Algorytmy Sortujące

Prezentowane materiały są przeznaczone dla uczniów szkół ponadgimnazjalnych.

Autor artykułu: mgr Jerzy Wałaszek, Wersja 4.1

Sortowanie szybkie
Quick Sort

Podrozdziały

Algorytm
Tworzenie partycji
Specyfikacja problemu
Lista kroków
Schemat blokowy

Programy
Program w języku Pascal
Program w języku C++
Program w języku Basic
Program w języku JavaScript

Badanie algorytmów sortujących
Podsumowanie
Zadania dla ambitnych

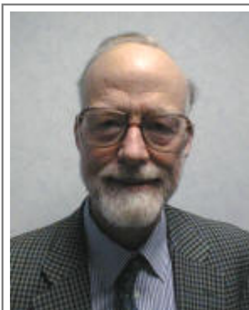
Algorytm

Algorytm sortowania szybkiego opiera się na strategii "dziel i zwyciężaj" (ang. divide and conquer), którą możemy krótko scharakteryzować w trzech punktach:

1. **DZIEL** - problem główny zostaje podzielony na podproblemy
2. **ZWYCIĘŻAJ** - znajdujemy rozwiązanie podproblemów
3. **POŁĄCZ** - rozwiązania podproblemów zostają połączone w rozwiązanie problemu głównego

Idea sortowania szybkiego jest następująca:

DZIEL :	najpierw sortowany zbiór dzielimy na dwie części w taki sposób, aby wszystkie elementy leżące w pierwszej części (zwanej lewą partycją) były mniejsze lub równe od wszystkich elementów drugiej części zbioru (zwanej prawą partycją).
ZWYCIĘŻAJ :	każdą z partycji sortujemy rekurencyjnie tym samym algorytmem.
POŁĄCZ :	połączenie tych dwóch partycji w jeden zbiór daje w wyniku zbiór posortowany.


prof. Tony Hoare

Sortowanie szybkie zostało wynalezione przez angielskiego informatyka, profesora Tony'ego Hoare'a w latach 60-tych ubiegłego wieku. W przypadku typowym algorytm ten jest najszybszym algorytmem sortującym z klasy złożoności obliczeniowej $O(n \log n)$ - stąd pochodzi jego popularność w zastosowaniach. Musimy jednak pamiętać, iż w pewnych sytuacjach (zależnych od sposobu wyboru piwotu oraz niekorzystnego ułożenia danych wejściowych) klasa złożoności obliczeniowej tego algorytmu może się degradować do $O(n^2)$, co więcej, poziom wywołań rekurencyjnych może spowodować przepełnienie stosu i zablokowanie komputera. Z tych powodów algorytmu sortowania szybkiego nie można stosować bezmyślnie w każdej sytuacji tylko dlatego, iż jest uważany za jeden z najszybszych algorytmów sortujących - zawsze należy przeprowadzić analizę możliwych danych wejściowych

właśnie pod kątem przypadku niekorzystnego - czasem lepszym rozwiązaniem może być zastosowanie wcześniej opisanego algorytmu sortowania przez kopcowanie, który nigdy nie degraduje się do klasy $O(n^2)$.

Tworzenie partycji

Do utworzenia partycji musimy ze zbioru wybrać jeden z elementów, który nazwiemy **piwotem**. W lewej partycji znajdą się wszystkie elementy niewiększe od piwotu, a w prawej partycji umieścimy wszystkie elementy niemniejsze od piwotu. Położenie elementów równych nie wpływa na proces sortowania, zatem mogą one występować w obu partycjach. Również porządek elementów w każdej z partycji nie jest ustalony.

Jako piwot można wybierać element pierwszy, środkowy, ostatni, medianę lub losowy. Dla naszych potrzeb wybierzemy element środkowy:

piwot ← d[(lewy + prawy) **div** 2]

piwot - element podziałowy
d[ ] - dzielony zbiór
lewy - indeks pierwszego elementu
prawy - indeks ostatniego elementu

Dzielenie na partycje polega na umieszczeniu dwóch wskaźników na początku zbioru - i oraz j. Wskaźnik i przebiega przez zbiór poszukując wartości mniejszych od $piwotu$. Po znalezieniu takiej wartości jest ona wymieniana z elementem na pozycji j. Po tej operacji wskaźnik j jest przesuwany na następną pozycję. Wskaźnik j zapamiętuje pozycję, na którą trafi następny element oraz na końcu wskazuje miejsce, gdzie znajdzie się $piwot$. W trakcie podziału $piwot$ jest bezpiecznie przechowywany na ostatniej pozycji w zbiorze.

Przykład:

Dla przykładu podzielimy na partycje zbiór:

{ 7 2 4 7 3 1 4 6 5 8 3 9 2 6 7 6 3 }

Lp.	Operacja	Opis
1.	7 2 4 7 3 1 4 6 **5** 8 3 9 2 6 7 6 3	Wyznaczamy na piwot element środkowy.
2.	7 2 4 7 3 1 4 6 **3** 8 3 9 2 6 7 6 **5**	Piwot wymieniamy z ostatnim elementem zbioru
3.	7 2 4 7 3 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Na początku zbioru ustawiamy dwa wskaźniki. Wskaźnik i będzie przeglądał zbiór do przedostatniej pozycji. Wskaźnik j zapamiętuje miejsce wstawiania elementów mniejszych od piwotu
4.	**7** **2** 4 7 3 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Wskaźnikiem i szukamy elementu mniejszego od piwotu

5.	**2** **7** 4 7 3 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Znaleziony element wymieniamy z elementem na pozycji *j*-tej. Po wymianie wskaźnik *j* przesuwamy o 1 pozycję.
6.	2 **7** **4** 7 3 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Szukamy
7.	2 **4** **7** 7 3 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Wymieniamy i przesuwamy *j*.
8.	2 4 **7** 7 **3** 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Szukamy
9.	2 4 **3** **7** **7** 1 4 6 3 8 3 9 2 6 7 6 **5** i j	Wymieniamy i przesuwamy *j*.
10.	2 4 3 **7** 7 **1** 4 6 3 8 3 9 2 6 7 6 **5** i j	Szukamy
11.	2 4 3 **1** **7** **7** 4 6 3 8 3 9 2 6 7 6 **5** i j	Wymieniamy i przesuwamy *j*.
12.	2 4 3 1 **7** 7 **4** 6 3 8 3 9 2 6 7 6 **5** i j	Szukamy
13.	2 4 3 1 **4** **7** **7** 6 3 8 3 9 2 6 7 6 **5** i j	Wymieniamy i przesuwamy *j*.
14.	2 4 3 1 4 **7** 7 6 **3** 8 3 9 2 6 7 6 **5** i j	Szukamy
15.	2 4 3 1 4 **3** **7** 6 **7** 8 3 9 2 6 7 6 **5** i j	Wymieniamy i przesuwamy *j*.
16.	2 4 3 1 4 3 **7** 6 7 8 **3** 9 2 6 7 6 **5** i j	Szukamy
17.	2 4 3 1 4 3 **3** **6** 7 8 **7** 9 2 6 7 6 **5** i j	Wymieniamy i przesuwamy *j*.
18.	2 4 3 1 4 3 3 **6** 7 8 7 9 **2** 6 7 6 **5** i j	Szukamy

19.	2 4 3 1 4 3 3 **2** **7** 8 7 9 **6** 6 7 6 **5** i j	Wymieniamy i przesuwamy j.
20.	2 4 3 1 4 3 3 2 **5** 8 7 9 6 6 7 6 7 ^ i Lewa partycja j Prawa partycja	Brak dalszych elementów do wymiany. $Piwot$ wymieniamy z elementem na pozycji j-tej. Podział na partycje zakończony.

Po zakończeniu podziału na partycje wskaźnik j wyznacza pozycję piwotu. Lewa partycja zawiera elementy mniejsze od $piwotu$ i rozciąga się od początku zbioru do pozycji j - 1. Prawa partycja zawiera elementy większe lub równe $piwotowi$ i rozciąga się od pozycji $j + 1$ do końca zbioru. Operacja podziału na partycje ma liniową klasę złożoności obliczeniowej - $O(n)$.

Specyfikacja problemu

Sortuj_szybko(*lewy, prawy*)

Dane wejściowe

d[ ] - Zbiór zawierający elementy do posortowania. Zakres indeksów elementów jest dowolny.

lewy - indeks pierwszego elementu w zbiorze, *lewy* Î C

prawy - indeks ostatniego elementu w zbiorze, *prawy* Î C

Dane wyjściowe

d[ ] - Zbiór zawierający elementy posortowane rosnąco

Zmienne pomocnicze

piwot - element podziałowy

i, *j* - indeksy, *i*, *j* Î C

Lista kroków

K01: $i \leftarrow [\frac{lewy + prawy}{2}]$

K02: *piwot* ← d[*i*]; d[*i*] ← d[*prawy*]; *j* ← *lewy*

K03: **Dla** *i* = *lewy*, *lewy* + 1, ..., *prawy* - 1: **wykonuj** K04...K05

K04: **Jeśli** d[*i*] ≥ *piwot*, **to wykonaj kolejny obieg pętli** K03

K05: d[*i*] ↔ d[*j*]; *j* ← *j* + 1

K06: d[*prawy*] ← d[*j*]; d[*j*] ← *piwot*

K07: **Jeśli** *lewy* < *j* - 1, **to Sortuj_szybko**(*lewy*, *j* - 1)

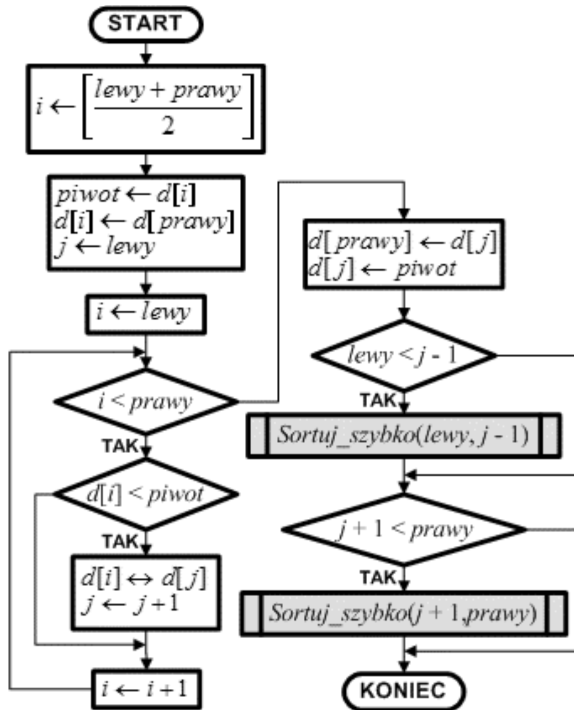
K08: **Jeśli** *j* + 1 < *prawy*, **to Sortuj_szybko**(*j* + 1, *prawy*)

K09: **Zakończ**

Algorytm sortowania szybkiego wywołujemy podając za $lewy$ indeks pierwszego elementu zbioru, a za $prawy$ indeks elementu ostatniego (czyli $Sortuj_szybko(1,n)$). Zakres indeksów jest dowolny - dzięki temu ten sam algorytm może również sortować fragment zbioru, co wykorzystujemy przy sortowaniu

wyliczonych partycji.

Schemat blokowy



Na element podziałowy wybieramy element leżący w środku dzielonej partycji. Wyliczamy jego pozycję i zapamiętujemy ją tymczasowo w zmiennej i. Robimy to po to, aby dwukrotnie nie wykonywać tych samych rachunków.

Element $d[i]$ zapamiętujemy w zmiennej piwot, a do $d[i]$ zapisujemy ostatni element partycji. Dzięki tej operacji *piwot* został usunięty ze zbioru.

Ustawiamy zmienną j na początek partycji. Zmienna ta zapamiętuje pozycję podziału partycji.

W pętli sterowanej zmienną i przeglądamy kolejne elementy od pierwszego do przedostatniego (ostatni został umieszczony na pozycji piwotu, a piwot zapamiętany). Jeśli i-ty element jest mniejszy od *piwotu*, to trafia on na początek partycji - wymieniamy ze sobą elementy na pozycjach i-tej i j-tej. Po tej operacji przesuwamy punkt podziałowy partycji j.

Po zakończeniu pętli element z pozycji j-tej przenosimy na koniec partycji, aby zwolnić miejsce dla *piwotu*, po czym wstawiamy tam *piwot*. Zmienna j wskazuje zatem wynikową pozycję *piwotu*. Pierwotna partycja została podzielona na dwie partycje:

partycja lewa od pozycji *lewy* do $j - 1$ zawiera elementy mniejsze od piwotu
partycja prawa od pozycji $j + 1$ do pozycji *prawy* zawiera elementy większe lub równe piwotowi.

Sprawdzamy, czy partycje te obejmują więcej niż jeden element. Jeśli tak, to wywołujemy rekurencyjnie algorytm sortowania szybkiego przekazując mu granice wyznaczonych partycji. Po powrocie z wywołań rekurencyjnych partycja wyjściowa jest posortowana rosnąco. Kończymy algorytm.

Programy

Efekt uruchomienia programu

```
   Sortowanie szybkie
------------------------
 (C)2005 Jerzy Walaszek

Przed sortowaniem:

  46   5  56  35  75  95  33  93   4  71   8   5  69  50  35  34  65  32  72  61

Po sortowaniu:

   4   5   5   8  32  33  34  35  35  46  50  56  61  65  69  71  72  75  93  95
```

DevPascal

```
// Sortowanie Szybkie
//--------------------------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// im. K. Brodzińskiego
// w Tarnowie
//--------------------------------------------------------

program Quick_Sort;

const N = 20; // Liczebność zbioru.

var
  d : array[1..N] of integer;

// Procedura sortowania szybkiego
//-------------------------------

procedure Sortuj_szybko(lewy, prawy : integer);
var
  i,j,piwot,x : integer;
begin
  i := (lewy + prawy) div 2;
  piwot := d[i]; d[i] := d[prawy];
  j := lewy;
  for i := lewy to prawy - 1 do
    if d[i] < piwot then
    begin
      x := d[i]; d[i] := d[j]; d[j] := x;
      inc(j);
    end;
  d[prawy] := d[j]; d[j] := piwot;
  if lewy < j - 1  then Sortuj_szybko(lewy, j - 1);
  if j + 1 < prawy then Sortuj_szybko(j + 1, prawy);
end;

// Program główny
//---------------

var
  i : integer;
begin
  writeln('   Sortowanie szybkie');
  writeln('------------------------');
  writeln(' (C)2005 Jerzy Walaszek ');
  writeln;

// Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi
// a następnie wyświetlamy jej zawartość

  randomize;
  for i := 1 to N do d[i] := random(100);
  writeln('Przed sortowaniem:'); writeln;
  for i := 1 to N do write(d[i] : 4);
  writeln;

// Sortujemy

  Sortuj_szybko(1,N);

// Wyświetlamy wynik sortowania

  writeln('Po sortowaniu:'); writeln;
```

	``
	  for i := 1 to N do write(d[i] : 4);
	  writeln;
	  writeln('Nacisnij Enter...');
	  readln;
	end.
	``

Code::Blocks	(see code below)

```
// Sortowanie Szybkie
//--------------------------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// im. K. Brodzińskiego
// w Tarnowie
//--------------------------------------------------------

#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <time.h>

using namespace std;

const int N = 20; // Liczebność zbioru.

int d[N];

// Procedura sortowania szybkiego
//-------------------------------

void Sortuj_szybko(int lewy, int prawy)
{
  int i,j,piwot;

  i = (lewy + prawy) / 2;
  piwot = d[i]; d[i] = d[prawy];
  for(j = i = lewy; i < prawy; i++)
  if(d[i] < piwot)
  {
    swap(d[i], d[j]);
    j++;
  }
  d[prawy] = d[j]; d[j] = piwot;
  if(lewy < j - 1)  Sortuj_szybko(lewy, j - 1);
  if(j + 1 < prawy) Sortuj_szybko(j + 1, prawy);
}

// Program główny
//---------------

int main()
{
  int i;

  srand((unsigned)time(NULL));

  cout << "   Sortowanie szybkie\n"
          "------------------------\n"
          " (C)2005 Jerzy Walaszek \n\n"
          "Przed sortowaniem:\n\n";

// Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi
// a następnie wyświetlamy jej zawartość

  for(i = 0; i < N; i++) d[i] = rand() % 100;
```

```
  for(i = 0; i < N; i++) cout << setw(4) << d[i];
  cout << endl;

// Sortujemy

  Sortuj_szybko(0,N - 1);

// Wyświetlamy wynik sortowania

  cout << "Po sortowaniu:\n\n";
  for(i = 0; i < N; i++) cout << setw(4) << d[i];
  cout << endl;
  return 0;
}
```

Free Basic

```
' Sortowanie szybkie
'--------------------------------------------------------
' (C)2012 I LO w Tarnowie
' I Liceum Ogólnokształcące
' im. K. Brodzińskiego
' w Tarnowie
'--------------------------------------------------------

Declare Sub Sortuj_szybko(lewy As Integer, prawy As Integer)

Const N = 20 ' liczebność zbioru
Dim Shared d(N) As Integer
Dim i As Integer

Print "  Sortowanie  szybkie"
Print "------------------------"
Print "(C)2005  Jerzy Walaszek"
Print
Print "Przed sortowaniem:": Print

' Wypełniamy tablicę liczbami pseudolosowymi i wyświetlamy je

Randomize
For i = 1 To N
  d(i) = Int(Rnd * 100): Print Using "####";d(i);
Next
Print

' Sortujemy

Sortuj_szybko(1,N)

' Wyświetlamy wynik sortowania

Print "Po sortowaniu:": Print
For i = 1 To N: Print Using "####";d(i);: Next
Print
Print "Nacisnij Enter..."
Sleep
End

' Procedura sortowania szybkiego
'--------------------------------

Sub Sortuj_szybko(lewy As Integer, prawy As Integer)

Dim As Integer i, j, piwot
```

```
  i = (lewy + prawy) \ 2
  piwot = d(i): d(i) = d(prawy)
  j = lewy
  For i = lewy To prawy - 1
    If d(i) < piwot Then
      Swap d(i), d(j)
      j += 1
    End If
  Next
  d(prawy) = d(j): d(j) = piwot
  If lewy < j - 1  Then Sortuj_szybko(lewy, j - 1)
  If j + 1 < prawy Then Sortuj_szybko(j + 1, prawy)

End Sub
```

JavaScript

```
<html>
  <head>
  </head>
  <body>
    <form style="BORDER-RIGHT: #ff9933 1px outset;
                 PADDING-RIGHT: 4px; BORDER-TOP: #ff9933 1px outset;
                 PADDING-LEFT: 4px; PADDING-BOTTOM: 1px;
                 BORDER-LEFT: #ff9933 1px outset; PADDING-TOP: 1px;
                 BORDER-BOTTOM: #ff9933 1px outset;
                 BACKGROUND-COLOR: #ffcc66" name="frmquicksort">
      <h3 style="text-align: center">Sortowanie Szybkie</h3>
      <p style="TEXT-ALIGN: center">
        (C)2012 I LO w Tarnowie - I LO w Tarnowie
      </p>
      <hr>
      <p style="TEXT-ALIGN: center">
        <input onclick="main()" type="button" value="Sortuj" name="B1">
      </p>
      <p id="t_out" style="TEXT-ALIGN: center">...</p>
    </form>

<script language=javascript>

// Sortowanie Szybkie
//-----------------------------------------------------------
// (C)2012 I LO w Tarnowie
// I Liceum Ogólnokształcące
// im. K. Brodzińskiego
// w Tarnowie
//-----------------------------------------------------------

var N = 20; // Liczebność zbioru.
var d = new Array(N)

// Procedura sortowania szybkiego
//-------------------------------

function Sortuj_szybko(lewy, prawy)
{
  var i,j,piwot,x;

  i = Math.floor((lewy + prawy) / 2);
  piwot = d[i]; d[i] = d[prawy];
  for(j = i = lewy; i < prawy; i++)
  if(d[i] < piwot)
  {
    x = d[i]; d[i] = d[j]; d[j] = x;
    j++;
```

```
  }
  d[prawy] = d[j]; d[j] = piwot;
  if(lewy < j - 1)  Sortuj_szybko(lewy, j - 1);
  if(j + 1 < prawy) Sortuj_szybko(j + 1, prawy);
}

// Program główny
//---------------

function main()
{
  var i,t;

// Najpierw wypełniamy tablicę d[] liczbami pseudolosowymi
// a następnie wyświetlamy jej zawartość

  for(i = 0; i < N; i++) d[i] = Math.floor(Math.random() * 100);
  t = "Przed sortowaniem:<BR><BR>";
  for(i = 0; i < N; i++) t += d[i] + " ";

// Sortujemy

  Sortuj_szybko(0, N - 1);

// Wyświetlamy wynik sortowania

  t += "<BR><BR>Po sortowaniu:<BR><BR>";
  for(i = 0; i < N; i++) t += d[i] + " ";
  document.getElementById("t_out").innerHTML = t
}

</script>

  </body>
</html>
```

Tutaj możesz przetestować działanie prezentowanego skryptu:

Sortowanie Szybkie

(C)2012 I LO w Tarnowie - I LO w Tarnowie

Sortuj

...

Badanie algorytmów sortowania

W celach badawczych testujemy czas wykonania algorytmu sortowania szybkiego w środowisku opisanym we wstępie. Program testujący jest następujący:

DevPascal

```
// Program testujący czas sortowania dla
// danego algorytmu sortującego
//--------------------------------------
// (C)2012 I LO w Tarnowie
```

```
// I Liceum Ogólnokształcące
// w Tarnowie
//------------------------------------------

program TestCzasuSortowania;

uses Windows;

const
  NAZWA = 'Sortowanie szybkie';
  K1    = '-------------------------------------------------------------';
  K2    = '(C)2011/2012 I Liceum Ogolnoksztalcace  w Tarnowie';
  K3    = '------n---------tpo---------tod---------tpp---------tpk---------tnp';
  K4    = '---------------------------------------------------------------------';
  MAX_LN = 8; // określa ostatnie LN
  LN : array[1..8] of integer = (1000,2000,4000,8000,16000,32000,64000,128000);

var
  d          : array[1..128000] of real; // sortowana tablica
  n          : integer;                  // liczba elementów
  qpf,tqpc   : int64;                    // dane dla pomiaru czasu
  qpc1,qpc2 : int64;

// Tutaj umieszczamy procedurę sortującą tablicę d
//-----------------------------------------------------------

procedure Sortuj_szybko(lewy, prawy : integer);
var
  i,j      : integer;
  piwot,x : real;
begin
  i := (lewy + prawy) div 2;
  piwot := d[i]; d[i] := d[prawy];
  j := lewy;
  for i := lewy to prawy - 1 do
    if d[i] < piwot then
    begin
      x := d[i]; d[i] := d[j]; d[j] := x;
      inc(j);
    end;
  d[prawy] := d[j]; d[j] := piwot;
  if lewy < j - 1  then Sortuj_szybko(lewy, j - 1);
  if j + 1 < prawy then Sortuj_szybko(j + 1, prawy);
end;

function Sort : extended;
begin
  QueryPerformanceCounter(addr(qpc1));
  Sortuj_szybko(1,n);
  QueryPerformanceCounter(addr(qpc2));
  Sort := (qpc2 - qpc1 - tqpc) / qpf;
end;

// Program główny
//-----------------
var
  i,j,k               : integer;
  tpo,tod,tpp,tpk,tnp : extended;
  f                   : Text;
begin
  if QueryPerformanceFrequency(addr(qpf)) then
  begin
    QueryPerformanceCounter(addr(qpc1));
    QueryPerformanceCounter(addr(qpc2));
```

```
    tqpc := qpc2 - qpc1;

    assignfile(f,'wyniki.txt'); rewrite(f);

// Wydruk na ekran

    writeln('Nazwa: ',NAZWA);
    writeln(K1);
    writeln(K2);
    writeln;
    writeln(K3);

// Wydruk do pliku

    writeln(f,'Nazwa: ',NAZWA);
    writeln(f,K1);
    writeln(f,K2);
    writeln(f,'');
    writeln(f,K3);
    for i := 1 to MAX_LN do
    begin
      n := LN[i];

// Czas sortowania zbioru posortowanego

      for j := 1 to n do d[j] := j;
      tpo := Sort;

// Czas sortowania zbioru posortowanego odwrotnie

      for j := 1 to n do d[j] := n - j;
      tod := Sort;

// Czas sortowania zbioru posortowanego
// z przypadkowym elementem na początku - średnia z 10 obiegów

      tpp := 0;
      for j := 1 to 10 do
      begin
        for k := 1 to n do d[k] := k;
        d[1] := random * n + 1;
        tpp += Sort;
      end;
      tpp /= 10;

// Czas sortowania zbioru posortowanego
// z przypadkowym elementem na końcu - średnia z 10 obiegów

      tpk := 0;
      for j := 1 to 10 do
      begin
        for k := 1 to n do d[k] := k;
        d[n] := random * n + 1;
        tpk += Sort;
      end;
      tpk /= 10;

// Czas sortowania zbioru nieuporządkowanego - średnia z 10 obiegów

      tnp := 0;
      for j := 1 to 10 do
      begin
        for k := 1 to n do d[k] := random;
        tnp += Sort;
```

```
      end;
      tnp /= 10;

      writeln(n:7,tpo:12:6,tod:12:6,tpp:12:6,tpk:12:6,tnp:12:6);
      writeln(f,n:7,tpo:12:6,tod:12:6,tpp:12:6,tpk:12:6,tnp:12:6);
    end;
    writeln(K4);
    writeln(f,K4);
    writeln(f,'Koniec');
    closefile(f);
    writeln;
    writeln('Koniec. Wyniki w pliku WYNIKI.TXT');
  end
  else writeln('Na tym komputerze program testowy nie pracuje !');
  writeln;
  write('Nacisnij klawisz ENTER...'); readln;
end.
```

Otrzymane wyniki są następujące (dla komputera o innych parametrach wyniki mogą się różnić co do wartości czasów wykonania, dlatego w celach porównawczych proponuję uruchomić podany program na komputerze czytelnika):

Zawartość pliku wygenerowanego przez program

```
Nazwa: Sortowanie szybkie
-----------------------------------------------------------
(C)2011/2012 I Liceum Ogolnoksztalcace  w Tarnowie

------n---------tpo---------tod---------tpp---------tpk---------tnp
   1000    0.000115    0.000118    0.000132    0.000136    0.000292
   2000    0.000222    0.000227    0.000270    0.000284    0.000616
   4000    0.000464    0.000479    0.000556    0.000569    0.001330
   8000    0.001058    0.001079    0.001216    0.001225    0.002825
  16000    0.002124    0.002185    0.002472    0.002549    0.006088
  32000    0.004593    0.004465    0.005211    0.005104    0.012763
  64000    0.009873    0.010195    0.010849    0.011224    0.027306
 128000    0.020037    0.021543    0.022796    0.022578    0.057542
-------------------------------------------------------------------
Koniec
```

Objaśnienia oznaczeń (wszystkie czasy podano w sekundach):

n - ilość elementów w sortowanym zbiorze

t_{po} - czas sortowania zbioru posortowanego

t_{od} - czas sortowania zbioru posortowanego malejąco

t_{pp} - czas sortowania zbioru posortowanego z losowym elementem na początku

t_{pk} - czas sortowania zbioru posortowanego z losowym elementem na końcu

t_{np} - czas sortowania zbioru z losowym rozkładem elementów

(Arkusz kalkulacyjny Excel do wyznaczania klasy czasowej złożoności obliczeniowej)

(Arkusz kalkulacyjny Excel do wyznaczania wzrostu prędkości sortowania)

Podsumowanie

Analizując wyniki obliczeń w arkuszu kalkulacyjnym otrzymanych czasów sortowania dla algorytmu sortowania szybkiego wyciągamy następujące wnioski:

Cechy Algorytmu Sortowania Szybkiego

klasa złożoności obliczeniowej optymistyczna	$O(n \log n)$
klasa złożoności obliczeniowej typowa	
klasa złożoności obliczeniowej pesymistyczna	$O(n^2)$
Sortowanie w miejscu	TAK
Stabilność	NIE

Klasy złożoności obliczeniowej szacujemy następująco:

- **optymistyczna** - dla zbiorów uporządkowanych (z niewielką liczbą elementów nie na swoich miejscach) - na podstawie czasów t_{po}, t_{pp}, t_{pk}
- **typowa** - dla zbiorów o losowym rozkładzie elementów - na podstawie czasu t_{np}
- **pesymistyczna** - dla zbiorów posortowanych odwrotnie - na podstawie czasu t_{od}. W przypadku tego algorytmu sortowania nie jest to przypadek pesymistyczny.

Własności algorytmu					
Algorytm	t_{po}	t_{od}	t_{pp}	t_{pk}	t_{np}
Sortowanie szybkie	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
	$t_{po} \approx t_{od}$		$t_{pp} \approx t_{pk}$		$t_{np} \approx \frac{8}{3} t_{od}$

1. Wszystkie otrzymane czasy sortowania są proporcjonalne do iloczynu $n \log_2 n$, wnioskujemy zatem, iż klasa złożoności obliczeniowej algorytmu sortowania szybkiego jest równa $O(n \log n)$.
2. Czasy sortowania dla poszczególnych przypadków są mniej więcej tego samego rzędu, zatem nie wystąpił tutaj przypadek pesymistyczny (zwróć uwagę na istotny fakt - to, co dla jednego algorytmu jest przypadkiem pesymistycznym, dla innego wcale nie musi takie być).
3. Czas sortowania zbiorów nieuporządkowanych jest wyraźnie dłuższy od czasów sortowania zbiorów uporządkowanych częściowo.
4. Czas sortowania zbioru uporządkowanego oraz uporządkowanego odwrotnie jest praktycznie taki sam.

Wzrost prędkości sortowania					
Algorytmy	t_{po}	t_{od}	t_{pp}	t_{pk}	t_{np}
Sortowanie przez scalanie Sortowanie szybkie	≈ 3	$\approx \frac{8}{3}$	$\approx \frac{5}{2}$	$\approx \frac{5}{2}$	≈ 1
	dobrze	**dobrze**	**dobrze**	**dobrze**	**brak**

5. Otrzymane wyniki potwierdzają, iż algorytm sortowania szybkiego jest najszybszym algorytmem sortującym. Jednakże w przypadku ogólnym notujemy jedynie bardzo nieznaczny wzrost prędkości sortowania w stosunku do algorytmu sortowania przez scalanie.
6. Ponieważ jak dotąd algorytm sortowania szybkiego jest najszybszym algorytmem sortującym, do dalszych porównań czasów sortowania zastosujemy czasy uzyskane w tym algorytmie.

Zadania dla ambitnych

1. Spróbuj znaleźć przypadek pesymistyczny dla algorytmu sortowania szybkiego opisanego w tym rozdziale.
2. Przebadaj algorytmy sortowania szybkiego, w których piwot wybierany jest:
   a. Na początku partycji
   b. Na końcu partycji

c. W miejscu losowym wewnątrz partycji

3. Dlaczego czasy sortowania zbioru uporządkowanego i uporządkowanego odwrotnie są prawie równe?
4. Uzasadnij, iż algorytm sortowania szybkiego nie posiada cechy stabilności.
5. Wyszukaj w Internecie informację na temat algorytmu Introsort.

List do administratora Serwisu Edukacyjnego I LO

Twój email: (jeśli chcesz otrzymać odpowiedź)

Temat:

Uwaga: ← **tutaj wpisz wyraz ilo , inaczej list zostanie zignorowany**

Poniżej wpisz swoje uwagi lub pytania dotyczące tego rozdziału (max. 2048 znaków).

Liczba znaków do wykorzystania: 2048

Wyślij Kasuj

W związku z dużą liczbą listów do naszego serwisu edukacyjnego nie będziemy udzielać odpowiedzi na prośby rozwiązywania zadań, pisania programów zaliczeniowych, przesyłania materiałów czy też tłumaczenia zagadnień szeroko opisywanych w podręcznikach.

Dokument ten rozpowszechniany jest zgodnie z zasadami licencji **GNU Free Documentation License**.

I Liceum Ogólnokształcące
im. Kazimierza Brodzińskiego
w Tarnowie
(C)2013 mgr Jerzy Wałaszek