1. Лабораторная работа № 4

«ПРОГРАММИРОВАНИЕ АЛГОРИТМОВ ОБРАБОТКИ ОДНОМЕРНЫХ МАССИВОВ»

* 1. Цель работы

Изучить способы представления массивов в памяти ЭВМ, получить практические навыки реализации алгоритмов обработки одномерных массивов, исследовать свойства алгоритма сортировки.

* 1. Постановка задачи

Получить навыки разработки и реализации алгоритмов обработки одномерных массивов.

* 1. Ход выполнения работы
     1. Описание алгоритма решения задачи

Вариант 29. Разделить поставленную задачу на три подзадачи и решить каждую из них независимо от других подзадач.

* + - 1. Поиск максимального по модулю элемента

Для выполнения достаточно перебрать весь массив и найти число, модуль которого есть наибольшее число.

* + - 1. Вычислить сумму элементов массива, расположенных после второго положительного элемента

Организация поиска второго положительного элемента в массиве. Возможны случаи:

1. В массиве отсутствуют положительные элементы или найден только один такой элемент. Сумму посчитать невозможно. Вывод соответствующего сообщения.
2. В массиве найден второй положительный элемент, но после него отсутствуют другие элементы. Сумму посчитать невозможно. Вывод соответствующего сообщения.
3. В массиве найден второй положительный элемент и после него существуют другие элементы. Сумму посчитать возможно. Вывод суммы.
   * + 1. Преобразовать массив.

Предложено преобразовать массив таким образом, чтобы элементы, больше 1, располагались после всех остальных. Для этого массив сортируется по возрастанию, например, прямым выбором. Затем запоминается индекс первого элемента, большего 1. Условно массив делится на левую часть – элементы, строго меньшие 1 –, и правую часть – элементы, большие или равные 1.

Предложено отсортировать каждую часть по убыванию методом прямого выбора. Для выполнения сортируется сначала левая часть, а затем правая.

* + 1. Структурная схема алгоритма

Структурная схема алгоритма представлена на Рисунке 4.1.

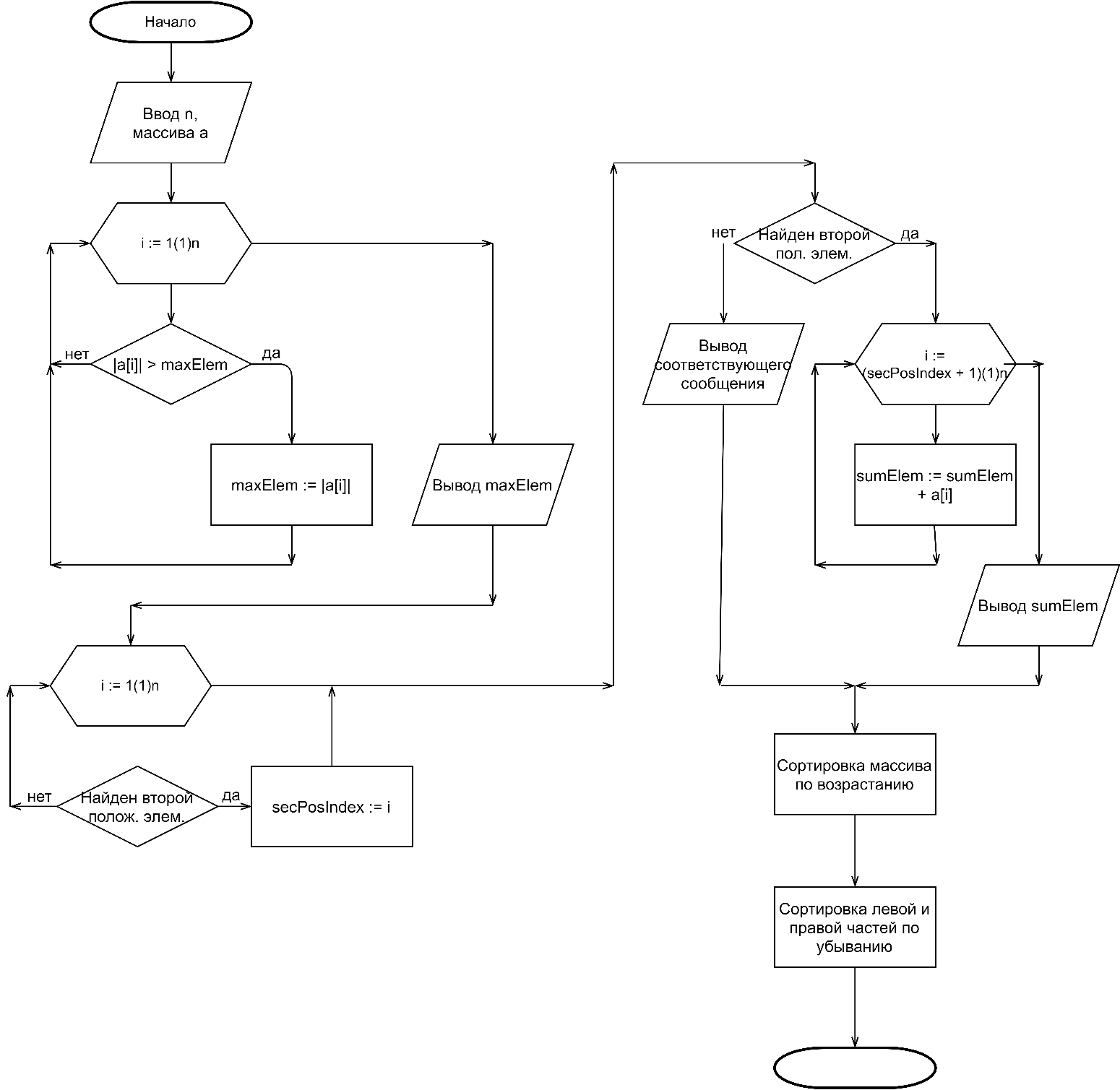


Рисунок 4.1 – Структурная схема алгоритма

* + 1. Текст программы

Program LW\_4\_29;

Const arrayLen = 50;

Var

a: array [1..arrayLen] of double; maxElem, sumElem, tmpVal: double;

countTrans, countComp, n, tmpI, i, j, index1, secPosIndex: integer;

firPosIsFind: boolean;

BEGIN

firPosIsFind := false; secPosIndex := 0; sumElem := 0; countComp := 0;

countTrans := 0;

{Ввод данных}

write('Введите количество элементов (не более ', arrayLen, '): '); readln(n);

if n > arrayLen then begin writeln(‘Ошибка’); Exit end;

for i := 1 to n do

begin

write('Введите ', i, ' элемент: '); readln(a[i])

end;

{Поиск максимального по модулю элемента}

maxElem := abs(a[1]);

for i := 2 to n do if abs(a[i]) > maxElem then maxElem := abs(a[i]);

{Вывод Результата №1}

writeln('Результат №1: ', maxElem, '.');

{Поиск второго положительного элемента}

for i := 1 to n do

begin

if a[i] > 0 then

if firPosIsFind then

begin

secPosIndex := i;

break

end

else

firPosIsFind := true;

end;

{Суммирование элементов после второго }

{положительного (если такой был найден)}

write('Результат №2: ');

if secPosIndex <> 0 then

if (secPosIndex + 1) < n then

begin

for i := (secPosIndex + 1) to n do sumElem := sumElem + a[i];

writeln(sumElem);

end

else

writeln('Нет элементов после второго положительного.')

else

writeln('В массиве отсутствует второй положительный элемент.');

{Сортировка массива по возрастанию }

for i := 1 to (n - 1) do

begin

tmpI := i; tmpVal := a[i];

for j := i + 1 to n do

if tmpVal > a[j] then begin tmpI := j; tmpVal := a[j]; end;

a[tmpI] := a[i]; a[i] := tmpVal;

end;

{Поиск индекса элемента, большего 1}

for i := 1 to n do

if a[i] > 1 then begin index1 := i; break end;

{Отладочная печать}

writeln();

writeln('-----------Отладочная печать----------------');

writeln('Результат промежуточной сортировки массива: ', a);

if index1 < (n + 1) then

writeln('Номер элемента, большего 1: ', index1)

else

writeln('В массиве нет элемента, большего 1.');

writeln();

{Сортировка левой части массива}

for i := 1 to (index1 - 1) do

begin

tmpVal := a[i]; tmpI := i;

for j := i + 1 to (index1 - 1) do

begin

countComp := countComp + 1;

if tmpVal < a[j] then tmpI := j;

end;

tmpVal := a[tmpI]; a[tmpI] := a[i]; a[i] := tmpVal; countTrans := countTrans + 1;

end;

{Сортировка правой части массива}

for i := index1 to n do

begin

tmpVal := a[i]; tmpI := i;

for j := i + 1 to n do

begin

countComp := countComp + 1;

if tmpVal < a[j] then tmpI := j;

end;

tmpVal := a[tmpI]; a[tmpI] := a[i]; a[i] := tmpVal; countTrans := countTrans + 1;

end;

{Печать результата №3 и отладочная печать. Завершение работы программы.}

writeln('Преобразованный массив', a);

writeln('Количество пересылок: ', countTrans);

writeln('Количество сравнений: ', countComp);

writeln(); writeln('Конец работы.'); read();

END.

* + 1. Описание тестовых примеров. Исследование зависимости количества операций пересылок и сравнения данных в ходе сортировки от размера массива

Тестовые примеры представлены на Рисунках 4.2–4.4.

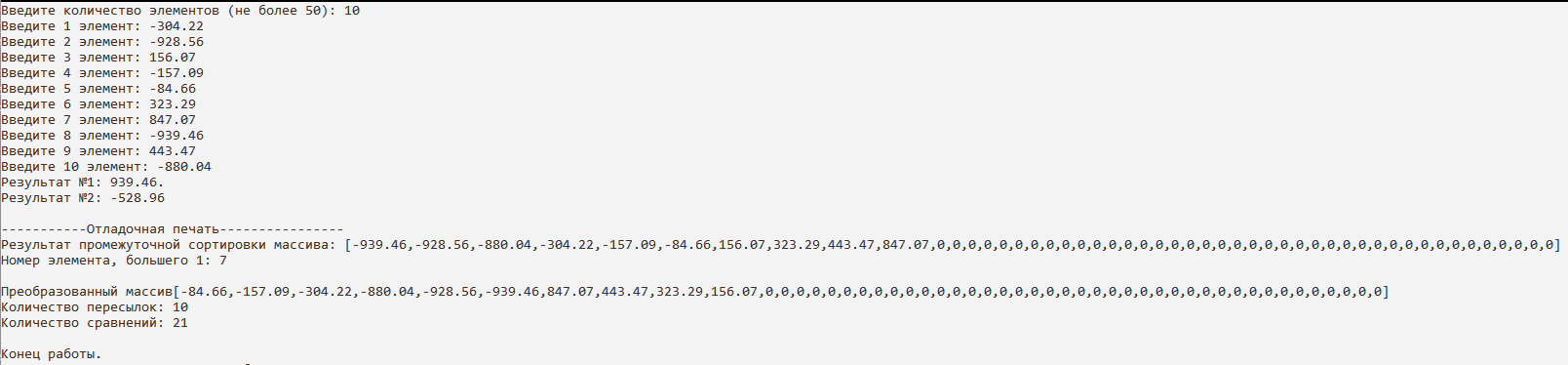


Рисунок 4.2 – Пример с 10 элементами в массиве

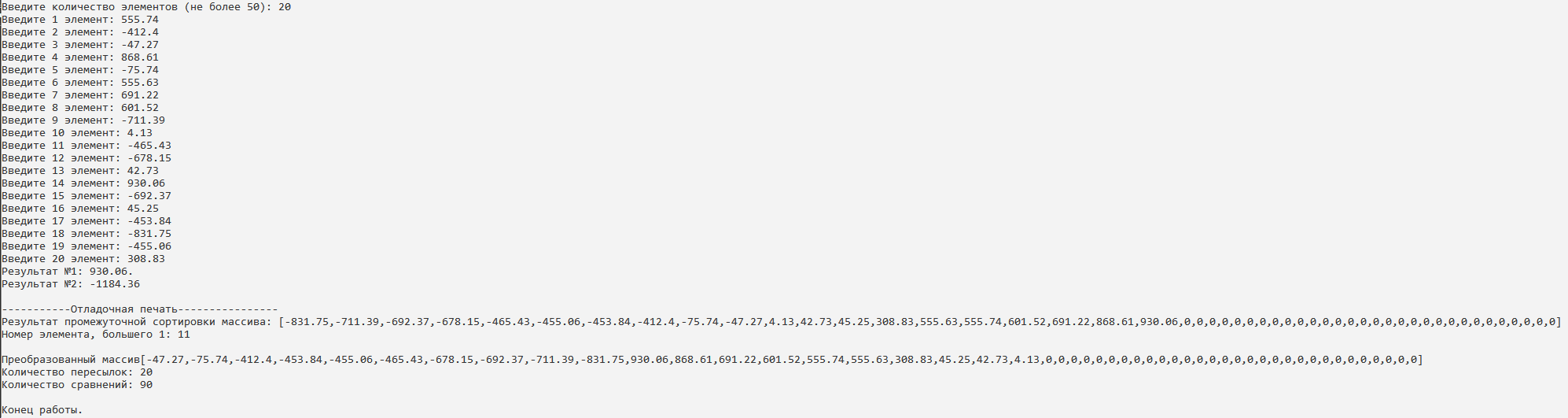


Рисунок 4.3 – Пример с 20 элементами в массиве

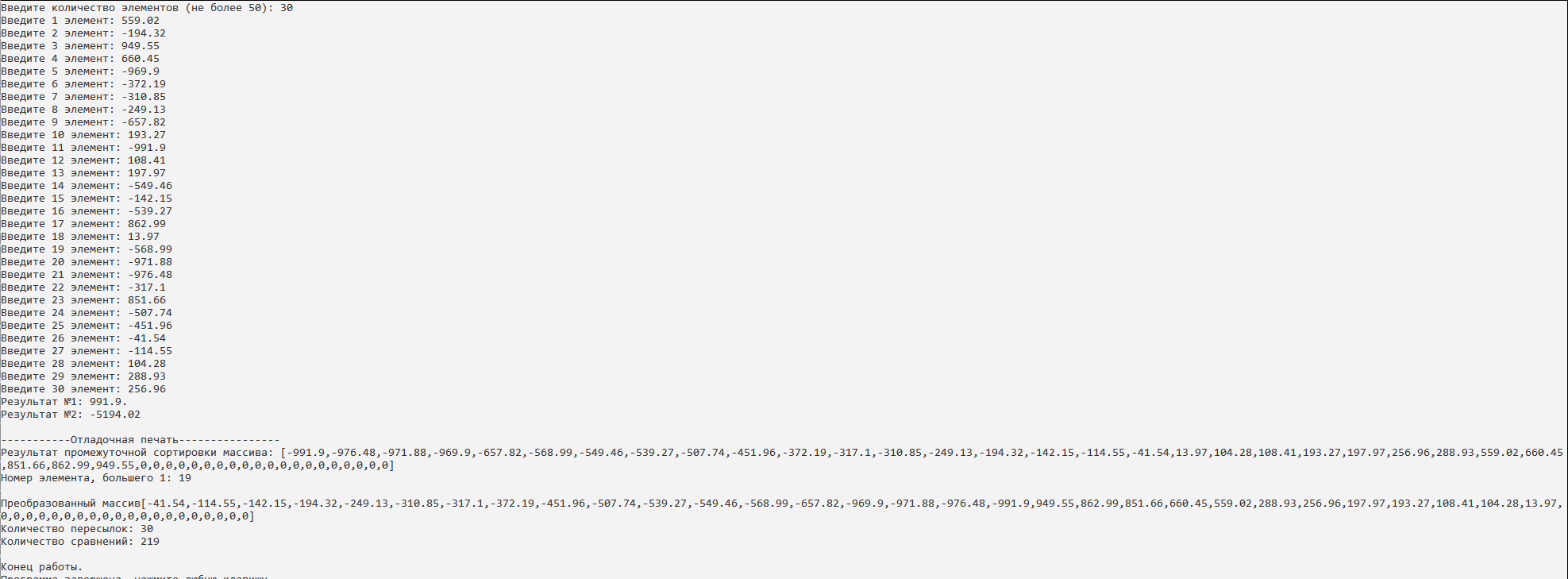


Рисунок 4.4 – Пример с 30 элементами в массиве

В каждом тесте изменяется число элементов в массиве, значения элементов подбираются случайным образом. Примеры показывают, что с увеличением количества элементов в массиве увеличивается количество сравнений и пересылок в программе. Проведено дополнительное тестирование программы, не отображённое в отчёте, и результаты представлены в виде графика (Рисунок 4.5).

Рисунок 4.5 – График зависимости

График показывает, что число пересылок равно количеству элементов в массиве, а количество сравнений увеличивается с увеличением количества элементов. Возможные причины отображены в выводе.

Вывод

В ходе выполнения данной лабораторной работы были получены навыки программирования алгоритмов обработки одномерных массивов, а именно алгоритмов сортировки элементов в массиве. В ходе тестирования программы выявлено, что количество сравнений и пересылок увеличивается с ростом количества элементов в массиве. Зависимость «количество пересылок–количество элементов» объясняется тем, что, по окончании сравнения текущего элемента с другими, происходит пересылка значений. Зависимость «количество сравнений–количество элементов» объясняется тем, что с увеличением количества элементов увеличивается как число элементов, которых надо сравнить, так и число элементов, с которыми можно сравнить. Полученные во время разработки навыки помогут разрабатывать более эффективные алгоритмы сортировки данных.