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;

}

}

}

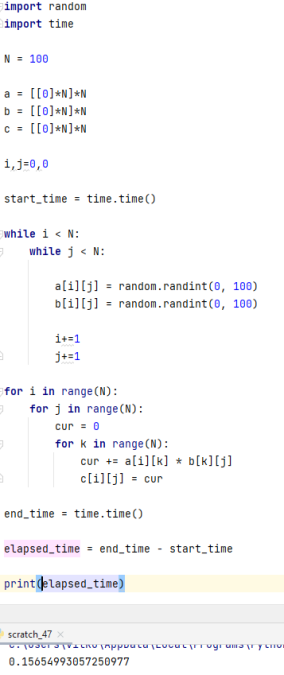
**return**(0);

}

**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.

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

{

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

{

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

}

}















Алгоритм Shell работает почти так же быстро, как и алгоритм qs.

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

(Далее будут сравниваться только 2000, 4000 и 10000 чисел, так как время выполнения на 100, 200, 4000 и 1000 числах – одинаковое.)

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

{

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

{

a[i][j] = a[i][j - 1] + 2;

}

}







Алгоритм Shell оказался немного быстрее алгоритма Qs.

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

(Попробовали изменить количество чисел на 10000, 15000, 20000.)

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

{

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

{

a[0][0] = N \* N;

a[i][j] = a[i][j - 1] - 2;

}

}

Алгоритм Shell оказался быстрее алгоритма Qs.

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

(Кол-во данных = 20000.)

for (i = 0; i < N/2; i++) // Задание 4 (2 часть)

{

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

{

a[i][j] = a[i][j - 1] + 2;

}

}

for (i = N/2 - 1; i < N; i++)

{

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

{

a[N/2-1][0] = N \* N;

a[i][j] = a[i][j - 1] - 2;

}

}





Алгоритм Shell оказался быстрее алгоритма Qs.

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

qsort(a, N, sizeof(int), comp);



Стандартная функция qsort работает быстрее, чем алгоритм Shell и Qs.

**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**

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

#include <iostream>

#include <stdio.h>

#include <stdlib.h>

#include <cstdlib>

#include <time.h>

#include <locale.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);

}

int comp(const void\* i, const void\* j)

{

return \*(int\*)i - \*(int\*)j;

}

const int N = 20000;

int a[N][N];

int main(void)

{

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

clock\_t start, end;

double cpu\_time\_used;

int i, j, k, l;

srand(time(NULL));

start = clock();

a[0][0] = N \* N;

//for (i = 0; i < N; i++) // Задания 1-3 (2 часть)

//{

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

// {

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

// //a[i][j] = a[i][j - 1] + 2;

// a[0][0] = N \* N;

// a[i][j] = a[i][j - 1] - 2;

// }

//}

for (i = 0; i < N/2; i++) // Задание 4,5 (2 часть)

{

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

{

a[i][j] = a[i][j - 1] + 2;

}

}

for (i = N/2 - 1; i < N; i++)

{

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

{

a[N/2-1][0] = N \* N;

a[i][j] = a[i][j - 1] - 2;

}

}

//shell(\*a, N);

//qs(\*a, 0, N - 1);

qsort(a, N, sizeof(int), comp);

end = clock();

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

printf("Алгоритм Qsort\n");

printf("Кол-во чисел: %d\n", N);

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

return 0;

}