Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Л и ОА в ИЗ»

на тему «Оценка времени выполнения программы»

**Выполнили:**

студентом группы 19ВВ3

Земляков В.Д.

**Принял:**

Митрохин М.А.

Пенза 2020

**Цель работы:**

Оценка времени выполнения программы.

**Ход работы:**

**Задание 1.**

1. Вычислили порядок сложности программы.

O(n2) + O(n2) + O(n3) = O(n3)

1. Оценили время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200,400, 1000, 2000, 4000, 10000.

**100:**

Время работы= 0,002 c

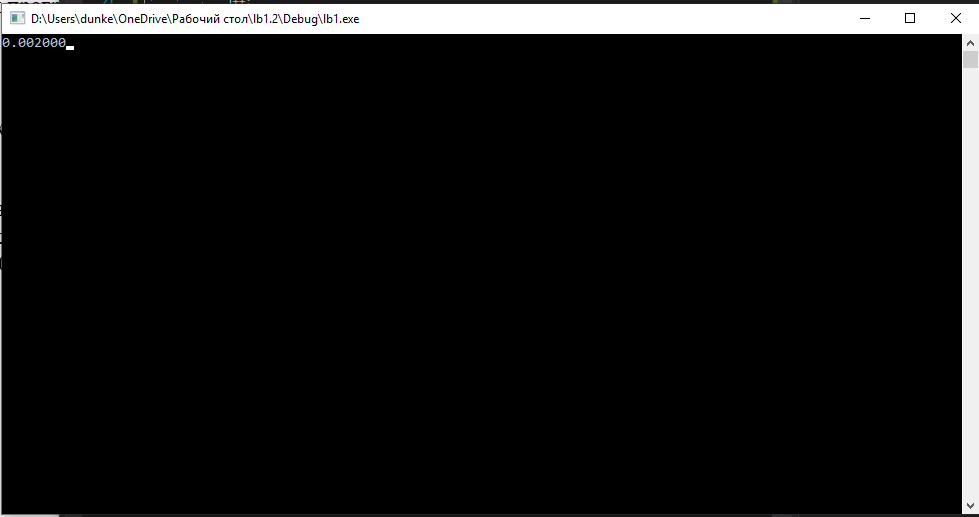


Рисунок 1. 100x100

**200:**

Время работы= 0,03 c

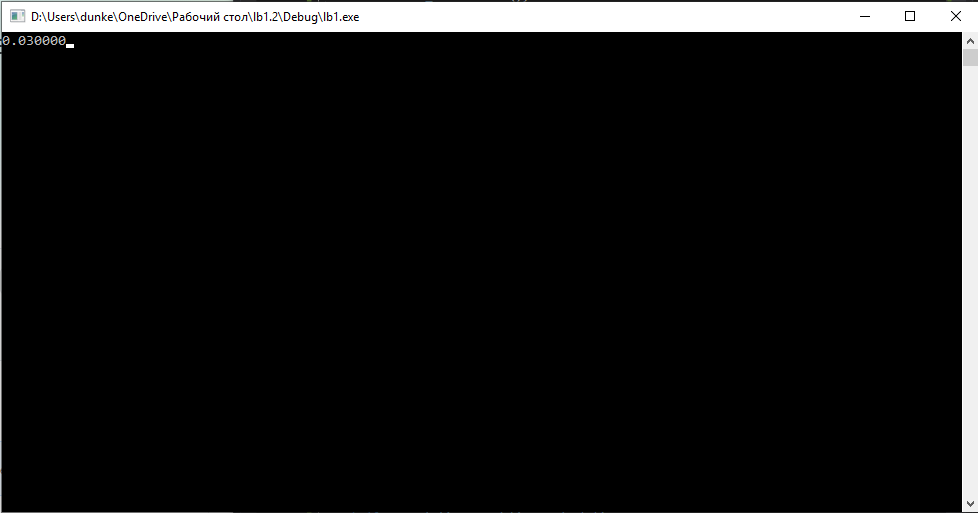


Рисунок 2.200x200

**400:**

Время работы= 0,268 c

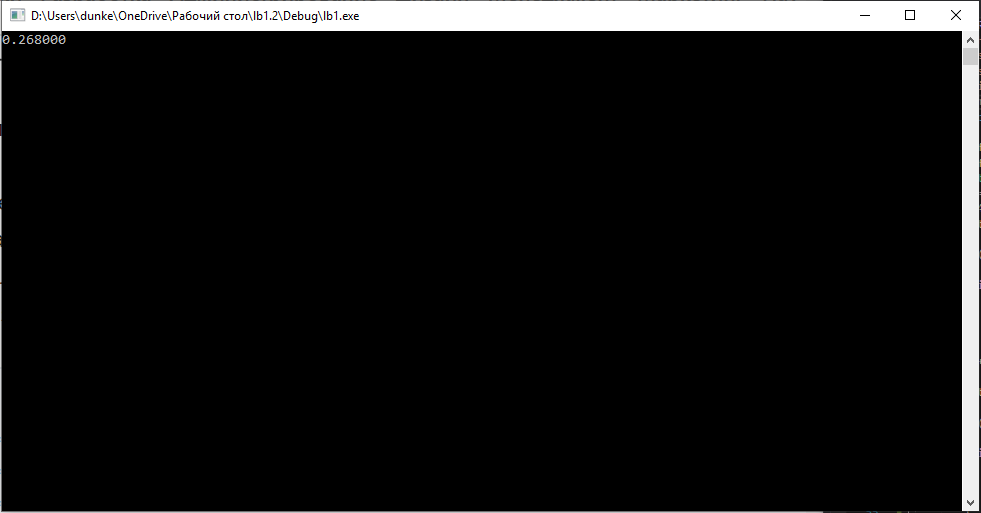


Рисунок 3.400x400

**1000:**

Время работы= 5,684 c

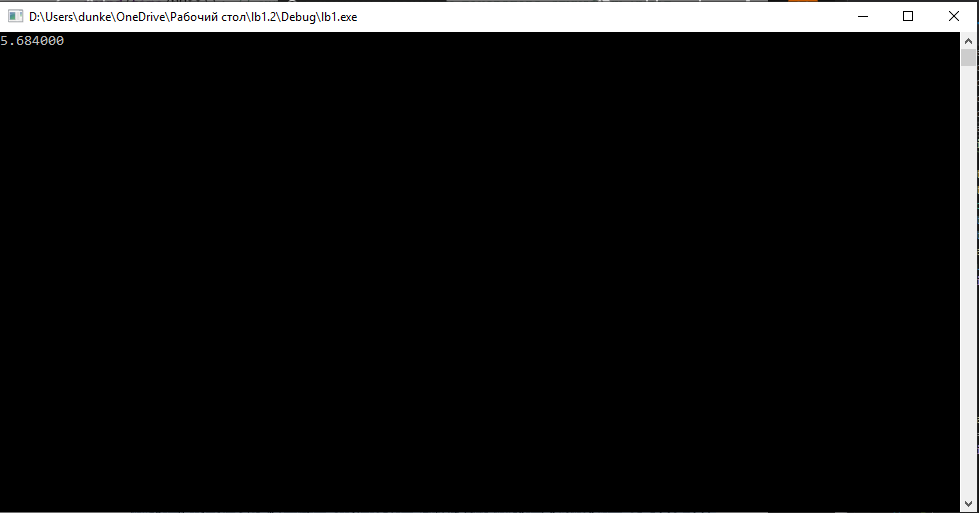


Рисунок 4.1000x1000

**2000:**

Время работы= 75,117 c

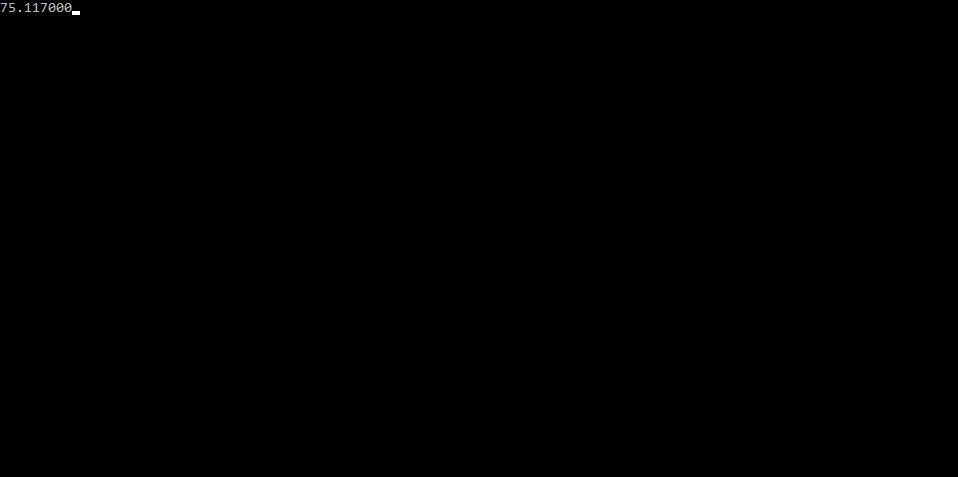


Рисунок 5.2000x2000

1. Построили график зависимости времени выполнения программы от размера матриц

и сравнили полученный результат с теоретической оценкой.

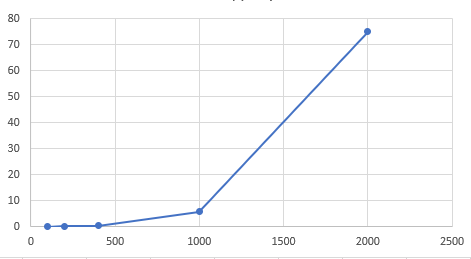


Рисунок 6. график зависимости

**Листинг (Задание 1):**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Header.h"

#include <stdio.h>

#include <stdlib.h>

#include<time.h>

#pragma comment(linker, "/STACK:100000000000000000")

int main(void)

{

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

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

int i = 0, j = 0, r;

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

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

while (i < 2000)

{

while (j < 2000)

{

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 < 2000)

{

while (j < 2000)

{

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

j++;

}

i++;

}

double start\_time = clock();

for (i = 0; i < 2000; i++)

{

for (j = 0; j < 2000; j++)

{

elem\_c = 0;

for (r = 0; r < 2000; r++)

{

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

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

}

}

}

double end\_time = clock();

double search\_time = end\_time - start\_time;

printf("%lf", search\_time/CLK\_TCK);

\_getch();

}

**Задание 2.**

**Результат работы программы:**

SHELL 10k

random: 0.008000

increase: 0.001000

decrease: 0.018000

saw: 0.009000

QS 10k

random: 0.004000

increase: 0.004000

decrease: 0.004000

saw: 0.006000

LQS 10k

random: 0.004000

increase: 0.003000

decrease: 0.004000

saw: 0.022000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SHELL 20k

random: 0.020000

increase: 0.001000

decrease: 0.089000

saw: 0.038000

QS 20k

random: 0.007000

increase: 0.013000

decrease: 0.009000

saw: 0.013000

LQS 20k

random: 0.011000

increase: 0.014000

decrease: 0.013000

saw: 0.042000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SHELL 50k

random: 0.266000

increase: 0.001000

decrease: 0.585000

saw: 0.279000

QS 50k

random: 0.019000

increase: 0.044000

decrease: 0.038000

saw: 0.063000

LQS 50k

random: 0.016000

increase: 0.021000

decrease: 0.021000

saw: 0.080000

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

SHELL 100

random: 0.803000

increase: 0.002000

decrease: 1.611000

saw: 0.802000

QS 100

random: 0.028000

increase: 0.042000

decrease: 0.039000

saw: 0.134000

LQS 100k

random: 0.031000

increase: 0.046000

decrease: 0.057000

saw: 0.176000

### Листинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "Header.h"

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

double 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;

double start\_time = clock();

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;

}

}

double end\_time = clock();

double search\_time = end\_time - start\_time;

return(search\_time);

}

double qs(int\* items, int left, int right)

{

int i, j;

int x, y;

i = left; j = right;

x = items[(left + right) / 2+100];

double start\_time = clock();

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);

double end\_time = clock();

double search\_time = end\_time - start\_time;

return(search\_time);

}

void First\_array(int \* Array\_random, int\* Array\_increase, int\* Array\_decrease, int\* Array\_saw,int count) {

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

Array\_random[i] = rand() % 100;

}

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

Array\_increase[i] = i;

}

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

Array\_decrease[i] = count - i;

}

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

if (i <= count / 2) {

Array\_saw[i] = i;

}

if (i > count / 2) {

Array\_saw[i] = count - i;

}

}

}

void Second\_array(int\* Array\_random, int\* Array\_increase, int\* Array\_decrease, int\* Array\_saw, int count) {

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

Array\_random[i] = rand() % 100;

}

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

Array\_increase[i] = i;

}

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

Array\_decrease[i] = count - i;

}

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

if (i <= count / 2) {

Array\_saw[i] = i;

}

if (i > count / 2) {

Array\_saw[i] = count - i;

}

}

}

void Third\_array(int\* Array\_random, int\* Array\_increase, int\* Array\_decrease, int\* Array\_saw, int count) {

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

Array\_random[i] = rand() % 100;

}

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

Array\_increase[i] = i;

}

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

Array\_decrease[i] = count - i;

}

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

if (i <= count / 2) {

Array\_saw[i] = i;

}

if (i > count / 2) {

Array\_saw[i] = count - i;

}

}

}

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

{

return \*i - \*j;

}

int main(){

FILE\* fp;

fp = fopen("output.txt", "w");

double start\_time,end\_time,search\_time;

srand(time(NULL));

int count;

int k10 = 10000;

int k20 = 20000;

int k50 = 50000;

int k100 = 100000;

//10k//

int\* Array\_random = new int[k10];

int\* Array\_increase = new int[k10];

int\* Array\_decrease = new int[k10];

int\* Array\_saw = new int[k10];

//SHELL//

fprintf(fp,"SHELL 10k\n");

First\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k10);

search\_time = shell(Array\_random, k10);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_increase, k10);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_decrease, k10);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_saw, k10);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//QS//

fprintf(fp, "QS 10k\n");

Second\_array(Array\_random, Array\_increase,Array\_decrease, Array\_saw, k10);

int left = 0;

int right = k10 - 1;

search\_time=qs(Array\_random,left, right);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_increase, left, right);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_decrease, left, right);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_saw, left, right);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//LQS//

fprintf(fp, "LQS 10k\n");

Third\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k10);

start\_time = clock();

qsort(Array\_random, k10, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_increase, k10, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_decrease, k10, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_saw, k10, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

fprintf(fp,"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

//20k//

Array\_random = new int[k20];

Array\_increase = new int[k20];

Array\_decrease = new int[k20];

Array\_saw = new int[k20];

//SHELL//

fprintf(fp, "SHELL 20k\n");

First\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k20);

search\_time = shell(Array\_random, k10);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_increase, k20);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_decrease, k20);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_saw, k20);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//QS//

fprintf(fp, "QS 20k\n");

Second\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k20);

right = k20 - 1;

search\_time = qs(Array\_random, left, right);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_increase, left, right);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_decrease, left, right);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_saw, left, right);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//LQS//

fprintf(fp, "LQS 20k\n");

Third\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k20);

start\_time = clock();

qsort(Array\_random, k20, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_increase, k20, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_decrease, k20, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_saw, k20, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

fprintf(fp, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

//50k//

Array\_random = new int[k50];

Array\_increase = new int[k50];

Array\_decrease = new int[k50];

Array\_saw = new int[k50];

//SHELL//

fprintf(fp, "SHELL 50k\n");

First\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k50);

search\_time = shell(Array\_random, k50);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_increase, k50);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_decrease, k50);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_saw, k50);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//QS//

fprintf(fp, "QS 50k\n");

Second\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k50);

right = k50 - 1;

search\_time = qs(Array\_random, left, right);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_increase, left, right);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_decrease, left, right);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_saw, left, right);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//LQS//

fprintf(fp, "LQS 50k\n");

Third\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k50);

start\_time = clock();

qsort(Array\_random, k50, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_increase, k50, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_decrease, k50, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_saw, k50, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

fprintf(fp, "\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");

//100k//

Array\_random = new int[k100];

Array\_increase = new int[k100];

Array\_decrease = new int[k100];

Array\_saw = new int[k100];

//SHELL//

fprintf(fp, "SHELL 100\n");

First\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k100);

search\_time = shell(Array\_random, k100);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_increase, k100);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_decrease, k100);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = shell(Array\_saw, k100);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//QS//

fprintf(fp, "QS 100\n");

Second\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k100);

right = k100 - 1;

search\_time = qs(Array\_random, left, right);

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_increase, left, right);

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_decrease, left, right);

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

search\_time = qs(Array\_saw, left, right);

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

//LQS//

fprintf(fp, "LQS 100k\n");

Third\_array(Array\_random, Array\_increase, Array\_decrease, Array\_saw, k100);

start\_time = clock();

qsort(Array\_random, k100, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "random: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_increase, k100, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "increase: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_decrease, k100, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "decrease: %lf\n", search\_time / CLK\_TCK);

start\_time = clock();

qsort(Array\_saw, k100, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end\_time = clock();

search\_time = end\_time - start\_time;

fprintf(fp, "saw: %lf\n\n", search\_time / CLK\_TCK);

fclose(fp);

system(“PAUSE”);

\_getch();

}

### Выводы

Оценил время выполнения программы.