Пензенский государственный университет

Кафедра «Вычислительная техника»

ОТЧЕТ

по лабораторной работе № 2

по дисциплине: "Логика и ОА в ИЗ"

на тему: "Оценка времени выполнения программ"

Выполнили:

Трундов Н.А.

Евдокимов Р.E.

Приняли:

Акифьев И.В.

Юрова О.В.

Пенза, 2023

**Общие сведения.**

**Цель:** Научиться оценивать время работы программы, пользоваться алгоритмами быстрой сортировки и сортировки Шелла, научиться оценивать сложность работы программы.

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;

}

}

}

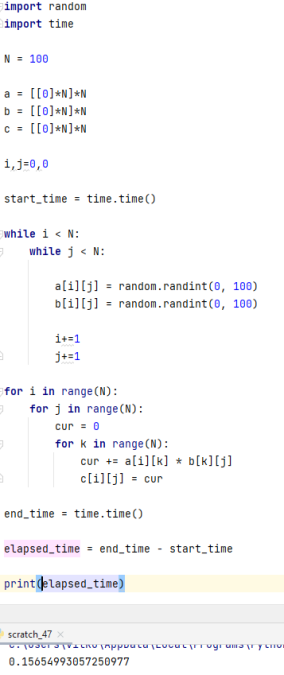
**return**(0);

}

**Задание 1:**

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

Сложность инициализации и заполнения массива a составляет O(1)  
  
Сложность инициализации и заполнения массива b составляет O(1)  
  
Сложность умножения матриц составляет O(n3)  
  
Таким образом, итоговая сложность программы будет определяться умножением матриц и составляет 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.











Результат работы в питоне



После кол-ва чисел = 4000 - время работы программы становится очень большим.

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.

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

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

items[i] = rand() % 1000;

}

}

1. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.

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

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

items[i] = i;

}

}

1. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

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

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

items[i] = count - i;

}

}

1. Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

void IncreasingDecreasingArr(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;

}

}

1. Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных. (Кол-во данных = 20000.)

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

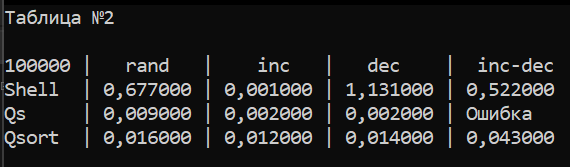
return (\*(int\*)a - \*(int\*)b);

}

qsort(arr, count, sizeof(int), compare);

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 100000 | Массив случайных чисел. (сек.) | Массив возрастающих чисел. (сек.) | Массив убывающих чисел. (сек.) | Массив возрастающих-убывающих чисел. (сек.) |
| Shell | 0.580 | 0.001 | 1.133 | 0.570 |
| Qs | 0.009 | 0.002 | 0.002 | Ошибка |
| Qsort | 0.013 | 0.009 | 0.008 | 0.034 |

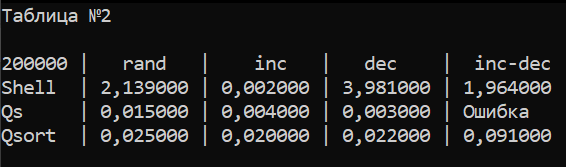
2 Компьютер.

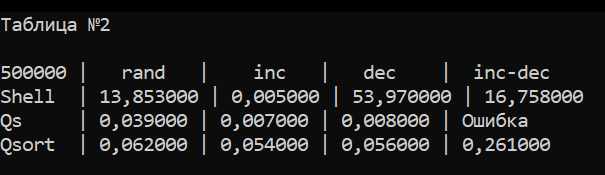


Как можно заметить по первой таблице – Shell является самым медленным алгоритмом из всех предложенных, однако на массиве возрастающих чисел он сработал быстрее остальных, а на убывающих – меделеннее остальных. Алгоритмы быстрой сотрировки работали быстрее алгоритма Shell, однако, написанный вручную алгоритм быстрой сортировки не сработал на массиве возростающих-убывающих чисел. Алгоритм Shell сработал очень медленно на массиве убывающих числе, а Qs b Qsort – так же, как и на массиве возростающих. Хоть алгоритм Qs является более быстрым, чем встроенный Qsort – однако работает он не всегда.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 200000 | Массив случайных чисел. (сек.) | Массив возрастающих чисел. (сек.) | Массив убывающих чисел. (сек.) | Массив возрастающих-убывающих чисел. (сек.) |
| Shell | 2.289 | 0.003 | 4.531 | 2.265 |
| Qs | 0.014 | 0.004 | 0.004 | Ошибка |
| Qsort | 0.025 | 0.016 | 0.019 | 0.072 |

2 Компьютер.





На второй таблице ситуация аналогичная, только можно заметить, что сложность работы алгоритма Shell в несколько раз больше алгоритма быстрой сортировки, потому что данные на Shell выросли гораздо сильнее, чем данные на Qs и Qsort.

**Вывод:**

Вспомнили, как вычислять сложность прогрммы. Вспомнили такие алгоритмы сортировки, как Shell, Qs, Встроенный Qsort. Вспомнили, как работать с таймером и замерять время работы определенных алгоритмов.

**Приложение А  
Листинг**

**Задача 1**

#define \_CRT\_SECURE\_NO\_WARNINGS // Исправление ошибок fopen

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int main(void)

{

setlocale(LC\_ALL, "RUS"); // Локализация Русского языка

clock\_t start, end;

double cpu\_time\_used;

int i, j, k, N, l;

printf("Введите размер матриц: ");

scanf("%d", &N);

int\*\* a = (int\*\*)malloc(N \* sizeof(int\*));

int\*\* b = (int\*\*)malloc(N \* sizeof(int\*));

int\*\* c = (int\*\*)malloc(N \* sizeof(int\*));

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

{

a[i] = (int\*)malloc(N \* sizeof(int));

b[i] = (int\*)malloc(N \* sizeof(int));

c[i] = (int\*)malloc(N \* sizeof(int));

}

srand(time(NULL));

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

{

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

{

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

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

}

}

start = clock();

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

{

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

{

l= 0;

for (k = 0; k < N; k++)

{

l += a[i][k] \* b[k][j];

c[i][j] = l;

}

}

}

end = clock();

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

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

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

{

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

return 0;

}

**Задача 2**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

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

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

items[i] = rand() % 1000;

}

}

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

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

items[i] = i;

}

}

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

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

items[i] = count - i;

}

}

void IncreasingDecreasingArr(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 = 100000; // Размер массива

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

double ransht, ranqst, ranqsot;

double incsht, incqst, incqsot;

double decsht, decqst, decqsot;

double idsht, idqsot; //idqst;

clock\_t start\_time, end\_time;

RandomArr(arr, count);

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

shell(arr, count);

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

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

RandomArr(arr, count);

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

qs(arr, 0, count - 1);

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

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

RandomArr(arr, count);

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

qsort(arr, count, sizeof(int), compare);

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

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

IncreasingArr(arr, count);

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

shell(arr, count);

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

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

IncreasingArr(arr, count);

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

qs(arr, 0, count - 1);

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

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

IncreasingArr(arr, count);

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

qsort(arr, count, sizeof(int), compare);

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

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

DecreasingArr(arr, count);

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

shell(arr, count);

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

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

DecreasingArr(arr, count);

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

qs(arr, 0, count - 1);

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

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

DecreasingArr(arr, count);

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

qsort(arr, count, sizeof(int), compare);

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

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

IncreasingDecreasingArr(arr, count);

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

shell(arr, count);

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

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

//IncreasingDecreasingArr(arr, count);

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

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

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

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

IncreasingDecreasingArr(arr, count);

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

qsort(arr, count, sizeof(int), compare);

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

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

printf("Таблица №2\n\n");

printf("%d | rand | inc | dec | inc-dec\n", count);

printf("Shell | %f | %f | %f | %f\n", ransht, incsht, decsht, idsht);

printf("Qs | %f | %f | %f | Ошибка\n", ranqst, incqst, decqst);//idqst);

printf("Qsort | %f | %f | %f | %f\n", ranqsot, incqsot, decqsot, idqsot);

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

return 0;

}