Министерство науки и высшего 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 22ВВВ1

Пяткин М.Э.

Матросов А.М.

Принял:

к.т.н. доцент Юрова О.В.

к.э.н. доцент Акифьев И.В.

Пенза 2023

**Цель работы**

Оценить время выполнения данных 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.

### Лабораторное задание

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).

Самый сложный участок программы – трижды вложенный цикл FOR. Каждый элемент требуется обработать n \* n \* n раз.

O(n3).

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 | 0,003 |
| 200 | 0,018 |
| 400 | 0,160 |
| 1000 | 3,965 |
| 2000 | 46,701 |
| 4000 | 450,186 |
| 10000 | 5693,117 |

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

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);

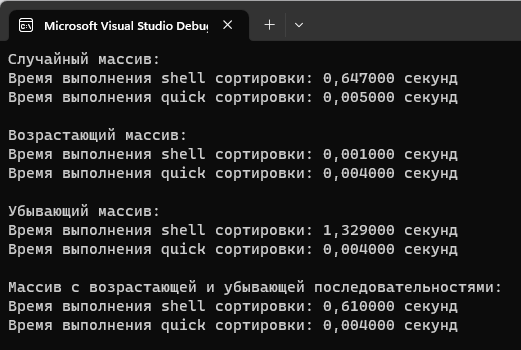
**if**(left < j) qs(items, left, j);

**if**(i < right) qs(items, i, right);

}

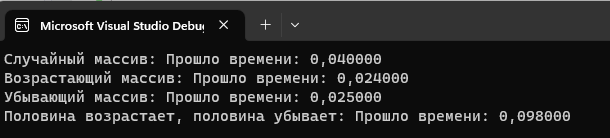
**Задание 2**:

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.



1. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

|  |  |  |  |
| --- | --- | --- | --- |
| (100000 элементов) | Время | | |
| qsort | qs | shell |
| Случайный | 0,040 | 0,005 | 0,647 |
| Возрастающий | 0,024 | 0,004 | 0,001 |
| Убывающий | 0,025 | 0,004 | 1,329 |
| Возраст./убыв. | 0,098 | 0,004 | 0,610 |



### Выводы

В ходе выполнения лабораторной работы, оценили время выполнения программы и алгоритмов на разных наборах данных.\

**Задание 1:** Вычислил порядок сложности алгоритма (O(n3))

Построил график зависимости времени выполнения программы от размера матриц и сравнил полученный результат с теоретической оценкой. В результате тестирования программы, определил, что вычисленный порядок сложности совпадает со сложностью программы.

**Задание 2:** В результате тестирования определил:

Со случайным набором данных быстрее всего справляется алгоритм qs

С возрастающим – qs

С убывающим – qs

Одна половина возрастает, другая убывает – qs

Наиболее долго сортируется убывающий массив, когда мы используем сортировку Шелла.

### Листинг

Задание 1:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define SIZE 100

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[SIZE][SIZE], b[SIZE][SIZE], c[SIZE][SIZE], elem\_c;

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

while (i < SIZE)

{

while (j < SIZE)

{

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 < SIZE)

{

while (j < SIZE)

{

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

j++;

}

i++;

}

start = clock(); // запускаем таймер

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

end = clock(); // останавливаем таймер

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

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

return(0);

}

Задание 2 (1-4):

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#include <string.h>

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);

}

void measureSortingTime(int\* items, int count, const char\* sortingAlgorithm)

{

clock\_t start\_time = clock();

if (strcmp(sortingAlgorithm, "shell") == 0) {

shell(items, count);

}

else if (strcmp(sortingAlgorithm, "quick") == 0) {

qs(items, 0, count - 1);

}

clock\_t end\_time = clock();

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

printf("Время выполнения %s сортировки: %f секунд\n", sortingAlgorithm, time\_taken);

}

void randomArraySortingTime()

{

int count = 100000; // Размер массива

int items[100000];

srand(time(NULL));

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

items[i] = rand() % 100; // Генерация случайных чисел от 0 до 99

}

measureSortingTime(items, count, "shell");

measureSortingTime(items, count, "quick");

}

void ascendingArraySortingTime()

{

int count = 100000; // Размер массива

int items[100000];

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

items[i] = i;

}

measureSortingTime(items, count, "shell");

measureSortingTime(items, count, "quick");

}

void descendingArraySortingTime()

{

int count = 100000; // Размер массива

int items[100000];

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

items[i] = count - i;

}

measureSortingTime(items, count, "shell");

measureSortingTime(items, count, "quick");

}

void mixedArraySortingTime()

{

int count = 100000; // Размер массива

int items[100000];

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

items[i] = i;

}

for (int i = count / 2; i < count; i++) {

items[i] = count - i;

}

measureSortingTime(items, count, "shell");

measureSortingTime(items, count, "quick");

}

int main() {

setlocale(LC\_ALL, "Rus");

printf("Случайный массив:\n");

randomArraySortingTime();

printf("\nВозрастающий массив:\n");

ascendingArraySortingTime();

printf("\nУбывающий массив:\n");

descendingArraySortingTime();

printf("\nМассив с возрастающей и убывающей последовательностями:\n");

mixedArraySortingTime();

return 0;

}

Задание 2 (5):

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

#define SIZE 100000 // Измените размер массива по желанию.

// Функция сравнения для qsort

int compare(const void\* a, const void\* b) {

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

}

void fill\_random(int array[], int size) {

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

array[i] = rand() % size;

}

}

void fill\_ascending(int array[], int size) {

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

array[i] = i;

}

}

void fill\_descending(int array[], int size) {

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

array[i] = size - i;

}

}

void fill\_halfs(int array[], int size) {

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

array[i] = i;

}

for (int i = size / 2; i < size; i++) {

array[i] = size - i;

}

}

void measure\_qsort(void (\*fill)(int[], int)) {

int array[SIZE];

fill(array, SIZE);

clock\_t start = clock();

qsort(array, SIZE, sizeof(int), compare);

clock\_t end = clock();

printf("Прошло времени: %f\n", ((double)end - start) / CLOCKS\_PER\_SEC);

}

int main() {

setlocale(LC\_ALL, "Rus");

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

printf("Случайный массив: ");

measure\_qsort(fill\_random);

printf("Возрастающий массив: ");

measure\_qsort(fill\_ascending);

printf("Убывающий массив: ");

measure\_qsort(fill\_descending);

printf("Половина возрастает, половина убывает: ");

measure\_qsort(fill\_halfs);

return 0;

}