Министерство образования Российской Федерации   
Пензенский государственный университет   
Кафедра «Вычислительная техника»

ОТЧЕТ

По лабораторной №2   
На тему: Оценка времени выполнения программ  
По дисциплине: «Логика и основы алгоритмизации в инженерных задачах»

Выполнили студенты группы 20ВВ2:

Борунов Д.А

Сурков М.С

Приняли:

Митрохин М.А. Юрова О.В.

**Пенза 2021г**

**Задание 1 Оценить время выполнения программы и кода, выполняющего перемножениематриц, используя функции библиотеки time.h для матриц размерами от 100, 200, 400, 1000, 2000, 4000, 10000.**

Для выполнения задания создали 3 динамических массива. В ходе выполнения программы размер массива меняется 6 раз. Вывод времени выполнения умножения матриц осуществляется в консоль и в файл в виде списка времени выполнения.

**Листинг**

int main(void)

{

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

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

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

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

int i = 0, j = 0, r, n, m, elem\_c, g\_mass=0;

double mass[6];

for (m = 0; m < 6; m++) {

printf("Size n\*n, n = ");

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

start = clock();

printf("Col = %d\n", n);

printf("Row = %d\n", n);

int\* a = (int\*)malloc(n \* n \* sizeof(int));

int\* b = (int\*)malloc(n \* n \* sizeof(int));

int\* c = (int\*)malloc(n \* n \* sizeof(int));

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

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

{

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

{

\*(a + i \* n + j) = rand() % 10;

}

}

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

{

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

{

\*(b + i \* n + j) = rand() % 10;

}

}

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

{

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

{

elem\_c = 0;

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

{

elem\_c = elem\_c + \*(a + i \* n + r) \* \*(b + r \* n + j);

\*(c + i \* n + j) = elem\_c;

}

}

}

free(a);

free(b);

free(c);

end = clock();

mass[g\_mass] = ((double)end - start) / ((double)CLOCKS\_PER\_SEC);

printf("\n %f second(s)\n", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

g\_mass++;

}

FILE\* file;

file = fopen("data.txt", "w");

for (i = 0; i < 6; i++)

{

fprintf(file, "%f", mass[i]);

fprintf(file, "\n");

}

return(0);

}

График зависимости времени выполнения алгоритма переумножения матриц от количества элементов в них:

В ходе выполнения задания, экспериментальным путем доказали, что сложность программы O(n^3)

**Задание 2**

**Сортировка Шелла (в процессе выполнения задания исходный код менялся)**

**Листинг**

#include <stdio.h>

#include <conio.h>

#include <stdlib.h>

#include <time.h>

#include <malloc.h>

#include <windows.h>

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 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

}

int main()

{

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

srand(time(NULL));

int\* mass;

int i, n = 100000;

mass = (int\*)malloc(n \* sizeof(int));

clock\_t start, end;

printf("\nСортировка на случайном наборе значений массива.\n\n");

printf("Алгоритм Шелла:");

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

{

mass[i] = rand() % 100;

}

start = clock();

shell(mass, n);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nАлгоритм Быстрой сортировки: ");

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

{

mass[i] = rand() % 10;

}

start = clock();

qs(mass, 0, n - 1);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nСтандартная функция сортировки qsort: ");

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

{

mass[i] = rand() % 10;

}

start = clock();

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

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

////////////////////////////////////////////////////////////////////////////////////////////////////////////

printf("\n\nСортировка массивa, представляющего собой возрастающую последовательность чисел.\n\n");

printf("Алгоритм Шелла:");

mass[0] = rand() % 10;

for (i = 1; i < n; i++)

{

mass[0]++;

mass[i] = mass[0];

}

start = clock();

shell(mass, n);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nАлгоритм Быстрой сортировки: ");

mass[0] = rand() % 10;

for (i = 1; i < n; i++)

{

mass[0]++;

mass[i] = mass[0];

}

start = clock();

qs(mass, 0, n - 1);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nСтандартная функция сортировки qsort: ");

mass[0] = rand() % 10;

for (i = 1; i < n; i++)

{

mass[0]++;

mass[i] = mass[0];

}

start = clock();

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

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

////////////////////////////////////////////////////////////////////////////////////////////////

printf("\n\nСортировка массивa, представляющего собой убывающую последовательность чисел.\n\n");

printf("Алгоритм Шелла:");

mass[0] = rand() % 100;

for (i = 1; i < n; i++)

{

mass[0]--;

mass[i] = mass[0];

}

start = clock();

shell(mass, n);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nАлгоритм Быстрой сортировки: ");

mass[0] = rand() % 100;

for (i = 1; i < n; i++)

{

mass[0]--;

mass[i] = mass[0];

}

start = clock();

qs(mass, 0, n - 1);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nСтандартная функция сортировки qsort: ");

mass[0] = rand() % 100;

for (i = 1; i < n; i++)

{

mass[0]--;

mass[i] = mass[0];

}

start = clock();

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

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

////////////////////////////////////////////////////////////////////////////////////////////////////////////

printf("\n\nСортировка массивa, одна половина которого представляет\n собой возрастающую последовательность чисел, а вторая, – убывающую.\n\n");

printf("Алгоритм Шелла:");

mass[0] = rand() % 10;

for (i = 1; i < n / 2; i++)

{

mass[0]++;

mass[i] = mass[0];

}

for (i = n / 2; i < n; i++)

{

mass[0]--;

mass[i] = mass[0];

}

start = clock();

shell(mass, n);

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nСтандартная функция сортировки qsort: ");

mass[0] = rand() % 100;

for (i = 1; i < n; i++)

{

mass[0]--;

mass[i] = mass[0];

}

start = clock();

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

end = clock();

printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\nАлгоритм Быстрой сортировки: ");

mass[0] = rand() % 100;

for (i = 1; i < n / 2; i++)

{

mass[0]++;

mass[i] = mass[0];

}

for (i = n / 2; i < n; i++)

{

mass[0]--;

mass[i] = mass[0];

}

start = clock();

qs(mass, 0, n-1);

end = clock();

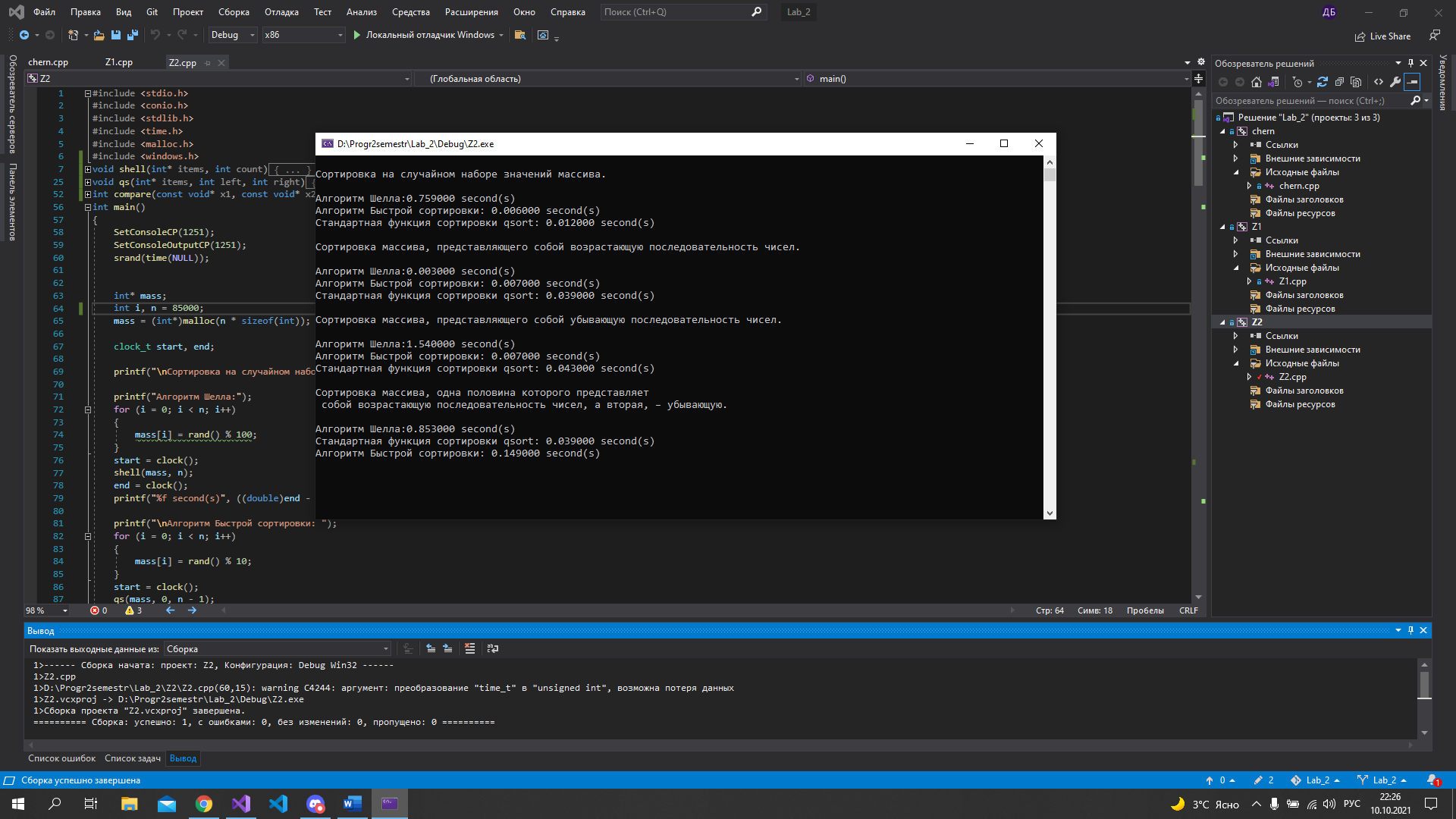
printf("%f second(s)", ((double)end - start) / ((double)CLOCKS\_PER\_SEC));

printf("\n");

\_getch();

}

**Сравнение времени работы различных алгоритмов для 85000 элементов массива**



**Вывод по заданию №2:** в ходе выполнения задания сделали следующие выводы:

1. Для сортировки случайного набора значений массива самым эффективным оказался алгоритм Быстрой сортировки, самым медленным – Алгоритм Шелла

2. Для сортировки массива, представляющего собой возрастающую последовательность чисел самым эффективным оказался алгоритм Быстрой сортировки

3. Для сортировки массивa, представляющего собой убывающую последовательность чисел самым эффективным оказался алгоритм Быстрой сортировки, самым медленным – Алгоритм Шелла

4. Для сортировки массивa, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую самым эффективным оказался алгоритм Быстрой сортировки, самым медленным – Алгоритм Шелла

**Вывод:** в ходе выполнения лабораторной работы выявили закономерности во времени работы каждого из данных алгоритмах, на разных наборах данных.