Пензенский государственный университет

Кафедра «Вычислительная техника»

**Отчёт**

По лабораторной работе №2

по дисциплине: «Логика и основы оптимизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили студенты группы 19ВВ4:

Земсков М.А

Приняли:

Митрохин М. А.

Юрова О.В.

Пенза 2020

**Цель работы:** научиться использовать библиотеку <time.h> для определения, как времени работы алгоритма, так и всей программы. Сравнить скорость выполнения различных видов сортировок при разных исходных данных.

**Практическая часть**

Дана программа, вычисляющая произведение двух матриц:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <iostream>

using namespace std;

int n = 100;

int a[100][100], b[100][100], c[100][100], elem\_c;

void main(void)

{

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

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

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

start = clock();

int i = 0, j = 0, r;

int n = 100;

int a[100][100], b[100][100], c[100][100], elem\_c;

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

while (i < n)

{

while (j < n)

{

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 < n)

{

while (j < n)

{

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

j++;

}

i++;

}

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

end = clock();

printf("Use %.0f ms", difftime(end, start));

\_getch();

}

**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.
4. O- символика данной программы: O(2n2 +n3)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| N | 100 | 200 | 400 | 1000 | 2000 | 4000 | 10000 |
| 1 | 2 | 19 | 176 | 6660 | 54716 | 527529 | 2.9 часа |
| 2 | 2 | 20 | 177 | 6617 | 53751 |
| 3 | 2 | 19 | 175 | 6567 | 53534 |

Даны реализации алгоритмов сортировки Шелла и быстрой сортировки:

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <time.h>

#include <locale.h>

#include <conio.h>

#include <string.h>

#include <stdlib.h>

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);

}

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;

}

}

}

int cmp(const void\* a, const void\* b) { //функция сравнения элементов массива

{

return \*(int\*)a - \*(int\*)b; // Если результат вычитания равен 0, то числа равны

}

}

int main()

{

int m[100000]; int x = 100000;

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

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

setlocale(LC\_ALL, "Rus");

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

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

printf("|Data\t|Shell\t|Qs\t|QSort\t|\n");

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

srand(time(NULL));

for (int i = 0; i < x; i++) { //Заполнение массива рандомными числами

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

}

start = clock();

shell(m, 10000);

end = clock();

printf("|rand\t|%.1f\t|", difftime(end, start));

srand(time(NULL));

for (int i = 0; i < x; i++) { //Заполнение массива рандомными числами

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

}

start = clock();

qs(m, 0, 10000 - 1);

end = clock();

printf("%.1f\t|", difftime(end, start));

srand(time(NULL));

for (int i = 0; i < x; i++) { //Заполнение массива рандомными числами

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

}

start = clock();

qsort(m, 10000, sizeof(int), cmp);

end = clock();

printf("%.1f\t|\n", difftime(end, start));

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

////////////////////////////////////////////////////////////////////////////////

for (int i = 1; i < x; i++) { //Возрастающая последовательность

m[i] = m[i - 1] + 1;

}

start = clock();

shell(m, 10000);

end = clock();

printf("|up\t|%.1f\t|", difftime(end, start));

for (int i = 1; i < x; i++) { //Возрастающая последовательность

m[i] = m[i - 1] + 1;

}

start = clock();

qs(m, 0, 10000 - 1);

end = clock();

printf("%.1f\t|", difftime(end, start));

for (int i = 1; i < x; i++) { //Возрастающая последовательность

m[i] = m[i - 1] + 1;

}

start = clock();

qsort(m, 10000, sizeof(int), cmp);

end = clock();

printf("%.1f\t|\n", difftime(end, start));

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

////////////////////////////////////////////////////////////////////////////////

for (int i = 0; i < x; i++) { //Убывающая последовательность

m[i] = x - i;

}

start = clock();

shell(m, 10000);

end = clock();

printf("|down\t|%.1f\t|", difftime(end, start));

for (int i = 0; i < x; i++) { //Убывающая последовательность

m[i] = x - i;

}

start = clock();

qs(m, 0, 10000 - 1);

end = clock();

printf("%.1f\t|", difftime(end, start));

for (int i = 0; i < x; i++) { //Убывающая последовательность

m[i] = x - i;

}

start = clock();

qsort(m, 10000, sizeof(int), cmp);

end = clock();

printf("%.1f\t|\n", difftime(end, start));

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

////////////////////////////////////////////////////////////////////////////////

for (int i = 0; i < x / 2; i++) { //Возрастающе-убывающая последовательность

m[i] = m[i - 1] + 1;

}

start = clock();

shell(m, 10000);

end = clock();

printf("|up/dw\t|%.1f\t|", difftime(end, start));

for (int i = 0; i < x / 2; i++) { //Возрастающе-убывающая последовательность

m[i] = m[i - 1] + 1;

}

start = clock();

qs(m, 0, 10000 - 1);

end = clock();

printf("%.1f\t|", difftime(end, start));

for (int i = 0; i < x / 2; i++) { //Возрастающе-убывающая последовательность

m[i] = m[i - 1] + 1;

}

start = clock();

qsort(m, 10000, sizeof(int), cmp);

end = clock();

printf("%.1f\t|\n", difftime(end, start));

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

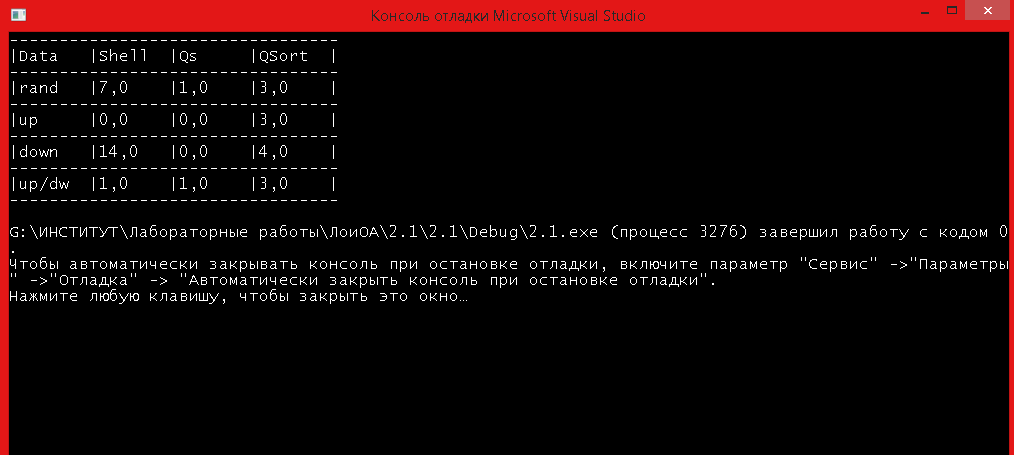
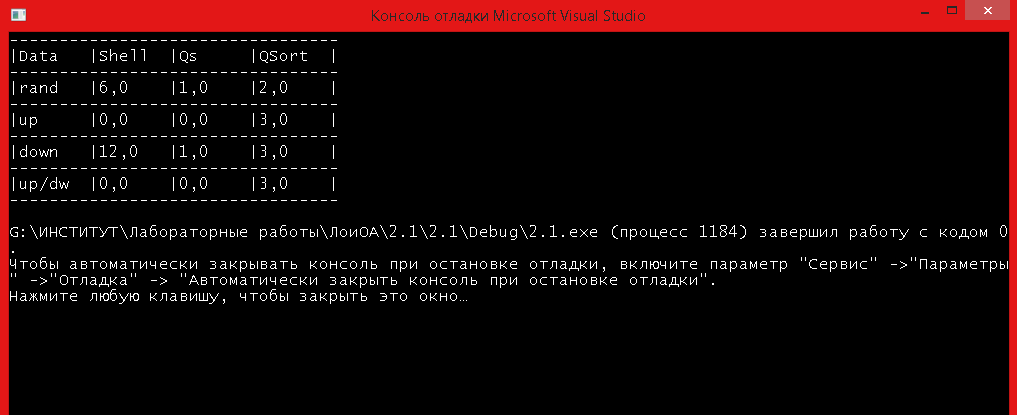
return (0);

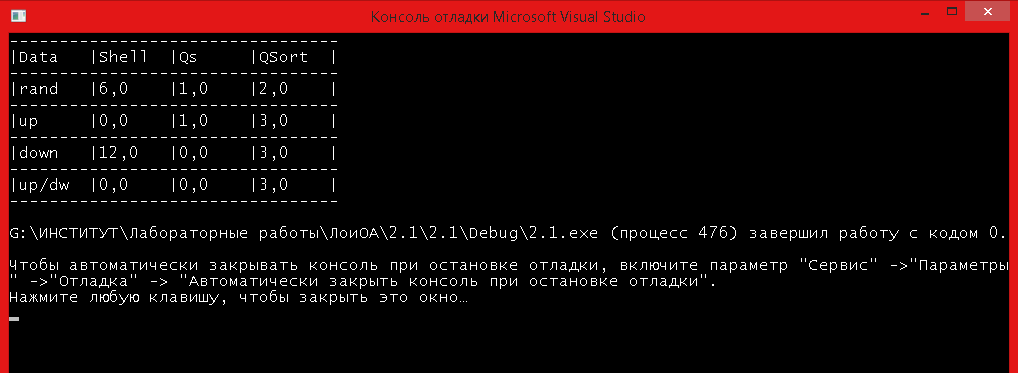
}

**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.

Оценка производилась на массиве из 100 000 элементов заполненном случайными числами.





**Вывод:** научились использовать библиотеку <time.h> для определения, как времени работы алгоритма, так и всей программы. Сравнили скорость выполнения различных видов сортировок при разных исходных данных. Сделали вывод о том, что написанная функция быстрой сортировки работает быстрее, чем встроенная в стандартную библиотеку. Алгоритм Шелла работает медленнее, чем другие алгоритмы, но если массив отсортирован, то он показывает лучшие результаты. Из этого можно сделать вывод, что алгоритм Шелла выгоднее применять, когда существует большая вероятность отсортированных исходных данных, если же данные хаотичны, то выгоднее использовать быструю сортировку.