Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Расчетно-графическая работа**

по дисциплине «Алгоритмы и структуры данных»

Выполнил

студент гр. 5130901/20003 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Вагнер А.А.

(подпись)

Руководитель \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  Гаранин В.А.

(подпись)

 «\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2024 г.

Санкт-Петербург   
2024

**Цель работы.**

Целями работы являются:

* изучение методик сортировки данных и анализ зависимости скорости сортировки от размера массива данных.
* Изучение системы автоматического документирования кода программы

**2. Задание:**

На основе программы, написанной в рамках лабораторной работы №6 (методы сортировки):

1. Создать документацию программному коду посредством использования системы автоматизированного документирования Doxygen с обязательным указанием параметров принимаемых функциями, назначением функций и автоматическим построением блок схемы программы. Документация является отдельным разделом расчетно-графической работы.
2. провести исследование используемых методов сортировки:

В соответствии с индивидуальным заданием (таблица 8):

* Провести сортировку различных массивов данных;
* Заполнить таблицу значений (таблица 9) числа сравнений и перестановок для сортируемых массивов;
* Провести нормирование значений числа сравнений и перестановок и заполнить таблицу нормированных значений (таблица 10) для сортируемых массивов;
* Построить графики зависимости числа сравнений и перестановок от размеров массива;
* Построить графики с логарифмической шкалой зависимости числа сравнений и перестановок от размеров массива;
* Построить графики зависимости нормированных значений сравнений и перестановок от размеров массива;
* Сделать вывод о качественных характеристиках методов сортировок для различных размеров данных.

1. Код программы

#include <time.h>

#include <windows.h>

#include <iostream>

#include <iomanip>

#include "Fun.h"

#include <chrono>

#include <math.h>

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

/// \brief Функция сортировки методом пузырька

/// \param x - Массив для сортировки

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

/// \param n - Ширина массива

/// \param m - Длина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void bubble(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Bubble sort" << std::endl;

for (size\_t i = 1; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++)

for (size\_t k = 1; k < m; k++) {

comparisons++;

if (abs(x[i][k]) < abs(x[i][k - 1])) {

int temp = x[i][k];

x[i][k] = x[i][k - 1];

x[i][k - 1] = temp;

permutations++;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t i = 0; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++)

for (size\_t k = 1; k < m; k++) {

comparisons++;

if (x[k][i] < x[k - 1][i]) {

int temp = x[k][i];

x[k][i] = x[k - 1][i];

x[k - 1][i] = temp;

permutations++;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

}

\endcode \*/

void bubble(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Bubble sort" << std::endl;

for (size\_t i = 1; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++)

for (size\_t k = 1; k < m; k++) {

comparisons++;

if (abs(x[i][k]) < abs(x[i][k - 1])) {

int temp = x[i][k];

x[i][k] = x[i][k - 1];

x[i][k - 1] = temp;

permutations++;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t i = 0; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++)

for (size\_t k = 1; k < m; k++) {

comparisons++;

if (x[k][i] < x[k - 1][i]) {

int temp = x[k][i];

x[k][i] = x[k - 1][i];

x[k - 1][i] = temp;

permutations++;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

}

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

/// \brief Функция сортировки методом отбора

/// \param x - Массив для сортировки

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

/// \param n - Ширина массива

/// \param m - Длина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void selection(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Selection sort" << std::endl;

for (size\_t i = 1; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++) {

int jmin = j;

for (size\_t k = j + 1; k < m; k++) {

comparisons++;

if (abs(x[i][k]) < abs(x[i][jmin])) jmin = k;

}

int temp = x[i][j];

x[i][j] = x[i][jmin];

x[i][jmin] = temp;

permutations++;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t i = 0; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++) {

int jmin = j;

for (size\_t k = j + 1; k < m; k++) {

comparisons++;

if (x[k][i] < x[jmin][i]) jmin = k;

}

int temp = x[j][i];

x[j][i] = x[jmin][i];

x[jmin][i] = temp;

permutations++;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

}

\endcode \*/

void selection(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Selection sort" << std::endl;

for (size\_t i = 1; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++) {

int jmin = j;

for (size\_t k = j + 1; k < m; k++) {

comparisons++;

if (abs(x[i][k]) < abs(x[i][jmin])) jmin = k;

}

int temp = x[i][j];

x[i][j] = x[i][jmin];

x[i][jmin] = temp;

permutations++;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t i = 0; i < n; i += 2) {

for (size\_t j = 0; j < m - 1; j++) {

int jmin = j;

for (size\_t k = j + 1; k < m; k++) {

comparisons++;

if (x[k][i] < x[jmin][i]) jmin = k;

}

int temp = x[j][i];

x[j][i] = x[jmin][i];

x[jmin][i] = temp;

permutations++;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

}

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

/// \brief Функция сортировки методом вставки

/// \param x - Массив для сортировки

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

/// \param n - Ширина массива

/// \param m - Длина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void insertion(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Insertion sort" << std::endl;

for (size\_t i = 1; i < n; i += 2) {

for (size\_t j = 1; j < m; j++) {

int temp = x[i][j];

int k = j - 1;

comparisons++;

while (k >= 0 && abs(x[i][k]) > abs(temp)) {

x[i][k + 1] = x[i][k];

k--;

comparisons++;

permutations++;

}

x[i][k + 1] = temp;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t i = 0; i < n; i += 2) {

for (size\_t j = 1; j < m; j++) {

int temp = x[j][i];

int k = j - 1;

comparisons++;

while (k >= 0 && x[k][i] > temp) {

x[k + 1][i] = x[k][i];

k--;

comparisons++;

permutations++;

}

x[k + 1][i] = temp;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

}

\endcode \*/

void insertion(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Insertion sort" << std::endl;

for (size\_t i = 1; i < n; i += 2) {

for (size\_t j = 1; j < m; j++) {

int temp = x[i][j];

int k = j - 1;

comparisons++;

while (k >= 0 && abs(x[i][k]) > abs(temp)) {

x[i][k + 1] = x[i][k];

k--;

comparisons++;

permutations++;

}

x[i][k + 1] = temp;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t i = 0; i < n; i += 2) {

for (size\_t j = 1; j < m; j++) {

int temp = x[j][i];

int k = j - 1;

comparisons++;

while (k >= 0 && x[k][i] > temp) {

x[k + 1][i] = x[k][i];

k--;

comparisons++;

permutations++;

}

x[k + 1][i] = temp;

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparisons << "\n" << "Permutations: " << permutations << std::endl;

reset(comparisons, permutations);

}

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

/// \brief Функция быстрой сортировки.

///

/// Вызывает функции linesQuickSort и columnsQuickSort.

/// \see linesQuickSort columnsQuickSort

/// \param x - Массив для сортировки

/// \param lines - Число строк в массиве

/// \param columns - Число столбцов в массиве

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void matrixQuickSort(int\*\* x, int lines, int columns, int &comparison, int &permutation) {

std::cout << std::endl << "Quick sort" << std::endl;

for (size\_t i = 1; i < lines; i+=2)

{

linesQuickSort(x, 0, lines - 1, i, comparison, permutation);

}

for (size\_t i = 0; i < lines; i++) {

for (size\_t j = 0; j < columns; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparison << "\n" << "Permutations: " << permutation << std::endl;

reset(comparison, permutation);

std::cout << std::endl;

for (size\_t i = 0; i < columns; i+=2)

{

columnsQuickSort(x, 0, lines - 1, i, comparison, permutation);

}

for (size\_t i = 0; i < lines; i++) {

for (size\_t j = 0; j < columns; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparison << "\n" << "Permutations: " << permutation << std::endl;

reset(comparison, permutation);

}

\endcode \*/

void matrixQuickSort(int\*\* x, int lines, int columns, int &comparison, int &permutation) {

std::cout << std::endl << "Quick sort" << std::endl;

for (size\_t i = 1; i < lines; i+=2)

{

linesQuickSort(x, 0, lines - 1, i, comparison, permutation);

}

for (size\_t i = 0; i < lines; i++) {

for (size\_t j = 0; j < columns; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparison << "\n" << "Permutations: " << permutation << std::endl;

reset(comparison, permutation);

std::cout << std::endl;

for (size\_t i = 0; i < columns; i+=2)

{

columnsQuickSort(x, 0, lines - 1, i, comparison, permutation);

}

for (size\_t i = 0; i < lines; i++) {

for (size\_t j = 0; j < columns; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Comparisons: " << comparison << "\n" << "Permutations: " << permutation << std::endl;

reset(comparison, permutation);

}

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

/// \brief Функция сортировки части строки методом быстрой сортировки

/// \see matrixQuickSort

/// \param x - Массив для сортировки

/// \param l - Индекс левой границы

/// \param r - Индекс правой границы

/// \param line - Индекс строки для сортировки

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void linesQuickSort(int\*\* x, int l, int r, int line, int &comparison, int &permutation) {

int leftI, rightI, centerValue;

leftI = l;

rightI = r;

if (l > r) return;

centerValue = x[line][(rightI + leftI) / 2];

while (leftI <= rightI) {

while (abs(centerValue) > abs(x[line][leftI])) {

comparison++;

leftI++;

}

while (abs(x[line][rightI]) > abs(centerValue)) {

comparison++;

rightI--;

}

if (leftI <= rightI) {

int temp = x[line][rightI];

x[line][rightI] = x[line][leftI];

x[line][leftI] = temp;

leftI++;

rightI--;

permutation++;

}

}

linesQuickSort(x, l, rightI, line, comparison, permutation);

linesQuickSort(x, leftI, r, line, comparison, permutation);

}

\endcode \*/

void linesQuickSort(int\*\* x, int l, int r, int line, int &comparison, int &permutation) {

int leftI, rightI, centerValue;

leftI = l;

rightI = r;

if (l > r) return;

centerValue = x[line][(rightI + leftI) / 2];

while (leftI <= rightI) {

while (abs(centerValue) > abs(x[line][leftI])) {

comparison++;

leftI++;

}

while (abs(x[line][rightI]) > abs(centerValue)) {

comparison++;

rightI--;

}

if (leftI <= rightI) {

int temp = x[line][rightI];

x[line][rightI] = x[line][leftI];

x[line][leftI] = temp;

leftI++;

rightI--;

permutation++;

}

}

linesQuickSort(x, l, rightI, line, comparison, permutation);

linesQuickSort(x, leftI, r, line, comparison, permutation);

}

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

/// \brief Функция сортировки части столбца методом быстрой сортировки

/// \see matrixQuickSort

/// \param x - Массив для сортировки

/// \param l - Индекс левой границы

/// \param r - Индекс правой границы

/// \param column - Индекс строки для сортировки

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void columnsQuickSort(int\*\* x, int l, int r, int column, int &comparison, int &permutation) {

int leftI, rightI, centerValue;

leftI = l;

rightI = r;

if (l > r) return;

centerValue = x[(rightI + leftI) / 2][column];

while (leftI <= rightI) {

while (centerValue > x[leftI][column]) {

comparison++;

leftI++;

}

while (x[rightI][column] > centerValue) {

comparison++;

rightI--;

}

if (leftI <= rightI) {

int temp = x[rightI][column];

x[rightI][column] = x[leftI][column];

x[leftI][column] = temp;

leftI++;

rightI--;

permutation++;

}

}

columnsQuickSort(x, l, rightI, column, comparison, permutation);

columnsQuickSort(x, leftI, r, column, comparison, permutation);

}

\endcode \*/

void columnsQuickSort(int\*\* x, int l, int r, int column, int &comparison, int &permutation) {

int leftI, rightI, centerValue;

leftI = l;

rightI = r;

if (l > r) return;

centerValue = x[(rightI + leftI) / 2][column];

while (leftI <= rightI) {

while (centerValue > x[leftI][column]) {

comparison++;

leftI++;

}

while (x[rightI][column] > centerValue) {

comparison++;

rightI--;

}

if (leftI <= rightI) {

int temp = x[rightI][column];

x[rightI][column] = x[leftI][column];

x[leftI][column] = temp;

leftI++;

rightI--;

permutation++;

}

}

columnsQuickSort(x, l, rightI, column, comparison, permutation);

columnsQuickSort(x, leftI, r, column, comparison, permutation);

}

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

/// \brief Функция сортировки методом Шелла

/// \param x - Массив для сортировки

/// \param comparisons - Число сравнений

/// \param permutations - Число замен

/// \param n - Ширина массива

/// \param m - Длина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void shellSort(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Shell sort" << std::endl;

for (size\_t p = 1; p < n; p += 2) {

for (int gap = n / 2; gap > 0; gap /= 2) {

for (int i = gap; i < n; i++) {

int temp = x[p][i];

int j;

for (j = i; j >= gap && abs(x[p][j - gap]) > abs(temp); j -= gap) {

x[p][j] = x[p][j - gap];

comparisons++;

permutations++;

}

x[p][j] = temp;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Number of comparisons: " << comparisons << "\n" << "Number of permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t p = 0; p < n; p += 2) {

for (int gap = n / 2; gap > 0; gap /= 2) {

for (int i = gap; i < n; i++) {

int temp = x[i][p];

int j;

for (j = i; j >= gap && x[j - gap][p] > temp; j -= gap) {

x[j][p] = x[j - gap][p];

comparisons++;

permutations++;

}

x[j][p] = temp;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Number of comparisons: " << comparisons << "\n" << "Number of permutations: " << permutations << std::endl;

}

\endcode \*/

void shellSort(int\*\* x, int& comparisons, int& permutations, int& n, int& m) {

std::cout << std::endl << "Shell sort" << std::endl;

for (size\_t p = 1; p < n; p += 2) {

for (int gap = n / 2; gap > 0; gap /= 2) {

for (int i = gap; i < n; i++) {

int temp = x[p][i];

int j;

for (j = i; j >= gap && abs(x[p][j - gap]) > abs(temp); j -= gap) {

x[p][j] = x[p][j - gap];

comparisons++;

permutations++;

}

x[p][j] = temp;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Number of comparisons: " << comparisons << "\n" << "Number of permutations: " << permutations << std::endl;

reset(comparisons, permutations);

std::cout << std::endl;

for (size\_t p = 0; p < n; p += 2) {

for (int gap = n / 2; gap > 0; gap /= 2) {

for (int i = gap; i < n; i++) {

int temp = x[i][p];

int j;

for (j = i; j >= gap && x[j - gap][p] > temp; j -= gap) {

x[j][p] = x[j - gap][p];

comparisons++;

permutations++;

}

x[j][p] = temp;

}

}

}

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

std::cout << std::endl << "Number of comparisons: " << comparisons << "\n" << "Number of permutations: " << permutations << std::endl;

}

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

/// \brief Функция наполнения массива случайными элементами

/// \param x - Массив для наполнения

/// \param lines - Длина массива

/// \param columns - Ширина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void initMatrix(int\*\* x, int lines, int columns)

{

for (size\_t i = 0; i < lines; i++)

for (size\_t j = 0; j < columns; j++)

x[i][j] = rand() - RAND\_MAX / 2;

}

\endcode \*/

void initMatrix(int\*\* x, int lines, int columns)

{

for (size\_t i = 0; i < lines; i++)

for (size\_t j = 0; j < columns; j++)

x[i][j] = rand() - RAND\_MAX / 2;

}

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

/// \brief Функция удаления массива

/// \param x - Массив для наполнения

/// \param lines - Длина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void deleteMatrix(int\*\* x, int lines)

{

for (size\_t i = 0; i < lines; i++)

delete[] x[i];

delete[] x;

}

\endcode \*/

void deleteMatrix(int\*\* x, int lines)

{

for (size\_t i = 0; i < lines; i++)

delete[] x[i];

delete[] x;

}

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

/// \brief Функция заполнения обработанной матрицы начальными значениями

/// \param x - Массив для наполнения

/// \param duplicate\_x - Массив с исходными значениями

/// \param lines - Длина массива

/// \param columns - Ширина массива

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void deleteMatrix(int\*\* x, int lines)

{

for (size\_t i = 0; i < lines; i++)

delete[] x[i];

delete[] x;

}

\endcode \*/

void rollbMatrix(int\*\* x, int\*\* duplicate\_x, int lines, int columns)

{

for (size\_t i = 0; i < lines; i++)

{

for (size\_t j = 0; j < columns; j++)

{

duplicate\_x[i][j] = x[i][j];

}

}

}

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

/// \brief Функция обнуления счётчиков сравнений и замен

/// \param &comparison - Ссылка на счётчик сравнений

/// \param &permutation - Ссылка на счётчик замен

///

/// \*\*Код функции:\*\*

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

/\*\* \code

void reset(int &comparison, int &permutation)

{

comparison = 0;

permutation = 0;

}

\endcode \*/

void reset(int &comparison, int &permutation)

{

comparison = 0;

permutation = 0;

}

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

/// \brief main()

///

/// \*\*Код:\*\*

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

/\*\* \code

int main()

{

int n, m, \*\* x, \*\* dublicate\_x, comparisons, permutations; // n - строки, m - столбцы;

std::cout << "Matrix height: ";

std::cin >> n;

std::cout << "Matrix width: ";

std::cin >> m;

std::cout << "\n";

comparisons = 0;

permutations = 0;

x = new int\* [n];

for (size\_t i = 0; i < n; i++) x[i] = new int[m];

dublicate\_x = new int\* [n];

for (size\_t i = 0; i < n; i++) dublicate\_x[i] = new int[m];

initMatrix(x, n, m);

rollbMatrix(x, dublicate\_x, n, m);

std::cout << "Original matrix:" << "\n";

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

bubble(x, comparisons, permutations, n, m);

rollbMatrix(dublicate\_x, x, n, m);

selection(x, comparisons, permutations, n, m);

rollbMatrix(dublicate\_x, x, n, m);

insertion(x, comparisons, permutations, n, m);

rollbMatrix(dublicate\_x, x, n, m);

matrixQuickSort(x, n, m, comparisons, permutations);

rollbMatrix(dublicate\_x, x, n, m);

shellSort(x, comparisons, permutations, n, m);

for (size\_t i = 0; i < n; i++)

delete[] x[i];

delete[] x;

for (size\_t i = 0; i < n; i++)

delete dublicate\_x[i];

delete[] dublicate\_x;

}

\endcode \*/

int main()

{

int n, m, \*\* x, \*\* dublicate\_x, comparisons, permutations; // n - строки, m - столбцы;

std::cout << "Matrix height: ";

std::cin >> n;

std::cout << "Matrix width: ";

std::cin >> m;

std::cout << "\n";

comparisons = 0;

permutations = 0;

x = new int\* [n];

for (size\_t i = 0; i < n; i++) x[i] = new int[m];

dublicate\_x = new int\* [n];

for (size\_t i = 0; i < n; i++) dublicate\_x[i] = new int[m];

initMatrix(x, n, m);

rollbMatrix(x, dublicate\_x, n, m);

std::cout << "Original matrix:" << "\n";

for (size\_t i = 0; i < n; i++) {

for (size\_t j = 0; j < m; j++)

std::cout << x[i][j] << "\t";

std::cout << std::endl;

}

bubble(x, comparisons, permutations, n, m);

rollbMatrix(dublicate\_x, x, n, m);

selection(x, comparisons, permutations, n, m);

rollbMatrix(dublicate\_x, x, n, m);

insertion(x, comparisons, permutations, n, m);

rollbMatrix(dublicate\_x, x, n, m);

matrixQuickSort(x, n, m, comparisons, permutations);

rollbMatrix(dublicate\_x, x, n, m);

shellSort(x, comparisons, permutations, n, m);

for (size\_t i = 0; i < n; i++)

delete[] x[i];

delete[] x;

for (size\_t i = 0; i < n; i++)

delete dublicate\_x[i];

delete[] dublicate\_x;

}

Результат работы кода:

Изображение выглядит как текст, снимок экрана

Автоматически созданное описание

Полученная страница содержит документацию всех функций, включённых в код лабораторной работы, а именно краткое описание, описание аргументов, код самой функции и ссылки на документацию связанных функций. Далее исследуем статистики разных методов сортировки

Табл.1 Сравнительная таблица числа сравнений и перестановок для методов сортировок

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод сортировки | Размер массива | 360 | 720 | 1080 | 1440 | 1800 | 2160 | 2520 |
| пузырьковая | сравнения | 46397160 | 372211920 | 1257380280 | 2981838240 | 5825521800 | 10068366960 | 15990309720 |
| перестановки | 11659700 | 93171033 | 314369565 | 745706964 | 1456885285 | 2518629325 | 3999316375 |
| отбор | сравнения | 23263200 | 186364800 | 629272800 | 1491955200 | 2914380000 | 5036515200 | 7998328800 |
| перестановки | 129240 | 517680 | 1165320 | 2072160 | 3238200 | 4663440 | 6347880 |
| вставка | сравнения | 11788940 | 93688713 | 315534885 | 747779124 | 1460123485 | 2523292765 | 4005664255 |
| перестановки | 11659700 | 93171033 | 314369565 | 745706964 | 1456885285 | 2518629325 | 3999316375 |
| шелл | сравнения | 1636960 | 7900972 | 19834301 | 38132759 | 66462469 | 92870425 | 123805178 |
| перестановки | 728320 | 3750892 | 9332381 | 19464599 | 37295269 | 46205785 | 60286058 |
| быстрая | сравнения | 824793 | 11585845 | 28694178 | 55006552 | 93194342 | 132226545 | 178779948 |
| перестановки | 385796 | 5420589 | 13259784 | 26653303 | 48800720 | 63104645 | 83652444 |

Табл.1 Сравнительная таблица нормированных величин числа сравнений и перестановок для методов сортировок

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Метод сортировки | Размер массива | 360 | 720 | 1080 | 1440 | 1800 | 2160 | 2520 |
| пузырьковая | сравнения | -0.43585348 | -0.31205973 | 0.024261168 | 0.679471264 | 1.759932616 | 3.372007279 | 5.622057309 |
| перестановки | -0.44905202 | -0.41808168 | -0.33403701 | -0.17014979 | 0.100063361 | 0.503474421 | 1.066063435 |
| отбор | сравнения | -0.44464326 | -0.38267259 | -0.2143891 | 0.113388221 | 0.653840416 | 1.460148509 | 2.585493526 |
| перестановки | -0.45343304 | -0.45328545 | -0.45303938 | -0.45269482 | -0.45225178 | -0.45171026 | -0.45107026 |
| вставка | сравнения | -0.44900292 | -0.41788499 | -0.33359424 | -0.16936247 | 0.101293719 | 0.505246301 | 1.068475321 |
| перестановки | -0.44905202 | -0.41808168 | -0.33403701 | -0.17014979 | 0.100063361 | 0.503474421 | 1.066063435 |
| шелл | сравнения | -0.45286018 | -0.45048016 | -0.44594607 | -0.43899355 | -0.42822964 | -0.4181959 | -0.4064422 |
| перестановки | -0.45320542 | -0.45205699 | -0.44993629 | -0.44608654 | -0.43931175 | -0.43592619 | -0.43057637 |
| быстрая | сравнения | -0.45316876 | -0.44908008 | -0.44257975 | -0.43258233 | -0.41807283 | -0.40324249 | -0.38555446 |
| перестановки | -0.45333556 | -0.45142258 | -0.44844407 | -0.44335518 | -0.43494024 | -0.42950545 | -0.42169828 |

Рис.1 Графики с логарифмической шкалой зависимости числа сравнений и перестановок от размера массива

Изображение выглядит как линия, График, диаграмма, текст

Автоматически созданное описание

Рис.2 Графики зависимости числа сравнений и перестановок от размера массива

Изображение выглядит как текст, линия, График, снимок экрана

Автоматически созданное описание

Рис.3 Графики зависимости нормированных значений числа сравнений и перестановок от размера массива

Изображение выглядит как линия, График, диаграмма, снимок экрана

Автоматически созданное описание

**Вывод**

В ходе выполнения данной лабораторной работы были изучены методы создания документации кода при помощи инструментария doxygen. Также разработаны таблицы для сравнения различных методов сортировки по количеству сравнений и перестановок. По табличным данным построены графики, указанные по заданию.