Name:N.RevathiReddy
Sec:04
ID:2100032517

DAA-SKILL-6

1.Greedy florist

A group of friends want to buy a bouquet of flowers. The florist wants to maximize his number of *new* customers and the money he makes. To do this, he decides he'll multiply the price of each flower by the number of that customer's previously purchased flowers plus . The first flower will be original price, , the next will be and so on.

Given the size of the group of friends, the number of flowers they want to purchase and the original prices of the flowers, determine the minimum cost to purchase all of the flowers. The number of flowers they want equals the length of the array.

Example

The length of , so they want to buy flowers total. Each will buy one of the flowers priced at the original price. Having each purchased flower, the first flower in the list, , will now cost . The total cost is .

Function Description

Complete the *getMinimumCost* function in the editor below.

getMinimumCost has the following parameter(s):

- *int c[n]:* the original price of each flower
- *int k:* the number of friends

Returns

- *int:* the minimum cost to purchase all flowers

Input Format

The first line contains two space-separated integers and , the number of flowers and the number of friends.

The second line contains space-separated positive integers , the original price of each flower.

Constraints

-
-
-
-

Sample Input 0

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
3 3
2 5 6
```

Sample Output 0

```
13
```

Explanation 0

There are flowers with costs and people in the group. If each person buys one flower, the total cost of prices paid is dollars. Thus, we print as our answer.

Sample Input 1

```
3 2
2 5 6
```

Sample Output 1

```
15
```

Explanation 1

There are flowers with costs and people in the group. We can minimize the total purchase cost like so:

1. The first person purchases flowers in order of decreasing price; this means they buy the more expensive flower () first at price dollars and the less expensive flower () second at price dollars.
2. The second person buys the most expensive flower at price dollars.

We then print the sum of these purchases, which is , as our answer.

Sample Input 2

```
5 3
1 3 5 7 9
```

Sample Output 2

```
29
```

Explanation 2

The friends buy flowers for , and , and for a cost of .

Code:

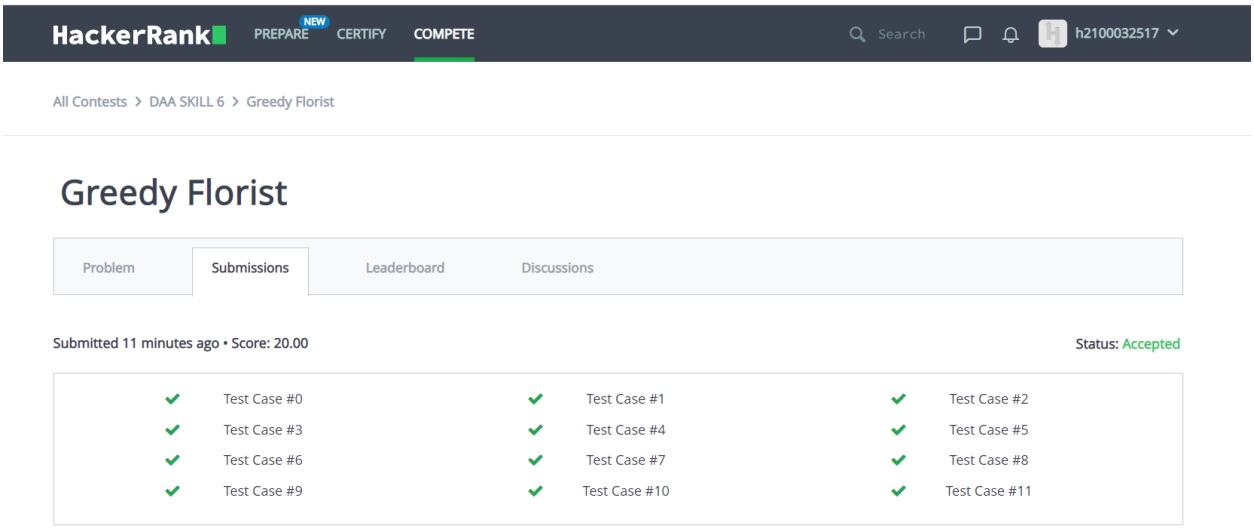
```
import math

import os
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
import random
import re
import sys
def getMinimumCost(k, c):
    c.sort(reverse=True)
    cost = 0
    previous_purchase = 0
    for i in range(n):
        cost += (previous_purchase +1) * c[i]
        if (i+1)%k==0:
            previous_purchase += 1
    return cost
if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')
    nk = input().split()
    n = int(nk[0])
    k = int(nk[1])
    c = list(map(int, input().rstrip().split()))
    minimumCost = getMinimumCost(k, c)
    fptr.write(str(minimumCost) + '\n')
    fptr.close()
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

output:



2.Max Min

You will be given a list of integers, , and a single integer . You must create an array of length from elements of such that its *unfairness* is minimized. Call that array . Unfairness of an array is calculated as

Where:

- *max* denotes the largest integer in
- *min* denotes the smallest integer in

Example

Pick any two elements, say .

Testing for all pairs, the solution provides the minimum unfairness.

Note: Integers in may not be unique.

Function Description

Complete the *maxMin* function in the editor below.

maxMin has the following parameter(s):

- *int k:* the number of elements to select
- *int arr[n]:*: an array of integers

Returns

- *int:* the minimum possible *unfairness*

Input Format

Name:N.RevathiReddy
Sec:04
ID:2100032517

The first line contains an integer , the number of elements in array .

The second line contains an integer .

Each of the next lines contains an integer where .

Constraints

Sample Input 0

```
7
3
10
100
300
200
1000
20
30
```

Sample Output 0

```
20
```

Explanation 0

Here ; selecting the integers , unfairness equals

```
max(10,20,30) - min(10,20,30) = 30 - 10 = 20
```

Sample Input 1

```
10
4
1
2
3
4
10
20
30
40
100
200
```

Sample Output 1

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
3
```

Explanation 1

Here ; selecting the integers , unfairness equals

```
max(1,2,3,4) - min(1,2,3,4) = 4 - 1 = 3
```

Sample Input 2

```
5
2
1
2
1
2
1
```

Sample Output 2

```
0
```

Explanation 2

Here . or give the minimum unfairness of .

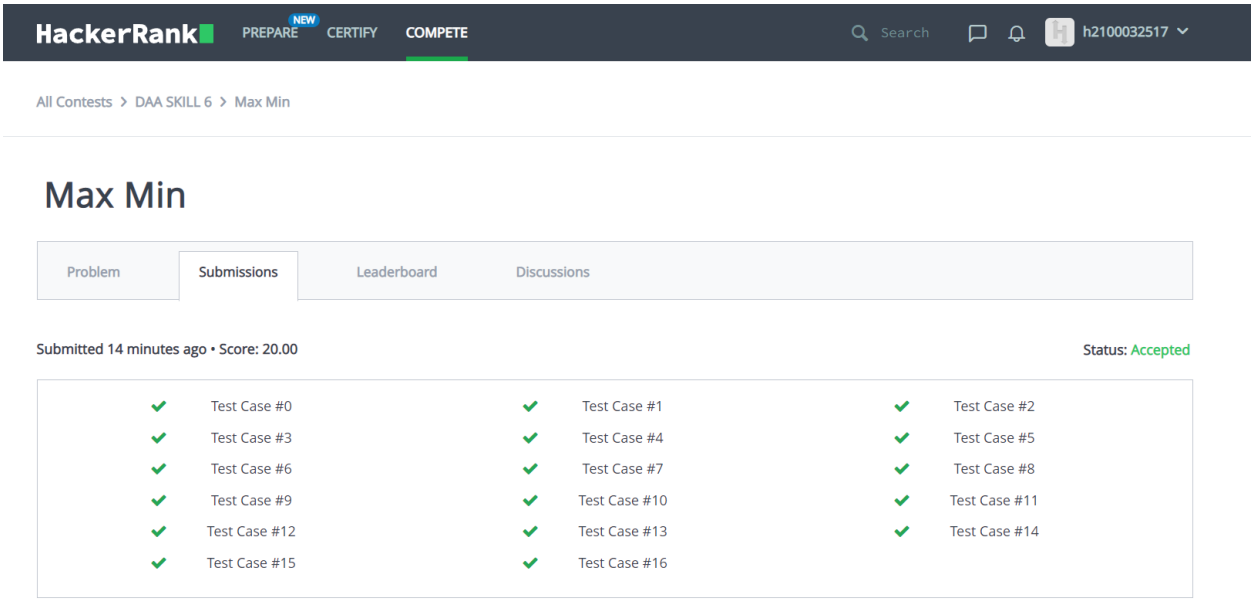
Code:

```
import math

import os

import random

import re

import sys

def maxMin(k, arr):

    arr.sort()

    result = arr[k-1] - arr[0]

    for i in range(n-k+1):

        if arr[i+k-1] - arr[i] < result:

            result = arr[i+k-1] - arr[i]

    return result
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
if __name__ == '__main__':
    fptr = open(os.environ['OUTPUT_PATH'], 'w')
    n = int(input())
    k = int(input())
    arr = []
    for _ in range(n):
        arr_item = int(input())
        arr.append(arr_item)
    result = maxMin(k, arr)
    fptr.write(str(result) + '\n')
    fptr.close()
```

output:



3.Goodland Electricity

Goodland is a country with a number of evenly spaced cities along a line. The distance between adjacent cities is unit. There is an energy infrastructure project planning meeting, and the government needs to know the fewest number of power plants needed to provide electricity to the entire list of cities. Determine that number. If it cannot be done, return -1.

Name:N.RevathiReddy
Sec:04
ID:2100032517

You are given a list of city data. Cities that may contain a power plant have been labeled . Others not suitable for building a plant are labeled . Given a distribution range of , find the lowest number of plants that must be built such that all cities are served. The distribution range limits supply to cities where distance is **less than k**.

Example

Each city is unit distance from its neighbors, and we'll use based indexing. We see there are cities suitable for power plants, cities and . If we build a power plant at , it can serve through because those endpoints are at a distance of and . To serve , we would need to be able to build a plant in city or . Since none of those is suitable, we must return -1. It cannot be done using the current distribution constraint.

Function Description

Complete the *pylons* function in the editor below.

pylons has the following parameter(s):

- *int k:* the distribution range
- *int arr[n]:* each city's suitability as a building site

Returns

- *int:* the minimum number of plants required or -1

Input Format

The first line contains two space-separated integers and , the number of cities in Goodland and the plants' range constant.
The second line contains space-separated binary integers where each integer indicates suitability for building a plant.

Constraints

-
- Each .

Subtask

- for of the maximum score.

Output Format

Print a single integer denoting the minimum number of plants that must be built so that all of Goodland's cities have electricity. If this is not possible for the given value of , print .

Name:N.RevathiReddy
Sec:04
ID:2100032517

Sample Input

```
STDIN       Function
-----       --------
6 2         arr[] size n = 6, k = 2
0 1 1 1 1 0   arr = [0, 1, 1, 1, 1, 0]
```

Sample Output

```
2
```

Explanation

Cities , , , and are suitable for power plants. Each plant will have a range of . If we build in cities cities, and , then all cities will have electricity.

Code:

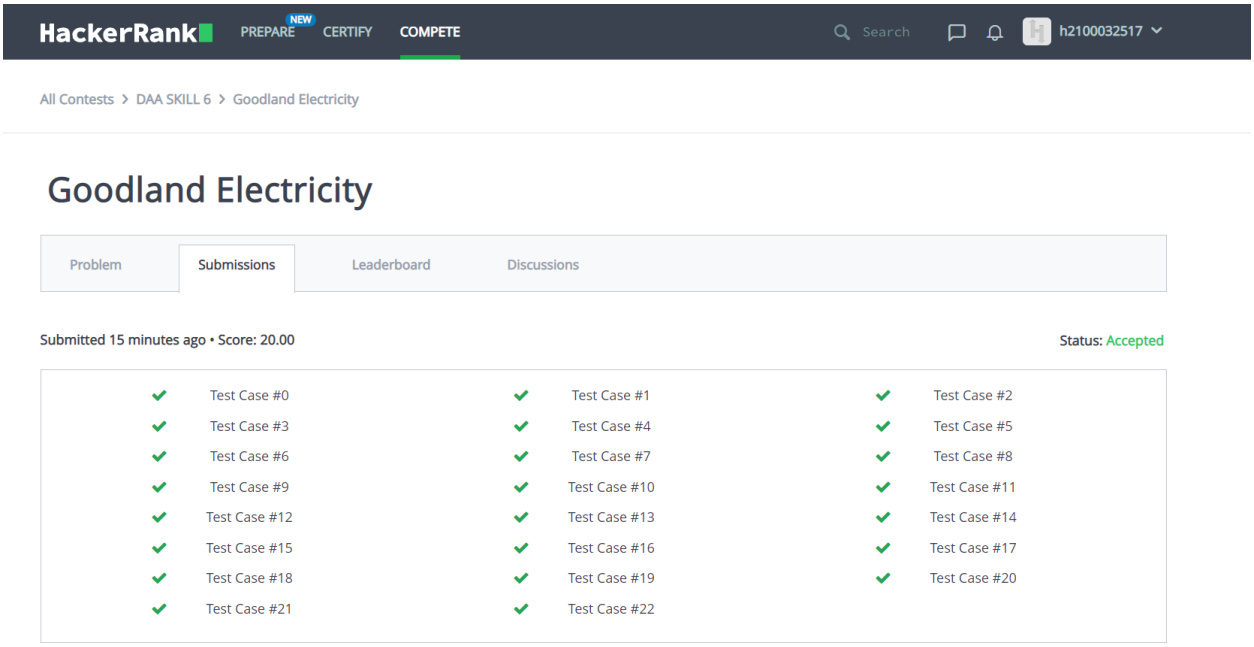
```
import math

def get_min(x,k,lights):
    best = -1
    best_ind=None
    for ind,i in enumerate(lights):
        #if abs(x - i)<k:
        if -k < x - i < k:
            best=i
            best_ind=ind
        elif i>x+k:
            break
    return best,best_ind
for qu in [1]:
    N,k = list(map(int,(input().strip().split(' '))))
    lights = list(map(int,(input().strip().split(' '))))
    assert(N==len(lights))
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
    lights = [i for i,val in enumerate(lights) if val==1]


    pos=0
    best=None
    best_ind=None
    count=0
    while 1:
        if pos>=N:
            break
        if best==lights[-1]:
            count=-1
            break
        if best_ind==None:
            best,best_ind = get_min(pos,k,lights)
        else:
            lights = lights[(best_ind+1):]
            best,best_ind = get_min(pos,k,lights)
        if best==-1:
            count=-1
            break
        count+=1
        pos = best + k
    print(count)
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

output:



4.Cloudy Day

Quibdó in Colombia is one among the cities that receive maximum rainfall in the world.

All year round, the city is covered in clouds. The city has many towns, located on a one-dimensional line. The positions and populations of each town on the number line are known to you. Every cloud covers all towns located at a certain distance from it. A town is said to be in *darkness* if there exists *at least* one cloud such that the town is within the cloud's range. Otherwise, it is said to be *sunny*.

Name:N.RevathiReddy
Sec:04

The city council has determined that they have enough money to remove *exactly one* cloud using their latest technology. Thus they want to remove the cloud such that the fewest number of people are left in darkness after the cloud is removed. What is the maximum number of people that will be in a sunny town after removing exactly one cloud?
Note: If a town is not covered by any clouds, then it is already considered to be sunny, and the population of this town must also be included in the final answer.
Complete the function `maximumPeople` which takes four arrays representing the populations of each town, locations of the towns, locations of the clouds, and the extents of coverage of the clouds respectively, and returns the maximum number of people that will be in a sunny town after removing exactly one cloud.

Input Format

The first line of input contains a single integer , the number of towns.
The next line contains space-separated integers . The integer in this line denotes the population of the town.
The next line contains space-separated integers denoting the location of the town on the one-dimensional line.
The next line consists of a single integer denoting the number of clouds covering the city.
The next line contains space-separated integers the of which denotes the location of the cloud on the coordinate axis.
The next line consists of space-separated integers denoting the range of the cloud.
Note: The range of each cloud is computed according to its location, i.e., the cloud is located at position and it covers every town within a distance of from it. In other words, the cloud covers every town with location in the range .

Constraints

Name:N.RevathiReddy
Sec:04
ID:2100032517

-
-
-

Output Format

Print a single integer denoting the maximum number of people that will be in a sunny town by removing exactly one cloud.

Sample Input 0

```
2
10 100
5 100
1
4
1
```

Sample Output 0

```
110
```

Explanation 0

In the sample case, there is only one cloud which covers the first town. Our only choice is to remove this sole cloud which will make all towns sunny, and thus, all people will live in a sunny town.

As you can see, the only cloud present, is at location on the number line and has a range , so it covers towns located at , and on the number line. Hence, the first town is covered by this cloud and removing this cloud makes all towns sunny.

Code:

import math

Name:N.RevathiReddy
Sec:04
ID:2100032517

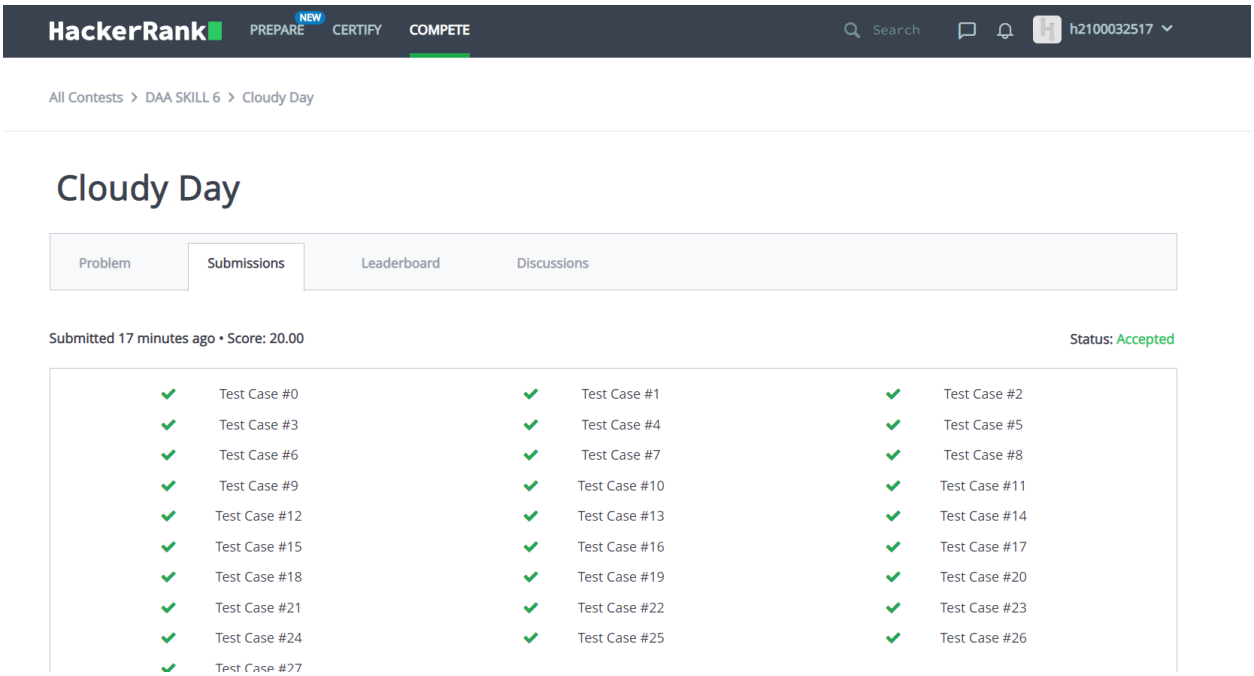
```
import os
import random
import re
import sys
from collections import defaultdict



def maximumPeople(towns, cloud_start, cloud_end):
    towns = sorted(towns)
    cloud_start = sorted(cloud_start)
    cloud_end = sorted(cloud_end)
    cloud_start_i = 0
    cloud_end_i = 0
    clouds = set()
    d = defaultdict(int)
    free = 0
    for town_i in range(len(towns)):
        town_x = towns[town_i][0]
        while cloud_start_i < len(cloud_start) and cloud_start[cloud_start_i][0] <= town_x:
            clouds.add(cloud_start[cloud_start_i][1])
            cloud_start_i += 1
        while cloud_end_i < len(cloud_end) and cloud_end[cloud_end_i][0] < town_x:
            clouds.remove(cloud_end[cloud_end_i][1])
            cloud_end_i += 1
        if len(clouds) == 1:
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
            towns[town_i][2] = list(clouds)[0]
            d[list(clouds)[0]] += towns[town_i][1]
        elif len(clouds) == 0:
            free += towns[town_i][1]
    return max(d.values(), default=0) + free
def main():
    n = int(input().strip())
    p = [int(x) for x in input().strip().split()]
    x = [int(x) for x in input().strip().split()]
    towns = [[xi, pi, -1] for xi, pi in zip(x, p)]
    m = int(input().strip())
    y = [int(x) for x in input().strip().split()]
    r = [int(x) for x in input().strip().split()]
    cloud_start = [[y[i]-r[i], i] for i in range(m)]
    cloud_end = [[y[i]+r[i], i] for i in range(m)]
    result = maximumPeople(towns, cloud_start, cloud_end)
    print(result)
if __name__ == "__main__":
    main()
```

output:

Name:N.RevathiReddy
Sec:04
ID:2100032517



5.The Power Sum

Find the number of ways that a given integer, , can be expressed as the sum of the powers of unique, natural numbers.

For example, if and , we have to find all combinations of unique squares adding up to . The only solution is .

Function Description

Complete the *powerSum* function in the editor below. It should return an integer that represents the number of possible combinations.

powerSum has the following parameter(s):

- *X*: the integer to sum to
- *N*: the integer power to raise numbers to

Input Format

The first line contains an integer .

The second line contains an integer .

Constraints

-
-

Output Format

Name:N.RevathiReddy
Sec:04
ID:2100032517

Output a single integer, the number of possible combinations caclulated.

Sample Input 0

```
10
2
```

Sample Output 0

```
1
```

Explanation 0

If and , we need to find the number of ways that can be represented as the sum of squares of unique numbers.
This is the only way in which can be expressed as the sum of unique squares.

Sample Input 1

```
100
2
```

Sample Output 1

```
3
```

Explanation 1

Sample Input 2

```
100
3
```

Sample Output 2

```
1
```

Explanation 2

 can be expressed as the sum of the cubes of .
. There is no other way to express as the sum of cubes.

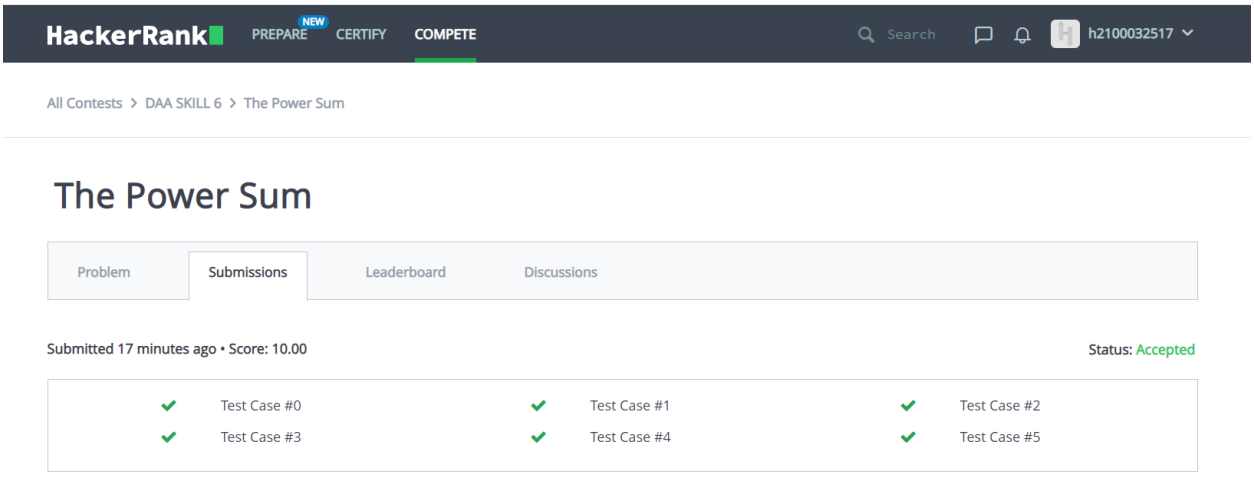
Code:

```
import sys

def powerSum(X, N, num):

    value = X - num**N
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
    if value < 0:
        return 0
    elif value == 0:
        return 1
    else:
        return powerSum(value, N, num+1) + powerSum(X, N, num+1)
if __name__ == "__main__":
    X = int(input().strip())
    N = int(input().strip())
    result = powerSum(X, N, 1)
    print(result)
```

output:



6.Password Cracker

There are *n* users registered on a website *CuteKittens.com*. Each of them has a unique password represented by *pass[1], pass[2], ..., pass[N]*. As this a very lovely site, many people want to access those awesomely cute pics of the kittens. But the adamant admin does not

Name:N.RevathiReddy
Sec:04
ID:2100032517
want the site to be available to the general public, so only those people who have passwords can access it.
Yu, being an awesome hacker finds a loophole in the password verification system. A string which is a *concatenation* of one or more passwords, in any order, is also accepted by the password verification system. Any password can appear or more times in that string. Given access to each of the passwords, and also have a string , determine whether this string be accepted by the password verification system of the website. If all of the string can be created by concatenating password strings, it is accepted. In this case, return the passwords in the order they must be concatenated, each separated by a single space on one line. If the password attempt will not be accepted, return 'WRONG PWASSWORD'.
Examples

Concatenate the passwords in index order to match 'abrakadabra'. Return 'abra ka dabra'.

Concatenate the passwords in index order to match 'kaabra'. Return 'ka abra'.

Concatenate the passwords in index order to match 'abba', to match 'baab', to match 'abab' or to match $baba'. No combination of 1 or more passwords can be concatenated to match 'aba'. Return 'WRONG PASSWORD'.
Function Description
Complete the *passwordCracker* function in the editor below.
passwordCracker has the following parameters:
- *string passwords[n]*: a list of password strings
- *string loginAttempt*: the string to attempt to create
Returns
- *string:* Return the passwords as a single string in the order required for the password to be accepted, each separated by a space. If it is not possible to form the string, return the string WRONG PASSWORD.

Input Format

The first line contains an integer *t*, the total number of test cases.

Name:N.RevathiReddy
Sec:04
ID:2100032517

Each of the next sets of three lines is as follows:

- The first line of each test case contains *n*, the number of users with passwords.
- The second line contains *n* space-separated strings, *passwords[i]*, that represent the passwords of each user.
- The third line contains a string, *loginAttempt*, which *Yu* must test for acceptance.

Constraints

-
-
-
- , where
-
- *loginAttempt* and *passwords[i]* contain only lowercase latin characters (*'a'-'z'*).

Sample Input 0

```
3
6
because can do must we what
wedowhatwemustbecausewecan
2
hello planet
helloworld
3
ab abcd cd
abcd
```

Sample Output 0

```
we do what we must because we can
WRONG PASSWORD
ab cd
```

Explanation 0

Sample Case #00: "wedowhatwemustbecausewecan" is the concatenation of passwords {"we", "do", "what", "we", "must", "because", "we", "can"}. That is
loginAttempt = pass[5] + pass[3] + pass[6] + pass[5] + pass[4] + pass[1] + pass[5] + pass[2]

Note that any password can repeat any number of times.

Sample Case #01: We can't create string "helloworld" using the strings {"hello", "planet"}.
Sample Case #02: There are two ways to create *loginAttempt ("abcd")*. Both pass[2] = "abcd" and pass[1] + pass[3] = "ab cd" are valid answers.

Name:N.RevathiReddy
Sec:04
ID:2100032517

Sample Input 1

```
3
4
ozkxyhkcst xvglh hpdnb zfzahm
zfzahm
4
gurwgrb maqz holpkhqx aowypvopu
gurwgrb
10
a aa aaa aaaa aaaaa aaaaaa aaaaaaa aaaaaaaa aaaaaaaaa aaaaaaaaaa
aaaaaaaaaab
```

Sample Output 1

```
zfzahm
gurwgrb
WRONG PASSWORD
```

Code:

```
import sys

sys.setrecursionlimit(5000)

testcases = int(input())

memo = {}


def solve(attempt, passwords, output, strLoc):
    global memo
    if strLoc not in memo and strLoc != len(attempt):
        fail_path = True
        for password in passwords:
            if len(password) <= (len(attempt) - strLoc):
                if password == (attempt[strLoc:(strLoc+len(password))]):
                    output.append(password)
                    output = solve(attempt, passwords, output, strLoc + len(password))
                    if len("".join(output)) == len(attempt):
```

Name:N.RevathiReddy
Sec:04
ID:2100032517

```
                fail_path = False
                break
        if fail_path:
            memo[strLoc] = True
    if strLoc in memo:
        output = output[:-1]
    return output
for i in range(testcases):
    memo = {}
    input()
    passwords = input().split(" ")
    attempt = input()
    output = solve(attempt, passwords, [], 0)
    if len("".join(output)) != len(attempt):
        print("WRONG PASSWORD")
    else:
        print(" ".join(output))
```

output:

Name:N.RevathiReddy
Sec:04
ID:2100032517