Министерство 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»

Выполнили:

студент группы 20ВВ2

Александров В.С.

Кирюткин И.А.

Сафронов Д.В.

Принял:

к.т.н., доцент

Митрохин .

**Пенза 2021**

**Цель работы:** Создать программу и на её примере оценить время выполнения программы для разного набора данных.

**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.

**Листинг :**

#define \_CRT\_SECURE\_NO\_WARNINGS

#pragma warning(disable:4996)

#include <iostream>

#include <stdio.h>

#include <string.h>

#include <cstdlib>

#include <windows.h>

#include <conio.h>

#include <ctime>

#include <cmath>

//int comp(const void\*, const void\*);

int compare(const void\* x1, const void\* x2) // функция сравнения элементов массива

{

return (\*(int\*)x1 - \*(int\*)x2); // если результат вычитания равен 0, то числа равны, < 0: x1 < x2; > 0: x1 > x2

}

void shellSort(int\* num, int size)

{

int increment = 3;

while (increment > 0)

{

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

{

int j = i;

int temp = num[i];

while ((j >= increment) && (num[j - increment] > temp))

{

num[j] = num[j - increment];

j = j - increment;

}

num[j] = temp;

}

if (increment > 1)

increment = increment / 2;

else if (increment == 1)

break;

}

}

void quickSort(int\* numbers, int left, int right)

{

int pivot;

int l\_hold = left;

int r\_hold = right;

pivot = numbers[left];

while (left < right)

{

while ((numbers[right] >= pivot) && (left < right))

right--;

if (left != right)

{

numbers[left] = numbers[right];

left++;

}

while ((numbers[left] <= pivot) && (left < right))

left++;

if (left != right)

{

numbers[right] = numbers[left];

right--;

}

}

numbers[left] = pivot;

pivot = left;

left = l\_hold;

right = r\_hold;

if (left < pivot)

quickSort(numbers, left, pivot - 1);

if (right > pivot)

quickSort(numbers, pivot + 1, right);

}

int main(void)

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

clock\_t start, stop;

start = clock();

int i = 0, j = 0, d, mass;// , \*\* a, \*\* b, \*\* c, mat\_znach;

printf("Введите размерность матриц: ");

std::cin >> mass;

typedef int Cell;

typedef Cell\* TCell;

TCell a = (TCell)malloc(mass \* mass \* sizeof(Cell));

TCell b = (TCell)malloc(mass \* mass \* sizeof(Cell));

TCell c = (TCell)malloc(mass \* mass \* sizeof(Cell));

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

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

a[i \* mass + j] = rand() % 10;

b[i \* mass + j] = rand() % 10;

c[i \* mass + j] = 0;

}

}

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

{

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

{

for (d = 0; d < mass; d++)

{

c[i\*mass+j] += a[i \* mass + d] \* b[d\*mass+j];

}

}

}

stop = clock();

printf("\nВремя выполнения умножения: %d\n", (stop - start) / CLK\_TCK);

FILE\* file;

file = fopen("data.txt", "w");

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

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

fprintf(file, "%5d", c[i \* mass + j]);

}

fprintf(file, "\n");

}

/////////////////////////////////////////////////////////////////////////////////////////////////////

int \*m, \*copy, znach, y=10000;

printf("\nВведите размер массива для сортирок: ");

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

m = (int\*)malloc(znach \* sizeof(int));

copy = (int\*)malloc(znach \* sizeof(int));

int nil = znach / 2, znaf=20;

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

{

//m[i] = i + 10;

//m[i] = y - 10;

//m[i] = rand() % 100;

if (i <= nil-1) {

m[i] = i + 10;

znaf = m[i];

}

else {

m[i] = znaf - 1;

znaf -= 1;

}

copy[i] = m[i];

}

start = clock();

shellSort(m, znach);

stop = clock();

fprintf(file, "\n");

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

{

fprintf(file, "%5d", m[i]);

}

printf("\nВремя выполнения shellSort: %d\n", (stop - start));

fprintf(file, "\nВремя выполнения shellSort: %d\n", (stop - start));

/////////////////////////////////////////////////////////////////////////////////////////////////////

int \*p;

p = (int\*)malloc(znach \* sizeof(int));

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

p[i] = copy[i];

}

start = clock();

quickSort(p, 0, znach - 1);

stop = clock();

fprintf(file, "\n");

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

{

fprintf(file, "%5d", p[i]);

}

printf("\nВремя выполнения quickSort: %d\n", (stop - start) );

fprintf(file, "\nВремя выполнения quickSort: %d\n", (stop - start));

/////////////////////////////////////////////////////////////////////////////////////////////////////

start = clock();

qsort(copy, znach, sizeof(int), compare);

stop = clock();

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

printf("%3d", copy[i]);

}

fprintf(file, "\n");

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

{

fprintf(file, "%5d", copy[i]);

}

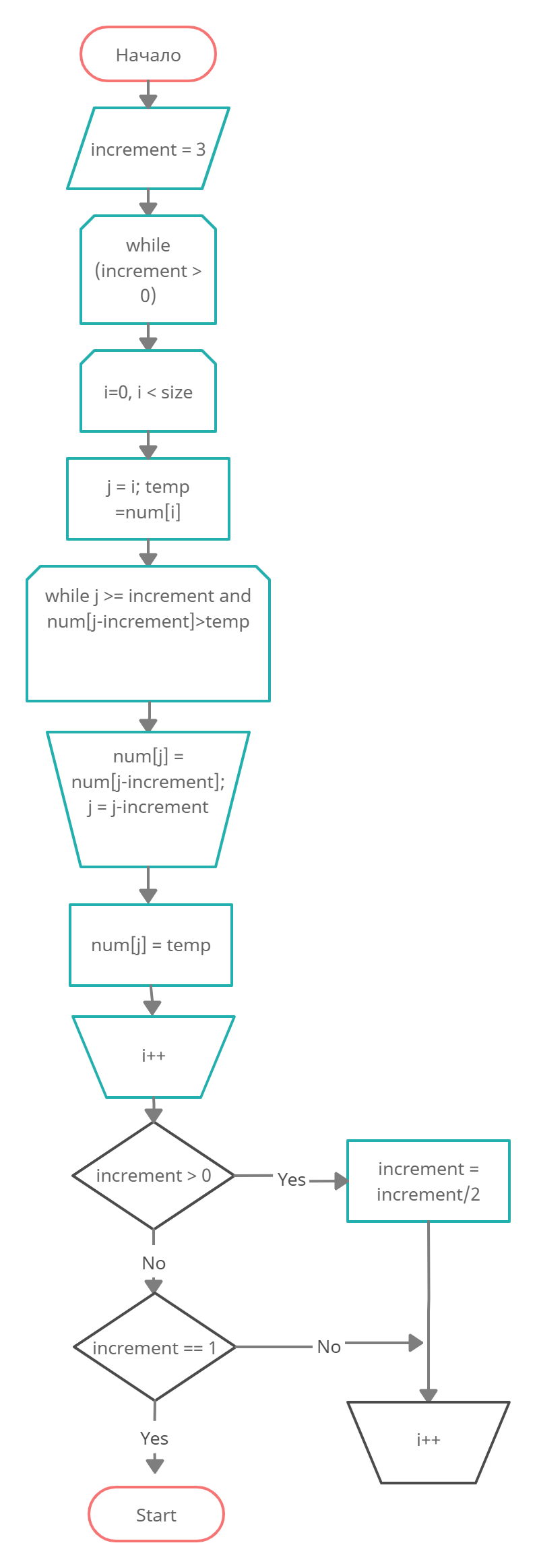
printf("\n Время выполнения qsort(): %d", (stop - start));

fprintf(file, "\n Время выполнения qsort(): %d\n", (stop - start));

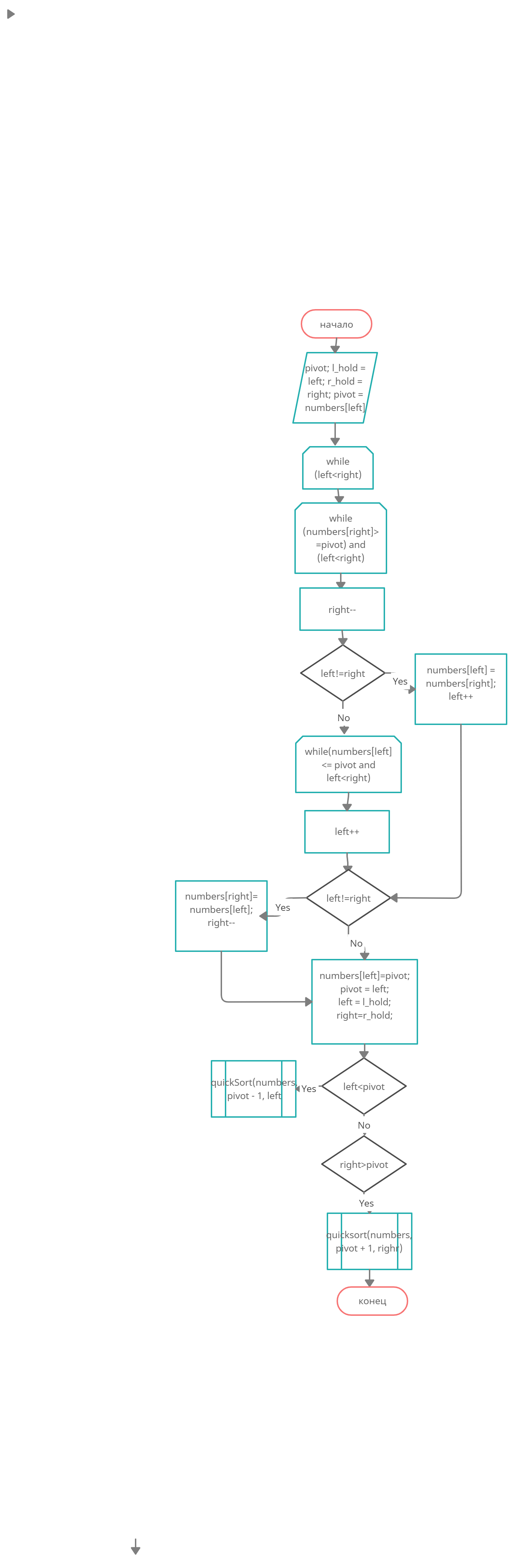
fclose(file);

return(0);

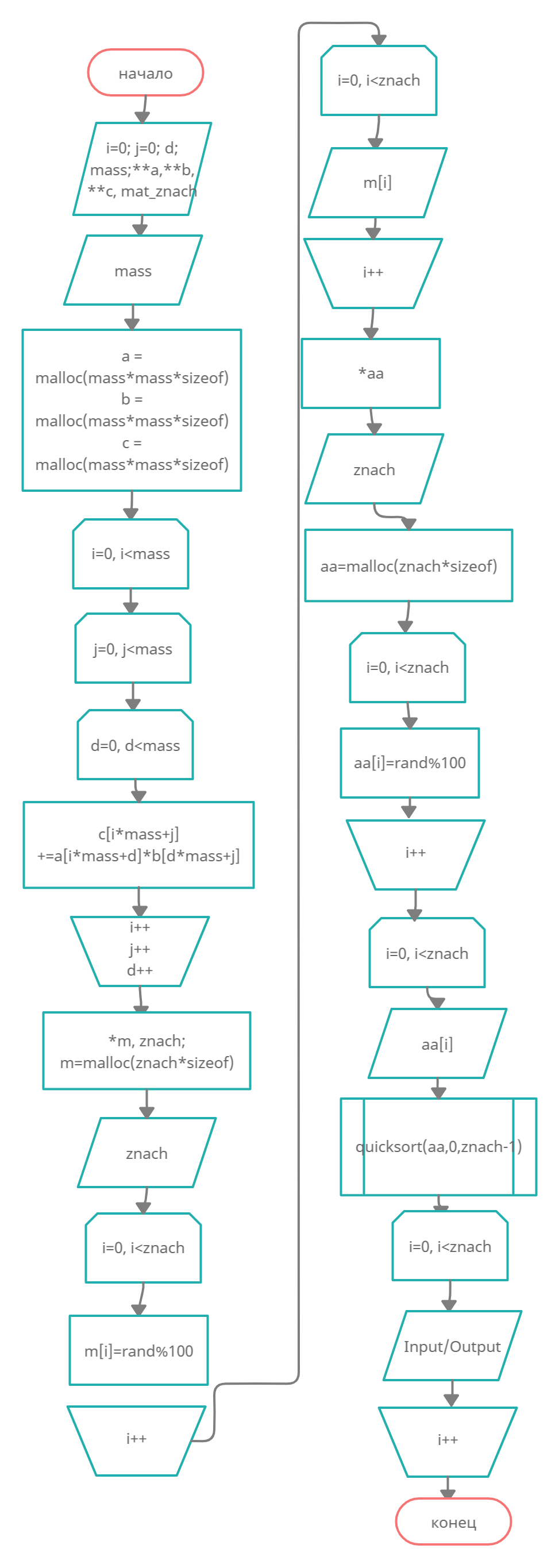
* }  
  **Блок-схема блока shellsort:**



**Блок-схема блока qs:**



**Блок-схема блока main:**



**Зависимость времени выполнения от объёма данных:**

****

Изменение сложности алгоритма в зависимости от входных данных

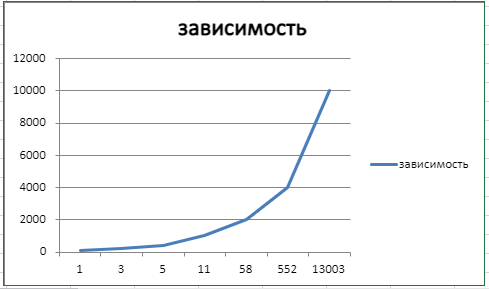
Провели тестирование алгоритмов на разливных последовательностях данных



Провели дополнительный тест для сравнения **shellSort** и **QS\_lib** на большей размерности массива



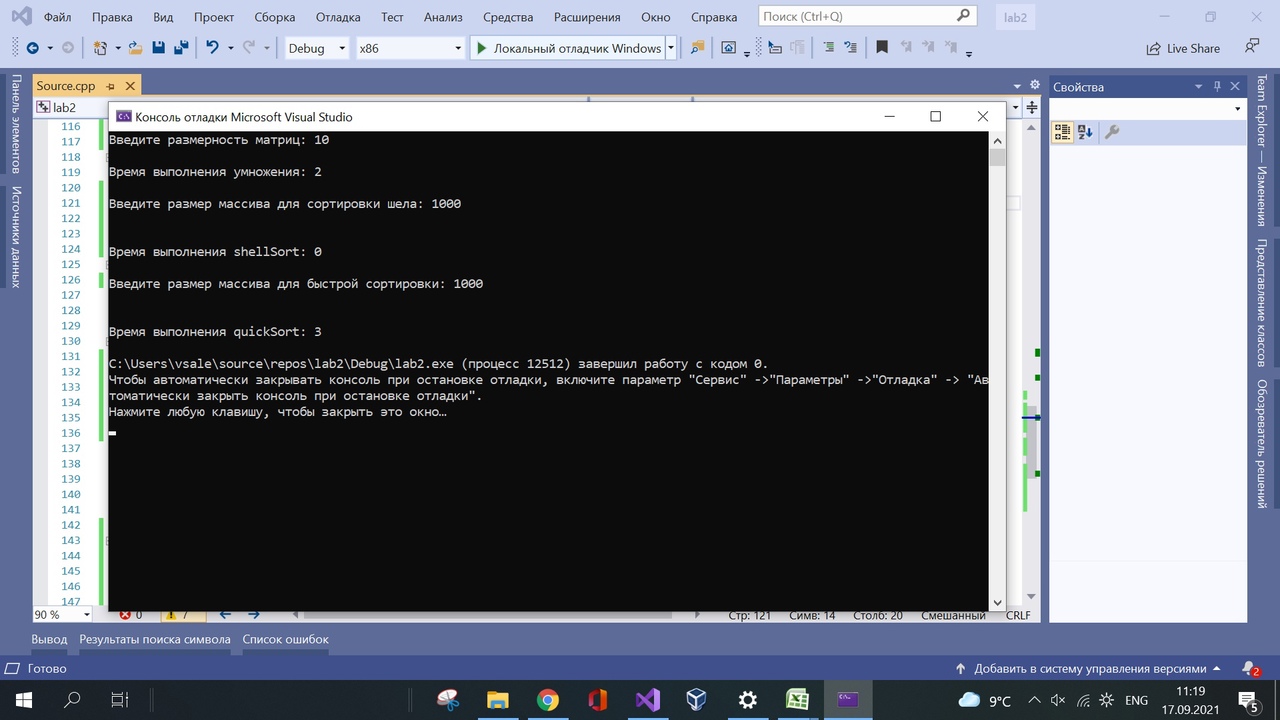
* На возрастающей последовательности эффективнее работает **shellSort** чем **QS\_lib** и **QS\_code**
* На убывающей последовательности эффективнее работает **shellSort и QS\_lib** и **QS\_code**
* На случайной последовательности эффективнее работает **QS\_code** чем **shellSort** и **QS\_lib**



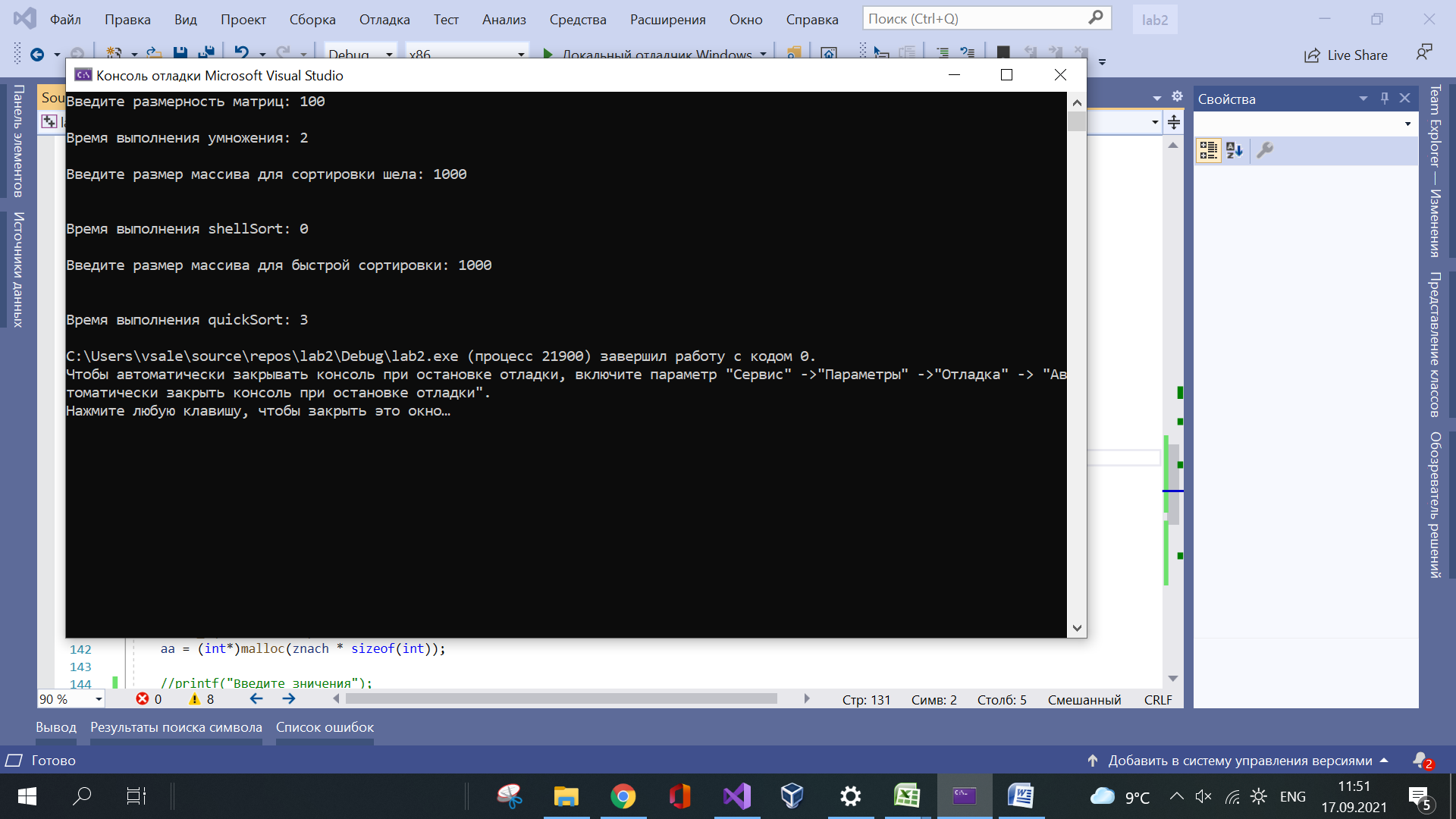
Зависимость время работы программы от входных данных

**Результаты программы :**

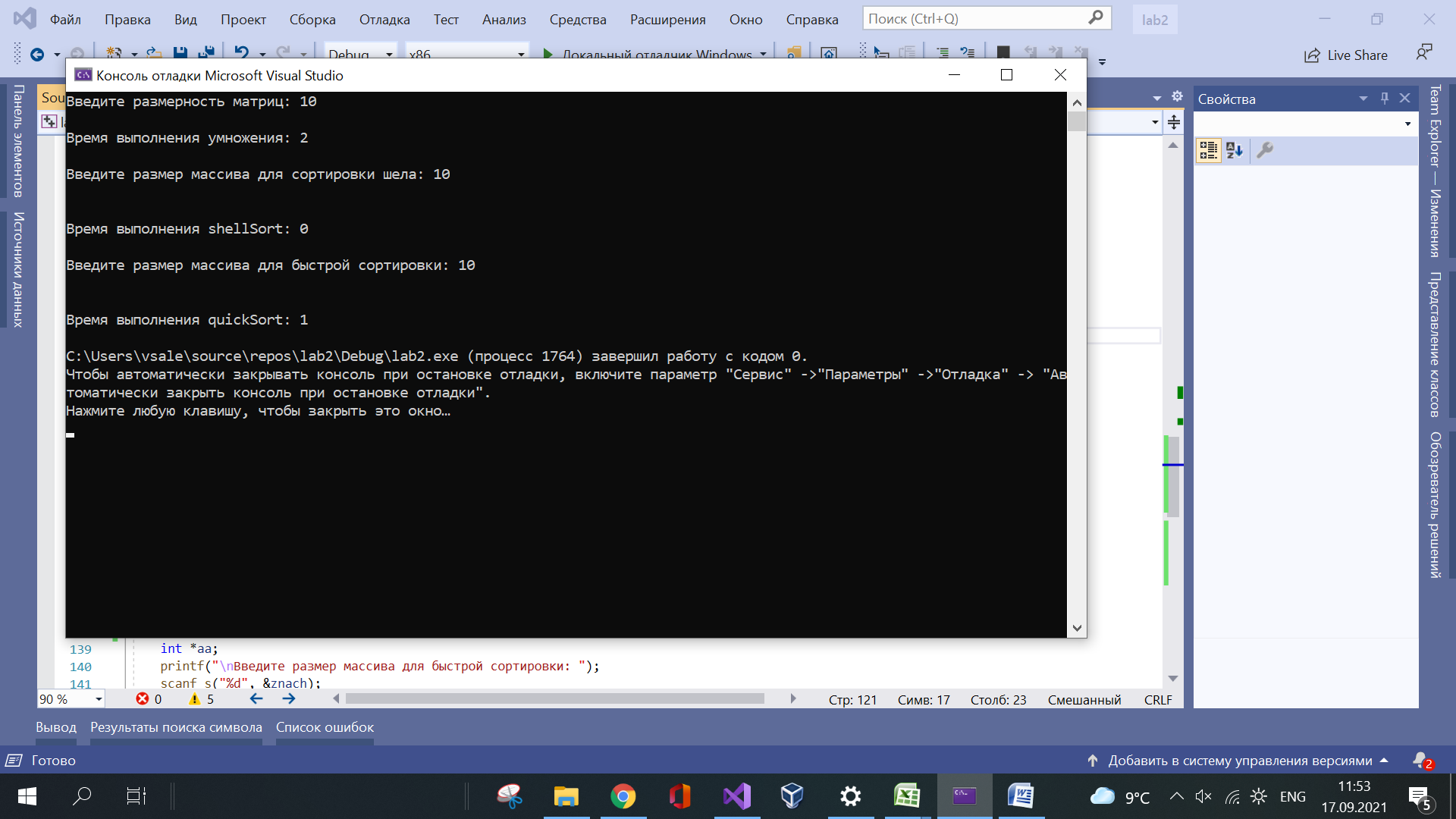
Возрастающая последовательность.



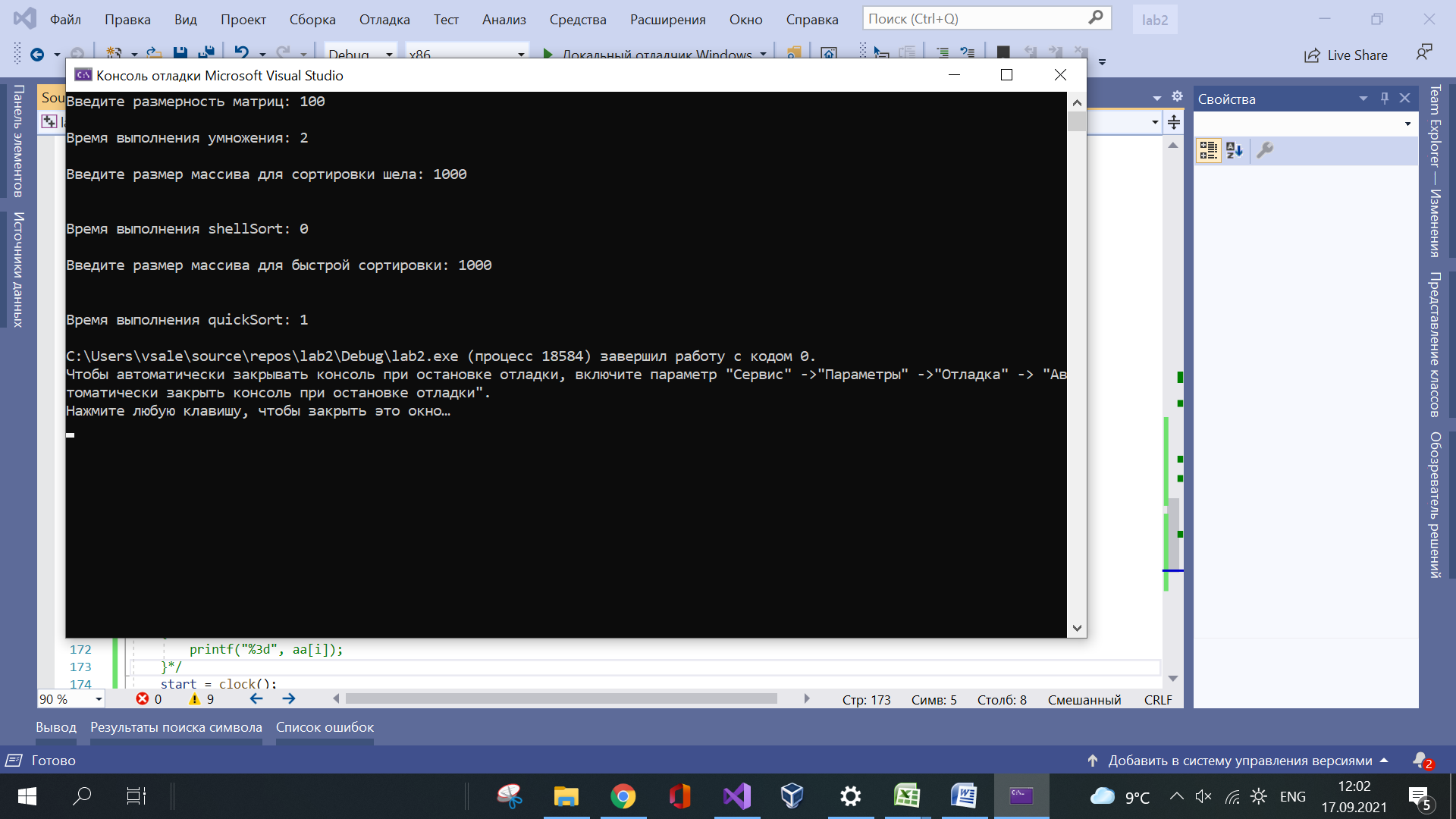
Случайная последовательность



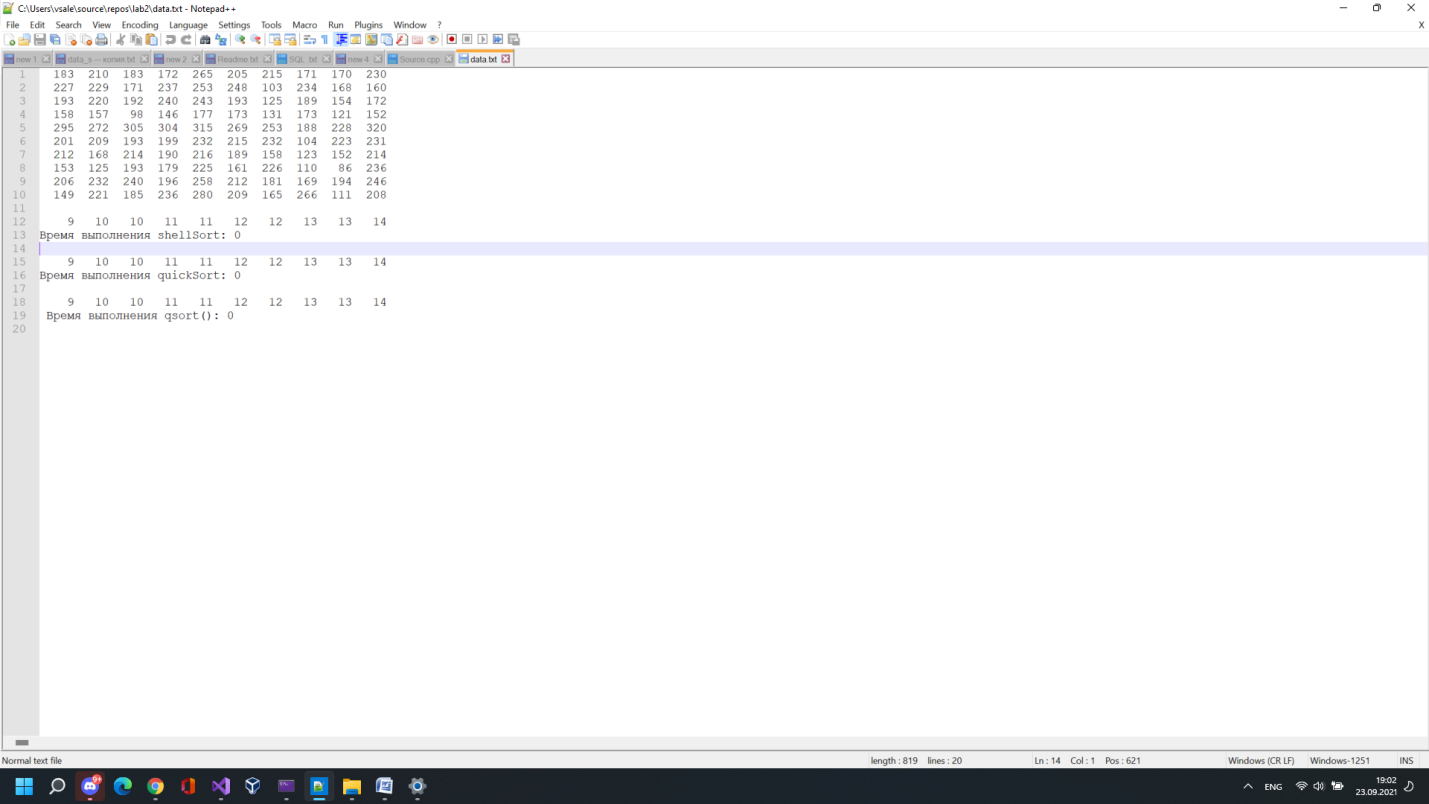
Возрастающая последовательность



Убывающая последовательность



Одна половина последовательности представляет собой возрастающую последовательность чисел, а вторая, – убывающую.



В ходе выполнения программы результаты сохраняются в файл

**Вывод:** напасли программу и на её примере оценили время выполнения алгоритма для разного набора данных.