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»

**Выполнил:**

Студент группы 19ВВ2

Отставнов А.М.

**Приняли:**

Митрохин М.А.

Юрова О.В.

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 c

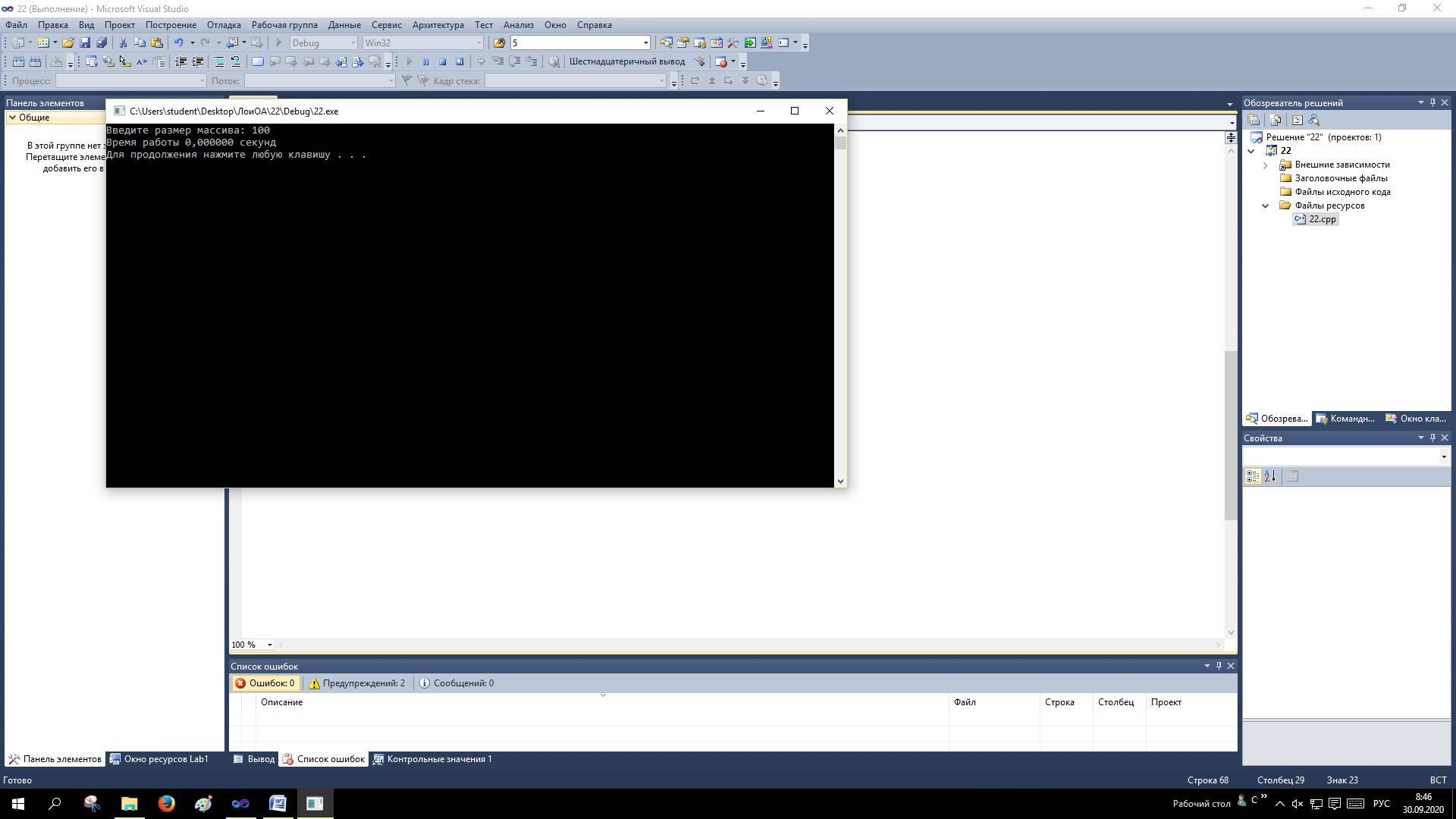


Рисунок 1. 100x100

**200:**

Время работы = 0,031 c

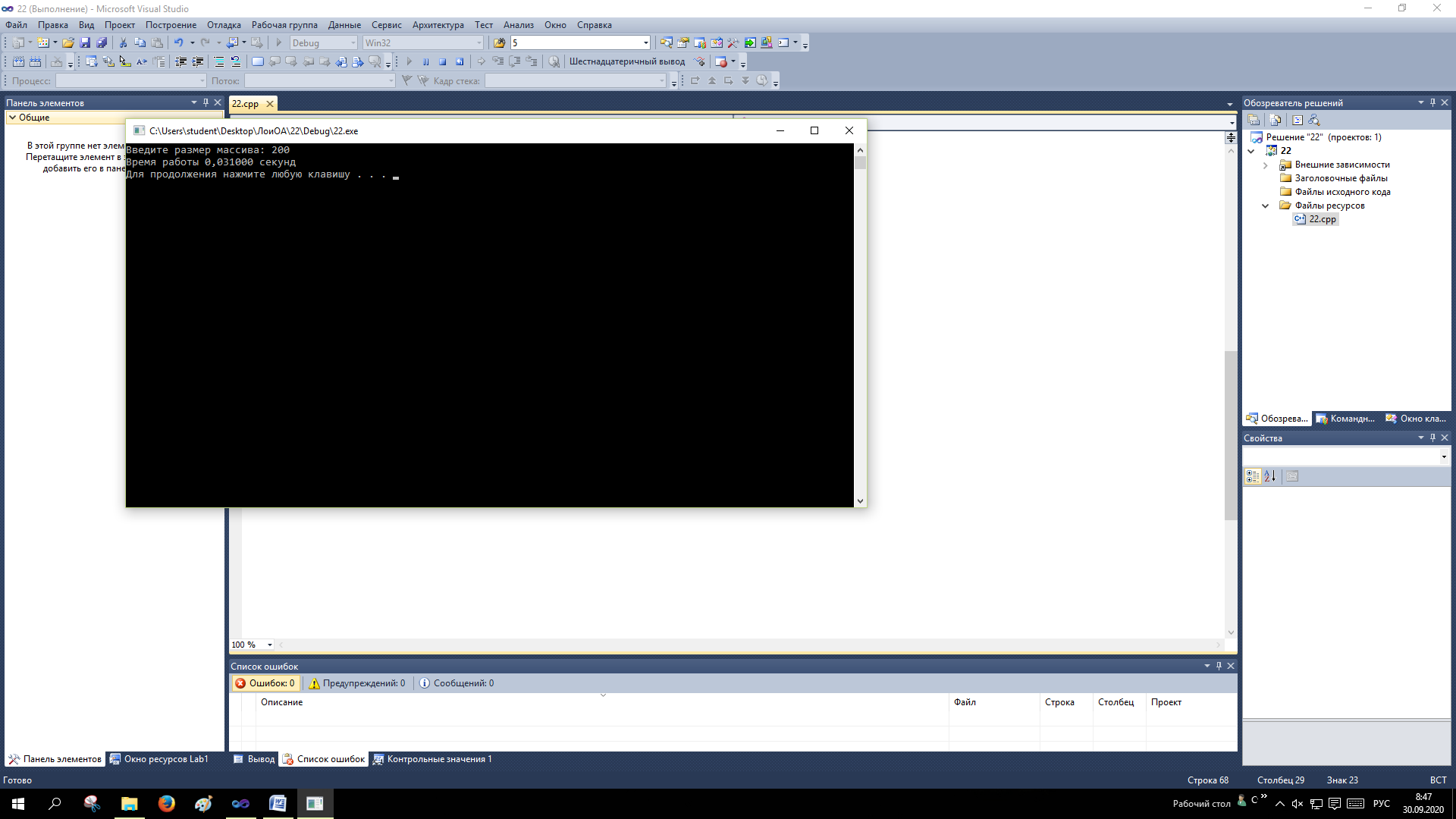
****

Рисунок 2.200x200

**400:**

Время работы = 0,281 c

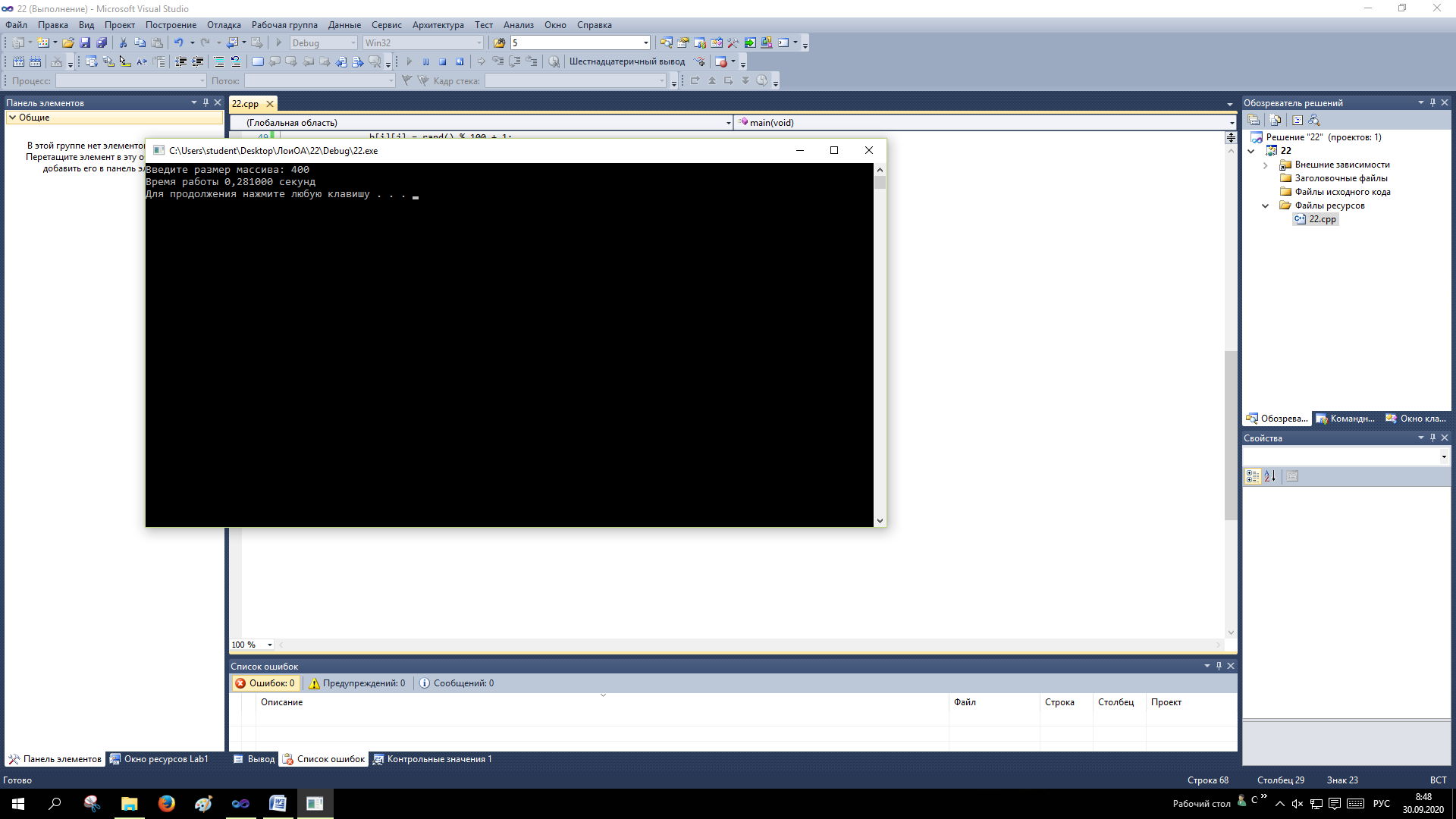


Рисунок 3.400x400

**1000:**

Время работы = 6,172 c

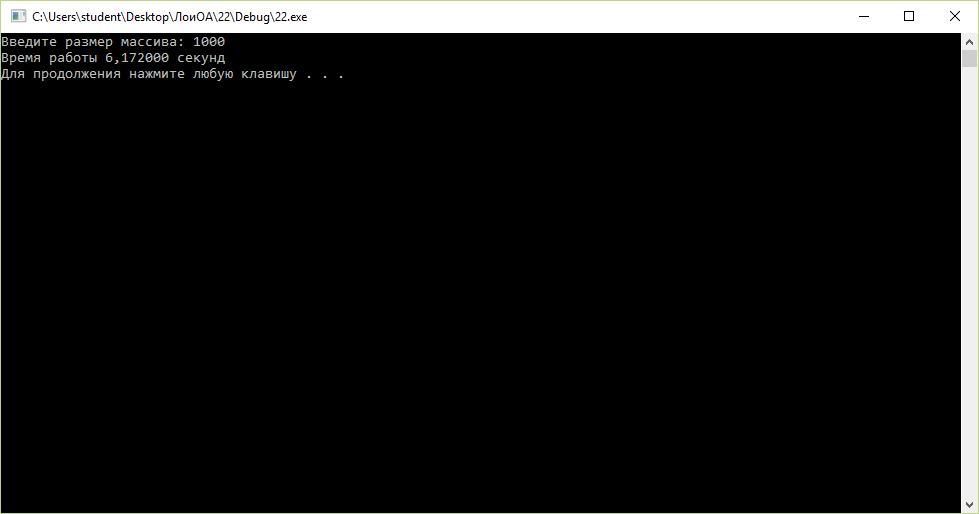


Рисунок 4.1000x1000

**2000:**

Время работы = 69,051 c

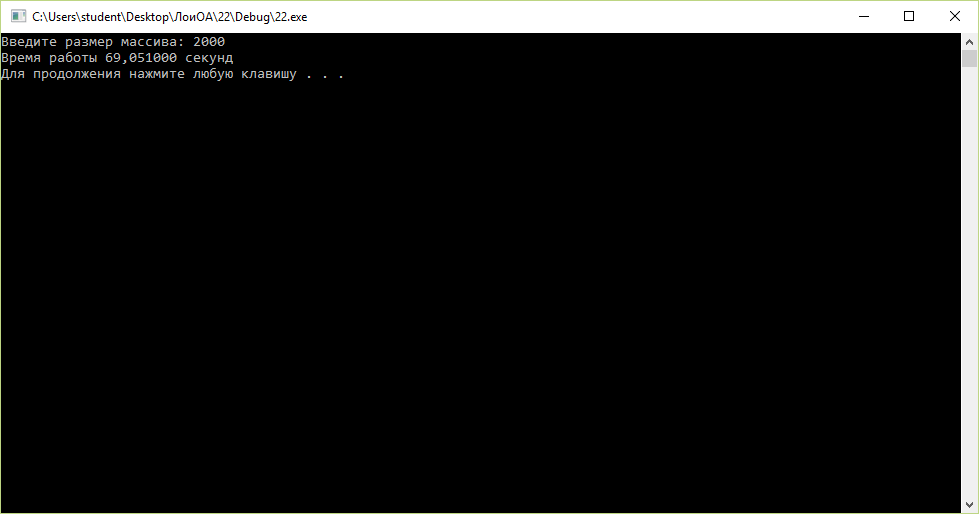


Рисунок 5.2000x2000

1. Построили график зависимости времени выполнения программы от размера матриц

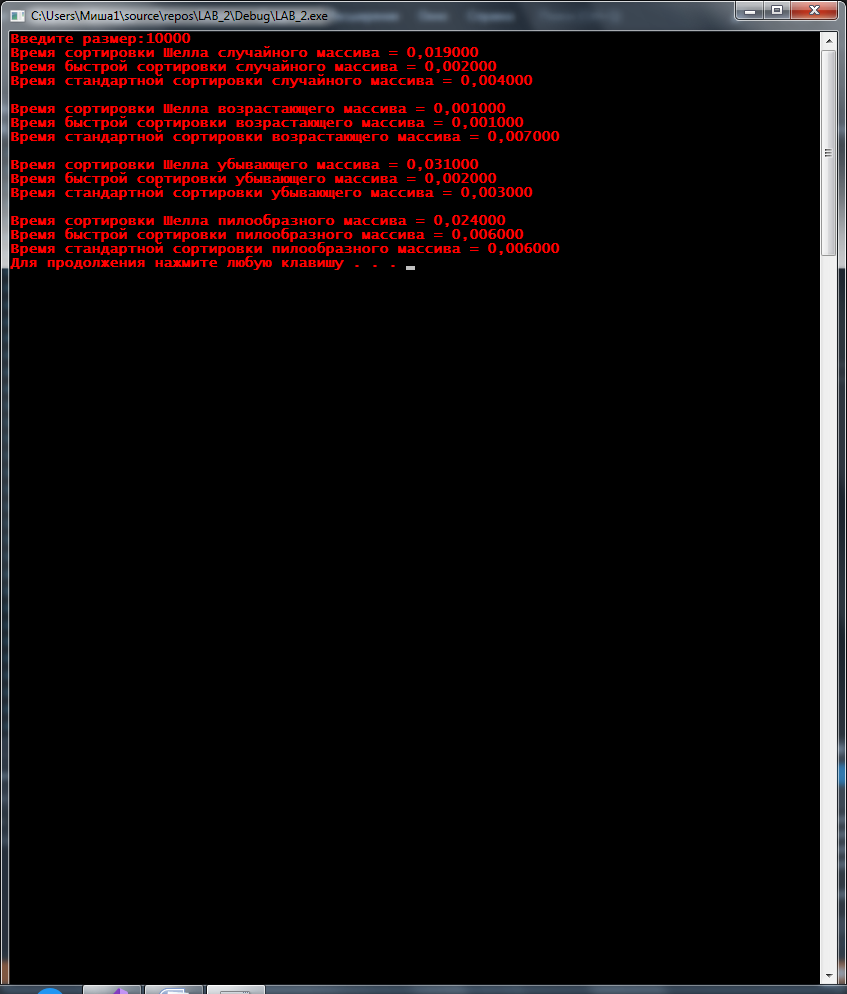
и сравнили полученный результат с теоретической оценкой.

Рисунок 6. график зависимости

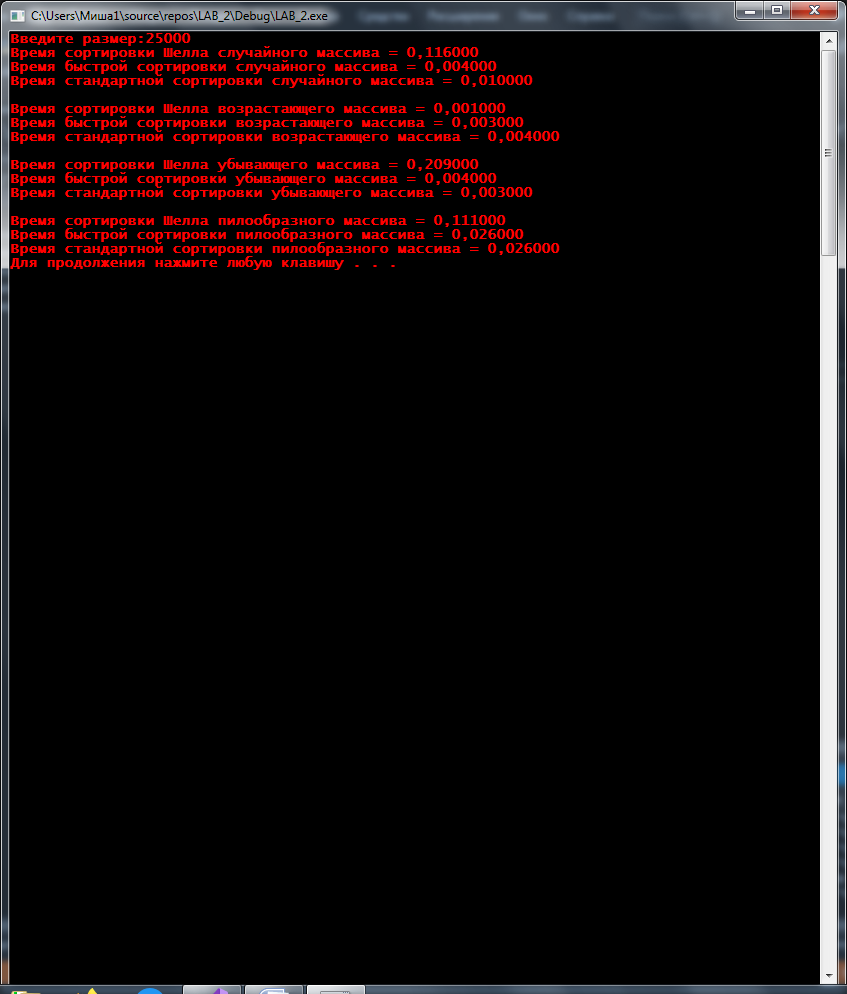
**Задание 2.**

**Результат работы программы:**

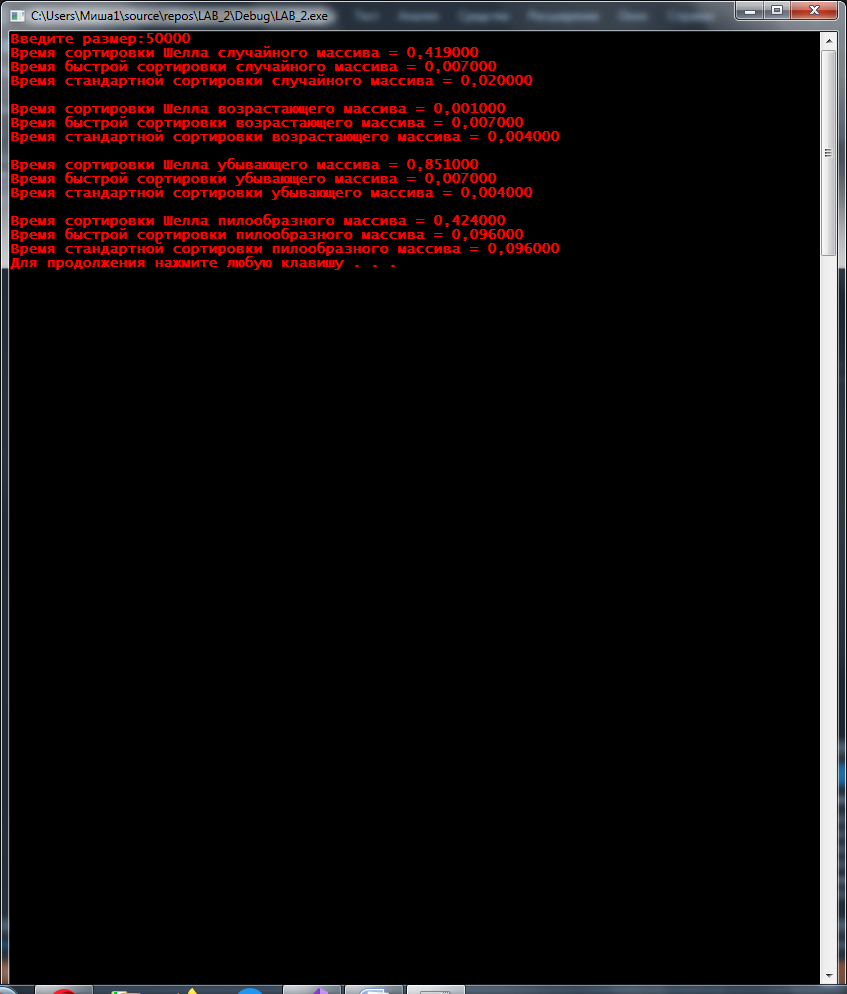
**1**) При сортировке 10000 элементов

****

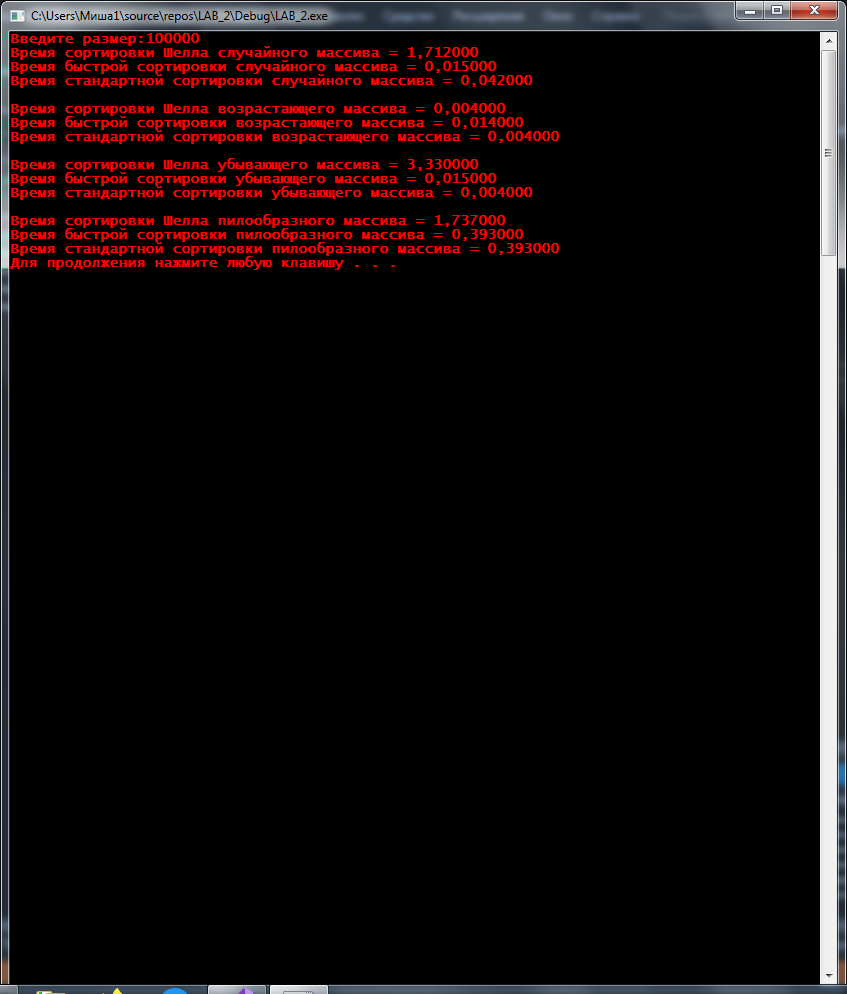
**2)** При сортировке 25000 элементов

****

**3)** При сортировке 50000 элементов

****

**4)** При сортировке 100000 элементов

****

### Листинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <random>

#include <stdlib.h>

#include<time.h>

#include<stdio.h>

#include<windows.h>

#include <malloc.h>

int main(void)

{

clock\_t start, end;

int i = 0, j = 0, r = 0;

int elem\_c, n;

printf("Enter lenght: ");

scanf("%d", &n);

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

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

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

}

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

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

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

}

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

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

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

}

srand(time(NULL));

while (i < n)

{

while (j < n)

{

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

j++;

}

i++;

}

srand(time(NULL));

i = 0; j = 0;

while (i < n)

{

while (j < n)

{

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

j++;

}

i++;

}

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;

}

}

}

end = clock();

printf("Time of sort = %lf sec.", (double)(end - start) / CLK\_TCK);

free(a);

free(b);

free(c);

system("pause");

return(0);}

void zadanie\_1()  
  
{  
  
clock\_t start, end;  
  
int i = 0, j = 0, r = 0;  
  
int elem\_c, n;  
  
printf("Enter lenght: ");  
  
scanf("%d", &n);  
  
int\*\* a = (int\*\*)malloc(sizeof(int\*) \* n);  
  
for (int i = 0; i < n; i++) {  
  
a[i] = (int\*)malloc(n \* sizeof(int));  
  
}  
  
int\*\* b = (int\*\*)malloc(sizeof(int) \* n);  
  
for (int i = 0; i < n; i++) {  
  
b[i] = (int\*)malloc(n \* sizeof(int));  
  
}  
  
int\*\* c = (int\*\*)malloc(sizeof(int) \* n);  
  
for (int i = 0; i < n; i++) {  
  
c[i] = (int\*)malloc(n \* sizeof(int));  
  
}  
  
srand(time(NULL));  
  
while (i < n)  
  
{  
  
while (j < n)  
  
{  
  
a[i][j] = rand() % 100 + 1;  
  
j++;  
  
}  
  
i++;  
  
}  
  
srand(time(NULL));  
  
i = 0; j = 0;  
  
while (i < n)  
  
{  
  
while (j < n)  
  
{  
  
b[i][j] = rand() % 100 + 1;  
  
j++;  
  
}  
  
i++;  
  
}  
  
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;  
  
}  
  
}  
  
}  
  
end = clock();  
  
printf("Time of sort = %lf sec.", (double)(end - start) / CLK\_TCK);  
  
free(a);  
  
free(b);  
  
free(c);  
  
system("pause");  
  
}  
  
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;  
  
x = items[(left + right) / 2 - 20];  
  
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 int\* i, const int\* j)  
  
{  
  
return \*i - \*j;  
  
}  
  
void zadani\_2() {  
  
setlocale(LC\_ALL, "RUS");  
int n;  
srand(time(NULL));  
printf("Введите размер:");  
scanf("%d", &n);  
  
int\* RAND = new int[n];  
int\* RAND\_3 = new int[n];  
int\* RAND\_2 = new int[n];  
  
int\* UP = new int[n];  
int\* UP\_2 = new int[n];  
int\* UP\_3 = new int[n];  
  
int\* DOWN = new int[n];  
int\* DOWN\_3 = new int[n];  
int\* DOWN\_2 = new int[n];  
  
int\* UP\_DOWN = new int[n];  
int\* UP\_DOWN\_2 = new int[n];  
int\* UP\_DOWN\_3 = new int[n];  
  
for (int i = 0; i < n; i++) {  
  
RAND[i] = rand() % 100;  
RAND\_2[i] = RAND[i];  
RAND\_3[i] = RAND[i];  
  
}  
  
for (int i = 0; i < n; i++) {  
  
UP[i] = i;  
UP\_2[i] = UP[i];  
UP\_3[i] = UP[i];  
}  
  
for (int i = 0; i < n; i++) {  
  
DOWN[i] = n - i;  
DOWN\_2[i] = DOWN[i];  
DOWN\_3[i] = DOWN[i];  
}  
  
for (int i = 0; i < n; i++) {  
  
if (i <= n / 2) {  
  
UP\_DOWN[i] = i;  
UP\_DOWN\_2[i] = UP\_DOWN[i];  
UP\_DOWN\_3[i] = UP\_DOWN[i];  
}  
  
if (i > n / 2) {  
  
UP\_DOWN[i] = n - i;  
UP\_DOWN\_2[i] = UP\_DOWN[i];  
UP\_DOWN\_3[i] = UP\_DOWN[i];  
}  
  
}  
  
double start = clock();  
shell(RAND, n);  
double end = clock();  
printf("Время сортировки Шелла случайного массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
  
int left = 0;  
double start\_new = clock();  
qs(RAND\_2, left, n - 1);  
double end\_new = clock();  
printf("Время быстрой сортировки случайного массива = %lf\n", (end\_new - start\_new) / CLOCKS\_PER\_SEC);  
  
start = clock();  
qsort(RAND\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
printf("Время стандартной сортировки случайного массива = %lf\n\n", (end - start) / CLOCKS\_PER\_SEC);  
start = clock();  
shell(UP, n);  
end = clock();  
printf("Время сортировки Шелла возрастающего массива = %lf\n", (end - start) /

CLOCKS\_PER\_SEC);  
  
start = clock();  
qs(UP\_2, left, n - 1);  
end = clock();  
printf("Время быстрой сортировки возрастающего массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
  
start = clock();  
qsort(UP\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
printf("Время стандартной сортировки возрастающего массива = %lf\n\n", (end - start) / CLOCKS\_PER\_SEC);  
  
start = clock();  
shell(DOWN, n);  
end = clock();  
printf("Время сортировки Шелла убывающего массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
  
start = clock();  
qs(DOWN\_2, left, n - 1);  
end = clock();  
printf("Время быстрой сортировки убывающего массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
  
start = clock();  
qsort(DOWN\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
printf("Время стандартной сортировки убывающего массива = %lf\n\n", (end - start) / CLOCKS\_PER\_SEC);  
  
start = clock();  
shell(UP\_DOWN, n);  
end = clock();  
printf("Время сортировки Шелла пилообразного массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
  
start = clock();  
qs(UP\_DOWN\_2, left, n - 1);  
end = clock();  
printf("Время быстрой сортировки пилообразного массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
start = clock();  
qsort(UP\_DOWN\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
printf("Время стандартной сортировки пилообразного массива = %lf\n", (end - start) / CLOCKS\_PER\_SEC);  
  
  
system("pause");  
  
}  
  
int main()  
{  
zadanie\_1();  
zadani\_2();  
return 0;  
}

### Вывод:

Быстрая сортировка является одной из самых быстрых сортировок общего назначения, но может привести к переполнению стека. Сортировка Шелла же в основном медленнее, но она не деградирует и является стабильной, в отличие от быстрой сортировки.