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:**

Студенты группы 24ВВВ1

Куничкина В.А.

Суркова Д.А.

**Приняли:**

к.т.н. Деев М.В.

к.т.н. доцент Юрова О.В.

Пенза 2025

# Цель работы: Оценить время работы программ.

**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.

**Теоретическая часть:**

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.

**Практическая часть:**

**Задание 1.**

**Листинг:**

#include "stdafx.h"

#include <conio.h>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

int\*\* create\_matrix(int n) {

int\*\* matrix = (int\*\*)malloc(n \* sizeof(int\*));

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

matrix[i] = (int\*)malloc(n \* sizeof(int));

}

return matrix;

}

void free\_matrix(int\*\* matrix, int n) {

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

free(matrix[i]);

}

free(matrix);

}

void fill\_matrix(int\*\* matrix, int n) {

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

for (int j = 0; j < n; j++) {

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

}

}

}

int main(void) {

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

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

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

int sizes[] = { 100, 200, 300, 400, 500, 600, 700, 800, 900, 1000, 2000, 4000, 10000 };

int num\_sizes = sizeof(sizes) / sizeof(sizes[0]);

srand(time(NULL));

printf("Измерение времени перемножения матриц\n");

printf("=====================================\n\n");

for (int s = 0; s < num\_sizes; s++) {

int n = sizes[s];

printf("Размер матрицы: %d x %d\n", n, n);

int\*\* a = create\_matrix(n);

int\*\* b = create\_matrix(n);

int\*\* c = create\_matrix(n);

fill\_matrix(a, n);

fill\_matrix(b, n);

clock\_t start, end;

double cpu\_time\_used;

start = clock();

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

for (int j = 0; j < n; j++) {

int elem\_c = 0;

for (int r = 0; r < n; r++) {

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

}

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

}

if (n >= 4000 && (i + 1) % (n / 10) == 0) {

printf("Прогресс: %d%%\n", (i + 1) \* 100 / n);

}

}

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

int hours = (int)cpu\_time\_used / 3600;

int minutes = ((int)cpu\_time\_used % 3600) / 60;

double seconds = cpu\_time\_used - hours \* 3600 - minutes \* 60;

printf("Время перемножения: ");

if (hours > 0) printf("%d ч ", hours);

if (minutes > 0 || hours > 0) printf("%d мин ", minutes);

printf("%.2f сек\n", seconds);

free\_matrix(a, n);

free\_matrix(b, n);

free\_matrix(c, n);

printf("-----------------------------\n");

}

printf("\nВсе измерения завершены!\n");

\_getch();

return 0;

}

Сложность программы: О(n3)

Время выполнения:

От 100 эл. – 0.00 с

От 200 эл. – 0.01 с

От 400 эл. – 0.14 с

От 500 эл. – 0.25 с

От 600 эл. – 0.42 с

От 700 эл. – 0.73 с

От 800 эл. – 1.10 с

От 900 эл. – 1.70

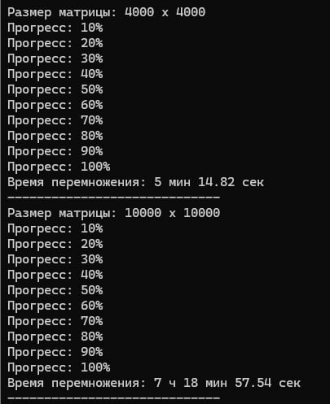
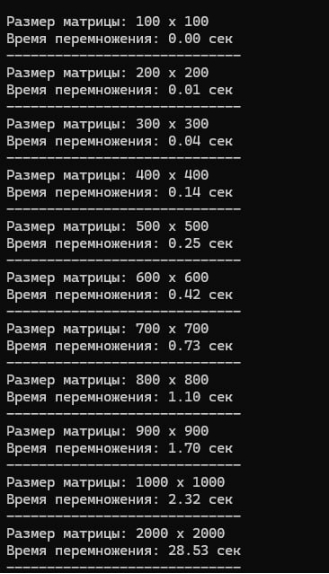
От 1000 эл. – 2.14 с

От 2000 эл. – 28.53 с

От 4000 эл. – 314.82 с

От 10000 эл.- 26337.54 с

**Результат работы программы:**



Построив диаграмму зависимости времени выполнения программы от размера матриц, мы подтверждаем то, что сложность данной программы: О(n3).

**Задание 2:**

**Листинг:**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <time.h>

#include <locale.h>

#include <conio.h>

#include <string.h>

#include <stdlib.h>

int m[10000];

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);

}

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;

}

}

}

int cmp(const void\* a, const void\* b) { //функция сравнения элементов массива

{

return \*(int\*)a - \*(int\*)b; // Если результат вычитания равен 0, то числа равны

}

}

int main()

{

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

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

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

//for (int i = 0; i < 10000; i++) { //Заполнение массива рандомными числами

// m[i] = rand() % 100 + 1;

//}

//for (int i = 1; i < 10000; i++) { //Возрастающая последовательность

// m[i] = m[i-1]+1;

//}

// for (int i = 0; i < 10000; i++) { //Убывающая последовательность

// m[i] = 10000-i;

// }

// for (int i = 0; i < 5000; i++) { //Возрастающе-убывающая последовательность

// m[i] = m[i - 1] + 1;

// }

// for (int i = 5; i < 10000; i++) {

// m[i] = 10000 - i;

// }

int start\_time = clock();

//shell(m, 10000); //Сортировка Шелла

//qs(m, 0, 10000 - 1); //Быстрая сортировка

qsort(m, 10000, sizeof(int), cmp); // Ф-я быстрой сортировки

int end\_time = clock();

int search\_time = end\_time - start\_time;

printf("%d", search\_time);

return (0);

}

Количество чисел в массиве: 10000. Время работы алгоритма измеряется в мс

Сортировка Шелла Быстрая сортировка Ф-я быстрой сортировки

Сл.набор.данных 10-16 0-2 3-4

Возраст.послед. 0 0 0

Убыв.посл. 20-24 0-1 4-7

Убыв-возраст. 20-23 0-1 7-10

30.000 чисел

Сортировка Шелла Быстрая сортировка Ф-я быстрой сортировки

Сл.набор.данных 130-206 4 14-19

Возраст.послед. 1-2 2-3 18-24

Убыв.посл. 271-296 2-4 18-27

Убыв-возраст. 250-264 2-5 31-38

**Вывод:**

1. Выяснили, что алгоритм быстрой сортировки, реализованный вручную, одинаково быстро работает на разных наборах данных в массиве
2. С увеличением количества данных в массиве, увеличивается, время, затрачиваемое на сортировку
3. Сортировка Шелла требует наибольше количество времени на работу, так как он сравнивает не только элементы, стоящие рядом, но и на определённом расстоянии
4. Функция быстрой сортировки из библиотеки требует большее количество времени, чем реализованный вручную