Министерство 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 21ВВ1.2

Брянцев Артём  
Сущёв Максим  
Тюрин Владислав

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2022

**Название**

Оценка времени выполнения 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.

**Листинг**

lb2.cpp

#include <iostream>

#include <fstream>

using namespace std;

void arrayfilling(int\*\* x, int size) {

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

\*(x + i) = (int\*)malloc(size \* sizeof(int));

for (int j = 0; j < size; j++)

{

\*(\*(x + i) + j) = rand() % 100 + 1;

}

}

}

void freedom(int\*\* x, int size) {

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

free(\*(x + i)); //освобождаем память

}

free(x); //освобождаем память

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

clock\_t startTime, stopTime;

double times;

const int size = 10000;

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

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

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

int elem\_c;

startTime = clock(); //время до поиска

arrayfilling(a, size);

arrayfilling(b, size);

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

\*(c + i) = (int\*)malloc(size \* sizeof(int));

for (int j = 0; j < size; j++)

{

elem\_c = 0;

for (int r = 0; r < size; r++)

{

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

}

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

}

}

stopTime = clock(); //время после поиска

times = double(stopTime - startTime) / CLK\_TCK;

ofstream fout("res.txt", ios\_base::app);

fout << "При размере " << size << " результат = " << times << " sec" << endl;

fout.close();

cout << times << " sec" << endl;

freedom(a, size);

freedom(b, size);

freedom(c, size);

}

lb2pt2.cpp

#include <iostream>

#include <fstream>

using namespace std;

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) //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);

}

int comp(const int\* i, const int\* j)

{

return \*i - \*j;

}

int main()

{

setlocale(LC\_ALL, "rus");

srand(time(NULL));

clock\_t startTime, stopTime;

double times;

const int size = 1000000;

int i = 0, m;

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

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

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

int\* d = (int\*)malloc(size \* sizeof(int));

for (i = 0; i < size; i++) {

a[i] = rand() % 10;

}

for (i = 0; i < size; i++) {

b[i] = i\*2;

}

c[0] = 0;

for (i = 1; i < size; i++) {

c[i] = c[0] - i\*2;

}

if (size % 2 == 0) {

m = size / 2;

}

else {

m = size / 2 + 1;

}

d[0] = 0;

for (i = 1; i < m; i++) {

d[i] = d[i-1] + 3;

}

for (i = m; i < size; i++) {

d[i] = d[i-1] - 3;

}

ofstream fout("res.txt", ios\_base::app);

fout.width(20);

fout << "Cортировка Шелла ";

fout << size << " ";

startTime = clock();

shell(a, size);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

shell(b, size);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

shell(c, size);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

shell(d, size);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec " << endl;

fout.width(20);

fout << "Быстрая сортировка ";

fout << size << " ";

startTime = clock();

qs(a, 0, size - 1);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

qs(b, 0, size - 1);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

qs(c, 0, size - 1);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

qs(d, 0, size - 1);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec " << endl;

fout.width(20);

fout << "Сортировка qsort ";

fout << size << " ";

startTime = clock();

qsort(a, size, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

qsort(b, size, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

qsort(c, size, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec ";

startTime = clock();

qsort(d, size, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

stopTime = clock();

times = double(stopTime - startTime) / CLK\_TCK;

fout << times << " sec " << endl << endl;

fout.close();

free(a);

free(b);

free(c);

free(d);

}

**Результат работы программы**

Результаты работы программы (задание 1) показаны на рисунке 1.

Изображение выглядит как текст

Автоматически созданное описание**Рисунок 1 — Результаты работы программы**

Результаты работы программы показаны на рисунке 2.

Изображение выглядит как текст

Автоматически созданное описание

**Рисунок 2 — Результаты работы программы**

Таблица 1 — График времени работы программы (задание 1)

|  |  |  |
| --- | --- | --- |
| № теста. | Количество элементов. | Время выполнения сортировки в секундах. |
| 1 | 100 | 0.005 |
| 2 | 200 | 0.032 |
| 3 | 400 | 0.379 |
| 4 | 1000 | 6.701 |
| 5 | 2000 | 61.351 |
| 6 | 4000 | 639.104 |
| 7 | 10000 | 6189.73 |

**Рисунок 3 — Результат тестирования**

Графики и таблицы по 2-ому заданию находятся в файле restest.xlsx.

### Выводы

В первой части лабораторной работы определили сложность алгоритма, равна O(n­3). А во второй части были разработаны программы с использованием массивов и реализованы алгоритмы сортировки Шелла, быстрой сортировки и qsort. Сортировка Шелла оказалась эффективна на возрастающем массиве, быстрая сортировка и qsort оказались эффективны на всех видах массивов.