Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили

студент группы 21ВВ3:

Ефимушкин Александр

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Цель работы**

Изучение способа оценки сложности программы, а также способа измерения времени работы программы.

**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.

**Теоретический материал**

*Для оценки времени выполнения программ языка Си или их частей мы использовали средства, предоставляемые библиотекой time.h. Данная библиотека содержит описания типов и прототипы функций для работы с датой и временем.*

*Для описания скорости роста функций используется О-символика. О-символика отражает порядок сложности алгоритма.*

**Описание метода решения задачи**

**Задание 1**

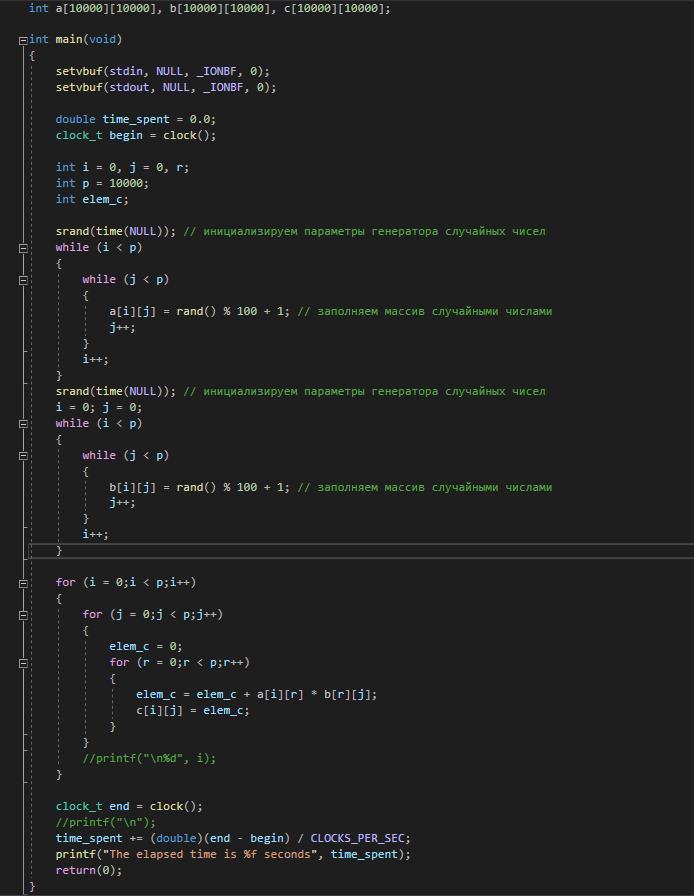
1. *Выводим объявление массивов для большего предоставления памяти*
2. *Добавляем в программу замер времени работы программы*
3. *Тестируем программу на различных значениях длины массива*

**Задание 2**

1. *Объявляем массива и заполняем его числами в зависимости от задания*
2. *Добавляем в программу замер времени работы программы*
3. *Тестируем программу на различных типах набора данных массива*

**Задание 1**

**Сложность**



O(n3)

O(n2)

O(n)

O(n)

O(n)

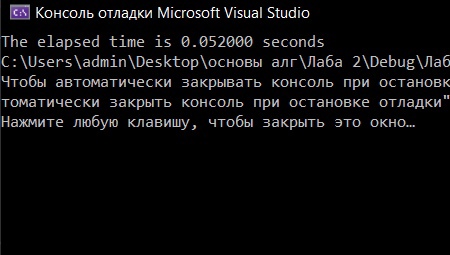
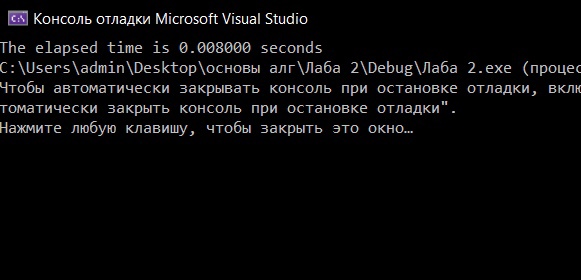
O(n2)

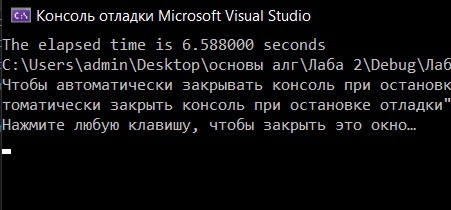
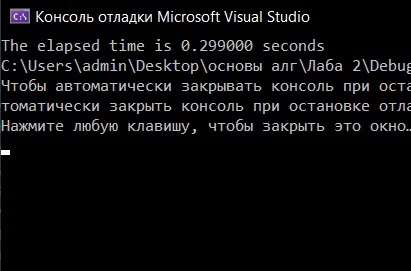
O(n2)

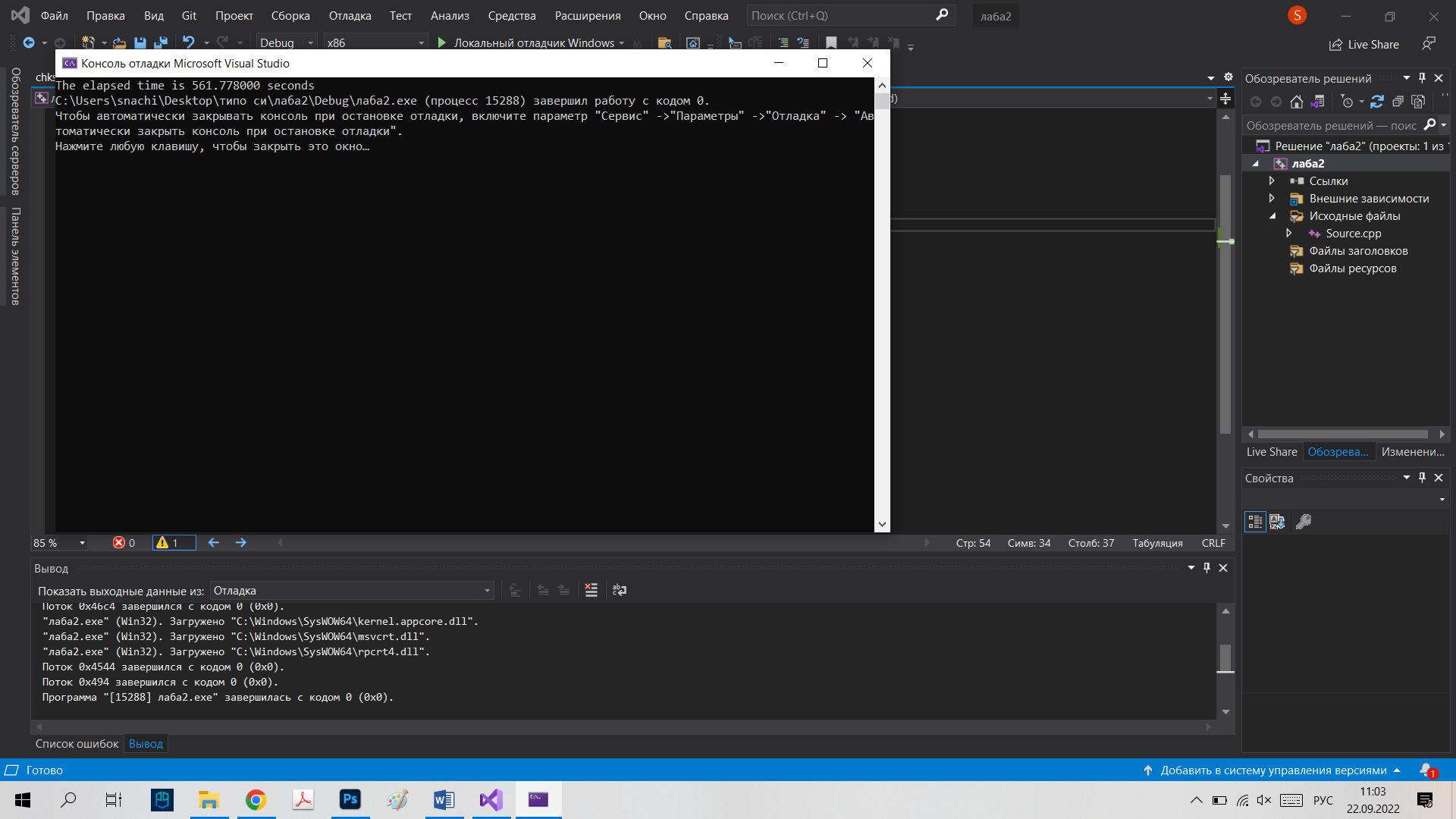
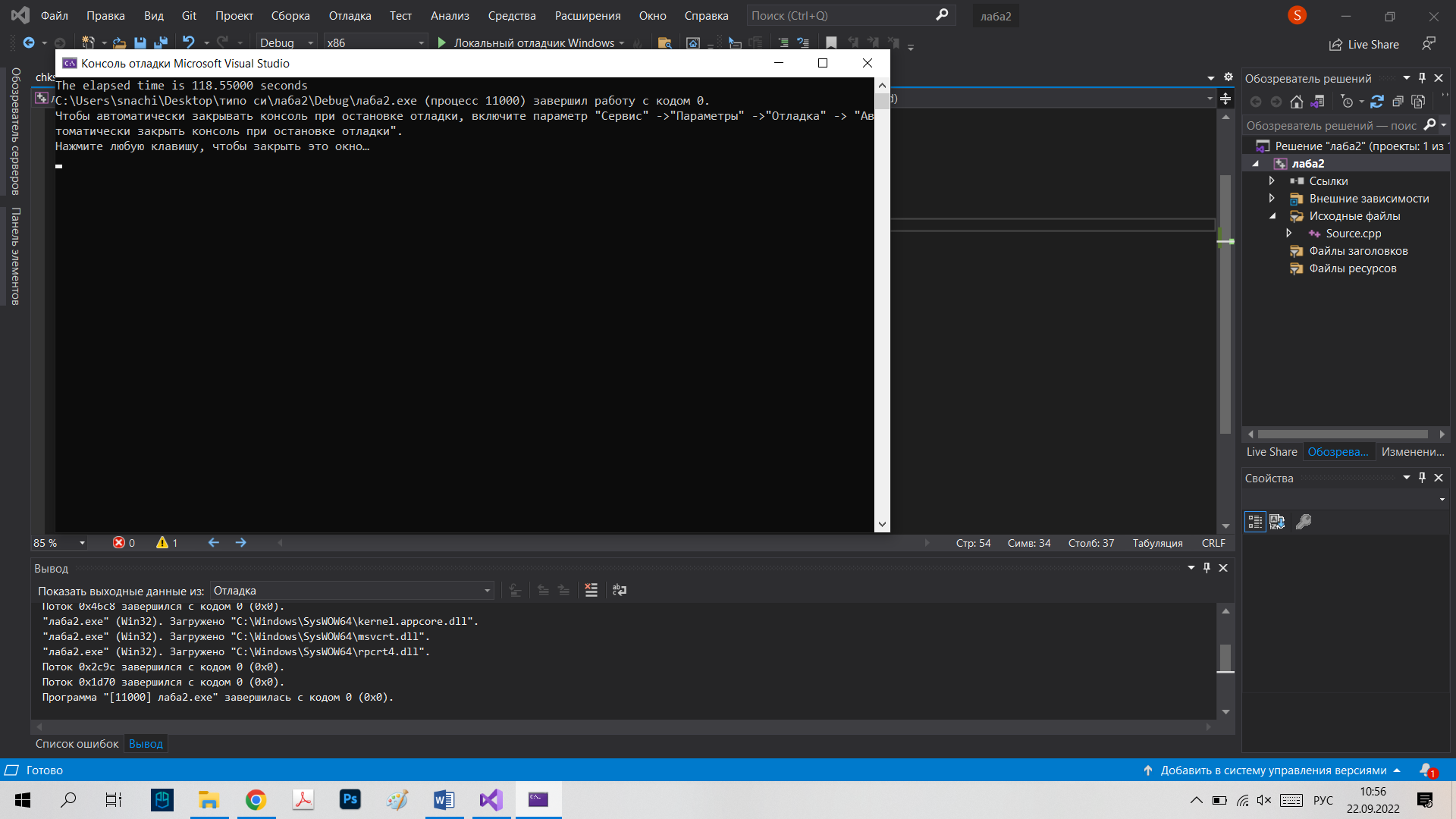
O(max(n3, n2, n2)) = O(n3)

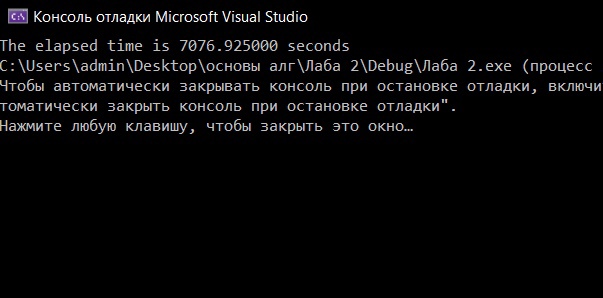
**Замеры**

**Результаты работы программы**









**Листинг**#include "stdio.h"

#include "stdlib.h"

#include "time.h"

int a[2000][2000], b[2000][2000], c[2000][2000], elem\_c, p = 2000;

int main(void)

{

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

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

double time\_spent = 0.0;

clock\_t begin = clock();

int i = 0, j = 0, r;

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

while (i< p)

{

while (j< p)

{

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< p)

{

while (j< p)

{

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

j++;

}

i++;

}

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

clock\_t end = clock();

time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

printf("The elapsed time is %f seconds", time\_spent);

return(0);

}

**Задание 2**

1. Проведем замеры времени работы в секундах алгоритмов сортировки на массиве длиной 123456 элементов

|  |  |  |  |
| --- | --- | --- | --- |
| Набор чисел в массиве | qsort | Алгоритм Шелла | Алгоритм быстрой сортировки |
| Случайный | 0.240 | 1.025 | 0.021 |
| Возрастает | 0.010 | 0.002 | 0.005 |
| Убывает | 0.028 | 2.101 | 0.004 |
| Возрастает и убывает | 0.105 | 1.038 | 0,005 |

**Листинг**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "stdio.h"

#include "stdlib.h"

#include "time.h"

#include <windows.h>

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)

{

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\* method(int p, int mod)

{

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

for (int k = 0; k < p; k++) { // заполняем массив случайными числами

if (mod == 0) a[k] = rand() % 100 + 1;

if (mod == 1) a[k] = k;

if (mod == 2) a[k] = p - k;

if (mod == 3){

if (k <= p / 2) {

a[k] = k;

}

else {

a[k] = p / 2 - k;

}

}

}

return a;

}

void print(int\* a, int p)

{

for (int k = 1; k <= p; k++) {

printf("%8.d", a[k - 1]);

if (k % 10 == 0) {

printf("\n");

}

}

}

int comp(const int\* i, const int\* j)

{

return \*i - \*j;

}

int main(void)

{

FILE\* fps;

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int\* item;

double time\_spent = 0.0;

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

int t[5], p = 100;

t[0] = 1000; t[1] = 2000; t[2] = 4000; t[3] = 10000; t[4] = 20000;

float u[4][7][3];

char name\_s[] = "C:/Users/snachi/Desktop/типо си/лаба2.2/u.txt";

if ((fps = fopen(name\_s, "r")) != NULL) {

fps = fopen(name\_s, "w");

for (int l = 0; l < 4; l++) {

for (int i = 0; i < 5; i++) {

item = method(t[i], l);

clock\_t begin = clock();

qsort(item, t[i], sizeof(int), (int(\*) (const void\*, const void\*)) comp);

//print(item, t[i]);

clock\_t end = clock();

time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

u[l][i][0] = time\_spent;

delete item;

}

for (int i = 0; i < 5; i++) {

p = t[i];

item = method(t[i], l);

clock\_t begin = clock();

shell(item, t[i]);

clock\_t end = clock();

time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

u[l][i][1] = time\_spent;

delete item;

}

for (int i = 0; i < 5; i++) {

p = t[i];

item = method(t[i], l);

clock\_t begin = clock();

qs(item, 0, t[i] - 1);

clock\_t end = clock();

time\_spent += (double)(end - begin) / CLOCKS\_PER\_SEC;

u[l][i][2] = time\_spent;

delete item;

}

}

fprintf(fps,"qsort:\n");

fprintf(fps," случайный порядок возростает убывает возростает и убывает:\n");

for (int l = 0; l < 5; l++) {

fprintf(fps,"%5.d -", t[l]);

fprintf(fps," %f %f %f %f\n", u[0][l][0], u[1][l][0], u[2][l][0], u[3][l][0]);

}

fprintf(fps,"\nсортировка Шелла:\n");

for (int l = 0; l < 5; l++) {

fprintf(fps,"%5.d -", t[l]);

fprintf(fps," %f %f %f %f\n", u[0][l][1], u[1][l][1], u[2][l][1], u[3][l][1]);

}

fprintf(fps,"\nбыстрая сортировка:\n");

for (int l = 0; l < 5; l++) {

fprintf(fps,"%5.d -", t[l]);

fprintf(fps," %f %f %f %f\n", u[0][l][2], u[1][l][2], u[2][l][2], u[3][l][2]);

}

}

}

**Вывод**

В ходе работы я изучили способ оценки сложности программы, а также способ измерения времени работы программы.