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**

**Файловый менеджер**

Выполнила: Молоткова Светлана Сергеевна, студентка 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*

Ежедневно мы с работаем с файлами. Это требует времени: ведь найти нужный файл в длинном списке довольно сложно.

Для простоты ориентирования в них необходима сортировка, например, по имени, размерам, дате создания или типу. Это поможет пользователю разобраться в большом количестве файлов быстрее.

Одна из актуальных сортировок является сортировка по размерам, которая будет реализована. Разные сортировки справляются с заданной задачей за разное время.

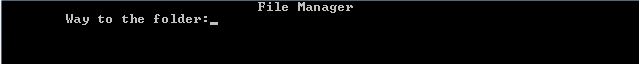
# *Постановка задачи*

* Задача: разработать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по размеру.
* Входные данные: путь до директории, в котором необходимо отсортировать содержимое; способ сортировки(6 вариантов возрастание или убывание)
* Выходные данные: отсортированный список имен файлов с указанием размеров, время сортировки.

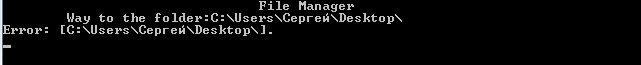
# *Руководство пользователя*

Прилагаются инструкции по работе с программой менеджера.

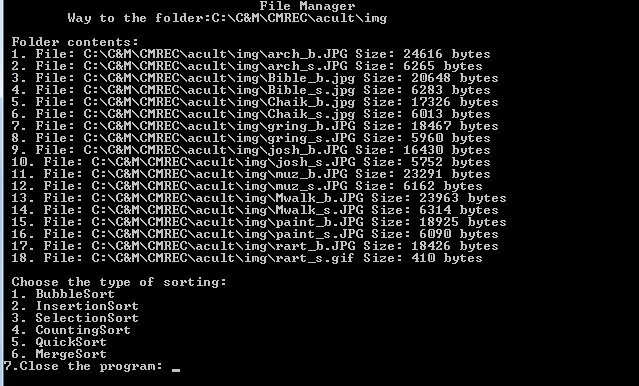
1. Запустите файл **Practice5.exe**. Затем на приветственный экран с названием программы и просьбой ввести путь к папке, из которой нужно отобразить файлы.



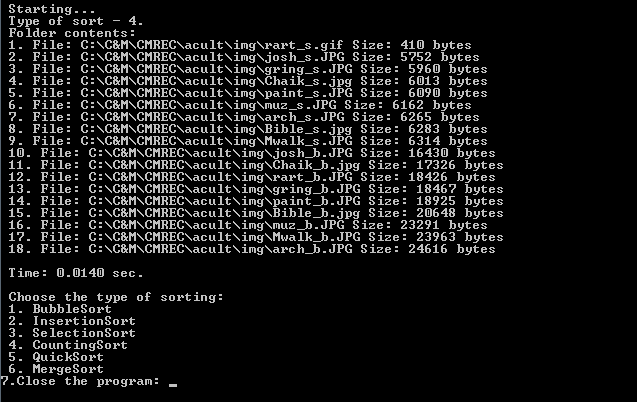
1. Программа после запуска.
2. Введите полный путь к папке и нажмите Enter. В случаях, когда указан неверный путь или папка пуста, отобразится сообщение об ошибке (см. Рис 2). Проверьте правильность ввода пути и повторите попытку.



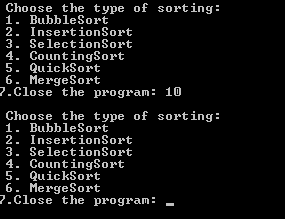
1. Сообщение об ошибке при некорректоного пути.



1. Программа после ввода корректного пути к непустой папке.
2. Программа выведет список файлов и их размеров, а так же варианты сортировок. Введите цифру от 1 до 6, соответствующую выбранному вами методу, и нажмите Enter. (Внимание: сортировка подсчетом в том случае, когда размеры файлов в которых находятся в слишком большом диапазоне будет работать некорректно.) Программа после ввода корректного номера (см. Рис 4). Если введен некорректный номер, меню на экране появится еще раз (см. Рис 5).



1. Экран программы после ввода нужного метода сортировки.
2. Если номер метода сортировки указан верно, файлы с указанием размеров будут выведены в порядке возрастания в байтах а также время, затраченное на сортировку.



1. Введен неверный номер метода сортировки.
2. После этого вы можете выбрать другой метод сортировки или закрыть программу. Для этого нажмите «крестик» в правом верхнем углу окна.

# *Руководство программиста*

## Структура программы

Весь код программы находится в файле **file\_manager.c**.

## Описание алгоритма

Файлы сортируются по возрастанию способами, описанными ниже.

### Сортировка выбором

Принципы алгоритма сортировки выбором с примером:

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.

1. Пример сортировки выбором.

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Индексы  Элементы | Массив | | | | | | min el\index: | | С каким элементом менять | | |
| *0* | *1* | *2* | *3* | *4* | *5* | |  | |  |
| 2 | 10 | -3 | 11 | 8 | 9 | |
|  | **-3** | 10 | 2 | 11 | 8 | 9 | | -3\2 | | 0( так как крайний левый) |
|  | **-3** | **2** | 10 | 11 | 8 | 9 | | 2\2 | | 10(-3 не учитывается) |
|  | **-3** | **2** | **8** | 11 | 10 | 9 | | 8\4 | | 10 |
|  | **-3** | **2** | **8** | **9** | 10 | 11 | | 9\5 | | 11 |

### Сортировка простыми вставками

Принципы алгоритма сортировки простыми вставками с примером:

Считаем первый элемент массива отсортированным.

Из оставшихся элементов выберем первый и сравним с последним отсортированным. Если отсортированный элемент больше – они меняются местами и элемент добавляется в новый отсортированный массив.

Эту операцию повторяем до тех пор, пока не встретится число меньше. Если число меньше, то в новом массиве, начиная с конца, сравниваем элементы с выбранным( если попадется элемент равный или меньший, то выбранный заменит его позицию, в противном случае переместится в начало).

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

1. Пример сортировки простыми вставками.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Индексы  Элементы | Массив | | | | | | | | Сравниваемый элемент(индекс) | Нужно ли менять с предыдущим |
| *0* | *1* | *2* | *3* | *4* | *5* |  |  |
| **9** | 6 | 7 | 8 | 4 | 5 |  |  | 1 | да |
|  | **6** | **9** | 7 | 8 | 4 | 5 |  |  | 2 | да |
| **6** | **7** | **9** | 8 | 4 | 5 |  |  | 3 | да |
| **6** | **7** | **8** | **9** | 4 | 5 |  |  | 4 | да |
| **4** | **6** | **7** | **8** | 9 | 5 |  |  | 5 | да |
| **4** | **5** | **6** | **7** | **8** | **9** |  |  |  |  |

### Пузырьковая сортировка

Принципы алгоритма сортировки пузырьком с примером:

Идем от последнего элемента массива, попарно сравнивая элементы. Если последний элемент больше предпоследнего, то не меняем их местами. Если меньше - меняем.

Пройдя так один раз весь массив, получим меньший элемент. Пункт 1 повторяем столько раз, сколько элементов в массиве, двигаясь каждый раз от конечного элемента к новому начальному до тех пор, пока массив не будет полностью отсортирован.

1. Пузырьковая сортировка(1 проход).

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Индексы  Элементы | Массив | | | | | | Сравниваемые элементы | | Меняем ли местами |
| *0* | *1* | *2* | *3* | *4* | *5* |  | | | |
| 5 | 10 | 8 | 7 | 3 | 4 | 3<4 | - | | |
| 5 | 10 | 8 | 3 | 7 | 4 | 7>3 | + | | |
| 5 | 10 | 3 | 8 | 7 | 4 | 8>3 | + | | |
| 5 | 3 | 10 | 8 | 7 | 4 | 10>3 | + | | |
| 3 | 5 | 10 | 8 | 7 | 4 | 5>3 | + | | |

### Сортировка подсчетом

Принципы алгоритма сортировки подсчетом с примером :

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

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

Далее идем по исходному массиву, каждый раз прибавляя 1 к элементу дополнительного массива, индекс которого равен значению текущего элемента в сортируемом массиве.

После этого заново заполняем массив: присвоим первоначальному массиву индексы дополнительного, столько раз, сколько хранится в ячейке под таким индексом .

Сортировка подсчетом используется для индексируемых данных.

1. Сортировка подсчетом.

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

### Быстрая сортировка

Принципы алгоритма быстрой сортировки с примером:

Выбираем ведущий элемент.

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

Сортируем аналогично две получившиеся части( дальше таких частей будет больше).

Закончить сортировку, когда каждая часть будет состоять из одного элемента.

1. Быстрая сортировка.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Массив | | | | | |
| *0* | *1* | *2* | *3* | *4* | *5* |
| 9 | 11 | 3 | **8** | 14 | 12 |
| 3 | 8 | 9 | 11 | **14** | 12 |
| 3 | 8 | 9 | 11 | 12 | 14 |

### Сортировка слиянием

Принцип алгоритма сортировки слиянием с примером:

Разбиваем массив на части таким образом ,что каждая из них состоит из 1 элемента, так как такой массив считается отсортированным.

Рассмотрим вначале пары элементов, меняя их порядок, если первое значение больше второго -меняем их местами. Сольем их в пары.

Затем рассмотрим пары, состоящие из двух элементов, упорядоченно составим из них массив из 4 элементов и т.д., то есть функция рекурсивная.

1. Сортировка слиянием.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Имеем 7 элементов, получили 4 пары, затем 2, затем слили их | Массив | | | | | | | |
| *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* |
| 3 | 9 | 5 | 2 | 1 | 7 | 3 | 8 |
| 3,9 | | 2,5 | | 1,7 | | 3,8 | |
| 2,3,5,9 | | | | 1,3,7,8 | | | |
| 12334789 | | | | | | | |

## Описание функций

void Menu(\*method)

* Назначение: вывод меню.
* Входные параметры: переменная, отвечающая за тип сортировки.

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

* Назначение: вводим путь к директории.
* Входные параметры: sDir – указатель на массив с введенным пользователем путем.

void Output(wchar\_t \*\*filesNames, ULONGLONG \*\*filesSizes, int \*filesIdex, int N)

* Назначение: вывод списка файлов и их размера в байтах и их количество.
* Входные параметры: wchar \*\*filesNames– двумерный массив с именами файлов, \*filesSize- массив с размерами файлов, \*filesIndex- массив индексов, N-количество файлов.

int ListDirectoryContents(const wchar\_t \*sDir, wchar\_t \*\*filesNames, ULONGLONG \*filesSizes)

* Назначение: запись массива из имен и размеров файлов и их количество.
* Входные параметры: const wchar\_t \*sDir -массив с путем до директории, wchar\_t \*\*filesName - двумерный массив с именами файлов из директории, ULONGLONG \*filesSize - массив с размерами в байтах.
* Выходные параметры: количество файлов.

void swap\_int(int \*z, int \*k)

* Назначение: обмен значений переменных типа int.
* Входные параметры:z, k– указатели на переменные, значения которых нужно поменять местами.

void swap\_ULONGLONG(ULONGLONG \*z, ULONGLONG \*k)

* Назначение: обмен значений переменных типа ULONGLONG.
* Входные параметры: z, k – указатели на переменные, значения которых нужно поменять местами.

void SelectionSort(ULONGLONG \*filesSize, int \*filesIndex, int N)

* Назначение: сортировка выбором.
* Входные параметры: filesSize – массив размеров, filesIndex – массив индексов файлов, N – количество элементов в массиве.

void InsertionSort(ULONGLONG \*filesSize, int \*filesIndex, int N)

* Назначение: сортировка вставками .
* Входные параметры: filesSize – массив размеров файлов, filesIndex – массив индексов файлов, N– количество элементов в массиве.

void BubbleSort(ULONGLONG \*filesSize, int \*filesIndex, int N)

* Назначение: пузырьковая сортировка.
* Входные параметры: filesSize – массив размеров, filesIndex – массив индексов файлов, N – количество элементов в массиве.

void CountingSort(ULONGLONG \*filesSizes, int \*filesIndex, int N)

* Назначение: сортировка подсчетом.
* Входные параметры: filesSize – массив размеров файлов, filesIndex – массив индексов файлов, N – количество элементов в массиве.

void QuickSplit(ULONGLONG \*filesSize, int \*i, int \*j, int mid, int \*filesIndex)

* Назначение: сортировка части подмассива.
* Входные параметры: \*filesSize –массив размеров файлов, i, j – указатели переменные, отвечающие за индексы крайних элементов подмассива, mid –переменная выбранного элемента, \*filesIndex –массив индексов файлов.
* Выходные параметры: отсутствуют.

void QuickSort(ULONGLONG \*filesSize, int \*filesIndex, int first, int last)

* Назначение: разделение массива на подмассивы.
* Входные параметры: filesSize – массив размеров файлов, filesIndex – массив индексов файлов, first, last– индексы крайних элементов подмассива.

void Merge(ULONGLONG \*filesSize, int \*filesIndex, int first, int midIndex, int last)

* Назначение: сортировка слиянием упорядоченных элементов массива.
* Входные параметры: filesSize – массив размеров файлов, которые нужно упорядочить, filesIndex – массив индексов файлов, размеры которых необходимо отсортировать, first, last, midIndex – индексы крайних элементов сортируемого фрагмента массива и середины.

void MergeSorted(ULONGLONG \*filesSize, int \*filesIndex, int first, int last)

* Назначение: разделение на подмассивы.
* Входные параметры: filesSize – массив, который части которого необходимо слить, filesIndex – массив индексов файлов, first, last– индексы первого и последнего элемента массива, части которого нужно слить.

void main()

* Назначение: основная функция.
* Входные параметры: отсутствуют.

# *Заключение*

В ходе выполнения практической работы был разработан и реализован файловый менеджер с функцией показа файлов в заданной директории, упорядоченных по размеру. Для сортировки использовалось 6 методов, которые может выбрать пользователь. Подсчет затраченного времени позволяет сделать вывод о лучшей сортировке. В таблице приведены сравнения затраченного времени сортировок. Выберем два наиболее значимых параметра: время и память. n- количество элементов массива.

1. Сравнение 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) |