# Lekcja: Algorytmy sortowania

## Sortowanie bąbelkowe

Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących. Można go potraktować jako ulepszenie opisanego w poprzednim rozdziale algorytmu sortowania głupiego. Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Algorytm sortowania bąbelkowego przy porządkowaniu zbioru nieposortowanego ma klasę czasowej złożoności obliczeniowej równą O(n2). Sortowanie odbywa się w miejscu.

Przykład:

Jako przykład działania algorytmu sortowania bąbelkowego posortujemy przy jego pomocy 5-cio elementowy zbiór liczb {5 4 3 2 1}, który wstępnie jest posortowany w kierunku odwrotnym, co możemy uznać za przypadek najbardziej niekorzystny, ponieważ wymaga przestawienia wszystkich elementów.

|  |  |  |
| --- | --- | --- |
| **Obieg** | **Zbiór** | **Opis operacji** |
| 1 | **5 4** 3 2 1 | Rozpoczynamy od pierwszej pary, która wymaga wymiany elementów |
| 4 **5 3** 2 1 | Druga para też wymaga zamiany elementów |
| 4 3 **5 2** 1 | Wymagana wymiana elementów |
| 4 3 2 **5 1** | Ostatnia para również wymaga wymiany elementów |
| 4 3 2 **1 5** | Stan po pierwszym obiegu. Zwróć uwagę, iż najstarszy element **(5)** znalazł się na końcu zbioru, a najmłodszy **(1)** przesunął się o jedną pozycję w lewo. |
| 2 | **4 3** 2 1 5 | Para wymaga wymiany |
| 3 **4 2** 1 5 | Para wymaga wymiany |
| 3 2 **4 1** 5 | Para wymaga wymiany |
| 3 2 1 **4 5** | Elementy są w dobrej kolejności, zamiana nie jest konieczna. |
| 3 2 **1** 4 5 | Stan po drugim obiegu. Zwróć uwagę, iż najmniejszy element **(1)**znów przesunął się o jedną pozycję w lewo. Z obserwacji tych można wywnioskować, iż po każdym obiegu najmniejszy element wędruje o jedną pozycję ku początkowi zbioru. Najstarszy element zajmuje natomiast swe miejsce końcowe. |
| 3 | **3 2** 1 4 5 | Para wymaga wymiany |
| 2 **3 1** 4 5 | Para wymaga wymiany |
| 2 1 **3 4** 5 | Dobra kolejność |
| 2 1 3 **4 5** | Dobra kolejność |
| **2 1 3 4 5** | Stan po trzecim obiegu. Wnioski te same. |
| 4 | **2 1** 3 4 5 | Para wymaga wymiany |
| 1 **2 3** 4 5 | Dobra kolejność |
| 1 2 **3 4** 5 | Dobra kolejność |
| 1 2 3 **4 5** | Dobra kolejność |
| **1 2 3 4 5** | Zbiór jest posortowany. Koniec |

**Dane wejściowe**

n - liczba elementów w sortowanym zbiorze, n ∈ 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, j - zmienne sterujące pętli, i, j ∈ N

**Lista kroków**

K01: Dla j = 1,2,...,n - 1:

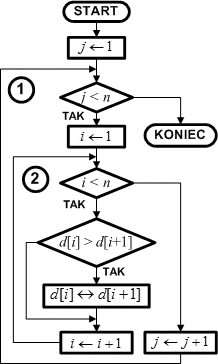
**Wykonuj krok K02**

K02: Dla i = 1,2,...,n - 1:

**Jeśli** d[i] > d[i + 1],

**to** d[i] ↔ d[i + 1]

K03: **Zakończ**



Sortowanie wykonywane jest w dwóch zagnieżdżonych pętlach. Pętla zewnętrzna nr 1 kontrolowana jest przez zmienną j. Wykonuje się ona n - 1 razy. Wewnątrz pętli nr 1 umieszczona jest pętla nr 2 sterowana przez zmienną i. Wykonuje się również n - 1 razy. W efekcie algorytm wykonuje w sumie:



obiegów pętli wewnętrznej, po których zakończeniu zbiór zostanie posortowany.

## Sortowanie przez wybór

Idea algorytmu sortowania przez wybór jest bardzo prosta. Załóżmy, iż chcemy posortować zbiór liczbowy rosnąco. Zatem element najmniejszy powinien znaleźć się na pierwszej pozycji. Szukamy w zbiorze elementu najmniejszego i wymieniamy go z elementem na pierwszej pozycji. W ten sposób element najmniejszy znajdzie się na swojej docelowej pozycji.

W identyczny sposób postępujemy z resztą elementów należących do zbioru. Znów wyszukujemy element najmniejszy i zamieniamy go z elementem na drugiej pozycji. Otrzymamy dwa posortowane elementy. Procedurę kontynuujemy dla pozostałych elementów dotąd, aż wszystkie będą posortowane.

Algorytm sortowania przez wybór posiada klasę czasowej złożoności obliczeniowej równą O(n2). Sortowanie odbywa się w miejscu.

Przykład:

Dla przykładu posortujmy tą metodą zbiór {4 7 2 9 3}. Kolorem zielonym oznaczyliśmy elementy zbioru, które są już posortowane.

|  |  |
| --- | --- |
| **Zbiór** | **Opis operacji** |
| **4 7 2 9 3** | Wyszukujemy najmniejszy element w zbiorze. Jest nim liczba 2. |
| **2 7 4 9 3** | Znaleziony element minimalny wymieniamy z pierwszym elementem zbioru - liczbą 4 |
| **2 7 4 9 3** | Wśród pozostałych elementów wyszukujemy element najmniejszy. Jest nim liczba 3. |
| **2 3 4 9 7** | Znaleziony element minimalny wymieniamy z drugim elementem zbioru - liczbą 7. |
| **2 3 4 9 7** | Znajdujemy kolejny element minimalny - liczbę 4. |
| **2 3 4 9 7** | Wymieniamy go z samym sobą - element ten nie zmienia zatem swojej pozycji w zbiorze. |
| **2 3 4 9 7** | Znajdujemy kolejny element minimalny |
| **2 3 4 7 9** | Wymieniamy go z liczbą 9 |
| **2 3 4 7 9** | Ostatni element jest zawsze na właściwej pozycji. Sortowanie zakończone |

Podana metoda sortuje zbiór rosnąco. Jeśli chcemy posortować zbiór malejąco, to zamiast elementu minimalnego poszukujemy elementu maksymalnego. Pozostała część procedury sortującej nie ulega zmianie.

**Dane wejściowe**

n - liczba elementów w sortowanym zbiorze, n ∈ 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, j - zmienne sterujące pętli, i, j ∈ N

pmin - pozycja elementu minimalnego w zbiorze d[ ], pmin ∈ N

**Lista kroków**

K01: **Dla** j = 1, 2, ..., n - 1:

**wykonuj kroki K02...K04**

K02: pmin ← j

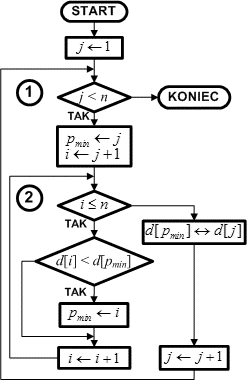
K03: Dla i = j + 1, j + 2, ..., n:

**jeśli** d[i] < d[pmin],

**to** pmin ← i

K04: d[j] ↔ d[pmin]

K05: **Zakończ**



## Zadanie 1

Zaimplementuj algorytm sortowania bąbelkowego.

Czy widzisz jakieś możliwości jego optymalizacji? Jeśli tak, to jakie?

## Zadanie 2

Zaimplementuj algorytm sortowania przez wybór

## Zadanie 3

Załóżmy, że masz tablicę liczb całkowitych o dowolnej długości. Liczby są zawsze z zakresu od 0 do 4. Napisz metodę, która posortuje wspomnianą tablicę. Czy możemy tutaj osiągnąć lepszą złożoność niż O(n^2)?