Porządkowanie ciągu elementów. Sortowanie (http://pl.wikipedia.org/wiki/Sortowanie)

Porządkowanie jest jednym z najważniejszych i najczęściej wykonywanych przez komputery zadań. Jeśli elementy w zbiorze są uporządkowane zgodnie z jakąś regułą (np. książki lub ich karty katalogowe według liter alfabetu, słowa w encyklopedii, daty, kredki według kolorów, czy osoby według wzrostu), to wykonanie wielu operacji na tym zbiorze staje się łatwiejsze i szybsze. Dotyczy to zwłaszcza: sprawdzenia, czy dany element znajduje się w zbiorze i znalezienia go, dołączenia nowego elementu w odpowiednie miejsce, aby zbiór pozostał nadal uporządkowany.

Porządkowanie dwóch liczb.

Aby uporządkować dwie liczby wystarczy wykonać jedno porównanie i jeśli jest to konieczne, należy zamienić te liczby miejscami, wykorzystując dodatkową zmienną. W przykładzie sprawdzamy czy zmienna **a** jest mniejsza od zmiennej **b**. Jeśli tak, to do zmiennej **a** zostanie wstawiona wartość ze zmiennej **b**, a do zmiennej **b**, wartość ze zmiennej **a**. Komputer niestety nie potrafi za pomocą "hokuspokus" zamienić miejscami tych dwóch komórek - konieczna jest dodatkowa pomoc w postaci trzeciej zmiennej (w naszym przykładzie **c**.

```
if a < b then
begin
      c:=a;
      a:=b;
      b:=c
end;
```

Porządkowanie trzech liczb.

Sposób porządkowania trzech liczb jest już bardziej skomplikowany i wymaga trzech porównać i ewentualnych zamian. Pokazuje to fragment programu:

```
if a < b then
  begin x:=a; a:=b; b:=x end;
if b < c then
  begin x:=b; b:=c; c:=x end;
if a < b then
  begin x:=a; a:=b; b:=x end;
```

Wyszukiwanie maksimum (minimum)

Jest często wykorzystywane podczas sortowania. W jednym przebiegu pętli można wyszukać element maksymalny (lub minimalny) albo oba jednocześnie.

```
m:=1;
FOR i:=2 TO ile do
  IF T[i] < T[m] THEN m:=I;
minimum:=T[m];
```

```
m:=1;
FOR i:=2 TO ile do
  IF T[i] > T[m] THEN m:=I;
maksimum:=T[m];
```

W zmiennej M nie jest pamiętany element maksymalny (minimalny), ale indeks tablicy, w którym ten element się znajduje - początkowo przyjmuje my, że będzie to pierwszy element tablicy. Sprawdzamy kolejne elementy tablicy i jeśli kolejny jest większy (mniejszy) od maksymalnego (minimalnego) - zapamiętujemy w zmiennej M jego indeks.

Sortowanie

Porządkowanie większej ilości liczb odbywa się już metodami ogólniejszymi. Istnieje wiele sposobów sortowania, w zależności od sytuacji jedne okazują się lepsze - szybsze od innych. Najważniejszym parametrem jest tutaj czas w jakim zostaną uporządkowane informacje. Czas ten zależny jest głównie od liczby porównań elementów, jakie trzeba zrealizować by uporządkować zbiór. Porównanie zajmuje procesorowi najwięcej czasu dlatego wprowadzono takie kryterium, ale też trzeba mieć świadomość, że czynności dodatkowe (jak wyszukanie minimalnego, zamiana, elementów, czy podział tablicy) mogą być czasochłonne.

Sortowanie naiwne - wyznaczany jest największy (najmniejszy) element zbioru i przenoszony na początek (koniec) zbioru. Liczba porównań **n(n-1)/2**

Sortowanie bąbelkowe - zamieniamy miejscami dwa elementy, które są nieuporządkowane i tak w koło do uporządkowania. Liczba porównań **n(n-1)/2**

Podane na końcu przykłady na sortowanie naiwne i bąbelkowe są najprostsze, bez dodatkowych usprawnień. Na przykład w sortowaniu bąbelkowym wystarczy zamienić

```
for i:=1 to ile-1 do          na
for i:=1 to ile-j do          aby liczbę porównań zmniejszyć o połowę
```

Bo skoro ostatni element jest już posortowany, to nie potrzeba go więcej sprawdzać.

Inne sposoby sortowania

Sortowanie przez scalanie - Zbiór elementów jest dzielony na coraz mniejsze fragmenty, aż do pojedynczych, które następnie są porządkowane i scalane (rekurencyjnie). Liczba porównań $\mathbf{n\log_2 n - n + 1}$

Sortowanie kubełkowe (koszykowe) - podobne do sposobu sortowania listów na poczcie, wybieramy list i wrzucamy do odpowiedniej przegródki, szczególnie nadaje się do porządkowania wyrazów, dat.

Porządkowanie przez wstawianie - wykorzystywany gdy elementy do uporządkowania napływają kolejno, np. jak w kartach, bierzemy kolejną kartę i wstawiamy w odpowiednie miejsce

Porządkowanie QuickSort (szybkie) - działa w oparciu o zasadę dziel i zwyciężaj (zamiast rozwiązywać skomplikowany problem, lepiej go podzielić na mniejsze-prostsze), dzielimy ciąg elementów na dwa, jeden złożony z elementów większych od pewnej liczby, a drugi z liczb mniejszych. Każdy z tych ciągów podobnie dzielimy, a następnie tak podzielone ciągi porządkujemy i w końcu łączymy w całość. Liczba porównań $\mathbf{n^2}$

SORTOWANIE QUICKSORT (http://pl.wikipedia.org/wiki/Quicksort#Pascal)

Jest obecnie najczęściej stosowanym algorytmem - szybki i stosunkowo nieskomplikowany. Jest to algorytm rekurencyjny (procedura wywołuje samą siebie), dlatego też nie przebiega liniowo, jak poprzednio naiwny, czy bąbelkowy, lecz na wielu poziomach i trudno jest prześledzić kolejne wartości zmiennych i tablicy.

W największym skrócie algorytm przebiega w następujący sposób:
- tablica dzielona jest na dwie części, te dwie na kolejne… i tak aż do wyczerpania się możliwości podziałów - zostaje po dwa elementy
- w tych najmniejszych tablicach elementy są porządkowane
- dwie najmniejsze tablice łączone są w większą i tam znów następuje posortowanie… i tak aż do ponownego połączenia się w oryginalną tablicę

(dla 20 elementów - 40 porównań)

```
PROCEDURE Quicksort (VAR A : tab; l,r: INTEGER);
VAR
    pivot,b,i,j :  INTEGER;
BEGIN
    IF l < r THEN
    BEGIN
        pivot := A[random(r-l) + l+1];  {losowanie elementu dzielącego}
        i := l-1;
        j := r+1;
        REPEAT
            REPEAT i := i+1 UNTIL pivot <= A[i];
            REPEAT j := j-1 UNTIL pivot >= A[j];
            b:=A[i]; A[i]:=A[j]; A[j]:=b
        UNTIL i >= j;
        A[j]:=A[i]; A[i]:=b;
        Quicksort(A,l,i-1);
        Quicksort(A,i,r)
    END
END;
```

SORTOWANIE NAIWNE (dla 20 elementów - 209 porównań) **SORT_NAI.PAS**

```
var
  T:array[1..100] of byte;
  i,j,min,poz,tym,ile:byte;

begin
  ile:=20;
  {losowanie}
  for i:=1 to ile do T[i]:=random(100);
  for i:=1 to ile do write(T[i]:3);
  writeln;

  {sortowanie naiwne}
  for j:=1 to ile-1 do
  begin
    {szukanie najmniejszego}
    min:=T[j];
    for i:=j to ile do
      if T[i]<min then
      begin
        min:=T[i];
        poz:=i
      end;

    {zamiana elementow}
    tym:=T[j];
    T[j]:=T[poz];
    T[poz]:=tym;
  end;
  for i:=1 to ile do write(T[i]:3);
  readln;
end.
```

- losowanie do tablicy
- wypisanie liczb na ekranie

- pętla zewnętrzna ustawia kolejny element, który szukamy

- najmniejszy na początku
- sprawdzamy wszystkie następne
- jeśli jest mniejszy to

  znaleziona wartość staje się najmniejsza
  zapamiętujemy pozycję tego najmniejszego

- zamieniamy miejscami elementy w tablicy
  kolejny (j) z tym, w którym znaleziono
  najmniejszą wartość (poz)

- wypisanie liczb na ekranie

SORTOWANIE BĄBELKOWE **SORT_BAB.PAS**
(http://pl.wikipedia.org/wiki/Sortowanie_b%C4%85belkowe)
(dla 20 elementów - 380 porównań, po modyfikacji - 190)

```
var
  T:array[1..100] of byte;
  i,j,tym,ile:byte;

begin
  ile:=20;
  {losowanie}
  for i:=1 to ile do T[i]:=random(100);
  for i:=1 to ile do write(T[i]:3);
  writeln;

  {sortowanie babelkowe}
  for j:=1 to ile do
  for i:=1 to ile-1 do
    if T[i] > T[i+1] then
    begin
      {zamiana elementow}
      tym:=T[i+1];
      T[i+1]:=T[i];
      T[i]:=tym;
    end;

  for i:=1 to ile do write(T[i]:3);
  readln;
end.
```

- losowanie do tablicy
- wypisanie liczb na ekranie

- pętla zewnętrzna - tyle razy szukamy
- pętla wewnętrzna - który element
- porównujemy dwa kolejne

- zamieniamy jeśli następny mniejszy

- wypisanie liczb na ekranie