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 20ВВ3

Пантюшов Егор

Шмелёв Данила

**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:**

#define\_CRT\_NONSTDC\_NO\_WARNINGS

#define\_CRT\_SECURE\_NO\_WARNINGS

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<locale.h>

int main(void)

{

setlocale(LC\_ALL, "Russian");

int \*\*mas\_a = 0;

int \*\*mas\_b = 0;

int \*\*mas\_c= 0;

intrazmer,elem\_c;

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

printf("Ввведите размер матриц:");

scanf("%d", &razmer);

start = clock();

mas\_a = malloc(sizeof(\*mas\_a) \* razmer);

for (inti = 0; i<razmer; i++)

mas\_a[i] = malloc(sizeof(\*\*mas\_a) \* razmer);

mas\_b = malloc(sizeof(\*mas\_b) \* razmer);

for (inti = 0; i<razmer; i++)

mas\_b[i] = malloc(sizeof(\*\*mas\_b) \* razmer);

mas\_c = malloc(sizeof(\*mas\_c) \* razmer);

for (inti = 0; i<razmer; i++)

mas\_c[i] = malloc(sizeof(\*\*mas\_c) \* razmer);

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

for (inti = 0; i<razmer; i++)

for (int j = 0; j <razmer; j++)

mas\_a[i][j] = rand()%10 \* razmer + 1;

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

for (inti = 0; i<razmer; i++)

for (int j = 0; j <razmer; j++)

mas\_b[i][j] = rand()%10 \* razmer + 1;

for (inti = 0; i<razmer; i++)

{

for (int j = 0; j <razmer; j++)

{

elem\_c = 0;

for (int r = 0; r <razmer; r++)

{

elem\_c = elem\_c + mas\_a[i][r] \* mas\_b[r][j];

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

}

}

}

/\*for (inti = 0; i<razmer; i++)

{

printf("\n");

for (int j = 0; j <razmer; j++)

printf("%d\t", mas\_a[i][j]);

}

printf("\n\n");

for (inti = 0; i<razmer; i++)

{

printf("\n");

for (int j = 0; j <razmer; j++)

printf("%d\t", mas\_b[i][j]);

}

printf("\n\n");

for (inti = 0; i<razmer; i++)

{

printf("\n");

for (int j = 0; j <razmer; j++)

printf("%d\t", mas\_c[i][j]);

}\*/

end = clock();

float time = end - start;

printf("Время выполнения работы программы=%.3f с\n", time / CLOCKS\_PER\_SEC);

return(0);

}

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

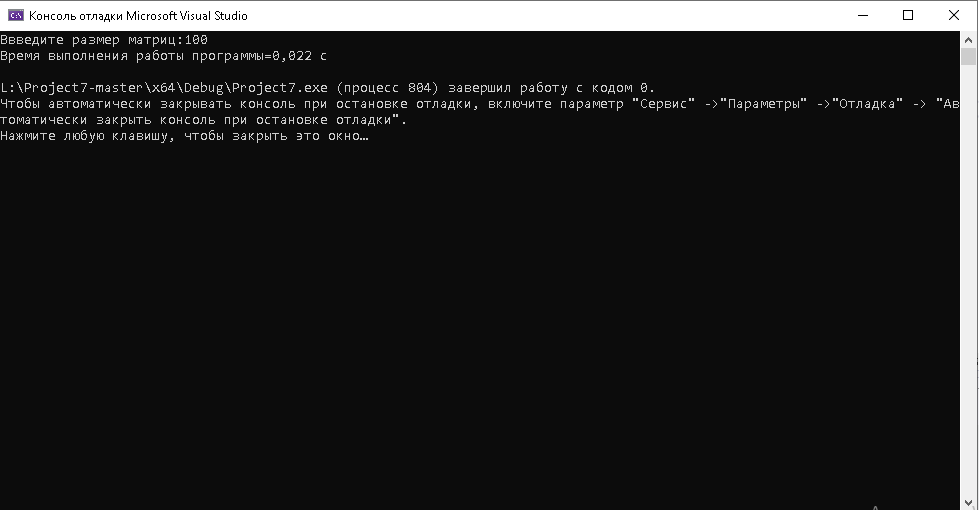


Рисунок 1 Результат работы программы для матрицы размерами 100

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

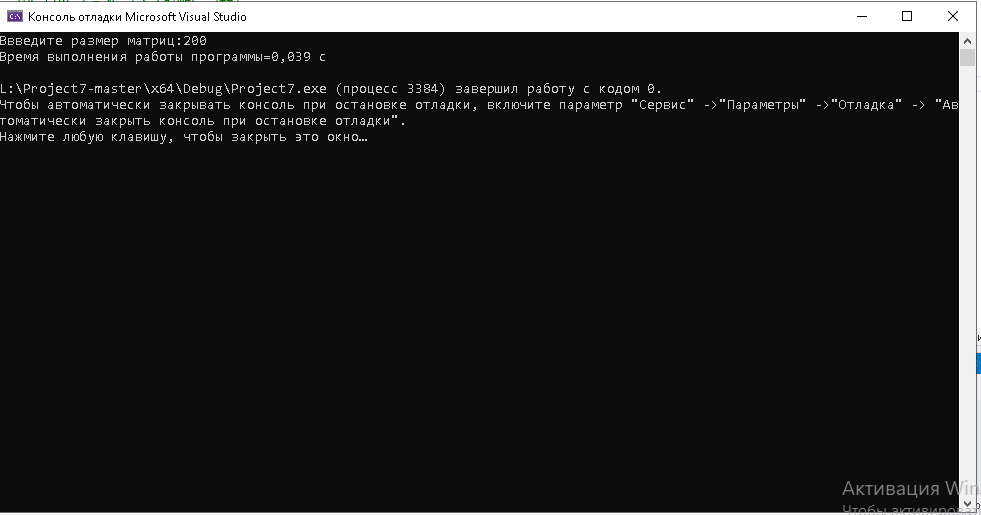


Рисунок 2 Результат работы программы для матрицы размерами 200

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

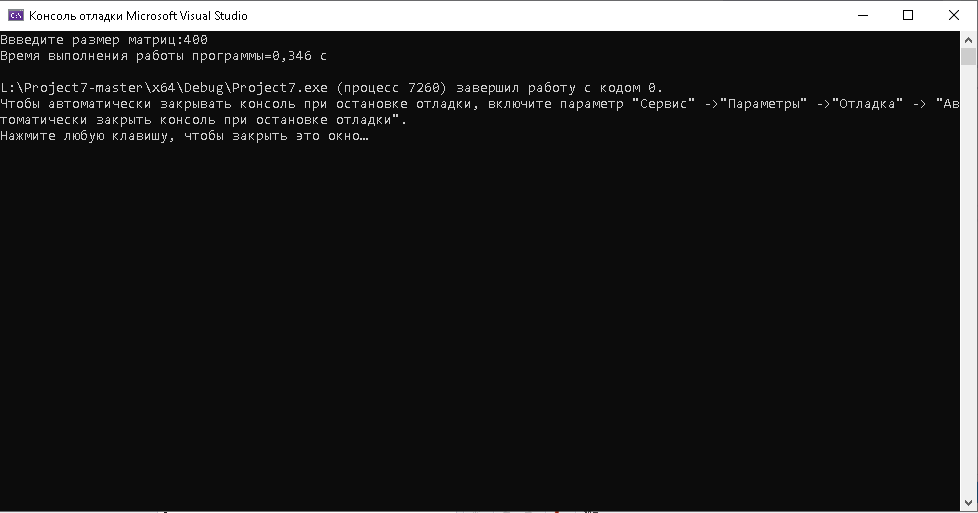


Рисунок 3 Результат работы программы для матрицы размерами 400

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

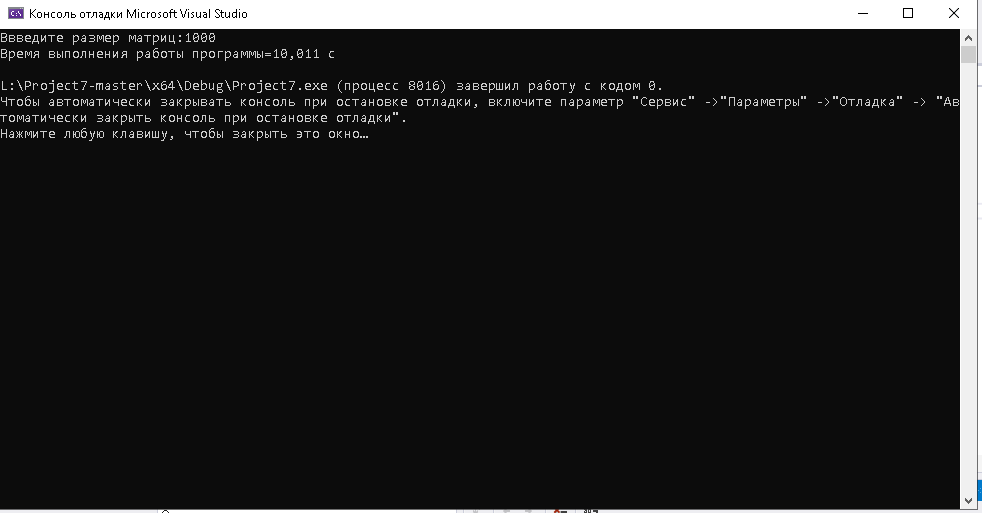


Рисунок 4 Результат работы программы для матрицы размерами 1000

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

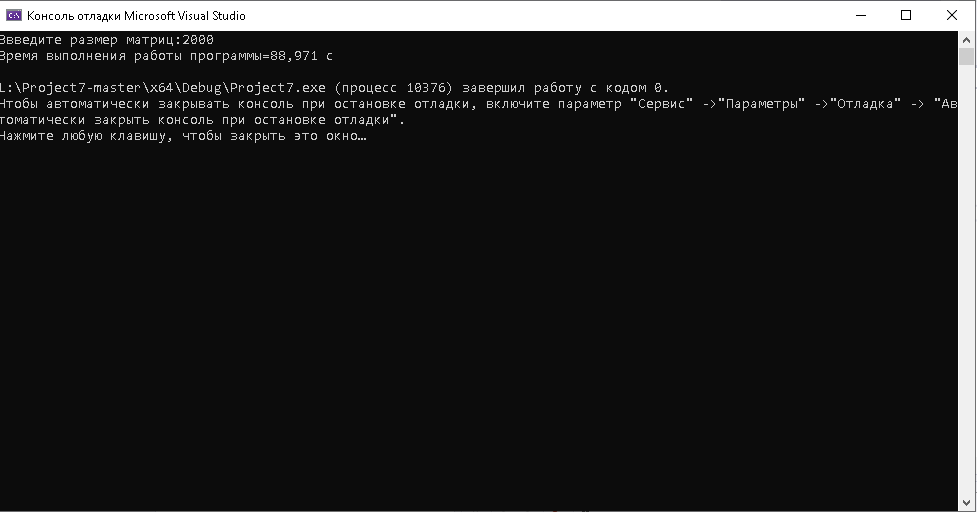


Рисунок 5 Результат работы программы для матрицы размерами 2000

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

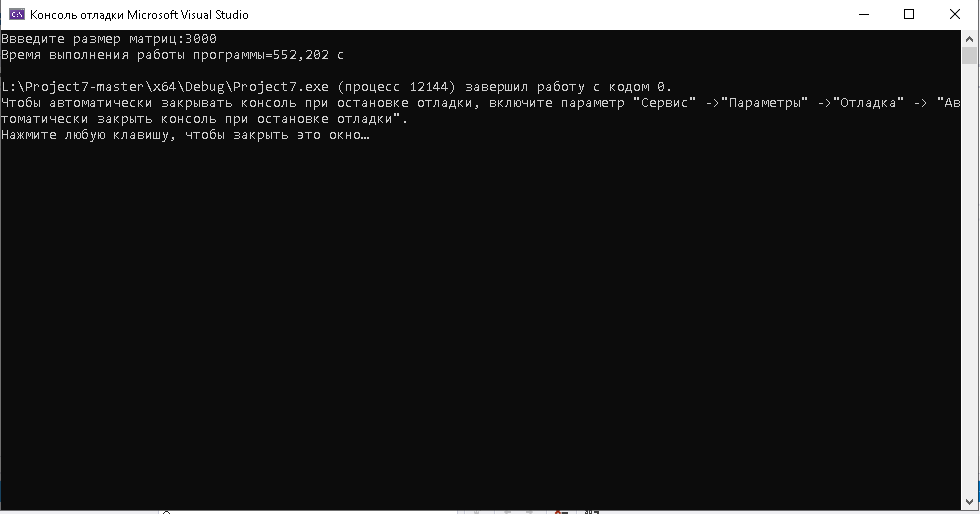


Рисунок 6 Результат работы программы для матрицы размерами 3000

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

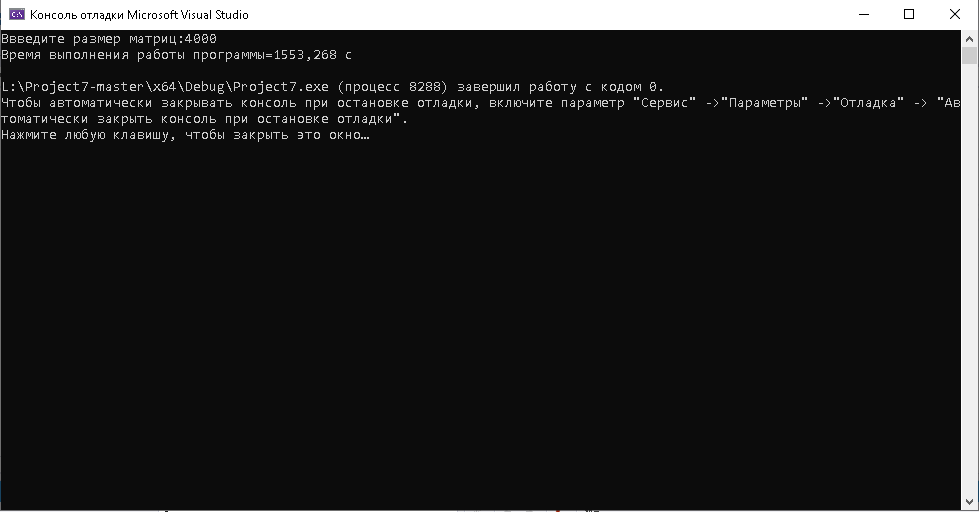


Рисунок 7 Результат работы программы для матрицы размерами 4000

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



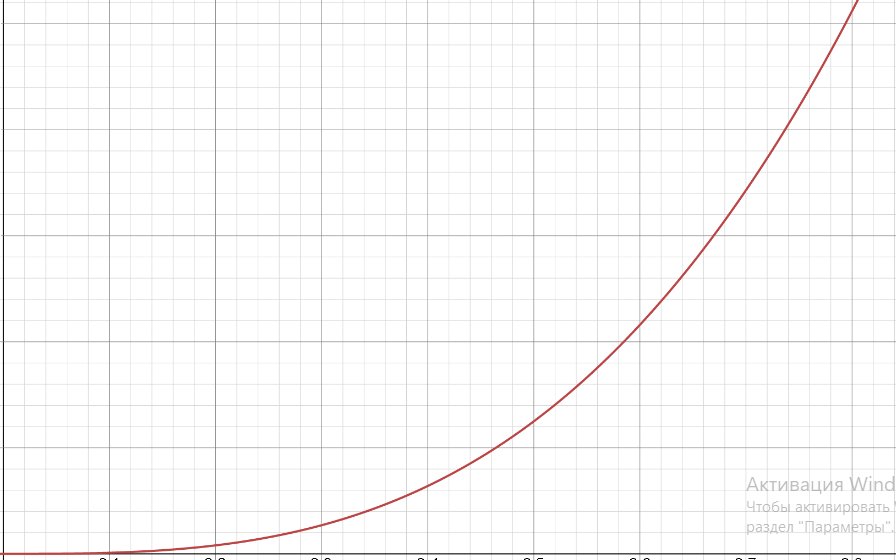


Рисунок 8 График кубической функции

Графики идентичны! Следовательно программа работает верно.

**Задание 2**

**Листинг:**

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#define HEADER ("Лабораторная работа №2\nВыполнили: Шмелёв Д.В. и Пантюшов Е.И.")

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

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

{

int i, j;

int x, y;

i = left; j = right;

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);

}

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;

}

}

}

int compare(const int\* a, const int\* b) {

return \*a - \*b;

}

int main()

{

//Задание 2

setlocale(LC\_ALL, "RUS");

srand(time(NULL));

clock\_t start, end;

int count;

printf("Введите размерность массива: ");

scanf("%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);

qs(items, 0, count - 1);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

//Пункт 2

/\*qs(items, 0, count - 1);

start = clock();

shell(items, count);

qs(items, 0, count - 1);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

//Пункт 3

/\*shell\_minus(items, count);

start = clock();

shell(items, count);

qs(items, 0, count - 1);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

//Пункт 4

/\*qs(items, count / 2, count - 1);

shell\_minus(items, count / 2);

start = clock();

shell(items, count);

qs(items, 0, count - 1);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

//Пункт 5

/\*start = clock();

qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

/\*shell(items, count);

start = clock();

qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

/\*shell\_minus(items, count);

start = clock();

qsort(items, count, sizeof(int), (int(\*)(const void\*, const void\*))compare);

end = clock();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

/\*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();

float res = end - start;

printf("Время сортировки: %f\n", res / 1000);\*/

system("PAUSE");

return 0;

}

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

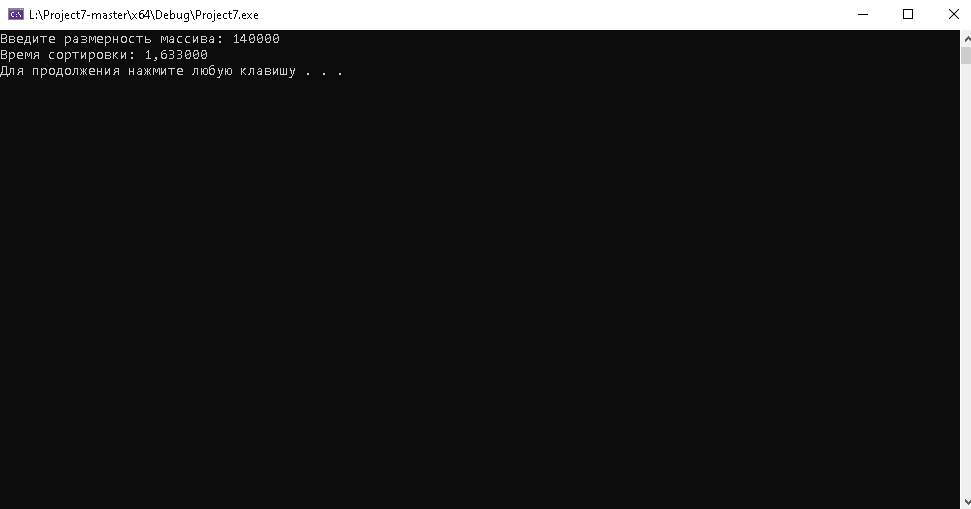


Рисунок 9 Сортировка шелла на случайном наборе значений массива

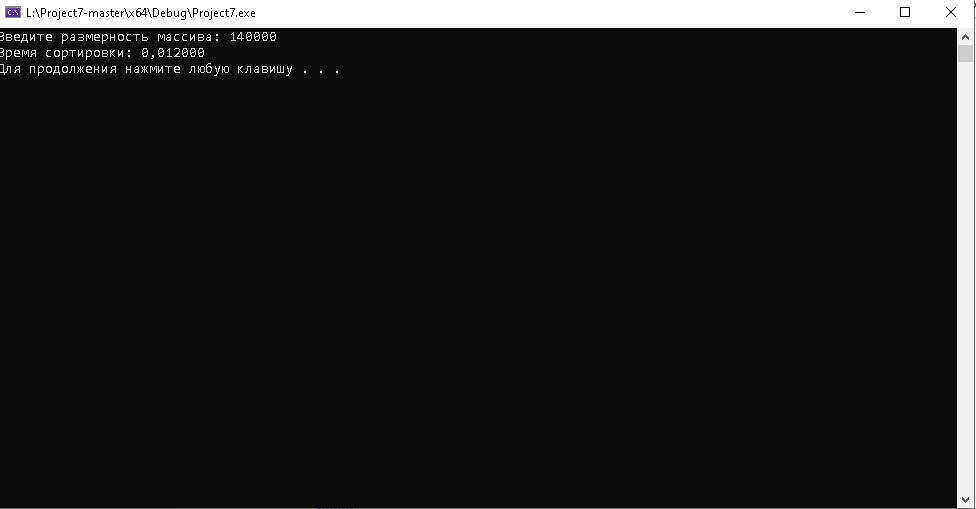


Рисунок 10 Быстрая сортировка на случайном наборе значений массива

2.Оценили время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

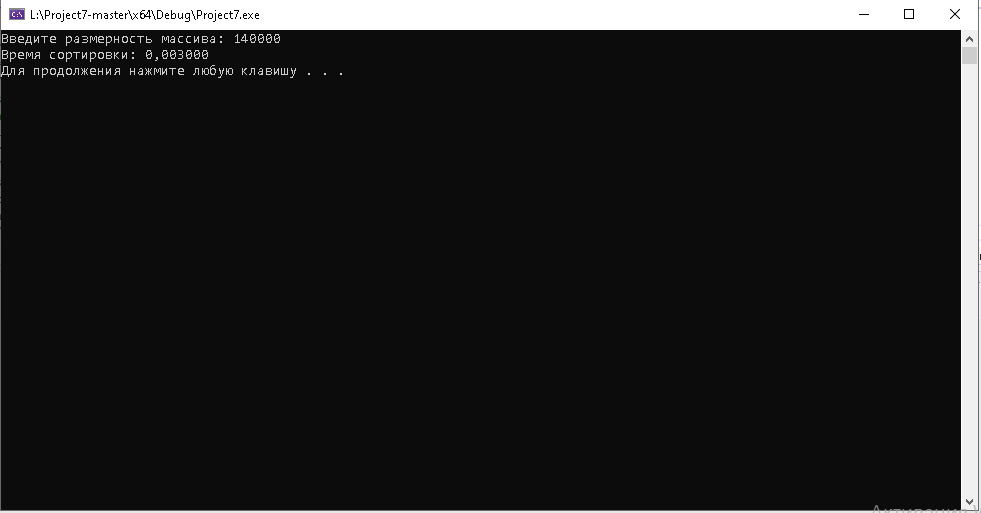


Рисунок 11 Сортировка шелла на массиве, представляющем собой возрастающую последовательность чисел

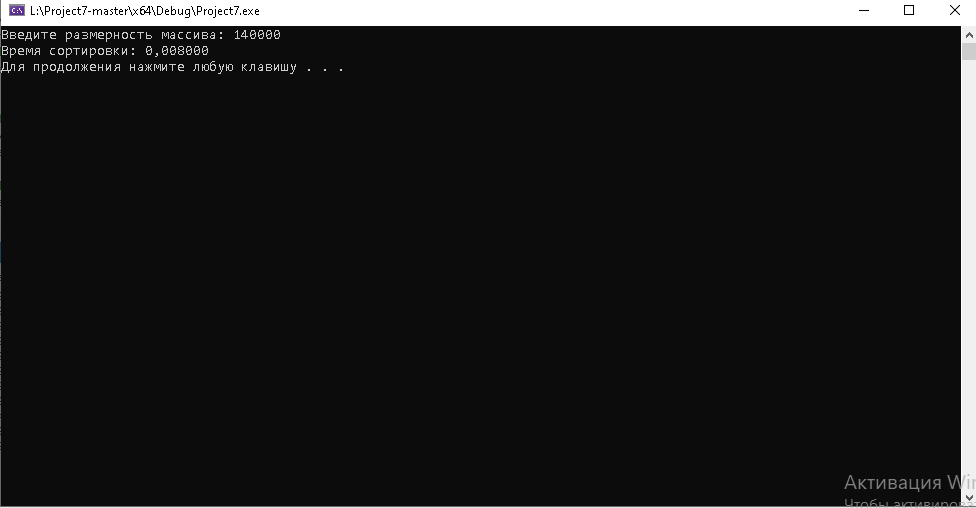


Рисунок 12 Быстрая сортировка на массиве, представляющем собой возрастающую последовательность чисел

3. Оценили время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

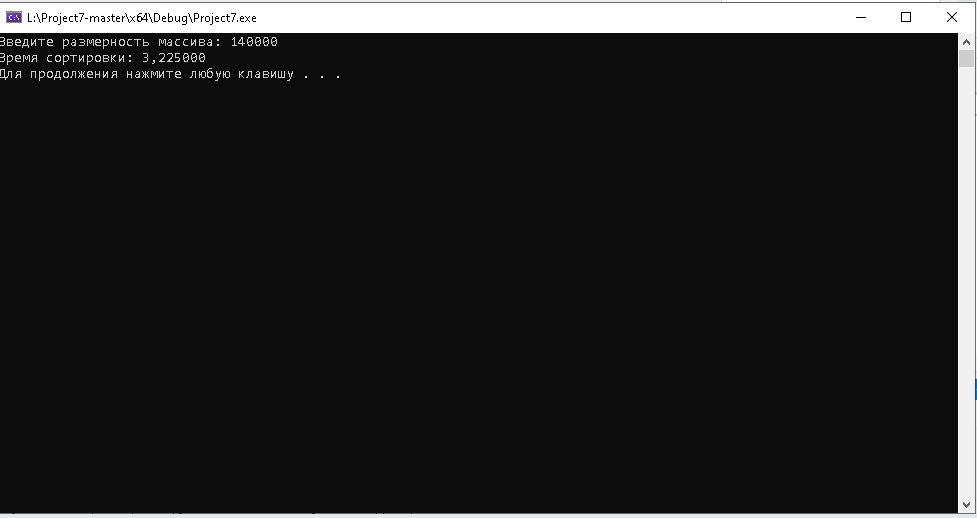


Рисунок 13 Сортировка Шелла на массиве, представляющем собой убывающую последовательность чисел

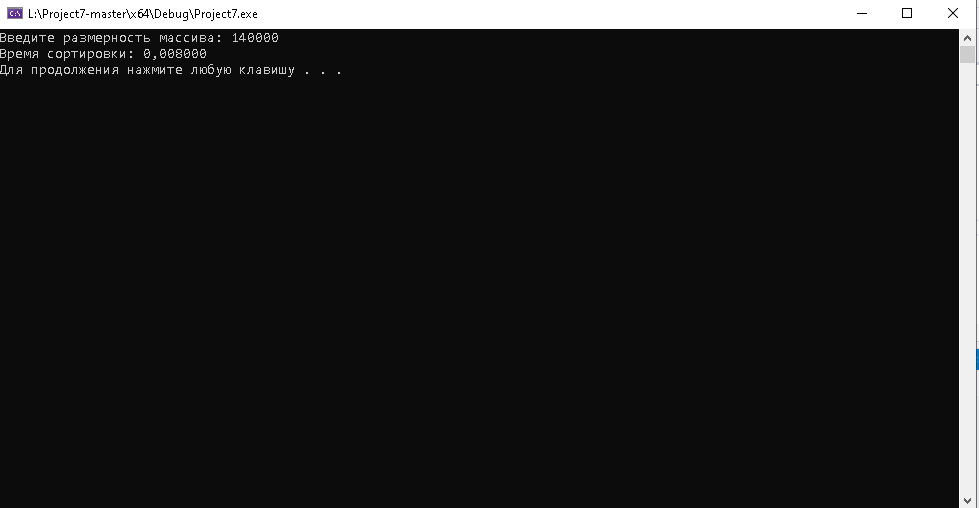


Рисунок 14 Пузырьковая сортировка на массиве, представляющем собой убывающую последовательность чисел

4. Оценили время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

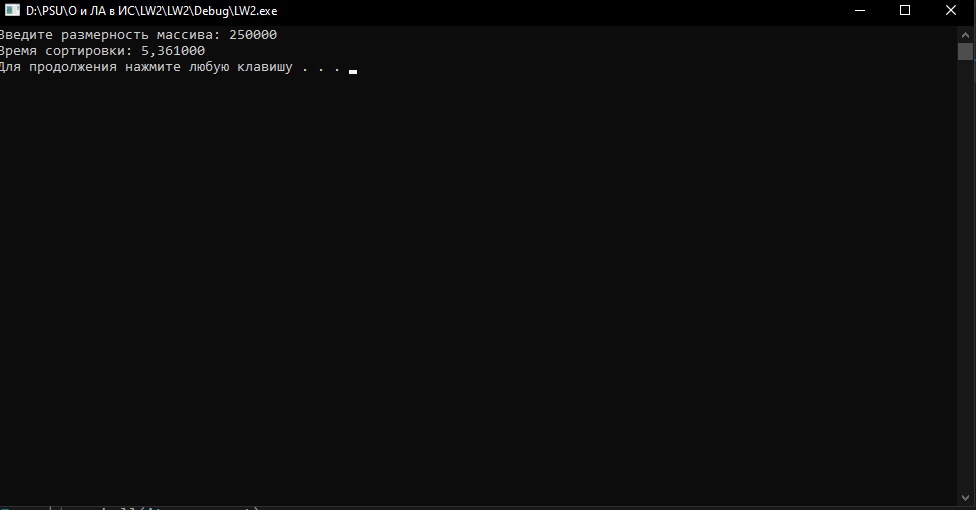


Рисунок 15 Сортировка Шелла на массиве, одна половина которого представляет собой возрастающую последовательность, а вторая,-убывающую

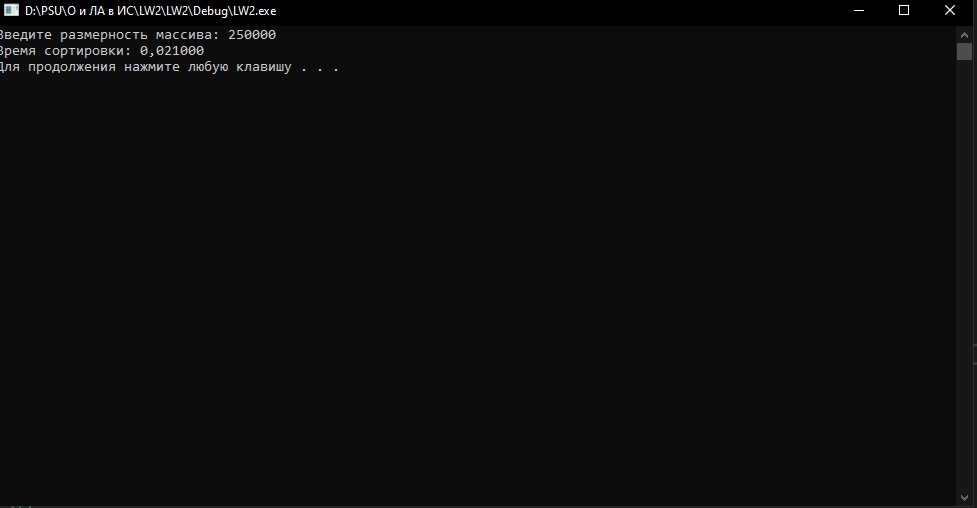


Рисунок 16 Быстрая сортировка на массиве, одна половина которого представляет собой возрастающую последовательность, а вторая,-убывающую

5. Оценили время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

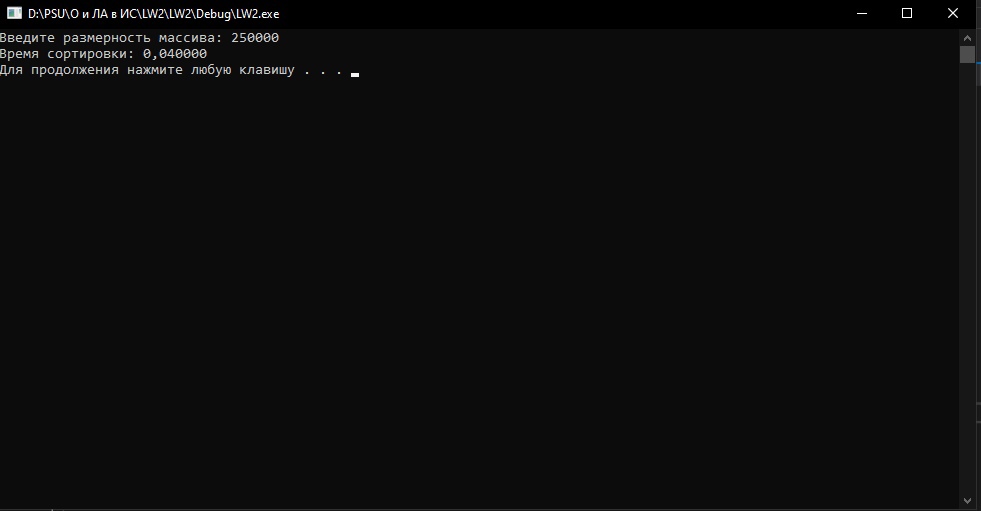


Рисунок 17 Работа функции qsort на случайном наборе значений массива

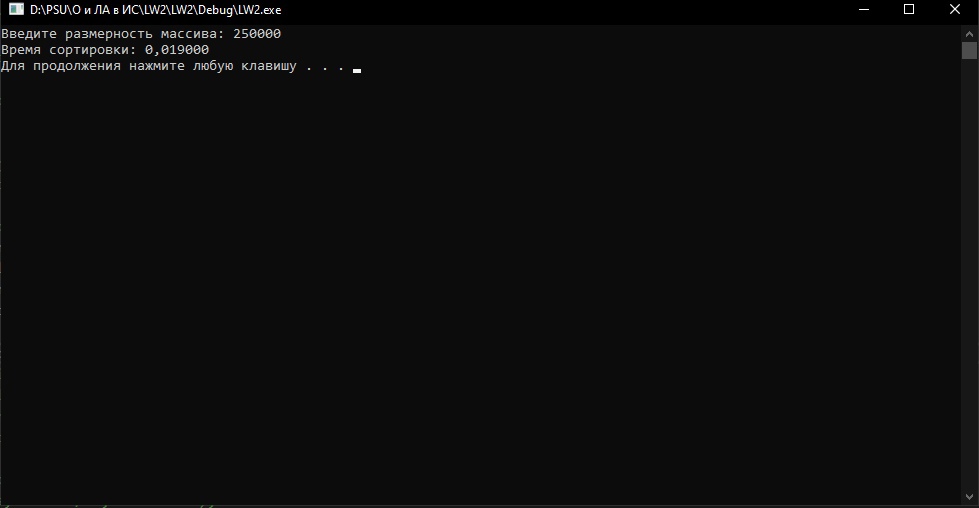


Рисунок 18 Работа функции qsort на массиве, представляющем собой возрастающую последовательность чисел

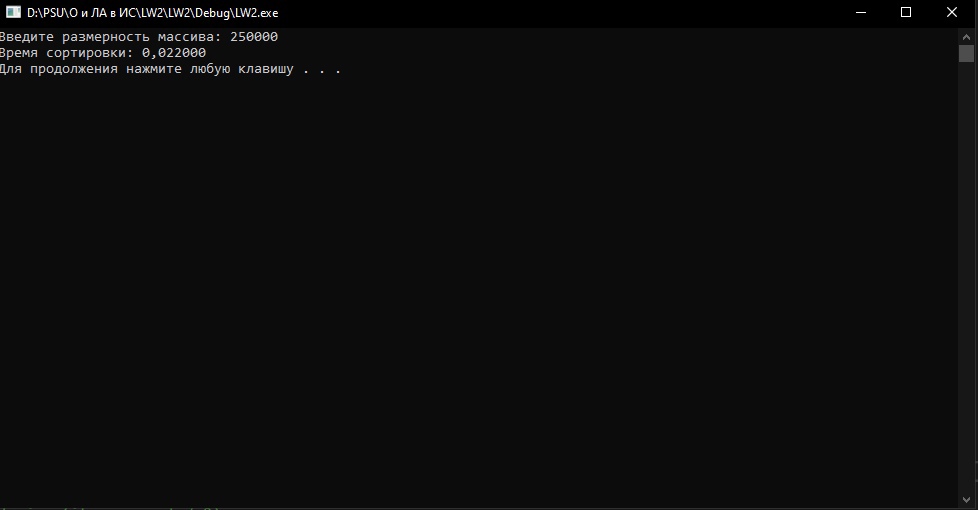


Рисунок 19 Работа функции qsort на массиве, представляющем собой убывающую последовательность чисел

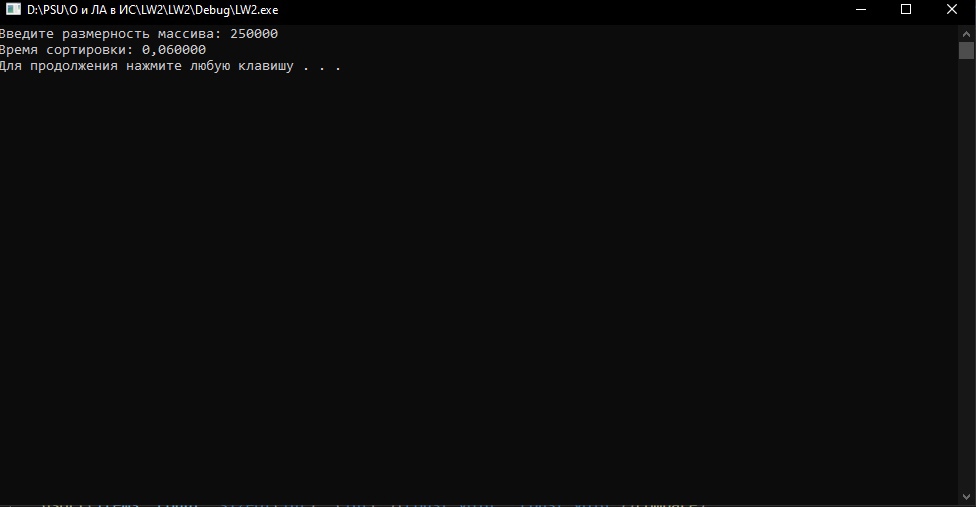


Рисунок 20 Работа функции qsort на массиве, одна половина которого представляет собой возрастающую последовательность, а вторая,-убывающую

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название сортировки | Случайный набор | Возрастающий | Убывающий | Пилообразный |
| Шелла | 1,633000 | 0,003000 | 3,225000 | 3,199 |
| Быстрая | 0,012000 | 0,008000 | 0,008000 | 0,015 |
| qsort | 0,063 | 0,038 | 0,041 | 0,069 |

**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(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.