МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

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 с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию размера**

**Выполнил:** студент группы 381806-1

Белик Юлия Андреевна

**Проверил:**кандидат технических наук доцент кафедры МОСТ ИИТММ

Кустикова Валентина Дмитриевна

Нижний Новгород

2018

Содержание

1. ……………………………………………………….3
2. …………………………….………………4
3. ……………………………………5
4. …………………………………...7
5. ………………………………………………7
6. ………………………………………………...7
7. ……………………………………………………7
8. …………………………………………………..7
9. ………………………………………………..7
10. ………………………………………………….8
11. ……………………………..………………………8
12. …………………………………………………...8
13. ………………………………………………….9
    * 1. ……………………………………………..9
      2. ……………………………………………………...9
      3. ……………………………………………………….9
      4. ……………………………………………………..9
      5. ……………………………………………………9
      6. ………………………………………………..10
      7. ………………………………………………….10
      8. ……………………………………10
      9. ……………………………………………………..10
      10. ………………………………………………………..11
      11. …………………………………………………...11
14. ...............................................................................12
15. Введение

В настоящее время наука и технологии шагают вперед семимильными шагами. Чтобы идти в ногу с прогрессом, приходится постоянносовершенствовать свои знания, умения и навыки, а на это требуется достаточно много времени. В целях освобождения времени для таких фундаментальных задач нужно минимализировать его затраты на небольшие задачи, которые можно решить один раз вместо того, чтобы каждый раз делать все заново.

Именно с этой целью пишутся программы, совершающие элементарные операции по обработке данных. Примером служит данная работа, в которой реализованы различные методы сортировок данных.

1. Постановка задачи

Разработать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию размера. Для этого необходимо реализовать следующие методы сортировки:

* Сортировка выбором
* Сортировка вставками
* Пузырьковая сортировка
* Сортировка подсчетом
* Сортировка слиянием
* Быстрая сортировка

Входные данные:

* Путь до директории, в которой необходимо отсортировать содержимое
* Метод сортировки

Выходные данные:

* Отсортированный список имен файлов с указанием размера
* Время сортировки

1. Руководство пользователя

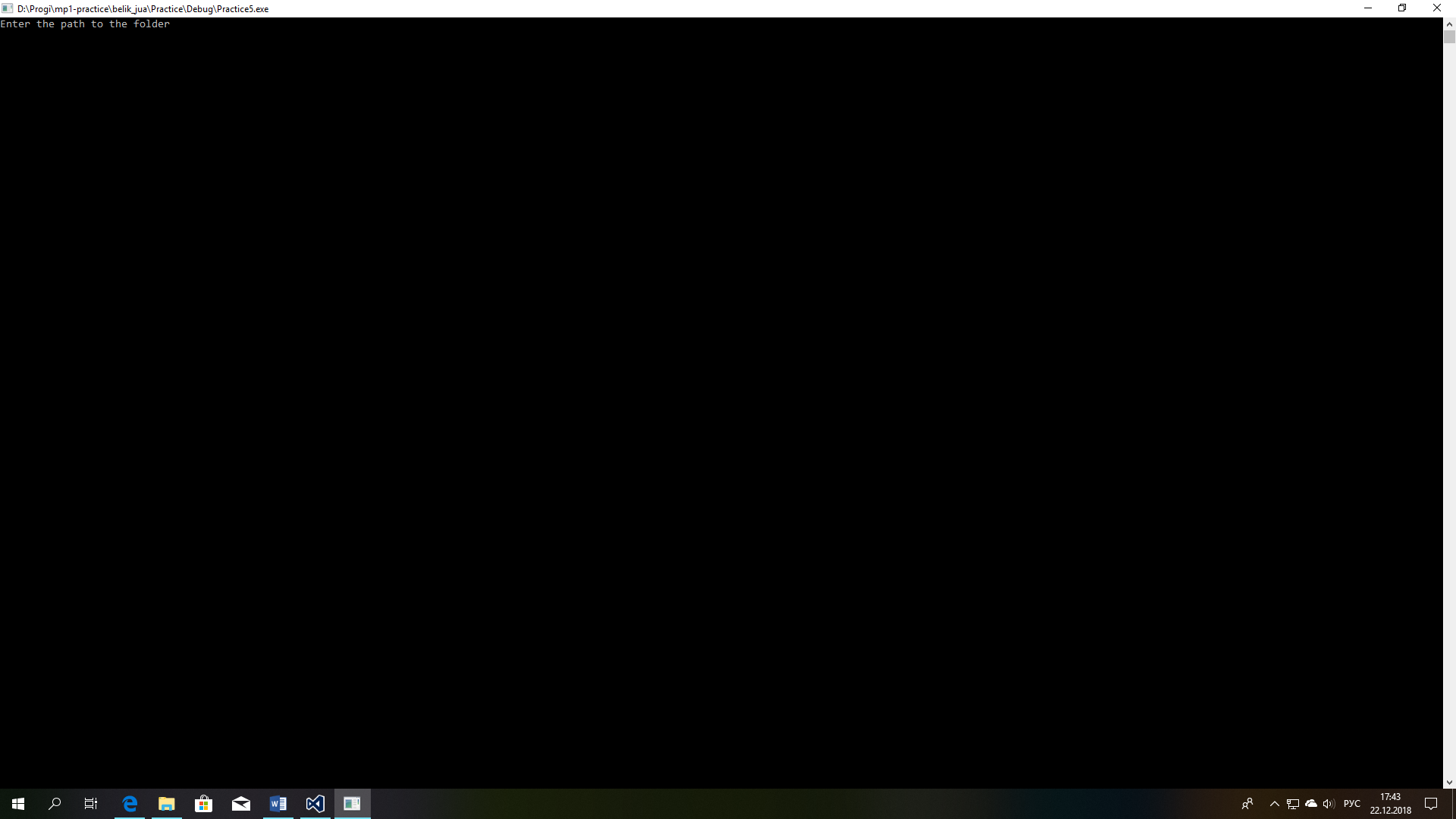
После запуска программа на экран выводится запрос на введение пути до директории, в которой необходимо отсортировать содержимое[1]

Рисунок 1. Запрос на ввод пути до директории

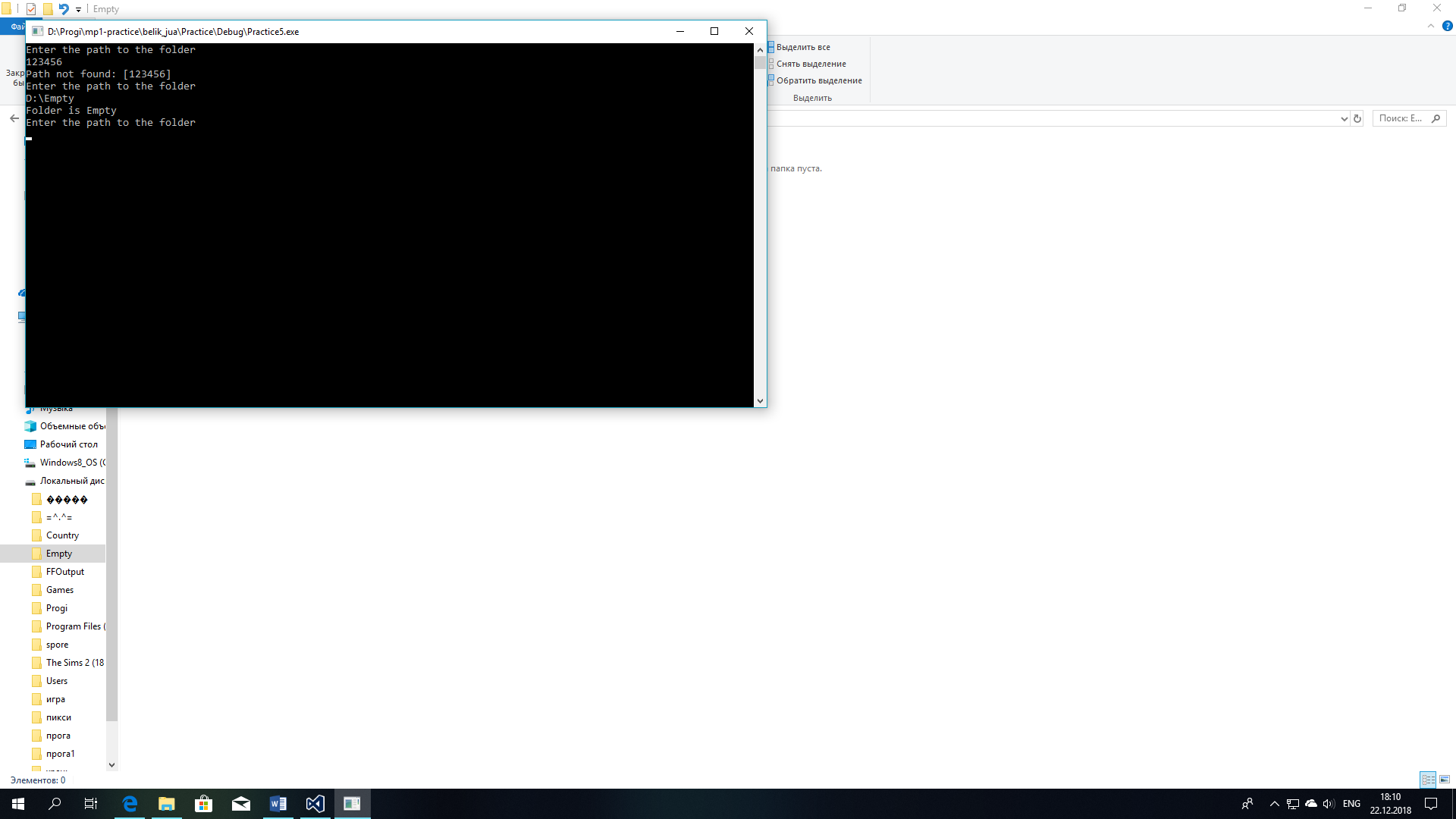
В случае некорректного ввода пути или указания пути в пустую директорию запрос на ввод пути повторится [].

Рисунок 2. Пример неверного пути до директории и пути в пустую папку

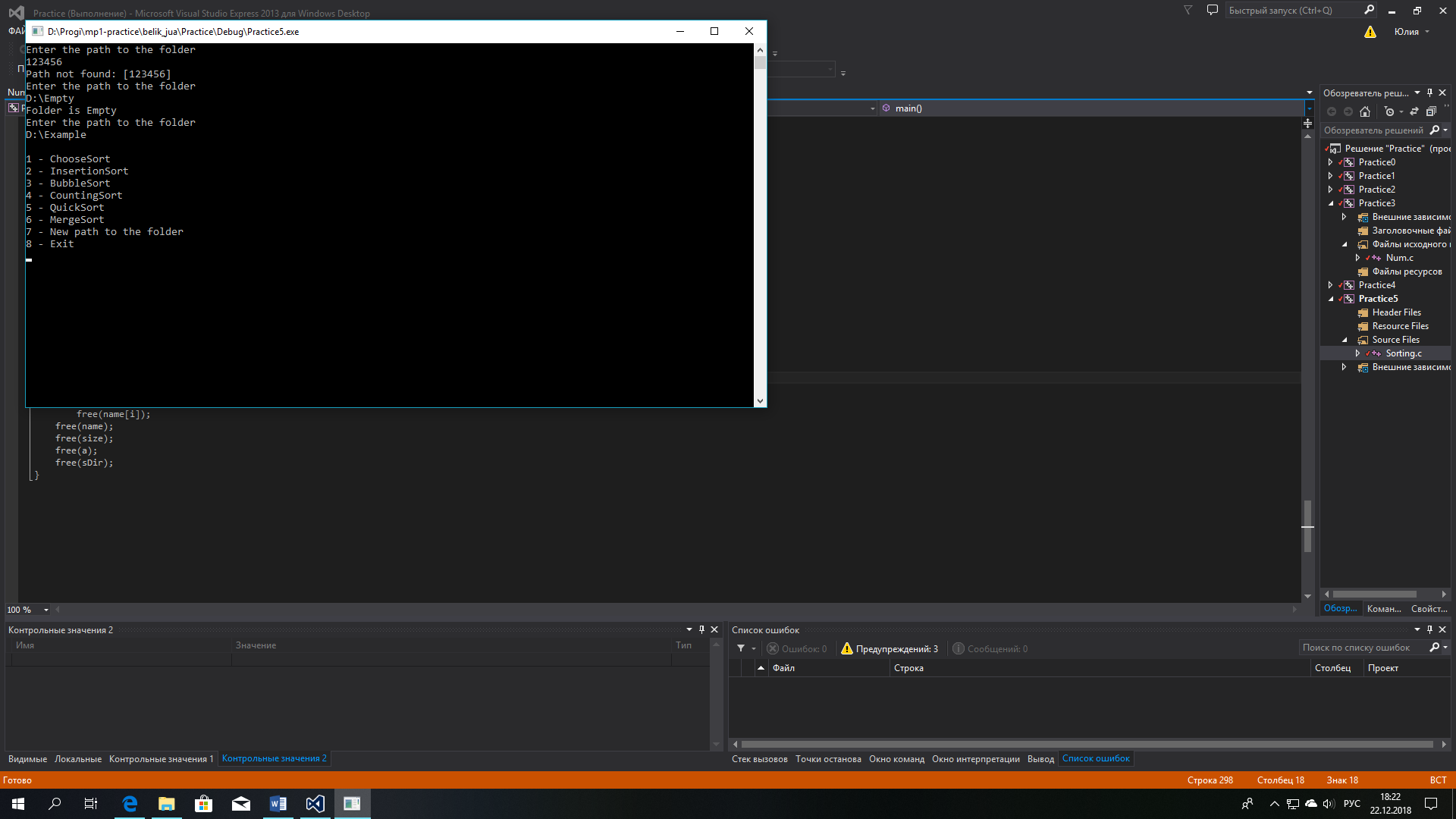
В случае корректно введенных данных на экране появится меню команд. Для выбора одной из команд необходимо ввести ее номер [].

Рисунок 3. Меню команд

В случае введения номера команды от 1 до 6 будет выполнена выбранная сортировка и на экран будут выведены отсортированные названия файлов и их размеры в килобайтах, время сортировки и меню команд [4].

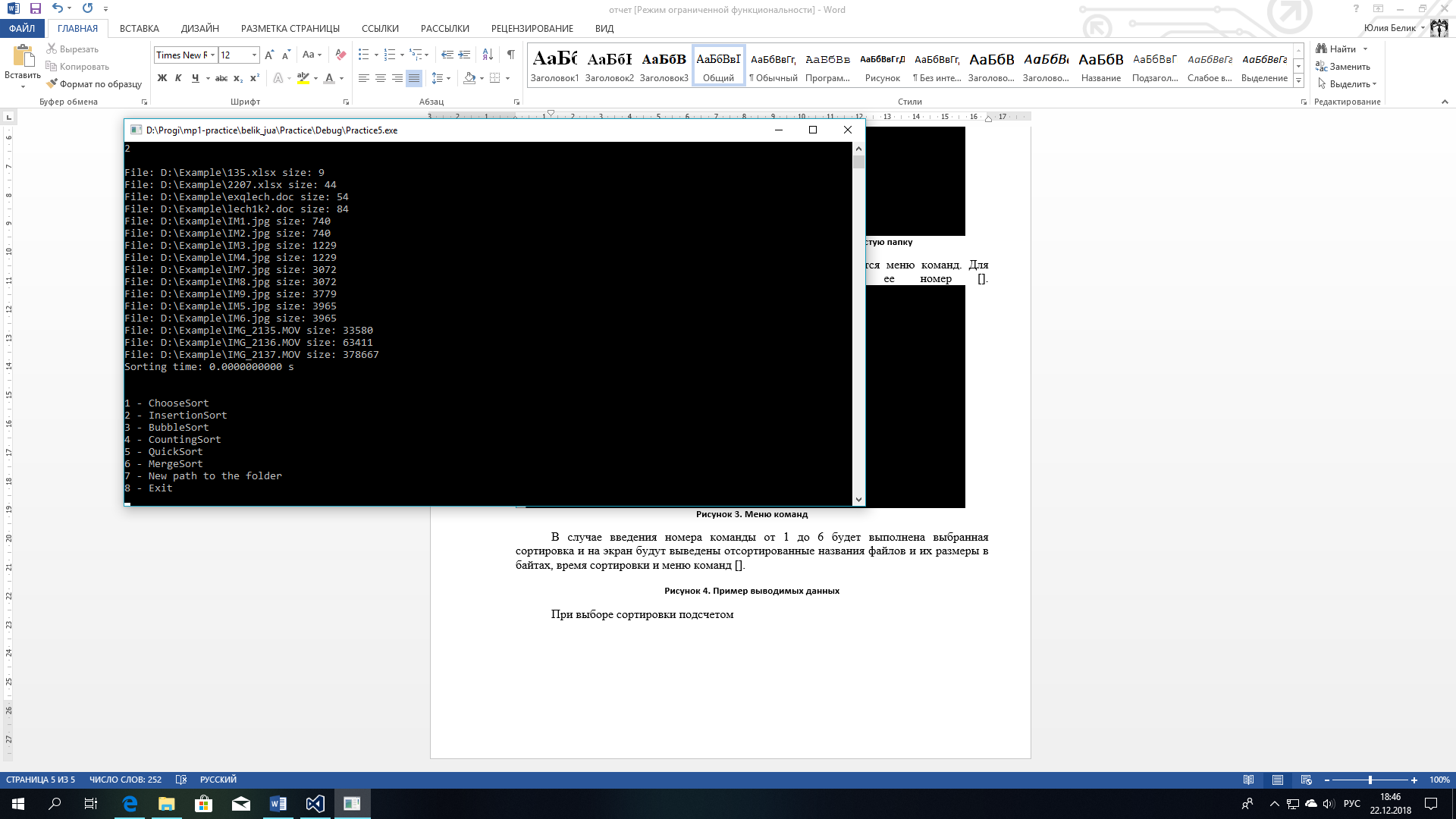


Рисунок 4. Пример выводимых данных

При введении 7 на экран выведется запрос на введение нового пути до директории[1].

Для завершения работы с программой необходимо выбрать 8 команду меню команд – Exit[3].

1. Руководство программиста
   1. Структура программы

Программа состоит из 1 файла *Sorting.c,* который содержит функцию *main*, функции сортировок, функцию вывода меню команд, функцию вывода отсортированных названий файлов, их размеров в килобайтах и времени сортировки.

* 1. Описание алгоритма
     1. Сортировка выбором – ChooseSort
* Находим позицию минимального элемента в массиве
* Меняем минимальный элемент с первым
* Продолжаем для массива, начинающегося со следующего элемента[1]

Таблица 1. Пример сортировки выбором

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 6 | 3 | 4 | 5 | 1 | 2 |
| 1 | 3 | 4 | 5 | 6 | 2 |
| 1 | 2 | 4 | 5 | 6 | 3 |
| 1 | 2 | 3 | 5 | 6 | 4 |
| 1 | 2 | 3 | 4 | 6 | 5 |
| 1 | 2 | 3 | 4 | 5 | 6 |

* + 1. Сортировка вставками – InsertionSort
* Начинаем с массива из одного (первого) элемента, считая, что он отсортирован
* Каждый следующий проход увеличиваем длину рассматриваемого массива на 1
* Проходим массив от последнего элемента к первому
* Пока (j-1)-й элемент больше j-того, меняем их местами [2]

Таблица 2. Пример сортировки вставками

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 12 | 9 | 3 | 7 | 14 | 11 |
| 9 | 12 | 3 | 7 | 14 | 11 |
| 3 | 9 | 12 | 7 | 14 | 11 |
| 3 | 7 | 9 | 12 | 14 | 11 |
| 3 | 7 | 9 | 12 | 14 | 11 |
| 3 | 7 | 9 | 11 | 12 | 14 |

* + 1. Пузырьковая сортировка – BubbleSort
* Проходим массив с первого элемента до n-ого (где n – количество элементов в массиве)
* Сравниваем попарно стоящие рядом элементы (1 и 2, 2 и 3, ..., n–1 и n)
* Если первый элемент пары больше второго, меняем элементы местами
* Для каждого следующего прохода количество элементов уменьшается на 1[3]

Таблица 3. Пример пузырьковой сортировки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 5 | 1 | 2 | 3 | 6 |
| 1 | 5 | 2 | 3 | 6 |
| 1 | 2 | 5 | 3 | 6 |
| 1 | 2 | 3 | 5 | 6 |
| 1 | 2 | 3 | 5 | 6 |
| 1 | 2 | 3 | 5 | 6 |

* + 1. Сортировка подсчетом – CountingSort
* Создаем дополнительный массив и заполняем его 0
* Проходим по всему исходному массиву и прибавляем 1 к элементу нового массива, индекс которого равен значению элемента исходного массива
* Заново заполняем исходный массив, при этом записываем элемент равный индексу нового массива столько раз, сколько равен элемент нового массива с этим индексом [4]

Таблица 4. Пример сортировки подсчетом

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

* + 1. Быстрая сортировка – QuickSort
* Выбираем опорный элемент
* Все меньшие элементы перемещаем влево от него, большие – вправо
* Рекурсивно продолжаем для левой и правой половин, пока размер подмассива не станет меньше [5]

Таблица 5. Пример быстрой сортировки

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 3 | 6 | 9 | 1 | 2 |
| 3 | 6 | 2 | 1 | 9 |
| 1 | 6 | 2 | 3 | 9 |
| 1 | 2 | 6 | 3 | 9 |
| 1 | 2 | 6 | 3 | 9 |
| 1 | 2 | 3 | 6 | 9 |

* + 1. Сортировка слиянием – MergeSort
* Рекурсивно разбиваем исходный массив на подмассивы из одного элемента
* Рекурсивно попарно сливаем подмассивы
* На каждом этапе слияния упорядочиваем подмассивы [6]

Таблица 6. Пример сортировки слиянием

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 7 | 2 | 1 | 3 | 9 | 5 | 0 | 6 |
| 7 | 2 | 1 | 3 | 9 | 5 | 0 | 6 |
| 7 | 2 | 1 | 3 | 9 | 5 | 0 | 6 |
| 7 | 2 | 1 | 3 | 9 | 5 | 0 | 6 |
| 2 | 7 | 1 | 3 | 5 | 9 | 0 | 6 |
| 1 | 2 | 3 | 7 | 0 | 5 | 6 | 9 |
| 0 | 1 | 2 | 3 | 5 | 6 | 7 | 9 |

* 1. Описание функций
     1. Подключение к директории с последующим формированием массивов названий файлов и их размеров

intListDirectoryContents(constwchar\_t \*sDir, ULONGLONG \*size, wchar\_t \*\*\*name)

* Назначение

Определение корректности введенных данных, определение количества файлов в директории, формирование и заполнение массивов названий файлов и их размеров

* Входные параметры

\*sDir – путь до каталога

\*size – массив размеров файлов

\*\*\*name – адрес массива названий

* Выходные параметры

N – количество файлов в каталоге

* + 1. Вывод меню команд

voidCommand()

* Назначение

Вывод на экран меню команд

* Входные параметры

Отсутствуют

* Выходные параметры

Отсутствуют

* + 1. Вывод отсортированного списка названий файлов с указанием размера и времени сортировки

void Print(int \*a, ULONGLONG \*size, wchar\_t \*\*name, int n, float time)

* Назначение

Вывод на экран отсортированного списка имен файлов с указанием размера и времени сортировки

* Входные параметры

\*a – массив индексов отсортированного массива размеров файлов

\*size – массив размеров файлов

\*\*name – массив названий файлов

time – время работы сортировки

* Выходные параметры

Отсутствуют

* + 1. Сортировка выбором

voidChooseSort(int \*a, ULONGLONG \*size, int n)

* Назначение

Сортировка массива размеров файлов сортировкой выбором с изменением массива индексов[1]

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

n – количество файлов в каталоге

* Выходные параметры

Отсутствуют

* + 1. Сортировка вставками

voidInsertionSort(int \*a, ULONGLONG \*size, int n)

* Назначение

Сортировка массива размеров файлов сортировкой вставками с изменением массива индексов[2]

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

n – количество файлов в каталоге

* Выходные параметры

Отсутствуют

* + 1. Пузырьковая сортировка

voidBubbleSort(int \*a, ULONGLONG \*size, int n)

* Назначение

Сортировка массива размеров файлов пузырьковой сортировкой с изменением массива индексов[3]

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

n – количество файлов в каталоге

* Выходные параметры

Отсутствуют

* + 1. Сортировка подсчетом

intCountingSort(int \*a, ULONGLONG \*size, int n)

* Назначение

Сортировка массива размеров файлов сортировкой подсчетом с изменением массива индексов[4]

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

n – количество файлов в каталоге

* Выходные параметры

Отсутствуют

* + 1. Разбиение для быстрой сортировки

voidQuicksplit(int \*a, ULONGLONG \*size, int \*i, int \*j, ULONGLONG p)

* Назначение

Функция необходимая для быстрой сортировки. Все элементы, меньшие опорного,перемещаются влево от него, большие – вправо.

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

\*i – адрес первого индекса

\*j – адрес второго индекса

p – опорный элемент

* Выходные параметры

Отсутствуют

* + 1. Быстрая сортировка

voidQuickSort(int \*a, ULONGLONG \*size, int n1, int n2)

* Назначение

Сортировка массива размеров файлов быстрой сортировкой с изменением массива индексов[5]

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

n1 – индекс первого элемента

n2 – индекс последнего элемента

* Выходные параметры

Отсутствуют

* + 1. Функция слияния

void Merge(int \*a, ULONGLONG \*size, int l, int m, int r)

* Назначение

Функция необходимая для сортировки слиянием. Соединяет подмассивы, упорядочивая их.

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

l – индекс первого элемента

m – индекс среднего

r – индекс последнего элемента

* Выходные параметры

Отсутствуют

* + 1. Сортировка слиянием

voidMergeSort(int \*a, ULONGLONG \*size, int l, int r)

* Назначение

Сортировка массива размеров файлов сортировкой слиянием с изменением массива индексов[6]

* Входные параметры

\*a – массив индексов массива размеров файлов

\*size – массив размеров файлов

l –индекс первого элемента

r –индекс последнего элемента

* Выходные параметры

Отсутствуют

Заключение

Разработан прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию размера. Реализованы следующие методы сортировок:

• Сортировка выбором

• Сортировка вставками

• Пузырьковая сортировка

• Сортировка подсчетом

• Сортировка слиянием

• Быстрая сортировка

Анализ методов сортировок

Для объективной оценки эффективности работы алгоритмов необходимо провести вычислительный эксперимент – определить, за какое время алгоритмы выполняют сортировку для различных значений размера каталога.

Эксперимент проводился следующим образом: перед каждым измерением создавалась папка заданного размера (для контроля были выбраны точки 100, 500, 1000, 2500, 5000, 7500 и 10000), а затем производилась сортировка. Время работы каждого алгоритма измерялось трижды, меньшее время вносилось в таблицу учета результатов эксперимента, представленную ниже [7].

Таблица 7. Зависимость времени работы сортировок от размера каталога

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | **100** | **500** | **1000** | **2500** | **5000** | **7500** | **10000** |
| **Choose** | 0 | 0.001 | 0.005 | 0.033 | 0.143 | 0.303 | 0.545 |
| **Insertion** | 0 | 0.001 | 0.003 | 0.024 | 0.088 | 0.220 | 0.377 |
| **Bubble** | 0 | 0.002 | 0.012 | 0.044 | 0.180 | 0.403 | 0.732 |
| **Counting** | 0 | 0.002 | 0.002 | 0.010 | 0.022 | 0.033 | 0.043 |
| **Quick** | 0 | 0 | 0 | 0.002 | 0.002 | 0.004 | 0.005 |
| **Merge** | 0 | 0.001 | 0.002 | 0.010 | 0.037 | 0.077 | 0.134 |

Таблица 1 позволяет определить, насколько эффективна каждая сортировка для определенных значений размеров каталога. Очевидно, что самой неэффективной является сортировка пузырьком, а наиболее эффективными – быстрая сортировка.

Для большей наглядности результаты измерения времени работы алгоритмов сортировки представлены ниже в виде графика [5].

Рисунок . График зависимости времени работы алгоритмов от размера каталога

Итак, график позволяет расположить алгоритмы сортировки в порядке уменьшения эффективности следующим образом:

1. Быстрая сортировка
2. Сортировка подсчетом
3. Сортировка слиянием
4. Сортировка вставками
5. Сортировка выбором
6. Пузырьковая сортировка