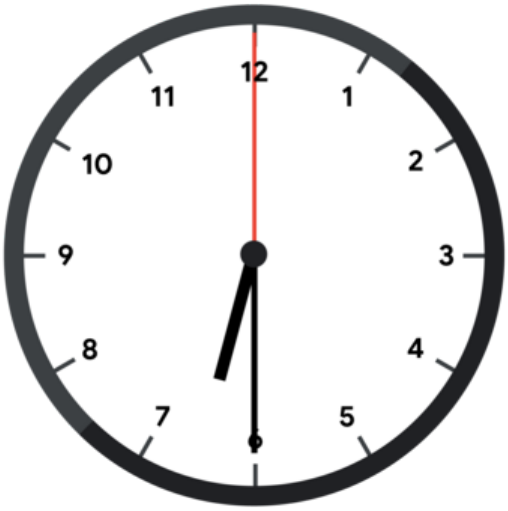
Code Jam 2021 - Round 1B

Broken Clock

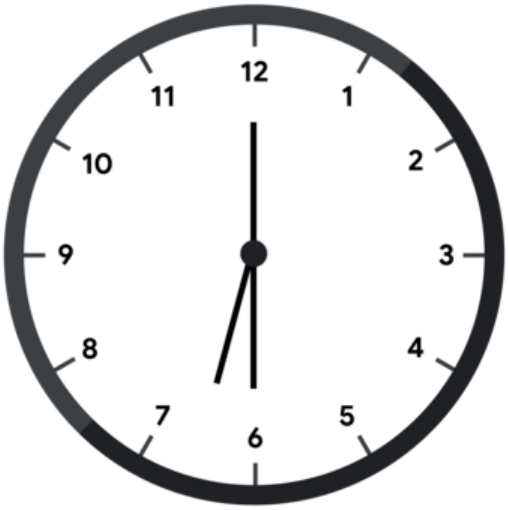
Problem

Emmett found an old clock in his attic. The clock is a circle with 3 hands that attach to the center and rotate clockwise at constant speeds. They are called the *hours hand*, the *minutes hand* and the *seconds hand*. At midnight, all hands point up. The hours hand completes a full revolution in 12 hours, the minutes hand completes a full revolution in 1 hour, and the seconds hand completes a full revolution in 1 minute. 1 hour is equal to 60 minutes, 1 minute is equal to 60 seconds, and 1 second is equal to 10^9 nanoseconds.

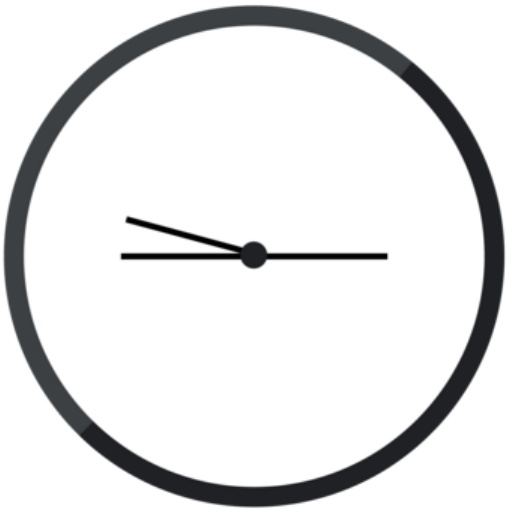
For example, the clock depicted below is showing that the time is exactly 6 hours and 30 minutes after midnight. The hours hand (short black) is halfway between 6 and 7 (completed $6.5/12$ of a revolution), the minutes hand (long black) is pointing straight down because it has completed exactly 6 and a half full revolutions and the seconds hand (red) is pointing straight up because it has completed an integer number of full revolutions.



Unfortunately, the hands are broken, so they all look identical and there is no way to know which hand is which. The clock in the picture above, with its hands broken, would look like this.



In addition, no markings remain that allow Emmett to know which way is up, so any rotation of the clock could be the correct one (the clock can only be rotated, not reflected). To continue with our example, the fully broken clock could look like this.



Emmett does know that the time was strictly before noon, that is, strictly less than 12 hours after midnight. Emmett has taken a picture of the clock. Given that picture (represented by the angles of the hands relative to a single arbitrary axis), figure out what time it could correspond to.

Notice that Emmett has already figured out a viable orientation of the clock in some cases (Test Set 1) and has managed to narrow down the possible times to a whole integer number of seconds (Test Sets 1 and 2) or nanoseconds (Test Set 3). Please see the limits sections for more details.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ lines follow. Each line describes a test case and contains three integers $\mathbf{A}$, $\mathbf{B}$, and $\mathbf{C}$: the angles of each hand, relative to an arbitrary axis and given in ticks in the clockwise direction. 1 *tick* is equal to $1/12 \times 10^{-10}$ degrees. This means that the hours hand rotates exactly 1 tick each nanosecond, the minutes hand rotates exactly 12 ticks each nanosecond and the seconds hand rotates exactly 720 ticks each nanosecond.

Output

For each test case, output one line containing `Case #`x`:` h m s n, where x is the test case number (starting from 1) and h, m, s, and n are integers: h is the number of full hours since midnight (between 0 and 11, inclusive), m is the number of full minutes since the last full hour (between 0 and 59, inclusive), s is the number of full seconds since the last full minute (between 0 and 59, inclusive) and n is the number of full nanoseconds since the last full second (between 0 and $10^9 - 1$, inclusive).

Limits

Time limit: 30 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$0 \leq \mathbf{A} \leq \mathbf{B} \leq \mathbf{C} < 360 \times 12 \times 10^{10}$.

Test Set 1 (Visible Verdict)

There is a time t that corresponds to the input such that:

- t is an integer number of seconds after midnight.
- t can be read from the input clock without rotating it.

Test Set 2 (Visible Verdict)

There is a time that corresponds to the input and is an integer number of seconds after midnight.

Test Set 3 (Visible Verdict)

There is a time that corresponds to the input and is an integer number of nanoseconds after midnight.

Sample

Note: there are additional samples that are not run on submissions down below.

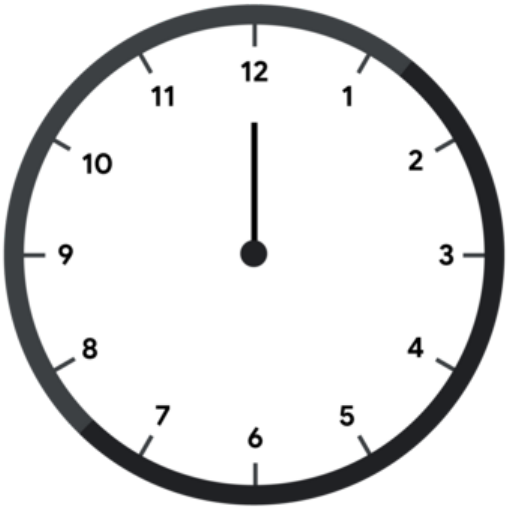
Sample Input

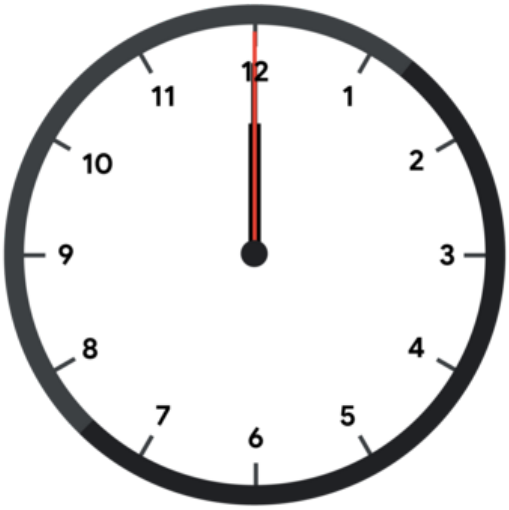
```
3
0 0 0
0 216000000000000 234000000000000
1476000000000 2160000000000
3723000000000
```

Sample Output

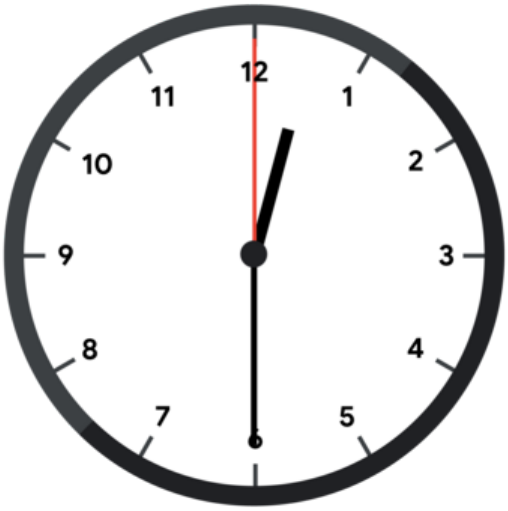
```
Case #1: 0 0 0 0
Case #2: 6 30 0 0
Case #3: 1 2 3 0
```

In Sample Case #1, all hands point up (as in the first picture below) which happens only exactly at midnight (as in the second picture below).



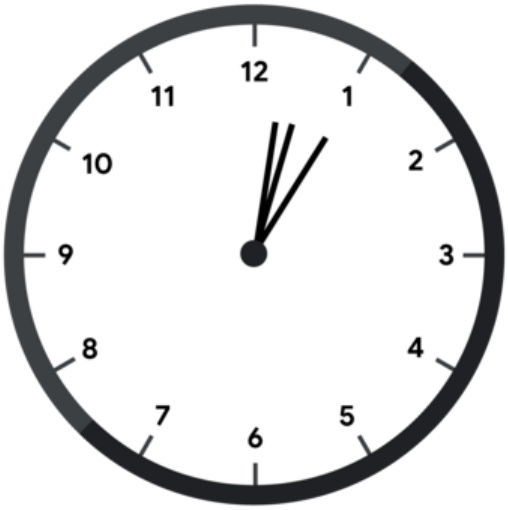


Sample Case #2 is the one pictured in the main part of the statement. The angles of the hands in degrees are 0, 180 and 195. These angles can correspond to 6h 30m 0s without rotating the clock, as the pictures in the main part of the statement show. Notice however, that at 0h 30m 0s (pictured below), the clock looks the same but rotated 180 degrees.



Even in Test Set 1, 0h 30m 0s would be a valid answer. The limit only says that there is one valid time that does not require rotating the clock, but times that work with rotation are also valid answers.

In Sample Case #3, the input represents the clock in the first picture below and the given output happens when interpreting the clock as in the second picture below.





Additional Sample - Test Set 2

The following additional sample fits the limits of Test Set 2. It will not be run against your submitted solutions.

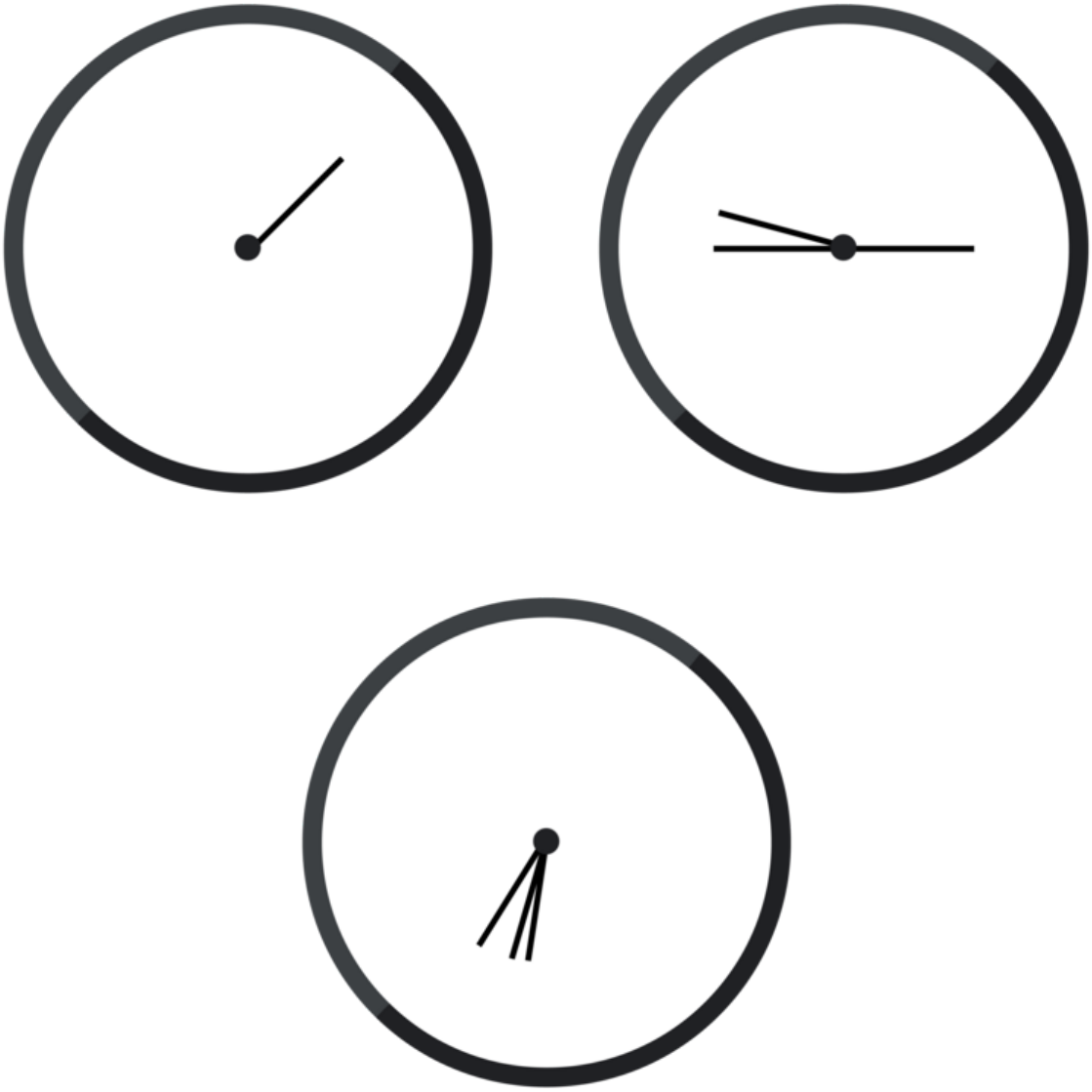
Sample Input

```
3
5400000000000 5400000000000
5400000000000
10800000000000 32400000000000
34200000000000
23076000000000 23760000000000
25323000000000
```

Sample Output

```
Case #1: 0 0 0 0
Case #2: 0 30 0 0
Case #3: 1 2 3 0
```

Sample Cases in this test set are the same as in the previous one, but the clock is rotated by 45, 90, and 180 degrees clockwise respectively, as shown below.



Additional Sample - Test Set 3

The following additional sample fits the limits of Test Set 3. It will not be run against your submitted solutions.

Sample Input	Sample Output
1 0 11 719	Case #1: 0 0 0 1

As explained above, 1 nanosecond after midnight the hands are moved by 1, 12, and 720 ticks, respectively. If the clock is also rotated counter-clockwise by 1 tick, the hand angles are exactly the ones given in the input.

Code Jam 2021 - Round 1B

Subtransmutation

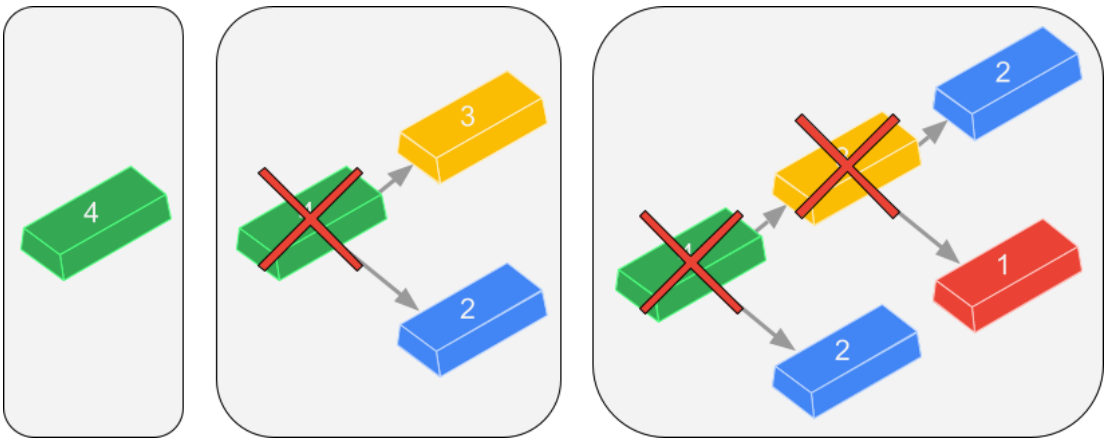
Problem

As the most skilled alchemist in your country, you were summoned yet again because powers beyond science were needed to satisfy your country's leader's ever increasing greed for rare metals.

Each metal is represented by a positive integer. You need to create $\mathbf{U_1}$ units of metal 1, $\mathbf{U_2}$ units of metal 2, ... and $\mathbf{U_N}$ units of metal $\mathbf{N}$. Metals $\mathbf{N}+1, \mathbf{N}+2, \ldots$ do exist, but you are not required to create any specific amount of them. You are allowed to create excess amounts of any metal, which can just be discarded.

Unfortunately, budget cuts have left you only the materials for a simple alchemy spell. For some fixed numbers $\mathbf{A}$ and $\mathbf{B}$, with $\mathbf{A} < \mathbf{B}$, you can take one unit of metal i and destroy it to create one unit of metal $(i - \mathbf{A})$ and one unit of metal $(i - \mathbf{B})$. If either of those integers is not positive, that specific unit is not created. In particular, if $i \leq \mathbf{A}$, the spell simply destroys the unit and creates nothing. If $\mathbf{A} < i \leq \mathbf{B}$ the spell destroys the unit and creates only a single unit of metal $(i - \mathbf{A})$.

You have been assigned an expert miner to assist you. The expert miner can fetch a single unit of any metal you want. From that unit, you can use your spell to create other metals and then use the spell on the resulting metals to create even more units. The picture below shows a single unit of metal 4 being converted into one unit of metal 1 and two units of metal 2 using two spells with $\mathbf{A} = 1$ and $\mathbf{B} = 2$.



Metals represented by larger integers are heavier and more difficult to handle, so you want to ask the expert miner for a single unit of metal represented by the smallest possible integer that is sufficient to complete your task, or say that there is no such metal.

Input

The first line of the input gives the number of test cases, $\mathbf{T}$. $\mathbf{T}$ test cases follow. Each test case consists of two lines. The first line of a test case contains three integers $\mathbf{N}$, $\mathbf{A}$, and $\mathbf{B}$, representing the largest metal number that you are required to create, and the two values that define the available spell as described above, respectively. The second line of a test case contains $\mathbf{N}$ integers $\mathbf{U_1}, \mathbf{U_2}, \ldots, \mathbf{U_N}$, representing the required units of metals $1, 2, \ldots, \mathbf{N}$, respectively.

Output

For each test case, output one line containing `Case #`x`: `y, where x is the test case number (starting from 1) and y is `IMPOSSIBLE` if it is not possible to create all required units starting from a single unit of metal. Otherwise, y is the smallest integer that represents a metal such that one unit of it is sufficient to create all the required units of metal.

Limits

Time limit: 30 seconds.
Memory limit: 1 GB.
$1 \leq \mathbf{T} \leq 100$.
$1 \leq \mathbf{N} \leq 20$.
$0 \leq \mathbf{U_i} \leq 20$, for all i.
$1 \leq \mathbf{U_N}$.
$2 \leq \mathbf{U_1} + \mathbf{U_2} + \cdots + \mathbf{U_N}$.

Test Set 1 (Visible Verdict)

$\mathbf{A} = 1$.
$\mathbf{B} = 2$.

Test Set 2 (Hidden Verdict)

$1 \leq \mathbf{A} < \mathbf{B} \leq 20$.

Sample

Note: there are additional samples that are not run on submissions down below.

Sample Input

```
3
2 1 2
1 2
5 1 2
2 0 0 0 1
3 1 2
1 1 1
```

Sample Output

```
Case #1: 4
Case #2: 6
Case #3: 5
```

In Sample Case #1, we require one unit of metal 1 and two units of metal 2. If we start with a single unit of metal 3, then applying the spell once will give us one unit of metal 1 and one unit of metal 2. There is no way to get an additional unit of metal 2. Similarly, starting with a single unit of metals 1 or 2 is not sufficient. However, a single unit of metal 4 is sufficient as is demonstrated in the picture in the main part of the statement.

In Sample Case #2, we can start with a single unit of metal 6 and apply the following operations:

- Apply spell on 6: $\{6\} \longrightarrow \{4, 5\}$.
- Apply spell on 4: $\{4, 5\} \longrightarrow \{2, 3, 5\}$.
- Apply spell on 2: $\{2, 3, 5\} \longrightarrow \{1, 3, 5\}$.
- Apply spell on 3: $\{1, 3, 5\} \longrightarrow \{1, 1, 2, 5\}$.

Note that even though we have an extra unit of metal 2, this solution is valid.

In Sample Case #3, we can start with a single unit of metal 5 and apply the following operations:

- Apply spell on 5: $\{5\} \longrightarrow \{3, 4\}$.
- Apply spell on 4: $\{3, 4\} \longrightarrow \{2, 3, 3\}$.
- Apply spell on 2: $\{2, 3, 3\} \longrightarrow \{1, 3, 3\}$.
- Apply spell on 3: $\{1, 3, 3\} \longrightarrow \{1, 1, 2, 3\}$.

There are other ways to apply spells which also work but they all require starting with a single unit of metal 5 or higher.

Additional Sample - Test Set 2

The following additional sample fits the limits of Test Set 2. It will not be run against your submitted solutions.

Sample Input

```
3
3 2 4
1 1 1
3 2 4
1 0 1
5 2 5
1 0 0 0 1
```

Sample Output

```
Case #1: IMPOSSIBLE
Case #2: 5
Case #3: 10
```

In the first Sample Case for Test Set 2, it is impossible to start with a single unit of any metal and apply the spell with $\mathbf{A} = 2$ and $\mathbf{B} = 4$ several times and be left with one unit of metals 1, 2, and 3.

Code Jam 2021 - Round 1B

Digit Blocks

Problem

You are going to build $\mathbf{N}$ towers of $\mathbf{B}$ cubic blocks each, one block at a time. Towers are built bottom-up: the i-th block to be placed in a tower ends up as the i-th from the bottom. You need to decide where to place each block before getting to see any of the upcoming blocks, and once placed, blocks cannot be moved.

Each block has a single decimal digit printed on it, and towers are built such that the faces with digits are all facing the front. The font is such that blocks cannot be rotated to obtain a different digit (for example, a block with a 6 on it cannot be rotated to obtain a block with a 9 on it, nor vice versa).

For example, suppose $\mathbf{N} = 3$ and $\mathbf{B} = 3$ and you currently have towers as shown in Picture 1. If a block with a 6 shows up next, you have two options: either place it on top of the tower with only two blocks (as shown in Picture 2) or start the third tower (as shown in Picture 3). Note that you cannot put it on top of the first tower since the first tower already has $\mathbf{B}$ blocks.



Picture 1



Picture 2



Picture 3

After the building is done, we read the $\mathbf{B}$ digit integer printed on the front of each tower from the top to the bottom (that is, the digit on the last block placed on a tower is the most significant digit). Notice that these integers may have any number of leading zeroes. Then, we add those $\mathbf{N}$ integers together to obtain the score of our building operation.

For example, in Picture 4 below, the integers read on each tower, from left to right, are 123, 345, and 96. The score of that building operation would be $123 + 345 + 96 = 564$.



Picture 4

The digit for each block is generated uniformly at random, and independently of any other information. In order for your solution to be judged correct, the sum of its scores over all $\mathbf{T}$ test cases must be at least $\mathbf{P}$.

Input and output

This is an interactive problem. You should make sure you have read the information in the Interactive Problems section of our FAQ.

Initially the judge will send you a single line containing four integers $\mathbf{T}$, $\mathbf{N}$, $\mathbf{B}$, and $\mathbf{P}$: the number of test cases, the number of towers, the number of blocks in each tower, and the minimum total score you need to reach to pass this test set.

Then, you must process $\mathbf{T}$ test cases. Each test case consists of $\mathbf{N} \times \mathbf{B}$ exchanges. Each exchange corresponds to placing one block. Within each exchange, the judge will first print a line containing a single integer $\mathbf{D}$ representing the digit printed on the block you need to place. You need to respond with a single line containing a single integer $\mathbf{i}$, the number (between 1 and $\mathbf{N}$) of the tower you want to place that block on.

After the last exchange of each test case except the last one, the judge will immediately start the next test case. After the last exchange of the last test case, the judge will print an additional line containing a single integer: `1` if your total score is at least $\mathbf{P}$ or `-1` if it is not.

If the judge receives an invalidly formatted line, an invalid tower number, or the number of a tower that already contains $\mathbf{B}$ blocks from your program, the judge will print a single number `-1`. After the judge prints `-1` for any of the reasons explained above, it will not print any further output. If your program continues to wait for the judge after receiving a `-1`, your program will time out, resulting in a Time Limit Exceeded error. Notice that it is your responsibility to have your program exit in time to receive a Wrong Answer judgment instead of a Time Limit Exceeded error. As usual, if the memory limit is exceeded, or your program gets a runtime error, you will receive the appropriate judgment.

You can assume that the digit for each block is generated uniformly at random, and independently for each digit, for each test case and for each submission. *Therefore even if you submit exactly the same code twice, the judge could use different random digits*.

Limits

Time limit: 60 seconds.
Memory limit: 1 GB.

$\mathbf{T} = 50$.
$\mathbf{N} = 20$.
$\mathbf{B} = 15$.
$\mathbf{D}$ is a decimal digit between 0 and 9.

Test Set 1 (Visible Verdict)

$\mathbf{P} = 860939810732536850$ (approximately 8.6×10^{17}).
Note that this boundary is chosen as approximately 90% of $\mathbf{T} \times S$, where $S = 19131995794056374.42...$ (approximately 1.9×10^{16}) is the highest possible *expected* score that a solution to this problem can achieve on one test case given unbounded running time.

The exact value of S as defined above can be found in lines 13 and 14 of the local testing tool.

Test Set 2 (Visible Verdict)

$\mathbf{P} = 937467793908762347$ (approximately 9.37×10^{17}).
Note that this boundary is chosen as approximately 98% of $\mathbf{T} \times S$.

Testing Tool

You can use this testing tool to test locally or on our platform. To test locally, you will need to run the tool in parallel with your code; you can use our interactive runner for that. For more information, read the instructions in comments in that file, and also check out the Interactive Problems section of the FAQ.

Instructions for the testing tool are included in comments within the tool. We encourage you to add your own test cases. Please be advised that although the testing tool is intended to simulate the judging system, it is **NOT** the real judging system and might behave differently. If your code passes the testing tool but fails the real judge, please check the Coding section of the FAQ to make sure that you are using the same compiler as us.

Download testing tool

Sample Interaction

Judge	Solution
Constraints	
`2 3 3 1500`	
Judge provides $\mathbf{T}$, $\mathbf{N}$, $\mathbf{B}$, $\mathbf{P}$	
Case 1	
`3`	
Judge provides a block with a 3 on it	
	`1`
	Solution places it on pile 1
`2`	
Judge provides a block with a 2 on it	
	`1`

Solution places it on pile 1

5

2

4

2

1

1

We are now at the state shown in Picture 1

6

3

3

2

9

3

0

3

We are now at the state shown in Picture 4 (sum = 564)

Case 2

7

Judge provides the first block in the second case

3

Solution places it on pile 3

(... 7 exchanges omitted ...)

8

Judge provides the final block in the second case

2

Solution places it on pile 2 (sum = 1285)

Both Cases Finished

1

Judge confirms that the sum of both test cases (1849) is at least 1500