Министерство науки и высшего образования 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 лабораторной работе**

**«Моделирование работы сортировщика файлов ОС»**

**Выполнил**:

студент группы 381803-1

Кузенкова А.Е.

**Проверил**:

доцент кафедры МОСТ, к.т.н.,

Сысоев А.В.

Нижний Новгород

2018

Содержание

[Введение 3](#_Toc529541653)

[1. Постановка задачи 4](#_Toc529541654)

[2. Руководство пользователя 5](#_Toc529541655)

[3.1. Описание структуры программы 6](#_Toc529541656)

[3.2. Описание алгоритмов 6](#_Toc529541657)

[4. Результаты экспериментов 7](#_Toc529541658)

[Заключение 8](#_Toc529541659)

[Литература 9](#_Toc529541660)

[Приложение 10](#_Toc529541661)

# Введение

На сегодняшний день одной из важнейших операций, применяемых каждым пользователем ПК, является сортировка массивов. Она используется для достаточно быстрого и удобного нахождения нужной информации. В условиях повышения сложности решаемых в программировании задач, в которых необходимо работать с большими объёмами информации, куда выгоднее сначала упорядочивать используемый массив данных в нужном порядке, что значительно снизит затраты во времени при поиске какого-либо элемента из этого массива. Упорядочивание данных можно проводить по разным критериям в зависимости от того, с чем собирается работать пользователь или разработчик: упорядочивание по возрастанию/убыванию; в алфавитном порядке; по дате и времени и т.д.

# Постановка задачи

Разработать прототип файлового менеджера с функцией показа файлов в заданном каталоге, упорядоченных по возрастанию/убыванию размера.

Входные данные:

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

Выходные данные:

* Отсортированный список имен файлов с указанием размера.
* Время сортировки.

Программа должна предоставлять пользователю возможность сменить метод сортировки и повторно формировать выходные данные.

Программа должна реализовывать диалог с пользователем посредством интерфейса, который включает:

* возможность ввода пути до заданного каталога;
* возможность выбора метода сортировки;
* возможность просмотра отсортированного списка файлов с указанием размера.

# Руководство пользователя

В начале программы пользователю предлагается ввести путь до директории, причем нужного формата.

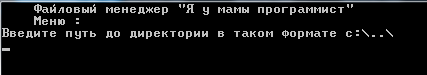


Рис. 1. Начало работы

При неправильном введении пути до директории, предлагает ввести заново

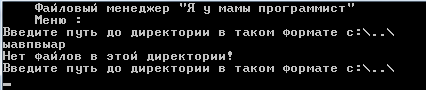


Рис. 2. Неправильный ввод пути

После введения пути, на экран выводится список файлов находящихся в данной директории, указывается их названия, размеры и количество файлов. Затем на экран выводится список предлагаемых сортировок.

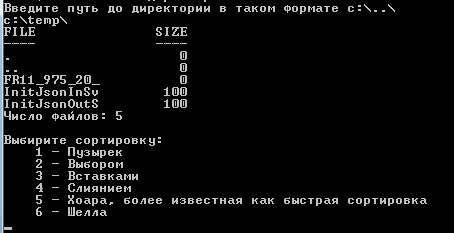


Рис. 3. Вывод списка файлов

При неправильном вводе номера сортировки, предлагает ввести заново.



Рис. 4. Пример неправильного ввода

После выбора сортировки предлагается выбрать, как отсортировать данные

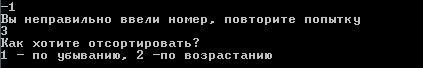


Рис. 5. Выбор порядка сортировки

При неправильном вводе номера, пользователя просят ввести заново. После правильного ввода выводит отсортированный список файлов с указанием времени работы этого вида сортировки.



Рис. 6. Вывод отсортированных файлов

Пользователю предлагается продолжить работу или выйти из программы, введя 0.



Рис. 7. Условие выхода из программы

Далее пользователь может ввести новый путь, либо отсортировать исходной директории с помощью другой сортировки. При вводе неправильного номера предлагает ввести заново.

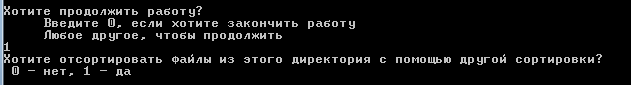


Рис. 8. Предложение остаться в старой директории

При вводе 1 программа выводит список файлов с предложением выбрать сортировку.

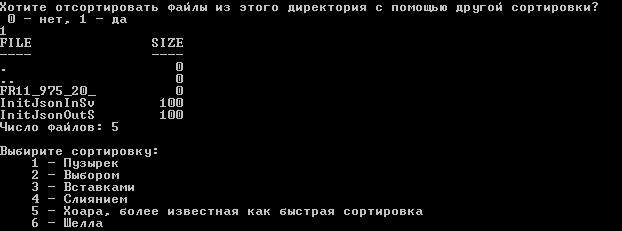


Рис. 9. Без смены директории

При вводе 0 программа предлагает ввести новый путь до директории.



Рис. 10. Попытка ввода новой директории

Таким образом, можно продолжать работу в данной программой с разными директориями. При желании пользователь всегда может выйти.

1. **Руководство программиста**

## Описание структуры программы

Программа написана на языке программирования СИ. Состоит из одного файла Source.cpp. Реализована структура datafail, которая хранит данные о файле: его размер и имя (максимум на 32 символа).

В программе реализованы следующие функции:

1. **Функция «Compare»**

Описание: сравнивает два элемента в большую или меньшую сторону, в зависимости от параметра k.

Принимает на вход две переменные типа datafail (a и b), которые нужно сравнить по их размеру, и целочисленную переменную k (определяет, как сравнивать).

На выход выдает 0 при (k = 1 и a < b ) или (k = 2 и a > b ), выдает 1 при (k = 1 и a > b ) или (k = 2 и a < b ).

1. **Функция «Buble\_sort»**

Описание: реализация сортировки пузырьком.

Принимает на вход массив типа datafail a, размер массива size (long), порядок сортировки k (int). В результате выполнения функции, массив a становится упорядоченным.

1. **Функция «Select\_sort»**

Описание: реализация сортировки выбором.

Принимает на вход массив типа datafail a, размер массива size (long), порядок сортировки k (int). В результате выполнения функции, массив a становится упорядоченным.

1. **Функция «Insert\_sort»**

Описание: реализация сортировки вставками.

Принимает на вход массив типа datafail a, размер массива size (long), порядок сортировки k (int). В результате выполнения функции, массив a становится упорядоченным.

1. **Функция «Merge\_sort»**

Описание: реализация сортировки слиянием.

Принимает на вход массив типа datafail a, индекс левой границы l (long), индекс правой границы r (long), порядок сортировки k (int). В результате выполнения функции, массив a становится упорядоченным.

1. **Функция «Merge»**

Описание: реализует слияние двух упорядоченных массивов.

Принимает на вход массив типа datafail a, индекс левой границы l (long), индекс середины mid (long), индекс правой границы r (long), порядок слияния k (int) (по убыванию или возрастанию элементов). В результате получается слитый массив a.

1. **Функция «Quick\_sort»**

Описание: реализация сортировки Хоара.

Принимает на вход массив типа datafail a, размер массива size (long), порядок сортировки k (int). В результате выполнения функции, массив a становится упорядоченным.

1. **Функция «Shell\_sort»**

Описание: реализация сортировки Шелла.

Принимает на вход массив типа datafail a, размер массива size (long), порядок сортировки k (int). В результате выполнения функции, массив a становится упорядоченным.

1. **Функция «Increment»**

Описание: Подсчет приращения для сортировки Шелла по формуле Седжвика.

Принимает на вход массив целого типа inc и размер массива size (long), который нужно отсортировать. В результате возвращает целого типа значение приращения.

## Описание алгоритмов

В программе используются следующие алгоритмы сортировок:

* пузырьком
* выбором
* вставками
* слиянием
* Хоара
* Шелла

**Сортировка пузырьком:**

Идея метода этой сортировки состоит в проходе по массиву снизу вверх. При этом сравниваются соседние элементы. Если они расположены неправильно, то их меняем местами. После нулевого прохода по массиву (при сортировке по возрастанию) самый «легкий» элемент оказывается «вверху» - отсюда аналогия с пузырьком. Следующий проход делается до второго элемента сверху, и второй по величине элемент будет поставлен на свое место. Далее делаются проходы по все уменьшающейся части массива до тех пор, пока не останется последовательность из одного элемента.

Ниже приведен псевдокод алгоритма (по возрастанию):

**Функция Пузырек** (**тип данных** a (масссив), **цел** size)

{

**цел** i, j;

**тип данных** с;

**цикл для** i от 0 до size-1

**цикл для** j от size-1 до i+1

**если** (a[ j ] < a[ j-1 ]), **то**

{

c = a[ j ];

a[ j ] = a[ j-1 ];

a[ j-1 ] = c;

}

}

**Сортировка выбором:**

Алгоритм этой сортировки заключается в постоянном нахождении минимального или максимального (в зависимости от того как мы хоти отсортировать) элемента в массиве. Найденный элемент меняется позицией с первым. Затем рассмотрению подлежит неотсортированная часть массива, исключая найденный ранее элемент. В этой части также проводится поиск минимального (максимального) элемента и после его нахождения, он меняется позицией со вторым элементом. Такой процесс поиска и обмена местами будет продолжаться, пока массив не будет отсортирован.

Ниже приведен псевдокод алгоритма (по возрастанию):

**Функция Выбором** (**тип данных** a (масссив), **цел** size)

{

**цел** i, j, m;

**тип данных** с;

**цикл для** i от 0 до size-1

{

m = i;

c = a[ i ];

**цикл для** j от i+1 до size-1

**если** (a[ j ] < c), **то**

{

c = a[ j ];

m = j;

}

a[ m ] = a [ i ];

a[ i ] = c;

}

}

**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. Алгоритм будет работать до тех пор, пока не закончится массив. Выбор элемента из массива можно осуществлять произвольно, но удобнее брать их по порядку.

Ниже приведен псевдокод алгоритма (по возрастанию):

**Функция Вставками** (**тип данных** a (масссив), **цел** size)

{

**цел** i, j;

**тип данных** с;

**цикл для** i от 0 до size-1

{

c = a[ i ];

j = i-1;

**цикл пока** (j >= 0) **и** (c < a[ j ])

{

a[ j+1 ] = a[ j ];

j = j-1;

}

a[ j+1 ] = c;

}

}

**Сортировка слиянием:**

Базируется на принципе «разделяй и властвуй». Массив делится пополам, и каждая его часть снова делится пополам до тех пор, пока он не будет состоять из 1 элемента (считается упорядоченным). Затем происходит слияние каждой упорядоченной части со своей половиной, сохраняя упорядоченность.

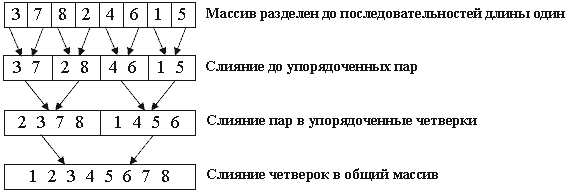


Рис. 11. Пример работы алгоритма на массиве 3 7 8 2 4 6 1 5...

Ниже приведен псевдокод алгоритма:

**Функция Cлиянием** (**тип данных** a (масссив), **цел** l, **цел** r)

{

**цел** mid;

**если** (l < r), **то**

{

mid = (l+r)/2;

Слиянием(a,l,mid);

Слиянием(a,mid+1,r);

Слияние (a,l,mid,r);

}

}

**Функция Cлияние** (**тип данных** a (масссив), **цел** l, **цел** mid, **цел** r)

{

**цел** pos1 = l;

**цел** pos2 = mid+1;

**цел** pos3 = 0;

**тип данных** buf (массив величины r - l+1);

**цикл пока** (pos1 <= mid) **и** (pos2 <= r)

{

**если** (a[ pos1 ] < a[ pos2 ]), **то**

{

buf[ pos3 ] = a[ pos1 ];

pos3 = pos3+1;

pos1 = pos1+1;

}

**иначе**

{

buf [pos3 ] = a[ pos2 ];

pos3 = pos3+1;

pos2 = pos2+1;

}

}

**цикл пока** (pos2 <= r)

{

buf[ pos3 ] = a[ pos2 ];

pos3 = pos3+1;

pos2 = pos2+1;

}

**цикл пока** (pos1 <= mid)

{

buf[ pos3 ] = a[ pos1 ];

pos3 = pos3+1;

pos1 = pos1+1;

}

**цикл для** pos3 от 0 до r-2

a[ pos3 + l ] = buf [ pos3 ];

}

**Сортировка Хоара:**

Метод основан на подходе "разделяй-и-властвуй". Общая схема такова:

1. из массива выбирается некоторый опорный элемент a[i],
2. запускается разделение массива, которое перемещает все ключи, меньшие, либо равные a[i], влево от него, а все ключи, большие, либо равные a[i] - вправо,
3. теперь массив состоит из двух подмножеств, причем левое меньше, либо равно правому
4. для обоих подмассивов, если в подмассиве более двух элементов, рекурсивно запускаем для него ту же процедуру.

В конце получится полностью отсортированная последовательность.

Ниже приведен псевдокод алгоритма:

**Функция Хоара** (**тип данных** a (масссив), **цел** l, **цел** size)

{

**цел** i, j;

**тип данных** buf, mid;

i = l;

j=size-1;

mid = a[ size/2 ];

**цикл до**

{

**цикл пока** (a[ i ] > mid)

i = i+1;

**цикл пока** (a[ j ] < mid)

j = j-1;

**если** (i <= j), **то**

{

buf = a[ i ];

a[ i ] = a[ j ];

a[ j ] = buf;

i = i+1;

j = j-1;

}

}

**пока** (i <= j)

**если** (j > 0), **то**

Хоара(a,l,j+1);

**если** (size > i), **то**

Хоара(a,i,size-i);

}

**Сортировка Шелла:**

Является модификацией сортировки простыми вставками. Идея метода Шелла состоит в сравнении элементов, стоящих не только рядом, но и на определённом расстоянии друг от друга. При данной сортировке сначала сравниваются и сортируются между собой значения, отстоящие один от другого на некотором расстоянии. После этого процедура повторяется для некоторых меньших расстояний, и так продолжается, пока расстояние не будет равно нулю (к этому моменту массив буде уже отсортирован). Единственной характеристикой сортировки Шелла является приращение - расстояние между сортируемыми элементами, в зависимости от прохода. В данной реализации приращение вычисляется по формуле Седжвика.

http://algolist.manual.ru/sort/gif/14_1.gif

Рис. 12. Приращение по формуле Седжвика

Ниже приведен псевдокод алгоритма:

**Функция Шелла** (**тип данных** a (масссив), **цел** size)

{

**цел** i, j, inc, s;

**цел** seq (массив на 40 элементов);

s = приращение (seq, size);

**тип данных** buf;

**цикл пока** (s >= 0)

{

inc = seq[ s ];

s = s-1;

**цикл для** i от inc до size-1

{

buf = a[ i ];

j = j-inc;

**цикл пока** (j >= 0) **и** (buf > a[ j ])

{

a[ j+inc ] = a[ j ];

j = j-inc;

}

a [ j+inc ] = buf;

}

}

}

# Результаты экспериментов

Для определения наиболее быстрой сортировки массивов был проведён эксперимент, в котором измерялось время, за которое каждая сортировка упорядочивает файлы по размеру. Сортировки проводились на разном числе файлов от 204 до 2974. Для каждой сортировки на определенном числе файлов было посчитано среднее значение.

В ходе эксперимента были получены следующие данные по времени сортировки:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Количество элементов в массиве** | **Сортировка Пузырьком** (в сек.) | **Сортировка Выбором** (в сек.) | **Сортировка Вставками** (в сек.) | **Сортировка Слиянием** (в сек.) | **Сортировка Хоара** (в сек.) | **Сортировка Шелла** (в сек.) |
| 204 | 0,0026 | 0,0018 | 0,001 | 0,0006 | 0,0004 | 0,0001 |
| 381 | 0,008 | 0,0063 | 0,0035 | 0,0008 | 0,00057 | 0,00057 |
| 600 | 0,02 | 0,015 | 0,0088 | 0,0014 | 0,0008 | 0,0008 |
| 838 | 0,038 | 0,0288 | 0,015 | 0,0022 | 0,0014 | 0,001 |
| 1527 | 0,1292 | 0,0954 | 0,0518 | 0,0034 | 0,0028 | 0,0026 |
| 2669 | 0,4098 | 0,2972 | 0,196 | 0,0064 | 0,0044 | 0,0046 |
| 2974 | 0,507 | 0,3662 | 0,2148 | 0,0072 | 0,005 | 0,0058 |

На основе полученных данных были построены графики, отображающие зависимость времени сортировки от количества элементов массива:

Так как сортировки слиянием, Хоара, Шелла работают быстрее, чем остальные три, был построен еще один график, чтобы наглядно убедиться какая из сортировок быстрее.

# Заключение

Анализируя полученные результаты, мы приходим к выводу, что сортировка Хоара является самой быстрой из рассматриваемых сортировок. Но стоит отметить, что данная сортировка является неустойчивой и при определенных входных данных может работать порядка n^2. Самой неэффективной по времени является сортировка «пузырек». Быстрее работают сортировка выбором и вставками. Сортировка слиянием показывает хороший результат, немного уступает Хоара и Шелла, но в отличии от них она устойчивая и дает на любых данных порядка nlogn. У слияния в свою очередь есть недостаток, нужно иметь дополнительный массив, что делает его менее выгодным по памяти. Сортировка Шелла работает чуть медленнее Хоара, но в отличие от последней ей не требуется памяти под стек, а также худшее гарантированное время работы сортировки Шелла выше, чем для быстрой сортировки.

Порядок сортировок по их быстроте:

1. Хоара
2. Шелла
3. Слиянием
4. Вставками
5. Выбором
6. Пузырек

В итоге, для файлового менеджера лучше подойдет сортировка Шелла, ибо более устойчива, чем Хоара и требует меньше памяти, что важно для хорошей работы файлового сортировщика.

.

# Литература

1. Кнут Д. Э. Искусство программирования. Том 3. Сортировка и поиск = The Art of Computer Programming. Volume 3. Sorting and Searching / под ред. В. Т. Тертышного (гл. 5) и И. В. Красикова (гл. 6). – 2-е изд. – Москва: Вильямс, 2007. – Т. 3. – 832 с.
2. Сайт Algolist. Сортировка выбором – <http://algolist.manual.ru/sort/select_sort.php>.
3. Сайт Algolist. Сортировка пузырьком – <http://algolist.manual.ru/sort/bubble_sort.php>.
4. Сайт Algolist. Сортировка вставками - <http://algolist.manual.ru/sort/insert_sort.php>
5. Сайт Algolist. Сортировка слиянием - <http://algolist.manual.ru/sort/merge_sort.php>
6. Сайт Algolist. Сортировка Хоара - <http://algolist.manual.ru/sort/shell_sort.php>
7. Сайт Algolist. Сортировка Шелла - <http://algolist.manual.ru/sort/shell_sort.php>

# Приложение

**Текст программы:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <locale.h>

#include <string.h>

#include <io.h>

struct datafail //структура для хранения имени и размера файла

{

unsigned long size;

char name[32];

};

//функция сравнения двух элементов

int compare(struct datafail a, struct datafail b, int k)

{

if (k == 1)

if (a.size > b.size)

return 1;

else

return 0;

if (k == 2)

if (a.size < b.size)

return 1;

else

return 0;

}

//функция сортировки пузырьком

void buble\_sort(struct datafail a[], long size, int k)

{

long i,j;

struct datafail c;

for (i = 0; i < size; i++)

for (j = (size-1); j > i; j--)

if (compare(a[j],a[j-1],k) == 1)

{

c = a[j];

a[j] = a[j - 1];

a[j - 1] = c;

}

}

//функция сортировки выбором

void select\_sort(struct datafail a[], long size, int k)

{

long i, j;

struct datafail c;

long m;//min или max

for (i = 0; i < size; i++)

{

m = i;

c = a[i];

for (j = (i + 1); j < size; j++)

{

if (compare(a[j], c, k) == 1)

{

c = a[j];

m = j;

}

}

a[m] = a[i];

a[i] = c;

}

}

//функция сортировки вставками

void insert\_sort(struct datafail a[], long size, int k)

{

long i, j;

struct datafail c;

for (i = 0; i < size; i++)

{

c = a[i];

for (j = i - 1; j >= 0 && (compare(c, a[j], k) == 1); j--)

a[j + 1] = a[j];

a[j + 1] = c;

}

}

//функция организующая слияние, двух упорядоченных последовательностей

void merge(struct datafail a[], long l, long mid, long r, int k)

{

long pos1 = l;

long pos2 = mid + 1;

long pos3 = 0;

struct datafail \*buf = new struct datafail[r - l + 1];

while (pos1 <= mid && pos2 <= r)

{

if (compare(a[pos1], a[pos2], k) == 1)

buf[pos3++] = a[pos1++];

else

buf[pos3++] = a[pos2++];

}

while (pos2 <= r)

buf[pos3++] = a[pos2++];

while (pos1 <= mid)

buf[pos3++] = a[pos1++];

for (pos3 = 0; pos3 < r - l + 1; pos3++)

a[l + pos3] = buf[pos3];

}

//функция сортировки слиянием

void merge\_sort(struct datafail a[], long l, long r, int k)

{

long mid;

if (l < r)

{

mid = (l + r) / 2;

merge\_sort(a, l, mid, k);

merge\_sort(a, mid + 1, r, k);

merge(a, l, mid, r, k);

}

}

//функция сортировки Хоара

void quick\_sort(struct datafail \*a, long size, int k)

{

long i = 0, j = size - 1;

struct datafail buf, mid;

mid = a[ size>>1 ];

do

{

while (compare(a[i],mid,k) == 1 )

i++;

while (compare(mid, a[j],k) == 1)

j--;

if (i <= j)

{

buf = a[i];

a[i] = a[j];

a[j] = buf;

i++;

j--;

}

}

while (i <= j);

if (j > 0)

quick\_sort(a, j+1,k);

if (size > i)

quick\_sort(a + i, size - i,k);

}

//функция считающее приращение для сортировки Шелла

int increment(long inc[], long size) {

int p1, p2, p3, s;

p1 = p2 = p3 = 1;

s = -1;

do {

if (++s % 2) {

inc[s] = 8 \* p1 - 6 \* p2 + 1;

}

else {

inc[s] = 9 \* p1 - 9 \* p3 + 1;

p2 \*= 2;

p3 \*= 2;

}

p1 \*= 2;

} while (3 \* inc[s] < size);

return s > 0 ? --s : 0;

}

//функция сортировки Шелла

void shell\_sort(struct datafail a[], long size, int k)

{

long inc, i, j, seq[40];

int s;

s = increment(seq, size);

while (s >= 0) {

inc = seq[s--];

for (i = inc; i < size; i++) {

struct datafail buf = a[i];

for (j = i - inc; (j >= 0) && (compare(buf,a[j],k) == 1); j -= inc)

a[j + inc] = a[j];

a[j + inc] = buf;

}

}

}

int main()

{

setlocale(LC\_ALL, "Rus");

struct \_finddata\_t c\_file;

intptr\_t hFile;

char path[260];

struct datafail a[10000];//массив с данными о файле

long i;

long N = 0;//Число файлов в каталоге

int flag = 0, flag1 = 0;

//order\_sort - по убыванию или возрастанию сортировать, number\_sort - какая сортировка

int order\_sort = 0, number\_sort = 0;

//переменные для подсчета времени

clock\_t t1, t2;

double tt = 0;

printf(" Файловый менеджер \"Я у мамы программист\"\n");

printf(" Меню :\n");

while (1 > 0)

{

if (flag1 == 0)

{ printf("Введите путь до директории в таком формате с:\\..\\\n");

gets\_s(path, 260);

strcat(path, "\*");//добавляем \* в конец пути

}

if ((hFile = \_findfirst(path, &c\_file)) == -1L)

printf("Нет файлов в этой директории!\n");

else

{

//считываем данные о файлах из директории

N = 0;

printf("FILE SIZE\n", ' ');

printf("---- ----\n", ' ');

do {

strncpy(a[N].name, c\_file.name, 32);

a[N].size = c\_file.size;

printf("%-12.12s %10u\n", a[N].name, a[N].size);

N++;

}

while (\_findnext(hFile, &c\_file) == 0);

\_findclose(hFile);

printf("Число файлов: %d\n\n", N);

//пользователь выбирает сортировку

printf("Выбирите сортировку:\n");

printf(" 1 - Пузырек\n");

printf(" 2 - Выбором\n");

printf(" 3 - Вставками\n");

printf(" 4 - Слиянием\n");

printf(" 5 - Хоара, более известная как быстрая сортировка\n");

printf(" 6 - Шелла\n");

flag = 0;

//проверка правильности ввода

do

{

if (flag == 1)

printf("Вы неправильно ввели номер, повторите попытку\n");

else

flag = 1;

scanf\_s("%d", &number\_sort);

}

while (number\_sort < 1 || number\_sort > 6);

printf("Как хотите отсортировать?\n1 - по убыванию, 2 -по возрастанию\n");

flag = 0;

//проверка правильности ввода

do

{

if (flag == 1)

printf("Вы неправильно ввели номер, повторите попытку\n");

else

flag = 1;

scanf\_s("%d", &order\_sort);

}

while (order\_sort < 1 || order\_sort > 2);

switch (number\_sort)

{

case 1: t1 = clock();

buble\_sort(a, N, order\_sort);

t2 = clock();

break;

case 2: t1 = clock();

select\_sort(a, N, order\_sort);

t2 = clock();

break;

case 3: t1 = clock();

insert\_sort(a, N, order\_sort);

t2 = clock();

break;

case 4: t1 = clock();

merge\_sort(a, 0, N - 1, order\_sort);

t2 = clock();

break;

case 5: t1 = clock();

quick\_sort(a, N, order\_sort);

t2 = clock();

break;

case 6: t1 = clock();

shell\_sort(a, N, order\_sort);

t2 = clock();

break;

}

//подсчет времени

tt = double(t2 - t1) / CLOCKS\_PER\_SEC;

//вывод результата

printf("FILE SIZE\n", ' ');

printf("---- ----\n", ' ');

for (i = 0; i < N; i++)

printf("%-12.12s %10u\n", a[i].name, a[i].size);

printf("\nВремя работы сортировки :%f\n\n", tt);

printf("Хотите продолжить работу?\n Введите 0, если хотите закончить работу\n Любое другое, чтобы продолжить\n");

scanf\_s("%d", &flag);

getchar();

if (flag == 0)

break;

printf("Хотите отсортировать файлы из этого директория с помощью другой сортировки?\n 0 - нет, 1 - да\n");

flag = 0;

do

{

if (flag == 1)

printf("Вы неправильно ввели номер, повторите попытку\n");

else

flag = 1;

scanf\_s("%d", &flag1);

}

while (flag1 < 0 || flag1 > 1);

getchar();

}

}

scanf\_s("%d", &N);

}