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: Гогов Владислав Игоревич, студент группы 381806-1

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

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc533274910)

[1. Постановка задачи 4](#_Toc533274911)

[2. Руководство пользователя 5](#_Toc533274912)

[3. Руководство программиста 8](#_Toc533274913)

[3.1. Структура программы 8](#_Toc533274914)

[3.2. Описание алгоритма 8](#_Toc533274915)

[3.2.1. Сортировка выбором 8](#_Toc533274916)

[3.2.2. Сортировка простыми вставками 8](#_Toc533274917)

[3.2.3. Пузырьковая сортировка 9](#_Toc533274918)

[3.2.4. Сортировка подсчетом 9](#_Toc533274919)

[3.2.5. Быстрая сортировка 10](#_Toc533274920)

[3.2.6. Сортировка слиянием 11](#_Toc533274921)

[3.3. Описание функций 12](#_Toc533274922)

[Заключение 15](#_Toc533274923)

# Введение

В наше время люди часто встречаются со списками или наборами данных, записаны в случайном порядке. Удобнее всего работать со списками данных, которые упорядочены по одному свойству.

Работая на компьютере, мы встречаем с файлами. Для более удобной работы с ними, лучше всего отсортировать данный список по одному свойству, например, по размеру.

Для этого была создана программа «Файловый менеджер», которая нужна для просмотра списка файлов в папке и их размеров, в выбранном каталоге пользователем. Это программа сортирует данный список одним из шестью способами (на выбор пользователя). Выводит отсортированный список с указанием размеров и время сортировки данных.

# Постановка задачи

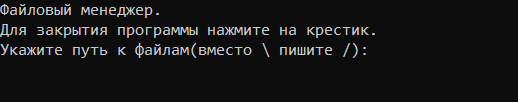
Задача: Разработать «Файловый менеджер», с возможностью сортировки файлов по возрастанию и замером времени сортировки данных.

Входные данные: Путь к файлам.

Выходные данные: Отсортированный список файлов и время, за которое сортируются данные.

# Руководство пользователя

1. Запустить файл **Practice5.exe.** Перед вами выводится экран с предложением ввести путь к папке. (см. Рис 1)

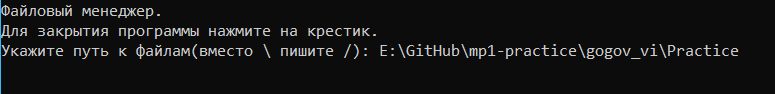


1. Запуск программы.
2. С помощью клавиатуры введите полный путь к файлам (указать директорию) и нажмите Enter.

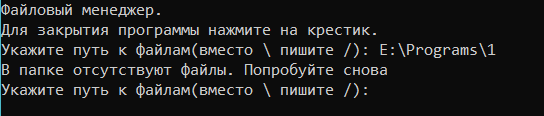
Например, E:\GitHub\mp1-practice\gogov\_vi\Practice. (см. Рис 2)

Если папка пустая, то выводится сообщение об этом. (см. Рис 3)

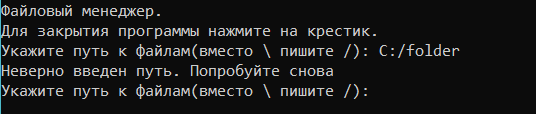
Если в пути присутствуют буквы кириллицы или будет указан неверный путь, то выводиться ошибка «Неверно введен путь. Попробуйте снова». (см .Рис 4)



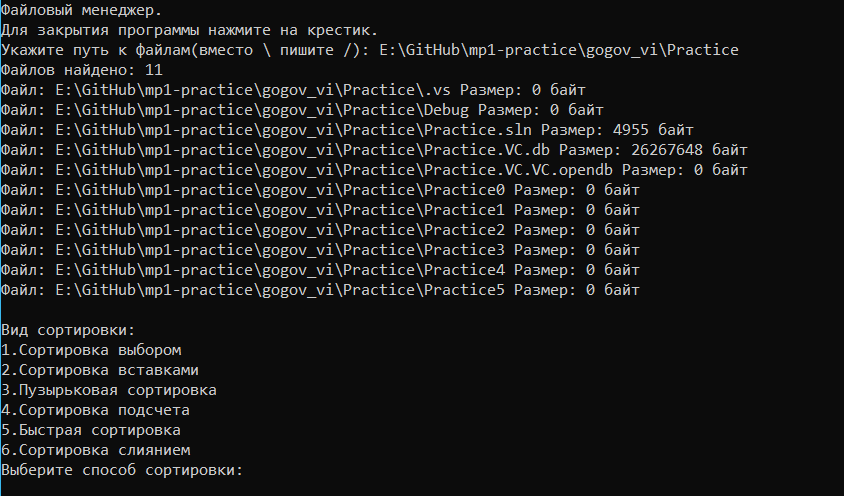
1. Ввод пути.



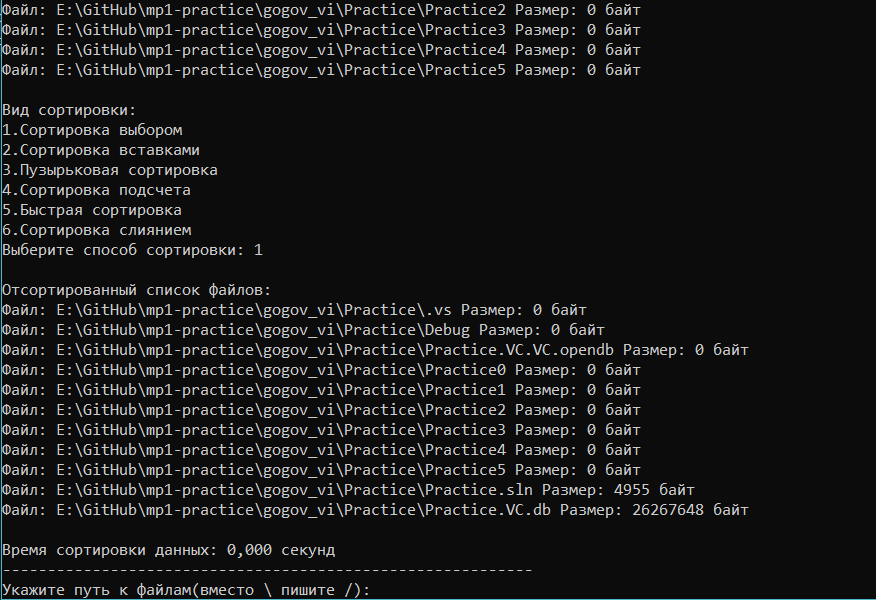
1. Сообщение о том, что папка пустая



1. Сообщение при вводе неверного пути.
2. После вывода списка файлов и их размеров на экране появится меню сортировок, вам необходимо ввести цифру от 1 до 6, чтобы выбрать способ сортировки или закрыть программу нажав на «крестик» (в правом верхнем углу окошка). При вводе неверного числа происходит вызов меню еще раз для выбора. (см. Рис 5)



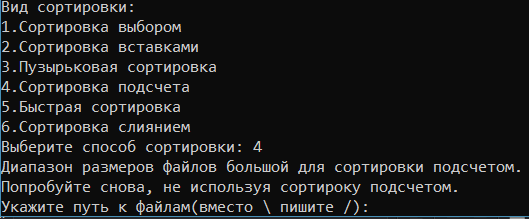
1. Вывод списка файлов в папке.
2. После выводится отсортированный список файлов и их размеров, а также время сортировки данных. Потом вы можете снова воспользоваться программой повторяя пункты 2 – 3 либо закрыть программу на «крестик» (в правом верхнем углу окошка). (см .Рис 6)



1. Вывод отсортированного списка файлов.

**Примечания**: Для сортировки номер 4 (Сортировки подсчетом), если размер файлов находятся в слишком большом диапазоне, то выводиться надпись об этом. (см. Рис 7)

После нужно повторить пункты 2 – 3 и выбрать другой тип сортировки, не используя сортировку под номером 4 (Сортировки подсчетом).



1. Ошибка при при выборе сортировки подсчетом для файлов, находящихся в большом диапазоне.

# Руководство программиста

## Структура программы

Practice5.c – исходный код программы.

## Описание алгоритма

Список файлов сортируется с помощью следующих методов.

### Сортировка выбором

Описание: алгоритм по сортировке массива. Сложность сортировки Ɵ(N2). Память: Ɵ(1).

Алгоритм:

Находим номер минимального элемента в текущем списке.

Производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент находится на данной позиции)

Сортируем остаток списка, исключив уже отсортированные элементы.

Пример: Таблица 1. Исходный массив: 6 1 7 2 9

1. Пример сортировки выбором.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № Шага | Массив | | | | | Описание |
| 1. | 6 | 1 | 7 | 2 | 9 | Меняем минимальный элемент с 1 позицией. |
| 2. | 1 | 6 | 7 | 2 | 9 | Меняем минимальный элемент со 2 позицией. |
| 3. | 1 | 2 | 7 | 6 | 9 | Меняем минимальный элемент с 3 позицией. |
| 4. | 1 | 2 | 6 | 7 | 9 | Обмена не требуется. |

Отсортированный массив: 1 2 6 7 9

### Сортировка простыми вставками

Описание: алгоритм по сортировке массива. Сложность сортировки: Ɵ(N2). Память: Ɵ(1).

Алгоритм:

Первый элемент считается отсортированным.

Берем следующий элемент массива, сравниваем его с последним отсортированным, если сравниваемый элемент больше, то меняем их местами.

Повторяем эти действия пока не встретиться число меньшее или не достигнем левого края массива.

Повторяем действия 1 – 3 пока массив не будет отсортирован.

Пример: Таблица 2. Исходный массив: 6 1 7 2 9

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № Шага | Массив до выполнения шага | | | | | Описание | Массив после выполнения шага. | | | | |
| 1. | 6 | 1 | 7 | 2 | 9 | Меняем элементы 1 и 6 местами. | 1 | 6 | 7 | 2 | 9 |
| 2. | 1 | 6 | 7 | 2 | 9 | Элемент на своем месте. | 1 | 6 | 7 | 2 | 9 |
| 3. | 1 | 6 | 7 | 2 | 9 | Меняем элементы 2 и 7 местами. | 1 | 6 | 2 | 7 | 9 |
| 1 | 6 | 2 | 7 | 9 | Меняем элементы 2 и 6 местами. | 1 | 2 | 6 | 7 | 9 |
| 4. | 1 | 2 | 6 | 7 | 9 | Элемент на своем месте. | 1 | 2 | 6 | 7 | 9 |

1. Пример сортировки простыми вставками.

Отсортированный массив: 1 2 6 7 9

### Пузырьковая сортировка

Описание: алгоритм по сортировке массива. Сложность сортировки: Ɵ(N2). Память: Ɵ(1).

Алгоритм: состоит из повторяющихся проходов по массиву. За каждый проход сравниваются два соседних элемента, если порядок нарушен, то происходит обмен элементов. При каждом проходе по массиву, наибольший элемент становится на свое место в конце массива, рядом с предыдущим наибольшим элементом.

Пример: Таблица 3. Исходный массив: 6 1 7 2 9

1. Пример пузырьковой сортировки.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| № Шага | 1 |  | 2 |  | 3 |  | 4 |
| Массив | 6 |  | 9 |  | 9 |  | 9 |
| 1 | 6 | 7 | 7 |
| 7 | 1 | 6 | 6 |
| 2 | 7 | 1 | 2 |
| 9 | 2 | 2 | 1 |

Отсортированный массив: 1 2 6 7 9

### Сортировка подсчетом

Описание: алгоритм по сортировке массива целых чисел в диапазоне K (K = Максимальный элемент – минимальный элемент + 1), работающий за линейное время. Сложность сортировки: Ɵ(N). Память: дополнительный массив из K элементов, где K диапазон чисел.

Алгоритм:

Создаем дополнительный массив C[K], состоящий из 0.

Проходим по изначальному массиву A[N] и значение C[A[i]] увеличиваем на единицу.

Записываем в массив A число i C[i] раз. ( i от 0 до k – 1)

Пример: Таблица 4. Исходный массив: 6 1 7 2 9

1. Пример сортировки подсчетом.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Индексы | | | | | | | | | |
| Массив | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| A | 6 | 1 | 7 | 2 | 9 |  |  |  |  |  |
| C | 0 | 1 | 1 | 0 | 0 | 0 | 1 | 1 | 0 | 1 |
| A | 1 | 2 | 6 | 7 | 9 |  |  |  |  |  |

Отсортированный массив: 1 2 6 7 9

### Быстрая сортировка

Описание: рекурсивный алгоритм по сортировке массива. Сложность сортировки: Ɵ(N\*log2N). Память: Ɵ(1).

Алгоритм:

Выбираем ключевой элемент и делим массив на две части. (Ключевой элемент можно выбрать случайным образом или взять средний элемент)

Перемещаем все элементы больше ключевого справа от него, а все элементы меньше ключевого влево. (Ключевой элемент приобрел правильную позицию)

Повторяем такие же действия для левых и правых частей массива.

Массив считается отсортированным, когда в каждой части останется по 1 элементу.

Пример: Таблица 5. Исходный массив: 6 1 7 2 9

1. Пример быстрой сортировки

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | Массив | | | | | | |
| № Шага | 0 | 1 | 2 | | 3 | | 4 |
| 1. | 6 | 1 | 7 | | 2 | | 9 |
|  | | | | | | | |
| 2. | 6 | 1 | 2 |  | | 7 | 9 |
|  | | | | | | | |
| 3. | 1 | 6 | 2 |  | | 7 | 9 |
|  | | | | | | | |
| 4. | 1 | 2 | 6 | | 7 | | 9 |

Отсортированный массив: 1 2 6 7 9

### Сортировка слиянием

Описание: рекурсивный алгоритм по сортировке массива. Сложность сортировки: Ɵ(N\*log2N). Память: Ɵ(N).

Алгоритм:

Рекурсивно делим массив на две части, примерно одинакового размера. Делим до тех пор, пока массив не будет состоять из 1 элемента.

Соединяем два упорядоченных массива в один, одновременно сортируя их. Если один из-под массивов закончился, то записываем все оставшиеся элементы в результирующий массив.

Пример: Таблица 6. Исходный массив: 6 1 7 2 9

1. Пример сортировки слиянием

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 6 | 1 | 7 | 2 | 9 |
|  | | | | |
| 1 | 6 | 7 | 2 | 9 |
|  | | |  | |
| 1 | 6 | 7 |
|  | | |
| 1 | 2 | 6 | 7 | 9 |

Отсортированный массив: 1 2 6 7 9

## Описание функций

void swapint(int \*n1, int \*n2)

**Назначение:** Меняет два элемента массива (тип int) местами.

**Входные параметры:** **n1** - 1 элемент, **n2** - 2 элемент.

**Выходные данные:** Отсутствуют

void swaplong(ULONGLONG \*n1, ULONGLONG \*n1)

**Назначение:** Меняет два элемента массива (тип ULONGLONG) местами.

**Входные параметры:** **n1** - 1 элемент, **n2** - 2 элемент.

**Выходные данные:** Отсутствуют

int ListDirectoryContents(const wchar\_t \*sDir, wchar\_t \*\*\*fileNames, ULONGLONG\*\* filesize)

**Назначение:** Проверяет правильность ввода пути, считает количество файлов в папке и записывает имена и размеры файлов в массивы.

**Входные параметры: sDir** – строка, содержащая путь к папке,

**fileNames** – Массив для записи полных имен файлов,

**filesize** – Массив для записи размеров файлов.

**Выходные данные:** Количество файлов в папке.

void entpath(wchar\_t \*\*path)

**Назначение:** Ввод пути к файлам.

**Входные параметры:** path – указатель на массив, в котором будет записанное введенный пользователем путь.

**Выходные данные:** Отсутствуют.

int\* choose(ULONGLONG\* a ,int n)

**Назначение:** Реализует сортировку выбором для списка файлов.

**Входные параметры:** **a** - Массив размеров файлов,

**n** – Количество элементов в массиве.

**Выходные данные:** Массив с новыми индексами для вывода отсортированного списка.

int\* insert(ULONGLONG\* a ,int n)

**Назначение:** Реализует сортировку вставками для списка файлов.

**Входные параметры:** **a** - Массив размеров файлов,

**n** – Количество элементов в массиве.

**Выходные данные:** Массив с новыми индексами для вывода отсортированного списка.

int\* bubble(ULONGLONG\* a ,int n)

**Назначение:** Реализует пузырьковую сортировку для списка файлов.

**Входные параметры:** **a** - Массив размеров файлов,

**n** – Количество элементов в массиве.

**Выходные данные:** Массив с новыми индексами для вывода отсортированного списка.

int\* countingsort(ULONGLONG\* a ,int n)

**Назначение:** Реализует сортировку подсчетом для списка файлов.

**Входные параметры:** **a** - Массив размеров файлов,

**n** – Количество элементов в массиве.

**Выходные данные:** Массив с новыми индексами для вывода отсортированного списка.

void quicksort(ULONGLONG\* size, int n1, int n2, int\* newId)

**Назначение:** Реализует быструю сортировку для списка файлов.

**Входные параметры:** **size** – Скопированный массив размеров файлов,

**n1** - Индекс первого элемента массива,

**n2** - Индекс последнего элемента массива,

**newId** - Массив индексов для массива размеров файлов.

**Выходные данные:** Отсутствуют.

int\* quicksort\_memory(ULONGLONG\* size, int n1, int n2)

**Назначение:** Создает дополнительные массивы для быстрой сортировки и запускает ее.

**Входные параметры:** **size** - Массив размеров файлов,

**n1** - Индекс первого элемента массива,

**n2** - Индекс последнего элемента массива,

**Выходные данные:** Массив с новыми индексами для вывода отсортированного списка.

void merge(ULONGLONG\* copy\_size, int\* id\_file, int l, int m, int r)

**Назначение:** Выполняет слияние двух подмассивов в сортировке слиянием.

**Входные параметры: copy\_size –** Скопированный массив размеров файлов,

**Id\_file –** Массив индексов файлов,

**l** - первый элемент массива,

**m –** средний элемент в массиве,

**r –** последний элемент массива.

**Выходные данные:** Отсутствуют.

void mergesort(ULONGLONG\* copy\_size, int\* id\_file, int l, int r)

**Назначение:** Реализует сортировку слиянием.

**Входные параметры: copy\_size –** Скопированный массив размеров файлов,

**Id\_file –** Массив индексов файлов,

**l** - первый элемент массива,

**r –** последний элемент массива.

**Выходные данные:** Отсутствуют.

int\* merge\_memory(ULONGLONG\* a, int n, int l, int r)

**Назначение:** Создает дополнительные массивы для сортировки слиянием и запускает ее.

**Входные параметры: a –** Массив размеров файлов,

**n** **-** Количество элементов в массиве,

**l** - первый элемент массива,

**r –** последний элемент массива.

**Выходные данные:** Массив с новыми индексами для вывода отсортированного списка.

void menu()

**Назначение:** Вывод меню, для выбора сортировки.

**Входные параметры:** Отсутствуют.

**Выходные данные:** Отсутствуют.

void print\_newId(ULONGLONG\* filesize, wchar\_t \*\*fileNames, int count\_files, int\* newId)

**Назначение:** Вывод отсортированного списка.

**Входные параметры: filesize –** Массив размеров файлов,

**fileNames –** Массив имен файлов,

**count\_files –** Количество файлов,

**newId –** Массив индексов для массива размеров файлов.

**Выходные данные:** Отсутствуют.

void main()

**Назначение:** Основная функция программы.

**Входные параметры:** Отсутствуют.

# Заключение

В ходе выполнения практической работы был разработан и реализован файловый менеджер с функцией показа списка файлов в заданной папке, сортировки файлов, вывода отсортированного списка и подсчет времени сортировки данных. Для сортировки используется 6 способов: сортировка выбором, сортировка простыми вставками, пузырьковая сортировка, сортировка подсчетом, быстрая сортировка, сортировка слиянием. В таблице 7 приведена характеристика данных сортировок время выполнения и требуемая память.

1. Сложность методов сортировки.

|  |  |  |
| --- | --- | --- |
| Методы сортировки | Средняя сложность | |
| Время выполнения | Память |
| Сортировка выбором | Ɵ(N2). | Ɵ(1). |
| Сортировка простыми вставками | Ɵ(N2). | Ɵ(1). |
| Пузырьковая сортировка | Ɵ(N2). | Ɵ(1). |
| Сортировка подсчетом | Ɵ(N). | (Max – min + 1) |
| Быстрая сортировка | Ɵ (N\*log2N) | Ɵ(1). |
| Сортировка слиянием | Ɵ (N\*log2N) | Ɵ(N). |

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