EASY

Find the Winner!

Recommended Time: 15 min Points: 5014 test cases (3 samples)

Coding**EASY**

General ProgrammingBasic ProgrammingLoopsConditionalsBasic Array OperationsStringsBasic String OperationsArraysCore SkillsProblem Solving

Languages: C, Clojure, C++, C++14, C# + 21 more

Andrea and Maria each have a deck of numbered cards in a pile face down. They play a game where they each alternately discard and flip the cards on the pile from top to bottom.

At the beginning of the game, someone will call out "Even" or "Odd". The first move depends on which is called. If "Even" is called, the player's top cards are flipped so they can see the face value. If instead "Odd" is called, the top card is removed from each deck and discarded, then each flips her next card. Andrea subtracts the value of Maria's card from her own and adds the result to her score. Likewise, Maria subtracts the value of Andrea's card from her own and adds the result to her score.

From this point forward, each alternately discards then flips a card. Each time two cards are flipped, the players' scores are computed as before. Once all the cards have been played, whoever has the most points wins.

As an example, Maria's cards, face down, are [3, 5, 6] and Andrea's are [4, 5, 7]. After calling "Even" at random, the game begins. The following table represents game play with cumulative score at the bottom. Maria's score is -2, Andrea's is +2 so Andrea wins.

Maria's Card	Andrea's Card	Maria's Score	Andrea's Score
3	4	3 - 4 = -1	4 - 3 = 1
5	5	Discard	Discard
6	7	6 - 7 = -1	7 - 6 = 1
Cumulative scores		-2	2

You must determine the name of the winner if there is one, otherwise they tie. Return the string Andrea, Maria or Tie.

Function Description

Complete the function *winner* in the editor below. The function must return a string denoting the outcome of the game. Return `Andrea` if Andrea won, or return `Maria` if Maria won. If their scores are tied, return `Tie`instead.

winner has the following parameter(s):

andrea: An array of n integers that denotes Andrea's array of card values.
maria: An array of n integers that denotes Maria's array of card values.
s: A string that represents the starting called out word

Constraints

- $2 \leq n \leq 10^5$
- $1 \leq a[i], m[i] \leq 10^3$, where $0 \leq i < n$
- String s will either the word `Odd` or the word `Even`.

Input Format For Custom Testing

Input from stdin will be processed as follows and passed to the function:

The first line contains an integer n, denoting the number of elements in *andrea*.
The next n lines each contain an integer describing $a_i$where $0 \leq i < n$.
The next line contains an integer, n, denoting the number of elements in *maria*.
The next n lines each contain an integer describing $m_i$where $0 \leq i < n$.
The next line contains string *s, Even* or *Odd*.

Sample Case 0

Sample Input 0

```
3
1
2
3
3
2
1
3
Even
```

Sample Output 0

```
Maria
```

Explanation 0

In this game, $andrea = [1, 2, 3]$ and $maria = [2, 1, 3]$. Because $s =$ `Even`, the only cards flipped are at indexes 0 and 2.

- When $i = 0$, Andrea gets $a[0] - m[0] = 1 - 2 = -1$ point and Maria gets $m[0] - a[0] = 2 - 1 = 1$ point.
- When $i = 2$, Andrea gets $a[2] - m[2] = 3 - 3 = 0$ points and Maria gets $m[2] - a[2] = 3 - 3 = 0$ points.

At the end of play, Andrea's cumulative score is -1 and Maria's is 1 so Maria wins.

Sample Case 1

Sample Input 1

```
3
1
2
3
3
2
1
3
Odd
```

Sample Output 1

```
Andrea
```

Explanation 1

In this game, $andrea = [1, 2, 3]$ and $maria = [2, 1, 3]$. Because $s =$ `Odd`, the only indices we can choose during moves are in the set $\{1\}$. The game proceeds like so:

- When $i = 1$, Andrea gets $a[1] - m[1] = 2 - 1 = 1$ point and Maria gets $m[1] - a[1] = 1 - 2 = -1$ point.

Andrea ends with 1 point, and Maria with -1. Andrea wins.

MEDIUM

Smart Sale

Recommended Time: 21 min

Points: 758 test cases (3 samples)

CodingMEDIUM

AlgorithmsData StructuresSortingProblem Solving +

Languages: C, Clojure, C++, C++14, C# + 20 more

A sales executive needs to simplify an assigned sales task. The task is to sell the items given to her in a bag, where each item has an ID number.
It is easier to sell items with the same ID numbers. The sales executive also decides to remove some items from the bag. The largest number of items that she can remove is also known to her. Find the minimum number of different IDs the final bag can contain after removing the allowed number of items.

For example, she starts with a bag that contains the following $n = 6$ items: $ids = [1,1,1,2,2,3]$ and she can remove $m = 2$ items. If she removes 2 of item 1, she will still have items of all three types. She could remove 2 of item 2, and have $ids = [1,1,1,3]$ with 2 discrete item types or remove 1 each of types 2 and 3 leaving $ids = [1,1,1,2]$. Either way, there are 2 item types left.

Function Description

Complete the function *deleteProducts* in the editor below. The function must return an integer that denotes the minimum number of different IDs the final bag can contain.

deleteProducts has the following parameters:

ids: an array of n integers that denotes the ID numbers of all items

m: integer that denotes the maximum number of items that can be deleted from the bag.

Constraints

- $1 \leq n \leq 100000$
- $1 \leq ids[i] \leq 1000000$
- $1 \leq m \leq 100000$

Input Format For Custom Testing

Locked stub code in the editor reads the following input from stdin and passes it to the function:

The first line contains an integer, n, denoting the size of the array ids.
Each line i of the n subsequent lines (where $0 \leq i < n$) contains an integer describing $ids[i]$.
The last line contains an integer, m, denoting maximum number of items that can be deleted.

Sample Case 0

Sample Input 0

```
4
1
1
1
1
2
```

Sample Output 0

```
1
```

Explanation 0

Initially, the bag contains 4 items of the same kind, so whatever item the executive deletes, the minimum different ID numbers will be 1.

Sample Case 1

Sample Input 1

```
6
1
2
3
1
2
2
3
```

Sample Output 1

```
1
```

Explanation 1

Initially, $ids = [1,2,3,1,2,2]$. 3 items can be deleted and it is optimal to choose items having IDs 1 and 3. The final bag contains 3 items, all with ID 2.

HARD

Build Offices

Recommended Time: 28 min

Points: 10015 test cases (3 samples)

CodingHARD

RecursionGraphsProblem SolvingCore SkillsAlgorithmsBacktracking

Languages: C, Clojure, C++, C++14, C# + 20 more

A company wants to develop an office park in an area that has been divided up into a grid where each cell represents a potential building lot. The goal is for the furthest of all lots to be as near as possible to an office building. Determine the building placement to minimize the distance the most distant lot is from an office building. Movement is restricted to horizontal and vertical, i.e. diagonal movement is not allowed.

For example, there are $n = 3$ office buildings to build on a grid that is $w = 4$ lots wide and $h = 4$ lots high. An optimal grid placement sets any lot within two units distance of an office building. In the distance grid below, offices are cells at distance 0.

```
1 0 1 2
2 1 2 1
1 0 1 0
2 1 2 1
```

That represents one optimal solution among several, this array rotated for example.

Function Description

Complete the function *findMinDistance* in the editor below. The function must return an integer that denotes the maximal value among shortest distances to the closest office for each cell.

findMinDistance has the following parameter(s):

- *w:* an integer, the width of the grid
- *h:* an integer, the height of the grid
- *n:* an integer, the number of buildings to place

Constraints

- $1 \leq w, h$
- $w \times h \leq 28$
- $1 \leq n \leq 5$
- $n \leq w \times h$

Input Format For Custom Testing

The first line contains an integer, w.

The second line contains an integer, h.

The final line contains an integer, n.

Sample Case 0

Sample Input 0

```
2
3
2
```

Sample Output 0

```
1
```

Explanation 0

```
w = 2
h = 3
n = 2
```

One of the optimal plans:

```
01
11
10
```

Here 0 denotes an office building; all values in the optimal plan are distances to the nearest office building. The maximum distance is 1.

Sample Case 1

Sample Input 1

```
5
1
1
```

Sample Output 1

```
2
```

Explanation 1

```
w = 5
h = 1
n = 1
```

The optimal plan:

```
21012
```

Here 0 denotes an office building; all values in the optimal plan are distances to the nearest office building. The maximum distance is 2.