Министерство н образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

студенты группы 22BВВ1

Митрошин Ю.Е

Коннов А.Д

Приняли:

Акифьев И.В

Юрова О.В

Пенза 2023

**Название**

Оценка времени выполнения программ

**Общие сведения.**

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.

2. 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.

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

    Возвращает разность time2-time1, выраженную в секундах.

**Практическая часть**

Дана программа, вычисляющая произведение двух матриц:

**#include** <stdio.h>

**#include** <stdlib.h>

**#include** <time.h>

**int** **main**(**void**)

{

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

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

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

**int** i=0, j=0, r;

**int** a[200][200], b[200][200], c[200][200], elem\_c;

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

**while**(i<200)

{

**while**(j<200)

{

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<200)

{

**while**(j<200)

{

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

j++;

}

i++;

}

**for**(i=0;i<200;i++)

{

**for**(j=0;j<200;j++)

{

elem\_c=0;

**for**(r=0;r<200;r++)

{

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

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

}

}

}

**return**(0);

}

**Задание 1:**

1. Вычислить порядок сложности программы (*О*-символику).

**Внутренний цикл выполняется n раз для каждой из n\*n итераций внешних циклов i и j. Таким образом, у нас есть O(n \* n \* n) = O(n^3) операций умножения и сложения.**

1. 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.

|  |  |
| --- | --- |
| Размер | Время |
| 100 | 0,013 |
| 200 | 0,020 |
| 400 | 0,169 |
| 1000 | 3,152 |
| 2000 | 38,794 |
| 4000 | 519,212 |
| 10000 | 5838,156 |

1. Построить график зависимости времени выполнения программы от размера матриц и сравнить полученный результат с теоретической оценкой.

**Вывод:** в ходе выполнения 1 части лабораторной работы была теоритически определена сложность алгоритма, равная O(n^3), после чего подтверждена практическими данными. Были составлены графики, подтверждающие наше умозаключение.

Даны реализации алгоритмов сортировки Шелла и быстрой сортировки:

**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);

}

**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.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100000 | возрастающий | убывающий | пилообразная | рандом |
| шелла | 0,001c | 1,312c | 0,653c | 0,631c |
| qs | 0,003c | 0,003c | 0,008c | 0,008c |
| qsort | 0,008c | 0,008c | 0,034c | 0,015c |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 300000 | возрастающий | убывающий | пилообразная | рандом |
| шелла | 0,004c | 11,765c | 5,901c | 5,925c |
| qs | 0,009c | 0,010c | 0,027c | 0,025c |
| qsort | 0,026c | 0,029c | 0,113c | 0,036c |

**Вывод:** исходя из полученных данных самым эффективным способом является сортировка qs, сортировка Шелла показала себя хуже всего. Быстрее всего происходила сортировка возрастающих и убывающих данных.

**Листинг**

для задания №1

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int a[4000][4000], b[4000][4000], c[4000][4000], elem\_c;

int main(void)

{

setlocale(LC\_ALL, "Rus");

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

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

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

int i = 0, j = 0, r;

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

while (i < 4000)

{

while (j < 4000)

{

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 < 4000)

{

while (j < 4000)

{

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

j++;

}

i++;

}

start = clock(); // начало замера времени

for (i = 0; i < 4000; i++)

{

for (j = 0; j < 4000; j++)

{

elem\_c = 0;

for (r = 0; r < 4000; r++)

{

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

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

}

}

}

end = clock(); // конец замера времени

double cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время выполнения программы: %f секунд\n", cpu\_time\_used);

return(0);

}

для задания №2

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

void fillArrayRandom(int\* items, int count) {

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

items[i] = rand() % 1000;

}

}

void fillArrayIncreasing(int\* items, int count) {

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

items[i] = i;

}

}

void fillArrayDecreasing(int\* items, int count) {

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

items[i] = count - i;

}

}

void fillArrayIncreasingDecreasing(int\* items, int count) {

int middle = count / 2;

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

items[i] = i;

}

for (int i = middle; i < count; i++) {

items[i] = count - i;

}

}

int compare(const void\* a, const void\* b) { //для стандартной функции qsort

return (\*(int\*)a - \*(int\*)b);

}

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 main() {

srand(time(NULL)); // Инициализация генератора случайных чисел

setlocale(LC\_ALL, "Rus");

int count = 200000; // Размер массива

int\* items = (int\*)malloc(sizeof(int) \* count); // Выделение памяти для массива

fillArrayRandom(items, count);

//fillArrayIncreasing(items, count);

//fillArrayDecreasing(items, count);

//fillArrayIncreasingDecreasing(items, count);

clock\_t start\_time = clock(); // Засекаем начальное время

//shell(items, count);

//qs(items, 0, count - 1);

//qsort(items, count, sizeof(int), compare);

clock\_t end\_time = clock(); // Засекаем конечное время

double elapsed\_time = (double)(end\_time - start\_time) / CLOCKS\_PER\_SEC;

printf("Время выполнения сортировки : %f секунд\n", elapsed\_time);

free(items); // Освобождаем память, выделенную для массива

return 0;

}

**Вывод:** в ходе лабораторной работы мы изучили средства, предоставляемые библиотекой **time.h**