Пензенский государственный университет Кафедра «Вычислительная техника»

**Отчет**о выполнении лабораторной работы №2 по дисциплине “Логика и основа алгоритмизации в инженерных задачах” на тему**:**

**Оценка времени выполнения программ**

Выполнил студент гр. 19ВВ4:   
Шахов К.М.

Проверили:

Юрова О.В.  
Митрохин М. А.

Пенза, 2020 г.

**Название**Оценка времени выполнения программ.

**Цель работы**Оценить время выполнения различных алгоритмов.

**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, 200,

400, 1000, 2000, 4000, 10000.

3. Построить график зависимости времени выполнения программы от размера матриц

и сравнить полученный результат с теоретической оценкой.

**Код программы**

Порядок сложности программы(О-символика) в данном случае будет высчитываться по формуле : **O(n3)**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <iostream>

#include <locale.h>

using namespace std;

int CLOCK\_PER\_MS = CLOCKS\_PER\_SEC / 1000;

clock\_t start, stop;

int\*\* fillMatrix(int size) {

int i = 0, j = 0;

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

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

a[k] = (int\*)malloc(size \* sizeof(int));

}

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

while (i < size)

{

while (j < size)

{

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

j++;

}

i++;

}

return a;

}

int\*\* matrix\_multiplication(int\*\* a, int\*\* b, int size)

{

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

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

int i = 0, j = 0, r;

int elem\_c;

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

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

c[k] = (int\*)malloc(size \* sizeof(int));

}

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

return(c);

}

void main() {

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

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

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

setlocale(LC\_ALL, "Rus");

int\*\* a;

int\*\* b;

int\*\* c;

int sizes[] = { 100, 200, 250, 300, 350, 400};

int times[6] = { 0 };

printf("Время выполнения операции (в мс) в зависимости от размера:\n");

for (int i = 0; i < 6; i++) {

a = fillMatrix(sizes[i]);

b = fillMatrix(sizes[i]);

start = clock();

c = matrix\_multiplication(a, b, sizes[i]);

stop = clock();

times[i] = (stop - start) / CLOCK\_PER\_MS;

printf("%d элементов - %d ms\n", sizes[i], times[i]);

for (int i = 0; i < 4; i++)

{

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

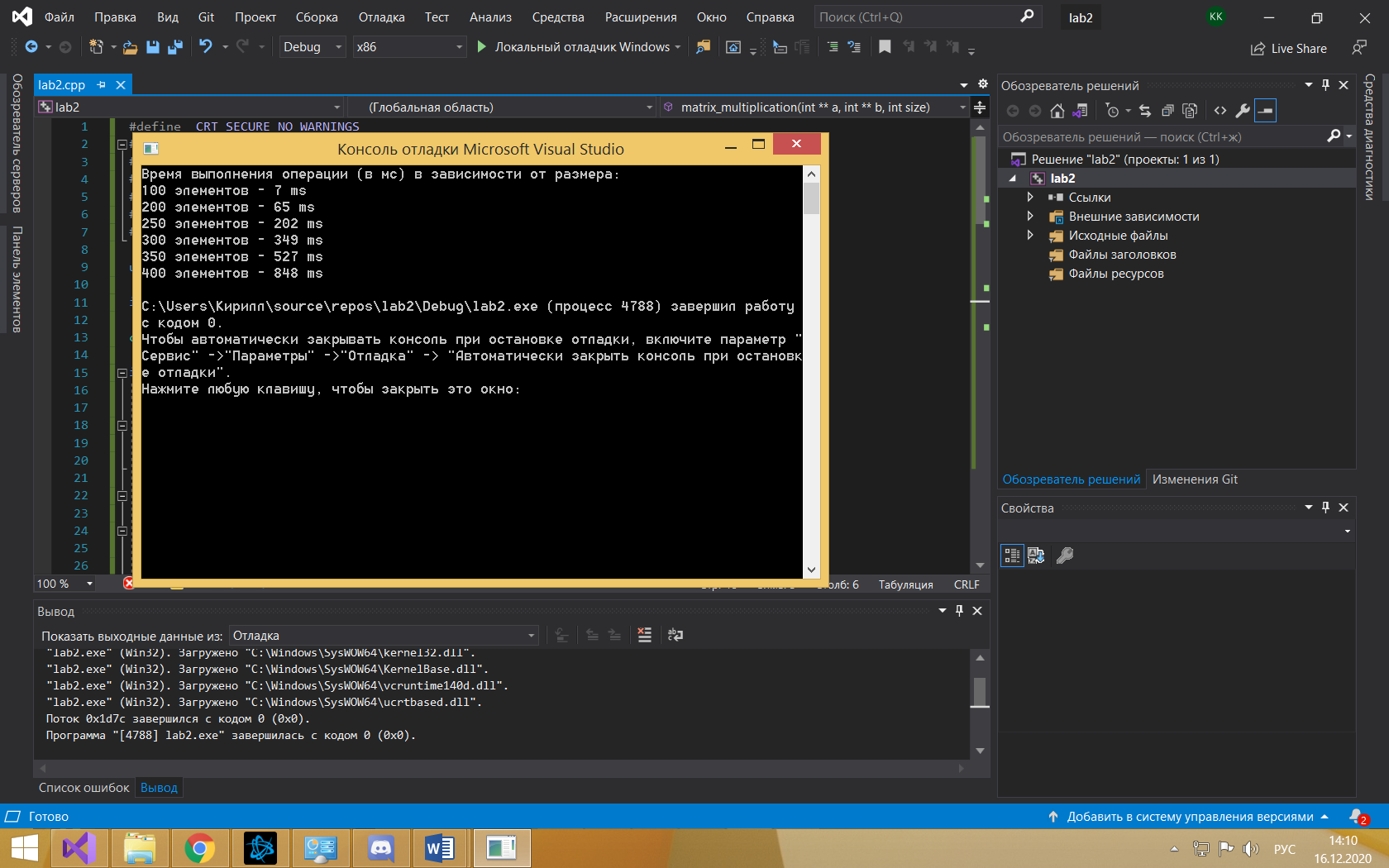
free(b);

free(c);

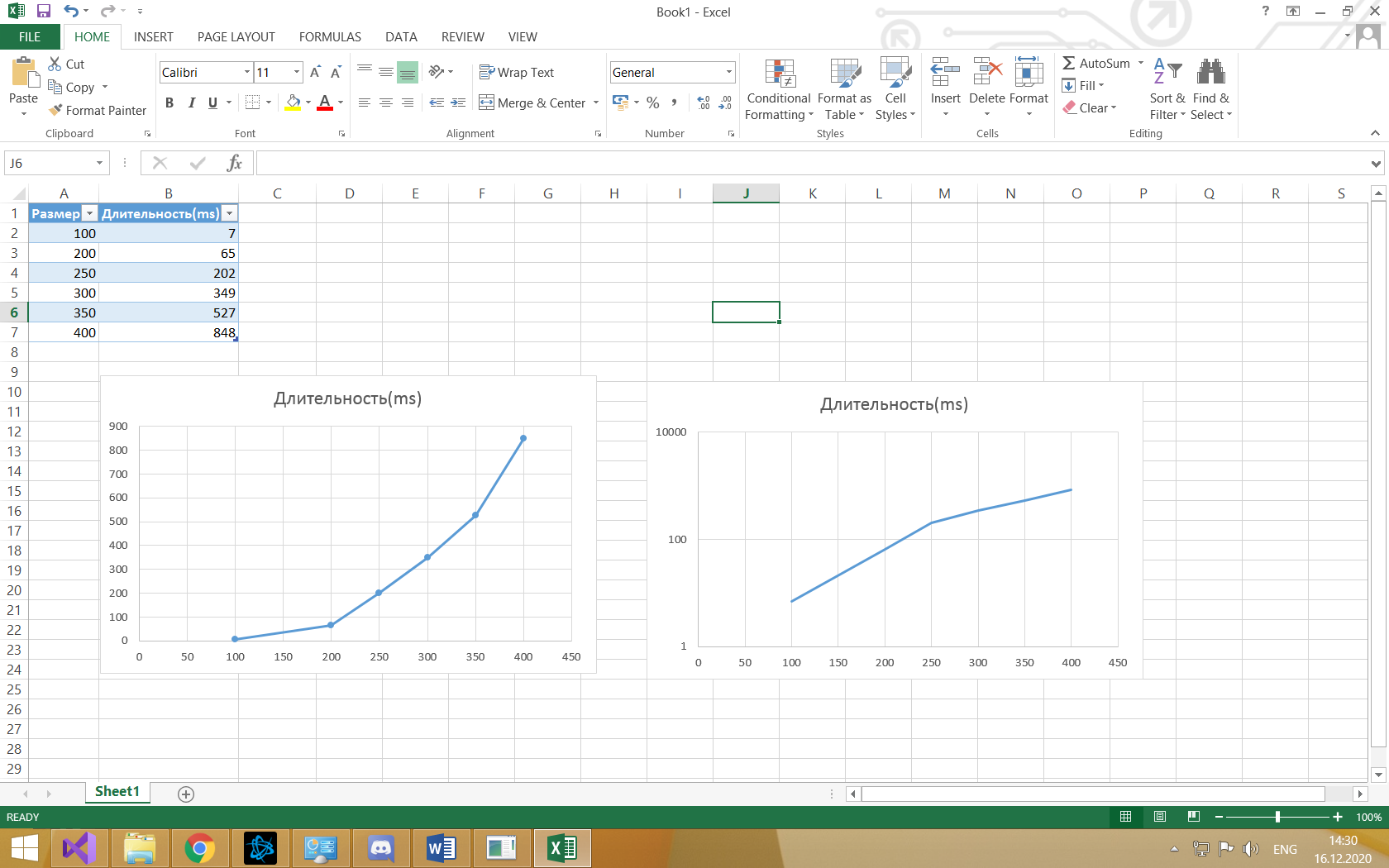
}

}

**Результаты работы программы**



**Рисунок 1 – результат выполнения задания 1**



**Рисунок 2 – графики зависимости размера от длительности**

**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.

**Код программы**

Lab 2:  
  
#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <iostream>

#include <locale.h>

using namespace std;

clock\_t start, stop;

int CLOCK\_PER\_MS = CLOCKS\_PER\_SEC / 1000;

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;

}

}

}

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

{

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

}

int\* copy\_list(int\* arr, int count) {

int\* new\_list = (int\*)malloc(count \* (sizeof(int)));

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

new\_list[i] = arr[i];

}

return new\_list;

}

void del\_list(int\* arr) {

free(arr);

}

void fillArray(int\*\* arr, int count) {

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

arr[0][i] = -1000 + rand() % 2000;

}

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

arr[1][i] = i \* 2 + 14;

}

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

arr[2][i] = arr[1][count - i];

}

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

switch (i / 2) {

case 0:

arr[3][i] = i \* 2 + 14;

break;

case 1:

arr[3][count - i - 1] = i \* 2 + 14;

break;

default:

break;

}

}

}

void sorting\_set(int\*\* set, int count, FILE\* file) {

int\* spisok;

for (int i = 0; i < 4; i++) {

printf(".");

fprintf(file, "%d. ", i + 1);

spisok = copy\_list(set[i], count);

start = clock() / CLOCK\_PER\_MS;

shell(spisok, count);

stop = clock() / CLOCK\_PER\_MS;

del\_list(spisok);

fprintf(file, "%dms ", (stop - start));

spisok = copy\_list(set[i], count);

start = clock() / CLOCK\_PER\_MS;

qs(spisok, 0, count - 1);

stop = clock() / CLOCK\_PER\_MS;

del\_list(spisok);

fprintf(file, "%dms ", (stop - start));

spisok = copy\_list(set[i], count);

start = clock() / CLOCK\_PER\_MS;

qsort(spisok, count, sizeof(int), compare);

stop = clock() / CLOCK\_PER\_MS;

del\_list(spisok);

fprintf(file, "%dms \n", (stop - start));

}

}

void main(void)

{

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

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

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

setlocale(LC\_ALL, "Rus");

FILE\* file;

char name[] = "logs.txt";

if ((file = fopen(name, "w")) == NULL) {

printf("Не удалось открыть файл");

\_getch();

}

int sizes[5] = { 10000, 20000, 40000, 80000, 160000 };

int pointer\_size = 0;

int\*\* matrix;

printf("waiting");

fprintf(file, "Наборы данных:\n1.Рандомный\n2.Возрастающий\n3.Убывающий\n4.Первая половина возрастает, вторая - убывает.\n");

while (pointer\_size < 5) {

matrix = (int\*\*)malloc(4 \* sizeof(int\*));

for (int i = 0; i < 4; i++) {

matrix[i] = (int\*)malloc((sizes[pointer\_size] \* sizeof(int)));

}

fillArray(matrix, sizes[pointer\_size]);

fprintf(file, "\nНабор данных размером в %d элементов\nshell || qs || qsort\n", sizes[pointer\_size]);

sorting\_set(matrix, sizes[pointer\_size], file);

pointer\_size++;

for (int i = 0; i < 4; i++)

{

free(matrix[i]);

}

free(matrix);

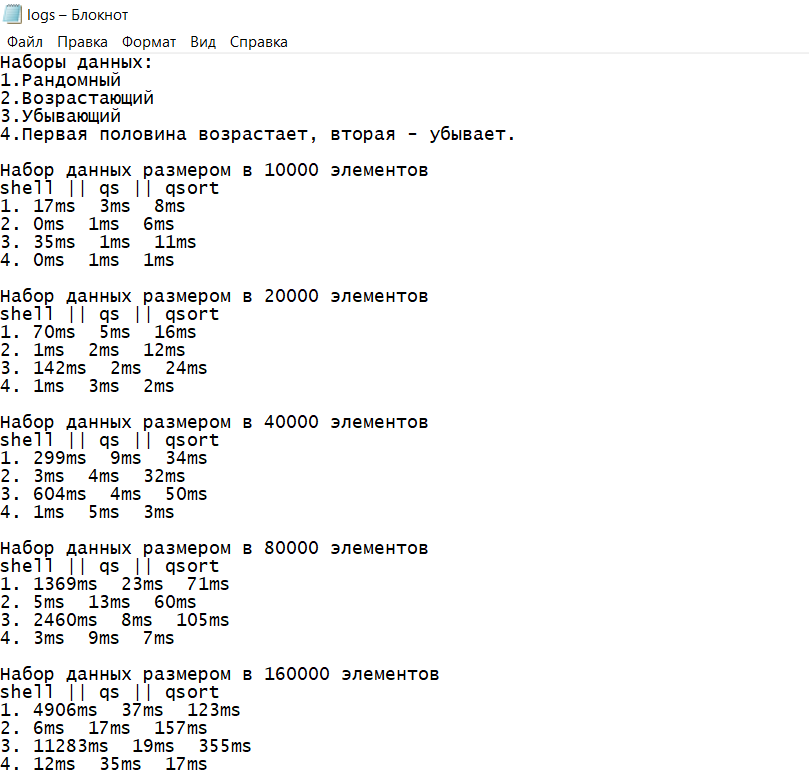
}

printf("\ndone");

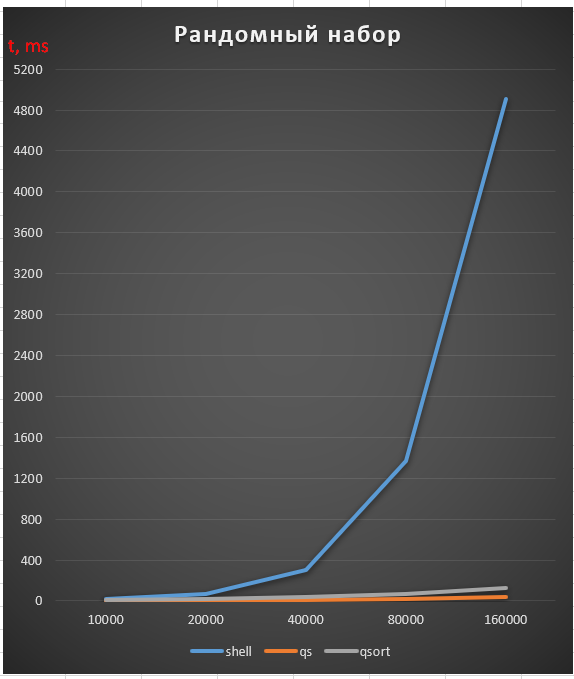
fclose(file);

}

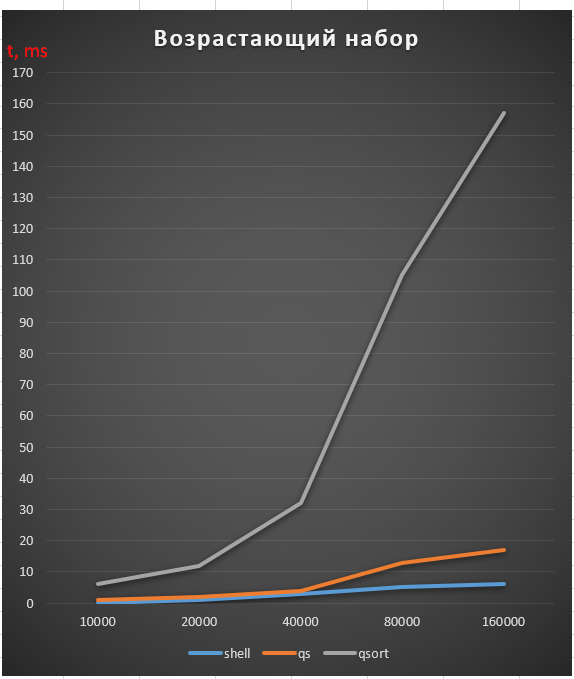
**Результаты работы программы**



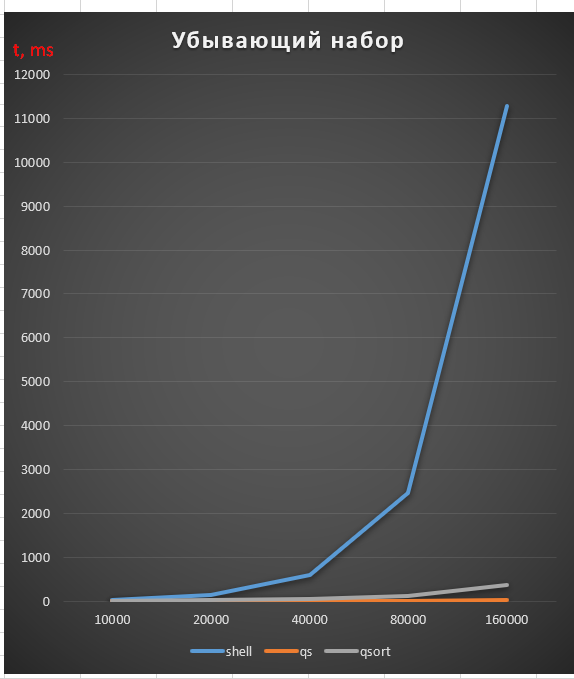
**Рисунок 3 – результат выполнение задания 2.**



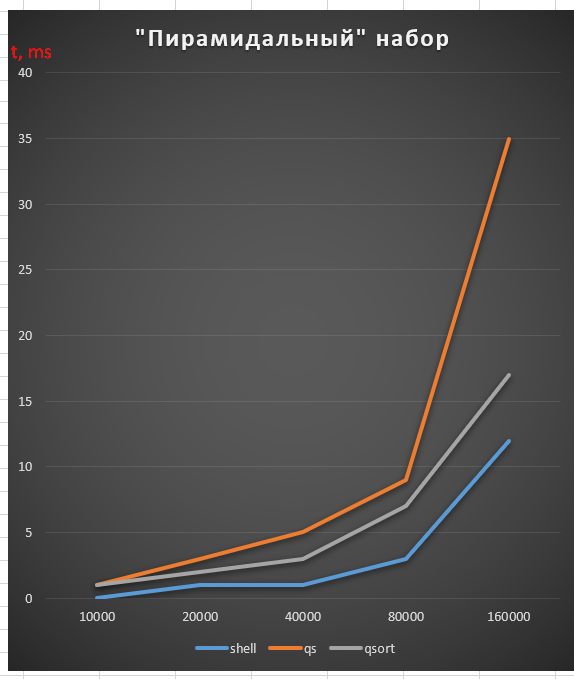
**Рисунок 4 – график зависимости времени от размера на 1-ом наборе данных.**



**Рисунок 5 - график зависимости времени от размера на 2-ом наборе данных.**



**Рисунок 6 - график зависимости времени от размера на 3-ом наборе данных.**



**Рисунок 7 - график зависимости времени от размера на 4-ом наборе данных.**

**Выводы**В ходе выполнения лабораторной работы было оценено время работы различных алгоритмов.

Алгоритм сортировки “shell” показал наиболее высокую эффективность в работе с : « Пирамидальным» и «Возрастающим» наборами данных.

Алгоритм сортировки “qs” лучше из всех справился с «Рандомным» и «Убывающим» наборами данных.

Алгориитм соритровки “qsort” отличиться эффективность ни в одном из наборов данных, к сожалению, не смог. Однако в трёх различных наборах данных(«Пирамидальный», «Убывающий», «Рандомный») показывал стабильное второе место по эффективности.