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»

Выполнил: студент группы 19ВВ2

Урядов В.Д.

Принял: Д.Т.Н., профессор

Митрохин М.А.

Пенза 2020

**Цель работы:** получить навыки работы с библиотекой time.h для оценки времени выполнения программ.

**Ход работы:**

**Задание 1**:

Вычислить порядок сложности программы. 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.

**Листинг:**

Header.h

#pragma once

int matrix(int SIDE, FILE\* fp);

sorts.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

#include "header.h"

#define LENGTH 7560

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 fillrand(int\* mas, int length) {

srand(time(0));

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

mas[i] = rand() % 100001;

}

return 0;

}

int fillincr(int\* mas, int length) {

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

mas[i] = i;

}

return 0;

}

int filldecr(int\* mas, int length) {

for (int i = 1; i <= length; i++) {

mas[i-1] = length - i;

}

return 0;

}

int fillincdecr(int\* mas, int length) {

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

mas[i] = i;

}

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

mas[i] = length - i;

}

return 0;

}

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

{

return \*i - \*j;

}

int main(void)

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

FILE \*fp;

char name[] = "moo.txt";

if ((fp = fopen(name, "w")) == NULL)

{

fprintf(fp, "Не удалось открыть файл");

getchar();

}

matrix(100, fp);

matrix(200, fp);

matrix(400, fp);

matrix(1000, fp);

matrix(2000, fp);

int\* to\_sort\_random = (int\*)malloc(LENGTH \* sizeof(int));

int\* to\_sort\_random\_cpy = (int\*)malloc(LENGTH \* sizeof(int));

fillrand(to\_sort\_random, LENGTH);

fprintf(fp, "Распределение\tqs\tШелл\tqsort\n");

clock\_t start, end;

memcpy(to\_sort\_random\_cpy, to\_sort\_random, LENGTH \* sizeof(int));

start = clock();

qs(to\_sort\_random\_cpy, 0, LENGTH - 1);

end = clock();

start = difftime(end, start);

fprintf(fp, "псевдослучайное\t%d\t", start);

memcpy(to\_sort\_random\_cpy, to\_sort\_random, LENGTH \* sizeof(int));

start = clock();

shell(to\_sort\_random\_cpy, LENGTH);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\t", start);

memcpy(to\_sort\_random\_cpy, to\_sort\_random, LENGTH \* sizeof(int));

start = clock();

qsort(to\_sort\_random\_cpy, LENGTH, sizeof(int), (int(\*) (const void \*, const void \*)) comp);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\n", start);

int\* to\_sort\_inc = (int\*)malloc(LENGTH \* sizeof(int));

int\* to\_sort\_inc\_cpy = (int\*)malloc(LENGTH \* sizeof(int));

fillincr(to\_sort\_inc, LENGTH);

memcpy(to\_sort\_inc\_cpy, to\_sort\_inc, LENGTH \* sizeof(int));

start = clock();

qs(to\_sort\_inc\_cpy, 0, LENGTH - 1);

end = clock();

start = difftime(end, start);

fprintf(fp, "возрастающее\t%d\t", start);

memcpy(to\_sort\_inc\_cpy, to\_sort\_inc, LENGTH \* sizeof(int));

start = clock();

shell(to\_sort\_inc\_cpy, LENGTH);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\t", start);

memcpy(to\_sort\_inc\_cpy, to\_sort\_inc, LENGTH \* sizeof(int));

start = clock();

qsort(to\_sort\_inc\_cpy, LENGTH, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\t", start);

int\* to\_sort\_dec = (int\*)malloc(LENGTH \* sizeof(int));

int\* to\_sort\_dec\_cpy = (int\*)malloc(LENGTH \* sizeof(int));

filldecr(to\_sort\_dec, LENGTH);

memcpy(to\_sort\_dec\_cpy, to\_sort\_dec, LENGTH \* sizeof(int));

fprintf(fp, "\n");

start = clock();

qs(to\_sort\_dec\_cpy, 0, LENGTH - 1);

end = clock();

start = difftime(end, start);

fprintf(fp, "убывающее\t%d\t", start);

memcpy(to\_sort\_dec\_cpy, to\_sort\_dec, LENGTH \* sizeof(int));

start = clock();

shell(to\_sort\_dec\_cpy, LENGTH);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\t", start);

memcpy(to\_sort\_dec\_cpy, to\_sort\_dec, LENGTH \* sizeof(int));

start = clock();

qsort(to\_sort\_dec\_cpy, LENGTH, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\t\n", start);

int\* to\_sort\_incdec = (int\*)malloc(LENGTH \* sizeof(int));

int\* to\_sort\_incdec\_cpy = (int\*)malloc(LENGTH \* sizeof(int));

fillincdecr(to\_sort\_incdec, LENGTH);

memcpy(to\_sort\_incdec\_cpy, to\_sort\_incdec, LENGTH \* sizeof(int));

start = clock();

qs(to\_sort\_incdec\_cpy, 0, LENGTH - 1);

end = clock();

start = difftime(end, start);

fprintf(fp, "возр.-убыв.\t%d\t", start);

memcpy(to\_sort\_incdec\_cpy, to\_sort\_incdec, LENGTH \* sizeof(int));

start = clock();

shell(to\_sort\_incdec\_cpy, LENGTH);

end = clock();

start = difftime(end, start);

fprintf(fp, "%d\t", start);

memcpy(to\_sort\_incdec\_cpy, to\_sort\_incdec, LENGTH \* sizeof(int));

start = clock();

qsort(to\_sort\_incdec\_cpy, LENGTH, sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

fprintf(fp, "%d\t", start);

fclose(fp);

}

Lab2\_1.cpp

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

int matrix(int SIDE, FILE\* fp)

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

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

clock\_t start, end; // объявляем переменные для определения времени выполнения

int i = 0, j = 0, r;

int elem\_c;

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

for (int i = 0; i < SIDE; i++)

{

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

}

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

for (int i = 0; i < SIDE; i++)

{

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

}

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

for (int i = 0; i < SIDE; i++)

{

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

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

while (i < SIDE)

{

while (j < SIDE)

{

a[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

srand(time(NULL)); // инициализируем параметры генератора случайных чисел

i = 0; j = 0;

while (i < SIDE)

{

while (j < SIDE)

{

b[i][j] = rand() % 100 + 1; // заполняем массив случайными числами

j++;

}

i++;

}

start = clock();

for (i = 0; i < SIDE; i++)

{

for (j = 0; j < SIDE; j++)

{

elem\_c = 0;

for (r = 0; r < SIDE; r++)

{

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

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

}

}

}

end = clock();

start = difftime(end, start);

fprintf(fp, "Время выполнения всей программы %d тиков. Размер %d\n", end, SIDE);

fprintf(fp, "Время перемножения матриц %d тиков. Размер %d\n", start, SIDE);

return(0);

}

Вывод программы:

Время выполнения всей программы 214 тиков. Размер 100

Время перемножения матриц 4 тиков. Размер 100

Время выполнения всей программы 245 тиков. Размер 200

Время перемножения матриц 31 тиков. Размер 200

Время выполнения всей программы 638 тиков. Размер 400

Время перемножения матриц 392 тиков. Размер 400

Время выполнения всей программы 13350 тиков. Размер 1000

Время перемножения матриц 12706 тиков. Размер 1000

Время выполнения всей программы 100343 тиков. Размер 2000

Время перемножения матриц 86975 тиков. Размер 2000

Распределение qs Шелл qsort

псевдослучайное 1 5 4

возрастающее 0 0 3

убывающее 0 9 3

возр.-убыв. 38 8 10

Порядок сложности программы вычисления произведения матриц O(n3)

**Задание 2:**

Время работы различных сортировок на массиве из 7560 целых неотрицательных чисел

|  |  |  |  |
| --- | --- | --- | --- |
| Распределение | с. Шелла, тиков | qs, тиков | qsort, тиков |
| Псевдослучайное | 5 | 1 | 4 |
| Возрастающее | 0 | 0 | 3 |
| Убывающее | 9 | 0 | 3 |
| Пилообразное | 8 | 38 | 10 |

**Вывод:**

Сортировка Шелла лучше остальных справилась с возрастающим массивом элементов, но с остальными массивами показала средние результаты.

Сортировка qs, данная в задании, работает быстро со всеми типами массивов, но ее эффективность падает значительно при обработке пилообразного массива

Сортировка qsort из стандартной библиотеки справилась со всеми типами массивов за приемлемое время. Эта сортировка является самой универсальной.