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: Анализ алгоритмов сортировки

Выполнил

студент: гр.551004 Довыдёнок М.А.

Проверил: Фадеева Е.П.

Минск 2016

Содержание

[1 Постановка задачи 3](#_Toc446161517)

[2 Описание алгоритмов 4](#_Toc446161518)

[3 Основные расчетные формулы 6](#_Toc446161519)

[4 Структура данных 7](#_Toc446161520)

[4.1 Структура данных основной программы 7](#_Toc446161521)

[4.2 Структура данных подпрограмм 7](#_Toc446161522)

[5 Схема алгоритма решения задачи по ГОСТ 19.701-90 9](#_Toc446161523)

[5.1 Схема основного алгоритма 9](#_Toc446161524)

[5.2 Схема алгоритма getArray(len, arType, result) 9](#_Toc446161525)

[5.3 Схема алгоритма swap(ar, i, j) 10](#_Toc446161526)

[5.4 Схема алгоритма outputTable(res, len) 10](#_Toc446161527)

[5.5 Схема алгоритма qSort(ar, result) 11](#_Toc446161528)

[5.6 Схема алгоритма sSort(ar, result) 12](#_Toc446161529)

[5.7 Схема алгоритма calculateChecks(len, result) 13](#_Toc446161530)

[6 Результаты расчетов и тестирование программы 14](#_Toc446161531)

[Приложение А 15](#_Toc446161532)

# Постановка задачи

Провести сравнительный анализ алгоритмов быстрой и шейкерной сортировок.

Размерности анализируемый массивов: 100, 250, 500, 1000, 2000, 3000

Типы анализируемых массивов: случайный, сортированный, перевернутый

Критерий анализа – количество сравнений

Разработать структуру данных для хранения результатов расчёта

Результаты свести в таблицу:

Таблица 1 – Пример результата

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Массив | | Количество сравнений | | | |
| Быстрая сортировка | | Шейкерная сортировка | |
| Размерность | Тип | Практич. | Теоретич. | Практич. | Теоретич. |
| 100 | Сорт. | … | … | … | … |
| Обрат. | … | … | … | … |
| Случ. | … | … | … | … |
| … | … | … | … | … | … |
| … | … | … | … | … |
| … | … | … | … | … |

# Описание алгоритмов

Таблица 2 – Описание основной программы

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Наименование  алгоритма | Назначение  алгоритма | Формальные  параметры | Рекомендуемый тип |
|  | Основной алгоритм | Получает массив res из алгоритма calculateChecks, вызывает outputTable для этого массива |  |  |
|  | GetArray  (len, arType, result) | Заполняет массив result значениями.  В зависимости от arType:  1,2, .., len  len, len – 1, .., 1  Случайными числами | len - Integer, arType – перечислимый тип из 3 значений, result – array [0..len-1] of integer.  Возвращаемый параметр: result | Функция  Возвращаем параметр: result |
|  | Swap(ar, i, j) | Меняет местами I и J элементы массива ar | ar – array of integer, i, j - integer.  Возвращаемый параметр: ar | Процедура |
|  | QSort(ar, result) | Сортирует массив ar алгоритмом быстрой сортировки, записывает количество сравнений в result  Использует алгоритм swap | ar – array of integer, result - integer  Возвращаемый параметр: result | Функция.  Возвращаемый параметр: result |
|  | SSort(ar, result) | Сортирует массив ar алгоритмом шейкерной сортировки, записывает количество сравнений в result.  Использует алгоритм swap | ar – array of integer, result - integer  Возвращаемый параметр: result | Функция.  Возвращаемый параметр: result |
|  | CalculateChecks  (len, result) | Для различных значений из массива len получает массив алгоритмом getArray, сортирует этот массив алгоритмами qSort и sSort.  Записывает количество сравнений в result | len – array [1..6] of integer, result – array [1..6, 1..6] of integer  Возвращаемый параметр: result | Функция.  Возвращаемый параметр: result |
|  | OutputTable  (res, len) | Выводит результаты из массива res, значения длины берёт из массива len | res – array[1..6, 1..6] of integer, len – array [1..6] of integer | Процедура |

# Основные расчетные формулы

Теоретический анализ алгоритма быстрой сортировки

Лучший случай: O(n \* log2(n))

Средний случай: O(2 \* ln(2) \* n \* log2(n))

Худший случай: O(n \* n)

Теоретический анализ алгоритма шейкерной сортировки

Независимо от массива: O(n \* n)

# Структура данных

## Структура данных основной программы

Таблица 3 – Структура данных основной программы

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| res | array [1..6, 1..6] of Integer | Массив результатов анализа |
| arrayLength[[1]](#footnote-1) | array [1..6] of Integer | Массив, проверяемых длин |

## Структура данных подпрограмм

Таблица 4 – Структура данных алгоритма getArray(len, arType, result)

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| len | Integer | Длина заполняемого массива |
| arType | (SORT, BACK, RAND) | Способ заполнения массива |
| result | Array [0..len - 1] of Integer | Заполняемый массив |
| i | Integer | Счетчик цикла |

Таблица 5 – Структура данных алгоритма swap(ar, i, j)

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| ar | Array of Integer | Массив, в котором производится обмен |
| temp | Integer | Переменная для обмена |
| i, j | Integer | Позиции меняемых элементов |

Таблица 6 – Структура данных алгоритма qSort(ar, result)

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| ar | Array of integer | Сортируемый массив |
| result | Integer | Количество сравнений |
| steps | array of record  l, r: Integer;  end | Массив, для сохранения индексов подмассивов |
| i, j | Integer | Счетчики циклов |
| l, r | Integer | Границы проверяемого подмассива |
| x | Integer | Опорный элемент |

Таблица 7 – Структура данных алгоритма sSort(ar, result)

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| ar | Array of integer | Сортируемый массив |
| result | Integer | Количество сравнений |
| l, r | Integer | Границы неотсортированной части |
| j | Integer | Счетчик цикла |

Таблица 8 – Структура данных алгоритма calculateChecks(len, result)

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| len | Array [1..6] of Integer | Массив, проверяемых длин |
| result | Array [1..6,1..6] of Integer | Массив, заполняемый количеством сравнений |
| i, j | Integer | Счетчики циклов |
| ar | Array of integer | Сортируемый массив |

Таблица 9 – Структура данных алгоритма outputTable(res, len)

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| res | Array [1..6,1..6] of Integer | Массив, заполняемый количеством сравнений |
| len | Array [1..6] of Integer | Массив, проверяемых длин |
| i | Integer | Счетчик цикла |

# Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема основного алгоритма



Рисунок 1 – Схема основного алгоритма

## Схема алгоритма getArray(len, arType, result)



Рисунок 2 – Схема алгоритма getArray(len, arType, result)

## Схема алгоритма swap(ar, i, j)



Рисунок 3 – Схема алгоритма swap(ar, i, j)

## Схема алгоритма outputTable(res, len)



Рисунок 4 – Схема алгоритма outputTable(res, len)

## Схема алгоритма qSort(ar, result)



Рисунок 5 – Схема алгоритма qSort(ar, result)

## Схема алгоритма sSort(ar, result)



Рисунок 6 – Схема алгоритма sSort(ar, result)

## Схема алгоритма calculateChecks(len, result)



Рисунок 7 – Схема алгоритма calculateChecks(len, result)

# Результаты расчетов и тестирование программы

Ожидаемый результат:

Количество сравнений практическое приблизительно

равно количеству сравнений теоретическому

Полученный результат:

Рисунок 8 – Результат работы программы



Рисунок 8 – Результат работы программы

Вывод:

Рисунок 9 – Вывод

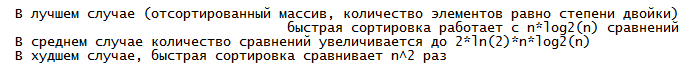


Рисунок 9 - Вывод

Приложение А

(обязательное)

Исходный код программы

program Lab10;

{$APPTYPE CONSOLE}

uses

Windows, Math;

type

TArray = array of integer;

TArrayType = (SORT, RAND, BACK);

TResult = array [1..6, 1..6] of Integer;

TArrayLength = array [1..6] of Integer;

const

arrayLength: TArrayLength =

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

var

res: TResult;

{ Возвращает массив, тип заполнения зависит от arType }

function getArray(len: Integer; arType: TArrayType):TArray;

var

i: Integer;

begin

randomize;

setLength(result, len);

case arType of

SORT:

for i := 0 to len - 1 do

result[i] := i;

RAND:

for i := 0 to len - 1 do

result[i] := random(10 \* len);

BACK:

for i := 0 to len - 1 do

result[i] := len - i;

end;

end;

{ Меняет местами два элемента в массиве }

procedure swap(var ar: TArray; i, j: Integer);

var

temp: Integer;

begin

temp := ar[i];

ar[i] := ar[j];

ar[j] := temp;

end;

{ Алгоритм Быстрой сортировки }

{ Возвращает количество сравнений }

function qSort(ar: TArray):Integer;

var

steps: array of record

l, r: Integer;

end;

i, j, l, r, x: Integer;

begin

result := 0;

setLength(steps, 1);

steps[0].l := 0;

steps[0].r := length(ar) - 1;

repeat

{ Определяем граница текущего подмассива }

l := steps[length(steps) - 1].l;

r := steps[length(steps) - 1].r;

setLength(steps, length(steps) - 1);

repeat

i := l;

j := r;

{ Сохраняем значение опорного элемента }

x := ar[(i + j) div 2];

while i <= j do

begin

inc(result, 2);

while ar[i] < x do

begin

inc(i);

inc(result);

end;

while ar[j] > x do

begin

dec(j);

inc(result);

end;

inc(result);

if i <= j then

begin

swap(ar, i, j);

inc(i);

dec(j);

end;

end;

inc(result);

if i < r then

begin

{ Запоминаем границы подмассива }

setLength(steps, length(steps) + 1);

steps[length(steps) - 1].l := i;

steps[length(steps) - 1].r := r;

end;

r := j;

inc(result);

until r <= l;

inc(result);

until length(steps) = 0;

end;

{ Алгоритм Шейкерной сортировки }

{ Возвращает количество сравнений }

function sSort(ar: TArray):Integer;

var

l, r, j: Integer;

begin

result := 0;

l := 1;

r := length(ar) - 1;

repeat

for j := r downto l do

begin

inc(result, 2);

if ar[j - 1] > ar[j] then

swap(ar, j - 1, j);

end;

inc(l);

for j := l to r do

begin

inc(result, 2);

if ar[j - 1] > ar[j] then

swap(ar, j - 1, j);

end;

dec(r);

inc(result);

until l > r;

end;

{ Вызывает подпрограммы создания и сортировки массива }

{ Возвращает количество сравнений в формате массива }

function calculateChecks(len: TArrayLength):TResult;

var

i, j: Integer;

ar: TArray;

begin

for i := 1 to 6 do

begin

ar := getArray(len[i], SORT);

result[i][1] := qSort(ar);

result[i][4] := sSort(ar);

ar := getArray(len[i], BACK);

result[i][2] := qSort(ar);

result[i][5] := sSort(ar);

result[i][3] := 0;

result[i][6] := 0;

for j := 1 to 100 do

begin

ar := getArray(len[i], RAND);

inc(result[i][3], qSort(ar));

inc(result[i][6], sSort(ar));

end;

result[i][3] := result[i][3] div 100;

result[i][6] := result[i][6] div 100;

end;

end;

{ Выводит таблицу результатов }

procedure outputTable(res: TResult; len: TArrayLength);

var

i: Integer;

begin

writeln(' Массив | Быстрая сортировка |

Шейкерная сортировка ');

writeln('-----------------| Количество

сравнений ');

writeln(' Размер | Тип | Практика | Теория |

Практика | Теория ');

writeln('--------------------------------------------

-----------------------');

for i := 1 to 6 do

begin

writeln(' | Сорт. | ', res[i][1]:8, ' | Луч.',

len[i] \* log2(len[i]):8:0, ' | ',

res[i][4]:8, ' | ');

writeln(len[i]:7, ' | Обрат. | ', res[i][2]:8, ' | Ср.',

2 \* ln(2) \* len[i] \* log2(len[i]):9:0, ' | ',

res[i][5]:8, ' | ', sqr(len[i]):9);

writeln(' | Случ. | ', res[i][3]:8, ' | Худ.',

sqr(len[i]):8, ' | ', res[i][6]:8, ' | ');

writeln('---------------------------------

----------------------------------');

end;

writeln(' В лучшем случае (отсортированный массив,

количество элементов равно степени двойки) ');

writeln(' быстрая

сортировка работает с n\*log2(n) сравнений ');

writeln(' В среднем случае количество сравнений

увеличивается до 2\*ln(2)\*n\*log2(n)');

writeln(' В худшем случае, быстрая сортировка

сравнивает n^2 раз');

end;

begin

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

res := calculateChecks(arrayLength);

outputTable(res, arrayLength);

readln;

end.

1. Константа [↑](#footnote-ref-1)