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_147_IVT
Autor:	Ing. Pavel Bezděk
Tematický okruh:	Algoritmy
Datum tvorby:	červenec 2013
Ročník:	4. ročník a oktáva
Anotace:	Algoritmus VII. – Nejhorší, průměrný a nejlepší případ algoritmu
Metodický pokyn:	Při výuce nutno postupovat individuálně.

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	**28. 7. 2013**		
Odpilotováno dne	**27. 1. 2014**	ve třídě	**8.Y**
Vzdělávací oblast	**Informatika a informační a komunikační technologie**		
Vzdělávací obor	**Informatika a výpočetní technika**		
Tematický okruh	**Algoritmus**		
Téma	**Algoritmus VII. - Nejhorší, průměrný a nejlepší případ algoritmu**		
Klíčová slova	**Algoritmus, nejhorší případ algoritmu**		

Složitost algoritmu v nejlepším, průměrném a nejhorším případě

Nejhorší, průměrný a nejlepší případ

Opět si vše vysvětlíme jen na časové složitosti. Pro paměťovou složitost je to obdobné. **Velká část algoritmů běží pro různé vstupy *stejné* velikosti různou dobu.** U takových algoritmů pak můžeme rozlišovat složitost v nejhorším případě, v nejlepším případě a třeba i průměrnou časovou složitost.

Časová složitost v nejhorším případě

Časová složitost z hlediska nejhoršího případu udává pro každé N, jak nejdéle může trvat výpočet podle algoritmu s libovolnými vstupními daty velikosti N. **Má význam horní meze, kterou doba výpočtu s daty velikosti N rozhodně nepřekročí. Analýzou algoritmu často snadno určíme jeho časovou složitost v nejhorším případě, nebo aspoň hrubší horní odhad této složitosti.**

Časová složitost v průměrném případě určuje pro každé N průměrnou délku výpočtu při zadání dat velikosti N. ***Má význam doby, kterou můžeme v průměru očekávat, že bude k výpočtu zapotřebí.*** Většinou lépe charakterizuje algoritmus, než složitost v nejhorším případě.

Obvykle je ale dost obtížné průměrnou časovou složitost zjistit. Proto častěji při posuzování algoritmů pracujeme se složitostí z hlediska nejhoršího případu.

Časová složitost v nejlepším případě Ve zvláštních případech mohou být vstupní data taková, že se algoritmu zjednoduší výpočet a jeho složitost je nižší, ale pouze v tom zvláštním případě.

Má význam dolní meze, doba výpočtu s daty velikosti N rozhodně nebude menší než v tomto případě, přičemž k této min. době může dojít jen ve zvláštním případě (výjimečně). **Většinou nepoužíváme !**

Hledání max. prvku v poli Pascal

```
Program Maximum;
uses Crt;
const  N=20;
type Pole = array[1..N] of integer;
var  P: Pole;  Max,i, Kolikaty: integer ;
begin
clrscr;
writeln;
Writeln('Vyhledani max. prvku v poli:');
writeln('Pole ma 20 prvku.');
writeln('Chces-li vice prvku pole,');
writeln('pak zmen konstantu N ve zdrojovem textu programu!');
writeln;
writeln('Mas-li mene prvku pole, nez je 20,');
writeln('pak do ostatnich (neobsazenych) prvku pole zadej nuly.');
writeln; writeln;
```

```
For i:=1 to N do begin
            write(i:3,' .prvek pole: ');
            readln(P[i]);
            end;
Max:=P[1]; Kolikaty:=1;
For i:=2 to N do
          begin   if P[i]>Max
                then begin
                    Max:=P[i];
                    Kolikaty:=i;
                    end;
          end;
writeln; writeln;
writeln('Maximun = ',Max);
writeln('Maximalni prvek je ',Kolikaty,'. prvek v poli.');
repeat until keypressed;
end.
```

Program maximální prvek pole v C++

```
#include <iostream>
//Maximalni prvek pole
using namespace std;
int i,maximum,pom,kolikaty;
const int index=10;
int pole[index];
int main()
{
 cout<<"Zadej prvky pole"<<endl;
 for (i=0;i<index;i++)
 {   cout<<"Zadej "<<i<<". prvek pole:";
     cin >>pole[i];
 }
 cout<< endl;
 cout<<"Zadali jste tyto prvky pole: "<<endl;
 for (i=0;i<index;i++)
 {   cout<<i<<". prvek pole: "<<pole[i]<<endl;
 }
   cout << endl;
 pom=pole[0]; kolikaty=0;
 for (i=1;i<index;i++)
 {   if (pom<pole[i]) {pom=pole[i]; kolikaty=i; }
 }
 maximum=pom;
 cout<<"Nejvetsi prvek pole je "<< maximum<< "  a je to "<<kolikaty<<". prvek pole"<< endl;
   return 0;
}
```

Časová složitost hledání max. prvku

Časovou složitost určujeme podle počtu základních operací – v našem případě operace přiřazení a porovnání.

Nejhorší případ: Prvky pole seřazeny od nejmenšího k největšímu

Pole: 4, 6, 8, 9, 15, 18, 22, 35, 41, 50

Max= první prvek pole 1x přiřazení Kolikaty:=1 1x přiřazení

Cyklus 2 až N P[i]>Max (N-1)x 1 porovnání

Max=P[i] (N-1)x 1 přiřazení Kolikaty:=i (N-1)x 1 přiřazení

Celkem 1+1+N-1+N-1+N-1=3N-1 **Časová složitost lineární 3N-1**

Nejlepší případ: Max. prvek je první v poli

Pole: 50, 41, 35, 22, 18, 15, 9, 8, 6, 4

Max= první prvek pole 1x přiřazení Kolikaty:=1 1x přiřazení

Cyklus 2 až N P[i]>Max (N-1)x 1 porovnání žádné přiřazení v cyklu

Celkem 1+1+N-1=N +1 **Časová složitost lineární N+1**

Průměrný případ: Prvky pole nejsou seřazeny

Pole: 22, 8, 4, 35, 18, 50, 6, 41, 9, 15

Max= první prvek pole 1x přiřazení Kolikaty:=1 1x přiřazení

Cyklus 2 až N P[i]>Max (N-1)x 1 porovnání

Max=P[i] **c** x 1 přiřazení **c** leží v intervalu <1,(N-1)>

Kolikaty:=i **c** x 1 přiřazení **c** leží v intervalu <1,(N-1)> **Časová složitost lineární N+1+2c**

Celkem 1+1+N-1+**c**+**c**=N+1+2**c** **c** leží v intervalu <1,(N-1)>

Asymptotická složitost hledání max. prvku

Asymptotická časová složitost

Nejlepší případ	**N+1**	**O(N)**
Průměrný případ	**N+1+2c**	**O(N)**
Nejhorší případ	**3N-1**	**O(N)**

Takže asymptotická složitost tohoto algoritmu je v nejhorším i nejlepším případě lineární, tak nikdy nebude horší nebo lepší než lineární.

Tedy je vždy lineární O(N)!

Asymptotická paměťová složitost

Ve všech případech potřebujeme N paměťových buněk pro pole a jednu buňku pro maximum a jednu pro Kolikaty. Celkem N+2.

Asymptotická paměťová složitost je vždy O(N) - lineární!

Třídění přímým vkládáním
INSERT SORT Pascal

```
program Prime_Vkladani;
uses CRT;  const N = 10;
type Pole = array[1..N] of integer;   var i:integer;  A:Pole;
 procedure PrimeVkladani(var A: Pole); {začátek deklarace procedury
PrimeVkladani}
var i,j: integer;            {indexy prvku}
 X: integer;              {pro výměnu prvku}
Hledat:Boolean;
begin  for i:=2 to N do          {zatřiďujeme číslo z pozice i}
  begin    X:=A[i];    j:=i-1;
  Hledat:=X<A[j];    while Hledat do           {hledáni správné pozice}
    begin      A[j+1]:=A[j];      j:=j-1;      if j=0 then Hledat:=false
                                                 else      Hledat:=X<A[j]
    end;
  A[j+1]:=X
   end
end;  {konec deklarace procedury PrimeVkladani}
begin
writeln('Zadej ', N, ' netříděných čísel:');                  for i:=1 to N do read(A[i]);
PrimeVkladani(A);    {volání procedury PrimeVkladani}
writeln('Setříděno:');    for i:=1 to N do write(A[i]:5);    repeat until keypressed;
end.
```

Třídění přímým vkládáním
INSERT SORT C++

```
#include <iostream>
//#include <conio.h>
using namespace std;
const int index=10;
int a[index];                // pole tridenych prvku
int pole[index];
int pole2[index];
int i;
void insert(int p[],int n);
using namespace std;
int main()
{   cout<<"Zadej prvky pole"<<endl;
 for (i=0;i<index;i++)  {  cout<<"Zadej "<<i<<". prvek pole:";     cin >>a[i]; }
 cout<< endl;
 cout<<"Zadali jste tyto prvky pole: "<<endl;
 for (i=0;i<index;i++)  {  cout<<i<<". prvek pole: "<<a[i]<<endl; }
   cout << endl;
int pocet=index;        /* vstupni parametr - pocet prvku */
            int k;
            for (k=0; k<pocet; k++)    /* pole tridenych prvku presuneme */
                        pole[k]=a[k];      /* do vstupniho pole pro trideni  */
      cout<< "Prvky pole, ktere mame setridit: "<<endl;
for (k=0; k<pocet; k++)              /* cyklus pro zobrazeni vysledku  */
                        cout<<k<<". prvek pole:"<< pole[k]<<endl;
```

```
                    for (k=0; k<pocet; k++)     /* pole tridenych prvku presuneme */
                    pole2[k+1]=a[k];            /* do vstupniho pole pro trideni  */
                    cout<<endl;
          insert(pole2,pocet);                  /* zde pouziji o 1 delsi pole2    */
          cout<<("Setridene pole (insertsort): \n");
          for (k=1; k<=pocet; k++)              /* cyklus pro zobrazeni vysledku  */
                    cout<<k-1<<". prvek pole:"<< pole2[k]<<endl;
                    cout<< endl;            /*zalomi radek po vytisteni vysledku */
          for (k=0; k<pocet; k++)
                    pole[k]=a[k];
          return 0;
}
void insert(int p[],int n)       /* trideni vkladanim - insertsort */
  /* zde jsou tridene prvky ulozeny v poli od indexu 1 */  /* p[0] plni ulohu zarazky   */
{
int i,j;
for (i=2;i<=n;i++)  {    p[0] = p[i]; j = i-1;        while (p[0] < p[j])
                         {   p[j+1] = p[j]; j--;  }
                         p[j+1] = p[0];
                    }
}
```

Schéma třídění Insert Sort - přímým vkládáním

	nesetříděná část pole
	porovnávané prvky
	setříděná část pole

	45	56	13	43	95	19	7	68
II. prvek pole	45	56	13	43	95	19	7	68

	45	56	13	43	95	19	7	68
III. prvek pole	45	56	13	43	95	19	7	68
	45	13	56	43	95	19	7	68

	13	45	56	43	95	19	7	68
IV. prvek pole	13	45	56	43	95	19	7	68
	13	45	43	56	95	19	7	68
	13	43	45	56	95	19	7	68

	13	43	45	56	95	19	7	68
V. prvek pole	13	43	45	56	95	19	7	68

	nesetříděná část pole
	porovnávané prvky
	setříděná část pole

VI. prvek pole

13	**43**	**45**	**56**	**95**	**19**	**7**	**68**
13	43	45	56	**95**	**19**	7	68
13	43	45	**56**	**19**	95	7	68
13	43	**45**	**19**	56	95	7	68
13	**43**	**19**	45	56	95	7	68
13	**19**	43	45	56	95	7	68

VII. prvek pole

13	**19**	**43**	**45**	**56**	**95**	**7**	**68**
13	19	43	45	56	**95**	**7**	68
13	19	43	45	**56**	**7**	95	68
13	19	43	**45**	**7**	56	95	68
13	19	**43**	**7**	45	56	95	68
13	**19**	**7**	43	45	56	95	68
13	**7**	19	43	45	56	95	68

VIII. prvek pole

7	**13**	**19**	**43**	**45**	**56**	**95**	**68**
7	13	19	43	45	56	**95**	**68**
7	13	19	43	45	**56**	**68**	95

7	**13**	**19**	**43**	**45**	**56**	**68**	**95**

Časová a paměťová složitost přímého vkládání

Začnu 2. prvkem v poli, porovnám ho s 1. prvkem poli (s prvkem s hodnotou indexu o 1 menší), pokud je prvek s vyšším indexem menší, prvky prohodím. Pokud je prvek s vyšším indexem větší nebo roven, nic neprovedu.

Pak vezmu 3. prvek v poli a porovnám s 2. prvkem (s prvkem s hodnotou indexu o 1 menší), po porovnání a případném prohození, porovnám 2. prvek s 1.prvkem.

Vezmu 4. prvek a porovnám s 3. prvkem, pak 3. prvek s 2. prvkem, 2. prvek s 1. prvkem. Na začátku pole se mi vytváří setříděná část pole, do které postupně vkládám prvky z nesetříděné části pole (na opačné straně pole).

Pak vezmu 5. prvek a porovnám s 4. prvkem, ……., až 2. a 1. prvek.

Nakonec beru N. prvek a porovnávám s (N-1). prvkem , (N-1) a (N-2) prvek, až se dostanu k porovnání 2. a 1. prvku

Složitost : nejlepší př.: $C= (N-1)$; $M=2(N-1)$ **průměr. př.:** $C = (N^2-N-2)/4$

$M=(N^2-9N-10)/4$

nejhorší př. : $C= (N^2-N)/2-1$; $M=N^2+3N-4)/2$

C: časová složitost

M: paměťová složitost

Asymptotická složitost : **časová $O(n^2)$ - nejhorší a prům. př.; $O(n)$ - nejlepší př.**

paměťová $O(n^2)$ - nejhorší a prům. př.; $O(n)$ - nejlepší př.

Když jsou data již správně setříděná (od nejmenší hodnoty k největší), není tedy již co třídit, je složitost časová i paměťová lineární.

Použité zdroje

BÖHM, Martin. *Programátorská kuchařka: Recepty z programátorské kuchařky* [online]. Praha: KSP MFF UK Praha, 2011/2012 [cit. 2013-07-28]. KSP, Korespondenční seminář z programování: Programátorské kuchařky, 24. ročník KSP. Dostupné z: http://ksp.mff.cuni.cz/tasks/24/cook1.html

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