# Упражнения: Други алгоритми за сортиране

## Сортиране чрез сливане

Сортирайте масив от елементи с помощта на популярния алгоритъм за сортиране чрез сливане (Merge Sort)

### Примери

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

### Подсказки

Създайте клас Mergesort с един-единствен метод Sort



Създайте един **допълнителен масив,** който ще е от полза при сливането на подмасивите



Сега, за реализацията на метода Merge()



Докато се сортират двата подмасива, ако **най-големият елемент отляво** е по-малък от **най-малкия елемент отдясно**, двата подмасива са **вече слети**



Ако обаче не са, **прехвърляме всички елементи в помощния масив**



После **ги сливаме обратно** в главния масив



Сега за създаването на рекурсивен метод Sort()



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



Ако обаче не е, трябва да **го разделите на два подмасива и да ги сортирате рекурсивно** и после да ги **слеете при свиването** на рекурсията (при последействията, след рекурсивното извикване)



Сега вече може да извикате метода Sort()



## Бързо сортиране

Сортирайте един масив от елементи с помощта на популярния алгоритъм за бързо сортиране (QuickSort)

### Примери

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

### Подсказки

Можете да научите повече за алгоритъма QuickSort от [Уикипедия](https://bg.wikipedia.org/wiki/%D0%91%D1%8A%D1%80%D0%B7%D0%BE_%D1%81%D0%BE%D1%80%D1%82%D0%B8%D1%80%D0%B0%D0%BD%D0%B5). Страхотен инструмент за визуализиране на този алгоритъм (а и на много други) ще намерите на [Visualgo.net](http://visualgo.net/sorting.html).

Алгоритъмът накратко:

* Бързото сортиране взима несортирани части на масив и ги сортира
* Избираме **опорен елемент**
  + Избираме първият елемент от несортираната част и го местим по такъв начин, че всички по-малки елементи да са му отляво и всички по-големи - отдясно
* С опорния елемент на правилната позиция, сега имаме две несортирани части - една отляво и една отдясно
* **Извикваме рекурсивно процедурата** за всяка част
* Дъното на рекурсията е когато частта е с размер 1 елемент, който по дефиниция е сортиран

Първо, дефинирайте **класът** и неговият **метод за сортиране**:



Сега реализирайте частния метод Sort(). Не забравяйте да се погрижите за **дъното на рекурсията**



Първо, открийте „опорния“ елемент и пренаредете елементите, после сортирайте рекурсивно лявата и дясната част:



Сега, за да изберем опорния елемент... трябва да създадем метод, наречен Partition()



Ако има **само един елемент**, масивът е вече разделен и индекса на опорния му елемент е индексът на единственият му елемент



Намиране на опорния елемент включва **пренареждане на всички елементи** в частта, така че да се удовлетвори условието **всички елементи наляво от опорния да са по-малки** от него и **всички елементи отдясно да са по-големи** от него



## 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 най-разбираема и защо.

## Брой на инверсиите

Приемаме, че брой на инверсиите е колко далеч (или близо) е масивът от това да бъде сортиран. Ако списъкът е вече сортиран, тогава броят на инверсиите е 0. Ако списъкът е сортиран в обратен ред, тогава броят на инверсиите е в своя максимум.

Два елемента a[i] и a[j] формират инверсия ако a[i] > a[j] и i < j.

Открийте и **отпечатайте броят на всички инверсии** в даден масив от входящи данни.

### Примери

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Инверсии** |
| 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 |

### Подсказки

Използвайте модифицирана версия на сортиране чрез сливане.

Полезно четиво: <http://www.geeksforgeeks.org/counting-inversions/>

## Най-често срещано число

Да се напише програма, която в масив от непоредни положителни двуцифрени числа намира числото, което се среща най-често в масива и извежда кое е то и колко пъти се среща. Ако повече от едно число се срещат максимален брой пъти, извежда най-голямото от най-често повтаряните числа.

### Вход

* Входните данни трябва да се прочетат от конзолата.
* На първия и единствен ред се подават неподредени положителни цели двуцифрени числа, отделени едно от друго с интервал.
* Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

### Изход

* Изходните данни трябва да бъдат отпечатани на конзолата.
* На първия ред трябва да бъдат изведени две числа, разделени с интервал - **броят повторения** на най-голямото от най-често срещаните числа и **самото това число**.

### Примери

|  |  |  |
| --- | --- | --- |
| **Вход** | **Изход** | **Коментари** |
| 10 13 10 99 | 2 10 | Най-често се повтаря 10 - 2 пъти. |
| 10 13 99 10 99 | 2 99 | И 10, и 99 се срещат по 2 пъти, но 99 е по-голямо от 10. |
| 10 13 99 13 99 13 | 3 13 | 13 се повтаря най-много - 3 пъти. |

### Подсказки

Използвайте модифицирана версия на алгоритъма за сортиране чрез броене.

## Най-бърз подреждач

Напишете програма, която **сравнява бързодействието** на различните алгоритми за сортиране в четири кръга и определя кой е победителят за всеки кръг, както и финалния победител (този с най-малко време общо за четирите кръга). Използвайте два или повече от следните алгоритми за сортиране:

* Сортиране чрез вмъкване
* Метод на мехурчето
* Алгоритъм на Шел
* Сортиране чрез сливане
* Бързо сортиране
* Bucket сортиране

Във всеки кръг, всеки от алгоритмите за сортиране трябва да бъде тестван върху един от следните четири масива:

1. Масив от **поредни числа от 1 до N**, т.е подредени в нарастващ ред (1, 2, 3, … N)
2. Масив от **поредни числа от N до 1**, т.е подредени в намаляващ ред (N, N-1, N-2, … 3, 2, 1)
3. Масив от **N случайни числа**, всяко в диапазона **от 1 до N** включително.
4. Масив от N числа, съдържащ **многократно повторение** на **числата от 1 до 10** до запълване на масива.

Тестването ще става чрез **сортирането на копие на масива в нарастващ ред** с помощта на всеки от алгоритмите. Засича се **времето** за сортиране на целия масив. Тествайте с различно големи масиви. Коментирайте в клас получените резултати за различните методи.

### Вход

* Входните данни трябва да се прочетат от конзолата.
* На първия и единствен ред ще ви бъде подадено число N, указващо броят на числата в масива, върху който ще тествате методите за сортиране.
* Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

### Изход

* Изходните данни трябва да бъдат отпечатани на конзолата.
* На четири реда трябва да бъдат изведени от 2 до 6 числа, разделени с интервал, съобразно броят на реализираните алгоритми за сортиране. Всяко число представлява времето на съответния алгоритъм сортиране на масива, използван в този кръг.
* Под тях изведете общото време за изпълнението на 4-те горни кръга за всеки от тестваните алгоритми.

## Топове плат

Студенти решили в свободното си време да работят като общи работници в голям склад за платове. Складът разполага с голямо количество платове от най-различни десени. Когато дойде някой клиент, той си поръчва желаното количество плат от някой десен, този плат се развива от топа, отрязва се и се предава на клиента, а останалото от топа се връща в склада. За да се оптимизира работата в склада, управителят на склада помолил студентите да сортират платовете по дължина - първо тези с останал 1 метър плат, после тези с 2 метра и т.н. Те решили да напишат програма, която да работи за всички подобни складове и да ги улесни работата на работниците в тях. Проблемът е, че складовете и работниците в тях може да са различни: в някои складове платовете са прекалено фини и нежни, а работниците груби и неуки и измерването и сравняването на дължините на платовете е най-тежката и бавна операция, а в други складове платовете са груби и на големи топове, а работниците слаби и хилави и пренасянето на топовете от едно място на друго е най-тежката и бавна операция. Направете програма, която да помага да се сортират платовете максимално бързо и с цената на минимум усилие. Сравнете в клас кой какъв метод е използвал и какъв резултат е получил за едни и същи входни данни. Уверете се, че има поне двама, които ползват един и същ метод за сравнение. Обсъдете получените резултати.

### Вход

* Входните данни трябва да се прочетат от конзолата.
* На първия ред се подават неподредени цели числа, отделени едно от друго с интервал. Всяко число указва дължината на плата в някой от топовете в склада.
* На втория ред се подават две цели числа, указващи условното **време за сравняване** на два топа **Tc** и **времето за размяна** на два топа **Ts**.
* На третия ред се подават две цели числа, указващи условното **усилие за сравняване** на два топа **Ec** и **усилието за размяна** на два топа **Es**.
* Входните данни винаги ще са валидни и в описания формат. Не е необходимо да бъдат изрично проверявани.

### Изход

* Изходните данни трябва да бъдат отпечатани на конзолата.
* На първия ред трябва да бъдат изведени две числа - **броят сравнения CC** и **броят размествания CS** за този метод и този масив.
* На вторият ред трябва да бъдат изведени две числа:
  + **общото време за сортиране на масива** (равно на Tc \* CC + Ts \* CS)
  + **общото усилие за сортиране на масива** (равно на Еc \* CC + Еs \* CS)

### Подсказки

Модифицирайте методите **Less( )** и **Swap( )**, така че да натрупвате броят на сравняванията и размените за всеки алгоритъм.