 Министерство образования Российской Федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в ИЗ»

на тему «Оценка времени выполнения программ»

Выполнили:

студенты группы 22ВВВ2

Попов Д. А.

Приняли:

доцент, профессор Митрохин М.А.

к.э.н., доцент Акифьев И.В.

Пенза 2023

**Название**

Оценка времени выполнения программ

**Цель работы**

Оценить время выполнения программы

**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.

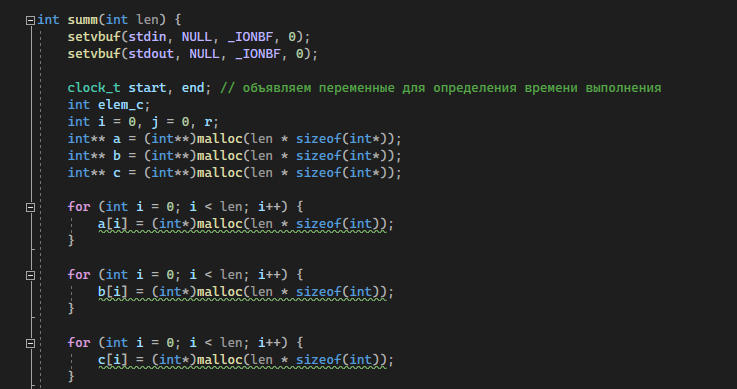
**Задание 2**:

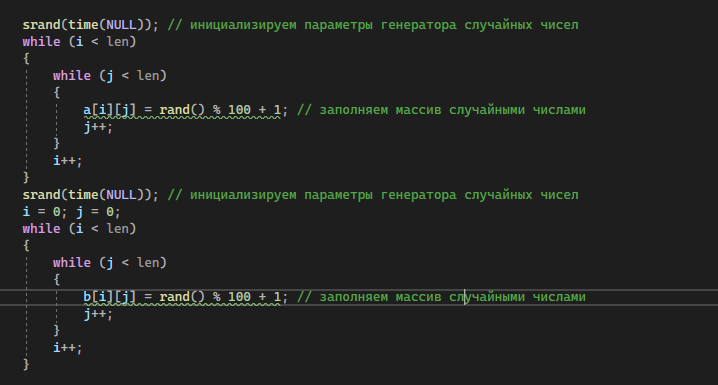
1. Оценить время работы каждого из реализованных алгоритмов на случайном наборе значений массива.
2. Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой возрастающую последовательность чисел.
3. Оценить время работы каждого из реализованных алгоритмов на массиве, 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.

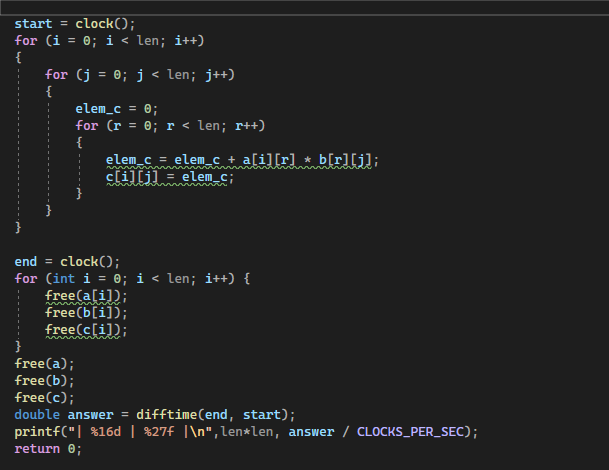
**Ход Работы**

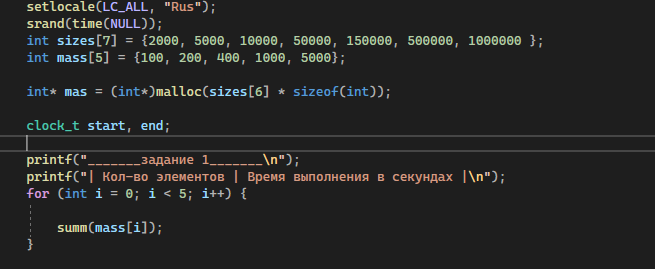
Выполняем 1 задание:

Создаем Динамические массивы

****

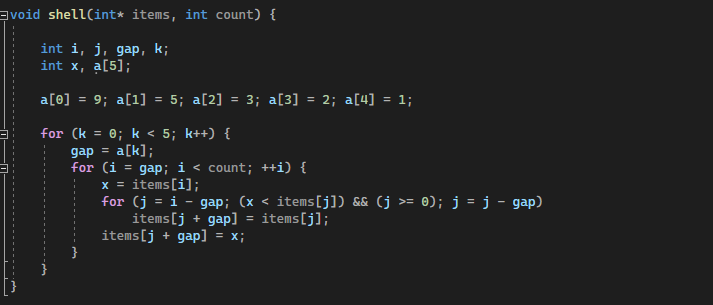
Заполняем массивы случайными числами

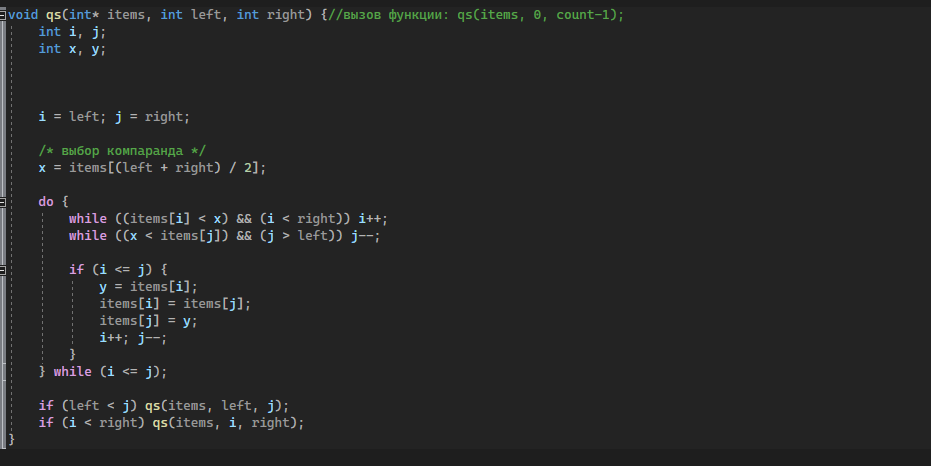
Вычисляем время перемножения матриц и выводим результат

В главной функции задаем массив с количеством элементов и массивы для следующего задания

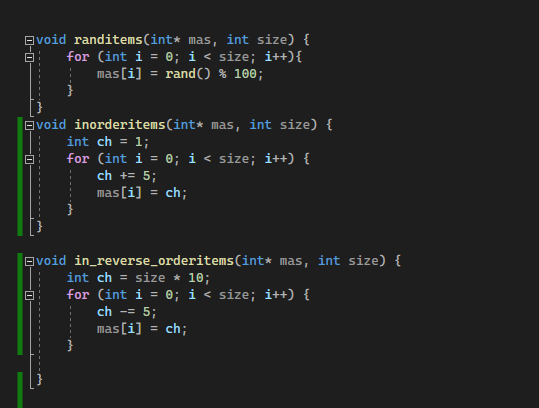
Выполняем 2 задание

Реализовали функции для сортировок

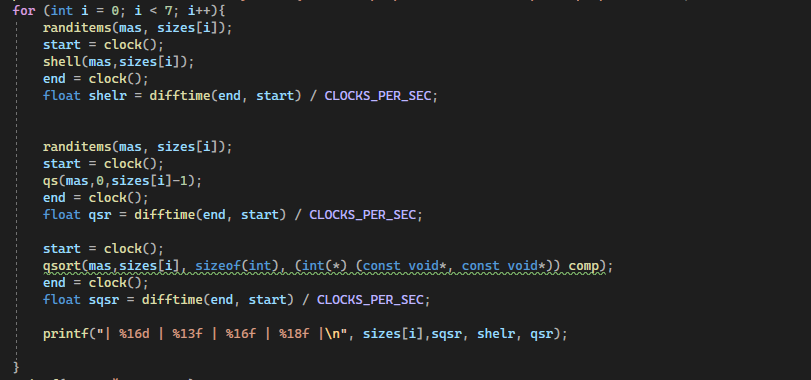




Создаем функции для задания массивов в разных порядках



Считаем и выводим результат для каждой функции и сортировки



**Листинг**

#include <stdio.h>

#include <time.h>

#include <malloc.h>

#include <random>

#include <locale.h>

int comp(const int\* i, const int\* j)//нужна для qsort

{

return \*i - \*j;

}

int summ(int len) {

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

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

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

int elem\_c;

int i = 0, j = 0, r;

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

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

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

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

a[i] = (int\*)malloc(len \* sizeof(int));

}

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

b[i] = (int\*)malloc(len \* sizeof(int));

}

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

c[i] = (int\*)malloc(len \* sizeof(int));

}

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

while (i < len)

{

while (j < len)

{

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 < len)

{

while (j < len)

{

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

j++;

}

i++;

}

start = clock();

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

{

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

{

elem\_c = 0;

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

{

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

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

}

}

}

end = clock();

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

double answer = difftime(end, start);

printf("| %16d | %27f |\n",len\*len, answer / CLOCKS\_PER\_SEC);

return 0;

}

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);

}

void randitems(int\* mas, int size) {

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

mas[i] = rand() % 100;

}

}

void inorderitems(int\* mas, int size) {

int ch = 1;

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

ch += 5;

mas[i] = ch;

}

}

void in\_reverse\_orderitems(int\* mas, int size) {

int ch = size \* 10;

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

ch -= 5;

mas[i] = ch;

}

}

void direct\_and\_reverse(int\* mas, int size) {

int ch1 = 1, ch2 = (size / 2) \* 10;

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

if (i < size) {

mas[i] = ch1;

ch1 += 5;

}

else{

mas[i] = ch2;

ch2 -= 5;

}

}

}

int main() {

setlocale(LC\_ALL, "Rus");

srand(time(NULL));

int sizes[7] = {2000, 5000, 10000, 50000, 150000, 500000, 1000000 };

int mass[5] = {100, 200, 500, 1000, 2000};

int\* mas = (int\*)malloc(sizes[6] \* sizeof(int));

clock\_t start, end;

printf("\_\_\_\_\_\_\_задание 1\_\_\_\_\_\_\_\n");

printf("| Кол-во элементов | Время выполнения в секундах |\n");

for (int i = 0; i < 5; i++) {

summ(mass[i]);

}

printf("\_\_\_\_\_\_\_задание 2\_\_\_\_\_\_\_\n");

printf("Случайный порядок\n");

printf("| Кол-во элементов | Функция Qsort | Сортировка Шелла | Быстрая сортировка |\n");

for (int i = 0; i < 7; i++){

randitems(mas, sizes[i]);

start = clock();

shell(mas,sizes[i]);

end = clock();

float shelr = difftime(end, start) / CLOCKS\_PER\_SEC;

randitems(mas, sizes[i]);

start = clock();

qs(mas,0,sizes[i]-1);

end = clock();

float qsr = difftime(end, start) / CLOCKS\_PER\_SEC;

start = clock();

qsort(mas,sizes[i], sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

float sqsr = difftime(end, start) / CLOCKS\_PER\_SEC;

printf("| %16d | %13f | %16f | %18f |\n", sizes[i],sqsr, shelr, qsr);

}

printf("Прямой порядок");

printf("\n| Кол-во элементов | Функция Qsort | Сортировка Шелла | Быстрая сортировка |\n");

for (int i = 0; i < 7; i++) {

inorderitems(mas, sizes[i]);

start = clock();

shell(mas, sizes[i]);

end = clock();

float shelr = difftime(end, start) / CLOCKS\_PER\_SEC;

inorderitems(mas, sizes[i]);

start = clock();

qs(mas, 0, sizes[i] - 1);

end = clock();

float qsr = difftime(end, start) / CLOCKS\_PER\_SEC;

inorderitems(mas, sizes[i]);

start = clock();

qsort(mas, sizes[i], sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

float sqsr = difftime(end, start) / CLOCKS\_PER\_SEC;

printf("| %16d | %13f | %16f | %18f |\n", sizes[i], sqsr, shelr, qsr);

}

printf("Обратный порядок");

printf("\n| Кол-во элементов | Функция Qsort | Сортировка Шелла | Быстрая сортировка |\n");

for (int i = 0; i < 7; i++) {

in\_reverse\_orderitems(mas, sizes[i]);

start = clock();

shell(mas, sizes[i]);

end = clock();

float shelr = difftime(end, start) / CLOCKS\_PER\_SEC;

in\_reverse\_orderitems(mas, sizes[i]);

start = clock();

qs(mas, 0, sizes[i] - 1);

end = clock();

float qsr = difftime(end, start) / CLOCKS\_PER\_SEC;

in\_reverse\_orderitems(mas, sizes[i]);

start = clock();

qsort(mas, sizes[i], sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

float sqsr = difftime(end, start) / CLOCKS\_PER\_SEC;

printf("| %16d | %13f | %16f | %18f |\n", sizes[i], sqsr, shelr, qsr);

}

printf("Обратный и прямой порядок");

printf("\n| Кол-во элементов | Функция Qsort | Сортировка Шелла | Быстрая сортировка |\n");

for (int i = 0; i < 7; i++) {

direct\_and\_reverse(mas, sizes[i]);

start = clock();

shell(mas, sizes[i]);

end = clock();

float shelr = difftime(end, start) / CLOCKS\_PER\_SEC;

direct\_and\_reverse(mas, sizes[i]);

start = clock();

qs(mas, 0, sizes[i] - 1);

end = clock();

float qsr = difftime(end, start) / CLOCKS\_PER\_SEC;

direct\_and\_reverse(mas, sizes[i]);

start = clock();

qsort(mas, sizes[i], sizeof(int), (int(\*) (const void\*, const void\*)) comp);

end = clock();

float sqsr = difftime(end, start) / CLOCKS\_PER\_SEC;

printf("| %16d | %13f | %16f | %18f |\n", sizes[i], sqsr, shelr, qsr);

}

return 0;

}

**Результат работы программы**

Результат работы программы показан на рисунках 1, 2 и 3.

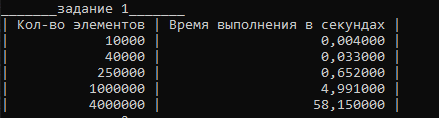


Рис. 1

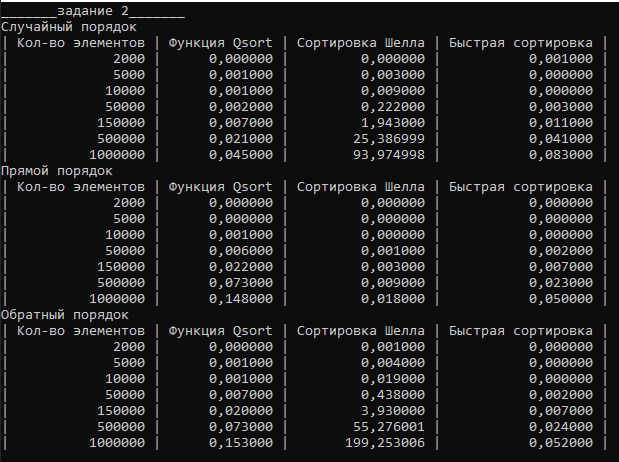


Рис.2

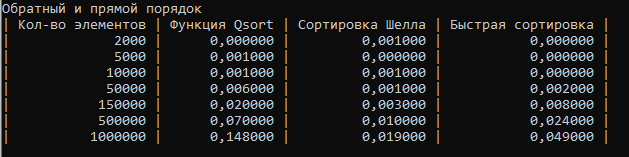


Рис.3

**Вывод:**

В ходе выполнения лабораторной работы мы выявили сильные и слабые стороны типов сортировок. Сортировку Шелла можно использовать только в частично отсортированном массиве, в остальных случаях нужно воспользоваться Быстрой сортировкой. Существенного отличия в скорости встроенной функции сортировки QSort от написанной самостоятельно не выявлено.

Ссылка на репозиторий: https://github.com/Roc25/LogLab2.git