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 программ»

**Выполнили:**

студенты группы 19ВВ2

Сидоров Н. Р.

Карамышев А. А.

Горбунов Д.А.

**Приняли:**

Юрова О. В.

Митрохин М. А.

Пенза 2020

**Название:**

Оценка времени выполнения программ.

**Цель работы:**

Научиться применять алгоритм оценки времени выполнения программ.

**Лабораторные задания:**

*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.

**Описание метода решения задач:**

*Для всех задач описание одинаковое!*

Для вычисления времени работы программы используются библиотеки *time.h* и *chrono* – первая с точностью до секунды, вторая до малых долей секунды, что позволяет с большей точностью определить малую скорость выполнения алгоритма.

Для *time.h* сначала нужно инициализировать переменные начала и конца работы:

time\_t start, end;

Затем до начала алгоритма нужно ввести функцию с переменной начала отсчета:

time(&start);

А в конце алгоритма ввести функцию с переменной конца отсчета:

time(&end);

Данная строка кода вычисляет время между точкой отсчета и конца, где переменная *seconds* и есть время работы программы:

int seconds = difftime(end, start);

Но такой таймер выдаст результат лишь в целых частях секунды, что не позволяет объективно оценить время работы.

Для работы со вторым таймером нужно определить класс:

class Timer

{

private:

using clock\_t = std::chrono::high\_resolution\_clock;

using second\_t = std::chrono::duration<double, std::ratio<1> >;

std::chrono::time\_point<clock\_t> m\_beg;

public:

Timer() : m\_beg(clock\_t::now())

{

}

void reset()

{

m\_beg = clock\_t::now();

}

double elapsed() const

{

return std::chrono::duration\_cast<second\_t>(clock\_t::now() - m\_beg).count();

}

};

Для самого времени отсчета нужна инициализация класса с какой-либо переменной, в моем случае *t:*

Timer t;

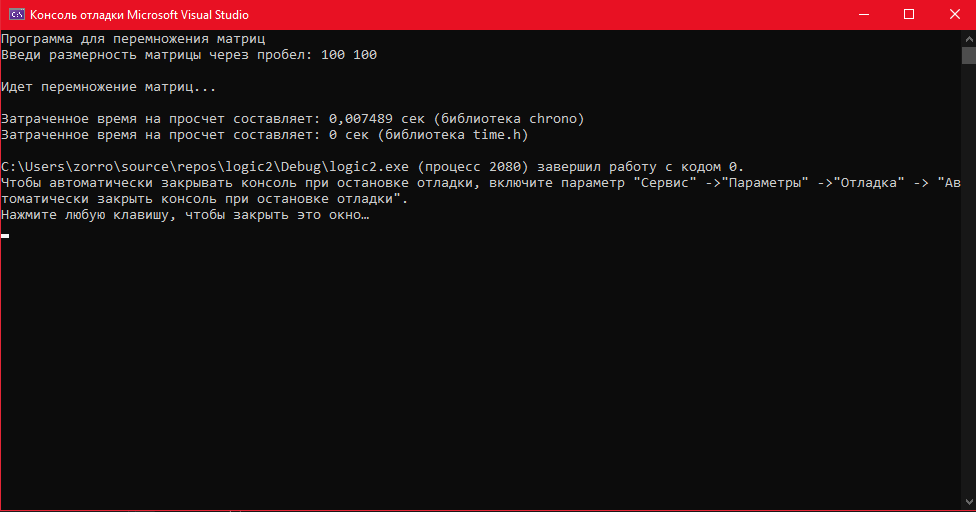
Вывод времени работы производится при помощи этой же переменной, но с добавлением *elapsed*, что означает «истекшее», т.е. сколько времени прошло до вывода переменной.

t1.elapsed()

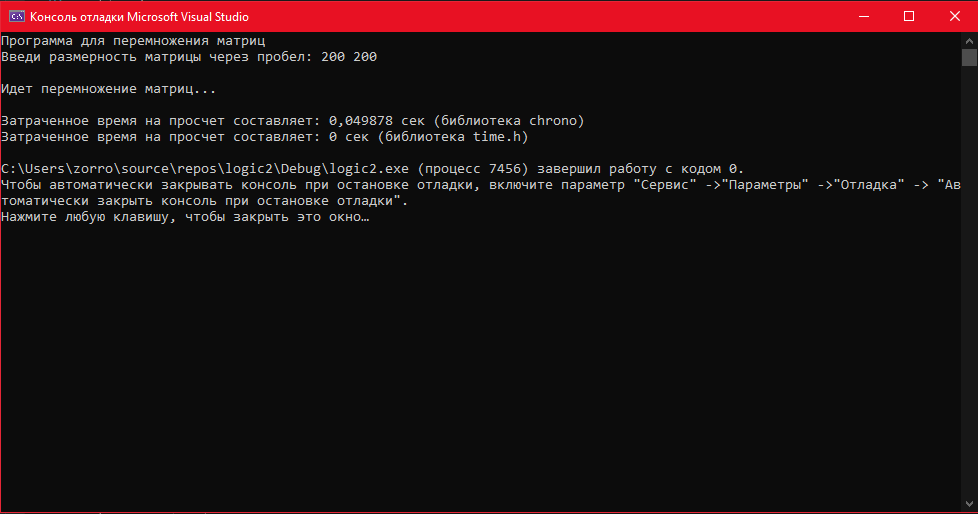
Полный код программы представлен в Приложении 1.

**Результат выполнения программы:**

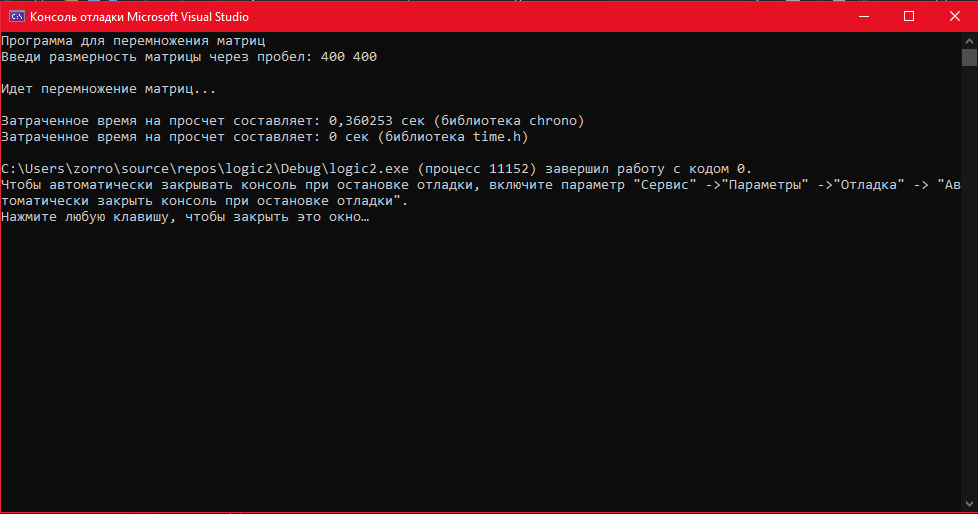
1. Матрица 100 на 100



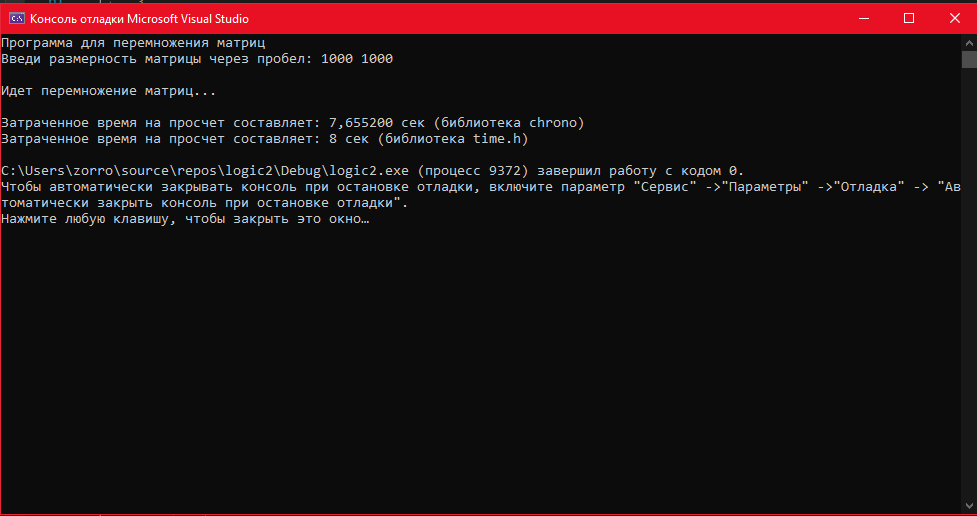
Матрица 200 на 200



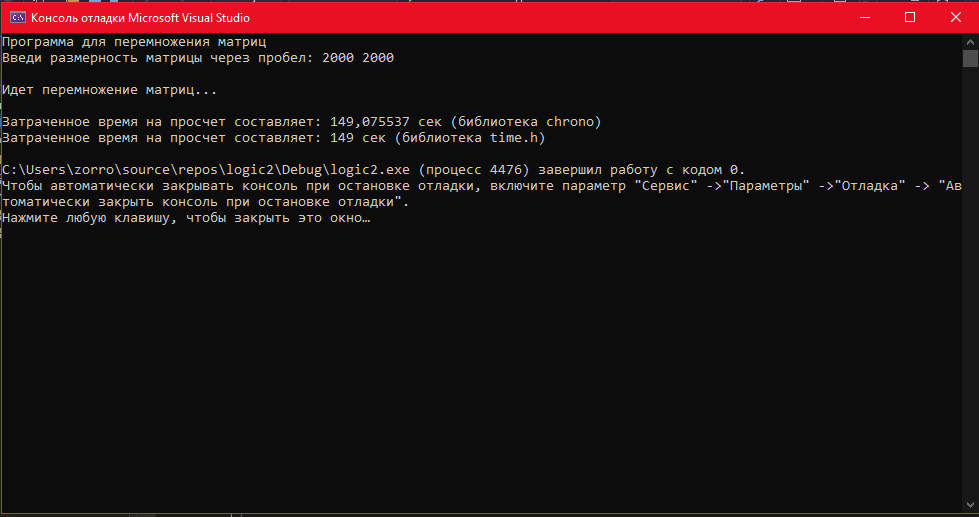
Матрица 400 на 400



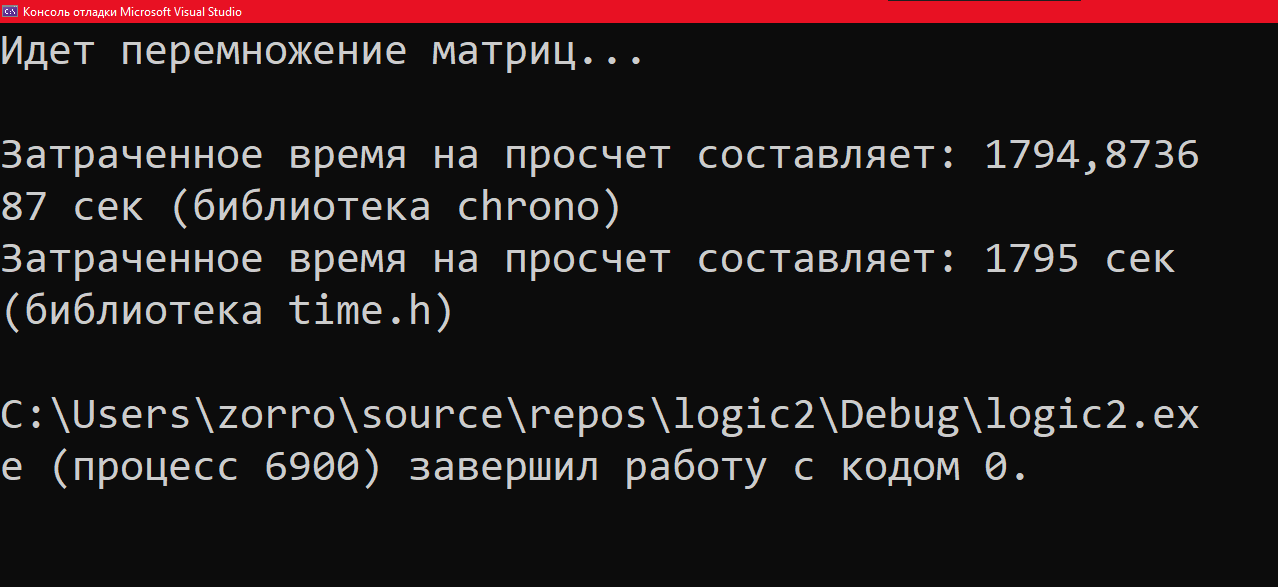
Матрица 1000 на 1000



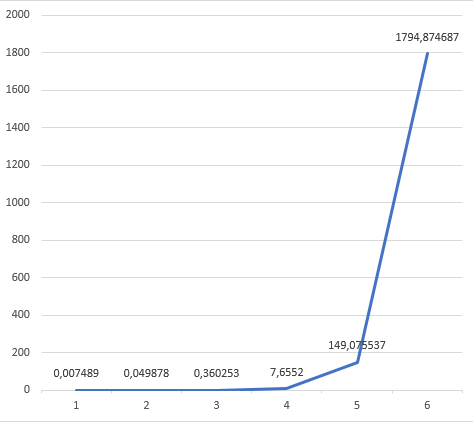
Матрица 2000 на 2000



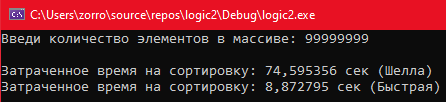
Матрица 4000 на 4000



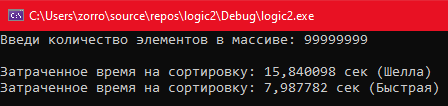
1. График зависимости времени работы от размера перемножаемых матриц:



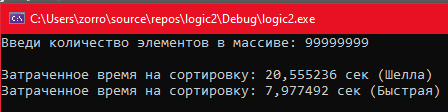
1. Сортировка случайно сгенерированного массива:



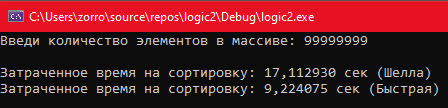
1. Сортировка возрастающего массива:



1. Сортировка убывающего массива:



1. Сортировка пирамидального массива:



**Вывод:**

Научились применять алгоритм оценки времени выполнения программ, а так же узнали в как в различных ситуациях ведут себя сортировка Шелла и быстрая сортировка. Выяснили, что:

1. На случайном наборе данных алгоритм быстрой сортировки работает намного быстрее, чем сортировка Шелла (75 vs 9)
2. На возрастающем наборе данных алгоритм Шелла работает намного быстрее (75 vs 15), а быстрая сортировка чуть быстрее (9 vs 8)
3. На убывающем наборе данных ситуация похожая (шелла работает чуть дольше)
4. Пирамидальный массив сортируется примерно за такое же время, как и убывающий.

Из этого можно сделать вывод, что сортировка Шелла очень плохо работает на случайном наборе данных, но намного быстрее работает на возрастающем/убывающем/пирамидальном массиве.

Быстрая сортировка в свою очередь примерно одинаково (+/-2 сек) работает на вышеперечисленных наборах данных.

**Приложение 1.**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <locale.h>

#include <windows.h>

#include <iostream>

#include <chrono>

class Timer

{

private:

using clock\_t = std::chrono::high\_resolution\_clock;

using second\_t = std::chrono::duration<double, std::ratio<1> >;

std::chrono::time\_point<clock\_t> m\_beg;

public:

Timer() : m\_beg(clock\_t::now())

{

}

void reset()

{

m\_beg = clock\_t::now();

}

double elapsed() const

{

return std::chrono::duration\_cast<second\_t>(clock\_t::now() - m\_beg).count();

}

};

int matrix(void)

{

setlocale(LC\_ALL, "Russian");

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

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

time\_t start, end; // объявляем переменные для определения времени выполнения

int n, m, elem\_c;

printf("Программа для перемножения матриц\nВведи размерность матрицы через пробел: ");

scanf\_s("%d %d", &n, &m);

int i = 0, j = 0, r;

int\*\* A;

int\*\* B;

int\*\* C;

A = new int\* [n];

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

A[i] = new int[m];

}

B = new int\* [n];

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

B[i] = new int[m];

}

C = new int\* [n];

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

C[i] = new int[m];

}

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

while (i < n)

{

while (j < m)

{

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 < m)

{

while (j < m)

{

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

j++;

}

i++;

}

printf("\nИдет перемножение матриц...\n");

time(&start);

Timer t;

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

{

for (j = 0; j < m; 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;

}

}

}

time(&end);

int seconds = difftime(end, start);

printf("\nЗатраченное время на просчет составляет: %f сек (библиотека chrono)", t.elapsed());

printf("\nЗатраченное время на просчет составляет: %d сек (библиотека time.h)\n", seconds);

printf("\nНажмите ENTER для выхода в главное меню... \n");

\_getch();

return 0;

}

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);

}

void ss(int n, int mass[])

{

int i, j, step;

int tmp;

for (step = n / 2; step > 0; step /= 2)

for (i = step; i < n; i++)

{

tmp = mass[i];

for (j = i; j >= step; j -= step)

{

if (tmp < mass[j - step])

mass[j] = mass[j - step];

else

break;

}

mass[j] = tmp;

}

}

int rand1()

{

int N;

printf("Введи количество элементов в массиве: ");

scanf\_s("%d", &N);

int\* mass;

mass = (int\*)malloc(N \* sizeof(int));

for (int i = 0; i < N; i++)

mass[i] = rand() % N + 1;

Timer t;

ss(N, mass);

printf("\nЗатраченное время на сортировку: %f сек (Шелла) ", t.elapsed());

Timer t1;

qs(mass, 0, N - 1);

printf("\nЗатраченное время на сортировку: %f сек (Быстрая) ", t1.elapsed());

\_getch();

return 0;

}

int up()

{

int a, N;

printf("Введи количество элементов в массиве: ");

scanf\_s("%d", &N);

a = 0;

int\* mass;

mass = (int\*)malloc(N \* sizeof(int));

for (int i = 0; i < N; i++)

mass[i] = a++;

Timer t;

ss(N, mass);

printf("\nЗатраченное время на сортировку: %f сек (Шелла) ", t.elapsed());

Timer t1;

qs(mass, 0, N - 1);

printf("\nЗатраченное время на сортировку: %f сек (Быстрая) ", t1.elapsed());

\_getch();

return 0;

}

int down()

{

int a, N;

printf("Введи количество элементов в массиве: ");

scanf\_s("%d", &N);

a = N;

int\* mass;

mass = (int\*)malloc(N \* sizeof(int));

for (int i = 0; i < N; i++)

mass[i] = a--;

Timer t;

ss(N, mass);

printf("\nЗатраченное время на сортировку: %f сек (Шелла) ", t.elapsed());

Timer t1;

qs(mass, 0, N - 1);

printf("\nЗатраченное время на сортировку: %f сек (Быстрая) ", t1.elapsed());

\_getch();

return 0;

}

int pyramid()

{

int a, N, M;

printf("Введи количество элементов в массиве: ");

scanf\_s("%d", &N);

a = 0;

int\* mass;

mass = (int\*)malloc(N \* sizeof(int));

M = N / 2;

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

mass[i] = a++;

}

for (int i = M; i < N; i++) {

mass[i] = a--;

}

/\*

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

printf("%d ", mass[i]);

}

\*/

Timer t;

ss(N, mass);

printf("\nЗатраченное время на сортировку: %f сек (Шелла) ", t.elapsed());

Timer t1;

qs(mass, 0, N - 1);

printf("\nЗатраченное время на сортировку: %f сек (Быстрая) ", t1.elapsed());

\_getch();

return 0;

}

int Menu(void)

{

system("CLS");

int c = 0;

setlocale(LC\_ALL, "Rus");

while (c < '0' || c > '5')

{

printf(

"::=================================::\n"

"|| 1 : Перемножение матриц ||\n"

"|| 2 : Рандомный массив ||\n"

"|| 3 : Возрастающий массив ||\n"

"|| 4 : Убывающий массив ||\n"

"|| 5 : Пирамидальный массив ||\n"

"::=================================::\n"

"> ");

c = \_getch();

system("cls");

}

return c;

}

void main(void)

{

int Select;

while ((Select = Menu()) != '0' && Select != 27)

switch (Select)

{

case '1':

matrix();

break;

case '2':

rand1();

break;

case '3':

up();

break;

case '4':

down();

break;

case '5':

pyramid();

break;

}

}