МИНИСТЕРВСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ

РОССИЙСКОЙ ФЕДЕРАЦИИ

ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Кафедра «Вычислительная техника»

**ОТЧЕТ**

по лабораторной работе №2

по курсу «Логика и ОА в ИЗ»

на тему «Оценка времени выполнения программ»

Выполнили:

Студенты группы 23ВВВ2

Севостьянов Олег

Кандаков Андрей

Принял:

Митрохин М.А.

Юрова О.В.

Пенза 2024

**Цель работы**

Повторение пройденного материала, связанного с библиот.

**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.

**Пояснение к коду**

Задание 1:

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

setvbuf(stdout, NULL, \_IONBF, 0) - Устанавливается буферизация для потоков ввода и вывода на небуферизированный режим. Это означает, что все данные будут сразу записываться или считываться, минуя буфер.

clock\_t start, end - Объявление переменных для хранения времени начала и конца выполнения программы (не используется в данном коде, но могла бы быть использована для замера времени выполнения).

int i = 0, j = 0, r - Объявляются индексы цикла `i`, `j`, и `r`.

int a[200][200], b[200][200], c[200][200], elem\_c - Объявляются три матрицы `a`, `b`, и `c` размером 200x200.

- `elem\_c` — переменная для временного хранения суммы произведений при умножении матриц.

long long op\_count = 0; // счётчик операций - Объявление переменной `op\_count` типа `long long` для подсчета количества операций.

while (i < 200)

{

j = 0;

while (j < 200)

{

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

j++;

op\_count++; // одна операция присвоения

}

i++;

}

- Этот цикл заполняет матрицу `a` случайными числами от 1 до 100. Каждое присваивание увеличивает счетчик операций `op\_count` на 1.

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;

op\_count += 3; // одна операция сложения, умножения и присвоения

}

}

}

- Здесь происходит умножение матриц `a` и `b`. Внутренний цикл по переменной `r` вычисляет сумму произведений элементов строк матрицы `a` и столбцов матрицы `b`, результат записывается в `c[i][j]`.

- Счетчик операций `op\_count` увеличивается на 3 для каждой итерации внутреннего цикла: за одно сложение, одно умножение и одно присвоение.

Задание 2:

**1. Функция shell()**

Реализует сортировку Шелла, которая является усовершенствованной сортировкой вставками.

void shell(int\* items, int count)

{

int i, j, gap, k;

int x, a[5];

*// Шаги для сортировки Шелла*

a[0] = 9; a[1] = 5; a[2] = 3; a[3] = 2; a[4] = 1;

* Переменные i, j, gap и k используются для управления циклами и шагом сортировки. Массив a[5] хранит последовательность шагов для сортировки Шелла.
* Эти шаги задают интервалы между сравниваемыми элементами, начиная с больших значений (например, 9) и постепенно уменьшая их до 1.

for (k = 0; k < 5; k++) {

gap = a[k];

for (i = gap; i < count; ++i) {

x = items[i];

for (j = i - gap; (j >= 0) && (items[j] > x); j -= gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

* Внешний цикл проходит по каждому шагу из массива a[].
* Внутренний цикл сортирует элементы на расстоянии gap. Для каждого элемента на позиции i, происходит сортировка вставками, но через шаг gap. Элементы сравниваются и перемещаются в правильные позиции.
* Таким образом, на каждом этапе происходит частичная сортировка с крупными интервалами, и по мере уменьшения интервалов до 1 происходит окончательная сортировка.

**2. Функция qs()**

Реализует ручную версию быстрой сортировки (quicksort).

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

{

int i, j;

int x, y;

* Объявляются переменные для указателей i и j, которые отслеживают начало и конец текущего подмассива, а также переменные x (опорный элемент) и y (для обмена значениями).

i = left;

j = right;

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

* Инициализация индексов i и j для обхода массива с обоих концов. Опорный элемент x выбирается как средний элемент массива.

do {

while ((items[i] < x) && (i < right)) i++;

while ((x < items[j]) && (j > left)) j--;

* Эти циклы ищут элементы, которые должны быть перемещены в правильные места относительно опорного элемента. i увеличивается, пока не найдёт элемент, который больше или равен x, и j уменьшается, пока не найдёт элемент, который меньше или равен x.

if (i <= j) {

y = items[i];

items[i] = items[j];

items[j] = y;

i++;

j--;

}

} while (i <= j);

* Если i <= j, элементы на позициях i и j меняются местами. Затем i увеличивается, а j уменьшается, и процесс повторяется, пока указатели не пересекутся.

if (left < j) qs(items, left, j);

if (i < right) qs(items, i, right);

}

* После того как массив разделён на две части, рекурсивно вызывается функция для каждой части (левая и правая) до тех пор, пока массив не будет полностью отсортирован.

**3. Функция compare()**

Эта функция используется стандартной функцией qsort() для сравнения двух элементов массива.

int compare(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

* Преобразует указатели a и b в целые числа и возвращает разницу между ними. Это стандартный метод сравнения для сортировки целых чисел: если результат отрицательный, элемент a меньше b; если положительный, наоборот.

**4. Функция main()**

const int count = 10000; *// размер массива*

int items1[10000], items2[10000], items3[10000];

* Определяются размер массива count и три массива items1, items2, items3, каждый из которых будет сортироваться разными методами. Все массивы имеют 10 000 элементов.

clock\_t start, end;

double shell\_time, quicksort\_time, qsort\_time;

* Переменные для хранения времени выполнения сортировок: start и end для отсчёта времени, shell\_time, quicksort\_time и qsort\_time для хранения времени выполнения каждой сортировки.

*// Генерация случайного массива*

srand(time(NULL));

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

items1[i] = rand() % 10000;

items2[i] = items1[i];

items3[i] = items1[i];

}

* Используется функция srand() для инициализации генератора случайных чисел. Цикл заполняет массивы случайными числами от 0 до 9999. Массивы items1, items2 и items3 инициализируются одинаковыми значениями, чтобы результаты всех сортировок были корректно сравнимы.

*// Сортировка Шелла*

start = clock();

shell(items1, count);

end = clock();

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

* Начало измерения времени с помощью функции clock(), вызов сортировки Шелла, затем измерение времени окончания сортировки. Время выполнения рассчитывается как разница времени начала и конца, переведённая в секунды.

*// Ручная быстрая сортировка*

start = clock();

qs(items2, 0, count - 1);

end = clock();

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

* То же самое выполняется для ручной реализации быстрой сортировки. Время выполнения сохраняется в переменной quicksort\_time.

*// Стандартная qsort*

start = clock();

qsort(items3, count, sizeof(int), compare);

end = clock();

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

* Вызов стандартной функции сортировки qsort() для третьего массива с использованием функции сравнения compare(). Время выполнения сохраняется в переменной qsort\_time.

*// Вывод результатов*

printf("Шелл-сортировка: %f секунд\n", shell\_time);

printf("Быстрая сортировка: %f секунд\n", quicksort\_time);

printf("Сортировка qsort: %f секунд\n", qsort\_time);

* Вывод времени выполнения каждой сортировки в секундах.

Задание 2.3/5

for (int r = 0; r < repeat; r++) {

start = clock();

shell(items1, count);

end = clock();

shell\_time += ((double)(end - start)) / CLOCKS\_PER\_SEC;

* Цикл for выполняет сортировки repeat раз. Сначала измеряется время выполнения сортировки Шелла, которое добавляется к shell\_time.

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

items1[i] = i;

items2[i] = i;

items3[i] = i;

}

* После каждой сортировки массивы восстанавливаются до исходного состояния для следующей сортировки.

start = clock();

qs(items2, 0, count - 1);

end = clock();

quicksort\_time += ((double)(end - start)) / CLOCKS\_PER\_SEC;

* Измеряется время выполнения быстрой сортировки и добавляется к quicksort\_time.

start = clock();

qsort(items3, count, sizeof(int), compare);

end = clock();

qsort\_time += ((double)(end - start)) / CLOCKS\_PER\_SEC;

}

* Измеряется время выполнения стандартной функции qsort и добавляется к qsort\_time.

shell\_time /= repeat;

quicksort\_time /= repeat;

qsort\_time /= repeat;

Задание 2.4/5

**Результат работы программы**

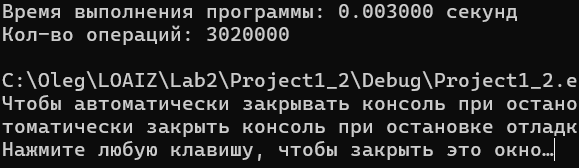


Рисунок 1 – результат работы программы (100х100)

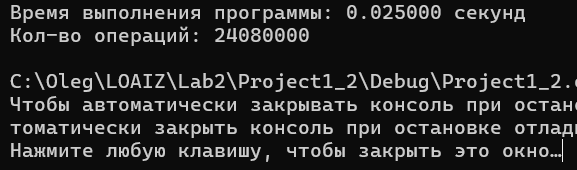


Рисунок 2 – результат работы программы (200х200)

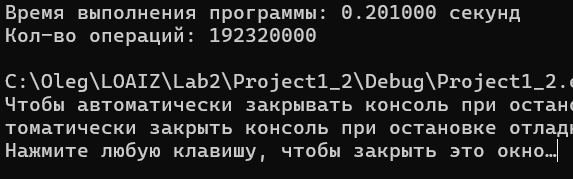


Рисунок 3 – результат работы программы (400x400)

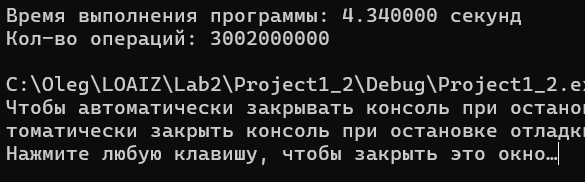
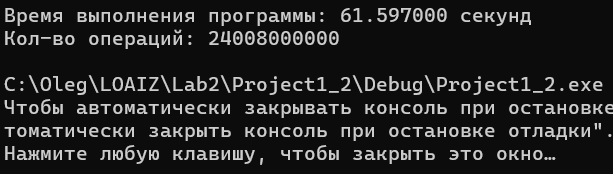


Рисунок 4 – результат работы программы (1000x1000)

****Рисунок 5 – результат работы программы (2000х2000)

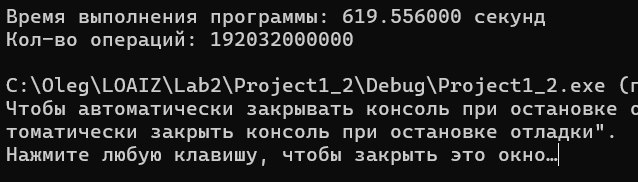


Рисунок 6 – результат работы программы (4000х4000)

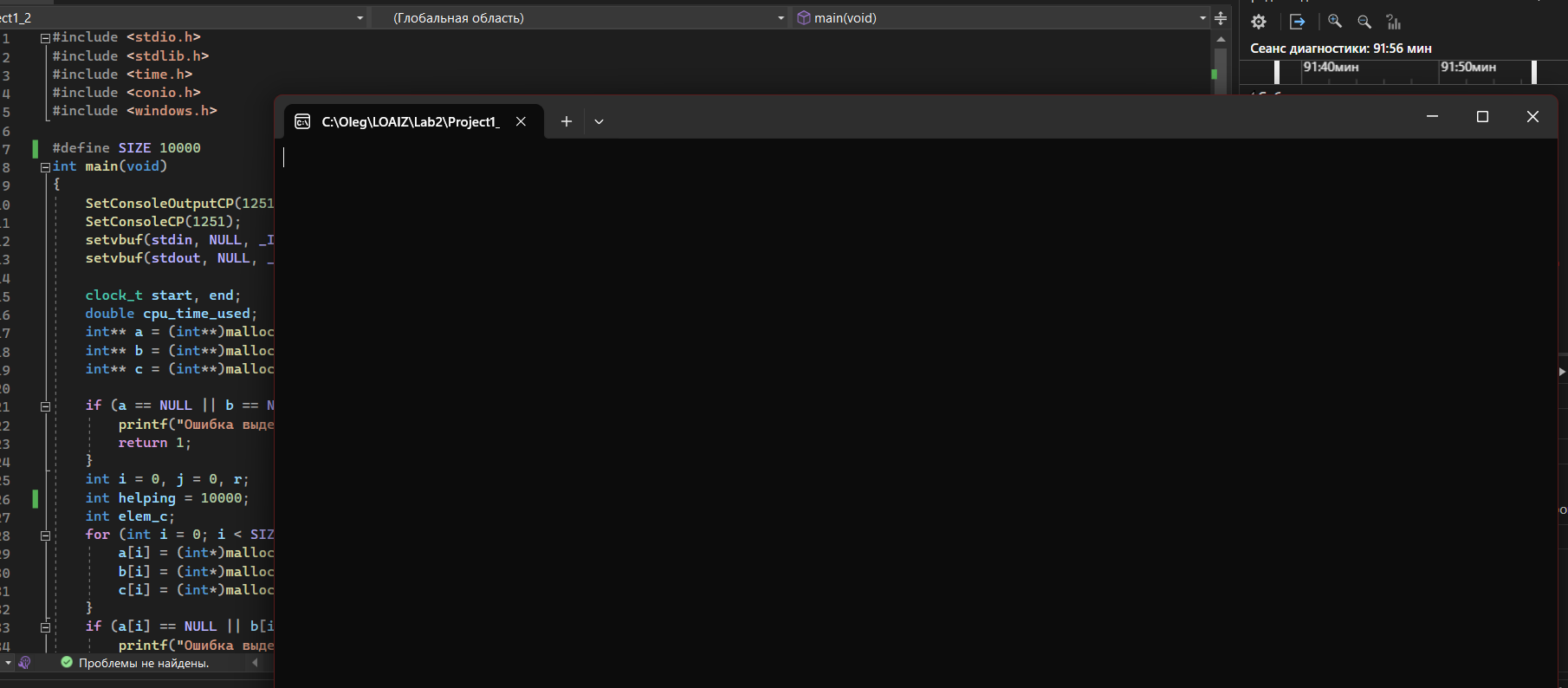
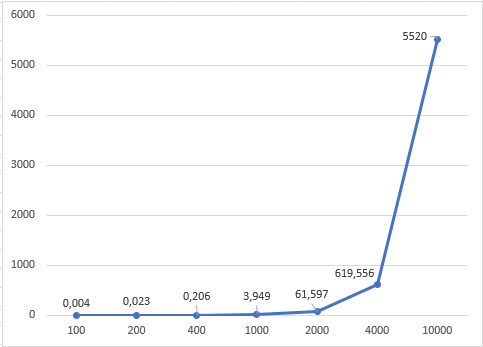


Рисунок 7 – результат работы программы (10000х10000)



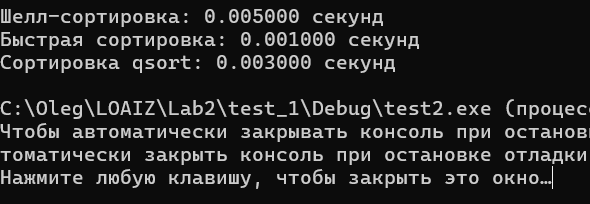


Рисунок 8 – результат работы программы (2.1/5)

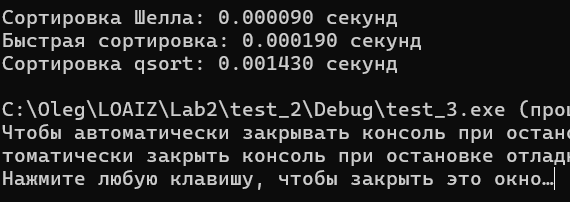


Рисунок 9 – результат работы программы (2.2/5)

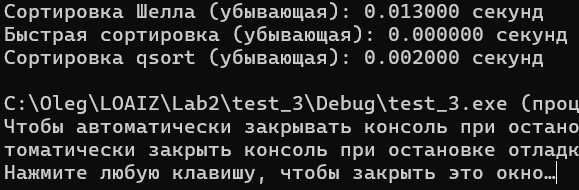


Рисунок 10 – результат работы программы (2.3/5)

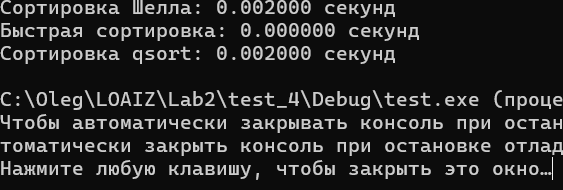


Рисунок 11 – результат работы программы (2.4/5)

**Вывод**

В ходе работы были закреплены знания работы с массивами и простыми структурами.

**Листинг**

**Задание 1**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <windows.h>

#define SIZE 10000

int main(void)

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

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

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

clock\_t start, end;

double cpu\_time\_used;

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

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

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

if (a == NULL || b == NULL || c == NULL) {

printf("Ошибка выделения памяти!\n");

return 1;

}

int i = 0, j = 0, r;

int helping = 10000;

int elem\_c;

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

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

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

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

}

if (a[i] == NULL || b[i] == NULL || c[i] == NULL) {

printf("Ошибка выделения памяти в строке %d!\n", i);

return 1;

}

long long op\_count = 0; // счётчик операций

start = clock();

srand(time(NULL));

while (i < helping)

{

j = 0;

while (j < helping)

{

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

j++;

op\_count++; // одна операция присвоения

}

i++;

}

srand(time(NULL));

i = 0;

j = 0;

while (i < helping)

{

j = 0;

while (j < helping)

{

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

j++;

op\_count++; // одна операция присвоения

}

i++;

}

// Умножение матриц

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

{

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

{

elem\_c = 0;

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

{

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

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

op\_count += 3; // одна операция сложения, умножения и присвоения

}

}

}

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

free(a[i]);

free(b[i]);

free(c[i]);

}

free(a);

free(b);

free(c);

end = clock();

cpu\_time\_used = ((double)(end - start)) / CLOCKS\_PER\_SEC;

printf("Время выполнения программы: %f секунд\n", cpu\_time\_used);

printf("Кол-во операций: %lld\n", op\_count);

return 0;

}

**Задание 2.1/5**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <windows.h>

// Реализация сортировки Шелла

void shell(int\* items, int count)

{

int i, j, gap, k;

int x, a[5];

// Шаги для сортировки Шелла

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; (j >= 0) && (items[j] > x); j -= gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

// Реализация быстрой сортировки (ручная)

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

{

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);

}

// Функция сравнения для qsort

int compare(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int count = 10000; // размер массива

int items1[10000], items2[10000], items3[10000];

clock\_t start, end;

double shell\_time, quicksort\_time, qsort\_time;

// Генерация случайного массива

srand(time(NULL));

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

items1[i] = rand() % 10000;

items2[i] = items1[i];

items3[i] = items1[i];

}

// Сортировка Шелла

start = clock();

shell(items1, count);

end = clock();

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

// Ручная быстрая сортировка

start = clock();

qs(items2, 0, count - 1);

end = clock();

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

// Стандартная qsort

start = clock();

qsort(items3, count, sizeof(int), compare);

end = clock();

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

// Вывод результатов

printf("Шелл-сортировка: %f секунд\n", shell\_time);

printf("Быстрая сортировка: %f секунд\n", quicksort\_time);

printf("Сортировка qsort: %f секунд\n", qsort\_time);

return 0;

}

**Задание 2.2/5**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <conio.h>

#include <windows.h>

// Реализация сортировки Шелла

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; (j >= 0) && (items[j] > x); j -= gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

// Реализация быстрой сортировки

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

{

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);

}

// Функция сравнения для qsort

int compare(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int count = 10000;

const int repeat = 100; // Количество повторений сортировки для точности

int items1[10000], items2[10000], items3[10000];

clock\_t start, end;

double shell\_time = 0, quicksort\_time = 0, qsort\_time = 0;

// Заполняем массивы данными

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

items1[i] = i;

items2[i] = items1[i];

items3[i] = items1[i];

}

// Повторяем сортировку несколько раз для усреднения времени

for (int r = 0; r < repeat; r++) {

// Сортировка Шелла

start = clock();

shell(items1, count);

end = clock();

shell\_time += ((double)(end - start)) / CLOCKS\_PER\_SEC;

// Обновляем массивы для следующей сортировки

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

items1[i] = i;

items2[i] = i;

items3[i] = i;

}

// Ручная быстрая сортировка

start = clock();

qs(items2, 0, count - 1);

end = clock();

quicksort\_time += ((double)(end - start)) / CLOCKS\_PER\_SEC;

// Обновляем массивы для следующей сортировки

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

items1[i] = i;

items2[i] = i;

items3[i] = i;

}

// Стандартная сортировка qsort

start = clock();

qsort(items3, count, sizeof(int), compare);

end = clock();

qsort\_time += ((double)(end - start)) / CLOCKS\_PER\_SEC;

}

// Усредняем время выполнения для каждой сортировки

shell\_time /= repeat;

quicksort\_time /= repeat;

qsort\_time /= repeat;

// Выводим результаты

printf("Сортировка Шелла: %f секунд\n", shell\_time);

printf("Быстрая сортировка: %f секунд\n", quicksort\_time);

printf("Cортировка qsort: %f секунд\n", qsort\_time);

return 0;

}

**Задание 2.3/5**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

// Реализация сортировки Шелла

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; (j >= 0) && (items[j] > x); j -= gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

// Реализация быстрой сортировки

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

{

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);

}

// Функция сравнения для qsort

int compare(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

int main()

{

SetConsoleOutputCP(1251);

SetConsoleCP(1251);

const int count = 10000; // Размер массива

int items1[10000], items2[10000], items3[10000];

clock\_t start, end;

double shell\_time, quicksort\_time, qsort\_time;

// Заполняем массивы данными (убывающая последовательность)

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

items1[i] = count - i;

items2[i] = items1[i];

items3[i] = items1[i];

}

// Сортировка Шелла

start = clock();

shell(items1, count);

end = clock();

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

// Быстрая сортировка (ручная)

start = clock();

qs(items2, 0, count - 1);

end = clock();

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

// Сортировка с использованием qsort

start = clock();

qsort(items3, count, sizeof(int), compare);

end = clock();

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

// Выводим результаты

printf("Сортировка Шелла (убывающая): %f секунд\n", shell\_time);

printf("Быстрая сортировка (убывающая): %f секунд\n", quicksort\_time);

printf("Сортировка qsort (убывающая): %f секунд\n", qsort\_time);

return 0;

}

**Задание 2.4/5**

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

#include <windows.h>

// Функция для сортировки Шелла

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; (j >= 0) && (items[j] > x); j -= gap)

items[j + gap] = items[j];

items[j + gap] = x;

}

}

}

// Функция для быстрой сортировки (реализованная вручную)

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

{

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);

}

// Функция сравнения для qsort

int compare(const void\* a, const void\* b)

{

return (\*(int\*)a - \*(int\*)b);

}

int main()

{

SetConsoleOutputCP(1251); // Для вывода на русском

SetConsoleCP(1251); // Для ввода на русском

const int count = 10000;

int items1[10000], items2[10000], items3[10000];

clock\_t start, end;

double shell\_time, quicksort\_time, qsort\_time;

// Заполнение массивов данными

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

items1[i] = i + 1;

items1[count / 2 + i] = count - i;

items2[i] = items1[i];

items2[count / 2 + i] = items1[count / 2 + i];

items3[i] = items1[i];

items3[count / 2 + i] = items1[count / 2 + i];

}

// Сортировка Шелла

start = clock();

shell(items1, count);

end = clock();

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

// Быстрая сортировка

start = clock();

qs(items2, 0, count - 1);

end = clock();

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

// Сортировка с помощью qsort

start = clock();

qsort(items3, count, sizeof(int), compare);

end = clock();

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

// Выводим результаты

printf("Сортировка Шелла: %f секунд\n", shell\_time);

printf("Быстрая сортировка: %f секунд\n", quicksort\_time);

printf("Cортировка qsort: %f секунд\n", qsort\_time);

return 0;

}