**HACKER RANK PROBLEMS**

# 1)Encryption

An English text needs to be encrypted using the following encryption scheme.  
First, the spaces are removed from the text. Let  be the length of this text.  
Then, characters are written into a grid, whose rows and columns have the following constraints:

For example, the sentence , after removing spaces is  characters long.  is between  and , so it is written in the form of a grid with 7 rows and 8 columns.

ifmanwas

meanttos

tayonthe

groundgo

dwouldha

vegivenu

sroots

* Ensure that
* If multiple grids satisfy the above conditions, choose the one with the minimum area, i.e. .

The encoded message is obtained by displaying the characters in a column, inserting a space, and then displaying the next column and inserting a space, and so on. For example, the encoded message for the above rectangle is:

imtgdvs fearwer mayoogo anouuio ntnnlvt wttddes aohghn sseoau

You will be given a message to encode and print.

**Function Description**

Complete the *encryption* function in the editor below. It should return a single string composed as described.

encryption has the following parameter(s):

* *s*: a string to encrypt

CODE:

**import** math

**import** os

**import** random

**import** re

**import** sys

*# Complete the encryption function below.*

**def** encryption(s):

        str1="".join(s.split())

**print**(str1)                        *#ashutoshdikshit*

        len1=**len**(str1)                     *# 15*

        sq=math.sqrt(len1)                 *#3*

        r=**int**(sq)                        *#row  =3*

        c=**int**(sq)+1                       *#coloumn=4*

        p=[(str1[i:i+c]) **for** i **in** **range**(0,**len**(str1),c)]

**print**("p",p)                             *#p ['ashu', 'tosh', 'diks', 'hit']*

        final=[]

**for** i **in** **range**(**len**(p)-1):

**print**(**len**(p)-1)

                fc=[]

**for** j **in** p:

*# print(j)*

                        fc.append(j[i])

        final.append(fc)

*#print(fc)*

**print**(final)

**if** \_\_name\_\_ == '\_\_main\_\_':

    s = **input**()

    result = encryption(s)

**2)Extra Long Factorials**

The *factorial* of the integer , written , is defined as:

Calculate and print the factorial of a given integer.

For example, if , we calculate  and get .

**Function Description**

Complete the *extraLongFactorials* function in the editor below. It should print the result and return.

extraLongFactorials has the following parameter(s):

* *n*: an integer

**Note:** Factorials of  can't be stored even in a  long long variable. Big integers must be used for such calculations. Languages like Java, Python, Ruby etc. can handle big integers, but we need to write additional code in C/C++ to handle huge values.

We recommend solving this challenge using BigIntegers.

CODE;

**import** math

**import** os

**import** random

**import** re

**import** sys

*# Complete the extraLongFactorials function below.*

**def** extraLongFactorials(n):

**print** (math.factorial(n))

**if** \_\_name\_\_ == '\_\_main\_\_':

    n = **int**(**input**())

    extraLongFactorials(n)

# 3)Bigger is Greater

[*Lexicographical order*](https://en.wikipedia.org/wiki/Lexicographical_order) is often known as alphabetical order when dealing with strings. A string is *greater* than another string if it comes later in a lexicographically sorted list.

Given a word, create a new word by swapping some or all of its characters. This new word must meet two criteria:

* It must be greater than the original word
* It must be the smallest word that meets the first condition

For example, given the word , the next largest word is .

Complete the function *biggerIsGreater* below to create and return the new string meeting the criteria. If it is not possible, return no answer.

**Function Description**

Complete the *biggerIsGreater* function in the editor below. It should return the smallest lexicographically higher string possible from the given string or no answer.

biggerIsGreater has the following parameter(s):

* *w*: a string
* **import** math
* **import** os
* **import** random
* **import** re
* **import** sys
* *# Complete the biggerIsGreater function below.*
* **def** biggerIsGreater(w):
* J = -1
* **for** i **in** **range**( **len**(w)-1, 0, -1 ):
* **for** j **in** **range**( i-1, J, -1 ):
* **if** w[i] > w[j] : I,J = i,j ; **break**
* **if** J == -1 : **return** "no answer"
* **return** w[:J]+w[I]+''.join(**sorted**(w[J:I]+w[I+1:]))
* **if** \_\_name\_\_ == '\_\_main\_\_':
* fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')
* T = **int**(**input**())
* **for** T\_itr **in** **range**(T):
* w = **input**()
* result = biggerIsGreater(w)
* fptr.write(result + '\n')
* fptr.close()

# 4)3D Surface Area

Madison, is a little girl who is fond of toys. Her friend Mason works in a toy manufacturing factory . Mason has a 2D board  of size  with  rows and  columns. The board is divided into cells of size  with each cell indicated by it's coordinate . The cell  has an integer  written on it. To create the toy Mason stacks  number of cubes of size  on the cell .

Given the description of the board showing the values of  and that the price of the toy is equal to the 3d surface area find the price of the toy.

**import** math

**import** os

**import** random

**import** re

**import** sys

*# Complete the surfaceArea function below.*

**def** surfaceArea(A):

        surf = 0

**for** i **in** **range**(**len**(A)):

**for** e **in** **range**(**len**(A[i])):

                        c = A[i][e]

                        s = (c\*4)+2

**if** i > 0:*.*

                                 s -= **min**(c,A[i-1][e])\*2

**if** e > 0:

                                s -= **min**(c,A[i][e-1])\*2

                        surf += s

**return** surf

**if** \_\_name\_\_ == '\_\_main\_\_':

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

    HW = **input**().split()

    H = **int**(HW[0])

    W = **int**(HW[1])

    A = []

**for** \_ **in** **range**(H):

        A.append(**list**(**map**(**int**, **input**().rstrip().split())))

    result = surfaceArea(A)

    fptr.write(**str**(result) + '\n')

    fptr.close()

# 5)Absolute Permutation

We define  to be a permutation of the first  natural numbers in the range . Let  denote the value at position  in permutation  using -based indexing.

 is considered to be an *absolute permutation* if  holds true for every .

Given  and , print the lexicographically smallest absolute permutation . If no absolute permutation exists, print -1.

For example, let  giving us an array . If we use  based indexing, create a permutation where every . If , we could rearrange them to :

pos[i] i |Difference|

3 1 2

4 2 2

1 3 2

2 4 2

**Function Description**

Complete the *absolutePermutation* function in the editor below. It should return an integer that represents the smallest lexicographically smallest permutation, or  if there is none.

absolutePermutation has the following parameter(s):

* *n*: the upper bound of natural numbers to consider, inclusive
* *k*: the integer difference between each element and its index

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

*# Complete the absolutePermutation function below.*

**def** absolutePermutation(n, k):

**if** k == 0:

**return** [i+1 **for** i **in** **range**(n)]

**elif** n % (2\*k) != 0 **or** 2\*k > n:

**return** [-1]

**return** [(i+1)+(1 **if** (i//k)%2==0 **else** -1)\*k **for** i **in** **range**(n)]

**if** \_\_name\_\_ == '\_\_main\_\_':

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

    t = **int**(**input**())

**for** t\_itr **in** **range**(t):

        nk = **input**().split()

        n = **int**(nk[0])

        k = **int**(nk[1])

        result = absolutePermutation(n, k)

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

        fptr.write('\n')

    fptr.close()

# 6) Larry's Array

Larry has been given a permutation of a sequence of natural numbers incrementing from  as an array. He must determine whether the array can be sorted using the following operation any number of times:

* Choose any  consecutive indices and rotate their elements in such a way that .

For example, if :

A rotate

[1,6,5,2,4,3] [6,5,2]

[1,5,2,6,4,3] [5,2,6]

[1,2,6,5,4,3] [5,4,3]

[1,2,6,3,5,4] [6,3,5]

[1,2,3,5,6,4] [5,6,4]

[1,2,3,4,5,6]

YES

On a new line for each test case, print YES if  can be fully sorted. Otherwise, print NO.

**Function Description**

Complete the *larrysArray* function in the editor below. It must return a string, either YES or NO.

larrysArray has the following parameter(s):

* *A*: an array of integers
* *#!/bin/python3*
* **import** math
* **import** os
* **import** random
* **import** re
* **import** sys
* *# Complete the larrysArray function below.*
* **def** larrysArray(A):
* **for** i **in** **range**(n-2):
* **for** x **in** **range**(n-2):
* c = 0
* **while** **True**:
* **if** A[x] == **min**(A[x], A[x+1], A[x+2]):
* **break**
* **if** c == 2:
* **return** 'NO'
* *#rotate elements*
* i, j, k = A[x:x+3]
* A[x:x+3] = k, i, j
* c += 1
* *#need to check if something like 2,4,3 on the end*
* **return** 'YES' **if** A[-3:] == **sorted**(A[-3:]) **else** 'NO'
* **if** \_\_name\_\_ == '\_\_main\_\_':
* fptr = **open**(os.environ['OUTPUT\_PATH'], 'w')
* t = **int**(**input**())
* **for** t\_itr **in** **range**(t):
* n = **int**(**input**())
* A = **list**(**map**(**int**, **input**().rstrip().split()))
* result = larrysArray(A)
* fptr.write(result + '\n')
* fptr.close()

# 7) Almost Sorted

Given an array of integers, determine whether the array can be sorted in *ascending order* using only one of the following operations one time.

1. Swap two elements.
2. Reverse one sub-segment.

Determine whether one, both or neither of the operations will complete the task. If both work, choose *swap*. For instance, given an array  either swap the  and , or reverse them to sort the array. Choose swap. The Output Format section below details requirements.

**Function Description**

Complete the *almostSorted* function in the editor below. It should print the results and return nothing.

almostSorted has the following parameter(s):

* *arr*: an array of integers
* *#!/bin/python3*
* **import** math
* **import** os
* **import** random
* **import** re
* **import** sys
* *# Complete the almostSorted function below.*
* **def** is\_sorted(arr):
* **for** i **in** **range**(1, **len**(arr)):
* **if** arr[i] < arr[i - 1]:
* **return** **False**
* **return** **True**
* **def** swap(arr, i, j):
* arr[i], arr[j] = arr[j], arr[i]
* **return** arr
* **def** almostSorted(arr):
* **if** is\_sorted(arr):
* **print**('yes')
* **return**

* **for** i **in** **range**(**len**(arr)-1):
* **if** arr[i] > arr[i+1]:
* **break**
* **for** j **in** **range**(**len**(arr) - 1, 0, -1):
* **if** arr[j] < arr[j-1]:
* **break**

* **if** is\_sorted(swap(arr[:], i, j)):
* **print**('yes')
* **print**('swap {} {}'.**format**(i+1, j+1))
* **return**
* **for** k **in** **range**(i, math.ceil((j+i)/2), 1):
* arr = swap(arr, k, i + j - k)
* **if** is\_sorted(arr):
* **print**('yes')
* **print**('reverse {} {}'.**format**(i+1, j+1))
* **return**
* **print**('no')
* **if** \_\_name\_\_ == '\_\_main\_\_':
* n = **int**(**input**())
* arr = **list**(**map**(**int**, **input**().rstrip().split()))
* almostSorted(arr)

# 8) The Full Counting Sort

In this challenge you need to print the string that accompanies each integer in a list sorted by the integers. If two strings are associated with the same integer, they must be printed in their original order so your sorting algorithm should be *stable*. There is one other twist. The first half of the strings encountered in the inputs are to be replaced with the character "".

Insertion Sort and the simple version of Quicksort are stable, but the faster in-place version of Quicksort is not since it scrambles around elements while sorting.

In this challenge, you will use counting sort to sort a list while keeping the order of the strings preserved.

For example, if your inputs are  you could set up a helper array with three empty arrays as elements. The following shows the insertions:

i string converted list

0 [[],[],[]]

1 a - [[-],[],[]]

2 b - [[-],[-],[]]

3 c [[-,c],[-],[]]

4 d [[-,c],[-,d],[]]

The result is then printed:  .

**Function Description**

Complete the *countSort* function in the editor below. It should construct and print out the sorted strings.

countSort has the following parameter(s):

* *arr*: a 2D array where each *arr[i]* is comprised of two strings: *x* and *s*.

**Note**: The first element of each , , must be cast as an integer to perform the sort.

**#include** **<**cmath**>**

**#include** **<**cstdio**>**

**#include** **<**vector**>**

**#include** **<**iostream**>**

**#include** **<**algorithm**>**

**using** **namespace** std;

**int** main() {

**long** **int** n;

    cin >> n;

    string ar[n];

**for**(**long** **int** i = 0; i < n/2; i++){

**int** x;

        cin >> x;

        string s;

        cin >> s;

        ar[x] = ar[x] + "-" + " ";

    }

**for**(**long** **int** i = n/2; i < n; i++){

**int** x;

        cin >> x;

        string s;

        cin >> s;

        ar[x] = ar[x] + s + " ";

    }

**for**(**int** i = 0; i < n; i++)

        cout << ar[i];

**return** 0;

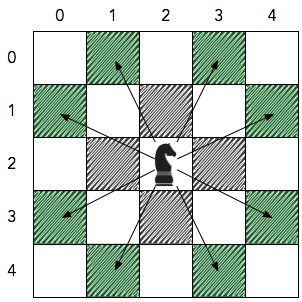
}

# 9) KnightL on a Chessboard

 is a chess piece that moves in an L shape. We define the possible moves of  as any movement from some position  to some  satisfying either of the following:

* and , or
* and

Note that  and  allow for the same exact set of movements. For example, the diagram below depicts the possible locations that  or  can move to from its current location at the center of a  chessboard:



Observe that for each possible movement, the Knight moves  units in one direction (i.e., horizontal or vertical) and  unit in the perpendicular direction.

Given the value of  for an  chessboard, answer the following question for each  pair where :

* What is the minimum number of moves it takes for  to get from position  to position ? If it's not possible for the Knight to reach that destination, the answer is -1 instead.

Then print the answer for each  according to the *Output Format* specified below.

*#!/bin/python3*

**import** math

**import** os

**import** random

**import** re

**import** sys

memo = [[]]

**def** adj(a, b, x, y, n):

**return** **filter**(**lambda** v : v[0] >= 0

**and** v[1] >= 0

**and** v[0] < n

**and** v[1] < n,

      [

        [x + a, y + b],

        [x + a, y - b],

        [x - a, y + b],

        [x - a, y - b],

        [x + b, y + a],

        [x + b, y - a],

        [x - b, y + a],

        [x - b, y - a],

    ])

**def** move(a, b, n):

**global** memo

*.*

**if** memo[a][b] **is** **not** **None**:

**return** memo[a][b]

    visited = [[**False**] \* n **for** \_ **in** **range**(n)]

    queue = [[0, 0, 0]]

**while** **len**(queue) > 0:

        x, y, l = queue.pop()

**if** x == n - 1 **and** y == n - 1:

            memo[a][b] = l

            memo[b][a] = l

**return** l

        neighbors = [

            [x\_i, y\_i]

**for** x\_i, y\_i **in** adj(a, b, x, y, n)

**if** visited[y\_i][x\_i] == **False**

        ]

**for** x\_i, y\_i **in** neighbors:

            visited[y\_i][x\_i] = **True**

            queue.insert(0, [x\_i, y\_i, l + 1])

    memo[a][b] = -1

    memo[b][a] = -1

**return** -1

**def** knightlOnAChessboard(n):

**global** memo

    memo = [[**None** **for** \_ **in** **range**(n + 1)] **for** \_ **in** **range**(n + 1)]

    results = []

**for** a **in** **range**(1, n):

        results.append([])

**for** b **in** **range**(1, n):

            results[a - 1].append(move(a, b, n))

**return** results

**if** \_\_name\_\_ == '\_\_main\_\_':

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

    n = **int**(**input**())

    result = knightlOnAChessboard(n)

    fptr.write('\n'.join([' '.join(**map**(**str**, x)) **for** x **in** result]))

    fptr.write('\n')

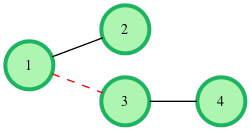
    fptr.close()

# 10) Even Tree

You are given a tree (a simple connected graph with no cycles).

Find the maximum number of edges you can remove from the tree to get a [forest](http://en.wikipedia.org/wiki/Tree_(graph_theory)) such that each connected component of the forest contains an even number of nodes.

As an example, the following tree with  nodes can be cut at most  time to create an even forest.



**Function Description**

Complete the *evenForest* function in the editor below. It should return an integer as described.

evenForest has the following parameter(s):

* *t\_nodes*: the number of nodes in the tree
* *t\_edges*: the number of undirected edges in the tree
* *t\_from*: start nodes for each edge
* *t\_to*: end nodes for each edge, (Match by index to *t\_from*.)
* **#include** **<**bits/stdc++.h**>**
* **using** **namespace** std;
* **#include** **<**iostream**>**
* **#include** **<**vector**>**
* **using** **namespace** std;
* **struct** Node{
* **int** num,root;
* };
* **int** main(){
* **int** N,M;
* **int** x,y;
* **int** count = 0;
* cin >> N >> M;
* vector <Node> nodes = vector<Node>(N);
* **for**(**int** i = 0;i < N;i++){
* nodes[i].num = 1;
* nodes[i].root = -1;
* }
* **for**(**int** i = 0;i < M;i++){
* cin >> x >> y;
* nodes[x-1].root = y-1;
* }
* **for**(**int** i = N-1;i > 0;i--)
* **if**(nodes[i].root >= 0)
* nodes[nodes[i].root].num += nodes[i].num;
* **for**(**int** i = 0;i < N;i++)
* **if**(nodes[i].root >= 0 && nodes[i].num%2== 0)
* count ++;
* cout << count;
* **return** 0;
* }