* Given an array of integers, calculate the ratios of its elements that are *positive*, *negative*, and *zero*. Print the decimal value of each fraction on a new line with 6 places after the decimal.

**Function Description:** Complete the *plusMinus* function in the editor below.

plusMinus has the following parameter(s):

* *int arr[n]*: an array of integers

**Print:** Print the ratios of positive, negative and zero values in the array. Each value should be printed on a separate line with 6 digits after the decimal. The function should not return a value.

**Input Format:** The first line contains an integer,n , the size of the array.  
The second line contains n space-separated integers that describe arr[n].

**Output Format: Print** the following 3 lines, each to 6 decimals:

1. proportion of positive values
2. proportion of negative values
3. proportion of zeros

**Sample Input:** STDIN Function

----- --------

6 arr[] size n = 6

-4 3 -9 0 4 1 arr = [-4, 3, -9, 0, 4, 1]

**Sample Output**: 0.500000

0.333333

0.166667

Ans:

def plusMinus(arr):

    pos=sum(1 for i in arr if i>0)/len(arr)

    neg=sum(1 for i in arr if i<0)/len(arr)

    zero=sum(1 for i in arr if i==0)/len(arr)

    print(f'{pos:.6f}',f'{neg:.6f}',f'{zero:.6f}',sep="\n")

if \_\_name\_\_ == '\_\_main\_\_':

    n=int(input().strip())

    arr=list(map(int,input().split()))

    plusMinus(arr)

* Given five positive integers, find the minimum and maximum values that can be calculated by summing exactly four of the five integers. Then print the respective minimum and maximum values as a single line of two space-separated long integers.

**Function Description:** Complete the *miniMaxSum* function in the editor below.

miniMaxSum has the following parameter(s):

* *arr*: an array of 5 integers

**Print:** Print two space-separated integers on one line: the minimum sum and the maximum sum of 4 of 5 elements.

**Input Format:** A single line of five space-separated integers.

**Output Format:** Print two space-separated long integers denoting the respective minimum and maximum values that can be calculated by summing exactly *four* of the five integers. (The output can be greater than a 32 bit integer.)

**Sample Input:** 1 2 3 4 5

**Sample Output:** 10 14

Ans:

def miniMaxSum(arr):

    print(sum(sorted(arr)[:-1]),sum(sorted(arr)[1:]))

if \_\_name\_\_ == '\_\_main\_\_':

    arr=list(map(int,input().split()))

    miniMaxSum(arr)

* Given a time in 12[-hour AM/PM format](https://en.wikipedia.org/wiki/12-hour_clock), convert it to military (24-hour) time.

Note: - 12:00:00AM on a 12-hour clock is 00:00:00 on a 24-hour clock.  
- 12:00:00PM on a 12-hour clock is 12:00:00 on a 24-hour clock.

**Function Description:** Complete the *timeConversion* function in the editor below. It should return a new string representing the input time in 24 hour format.

timeConversion has the following parameter(s):

* *string s*: a time in 12 hour format

**Returns:** *string*: the time in 24 hour format

**Input Format:** A single string s that represents a time in 12-hour clock format (i.e.:hh:mm:ssAM or hh:mm:ssPM).

**Sample Input:** 07:05:45PM

**Sample Output:** 19:05:45

Ans:

import os

def timeConversion(s):

    h=s[0:2]

    if s[-2:]=='PM' and h!="12":h=str(int(h)+12)

    elif s[-2:]=='AM' and h=="12":h="00"

    return(h+s[2:-2])

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'],'w')

    result=timeConversion(input())

    fptr.write(result+'\n')

    fptr.close()

* Given an array of integers, where all elements but one occur twice, find the unique element.

**Example:** a=[1,2,3,4,3,2,1]

The unique element is 4.

**Function Description:** Complete the *lonelyinteger* function in the editor below.

lonelyinteger has the following parameter(s):

* *int a[n]*: an array of integers

**Returns:** *int:* the element that occurs only once

**Input Format:** The first line contains a single integer, n, the number of integers in the array.  
The second line contains n space-separated integers that describe the values in a.

Ans:

import os

from collections import Counter

def lonelyinteger(a):

    c=Counter(a)

    return(next(k for k,v in c.items() if v==1))

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())

    a = list(map(int, input().rstrip().split()))

    result = lonelyinteger(a)

    fptr.write(str(result) + '\n')

    fptr.close()

* Given a square matrix, calculate the absolute difference between the sums of its diagonals.

For example, the square matrix arr is shown below:

1 2 3

4 5 6

9 8 9

The left-to-right diagonal =1+5+9=15 . The right to left diagonal =3+5+9=17 . Their absolute difference is |15-17|=2.

**Function description:** Complete the diagonalDifference function in the editor below.

diagonalDifference takes the following parameter:

* *int arr[n][m]*: an array of integers

**Return:** *int*: the absolute diagonal difference

**Input Format:** The first line contains a single integer, n, the number of rows and columns in the square matrix arr.  
Each of the next n lines describes a row, arr[i], and consists of n space-separated integers arr[i][j].

**Output Format:** Return the absolute difference between the sums of the matrix's two diagonals as a single integer.

**Sample Input:** 3

11 2 4

4 5 6

10 8 -12

**Sample Output:** 15

Ans:

import os

def diagonalDifference(arr):

    lds=sum(arr[i][i] for i in range(len(arr)))

    rds=sum(arr[i][len(arr)-i-1] for i in range(len(arr)))

    return abs(lds-rds)

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())

    arr = []

    for \_ in range(n):

        arr.append(list(map(int, input().rstrip().split())))

    result = diagonalDifference(arr)

    fptr.write(str(result) + '\n')

    fptr.close()

* **Comparison Sorting:** Quicksort usually has a running time of n\*log(n), but is there an algorithm that can sort even faster? In general, this is not possible. Most sorting algorithms are *comparison sorts*, i.e. they sort a list just by comparing the elements to one another. A comparison sort algorithm cannot beat n\*log(n) (worst-case) running time, since n\*log(n) represents the minimum number of comparisons needed to know where to place each element. For more details, you can see [these notes](http://www.cs.cmu.edu/~avrim/451f11/lectures/lect0913.pdf) (PDF).

**Alternative Sorting:** Another sorting method, the *counting sort*, does not require comparison. Instead, you create an integer array whose index range covers the entire range of values in your array to sort. Each time a value occurs in the original array, you increment the counter at that index. At the end, run through your counting array, printing the value of each non-zero valued index that number of times.

**Challenge:** Given a list of integers, count and return the number of times each value appears as an array of integers.

**Function Description:** Complete the *countingSort* function in the editor below.

countingSort has the following parameter(s):

* *arr[n]:* an array of integers

**Returns:** *int[100]:* a frequency array

**Input Format:** The first line contains an integer n, the number of items in arr.  
Each of the next n lines contains an integer arr[i] where 0<=i<n.

**Sample Input:** 100

63 25 73 1 98 73 56 84 86 57 16 83 8 25 81 56 9 53 98 67 99 12 83 89 80 91 39 86 76 85 74 39 25 90 59 10 94 32 44 3 89 30 27 79 46 96 27 32 18 21 92 69 81 40 40 34 68 78 24 87 42 69 23 41 78 22 6 90 99 89 50 30 20 1 43 3 70 95 33 46 44 9 69 48 33 60 65 16 82 67 61 32 21 79 75 75 13 87 70 33

**Sample Output**: 0 2 0 2 0 0 1 0 1 2 1 0 1 1 0 0 2 0 1 0 1 2 1 1 1 3 0 2 0 0 2 0 3 3 1 0 0 0 0 2 2 1 1 1 2 0 2 0 1 0 1 0 0 1 0 0 2 1 0 1 1 1 0 1 0 1 0 2 1 3 2 0 0 2 1 2 1 0 2 2 1 2 1 2 1 1 2 2 0 3 2 1 1 0 1 1 1 0 2 2

**Ans:**

from collections import Counter

import os

def countingSort(arr):

    c=Counter(arr)

    result=[]

    for i in range(100):result.append(c[i])

    return result

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())

    arr = list(map(int, input().rstrip().split()))

    result = countingSort(arr)

    fptr.write(' '.join(map(str, result)))

    fptr.write('\n')

    fptr.close()

* **Find the Median:** The median of a list of numbers is essentially its middle element after sorting. The same number of elements occur after it as before. Given a list of numbers with an odd number of elements, find the median?

**Function Description:** Complete the findMedian function in the editor below.

findMedian has the following parameter(s): int arr[n]: an unsorted array of integers

**Returns**: int: the median of the array

Ans:

def findMedian(arr):

import numpy as np

return(int(np.median(arr)))

* Sean invented a game involving a 2n\*2n matrix where each cell of the matrix contains an integer. He can reverse any of its rows or columns any number of times. The goal of the game is to maximize the sum of the elements in the n\*n submatrix located in the upper-left quadrant of the matrix.

Given the initial configurations for q matrices, help Sean reverse the rows and columns of each matrix in the best possible way so that the sum of the elements in the matrix's upper-left quadrant is maximal.

**Function Description:** Complete the *flippingMatrix* function in the editor below.

flippingMatrix has the following parameters:  
- *int matrix[2n][2n]:* a 2-dimensional array of integers

**Returns:** - *int:* the maximum sum possible.

**Input Format:** The first line contains an integer q, the number of queries.

The next q sets of lines are in the following format:

* The first line of each query contains an integer, n.
* Each of the next 2n lines contains 2n space-separated integers matrix[i][j] in row i of the matrix.

**Sample Input:** STDIN Function

----- --------

1 q = 1

2 n = 2

112 42 83 119 matrix = [[112, 42, 83, 119], [56, 125, 56, 49],

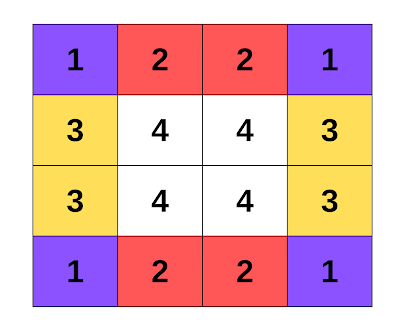
56 125 56 49 [15, 78, 101, 43], [62, 98, 114, 108]]

15 78 101 43

62 98 114 108

**Sample Output**: 414

Note: Here we can find solution using following pattern,



Maximise the indexes which have same color

So simply we have to find Max of same number of box like (1,1,1,1). And last return Sum of all Max numbers.

Ans:

def flippingMatrix(matrix):

nrows=len(matrix)

return sum([max([matrix[i][j],matrix[nrows-1-i][j],matrix[i][nrows-1- j],matrix[nrows-1-i][nrows-1-j]]) for i in range(nrows//2) for j in range(nrows//2)])

* Julius Caesar protected his confidential information by encrypting it using a cipher. [Caesar's cipher](https://en.wikipedia.org/wiki/Caesar_cipher) shifts each letter by a number of letters. If the shift takes you past the end of the alphabet, just rotate back to the front of the alphabet. In the case of a rotation by 3, w, x, y and z would map to z, a, b and c.

Original alphabet: abcdefghijklmnopqrstuvwxyz

Alphabet rotated +3: defghijklmnopqrstuvwxyzabc

**Note:** The cipher *only* encrypts letters; symbols, such as -, remain unencrypted.

**Function Description:** Complete the *caesarCipher* function in the editor below.

caesarCipher has the following parameter(s):

* *string s*: cleartext
* *int k*: the alphabet rotation factor

**Returns:** *string:* the encrypted string

**Input Format:** The first line contains the integer, n, the length of the unencrypted string.  
The second line contains the unencrypted string, s.  
The third line contains k, the number of letters to rotate the alphabet by.

**Sample Input:** 11

middle-Outz

2

**Sample Output:** okffng-Qwvb

Ans:

import os

def caesarCipher(s,k):

    result=[]

    for char in s:

        ascii\_offset=65 if char.isupper() else 97

        result.append(chr((ord(char)-ascii\_offset+k)%26+ascii\_offset) if char.isalpha() else char)

    return ''.join(result)

if \_\_name\_\_ == '\_\_main\_\_':

    fptr = open(os.environ['OUTPUT\_PATH'], 'w')

    n = int(input().strip())

    s = input()

    k = int(input().strip())

    result = caesarCipher(s, k)

    fptr.write(result + '\n')

    fptr.close()