Algorytmy sortowania:

Sortowanie bąbelkowe (ang. bubble sort) - prosta metoda sortowania o złożoności czasowej O(n2) i pamięciowej O(1).

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

Przykład działania: Ciąg wejściowy [4,2,5,1,7]. Każdy wiersz symbolizuje wypchnięcie kolejnego największego elementu na koniec ("wypłynięcie największego bąbelka"). Niebieskim kolorem oznaczono końcówkę ciągu już posortowanego.

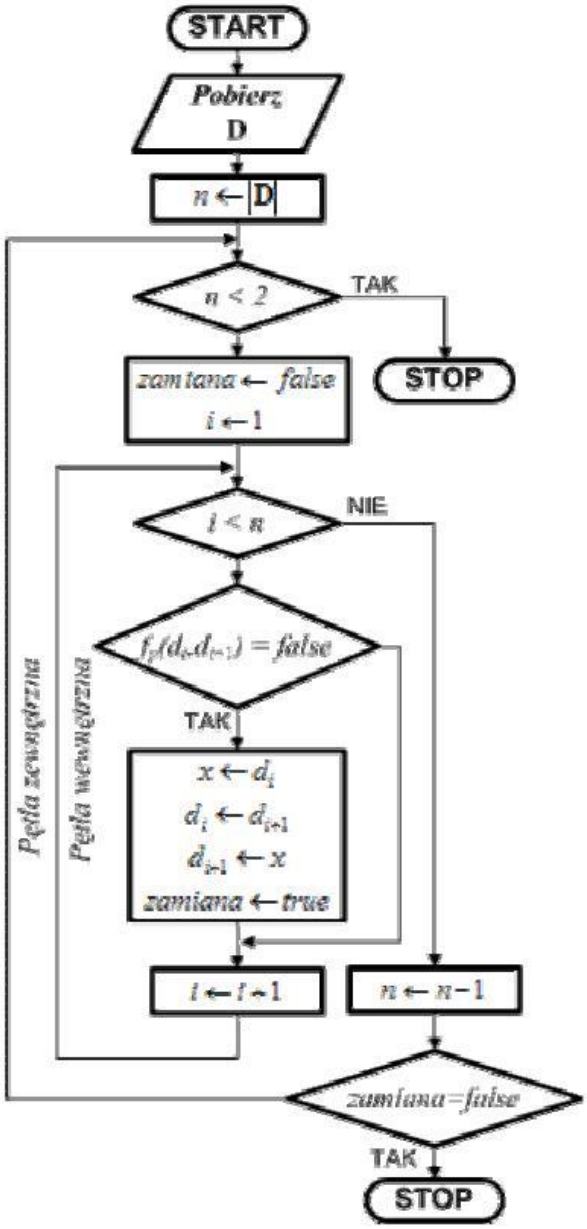
$$[\underbrace{4,2}_{4>2},5,1,7] \rightarrow [2,\underbrace{4,5}_{4<5},1,7] \rightarrow [2,4,\underbrace{5,1}_{5>1},7] \rightarrow [2,4,1,\underbrace{5,7}_{5<7}]$$
$$[\underbrace{2,4}_{2<4},1,5,7] \rightarrow [2,\underbrace{4,1}_{4>1},5,7] \rightarrow [2,1,\underbrace{4,5}_{4<5},7]$$
$$[\underbrace{2,1}_{2>1},4,5,7] \rightarrow [1,\underbrace{2,4}_{2<4},5,7]$$
$$[\underbrace{1,2}_{1<2},4,5,7]$$

Pseudokod: Pseudokod algorytmu dla tablicy o rozmiarze "r" (elementy tablicy są numerowane od 0 do r-1):

```
sortuj(tablica tab)
  for i=0 to r-2 do
     for j=r-1 downto i+1 do
        if (tab[j-1]>tab[j])
           zamień elementy tab[j] i tab[j-1]
```

Algorytm bąbelkowy – schemat blokowy:

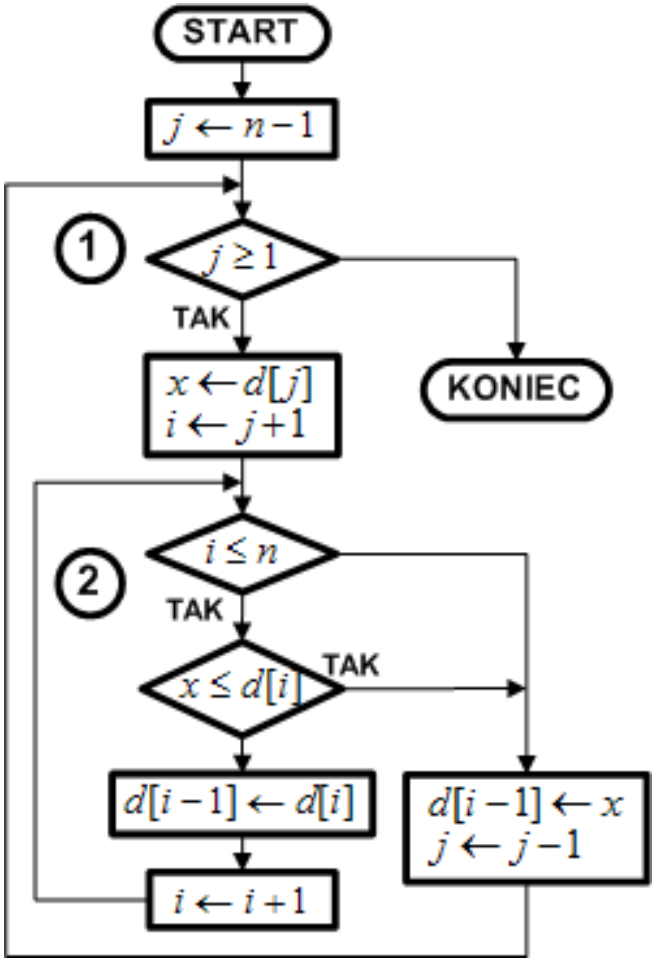
Schemat blokowy algorytmu sortowania bąbelk



D	- zbiór elementów do posortowania
n	- liczba elementów w zbiorze D
$zamiana$	- zmienna logiczna określająca, czy w danym obiegu sortującym zamieniano miejscami jakieś elementy zbioru D
i	- indeks przeglądanych elementów
$f_p( )$	- funkcja porównująca
x	- zmienna pomocnicza, wykorzystywana przy wymianie zawartości elementów
d_i	- i-ty element zbioru D

Sortowanie przez wstawianie (ang. Insert Sort, Insertion Sort) - jeden z najprostszych algorytm sortowania, którego zasada działania odzwierciedla sposób w jaki ludzie ustawiają karty - kolejne element wejściowe są ustawiane na odpowiednie miejsca docelowe. Jest efektywny dla niewielkiej liczby elementów, jego złożoność wynosi O(n2)[1]. Pomimo tego, że jest znacznie mniej wydajny od algorytmów takich jak quicksort czy heapsort, posiada pewne zalety:

* jest wydajny dla danych wstępnie posortowanych
* jest wydajny dla zbiorów o niewielkiej liczebności
* jest stabilny



Specyfikacja problemu

Dane wejściowe

n - liczba elementów w sortowanym zbiorze, n Î N

d[ ] - zbiór n-elementowy, który będzie sortowany. Elementy zbioru mają indeksy od 1 do n.

Dane wyjściowe

d[ ] - posortowany zbiór n-elementowy. Elementy zbioru mają indeksy od 1 do n.

Zmienne pomocnicze

i - indeks środkowego elementu listy uporządkowanej, i Î N

j - licznik obiegów pętli zewnętrznej, indeks wybranego elementu, j Î N

ip, ik - początkowa i końcowa pozycja indeksów w partycji listy uporządkowanej, ip, ik Î N

x - zawiera wybrany ze zbioru element

Lista kroków

K01: Dla j = n - 1, n - 2,...,1: wykonuj K02...K07
K02: x ← d[j]; ip ← j; ik ← n + 1
K03: Dopóki ik - ip > 1: wykonuj K04...K05
K04: i ← (ip + ik) div 2
K05: Jeśli x ≤ d[i], to ik ← i
inaczej ip ← i
K06: Dla i = j, j + 1, ..., ip - 1: wykonuj d[i] ← d[i + 1]
K07: d[ip] ← x
K08: Zakończ

Sortowanie przez scalanie (ang. merge sort), to rekurencyjny algorytm sortowania danych, mający zastosowanie przy danych dostępnych sekwencyjnie (po kolei, jeden element na raz), na przykład w postaci listy jednokierunkowej (tj. łączonej jednostronnie) albo pliku sekwencyjnego. Odkrycie algorytmu przypisuje się Johnowi von Neumannowi.

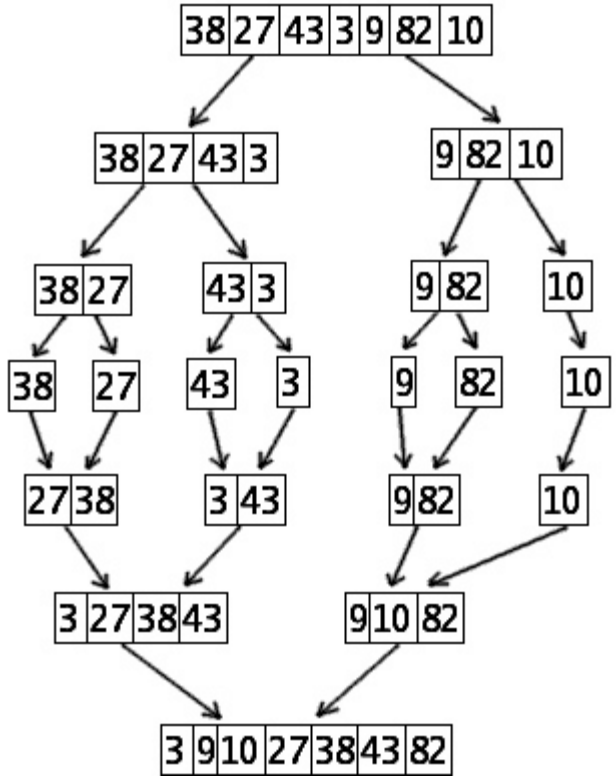
Algorytm ten jest dobrym przykładem algorytmów typu Dziel i zwyciężaj (ang. divide and conquer), których ideą działania jest podział problemu na mniejsze części, których rozwiązanie jest już łatwiejsze.

Wyróżnić można trzy podstawowe kroki:

1. Podziel zestaw danych na dwie, równe części (w przypadku nieparzystej liczby wyrazów jedna część będzie o 1 wyraz dłuższa);
2. Zastosuj sortowanie przez scalanie dla każdej z nich oddzielnie, chyba że pozostał już tylko jeden element;
3. Połącz posortowane podciągi w jeden.

Procedura scalania dwóch ciągów A[1..n] i B[1..m] do ciągu C[1..m+n]:

1. Utwórz wskaźniki na początki ciągów A i B → i=1, j=1
2. Jeżeli ciąg A wyczerpany (i>n), dołącz pozostałe elementy ciągu B do C i zakończ pracę.
3. Jeżeli ciąg B wyczerpany (j>m), dołącz pozostałe elementy ciągu A do C i zakończ pracę.
4. Jeżeli A[i] ≤ B[j] dołącz A[i] do C i zwiększ i o jeden, w przeciwnym przypadku dołącz B[j] do C i zwiększ j o jeden
5. Powtarzaj od kroku 2 aż wszystkie wyrazy A i B trafią do C



Rysunek 1: przykład

LINKI:
Link do WWW z wieloma innymi algorytmami sortującymi i wyszukującymi: http://edu.i-lo.tarnow.pl/inf/alg/003_sort/index.php