Digitální učební materiál

Číslo projektu:	CZ.1.07/1.5.00/34.0548
Název školy:	Gymnázium, Trutnov, Jiráskovo náměstí 325
Název materiálu:	VY_32_INOVACE_151_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	srpen 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus XI. – Třídění dat III. - Bubble Sort
Metodický pokyn:	Při výuce nutno postupovat individuálně. Části DUM – „ Pro hloubavé“ jsou určeny pro zájemce o studium na technických a matematicko-fyzikálních oborech vysokých škol.

esf evropský sociální fond v ČR
EVROPSKÁ UNIE
MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY
OP Vzdělávání pro konkurenceschopnost
INVESTICE DO ROZVOJE VZDĚLÁVÁNÍ

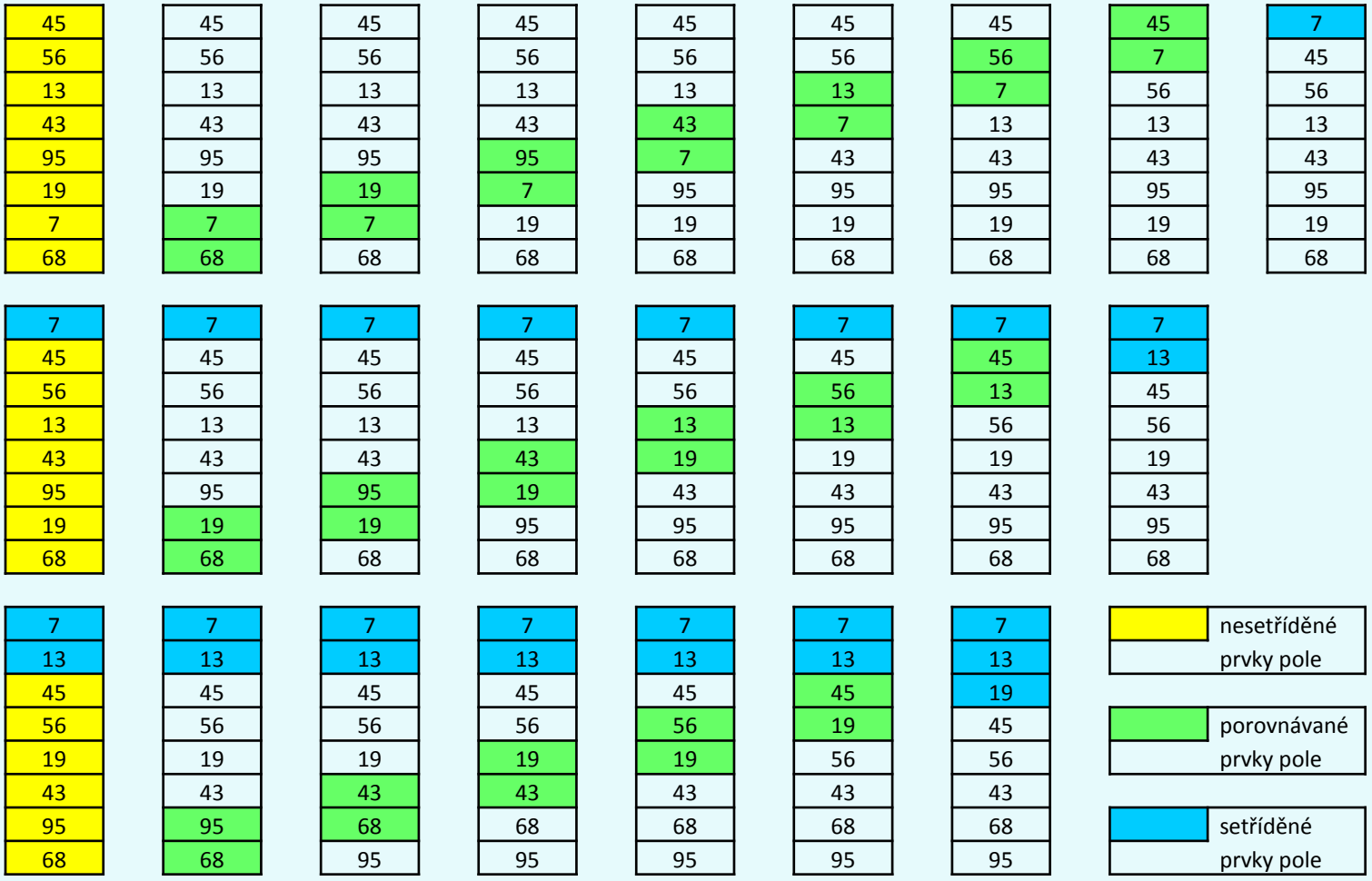
Autor	**Ing. Pavel Bezděk**		
Vytvořeno dne	**15. 8. 2013**		
Odpilotováno dne	**6. 3. 2014**	ve třídě	**4. A**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus XI. - Třídění dat III. - Bubble Sort**		
Klíčová slova	**Algoritmus, Bubble Sort**		

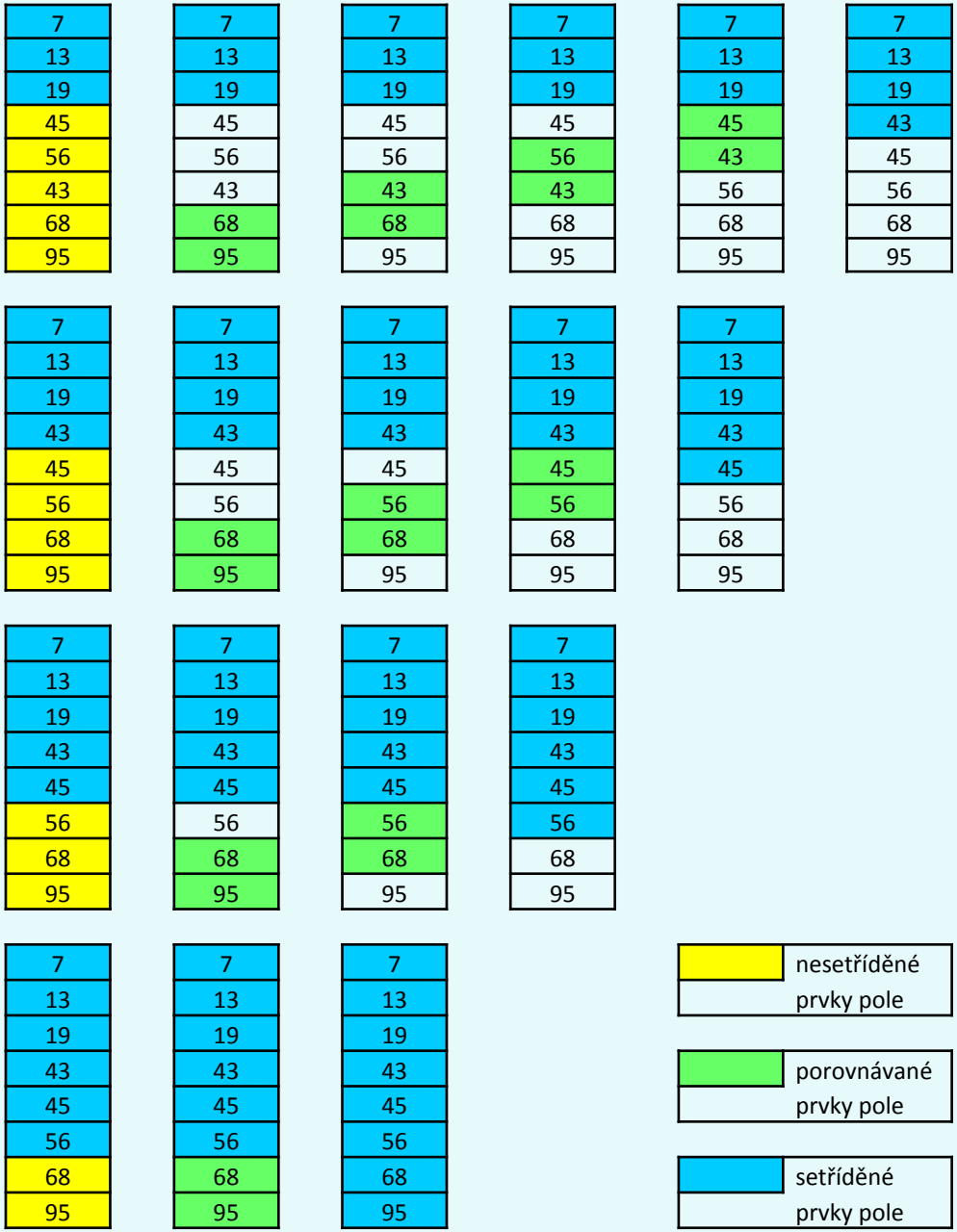
Bubble Sort
bublinkové třídění

Bublinkové třídění
BUBBLE SORT

```
program Bublinkove_Trideni;
uses CRT;  const N = 10;
type Pole = array[1..N] of integer;
var i:integer;    A:Pole;
procedure BublinkoveTrideni(var A: Pole);
 {zacatek deklarace procedury BublinkoveTrideni}
var i,j: integer;                    {indexy prvku}
  X: integer;                        {pro vymenu prvku}
  begin  for i:=2 to N do
    for j:=N downto i do      if A[j-1] > A[j] then {vymenit sousedni prvky}
    begin X:=A[j-1]; A[j-1]:=A[j]; A[j]:=X  end
   end;     { konec deklarace procedury BublinkoveTrideni}
begin { zacatek tela programu}
writeln('Zadej netridena cisla:');
for i:=1 to N do read(A[i]);
BublinkoveTrideni(A);            {volani procedury BublinkoveTrideni}
writeln('Setrideno:');
for i:=1 to N do write(A[i]:5);  repeat until keypressed;
end.
```

Schéma bublinkového třídění




7 13 19 45 56 43 68 95
7 13 19 45 56 43 68 95
7 13 19 45 56 43 68 95
7 13 19 45 56 43 68 95
7 13 19 45 43 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
7 13 19 43 45 56 68 95
nesetříděné prvky pole
porovnávané prvky pole
setříděné prvky pole

Časová a paměťová složitost Bubble Sort

Složitost : **nejlepší př.:** $C= (N^2-N)/2$ $M=0$

průměr. př.: $C= (N^2-N)/2)$ $M=(N^2-N)/0,75$

nejhorší př. : $C= (N^2-N)/2$ $M=(N^2-N)*1,5$

C: časová složitost M: paměťová složitost

Asymptotická složitost :

časová $O(n^2)$ **nejhorší, průměrný i nejlepší případ**

$\Omega(n^2)$

paměťová $O(n^2)$ **- nejhorší a průměrný případ**

$O(1)$ **- nejlepší případ**

Princip třídění Bubble Sort

Vstupní posloupnost čísel (sekvenci) je třeba uspořádat tak, že sekvence bude neklesající. K tomuto účelu využijeme třídící algoritmus, který se nazývá Bubble Sort (bublinkové řazení). Bubble Sort řadí sekvenci na základě průchodů sekvencí.

Při každém průchodu se provádí výměna sousedních prvků, pro které neplatí $\mathbf{a_i < a_{i+1}}$.

Začnu se polem **1..N**, porovnávám poslední a předposlední prvek pole, když je prvek s menším indexem větší, prvky prohodím. Je-li menší nebo roven, nic neudělám. Vezmu další dvojici prvků, předposlední a předpředposlední a porovnám, a tak až po 2. a 1. prvek. Tím se nejmenší prvek v poli dostane na 1. místo v poli (probublá nahoru, když si představím, že 1.prvek pole mám nahoře a poslední dole).

Totéž udělám s polem **2..N** a nechám probublat 2. nejmenší prvek (teď nejmenší prvek pole **2..N**) na 2. místo v poli **1..N**.

A totéž postupně s poli **3..N**, **4..N**, až **(N-1)..N**.

Nejmenší prvek v těchto polích vždy probublá nahoru a tím dojde k setřídění celého pole **1..N**

Pro hloubavé

Shaker sort - přetřásání

Shaker sort (*shake sort*, *cocktail sort*) je stabilní řadící algoritmus s asymptotickou časovou složitostí $O(n^2)$.

Shakersort je vylepšením BubbleSortu.

Princip

Shakersort na rozdíl od *bubble sortu* neřadí pole pouze jedním směrem, ale oběma.

Každá iterace algoritmu se tedy skládá z dvou fází - při **dopředné** stoupá nejlehčí bublinka vzhůru (nejmenší číslo), pří **zpětné** klesá nejtěžší bublinka ke dnu (největší číslo). Tímto postupem se předejde nedostatku *bubble sortu* tzv. *problému želv a zajíců*, který spočívá v tom, že vysoké hodnoty probublají na konec pole rychle, ale ty nízké postupují na začátek velmi pomalu.

Tato výhoda se však uplatní hlavně, když třídíme pole, které je už téměř setříděné. Odpadnou některé operace, která bychom dělali v Bubble Sort.

To je ale v praxi výjimečný případ.

Procedura ShakerSort

Pascal

```
program Shakersort1;
var pole : array[1..10]of integer;
procedure ShakerSort(max: integer);
  var k, l, r, pom, i, j: integer;
  begin
     l := 2; r := max; k := max;
     repeat
       for j := r downto l do if pole[j-1] > pole[j] then
          begin pom := pole[j-1]; pole[j-1] := pole[j]; pole[j] := pom; k := j;
end;
       l := k + 1;
       for j := l to r do if pole[j-1] > pole[j] then
          begin pom := pole[j-1]; pole[j-1] := pole[j]; pole[j] := pom; k := j;
end;
       r := k - 1;
    until l > r;
 end;
begin
end.
```

Procedura ShakerSort

Pascal

jiná verze

```
procedure ShakeSort(var X : ArrayType; N : integer);
var   L,   R,   K,   J : integer;
begin
 L := 2;   R := N;   K := N;
 repeat
  for J := R downto L do
   if (X[J] < X[J - 1]) then
     begin   Swap(X[J], X[J - 1]);   K := J   end;
  L := K + 1;
  for J := L to R do       if (X[J] < X[J - 1]) then
     begin  Swap(X[J], X[J - 1]);  K := J   end;
  R := K - 1;
 until L >= R
end;
procedure Swap(var X, Y : integer); {Vymena}
var   Temp : integer;
begin   Temp := X;   X := Y;   Y := Temp end;
```

Doporučená videa

Bubble Sort:

http://www.youtube.com/watch?v=lyZQPjUT5B4

http://www.youtube.com/watch?v=VV18nfE4erU

Použité zdroje

WIRTH, Niklaus. *Algoritmy a štruktúry údajov*. 2.vyd. Bratislava: Alfa, 1989, 481 s. ISBN 80-050-0153-3.

MIČKA, Pavel. *Algoritmy.net: Příručka vývojáře* [online]. [cit. 2013-08-15]. Dostupné z: http://www.algoritmy.net/article/93/Shaker-sort Licence MIT.