Exercise Session n. 2

Clone the following repository: **https://github.com/jindosanda/algo_tests** and place your programs inside the exercises folder under the appropriate session folder (session_2/exercises)
Read the README files to run the tests!

Let's solve some simple programming exercises.

The problems go from easiest to hardest, let's code!

Count Lower

Write a function `count_lower(A,x)` that returns the number of elements in A whose value is less than x.

Examples

```
>>> count_lower([0,7,10,-2,3], 7)
3
>>> count_lower([3, 2, 1, 1, 3, 2, 2, 3, 1], 2)
3
>>> count_lower([3, 2, 1, 1, 3, 2, 2, 3, 1], 10)
9
>>> count_lower([], 1000000000000000)
0
>>> count_lower([2, 3, 4, 5], 1)
0
```

Leap Year

Write a function `leap_year(y)` that, given a year number y in the Gregorian calendar, return `True` if y is a leap year, or `False` otherwise. Recall that a leap year is one whose number is divisible by 4, excluding the year numbers divisible by 100, but including the year numbers divisible by 400.

Examples

```
>>> leap_year(2000)
True
>>> leap_year(1969)
False
>>> leap_year(2023)
False
>>> leap_year(1984)
True
>>> leap_year(2022)
False
>>> leap_year(2200)
False
>>> leap_year(2400)
True
>>> leap_year(1900)
False
```

Multiples of Three

Write a function `multiples_of_three(A)` that takes an array A of integers and prints the count of all the elements of A that are multiples of three.

Examples

```
>>> multiples_of_three([34, 31, 45, 5, 38, 19, 19, 26, 25, 19,
2
>>> multiples_of_three([7, 2, 0])
0
```

Check Sorted

Write a function `check_sorted(A)` that returns `True` if the given sequence A is sorted in non-decreasing order, or `False` otherwise.

Examples

```
>>> check_sorted([1,2,3])
True
>>> check_sorted([1,3,2])
False
>>> check_sorted([])
True
```

```
>>> check_sorted([7])
True
>>> check_sorted([50,50,50])
True
>>> check_sorted([43,51,51])
True
>>> check_sorted([43,51,50,51,70])
False
```

Find the Peak

Write an algorithm `find_peak(A)` that, given a "peak" sequence A, finds the maximal value x in A. A "peak" sequence A consists of an increasing sequence $A_{1,...,i}$ attached to a decreasing sequence $A_{i,...,n}$, for some index i. These two sequences share exactly one element, namely A_i.

Examples

```
>>> find_peak([1,2,3])
3
>>> find_peak([1,2,7,8,9])
9
>>> find_peak([9,6,5])
9
>>> find_peak([1,2,1,-3])
2
>>> find_peak([1,2,1,0,-3])
2
>>> find_peak([1,2,3,2,1])
3
>>> find_peak([1,2,3,4])
4
>>> find_peak([1,2])
2
>>> find_peak([4,7,4])
7
>>> find_peak([7])
7
>>> find_peak([7,4])
7
>>> find_peak([4,7])
7
```

Partition Even/Odd

Write a function `partition_even_odd(A)` that takes an array of integers A and sorts the elements of A so that all the even elements precede all the odd elements. The function must sort A *in-place*. This means that it must operate directly on A just by swapping elements, without using an additional array.

Examples

```
>>> A = [-1,1,7,5,-2,1,2,7,7,5,5,1,1,4,1]
>>> partition_even_odd(A)
>>> print(A)
[-2,2,4,-1,1,7,5,1,7,7,5,5,1,1,1]
```

Notice that in general the solution is not unique. For example, the following result would be equally correct:

```
>>> A = [-1,1,7,5,-2,1,2,7,7,5,5,1,1,4,1]
>>> partition_even_odd(A)
>>> print(A)
[2,4,-2,5,1,5,1,7,7,7,-1,1,1,5,1]
```