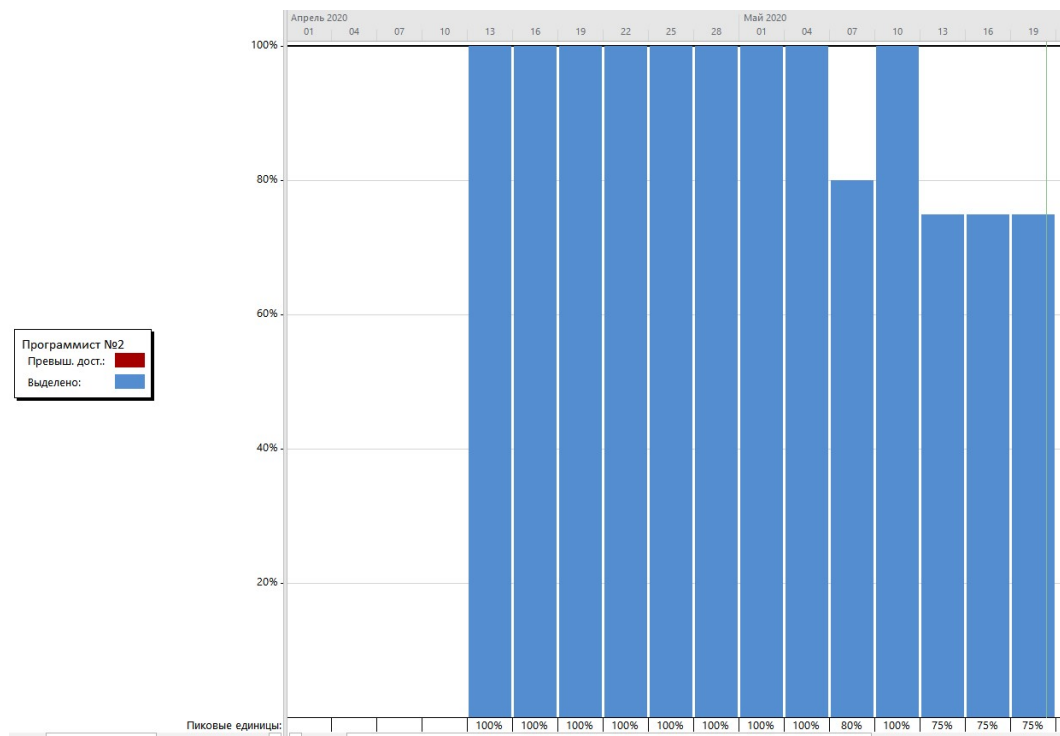


Рисунок 4 – График загрузки ресурса «Программист №2» по дням

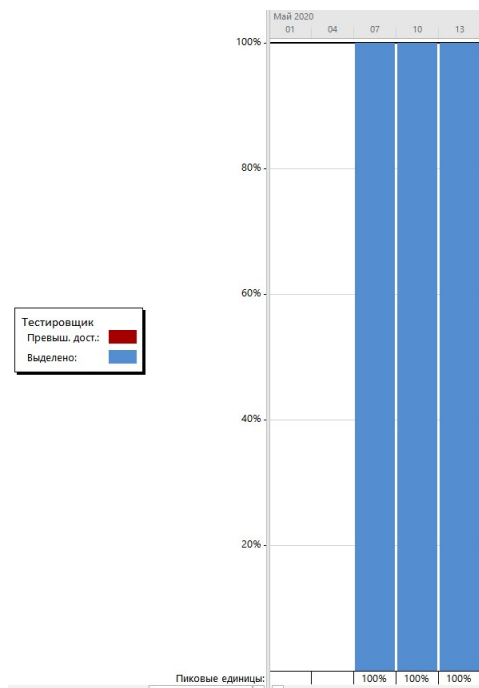


Рисунок 5 – График загрузки ресурса «Тестировщик» по дням

Как видно из графиков, ни один из работников не привлекается к работе сверхурочно.

Денежные затраты на выполнение каждой задачи отображены на рисунке 6 в виде гистограммы.

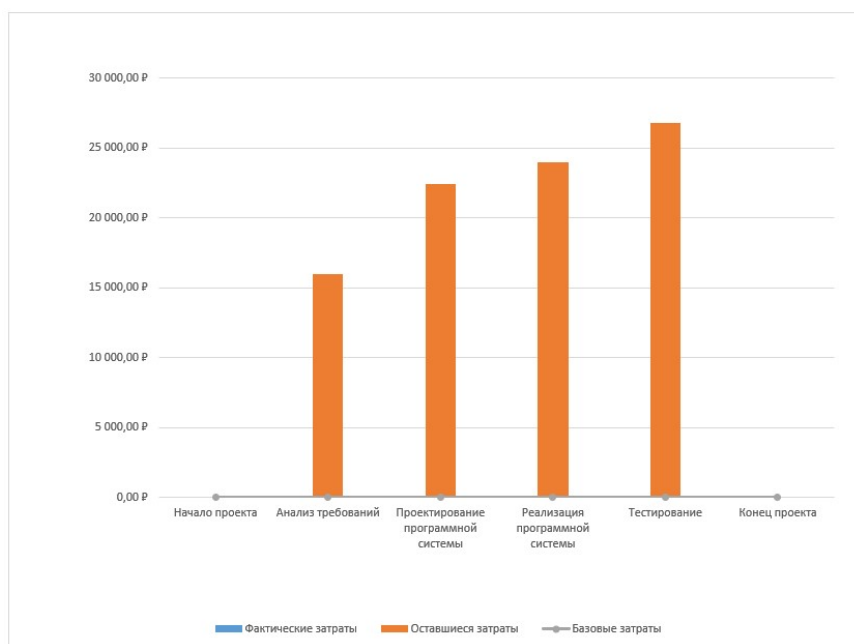


Рисунок 6 – Гистограмма денежных затрат на выполнение задач

Самой затратной опорной задачей является «тестирование». Затем, по убывающей: «реализация», «проектирование», «анализ требований».

На рисунке 7 представлена гистограмма загруженности ресурсов.

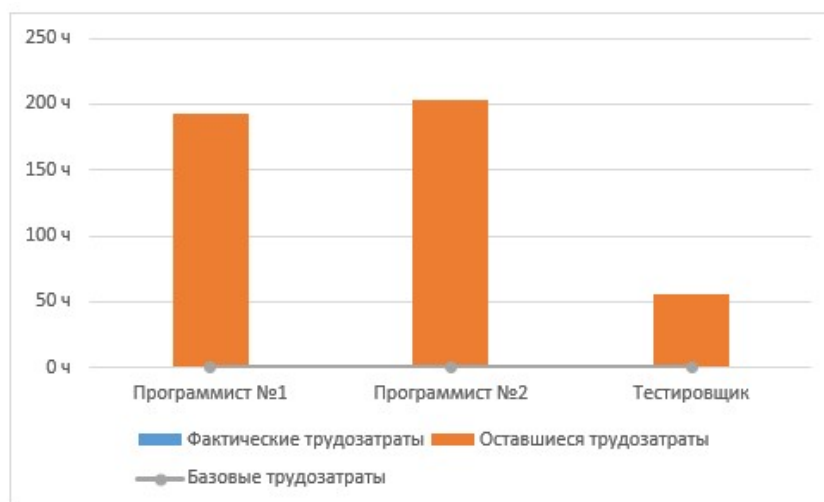


Рисунок 7 – Гистограмма загруженности ресурсов

Наименее загруженным ресурсом является «Тестировщик». «Программист №2» – наиболее загруженный ресурс, однако «Программист №1» загружен приблизительно на столько же.

Необходимый фонд рабочего времени составляет 451,44 человеко-часов. Бюджет проекта оценен в 89 179,20 рублей. Спланированная длительность разработки проекта составляет 29 рабочих дней.

### **3 Анализ предметной области и требований к программе оценки временных характеристик алгоритмов сортировки**

#### **3.1 Анализ предметной области**

Алгоритм сортировки – это алгоритм, принимающий на вход упорядоченное множество (последовательность) и возвращающий на выходе другое упорядоченное множество, состоящее из тех же элементов, что и входное, но в порядке неубывания/невозрастания [5].

Программа оценки временных характеристик алгоритмов сортировок – это программное обеспечение, позволяющее количественно сравнить оценки времени выполнения тестируемых алгоритмов сортировки.

Программное оценивание осуществляется путем экспериментального выполнения алгоритма с запоминанием конкретного времени, которое было затрачено при данном выполнении. Такой метод носит название «тест производительности» [6]. В зарубежной литературе для обозначения в том числе такого метода используется термин «benchmark», который также употребляется в русскоязычной научно-популярной литературе в виде слова «бенчмарк» [7]. Следует отметить, что при реализации такого метода необходимо свести к минимуму использование ресурсов процессора прочими процессами для того, чтобы измеренное время зависело только от исполнения операций, необходимых для выполнения сортировки.

Таким образом, задача теста производительности алгоритмов сортировки сводится к созданию массива, его сортировке и выдаче результатов. Следовательно, в предметной области приложений выбранного класса необходимо особо выделить следующие сущности:

- «массив»;
- «метод генерации массивов»;
- «метод сортировки массивов».

Модель предметной области расположена на рисунке 8.



Рисунок 8 – Модель предметной области

«Массив» содержит в себе последовательность целых чисел. «Метод генерации массивов» порождает сущность «массив» по определенным правилам. «Метод сортировки массивов» модифицирует «массив» путем его сортировки в соответствии с определенным алгоритмом сортировки.

Полноценно оценить алгоритм можно лишь при динамической длине сортируемой последовательности, что предоставит исследователю сведения о том, как меняется время, затраченное на сортировку, при изменении длины входящей последовательности [8]. Данные о затраченном времени удобно воспринимаются в виде графика времени от длины последовательности.

Возможность выбора типа генерации последовательностей является важным свойством подобного программного обеспечения – алгоритмы проявляют свои временные характеристики по-разному, в зависимости от класса последовательности, переданной на вход (полностью упорядоченная, частично упорядоченная, полностью неупорядоченная и т.д.).

Возможность одновременного запуска множества тестов (разные алгоритмы сортировки с разными классами и длинами последовательностей)

предоставляет пользователю дополнительные удобства при взаимодействии с программой.

Технология параллельного выполнения тестов (использование максимальной мощности процессора без влияния на результаты исследования) существенно сокращает время, которое необходимо потратить на получение результатов теста; внедрение такой технологии в программное обеспечение является очевидным достоинством приложения.

Количество доступных для исследования алгоритмов напрямую влияет на потенциальную полезность программного обеспечения (чем больше доступных методов сортировки, тем оправданнее будет результат исследования и следующий за ним выбор внедряемого алгоритма). Возможность добавления в ПО новых исследуемых алгоритмов позволяет расширить круг заинтересованных пользователей, отдалает срок морального износа приложения, повышает удобство в модернизации программы.

Запуск программы изнутри операционной системы неизбежно ведет к некоторой дополнительной погрешности измерений, так как обеспечение работы ОС и множества фоновых процессов требует ресурсов процессора. Для уменьшения величины погрешности измерения могут проводиться в цикле повторно для нахождения минимальной оценки, которая будет наиболее близкой к реальной.

Так как существует вероятность получения неблагоприятной для алгоритма последовательности при полностью случайной генерации, в программе может быть предусмотрена возможность генерации нескольких разных последовательностей по одному и тому же методу генерации для нахождения средней оценки, что позволит снизить искажение измерения.

Таким образом, были выделены основные аспекты программного обеспечения, предназначенного для оценки временных характеристик алгоритмов сортировки. Сравнение того, как они реализованы (в первую очередь количественно) в исследуемых аналогичных продуктах, позволит выделить среди них наилучший аналог, на базе характеристик которого будет построена стратегия повышения качества разрабатываемого продукта.

### **3.2 Анализ программ-аналогов оценки временных характеристик алгоритмов сортировки**

Анализ аналогичных программных продуктов позволит выявить сильные и слабые сторон представителей программ данного класса. На основе выполненного анализа выявляется наилучший продукт (прототип), на базе характеристик которого (в первую очередь количественных) составляется стратегия повышения качества разрабатываемого продукта. Следование этой стратегии при разработке позволит программному продукту выйти на рынок в качестве наилучшего представителя приложений выбранного класса.

Было принято решение изучить представителей программ оценки временных характеристик алгоритмов сортировки в соответствии с аспектами, выделенными при анализе предметной области; изучение будет подытожено сравнительным количественным анализом представленных приложений и построением стратегии повышения качества разрабатываемого продукта.

#### **3.2.1 Анализ приложения «Benchmarking Sorting Algorithms»**

«Benchmarking Sorting Algorithms» [9] – простое приложение с открытым кодом, выложенное на ресурсе «github.com» пользователем под именем «arisath». Продукт представляет из себя консольное приложение, работающее под управлением виртуальной машины «Java». Программа однократно выполняет замеры времени, требуемое для сортировки каждым из пяти доступных алгоритмов.

Преимущества:

- бесплатный продукт;
- открытый код;
- простая реализация.

Недостатки:

- отсутствует настройка параметров исследований;
- жестко заданные параметры существенно ограничивают ценность исследований, которые можно произвести с помощью данного продукта;

- нет возможности расширения функциональности приложения без модификации исходного кода.

Таким образом, данный продукт представляет из себя довольно примитивного представителя приложений выбранного класса.

### 3.2.2 Анализ приложения «Sorting benchmark»

«Sorting benchmark» (с) 2017 Burak Karakan [10] – простое приложение с открытым кодом, выложенное в свободный доступ на ресурсе «github.com» пользователем под именем «karakanb». Программа принимает на вход путь к текстовому файлу с входящей последовательностью, путь к файлу, в котором необходимо записать результат тестирования и название метода сортировки. Приложение сортирует входящую последовательность с помощью указанного метода и записывает в выходящий файл затраченное время.

Преимущества:

- бесплатный продукт;
- открытый код;
- простая реализация.

Недостатки:

- работа с приложением подразумевает ручное создание и модификацию текстовых файлов;
- в рамках одного теста предоставляется возможность применения только одного метода сортировки;
- отсутствует встроенная возможность генерации последовательностей;
- нет возможности расширения функциональности приложения без модификации исходного кода.

Данный продукт, так же, как и предыдущий, реализован достаточно примитивно.

### **3.2.3 Анализ приложения «Анализ алгоритмов внутренней сортировки»**

«Анализ алгоритмов внутренней сортировки» (с) 2005 Слета В., Копов С. [11] – оконное приложение с закрытым кодом для платформы «Windows», предназначенное для сравнительного анализа временных характеристик алгоритмов сортировки. Приложение предоставляет удобный интерфейс: позволяет выбрать один из четырех методов генерации последовательности и несколько алгоритмов сортировки одновременно (максимум до шести из шести возможных); данные отображаются в виде графиков; присутствует возможность повторных измерений для снижения степени погрешности; при каждом тесте измерения проводятся на диапазоне значений длины сортируемой последовательности.

Преимущества:

- бесплатный продукт;
- возможность выбора метода генерации;
- возможность выбора сразу нескольких алгоритмов сортировки;
- предусмотрен механизм повторного измерения;
- удобный оконный интерфейс;
- вывод результатов в виде графика.

Недостатки:

- закрытый код;
- нет многопоточности;
- нет возможности добавления пользовательских методов сортировки и генерации.

В отличии от ранее рассмотренных продуктов, данное приложение выполнено довольно качественно.

### **3.2.4 Сравнительный анализ исследованных программ-аналогов**

Количественное сравнение выбранных программ-аналогов позволит выбрать среди них наилучшего представителя, на основе характеристик

которого, будет составлена стратегия повышения качества разрабатываемого продукта.

В таблице 1 приведены результаты количественного сравнения исследуемых программных продуктов.

Таблица 1 – Результаты количественного сравнения программных продуктов

| ПО                                                            | «Benchmarking Sorting Algorithms» | «Sorting benchmark» | «Анализ алгоритмов внутренней сортировки» |
|---------------------------------------------------------------|-----------------------------------|---------------------|-------------------------------------------|
| К-во доступных методов сортировки                             | 5                                 | 3                   | 6                                         |
| К-во одновременно выбранных методов сортировки                | 5                                 | 1                   | 1-6                                       |
| К-во доступных методов генерации                              | 1                                 | 0                   | 4                                         |
| К-во одновременно выбранных методов генерации                 | 1                                 | 0                   | 1                                         |
| К-во задействованных процессоров                              | 1                                 | 1                   | 1                                         |
| К-во повторных измерений                                      | 1                                 | 1                   | 1-50000                                   |
| К-во генераций                                                | 1                                 | 1                   | 1                                         |
| Длина последовательности                                      | 100000                            | Не ограничено       | 100-100000                                |
| К-во возможных длин последовательностей в рамках одного теста | 1                                 | 1                   | 1-100                                     |

Результаты произведенного количественного сравнения позволяют утверждать, что лучшим из исследуемых программных продуктов является «Анализ алгоритмов внутренней сортировки»: из девяти представленных категорий приложение оказалось наилучшим в пяти, поделило первое место в трех и поделило второе место в одном. Скриншот этого приложения расположен на рисунке 9.

Данное приложение будет использовано как прототип, на основе характеристик которого будет составлена стратегия повышения качества разрабатываемой программы оценки временных характеристик алгоритмов сортировки.

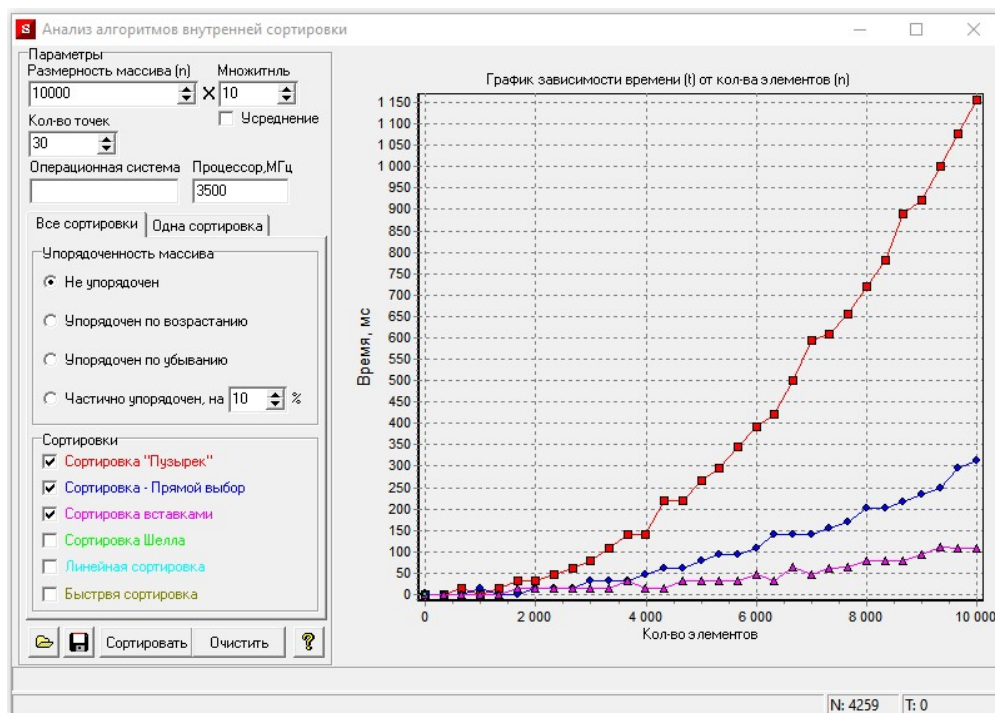


Рисунок 9 – Скриншот приложения-прототипа «Анализ алгоритмов внутренней сортировки» (с) 2005 Слета В., Копов С.

Для повышения качества разрабатываемого продукта необходимо улучшить определенные количественные характеристики приложения по сравнению с выбранным прототипом.

### 3.2.5 Стратегии повышения качества разрабатываемой программы

На основе произведенного анализа приложений-аналогов, были сформулированы задачи, позволяющие повысить качество продукта по сравнению с прототипом:

- снять ограничение на максимальное количество доступных методов сортировки и генерации последовательностей путем внедрения системы расширения приложения (расширение должно осуществляться с помощью плагинов, содержащих в себе дополнительные методы сортировки и генерации; расширение осуществляется без изменения исходного кода самого приложения);
- ограничить максимальное количество одновременно выбираемых методов сортировки и генерации последовательностей исключительно общим

поддерживаемым приложением количеством методов сортировки и генерации соответственно;

- внедрить поддержку многопоточности, т.е. ограничить максимальное количество задействованных процессоров исключительно общим количеством логических процессоров, имеющихся на конкретной машине пользователя (повышение производительности приложения этим способом должно ускорять выполнение тестов и, в то же время, обязано осуществлять это без какого-либо воздействия на результаты исследования);

- внедрить механизм дополнительных генерируемых последовательностей в каждой точке измерения, т.е. увеличить количество генераций на точку до заведомого чрезмерного значения (например, 10 000).

Следование составленной стратегии повышения качества продукта позволит разрабатываемому приложению выйти на рынок как более качественному представителю программ выбранного класса.

### **3.3 Анализ функциональных требований**

В процессе анализа функциональных требований необходимо определить, что именно программная система должна уметь делать, и какие способы воздействия на нее есть у пользователя [12].

Из поставленных требования следует, что прежде всего программа должна предоставлять пользователю возможность сравнения временных характеристик выбранных методов сортировки.

Для настройки процесса исследования пользователю должны быть доступны следующие действия:

- выбор методов сортировки и генерации массивов (причем, в соответствие с поставленными требованиями пользователь должен иметь возможность выбирать несколько методов сортировки и генерации одновременно);

- указание левой и правой границы длин массивов, на которых будет производиться исследование (первая исследуемая точка, то есть длина

сортируемого массива, должна находиться в левой границе, последняя – как можно правее, но не находясь за пределами правой границы);

- установка количества точек, в которых будет производиться тестирование (в соответствии с указанными границами и количеством шагов системой определяется интервал между точками исследования);
- выбор коэффициента усреднения для понижения погрешности.

Запуск процесса исследования должен осуществляться по инициативе пользователя. Проводимое исследование, очевидно, должно быть осуществлено в соответствии с теми параметрами, которые указал пользователь.

Диаграмма вариантов использования расположена на рисунке 10.

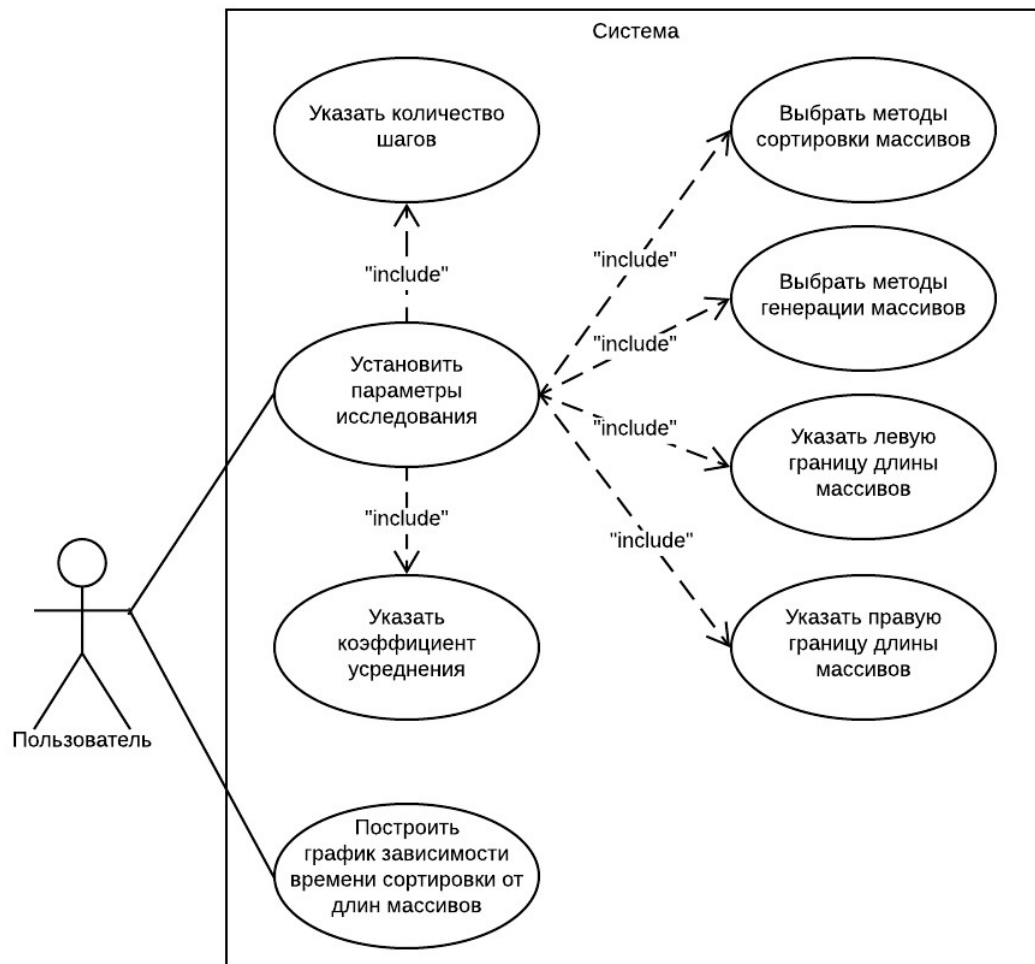


Рисунок 10 – Диаграмма вариантов использования

Диаграмма структурирует поставленные функциональные требования: все функциональные возможности, доступные пользователю, отображены в виде прецедентов и сгруппированы.

Спецификация прецедента «Построить график зависимости времени сортировки от длин массивов» представлена на рисунке 11.

|                                                                                                                                                                                                                                                                                                                                                                                   |
|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Прецедент: «Построить график зависимости времени сортировки от длин массивов»                                                                                                                                                                                                                                                                                                     |
| ID: 1                                                                                                                                                                                                                                                                                                                                                                             |
| Краткое описание:<br>Пользователь задаёт параметры измерений и запускает построение графиков.                                                                                                                                                                                                                                                                                     |
| Главные актёры:<br>Пользователь.                                                                                                                                                                                                                                                                                                                                                  |
| Второстепенные актёры:<br>Нет.                                                                                                                                                                                                                                                                                                                                                    |
| Предусловия:<br>Приложение запущено, измерения не запущены.                                                                                                                                                                                                                                                                                                                       |
| Основной поток:<br>1. Пользователь выбирает желаемые методы сортировки.<br>2. Пользователь выбирает желаемые методы генерации.<br>3. Пользователь задаёт левую границу графиков.<br>4. Пользователь задаёт правую границу графиков.<br>5. Пользователь задаёт количество шагов.<br>6. Пользователь задаёт коэффициент усреднения.<br>7. Пользователь запускает процесс измерения. |
| Постусловия:<br>1. Графики построены.                                                                                                                                                                                                                                                                                                                                             |
| Альтернативные потоки:<br>Нет.                                                                                                                                                                                                                                                                                                                                                    |

Рисунок 11 – Спецификация прецедента «Построить график зависимости времени сортировки от длин массивов»

Представленный сценарий описывает, каким образом должно осуществляться исполнение прецедента «Построить график зависимости времени сортировки от длин массивов». Пользователь последовательно настраивает определенные параметры исследования, затем инициирует сам процесс исследования. В итоге пользователь наблюдает результаты исследования в виде построенных графиков.

Функциональные требования были проанализированы, в результате чего они были подробно описаны при помощи диаграммы вариантов использования и сценария основного прецедента.

### **3.4 Анализ нефункциональных требований**

В результате анализа нефункциональных требований было принято решение осуществить разработку программы на языке программирования «C#» [13]. Главными достоинствами языка, повлиявшими на выбор, являются его надежность, простота использования и широкие возможности по использованию библиотек «.NET Framework» [14].

Система плагинов, которую необходимо внедрить, будет реализована в формате dll-библиотек [15]. При таком подходе код подключаемых плагинов заключается внутрь dll-файлов, которые помещаются в особое хранилище плагинов, доступное для исполняемой программы.

С помощью плагинов пользователи смогут добавлять дополнительные алгоритмы сортировки и генерации массивов, необходимые для исследования.

## 4 Проектирование программы оценки временных характеристик алгоритмов сортировки

Многие аспекты проектирования удобно отображаются в графическом виде, поэтому в данной работе для описания моделируемой системы будут применяться диаграммы UML 2.0. Для построения таких диаграмм будет использован онлайн-ресурс «Lucidcharts» [16]. Выбор сделан в пользу этого ресурса благодаря его удобному интерфейсу и наличию встроенного облачного хранилища документов.

### 4.1 Проектирование пользовательского интерфейса

Интерфейс должен не только соответствовать всем поставленным требованиям, но и быть удобным для пользователя. Было принято решение осуществить интерфейс в виде оконного приложения. Для главного окна создаваемого программного обеспечения был разработан макет.

Макет интерфейса приложения представлен на рисунке 12. На этом рисунке также пронумерованы функциональные области интерфейса.

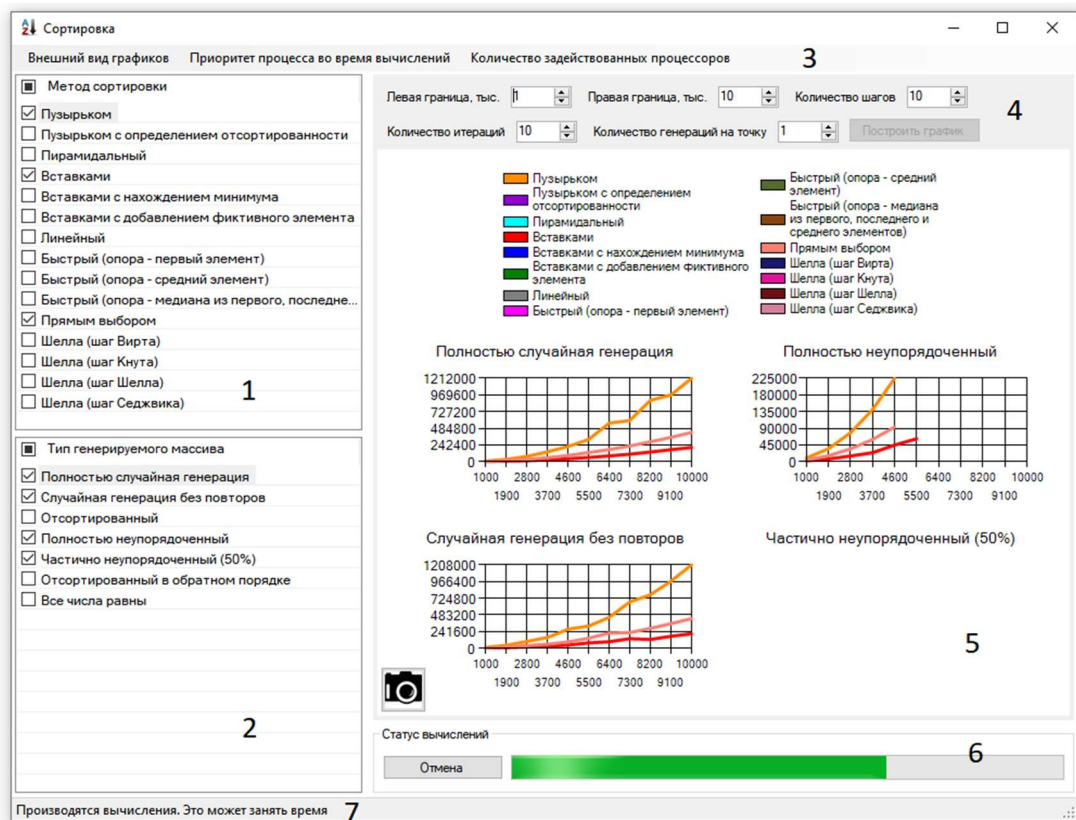


Рисунок 12 – Макет графического интерфейса

Назначение обозначенных на макете функциональных областей определено в таблице 2.

Таблица 2 – Назначение областей графического интерфейса

| Номер | Назначение области                                                                           |
|-------|----------------------------------------------------------------------------------------------|
| 1     | Перечень доступных методов сортировки                                                        |
| 2     | Перечень доступных методов генерации                                                         |
| 3     | Настройки приложения                                                                         |
| 4     | Настройки исследования и его запуск                                                          |
| 5     | Вывод результатов исследования в графическом виде (присутствует кнопка сохранения скриншота) |
| 6     | Вывод прогресса исследования (присутствует кнопка отмены)                                    |
| 7     | Отображение статуса приложения                                                               |

Среди настроек приложения должны присутствовать следующие:

- выбор внешнего вида графика (в виде ломанной линии, кривой линии и столбцов);
- выбор приоритета процесса;
- выбор количества задействованных процессоров (в соответствии со стратегией повышения качества продукта – вплоть до максимального возможного на машине, на которой запущено приложение).

Методы генерации и сортировки, которые должны быть задействованы при исследовании, выбираются путем нажатия на соответствующую позицию в перечнях методов. В соответствии со стратегией повышения качества продукта, одновременно может быть выбрано произвольное количество методов, но, очевидно, не более чем их доступно вообще.

Результаты исследования выводятся в виде графиков на соответствующей области. Каждому выбранному методу генерации соответствует одна область графиков (невыведенные методы никак не отображаются). Каждому методу сортировки соответствует цвет графиков, отображенный на легенде. Рисуются графики только тех методов сортировки, которые выбраны. Построенные графики можно сохранить в виде скриншота при помощи соответствующей кнопки.

Результаты исследования должны быть доступны еще до момента окончания всего исследования; прогресс исследования отображен на соответствующей полосе прогресса. Пользователь должен иметь возможность останавливать исследование с помощью кнопки отмены.

Спроектированный интерфейс соответствует поставленным требованиям и составленной стратегии повышения качества продукта.

## 4.2 Проектирование классов программы

При проектировании системы классов в первую очередь необходимо тем или иным образом отобразить на ней выделенные при анализе предметной области сущности, а именно:

- «массив»;
- «метод генерации массивов»;
- «метод сортировки массивов».

Сущность «массив» будет реализована с помощью стандартного класса целочисленного массива в языке «C#», т.е. «int[]» [17]. Это решение объясняется тем, что все функции, необходимые для работы с массивами (создание массива определенной длины, редактирование элементов), уже реализованы в языке на синтаксическом уровне.

В то же время сущности «метод генерации массивов» и «метод сортировки массивов», выявленные при анализе предметной области, будут реализованы с помощью пользовательских классов. Эти классы должны быть абстрактными, так как сами по себе они не содержат конкретных алгоритмов создания массивов и их сортировки соответственно; эти алгоритмы должны быть заключены внутри их классов-потомков.

Так как указанные сущности сами по себе не содержат логики их систематического использования в рамках поставленной задачи, необходимо описать логику во внешнем объекте. В соответствии с выбранной парадигмой программирования, ООП, логика проведения процесса исследования должна быть вынесена в определенный класс [18].

Диаграмма основных классов расположена на рисунке 13.

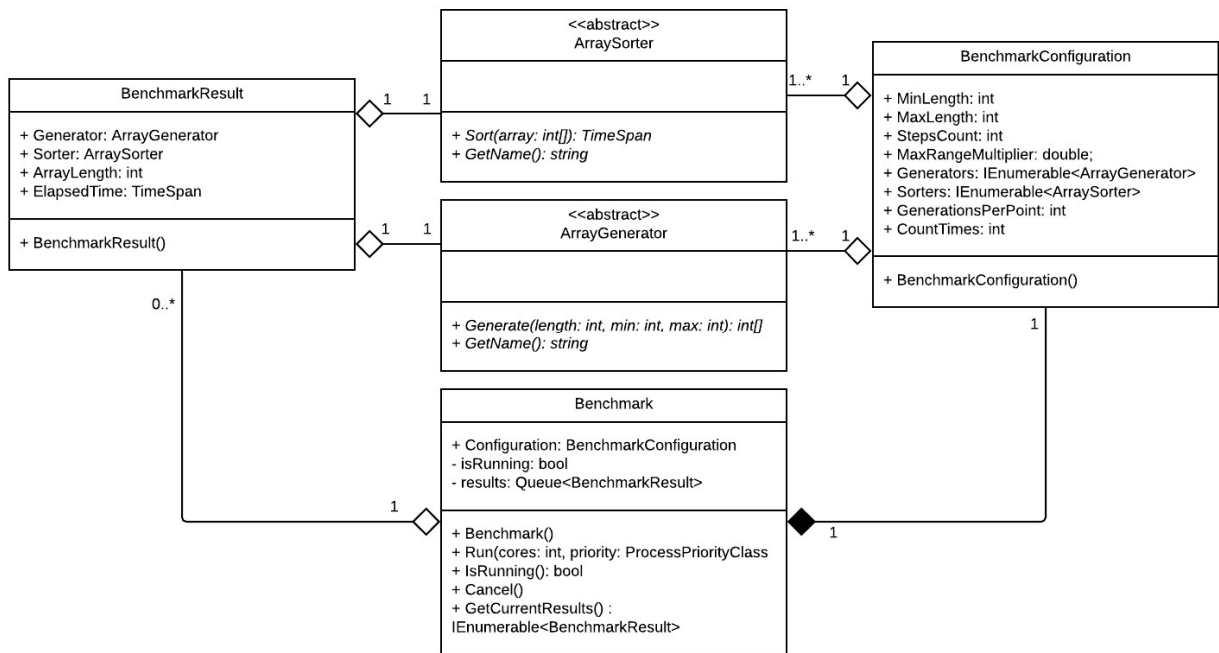


Рисунок 13 – Диаграмма классов

Необходимо подробно описать каждый класс в отдельности. Осуществим это с помощью ряда таблиц, уточняющих диаграмму классов. В таблице 3 приведено назначение спроектированных классов.

Таблица 3 – Назначение классов

| Класс                  | Назначение                                |
|------------------------|-------------------------------------------|
| ArraySorter            | Абстрактный сортировщик массива           |
| ArrayGenerator         | Абстрактный генератор массива             |
| Benchmark              | Класс, отвечающий за проведение измерений |
| BenchmarkConfiguration | Конфигурация исследования                 |
| BenchmarkResult        | Результат элементарного исследования      |

Абстрактный класс «ArraySorter» предназначен для сортировки массивов. Главный метод «Sort» сортирует массив и возвращает затраченное время. Реализация этого метода должна быть осуществлена в классе-наследнике (реализацией может быть, например, быстрая сортировка, сортировка Шелла и т.д.). Также этот класс содержит метод, который возвращает название алгоритма сортировки; этот метод будет использоваться графическим интерфейсом. Назначение членов класса «ArraySorter» приведено в таблице 4.

Таблица 4 – Назначение членов класса «ArraySorter»

| Сигнатура                                        | Тип   | Назначение                                              |
|--------------------------------------------------|-------|---------------------------------------------------------|
| <code>abstract TimeSpan Sort(int[] array)</code> | Метод | Сортирует массив «array» и возвращает затраченное время |
| <code>abstract String GetName()</code>           | Метод | Возвращает название сортировщика                        |

Абстрактный класс «ArrayGenerator» предназначен для создания массивов. Главный метод «Generate» создает массив в соответствие с указанными параметрами (длина, минимальное и максимальное допустимое значение элементов массива). Реализация этого метода должна быть осуществлена в классе-наследнике (реализацией может быть, например, полностью случайная генерация, генерация упорядоченной последовательности и т.д.). Так же, как и класс сортировщика, этот класс содержит абстрактный метод для возврата названия. Назначение членов класса «ArrayGenerator» приведено в таблице 5.

Таблица 5 – Назначение членов класса «ArrayGenerator»

| Сигнатура                                                          | Тип   | Назначение                                                                        |
|--------------------------------------------------------------------|-------|-----------------------------------------------------------------------------------|
| <code>abstract int[] Generate(int length, int min, int max)</code> | Метод | Генерирует массив длины length, элементы которого не менее «min» и не более «max» |
| <code>abstract String GetName()</code>                             | Метод | Возвращает название генератора                                                    |

Класс «Benchmark» содержит в себе основную логику проведения исследования. С помощью методов этого класса можно запускать и отменять исследование, а также получать его результаты. Было принято решение вынести параметры и результаты исследования в отдельные классы для упрощения структуры основного класса. Конфигурация исследования вынесена в класс «BenchmarkConfiguration», а результаты – в «BenchmarkResult». Назначение членов класса «Benchmark» приведено в таблице 6.

Таблица 6 – Назначение членов класса «Benchmark»

| Сигнатура                                          | Тип         | Назначение                                                                                                                          |
|----------------------------------------------------|-------------|-------------------------------------------------------------------------------------------------------------------------------------|
| BenchmarkConfiguration                             | Свойство    | Конфигурация исследования                                                                                                           |
| bool isRunning                                     | Поле        | Состояние исследования: запущено/нет                                                                                                |
| Queue<BenchmarkResult> results                     | Поле        | Очередь результатов исследования                                                                                                    |
| Benchmark()                                        | Конструктор | Создает стандартный экземпляр класса                                                                                                |
| void Run(int cores, ProcessPriorityClass priority) | Метод       | Запускает процесс исследования с возможностью задействовать не более, чем «cores» ядер процессора с приоритетом процесса «priority» |
| void Cancel()                                      | Метод       | Прерывает запущенное исследование                                                                                                   |
| bool IsRunning()                                   | Метод       | Возвращает состояние исследования: запущено/нет                                                                                     |
| IEnumerable<BenchmarkResult> GetCurrentResults()   | Метод       | Возвращает доступные на данный момент результаты исследования                                                                       |

Класс «BenchmarkConfiguration» нужен для хранения и изменения параметров исследования. Благодаря тому, что все настраиваемые свойства исследования хранятся в этом классе, основной класс («Benchmark») избавлен от необходимости иметь множество полей, что облегчает его структуру. Назначение членов класса «BenchmarkConfiguration» приведено в таблице 7.

Таблица 7 – Назначение членов класса «BenchmarkConfiguration»

| Сигнатура                              | Тип         | Назначение                                                        |
|----------------------------------------|-------------|-------------------------------------------------------------------|
| int MinLength                          | Свойство    | Левая граница исследования                                        |
| int MaxLength                          | Свойство    | Правая граница исследования                                       |
| int StepsCount                         | Свойство    | Количество точек исследования                                     |
| double MaxRangeMultiplier              | Свойство    | Во сколько раз наибольшее число массива может превышать его длину |
| IEnumerable<ArrayGenerator> Generators | Свойство    | Генераторы массивов, которые задействуются при исследовании       |
| IEnumerable<ArraySorter> Sorters       | Свойство    | Сортировщики массивов, которые задействуются при исследовании     |
| int GenerationsPerPoint                | Свойство    | Количество генераций на каждую точку                              |
| int CountTimes                         | Свойство    | Количество повторных измерений                                    |
| BenchmarkConfiguration()               | Конструктор | Создает стандартный экземпляр класса                              |

Класс «BenchmarkResult» нужен для хранения результатов элементарного исследования, т.е. сортировки одного массива. В этом классе хранится длина отсортированного массива, затраченное на сортировку время и ссылки на примененные при исследовании методы генерации и сортировки. Назначение членов класса «BenchmarkResult» приведено в таблице 8.

Таблица 8 – Назначение членов класса «BenchmarkResult»

| Сигнатура                | Тип         | Назначение                                                                        |
|--------------------------|-------------|-----------------------------------------------------------------------------------|
| ArrayGenerator Generator | Свойство    | Генератор, с помощью которого был создан подвергшийся исследованию массив         |
| ArraySorter Sorter       | Свойство    | Сортировщик, с помощью которого был отсортирован подвергшийся исследованию массив |
| int ArrayLength          | Свойство    | Длина подвергнувшегося исследованию массива                                       |
| TimeSpan ElapsedTime     | Свойство    | Время, затраченное на сортировку подвергнувшегося исследованию массива            |
| BenchmarkResult()        | Конструктор | Создает стандартный экземпляр класса                                              |

Таким образом, с помощью диаграммы классов и ряда уточняющих таблиц были спроектированы и описаны 5 классов, реализация которых требуется при разработке системы.

#### 4.3 Проектирование процесса исследования алгоритмов сортировки

Процесс исследования в целом включает в себя настройку параметров исследования, осуществление сортировки и выдачу результатов.

Диаграмма последовательности, отображающая процесс исследования в целом, расположена на рисунке 14. Из представленной диаграммы видно, что процессу исследования должна предшествовать настройка конфигурации, которую пользователь осуществляет посредством графического интерфейса. Пользователь инициирует начало процесса исследования и наблюдает за ним с помощью графиков, которые выводятся на экран.

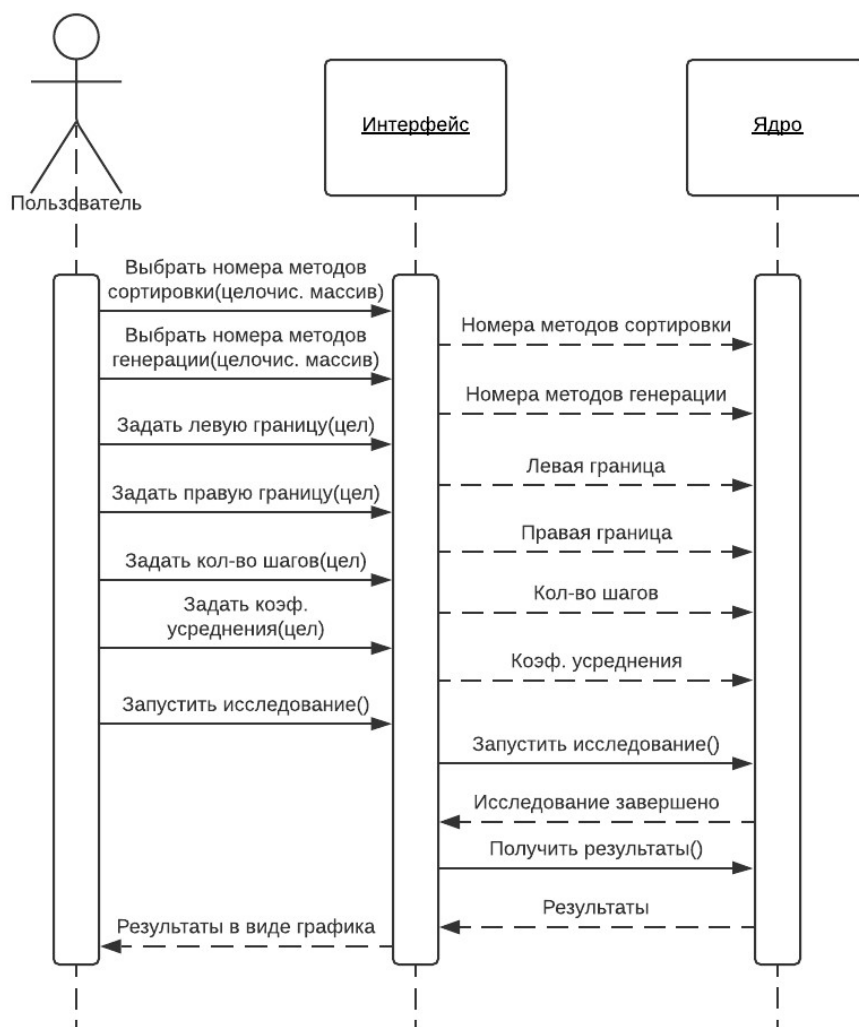


Рисунок 14 – Диаграмма последовательности процесса исследования

В укрупненном виде процесс непосредственного исследования должен включать в себя следующие пункты:

- генерацию определенного количества массивов каждым выбранным генератором для каждой указанной точки исследования (т.е. для всех выбранных длин массива);
- сортировку каждого получившегося массива каждым из выбранных методов сортировки;
- выдачу результатов по каждому проведенному элементарному исследованию.

Так как приложение должно быть оптимизировано для увеличения быстродействия программы, проведение элементарных исследований, т.е.

единичных сортировок одного массива, будет осуществляться параллельно друг другу. Для распараллеливания действий будет использован встроенный в «.NET Framework» механизм планирования задач. Так как оптимизация должна осуществляться без какого-либо искажения результатов, то каждое выделенное для исследования ядро процессора должно выполнять лишь одну задачу одновременно; следующая задача поступает на исполнение только после того, как ядро освободилось.

Сама по себе сортировка одного массива, занимает лишь небольшой промежуток времени, следовательно, использовать отдельные потоки как основу для выполнения задач нецелесообразно. Поэтому выбор был сделан в пользу пула потоков [19].

Выдача результатов не должна задерживаться до окончания исследования, иными словами, пользователь должен иметь возможность наблюдать за процессом исследования в реальном времени. Таким образом, одно из ядер процессора должно быть выделено для регулярного обновления интерфейса. Предварительно оценим необходимый интервал обновления графиков в две секунды.

Укрупненная схема процесса исследования представлена на диаграмме деятельности, которая расположена на рисунке 15.

Программа определяет интервал между соседними точками исследования, затем начинает построение задач исследования в каждой из точек. С помощью каждого из выбранных методов генерации создается определенное количество массивов. Каждый из выбранных методов сортировки должен отсортировать каждый массив указанное количество раз и вернуть затраченное время. Множество задач, таким образом, проецируется на множество комбинаций из всех массивов и всех выбранных методов сортировки. Каждая из таких задач планируется к выполнению на предварительно созданном пуле потоков. После того, как все задачи запланированы, начинается их параллельное исполнение, в то время как отдельный поток выполняет регулярное обновление графиков в соответствии с вновь появившимися результатами исследования.

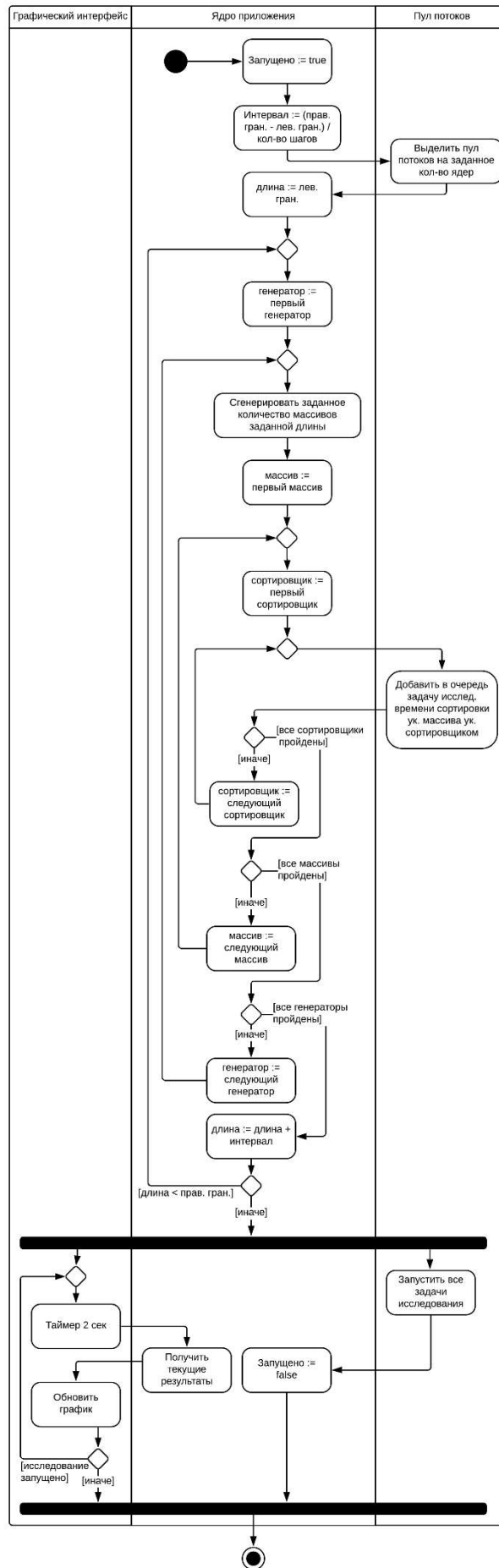


Рисунок 15 – Диаграмма деятельности процесса исследования

Таким образом, был спроектирован процесс исследования алгоритмов сортировки в рамках разрабатываемого приложения.

#### **4.4 Проектирование системы плагинов программы**

Для реализации системы плагинов будет использован механизм подключения dll-библиотек.

Часть кода, представляющего собой программный интерфейс для расширения приложения, будет вынесена в отдельную dll-библиотеку. В этой библиотеке будут заключены абстрактные классы сортировщика («ArraySorter») и генератора («ArrayGenerator») массивов.

Пользователь-программист, который заинтересован в написании плагинов, будет должен подключить эту библиотеку к своему проекту для написания реализаций нужных ему методов сортировки и генерации массивов. Ему будет необходимо скомпилировать свой проект как «библиотеку классов .NET», а получившийся dll-файл поместить в корневую папку приложения.

Все корректно созданные плагины, расположенные в корневой папке приложения, будут видны при запуске. Любой пользователь, обладающий плагином, сможет пользоваться им во время исследования временных характеристик алгоритмов сортировки массивов.

## 5 Реализация программы оценки временных характеристик алгоритмов сортировки

На основе результатов проектирования было разработано программное обеспечение оценки временных характеристик алгоритмов сортировки массивов.

Исходный код приложения расположен в приложении А.

Скриншоты программы расположены на рисунках 16 и 17.

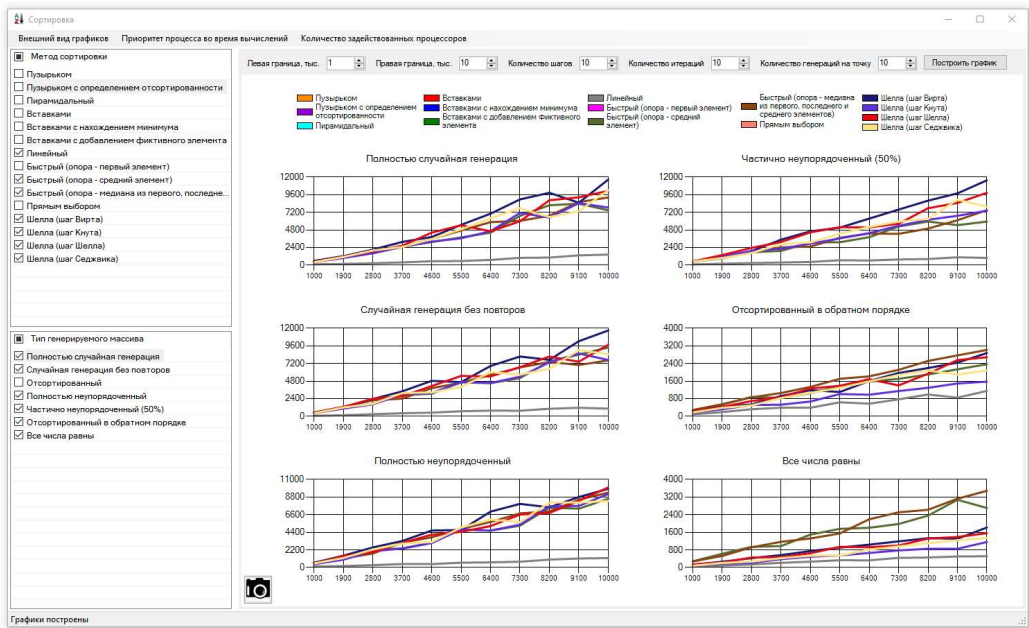


Рисунок 16 – Скриншот выполненного исследования

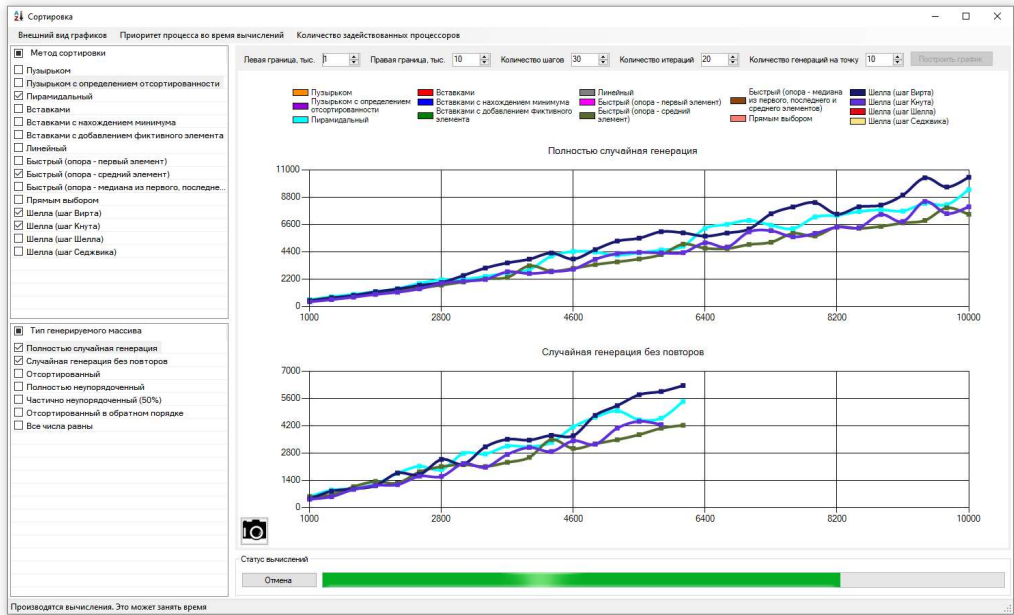


Рисунок 17 – Скриншот выполняемого исследования

Программная система включает в себя:

- основную исполняемую программу («Sorting.exe»);
- библиотеку, содержащую программный интерфейс приложения для расширения («api.dll»);
- семь стандартных плагинов.

Перечислим реализованные плагины:

- «insert.dll»;
- «bubble.dll»;
- «linear.dll»;
- «selsort.dll»;
- «shell.dll»;
- «heapsort.dll»;
- «qsort.dll».

Диаграмма компонентов, отображающая файловую структуру стандартного пакета приложения, изображена на рисунке 18.

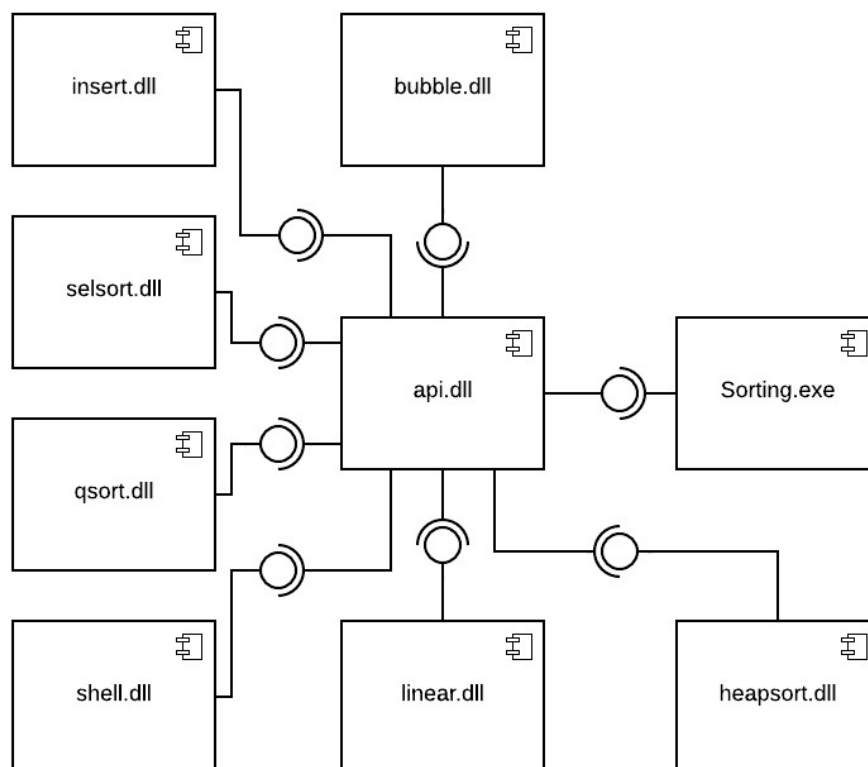


Рисунок 18 – Диаграмма компонентов

Отображенная на диаграмме структура реализованной системы соответствует составленному проекту.

5.1 Реализация исполняемой программы

Проект «Sorting» был скомпилирована в исполняемый файл «Sorting.exe».

Диаграмма классов, реализованных внутри проекта, отображена на рисунке 19.

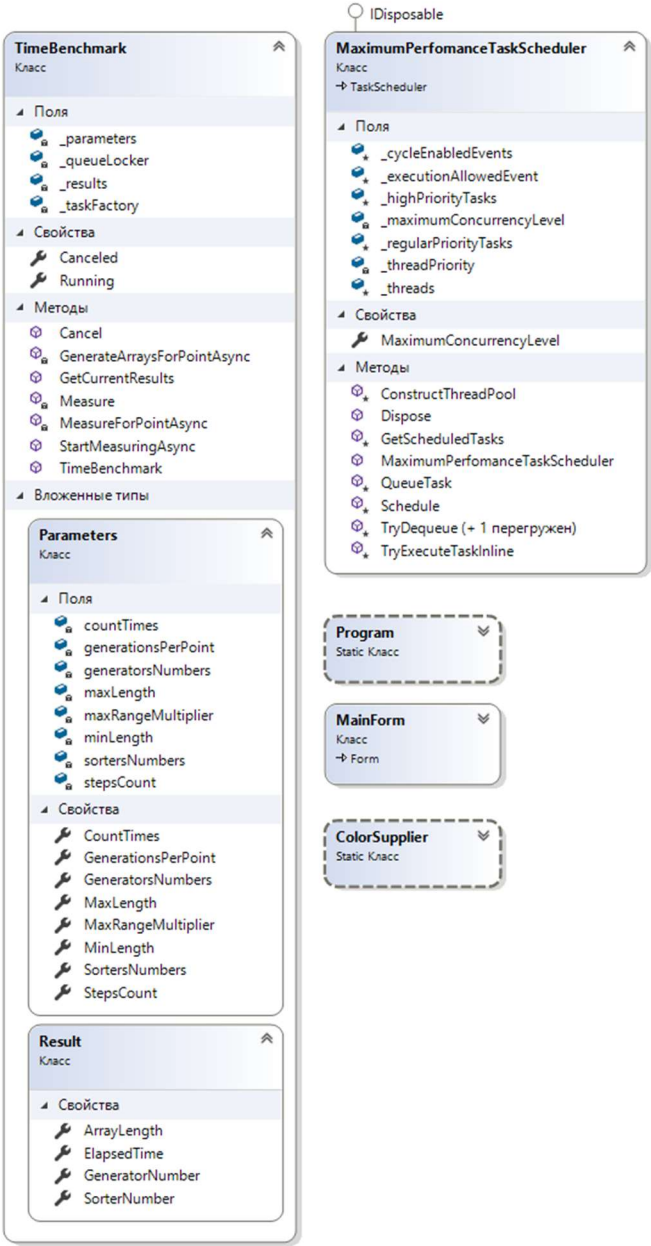


Рисунок 19 – Диаграмма классов проекта «Sorting»

Спроектированный класс «Benchmark» реализован реальным классом «TimeBenchmark»; «BenchmarkConfiguration» и «BenchmarkResult» – как вложенные классы «TimeBenchmark», то есть «TimeBenchmark.Parameters» и «TimeBenchmark.Result» соответственно.

Класс «MaximumPerformanceTaskScheduler» является реализацией описанного при проектировании планировщика задач на базе пула потоков. «Program» содержит входную точку приложения. «MainForm» – класс главного окна приложения. Статичный класс «ColorSupplier» нужен для получения цветов графиков, поэтому используется только в рамках логики графического интерфейса.

## 5.2 Реализация программного интерфейса приложения

Программный интерфейс приложения для расширения с помощью пользовательских плагинов (проект «SortingExtensionAPI») был скомпилирован в библиотеку «api.dll».

Диаграмма классов, реализованных внутри проекта, отображена на рисунке 20.

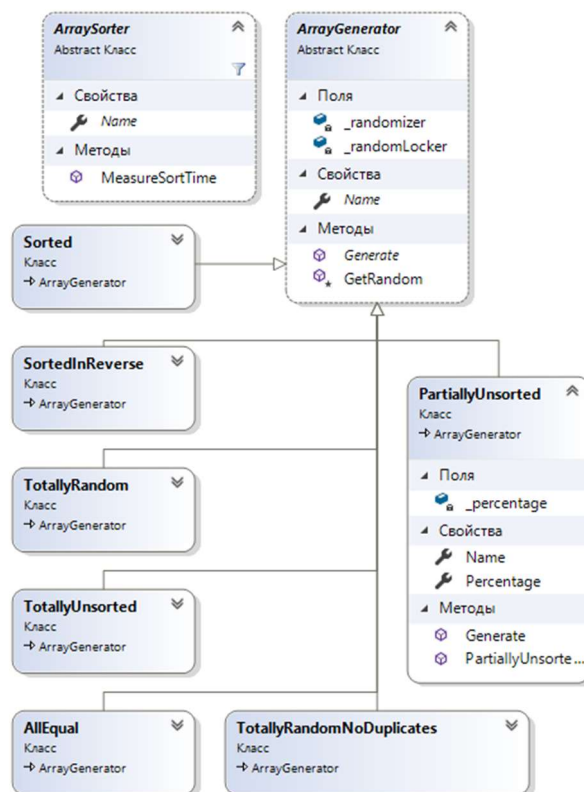


Рисунок 20 – Диаграмма классов проекта «SortingExtensionAPI»

Классы «ArraySorter» и «ArrayGenerator» являются реализованной версией спроектированных сортировщика и генератора массивов соответственно. Также внутри этой библиотеки содержатся наиболее часто встречаемые типы генераций (таблица 9).

Таблица 9 – Методы генерации массивов

| Название класса           | Тип генерируемого массива                                        |
|---------------------------|------------------------------------------------------------------|
| Sorted                    | Упорядоченный                                                    |
| SortedInReverse           | Упорядоченный в обратном порядке                                 |
| TotallyRandom             | Абсолютно случайный                                              |
| TotallyUnsorted           | Без единого элемента, который стоит на своем месте               |
| AllEqual                  | Все элементы равны                                               |
| TotallyRandomNoDuplicates | Случайный без повторяющихся элементов                            |
| PartiallyUnsorted         | Лишь некоторая определенная часть элементов стоит на своем месте |

Включение в необходимую библиотеку указанных генераторов не только упрощает файловую структуру программы (эти типы генераций настолько популярны, что нет никакого смысла выносить их в отдельный плагин – они все равно, скорее всего, потребуются при любом исследовании), но и позволяет использовать метод жесткого кодирования для генерации частично упорядоченных массивов. При выборе частично неупорядоченной генерации пользователем, на графическом интерфейсе появляется поле ввода процента неупорядоченности (рисунок 21).

☒ Тип генерируемого массива  
☒ Полностью случайная генерация  
☒ Случайная генерация без повторов  
☒ Отсортированный  
☒ Полностью неупорядоченный  
☒ Частично неупорядоченный (50%)  
☒ Отсортированный в обратном порядке  
☒ Все числа равны

Процентная доля неупорядоченности

50

Рисунок 21 – Выбор частично неупорядоченной генерации

Тем не менее, при надобности пользователи-программисты все же могут внедрять свои собственные алгоритмы генерации с помощью системы плагинов.

### 5.3 Реализация стандартных плагинов

Стандартные плагины были скомпилированы в семь dll-библиотек.

Плагин «qsort.dll» содержит в себе реализацию быстрой сортировки [20] с опорным элементом в трех вариантах: в начале (класс «PivotIs.First»), в середине (класс «PivotIs.Middle») и в медиане из первого, среднего и последнего элементов (класс «PivotIs.MedianOf3»). Библиотека является результатом компиляции проекта «QuickSort», диаграмма классов которого отображена на рисунке 22.

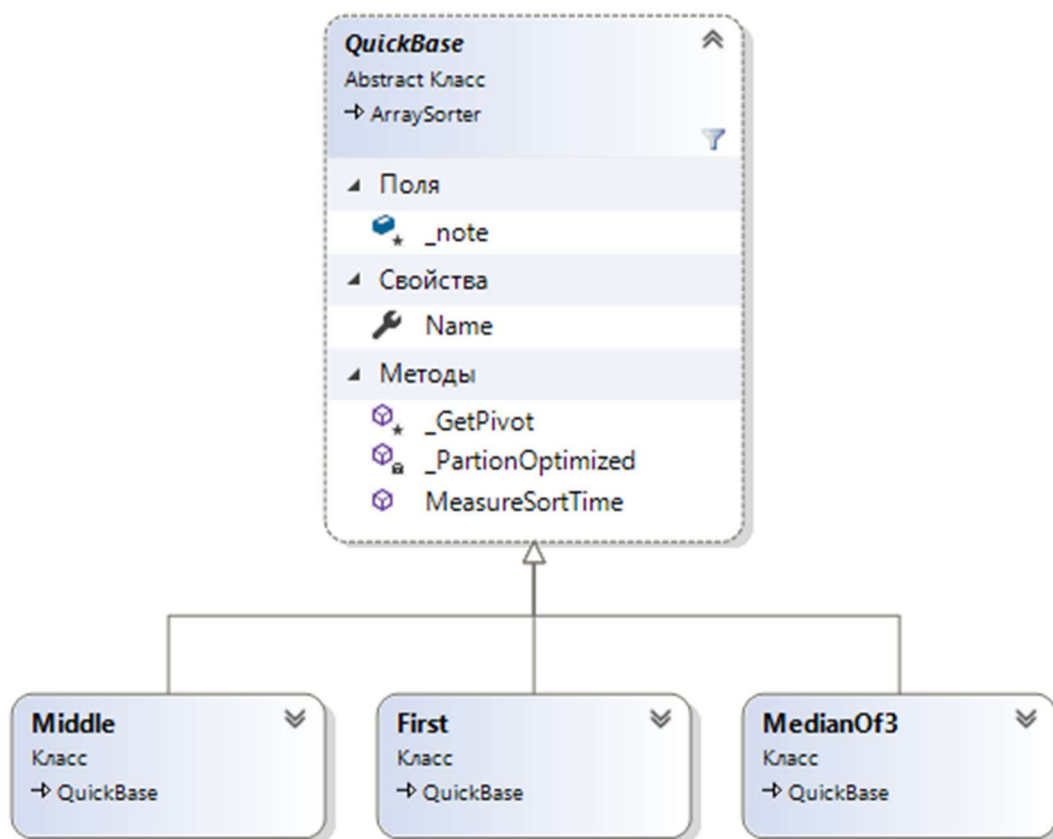


Рисунок 22 – Диаграмма классов проекта «QuickSort»

Плагин «shell.dll» содержит в себе реализацию сортировки Шелла [21] с шагом в четырех вариантах: Шелла (класс «StepBy.Shell»), Вирта (класс «StepBy.Wirth»), Кнута (класс «StepBy.Knuth») и Седжвика (класс «StepBy.Sedgwick»). Библиотека является результатом компиляции проекта «ShellSort», диаграмма классов которого отображена на рисунке 23.

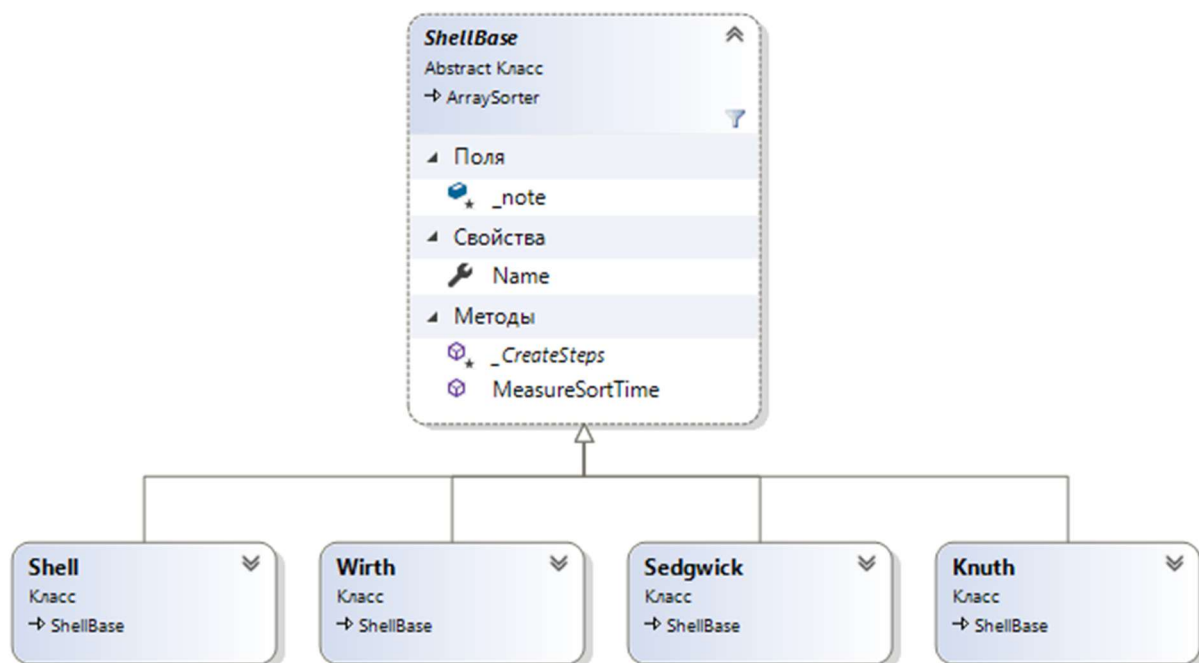


Рисунок 23 – Диаграмма классов проекта «ShellSort»

Плагин «bubble.dll» содержит в себе реализацию сортировки пузырьком [22] в двух вариантах: стандартную (класс «BubbleSort») и с отслеживанием упорядоченности после каждого прохода (класс «BubbleSortDetectsIfSorted»). Библиотека является результатом компиляции проекта «BubbleSort».

Плагин «insert.dll» содержит в себе реализацию сортировки вставками [23] в трех вариантах: стандартную (класс «InsertionSort»), с добавлением фиктивного элемента (класс «InsertionSortEnclosesFake») и с предварительным определением минимального элемента (класс «IntersectionSortFindsMinimum»). Библиотека является результатом компиляции проекта «InsertionSort».

Плагин «linear.dll» содержит в себе реализацию линейной сортировки [24] с предварительным определением максимального элемента (класс «LinearSortUniversal»). Библиотека является результатом компиляции проекта «LinearSort».

Плагин «heapsort.dll» содержит в себе реализацию пирамидальной сортировки [25] (класс «HeapSort»). Библиотека является результатом компиляции проекта «HeapSort».

Плагин «selsort.dll» содержит в себе реализацию сортировки прямым выбором [26] (класс «SelectionSort»). Библиотека является результатом компиляции проекта «SelectionSort».

Итого, стандартные плагины предоставляют возможность по исследованию пятнадцати алгоритмов сортировки.

## 6 Контроль качества разработанной программы

### 6.1 Модульное тестирование

Модульное тестирование – процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы, наборы из одного или более программных модулей вместе с соответствующими управляющими данными, процедурами использования и обработки [27]. Отличительной особенностью таких тестов является проверка модулей системы по отдельности. Обычно модульные тесты применяются для контроля качества наименьшей логики приложения, например, класса или – чаще всего – метода. Для достижения максимальной эффективности модульного тестирования необходимо постараться как можно полнее изолировать содержание теста от внешней логики.

Для осуществления модульного тестирования будет использована библиотека « NUnit » [28].

Было произведено модульное тестирование всех плагинов. Результаты тестирования отображены на рисунке 24.

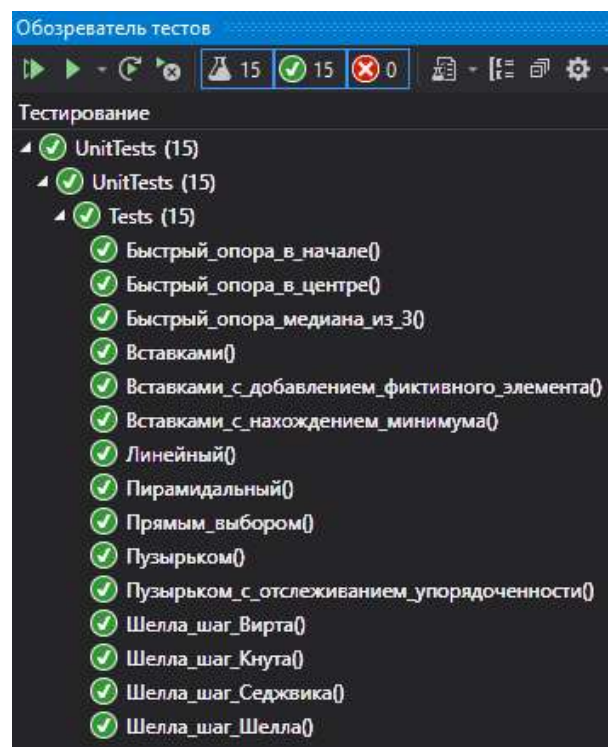


Рисунок 24 – Модульное тестирование

Как видно из рисунка, все тесты были пройдены. Каждый из тестов представляет из себя проверку корректности алгоритма сортировки с соответствующим названием (названия тестов приведены на русском языке для большей наглядности).

Для тестирования генерируется случайный массив, который сортируется выбранным методом сортировки. Тест считается выполненным, если результат сортировки совпадает с результатом стандартной функции упорядочивания последовательностей «OrderBy» [29] (иными словами, если сортировщик действительно возвращает отсортированный входной массив). Общая часть всех тестов вынесена за пределы конкретных тестов. Общий код расположен в листинге 1.

Листинг 1 – Листинг общей части кода модульного тестирования

```
int length = 100;
int max = 1000;
bool Same(int[] a, int[] b)
{
    if (a.Length != b.Length)
    {
        return false;
    }
    for (int i = 0; i < a.Length; i++)
    {
        if (a[i] != b[i])
        {
            return false;
        }
    }
    return true;
}
void Test(ArraySorter sorter)
{
    var sample = new SortingExtensionAPI.ArrayGenerators.TotallyRandom()
        .Generate(length, max);
    var @true = sample.OrderBy(x => x).ToArray();
    sorter.MeasureSortTime(sample);
    var result = Same(sample, @true);
    Assert.IsTrue(result);
}
```

Каждый модульный тест сводится к созданию экземпляра нужного класса сортировщика и вызову метода «Test». Код модульных тестов расположен в листинге 2.

## Листинг 2 – Листинг модульных тестов

```
[TestCase]public void Пузырьком(){
    Test(new Bubble.BubbleSort());
}
[TestCase]public void Пузырьком_с_отслеживанием_упорядоченности(){
    Test(new Bubble.BubbleSortDetectsIfSorted());
}
[TestCase]public void Линейный(){
    Test(new LinearSort.LinearSortUniversal());
}
[TestCase]public void Быстрый_опора_в_начале(){
    Test(new Quick.PivotIs.First());
}
[TestCase]public void Быстрый_опора_в_центре(){
    Test(new Quick.PivotIs.Middle());
}
[TestCase]public void Быстрый_опора_медиана_из_3(){
    Test(new Quick.PivotIs.MedianOf3());
}
[TestCase]public void Шелла_шаг_Шелла(){
    Test(new Shell.StepBy.Shell());
}
[TestCase]public void Шелла_шаг_Вирта(){
    Test(new Shell.StepBy.Wirth());
}
[TestCase]public void Шелла_шаг_Кнута(){
    Test(new Shell.StepBy.Knuth());
}
[TestCase]public void Шелла_шаг_Седжвика(){
    Test(new Shell.StepBy.Sedgwick());
}
[TestCase]public void Пирамидальный(){
    Test(new HeapSort.HeapSort());
}
[TestCase]public void Прямым_выбором(){
    Test(new SelectionSort.SelectionSort());
}
[TestCase]public void Вставками(){
    Test(new InsertionSort.InsertionSort());
}
[TestCase]public void Вставками_с_добавлением_фиктивного_элемента(){
    Test(new InsertionSort.InsertionSortEnclosesFake());
}
[TestCase]public void Вставками_с_нахождением_минимума(){
    Test(new InsertionSort.InsertionSortFindsMinimum());
}
```

Произведенное модульное тестирование подтвердило корректность плагинов, которые будут поставляться в стандартном пакете приложения.

## 6.2 Структурное тестирование

Структурное тестирование основывается на детальном изучении логики программы и подборе тестов, обеспечивающих максимально возможное число проверяемых операторов, логических ветвлений и условий [30]. При помощи

данного метода будет протестирован прецедент запуска процесса исследования.

Блок-схема, на которой будет основано структурное тестирование указанного прецедента, изображена на рисунке 25.

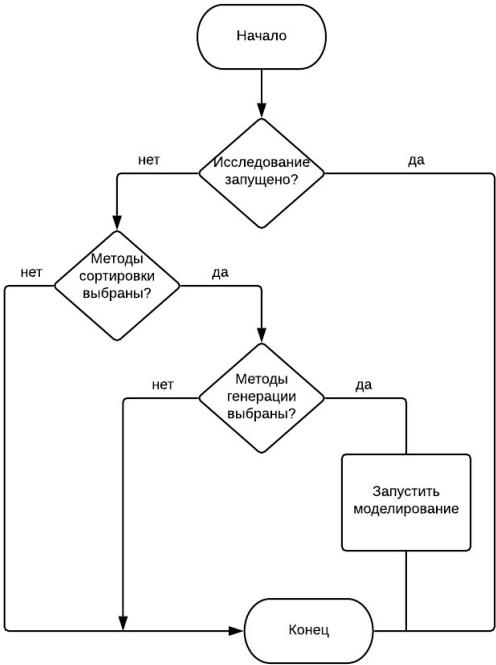


Рисунок 25 – Блок-схема для структурного тестирования

По данной блок-схеме был построен управляющий граф (рисунок 26).

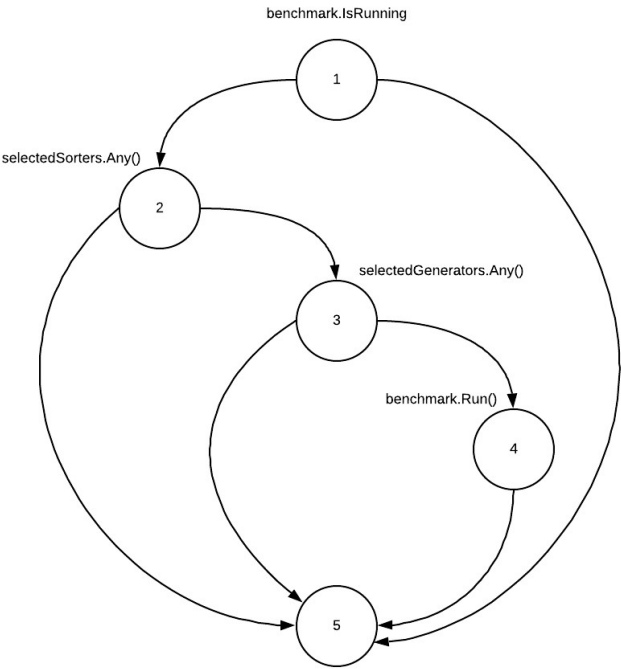


Рисунок 26 – Управляющий граф для структурного тестирования

Первый этап структурного тестирования – тестирование методом покрытия операторов. Тесты, составляемые для такого метода, в совокупности подразумевают покрытие всех операторов; иными словами, каждый оператор должен быть выполнен, по крайней мере, один раз. Список тестов методом покрытия операторов расположен в таблице 10.

Таблица 10 – Таблица покрытия операторов

| Операторы | Тест                                                                                                                           | Фактический результат                                           | Оценка  |
|-----------|--------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------|---------|
| 4, 5      | Исследование не запущено, выбран хотя бы один метод сортировки, выбран хотя бы один метод генерации, моделирование запускается | Запустилось исследование                                        | Успешно |
| 5         | Моделирование уже запущено                                                                                                     | Ничего не произошло (кнопка запуска исследования заблокирована) | Успешно |

Следующий этап – тестирование методом покрытия условий. В этом случае составляются тесты, достаточные для того, чтобы все возможные результаты каждого условия выполнялись, по крайней мере, один раз. Для проверки каждого условия было составлено четыре теста (таблица 11).

Таблица 11 – Таблица покрытия условий

| Условия                  | Тест                                                                                                   | Фактический результат    | Оценка  |
|--------------------------|--------------------------------------------------------------------------------------------------------|--------------------------|---------|
| 1 – да                   | Моделирование уже запущено                                                                             | Ничего не произошло      | Успешно |
| 1 – нет, 2 – нет         | Моделирование не запущено, не выбрано ни одного метода сортировки                                      | Ничего не произошло      | Успешно |
| 1 – нет, 2 – да, 3 – нет | Моделирование не запущено, выбран хотя бы один метод сортировки, не выбрано ни одного метода генерации | Ничего не произошло      | Успешно |
| 1 – нет, 2 – да, 3 – да  | Моделирование не запущено, выбран хотя бы один метод сортировки, выбран хотя бы один метод генерации   | Запустилось исследование | Успешно |

Завершающий этап структурного тестирования – тестирование методом покрытия маршрутов, который подразумевает, что должны быть составлены тесты, которые покрывают все возможные маршруты. Для проверки всех маршрутов было составлено четыре теста (таблица 12).

Таблица 12 – Таблица покрытия маршрутов

| Маршрут       | Тест                                                                                                   | Фактический результат    | Оценка  |
|---------------|--------------------------------------------------------------------------------------------------------|--------------------------|---------|
| 1, 5          | Моделирование уже запущено                                                                             | Ничего не произошло      | Успешно |
| 1, 2, 5       | Моделирование не запущено, не выбрано ни одного метода сортировки                                      | Ничего не произошло      | Успешно |
| 1, 2, 3, 5    | Моделирование не запущено, выбран хотя бы один метод сортировки, не выбрано ни одного метода генерации | Ничего не произошло      | Успешно |
| 1, 2, 3, 4, 5 | Моделирование не запущено, выбран хотя бы один метод сортировки, выбран хотя бы один метод генерации   | Запустилось исследование | Успешно |

Структурное тестирование прошло успешно, что отражает корректность запуска процесса исследования.

### 6.3 Вычисление метрик кода

Определенный код программы можно оценить с помощью специальных метрик кода, вычисленное значение которых позволяет делать те или иные выводы о написанном коде [31].

Для оценки качества кода, который обеспечивает циклическое выполнение задач на потоке из пула потоков (листинг 3), будут определены и проанализированы некоторые метрики.

Одной из самых простых метрик кода является количество строк в исходном коде программы, функции и т.д. Вычисленный параметр позволяет оценить трудоемкость написанного кода. Выделяют все непустые строки файла на выделенном отрезке (физические строки), строки, которые тем или

иным образом определяют логику программы (логические строки) и строки-комментарии.

Листинг 3 – Листинг кода выполнение задач внутри каждого потока из пула

```
var thread = Thread.CurrentThread;

// Пока не попросили закончить работу, планировать задачи
while (_executionAllowedEvent.WaitOne(0))
{
    // Если поток не приостановлен, то выполнять планирование задач
    _cycleEnabledEvents[thread].WaitOne();

    // Если задача есть, то попробовать выполнить ее
    if (TryDequeue(out var t))
    {
        TryExecuteTask(t);
    }
    // Если задач нет, то поставить поток на паузу
    else
    {
        _cycleEnabledEvents[thread].Reset();
    }
}
```

В представленном коде присутствуют:

- 17 физических строк кода;
- 7 логических строк кода;
- 4 строк комментариев.

На основе подсчитанного количества строк кода сделан вывод, что представленный код не требует большого вложения труда в его написание.

Индекс сопровождаемости *MI* (Maintainability Index) [32] позволяет оценить относительную сложность сопровождаемости кода. Данный показатель принимает значение от нуля до ста; чем вычисленное значение меньше, тем проще поддержка. Основу этой метрики составляет четыре измеряемых характеристики программы:

- число уникальных операторов программы, включая символы-разделители, имена процедур и знаки операций (словарь операторов  $n_1$ );
- число уникальных операндов программы (словарь операндов  $n_2$ );
- общее число операторов в программе  $N_1$ ;
- общее число операндов в программе  $N_2$ .

Таблица 13 содержит словарь операторов вместе с частотой их вхождения. Аналогичный словарь для операндов расположен в таблице 14.

Указанные параметры необходимы для вычисления индекса сопровождаемости кода по формуле:

$$MI = MAX(0, (171 - 5.2 \cdot \ln(HV) - 0.23 \cdot CC - 16.2 \cdot \ln(LoC))) \cdot \frac{100}{171}.$$

Таблица 13 – Словарь операторов

| Операторы        | Число вхождений |
|------------------|-----------------|
| =                | 1               |
| .                | 4               |
| вызов функции () | 4               |
| if               | 1               |
| else             | 1               |
| []               | 1               |
| while            | 1               |
| $n_1 = 7$        | $N_1 = 13$      |

Таблица 14 – Словарь операндов

| Операнды               | Число вхождений |
|------------------------|-----------------|
| Thread                 | 1               |
| thread                 | 3               |
| CurrentThread          | 1               |
| _executionAllowedEvent | 1               |
| WaitOne                | 2               |
| _cycleEnabledEvents    | 1               |
| 0                      | 1               |
| TryDequeue             | 1               |
| t                      | 2               |
| TryExecuteTask         | 1               |
| Reset                  | 1               |
| $n_2 = 11$             | $N_2 = 15$      |

Необходимо рассчитать длину программы  $N$  по формуле:

$$N = N_1 + N_2 = 13 + 15 = 28.$$

Затем вычислить словарь программы  $n$  по формуле:

$$n = n_1 + n_2 = 7 + 11 = 18.$$

Также нужно рассчитать вычислительную сложность  $HV$  по формуле:

$$HV = N \cdot \log_2(n) \approx 28 \cdot 4.17 \approx 116.76.$$

*CC* (Cyclomatic Complexity) – структурная сложность кода, т.е. количество различных ветвей в коде. В данном случае *CC* равен двум.

*LoC* (Lines of Code) – количество строк кода. В данном случае *LoC* равен семнадцати.

Подставив найденные значения, получаем:

$$MI = \text{MAX}(0, (171 - 5.2 \cdot \ln(116.76) - 0.23 \cdot 2 - 16.2 \cdot \ln(17))) \cdot \frac{100}{171} \approx 58.$$

Вычисленное значение комплексного показателя качества кода равно пятидесяти восьми, то есть лежит в допустимых границах. Из этого сделан вывод, что сложность поддержки представленного кода является средней – алгоритм имеет приемлемый индекс сопровождаемости.

## **7 Применение разработанной программы для исследования временных характеристик алгоритмов сортировки**

Созданное приложение позволяет количественно сравнивать временные характеристики алгоритмов сортировки массивов. Было принято решение применить программу для получения оценок временных характеристик сортировки Шелла (с шагом Вирта, Кнута и Шелла) и быстрой сортировки (с опорным элементом в начале, в середине и медиане и первого, среднего и последнего элементов). Ниже следует сравнительный анализ полученных данных. Все измерения производились на массивах длиной от 10 элементов до 500 тыс. элементов включительно. Скриншоты исследования расположены в приложении Б.

При сортировке упорядоченных как в прямом, так и в обратном порядке массивов быстрый метод с опорным элементом в первой позиции показал невероятно плохие результаты. По графику видно, что функция имеет сильно возрастающую первую производную, т.е. на таких массивах данный алгоритм имеет ярко выраженный экспоненциальный характер (рисунок Б.1).

На этих же массивах вариации быстрой сортировки с опорой посередине и на медиане из первого, среднего, и последнего элементов проигрывают всем вариантам сортировки Шелла, хотя значительно менее выражено, чем вариация с опорой в первой позиции (рисунок Б.2).

Сортировка Шелла с шагом Кнута стабильно выигрывает у сортировки с шагом Вирта, порой с заметным отрывом на полностью неупорядоченных, частично (50%) неупорядоченных и отсортированных как в прямом, так и в обратном порядке массивах (рисунок Б.3).

Сортировка с шагом Шелла на неупорядоченных массивах выдает функцию колебательного характера с заметной на графике амплитудой, которая растет с увеличением длины массива (в окрестности 10 тыс. достигает одной десятой миллисекунды; в окрестности 500 тыс. – одной миллисекунды). Как правило эта вариация хуже шага Кнута, но лучше шага Вирта; хотя на неупорядоченных массивах в определенных точках она способна в

незначительной степени нарушать эти правила из-за большой амплитуды колебаний (рисунок Б.4).

Замеры времени на случайно сгенерированных массивах показали, что все представленные вариации быстрой сортировки выдают в целом параллельные графики. Сортировки с опорой в первой и в средней позиции выдают практически идентичные графики; в медиане – практически всегда уступает остальным вариациям (на сотую миллисекунды в окрестности 10 тыс.; на одну миллисекунду в окрестности 200 тыс.; на две миллисекунды в окрестности 500 тыс.). (рисунок Б.5).

Сравнение быстрой сортировки и сортировки Шелла (в целом) было проведено на примере сортировки Шелла с шагом Кнута (так как шаг Шелла выдаёт нестабильную колебательную функцию, а шаг Вирта попросту проигрывает в скорости) и быстрой сортировки с опорой в середине (так как в среднем этот метод быстрее, чем с опорой в медиане, и в то же время «не боится» отсортированных массивов, как это делает метод с опорой в начале).

Замечательно, что для неупорядоченных массивов сначала по скорости выигрывает сортировка Шелла, затем наступает период, когда нельзя точно определить лучший метод, затем быстрая сортировка начинает стабильно выигрывать. Более точное сравнение представлено в таблице 15.

Таблица 15 – Сравнение быстрой сортировки и сортировки Шелла

| Неупорядоченность, % | Сортировка Шелла стабильно выигрывает до | Быстрая сортировка стабильно выигрывает после | Рис. |
|----------------------|------------------------------------------|-----------------------------------------------|------|
| 100                  | 17000                                    | 27000                                         | Б.6  |
| 1                    | 3800                                     | 5900                                          | Б.7  |
| 50                   | 1500                                     | 3000                                          | Б.8  |
| 90                   | 10100                                    | 12900                                         | Б.9  |
| 75                   | 3000                                     | 9800                                          | Б.10 |
| 0                    | Всегда                                   | Никогда                                       | Б.11 |
| 10                   | 1400                                     | 1400                                          | Б.12 |
| Случайная генерация  | 10000                                    | 27000                                         | Б.13 |

Таким образом, с помощью созданного приложения был произведен сравнительный анализ сортировки Шелла и быстрой сортировки.

## Заключение

В ходе выполнения данной работы были осуществлены следующие задачи:

- сформулирована поставленная задача;
- спланирована разработка программной системы;
- оценен бюджет, необходимый фонд рабочего времени и длительность разработки;
- осуществлен анализ требований;
- произведено проектирование программной системы;
- реализована и описана спроектированная программная система;
- созданная программная система протестирована;
- с помощью созданной программной системы были получены и проанализированы оценки временных характеристик алгоритмов быстрой сортировки и сортировки Шелла.

Из того, что тестирование было произведено успешно, следует корректность работы созданной программной системы.

### Список использованных источников

1. Intuit // Язык UML 2 в анализе и проектировании программных систем и бизнес-процессов [Электронный ресурс]. – Режим доступа: <https://www.intuit.ru/studies/courses/480/336/info> (дата обращения: 20.04.2020).
2. Самуйлов С.В., Фрунзе П.М. Методика анализа алгоритмов обработки данных. В сборнике: Новые информационные технологии и системы. Материалы XV Международной научно-технической конференции, посвященной 75-летию Пензенского государственного университета. 2018. С. 221-222.
3. Microsoft // Microsoft Project [Электронный ресурс]. – Режим доступа: <https://www.microsoft.com/ru-ru/microsoft-365/project/project-management-software> (дата обращения: 20.04.2020).
4. Atlassian Agile Coach // Что такое диаграмма Ганта? [Электронный ресурс]. – Режим доступа: <https://www.atlassian.com/ru/agile/project-management/gantt-chart> (дата обращения: 21.04.2020).
5. Википедия // Алгоритм сортировки [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Алгоритм\\_сортировки](https://ru.wikipedia.org/wiki/Алгоритм_сортировки) (дата обращения: 17.05.2020).
6. Википедия // Тест производительности [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Тест\\_производительности](https://ru.wikipedia.org/wiki/Тест_производительности) (дата обращения: 17.05.2020).
7. Викисловарь // Бенчмарк [Электронный ресурс]. – Режим доступа: <https://ru.wiktionary.org/wiki/бенчмарк> (дата обращения: 17.05.2020).
8. Аллея Науки // Самуйлов С.В., Фрунзе П.М. Подход к анализу алгоритмов обработки данных на примере алгоритмов внутренней сортировки [Электронный ресурс]. – Режим доступа: [https://alley-science.ru/domains\\_data/files/14December18/PODHOD%20K%20ANALIZU%20ALGORITMOV%20OBRABOTKI%20DANNYH%20NA%20PRIMERE%20ALGORITMOV%20VNUTRENNEY%20SORTIROVKI.pdf](https://alley-science.ru/domains_data/files/14December18/PODHOD%20K%20ANALIZU%20ALGORITMOV%20OBRABOTKI%20DANNYH%20NA%20PRIMERE%20ALGORITMOV%20VNUTRENNEY%20SORTIROVKI.pdf) (дата обращения: 22.04.2020).

9. 4. Github // arisath // Benchmarking Sorting Algorithms [Электронный ресурс]. – Режим доступа: <https://github.com/arisath/Benchmarking-Sorting-Algorithms> (дата обращения: 20.04.2020).

10. Github // karakanb // sorting benchmark [Электронный ресурс]. – Режим доступа: <https://github.com/karakanb/sorting-benchmark> (дата обращения: 20.04.2020).

11. Google Drive // IntSort (с) Слета В и Копов С.exe [Электронный ресурс]. – Режим доступа: <https://drive.google.com/open?id=19kGWs7JuEioSstz9rvdjh6mOzhd-TwhL> (дата обращения: 24.04.2020).

12. Business Analytics // Формирование требований и классификация требований [Электронный ресурс]. – Режим доступа: <https://analytics.infozone.pro/formation-requirements-and-classification-requirements/> (дата обращения: 17.05.2020).

13. Microsoft // C# [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/> (дата обращения: 24.05.2020).

14. Microsoft // .NET Framework [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/> (дата обращения: 24.05.2020).

15. Microsoft // Что такое dll? [Электронный ресурс]. – Режим доступа: <https://support.microsoft.com/ru-ru/help/815065/what-is-a-dll> (дата обращения: 24.05.2020).

16. Lucidchart // Приложение для построения диаграмм [Электронный ресурс]. – Режим доступа: <https://app.lucidchart.com/> (дата обращения: 01.06.2020).

17. Microsoft // Массивы в C# [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/csharp/tour-of-csharp/arrays> (дата обращения: 24.05.2020).

18. Википедия // Объектно-ориентированное программирование [Электронный ресурс]. – Режим доступа:

[https://ru.wikipedia.org/wiki/Объектно-ориентированное\\_программирование](https://ru.wikipedia.org/wiki/Объектно-ориентированное_программирование)  
(дата обращения: 24.05.2020).

19. RSDN // Эффективная многопоточность. Организация пула потоков [Электронный ресурс]. – Режим доступа: <https://rdsn.org/article/baseserv/threadpool.xml> (дата обращения: 17.05.2020).

20. Algotist // Быстрая сортировка [Электронный ресурс]. – Режим доступа: [http://algotist.ru/sort/quick\\_sort.php](http://algotist.ru/sort/quick_sort.php) (дата обращения: 17.05.2020).

21. Algotist // Сортировка Шелла [Электронный ресурс]. – Режим доступа: [http://algotist.ru/sort/shell\\_sort.php](http://algotist.ru/sort/shell_sort.php) (дата обращения: 17.05.2020).

22. Algotist // Сортировка пузырьком [Электронный ресурс]. – Режим доступа: [http://algotist.ru/sort/bubble\\_sort.php](http://algotist.ru/sort/bubble_sort.php) (дата обращения: 17.05.2020).

23. Algotist // Сортировка вставками [Электронный ресурс]. – Режим доступа: [http://algotist.ru/sort/insert\\_sort.php](http://algotist.ru/sort/insert_sort.php) (дата обращения: 17.05.2020).

24. Википедия // Сортировка подсчетом [Электронный ресурс]. – Режим доступа: [https://ru.wikipedia.org/wiki/Сортировка\\_подсчётом](https://ru.wikipedia.org/wiki/Сортировка_подсчётом) (дата обращения: 17.05.2020).

25. Algotist // Пирамидальная сортировка [Электронный ресурс]. – Режим доступа: [http://algotist.ru/sort/pyramid\\_sort.php](http://algotist.ru/sort/pyramid_sort.php) (дата обращения: 17.05.2020).

26. Algotist // Сортировка выбором [Электронный ресурс]. – Режим доступа: [http://algotist.ru/sort/select\\_sort.php](http://algotist.ru/sort/select_sort.php) (дата обращения: 17.05.2020).

27. Студопедия // Модульное тестирование [Электронный ресурс]. – Режим доступа: [https://studopedia.ru/2\\_118908\\_modulnoe-testirovanie.html](https://studopedia.ru/2_118908_modulnoe-testirovanie.html) (дата обращения: 20.05.2020).

28. NUnit // Расширение для модульного тестирования [Электронный ресурс]. – Режим доступа: <https://nunit.org/> (дата обращения: 20.05.2020).

29. Microsoft // Метод Enumerable.OrderBy [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/dotnet/api/system.linq.enumerable.orderby?view=netcore-3.1> (дата обращения: 25.05.2020).

30. Студопедия // Структурное тестирование [Электронный ресурс]. – Режим доступа: [https://studopedia.ru/3\\_80803\\_strukturnoe-testirovanie.html](https://studopedia.ru/3_80803_strukturnoe-testirovanie.html) (дата обращения: 20.05.2020).
31. Microsoft // Метрики кода [Электронный ресурс]. – Режим доступа: <https://docs.microsoft.com/ru-ru/visualstudio/code-quality/code-metrics-values?view=vs-2019> (дата обращения: 29.05.2020).
32. Хабр // Программный код и его метрики [Электронный ресурс]. – Режим доступа: <https://habr.com/ru/company/intel/blog/106082/> (дата обращения: 29.05.2020).

Приложение А

ИСХОДНЫЙ КОД РАЗРАБОТАННОЙ ПРОГРАММЫ

(рекомендуемое)

## Проект «Sorting»

### Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using SortingExtensionAPI;
using System.IO;
using System.Reflection;
using static System.IO.Directory;

namespace Sorting
{
    static class Program
    {
        private static Random _randomizer = new Random();

        public static readonly List<ArraySorter> Methods = new List<ArraySorter>();
        public static readonly List<ArrayGenerator> Generators = new List<ArrayGenerator>();

        [STAThread]
        static void Main()
        {
            // Формат
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }
    }
}
```

### TimeBenchmark.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Collections.Concurrent;
using System.Diagnostics;
using System.Threading;
using System.Runtime;

namespace Sorting
{
    class TimeBenchmark
    {
        private readonly Parameters _parameters;

        public class Parameters
        {
            private int minLength = 10;
            private int maxLength;
            private int stepsCount;
            private double maxRangeMultiplier = 2.0;
            private int[] generatorsNumbers;
            private int[] sortersNumbers;
            private int generationsPerPoint = 1;
            private int countTimes = 1;

            public int CountTimes { get => countTimes; set => countTimes = value > 0 ? value : throw new
ArgumentOutOfRangeException(); }
            public double MaxRangeMultiplier { get => maxRangeMultiplier; set => maxRangeMultiplier =
value > 0 ? value : throw new ArgumentOutOfRangeException(); }
            public int GenerationsPerPoint { get => generationsPerPoint; set => generationsPerPoint =
value > 0 ? value : throw new ArgumentOutOfRangeException(); }
            public IEnumerable<int> GeneratorsNumbers
            {
                get => generatorsNumbers;
                set
                {
                    bool ok(int no)
                    {
                        try
                        {
                            Program.Generators[no].Generate(2, 10);
                            return true;
                        }
                        catch { }
                    }
                }
            }
        }
    }
}
```

```

        }
        catch (NotSupportedException) { return false; }
    }
    generatorsNumbers = value.Where(no => ok(no)).ToArray();
}
}
public IEnumerable<int> SortersNumbers
{
    get => sortersNumbers;
    set
    {
        bool ok(int no)
        {
            try
            {
                Program.Methods[no].MeasureSortTime(new int[2]);
                return true;
            }
            catch (NotSupportedException) { return false; }
        }
        sortersNumbers = value.Where(no => ok(no)).ToArray();
    }
}
public int MinLength { get => minLength; set => minLength = value > 0 ? value : 10; }
public int MaxLength { get => maxLength; set => maxLength = value; }
public int StepsCount { get => stepsCount; set => stepsCount = value; }
}

public TimeBenchmark(parameters parameters)
{
    _parameters = parameters;
}

TaskFactory _taskFactory;

public class Result
{
    public int GeneratorNumber { get; set; }
    public int SorterNumber { get; set; }
    public int ArrayLength { get; set; }
    public TimeSpan ElapsedTime { get; set; }
}

Queue<Result> _results;
object _queueLocker = new object();

public Result[] GetCurrentResults()
{
    lock (_queueLocker)
    {
        var currentLength = _results.Count;
        var currentResults = new Result[currentLength];
        for (var i = 0; i < currentLength; i++)
        {
            currentResults[i] = _results.Dequeue();
        }
        return currentResults;
    }
}

public static bool Running { get; private set; } = false;

public async Task StartMeasuringAsync(int coresCount, ProcessPriorityClass RegularProcessPriority,
ProcessPriorityClass CalculationProcessPriority, Action BeforeMeasuringIfOK)
{
    if (!Running)
    {
        Canceled = false;
        Running = true;
        _results = new Queue<Result>();

        BeforeMeasuringIfOK();

        Process.GetCurrentProcess().PriorityClass = CalculationProcessPriority;
        var schd = new MaximumPerformanceTaskScheduler(coresCount, ThreadPriority.Highest);
        var oldGCConcurrencyMode = GCSettings.LatencyMode;
        GCSettings.LatencyMode = GCConcurrencyMode.SustainedLowLatency;
        //GC.TryStartNoGCRegion(200L << 20);
        try
    
```

```

        {
            _taskFactory = new TaskFactory(schd);

            await Task.Run((Action)Measure);
        }
        finally
        {
            //GC.EndNoGCRegion();
            if (schd != null) schd.Dispose();
            GCSettings.LatencyMode = oldGCLatencyMode;
            Process.GetCurrentProcess().PriorityClass = RegularProcessPriority;
            Running = false;
        }
    }
    else
    {
        throw new InvalidOperationException("Измерения уже запущены");
    }
}

public void Cancel()
{
    if (Running)
    {
        Canceled = true;
    }
}

public bool Canceled { get; private set; }

private void Measure()
{
    var tasks = new Queue<Task>();

    int step = (_parameters.MaxLength - _parameters.MinLength) > 0
        ? (_parameters.MaxLength - _parameters.MinLength) / _parameters.StepsCount
        : 1;

    Task.WaitAll(_taskFactory.StartNew(delegate
    {
        foreach (var generatorNo in _parameters.GeneratorsNumbers)
        {
            for (var length = _parameters.MinLength; length <= _parameters.MaxLength; length +=
step)
            {
                {
                    var generatingTask = GenerateArraysForPointAsync(generatorNo, length);
                    tasks.Enqueue(generatingTask);
                    foreach (var sorterNo in _parameters.SortersNumbers)
                    {
                        var measuringTask = MeasureForPointAsync(generatingTask, sorterNo,
generatorNo);
                        tasks.Enqueue(measuringTask);
                    }
                }
            }
        }
    }));

    Task.WaitAll(tasks.ToArray());
}

private async Task<int[][]> GenerateArraysForPointAsync(int generatorNo, int length)
{
    var arrays = new int[_parameters.GenerationsPerPoint][];
    await _taskFactory.StartNew(
        delegate
        {
            if (!Canceled)
            {
                for (var i = 0; i < _parameters.GenerationsPerPoint; i++)
                {
                    arrays[i] = Program.Generators[generatorNo].Generate(length, (int)(length *
_parameters.MaxRangeMultiplier));
                }
            }
        }, TaskCreationOptions.PreferFairness);

    return arrays;
}

```

```

        private async Task MeasureForPointAsync(Task<int[][]> generatingTask, int sorterNo, int
generatorNo)
        {
            await generatingTask.ContinueWith(
                t =>
                {
                    if (!Canceled)
                    {
                        var arrays = t.Result;
                        TimeSpan totalTime = TimeSpan.Zero;
                        foreach (var array in arrays)
                        {
                            TimeSpan timePerArrayMin = new TimeSpan(1, 0, 2);
                            for (var recount = 0; recount < _parameters.CountTimes; recount++)
                            {
                                var copy = array.ToArray();

                                var sortTime = Program.Methods[sorterNo].MeasureSortTime(copy);

                                if (sortTime < timePerArrayMin)
                                    timePerArrayMin = sortTime;
                            }
                            totalTime += timePerArrayMin;
                        }
                        var averageTime = TimeSpan.FromTicks(totalTime.Ticks / arrays.Length);
                        var result = new Result()
                        {
                            ElapsedTime = averageTime,
                            GeneratorNumber = generatorNo,
                            SorterNumber = sorterNo,
                            ArrayLength = arrays[0].Length
                        };
                        lock (_queueLocker)
                        {
                            _results.Enqueue(result);
                        }
                    }
                });
        }
    }
}

```

#### MaximumPerformanceTaskScheduler.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Threading;
using System.Collections.Concurrent;

namespace Sorting
{
    class MaximumPerformanceTaskScheduler : TaskScheduler, IDisposable
    {
        public override int MaximumConcurrencyLevel => _maximumConcurrencyLevel;

        int _maximumConcurrencyLevel;

        protected override bool TryExecuteTaskInline(Task task, bool taskWasPreviouslyQueued)
        {
            return false;
        }

        protected override void QueueTask(Task task)
        {
            if (task.CreationOptions.HasFlag(TaskCreationOptions.PreferFairness))
            {
                _regularPriorityTasks.Enqueue(task);
            }
            else
            {
                _highPriorityTasks.Enqueue(task);
            }
            foreach (var e in _cycleEnabledEvents.Values)
            {
                e.Set();
            }
        }
    }
}

```

```

protected override IEnumerable<Task> GetScheduledTasks()
{
    throw new NotSupportedException();
}

protected override bool TryDequeue(Task task)
{
    throw new NotSupportedException();
}

protected bool TryDequeue(out Task task)
{
    if (_highPriorityTasks.TryDequeue(out task))
    {
        return true;
    }
    if (_regularPriorityTasks.TryDequeue(out task))
    {
        return true;
    }
    return false;
}

protected ConcurrentQueue<Task> _regularPriorityTasks;
protected ConcurrentQueue<Task> _highPriorityTasks;

protected Thread[] _threads;

public MaximumPerformanceTaskScheduler(int maxCores, ThreadPriority threadPriority)
{
    _threadPriority = threadPriority;
    _maximumConcurrencyLevel = Math.Min(Math.Max(1, maxCores), Environment.ProcessorCount);
    _regularPriorityTasks = new ConcurrentQueue<Task>();
    _highPriorityTasks = new ConcurrentQueue<Task>();
    ConstructThreadPool();
}

ThreadPriority _threadPriority;

protected void ConstructThreadPool()
{
    _threads = new Thread[_maximumConcurrencyLevel];
    _cycleEnabledEvents = new Dictionary<Thread, ManualResetEvent>();
    _executionAllowedEvent = new ManualResetEvent(true);
    for (var i = 0; i < _maximumConcurrencyLevel; i++)
    {
        _threads[i] = new Thread(Schedule);
        var t = _threads[i];
        t.Priority = _threadPriority;
        t.IsBackground = true;

        _cycleEnabledEvents.Add(t, new ManualResetEvent(false));

        t.Start();
    }
}

protected Dictionary<Thread, ManualResetEvent> _cycleEnabledEvents;
protected ManualResetEvent _executionAllowedEvent;

protected void Schedule()
{
    var thread = Thread.CurrentThread;

    // Пока не попросили закончить работу, планировать задачи
    while (_executionAllowedEvent.WaitOne(0))
    {
        // Если поток не приостановлен, то выполнять планирование задач
        _cycleEnabledEvents[thread].WaitOne();

        // Если задача есть, то попробовать выполнить ее
        if (TryDequeue(out var t))
        {
            TryExecuteTask(t);
        }
        // Если задач нет, то поставить поток на паузу
        else
    }
}

```

```

        {
            _cycleEnabledEvents[thread].Reset();
        }
    }
}

public void Dispose()
{
    // Все потоки узнают, что пора заканчивать работу
    _executionAllowedEvent.Reset();

    // Если они были приостановлены из-за отсутствия задач, то запустятся
    foreach (var e in _cycleEnabledEvents.Values)
    {
        e.Set();
    }
}
}
}
}

```

### MainForm.cs

```

using System.Diagnostics;
using System.Threading.Tasks;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.Threading;
using System.Windows.Forms.DataVisualization.Charting;
using SortingExtensionAPI;
using System.Reflection;

namespace Sorting
{
    public partial class MainForm : Form
    {
        public MainForm()
        {
            InitializeComponent();
        }

        void ConnectPlugins()
        {
            foreach (var lib in Directory.GetFiles(Directory.GetCurrentDirectory(), "*.dll",
                SearchOption.TopDirectoryOnly))
            {
                try
                {
                    var assembly = Assembly.LoadFile(lib);
                    foreach (var type in assembly.GetTypes())
                    {
                        if (!type.IsAbstract)
                        {
                            if (typeof(ArraySorter).IsAssignableFrom(type))
                            {
                                var sorter = (ArraySorter)Activator.CreateInstance(type);
                                Program.Methods.Add(sorter);
                            }
                            else if (typeof(ArrayGenerator).IsAssignableFrom(type))
                            {
                                var generator = (ArrayGenerator)Activator.CreateInstance(type);
                                Program.Generators.Add(generator);
                            }
                        }
                    }
                }
                catch (Exception ex)
                {
                    MessageBox.Show(ex.Message, lib, MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        private void MainForm_Load(object sender, EventArgs e)

```

```

{
    ConnectPlugins();

    _regularPriority = Process.GetCurrentProcess().PriorityClass;

    _InitializeMenu();
    _InitializeChart();
    for (var i = 0; i < Program.Methods.Count; i++)
    {
        lvMethods.Items.Add(Program.Methods[i].Name).Checked = true;
        lvMeasuring.Items.Add(Program.Methods[i].Name).BackColor =
ColorSupplier.GetTableRowColor();
    }
    lvMethods.Columns[0].Tag = true;
    if (lvMethods.Items.Count > 0)
    {
        lvMethods.Items[0].Selected = true;
    }
    for (var i = 0; i < Program.Generators.Count; i++)
    {
        lvGenerators.Items.Add(Program.Generators[i].Name).Checked = true;
        lvMeasuring.Columns.Add(Program.Generators[i].Name).TextAlign = HorizontalAlignment.Right;
        foreach (ListViewItem item in lvMeasuring.Items)
        {
            item.SubItems.Add("-");
        }
    }
    lvGenerators.Columns[0].Tag = true;
    if (lvGenerators.Items.Count > 0)
    {
        lvGenerators.Items[0].Selected = true;
    }
    _AdjustDemonstrationColumns();
    _AdjustMeasuringColumns();
    lvDemonstration.Columns[0].Width = 147;
    {
        var legend = Chart.Legends[0];
        for (var methodNo = 0; methodNo < Program.Methods.Count; methodNo++)
        {
            legend.CustomItems.Add(ColorSupplier.GetChartSeriesColor(methodNo),
Program.Methods[methodNo].Name);
        }
    }
    _demoAdjustingTimer.Interval = 500;
    _demoAdjustingTimer.Tick += delegate { _AdjustDemonstrationColumns();
_demoAdjustingTimer.Stop(); };
}

private void _InitializeChart()
{
    _sampleChartSeries.MarkerSize = 7;
    _sampleChartSeries.MarkerStyle = MarkerStyle.None;

    for (var generatorNo = 0; generatorNo < Program.Generators.Count; generatorNo++)
    {
        var area = Chart.ChartAreas.Add("ChartArea" + (Chart.ChartAreas.Count - 1).ToString());
        area.Position.Auto = true;
        var title = Chart.Titles.Add("Title" + Chart.Titles.Count);
        title.DockedToChartArea = area.Name;
        title.IsDockedInsideChartArea = false;
        title.Text = Program.Generators[generatorNo].Name;
        title.Font = new Font("Microsoft Sans Serif", 10.5f);
        for (var methodNo = 0; methodNo < Program.Methods.Count; methodNo++)
        {
            var series = Chart.Series.Add("Series" + Chart.Series.Count);
            series.Legend = "InvisibleLegend";
            series.ChartArea = area.Name;
            series.ChartType = _sampleChartSeries.ChartType;
            series.MarkerStyle = _sampleChartSeries.MarkerStyle;
            series.BorderWidth = _sampleChartSeries.BorderWidth;
            series.MarkerSize = _sampleChartSeries.MarkerSize;
            series.Color = ColorSupplier.GetChartSeriesColor(methodNo);

            _chartSeriesByGeneratorAndMethodIndices.Add(new GeneratorSorterKey(generatorNo,
methodNo), series);
        }
    }
}

```

```

void _InitializeMenu()
{
    // Типы графиков
    var chartTypesInserted = 0;
    ToolStripMenuItem addChartTypeOption(string text, SeriesChartType type, Keys hot)
    {
        var it = new ToolStripMenuItem(text);
        it.Tag = type;
        if (chartTypesInserted == 0)
        {
            it.Checked = true;
        }
        внешнийВидГрафиковToolStripMenuItem.DropDownItems.Insert(chartTypesInserted++, it);
        it.Click += _Handle_ChartTypeChanged;
        it.ShortcutKeys = hot;
        return it;
    }
    addChartTypeOption("Столбцы", SeriesChartType.Column, Keys.Control | Keys.Q);
    addChartTypeOption("Линейная интерполяция", SeriesChartType.Line, Keys.Control |
Keys.L).PerformClick();
    addChartTypeOption("Сплайн-интерполяция", SeriesChartType.Spline, Keys.Control | Keys.S);
    addChartTypeOption("Точки", SeriesChartType.Point, Keys.Control | Keys.D);

    // Ширина линии графиков
    var widthOptionsInserted = 0;
    ToolStripMenuItem addWidthOption(int width, Keys hot)
    {
        var it = new ToolStripMenuItem(width.ToString());
        it.Tag = width;
        it.Click += _Handle_ChartLineWidthChanged;
        it.ShortcutKeys = hot;
        толщинаЛинииToolStripMenuItem.DropDownItems.Insert(widthOptionsInserted++, it);
        return it;
    }
    addWidthOption(1, Keys.Control | Keys.D1);
    addWidthOption(2, Keys.Control | Keys.D2);
    addWidthOption(3, Keys.Control | Keys.D3).PerformClick();
    addWidthOption(4, Keys.Control | Keys.D4);
    addWidthOption(5, Keys.Control | Keys.D5);

    // Режим процесса
    var priorityOptionsInserted = 0;
    ToolStripMenuItem addProcessPriorityOption(string text, ProcessPriorityClass priority)
    {
        var it = new ToolStripMenuItem(text);
        it.Tag = priority;
        it.Click += _Handle_PriorityChanged;

        приоритетПроцессаВоВремяВычисленийToolStripMenuItem.DropDownItems.Insert(priorityOptionsInserted++, it);
        return it;
    }
    addProcessPriorityOption("Средний", ProcessPriorityClass.Normal).PerformClick();
    addProcessPriorityOption("Выше среднего", ProcessPriorityClass.AboveNormal);
    addProcessPriorityOption("Высокий", ProcessPriorityClass.High).PerformClick();
    addProcessPriorityOption("НЕ НАЖИМАЙ! УБЬЁТ!", ProcessPriorityClass.RealTime).Click
        += delegate
        {
            MessageBox.Show(
                "Режим реального времени предоставляет максимально возможную производительность, "
                + "однако во время вычислений не гарантируется работоспособность чего-либо еще, будь то
                графический интерфейс этого приложения (включая кнопку отмены), "
                + "другие программы, диспетчер задач и даже курсор мыши. Не используйте этот режим, если не
                уверены, что вычисления займут разумное время."
                + "\n\nДанный режим разблокирован, только если приложение запущено от имени администратора.
                Иначе Windows будет использовать \"Высокий\" приоритет.",
                "Все будут живы, однако...", MessageBoxButtons.OK, MessageBoxIcon.Warning
            );
        };

    // Количество процессоров, используемых для вычисления
    var procCountOptionInserted = 0;
    ToolStripMenuItem addProcessorsCountOption(int count)
    {
        var it = new ToolStripMenuItem(count.ToString());
        it.Tag = count;
        it.Click += _Handle_ProcessorCountChanged;

        количествоЗадействованныхЛогическихПроцессоровToolStripMenuItem.DropDownItems.Insert(procCountOptionInsert
ed++, it);

```

```

        return it;
    }
    var procCount = Environment.ProcessorCount;
    for (var i = 1; i <= procCount; i++)
    {
        var it = addProcessorsCountOption(i);
        if (i == procCount - 1) it.PerformClick();
    }
}

ProcessPriorityClass _calculationPriority;
ProcessPriorityClass _regularPriority;

void _Handle_ProcessorCountChanged(object sender, EventArgs e)
{
    var it = ((ToolStripMenuItem)(sender));
    if (!it.Checked)
    {
        it.Checked = true;
        _processorCount = (int)(it.Tag);
        foreach (ToolStripMenuItem unchk in
количествоЗадействованныхЛогическихПроцессоровToolStripMenuItem
            .DropDownItems
            .OfType<ToolStripItem>()
            .Where(taken => taken != it))
        {
            unchk.Checked = false;
        }
    }
}

int _processorCount;

void _Handle_PriorityChanged(object sender, EventArgs e)
{
    var it = ((ToolStripMenuItem)(sender));
    if (!it.Checked)
    {
        it.Checked = true;
        _calculationPriority = (ProcessPriorityClass)(it.Tag);
        foreach (ToolStripMenuItem unchk in приоритетПроцессаВоВремяВычисленийToolStripMenuItem
            .DropDownItems
            .OfType<ToolStripItem>()
            .Where(taken => taken != it))
        {
            unchk.Checked = false;
        }
    }
}

void _Handle_ChartLineWidthChanged(object sender, EventArgs e)
{
    var it = ((ToolStripMenuItem)(sender));
    if (!it.Checked)
    {
        var width = Convert.ToInt32(it.Text);
        foreach (ToolStripMenuItem unchk in
толщинаЛинииToolStripMenuItem.DropDownItems.OfType<ToolStripItem>())
        {
            unchk.Checked = false;
        }
        it.Checked = true;
        _sampleChartSeries.BorderWidth = width;
        foreach (var s in Chart.Series)
        {
            s.BorderWidth = width;
        }
    }
}

[Obsolete]
private void _CreateMeasuringThread(/*Queue<MeasuringRequest> requests*/)
{
    /*
    void calculate()
    {
        if (requests.Count > 0)
        {
            float percentagePerElementCalculated = 100.0f / requests.Count;

```

```

float progress = 0.0f;
while (requests.Count > 0 && !_calculationCanceled)
{
    var request = requests.Dequeue();
    var time = Program.Methods[request.MethodNo].MeasureSortTime(request.Array);
    progress += percentagePerElementCalculated;
    this.Invoke(new Action(
        delegate
        {
            lvMeasuring.Items[request.MethodNo].SubItems[request.GeneratorNo + 1].Text
                = string.Format(radMs.Checked ?
                    "{0:0.0000}" : "{1:N0}", time.TotalMilliseconds, time.Ticks);
            progressBar.Value = (int)Math.Floor(progress);
        }
    ));
}
if (!_calculationCanceled)
{
    this.Invoke(new Action(
        delegate
        {
            statusLabel.Text = "Временные характеристики методов определены";
        }
    ));
}
else
{
    while (requests.Count > 0)
    {
        this.Invoke(new Action(
            delegate
            {
                var request = requests.Dequeue();
                lvMeasuring.Items[request.MethodNo].SubItems[request.GeneratorNo + 1].Text =
                    "Отмена";
            }
        ));
    }
    this.Invoke(new Action(
        delegate
        {
            statusLabel.Text = "Процесс измерения был отменен";
        }
    ));
    _calculationCanceled = false;
}

Process.GetCurrentProcess().PriorityClass = _regularPriority;

this.Invoke(new Action(
    delegate
    {
        _Handle_CalculationDone();
    }
));
}

var thread = new Thread(new ThreadStart(calculate));
thread.IsBackground = true;
thread.Priority = ThreadPriority.Highest;
return thread;
*/
}

private System.Windows.Forms.Timer _PrepareFormForCalculation()
{
    btnMeasureCharacteristics.Enabled = false;
    btnSampleSorting.Enabled = false;
    btnDirectInputSort.Enabled = false;
    btnBuildCharts.Enabled = false;
    numPercentage.Enabled = false;
    gbStatus.Visible = true;
    btnCancel.Enabled = true;
    statusLabel.Text = "Производятся вычисления. Это может занять время";
    progressBar.Value = 0;

    return new System.Windows.Forms.Timer();
}

```

```

}

[Obsolete]
private void btnMeasureCharacteristics_Click(object sender, EventArgs e)
{
    /*
    Process.GetCurrentProcess().PriorityClass = _calculationPriority;

    // Подготовка формы
    _PrepareFormForCalculation();

    // Инициализация данных для вычисления и запуск потока
    var length = (int)numElementsForCharacteristicsCount.Value * 1000;
    var requests = new Queue<MeasuringRequest>();
    for (var generatorNo = 0; generatorNo < Program.Generators.Count; generatorNo++)
    {
        if (lvGenerators.Items[generatorNo].Checked)
        {
            var sortable = Program.Generators[generatorNo].Generate(length, length * 2);
            for (var methodNo = 0; methodNo < Program.Methods.Count; methodNo++)
            {
                lvMeasuring.Items[methodNo].SubItems[generatorNo + 1].Text = "-";
                if (lvMethods.Items[methodNo].Checked)
                {
                    if (!_calculationCanceled)
                    {
                        var copy = new int[length];
                        sortable.CopyTo(copy, 0);
                        var request = new MeasuringRequest
                        {
                            Array = copy,
                            MethodNo = methodNo,
                            GeneratorNo = generatorNo
                        };
                        requests.Enqueue(request);
                    }
                    else
                    {
                        Process.GetCurrentProcess().PriorityClass = _regularPriority;
                        Invoke(new Action(delegate
                        {
                            _calculationCanceled = false;
                            _Handle_CalculationDone();
                            statusLabel.Text = "Генерация массивов отменена";
                        }));
                        return;
                    }
                }
            }
        }
    }
    }
    else
    {
        for (var methodNo = 0; methodNo < Program.Methods.Count; methodNo++)
        {
            lvMeasuring.Items[methodNo].SubItems[generatorNo + 1].Text = "-";
        }
    }
    }
    var thread = _CreateMeasuringThread(requests);
    thread.Start();
    */
}

private void _Handle_CalculationDone(string result)
{
    gbStatus.Visible = false;
    btnMeasureCharacteristics.Enabled = true;
    btnSampleSorting.Enabled = true;
    btnBuildCharts.Enabled = true;
    btnDirectInputSort.Enabled = true;
    numPercentage.Enabled = true;

    statusLabel.Text = result;
}

bool _demoCanBeAdjusted = true;

private void _OutputSampleStep(object sender, DemonstrationCallbackEventArgs e)
{

```

```

var stepNo = lvDemonstration.Items.Count;
var item = lvDemonstration.Items.Add(e.DescriptionOfChangeShort);
var backColor = ColorSupplier.GetTableRowColor();
item.BackColor = backColor;
item.UseItemStyleForSubItems = false;
for (var i = 0; i < e.Array.Length; i++)
{
    var sb = lvDemonstration.Items[stepNo].SubItems.Add(e.HiddenElements.Contains(i) ? "" :
e.Array[i].ToString());
    sb.BackColor = backColor;
}

if (e.IndicesGroups != null)
{
    foreach (var g in e.IndicesGroups)
    {
        var color = ColorSupplier.GetDemonstrationGroupColor(g.GroupNumber);
        foreach (var i in g)
        {
            try
            {
                var sb = item.SubItems[i + 1];
                sb.BackColor = color;
            }
            catch (IndexOutOfRangeException) { }
        }
    }
}

if (e.SwappedPairIndices != null)
{
    var sb1 = item.SubItems[e.SwappedPairIndices.Item1 + 1];
    sb1.BackColor = ColorSupplier.GetSwappedColor();
    sb1.Font = new Font(sb1.Font, FontStyle.Bold);

    var sb2 = item.SubItems[e.SwappedPairIndices.Item2 + 1];
    sb2.BackColor = ColorSupplier.GetSwappedColor();
    sb2.Font = new Font(sb2.Font, FontStyle.Bold);
}
}

DiagramDemonstrationRecorder _chartDemoMng = new DiagramDemonstrationRecorder();

private void _SortSample(int[] array)
{
    var length = array.Length;

    lvDemonstration.Items.Clear();
    while (lvDemonstration.Columns.Count > 1)
    {
        lvDemonstration.Columns.RemoveAt(1);
    }

    for (var i = 0; i < length; i++)
    {
        var col = lvDemonstration.Columns.Add(i.ToString());
        col.TextAlign = HorizontalAlignment.Center;
    }

    _AdjustDemonstrationColumns();
    _demoCanBeAdjusted = false;

    var usingDiagram = tabDemo.SelectedIndex == 1;

    if (usingDiagram)
    {
        _chartDemoMng.Clear();
        lbDemoSteps.Items.Clear();
        var a = demoChart.ChartAreas[0];
        a.AxisX.Minimum = -1;
        a.AxisX.Maximum = array.Length;
        a.AxisY.Minimum = 0;
        a.AxisY.Maximum = array.Max(x => x);
    }

    EventHandler<DemonstrationCallbackEventArgs> callback = !usingDiagram
        ? _OutputSampleStep
        : new EventHandler<DemonstrationCallbackEventArgs>((o, e) => {
lbDemoSteps.Items.Add(_chartDemoMng.AddFrame(o, e)); });
}

```

```

var beforeSortArgs = new DemonstrationManager(array).E;
beforeSortArgs.DescriptionOfChangeShort = "Исходный";
callback(this, beforeSortArgs);

_SelectedMethod.DemonstrationCallback += callback;

try
{
    try
    {
        _SelectedMethod.SortArrayDemonstration(array);
    }
    catch (NotSupportedException)
    {
        _SelectedMethod.MeasureSortTime(array);
    }
    finally
    {
        _SelectedMethod.DemonstrationCallback -= callback;
    }

    statusLabel.Text = "Метод \"\" + _SelectedMethod.Name + "\"\": массив отсортирован";

    var afterSortArgs = new DemonstrationManager(array).E;
    afterSortArgs.DescriptionOfChangeShort = "Отсортирован";
    callback(this, afterSortArgs);

    if (usingDiagram)
    {
        lbDemoSteps.SelectedIndex = 0;
        lbDemoSteps.Focus();
    }
}
catch (NotSupportedException)
{
    MessageBox.Show("Метод не реализует ни оптимальный, ни демонстрационный режим");
}
finally
{
    _demoCanBeAdjusted = true;
}
}

private void btnSampleSorting_Click(object sender, EventArgs e)
{
    if (_SelectedMethod != null && _SelectedGenerator != null)
    {
        var length = (int)numElementsForExampleCount.Value;
        var maxValue = (int)numRangeForExample.Value;
        var array = _SelectedGenerator.Generate(length, maxValue);
        _SortSample(array);
    }
    else
    {
        var msg = "Сначала выберите метод сортировки и генерации (щелчок по названию в таблице)";
        statusLabel.Text = msg;
        MessageBox.Show(msg, "Ты делаешь это неправильно...", MessageBoxButtons.OK,
            MessageBoxIcon.Information);
    }
}

private void btnClearInput_Click(object sender, EventArgs e)
{
    txtInput.Clear();
    txtInput.Focus();
}

private void btnDirectInputSort_Click(object sender, EventArgs e)
{
    if (_SelectedMethod != null)
    {
        var inputWords = txtInput.Text.Split(new char[] { ' ' },
            StringSplitOptions.RemoveEmptyEntries);
        var length = inputWords.Count();
        var array = new int[length];
        if (inputWords.Length >= numElementsForExampleCount.Minimum
            && inputWords.Length <= numElementsForExampleCount.Maximum)
        {
            for (var i = 0; i < length; i++)

```

```

        {
            var word = inputWords[i];
            Int32 number;
            var parsed = Int32.TryParse(word, out number);
            if (parsed)
            {
                array[i] = number;
            }
            else
            {
                statusLabel.Text = "Введённое число не удовлетворяет формату Integer32";
                return;
            }
        }
        _SortSample(array);
    }
    else
    {
        statusLabel.Text = "Количество введенных чисел должно не должно не меньше, чем "
            + numElementsForExampleCount.Minimum
            + ", и не больше, чем "
            + numElementsForExampleCount.Maximum;
    }
}
else
{
    var msg = "Сначала выберите метод сортировки (щелчок по названию в таблице)";
    statusLabel.Text = msg;
    MessageBox.Show(msg, "Ты делаешь это неправильно...", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
}

private void txtInput_KeyPress(object sender, KeyPressEventArgs e)
{
    if (!Char.IsDigit(e.KeyChar) && e.KeyChar != ' ' && e.KeyChar != (char)8)
    {
        e.Handled = true;
    }
}

private void btnCopyFromSample_Click(object sender, EventArgs e)
{
    var usingTable = tabDemo.SelectedIndex == 0;
    var text = "";
    if (usingTable && lvDemonstration.Items.Count > 0)
    {
        for (var i = 1; i < lvDemonstration.Items[0].SubItems.Count; i++)
        {
            var subItem = lvDemonstration.Items[0].SubItems[i];
            text += subItem.Text + " ";
        }
    }
    if (!usingTable && demoChart.Series.Count == 1)
    {
        foreach (var p in demoChart.Series.Single().Points)
        {
            text += $"{p.YValues.Single()} ";
        }
    }
    txtInput.Text = text;
}

Series _sampleChartSeries = new Series();

struct GeneratorSorterKey
{
    public int SorterNo;
    public int GeneratorNo;
    public GeneratorSorterKey(int generator, int sorter)
    {
        SorterNo = sorter;
        GeneratorNo = generator;
    }
}

private readonly Dictionary<GeneratorSorterKey, Series> _chartSeriesByGeneratorAndMethodIndices =
    new Dictionary<GeneratorSorterKey, Series>();

```

```

[Obsolete]
private void btnCreateChart_Click(object sender, EventArgs e)
{
    /*
    Process.GetCurrentProcess().PriorityClass = _calculationPriority;

    // Подготовка формы к вычислениям
    _PrepareFormForCalculation();

    // Настройка графиков
    var x_from = numChartLeftBorder.Value > 0 ? (int)numChartLeftBorder.Value * 1000 : 10;
    var x_to = (int)numChartRightBorder.Value * 1000;
    var x_step = (x_to - x_from) / (int)numChartStepsCount.Value;
    foreach (var series in Chart.Series)
    {
        series.Points.Clear();
    }
    var generationsPerPoint = (int)numGenerationsPerPoint.Value;
    var iterationsCount = (int)numChartIterationsCount.Value;

    var coresCount = Environment.ProcessorCount;
    var queues = new Queue<ChartBuildingRequest>(coresCount);
    for (var i = 0; i < coresCount; i++)
    {
        queues[i] = new Queue<ChartBuildingRequest>();
    }

    var checkedMethodsIndices = new HashSet<int>();
    foreach (var i in lvMethods.CheckedIndices)
    {
        checkedMethodsIndices.Add((int)i);
    }

    var checkedGeneratorsIndices = new HashSet<int>();
    foreach (var i in lvGenerators.CheckedIndices)
    {
        checkedGeneratorsIndices.Add((int)i);
    }

    var generatingTasks = new Queue<Task>();

    var core = -1;
    object coreLocker = new object();

    foreach (var generatorNo in checkedGeneratorsIndices)
    {
        var area = Chart.ChartAreas.FindByName("ChartArea" + generatorNo.ToString());
        area.AxisX.Minimum = x_from;
        area.AxisX.Maximum = x_to;
        for (var x = x_from; x <= x_to; x += x_step)
        {
            async Task startGenerationTaskAsync(int currentX, int currentGeneratorNo)
            {
                await Task.Run(delegate
                {
                    var arrays = new int[generationsPerPoint][];
                    for (var generationNo = 0; generationNo < generationsPerPoint; generationNo++)
                    {
                        arrays[generationNo] =
Program.Generators[currentGeneratorNo].Generate(currentX, currentX * 2);
                    }
                    foreach (int methodNo in checkedMethodsIndices)
                    {
                        if (!_calculationCanceled)
                        {
                            lock (coreLocker)
                            {
                                core = (core + 1) % coresCount;

                                var requests = queues[core];
                                requests.Enqueue(new ChartBuildingRequest
                                {
                                    Arrays = arrays,
                                    MethodNo = methodNo,
                                    GeneratorNo = currentGeneratorNo
                                });
                            }
                        }
                    }
                });
            }
        }
    }
}

```

```

        else
        {
            return;
        }
    }
    });
}
var task = startGenerationTaskAsync(x, generatorNo);
generatingTasks.Enqueue(task);
}
}
await Task.WhenAll(generatingTasks);
if (!_calculationCanceled)
{
    _CalculateForChart(queues, iterationsCount, generationsPerPoint);
}
else
{
    _calculationCanceled = false;
    _Handle_CalculationDone();
    statusLabel.Text = "Генерация массивов отменена";
    Process.GetCurrentProcess().PriorityClass = _regularPriority;
}
*/
}

[Obsolete]
private void _CalculateForChart(/*Queue<ChartBuildingRequest>[] queues,*/ int iterationsCount, int
generationsPerPoint)
{
    /*
    object progressLocker = new object();
    object additionLocker = new object();
    object finishLocker = new object();
    object dequeueLocker = new object();

    double progress = 0.0;
    double progressPerElement = 100.0 / queues.Sum(q => q.Count) / iterationsCount /
generationsPerPoint;
    int threadsDone = 0;

    void finishThread()
    {
        lock (finishLocker)
        {
            threadsDone += 1;
            if (threadsDone >= queues.Length)
            {
                if (!_calculationCanceled)
                {
                    this.Invoke(new Action(delegate
                    {
                        statusLabel.Text = "Графики построены";
                    }
                    ));
                }
                else
                {
                    this.Invoke(new Action(delegate
                    {
                        statusLabel.Text = "Операция построения графиков была отменена";
                    }
                    ));
                }
                this.Invoke(new Action(delegate
                {
                    _Handle_CalculationDone();
                }
                ));
                Process.GetCurrentProcess().PriorityClass = _regularPriority;
            }
        }
    }

    for (var queueNo = 0; queueNo < queues.Length; queueNo++)
    {
        var requests = queues[queueNo];

        void calculate()

```

```

{
    if (requests.Count > 0)
    {
        while (requests.Count > 0 && !_calculationCanceled)
        {
            void handleRequest(CharBuildingRequest request)
            {
                var length = request.Arrays[0].Length;
                long time = 0;
                for (var generationNo = 0; generationNo < generationsPerPoint;
generationNo++)
                {
                    for (var i = 0; i < iterationsCount; i++)
                    {
                        var copy = new int[length];
                        request.Arrays[generationNo].CopyTo(copy, 0);

                        // Вычисление времени
                        time +=
Program.Methods[request.MethodNo].MeasureSortTime(copy).Ticks;

                        lock (progressLocker)
                        {
                            progress += progressPerElement;
                            this.Invoke(new Action(delegate
                            {
                                progressBar.Value = (int)Math.Floor(progress);
                            }
                            ));
                        }
                    }
                }
                time /= (iterationsCount * generationsPerPoint);
                lock (additionLocker)
                {
                    this.Invoke(new Action(delegate
                    {
                        var s = _chartSeriesByGeneratorAndMethodIndices[new
SorterGeneratorKey(request.GeneratorNo, request.MethodNo)];
                        var xy = new DataPoint(length, time);
                        var insertionIndex = 0;
                        while (insertionIndex < s.Points.Count &&
s.Points[insertionIndex].XValue < length)
                        {
                            insertionIndex++;
                            if (insertionIndex < s.Points.Count)
                                s.Points.Insert(insertionIndex, xy);
                            else
                                s.Points.Add(xy);
                        }
                    }
                    ));
                }

                async void handleRequestOnNewTask(CharBuildingRequest request)
                {
                    await Task.Run(delegate { handleRequest(request); });
                    finishThread();
                }

                ChartBuildingRequest dequeued;
                lock (dequeueLocker)
                {
                    dequeued = requests.Dequeue();
                }

                var canShareRequest = false;
                lock (finishLocker)
                {
                    if (threadsDone > 0)
                    {
                        canShareRequest = true;
                        threadsDone -= 1;
                    }
                }
                if (!canShareRequest)
                {
                    handleRequest(dequeued);
                }
                else

```

```

        {
            handleRequestOnNewTask(dequeued);
        }
    }
    finishThread();
}

var thread = new Thread(new ThreadStart(calculate));
thread.IsBackground = true;
thread.Priority = ThreadPriority.Highest;
thread.Start();
}
*/
}

readonly int TabIndexDemo = 1;
readonly int TabIndexMeasuring = 2;
readonly int TabIndexChart = 0;

private void MainForm_SizeChanged(object sender, EventArgs e)
{
    //_AdjustDemonstrationColumns();
    //_AdjustMeasuringColumns();
}

private void _AdjustDemonstrationColumns()
{
    if (lvDemonstration.Columns.Count > 1 && _demoCanBeAdjusted && tabControl.SelectedIndex ==
TabIndexDemo)
    {
        var firstColumnWidth = lvDemonstration.Columns[0].Width;
        var widthPerColumn = (lvDemonstration.Width - firstColumnWidth -
SystemInformation.VerticalScrollBarWidth) / (lvDemonstration.Columns.Count - 1);
        for (var i = 1; i < lvDemonstration.Columns.Count; i++)
        {
            var column = lvDemonstration.Columns[i];
            column.Width = widthPerColumn;
        }
    }
}

private void _AdjustMeasuringColumns()
{
    if (tabControl.SelectedIndex == TabIndexMeasuring)
    {
        var widthPerColumn = (lvMeasuring.Width - SystemInformation.VerticalScrollBarWidth) /
lvMeasuring.Columns.Count;
        foreach (ColumnHeader column in lvMeasuring.Columns)
        {
            column.Width = widthPerColumn;
        }
    }
}

private ArraySorter _SelectedMethod
{ get => lvMethods.SelectedIndices.Count > 0 ? Program.Methods[lvMethods.SelectedIndices[0]] :
null; }

private ArrayGenerator _SelectedGenerator
{ get => lvGenerators.SelectedIndices.Count > 0 ?
Program.Generators[lvGenerators.SelectedIndices[0]] : null; }

private void lvGenerators_MouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Right || e.Button == MouseButtons.Left)
    {
        if (lvGenerators.SelectedIndices.Count > 0)
        {
            var index = lvGenerators.SelectedIndices[0];
            if (Program.Generators[index] is
SortingExtensionAPI.ArrayGenerators.PartiallyUnsorted)
            {
                gbPercentage.Visible = true;
                numPercentage.Value = (Program.Generators[lvGenerators.SelectedIndices[0]] as
SortingExtensionAPI.ArrayGenerators.PartiallyUnsorted).Percentage;
                if (e.Button == MouseButtons.Right)
                {

```

```

        numPercentage.Focus();
        numPercentage.Select(0, numPercentage.Text.Length);
    }
    return;
}
}
}
}
gbPercentage.Visible = false;
MessageBox.Show("click");
}

private void gpPercentage_Enter(object sender, EventArgs e)
{
    numPercentage.Value = (Program.Generators[lvGenerators.SelectedIndices[0]] as
SortingExtensionAPI.ArrayGenerators.PartiallyUnsorted).Percentage;
    numPercentage.Focus();
    numPercentage.Select(0, numPercentage.Text.Length);
}

private void btnChartScreenShot_Click(object sender, EventArgs e)
{
    var directory = "chart";
    if (!Directory.Exists(directory))
        Directory.CreateDirectory(directory);
    var fileNo = 0;
    var file = String.Empty;
    do
    {
        file = String.Format("{0}\\Screenshot {1}.png", directory, fileNo++.ToString());
    } while (File.Exists(file));
    Chart.SaveImage(file, ChartImageFormat.Png);
    statusLabel.Text = String.Format("Скриншот сделан и сохранен в папке \"{0}\"",
Path.GetFullPath(directory));
}

private void numChartLeftBorder_ValueChanged(object sender, EventArgs e)
{
    if (numChartLeftBorder.Value >= numChartRightBorder.Value)
    {
        numChartRightBorder.Value = numChartLeftBorder.Value + 1;
    }
}

private void numChartRightBorder_ValueChanged(object sender, EventArgs e)
{
    if (numChartRightBorder.Value <= numChartLeftBorder.Value)
    {
        numChartLeftBorder.Value = numChartRightBorder.Value - 1;
    }
}

private void tabControl_SelectedIndexChanged(object sender, EventArgs e)
{
    if (tabControl.SelectedIndex == TabIndexDemo)
    {
        _AdjustDemonstrationColumns();
    }
    else if (tabControl.SelectedIndex == TabIndexMeasuring)
    {
        _AdjustMeasuringColumns();
    }
}

private void btnCancel_Click(object sender, EventArgs e)
{
    btnCancel.Enabled = false;
}

private void _Handle_ChartTypeChanged(object sender, EventArgs e)
{
    {
        foreach (ToolStripMenuItem it in
внешнийВидГрафиковToolStripMenuItem.DropDownItems.Cast<ToolStripItem>().TakeWhile(taken => taken is
ToolStripMenuItem))
        {
            it.Checked = false;
        }
    }
}

```

```

        var it = ((ToolStripMenuItem)(sender));
        it.Checked = true;
        _sampleChartSeries.ChartType = (SeriesChartType)(it.Tag);
    }
    foreach (var s in Chart.Series)
    {
        s.ChartType = _sampleChartSeries.ChartType;
    }
}

private void lvGenerators_ItemChecked(object sender, ItemCheckedEventArgs e)
{
    var index = lvGenerators.Items.IndexOf(e.Item);
    var area = Chart.ChartAreas.FindByName("ChartArea" + index.ToString());
    area.Visible = e.Item.Checked;

    var items = lvGenerators.Items.Cast<ListViewItem>();
    var chItems = items.Where(it => it.Checked);
    var col = lvGenerators.Columns[0];
    col.Tag = items.Count() == chItems.Count() ? new bool?(true) : (chItems.Count() > 0 ? null :
new bool?(false));
    col.Text = col.Text; // =)
}

private void lvMethods_ItemChecked(object sender, ItemCheckedEventArgs e)
{
    var methodNo = lvMethods.Items.IndexOf(e.Item);
    for (var generatorNo = 0; generatorNo < Program.Generators.Count; generatorNo++)
    {
        var series = _chartSeriesByGeneratorAndMethodIndices
            [new GeneratorSorterKey(generatorNo, methodNo)];
        var area = Chart.ChartAreas.FindByName("ChartArea" + generatorNo.ToString());
        series.ChartArea = e.Item.Checked
            ? area.Name
            : _invisibleAreaName;
        _RescaleChartYAxis(area);
    }
    var items = lvMethods.Items.Cast<ListViewItem>();
    var chItems = items.Where(it => it.Checked);
    var col = lvMethods.Columns[0];
    col.Tag = items.Count() == chItems.Count() ? new bool?(true) : (chItems.Count() > 0 ? null :
new bool?(false));
    col.Text = col.Text; // =)
}

void _RescaleChartYAxis(ChartArea area)
{
    var seriesBelongedToArea = Chart.Series.Where(s => s.ChartArea == area.Name && s.Points.Count
> 0);
    if (seriesBelongedToArea.Count() > 0)
    {
        var maximum = seriesBelongedToArea.Max(s => s.Points.Max(p => p.YValues.Single()));
        var roundMaximum = maximum > 1000
            ? Math.Ceiling(maximum / 1000) * 1000
            : maximum;
        area.AxisY.Maximum = roundMaximum;
        area.AxisY.Interval = roundMaximum / 5;
    }
}

readonly string _invisibleAreaName = "InvisibleChartArea";

private void numPercentage_ValueChanged(object sender, EventArgs e)
{
    var index = lvGenerators.SelectedIndices[0];
    var generator = Program.Generators[index] as
SortingExtensionAPI.ArrayGenerators.PartiallyUnsorted;
    generator.Percentage = (int)numPercentage.Value;
    lvGenerators.SelectedItems[0].Text = generator.Name;
    Chart.Titles[index].Text = generator.Name;
    lvMeasuring.Columns[index + 1].Text = generator.Name;
}

private void lvGenerators_SelectedIndexChanged(object sender, EventArgs e)
{
    if (lvGenerators.SelectedIndices.Count > 0)
    {
        var index = lvGenerators.SelectedIndices[0];

```

```

        if (Program.Generators[index] is SortingExtensionAPI.ArrayGenerators.PartiallyUnsorted)
        {
            gbPercentage.Visible = true;
            numPercentage.Value = (Program.Generators[lvGenerators.SelectedIndices[0]] as
SortingExtensionAPI.ArrayGenerators.PartiallyUnsorted).Percentage;
            return;
        }
    }
    gbPercentage.Visible = false;
}

private void lvMeasuring_SizeChanged(object sender, EventArgs e)
{
    _AdjustMeasuringColumns();
}

System.Windows.Forms.Timer _demoAdjustingTimer = new System.Windows.Forms.Timer();

private void lvDemonstration_SizeChanged(object sender, EventArgs e)
{
    _demoAdjustingTimer.Stop();
    _demoAdjustingTimer.Start();
}

private void lvMethods_SizeChanged(object sender, EventArgs e)
{
    lvMethods.Columns[0].Width = lvMethods.ClientSize.Width;
}

private void lvGenerators_SizeChanged(object sender, EventArgs e)
{
    lvGenerators.Columns[0].Width = lvGenerators.ClientSize.Width;
}

private void ставитьМеткиВУзлахToolStripMenuItem_Click(object sender, EventArgs e)
{
    var ms = ставитьМеткиВУзлахToolStripMenuItem.Checked
        ? MarkerStyle.Square
        : MarkerStyle.None;
    _sampleChartSeries.MarkerStyle = ms;
    foreach (var s in Chart.Series)
    {
        s.MarkerStyle = ms;
    }
}

private void lbDemoSteps_SelectedIndexChanged(object sender, EventArgs e)
{
    var i = lbDemoSteps.SelectedIndex;
    if (i != -1)
    {
        demoChart.Series.Clear();
        demoChart.Series.Add(_chartDemoMng.GetSeriesAt(i));
    }
}

private async void btnCreateChart_Click_New(object sender, EventArgs e)
{
    var args = new TimeBenchmark.Parameters()
    {
        CountTimes = (int)numChartIterationsCount.Value,
        MinLength = (int)numChartLeftBorder.Value * 1000,
        MaxLength = (int)numChartRightBorder.Value * 1000,
        StepsCount = (int)numChartStepsCount.Value,
        GenerationsPerPoint = (int)numGenerationsPerPoint.Value,
        GeneratorsNumbers = lvGenerators.CheckedItems.Cast<ListViewItem>().Select(it =>
lvGenerators.Items.IndexOf(it)),
        SortersNumbers = lvMethods.CheckedItems.Cast<ListViewItem>().Select(it =>
lvMethods.Items.IndexOf(it)),
        MaxRangeMultiplier = 2.0
    };

    var bench = new TimeBenchmark(args);

    void cancelAsked(object senderLocal, EventArgs eLocal)
    {
        bench.Cancel();
    }

    btnCancel.Click += cancelAsked;
}

```

```

        System.Windows.Forms.Timer timer = null;

        var progressPerResult = (double)progressBar.Maximum / args.StepsCount /
args.SortersNumbers.Count() / args.GeneratorsNumbers.Count();
        var progress = 0.0;

        void handleResults()
        {
            foreach (var result in bench.GetCurrentResults())
            {
                var s = _chartSeriesByGeneratorAndMethodIndices[new
GeneratorSorterKey(result.GeneratorNumber, result.SorterNumber)];
                var xy = new DataPoint(result.ArrayLength, result.ElapsedTime.Ticks);
                var insertionIndex = 0;
                while (insertionIndex < s.Points.Count && s.Points[insertionIndex].XValue <
result.ArrayLength)
                    insertionIndex++;
                if (insertionIndex < s.Points.Count)
                    s.Points.Insert(insertionIndex, xy);
                else
                    s.Points.Add(xy);

                _RescaleChartYAxis(Chat.ChartAreas.FindByName("ChartArea" + result.GeneratorNumber));

                progress += progressPerResult;
                var iProgress = (int)progress;
                progressBar.Value = iProgress <= progressBar.Maximum
                    ? iProgress
                    : progressBar.Maximum;
            }
        }

        try
        {
            void prepare()
            {
                timer = _PrepareFormForCalculation();
                timer.Interval = 1000;
                timer.Tick += new EventHandler((o, eLocal) => handleResults());

                foreach (var s in Chart.Series) s.Points.Clear();

                foreach (var generatorNo in args.GeneratorsNumbers)
                {
                    var area = Chart.ChartAreas.FindByName("ChartArea" + generatorNo.ToString());
                    area.AxisX.Minimum = args.MinLength;
                    area.AxisX.Maximum = args.MaxLength;
                }

                timer.Start();
            }

            await bench.StartMeasuringAsync(_processorCount, _regularPriority, _calculationPriority,
prepare);

            timer.Stop();
            handleResults();

            _Handle_CalculationDone(bench.Canceled ? "Вычисления отменены" : "Графики построены");
        }
        catch (InvalidOperationException ex)
        {
            {
                MessageBox.Show(ex.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            }
        }
        finally
        {
            {
                btnCancel.Click -= cancelAsked;
            }
        }
    }

    private async void btnMeasureCharacteristics_Click_New(object sender, EventArgs e)
    {
        var args = new TimeBenchmark.Parameters()
        {
            CountTimes = (int)numChartIterationsCount.Value,
            MinLength = (int)numElementsForCharacteristicsCount.Value * 1000,
            MaxLength = (int)numElementsForCharacteristicsCount.Value * 1000,
            GenerationsPerPoint = 1,

```

```

        GeneratorsNumbers = lvGenerators.CheckedItems.Cast<ListViewItem>().Select(it =>
lvGenerators.Items.IndexOf(it)),
        SortersNumbers = lvMethods.CheckedItems.Cast<ListViewItem>().Select(it =>
lvMethods.Items.IndexOf(it)),
        MaxRangeMultiplier = 2.0
    };
    var bench = new TimeBenchmark(args);

    void cancelAsked(object senderLocal, EventArgs eLocal)
    {
        bench.Cancel();
    }
    btnCancel.Click += cancelAsked;

    var progressPerResult = (double)progressBar.Maximum / args.SortersNumbers.Count() /
args.GeneratorsNumbers.Count();
    var progress = 0.0;

    void handleResults()
    {
        foreach (var result in bench.GetCurrentResults())
        {
            var it = lvMeasuring.Items[result.SorterNumber];
            var sb = it.SubItems[result.GeneratorNumber + 1];
            sb.Text = radMs.Checked
                ? result.ElapsedTime.TotalMilliseconds.ToString("F4")
                : result.ElapsedTime.Ticks.ToString("N0");
            progress += progressPerResult;
            var iProgress = (int)progress;
            progressBar.Value = iProgress <= progressBar.Maximum
                ? iProgress
                : progressBar.Maximum;
        }
    }

    System.Windows.Forms.Timer timer = null;

    try
    {
        void prepare()
        {
            timer = _PrepareFormForCalculation();
            timer.Interval = 1000;
            timer.Tick += new EventHandler((o, eLocal) => handleResults());

            foreach (var sb in lvMeasuring.Items.Cast<ListViewItem>()
                .SelectMany(it => it.SubItems.Cast<ListViewItem.ListViewSubItem>()
                    .Where(itsb => it.SubItems.IndexOf(itsb) > 0)))
            {
                sb.Text = "-";
            }

            timer.Start();
        }

        await bench.StartMeasuringAsync(_processorCount, _regularPriority, _calculationPriority,
prepare);

        timer.Stop();
        handleResults();

        _Handle_CalculationDone(bench.Canceled ? "Вычисления отменены" : "Таблица построена");
    }
    catch (InvalidOperationException ex)
    {
        MessageBox.Show(ex.Message, "", MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }
    finally
    {
        btnCancel.Click -= cancelAsked;
    }
}

private void checkableDrawColumnHeader(object sender, DrawListViewColumnHeaderEventArgs e)
{
    System.Windows.Forms.VisualStyles.CheckBoxState getCheckBoxState(bool? state)
    {
        return state != null ? (bool)state
            ? System.Windows.Forms.VisualStyles.CheckBoxState.CheckedNormal

```



```

public static Color GetChartSeriesColor(int i)
{
    if (i < _chartColors.Count)
    {
        return _chartColors[i];
    }
    var color = GetRandom();
    lock (_collectionMutatingLocker)
    {
        _chartColors.Add(color);
    }
    return color;
}

public static Color GetRegularGroupColor() => Color.Gray;

private static List<Color> _demoColors = new List<Color>
{
    Color.SandyBrown,
    Color.DarkViolet,
    Color.Green,
    Color.Salmon,
    Color.DeepSkyBlue,
    Color.GreenYellow,
};

public static Color GetDemonstrationGroupColor(int i)
{
    if (i < _demoColors.Count)
    {
        return _demoColors[i];
    }
    var color = GetRandom();
    lock (_collectionMutatingLocker)
    {
        _demoColors.Add(color);
    }
    return color;
}

public static Color GetSwappedColor() => Color.Red;

private static Random _random = new Random();

private static object _randomLocker = new object();

public static Color GetRandom()
{
    lock (_randomLocker)
    {
        var r = _random.Next(256);
        var g = _random.Next(256);
        var b = _random.Next(256);
        return Color.FromArgb(255, r, g, b);
    }
}

static Color[] _rowColors = { Color.White, Color.Moccasin };
static int _rowColorNo = 0;
static object _rowLocker = new object();

public static Color GetTableRowColor()
{
    lock (_rowLocker)
    {
        _rowColorNo = (_rowColorNo + 1) % 2;
        return _rowColors[_rowColorNo];
    }
}
}
}

```

#### **DiagramDemonstrationRecorder.cs**

```

using System;
using System.Collections.Generic;
using System.Drawing;
using System.Linq;
using System.Text;

```

```

using System.Threading.Tasks;
using System.Windows.Forms.DataVisualization.Charting;
using SortingExtensionAPI;

namespace Sorting
{
    class DiagramDemonstrationRecorder
    {
        List<Series> _series = new List<Series>();

        Series NewSeries()
        {
            _series.Add(new Series());
            var s = _series.Last();
            s.ChartType = SeriesChartType.Column;
            return s;
        }

        public string AddFrame(object sender, DemonstrationCallbackEventArgs e)
        {
            var groups = e.IndicesGroups.ToList();
            groups.Reverse();

            var s = NewSeries();

            foreach (var index in Enumerable.Range(0, e.Array.Length))
            {
                var element = e.Array[index] != 0
                    ? e.Array[index]
                    : 0.2;
                var p = s.Points.AddXY(index, element);
                var upperGroup = groups.Find(g => g.Collection.Contains(index));
                s.Points[p].Color = upperGroup != null
                    ? ColorSupplier.GetDemonstrationGroupColor(upperGroup.GroupNumber)
                    : ColorSupplier.GetRegularGroupColor();
            }

            foreach (var h in e.HiddenElements)
            {
                s.Points[h].Color = Color.FromArgb(0, 0, 0, 0);
            }

            if (e.SwappedPairIndices != null)
            {
                s.Points[e.SwappedPairIndices.Item1].Color = ColorSupplier.GetSwappedColor();
                s.Points[e.SwappedPairIndices.Item2].Color = ColorSupplier.GetSwappedColor();
            }

            return e.DescriptionOfChangeShort;
        }

        public Series GetSeriesAt(int i) => _series[i];

        public void Clear() => _series.Clear();
    }
}

```

## Проект «SortingExtensionAPI»

### ArraySorter.cs

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;

namespace SortingExtensionAPI
{
    /// <summary>
    /// Метод сортировки
    /// </summary>
    public abstract class ArraySorter
    {
        /// <summary>
        /// Событие вызывается при изменении массива во время демонстрационной сортировки
        /// </summary>
        public event EventHandler<DemonstrationCallbackEventArgs> DemonstrationCallback;
    }
}

```

```

    /// <summary>
    /// Вызывает событие DemonstrationCallback
    /// </summary>
    /// <param name="e">Аргументы события</param>
    protected void Demonstrate(DemonstrationCallbackEventArgs e) =>
    DemonstrationCallback?.Invoke(this, e);

    /// <summary>
    /// Вызывает событие DemonstrationCallback
    /// </summary>
    /// <param name="e">Аргументы события</param>
    /// <param name="descriptionLong">Описание шага</param>
    /// <param name="descriptionShort">Короткое описание шага</param>
    protected void Demonstrate(DemonstrationCallbackEventArgs e, string descriptionShort, string
descriptionLong)
    {
        e.DescriptionOfChangeShort = descriptionShort;
        e.DescriptionOfChange = descriptionLong;
        DemonstrationCallback?.Invoke(this, e);
    }

    /// <summary>
    /// Вызывает событие DemonstrationCallback
    /// </summary>
    /// <param name="e">Аргументы события</param>
    /// <param name="descriptionShort">Короткое описание шага</param>
    protected void Demonstrate(DemonstrationCallbackEventArgs e, string descriptionShort)
    {
        e.DescriptionOfChangeShort = descriptionShort;
        DemonstrationCallback?.Invoke(this, e);
    }

    /// <summary>
    /// Оптимизированная функция сортировки, пригодная для наиболее точного измерения временных
характеристик метода
    /// </summary>
    /// <param name="array"></param>
    /// <returns>Возвращает время, затраченное на сортировку</returns>
    public virtual TimeSpan MeasureSortTime(int[] array)
    {
        throw new NotSupportedException();
    }

    /// <summary>
    /// Демонстрационная функция сортировки, поддерживающая обратную связь с помощью события
DemonstrationCallback
    /// </summary>
    /// <param name="array">Входной массив</param>
    public virtual void SortArrayDemonstration(int[] array)
    {
        throw new NotSupportedException();
    }

    /// <summary>
    /// Название метода
    /// </summary>
    public abstract string Name { get; }
}

```

### ArrayGenerator.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace SortingExtensionAPI
{
    public abstract class ArrayGenerator
    {
        public abstract string Name { get; }
        public abstract int[] Generate(int length, int maxRange);
        private static readonly Random _randomizer = new Random();
        private static object _randomLocker = new object();
        protected int GetRandom(int min, int max)
        {
            lock (_randomLocker)
            {
                return _randomizer.Next(min, max + 1);
            }
        }
    }
}

```



```

        array[whereToPlace] = array[i];
        array[i] = tmp;
        elementIsUnsortedAt[i] = true;
        elementIsUnsortedAt[whereToPlace] = true;
    }
}
}

public class PartiallyUnsorted : ArrayGenerator
{
    public override string Name => String.Format("Частично неупорядоченный ({0}%)",
_percentage.ToString() );
    private int _percentage;
    public PartiallyUnsorted()
    {
        Percentage = 50;
    }
    public PartiallyUnsorted(int percentage)
    {
        Percentage = percentage;
    }
    public int Percentage
    {
        get => _percentage;
        set
        {
            if (value >= 0 || value <= 100)
            {
                _percentage = value;
            }
            else
            {
                throw new ArgumentOutOfRangeException("Значение процентной доли должно лежать в
диапазоне от 0 до 100");
            }
        }
    }
}

public override int[] Generate(int length, int maxRange)
{
    var result = new Sorted().Generate(length, maxRange);
    var unsortedElementsCount = length * Percentage / 100;
    var unsortedElementsIndices = new int[unsortedElementsCount];
    var unsortedElements = new int[unsortedElementsCount];
    {
        int[] availableIndices = new int[length];
        for (var i = 0; i < length; i++)
        {
            availableIndices[i] = i;
        }
        for (var i = 0; i < unsortedElementsCount; i++)
        {
            var number = GetRandom(i, length - 1);
            var index = availableIndices[number];
            unsortedElementsIndices[i] = index;
            availableIndices[number] = availableIndices[i];
        }
    }
    unsortedElementsIndices = unsortedElementsIndices.OrderBy(x => x).ToArray();
    for (var i = 0; i < unsortedElementsCount; i++)
    {
        var index = unsortedElementsIndices[i];
        unsortedElements[i] = result[index];
    }
    new TotallyUnsorted().Unsort(unsortedElements);
    for (var i = 0; i < unsortedElementsCount; i++)
    {
        var whereToPlace = unsortedElementsIndices[i];
        var whatToPlace = unsortedElements[i];
        result[whereToPlace] = whatToPlace;
    }
    return result;
}

public class SortedInReverse : ArrayGenerator
{
    public override string Name => "Отсортированный в обратном порядке";

```

```

        public override int[] Generate(int length, int maxRange)
        {
            var array = new TotallyRandomNoDuplicates().Generate(length, maxRange);
            return array.OrderByDescending(x => x).ToArray();
        }
    }

    public class AllEqual : ArrayGenerator
    {
        public override string Name => "Все числа равны";
        public override int[] Generate(int length, int maxRange)
        {
            var array = new int[length];
            var number = GetRandom(0, maxRange - 1);
            for (var i = 0; i < length; i++)
            {
                array[i] = number;
            }
            return array;
        }
    }
}

```

### DemonstrationManager.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SortingExtensionAPI
{
    public class DemonstrationManager
    {
        public DemonstrationCallbackEventArgs E { get; private set; }

        public void Reset() => E = new DemonstrationCallbackEventArgs();

        public DemonstrationManager(int[] array)
        {
            Reset();
            E.Array = array;
        }

        public void SelectSwapped(int i1, int i2) => E.SwappedPairIndices = new Tuple<int, int>(i1, i2);

        public void UnselectSwapped() => E.SwappedPairIndices = null;

        public void SetGroups(params DemonstrationIndicesGroup[] groups) => E.IndicesGroups =
            groups.ToList();

        public void AddGroups(params DemonstrationIndicesGroup[] groups)
        {
            foreach (var g in groups)
            {
                E.IndicesGroups.Add(g);
            }
        }

        public DemonstrationIndicesGroup AddGroup(IEnumerable<int> collection, int groupNo)
        {
            var group = new DemonstrationIndicesGroup(collection, groupNo);
            E.IndicesGroups.Add(group);
            return group;
        }

        public void SetGroup(IEnumerable<int> group, int groupNo)
        {
            E.IndicesGroups = new List<DemonstrationIndicesGroup> { new DemonstrationIndicesGroup(group,
                groupNo) };
        }

        public DemonstrationIndicesGroup AddSingle(int index, int groupNo)
        {
            var group = new DemonstrationIndicesGroup(new List<int>() { index }, groupNo);
            E.IndicesGroups.Add(group);
            return group;
        }
    }
}

```

```

        public void RemoveLastAddedGroup()
        {
            if (E.IndicesGroups.Count > 0)
                E.IndicesGroups.Remove(E.IndicesGroups.Last());
        }

        public DemonstrationIndicesGroup GetLastAddedGroup()
        {
            return E.IndicesGroups.Count > 0 ? E.IndicesGroups.Last() : null;
        }

        public void UnsetGroups() => E.IndicesGroups = null;
    }
}

```

### **DemonstrationCallbackEventArgs.cs**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SortingExtensionAPI
{
    /// <summary>
    /// Представляет класс, содержащий аргументы события DemonstrationCallback
    /// </summary>
    public class DemonstrationCallbackEventArgs : EventArgs
    {
        /// <summary>
        /// Сортируемый массив
        /// </summary>
        public int[] Array;

        /// <summary>
        /// Индексы пары элементов, которые меняются местами
        /// </summary>
        public Tuple<int, int> SwappedPairIndices;

        /// <summary>
        /// Индексы элементов по группам
        /// </summary>
        public List<DemonstrationIndicesGroup> IndicesGroups = new List<DemonstrationIndicesGroup>();

        /// <summary>
        /// Текстовое описание изменения
        /// </summary>
        public string DescriptionOfChange;

        /// <summary>
        /// Короткое описание изменения, например, номер прохода
        /// </summary>
        public string DescriptionOfChangeShort;

        /// <summary>
        /// Скрытые элементы
        /// </summary>
        public List<int> HiddenElements = new List<int>();
    }
}

```

### **DemonstrationIndicesGroup.cs**

```

using System;
using System.Collections.Generic;
using System.Collections;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace SortingExtensionAPI
{
    public class DemonstrationIndicesGroup
    {
        public List<int> Collection;
        public int GroupNumber;
        public DemonstrationIndicesGroup(IEnumerable<int> group, int groupNo)
        {
            Collection = group.ToList();
        }
    }
}

```

```

        GroupNumber = groupNo;
    }
    public DemonstrationIndicesGroup()
    {
        Collection = new List<int>();
        GroupNumber = 0;
    }
    public IEnumerator<int> GetEnumerator() => Collection.GetEnumerator();
    public static implicit operator List<int>(DemonstrationIndicesGroup group)
    {
        return group.Collection;
    }
}
}
}

```

## Проект «Bubble»

### BubbleSort.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using SortingExtensionAPI;

namespace Bubble
{
    public class BubbleSort : ArraySorter
    {
        public override string Name => "Пузырьком";

        public override TimeSpan MeasureSortTime(int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            for (var i = 0; i < array.Length; i++)
            {
                for (var j = 1; j < array.Length - i; j++)
                {
                    if (array[j - 1] > array[j])
                    {
                        var temp = array[j - 1];
                        array[j - 1] = array[j];
                        array[j] = temp;
                    }
                }
            }

            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)
        {
            var demoMng = new DemonstrationManager(array);
            demoMng.AddGroup(new List<int> {-1, -1}, 4);
            var comparableGroup = demoMng.GetLastAddedGroup();
            demoMng.AddGroup(new List<int> {}, 2);
            var sortedGroup = demoMng.GetLastAddedGroup();

            for (var i = 0; i < array.Length; i++)
            {
                for (var j = 1; j < array.Length - i; j++)
                {
                    comparableGroup.Collection[0] = j;
                    comparableGroup.Collection[1] = j - 1;
                    Demonstrate(demoMng.E, "Сравнение");

                    if (array[j - 1] > array[j])
                    {
                        var temp = array[j - 1];
                        array[j - 1] = array[j];
                        array[j] = temp;

                        demoMng.SelectSwapped(j, j - 1);
                        Demonstrate(demoMng.E, "Обмен");
                    }
                }
            }
        }
    }
}

```

```

        demoMng.UnselectSwapped();
    }
}

sortedGroup.Collection.Add(array.Length - 1 - i);
Demonstrate(demoMng.E, "Проход сделан");
}
}
}
}
}

```

### BubbleSortDetectsIfSorted.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Diagnostics;
using SortingExtensionAPI;

namespace Bubble
{
    public class BubbleSortDetectsIfSorted : ArraySorter
    {
        public override string Name
        { get => "Пузырьком с определением отсортированности"; }

        public override TimeSpan MeasureSortTime(int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            for (var i = 0; i < array.Length; i++)
            {
                var swapped = false;
                for (var j = 1; j < array.Length - i; j++)
                {
                    if (array[j - 1] > array[j])
                    {
                        var temp = array[j - 1];
                        array[j - 1] = array[j];
                        array[j] = temp;
                        swapped = true;
                    }
                }
                if (!swapped)
                    break;
            }

            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)
        {
            var demoMng = new DemonstrationManager(array);
            demoMng.AddGroup(new List<int> { -1, -1 }, 4);
            var comparableGroup = demoMng.GetLastAddedGroup();
            demoMng.AddGroup(new List<int>(), 2);
            var sortedGroup = demoMng.GetLastAddedGroup();

            for (var i = 0; i < array.Length; i++)
            {
                var swapped = false;
                for (var j = 1; j < array.Length - i; j++)
                {
                    comparableGroup.Collection[0] = j;
                    comparableGroup.Collection[1] = j - 1;
                    Demonstrate(demoMng.E, "Сравнение");

                    if (array[j - 1] > array[j])
                    {
                        var temp = array[j - 1];
                        array[j - 1] = array[j];
                        array[j] = temp;
                        swapped = true;

                        demoMng.SelectSwapped(j, j - 1);
                        Demonstrate(demoMng.E, "Обмен");
                    }
                }
            }
        }
    }
}

```



```

    }

    if (largestIndex == index)
    {
        return;
    }
    var temp = array[index];
    array[index] = array[largestIndex];
    array[largestIndex] = temp;

    index = largestIndex;
}
}

public override void SortArrayDemonstration(int[] array)
{
    var dm = new DemonstrationManager(array);
    dm.AddGroup(new List<int>(), 2);

    for (var index = array.Length / 2 - 1; index >= 0; index--)
    {
        HeapifyDemo(array, index, array.Length, dm);
    }

    for (var high = array.Length - 1; high > 0; high--)
    {
        var temp = array[0];
        array[0] = array[high];
        array[high] = temp;

        dm.SelectSwapped(0, high);
        Demonstrate(dm.E, "Обмен");
        dm.UnselectSwapped();

        HeapifyDemo(array, 0, high, dm);
    }
}

void HeapifyDemo(int[] array, int rootIndex, int high, DemonstrationManager dm)
{
    var heapGroup = dm.AddGroup(new List<int>(), 0);
    var largestGroup = dm.AddGroup(new List<int>(), 1);
    while (true)
    {
        heapGroup.Collection.Clear();
        largestGroup.Collection.Clear();

        var childLeftIndex = 2 * rootIndex + 1;
        var childRightIndex = childLeftIndex + 1;
        int largestIndex = rootIndex;

        if (high < array.Length)
        {
            dm.E.IndicesGroups[0].Collection.Add(high);
        }
        heapGroup.Collection = new List<int> { rootIndex, childLeftIndex, childRightIndex }
        }.Where(el => el < high).ToList();
        Demonstrate(dm.E, "Вход в узел");
        largestGroup.Collection = new List<int> { largestIndex };

        if (childLeftIndex < high && array[childLeftIndex] > array[largestIndex])
        {
            largestIndex = childLeftIndex;
        }
        if (childRightIndex < high && array[childRightIndex] > array[largestIndex])
        {
            largestIndex = childRightIndex;
        }

        if (largestIndex == rootIndex)
        {
            largestGroup.Collection[0] = largestIndex;
            Demonstrate(dm.E, "Максимум - корень");

            dm.E.IndicesGroups.Remove(heapGroup);
            dm.E.IndicesGroups.Remove(largestGroup);
            return;
        }
    }
}

```

```

        largestGroup.Collection[0] = largestIndex;
        Demonstrate(dm.E, "Максимум - потомок");

        var temp = array[rootIndex];
        array[rootIndex] = array[largestIndex];
        array[largestIndex] = temp;

        dm.SelectSwapped(rootIndex, largestIndex);
        Demonstrate(dm.E, "Обмен");
        dm.UnselectSwapped();

        rootIndex = largestIndex;
    }
}
}
}
}

```

## Проект «InsertionSort»

### InsertionSort.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SortingExtensionAPI;
using System.Diagnostics;

namespace InsertionSort
{
    public class InsertionSort : ArraySorter
    {
        public override string Name => "Вставками";

        public override TimeSpan MeasureSortTime(int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            for (var placeableIndex = 1; placeableIndex < array.Length; placeableIndex++)
            {
                var placeable = array[placeableIndex];
                var comparableIndex = placeableIndex - 1;
                for (; comparableIndex >= 0; comparableIndex--)
                {
                    if (array[comparableIndex] > placeable)
                    {
                        array[comparableIndex + 1] = array[comparableIndex];
                    }
                    else break;
                }
                array[comparableIndex + 1] = placeable;
            }

            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)
        {
            var dm = new DemonstrationManager(array);

            var sortedGrp = dm.AddSingle(0, 2);
            var placeGrp = dm.AddGroup(new List<int>(), 1);
            var compGrp = dm.AddGroup(new List<int>(), 4);

            for (var placeableIndex = 1; placeableIndex < array.Length; placeableIndex++)
            {
                placeGrp.Collection.Add(placeableIndex);
                Demonstrate(dm.E, "Запоминание");
                placeGrp.Collection.Clear();

                sortedGrp.Collection.Add(placeableIndex);

                var placeable = array[placeableIndex];
                var comparableIndex = placeableIndex - 1;
            }
        }
    }
}

```

```

        compGrp.Collection.Add(-1);

        for (; comparableIndex >= 0; comparableIndex--)
        {
            compGrp.Collection[0] = comparableIndex;
            Demonstrate(dm.E, "Сравнение");

            if (array[comparableIndex] > placeable)
            {
                array[comparableIndex + 1] = array[comparableIndex];

                dm.SelectSwapped(comparableIndex + 1, comparableIndex + 1);
                Demonstrate(dm.E, "Сдвиг");
                dm.UnselectSwapped();
            }
            else break;
        }
        array[comparableIndex + 1] = placeable;

        dm.SelectSwapped(comparableIndex + 1, comparableIndex + 1);
        compGrp.Collection.Clear();
        Demonstrate(dm.E, "Вставка");
        dm.UnselectSwapped();
    }
}
}
}
}

```

### InsertionSortEnclosesFake.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SortingExtensionAPI;
using System.Diagnostics;

namespace InsertionSort
{
    public class InsertionSortEnclosesFake : ArraySorter
    {
        public override string Name => "Вставками с добавлением фиктивного элемента";

        public override TimeSpan MeasureSortTime(int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            var realFirst = array[0];
            array[0] = -1;

            for (var placeableIndex = 2; placeableIndex < array.Length; placeableIndex++)
            {
                var placeable = array[placeableIndex];
                var comparableIndex = placeableIndex;
                while (array[--comparableIndex] > placeable)
                {
                    array[comparableIndex + 1] = array[comparableIndex];
                }
                array[comparableIndex + 1] = placeable;
            }
            {
                var comparableIndex = 1;
                for (; comparableIndex < array.Length; comparableIndex++)
                {
                    if (array[comparableIndex] < realFirst)
                    {
                        array[comparableIndex - 1] = array[comparableIndex];
                    }
                    else break;
                }
                array[comparableIndex - 1] = realFirst;
            }
            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)

```

```

{
    var dm = new DemonstrationManager(array);

    var realFirst = array[0];
    array[0] = -1;

    dm.SelectSwapped(0, 0);
    Demonstrate(dm.E, "Подмена");
    dm.UnselectSwapped();
    var borderGrp = dm.AddSingle(0, 5);
    var sortedGrp = dm.AddSingle(1, 2);
    var compGrp = dm.AddGroup(new List<int>(), 4);
    {
        var placeGrp = dm.AddGroup(new List<int>(), 1);

        for (var placeableIndex = 2; placeableIndex < array.Length; placeableIndex++)
        {
            sortedGrp.Collection.Add(placeableIndex);

            placeGrp.Collection.Add(placeableIndex);
            Demonstrate(dm.E, "Запоминание");
            placeGrp.Collection.Clear();

            compGrp.Collection.Add(placeableIndex - 1);
            Demonstrate(dm.E, "Сравнение");

            var placeable = array[placeableIndex];
            var comparableIndex = placeableIndex - 1;
            while (array[comparableIndex] > placeable)
            {
                array[comparableIndex + 1] = array[comparableIndex--];

                dm.SelectSwapped(comparableIndex + 2, comparableIndex + 2);
                Demonstrate(dm.E, "Сдвиг");
                dm.UnselectSwapped();

                compGrp.Collection[0] = comparableIndex;
                Demonstrate(dm.E, "Сравнение");
            }
            array[comparableIndex + 1] = placeable;
            dm.SelectSwapped(comparableIndex + 1, comparableIndex + 1);
            Demonstrate(dm.E, "Вставка");
            dm.UnselectSwapped();
            compGrp.Collection.Clear();
        }
    }
    {
        var comparableIndex = 1;

        for (; comparableIndex < array.Length; comparableIndex++)
        {
            compGrp.Collection.Clear();
            compGrp.Collection.Add(comparableIndex);
            Demonstrate(dm.E, "Сравнение");

            if (array[comparableIndex] < realFirst)
            {
                array[comparableIndex - 1] = array[comparableIndex];

                dm.SelectSwapped(comparableIndex - 1, comparableIndex - 1);
                sortedGrp.Collection.Add(0);
                Demonstrate(dm.E, "Сдвиг");
                dm.UnselectSwapped();
            }
            else
            {
                compGrp.Collection.Clear();
                break;
            }
        }
        array[comparableIndex - 1] = realFirst;

        dm.SelectSwapped(comparableIndex - 1, comparableIndex - 1);
        Demonstrate(dm.E, "Вставка");
        dm.UnselectSwapped();
    }
}
}
}
}
}

```

**InsertionSortFindsMinimum.cs**

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using SortingExtensionAPI;
using System.Diagnostics;

namespace InsertionSort
{
    public class InsertionSortFindsMinimum : ArraySorter
    {
        public override string Name => "Вставками с нахождением минимума";

        public override TimeSpan MeasureSortTime(int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            {
                var min = 0;
                for (var i = 1; i < array.Length; i++)
                {
                    if (array[i] < array[min]) min = i;
                }
                var temp = array[min];
                array[min] = array[0];
                array[0] = temp;
            }

            for (var placeableIndex = 2; placeableIndex < array.Length; placeableIndex++)
            {
                var placeable = array[placeableIndex];
                var comparableIndex = placeableIndex;
                while (array[--comparableIndex] > placeable)
                {
                    array[comparableIndex + 1] = array[comparableIndex];
                }
                array[comparableIndex + 1] = placeable;
            }

            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)
        {
            var dm = new DemonstrationManager(array);

            {
                var compGrp = dm.AddSingle(0, 4);
                var minGrp = dm.AddSingle(0, 5);
                Demonstrate(dm.E, "Минимум");

                var min = 0;
                for (var i = 1; i < array.Length; i++)
                {
                    compGrp.Collection[0] = i;
                    Demonstrate(dm.E, "Сравнение");

                    if (array[i] < array[min])
                    {
                        min = i;
                        minGrp.Collection[0] = min;
                        Demonstrate(dm.E, "Минимум");
                    }
                }
                dm.E.IndicesGroups.Remove(compGrp);

                var temp = array[min];
                array[min] = array[0];
                array[0] = temp;

                dm.SelectSwapped(0, min);
                Demonstrate(dm.E, "Обмен");
                dm.UnselectSwapped();
                minGrp.Collection[0] = 0;
            }
        }
    }
}

```



```

        for (var timesElementWritten = 0; timesElementWritten < timesElementGot[element];
timesElementWritten++)
        {
            array[placeableIndex++] = element;
        }
    }

    sw.Stop();
    return sw.Elapsed;
}

public override void SortArrayDemonstration(int[] array)
{
    var max = array[0];
    for (var i = 1; i < array.Length; i++)
    {
        if (array[i] > max)
            max = array[i];
    }

    var demoMngs = new DemonstrationManager[max + 1];
    var indices = Enumerable.Range(0, array.Length);

    var timesElementGot = new int[max + 1];
    for (var i = 0; i < array.Length; i++)
    {
        var element = array[i];
        timesElementGot[element] += 1;

        if (demoMngs[element] == null)
        {
            demoMngs[element] = new DemonstrationManager(new int[array.Length]);
            demoMngs[element].AddGroup(new List<int>(), -1);
            demoMngs[element].E.HiddenElements = indices.ToList();
        }

        demoMngs[element].E.Array[i] = timesElementGot[element];
        (demoMngs[element].E.HiddenElements as List<int>).Remove(i);
        demoMngs[element].GetLastAddedGroup().Collection.Add(i);
    }

    var dmSortable = new DemonstrationManager(array);
    {
        var group_a = dmSortable.AddGroup(new List<int>(), 2);
        var group_b = dmSortable.AddGroup(new List<int>(), 3);
        var group = group_a;

        var placeableIndex = 0;
        for (var element = 0; element <= max; element++)
        {
            if (timesElementGot[element] != 0)
                group = group == group_a ? group_b : group_a;
            for (var timesElementWritten = 0; timesElementWritten < timesElementGot[element];
timesElementWritten++)
            {
                group.Collection.Add(placeableIndex);
                array[placeableIndex++] = element;
            }
        }
    }

    {
        var groupNo = 0;
        for (var element = 0; element <= max; element++)
        {
            var dm = demoMngs[element];
            if (dm == null)
            {
                dm = new DemonstrationManager(new int[array.Length]);
                dm.E.HiddenElements = indices.ToList();
            }
            else
            {
                dm.GetLastAddedGroup().GroupNumber = groupNo;
                groupNo = (groupNo + 1) % 2;
            }
            Demonstrate(dm.E, element.ToString());
        }
    }
}

```

```

    }
    Demonstrate(dmSortable.E, "Построен");
}
}
}

```

## Проект «QuickSort»

### Quick.cs

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

using SortingExtensionAPI;

namespace Quick
{
    public abstract class QuickBase : ArraySorter
    {
        public override string Name { get => String.Format("Быстрый ({0})", _note); }
        protected string _note;

        private int _PartitionOptimized(int[] array, int leftBound, int rightBound, int pivot)
        {
            var leftIndex = leftBound;
            var rightIndex = rightBound;
            while (true)
            {
                while (array[leftIndex] < pivot)
                {
                    leftIndex++;
                }
                while (array[rightIndex] > pivot)
                {
                    rightIndex--;
                }
                if (leftIndex < rightIndex)
                {
                    var temp = array[rightIndex];
                    array[rightIndex--] = array[leftIndex];
                    array[leftIndex++] = temp;
                }
                else
                {
                    break;
                }
            }
            return rightIndex;
        }

        protected void _SelectPivotDemo(int i, DemonstrationManager demoMng)
        {
            var g = new int[1] { i };
            demoMng.AddGroups(new DemonstrationIndicesGroup(g, 1));
            Demonstrate(demoMng.E, "Опора взята");
            demoMng.E.IndicesGroups.RemoveAt(demoMng.E.IndicesGroups.Count - 1);
        }

        private int _PartitionDemo(int[] array, int leftBound, int rightBound, int pivot,
            DemonstrationManager demoMng)
        {
            var leftIndex = leftBound;
            var rightIndex = rightBound;
            while (true)
            {
                demoMng.AddSingle(leftIndex, 4);
                Demonstrate(demoMng.E, "Поиск большего");
                demoMng.E.IndicesGroups.Remove(demoMng.E.IndicesGroups.Last());

                while (array[leftIndex] < pivot)
                {
                    leftIndex++;

                    demoMng.AddSingle(leftIndex, 4);
                }
            }
        }
    }
}

```

```

        Demonstrate(demoMng.E, "Поиск большего");
        demoMng.E.IndicesGroups.Remove(demoMng.E.IndicesGroups.Last());
    }

    demoMng.AddSingle(leftIndex, 4);

    demoMng.AddSingle(rightIndex, 4);
    Demonstrate(demoMng.E, "Поиск меньшего");
    demoMng.E.IndicesGroups.Remove(demoMng.E.IndicesGroups.Last());

    while (array[rightIndex] > pivot)
    {
        rightIndex--;

        demoMng.AddSingle(rightIndex, 4);
        Demonstrate(demoMng.E, "Поиск меньшего");
        demoMng.E.IndicesGroups.Remove(demoMng.E.IndicesGroups.Last());
    }

    demoMng.RemoveLastAddedGroup();

    if (leftIndex < rightIndex)
    {
        var temp = array[rightIndex];
        array[rightIndex] = array[leftIndex];
        array[leftIndex] = temp;

        demoMng.SelectSwapped(leftIndex++, rightIndex--);
        Demonstrate(demoMng.E, "Обмен");
        demoMng.UnselectSwapped();
    }
    else
    {
        break;
    }
}
return rightIndex;
}

protected virtual int _GetPivotDemo(int[] array, int leftBound, int rightBound,
DemonstrationManager demoMng)
{
    throw new NotImplementedException();
}

public override void SortArrayDemonstration(int[] array)
{
    var demoMng = new DemonstrationManager(array);

    var bounds = new Stack<Tuple<int, int>>();
    bounds.Push(new Tuple<int, int>(0, array.Length - 1));
    while (bounds.Count > 0)
    {
        var popped = bounds.Pop();
        var leftBound = popped.Item1;
        var rightBound = popped.Item2;
        {
            var workingArea = new int[rightBound - leftBound + 1];
            for (var i = 0; i < workingArea.Length; i++)
            {
                workingArea[i] = i + leftBound;
            }
            demoMng.SetGroups(new DemonstrationIndicesGroup(workingArea, 0));
        }
        var separationIndex = _PartionDemo(array, leftBound, rightBound, _GetPivotDemo(array,
leftBound, rightBound, demoMng), demoMng);

        var lessIndices = new int[separationIndex + 1 - leftBound];
        for (var i = 0; i < lessIndices.Length; i++)
        {
            lessIndices[i] = leftBound + i;
        }

        var greaterIndices = new int[rightBound - separationIndex];
        for (var i = 0; i < greaterIndices.Length; i++)
        {
            greaterIndices[i] = 1 + i + separationIndex;
        }
    }
}

```

```

        demoMng.SetGroups(new DemonstrationIndicesGroup(lessIndices, 2), new
DemonstrationIndicesGroup(greaterIndices, 5));
        Demonstrate(demoMng.E, "Разбиение");

        if (separationIndex + 1 < rightBound)
        {
            bounds.Push(new Tuple<int, int>(separationIndex + 1, rightBound));
        }
        if (separationIndex > leftBound)
        {
            bounds.Push(new Tuple<int, int>(leftBound, separationIndex));
        }
    }
}

protected virtual int _GetPivot(int[] array, int leftBound, int rightBound)
{
    throw new NotImplementedException();
}

public override TimeSpan MeasureSortTime(int[] array)
{
    var sw = new Stopwatch();
    sw.Start();

    var bounds = new Stack<Tuple<int, int>>();
    bounds.Push(new Tuple<int, int>(0, array.Length - 1));
    while (bounds.Count > 0)
    {
        var popped = bounds.Pop();
        var leftBound = popped.Item1;
        var rightBound = popped.Item2;
        var separationIndex = _PartitionOptimized(array, leftBound, rightBound, _GetPivot(array,
leftBound, rightBound));

        if (separationIndex + 1 < rightBound)
        {
            bounds.Push(new Tuple<int, int>(separationIndex + 1, rightBound));
        }
        if (separationIndex > leftBound)
        {
            bounds.Push(new Tuple<int, int>(leftBound, separationIndex));
        }
    }

    sw.Stop();
    return sw.Elapsed;
}
}

namespace PivotIs
{
    public class First : QuickBase
    {
        public First()
        {
            _note = "опора - первый элемент";
        }

        protected override int _GetPivot(int[] array, int leftBound, int rightBound)
        {
            return array[leftBound];
        }

        protected override int _GetPivotDemo(int[] array, int leftBound, int rightBound,
DemonstrationManager demoMng)
        {
            _SelectPivotDemo(leftBound, demoMng);
            return array[leftBound];
        }
    }

    public class Middle : QuickBase
    {
        public Middle()
        {
            _note = "опора - средний элемент";
        }
    }
}

```



```

namespace SelectionSort
{
    public class SelectionSort : ArraySorter
    {
        public override string Name
        { get => "Прямой выбор"; }

        public override TimeSpan MeasureSortTime(int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            for (var insertionIndex = array.Length - 1; insertionIndex > 0; insertionIndex--)
            {
                var maxIndex = 0;
                for (var comparableIndex = 1; comparableIndex <= insertionIndex; comparableIndex++)
                {
                    if (array[comparableIndex] > array[maxIndex])
                    {
                        maxIndex = comparableIndex;
                    }
                }
                var max = array[maxIndex];
                array[maxIndex] = array[insertionIndex];
                array[insertionIndex] = max;
            }

            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)
        {
            var demoMng = new DemonstrationManager(array);

            demoMng.SetGroup(new List<int>(), 2);
            var sortedGroup = demoMng.GetLastAddedGroup();

            demoMng.AddGroup(new List<int>(), 3);
            var selGroup = demoMng.GetLastAddedGroup();

            demoMng.AddGroup(new List<int>(), 4);
            var maxGroup = demoMng.GetLastAddedGroup();

            for (var insertionIndex = array.Length - 1; insertionIndex > 0; insertionIndex--)
            {
                var maxIndex = 0;

                maxGroup.Collection.Add(maxIndex);
                Demonstrate(demoMng.E, "Максимум");

                selGroup.Collection.Add(-1);

                for (var comparableIndex = 1; comparableIndex <= insertionIndex; comparableIndex++)
                {
                    selGroup.Collection[0] = comparableIndex;
                    Demonstrate(demoMng.E, "Сравнение");

                    if (array[comparableIndex] > array[maxIndex])
                    {
                        maxIndex = comparableIndex;

                        maxGroup.Collection[0] = maxIndex;
                        Demonstrate(demoMng.E, "Максимум");
                    }
                }

                var max = array[maxIndex];
                array[maxIndex] = array[insertionIndex];
                array[insertionIndex] = max;

                selGroup.Collection.Clear();
                maxGroup.Collection.Clear();
                demoMng.SelectSwapped(maxIndex, insertionIndex);
                Demonstrate(demoMng.E, "Обмен");
                demoMng.UnselectSwapped();

                sortedGroup.Collection.Add(insertionIndex);
            }
        }
    }
}

```

```

    }
  }
}

```

## Проект «ShellSort»

### Shell.cs

```

using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;
using System.Text;
using SortingExtensionAPI;

namespace Shell
{
    public abstract class ShellBase : ArraySorter
    {
        public override string Name { get => String.Format("Шелла ({0})", _note); }
        protected string _note;

        protected abstract IEnumerable<int> _CreateSteps(int arrayLength);

        public override TimeSpan MeasureSortTime (int[] array)
        {
            var sw = new Stopwatch();
            sw.Start();

            var steps = _CreateSteps(array.Length);
            foreach (var step in steps)
            {
                for (var subArrayNo = 0; subArrayNo < step; subArrayNo += 1)
                {
                    for (var placeableIndex = subArrayNo + step;
                        placeableIndex < array.Length;
                        placeableIndex += step)
                    {
                        var placeable = array[placeableIndex];
                        var comparableIndex = placeableIndex - step;
                        while (comparableIndex >= subArrayNo)
                        {
                            if (array[comparableIndex] > placeable)
                            {
                                array[comparableIndex + step] = array[comparableIndex];
                                comparableIndex -= step;
                            }
                            else
                            {
                                break;
                            }
                        }
                        array[comparableIndex + step] = placeable;
                    }
                }
            }

            sw.Stop();
            return sw.Elapsed;
        }

        public override void SortArrayDemonstration(int[] array)
        {
            var demoMng = new DemonstrationManager(array);

            var steps = _CreateSteps(array.Length);
            foreach (var step in steps)
            {
                for (var subArrayNo = 0; subArrayNo < step; subArrayNo += 1)
                {
                    var subArray = new List<int>();
                    for (var i = subArrayNo; i < array.Length; i += step)
                    {
                        subArray.Add(i);
                    }
                    demoMng.SetGroup(subArray, 3);
                }
            }
        }
    }
}

```



```

{
    public class Wirth : ShellBase
    {
        public Wirth()
        {
            _note = "шаг Вирта";
        }
        protected override IEnumerable<int> _CreateSteps(int arrayLength)
        {
            var stack = new Stack<int>();
            var step = 1;
            do
            {
                stack.Push(step);
                step = step * 2 + 1;
            } while (step < arrayLength);
            if (stack.Peek() != 1)
                stack.Pop();
            return stack;
        }
    }

    public class Knuth : ShellBase
    {
        public Knuth()
        {
            _note = "шаг Кнута";
        }
        protected override IEnumerable<int> _CreateSteps(int arrayLength)
        {
            var stack = new Stack<int>();
            var step = 1;
            do
            {
                stack.Push(step);
                step = step * 3 + 1;
            } while (step < arrayLength);
            if (stack.Peek() != 1)
                stack.Pop();
            return stack;
        }
    }

    public class Shell : ShellBase
    {
        public Shell()
        {
            _note = "шаг Шелла";
        }
        protected override IEnumerable<int> _CreateSteps(int arrayLength)
        {
            var queue = new Queue<int>();
            var step = arrayLength / 2;
            do
            {
                queue.Enqueue(step);
                step /= 2;
            } while (step > 0);
            return queue;
        }
    }

    public class Sedgwick : ShellBase
    {
        public Sedgwick()
        {
            _note = "шаг Седжвика";
        }
        protected override IEnumerable<int> _CreateSteps(int arrayLength)
        {
            var stack = new Stack<int>();
            var i = 1;
            var step = 1;
            do
            {
                stack.Push(step);
                step = (8 << i) - (6 << (++i >> 1)) + 1;
                if (step * 3 <= arrayLength)
            {

```

```

        stack.Push(step);
        step = (9 << i) - (9 << (i++ >> 1)) + 1;
    }
    else break;
}
while (step * 3 <= arrayLength);
return stack;
}
}
}
}
}

```

Приложение Б

СКРИНШОТЫ ИССЛЕДОВАНИЯ ВРЕМЕННЫХ  
ХАРАКТЕРИСТИК АЛГОРИТМОВ СОРТИРОВКИ ПРИ  
ПОМОЩИ РАЗРАБОТАННОЙ ПРОГРАММЫ  
(рекомендуемое)

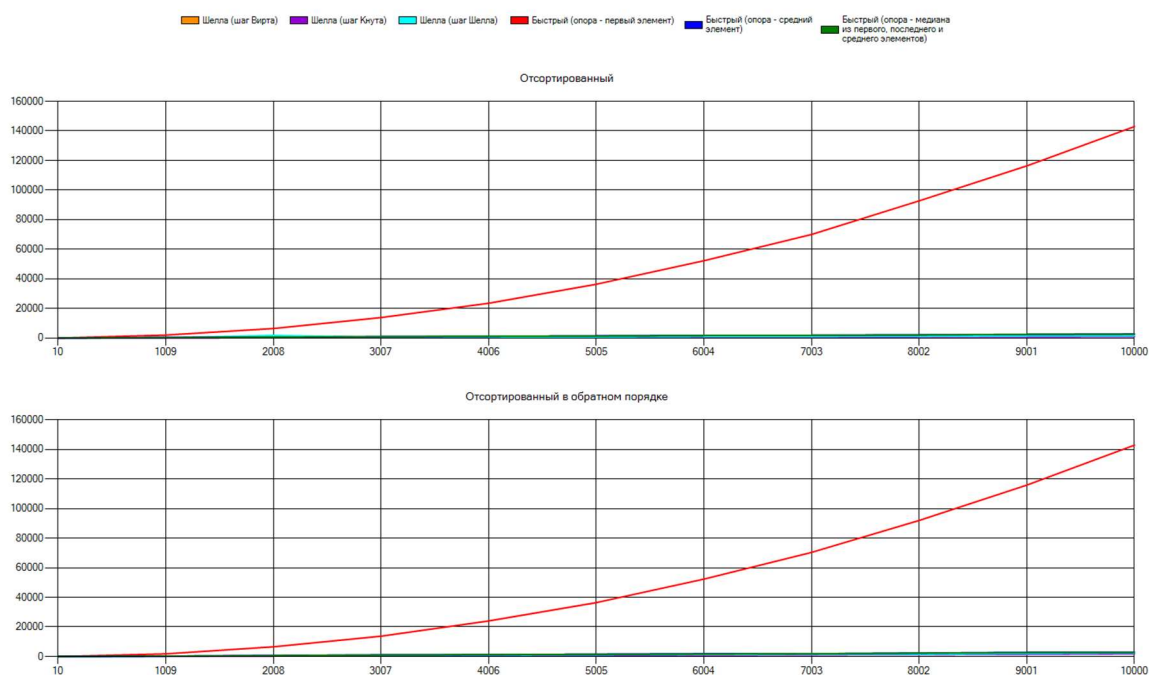


Рисунок Б.1 – Сравнение эффективности сортировок Шелла и быстрых сортировок на упорядоченных массивах

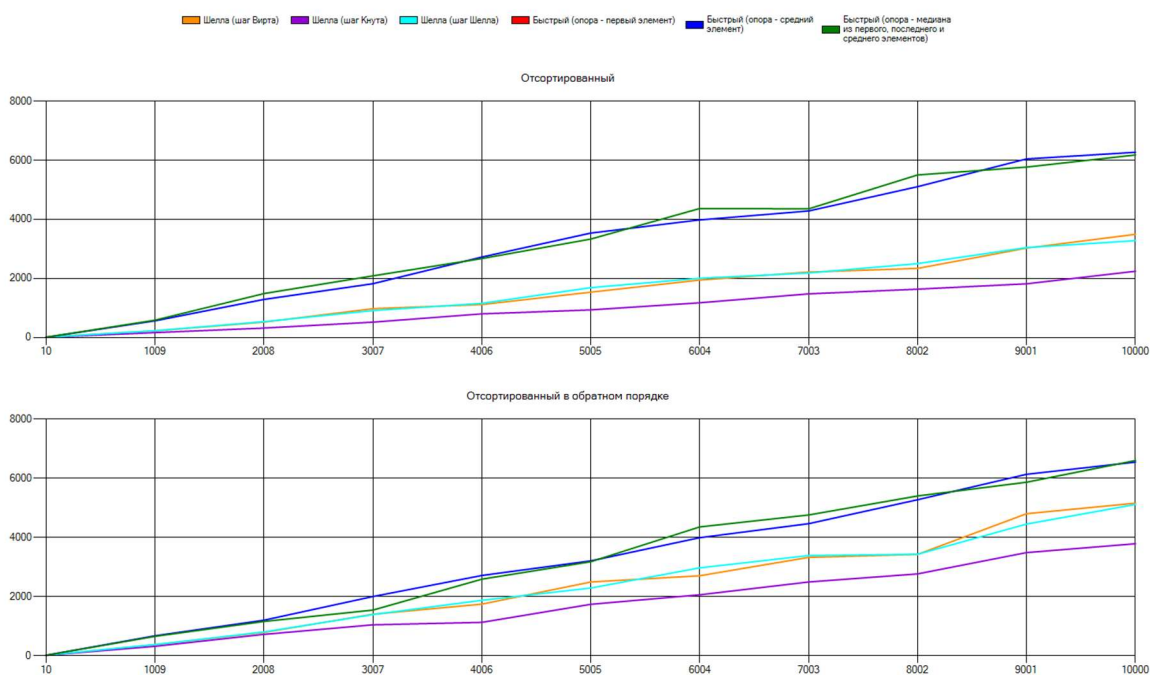


Рисунок Б.2 – Сравнение эффективности сортировок Шелла и быстрых сортировок на упорядоченных массивах (кроме быстрой сортировки с опорой в первом элементе)

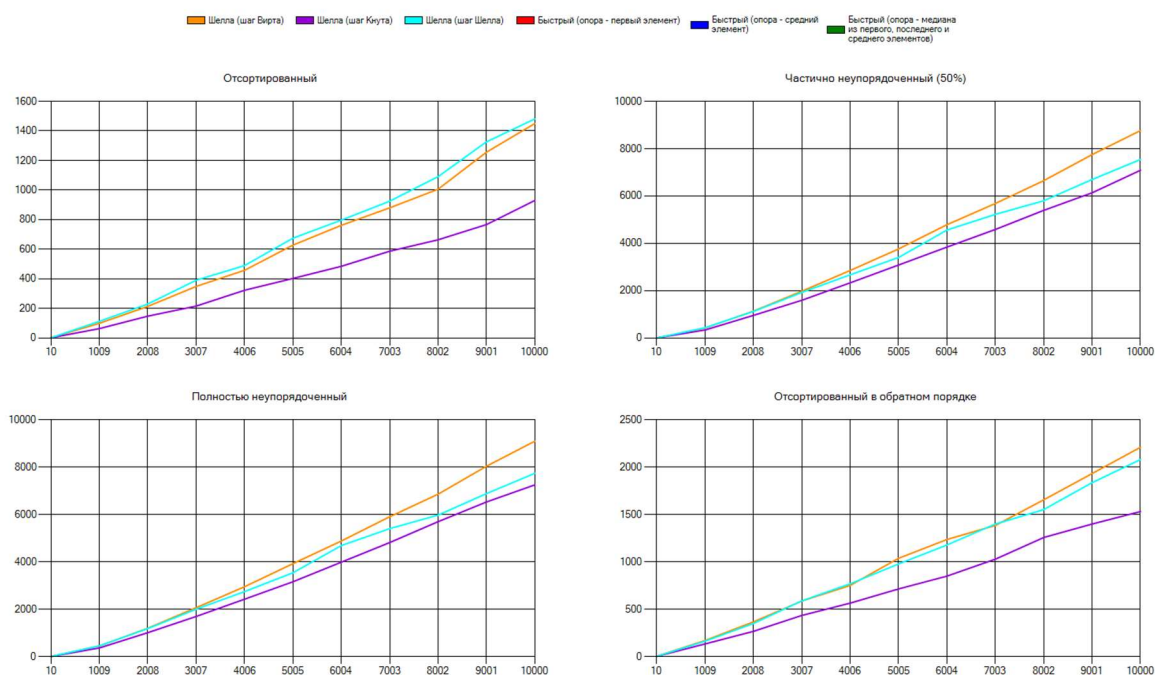


Рисунок Б.3 – Сравнение вариаций сортировки Шелла на некоторых типах массивов

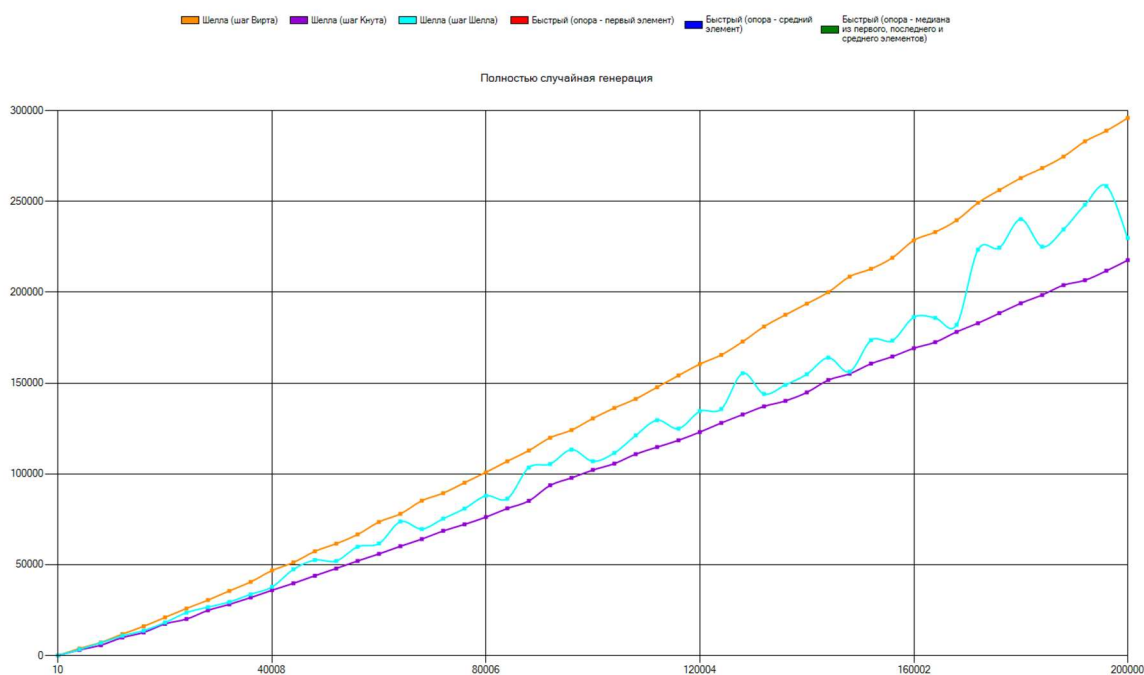


Рисунок Б.4 – Сравнение вариаций сортировки Шелла на полностью случайных массивах

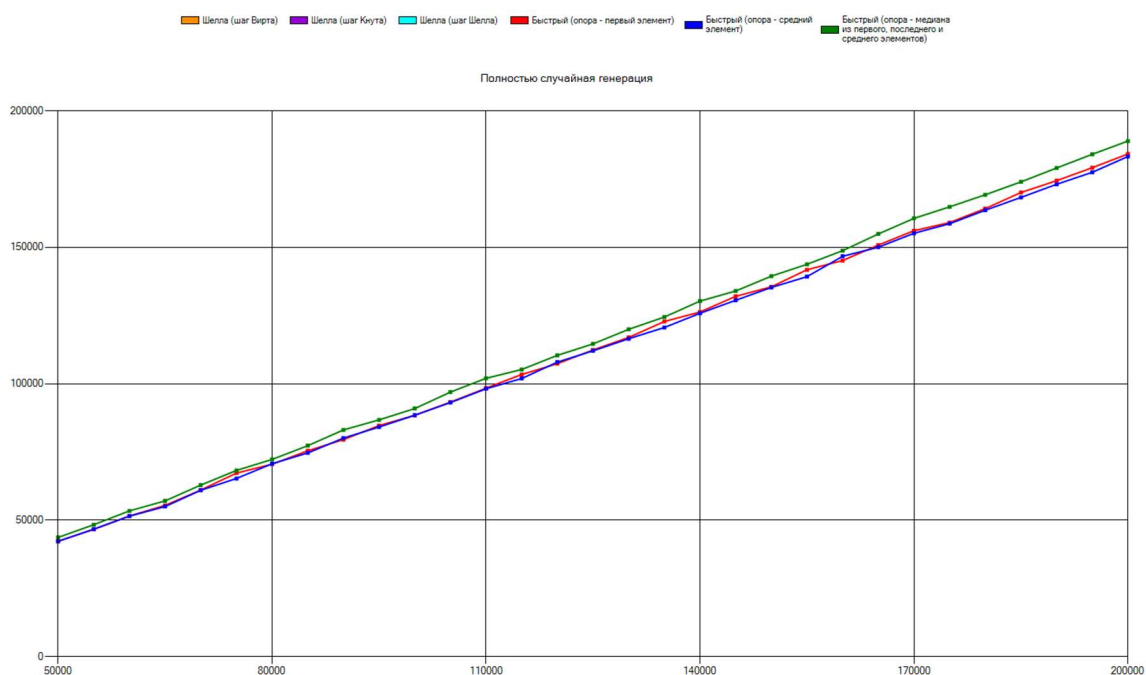


Рисунок Б.5 – Сравнение вариаций быстрой сортировки на полностью случайных массивах

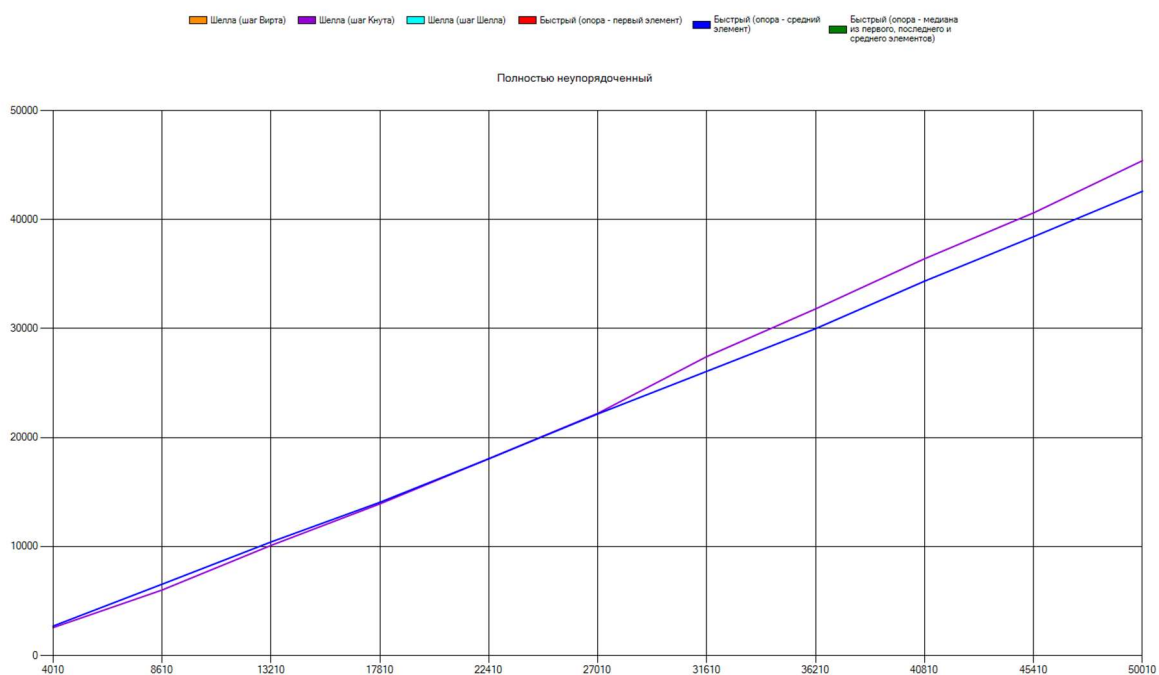


Рисунок Б.6 – Сравнение сортировки Шелла и быстрой сортировки на полностью неупорядоченных массивах

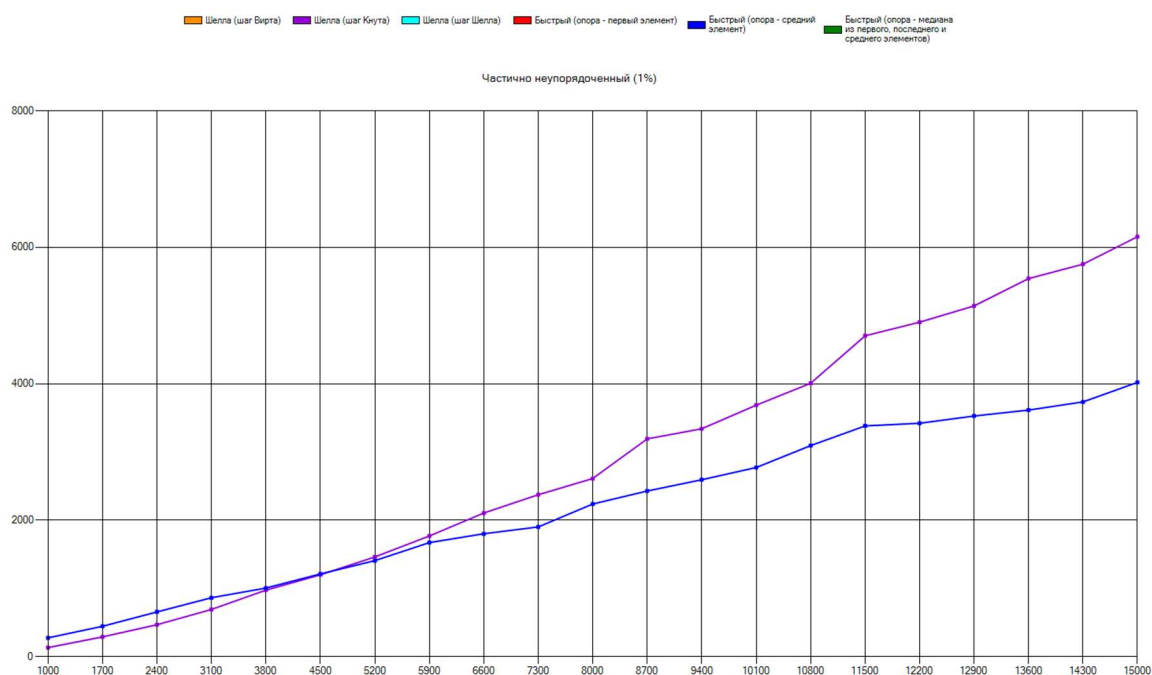


Рисунок Б.7 – Сравнение сортировки Шелла и быстрой сортировки на частично (1%) неупорядоченных массивах

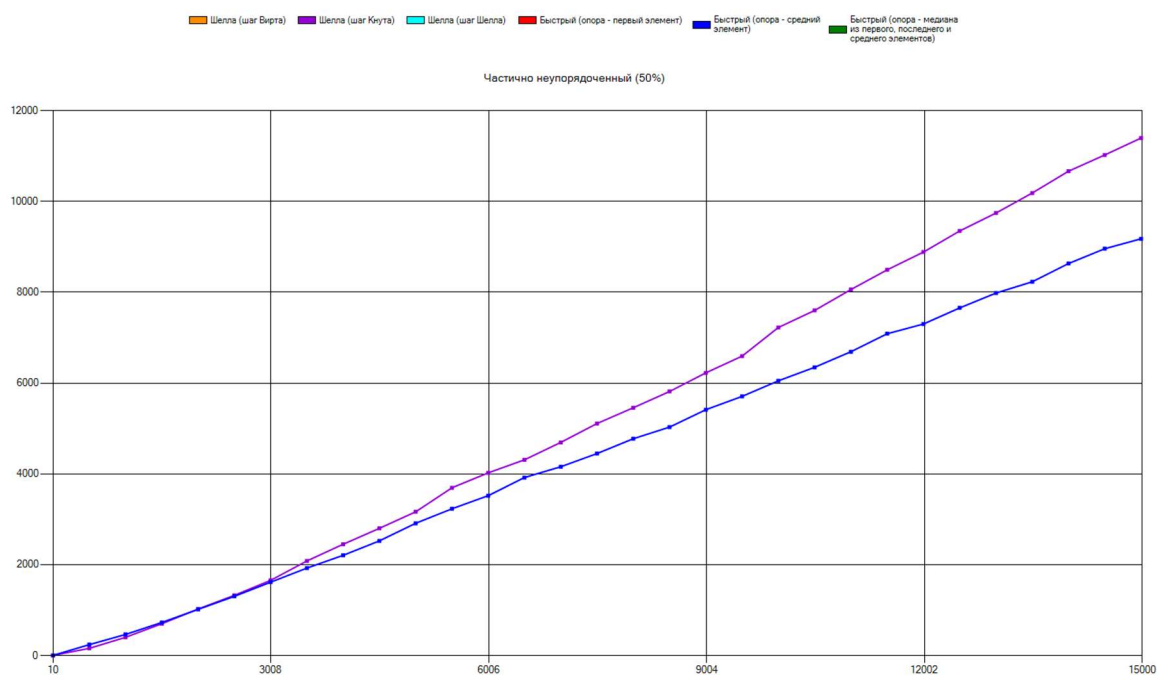


Рисунок Б.8 – Сравнение сортировки Шелла и быстрой сортировки на частично (50%) неупорядоченных массивах

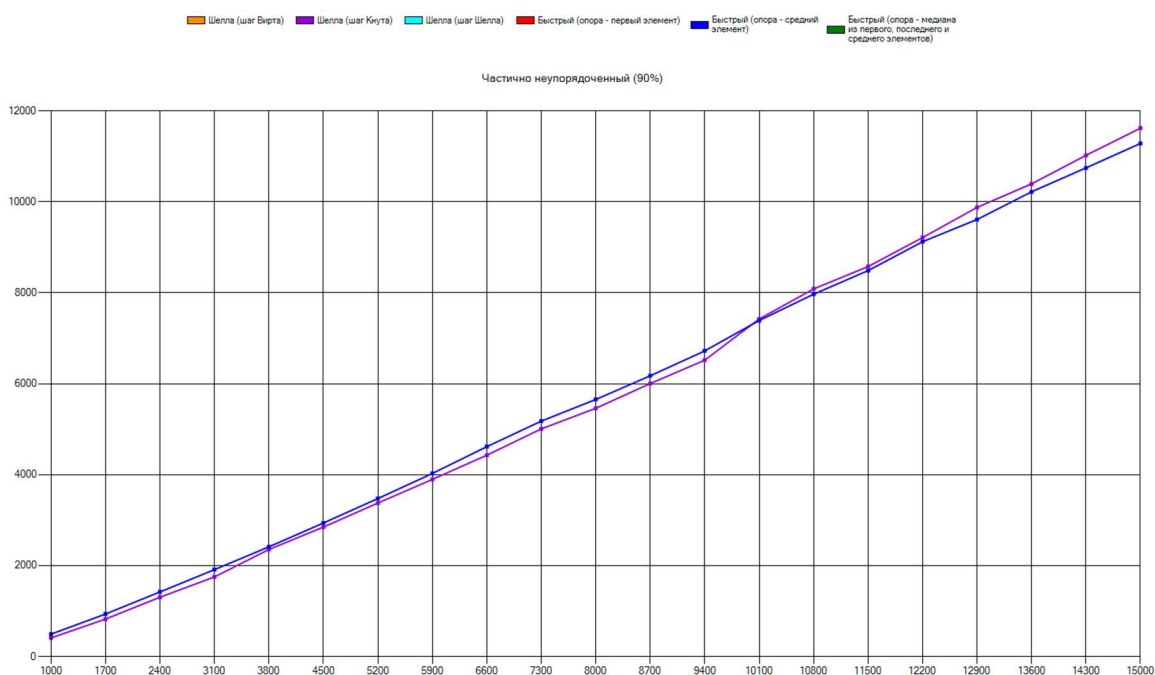


Рисунок Б.9 – Сравнение сортировки Шелла и быстрой сортировки на частично (90%) неупорядоченных массивах

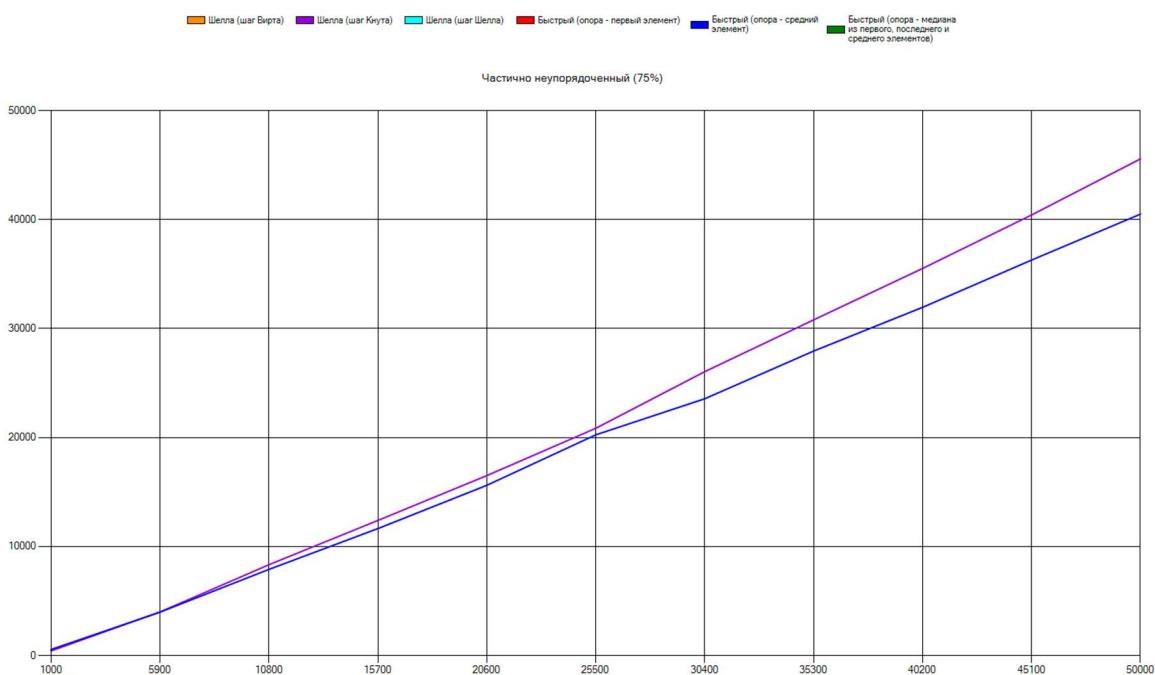


Рисунок Б.10 – Сравнение сортировки Шелла и быстрой сортировки на частично (75%) неупорядоченных массивах

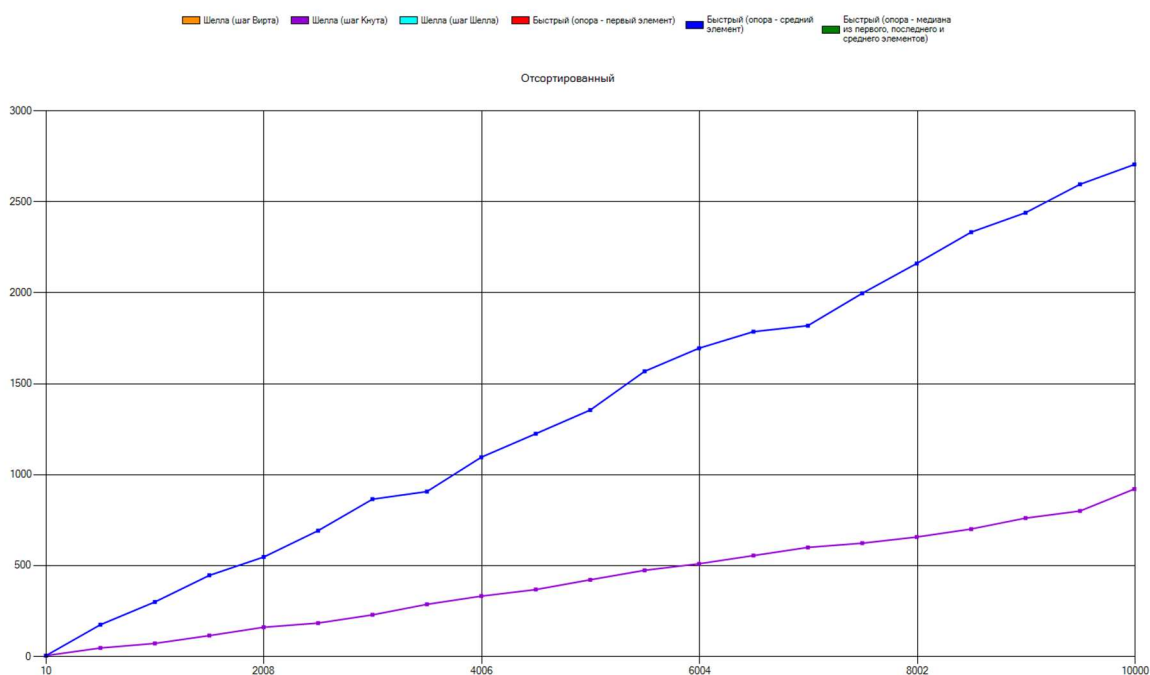


Рисунок Б.11 – Сравнение сортировки Шелла и быстрой сортировки на упорядоченных массивах

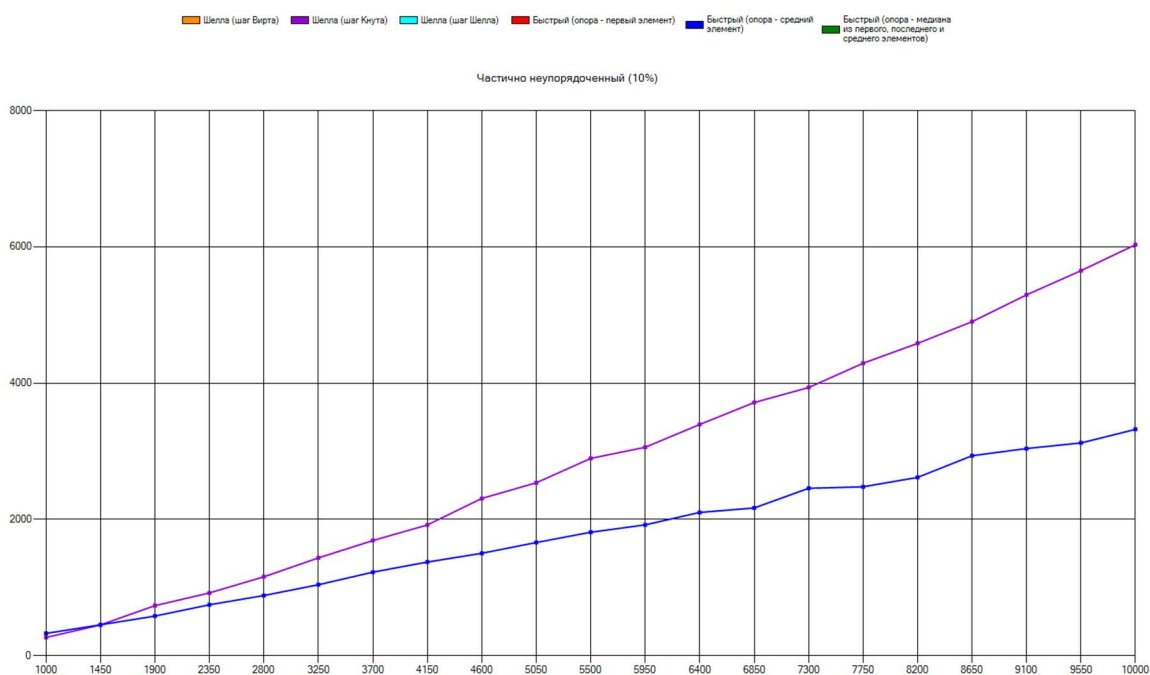


Рисунок Б.12 – Сравнение сортировки Шелла и быстрой сортировки на частично (10%) неупорядоченных массивах

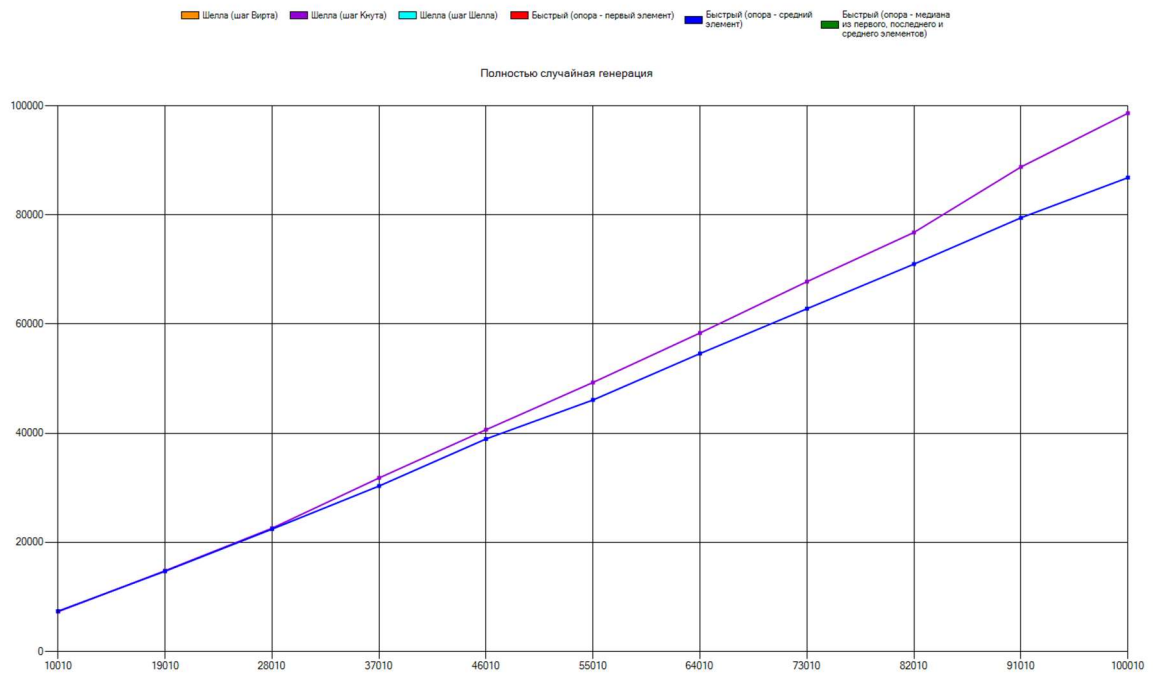


Рисунок Б.13 – Сравнение сортировки Шелла и быстрой сортировки на полностью случайных массивах