Vnitřní řazení v poli (in sito)

Je dán vektor N čísel (celých, desetinných, znaků, řetězců znaků). Vytvořte proceduru pro jejich vzestupné seřazení.

Řešení: Za předpokladu, že jsou definovány typy

```
type    INDEX = 0..MAXPOCET;
        SLOZKA = integer;
         {může být real, string[20] atp.}
         A = array[INDEX] of SLOZKA;
```

kde `MAXPOCET` je definovaná konstanta např.

```
const MAXPOCET=20;
```

omezující použitelnost dále uvedených procedur na zpracování N-členných vektorů (`N` je maximálně `MAXPOCET`).

K seřazení položek použijeme metodu pří-mého výběru (straight selection).

Princip:

- Ze všech položek vektoru vyber nejmenší hodnotu.
- Vyměň vzájemně nalezenou hodnotu s hodnotou v prvé položce vektoru.
- Poté vyber nejmenší prvek ze zbylých N-1 prvků (z 2. až N-té položky) a vyměň ho s druhou položkou atd. až zůstane poslední N-tý (maximální) prvek.

```
procedure STRSEL(var POLE:A; N:INDEX);
    var I,J,K: INDEX;
        POM: SLOZKA;
    begin for I:=1  to N-1 do
              begin K:=I;  POM:=POLE[I];
                   for J:= I+1  to N do
                       if POLE[J]<POM then
                          begin K:=J;

  POM:=POLE[J]
                                end;
                   POLE[K]:=POLE[I];
                   POLE[I]:=POM
              end
    end;
```

K seřazení položek použijeme metodu pří-mého vkládání (straight insertion).

Princip:

- V i-tém kroku se zpracovává i-tá položka.
- Buď se ponechá na svém původním místě (je-li větší jak i-1. položka), nebo se vloží na správné místo.
- Tomu předchází uvolnění místa příslušné položky posunutím hodnot, počínaje i-1. položkou a konče příslušnou položkou, vždy o jedno místo vpravo.

```
procedure STRINS(var POLE:A; N:INDEX);
    var I,J :INDEX;
        POM :SLOZKA;
    begin for I:=2 to N do
              begin POM:=POLE[I];
                    POLE[0]:=POM;
                    J:=I-1;
                    while POM<POLE[J] do
                          begin
                             POLE[J+1]:=POLE[J];
                             J:=J-1
                          end;
                    POLE[J+1]:=POM
                       {též POLE[I]:=POM}
              end
    end;
```

K seřazení položek vektoru použijeme některou z metod přímé výměny. Metody jsou založeny na principu srovnávání a případné výměny sousedních položek vektoru tak dlouho, dokud není vektor seřazený:

a) bublinové třídění - Bubble sort

Princip:

- V každém kroku porovnáváme všechny položky vektoru po sousedních dvojicích (J=1..N-1).
- Je-li prvý prvek dvojice větší než druhý, provedeme výměnu.
- V i-tém kroku nám probublá i-tá nejmenší hodnota na i-té místo od začátku (vrcholu) vektoru.
- Po N-1 krocích máme jistotu, že vektor je setříděný.

```
procedure BUBBLE(var POLE:A; N:INDEX);
  var I,J :INDEX;
      POM :SLOZKA;
  begin for I:=2  to N  do
            for J:=N downto 2 do
                    {s výhodou downto I}
                if POLE[J-1]>POLE[J] then
                   begin POM:=POLE[J-1];
                         POLE[J-1]:=POLE[J];
                         POLE[J]:=POM
                   end
            end
  end;
```

b) u bublinového řazení s výhodou (tzv. Ripple sort) je vzata v úvahu skutečnost, že vektor může být setříděný dříve než po N-1 krocích.

Princip:

- Postupujeme zkoumáním J-té dvojice, pro J=1..N-I, v rámci I-tého kroku
- Testujeme, zda v I-tém kroku došlo k výměně alespoň u jedné dvojice.
- Pokud ne, je již vektor setříděn.

```
procedure RIPPLE(var POLE:A; N:INDEX);
    var J,I : INDEX;
        POM : SLOZKA;
        SETRIDENO : Boolean;
    begin I:=1;
          repeat SETRIDENO := true; {setříděno}
                 for J:=1 to POC do
                     if POLE[J+1]<POLE[J] then
                       begin POM:=POLE[J];
                             POLE[J]:=POLE[J+1];
                             POLE[J+1]:=POM;
                             SETRIDENO:=false
                          end;
                     I:=I+1
          until (I = N) or SETRIDENO
    end;
```

c) Další modifikací je Shaker sort.

Princip:

- V určitém kroku prohlížíme řazený vektor dvakrát.
- Nejprve např. od poslední (dolní) D. dvojice po horní H. dvojici.
- Zapamatujeme si číslo J-té poslední dvojice, u které jsme byli nuceni provést výměnu.
- Vzápětí prohlížíme vektor podruhé, tentokráte shora (od H = J+1. dvojice) dolů (po poslední dvojici (D)).
- Opět si zapamatujeme místo poslední výměny (I=J) a jdeme totéž opakovat do dalšího kroku.
- Seřazení vektoru nastává v okamžiku, kdy ani při průchodu zdola nahoru, ani při průchodu shora dolů neprovádíme výměnu.

```
procedure SHAKER(var POLE:A; N:INDEX);
    var I,J,H,D : INDEX;
        POM     : SLOZKA;
    begin H := 2;     D := N;     I := N;
          repeat for J := D downto H do
                    if POLE[J-1] > POLE[J] then
                       begin POM := POLE[J-1];
                             POLE[J-1] := POLE[J];
                             POLE[J] := POM;
                             I := J
                       end;
                 H := I + 1;
                 for J:= H to D do
                    if POLE[J-1] > POLE[J] then
                       begin POM := POLE[J-1];
                             POLE[J-1] := POLE[J];
                             POLE[J] := POM;
                             I := J
                       end;
                    D := I-1;
          until H > D;
    end;
```

Předchozí 3 metody (Bubble, Ripple a Shaker sort) vycházely z principu, že v každém kroku se prohlédly všechny dvojice sousedních položek vektoru.

Poslední z ukázaných jednoduchých metod řazení, je metoda Shuttle sort.

Princip:

- Metoda vychází opět z prohlížení sousedních položek tříděného vektoru, avšak jakmile je zjištěna nutnost provedení výměny, je provedena a vektor je opět prohlížen od prvých dvou položek.

```
procedure SHUTT(var POLE:A; N:INDEX);
  var I :INDEX;
      POM :INTEGER;
  begin I:=2;
        while I<=N do
              if POLE[I]<POLE[I-1]
                 then begin POM:=POLE[I];
                            POLE[I]:=POLE[I-1];
                            POLE[I-1]:=POM;
                            I:=2
                      end
                else I :=I+1
  end;
```

K seřazení většího množství dat se používají duchaplnější metody, z nichž jmenujme alespoň Quick sort, Shell sort a Heap sort.

Metoda Quick sort vychází z následujícího principu:

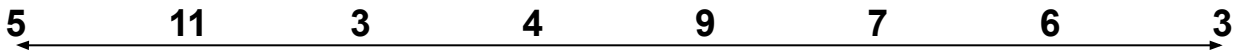
- Z vektoru se vyhledá vhodná hodnota POM1 (např. hodnota umístěná uprostřed).
- Pak se prochází vektor řazených hodnot zleva, až se najde číslo větší než POM1, načež se prohlíží vektor zprava, až se najde hodnota menší než POM1. Tato nalezená čísla se vzájemně vymění.
- Uvedený postup aplikujeme tak dlouho, dokud se výměna nedá provést a vektor je rozdělen na dvě části (např. poloviny); v levé polovině jsou čísla menší než nějaká hodnota POM1, v pravé polovině pak jsou čísla větší než nějaká hodnota POM1.
- Celý proces opakujeme s každou polovinou hodnot samostatně (dále čtvrtinou atd.), až nakonec seřadíme sousední dvojice a tím je úloha vyřešena.

```
procedure QUICK(var POLE:A; N: INDEX);
    const M = 12;
    type STRUKTURA = array[1..M] of record H, D : INDEX
                                        end;
    var I, J, H, D : INDEX;        S : 0..M;
        POM1, POM2 : SLOZKA;       Z : STRUKTURA;
begin S := 1;   Z[1].H := 1;   Z[1].D := N;
      repeat H := Z[S].H; D := Z[S].D; S := S-1;
             repeat I := H; J := D; POM1:=POLE[(H+D) div 2];
                    repeat while POLE[I] < POM1 do I := I+1;
                           while POM1 < POLE[J] do J := J-1;
                                 if I <= J then
                                    begin POM2 := POLE[I];
                                          POLE[I] := POLE[J];
                                          POLE[J] := POM2;
                                          I := I+1;  J := J-1
                                    end
                    until I > J;
                    if I < D then begin S := S+1;
                                        Z[S].H:=I; Z[S].D:=D;
                                  end;
                    D := J
             until H >= D
      until S = 0
end;                          { konec procedury QUICK }
```

Rekurzivní obdoba předchozího algoritmu:

```
procedure QUICKR(var POLE:A; N: INDEX);
    procedure SORT(H,D : INDEX);
        var I, J : INDEX;  POM1, POM2 : SLOZKA;
        begin I := H;   J := D;
              POM1 := POLE[(H+D) div 2];
              repeat while POLE[I] < POM1 do I := I+1;
                     while POM1 < POLE[J] do J := J-1;
                     if I <= J then
                        begin POM2 := POLE[I];
                              POLE[I] := POLE[J];
                              POLE[J] := POM2;
                              I := I+1;  J := J-1
                        end
              until I > J;
              if H < J then SORT(H,J);
              if I < D then SORT(I,D);
        end;                        {konec procedury SORT}
    begin SORT(1,N)
    end;                          {konec procedury QUICKR}
```

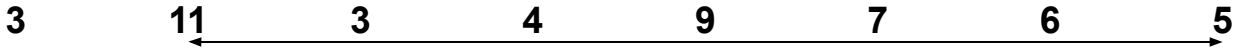
Verzí je možno vytvořit mnoho. Přístup



QUICK1
DM 1
HM 8

PRESKUP
DM 1
HM 8

J 1 8 POM 5
D 1 X
H 8



D 2
J 2

H 7 6 5 4
J 4

D 3 4

. . .

QUICK1
DM 1
HM 3

. . .

QUICK1
DM 5
HM 8

postihuje algoritmus s procedurami PRESKUP a QUICK1

```
Procedure PRESKUP(DM,HM:byte; var J:byte);
   var D, H  :byte;
       POM,X : COSI;
   begin POM:=A[DM]; J:=DM; H:=HM; D:=DM;
         repeat while (H>D) and (A[H]>=POM) do H:=H-1;
                J:=H;
                if H<>D then begin X:=A[D];
                                   A[D] := A[H];
                                   A[H]:=X;
                                   while (D<H) and (A[D]<=POM) do D:=D+1;
                                   J:=D;
                                   if H<>D then begin X:=A[H];
                                                      A[H]:=A[D];
                                                      A[D]:=X
                                                end
                             end
         until D=H;
         A[J]:=POM
   end;
Procedure QUICK1(DM,HM:byte);
   var J:byte;
   begin if DM<HM then begin PRESKUP(DM,HM,J);
                             QUICK1(DM,J-1);
                             QUICK1(J+1,HM)
                       end;
   end;
```

Algoritmus založený na metodě QUICKsort přináší také následující řešení:

```
Program QUICKSORT000;
  uses CRT;
  type COSI=integer;
  var  A   : array[1..20] of COSI;
       N,I : byte;
  Procedure QUICKRRR(DM,HM:byte);
  var D, H  :byte;
      POM,X : COSI;
     Procedure ROZDEL;
     begin POM:=A[(DM+HM) div 2]; H:=HM; D:=DM;
           repeat while  A[H]>POM do H:=H-1;
                  while  A[D]<POM do D:=D+1;
                  if D<=H then begin X:=A[D];
                                     A[D] := A[H];
                                     A[H]:=X;
                                     D:=D+1;
                                     H:=H-1
                               end
           until D>H;
     end;
  begin ROZDEL;
        if DM<H then QUICKRRR(DM,H);
        if HM>D then QUICKRRR(D,HM);
  end;
begin ClrScr;
     Write('Zadej pocet vstupnich hodnot: '); Readln(N);
     for I:=1 to N do begin Write('Zadej ',I,'. hodnotu: '); Readln(A[I]) end;
     QUICKRRR(1,N);
     for I:=1 to N do write(A[I]:3)
end.
```

Použití každé z výše uvedených řadících procedur vyžaduje ***stejné*** definice a deklarace. Můžeme tedy vytvořit program např. ve tvaru:

```
program SETRIDENI;
{*  Definice typů, které byly předpokládány na začátku *}
    var  POCET, K: INDEX;          HODNOTY : A;
{*  Na tomto místě zapíšeme kteroukoliv z výše uvedených  *}
{*  deklarací procedur na řazení (např. proceduru SHUTT)  *}
  begin write('Zadej pocet nacitanych hodnot k razeni: ');
        readln(POCET);
        for K:=1 to POCET do begin write('Zadej ',K,'. hodnotu: ');
                                   readln(HODNOTY[K])
                            end;
{*  Na tomto místě zavoláme deklarovanou proceduru,  *}
{*  např. SHUTT(HODNOTY,POCET);  *}
        writeln('Tisk serazenych hodnot:');
        for K:=1 to POCET do writeln(K,'. hodnota je ',HODNOTY[K]:4);
  end.
```

Metoda Shell sort, neboli Shellova metoda řazení (řazení se snižujícím se přírůstkem) .

Princip:

- Rozdělení řazených hodnot na 4 skupiny tak, že prvky každé skupiny jsou od sebe vzdáleny o 4 složky (1. skupinu tvoří 1., 5., 9. ... složka řazeného vektoru, 2. skupinu 2., 6., 10. ... atd).
- Každou skupinu seřadíme zvlášť nějakou známou metodou řazení.
- V další etapě rozdělíme pole řazených hodnot na 2 skupiny tak, že prvky každé skupiny jsou od sebe vzdáleny o 2 složky (tj. složky 1., 3., 5. ... resp. 2., 4., 6. ...).
- V poslední fázi seřadíme celou posloupnost případnou výměnou sousedních dvojic.

Následující program má stejnou logiku jako poslední výše uvedený program. Za povšimnutí stojí pouze jiná definice typu A nutná pro použití procedury SHELL:

```
Program SETRIDENI;
    const MAXPOCET= 30;
          PS      = 4;
    type SLOZKA   = integer;
         INDEX    = 0..MAXPOCET;
         A = array[-9..MAXPOCET] of SLOZKA;
    var POCET, K: INDEX;
        HODNOTY : A;
```

```
procedure SHELL(var POLE:A; N: INDEX);
    type STRUKTURA = array[1..PS] of INDEX;
    var I, J, R, S : integer;
        POM        : SLOZKA;
        M          : 0..PS;
        PO         : STRUKTURA;
    begin PO[1]:=9;    PO[2]:=5;    PO[3]:=3;    PO[4]:=1;
          for M:=1 to PS do
              begin R:=PO[M]; S:=-R;
                    for I:=R+1 to N do
                        begin POM:=POLE[I]; J:=I-R;
                              if S=0 then S:=-R;
                              S:=S+1;   POLE[S]:=POM;
                              while POM<POLE[J] do
                                    begin POLE[J+R]:=POLE[J];
                                          J:=J-R
                                    end;
                              POLE[J+R]:=POM
                        end
              end
    end; { konec procedury SHELL }
```

```
begin
      write('Zadej pocet nacitanych hodnot ke
  trideni: ');
      readln(POCET);
      for K:=1 to POCET do
          begin write('Zadej ',K,'. hodnotu: ');
                readln(HODNOTY[K])
          end;
      SHELL(HODNOTY,POCET);
      writeln('Tisk setridenych hodnot:');
      for K:=1 to POCET do
          writeln(K,'. hodnota je ',HODNOTY[K]:4);
end.
```

K posouzení efektivnosti zmíněných metod poslouží následující tab. časové náročnosti uvedených metod. Z tabulky je patrné, že na metodě bubli-nového řazení je zajímavý snad její název.

stav vektoru	**setříděný**		**náhodný**		**opačně setř.**	
počet prvků	256	512	256	512	256	512
Přímý výběr	489	1907	509	1956	695	2675
Přímé vkládání	12	23	366	1444	704	2836
Bubble sort	540	2165	1026	4054	1442	5931
Ripple sort	5	8	1104	4270	1645	6542
Shaker sort	5	9	961	3642	1619	6520
Shell sort	58	116	127	349	157	492
Quick sort	31	69	60	146	37	79

Závěrem tohoto příkladu ještě jednou poznamenejme, že tvar všech výše uvedených algoritmů zůstává zcela zachován i pro řazení číselných reálných hodnot, případně řetězců znaků, či jen znaků. V části definicí programu je třeba pouze změnit definici typu SLOZKA na např.

```
SLOZKA = real nebo
SLOZKA = string[20] nebo
SLOZKA = char atp.
```