Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное учреждение высшего образования

«Пермский национальный исследовательский политехнический университет»

ПНИПУ

**Лабораторная работа**

**«Сортировки»**

Выполнил:

студент группы РИС-23-2б

Ившин Максим Сергеевич

Проверила:

доцент кафедры ИТАС

О.А. Полякова

2024 г.

**Разработка алгоритма**

**Постановка задачи:**

Реализовать сортировки: блочную, подсчетом, Хоара, слиянием, натуральным слиянием, Шелла, многофазную, быструю.

**Анализ задачи:**

1. Создадим главный файл для основного кода**.**
2. Для удобства все сортировки будут находиться в отдельных файлах
3. Для использования сортировок в главном файле будем подключать файлы с кодом сортировок для дальнейшего использования

Блочная сортировка

#pragma once

#include <iostream>

#include <vector>

using namespace std;

#include "HoarSort.h"

void BucketSort(vector<int> &vect)

{

    int maxValue = vect[0];

    int minValue = vect[0];

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

    {

        if (maxValue < vect[i])

            maxValue = vect[i];

        else if (minValue > vect[i])

            minValue = vect[i];

    }

    int bucketsCount = maxValue / 10 - minValue / 10 + 1; // вычисление количества корзин

    vector<int> \*buckets = new vector<int>[bucketsCount];

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

    {

        buckets[vect[i] / 10 - minValue / 10].push\_back(vect[i]); // заполнение корзин

    }

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

    {

        sortShell(buckets[i]); // сортировка каждой корзины

    }

    int index = 0; // индекс главного массива

    for (int i = 0; i < bucketsCount; i++) // заполнение главного массива

    {

        for (int j = 0; j < int(buckets[i].size()); j++)

        {

            vect[index++] = buckets[i][j];

        }

    }

    delete[] buckets; // очистка памяти

}

Сортировка подсчетом

#pragma once

#include<vector>

#include<iostream>

using namespace std;

void CountingSort(vector<int>& vect)

{

    int max\_element = vect[0];

    int min\_element = vect[0];

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

    {

        if (max\_element < vect[i]) max\_element = vect[i];

        else if (min\_element > vect[i]) min\_element = vect[i];

    }

    vector<int> temp(max\_element - min\_element + 1);

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

    {

        temp[vect[i] - min\_element]++;

    }

    for (int i = 0; i < max\_element - min\_element + 1; i++)

    {

        if (temp[i])

        {

            cout << i + min\_element << "[" << temp[i] << "] ";

        }

    }

    cout << endl;

    int index = 0;

    for (int i = 0; i < max\_element - min\_element + 1; i++)

    {

        for (int step = 0; step < temp[i]; step++)

        {

            vect[index] = i + min\_element;

            index++;

        }

    }

}

Сортировка хоара

#pragma once

#include <vector>

#include <iostream>

#include "usefulFuncs.h"

using namespace std;

void sortShell(vector<int>& arr)

{

    int d = arr.size() / 2;

    while (d != 0) {

        for (int i = 0; i < arr.size() - d; ++i)

        {

            int sort = i;

            while (sort != 0 && arr[sort] > arr[sort + d])

            {

                int temp = arr[sort];

                arr[sort] = arr[sort + d];

                arr[sort + d] = temp;

                sort--;

            }

        }

        d = d / 2;

    }

}

void HoarSort(vector<int> &vect, int startIndex, int endIndex, bool visualization)

{

    int left, right, support\_element, tmp;

    left = startIndex;

    right = endIndex;

    support\_element = vect[(startIndex + endIndex) / 2];

    do

    {

        while ((vect[left] < support\_element) && (left < endIndex))

            left++;

        while ((support\_element < vect[right]) && (right > startIndex))

            right--;

        if (left <= right)

        {

            if (visualization) cout << vect[left] << " <-> " << vect[right] << endl; // визуализация

            tmp = vect[left];

            vect[left] = vect[right];

            vect[right] = tmp;

            left++;

            right--;

            if (visualization)

            {

                print(vect, vect.size());

                cout << endl;

            }

        }

    } while (left <= right);

    if (startIndex < right)

        HoarSort(vect, startIndex, right, visualization);

    if (left < endIndex)

        HoarSort(vect, left, endIndex, visualization);

}

void HoarSort(vector<int> &vect, bool visual = true)

{

    HoarSort(vect, 0, vect.size() - 1, visual);

}

Сортировка слиянием

#pragma once

#include <iostream>

#include <vector>

using namespace std;

void MergeSort(vector<int>& vect, int startIndex, int endIndex);

void MergeSort(vector<int>& vect)

{

    MergeSort(vect, 0, vect.size());

}

void Merge(vector<int>& vect, int startIndex, int endIndex)

{

    vector<int> temp;

    int start\_first\_part = startIndex;

    int end\_first\_part = (startIndex + endIndex) / 2;

    int start\_second\_part = end\_first\_part;

    while (int(temp.size()) < endIndex - startIndex)

    {

        if (start\_first\_part >= end\_first\_part

            || (start\_second\_part < endIndex

                && vect[start\_second\_part] <= vect[start\_first\_part]))

        {

            temp.push\_back(vect[start\_second\_part]);

            start\_second\_part++;

        }

        else

        {

            temp.push\_back(vect[start\_first\_part]);

            start\_first\_part++;

        }

    }

    for (int i = startIndex; i < endIndex; ++i)

        vect[i] = temp[i - startIndex];

}

void MergeSort(vector<int>& vect, int startIndex, int endIndex)

{

    if ((endIndex - startIndex) < 2)

        {

            cout << "sorted return: "<< vect[startIndex] << endl;

            return;

        }

    if ((endIndex - startIndex) == 2)

    {

        if (vect[startIndex] > vect[startIndex + 1])

        {

            swap(vect[startIndex], vect[startIndex + 1]);

            cout << "sorted return: "<< vect[startIndex] << ' ' << vect[startIndex + 1] << endl;

            return;

        }

    }

    cout << vect[startIndex];

    for (int i = startIndex+1; i < (startIndex + endIndex) / 2; i++)

    {

        cout << ' ' << vect[i];

    }

    cout << " <-> ";

    cout << vect[(startIndex + endIndex) / 2];

    for (int i = (startIndex + endIndex) / 2+1; i < endIndex; i++)

    {

        cout << ' ' << vect[i];

    }

    cout << endl;

    MergeSort(vect, startIndex, (startIndex + endIndex) / 2);

    MergeSort(vect, (startIndex + endIndex) / 2, endIndex);

    Merge(vect, startIndex, endIndex);

}

Многофазная сортировка

#include <vector>

#include <iostream>

using namespace std;

template <typename T>

void print\_mas(T mas, int lenght\_mas)

{

    cout << mas[0];

    for (int i = 1; i < lenght\_mas; i++)

        cout << ' ' << mas[i];

    cout << '\n';

}

void natural\_merge(vector<int>& vect1, vector<int>& vect2)

{

    for (int i = 0; i < vect2.size(); i++)

    {

        vect1.push\_back(vect2[i]);

        for (int j = vect1.size() - 1; j > 0; j--)

        {

            if (vect1[j] < vect1[j - 1])

            {

                swap(vect1[j], vect1[j - 1]);

            }

            else

            {

                break;

            }

        }

    }

}

using namespace std;

pair<int, int> ideal\_series(pair<int, int> last\_count\_series, int count\_all\_series)

{

    int current\_num = last\_count\_series.first + last\_count\_series.second;

    if (count\_all\_series > current\_num)

    {

        ideal\_series({ last\_count\_series.second, current\_num }, count\_all\_series);

    }

    else

    {

        return last\_count\_series;

    }

}

void series\_merge(vector<vector<int>>& series\_1, vector<vector<int>>& series\_2, vector<vector<int>>& series\_3)

{

    if (series\_1.size() == 1 && series\_2.size() == 0 && series\_3.size() == 0)

        return;

    else if (series\_2.size() == 1 && series\_1.size() == 0 && series\_3.size() == 0)

        return;

    else if (series\_3.size() == 1 && series\_1.size() == 0 && series\_2.size() == 0)

        return;

    int index\_1 = series\_1.size() - 1;

    int index\_2 = series\_2.size() - 1;

    while (index\_2 != -1) // пока series\_2 не пуста

    {

        natural\_merge(series\_1[index\_1], series\_2[index\_2]);

        series\_3.push\_back(series\_1[index\_1]);

        series\_1.pop\_back();

        series\_2.pop\_back();

        index\_1--;

        index\_2--;

    }

    series\_merge(series\_3, series\_1, series\_2); // 1-ый, там, где больше серий, 2-ой, там где меньше серий (потом пустой), 3-ий, куда сливаются

}

void multiphase\_merge\_sort(int\* mas, int lenght\_mas)

{

    pair<int, int> ideal\_pair;

    vector<int> new\_series;

    vector<vector<int>> series, series\_1, series\_2, series\_3;

    for (int i = 0; i <= lenght\_mas - 1; i++)

    {

        if (i == lenght\_mas - 1)

        {

            new\_series.push\_back(mas[i]);

            series.push\_back(new\_series);

            new\_series.clear();

        }

        else

        {

            new\_series.push\_back(mas[i]);

            if (mas[i] > mas[i + 1])

            {

                series.push\_back(new\_series);

                new\_series.clear();

            }

        }

    }

    ideal\_pair = ideal\_series({ 0, 1 }, series.size());

    for (int i = 0; i < ideal\_pair.second; i++)

    {

        series\_1.push\_back(series[i]);

    }

    for (int i = 0; i < ideal\_pair.first; i++)

    {

        if (series\_1.size() + i + 1 > series.size())

        {

            series\_2.push\_back({});

        }

        else

        {

            series\_2.push\_back(series[series\_1.size() + i]);

        }

    }

    series\_merge(series\_1, series\_2, series\_3);

    for (int i = 0; i < lenght\_mas; i++)

    {

        mas[i] = series\_1[0][i];

    }

}

int main()

{

    system("chcp 1251 > null");

    cout << "Введите кол-во эл-ов массива: ";

    int temp\_int;

    cin >> temp\_int;

    int\* mas = new int[temp\_int];

    for (int i = 0; i < temp\_int; i++)

    {

        cout << i + 1 << " элемент: ";

        cin >> mas[i];

    }

    multiphase\_merge\_sort(mas, temp\_int);

    cout << "Отсортированный массив: ";

    print\_mas(mas, temp\_int);

}

Сортировка натуральным слиянием

#include <iostream>

#include <vector>

using namespace std;

void natural\_merge(vector<int>& vect1, vector<int>& vect2)

{

    for (int i = 0; i < vect2.size(); i++)

    {

        vect1.push\_back(vect2[i]);

        for (int j = vect1.size() - 1; j > 0; j--)

        {

            if (vect1[j] < vect1[j - 1])

            {

                swap(vect1[j], vect1[j - 1]);

            }

            else

            {

                break;

            }

        }

    }

}

void natural\_merge\_sort(vector <int>& vect)

{

    int size = vect.size();

    vector <vector<int>> series;

    vector<int> new\_series;

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

    {

        if (i == size - 1)

        {

            new\_series.push\_back(vect[i]);

            series.push\_back(new\_series);

            new\_series.clear();

            continue;

        }

        new\_series.push\_back(vect[i]);

        if (vect[i] <= vect[i + 1])

        {

            continue;

        }

        else

        {

            series.push\_back(new\_series);

            new\_series.clear();

        }

    }

    while (series.size() != 1)

    {

        natural\_merge(series[0], series[series.size() - 1]);

        series.pop\_back();

    }

    vect = series[0];

}

int main()

{

    setlocale(LC\_ALL, "RU");

    srand(time(0));

    cout << "Введите размер массива: ";

    int lenght\_mas;

    cin >> lenght\_mas;

    vector<int> vect;

    for (int i = 0; i < lenght\_mas; i++)

    {

        vect.push\_back(rand() % 101);

        cout << vect[i] << ' ';

    }

    cout << endl;

    natural\_merge\_sort(vect);

    cout << vect[0];

    for (int i = 1; i < vect.size(); i++)

    {

        cout << ' ' << vect[i];

    }

    return 0;

}

Сортировка Шелла

#pragma once

#include <vector>

#include <iostream>

using namespace std;

void ShellSort(vector <int> vect)

{

    int j, tmp;

    bool flag;

    for (int step = int(vect.size()) / 2; step > 0; step /= 2)

    {

        flag = true;

        for (int i = step; i < int(vect.size()) / 2; i++)

        {

            tmp = vect[i];

            for (j = i; j >= step && flag; j -= step)

            {

                if (tmp < vect[j - step])

                {

                    vect[j] = vect[j - step];

                }

                else

                {

                    break;

                }

            }

            vect[j] = tmp;

        }

    }

}

Быстрая сортировка

#include <iostream>

#include <vector>

#include <functional>

using namespace std;

template <typename T>

void print\_mas(T mas, int n)

{

    cout << mas[0];

    for (int i = 1; i < n; i++)

        cout << ' ' << mas[i];

    cout << '\n';

}

void FastSort(int\* mas, int size)

{

    int i = 0;

    int j = size - 1;

    int pivot = mas[size / 2];

    do

    {

        while (mas[i] < pivot)

        {

            i++;

        }

        while (mas[j] > pivot)

        {

            j--;

        }

        if (i <= j)

        {

            swap(mas[i], mas[j]);

            i++;

            j--;

        }

    } while (i <= j);

    if (j > 0)

    {

        FastSort(mas, j + 1);

    }

    if (i < size)

    {

        FastSort(&mas[i], size - i);

    }

}

int main()

{

    srand(time(0));

    cout << "Размер массива: ";

    int temp\_int;

    cin >> temp\_int;

    int\* mas = new int[temp\_int];

    cout << "Исходный массив: ";

    for (int i = 0; i < temp\_int; i++)

    {

        mas[i] = rand() % 100 - 50;

        cout << mas[i] << ' ';

    }

    cout << "\nОтсортированный массив: ";

    FastSort(mas, temp\_int);

    print\_mas(mas, temp\_int);

    delete[] mas;

}

**GitHub:** *https://github.com/geroineee/PNRPU*