|  |  |
| --- | --- |
| Министерство образования Республики Беларусь | |
| Учреждение образования | |
| БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ | |
| ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ | |
|  | |
|  | |
| Факультет компьютерных систем и сетей | |
| Кафедра программного обеспечения информационных технологий | |
| Дисциплина: **Название дисциплины (АББРЕВИАТУРА)** | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
| **ОТЧЁТ** | |
| по лабораторной работе № **X** | |
|  | |
| Тема работы: **Название темы** | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
|  | |
| Выполнил: | Фамилия И.О. |
|  | гр. **XXXXXX** |
|  | Вариант **X** |
|  |  |
| Проверил: | Фамилия И.О. |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
| Минск **202X** | |

содержание

[1 Постановка задачи 3](#_Toc98952052)

[2 Методика решения задачи 4](#_Toc98952053)

[3 Описание алгоритмов решения задачи 6](#_Toc98952054)

[4 Структура данных 7](#_Toc98952055)

[4.1 Структура данных программы 7](#_Toc98952056)

[4.2 Структура данных алгоритма Swap 7](#_Toc98952057)

[4.3 Структура данных алгоритма Fill 7](#_Toc98952058)

[4.4 Структура данных алгоритма BubbleSort 8](#_Toc98952059)

[4.5 Структура данных алгоритма HeapSort 8](#_Toc98952060)

[4.6 Структура данных алгоритма SiftDown 9](#_Toc98952061)

[5 Схема алгоритма решения задачи по ГОСТ 19.701-90 10](#_Toc98952062)

[5.1 Схема алгоритма решения задачи 10](#_Toc98952063)

[5.2 Схема алгоритма Swap 11](#_Toc98952064)

[5.3 Схема алгоритма Fill 12](#_Toc98952065)

[5.4 Схема алгоритма BubbleSort 13](#_Toc98952066)

[5.5 Схема алгоритма HeapSort 14](#_Toc98952067)

[5.6 Схема алгоритма SiftDown 15](#_Toc98952068)

[6 Результаты расчетов 16](#_Toc98952069)

[Приложение А 17](#_Toc98952070)

[Приложение Б 21](#_Toc98952071)

# Постановка задачи

Провести сравнительный анализ по числу сравнений следующих видов сортировки:

1. Пузырёк с флажком;
2. Пирамидальная сортировка.

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

Типы массивов:

1. Случайный;
2. Отсортированный;
3. Перевернутый.

# Методика решения задачи

## Подготовка массивов

Для каждой размерности N генерируется массив по определённым правилам.

Сразу отсортированный масив создаётся посредством присваивания его элементам значений, равных индексам элементов:

A[I]:=I

Отсортированный в другую сторону масив создаётся ровно наоборот:

A[I]:=N-I

Случайный массив генерируется посредством присваивания его элементам случайных значений в диапазоне от 1 до верхней границы массива:

[I]:=Random(N)

## Подсчёт сравнений

После создания массивов начинается сортировка с подсчётом значений. Чтобы подсчёт проходил в равных условиях, массив предварительно копируется, а затем задействуется во втором виде сортировки.

Каждое сравнение увеличивает число сравнений на 1; если же сравнение сложное, т.е. содержащее логические операции or, and и так далее, то количество сравнений наращивается столько раз, сколько частей содержится в сравнении.

## Сортировки

Пузырёк с флажком является улучшенной сортировкой обменами. Суть заключается в том, что на каждой итерации осуществляется проверка, была ли хоть одна перестановка в процессе итерации. Если перестановок не было, сортировка прекращается. Таким образом, отсортированный заранее массив не будет сортироваться как положено, а будет проверен целиком всего один раз.

Пирамидальная сортировка является улучшенной сортировкой выбором. В сортировке выбором больше всего времени уходит на поиск максимального элемента в неотсортированной части. Пирамидальная сортировка лишена этого недостатка, поскольку на каждой итерации она сохраняет больше информации, формируя бинарную кучу. Бинарная куча – это граф-дерево, каждый узел которого может иметь не более двух потомков.

В нашем случае это не бинарное дерево в прямом смысле, а обычный массив, элементы которого мысленно имеют свою роль: для каждого элемента i есть элементы 2i+1 и 2i+2 , принятые за левый и правый дочерний.

Для вычисления теоретического числа сравнений при использовании алгоритма сортировки пузырьком с флажком используется следующая формула:

*Cmax = (N2-N)/2; Cmin = N-1*

Максимальное количество сравнений будет произведено в обратном массиве. Минимальное – в отсортированном. Случайно сгенерированные массивы по количеству сравнений попадут в диапазон между максимальным и минимальным значением, зачастую ближе к максимальному, нежели к минимальному.

Для вычисления теоретического числа сравнений при использовании алгоритма пирамидальной сортировки используется следующая формула:

*Cср = 2 \* N \* log2N*

Максимальное количество сравнений будет произведено в отсортированном массиве. Минимальное – в обратном. В среднем количество будет стремиться к теоретическому, но точное количество сравнений теоретически установить невозможно.

# 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 |  |  |
| 3. | Swap  (A, B) | Обменивает элементы A и B местами | A, B – получает защищённый адрес от фактического параметра | Процедура |
| 4. | Fill  (Arr, N, Opt) | Заполняет массив Arr размерности N согласно правилу Opt | M – получает защищённый адрес от фактического параметра | Процедура |
| 5. | BubbleSort (Arr, N, Comp) | Сортирует обменами с флажком массив Arr размерности N, сохраняет число сравнений в Comp | Arr, N - получает защищённый адрес от фактического параметра  Comp – получает значение от фактического параметра | Процедура |
| 6. | SiftDown (Arr, Node, N, Comp) | Обменивает дочерний и родительский элементы, начиная с индекса Node, массив Arr размерности N, сохраняет число сравнений в Comp | Arr, N - получает защищённый адрес от фактического параметра  Node – получает адрес от фактического параметра  Comp – получает значение от фактического параметра | Процедура |
| 7. | HeapSort(Arr, N, Comp) | Пирамидально сортирует массив Arr размерности N, сохраняет число сравнений в Comp | Arr, N - получает защищённый адрес от фактического параметра  Comp – получает значение от фактического параметра | Процедура |

# Структура данных

## Структура данных программы

Таблица 2 – Структура данных

|  |  |  |
| --- | --- | --- |
| Элементы данных | Рекомендуемый тип | Назначение |
| I, J | Integer | Параметры цикла |
| Comp | Integer | Число сравнений |
| Arr1, Arr2 | Array[1..3000] Of Integer | Массивы для сортировки |
| N | Array[1..6] Of Integer | Массив из размерностей массивов |
| S | Array[1..3] Of String | Массив из имён 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 |
| 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 – 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 HeapSort

|  |
| --- |
|  |
| Рисунок 5 – Схема алгоритма HeapSort |

## Схема алгоритма SiftDown

|  |
| --- |
|  |
| Рисунок 6 – Схема алгоритма SiftDown |

# Результаты расчетов

Вследствие результатов программы мы получаем следующие результаты:

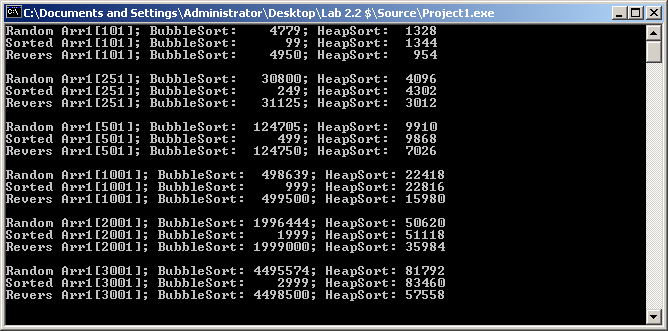


Рисунок 7 – Результаты расчетов

Приложение А

(обязательное)

Исходный код программы (постановка 1)

Program Lab2;

{Comp 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;

Comp: Integer;

Arr1, Arr2: TArr;

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

S: Array[1..3] Of String = ('Random', 'Sorted', 'Revers');

//I,J - loop params

//N - array sizes

//Comp - quantity of comparisons

//Arr1, Arr2 - our arrays

//N - num of elements

//S - name of array

//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;

Case Opt Of

1:

For I:= Low(Arr) To N Do

Arr[I]:= Random(N);

2:

For I:= Low(Arr) To N Do

Arr[I]:= I;

3:

For I:= Low(Arr) To N Do

Arr[I]:= N - I;

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;

Begin

For I:= 1 To 6 Do

Begin

For J:= 1 To 3 Do

Begin

Fill(Arr1, N[I], J);

//Show arr size

Write(S[J], ' Arr[', N[I], ']; ');

//Copy array

Arr2:= Arr1;

//Enzero to show true value

Comp:= 0;

BubbleSort(Arr1, N[I], Comp);

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

//Enzero to show true value

Comp:= 0;

HeapSort(Arr2, N[I], Comp);

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

WriteLn;

End;

WriteLn;

End;

ReadLn;

End.

Приложение Б

(обязательное)

Тестовые наборы

Результаты на практике подтверждают теоретические результаты, рассчитанные в Microsoft Excel.

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