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:**

студенты группы 21ВВ3

Чинов Даниил

Тюкалов Василий

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пенза 2021

# Название

Оценка времени выполнения программ

**Цель работы**–изучение средств, предоставляемых библиотекой time.h,методов сортировок.

# Методические указания

Для оценки времени выполнения программ языка Си или их частей могут

использоваться средства, предоставляемые библиотекой time.h. Данная библиотека содержит описания типов и прототипы функций для работы с датой и временем.

# Лабораторное задание

Задание 1:

1. Вычислить порядок сложности программы (О-символику).

2. Оценить время выполнения программы и кода, выполняющего перемножениематриц, 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 матрици сравнить полученный результат с теоретической оценкой.

Задание 2:

1. Оценить время работы каждого из реализованных алгоритмов на случайном наборезначений 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, авторая, – убывающую.

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

**Ход работы:**

Задание 1:

1.Вычислили порядок сложности программы (О-символику).

O(N^2)+O(N^2)+O(N^3)=O(N^3)

2.Оценили время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 100, 200,400, 1000, 2000, 4000, 10000.

**Листинг:**

// Выполнили студенты группы 21ВВ3: Тюкалов В.Е. и Чинов Д.Д.

#include <iostream>

#include <stdio.h>

#include <cstdlib>

#include <stdlib.h>

#include <time.h>

int main(void)

{

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

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

setlocale(LC\_ALL, "rus");

int i = 0, j = 0, r = 0, elem\_c = 0, N = 0;

printf("Введите размер матрицы: ");

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

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

int\*\* b = (int\*\*)malloc(sizeof(int\*) \* N);

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

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

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

b[k] = (int\*)malloc(sizeof(int) \* N);

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

}

time\_t start = clock();

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

while (i < N)

{

while (j < N)

{

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

j++;

}

i++;

j = 0;

}

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

i = 0; j = 0;

while (i < N)

{

while (j < N)

{

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

j++;

}

i++;

j = 0;

}

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

{

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

{

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

{

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

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

}

}

}

time\_t end = clock();

float res = end - start;

printf("Время работы алгоритма: %f c", res / CLOCKS\_PER\_SEC);

for (int k = N-1; k > 0; k--) {

free(a[k]);

free(b[k]);

free(c[k]);

}

free(a);

free(b);

free(c);

return(0);

}

**Результат работы программы для матриц размерами 100:**

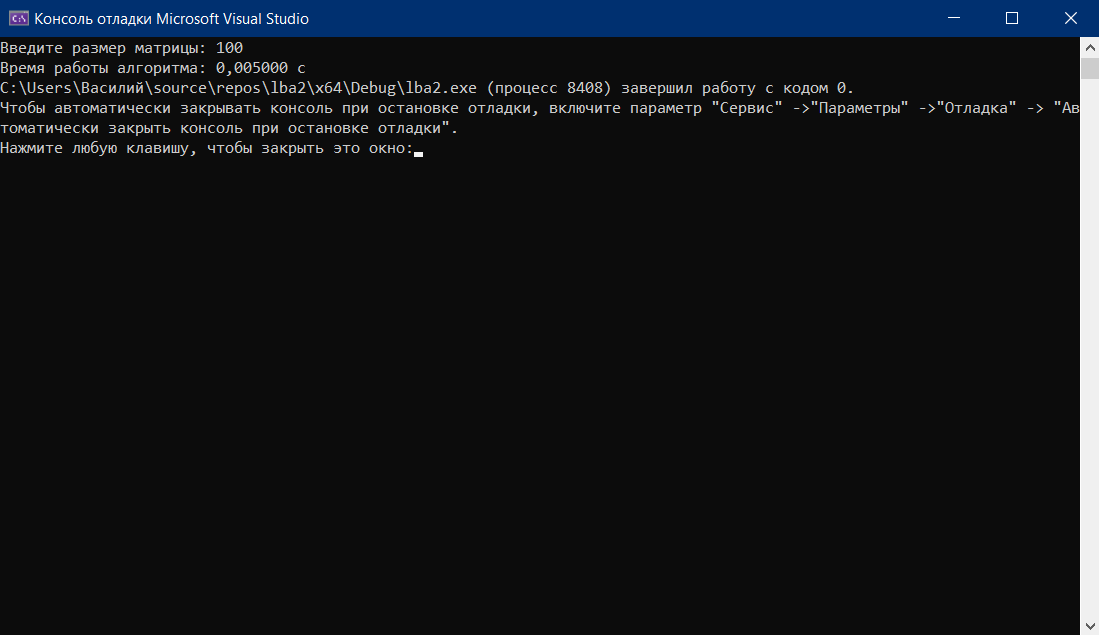


Рисунок 1 Результат работы программы для матрицы размерами 100

**Результат работы программы для матриц размерами 200:**

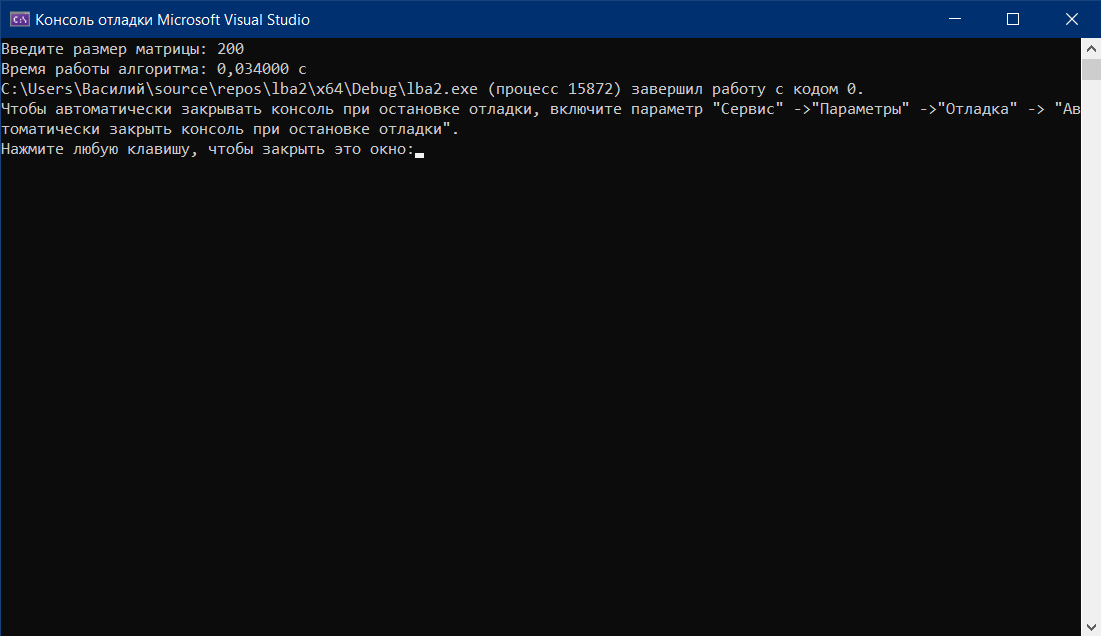


Рисунок 2 Результат работы программы для матрицы размерами 200

**Результат работы программы для матриц размерами 400:**

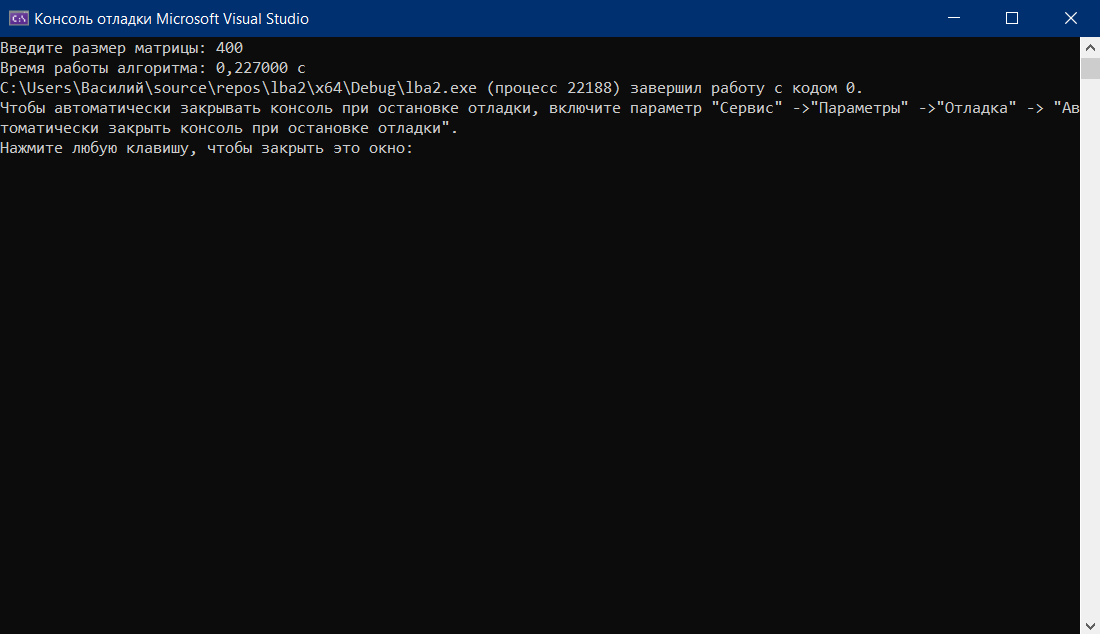


Рисунок 3 Результат работы программы для матрицы размерами 400

**Результат работы программы для матриц размерами 1000:**

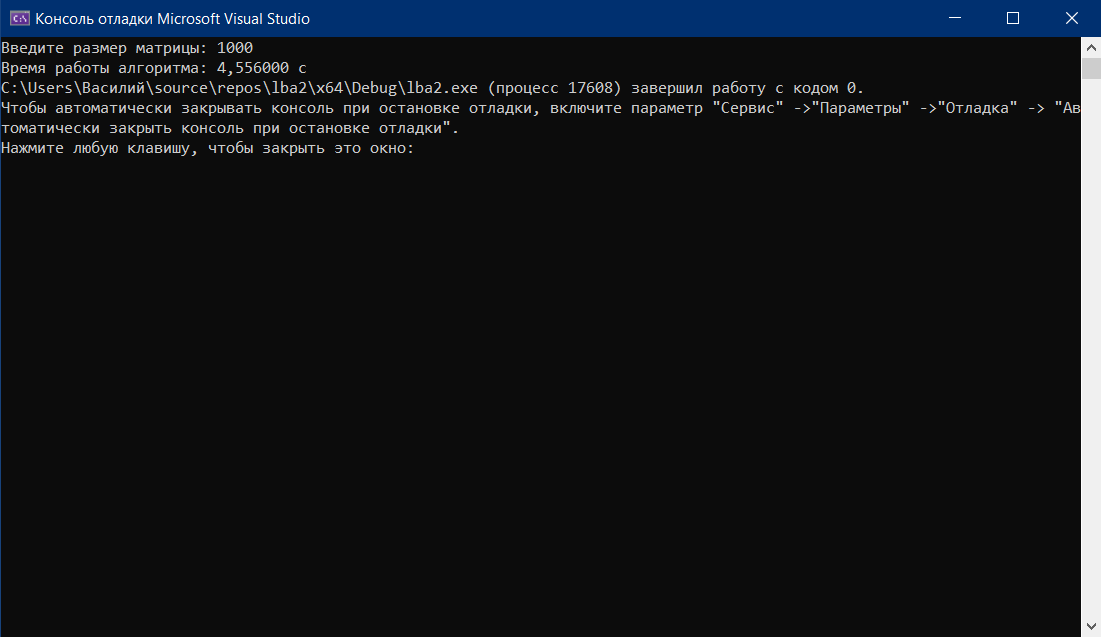


Рисунок 4 Результат работы программы для матрицы размерами 1000

**Результат работы программы для матриц размерами 2000:**

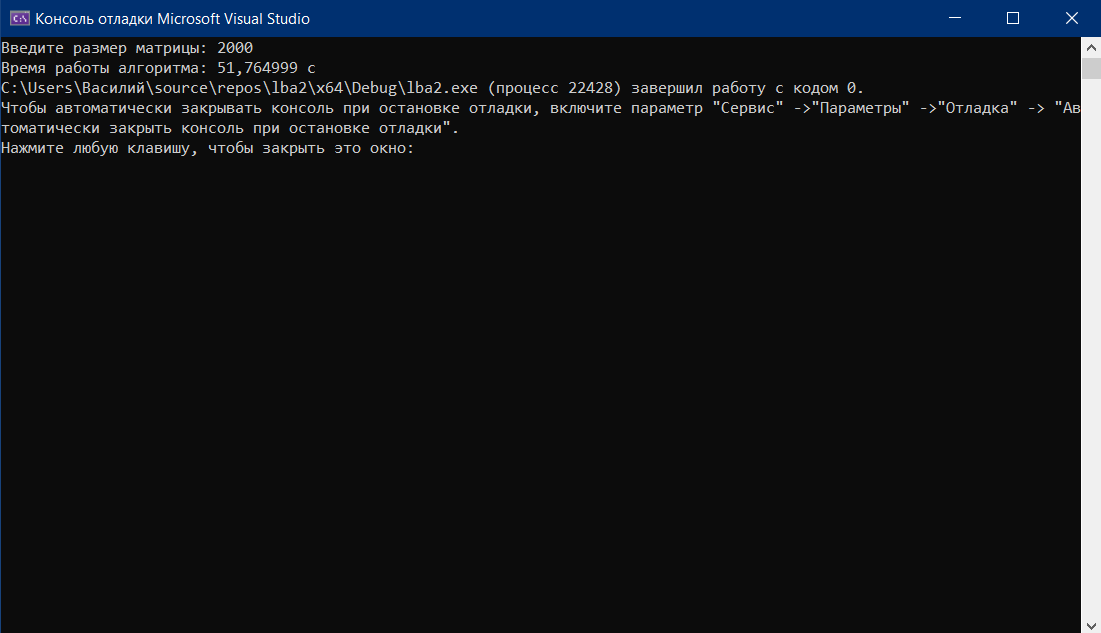


Рисунок 5 Результат работы программы для матрицы размерами 2000

**Результат работы программы для матриц размерами 3000:**

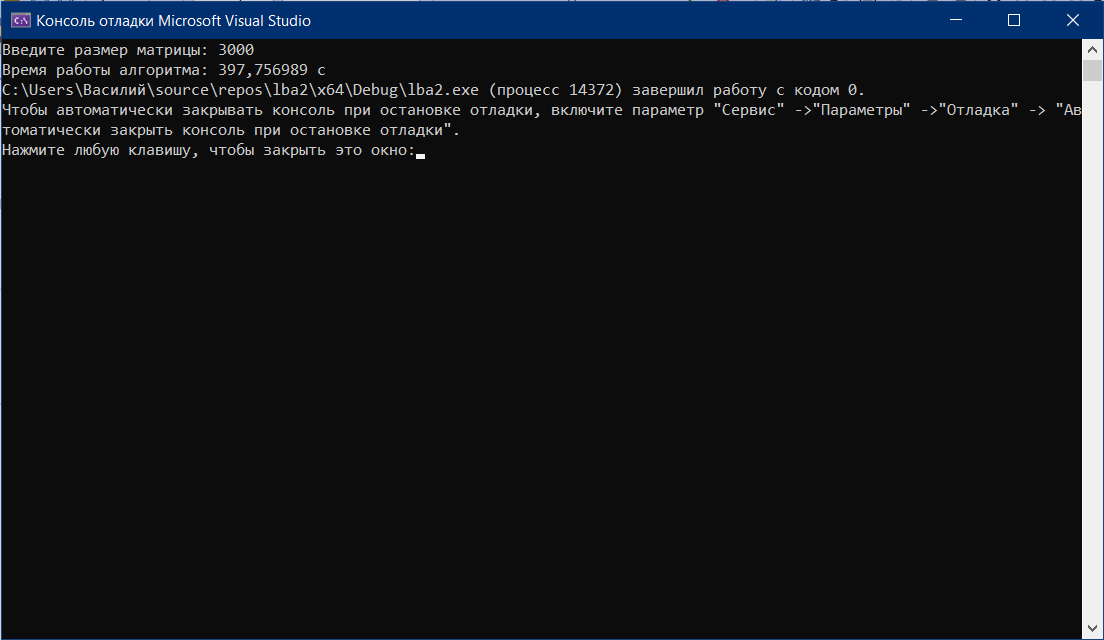


Рисунок 6 Результат работы программы для матрицы размерами 3000

**Результат работы программы для матриц размерами 4000:**

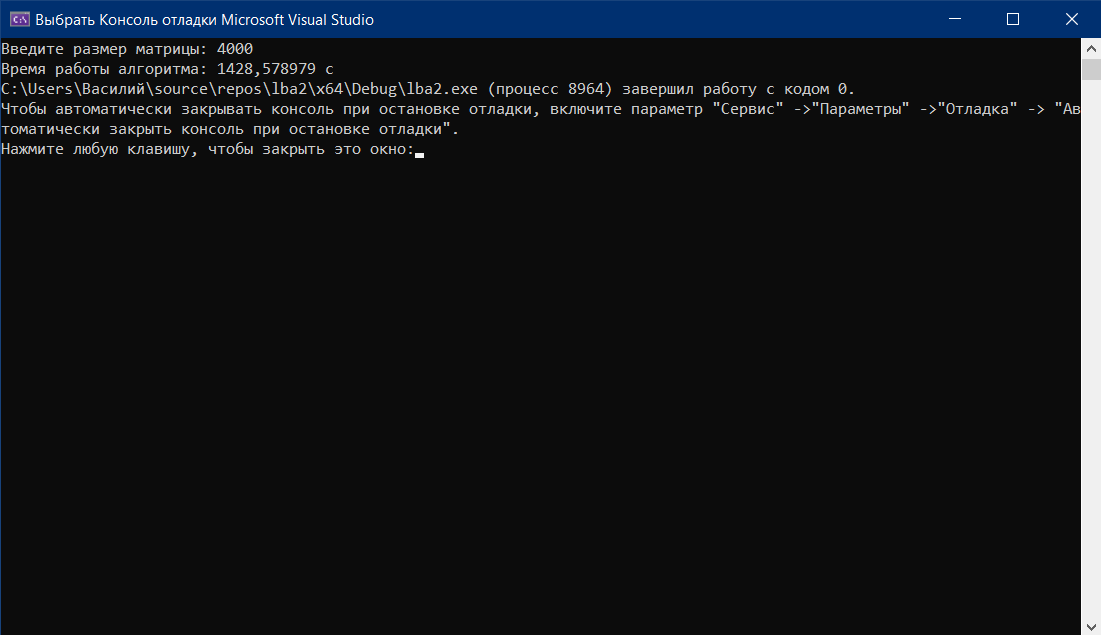


Рисунок 7 Результат работы программы для матрицы размерами 4000

3.Построили график зависимости времени выполнения программы от размера матриц и сравнили полученный результат с теоретической оценкой.

Рисунок 8 График зависимости времени выполнения программы от размера матриц

**Задание 2**

**Листинг:**

1. Оценили время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

[#include](https://vk.com/im?sel=206827319&st=%23include) <iostream>  
[#include](https://vk.com/im?sel=206827319&st=%23include) <stdio.h>  
[#include](https://vk.com/im?sel=206827319&st=%23include) <cstdlib>  
[#include](https://vk.com/im?sel=206827319&st=%23include) <stdlib.h>  
[#include](https://vk.com/im?sel=206827319&st=%23include) <time.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 shell\_minus(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 int\* a, const int\* b) {  
return \*a - \*b;  
}  
  
  
int main() {  
setlocale(LC\_ALL, "RUS");  
srand(time(NULL));  
  
clock\_t start, end;  
int count;  
float res\_qs[4], res\_shell[4], res\_qsort[4];  
  
printf("Введите размерность массива: ");  
scanf\_s("%d", &count);  
  
int\* items = (int\*)malloc(count \* sizeof(items));  
for (int i = 0; i < count; i++)  
{  
items[i] = rand() % 100;  
}  
  
//Пункт 1  
start = clock();  
shell(items, count);  
end = clock();  
res\_shell[0] = (end - start);  
start = clock();  
qs(items, 0, count - 1);  
end = clock();  
res\_qs[0] = (end - start);  
  
//Пункт 2  
qs(items, 0, count - 1);  
start = clock();  
shell(items, count);  
end = clock();  
res\_shell[1] = (end - start);  
start = clock();  
qs(items, 0, count - 1);  
end = clock();  
res\_qs[1] = (end - start);  
  
//Пункт 3  
shell\_minus(items, count);  
start = clock();  
shell(items, count);  
end = clock();  
res\_shell[2] = (end - start);  
start = clock();  
qs(items, 0, count - 1);  
end = clock();  
res\_qs[2] = (end - start);  
  
//Пункт 4  
qs(items, count / 2, count - 1);  
shell\_minus(items, count / 2);  
start = clock();  
shell(items, count);  
end = clock();  
res\_shell[3] = (end - start);  
start = clock();  
qs(items, 0, count - 1);  
end = clock();  
res\_qs[3] = (end - start);  
  
//Пункт 5  
start = clock();  
qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
res\_qsort[0] = end - start;  
  
shell(items, count);  
start = clock();  
qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
res\_qsort[1] = end - start;  
  
shell\_minus(items, count);  
start = clock();  
qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
res\_qsort[2] = end - start;  
  
qs(items, count / 2, count - 1);  
shell\_minus(items, count / 2);  
start = clock();  
qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);  
end = clock();  
res\_qsort[3] = end - start;  
  
printf(" \n |rand |up |down |up-down\n");  
printf(" —--------------------------------------------------\n");  
printf(" shell |%5f |%5f |%5f |%5f\n", res\_shell[0] / 1000, res\_shell[1] / 1000, res\_shell[2] / 1000, res\_shell[3] / 1000);  
printf(" qs |%5f |%5f |%5f |%5f\n", res\_qs[0] / 1000, res\_qs[1] / 1000, res\_qs[2] / 1000, res\_qs[3] / 1000);  
printf(" qsort |%5f |%5f |%5f |%5f\n", res\_qsort[0] / 1000, res\_qsort[1] / 1000, res\_qsort[2] / 1000, res\_qsort[3] / 1000);  
  
return 0;  
}

1. Результат оценки времени работы трёх алгоритмов: сортировка Шела, быстрая сортировка, стандартная функция qsort.

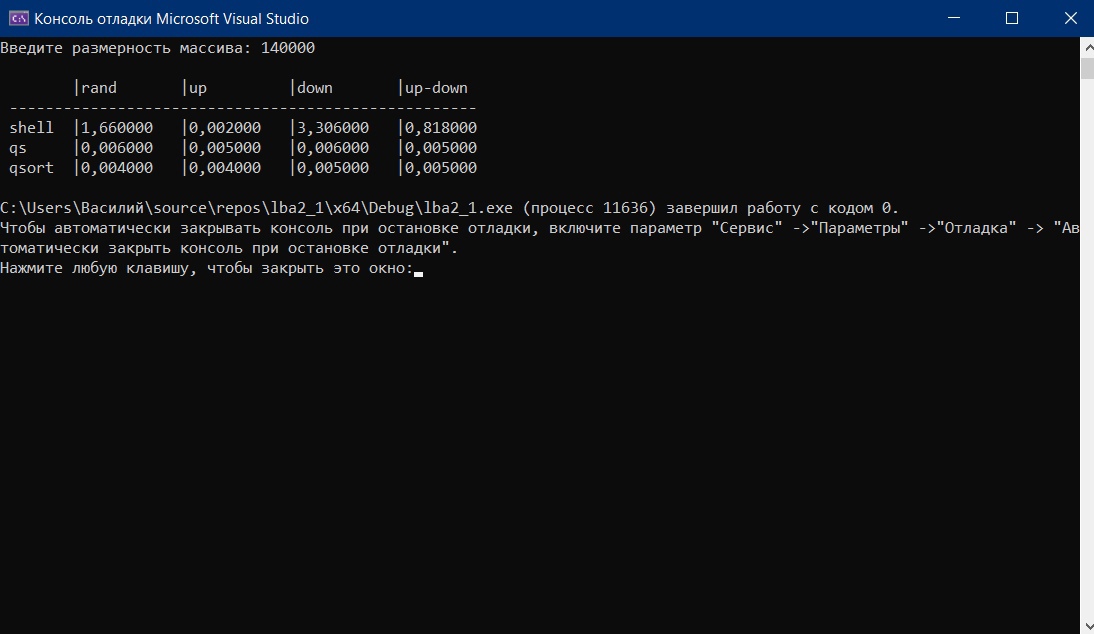


Рисунок 8 Результат работы трёх алгоритмов при размерности 140000

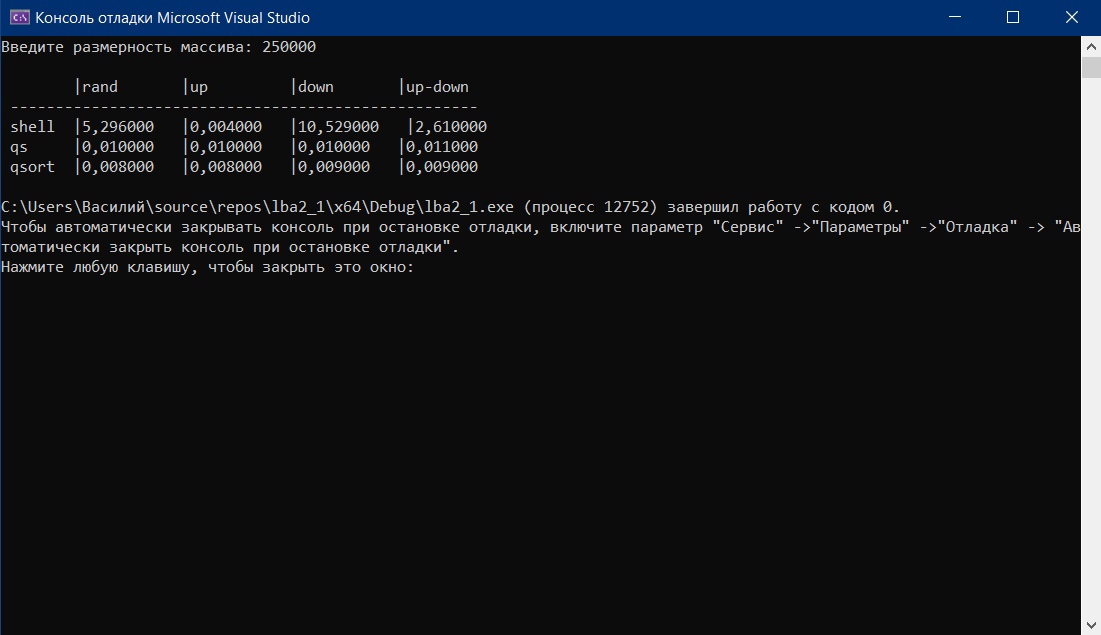


Рисунок 9 Результат работы трёх алгоритмов при размерности 250000

**Вывод:** в процессе работы были изучены средства, предоставляемые библиотекой time.h, различные виды сортировок. Были получены навыки по сравнению сортировок и определению их эффективности. Была определена сложность программы O(n^3) и экспериментально подтверждена правильность нахождения времени работы алгоритмов. Также была определена эффективность сортировок в зависимости от последовательности чисел. Так, было установлено, что при Возрастающей последовательности сортировка Шелла будет самой эффективной, а при случайном наборе, убывающем и пилообразном-Быстрая сортировка.