Министерство науки и высшего образования РФ

Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Высшая школа киберфизических систем и управления

УДК 004.421

УТВЕРЖДАЮ

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_г.

Отчет

по дисциплине «Теория и технология программирования»

Лабораторная работа №6 «Методы сортировки», вариант 17

Выполнил:

Студент

гр.3530902/90001 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Непушкин С.А. подпись, дата

Доцент ВШКФСиУ,

Кандидат технических наук

С. В. Хлопин \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата

Санкт-Петербург

2020

**1. Цель работы.**

Цель задания - изучить методы сортировки данных.

**2. Задание:**

Составить программу для сортировки массива данных методами: пузырьковой, отбора, вставки, Шелла и быстрой сортировки. Вывести на экран неупорядоченную (один раз) и упорядоченные (для каждого из методов) массивы данных. Составить сравнительную таблицу эффективности методов, в которой необходимо указать число сравнений и перестановок переменных в каждом методе сортировки.

Неупорядоченная матрица задается один раз случайным образом, далее она используется для каждого из методов сортировки.

Индивидуальный вариант:

Упорядочить каждый столбец матрицы по убыванию.

**3. Текст программы**

#include <iostream>

#include <stdio.h>

#include <locale.h>

#include <time.h>

using namespace std;

int isDigit(char c)

{

return ((c >= '0') && (c <= '9'));

}

int charToDigit(char c)

{

if (isDigit(c))

{

return c - '0';

}

return -1;

}

//0 - Ошибка; 1 - Ввелось корректно

int inputInt(int &var)

{

char c = 0;

int value = -1;

while ((c = getchar()) != '\n')

{

if (!isDigit(c))

{

return 0;

}

if (value == -1)

{

value = 0;

}

value = value \* 10 + charToDigit(c);

}

if (value < 1)

{

return 0;

}

var = value;

return 1;

}

void initMatrix(int\*\* &matrix, int rows, int cols)

{

matrix = new int\*[rows];

for (int i = 0; i < rows; i++)

{

matrix[i] = new int[cols];

for (int j = 0; j < cols; j++)

{

matrix[i][j] = rand() % 100;

}

}

}

int\*\* copyMatrix(int\*\* matrix, int rows, int cols)

{

int\*\* res = nullptr;

initMatrix(res, rows, cols);

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

{

res[i][j] = matrix[i][j];

}

}

return res;

}

void deleteMatrix(int\*\* &matrix, int rows, int cols)

{

for (int i = 0; i < rows; i++)

{

delete[] matrix[i];

}

delete matrix;

}

void printMatrix(int\*\* &matrix, int rows, int cols)

{

for (int i = 0; i < rows; i++)

{

for (int j = 0; j < cols; j++)

{

printf("%2d ", matrix[i][j]);

}

printf("\n");

}

}

void swap(int &a, int &b)

{

int c = a;

a = b;

b = c;

}

void mixColMatrix(int\*\* matrix, int rows, int col)

{

for (int i = 0; i < rows; i++)

{

swap(matrix[i][col], matrix[rand() % rows][col]);

}

}

void mixMatrix(int\*\* matrix, int rows, int cols)

{

for (int j = 0; j < cols; j++)

{

for (int i = 0; i < rows; i++)

{

swap(matrix[i][j], matrix[rand() % rows][j]);

}

}

}

pair<int, int> bubbleColsSort(int\*\* inmatrix, int rows, int cols)

{

int\*\* matrix = copyMatrix(inmatrix, rows, cols);

int per = 0;

int srav = 0;

for (int k = 0, shift = 1; k < cols; k++)

{

for (int i = 0; (i < rows) && (shift > 0); i++)

{

shift = 0;

for (int j = 0; j < rows - i - 1; j++)

{

if (matrix[j][k] < matrix[j + 1][k])

{

swap(matrix[j][k], matrix[j + 1][k]);

per++;

shift++;

}

srav++;

}

}

shift = 1;

}

printf("Перестановки: %d \t Сравнения: %d \n", per, srav);

printMatrix(matrix, rows, cols);

deleteMatrix(matrix, rows, cols);

return pair<int, int>(per, srav);

}

pair<int, int> selectionColsSort(int\*\* inmatrix, int rows, int cols)

{

int\*\* matrix = copyMatrix(inmatrix, rows, cols);

int per = 0;

int srav = 0;

if (rows != 1)

{

for (int k = 0; k < cols; k++)

{

for (int i = 0; i < rows - 1; i++)

{

int index = i;

for (int j = i + 1; j < rows; j++)

{

if (matrix[j][k] > matrix[index][k])

{

index = j;

}

srav++;

}

if (index != i)

{

swap(matrix[index][k], matrix[i][k]);

per++;

}

}

}

}

printf("Перестановки: %d \t Сравнения: %d \n", per, srav);

printMatrix(matrix, rows, cols);

deleteMatrix(matrix, rows, cols);

return pair<int, int>(per, srav);

}

pair<int, int> insertColsSort(int\*\* inmatrix, int rows, int cols)

{

int\*\* matrix = copyMatrix(inmatrix, rows, cols);

int per = 0;

int srav = 0;

for (int k = 0; k < cols; k++)

{

for (int i = 1; i < rows; i++)

{

int temp = matrix[i][k];

int j = 0;

for (j = i; j > 0 && temp > matrix[j - 1][k]; j--)

{

matrix[j][k] = matrix[j - 1][k];

per++;

srav++;

}

matrix[j][k] = temp;

}

}

printf("Перестановки: %d \t Сравнения: %d \n", per, srav);

printMatrix(matrix, rows, cols);

deleteMatrix(matrix, rows, cols);

return pair<int, int>(per, srav);

}

pair<int, int> shellColsSort(int\*\* inmatrix, int rows, int cols)

{

int\*\* matrix = copyMatrix(inmatrix, rows, cols);

int per = 0;

int srav = 0;

for (int k = 0; k < cols; k++)

{

for (int diff = rows / 2; diff > 0; diff /= 2)

{

for (int i = diff; i < rows; i++)

{

int temp = matrix[i][k];

int j = 0;

for (j = i; j >= diff && temp > matrix[j - diff][k]; j -= diff)

{

matrix[j][k] = matrix[j - diff][k];

srav++;

per++;

}

matrix[j][k] = temp;

}

}

}

printf("Перестановки: %d \t Сравнения: %d \n", per, srav);

printMatrix(matrix, rows, cols);

deleteMatrix(matrix, rows, cols);

return pair<int, int>(per, srav);

}

pair<int, pair<int, int>> partition(int\*\* inmatrix, int low, int high, int col)

{

int per = 0;

int srav = 0;

int pivot = inmatrix[high][col];

int i = low;

for (int j = low; j <= high - 1; j++)

{

if (inmatrix[j][col] > pivot)

{

swap(inmatrix[i][col], inmatrix[j][col]);

i++;

per++;

}

srav++;

}

if (i != high)

{

swap(inmatrix[i][col], inmatrix[high][col]);

per++;

}

return pair<int, pair<int, int>>(i, pair<int, int>(per, srav));

}

pair<int, int> quickSort(int\*\* inmatrix, int low, int high, int col)

{

pair<int, int> res(0, 0);

if (low < high)

{

pair<int, pair<int, int>> pi = partition(inmatrix, low, high, col);

res.first += pi.second.first;

res.second += pi.second.second;

quickSort(inmatrix, low, pi.first - 1, col);

quickSort(inmatrix, pi.first + 1, high, col);

}

return res;

}

pair<int, int> quickColsSort(int\*\* inmatrix, int rows, int cols)

{

int\*\* matrix = copyMatrix(inmatrix, rows, cols);

int per = 0;

int srav = 0;

for (int k = 0; k < cols; k++)

{

pair<int, int> temp = quickSort(matrix, 0, rows - 1, k);

per += temp.first;

srav += temp.second;

}

printf("Перестановки: %d \t Сравнения: %d \n", per, srav);

printMatrix(matrix, rows, cols);

deleteMatrix(matrix, rows, cols);

return pair<int, int>(per, srav);

}

int colSorted(int\*\* matrix, int rows, int col)

{

for (int i = 0; i < rows - 1; i++)

{

if (matrix[i][col] < matrix[i + 1][col])

{

return 0;

}

}

return 1;

}

pair<int, int> monkeyColSort(int\*\* inmatrix, int rows, int cols)

{

int\*\* matrix = copyMatrix(inmatrix, rows, cols);

int per = 0;

int srav = 0;

for (int k = 0; k < cols; k++)

{

while (!colSorted(matrix, rows, k)) {

mixColMatrix(matrix, rows, k);

}

cout << endl;

printMatrix(matrix, rows, cols);

}

printf("Перестановки: %d \t Сравнения: %d \n", per, srav);

printMatrix(matrix, rows, cols);

deleteMatrix(matrix, rows, cols);

return pair<int, int>(per, srav);

}

int main()

{

srand(time(0));

setlocale(LC\_ALL, "Russian");

printf("Введите количество строк матрицы : ");

int n = 0;

if (!inputInt(n))

{

printf("Некорректный ввод числа\n");

return 0;

}

printf("Введите количество столбцов матрицы : ");

int m = 0;

if (!inputInt(m))

{

printf("Некорректный ввод числа\n");

return 0;

}

int\*\* matrix = nullptr;

initMatrix(matrix, n, m);

printMatrix(matrix, n, m);

//monkeyColSort(matrix, n, m);

pair<int, int> res[5];

printf("\n\nСортировка пузырьком:\n");

res[0] = bubbleColsSort(matrix, n, m);

printf("\n\nСортировка отбором:\n");

res[1] = selectionColsSort(matrix, n, m);

printf("\n\nСортировка вставками:\n");

res[2] = insertColsSort(matrix, n, m);

printf("\n\nСортировка Шелла:\n");

res[3] = shellColsSort(matrix, n, m);

printf("\n\nБыстрая сортировка:\n\n");

res[4] = quickColsSort(matrix, n, m);

deleteMatrix(matrix, n, m);

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

printf("\_\_\_\_\_\_\_\_\_\_\_\_|Сортировка пузырьком| Сортировка отбором |Сортировка вставками| Сортировка Шелла | Быстрая сортировка |\n");

printf("Перестановки|");

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

{

printf("%20d|", res[i].first);

}

printf("\nСравнения |");

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

{

printf("%20d|", res[i].second);

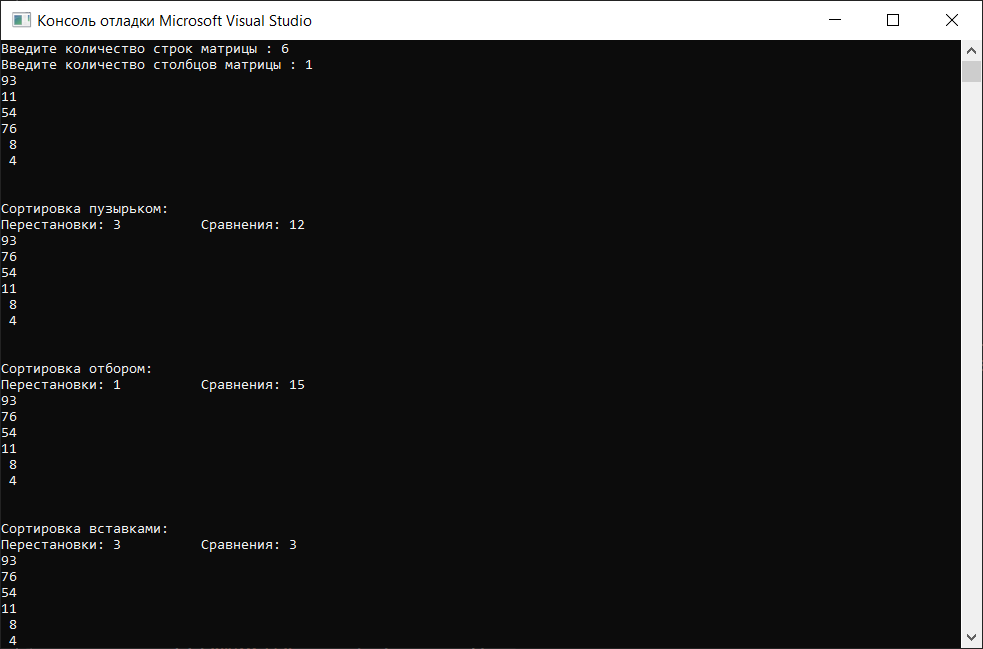
}

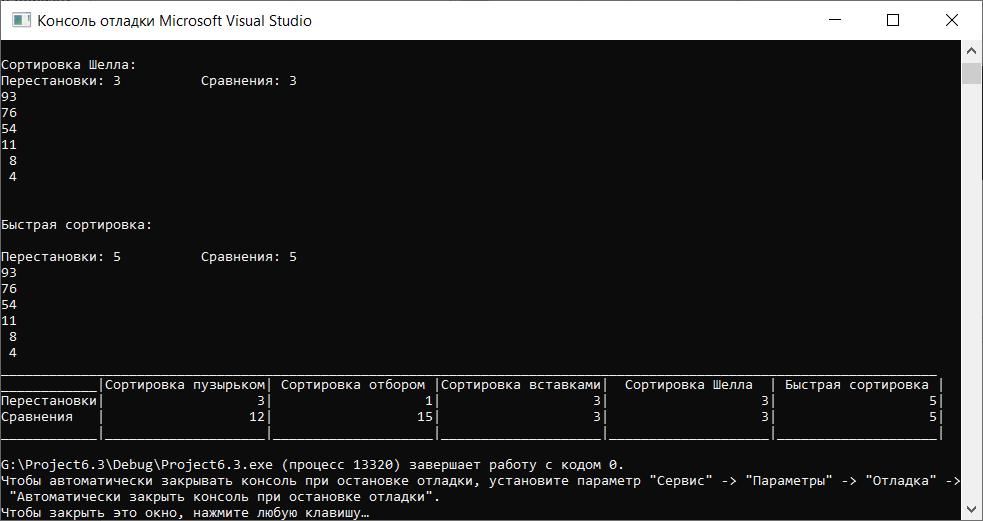
printf("\n\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\n");

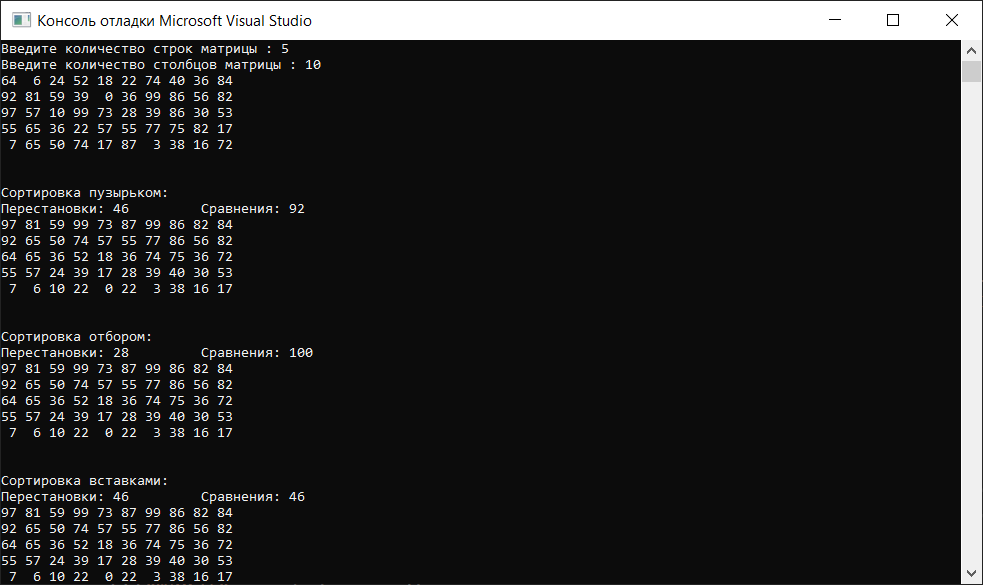
return 0;

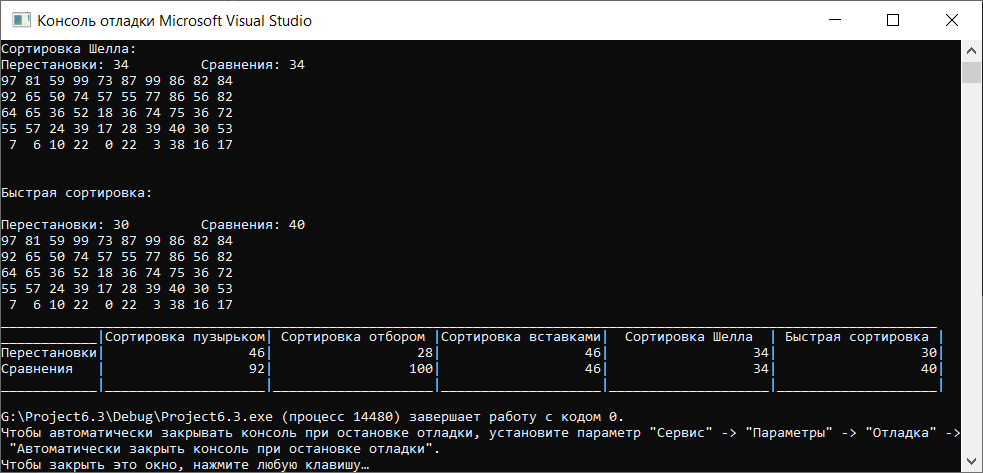
}

**4. Пример работы программы**









**Вывод**

В ходе лабораторной работы я:

-столкнулся с проблемой подсчета сравнений в быстрой сортировке, но успешно решил ее

-узнал о классе «pair»

-узнал, как можно оптимизировать метод сортировки «пузырьком»

-подробно разобрал метод быстрой сортировки и его алгоритм

-столкнулся с проблемой сохранения неотсортированного массива, но успешно решил ее