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»

Выполнил студент группы 19ВВ2:

Махнаев Н.С.

Приняли:

Митрохин М. А.

Юрова О. В.

Пенза 2020

**Цель работы:**

Научиться замерять время работы программ и отдельных алгоритмов и применять их на практике для нахождения более быстрых методов решения задач.

**Лабораторное задание:**

Дана программа, вычисляющая произведение двух матриц:

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

int main(void)

{

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

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

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

int i = 0, j = 0, r;

int a[200][200], b[200][200], c[200][200], elem\_c;

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

while (i < 200)

{

while (j < 200)

{

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 < 200)

{

while (j < 200)

{

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

j++;

}

i++;

}

for (i = 0; i < 200; i++)

{

for (j = 0; j < 200; j++)

{

elem\_c = 0;

for (r = 0; r < 200; r++)

{

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

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

}

}

}

return(0);

}

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.

Даны реализации алгоритмов сортировки Шелла и быстрой сортировки:

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 1:**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <locale.h>

int main(void)

{

setlocale(LC\_ALL, "Rus");

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

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

clock\_t start, end;

int i = 0, j = 0, r;

int a[100][100], b[100][100], c[100][100], elem\_c;

srand(time(NULL));

while (i < 100)

{

while (j < 100)

{

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

j++;

}

i++;

}

srand(time(NULL));

while (i < 100)

{

while (j < 100)

{

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

j++;

}

i++;

}

start = clock();

for (i = 0; i < 100; i++)

{

for (j = 0; j < 100; j++)

{

elem\_c = 0;

for (r = 0; r < 100; r++)

{

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

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

}

}

}

end = clock();

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

printf("\n Время выполнения: %f\n", seconds);

return(0);

}

**Листинг 2:**

#include <stdio.h>

#include <malloc.h>

#include <random>

#include <time.h>

#include <ctime>

#include <locale.h>

int compare(const void\* x1, const void\* x2) // функция сравнения элементов массива

{

return (\*(int\*)x1 - \*(int\*)x2); // если результат вычитания равен 0, то числа равны, < 0: x1 < x2; > 0: x1 > x2

}

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 main() {

setlocale(LC\_ALL, "Rus");

clock\_t time1, time2;

int n;

printf("Случайная последовательность\n ");

printf("Введите количество элементов массива: ");

scanf\_s("%d", &n);

int\* arr = new int[n];

srand(time(NULL));

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

arr[i] = ((rand() % 2001) - 1000);

}

printf("\n");

time1 = clock();

qs(arr, 0, n-1);

time2 = clock();

double seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировкой быстрой: %f\n", seconds);

srand(time(NULL));

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

arr[i] = ((rand() % 2001) - 1000);

}

time1 = clock();

shell(arr, n);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировки Шелла: %f\n", seconds);

printf("\nВозрастающая последовательность\n ");

srand(time(NULL));

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

arr[i] = i;

}

time1 = clock();

qs(arr, 0, n - 1);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировкой быстрой: %f\n", seconds);

time1 = clock();

shell(arr, n);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировки Шелла: %f\n", seconds);

printf("\nУбывающая последовательность\n ");

int count = n;

srand(time(NULL));

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

arr[i] = count;

count--;

}

time1 = clock();

qs(arr, 0, n - 1);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировкой быстрой: %f", seconds);

count = n;

srand(time(NULL));

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

arr[i] = count;

count--;

}

time1 = clock();

shell(arr, n);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировки Шелла: %f\n", seconds);

printf("Возрастающая и убывающая\n");

count = n;

srand(time(NULL));

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

if (i < n / 2) {

arr[i] = i;

}

else {

arr[i] = count;

count--;

}

}

time1 = clock();

qs(arr, 0, n - 1);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировкой быстрой: %f\n", seconds);

count = n;

srand(time(NULL));

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

if (i < n / 2) {

arr[i] = i;

}

else {

arr[i] = count;

count--;

}

}

time1 = clock();

shell(arr, n);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Время сортировки Шелла: %f\n", seconds);

printf("\nСтандартная qsort\n");

srand(time(NULL));

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

arr[i] = ((rand() % 2001) - 1000);

}

time1 = clock();

qsort(arr, n, sizeof(int), compare);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Случайная последовательность: %f\n", seconds);

srand(time(NULL));

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

arr[i] = i;

}

time1 = clock();

qsort(arr, n, sizeof(int), compare);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Возрастающая: %f\n", seconds);

srand(time(NULL));

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

arr[i] = count;

count--;

}

time1 = clock();

qsort(arr, n, sizeof(int), compare);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Убывающая: %f\n", seconds);

count = n;

srand(time(NULL));

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

if (i < n / 2) {

arr[i] = i;

}

else {

arr[i] = count;

count--;

}

}

time1 = clock();

qsort(arr, n, sizeof(int), compare);

time2 = clock();

seconds = (double)(time2 - time1) / CLOCKS\_PER\_SEC;

printf("\n Наполовину: %f\n", seconds);

return (0);

}

**Результаты работы:**

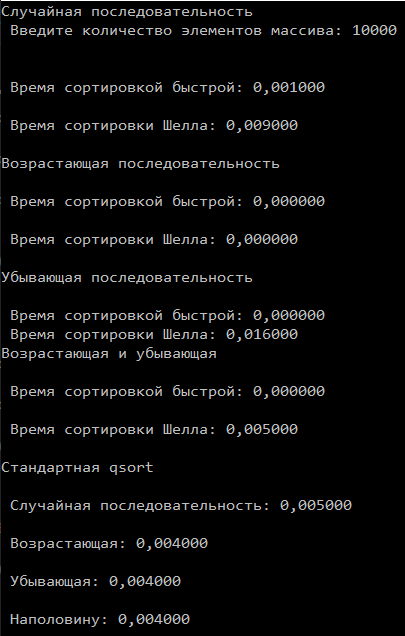
Задание 1:

1. Имеем два вложенных цикла: сложность первого O(n^3), второго O(n^2). Так как они вызываются в main поочерёдно, то их сложность складывается: O(n^3) + O(n^2) = O(n^3).
2. 100 элементов: 

200 элементов: 

Задание 2:

Вводим в программу 10000 элементов массива



**Вывод:**

Сортировка Шелла лучше остальных справилась с возрастающим массивом элементов, но с остальными массивами показала средние результаты.

Сортировка qs, данная в задании, работает быстро со всеми типами массивов, но ее эффективность падает значительно при обработке пилообразного массива

Сортировка qsort из стандартной библиотеки справилась со всеми типами массивов за приемлемое время. Эта сортировка является самой универсальной.