Белорусский государственный технологический университет

Факультет информационных технологий

Кафедра программной инженерии

 Лабораторная работа 15

По дисциплине «Основы алгоритмизации и программирования»

На тему «Анализ алгоритмов сортировок»

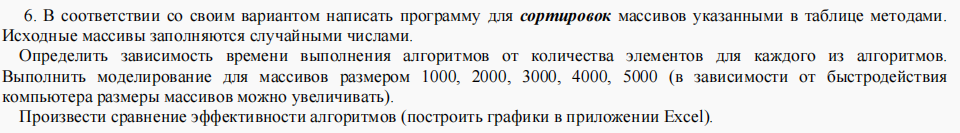
Выполнил:

Студент 1 курса 6 группы

Романов Игорь Вячеславович

Преподаватель: асс. Андронова М.В.

2024, Минск



Вариант 12



#include <iostream>

#include <ctime> // Для работы с временем

#include <cstdlib> // Для функции rand()

#include <Windows.h>

using namespace std;

// Функция для пузырьковой сортировки

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

for (int i = 0; i < n - 1; ++i) { // Проходим по всему массиву

for (int j = 0; j < n - 1 - i; ++j) { // Внутренний цикл для пары элементов

if (arr[j] > arr[j + 1]) { // Если элемент слева больше элемента справа

// Обмен значениями

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

// Функция для слияния двух подмассивов

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1; // Размер левого подмассива

int n2 = r - m; // Размер правого подмассива

// Создаем временные массивы для хранения значений

int\* L = new int[n1];

int\* R = new int[n2];

// Копируем данные из оригинального массива во временные массивы

for (int i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

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

// Индексы для прохода по временным массивам

int i = 0, j = 0, k = l;

// Объединяем временные массивы обратно в основной массив

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

++i;

}

else {

arr[k] = R[j];

++j;

}

++k;

}

// Копируем оставшиеся элементы из L[], если есть

while (i < n1) {

arr[k] = L[i];

++i;

++k;

}

// Копируем оставшиеся элементы из R[], если есть

while (j < n2) {

arr[k] = R[j];

++j;

++k;

}

// Освобождаем временные массивы

delete[] L;

delete[] R;

}

// Функция для сортировки слиянием

void mergeSort(int arr[], int l, int r) {

if (l < r) {

int m = l + (r - l) / 2; // Находим середину массива

// Рекурсивно сортируем первую и вторую половины

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Сливаем отсортированные половины

merge(arr, l, m, r);

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int sizes[] = { 1000, 2000, 3000, 4000, 5000 }; // Размеры массивов для моделирования

for (int size : sizes) { // Проходим по каждому размеру массива

// Создаем массив A и заполняем случайными числами

int\* A = new int[size];

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

A[i] = rand() % 10000; // Заполняем массив случайными числами от 0 до 9999

}

// Создаем массив B и копируем элементы с четными индексами из A

int\* B = new int[size / 2];

int index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

// Время выполнения пузырьковой сортировки

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

bubbleSort(B, size / 2); // Выполняем пузырьковую сортировку

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

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

cout << "Размер массива: " << size << ", Время пузырьковой сортировки: " << bubbleSortTime << " секунд" << endl;

// Копируем элементы с четными индексами снова для сортировки слиянием

index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

// Время выполнения сортировки слиянием

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

mergeSort(B, 0, size / 2 - 1); // Выполняем сортировку слиянием

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

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

cout << "Размер массива: " << size << ", Время сортировки слиянием: " << mergeSortTime << " секунд" << endl;

// Освобождаем память

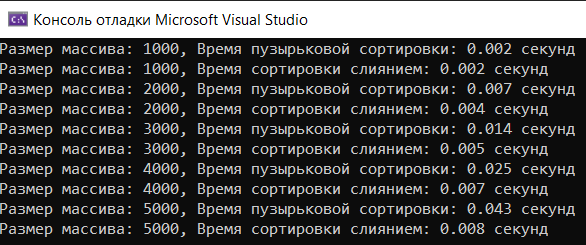
delete[] A;

delete[] B;

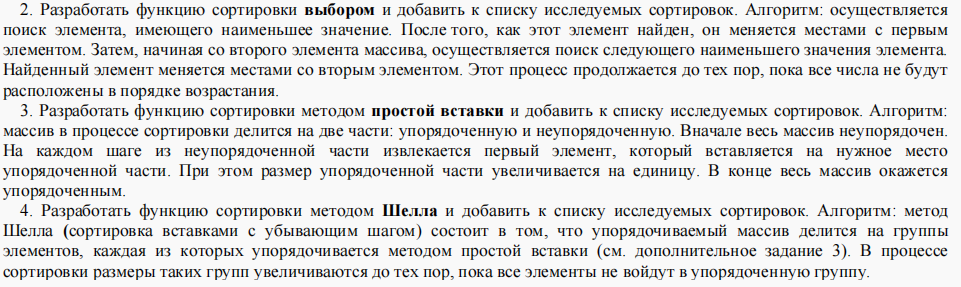
}

return 0;

}



Доп. задания



Все 3 задания выполнены в одном проекте

#include <iostream>

#include <ctime> // Для работы с временем

#include <cstdlib> // Для функции rand()

#include <Windows.h>

using namespace std;

// Функция для пузырьковой сортировки

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

// Внешний цикл проходит по всем элементам массива

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

// Внутренний цикл сравнивает пары элементов и меняет их местами, если они в неправильном порядке

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

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

// Обмен значениями

int temp = arr[j];

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

arr[j + 1] = temp;

}

}

}

}

// Функция для слияния двух подмассивов

void merge(int arr[], int l, int m, int r) {

int n1 = m - l + 1; // Размер левого подмассива

int n2 = r - m; // Размер правого подмассива

// Создаем временные массивы для хранения значений

int\* L = new int[n1];

int\* R = new int[n2];

// Копируем данные из оригинального массива во временные массивы

for (int i = 0; i < n1; ++i)

L[i] = arr[l + i];

for (int j = 0; j < n2; ++j)

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

// Индексы для прохода по временным массивам

int i = 0, j = 0, k = l;

// Объединяем временные массивы обратно в основной массив

while (i < n1 && j < n2) {

if (L[i] <= R[j]) {

arr[k] = L[i];

++i;

}

else {

arr[k] = R[j];

++j;

}

++k;

}

// Копируем оставшиеся элементы из L[], если есть

while (i < n1) {

arr[k] = L[i];

++i;

++k;

}

// Копируем оставшиеся элементы из R[], если есть

while (j < n2) {

arr[k] = R[j];

++j;

++k;

}

// Освобождаем временные массивы

delete[] L;

delete[] R;

}

// Функция для сортировки слиянием

void mergeSort(int arr[], int l, int r) {

if (l < r) {

// Находим середину массива

int m = l + (r - l) / 2;

// Рекурсивно сортируем первую и вторую половины

mergeSort(arr, l, m);

mergeSort(arr, m + 1, r);

// Сливаем отсортированные половины

merge(arr, l, m, r);

}

}

// Функция для сортировки выбором

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

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

// Ищем минимальный элемент в неотсортированной части массива

int minIndex = i;

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

if (arr[j] < arr[minIndex]) {

minIndex = j;

}

}

// Меняем найденный минимальный элемент с первым элементом

int temp = arr[minIndex];

arr[minIndex] = arr[i];

arr[i] = temp;

}

}

// Функция для сортировки методом вставки

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

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

int key = arr[i]; // Элемент для вставки

int j = i - 1;

// Сдвигаем элементы массива, чтобы освободить место для вставки

while (j >= 0 && arr[j] > key) {

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

--j;

}

arr[j + 1] = key; // Вставляем элемент

}

}

// Функция для сортировки методом Шелла

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

// Начальный шаг делим на 2 каждый раз

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

// Сортируем элементы с заданным шагом

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

int temp = arr[i];

int j;

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

arr[j] = arr[j - gap];

}

arr[j] = temp;

}

}

}

int main() {

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

// Массив размеров для моделирования

int sizes[] = { 1000, 2000, 3000, 4000, 5000 };

// Проходим по каждому размеру массива

for (int size : sizes) {

// Создаем массив A и заполняем случайными числами

int\* A = new int[size];

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

A[i] = rand() % 10000; // Заполняем массив случайными числами от 0 до 9999

}

// Создаем массив B и копируем элементы с четными индексами из A

int\* B = new int[size / 2];

int index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

clock\_t start, end;

// Пузырьковая сортировка

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

bubbleSort(B, size / 2); // Выполняем пузырьковую сортировку

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

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

cout << "Размер массива: " << size << ", Время пузырьковой сортировки: " << bubbleSortTime << " секунд" << endl;

// Копируем элементы снова для сортировки слиянием

index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

// Сортировка слиянием

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

mergeSort(B, 0, size / 2 - 1); // Выполняем сортировку слиянием

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

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

cout << "Размер массива: " << size << ", Время сортировки слиянием: " << mergeSortTime << " секунд" << endl;

// Копируем элементы снова для сортировки выбором

index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

// Сортировка выбором

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

selectionSort(B, size / 2); // Выполняем сортировку выбором

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

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

cout << "Размер массива: " << size << ", Время сортировки выбором: " << selectionSortTime << " секунд" << endl;

// Копируем элементы снова для сортировки методом вставки

index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

// Сортировка методом вставки

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

insertionSort(B, size / 2); // Выполняем сортировку методом вставки

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

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

cout << "Размер массива: " << size << ", Время сортировки методом вставки: " << insertionSortTime << " секунд" << endl;

// Копируем элементы снова для сортировки методом Шелла

index = 0;

for (int i = 0; i < size; i += 2) {

B[index++] = A[i];

}

// Сортировка методом Шелла

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

shellSort(B, size / 2); // Выполняем сортировку методом Шелла

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

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

cout << "Размер массива: " << size << ", Время сортировки методом Шелла: " << shellSortTime << " секунд" << endl;

// Освобождаем память, выделенную под массивы

delete[] A;

delete[] B;

cout << "----------------------------------------" << endl;

}

return 0;

}

