SRINI DEVADAS: How many of you have attended a celebrity party? I haven't. Well, if you count MIT professors as being celebrities, I've been to a lot of them, but that's not the case, right? People don't really think of us as celebrities, which is fine with me. So this puzzle, I call it the best time to party. And-- I'm having a little trouble erasing the-- oh you're right. There's another one? All right. Well, I'm probably going to run out of this. But you know what? This time I'll use-- I don't think I need this super thick chalk. There's a slightly-- I'll use this one.

So setup here-- with all of these things, there's always a little bit of setup-- is, you got a ticket to this party. There are going to be all these celebrities. And it's a timed ticket with some flexibility. You only get to stay for a certain amount of time before you have to leave and other people are going to come in and hobnob with these celebrities. But you get to pick the particular time that you go, right? And so, initially, we'll set things up in a fairly straightforward way. We're going to say that you have an hour, a particular hour, maybe it's 6:00 to 7:00, 7:00 to 8:00, what have you. You get to choose. That's part of the puzzle.

How do you pick the best time to party? And you do have a schedule. And amazingly, these celebrities are going to stick to those schedules. But the schedule has been published with respect to when particular celebrities are going to be at this party. All right? And as you can imagine, as the title of the puzzle gives away, you want to choose the hour such that the maximum number of celebrities-- so you're not particularly interested in particular celebrities or one specific celebrity, you just wanted sort of max out number of selfies so you can put it on your Facebook page, Instagram, what have you. And you have that schedule and you're trying to optimize the number.

So here you go. We're going to have a-- let's just solve this again manually first, and then we'll think of algorithms. We're going to have intervals again. And so think of these as hours. Beyonce comes in at 6:00, leaves at 7. Taylor from 7:00 through 9:00. Brad 10:00 to 11:00. Katie 10:00 to 12:00. Tom 8:00 to 10:00. Going to run out of room here.

All right. So that's the schedule that's published, right? And we can think of these as hours-- comes and goes. There's one small subtlety, which is, I'm going to think of these as closed intervals on the left end and open at the right end. So all that means is, if this is 6:00 PM, and if you come in at 6:00 PM on the dot, well, you certainly get to see Beyonce. And if you come in at 6:59, you get to see Beyonce, but if you come in at 7:00, she's gone. Right? And the same thing with Taylor at 9:00. So you you have to choose your time properly. So what's the best time? You get to go for any hour that you choose. What's the best time to go to get the maximum number of selfies? Anyone?

You can do this manually. Tell me what the time is and then I'll ask you what your algorithm was? Right? So-- someone who hasn't answered yet. Yep?

AUDIENCE: Either 6:00 or 10:00.

SRINI DEVADAS: Either 6:00 or-- what's your name?

AUDIENCE: Kevin.

SRINI DEVADAS: Oh! Kevin. Right. Sorry. You're the other Kevin. All right. So-- 6:00 o'clock or 10:00 o'clock, because at 6:00 o'clock-- so who are the people you'd see at 6:00 o'clock?

AUDIENCE: Beyonce and Alicia Keys.

SRINI DEVADAS: That's only two.

AUDIENCE: Yeah.

SRINI DEVADAS: So can you do better than two? Can someone do better than what Kevin did? Yeah, go ahead. Kanishka

AUDIENCE: That will be Brad. So at 10:00, you'll get Brad. Katy, and Drake.

SRINI DEVADAS: Yes. So you could do three. At 10:00, you would get Brad. You would get Katie, and then you'd also get Drake. OK? So that's three. OK? So that's the best you can do in this puzzle, right? But you may have hundreds of celebrities and you want to pick the right time. So obviously, we want to write a computer program here, right? What algorithm, Kevin or Kanishka did you use to do this? How do you do it in your head?

AUDIENCE: I first of all saw which were the good candidates or shortlisting them essentially.

SRINI DEVADAS: Did you go, either one of you, did you go hour by hour? Did you go 6:00 o'clock-- how many people are there? Or did you try and intersect, or what did you do? What's a reasonable way of proceeding here? Right? What's a reasonable way of proceeding? Someone other than-- yeah, back there.

AUDIENCE: Look at the most frequent amount of time.

SRINI DEVADAS: The frequent amount of time? So look at the hours. You say 6:00 o'clock and you try and see how many sixes are up there? Or-- right. What's your name?

AUDIENCE: Nisha.

SRINI DEVADAS: So Nisha says she's going to go look for 6:00 o'clock and see if many sixes are up there. Right? And there's two sixes up there. And maybe that's why Kevin said six, right? On the other hand, clearly that didn't quite work in this case, because what happened was, 10:00 o'clock-- there's only two of those. And so that looks good, too, 6:00 and 10:00, but you missed the fact that 10 exists in 9:00 and 11:00, correct? So you're not explicitly seeing these things. But that gives you an algorithm. I mean, it does give you an algorithm. The algorithm is-- let me go ahead and do a little bit better than what you described, a little more relaxed check, right? It's not just an identical, I want six to be in the table. But I want to see if six is contained in the interval associated with any particular celebrity. Correct?

So if I did six, I'd say six is contained in this interval. It's not contained in this interval. Not, not, not, not, yes. Right? So I'd get two. And then when I've finally got to 10, I'd get contained, contained, contained, and I'd get three. Right? So there's clearly an algorithm here that depends on the number of hours that you're looking at. Depends, really, on the granularity of time, OK? So this particular algorithm would say, let me go ahead and enumerate the different times that I could possibly go. I could start 6:00. I could start at 7:00. But maybe we'll start at 6:30. I could go from 6:30 to 7:30. You're allowed to do that, right? The point is, if you stick with the original description, you could go ahead and enumerate 6:00 through 12:00, and then you could generate a bunch of different numbers associated with the number of celebrities and you could go ahead and pick what time you want to go. Right?

So it's a little bit like our previous puzzle in the sense that there's a relatively straightforward strategy that enumerates different hours and goes through and does a bunch of computation and then picks the maximum, in this case. Previously, it was the minimum for, you will all conform. Here, it's the maximum, right? What is a disadvantage of this particular algorithm that we just described? What is one potential disadvantage? Yeah? Go ahead, Fadi.

AUDIENCE: Well, it's not linear, so you have to make multiple passes on. First of all, you have to enumerate all of the possible times. And then for every time, you have to go through the list over and over again. So there's a lot of computational--

SRINI DEVADAS: There's a lot of computation. There's a lot of computational overhead. Let me show you what the code looks like for the straightforward algorithm, all right? And so it's going to, again, be intuitive. It's going to do exactly what we just described, right? And so if you look at what you have up there-- schedule up there is simply a set of intervals, right? And they're tuples. And so I have something that's bigger than this example here, but it includes that. And there's a whole bunch of things in there.

And now I'm going to look at-- this is just a start time for a particular celebrity. This is the end time for that celebrity. I'm just setting that up. And I'm going to go ahead and-- I'm looking at for each c in schedule, I'm going to look at this. I'm going to look at that. I'm going to look at that. I'm going to figure out what the start time is and the end is. And I'm just doing this simply because I want to figure out what the earliest start time for any of these celebrities are.

So in this case, it's six. And the latest time that a celebrity is around is 12, right? So that's all of this first thing does. It's just finding the range that I have to deal with. So there's not much there.

I'm going to go ahead and compute what's called the celebrity density. And I'll show you the code for that. But that's essentially what we describe, which is for a particular hour between start and end, how many celebrities do I see at that particular hour, right? And then for the next hour, how many celebrities do I see? I'm just calling it the density, right?

And then, the count is a list that is going to have celebrity densities for particular times that correspond to the indices of count. And I'm just going to go through that list and I'm going to figure out the time that has max count, or the maximum number of celebrities. And there's a couple of different ways of doing that. Don't worry about that, but I'm happy to answer questions if you have them.

And then, I just say the best time to end the party is at this time. Time, which was discovered here, and the number of celebrities who are going to be attending is max count. OK. So this is essentially the code for the algorithm.

And this thing here is doing the slightly more relaxed check that I alluded to, which was like when you have 6:00, you're not just looking for 6:00 here, you're looking for 6:00 within this interval that looks like this. So 6:00 is definitely within this, but it would not be-- 6:00 is not inside 5:00 and 6:00 as I described to you before, right?

So that's simply the check. That's why you have a less than equal to here and a strictly greater than over here, just to take care of that closed and open part of it. So again, if you don't understand every nuance in this code, it's not that big a deal. But hopefully you have the overall picture, right? Makes sense, right?

So as you can imagine, there's a much better way. There's a much better way that isn't as exhaustive as this one. And do people have a sense of what a better way would be, or would you like a hint? And anyone want to conjecture a different way of solving this problem that's-- yeah, go ahead.

AUDIENCE: The exhaustiveness of this algorithm?

SRINI DEVADAS: Ah, so the exhaustiveness of this algorithm is as follows. So basically what I'm saying is-- what I'm saying is I'm going to go look at the range of times. And all of these are per hour, right? Everything is per hour.

I'm going to look at the range of times, and I'm going to go ahead and include 6:00 through 12:00 here. And I'm going to go 6:00, and I'm going to go figure out at 6:00 how many celebrities exist. And then at 7:00, how many celebrities exist, and then at 8:00 all the way to 12:00, right? And I'm going to get in count, which is that list there, I'm going to get for each-- for count 6:00, I'm going to say two. And for count 10:00, I'm going to get three, because 10 is the hour.

And then I'm going to go through that list and figure out that 10 had the maximum number in it. And that's the exhaustiveness, right? That make sense?

So if we're good with that, this algorithm works. But it's a little painful, right? So what-- any ideas as to how we could do things quite differently, which would be more efficient? And let me give out the word. I want an incremental way of doing this.

How could I do this? I have a starting time, and I want to compute the same densities in an incremental way. Someone else other than Fadi? Yeah, back there.

AUDIENCE: Well, you have to start the n time thing just for each list of whatever tuple or whatever you could keep like a count-- a different count, a dictionary of the start time, and then increment through each of the start times until the end and just add one each time. So you're basically iterating through each set of times and finding the count of each time and whichever time is the most.

SRINI DEVADAS: You're absolutely on the right track. And it turns out you don't even need dictionaries. You can do this with just plain lists. But the idea here is actually a very compelling one. It's something that you'll see in other algorithms.

Whenever you have something where you're doing repeated computation, there's always-- many a time, there's a way of removing redundancy and turning into incremental computation, right? And so the insight here is the following. The only time that celebrity density, which is the number of celebrities that are in the room, changes is obviously when a celebrity enters or when a celebrity leaves, right? I mean, that's it.

I mean, we're all celebrities here. And I guess we did have someone leave, so there you go. And then maybe someone is going to come in, right?

So I just need to monitor sort of input and output, or entry and exit. And I also know what the endpoints are with respect to the 6:00, 6:00 PM versus 12:00, correct? So what I can do here is the following.

Suppose I had a chart that looks like this, right? And what I'm drawing here are these intervals. They aren't exactly what I had up there, but I'm drawing those intervals up, right? So you could think of this, if you'd like, as being 6:00. You can think of these two lined up as being 7:00, and so on and so forth.

So I could draw these out. And I don't really have to draw these out in a computer program, but I can certainly draw them out in terms of giving you the intuition, right? So what I'm going to do is I'm going to start with zero. So before 6:00, there's absolutely no one in the room, so I'm at zero.

And the moment I see-- I take one of these things, and if I see a celebrity, I can increment the count. So here at 5:00 I'm at zero, at 6:00 I get one. And at 7:00, no one has left, and two more people have come in, so I get three.

And now, the next thing happens out here. Let's say this one here is 8:00, I would get four for that. And then out here it's 9:00. I go 4 minus 1 equals 3, because this celebrity left.

So essentially, what I want to do is not have to compute this over and over for every time running through all of the list of celebrities like I did before, for 6:00, 7:00, through all the celebrities for 7:00-- I meant through all the celebrities. But if I sort the celebrities in increasing order of start times, essentially this picture is important and it requires sorting, because I went from left to right.

And when I go from left to right, I'm essentially saying I want to see things that come earlier before the ones that come later, right? That's essentially what time goes to the right, right? So I'm going from left to right.

So if I take these intervals and I sort them, and basically sort them by the start times, then all I need to do is, in the sorted list of intervals, I just showed you the computation that we have to perform. The computation that we have to perform is-- this was the first interval in the sorted list of intervals, because it was the leftmost interval. And I take that and I go ahead and increment it to one, right?

Then, the next thing I do is I don't have to worry about any time. Even if there was a time, if this was 6:00 and this was 8:00, for example, as long as there's no celebrity that came in at 7:00, there's no reason for me to change anything in my data structure, right? The only times that are interesting are when celebrities enter and leave.

So the other nice thing about this algorithm is that it will work if Beyonce came in at 6:39:39, 39 seconds-- 39 minutes, 39 seconds and left at 6:51 whatever, because I'm only concerned with the entry of a celebrity and the exit of a celebrity that correspond to these points that you see here, right? Does the algorithm make sense from an intuitive standpoint? Do you see what's happening here?

Hopefully the picture gives you a sense for how you're doing the correct computation. And the only times that things change are when-- at the end points of these intervals, right? When you see someone, increment, and when someone leaves, decrement, right?

So this set of numbers that you generate is essentially the density down at the bottom that corresponds to, at particular times, not necessarily on the hour, but whenever the celebrities come and go. And so that list that you get is all that you need to look at to figure out what the maximum is, right? So ultimately down below, in this particular example, perhaps four was the maximum.

And it started with zero. It climbs up, maybe goes to four, maybe goes to five, and then goes down again, maybe climbs up again. Who cares? Once you get to the point where you've written the program, size doesn't matter anymore, right?

So the last thing I'll do is I'll show you the code for this. It will be up on the website. And feel free to come to office hours and ask me questions about the code, or before lecture tomorrow.

But this is the entirety of the code. And I'm not even using the built in sort functions in Python. I went ahead and wrote it just to give you a sense of how long the code would be. But it's a much more elegant algorithm. It would run faster on large examples.

And I don't want to get into asymptotic complexity. That's really beyond this class. But as you can imagine, rather than going through the list of celebrities over and over, we're actually doing sorting exactly once, right? So that's better.

But essentially, what happens here is simple. You go ahead and you create each of these times. So this is particular celebrity, particular time, and this is the start point of the celebrity. So you have to mark that. This is very different from that.

What happens here is I am adding. What happens here is I'm subtracting. So I have to know whether this point is very different from that point. This is the same as that, same as that. And all of these are similar, but I have to differentiate between the two for obvious reasons.

So I have to have start and end. And then I go ahead and take these times. Each of these times that you see with these black dots here, they all need to be sorted, because that's the only way I'm going to go from left to right, right? That's the only way, by going through that list of times.

So I'm going to go ahead and do that sorting. And I could have done times dot sort by the way. I don't have to have sort list, but I just, as I said, I wanted to show you the code without using any Python library functions. And then choose time is simply looking through that list, finding that maximum number, be it four or five, and returning the index of that, because that tells you what the particular time is.

And that time will always be-- can someone tell me? Last question and I'll let you go. What am I going to get-- on any given example, what am I going to get as the best time, any given example?

I'll get you if Ganatra. Yeah?

AUDIENCE: A time when more people come and leave.

SRINI DEVADAS: A time when more people come and leave. That's correct, but I want something stronger. What's your name?

AUDIENCE: Josh.

SRINI DEVADAS: Josh? Yeah, Josh is right, but I want something even stronger than a time that more people come than leave. I mean, it's correct, but a variant observation. Someone else?

I want to say something about-- there's two sets of points here, right? There's two sets of points here. Which set of points are going to be the ones that are in play, with respect to the maximum time. That's maybe a better question. Go ahead.

AUDIENCE: The arrival time of someone.

SRINI DEVADAS: That's right. That arrival time of a celebrity, right? And it's true that what you said, George is correct. Ultimately, the best time is going to be when the maximum number of arrivals happen. But the bottom line is, I'm never going to-- this is actually an interesting observation, and there's an exercise related to this observation for this puzzle that you might want to look at that I'll put up on the website as soon as we're done here.

You're always going to pick one of these beginning times, because that's when the increment happens. You'd never pick one of these, because that's when a decrement happens, right? So that's all I wanted to say.

And the reason I wanted to say that, as I said, there's an extra question on top of this, and yet another algorithm to solve this that is really based on that final observation that we made. All right. Well, thank you. Sorry for going over, but as I said, this is IAP. Forgive me.