ASSIGNMENT PROBLEM SET

PROBLEM 1.

A sequence of numbers is called a zig-zag sequence if the differences between successive numbers strictly alternate between positive and negative.
The first difference (if one exists) may be either positive or negative. A sequence with fewer than two elements is trivially a zig-zag sequence.

For example, 1,7,4,9,2,5 is a zig-zag sequence because the differences (6,-3,5,-7,3) are alternately positive and negative. In contrast, 1,4,7,2,5 and
1,7,4,5,5 are not zig-zag sequences, the first because its first two differences are positive and the second because its last difference is zero.

Given a sequence of integers, sequence, return the length of the longest subsequence of sequence that is a zig-zag sequence. A subsequence is
obtained by deleting some number of elements (possibly zero) from the original sequence, leaving the remaining elements in their original order.

Definition

Class: ZigZag

Method: longestZigZag

Parameters: int[]

Returns: int

Method signature: int longestZigZag(int[] sequence)

(be sure your method is public)

Constraints

- sequence contains between 1 and 50 elements, inclusive.

- Each element of sequence is between 1 and 1000, inclusive.

Examples

1)

{ 1, 7, 4, 9, 2, 5 }

Returns: 6

The entire sequence is a zig-zag sequence.

2)

{ 1, 17, 5, 10, 13, 15, 10, 5, 16, 8 }

Returns: 7

There are several subsequences that achieve this length. One is 1,17,10,13,10,16,8.

3)

{ 44 }

Returns: 1

4)

{ 1, 2, 3, 4, 5, 6, 7, 8, 9 }

Returns: 2

5)

{ 70, 55, 13, 2, 99, 2, 80, 80, 80, 80, 100, 19, 7, 5, 5, 5, 1000, 32, 32 }

Returns: 8

6)

{ 374, 40, 854, 203, 203, 156, 362, 279, 812, 955,

600, 947, 978, 46, 100, 953, 670, 862, 568, 188,

67, 669, 810, 704, 52, 861, 49, 640, 370, 908,

477, 245, 413, 109, 659, 401, 483, 308, 609, 120,

249, 22, 176, 279, 23, 22, 617, 462, 459, 244 }

Returns: 36

****

PROBLEM 2.

Penguin Pals is a match making service that matches penguins to new friends, using the following procedure:

. Each penguin is asked a single question: "Do you prefer the color blue, or the color red?"
. All penguins are arranged so that they stand on a circle, equally spaced.
. The organizers draw some straight lines, connecting some pairs of penguins. Each penguin may only be connected to at most one other
penguin. Two penguins cannot be connected if they prefer a different color.
. Each penguin who is connected to some other penguin follows the line to find their match.

The only problem with the above system was that it allowed penguins to collide if two lines crossed each other. Therefore, a new additional rule was
adopted: no two lines may cross. Penguin Pals now has some penguins arranged on a circle (after step 2 of the above procedure). They need to know
the maximum number of pairs of penguins they can create.

You are given a String colors whose i-th character represents the prefered color of the i-th penguin (0-based index) in the circular arrangement. The i-
th character is 'R' if the i-th penguin prefers red and 'B' if the i-th penguin prefers blue. Return the maximum number of matched pairs that can be
formed.

Definition

Class: PenguinPals

Method: findMaximumMatching

Parameters: String

Returns: int

Method signature: int findMaximumMatching(String colors)

(be sure your method is public)

Constraints

- colors will contain between 1 and 50 characters, inclusive.

- Each character of colors will be either 'R' or 'B'.

Examples

1)

"RRBRBRBB"

Returns: 3

In this picture the penguins have been colored in their preferred color.

2)

"RRRR"

Returns: 2

3)

"BBBBB"

Returns: 2

4)

"RBRBRBRBR"

Returns: 4

5)

"RRRBRBRBRBRB"

Returns: 5

6)

"R"

Returns: 0

****

PROBLEM 3.

In this problem, all strings consist of uppercase English letters only. That is, there are 26 distinct letters.

The weight of a string S can be computed as follows: for each letter that occurs at least once in S, its leftmost and rightmost occurrences L and R are
found and the weight is increased by R-L.

For example, if S="ABCACAZ", the weight of S is (5-0) + (1-1) + (4-2) + (6-6) = 7. (The leftmost occurrence of 'A' is at the index L=0, the
rightmost one is at R=5, so 'A' contributes 5-0 = 5 to the weight of S. The only 'B' contributes 0, the pair of 'C's adds 2, and the only 'Z' adds 0.)

A string is S called light if no other string of the same length has a smaller weight.

You are given a int[] L. Manao is going to choose some light strings. The elements of L specify the lengths of these strings. For example, if L = { 3,
42, 1 }, Manao will first choose a light string of length 3, then a light string of length 42, and finally a light string of length 1.

Then, Manao is going to concatenate all of the chosen strings, in the given order. Compute and return the smallest possible weight of a string Manao
may obtain at the end.

Definition

Class: StringWeight

Method: getMinimum

Parameters: int[]

Returns: int

Method signature: int getMinimum(int[] L)

(be sure your method is public)

Constraints

- L will contain between 1 and 50 elements, inclusive.

- Each element of L will be between 1 and 100, inclusive.

Examples

1)

{1}

Returns: 0

Every string of length 1 has weight 0.

2)

{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}

Returns: 1

Manao is going to concatenate 27 strings of length 1. If Manao takes 25 distinct strings and 2 equal strings and places the equal strings side by side,
the weight of the resulting string will be 1.

3)

{26, 2, 2}

Returns: 8

One possible concatenation of minimum weight is "ABC...XYZ"+"YZ"+"YZ".

4)

{25, 25, 25, 25}

Returns: 1826

5)

{14, 6, 30, 2, 5, 61}

Returns: 1229

****

PROBLEM 4.

Fox Ciel was very bored, so she invented a single player game. The rules of the game are:

. You start with 0 points.
. You make exactly nA+nB moves.
. You have two types of moves available. These are called move A and move B.
. Exactly nA times you will make move A. Exactly nB times you will make move B. The moves can be in any order.

The moves affect your score in the following ways:

. Each time you make move A, you add scoreA to your score.
. Each time you make move B, you multiply your score by scoreB.
. You are given int nA, int nB, int paramA and int paramB. Calculate scoreA and scoreB as follows ("/" denotes exact division, without any
rounding):

scoreA = paramA/1000.0

scoreB = paramB/1000.0

Return the maximum possible score after nA+nB moves.

Definition

Class: FoxPlayingGame

Method: theMax

Parameters: int, int, int, int

Returns: double

Method signature: double theMax(int nA, int nB, int paramA, int paramB)

(be sure your method is public)

Notes

- The returned value must have an absolute or relative error less than 1e-9.

Constraints

- nA will be between 0 and 50, inclusive.

- nB will be between 0 and 50, inclusive.

- paramA will be between -10000 and 10000, inclusive.

- paramB will be between -2000 and 2000, inclusive.

Examples

1)

5

4

3000

2000

Returns: 240.0

scoreA = 3000/1000 = 3

scoreB = 2000/1000 = 2

The optimal order of operations is:

(3 + 3 + 3 + 3 + 3) * 2 * 2 * 2 * 2 = 240

2)

3

3

2000

100

Returns: 6.0

scoreA = 2000/1000 = 2

scoreB = 100/1000 = 0.1

Multiplying the score by scoreB decreases its absolute value, so it's better to do all multiplications before additions. Thus, the optimal order of
operations is:

0 * 0.1 * 0.1 * 0.1 + 2 + 2 + 2 = 6
****

PROBLEM 5.

Manao is the coach of a soccer team. There are N boys numbered from 0 to N-1 on the team. Each of them has some level of soccer skill. Manao
would like to divide the boys into two teams for today's training. The division should satisfy the following conditions:

. Each boy is assigned to a team.
. The total skill of first team is strictly greater than the total skill of the second team.
. For each player P in the first team: if we were to move P into the second team, the total skill of the first team would become strictly smaller
than the total skill of the second team.
. The total skill of a team is the sum of skills of the boys on that team.

You are given a int[] skill containing N elements. The i-th element in skill is the skill of boy i. Manao is interested in the number of ways to perform
such a division into teams. Please compute and return this number.

Definition

Class: YetAnotherTwoTeamsProblem

Method: count

Parameters: int[]

Returns: long

Method signature: long count(int[] skill)

(be sure your method is public)

Constraints

- skill will contain between 2 and 50 elements, inclusive.

- Each element of skill will be between 1 and 60,000, inclusive.

Examples

1)

{5, 4, 7, 6}

Returns: 2

The two possible divisions are:

Boys 0 and 2 against boys 1 and 3.

Boys 2 and 3 against boys 0 and 1.

2)

{1, 1, 1, 1, 1}

Returns: 10

Manao can put any three boys on the first team.

3)

{1, 2, 3, 5, 10}

Returns: 5

The possible choices of the first team are:

Boys 0 and 4.

Boys 1 and 4.

Boys 2 and 4.

Boys 3 and 4.

Boys 0, 1, 2 and 3.

4)

{1, 2, 3, 4, 10}

Returns: 0

5)

{999, 999, 999, 1000, 1000, 1001, 999, 1000,

1000, 1000, 1000, 999, 1000, 512, 511, 1001, 1001, 1001, 1001, 1001, 1000}

Returns: 17672631900

****

PROBLEM 6.

There are some cities and some roads connecting them together. The road network has the topology of a perfect binary tree (see below for a picture),
in which the cities are nodes and the roads are edges.

You are given the int treeHeight giving the height of the tree. (The height of a perfect binary tree is the number of edges on the path between the root
node and any leaf node.) Thus, there are 2^(treeHeight+1)-1 cities and 2^(treeHeight+1)-2 roads in total.

The picture below shows how the road network looks like when treeHeight = 2.

We want to send some cars into the road network. Each car will be traveling from its starting city to its destination city without visiting the same city
twice. (Note that the route of each car is uniquely determined by its starting and its destination city.) It is possible for the starting city to be equal to
the destination city, in that case the car only visits that single city.

Our goal is to send out the cars in such a way that each city will be visited by exactly one car. Compute and return the smallest number of cars we
need in order to do so.

Definition

Class: TrafficCongestionDivTwo

Method: theMinCars

Parameters: int

Returns: long

Method signature: long theMinCars(int treeHeight)

(be sure your method is public)

Notes

- The answer will always fit into a 64-bit signed integer data type.

Constraints

- treeHeight will be between 0 and 60, inclusive.

Examples

1) 1

Returns: 1

In this case, one car can visit all the cities.

2) 2

Returns: 3

Here is one way to visit all cities exactly once by three cars:

3) 3

Returns: 5

****

PROBLEM 7.

There are some deer in the zoo. Each deer has two antlers. You are given int[]s antler1 and antler2. These two int[]s will contain the same number of
elements. For each index i, antler1[i] and antler2[i] are the weights of the two antlers of one of the deer.

You are also given an int capacity. A deer is unbalanced if the weight difference between his antlers is strictly more than capacity.

You decided to perform some operations on the deer. Your goal is to make all deer balanced. In each operation, you can choose some two antlers
(each on a different deer) and swap them.

Return the minimal number of operations required to make all deer balanced. If this is impossible, return -1 instead.

Definition

Class: AntlerSwapping

Method: getmin

Parameters: int[], int[], int

Returns: int

Method signature: int getmin(int[] antler1, int[] antler2, int capacity)

(be sure your method is public)

Constraints

- antler1 and antler2 will contain the same number of elements.

- antler1 and antler2 will contain between 1 and 16 elements, inclusive.

- Each element of antler1 and antler2 will be between 1 and 1,000,000, inclusive.

- capacity will be between 0 and 1,000,000, inclusive.

Examples

1)

{3, 2, 2}

{3, 5, 5}

0

Returns: 1

There are three deer in the zoo. As capacity=0, a deer is only balanced if he has two antlers of exactly equal weight. Currently, deer 0 is balanced and
the other two are not. We can fix that in a single operation. For example, we can swap deer 1's antler 1 and deer 2's antler 2. After this operation, deer
1 will have two antlers that weigh 5 each, and deer 2 will have two antlers that weigh 2 each.

2)

{4, 2, 6, 4, 8, 5, 2, 3}

{3, 4, 5, 2, 8, 5, 7, 6}

1

Returns: 2

One of the optimal ways is as follows:

Swap deer 1's antler with weight 2 and deer 3's antler with weight 4.

Swap deer 6's antler with weight 7 and deer 7's antler with weight 3.

3)

{12, 34, 56, 78}

{1234, 2345, 3456, 4567}

100

Returns: -1

If it is impossible to achieve the goal, return -1.

4)

{47, 58, 2013}

{49, 55, 2013}

3

Returns: 0

****

PROBLEM 8.

Once upon a time, there was a civilization called Ruritania. It had n building sites numbered from 0 to n-1. There were various types of buildings
such as libraries, markets, and palaces. Each building type was assigned an integer from 1 to 50. The building at site i (0-based index) was of type
kind[i].

With the passing of millennia, Ruritania declined and its building sites were covered in sand, concealing all the buildings. Due to wind and terrain,
the depth of the sand varied. The building at site i (0-based index) was buried depth[i] meters below the surface.

Recently, an intrepid archeologist excavated K building sites using a machine that could dig to a maximum depth of D meters. Thus, he only
discovered buildings that had been buried at most D meters below the surface.

You are given int[]s kind, depth, and found as well as the int K. The types of buildings discovered by the excavation are given by the int[] found,
which contains at most one value for each building type even if several buildings of a type were excavated.

Return the number of K-tuples of sites that could have been excavated to arrive at the given values. If the given information is not consistent with any
configuration of building sites, return 0.

Definition

Class: Excavations

Method: count

Parameters: int[], int[], int[], int

Returns: long

Method signature: long count(int[] kind, int[] depth, int[] found, int K)

(be sure your method is public)

Constraints

- kind will contain N elements, where N is between 1 and 50, inclusive.

- Each element of kind will be between 1 and 50, inclusive.

- depth will contain N elements.

- Each element of depth will be between 1 and 100,000, inclusive.

- found will contain between 1 and 50 elements, inclusive.

- Each element of found will occur in kind at least once.

- The elements of found will be distinct.

- K will be between the number of elements in found and N, inclusive.

Examples

1)

{1, 1, 2, 2}

{10, 15, 10, 20}

{1}

2

Returns: 3

There are four building sites. Two have buildings of type 1 and two have buildings of type 2. The type 1 buildings are at depths 10 and 15. The type 2
buildings are at depths 10 and 20. The archeologist has excavated two sites and discovered only type 1 buildings. He must have excavated one of
three possible pairs of sites:

Sites 0 and 1. The archeologist's machine excavates to a maximum depth D of at least 10.

Sites 0 and 3. The machine excavates to a maximum depth D that falls in the interval [10, 20).

Sites 1 and 3. The machine excavates to a maximum depth that falls in the interval [15, 20).

The other pairs of sites could not have been excavated. For example, the archeologist could not have excavated sites 0 and 2, because he would have
found either none or both of the buildings.

2)

{1, 1, 2, 2}

{10, 15, 10, 20}

{1, 2}

2

Returns: 4

The archeologist could have chosen any pair of sites containing a type 1 and a type 2 building. With a large enough value of D, he could have
excavated both.

3)

{1, 2, 3, 4}

{10, 10, 10, 10}

{1, 2}

3

Returns: 0

The archeologist cannot have excavated three sites, or else he would have found three types of buildings.

****

PROBLEM 9.

Magical Girls are girls who have magical powers. They fight against evil to protect the Earth. Cosmic enemies have just attacked the Earth, and the
Magical Girls are going to fight them.

You are given a int[] magicalGirlStrength that describes the Magical Girls: for each i, magicalGirlStrength[i] is the strength of one of the girls. You
are also given a int[] enemyStrength and a long[] enemyCount that describe their enemies: for each i, there are enemyCount[i] enemies that each have
strength enemyStrength[i].

Each Magical Girl will always fight one enemy at a time. A Magical Girl will defeat her enemy if her strength is greater than or equal to the strength
of that enemy.

At the beginning of the fight the fatigue of each Magical Girl is 0. Each time a Magical Girl defeats an enemy, her fatigue increases by 1.

The Magical Girls want to defeat all the enemies. That is, each of the enemies must be defeated by one of the Magical Girls. Additionally, the
Magical Girls want to minimize the maximum fatigue among them.

If it is impossible to defeat all of the enemies, return -1. Otherwise, return the smallest F with the following property: the Magical Girls can defeat all
enemies in such a way that at the end the fatigue of each girl is at most F.

Definition

Class: SpaceWarDiv1

Method: minimalFatigue

Parameters: int[], int[], long[]

Returns: long

Method signature: long minimalFatigue(int[] magicalGirlStrength, int[] enemyStrength, long[] enemyCount)

(be sure your method is public)

Notes

- The elements of enemyStrength are not necessarily pairwise distinct.

Constraints

- magicalGirlStrength will contain between 1 and 50 elements, inclusive.

- Each element of magicalGirlStrength will be between 1 and 10,000, inclusive.

- enemyStrength and enemyCount will each contain between 1 and 50 elements, inclusive.

- enemyStrength and enemyCount will contain the same number of elements.

- Each element of enemyStrength will be between 1 and 10,000, inclusive.

- Each element of enemyCount will be between 1 and 100,000,000,000,000 (10^14), inclusive.

Examples

1)

{2, 3, 5}

{1, 3, 4}

{2, 9, 4}

Returns: 7

There are 3 Magical Girls, their strength are 2, 3, and 5. There are 3 kinds of enemies: 2 enemies with strength 1 each, 9 enemies with strength 3
each, and 4 enemies with strength 4 each.

This is one of the ways how to minimize the maximal fatigue:

Magical girl 0 defeats 2 enemies with strength 1.

Magical girl 1 defeats 7 enemies with strength 3.

Magical girl 2 defeats 2 enemies with strength 3 and 4 enemies with strength 4.

2)

{2, 3, 5}

{1, 1, 2}

{2, 9, 4}

Returns: 5

Each of the Magical Girls can defeat any of the enemies. The optimal strategy is that each girl should defeat 5 of the enemies.

****

PROBLEM 10.

The Primal Grez are ferocious creatures. They constantly fight each other. When a Grez wins a battle against another Grez, the winner captures the
loser's essence, thus becoming stronger. More formally:

The power level of each Grez is a positive integer.

A Grez can only defeat creatures that have a strictly smaller power level.

When a Grez of power level X defeats a Grez of power level Y, the first Grez's power level is increased to X + Y/2. Note that Y/2 represents integer
division, i.e., the fractional part is discarded. For example, for Y=3 we have Y/2 = 1.

Your goal is to help a Grez that currently has power level equal to int initialLevel battle against a set of other Grez. For each i, Grez number i (0-
based index) has a power level equal to grezPower[i]. Your Grez can challenge the other creatures in any order.

You are given the int initialLevel and the int[] grezPower. Return the maximum number of creatures your Grez can defeat, one after another.

Definition

Class: PrimalUnlicensedCreatures

Method: maxWins

Parameters: int, int[]

Returns: int

Method signature: int maxWins(int initialLevel, int[] grezPower)

(be sure your method is public)

Constraints

- initialLevel will be between 1 and 1000, inclusive.

- grezPower will contain between 1 and 50 elements, inclusive.

- Each element of grezPower will be between 1 and 1000, inclusive.

Examples

1)

31

{10, 20, 30}

Returns: 3

It is possible to defeat all the available opponents. For example:

Defeat the creature with power level 30. Your creature's power level becomes 31 + 15 = 46.

Defeat the creature with power level 10. Your creature's power level becomes 46 + 5 = 51.

Defeat the creature with power level 20. Your creature's power level becomes 51 + 10 = 61.

2)

20

{24, 5, 6, 38}

Returns: 3

It is best to defeat creatures 1 and 2 before facing creature 0. Your creature's power level will be 25 when facing creature 0. It is not possible to defeat
creature 3.

3)

20

{3, 3, 3, 3, 3, 1, 25 }

Returns: 6

It is possible to defeat the 6 weakest creatures. After that your creature's power level will be 25, which is not strong enough to defeat another level 25
creature.

4)

4

{3, 13, 6, 4, 9}

Returns: 5

5)

7

{7, 8, 9, 10}

Returns: 0

All the available opponents are too strong for your creature to defeat.

PROBLEM 11.

The factorial of the k-th order of a number n is denoted n!k and defined by the following recurrences:

1) n!k = n!(k-1) * (n-1)!k for n > 0 and k > 0

2) n!k = 1 for n = 0

3) n!k = n for k = 0

For example, 7!1 = 7! (the traditional factorial), and 5!3 = 5!2 * 4!3 = (5!1 * 4!2) * 4!3.

You are given ints N and K. Compute the number of distinct divisors of the number N!K. Return the computed number modulo 1,000,000,009.

Definition

Class: MegaFactorialDiv2

Method: countDivisors

Parameters: int, int

Returns: int

Method signature: int countDivisors(int N, int K)

(be sure your method is public)

Constraints

- N will be between 1 and 1000, inclusive.

- K will be between 1 and 100, inclusive.

Examples

1)

3

1

Returns: 4

3!1 = 3! = 6. The divisors of 6 are 1, 2, 3 and 6.

2)

3

2

Returns: 6

3!2 = 3!1 * 2!2 = 3! * 2!1 * 1!2 = 3! * 2! * 1!1 * 0!2 = 3! * 2! * 1! * 1 = 12. The divisors of 12 are 1, 2, 3, 4, 6 and 12.

3)

4

2

Returns: 18

4!2 is equal to 288.

4)

6

3

Returns: 1392

5)

100

2

Returns: 321266186

****

PROBLEM 12.

You are interested in a sequence of integers S = (S[1], S[2], ..., S[K]) with the following properties:

K >= 1.

For all i, S[i] is a non-negative integer.

S[1] + S[2] + ... + S[K] = N.

S[1] mod M, S[2] mod M, ..., S[K] mod M are all different.

You are given a long N and an int M. Return the number of different valid sequences S, modulo 1,000,000,007. Two sequences S1 and S2 are
considered different if they either have different number of elements, or if there is an index i such that S1[i] is different from S2[i].

Definition

Class: DistinctRemainders

Method: howMany

Parameters: long, int

Returns: int

Method signature: int howMany(long N, int M)

(be sure your method is public)

Constraints

- N will be between 1 and 10^18, inclusive.

- M will be between 1 and 50, inclusive.

Examples

1)

3

2

Returns: 5

The 5 sequences are:

(3)

(0, 3)

(1, 2)

(2, 1)

(3, 0)

2)

3

3

Returns: 9

The 9 sequences are:

(3)

(1, 2)

(2, 1)

(0, 1, 2)

(0, 2, 1)

(1, 0, 2)

(1, 2, 0)

(2, 0, 1)

(2, 1, 0)

3)

58

1

Returns: 1

The only sequence is (58).

****

PROBLEM 13.

Little Elephant from the Zoo of Lviv likes permutations. A permutation of size N is a sequence (a1, ..., aN) that contains each of the numbers from 1
to N exactly once. For example, (3,1,4,5,2) is a permutation of size 5.

Given two permutations A = (a1, ..., aN) and B = (b1, ..., bN), the value magic(A,B) is defined as follows: magic(A,B) = max(a1,b1) + max(a2,b2) +
... + max(aN,bN).

You are given the int N. You are also given another int K. Let X be the number of pairs (A,B) such that both A and B are permutations of size N, and
magic(A,B) is greater than or equal to K. (Note that A and B are not required to be distinct.) Return the value (X modulo 1,000,000,007).

Definition

Class: LittleElephantAndPermutationDiv1

Method: getNumber

Parameters: int, int

Returns: int

Method signature: int getNumber(int N, int K)

(be sure your method is public)

Constraints

- N will be between 1 and 50, inclusive.

- K will be between 1 and 2500, inclusive.

Examples

1)

1

1

Returns: 1

For N=1 the only pair of permutations is ((1), (1)). The magic of this pair of permutations is 1, so we count it.

2)

2

1

Returns: 4

Now there are four possible pairs of permutations. They are shown below, along with their magic value.

magic((1,2), (1,2)) = 1+2 = 3

magic((1,2), (2,1)) = 2+2 = 4

magic((2,1), (1,2)) = 2+2 = 4

magic((2,1), (2,1)) = 2+1 = 3

In all four cases the magic value is greater than or equal to K.

3)

3

8

Returns: 18

4)

10

74

Returns: 484682624

4)

50

1000

Returns: 539792695