Лабораторная работа №3

Сортировка массивов

Составить программу, проводящую сравнительную характеристику методов сортировки массивов.

Программа должна выполнять следующие действия:

1. Производить сортировку массива соответствующими методами.

2. Иллюстрировать работу каждого метода на небольших массивах (размером до 10 элементов).

3. Производить сортировку каждым из методов случайного массива, уже отсортированного массива,

массива, отсортированного в обратном порядке. Засечь время. Размер массива при этом должен выбираться

пользователем. После проведения сортировки, вывести данные о скорости работы методов.

Вариант 3. Метод прямого обмена, шейкерная сортировка, быстрая сортировка.

Код:

FileName.h

#include <iostream>

#include <algorithm>

#include <chrono>

#include <iomanip>

using namespace std;

class arrays

{

public:

void massout(int\* m, int n) {

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

{

cout << m[i] << '\t';

}

cout << endl;

}

void sorting(int\* m, int n) {

sort(m, m + n);

}

void revSorting(int\* m, int n) {

sort(m, m + n);

reverse(m, m + n);

}

void bubble(int\* m, int n, chrono::duration<double>\* time, int x) //прямой обмен

{

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

auto start = chrono::high\_resolution\_clock::now();

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

for (int j = n - 1; j > i; j--)

{

if (tmp[j] < tmp[j - 1]) swap(tmp[j], tmp[j - 1]);

}

}

auto end = chrono::high\_resolution\_clock::now();

time[x] = (end - start) \* 1000;

}

void Examp\_bubble(int\* m, int n) //прямой обмен для примера

{

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

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

for (int j = n - 1; j > i; j--)

{

if (tmp[j] < tmp[j - 1]) swap(tmp[j], tmp[j - 1]);

}

}

massout(tmp, n);

}

void shakerNew(int\* m, int n, chrono::duration<double>\* time, int x) //шейкер

{

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

auto start = chrono::high\_resolution\_clock::now();

int left = 0, right = n - 1;

int k = right;

do {

for (int j = right; j > left; j--) {

if (tmp[j] < tmp[j - 1]) {

swap(tmp[j], tmp[j - 1]);

k = j;

}

}

left=k+1;

for (int j = left; j < right; j++) {

if (tmp[j] > tmp[j + 1]) {

swap(tmp[j], tmp[j + 1]);

k = j;

}

}

right=k-1;

} while (left <= right);

auto end = chrono::high\_resolution\_clock::now();

time[x] = (end - start) \* 1000;

}

void shaker(int\* m, int n, chrono::duration<double>\* time, int x) //шейкер

{

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

auto start = chrono::high\_resolution\_clock::now();

int left = 0, right = n - 1;

int k = right;

do {

for (int j = right; j > left; j--) {

if (tmp[j] < tmp[j - 1]) {

swap(tmp[j], tmp[j - 1]);

}

}

left++;

for (int j = left; j < right; j++) {

if (tmp[j] > tmp[j + 1]) {

swap(tmp[j], tmp[j + 1]);

}

}

right--;

} while (left <= right);

auto end = chrono::high\_resolution\_clock::now();

time[x] = (end - start) \* 1000;

}

void Examp\_shakerNew(int\* m, int n) //шейкер для примера

{

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

int left = 0, right = n - 1;

int k = right;

do {

for (int j = right; j > left; j--) {

if (tmp[j] < tmp[j - 1]) {

swap(tmp[j], tmp[j - 1]);

k = j;

}

}

left=k+1;

for (int j = left; j < right; j++) {

if (tmp[j] > tmp[j + 1]) {

swap(tmp[j], tmp[j + 1]);

k = j;

}

}

right=k-1;

} while (left <= right);

massout(tmp, n);

}

void Examp\_shaker(int\* m, int n) //шейкер для примера

{

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

int left = 0, right = n - 1;

int k = right;

do {

for (int j = right; j > left; j--) {

if (tmp[j] < tmp[j - 1]) {

swap(tmp[j], tmp[j - 1]);

}

}

left++;

for (int j = left; j < right; j++) {

if (tmp[j] > tmp[j + 1]) {

swap(tmp[j], tmp[j + 1]);

}

}

right--;

} while (left <= right);

massout(tmp, n);

}

void quicksortPre(int\* m, int left, int right) { // быстрая сортировка

int i = left, j = right, key = m[(left + right) / 2];

{

do {

while (m[i] < key) i++;

while (m[j] > key) j--;

if (i <= j)

{

swap(m[i], m[j]); i++; j--;

}

} while (i <= j);

if (i < right) quicksortPre(m, i, right);

if (j > left) quicksortPre(m, left, j);

}

}

void quickSort(int\* m, int n, chrono::duration<double>\* time, int x) {

auto start = chrono::high\_resolution\_clock::now();

quicksortPre(m, 0, n - 1);

auto end = chrono::high\_resolution\_clock::now();

time[x] = (end - start) \* 1000;

}

void Examp\_quicksortPre(int\* m, int left, int right) { // быстрая сортировка для примера

int i = left, j = right, key = m[(left + right) / 2];

{

do {

while (m[i] < key) i++;

while (m[j] > key) j--;

if (i <= j)

{

swap(m[i], m[j]); i++; j--;

}

} while (i <= j);

if (i < right) Examp\_quicksortPre(m, i, right);

if (j > left) Examp\_quicksortPre(m, left, j);

}

}

void Examp\_quickSort(int\* m, int n) {

int\* tmp = new int[n];

copy(m, m + n, tmp); //копия массива

Examp\_quicksortPre(tmp, 0, n - 1);

massout(tmp, n);

}

};

FileName.cpp

#include <iostream>

#include <ctime>

#include <random>

#include <algorithm>

#include "Source.h"

#include <Windows.h>

using namespace std;

int main() {

HANDLE hConsole = GetStdHandle(STD\_OUTPUT\_HANDLE);

SetConsoleTextAttribute(hConsole, FOREGROUND\_GREEN | FOREGROUND\_INTENSITY);

setlocale(LC\_ALL, "RU");

chrono::duration<double>\* duration = new chrono::duration<double>[9];

// srand(time(0));

int n\_exmp = 10;

int\* exampArr = new int[n\_exmp];//массив для иллюстраци сортировки

int i = -1;

for (int i = 0; i < n\_exmp; i++)

{

exampArr[i] = rand() % 100;

}

arrays obj;

//cout << "Элементы первого массива:" << endl;

//obj.massout(arrRand, elems);

//obj.sorting(arrSorted, elems);

//cout << "Элементы второго массива:" << endl;

//obj.massout(arrSorted, elems);

//obj.rewSorting(arrRevSorted, elems);

//cout << "Элементы третьего массива:" << endl;

//obj.massout(arrRevSorted, elems);

cout << "Массив для иллюстрации сортировки: " << endl;

obj.massout(exampArr, n\_exmp);

cout << endl << "Пример сортировки всеми методами: " << endl;

cout << "Пузырьком: " << '\t' << '\t';

obj.Examp\_bubble(exampArr, n\_exmp);

cout << "Шейкером: " << '\t' << '\t';

obj.Examp\_shaker(exampArr, n\_exmp);

cout << "Быстрой сортировкой: " << '\t';

obj.Examp\_quickSort(exampArr, n\_exmp);

cout << endl << endl << endl;

int choose = 0;

int elems = 0;

cout << "Введите количество элементов массива: ";

cin >> elems;

try

{

int\* check = new int[elems];

}

catch (const std::exception&)

{

cout << "Неверно задан размер массива." << endl;

return 0;

}

int\* arrRand = new int[elems]; // Случайный массив

int\* arrSorted = new int[elems];// Отсортированный массив

int\* arrRevSorted = new int[elems];// Отсортированный в обратном порядке массив

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

{

arrRand[i] = rand() % 100; // заполняем элементы массива

arrSorted[i] = rand() % 100;

arrRevSorted[i] = rand() % 100;

}

obj.sorting(arrSorted, elems);

obj.revSorting(arrRevSorted, elems);

cout << endl << endl;

obj.bubble(arrRand, elems, duration, 0);

//cout << fixed << setprecision(10) << duration[0].count();

obj.bubble(arrSorted, elems, duration, 1);

//cout << fixed << setprecision(10) << duration[1].count();

obj.bubble(arrRevSorted, elems, duration, 2);

//cout << fixed << setprecision(10) << duration[2].count();

obj.shaker(arrRand, elems, duration, 3);

obj.shaker(arrSorted, elems, duration, 4);

obj.shaker(arrRevSorted, elems, duration, 5);

obj.quickSort(arrRand, elems, duration, 6);

obj.quickSort(arrSorted, elems, duration, 7);

obj.quickSort(arrRevSorted, elems, duration, 8);

cout << endl;

cout << " \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_" << endl;

cout << "| | Обычный массив | Отсортированный массив | Обратно отсортированный массив |" << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout << "| | | | |" << endl;

cout << "| Сортировка пузырьком |" << '\t' << fixed << setprecision(7) << duration[0].count() << "ms " << fixed << setprecision(7) << duration[1].count() << "ms " << fixed << setprecision(7) << duration[2].count() << "ms " << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout << "| | | | |" << endl;

cout << "| Шейкерная сортировка |" << '\t' << fixed << setprecision(7) << duration[3].count() << "ms " << fixed << setprecision(7) << duration[4].count() << "ms " << fixed << setprecision(7) << duration[5].count() << "ms " << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

cout << "| | | | |" << endl;

cout << "| Быстрая сортировка |" << '\t' << fixed << setprecision(7) << duration[6].count() << "ms " << fixed << setprecision(7) << duration[7].count() << "ms " << fixed << setprecision(7) << duration[8].count() << "ms " << endl;

cout << "|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|" << endl;

system("Pause");

return 0;

}

1. Определение класса arrays
   1. **massout(int\* m, int n)**: Выводит элементы массива размера **n** на экран.
   2. **sorting(int\* m, int n)**: Сортирует массив размера **n** в порядке возрастания.
   3. **rewSorting(int\* m, int n)**: Сортирует массив размера **n** в порядке убывания.
   4. **bubble(int\* m, int n)**: Сортирует массив размера **n** с помощью сортировки пузырьком и выводит время выполнения.

Этот метод реализует сортировку пузырьком, один из простейших алгоритмов сортировки. Алгоритм проходит по массиву множество раз, сравнивая соседние элементы и меняя их местами, если они находятся в неправильном порядке (текущий элемент меньше предыдущего). Каждый проход по массиву сортирует хотя бы один элемент на своем месте, так что после n проходов весь массив становится отсортированным.

Время выполнения измеряется с помощью класса **chrono**.

* 1. **shaker(int\* m, int n)**: Этот метод также известен как сортировка перемешиванием. Он является улучшением сортировки пузырьком, обеспечивая более эффективный обмен элементов. Алгоритм состоит из двух проходов по массиву: сначала слева направо, а затем справа налево. На каждом проходе сортировка перемещает наибольший элемент в конец массива, затем наименьший элемент в начало. Это продолжается до тех пор, пока весь массив не будет отсортирован.

Время выполнения также измеряется с помощью класса **chrono**. (Плохая реализация)

* 1. void shakerNew(int\* m, int n, chrono::duration<double>\* time, int x):

Нормальная реализация метода shaker

* 1. **quicksortPre(int\* m, int left, int right)**: Вспомогательная функция для реализации быстрой сортировки.

Инициализация переменных:

**i** - указатель на левую часть массива.

**j** - указатель на правую часть массива.

**key** - значение элемента, выбранного в качестве опорного (pivot). В данном коде опорным элементом выбирается элемент посередине между **left** и **right**.

Цикл do-while: внутри этого цикла происходит разделение массива на две части: все элементы, меньшие **key**, помещаются слева, а все элементы, большие **key**, помещаются справа. Это достигается путем увеличения **i** до тех пор, пока **m[i]** меньше **key**, и уменьшения **j** до тех пор, пока **m[j]** больше **key**. Если **i <= j**, происходит обмен местами элементов **m[i]** и **m[j]**. Этот процесс продолжается до тех пор, пока **i** не станет больше **j**.После завершения разделения массива на две части, рекурсивно вызываются два новых вызова **quicksort**:

Для правой части массива, если **i < right**.

Для левой части массива, если **j > left**.

* 1. **quickSort(int\* m, int n)**: Сортирует массив размера **n** с помощью быстрой сортировки и выводит время выполнения.

*FIleName.cpp*

1. **Вывод примера работы трёх методов сортировки на небольшом массиве**
2. **Запрос на ввод количества элементов массива**:
   * Пользователь вводит количество элементов “elems” массива с помощью **cin**.
   * Проверка массива на исключение через try,catch
3. **Выделение памяти под массивы**:
   * Создаются указатели на массивы **arrRand**, **arrSorted**, **arrRevSorted** с помощью оператора **new**.
4. **Заполнение массивов случайными числами**:
   * Используется цикл **for**, который заполняет каждый элемент массива случайным числом от 0 до Elems с помощью функции **rand()**.
5. **Создание экземпляра класса arrays**:
   * Создается объект **obj** класса **arrays**, который будет использоваться для сортировки массивов.
   * Сортировка каждого массива тремя методами, сохранение времени работы каждого массив-метода в массив
6. **Табличный вывод работы программы:**
   * Визуализация табличной формы времени работы каждого метода для каждого массива

