Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №1

# по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Простые структуры данных»

**Выполнили:**

студенты группы 20ВВ3

Китаев Андрей

Новиков Иван

Халилов Алмаз

**Приняли:**

Юрова О.В.

Митрохин М.А.

Пеза 2021

# **Название**

**Оценка времени выполнения программ**

**Цель работы** – изучение библиотеки time.h для оценки времени выполнения программы.

# **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.

Типы данных:

1. clock\_t - возвращается функцией clock(). Обычно определён как int или long int.

2. time\_t - возвращается функцией time(). Обычно определён как int или long int.

3. struct tm - нелинейное, дискретное календарное представление времени.

Основные функции:

1. clock\_t clock(void) - возвращает время, измеряемое процессором в тактах от начала выполнения программы, или −1, если оно не известно. Пересчет этого времени в секунды выполняется по формуле:

clock() / CLOCKS\_PER\_SEC

где CLOCKS\_PER\_SEC – константа, определяющая количество тактов системных часов в секунду.

2. time\_t time(time\_t \*tp)

Возвращает текущее календарное время или −1, если это время не известно. Если указатель tp не равен NULL, то возвращаемое значение записывается также и в \*tp.

3. double difftime(time\_t time2,time\_t time1)

Возвращает разность time2-time1, выраженную в секундах.

# **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.

**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.

**Ход Работы**

**Листинг**

#define \_CRT\_NONSTDC\_NO\_WARNINGS

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

void multiply()

{

setlocale(LC\_ALL, "RUS");

int i = 0, j = 0, r = 0;

int elem\_c;

clock\_t start, end;

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

int\*\* a = 0;

int\*\* b = 0;

int\*\* c = 0;

int size;

printf("Введите размер матриц: ");

scanf("%d", &size);

start = clock();

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

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

{

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

}

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

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

{

b[i] = (int\*)malloc(size \* sizeof(int));

}

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

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

{

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

}

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

{

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

{

a[i][j] = rand() % 99;

}

}

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

{

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

{

b[i][j] = rand() % 99;

}

}

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;

}

}

}

end = clock();

float time = end - start;

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

system("pause");

}

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 comp(const int\* a, const int\* b)

{

return \*a - \*b;

}

int main()

{

setlocale(LC\_ALL, "RUS");

int variant;

system("cls"); // очищаем экран

printf("Главное меню\n\n");

printf("Введите номер действия:\n");

printf("1. Умножение матриц\n");

printf("2. Сортировка Шелла\n");

printf("3. Быстрая сортировка\n");

printf("4. Быстрая сортировка(из ст. библиотеки)\n");

printf("5. Выход\n");

printf(">");

do {

scanf("%d", &variant);

switch (variant) {

case 1:

{

system("cls");

multiply();

}

break;

case 2:

{

system("cls");

clock\_t start, end;

int count;

int data;

printf("Сортировка Шелла\n\n");

printf("Введите размерность массива: ");

scanf("%d", &count);

int\* items = (int\*)malloc(count \* sizeof(items));

printf("По возр. или по уб.:\n");

printf("1. По возрастанию\n");

printf("2. По убыванию\n");

printf("3. Случайный набор\n");

printf("4. Пила\n");

printf(">");

scanf("%d", &data);

switch (data)

{

case 1:

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

{

items[i] = i;

}

break;

case 2:

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

{

items[i] = count - i;

}

break;

case 3:

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

{

items[i] = rand() % 100;

}

break;

case 4:

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

{

if (i <= count / 2) {

items[i] = i;

}

else {

items[i] = count - i;

}

}

break;

}

start = clock();

shell(items, count);

end = clock();

float time = end - start;

printf("Время выполнения работы программы=%.10f с\n", time / CLOCKS\_PER\_SEC);

}

break;

case 3:

{

system("cls");

clock\_t start, end;

int count;

int data;

printf("Сортировка Быстрая\n\n");

printf("Введите размерность массива: ");

scanf("%d", &count);

int\* items = (int\*)malloc(count \* sizeof(items));

printf("По возр. или по уб.:\n");

printf("1. По возрастанию\n");

printf("2. По убыванию\n");

printf("3. Случайный набор\n");

printf("4. Пила\n");

printf(">");

scanf("%d", &data);

switch (data)

{

case 1:

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

{

items[i] = i;

}

break;

case 2:

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

{

items[i] = count - i;

}

break;

case 3:

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

{

items[i] = rand() % 100;

}

break;

case 4:

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

{

if (i < count / 2) {

items[i] = i;

}

else {

items[i] = count - i;

}

}

start = clock();

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

end = clock();

float time = end - start;

printf("Время выполнения работы программы=%.10f с\n", time / CLOCKS\_PER\_SEC);

exit(0);

}

start = clock();

qs(items, 0, count - 1);

end = clock();

float time = end - start;

printf("Время выполнения работы программы=%.10f с\n", time / CLOCKS\_PER\_SEC);

}

break;

case 4:

{

system("cls");

clock\_t start, end;

int count;

int data;

printf("Сортировка Qsort\n\n");

printf("Введите размерность массива: ");

scanf("%d", &count);

int\* items = (int\*)malloc(count \* sizeof(items));

printf("По возр. или по уб.:\n");

printf("1. По возрастанию\n");

printf("2. По убыванию\n");

printf("3. Случайный набор\n");

printf("4. Пила\n");

printf(">");

scanf("%d", &data);

switch (data)

{

case 1:

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

{

items[i] = i;

}

break;

case 2:

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

{

items[i] = count - i;

}

break;

case 3:

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

{

items[i] = rand() % 100;

}

break;

case 4:

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

{

if (i < count / 2) {

items[i] = i;

}

else {

items[i] = count - i;

}

}

break;

}

start = clock();

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

end = clock();

float time = end - start;

printf("Время выполнения работы программы=%.10f с\n", time / CLOCKS\_PER\_SEC);

}

break;

case 5:

{

exit(0);

}

break;

}

if (variant != 5)

system("pause"); // задерживаем выполнение, чтобы пользователь мог увидеть результат выполнения выбранного пункта

} while (variant != 5);

}

**Задание 1:**

1. Вычислили порядок сложности алгоритма:
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.

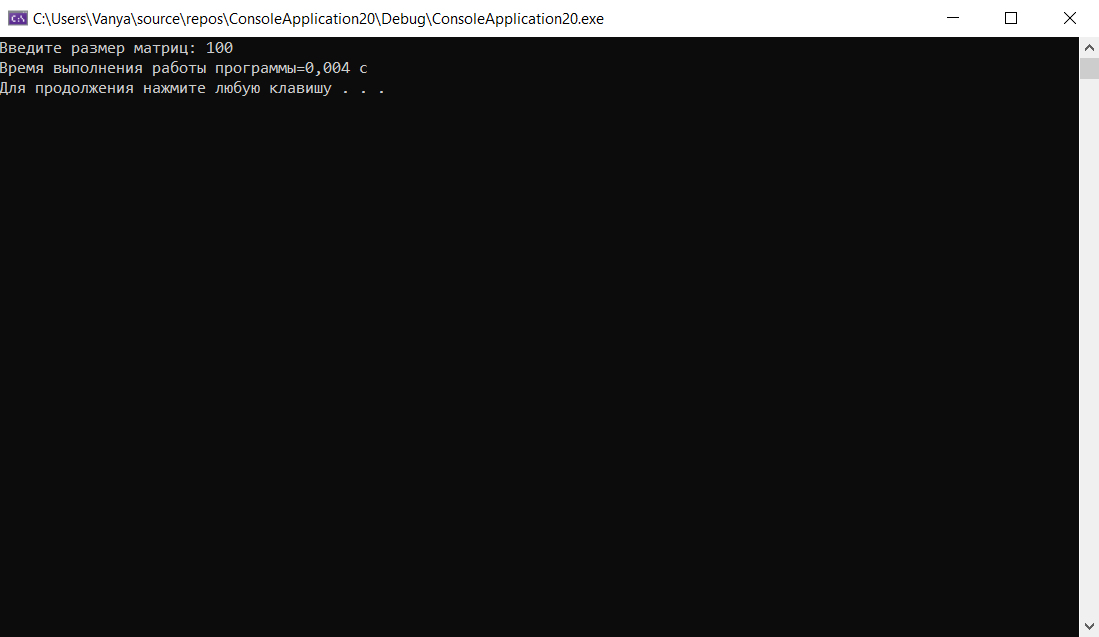


Рисунок 1. Результат работы программы для матрицы размерами 100

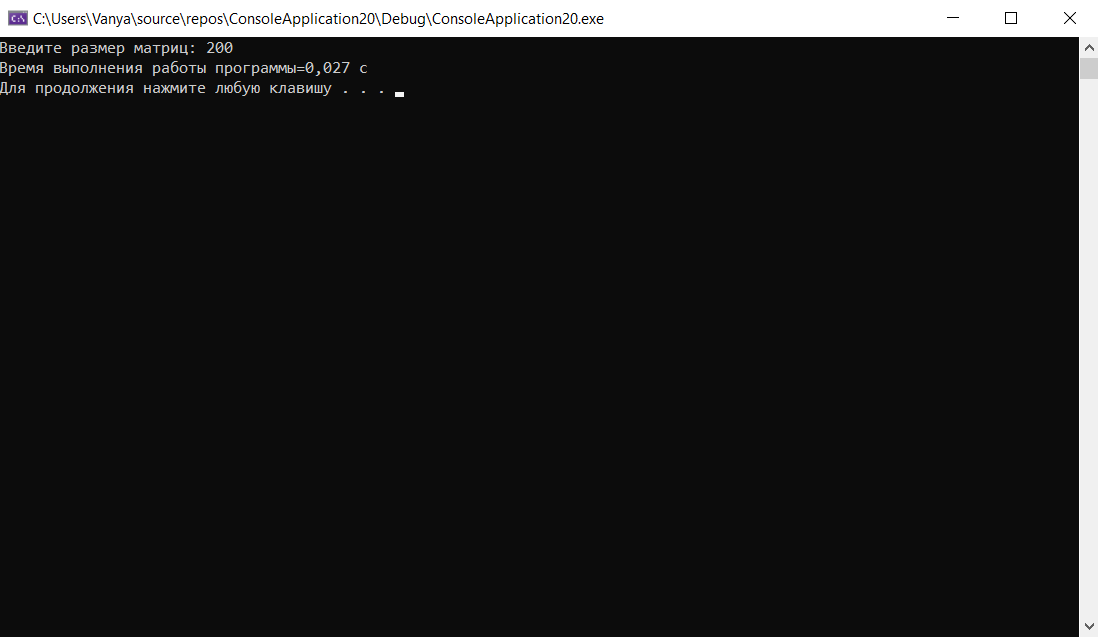


Рисунок 2. Результат работы программы для матрицы размерами 200

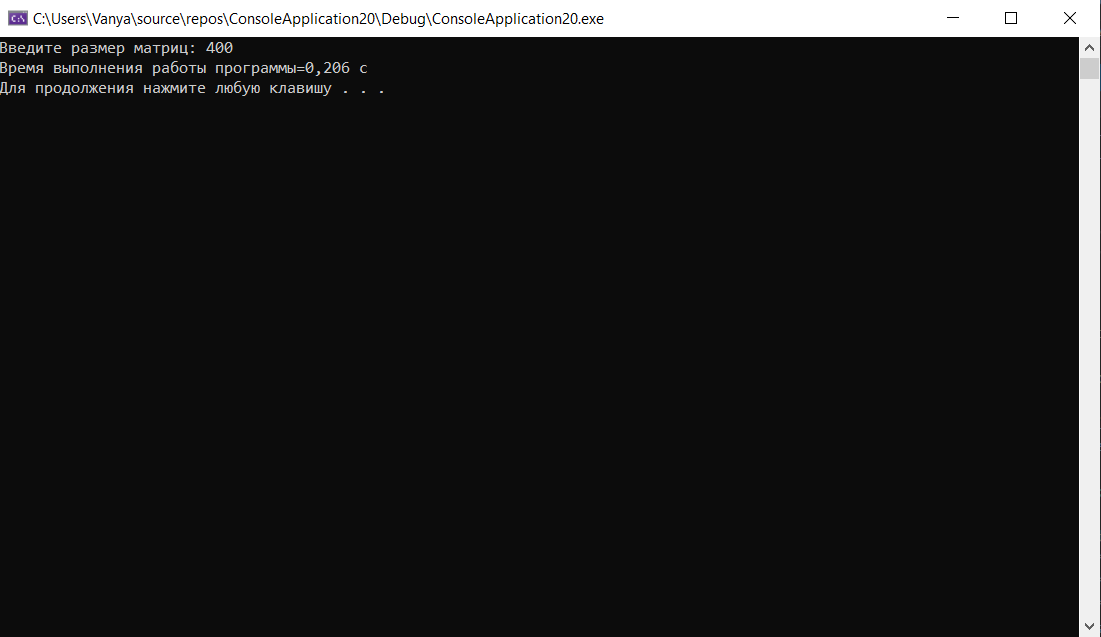


Рисунок 3. Результат работы программы для матрицы размерами 400

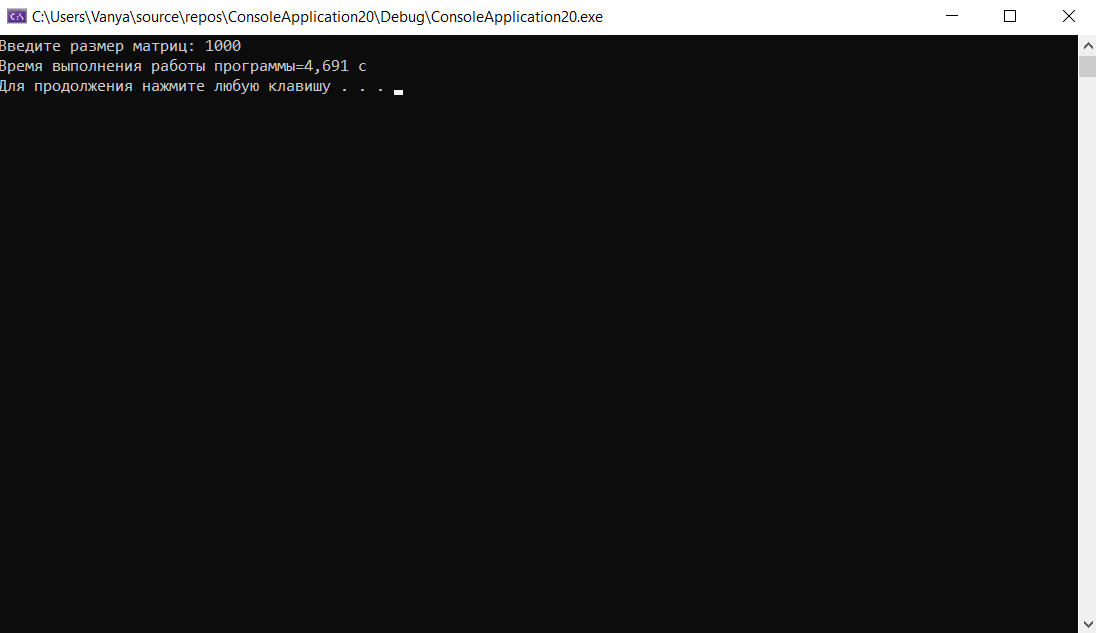


Рисунок 4. Результат работы программы для матрицы размерами 1000

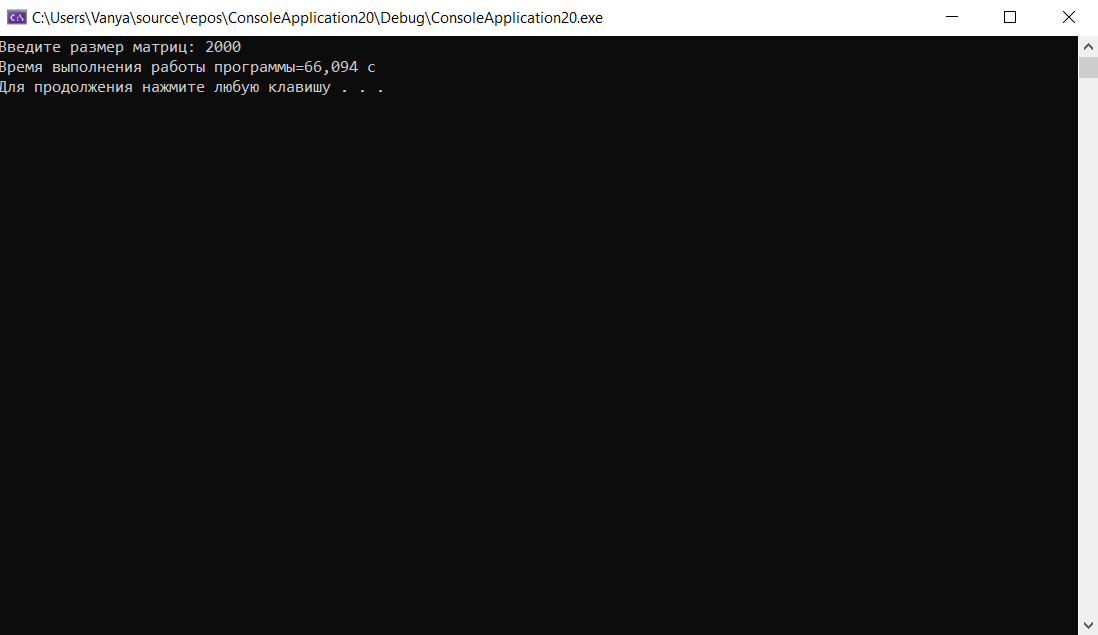


Рисунок 5. Результат работы программы для матрицы размерами 2000

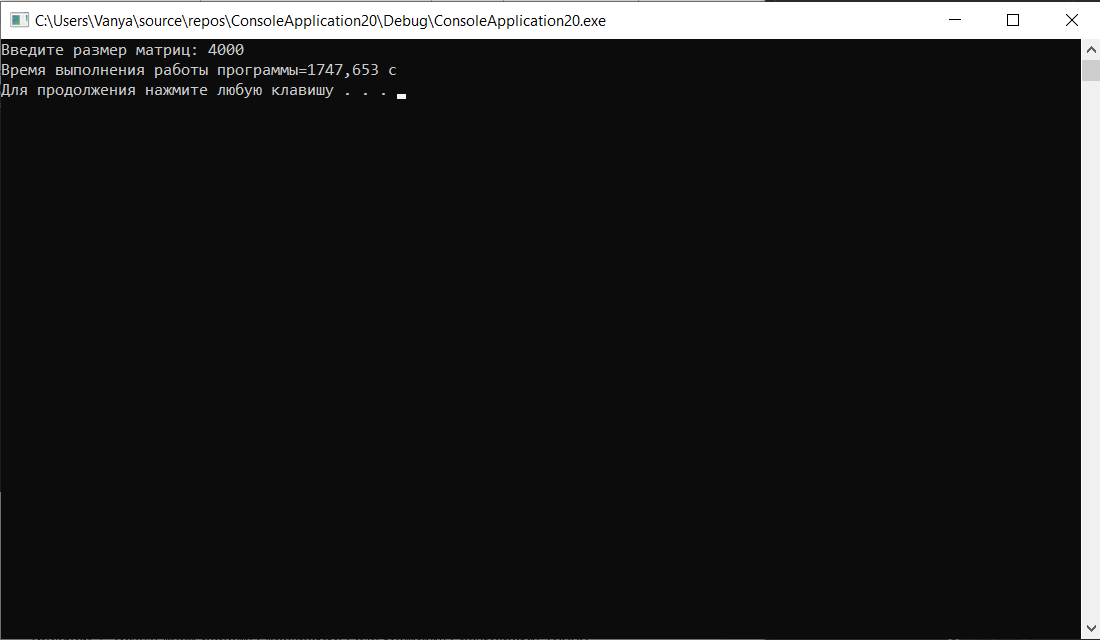
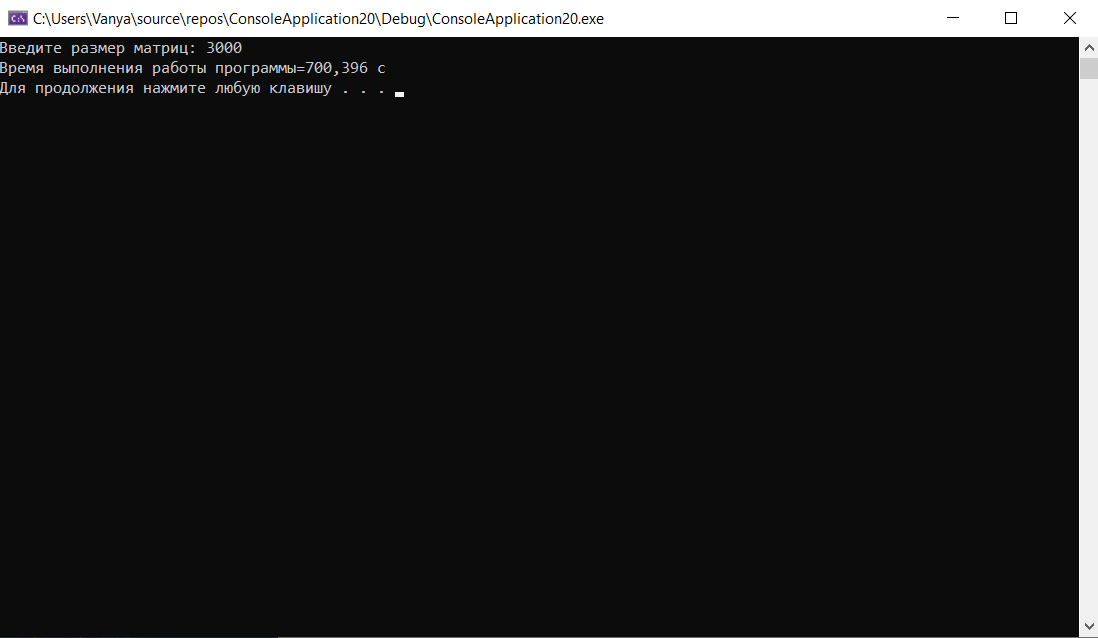
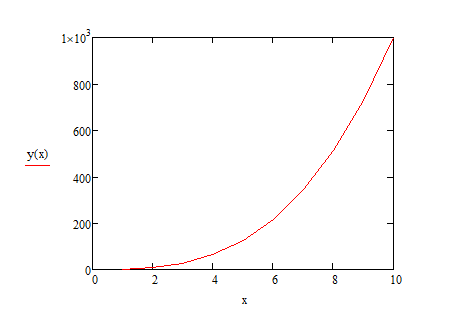


Рисунок 6. Результат работы программы для матрицы размерами 4000

Рисунок 7. Результат работы программы для матрицы размерами 3000

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

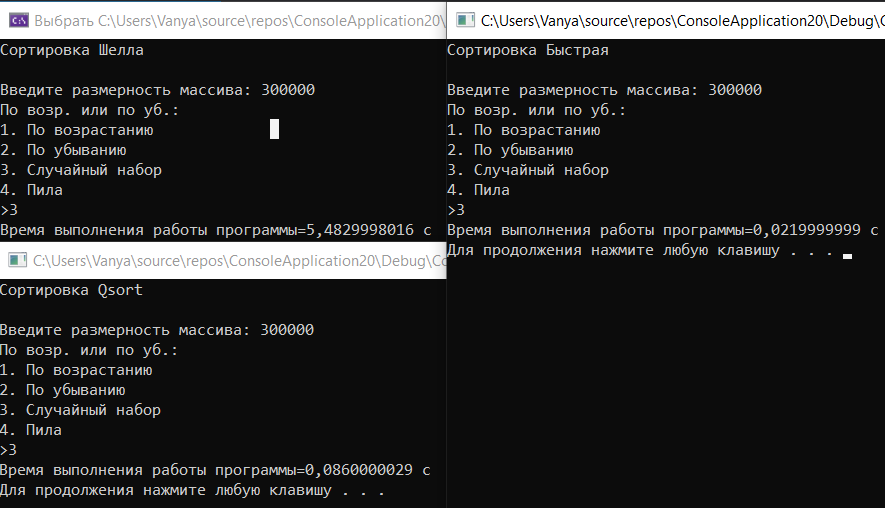


*Рисунок 8. График функции y(x) = x3*

Графики совпадают.

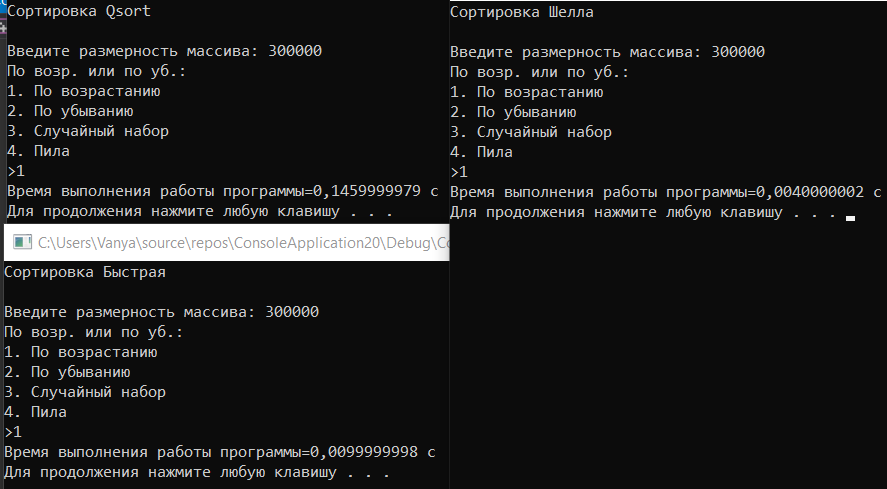
**Задание 2:**

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



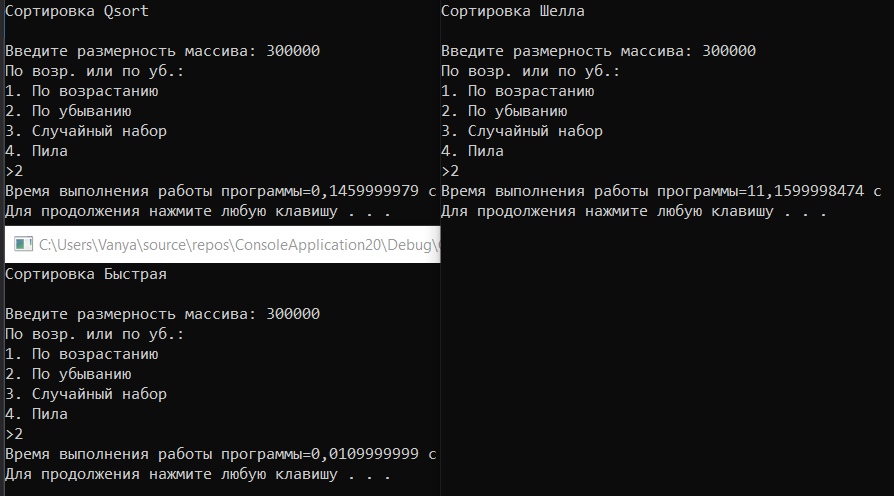
*Рисунок 9. Результаты работы для случ. набора.*

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



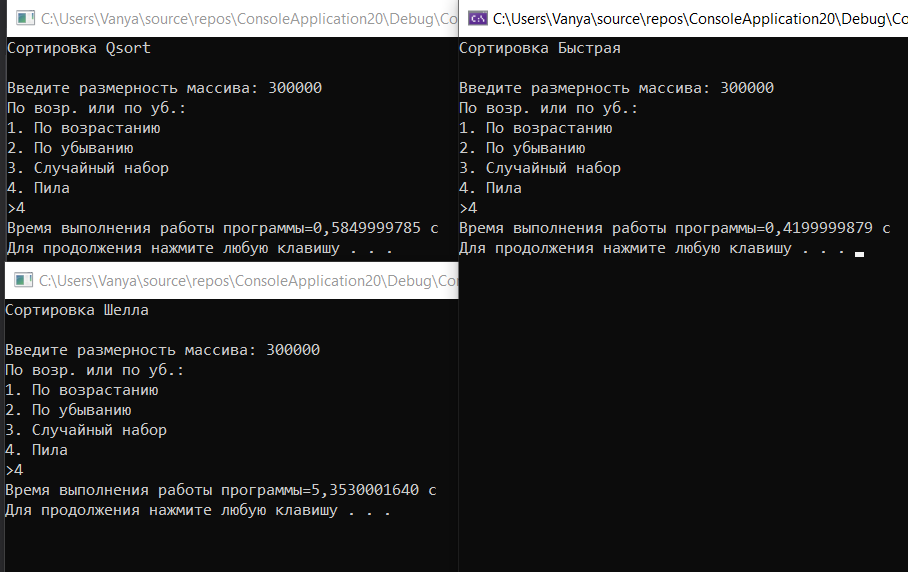
*Рисунок 10. Результаты работы для возраст. набора.*

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



*Рисунок 11. Результаты работы для уб. набора.*

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



*Рисунок 12. Результаты работы для воз. - уб. набора.*

1. Оценили время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных. (см. Рисунки 8 - 11)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайный | Возрастающий | Убывающий | Возрастающий-  убывающий |
| Шелл | 5,482 | 0,004 | 11,159 | 5.353 |
| Быстрая | 0,021 | 0,009 | 0,010 | 0.419 |
| qsort | 0,086 | 0,145 | 0.145 | 0.584 |

**Вывод:** изучили библиотеку time.h для оценки времени выполнения программы, научились определять сложность алгоритма программы, работать с различными методами сортировки.

Эффективность:

При случайном наборе, убывающем и возрастающе-

убывающем самая эффективная сортировка – быстрая.

При возрастающей последовательности – сортировка Шелла.