Министерство науки и высшего образования РФ

Федеральное государственное автономное образовательное учреждение высшего образования "Национальный исследовательский Нижегородский государственный университет им. Н.И. Лобачевского" (ННГУ)

Институт информационных технологий, математики и механики

Отчет по учебной практике

**Прототип файлового менеджера с функцией показа файлов в заданном каталоге**

**Выполнил:**

Студент группы 381806-3 Тюрмина А.Н.

**Проверил:**

Доцент кафедры МОСТ ИИТММ

Кустикова В.Д.

Нижний Новгород

1. г.

**Содержание**

[Введение 3](#_Toc533556253)

[1 Постановка задач 4](#_Toc533556254)

[2 Руководство пользователя 5](#_Toc533556255)

[3 Руководство программиста 7](#_Toc533556256)

[3.1 Структура программы 7](#_Toc533556257)

[3.2 Описание алгоритмов 7](#_Toc533556258)

[3.2.1 Сортировка выбором 7](#_Toc533556259)

[3.2.2 Сортировка вставками 7](#_Toc533556260)

[3.2.3 Сортировка пузырьком 8](#_Toc533556261)

[3.2.4 Сортировка подсчетом 9](#_Toc533556262)

[3.2.5 Сортировка слиянием 10](#_Toc533556263)

[3.2.6 Быстрая сортировка(Хоара) 10](#_Toc533556264)

[3.3 Описание функций 11](#_Toc533556265)

[Заключение 13](#_Toc533556266)

[Литература 14](#_Toc533556267)

# Введение

Одним из важнейших процедур обработки информации является сортировка. Целью алгоритмов сортировки является упорядочение последовательности элементов данных. Сортировки необходимы для выполнения различных задач, в которых необходима упорядоченность данных. Поиск элемента в неотсортированном массиве занимает времени больше, чем в отсортированном, что наводит на вывод, что благодаря сортировке повышается производительность труда. Существует множество метод сортировки данных. Однако каждый алгоритм сортировки можно разбить на три основные части:

* Сравнение, определяющее упорядоченность пары элементов.
* Перестановка, меняющая местами пару элементов.
* Сортирующий алгоритм, который осуществляет сравнение и перестановку элементов данных до тех пор, пока все эти элементы не буду упорядочены.

Важной характеристикой любого алгоритма является время работы. Время сортировки будет пропорционально количеству сравнений и перестановки элементов данных в процессе их сортировки.

# Постановка задач

В данной работе необходимо реализовать программу файлового менеджера, которая отсортирует файлы по возрастанию размера. При этом используются различные методы сортировки, такие как:

* «пузырьком»
* Выбором
* Вставками
* Слиянием
* Хоара (так же известная, как «Быстрая»)
* подсчетом

Входные данные:

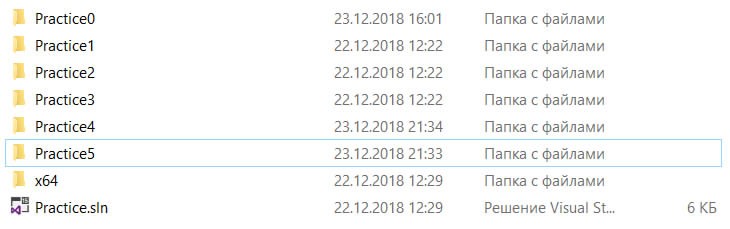
* Путь до директории, в которой необходимо отсортировать содержимое
* Метод сортировки

Выходные данные:

* Отсортированный список имен файлов с указанием размера
* Время сортировки

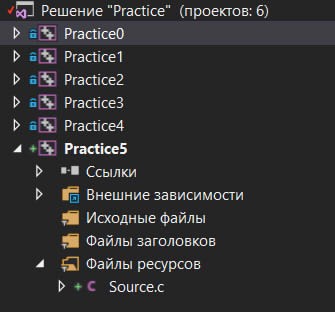
# Руководство пользователя

Запустить проект под названием: «Practice.sln».



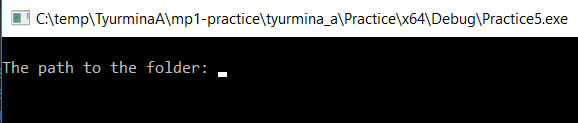
1. Нахождение в папке

Выбрать Practice5 – Файлы ресурсов – Source.c и нажать F5.



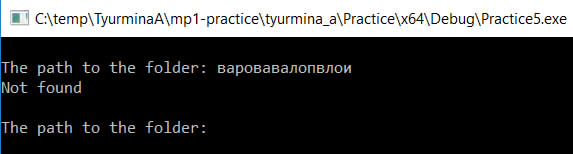
1. Выбрать нужный среди проектов

Запустится программа. Она предложит ввести путь. ()



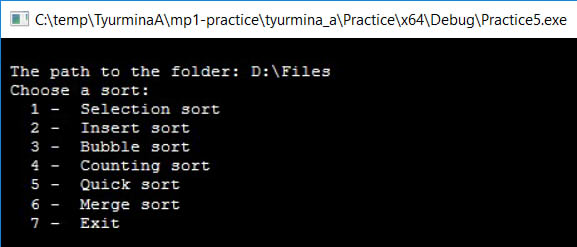
1. Начальное положение

Если вводить неверный путь, то программа напишет «Not found», что значит, что путь не найден и предложит ввести снова. ()



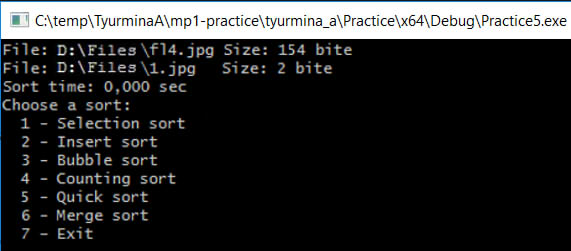
1. Условия выхода из программы

Вводя верный путь, предлагается на выбор один из методов сортировки. ()



1. Предложение о смене директории

Выбираете нужную сортировку и вам выводятся уже отсортированные файлы, а так же время, потраченное на сортировку. !!!!!!!!!!



Далее, вы можете отсортировать снова, но уже используя другой метод или же выйти, нажав цифру 7.

# Руководство программиста

## Структура программы

Файл Source.c, в котором содержатся весь код программы. Сама программа состоит из функций сортировок, ввода пути, вывода отсортированных файлов, их размер и время, затраченное на сортировку.

## Описание алгоритмов

Файлы сортируются по возрастанию способами, указанными ниже.

### Сортировка выбором

Идея метода состоит в том, чтобы создавать отсортированную последовательность путем присоединения к ней одного элемента за другим в правильном порядке.

Будем строить готовую последовательность, начиная с левого конца массива. Алгоритм состоит из n последовательных шагов, начиная от нулевого и заканчивая (n-1)-м.

На i-м шаге выбираем наименьший из элементов a[i] ... a[n] и меняем его местами с a[i]. Последовательность шагов при n=5 изображена на рисунке ниже.

1. Пример сортировки выбором

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Индекс  Шаг | Массив | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 |
| Исходный массив | 4 | 9 | 7 | 6 | 2 | 3 |
| 1 | **2** | 9 | 7 | 6 | 4 | 3 |
| 2 | **2** | **3** | 7 | 6 | 4 | 9 |
| 3 | **2** | **3** | **4** | 6 | 7 | 9 |
| 4 | **2** | **3** | **4** | **6** | 7 | 9 |
| 5 | **2** | **3** | **4** | **6** | **7** | 9 |
| Отсортированный массив | **2** | **3** | **4** | **6** | **7** | **9** |

### Сортировка вставками

Суть алгоритма заключается в том, что на каждом шаге выбирается элемент из неотсортированной части массива, после чего он вставляется на нужную позицию в уже отсортированной части массива. Выбор элемента из массива можно осуществлять произвольно, но удобнее брать их по порядку.

1. Пример сортировки вставками

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Индекс  Шаг | Массив | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 |
| Исходный массив | 2 | 0 | -3 | -8 | 1 | 3 |
| 1 | 0 | 2 | -3 | -8 | 1 | 3 |
| 2 | 0 | -3 | 2 | -8 | 1 | 3 |
| -3 | 0 | 2 | -8 | 1 | 3 |
| 3 | -3 | 0 | -8 | 2 | 1 | 3 |
| -3 | -8 | 0 | 2 | 1 | 3 |
| **-8** | **-3** | **0** | 2 | 1 | 3 |
| 4 | **-8** | **-3** | **0** | **1** | **2** | 3 |
| Отсортированный массив | **-8** | **-3** | **0** | **1** | **2** | **3** |

### Сортировка пузырьком

Как известно, воздух легче воды, поэтому пузырьки воздуха всплывают. Это просто аналогия. В сортировке методом пузырька по возрастанию более легкие (с меньшим значением) элементы постепенно «всплывают» в начало массива, а более тяжелые (с большим значением) друг за другом опускаются на дно (в конец массива).

1. Пример сортировки пузырьком

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Индекс  Шаг | Массив | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 |
| Исходный массив | 1 | 5 | 2 | 7 | 6 | 3 |
| 1 | **1** | **2** | 5 | 7 | 6 | 3 |
| 2 | 1 | 2 | 5 | 7 | 3 | 6 |
| 1 | 2 | 5 | 3 | 7 | 6 |
| **1** | **2** | **3** | 5 | 7 | 6 |
| 3 | **1** | **2** | **3** | **5** | **6** | 7 |
| Отсортированный массив | **1** | **2** | **3** | **5** | **6** | **7** |

### Сортировка подсчетом

Идея состоит в том, что для каждого элемента найти, сколько элементов, меньших определенного числа, и поместить это число на соответствующее место. Делается это так: за линейный проход по массиву мы для каждого из возможных значений подсчитываем, сколько элементов имеют такое значение. Потом добавляем к каждому из найденных чисел сумму всех предыдущих. Получая, таким образом, сколько элементов, значения которых не больше данного значения. Далее, опять-таки за линейный проход, формируем из исходного массива новый отсортированный. При этом следим, чтобы два одинаковых элемента не были записаны в одно место.

1. Пример сортировки подсчетом

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Индекс | Массив | | | | | | |
| 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Исходный массив | 2 | 1 | 0 | 1 | 1 | 2 | 3 |
| Отсортированный массив | **0** | **1** | **1** | **1** | **2** | **2** | **3** |

1. Массив подсчета

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Индексы | 0 | 1 | 2 | 3 |
| Исходные данные | 0 | 0 | 0 | 0 |
|  | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 0 |
| 1 | 2 | 1 | 0 |
| 1 | 3 | 1 | 0 |
| 1 | 3 | 2 | 0 |
| Итоговые данные | 1 | 3 | 2 | 1 |

### Сортировка слиянием

В основе сортировки слиянием лежит принцип «разделяй и властвуй». Массив разделяется на равные или практически равные части, каждая из которых сортируется отдельно. После чего уже упорядоченные части сливаются воедино.

1. Пример сортировки слиянием

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № шага | Массив | | | | | | | |
| Исходный массив | 2 | 6 | 7 | 1 | 3 | 5 | 0 | 4 |
| 1 | 2 | 6 | | 1 | 7 | | 3 | 5 | | 0 | 4 | |
| 2 | 1 | 2 | 6 | 7 | | | | 0 | 3 | 4 | 5 | | | |
| Отсортированный массив | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |

### Быстрая сортировка (Хоара)

Принцип работы таков: значение какого-нибудь элемента, обычно центрального, записывается в переменную Х. Просматриваются элементы массива. При движении слева - направо ищем элемент больше или равный Х. А при движении справа – налево ищем элемент меньше или равный Х. Найденные элементы меняются местами и продолжается встречный поиск.

После этого массив окажется разделенным на две части. В первой находятся элементы меньше либо равные Х, а справа – больше или равные Х.

Выбор среднего значения можно делать или случайно или поиском некоторого среднего значения, например посередине массива или выбрав действительно средний по значению элемент.

1. Пример быстрой сортировки

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Индексы  Шаги | 0 | 1 | 2 | 3 | 4 | 5 | 6 |
| Исходный массив | 5 | 8 | 6 | 4 | 2 | 1 | 1 |
| 1 | 2 | 1 | 1 | 4 | 5 | 8 | 6 |
| Отсортированный массив | 1 | 1 | 2 | 4 | 7 | 8 | 9 |

## Описание функций

**Функция** void SelectionSort (int \*a, ULONGLONG \*size, int n)

Функция сортировки массива размера файлов методом выбора.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* n – количество файлов

**Функция** void InsertSort (int \*a, ULONGLONG \*size, int n)

Функция сортировки массива размера файлов методом вставки.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* n – количество файлов

**Функция** void BubbleSort (int \*a, ULONGLONG \*size, int n)

Функция сортировки массива размера файлов методом пузырька.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* n – количество файлов

**Функция** int CountingSort (int \*a, ULONGLONG \*size, int n)

Функция сортировки массива размера файлов методом подсчета.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* n – количество файлов

**Функция** void QuickSplit (int \*a, ULONGLONG \*size, int \*i, int \*j, ULONGLONG p)

Функция создана для быстрой сортировки, т.е. сортирует файлы меньше выбранного влево, а те, что больше – вправо.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* \*i – адрес первого индекса
* \*j - адрес второго индекса
* p - выбранный элемент, по которому сортируются элементы

**Функция** void QuickSort (int \*a, ULONGLONG \*size, int N1, int N2)

Функция сортировки массива размера файлов методом Хоара.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* N1 – индекс первого элемента
* N2 - индекс второго элемента

**Функция** void Merge (int \*a, ULONGLONG \*size, int N1, int med, int N2)

Функция нужна для выполнения сортировки слиянием. Соединяет массив разделенный на части воедино, упорядочивая их.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* N1 – индекс первого элемента
* med - индекс среднего
* N2 - индекс последнего элемента

**Функция** void MergeSort (int \*a, ULONGLONG \*size, int N1, int N2)

Функция сортировки массива размера файлов методом слияния.

Входные данные:

* \*a – массив индексов массива размера файлов
* \*size – массив размеров файла
* N1 – индекс первого элемента
* N2 - индекс второго элемента

**Функция** int FileDirectory (const wchar\_t \*sDirect, ULONGLONG \*\*sizes, wchar\_t \*\*\*names)

Эта функция определяет правильность ввода данных, так же считает количество файлов в директории, формирует массив на основе данных (название файла и его размер).

Входные данные:

* \*sDirect - путь до каталога
* \*\*ssizes - массив размеров
* \*\*\*nnames - массив названий

**Функция** void input (wchar\_t \*\*strin)

В этой функции вводится путь к файлам.

Входные данные:

* \*\*strin – указатель на массив, в который будет записано введенное значение пути к файлам.

**Функция** void CommandMenu ()

Выводит на экран меню команд

**Функция** void Output (int \*Idxes, wchar\_t \*\*names, ULONGLONG \*sizes, int count)

Выводит на экран отсортированный список файлов по возрастанию размера, а также время сортировки.

Входные данные:

* \*Idxes – массив индексов отсортированного массива размера файлов
* \*\*names – массив названий
* \*sizes – массив размеров
* \*count – количество файлов

# Заключение

Выше были приведены различные методы сортировки. Для определения наиболее быстрой сортировки массивов был проведён эксперимент, в котором измерялось время, за которое каждая сортировка упорядочивает файлы по размеру. Для сортировки использовалось 6 методов, которые может выбрать пользователь. В таблице представлено сравнение времени работы и значения память, необходимые для разных видов сортировок.

1. Сложность методов сортировки

|  |  |  |
| --- | --- | --- |
| **Методы сортировки** | **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) |

# Литература

1. Сайт Algolist. Сортировка выбором – <http://algolist.manual.ru/sort/select_sort.php>.
2. Сайт vscode Сортировка вставками - <https://vscode.ru/prog-lessons/sortirovka-metodom-podscheta-na-si.html>
3. Сайт Pascal. Основы программирования. Сортировка пузырьком - <https://pas1.ru/bubbles>
4. Сайт vingrad. Сортировка подсчетом - <http://base.vingrad.ru/view/3011-Sortirovka-podschetom->
5. Сайт Kvodo. Сортировка слиянием - <http://kvodo.ru/mergesort.html>
6. Сайт STUDLAB.com – Студенческая лаборатория. Сортировка Хоара - <http://studlab.com/news/sortirovka_khoara/2011-05-31-116>