NCTU Competitive Programming I
2020 Spring
Midterm Exam 1 Extension

ID	Problem Name	Time Limit
A	Aries	10 sec
B	Be Easy	1 sec
C	Cursed Power	5 sec
D	Disjoint Sets	5 sec

NCTU Competitive Programming I
2020 Spring
Midterm Exam 1

Rules

Any violation of the rules is considered cheating and will result in a disqualification.

1. You should take place in the exam using your personal and only DOMjudge account given prior to the exam.
2. You must not discuss problems or share ideas/solutions with any other people.
3. Should you have a question about the problems or face any exam-related issue, please use the "request clarification" feature of DOMjudge. Do not ask or discuss with any other people.
4. Announcements made during the exam can be viewed using the "Clarifications" interface of DOMjudge.
5. You may use existing source code as parts of your solution. However, the code you use must either be in the lecture notes or be written by you prior to the exam.
6. You may search for and read documents or references online, but copying solution codes are not allowed.
7. Any malicious action interfering the exam is prohibited.

Scoring

1. You must submit your solutions via DOMjudge. The judge system will only respond to submissions that are submitted within the exam duration (300 minutes). The response to each run must be one of the following:
   - `Correct`: The judge accepts your code.
   - `Compiler-Error`: Your code cannot be successfully compiled.
   - `TimeLimit`: Your program consumes too much time.
   - `Run-Error`: Your program terminates with an non-zero return code, which often means your program is terminated by the operating system.
   - `Wrong-Answer`: The judge rejects the output of your program.
   - `No-Output`: Your program does not generate any output.

2. The score you get for midterm exams is based on the total number of problems to which a correct solution is submitted:

Solved	Score
0	20
1	30
2	35
3	39
4	43
5	46
6	48
7	49
8	50
9	51

Hint

The problems are sorted by their names (lexicographically) and not by increasing difficulty; problems that appear first are not necessarily easier. Because of this, it is recommended that you read every problem.

Almost blank page

Problem A
Aries

Time limit: 10 seconds

Memory limit: 512 megabytes

Problem Description

The "Aries" is a puzzle that consists of a frame of numbered square tiles in random order. The objective of the puzzle is to place the tiles in order by making clockwise rotation of any 2×2 sub-block. With following Aries, one can clockwise rotate the sub-block at the center-top of the left Aries which contains tiles 3, 7, 6, and 2, making the tiles re-ordered like the right Aries in the following.

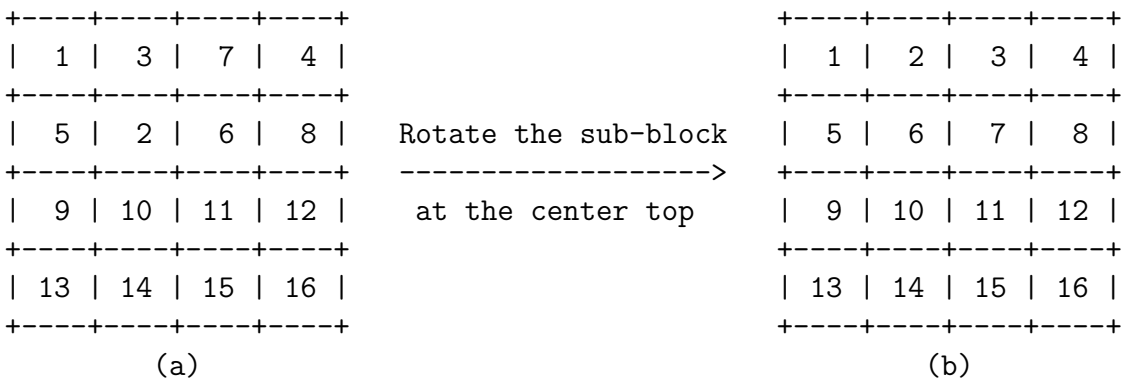

Figure 1: The left Aries can be solved by 1 clockwise rotation.

Given a messy Aries puzzle, please calculate the minimum count of clockwise rotation required to solve the Aries (make the given Aries into Figure 1(b)). If the Aries is too hard, that is, if it takes over 10 rotations to reorder the tiles, report this problem.

Input Format

The first line contains an integer T indicating the number of testcases.

A test case consists of 4 lines, each of which contains 4 space-separated numbers. The j-th number at the i-th line is the number written on the tile at the i-th row and j-th column of the frame. It is guaranteed that the Aries can be solved even if it is difficult.

Output Format

For each test case, output the minimum count of clockwise rotations for solving the Aries. If the Aries is too hard, output "`HARD ARIES`" (without quotes) in one line.

Technical Specification

- $0 < T \leq 250$
- In each test case, each of the numbers from 1 to 16 appears exactly once.

Sample Input 1

```
4
1 3 7 4
5 2 6 8
9 10 11 12
13 14 15 16
1 3 7 4
5 2 6 8
10 14 11 12
9 13 15 16
1 3 4 8
5 2 14 15
11 6 10 12
9 7 13 16
1 4 12 16
8 9 2 3
6 5 11 10
13 14 7 15
```

Sample Output 1

```
1
2
8
HARD ARIES
```

Problem B
Be Easy

Time limit: 1 second
Memory limit: 512 megabytes

Problem Description

Be easy! This problem is slightly harder than the problem B in a CodeForces Division 3 contest. This is a problem related to binary numbers. If you are not familiar with binary numbers, you may use Google or read Wikipedia.

Min-Zheng asks you to generate a binary number with two parameters n and k. The length of the number is n bits, and the number of 1's among the n bits is 3. Min-Zheng wants you to hand in the k-th smallest of all binary numbers satisfying the constraints mentioned above. For example, 000111_2 and 001011_2 are the smallest and the second smallest 6-bit binary numbers satisfying the constraints, respectively. If $n = 6$ and $k = 1$, then Min-Zheng wants you to hand in 000111_2. If $n = 6$ and $k = 2$, then 001011_2 is what Min-Zheng wants.

As a problem in the extended midterm, Min-Zheng sets a wide range for the parameters. That makes the binary numbers are too hard to be verified by a human. Since there are exact three 1's in the binary number, your program should output the indices of the 1's in ascending order. That is, output `4 5 6` for 000111_2 and output `3 5 6` for 001011_2.

Input Format

The first line contains a positive integer T indicating the number of test cases. Then T lines follow, and each of them represents a test case. A test case consists of two positive integers n and k as described above.

Output Format

For each test case, output the indices of 1's of the binary number that Min-Zheng wants you to hand in. The indices must be output in one line in ascending order. Moreover, you must output exact one blank between two numbers.

Technical Specification

- T is at most 100.
- n is at most 10000.
- k is at most $\binom{n}{3} = \frac{n(n-1)(n-2)}{6}$. You may need to use a 64-bit integer to save k.
- T, n and k are positive.

Sample Input 1

```
5
6 1
6 2
6 3
6 4
6 20
```

Sample Output 1

```
4 5 6
3 5 6
3 4 6
3 4 5
1 2 3
```

Problem C
Cursed Power

Time limit: 5 seconds

Memory limit: 512 megabytes

Problem Description

Fast exponentiation algorithms are efficient methods to compute b^k (aka the k-th of power of a base b). The time of such methods are in $o(k)$ and are typically in $O(\log k)$. There are many variants of the power operations, and we can efficiently compute them by similar approches like exponentiation by squaring. One famous variant, the modular power operation *modPow*, has implementation in Java (`BigInteger.modPow`) and Python (`pow` supports modular). It takes one more parameter m, the modulo number, and it is defined as follows.

$$modPow(b, k, m) = b^k \mod m$$

In the last exam, there was a problem related to power operations. Unfortunately, no students solved that problem. Min-Zheng was unhappy, because he taught exponentiation by squaring before the exam. So, he showed how to solve the problem in the lecture last week. Min-Zheng wants to verify if the students learned the idea of exponentiation by squaring and its application. He decides to make a problem which is solvable by similar approaches.

He now defines the "cursed power" operation *cow*, a variant of modular power, as follows.

$$cow(b, k, m) = k \cdot b^k \mod m$$

The problem is, given three positive integers b, n, m, compute the value of

$$\left(\sum_{i=1}^{n} cow(b, i, m)\right) \mod m$$

For $b = 2$, $n = 3$, $m = 5$, the value is $2^1 + 2 \cdot 2^2 + 3 \cdot 2^3 \mod 5 = 2 + 8 + 24 \mod 5 = 4$.

Input Format

The first line conteains an integer T indicating the number of test cases. T lines follow, and each of them represents a test case. Each test case consists of three positive integers b, n, m separated by blanks.

Output Format

Output the value of $(\sum_{i=1}^{n} cow(b, i, m)) \mod m$ in a line.

Technical Specification

- $1 \leq T \leq 10000$
- $b, n, m \in [1, 2^{30} - 1]$
- m can be a comsopite number.

Sample Input 1

```
4
2 3 4
2 3 5
1 1 1
1023456789 1067895432 1000000007
```

Sample Output 1

```
2
4
0
69338362
```

Problem D
Disjoint Sets

Time limit: 5 seconds

Memory limit: 512 megabytes

Problem Description

In competitive programming, there are many problems related to operations on sequence. It is not surprising we have this kind of problems in the midterm. However, this is the last one.

You are given a sequence $(a_1, a_2, \ldots, a_n)$ of n integers. You must implement three kinds of operations as the follows.

- $op1(v, p)$: reassign a_v to an integer p. You may assume $v \in [1, n]$ and $p \in [1, 10^9]$.
- $op2(v, p)$: for all elements in the sequence which are equal to v, reassign them to p. You may assume that $v, p \in [1, 10^9]$.
- $op3(v, p)$: print the minimum non-negative integer x such that either a_{v+x} or a_{v-x} is p. If there does not exist any element equal to p in the sequence, print `-1`. You may assume that $v \in [1, n]$ and $p \in [1, 10^9]$.

For example, you are given a sequence (1,2,1) and three operations $op2(1, 2)$, $op1(1, 1)$, and $op3(3, 1)$. If your program executes the operations in the given order, it should output 2.

1. After $op2(1, 2)$ executed, the sequence becomes $(2, 2, 2)$.
2. After $op1(1, 1)$ executed, the sequence becomes $(1, 2, 2)$.
3. The minimum value of non-negative x such that either a_{3+x} or a_{3-x} equals 1 is 2: $a_{3-2} = a_1 = 1$.

You probably wonder why the name of this problem is "Disjoint Sets". Actually, this problem can be solved by maintaining many sets, and they are disjoint. I.e., the instersection of any two maintained sets is empty. However, the classical "disjoint sets" data structure does not support the operations that we mentioned above. Please utilize the set data structures (based on balanced binary search tree like `TreeSet` in JAVA) and an efficient map data structure to implement your program to get good performance.

Input Format

The first line contains two positive integers n and q. n is the number of elements in the sequence, and q is the number of operations to be applied.

The second line contains n numbers $a_1, a_2, \ldots, a_n$ where a_i and a_{i+1} are separated by a blank for $i \in [1, n)$.

Then, q lines follows. On each of them, there are three integers T, v and p indicating an

operation $opT(v, p)$. The operations will be executed in the given order.

Output Format

For each $op3$, please output the number described above in a line.

Technical Specification

- $1 \leq n \leq 10^5$
- $1 \leq q \leq 10^5$
- $1 \leq a_i \leq 10^9$ for $i \in [1, n]$
- Arguments of all operations satisfy the assumptions in the problem description.

Sample Input 1

```
10 10
1 2 3 4 5 6 7 8 9 10
3 2 5
1 2 5
3 2 5
2 5 2
3 2 5
2 2 5
3 2 5
1 10 2
2 2 5
3 9 5
```

Sample Output 1

```
3
0
-1
0
1
```

Hint

Don't start programming before you get an analysis showing your approach has time complexity $O((n + q) \log^2(n + q))$. The problem setter claimed: this is the hardest problem in the midterm.