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. 251003 Габрусь С.П.

Проверил: Фадеева Е.П.

Минск 2022

**СОДЕРЖАНИЕ**

[1 Постановка задачи 3](#_Toc81231046)

[2 Текстовый алгоритм решения задачи 4](#_Toc81231047)

[3 Структура данных 5](#_Toc81231048)

[4 Схема алгоритма решения задачи по ГОСТ 19.701-90 6](#_Toc81231049)

[5 Результаты расчетов 7](#_Toc81231050)

[Приложение А 8](#_Toc81231051)

[Приложение Б 9](#_Toc81231052)

# Постановка задачи

Для сортировок методами выбора и Шелла провести сравнительный анализ указанных методов сортировки массивов.

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

Типы массивов: случайный, сортированный, перевернутый.

Разработать структуру данных для хранения результатов расчета. Согласно полученным результатам расчетов сделать соответствующие выводы.

Результаты расчетов свести в таблицу. Оформление таблицы должно соответствовать таблице 1.

Таблица 1 – Пример оформления таблицы

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Размерность массива | Тип массива | Сортировка Шелла | | Сортировка выбором | |
| Количество экспериментальное | Количество теоретическое | Количество экспериментальное | Количество теоретическое |
| N=100 |  |  |  |  |  |
| … |  |  |  |  |  |
| N=3000 |  |  |  |  |  |

# Методика решения

## Структура данных для хранения результатов анализа

Для хранения результатов анализа целесообразным будет использовать пользовательский тип следующего вида:

TResults = record

RandArrays: TAmOfIters;

SortedArrays: TAmOfIters;

ReversedArrays: TAmOfIters;

end;

, где TAmOfIters:

TAmOfIters = record

ShellTheor: integer;

ShellPract: integer;

SelectionTheor: integer;

SelectionPract: integer;

end;

Причем инициализируем массив следующего вида:

ResultsOfSorts: array [1..6] of TResults;

, где каждому элементу типа TResults будет соответствовать количество элементов в анализируемых массивах из AmountsOfEl:

AmountsOfEl: array [1..6] of Integer =

(100, 250, 500, 1000, 2000, 3000);

Данная структура позволяет удобно и компактно хранить информацию о количестве сравнений элементов при сортировке для различных типов входных данных (рандомные, отсортированные и отсортированные в обратном порядке) и для разного количества элементов в массиве.

Использование структуры TAmOfIters позволяет раздельно хранить теоретическое и практическое количество сравнений, что может быть полезным при анализе эффективности алгоритмов сортировки.

Также использование массива ResultsOfSorts, где каждому элементу соответствует конкретное количество элементов в анализируемых массивах, обеспечивает удобный доступ к результатам сортировок и позволяет быстро находить нужную информацию.

В целом, данная структура данных позволяет компактно, и удобно хранить и анализировать результаты сортировок для различных типов данных и размеров массивов, что делает ее подходящей для данного типа задачи.

## Результаты анализа

# Описание алгоритмов решения задачи

Таблица 2 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п. | Наименование алгоритма | Описание алгоритма | Формальные  параметры | Предполагаемый тип реализации |
| 1 | MakeArray  (  AArrToMake, AAmountOfEl, ARequest  ) | Заполняет переданный массив ArrToMake элементами, в количестве AmountOfEl, в соответствие с запросом Request | ArrToMake – получает от фактического параметра адрес, возвращаемый параметр;  AmountOfEl – получает от фактического параметра адрес c защитой;  Request – получает от фактического параметра адрес c защитой | Процедура |
| 2 | Swap  (  AFirstEl,  ASecondEl  ) | Меняет значения передаваемых параметров между собой | AFirstEl – получает от фактического параметра адрес, возвращаемый параметр;  ASecondEl – получает от фактического параметра адрес, возвращаемый параметр | Процедура |
| 3 | ShellSort  (  AArrToSort,  AAmountOfEl,  ARealAmOfIters  ) | Процедура, анализирующая количество сравнений элементов массива для метода Шелла | AArrToSort – получает от фактического параметра значение;  AAmountOfEl – получает от фактического параметра адрес c защитой;  ARealAmOfIters – получает от фактического параметра адрес, возвращаемый параметр | Процедура |

Продолжение таблицы 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 4 | SelectionSort  (  AArrToSort,  AAmountOfEl,  ARealAmOfIters  ) | Процедура, анализирующая количество сравнений элементов массива для метода выбора | AArrToSort – получает от фактического параметра значение;  AAmountOfEl – получает от фактического параметра адрес c защитой;  ARealAmOfIters – получает от фактического параметра адрес, возвращаемый параметр | Процедура |
| 5 | ShellTheorCalc  (  AAmOfElements,  Result  ) | Процедура, вычисляющая теоретическое количество сравнений для метода Шелла | AAmOfElements – получает от фактического параметра адрес c защитой;  Result – получает от фактического параметра адрес, возвращаемый параметр | Функция  Result – возвращаемый параметр |
| 6 | SelectionTheorCalc  (  AAmOfElements,  Result  ) | Процедура, вычисляющая теоретическое количество сравнений для метода выбора | AAmOfElements – получает от фактического параметра адрес c защитой;  Result – получает от фактического параметра адрес, возвращаемый параметр | Функция  Result – возвращаемый параметр |

# Структура данных

Таблица 3 – Структура типов основного алгоритма

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| TArray | Array [1..3000] of Integer | Тип, предназначенный для объявления массивов чисел, необходимых к последующей сортировке |
| TResults | Record  RandArrays: TAmOfIters;  SortedArrays: TAmOfIters;  ReversedArrays: TAmOfIters;  End; | Тип, представляющий собой вложенную запись и предназначенный для хранения результатов анализа сортировок для заданного количества элементов:  RandArrays – результаты анализа сортировок для массива со случайным набором чисел;  SortedArrays – результаты анализа сортировок для отсортированного по возрастанию массива;  ReversedArrays – результаты анализа сортировок для отсортированного по убыванию массива |
| TAmOfIters | Record  ShellTheor: integer;  ShellPract: integer;  SelectionTheor: integer;  SelectionPract: integer;  End; | Тип, вложенный в тип TResults и предназначенный для хранения теоретических и практических результатов анализа сортировок Шелла и выбором:  ShellTheor – теоретическое количество сравнений в сортировке Шелла;  ShellPract – практическое количество сравнений в сортировке Шелла;  SelectionTheor – теоретическое количество сравнений в сортировке выбором;  SelectionPract – практичесское количество сравнений в сортировке выбором |

Таблица 4 – Структура данных основного алгоритма

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| AmountsOfEl | Array [1..6] of Integer = (100, 250, 500, 1000, 2000, 3000) | Возможное количество элементов, сортировку которых необходимо проанализировать |
| RandElements,  SortElements,  RevElements | TArray | Массивы, состоящие из случайно взятых чисел, чисел, отсортированных по возрастанию и чисел, отсортированных по убыванию, соответственно |
| ResultsOfSorts | Array [1..6] of TResults | Массив, состоящий из элементов типа TResults, причем каждый элемент соответствует по индексам размерам взятых массивов из AmountsOfEl |
| i | Integer | Счетчик для циклов |

Таблица 5 – Структура данных алгоритма MakeArray(AArrToMake, AAmountOfEl, ARequest)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| i | Integer | Счетчик для циклов | Локальный |
| AArrToMake | TArray | Массив, необходимый для заполнения элементами | Формальный |
| AAmountOfEl | Integer | Количество элементов, необходимых для заполнения | Формальный |
| ARequest | String | Правило, по которому будет заполняться переданный массив:  ‘Rand’ – в случайном порядке  ‘Sorted’ – в порядке возрастания  ‘Reversed’ – в порядке убывания | Формальный |

Таблица 6 – Структура данных алгоритма Swap(AFirstEl, ASecondEl);

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Buf | Integer | Буферная переменная для обмена значений | Локальный |
| AFirstEl,  ASecondEl | Integer | Переменные, значения которых необходимо поменять | Формальный |

Таблица 7 – Структура данных алгоритма ShellSort(AArrToSort, AAmountOfEl, ARealAmOfIters)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| i, j | Integer | Счетчики для циклов | Локальный |
| AmOfSteps | Integer | Количество расстояний для шагов | Локальный |
| CurrStep | Integer | Текущее расстояние для шага | Локальный |
| AArrToSort | TArray | Сортируемый массив | Формальный |
| AAmountOfEl | Integer | Количество элементов, необходимых для сортировки | Формальный |
| ARealAmOfIters | Integer | Практическое количество сравнений элементов массива | Формальный |

Таблица 8 – Структура данных алгоритма SelectionSort(AArrToSort, AAmountOfEl, ARealAmOfIters)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| i, j | Integer | Счетчики для циклов | Локальный |
| СurrMin | Integer | Минимальный элемент для текущей проходки по массиву | Локальный |
| AArrToSort | TArray | Сортируемый массив | Формальный |
| AAmountOfEl | Integer | Количество элементов, необходимых для сортировки | Формальный |

Продолжение таблицы 8

|  |  |  |  |
| --- | --- | --- | --- |
| ARealAmOfIters | Integer | Практическое количество сравнений элементов массива | Формальный |

Таблица 9 – Структура данных алгоритма ShellTheorCalc(AAmOfElements, Result)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| AAmOfElements | Integer | Количество элементов в массиве | Формальный |
| Result | Integer | Теоретическое количество сравнений элементов массива при сортировке методом Шелла | Формальный |

Таблица 10 – Структура данных алгоритма SelectionTheorCalc(AAmOfElements, Result)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| AAmOfElements | Integer | Количество элементов в массиве | Формальный |
| Result | Integer | Теоретическое количество сравнений элементов массива при сортировке методом выбора | Формальный |

# Схема алгоритма решения задачи по ГОСТ 19.701-90



Рисунок 1 – Схема основного алгоритма решения задачи по ГОСТ 19.701-90 (часть 1)



Рисунок 2 – Схема основного алгоритма решения задачи по ГОСТ 19.701-90 (часть 2)



Рисунок 3 – Схема алгоритма MakeArray по ГОСТ 19.701-90



Рисунок 4 – Схема алгоритма Swap по ГОСТ 19.701-90



Рисунок 5 – Схема алгоритма ShellTheorCalc по ГОСТ 19.701-90



Рисунок 6 – Схема алгоритма SelectionTheorCalc по ГОСТ 19.701-90



Рисунок 7 – Схема алгоритма ShellSort по ГОСТ 19.701-90 (часть 1)



Рисунок 8 – Схема алгоритма ShellSort по ГОСТ 19.701-90 (часть 2)



Рисунок 9 – Схема алгоритма SelectionSort по ГОСТ 19.701-90 (часть 1)



Рисунок 10 – Схема алгоритма SelectionSort по ГОСТ 19.701-90 (часть 2)

# Результаты расчетов

После выполнения программы на экран выводятся результаты анализа сортировок в табличном виде:

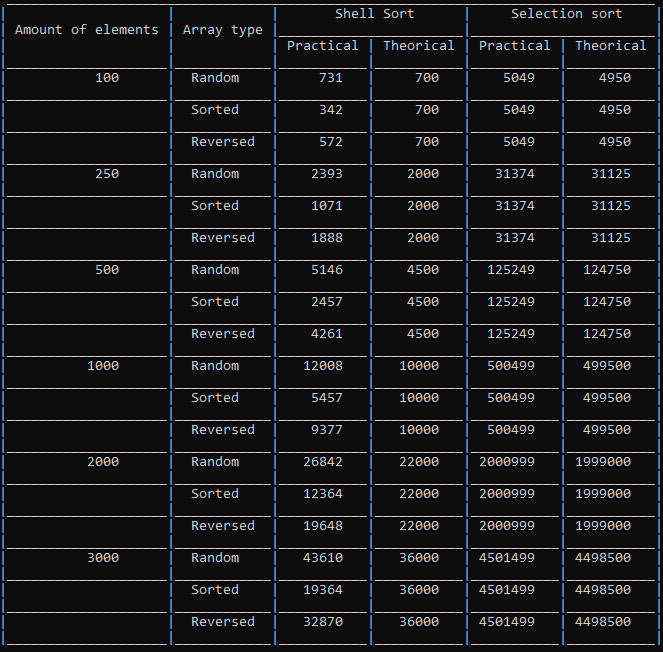


Рисунок 11 – Результаты расчетов

Приложение А

(обязательное)

Исходный код основного блока программы

Program lab2;

{$APPTYPE CONSOLE}

uses

System.SysUtils,

SortAnalyze in 'SortAnalyze.pas';

Const

AmountsOfEl: array [1..6] of Integer = (100, 250, 500, 1000, 2000, 3000);

//AmountsOfEl - possible amounts of elements

Var

RandElements, SortElements, RevElements: TArray;

ResultsOfSorts: array [1..6] of TResults;

i: integer;

//Elements - user elements to sort

//ResultsOfSorts - struct of theorical and practical

//amounts of iterations

//i - iterator for cycles

//Proc for making array

Procedure MakeArray(var AArrToMake: TArray; const AAmountOfEl: integer; const ARequest: string);

Var

i: integer;

//i - iterator for cycles

Begin

//Arr with random numbers

if ARequest = 'Rand' then

for i := 1 to AAmountOfEl do

begin

Randomize;

AArrToMake[i] := Random(100) + 1;

end

else

//Arr with sorted numbers

if ARequest = 'Sorted' then

for i := 1 to AAmountOfEl do

begin

AArrToMake[i] := i;

end

else

//Arr with sorted and reversed numbers

if ARequest = 'Reversed' then

for i := AAmountOfEl downto 1 do

begin

AArrToMake[AAmountOfEl - i + 1] := i;

end;

End;

Begin

//Making arrays with random, sorted and reversed

//elements

MakeArray(RandElements, 3000, 'Rand');

MakeArray(SortElements, 3000, 'Sorted');

MakeArray(RevElements, 3000, 'Reversed');

//Going trough all amounts of elements

for i := 1 to length(AmountsOfEl) do

begin

//Getting theorical amount of itarations

ResultsOfSorts[i].RandArrays.ShellTheor := ShellTheorCalc(AmountsOfEl[i]);

ResultsOfSorts[i].RandArrays.SelectionTheor :=

SelectionTheorCalc(AmountsOfEl[i]);

ResultsOfSorts[i].SortedArrays.ShellTheor := ShellTheorCalc(AmountsOfEl[i]);

ResultsOfSorts[i].SortedArrays.SelectionTheor := SelectionTheorCalc(AmountsOfEl[i]);

ResultsOfSorts[i].ReversedArrays.ShellTheor := ShellTheorCalc(AmountsOfEl[i]);

ResultsOfSorts[i].ReversedArrays.SelectionTheor := SelectionTheorCalc(AmountsOfEl[i]);

//RANDOM PART

//Getting practical amount of itarations

//Shell sort

ShellSort(RandElements, AmountsOfEl[i],

ResultsOfSorts[i].RandArrays.ShellPract);

//Selection sort

SelectionSort(RandElements, AmountsOfEl[i],

ResultsOfSorts[i].RandArrays.SelectionPract);

//SORTED PART

//Getting practical amount of itarations

//Shell sort

ShellSort(SortElements, AmountsOfEl[i],

ResultsOfSorts[i].SortedArrays.ShellPract);

//Selection sort

SelectionSort(SortElements, AmountsOfEl[i],

ResultsOfSorts[i].SortedArrays.SelectionPract);

//REVERSED PART

//Getting practical amount of itarations

//Shell sort

ShellSort(RevElements, AmountsOfEl[i],

ResultsOfSorts[i].ReversedArrays.ShellPract);

//Selection sort

SelectionSort(RevElements, AmountsOfEl[i],

ResultsOfSorts[i].ReversedArrays.SelectionPract);

end;

//Outputting results in table form

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

writeln('| | | Shell Sort | Selection sort |');

writeln('| Amount of elements | Array type |\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|');

writeln('| | | Practical | Theorical | Practical | Theorical |');

for i := 1 to length(AmountsOfEl) do

begin

//Part with random

writeln('|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_

\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|');

writeln('|', AmountsOfEl[i]:14, ' | Random |', ResultsOfSorts[i].RandArrays.ShellPract:8, ' |', ResultsOfSorts[i].RandArrays.ShellTheor:8, ' |', ResultsOfSorts[i].RandArrays.SelectionPract:8, ' |', ResultsOfSorts[i].RandArrays.SelectionTheor:8, ' |');

//Part with sorted

writeln('|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_

\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|');

writeln('| | Sorted |',

ResultsOfSorts[i].SortedArrays.ShellPract:8, ' |', ResultsOfSorts[i].SortedArrays.ShellTheor:8, ' |', ResultsOfSorts[i].SortedArrays.SelectionPract:8, ' |', ResultsOfSorts[i].SortedArrays.SelectionTheor:8, ' |');

//Part with reverse

writeln('|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_

\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|');

writeln('| | Reversed |', ResultsOfSorts[i].ReversedArrays.ShellPract:8, ' |', ResultsOfSorts[i].ReversedArrays.ShellTheor:8, ' |', ResultsOfSorts[i].ReversedArrays.SelectionPract:8, ' |', ResultsOfSorts[i].ReversedArrays.SelectionTheor:8, ' |');

end;

writeln('|\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_

\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|\_\_\_\_\_\_\_\_\_\_\_|');

readln;

End.

Приложение Б

(обязательное)

Тестовые наборы

Тест 1

Тестовая ситуация:

Исходные данные:

Ожидаемый результат:

Полученный результат:

Тест 2

Тестовая ситуация:

Исходные данные:

Ожидаемый результат:

Полученный результат:

…

Приложение С

Исходный код блока SortAnalyze

Unit SortAnalyze;

Interface

Type

TArray = array [1..3000] of Integer;

TAmOfIters = record

ShellTheor: integer;

ShellPract: integer;

SelectionTheor: integer;

SelectionPract: integer;

end;

TResults = record

RandArrays: TAmOfIters;

SortedArrays: TAmOfIters;

ReversedArrays: TAmOfIters;

end;

Procedure ShellSort(AArrToSort: TArray; const

AAmountOfEl: integer; var ARealAmOfIters: integer);

Procedure SelectionSort(ArrToSort: TArray; const

AAmountOfEl: integer; var ARealAmOfIters: integer);

Function ShellTheorCalc(const AAmOfElements: Integer): Integer;

Function SelectionTheorCalc(const AAmOfElements:

Integer): Integer;

Implementation

//Proc to swap value of variables

Procedure Swap(var AFirstEl, ASecondEl: integer);

Var

Buf: integer;

//Buf - variable to swap to elements

Begin

Buf := AFirstEl;

AFirstEl := ASecondEl;

ASecondEl := Buf;

End;

//Proc for ShellSort

Procedure ShellSort(AArrToSort: TArray; const

AAmountOfEl: integer; var ARealAmOfIters: integer);

Var

i, j, AmOfSteps, CurrStep: integer;

//i, j - iterators

//AmOfStep - amount of steps for sorting

//CurrStep - step for current itteration

Begin

//Determinating amount of steps

AmOfSteps := round(ln(AAmountOfEl)/ln(3) - 1);

//Finding max step

CurrStep := 1;

for i := 1 to AmOfSteps do

CurrStep := CurrStep \* 3 + 1;

//Sorting array

while CurrStep >= 1 do

begin

//Going trough array

i := 1;

while i + CurrStep <= AAmountOfEl do

begin

inc(ARealAmOfIters);

//Going back to beggining and swapping while

//needed

if AArrToSort[i] > AArrToSort[i + CurrStep] then

begin

//j is for saving value of i

j := i;

repeat

Swap(AArrToSort[j], AArrToSort[j + CurrStep]);

j := j - CurrStep;

inc(ARealAmOfIters);

until (AArrToSort[j] <= AArrToSort[j + CurrStep]) or (j < 1);

end;

inc(i);

end;

//Changing step

CurrStep := CurrStep div 3;

end;

End;

//Proc for selection sort

Procedure SelectionSort(ArrToSort: TArray; const AAmountOfEl: integer; var ARealAmOfIters: integer);

Var

i, j, СurrMin: integer;

//i, j - iterators

//currMin - max for current iteration

Begin

//Starting from first element

i := 1;

//Sorting array

while i < AAmountOfEl do

begin

//Assigning starting value for min

СurrMin := ArrToSort[i];

//Finding current min

for j := i + 1 to AAmountOfEl do

begin

inc(ARealAmOfIters);

if ArrToSort[j] < СurrMin then

СurrMin := ArrToSort[j];

end;

//Swapping with current first element if less value

//was found

inc(ARealAmOfIters);

if СurrMin <> ArrToSort[i] then

Swap(СurrMin, ArrToSort[i]);

inc(i);

end;

End;

Function ShellTheorCalc(const AAmOfElements: Integer): Integer;

begin

Result := AAmOfElements \* Round(ln(AAmOfElements) / ln(2));

end;

Function SelectionTheorCalc(const AAmOfElements:

Integer): Integer;

begin

Result := round((sqr(AAmOfElements) - AAmOfElements) / 2);

end;

end.