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**

**МЕТОДЫ СОРТИРОВКИ**

Выполнил: Власов Максим Сергеевич, студент группы 381806-1

Проверил: к. т. н., доцент кафедры МОСТ Кустикова В. Д.

Нижний Новгород

2018

**Содержание**

[Введение 3](#_Toc533250665)

[1. Постановка задачи 4](#_Toc533250666)

[2. Руководство пользователя 5](#_Toc533250667)

[3. Руководство программиста 8](#_Toc533250668)

[3.1. Структура программы 8](#_Toc533250669)

[3.2. Описание алгоритма 8](#_Toc533250670)

[3.2.1. Сортировка выбором 8](#_Toc533250671)

[3.2.2. Сортировка простыми вставками 8](#_Toc533250672)

[3.2.3. Пузырьковая сортировка 9](#_Toc533250673)

[3.2.4. Сортировка подсчетом 11](#_Toc533250674)

[3.2.5. Быстрая сортировка 11](#_Toc533250675)

[3.2.6. Сортировка слиянием 12](#_Toc533250676)

[3.3. Описание функций 13](#_Toc533250677)

[Заключение 16](#_Toc533250678)

[Литература 17](#_Toc533250679)

# Введение

Как правило, как людям, так и компьютерам удобнее работать с такими наборами или списками данных, которые упорядочены по определенному свойству. Одним из способов упорядочения множества данных является сортировка.

Работа с файлами – распространенный способ взаимодействия пользователя с операционной системой. Для того чтобы со списком файлов было удобнее работать, его необходимо сначала отсортировать, например, по размеру.

Таким образом, для этой цели и был создан прототип файлового менеджера, с помощью которого можно легко вывести на экран отсортированный одним из 6 способов набор файлов с указанием размеров.

# Постановка задачи

**Задача:** разработать и реализовать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по размеру.

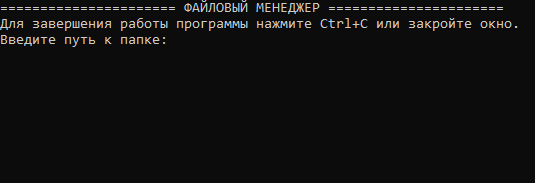
**Входные данные:** путь к каталогу, к которому необходимо обратиться; способ сортировки.

**Выходные данные:** отсортированный по размеру список файлов в заданном каталоге.

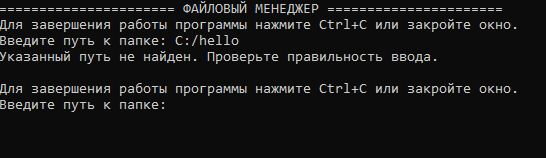
# Руководство пользователя

В данном руководстве содержатся пошаговые инструкции по работе с программой, для того чтобы вы могли как можно быстрее приступить к использованию прототипа файлового менеджера.

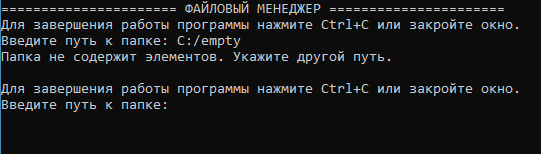
1. Запустите файл **Practice5.exe** из папки с программой. Перед вами отобразится приветственный экран с предложением ввести путь к папке, из которой вы хотите отобразить файлы.



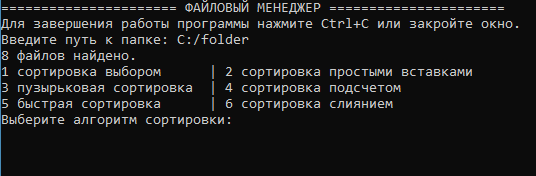
1. Программа после запуска.
2. С помощью клавиатуры введите полный путь к папке и нажмите Enter. Если вы указали неверный путь (попка с таким названием не существует или недоступна), то отобразится сообщение об ошибке (см. Рис 2). Если выбранная папка является пустой, то программа также сообщит вам об этом (см. Рис 3). Проверьте правильность ввода пути и повторите попытку.



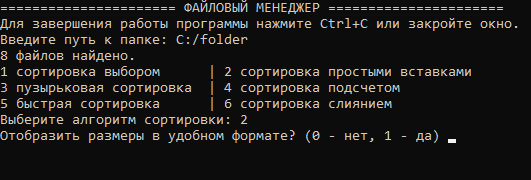
1. Сообщение об ошибке при вводе неверного пути.



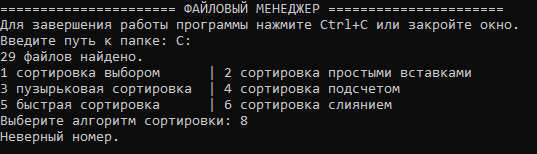
1. Сообщение о том, что папка пустая.



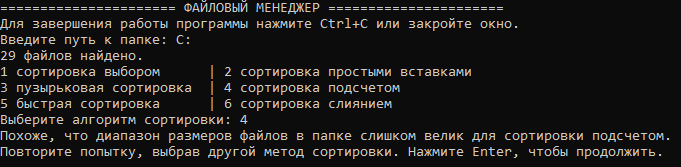
1. Программа после ввода корректного пути к непустой папке.
2. Если папка по указанному пути существует и содержит хотя бы 1 элемент, то вы увидите список доступных методов сортировки (см. Рис 4). Введите цифру от 1 до 6, соответствующую выбранному вами методу, и нажмите Enter. Цифры указаны напротив их названий. Обратите внимание: сортировка подсчетом может быть недоступна для некоторых директорий, размеры файлов в которых находятся в слишком большом диапазоне (см. Рис 7). Если вы ввели некорректный номер, то вам будет предложено повторить попытку (см. Рис 6).



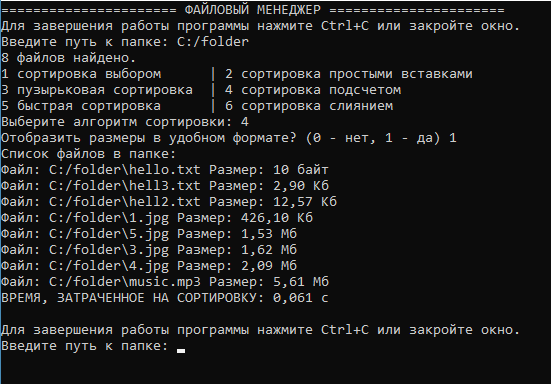
1. Экран программы после ввода номера метода сортировки.
2. Если номер метода сортировки указан верно, то далее вам необходимо будет выбрать, как выводить в размеры – в удобном виде (размеры файлов будут переведены из байт в килобайты, мегабайты или гигабайты, если это нужно) или в байтах (см. Рис 5). Введите 0, если хотите отобразить размеры в байтах, иначе введите 1, после чего нажмите Enter. После этого программа выполнит сортировку.



1. Введен неверный номер метода сортировки.



1. Сообщение об ошибке при попытке отсортировать с помощью сортировки подсчетом файлы с большим диапазоном размеров.
2. По завершении сортировки программа выведет на экран отсортированный по размерам с помощью выбранного вами метода список файлов в указанной папке, а также время, затраченное на сортировку (см. Рис 8). После этого вы снова можете перейти к вводу пути к папке или, если нужно закрыть программу, нажмите сочетание клавиш Ctrl+C либо «крестик» в правом верхнем углу окна.



1. Сортировка успешно завершена.

# Руководство программиста

## Структура программы

Весь код программы находится в файле **main.c**.

## Описание алгоритма

Сортировка файлов возможна с помощью одного из 6 методов, описанных ниже.

### Сортировка выбором

Общий алгоритм сортировки выбором выглядит следующим образом:

С помощью линейного поиска находим минимальное значение в массиве.

Найденный элемент меняем с крайним левым элементом, после чего считаем, что фрагмент массива, состоящий из этого элемента, отсортирован.

Повторяем поиск и перестановки до тех пор, пока весь массив не будет полностью отсортирован.

Пример сортировки выбором рассмотрен в таблице 1 (полужирным выделен отсортированный фрагмент массива, курсивом – индексы).

1. Пример сортировки выбором.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Итерация | Массив | | | | | | | | Индекс минимального элемента | С каким элементом менять |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | 3 | 0 | -5 | 1 | 8 | 4 | 2 | -1 | 2 | 0 |
| 2 | **-5** | 0 | 3 | 1 | 8 | 4 | 2 | -1 | 7 | 1 |
| 3 | **-5** | **-1** | 3 | 1 | 8 | 4 | 2 | 0 | 7 | 2 |
| 4 | **-5** | **-1** | **0** | 1 | 8 | 4 | 2 | 3 | 3 | 3 |
| 5 | **-5** | **-1** | **0** | **1** | 8 | 4 | 2 | 3 | 6 | 4 |
| 6 | **-5** | **-1** | **0** | **1** | **2** | 4 | 8 | 3 | 7 | 5 |
| 7 | **-5** | **-1** | **0** | **1** | **2** | **3** | 8 | 4 | 7 | 6 |
| 8 | **-5** | **-1** | **0** | **1** | **2** | **3** | **4** | 8 | 7 | 7 |
| - | **-5** | **-1** | **0** | **1** | **2** | **3** | **4** | **8** | - | - |

### Сортировка простыми вставками

Общий алгоритм сортировки простыми вставками выглядит следующим образом:

Сначала первый элемент массива считается полностью отсортированным.

Далее из оставшихся элементов берется первый и сравнивается с последним отсортированным. Если отсортированный элемент больше – они меняются местами.

Эту операцию повторяем до тех пор, пока не встретится число меньше или пока элемент не достигнет левого края.

Действия выше повторяем до тех пор, пока массив не будет полностью отсортирован.

Пример сортировки выбором рассмотрен в таблице 2 (полужирным выделен отсортированный фрагмент массива, курсивом – индексы).

1. Пример сортировки простыми вставками.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Итерация | Массив | | | | | | | | Индекс сравниваемого элемента | Нужно ли менять с предыдущим |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | **5** | 9 | 8 | 2 | 6 | 3 | 1 | 4 | 1 | Нет |
| 2 | **5** | **9** | 8 | 2 | 6 | 3 | 1 | 4 | 2 | Да |
| **5** | 8 | **9** | 2 | 6 | 3 | 1 | 4 | 1 | Нет |
| 3 | **5** | **8** | **9** | 2 | 6 | 3 | 1 | 4 | 3 | Да |
| **5** | **8** | 2 | **9** | 6 | 3 | 1 | 4 | 2 | Да |
| **5** | 2 | **8** | **9** | 6 | 3 | 1 | 4 | 1 | Да |
| 2 | **5** | **8** | **9** | 6 | 3 | 1 | 4 | 0 | Нет |
| 4 | **2** | **5** | **8** | **9** | 6 | 3 | 1 | 4 | 4 | Да |
| **2** | **5** | **8** | 6 | **9** | 3 | 1 | 4 | 3 | Да |
| **2** | **5** | 6 | **8** | **9** | 3 | 1 | 4 | 2 | Нет |
| 5 | **2** | **5** | **6** | **8** | **9** | 3 | 1 | 4 | 5 | Да |
| **2** | **5** | **6** | **8** | 3 | **9** | 1 | 4 | 4 | Да |
| **2** | **5** | **6** | 3 | **8** | **9** | 1 | 4 | 3 | да |
| **2** | **5** | 3 | **6** | **8** | **9** | 1 | 4 | 2 | Да |
| **2** | 3 | **5** | **6** | **8** | **9** | 1 | 4 | 1 | Нет |
| 6 | **2** | **3** | **5** | **6** | **8** | **9** | 1 | 4 | 6 | Да |
| **2** | **3** | **5** | **6** | **8** | 1 | **9** | 4 | 5 | Да |
| **2** | **3** | **5** | **6** | 1 | **8** | **9** | 4 | 4 | Да |
| **2** | **3** | **5** | 1 | **6** | **8** | **9** | 4 | 3 | Да |
| **2** | **3** | 1 | **5** | **6** | **8** | **9** | 4 | 2 | Да |
| **2** | 1 | **3** | **5** | **6** | **8** | **9** | 4 | 1 | Да |
| 1 | **2** | **3** | **5** | **6** | **8** | **9** | 4 | 0 | Нет |
| 7 | **1** | **2** | **3** | **5** | **6** | **8** | **9** | 4 | 7 | Да |
| **1** | **2** | **3** | **5** | **6** | **8** | 4 | **9** | 6 | Да |
| **1** | **2** | **3** | **5** | **6** | 4 | **8** | **9** | 5 | Да |
| **1** | **2** | **3** | **5** | 4 | **6** | **8** | **9** | 4 | Да |
| **1** | **2** | **3** | 4 | **5** | **6** | **8** | **9** | 3 | Нет |
| - | **1** | **2** | **3** | **4** | **5** | **6** | **8** | **9** | - | - |

### Пузырьковая сортировка

Общий алгоритм сортировки простыми вставками выглядит следующим образом:

Идем от начала массива, попарно сравнивая элементы. Если первый элемент больше второго, меняем их местами.

При достижении конца массива последний элемент считается полностью отсортированным.

Действия выше повторяем, двигаясь каждый раз от первого элемента до конца неотсортированного фрагмента массива до тех пор, пока он не будет полностью отсортирован.

Пример пузырьковой сортировки рассмотрен в таблице 3 (полужирным выделен отсортированный фрагмент массива, курсивом – индексы).

1. Пример пузырьковой сортировки.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Итерация | Массив | | | | | | | | Индексы сравниваемых элементов | Нужно ли менять элементы местами |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | 8 | 6 | 4 | 2 | 9 | 5 | 3 | 7 | 0, 1 | Да |
| 6 | 8 | 4 | 2 | 9 | 5 | 3 | 7 | 1, 2 | Да |
| 6 | 4 | 8 | 2 | 9 | 5 | 3 | 7 | 2, 3 | Да |
| 6 | 4 | 2 | 8 | 9 | 5 | 3 | 7 | 3, 4 | Нет |
| 6 | 4 | 2 | 8 | 9 | 5 | 3 | 7 | 4, 5 | Да |
| 6 | 4 | 2 | 8 | 5 | 9 | 3 | 7 | 5, 6 | Да |
| 6 | 4 | 2 | 8 | 5 | 3 | 9 | 7 | 6, 7 | Да |
| 2 | 6 | 4 | 2 | 8 | 5 | 3 | 7 | **9** | 0, 1 | Да |
| 4 | 6 | 2 | 8 | 5 | 3 | 7 | **9** | 1, 2 | Да |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | **9** | 2, 3 | Нет |
| 4 | 2 | 6 | 8 | 5 | 3 | 7 | **9** | 3, 4 | Да |
| 4 | 2 | 6 | 5 | 8 | 3 | 7 | **9** | 4, 5 | Да |
| 4 | 2 | 6 | 5 | 3 | 8 | 7 | **9** | 5, 6 | Да |
| 3 | 4 | 2 | 6 | 5 | 3 | 7 | **8** | **9** | 0, 1 | Да |
| 2 | 4 | 6 | 5 | 3 | 7 | **8** | **9** | 1, 2 | Нет |
| 2 | 4 | 6 | 5 | 3 | 7 | **8** | **9** | 2, 3 | Да |
| 2 | 4 | 5 | 6 | 3 | 7 | **8** | **9** | 3, 4 | Да |
| 2 | 4 | 5 | 3 | 6 | 7 | **8** | **9** | 4, 5 | Нет |
| 4 | 2 | 4 | 5 | 3 | 6 | **7** | **8** | **9** | 0, 1 | Нет |
| 2 | 4 | 5 | 3 | 6 | **7** | **8** | **9** | 1, 2 | Нет |
| 2 | 4 | 5 | 3 | 6 | **7** | **8** | **9** | 2, 3 | Да |
| 2 | 4 | 3 | 5 | 6 | **7** | **8** | **9** | 3, 4 | Нет |
| 5 | 2 | 4 | 3 | 5 | **6** | **7** | **8** | **9** | 0, 1 | Нет |
| 2 | 4 | 3 | 5 | **6** | **7** | **8** | **9** | 1, 2 | Да |
| 2 | 3 | 4 | 5 | **6** | **7** | **8** | **9** | 2, 3 | Нет |
| 6 | 2 | 3 | 4 | **5** | **6** | **7** | **8** | **9** | 0, 1 | Нет |
| 2 | 3 | 4 | **5** | **6** | **7** | **8** | **9** | 1, 2 | Нет |
| 7 | 2 | 3 | **4** | **5** | **6** | **7** | **8** | **9** | 0, 1 | Нет |
| - | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | - | - |

### Сортировка подсчетом

Общий алгоритм сортировки подсчетом выглядит следующим образом:

Линейным поиском ищем минимальный (min) и максимальный (max) элемент в сортируемом массиве.

Создаем дополнительный массив, длина которого равна |max – min| + 1, и заполняем нулями.

Далее идем по исходному массиву, каждый раз инкрементируя элемент дополнительного массива, индекс которого равен значению текущего элемента в сортируемом массиве.

После этого заново заполняем массив следующим образом: идем по дополнительному массиву, вставляя в сортируемый массив k элементов со значением n + min, где n – индекс текущего элемента в дополнительном массиве, k – значение этого элемента.

Сортировку подсчетом можно использовать только для индексируемых (нумеруемых) данных. Пример пузырьковой сортировки рассмотрен в таблице 1 (полужирным выделен отсортированный фрагмент массива, курсивом – индексы).

1. Пример сортировки подсчетом.

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *min = 0*  *max = 3* | Сортируемый массив | | | | | | | | Дополнительный массив | | | |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *0* | *1* | *2* | *3* |
| Итерации | 3 | 2 | 3 | 0 | 1 | 1 | 3 | 3 | 0 | 0 | 0 | 0 |
| 1 | ↑ |  |  |  |  |  |  |  | 0 | 0 | 0 | 1 |
|  | ↑ |  |  |  |  |  |  | 0 | 0 | 1 | 1 |
|  |  | ↑ |  |  |  |  |  | 0 | 0 | 1 | 2 |
|  |  |  | ↑ |  |  |  |  | 1 | 0 | 1 | 2 |
|  |  |  |  | ↑ |  |  |  | 1 | 1 | 1 | 2 |
|  |  |  |  |  | ↑ |  |  | 1 | 2 | 1 | 2 |
|  |  |  |  |  |  | ↑ |  | 1 | 2 | 1 | 3 |
|  |  |  |  |  |  |  | ↑ | 1 | 2 | 1 | 4 |
| 2 | 0 | 2 | 3 | 0 | 1 | 1 | 3 | 3 | ↑ |  |  |  |
| **0** | 1 | 1 | 0 | 1 | 1 | 3 | 3 |  | ↑ |  |  |
| **0** | **1** | **1** | 2 | 1 | 1 | 3 | 3 |  |  | ↑ |  |
| **0** | **1** | **1** | **2** | 3 | 3 | 3 | 3 |  |  |  | ↑ |
| - | **0** | **1** | **1** | **2** | **3** | **3** | **3** | **3** | - | | | |

### Быстрая сортировка

Общий алгоритм быстрой сортировки выглядит следующим образом:

Выбираем ведущий элемент (можно взять средний элемент или выбрать случайным образом).

Переставляем значения в массиве таким образом, чтобы элементы, меньшие ведущего, оказались слева, а большие – справа.

Сортируем правую и левую часть таким же методом.

Массив считается отсортированным, когда в каждой части останется по одному элементу.

Пример быстрой сортировки рассмотрен в таблице 5 (полужирным выделен отсортированный фрагмент массива, курсивом – индексы, желтым – ведущие элементы).

1. Пример быстрой сортировки.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | Массив | | | | | | | |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | 3 | 7 | 4 | 6 | 4 | 5 | 8 | 7 |
| 3 | 5 | 4 | 6 | 4 | 7 | 8 | 7 |
| 2 | 3 | 5 | 4 | 4 | **6** | 7 | 8 | 7 |
| 3 | 4 | 4 | **5** | 7 | 7 | **8** |
| 3 | 3 | 4 | 4 | 7 | 7 |
| - | **3** | **4** | **4** | **5** | **6** | **7** | **7** | **8** |

### Сортировка слиянием

Общий алгоритм сортировки слиянием выглядит следующим образом:

Разбиваем массив на две части.

Продолжаем разбивать массив на две части, пока в каждой из них не останется по 1 элементу – массив из одного элемента считается отсортированным.

Выполняем последовательное слияние отсортированных частей массива.

Пример сортировки слиянием рассмотрен в таблице 6 (полужирным выделен отсортированный фрагмент массива, курсивом – индексы, синим и красным – части массива, слияние которых выполняется).

1. Пример сортировки слиянием.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Шаг | Массив | | | | | | | |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 1 | 2 | 8 | 4 | 1 | 0 | 2 | 8 | 3 |
| 2 | 8 | 4 | 1 | 0 | 2 | 8 | 3 |
| 2 | 8 | 1 | 4 | 0 | 2 | 8 | 3 |
| 2 | 8 | 1 | 4 | 0 | 2 | 8 | 3 |
| 2 | 2 | 8 | 1 | 4 | 0 | 2 | 3 | 8 |
| 1 | 2 | 4 | 8 | 0 | 2 | 3 | 8 |
| 3 | 1 | 2 | 4 | 8 | 0 | 2 | 3 | 8 |
| - | **0** | **1** | **2** | **2** | **3** | **4** | **8** | **8** |

## Описание функций

void print\_menu()

**Назначение:** печать меню.

**Входные параметры:** отсутствуют.

void user\_input(wchar\_t \*\*wa)

**Назначение:** ввод пути к заданному каталогу.

**Входные параметры:** wa – указатель на массив, в который будет записано введенное пользователем значение.

int dir\_contents(const wchar\_t \*sDir, wchar\_t \*\*\*fileNames, ULONGLONG \*\*fileSizes)

**Назначение:** подсчет количества элементов в указанной папке и получение их названий и размеров.

**Входные параметры:** sDir – строка, содержащая путь к папке, fileNames, fileSizes – массивы для записи полных имен и размеров файлов.

**Выходные данные:** количество элементов в папке или -1, если папка недоступна.

void output\_by\_idxes(int \*filesIdxes, wchar\_t \*\*fileNames, ULONGLONG \*fileSizes, int filesCount, int is\_format)

**Назначение:** вывод списка файлов с размерами с использованием упорядочивающего массива.

**Входные параметры:** filesIdxes – упорядочивающий массив, содержащий последовательность индексов отсортированных элементов связанных массивов имен и размеров, fileNames – указатель на массив имен, fileSizes – указатель на массив размеров, filesCount – количество файлов, is\_format – флаг, указывающий, нужно ли выводить размеры файлов в удобочитаемом виде (см. функцию [output\_formatted\_size](#output_formatted_size)).

void output\_formatted\_size(ULONGLONG size)

**Назначение:** печать размера файла в удобочитаемом виде (перевод байтов в килобайты, мегабайты и гигабайты, если это необходимо).

**Входные параметры:** size – размер файла в байтах.

int find\_by\_size(ULONGLONG size, int k, ULONGLONG \*sizes, int n)

**Назначение:** поиск k-го файла с заданным размером в массиве с размерами.

**Входные параметры:** size – искомый размер файла (в байтах), k – порядковый номер элемента с таким размером, sizes – массив размеров файлов, n – размер массива.

**Выходные данные:** индекс найденного значения, либо -1 (если такое значение не найдено) или

-2 (если k превышает размер массива n).

void swap\_int(int \*var1, int \*var2)

**Назначение:** обмен значений переменных типа int.

**Входные параметры:** var1, var2 – указатели на переменные, значения которых нужно поменять местами.

void swap\_ULL(ULONGLONG \*var1, ULONGLONG \*var2)

**Назначение:** обмен значений переменных типа ULONGLONG.

**Входные параметры:** var1, var2 – указатели на переменные, значения которых нужно поменять местами.

void choosing\_sort(ULONGLONG \*sizes, int \*idxes, int n)

**Назначение:** выполнение сортировки выбором для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, n – количество элементов в массиве.

void insert\_sort(ULONGLONG \*sizes, int \*idxes, int n)

**Назначение:** выполнение сортировки простыми вставками для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, n – количество элементов в массиве.

void bubble\_sort(ULONGLONG \*sizes, int \*idxes, int n)

**Назначение:** выполнение пузырьковой сортировки для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, n – количество элементов в массиве.

int counting\_sort(ULONGLONG \*sizes, int \*idxes, int n)

**Назначение:** выполнение сортировки подсчетом для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, n – количество элементов в массиве.

**Выходные данные:** результат выполнения (1 – ошибка, 0 – успех).

void start\_quick\_sort(ULONGLONG \*sizes, int \*idxes, int n1, int n2)

**Назначение:** подготовка и запуск быстрой сортировки для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, n1, n2 – индексы крайних элементов сортируемого фрагмента массива.

void quick\_sort(ULONGLONG \*sizes, int \*idxes, int n1, int n2)

**Назначение:** выполнение быстрой сортировки для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, n1, n2 – индексы крайних элементов сортируемого фрагмента массива.

void start\_merge\_sort(ULONGLONG \*sizes, int \*idxes, int l, int r)

**Назначение:** подготовка и запуск сортировки слиянием для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, l, r – индексы крайних элементов сортируемого фрагмента массива.

void merge\_sort(ULONGLONG \*sizes, int \*idxes, int l, int r)

**Назначение:** выполнение сортировки слиянием для указанного массива.

**Входные параметры:** sizes – массив размеров файлов, которые нужно упорядочить, idxes – массив индексов файлов, размеры которых необходимо отсортировать, l, r – индексы крайних элементов сортируемого фрагмента массива.

void merge\_sorted(ULONGLONG \*sizes, int \*idxes, int l, int m, int r)

**Назначение:** выполнение слияния отсортированных частей массива.

**Входные параметры:** sizes – массив, который части которого необходимо слить, idxes – массив индексов файлов, l, m, r – индексы правого, среднего и левого элемента фрагмента массива, части которого необходимо рекурсивно слить.

void main()

**Назначение:** основная функция (точка входа).

**Входные параметры:** отсутствуют.

# Заключение

В ходе выполнения практической работы «Методы сортировки» был разработан и реализован прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по размеру. Для сортировки используется 6 методов: сортировка выбором, сортировка простыми вставками, пузырьковая сортировка, сортировка подсчетом, быстрая сортировка, сортировка слиянием. В таблице ниже приведена сравнительная характеристика сложности работы данных алгоритмов.

1. Сложность методов сортировки

|  |  |  |
| --- | --- | --- |
| Методы сортировки | Средняя сложность алгоритма | |
| Время выполнения | Память |
| Сортировка выбором | O(n2) | O(1) |
| Сортировка простыми вставками | O(n2) | O(1) |
| Пузырьковая сортировка | O(n2) | O(1) |
| Сортировка подсчетом | O(n) | max(n) – min(n) + 1 |
| Быстрая сортировка | O(n·log2 n) | O(1) |
| Сортировка слиянием | O(n·log2 n) | O(n) |

На графиках ниже показана зависимость времени сортировки от количества элементов для разных оценок. По горизонтальной оси располагается количество элементов.

1. Тестирование сортировок

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Метод сортировки |  | Время выполнения сортировки (в секундах) | | | | |
| Кол-во файлов | 100 | 1 000 | 5 000 | 10 000 | 50 000 |
| Сортировка выбором | | 0.000 | 0.003 | 0.048 | 0.143 | 4.724 |
| Сортировка простыми вставками | | 0.000 | 0.003 | 0.061 | 0.201 | 6.043 |
| Пузырьковая сортировка | | 0.000 | 0.017 | 0.318 | 1.487 | 40.006 |
| Сортировка подсчетом | | 0.000 | 0.003 | 0.073 | 0.145 | 3.592 |
| Быстрая сортировка | | 0.000 | 0.001 | 0.005 | 0.005 | 0.023 |
| Сортировка слиянием | | 0.000 | 0.003 | 0.014 | 0.022 | 0.090 |