|  |
| --- |
| Пензенский государственный университет  Кафедра «Вычислительной техники» |
| Отчет |
|  |
| по дисциплине: «Логика и основы алгоритмизации в инженерных задачах» |
| на тему: «Оценка времени выполнения программ» |
|  |

**Выполнил:**

*Костин Д.А.*

**Принял:**

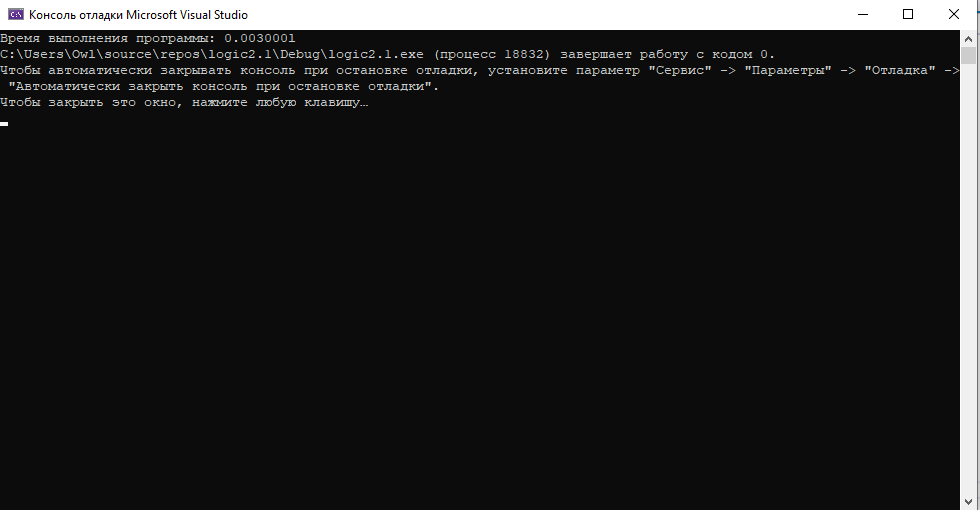
*Митрохин М. А.*

**Лабораторная работа № 2**

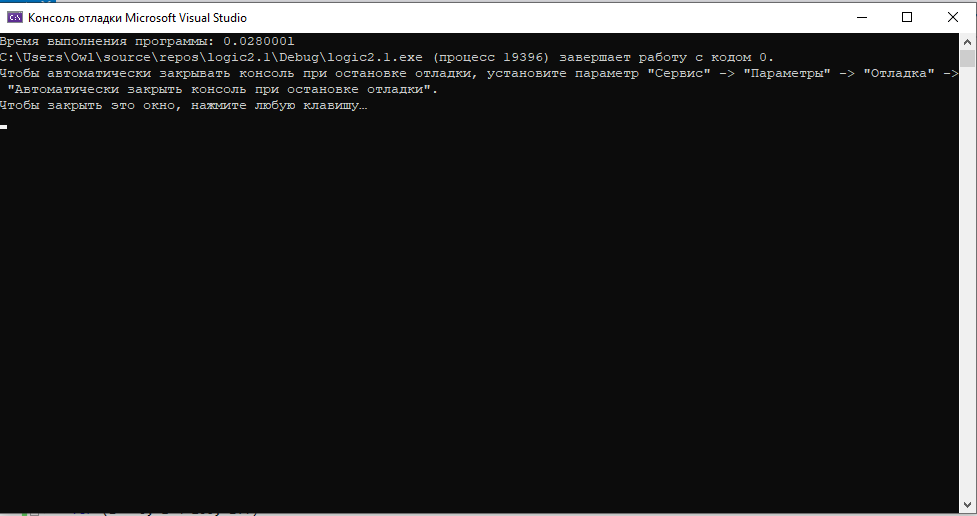
**Задание 1:**

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.

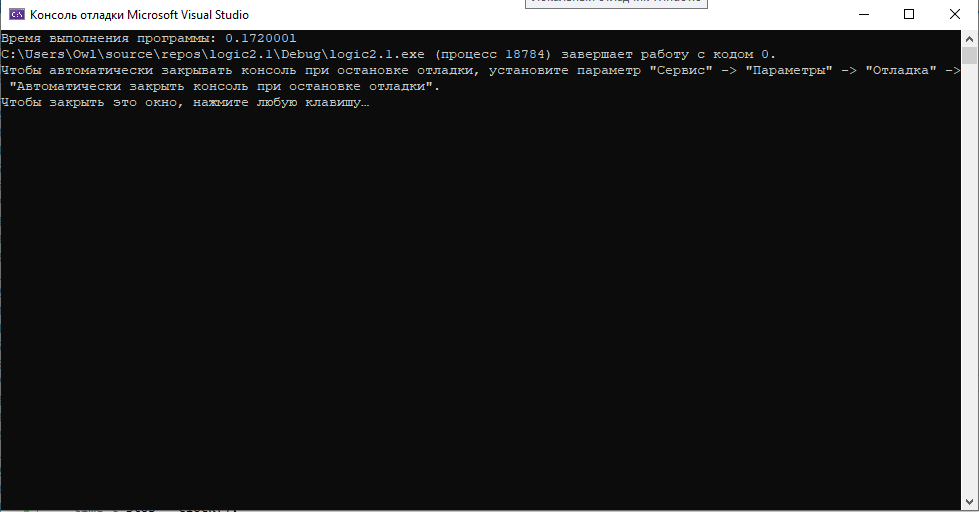
Время выполнения программы при размере матрицы 100



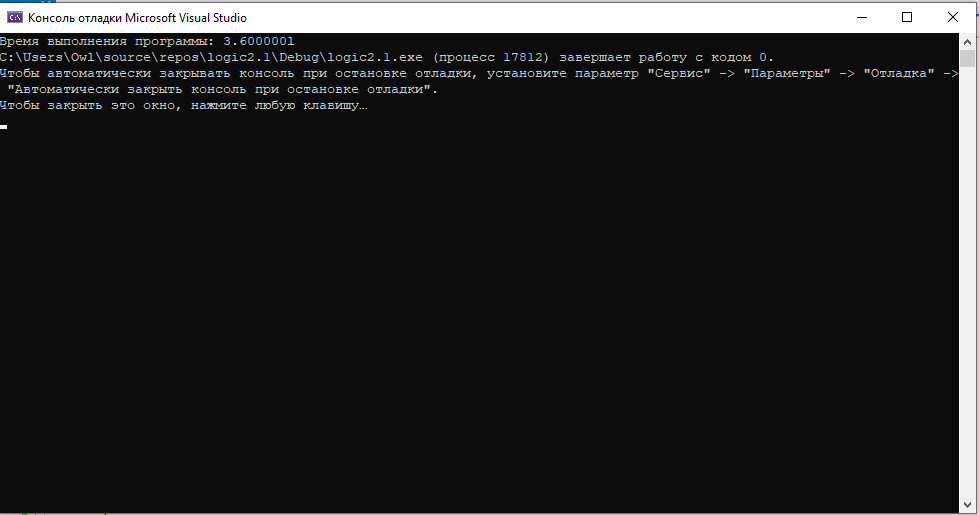
Время выполнения программы при размере матрицы 200



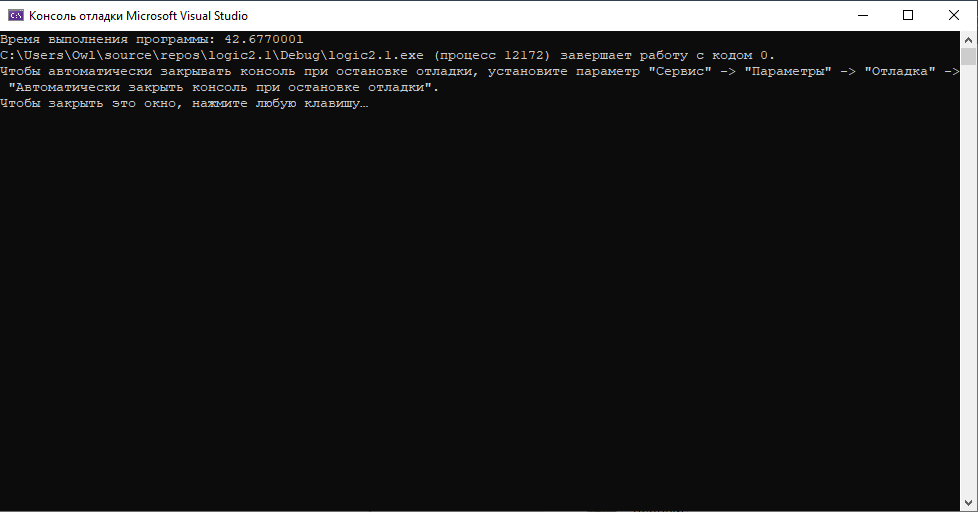
Время выполнения программы при размере матрицы 400



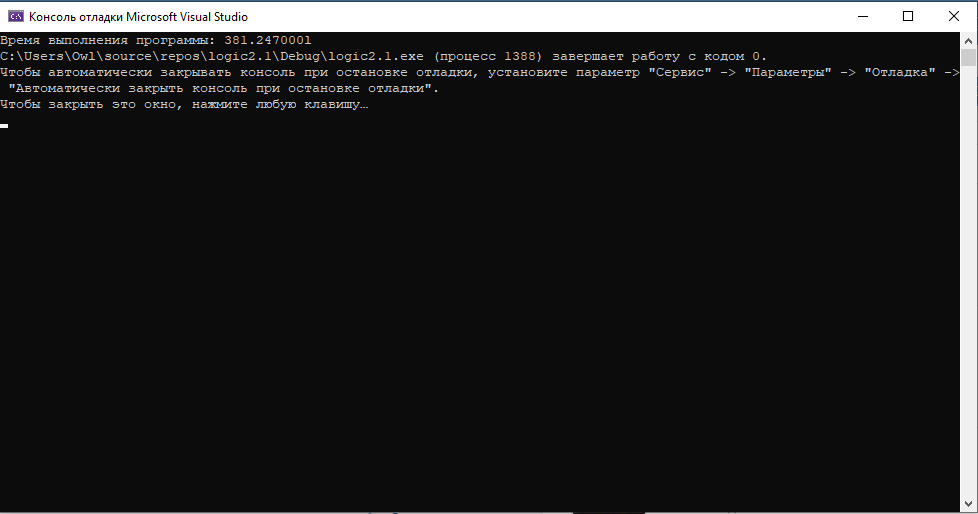
Время выполнения программы при размере матрицы 1000



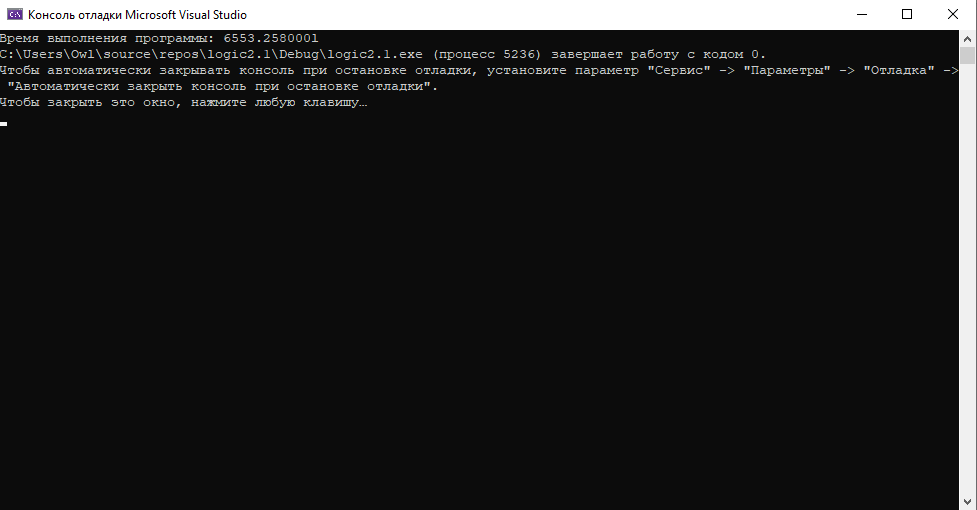
Время выполнения программы при размере матрицы 2000

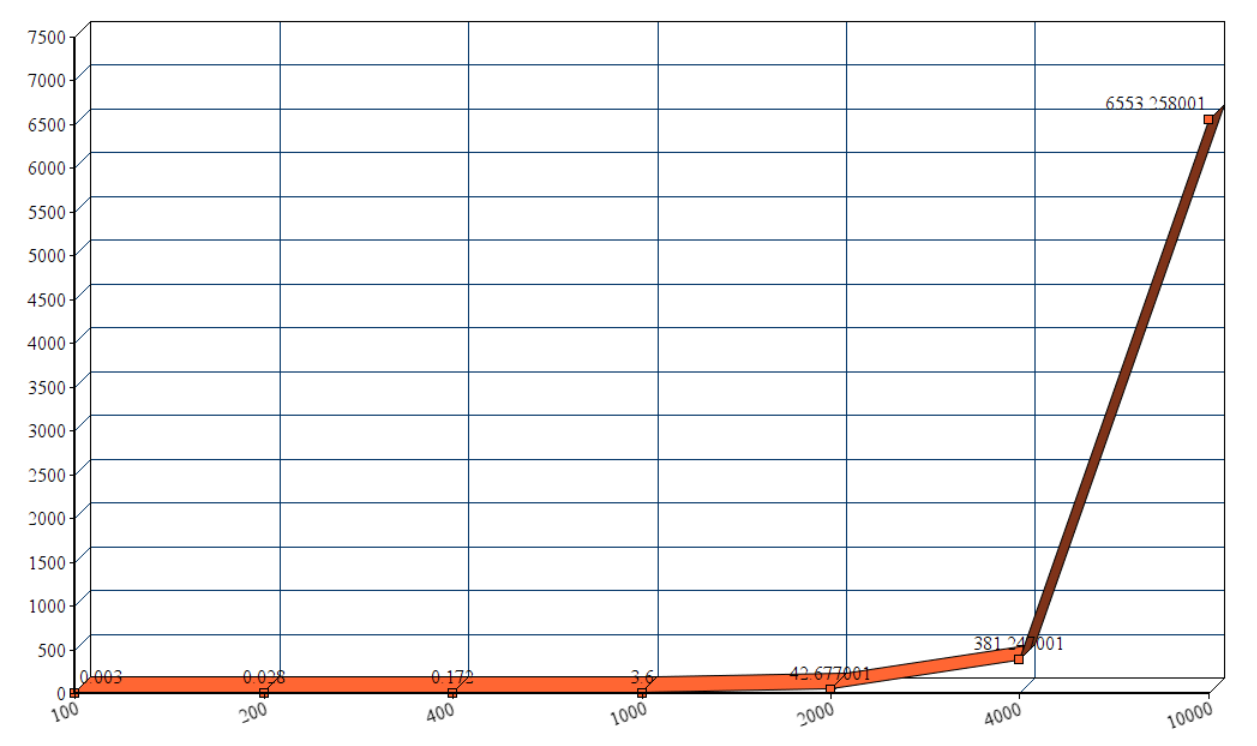


Время выполнения программы при размере матрицы 4000



Время выполнения программы при размере матрицы 10000



График зависимости времени выполнения от размера матриц:

**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

#include <stdlib.h>

#include <stdio.h>

#include <conio.h>

#include <locale.h>

#include <time.h>

#include < string.h >

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);

}

/\* сравнение двух целых \*/

int comp(const void\* i, const void\* j) {

return(int)i - (int)j;

}

void mas(int n, double\*\* ar,int b) {

//srand(time(NULL));

int \*a1, \*a2, \*a3, \*a4,\*a5;

int m=0;

double t;

a1 = (int\*)malloc(n \* sizeof(int));

a2 = (int\*)malloc(n \* sizeof(int));

a3 = (int\*)malloc(n \* sizeof(int));

a4 = (int\*)malloc(n \* sizeof(int));

a5 = (int\*)malloc(n \* sizeof(int));

for (int i = 0; i < n; i++) { //случайный набор значений

a1[i] = rand() % 100;

}

for (int i = 0; i < n; i++) { //возрастающая последовательность чисел

a2[i] = i;

}

for (int i = n; i > 0; i--) {//убывающая последовательность чисел

a3[i] = i;

}

for (int i = 0; i < n; i++) { //первая половина которого возроставет, а вторая убывает

if (i < n / 2) {

a4[i] = i;

//printf("%d\n", a4[i]);

}

else {

a4[i] = i - m;

m = m + 2;

//printf("%d\n", a4[i]);

}

}

//Сортировка Шелла

memcpy(a5, a1, n);

time\_t start = clock();

shell(a5, n);

time\_t stop = clock();

t = (stop - start) / 1000.0;

ar[b-3][0] = t;

printf("Время выполнения сортировки Шелла со случайным набором значений массива %fl\n", t);

//fprintf(f1, "Сортировка Шелла:\n");

//fprintf(f1, "1) %fl\n", t);

memcpy(a5, a2, n);

start = clock();

shell(a5, n);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 3][1] = t;

printf("Время выполнения сортировки Шелла c возрастающей последовательностью чисел %fl\n", t);

//fprintf(f1, "2) %fl\n", t);

memcpy(a5, a3, n);

start = clock();

shell(a5, n);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 3][2] = t;

printf("Время выполнения сортировки Шелла c убывающей последовательностью чисел %fl\n", t);

//fprintf(f1, "3) %fl\n", t);

memcpy(a5, a4, n);

start = clock();

shell(a5, n);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 3][3] = t;

printf("Время выполнения сортировки Шелла первая половина которого возроставет, а вторая убывает %fl\n", t);

//fprintf(f1, "4) %fl\n", t);

printf("\*\*\*\*\*\*\*\*\*\*\*\n");

// Быстрая сортировка

//fprintf(f1, "Быстрая сортировка:\n");

memcpy(a5, a1, n);

start = clock();

qs(a5, 0, n - 1);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 2][0] = t;

printf("Время выполнения быстрой сортировки со случайным набором значений массива %fl\n", t);

//fprintf(f1, "1) %fl\n", t);

memcpy(a5, a2, n);

start = clock();

qs(a5, 0, n - 1);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 2][1] = t;

printf("Время выполнения быстрой сортировки c возрастающей последовательностью чисел %fl\n", t);

//fprintf(f1, "2) %fl\n", t);

memcpy(a5, a3, n);

start = clock();

qs(a5, 0, n - 1);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 2][2] = t;

printf("Время выполнения быстрой сортировки c убывающей последовательностью чисел %fl\n", t);

//fprintf(f1, "3) %fl\n", t);

memcpy(a5, a4, n);

start = clock();

qs(a5, 0, n - 1);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 2][3] = t;

printf("Время выполнения быстрой сортировки первая половина которого возроставет, а вторая убывает %fl\n", t);

//fprintf(f1, "4) %fl\n", t);

//стандартной функции qsort

printf("\*\*\*\*\*\*\*\*\*\*\*\n");

//fprintf(f1, "Стандартной функции qsort\n");

memcpy(a5, a1, n);

start = clock();

qsort(a5, n, sizeof(int), comp);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 1][0] = t;

printf("Время выполнения стандартной функции qsort со случайным набором значений массива %fl\n", t);

//fprintf(f1, "1) %fl\n", t);

memcpy(a5, a2, n);

start = clock();

qsort(a5, n, sizeof(int), comp);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 1][1] = t;

printf("Время выполнения стандартной функции qsort c возрастающей последовательностью чисел %fl\n", t);

//fprintf(f1, "2) %fl\n", t);

memcpy(a5, a3, n);

start = clock();

qsort(a5, n, sizeof(int), comp);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 1][2] = t;

printf("Время выполнения стандартной функции qsort c убывающей последовательностью чисел %fl\n", t);

//fprintf(f1, "3) %fl\n", t);

memcpy(a5, a4, n);

start = clock();

qsort(a5, n, sizeof(int), comp);

stop = clock();

t = (stop - start) / 1000.0;

ar[b - 1][3] = t;

printf("Время выполнения стандартной функции qsort первая половина которого возроставет, а вторая убывает %fl\n", t);

//fprintf(f1, "4) %fl\n", t);

}

int main() {

system("chcp 1251");

system("cls");

FILE\* f1;

int n=0,arr,m=4,c,b=0,z=0,j=0,g=0;

double\*\* a;

double\*\* bb;

int\* aa;

double v;

char f[] = "my.txt";

f1 = fopen(f, "w");

printf("Введите колличество массивов");

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

c = arr \* 3;

g = 12;

a = (double\*\*)malloc(c \* sizeof(double\*));

bb = (double\*\*)malloc(g \* sizeof(double\*));

aa = (int\*)malloc(arr \* sizeof(int\*));

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

a[i] = (double\*)malloc(m \* sizeof(double));

}

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

bb[i] = (double\*)malloc(arr \* sizeof( double));

}

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

b = b + 3;

printf("Введите размер массива\n");

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

aa[i] = n;

mas(n, a, b);

}

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

fprintf(f1, " %d", aa[i]);

}

fprintf(f1, "%\n");

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

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

bb[k][i] = a[z][j];

if (j < m ) {

j++;

}

else {

j = 0;

if (z != c-1) {

z++;

}

}

if (j == 4) {

j = 0;

if (z != c - 1) {

z++;

}

}

}

}

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

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

fprintf(f1, " %lf", bb[i][k]);

}

fprintf(f1, "%\n");

}

}

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | **2000** | **4000** | **10000** | **20000** | **100000** | **200000** |
| **Шелла 1** | 0.001000 | 0.001000 | 0.005000 | 0.020000 | 0.511000 | 2.169000 |
| **Шелла 2** | 0.000001 | 0.001000 | 0.004000 | 0.018000 | 0.462000 | 2.020000 |
| **Шелла 3** | 0.000001 | 0.000001 | 0.004000 | 0.015000 | 0.384000 | 1.579000 |
| **Шелла 4** | 0.000001 | 0.000001 | 0.003000 | 0.016000 | 0.309000 | 1.238000 |
| **Быстрая 1** | 0.000001 | 0.000001 | 0.000001 | 0.001000 | 0.005000 | 0.011000 |
| **Быстрая 2** | 0.000001 | 0.000001 | 0.000001 | 0.001000 | 0.007000 | 0.013000 |
| **Быстрая 3** | 0.000001 | 0.000001 | 0.001000 | 0.001000 | 0.006000 | 0.015000 |
| **Быстрая 4** | 0.000001 | 0.001000 | 0.001000 | 0.002000 | 0.008000 | 0.016000 |
| **qSort 1** | 0.000001 | 0.001000 | 0.002000 | 0.008000 | 0.025000 | 0.058000 |
| **qSort 2** | 0.001000 | 0.001000 | 0.002000 | 0.006000 | 0.025000 | 0.057000 |
| **qSort 3** | 0.000001 | 0.001000 | 0.002000 | 0.006000 | 0.025000 | 0.053000 |
| **qSort 4** | 0.000001 | 0.001000 | 0.002000 | 0.005000 | 0.025000 | 0.054000 |

**Вывод:** Быстрая сортировка оказалась быстрейщей на всех наборах данных сортировка шелла-самая медленная на болльших наборах данных.