«Министерство образования и науки Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования «Алтайский государственный технический университет им. И.И. Ползунова»

Факультет информационных технологий

Кафедра прикладной математики

Отчёт защищён с оценкой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Преподаватель Бубнова Н.Д.

«\_\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_2021 г.

Отчёт

Лабораторной работе №1

**«**Внутренняя сортировка**»**

Студент группы ПИ 92 В.М. Шульпов

Старший преподаватель Бубнова Н.Д.

Барнаул 2021

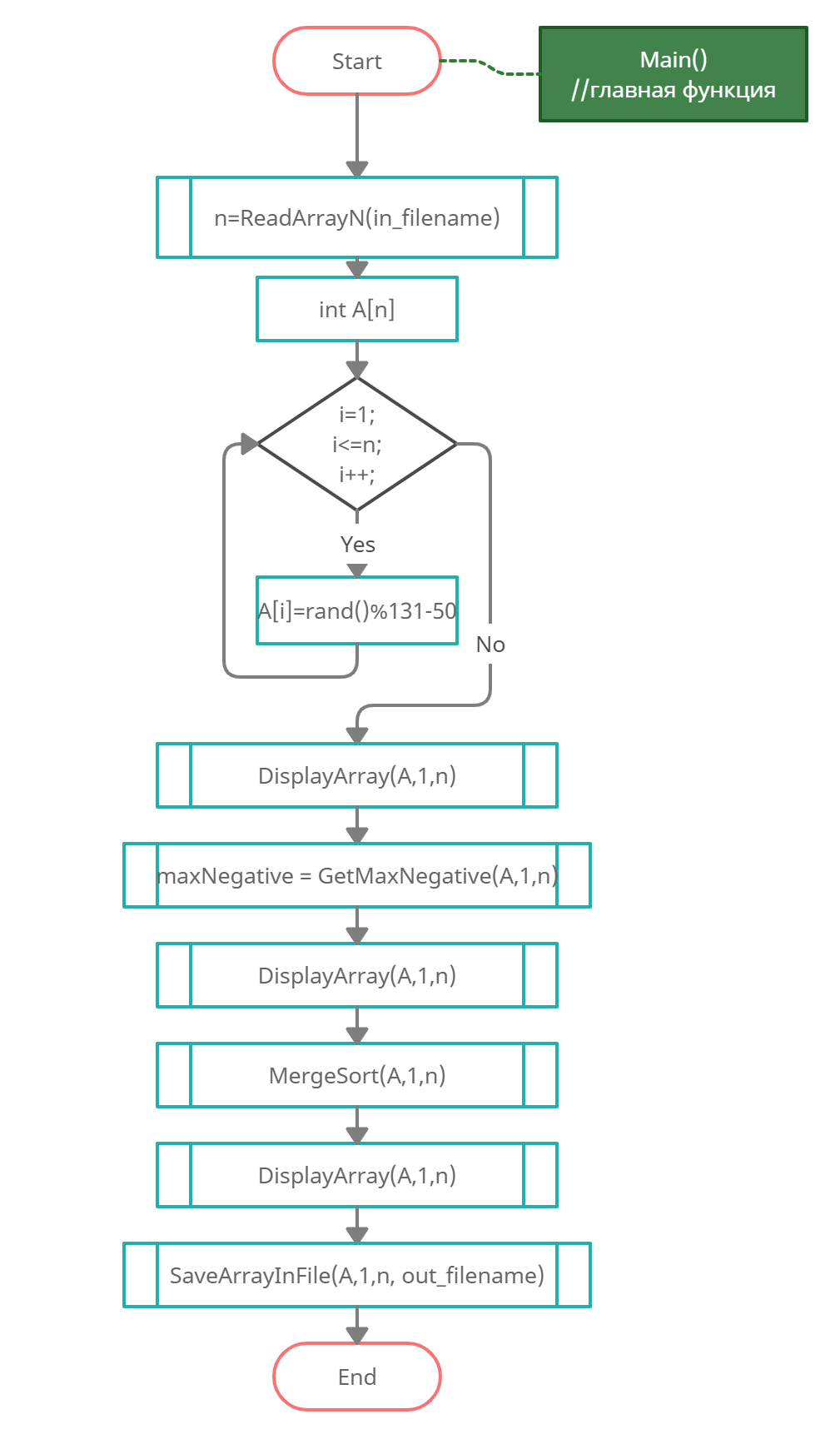
**Задание:**

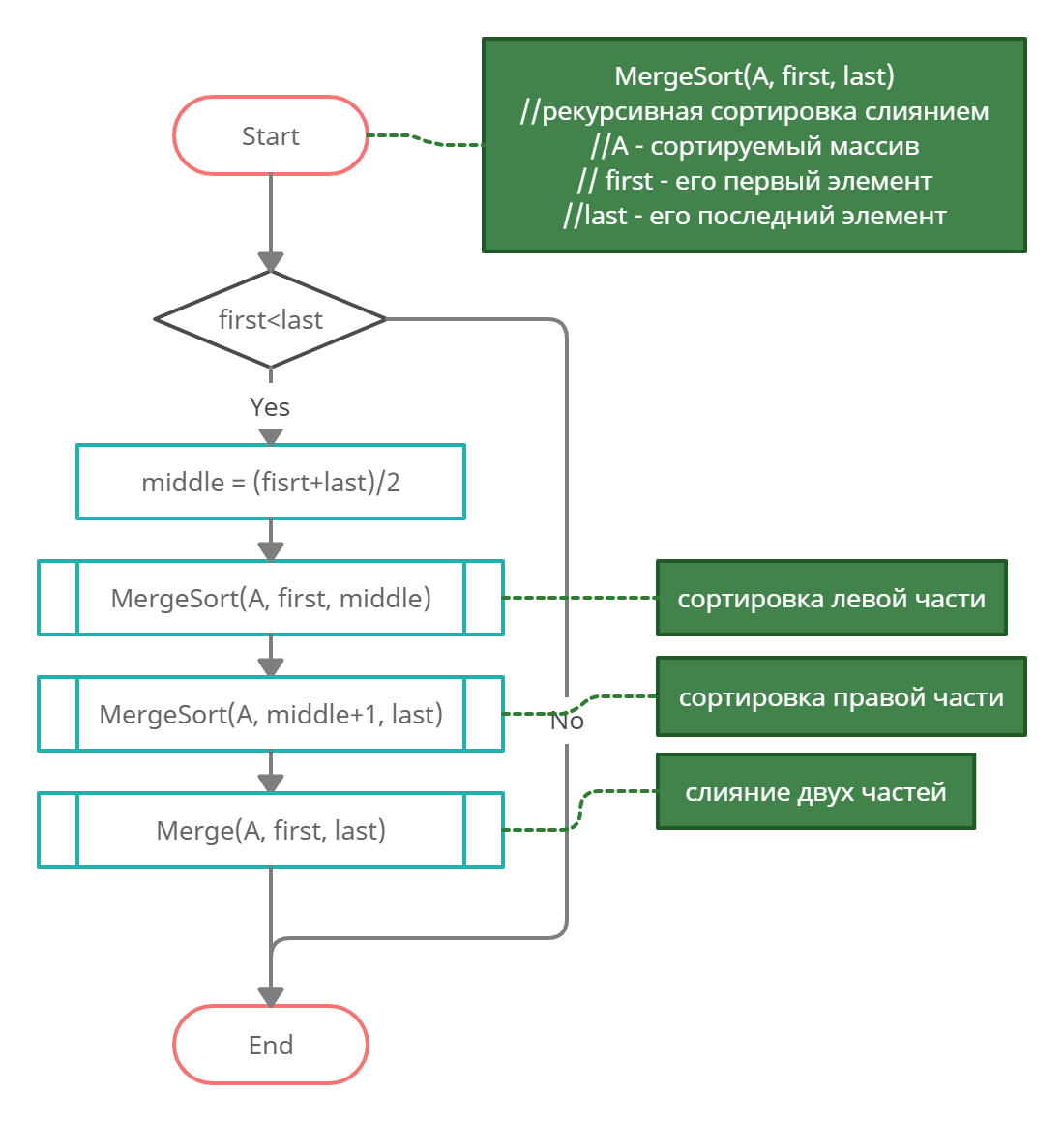
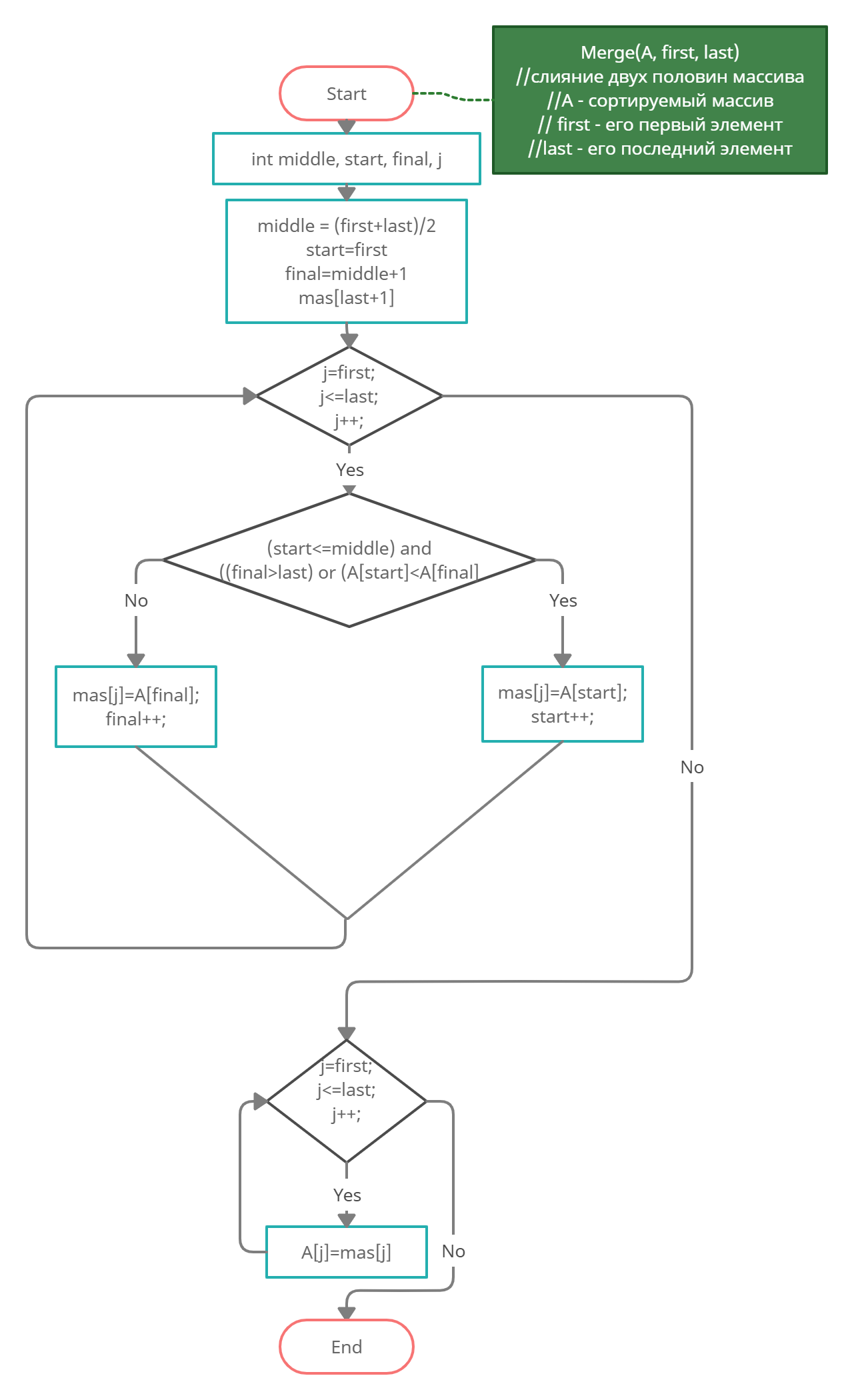
1. Выбрать вариант. Номер варианта должен совпадать с номером в списке группы вличном кабинете2. Ознакомиться с постановкой задачи3. Разработать алгоритм решения задачи.4. Написать программу для ее реализации5. Разработать тесты6. Исходные данные поместить в файл input.dat7. Результаты вывести на экран. Исходные данные и результаты вывести также ввыходной файл output.dat8. Оформить отчет

**Вариант 27**

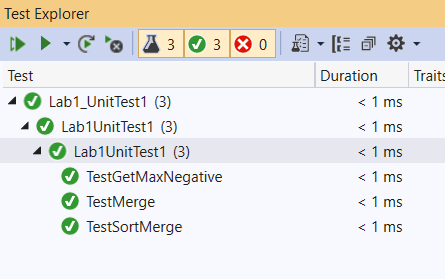
Элементы массива Т сгенерировать датчиком псевдослучайных чисел. Количество элементов массива m≤500, значения элементов принадлежат отрезку [-50, 80]. Выполнить сортировку элементов преобразованного массива по убыванию методом слияния*.* К каждому элементу ti массива T(m) добавить значение, равное максимальному из отрицательных элементов этого массива.

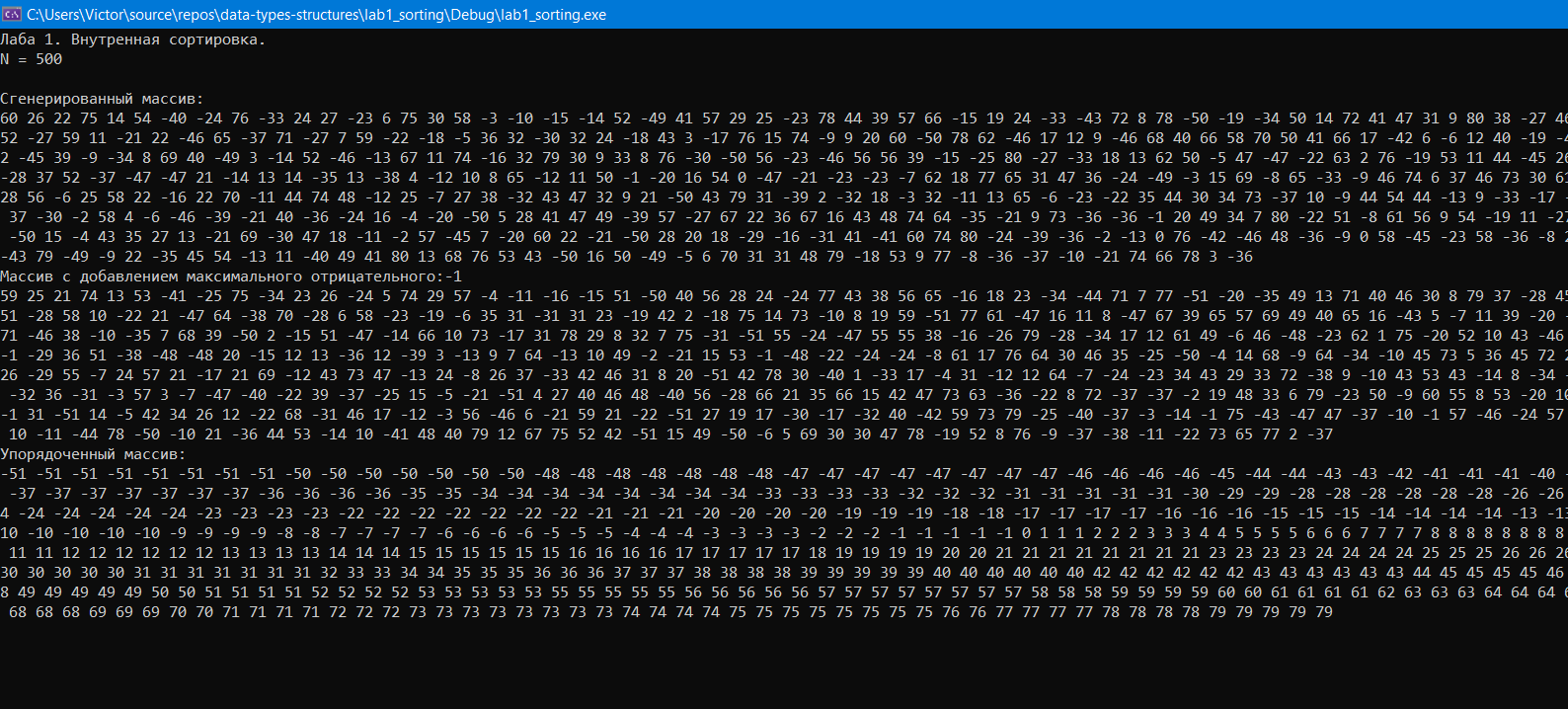
Алгоритм





тесты





Код

**lab1\_sorting.cpp**

#include <fstream>

#include <iostream>

#include <string>

#include <ctime>

#include "lab1\_sorting.h"

using namespace std;

void Merge(int\* A, int first, int last);//функция, сливающая массивы

void MergeSort(int\* A, int first, int last);//рекурсивная процедура сортировки

int GetMaxNegative(int\* A, int first, int last);//получить максимальное отриц число массива

bool SaveArrayInFile(int\* A, int first, int last, const std::string& filename);//сохранения массива в файл

int ReadArrayN(const std::string& filename);//чтения количества элементов массива из файла

void DisplayArray(int\* A, int first, int last);//вывод массива

//главная функция

void main() {

srand(time(NULL));

setlocale(LC\_ALL, "Rus");

cout << "Лаба 1. Внутренная сортировка." << endl;

string in\_filneame = "input.dat";

string out\_filneame = "output.dat";

int i, n;

int maxNegative;

try{

n = ReadArrayN(in\_filneame);

cout << "N = " << n << endl;

}

catch (exception exc){

cout << "\n" << exc.what() << endl;

}

int\* A = new int[n+1];

for (int i = 1; i <= n; i++) {

A[i] = rand() % 131-50; // [-50; 80]

}

cout << "\nСгенерированный массив: " << endl; //вывод упорядоченного массива

DisplayArray(A, 1, n);

try{

maxNegative = GetMaxNegative(A, 1, n);

for (int i = 1; i <= n; i++) {

A[i] += maxNegative;

}

cout << "\nМассив с добавлением максимального отрицательного:" << maxNegative << endl; //вывод массива с добавлением максимального отрицательного

DisplayArray(A, 1, n);

}

catch(exception ex){

cout << "\n" << ex.what() << endl;

}

MergeSort(A, 1, n); //вызов сортирующей процедуры

cout << "\nУпорядоченный массив: " << endl; //вывод упорядоченного массива

DisplayArray(A, 1, n);

SaveArrayInFile(A, 1, n, out\_filneame);

delete[] A;

system("pause>>void");

}

**lab1\_sorting.h**

#pragma once

#include <fstream>

#include <iostream>

#include <string>

#include <ctime>

using namespace std;

//функция, сливающая массивы

void Merge(int\* A, int first, int last) {

int middle, start, final, j;

int\* mas = new int[last + 1];

middle = (first + last) / 2; //вычисление среднего элемента (округляется в меньшую сторону)

start = first; //начало левой части

final = middle + 1; //начало правой части

for (j = first; j <= last; j++) //выполнять от начала до конца

if ((start <= middle) && ((final > last) || (A[start] < A[final]))) {

mas[j] = A[start];

start++;

}

else {

mas[j] = A[final];

final++;

}

//возвращение результата в список

for (j = first; j <= last; j++)

A[j] = mas[j];

delete[] mas;

};

//рекурсивная процедура сортировки

void MergeSort(int\* A, int first, int last) {

if (first < last) {

int middle = (first + last) / 2;

MergeSort(A, first, middle); //сортировка левой части

MergeSort(A, middle + 1, last); //сортировка правой части

Merge(A, first, last); //слияние двух частей

}

}

//получить максимальное отриц число массива

int GetMaxNegative(int\* A, int first, int last) {

int maxNegative=1;

int flag = 0;

for (int i = 1; i <= last; i++) {

if (A[i] < 0 && (flag == 0 || maxNegative < A[i])) {

maxNegative = A[i];

flag = 1;

}

}

if (flag == 0) throw exception("Exception: no negative numbers");

else return maxNegative;

}

//сохранения массива в файл

bool SaveArrayInFile(int\* A, int first, int last, const std::string& filename)

{

/\*

Первое число в файле будет N, остальные - элементы массива

\*/

ofstream out;

// Open the file.

out.open(filename, std::ios::out | std::ios::binary);

// Validate that the file is open.

if (!out.is\_open())

return false;

out << last << "\n\n";

for (int i = first; i <= last; i++) {

out << A[i] << endl;

}

out.close();

return true;

}

//чтения количества элементов массива из файла

int ReadArrayN(const std::string& filename) {

ifstream in;

string str\_n;

int n = 0;

// Open the file.

in.open(filename, std::ios::in | std::ios::binary);

if (in.is\_open()) {

getline(in, str\_n);

}

else

{

throw exception("Exception: file did't open");

}

try {

n = stoi(str\_n);

}

catch (exception exc) {

throw exception("Exception: convert failure");

}

return n;

}

//вывод массива

void DisplayArray(int\* A, int first, int last) {

for (int i = first; i <= last; i++) {

cout << A[i] << " ";

}

}**lab1\_UnitTest1**

#include "pch.h"

#include "CppUnitTest.h"

#include "..\lab1\_sorting\lab1\_sorting.h"

using namespace Microsoft::VisualStudio::CppUnitTestFramework;

namespace Lab1UnitTest1

{

TEST\_CLASS(Lab1UnitTest1)

{

public:

TEST\_METHOD(TestMerge)

{

//тест 1 (8 элементов, массивы не были отсортированы)

{

int inA[] = { 0, 8, 9, 6, 7, 8, 12, 3, 4 };

Merge(inA, 1, 8);

int outA[] = { 0,8,8,9,6,7,12,3,4 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 2 (6 элементов, массивы отсортированы)

{

int inA[] = { 0, 22, 33, 44, 2, 3, 4 };

Merge(inA, 1, 6);

int outA[] = { 0,2,3,4,22,33,44 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 3 (5 элементов, массивы отсортированы)

{

int inA[] = { 0, 22, 33, 44, 2, 3 };

Merge(inA, 1, 5);

int outA[] = { 0,2,3,22,33,44 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 4 (5 элементов, массивы не были отсортированы)

{

int inA[] = { 0, 5, 4, 3, 2, 1 };

Merge(inA, 1, 5);

int outA[] = { 0,3,2,1,5,4,3 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 5 (1 элемент)

{

int inA[] = { 0, 55 };

Merge(inA, 1, 1);

int outA[] = { 0,55 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 6 (2 элемента)

{

int inA[] = { 0, 55, 66 };

Merge(inA, 1, 2);

int outA[] = { 0,55,66};

Assert::IsTrue(compareArray(inA, outA));

}

}

TEST\_METHOD(TestSortMerge)

{

//тест 1 (сортировка неотсортированного 8 элем)

{

int inA[] = { 0, 8,7,6,5,4,3,2,1 };

Merge(inA, 1, 8);

int outA[] = { 0,1,2,3,4,5,6,7,8 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 2 (сортировка отсортированного 8 элем)

{

int inA[] = { 0,1,2,3,4,5,6,7,8 };

Merge(inA, 1, 8);

int outA[] = { 0,1,2,3,4,5,6,7,8 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 3 (сортировка отсортированного 3 элем)

{

int inA[] = { 0,1,2,3};

Merge(inA, 1, 3);

int outA[] = { 0,1,2,3};

Assert::IsTrue(compareArray(inA, outA));

}

//тест 4 (сортировка неотсортированного 3 элем)

{

int inA[] = { 0,3,2,1 };

Merge(inA, 1, 3);

int outA[] = { 0,1,2,3 };

Assert::IsTrue(compareArray(inA, outA));

}

//тест 5 (сортировка массива одинаковых элементов)

{

int inA[] = { 0,1,1,2,1,2};

Merge(inA, 1, 5);

int outA[] = { 0,1,1,1,2,2 };

Assert::IsTrue(compareArray(inA, outA));

}

}

TEST\_METHOD(TestGetMaxNegative)

{

//тест 1

{

int inA[] = { 0,1,1,2,3,-1 };

Assert::IsTrue(GetMaxNegative(inA, 1, 5) == -1);

}

//тест 2 (на отриц и полож)

{

int inA[] = { 0,-1,1,-0,-1,0 };

Assert::IsTrue(GetMaxNegative(inA, 1, 5) == -1);

}

//тест 3

{

int inA[] = { 0,-1,-1,-2,-3,-99 };

Assert::IsTrue(GetMaxNegative(inA, 1, 5) == -1);

}

}

void throwException()

{

throw 1;

}

bool compareArray(int\* mas\_a, int\* mas\_b) {

//метод сравнения массивов на идентичность

bool flag;

for (int i = 0; i < sizeof(mas\_a) / sizeof(mas\_a[0]) - 1; i++) {

flag = false;

for (int j = 0; j < sizeof(mas\_b) / sizeof(mas\_b[0]) - 1; j++) {

if (mas\_a[i] == mas\_b[j]) { flag = true; break; }

if (flag == false) return false;

}

}

return true;

}

};

}