Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительной техники»

**Отчет**

По лабораторной работе №2

По курсу «Логика и основы алгоритмизации в инженерных задачах»

На тему «Оценка времени выполнения программ»

Выполнили студенты гр.20ВВ4

Макеев Д.И

Демаев А.И

Шалеев Я.А

Проверили:

Юрова О.В

Акифьев И.В.

Пенза, 2021

Цель работы: теоретическая оценка сложности алгоритма и измерение времени выполнения.

**Задание 1:**

1. Порядок сложности приведенной программы: O(n)=n3
2. Написали программу, выполняющую приведенную функцию для матриц разного размера.

**Листинг:**

#include<stdio.h>

#include<stdlib.h>

#include<time.h>

#include<locale.h>

void function(int);

void main(void)

{

setlocale(LC\_ALL, "rus");

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

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

printf\_s("\nМатрица 100\*100: \n");

function(100);

printf\_s("\nМатрица 200\*200: \n");

function(200);

printf\_s("\nМатрица 400\*400: \n");

function(400);

printf\_s("\nМатрица 1000\*1000: \n");

function(1000);

printf\_s("\nМатрица 2000\*2000: \n");

function(2000);

printf\_s("\nМатрица 4000\*4000: \n");

function(4000);

printf\_s("\nМатрица 10000\*10000: \n");

function(10000);

}

void function(intrasmer)

{

int\*\* mas1, \*\* mas2, \*\* mas3;

inti = 0, j = 0, r = 0; intelem\_c;

time\_t start, end,start1, end1;

doublerasn=0, rasn1=0;

start1 = time(NULL);

mas1=(int\*\*)malloc(rasmer \* sizeof(int));

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

{

mas1[i] = (int\*)malloc(rasmer \* sizeof(int));

}

i = 0;

mas2 = (int\*\*)malloc(rasmer \* sizeof(int));

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

{

mas2[i] = (int\*)malloc(rasmer \* sizeof(int));

}

i = 0;

mas3 = (int\*\*)malloc(rasmer \* sizeof(int));

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

{

mas3[i] = (int\*)malloc(rasmer \* sizeof(int));

}

i = 0;

srand(time(NULL));

while (i<rasmer)

{

j = 0;

while (j <rasmer)

{

mas1[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

srand(time(NULL));

i = 0;

while (i<rasmer)

{

j = 0;

while (j <rasmer)

{

mas2[i][j] = rand() % 100 + 1;

j++;

}

i++;

}

//Перемножениематриц:

start = time(NULL);

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

{

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

{

elem\_c = 0;

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

{

elem\_c = elem\_c + mas1[i][r] \* mas2[r][j];

mas3[i][j] = elem\_c;

}

}

}

end = time(NULL);

rasn = difftime(end, start);

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

{

free(mas1[i]);

free(mas2[i]);

free(mas3[i]);

}

free(mas1);

free(mas2);

free(mas3);

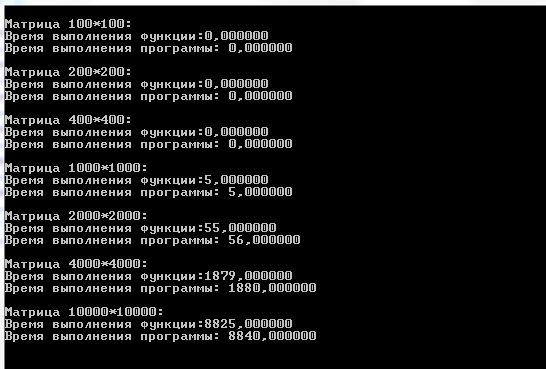
end1 = time(NULL);

rasn1 = difftime(end1, start1);

printf\_s("Времявыполненияфункции: %lf\n", rasn1);

printf\_s("Время выполнения умножения: %lf\n", rasn);

}

1. Пояснение: функция function() имеет один аргумент – размер матрицы. Функция выделяет память для трех матриц заданного размера, две заполняет случайными числами, и выполняет умножение этих матриц, занося результат в третью. Функция измеряет время выполнения всего кода функции и цикла, выполняющего умножение, и выводит данные в консоль.
2. Результат работы:
3. Построили графики зависимости:

1. График теоретической оценки сложности алгоритма:

2. График реальной зависимости:

**Вывод:**время умножения двух матриц изменяется неравномерно, матрицы 100\*100, 200\*200 и 400\*400 умножаются за примерно одинаковое время. При дальнейшем увеличении размера происходит резкое увеличение времени выполнения, время растет намного быстрее, чем размер матриц.

**Задание 2:**

1. Написали программу, выполняющую сортировку массива с помощью алгоритма Шелла, алгоритма быстрой сортировки и стандартной функции qsort().

**Листинг:**

#include<stdio.h>

#include<locale.h>

#include<time.h>

#include<stdlib.h>

#defineN 5000

void shell(int\* items, intcount);

voidqs(int\* items, intleft, intright);

int comp(constint\* i, constint\* j);

voidmain()

{

doublediff=0;

setlocale(LC\_ALL, "rus");

int m[N] = { 0 };

int\* mas;

int i = 0;

//Случайные числа

//Алгоритм Шелла

mas = &m[0];

srand(time(NULL));

for (i; i< N; ++i) m[i] =rand()%100;

mas = &m[0];

diff = clock();

shell(m, N);

diff = (clock() - diff)/1000;

printf\_s("Время сортировки по алгоритму Шелла для случайного массива:%lf \n",diff);

//Быстрая сортировка

mas = &m[0];

for (i=0; i< N; ++i) m[i] = rand() % 100;

diff = clock();

qs(mas, 0, N-1);

diff = (clock() - diff)/1000;

printf\_s("Время быстрой сортировки: %lf\n",diff);

//Стандартная сортировка

mas = &m[0];

for (i = 0; i< N; ++i) m[i] = rand() % 100;

mas = &m[0];

diff = clock();

qsort(mas, N, sizeof(int), (int(\*) (constvoid\*, constvoid\*)) comp);

diff = (clock() - diff)/1000;

printf\_s("Время стандартной сортировки:%lf \n",diff);

printf\_s("\n");

//Возрастающая последовательность:

//алгоритм Шелла:

mas = &m[0];

for (i = 0; i< N; i++) mas[i] = i + 1;

mas = &m[0];

diff = clock();

shell(m, N);

diff = (clock() - diff)/1000;

printf\_s("Время сортировки по алгоритму Шелла для возрастающей последовательноси:%lf \n", diff);

//Быстрая сортировка:

for (i = 0; i< N; i++) m[i] = i + 1;

diff = clock();

qs(mas, 0, N - 1);

diff = (clock() - diff)/1000;

printf\_s("Время быстрой сортировки: %lf\n", diff);

//Стандартная сортировка

mas = &m[0];

for (i = 0; i< N; i++) mas[i] = i + 1;

mas = &m[0];

diff = clock();

qsort(mas, N, sizeof(int), (int(\*) (constvoid\*, constvoid\*)) comp);

diff = (clock() - diff)/1000;

printf\_s("Время стандартной сортировки:%lf \n",diff);

printf\_s("\n");

//Убывающая последовательность:

//Алгоритм Шелла:

mas = &m[0];

for (i = 0; i< N; i++) mas[i] = N - i;

mas = &m[0];

diff = clock();

shell(m, N);

diff = (clock() - diff)/1000;

printf\_s("Время сортировки по алгоритму Шелла для убывающей последовательности:%lf \n", diff);

//Быстрая сортировка:

mas = &m[0];

for (i = 0; i< N; i++) mas[i] = N - i;

mas = &m[0];

diff = clock();

qs(mas, 0, N - 1);

diff = (clock() - diff)/1000;

printf\_s("Время быстрой сортировки: %lf\n", diff);

//Стандартная функция:

mas = &m[0];

for (i = 0; i< N; i++) mas[i] = N - i;

mas = &m[0];

diff = clock();

qsort(mas, N, sizeof(int), (int(\*) (constvoid\*, constvoid\*)) comp);

diff = (clock() - diff)/1000;

printf\_s("Время стандартной сортировки:%lf \n",diff);

printf\_s("\n");

//Половина массива возрастающая, половина - убывающая

//Алгоритм Шелла

mas = &m[0];

for (i = 0; i< N/2; i++) mas[i] = i+1;

for (i = N / 2; i< N; i++) mas[i] = N - i;

mas = &m[0];

diff = clock();

shell(m, N);

diff = (clock() - diff)/1000;

printf\_s("Время сортировки по алгоритму Шелла для возрастающей и убывающей последовательности:%lf \n",diff);

//Быстрая сортировка:

mas = &m[0];

for (i = 0; i< N / 2; i++) mas[i] = i + 1;

for (i = N / 2; i< N; i++) mas[i] = N - i;

mas = &m[0];

diff = clock();

qs(mas, 0, N - 1);

diff = (clock() - diff)/1000;

printf\_s("Время быстрой сортировки: %lf\n", diff);

//Стандартная сортировка:

mas = &m[0];

for (i = 0; i< N / 2; i++) mas[i] = i + 1;

for (i = N / 2; i< N; i++) mas[i] = N - i;

mas = &m[0];

diff = clock();

qsort(mas, N, sizeof(int), (int(\*) (constvoid\*, constvoid\*)) comp);

diff = clock() - diff;

diff = (clock() - diff)/1000;;

printf\_s("Время стандартной сортировки:%lf \n",diff);

getchar();

}

void shell(int \*items, intcount)

{

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;

}

}

}

voidqs(int\* items, intleft, intright)

{

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);

}

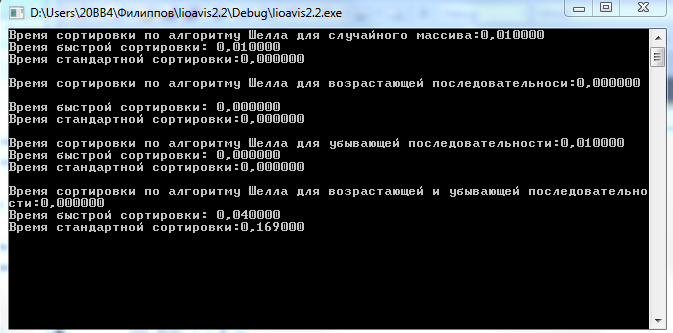
int comp(constint\* i, constint\* j)

{

return \*i - \*j;

}

1. Пояснения: в программе имеются функции: shell() для выполненияалгоритма Шелла, qs() для выполнения алгоритма быстрой сортировки. Функция comp() нужна для работы стандартной функции сортировки qsort(). Программа создает массив размера N, и передает указатель на него в функции, при этом перед каждым вызовом массив заполняется данными в соответствии с заданием.
2. Результат:



**Вывод:** наибольшее время сортировки - у массива, половина которого возрастает, а половина - убывает. Для случайного массива наименьшее время заняла сортировка стандартной функцией. Возрастающая последовательность затратила меньше всего времени (так как она уже отсортирована). Для смешанной последовательности(возрастающей и убывающей) самым быстрым оказался алгоритм Шелла.