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**

**«Сортировка поразрядная»**

**Выполнил**:

студент группы 382003-1

Филатов Андрей Александрович

**Проверил**:

ассистент каф. МОСТ,

Волокитин В.Д.

Нижний Новгород

2020

**Содержание**

[Постановка задачи 3](#_Toc26962562)

[Метод решения 4](#_Toc26962563)

[Руководство пользователя 5](#_Toc26962564)

[Описание программной реализации 6](#_Toc26962565)

[Подтверждение корректности 7](#_Toc26962566)

[Результаты экспериментов 8](#_Toc26962567)

[Заключение 9](#_Toc26962568)

[Приложение 10](#_Toc26962569)

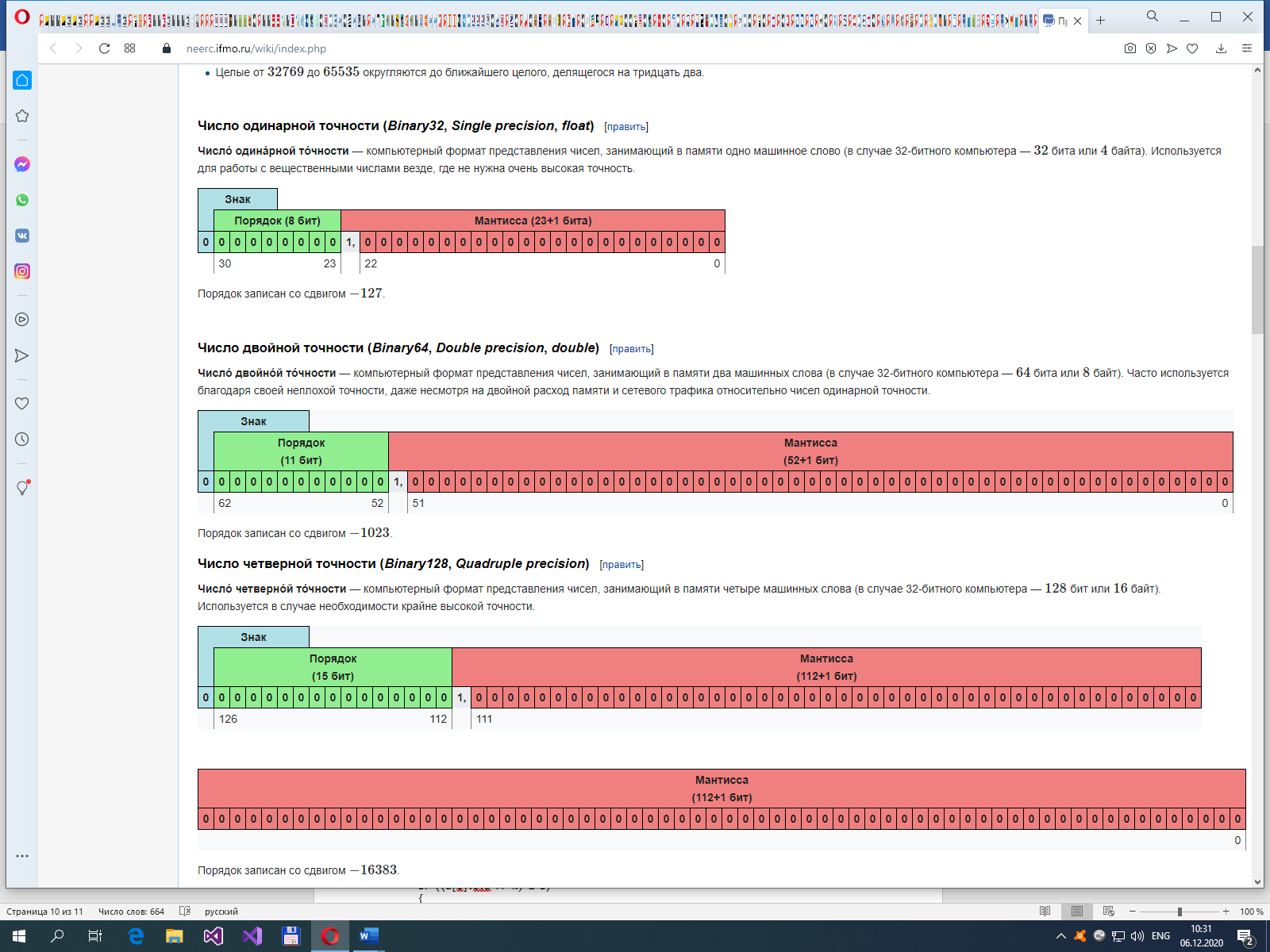
# Постановка задачи

Написать на языке С поразрядную сортировку (Radix *sort*) для чисел double.

# Метод решения

**Число двойной точности –** это компьютерный формат представления чисел, занимающий в памяти два машинных слова (в случае 32-битного компьютера — 64 бита или 8 байт).

 В наиболее распространённом формате (стандарт IEEE 754) число с плавающей запятой представляется в виде набора битов, часть из которых кодирует собой мантиссу числа, другая часть — показатель степени, и ещё один бит используется для указания знака числа (0 — если число положительное, 1 — если число отрицательное). При этом порядок записывается как целое число, а мантисса- своей дробной частью в двоичной системе счисления.



 Запишем код числа «−312,3125» в машинном представлении

1. Двоичная запись модуля этого числа имеет вид 100111000,0101
2. Имеем 100111000,0101 == 1,001110000101×28.
3. Получаем смещенный порядок 8 + 1023 == 1031. Далее имеем 1031(10) == 100000001111(2).
4. Окончательно

|  |  |  |
| --- | --- | --- |
| 1 | 10000000111 | 0011100001010000000000000000000000000000000000000000 |
| 63 | 62..52 | 51..0 |

Данное представление будем использовать для поразрядной сортировки. Сам алгоритм поразрядной сортировки состоит в последовательной сортировке по каждому разряду, в порядке от старшего разряда к младшему, после чего последовательности будут расположены в требуемом порядке.

Посчитаем сложность данного алгоритма. Пусть список для сортировки содержит n w-значных чисел, и каждая цифра может принимать значения 0 и 1.

Идём двумя указателями с двух сторон массива, меняя элементы, стоящие не на своём месте. Проход по массиву . Для w разрядов

Т.к. количество разрядов — константа, то сложность поразрядной сортировки составляет , то есть она линейно зависит от количества сортируемых чисел.

Поразрядная сортировка не требует дополнительной памяти для работы – дополнительная память .

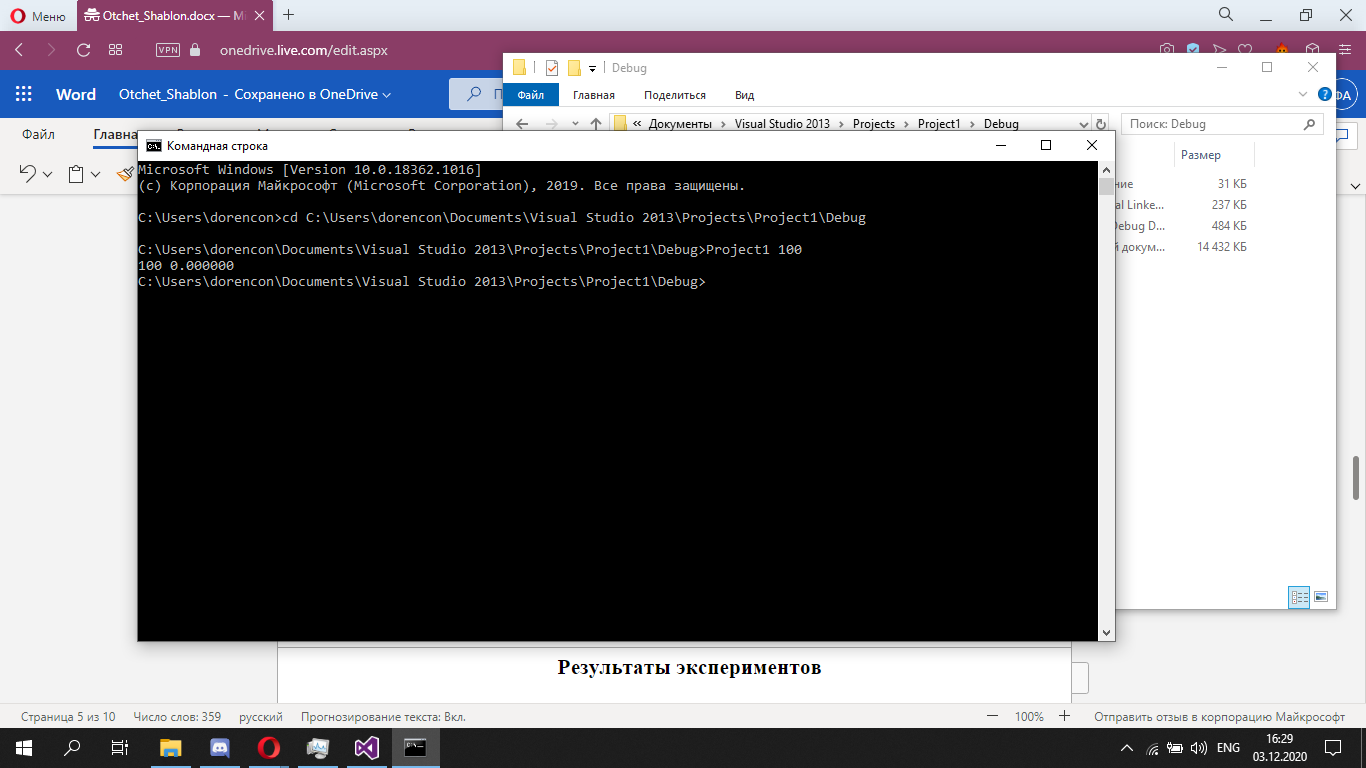
Количество обменов – в лучшем случае 0 в худшем .

# Руководство пользователя

Для использования программы пользователь должен иметь файл с именем test.txt, содержащий неотсортированный массив.

Для удобства тестирования, программа запускается из командной строки с одним аргументом. Аргумент – сколько последовательных чисел прочитать из файла.

Пример командной строки



Результат работы программы – отсортированный массив, будет выведен на экран.

# Описание программной реализации

Программа состоит из одного текстового (тестового) файла и исполняемого файла.

Будем использовать побитное представление числа (в стандарте IEEE 754 с двойной точностью) для поразрядной сортировки. Для доступа к битовому представлению числа будем использовать объединение (**union**), поскольку его члены ссылаются на общую память, поэтому **union** может быть использован для упаковки/распаковки данных. Это конкретное применение объединений включает в себя запись значения в один член объединения и чтение другого его члена.

# Подтверждение корректности

Докажем, что данный алгоритм работает верно, используя метод математической индукции по номеру разряда. Пусть w — количество разрядов в сортируемых объектах.

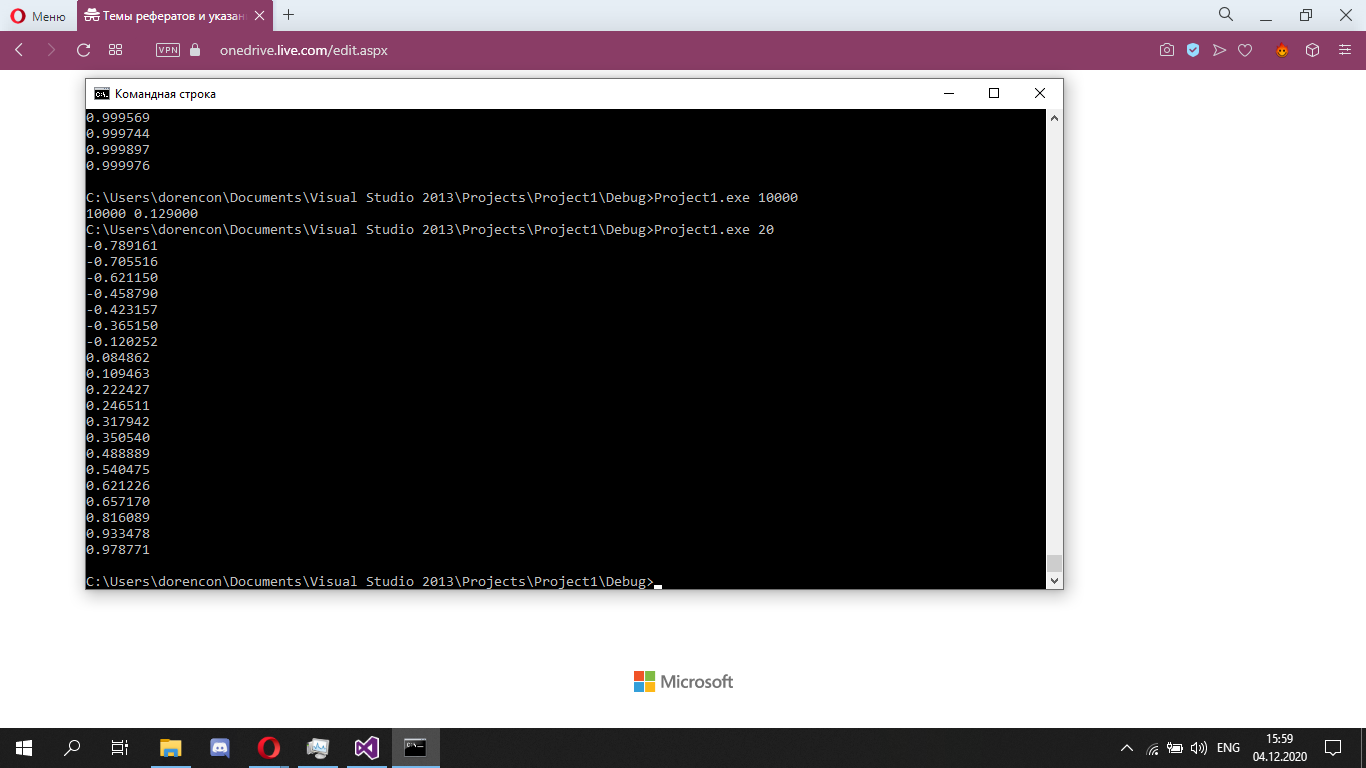
**База**: w=1. Очевидно, что алгоритм работает, верно, потому что в таком случае мы просто сортируем старшие разряды какой-то заранее выбранной устойчивой сортировкой.

**Переход**: пусть для w=k алгоритм правильно отсортировал последовательности по k старшим разрядам.

В таком случае массив будет разделён на групп. В первой группе находится элементы, которые можно представить в двоичном виде, как 64 единицы, а в последней, как 0 и 63 единицы.

Таким образом доказано, что данный алгоритм работает корректно для чисел с любым количеством разрядов.

Для подтверждения корректности в программе были произведены тесты с различным количеством элементов. Пример работы программы на 20 элементах.



# Результаты экспериментов

По данным экспериментов видно, что поразрядная сортировка — линейный алгоритм, время его работы пропорционально размеру сортируемого списка.

Тесты проводились на ноутбуке Aser Intel Core i5-7200U CPU 2.50 GHz

# Заключение

Изначально поразрядная сортировка предназначалась для сортировки целых чисел, записанных цифрами. Но так как в памяти компьютеров любая информация записывается целыми числами, алгоритм пригоден для сортировки любых объектов, запись которых можно поделить на «разряды», содержащие сравнимые значения. Например, так сортировать можно не только числа, записанные в виде набора цифр, но и строки, являющиеся набором символов, и вообще произвольные значения в памяти, представленные в виде набора байт.

# Приложение

#if defined(\_WIN32) || defined(\_\_WIN32\_\_) || defined(WIN32)

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#endif

#include <stdio.h>

#include <conio.h>

#include <time.h>

struct bytes8

{

long long a : 64;

};

typedef struct bytes8 bytes;

union doublebit

{

bytes b;

double d;

};

typedef union doublebit doublebit;

void radix(doublebit \*a, int l, int r, int k)

{

if ((l >= r) || (k < 0))

return;

int i = l, j = r;

while (i < j)

{

if (!((a[i].b.a >> k) & 1))

{

i++;

continue;

}

if ((a[j].b.a >> k) & 1)

{

j--;

continue;

}

long long temp = a[i].b.a;

a[i].b.a = a[j].b.a;

a[j].b.a = temp;

}

if (!((a[i].b.a >> k) & 1))

{

i++;

}

radix(a, l, i - 1, k - 1);

radix(a, i, r, k - 1);

}

void radixmn(doublebit \*a, int l, int r, int k)

{

if ((l >= r) || (k < 0))

return;

int i = l, j = r;

while (i < j)

{

if ((a[i].b.a >> k) & 1)

{

i++;

continue;

}

if (!((a[j].b.a >> k) & 1))

{

j--;

continue;

}

long long temp = a[i].b.a;

a[i].b.a = a[j].b.a;

a[j].b.a = temp;

}

if ((a[i].b.a >> k) & 1)

{

i++;

}

radixmn(a, l, i - 1, k - 1);

radixmn(a, i, r, k - 1);

}

int main(int argc, char \*\*argv)

{

int n = atoi(argv[1]);

FILE \*f;

f = fopen("test.txt", "r");

doublebit\* a;

a = malloc(n \* sizeof(double));

for (int i = 0; i < n; i++)

{

fscanf(f, "%lf", &a[i]);

}

fclose(f);

int i = 0, j = n - 1;

int t0 = clock();

while (i < j)

{

if ((a[i].b.a >> 63) & 1)

{

i++;

continue;

}

if (!((a[j].b.a >> 63) & 1))

{

j--;

continue;

}

long long temp = a[i].b.a;

a[i].b.a = a[j].b.a;

a[j].b.a = temp;

}

if ((a[i].b.a >> 63) & 1)

{

i++;

}

radixmn(a, 0, i - 1, 62);

radix(a, i, n - 1, 62);

int t1 = clock();

/\*for (int i = 0; i < n; i++)

{

printf("%lf ", a[i].d);

}\*/

printf("%d %lf", n, (double)(t1 - t0) / CLOCKS\_PER\_SEC);

return 0;

}