Interview Cake

Your company built an in-house calendar tool called HiCal. You want to add a feature to see the times in a day when *everyone* is available.

To do this, you'll need to know when *any* team is having a meeting. In HiCal, a meeting is stored as tuples of integers `(start_time, end_time)`. These integers represent the number of 30-minute blocks past 9:00am.

For example:

```python
(2, 3)  # meeting from 10:00 - 10:30 am
(6, 9)  # meeting from 12:00 - 1:30 pm
```

Write a function `merge_ranges()` that takes a list of meeting time ranges and returns a list of condensed ranges.

For example, given:

```python
[(0, 1), (3, 5), (4, 8), (10, 12), (9, 10)]
```

your function would return:

```python
[(0, 1), (3, 8), (9, 12)]
```

Do not assume the meetings are in order. The meeting times are coming from multiple teams.

Write a solution that's efficient even when we can't put a nice upper bound on the numbers representing our time ranges. Here we've simplified our times down to the number of 30-minute slots past 9:00 am. But we want the function to work even for very large numbers, like Unix timestamps. In any case, the spirit of the challenge is to merge meetings where `start_time` and `end_time` don't have an upper bound.

Gotchas

Look at this case:

```
[(1, 2), (2, 3)]
```
Python

These meetings should probably be merged, although they don't exactly "overlap"—they just "touch." Does your function do this?

Look at this case:

```
[(1, 5), (2, 3)]
```
Python

Notice that although the second meeting starts later, it ends before the first meeting ends. Does your function correctly handle the case where a later meeting is "subsumed by" an earlier meeting?

Look at this case:

```
[(1, 10), (2, 6), (3, 5), (7, 9)]
```
Python

Here *all* of our meetings should be merged together into just `(1, 10)`. We need keep in mind that after we've merged the first two we're not done with the result—the result of that merge *may itself need to be merged into other meetings as well.*

Make sure that your function won't "leave out" the *last* meeting.

We can do this in $O(n \lg n)$ time.

Breakdown

If you subscribe to our weekly newsletter (/free-weekly-coding-interview-problem-newsletter), you'll get another free full access question every week.

To see the full solution and breakdown for *all* of our questions, you'll have to upgrade to full access (/upgrade).

Upgrade now ➜ (/upgrade)

Solution

This part requires full access (/upgrade)

You've already used up your free full access questions!

If you subscribe to our weekly newsletter (/free-weekly-coding-interview-problem-newsletter), you'll get another free full access question every week.

To see the full solution and breakdown for *all* of our questions, you'll have to upgrade to full access (/upgrade).

Upgrade now ➜ (/upgrade)

Complexity

$O(n \lg n)$ time and $O(n)$ space.

Even though we only walk through our list of meetings once to merge them, we *sort* all the meetings first, giving us a runtime of $O(n \lg n)$. It's worth noting that *if* our input were sorted, we could skip the sort and do this in $O(n)$ time!

We create a new list of merged meeting times. In the worst case, none of the meetings overlap, giving us a list identical to the input list. Thus we have a worst-case space cost of $O(n)$.

Bonus

1. What if we *did* have an upper bound on the input values? Could we improve our runtime? Would it cost us memory?
2. Could we do this "in-place" on the input list and save some space? What are the pros and cons of doing this in-place?

What We Learned