

## Оглавление

1. Введение.....	2
2. Постановка задачи.....	3
3. Выбор решения.....	4
4. Описание программы.....	5
5. Схемы программы.....	7
6. Тестирование программы.....	12
7. Отладка.....	15
8. Совместная разработка.....	16
9. Заключение.....	19
Список используемой литературы.....	20
Приложение А.....	21
Приложение Б “Листинг”.....	23

## **1. Введение**

Шейкерная сортировка, или сортировка перемешиванием, является модификацией пузырьковой сортировки. Она получила свое название из-за своего "трясущегося" движения элементов в процессе сортировки.

Алгоритм шейкерной сортировки начинает сортировку с начала списка элементов. Он сравнивает каждую пару соседних элементов и, если текущий элемент больше следующего, меняет их местами. Таким образом, после первого прохода самый большой элемент будет находиться в конце списка.

Затем алгоритм повторяет те же шаги, но на этот раз двигается в обратном направлении, от конца списка к началу. После прохода в обратном направлении, самый маленький элемент будет находиться в начале списка.

Алгоритм продолжает повторять эти шаги до тех пор, пока все элементы не будут отсортированы. Это происходит потому, что на каждом проходе по списку самый большой и самый маленький элементы "всплывают" на свои места.

## 2. Постановка задачи

По программе: Необходимо заполнить массив из  $n$ -ого количества элементов случайными числами, записать данные элементы в отдельный файл. После этого выполнить шейкерную сортировку над данными, находящимися в массиве, записать отсортированные данные в другой файл.

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

Оформить отчет по проведенной практике.

### Достоинства данного алгоритма сортировки:

- Алгоритм выполняет сортировку “по месту”, то есть не требует дополнительной памяти для работы.
- Алгоритм имеет ту же сложность времени выполнения как и пузырьковая сортировка, но он может быть немного эффективнее в некоторых случаях.

### Недостатки данного алгоритма сортировки:

- Алгоритм все равно имеет квадратичную сложность времени выполнения, что делает его неэффективным для больших списков элементов.
- Шейкерная сортировка не является стабильной, то есть она может изменять порядок равных элементов.

### Типичные сценарии применения:

- Сортировка массива чисел или строк.
- Сортировка элементов внутри базы данных или таблицы.
- Сортировка небольших списков данных в программе.
- Сортировка элементов в игре или программе.

### **3. Выбор решения**

Нашей бригадой было выбрано вести разработку в среде Microsoft Visual Studio на языке C/C++.

Все действия программы распределены по разным файлам.

Массив данных заполняется случайными элементами с использованием цикла for.

После заполнения массива, данные переписываются в файл “array\_before\_sort.txt”, а массив сортируется с помощью функции shaker\_sort.

После того, как массив будет отсортирован, данные из массива переписываются в другой файл “array\_after\_sort.txt ”.

Программа завершает свою работу.

Если файл не был найден или не может быть создан – программа выдает ошибку.

#### 4. Описание программы

При запуске программы происходит вывод сообщения “Введите размер массива : Размер =” после чего необходимо вписать нужное количество сортируемых данных.

```
cout << "Введите размер массива : " << endl;
do
{
    cout << "Размер = ";
    cin >> size;
    if (!(cin.good())) || (size < 0)
    {
        cout << "Неверно введен размер массива size " << endl;
        cin.clear();
        cin.get();
    }
} while (!(cin.good())) || (size < 0);
```

Затем, как данные были введены, генерируется массив из случайных чисел.

```
int* mas = new int[size];
for (int i = 0; i < size; i++) // генерация чисел
{
    mas[i] = (rand() % 2001) - 1000;
}
```

Далее вызывается функция `shaker_sort`, которая сортирует данные, находящиеся в массиве, выполняя шейкерную сортировку.

```
shaker_sort(mas, 0, (size - 1))
```

Далее выполняется сортировка, функция проходит по списку элементов несколько раз, сравнивая пары соседних элементов и меняя их местами, если они стоят в неправильном порядке.

```
int hod = 1;
while ((left < right) && hod > 0)
{
    hod = 0;
    for (int i = left; i < right; i++)
    {
        if (mas[i] > mas[i + 1])
        {
            int zam = mas[i];
            mas[i] = mas[i + 1];
            mas[i + 1] = zam;
        }
    }
}
```

```

        mas[i + 1] = zam;
        hod = 1;
    }
}
right--;
for (int i = right; i > left; i--)
{
    if (mas[i - 1] > mas[i])
    {
        int zam = mas[i];
        mas[i] = mas[i - 1];
        mas[i - 1] = zam;
        hod = 1;
    }
}
left++;
}

```

Подробный алгоритм работы программы и функции сортировки представлен в пункте 5 на рисунках 1 “Блок-схема алгоритма”, рисунке 2 “Блок-схема функции zapis\_before ”, рисунке 3 “Блок-схема функции zapis\_after”, рисунке 4 “Блок-схема функции main” и на рисунке 5 “Блок-схема программы”.

Листинг программы приведен в приложении Б.

## 5. Схемы программы

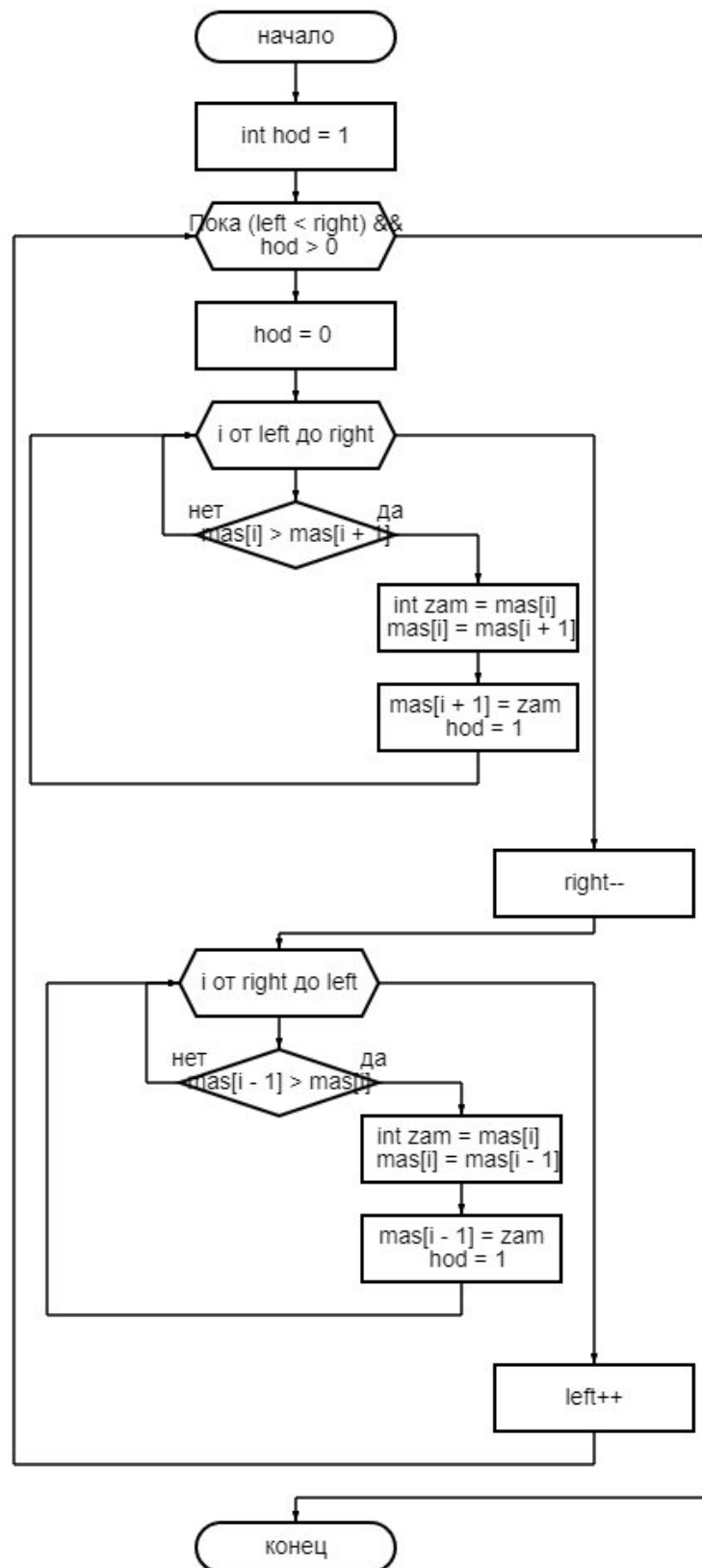


Рисунок 1 “Блок-схема функции shaker\_sort”

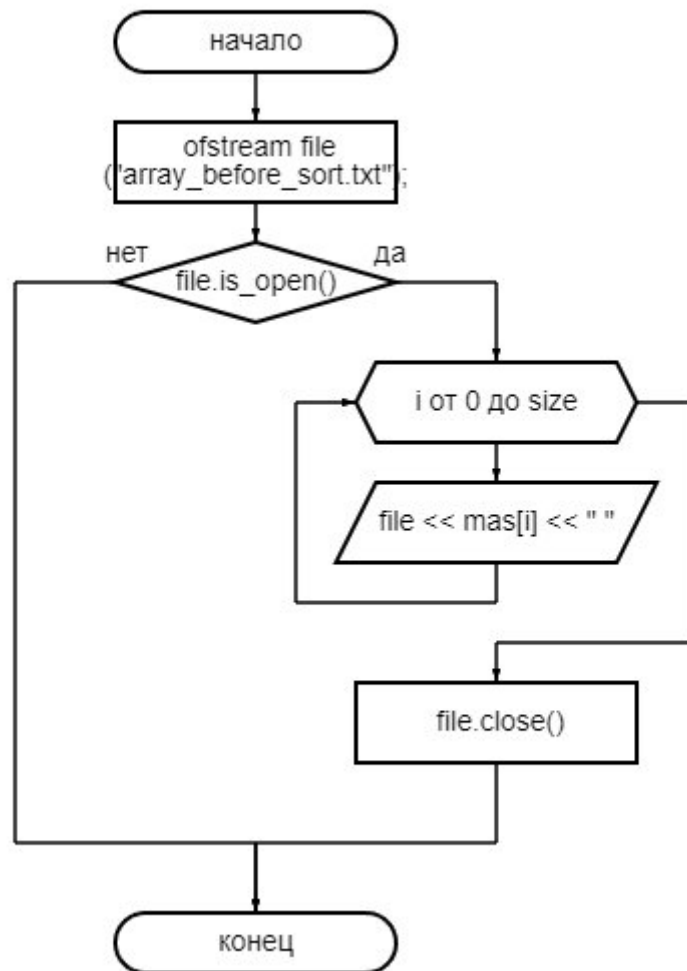


Рисунок 2 “Блок-схема функции zapis\_before ”



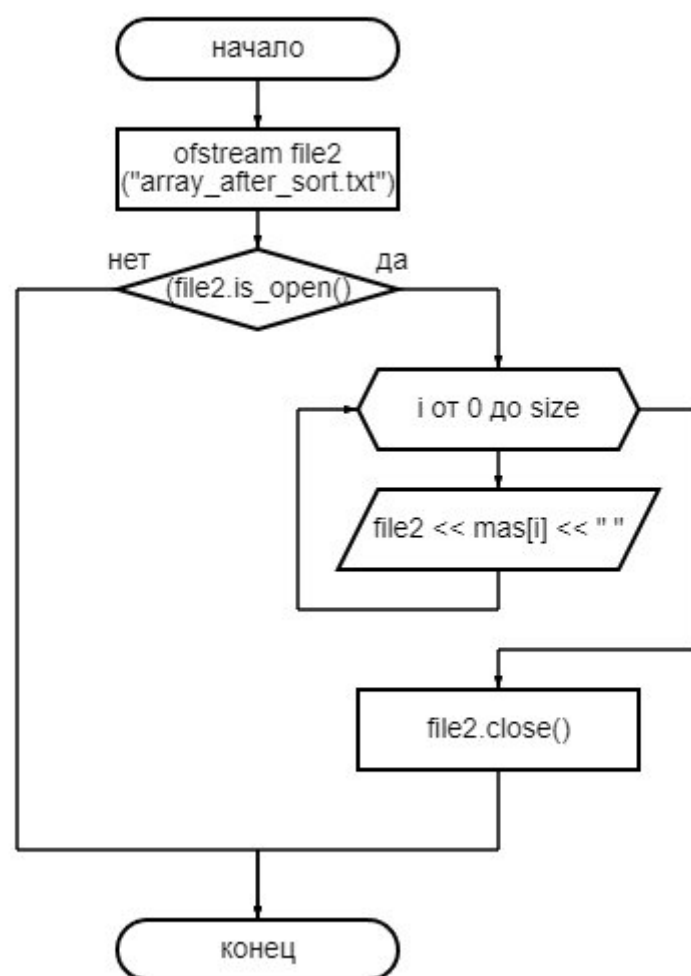


Рисунок 3 “Блок-схема функции zapis\_after”

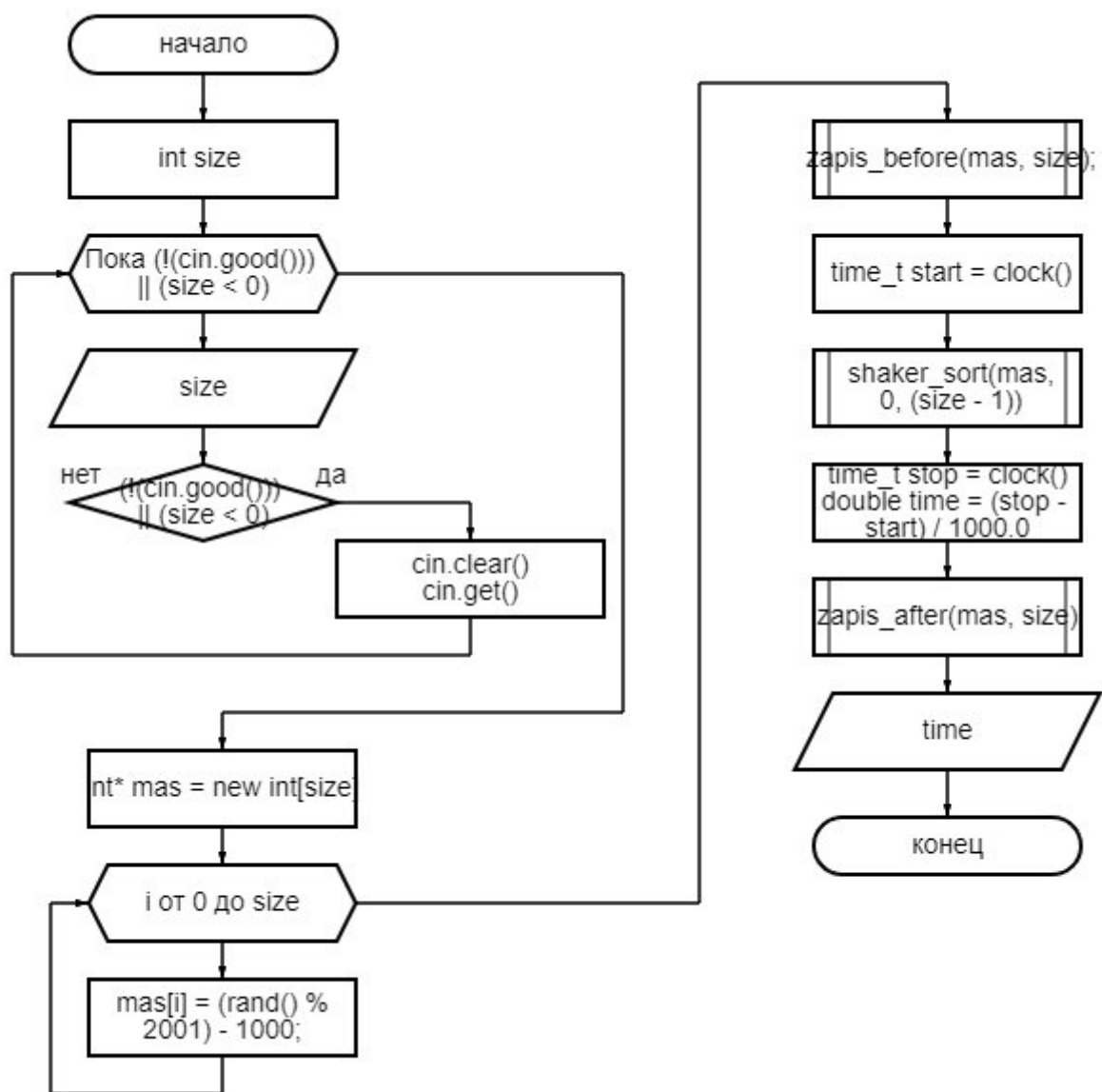


Рисунок 4 “Блок-схема функции main”



Рисунок 5 “Блок-схема программы”

## 6. Тестирование программы

Тестовый набор данных представлен в таблице 1 и 2.

Снизу представлены диаграммы. (Диаграмма 1 и Диаграмма 2, в которых представлена линейная зависимость количества данных от времени, затраченного на выполнение сортировки)

Результаты тестирования приведены в Приложении А на рисунках А.9 - А.18.

Таблица 1 – Тестовый набор данных на диапазоне от 1 до 10000

	Кол-во данных	Время выполнения сортировки (сек)
1	1000	0,001
2	3000	0,009
3	5000	0,022
4	7000	0,043
5	9000	0,071

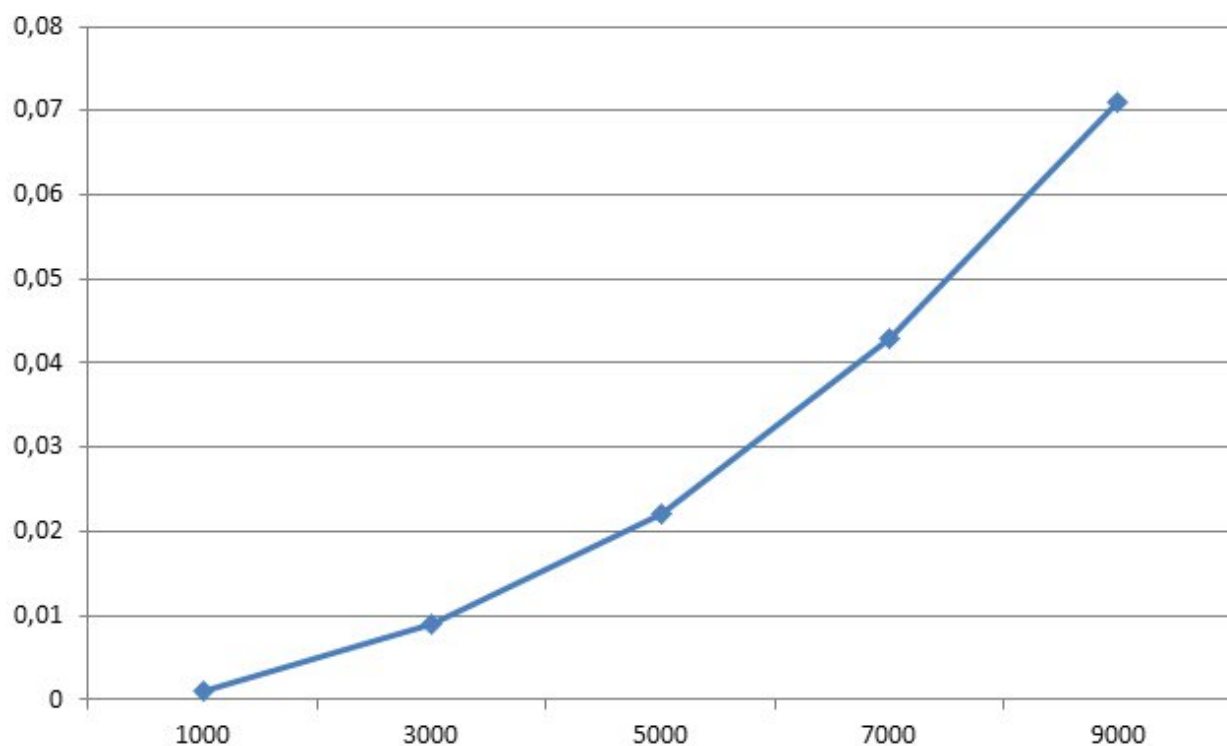


Диаграмма 1 “Тестирование программы на диапазоне от 1 до 10000”

Таблица 2 – Тестовый набор данных на диапазоне от 1 до 100000

	Кол-во данных	Время выполнения сортировки (сек)
1	10000	0,087
2	30000	0,928
3	50000	3,06
4	70000	6,213
5	90000	10,553

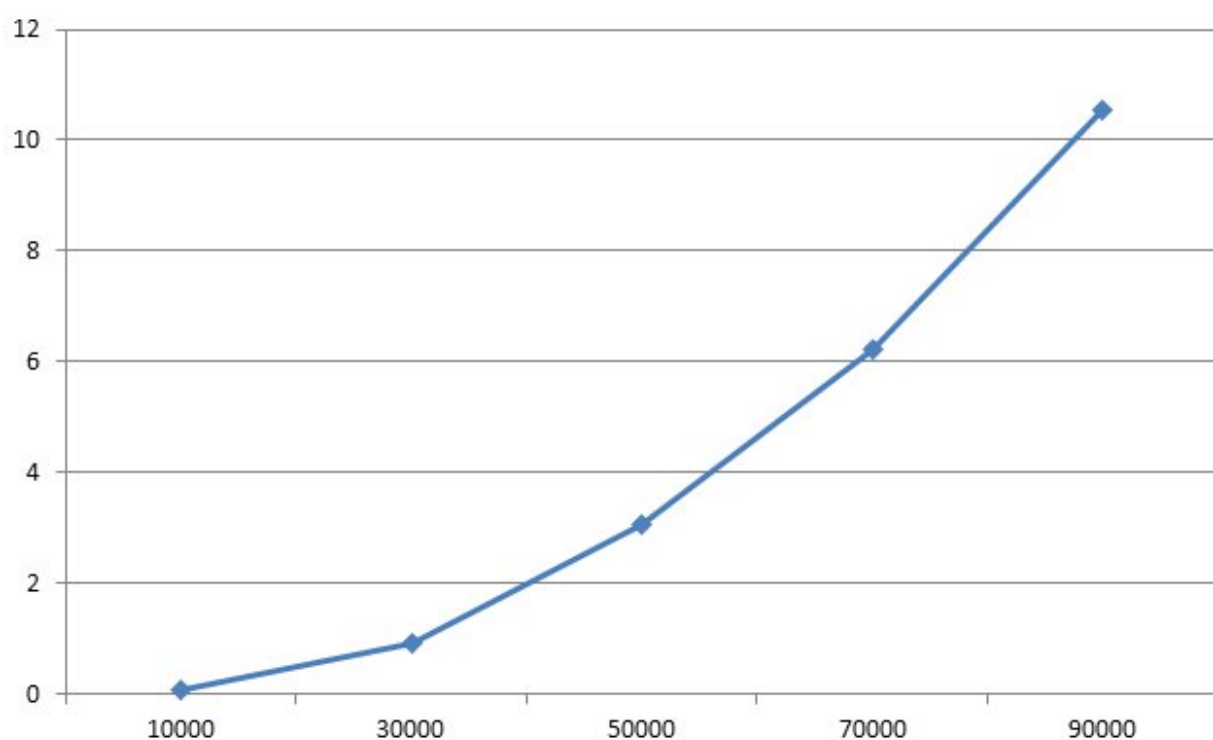


Диаграмма 2 “Тестирование на диапазоне от 1 до 100000”

В результате выполнения тестирования было выявлено время работы данного алгоритма и получен конечный результат, описанный в таблицах.

Тестирование данной программы производилось с помощью встроенных строчек кода, реализующих таймер, засекающий время работы программы.

Лёвин А.Д. выполнил тестировку и засек результаты, которые в последующем внес в таблицы №1 и №2. Можно заметить, что время работы сортировки увеличивается с увеличением количества данных в массиве.

## **7. Отладка**

В качестве среды разработки была выбрана программа Microsoft Visual Studio , которая содержит в себе все необходимые средства для разработки и отладки модулей и программ.

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

Тестирование проводилось в рабочем порядке, в процессе разработки, после завершения написания программы.



## 8. Совместная разработка

Во время работы над данной практикой, нашей бригадой осуществлялась совместная работа в GitHub.

Данная программа была написана владельцем репозитория – Дунюшкиным В.А.

После написания программы, она была выгружена на удаленный репозиторий Github, на отдельную ветку под названием Summer-Practicate.

(См. Рисунок 6)

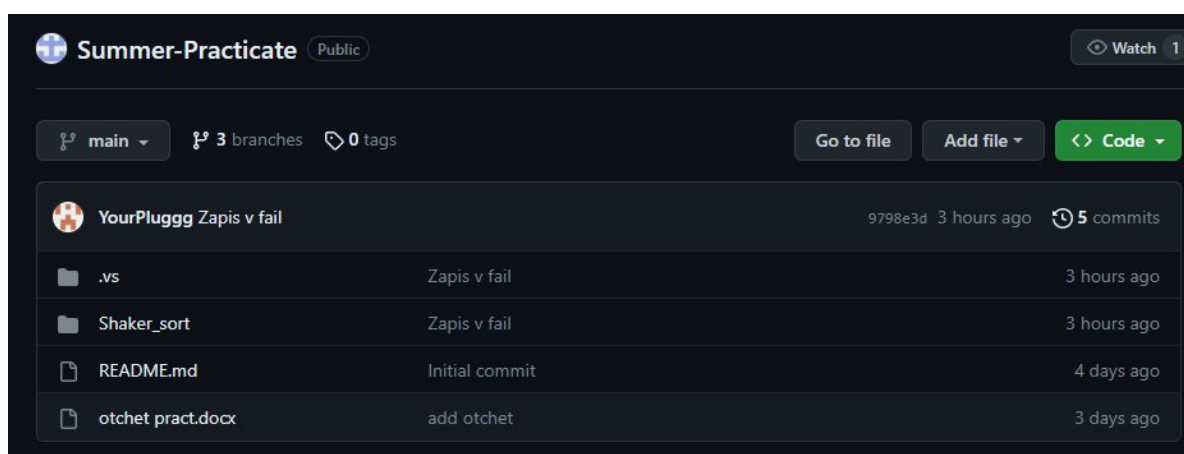


Рисунок 6

Далее, второй участник – Колобов И.О. загрузил программу на компьютер с помощью git clone и добавил запись в файл. После чего создал собственную ветку, и выгрузил обновленный код программы на удаленный репозиторий Github. (См. Рисунок 7)



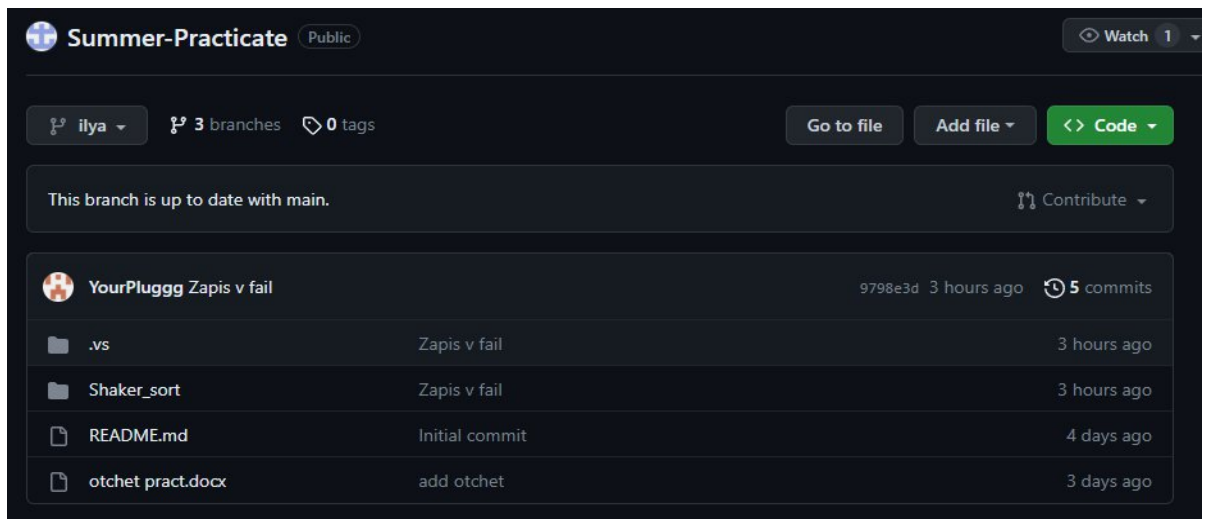


Рисунок 7

После этого, третий участник – Лёвин А.Д. загрузил данную программу себе на компьютер, с помощью `git clone` и протестировал и отладил её. После чего создал собственную ветку, и выгрузил обновленный код программы на удаленный репозиторий GitHub.

(См. Рисунок 8)

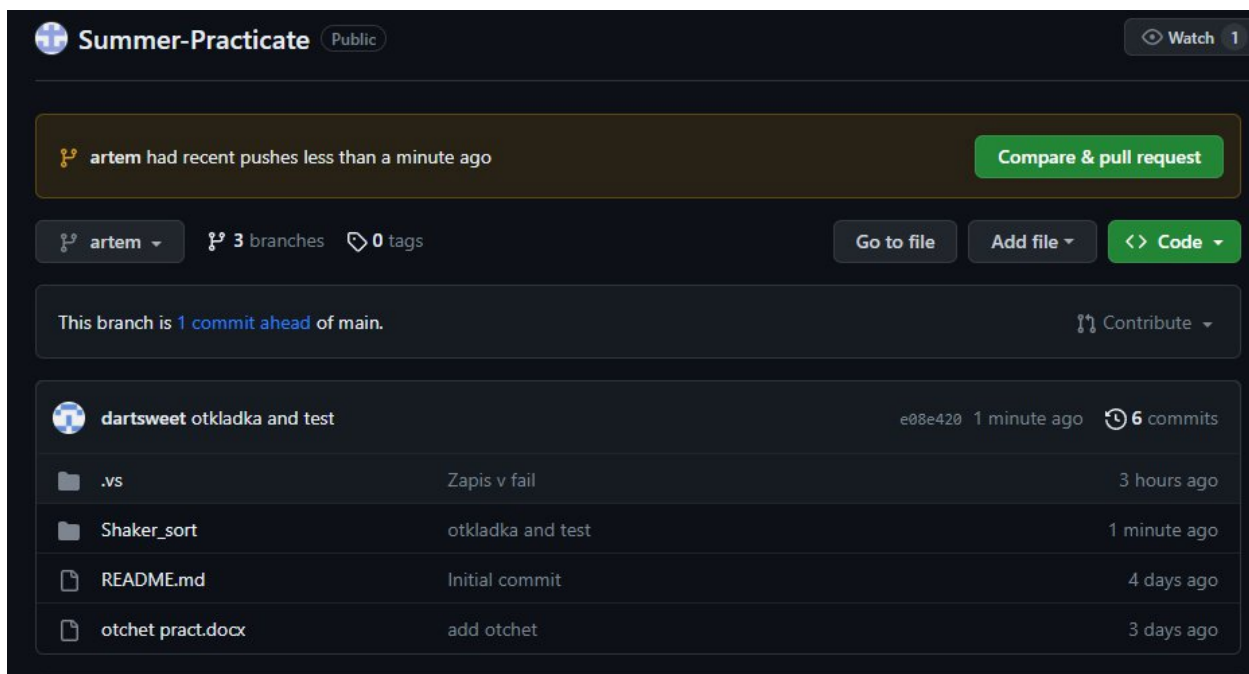


Рисунок 8

Ссылка на удаленный репозиторий:

<https://github.com/Issy4444Mega/Summer-Practicate.git>

## **9. Заключение**

Нашей бригадой были получены навыки совместной работы с помощью сервиса GitHub, навыки использования программы Git Bash. Нами так же был изучен алгоритм шейкерной сортировки. Дунюшкин В.А. написал программу, выполняющую данную сортировку над массивом случайно сгенерированных чисел. Колобов И.О. реализовал работу с файлами. Лёвин А.Д. выполнил тестирование и отладку данной программы, оформил отчет по данной практике.

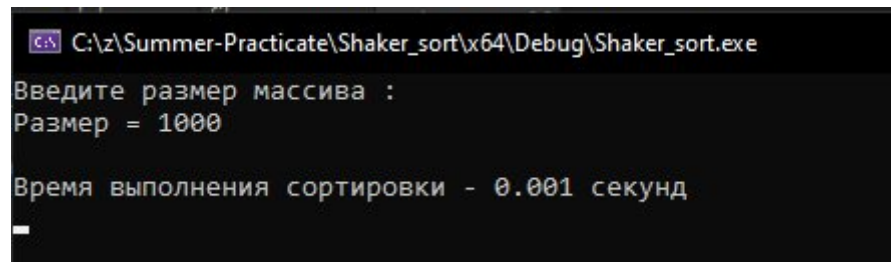
Так же при выполнении практической работы были улучшены наши базовые навыки программирования на языках C/C++. Улучшены навыки отладки, тестирования программ и работы со сложными типами данных.

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

### **Список используемой литературы**

1. Керниган Б. Ритчи Д. Язык программирования С. 1985 г.
2. Прата С. Язык программирования С++. 2019 г.
3. А.А. Тюгашев. Языки программирования. Учебное пособие. 2018 г.
4. Деннис Ритчи, Брайан Керниган. Язык программирования Си. 1978 г.
5. Солдатенко, И.С. Основы программирования на языке Си: учеб. Пособие. 2017г.
- 6.. Кормен, Т., Лейзерсон, Ч., Ривест, Р., Штайн, К. Алгоритмы: построение и анализ = Introduction to Algorithms / Под ред. И. В. Красикова. — 2-е изд. 2005.
7. Бхаргава А. Грожаем алгоритмы. Иллюстрированное пособие для программистов и любопытствующих. - СПб.: Питер, 2017. - 288 с. : ил. - (Серия «Библиотека программиста»).

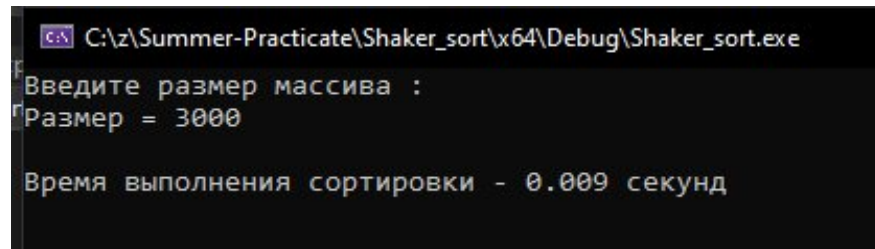
## Приложение А



```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 1000

Время выполнения сортировки - 0.001 секунд
_
```

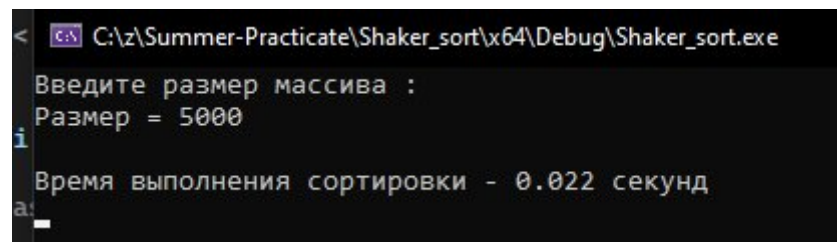
Рисунок А.9



```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 3000

Время выполнения сортировки - 0.009 секунд
```

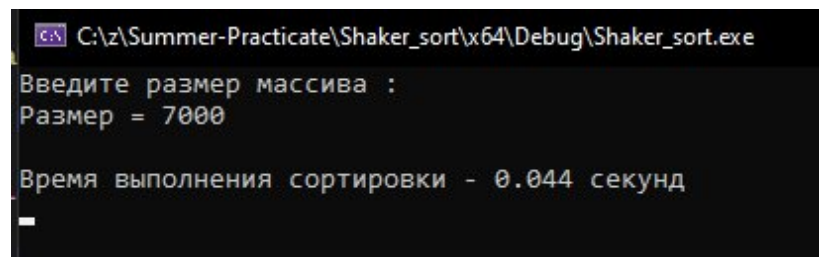
Рисунок А.10



```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 5000

Время выполнения сортировки - 0.022 секунд
_
```

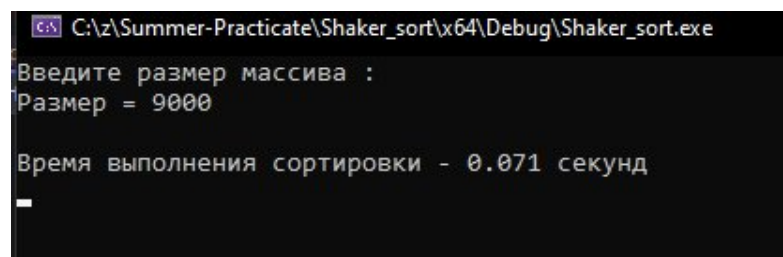
Рисунок А.11



```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 7000

Время выполнения сортировки - 0.044 секунд
_
```

Рисунок А.12



```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 9000

Время выполнения сортировки - 0.071 секунд
_
```

Рисунок А.13

```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 10000
Время выполнения сортировки - 0.087 секунд
```

Рисунок А.14

```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 30000
Время выполнения сортировки - 0.955 секунд
```

Рисунок А.15

```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 50000
Время выполнения сортировки - 3.015 секунд
```

Рисунок А.16

```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 70000
Время выполнения сортировки - 6.246 секунд
```

Рисунок А.17

```
C:\z\Summer-Practicat\Shaker_sort\x64\Debug\Shaker_sort.exe
Введите размер массива :
Размер = 90000
Время выполнения сортировки - 10.604 секунд
```

Рисунок А.18

## Приложение Б “Листинг”

### Файл Shaker\_sort.cpp

```
#include "h.h"

int main()
{
    setlocale(LC_ALL, "Russian"); //русификация

    srand(time(NULL));
    int size;
    cout << "Введите размер массива : " << endl;
    do
    {
        cout << "Размер = ";
        cin >> size;
        if ((!(cin.good())) || (size < 0))
        {
            cout << "Неверно введен размер массива size " << endl;
            cin.clear();
            cin.get();
        }
    } while ((!(cin.good())) || (size < 0));

    int* mas = new int[size];

    for (int i = 0; i < size; i++) // генерация чисел
    {
        mas[i] = (rand() % 2001) - 1000;
    }
    zapis_before(mas, size);

    //
    time_t start = clock(); //время до сортировки
    shaker_sort(mas, 0, (size - 1)); //вызов функции сортировки
    time_t stop = clock(); //время после сортировки
    double time = (stop - start) / 1000.0; //время сортировки

    zapis_after(mas, size);
    cout << endl << "Время выполнения сортировки - " << time << " секунд " <<
endl;

    getchar(); getchar();
}
```

### Файл h.h

```
#pragma once
#include <iostream>
#include <fstream>
#include <ctime>
#include <cstdlib>

using namespace std;

void shaker_sort(int* mas, int left, int right);
void zapis_before(int* mas, int size);
void zapis_after(int* mas, int size);
```

### Файл file.cpp

```
#include "h.h"

void zapis_before(int* mas, int size) {

    ofstream file("array_before_sort.txt"); // открытие файла для записи до
    сортировки
    if (file.is_open())
    {
        for (int i = 0; i < size; i++)
        {
            file << mas[i] << " "; // запись элементов массива в файл
        }
        file.close(); // закрытие файла
    }
}

void zapis_after(int* mas, int size) {

    ofstream file2("array_after_sort.txt"); // открытие файла для записи после
    сортировки
    if (file2.is_open())
    {
```



```

    for (int i = 0; i < size; i++)
    {
        file2 << mas[i] << " "; // запись элементов массива в файл
    }
    file2.close(); // закрытие файла
}
}

```

### Файл sort.cpp

```

#include "h.h"
void shaker_sort(int* mas, int left, int right)
{
    int hod = 1;
    while ((left < right) && hod > 0)
    {
        hod = 0;
        for (int i = left; i < right; i++)
        {
            if (mas[i] > mas[i + 1])
            {
                int zam = mas[i];
                mas[i] = mas[i + 1];
                mas[i + 1] = zam;
                hod = 1;
            }
        }
        right--;
        for (int i = right; i > left; i--)
        {
            if (mas[i - 1] > mas[i])
            {
                int zam = mas[i];
                mas[i] = mas[i - 1];
                mas[i - 1] = zam;
                hod = 1;
            }
        }
        left++;
    }
}

```