2. The Best Time To Party

No one looks back on their life and remembers the nights they got plenty of sleep.
– Author Unknown.

Programming constructs and algorithmic paradigms covered in this puzzle: Tuples, lists of tuples, nested for loops and floating-point numbers. List splicing. Sorting.

There is a party to celebrate celebrities that you get to attend because you won a ticket at your office lottery. Because of the high demand for tickets you only get to stay for one hour but you get to pick which one since you received a special ticket. You have access to a schedule that lists when exactly each celebrity is going to attend the party. You want to get as many pictures with celebrities as possible to improve your social standing. This means you wish to go for the hour when you get to hob-nob with the *maximum* number of celebrities and get selfies with each of them.

We are given a list of intervals that correspond to when each celebrity comes and goes. Assume that these intervals are `[i, j)`, where `i` and `j` correspond to hours. That is, the interval is closed on the left hand side and open on the right hand side. This just means that the celebrity will be partying on and through the $\texttt{i}^{th}$ hour, but will have left when the $\texttt{j}^{th}$ hour begins. So even if you arrive on dot on the $\texttt{j}^{th}$ hour, you will miss this particular celebrity.

Here's an example:

Celebrity	Comes	Goes
Beyoncé	6	7
Taylor	7	9
Brad	10	11
Katy	10	12
Tom	8	10
Drake	9	11
Alicia	6	8

When is the best time to attend the party? That is, which hour should you go to?

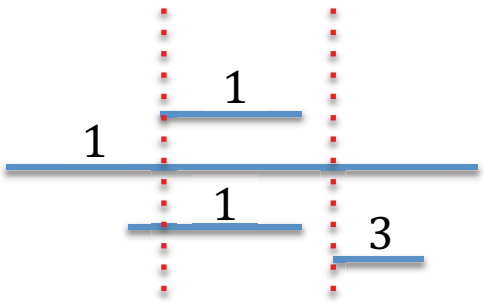
Exercises

Exercise 1: Suppose you are yourself a busy celebrity and don't have complete freedom in choosing when you can go to the party. Add arguments to the procedure

`bestTimeToPartySmart` and modify it so it determines the maximum number of celebrities you can see within a given time range between `ystart` and `yend`. As with celebrities the interval is `[ystart, yend)` so you are available at all times `t` such that `ystart <= t < yend`.

Exercise 2: There is an alternative way of computing the best time to party that does not depend on the granularity of time. We choose each celebrity interval in turn, and determine how many other celebrity intervals contain the chosen celebrity's start time. We pick the time to attend the party to be the *start* time of the celebrity whose start time is contained in the maximum number of other celebrity intervals. Code this algorithm and verify that it produces the same answer as the algorithm based on sorting.

Puzzle Exercise 3: Imagine that there is a weight associated with each celebrity dependent on how much you like that particular celebrity. This can be represented in the schedule as a 3-tuple, e.g., (6.0, 8.0, 3). The start time is 6.0, end time is 8.0 and the weight is 3. Modify the code so you find the time that the celebrities with *maximum total weight* are available. For example, given:



We want to return the time corresponding to the right dotted line even though there are only two celebrities available at that time. This is because the weight associated with those two celebrities is 4, which is greater than the total weight of 3 associated with the three celebrities available during the first dotted line.

Here's a more complex example:

```
sched3 = [(6.0, 8.0, 2), (6.5, 12.0, 1), (6.5, 7.0, 2),
          (7.0, 8.0, 2), (7.5, 10.0, 3), (8.0, 9.0, 2),
          (8.0, 10.0, 1), (9.0, 12.0, 2),
          (9.5, 10.0, 4), (10.0, 11.0, 2),
          (10.0, 12.0, 3), (11.0, 12.0, 7)]
```

For this schedule of celebrities, you want to attend at `11.0 o'clock` where the weight of attending celebrities is 13 and maximum!

MIT OpenCourseWare
https://ocw.mit.edu

6.S095 Programming for the Puzzled
January IAP 2018

For information about citing these materials or our Terms of Use, visit: https://ocw.mit.edu/terms.