Министерство науки и высшего образования Российской Федерации  
Пензенский государственный университет  
Кафедра вычислительная техника

**ОТЧËТ**  
по лабораторной работе №2  
по дисциплине: «Оценка времени выполнения программ»

Выполнили студенты группы 22ВВВ2:  
Беляев Д.

Приняли:  
Акифьев И. В.

Митрохин М. А.

Пенза 2023

**Название**

Оценка времени выполнения программ

**Цель работы**

Научиться измерять время работы и сложность кода.

**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**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define ARRAY\_SIZE\_1 100

#define ARRAY\_SIZE\_2 100000

#pragma region Exercize1

int\*\* CreateDoubleArray(int size)

{

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

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

{

array[i] = (int\*)malloc(size \* sizeof(int));

}

return array;

}

void ClearDoubleArray(int\*\* array, int size)

{

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

{

free(array[i]);

}

free(array);

}

void Exercize1()

{

int i = 0, j = 0, r;

int elem\_c;

int\*\* a = CreateDoubleArray(ARRAY\_SIZE\_1);

int\*\* b = CreateDoubleArray(ARRAY\_SIZE\_1);

int\*\* c = CreateDoubleArray(ARRAY\_SIZE\_1);

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

while (i < ARRAY\_SIZE\_1)

{

while (j < ARRAY\_SIZE\_1)

{

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 < ARRAY\_SIZE\_1)

{

while (j < ARRAY\_SIZE\_1)

{

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

j++;

}

i++;

}

for (i = 0; i < ARRAY\_SIZE\_1; i++)

{

for (j = 0; j < ARRAY\_SIZE\_1; j++)

{

elem\_c = 0;

for (r = 0; r < ARRAY\_SIZE\_1; r++)

{

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

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

}

}

}

ClearDoubleArray(a, ARRAY\_SIZE\_1);

ClearDoubleArray(b, ARRAY\_SIZE\_1);

ClearDoubleArray(c, ARRAY\_SIZE\_1);

}

#pragma endregion

#pragma region Exercize2

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\* CopyArray(int\* from, int size)

{

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

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

to[i] = from[i];

return to;

}

int array1[ARRAY\_SIZE\_2] = { 0 };

int\* array2;

int\* array3;

void RandomArray()

{

for (int i = 0; i < ARRAY\_SIZE\_2; i++)

array1[i] = rand() % 100 + 1;

array2 = CopyArray(array1, ARRAY\_SIZE\_2);

array3 = CopyArray(array1, ARRAY\_SIZE\_2);

}

void UpArray()

{

for (int i = 0; i < ARRAY\_SIZE\_2; i++)

array1[i] = i;

array2 = CopyArray(array1, ARRAY\_SIZE\_2);

array3 = CopyArray(array1, ARRAY\_SIZE\_2);

}

void DownArray()

{

for (int i = 0, j = ARRAY\_SIZE\_2 - 1; i < ARRAY\_SIZE\_2; i++, j--)

array1[i] = j;

array2 = CopyArray(array1, ARRAY\_SIZE\_2);

array3 = CopyArray(array1, ARRAY\_SIZE\_2);

}

void SplitArray()

{

int secondPart = ARRAY\_SIZE\_2 - ARRAY\_SIZE\_2 \* 0.5;

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

array1[i] = i;

for (int i = secondPart, j = ARRAY\_SIZE\_2 - 1; i < ARRAY\_SIZE\_2; i++, j--)

array1[i] = j;

array2 = CopyArray(array1, ARRAY\_SIZE\_2);

array3 = CopyArray(array1, ARRAY\_SIZE\_2);

}

void TestSpeedShell()

{

shell(array1, ARRAY\_SIZE\_2);

}

void TestSpeedQs()

{

qs(array2, 0, ARRAY\_SIZE\_2 - 1);

}

int compare(const void\* x1, const void\* x2) // функция сравнения элементов массива

{

return (\*(int\*)x1 - \*(int\*)x2); // если результат вычитания равен 0, то числа равны, < 0: x1 < x2; > 0: x1 > x2

}

void TestSpeedDefault()

{

qsort(array3, ARRAY\_SIZE\_2, sizeof(int), compare);

}

#pragma endregion

double CalculateSpeed(void (\*func)())

{

double time\_spent = 0.0;

clock\_t begin = clock();

func();

clock\_t end = clock();

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

return time\_spent;

}

int main()

{

srand(time(NULL));

printf("----- Exercize 1 -----\n");

printf("Time spend: %fsec\n", CalculateSpeed(Exercize1));

printf("\n----- Exercize 2 -----\n");

printf("--Random Array--\n");

RandomArray();

printf("Shell sort: %fsec\n", CalculateSpeed(TestSpeedShell));

printf("Quick sort: %fsec\n", CalculateSpeed(TestSpeedQs));

printf("Default c sort: %fsec\n", CalculateSpeed(TestSpeedDefault));

printf("--Raise Array--\n");

UpArray();

printf("Shell sort: %fsec\n", CalculateSpeed(TestSpeedShell));

printf("Quick sort: %fsec\n", CalculateSpeed(TestSpeedQs));

printf("Default c sort: %fsec\n", CalculateSpeed(TestSpeedDefault));

printf("--Down Array--\n");

DownArray();

printf("Shell sort: %fsec\n", CalculateSpeed(TestSpeedShell));

printf("Quick sort: %fsec\n", CalculateSpeed(TestSpeedQs));

printf("Default c sort: %fsec\n", CalculateSpeed(TestSpeedDefault));

printf("--Split Array--\n");

SplitArray();

printf("Shell sort: %fsec\n", CalculateSpeed(TestSpeedShell));

printf("Quick sort: %fsec\n", CalculateSpeed(TestSpeedQs));;

printf("Default c sort: %fsec\n", CalculateSpeed(TestSpeedDefault));

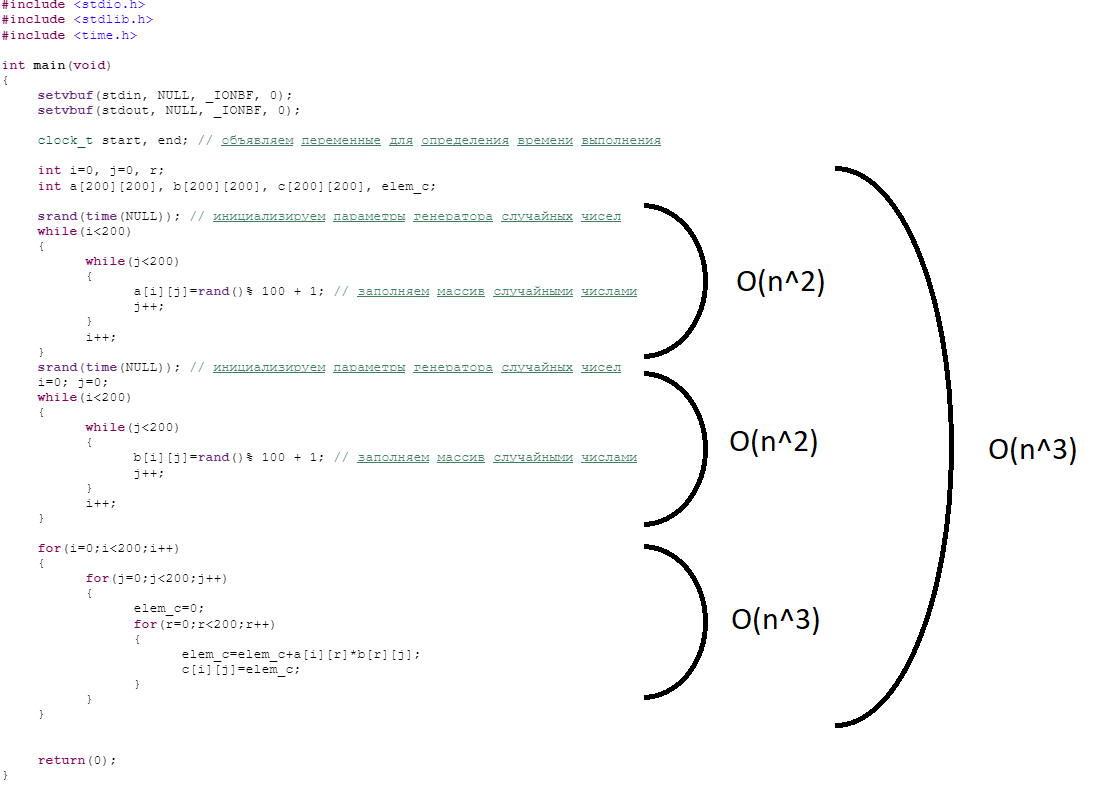
return 0;

}

**Задания**

Задание 1

1. Сложность программы: O(n^3)



1. Измерение время выполнения работы (Release)

Матрица размером 100х100: 0.000000sec

Матрица размером 200х200: 0.006000sec

Матрица размером 400х400: 0.045000sec

Матрица размером 1000х1000: 0.830000sec

Матрица размером 2000х2000: 11.241000sec

Матрица размером 4000х4000: 220.994000sec

Матрица размером 10000х10000: n/a

1. График зависимости

Задание 2 (100000 чисел):

1. Случайный набор чисел

Shell sort: 0.515000sec

Quick sort: 0.005000sec

Default c sort: 0.013000sec

1. Возрастающая последовательность чисел

Shell sort: 0.001000sec

Quick sort: 0.002000sec

Default c sort: 0.012000sec

1. Убывающая последовательность чисел

Shell sort: 1.007000sec

Quick sort: 0.002000sec

Default c sort: 0.009000sec

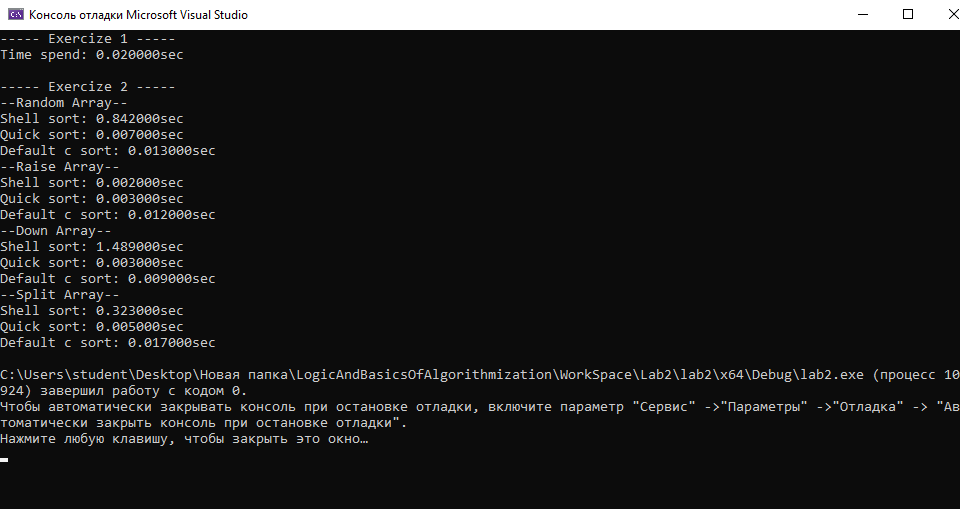
1. Половина - возрастающая последовательность чисел, половина – убывающая

Shell sort: 0.250000sec

Quick sort: 0.002000sec

Default c sort: 0.017000sec

**Результат работы программы**



**Вывод**

Я научился измерять время работы и сложность кода.