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 теории алгоритмов и структуры данных

Тема: Методы внешней сортировки: метод прямого слияния, сбалансированного слияния, метод многофазной сортировки

Вариант: NULL

Выполнил работу

студент группы РИС-20-1б

Рябов Никита Андреевич

Проверила

Доцент кафедры ИТАС Полякова О.А.

Пермь, 2021

**Постановка задачи**

Сортировать множество при помощи алгоритмов внешней сортировки: естественное слияние, сбалансированное слияние, многофазное слияние.

**Анализ задачи**

*Какие предстоит выполнить действия.*

Пробегает по вектору печатает элементы

void PrintVector(vector<int> vec)

{

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

cout << vec[i] << " ";

cout << endl;

}

В цикле печатает скобки, внутри которых печатается вектор. Вектор печатается по тому же принципу, что и в функции выше

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

{

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;

}

Наименьший из последних элементов складывается в результирующую серию. Так продолжается до окончания короткой серии. Затем остатки длинной серии приписываются к результату

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

{

int i = 0, j = 0;

vector<int> res;

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

{

if (ser1[i]<ser2[j])

{

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

i++;

}

else

{

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

j++;

}

}

if (i < ser1.size()) for (int k=i;k<ser1.size();k++) res.push\_back(ser1[k]);

else if (j < ser2.size()) for (int k=j;k<ser2.size();k++) res.push\_back(ser2[k]);

return res;

}

Пробегает по серии. Если элемент больше предыдущего, он записывается в серию. Иначе текущая серия записывается в вектор серий, начинается новая серия и уже туда записывается текущий элемент

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;

}

Делит на серии и сливает до тех пор, пока не останется одна серия

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];

}

Делит на 2 (примерно) равные серии. Рекурсивно вызывает себя от получившихся серий и сливает результаты. Возвращает результат слияния

vector<int> BalancedMergeSort(vector<int> vec)

{

if (vec.size()==1) return vec;

else if (vec.size()==2)

{

if (vec[0]>vec[1]) swap(vec[0],vec[1]);

return vec;

}

else

{

vector <int> new\_vec1;

vector <int> new\_vec2;

for (int i=0;i<vec.size()/2+1;i++) new\_vec1.push\_back(vec[i]);

for (int i=vec.size()/2+1;i<vec.size();i++) new\_vec2.push\_back(vec[i]);

return MergeSeries(BalancedMergeSort(new\_vec1),BalancedMergeSort(new\_vec2));

}

}

Построена поверх функции печати серии, по сути представляет трехкратный вызов последней

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;

}

Удаляет известное число элементов с конца

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

{

for (int i = 0; i <num; i++) vec.pop\_back();

return vec;

}

Берет 2 числа и сохраняет в них 2 последних члена последовательности Фибоначчи до тех пор, пока их сумма не превысит данное число

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

{

size1 = 0;

size2 = 1;

while (size1 + size2 < numberOfSeries)

{

int t = size2;

size2 = size2 + size1;

size1 = t;

}

}

Делит серию на несколько равных. Если не получается – выгружает остаток в еще одну дополнительную серию

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

{

int num=1; //кол-во элементов в серии

vector<vector<int>> res;

int i=0;

vector<int> newSerie;

while (i<vec.size())

{

if ((i+1)%num==0 || i==vec.size()-1)

{

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

res.push\_back(newSerie);

newSerie.clear();

}

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

i++;

}

return res;

}

Находит минимальное кол-во серий в файлах. Сливает попарно именно это кол-во серий. Удаляет использованные серии из исходных файлов.

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

{

vector<vector<int>> res;

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

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

{

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

}

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

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

return res;

}

Получает нужные числа Фибоначчи, делит вектор на серии, серии на 2 файла. Остаток заполняет пустыми сериями. Сортирует серии внутри файлов.

Находит пустой файл и сливает туда 2 оставшихся. Последнее действие повторяется до тех пор, пока есть только один пустой файл. Удаляет элементы мнимых серий и возвращает результат

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 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" << endl;

PrintFiles(F1, F2, F3);

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);

cout << "Слияние файлов" << endl;

int countEmptyFiles = 1;

while (countEmptyFiles < 2)

{

int num = 1;

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

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

switch (num)

{

case 1:

F1 = MergeFiles(F2, F3);

break;

case 2:

F2 = MergeFiles(F1, F3);

break;

case 3:

F3 = MergeFiles(F1, F2);

break;

}

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],size1+size2-series.size());

if (F2.size() != 0) return DeleteEmptyElements(F2[0],size1+size2-series.size());

return DeleteEmptyElements(F3[0],size1+size2-series.size());

}

*С каким типом данных действуем*

Целые числа

*В каком виде представлены данные*

Вектора целых чисел и вектора векторов целых чисел

*Какие поля*

Структуры в данной программе отсутствуют

*Операторы ввода/вывода*

В программе используются операторы cin/cout.

**Блок-схема**

****

****

****

****

****

****

****

****

****

****

****

****

****

**Код**

#include <iostream>

#include <vector>

using namespace std;

void PrintVector(vector<int> vec)

{

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

cout << vec[i] << " ";

cout << endl;

}

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

{

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;

}

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

{

int i = 0, j = 0;

vector<int> res;

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

{

if (ser1[i]<ser2[j])

{

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

i++;

}

else

{

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

j++;

}

}

if (i < ser1.size()) for (int k=i;k<ser1.size();k++) res.push\_back(ser1[k]);

else if (j < ser2.size()) for (int k=j;k<ser2.size();k++) res.push\_back(ser2[k]);

return res;

}

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> BalancedMergeSort(vector<int> vec)

{

if (vec.size()==1) return vec;

else if (vec.size()==2)

{

if (vec[0]>vec[1]) swap(vec[0],vec[1]);

return vec;

}

else

{

vector <int> new\_vec1;

vector <int> new\_vec2;

for (int i=0;i<vec.size()/2+1;i++) new\_vec1.push\_back(vec[i]);

for (int i=vec.size()/2+1;i<vec.size();i++) new\_vec2.push\_back(vec[i]);

return MergeSeries(BalancedMergeSort(new\_vec1),BalancedMergeSort(new\_vec2));

}

}

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

{

for (int i = 0; i <num; i++) vec.pop\_back();

return vec;

}

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

{

size1 = 0;

size2 = 1;

while (size1 + size2 < numberOfSeries)

{

int t = size2;

size2 = size2 + size1;

size1 = t;

}

}

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

{

int num=1; //кол-во элементов в серии

vector<vector<int>> res;

int i=0;

vector<int> newSerie;

while (i<vec.size())

{

if ((i+1)%num==0 || i==vec.size()-1)

{

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

res.push\_back(newSerie);

newSerie.clear();

}

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

i++;

}

return res;

}

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

{

vector<vector<int>> res;

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

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

{

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

}

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

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

return res;

}

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 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" << endl;

PrintFiles(F1, F2, F3);

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);

cout << "Слияние файлов" << endl;

int countEmptyFiles = 1;

while (countEmptyFiles < 2)

{

int num = 1;

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

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

switch (num)

{

case 1:

F1 = MergeFiles(F2, F3);

break;

case 2:

F2 = MergeFiles(F1, F3);

break;

case 3:

F3 = MergeFiles(F1, F2);

break;

}

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],size1+size2-series.size());

if (F2.size() != 0) return DeleteEmptyElements(F2[0],size1+size2-series.size());

return DeleteEmptyElements(F3[0],size1+size2-series.size());

}

int main()

{

setlocale(LC\_ALL, "Russian");

cout << "Исходный вектор" << endl;

vector<int> vec = {9,3,1,6,4,2,8,7,0,5};

PrintVector(vec);

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

PrintVector(NaturalMergeSort(vec));

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

PrintVector(BalancedMergeSort(vec));

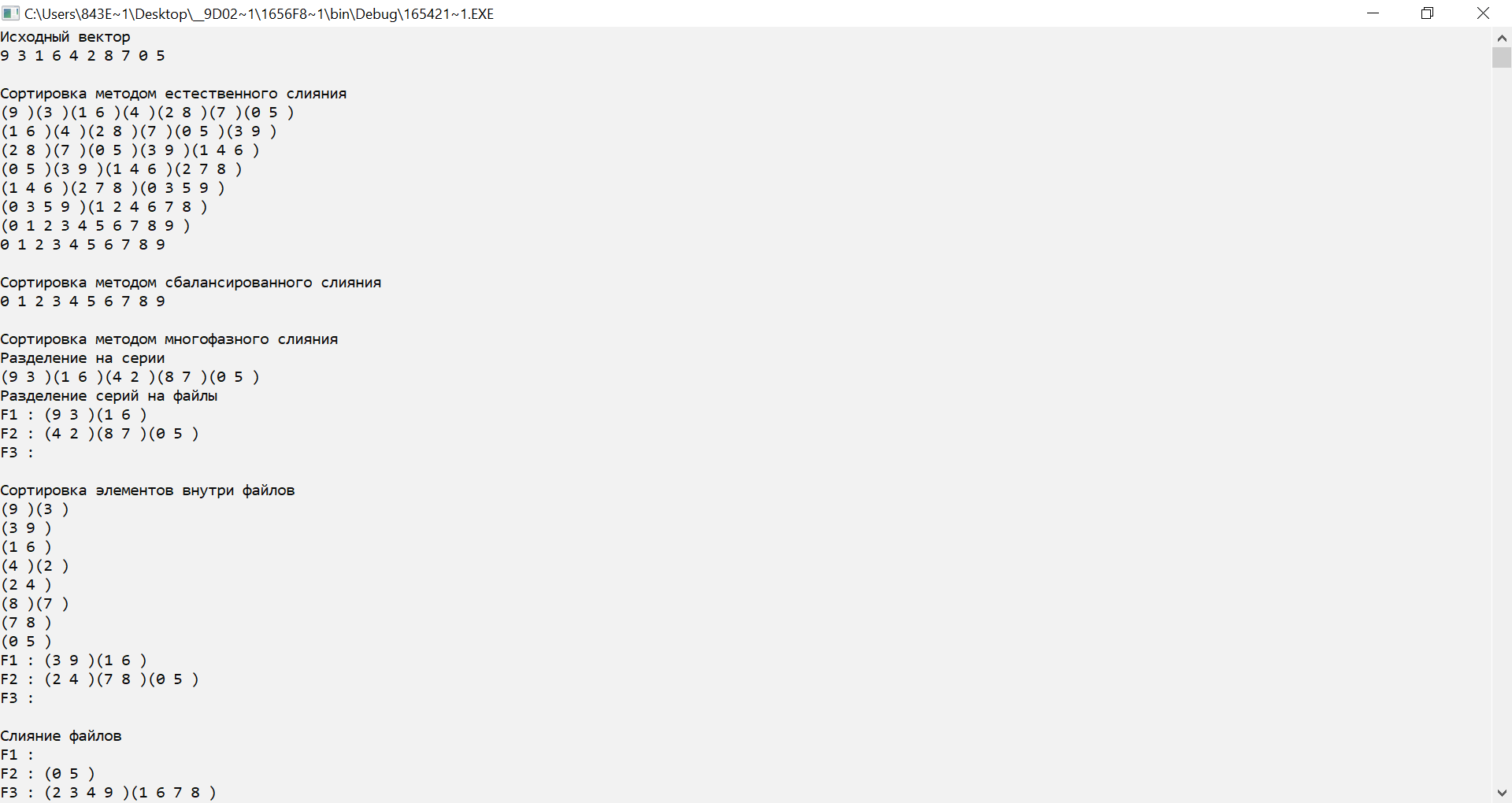
cout << endl << "Сортировка методом многофазного слияния" << endl;

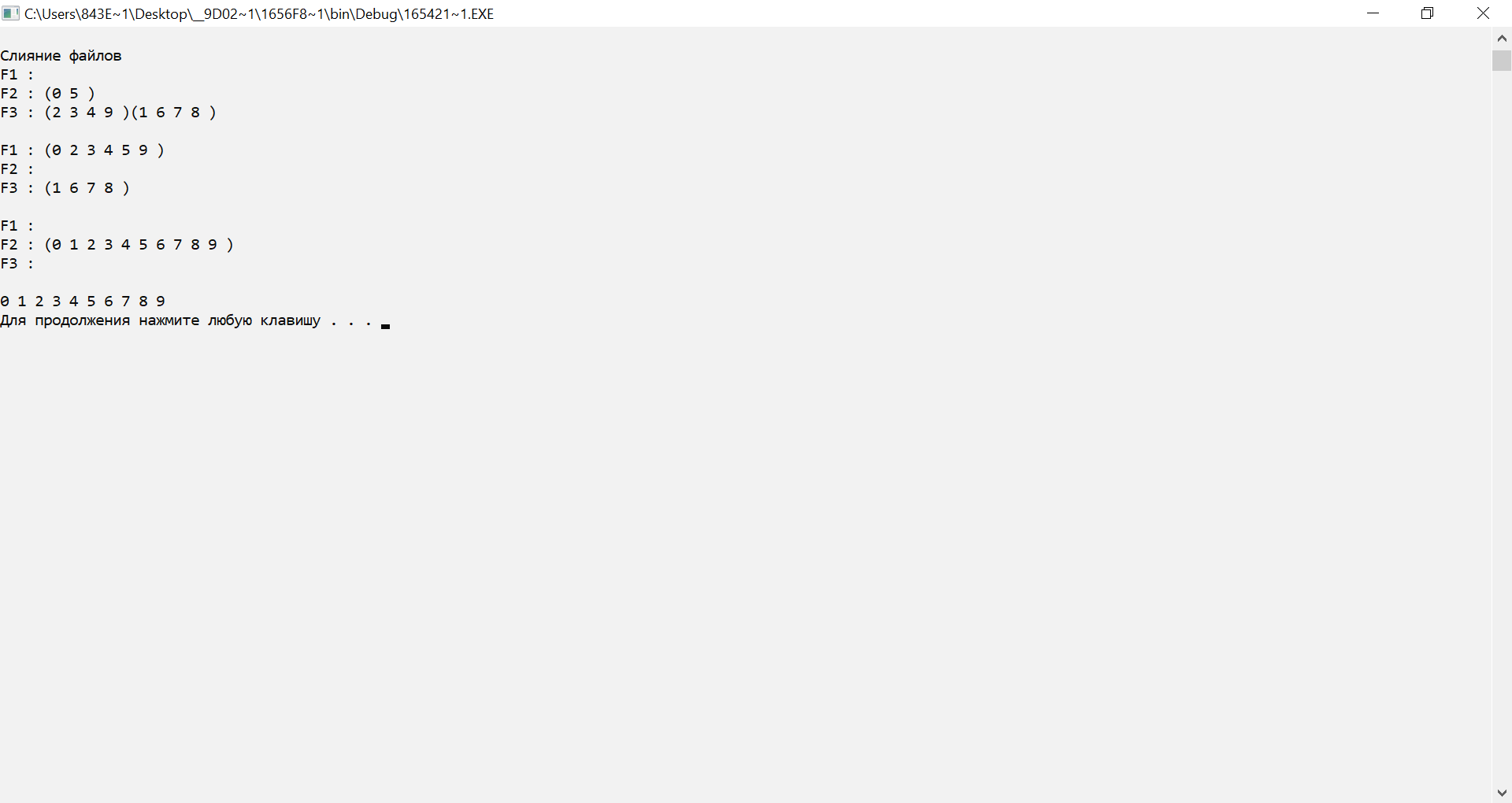
PrintVector(PolyphaseMergeSort(vec));

system("pause");

}

**Скриншоты**

****

****