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)

ОТЧЁТ

по лабораторной работе №2

Тема работы: Сравнение алгоритмов сортировки

Выполнил

студент: гр. 151004 Иванов И.И.

Проверил: Фадеева Е.П.

Минск 2022

содержание

[1 Постановка задачи 3](#_Toc97843896)

[2 Описание алгоритмов решения задачи 4](#_Toc97843897)

[3 Структура данных 5](#_Toc97843898)

[3.1 Структура данных программы 5](#_Toc97843899)

[3.2 Структура данных алгоритма Swap 5](#_Toc97843900)

[3.3 Структура данных алгоритма Fill 5](#_Toc97843901)

[3.4 Структура данных алгоритма BubbleSort 6](#_Toc97843902)

[3.5 Структура данных алгоритма HeapSort 6](#_Toc97843903)

[3.6 Структура данных алгоритма SiftDown 6](#_Toc97843904)

[4 Схема алгоритма решения задачи по ГОСТ 19.701-90 8](#_Toc97843905)

[4.1 Схема алгоритма решения задачи 8](#_Toc97843906)

[4.2 Схема алгоритма Swap 9](#_Toc97843907)

[4.3 Схема алгоритма Fill 10](#_Toc97843908)

[4.4 Схема алгоритма BubbleSort 11](#_Toc97843909)

[4.5 Схема алгоритма SiftDown 12](#_Toc97843910)

[4.1 Схема алгоритма HeapSort 13](#_Toc97843911)

[4.1 Схема алгоритма Generate 14](#_Toc97843912)

[5 Результаты расчетов 15](#_Toc97843913)

[Приложение А 16](#_Toc97843914)

[Приложение Б 21](#_Toc97843915)

# Постановка задачи

Провести сравнительный анализ пузырька с флажком и пирамидальной сортировки по количеству сравнений.

Проверять на массивах со 100, 250, 500, 1000, 2000, 3000 элеметами.

Проверять на массивах трёх типов: рандомные, отсортированные и отсортированные задом наперёд.

# Описание алгоритмов решения задачи

Таблица 1 – Описание алгоритмов решения задачи

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| №  п.п | Наименование алгоритма | Назначение алгоритма | Формальные параметры | Предполагаемый тип реализации |
| 1. | Основной алгоритм | Вызывает следующие подпрограммы:  Swap, Fill,  BubbleSort, SiftDown, HeapSort,  Generate |  |  |
| 3. | Swap(A, B) | Обменивает элементы A и B местами | A, B – получает защищённое значение от фактического параметра | Процедура |
| 4. | Fill(Arr, N, Opt) | Заполняет массив Arr размерности N согласно правилу Opt | M – получает защищённое значение от фактического параметра | Процедура |
| 5. | BubbleSort (Arr, N, Comp) | Сортирует массив Arr размерности N, сохраняет число сравнений в Comp | Arr, N, Comp - получает защищённое значение от фактического параметра | Процедура |
| 6. | SiftDown (Arr, Node, N, Comp) | Сортирует, начиная с индекса Node, массив Arr размерности N, сохраняет число сравнений в Comp | Arr, N, Comp - получает защищённое значение от фактического параметра  Node – получает адрес от фактического параметра | Процедура |
| 7. | HeapSort(Arr, N, Comp) | Сортирует массив Arr размерности N, сохраняет число сравнений в Comp | Arr, N, Comp - получает защищённое значение от фактического параметра | Процедура |
| 8. | Generate(N, Opt) | Выводит данные о массиве размерности N, заполненном согласно правилу Opt | N, Opt - получает защищённое значение от фактического параметра | Процедура |

# Структура данных

## Структура данных программы

Таблица 2 – Структура данных

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| I, J | Integer | Параметры цикла |
| Nums | Array[1..6] Of Integer | Массив из размерностей массивов |

## Структура данных алгоритма Swap

Таблица 3 – Структура данных алгоритма Swap(A, B)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| A | Integer | Обмениваемое число | Формальный |
| B | Integer | Обмениваемое число | Формальный |
| T | Integer | Сохранённое число | Локальный |

## Структура данных алгоритма Fill

Таблица 4 – Структура данных алгоритма Fill(Arr, N, Opt)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Opt | Integer | Выбор условия, по которому заполняется массив | Формальный |
| I | Integer | Параметр цикла | Локальный |

## Структура данных алгоритма BubbleSort

Таблица 5 – Структура данных алгоритма BubbleSort(Arr, N, Comp)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Comp | Integer | Число сравнений | Формальный |
| I, J | Integer | Параметры цикла | Локальный |
| Sorted | Boolean | «Флажок» сортировки | Локальный |

## Структура данных алгоритма HeapSort

Таблица 6 – Структура данных алгоритма HeapSort (Arr, N, Comp)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| N | Integer | Количество элементов массива | Формальный |
| Comp | Integer | Число сравнений | Формальный |
| NodeCurr | Integer | Текущий индекс | Локальный |
| NodeLast | Integer | Сохранённый индекс | Локальный |

## Структура данных алгоритма SiftDown

Таблица 7 – Структура данных алгоритма SiftDown(Arr, Node, N, Comp)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| Arr | Array[1..3000] | Массив | Формальный |
| Node | Integer | Текущий индекс | Формальный |

Продолжение Таблицы 7

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| N | Integer | Количество элементов массива | Формальный |
| Comp | Integer | Число сравнений | Формальный |
| Root | Integer | Текущий индекс | Локальный |
| Child | Integer | Дочерний индекс | Локальный |
| Sifted | Boolean | Закончена ли сборка дерева | Локальный |

# Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма решения задачи

|  |
| --- |
|  |
| Рисунок 1 – Схема алгоритма решения задачи |

## Схема алгоритма Swap

|  |
| --- |
|  |
| Рисунок 2 – Схема алгоритма Swap |

## Схема алгоритма Fill

|  |
| --- |
|  |
| Рисунок 3 – Схема алгоритма Fill |

## Схема алгоритма BubbleSort

|  |
| --- |
|  |
| Рисунок 4 – Схема алгоритма BubbleSort |

## Схема алгоритма SiftDown

|  |
| --- |
|  |
| Рисунок 5 – Схема алгоритма SiftDown |

## Схема алгоритма HeapSort

|  |
| --- |
|  |
| Рисунок 6 – Схема алгоритма HeapSort |

## Схема алгоритма Generate

|  |
| --- |
|  |
| Рисунок 7 – Схема алгоритма Generate |

# Результаты расчетов

Вследствие результатов программы мы получаем следующие результаты:

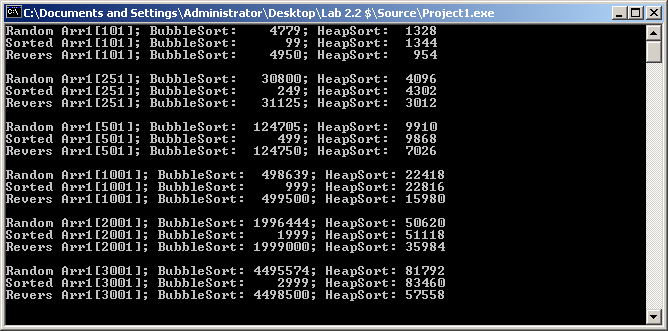


Рисунок 6 – Результаты расчетов

Приложение А

(обязательное)

Исходный код программы (постановка 1)

Program Lab2;

{Compare sorts}

//Use app

{$APPTYPE CONSOLE}

//Declare modules

Uses

SysUtils;

//Declare types

Type

TArr = Array[1..3000] Of Integer;

//TArr - our array

//Declare Vars

Var

I, J: Integer;

Nums: Array[1..6] Of Integer = (100, 250, 500, 1000, 2000, 3000);

//I,J - loop params

//Nums - array sizes

//Swaps 2 elements

//A, B - elements

Procedure Swap(Var A, B: Integer);

Var

T: Integer;

//T - temp

Begin

T:= A;

A:= B;

B:= T;

End;

//Fills array with elements

//Arr - array, N - array size, Opt - filler type

Procedure Fill(Var Arr: TArr; Const N, Opt: Integer);

Var

I: Integer;

//I - select type of filling

Begin

Randomize;

For I:= Low(Arr) To N Do

Begin

Case Opt Of

1:

Arr[I]:= Random(N);

2:

Arr[I]:= I;

3:

Arr[I]:= N - I;

End;

End;

End;

//Sorts an array and calculates the number of comparisons

//Arr - array, N - array size, Comp - comparisons

Procedure BubbleSort(Var Arr: TArr; Const N: Integer; Var Comp: Integer);

Var

I, J: Integer;

Sorted: Boolean;

//I, J - loop params

//Sorted - condition to exit

Begin

Sorted:= False;

I:= 1;

While (I <= N - 1) And Not Sorted Do

Begin

Sorted:= True;

For J:= 1 To N - I Do

Begin

If Arr[J] > Arr[J + 1] Then

Begin

Sorted:= False;

Swap(Arr[J], Arr[J + 1]);

End;

Inc(Comp);

End;

Inc(I);

End;

End;

//Sorts an array and calculates the number of comparisons

//Arr - array, Node - current index, N - array size, Comp - comparisons

Procedure SiftDown(Var Arr: TArr; Node: Integer; Const N: Integer; Var Comp: Integer);

Var

Root, Child: Integer;

Sifted: Boolean;

//Root, Child - indexes

//Sifted - condition to exit

Begin

Root:= Node;

Sifted:= False;

While (Not Sifted) And (Root \* 2 - Node + 1 <= N) Do

Begin

Comp:= Comp + 2;

Child:= Root \* 2 - Node + 1;

If (Child + 1 <= N) And (Arr[Child] < Arr[Child + 1]) Then

Inc(Child);

If Arr[Root] < Arr[Child] Then

Begin

Swap(Arr[Root], Arr[Child]);

Root:= Child;

End

Else

Sifted:= True;

End;

End;

//Sorts an array and calculates the number of comparisons

//Arr - array, N - array size, Comp - comparisons

Procedure HeapSort(Var Arr: TArr; Const N: Integer; Var Comp: Integer);

Var

NodeLast, NodeCurr: Integer;

//NodeLast, NodeCurr - indexes

//Comps - counter

Begin

NodeCurr:= N Div 2 - 1;

While NodeCurr >= Low(Arr) Do

Begin

SiftDown(Arr, NodeCurr, N, Comp);

Dec(NodeCurr);

End;

NodeLast:= N;

While NodeLast > Low(Arr) Do

Begin

Swap(Arr[Low(Arr)], Arr[NodeLast]);

Dec(NodeLast);

SiftDown(Arr, Low(Arr), NodeLast, Comp);

End;

End;

//Generates log info

//N - array size, I - filler type

Procedure Generate(Const N, Opt: Integer);

Var

Compare: Integer;

Arr1, Arr2: TArr;

Str: String;

//Compare - counter

//Arr1, Arr2 - similar arrays for the test

//Str - displaying

Begin

//Options for all cases

Case Opt Of

1:

Str:= 'Random';

2:

Str:= 'Sorted';

3:

Str:= 'Revers';

End;

Fill(Arr1, N, Opt);

//Show arr size

Write(Str, ' Arr1[', N, ']; ');

//Copy array

Arr2:= Arr1;

//Enzero to show true value

Compare:= 0;

BubbleSort(Arr1, N, Compare);

Write('BubbleSort: ', Compare: 7, '; ');

//Enzero to show true value

Compare:= 0;

HeapSort(Arr2, N, Compare);

Write('HeapSort: ', Compare: 5);

WriteLn;

End;

Begin

For I:= 1 To 6 Do

Begin

For J:= 1 To 3 Do

Generate(Nums[I], J);

WriteLn;

End;

ReadLn;

End.

Приложение Б

(обязательное)

Тестовые наборы

Формула среднего числа сравнений для BubbleSort: C = (N2-N)/2

Формула среднего числа сравнений для HeapSort: C = 2 \* N \* log2N

Результаты на практике подтверждают результаты в теории. Если подставить вместо N количество элементов в массиве, то результат, рассчитанный по формуле, будет приблизительно равен практическому результату.

Поскольку в постановке задачи применялся не простой BubbleSort, а BubbleSort с флажком, то отсортированный массив будет проверен всего 1 раз, а количество сравнений будет высчитываться не по обычной формуле, а по формуле C = N – 1.

|  |  |
| --- | --- |
|  |  |