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”

Выполнил:

студент группы ИВТ-20-2Б

Чувашев Максим

Проверила: доцент кафедры ИТАС

Полякова О.А.

Пермь, 2021

**Постановка задачи**

Отсортировать множество элементов по возрастанию методами естественного, сбалансированного и многофазного слияния.

**Анализ задачи**

1. Для решения задачи необходимо:
   1. Реализовать функцию void InitSize(int\* size) для инициализации размера массива структур
   2. Реализовать функцию void InitArr(int\* arr, int\* size) для инициализации массива структур
   3. Реализовать функцию void PrintArr(int\* arr, int size) для печати массива объектов структур в консоль
   4. Реализовать функцию void Balance(int\* arr, int size, int l, int r) для реализации сортировки методом сбалансированного слияния
   5. Реализовать функцию void printVE(vector<vector<int>> VECT) для печати отсортированного массива
   6. Реализовать функцию void Naturalfusion(int\* arr, int\* size, vector<vector<int>>\* VECT) для реализации сортировки натуральным слиянием
   7. Реализовать функцию void Polyphasesort(int \*arr1, int kol), которая реализует многофазную сортировку
2. Для решения задачи данные были представлены в следующем виде:
   1. Вектор arr для хранения всех элементов множества.
3. Для операций ввода и вывода использовались следующие операторы и функции:
   1. Ввод и вывод производится с помощью операторов ввода и вывода cin и cout.

cout << "Введите размер массива:\t";

cin >> \*size;

1. Поставленные задачи будут решены следующими действиями:
   1. В функции void InitSize(int\* size) пользователь вводит размер массива структур

void InitSize(int\* size)

{

cout << "Введите размер массива:\t";

cin >> \*size;

while ((\*size) < 1)

{

cout << "Размер должен быть натуральным числом!" << endl;

cout << "Введите размер массива:\t";

cin >> \*size;

}

}

* 1. В функции void InitArr(int\* arr, int\* size) производится инициализация массива случайными числами

void InitArr(int\* arr, int\* size)

{

int r;

cout << "Введите число до которого будут генерироваться числа:\t";

cin >> r;

while (r < 1)

{

cout << "Число должно быть натуральным числом!" << endl;

cout << "Введите число до которого будут генерироваться числа:\t";

cin >> r;

}

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

{

arr[i] = rand() % r;

}

}

* 1. В функции void PrintArr(int\* arr, int size) производится вывод всех элементов массива в консоль

void PrintArr(int\* arr, int size)

{

cout << endl;

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

{

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

}

cout << endl;

}

* 1. В функции void Balance(int\* arr, int size, int l, int r) производится сортировка с помощью сбалансированного слияния.

void Balance(int\* arr, int size, int l, int r)

{

if (r == l)

{

return;

}

if ((r - l) == 1) {

if (arr[r] < arr[l])

{

swap(arr[r], arr[l]);

}

return;

}

int m = (r + l) / 2;

Balance(arr, size, l, m);

Balance(arr, size, m + 1, r);

int\* buf = new int[size];

int xl = l;

int xr = m + 1;

int cur = 0;

while (r - l + 1 != cur) {

if (xl > m)

{

buf[cur++] = arr[xr++];

}

else if (xr > r)

{

buf[cur++] = arr[xl++];

}

else if (arr[xl] > arr[xr])

{

buf[cur++] = arr[xr++];

}

else

{

buf[cur++] = arr[xl++];

}

}

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

{

arr[i + l] = buf[i];

}

}

* 1. В функции void printVE(vector<vector<int>> VECT) печатается отсортированный массив

void printVE(vector<vector<int>> VECT)

{

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

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

{

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

{

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

}

}

cout << endl << endl;

}

* 1. В функции void Naturalfusion(int\* arr, int\* size, vector<vector<int>>\* VECT) производится сортировка натуральным слиянием. Создается вспомогательный вектор, далее проходимся и заполняем его. Далее соединяем в серии, сравниваем серии и записываем в новый вектор результат

void Naturalfusion(int\* arr, int\* size, vector<vector<int>>\* VECT)

{

vector<vector<int>> VHELP;

int i = 0;

while (i < \*size)

{

vector<int> HELP;

HELP.push\_back(arr[i]);

i++;

while (i < (\*size) && arr[i - 1] < arr[i])

{

HELP.push\_back(arr[i]);

i++;

}

VHELP.push\_back(HELP);

}

int k = 0;

while (VHELP.size() > 1)

{

vector<int> splitlist;

vector<int> list1 = VHELP[k];

vector<int> list2 = VHELP[k + 1];

int n1 = 0;

int n2 = 0;

while (n1 < list1.size() && n2 < list2.size())

{

if (list1[n1] <= list2[n2])

{

splitlist.push\_back(list1[n1]);

n1++;

}

else

{

splitlist.push\_back(list2[n2]);

n2++;

}

}

while (n1 < list1.size())

{

splitlist.push\_back(list1[n1]);

n1++;

}

while (n2 < list2.size())

{

splitlist.push\_back(list2[n2]);

n2++;

}

VHELP.push\_back(splitlist);

VHELP.erase(VHELP.begin());

VHELP.erase(VHELP.begin());

}

\*VECT = VHELP;

VHELP.clear();

printVE(\*VECT);

}

* 1. Функция void PrintSeries(vector<vector<int>> series) которая печатает серии

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

{

cout << series.size() << ":\t";

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;

}

* 1. Обмен происходит в функции Transition(), где в цикле for выполняется заполнение пустого файла из файла, который заполнен меньшим (min) количеством серий по сравнению с другим (max) и удаление того же количества серий из файла с большим количеством серий (max)

void Transition(vector<vector<int>>& min, vector<vector<int>>& max, vector<vector<int>>& null)

{

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

vector<int> serie1 = max[i], serie2 = min[i];

vector<int> newSerie = serie1;

for (int j = 0; j < serie2.size(); j++) {

newSerie.push\_back(serie2[j]);

}

InsertionSort(newSerie);

null.push\_back(newSerie);

}

max.erase(max.begin(), max.begin() + min.size());

min.clear();

}

* 1. Генерация по количеству серий в файлах F1, F2, F3

void Fib(int& size1, int& size2, vector<int>& arr)

{

size1 = 0;

size2 = 1;

while (size1 + size2 < arr.size()) {

int tmp = size2;

size2 = size2 + size1;

size1 = tmp;

}

}

* 1. Функция PolyphaseMergeSort(), в цикле while производится заполнение вектора векторов List по сериям по признаку возрастания. Далее идет заполнение файлов F1, F2 сериями. Далее в цикле while осуществляется обмен между файлами до того момента пока два файла не станут пустыми (признак окончания сортировки).

void Polyphasesort(int \*arr1, int kol)

{

vector<int> arr;

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

{

arr.push\_back(arr1[i]);

}

vector<vector<int>>List;

bool flag = true;

int i = 0;

while (i < arr.size()) {

vector<int> tmp;

tmp.push\_back(arr[i]);

i++;

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

tmp.push\_back(arr[i]);

i++;

}

List.push\_back(tmp);

}

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

//числа фибоначи

int size1, size2;

Fib(size1, size2, arr);

//инициализация f1, f2

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

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

}

for (int i = 0; i < List.size() - size1; i++) {

F2.push\_back(List[i + size1]);

}

// заполнение пустыми сериями

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

vector<int> serie;

serie.push\_back(-1);

F2.push\_back(serie);

}

//вывод на консоль заполненного F1

cout << "F1 " << endl;

PrintSeries(F1);

//вывод на консоль заполненного F2

cout << endl << "F2 " << endl;

PrintSeries(F2);

cout << endl;

//цикл, в котором осуществляется обемен между файлами

while (!(F1.size() == 0 && F2.size() == 0 || F2.size() == 0 && F3.size() == 0 || F3.size() == 0 && F1.size() == 0)) {

if ((F1.size() >= F2.size() && F3.size() == 0))

{

Transition(F2, F1, F3);

}

if ((F2.size() >= F1.size() && F3.size() == 0))

{

Transition(F1, F2, F3);

}

if ((F3.size() >= F2.size() && F1.size() == 0))

{

Transition(F2, F3, F1);

}

if ((F2.size() >= F3.size() && F1.size() == 0))

{

Transition(F3, F2, F1);

}

if ((F1.size() >= F3.size() && F2.size() == 0))

{

Transition(F3, F1, F2);

}

if ((F3.size() >= F1.size() && F2.size() == 0))

{

Transition(F1, F3, F2);

}

if (F1.size() != 0)

{

cout << "\nF1" << endl;

PrintSeries(F1);

}

if (F2.size() != 0)

{

cout << "\nF2" << endl;

PrintSeries(F2);

}

if (F3.size() != 0)

{

cout << "\nF3" << endl;

PrintSeries(F3);

}

}

cout << "\nИзменённый массив:";

if (F1.size() != 0)

{

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

{

if (F1[0][i] != -1) {

cout << F1[0][i] << " ";

}

}

}

if (F2.size() != 0)

{

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

{

if (F2[0][i] != -1)

cout << F2[0][i] << " ";

}

}

if (F3.size() != 0)

{

for (int i = 0; i < F3[0].size(); i++)

{

if (F3[0][i] != -1)

cout << F3[0][i] << " ";

}

}

cout << endl;

}

* 1. В функции bool Choice(int\* arr, int\* size, bool f, int foo, int f1, vector<vector<int>>\* VECT) пользователь производит выбор действий.

bool Choice(int\* arr, int\* size, bool f, int foo, int f1, vector<vector<int>>\* VECT)

{

switch (foo)

{

case 1: InitArr(arr, size); break;

case 2: PrintArr(arr, \*size); break;

case 3: Balance(arr, \*size, 0, (\*size - 1)); break;

case 4: Naturalfusion(arr, size, VECT); break;

case 5: Polyphasesort(arr, \*size); break;

case 6: f = true; break;

default:

break;

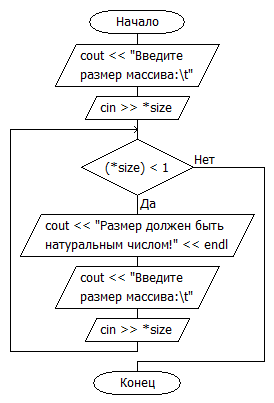
}

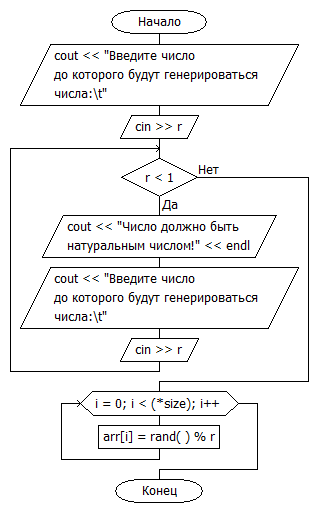
return f;

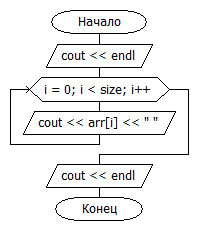
}

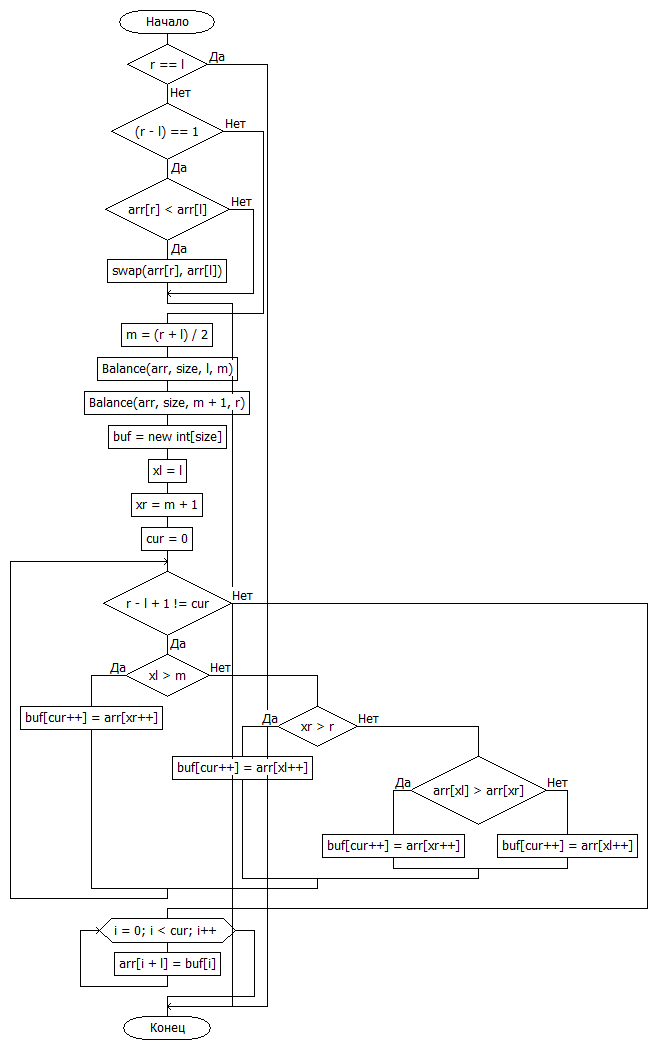
**Блок-схема**

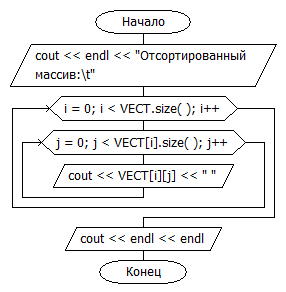


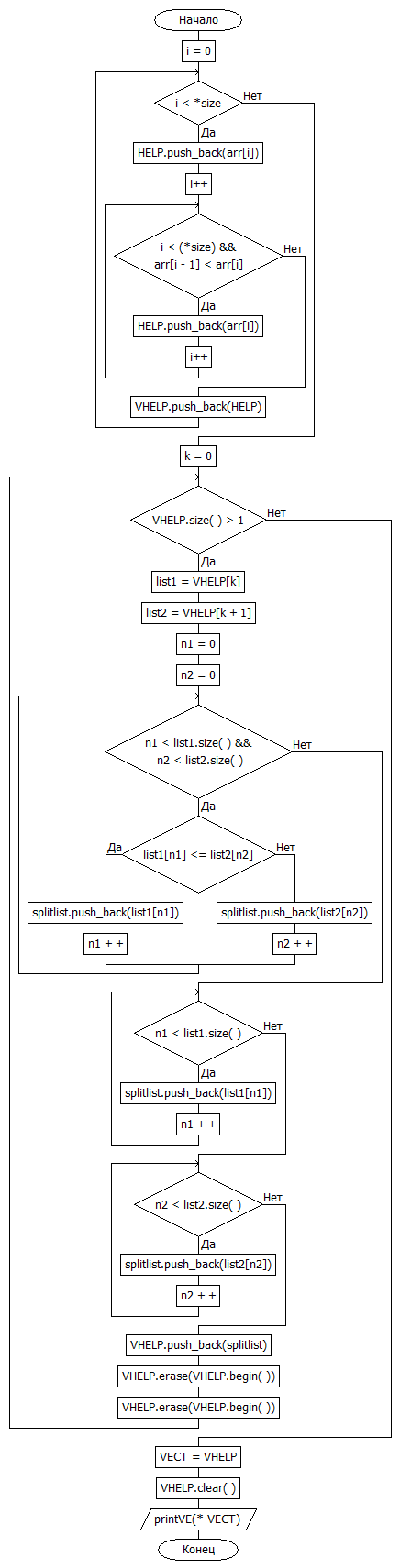


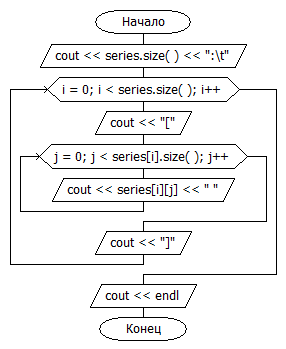


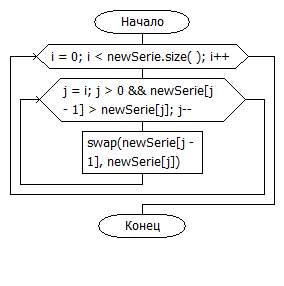


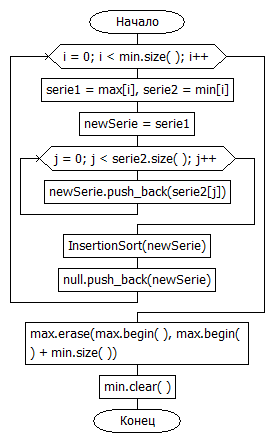


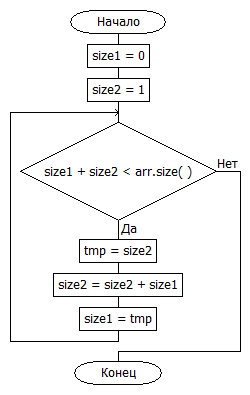


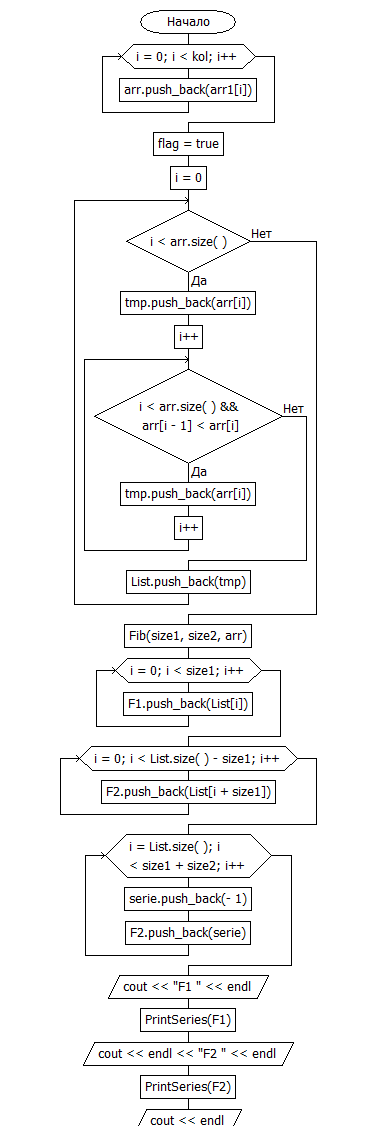


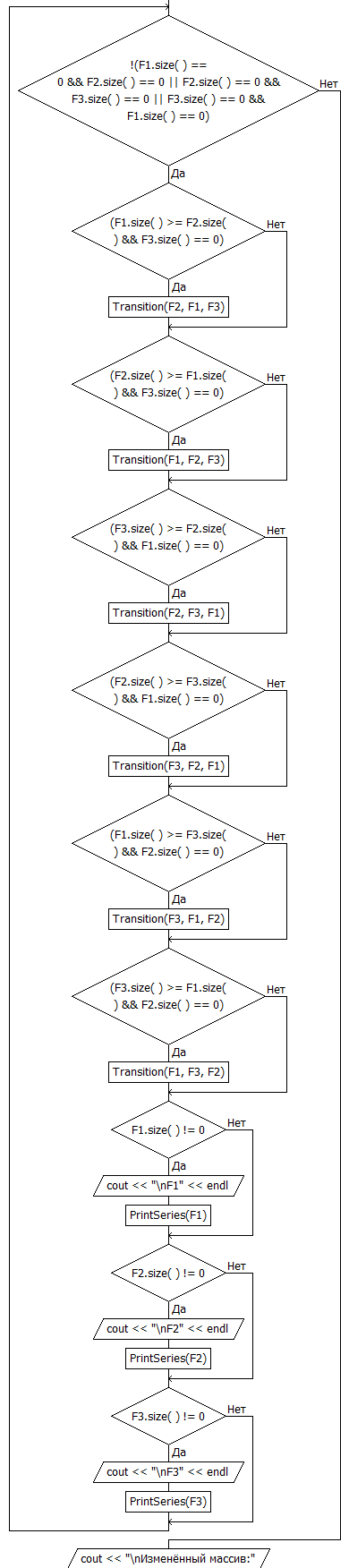


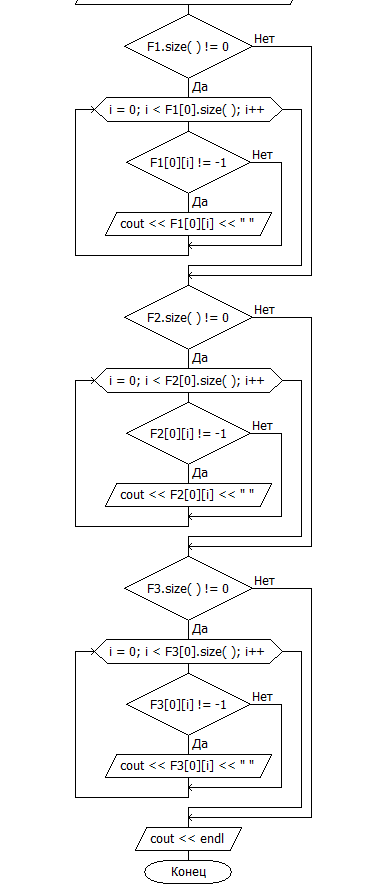


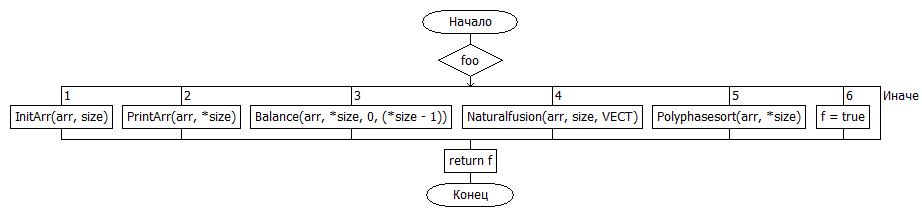


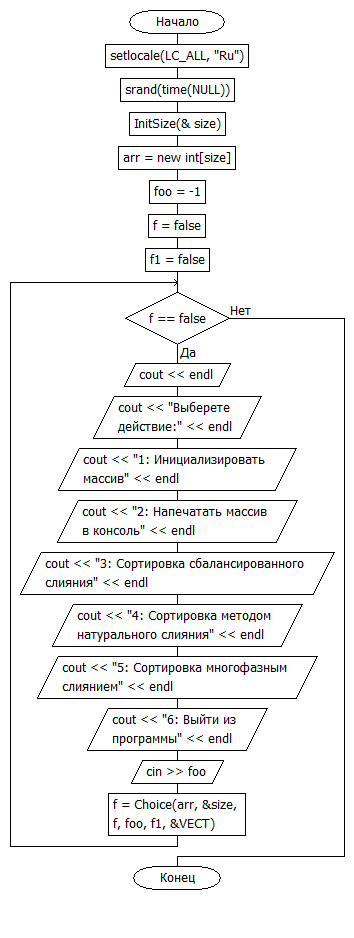












**Код**

#include <iostream>

#include <ctime>

#include <algorithm>

#include <vector>

using namespace std;

struct HELP

{

int ind1;

int ind2;

vector <int> ve1;

vector <int> ve2;

vector <int> ve3;

};

struct FIB

{

int f1[6] = { 5, 0, 3, 1, 0, 1 };

int f2[6] = { 8, 3, 0, 2, 1, 0 };

int f3[6] = { 0, 5, 2, 0, 1, 0 };

};

void InitSize(int\* size)

{

cout << "Введите размер массива:\t";

cin >> \*size;

while ((\*size) < 1)

{

cout << "Размер должен быть натуральным числом!" << endl;

cout << "Введите размер массива:\t";

cin >> \*size;

}

}

void InitArr(int\* arr, int\* size)

{

int r;

cout << "Введите число до которого будут генерироваться числа:\t";

cin >> r;

while (r < 1)

{

cout << "Число должно быть натуральным числом!" << endl;

cout << "Введите число до которого будут генерироваться числа:\t";

cin >> r;

}

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

{

arr[i] = rand() % r;

}

}

void PrintArr(int\* arr, int size)

{

cout << endl;

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

{

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

}

cout << endl;

}

void Balance(int\* arr, int size, int l, int r)

{

if (r == l)

{

return;

}

if ((r - l) == 1) {

if (arr[r] < arr[l])

{

swap(arr[r], arr[l]);

}

return;

}

int m = (r + l) / 2;

Balance(arr, size, l, m);

Balance(arr, size, m + 1, r);

int\* buf = new int[size];

int xl = l;

int xr = m + 1;

int cur = 0;

while (r - l + 1 != cur) {

if (xl > m)

{

buf[cur++] = arr[xr++];

}

else if (xr > r)

{

buf[cur++] = arr[xl++];

}

else if (arr[xl] > arr[xr])

{

buf[cur++] = arr[xr++];

}

else

{

buf[cur++] = arr[xl++];

}

}

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

{

arr[i + l] = buf[i];

}

}

void printVE(vector<vector<int>> VECT)

{

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

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

{

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

{

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

}

}

cout << endl << endl;

}

void Naturalfusion(int\* arr, int\* size, vector<vector<int>>\* VECT)

{

vector<vector<int>> VHELP;

int i = 0;

while (i < \*size)

{

vector<int> HELP;

HELP.push\_back(arr[i]);

i++;

while (i < (\*size) && arr[i - 1] < arr[i])

{

HELP.push\_back(arr[i]);

i++;

}

VHELP.push\_back(HELP);

}

int k = 0;

while (VHELP.size() > 1)

{

vector<int> splitlist;

vector<int> list1 = VHELP[k];

vector<int> list2 = VHELP[k + 1];

int n1 = 0;

int n2 = 0;

while (n1 < list1.size() && n2 < list2.size())

{

if (list1[n1] <= list2[n2])

{

splitlist.push\_back(list1[n1]);

n1++;

}

else

{

splitlist.push\_back(list2[n2]);

n2++;

}

}

while (n1 < list1.size())

{

splitlist.push\_back(list1[n1]);

n1++;

}

while (n2 < list2.size())

{

splitlist.push\_back(list2[n2]);

n2++;

}

VHELP.push\_back(splitlist);

VHELP.erase(VHELP.begin());

VHELP.erase(VHELP.begin());

}

\*VECT = VHELP;

VHELP.clear();

printVE(\*VECT);

}

vector<vector<int>> Splitser(vector<int> HELP, int size1, int size2)

{

vector<vector<int>> hsort(size1 + size2);

int count;

if (HELP.size() < (size1 + size2))

{

count = 1;

}

else

{

if (HELP.size() % (size1 + size2) == 0)

{

count = HELP.size() / (size1 + size2);

}

else

{

count = HELP.size() / (size1 + size2);;

count++;

}

}

int c = 0;

int k = 0;

while (c < (size1 + size2))

{

vector<int> HH(count);

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

{

if (k < HELP.size())

{

HH[i] = HELP[k];

k++;

}

else

{

HH[i] = -1;

}

}

hsort[c] = HH;

c++;

}

return hsort;

}

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

{

cout << series.size() << ":\t";

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 InsertionSort(vector<int>& newSerie)

{

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

for (int j = i; j > 0 && newSerie[j - 1] > newSerie[j]; j--) {

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

}

}

}

void Transition(vector<vector<int>>& min, vector<vector<int>>& max, vector<vector<int>>& null)

{

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

vector<int> serie1 = max[i], serie2 = min[i];

vector<int> newSerie = serie1;

for (int j = 0; j < serie2.size(); j++) {

newSerie.push\_back(serie2[j]);

}

InsertionSort(newSerie);

null.push\_back(newSerie);

}

max.erase(max.begin(), max.begin() + min.size());

min.clear();

}

void Fib(int& size1, int& size2, vector<int>& arr)

{

size1 = 0;

size2 = 1;

while (size1 + size2 < arr.size()) {

int tmp = size2;

size2 = size2 + size1;

size1 = tmp;

}

}

void Polyphasesort(int \*arr1, int kol)

{

vector<int> arr;

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

{

arr.push\_back(arr1[i]);

}

vector<vector<int>>List;

bool flag = true;

int i = 0;

while (i < arr.size()) {

vector<int> tmp;

tmp.push\_back(arr[i]);

i++;

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

tmp.push\_back(arr[i]);

i++;

}

List.push\_back(tmp);

}

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

//числа фибоначи

int size1, size2;

Fib(size1, size2, arr);

//инициализация f1, f2

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

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

}

for (int i = 0; i < List.size() - size1; i++) {

F2.push\_back(List[i + size1]);

}

// заполнение пустыми сериями

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

vector<int> serie;

serie.push\_back(-1);

F2.push\_back(serie);

}

//вывод на консоль заполненного F1

cout << "F1 " << endl;

PrintSeries(F1);

//вывод на консоль заполненного F2

cout << endl << "F2 " << endl;

PrintSeries(F2);

cout << endl;

//цикл, в котором осуществляется обемен между файлами

while (!(F1.size() == 0 && F2.size() == 0 || F2.size() == 0 && F3.size() == 0 || F3.size() == 0 && F1.size() == 0)) {

if ((F1.size() >= F2.size() && F3.size() == 0))

{

Transition(F2, F1, F3);

}

if ((F2.size() >= F1.size() && F3.size() == 0))

{

Transition(F1, F2, F3);

}

if ((F3.size() >= F2.size() && F1.size() == 0))

{

Transition(F2, F3, F1);

}

if ((F2.size() >= F3.size() && F1.size() == 0))

{

Transition(F3, F2, F1);

}

if ((F1.size() >= F3.size() && F2.size() == 0))

{

Transition(F3, F1, F2);

}

if ((F3.size() >= F1.size() && F2.size() == 0))

{

Transition(F1, F3, F2);

}

if (F1.size() != 0)

{

cout << "\nF1" << endl;

PrintSeries(F1);

}

if (F2.size() != 0)

{

cout << "\nF2" << endl;

PrintSeries(F2);

}

if (F3.size() != 0)

{

cout << "\nF3" << endl;

PrintSeries(F3);

}

}

cout << "\nИзменённый массив:";

if (F1.size() != 0)

{

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

{

if (F1[0][i] != -1) {

cout << F1[0][i] << " ";

}

}

}

if (F2.size() != 0)

{

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

{

if (F2[0][i] != -1)

cout << F2[0][i] << " ";

}

}

if (F3.size() != 0)

{

for (int i = 0; i < F3[0].size(); i++)

{

if (F3[0][i] != -1)

cout << F3[0][i] << " ";

}

}

cout << endl;

}

bool Choice(int\* arr, int\* size, bool f, int foo, int f1, vector<vector<int>>\* VECT)

{

switch (foo)

{

case 1: InitArr(arr, size); break;

case 2: PrintArr(arr, \*size); break;

case 3: Balance(arr, \*size, 0, (\*size - 1)); break;

case 4: Naturalfusion(arr, size, VECT); break;

case 5: Polyphasesort(arr, \*size); break;

case 6: f = true; break;

default:

break;

}

return f;

}

int main()

{

setlocale(LC\_ALL, "Ru");

srand(time(NULL));

int size;

InitSize(&size);

int\* arr = new int[size];

vector<vector<int>> VECT;

int foo = -1;

bool f = false;

bool f1 = false;

while (f == false)

{

cout << endl;

cout << "Выберете действие:" << endl;

cout << "1: Инициализировать массив" << endl;

cout << "2: Напечатать массив в консоль" << endl;

cout << "3: Сортировка сбалансированного слияния" << endl;

cout << "4: Сортировка методом натурального слияния" << endl;

cout << "5: Сортировка многофазным слиянием" << endl;

cout << "6: Выйти из программы" << endl;

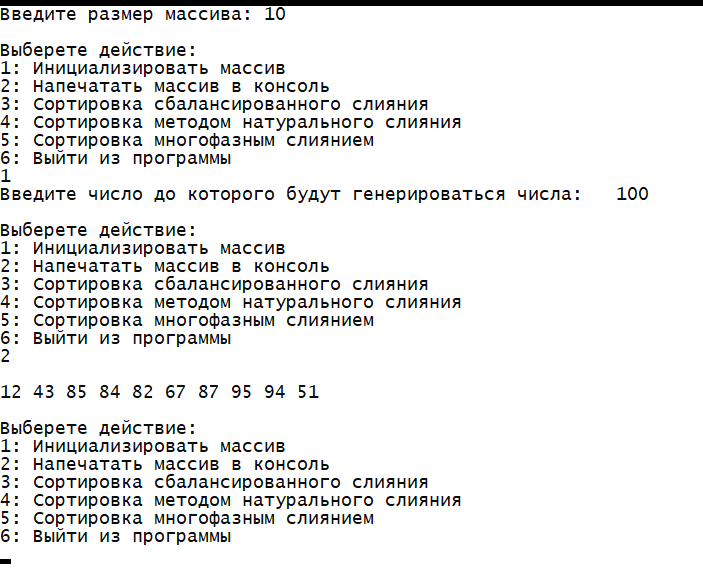
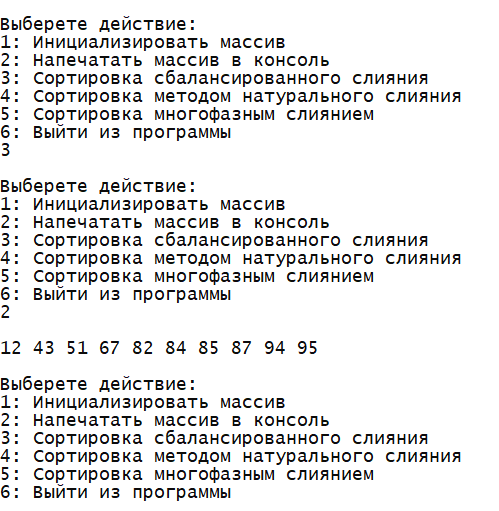
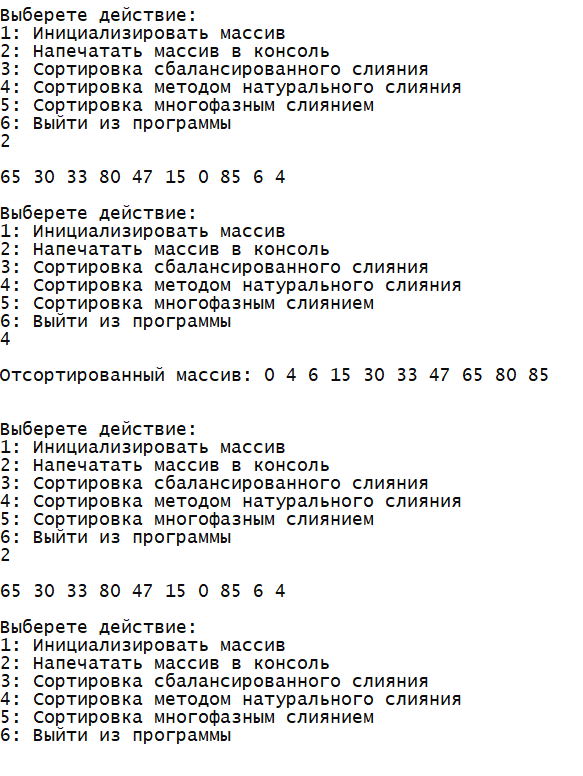
cin >> foo;

f = Choice(arr, &size, f, foo, f1, &VECT);

}

}

**Скриншоты**

* 1. 
  2. 
  3. 
  4. 