Code Jam 2018 - Round 1A

Bit Party

Problem

These days, robots can drive cars, but can they throw a good party? The Code Jam team's research into this topic is still at an early stage. We just deployed **R** robot shoppers to our local supermarket to buy party supplies for our World Finals in Toronto, but their first-order model of a Canadian party was very simple: they just bought **B** "bits" (a bit being a small donut-like treat found in the area). We will work on improving their AI later, but for now, we want to help them purchase all of those bits as quickly as possible.

The supermarket has **C** cashiers who can scan customers' purchases. The i-th cashier will:

- accept a maximum of $\mathbf{M_i}$ items per customer
- take $\mathbf{S_i}$ seconds to scan each item
- spend a further $\mathbf{P_i}$ seconds handling payment and packaging up the bits.

That is, a customer who brings N bits to the i-th cashier (where N must be less than or equal to $\mathbf{M_i}$) will spend a total of $\mathbf{S_i} \times N + \mathbf{P_i}$ seconds interacting with that cashier.

Before the robots interact with any cashiers, you will distribute the bits among the robots however you want. (Bits must remain intact; you cannot break them up into fractional pieces!) Any robot that gets no bits will not get to interact with a cashier, and will go away disappointed.

Then, for each robot with at least one bit, you will choose a *different* single cashier. (Two robots cannot use the same cashier, and a robot cannot use more than one cashier.) The robots all start interacting with their cashiers at time 0. Note that once a robot finishes interacting with its cashier, it cannot be given more bits and cannot interact with other cashiers.

If you help the robots make optimal choices, what is the earliest time at which all of the robots can finish interacting with their cashiers?

Input

The first line of the input gives the number of test cases, **T**. **T** test cases follow. Each begins with one line with three integers **R**, **B**, and **C**: the numbers of robot shoppers, bits, and cashiers. Then, there are **C** more lines. The i-th of these represents the i-th cashier, and it has three integers $\mathbf{M_i}$, $\mathbf{S_i}$, and $\mathbf{P_i}$: the maximum number of bits, scan time per bit (in seconds), and payment/packaging time (in seconds) for that cashier, as described above.

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 the earliest time (in seconds) at which all robots can finish interacting with their cashiers.

Limits

$1 \le \mathbf{T} \le 100$.
$1 \le \mathbf{M_i} \le 10^9$, for all i.

$1 \le \mathbf{S_i} \le 10^9$, for all i.

$1 \le \mathbf{P_i} \le 10^9$, for all i.

The sum of the **R** largest values of $\mathbf{M_i} \ge \mathbf{B}$. (It is possible for at least one subset of **R** cashiers to handle all of the bits.)

Time limit: 15 seconds per test set.

Memory limit: 1GB.

Test set 1 (Visible)

$1 \le \mathbf{R} \le \mathbf{C} \le 5$.

$1 \le \mathbf{B} \le 20$.

Test set 2 (Hidden)

$1 \le \mathbf{R} \le \mathbf{C} \le 1000$.

$1 \le \mathbf{B} \le 10^9$.

Sample

Sample Input

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

Sample Output

```
Case #1: 5
Case #2: 4
Case #3: 7
```

In Sample Case #1, there are two robots, two bits, and two cashiers, and each cashier can only handle one item. So, you must give one bit to each robot. Cashier 1 takes 5 seconds, and Cashier 2 takes 3 seconds, so the time required is 5 seconds.

Sample Case #2 is similar to the previous case, except that now Cashier 2 can handle up to 2 items. So, it is best to give all the bits to one robot and have that robot use Cashier 2. This takes 1 second per item plus 2 seconds = 4 seconds.

In Sample Case #3, the optimal strategy is to send one robot with 2 bits to cashier 2, and two robots with 1 bit each to any of the other cashiers.