Министерство образования Российской федерации

Пензенский государственный университет

Кафедра «Вычислительная техника»

**ОТЧЁТ**

по лабораторной работе №2

по курсу «Логика и основы алгоритмизации в инженерных задачах»

на тему «Оценка времени выполнения программ»

Выполнили ст. группы 22ВВВ1:

Лёвин А.Д.

Колобов И.О.

Приняли:

К.э.н., доцент Акифьев И. В.

К.т.н., доцент Юрова О. В.

Пенза 2023

**Цель работы:**

Изучение простых структур данных и массивов, получение навыков составления простейших алгоритмов.

**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.

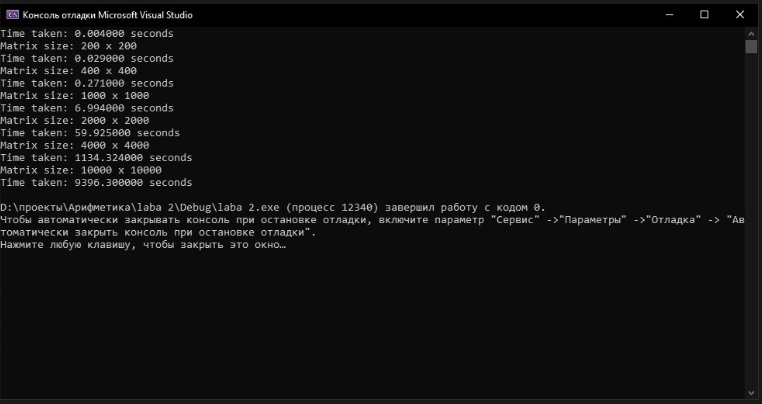
3) Оценить время работы каждого из реализованных алгоритмов на массиве, представляющем собой убывающую последовательность чисел.

4) Оценить время работы каждого из реализованных алгоритмов на массиве, одна половина которого представляет собой возрастающую последовательность чисел, а вторая, – убывающую.

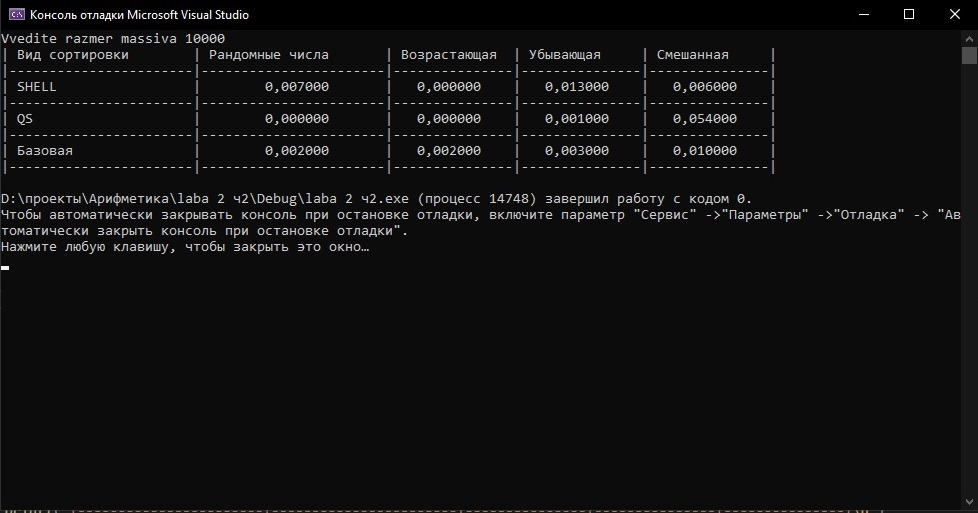
5) Оценить время работы стандартной функции qsort, реализующей алгоритм быстрой сортировки на выше указанных наборах данных.

**Результаты работы программы:**

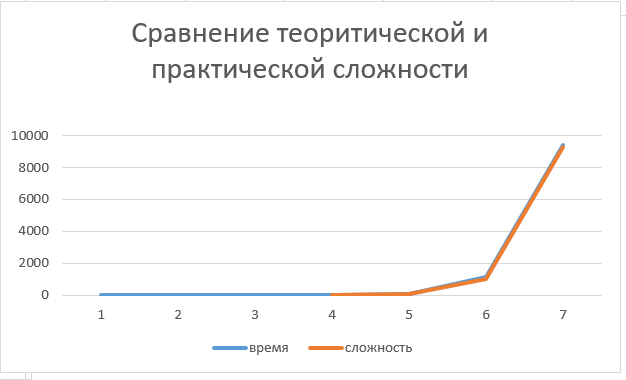
Задание 1:

****

Задание 2:



**График сравнения:**

****

**Листинг:**

***Задание 1:***

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

void test\_matrix\_mult\_time(int size) {

int i, j, r;

int\*\* a, \*\* b, \*\* c;

int elem\_c;

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

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

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

for (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));

}

i = 0; j = 0;

srand(time(NULL));

while (i < size) {

while (j < size) {

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

j++;

}

i++;

}

i = 0; j = 0;

srand(time(NULL));

while (i < size) {

while (j < size) {

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

j++;

}

i++;

}

clock\_t start = clock();

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

for (j = 0; j < size; j++) {

elem\_c = 0;

for (r = 0; r < size; r++) {

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

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

}

}

}

clock\_t end = clock();

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

printf("Matrix size: %d x %d\n", size, size);

printf("Time taken: %f seconds\n", time\_taken);

free(a);

free(b);

free(c);

}

int main() {

test\_matrix\_mult\_time(100);

test\_matrix\_mult\_time(200);

test\_matrix\_mult\_time(400);

test\_matrix\_mult\_time(1000);

test\_matrix\_mult\_time(2000);

test\_matrix\_mult\_time(4000);

test\_matrix\_mult\_time(10000);

return 0;

}

***Задание 2:***

**Файл Header.h**

#pragma once

#include <iostream>

#include <ctime>

#include <fstream>

#include <cstdlib>

using namespace std;

std::ofstream out;

void shell(int\* items, int count);

void qs(int\* chasti, int left, int right);

//void qsortRecursive(int\* mas, int size);

int qsortRecursive(const void\* a, const void\* b);

**Файл SHELLsort.cpp**

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;

}

}

}

**Файл QSsort.cpp**

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

{

// Создаем стек

int stack[100];

int top = -1;

int x, y;

int j, i;

// Добавляем начальные значения в стек

stack[++top] = left;

stack[++top] = right;

// Пока стек не пуст

while (top >= 0) {

// Извлекаем пару индексов из стека

right = stack[top--];

left = stack[top--];

i = left; j = right;

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

do {

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

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

if (i <= j) {

y = chasti[i];

chasti[i] = chasti[j];

chasti[j] = y;

i++; j--;

}

} while (i <= j);

// Добавляем новые пары индексов в стек

if (left < j) {

stack[++top] = left;

stack[++top] = j;

}

if (i < right) {

stack[++top] = i;

stack[++top] = right;

}

}

}

**Файл BAZsort.cpp**

int qsortRecursive(const void\* a, const void\* b) {

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

}

**Файл Main.cpp**

#include "Header.h"

int main() {

//SHELL

setlocale(LC\_ALL, "Russian");

int count;

std::cout << "Vvedite razmer massiva ";

std::cin >> count;

// 1. Рандомные числа

int\* items1 = new int[count];

srand(time(NULL));

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

items1[i] = rand() % 100;

}

out.open("СОРТИРОВКА.txt");

out << "SHELL сортировка:\n";

out << "Массив из рандомных чисел:\n";

out << "До сортировки:\n";

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

out << items1[i] << " ";

}

out << "\n\n";

out.close();

clock\_t start = clock();

shell(items1, count);

clock\_t end = clock();

double t1 = (double)(end - start) / CLOCKS\_PER\_SEC;

out.open("СОРТИРОВКА.txt", std::ios\_base::app);

out << "После сортировки:\n";

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

out << items1[i] << " ";

}

out << "\n\n";

// 2. Возрастающая

int\* items2 = new int[count];

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

items2[i] = i;

}

out << "Массив, состоящий из возрастающей последовательности:\n";

out << "До сортировки:\n";

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

out << items2[i] << " ";

}

out << "\n\n";

start = clock();

shell(items2, count);

end = clock();

double t2 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << items2[i] << " ";

}

out << "\n\n";

// 3. Убывающая

int\* items3 = new int[count];

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

items3[i] = count - i;

}

out << "Массив, состоящий из убывающей последовательности:\n";

out << "До сортировки:\n";

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

out << items3[i] << " ";

}

out << "\n\n";

start = clock();

shell(items3, count);

end = clock();

double t3 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << items3[i] << " ";

}

out << "\n\n";

// 4. Смешанная

int\* items4 = new int[count];

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

items4[i] = i;

}

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

items4[i] = count - i;

}

out << "Массив, состоящий из смешанной последовательности:\n";

out << "До сортировки:\n";

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

out << items4[i] << " ";

}

out << "\n\n";

start = clock();

shell(items4, count);

end = clock();

double t4 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << items4[i] << " ";

}

out << "\n\n";

//QSsort

// 1. Рандомные числа

int\* arr1 = new int[count];

srand(time(0));

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

arr1[i] = rand() % 100;

}

out << "QS сортировка:\n";

out << "Массив из рандомных чисел:\n";

out << "До сортировки:\n";

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

out << arr1[i] << " ";

}

out << "\n\n";

start = clock();

qs(arr1, 0, count - 1);

end = clock();

double time1 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << arr1[i] << " ";

}

out << "\n\n";

// 2. Возрастающая последовательность

int\* arr2 = new int[count];

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

arr2[i] = i;

}

out << "Массив, состоящий из возрастающей последовательности:\n";

out << "До сортировки:\n";

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

out << arr2[i] << " ";

}

out << "\n\n";

start = clock();

qs(arr2, 0, count - 1);

end = clock();

double time2 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << arr2[i] << " ";

}

out << "\n\n";

// 3. Убывающая последовательность

int\* arr3 = new int[count];

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

arr3[i] = count - i;

}

out << "Массив, состоящий из убывающей последовательности:\n";

out << "До сортировки:\n";

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

out << arr3[i] << " ";

}

out << "\n\n";

start = clock();

qs(arr3, 0, count - 1);

end = clock();

double time3 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << arr3[i] << " ";

}

out << "\n\n";

// 4. Смешанная последовательность

int\* arr4 = new int[count];

for (int i = 0; i < count /

2; i++) {

arr4[i] = i;

}

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

arr4[i] = count - i;

}

out << "Массив, состоящий из смешанной последовательности:\n";

out << "До сортировки:\n";

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

out << arr4[i] << " ";

}

out << "\n\n";

start = clock();

qs(arr4, 0, count - 1);

end = clock();

double time4 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << arr4[i] << " ";

}

out << "\n\n";

//BAZsort

// 1. Рандомные числа

int\* add1 = new int[count];

srand(time(0));

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

add1[i] = rand() % 100;

}

out << "BAZ сортировка:\n";

out << "Массив из рандомных чисел:\n";

out << "До сортировки:\n";

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

out << add1[i] << " ";

}

out << "\n\n";

start = clock();

qsort(add1, count, sizeof(int), qsortRecursive);

end = clock();

double tm1 = (double)(end - start) / CLOCKS\_PER\_SEC;

/\*clock\_t tm1 = clock();

qsortRecursive(add1, count);

tm1 = clock() - tm1;\*/

out << "После сортировки:\n";

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

out << add1[i] << " ";

}

out << "\n\n";

// 2. Возрастающая

int\* add2 = new int[count];

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

add2[i] = i;

}

out << "Массив, состоящий из возрастающей последовательности:\n";

out << "До сортировки:\n";

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

out << add2[i] << " ";

}

out << "\n\n";

start = clock();

//qsortRecursive(add2, count);

qsort(add2, count, sizeof(int), qsortRecursive);

end = clock();

double tm2 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << add2[i] << " ";

}

out << "\n\n";

// 3. Убывающая

int\* add3 = new int[count];

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

add3[i] = count - i;

}

out << "Массив, состоящий из убывающей последовательности:\n";

out << "До сортировки:\n";

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

out << add3[i] << " ";

}

out << "\n\n";

start = clock();

//qsortRecursive(add3, count);

qsort(add3, count, sizeof(int), qsortRecursive);

end = clock();

double tm3 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << add3[i] << " ";

}

out << "\n\n";

// 4. Смешанная

int\* add4 = new int[count];

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

add4[i] = i;

}

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

add4[i] = count - i;

}

out << "Массив, состоящий из смешанной последовательности:\n";

out << "До сортировки:\n";

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

out << add4[i] << " ";

}

out << "\n\n";

start = clock();

qsort(add4, count, sizeof(int), qsortRecursive);

end = clock();

double tm4 = (double)(end - start) / CLOCKS\_PER\_SEC;

out << "После сортировки:\n";

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

out << add4[i] << " ";

}

out << "\n\n";

// Вывод времени

printf("| Вид сортировки\t| Рандомные числа\t| Возрастающая\t| Убывающая\t| Смешанная\t|\n");

printf("|-----------------------|-----------------------|---------------|---------------|---------------|\n");

printf("| SHELL \t\t| %15f \t| %10f \t| %10f \t| %10f \t|\n", t1, t2, t3, t4);

printf("|-----------------------|-----------------------|---------------|---------------|---------------|\n");

printf("| QS \t\t| %15f \t| %10f \t| %10f \t| %10f \t|\n", time1, time2, time3, time4);

printf("|-----------------------|-----------------------|---------------|---------------|---------------|\n");

printf("| Базовая \t\t| %15f \t| %10f \t| %10f \t| %10f \t|\n", tm1, tm2, tm3, tm4);

printf("|-----------------------|-----------------------|---------------|---------------|---------------|\n");

getchar();

getchar();

}

**Вывод:**

Сортировка Shell, Qsort и базовая сортировка языка C (например, сортировка пузырьком или сортировка вставками) - это различные алгоритмы сортировки с разными характеристиками и преимуществами.  
  
1. Сортировка Shell:  
- Преимущество: Сортировка Shell является улучшением базовых сортировок и позволяет достичь более высокой производительности. Она основана на идее упорядочивания элементов в массиве с большими промежутками и последующего уменьшения этих промежутков для более точной сортировки. Поэтому сортировка Shell может быть эффективна для больших массивов данных.  
- Условие: Сортировка Shell может быть хорошим выбором, когда вам нужно отсортировать большой массив данных, особенно если вы не знаете много о его структуре или распределении.  
  
2. Qsort:  
- Преимущество: Сортировка Qsort - это стандартная функция сортировки языка C, которая обычно реализована с использованием быстрой сортировки (quick sort). Она обладает отличной производительностью в среднем случае и хорошо справляется с большими массивами данных. Быстрая сортировка также является сортировкой "на месте", то есть не требует дополнительной памяти, что важно при ограниченных ресурсах.  
- Условие: Сортировка Qsort может быть предпочтительна, когда вы работаете с большими массивами данных и нуждаетесь в хорошей производительности в среднем или лучшем случае. Однако она может показать себя неэффективной в худшем случае, имеет сложность O(n^2), и может вызывать проблемы с памятью, если сортируется очень большой массив данных.  
  
3. Базовая сортировка языка C:  
- Преимущество: Базовые сортировки, такие как сортировка пузырьком или сортировка вставками, просты в реализации и понимании. Они хорошо подходят для небольших массивов данных или уже частично отсортированных массивов. Они также являются стабильными сортировками, то есть они не меняют порядок элементов с равными значениями.  
- Условие: Базовые сортировки могут быть предпочтительными, когда вы работаете с небольшими массивами данных или уже частично отсортированными массивами. Они могут быть более простыми в реализации и могут дать хорошие результаты, если у вас нет возможности использовать более сложные алгоритмы сортировки или если вам нужно сохранить порядок элементов с равными значениями.  
  
В итоге, выбор между сортировками Shell, Qsort и базовой сортировкой на языке C будет зависеть от вашего конкретного случая использования и требований к производительности. Если вы имеете дело с большими массивами данных и ищете лучшую производительность, то сортировка Shell или Qsort может быть хорошим выбором. Если у вас небольшой массив данных или вам важна простота реализации, то базовая сортировка языка C может быть предпочтительнее.