1007 Snatch Groceries

Time Limit: 2000/1000 MS (Java/Others)

Memory Limit: 65536/65536 K (Java/Others)

Problem Description

“SNATCH GROCERIES first, then get a covid test” has quickly become an anthem for the lockdown that started suddenly in Shanghai in the early hours of March 28th. We can describe scenes of panic buying—qiang cai, or snatching groceries—and the threat of being locked out of one’s home amid a frenzied bid to control an outbreak of covid-19 in China’s main business and finance hub. Here is the question, how does the server determine who succeeded When millions of people press the order button on their mobile phones at the same time.

Processing such a large number of requests requires a distributed system. In a distributed system, time is a tricky business, because communication is not instantaneous: it takes time for a message to travel across the network from one machine to another. The time when a message is received is always later than the time when it is sent, but due to variable delays in the network, we don’t know how much later. This fact sometimes makes it difficult to determine the order in which things happened when multiple machines are involved

Moreover, each machine on the network has its own clock, which is an actual hardware device: usually a quartz crystal oscillator. These devices are not perfectly accurate, so each machine has its own notion of time, which may be slightly faster or slower than on other machines. It is possible to synchronize clocks to some degree: the most commonly used mechanism is the Network Time Protocol (NTP), which allows the computer clock to be adjusted according to the time reported by a group of servers. The servers in turn get their time from a more accurate time source, such as a GPS receiver.

Modern computers have at least two different kinds of clocks: a time-of-day clock and a monotonic clock. A time-of-day clock does what you intuitively expect of a clock: it returns the current date and time according to some calendar (also known as wall-clock time). For example, clock_gettime(CLOCK_REALTIME) on Linux return the number of seconds (or milliseconds) since the epoch: midnight UTC on January 1, 1970, according to the Gregorian calendar, not counting leap seconds. Some systems use other dates as their reference point. Time-of-day clocks are usually synchronized with NTP, You may be able to read a machine's time-of-day clock with microsecond or even nanosecond resolution. But even if you can get such a fine-grained measurement, that doesn't mean the value is actually accurate to such precision. In fact, it most likely is not—as mentioned previously, the drift in an imprecise quartz clock can easily be several milliseconds, even if you synchronize with an NTP server on the local network every minute. With an NTP server on the public internet, the best possible accuracy is probably to the tens of milliseconds, and the error may easily spike to over 100 ms when there is network congestion.

Thus, it doesn't make sense to think of a clock reading as a point in time—it is more like a range of times, within a confidence interval: for example, a system may be 95% confident that the time now is between 10.3 and 10.5 seconds past the minute, but it doesn't know any more precisely than that. If we only know the time +/– 100 ms, the microsecond digits in the timestamp are essentially meaningless.

An interesting exception is Google's TrueTime API in Spanner, which explicitly reports the confidence interval on the local clock. When you ask it for the current time, you get back two values: [earliest, latest], which are the earliest possible and the latest possible timestamp. Based on its uncertainty calculations, the clock knows that the actual current time is somewhere within that interval. The width of the interval depends, among other things, on how long it has been since the local quartz clock was last synchronized with a more accurate clock source.

TL;DR: Spanner implements snapshot isolation across data centers in this way. It uses the clock's confidence interval as reported by the TrueTime API, and is based on the following observation: if you have two confidence intervals, each consisting of an earliest and latest possible timestamp ($A = [A_{earliest}, A_{latest}], B = [B_{earliest}, B_{latest}]$), and those two intervals do not overlap (i.e.,$A_{earliest} < A_{latest} < B_{earliest} < B_{latest}$), then B definitely happened after A——there can be no doubt. Only if the intervals overlap are we unsure in which order A and B happened.

Now we use Spanner as a solution, there are millions of people snatching groceries, and everyone is given the clock's confidence interval. The server executes each request in chronological order, and terminate in case of intervals overlap. Here is the question, how many people can get their food before the server is terminated.

Input

First line has one integer T, indicating there are T test cases. In each case:

First line has one integers n, indicating there are n people.

For next n lines, each line has 2 integers $earliest, latest$, indicates the clock's confidence interval.

$T \leq 10, 1 \leq n \leq 10^5, 0 \leq earliest_i < latest_i \leq 10^9$

Output

In each case, print one integer, indicates the answer.

Sample Input

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

Sample Output

```
3
0
```