МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Оценка времени выполнения программ"

Выполнили:

Студенты группы 24ВВВ3

Плотников И.А.

Виноградов Б.С.

Приняли:

Деев М.В.

Юрова О. В.

Пенза 2025

**Цель**

Оценка времени выполнения и сложности 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.

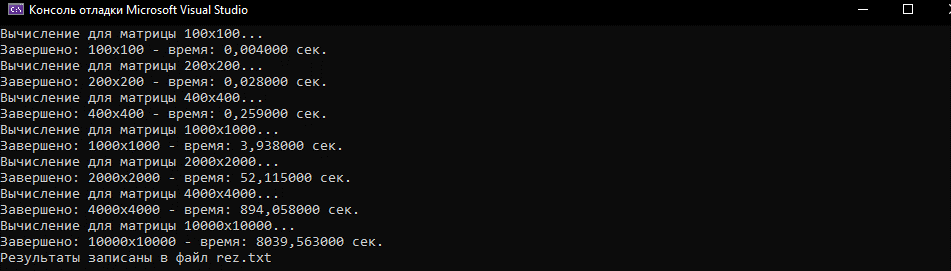
**Пояснительный текст к программам**

Первая программа предназначена для измерения времени выполнения умножения двух квадратных матриц различных размеров. Результаты замеров сохраняются в текстовый файл rez.txt и выводятся на экран.

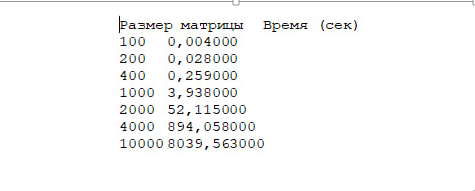
Вторая программа предназначена для измерения времени работы алгоритмов сортировки данных на разных наборах входных данных. Результаты замеров сохраняются в текстовый файл rez2.txt и выводятся на экран.

**Результаты программ**

1 Рис. - Результат работы **ex1\_lab2.c**



2 Рис. – Файл **rez.txt**



3 Рис. - Результат работы **ex2\_lab2.c**

Изображение выглядит как текст, снимок экрана, программное обеспечение, Шрифт

Содержимое, созданное искусственным интеллектом, может быть неверным.

4 Рис. – Файл **rez2.txt**Изображение выглядит как текст, снимок экрана, программное обеспечение, Мультимедийное программное обеспечение

Содержимое, созданное искусственным интеллектом, может быть неверным.

**1. Вычисление порядка сложности программы (О-символика)**

Главный "тяжёлый" участок программы — это умножение двух матриц размера n x n в функции multiplyArr.

Умножение матриц в коде реализовано тройным вложенным циклом:

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

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

c[i][j] = 0;

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

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

}

}

}

* Внешний цикл i выполняется n раз.
* Второй цикл j выполняется n раз.
* Внутренний цикл k выполняется n раз.

Это значит, что временная сложность алгоритма — **O(n³)**.

**2. Оценка времени выполнения программы и кода, выполняющего перемножение**

|  |  |
| --- | --- |
| **Размер матрицы** | **Время выполнения, (с)** |
| 100 х 100 | 0,004 |
| 200 х 200 | 0,028 |
| 400 х 400 | 0,259 |
| 1000 х 1000 | 3,938 |
| 2000 х 2000 | 52,115 |
| 4000 х 4000 | 894,058 |
| 10000 х 10000 | 8039,563 |

1. **График зависимости времени выполнения операции**
2. **Оценка времени выполнения алгоритмов сортировки**

|  |  |  |  |
| --- | --- | --- | --- |
| **Алгоритм**  **Набор**  **данных** | **shell** | **qs** | **std** |
| **Случайный** | **0.531948** | **0.007970** | **0.006791** |
| **Возрастающий** | **0.000984** | **0.001599** | **0.000366** |
| **Убывающий** | **1.074149** | **0.001666** | **0.001546** |
| **Смешанный** | **0.534566** | **1.559812** | **0.003458** |

**Листинг**

**Файл ex1\_lab2.c**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void clearScreen();

int\*\* createArr(int n);

void freeArr(int\*\* arr, int n);

void fillArr(int\*\* arr, int n);

double multiplyArr(int n);

void runCalculations();

int main() {

clearScreen();

runCalculations();

return 0;

}

void clearScreen() {

#ifdef \_WIN32

system("cls");

#else

system("clear");

#endif

}

int\*\* createArr(int n) {

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

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

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

}

return arr;

}

void freeArr(int\*\* arr, int n) {

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

free(arr[i]);

}

free(arr);

}

void fillArr(int\*\* arr, int n) {

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

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

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

}

}

}

double multiplyArr(int n) {

int\*\* a = createArr(n);

int\*\* b = createArr(n);

int\*\* c = createArr(n);

fillArr(a, n);

fillArr(b, n);

clock\_t start = clock();

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

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

c[i][j] = 0;

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

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

}

}

}

clock\_t end = clock();

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

freeArr(a, n);

freeArr(b, n);

freeArr(c, n);

return timeTaken;

}

void runCalculations() {

int sizes[] = { 100, 200, 400, 1000, 2000, 4000, 10000 };

int numSize = sizeof(sizes) / sizeof(sizes[0]);

FILE\* file = fopen("rez.txt", "w");

if (file == NULL) {

printf("Ошибка открытия файла!\n");

return;

}

fprintf(file, "Размер матрицы\tВремя (сек)\n");

srand(time(NULL));

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

int n = sizes[i];

printf("Вычисление для матрицы %dx%d...\n", n, n);

double timeTaken = multiplyArr(n);

fprintf(file, "%d\t%.6f\n", n, timeTaken);

printf("Завершено: %dx%d - время: %.6f сек.\n", n, n, timeTaken);

}

fclose(file);

printf("Результаты записаны в файл rez.txt\n");

}

**Файл ex2\_lab2.c**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <string.h>

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) {

if (left >= right) {

return;

}

while (left < right) {

int i = left;

int j = right;

int pivot = items[(left + right) / 2];

while (i <= j) {

while (items[i] < pivot) i++;

while (items[j] > pivot) j--;

if (i <= j) {

int temp = items[i];

items[i] = items[j];

items[j] = temp;

i++;

j--;

}

}

if (j - left < right - i) {

qs(items, left, j);

left = i;

}

else {

qs(items, i, right);

right = j;

}

}

}

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

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

}

void generateRandom(int\* arr, int n) {

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

arr[i] = rand() % 10000;

}

}

void generateAscending(int\* arr, int n) {

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

arr[i] = i;

}

}

void generateDescending(int\* arr, int n) {

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

arr[i] = n - i;

}

}

void generateMixed(int\* arr, int n) {

for (int i = 0; i < n / 2; i++) {

arr[i] = i;

}

for (int i = n / 2; i < n; i++) {

arr[i] = n - i;

}

}

int\* copyArray(const int\* source, int n) {

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

if (copy == NULL) {

printf("Ошибка выделения памяти!\n");

exit(1);

}

memcpy(copy, source, n \* sizeof(int));

return copy;

}

double measureShellTime(int\* arr, int n) {

int\* temp = copyArray(arr, n);

clock\_t start = clock();

shell(temp, n);

clock\_t end = clock();

free(temp);

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

}

double measureQsTime(int\* arr, int n) {

int\* temp = copyArray(arr, n);

clock\_t start = clock();

qs(temp, 0, n - 1);

clock\_t end = clock();

free(temp);

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

}

double measureStdQsortTime(int\* arr, int n) {

int\* temp = copyArray(arr, n);

clock\_t start = clock();

qsort(temp, n, sizeof(int), compare);

clock\_t end = clock();

free(temp);

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

}

int main() {

const int n = 100000;

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

if (originalArray == NULL) {

printf("Ошибка выделения памяти!\n");

return 1;

}

const char\* arrayTypes[] = {

"Случайный",

"Возрастающий",

"Убывающий",

"Смешанный"

};

void (\*generators[])(int\*, int) = {

generateRandom,

generateAscending,

generateDescending,

generateMixed

};

FILE\* file = fopen("rez2.txt", "w");

if (file == NULL) {

printf("Ошибка открытия файла!\n");

free(originalArray);

return 1;

}

fprintf(file, "Сортировка Шелла Быстрая сортировка Стандартная qsort\n");

srand(time(NULL));

for (int i = 0; i < 4; i++) {

printf("Тестирование на массиве: %s\n", arrayTypes[i]);

generators[i](originalArray, n);

double timeShell = measureShellTime(originalArray, n);

double timeQs = measureQsTime(originalArray, n);

double timeStdQsort = measureStdQsortTime(originalArray, n);

fprintf(file, " %.6f\t\t%.6f\t\t%.6f\n",

timeShell, timeQs, timeStdQsort);

printf("Результаты: Shell: %.6f, QS: %.6f, Std: %.6f\n\n",

timeShell, timeQs, timeStdQsort);

}

fclose(file);

free(originalArray);

printf("Результаты записаны в файл rez2.txt\n");

return 0;

}

**Вывод:**

В ходе выполнения лабораторной работы были разработаны программы для выполнения заданий Лабораторной работы №2. В процессе выполнения работы была произведена оценка сложности программы и выявлена зависимость скорости выполнения от исходного набора данных.