МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Утвержден на заседании кафедры

«Вычислительная техника»_____
"___" _____ 20 г.
Заведующий кафедрой
_____М.А. Митрохин

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ: ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

(2024/2025 учебный год)

_____ Гераськин Денис Владимирович

Направление подготовки (специальность) 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»
Наименование профиля подготовки (специализация) «Эксплуатация вычислительных машин, комплексов, систем и сетей»
Форма обучения – очная Срок обучения в соответствии с ФГОС – 5 лет

Год обучения _____1_____семестр _____2_____

Период прохождения практики с 20.06.2025 по 17.07.2025

Кафедра «Вычислительная техника»

Заведующий кафедрой д.т.н., Митрохин М.А.
(должность, ученая степень, ученое звание, Ф.И.О.)

Руководитель практики к.т.н., доцент, Карамышева Н.С.
(должность, ученая степень, ученое звание)

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Утвержден на заседании кафедры

«Вычислительная техника»_____
"___" _____ 20 г.
Заведующий кафедрой
_____М.А. Митрохин

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ: ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

(2024/2025 учебный год)

_____Гераськин Денис Владимирович

Направление подготовки (специальность) 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»
Наименование профиля подготовки (специализация) «Эксплуатация вычислительных машин, комплексов, систем и сетей»
Форма обучения – очная Срок обучения в соответствии с ФГОС – 5 лет

Год обучения _____1_____семестр _____2_____

Период прохождения практики с 20.06.2025 по 17.07.2025

Кафедра «Вычислительная техника»

Заведующий кафедрой д.т.н., Митрохин М.А.
(должность, ученая степень, ученое звание, Ф.И.О.)

Руководитель практики к.т.н., доцент, Карамышева Н.С.
(должность, ученая степень, ученое звание

№ п/п	Планируемая форма работы во время практики	Количество часов	Календарные сроки проведения работы	Подпись руководителя практики от вуза
1	Выбор темы и разработка индивидуального плана проведения работ	26	20.06.25 – 24.06.25	
2	Подбор и изучение материала по теме работы	26	24.06.25 – 26.06.25	
3	Разработка алгоритма	26	26.06.25 – 28.06.25	
4	Описание алгоритма и программы	30	04.07.25 – 09.07.25	
5	Тестирование	30	09.07.25 – 12.07.25	
6	Получение и анализ результатов	38	12.07.25 – 14.07.25	
7	Оформление отчёта	40	14.07.25 – 17.07.25	
	Общий объём часов	216		

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ОТЧЁТ

О ПРОХОЖДЕНИИ УЧЕБНОЙ (ОЗНАКОМИТЕЛЬНОЙ) ПРАКТИКИ

(2024/2025 учебный год)

Гераськин Денис Владимирович

Направление подготовки (специальность) 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»
Наименование профиля подготовки (специализация) «Эксплуатация вычислительных машин, комплексов, систем и сетей»
Форма обучения – очная Срок обучения в соответствии с ФГОС – 5 года

Год обучения 1 семестр 2

Период прохождения практики с 20.06.2025 по 17.07.2025

Кафедра «Вычислительная техника»

Гераськин Д.В. выполнял практическое задание «Сортировка пузырьком». На первоначальном этапе были изучен и проанализирован алгоритм пузырьковой сортировки, был выбран метод решения и язык программирования С, на котором была написана программа по реализации сортировки массива методом пузырька, также, осуществлена реализация работы с файлами.

Специалист Гераськин Д.В. _____ "___" _____ 2025 г.

Руководитель практики Карамышева Н.С._____ "___" _____ 2025 г.

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ
ФАКУЛЬТЕТ ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

ОТЗЫВ

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ: ОЗНАКОМИТЕЛЬНОЙ ПРАКТИКЕ

(2024/2025 учебный год)

_____Гераськин Денис Владимирович

Направление подготовки (специальность) 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения»
Наименование профиля подготовки (специализация) «Эксплуатация вычислительных машин, комплексов, систем и сетей»
Форма обучения – очная Срок обучения в соответствии с ФГОС – 5 лет

Год обучения _____1_____семестр _____2_____

Период прохождения практики с 20.06.2025 по 17.07.2025

Кафедра «Вычислительная техника»

В процессе выполнения практики Гераськин Д.В. решал следующие задачи: разработка алгоритма быстрой сортировки, анализ результатов сортировки, сравнение существующих методов с разработанным алгоритмом.

За период прохождения практики были освоены основные понятия и технологии программирования, изучены основные инструменты языков C/C++. И получены следующие результаты: разработан алгоритм быстрой сортировки, получены результаты работы алгоритма, сделаны выводы эффективности работы алгоритма быстрой сортировки. Во время выполнения работы Гераськин Д.В. показал себя ответственным, добросовестным учеником, знающим свой предмет, имеющим представление о современном состоянии науки, владеющим современными общенаучными знаниями по информатике и вычислительной технике; программированию.

За выполнение работы Гераськин Д.В. заслуживает оценки «_____».
Руководитель практики к.т.н., Карамышева Н.С. .« »_____2025 г.

Содержание

Введение

В наше время сортировка данных играет очень важную роль и является одним из главных процессов обработки данных на текущий момент. В контексте профессиональной деятельности очень часто можно встретить задачу, которая подразумевает сортировку данных различными способами и методами.

Самих методов и алгоритмов сортировки достаточно много. Применяются они с целью осуществления ускорения и упрощения поиска данных.

Сама важность и необходимость сортировки заключается в том, что на её примере можно достаточно наглядно показать принцип работы определённых алгоритмов. При всём этом, алгоритмов сортировки очень много и каждый имеет свои преимущества и свои недостатки относительно друг друга. Так, в определённых ситуациях можно применить простой алгоритм и этого будет более чем достаточно, а иногда нужно применить достаточно сложный алгоритм в достижении нужного результата. Сама цель сортировки заключается в перестановке некоторого количества объектов в определённом порядке по определённым правилам.

Сортировка пузырьком — это один из квадратичных алгоритмов сортировки. Основное преимущество этого метода заключается в простоте его реализации и минимальных требованиях к памяти. Но данный метод имеет недостаток, который виден при работе с большими массивами и объёмами данных, при этом будет наблюдаться низкая скорость работы алгоритма. Такая сортировка очень хороший пример для обучения сортировке, но не является оптимальным методом в реальных приложениях с большим объёмом данных.

1 Постановка задачи

Поставленная задача: необходимо заполнить массив из *n*-ого количества элементов случайными числами, записать данные элементы в отдельный файл. После этого выполнить сортировку методом пузырька над этими данными, которые находятся в массиве, записать отсортированные данные в другой файл, затем посчитать время выполнения и количество перестановок значений массива при сортировке.

Использовать сервис *GitHub* для совместной работы. Создать и выложить коммиты, характеризующие действия, выполненные каждым участником бригады.

Оформить отчет по проведенной практике.

1.1 Достоинства алгоритма сортировки пузырьком

- не используется дополнительная память;
- для маленьких массивов предпочтительнее, чем другие алгоритмы;
- легкость реализации. Алгоритм очень прост для понимания и реализации на любом языке программирования.

1.2 Недостатки алгоритма сортировки пузырьком

- низкая эффективность. Сложность алгоритма составляет $O(n^2)$, что делает его крайне медленным для работы с большими массивами данных;
- ресурсоёмкий по числу операций;
- для заранее отсортированных данных все равно выполняется несколько лишних проходов, что делает алгоритм не эффективным в реальных условиях.

1.3 Типичные сценарии применения данного алгоритма

- обучение основам алгоритмизации и программирования в образовательных целях;
- стандартная ситуация в быте (раскладка книг, документов или файлов по алфавиту или дате);
- создание наглядных анимаций, которые демонстрируют принципы работы сортировки (например, в визуализаторах алгоритмов или обучающих роликах).

2 Выбор решения

Для написания данной программы будет использован язык программирования Си. Этот язык является распространённым языком программирования. При разработке языка Си был принят компромисс между низким уровнем языка ассемблера и высоким уровнем других языков. Си – это язык программирования общего назначения, хорошо известный своей эффективностью, экономичностью и переносимостью. Указанные преимущества Си обеспечивают хорошее качество разработки почти любого вида программного продукта.

В качестве среды программирования была выбрана программа *Microsoft Visual Studio Code*. *Microsoft Visual Studio Code* — это редактор кода для разных языков программирования.

Для удобства совместной разработки был использован сервис *WEEEK*. *WEEEK* — сервис для управления личными и командными проектами. В основе *WEEEK* лежит недельный планер и канбан-методология: доски, колонки и т. д. Проект динамично разрабатывается, регулярно расширяя функционал и возможности. Ведется активная работа с пожеланиями пользователей в еженедельном патчноте *WEEEK*.

3 Описание программы

При запуске программы пользователю выводится меню с действиями:

a) 1. Сгенерировать числа и записать в файл;

b) 2. Отсортировать числа из файла;

c) 3. Проверить сортировку чисел;

d) 4. Сохранить отсортированные числа;

e) 0. Выход.

Пользователю требуется выбрать тот пункт, который ему требуется. При выборе пунктов числа 1-4 и 0 являются ключами к действиям, т.е нажимая одну из цифр на клавиатуре пользователь выберет нужное ему действие и после выбора определённого пункта, выводится сообщение, в котором пользователю необходимо ввести определенные данные.

После того как пользователь ввёл название файла, количество чисел и диапазон значений, массив с числами генерируется в файл (в данном проекте файл может быть txt или csv). Название файла и расширение пользователь задаёт сам. После эти данные читаются из файла.

Затем этим данные сортируются методом пузырька по возрастанию или по убыванию. Происходит это путём сравнивания соседних элементов. Если текущий элемент больше чем следующий стоящий, то происходит перестановка, т.е текущий встаёт на место следующего, а тот что был впереди становится сзади, т.о. самые большие элементы «всплывают» в конец массива. Действия повторяются пока массив не будет отсортирован.

Далее пользователь может проверить, а отсортирован ли массив, текстовые сообщения в консоли дадут ему понять, в каком состоянии данные. После этого можно сохранить отсортированный массив в файл, который пользователь также сам задаёт.

На цифру 0 можно завершить выполнение программы.

Подробный алгоритм будет приложен в подразделе 4.1 в виде рисунка.

3.1 Описание личного вклада в проект

Моя задача заключалась в тестировании и анализе работы программы, а также в фиксации количества операций и времени выполнения в разных сценах.

Были созданы файлы с разным количеством чисел (от 10000 до 90000) и диапазоном (от -90000 до 90000), отсортированы и сохранены в файлы. Была проведена проверка на количество вводимых чисел в файл (максимум 100000).

Результаты тестов (количество чисел, диапазон генерации, время выполнения и количество операций) приведены в таблице 1 и приложении А.

Также мною был добавлен счетчик количества операций сортировки (`*swap_count)++`) и была написана функция *rev_bubble_sort*, которая проходит по всем числам сортируя массив из введенных чисел по убыванию.

```
int tmp = array[j];
array[j] = array[j + 1];
array[j + 1] = tmp;
(*swap_count)++;
```

Также для случая, когда пользователь мог забыть нажать кнопку сортировки предусмотрена функция *check1,* которая проверяет отсортирован ли массив по убыванию.

4 Схемы программы

4.1 Блок-схема алгоритма

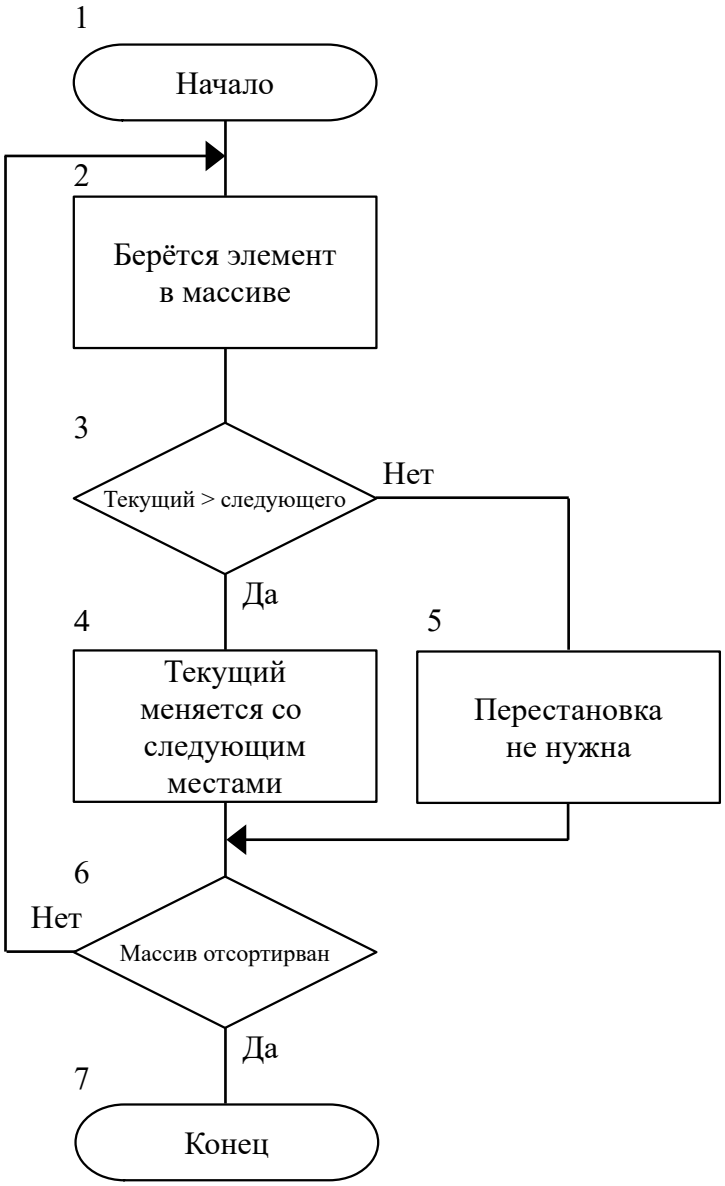
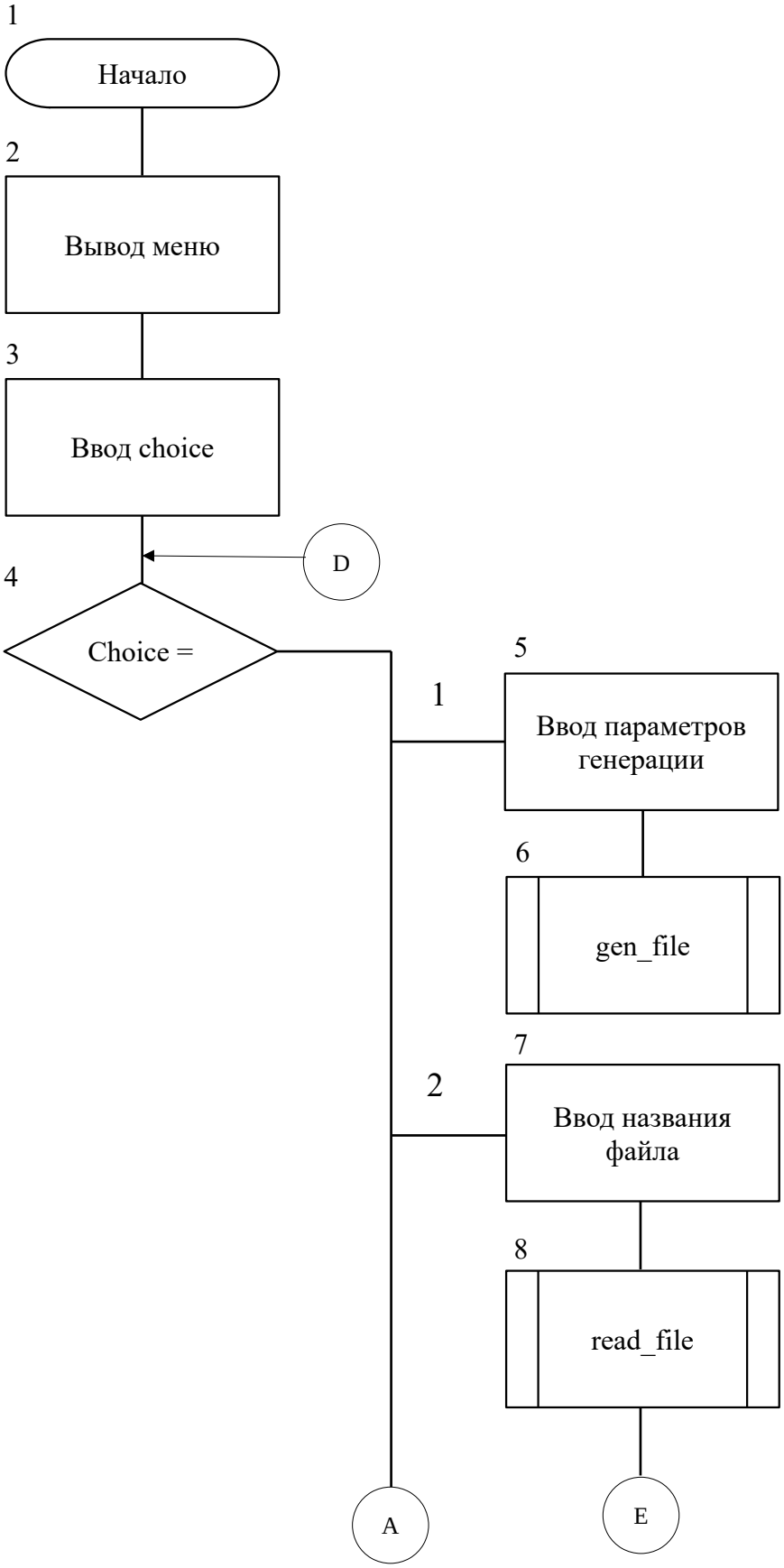
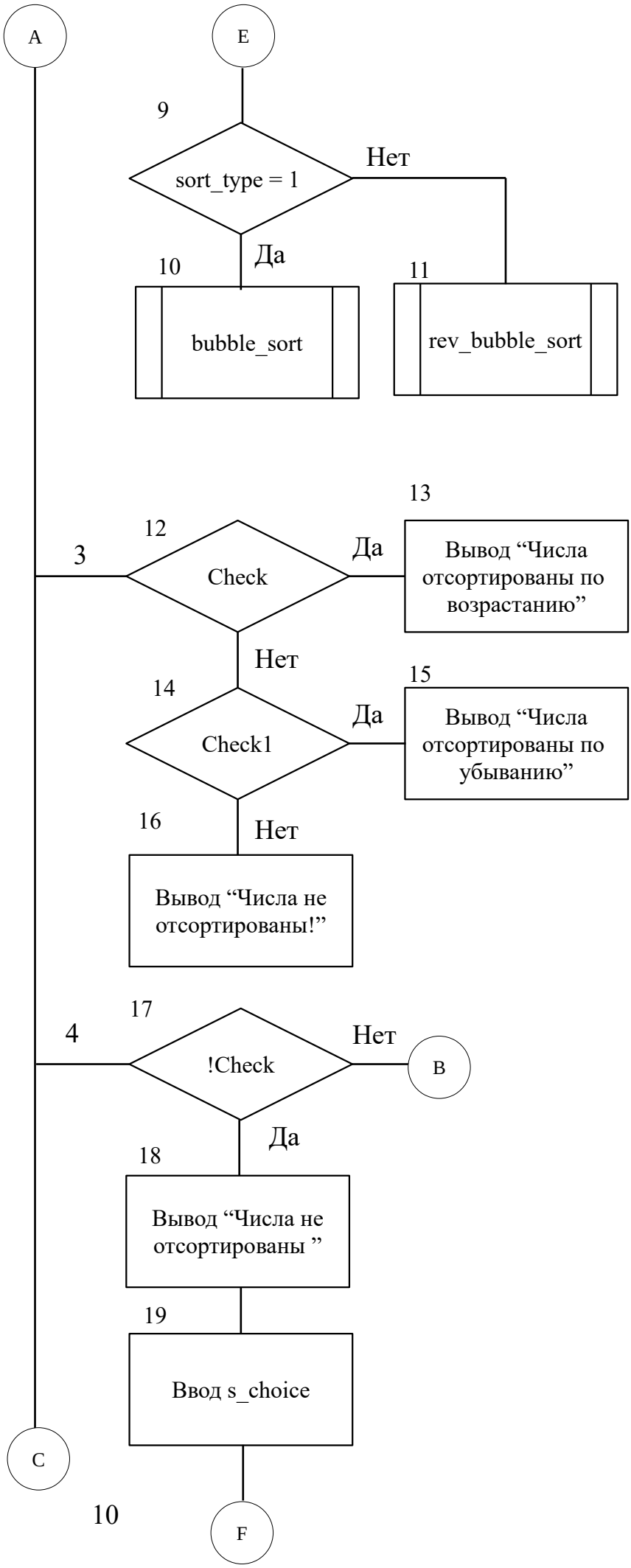


Рисунок 1 — Блок-схема алгоритма

4.2 Блок-схемы программы




A
E
9
sort_type = 1
Нет
Да
10
bubble_sort
11
rev_bubble_sort
13
3
12
Check
Да
Вывод “Числа отсортированы по возрастанию”
Нет
15
14
Check1
Да
Вывод “Числа отсортированы по убыванию”
16
Нет
Вывод “Числа не отсортированы!”
17
4
!Check
Нет
B
Да
18
Вывод “Числа не отсортированы ”
19
Ввод s_choice
C
F

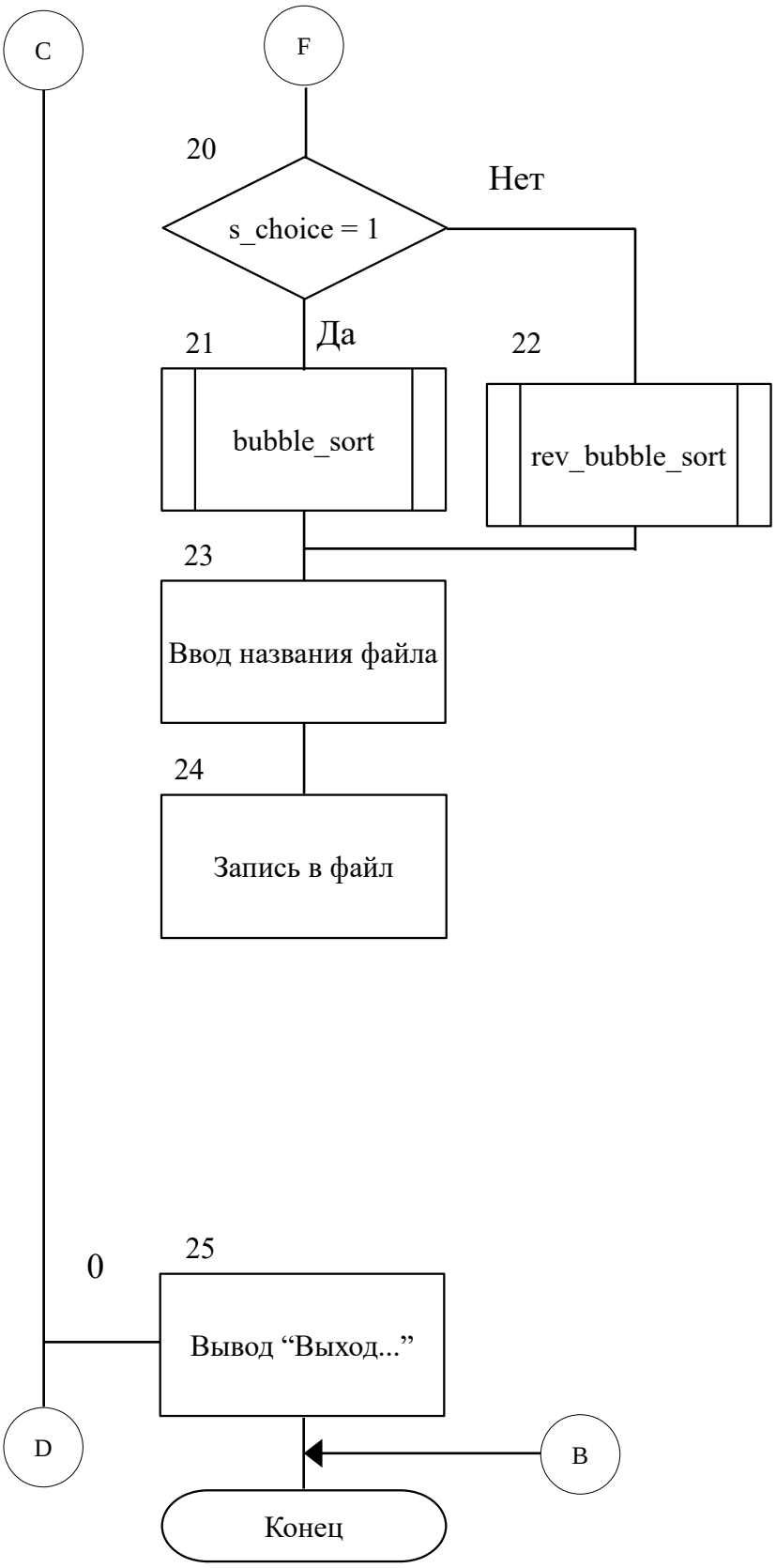


Рисунок 2 — Блок-схема функции *main*

5 Тестирование программы

5.1 Тестирование на разных наборах данных

Тестовый набор данных представлен в таблице 1. Результаты тестирования приведены в Приложении А на рисунках А.1 — А.9.

Таблица 1 — Тестовый набор данных

	Размер массива	Время выполнения в секундах	Кол-во перестановок
1	10000	0.2650	50177655
2	20000	0.5720	99024468
3	30000	1.5080	225030561
4	40000	2.8580	399399969
5	50000	4.2990	627876185
6	60000	6.4490	901742927
7	70000	8.8070	1229572298
8	80000	11.8930	1602882399
9	90000	14.9900	2023834095

5.2 Анализ полученных результатов тестирования (анализ работы алгоритма)

На основании анализа данных, полученных в результате тестирования алгоритма сортировки пузырьком, можно сделать вывод, что с увеличением количества элементов пропорционально увеличивается время работы программы.

Рисунок 3 — Результаты тестирования

6 Отладка

В качестве среды разработки была выбрана программа *Microsoft Visual Studio Code*, которая содержит в себе необходимый функционал для разработки и отладки модулей и программы.

Для отладки программы использовались точки остановки и пошаговое выполнение кода программы, анализ содержимого локальных переменных.

Точка останова (*breakpoint*) - это специальное место в коде программы, где отладчик временно приостанавливает выполнение, чтобы разработчик мог проанализировать состояние программы, проверить значения переменных, отследить ход выполнения этой самой программы.

Проверялись фрагменты кода, с помощью размещения точек останова: в начале *main*() для контроля входа в программу; перед вызовом *gen_file*() и *read_file*(); внутри циклов сортировки *bubble_sort* и *rev_bubble_sort;* также перед проверкой результатов в функциях *check* и *check1*.

При проведении отладки использовались команды такие как: (*Step Over*) - для обхода вызовов стандартных функций; (*Step Into*) - для входа в пользовательские функции; (*Step Out*) - для выхода из текущей функции.

7 Совместная разработка

Для удобства совместной разработки был использован сервис *WEEEK*. Определили задачи проекта, назначили приоритет задачам.

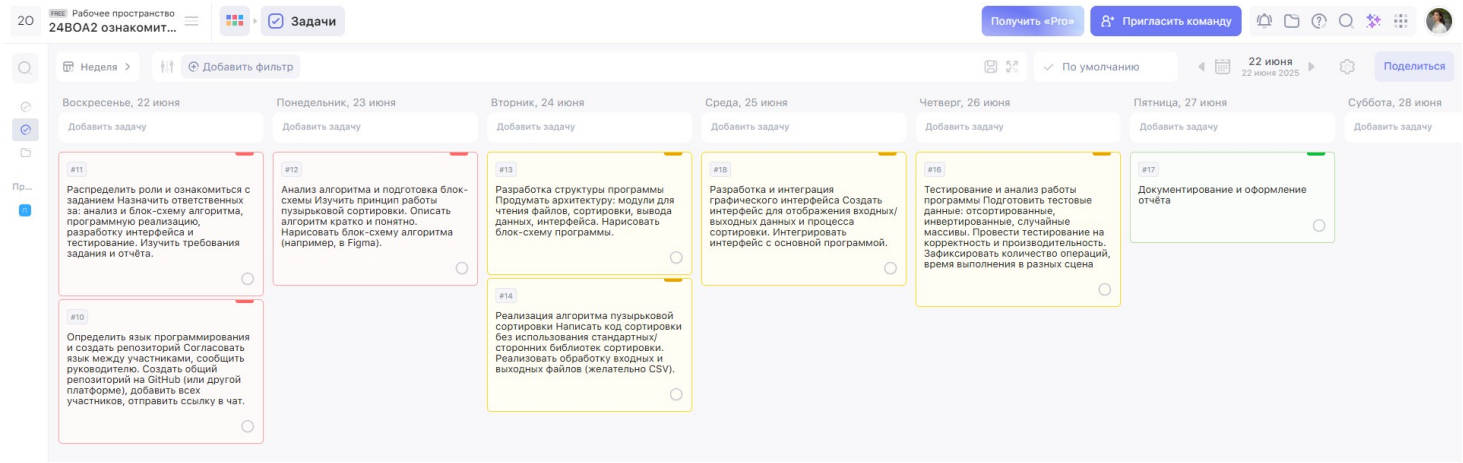


Рисунок 4 — Определение задач проекта

Затем разделили роли, назначили исполнителей задачам.

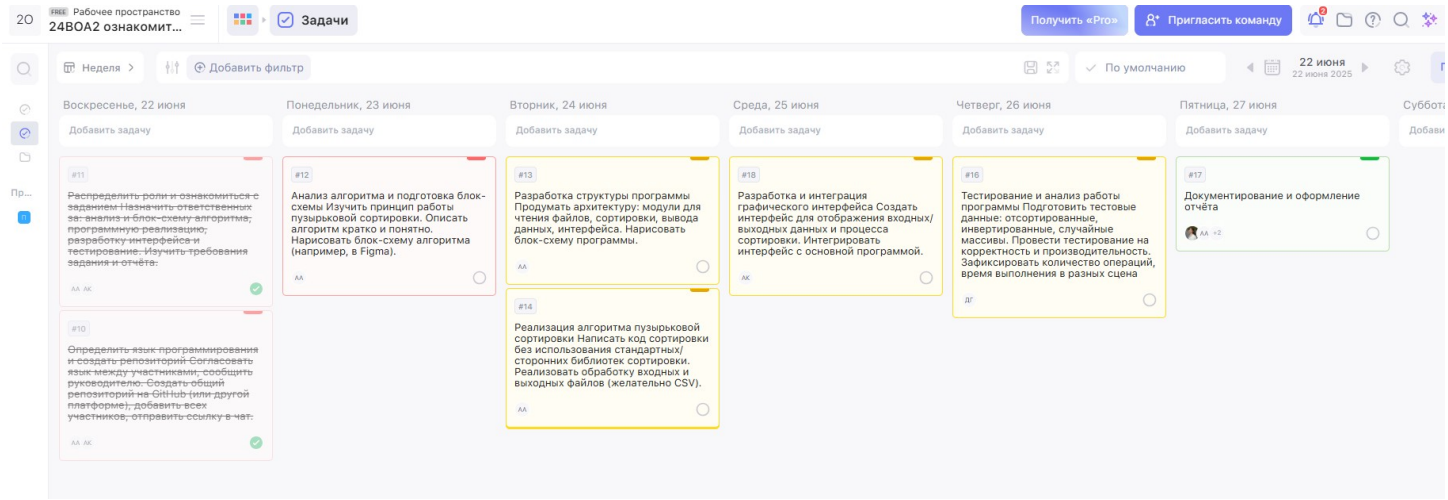


Рисунок 5 — Распределение задач проекта

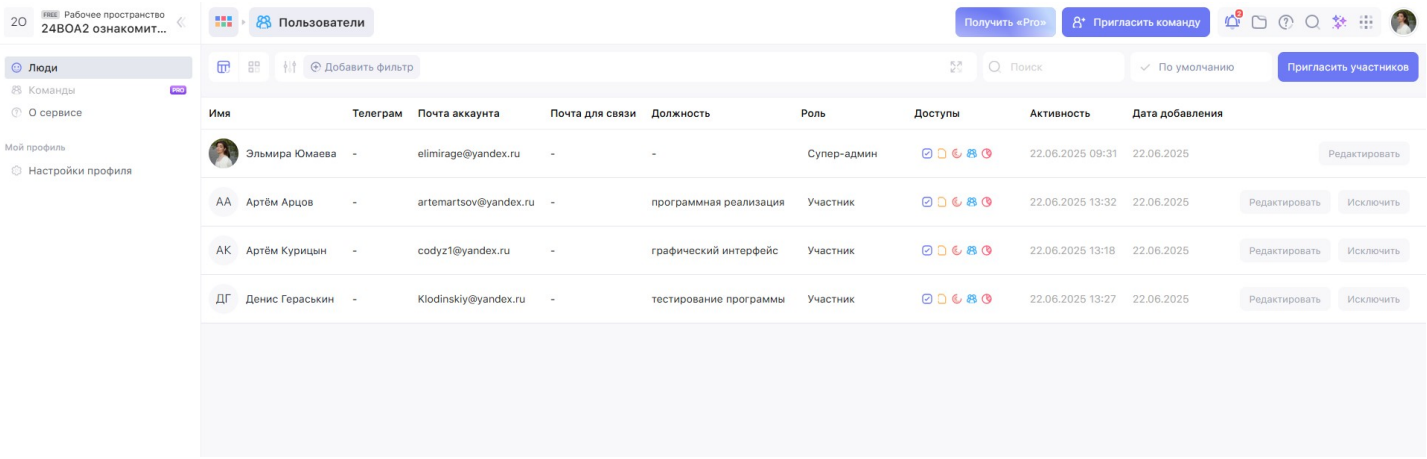


Рисунок 6 — Участники *WEEEK*

Обсуждение, выполнение и корректировка работы проводилась на площадке *Discord*.

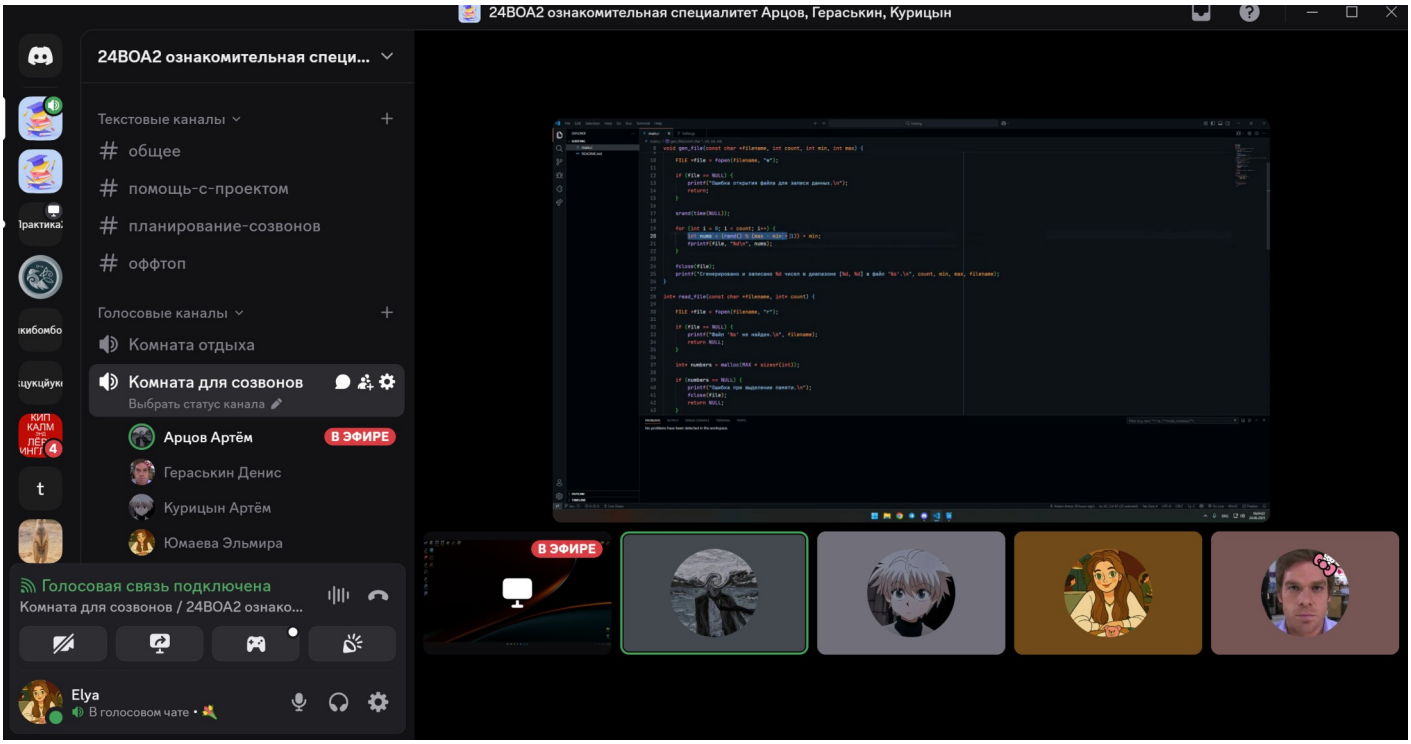


Рисунок 7 — Работа в *Discord*

Во время работы над данной практикой наша бригада осуществляла совместную работу в *GitHub*.

Мной было проведено тестирование и анализ программы, была написана функция сортировки по убыванию и был добавлен счетчик количества операций, результат был загружен на удаленный репозиторий *Github*, сначала на ветку *dev*, затем после завершения работы над программой результаты были слиты в ветку *main*.

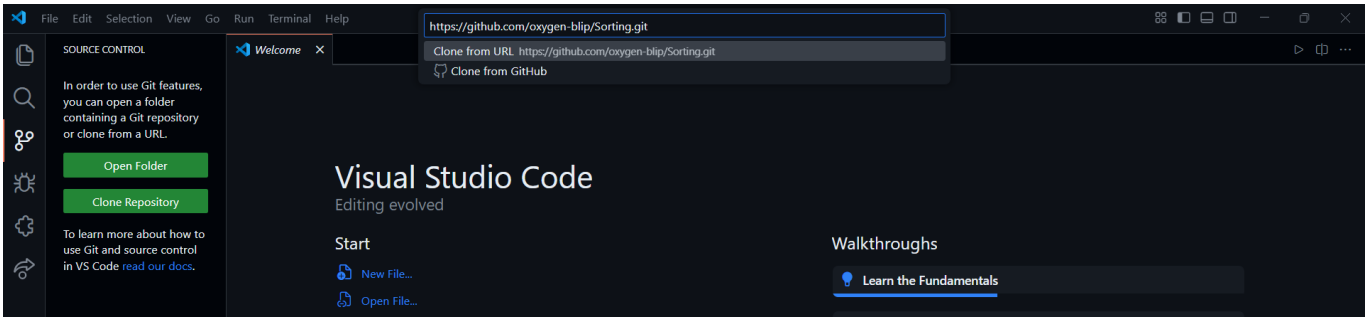


Рисунок 8 — Клонирование репозитория

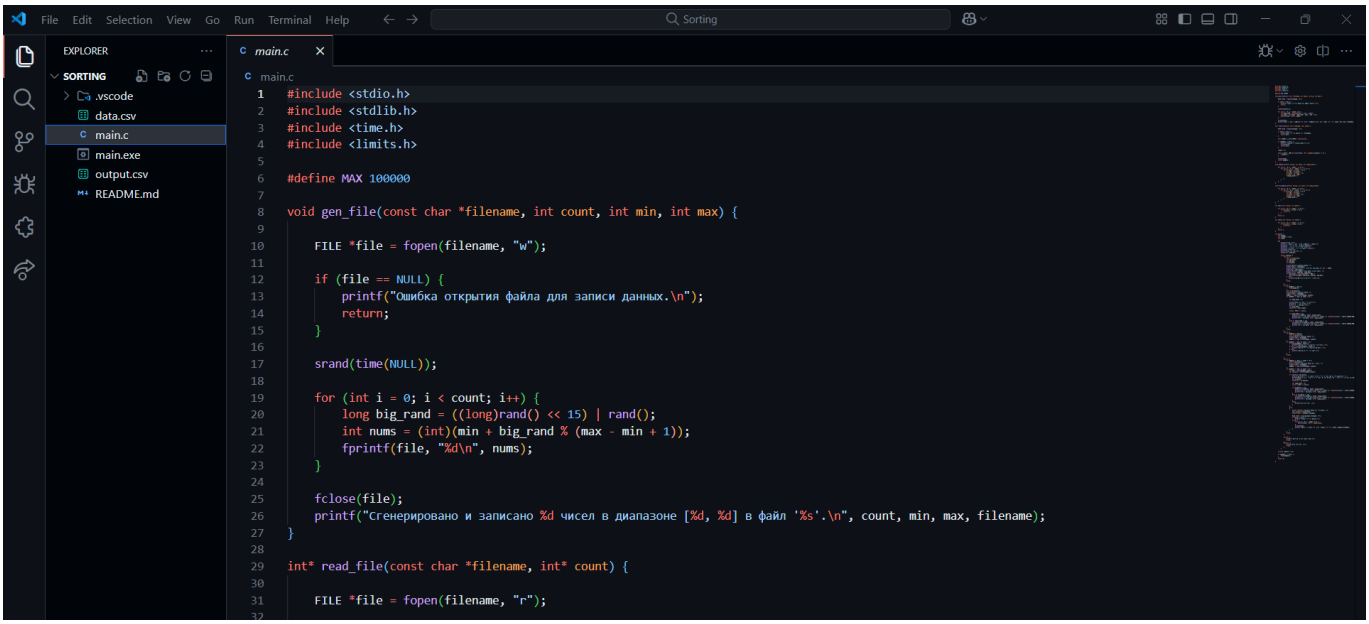


Рисунок 9 — Результат клонирования

```
commit 0d04028fd7909c63e9a2fd8f49d9020b12f43c72
Author: Klodinsky <klodinsky135@gmail.com>
Date:   Thu Jun 26 22:33:50 2025 +0300

    фикс case 4

commit 44bf88d49d9bad11329cbdb44cc5e9559cb208f4
Author: Klodinsky <klodinsky135@gmail.com>
Date:   Thu Jun 26 21:52:46 2025 +0300

    + функция, позволяющая сортировать числа от большего к меньшему;
    + функция, проверяющая выполнения этой функции сортировки;
    + возможность выбора типа сортировки (по возрастанию, по убыванию)

commit 25fd47ec0e5fde67ace9964012f944b8c60667dd
Author: ArtemPGU <hgffvhhytfvb@gmail.com>
Date:   Wed Jun 25 18:14:35 2025 +0300

    fix(menu)

commit ff988a9f24affd8c59cec5bf674ebe21d781d1eb
Author: ArtemPGU <hgffvhhytfvb@gmail.com>
Date:   Wed Jun 25 18:10:33 2025 +0300

    Menu creation

commit be7fcb0844e47b02ce0b257ba548e5917ca5f30f
Author: Artem Artsov <alearc@mail.ru>
Date:   Tue Jun 24 09:04:53 2025 +0300
```

Рисунок 10 — Загрузки результатов в ветку *dev*

Ссылка на удаленный репозиторий:

https://github.com/oxygen-blip/Sorting

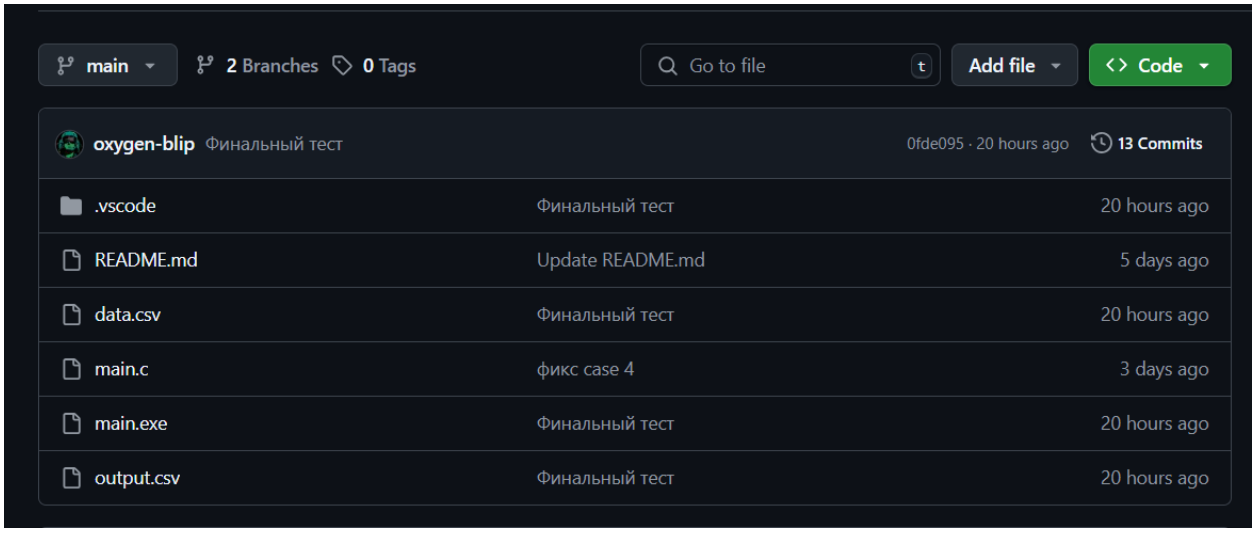


Рисунок 11 — Ветка *main*

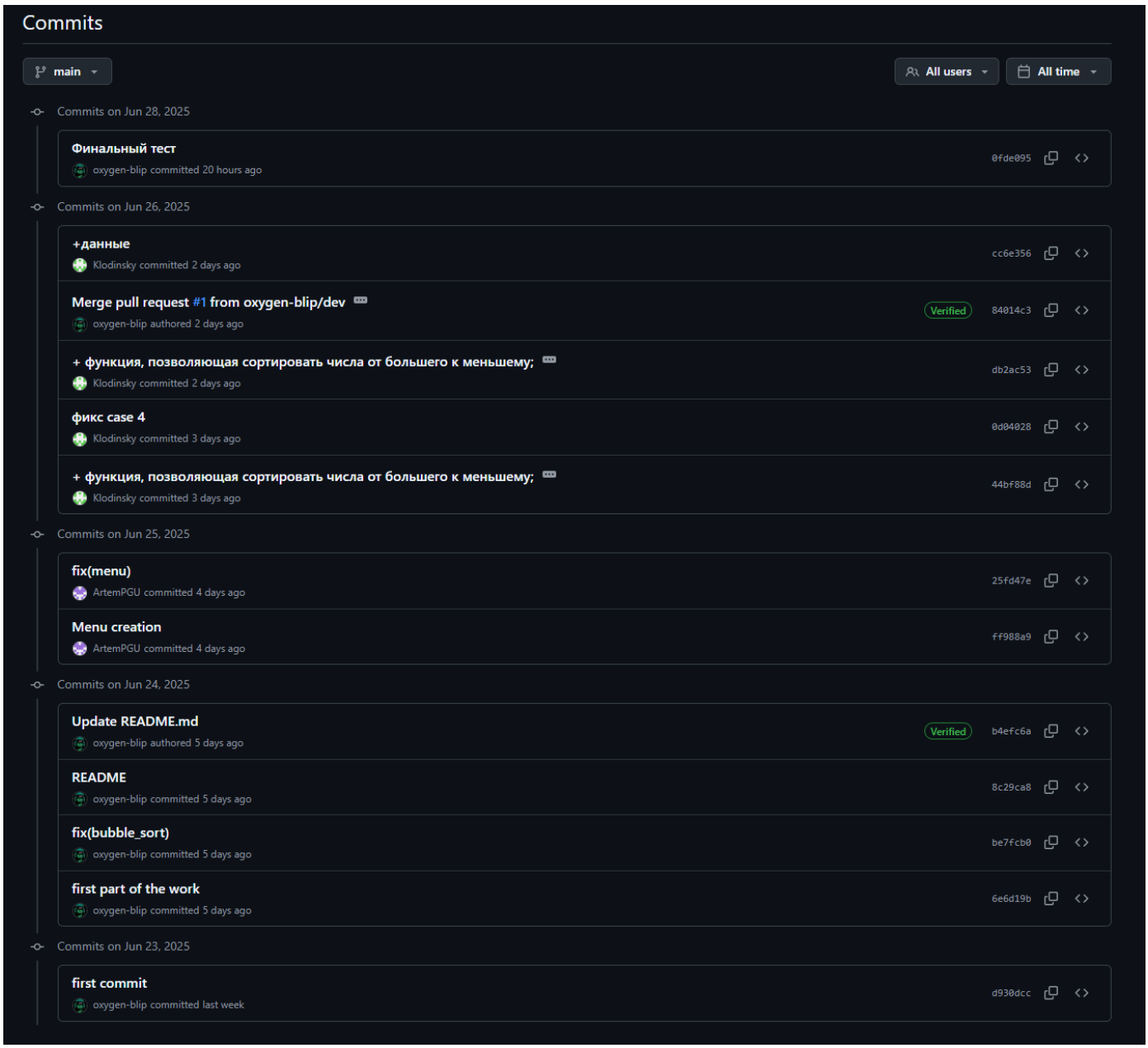


Рисунок 12 — История коммитов

Заключение

При выполнении данной работы были получены навыки совместной работы с помощью сервисов *GitHub* и *WEEEK*, навыки использования программы *Git Bash*. Был изучен алгоритм сортировки вставками.

Мной был написан алгоритм, который реализует сортировку методом пузырька, а также реализована работа с текстовыми файлами.

При выполнении практической работы были улучшены базовые навыки программирования на языке *C*. Улучшены навыки отладки, тестирования программ и работы со сложными типами данных.

В дальнейшем программу можно улучшить путем подключения упрощающих реализацию данной сортировки библиотек и улучшения графического интерфейса.

Список используемой литературы

1. Керниган, Брайан У., Ритчи, Деннис М. Язык программирования C, 2-е издание.: Пер. с англ. – М.,2009.

2. ГОСТ 19.701-90. Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения. - Введ. 1991-01-01. - М.: Стандартинформ, 2006. - 64 с.

3. Bubble Sort Algorithm [Электронный ресурс] // GeeksforGeeks. - URL: https://www.geeksforgeeks.org/bubble-sort/ (дата обращения: 28.06.2025).

Приложение А. Результаты тестирования программы

```
Введите количество чисел для генерации (до 100000): 10000
Введите диапазон генерации(через пробел): -10000 10000
Сгенерировано и записано 10000 чисел в диапазоне [-10000, 10000] в файл 'd.scv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 0.2650 секунд.
Кол-во операций: 50177655
```

Рисунок А.1

```
Введите количество чисел для генерации (до 100000): 20000
Введите диапазон генерации(через пробел): -20000 20000
Сгенерировано и записано 20000 чисел в диапазоне [-20000, 20000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 0.5720 секунд.
Кол-во операций: 99024468
```

Рисунок А.2

```
Введите количество чисел для генерации (до 100000): 30000
Введите диапазон генерации(через пробел): -30000 30000
Сгенерировано и записано 30000 чисел в диапазоне [-30000, 30000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 1.5080 секунд.
Кол-во операций: 225030561
```

Рисунок А.3

```
Введите количество чисел для генерации (до 100000): 40000
Введите диапазон генерации(через пробел): -40000 40000
Сгенерировано и записано 40000 чисел в диапазоне [-40000, 40000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 2.8580 секунд.
Кол-во операций: 399399969
```

Рисунок А.4

```
Введите количество чисел для генерации (до 100000): 50000
Введите диапазон генерации(через пробел): -50000 50000
Сгенерировано и записано 50000 чисел в диапазоне [-50000, 50000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 4.2990 секунд.
Кол-во операций: 627876185
```

Рисунок А.5

```
Введите количество чисел для генерации (до 100000): 60000
Введите диапазон генерации(через пробел): -60000 60000
Сгенерировано и записано 60000 чисел в диапазоне [-60000, 60000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 6.4490 секунд.
Кол-во операций: 901742927
```

Рисунок А.6

```
Введите количество чисел для генерации (до 100000): 70000
Введите диапазон генерации(через пробел): -70000 70000
Сгенерировано и записано 70000 чисел в диапазоне [-70000, 70000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 8.8070 секунд.
Кол-во операций: 1229572298
```

Рисунок А.7

```
Введите количество чисел для генерации (до 100000): 80000
Введите диапазон генерации(через пробел): -80000 80000
Сгенерировано и записано 80000 чисел в диапазоне [-80000, 80000] в файл 'd.csv'.

Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv

Сортировка завершена за 11.8930 секунд.
Кол-во операций: 1602882399
```

Рисунок А.8

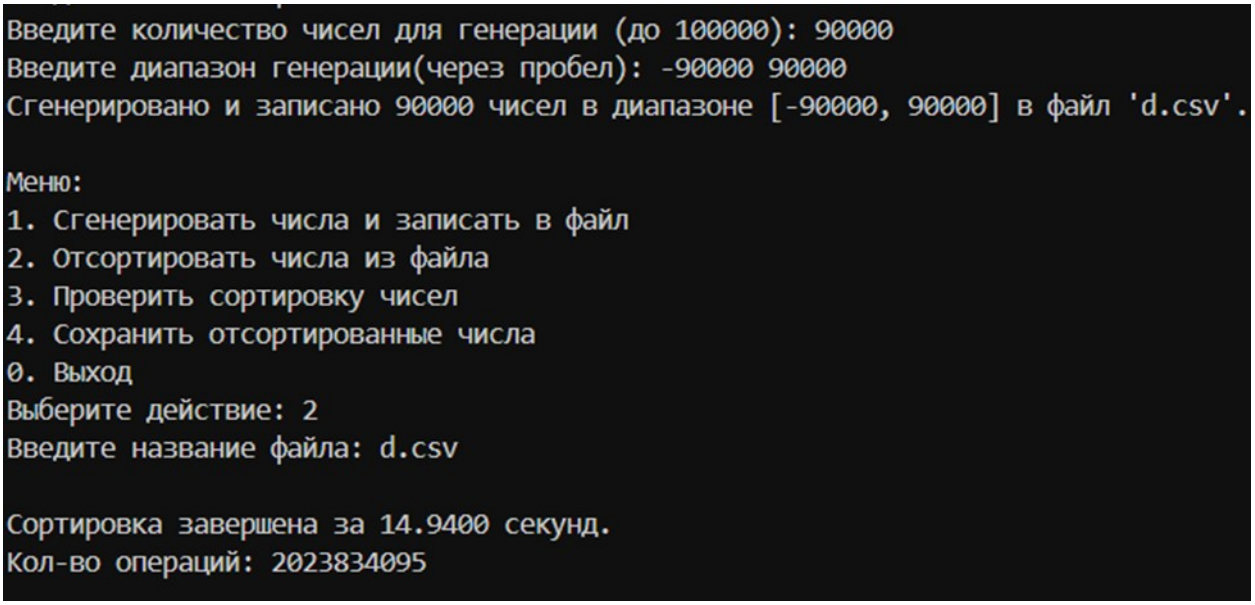

Введите количество чисел для генерации (до 100000): 90000
Введите диапазон генерации(через пробел): -90000 90000
Сгенерировано и записано 90000 чисел в диапазоне [-90000, 90000] в файл 'd.csv'.
Меню:
1. Сгенерировать числа и записать в файл
2. Отсортировать числа из файла
3. Проверить сортировку чисел
4. Сохранить отсортированные числа
0. Выход
Выберите действие: 2
Введите название файла: d.csv
Сортировка завершена за 14.9400 секунд.
Кол-во операций: 2023834095

Рисунок А.9

Приложение Б. Листинг программы

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <limits.h>

#define MAX 100000

void gen_file(const char *filename, int count, int min, int max)
{

    FILE *file = fopen(filename, "w");

    if (file == NULL) {
     printf("Ошибка открытия файла для записи данных.\n");
     return;
    }

    srand(time(NULL));

    for (int i = 0; i < count; i++) {
     long big_rand = ((long)rand() << 15) | rand();
     int nums = (int)(min + big_rand % (max - min + 1));
     fprintf(file, "%d\n", nums);
    }

    fclose(file);
    printf("Сгенерировано и записано %d чисел в диапазоне [%d,
%d] в файл '%s'.\n", count, min, max, filename);
}

int* read_file(const char *filename, int* count) {

    FILE *file = fopen(filename, "r");

    if (file == NULL) {
     printf("Файл '%s' не найден.\n", filename);
     return NULL;
    }

    int* numbers = malloc(MAX * sizeof(int));

    if (numbers == NULL) {
     printf("Ошибка при выделении памяти.\n");
     fclose(file);
     return NULL;
    }

    *count = 0;

    while (*count < MAX && fscanf(file, "%d", &numbers[*count])
== 1) {
```

```
     (*count)++;
    }

    fclose(file);
    return numbers;
}

void bubble_sort(int *array, int count, int *swap_count) {

    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                int tmp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = tmp;
               (*swap_count)++;

            }
        }
    }
}

void rev_bubble_sort(int *array, int count, int *swap_count){

    for (int i = 0; i < count - 1; i++) {
        for (int j = 0; j < count - i - 1; j++) {
            if (array[j] < array[j + 1]) {
                int tmp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = tmp;
                (*swap_count)++;
            }
        }
    }
}

int check (int *array, int count) {

    for (int i = 0; i < count - 1; i++) {
        if (array[i] > array[i + 1]) {
            return 0;
        }
    }
    return 1;
}

int check1 (int *array, int count) {

    for (int i = 0; i < count - 1; i++) {
        if (array[i] < array[i + 1]) {
            return 0;
        }
    }
```

```
    return 1;
}

int main() {
    int choice;
    int* numbers = NULL;
    int count;

    do{
     printf("\nМеню:\n");
        printf("1. Сгенерировать числа и записать в файл\n");
        printf("2. Отсортировать числа из файла\n");
        printf("3. Проверить сортировку чисел\n");
        printf("4. Сохранить отсортированные числа\n");
        printf("0. Выход\n");
        printf("Выберите действие: ");
        scanf("%d", &choice);

     switch (choice) {
          case 1: {
               char filename[256];
               int num_count;
               int num_min;
               int num_max;

               printf("Введите название файла: ");
               scanf("%255s", &filename);
               printf("Введите количество чисел для генерации (до
%d): ", MAX);
               scanf("%d", &num_count);
               printf("Введите диапазон генерации(через пробел):
");
               scanf("%d %d", &num_min, &num_max);
               if (num_count > 0 && num_count <= MAX) {
                    gen_file(filename, num_count, num_min,
num_max);
               } else {
                    printf("Недопустимое количество чисел!\n");
               }
               break;

          }
          case 2: {
               if (numbers != NULL) {
                    free(numbers);
               }
               char filename[256];
               printf("Введите название файла: ");
               scanf("%255s", filename);
               numbers = read_file(filename, &count);
               if (numbers != NULL && count > 0) {

                    int swap_count = 0;
```

```
                    printf("Выберите тип сортировки:\n");
                    printf("1. По возрастанию\n");
                    printf("2. По убыванию\n");
                    int sort_type;
                    scanf("%d", &sort_type);

                    clock_t start = clock();

                    if (sort_type == 1){
                         bubble_sort(numbers, count,
&swap_count);
                         printf("\nСортировка завершена за %.4f
секунд.\n", (double)(clock() - start)/CLOCKS_PER_SEC);
                         printf("Кол-во операций: %d\n",
swap_count);
                    }
                    else if (sort_type == 2){
                         rev_bubble_sort(numbers, count,
&swap_count);
                         printf("\nСортировка завершена за %.4f
секунд.\n", (double)(clock() - start)/CLOCKS_PER_SEC);
                         printf("Кол-во операций: %d\n",
swap_count);
                    }
               }
               break;
          }
          case 3: {
               if (numbers == NULL) {
                    char filename[256];
                    printf("Введите название файла: ");
                    scanf("%255s", filename);
                    numbers = read_file(filename, &count);
               }
               if (numbers != NULL && count > 0) {
                    if (check(numbers, count)) {
                        printf("\nЧисла отсортированы по
возрастанию.\n");
                    } else if (check1(numbers, count)) {
                        printf("\nЧисла отсортированы по
убыванию.\n");
                    } else{
                         printf("\nЧисла не отсортированы.\n");
                    }
                }
                break;

          }
          case 4: {
               if (numbers == NULL || count <= 0) {
                    char filename[256];
```

```
                    printf ("Введите назвиние файла для чтения:
");
                    scanf("%255s", filename);
                    numbers = read_file(filename, &count);
               }
               if (numbers != NULL && count > 0) {
                    int sort = check(numbers,count);
                    int rev_sort = check1(numbers,count);

                    if (!sort && !rev_sort){
                         printf("Числа не отсортированы! Хотите
отсортировать перед сохранением?\n ");
                         printf("Введите:\n1 - чтобы
отсортировать по возрастанию\n2 - чтобы отсортировать по
убыванию\nВаш выбор:");
                         int s_choice;
                         scanf("%d", &s_choice);

                         int swap_count = 0;
                         clock_t start = clock();

                         if (s_choice == 1){
                              bubble_sort(numbers, count,
&swap_count);
                              printf("\nСортировка завершена за
%.4f секунд.\n", (double)(clock() - start)/CLOCKS_PER_SEC);
                              printf("Кол-во операций: %d\n",
swap_count);
                         }
                         else if (s_choice == 2){
                              rev_bubble_sort(numbers, count,
&swap_count);
                              printf("\nСортировка завершена за
%.4f секунд.\n", (double)(clock() - start)/CLOCKS_PER_SEC);
                              printf("Кол-во операций: %d\n",
swap_count);
                         }
                         else{
                              printf("Неверный выбор.\n");
                         }
                    }
                    else
                    {
                         printf ("Введите название файла для
сохранения: ");
                         char output_filename[256];
                         scanf("%255s", &output_filename);

                         FILE *file = fopen(output_filename,
"w");
                         if (file == NULL) {
                              printf("Ошибка открытия файла!\n");
                         } else {
```

```
                                for (int i = 0; i < count; i++) {
                                     fprintf(file, "%d\n",
numbers[i]);
                                }
                                fclose(file);
                                printf("Успешно сохранено %d чисел
в файле '%s'.\n", count, output_filename);
                           }
                      }
                }
                break;
          }

          case 0: {
                printf("Работа программы завершена.\n");
                break;
          }
          default: {
                printf("Неверный выбор.\n");
                break;
          }
     }

    } while (choice != 0);

    if (numbers != NULL) {
     free(numbers);
    }
    return 0;

}
```