Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

Студенты группы 23ВВВ2

Герасимов В.Р.

Мадамкин В. М.

Приняли:

Юрова О. В.

Митрохин М. А.

Пенза 2024

**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.

**Ход работы**

Задание 1:

1. Сложность программы составляет O(n3).
2. Оценили время выполнения программы и кода.

Время для перемножения матриц размером 100 изображено на рисунке №1:

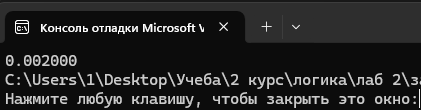


Рисунок №1

Время для перемножения матриц размером 200 изображено на рисунке №2:

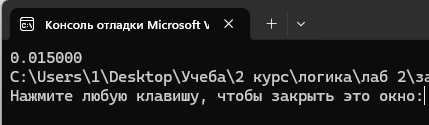


Рисунок №2

Время для перемножения матриц размером 400 изображено на рисунке №3:

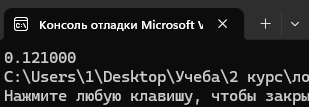


Рисунок №3

Время для перемножения матриц размером 1000 изображено на рисунке №4:

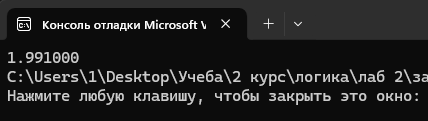


Рисунок №4

Время для перемножения матриц размером 2000 изображено на рисунке №5:

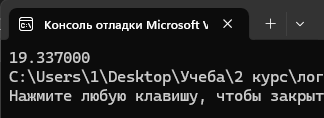


Рисунок №5

Время для перемножения матриц размером4000 изображено на рисунке №6:

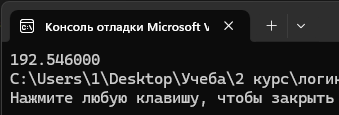


Рисунок №6

Время для перемножения матриц размером 10000 изображено на рисунке №7:

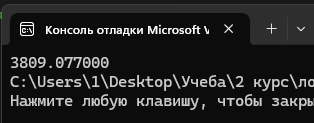


Рисунок №7

1. Построили график зависимости времени выполнения программы от размера матриц (рисунок 1) и сравнили полученный результат с теоретической оценкой.

Рисунок 1

Проанализировав данный рисунок, можно сказать, что, во-первых, график демонстрирует прямую зависимость: с увеличением размеров массивов увеличивается время выполнения программы; а во-вторых, фактический результат практически совпал с нашей теоретической оценкой.

**Задание 2**:

Оценили время работы каждого из реализованных алгоритмов на случайном наборе значений массива, на массивах, представляющих собой, возрастающую, убывающую, возрастающе-убывающую последовательности чисел. При возрастающей и возрастающе-убывающей последовательностях сортировка шелла является самой быстрой, что нельзя сказать о случайном наборе и убывающей последовательности, в них данный вид сортировки продемонстрировал себя хуже всего. Сортировка qs при случайной и убывающей последовательности показывает достойный результат, однако на возрастающей и возрастающе-убывающей последовательностях уступает сортировке шелла. Что касается сортировки qsort, она показала средний результат, уступив во всех результатах сортировке qs, но на некоторых последовательностях показала результат лучше чем сортировка шелла. (см. таблицу №1)

Таблица №1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Сортировка/набор | Случайный  набор | Возраст. Посл-ть | Убывающ. Посл-ть | Возр-убыв.  Посл-ть |
| Shell | 0.506 | 0.001 | 1.009 | 0.002 |
| Qs | 0.008 | 0.002 | 0.002 | 0.003 |
| Qsort() | 0.013 | 0.010 | 0.009 | 0.004 |

Время для сортировки матриц размером 100000 элементов разными способами при случайном наборе изображено на рисунке №8:

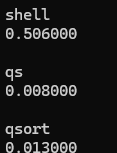


Рисунок №8

Время для сортировки матриц размером 100000 элементов разными способами при возрастающей последовательности элементов изображено на рисунке №9:

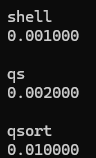


Рисунок №9

Время для сортировки матриц размером 100000 элементов разными способами при убывающей последовательности элементов изображено на рисунке №10:

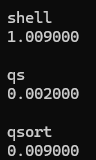


Рисунок №10

Время для сортировки матриц размером 100000 элементов разными способами при возрастающе-убывающей последовательности элементов изображено на рисунке №11:

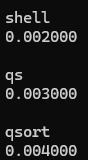


Рисунок №11

**Вывод**

С увеличением размеров массивов увеличивается время выполнения программы; а во-вторых, фактический результат практически совпал с нашей теоретической оценкой. При возрастающей и возрастающе-убывающей последовательностях сортировка шелла является самой быстрой, что нельзя сказать о случайном наборе и убывающей последовательности, в них данный вид сортировки продемонстрировал себя хуже всего. Сортировка qs при случайной и убывающей последовательности показывает достойный результат, однако на возрастающей и возрастающе-убывающей последовательностях уступает сортировке шелла. Что касается сортировки qsort, она показала средний результат, уступив во всех результатах сортировке qs, но на некоторых последовательностях показала результат лучше чем сортировка шелла.

Листинг

Задание 1

1.cpp

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int a[10000][10000], b[10000][10000], c[10000][10000], elem\_c;

int main(void)

{

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

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

double tim;

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

int i = 0, j = 0, r;

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

while (i < 10000)

{

while (j < 10000)

{

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 < 10000)

{

while (j < 10000)

{

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

j++;

}

i++;

}

start = clock();

for (i = 0; i < 10000; i++)

{

for (j = 0; j < 10000; j++)

{

elem\_c = 0;

for (r = 0; r < 10000; r++)

{

elem\_c = elem\_c + a[i][r] \* b[r][j];

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

}

}

}

end = clock();

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

printf("%f", tim);

return(0);

}