SORTOWANIE PRZEZ WSTAWIANIE | JAVA

WSTĘP

**Sortowanie przez wstawianie** jest prostym algorytmem sortującym, którego działanie możemy intuicyjnie kojarzyć z układaniem kart na ręce przy jakiejś karcianej rozgrywce. Zasadniczo polega on na wstawianiu kolejnych elementów w odpowiednie miejsce już posortowanej tablicy.

IMPLEMENTACJA W JAVIE

W załączonym przeze mnie kodzie, napisanym w języku Java, znajduję się prosty program, który po uruchomieniu umożliwia komunikację z użytkownikiem. Program pozwala na własnoręczne wpisanie tablicy znaków całkowitoliczbowych (int) oraz wypisuje już posortowaną tablicę. Wszystko dzieje się za sprawą algorytmu sortowania przez wstawianie, którego implementacja znajduje się (w liniach 8-18 przesłanego pliku InsertionSort.java) pod postacią funkcji *insertionSort*, która wygląda następująco:

1   void insertionSort(int arr[]) {

2       int n = arr.length;

3       for(int i = 1; i < n; i++) {

4           int j;

5           int t = arr[i];

6           for(j = i; j > 0 && arr[j-1] > t; j--) {

7               arr[j] = arr[j-1];

8           }

9           arr[j] = t;

10       }

11   }

*UWAGA: Dla uproszczenia założyłem w implementacji algorytmu, że działamy na typie całkowitoliczbowym, jednak warto zaznaczyć, że zarówno algorytm jak i cały program działałby analogicznie i równie poprawnie dla dowolnego innego porównywalnego typu.*

OPIS DZIAŁANIA ZAIMPLEMENTOWANEGO ALGORYTMU

Działanie poszczególnych linijek widocznego powyższego kodu:

1 Funkcja *insertionSort,* odpowiedzialna za algorytm sortowania przyjmuje jako argument tablicę elementów ustalonego typu (w naszym przypadku jest to int). Sortowanie tablicy odbywa się na przekazanej w argumencie tablicy i tam też dzieją się wszystkie dokonywane zmiany, dlatego też funkcja nie ma potrzeby zwracania żadnej wartości i jest typu void.

2 W drugiej linii tworzymy zmienną n (typu int), do której przypisujemy długość tablicy. Będzie nam to potrzebne do wykonywania operacji na tej tablicy – aby nie wyjść poza tablicę. Alternatywnym rozwiązaniem byłoby przekazywanie ilości elementów w argumencie funkcji, jednak uznałem, że aktualne rozwiązanie, na które pozwala nam Java będzie bardziej uniwersalne.

3 Rozpoczynamy pętlę odpowiedzialną za wstawianie poszczególnych elementów naszej tablicy w odpowiednie miejsce posortowanej tablicy, która będzie się tworzyła po lewej stronie od aktualnie przerabianego indeksu tablicy. Warto zauważyć, że nie ma potrzeby wykonywania tej pętli dla elementu arr[0]. Ponieważ dla arr[1] tablica ‘’po lewej stronie’’ od niego jest już posortowana (zawiera tylko jeden element), możemy rozpocząć wykonywanie pętli   
od i = 1.

4 Tworzymy zmienną j, która zostanie wykorzystana do iteracji w celu znalezienia odpowiedniego miejsca dla aktualnie obsługiwanej wartości oraz zamiany elementów.

5 Tworzymy pomocniczą zmienną t, do której przypisujemy wartość elementu, którego będziemy wstawiać, aby mieć możliwość nadpisania arr[i].

6 i 7 W zasadzie w tych dwóch liniach odbywa się główna część naszego algorytmu, czyli tytułowe wstawianie. Mamy wstawić i-ty element w odpowiednie miejsce do podtablicy   
arr[0, i-1]. Aby tego dokonać ustawiamy zmienną j na i, a następnie zmniejszamy j do momentu, gdy element o indeksie j-1 będzie mniejszy lub równy od wstawianego elementu lub znajdziemy się na pierwszym elemencie w tablicy (j == 0). Z każdą iteracją pętli przesuwamy wartość elementu na j-1 miejscu o jedno pole tablicy w prawo. Ostatnim krokiem pętli jest zmniejszenie wartości iteratora j, co pozwoli nam wykonać ostatni etap wstawiania.

9 Na koniec ustawiamy wstawianą wartość na jej właściwe miejsce – arr[j].

*UWAGA: Zaimplementowany przeze mnie algorytm sortuje tablicę w porządku rosnącym. Aby posortować malejąco wystarczy jedynie obrócić znak nierówności w warunku pętli for w linijce 6:  
arr[j-1] > t.*

WAŻNE CECHY ALGORYTMU

* Algorytm sortowania przez wstawianie sortuje **w miejscu** – wszystkie sortowane elementy znajdują się przez cały czas w tablicy wejściowej
* Jest algorytmem sortującym **stabilnie** – nie zamienia kolejności elementów o tych samych wartościach. Wynika to z faktu, że zatrzymujemy pętlę odpowiedzialną za wstawianie danego elementu w momencie, gdy natrafimy na element o silnie większej (lub mniejszej) wartości. Oznacza to, że będziemy wpisywać nasz element za nim, a musiał on być wcześniej w tablicy niż element przez nas aktualnie wstawiany, jeżeli znajduje się już w posortowanej części tablicy.

ZŁOŻONOŚĆ CZASOWA ALGORYTMU

* przypadek pesymistyczny – O(n2)

void insertionSort(int arr[]) { O(n2)

    int n = arr.length; O(1)

    for(int i = 1; i < n; i++) { O(n2)

        int j; O(1)

        int t = arr[i]; O(1)

        for(j = i; j > 0 && arr[j-1] > t; j--) { O(n)

            arr[j] = arr[j-1]; O(1)

        }

        arr[j] = t; O(1)

    }

}

* przypadek optymistyczny – O(n)

void insertionSort(int arr[]) { O(n)

    int n = arr.length; O(1)

    for(int i = 1; i < n; i++) { O(n)

        int j; O(1)

        int t = arr[i]; O(1)

        for(j = i; j > 0 && arr[j-1] > t; j--) { 0

            arr[j] = arr[j-1]; O(1)

        }

        arr[j] = t; O(1)

    }

}

* przypadek średni – O(n2)

void insertionSort(int arr[]) { O(n2)

    int n = arr.length; O(1)

    for(int i = 1; i < n; i++) { O(n2)

        int j; O(1)

        int t = arr[i]; O(1)

        for(j = i; j > 0 && arr[j-1] > t; j--) { O(n)

            arr[j] = arr[j-1]; O(1)

        }

        arr[j] = t; O(1)

    }

}