**Министерство науки и высшего образования РФ**

**Пензенский государственный университет**

**Кафедра «Вычислительная техника»**

**ОТЧЁТ**

По лабораторной работе №2

По дисциплине «Логика и основы алгоритмизации в инженерных задачах»

На тему «Оценка времени выполнения программ»

**Выполнил студент группы 24ВВВ2:**

Лисицын Н.С.

**Приняли:**

Юрова О.В.

Деев М.В.

**Пенза, 2025**

**Цель работы**

Освоение методов измерения времени выполнения программ и их частей на языке Си с использованием функций стандартной библиотеки time.h.

**Лабораторное задание**

* 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.
* построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.
* оценить время работы из реализованных алгоритмов.

**Описание метода решения задачи**

**Задание №1:**

Реализуем функции:

* Память: динамическое создание матриц через двойной malloc.
* Данные: заполнение матриц случайными числами от 1 до 100.
* Замер общего времени работы программы.
* Отдельный замер времени только операции умножения матриц.
* Алгоритм: классическое перемножение матриц "в лоб".
* Завершение: вывод результатов замеров и освобождение памяти.

**Задание №2:**

* Алгоритмы: сортировки — Shell Sort, Quick Sort, qsort.
* Тестирование: сравнивается время работы алгоритмов на четырёх типах массивов.
* Измерения: для точного замера времени каждый алгоритм запускается на копии исходного массива с помощью clock().
* Процесс: программа генерирует тестовые данные, выполняет сортировки, выводит результаты и позволяет провести повторное тестирование.

**Листинг задания №1**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 100

int main(void)

{

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

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

clock\_t start\_all;

float duration\_all;

start\_all = clock();

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

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

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

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

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

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

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

}

srand(time(NULL));

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

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

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

}

}

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

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

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

}

}

clock\_t start;

float duration;

start = clock();

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

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

int elem\_c = 0;

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

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

}

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

}

}

duration = (float)(clock() - start) / CLOCKS\_PER\_SEC;

duration\_all = (float)(clock() - start\_all) / CLOCKS\_PER\_SEC;

printf("Matrix multiplication execution time: %f seconds\n", duration);

printf("Total execution time: %f seconds\n", duration\_all);

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

return 0;

}

**Листинг задания №2**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void shell(int\* items, int count)

{

int i, j, gap, k;

int x;

int a[5] = { 9, 5, 3, 2, 1 };

for (k = 0; k < 5; k++) {

gap = a[k];

for (i = gap; i < count; ++i) {

x = items[i];

for (j = i - gap; (j >= 0) && (x < items[j]); j = j - gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

void qs(int\* items, int left, int right)

{

int i, j;

int x, y;

i = left; j = right;

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

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

}

int\* copy\_array(const int\* src, int count) {

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

if (dest == NULL) {

printf("Memory allocation error\n");

exit(1);

}

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

dest[i] = src[i];

}

return dest;

}

void generate\_random\_array(int\* arr, int count) {

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

arr[i] = rand() % 1000;

}

}

double measure\_time(void (\*sort\_func)(int\*, int), int\* arr, int count) {

clock\_t start, end;

start = clock();

sort\_func(arr, count);

end = clock();

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

}

double measure\_time\_qs(void (\*sort\_func)(int\*, int, int), int\* arr, int count) {

clock\_t start, end;

start = clock();

sort\_func(arr, 0, count - 1);

end = clock();

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

}

double measure\_time\_qsort(int\* arr, int count) {

clock\_t start, end;

start = clock();

qsort(arr, count, sizeof(int), compare);

end = clock();

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

}

int input\_array\_size() {

int size;

printf("Enter array size for testing (100 - 1000000): ");

if (scanf\_s("%d", &size) != 1) {

printf("Invalid input. Using default size 10000\n");

return 10000;

}

if (size < 100) {

printf("Size too small, set to 100\n");

return 100;

}

if (size > 1000000) {

printf("Size too large, set to 1000000\n");

return 1000000;

}

return size;

}

int main() {

srand(time(NULL));

int count;

char choice;

do {

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

printf(" SORTING ALGORITHMS BENCHMARK\n");

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

count = input\_array\_size();

printf("\n");

if (count > 50000) {

printf("Warning: Large array size (%d elements)\n", count);

printf("Testing may take significant time...\n\n");

}

int\* random\_array = (int\*)malloc(count \* sizeof(int));

int\* ascending\_array = (int\*)malloc(count \* sizeof(int));

int\* descending\_array = (int\*)malloc(count \* sizeof(int));

int\* mixed\_array = (int\*)malloc(count \* sizeof(int));

if (random\_array == NULL || ascending\_array == NULL ||

descending\_array == NULL || mixed\_array == NULL) {

printf("Memory allocation error\n");

if (random\_array) free(random\_array);

if (ascending\_array) free(ascending\_array);

if (descending\_array) free(descending\_array);

if (mixed\_array) free(mixed\_array);

return 1;

}

printf("Generating test arrays...\n");

generate\_random\_array(random\_array, count);

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

ascending\_array[i] = i;

descending\_array[i] = count - i - 1;

}

int half = count / 2;

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

mixed\_array[i] = i;

}

for (int i = half; i < count; i++) {

mixed\_array[i] = count - i - 1 + half;

}

printf("Testing with array size %d:\n\n", count);

printf("=== 1. Random Array ===\n");

int\* temp\_arr = copy\_array(random\_array, count);

double time = measure\_time(shell, temp\_arr, count);

printf("Shell Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(random\_array, count);

time = measure\_time\_qs(qs, temp\_arr, count);

printf("Quick Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(random\_array, count);

time = measure\_time\_qsort(temp\_arr, count);

printf("Standard qsort: %.6f seconds\n\n", time);

free(temp\_arr);

printf("=== 2. Ascending Array ===\n");

temp\_arr = copy\_array(ascending\_array, count);

time = measure\_time(shell, temp\_arr, count);

printf("Shell Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(ascending\_array, count);

time = measure\_time\_qs(qs, temp\_arr, count);

printf("Quick Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(ascending\_array, count);

time = measure\_time\_qsort(temp\_arr, count);

printf("Standard qsort: %.6f seconds\n\n", time);

free(temp\_arr);

printf("=== 3. Descending Array ===\n");

temp\_arr = copy\_array(descending\_array, count);

time = measure\_time(shell, temp\_arr, count);

printf("Shell Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(descending\_array, count);

time = measure\_time\_qs(qs, temp\_arr, count);

printf("Quick Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(descending\_array, count);

time = measure\_time\_qsort(temp\_arr, count);

printf("Standard qsort: %.6f seconds\n\n", time);

free(temp\_arr);

printf("=== 4. Mixed Array ===\n");

temp\_arr = copy\_array(mixed\_array, count);

time = measure\_time(shell, temp\_arr, count);

printf("Shell Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(mixed\_array, count);

time = measure\_time\_qs(qs, temp\_arr, count);

printf("Quick Sort: %.6f seconds\n", time);

free(temp\_arr);

temp\_arr = copy\_array(mixed\_array, count);

time = measure\_time\_qsort(temp\_arr, count);

printf("Standard qsort: %.6f seconds\n\n", time);

free(temp\_arr);

free(random\_array);

free(ascending\_array);

free(descending\_array);

free(mixed\_array);

printf("Test with different array size? (y/n): ");

if (scanf\_s(" %c", &choice, 1) != 1) {

choice = 'n';

}

} while (choice == 'y' || choice == 'Y');

printf("Program finished.\n");

return 0;

}

**Результат работы программы задания №1**





**Результат работы программы задания №2**

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Содержимое, созданное искусственным интеллектом, может быть неверным.

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Содержимое, созданное искусственным интеллектом, может быть неверным.

**Вывод**

В ходе данной лабораторной работы были освоены методы измерения времени выполнения программ и их частей на языке Си с использованием функций стандартной библиотеки time.h.

**GitHub**

Ссылка репозитория на GitHub:

<https://github.com/NikitaFoxze/LogicLabs.git>