ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

КАФЕДРА ВЫЧИСЛИТЕЛЬНОЙ ТЕХНИКИ

Лабораторная работа по дисциплине

«Л и ОА в ИТ»

на тему

«Оценка времени выполнения программ»

Выполнил студент группы 19ВВ1:

Петухов М. Д.

Принял:

Митрохин М. А.

Пенза 2020

**Теоретическая часть:**

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:**

**Задание 1:**

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**:

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.

**Практическая часть:**

**Листинг:**

#include "stdafx.h"

#include <time.h>

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

int a[200][200], b[200][200], c[200][200], rt[50000];

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

{

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

}

int main(void)

{

int i=0, j=0, r;

int 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;

}

}

}

long int t,t1,t2,t3;

t=clock();

printf("%ld\n",t);

j=0;

while(j<50000)

{

if (j<25000) rt[j]=j;

else

rt[j]=50000-j;

j++;

}

shell(rt,50000);

t1=clock();

printf("%ld\n",t1-t);

qs (rt,0,50000);

t2=clock();

printf("%ld\n",t2-t1);

qsort(rt, 50000, sizeof(int),compare);

t3=clock();

printf("%ld\n",t3-t2);

getch();

return(0);

}

**Задание 1:**

Кол во тактов в секунду у меня равно 1000.

1. Алгоритм работает за линейное время. О-символика равна О(n^3).
2. Итоги работы программы в тактах для массивов размерами 100, 200, 400, 1000, 2000, 4000 соответственно.



0,06 секунд.



0,124 секунд



0,637 секунд



21,159 секунд



3 минуты 34,013 секунд



20 минут 7,517 секунд.

3.

Полученный результат сходится с теоретической оценкой.

**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.

Массив из 50000 элементов. Значения в тактах. Значения в программе: Сортировка Шелла, Быстрая сортировка и qsort соответственно.

1.



2.



3.



4.



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайном наборе значений массива | Возрастающую последовательность чисел | Убывающую последовательность чисел | Одна половина возрастающую, вторая убывает |
| Сортировка Шелла | 659 | 3 | 1353 | 681 |
| Быстрая сортировка | 7 | 7 | 7 | 5 |
| qsort | 18 | 40 | 43 | 39 |

Вывод: Я проанализировал время работы разных программ и алгоритмов. Алгоритм Шелла работает эффективно только если сортируемые значения расположены в порядке возрастания. Быстрая сортировка и qsort одинаково эффективны, причем в данном варианте быстрая сортировка немного быстрее.