

Федеральное государственное образовательное бюджетное учреждение  
высшего профессионального образования  
«Нижегородский Государственный Университет им.  
Н.И.Лобачевского» (ННГУ)

Национальный исследовательский Университет  
Институт Информационных Технологий Математики и Механики

## **Отчёт по лабораторной работе**

### **Работа с файлами и массивами, методы сортировки**

Выполнил:  
студент группы 3821Б1ПМЗ

Заботин М.А

Проверил:  
заведующий лабораторией  
суперкомпьютерных технологий и  
высокопроизводительных вычислений

Лебедев И.Г

Нижний Новгород  
2021 г.

## Оглавление

1. Введение	3
2. Постановка задачи	4
3. Руководство пользователя	5
3.1 Первая программа	5
3.2 Вторая программа	7
4. Руководство программиста	10
4.1 Описание структур данных	10
4.1.1 Библиотеки	10
4.1.2 Функции	10
4.1.3 Переменные	11
4.2 Описание структуры программы	12
4.2.1 Структура первой программы	12
4.2.2 Структура второй программы	14
4.3 Описание алгоритмов	19
4.3.1 Описание алгоритма заполнения файла случайными числами	19
4.3.2 Описание алгоритма сортировки пузырьком.	20
4.3.3 Описание алгоритма сортировки вставками	22
4.3.4. Описание алгоритма быстрой сортировки	24
5. Эксперименты	26
6. Заключение	28
7. Литература	29
Приложение	30
Приложение 1	30
Приложение 2	32

## 1. Введение

Программирование - это интересный, полезный и увлекательный процесс, благодаря которому мы можем с помощью специальных команд обучать компьютер, делать для нас разнообразные полезные задачи, от выполнения операций с числами и навигации, до управления самолетами, спутниками и марсоходами.

Чтобы программировать сложные алгоритмы, необходимо постоянно пополнять свои знания о структурах и методах изучаемого языка программирования.

Работа с текстовыми файлами является практически неотъемлемой частью в программировании, потому что многие данные хранятся именно в текстовых файлах, которые достаточно просты и удобны в использовании. Но бывает, что данные находятся в хаотичном порядке. С такими данными иногда неудобно работать, поэтому, чтобы упорядочить данные в файле существует несколько видов сортировок, которые отличаются временем и областью работы, а также уровнем сложности.

В данной лабораторной работе для изучения особенностей работ с файлами а также методы и принципы работы нескольких видов сортировок, а именно сортировки пузырьком, быстрой сортировки и сортировки вставками, была поставлена задача на создание двух программ на языке программирования «С».

## 2. Постановка задачи

Реализовать сортировки массивов данных задаваемых: обязательно случайно, дополнительно с клавиатуры. Реализовать сортировки: пузырьком, вставкой, быстрая. Сравнить время работы, сделать выводы.

Первая программа создает текстовый файл с записанными в него числами. Программа принимает количество чисел  $n$ , максимальное и минимальное значение. Вторая программа читает текстовый файл с набором чисел, выводит консольный интерфейс (печать, сортировка, сброс, выход), выполняет выбранные действия.

### 3. Руководство пользователя

#### 3.1 Первая программа

Первая программа отвечает за создание текстового файла, заполняемого вручную, либо случайными числами из некоторого диапазона.

При запуске программы в папке создаётся пустой текстовый файл, а на консоль выводится сообщение «Введите количество элементов», после чего необходимо ввести количество чисел, с которыми пользователь хочет работать и нажать «enter». (см Рис.1)

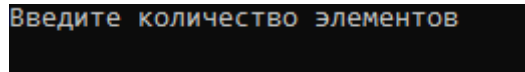


Рисунок 1. Консоль после запуска программы

После ввода количества элементов программа предложит заполнить массив вручную или случайными числами из какого-то диапазона. (См Рис.2)

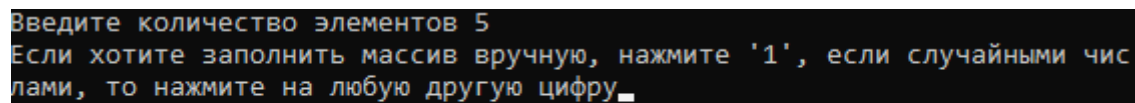


Рисунок 2. Консоль после ввода количества элементов

Если пользователь выбирает ввести элементы вручную и нажимает «1», то программа начинает принимать числа, не важно, дробные или целые, в количестве, которое пользователь указал ранее. После введения каждого числа необходимо нажать «enter». (См Рис.3)

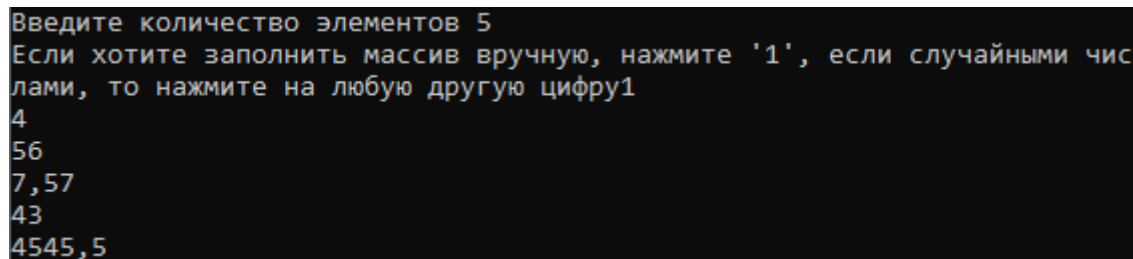


Рисунок 3. Ручной ввод данных

После ручного ввода данных в файле на первой строке будут указано количество чисел, а на следующих числа, введенные пользователем. (См Рис.4)

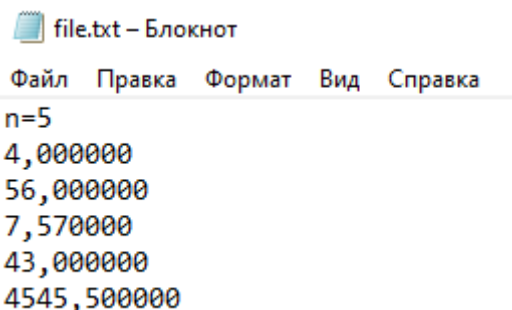
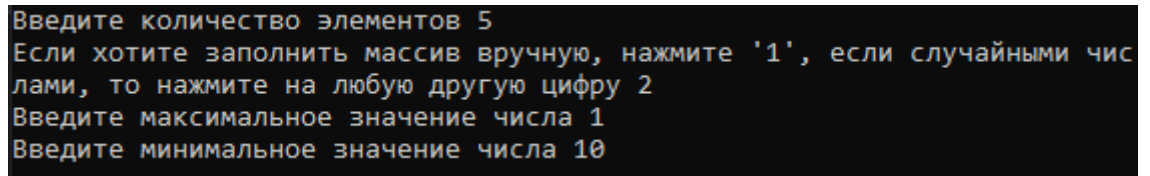


Рисунок 4. Вид файла после ручного ввода данных.

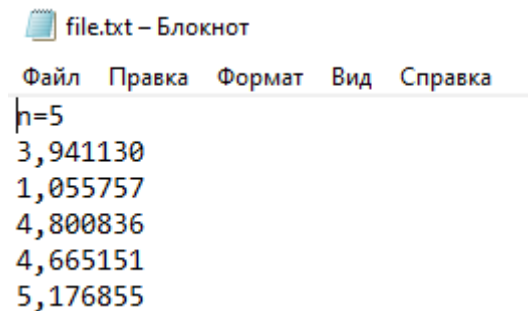
Если же пользователь захочет заполнить файл случайными числами из некоторого диапазона, то программа попросит пользователя ввести минимальное и максимальное число диапазона. После чего необходимо будет нажать «enter». (См Рис.5)



```
Введите количество элементов 5
Если хотите заполнить массив вручную, нажмите '1', если случайными числами, то нажмите на любую другую цифру 2
Введите максимальное значение числа 1
Введите минимальное значение числа 10
```

Рисунок 5. Вид консоли при заполнении файла случайными числами.

После работы программы и заполнения файла случайными числами, в файле на 1 строке всё также будет указано количество элементов в файле, а на следующих случайно сгенерированные числа. (См Рис.6)



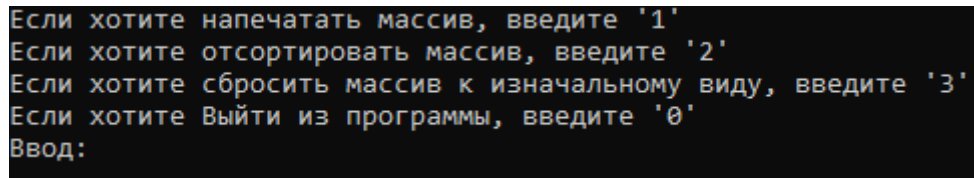
```
file.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
n=5
3,941130
1,055757
4,800836
4,665151
5,176855
```

Рисунок 6. Вид файла после заполнения случайными числами из диапазона

### 3.2 Вторая программа

Во второй программе реализован простой и понятный интерфейс, который позволяет пользователю выполнять несколько операций с текстовым файлом, а именно, печать на консоль, сортировка данных в файле несколькими способами, сброс файла к изначальному виду. Если пользователь больше не захочет находиться в программе, то реализована функция выхода, которая завершает работу программы.

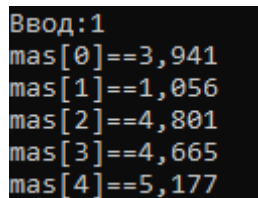
В начале программы, как и после выполнения любого действия, отображается меню выбора действия, с помощью которого пользователь может выбрать интересующее его действие. (См Рис.7)



```
Если хотите напечатать массив, введите '1'  
Если хотите отсортировать массив, введите '2'  
Если хотите сбросить массив к изначальному виду, введите '3'  
Если хотите Выйти из программы, введите '0'  
Ввод:
```

Рисунок 7. Окно выбора действия

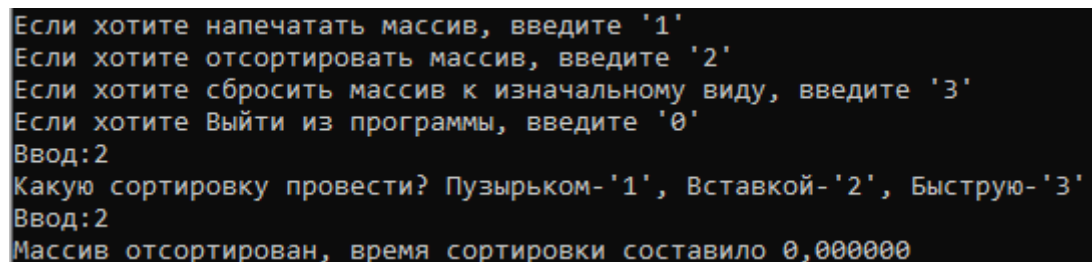
Если пользователь хочет распечатать текущий файл, то необходимо нажать «1» и программа произведёт печать файла на консоль (См Рис.8)



```
Ввод:1  
mas[0]==3,941  
mas[1]==1,056  
mas[2]==4,801  
mas[3]==4,665  
mas[4]==5,177
```

Рисунок 8. Печать массива на консоль

Если пользователь хочет отсортировать массив, то необходимо нажать «2», после чего программа уточнит вид используемой сортировки, (для выбора вида сортировки снова придётся нажать на клавишу «1» или «2» или «3», в зависимости от того, какую сортировку пользователь хочет провести), после чего выведет сообщение о проведении сортировки и время в секундах, которое было затрачено. (См Рис.9) После этого исходный файл будет отсортирован. (См Рис.10 и Рис.11)



```
Если хотите напечатать массив, введите '1'  
Если хотите отсортировать массив, введите '2'  
Если хотите сбросить массив к изначальному виду, введите '3'  
Если хотите Выйти из программы, введите '0'  
Ввод:2  
Какую сортировку провести? Пузырьком-'1', Вставкой-'2', Быструю-'3'  
Ввод:2  
Массив отсортирован, время сортировки составило 0,000000
```

Рисунок 9. Результат работы программы, при сортировке данных файла

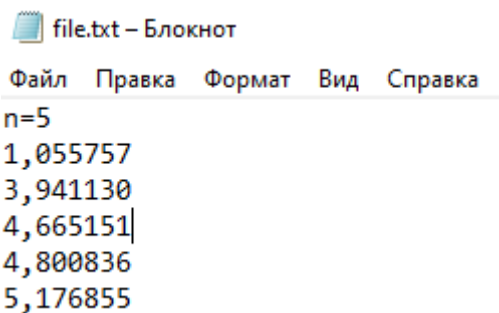


Рисунок 10. Вид файла после сортировки

```
mas[0]==1,056
mas[1]==3,941
mas[2]==4,665
mas[3]==4,801
mas[4]==5,177
```

Рисунок 11. Распечатанный после сортировки массив

Если пользователь хочет вернуть файл к исходному, неотсортированному виду, то необходимо нажать «3», после чего на консоли появится сообщение о возвращении файла к исходному виду. (См Рис.12) После этого файл будет выглядеть так, как он был подан в программу. (См Рис.13)

```
Ввод:3
Файл был приведён к изначальному виду
```

Рисунок 12. Вид консоли после приведения файла к изначальному виду

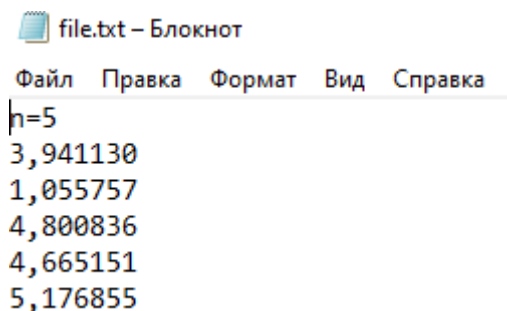


Рисунок 13. Приведение файла к исходному виду

Если же пользователь хочет завершить работу с программой, то необходимо нажать «0», после чего программа выведет на консоль сообщение о завершении работы и завершит работу. (См Рис.14)

```
Ввод:0
Программа завершила работу
```

Рисунок 14. Завершение работы программы

Если пользователь введёт некорректные данные, то программа выведет на консоль соответствующее сообщение и заново предоставит пользователю информацию о доступных действиях. (См Рис.15)



```
Ввод:564
Введите корректные данные

Если хотите напечатать массив, введите '1'
Если хотите отсортировать массив, введите '2'
Если хотите сбросить массив к изначальному виду, введите '3'
Если хотите Выйти из программы, введите '0'
Ввод: _
```

Рисунок 15. Результат работы программы, при некорректных введенных данных

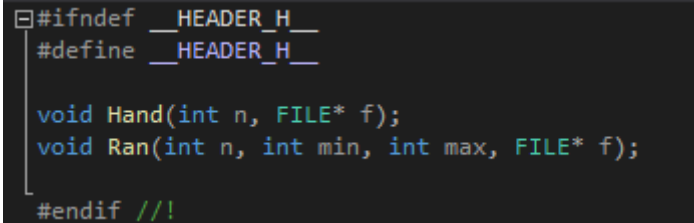
## 4. Руководство программиста

### 4.1 Описание структур данных

#### 4.1.1 Библиотеки

В обеих программах используется несколько библиотек:

- 1) «stdio.h», стандартный заголовочный файл
- 2) «stdlib.h», для работы со случайными числами
- 3) «locale.h», для корректного отображения русскоязычных символов
- 4) «time.h», для измерения времени работы программы и работы со случайными числами
- 5) Также в обеих программах используется заголовочный файл «Header.h», в котором объявляются функции, для удобной работы с ними. (См Рис.16 и Рис.17)

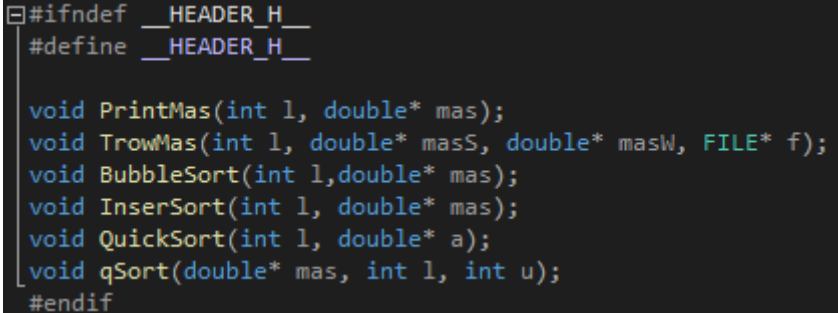


```
#ifndef __HEADER_H__
#define __HEADER_H__

void Hand(int n, FILE* f);
void Ran(int n, int min, int max, FILE* f);

#endif //!
```

Рисунок 16. Вид заголовочного файла в первой программе



```
#ifndef __HEADER_H__
#define __HEADER_H__

void PrintMas(int l, double* mas);
void TrowMas(int l, double* masS, double* masW, FILE* f);
void BubbleSort(int l, double* mas);
void InserSort(int l, double* mas);
void QuickSort(int l, double* a);
void qSort(double* mas, int l, int u);

#endif
```

Рисунок 17. Вид заголовочного файла во второй программе

#### 4.1.2 Функции

В первой программе используется 2 функции:

- 1) «Hand», которая отвечает за ручной ввод данных в файл (на вход подаётся количество элементов в файле и сам файл)
- 2) «Ran», которая заполняет файл случайно сгенерированными числами (на вход подаётся количество элементов в файле, минимальное и максимальное значение элементов и сам файл)

Во второй программе используется 6 функций:

- 1) «PrintMas», используется для печати чисел из файла на консоль (на вход подаётся количество элементов в файле и массив, который содержит те же данные, какие в файле)

- 2) «TrowMas», которая позволяет сбросить файл и массив к исходным данным (на вход подаётся количество элементов в файле, массив с данными файла, массив, которых хранит в себе данные об изначальном виде файла и сам файл)
- 3) «BubbleSort», которая выполняет сортировку массива «методом пузырька» (на вход подаётся количество элементов и массив с текущими данными файла)
- 4) «InserSort», которая выполняет сортировку массива «методом вставок» (на вход подаётся количество элементов и массив с текущими данными файла)
- 5) «qSort», которая выполняет сортировку массива «методом быстрой сортировки» (на вход подаётся массив, с элементами файла, а также вспомогательные переменные для сортировки массива)
- 6) «QuickSort», которая является вспомогательной, для запуска «Быстрой сортировки» (на вход подаётся количество элементов и массив с текущими данными файла)

#### **4.1.3 Переменные**

В первой программе используется 5 глобальных переменных:

- 1) Переменная «\*file» типа «FILE», которая отвечает за хранения данных файла
- 2) Переменная «n» типа «long int», которая хранит в себе количество элементов в файле
- 3) Переменная «state» типа «int», которая позволяет сделать выбор действия
- 4) 2 переменные «min» и «max» типа «double», для хранения диапазона выбора случайных чисел

Также используется несколько локальных переменных, которые определяются в функциях или служат образами глобальных переменных.

Во второй программе используется 9 глобальных переменных

- 1) Переменная «\*file» типа «FILE», которая отвечает за хранения данных файла
- 2) Переменная «N» и «i» типа «long int», первая из которых хранит в себе количество элементов в файле, а вторая является итератором циклов и способствует перебору элементов в файле
- 3) Переменные «t1» и «t2» типа «clock\_t», для подсчёта времени выполнения сортировок
- 4) Переменные «state» и «sort» типа «int», первая из которых отвечает за выбор пользователем интересующего действия, а вторая за выбор способа сортировки файла
- 5) Переменные «\*SaveMas» и «\*SortMas» типа «double», первая из которых хранит в себе изначальный вид данных в файле, а вторая текущий

Также используется несколько локальных переменных, которые определяются в функциях или служат образами глобальных переменных.

- 6)

## 4.2 Описание структуры программы

### 4.2.1 Структура первой программы

1) Заголовочный файл «Header.h» первой программы содержит объявление двух функций, случайного и ручного ввода чисел.

Фрагмент кода 1.

```
#ifndef __HEADER_H__
#define __HEADER_H__
void Hand(int n, FILE* f);
void Ran(int n, int min, int max, FILE* f);
#endif
```

2) В исходном файле «functions.c» сначала идёт подключение необходимых для нормальной работы библиотек

Фрагмент кода 2.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include "Header.h"
#include <time.h>
```

После чего в этом исходном файле идёт реализация двух основных функций в первой программе, а именно заполнение файла вручную с клавиатуры или случайными числами из заданного диапазона (обе получают на вход количество элементов в файле и сам файл, а дальше с помощью цикла «for» заполняют файл либо использованием случайных чисел, либо считывают данные с клавиатуры)

Фрагмент кода 3.

```
void Hand(int n, FILE* f)
{
    double t;
    for (int i = 0; i < n; i++)
    {
        scanf_s("%lf", &t);
        fprintf(f, "%lf\n", t);
    }
}

void Ran(int n, int min, int max, FILE* f)
{
    double t;
    srand(time(0));
    for (int i = 0; i < n; i++)
    {
        t = (((double)rand()) / RAND_MAX * (max - min) + min);
        fprintf(f, "%lf\n", t);
    }
}
```

3) В исходном файле «main.c» сначала тоже идёт объявление всех необходимых для нормальной работы библиотек, как и в первом файле

Фрагмент кода 4.

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include "Header.h"
```

```
#include <time.h>
```

После чего начинается главная функция «main», в которой сначала идёт объявление и инициализация всех необходимых переменных, установка русской локализации, а также создание и открытие файла для записи.

Фрагмент кода 5.

```
int main()
{
    FILE *file;
    long int n = 0;
    int state = 0;
    double min=0, max=0;
    setlocale(LC_ALL, "Rus");
    fopen_s(&file, "..\\file.txt", "w");
```

После чего программа просит ввести пользователя количество элементов, которое он хочет видеть в файле, запись этого числа в файл и спрашивает, как пользователь хочет заполнить файл, вручную или с помощью генерации случайных чисел. С помощью переменной «state», программа определяет, как именно пользователь хочет заполнить файл.

Фрагмент кода 6.

```
printf("Введите количество элементов");
scanf_s("%d", &n);
fprintf_s(file, "n=%d\n", n);
printf("Если хотите заполнить массив вручную, нажмите '1', если случайными числами, то нажмите на любую другую цифру");
scanf_s("%d", &state);
```

Далее в работу вступает условный оператор «if», который проверяет значение переменной «state» и если оно равно «1», то программа вызывает функцию «Hand» ручного ввода данных в файл и начинает считывать данные с клавиатуры. Если же пользователь введёт любую другую цифру, то программа запросит ввода с клавиатура минимального и максимального числа из диапазона, и после запустит функцию «Ran» и заполнит файл случайными числами. После чего программа закрывает файл и заканчивает работу.

Фрагмент кода 7.

```
if (state == 1)
    Hand(n, file);
else
{
    printf("Введите максимальное значение числа");
    scanf_s("%lf", &max);
    printf("Введите минимальное значение числа");
    scanf_s("%lf", &min);
    Ran(n, min, max, file);
}
fclose(file);
return 0;
}
```

#### 4.2.2 Структура второй программы

1) Заголовочный файл второй программы «Header.h» содержит в себе объявление 6 функций, для удобной работы с ними.

Фрагмент кода 8.

```
#ifndef __HEADER_H__
#define __HEADER_H__
void PrintMas(int l, double* mas);
void TrowMas(int l, double* masS, double* masW, FILE* f);
void BubbleSort(int l, double* mas);
void InserSort(int l, double* mas);
void QuickSort(int l, double* a);
void qSort(double* mas, int l, int u);
#endif
```

2) Исходный файл «functions.c» содержит в себе описание всех 6 функций, которые используются в программе. Начинается этот исходный файл с объявления всех необходимых для нормальной работы программы функций, а также подключение заголовочного файла.

Фрагмент кода 9.

```
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include "Header.h"
```

После чего начинается поочерёдная реализация всех 6 функций, использующихся в программе. Сначала идёт реализация функции, которая распечатывает текущие данные файлы (на вход подаётся количество элементов в файле и массив, который отображает текущий вид файла). С помощью цикла «for» функция печатает данные на экран.

Фрагмент кода 10.

```
void PrintMas(int l, double* mas)
{
    int i;
    for (i = 0; i < l; i++)
        printf("mas[%d]==%.31f\n", i, mas[i]);
}
```

После чего идёт реализация функции, которая сбрасывает данные в файле к изначальному виду (на вход подаётся количество элементов в файле, массив, хранящий данные о изначальном виде файла, массив, отображающий текущий вид файла и сам файл)

Фрагмент кода 11.

```
void TrowMas(int l, double* masS, double* masW, FILE* f)
{
    int i;
    fprintf_s(f, "n=%d\n", l);
    for (i = 0; i < l; i++)
    {
        fprintf(f, "%1f\n", masS[i]);
        masW[i] = masS[i];
    }
}
```

```
}
```

Далее в этом исходном файле начинается реализация всех функций, отвечающих за сортировку (пузырьком, вставками, быстрой), а также вспомогательной функции, необходимой для запуска при начальных условиях функции, отвечающей за реализацию быстрой сортировки (на вход всех функций подаётся количество элементов в файле, а также массив, с текущими данными файла)

Фрагмент кода 12.

```
void BubbleSort(int l, double* mas)
{
    int i, j;
    double tmp=0.0;
    for (i=0;i<l;i++)
        for (j=0;j<l-i-1;j++)
            if (mas[j] > mas[j+1])
            {
                tmp = mas[j];
                mas[j] = mas[j+1];
                mas[j+1] = tmp;
            }
}

void InserSort(int l, double* mas)
{
    int i, j;
    double tmp;
    for (i = 0; i < l; i++)
    {
        tmp = mas[i];
        for (j = i - 1; j >= 0; j--)
            if (mas[j] > tmp)
                mas[j + 1] = mas[j];
            else
                break;
        mas[j + 1] = tmp;
    }
}

void qSort(double* mas, int l, int u)
{
    int i = l;
    int j = u;
    double tmp = 0;
    double x = mas[(int)((l + u) / 2)];
    do
    {
        while (mas[i] < x)
            ++i;
        while (mas[j] > x)
            --j;
        if (i <= j)
        {
            tmp = mas[i];
            mas[i] = mas[j];
            mas[j] = tmp;
            i++;
            j--;
        }
    } while (i < j);
    if (l < j)
```

```

        qSort(mas, l, j);
    if (i < u)
        qSort(mas, i, u);
}
void QuickSort(int l, double* a)
{
    qSort(a, 0, l - 1);
}

```

3) Исходный файл «main.c» также начинается с объявления всех необходимых библиотек.

Фрагмент кода 13.

```

#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include "Header.h"
#include <time.h>
#define _CRT_SECURE_NO_WARNINGS_

```

После чего начинается главная функция «main», в которой первоначально идёт объявление всех необходимых глобальных переменных и установка русской локализации.

Фрагмент кода 14.

```

int main()
{
    FILE* file;
    clock_t t1, t2;
    long int N, i;
    int state=1, sort=0;
    double *WorkMas, *SaveMas;
    N = 0; i = 0;
    setlocale(LC_ALL, "Rus");
}

```

Далее идёт открытие файла для чтения, считывание из него количества элементов, выделение памяти для массивов, один из которых сохраняет изначальный вид файла, а другой текущий. Также после выделения памяти оба массива заполняются элементами из файла с помощью цикла «for», и закрытие файла.

Фрагмент кода 15.

```

fopen_s(&file, "..\\file.txt", "r");
fscanf_s(file, "n=%d\n", &N);
WorkMas = (double*)malloc(N * sizeof(double));
SaveMas = (double*)malloc(N * sizeof(double));
for (i = 0; i < N; i++)
{
    fscanf_s(file, "%lf\n", &WorkMas[i]);
    SaveMas[i] = WorkMas[i];
}
fclose(file);

```

После этого запускается цикл «while», который выдаёт пользователю список команд, которые возможно совершить, до того момента, пока пользователь не захочет выйти из программы. Для этого используется переменная «state», которая каждый раз считывается с клавиатуры и определяет, какое действие будет совершено.

Фрагмент кода 16.

```

while (state!=0)
{
    printf("Если хотите напечатать массив, введите '1'\n");
    printf("Если хотите отсортировать массив, введите '2'\n");
    printf("Если хотите сбросить массив к изначальному виду, введите '3'\n");
}

```



```
printf("Если хотите Выйти из программы, введите '0'\nВвод:");
scanf_s("%d", &state);
```

Далее в работу вступает условный оператор «if», с помощью которого переменная «state» сравнивается с 4 числами и определяет дальнейшее действие. Если «state» равна «1», то запускается функция печати текущего вида файла. Если «2», то программа спросит у пользователя, какую сортировку он хочет провести, быструю, вставками или пузырьком, если же пользователь введёт неправильную цифру, то программа будет спрашивать о выборе сортировки до тех пор, пока пользователь не определится с выбором (определение идёт с помощью считывания с клавиатуры значения переменной «sort»). После выбора запустится соответствующая сортировка, а также засечётся время выполнения этой сортировки. Потом файл откроется для записи, в него сразу будет добавлено количество элементов, после чего запишется отсортированный массив. После этого файл закроется и на экран будет выведено время сортировки. Если «3», то файл снова откроется для записи и с помощью массива, хранящего изначальный вид файла и соответствующей функции файл будет приведён к исходному виду, после чего закрыт. Если пользователь введёт «0», то программа очистит динамическую память массивов и завершит работу. При вводе некорректных данных на экран выведется соответствующее информационное сообщение.

Фрагмент кода 17.

```
if (state == 1)
    PrintMas(N, WorkMas);
else if (state == 2)
{
    while ((sort != 1) && (sort != 2) && (sort != 3))
    {
        printf("Какую сортировку провести? Пузырьком-'1', Вставкой-'2', Быструю-'3'\nВвод:");
        scanf_s("%d", &sort);
    }
    if (sort == 1)
    {
        t1 = clock();
        BubbleSort(N, WorkMas);
        t2 = clock();
    }
    if (sort == 2)
    {
        t1 = clock();
        InsertSort(N, WorkMas);
        t2 = clock();
    }
    if (sort == 3)
    {
        t1 = clock();
        QuickSort(N, WorkMas);
        t2 = clock();
    }
    fopen_s(&file, "..\\file.txt", "w");
    fprintf(file, "n=%d\n", N);
    for (i = 0; i < N; i++)
        fprintf(file, "%lf\n", WorkMas[i]);
    fclose(file);
    sort = 0;
    printf("Массив отсортирован, время сортировки составило %lf\n",
        ((double)(t2 - t1)) / CLOCKS_PER_SEC);
}
else if (state == 3)
{
    fopen_s(&file, "..\\file.txt", "w");
```

```

        TrowMas(N, SaveMas, WorkMas, file);
        fclose(file);
        printf("Файл был приведён к изначальному виду\n");
    }
    else if (state == 0)
    {
        printf("Программа завершила работу\n");
        break;
    }
    else
        printf("Введите корректные данные\n");
        printf("\n");
    }
    free(WorkMas);
    free(SaveMas);
    return 0;
}

```

### 4.3 Описание алгоритмов

Обе программы полностью состоят из разных алгоритмов, которые реализуют ту или иную задачу, для выполнения основной задачи. Далее будут рассмотрены самые интересные алгоритмы, а именно три вида сортировок из второй программы и алгоритм заполнения файла случайными числами из первой программы.

#### 4.3.1 Описание алгоритма заполнения файла случайными числами

Алгоритм реализован отдельной функцией «Ran» в исходном файле «functions.c». На вход функции подаётся количество элементов, которое пользователь хочет видеть в файле, минимальное и максимальное число диапазона, а также сам файл. После запуска функции создаётся локальная переменная «i», которая будет являться итератором цикла «for», который запускается сразу после. С помощью функции «fprintf» Цикл поочерёдно заполняет каждую строку файла случайным числом из диапазона, заданного пользователем (с помощью библиотеки «time.h» и использовании параметра «time(0)», в качестве аргумента функции «srand()», числа с каждым новым созданием файла получаются абсолютно случайные) (См Рис.18)

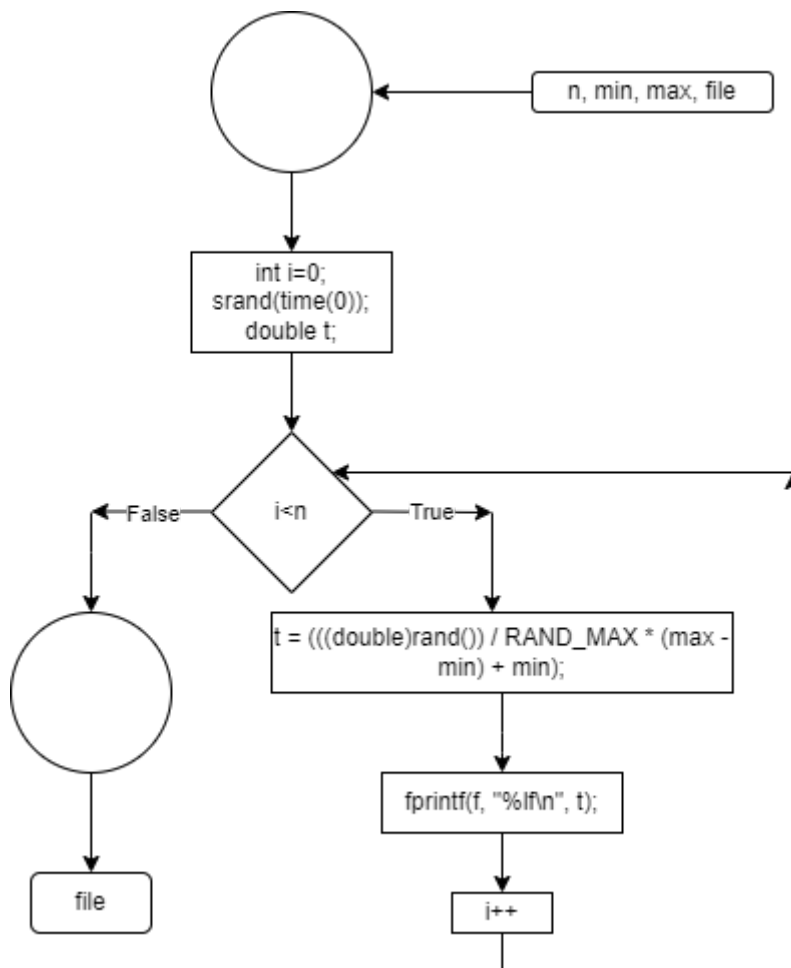


Рисунок 18. Блок-схема алгоритма заполнения массива.

Фрагмент кода 18.

```
void Ran(int n, int min, int max, FILE* f)
{
    double t;
    srand(time(0));
```

```

for (int i = 0; i < n; i++)
{
    t = (((double)rand()) / RAND_MAX * (max - min) + min);
    fprintf(f, "%lf\n", t);
}
}

```

#### 4.3.2 Описание алгоритма сортировки пузырьком.

Сортировка пузырьком является самой простой для понимания из сортировок, представленных в данной лабораторной работе. Посредством сравнения и заменой местами поочерёдно всего файла самые маленькие элементы всплывают вверх, как пузырёк воздуха в воде. Сортировка реализована в отдельной функции «BubbleSort» в исходном файле «functions.c». На вход функция получает количество элементов в файле и массив, отображающий текущий вид файла. Начинается алгоритм с создания двух итераторов циклов, «i» и «j». Также создаётся локальная переменная «tmp», необходимая для того, чтобы менять местами соседние ячейки массива. После запускается цикл «for» с итератором «i» который изменяется от 0 до длины массива (l). Сразу же в этом цикле запускается вложенный цикл «for» с итератором «j», который изменяется от 0 до (l-i-1), то есть для каждой итерации первого цикла второй цикл доходит до номера элемента, этой итерации. Далее с помощью условного оператора «if» Проверяется, не оказался ли текущий элемент массива больше следующего, и если так происходит, то с помощью переменной «tmp» осуществляется замена значений ячеек массива, а именно, сначала значение переменной становится равно значению текущей ячейки массива, после значение текущей ячейки заменяется следующим, а после значение следующей ячейки заменяется значением переменной «tmp» Так алгоритм проходит по всему массиву и сортирует его. (См Рис.19)

Фрагмент кода 19.

```

void BubbleSort(int l, double* mas)
{
    int i, j;
    double tmp=0.0;
    for (i=0;i<l;i++)
        for (j=0;j<l-i-1;j++)
            if (mas[j] > mas[j+1])
            {
                tmp = mas[j];
                mas[j] = mas[j+1];
                mas[j+1] = tmp;
            }
}

```

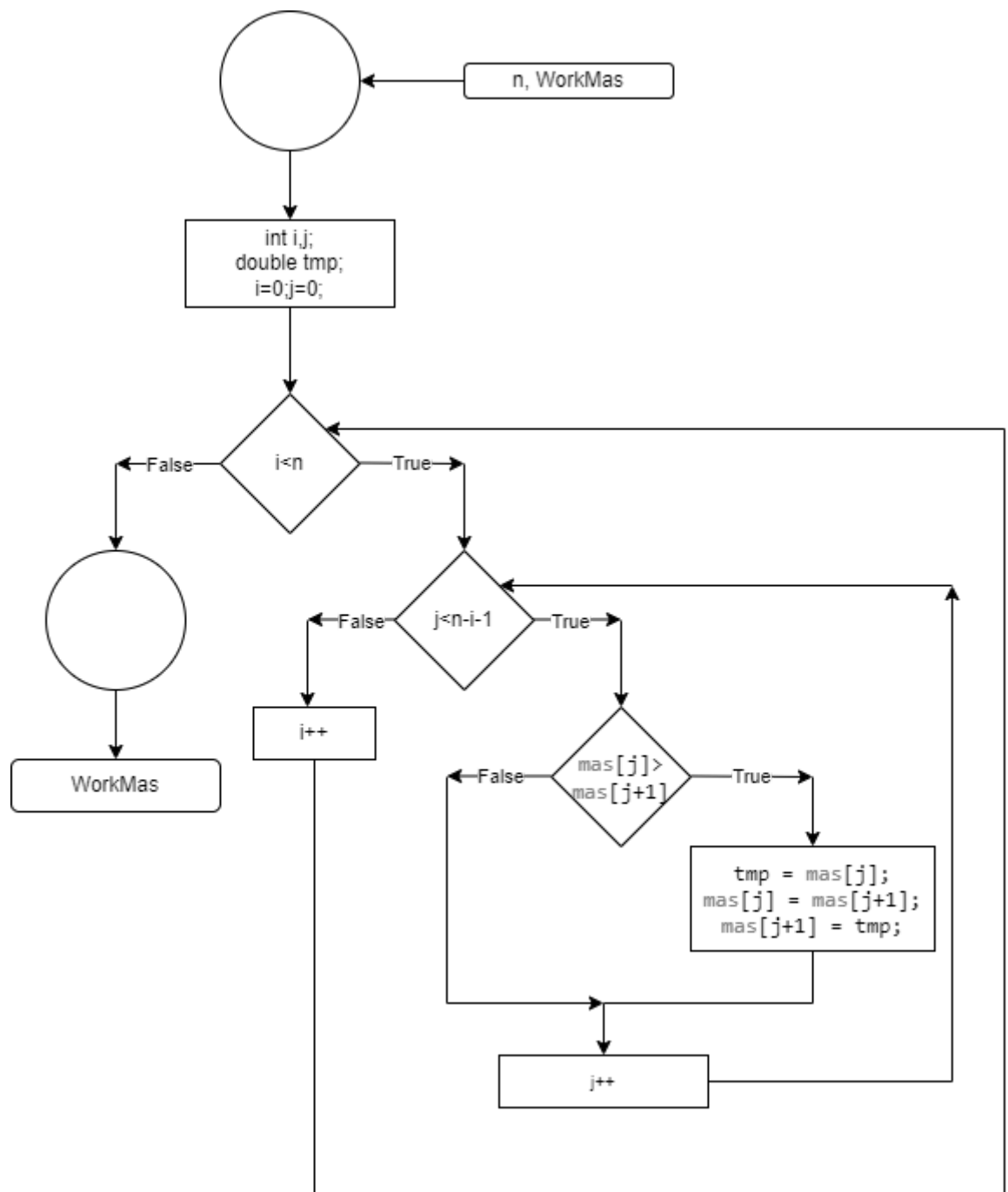


Рисунок 19. Блок-схема алгоритма сортировки пузырьком.

### 4.3.3 Описание алгоритма сортировки вставками

Сортировка вставками чем-то напоминает сортировку пузырьком, ведь здесь тоже используются 2 вложенных цикла «for». Вначале также создаются 2 итератора для циклов «i» и «j» и переменная «tmp», которая будет хранить в себе данные ячейки массива, которую мы сортируем на определённой итерации. После запускается цикл «for» с итератором «i», который изменяется от 0 до количества чисел в массиве. На каждой итерации значение текущей ячейки массива помещается в переменную «tmp», после чего запускается второй цикл «for», с итератором «j», который изменяется от «i-1» до нуля. Если значение текущей ячейки массива больше «tmp», то следующий элемент массива становится равным текущему, иначе второй цикл прекращает работу и следующей ячейке массива приравнивается значение переменной «tmp». Получается, что данный вид сортировки разбивает массив на левую отсортированную часть и правую, несортированную. После чего берётся поочерёдно каждый элемент неотсортированной части массива и с помощью сравнений находится место, чтобы слева от него все числа были меньше, а справа больше. Таким образом постепенно переносятся все числа из правой части в левую, вставляя на определённые места и массив сортируется. (См Рис.20)

Фрагмент кода 20.

```
void InserSort(int l, double* mas)
{
    int i, j;
    double tmp;
    for (i = 0; i < l; i++)
    {
        tmp = mas[i];
        for (j = i - 1; j >= 0; j--)
            if (mas[j] > tmp)
                mas[j + 1] = mas[j];
            else
                break;
        mas[j + 1] = tmp;
    }
}
```

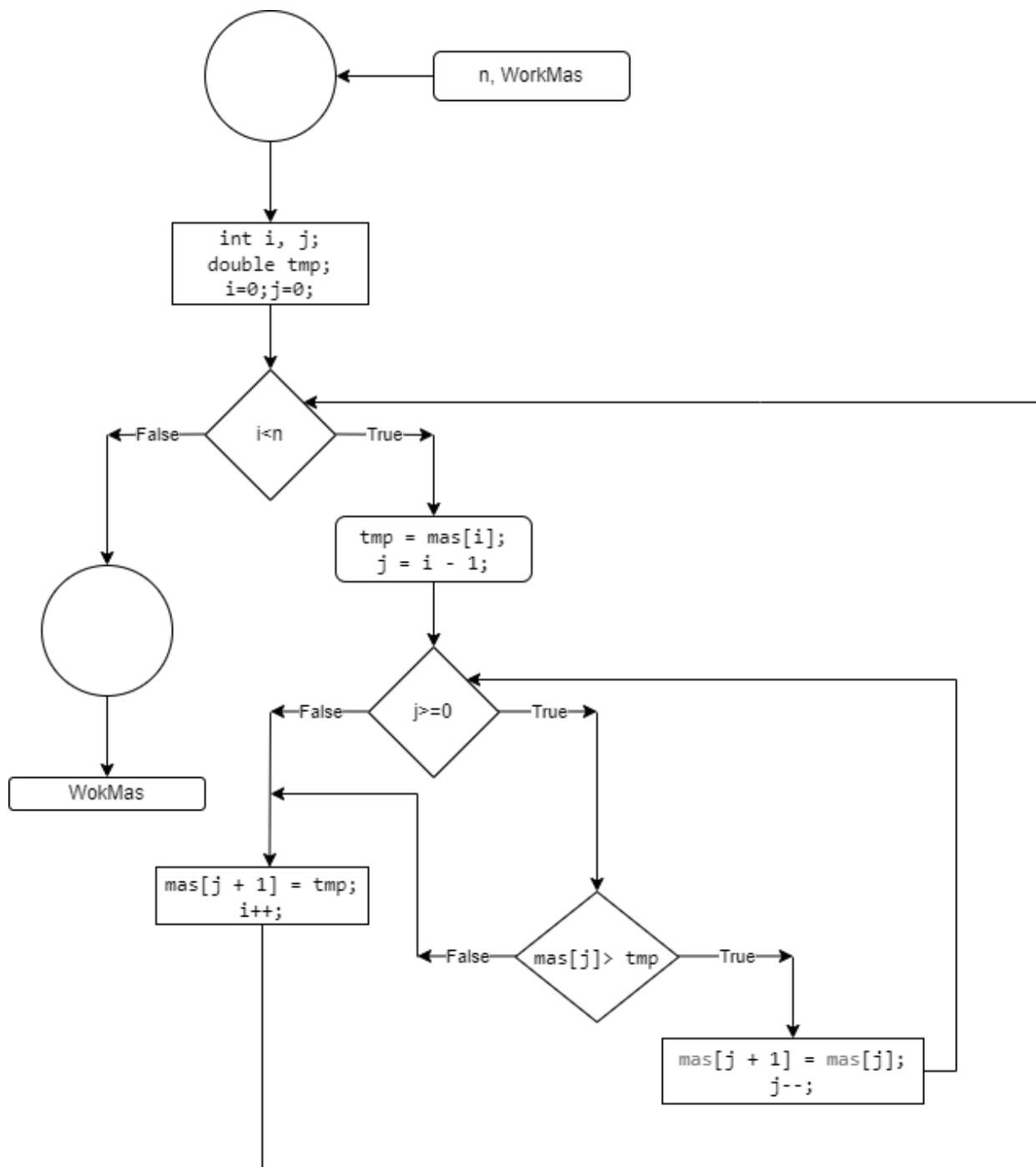


Рисунок 20. Блок-схема алгоритма сортировки вставками.

#### 4.3.4. Описание алгоритма быстрой сортировки

Быстрая сортировка парадоксально является самой быstroдействующей из представленных в данной лабораторной работе. Благодаря рекурсивному алгоритму, а также используя в некоторой степени принцип вложенных отрезков, этот способ сортировки способен упорядочить массив любых размеров практически мгновенно. Происходит это потому что по сути каждый раз массив делится пополам и не нужно использовать вложенные циклы, затрачивающие много времени и ресурсов. На вход алгоритма всё также подаётся количество элементов «n» и массив, отображающий текущий вид файла. Для запуска на начальных условиях используется функция «QuickSort», в которую собственно и поступает количество элементов и массив.

Фрагмент кода 21.

```
void QuickSort(int l, double* a)
{
    qSort(a, 0, l - 1);
}
```

Когда запускается основной алгоритм, на вход он получает массив, а также границы отрезка, вначале это 0 и «n-1». Сначала создаются 2 итератора цикла, «i» и «j», также создаётся переменная «tmp», необходимая в дальнейшем для смены элементов местами. Также создаётся переменная, которая равна срединному элементу полученного отрезка. После запускается цикл «while», который работает, пока границы отрезка не сойдутся на центральном элементе. На каждом шаге слева ищется элемент, который больше срединного, а справа меньше срединного. Когда такие элементы находятся, они меняются местами с помощью переменной «tmp», если конечно отрезок не сошёлся на центральном элементе. После того, как алгоритм оставит справа все элементы меньше срединного, а справа больше (скорее всего они будут неупорядоченными), запускается рекурсия данного алгоритма для каждого из двух полученных отрезков. Такой цикл рекурсий продолжается до тех пор, пока принципом вложенных отрезков не дойдёт до одноэлементного отрезка. После этого массив будет отсортирован. (См рис.21)

Фрагмент кода 22.

```
void qSort(double* mas, int l, int u)
{
    int i = l;
    int j = u;
    double tmp = 0;
    double x = mas[(int)((l + u) / 2)];
    do
    {
        while (mas[i] < x)
            ++i;
        while (mas[j] > x)
            --j;
        if (i <= j)
        {
            tmp = mas[i];
            mas[i] = mas[j];
            mas[j] = tmp;
            i++;
            j--;
        }
    } while (i < j);
    if (l < j)
        qSort(mas, l, j);
    if (i < u)
        qSort(mas, i, u);
}
```



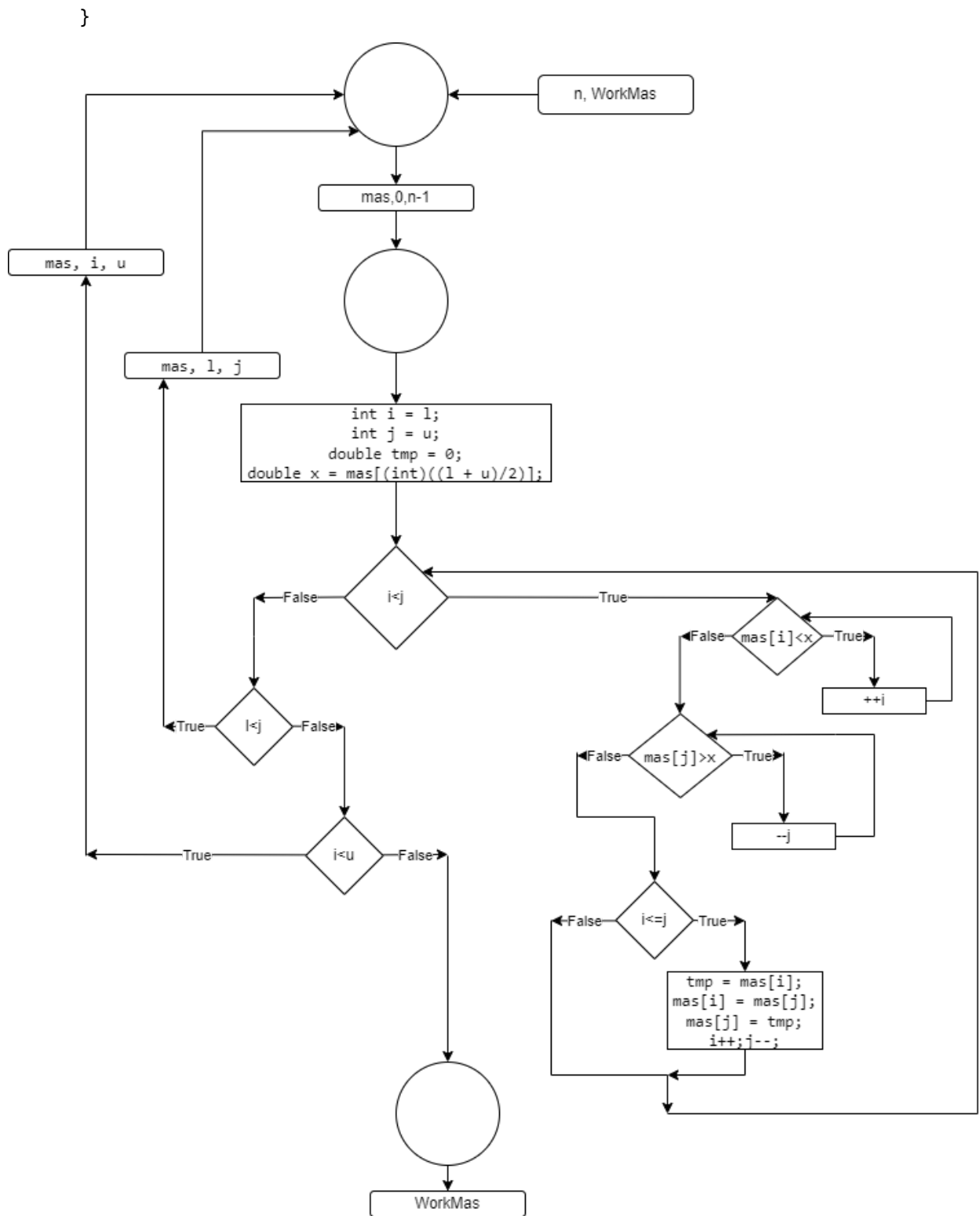
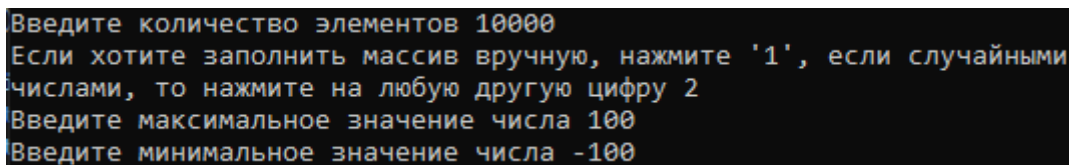


Рисунок 21. Блок-схема алгоритма быстрой сортировки.

## 5. Эксперименты

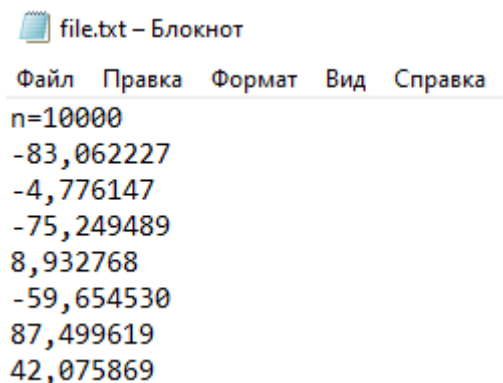
Главным критерием любой сортировки является время её выполнения, ведь именно от того, насколько быстро выполнится сортировка зависит и время выполнения программы, в которой эта сортировка необходима и реализована. Для проведения экспериментов будут созданы поочерёдно 3 текстовых файла с разным количеством элементов с помощью первой программы и отсортированы каждым методом (после каждой сортировки файл будет возвращён к изначальному виду, чтобы сортировки проводились для одинаковых изначальных данных). Очевидно, что для совсем маленьких объёмов данных все сортировки будут работать одинаково быстро и практически мгновенно, поэтому эксперименты будут проводиться для 10000, 100000 и 500000 элементов, которыми будут являться случайные числа в диапазоне от -100 до 100. Для удобства данные будут занесены в таблицу. Предварительно можно сделать выводы, что быстрая сортировка окажется быстрее всех по времени, ведь время её выполнения составляет величину  $O(n \log(n))$ , ведь по сути каждый раз части массива делятся на половины, в то время как для сортировки пузырьком и вставками время выполнения составляет величину  $O(n^2)$ , из-за вложенных циклов.

Сначала запускается первая программа и создаётся файл, потом запускается вторая программа, сортируется файл, засекается время, после чего файл сбрасывается к начальным данным и производятся другие сортировки. Так повторяется для 3 разных количеств элементов. (См Рис.22), (См Рис.23)



```
Введите количество элементов 10000
Если хотите заполнить массив вручную, нажмите '1', если случайными
числами, то нажмите на любую другую цифру 2
Введите максимальное значение числа 100
Введите минимальное значение числа -100
```

Рисунок 22. Запуск первой программы и заполнение файла.



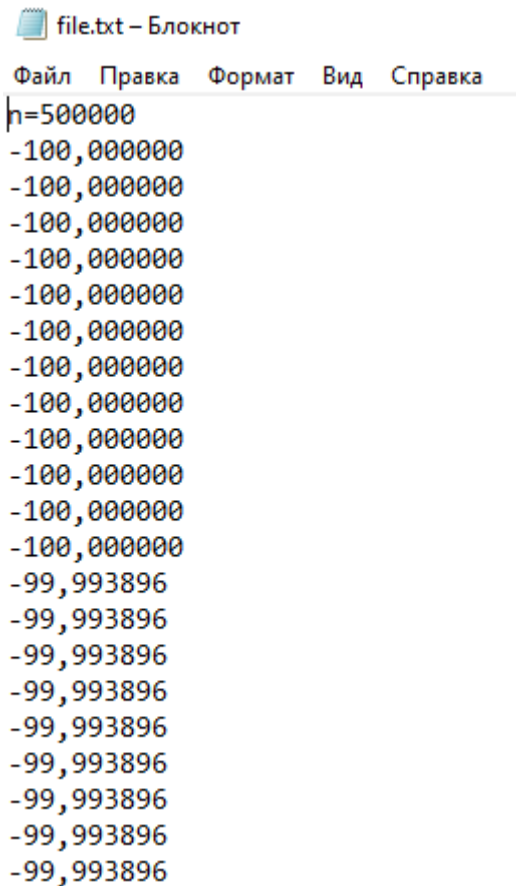
```
file.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
n=10000
-83,062227
-4,776147
-75,249489
8,932768
-59,654530
87,499619
42,075869
```

Рисунок 23. Вид файла при заполнении случайными числами.

Таблица 1. Сравнение времени работы сортировок

Количество элементов	Bubble sort	Insert sort	Quick sort
10000	0,230 с	0,056 с	0,001 с
100000	25.768 с	5.541 с	0,011 с
500000	662.188 с	138.561 с	0,067 с

В результате проведённых экспериментов с разным количеством элементов в файлах видно, что с самых малых объёмов данных сортировка пузырьком работает в десятки раз медленнее других. Быстрее неё работает сортировка вставками. И меньше всего на сортировку затрачивает быстрая сортировка. После каждой из сортировок файл упорядочен и удобен в дальнейшем использовании. (См Рис.24)



```
file.txt – Блокнот
Файл  Правка  Формат  Вид  Справка
n=500000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-100,000000
-99,993896
-99,993896
-99,993896
-99,993896
-99,993896
-99,993896
-99,993896
-99,993896
-99,993896
-99,993896
```

Рисунок 24. Файл после выполнения сортировки.

Любая из приведённых в лабораторной работе сортировок справляется с задачей и позволяет упорядочить данные в файле. Сортировка пузырьком и вставками достаточно медленны, но могут быть применимы при небольших объёмах данных, ведь они просты и понятны в использовании. Но если объём данных слишком большой, то лучше всего использовать быструю сортировку, ведь, как видно из экспериментов, она всегда отрабатывает практически мгновенно, затрачивая на своё выполнение даже меньше секунды времени.

## 6. Заключение

В результате проведённой лабораторной работы на языке программирования «С» были написаны 2 программы, которые полностью выполняют поставленную задачу, «Первая программа создает текстовый файл с записанными в него числами. Программа принимает количество чисел  $n$ , максимальное и минимальное значение. Вторая программа читает текстовый файл с набором чисел, выводит консольный интерфейс (печать, сортировка, сброс, выход), выполняет выбранные действия» интерфейс которых прост и понятен в использовании. Как и сказано в задаче, первая программа создаёт файл и заполняет его числами, а вторая предоставляет пользователю выбор действий и при необходимости сортирует данные файла (реализованы быстрая сортировка, сортировка вставками и пузырьком).

В результате проведённых экспериментов можно сделать выводы, насчёт методов сортировок (пузырьком, быстрой, вставками). Самой быстродействующей сортировкой является быстрая. Сортировка пузырьком оказалась самой медленной, даже при 10000 элементах она затрачивает на работу в десятки раз больше времени. Сортировка вставками выполняется быстрее, чем пузырьковая. Но сортировки пузырьком и вставками гораздо проще в написании и понимании. Если необходимо отсортировать большой объём данных, то быстрая сортировка справится лучше всех, но если речь заходит о каких-то малых объёмах данных, то прекрасно справляются и 2 других метода сортировки, которые проще в понимании и быстрее в написании. Каждая сортировка может найти применение, ведь когда-то важно количество времени, затраченное на написание алгоритма, а когда-то на работу программы.

Сортировки, это очень важная часть любого языка программирования, ведь работать с упорядоченными данными гораздо быстрее и проще. В результате лабораторной работы были изучены 3 метода сортировок, которые найдут применение в любой ситуации. Вместе с сортировками в программировании есть и другие интересные и полезные алгоритмы, упрощающие выполнение задачи, поэтому в дальнейшем необходимо продолжить изучение структур языка программирования «С», его алгоритмов и составляющих, чтобы писать более сложные программы и реализовывать усовершенствованные алгоритмы для исполнения каких-либо потребностей.

## 7. Литература

1. Т.А. Павловская Учебник по программированию на языках высокого уровня(C/C++) – Режим доступа: <http://cph.phys.spbu.ru/documents/First/books/7.pdf>
2. Бьерн Страуструп. Язык программирования C++ - Режим доступа: [http://8361.ru/6sem/books/Strastrup-Yazyk\\_programmirovaniya\\_c.pdf](http://8361.ru/6sem/books/Strastrup-Yazyk_programmirovaniya_c.pdf)

# Приложение

## Приложение 1

Заголовочный файл «Header.h» первой программы

```
#ifndef __HEADER_H__
#define __HEADER_H__
void Hand(int n, FILE* f);
void Ran(int n, int min, int max, FILE* f);
#endif
```

Исходный файл «functions.c» первой программы

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include "Header.h"
#include <time.h>
void Hand(int n, FILE* f)
{
    double t;
    for (int i = 0; i < n; i++)
    {
        scanf_s("%lf",&t);
        fprintf(f, "%lf\n",t);
    }
}
void Ran(int n, int min, int max, FILE* f)
{
    double t;
    srand(time(0));
    for (int i = 0; i < n; i++)
    {
        t = (((double)rand()) / RAND_MAX * (max - min) + min);
        fprintf(f, "%lf\n", t);
    }
}
```

Исходный файл «main.c» первой программы

```
#include <stdio.h>
#include <stdlib.h>
#include <locale.h>
#include "Header.h"
#include <time.h>
int main()
{
    FILE *file;
    long int n = 0;
    int state = 0;
    double min=0, max=0;
    setlocale(LC_ALL, "Rus");
    fopen_s(&file, "..\\file.txt", "w");
    printf("Введите количество элементов");
    scanf_s("%d", &n);
    fprintf_s(file,"n=%d\n", n);
    printf("Если хотите заполнить массив вручную, нажмите '1', если случайными числами, то нажмите на любую другую цифру");
    scanf_s("%d", &state);
    if (state == 1)
        Hand(n,file);
    else
    {
        printf("Введите максимальное значение числа");
```

```
scanf_s("%lf", &max);  
printf("Введите минимальное значение числа");  
scanf_s("%lf", &min);  
Ran(n,min,max,file);  
}  
fclose(file);  
return 0;  
}
```

## Приложение 2

Заголовочный файл «Header.h» второй программы

```
#ifndef __HEADER_H__
#define __HEADER_H__
void PrintMas(int l, double* mas);
void TrowMas(int l, double* masS, double* masW, FILE* f);
void BubbleSort(int l, double* mas);
void InserSort(int l, double* mas);
void QuickSort(int l, double* a);
void qSort(double* mas, int l, int u);
#endif
```

Исходный файл «functions.c» второй программы

```
#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include "Header.h"
void PrintMas(int l, double* mas)
{
    int i;
    for (i = 0; i < l; i++)
        printf("mas[%d]==%.31f\n", i, mas[i]);
}
void TrowMas(int l, double* masS, double* masW, FILE* f)
{
    int i;
    fprintf_s(f, "n=%d\n", l);
    for (i = 0; i < l; i++)
    {
        fprintf(f, "%1f\n", masS[i]);
        masW[i] = masS[i];
    }
}
void BubbleSort(int l, double* mas)
{
    int i, j;
    double tmp=0.0;
    for (i=0;i<l;i++)
        for (j=0;j<l-i-1;j++)
            if (mas[j] > mas[j+1])
            {
                tmp = mas[j];
                mas[j] = mas[j+1];
                mas[j+1] = tmp;
            }
}
void InserSort(int l, double* mas)
{
    int i, j;
    double tmp;
    for (i = 0; i < l; i++)
    {
        tmp = mas[i];
        for (j = i - 1; j >= 0; j--)
            if (mas[j] > tmp)
                mas[j + 1] = mas[j];
            else
                break;
        mas[j + 1] = tmp;
    }
}
```



```

void qSort(double* mas, int l, int u)
{
    int i = l;
    int j = u;
    double tmp = 0;
    double x = mas[(int)((l + u) / 2)];
    do
    {
        while (mas[i] < x)
            ++i;
        while (mas[j] > x)
            --j;
        if (i <= j)
        {
            tmp = mas[i];
            mas[i] = mas[j];
            mas[j] = tmp;
            i++;
            j--;
        }
    } while (i < j);
    if (l < j)
        qSort(mas, l, j);
    if (i < u)
        qSort(mas, i, u);
}
void QuickSort(int l, double* a)
{
    qSort(a, 0, l - 1);
}

```

Исходный файл «main.c» второй программы

```

#include <stdio.h>
#include <locale.h>
#include <stdlib.h>
#include "Header.h"
#include <time.h>
#define _CRT_SECURE_NO_WARNINGS_
int main()
{
    FILE* file;
    clock_t t1, t2;
    long int N, i;
    int state=1, sort=0;
    double *WorkMas, *SaveMas;
    N = 0; i = 0;
    setlocale(LC_ALL, "Rus");
    fopen_s(&file, "..\\file.txt", "r");
    fscanf_s(file, "n=%d\n", &N);
    WorkMas = (double*)malloc(N * sizeof(double));
    SaveMas = (double*)malloc(N * sizeof(double));
    for (i = 0; i < N; i++)
    {
        fscanf_s(file, "%lf\n", &WorkMas[i]);
        SaveMas[i] = WorkMas[i];
    }
    fclose(file);
    while (state!=0)
    {
        printf("Если хотите напечатать массив, введите '1'\n");
        printf("Если хотите отсортировать массив, введите '2'\n");
        printf("Если хотите сбросить массив к изначальному виду, введите '3'\n");
    }
}

```

```

printf("Если хотите Выйти из программы, введите '0'\nВвод:");
scanf_s("%d", &state);
if (state == 1)
    PrintMas(N, WorkMas);
else if (state == 2)
{
    while ((sort != 1) && (sort != 2) && (sort != 3))
    {
        printf("Какую сортировку провести? Пузырьком-'1', Вставкой-'2', Быструю-'3'\nВвод:");
        scanf_s("%d", &sort);
    }
    if (sort == 1)
    {
        t1 = clock();
        BubbleSort(N, WorkMas);
        t2 = clock();
    }
    if (sort == 2)
    {
        t1 = clock();
        InserSort(N, WorkMas);
        t2 = clock();
    }
    if (sort == 3)
    {
        t1 = clock();
        QuickSort(N, WorkMas);
        t2 = clock();
    }
    fopen_s(&file, "..\\file.txt", "w");
    fprintf(file, "n=%d\n", N);
    for (i = 0; i < N; i++)
        fprintf(file, "%lf\n", WorkMas[i]);
    fclose(file);
    sort = 0;
    printf("Массив отсортирован, время сортировки составило %lf\n",
        ((double)(t2 - t1)) / CLOCKS_PER_SEC);
}
else if (state == 3)
{
    fopen_s(&file, "..\\file.txt", "w");
    TrowMas(N, SaveMas, WorkMas, file);
    fclose(file);
    printf("Файл был приведён к изначальному виду\n");
}
else if (state == 0)
{
    printf("Программа завершила работу\n");
    break;
}
else
    printf("Введите корректные данные\n");
    printf("\n");
}
free(WorkMas);
free(SaveMas);
return 0;
}

```