# Упражнения: Алгоритми за търсене и сортиране

Тествайте решенията си в Judge: <https://judge.softuni.org/Contests/4175/14-Sorting-and-Searching-Algorithms>.

# Част I – Сортиране

## Сортиране

Имплементирайте един или повече от следните **алгоритми за сортиране**:

* Insertion
* Bubble
* Shell

**Прочете** редица от числа от конзолата. След това ги **сортирайте** и ги **отпечатайте**.

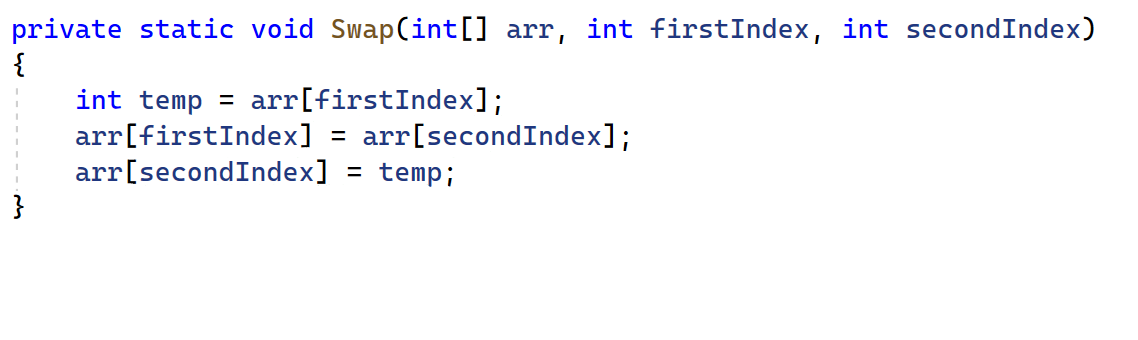
### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 4 3 2 1 | 1 2 3 4 5 |
| 1 4 2 -1 0 | -1 0 1 2 4 |

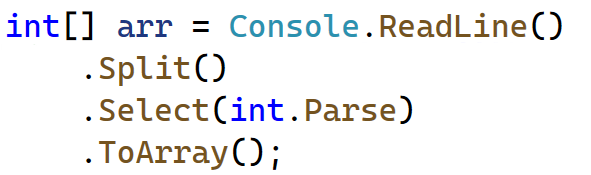
### Насоки

#### Сортиране чрез вмъкване

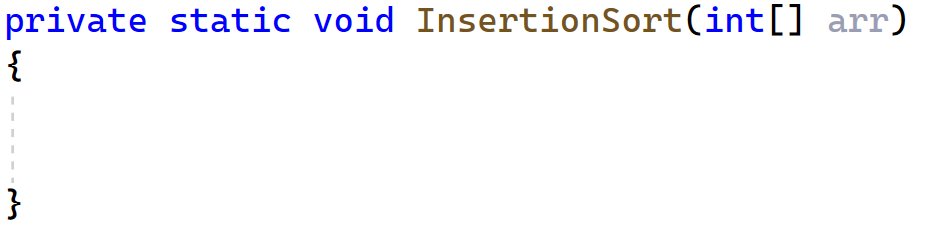
1. Първо създайте метода Swap(int[] arr, int firstIndex, int secondIndex), който ще **разменя** две числа:



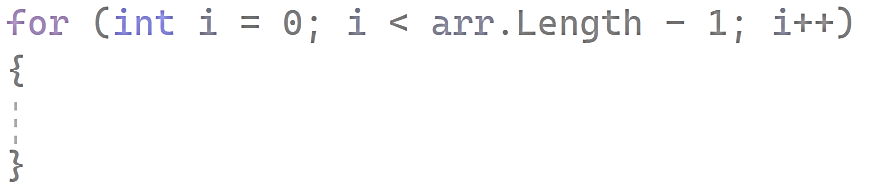
1. В Main() **метода** прочетете **числата**, които ще сортирате:



1. След това създайте метод InsertionSort(int[] arr), който приема като аргумент **масив от числа**:



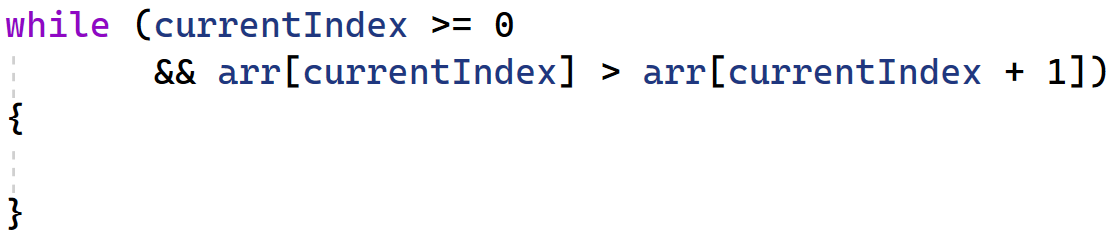
1. В него създайте for**-цикъл**, който ще итерира до дължината на **масива** **– 1**:



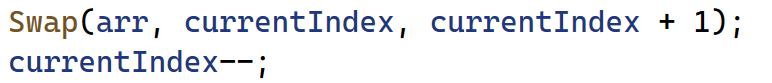
1. В **for-цикъла** създайте променлива **currentIndex** чрез която ще съхраним индекса на текущия елемент:



1. След това създайте **while-цикъл**, който ще итерира докато текущия **индекс** е **по-голям** или **равен** на **0** и елемента на **текущия индекс** е **по-голям** от **следващия елемент**:



1. В **while-цикъла** сменете **позициите** на **двата** **елемента** чрез метода Swap(int[] arr, int firstIndex, int secondIndex) и намалете индекса на **текущия** **индекс** с **1**:



1. Сега се върнете отново в Main() метода и извикайте метода InsertionSort(int[] arr):

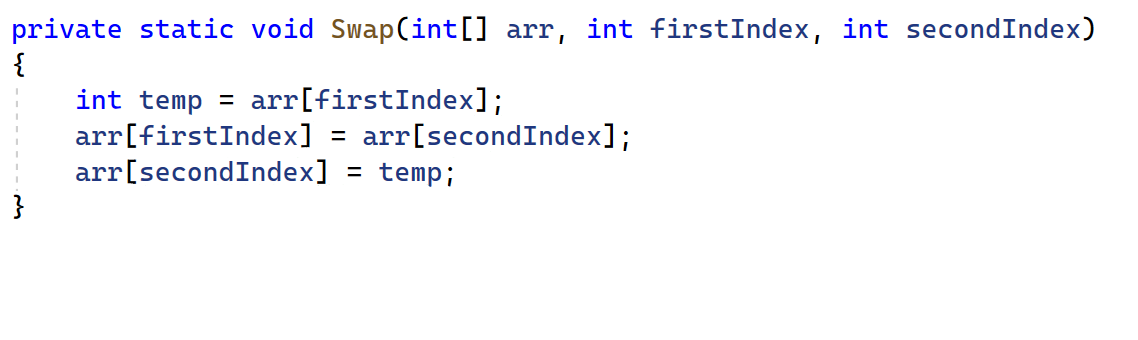


1. Накрая отпечатайте елементите:

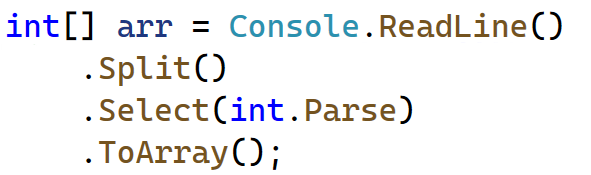


#### Сортиране чрез Bubble sort

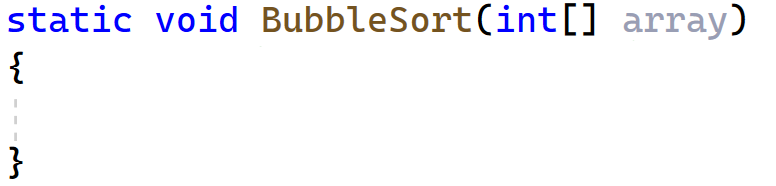
1. Първо създайте метода Swap(int[] arr, int firstIndex, int secondIndex), който ще **разменя** две числа:



1. В Main() **метода** прочетете **числата**, които ще сортирате:



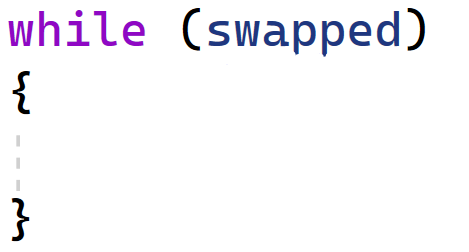
1. След това създайте метод BubbleSort(int[] arr), който приема като аргумент **масив от числа**:



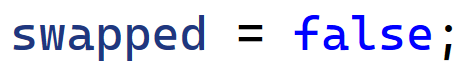
1. В него създайте променливата swapped със стойност true:



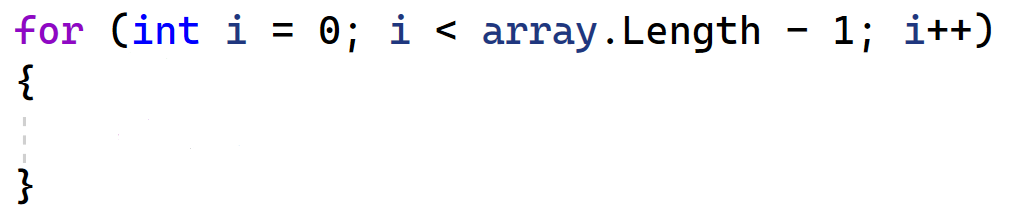
1. След това създайте while**-цикъл**, който ще итерира докато swapped е true:



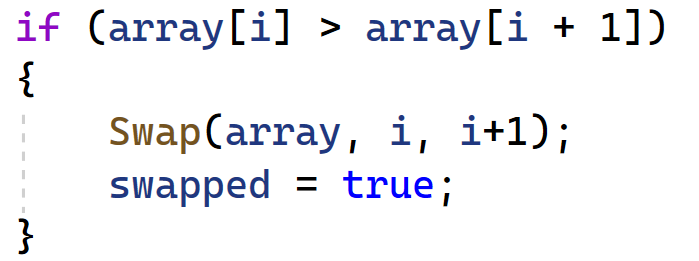
1. В цикъла **задайте** стойност на swapped false:



1. Създайте for**-цикъл**, който ще итерира до дължината на **масива** **– 1**:



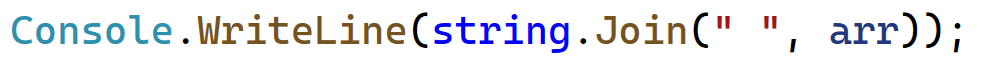
1. В него създайте проверка дали **елемента** с **текущия индекс** е по-голям от **елемента с текущия индекс + 1**. Ако това е така, разменете елементите и сложете стойност на swapped true:



1. Сега се върнете отново в Main() метода и извикайте метода BubbleSort(int[] arr):

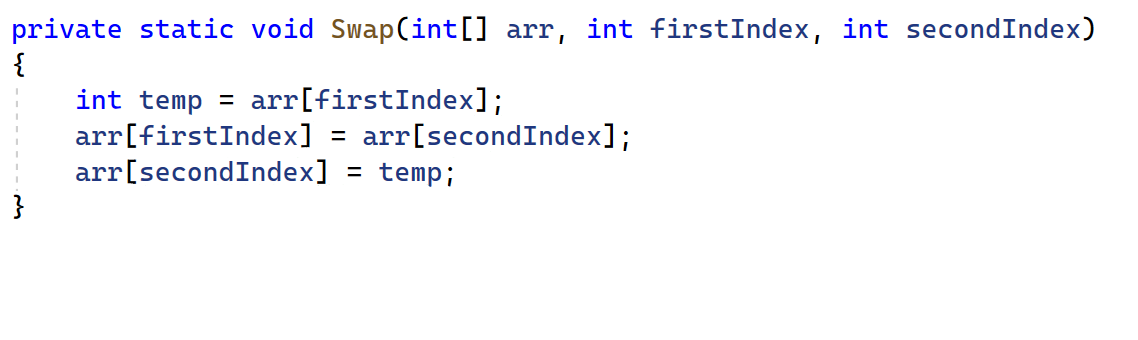


1. Накрая отпечатайте елементите:

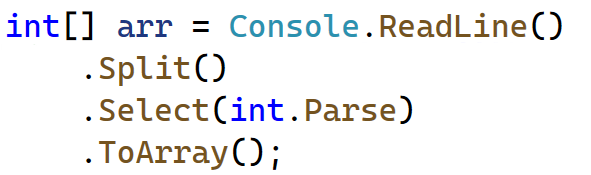


#### Сортиране чрез Shell sort

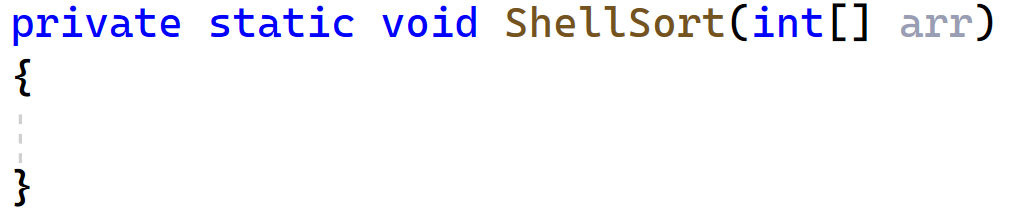
1. Първо създайте метода Swap(int[] arr, int firstIndex, int secondIndex), който ще **разменя** две числа:



1. В Main() **метода** прочетете **числата**, които ще сортирате:



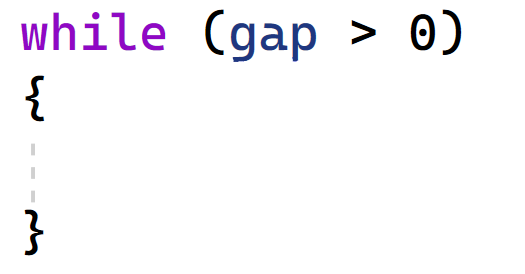
1. След това създайте метод ShellSort(int[] arr), който приема като аргумент **масив от числа**:



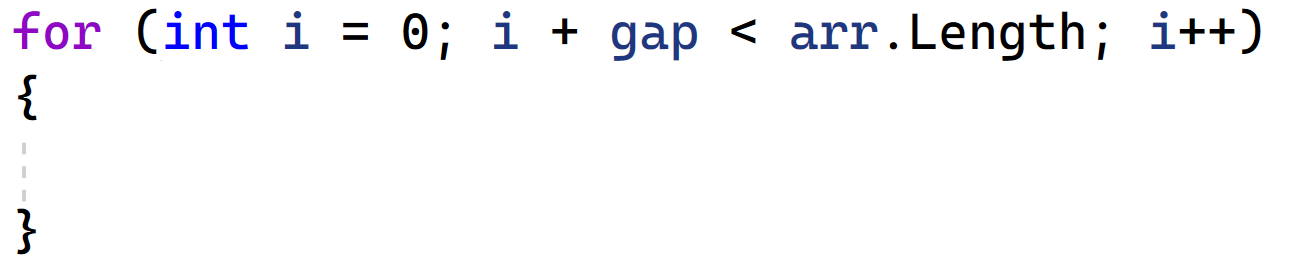
1. В него създайте променливата **gap** със **стойност** дължината на масива делено на 2:



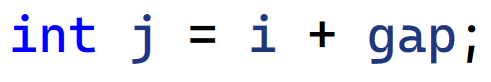
1. След това създайте **while-цикъл**, който ще **итерира** докато променливата gap е по-голяма от **0**:



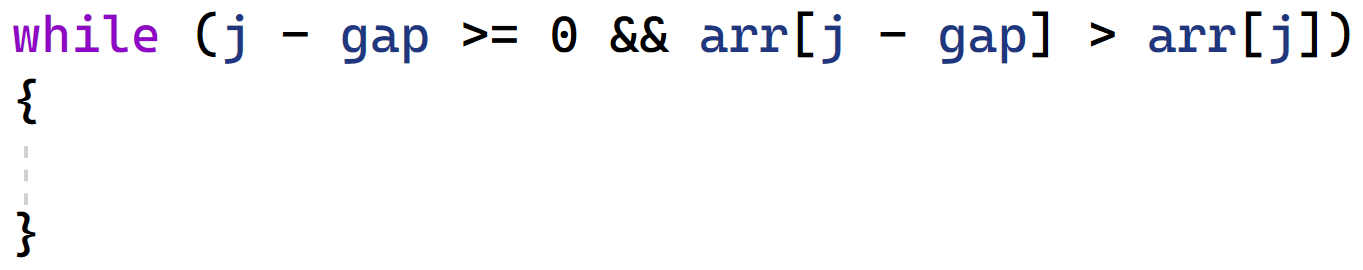
1. В него създайте for**-цикъл**, който ще итерира докато сумата на i и gap е по-малка от дължината на **масива**:



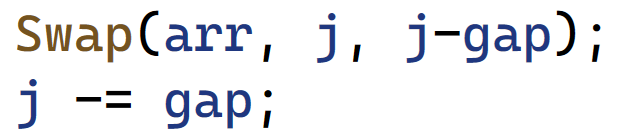
1. Създайте променлива j със стойност сумата на i и **gap**:



1. След това създайте още един while**-цикъл**, който ще итерира докато разликата на j **–** gap е по-голяма или равна на **0** и **елемента** на **масива** с индекс i **–** gap е по-голям от елемента на масива с индекс j:



1. Извикайте метода Swap(int[] arr, int firstIndex, int secondIndex) като му подайте стойностите j и j **–** gap и намалете стойността на j с gap:



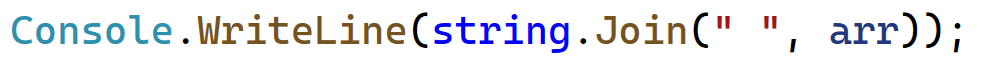
1. След for**-цикъла** намалете стойността на gap с **2**:



1. Сега се върнете отново в Main() метода и извикайте метода BubbleSort(int[] arr):



1. Накрая отпечатайте елементите:



## Merge Sort

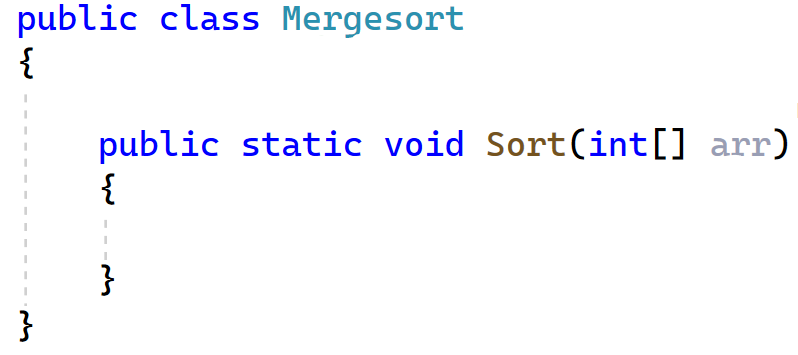
Сортирайте масив от числа чрез известния алгоритъм **Merge.**

### Примери

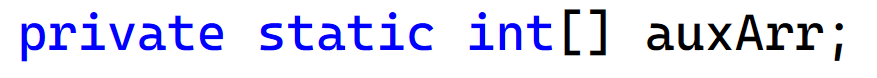
|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 4 3 2 1 | 1 2 3 4 5 |

### Насоки

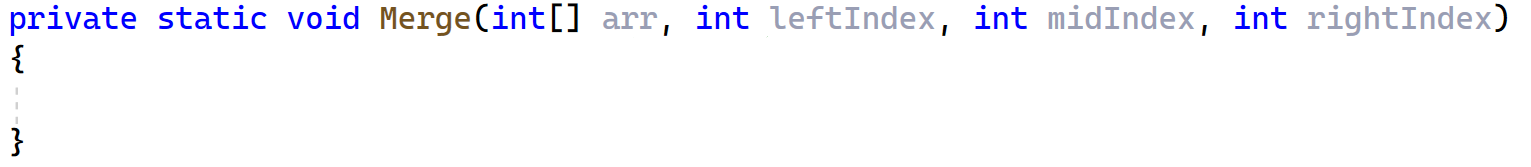
1. Създайте клас Mergesort с метод за сортиране:



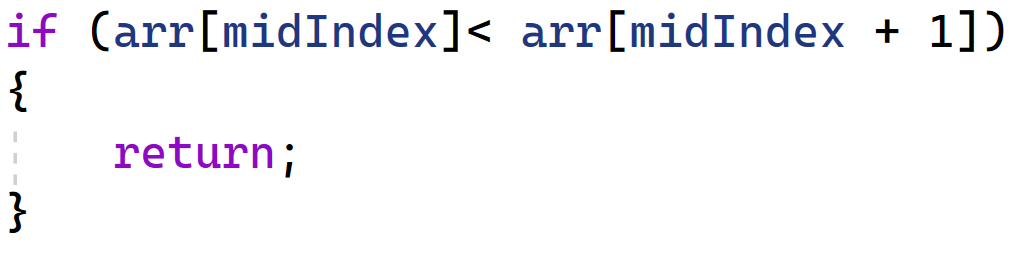
1. Създайте **помощен масив**, който ще помогне за обединяването на под масивите:



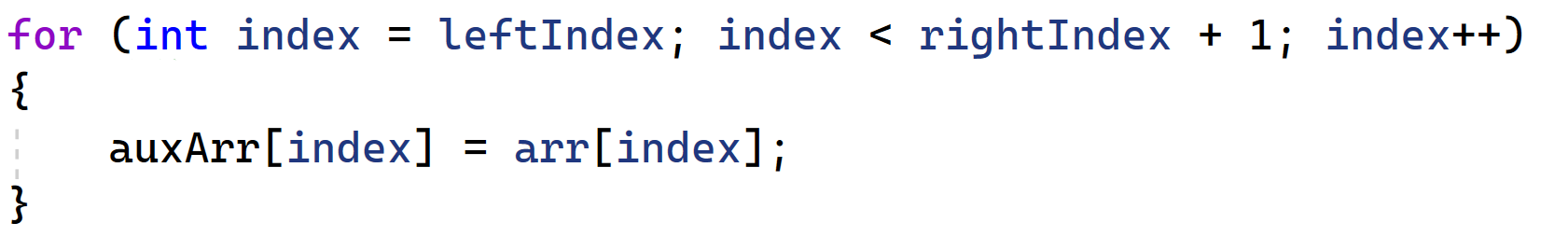
1. Създайте метод Merge(int[] arr, int leftIndex, int midIndex, int rightIndex):



1. В него проверете дали **средният** **елемент** е по-малък от **следващия елемент**. Ако е така, върнетe return:

****

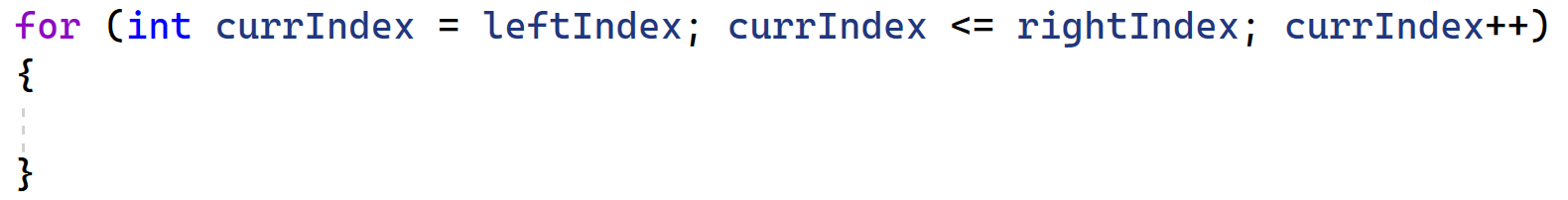
1. След това прехвърлете **всички елементи в помощния масив:**



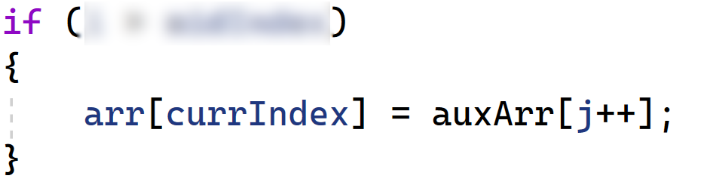
1. Създайте две променливи чрез които ще следим **текущата позиция** на **елементите** в **лявата** и **дясната** част на подмасива:



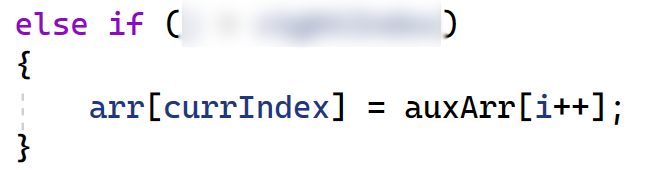
1. Създайте for**-цикъл**, който ще итерира докато currIndex e по-малък от rightIndex:



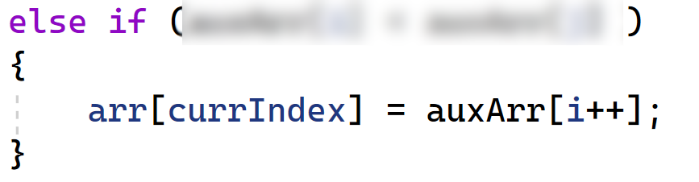
1. Създайте проверка дали променливата **i** е извън границата. Ако е така, на масива arr с индекс currIndex присвоете елемента на помощния масив auxArr с индекс j++. Това означава, че всички елементи в лявата половина на масива са **били обединени**:



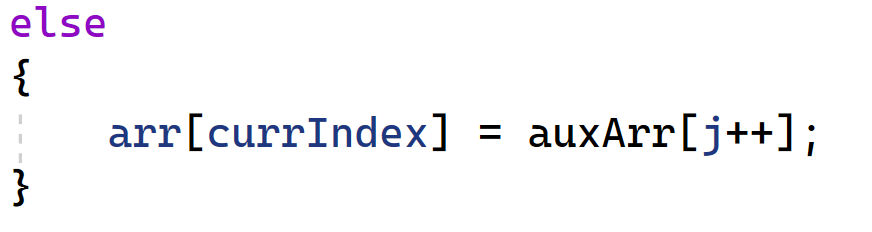
1. Създайте проверка дали променливата j е извън границата. Ако е така, на масива arr с индекс currIndex присвоете елемента на помощния масив auxArr с индекс i++. Това означава, че всички елементи в дясната половина на масива са **били обединени**:



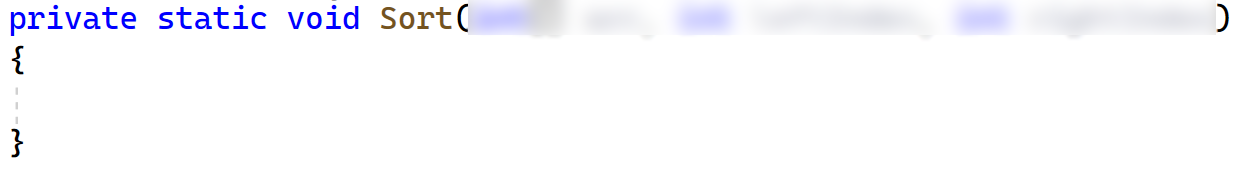
1. Създайте проверка дали елементът от помощния масив auxArr на позиция i е по-голям от елемента от помощния масив auxArr на позиция j. Ако е така, задайте стойност на масива arr с индекс currIndex стойността на елемента на помощния масива auxArr с индекс **i++**:



1. В противен случай, задайте стойност на масива **arr** с индекс currIndex стойността на елемента на помощния масива auxArr с индекс j++:



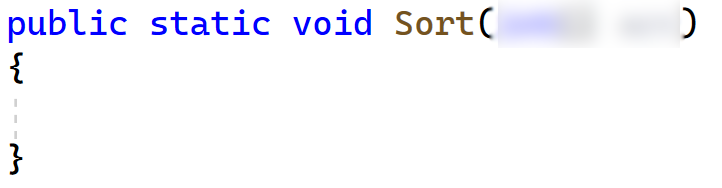
1. След това създайте рекурсивен метод Sort(int[] arr, int leftIndex, int rightIndex):



1. В него създайте дъно – това е когато променливата е leftIndex по-голяма или равна на rightIndex.
2. Създайте променлива midIndex, която има стойност leftIndex + (rightIndex - leftIndex) / 2.
3. След това извикайте метода Sort() като му подайте аргументите arr, leftIndex, midIndex и извикайте още веднъж метод като му подайте стойностите arr, midIndex+1, rightIndex:



1. След двата метода извикайте Merge(int[] arr, int leftIndex, int midIndex, int rightIndex).
2. Създайте метод Sort(int[] arr), който трябва да приема масив:



1. В него задайте инициализирайте помощния масив и извикайте метода Sort(int[] arr, int leftIndex, int rightIndex):



1. Накрая Main() метода прочетете числата, извикайте метода Sort(int[] arr) и отпечатайте.

## Quicksort

Сортирайте масив от числа чрез известния алгоритъм **Quicksort**.

### Примери

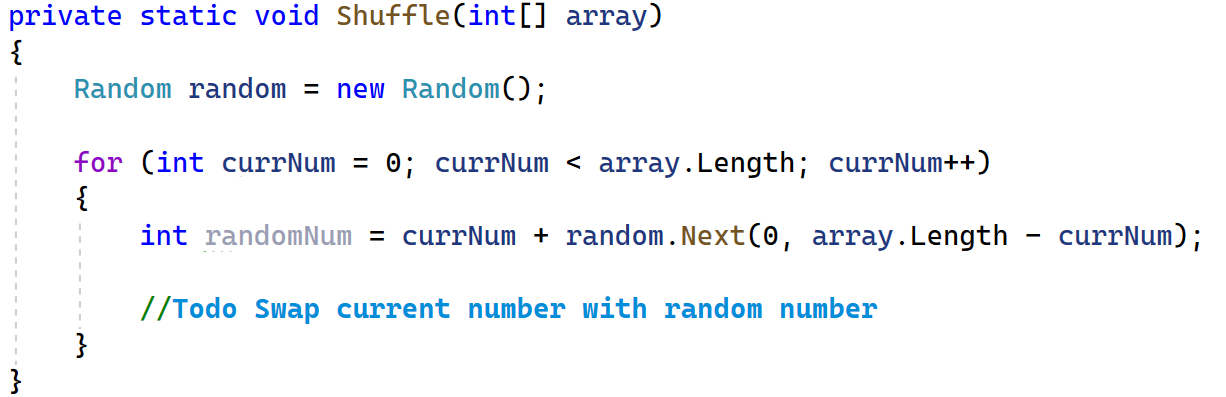
|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 5 4 3 2 1 | 1 2 3 4 5 |

### Насоки

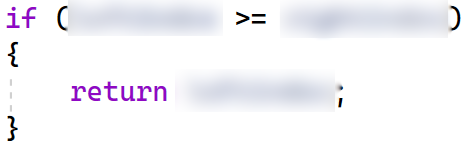
Можете да разгледате алгоритъма Quicksort от [Wikipedia](https://en.wikipedia.org/wiki/Quicksort).

Можете да видите визуализация на алгоритъма на [Visualgo.net](https://visualgo.net/en/sorting).

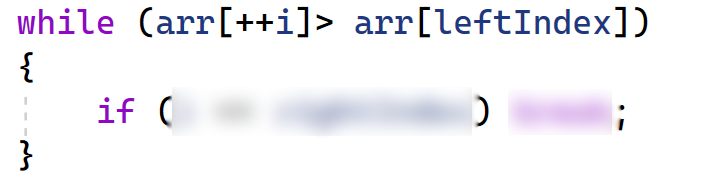
1. Създайте клас Quick.
2. Създайте метода Swap(int[] arr, int firstIndex, int secondIndex), който ще **разменя** две числа.
3. Създайте метод Shuffle(int[] array), който ще разменя елементите на случаен принцип чрез класа Random:



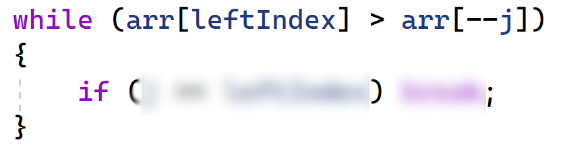
1. Създайте метод Partition(int[] arr, int leftIndex, int rightIndex).
2. В него проверете дали **левият индекс** е **по-голям или равен** на **десния индекс**. Ако това е така, подмасивът **има един или нула елемента** и не се нуждае от разделяне. В този случай методът **връща левия индекс** като резултат:



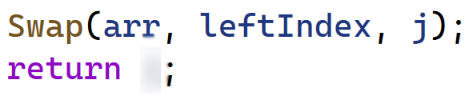
1. След това създайте **променлива** i със стойност leftIndex, която ще сочи към следващия елемент от ляво на дясно в подмасива и **променлива** j със стойност rightIndex **+ 1**, която ще сочи към предишния елемент от дясно на ляво в подмасива.
2. Създайте while**-цикъл**.
3. Проверете дали елементът на масива с индекс i + 1 е **по-малък** от елементът на масива с индекс leftIndex. Ако това е вярно, **цикълът** продължава да **увеличава** i, докато не намери **по-голям или равен елемент** или не достигне **десния край на подмасива**:



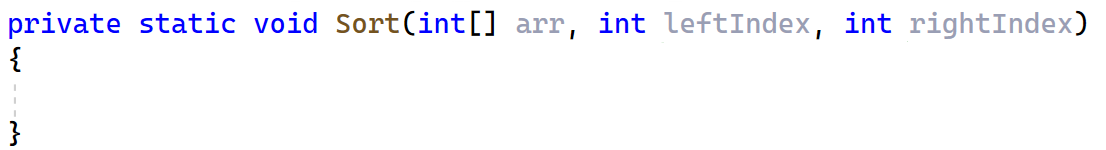
1. Проверете дали елементът на масива с индекс leftIndex е по-малък от елементът на масива с индекс j **- 1**. Ако това е така, цикълът продължава да намалява j, докато не намери **по-малък или равен елемент** или не достигне **левия край на подмасива**:



1. След това проверете дали променливата i е по-голяма или равна на j. Ако това е така, значи i и j са се разминали и няма повече елементи за размяна и цикълът **прекъсва** и преминава към следващата стъпка. В противен случай, **извикайте** **метода** Sort(int[] arr, int leftIndex, int rightIndex):
2. След while**-цикъла** извикайте **метода** Sort(int[] arr, int leftIndex, int rightIndex) и върнете j:



1. Създайте метод Sort(int[] arr, int leftIndex, int rightIndex), който използва метода Partition(int[] arr, int leftIndex, int rightIndex) за разделяне на подмасива и ще извика себе си рекурсивно за сортирането на левия и десния подмасив:



1. В метода добавете **рекурсивно дъно** – проверете дали leftIndex е по-голям или равен на rightIndex. Ако е така върнете return.
2. Създайте променлива pivot, която има стойността на метода Partition(int[] arr, int leftIndex, int rightIndex):



1. Извикайте същия методa Sort(int[] arr, int leftIndex, int rightIndex), за да сортирате левия подмасив, който се състои от елементите от leftIndex до pivot **- 1**. След това извикайте отново същия метод, за да сортирате десния подмасив, който се състои от елементите от pivot **+ 1** до rightIndex:



1. След това създайте метод Sort(int[] arr)**, който ще извиква метода** Shuffle(int[] array)**и след това ще извика метода** Sort(int[] arr, int leftIndex, int rightIndex)**.**
2. Накрая Main() метода прочетете числата, извикайте метода Sort(int[] arr)**,** и отпечатайте.

## Брой обръщания

Имаме масив. **Броят на обръщанията на масива** е равен на броя на **стъпките**, с които е отдалечен (или приближен) до неговото сортирането. Ако масивът е сортиран, броят на обръщанията е равен на **0**. Ако масивът е сортиран в **обратен** **ред**, броят на обръщанията е равен на **дължината на масива**.

Два елемента a[i] и a[j] са обърнати, когато a[i] > a[j] и i < j.

Намерете и **отпечатайте броя на обръщанията** на масива.

### Примери

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обръщания** |
| **2 4 1 3 5** | 3 | 2 1  4 1  4 3 |
| **5 4 3 2 1** | 10 | 5 4  5 3  5 2  5 1  4 3  4 2  4 1  3 2  3 1 |

### Насоки

* Използвайте модифицирана версия на алгоритъма **Merge**.
* Полезно четиво: [http://www.geeksforgeeks.org/counting-inversions](http://www.geeksforgeeks.org/counting-inversions/).

## Сортиране на числа по име

Даден ви е масив с числа. Трябва да ги подредите по техните **имена** в английската азбука. Например числата са **0, 1, 2, 3, 4, 5, 6, 7, 8, 9**. Техният сортиран вариант е **8, 5, 4, 9, 1, 7, 6, 3, 2, 0** (eight, five, four, nine,one, seven, six, three, two, zero, подредени по азбучен ред).

Числата по-големи от 10 са презентирани по подобен начин. Например **88** е ‘**eight-eight**’ и **1234** е ‘**one-two-three-four’**. Ако някое число започва със същото име на друго числа, като **11** (**one-one**) и **111** (**one-one-one**), то по-малкото число идва първо.

Във входа няма да има негативни числа.

### Вход

* Входа ще бъде на един ред – числа, разделени със **запетая** и **интервал**

### Изход

* Накрая отпечатайте сортираните числа във формата: {**n1, n2, n3 … n**}

### Бележки

* Числа ще бъдат **неотрицателни**
* **Няма** да има **повече от 50** числа
* Позволено време/памет: 100 ms / 16 MB

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 | 8, 5, 4, 9, 1, 7, 6, 3, 2, 0 |
| 1111, 1, 111, 11 | 1, 11, 111, 1111 |
| 17, 32, 45, 88, 44 | 88, 45, 44, 17, 32 |

### Насоки

1. Преди Main() метода създайте **масив от низове**, който съдържа **цифрите** **написани** на **английски**.
2. Създайте метод Num2Text(string num, StringBuilder numToText).
3. В него обходете **цифрите** на числото чрез foreach**-цикъл**.
4. Добавете **текста** на всяка цифра в numToText.
5. В Main() метода прочетете числата.
6. Създайте **речник**, за да запазите **числата** и **техните имена**.
7. Обходете прочетените числа чрез foreach**-цикъл**.
8. В него създайте StringBuilder.
9. Извикайте метода Num2Text(string num, StringBuilder numToText).
10. В речника добавете числото и StringBuilder-а.
11. След foreach**-цикъл** сортирайте **числата** по **техния текст** в **азбучен ред**.
12. Накрая отпечатайте **сортираните числа**.

# Част II – Търсене

## Имплементиране на двоично търсене

Имплементирайте алгоритъм, който намира **индекс** в сортиран масив от числа за **логаритмично време**.

### Примери

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Обяснения** |
| 1 2 3 4 5  1 | 0 | Индекс на 1 е 0 |
| -1 0 1 2 4  1 | 2 | Индекс на 1 е 2 |
| 1 2 3 4 5  6 | -1 | В масива няма 6 |

### Насоки

Ако не сте запознати с концепцията, можете да прочете за двоично търсене в [Wikipedia](https://en.wikipedia.org/wiki/Binary_search_algorithm).

[Тук](https://www.cs.usfca.edu/~galles/visualization/Search.html) можете да намерете инструмент, които показва визуализация на търсенето.

1. Започнете като създадете клас BinarySearch с метод IndexOf(int[] arr, int key).
2. Създайте променлива leftIndex със стойност 0 и rightIndex със стойност **дължината на масива – 1**
3. Създайте while**-цикъл**, който ще итерира докато променливата leftIndex е **по-малък или равен** на rightIndex.
4. В while**-цикъла** създайте midIndex със стойност leftIndex + (rightIndex - leftIndex )/2**.**
5. Ако променливата key е по-малка от елемента с **индекс** midIndex, задайте стойност на rightIndex midIndex **- 1**.
6. Ако променливата key e по-голяма от елемента с индекс midIndex, задайте стойност на leftIndex midIndex + 1.
7. В противен случай, върнете midIndex.
8. Накрая на метода върнете **-1**.

## Търсене

Имплементирайте един или няколко от следните алгоритми за търсене:

* Линейно търсене
* Търсене на двоично търсене

**Ще прочетете** редица от числа на първия ред. На следващия ще получите **число**. Намерете индекса на числото от масива. Ако няма такова число, върнете **-1.**

### Примери

|  |  |
| --- | --- |
| **Вход** | **Изход** |
| 1 2 3 4 5  1 | 0 |
| 1 2 3 4 5  6 | -1 |

## Игла

Вашата задача е да намерите правилното място на числата в масива. От конзолата ще прочетете редица от **неподредени** числа със случайно разпределени "**дупки**" (нули) сред тях.

След това ще получите **игли** – числа, който трябва да **вмъкнете** в редицата, така че тя да остане **ненамаляваща** (без "дупки"). За всяка дупка намерете **най-левия** **индекс**, където може да бъде добавена игла.

### Вход

* Входът ще се чете от конзолата.
* На първия ред ще получите числата C и N, разделени с **интервал**.
* На следващия ред ще получите **C положителни числа**, образуващи неподредена редица (пренебрегвайки нулите)
* На третия ред ще получите **N положителни числа** – **иглите**
* Входът винаги ще бъде валиден и във формата по-горе. Няма нужда от допълнителни проверки.

### Изход

* Изходът трябва да бъде на **един ред**.
* На изхода трябва да принтирате **N числа**, разделени с интервал. Всяка цифра представлява н**ай-левият индекс**, в който може да се постави съответната игла.

### Бележки

* N ще бъде в обхвата [1 … 1000].
* C ще бъде в обхвата [1 … 50000].
* Максималното време на програмата е 0.1 секунда. Максималната памет е 16 MB.

### Примери

|  |
| --- |
| **Вход** |
| 23 9  3 5 11 0 0 0 12 12 0 0 0 12 12 70 71 0 90 123 140 150 166 190 0  5 13 90 1 70 75 7 188 12 |
| **Изход** |
| 1 13 15 0 13 15 2 21 3 |
| **Обяснения** |
| 5 идва на индекс 1 – между 3 и 5  13 идва на индекс 13 – 12 и 70  90 идва на индекс x 15 – между 71 и 0  1 идва на индекс 0 – преди 3  И т.н. |

|  |
| --- |
| **Вход** |
| 11 4  2 0 0 0 0 0 0 0 0 0 3  4 3 2 1 |
| **Изход** |
| 11 1 0 0 |

# Част III – Shuffling

## Shuffle Words

Ще прочетете от конзолата **думи**, разделени с **интервал**. Използвайте алгоритъма **Fisher-Yates Shuffle**, за да **разбъркате** думите. Отпечатайте разбърканите думи, всяка **на нов ред**.

### Примери

Имайте предвид, че това е само примерен резултат, тъй като резултатът след разбъркване винаги е различен поради класа **Random()**.

|  |  |
| --- | --- |
| **Вход** | **Примерен изход** |
| first second third fourth fifth | second  fifth  third  fourth  first |