Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**О Т Ч Ё Т**

**по лабораторной работе №16**

Дисциплина: основы алгоритмизации и программирования

Тема: Внешние сортировки с С++

Вариант 7

Выполнил работу

студент группы РИС-20-1б

Коваленко Н. А.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Цель задачи**

Цель – написать программу для осуществления внешних сортировок

Для достижения поставленной цели необходимо решить следующие задачи:

* Проанализировать алгоритмы сортировок слияния
* Понять, как воспользоваться алгоритмами, в условиях поставленной задачи
* Создать соответствующие функции для реализации алгоритмов внешних сортировок

Постановка задачи

Отсортировать исходные данные

Анализ задачи

1. Определить какие операции должны быть выполнены по заданию:

* Функция сортировки естественного слияния

void mergeSortN(int\* v)

* Функция сортировки сбалансированного слияния

void mergeSortB(int\* v)

1. Для решения задачи используются переменные:

Несколько целочисленных переменных для управления циклами, контроля размеров массивов и контроля длины строк

int size = -1, left = 0;

int l = left;

int j = right;

Переменная типа string для хранения названия города

string name;

Вектор для удобного хранения структур и обращения к ним

vector <City> city(size);

1. Ввод данных осуществляется посредством функции

cin >> size;

Вывод данных осуществляется посредством функции

cout << "Численость населения: ";

1. Для решения задачи будут использованы циклы. Например, цикл для ввода информации в структуры

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

{

cout << "Город: ";

cin >> city[i].name;

cout << "Численость населения: ";

cin >> city[i].population;

}

Код

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

int num = 0;

const int r1 = 20;

const int r2 = 32;

int r3 = 25;

void mergeSortN(int\* v)

{

cout << "Сортировка методом естественного слияния" << endl;

int p = 0;

int i = 0;

int j;

int\*\* a = new int\* [r1];

int\* b = new int[r1];

while (p < r1 - 1)

{

j = 0;

a[i] = new int[r1];

a[i][j] = v[p];

j++;

cout << "(" << v[p];

while (v[p] <= v[p + 1] && p < r1 - 1)

{

a[i][j] = v[p + 1];

cout << " " << v[p + 1];

p++;

j++;

}

b[i] = j;

i++;

p++;

cout << ")";

cout << " ";

if (p == r1 - 1 && v[p - 1] > v[p])

{

j = 0;

a[i] = new int[r1];

a[i][j] = v[p];

cout << "(" << v[p] << ")";

j++;

b[i] = j;

i++;

}

}

cout << endl;

int kolvo\_series = i;

//djfgdngjdfngjkdnfjgkndfgkjdnfkj

int l;

int k;

int mas[r1];

int c = kolvo\_series;

while (c != 1)

{

for (i = 0; i < kolvo\_series / 2; i++)

{

l = 0;

j = 0;

for (p = 0; p < b[i] + b[kolvo\_series - i - 1]; p++)

{

if ((a[i][l] < a[kolvo\_series - i - 1][j] || j >= b[kolvo\_series - i - 1]) && l < b[i])

{

mas[p] = a[i][l];

l++;

}

else if ((a[i][l] > a[kolvo\_series - i - 1][j] || l >= b[i]) && j < b[kolvo\_series - i - 1])

{

mas[p] = a[kolvo\_series - i - 1][j];

j++;

}

else if (a[i][l] == a[kolvo\_series - i - 1][j] && j < b[kolvo\_series - i - 1] && l < b[i])

{

mas[p] = a[i][l];

mas[p + 1] = a[kolvo\_series - i - 1][j];

p++;

l++;

j++;

}

}

c--;

for (k = 0; k < p; k++)

{

a[i][k] = mas[k];

}

b[i] = p;

for (int y = 0; y < c; y++)

{

cout << "( ";

for (int u = 0; u < b[y]; u++)

{

cout << a[y][u] << " ";

}

cout << ") ";

}

cout << endl;

}

if (kolvo\_series % 2 == 1)kolvo\_series = kolvo\_series / 2 + 1;

else kolvo\_series = kolvo\_series / 2;

}

cout << endl;

delete[] a;

}

void mergeSortB(int\* v)

{

cout << "Сортировка методом сбалансированного слияния" << endl;

int mid = r2 / 2; // находим середину сортируемой последовательности

if (r2 % 2 == 1)

mid++;

int h = 1;

int\* c = (int\*)malloc(r2 \* sizeof(int));

int step;

while (h < r2)

{

step = h;

int i = 0; // индекс первого пути

int j = mid; // индекс второго пути

int k = 0; // индекс элемента в результирующей последовательности

while (step <= mid)

{

while ((i < step) && (j < r2) && (j < (mid + step)))

{ // пока не дошли до конца пути

// заполняем следующий элемент формируемой последовательности

// меньшим из двух просматриваемых

if (v[i] < v[j])

{

c[k] = v[i];

i++; k++;

}

else {

c[k] = v[j];

j++; k++;

}

}

while (i < step)

{ // переписываем оставшиеся элементы первого пути (если второй кончился раньше)

c[k] = v[i];

i++; k++;

}

while ((j < (mid + step)) && (j < r2))

{ // переписываем оставшиеся элементы второго пути (если первый кончился раньше)

c[k] = v[j];

j++; k++;

}

step = step + h; // переходим к следующему этапу

}

h = h \* 2;

// Переносим упорядоченную последовательность (промежуточный вариант) в исходный массив

for (i = 0; i < r2; i++)

v[i] = c[i];

}

for (int i = 0; i < r2; i++)

cout << v[i] << " ";

}

vector<vector<int>> SplitToEqualSeries(vector<int> vec)

{

// find available number of elements in serie

int numberOfElements = 2;

while (vec.size() % numberOfElements != 0 && numberOfElements < 1000) numberOfElements++;

vector<vector<int>> res;

for (int i = 0; i < vec.size() / numberOfElements; i++)

{

vector<int> newSerie;

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

{

newSerie.push\_back(vec[i \* numberOfElements + j]);

}

res.push\_back(newSerie);

}

return res;

}

void Fibbonachi(int numberOfSeries, int& size1, int& size2)

{

size1 = 0;

size2 = 1;

while (size1 + size2 < numberOfSeries)

{

int t = size2;

size2 = size2 + size1;

size1 = t;

}

}

void PrintSeries(vector<vector<int>> series)

{

cout << series.size() << " - ";

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

{

cout << "(";

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

{

cout << series[i][j] << " ";

}

cout << ")";

}

cout << endl;

}

void PrintFiles(vector<vector<int>> f1, vector<vector<int>> f2, vector<vector<int>>f3)

{

cout << "F1 : ";

PrintSeries(f1);

cout << "F2 : ";

PrintSeries(f2);

cout << "F3 : ";

PrintSeries(f3);

cout << endl;

}

vector<int> MergeSeries(vector<int> ser1, vector<int> ser2)

{

int i = 0, j = 0;

vector<int> res;

while (i < ser1.size() || j < ser2.size())

{

while (i < ser1.size() && (j == ser2.size() || ser1[i] <= ser2[j]))

{

res.push\_back(ser1[i]);

i++;

}

while (j < ser2.size() && (i == ser1.size() || ser2[j] <= ser1[i]))

{

res.push\_back(ser2[j]);

j++;

}

}

return res;

}

vector<vector<int>> MergeFiles(vector<vector<int>>& f1, vector<vector<int>>& f2)

{

// merge

vector<vector<int>> res;

int min = f1.size() < f2.size() ? f1.size() : f2.size();

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

{

res.push\_back(MergeSeries(f1[i], f2[i]));

}

// delete non-needed elements

f1.erase(f1.begin(), f1.begin() + min);

f2.erase(f2.begin(), f2.begin() + min);

return res;

}

vector<int> DeleteEmptyElements(vector<int> vec)

{

for (int i = vec.size() - 1; i >= 0; i--)

if (vec[i] == INT32\_MAX)

vec.erase(vec.begin() + i);

return vec;

}

vector<vector<int>> SplitToSeries(vector<int> vec)

{

int i = 0;

vector<vector<int>> res;

while (i < vec.size())

{

vector<int> newSerie;

newSerie.push\_back(vec[i]);

i++;

while (i < vec.size() && vec[i - 1] < vec[i])

{

newSerie.push\_back(vec[i]);

i++;

}

res.push\_back(newSerie);

}

return res;

}

vector<int> NaturalMergeSort(vector<int> vec)

{

vector<vector<int>> series = SplitToSeries(vec);

PrintSeries(series);

while (series.size() > 1)

{

series.push\_back(MergeSeries(series[0], series[1]));

series.erase(series.begin(), series.begin() + 2);

PrintSeries(series);

}

return series[0];

}

vector<int> PolyphaseMergeSort(vector<int> vec)

{

vector<vector<int>> series = SplitToEqualSeries(vec);

int size1, size2;

Fibbonachi(series.size(), size1, size2);

vector<vector<int>> F1, F2, F3; // files

cout << "Разделение на равные серии" << endl;

PrintSeries(series);

for (int i = 0; i < size1; i++)

{

F1.push\_back(series[i]);

}

for (int i = size1; i < series.size(); i++)

{

F2.push\_back(series[i]);

}

for (int i = series.size(); i < size1 + size2; i++)

{

vector<int> serie;

for (int j = 0; j < F2[0].size(); j++)

serie.push\_back(INT32\_MAX);

F2.push\_back(serie);

}

cout << "Разделение серий на файлы F1 и F2 (" << size1 << " " << size2 << ")" << endl;

PrintFiles(F1, F2, F3);

// sort elements inside series

cout << "Сортировка элементов внутри файлов" << endl;

for (int i = 0; i < F1.size(); i++)

{

F1[i] = NaturalMergeSort(F1[i]);

}

for (int i = 0; i < F2.size(); i++)

{

F2[i] = NaturalMergeSort(F2[i]);

}

PrintFiles(F1, F2, F3);

// sort, finally

cout << "Слияние файлов" << endl;

int countEmptyFiles = 1;

while (countEmptyFiles < 2)

{

// search empty file

int indOfEmptyFile = 1;

if (F2.size() == 0) indOfEmptyFile = 2;

if (F3.size() == 0) indOfEmptyFile = 3;

// merge

switch (indOfEmptyFile)

{

case 1: F1 = MergeFiles(F2, F3); break;

case 2: F2 = MergeFiles(F1, F3); break;

case 3: F3 = MergeFiles(F1, F2); break;

}

// check for end of cycle

countEmptyFiles = 0;

if (F1.size() == 0) countEmptyFiles++;

if (F2.size() == 0) countEmptyFiles++;

if (F3.size() == 0) countEmptyFiles++;

PrintFiles(F1, F2, F3);

}

if (F1.size() != 0) return DeleteEmptyElements(F1[0]);

if (F2.size() != 0) return DeleteEmptyElements(F2[0]);

return DeleteEmptyElements(F3[0]);

}

int main()

{

int arr[r1];

setlocale(LC\_ALL, "rus");// кириллица

for (int i = 0; i < r1; i++)

{

arr[i] = -100 + rand() % 200;

cout << arr[i] << " ";

}

cout << endl;

mergeSortN(arr);

int arr2[r2];

for (int i = 0; i < r2; i++)

{

arr2[i] = -100 + rand() % 200;

cout << arr2[i] << " ";

}

cout << endl;

mergeSortB(arr2);

cout << endl;

cout << endl;

cout << endl;

srand(time(0));

vector<int> vec3;

for (int i = 0; i < r3; i++)

{

vec3.push\_back(-100 + rand() % 200);

}

vec3 = PolyphaseMergeSort(vec3);

for (int i = 0; i < vec3.size(); i++)

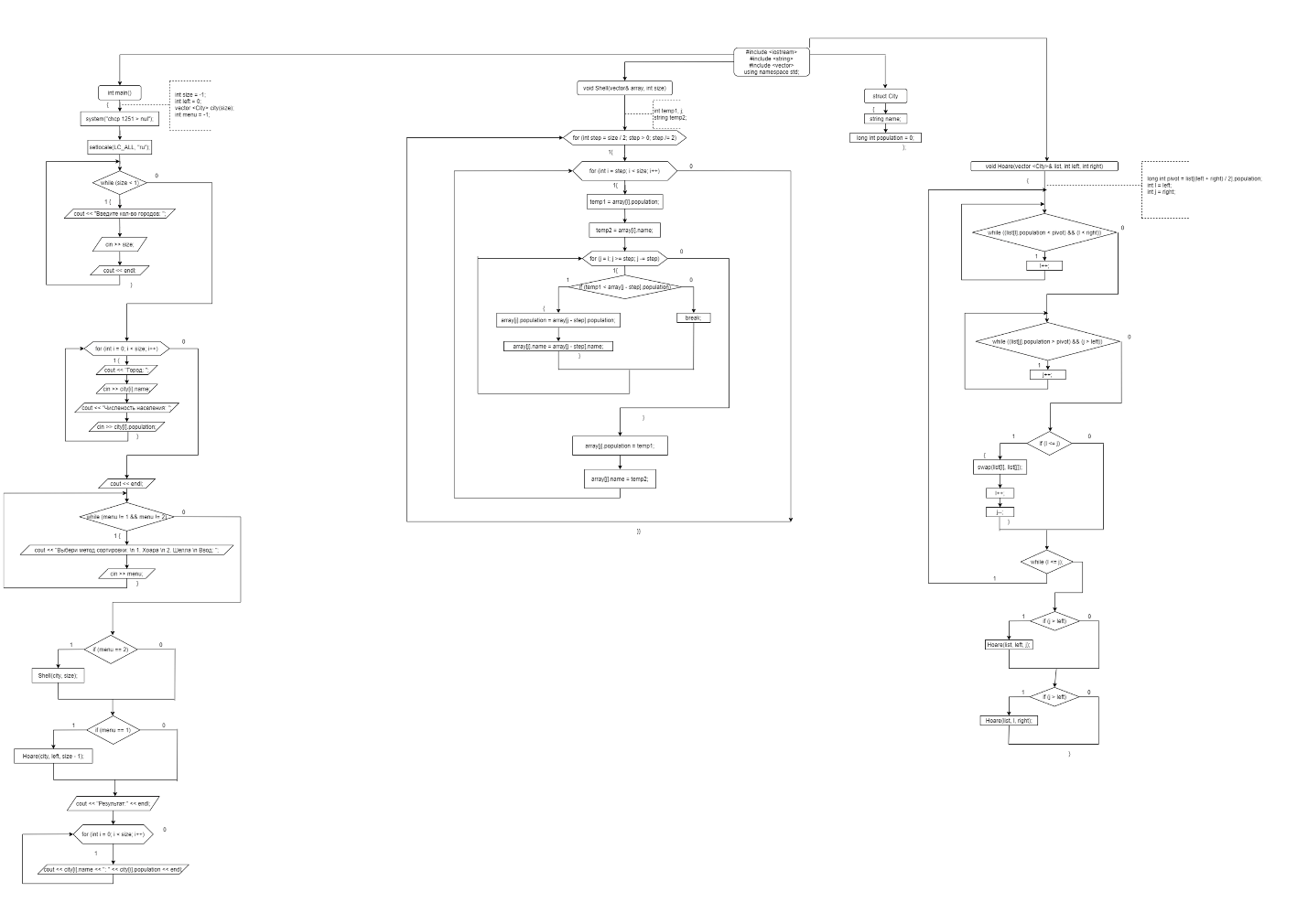
cout << vec3[i] << " ";

cout << endl;

return 0;

}

Блок-схема:



Работа кода

