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](#_Toc96877147)

[2 Описание алгоритмов решения задачи 4](#_Toc96877148)

[3 Структура данных 5](#_Toc96877149)

[3.1 Структура данных программы 5](#_Toc96877150)

[3.2 Структура данных алгоритма MulM 5](#_Toc96877151)

[3.3 Структура данных алгоритма Disp 5](#_Toc96877152)

[3.4 Структура данных алгоритма SumM 6](#_Toc96877153)

[3.5 Структура данных алгоритма MulN 6](#_Toc96877154)

[4 Схема алгоритма решения задачи по ГОСТ 19.701-90 8](#_Toc96877155)

[4.1 Схема алгоритма решения задачи 8](#_Toc96877156)

[4.2 Схема алгоритма Disp 9](#_Toc96877157)

[4.3 Схема алгоритма SumM 10](#_Toc96877158)

[4.4 Схема алгоритма MulN 11](#_Toc96877159)

[4.5 Схема алгоритма MulM 12](#_Toc96877160)

[5 Результаты расчетов 13](#_Toc96877161)

[Приложение А 14](#_Toc96877162)

[Приложение Б 17](#_Toc96877163)

# 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:  Disp, MulN,  MulM, SumM |  |  |
| 3. | MulM(M1,  M2) | Умножает матрицы M1 и M2 с последующим возвращением результата | M1, M2 – получает защищённое значение от фактического параметра | Функция |
| 4. | Disp(M) | Выводит на экран матрицу M с последующим возвращением результата | M – получает защищённое значение от фактического параметра | Функция |
| 5. | SumM(M1,  M2, IsPlus) | Вычисляет сумму (разность, если IsPlus = false) двух матриц M1 и M2 с последующим возвращением результата | M1, M2, IsPlus – получает защищённое значение от фактического параметра | Функция |
| 6. | MulN(M, N) | Умножает матрицу Matr на число Num с последующим возвращением результата | M, N - получает защищённое значение от фактического параметра | Функция |

# Структура данных

## Структура данных программы

Таблица 2 – Структура данных

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| А | TMatrix | Основная вводимая матрица |
| B | TMatrix | Основная вводимая матрица |
| C1, C2, C3, C4, C5, C6, C7 | TMatrix | Матрицы для промежуточных значений |

## Структура данных алгоритма MulM

Таблица 3 – Структура данных алгоритма MulM(const M1:TMatrix; const M2:TMatrix): TMatrix

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| M1 | TMatrix | Матрица | Формальный |
| M2 | TMatrix | Матрица | Формальный |
| I | Integer | Параметр цикла. Значение строки матрицы | Локальный |
| J | Integer | Параметр цикла. Значение столбца матрицы | Локальный |

## Структура данных алгоритма Disp

Таблица 4 – Структура данных алгоритма Disp(const M:TMatrix)

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| M | TMatrix | Матрица | Формальный |
| I | Integer | Параметр цикла. Значение строки матрицы | Локальный |

Продолжение Таблицы 4

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| J | Integer | Параметр цикла. Значение столбца матрицы | Локальный |

## Структура данных алгоритма SumM

Таблица 5 – Структура данных алгоритма SumM(const M1:TMatrix; const M2:TMatrix; const IsPlus: Boolean): TMatrix

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| M1 | TMatrix | Матрица | Формальный |
| M2 | TMatrix | Матрица | Формальный |
| IsPlus | Boolean | Выбор, что делать: вычитание или сложение | Формальный |
| I | Integer | Параметр цикла. Значение строки матрицы | Локальный |
| J | Integer | Параметр цикла. Значение столбца матрицы | Локальный |

## Структура данных алгоритма MulN

Таблица 6 – Структура данных алгоритма MulN (const M:TMatrix; const N:Real): TMatrix

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| N | Real | Число, на которое умножается матрица | Формальный |
| M | TMatrix | Матрица | Формальный |
| I | Integer | Параметр цикла. Значение строки матрицы | Локальный |

Продолжение Таблицы 6

|  |  |  |  |
| --- | --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение | Тип параметра |
| J | Integer | Параметр цикла. Значение столбца матрицы | Локальный |

# Схема алгоритма решения задачи по ГОСТ 19.701-90

## Схема алгоритма решения задачи

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

## Схема алгоритма Disp

|  |
| --- |
|  |
| Рисунок 2 – Схема алгоритма Disp |

## Схема алгоритма SumM

|  |
| --- |
|  |
| Рисунок 3 – Схема алгоритма SumM |

## Схема алгоритма MulN

|  |
| --- |
|  |
| Рисунок 4 – Схема алгоритма MulN |

## Схема алгоритма MulM

|  |
| --- |
|  |
| Рисунок 5 – Схема алгоритма MulM |

# Результаты расчетов

Вследствие результатов программы мы получаем следующие результаты:

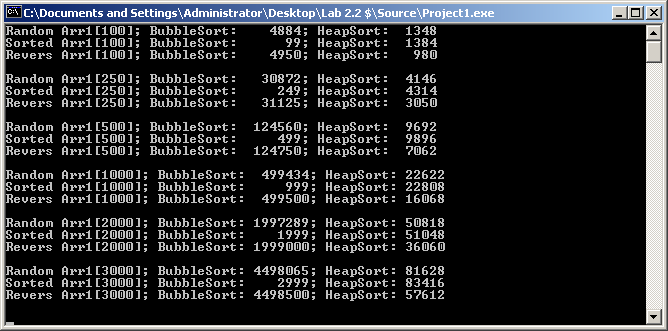


Рисунок 6 – Результаты расчетов

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[0..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

Procedure Swap(Var A, B: Integer);

Var

T: Integer;

//T - temp

Begin

T:= A;

A:= B;

B:= T;

End;

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

Var

I: Integer;

//I - select type of filling

Begin

Randomize;

For I:= 1 To N Do

Begin

Case Opt Of

1:

Arr[I]:= Random(N);

2:

Arr[I]:= I;

3:

Arr[I]:= N - I;

End;

End;

End;

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;

Procedure SiftDown(Var Arr: TArr; Node, NodeLast: 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 <= NodeLast) Do

Begin

Comp:= Comp + 2;

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

If (Child + 1 <= NodeLast) 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;

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

Var

NodeLast, NodeCurr, Comps: Integer;

//NodeLast, NodeCurr - indexes

//Comps - counter

Begin

NodeCurr:= Count Div 2 - 1;

While NodeCurr >= 0 Do

Begin

Comps:= 0;

SiftDown(Arr, NodeCurr, Count, Comps);

Comp:= Comp + Comps;

Dec(NodeCurr);

End;

NodeLast:= Count;

While NodeLast > 0 Do

Begin

Swap(Arr[0], Arr[NodeLast]);

Comps:= 0;

Dec(NodeLast);

SiftDown(Arr, 0, NodeLast, Comps);

Comp:= Comp + Comps;

End;

End;

Procedure Generate(Const N, Sort: 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 Sort Of

1:

Begin

Str:= 'Random';

Fill(Arr1, N, Sort);

End;

2:

Begin

Str:= 'Sorted';

Fill(Arr1, N, Sort);

End;

3:

Begin

Str:= 'Revers';

Fill(Arr1, N, Sort);

End;

End;

//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.

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