Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

# по дисциплине: «Логика и основы алгоритмизации в инженерных задачах»

на тему: «Оценка времени выполнения программ»

Выполнили:

студенты группы 20ВВ3

Тихонов Андрей

Скирдова Вероника

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2021

**Название**

Оценка времени выполнения программ

**Цель работы**

Изучение способов перемножения матриц, сортировка Шелла, быстрой сортировки и расчет сложности выполнения алгоритма, расчет времени работы программы на разных наборах данных.

**Методические указания**

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.

Основные функции:

1. clock\_t clock(void) - возвращает время, измеряемое процессором в тактах от начала выполнения программы, или −1, если оно не известно. Пересчет этого времени в секунды выполняется по формуле:

clock() / CLOCKS\_PER\_SEC

где CLOCKS\_PER\_SEC – константа, определяющая количество тактов системных часов в секунду.

1. time\_t time(time\_t \*tp)

Возвращает текущее календарное время или −1, если это время не известно. Если указатель tp не равен NULL, то возвращаемое значение записывается также и в \*tp.

1. double difftime(time\_t time2,time\_t time1)

Возвращает разность time2-time1, выраженную в секундах.

**Лабораторное задание**

**Задание 1:**

Дана программа, вычисляющая произведение двух матриц.

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:**

Даны реализации алгоритмов сортировки Шелла и быстрой сортировки:

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.

**Задание 1.**

**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int a[2000][2000], b[2000][2000], c[2000][2000], elem\_c;

int main(void)

{

setlocale(LC\_ALL, "Russian");

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

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

time\_t begin\_clock, end\_clock;

clock\_t start, end;

int i = 0, j = 0, r;

srand(time(NULL));

while (i < 2000)

{

while (j < 2000)

{

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

j++;

}

i++;

}

srand(time(NULL));

i = 0; j = 0;

while (i < 2000)

{

while (j < 2000)

{

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

j++;

}

i++;

}

start = clock();

for (i = 0; i < 2000; i++)

{

for (j = 0; j < 2000; j++)

{

elem\_c = 0;

for (r = 0; r < 2000; r++)

{

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

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

}

}

}

end = clock();

double time = difftime(end, start) / CLOCKS\_PER\_SEC;

printf(" Время умножения двух массивов составило - %f секунд\n", time);

return(0);

}

**Результат работы программы**

1. Сложности программы:O(2n2)+O(n3)
2. Время выполнения программы, перемножающей две матрицы различного размера.

Матрицы размерами 100х100 (рис. 1)



Рисунок 1

Матрицы размерами 200х200 (рис. 2)



Рисунок 2

Матрицы размерами 400х400 (рис. 3)



Рисунок 3

Матрицы размерами 1000х1000 (рис. 4)



Рисунок 4

Матрицы размерами 2000х2000 (рис. 5)



Рисунок 5

Матрицы размерами 4000х4000 (рис. 6)



Рисунок 6

Матрицы размерами 10000х10000 (рис. 7)



Рисунок 7

1. График зависимости времени выполнения программы от размера матриц

Рисунок 8

На основании анализа данных, полученных в результате тестирования алгоритма перемножения двух матриц, можно сделать вывод о том, что время, затраченное на работу программы, линейно зависит от размера матриц, то есть с увеличением количества элементов в массиве так же увеличивается и время работы программы. Зависимость времени перемножения от размера матриц представлена на рисунке 8.

**Задание 2.**

**Листинг**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale>

int mas[1000000];

int size = 1000000;

int compare(const void\* x1, const void\* x2)

{

return (\*(int\*)x1 - \*(int\*)x2);

}

void shell(int\* items, int count)

{

clock\_t start, end;

int i, j, gap, k;

int x, a[5];

a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;

start = clock();

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;

}

}

end = clock();

double time = difftime(end, start) / CLOCKS\_PER\_SEC;

printf(" Время затраченное на сортировку Шелла составило - %f секунд", time);

}

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 main() {

setlocale(LC\_ALL, "Russian");

for (int i = 0; i < 100000; i++) {

mas[i] = rand() % 100;

}

//

mas[0] = 1;

for (int i = 0; i < 1000000; i++) {

mas[i] = mas[i] + 1;

}

//

mas[0] = 10000000;

for (int i = 0; i < 1000000; i++) {

mas[i] = mas[i] - 1;

}

//

mas[0] = 1;

for (int i = 0; i < 500000; i++) {

mas[i] = mas[i] - 1;

}

for (int i = 500000; i < 1000000; i++) {

mas[i] = mas[i] - 1;

}

//

shell(mas, size);

clock\_t start, end;

start = clock();

qs(mas, 0, size - 1);

end = clock();

double time = difftime(end, start) / CLOCKS\_PER\_SEC;

printf("\n Время затраченное на быструю сортировку составило - %f секунд", time);

//

clock\_t start, end;

start = clock();

qsort(mas, 1000000, sizeof(int), compare);

end = clock();

double time = difftime(end, start) / CLOCKS\_PER\_SEC;

printf("\n Время, затраченное на быструю сортировку c помощью стандартной библиотеки, составило - %f секунд\n", time);

}

**Результат работы программы**

Сортировка Шелла

Случайный набор (рис. 9)



Рисунок 9

По возрастанию (рис. 10)



Рисунок 10

По убыванию (рис. 11)



Рисунок 11

Первая половина возрастает, вторая убывает (рис. 12)



Рисунок 12

Быстрая сортировка

Случайный набор (рис. 13)



Рисунок 13

По возрастанию (рис. 14)



Рисунок 14

По убыванию (рис. 15)



Рисунок 15

Первая половина возрастает, вторая убывает (рис. 16)



Рисунок 16

Быстрая сортировка с использованием стандартной функции

Случайный набор (рис. 17)



Рисунок 17

По возрастанию (рис. 18)



Рисунок 18

По убыванию (рис. 19)



Рисунок 19

Первая половина возрастает, вторая убывает (рис. 20)



Рисунок 20

**Вывод**

В ходе выполнения данной лабораторной работы были изучены способы перемножения матриц, сортировки Шелла, быстрой сортировки, выполнен расчет сложности выполнения алгоритма, расчет времени работы программы на разных наборах данных.