Smart Servers!

The work is going on at good pace with the OSIO team. Since OSIO is a huge product, there are several teams simultaneously working on it and Bayesian team is one of them which handles the analytics stuff.

Now, performing all the tasks related to analytics such as Data Gathering, Data filtering and training the model requires a lot of resources and hence the team decided to have 3 dedicated servers which work as follows:

- The first two servers are the main servers which handle all the requests. If any of the servers fail, the respective server would automatically forward the request to the third server. Since the servers have a limited processing capacity, it is made sure that the requests they receive can only consume processing power in the inclusive range of L and R. Assume that each server is running infinite processes for every processing power in the inclusive range of L and R.
- The third server is a back-up server which would handle the requests in case either/both of the main servers fail to serve the request due to any reason. It also has a predefined processing capacity of C.

Now, one fine day both the main servers failed due to a technical issue. So as described, they started forwarding the requests to the back-up server. Since, the back-up server has a limited capacity of C it can't handle all the requests at the same time from both the servers. Since servers are programmed to be smart, the two main servers forward the request to the back up server in the following way:

- First server always sends the request first and they take alternating turns.
- During each turn, the server can choose any request he receives, which as discussed has processing power consumption in the inclusive range of L and R.
- The back-up server starts dumping the requests when the running sum of processing power of sent requests is $\geq C$.
- Each server forwards the request to the back-up server optimally i.e if there is a request that has a processing capacity of p and the server forwards that request to the back-up server and the back-up server wouldn't dump that request, then the server would send that request.

Given the values of L, R and C for T test cases, print the number of the server which sends the last valid request (i.e not dumped by the back-up server) on a new line.

Input Format

The first line contains an integer, T, denoting the number of tests. Each of the subsequent lines contains three space-separated integers describing the respective values of L, R and C for a test.

Constraints

- $1 \leq T \leq 10^5$
- $1 \leq L \leq R \leq 10^6$
- $1 \leq C \leq 10^7$

Output Format

For each test, print the number on a new line (i.e `first` or `second`).

Sample Input 0

```
2
1 10 11
7 20 15
```

Sample Output 0

```
second
first
```

Explanation 0

We must evaluate the following $T = 2$ tests:

1. $L = 1$, $R = 10$ and $C = 11$. Regardless of which process the first server chooses, the second server can always choose the process that will bring the running sum of the processing power of sent requests ≥ 11. Because the second server will always be the last one to send the valid request, we print `second` on a new line.

2. $L = 7$, $R = 20$ and $C = 15$. The first server can choose the process with process consumption ≥ 15, at which point the running sum of the processing power of sent requests is ≥ 15. Because, the first server would be the last one to send the valid request, we would print `first` on a new line.

    - ***Note***: Any ***partially served*** request is also a valid request. For example, in the second test case above, C is 15 and the server is allowed to send any request with processing power (P) ≥ 15 in order to behave optimally. Now, initially the running sum of processing power of sent requests is 0 and if the server sends a request with any $P > 15$ then the server would only be able to ***partially service the request*** and the sum of processing power of sent requests would be > C, hence the server would start dumping **upcoming requests** and **not the current request**.