МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по дисциплине: "Логика и основы алгоритмизации в инженерных задачах"

на тему: "Оценка времени выполнения программ"

Выполнили:

Студенты группы 23ВВВ2

Кочегин Валерий

Лисов Егор

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2024

**Цель работы**

Изучить и измерить производительность алгоритмов, сравнить с предполагаемой сложностью программы.

**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.

**Ход работы**

**Задание 1.**

Программа имеет порядок сложности n3, т. к. она имеет 3 вложенных цикла.

**Таблица 2 – Результат работы программы 1**

|  |  |
| --- | --- |
| **Размер матрицы** | **Время выполнения программы** |
| **100** |  |
| **200** |  |
| **400** |  |
| **1000** |  |
| **2000** |  |
| **4000** |  |
| **10000** |  |

**// Что-то про результат написать, график и совпал или нет с теорией.**

**Задание 2.**

**Таблица 1 – Результат работы программы 2**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайный | Возрастающий | Убывающий | Половинчатый |
| Shell | 0.892 | 0.002 | 0.002 | 0.001 |
| Qs | 0.004 | 0.004 | 0.003 | 0.005 |
| Qsort | 0.002 | 0.001 | 0.002 | 0.002 |

Метод сортировки Shell потратил больше всех времени на Случайны массив, но при этом на Половинчатый потратил меньше всех времени.

Метод сортировки Qs больше всех времени потратил на Половинчатый массив, данный метод на других видах массивов не показывает наилучшего результата

Метод сортировки Qsort потратил меньше всего времени на сортировку всех видов массивов, кроме Половинчатого.

**Пояснительный текст к программе**

**Задание 1**

Программа выполняет умножение двух матриц и измеряет время, затраченное на данную операцию для различных размеров матриц.

**Задание 2**

Программа измеряет время выполнения сортировки тремя способами четырех видов массивов: случайный, возрастающий, убывающий, половина массива возрастает, другая – убывает.

**Результаты работы программы**

// Скрин 1 задания

Рисунок 1 — Результаты работы программы пункта №1

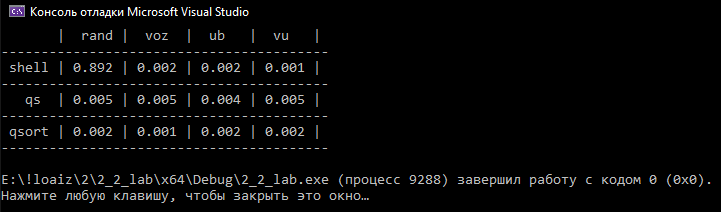


Рисунок 2 — Результаты работы программы пункта №2

### Вывод

В ходе выполнения лабораторной работы была разработана программы, выполняющие задания Лабораторной работы №2. В процессе выполнения работы были использованы знания о расчёте сложности алгоритмов и затратах времени в зависимости от величины входных данных.

**Листинг**

### Задание 1

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#define KOL\_SIZE 7

int main(void)

{

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

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

const int ARR\_SIZE[KOL\_SIZE] = { 100, 200, 400, 1000, 2000, 4000, 10000 };

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

double itog\_vr;

int i, j, r;

int elem\_c;

printf(" Size | Time\n-----------------------\n");

for (int k = 0; k < KOL\_SIZE; k++)

{

int size = ARR\_SIZE[k];

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

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

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

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

{

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

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

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

}

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();

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();

itog\_vr = ((double)(end - start)) \* 1000.0 / CLOCKS\_PER\_SEC;

printf("%6d | %lf\n", ARR\_SIZE[k], itog\_vr);

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

{

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

}

return(0);

}

**Задание 2**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <Windows.h>

const int size = 100000;

int randarr1[size], randarr2[size], randarr3[size];

int vozrast1[size], vozrast2[size], vozrast3[size];

int ubiv1[size], ubiv2[size], ubiv3[size];

int vozubiv1[size], vozubiv2[size], vozubiv3[size];

int k = 0;

double vremya[12];

void zaparr(int\* a, int\* b, int\* c, int\* d, int size)

{

int i;

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

a[i] = rand() % 1000;

b[i] = i;

c[i] = size - i;

if (i < (size / 2)) { d[i] = i; }

else { d[i] = size - i - 1; }

}

}

void kopirka(int\* a, int\* b, int\* c, int size)

{

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

b[i] = a[i];

c[i] = a[i];

}

}

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\* a, const void\* b) {

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

}

void schetShell(int\* a, int\* k)

{

clock\_t start = clock();

shell(a, size);

clock\_t end = clock();

vremya[\*k] = ((double)(end - start) / CLOCKS\_PER\_SEC);

(\*k)++;

}

void schetQs(int\* a, int\* k)

{

clock\_t start = clock();

qs(a, 0, size - 1);

clock\_t end = clock();

vremya[\*k] = ((double)(end - start) / CLOCKS\_PER\_SEC);

(\*k)++;

}

void schetQsort(int\* a, int\* k)

{

clock\_t start = clock();

qsort(a, size, sizeof(int), compare);

clock\_t end = clock();

vremya[\*k] = ((double)(end - start) / CLOCKS\_PER\_SEC);

(\*k)++;

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

zaparr(randarr1, vozrast1, ubiv1, vozubiv1, size);

kopirka(&randarr1[size], &randarr2[size], &randarr3[size], size);

kopirka(&vozrast1[size], &vozrast2[size], &vozrast3[size], size);

kopirka(&ubiv1[size], &ubiv2[size], &ubiv3[size], size);

kopirka(&vozubiv1[size], &vozubiv2[size], &vozubiv3[size], size);

schetShell(randarr1, &k);

schetShell(vozrast1, &k);

schetShell(ubiv1, &k);

schetShell(vozubiv1, &k);

schetQs(randarr2, &k);

schetQs(vozrast2, &k);

schetQs(ubiv2, &k);

schetQs(vozubiv2, &k);

schetQsort(randarr3, &k);

schetQsort(vozrast3, &k);

schetQsort(ubiv3, &k);

schetQsort(vozubiv3, &k);

printf(" | rand | voz | ub | vu |\n-----------------------------------------\n");

printf(" shell | %3.3lf | %3.3lf | %3.3lf | %3.3lf |\n-----------------------------------------\n", vremya[0], vremya[1], vremya[2], vremya[3]);

printf(" qs | %3.3lf | %3.3lf | %3.3lf | %3.3lf |\n-----------------------------------------\n", vremya[4], vremya[5], vremya[6], vremya[7]);

printf(" qsort | %3.3lf | %3.3lf | %3.3lf | %3.3lf |\n-----------------------------------------\n", vremya[8], vremya[9], vremya[10], vremya[11]);

return 0;

}