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:
Пяткин Р. С.
Гусаров Е. Е.

Принял:
Юрова О. В.
Деев М. 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, размером от 100, 200, 400, 1000, 2000, 4000, 10000.
3. Построить график зависимости времени выполнения программы от размера матрици 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 sort, 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 results.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 sorting_results.txt и выводятся на экран.

Результаты программ

LAB_2 > results.txt

```
Результаты умножения матриц для разных диапазонов значений:

Диапазон значений: 0-10
Размер матрицы  Время (секунды)
================================
100x100:        0.00435827
200x200:        0.0272267
400x400:        0.153792
1000x1000:      2.83741
2000x2000:      38.6925
4000x4000:      385.342
10000x10000:        8332.69

Диапазон значений: 10-100
Размер матрицы  Время (секунды)
================================
100x100:        0.00234322
200x200:        0.0181734
400x400:        0.17593
1000x1000:      2.78803
2000x2000:      39.6948
4000x4000:      384.001
10000x10000:        8319.51

Диапазон значений: 100-1000
Размер матрицы  Время (секунды)
================================
100x100:        0.00255873
200x200:        0.0189183
400x400:        0.149839
1000x1000:      2.78978
2000x2000:      40.1352
4000x4000:      384.569
10000x10000:        8320.15

```

1 Рис. — results.txt

LAB_2 > sorting_results.txt

```
=== СРАВНЕНИЕ АЛГОРИТМОВ СОРТИРОВКИ ===
   Размер массива: 100000 элементов
========================================

1. СОРТИРОВКА ШЕЛЛА (SHELL SORT):
---------------------------------
   По возрастанию  0.003119 секунд
   По убыванию     1.684795 секунд
   Смешанный       0.840893 секунд
   Случайный       0.833954 секунд

2. БЫСТРАЯ СОРТИРОВКА (QUICK SORT):
-----------------------------------
   По возрастанию  0.002506 секунд
   По убыванию     0.002974 секунд
   Смешанный       3.290135 секунд
   Случайный       0.010348 секунд

3. СТАНДАРТНАЯ СОРТИРОВКА (STD::SORT):
--------------------------------------
   По возрастанию  0.007243 секунд
   По убыванию     0.005627 секунд
   Смешанный       0.029124 секунд
   Случайный       0.015400 секунд

========================================
ТЕСТИРОВАНИЕ ЗАВЕРШЕНО

```

2 Рис. — sorting_result.txt

1. Вычисление порядка сложности программы (О-символика)

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

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];
            }
      }
}
```

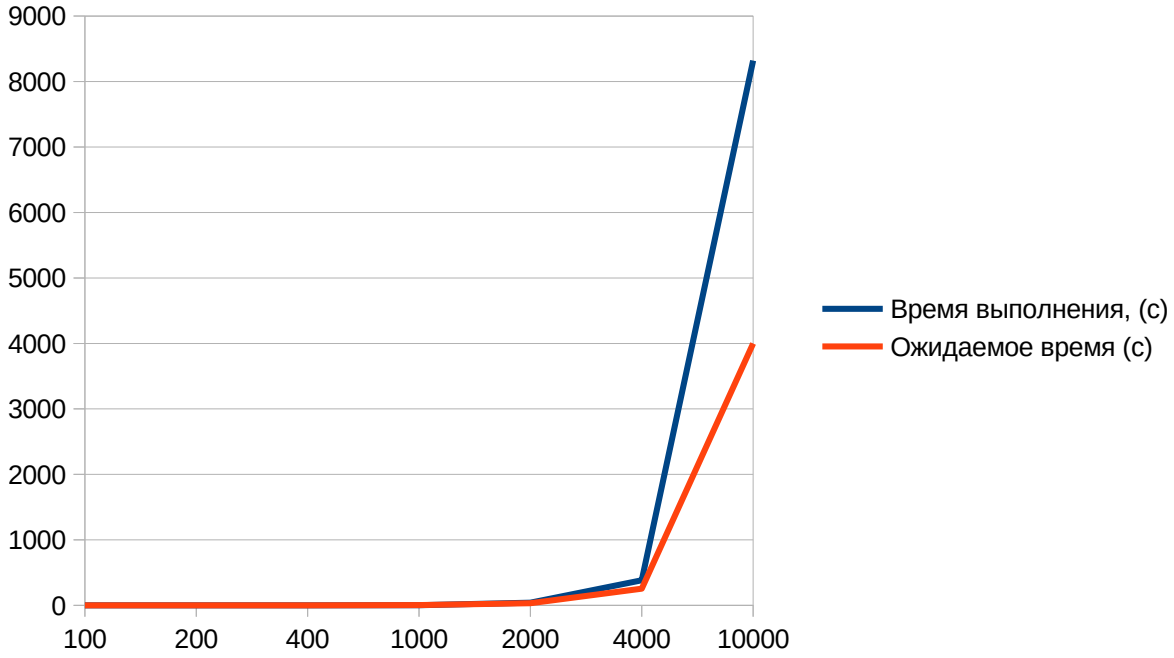
- Внешний цикл i выполняется n раз.
- Второй цикл j выполняется n раз.
- Внутренний цикл k выполняется n раз.

Это значит, что временная сложность алгоритма — **$O(n^3)$**.

2. Оценка времени выполнения программы и кода, выполняющего перемножение

Размер матрицы	**Время выполнения, (с)**	**Ожидаемое время (с)**
100	0,002	0,004
200	0,018	0,032
400	0,175	0,256
1000	2,788	4
2000	39,694	32
4000	384,001	256
10000	8319,51	4000

3. График зависимости времени выполнения операции



4. Оценка времени выполнения алгоритмов сортировки

Алгоритм Набор данных	shell	qs	std
Случайный	**0.833954**	**0.010348**	**0.015400**
Возрастающий	**0.003119**	**0.002506**	**0.007243**
Убывающий	**1.684893**	**0.002974**	**0.005627**
Смешанный	**0.840893**	**3.290135**	**0.029124**

Из результата следует, что для сортировки случайных, возрастающих и убывающих данных следует воспользоваться сортировкой qs, для сортировки смешанных данных лучше использовать встроенную функцию сортировки sort.

Листинг

Файл lab_2_1.cpp

```
#include <iostream>
#include <cstdlib>
#include <ctime>
#include <stdexcept>
#include <chrono>
#include <fstream>
```

```
using namespace std;

int** create_matrix(int n) {
int** arr = new int*[n];
for (int i = 0; i < n; i++) {
arr[i] = new int[n];
}
return arr;
}

void fill_matrix(int n, int** matrix, int range_type) {
int min_val, max_val;
switch(range_type) {
case 1: // 0-10
min_val = 0;
max_val = 10;
break;
case 2: // 10-100
min_val = 10;
max_val = 100;
break;
case 3: // 100-1000
min_val = 100;
max_val = 1000;
break;
default:
min_val = 0;
max_val = 100;
}
int range = max_val - min_val + 1;
for (int i = 0; i < n; i++) {
for (int j = 0; j < n; j++) {
matrix[i][j] = rand() % range + min_val;
}
}
}

void delete_matrix(int** matrix, int n) {
if (matrix == nullptr) return;
for (int i = 0; i < n; i++) {
delete[] matrix[i];
}
delete[] matrix;
}

double mult_matrix(int n, int range_type){
int** a = create_matrix(n);
int** b = create_matrix(n);
```

```
int** c = create_matrix(n);
fill_matrix(n, a, range_type);
fill_matrix(n, b, range_type);
auto start = chrono::high_resolution_clock::now();
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];
}
}
}
auto end = chrono::high_resolution_clock::now();
chrono::duration<double> duration = end - start;
double seconds = duration.count();

delete_matrix(a, n);
delete_matrix(b, n);
delete_matrix(c, n);
return seconds;
}

int main() {
srand(time(0));
int sizes[] = {100, 200, 400, 1000, 2000, 4000, 10000};
int num_sizes = sizeof(sizes) / sizeof(sizes[0]);
const char* range_names[] = {"0-10", "10-100", "100-1000"};
ofstream outfile("results.txt");
cout << "Выполняются вычисления..." << endl;
outfile << "Результаты умножения матриц для разных диапазонов значений:\n\n";
for (int r = 0; r < 3; r++) {
outfile << "Диапазон значений: " << range_names[r] << "\n";
outfile << "Размер матрицы\tВремя (секунды)\n";
outfile << "================================\n";
cout << "\nДиапазон значений: " << range_names[r] << endl;
cout << "================================" << endl;
for (int i = 0; i < num_sizes; i++){
double time = mult_matrix(sizes[i], r+1);
outfile << sizes[i] << "x" << sizes[i] << ":\t " << time << "\n";
cout << "Матрица " << sizes[i] << "x" << sizes[i]
<< ":\t" << time << " сек" << endl;
}
outfile << "\n";
cout << "\n";
}
outfile.close();
cout << "\nРезультаты сохранены в файл 'results.txt'" << endl;
return 0;
```

```
}
```

Файл lab_2_2.cpp

```
#include <iostream>
#include <cstdlib>

#include <ctime>
#include <algorithm>
#include <chrono>
#include <fstream>

using namespace std;

double sort_arr(int* items, int n){
auto start = chrono::high_resolution_clock::now();
sort(items, items + n);
auto end = chrono::high_resolution_clock::now();
chrono::duration<double> duration = end - start;
double seconds = duration.count();
return seconds;
}

double shell(int* items, int count) {
int i, j, gap, k;
int x, arr[5] = {9, 5, 3, 2, 1};
auto start = chrono::high_resolution_clock::now();
for (k = 0; k < 5; k++) {
gap = arr[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;
}
}
auto end = chrono::high_resolution_clock::now();
chrono::duration<double> duration = end - start;
double seconds = duration.count();
return seconds;
}

void quick_sort_helper(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) quick_sort_helper(items, left, j);
if(i < right) quick_sort_helper(items, i, right);
}

double qs(int* items, int left, int right) {
auto start = chrono::high_resolution_clock::now();
quick_sort_helper(items, left, right);
auto end = chrono::high_resolution_clock::now();
chrono::duration<double> duration = end - start;
double seconds = duration.count();
return seconds;
}

void create_arr(int* items, int n, int sp){
switch (sp){
case 0:
for(int i = 0; i < n; i++){
items[i] = i;
}
break;
case 1:
for(int i = 0; i < n; i++){
items[i] = n - i;
}
break;
case 2:
for(int i = 0; i < n; i++){
if(i <= (n / 2)){
items[i] = i;
}
else{
items[i] = n - i;
}
}
break;
default:
```

```
for(int i = 0; i < n; i++){
items[i] = rand() % 101;
}
break;
}
}

void delete_arr(int* items, int n){
for(int i = 0; i < n; i++){
items[i] = 0;
}
}

string get_array_type_name(int type) {
switch(type) {
case 0: return "По возрастанию ";
case 1: return "По убыванию ";
case 2: return "Смешанный ";
default: return "Случайный ";
}
}

int main() {
srand(time(0));
int n = 100000;
int* main_arr = new int[n];
ofstream outfile("sorting_results.txt");
if (!outfile.is_open()) {
cerr << "Ошибка открытия файла для записи!" << endl;
return 1;
}
cout << "=== СРАВНЕНИЕ АЛГОРИТМОВ СОРТИРОВКИ ===" << endl;
cout << "\tРазмер массива: " << n << " элементов" << endl;
cout << "=======================================" << endl << endl;
outfile << "=== СРАВНЕНИЕ АЛГОРИТМОВ СОРТИРОВКИ ===" << endl;
outfile << "\tРазмер массива: " << n << " элементов" << endl;
outfile << "=======================================" << endl << endl;

cout << "1. СОРТИРОВКА ШЕЛЛА (SHELL SORT):" << endl;
cout << "---------------------------------" << endl;
outfile << "1. СОРТИРОВКА ШЕЛЛА (SHELL SORT):" << endl;
outfile << "---------------------------------" << endl;
for(int i = 0; i < 4; i++){
create_arr(main_arr, n, i);
double seconds = shell(main_arr, n);
delete_arr(main_arr, n);
string result = " " + get_array_type_name(i) + to_string(seconds) + " секунд";
cout << result << endl;
outfile << result << endl;
}
```

```
cout << endl;
outfile << endl;
cout << "2. БЫСТРАЯ СОРТИРОВКА (QUICK SORT):" << endl;
cout << "----------------------------------" << endl;
outfile << "2. БЫСТРАЯ СОРТИРОВКА (QUICK SORT):" << endl;
outfile << "----------------------------------" << endl;
for(int i = 0; i < 4; i++){
create_arr(main_arr, n, i);
double seconds = qs(main_arr, 0, n-1);
delete_arr(main_arr, n);
string result = " " + get_array_type_name(i) + to_string(seconds) + " секунд";
cout << result << endl;
outfile << result << endl;
}
cout << endl;
outfile << endl;
cout << "3. СТАНДАРТНАЯ СОРТИРОВКА (STD::SORT):" << endl;
cout << "--------------------------------------" << endl;
outfile << "3. СТАНДАРТНАЯ СОРТИРОВКА (STD::SORT):" << endl;
outfile << "--------------------------------------" << endl;
for(int i = 0; i < 4; i++){
create_arr(main_arr, n, i);
double seconds = sort_arr(main_arr, n);
delete_arr(main_arr, n);
string result = " " + get_array_type_name(i) + to_string(seconds) + " секунд";
cout << result << endl;
outfile << result << endl;
}
cout << endl;
outfile << endl;
cout << "====================================" << endl;
cout << "ТЕСТИРОВАНИЕ ЗАВЕРШЕНО" << endl;
cout << "Результаты сохранены в файл: sorting_results.txt" << endl;
outfile << "====================================" << endl;
outfile << "ТЕСТИРОВАНИЕ ЗАВЕРШЕНО" << endl;

outfile.close();
delete[] main_arr;
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.