Пензенский государственный университет  
Кафедра «Вычислительной техники»

**Отчет**по лабораторной работе №2  
по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»  
на тему: «Оценка времени выполнения программ»

**Выполнил студент группы 19ВВ1:**

Бабалаев А.А.

**Приняли:**

д.т.н. профессор Митрохин М. А.

к.т.н. Юрова О.В.

Пенза 2020.

# 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.

**Задание 1.**

**Листинг:**

// L2.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

const int N = 1000;

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

int main(void)

{

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

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

double start, end; // объявляем переменные для определения времени выполнения

FILE\* fp = NULL;

char name[] = "2.txt";

int i = 0, j = 0, r;

fp = fopen(name, "a");

if (fp != NULL) {

printf("Файл запущен удачно \n");

}

else {

printf("Ошибка!\n");

exit(0);

}

fprintf(fp, " %d Элементов - ",N);

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++;

}

start = clock();

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();

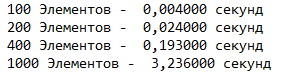
fprintf(fp, " %lf секунд\n", (end - start) / CLOCKS\_PER\_SEC);

fclose(fp);

return(0);

}

1. Сложность программы: О(n3)
2. Время выполнения:



1. Построил график зависимости времени выполнения программы от размера матриц.

Построив диаграмму зависимости времени выполнения программы от размера матриц, мы подтверждаем то, что сложность данной программы: О(n3).

**Задание 2:**

**Листинг:**

// L2.1.cpp : Этот файл содержит функцию "main". Здесь начинается и заканчивается выполнение программы.

//

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

const int N = 10000;

int m[N];

double start, end;

FILE\* fp = NULL;

char name[] = "2.txt";

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 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 timesort(int\* items) {

fprintf(fp, " Сортировка Шелла\t");

start = clock();

shell(items, N); //Сортировка Шелла

end = clock();

fprintf(fp, " %lf \n", (end - start) / CLOCKS\_PER\_SEC);

fprintf(fp, " Быстрая сортировка \t");

start = clock();

qs(items, 0, N - 1); //Быстрая сортировка

end = clock();

fprintf(fp, " %lf \n", (end - start) / CLOCKS\_PER\_SEC);

fprintf(fp, " Ф-я Быстрая сортировка \t");

start = clock();

qsort(items, N, sizeof(int), compare);

end = clock();

fprintf(fp, " %lf \n", (end - start) / CLOCKS\_PER\_SEC);

}

int main()

{

srand(time(NULL));

int i = 0, j = 0, r;

fp = fopen(name, "a");

if (fp != NULL) {

printf("Файл запущен удачно \n");

}

else {

printf("Ошибка!\n");

exit(0);

}

fprintf(fp, " Заполнение массива рандомными числами \n");

for (i = 0; i < N; i++) {

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

}

timesort(m);

fprintf(fp, " Возрастающая последовательность \n");

for (i = 0; i < N; i++) {

m[i] = i;

}

timesort(m);

fprintf(fp, " Убывающая последовательность \n");

for (i = 0; i < N; i++) {

m[i] = N-i;

}

timesort(m);

fprintf(fp, " Возрастающе-убывающая последовательность \n");

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

m[i] = i;

}

for (i = N/2; i < N; i++) {

m[i] = N-i+1;

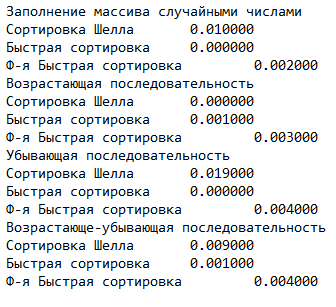
}

timesort(m);

fclose(fp);

return(0);

}



**Вывод:** научился оценивать сложность программы, измерять время работы программы и ее отдельных участков. Выяснил - какие методы сортировок занимают больше времени, а какие меньше.