Министерство образования и науки Российской Федерации

Пензенский государственный университет

Факультет вычислительной техники

**Отчёт**

По лабораторной работе №2

По курсу "ЛиОАвИЗ"

Выполнили:

Студент группы 20ВВ1

Репин И.В.

Приняла:

Юрова О.В.

Пенза 2021

**Название:** Оценка времени выполнения программ

**Цель работы:** выполнить ряд заданий

**Лабораторное задание:**

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, 120, 140, 180, 200.

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 задачи:**

Задание 1.

1. Оценивая порядок сложности алгоритма, необходимо использовать только ту часть, которая возрастает быстрее всего. В нашем случае рабочий цикл описывается выражением N^2+N^2+N^3, таким образом порядок сложности O(N^3)
2. В начале пишем start = clock(), в конце end = clock(), разница – время затраченное на работу программы (результаты смотреть в п. Результаты работы программы)
3. График зависимости времени работы от объема данных

Вывод: полученные данные подтверждают достоверность нахождения порядка сложности программы

Задание 2.

1. Сгенерировали следующие массивы: со случайными значениями, с возрастающей последовательностью, с убывающей последовательностью, с возрастающей и убывающей последовательностью.
2. Передали поочередно значения каждого их этих массивов в массив sortarray[] и отсортировали его каждым из 2 способов.

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>;

#include <stdlib.h>;

#include <time.h>;

void shell(int\* items, int count);

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

void shell(int\* items, int count)

{

printf("\nArray before sort:\n");

for (int k = 0; k < count; k++) {

printf("%i ", items[k]);

}

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

start = clock();

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;

}

}

end = clock();

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

printf("\nArray after sort:\n");

for (int k = 0; k < count; k++) {

printf("%i ", items[k]);

}

printf("\nShell complited in %f seconds", time\_spent);

}

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

{

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

start = clock();

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){

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

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

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

start = clock();

int i = 0, j = 0, r;

int a[200][200], b[200][200], c[200][200], elem\_c, randarray[10], highterarray[10], lowerarray[10], combarray[10], sortarray[10];

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

while (i < 200)

{

while (j < 200)

{

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 < 200)

{

while (j < 200)

{

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

j++;

}

i++;

}

for (i = 0; i < 200; i++)

{

for (j = 0; j < 200; j++)

{

elem\_c = 0;

for (r = 0; r < 200; r++)

{

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

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

}

}

}

end = clock();

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

printf("Complited in %f seconds", time\_spent);

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

//random array

for (i = 0; i < 10; i++)

{

randarray[i] = rand() % 10 + 1;

}

//возрастающий массив

for (i = 0; i < 10; i++)

{

highterarray[i] = i;

}

//убывающий массив

for (i = 0; i < 10; i++)

{

lowerarray[i] = 10-i;

}

//комбинированный массив

for (i = 0; i < 10; i++)

{

if (i < 5) {

combarray[i] = i;

}

else {

combarray[i] = 10-i;

}

}

printf("\n\nRandom array:");

for (i = 0; i < 10; i++)

{

sortarray[i] = randarray[i];

}

shell(sortarray, 10);

for (i = 0; i < 10; i++)

{

sortarray[i] = randarray[i];

}

printf("\nArray before sort:\n");

for (int k = 0; k < 10; k++) {

printf("%i ", sortarray[k]);

}

start = clock();

qs(sortarray, 0, 9);

end = clock();

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

printf("\nArray after sort:\n");

for (int k = 0; k < 10; k++) {

printf("%i ", sortarray[k]);

}

printf("\nQsort complited in %f seconds", time\_spent);

printf("\n\nhighterarray:");

for (i = 0; i < 10; i++)

{

sortarray[i] = highterarray[i];

}

shell(sortarray, 10);

for (i = 0; i < 10; i++)

{

sortarray[i] = highterarray[i];

}

start = clock();

qs(sortarray, 0, 9);

end = clock();

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

printf("\nArray after sort:\n");

for (int k = 0; k < 10; k++) {

printf("%i ", sortarray[k]);

}

printf("\nQsort complited in %f seconds", time\_spent);

printf("\n\nlowerarray:");

for (i = 0; i < 10; i++)

{

sortarray[i] = lowerarray[i];

}

shell(sortarray, 10);

for (i = 0; i < 10; i++)

{

sortarray[i] = lowerarray[i];

}

start = clock();

qs(sortarray, 0, 9);

end = clock();

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

printf("\nArray after sort:\n");

for (int k = 0; k < 10; k++) {

printf("%i ", sortarray[k]);

}

printf("\nQsort complited in %f seconds", time\_spent);

printf("\n\ncombarray:");

for (i = 0; i < 10; i++)

{

sortarray[i] = combarray[i];

}

shell(sortarray, 10);

for (i = 0; i < 10; i++)

{

sortarray[i] = combarray[i];

}

start = clock();

qs(sortarray, 0, 9);

end = clock();

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

printf("\nArray after sort:\n");

for (int k = 0; k < 10; k++) {

printf("%i ", sortarray[k]);

}

printf("\nQsort complited in %f seconds", time\_spent);

return(0);

}

**Результаты работы программы:**

1. Задание:

Результаты работы программы представлен на рисунках 1, 2, 3, 4, 5

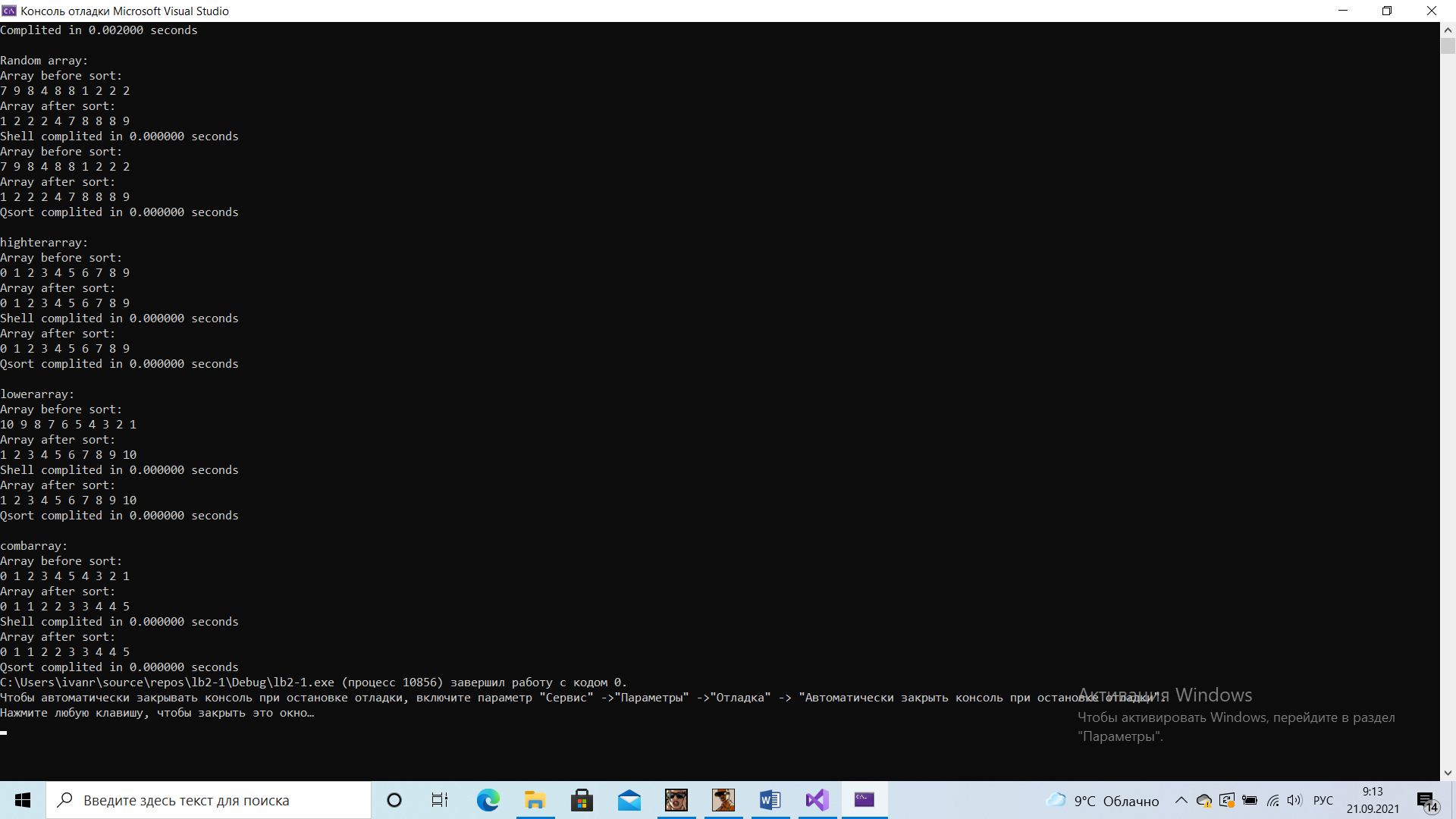


Рисунок 1.

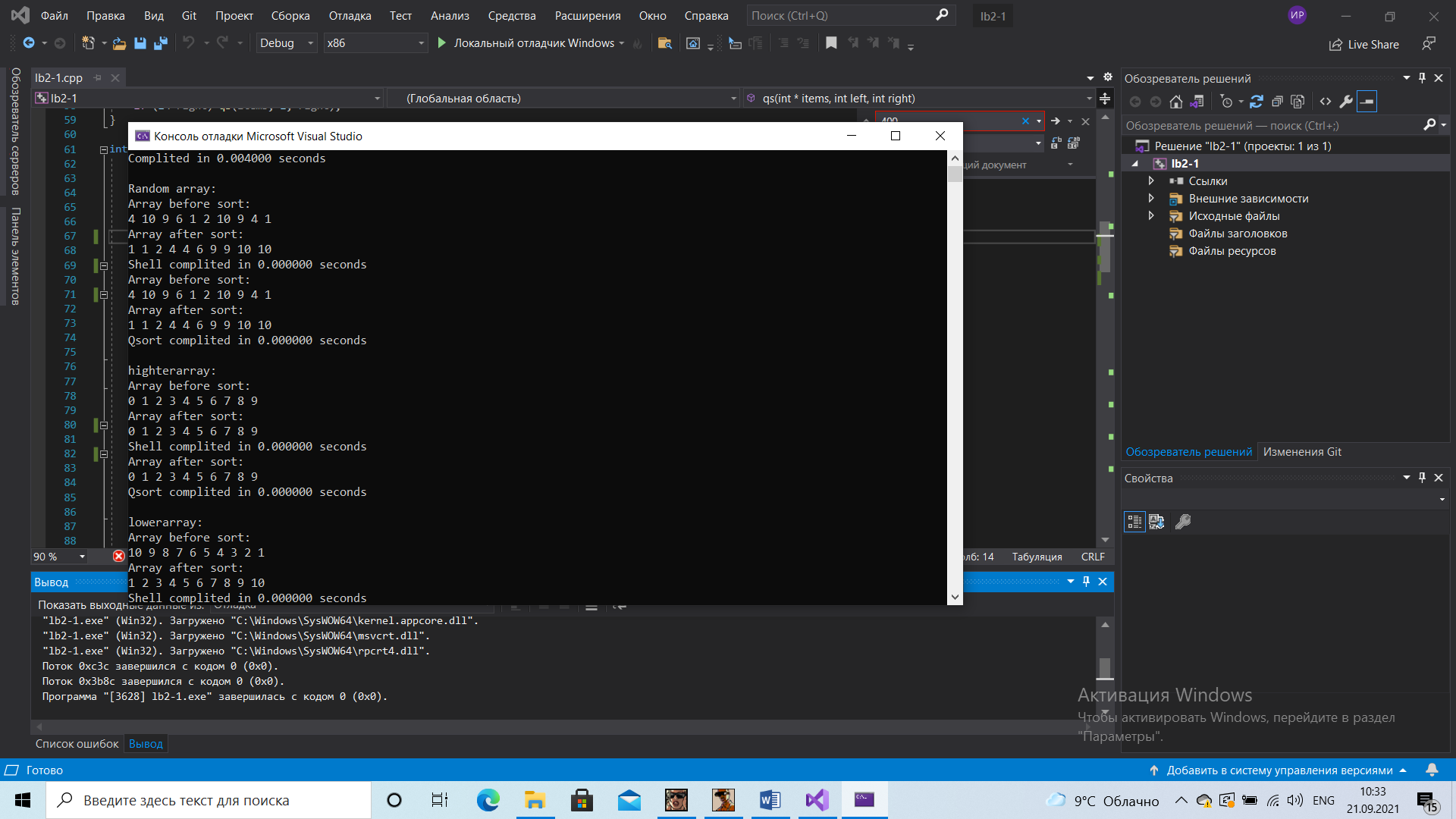


Рисунок 2.

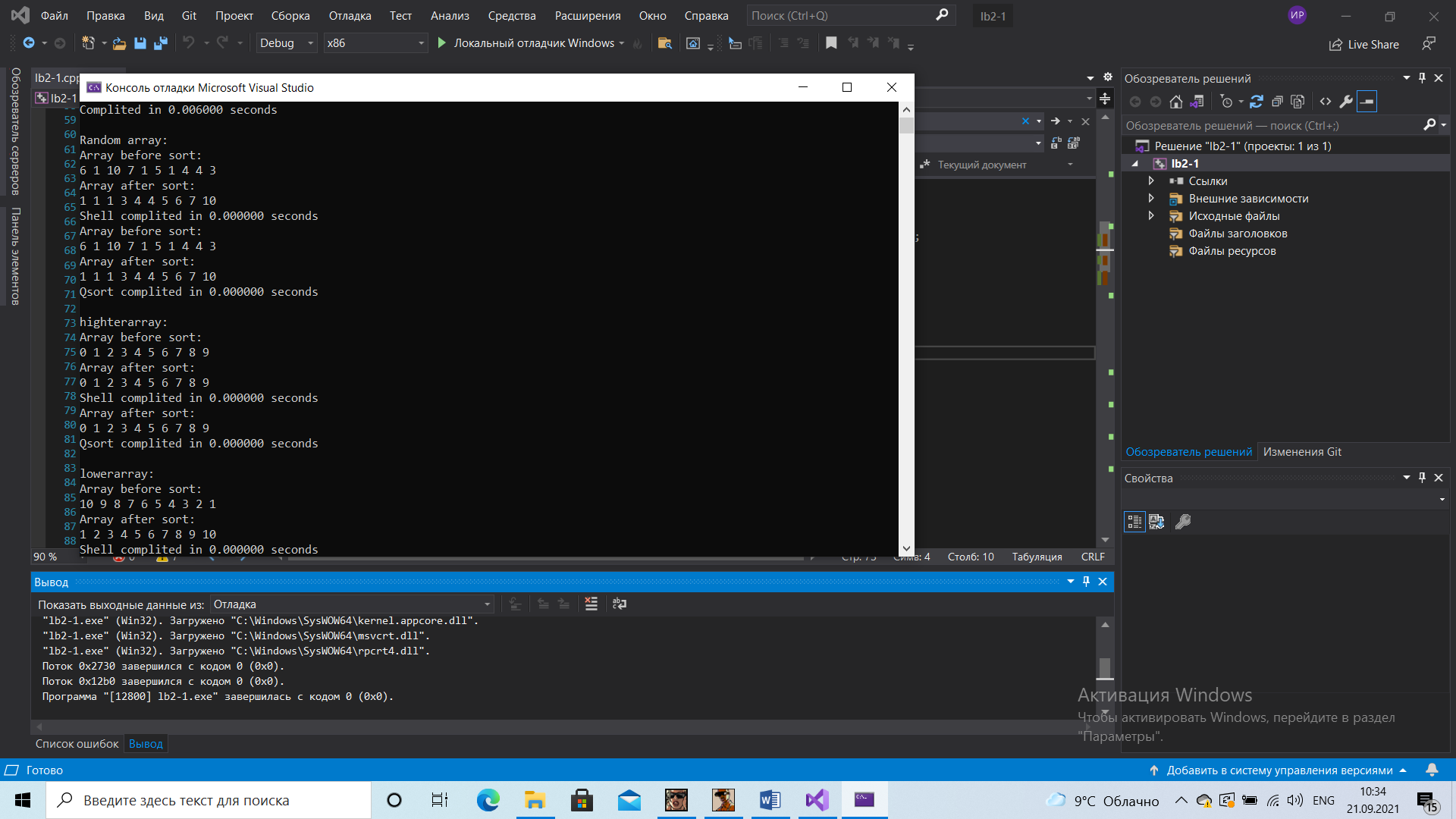


Рисунок 3.

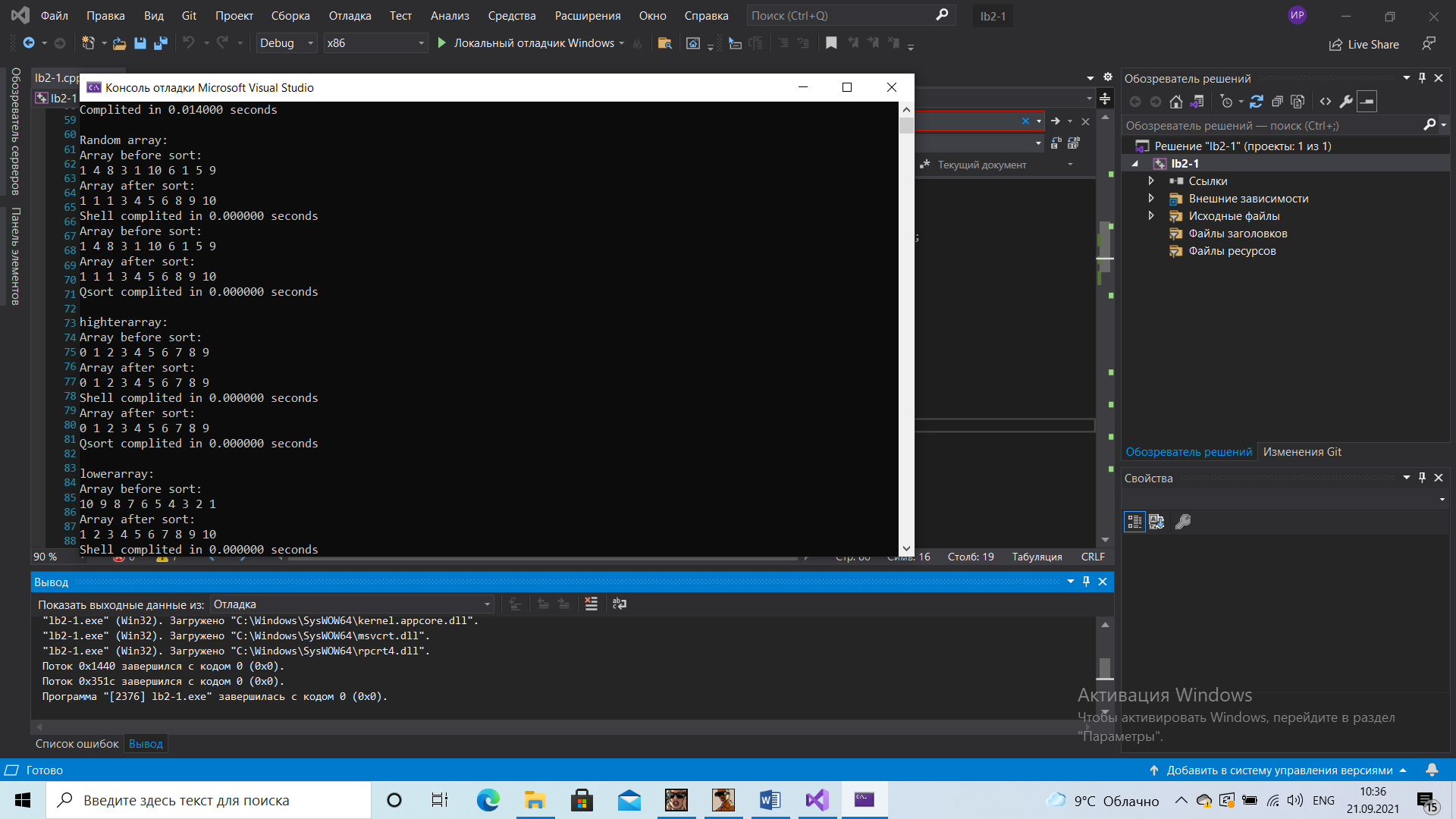


Рисунок 4.

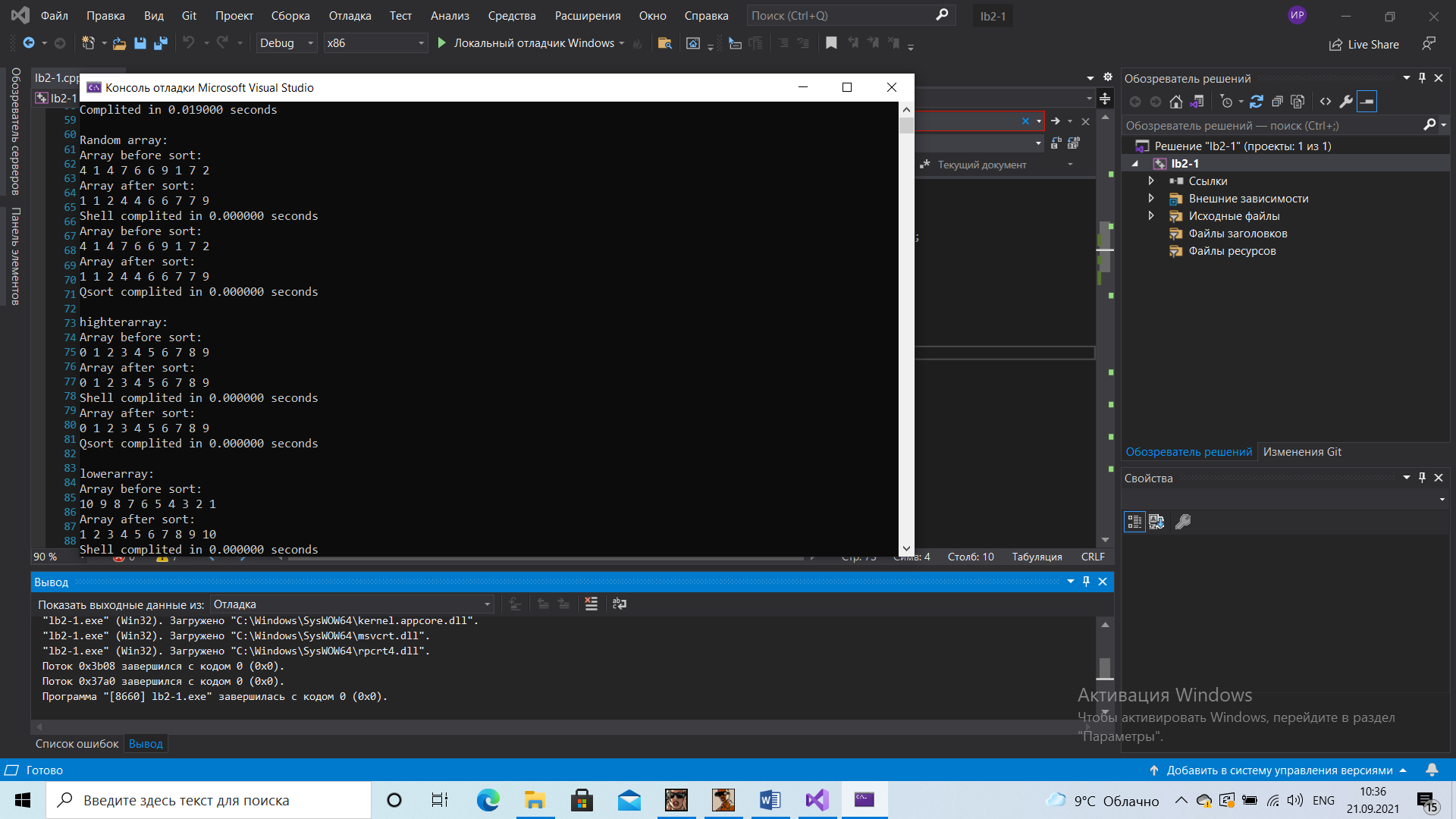
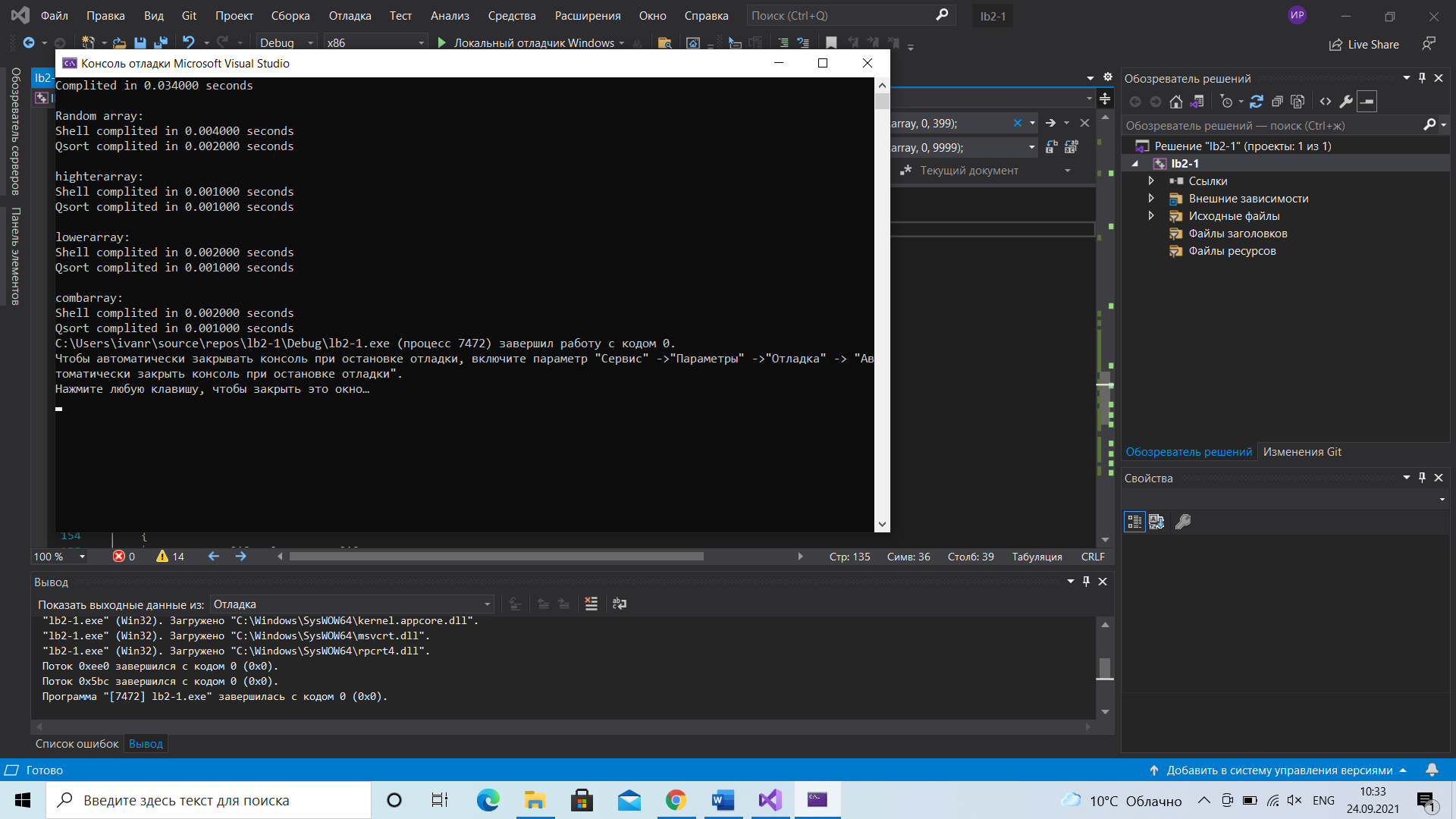


Рисунок 5.

2 Задание:

Результат работы программы представлен на рисунке 6



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайные | Возрастающие | Убывающие | Комбинированный |
| Shell (ms) | 0.004 | 0.001 | 0.002 | 0.002 |
| Qsort (ms) | 0.002 | 0.001 | 0.001 | 0.001 |

**Вывод:**

При работе с набором случайных, убывающих или сначало возрастающих, а потом убывающих эффективнее использовать Qsort

При работе с возрастающим набором чисел одинаково эффективны оба алгоритма