Содержание

[1 Постановка задачи 5](#_Toc132066860)

[1.1 Начальная постановка задачи 5](#_Toc132066861)

[1.2 Конечная постановка 5](#_Toc132066862)

[2 Методика решения 6](#_Toc132066863)

[2.1 Быстрая сортировка 6](#_Toc132066864)

[2.2 Сортировка вставками 6](#_Toc132066865)

[3 Описание алгоритмов решения задачи 7](#_Toc132066866)

[4 Структура данных 9](#_Toc132066867)

[5 Схема алгоритма решения задачи по ГОСТ 19.701-90 12](#_Toc132066868)

[5.1 Схема алгоритма GenerateArray 14](#_Toc132066869)

[5.2 Схема алгоритма outputArray 15](#_Toc132066870)

[5.3 Схема алгоритма insertionSort 16](#_Toc132066871)

[5.4 Схема алгоритма quickSort 18](#_Toc132066873)

[5.5 Схема алгоритма swap 21](#_Toc132066874)

[6 Результаты расчетов 22](#_Toc132066875)

[Приложение А 23](#_Toc132066876)

[Приложение Б 30](#_Toc132066877)

# Постановка задачи

Провести сравнительный анализ быстрой сортировки и сортировки методом «“плавающего пузырька” по числу перестановок.

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

Типы массивов:

* случайный;
* сортированный;
* перевернутый.

Разработать структуру данных для хранения результатов расчетов.

# Методика решения

## Быстрая сортировка

Для теоретического расчета количества перестановок элементов массивов использовались формулы:

1. Случайный массив .
2. Сортированный массив .
3. Перевернутый массив .

Где N – число перестановок, Size – число элементов в массиве.

## Сортировка плавающим пузырьком

Для теоретического расчета количества сравнений элементов массивов использовались формулы:

1. Случайный массив
2. Сортированный массив .
3. Перевернутый массив

Где N – число перестановок, Size – число элементов в массиве.

# Описание алгоритмов решения задачи

Таблица – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п. | Наименование алгоритма | Назначение алгоритма | Формальные  параметры | Предпола–гаемый тип реализации |
| 1 | Основной алгоритм | Вызов следующих подпрограмм:  GenerateArray,  QuickSort, BubleSort |  |  |
| 2 | GenerateArray  (x,  i,  Method) | Выполняет генерацию массива х длиной i. Method определяется пользователем и задает способ генерации массива: 1 – случайный, 2 – сортированный, 3 – перевернутый | x– получает от фактического параметра адрес;  i – получает от фактического параметра адрес c защитой;  Method – получает от фактического параметра адрес с защитой | Процедура |
| 3 | outputArray(x, i) | Выводит массив x длиной i | x – получает от фактического параметра адрес c защитой;  i – получает от фактического параметра адрес c защитой | Процедура |
| 4 | BubleSort(arr, n,  compareCount) | Сортировка вставками массива arr длиной n.  compareCount – счетчик операций сравнения | arr – получает от фактического параметра копию значения;  n – получает от фактического параметра адрес c защитой; | Процедура |

Продолжение Таблицы 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  | compareCount – получает от фактического параметра адрес |  |
| 5 | Quicksort(arr; high, compareCount) | Быстрая сортировка массива arr длиной count.  compareCount – счетчик операций сравнения | arr – получает от фактического параметра копию значения;  count – получает от фактического параметра адрес c защитой; high – получает от фактического параметра размерность массива flag – получает от фактического параметра адрес c защитой;  compareCount – получает от фактического параметра адрес | Процедура |
| 6 | swap(x, y) | Переставляет элементы x и y местами | x – получает от фактического параметра адрес;  y – получает от фактического параметра адрес | Процедура |

# Структура данных

Таблица – Структура данных основной программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| Table | Тип пользователя «запись» со следующим набором полей:  Record  FillingMethod: byte;  count, NExpBuble,  NExpQuick: Integer;  NTeorBuble,  NTeorQuick: real;  end; | Запись содержит поля:  FillingMethod – способ заполнения массива;  count – количество элементов в массиве;  NExpBuble – экспериментальное количество сравнений в сортировке вставками;  NExpQuick – экспериментальное количество сравнений в быстрой сортировке;  NTeorBuble – теоретическое количество сравнений в сортировке вставками;  NTeorQuick – теоретическое количество сравнений в быстрой сортировке |
| Res | Array of Table | Массив для построения таблицы |
| J | Integer | Счетчик цикла |
| Info | Table | Запись с информацией о сортировках |
| MainArray | array of Integer | Массив для передачи в процедуры сортировки |

Таблица – Структура данных алгоритма GenerateArray(x, i, Method)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| j | Integer | Счетчик цикла | Локальный |
| x | Array of Integer | Генерируемый массив | Формальный |
| i | Integer | Длина массива | Формальный |
| Method | Byte | Метод заполнения массива | Формальный |

Таблица – Структура данных алгоритма outputArray(x, i)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | | Тип параметра |
| j | Integer | Счетчики цикла | | Локальный |
| x | Array of Integer | Массив для вывода | Формальный | |
| i | Integer | Длина массива | Формальный | |

Таблица 5 – Структура данных алгоритма BubleSort(arr, n,

compareCount)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | | Назначение | Тип параметра |
| k, j | Integer | | Счетчики циклов | Локальный |
| Arr | | Array of Integer | Сортируемый массив | Формальный |
| CompareCount | | Integer | Счетчик операций сравнения | Формальный |
| I | | Integer | Длина массива | Формальный |

Таблица 6 – Структура данных алгоритма Quicksort(arr; high, compareCount)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| pivot | Integer | Осевой элемент | Локальный |
| High | Integer | Длина массива для сортировки | Локальный |
| i, j | Integer | Счетчики циклов | Локальный |
| tmp | Integer | Переставляемый элемент | Локальный |
| arr | Array of Integer | Сортируемый массив | Формальный |
| CompareCount | Integer | Счетчик операций перестановок | Формальный |
| n | Integer | Длина массива | Формальный |

Таблица – Структура данных алгоритма swap(x, y)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| tmp | Integer | Переменная для перестановки двух элементов | Локальный |
| x, y | Integer | Переставляемые элементы | Формальный |

# Схема алгоритма решения задачи по ГОСТ 19.701-90



Рисунок – Схема алгоритма в соответствии с ГОСТ 19.701-90 (часть 1)



Рисунок – Схема алгоритма в соответствии с ГОСТ 19.701-90 (часть 2)

## Схема алгоритма GenerateArray



Рисунок – Схема алгоритма GenerateArray (часть 2)

## Схема алгоритма outputArray



Рисунок – Схема алгоритма outputArray

## Схема алгоритма BubleSort

## 

Рисунок – Схема алгоритма insertionSort (часть 1)



Рисунок – Схема алгоритма BubleSort (часть 2)

## Схема алгоритма QuickSort



Рисунок – Схема алгоритма QuickSort (часть 1)



Рисунок – Схема алгоритма QuickSort (часть 2)



Рисунок 11 – Схема алгоритма QuickSort (часть 3)

## Схема алгоритма swap



Рисунок 12 – Схема алгоритма swap

# Результаты расчетов

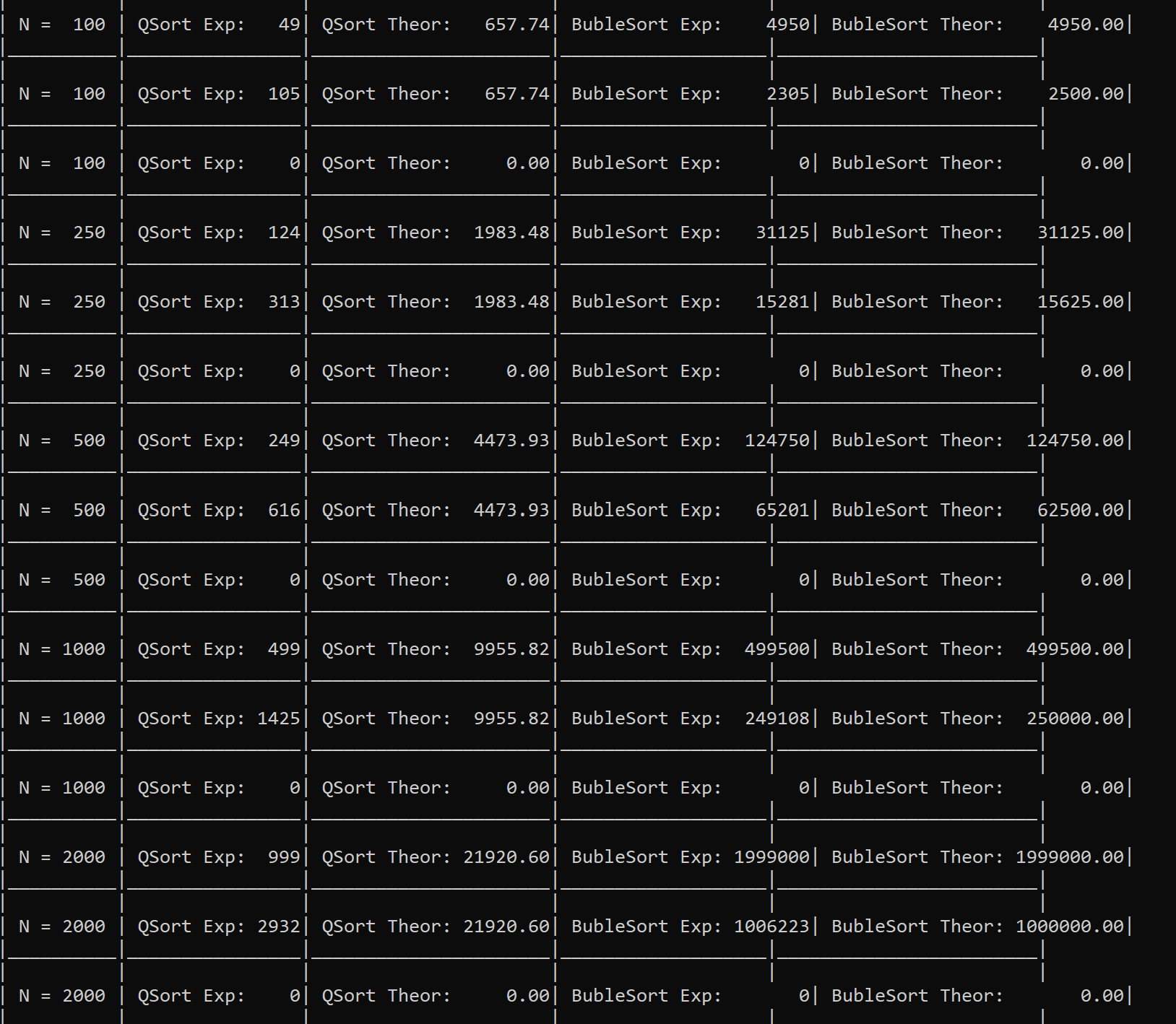


Рисунок 13 – Результаты расчётов

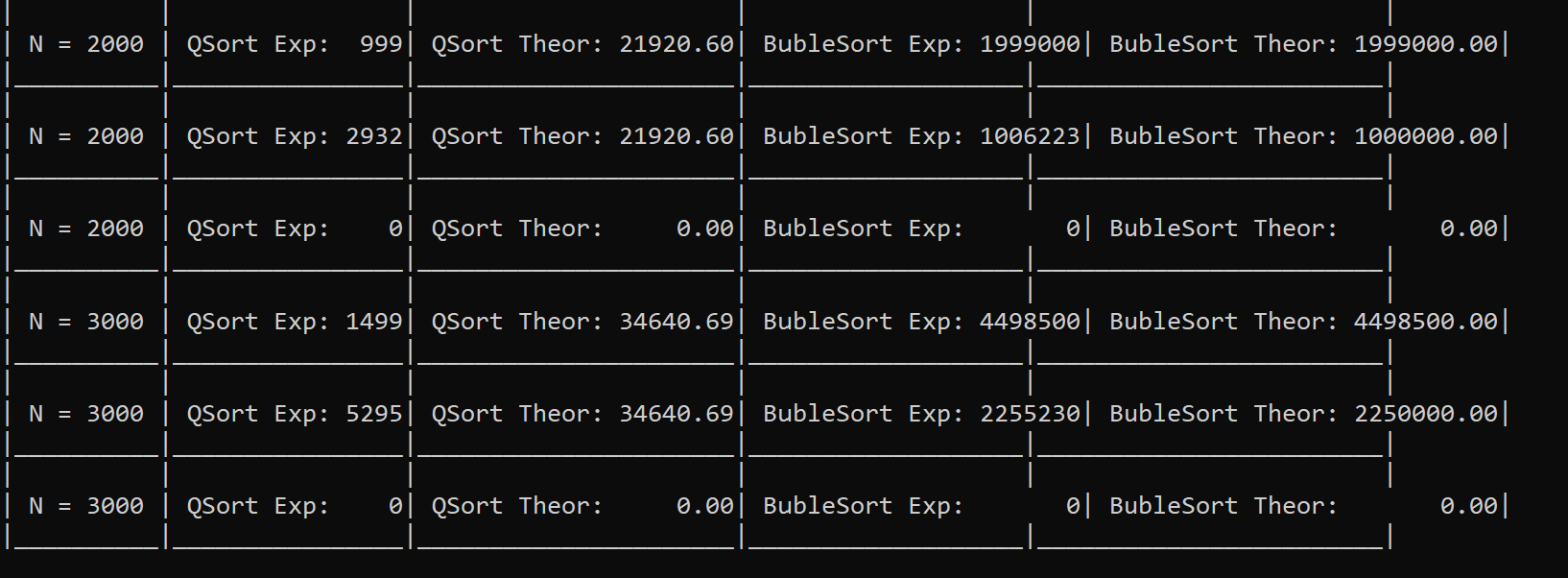


Рисунок 14 – Результаты расчётов

Приложение А

(обязательное)

Исходный код программы

program lab2;

{

Conduct a comparative analysis of quick sorting

and sorting by inserts according to the number

of comparisons.

The dimension of the array and its type are entered by

the user.

Develop a data structure for storing calculation

results.

}

// Determining the Console Program Type

{$APPTYPE CONSOLE}

uses

System.SysUtils;

type

Table = Record

FillingMethod: byte;

count, NExpBuble, NExpQuick: Integer;

NTeorBuble, NTeorQuick: real;

end;

Res = Array [0 .. 17] of Table;

var

Flag: byte;

Arr: array of Integer;

Info: Res;

j: Integer;

{

Flag - a variable for the user to select the output of

the array

Arr - array for sorting

Info - information about sorting

}

// Procedure for swapping two elements using the third

procedure swap(var x, y: Integer);

var

tmp: Integer;

// tmp - saving the value

begin

tmp := x;

x := y;

y := tmp;

end;

// Procedure for creating an array

procedure GenerateArray(var x: array of Integer; const i: Integer;

const Method: byte);

var

j: Integer;

// j - counter for the loop

begin

// Depending on the user's choice, the array is filled

// with the necessary elements

for j := 0 to i - 1 do

case Method of

1:

x[j] := random(10000);

2:

x[j] := j;

3:

x[j] := i - 1 - j;

end;

end;

// Output of array values

procedure outputArray(const x: array of Integer; const i: Integer);

var

j: Integer;

// j - counter for output of values

begin

for j := 0 to i - 1 do

write(x[j], ' ');

end;

// Procedure for sorting buble

procedure BubleSort(Arr: array of Integer;

const i:integer; var ChangeCount:int) ;

var j,k:integer;

begin

ChangeCount:=0;

for k:=0 to i-2 do

for j:=0 to i-2 do

if Arr[j]>Arr[j+1] then

begin

inc(ChangeCount);

swap(Arr[j],Arr[j+1]);

k:=j;

while k>= 2 do

if Arr[k]<Arr[k-1] then

begin

inc(ChangeCount);

swap(Arr[k],Arr[k+1]);

j:=j+1;

end;

end;

// Procedure for performing quick sorting

procedure QuickSort(Arr: array of Integer; high: Integer; const Flag: byte;

var ChangeCount: Integer);

var

Pivot, i, j, stlast, low: Integer;

Stack: Array [1 .. 13, 1 .. 2] of Integer;

// define the stack for storing subarray indices

begin

low := 1;

changeCount := 0;

stlast := 2;

Stack[1, 1] := low;

// initialize the stack with the indices of the initial subarray

Stack[1, 2] := high;

// loop until the stack is empty

while stlast <> 1 do

begin

low := Stack[stlast - 1, 1];

// get the indices of the subarray at the top of the stack

high := Stack[stlast - 1, 2];

stlast := stlast - 1; // pop the top element off the stack

if Low < High then

// if the subarray has more than one element, partition it

begin

Pivot := (Low + High) div 2; // choose a pivot element

i := Low;

j := High;

// loop until the indices cross

while i < j do

begin

while (Arr[i] <= Arr[Pivot]) //and (i < High) do

// move the left pointer until it points to an element greater than the pivot

Inc(i);

while Arr[j] > Arr[Pivot] and (j > Low) do

// move the right pointer until it points to an element smaller than or equal to the pivot

Dec(j);

if i < j then

// if the pointers have not crossed, swap the elements

// they point to

begin

swap(Arr[i], Arr[j]); // swap the elements

Inc(changeCount); // increment the comparison counter

end;

end;

// swap the pivot element with the element at the right pointer

//swap(Arr[Pivot], Arr[j]);

// push the indices of the left and right subarrays onto the stack

Stack[stlast][1] := Low;

Stack[stlast, 2] := j - 1;

stlast := stlast + 1;

Stack[stlast][1] := j + 1;

Stack[stlast, 2] := high;

stlast := stlast + 1;

end;

end;

// output the resulting array if the user wants it

if Flag = 1 then

outputArray(Arr, high);

end;

begin

for j := 0 to 17 do

begin

Arr := 0;

case j mod 3 of

0:

Info[j].FillingMethod := 3;

1:

Info[j].FillingMethod := 1;

2:

Info[j].FillingMethod := 2;

end;

case j div 3 of

0:

Info[j].count := 100;

1:

Info[j].count := 250;

2:

Info[j].count := 500;

3:

Info[j].count := 1000;

4:

Info[j].count := 2000;

5:

Info[j].count := 3000;

end;

// Setting the array length and generating it

SetLength(Arr, Info[j].count);

GenerateArray(Arr, Info[j].count, Info[j].FillingMethod);

// Calculation of theoretical values of comparisons

case Info[j].FillingMethod of

1:

begin

Info[j].NTeorQuick := (Info[j].count - 1) \* ln(Info[j].count)

/ ln(2);

Info[j].NTeorBuble := Info[j].count \* (Info[j].count) / 4;

end;

2:

begin

Info[j].NTeorQuick := 0;

Info[j].NTeorBuble := 0;

end;

3:

begin

Info[j].NTeorQuick := (Info[j].count - 1) \* ln(Info[j].count)

/ ln(2);

Info[j].NTeorBuble := Info[j].count \* (Info[j].count - 1) / 2;

end;

end;

// Implementation of sorting

writeln('| | | | | |');

QuickSort(Arr, Info[j].count, Info[j].NExpQuick);

BubleSort(Arr, Info[j].count, Info[j].NExpBuble);

writeln('| N = ', Info[j].count:4, ' | QSort Exp: ', Info[j].NExpQuick:4,

'| QSort Theor: ', Info[j].NTeorQuick:8:2, '| BubleSort Exp: ',

Info[j].NExpBuble:7, '| BubleSort Theor: ', Info[j].NTeorBuble:10:2, '|');

writeln('|\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|');

// writeln('| N = 3000 | QSort Exp: 0| QSort Theor: 0.00| BubleSort Exp: 0| BubleSort Theor: 0.00|');

end;

Readln;

end.

Приложение Б

(обязательное)

Тестовые наборы

Таблица – расчеты

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность массива | Тип массива | Быстрая сортировка | | Сортировка пузырьком | |
| Экспериментальное | Теоретическое | Экспериментальное | Теоретическое |
| N=100 | случайный | 49 | 657 | 4950 | 4950 |
| N=100 | сортированный | 0 | 0 | 0 | 0 |
| N=100 | перевернутый | 105 | 657 | 2305 | 2500 |
| N=250 | случайный | 124 | 1983 | 31125 | 31125 |
| N=250 | сортированный | 0 | 0 | 0 | 0 |
| N=250 | перевернутый | 313 | 1983 | 15281 | 15625 |
| N=500 | случайный | 249 | 4473 | 124750 | 124750 |
| N=500 | сортированный | 0 | 0 | 0 | 0 |
| N=500 | перевернутый | 616 | 4473 | 65201 | 62500 |
| N=1000 | случайный | 499 | 9955 | 499500 | 499500 |
| N=1000 | сортированный | 0 | 0 | 0 | 0 |
| N=1000 | перевернутый | 1425 | 9955 | 249108 | 250000 |
| N=2000 | случайный | 999 | 21920 | 1999000 | 1999000 |
| N=2000 | сортированный | 0 | 0 | 0 | 0 |
| N=2000 | перевернутый | 2932 | 21920 | 1006223 | 1000000 |
| N=3000 | случайный | 1499 | 34640 | 4498500 | 4498500 |
| N=3000 | сортированный | 0 | 0 | 0 | 0 |
| N=3000 | перевернутый | 5295 | 34640 | 2255230 | 2250000 |