Министерство науки и высшего образования РФ

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

студенты группы 22ВВП1

Воробьева М.М.

Расторгуева К.В.

Приняли:

Акифьев И.В.

Митрохин М.А.

Пенза 2023

**Название**

Оценка времени выполнения программ

**Цель работы**

Оценить время выполнения программ языка Си или их частей.

**Лабораторное задание**

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

**Задание 2**:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.
5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Пояснительный текст к программе**

**Задание 1:**

В программе объявляются 3 двумерных массива: *a*, *b*, *c*.

int\*\* a; //указатель на двумерный массив а

int\*\* b; //указатель на двумерный массив b

int\*\* c; //указатель на двумерный массив c

Для создания двумерных массивов используется функция динамического выделения памяти *malloc()*. Сначала выделяется память по одномерный массив

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

Затем для каждой ячейки созданного массива выделяется одномерный массив

for (i = 0; i < n; i++){ //выделения памяти для каждой ячейки одномерного массива

a[i] = (int\*)malloc(n \* sizeof(int)); //для создания двумерного массива

}

Массивы *a* и *b* задаются генератором случайных чисел. Создаётся цикл, проходящий по всем ячейкам массива а и записывающий случайное значение.

srand(time(NULL));

while (i < n){

while (j < n){

a[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

Аналогично массив b

i = 0; j = 0;

while (i < n)

{

while (j < n)

{

b[i][j] = rand() % 100 + 1; // заполняем массив b случайными числами

j++;

}

i++;

}

Затем засекается время начала цикла, в котором происходит перемножение массивов *a* и *b*:

clock\_t start = clock();

Умножаем элементы r-ой строки первой матрицы на элементы r-ой столбца второй матрицы, получившееся произведение складываем. Результат умножения сохраняется в массив *c*.

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

{

for (j = 0; j < n; j++)

{

elem\_c = 0;

for (r = 0; r < n; r++)

{

elem\_c = elem\_c + a[i][r] \* b[r][j];

c[i][j] = elem\_c;

}

}

}

После формирования результата засекается время окончания:

clock\_t end = clock();

На экран выводится время умножения массивов *a* и *b*:

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

Перемножение различной размерность происходи в цикле while шаг которого регулируется выражением n = n + 1000;

Назначение отдельных операторов программы указывается в виде комментариев. Программа завершает свою работу после нажатия на любую клавишу на клавиатуре.

**Задание 2**:

В программе использованы 3 вида сортировок: сортировка Шелла(*shell)*, быстрая сортировка (*qs*) и стандартная функция *qsort*, реализующая алгоритм быстрой сортировки. Массив *mas* заполняется несколько раз: 1 раз -случайными числами, 2 раз - возрастающей последовательностью чисел, 3 раз - убывающей последовательностью чисел, 4 раз - одна половина – возрастающая последовательность чисел, а вторая, – убывающая. Для каждого случая вызывается каждый вид сортировки и засекается время работы алгоритма сортировки.

Рассмотрим алгоритм измерения времени сортировки на примере сортировки Шелла в случае заполнения массива случайными числами.

Массивы mas, mas\_save\_q, mas\_save\_qsort заполняются одинаковыми случайными числами для создания одинаковых условий сортировки.

srand(time(NULL)); //заполнение массивов

for (int i = 0; i < N; i++) { //mas, mas\_save\_q, mas\_save\_qsort

mas[i] = rand() % 100 - 20; //одинаковыми случайными числами

mas\_save\_q[i] = mas[i];

mas\_save\_qsort[i] = mas[i];

fprintf(Shell1, " %d ", mas[i]); //вывод данных в файл до сортировки

}

Засекается время начала сортировки. Вызывается функция сортировки Шелла shell().Засекается время окончания сортировки. Затем высчитывается время сортировки в секундах и записывается в переменную TSh1

start = clock();

shell(mas, N);

end = clock();

TSh2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

Аналогичный алгоритм используется для быстрой сортировки (*qs*) и стандартной функции *qsort*

start = clock();

qs(mas, 0, N - 1);

end = clock();

TQS2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(mas, N, sizeof(int), compare);

end = clock();

TQSort2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

Массив до сортировки и после записывается в файл. Время работы сортировки Шелла сохраняется в переменных *TSh2*, *TQS2*, *TQSort2* и используется в выводе итоговой таблицы.

Аналогичный подсчет времени производится для возрастающей, убывающей последовательности чисел, последовательности, где одна половина – возрастающая последовательность чисел, а вторая, – убывающая

**Результат выполнения программы**

Результат работы программы для задания 1 представлены на рис. 1-2.

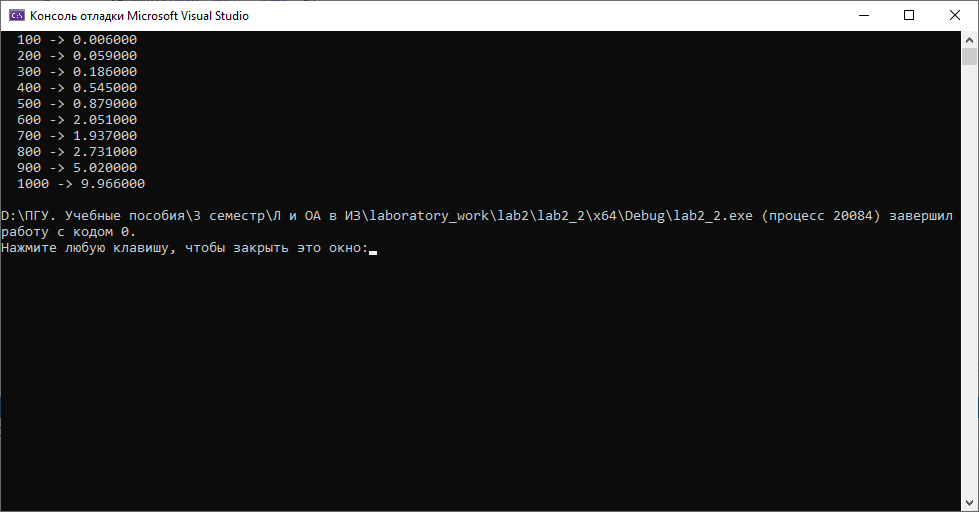


Рисунок 1

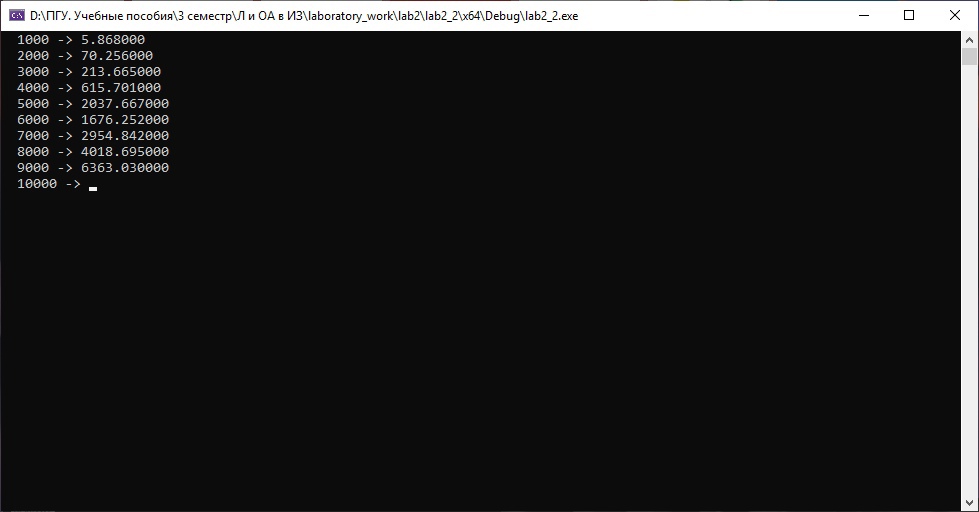


Рисунок 2

Результат работы программы для задания 2 представлены на рис. 3-5.

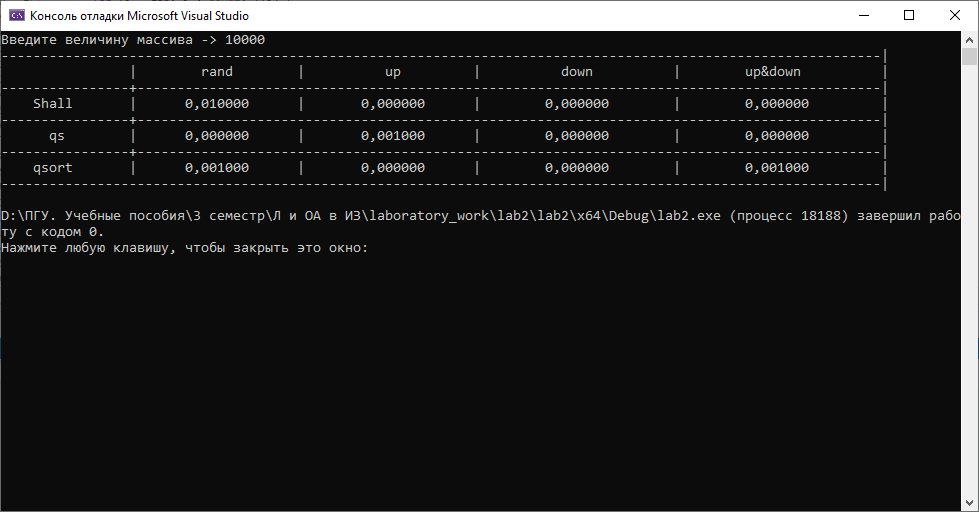


Рисунок 3

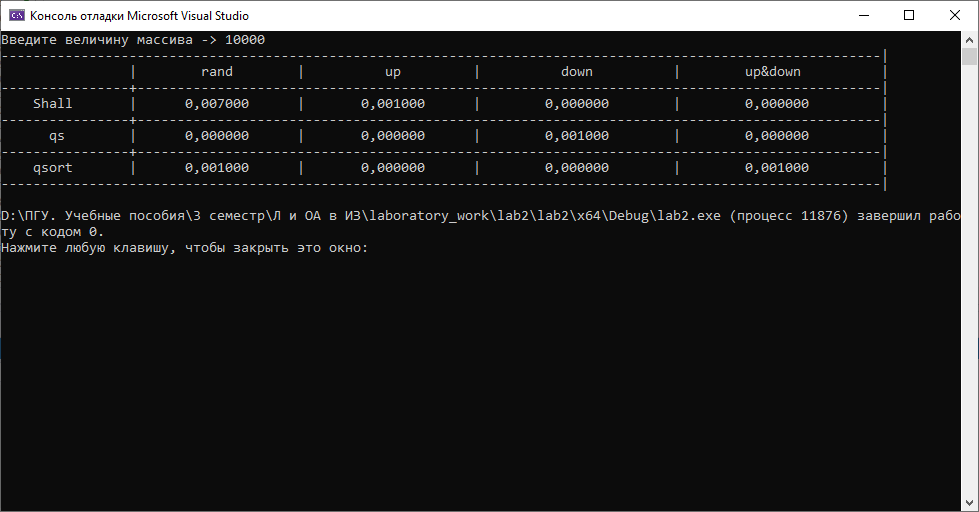


Рисунок 4

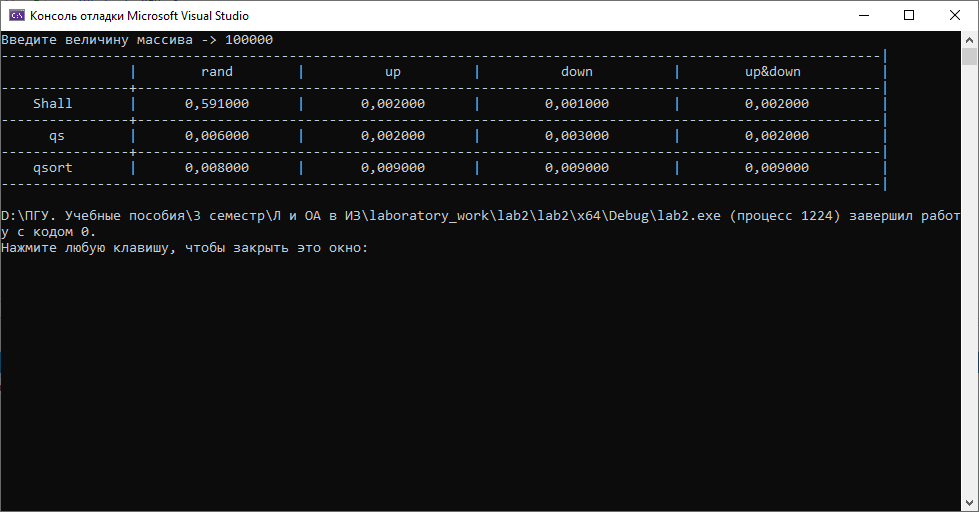
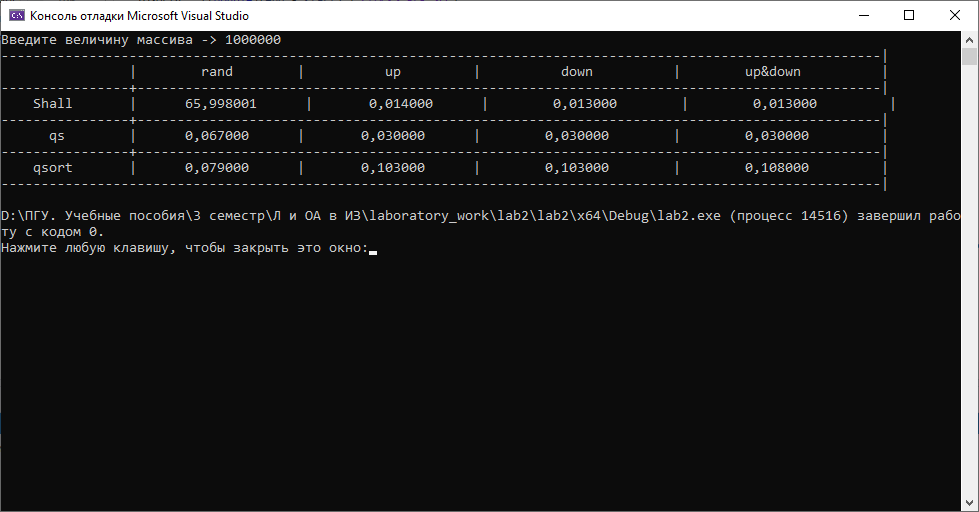
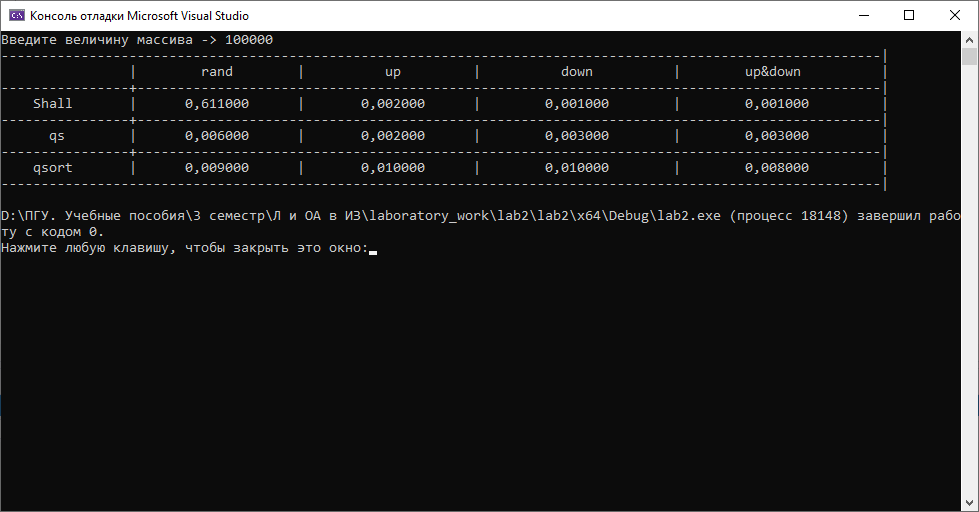


Рисунок 5



**Анализ результатов**

**Задание 1**:

Порядок сложности программы = O(n3)

График зависимости времени выполнения программы от размера матриц представлен на рис. 6.



Рисунок 6

График зависимости времени выполнения программы от размера матриц при постоянном коэффициенте k = 0,2 представлен на рис. 7.



X3

экспериментальные данные

k = 0,2

х3

экспериментальные данные

Рисунок 7

**Задание 2**:

Для сортировки массива, заданного случайными числами, самой быстрой является функция *qs*.

Для сортировки массива, в которомодна половина – возрастающая последовательность чисел, а вторая, – убывающая, самым быстрым является алгоритм сортировки Шелла.

Функция *qsort* работает медленнее, так как сортировка универсальная: работает для различных типов данных.

Таким образом, самый оптимальный алгоритм сортировки – быстрая сортировка *qs*.

**Вывод**

В ходе выполнения лабораторной работы была осуществлена оценка времени программ языка Си и их частей. Также было проведено сравнение времени работы 3 видов сортировки: сортировки Шелла(*shell)*, быстрой сортировки (*qs*) и стандартной функции *qsort*, реализующей алгоритм быстрой сортировки. Из всех видов сортировки самой эффективной является алгоритм быстрой сортировки (*qs*). Функция *qsort* работает медленнее, так как сортировка универсальная: работает для различных типов данных.

**Приложение**

**Программа**

**Задание 1:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

setvbuf(stdin, NULL, \_IONBF, 0);

setvbuf(stdout, NULL, \_IONBF, 0);

//clock\_t start, end; // объявляем переменные для определения времени выполнения

int i = 0, j = 0, r, elem\_c;

int n = 1000;

int\*\* a;

int\*\* b;

int\*\* c;

//int a[200][200], b[200][200], c[200][200], elem\_c;

while (n <= 10000) {

printf(" % d -> ", n);

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

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

{

a[i] = (int\*)malloc(n \* sizeof(int));

}

b = (int\*\*)malloc(n \* sizeof(int\*));

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

b[i] = (int\*)malloc(n \* sizeof(int));

}

c = (int\*\*)malloc(n \* sizeof(int\*));

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

{

c[i] = (int\*)malloc(n \* sizeof(int));

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

while (i < n)

{

while (j < n)

{

a[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

i = 0; j = 0;

while (i < n)

{

while (j < n)

{

b[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

clock\_t start = clock();

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

{

for (j = 0; j < n; j++)

{

elem\_c = 0;

for (r = 0; r < n; r++)

{

elem\_c = elem\_c + a[i][r] \* b[r][j];

c[i][j] = elem\_c;

}

}

}

clock\_t end = clock();

printf("%f\n", (double)(end - start) / CLOCKS\_PER\_SEC);

n = n + 1000;

}

return(0);

}

**Задание 2:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <locale.h>

#include <stdlib.h>

#include <time.h>

#include <iostream>

void shell(int\* items, int count){

int i, j, gap, k;

int x, a[5];

a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;

for (k = 0; k < 5; k++) {

gap = a[k];

for (i = gap; i < count; ++i) {

x = items[i];

for (j = i - gap; (x < items[j]) && (j >= 0); j = j - gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

void qs(int\* items, int left, int right){ //вызов функции: qs(items, 0, count-1);

int i, j;

int x, y;

i = left; j = right;

/\* выбор компаранда \*/

x = items[(left + right) / 2];

do {

while ((items[i] < x) && (i < right)) i++;

while ((x < items[j]) && (j > left)) j--;

if (i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++; j--;

}

} while (i <= j);

if (left < j) qs(items, left, j);

if (i < right) qs(items, i, right);

}

int compare(const void\* i, const void\* j){

return \*(int\*)i - \*(int\*)j;

}

int N;

int\* otvet;

float TSh1, TSh2, TSh3, TSh4;

float TQS1, TQS2, TQS3, TQS4;

float TQSort1, TQSort2, TQSort3, TQSort4;

FILE\* Shell1, \*Shell2, \*Shell3, \*Shell4;

int main() {

setlocale(0, "rus");

printf("Введите величину массива -> ");

scanf("%d", &N);

int\* mas = (int\*)malloc(N \* sizeof(int));

int\* mas\_save\_q = (int\*)malloc(N \* sizeof(int));

int\* mas\_save\_qsort = (int\*)malloc(N \* sizeof(int));

Shell1 = fopen("ShellRand.txt", "w"); //Случайные числа

fprintf(Shell1, "До сортировки\n");

srand(time(NULL));

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

mas[i] = rand() % 100 - 20;

mas\_save\_q[i] = mas[i];

mas\_save\_qsort[i] = mas[i];

fprintf(Shell1, " %d ", mas[i]);

}

clock\_t start = clock();

shell(mas, N);

clock\_t end = clock();

TSh1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(mas\_save\_q, 0, N-1);

end = clock();

TQS1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(mas\_save\_qsort, N, sizeof(int), compare);

end = clock();

TQSort1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

fprintf(Shell1, "\n\n\nПосле сортировки\n");

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

fprintf(Shell1, " %d ", mas[i]);

}

fclose(Shell1);

Shell2 = fopen("ShellUp.txt", "w"); //Возрастающая последовательность чисел

fprintf(Shell2, "До сортировки\n");

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

mas[i] = i;

fprintf(Shell2, " %d ", mas[i]);

}

start = clock();

shell(mas, N);

end = clock();

TSh2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(mas, 0, N - 1);

end = clock();

TQS2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(mas, N, sizeof(int), compare);

end = clock();

TQSort2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

fprintf(Shell2, "\n\n\nПосле сортировки\n");

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

fprintf(Shell2, " %d ", mas[i]);

}

fclose(Shell2);

Shell3 = fopen("ShellDown.txt", "w"); //Убывающая последовательность чисел

fprintf(Shell3, "До сортировки\n");

for (int i = N; i >= 0; i--) {

mas[i] = i;

mas\_save\_q[i] = mas[i];

mas\_save\_qsort[i] = mas[i];

fprintf(Shell3, " %d ", mas[i]);

}

start = clock();

shell(mas, N);

end = clock();

TSh3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(mas\_save\_q, 0, N - 1);

end = clock();

TQS3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(mas\_save\_qsort, N, sizeof(int), compare);

end = clock();

TQSort3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

fprintf(Shell3, "\n\n\nПосле сортировки\n");

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

fprintf(Shell3, " %d ", mas[i]);

}

fclose(Shell3);

Shell4 = fopen("ShellUpDown.txt", "w"); //Одна половина - возрастающая последовательность чисел, а вторая – убывающая

fprintf(Shell4, "До сортировки\n");

for (int i = 0; i < N/2; i++) {

mas[i] = i;

mas\_save\_q[i] = mas[i];

mas\_save\_qsort[i] = mas[i];

fprintf(Shell4, " %d ", mas[i]);

}

for (int i = N; i >= N/2; i--) {

mas[i] = i;

mas\_save\_q[i] = mas[i];

mas\_save\_qsort[i] = mas[i];

fprintf(Shell4, " %d ", mas[i]);

}

start = clock();

shell(mas, N);

end = clock();

TSh4 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(mas\_save\_q, 0, N - 1);

end = clock();

TQS4 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(mas\_save\_qsort, N, sizeof(int), compare);

end = clock();

TQSort4 = ((double)(end - start) / CLOCKS\_PER\_SEC);

fprintf(Shell4, "\n\n\nПосле сортировки\n");

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

fprintf(Shell4, " %d ", mas[i]);

}

fclose(Shell4);

printf("--------------------------------------------------------------------------------------------------------------|\n");

printf(" | rand | up | down | up&down |\n");

printf("----------------+---------------------------------------------------------------------------------------------|\n");

printf(" Shall | %f | %f | %f | %f |\n", TSh1, TSh2, TSh3, TSh4);

printf("----------------+---------------------------------------------------------------------------------------------|\n");

printf(" qs | %f | %f | %f | %f |\n", TQS1, TQS2, TQS3, TQS4);

printf("----------------+---------------------------------------------------------------------------------------------|\n");

printf(" qsort | %f | %f | %f | %f |\n", TQSort1, TQSort2, TQSort2, TQSort4);

printf("--------------------------------------------------------------------------------------------------------------|\n");

return 0;}