Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

студент группы 23ВВВ2

Кокарев Д.С.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2024

**Название**

Оценка времени выполнения программ.

**Цель работы**

Оценить время выполнения программ языка Си или их частей.

**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.

**Описание метода решения задачи**

Задание 1:

1. Вычислил порядок сложности программы (О-символику).



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| A = O(1)  B = O(n)  C = O(1)  D = O(n)  E = O(n2)  F = O(n3) | G = O(1)  H = O(n)  I = O(1)  J = O(n)  K = O(n2) | L = O(1) | M = O(1)  N = O(n)  O = O(1)  P = O(n)  Q = O(n2) | R = O(1) |
| S = O(n3) | | | | |

Порядок сложности программы O(n3).

2. Используя команды:

clock\_t start = clock();

clock\_t end = clock();

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

оценил время выполнения программы и кода, выполняющего перемножение двух матриц между собой.

3. Таблица зависимости времени выполнения программы от размера матриц.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Размер матрицы | 100 | 200 | 400 | 1000 | 2000 | 4000 | 10000 |
| время | 0,004 | 0,024 | 0,284 | 4,021 | 47,722 | 869,145 | 5693,753 |
| O(n3) | 0,009 | 0,072 | 0,576 | 9 | 72 | 576 | 9000 |

Задание 2:

Оценил время работы алгоритма сортировки Шелла, быстрой сортировки и стандартной функции qsort на случайном наборе значений массива; на массиве, представляющем собой возрастающую последовательность чисел; на массиве, представляющем собой убывающую последовательность чисел; на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая – убывающую.

Используя команды:

clock\_t start = clock();

clock\_t end = clock();

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

**Таблицы сравнения времени работы сортировок**

Массив из 50.000 элементов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировки | Для массива случайных чисел | Для возрастающей последовательности | Для убывающей последовательности | Для смешанной последовательности |
| Сортировка Шелла | 0,202 | 0,001 | 0,396 | 0,199 |
| Быстрая сортировка | 0,003 | 0,002 | 0,002 | 0,055 |
| qsort | 0,005 | 0,004 | 0,004 | 0,012 |

Массив из 100.000 элементов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировки | Для массива случайных чисел | Для возрастающей последовательности | Для убывающей последовательности | Для смешанной последовательности |
| Сортировка Шелла | 0,079 | 0,001 | 1,058 | 0,792 |
| Быстрая сортировка | 0,007 | 0,003 | 0,004 | 0,198 |
| qsort | 0,010 | 0,009 | 0,009 | 0,021 |

Массив из 200.000 элементов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировки | Для массива случайных чисел | Для возрастающей последовательности | Для убывающей последовательности | Для смешанной последовательности |
| Сортировка Шелла | 3,241 | 0,003 | 10,013 | 4,808 |
| Быстрая сортировка | 0,015 | 0,007 | 0,007 | 0,659 |
| qsort | 0,019 | 0,018 | 0,019 | 0,057 |

Исходя из данных таблицы, можно сделать вывод о том, что для сортировки массива случайных чисел лучше выбирать быструю сортировку, для возрастающей последовательности – сортировку Шелла, для убывающей последовательности – быструю сортировку, а для смешанной последовательности – сортировку qsort.

**Листинг**

**lab2.cpp**

//Задание 1

#include <iostream>

#include <ctime>

#include <locale>

int main() {

setlocale(LC\_ALL, "Russian");

int N;

std::cout << "Введите размер массива: ";

std::cin >> N;

int\*\* a = new int\* [N];

for (int i = 0; i < N; i++) a[i] = new int[N];

int\*\* b = new int\* [N];

for (int i = 0; i < N; i++) b[i] = new int[N];

int\*\* c = new int\* [N];

for (int i = 0; i < N; i++) c[i] = new int[N];

srand(time(NULL));

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

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

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

}

}

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

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

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

}

}

clock\_t 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;

}

}

clock\_t end = clock();

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

std::cout << "Время выполнения: " << time\_spent << " секунд" << std::endl;

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

delete[] a[i];

delete[] b[i];

delete[] c[i];

}

delete[] a;

delete[] b;

delete[] c;

return 0;

}

//Задание 2

#include <iostream>

#include <ctime>

#include <algorithm>

#include <cstring>

#include <locale>

#include <cstdlib>

#include <chrono>

#include <time.h>

using namespace std;

void calc(int\* a, int n);

void shell(int\* items, int count);

void output(int\* items, int count);

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

int cmp(const void\* x1, const void\* x2);

int main() {

setlocale(LC\_ALL, "Russian");

srand(time(NULL));

int N;

cout << "Введите размер массива: ";

cin >> N;

int\* A = new int[N];

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

A[i] = rand() % 200 - 100;

}

cout << "\nДля массива из случайных чисел\n\n";

calc(A, N);

cout << "\n";

// По возрастанию

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

A[i] = i \* 2 - N;

}

cout << "Для возрастающей последовательности\n\n";

calc(A, N);

cout << "\n";

// По убыванию

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

A[i] = -(i \* 2 - N);

}

cout << "Для убывающей последовательности\n\n";

calc(A, N);

cout << "\n";

// Половина убывает, половина возрастает

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

A[i] = abs(i \* 2 - N);

}

cout << "Для смешанной последовательности\n\n";

calc(A, N);

delete[] A;

return 0;

}

void calc(int\* a, int n) {

clock\_t start, end;

double time;

int\* B = new int[n];

memcpy(B, a, sizeof(int) \* n);

start = clock();

shell(B, n);

end = clock();

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

cout << "Сортировка Шелла: " << time << " секунд\n";

memcpy(B, a, sizeof(int) \* n);

start = clock();

qs(B, 0, n - 1);

end = clock();

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

cout << "Быстрая сортировка: " << time << " секунд\n";

memcpy(B, a, sizeof(int) \* n);

start = clock();

qsort(B, n, sizeof(int), cmp);

end = clock();

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

cout << "qsort: " << time << " секунд\n";

delete[] B;

}

void output(int\* items, int count) {

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

cout << items[i] << " ";

}

cout << endl;

}

void shell(int\* items, int count) {

int i, j, gap, k;

int x, 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; (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) {

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 cmp (const void\* x1, const void\* x2) {

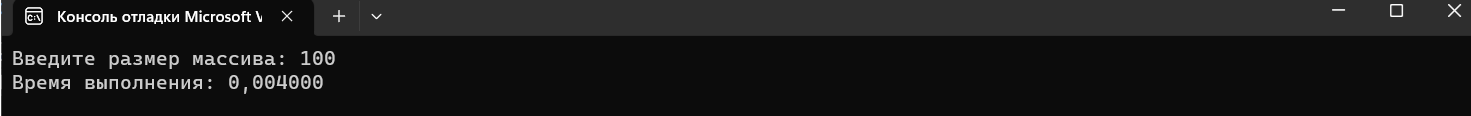
return (\*(int\*)x1 - \*(int\*)x2);

}

**Результаты работы программы**

Результаты работы программы представлены на рисунках:

Задание 1:

Рисунок 1 - Матрица 100 x 100

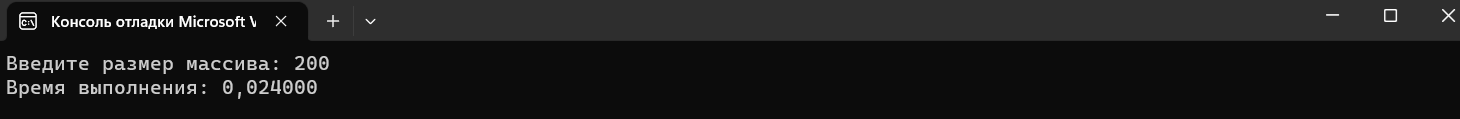
Рисунок 2 - Матрица 200 x 200

Рисунок 3 - Матрица 400 x 400

Рисунок 4 - Матрица 1000 x 1000

Рисунок 5 - Матрица 2000 x 2000

Рисунок 6 - Матрица 4000 x 4000

Рисунок 7 - Матрица 10000 x 10000

Задание 2:

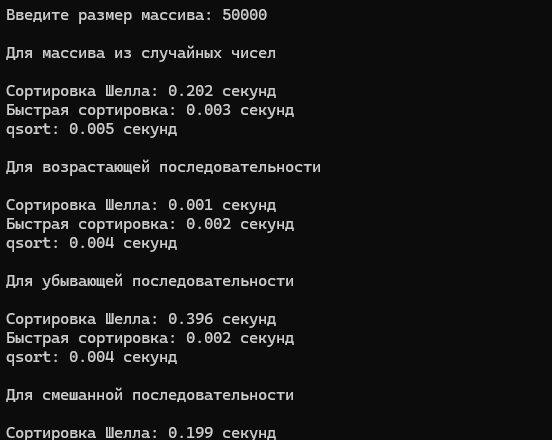


Рисунок 8 - Массив из 50.000 элементов

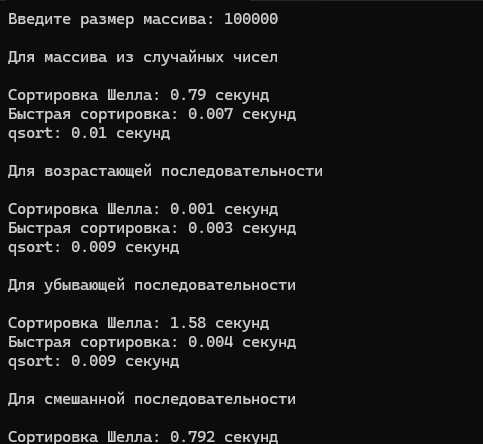


Рисунок 9 - Массив из 100.000 элементов

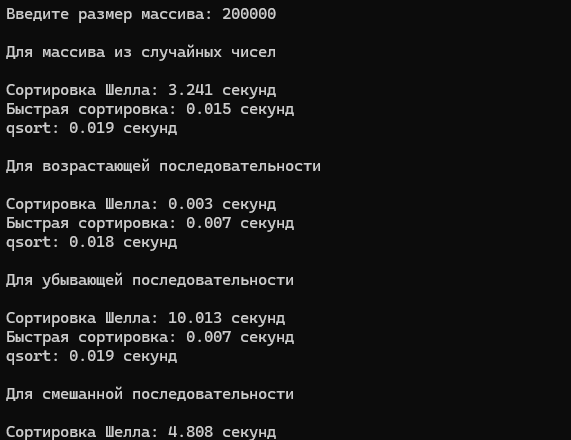


Рисунок 10 - Массив из 200.000 элементов

**Вывод**

В ходе выполнения лабораторной работы были разработаны алгоритмы для оценки времени выполнения программ. Полученные результаты совпали с ожидаемыми, что подтверждает корректность работы программы. Также в процессе выполнения работы был приобретен опыт создания проектов в среде Microsoft Visual Studio и улучшены навыки программирования алгоритмов.