Учебный курс "Введение в методы программирования"

Лабораторный практикум

Лабораторная работа 1: Упорядочивание (сортировка) массивов

1. Постановка учебно-практической задачи

Выполнение лабораторной работы направлено на освоение основных приемов использования массивов, методов доступа к элементам массивов, их реорганизации и модификации. В качестве практической проблемы, требующей решения, рассматривается известная задача *сортировки* (*упорядочивания*) массива в порядке возрастания (убывания) его элементов. При решении этой задачи требуется исходный массив, содержащий произвольные целые числа, преобразовать к виду, когда каждый элемент массива находится перед другим элементом этого массива, если его значение меньше (больше), чем значение сравниваемого элемента.

В данной лабораторной работе необходимо изучить ряд известных алгоритмов сортировки и создать комплекс программ, реализующий

- линейный поиск элемента в массиве;
- метод двоичного поиска элемента в упорядоченном массиве;
- метод сортировки выбором;
- метод сортировки пузырьком;
- метод сортировки включением;
- метод быстрой сортировки [+)] .

Разрабатываемый программный комплекс должен обеспечивать

- вывод на экран меню;
- ввод исходной информации;
- формирования массивов с большим числом элементов;
- выбор метода сортировки;
- сортировку массива;
- печать результата;
- замеры времени выполнения сортировок массива.

Демонстрация работоспособности разработанных программных средств должна обеспечивать два варианта контроля: контроль работоспособности каждого из методов и контроль временных характеристик всех реализованных методов.

2. Учебно-методические цели работы

Данная задача представляет собой упрощенный учебно-ознакомительный вариант широко распространенной задачи сортировки, которая применяется в широком классе реальных задач, и умение и знание алгоритмов сортировки и их особенностей является необходимой частью образования специалиста по программному обеспечению. Наряду с ознакомлением с проблемой упорядочивания массивов и методами ее решения, выполнение работы направлено на достижение следующих учебно-методических целей:

- приобретение навыков и приемов использования массивов, доступа к их элементам и преобразований массивов;
- практическое освоение принципов модульного программирования (использование процедур и функций);

[+)] Задача повышенной сложности, рассматривается как тема дополнительного задания

- освоение и изучение встроенных функций языка Zonnon и модуля System (применение генератора псевдослучайных чисел и замеры датчиков времени);
- получения навыков сравнительного анализа эффективности разных алгоритмов.

3. Рекомендации по выполнению работы

Программа должна обеспечить сортировку массивов размером произвольной длины до 10000 элементов и выводить для контроля:

- при небольшом количестве элементов (например, менее 20) - неупорядоченный массив и массив после сортировки для каждого из предложенных алгоритмов;

- при значительном объеме данных (более 20) выводить время сортировки одного и того же массива для всех четырех предложенных алгоритмов.

Содержимое массива рекомендуется формировать с помощью генератора псевдослучайных чисел, замеры времени производить средствами модуля System (Random, DateTime).

В ходе выполнения лабораторной работы предполагается реализация методов *линейного* и *двоичного поиска* элемента массива и разработка программ для четырех широко используемых *алгоритмов сортировки*:

- метод выбора;
- метод пузырька;
- метод включения;
- метод быстрой сортировки.

Для простоты изложения описания алгоритмов будут проводиться на примере задач сортировки массивов по возрастанию.

3.1. Поиск элемента массива на основе линейного просмотра.

Результатом работы алгоритма линейного поиска значения Val в массиве A являются индекс Pos и логическая переменная ResultOk, которая принимает значение TRUE, если такой элемент содержится в массиве A, и FALSE - в противном случае. Индекс Pos принимает значение, равное номеру искомого элемента, если такой найден, и значение, равное -1 - в противном случае.

Алгоритм линейного поиска

Шаг 1. Полагается Pos:=-1 и ResultOk:=FALSE, и значение переменной цикла J:=0.

Шаг 2. Если A[J]=Val, то переменным Pos и ResultOk присваиваются соответственно значения Pos:=J, ResultOk:=TRUE и алгоритм завершает работу. В противном случае значение переменной цикла увеличивается на единицу J:=J+1.

Шаг 3. Если J<Last, где Last - число элементов массива A, то выполняется Шаг 2, в противном случае - работа алгоритма завершена.

Конец алгоритма.

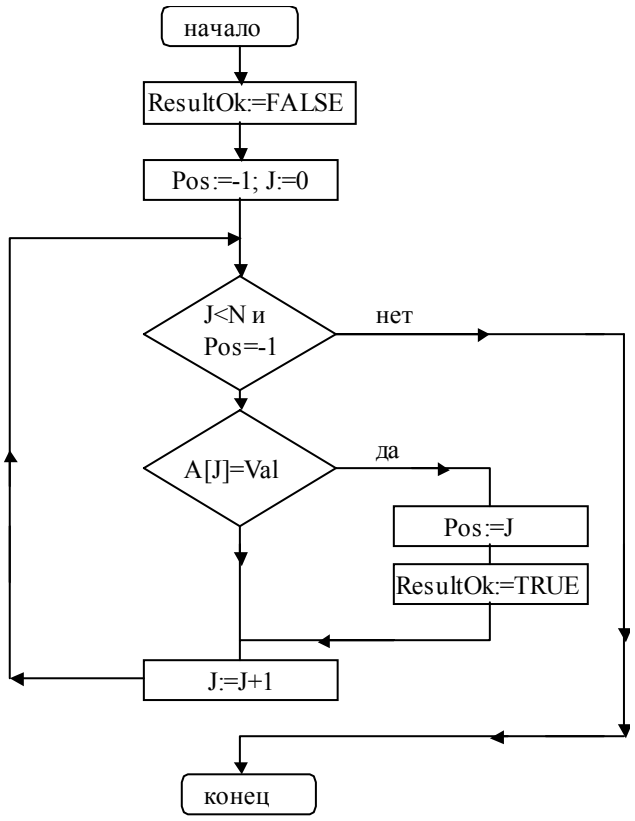

Рис 1.1. Блок-схема алгоритма линейного поиска

3.2. Метод двоичного поиска

Результатом работы алгоритма является индекс Pos, показывающий на место в упорядоченном массиве A с номерами элементов от First до Last, на которое необходимо поставить значение Val так, чтобы вновь образованный массив остался упорядоченным. Формируется в качестве результата и логическая переменная ResultOk, которая принимает значение TRUE, если искомый элемент содержится в массиве A, и - FALSE - в противном случае.

Алгоритм двоичного поиска

Шаг 1. Пока справедливо условие First<Last, выполняется Шаг 2.

Шаг 2. Вычисляется середина массива Middle:= (Last+First) div 2. Если Val равно A[Middle], то полагается First:=Middle и Last:=First , в противном случае проверяется условие - Val больше A[Middle]? Если это условие справедливо, то полагается First:=Middle+1, в противном случае полагается Last:=Middle-1. После чего управление передается на Шаг 1.

Шаг 3. Полагается Pos:=First.

Шаг 4. Проверка успеха поиска элемента Val в массиве. Полагается ResultOk:=FALSE. После чего проверяется, содержится ли элемент со значением Val в массиве, и при положительном ответе на этот вопрос переменной ResultOk присваивается значение TRUE.

Конец алгоритма.

3.3. Метод сортировки выбором

Исходный массив длиной N разбивается на две части: итог и остаток. Участок массива, называемый *итогом*, располагается с начала массива и должен быть упорядоченным, а участок массива, называемый *остатком*, располагается вплотную за итогом и содержит исходные числа не отсортированной части исходного массива. Пусть первый элемент остатка является J-ым элементом массива.

Алгоритм сортировки выбором

Шаг 1. Полагается J:=0, т.е. считается, что итоговый участок - пуст.

Шаг 2. В остатке массива ищется минимальный и меняется местом с первым элементом остатка (J-ым элементом массива). После чего значение J увеличивается на единицу, тем самым расширяя итоговый участок массива (отсортированную часть исходного массива).

Шаг 3. Если J < N-1, то повторяется Шаг 2. В противном случае - конец алгоритма, т.к. итог становится равным всему массиву.

Конец алгоритма.

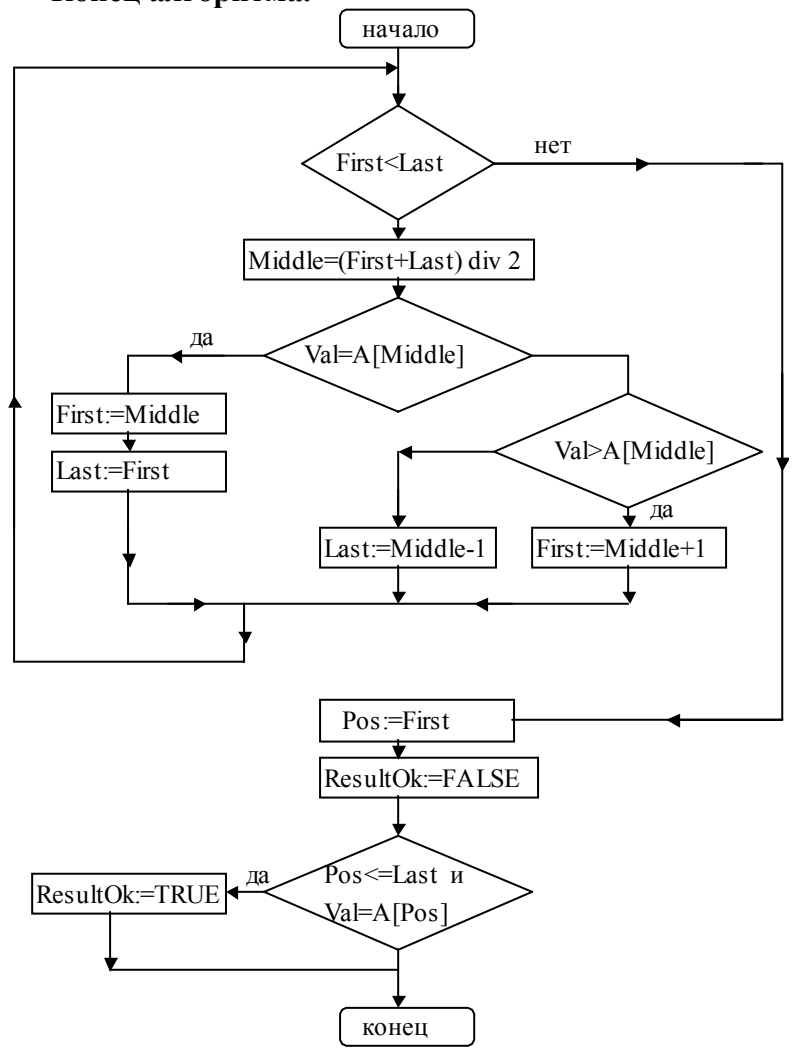

Рис 1.2. Блок-схема алгоритма двоичного поиска

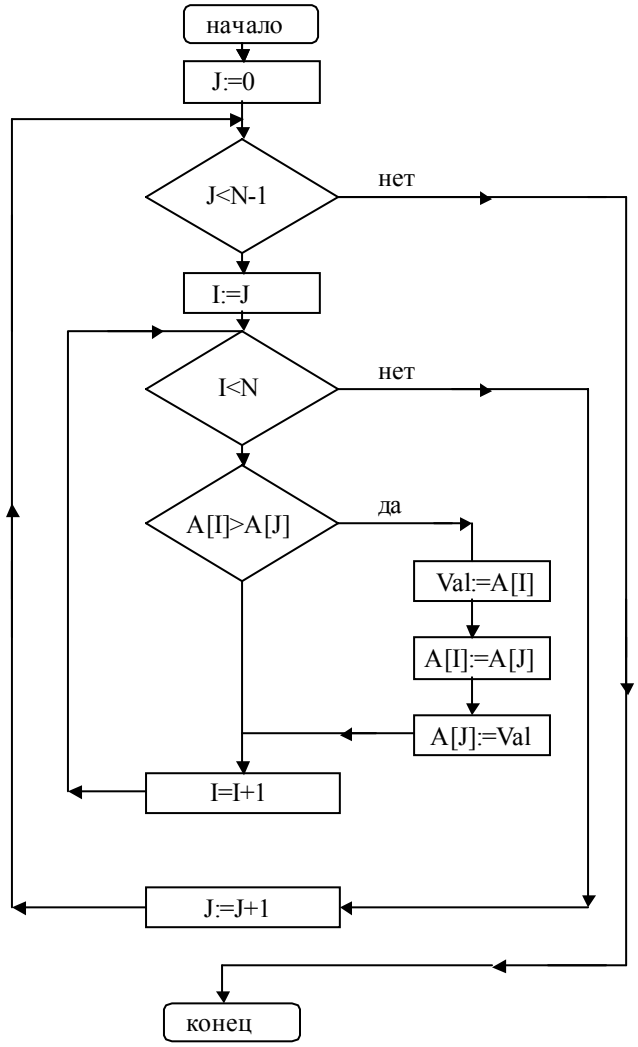

Рис 1.3. Блок-схема алгоритма сортировки методом выбора

3.4. Метод сортировки пузырьком

Аналогично, как и в методе выбора, исходный массив длиной N разбивается на две части: отсортированную (*итог*) и не отсортированную (*остаток*). Пусть первый элемент остатка будет J-ым элементом массива.

Алгоритм сортировки пузырьком

Шаг 1. Пусть J:=1 , т.е. итоговый участок состоит из одного элемента.

Шаг 2. Берется первый элемент остатка и перемещается на место в итоговый участок массива так, чтобы итог остался упорядоченным. Первый элемент остатка назовем *перемещаемым*. Перемещение выполняется путем сравнения перемещаемого элемента с предшествующим ему элементом. Если предшествующий элемент меньше сравниваемого элемента, то процесс перемещения закончен. В противном случае сравниваемые элементы переставляются и, если элемент не достиг начала массива, то повторяется Шаг 2.

Шаг 3. После того, как первый элемент остатка переместился в итоговый участок, увеличивается на единицу значение переменной J, тем самым увеличивая отсортированную часть массива. Если J<N, то управление передается на Шаг 2, в противном случае - работа алгоритма завершена.

Конец алгоритма.

3.5. Метод сортировки включением

Этот метод похож на метод пузырька. Происходит такое же разбиение массива на отсортированную и не отсортированную части, но перемещение первого элемента остатка на принадлежащее ему место в итоге делается не сравнением двух соседних элементов, а с помощью метода двоичного поиска, который удобно оформить в виде отдельной процедуры.

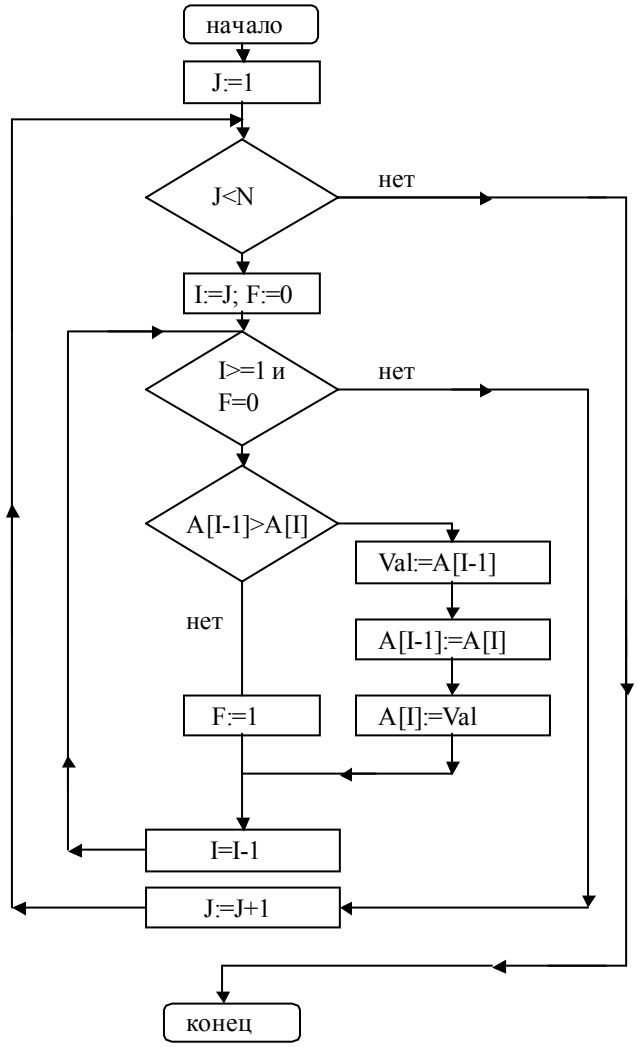

Рис 1.4. Блок-схема алгоритма сортировки методом пузырька

Алгоритм метода включения

Шаг 1. Пусть J=1 , т.е. итоговый участок состоит из одного элемента.

Шаг 2. Берется первый элемент остатка и перемещается в отсортированную часть массива так, чтобы итоговый участок остался упорядоченным. Делается это с помощью обращения к процедуре двоичного поиска, которая в качестве выходного параметра дает номер элемента массива, на месте которого должен бы находиться перемещаемый элемент. Если этот номер указывает на место в итоговом участке массива, то сдвигаются все элементы итогового участка массива, начиная с этого номера на одно место вправо, а перемещаемый элемент ставится на освободившееся место.

Шаг 3. После того, как первый элемент остатка переместился в итоговый участок, увеличивается на единицу значение переменной J, тем самым увеличивая отсортированную часть массива. Если J < N, то управление передается на Шаг 2, в противном случае - работа алгоритма завершена.

Конец алгоритма.

3.6. Метод быстрой сортировки

Исходным является массив A с номерами элементов от First до Last. В алгоритме используются еще два индекса массива, обозначенные как Index и ContrIndex. Первый из них всегда указывает на переставляемый элемент, а второй — на элемент, который сравнивается по значению с переставляемым. В процессе вычислений применяются переменная h (равная либо 1, либо -1) - шаг движения индексов навстречу друг другу, используемая для обозначения направления движения индекса ContrIndex, и логическая переменная Condition (равная либо TRUE, либо FALSE), используемая для изменения условия сравнения на противоположное при обратном движении индекса ContrIndex.

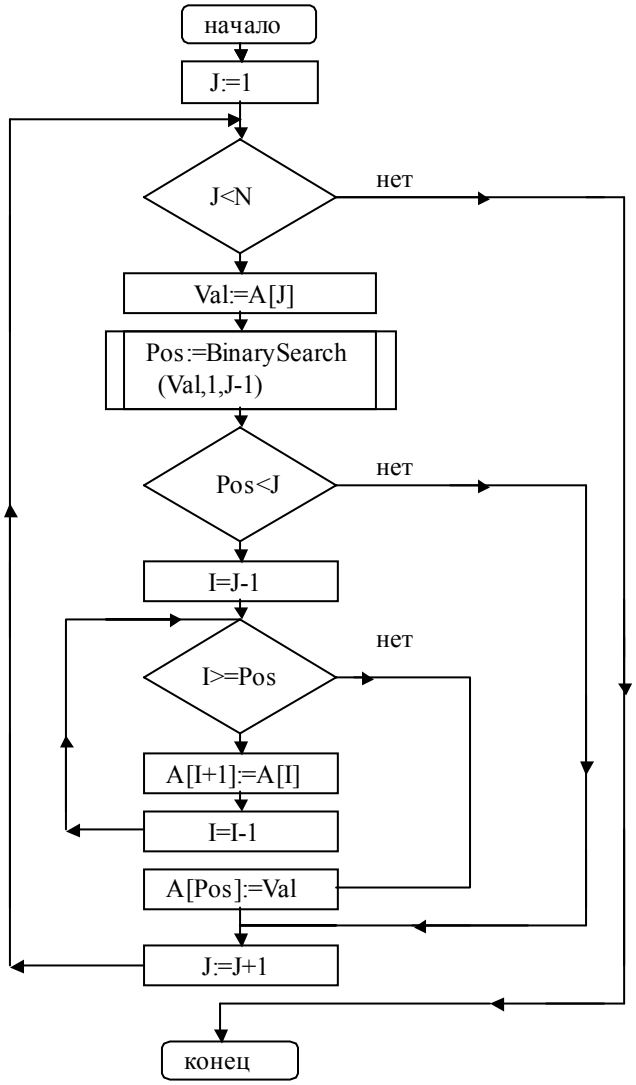

Рис 1.5. Блок-схема алгоритма сортировки методом включения

Алгоритм быстрой сортировки

Шаг 1. Если First >= Last, то происходит выход из алгоритма. В противном случае полагается h:=1, Condition:=TRUE, Index:=First, ContrIndex:=Last и делаются шаги: Шаг 2 - Шаг 3.

Шаг 2. Пока Index не равно ContrIndex, делаются шаги: Шаг 2a -Шаг 2b.

Шаг 2a. Если справедливо ((A[Index]>A[ContrIndex])=Condition), то переставляются как сами элементы, на которые указывают Index и ContrIndex, (Val:=A[Index], A[Index]:=A[ContrIndex], A[ContrIndex]:=Val), так и сами вспомогательные индексы массивов (Val:=Index, Index:=ContrIndex, ContrIndex:=Val). Затем меняется направление движения (h:= -h) и условие сравнения (Condition: = not Condition). В процессе таких перестановок слева от переставляемого элемента всегда будут находиться меньшие значения, а справа - большие значения.

Шаг 2b. Сдвигается вспомогательный индекс массива ContrIndex навстречу индексу Index , т.е. ContrIndex:= ContrIndex +h.

Шаг 3. Перед выполнением этого шага индексы Index = ContrIndex и элемент A[Index] находится на нужном месте. Т.е. исходный массив разбит на три части: часть массива до этого элемента, значения в котором меньше величины A[Index], часть массива после этого элемента с значениями большими значения A[Index] и сам этот элемент A[Index]. Поэтому для дальнейшего упорядочивания массива достаточно рекурсивно обратиться к алгоритму быстрой сортировки два раза: для первой и второй частей массива. Т.к. длина сортируемых участков массива уменьшается, то в итоге алгоритм конечен и после применения алгоритма массив будет полностью отсортирован.

Конец алгоритма.

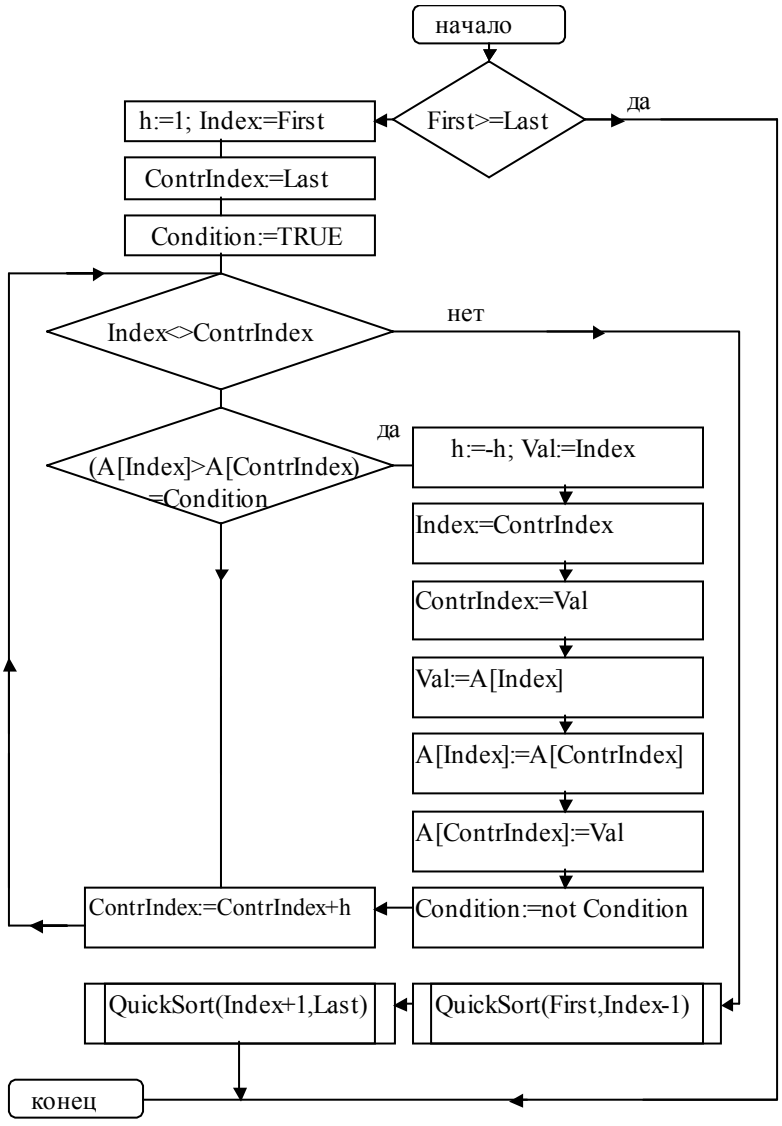

Рис 1.6. Блок-схема алгоритма быстрой сортировки

4. Рекомендации по организации программного комплекса

Разработку программы рекомендуется выполнить в виде программного комплекса, включающего:

- модули поиска;
- модули сортировки;
- модуль формирования массива;
- печать массива;
- модуль замера времени;
- программу - монитор, реализующую диалог с пользователем.

4.1. Состав разрабатываемых процедур и функций

Рекомендуется завести глобальные переменные и типы данных следующего вида:

```
const     Lmax= 10000, NLimitPrintMas=20;
type mas = array Lmax of integer;
var A:mas;
    CopyA:mas;
    uhr1,uhr2,min1,min2,sec1,sec2,ds1,ds2:integer{16};
    N:integer;
```

где

- A - исходный массив,
- CopyA -копия массива A, которая подвергается сортировке,
- N - длина массива A,
- NLimitPrintMas - константа, ограничивающая длину массива, выводимого на печать.

Остальные переменные используются для замеров времени перед началом и в конце работы каждого алгоритма сортировки с использованием функции DateTime.Now(); модуля System.DateTime, который должен быть подключен к основной программе.

Алгоритмы сортировок рекомендуется реализовать в виде процедур вида:

```
SelectSort(var A:mas; N:integer);
BubbleSort(var A:mas; N:integer);
InsertionSort(var A:mas; N:integer);
QuickSort(var A:mas; N:integer; First,Last: integer);
ScanSearch(var A:mas; N:integer; Val:integer;
 var Pos:integer; var ResultOk: boolean);
BinarySearch(var A:mas;   N:integer;   Val,First,Last:integer;
 var Pos:integer; var ResultOk:boolean)
```

В виде вспомогательных модулей необходимо разработать:

- процедуру вычисления времени счета

  `Period (var Min,Sec,Dsec : integer{16}),`

  которая находит временную разницу между двумя замерами времени

  `uhr1,min1,sec1,ds1,`

  `uhr2,min2,sec2,ds2,`
- процедуру печати

  `PrintRes(var A: mas; N:integer),`
- процедуру копирования исходного массива, которая должна вызываться каждый раз перед новой сортировкой

```
CopyMas(var A,B: mas; N:integer).
```

Для формирования исходного массива необходимо написать процедуру

```
FormMas(var A: mas; N:integer),
```

которая, используя функции генератора псевдослучайных чисел модуля DOS Randomize, Random, заполняет исходный массив длиной N.

Число N должен задавать пользователь. Причем, если это число меньше, чем NLimitPrintMas, то в процедуре печати результата PrintRes содержимое массива распечатывается, а временные замеры не выводятся на печать. В противном случае данная процедура выводит на экран только замеры времени последней сортировки с помощью процедуры SortTime. Приведенные рекомендации позволяют легко построить основную программу, обеспечивающую простейший интерфейс с пользователем и упростить алгоритм отладки, когда на месте процедур сортировок, находящихся на стадии разработки, временно могут находиться тексты программ без блоков реализации (*заглушки*).

4.2. Рекомендации по организации управляющей программы

Предлагается организовать следующий сценарий работы программного комплекса. При запуске программы на экране высвечивается некоторая информационная заставка программы с приглашением к работе. После нажатия любой клавиши на экране дисплея должно появиться меню, содержащее все основные действия программы:

- задание количества элементов формируемого массива;
- задание режима табулирования количества элементов в формируемом массиве от некоторого начального значения NBegin до значения NEnd с некоторым шагом NStep;
- формирование массива;
- сортировка массива методом выбора;
- сортировка массива методом пузырька;
- сортировка массива методом включения;
- быстрая сортировка;
- сортировка одного и того же сформированного массива всеми видами сортировок;
- поиск методом линейного просмотра первых 1000 чисел натурального ряда в отсортированном массиве с замером общего времени поиска;
- поиск методом двоичного поиска первых 1000 чисел натурального ряда в отсортированном массиве с замером общего времени поиска;
- печать результатов выполненной операции;
- печать времени выполнения последней операции;
- выход из программы.

При задании режима табулирования элементов формируемого массива пользователь задает три числа:

- NBegin - число элементов для первого формируемого массива, который затем обрабатывается;
- NEnd - число элементов для последнего формируемого массива;
- NStep - число элементов, на которое увеличивается длина следующего формируемого массива путем прибавления NStep к длине предыдущего сформированного массива.

В заданном режиме обрабатывается несколько сформированных массивов с разным числом элементов. Количество таких массивов равно ((NEnd-NBegin) div NStep)+1. При этом полагается, что все заданные параметры - положительные целые числа больше нуля и NEnd>NBegin. Сортировка и вывод результатов при выборе такого режима производится для всех созданных массивов. Для сортировки массивов используется

алгоритм сортировки, выбранный пользователем в меню. После выполнения сортировки для одного массива программа выводит временные характеристики вычислений, подсказку - сообщение о своей работе и ждет нажатия клавиши для продолжения вычислений для нового массива с увеличенным числом элементов на NStep элементов.

Кроме того, в качестве дополнительного задания повышенной сложности рекомендуется в качестве основных действий программы, включаемых в меню, предусмотреть возможность демонстрационных показов работы алгоритмов сортировки с пошаговым отображением работы каждого алгоритма после каждой итерации.

4.3. Рекомендации по стилю программирования

Оформление программы, наглядность текста программы зависит от выбранного стиля программирования разработчика. Стиль программирования является составной частью культуры программирования. Правилом для выбора хорошего стиля рекомендуется следовать девизу: “Ничего лишнего, но и ничего непонятного”. Основные рекомендации при выборе стиля могут состоять в следующем:

- **наименования** переменных, типов, процедур, функций и т.п. должны отображать их смысл и назначение;
- **комментарии** должны пояснять основные действия в программе и назначение используемых в программе структур данных;
- в программе должны применяться **пробелы, отступы, абзацы** для повышения ее читабельности.

Поясним приведенные рекомендации на примере написания программ сортировки и бинарного поиска.

Задача. Известен номер группы первокурсников и список студентов из этой группы. По заданной фамилии студента определить, занимается ли он в данной группе и под каким номером находится в списке группы.

Программа.

```
module Main;
const
  Lmax = 30;    (* максимальная численность группы *)

type
  TIndex = integer; (* тип индексов массива *)
  TName  = string; (*  тип данных для фамилии *)
  TList  = array Lmax of TName; (*  тип для списка фамилий *)
  byte   = integer{8};

var
  StudentList  :TList;    (*  исходный список фамилий  *)
  Name         :TName;    (*     интересующая фамилия  *)
  StudentNumb  :byte;     (* кол-во студентов в группе *)
  Numb         :integer;  (*    номер в списке группы  *)
  NumbGr       :integer;  (*             номер группы  *)
  i            :byte;
  ResultOk     :boolean;  (* результат поиска фамилии  *)


(* процедура сортировки массива  A методом выбора *)
procedure SelectSort  (var A:TList; N:byte);
var
  i, j  :TIndex;
```

```
  Val   :TName;
begin
  for j:=0  to   N - 2   do
    for i:=j+1  to  N - 1  do
      if A[i] < A[j]  then
        Val:=A[i]; A[i]:=A[j]; A[j]:=Val;
      end
    end
  end
end SelectSort;

(* бинарный поиск элемента в упорядоченном массиве *)
procedure BinarySearch (
  var A  : TList;       (*          исходный массив *)
  First  : TIndex;      (* начальный индекс поиска  *)
  Last   : TIndex;      (*  конечный индекс поиска  *)
  Val    : TName;       (*        искомое значение  *)
  var Pos: integer;
     (* индекс массива - место нахождения элемента *)
  var ResultOk : boolean         (* результат поиска *)
                      );
var
  Middle, TempLast  :TIndex;
begin
(* запоминание длины массива *)
TempLast:=Last;
(*
 замена в цикле отрезка неопределенности от First до Last после
разбиения его на две равные части на новый, равный одной из этих
частей, до тех пор, пока его длина не будет равна единице
*)
while  First<Last  do
    (* вычисление середины отрезка *)
    Middle:=(First + Last)  div  2;
    if  Val=A[Middle]  then
      (* элемент найден, выход из цикла *)
      First:=Middle; Last:=First;
      (*
        выбор в качестве отрезка неопределенности левой или правой
        части предыдущего отрезка
       *)
      elsif Val>A[Middle]
          then  First:=Middle + 1
          else  Last :=Middle - 1;
    end
  end;
Pos:=First;
(* проверка - содержится ли элемент с искомым значением в массиве *)
ResultOk:=false;
if  Pos<=TempLast  then
  if  Val=A[Pos]  then  ResultOk:=true; end
end
end BinarySearch;

(* основная программа *)
begin
(* ввод исходных данных *)
writeln('Введите номер группы '); readln(NumbGr);
```

```
  repeat
    writeln('Введите число студентов в группе ', NumbGr, '<=',Lmax);
    readln(StudentNumb);
  until  (StudentNumb>0) & (StudentNumb<=Lmax);
  writeln('Введите список студентов');
  for  i:=0  to  StudentNumb-1  do
    writeln('?'); readln(StudentList[i]);
    end;
  writeln('Введите интересующую Вас фамилию');
  readln(Name);

  (* обращение к процедуре сортировки *)
  SelectSort(StudentList, StudentNumb);

  (* обращение к процедуре двоичного поиска *)
  BinarySearch(StudentList,0,StudentNumb-1, Name, Numb, ResultOk);

  (* печать результата *)
  if  ResultOk
    then
      writeln('Студент ',Name,' значится в группе ', NumbGr,' под
номером ',Numb+1)
    else
      writeln(Name,' не числится в группе ', NumbGr);
  end;
  end Main.
```

В приведенной программе использованы алгоритмы, описанные в данной лабораторной работе. Основные правила, применявшиеся при написании программы, выглядят следующим образом:

- все ключевые слова алгоритмического языка Zonnon пишутся строчными буквами;
- наименования процедур и функций начинаются с прописной буквы;
- наименования переменных, констант и типов данных пользователя могут начинаться как с большой, так и с маленькой буквы;
- сложные имена, состоящие из нескольких частей (корней), пишутся маленькими буквами, но каждая часть имени начинается с заглавного символа;
- все вводимые пользователем переменные, константы, типы данных сопровождаются комментарием;
- для процедур и функций пишется комментарий, поясняющий их назначение и смысл употребляемых параметров;
- операторы, выполняющиеся как последовательность одноуровневых действий, должны начинаться с одинаковой позиции в своих строках;
- оператор, находящийся на следующем подуровне, должен начинаться с отступом на две позиции от оператора верхнего уровня;
- для оператора if then else части обеих ветвей оператора должны начинаться с одной и той же позиции в своих строках;
- для повышения читабельности программы и выравнивания текста допускается использовать пробелы и пустые строки.

Данные правила не являются жесткими требованиями. Рекомендуется и дозволяется использовать и другие правила оформления программы. В качестве практического стиля программирования может быть использован подход разработчиков Zonnon, с которым можно непосредственно ознакомиться при работе с программным продуктом в приводимых программах - примерах.

5. Рекомендуемые темы дополнительных заданий

1). Построение диаграмм и графиков эффективности алгоритмов сортировки.

2). Включение других известных алгоритмов сортировок.

3). Сортировка готовых текстов как набора слов с подсчетом количества употребляемых терминов.