Министерство образования и науки РФ

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

**Отчет по учебной практике**

**Прототип файлового менеджера**

Выполнила: Лютова Татьяна Сергеевна

Студент группы 381806-1

Проверил: доцент кафедры МОСТ Кустикова В.Д.

Нижний Новгород, 2018

**Содержание:**

[**Введение** 3](#_Toc533555443)

[**1.** **Постановка задачи** 4](#_Toc533555444)

[**2.** **Руководство пользователя** 5](#_Toc533555445)

[**3.** **Руководство программиста** 7](#_Toc533555446)

[**3.1** **Структура программы** 7](#_Toc533555447)

[**3.2** **Описание алгоритма** 7](#_Toc533555448)

[ Сортировка выбором 7](#_Toc533555449)

[ Сортировка вставками 7](#_Toc533555450)

[ Пузырьковая сортировка 8](#_Toc533555451)

[ Сортировка подсчетом 9](#_Toc533555452)

[ Сортировка Хоара (быстрая сортировка) 10](#_Toc533555453)

[ Сортировка слиянием 11](#_Toc533555454)

[**3.3 Описание функций** 13](#_Toc533555455)

[**Заключение** 16](#_Toc533555456)

# **Введение**

21 век – век технологий. И в наше время людям ежедневно приходится работать с компьютерами (дома, на работе или на учебе). Большинство из нас сталкивалось с проблемой поиска необходимого документа в огромном количестве файлов.

Сортировки – это удобный метод упорядочить данные по определенному признаку. С их помощью людям будет гораздо проще работать с документами, затрачивая при этом минимальное количество времени.

С этой целью был создан прототип файлового менеджера, который может отсортировать шестью различными способами необходимые файлы по возрастанию или убыванию размеров.

# **Постановка задачи**

**Задача:**

Реализовать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию или убыванию размера.

**Входные данные:**

1. Путь до директория, в котором нужно отсортировать содержимое.
2. Метод сортировки
   1. Сортировка выбором
   2. Сортировка вставками
   3. Сортировка подсчетом
   4. Пузырьковая сортировка
   5. Быстрая сортировка
   6. Сортировка слиянием

**Выходные данные:**

1. Отсортированный список имен файлов с указанием размеров.
2. Время сортировки.

# **Руководство пользователя**

В данном пункте продемонстрирована инструкция по работе с программой.

* Запустив программу, вы увидите окно (рис.1), на данном шаге вам нужно ввести путь к папке, файлы которой вам необходимо отсортировать и нажать Enter.

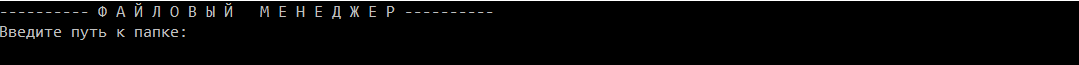


Рис.1

* Указав путь и нажав Enter, вы увидите на экране файлы, содержащиеся в папке, которую вы выбрали (рис.2)

Далее вам будет предложено выбрать 1 из 6 вариантов сортировки или выйти из программы  
На этом этапе вам необходимо ввести цифру необходимого вам пункта и нажать Enter.

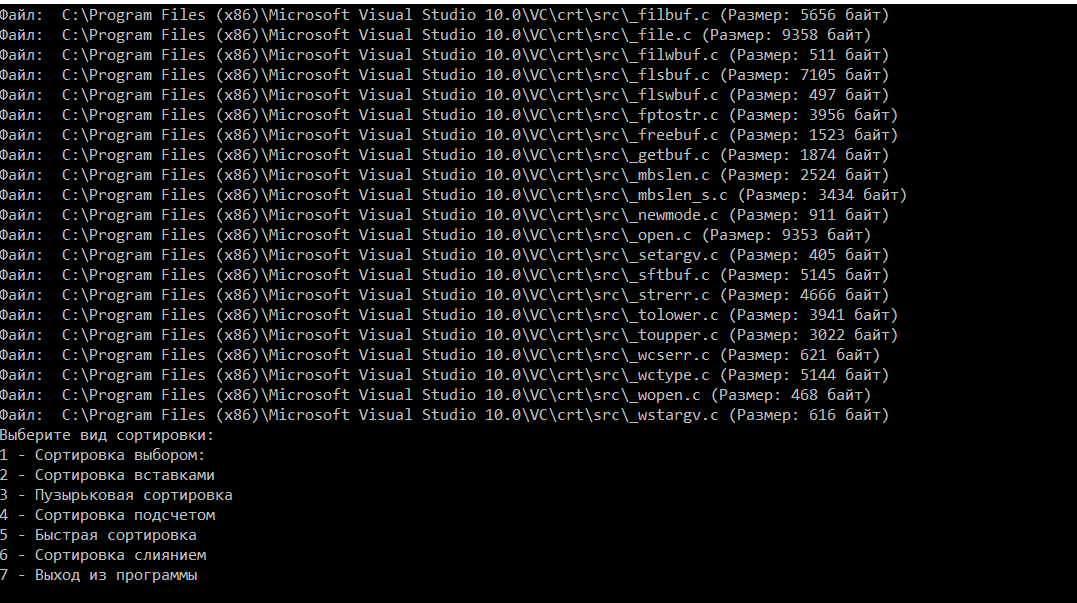


Рис.2

Для наглядности приведем пример и выполним сортировку выбором.

Вводим цифру 1и нажимаем Enter.

На экране будут выведены файлы, отсортированные по возрастанию размера и время, за которое была выполнена сортировка (в данном примере понадобилось 0,003 сек), а также сообщение с предложением отсортировать файлы другим способом или выйти (рис.3)

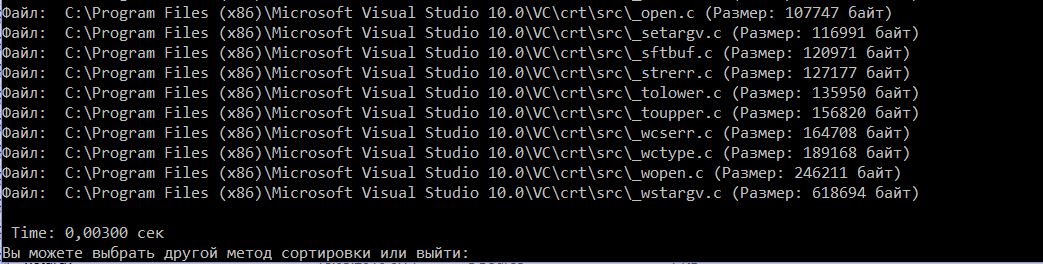


Рис.3

* Если же вы введете цифру, которая меньше (равна) нулю или больше семи, то программа выведет ошибку и выведет исходные неотсортированные файлы. В данном случае вам стоит ввести другое число в диапазоне от 1 до 7. (рис.4)

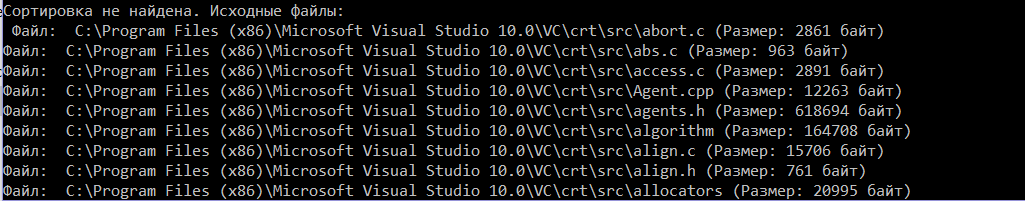


Рис.4

# **Руководство программиста**

## **3.1 Структура программы**

Код программы находится в одном файле main.c

## **3.2 Описание алгоритма**

Сортировка файлов осуществляется одним из шести видов сортировки

### Сортировка выбором

*Принцип работы.*

Проходя по массиву, находим минимальный элемент. Меняем его местами с крайним левым (нулевым). Далее считаем, что эта часть массива уже отсортирована и работаем с другой частью. Снова выбираем минимальный элемент (из всех элементов кроме нулевого) и меняем его с 1 элементом. Продолжаем так до тех пор, пока весь массив не будет отсортирован.

*Пример:*

Нам дан массив, состоящий из 6 элементов. (рис.5) Мы находим минимальный (в данном случае это четвертый элемент) и меняем его с нулевым. Далее рассмотрим подмассив от 1- го до 5- го элемента. Минимальный элемент равен пятому. Значит меняем пятый элемент с 1 (шаг 0). Аналогично проходим по всему массиву (1, 2, 3 шаги).

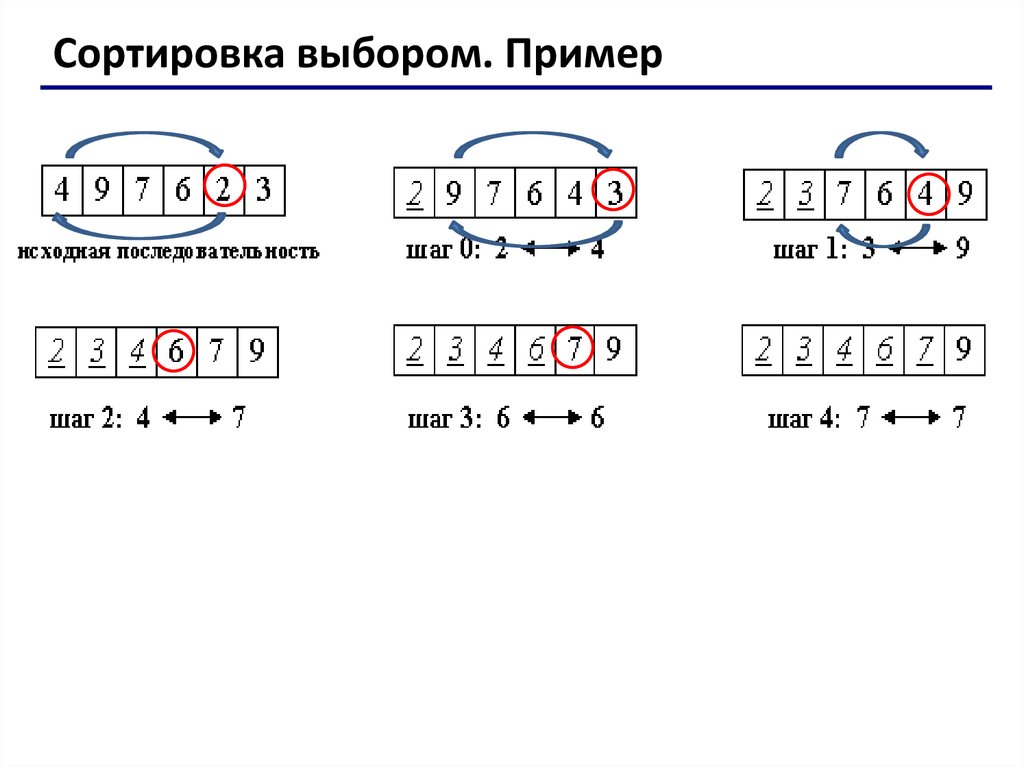


Рис.5

На 4 шаге мы получаем отсортированный по возрастанию массив.

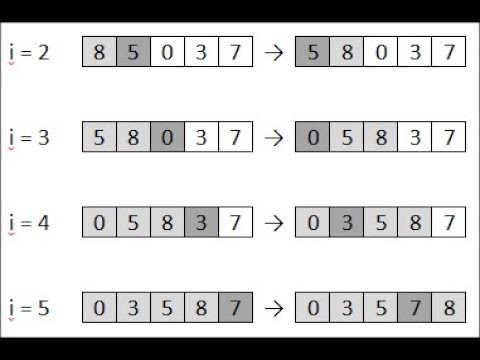
### Сортировка вставками

*Принцип работы:*

Массив можно условно разделить на две части — отсортированная часть и неотсортированная. В начале сортировки первый элемент массива считается отсортированным, все остальные — нет. Рассмотрим второй элемент, если он меньше первого, то они меняются местами, если нет, то идем дальше по массиву. И продолжаем так пока не отсортируем весь массив. Таким образом, за один шаг сортировки отсортированная часть массива увеличивается, а неотсортированная часть массива уменьшается на один элемент.

*Пример:*

Нам дан массив, состоящий из 5 элементов. Считаем, что нулевой элемент уже отсортирован. Первый элемент (5) меньше нулевого, значит меняем их местами (Шаг 1). Рассмотрим второй элемент, он меньше первого, меняем местами с первым и меньше нулевого, значит меняем и с нулевым (Шаг 2). Далее проделываем аналогичные операции, пока весь массив не будет отсортирован.

.

ШАГ 1

ШАГ 2

ШАГ 3

ШАГ 4

Рис.6

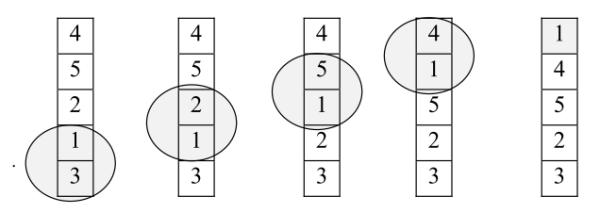
### Пузырьковая сортировка

*Принцип работы:*

Идея метода состоит в том, чтобы попарно сравнивать соседние элементы. Каждый проход начинается с начала последовательности. Сравнивается нулевой элемент с первым: если порядок между ними нарушен, то они меняются местами. Затем сравниваются первый со вторым, второй с третьим и так далее до конца массива; элементы с неправильным порядком в паре меняются местами. В итоге, после первого прохода, максимальный (или минимальный, в зависимости от вида сортировки: по возрастанию/по убыванию) элемент будет находится на последнем месте в массиве, он как бы “всплывет” наверх. На следующем проходе рассматривается последовательность от 1 до N-1, затем от 1 до N-2, и так до конца.

*Пример:*

Нам дан массив, состоящий из 5 элементов. Отсортируем массив по убыванию. Рассмотрим первый подход. Нулевой элемент меньше второго, значит меняем их местами (1 столбец). Далее смотрим первый элемент и второй, порядок снова нарушен, меняем их местами (2 столбец) и так далее до конца массива, пока наименьший элемент не будет наверху. На этом 1 подход закончен. Далее алгоритм аналогичен, продолжаем его до тех пор, пока весь массив не будет отсортирован.

**

**1 2 3 4 5**

Рис.7

### Сортировка подсчетом

*Принцип работы:*

Создадим дополнительный массив, размер которого равен разности между максимальным и минимальным элементом исходного массива. Заполняем это массив нулями. В процессе сортировки на каждом i-ом проходе мы попарно сравниваем i-й элемент со всеми элементами исходного массива. Если установлено, что mass[i]> mass[j], то увеличиваем счетчик К на единицу. По окончании текущего прохода счетчик К указывает количество элементов, меньших, чем mass[i], поэтому элемент mass[i] занимает в отсортированной последовательности позицию К + 1.

*Пример:*

Нам дан массив, состоящий из 5 элементов {2, 1, 0, 1, 2}. Создаем дополнительный массив, состоящий из 5 элементов {0, 0, 0, 0, 0, 0} (Рис. 8).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 0 | 1 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 |

Рис.8

Проходим весь массив от нулевого элемента до последнего. Нулевой элемент равен двум, значит в дополнительный массив мы пишем единицу на месте второго индекса (Рис. 9)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 0 | 1 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 0 | 1 | 0 | 0 |

Рис.9

Идем дальше, второй элемент равен единице, значит на место первого индекса дополнительного массива ставим единицу (Рис. 10)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 0 | 1 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 0 | 0 |

Рис.10

Аналогично заполняем весь массив. В конце у нас получится отсортированный по возрастанию массив (Рис.11)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2 | 1 | 0 | 1 | 3 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 2 | 3 |

Рис.11

### Сортировка Хоара (быстрая сортировка)

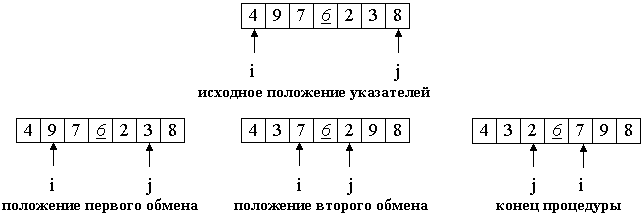
*Принцип работы:*

Отличительной особенностью быстрой сортировки является операция разбиения массива на две части относительно опорного элемента. Например, если последовательность требуется упорядочить по возрастанию, то в левую часть будут помещены все элементы, значения которых меньше значения опорного элемента, а в правую элементы, чьи значения больше или равны опорному. Если длина какой-то из получившихся в результате разбиения частей превышает один элемент, то для нее нужно рекурсивно выполнить упорядочивание, т. е. повторно запустить алгоритм на каждом из отрезков.

*Пример:*

Нам дан массив, состоящий из 7 элементов. В качестве опорного выберем третий элемент (Шаг 1). Начинаем с нулевого элемента. Он меньше опорного, значит идем дальше, первый элемент больше опорного, мы его запоминаем и переходим к правой части массива. Седьмой элемент больше опорного, а вот шестой уже меньше. Мы меняем местами первый элемент и шестой (Шаг 2). Идем дальше по массиву. Второй элемент больше опорного, переходим к правой части. Четвертый элемент меньше опорного, значит меняем местами второй и четвертый (Шаг 4). Слева получились элементы, которые меньше опорного, справа – которые больше (Шаг 5).

Далее повторяем алгоритм уже для отдельных частей.



**Шаг 2 Шаг 3 Шаг 4**

**Шаг 1**

Рис.12

### Сортировка слиянием

*Принцип работы:*

Процедура слияния предполагает объединение двух предварительно упорядоченных подпоследовательностей размерности n/2 в единую последовательность размерности n. Начальные элементы предварительно упорядоченных последовательностей сравниваются между собой, и из них выбирается наименьший. Соответствующий указатель перемещается на следующий элемент. Процедура повторяется до тех пор, пока не достигнут конец одной из подпоследовательностей. Оставшиеся элементы другой подпоследовательности при этом передаются в результирующую последовательность в неизменном виде.

*Пример:*

Нам дан массив, состоящий из 8 элементов, мы делим его на последовательности длины один и начинаем слияние. Нулевой элемент больше первого, поэтому мы меняем их местами и записываем в последовательность длины 2. Второй и третий элементы уже упорядочены, поэтому их записываем, не меняя. С остальными парами проделываеем эту же операцию. Далее первую и вторую пары мы соединяем в упорядоченную последовательность длины 4. Третья и четвертая пары также образуют новую упорядоченную последовательность. На последнем шаге мы получили две упорядоченные последовательности длины 4, которые сортируя объединяем в одну. В итоге у нас получился сортированный по возрастанию массив (Рис.13)

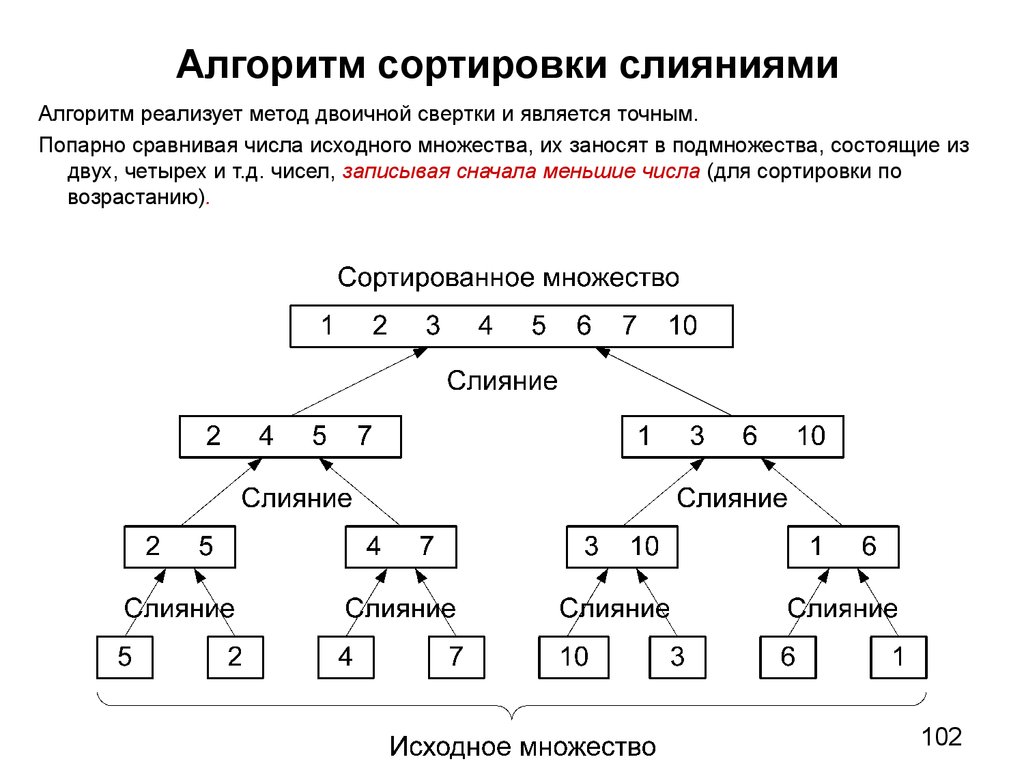


Рис.13

## **3.3 Описание функций**

int ListDirectoryContents(const wchar\_t \*sDir, wchar\_t \*\*filesName, ULONGLONG \*filesSize)

**Назначение:** Получение списка файлов из директория с указанием размеров

**Входные параметры:** wchar\_t \*sDir – массив, содержащий путь к заданному каталогу, wchar\_t \*\*filesName - указатель на массив, содержащий имена файлов, ULONGLONG \*filesSize - целочисленный массив, в котором содержатся размеры файлов в байтах.

void InputFiles(wchar\_t \*\*sDir)

**Назначение:** Ввод пути до каталога

**Входные параметры:** wchar\_t \*\*sDir - указатель на массив, в который будет записан введенный пользователем путь до каталога.

void OutputFiles(ULONGLONG \*fileSize, wchar\_t \*\*fileNames, unsigned long \*fileIdx, int n)

**Назначение:** Вывод списка файлов каталога с указанием их размера в байтах

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, wchar\_t \*\*fileNames – указатель на массив, содержащий имена файлов, unsigned long \*fileIdx – массив индексов файлов каталога, int n – переменная, отвечающая за количество файлов в каталоге.

void Sort()

**Назначение:** Печать доступных сортировок.

void chooseSort(ULONGLONG \*fileSize, unsigned long \*fileIdx, int n)

**Назначение:** осуществление сортировки выбором для массива.

**Входные параметры**: ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, unsigned long \*fileIdx – массив индексов файлов каталога, int n – переменная, отвечающая за количество файлов в каталоге.

void insertSort(ULONGLONG \*fileSize, int \*fileIdx, int n)

**Назначение:** осуществление сортировки простыми вставками для массива.

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int\*fileIdx – массив индексов файлов каталога, int n – переменная, отвечающая за количество файлов в каталоге.

void bubbleSort(ULONGLONG \*fileSize, int \*fileIdx, int n)

**Назначение:** осуществление сортировки пузырьком для массива.

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int \*fileIdx – массив индексов файлов каталога, int n – переменная, отвечающая за количество файлов в каталоге.

void countSort(ULONGLONG \*fileSize, int \*fileIdx, int n)

**Назначение:** осуществление сортировки пузырьком для массива.

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int \*fileIdx – массив индексов файлов каталога, int n – переменная, отвечающая за количество файлов в каталоге.

void quick\_split(ULONGLONG \*fileSize, int fileIdx, int \*i, int \*j, int p)

**Назначение:** осуществление быстрой сортировки для массива (дополнительная функция для функции quick\_sort)

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int\*fileIdx – массив индексов файлов каталога, int \*i, int j – указатели на переменные, отвечающие за индексы крайних элементов массива, int p – переменная, отвечающая за опорный элемент.

void quick\_sort(ULONGLONG \*fileSize, int fileIdx, int n1, int n2)

**Назначение:** осуществление быстрой сортировки для массива

**Входные параметры**: ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int\*fileIdx – массив индексов файлов каталога, int n1, int n2 –переменные, отвечающие за крайние элементов массива.

void merge(ULONGLONG \*fileSize, int \*fileIdx, int l, int m, int r)

**Назначение:** осуществление сортировки слиянием для массива (дополнительная функция для функции merge\_sort)

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int\*fileIdx – массив индексов файлов каталога, int l, int r – переменные, отвечающие за крайние элементы массива, int m – переменная, отвечающая за средний элемент.

void merge\_sort(ULONGLONG \*fileSize, int fileIdx, int l, int r)

**Назначение:** осуществление сортировки слиянием для массива.

**Входные параметры:** ULONGLONG \*fileSize – целочисленный массив, в котором содержатся размеры файлов в байтах, int\*fileIdx – массив индексов файлов каталога, int l, int r – переменные, отвечающие за крайние элементы массива.

void main()

**Назначение:** основная функция программы.

# **Заключение**

В данной практической работе был разработан прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию или убыванию размера с помощью шести различных сортировок:

* Сортировки выбором
* Сортировки вставками
* Пузырьковой сортировки
* Сортировки подсчетом
* Быстрой сортировки
* Сортировки слиянием

В процессе работы с программой пользователь может увидеть время, которое было затрачено на сортировку, а также сменить тип сортировки, если это необходимо.

Ниже приведена таблица сложности алгоритмов разных сортировок (Таблица1)

Таблица 1

|  |  |  |
| --- | --- | --- |
| Метод  Сортировки | Средняя сложность алгоритма | |
| Время выполнения | Память |
| Сортировка выбором | О(n²) | О(1) |
| Сортировка вставками | О(n²) | О(1) |
| Пузырьковая сортировка | О(n²) | О(1) |
| Сортировка подсчетом | О(n²) | max(n) – min(n) + 1 |
| Быстрая сортировка | O() | О(1) |
| Сортировка слиянием | O | О(n) |

Кроме того, я исследовала зависимость количества файлов от времени сортировки. На основе полученных данных я составила таблицу, приведенную ниже (Таблица2).

По вертикали указываются методы сортировки, а по горизонтали количество файлов в каталоге.

Таблица 2

|  |  |  |  |
| --- | --- | --- | --- |
|  | **100** | **1157** | **2115** |
| Сортировка выбором | 0,000 | 0,003 | 0,015 |
| Сортировка вставками | 0,000 | 0,002 | 0,012 |
| Пузырьковая сортировка | 0,000 | 0,007 | 0,022 |
| Сортировка подсчетом | 0,000 | 0,004 | 0,016 |
| Сортировка Хоара | 0,000 | 0,000 | 0,000 |
| Сортировка слиянием | 0,000 | 0,000 | 0,001 |

На основе полученной таблицы можно построить график зависимости времени сортировки от метода

Таким образом, проанализировав все методы сортировки, мы можем сделать вывод, что быстрая сортировка или сортировка Хоара является наиболее эффективной. Она занимает меньше всего времени и памяти при выполнении.