ФЕДЕРАЛЬНОЕ Государственное АВТОНОМНОЕ образовательное учреждение

высшего образования

**БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ НАЦИОНАЛЬНЫЙ**

**ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ**

(НИУ «БелГУ»)

ИНСТИТУТ ИНЖЕНЕРНЫХ И ЦИФРОВЫХ ТЕХНОЛОГИЙ

Кафедра математического и программного обеспечения информационных систем

**Отчет по лабораторной работе №3**

**по дисциплине: «Структуры и алгоритмы компьютерной обработки данных»**

**Тема работы «Сравнительный анализ методов сортировки»**

студента очного отделения

2 курса 12001801 группы

Скрипниченко Илья Игоревич

Проверил(а):

Курлов Василий Васильевич

Белгород 2019

**Лабораторная работа №3**

**Цель работы:** изучение методов сортировки массивов и приобретение навыков в проведении сравнительного анализа различных методов сортировки.

**Листинг программы**

#include "pch.h"

#include <iostream>

void fill\_arr(int\* arr, short sz); //Заполнение массива

void get\_disordered\_arr(const int\* arr1, int\* arr2, const short sz); //Заполнения неупорядоченного массива

void swap\_el(int\* arr, int i); //Обмен элементами

void get\_order(const int\* arr1, int\* arr2, short sz); //Заполнение "упорядоченного" массива этими значениями и их упорядочивание

void get\_reorder(const int\* arr1, int\* arr2, short sz); //Заполнение "упорядоченного в обратном порядке" массива значениями "упорядоченного массива" и их упорядочивание в обратном порядке

void show(const int\* arr, short sz); //Вывести элементы массива

unsigned insertion\_sort(int\* arr, short sz); //Сортировка вставками

unsigned sort\_by\_selection(int\* arr, short sz); //Сортировка выбором

unsigned bubble\_sort(int\* arr, short sz); //Сортировка обменом

unsigned better\_bubble\_sort1(int\* arr, short sz); //Улучшенная сортировка обменом 1

unsigned better\_bubble\_sort2(int\* arr, short sz); //Улучшенная сортировка обменом 2

unsigned shell\_sort(int\* arr, short sz); //Сортировка Шелла

unsigned hoara\_sort(int\* arr, int first, int last); //Сортировка Хоара

unsigned make\_heap(int\* arr, int root, int bottom); //Построение пирамиды

unsigned heap\_sort(int\* arr, short sz); //Пирамидальная сортировка

int main()

{

    using namespace std;

    setlocale(0, "");

    cout << "Программа анализа эффективности различных сортировок\n";

    cout << "массивов с разным количеством элементов\n";

    cout << "Нажмите /enter/ чтобы начать";

    while (cin.get() != 'q')

    {

        cout << "\nВведите размер массива --> ";

        short size;

        cin >> size;

        int\* default\_arr = new int[size];

        fill\_arr(default\_arr, size);

        cin.clear();

        /////////////////Анализ сортировки вставками/////////////////

        cout << "Анализ сортировки вставками\n";

        int\* ordered\_arr = new int[size];

        int\* reordered\_arr = new int[size];

        int\* disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "Упорядоченный массив размера " << size << ":\n";

        cout << "Сортировка вставками: " << insertion\_sort(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Сортировка вставками: " << insertion\_sort(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Сортировка вставками: " << insertion\_sort(disordered\_arr, size);

        show (disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ сортировки выбором/////////////////

        cout << "\n\n\nАнализ сортировки выбором";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Сортировка выбором: " << sort\_by\_selection(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Сортировка выбором: " << sort\_by\_selection(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Сортировка выбором: " << sort\_by\_selection(disordered\_arr, size);

        show (disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ сортировки обменом/////////////////

        cout << "\n\n\nАнализ сортировки обменом";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Сортировка обменом: " << bubble\_sort(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Сортировка обменом: " << bubble\_sort(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Сортировка обменом: " << bubble\_sort(disordered\_arr, size);

        show (disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ улучшенной сортировки обменом 1/////////////////

        cout << "\n\n\nАнализ улучшенной сортировки обменом 1";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Улучшенная сортировка обменом 1: " << better\_bubble\_sort1(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Улучшенная сортировка обменом 1: " << better\_bubble\_sort1(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Улучшенная сортировка обменом 1: " << better\_bubble\_sort1(disordered\_arr, size);

        show (disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ улучшенной сортировки обменом 2/////////////////

        cout << "\n\n\nАнализ улучшенной сортировки обменом 2";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Улучшенная сортировка обменом 2: " << better\_bubble\_sort2(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Улучшенная сортировка обменом 2: " << better\_bubble\_sort2(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Улучшенная сортировка обменом 2: " << better\_bubble\_sort2(disordered\_arr, size) << endl;

        show(disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ сортировки Шелла/////////////////

        cout << "\n\n\nАнализ сортировки Шелла";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Сортировка Шелла: " << shell\_sort(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Сортировка Шелла: " << shell\_sort(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Сортировка Шелла: " << shell\_sort(disordered\_arr, size) << endl;

        show (disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ сортировки Хоара/////////////////

        cout << "\n\n\nАнализ сортировки Хоара";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Сортировка Хоара: " << hoara\_sort(ordered\_arr, 0, size - 1);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Сортировка Хоара: " << hoara\_sort(reordered\_arr, 0, size - 1);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Сортировка Хоара: " << hoara\_sort(disordered\_arr, 0, size - 1) << endl;

        show (disordered\_arr, size);

        cout << endl;

        show (ordered\_arr, size);

        cout << endl;

        show (reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        /////////////////Анализ пирамидальной сортировки/////////////////

        cout << "\n\n\nАнализ пирамидальной сортировки";

        ordered\_arr = new int[size];

        reordered\_arr = new int[size];

        disordered\_arr = new int[size];

        get\_disordered\_arr(default\_arr, disordered\_arr, size);

        get\_order(disordered\_arr, ordered\_arr, size);

        get\_reorder(ordered\_arr, reordered\_arr, size);

        cout << "\nУпорядоченный массив размера " << size << ":\n";

        cout << "Пирамидальная сортировка: " << heap\_sort(ordered\_arr, size);

        cout << "\nУпорядоченный в обратном порядке массив размера " << size << ":\n";

        cout << "Пирамидальная сортировка: " << heap\_sort(reordered\_arr, size);

        cout << "\nНеупорядоченный массив размера " << size << ":\n";

        cout << "Пирамидальная сортировка: " << heap\_sort(disordered\_arr, size) << endl;

        show(disordered\_arr, size);

        cout << endl;

        show(ordered\_arr, size);

        cout << endl;

        show(reordered\_arr, size);

        cout << endl;

        delete[] ordered\_arr;

        delete[] reordered\_arr;

        delete[] disordered\_arr;

        delete[] default\_arr;

        cout << "Введите любой символ чтобы проанализировать другой массив (q - выход) --> ";

        cin.get();

    }

    return 0;

}

void fill\_arr(int\* arr, const short sz)

{

    using namespace std;

    setlocale(0, "");

    cout << "\nВведите значения элементов массива:\n";

    for (int i = 0; i < sz; i++)

    {

        cout << "[" << i + 1 << "] --> ";

        cin >> arr[i];

    }

    cout << endl;

}

void get\_disordered\_arr(const int\* arr1, int\* arr2, const short sz)

{

    for (int i = 0; i < sz; i++)

        arr2[i] = arr1[i];

}

void swap\_el(int\* arr, int i)

{

    int help;

    help = arr[i - 1];

    arr[i - 1] = arr[i];

    arr[i] = help;

}

void get\_order(const int\* arr1, int\* arr2, const short sz)

{

    for (int i = 0; i < sz; i++)

    {

        arr2[i] = arr1[i];

    }

    for (int i = 1; i < sz; i++)

    {

        for (int j = i; j > 0 && arr2[j - 1] > arr2[j]; j--)

            swap\_el(arr2, j);

    }

}

void get\_reorder(const int\* arr1, int\* arr2, short sz)

{

    int temp;

    for (int i = 0; i < sz; i++)

    {

        arr2[i] = arr1[i];

    }

    for (int i = 0; i < sz - 1; i++)

    {

        for (int j = i + 1; j < sz; j++)

        {

            if (arr2[i] < arr2[j])

            {

                temp = arr2[i];

                arr2[i] = arr2[j];

                arr2[j] = temp;

            }

        }

    }

}

void show(const int\* arr, short sz)

{

    std::cout << std::endl;

    for (int i = 0; i < sz; i++)

    {

        std::cout << arr[i] << " - ";

    }

    std::cout << std::endl;

}

unsigned insertion\_sort(int\* arr, short sz)

{

    unsigned count = 0;

    for (int i = 1; i < sz; i++)

    {

        if (arr[i] >= arr[i - 1])

            count++;

        for (int j = i; j > 0 && arr[j - 1] > arr[j]; j--, count++)

            swap\_el(arr, j);

    }

    return count;

}

unsigned sort\_by\_selection(int\* arr, short sz)

{

    unsigned count = 0;

    int value, key;

    for (int i = 0; i < sz - 1; i++)

    {

        value = arr[i];

        key = i;

        for (int j = i + 1; j < sz; j++)

        {

            count++;

            if (arr[j] < arr[key])

                key = j;

        }

        count++;

        if (key != i)

        {

            arr[i] = arr[key];

            arr[key] = value;

        }

    }

    return count;

}

unsigned bubble\_sort(int\* arr, short sz)

{

    unsigned count = 0;

    for (int i = 0; i < sz - 1; i++)

    {

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

        {

            count++;

            if (arr[j - 1] > arr[j])

            {

                swap\_el(arr, j);

            }

        }

    }

    return count;

}

unsigned better\_bubble\_sort1(int\* arr, short sz)

{

    unsigned count = 0;

    bool check;

    for (int i = 0; i < sz - 1; i++)

    {

        check = false;

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

        {

            count++;

            if (arr[j - 1] > arr[j])

            {

                swap\_el(arr, j);

                check = true;

            }

        }

        if (check == false)

            break;

    }

    return count;

}

unsigned better\_bubble\_sort2(int\* arr, short sz)

{

    unsigned count = 0;

    int last\_swaped\_el\_num;

    for (int i = 0; i < sz - 1; i++)

    {

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

        {

            count++;

            if (arr[j - 1] > arr[j])

            {

                swap\_el(arr, j);

                last\_swaped\_el\_num = j - 1;

            }

            else last\_swaped\_el\_num = j;

        }

        i = last\_swaped\_el\_num;

    }

    return count;

}

unsigned shell\_sort(int\* arr, short sz)

{

    unsigned count = 0;

    int step = sz / 2;

    while (step > 0)

    {

        for (int i = 0; i < (sz - step); i++)

        {

            if (arr[i] < arr[i + step])

                count++;

            for (int j = i; j >= 0 && arr[j] > arr[j + step]; j--, count++)

            {

                int temp = arr[j];

                arr[j] = arr[j + step];

                arr[j + step] = temp;

            }

        }

        step /= 2;

    }

    return count;

}

unsigned hoara\_sort(int\* arr, int first, int last)

{

    int i = first, j = last;

    int tmp;

    int middle = arr[(first + last) / 2];

    unsigned count = 0;

    do {

        while (arr[i] < middle)

        {

            i++;

            count++;

        }

        while (arr[j] > middle)

        {

            j--;

            count++;

        }

        if (i <= j)

        {

            count++;

            if (i < j)

            {

                tmp = arr[i];

                arr[i] = arr[j];

                arr[j] = tmp;

            }

            i++;

            j--;

        }

    } while (i <= j);

    count++;

    if (i < last)

        hoara\_sort(arr, i, last);

    count++;

    if (first < j)

        hoara\_sort(arr, first, j);

    return count;

}

unsigned make\_heap(int\* arr, int root, int bottom)

{

    unsigned count = 0;

    int max\_child;

    bool finish = false;

    while ((root \* 2 <= bottom) && (!finish))

    {

        if (root \* 2 == bottom)

            max\_child = root \* 2;

        else

            if (arr[root \* 2] > arr[root \* 2 + 1])

                max\_child = root \* 2;

            else  max\_child = root \* 2 + 1;

        count++;

        if (arr[root] < arr[max\_child])

        {

            int tmp = arr[root];

            arr[root] = arr[max\_child];

            arr[max\_child] = tmp;

            root = max\_child;

        }

        else finish = true;

    }

    return count;

}

unsigned heap\_sort(int\* arr, short sz)

{

    unsigned count = 0;

    for (int i = (sz / 2) - 1; i >= 0; i--)

        count += make\_heap(arr, i, sz - 1);

    for (int i = sz - 1; i >= 1; i--, count++)

    {

        int tmp = arr[0];

        arr[0] = arr[i];

        arr[i] = tmp;

        count += make\_heap(arr, 0, i - 1);

    }

    return count;

}

**Результаты работы программы**

Таблица 1. Упорядоченный массив

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Сортировка** | **Количество элементов в массиве** | | | | | | | | |
| **5** | **10** | **15** | **20** | **25** | **30** | **35** | **40** | **45** |
| **Вставками** | 4 | 9 | 14 | 19 | 24 | 29 | 34 | 39 | 44 |
| **Выбором** | 14 | 54 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| **Обменом** | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| **Обменом 1** | 4 | 9 | 14 | 19 | 24 | 29 | 34 | 39 | 44 |
| **Обменом 2** | 6 | 25 | 56 | 100 | 156 | 225 | 306 | 400 | 506 |
| **Шелла** | 7 | 22 | 33 | 62 | 78 | 93 | 140 | 160 | 176 |
| **Хоара** | 7 | 12 | 17 | 22 | 27 | 32 | 36 | 42 | 46 |
| **Пирамидальная** | 14 | 39 | 67 | 99 | 128 | 166 | 203 | 242 | 275 |

Таблица 2. Упорядоченный в обратном порядке массив

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Сортировка** | **Количество элементов в массиве** | | | | | | | | |
| **5** | **10** | **15** | **20** | **25** | **30** | **35** | **40** | **45** |
| **Вставками** | 10 | 45 | 104 | 190 | 300 | 434 | 592 | 778 | 982 |
| **Выбором** | 14 | 54 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| **Обменом** | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| **Обменом 1** | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| **Обменом 2** | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| **Шелла** | 8 | 24 | 41 | 66 | 81 | 109 | 144 | 169 | 189 |
| **Хоара** | 5 | 7 | 10 | 12 | 15 | 17 | 20 | 22 | 25 |
| **Пирамидальная** | 12 | 35 | 60 | 91 | 119 | 150 | 181 | 216 | 250 |

Таблица 3. Неупорядоченный массив

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Сортировка** | **Количество элементов в массиве** | | | | | | | | |
| **5** | **10** | **15** | **20** | **25** | **30** | **35** | **40** | **45** |
| **Вставками** | 7 | 35 | 79 | 129 | 166 | 242 | 352 | 522 | 662 |
| **Выбором** | 14 | 64 | 119 | 209 | 324 | 464 | 629 | 819 | 1034 |
| **Обменом** | 10 | 45 | 105 | 190 | 300 | 435 | 595 | 780 | 990 |
| **Обменом 1** | 9 | 39 | 105 | 175 | 285 | 435 | 567 | 780 | 954 |
| **Обменом 2** | 8 | 39 | 94 | 181 | 288 | 435 | 542 | 780 | 990 |
| **Шелла** | 9 | 24 | 44 | 75 | 84 | 132 | 148 | 173 | 224 |
| **Хоара** | 6 | 8 | 12 | 19 | 23 | 30 | 27 | 41 | 45 |
| **Пирамидальная** | 13 | 38 | 63 | 97 | 127 | 162 | 196 | 222 | 274 |

**Графики зависимостей ФВС**

вставками, обменом, обменом 1, обменом 2