PayU Latam's Code Challenge #3

Disclaimer

By reading this disclaimer, I agree not to copy code from any source (including websites, books, or friends and colleagues) to complete this assessment. I also agree not to share this code challenge/assessment through any means. I may, however, reference programming language documentation or use an IDE that has code completion features.

Code Challenge

NetworkPrioritization

One constant task that is needed in PayU is prioritizing different payment networks that serve the same payment method. This is required for having better response time o lower costs. For example, we can have *Payment network 0* and *Payment network 1* payment networks which both process Visa credit cards. The prioritization is currently done by ordering the networks by different criteria, as of now by response time and/or cost, from lowest to highest.

For example, let say we want to prioritize by **response time**, and we have *Payment network 0* and *Payment network 1*, payment network 0 has an average response time of *20 milliseconds*, and payment network 1 has an average response time of *12 milliseconds*; in this case the order of the payment networks would be `{1, 0}`, since payment network 1 has a lower average response time.

Additionally when multiple networks have the same values for the criteria choosed for ordering, let say **response time** those payment networks with the same average response time are then ordered by **cost**, with the lowest cost being the first and so on.

Input

You will receive two arrays and a criteria for ordering.

The first array will contain the **average response time** for each network, ***each position of the array is a payment network***, position 0 of the array is payment network 0, postion 1 of the array is the payment network

1 and so on. `{20, 15, 100, 15, 50, 9}`

The second array will contain the **cost** per transaction in USD cents for each network, ***each position of the array is a payment network***, position 0 of the array is payment network 0, position 1 of the array is the payment network 1 and so on. `{50, 55, 30, 60, 40, 70}`

Finally, the criteria for ordering could be either `RESPONSE_TIME` or `COST`.

Expected output

The expected output is an array where each position is the payment network *(index of response time and cost arrays)* ordered by the specified criteria.
In the case of multiple equal values for the criteria, those with equal value must be ordered according to the *alternate* ordering criteria.

Example

Sample input

```
response time array:
{20, 15, 100, 15, 50, 9}
[0   1   2    3   4   5]
cost array:
{50, 60,  30, 55, 40, 70}
[0   1   2    3   4   5]
criteria:
RESPONSE_TIME
```

Sample output

```
{5, 3, 1, 0, 4, 2}
```

Explenation

Since the criteria for ordering is `RESPONSE_TIME` we first order the payment networks by the response time array. The fastest response time in this example is *9 milliseconds* which corresponds to the payment network 5 *(or position 5 in the array)*.
`{5, 1, 3, 0, 4 2}`
However, since both payment network, 1 and 3 have the same response time we have to order these two networks by the *alternate ordering criteria* in this case `COST`. For this example, the least expensive network between 3 and 1 is 3 since it cost USD 55 cents. This gives us the final result:

`{5, 3, 1, 0, 4, 2}`

Completing the code challenge

To complete the code challenge you must complete the method `prioritizeNetwork` inside the `NetworkPrioritization` class so that every unit test in `NetworkPrioritizationTest` passes.

Good luck!