Пензенский государственный университет

Кафедра "Вычислительной техники"

**Отчёт**

По лабораторной №2

по дисциплине "Л и ОА в ИТ"

на тему "Оценка времени выполнения программ"

***Выполнили студенты группы 20ВВ4:***

*Волков А.Ю.*

*Исаев С.Д.*

***Приняли:***

*Акифьев И.В.*

*Юрова О. В.*

Пенза 2021

# Цель работы: Оценить время работы 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.

**Теоретическая часть:**

Для оценки времени выполнения программ языка Си или их частей могут использоваться средства, предоставляемые библиотекой **time.h**. Данная библиотека содержит описания типов и прототипы функций для работы с датой и временем.

Типы данных:

1. clock\_t - возвращается функцией clock(). Обычно определён как int или long int.

2. time\_t - возвращается функцией time(). Обычно определён как int или long int.

3. struct tm - нелинейное, дискретное календарное представление времени.

**Практическая часть:**

**Задание 1.**

**Листинг:**

#include "stdio.h";

#include "stdlib.h";

#include "time.h";

unsigned short int a[1000][1000], b[1000][1000], c[10000][10000];

int main(void)

{

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

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

int i = 0, j = 0, r;

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

short int start\_time = clock();

int elem\_c;

while (i < 1000)

{

while (j < 1000)

{

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 < 1000)

{

while (j < 1000)

{

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

j++;

}

i++;

}

for (i = 0;i < 1000;i++)

{

for (j = 0;j < 1000;j++)

{

elem\_c = 0;

for (r = 0;r < 1000;r++)

{

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

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

}

}

}

short int end\_time = clock(); // конечное время

short int search\_time = end\_time - start\_time;

printf("%f", search\_time/1000.0);

return(search\_time);

}

Сложность программы: О(n2+n)

Время выполнения:

От 100 эл. - 0

От 200 эл. - 0

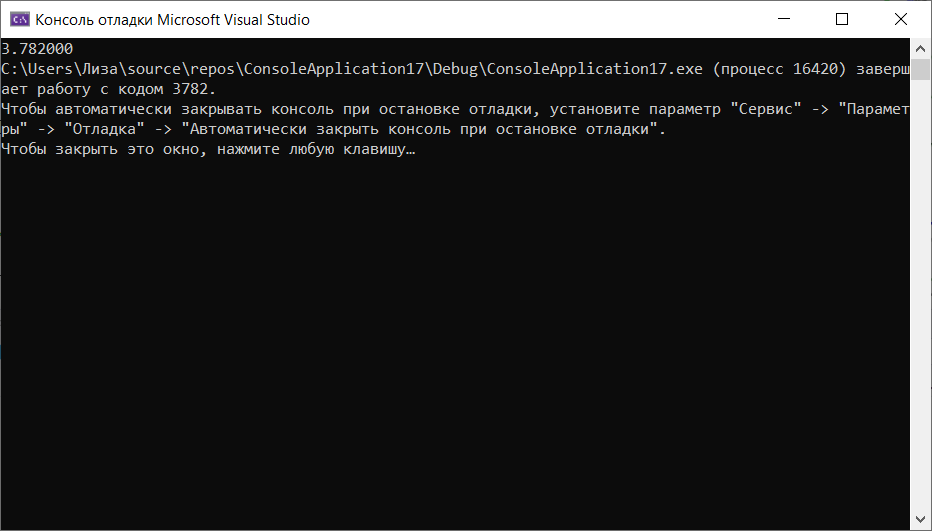
От 400 эл. - 0

От 1000 эл. - 0.004000-0.006000 мс

От 2000 эл. - 0.012000-0.016000 мс

От 4000 эл. - 0.037000-0.041000 мс

От 10000 эл.- 3.705000-3.841000 мс

**Результат работы программы:**

Построив диаграмму зависимости времени выполнения программы от размера матриц, мы подтверждаем то, что сложность данной программы: О(n2+n).

**Задание 2:**

**Листинг:**

#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>

int m[10000];

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()

{

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

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

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

//for (int i = 0; i < 10000; i++) { //Заполнение массива рандомными числами

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

//}

//for (int i = 1; i < 10000; i++) { //Возрастающая последовательность

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

//}

// for (int i = 0; i < 10000; i++) { //Убывающая последовательность

// m[i] = 10000-i;

// }

// for (int i = 0; i < 5000; i++) { //Возрастающе-убывающая последовательность

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

// }

// for (int i = 5; i < 10000; i++) {

// m[i] = 10000 - i;

// }

int start\_time = clock();

//shell(m, 10000); //Сортировка Шелла

//qs(m, 0, 10000 - 1); //Быстрая сортировка

qsort(m, 10000, sizeof(int), cmp); // Ф-я быстрой сортировки

int end\_time = clock();

int search\_time = end\_time - start\_time;

printf("%d", search\_time);

return (0);

}

Количество чисел в массиве: 10000. Время работы алгоритма измеряется в мс

Сортировка Шелла Быстрая сортировка Ф-я быстрой сортировки

Сл.набор.данных 10-16 0-2 3-4

Возраст.послед. 0 0 0

Убыв.посл. 20-24 0-1 4-7

Убыв-возраст. 20-23 0-1 7-10

30.000 чисел

Сортировка Шелла Быстрая сортировка Ф-я быстрой сортировки

Сл.набор.данных 130-206 4 14-19

Возраст.послед. 1-2 2-3 18-24

Убыв.посл. 271-296 2-4 18-27

Убыв-возраст. 250-264 2-5 31-38

**Вывод:**

1. Выяснили, что алгоритм быстрой сортировки, реализованный вручную, одинаково быстро работает на разных наборах данных в массиве
2. С увеличением количества данных в массиве, увеличивается, время, затрачиваемое на сортировку
3. Сортировка Шелла требует наибольше количество времени на работу, так как он сравнивает не только элементы, стоящие рядом, но и на определённом расстоянии
4. Функция быстрой сортировки из библиотеки требует большее количество времени, чем реализованный вручную