Министерство науки и высшего 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»

Выполнил:

Студент группы 23ВВВ2

Стрельцов А.П.

Федоров Б.М.

Приняли:

Юрова О. В.

Митрохин М. А.

Пенза 2024

**Цель работы:** изучить новые функции, применить их в лабораторной работе и оценить время выполнения данных нам программ.

**Лабораторное задание.**

**Задание 1**:

1. **Вычислить порядок сложности программы (*О*-символику).**

***Инициализация матриц:***

Программа заполняет две матрицы случайными числами. Это происходит в двух вложенных циклах, каждый из которых выполняется 200 раз. Таким образом, сложность инициализации каждой матрицы составляет O(n^2), где n = 200.

***Умножение матриц:***

Умножение матриц реализовано с помощью трех вложенных циклов:

* Внешний цикл проходит по строкам первой матрицы (200 итераций).
* Средний цикл проходит по столбцам второй матрицы (200 итераций).
* Внутренний цикл выполняет суммирование произведений элементов (200 итераций).

Таким образом, сложность умножения матриц составляет O(n^3).

1. **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.**

* Время выполнения для матриц размера 100x100: 0,006000 секунд
* Время выполнения для матриц размера 200x200: 0,042000 секунд
* Время выполнения для матриц размера 400x400: 0,371000 секунд
* Время выполнения для матриц размера 1000x1000: 5,211000 секунд
* Время выполнения для матриц размера 2000x2000: 38,228000 секунд
* Время выполнения для матриц размера 4000x4000: 720,308105 секунд
* Время выполнения для матриц размера 10000x10000: 4684,081055 секунд

1. **Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.**

По графику видно, что время выполнения программы растет в геометрической прогрессии в зависимости от размера массива.

**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.

Shell сортировка на случайном наборе чисел показала наибольший по времени результат в 592 мс, что показало низкую эффективность на данном типе массивов.

Три оставшихся массива при сортировке показали высокие результаты.

Представленный в условиях лабораторной работы алгоритм QS, показал высокие результаты на всех видах массивов. Алгоритм сортировки QSort уступил по скорости сортировки случайного массива, но показал лучший результат на последних трех массивах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайный | Возрастающий | Убывающий | Половинчатый |
| Shell | 0.592 | 0.001 | 0.002 | 0.001 |
| QS | 0.007 | 0.003 | 0.003 | 0.003 |
| QSort | 0.011 | 0.001 | 0.001 | 0.001 |

**Пояснительный текст к программе**

Задание 1

Программа выполняет умножение матриц и измеряет время, затраченное на эту операцию для различных размеров матриц.

Задание 2

Программа измеряет время выполнения сортировки тремя различными способами на 4 видах массивов: случайный, возрастающий, убывающий и массиве, одна половина которого представляет собой возрастающую последовательность чисел, а другая, – убывающую.

**Листинг**

**Задание 1**

**laba 2 1-2.CPP**

#include <locale.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

setlocale(LC\_ALL, "RUS");

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

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

int i, j, r, n;

int\*\* a, \*\* b, \*\* c;

int elem\_c;

n = 200;

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

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

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

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

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

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

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

}

// Заполняем матрицы случайными числами

srand(time(NULL));

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

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

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

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

}

}

// Измеряем время выполнения перемножения матриц

clock\_t start, end;

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

}

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

}

}

end = clock();

// Вычисляем время выполнения в секундах

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

printf("Время выполнения для матриц размера %dx%d: %f секунд\n", n, n, time\_taken);

// Освобождаем память

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

return 0;

}

**Задание 2**

**laba 2-2.CPP**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

#include <ctime>

const int size = 100000;

int randomArray[size], randomArray2[size], randomArray3[size];

int increasingArray[size], increasingArray2[size], increasingArray3[size];

int decreasingArray[size], decreasingArray2[size], decreasingArray3[size];

int halfIncHalfDecArray[size], halfIncHalfDecArray2[size], halfIncHalfDecArray3[size];

void copyArray(int\* source, int\* destination, int\* destination2, int size) {

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

destination[i] = source[i];

destination2[i] = source[i];

}

}

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;

}

}

}

int compare(const void\* a, const void\* b) {

return (\*(int\*)a - \*(int\*)b);

}

void qs(int\* items, int left, int right) {

int i, j;

int x, y;

i = left; j = right;

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 main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

double timeRandomShell, timeRandomQuick, timeRandomQSort, timeDescendingShell, timeDescendingQuick, timeDescendingQSort, timeAscendingShell, timeAscendingQuick, timeAscendingQSort, timeHalfShell, timeHalfQuick, timeHalfQSort;

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

randomArray[i] = rand() % 1000;

increasingArray[i] = i;

decreasingArray[i] = size - i;

halfIncHalfDecArray[i] = (i < size / 2) ? i : size - i;

}

copyArray(randomArray, randomArray2, randomArray3, size);

copyArray(&increasingArray[size], &increasingArray2[size], &increasingArray3[size], size);

copyArray(&decreasingArray[size], &decreasingArray2[size], &decreasingArray3[size], size);

copyArray(&halfIncHalfDecArray[size], &halfIncHalfDecArray2[size], &halfIncHalfDecArray3[size], size);

clock\_t start = clock();

shell(randomArray, size);

clock\_t end = clock();

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

start = clock();

qs(randomArray2, 0, size - 1);

end = clock();

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

start = clock();

qsort(randomArray3, size, sizeof(int), compare);

end = clock();

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

printf("Random Array:\n");

printf("Shell: %lf seconds\n", timeRandomShell);

printf("Quick Sort: %lf seconds\n", timeRandomQuick);

printf("QSort: %lf seconds\n\n", timeRandomQSort);

start = clock();

shell(decreasingArray, size);

end = clock();

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

start = clock();

qs(decreasingArray2, 0, size - 1);

end = clock();

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

start = clock();

qsort(decreasingArray3, size, sizeof(int), compare);

end = clock();

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

printf("Descending Array:\n");

printf("Shell: %lf seconds\n", timeDescendingShell);

printf("Quick Sort: %lf seconds\n", timeDescendingQuick);

printf("QSort: %lf seconds\n\n", timeDescendingQSort);

start = clock();

shell(increasingArray, size);

end = clock();

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

start = clock();

qs(increasingArray2, 0, size - 1);

end = clock();

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

start = clock();

qsort(increasingArray3, size, sizeof(int), compare);

end = clock();

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

printf("Ascending Array:\n");

printf("Shell: %lf seconds\n", timeAscendingShell);

printf("Quick Sort: %lf seconds\n", timeAscendingQuick);

printf("QSort: %lf seconds\n\n", timeAscendingQSort);

start = clock();

shell(halfIncHalfDecArray, size);

end = clock();

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

start = clock();

qs(halfIncHalfDecArray2, 0, size - 1);

end = clock();

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

start = clock();

qsort(halfIncHalfDecArray3, size, sizeof(int), compare);

end = clock();

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

printf("Half-and-Half Array:\n");

printf("Shell: %lf seconds\n", timeHalfShell);

printf("Quick Sort: %lf seconds\n", timeHalfQuick);

printf("QSort: %lf seconds\n\n", timeHalfQSort);

return 0;

}

**Результат работы программы**

Результаты работы программы для задания № 1 пункт 2 показаны на рисунках ниже.

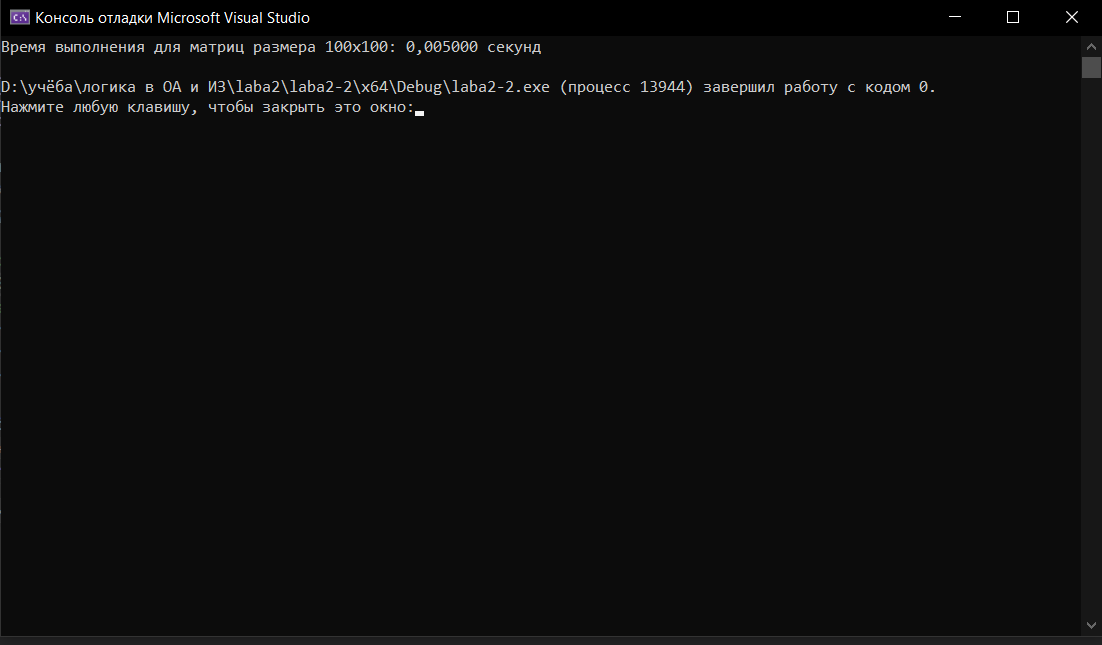


Рисунок 1 - Результат работы программы для матрицы 100x100.

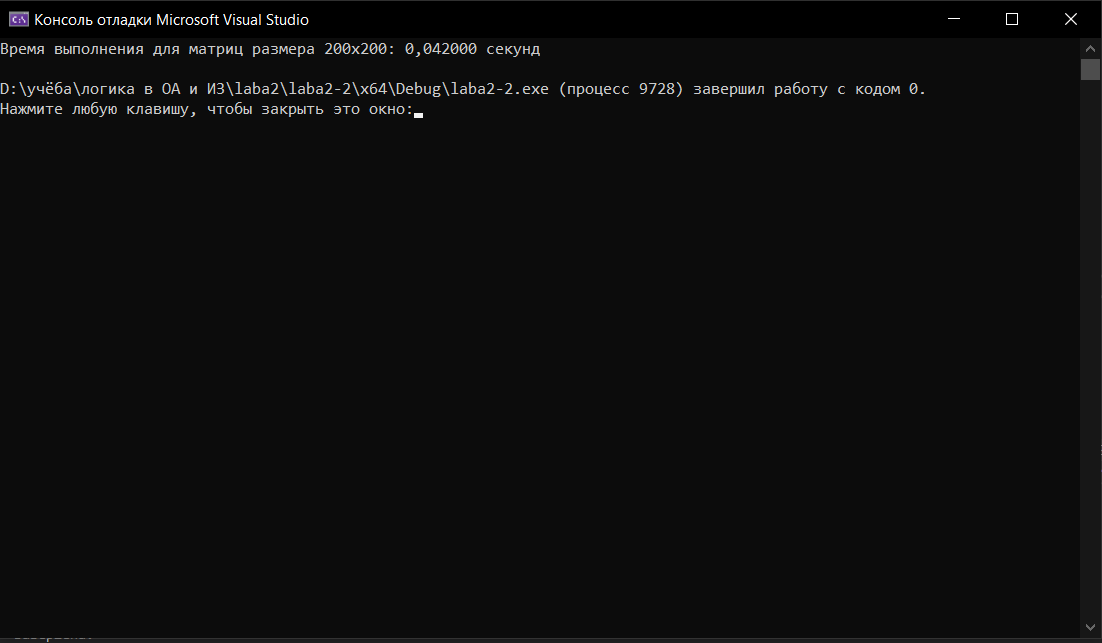


Рисунок 2 - Результат работы программы для матрицы 200x200.

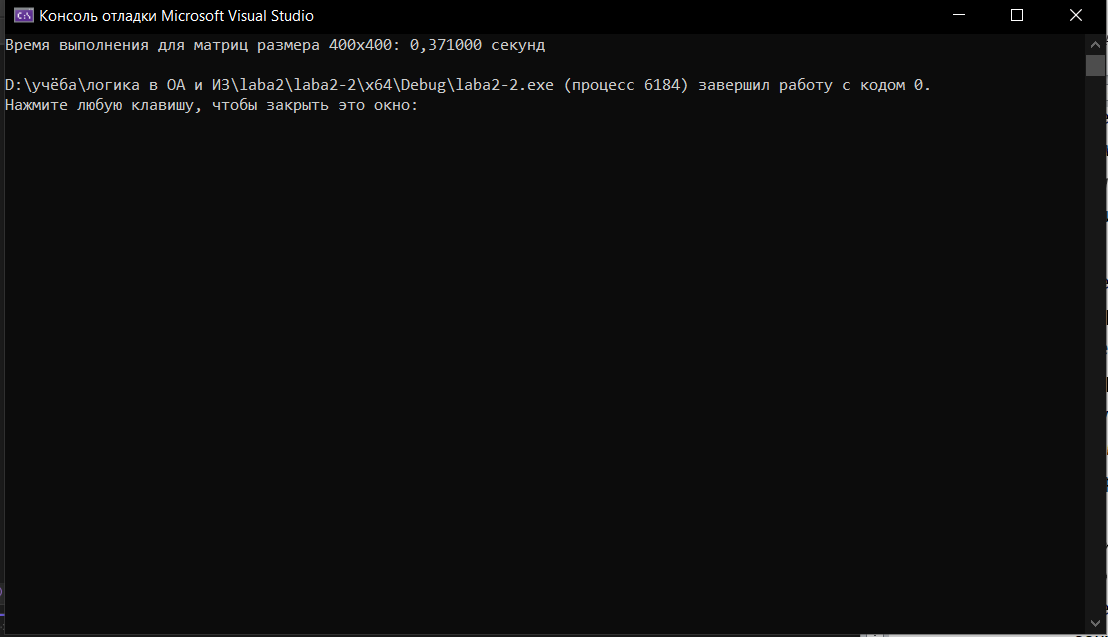


Рисунок 3 - Результат работы программы для матрицы 400x400.

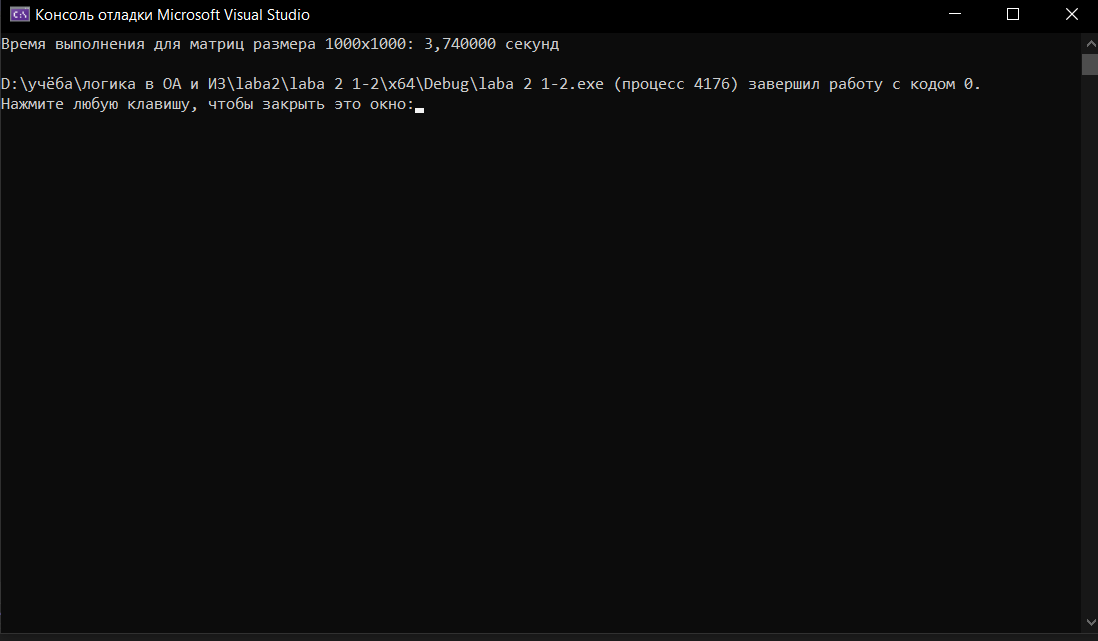


Рисунок 4 - Результат работы программы для матрицы 1000x1000.

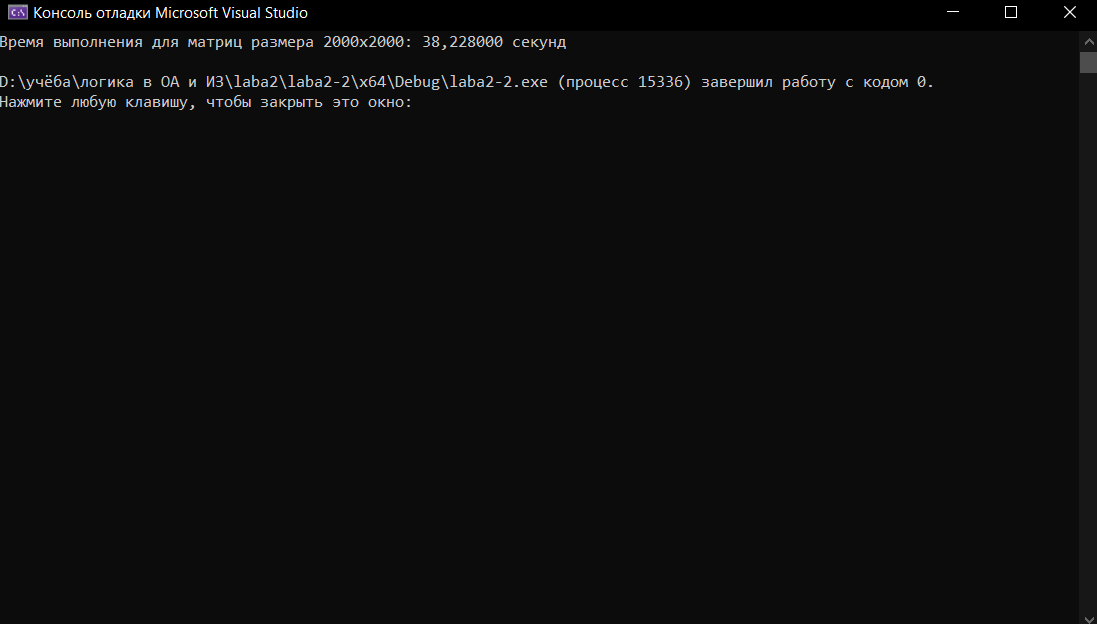


Рисунок 5 - Результат работы программы для матрицы 2000x2000.

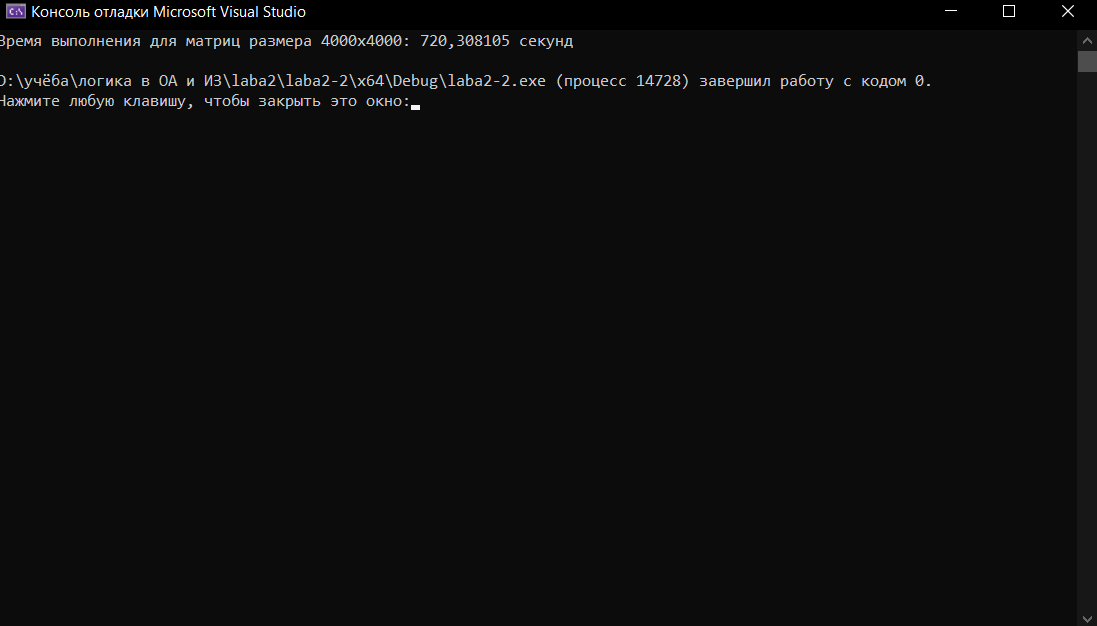


Рисунок 6 - Результат работы программы для матрицы 4000x4000.

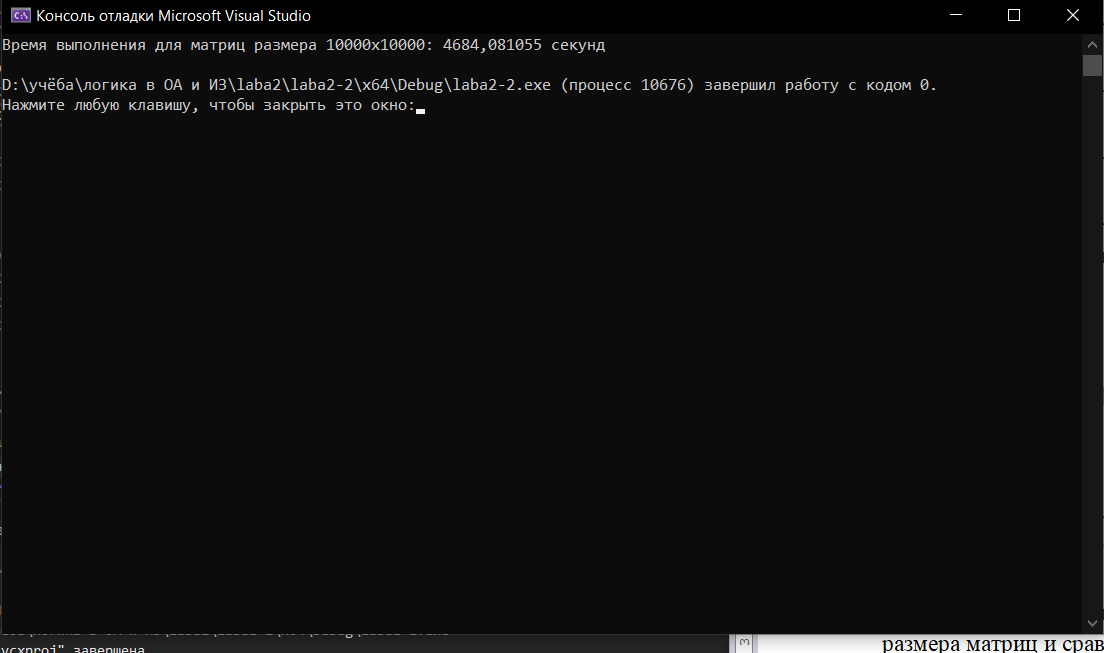


Рисунок 7 - Результат работы программы для матрицы 10000x10000.

**Изображение выглядит как текст, снимок экрана, программное обеспечение, компьютер

Автоматически созданное описание**

Рисунок 8 — Результаты работы программы пункта №2.

**Вывод**

В ходе выполнения лабораторной работы была разработана программы, выполняющие задания лабораторной работы. В процессе выполнения работы были использованы знания о расчёте сложности алгоритмов и затратах времени в зависимости от величины входных данных.

.