# Упражнения: Сортиране чрез вмъкване и по метода на мехурчето

## Сортиране чрез метода на мехурчето

Сортирайте един масив от елементи с помощта на метода на мехурчето (Bubble Sort).

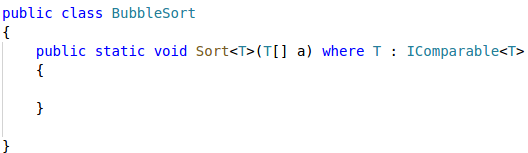
### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 4 3 2 1 | 1 2 3 4 5 |

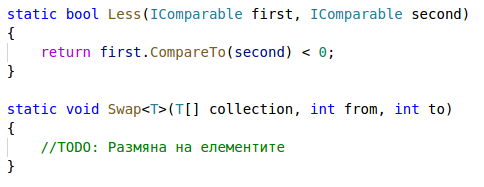
### Подсказки

Първо разгледайте описанието на алгоритъма от [Уикипедия](https://bg.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%BD%D0%B0_%D0%BC%D0%B5%D1%85%D1%83%D1%80%D1%87%D0%B5%D1%82%D0%BE) и визуализацията му във [Visualgo.net](https://visualgo.net/en/sorting?slide=6). Или пък изберете по-нестандартната визуализация като [унгарски народен танц](https://www.youtube.com/watch?v=lyZQPjUT5B4), представен от [AlgoRythmics](https://www.youtube.com/channel/UCIqiLefbVHsOAXDAxQJH7Xw). ☺

После създайте клас BubbleSort с един-единствен метод Sort



Създайте и два помощни метода - за сравняване на два елемента и за размяна на местата им:



В реализацията на метода Sort() опишете механизъма на самото сортиране:

* Обхождаме целия масив и **сравняваме всеки 2 съседни елемента**
* Ако първия е по-голям от втория, **те разменят местата си**.
* **Това се повтаря** до подреждането на масива.

Алгоритъмът може да бъде реализиран с вложени цикли:

* Вътрешният цикъл е за сравняване на елементите и размяна на местата им, ако първият е по-голям от втория. В резултат на това обхождане и размяна най-голямото („най-тежкото“) число „потъва“ до „дъното“ (края) на масива, а по-малките („по-леките“) числа „изплуват“ като мехурчета с по една позиция, ако са били след по-големи. Оттук и името на метода. Ако при някое обхождане няма нито една размяна, значи масивът вече е подреден. Помислете дали не можете да използвате това за оптимизиране на алгоритъма.
* Външният цикъл ни служи да укажем до кой елемент ще сравняваме съседните елементи. Помислете дали всеки път ще е нужно да проверяваме до края на масива.

## Сортиране чрез вмъкване

Сортирайте един масив от елементи с помощта на метода на сортиране чрез вмъкване (Insertion Sort). Но първо разгледайте описанието на алгоритъма от [Уикипедия](https://bg.wikipedia.org/wiki/%D0%A1%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%B0%D0%BD%D0%B5_%D1%87%D1%80%D0%B5%D0%B7_%D0%B2%D0%BC%D1%8A%D0%BA%D0%B2%D0%B0%D0%BD%D0%B5) и визуализацията му във [Visualgo.net](https://visualgo.net/en/sorting?slide=8) или пък тази под формата на [румънски народен танц](https://www.youtube.com/watch?v=ROalU379l3U), представен от [AlgoRythmics](https://www.youtube.com/channel/UCIqiLefbVHsOAXDAxQJH7Xw). ☺ Можете да направите сравнение между този и другите алгоритми за сортиране, които познавате, и чрез чудесната [визуализация на David Galles от USF](https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html) .

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 4 3 2 1 | 1 2 3 4 5 |

### Подсказки

Използвайте указанията на предната задача. Алгоритъмът накратко:

* Преместваме първия несортиран елемент наляво на мястото му
* Повтаряме това докато не останат несортирани елементи

## Визуализация на сортирането

Модифицирайте кода на предните две задачи, така че да служи и за визуализация на сортирането по тези два метода. За целта на конзолата отпечатвайте всяка съществена стъпка от алгоритъма, кои елементи ще се разменят и как изглежда масивът след всяка размяна. Обсъдете в клас коя визуализация се е получила най-прегледна и разбираема и защо.

### Подсказки

Модифицирайте метода **Swap( )**, така че да извежда кои елементи се разменят и да отпечатва масива след размяната.

## Уравновесен масив

Да се напише програма, която позволява да се въведе размер на масив и стойностите на неговите елементи и после го подрежда така, че стойностите на елементите да нарастват от началото до средата на масива и след това отново да намаляват от средата до края. Елементите от първата половина на масива не трябва да преминават във втората му половина. Полученият масив трябва да бъде отпечатан.

### Вход

* Входните данни трябва да се прочетат от конзолата.
* На първия ред се подава цяло четно число N, съдържащо броят на числата в масива
* На втория ред се подават N цели числа, отделени едно от друго с интервал. Това са стойностите на елементите в масива.
* Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

### Изход

* Изходните данни („уравновесеният“ масив) трябва да бъдат отпечатани на конзолата.

### Пример

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Коментар** |
| 10  **4 2 6 3 8** 1 7 4 2 9 | **2 3 4 6 8** 9 7 4 2 1 | За яснота елементите от първата половина на масива **са удебелени**. |

## Сортиране по произволна колона

Напишете програма, която сортира таблица от числови стойности по произволна колона.

### Вход

* Входните данни трябва да се прочетат от конзолата.
  + На първия ще ви бъдат подадени три цели числа R, C и S, разделени с интервал. R e броят на редовете в таблицата, C - броят на колоните, a S - номерът на колоната, по която трябва да бъде сортиран масива.
  + На следващите R реда ще са числата от всеки от редовете в таблицата, C на брой, разделени с интервали.
* Входните данни винаги ще са валидни и в описания формат.

### Изход

* Изходните данни трябва да бъдат отпечатани на конзолата.
* На R реда трябва да бъдат изведени числата от таблицата, сортирани по указаната колона.

### Подсказки

* Ще се наложи да ползвате двумерен масив или масив от масиви.
* Тествайте задачата с различно големи масиви и различни алгоритми за сортиране.
* Обърнете внимание дали при стабилни и нестабилни алгоритми се получава еднакъв резултат.
* Коментирайте в клас има ли начин за минимизиране на размяната на редовете при сортирането.

### Примери

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Вход** | **Изход** | **Коментар** | **Вход** | **Изход** | **Коментар** |
| 3 4 **1**  1 2 3 4  3 1 2 4  2 3 1 2 | 1 2 3 4  2 3 1 2  3 1 2 4 | Има 3 реда и 4 колони.  Вторият и третият ред трябва да разменят местата си, ако сортираме по колона **1**. | 4 2 **2**  1 2  3 1  2 3  4 4 | 3 1  1 2  2 3  4 4 | Има 4 реда и 2 колони.  Първият и вторият ред трябва да разменят местата си, ако сортираме по колона **2**. |