**«Пермский национальный исследовательский  
политехнический университет»**

**О Т Ч Ё Т**

**по лабораторной работе №16**

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

Вариант 5

Выполнил работу

студент группы ИВТ-20-2б

Нефедов Л.В.

Проверила

Доцент кафедры ИТАС

Полякова О.А.

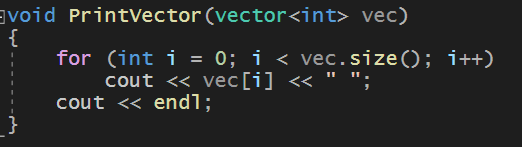
Пермь, 2021

**Постановка задачи**

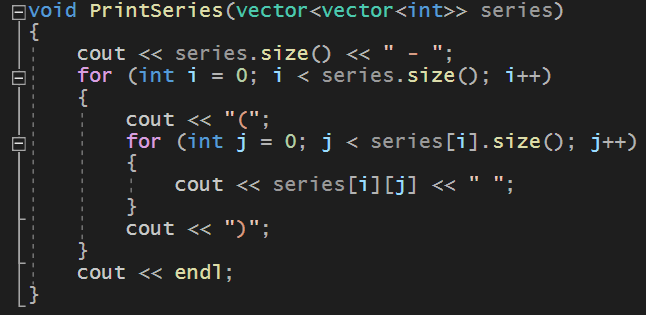
Нужно отсортировать числа тремя методами сортировок—методом прямого слияния, сбалансированного слияния, и методом многофазной сортировки. Все числа формируются рандомным методом.

**Анализ задачи**

1. Для решения задачи необходимо…
   1. Создать функцию PrintVector типа void, которая будет выводить числа на экран;



* 1. Создать функцию PrintSeries, которая будет выводить в консоль поэтапно сортировку чисел;



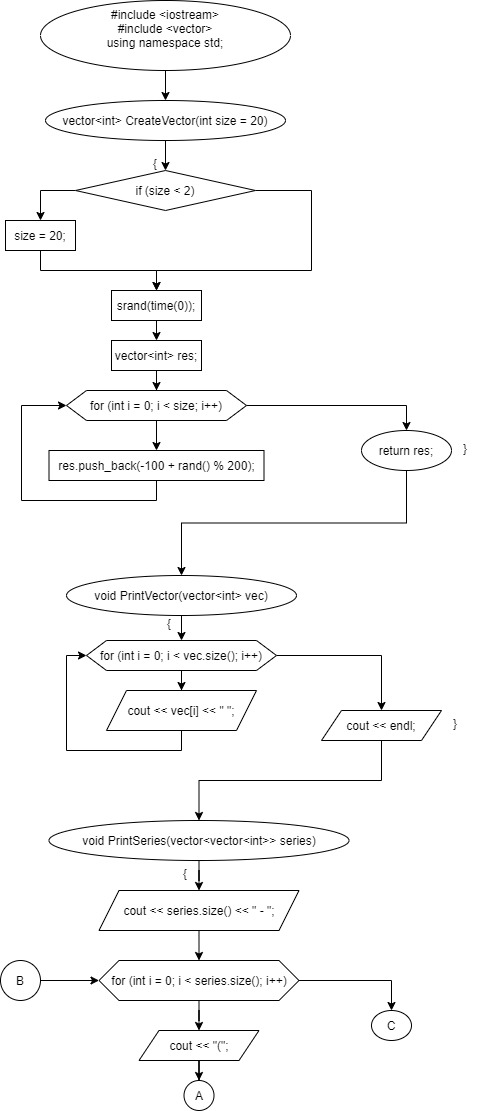
1. В программе были использованы следующие типы данных:
   1. Тип данных int для счетчика цикла;

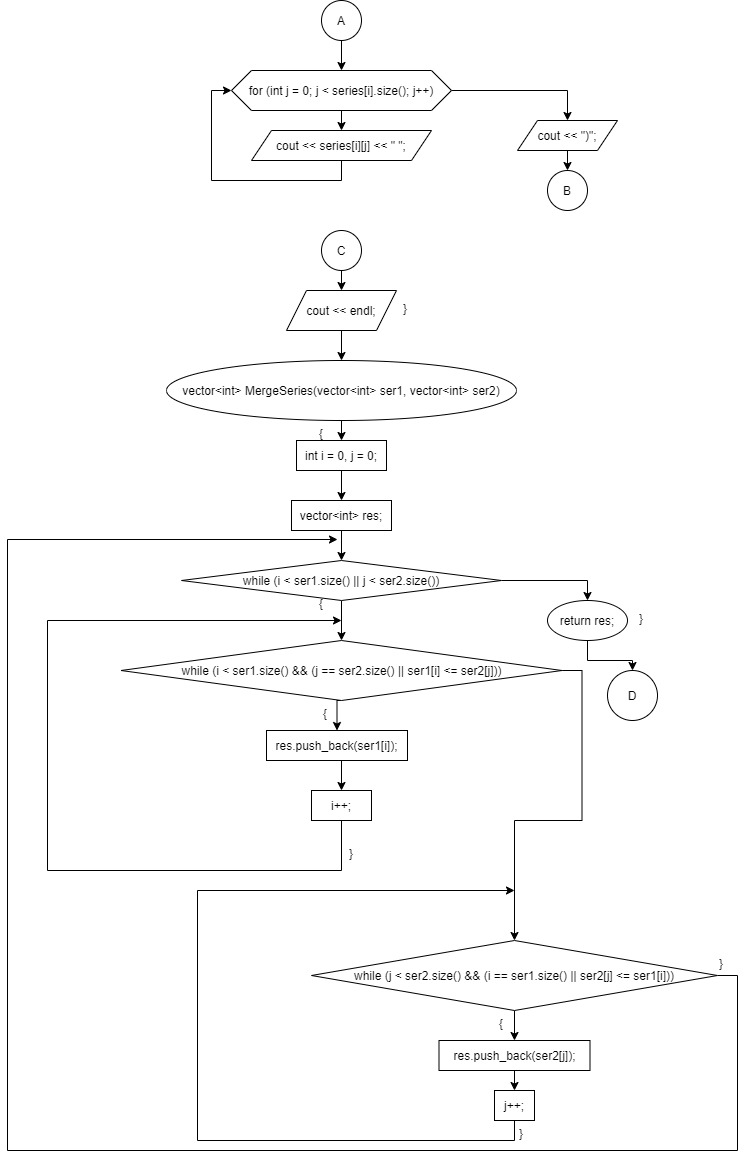


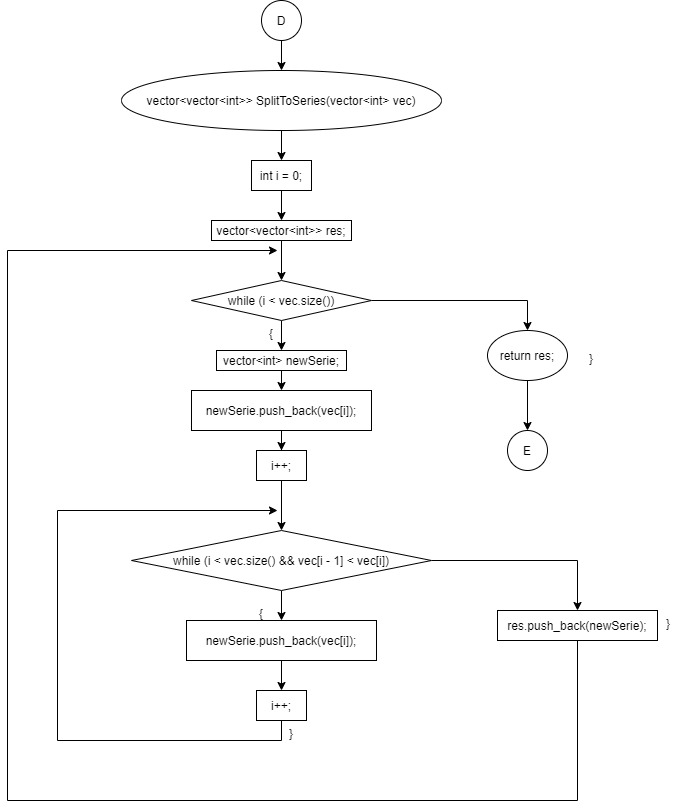
* 1. Тип данных bool для проверки различных условий внутри программы;

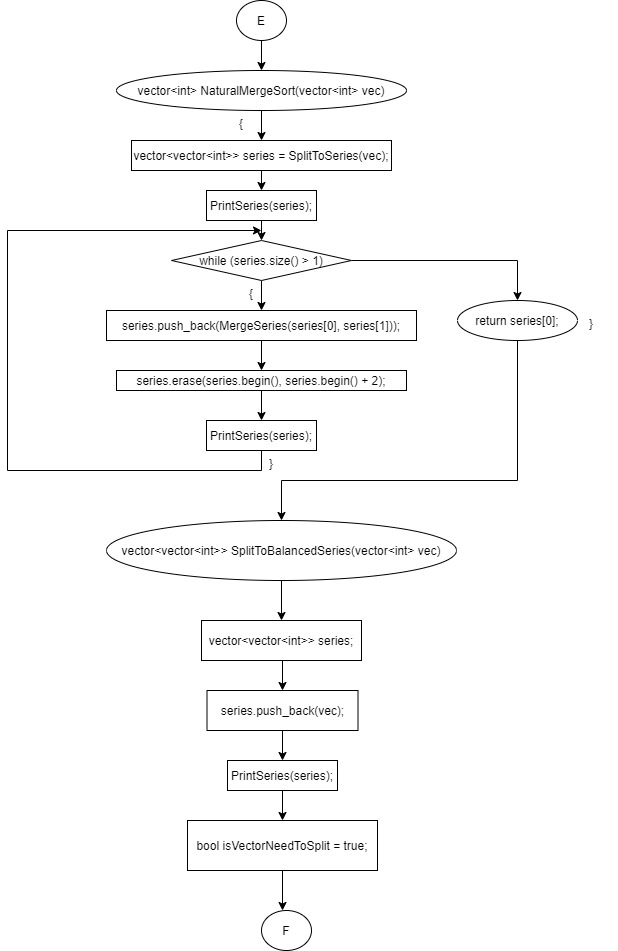


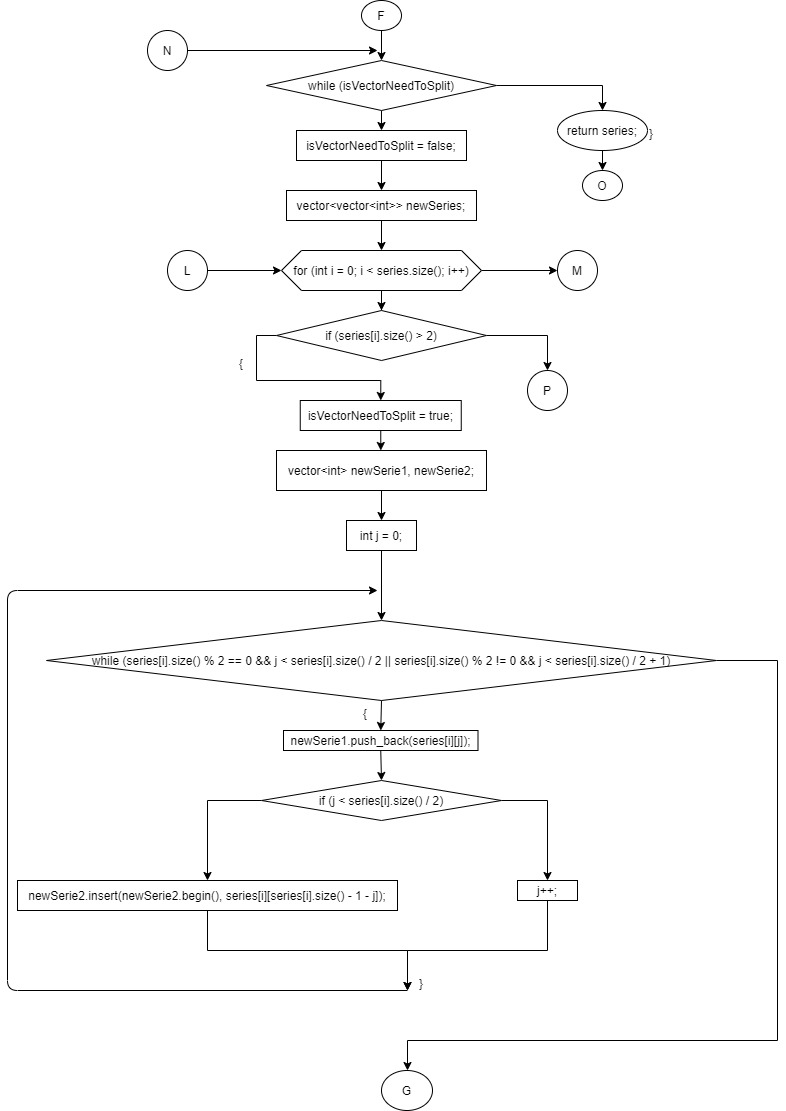
**Блок—схема программы**

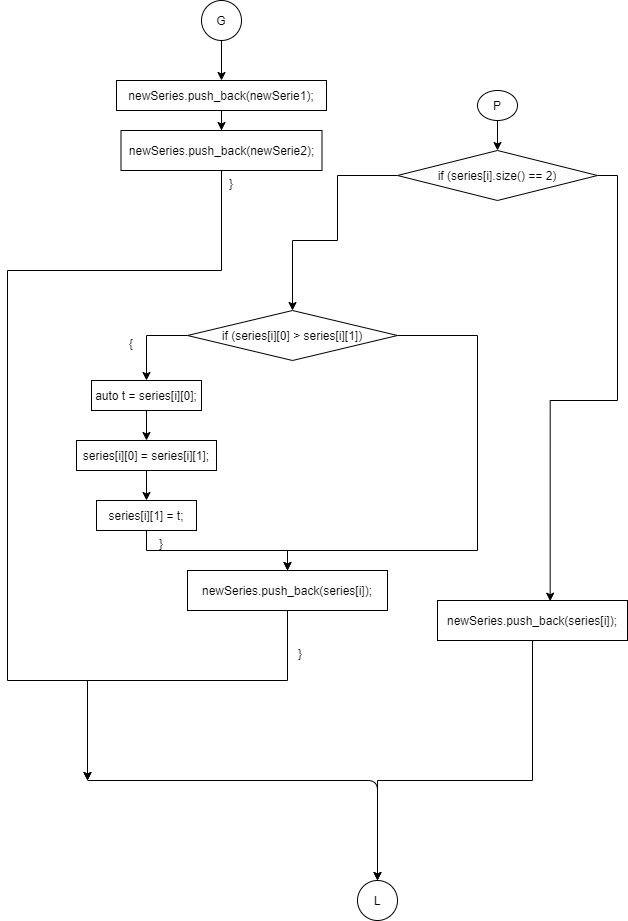


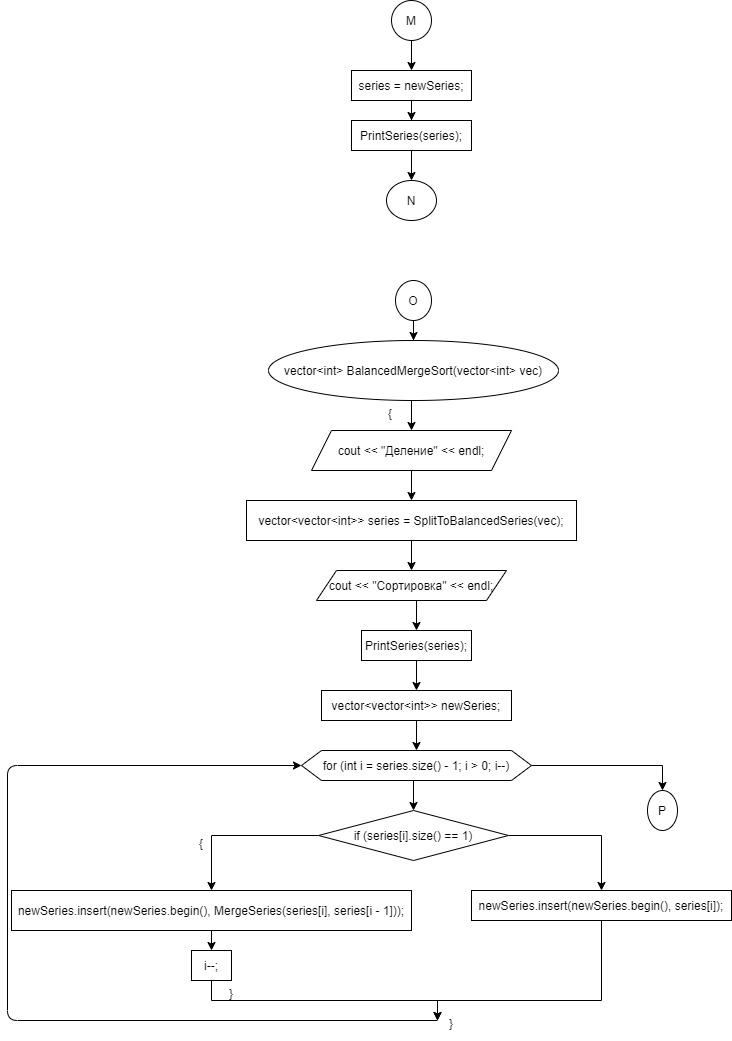


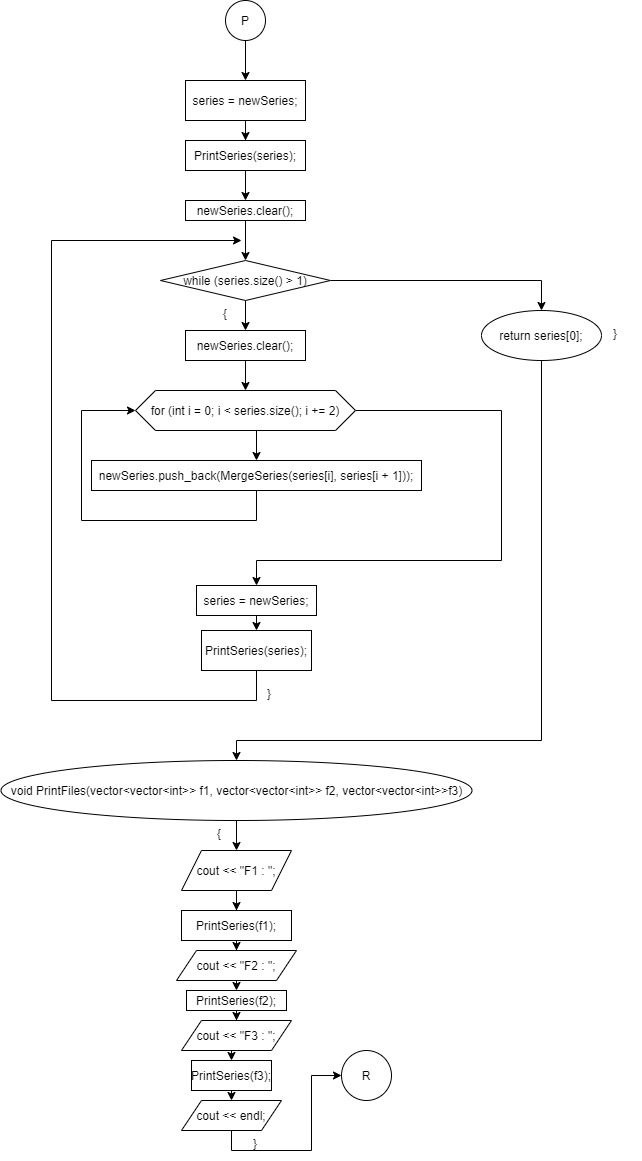


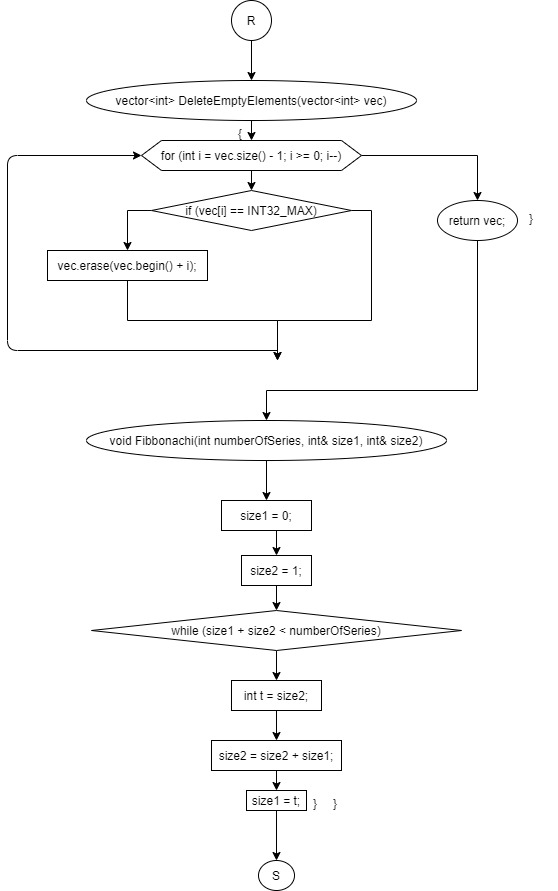


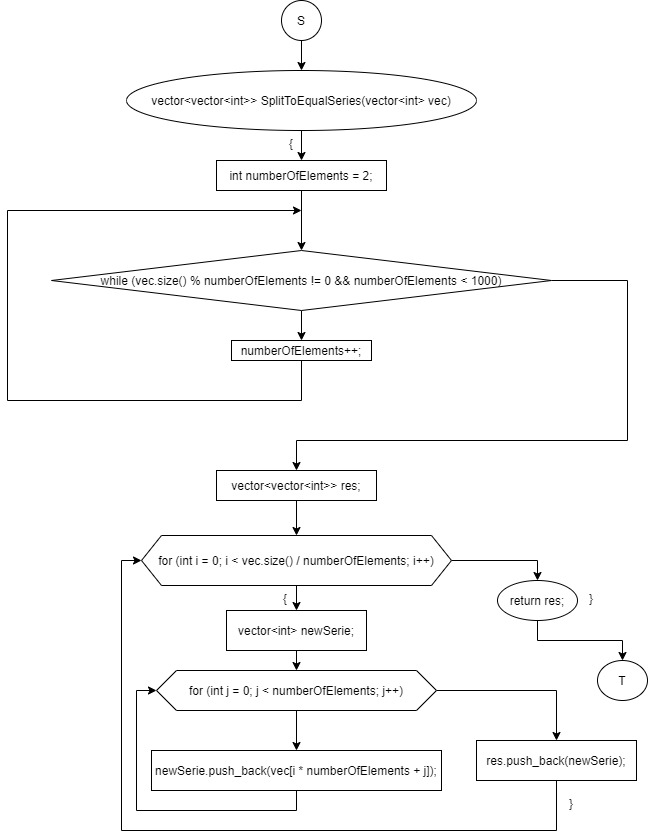


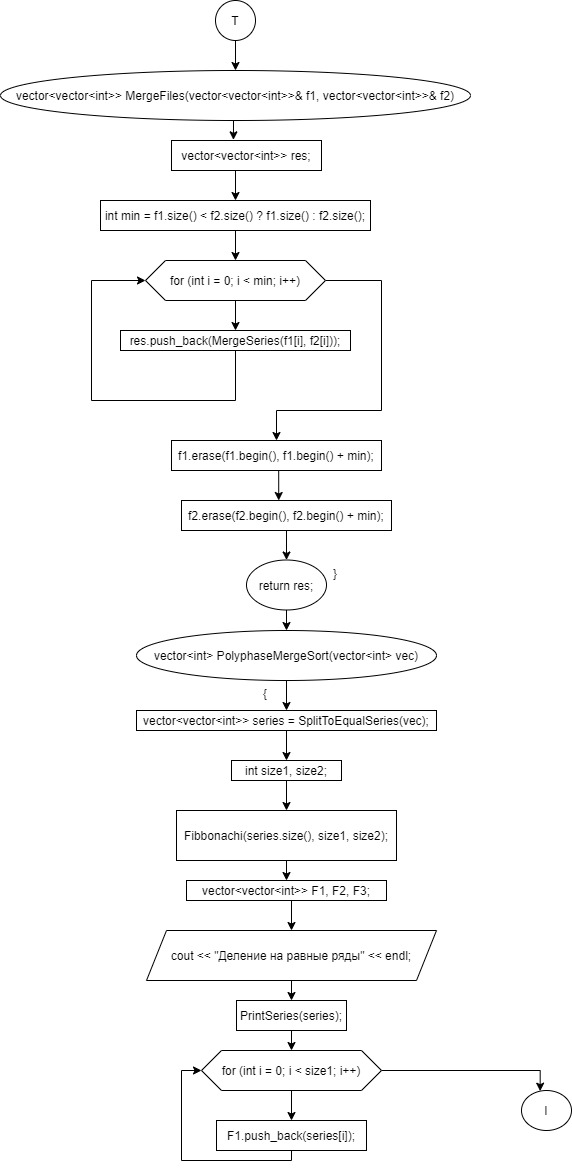


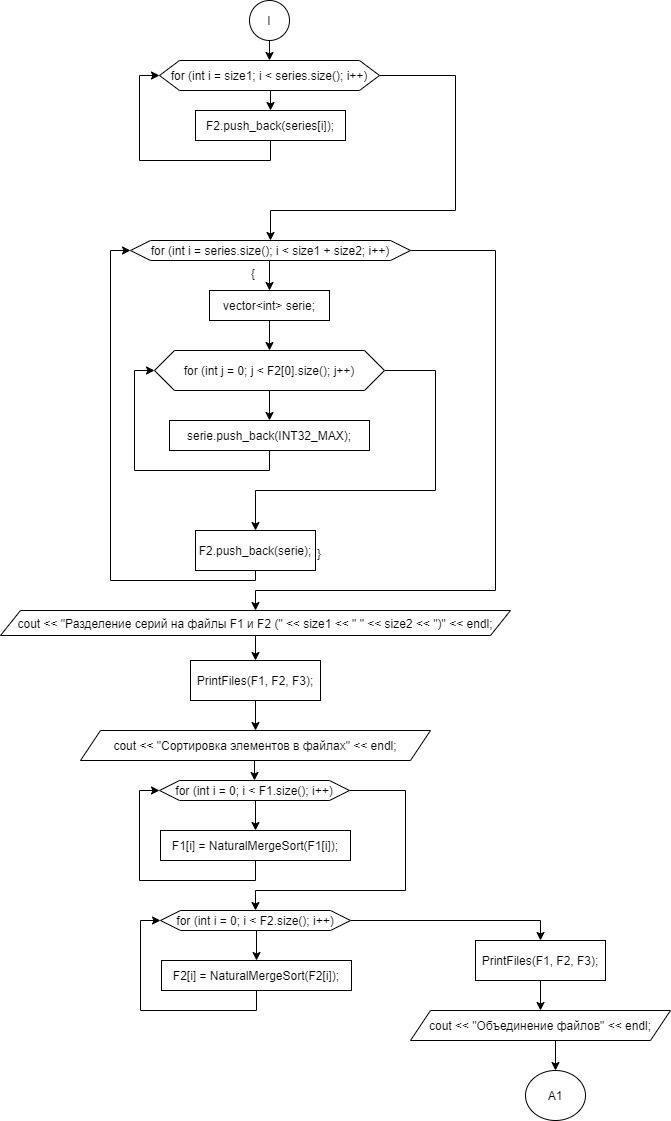


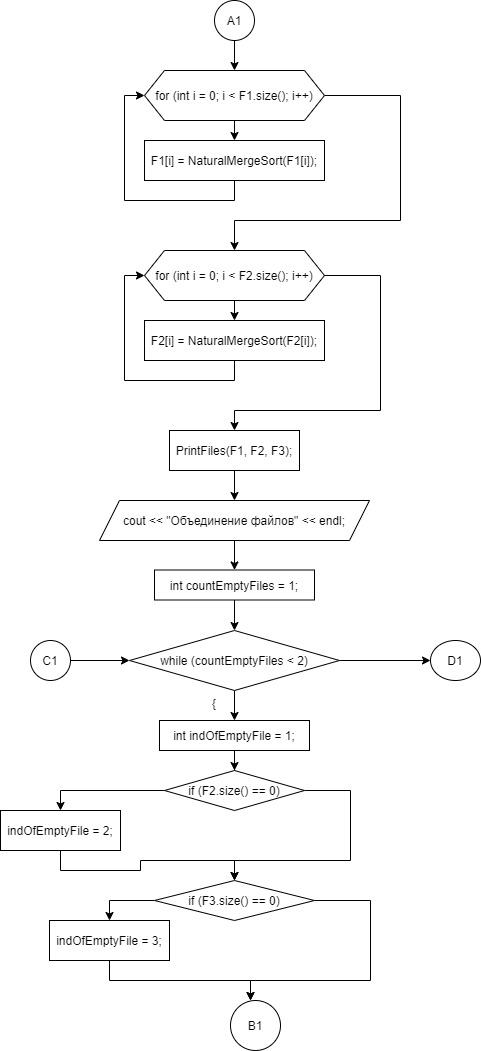


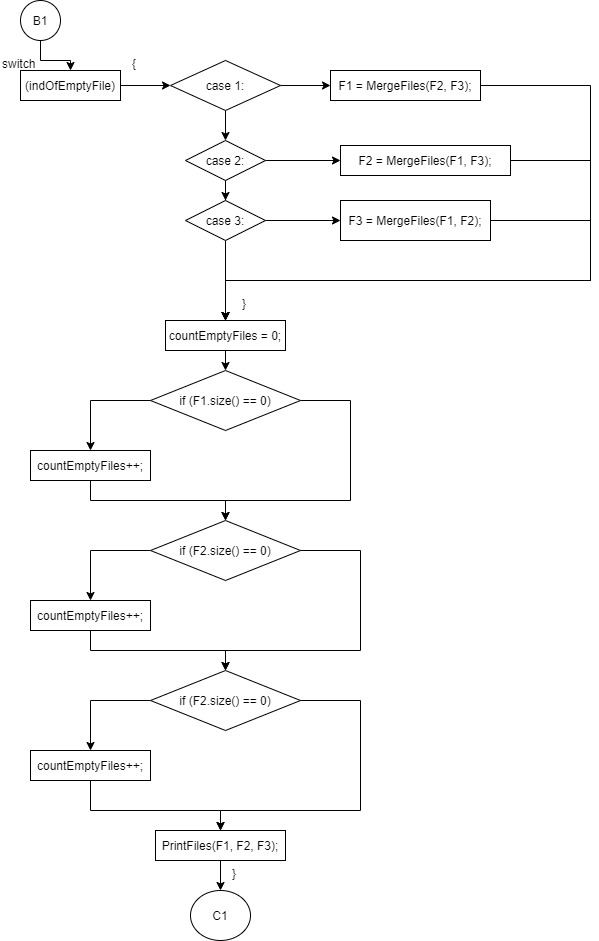


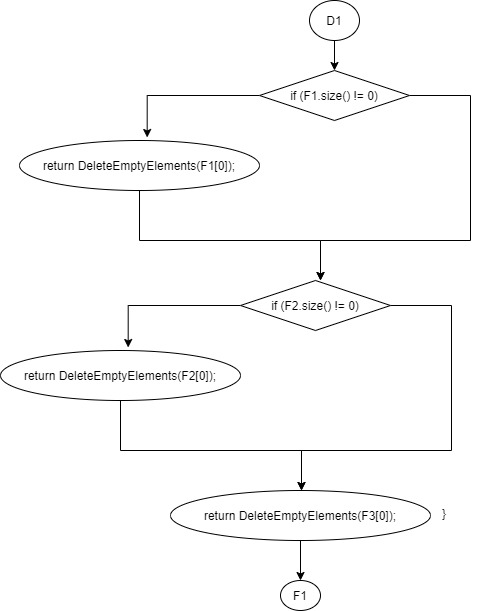


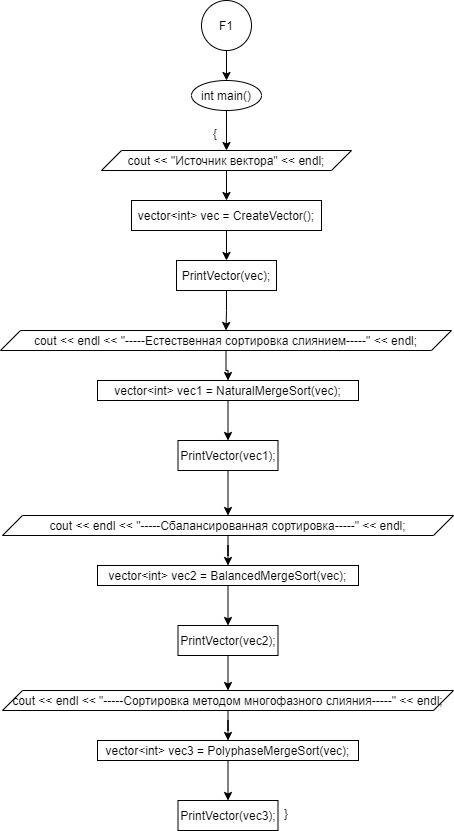












**Решение**

#include <iostream>

#include <vector>

using namespace std;

// === использование для всех методов ===

// минимальный размер 2

vector<int> CreateVector(int size = 20)

{

if (size < 2) size = 20;

srand(time(0));

vector<int> res;

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

{

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

}

return res;

}

void PrintVector(vector<int> vec)

{

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

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

cout << endl;

}

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;

}

// серии должны быть отсортированы

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;

}

// === естественная часть сортировки слиянием ===

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];

}

// === сбалансированная сортировка ===

vector<vector<int>> SplitToBalancedSeries(vector<int> vec)

{

vector<vector<int>> series;

series.push\_back(vec);

PrintSeries(series);

bool isVectorNeedToSplit = true;

while (isVectorNeedToSplit)

{

isVectorNeedToSplit = false;

vector<vector<int>> newSeries;

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

{

// деление серии, если ее размерность больше двух

if (series[i].size() > 2)

{

isVectorNeedToSplit = true;

vector<int> newSerie1, newSerie2;

int j = 0;

// разное условие для нечетного и нечетного числа элементов в серии

while (series[i].size() % 2 == 0 && j < series[i].size() / 2 || series[i].size() % 2 != 0 && j < series[i].size() / 2 + 1)

{

newSerie1.push\_back(series[i][j]);

if (j < series[i].size() / 2)

{

newSerie2.insert(newSerie2.begin(), series[i][series[i].size() - 1 - j]);

}

j++;

}

newSeries.push\_back(newSerie1);

newSeries.push\_back(newSerie2);

}

// сортировка двух элементов в серии

else if (series[i].size() == 2)

{

if (series[i][0] > series[i][1])

{

auto t = series[i][0];

series[i][0] = series[i][1];

series[i][1] = t;

}

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

}

// последний элемент в конец

else

{

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

}

}

series = newSeries;

PrintSeries(series);

}

return series;

}

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

{

cout << "Деление" << endl;

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

cout << "Сортировка" << endl;

PrintSeries(series);

// сортировка нескольких элементов

vector<vector<int>> newSeries;

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

{

if (series[i].size() == 1)

{

newSeries.insert(newSeries.begin(), MergeSeries(series[i], series[i - 1]));

i--;

}

else

{

newSeries.insert(newSeries.begin(), series[i]);

}

}

series = newSeries;

PrintSeries(series);

// сортировка других элементов

newSeries.clear();

while (series.size() > 1)

{

newSeries.clear();

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

{

newSeries.push\_back(MergeSeries(series[i], series[i + 1]));

}

series = newSeries;

PrintSeries(series);

}

return series[0];

}

// === Многофазная часть сортировки слиянием ===

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;

}

// empty == INT32\_MAX

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;

}

// функция Фибоначчи для поиска размерности рядов

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)

{

// поиск доступного количества элементов в серии

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;

}

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

{

// сортировка

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]));

}

//удаление ненужных элементов

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

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

return res;

}

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

{

// разделение на файлы

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

int size1, size2;

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

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

cout << "Деление на равные ряды" << 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++)

{

// создание пустой серии

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);

// Сортировка элементов внутри серий

cout << "Сортировка элементов в файлах" << 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);

// Финальная сортировка

cout << "Объединение файлов" << endl;

int countEmptyFiles = 1;

while (countEmptyFiles < 2)

{

// Поиск пустого файла

int indOfEmptyFile = 1;

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

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

// сортировка

switch (indOfEmptyFile)

{

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

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

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

}

// проверка окончания цикла

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()

{

system("chcp 1251>nul");

cout << "Источник вектора" << endl;

vector<int> vec = CreateVector();

PrintVector(vec);

cout << endl << "-----Естественная сортировка слиянием-----" << endl;

vector<int> vec1 = NaturalMergeSort(vec);

PrintVector(vec1);

cout << endl << "-----Сбалансированная сортировка-----" << endl;

vector<int> vec2 = BalancedMergeSort(vec);

PrintVector(vec2);

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

vector<int> vec3 = PolyphaseMergeSort(vec);

PrintVector(vec3);

}

**Скриншоты**

