МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ  
РОССИЙСКОЙ ФЕДЕРАЦИИ  
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ  
Кафедра «Вычислительной техники»

**ОТЧЁТ**

по лабораторной работе №2  
по курсу «Логика и основы алгоритмизации в инженерных задачах»  
на тему «Оценка времени выполнения программ»

Выполнил:

Студент группы 23ВВВ2

Чупраков С. В.

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.**

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

В программе присутствует вложенный цикл 3 порядка значит сложность программы достигает n3

1. Выполнили измерения времени выполнения кода.

|  |  |
| --- | --- |
| **Размер матрицы** | **Время выполнения программы и кода** |
| **100** | **2.0000** |
| **200** | **16.0000** |
| **400** | **129.0000** |
| **1000** | **2418.0000** |
| **2000** | **25176.0000** |
| **4000** | **309660.0000** |
| **10000** | **7411483.0000** |

1. **Общая тенденция**:
   * Время выполнения операции увеличивается значительно быстрее, чем размер матрицы. Это указывает на нелинейную зависимость.
2. **Квадратичная или кубическая зависимость**:
   * Время выполнения операций над матрицами часто зависит от размера матрицы в степени, близкой к кубической. Это связано с тем, что многие операции, такие как умножение матриц, имеют временную сложность O(n3)*O*(*n*3), где *n* — размер стороны матрицы.
3. **Пример расчета**:
   * Если мы посмотрим на увеличение времени от 100 до 200 (увеличение размера в 2 раза), время увеличивается с 2 до 16, в 8 раз.
   * При увеличении размера с 200 до 400 (также в 2 раза), время увеличивается с 16 до 129, что примерно в 7.5 раз.
   * Это подтверждает гипотезу о кубической зависимости, так как 23=8.
4. **Заключение**:
   * Данные показывают, что время выполнения операции увеличивается примерно пропорционально кубу увеличения размера матрицы. Это типично для операций, таких как умножение матриц, где каждая дополнительная строка и столбец требуют значительно больше вычислений.

Таким образом, можно предположить, что временная сложность операции над матрицей в данном случае близка к O(n3)*O*(*n*3). Данное предположение было доказано с помощью анализа графика.

**Задание 2.**

Shell сортировка на случайном наборе чисел показала наибольший по времени результат в 528 мс, что показало низкую эффективность на данном типе массивов.

Три оставшихся массива при сортировке показали высокие результаты.

Представленный в условиях лабораторной работы алгоритм QS, показал высокие результаты на всех видах массивов. Алгоритм сортировки QSort уступил по скорости сортировки случайного массива, но показал лучший результат на последних трех массивах.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Случайный | Возрастающий | Убывающий | Половинчатый |
| Shell | 0.528 | 0.001 | 0.002 | 0.001 |
| QS | 0.004 | 0.004 | 0.003 | 0.003 |
| QSort | 0.012 | 0.001 | 0.001 | 0.001 |

**Пояснительный текст к программе**

Задание 1

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

Задание 2

Программа создает измеряет время выполнения сортировки тремя различными способами на 4 видах массивов: случайный, возрастающий, убывающий и массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

### Результаты работы программы

Результаты работы программы показаны на рисунках 1-5.

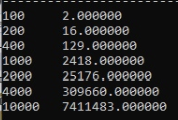


Рисунок 1 — Результаты работы программы пункта №1

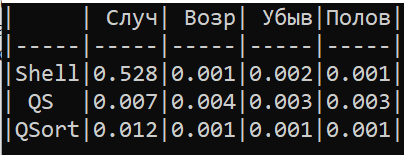


Рисунок 2 — Результаты работы программы пункта №2

### 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>

#include <Windows.h>

#define NUM\_SIZES 7

const int SIZES[NUM\_SIZES] = { 100, 200, 400, 1000, 2000, 4000, 10000 };

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

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

printf("Размер\tВремя выполнения (мс)\n");

printf("---------------------------\n");

for (int sizeIndex = 0; sizeIndex < NUM\_SIZES; sizeIndex++)

{

int SIZE = SIZES[sizeIndex];

clock\_t start=0, end =0; // объявляем переменные для определения времени выполнения

double cpu\_time\_used;

int i, j, r;

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));

}

// Запускаем таймер

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

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

{

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

{

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

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

}

}

start = clock();

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

{

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

{

int elem\_c = 0;

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

{

elem\_c += a[i][r] \* b[r][j]; // считаем элемент результата

}

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

}

}

end = clock();

// Заканчиваем таймер

cpu\_time\_used = ((double)(end - start)) \* 1000.0 / CLOCKS\_PER\_SEC; // переводим время в миллисекунды

// Вывод времени выполнения

printf("%d\t%f\n", SIZE, cpu\_time\_used);

// Освобождаем память

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 randomArray[size], randomArray2[size], randomArray3[size];

int increasingArray[size], increasingArray2[size], increasingArray3[size];

int decreasingArray[size], decreasingArray2[size], decreasingArray3[size];

int halfIncHalfDecArray[size], halfIncHalfDecArray2[size], halfIncHalfDecArray3[size];

void copyArray(int\* source, int\* destination, int\* destination2, int size) {

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

destination[i] = source[i];

destination2[i] = source[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;

}

}

}

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

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

}

void qs(int\* items, int left, int right) {

int i, j;

int x, y;

i = left; j = right;

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() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

// размер массивов

double timeR1, timeR2, timeR3, timeI1, timeI2, timeI3, timeD1, timeD2, timeD3, timeH1, timeH2, timeH3;

// Заполнение массивов

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

randomArray[i] = rand() % 1000; // случайные числа от 0 до 999

increasingArray[i] = i; // возрастающий

decreasingArray[i] = size - i; // убывающий

halfIncHalfDecArray[i] = (i < size / 2) ? i : size - i; // половина возрастающий, половина убывающий

}

copyArray(randomArray, randomArray2, randomArray3, size);

copyArray(&increasingArray[size], &increasingArray2[size], &increasingArray3[size], size);

copyArray(&decreasingArray[size], &decreasingArray2[size], &decreasingArray3[size], size);

copyArray(&halfIncHalfDecArray[size], &halfIncHalfDecArray2[size], &halfIncHalfDecArray3[size], size);

// Сортировка и замер времени для случайного массива

clock\_t start = clock();

shell(randomArray, size);

clock\_t end = clock();

timeR1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(randomArray2, 0, size - 1);

end = clock();

timeR2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(randomArray3, size, sizeof(int), compare);

end = clock();

timeR3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

shell(&decreasingArray[size], size);

end = clock();

timeI1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(&decreasingArray2[size], 0, size - 1);

end = clock();

timeI2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(&decreasingArray3[size], size, sizeof(int), compare);

end = clock();

timeI3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

shell(&increasingArray[size], size);

end = clock();

timeD1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(&increasingArray2[size], 0, size - 1);

end = clock();

timeD2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(&increasingArray3[size], size, sizeof(int), compare);

end = clock();

timeD3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

shell(&halfIncHalfDecArray[size], size);

end = clock();

timeH1 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qs(&halfIncHalfDecArray2[size], 0, size - 1);

end = clock();

timeH2 = ((double)(end - start) / CLOCKS\_PER\_SEC);

start = clock();

qsort(&halfIncHalfDecArray3[size], size, sizeof(int), compare);

end = clock();

timeH3 = ((double)(end - start) / CLOCKS\_PER\_SEC);

printf("| | Случ| Возр| Убыв|Полов|\n");

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

printf("|Shell|%3.3lf|%3.3lf|%3.3lf|%3.3lf|\n", timeR1, timeI1, timeD1, timeH1);

printf("| QS |%3.3lf|%3.3lf|%3.3lf|%3.3lf|\n", timeR2,timeI2,timeD2 , timeH2 );

printf("|QSort|%3.3lf|%3.3lf|%3.3lf|%3.3lf|\n",timeR3,timeI3,timeD3, timeH3 );

return 0;

}