Министерство образования Республики Беларусь

Учреждение образования

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей

Кафедра программного обеспечения информационных технологий

Дисциплина: Основы алгоритмизации и программирования (ОАиП)

ОТЧЕТ

по лабораторной работе №2

Тема работы: Исследование алгоритмов сортировки массивов

Выполнил

студент: гр. 151004 Данилов Ф.А.

Проверил: Фадеева Е.П.

Минск 2022

Содержание

[1 Постановка задачи 4](#_Toc103186148)

[2 Методика решения задачи 5](#_Toc103186149)

[2.1 Быстрая сортировка 5](#_Toc103186150)

[2.2 Сортировка методом простых вставок 5](#_Toc103186151)

[2.3 Результаты сравнений методов 6](#_Toc103186152)

[3 Описание алгоритмов решения задачи 7](#_Toc103186153)

[4 Структура данных 9](#_Toc103186154)

[4.1 Структура данных программы 9](#_Toc103186155)

[4.2 Структура данных алгоритма RandomArray 10](#_Toc103186156)

[4.3 Структура данных алгоритма SortedArray 10](#_Toc103186157)

[4.4 Структура данных алгоритма InvertedArray 11](#_Toc103186158)

[4.5 Структура данных алгоритма QuickSortTest 11](#_Toc103186159)

[4.6 Структура данных алгоритма InsertionSortTest 12](#_Toc103186160)

[5 Схема алгоритма решения задачи 13](#_Toc103186161)

[5.1 Схема алгоритма RandomArray 14](#_Toc103186162)

[5.2 Схема алгоритма SortedArray 15](#_Toc103186163)

[5.3 Схема алгоритма InvertedSort 16](#_Toc103186164)

[5.4 Схема алгоритма QuickSortTest 17](#_Toc103186165)

[5.5 Схема алгоритма InsertionSortTest 19](#_Toc103186166)

[6 Результаты расчетов программы 20](#_Toc103186167)

[Приложение А 21](#_Toc103186168)

[Приложение Б 27](#_Toc103186169)

# Постановка задачи

Необходимо провести сравнительный анализ следующих сортировок:

* метод простых вставок;
* быстрая сортировка.

Анализ алгоритмов сортировок провести по критерию количества перестановок.

Размерности массивов соответственно: 100, 250, 500, 1000, 2000, 3000.

Типы массивов:

* случайный;
* сортированный;
* перевернутый.

Результаты расчетов свести в таблицу:

Таблица 1 – Оформление результатов расчетов

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность массива | Тип  Массива | 1-ый метод сортировки | | 2-ой метод сортировки | |
| Кол-во экспериментальное | Кол-во теоретическое | Кол-во экспериментальное | Кол-во теоретическое |
| N = 100 |  |  |  |  |  |
| … |  |  |  |  |  |
| N = 3000 |  |  |  |  |  |

# Методика решения задачи

## Быстрая сортировка

QuickSort является существенно улучшенным вариантом алгоритма сортировки с помощью прямого обмена, известного в том числе своей низкой эффективностью. Принципиальное отличие состоит в том, что в первую очередь производятся перестановки на наибольшем возможном расстоянии и после каждого прохода элементы делятся на две независимые группы.

Общая идея алгоритма состоит в следующем:

Выбрать из массива элемент, называемый опорным. Это может быть любой из элементов массива. От выбора опорного элемента не зависит корректность алгоритма, но в отдельных случаях может сильно зависеть его эффективность.

Сравнить все остальные элементы с опорным и переставить их в массиве так, чтобы разбить массив на три непрерывных отрезка, следующих друг за другом: “элементы меньшие опорного”, “равные” и “большие”.

Для отрезков “меньших” и “больших” значений выполнить рекурсивно ту же последовательность операций, если длина отрезка больше единицы.

На практике массив обычно делят не на три, а на две части: например, “меньшие опорного” и “равные и большие”. Такой подход в общем случае эффективнее, так как упрощает алгоритм разделения.

Для случая отсортированного массива число переприсваиваний вычисляется по следующей формуле:

*.*

Для случая массива, заполненного случайным образом число переприсваиваний вычисляется по следующей формуле:

.

Для наихудшего случая, когда опорные элементы являются максимальными или минимальными элементами подмассивов, число переприсваиваний вычисляется по следующей формуле:

.

В перечисленных формулах M – число перестановок, n – размер исходного массива.

## Сортировка методом простых вставок

Для сортировки методом простых вставок число переприсваиваний и, как следствие, выбор формулы зависит от типа массива:

1. Если массив изначально отсортирован, число переприсваиваний вычисляется по следующей формуле:

M – число перестановок,

n – размер исходного массива.

1. Если массив заполнен случайными числами, число переприсваиваний вычисляется по следующей формуле:

M – число перестановок,

n – размер исходного массива.

1. Если массив отсортирован в обратном порядке, число пере присваиваний будет вычисляться по следующей формуле:

M – число перестановок

n – размер исходного массива

## Результаты сравнений методов

Для хранения результатов вычислений используется массив значений, которые выводятся на экран при каждой длине исходных массивов.

# Описание алгоритмов решения задачи

Таблица 2 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № п.п. | Наименование алгоритма | Назначение  алгоритма | Формальные параметры | Предполагаемый тип реализации |
| 1 | Основной алгоритм | Сравнение двух алгоритмов сортировки. Вызывает подпрограммы  Create(Id, Length, A)  QuickSort(A, Count)  InsertionSort(A, Count) |  |  |
| 2 | Create  (Id, Length, A) | Заполнение массива A длиной Length значениями в зависимости от параметра Id.  Id = 1 – Случайные значения;  Id = 2 – Перевернутый массив;  Id = 3 – Отсортированный массив | A – получает от фактического параметра адрес, возвращаемый параметр;  Id – получает от фактического параметра значение;  Length – получает от фактического параметра значение | Функция. A – возвращаемый функцией параметр |
| 3 | QuickSort  (A, Count) | Сортировка массива по методу быстрой сортировки и подсчет количества переприсваиваний.  A – массив;  Count – количество перестановок | A – получает от фактического параметра адрес;  Count – получают от фактического параметра адрес, возвращаемый параметр | Функция.  Count – возвращаемый функцией параметр |

Таблица 3 – Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | InsertionSort (A, Count) | Сортировка массива по методу простых вставок и подсчет количества переприсваиваний.  A – массив;  Count – количество перестановок | A – получает от фактического параметра адрес;  Count – получают от фактического параметра адрес, возвращаемый параметр | Функция.  Count – возвращаемый функцией параметр |

# Структура данных

## Структура данных программы

Таблица 4 – Структура данных основной программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| ArrLengths | Array [1..6] of Integer | Массив, который хранит размерности массивов применяемых для сортировки, соответственно: 100, 250, 500, 1000, 2000, 3000 |
| I | Integer | Номер текущей сортировки |
| ArrRandom | Array of Integer | Массив, значения которого случайно сгенерированы |
| ArrReversed | Array of Integer | Массив, значения которого отсортированы по убыванию |
| ArrSorted | Array of Integer | Массив, значения которого отсортированы по возрастанию |
| Count | Array[1..6] of Integer | Массив, который хранит значения перестановок сортировок |

## Структура данных алгоритма Create

Таблица 5 – Структура данных алгоритма Crearte(Id, Length, A)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Id | Integer | Значение, по которому создается массив.  Id = 1 – Случайный массив;  Id = 2 – Перевернутый массив;  Id = 3 – Отсортированный массив | Формальный |
| Length | Integer | Длина массива | Формальный |
| A | Array [1..Length] of Integer | Массив целочисленных значений, на основе которого проводится сравнение методов сортировок | Формальный |
| I | Integer | Индекс массива, итератор цикла | Локальный |

## Структура данных алгоритма QuickSort

Таблица 6 – Структура данных алгоритма QuickSort(A, Count)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| A | Array [1..3000] of Integer | Массив целочисленных значений, на основе которого проводится сравнение методов сортировок | Формальный |

Таблица 7 – Продолжение таблицы 6

|  |  |  |  |
| --- | --- | --- | --- |
| Count | Integer | Переменная для хранения количества перестановок | Формальный |
| i, j | Integer | Итератор цикла | Локальный |
| L, R | Integer | Границы сортируемой части массива | Локальный |
| x | Integer | Опорный элемент | Локальный |
| St | Array [1..Length, 1..2] of Integer | Стек “вызовов” сортировки | Локальный |
| s | Integer | Индекс последнего элемента стека Stack | Локальный |

## Структура данных алгоритма InsertionSort

Таблица 8 – Структура данных алгоритма InsertionSort(A, Count)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| A | Array [1..3000] of Integer | Массив целочисленных значений, на основе которого проводится сравнение методов сортировок | Формальный |
| Count | Integer | Переменная для хранения количества перестановок | Формальный |
| i, j | Integer | Итератор цикла | Локальный |
| temp | Integer | Ключевой элемент для сортировки вставками | Локальный |

# Схема алгоритма решения задачи

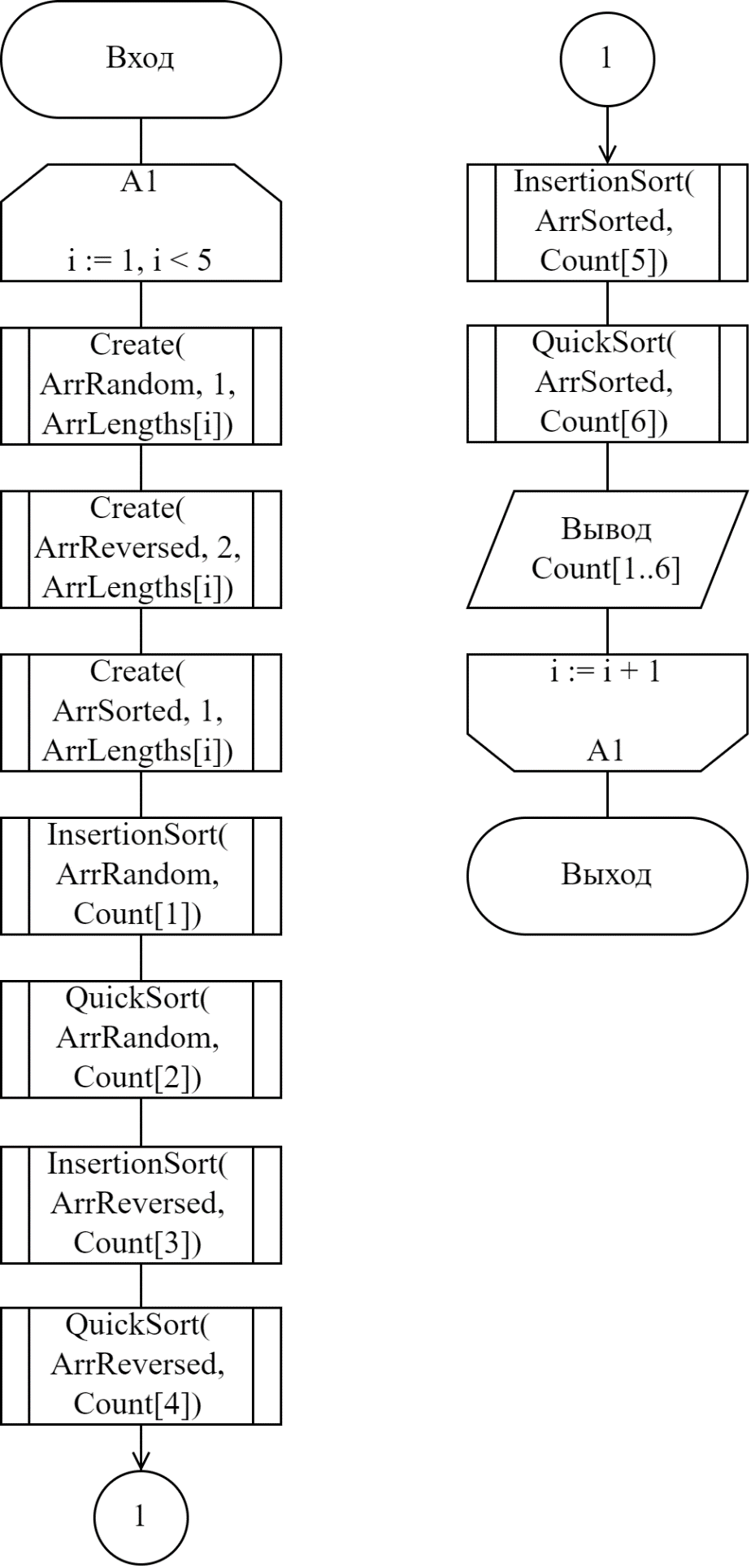


Рисунок 1 – Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма Create

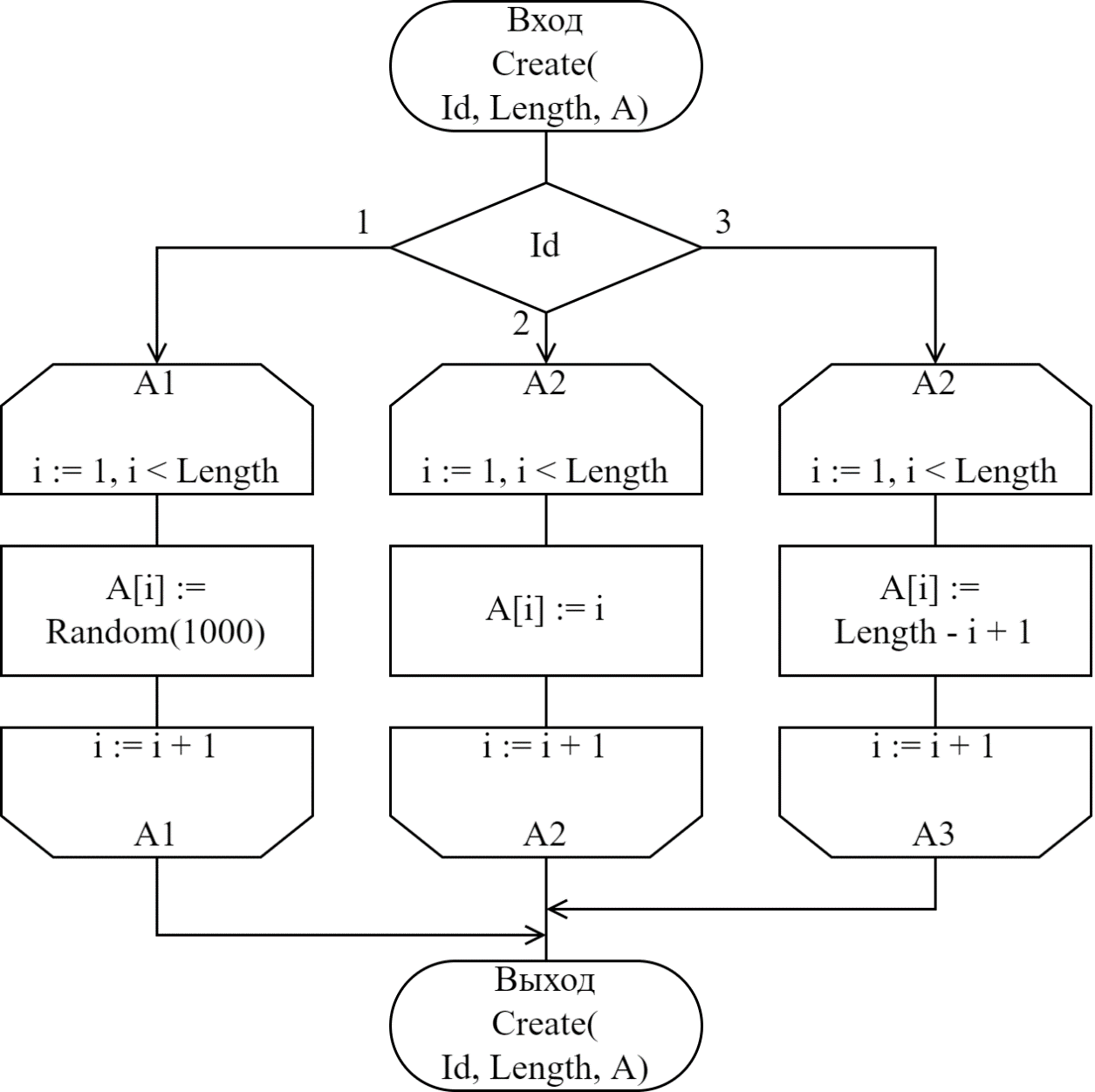


Рисунок 2 – Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма InsertionSort

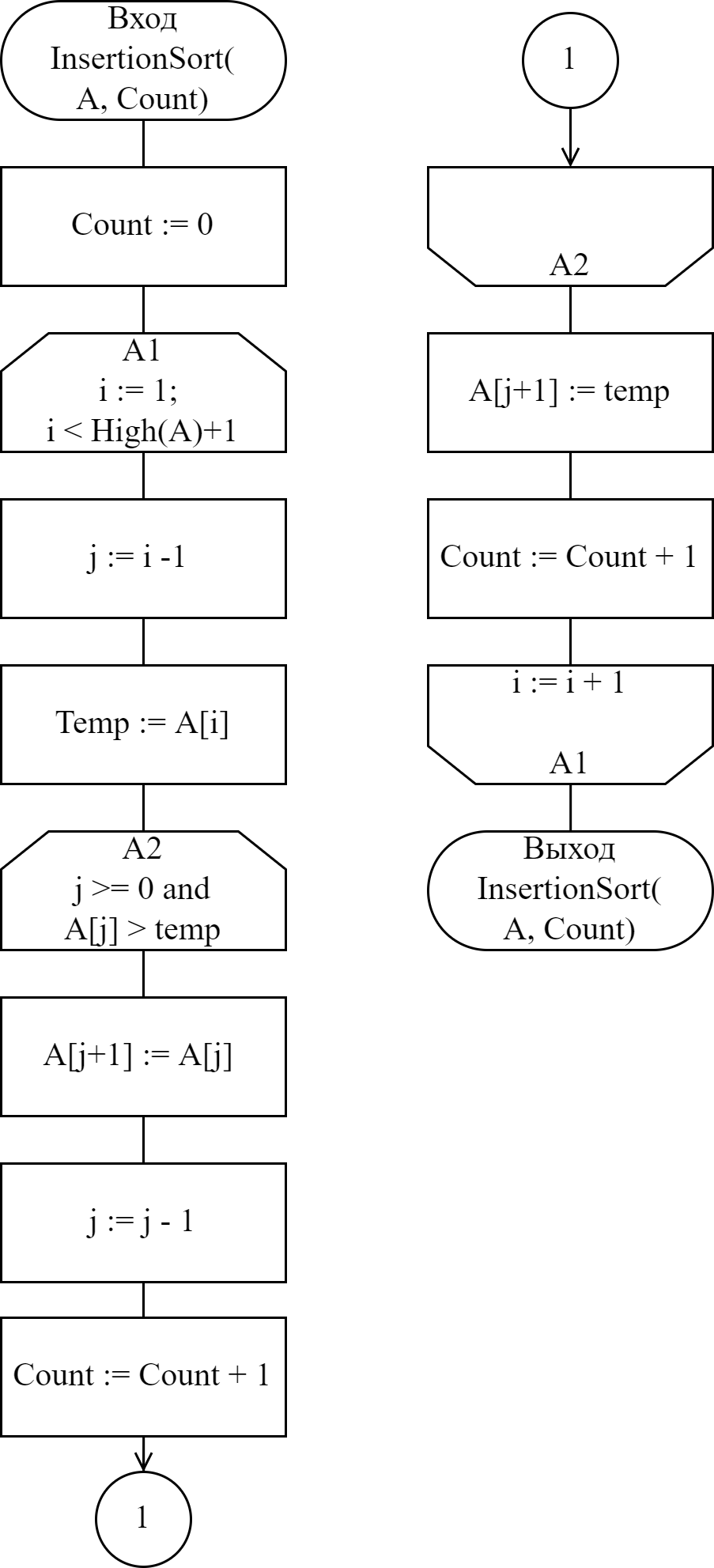


Рисунок 3 – Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма QuickSort

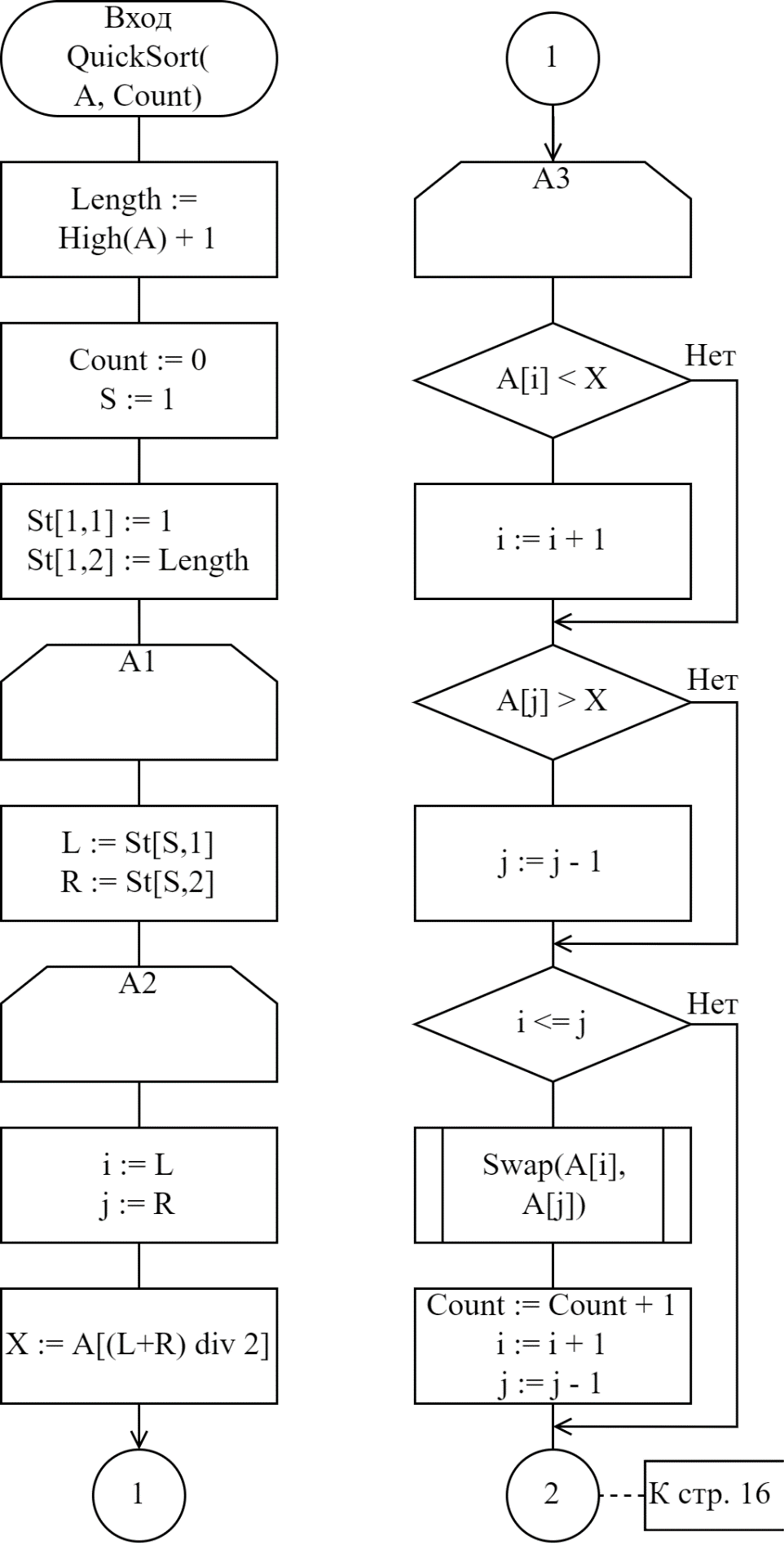


Рисунок 4 – Схема алгоритма решения задачи по ГОСТ 19.701-90

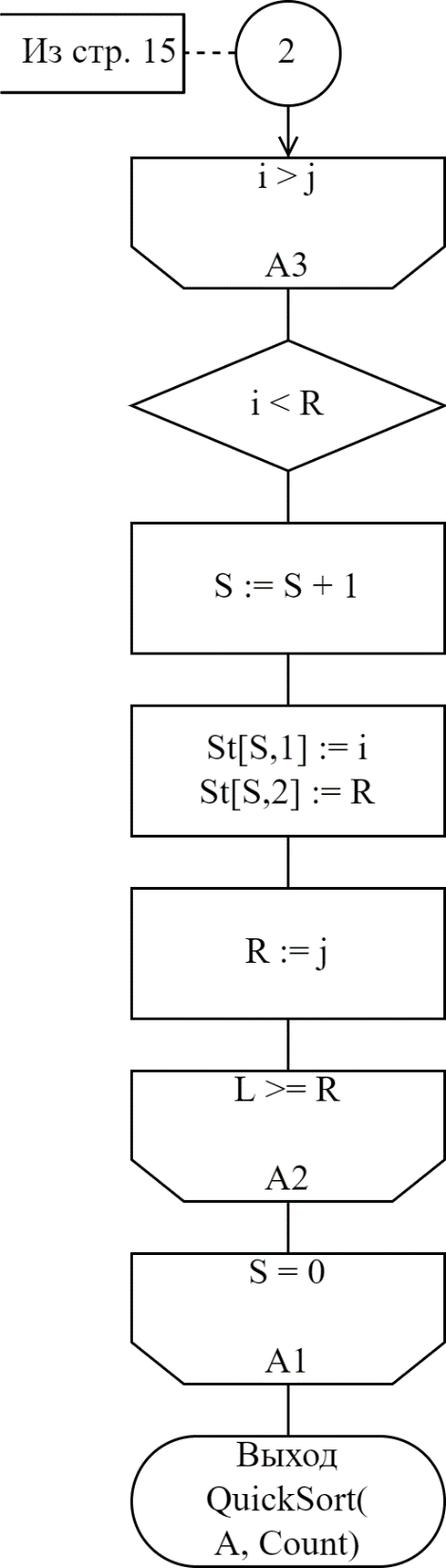


Рисунок 5 – Схема алгоритма решения задачи по ГОСТ 19.701-90

# Результаты расчетов программы

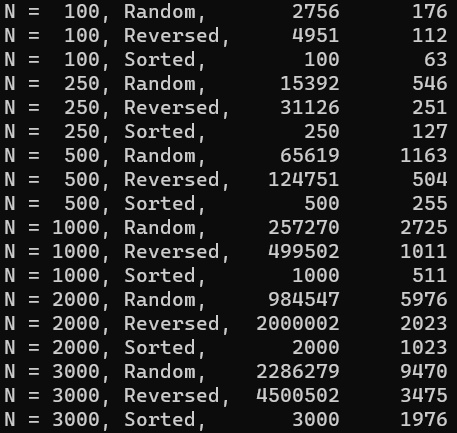


Рисунок 6 – Результаты расчётов программы

Приложение А

(обязательное)

Исходный код программы

program Lab2;

{$APPTYPE CONSOLE}

{$R \*.res}

uses

System.SysUtils;

// TArray - user generated dynamic array of Integer

type

TArray = array of Integer;

// Create Array based on id with set length

// id = 1 -> Create Random array

// id = 2 -> Create Reversed array

// id = 3 -> Create Sorted array

function Create(id, length: Integer): TArray;

// i - index

var

i: Integer;

begin

// Set length of array

SetLength(Result, Length);

Case id of

// Create Random array

1:

begin

// Randomize seed

Randomize;

// Randomize values of array

for i := 1 to Length do

Result[i] := Random(1000);

end;

// Create Reversed array

2:

begin

// Make Arr[i] val equal to Length - index

for i := 1 to Length do

Result[i] := Length - i + 1;

end;

// Create Sorted array

3:

begin

// Make Arr[i] val equal to index

for i := 1 to Length do

Result[i] := i;

end;

// Error

else

begin

writeln('Error!');

Result := nil;

end;

End;

end;

// Insertion sort

// ArrGiven - given array

function InsertionSort(ArrGiven: TArray): Integer;

// Arr - array which is sorted

// i, j - indexes

// temp - temp value to swap

var

Arr: TArray;

i,j,temp: Integer;

begin

// Copy given array

Arr := Copy(ArrGiven);

// Make Result equal to 0

// to increment without errors

Result := 0;

// Sort array from 1 to Length

for i := 1 to High(Arr)+1 do

begin

// Set default j index to i - 1

j := i - 1;

// Save value of Arr[i]

temp := Arr[i];

// While Arr[j] > Arr[i] move it to Arr[j+1]

while (j >= 0) and (Arr[j]>temp) do

begin

// Move Arr[j] to Arr[j+1]

Arr[j+1] := Arr[j];

// Decrement index to move

// Arr[j] further

j := j - 1;

// Increment Result after swap

Result := Result + 1;

end;

// Make Arr[j+1] := temp after moving it forward

Arr[j+1] := temp;

// Increment Result after swap

Result := Result + 1;

end;

end;

// Quick sort

// ArrGiven - given array

function QuickSort(ArrGiven: TArray): Integer;

var

// Arr - copy of given array;

// St - stack

// length - length of Arr

// i,j - indexes

// S, L, x - parameters for non recursive quick sort

// temp - temp parameter for swap

Arr: TArray;

St: Array of Array of Integer;

length: Integer;

i, j, S, L, R, x, temp: Integer;

begin

// Copy given array

Arr := Copy(ArrGiven);

// Get length of array

Length := High(Arr) + 1;

SetLength(St, Length, Length);

Result := 0;

S := 1;

St[1,1] := 1;

St[1,2] := Length;

repeat

L := St[S,1];

R := St[S,2];

S := S - 1;

repeat

i := L;

j := R;

x := Arr[(L+R) div 2];

repeat

while Arr[i] < x do

i := i + 1;

while Arr[j] > x do

j := j - 1;

if i <= j then

begin

temp := Arr[i];

Arr[i] := Arr[j];

Arr[j] := temp;

Result := Result + 1;

i := i + 1;

j := j - 1;

end;

until i > j;

if i < R then

begin

S := S + 1;

St[S,1] := i;

St[S,2] := R;

end;

R := j;

until L >= R;

until S = 0;

end;

// ArrLengths - array that contains lengths

// of arrays to calculate

const

ArrLengths: TArray =

[100, 250, 500, 1000, 2000, 3000];

// i - index

// ArrSorted - sorted array length ArrLengths[i]

// ArrReversed - reversed array length ArrLengths[i]

// ArrRandom - random array length ArrLengths[i]

var

i: Integer;

ArrSorted, ArrReversed, ArrRandom: TArray;

begin

// Iterate through lengths

for i := 0 to 5 do

begin

// Create 3 types of array

ArrRandom := Create(1, ArrLengths[i]);

ArrReversed := Create(2, ArrLengths[i]);

ArrSorted := Create(3, ArrLengths[i]);

// Output result of random array sorts

write('N = ', ArrLengths[i]:4, ', Random, ');

write(InsertionSort(ArrRandom):8, ' ');

writeln(QuickSort(ArrRandom):8);

// Output result of reversed array sorts

write('N = ', ArrLengths[i]:4, ', Reversed, ');

write(InsertionSort(ArrReversed):8, ' ');

writeln(QuickSort(ArrReversed):8);

// Output results of sorted array sorts

write('N = ', ArrLengths[i]:4, ', Sorted, ');

write(InsertionSort(ArrSorted):8, ' ');

writeln(QuickSort(ArrSorted):8);

end;

// Stop console from closing

readln;

end.

Приложение Б

(обязательное)

Результаты расчета

Таблица 9 – Результаты расчетов программы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность массива | Тип  массива | 1-ый метод сортировки | | 2-ой метод сортировки | |
| Кол-во экспериментальное | Кол-во теоретическое | Кол-во экспериментальное | Кол-во теоретическое |
| N = 100 | Случайный | 2756 | 2750 | 176 | 180 |
| Сортированный | 100 | 100 | 63 | 60 |
| Перевернутый | 4951 | 4950 | 112 | 110 |
| N = 250 | Случайный | 15392 | 15400 | 546 | 560 |
| Сортированный | 250 | 250 | 127 | 125 |
| Перевернутый | 31126 | 31100 | 251 | 250 |
| N = 500 | Случайный | 65619 | 65500 | 1163 | 1200 |
| Сортированный | 500 | 500 | 255 | 250 |
| Перевернутый | 124751 | 124750 | 504 | 500 |
| N = 1000 | Случайный | 257270 | 260000 | 2725 | 2800 |
| Сортированный | 1000 | 1000 | 511 | 500 |
| Перевернутый | 499502 | 500000 | 1011 | 1000 |
| N = 2000 | Случайный | 984547 | 990000 | 5976 | 6000 |
| Сортированный | 2000 | 2000 | 1023 | 1023 |

Таблица 10 **–** Продолжение таблицы 9

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Перевернутый | 2000002 | 2000002 | 2023 | 2000 |
| N = 3000 | Случайный | 2286279 | 2252249 | 9470 | 94703 |
| Сортированный | 3000 | 3000 | 1976 | 1976 |
| Перевернутый | 4500502 | 4498500 | 3475 | 11052 |

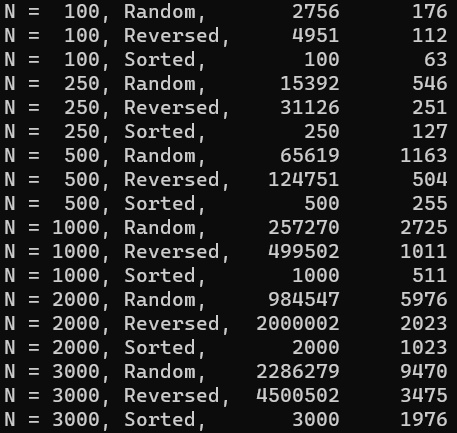


Рисунок 7 – Результаты расчетов программы