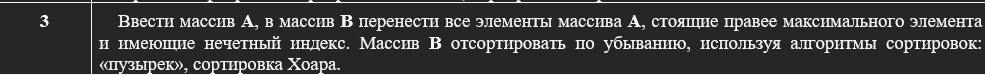
Лабораторная работа 15

По дисциплине «Основы алгоритмизации и программирования»

На тему «Анализ алгоритмов сортировок»

***Задание#1***



*Macros.h*

#include <iostream>

#include <vector>

#include <ctime>

#include <cmath>

#include <iomanip>

#include <stack>

#include <map>

#include <cstring>

#include <string>

#include <chrono>

#include <fstream>

#include <Windows.h>

#include <queue>

#include <unordered\_map>

#define vi cout <<

#define vv cin >>

#define nl cout << "\n";

#define nw cout << "\t";

#define SCOCP1251 SetConsoleOutputCP(1251);

#define SCCP1251 SetConsoleCP(1251);

#define ret return

#define pause system("pause");

using namespace :: std;

*.cpp*

#include "Macros.h"

using namespace std;

// Функция для сортировки массива методом пузырька в порядке убывания

void bubbleSortDescending(int arr[], int n) {

for (int i = 0; i < n - 1; ++i) { // Проходим по каждому элементу массива

for (int j = 0; j < n - 1 - i; ++j) { // Для каждой пары элементов

if (arr[j] < arr[j + 1]) { // Если текущий элемент меньше следующего

// Меняем их местами

int temp = arr[j];

arr[j] = arr[j + 1];

arr[j + 1] = temp;

}

}

}

}

// Функция для быстрой сортировки массива в порядке убывания

void quickSortDescending(int arr[], int low, int high) {

if (low < high) {

int pivot = arr[high]; // Выбираем опорный элемент

int i = (low - 1); // Индекс элемента, меньшего опорного

// Разделяем массив

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

// Если текущий элемент больше или равен опорному

if (arr[j] >= pivot) {

++i; // Увеличиваем индекс меньшего элемента

swap(arr[i], arr[j]); // Меняем местами текущий и меньший элементы

}

}

swap(arr[i + 1], arr[high]); // Меняем местами опорный элемент и элемент после последнего меньшего элемента

// Рекурсивно сортируем две части массива

quickSortDescending(arr, low, i);

quickSortDescending(arr, i + 2, high);

}

}

int main() {

SCCP1251

SCOCP1251

srand(time(NULL));

int size; // Размер массива A

vi "Введите размер массива: ";

vv size;

// Создаем массив A и заполняем его случайными числами

int\* A = new int[size];

vi "Массив A: ";

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

A[i] = rand() % 100; // Заполняем массив случайными числами от 0 до 99

vi A[i] << " ";

}

nl

// Находим максимальный элемент и его индекс

int maxElement = A[0], maxIndex = 0;

for (int i = 1; i < size; i++) {

if (A[i] > maxElement) {

maxElement = A[i];

maxIndex = i;

}

}

// Создаем массив B и копируем в него элементы из A, стоящие правее максимального элемента и имеющие нечетный индекс

vector<int> B;

for (int i = maxIndex + 1; i < size; i++) {

B.push\_back(A[i]);

}

nl

// Измеряем время выполнения пузырьковой сортировки

clock\_t start = clock(); // Запоминаем время начала сортировки

bubbleSortDescending(B.data(), B.size()); // Выполняем пузырьковую сортировку

clock\_t end = clock(); // Запоминаем время окончания сортировки

double bubbleSortTime = double(end - start) / CLOCKS\_PER\_SEC; // Вычисляем время выполнения

vi "Время пузырьковой сортировки: " << bubbleSortTime << " секунд"; nl

// Измеряем время выполнения быстрой сортировки

start = clock(); // Запоминаем время начала сортировки

quickSortDescending(B.data(), 0, B.size() - 1); // Выполняем быструю сортировку

end = clock(); // Запоминаем время окончания сортировки

double quickSortTime = double(end - start) / CLOCKS\_PER\_SEC; // Вычисляем время выполнения

vi "Время быстрой сортировки: " << quickSortTime << " секунд"; nl

for (int i = 0; i < B.size(); i++)

{

vi B[i]; nw

}

nl

// Освобождаем память

delete[] A;

pause

ret 0;

}

