**Министерство науки и высшего образования Российской Федерации**

**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 студ. группы 24ВВВ4:

Агуреев А.П.

Невежин И.С.

Приняли:

к.э.н. доцент Акифьев И.В.\_\_\_\_

к.т.н. доцент Юрова О.В.\_\_\_\_

Пенза 2025

**Цель работы:**

Получение практических навыков использования функций для измерения и оценки времени выполнения программы и ее отдельных частей.

**Лабораторные задания:**

**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.

**Листинг:**

#include <stdio.h>

#include <time.h>

#include <stdlib.h>

#include <iostream>

using namespace std;

//#define ARR\_SIZE 1000

#define LEIGHT 10000

void first\_task() {

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

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

int ARR\_SIZE[] = {100, 200, 300, 320, 340, 350, 360};

for (int z = 0; z < (sizeof(ARR\_SIZE) / sizeof(ARR\_SIZE[0])); z++) {

start\_code = clock();

int\*\* a = (int\*\*)malloc(ARR\_SIZE[z] \* sizeof(int\*));

for (int i = 0; i < ARR\_SIZE[z]; i++) {

a[i] = (int\*)malloc(ARR\_SIZE[z] \* sizeof(int));

for (int j = 0; j < ARR\_SIZE[z]; j++) {

a[i][j] = rand() % 100 + 1;

}

}

int\*\* b = (int\*\*)malloc(ARR\_SIZE[z] \* sizeof(int\*));

for (int i = 0; i < ARR\_SIZE[z]; i++) {

b[i] = (int\*)malloc(ARR\_SIZE[z] \* sizeof(int));

for (int j = 0; j < ARR\_SIZE[z]; j++) {

b[i][j] = rand() % 100 + 1;

}

}

int\*\* c = (int\*\*)malloc(ARR\_SIZE[z] \* sizeof(int\*));

for (int i = 0; i < ARR\_SIZE[z]; i++) {

c[i] = (int\*)malloc(ARR\_SIZE[z] \* sizeof(int));

for (int j = 0; j < ARR\_SIZE[z]; j++) {

c[i][j] = 0;

}

}

start = clock();

for (int i = 0; i < ARR\_SIZE[z]; i++) {

for (int j = 0; j < ARR\_SIZE[z]; j++) {

for (int r = 0; r < ARR\_SIZE[z]; r++) {

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

}

}

}

end = clock();

for (int i = 0; i < ARR\_SIZE[z]; i++) {

free(a[i]); free(b[i]); free(c[i]);

}

free(a); free(b); free(c);

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

double time\_spent\_code = (double)(end - start\_code) / CLOCKS\_PER\_SEC;

cout << "Для матрицы размером "<<ARR\_SIZE[z] << " элементов:\n" << "Время выполнения программы: " << time\_spent\_code << "\nВремя выполнения умножения матриц: " << time\_spent << "\n\n";

}

}

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 void\* a, const void\* b) {

return (\*(int\*)a) - (\*(int\*)b);

}

void second\_task() {

int point, arr[LEIGHT], arr\_2[LEIGHT], arr\_3[LEIGHT];

clock\_t start, end;

do {

cout << "Введите пункт задания (1-4), 0 - назад: \n";

cin >> point;

if (point < 1 || point > 4) {

if (point == 0)

continue;

cout << "Такого пункта нет!\n";

continue;

}

switch (point) {

case 1:

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

arr[i] = rand() % 10000;

arr\_2[i] = arr[i];

arr\_3[i] = arr[i];

}

break;

case 2:

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

arr[i] = i + 1;

arr\_2[i] = arr[i];

arr\_3[i] = arr[i];

}

break;

case 3:

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

arr[i] = LEIGHT - i;

arr\_2[i] = arr[i];

arr\_3[i] = arr[i];

}

break;

case 4:

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

if (i < (LEIGHT / 2)) {

arr[i] = i + 1;

arr\_2[i] = arr[i];

arr\_3[i] = arr[i];

}

else {

arr[i] = LEIGHT - i;

arr\_2[i] = arr[i];

arr\_3[i] = arr[i];

}

}

break;

}

start = clock();

shell(arr, LEIGHT);

end = clock();

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

cout << "Время работы функции SHELL: " << time\_spend;

start = clock();

qs(arr\_2, 0, LEIGHT - 1);

end = clock();

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

cout << "\nВремя работы функции QS: " << time\_spend << '\n';

start = clock();

qsort(arr\_3, LEIGHT, sizeof(int), comp);

end = clock();

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

cout << "Время работы функции qsort: " << time\_spend << '\n';

} while (point != 0);

}

int main(void)

{

setlocale(LC\_ALL, "Russian");

unsigned short number;

bool flag = 1;

while (flag != 0) {

cout << "Введите номер задания:\n";

cin >> number;

switch (number) {

case 1:

first\_task();

break;

case 2:

second\_task();

break;

}

cout << "\nВведите 1, чтобы продолжить выбирать задания (0, чтобы выйти):\n";

cin >> flag;

}

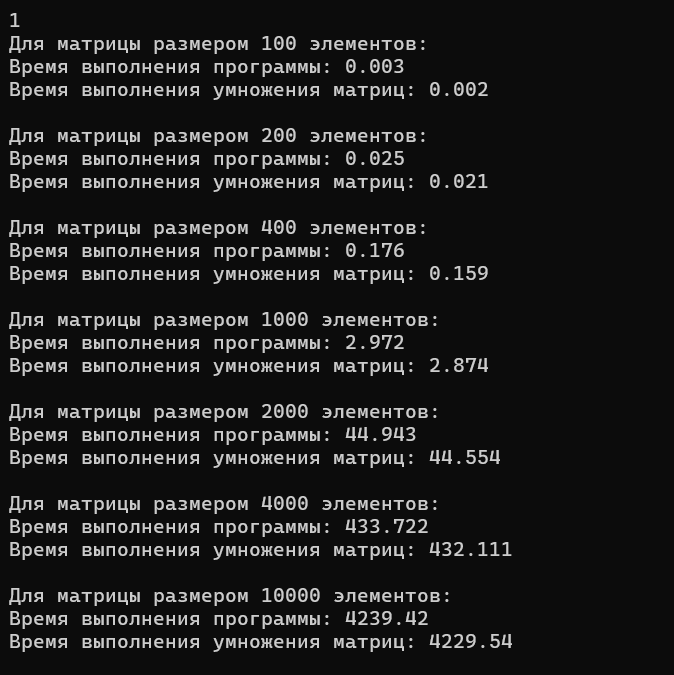
return(0);

}

**Практическая часть:**

**Задание 1:**

1. Порядок сложности программы определили как O(n^3), так как здесь реализован алгоритм перемножения 2 матриц, который реализуется 3 вложенными циклами.



|  |  |  |
| --- | --- | --- |
| Размер матриц (эл) | t работы программы (с) | t работы умножения (с) |
| 100 | 0.003 | 0.002 |
| 200 | 0.025 | 0.021 |
| 400 | 0.176 | 0.159 |
| 1000 | 2.972 | 2.874 |
| 2000 | 44.943 | 44.554 |
| 4000 | 433.722 | 432.111 |
| 10000 | 4239.42 | 4229.54 |

1. График зависимости времени от размера матриц.

**Вывод к 1 заданию:**

Исходя из опыта, можно сказать, что разница между временем выполнения программы и выполнения алгоритма с порядком сложности O(n^3), то есть перемножением 2 матриц, минимально. То есть время выполнения всей программы полностью зависит от этого участка кода. На графике видно, что с увеличением размера матрицы растёт и время выполнения программы: при количестве элементов свыше 2000 время резко увеличивается, с этого момента время выполнения программы с каждым дальнейшим повышением размерности увеличивается примерно в 10 раз.

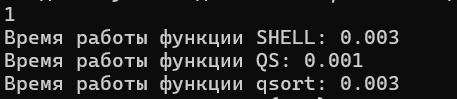
**Задание 2:**

1. Оценили время работы каждого из реализованных алгоритмов на случайном наборе значений массива. (Первый опыт: массив из 10 тыс. элементов, числа от 0 до 10000; Второй опыт: массив из 5 тыс. элементов, числа от 0 до 5000).

1)  
Изображение выглядит как текст, Шрифт, снимок экрана, черный

Содержимое, созданное искусственным интеллектом, может быть неверным.

2)



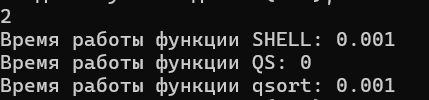
1. Оценили время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

1)

Изображение выглядит как текст, Шрифт, снимок экрана, типография

Содержимое, созданное искусственным интеллектом, может быть неверным.

2)



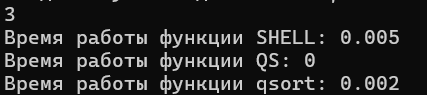
1. Оценили время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

1)

Изображение выглядит как текст, Шрифт, снимок экрана, черный

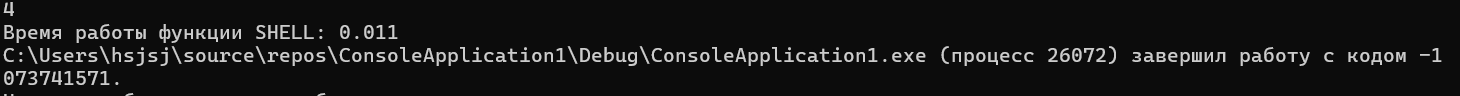
Содержимое, созданное искусственным интеллектом, может быть неверным.

2)

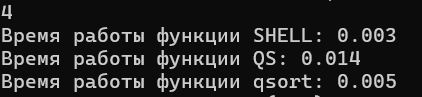


1. Оценили время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, - убывающую.

1)



2)



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

**Таблица:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размер массива | Функция | Случайные элементы | Возрастающая последовательность | Убывающая последовательность | 1ая половина – возрастающая  2ая половина - убывающая |
| 10 тыс. эл. | SHELL | 0,011 | 0 | 0,021 | 0,011 |
| QS | 0,002 | 0,001 | 0,001 | ошибка |
| QSORT | 0,005 | 0,003 | 0,004 | ошибка |
| 5 тыс. эл. | SHELL | 0,003 | 0,001 | 0,005 | 0,003 |
| QS | 0,001 | 0 | 0 | 0,014 |
| QSORT | 0,003 | 0,001 | 0,002 | 0,005 |

**Вывод ко 2 заданию:**

Исходя из опыта, можно сказать, что эффективность функций сортировки зависит не только от размера обрабатываемого массива, но и от характера его заполнения. На таблице видно, что разница во времени работы программы исходя из размерности массива есть, и она не маленькая. Также видно проблему сортировки в массиве, который заполнялся так, что 1-ая половина – возрастает, а 2-ая половина убывает. Проблема заключается в функции QS, она попросту неэффективна при таком заполнении массива, так как работает сильно дольше других функций, и вдобавок она вызвала ошибку с массивом размерностью 10 тыс. элементов, при этом в остальных случаях она была одной и самых эффективных вариантов обработки. Это показывает, что не существует универсального метода сортировки, который был бы одинаково эффективен для всех типов входных данных. При выборе алгоритма сортировки необходимо учитывать как размер массива, так и особенности его заполнения, чтобы избежать падения производительности или ошибок.

**Вывод ко всей лабораторной работе:**

Получили практические навыки в использовании функций для измерения и оценки времени выполнения программы и ее отдельных частей.