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

Трошкин В.К.

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

#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);

matrix(4000, fp);

matrix(10000, 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");

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_random[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_random\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_random\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_random\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_inc[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_inc\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_inc\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_inc\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_dec[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_dec\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_dec\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_dec\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_incdec[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_incdec\_cpy[i]);

}\*/

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);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_incdec\_cpy[i]);

}\*/

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();

start = difftime(end, start);

/\*for (int i=0; i < LENGTH; i++) {

fprintf(fp, "%d ", to\_sort\_incdec\_cpy[i]);

}\*/

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

fclose(fp);

}

Lab2\_1.cpp

#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);

}

Порядок сложности программы вычисления произведения матриц O(n3)

Вывод программы:

Время выполнения всей программы 166 тиков. Размер 100

Время перемножения матриц 6 тиков. Размер 100

Время выполнения всей программы 205 тиков. Размер 200

Время перемножения матриц 39 тиков. Размер 200

Время выполнения всей программы 614 тиков. Размер 400

Время перемножения матриц 407 тиков. Размер 400

Время выполнения всей программы 8826 тиков. Размер 1000

Время перемножения матриц 8204 тиков. Размер 1000

Время выполнения всей программы 122191 тиков. Размер 2000

Время перемножения матриц 113335 тиков. Размер 2000

Время выполнения всей программы 891242 тиков. Размер 4000

Время перемножения матриц 768936 тиков. Размер 4000

Время выполнения всей программы 11137150 тиков. Размер 10000

Время перемножения матриц 10244987 тиков. Размер 10000

Распределение qs Шелл qsort

псевдослучайное 1 6 16

возрастающее 0 1 3

убывающее 0 11 3

возр.-убыв. 41 6 10

**Задание 2:**

1. Оценил время работы каждого из реализованных алгоритмов на случайном наборе значений массива.

2. Оценил время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

3. Оценил время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

4. Оценил время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

5. Оценил время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

Время работы различных сортировок на массиве из 7560 целых неотрицательных чисел

|  |  |  |  |
| --- | --- | --- | --- |
| Распределение | с. Шелла, тиков | qs, тиков | qsort, тиков |
| псевдослучайное | 6 | 1 | 4 |
| возрастающее | 1 | 1 | 3 |
| убывающее | 11 | 1 | 3 |
| возр.-убыв. | 6 | 41 | 10 |

С возрастающим (то есть уже отсортированным) массивом все сортировки справились быстро, только qsort из стандартной библиотеки оказался немного медленнее.

Сортировка Шелла достаточно медленно работает с псевдослучайным (равномерным) распределением, как и с убывающим, однако быстрее двух реализаций quicksort на массиве, в котором числа возрастают до середины, а затем убывают.

Реализация qs из условия работает очень быстро как с равномерным, так и с убывающим распределением, однако катастрофически замедляется на возрастающе-убывающем. Другой её минус – ограничение, связанное с глубиной стека, так как реализация рекурсивная.

Реализация qsort из стандартной библиотеки менее сильно замедляется на возрастающе-убывающем массиве, хотя и всё ещё уступает в этом сортировке Шелла, зато сильно обгоняет её на убывающих значениях. Неплохо справляется она и с другими распределениями. Ни на одном распределении она не лучшая, но и не деградирует сильно ни на одном из массивов. Это универсальная сортировка. Её стоит использовать, когда нет уверенности, на что будет похож массив данных, которые необходимо отсортировать.

Две другие сортировки же можно применять, когда есть уверенность в том, к какому распределению будет ближе массив входных данных.

**Вывод:** научился оценивать время работы программ при помощи time.h.