Sortowanie przez wstawianie
Insertion Sort

Algorytm **sortowania przez wstawianie** można porównać do sposobu układania kart pobieranych z talii. Najpierw bierzemy pierwszą kartę. Następnie pobieramy kolejne, aż do wyczerpania talii. Każdą pobraną kartę porównujemy z kartami, które już trzymamy w ręce i szukamy dla niej miejsca przed pierwszą kartą starszą (młodszą w przypadku porządku malejącego). Gdy znajdziemy takie miejsce, rozsuwamy karty i nową wstawiamy na przygotowane w ten sposób miejsce (stąd pochodzi nazwa algorytmu - sortowanie przez wstawianie, ang. Insertion Sort). Jeśli nasza karta jest najstarsza (najmłodsza), to umieszczamy ją na samym końcu. Tak porządkujemy karty. A jak przenieść tę ideę do świata komputerów i zbiorów liczbowych?

Algorytm sortowania przez wstawianie będzie składał się z dwóch pętli. Pętla główna (zewnętrzna) symuluje pobieranie kart, czyli w tym wypadku elementów zbioru. Odpowiednikiem kart na ręce jest tzw. lista uporządkowana (ang. sorted list), którą sukcesywnie będziemy tworzyli na końcu zbioru (istnieje też odmiana algorytmu umieszczająca listę uporządkowaną na początku zbioru). Pętla sortująca (wewnętrzna) szuka dla pobranego elementu miejsca na liście uporządkowanej. Jeśli takie miejsce zostanie znalezione, to elementy listy są odpowiednio rozsuwane, aby tworzyć miejsce na nowy element i element wybrany przez pętlę główną trafia tam. W ten sposób lista uporządkowana rozrasta się. Jeśli na liście uporządkowanej nie ma elementu większego od wybranego, to element ten trafia na koniec listy. Sortowanie zakończymy, gdy pętla główna wybierze wszystkie elementy zbioru.

Algorytm sortowania przez wstawianie posiada klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

Wstawianie elementu na listę uporządkowaną

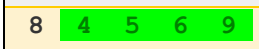
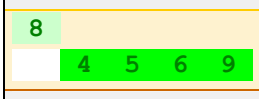
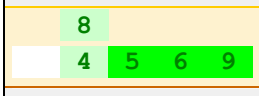
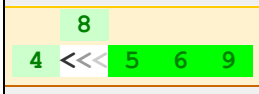
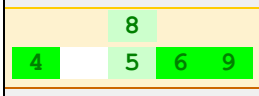
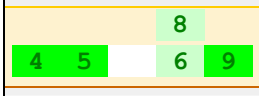
Najważniejszą operacją w opisywanym algorytmie sortowania jest wstawianie wybranego elementu na listę uporządkowaną. Zasady są następujące:

1. Na początku sortowania lista uporządkowana zawiera tylko jeden, ostatni element zbioru. Jednoelementowa lista jest zawsze uporządkowana.
2. Ze zbioru zawsze wybieramy element leżący tuż przed listą uporządkowaną. Element ten zapamiętujemy w zewnętrznej zmiennej. Miejsce, które zajmował, możemy potraktować jak puste.
3. Wybrany element porównujemy z kolejnymi elementami listy uporządkowanej.

4. Jeśli natrafimy na koniec listy, element wybrany wstawiamy na puste miejsce - lista rozrasta się o nowy element.
5. Jeśli element listy jest większy od wybranego, to element wybrany wstawiamy na puste miejsce - lista rozrasta się o nowy element.
6. Jeśli element listy nie jest większy od wybranego, to element listy przesuwamy na puste miejsce. Dzięki tej operacji puste miejsce wędruje na liście przed kolejny element. Kontynuujemy porównywanie, aż wystąpi sytuacja z punktu 4 lub 5.

Przykład:

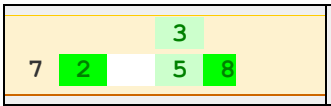
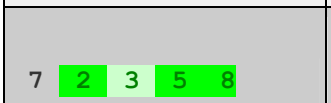
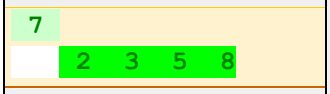
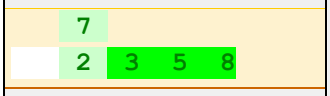
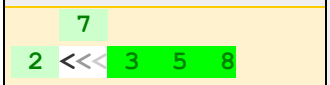
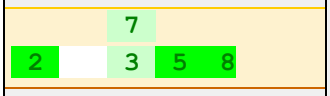
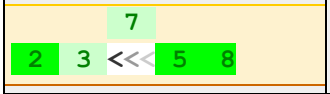
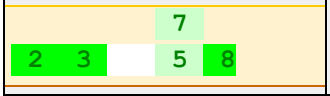
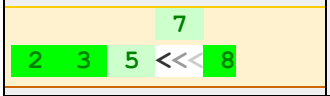
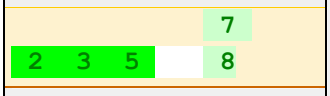
Dla przykładu wstawmy wg opisanej metody pierwszy element zbioru listę uporządkowaną utworzoną z pozostałych elementów $\{8\ 4\ 5\ 6\ 9\}$. Elementy listy uporządkowanej zaznaczyliśmy kolorem zielonym. Puste miejsce zaznaczyliśmy kolorem białym:

Zbiór	Opis operacji
	Element 8 znajduje się tuż przed listą uporządkowaną.
	Wybieramy ze zbioru element 8. Zajmowane przez niego miejsce staje się puste.
	Porównujemy 8 z pierwszym elementem listy uporządkowanej, z liczbą 4.
	Ponieważ element 4 jest mniejszy od elementu wybranego 8, przesuwamy go na puste miejsce. Zwróć uwagę, iż puste miejsce wędruje w kierunku końca listy uporządkowanej.
	Porównujemy 8 z kolejnym elementem listy uporządkowanej, z liczbą 5.
	5 jest5 mniejsze od 8, zatem wędruje na puste miejsce, które przesuwa się przed kolejny element listy uporządkowanej, liczbę 6.
	Porównujemy 8 z 6.
	6 jest mniejsze od 8, wędruje na puste miejsce.
	Porównujemy 8 z 9.
	Tym razem element wybrany wędruje na puste miejsce, ponieważ jest mniejszy od elementu 9 listy uporządkowanej. Operacja wstawiania jest zakończona. Lista rozrasta się o jeden element.

Przykład:

Wykorzystajmy podane informacje do posortowania opisywaną metodą zbioru { 7 3 8 5 2 }.

Zbiór	Opis operacji
7 3 8 5 **2**	Ostatni element jest zalążkiem listy uporządkowanej.
7 3 8 **5** **2**	Ze zbioru wybieramy element leżący tuż przed listą uporządkowaną.
7 3 8 **5** **2**	Wybrany element porównujemy z elementem listy.
7 3 8 2 **5** <<<	Ponieważ element listy jest mniejszy od elementu wybranego, to przesuwamy go na puste miejsce.
7 3 8 **2** **5**	Na liście nie ma już więcej elementów do porównania, więc element wybrany wstawiamy na puste miejsce. Lista uporządkowana zawiera już dwa elementy.
7 3 **8** **2** **5**	Ze zbioru wybieramy 8
7 3 **8** **2** **5**	8 porównujemy z 2
7 3 2 <<< **8** **5**	2 jest mniejsze, zatem przesuwamy je na puste miejsce.
7 3 **2** **8** **5**	8 porównujemy z 5
7 3 **2** **5** <<< **8**	5 jest mniejsze, przesuwamy je na puste miejsce
7 3 **2** **5** **8**	Lista nie zawiera więcej elementów, zatem 8 wstawiamy na puste miejsce. Na liście uporządkowanej mamy już trzy elementy.
7 **3** **2** **5** **8**	Ze zbioru wybieramy 3
7 **3** **2** **5** **8**	3 porównujemy z 2
7 2 <<< **3** **5** **8**	2 jest mniejsze, wędruje zatem na puste miejsce

	3 porównujemy z 5.
	Tym razem mniejszy jest element wybrany. Znaleźliśmy jego miejsce na liście, więc wstawiamy go na puste miejsce. Lista zawiera już 4 elementy.
	Ze zbioru wybieramy ostatni element - liczbę 7.
	7 porównujemy z 2
	2 jest mniejsze, przesuwamy je na puste miejsce
	7 porównujemy z 3
	3 jest mniejsze, przesuwamy je na puste miejsce
	7 porównujemy z 5
	5 jest mniejsze, przesuwamy je na puste miejsce
	7 porównujemy z 8
	Element wybrany jest mniejszy, wstawiamy go na puste miejsce. Lista uporządkowana objęła już cały zbiór. Sortowanie jest zakończone.

Specyfikacja problemu

Dane wejściowe

n - liczba elementów w sortowanym zbiorze, $n \in \mathrm{N}$

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

x - zawiera wybrany ze zbioru element.

Lista kroków

K01: **Dla** *j* = *n* - 1, *n* - 2, ..., 1: **wykonuj** K02...K04

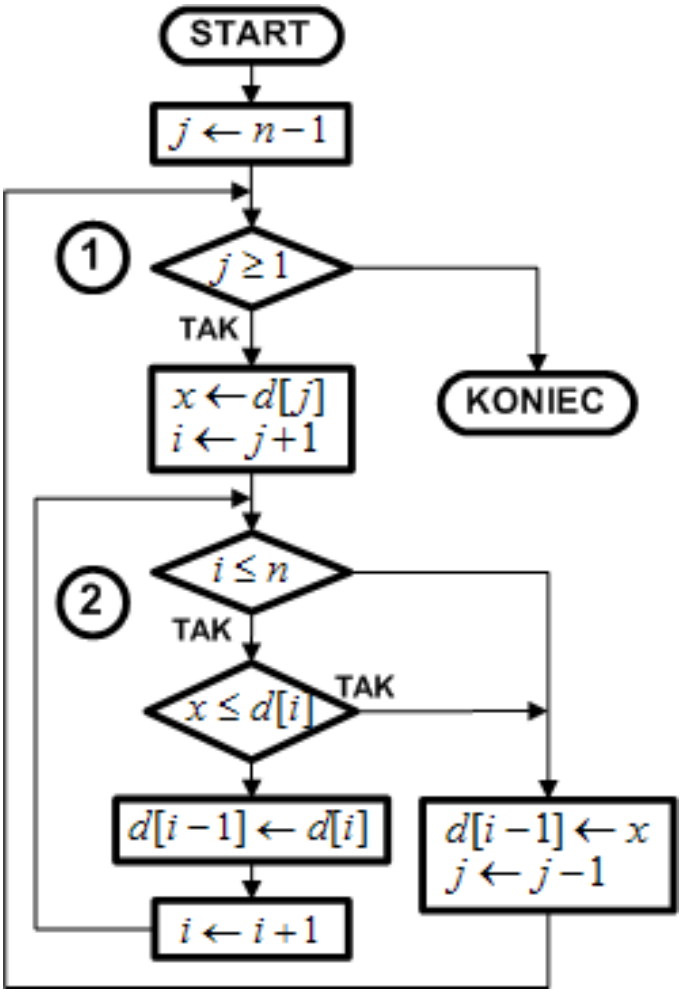
K02: *x* ← d[*j*]; *i* ← *j* + 1

K03: **Dopóki** (*i* ≤ *n*) □ (*x* > d[*i*]): **wykonuj** d[*i* - 1] ← d[*i*]; *i* ← *i* + 1

K04: d[*i* - 1] ← *x*

K05: **Zakończ**

Schemat blokowy



Pętlę główną rozpoczynamy od przedostatniej pozycji w zbiorze. Element na ostatniej pozycji jest zalążkiem listy uporządkowanej. Dlatego licznik pętli nr 1 przyjmuje wartość początkową $j = n - 1$.

Ze zbioru wybieramy element $d[j]$ i umieszczamy go w zmiennej pomocniczej x. Miejsce zajmowane przez ten element staje się puste.

Uwaga techniczna

W rzeczywistości na pozycji j pozostaje dalej ten sam element. Jednakże zapamiętaliśmy jego wartość, zatem pozycja ta może być zapisana inną informacją - elementu nie utracimy, ponieważ przechowuje go zmienna x. Dlatego pozycję tę możemy potraktować jako pustą, tzn. z nieistotną zawartością.

Pętlę wewnętrzną rozpoczynamy od pozycji następnej w stosunku do j. Pozycja ta zawiera pierwszy element listy uporządkowanej, która tworzona jest na końcu sortowanego zbioru. Pętlę wewnętrzną przerywamy w dwóch przypadkach - gdy licznik pętli wyjdzie poza indeks ostatniego elementu w zbiorze lub gdy element wybrany, przechowywany w zmiennej pomocniczej x, jest mniejszy lub równy bieżąco testowanemu elementowi listy uporządkowanej (dla sortowania malejącego należy zastosować w tym miejscu relację większy lub równy). W obu przypadkach na puste miejsce ma trafić zawartość zmiennej x i pętla zewnętrzna jest kontynuowana. Zauważ, iż pozycja tego miejsca w zbiorze jest równa $i - 1$.

Jeśli żaden z warunków przerwania pętli wewnętrznej nr 2 nie wystąpi, to przesuwamy bieżący element listy na puste miejsce i kontynuujemy pętlę wewnętrzną.

Podsumowując: pętla zewnętrzna wybiera ze zbioru kolejne elementy o indeksach od $n - 1$ do 1, pętla wewnętrzna szuka dla wybranych elementów miejsca wstawienia na liście uporządkowanej, po znalezieniu którego pętla zewnętrzna wstawia wybrany element na listę. Gdy pętla zewnętrzna przebiegnie wszystkie elementy o indeksach od $n - 1$ do 1, zbiór będzie posortowany.

Algorytm posiada klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.