Algorytmy sortowania (ang. sorting algorithms)

Sortowanie (ang. *sorting*) – uporządkowanie danych lub informacji (kart bibliotecznych, słów w encyklopedii, dat urodzin, numerów telefonów itp.) poprzez ustawienie ich w określonej kolejności (najczęściej rosnąco lub malejąco).

Dzięki posortowaniu operacje na uporządkowanym zbiorze stają się łatwiejsze i szybsze. Dotyczy to m.in.:

- sprawdzenia, czy element o ustalonej wartości cechy znajduje się w zbiorze,
- znalezienia elementu w zbiorze, jeśli w nim jest,
- dołączenia nowego elementu w odpowiednie miejsce, aby zbiór pozostał nadal uporządkowany.

Istnieje wiele metod sortowania, wśród nich najpopularniejsze to:

1. Bąbelkowe (*bubble sort*) – stabilne
2. Przez wstawianie (*insertion sort*) – stabilne
3. Przez wybór (*selection sort*) – niestabilne
4. Przez scalanie (*merge sort*) – stabilne
5. Szybkie (*quick sort*) – niestabilne
6. Stogowe (*heap sort*) – niestabilne

Sortowanie bąbelkowe (ang. *bubble sort*)

Algorytm sortowania bąbelkowego polega na tym, że porównywane są elementy leżące obok siebie. Jeśli są w niewłaściwej kolejności, zamieniane są miejscami. W pierwszym przebiegu tego algorytmu na swoje miejsce trafia ostatni, n-ty element, w drugim (n – 1), w trzecim (n – 2) itd. Porównywać można element pierwszy z drugim lub przedostatni z ostatnim. Zatem ciąg elementów do sortowania skracany jest o jeden w każdym przebiegu. Złożoność obliczeniowa tego algorytmu jest rzędu $O(n^2)$, a sortowanie odbywa się w miejscu (łac. *in situ*), tj. w tej samej tablicy, a więc bez używania dodatkowych struktur.

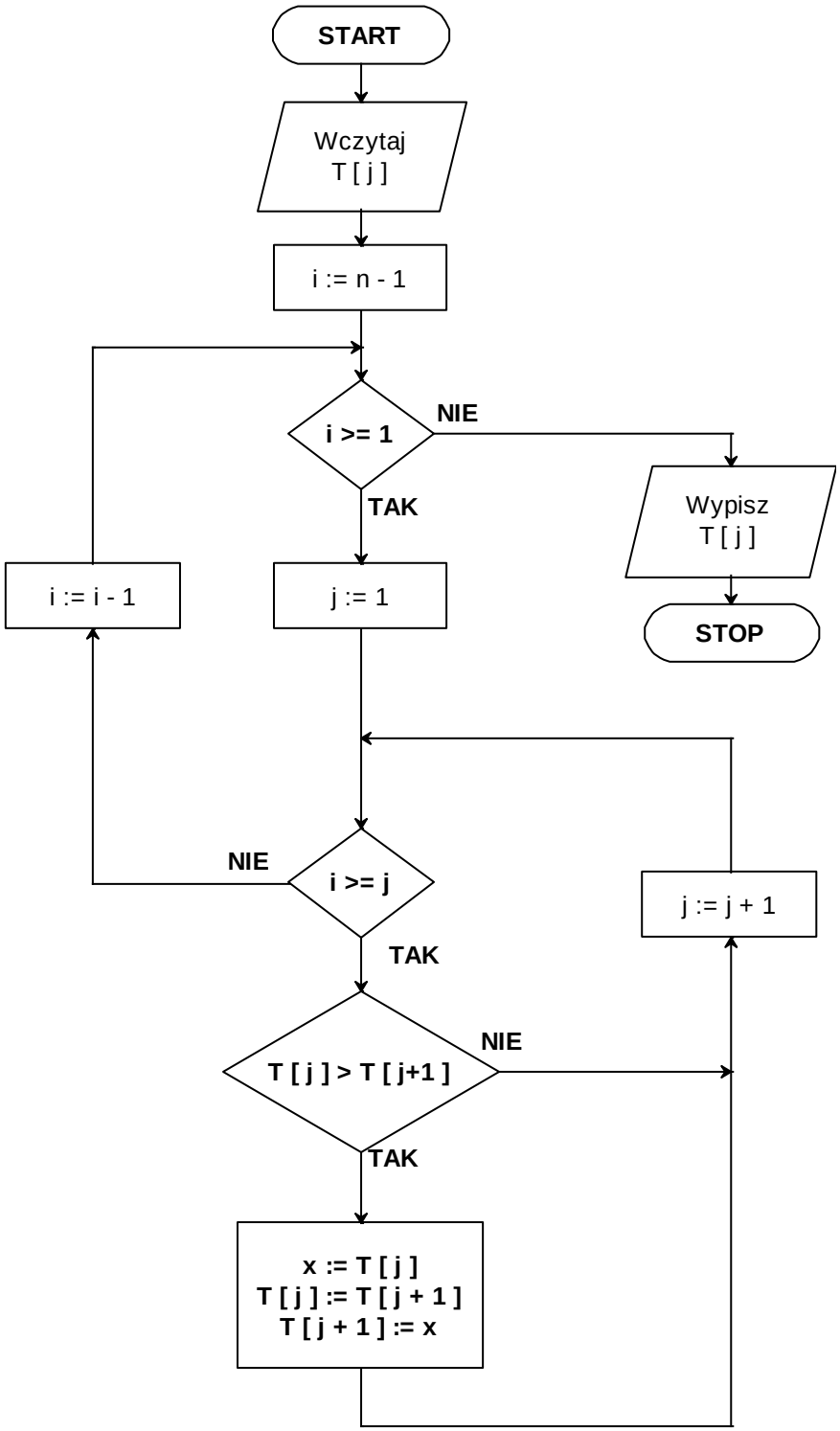
Przykład
Posortować rosnąco zbiór trzech liczb {3, 2, 1}.

Numer przebiegu	Zbiór	Uwagi
1	3, 2, 1	Porównanie i zamiana elementów pierwszej pary {3, 2}
	2, 3, 1	Porównanie i zamiana elementów drugiej pary {3, 1}
	2, 1, 3	Zakończenie pierwszego przebiegu
2	2, 1, 3	Porównanie i zamiana elementów pierwszej pary {2, 1}
	1, 2, 3	Porównanie elementów drugiej pary {2, 3}
	1, 2, 3	Zakończenie drugiego przebiegu
3	1, 2, 3	Porównanie elementów pierwszej pary {1, 2}
	1, 2, 3	Porównanie elementów drugiej pary {2, 3}
	1, 2, 3	Zakończenie trzeciego przebiegu. Zbiór posortowany

Specyfikacja
Wejście:
T[j] – *j-ty* element tablicy **T** złożonej z **n** przypadkowych liczb całkowitych
i, j – zmienne licznikowe pętli (liczby całkowite dodatnie)

Wyjście:
T[j] – *j-ty* element tablicy **T** złożonej z **n** liczb całkowitych uporządkowanych rosnąco

Schemat blokowy algorytmu sortowania bąbelkowego



Uwagi:

1. W pętli górnej (zewnętrznej) pierwszy obieg ma numer **n-1**, następny **n-2** itd. Dzięki temu przedostatni element skracanej pętli porównywany jest z ostatnim.

2. W pętli dolnej (wewnętrznej) sprawdzamy, czy elementy **T[j]** i **T [j+1]** są we właściwej kolejności. Jeśli nie, zamieniamy je z sobą.
3. Jest to jedna z wielu postaci algorytmu sortowania bąbelkowego.

Zadanie domowe 1. (nadobowiązkowe)
Na podstawie powyższego opisu algorytmu sortowania bąbelkowego napisz program w języku C# (C++, Python, Java).

Sortowanie przez wstawianie (ang. *insertion sort*)

Algorytm tego sortowania jest stabilny i bardzo efektywny dla niewielkiej liczby elementów. Jego działanie przypomina sposób układania przez ludzi talii kart w ręce. Pierwsza karta trafia ze stołu do pustej lewej ręki, a każda następna pobrana ze stołu wkładana jest we właściwe miejsce. Aby znaleźć to miejsce karta porównywana jest z kartami, które są już w ręce. Po dojściu do tego miejsca karty w lewej ręce są rozsuwane, by zrobić miejsce dla nowej, która tam trafia.

Algorytm ma więc strukturę listy uporządkowanej. Na początku lista zawiera tylko jeden, ostatni element zbioru (lista jednoelementowa jest zawsze uporządkowana). Jeśli przy przeglądaniu listy algorytm trafia na koniec listy, wówczas lista rozrasta się o nowy element. Algorytm jest rzędu $O(n^2)$, a sortowanie odbywa się w miejscu, w tej samej tablicy (*in situ*).

Przykład
Uporządkować metodą *insertion-sort* zbiór liczb {7, 5, 2, 4}

Zbiór	Działanie
7, 5, 2, **4**	Ostatni element (**4**) stanowi jednoelementową listę uporządkowaną
7, 5, □, **4**	Ze zbioru wybieramy do analizy element leżący tuż przed listą, czyli **2**. Jego miejsce pozostawiamy puste.
7, 5, **2**, **4**	Ponieważ **2** jest mniejsze niż **4**, wstawiamy **2** w puste miejsce. Lista uporządkowana zawiera dwa elementy.
7, □, **2**, **4**	Bierzemy do analizy kolejny element sprzed listy, czyli **5**.
7, **2**, □, **4**	Ponieważ **5** jest większe niż **2**, wstawiamy **2** w puste miejsce (puste miejsce przesuwa się w prawo).
7, **2**, **4**, □	Ponieważ **5** jest większe niż **4**, wstawiamy **4** w puste miejsce.
7, **2**, **4**, **5**	Puste miejsce dotarło do końca. Lista nie zawiera więcej elementów, zatem liczba **5** trafiła na swoje miejsce. Lista uporządkowana zawiera trzy elementy.
□, **2**, **4**, **5**	Bierzemy do analizy liczbę **7**.
2, □, **4**, **5**	Porównujemy **7** z **2** i przesuwamy **2** na puste miejsce.
2, **4**, □, **5**	Porównujemy **7** z **4** i przesuwamy **4** na puste miejsce.
2, **4**, **5**, □	Porównujemy **7** z **5** i przesuwamy **5** na puste miejsce.
2, **4**, **5**, **7**	Lista nie zawiera więcej elementów, zatem liczba **7** trafiła na swoje miejsce. Ponieważ nie ma już elementów przed listą, sortowanie zostało zakończone.

Specyfikacja
Wejście:
T[j] – *j-ty* element tablicy **T** złożonej z **n** przypadkowych liczb całkowitych
i, j – zmienne licznikowe pętli (liczby całkowite dodatnie)
ile – liczebność zbioru przeznaczonego do uporządkowania (liczba całkowita dodatnia)

Wyjście:
T[j] – *j-ty* element tablicy **T** złożonej z **n** liczb całkowitych uporządkowanych rosnąco

Lista kroków

1. Wczytaj tablicę **T** liczb do uporządkowania
2. Począwszy od **i = ile – 1**, skończywszy na i = **1**, wykonuj kroki **3 ÷ 8**
3. **x := T[i]**
4. **j := i + 1**
5. Dopóki **j <= ile** oraz **x > T [j]**, dopóty wykonuj kroki **6 ÷ 7**
6. **T [j – 1] := T [j]**
7. **j := j + 1**
8. **T [j – 1] := x**
9. Wydrukuj tablicę **T** uporządkowanych liczb
10. Zakończ działanie

Zadanie domowe 2. (obowiązkowe)
Na podstawie opisu algorytmu sortowania przez wstawianie skonstruuj schemat blokowy odzwierciedlający jego przebieg. Dodatkowo (nadobowiązkowo) napisz program w języku C# (C++, Python, Java), który realizuje ten algorytm.

Sortowanie przez wybieranie (ang. *selection sort*)

Zakładając, że chcemy posortować ciąg liczbowy rosnąco, algorytm sortowania przez wybór polega na znalezieniu najmniejszego elementu zbioru, a następnie zamianie go z pierwszym wyrazem ciągu. W ten sposób pierwszy element znajdzie się na właściwej pozycji. Możemy zatem skrócić ciąg o jeden element i postąpić w sposób identyczny z resztą. Algorytm realizuje sortowanie w miejscu (*in situ*) i ma złożoność obliczeniową rzędu $O(n^2)$.

Przykład
Uporządkować rosnąco ciąg liczb całkowitych {4, 7, 2, 9, 3}.

Ciąg wejściowy nieuporządkowany:		Ciąg wyjściowy uporządkowany:
	4 7 2 9 3	
Krok 1.	**4** 7 **2** 9 3	2
Krok 2.	**7** 4 9 **3**	2 3
Krok 3.	**4** 9 7	2 3 4
Krok 4.	**9** **7**	2 3 4 7
Krok 5.	**9**	2 3 4 7 9

Uwagi:

1. W kroku trzecim liczba 4 wymieniana jest sama ze sobą, bo jest na właściwym miejscu.
2. W kroku piątym liczba 9 dopisana jest na koniec zbioru, bo ostatni element zawsze znajduje się na swoim (ostatnim) miejscu.

Zadanie domowe 3. (obowiązkowe)
Na podstawie opisu algorytmu sortowania przez wybieranie skonstruuj pseudokod (mile widziany opis w języku angielskim) oraz schemat blokowy dokładnie odzwierciedlający jego przebieg. Dodatkowo (nadobowiązkowo) napisz program w języku C# (C++, Python, Java), który realizuje ten algorytm.

Uwagi natury ogólnej do zadań domowych:

1. Przede wszystkim należy dokładnie zapoznać się z treścią zadań i uwag (najlepiej ze zrozumieniem), albowiem bywa z tym bardzo różnie.

2. Dodatkowo (nieobowiązkowo) mile widziane programy komputerowe, realizujące dokładnie zaprojektowany algorytm. Do wyboru masz jeden z następujących języków: C++, Python, C#, Java.
3. Rozwiązania wyłącznie w postaci elektronicznej (każde zadanie należy potraktować osobno), zawierające wyłącznie pliki w formacie PDF ze schematami blokowymi oraz ewentualnie dodatkowo programy komputerowe (**tylko i wyłącznie** plik zawierający kod źródłowy) należy przesłać w nieprzekraczalnym terminie do dnia 9 grudnia 2022 r. (do północy) na adres: **ks.master@o2.pl**. W nazwach plików należy umieścić datę otrzymania zadania, czyli w tym wypadku 27.11.2022, numer zadania oraz swoje nazwisko (ewentualnie imię).
4. W nazwach plików używamy wyłącznie liter alfabetu angielskiego i nigdy nie używamy międzynarodowych znaków diakrytycznych oraz spacji. Do rozdzielania wyrazów służy notacja „wielbłądzia" (ang. *camel case*) albo podkreślnik (ang. *underscore*).