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”

**Вар.20**

Выполнил работу:

студент группы ИВТ-20-2Б

Кузнецов Н.Д.

Проверил:

Доцент кафедры ИТАС

Полякова О.А.

Пермь, 2020

**Цель задачи**

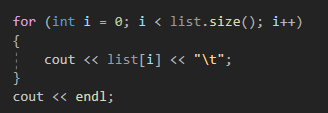
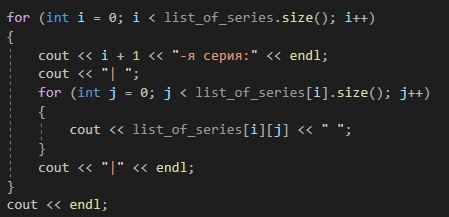
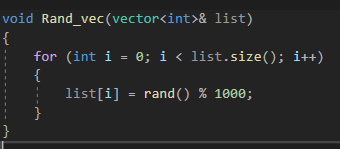
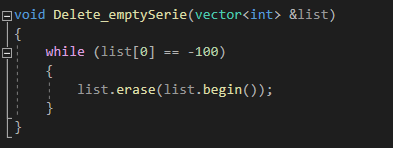
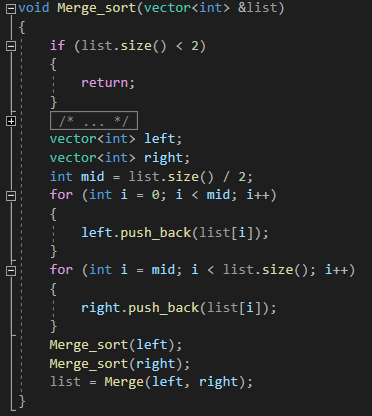
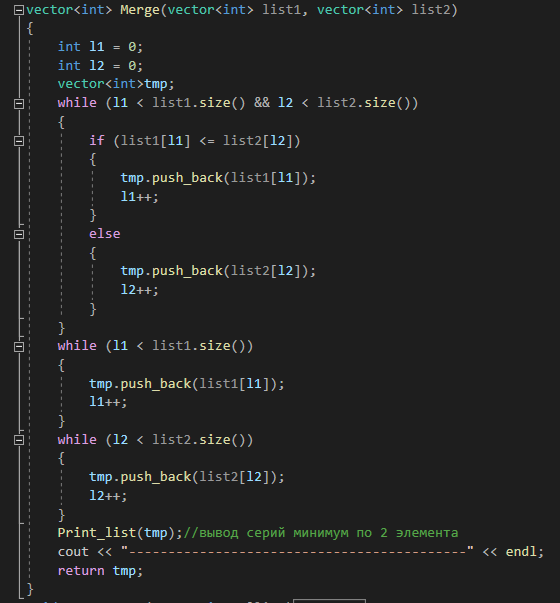
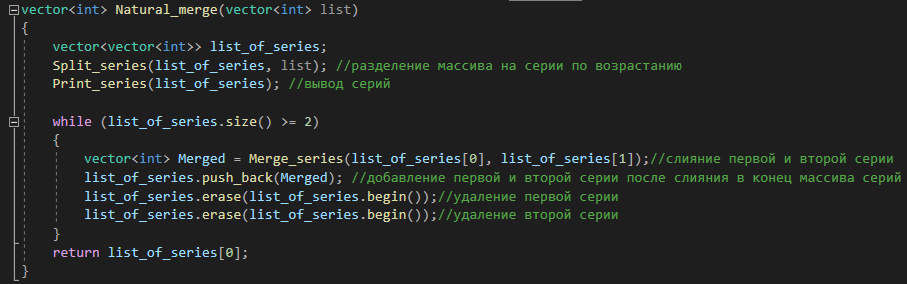
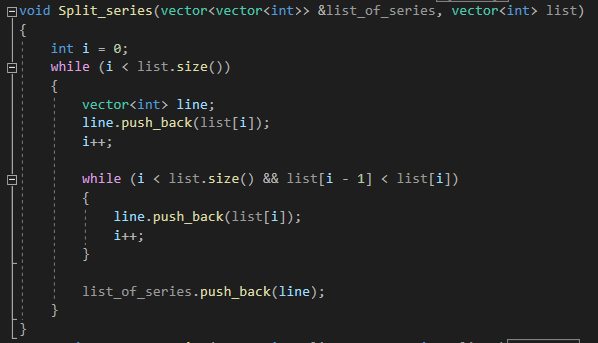
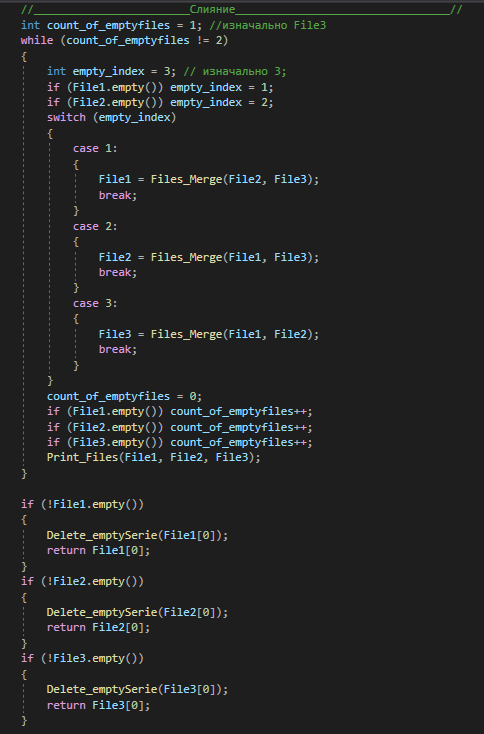
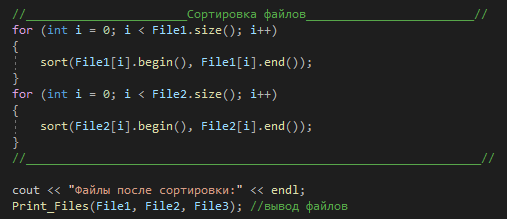
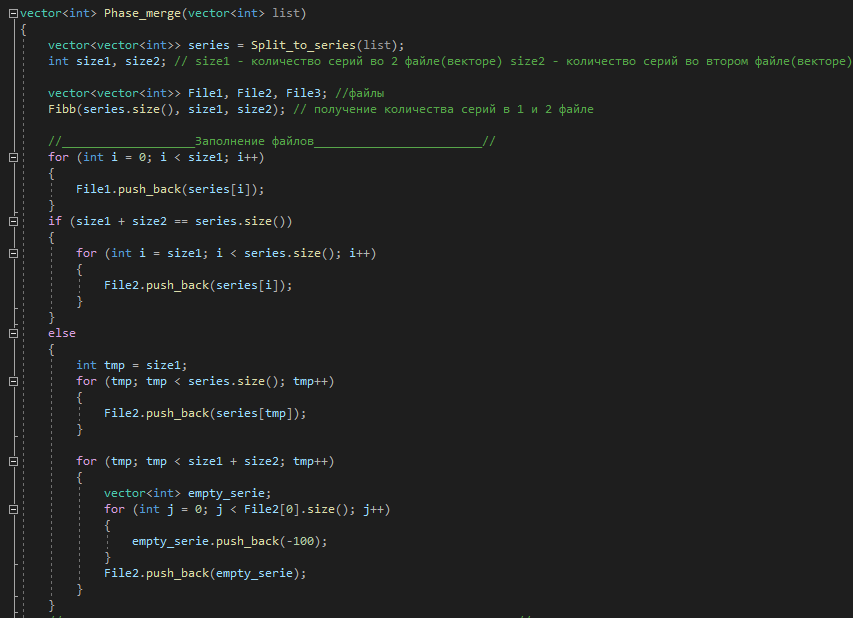
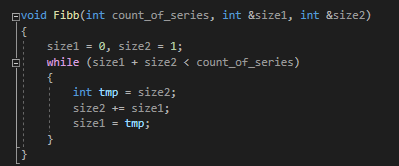
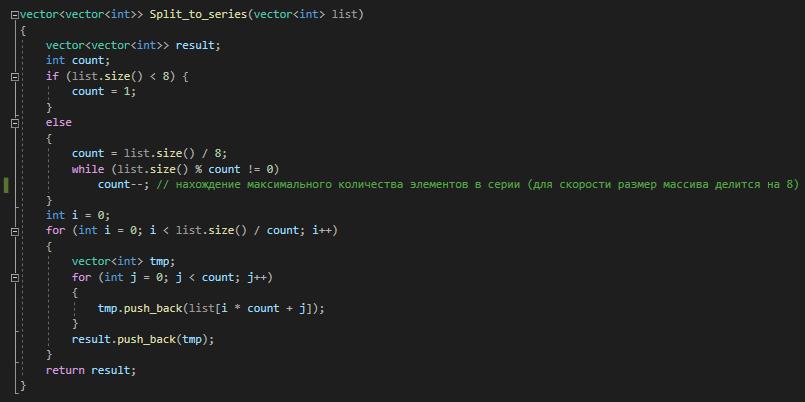
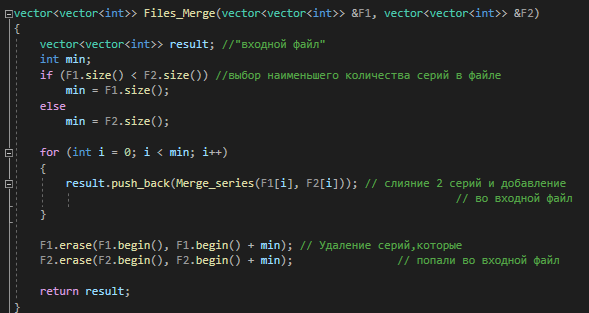
1. Получить практические навыки работы с внешними сортировками;
2. Для достижения поставленной цели необходимо решить следующие задачи:

* Организовать ввод-вывод данных.
* Реализовать функцию слияния серий элементов.
* Реализовать функции внешних сортировок.
* Реализовать функции, разбивающие массивы на серии.
* Разработать функцию, заполняющую массив случайными числами.
* Разработать программу.

**Постановка задачи**

1. Создать динамический массив целых чисел, отсортировать их по возрастанию с помощью внешних сортировок. Массив заполняется с помощью ДСЧ.

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать функцию заполнения массива случайными числами.  
      
   2. Реализовать функцию вывода массива.  
      
   3. Реализовать функции разбиения массива на серии (Split\_series и Split\_to\_series). Функция Split\_series используется для разбиения массива на серии по принципу естественного слияния. Split\_to\_series разбивает массив на серии с максимальным количеством элементов.  
      
   4. Реализовать функции слияния серий (Merge, Merge\_series, Files\_Merge).  
      
   5. Реализовать функции внешних сортировок (Phase\_merge, Natural\_merge, Merge\_sort). Функция Phase\_merge использует вспомогательные функции Fibb, Delete\_emptySerie.
      1. Функция Fibb считает количество серий в 1 и 2 файле, с помощью чисел Фибоначчи.
      2. В случае, если количество серий в исходном массиве не удовлетворяет условию фибоначчи, то он заполняется пустыми сериями до нужного количества серий, для удаления пустых серий из отсортированного массива разработана функция Delete\_emptySerie.
2. В ходе работы были использованы следующие типы данных:
   1. int - для хранения целых чисел массива.
   2. vector - динамическая структура для хранения массива целых чисел.  
      
3. Для решения задачи данные были представлены в следующем виде:
   1. Vector<int> - динамический массив целых чисел.  
      
4. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Заполнение массива происходит с помощью ДСЧ (см.п.1.1).
   2. Вывод одномерного массива (исходного и отсортированного) осуществляется с помощью оператора cout.  
      
   3. Для вывода файлов и массива серий так же используется оператор cout.  
      
5. Поставленные задачи будут решены следующими действиями:
   1. Заполнения массива с помощью ДСЧ осуществляется циклом for.  
      
   2. Вывод исходного и конечного массивов осуществляется с помощью цикла for (см п.4.2).
   3. Вывод массива серий и файла осуществляется с помощью вложенного цикла for в цикл for (см п.4.3).
   4. Функция Delete\_emptySerie удаляет добавленные серии с помощью цикла while, элементы удаляются до тех пор, пока они равны - 100.  
      
   5. Функция Merge\_sort работает по принципу рекурсии. Внутри функции создаются два подмассива - левая часть массива, переданного в функцию, и правая. Они заполняются по принципу деления массива на 2. Далее функция вызывает саму себе с параметром left, второй раз с параметром right. Далее вызывается функция Merge c параметрами left и right.  
      
      1. Функция Merge принимает в качестве параметра два вектора, возвращает результат слияния этих двух векторов. Внутри функции создается результирующий вектор tmp. Первый цикл while работает до тех пор, пока один из указателей на индекс подмассива не выйдет за его границы. Внутри цикла используется оператор ветвления if: если элемент первого подмассива больше элемента второго подмассива, то он добавляется в результирующий массив, указатель на первый элемент инкрементируется, иначе такой же алгоритм для второго подмассива. Следующие циклы while проверяют вышел ли указатель на элемент подмассива за его границы. В случае, если указатель не вышел за границы, в результирующий массив добавляются элементы до тех пор, пока указатель не выйдет за границы.  
         
   6. В функции Natural\_merge создается массив серий переданного вектора (list\_of\_series). С помощью функции Split\_series массив разбивается на серии элементов, уже идущих по возрастанию. Далее циклом while первые две серии массива сливаются и добавляются в конец, далее серии, которые слились - удаляются. Цикл работает до тех пор, пока его размер больше равен 2. После работы цикла функция возвращает первый элемент из массива серий.  
      
      1. Функция Split\_series принимает в качестве параметров исходный массив и массив серий. Создается указатель на нулевой элемент массива. Разделение массива осуществляется с помощью цикла while, вложенного в while. Первый цикл работает до тех пор, пока указатель не вышел за границы исходного массива, и добавляется в массив серий один элемент (1 элемент - уже отсортирован), далее вторым циклом проверяется возрастание подряд идущих элементов. Если элементы возрастают, то они добавляются в текущую серию.  
         
      2. Функция Merge\_series аналогично функции merge (см.п.5.5.1) сливает две серии, переданные в функцию.
   7. В функции Phase\_merge создается массив серий с одинаковым количеством элементов (series). Далее создаются так называемые файлы (массив серий), два из которых служат в качестве выходных, третий в качестве входного. Далее с помощью функции Fibb находится количество серий в 1 и 2 файлах. Далее с помощью двух циклов for заполняются 1 и 2 файлы. Если количество серий меньше, чем сумма размеров двух файлов, то второй файл заполняется пустыми сериями. Далее с помощью цикла for и стандартной сортировки массивов сортируются серии в файлах. Для самого слияния серий из файлов используется функция Files\_Merge. Для получения отсортированного массива вводится целочисленная переменная, которая отвечает за размер пустых файлов, цикл работает до тех пор, пока размер не равен 2. Внутри цикла так же используется целочисленная переменная, отвечающая за индекс пустого файла. С помощью оператора if проверяется размер всех трех файлов, переменной присваивается индекс файла. Далее используется оператор ветвления switch, который проверяет значение индекса пустого файла, в случае совпадения пустой файл заполняется сериями, которые сливаются из двух других файлов. Далее переменной count\_of\_emptyfiles присваивается нулевое значение и с помощью операторов if подсчитывается количество пустых файлов. После отработки цикла все файлы проверяются на пустоту, если файл не пустой, то из него удаляются все пустые серии и функция возвращает его первую серию (отсортированный массив).   
      
      1. Функция Fibb принимает количество серий в исходном массиве и размеры 1 и 2 файлов. Size1 присваивается нулевое значение, size2 = 1. С помощью цикла while осуществляется подсчет количества серий в файлах (числа фибоначчи).  
         
      2. Функция Split\_to\_series считает количество элементов в серии, если размер массива меньше 8, то его массив делится на серии по 1 элементу, иначе переменной count присваивается значение размера массива, деленного на 8, далее с помощью цикла while находится максимальное число, которое делит размер массива без остатка. Далее с помощью 2 циклов for заполняется массив серий, где каждая серия имеет размер count.  
         
      3. Функция Files\_Merge сравнивает количество серий в двух файлах и находит наименьшее значение (min). Далее с помощью цикла for сливаются все серии до индекса min (с помощью функции Merge\_series), после из файлов удаляются по min серий.  
         

**Код программы**

#include <iostream>

#include <vector>

#include <ctime>

#include <algorithm>

using namespace std;

void Print\_list(vector<int> list)

{

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

{

cout << list[i] << "\t";

}

cout << endl;

}

void Rand\_vec(vector<int>& list)

{

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

{

list[i] = rand() % 1000;

}

}

int GetSize()

{

int size;

cout << "Введите размер массива:\t";

while (!(cin >> size) || size < 1)

{

cin.clear();

while (cin.get() != '\n')

{

cin.ignore();

}

cout << "Введите корректное число:\t";

}

return size;

}

//balanced merge

vector<int> Merge(vector<int> list1, vector<int> list2)

{

int l1 = 0;

int l2 = 0;

vector<int>tmp;

while (l1 < list1.size() && l2 < list2.size())

{

if (list1[l1] <= list2[l2])

{

tmp.push\_back(list1[l1]);

l1++;

}

else

{

tmp.push\_back(list2[l2]);

l2++;

}

}

while (l1 < list1.size())

{

tmp.push\_back(list1[l1]);

l1++;

}

while (l2 < list2.size())

{

tmp.push\_back(list2[l2]);

l2++;

}

Print\_list(tmp);//вывод серий минимум по 2 элемента

cout << "-------------------------------------------" << endl;

return tmp;

}

void Merge\_sort(vector<int> &list)

{

if (list.size() < 2)

{

return;

}

/\*else if (list.size() == 2)

{

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

return;

}\*/

vector<int> left;

vector<int> right;

int mid = list.size() / 2;

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

{

left.push\_back(list[i]);

}

for (int i = mid; i < list.size(); i++)

{

right.push\_back(list[i]);

}

Merge\_sort(left);

Merge\_sort(right);

list = Merge(left, right);

}

//natural merge

void Print\_series(vector<vector<int>> list\_of\_series)

{

for (int i = 0; i < list\_of\_series.size(); i++)

{

cout << i + 1 << "-я серия:" << endl;

cout << "| ";

for (int j = 0; j < list\_of\_series[i].size(); j++)

{

cout << list\_of\_series[i][j] << " ";

}

cout << "|" << endl;

}

cout << endl;

}

void Split\_series(vector<vector<int>> &list\_of\_series, vector<int> list)

{

int i = 0;

while (i < list.size())

{

vector<int> line;

line.push\_back(list[i]);

i++;

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

{

line.push\_back(list[i]);

i++;

}

list\_of\_series.push\_back(line);

}

}

vector<int> Merge\_series(vector<int> list1, vector<int> list2) // Используется в естественном и многофазном слиянии

{

int l1 = 0, l2 = 0;

vector <int> tmp;

while (l1 < list1.size() && l2 < list2.size())

{

if (list1[l1] <= list2[l2])

{

tmp.push\_back(list1[l1]);

l1++;

}

else

{

tmp.push\_back(list2[l2]);

l2++;

}

}

while (l1 < list1.size())

{

tmp.push\_back(list1[l1]);

l1++;

}

while (l2 < list2.size())

{

tmp.push\_back(list2[l2]);

l2++;

}

return tmp;

}

vector<int> Natural\_merge(vector<int> list)

{

vector<vector<int>> list\_of\_series;

Split\_series(list\_of\_series, list); //разделение массива на серии по возрастанию

Print\_series(list\_of\_series); //вывод серий

while (list\_of\_series.size() >= 2)

{

vector<int> Merged = Merge\_series(list\_of\_series[0], list\_of\_series[1]);//слияние первой и второй серии

list\_of\_series.push\_back(Merged); //добавление первой и второй серии после слияния в конец массива серий

list\_of\_series.erase(list\_of\_series.begin());//удаление первой серии

list\_of\_series.erase(list\_of\_series.begin());//удаление второй серии

}

return list\_of\_series[0];

}

//Phase\_merge

void Print\_Files(vector<vector<int>> File1, vector<vector<int>> File2, vector<vector<int>> File3)

{

cout << "F1:" << endl;

Print\_series(File1);

cout << "=============================================" << endl;

cout << "F2:" << endl;

Print\_series(File2);

cout << "=============================================" << endl;

cout << "F3:" << endl;

Print\_series(File3);

cout << "=============================================" << endl;

}

void Fibb(int count\_of\_series, int &size1, int &size2)

{

size1 = 0, size2 = 1;

while (size1 + size2 < count\_of\_series)

{

int tmp = size2;

size2 += size1;

size1 = tmp;

}

}

vector<vector<int>> Split\_to\_series(vector<int> list)

{

vector<vector<int>> result;

int count;

if (list.size() < 8) {

count = 1;

}

else

{

count = list.size() / 8;

while (list.size() % count != 0)

count--; // нахождение максимального количества элементов в серии (для скорости размер массива делится на 8)

}

int i = 0;

for (int i = 0; i < list.size() / count; i++)

{

vector<int> tmp;

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

{

tmp.push\_back(list[i \* count + j]);

}

result.push\_back(tmp);

}

return result;

}

vector<vector<int>> Files\_Merge(vector<vector<int>> &F1, vector<vector<int>> &F2)

{

vector<vector<int>> result; //"входной файл"

int min;

if (F1.size() < F2.size()) //выбор наименьшего количества серий в файле

min = F1.size();

else

min = F2.size();

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

{

result.push\_back(Merge\_series(F1[i], F2[i])); // слияние 2 серий и добавление

// во входной файл

}

F1.erase(F1.begin(), F1.begin() + min); // Удаление серий,которые

F2.erase(F2.begin(), F2.begin() + min); // попали во входной файл

return result;

}

void Delete\_emptySerie(vector<int> &list)

{

while (list[0] == -100)

{

list.erase(list.begin());

}

}

vector<int> Phase\_merge(vector<int> list)

{

vector<vector<int>> series = Split\_to\_series(list);

int size1, size2; // size1 - количество серий во 2 файле(векторе) size2 - количество серий во втором файле(векторе)

vector<vector<int>> File1, File2, File3; //файлы

Fibb(series.size(), size1, size2); // получение количества серий в 1 и 2 файле

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Заполнение файлов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

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

{

File1.push\_back(series[i]);

}

if (size1 + size2 == series.size())

{

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

{

File2.push\_back(series[i]);

}

}

else

{

int tmp = size1;

for (tmp; tmp < series.size(); tmp++)

{

File2.push\_back(series[tmp]);

}

for (tmp; tmp < size1 + size2; tmp++)

{

vector<int> empty\_serie;

for (int j = 0; j < File2[0].size(); j++)

{

empty\_serie.push\_back(-100);

}

File2.push\_back(empty\_serie);

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

Print\_Files(File1, File2, File3); //вывод файлов

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Сортировка файлов\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

for (int i = 0; i < File1.size(); i++)

{

sort(File1[i].begin(), File1[i].end());

}

for (int i = 0; i < File2.size(); i++)

{

sort(File2[i].begin(), File2[i].end());

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

cout << "Файлы после сортировки:" << endl;

Print\_Files(File1, File2, File3); //вывод файлов

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Слияние\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_//

int count\_of\_emptyfiles = 1; //изначально File3

while (count\_of\_emptyfiles != 2)

{

int empty\_index = 3; // изначально 3;

if (File1.empty()) empty\_index = 1;

if (File2.empty()) empty\_index = 2;

switch (empty\_index)

{

case 1:

{

File1 = Files\_Merge(File2, File3);

break;

}

case 2:

{

File2 = Files\_Merge(File1, File3);

break;

}

case 3:

{

File3 = Files\_Merge(File1, File2);

break;

}

}

count\_of\_emptyfiles = 0;

if (File1.empty()) count\_of\_emptyfiles++;

if (File2.empty()) count\_of\_emptyfiles++;

if (File3.empty()) count\_of\_emptyfiles++;

Print\_Files(File1, File2, File3);

}

if (!File1.empty())

{

Delete\_emptySerie(File1[0]);

return File1[0];

}

if (!File2.empty())

{

Delete\_emptySerie(File2[0]);

return File2[0];

}

if (!File3.empty())

{

Delete\_emptySerie(File3[0]);

return File3[0];

}

}

int main()

{

setlocale(LC\_ALL, "ru");

srand(time(0));

int size = GetSize();

vector <int> list(size);

Rand\_vec(list);

cout << "Исходный массив:" << endl;

//Print\_list(list);

cout << "======================================================================================================================" << endl << endl;

cout << "Массив, отсортированный сбалансированным слиянием:" << endl;

cout << "Серии и конечный массив" << endl << endl;

Merge\_sort(list);

cout << "======================================================================================================================" << endl << endl;

cout << "Исходный массив:" << endl;

Rand\_vec(list);

Print\_list(list);

cout << "======================================================================================================================" << endl << endl;

cout << "Массив, отсортированный натуральным слиянием:" << endl;

list = Natural\_merge(list);

Print\_list(list);

cout << "======================================================================================================================" << endl << endl;

cout << "Исходный массив:" << endl;

Rand\_vec(list);

Print\_list(list);

cout << "======================================================================================================================" << endl << endl;

cout << "Массив, отсортированный многофазныйм слиянием:" << endl;

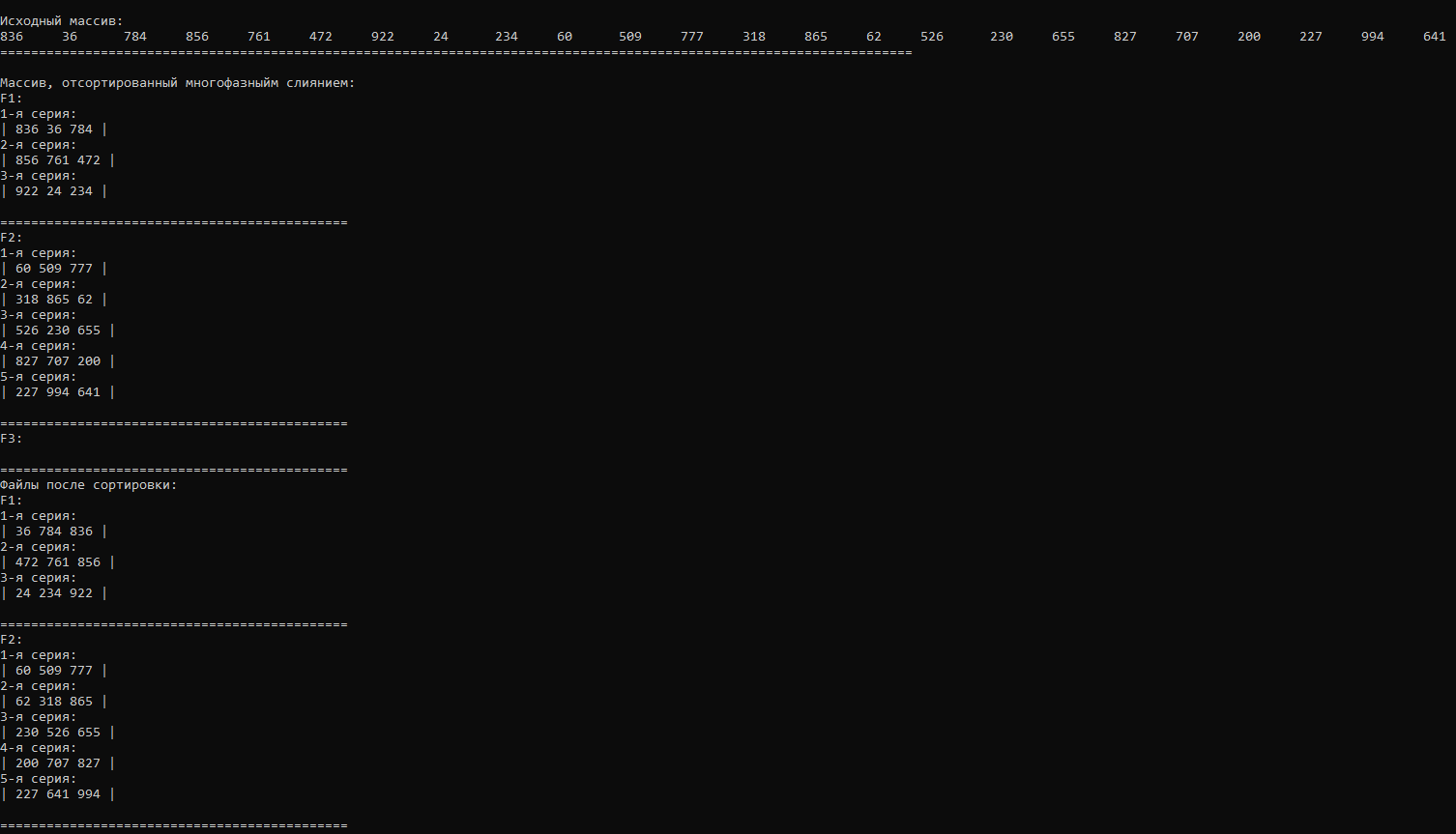
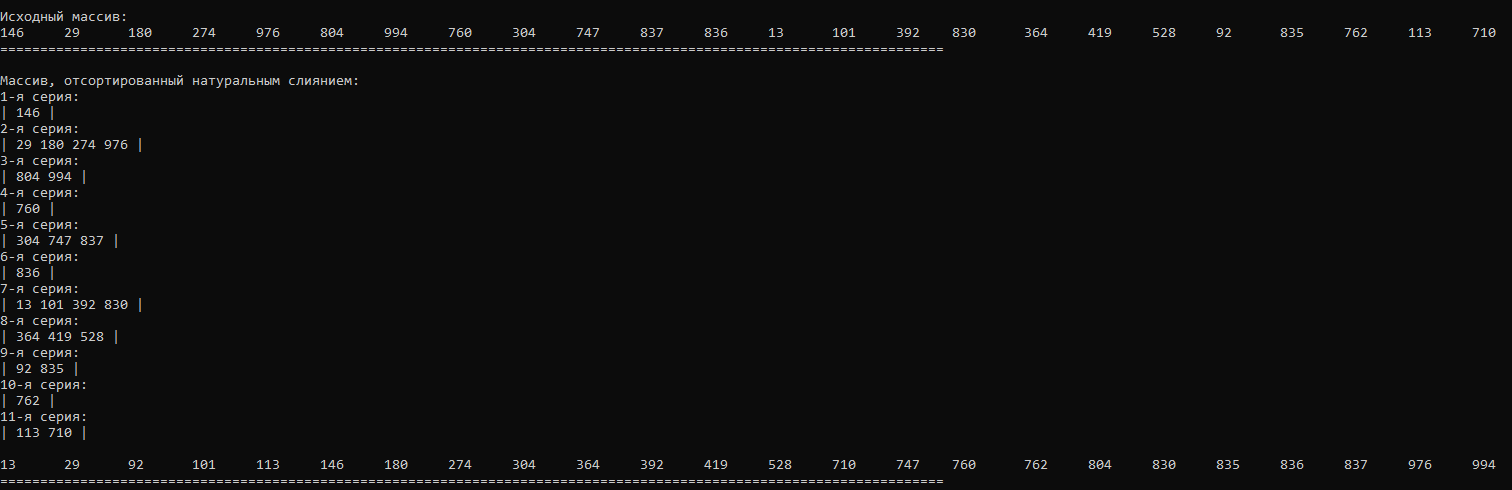
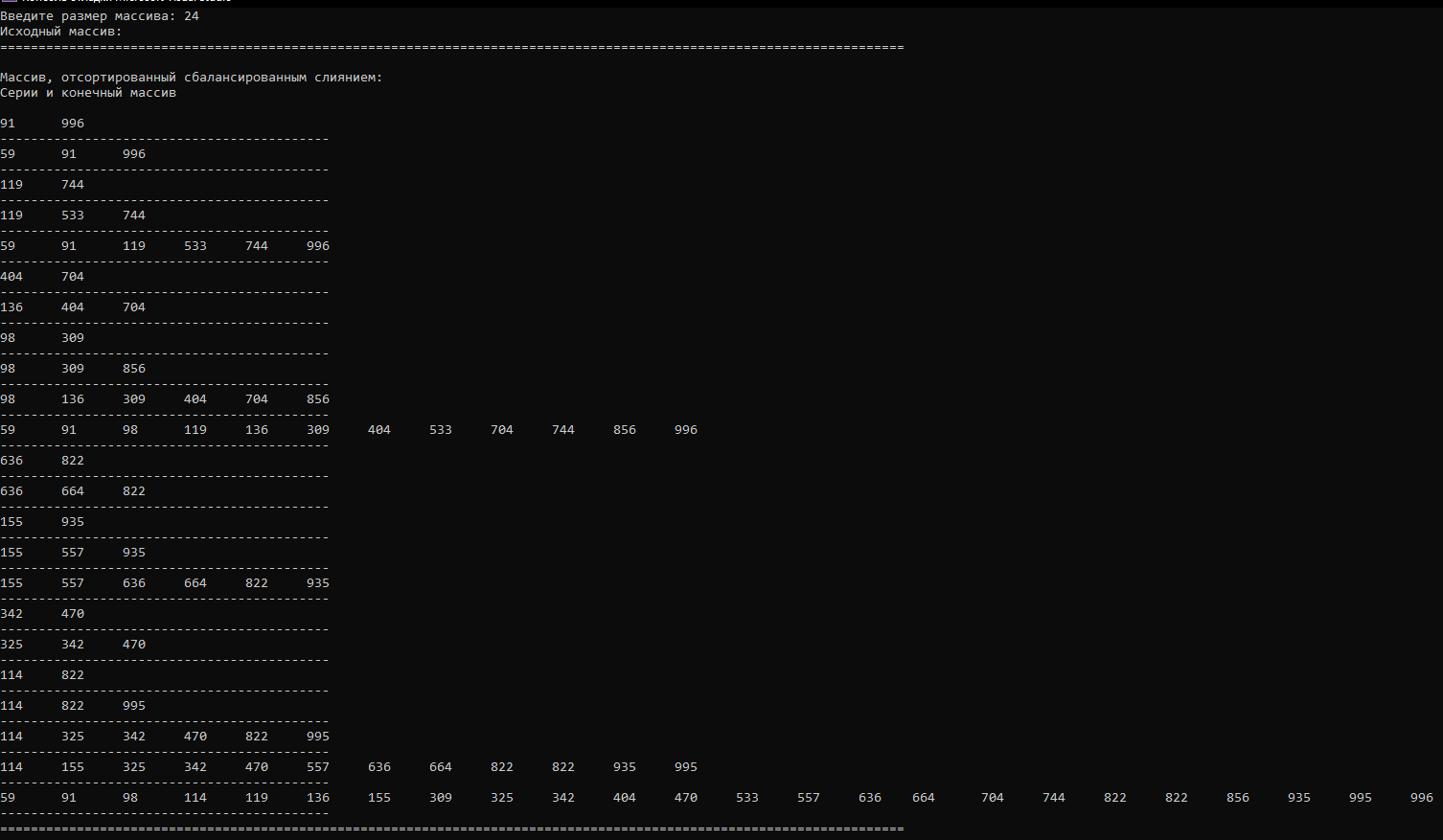
list = Phase\_merge(list);

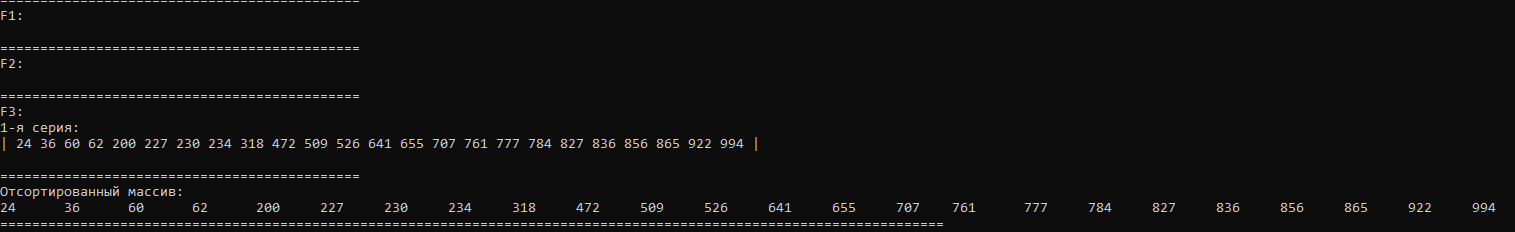
cout << "Отсортированный массив:" << endl;

Print\_list(list);

cout << "======================================================================================================================" << endl;

}

**Работа кода**  




**Блок-схема**

