Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**Выполнил:**

Студентка группы 19ВВ2:

Дерябина А.

**Приняли:**

Митрохин М.А.

Юрова О.В.

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»»

Пенза 2020

### Название:

### Оценка времени выполнения программ.

### Задание:

1. Вычислить порядок сложности программы (O-символику).
2. Оценить время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 50, 100, 150, 200, 250.
3. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.
4. написать программу, вычисляющую сумму значений в каждом столбце (или строке) двумерного массива.
5. написать программу, осуществляющую поиск среди структур student структуру с заданными параметрами (фамилией, именем и т.д.).

### Цель работы:

Разработать программы по данным заданиям.

**Общие сведения:**

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 longint.

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

3. structtm - нелинейное, дискретное календарное представление времени.

Основные функции:

1. clock\_tclock(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.

### Описание метода решения задачи:

**Задача 1:**

1. Вычисляем порядок сложности программы (O-символику)

2. Оцениваем время выполнения программы и кода, выполняющего перемножение матриц, используя функции библиотеки time.h для матриц размерами от 50, 100, 150, 200, 250.

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

**Задача 2:**

1. Оцениваем время работы каждого из реализованных алгоритмов на случайном наборе значений массива.  
2. Оценили время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.  
3. Оценили время работы каждого из реализованных алгоритмов 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.  
5. Оценили время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**График:**

### Листинг:

**Задание 1:**

Порядок сложности программы O(n­­3).

#include "stdafx.h"

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int main(void)

{

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

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

FILE\* S;

char name[] = "data.txt";

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

const unsigned int h = 150;

int i = 0, j = 0, r;

int a[h][h], b[h][h], c[h][h], elem\_c;

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

while (i < h)

{

while (j < h)

{

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

j++;

}

i++;

}

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

i = 0; j = 0;

while (i < h)

{

while (j < h)

{

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

j++;

}

i++;

}

start = clock();

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

end = clock();

double seconds = (double)(end - start) / CLOCKS\_PER\_SEC;

printf("Function execution time - %fms\n\n", seconds);

S = fopen(name, "a");

fprintf(S, "Function execution time - %fms arr[%d]\n\n", seconds, h);

fclose(S);

system("pause");

return(0);

}

**Задание 2:**

#include "stdafx.h"

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

int compare(const void \* x1, const void \* x2)

{

return(\*(int\*)x1 - \*(int\*)x2);

}

int main(void)

{

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

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

FILE\* S;

char name[] = "data2.txt";

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

int arr[100000];

srand(time(NULL));

for (int i = 0; i< 100000; i++)

{

arr[i] = rand() % 100;

}

int i, j, step;

int tmp;

start = clock();

for (step = 100000 / 2; step > 0; step /= 2)

{

for (i = step; i<100000; i++)

{

tmp = arr[i];

for (j = i; j >= step; j -= step)

{

if (tmp<arr[j - step])

arr[j] = arr[j - step];

else

break;

}

arr[j] = tmp;

}

}

end = clock();

printf("\n\ntime start - %fms\n\n", start/1000.0);

printf("time end - %fms\n\n", end/1000.0);

printf("\nFunction execution time - %fms\n\n", (end - start)/1000.0);

printf("Program execution time - %fms\n\n", clock()/1000.0);

S = fopen(name, "a");

fprintf(S, "\n\ntime start - %fms\n\n", start/1000.0);

fprintf(S, "time end - %fms\n\n", end/1000.0);

fprintf(S, "\nFunction execution time - %fms\n\n", (end - start)/1000.0);

fprintf(S, "Program execution time - %fms\n\n", clock()/1000.0);

fclose(S);

system("pause");

return(0);

}

### Результаты работы программы:

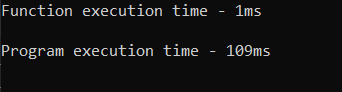


Рисунок 1 - Время выполнения программы и кода для матрицы размером 50

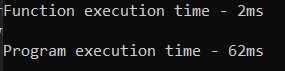


Рисунок 2 - Время выполнения программы и кода для матрицы размером 100

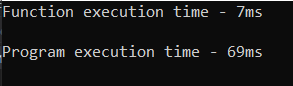


Рисунок 3 - Время выполнения программы и кода для матрицы размером 150

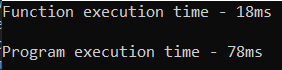


Рисунок 4 - Время выполнения программы и кода для матрицы размером 200

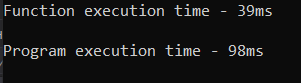


Рисунок 5 - Время выполнения программы и кода для матрицы размером 250

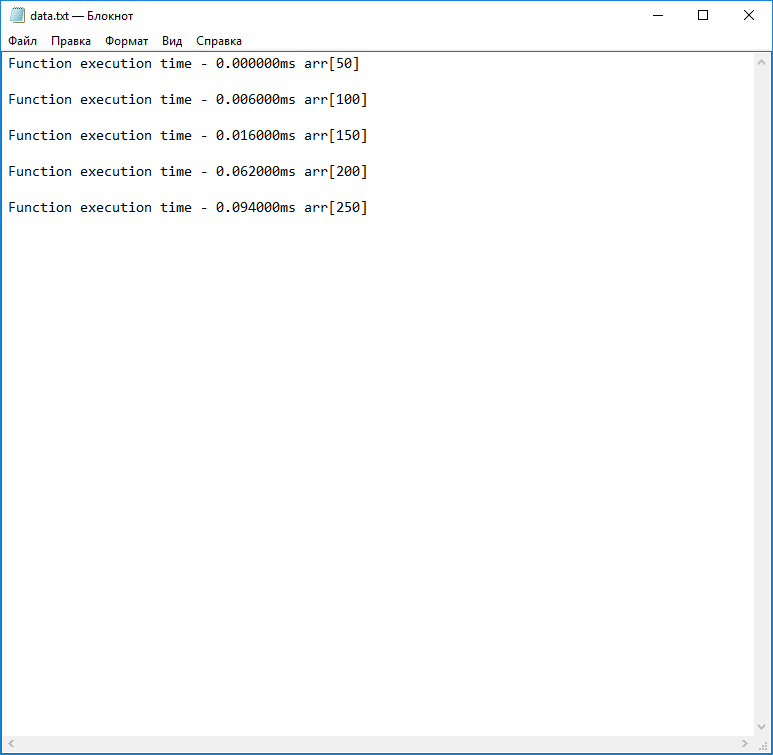


Рисунок 6 - Результаты записанные в файл

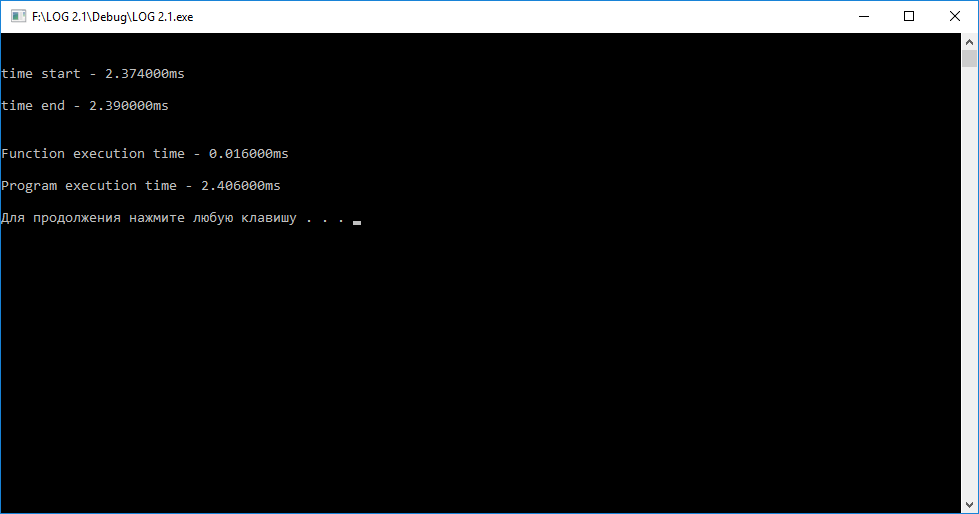


Рисунок 7 - Время работы алгоритма на случайном наборе элементов

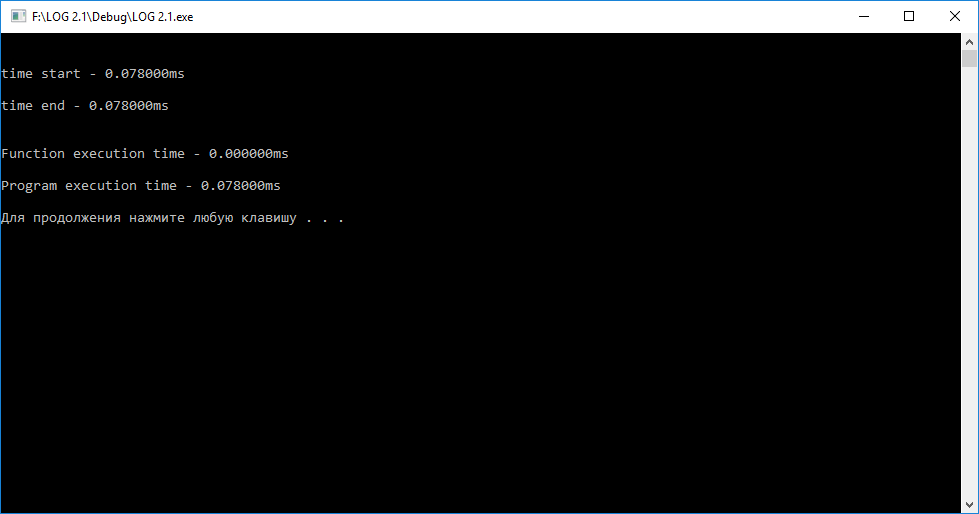


Рисунок 8 - Время работы алгоритма на возрастающем наборе чисел

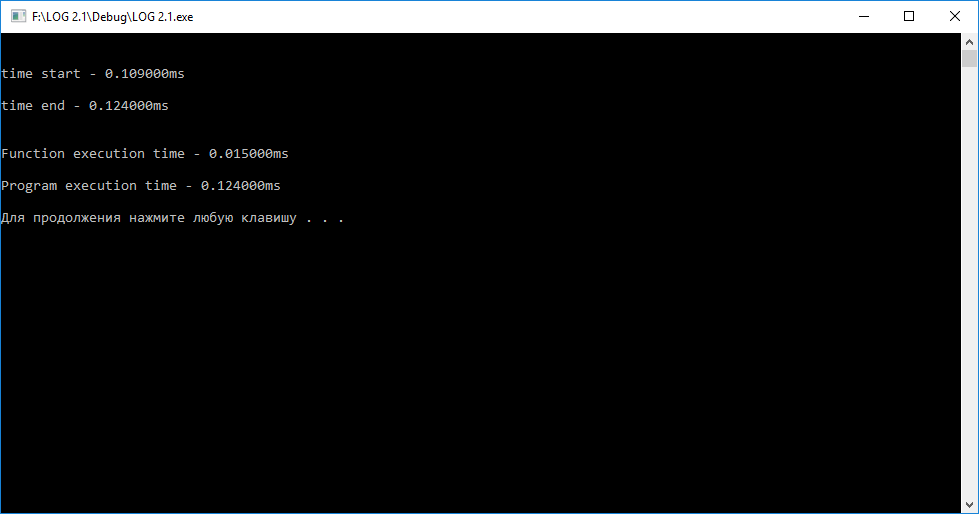


Рисунок 9 - Время работы алгоритма на убывающем наборе чисел

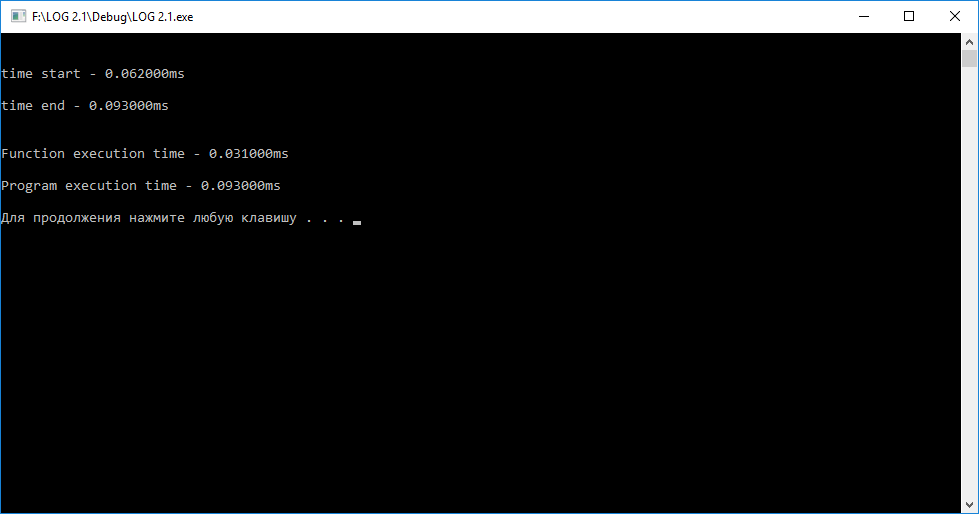


Рисунок 10 - Время работы алгоритма когда одна половина возрастающий набор чисел, а другая половина убывающий набор чисел

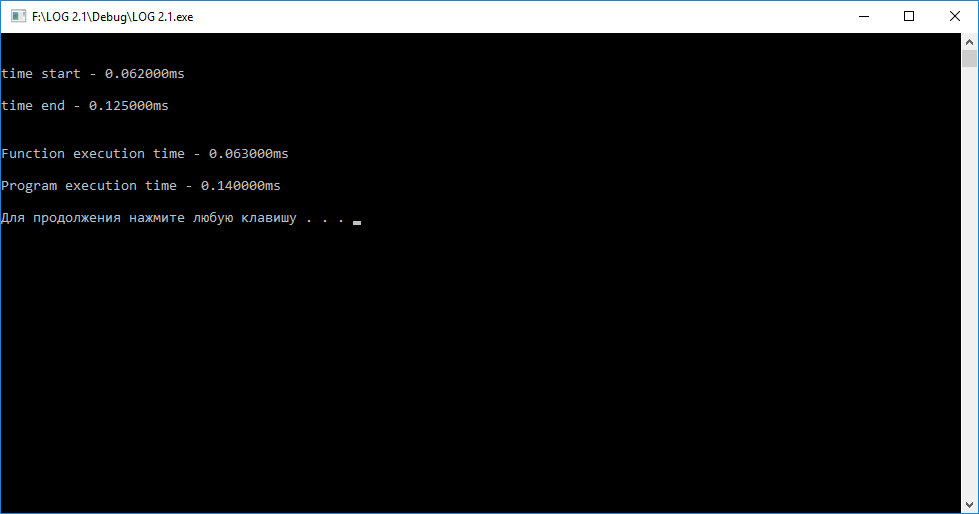


Рисунок 11 - Время работы стандартной функции qsort

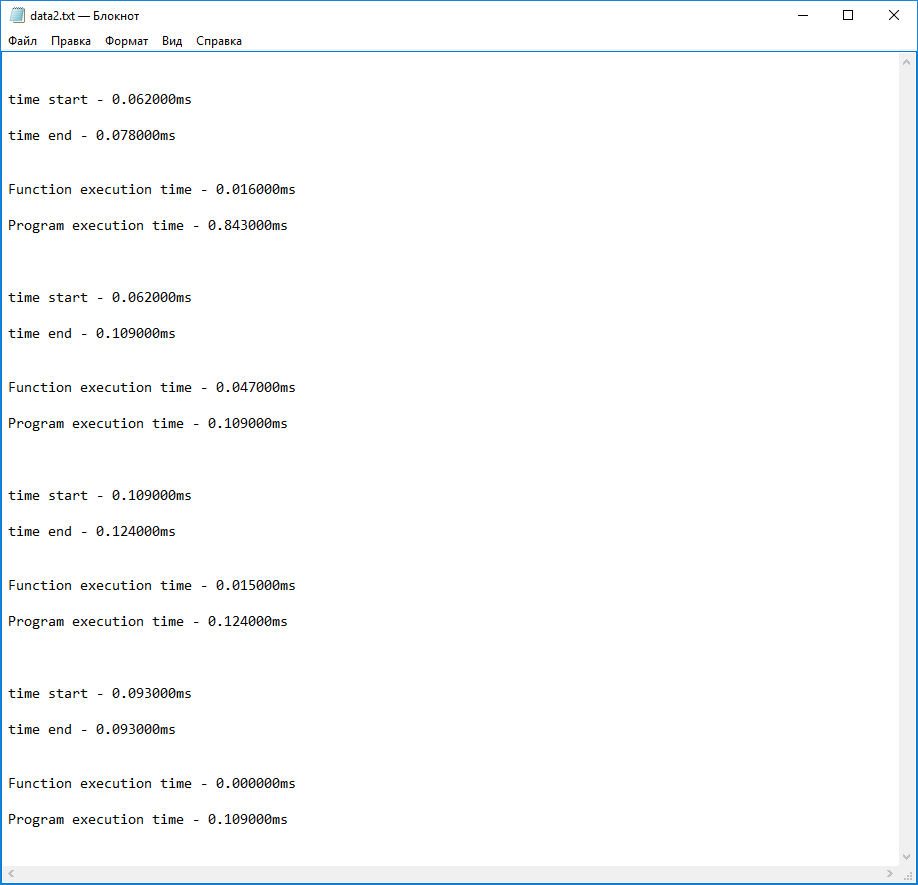


Рисунок 12 - Результаты записанные в файл

### Вывод:

В ходе выполнения лабораторной работы мы убедились в том, что время сортировки зависит от того, в каком порядке расположены элементы массива и количества элементов. Убедились, что если элементы изначально отсортированы, то время сортировки будет минимальным. Были разработаны программы, соответствующие данным заданиям.