# 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 |

## 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 един метод за сортиране:

Graphical user interface, text, application

Description automatically generated

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



1. Нека да започнем с имплементирането на метода Merge():



1. При сортирането на двата подмасива, ако **най-големият елемент вляво** е по-малък от **най-малкия елемент** **вдясно**, то двата подмасива са **вече сортирани**

A picture containing text

Description automatically generated

1. Ако не са сортирани, прехвърлете **всички елементи в помощния масив**

Graphical user interface

Description automatically generated with low confidence

1. След **това ги обединете** в главния масив

Graphical user interface, text

Description automatically generated with medium confidence

1. Сега трябва да направим рекурсивния метод Sort()



1. Ако има само един елемент в подмасива, то той **вече е сортиран**

A picture containing text

Description automatically generated

1. Ако не е, трябва да го **разделим на два подмасива**. След това ги **сортираме рекурсивно** и накрая ги **обединяваме по пътя нагоре от рекурсията** (след рекурсивното извикване)

Text

Description automatically generated with low confidence

1. Сега можем да извикаме метода Sort()

Text

Description automatically generated with medium confidence

## 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).

Алгоритъмът накратко:

* Quicksort взима несортирана част от масива и я сортира
* Избираме **пивот**
  + Взимаме първия елемент от несортираната част. Преместваме го, така че всички **по-малки** елементи да са **отляво**, а всички **по-големи** - **отдясно**
* След като пивота е преместен на правилното си място, вече имаме **два несортирани дяла** - един вляво от него и един вдясно.
* **Извикваме процедурата рекурсивно** за всяка една част
* **Дъното** на рекурсията е, когато **размерът на частта е 1**, която по дефиниция е сортирана.

1. Първо, дефинираме **класа** и **метода за сортиране:**

Graphical user interface, text, application, email

Description automatically generated

1. Сега трябва да имплементираме частния метод Sort(). Не забравяйте да направете **дъно на рекурсията**

Graphical user interface, text, application

Description automatically generated

1. Първо намираме индекса на пивота и пренареждаме елементите. След това сортираме лявата и дясната част рекурсивно:

A picture containing text, orange, close

Description automatically generated

1. Сега трябва да изберем точката на пивот. Трябва да създадем метода Partition()



1. Ако има само един елемент, той вече е разделен и индексът на пивота е индексът на единствения му елемент.

Text

Description automatically generated with medium confidence

1. Намирането на точката на павиот включва пренареждане на всички елементи в частта, така че то да отговаря на условието всички елементи в лявата част на точката на пивот да са по-малки от нея, а всички елементи в дясната ѝ част да са по-големи от нея.

Graphical user interface, text, application

Description automatically generated

## 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

* Използвайте функцията Num2Text(int), която трансформира число във текст, както по-горе.
* Използвайте Dictionary<string, string>, за да пазите **числата** и техните **имена**. Сортирайте го с метода .OrderBy().

# 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.
* Накратко, ако имате **сортирана** колекция със **сравними елементи**, може да използваме **двоичното търсене**. Така може да елиминираме **половината** елементи и да приключим за **логаритмично време**.
* **Двоичното търсене** е разделяй и владей алгоритъм. Започваме от **средата** на колекцията. Ако не сме намерили елемент, имаме три възможни опции:
  + Ако е елементът е **по-малък** – започваме да разглеждаме елементите **отляво**, защото елементите отдясно са по-големи
  + Ако елементът е **по-голям** – започваме да разглеждаме елементите **отдясно**
  + Ако елементът **не е наличен**, връщаме **-1**.

1. Започваме като създаваме **клас** с **метод**:

Graphical user interface, text, application

Description automatically generated

1. В метода декларираме две променливи за **границите** на търсенето и **while-цикъл**:

Text

Description automatically generated

1. В while-цикъла създаваме променлива за намиране на **средата:**



1. Ако **ключът** е вляво от средата, премества дясната граница. Ако **ключът** е вдясно от средата, преместваме дясната граница

Graphical user interface, text, application

Description automatically generated

Това е! Добра работа!

## Търсене

Имплементирайте един или няколко от следните алгоритми за търсене:

* Линейно търсене
* Търсене на фибоначи

**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 |