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

Вожжов И.В

Юдин Д.А

Анохин А.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.

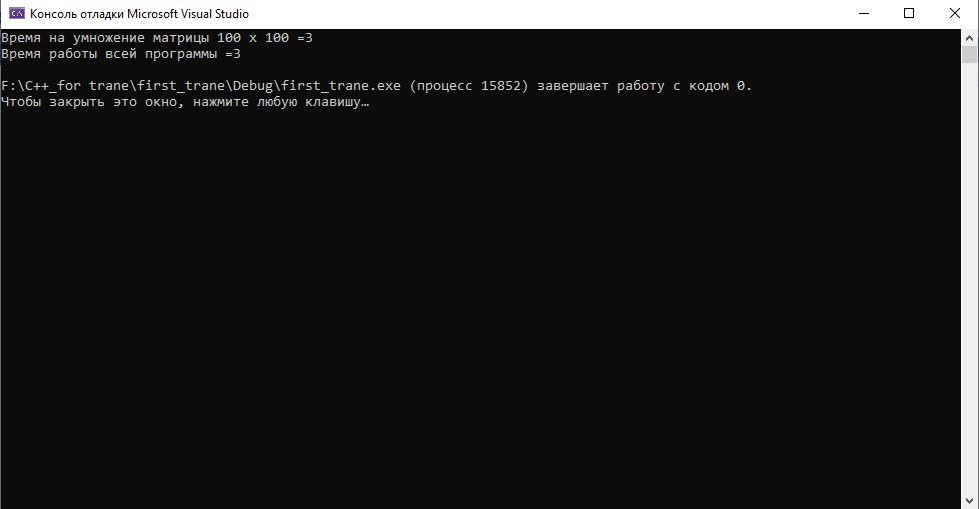
4. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

5. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

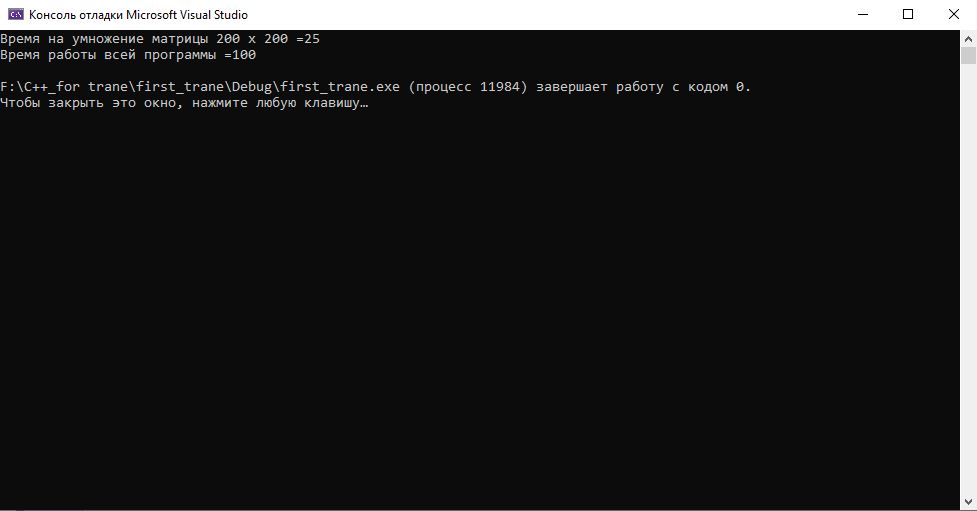
**Ход работы**

Для нахождения времени выполнения программы использовалась библиотека “chrono”

1)умножение матриц 100 на 100

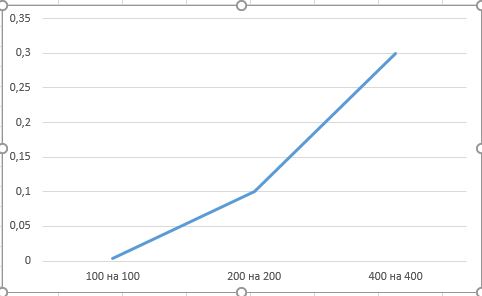


умножение матриц 200 на 200

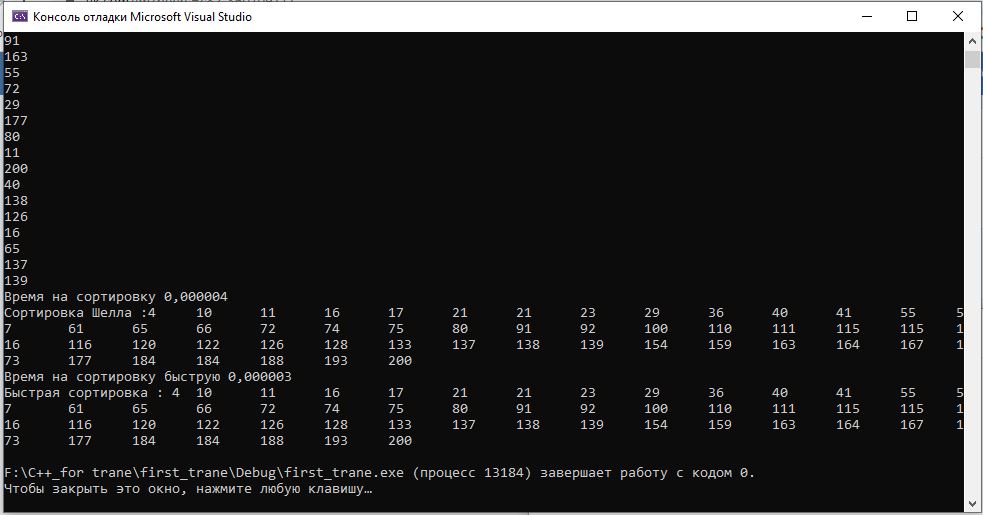


(результат представлен в миллисекундах)

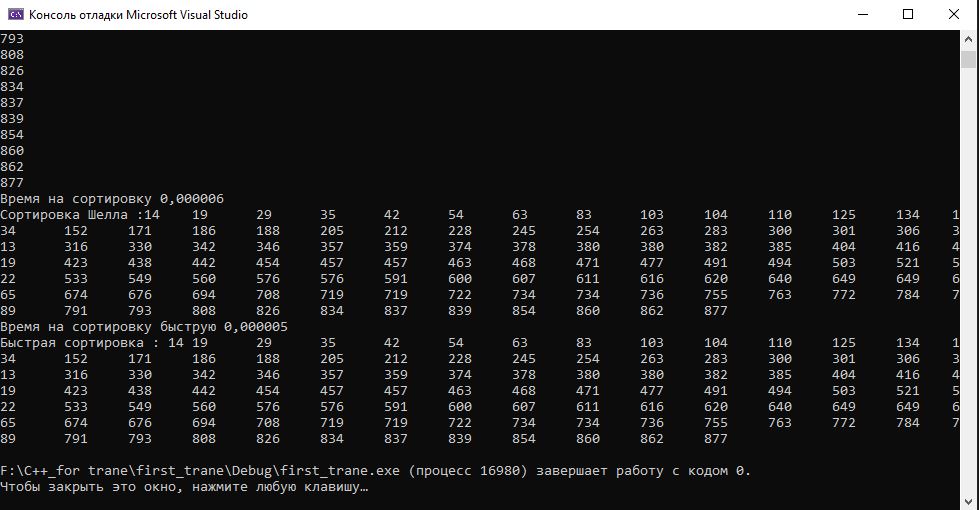
2) график зависимости времени работы от размера матрицы :



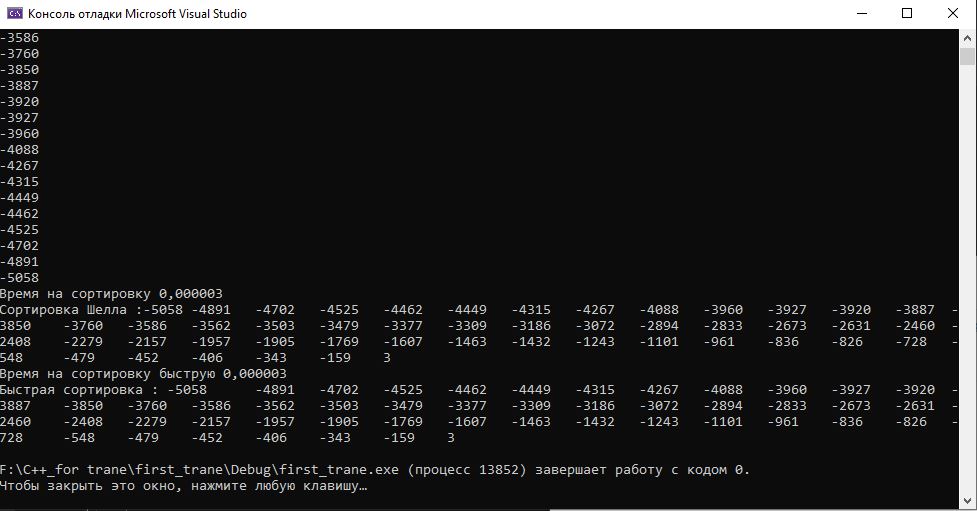
3) сортировка массива из случайных чисел :



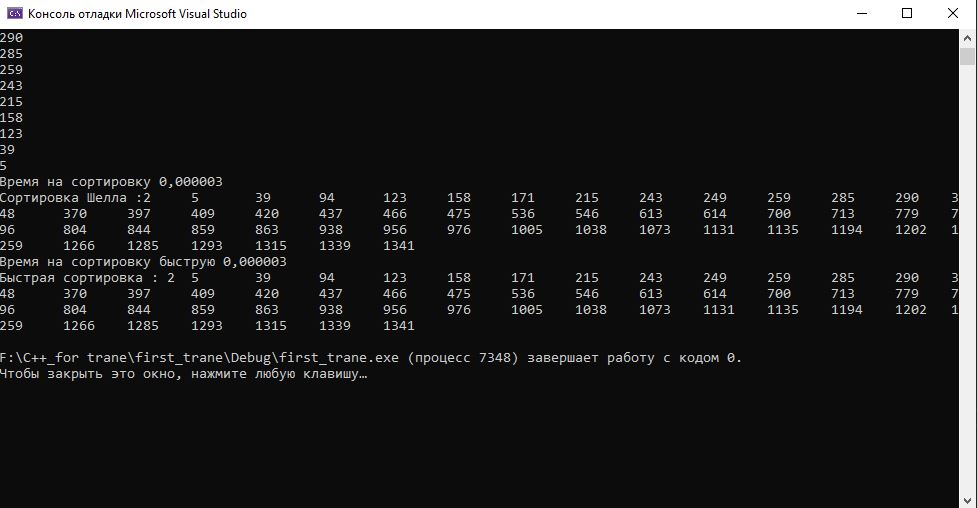
4) сортировка возрастающей последовательности



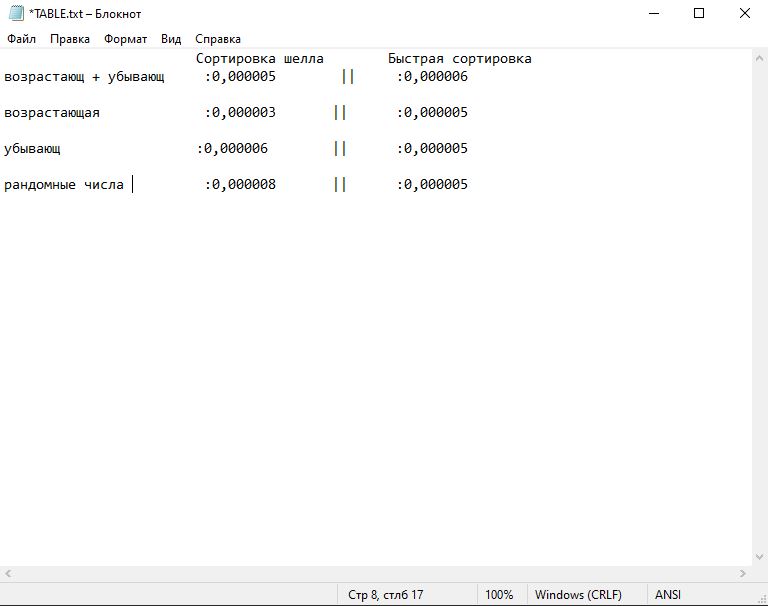
5) сортировка убывающей последовательности



6) сортировка массива, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.



Записали полученные данные в файл TABLE.txt и представили в виде таблицы:



Вывод:

Из данной лабораторной работы мы научились находить время выполнения программы. Разобрали разные варианты нахождения времени выполнения программы ,а также оценили сортировки Шелла и быструю в разных условиях . Также выяснилось ,что быстрая сортировка при убывающей и случайной последовательностями более эффективна чем сортировка Шелла. В свою очередь сортировка Шелла быстрее при возрастающей последовательности.

Листинг

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <iostream>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <ctime>

#include <conio.h>

#include <windows.h>

#include <chrono>

using namespace std;

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;

}

}

}

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();

}

};

void qs(int\* items, int left, int right) //вызов функции: qs(items, 0, count-1);

{

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);

}

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

int n;

//int a[400][400], b[400][400], c[400][400], elem\_c;

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

cout << "Введите размер массива : "; cin >> n; cout << endl;

int \*A = new int [n];

int m = n / 2;

int raz =1+ rand() % 10;

while (i < n)

{

A[i] =rand()%201;

cout << A[i]<<"\t"<< endl;

i++;

}

time (&start);

Timer t;

shell(A, n);

time (&end);

int search\_t = difftime(end,start);

//cout << "Время на сортировку Шелла: " << search\_t<< endl;

printf("Время на сортировку %f\n", t.elapsed());

cout << "Сортировка Шелла :";

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

cout << A[i] << "\t";

}

cout << endl;

Timer t1;

qs(A, 0, n-1);

//cout << "Время на Быструю сортировку: " <<t1.elapsed() << endl;

printf("Время на сортировку быструю %f\n", t1.elapsed());

cout << "Быстрая сортировка : ";

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

cout << A[i]<<"\t";

cout << endl;

/\* while (i < 400)

{

while (j < 400)

{

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 < 400)

{

while (j < 400)

{

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

j++;

}

i++;

}

unsigned int start\_time = clock();

for (i = 0; i < 400; i++)

{

for (j = 0; j < 400; j++)

{

elem\_c = 0;

for (r = 0; r < 400; r++)

{

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

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

}

}

}

unsigned int end\_time = clock();

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

unsigned int time1 = clock();

cout << "Время на умножение матрицы 400 х 400 =" << search\_time << "секунд" << std::endl;

cout << "Время работы всей программы =" <<time1 << std::endl;

cout << a;\*/

return(0);

}