Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и кибербезопасности

Высшая школа компьютерных технологий и информационных систем

**Лабораторная работа № 6**

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

Цель задания - изучить методы сортировки данных.

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. Упорядочить каждый столбец матрицы по убыванию абсолютных величин

Код программы

#include <time.h>

#include <windows.h>

#include <iostream>

#include <iomanip>

#include "Header.h"

#include <chrono>

#include <math.h>

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);

}

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);

}

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);

}

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);

}

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);

}

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);

}

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;

}

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;

}

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

{

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

delete[] x[i];

delete[] x;

}

void clearMatrix(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];

}

}

}

void reset(int &comparison, int &permutation)

{

comparison = 0;

permutation = 0;

}

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);

clearMatrix(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;

}

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

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

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

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

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

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

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

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

clearMatrix(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 программы:

Matrix height: 4

Matrix width: 4

Original matrix:

-16342 2084 -10049 10117

2786 -659 -4905 12975

10579 8081 -10678 11762

6898 444 -6422 -15892

Bubble sort

-16342 2084 -10049 10117

-659 2786 -4905 12975

10579 8081 -10678 11762

444 -6422 6898 -15892

Comparisons: 18

Permutations: 3

-16342 2084 -10678 10117

-659 2786 -10049 12975

444 8081 -4905 11762

10579 -6422 6898 -15892

Comparisons: 18

Permutations: 3

Selection sort

-16342 2084 -10049 10117

-659 2786 -4905 12975

10579 8081 -10678 11762

444 -6422 6898 -15892

Comparisons: 12

Permutations: 6

-16342 2084 -10678 10117

-659 2786 -10049 12975

444 8081 -4905 11762

10579 -6422 6898 -15892

Comparisons: 12

Permutations: 6

Insertion sort

-16342 2084 -10049 10117

-659 2786 -4905 12975

10579 8081 -10678 11762

444 -6422 6898 -15892

Comparisons: 9

Permutations: 3

-16342 2084 -10678 10117

-659 2786 -10049 12975

444 8081 -4905 11762

10579 -6422 6898 -15892

Comparisons: 9

Permutations: 3

Quick sort

-16342 2084 -10049 10117

-659 2786 -4905 12975

10579 8081 -10678 11762

444 -6422 6898 -15892

Comparisons: 8

Permutations: 11

-16342 2084 -10678 10117

-659 2786 -10049 12975

444 8081 -4905 11762

10579 -6422 6898 -15892

Comparisons: 6

Permutations: 11

Shell sort

-16342 2084 -10049 10117

-659 2786 -4905 12975

10579 8081 -10678 11762

444 -6422 6898 -15892

Number of comparisons: 3

Number of permutations: 3

-16342 2084 -10678 10117

-659 2786 -10049 12975

444 8081 -4905 11762

10579 -6422 6898 -15892

Number of comparisons: 3

Number of permutations: 3

Сравнительная таблица

|  |  |  |
| --- | --- | --- |
| Метод | Число сравнений | Число замен |
| Пузырьковая | 36 | 6 |
| Отбора | 24 | 12 |
| Вставки | 18 | 6 |
| Шелла | 6 | 6 |
| Быстрая | 14 | 22 |

Вывод

В ходе выполнения лабораторной работы выполнено задание, получены навыки сортировки двумерных массивов, результаты выполнения программы соответствуют ожидаемым.