Министерство образования и науки Российской Федерации  
Федеральное‌ ‌государственное‌ ‌бюджетное‌ ‌образовательное‌ ‌учреждение‌

высшего‌ ‌образования‌

**«Пермский национальный исследовательский  
политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЁТ**

**по лабораторной работе №16**

Дисциплина: основы алгоритмизации и программирования

Тема: Внешние сортировки

Выполнила:

Студентка группы ИВТ-20-2б

Ананина Арина Юрьевна

Проверил:

Доцент кафедры ИТАС

Полякова О. А.

**Пермь, 2020**

1. Постановка задачи

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

2. Анализ задачи

1. Используемые типы данных.

1.1 int - для хранения целых чисел массива и его размера.

1.2 vector - динамическая структура для хранения массива целых чисел.  
vector <int> list(size);

2. Действия над используемыми данными.

2.1 Переменные типа int записываются в вектор и хранят числа, которые необходимо отсортировать.

3. Виды данных в программе.

3.1 Для решения задачи данные были представлены в следующем виде:

Vector<int> - динамический массив целых чисел.

4. Структуры в программе не используются.

5. Организация ввода и вывода.

5.1 Заполнение массива происходит с помощью датчика случайных чисел.

void CreatArray(vector<int>& list)

{

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

{

list[i] = rand() % 1000;

}

}

5.2 Вывод осуществляется в консоль через оператор cout.

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

{

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

}

6. Действия для решения задачи.

6.1 Метод сбалансированного слияния.

Функция balanced\_sort получает в качестве входных параметров ссылку на первый элемент вектора. Эта функция является рекурсивной, она делит исходный вектор пополам, и каждую из половины также пополам пока в делимых векторах не останется два или один элемент. Далее начинается слияние каждой части из разделенных , при этом осуществляется сортировка этих частей с помощью функции Merger.

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++;

}

6.2 Метод естественного слияния.

Функция natural\_sort в качестве входных параметров принимает вектор, в вектор векторов записываются серии из входного вектора с помощью функции Split\_series. Разделение на серии происходит по принципу, в серию записываются числа по порядку до тех пор, пока идет возрастание, как только оно нарушается, происходит запись в следующую серию. Далее в главной функции осуществляется проход по вектору векторов с помощью цикла пока его размер не станет равен одному (это и будет уже отсортированный список). Внутри цикла происходит слияние двух первых серий, и слившаяся серия перемещается в конец вектора векторов, после эти две первые серии удаляются. Слияние происходит с помощью функции Merge\_series по тому же принципу, что и слияние в п.6.1. Функция возвращает первый элемент вектора векторов, это и есть отсортированный список чисел.

vector<int> natural\_sort(vector<int> list)

{

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

Split\_series(list\_of\_series, list);

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

{

vector<int> Merged = Merge\_series(list\_of\_series[0], list\_of\_series[1]);

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

list\_of\_series.erase(list\_of\_series.begin());

list\_of\_series.erase(list\_of\_series.begin());

}

return list\_of\_series[0];

6.3 Метод многофазного слияния.

Исходный вектор делится на серии по несколько чисел в каждой, кол-во чисел в каждой должно быть одно и больше 8, по этому принципу ищется нужное число. Происходит разделение вектора на серии, и каждая из серий сортируется.

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

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

}

Далее необходимо найти два числа Фибаначи для числа серий, это осуществляется с помощью функции fib.

void fib(int length, int& size1, int& size2)

{

int t;

size1 = 0;

size2 = 1;

do {

t = size2;

size2 = size1 + size2;

size1 = t;

} while (size1 + size2 < length);

}

Далее серии записываются в вектор векторов векторов, назовём их линии. В первой линии больше всего серий, во второй остальное число , а третья линия пустая. При чем числа Фибаначи были взяты так, чтобы в сумме они превышали кол-во серий, а при заполнении линий недостающие серии были заменены пустыми сериями(заполненными нулями). Из первой и второй линий берется число серий, равных второй линии и записывается в третью линию. После, третья линия перемещается в начало и шаги повторяются до тех пор, пока вторая линия не будет пуста, это будет говорить о том, что список отсортирован и находится в первой линии. Остается только удалить все пустые серии, и сортировка закончена.

// Заполняем линии

vector<vector<int>> line1;

for (int i = 0; i < size2; i++) {

line1.push\_back(series[i]);

}

lines.push\_back(line1);

vector<vector<int>> line2;

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

line2.push\_back((i < series.size()) ? series[i] : get\_empty(seriesSize));

}

lines.push\_back(line2);

vector<vector<int>> line3;

lines.push\_back(line3);

// Проходы

while (!lines[1].empty()) {

lines[2] = merge\_lines(lines[0], lines[1]);

lines.insert(lines.begin(), lines[2]);

lines.pop\_back();

}

int lastEmptyIndex = 0;

while (lines[0][0][lastEmptyIndex] == NULL) {

lastEmptyIndex++;

}

lines[0][0].erase(lines[0][0].begin(), lines[0][0].begin() + lastEmptyIndex);

3. Код программы

#include <iostream>

#include <vector>

#include <ctime>

#include <algorithm>

#include <windows.h>

using namespace std;

void PrintArray(vector<int> list)

{

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

{

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

}

cout << endl;

}

void CreatArray(vector<int>& list)

{

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

{

list[i] = rand() % 1000;

}

}

vector<int> Merger(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++;

}

return tmp;

}

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

{

if (list.size() < 2)

{

return;

}

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

{

if (list[0] > list[1])

{

int tmp = list[0];

list[0] = list[1];

list[1] = tmp;

}

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]);

}

balanced\_sort(left);

balanced\_sort(right);

list = Merger(left, right);

}

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)

{

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\_sort(vector<int> list)

{

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

Split\_series(list\_of\_series, list);

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

{

vector<int> Merged = Merge\_series(list\_of\_series[0], list\_of\_series[1]);

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

list\_of\_series.erase(list\_of\_series.begin());

list\_of\_series.erase(list\_of\_series.begin());

}

return list\_of\_series[0];

}

void fib(int length, int& size1, int& size2)

{

int t;

size1 = 0;

size2 = 1;

do {

t = size2;

size2 = size1 + size2;

size1 = t;

} while (size1 + size2 < length);

}

vector<vector<int>> split\_to\_series(vector<int> list, int& seriesSize)

{

vector<vector<int>> result;

int length = 8;

if (list.size() >= length) {

while (list.size() % length != 0) {

length++;

}

}

else {

length = list.size();

}

seriesSize = list.size() / length;

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

vector<int> series;

for (int j = 0; j < seriesSize; j++) {

series.push\_back(list[i \* seriesSize + j]);

}

result.push\_back(series);

}

return result;

}

vector<int> get\_empty(int size)

{

vector<int> result;

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

result.push\_back(NULL);

}

return result;

}

vector<vector<int>> merge\_lines(vector<vector<int>>& series1, vector<vector<int>>& series2)

{

vector<vector<int>> result;

int size = series2.size();

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

result.push\_back(Merge\_series(series1[i], series2[i]));

}

series1.erase(series1.begin(), series1.begin() + size);

series2.clear();

return result;

}

vector<int> phase\_sort(vector<int> list)

{

if (!list.size()) {

return list;

}

vector<vector<vector<int>>> lines;

int seriesSize;

vector<vector<int>> series = split\_to\_series(list, seriesSize);

// Находим размеры линий

int size1, size2;

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

// Сортируем серии

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

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

}

// Заполняем линии

vector<vector<int>> line1;

for (int i = 0; i < size2; i++) {

line1.push\_back(series[i]);

}

lines.push\_back(line1);

vector<vector<int>> line2;

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

line2.push\_back((i < series.size()) ? series[i] : get\_empty(seriesSize));

}

lines.push\_back(line2);

vector<vector<int>> line3;

lines.push\_back(line3);

// Проходы

while (!lines[1].empty()) {

lines[2] = merge\_lines(lines[0], lines[1]);

lines.insert(lines.begin(), lines[2]);

lines.pop\_back();

}

int lastEmptyIndex = 0;

while (lines[0][0][lastEmptyIndex] == NULL) {

lastEmptyIndex++;

}

lines[0][0].erase(lines[0][0].begin(), lines[0][0].begin() + lastEmptyIndex);

return lines[0][0];

}

int main()

{

system("color F0");

setlocale(LC\_ALL, "RUS");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

int size;

do

{

cout << "Введите кол-во чисел в списке: ";

cin >> size;

} while (size <= 0);

vector <int> list(size);

CreatArray(list);

PrintArray(list);

int vvod;

cout << "1 - Сортировка методом сбалансированного слияния " << endl;

cout << "2 - Сортировка методом естественного слияния " << endl;

cout << "3 - Сортировка методом многофазного слияния " << endl;

cin >> vvod;

if (vvod == 1)

{

cout << "Отсортированный список: " << endl;

balanced\_sort(list);

PrintArray(list);

}

else if (vvod == 2)

{

cout << "Отсортированный список: " << endl;

list = natural\_sort(list);

PrintArray(list);

}

else if (vvod == 3)

{

cout << "Отсортированный список: " << endl;

list = phase\_sort(list);

PrintArray(list);

}

else

{

cout << "Введено некорректное значение:(" << endl;

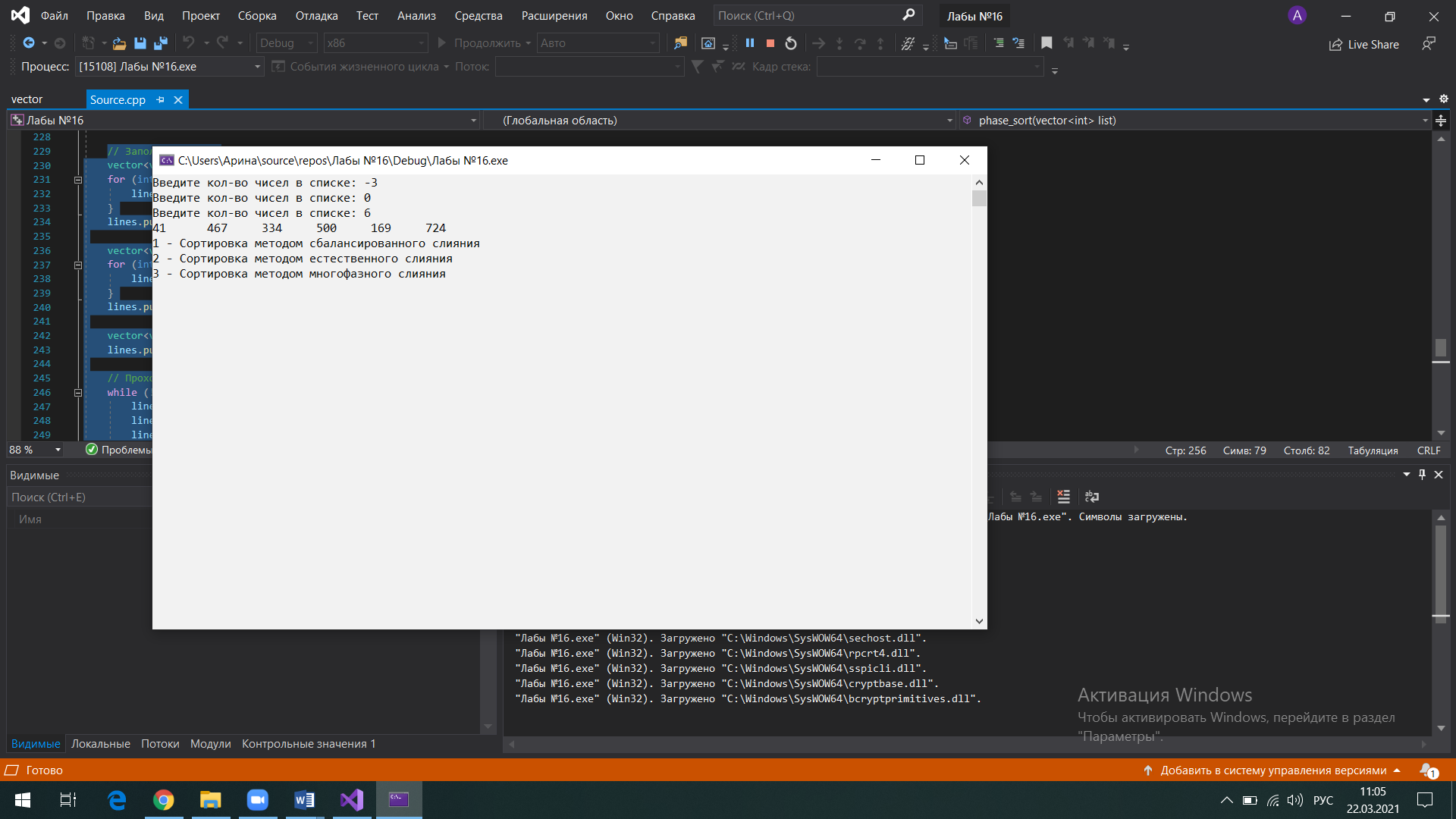
}

return 0;

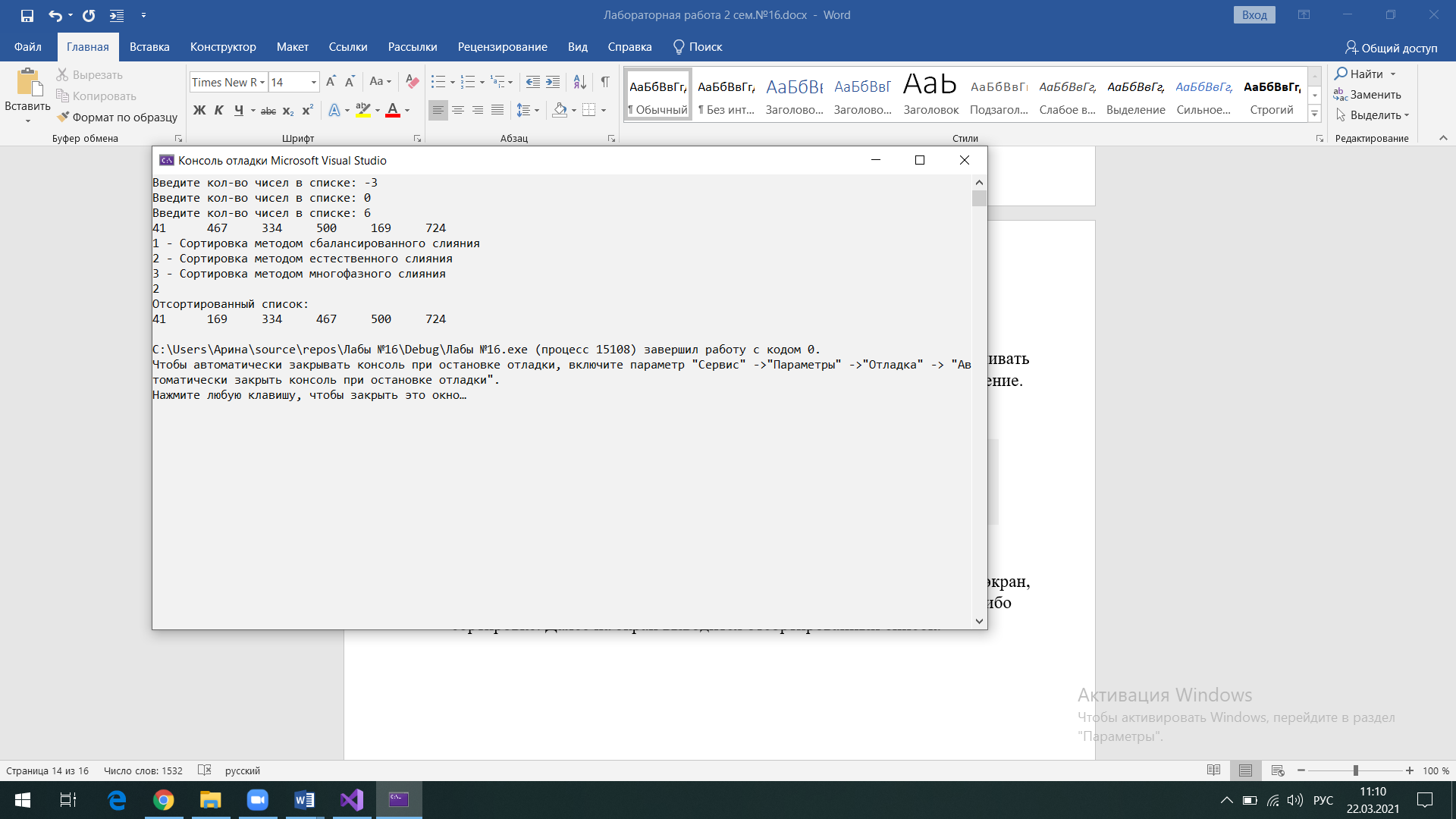
}

4. Выполнение программы.

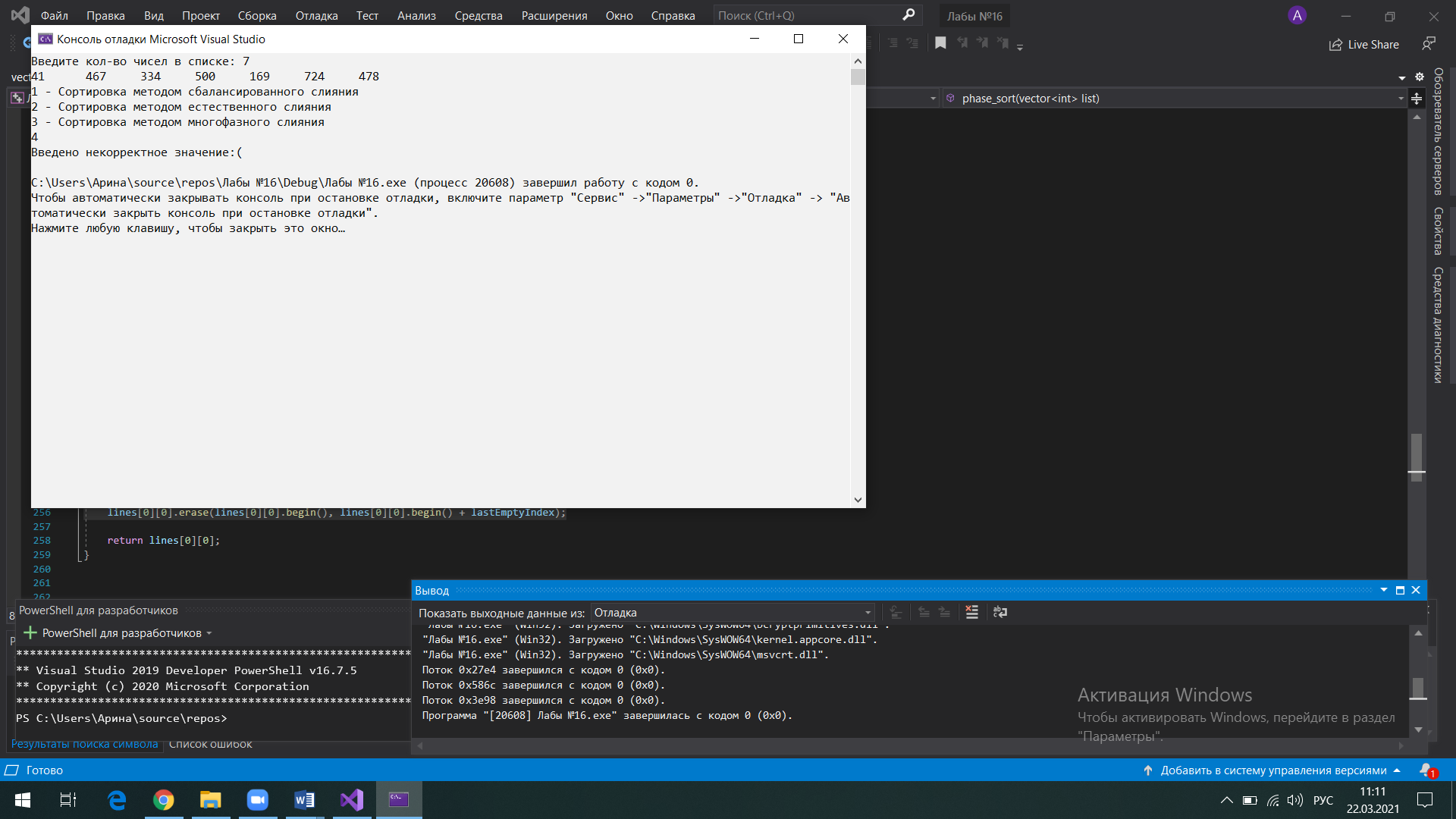
Изначально программа запрашивает ввод количества чисел, если пользователь введёт отрицательное значение, программа будет запрашивать ввод снова и снова, до тех пор, пока не будет введено корректное значение.



После того, как будет сформирован список чисел, и они выведутся на экран, пользователю необходимо ввести команду, соответствующую какой-либо сортировке. Далее на экран выводится отсортированный список.



Если пользователь введёт несуществующую команду, то программа выведет сообщение об этом.



5. Блок схема

