Министерство науки и высшего образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная кафедра»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили:

студент группы 23ВВВ2

Конкин Д.С.

Приняли:

Юрова О.В.

Митрохин М.А.

Пенза 2024

**Название**

Оценка времени выполнения программ

**Цель работы**

Изучение библиотекой **time.h**

**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.

**Листинг**

**Main.cpp**

#include "task.h"

int main(void) //O(n ^ 3) + (O(n^2)\*6) + (O(n)\*6) + (O(1)\*28)

{

setvbuf(stdin, NULL, \_IONBF, 0); //O(1)

setvbuf(stdout, NULL, \_IONBF, 0); //O(1)

clock\_t start\_install\_mas\_a,

start\_install\_mas\_b,

start\_multi\_mas,

start\_print\_mas\_a,

start\_print\_mas\_b,

start\_print\_mas\_result,

end\_1, end\_2, end\_3,

end\_4, end\_5, end\_6; //O(1)

int mas\_a [SIZE\_MAS][SIZE\_MAS]{0},

mas\_b [SIZE\_MAS][SIZE\_MAS]{0},

mas\_result [SIZE\_MAS][SIZE\_MAS]{0}; //O(1)

start\_install\_mas\_a = clock(); //O(1)

install\_mas(mas\_a); //O(n^2) + O(n) + O(1)

end\_1 = clock(); //O(1)

start\_install\_mas\_b = clock(); //O(1)

install\_mas(mas\_b); //O(n^2) + O(n) + O(1)

end\_2 = clock(); //O(1)

start\_print\_mas\_a = clock(); //O(1)

print\_mas(mas\_a); //O(n^2) + O(n)+ (O(1)\*2)

end\_3 = clock(); //O(1)

start\_print\_mas\_b = clock(); //O(1)

print\_mas(mas\_b); //O(n^2) + O(n)+ (O(1)\*2)

end\_4 = clock(); //O(1)

start\_multi\_mas = clock(); //O(1)

multi\_mas(mas\_a, mas\_b, mas\_result); //O(n^3) + O(n^2) + O(n) +O(1)

end\_5 = clock(); //O(1)

start\_print\_mas\_result = clock(); //O(1)

print\_mas(mas\_result); //O(n^2) + O(n)+ (O(1)\*2)

end\_6 = clock(); //O(1)

cout <<"start\_install\_mas\_a \t"<< double(end\_1 - start\_install\_mas\_a) / CLOCKS\_PER\_SEC << " seconds\n"; //O(1)

cout <<"start\_install\_mas\_b \t" << double(end\_2 - start\_install\_mas\_b) / CLOCKS\_PER\_SEC << " seconds\n"; //O(1)

cout <<"start\_print\_mas\_a \t" << double(end\_3 - start\_print\_mas\_a) / CLOCKS\_PER\_SEC << " seconds\n"; //O(1)

cout << "start\_print\_mas\_b \t" << double(end\_4 - start\_print\_mas\_b) / CLOCKS\_PER\_SEC << " seconds\n"; //O(1)

cout << "start\_multi\_mas \t" << double(end\_5 - start\_multi\_mas) / CLOCKS\_PER\_SEC << " seconds\n"; //O(1)

cout << "start\_print\_mas\_result \t" << double(end\_6 - start\_print\_mas\_result) / CLOCKS\_PER\_SEC << " seconds\n"; //O(1)

return 0; //O(1)

}

**task.h**

#pragma once

#include <iostream>

#include <cstdio>

using namespace std;

constexpr size\_t SIZE\_MAS = 200;

void install\_mas(int(&mas)[SIZE\_MAS][SIZE\_MAS]);

void print\_mas(int(&mas)[SIZE\_MAS][SIZE\_MAS]);

void multi\_mas(int(&mas\_a)[SIZE\_MAS][SIZE\_MAS], int(&mas\_b)[SIZE\_MAS][SIZE\_MAS], int(&mas\_result)[SIZE\_MAS][SIZE\_MAS]);

**install\_mas.cpp**

#include "task.h"

void install\_mas(int (&mas)[SIZE\_MAS][SIZE\_MAS]) //O(n^2) +O(n) + O(1)

{

srand(time(NULL));

for (int i = 0; i < SIZE\_MAS; ++i) //O(n)

for (int j = 0; j < SIZE\_MAS; ++j) //O(n^2)

mas[i][j] = rand() % 100 + 1; //O(1)

}

**print\_mas.cpp**

#include "task.h"

void print\_mas(int(&mas)[SIZE\_MAS][SIZE\_MAS]) //O(n^2) + O(n)+ (O(1)\*2)

{

for (int i = 0; i < SIZE\_MAS; ++i) { //O(n)

for (int j = 0; j < SIZE\_MAS; ++j) //O(n^2)

cout << mas[i][j] << " "; // O(1)

cout << endl; // O(1)

}

}

**multi\_mas.cpp**

#include "task.h"

void multi\_mas(int(&mas\_a)[SIZE\_MAS][SIZE\_MAS], int(&mas\_b)[SIZE\_MAS][SIZE\_MAS], int(&mas\_result)[SIZE\_MAS][SIZE\_MAS]) //O(n^3) + O(n^2) + O(n) +O(1)

{

for (int i = 0; i < SIZE\_MAS; i++) //O(n)

for (int j = 0; j < SIZE\_MAS; j++) //O(n^2)

for (int k = 0; k < SIZE\_MAS; k++) //O(n^3)

mas\_result[i][j] += (mas\_a[i][k] \* mas\_b[k][j]); // O(1)

}

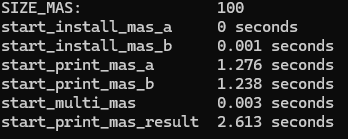


Рис.1.1

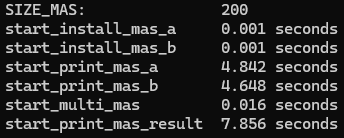


Рис.1.2

После назначение матрицы 400 шло переполнение кучи, было принято решение переписать код под динамическое выделение памяти

**Main.cpp**

#include "task.h"

int main() //O(n ^ 3) + (O(n^2)\*7) + (O(n)\*7) + (O(1)\*18)

{

Task Tz; //O(n ^ 3) + (O(n^2)\*7) + (O(n)\*7) + (O(1)\*17)

return (0); //O(1)

}

**task.h**

#pragma once

#include <iostream>

#include <cstdio>

using namespace std;

class Task //O(n ^ 3) + (O(n^2)\*7) + (O(n)\*7) + (O(n)\*16)

{

size\_t size = 1000; //O(1)

double time\_ = 0; //O(1)

int\*\* data\_a = nullptr; //O(1)

int\*\* data\_b = nullptr; //O(1)

int\*\* data\_result = nullptr; //O(1)

public:

Task() //O(n ^ 3) + (O(n^2)\*7) + (O(n)\*7) + (O(n)\*17)

{

install(data\_a); //O(n^2) + O(n) + O(1)

install(data\_b); //O(n^2) + O(n) + O(1)

install(data\_result); //O(n^2) + O(n) + O(1)

for (size\_t i = 0; i < size; i++) //O(n)

for (size\_t j = 0; j < size; j++) { //O(n^2)

data\_a[i][j] = rand() % 100 + 1; //O(1)

data\_b[i][j] = rand() % 100 + 1; //O(1)

}

multiplay(); //O(n ^ 3) + O(n^2) + O(n) + (O(1)\*2)

print\_time(); O(1)\*2

}

~Task()

{

deinstall(data\_a); //O(n^2) + O(n) + (O(1)\*2)

deinstall(data\_b); //O(n^2) + O(n) + (O(1)\*2)

deinstall(data\_result); //O(n^2) + O(n) + (O(1)\*2)

}

void install(int\*\*& data) //O(n^2) + O(n) + O(1)

{

data = new int\* [size]; //O(n)

for (size\_t i = 0; i < size; i++) { //O(n^2)

data[i] = new int[size]; //O(1)

}

}

void deinstall(int\*\* data) //O(n^2) + O(n) + (O(1)\*2)

{

if (data) //O(1)

{

for (size\_t i = 0; i < size; i++) //O(n)

delete[] data[i]; //O(1)

delete[] data; //O(n^2)

}

}

void multiplay() //O(n ^ 3) + O(n^2) + O(n) + (O(1)\*2)

{

clock\_t start = clock(); //O(1)

for (int i = 0; i < size; i++) //O(n)

for (int j = 0; j < size; j++) //O(n^2)

for (int k = 0; k < size; k++) //O(n^3)

data\_result[i][j] += (data\_a[i][k] \* data\_b[k][j]); //O(1)

clock\_t end = clock(); //O(1)

time\_ = static\_cast<double>(end - start) / CLOCKS\_PER\_SEC; //O(1)

}

void print\_time() //O(1)\*2

{

std::cout << "size: " << size << std::endl; //O(1)

std::cout << "teme multiplay: " << time\_ << std::endl; //O(1)

}};

**Порядок сложности**

O(n ^ 3) + (O(n^2)\*7) + (O(n)\*7) + (O(1)\*18)

**Тест**

{6516FD6D-0F1A-4AC9-89A0-591BE8291539}

Рис.2.1

{10B3A089-A2BB-4C22-B43F-FB2C1C90EC9B}

Рис.2.2

{1E5B39EB-64B2-4641-BE78-2070396DC218}

Рис.2.3

{5ACD6E82-E1AB-4A10-A4ED-63A38961024D}

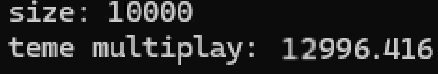
Рис.2.4

{C58F02AE-EE46-454B-9577-61E8CEE36A51}

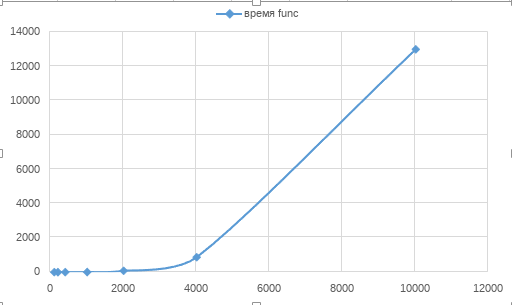
Рис.2.5

{FE14CA44-F0CD-44CE-BB22-1C57F8A3F0B2}

Рис.2.6



|  |  |  |
| --- | --- | --- |
| id | size | time |
| 1 | 100 | 0.003 |
| 2 | 200 | 0.035 |
| 3 | 400 | 0.203 |
| 4 | 1000 | 6.413 |
| 5 | 2000 | 81.133 |
| 6 | 4000 | 870.548 |
| 7 | 10000 | 12996.416 |



**Задание 2**:

**Листинг (**shell**)**

**Main.cpp**

#include "task.h"

int main()

{

Task Tz;

return (0);

}

**task.h**

#pragma once

#include <iostream>

#include <cstdio>

using namespace std;

class Task

{

size\_t size = 10;

double time\_ = 0;

int\* data\_rand = nullptr;

int\* data\_up = nullptr;

int\* data\_Down = nullptr;

int\* data\_half = nullptr;

std::string data\_name[4] = { "data\_rand", "data\_up", "data\_Down", "data\_half" };

int count = 0;

public:

Task()

{

install(data\_rand);

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

data\_rand[i] = rand() % 100 + 1;

install(data\_up);

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

data\_up[i] = i;

install(data\_Down);

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

data\_Down[i] = size - i;

install(data\_half);

for (int i = 0; i < size / 2; ++i)

data\_half[i] = i + 1;

for (int i = size / 2; i < size; ++i)

data\_half[i] = size - i;

sort\_print\_mass\_time(data\_rand);

sort\_print\_mass\_time(data\_up);

sort\_print\_mass\_time(data\_Down);

sort\_print\_mass\_time(data\_half);

}

~Task()

{

deinstall(data\_rand);

deinstall(data\_up);

deinstall(data\_Down);

deinstall(data\_half);

}

void install(int\*& data)

{

data = new int [size];

}

void deinstall(int\* data)

{

if (data)

delete[] data;

}

void sort\_print\_mass\_time(int\* arr)

{

clock\_t start\_time = clock();

shell(arr);

clock\_t end\_time = clock();

time\_ = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

print\_time();

}

void shell(int\* arr)

{

for (int gap = size / 2; gap > 0; gap /= 2)

for (int i = gap; i < size; i++)

for (int j = i; j >= gap && arr[j - gap] > arr[j]; j -= gap)

swap(arr[j], arr[j - gap]);

}

void print\_time() {

std::cout << data\_name[count] << std::endl;

count++;

std::cout << "Size: " << size << std::endl;

std::cout << "Time shell: " << time\_ << " seconds\n" << std::endl;

}

};

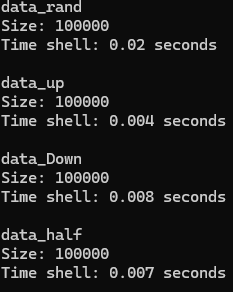
****

Рис.3.1

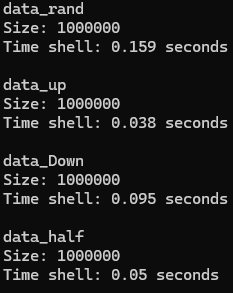
****

Рис.3.2

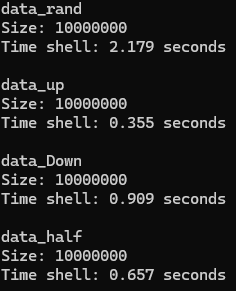
****

Рис.3.3

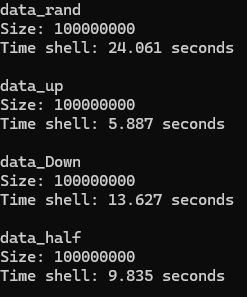
****

Рис.3.4

**Листинг (**qsort**)**

**Main.cpp**

#include "task.h"

int main()

{

Task Tz;

return (0);

}

**task.h**

#pragma once

#include <iostream>

#include <cstdio>

using namespace std;

class Task

{

size\_t size = 10000;

double time\_ = 0;

int\* data\_rand = nullptr;

int\* data\_up = nullptr;

int\* data\_Down = nullptr;

int\* data\_half = nullptr;

std::string data\_name[4] = { "data\_rand", "data\_up", "data\_Down", "data\_half" };

int count = 0;

public:

Task()

{

install(data\_rand);

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

data\_rand[i] = rand() % 100 + 1;

install(data\_up);

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

data\_up[i] = i;

install(data\_Down);

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

data\_Down[i] = size - i;

install(data\_half);

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

data\_half[i] = (i < size / 2) ? (i + 1) : (size - i);

sort\_print\_mass\_time(data\_rand);

sort\_print\_mass\_time(data\_up);

sort\_print\_mass\_time(data\_Down);

sort\_print\_mass\_time(data\_half);

}

~Task()

{

deinstall(data\_rand);

deinstall(data\_up);

deinstall(data\_Down);

deinstall(data\_half);

}

void install(int\*& data)

{

data = new int [size];

}

void deinstall(int\* data)

{

delete[] data;

}

void sort\_print\_mass\_time(int\* arr)

{

clock\_t start\_time = clock();

qs(arr, 0, size - 1);

clock\_t end\_time = clock();

time\_ = static\_cast<double>(end\_time - start\_time) / CLOCKS\_PER\_SEC;

print\_time();

}

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

{

while (left < right)

{

int pivot = items[(left + right) / 2];

int i = left;

int j = right;

do

{

while (items[i] < pivot) i++;

while (pivot < items[j]) j--;

if (i <= j)

{

std::swap(items[i], items[j]);

i++;

j--;

}

} while (i <= j);

// Рекурсивный вызов для двух подмассивов

if (j - left < right - i)

{

qs(items, left, j);

left = i;

}

else

{

qs(items, i, right);

right = j;

}

}

}

void print\_time() {

std::cout << data\_name[count] << std::endl;

count++;

std::cout << "Size: " << size << std::endl;

std::cout << "Time shell: " << time\_ << " seconds\n" << std::endl;

}

};

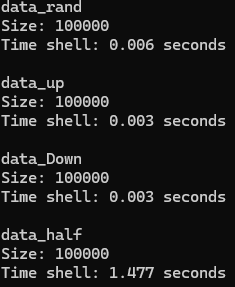


Рис.4.1

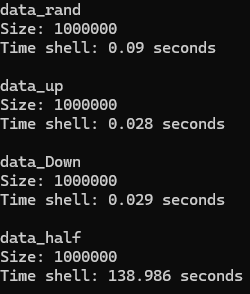
****

Рис.4.2

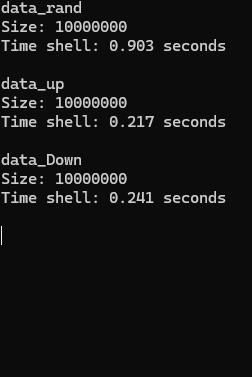
****

Рис.4.3

При превышение 1 000 000 элементов в массиве data\_half ,массив начинает рекурсивно вызвать себя до бесконечности.Так как скорее всего имеет вид:

( O(n \log n) )

**Вывод:**

При выборе алгоритма нужно учитывать, с какими данными будет взаимодействовать т.к. у каждого есть преимущества и недостатки, что мы и наблюдаем при выполнении лабораторной работы.