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 19ВВ3

Субботкин М.В.

**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

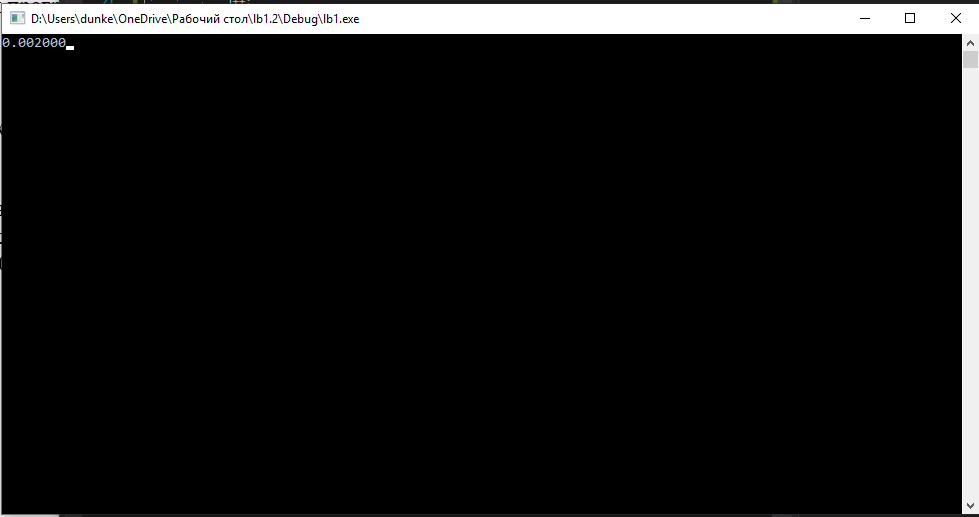


Рисунок 1. 100x100

**200:**

Время работы= 0,03 c

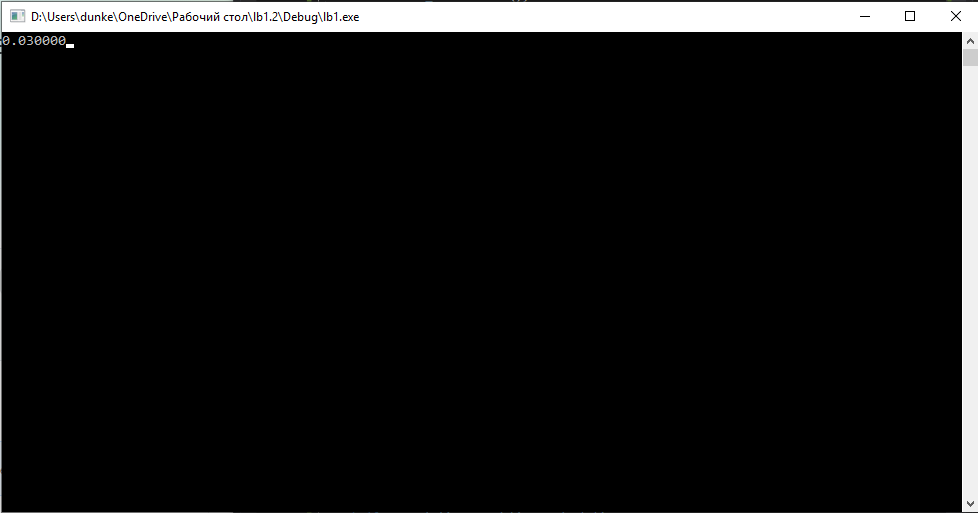


Рисунок 2.200x200

**400:**

Время работы= 0,268 c

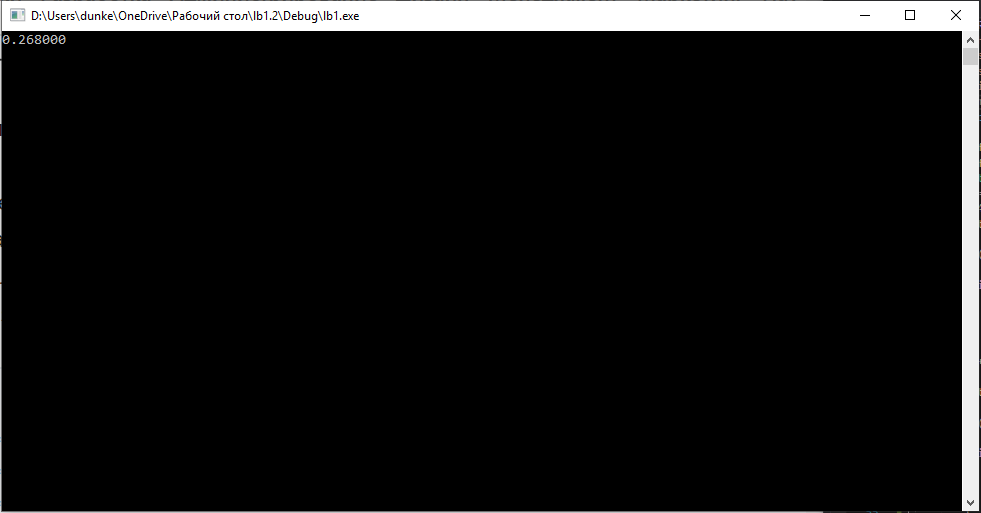


Рисунок 3.400x400

**1000:**

Время работы= 5,684 c

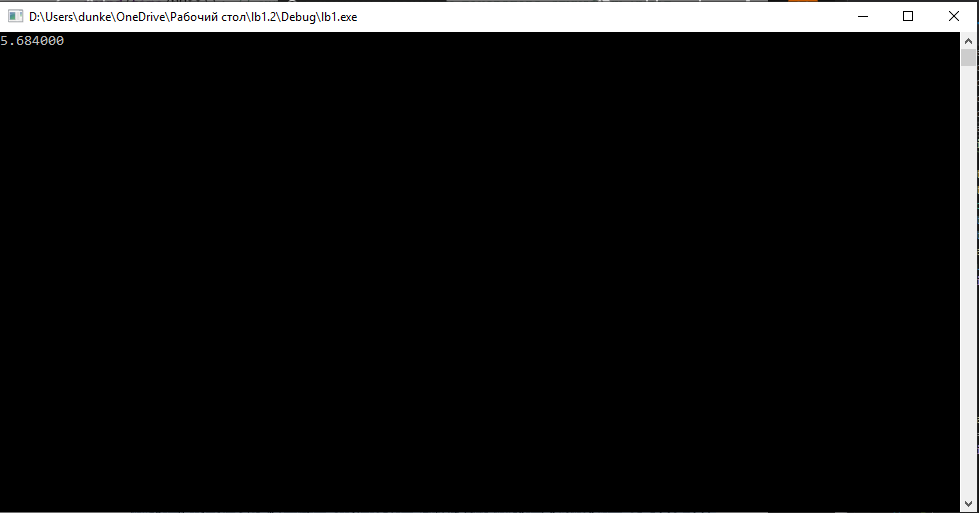


Рисунок 4.1000x1000

**2000:**

Время работы= 75,117 c

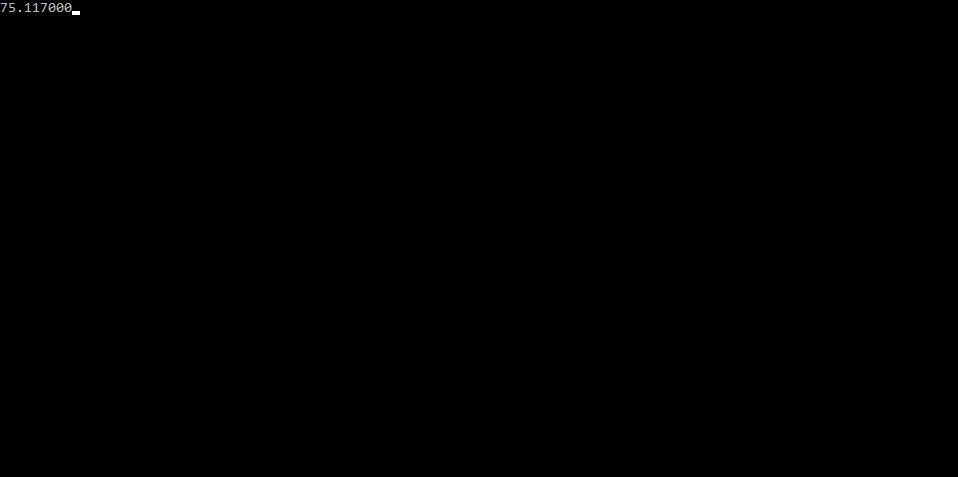


Рисунок 5.2000x2000

1. Построили график зависимости времени выполнения программы от размера матриц

и сравнили полученный результат с теоретической оценкой.

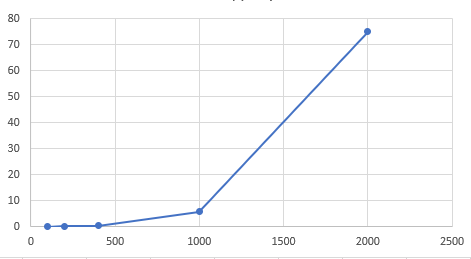


Рисунок 6. график зависимости

**Листинг (Задание 1):**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <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);

}

**Задание 2.**

**Результат работы программы:**

Количество Шелла qsort qsort(библиотечный)

10000 0.016 0.002 0.005 РАНДОМ

10000 0.001 0.001 0.000 ВОЗРАСТАЮЩИЙ

10000 0.016 0.005 0.005 ПИЛА

10000 0.713 0.007 0.0130 УБЫВАЮЩИЙ

50000 0.283 0.007 0.02 РАНДОМ

50000 0.001 0.007 0.000 ВОЗРАСТАЮЩИЙ

50000 0.28 0.139 0.102 ПИЛА

50000 0.593 0.007 0.023 УБЫВАЮЩИЙ

100000 1.248 0.014 0.035 РАНДОМ

100000 0.003 0.013 0.000 ВОЗРАСТАЮЩИЙ

100000 1.192 0.47 0.47 ПИЛА

100000 2.753 0.015 0.0510 УБЫВАЮЩИЙ

### Листинг:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <random>

#include <time.h>

#include <stdlib.h>

#include <stdio.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;

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 cmpfunc(const int \*i, const int \*j) {

return \*i - \*j;

}

int main() {

FILE\* sort;

sort = fopen("C:\\sortt.txt","a");

int n;

srand(time(NULL));

printf("Enter lenght:");

scanf("%d",&n);

fprintf(sort,"Number of elements = %d\n", n);

int\* arr\_random =new int[n];

int\* arr\_vozrast =new int[n];

int\* arr\_ubiva = new int[n];

int\* arr\_pila= new int[n];

int\* arr\_random\_3 = new int[n];

int\* arr\_random\_2 = new int[n];

int\* arr\_vozrast\_2 = new int[n];

int\* arr\_vozrast\_3 = new int[n];

int\* arr\_ubiva\_3 = new int[n];

int\* arr\_ubiva\_2 = new int[n];

int\* arr\_pila\_2 = new int[n];

int\* arr\_pila\_3 = new int[n];

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

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

arr\_random\_2[i] = arr\_random[i];

arr\_random\_3[i] = arr\_random[i];

}

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

arr\_vozrast[i] =i;

arr\_vozrast\_2[i] = arr\_vozrast[i];

arr\_vozrast\_3[i] = arr\_vozrast[i];

}

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

arr\_ubiva[i] = n-i;

arr\_ubiva\_2[i] = arr\_ubiva[i];

arr\_ubiva\_3[i] = arr\_ubiva[i];

}

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

if (i <= n / 2) {

arr\_pila[i] = i;

arr\_pila\_2[i] = arr\_pila[i];

arr\_pila\_3[i] = arr\_pila[i];

}

if (i > n / 2) {

arr\_pila[i] = n - i;

arr\_pila\_2[i] = arr\_pila[i];

arr\_pila\_3[i] = arr\_pila[i];

}

}

double start = clock();

shell(arr\_random, n);

double end = clock();

fprintf(sort,"Time shell random = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

int left = 0;

double start\_new = clock();

qs(arr\_random\_2, left, n-1);

double end\_new = clock();

fprintf(sort,"Time qsort random = %lf\n", (end\_new - start\_new) / CLOCKS\_PER\_SEC);

start = clock();

qsort(arr\_random\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))cmpfunc);

end = clock();

fprintf(sort,"Standart qsort random = %lf\n" , (end - start) / CLOCKS\_PER\_SEC);

start = clock();

shell(arr\_vozrast, n);

end = clock();

fprintf(sort,"Time shell vozrast = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(arr\_vozrast\_2, left, n - 1);

end = clock();

fprintf(sort,"Time qsort vozrast = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(arr\_vozrast\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))cmpfunc);

fprintf(sort,"Standart qsort vozrast = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

end = clock();

start = clock();

shell(arr\_pila, n);

end = clock();

fprintf(sort,"Time shell pila = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(arr\_pila\_2, left, n - 1);

end= clock();

fprintf(sort,"Time qsort pila = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

qsort(arr\_pila\_3, n, sizeof(int), (int(\*)(const void\*, const void\*))cmpfunc);

fprintf(sort,"Standart qsort pila = %lf\n", (end - start) / CLOCKS\_PER\_SEC);

return 0;

}

### Выводы

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.