RacerRater (Programming)

*Note: Please see the hint under the directions for one way to solve the problem. Only attempt an alternative solution if you are confident you can get it working in time.
Suppose you are a fan of auto-racing and want to figure out which drivers are likely to perform well in an upcoming race. Luckily you have access to a log of the times that each racer started and finished their test race the day before.
The particular rating algorithm you have chosen is to assign each racer R a score that equals the number of other racers who both started after R started and finished before R finished.
Note that a lower score generally suggests that the racer is faster, and this rating algorithm keeps from penalizing fast racers who have slow times simply because they are stuck behind a crash or slow racer. Additionally, this rating algorithm does not reward fast racers who pass tons of slow racers in comparison to fast racers who race when there are not many slow racers on the track to pass (compare this with rating a racer based on the net number of passes).

More formally, you want to write a program that will read the test race log from

have the following format:

racerId startTime endTime

where racerId is an integer in the range [0,10^9] and startTime and endTime are both integers such that 0 <= startTime < endTime <= 10^18. Each racerId will be distinct. Also, the collection of all start and end times will not contain any duplicate elements.

Given such an input, you should print output in the following format:

racerId score

where score is the score as defined above for racer racerId. The output lines should be sorted in ascending order of score with ties broken by sorting by racerId, also in ascending order. This can be accomplished with a simple sort at the end.
Directions:
Please code this problem in Java, C, C++, or Python (all else being equal, we prefer Java). Your solution should run in o(N^2) time on all inputs (i.e., strictly less than O(N^2) -- a Theta(N^2) algorithm such as naive brute force will not be fast enough -- please see http://en.wikipedia.org/wiki/Big_O_notation if you are not familiar with big-o, little-o, and Theta). A very fast Theta(N^2)-time implementation may pass all test cases, but please strive for better asymptotic performance as we review all submissions manually for code quality and asymptotic performance.
Hint: One possible way to achieve o(N^2) time (there are several other acceptable methods) is to use a data structure with K buckets (e.g., K = 300 or some function of input size), each of which is initially empty and is defined by two times. Each bucket will eventually contain racers whose start times fall between the bucket's start and end time. The bucket boundaries should be chosen such that they ultimately will contain the same number of racers. You can now iterate through the racers in end

time order and, as you iterate over each racer, fill in the bucketed data structure such that you can use it to quickly count the number of racers that finished before and started after that racer.

What We Are Looking For:

For this problem, we simply want to see that you can implement the algorithm correctly, without particular regard to principles of object orientation or modularity. Do give us at least minimal documentation to help us understand what you are trying to accomplish in certain key places of the algorithm.

Example:

input:
5
2 100 200
3 110 190
4 105 145
1 90 150
5 102 198
output:
3 0
4 0
1 1
5 2
2 3

Note in the above example that racer 3 has a score of 0 because no one starts after racer 3 (a drawback to this scoring system is the last racer always has a score of 0). Racer 4 also has a score of 0 because the only racer who starts after racer 4's start time (racer 3) has a later finish time. Racer 3 is listed ahead of racer 4 despite having a slower time because racer 3's id is lower. At the other end, racer 2 has a score of 3 because racers 3, 4, and 5 start after racer 2 and finish before racer 2 finishes.

YOUR ANSWER

C

```c
#include <stdio.h>
int main() {
    /* Enter your code here. Read input from STDIN. Print output to STDOUT */
    return 0;
}
```

Draft saved 11:36 am Line: 1 Col: 1

Compile & Test | Submit code & Continue

Download sample testcases *The input/output files have Unix line endings. Do not use Notepad to edit them on windows.*