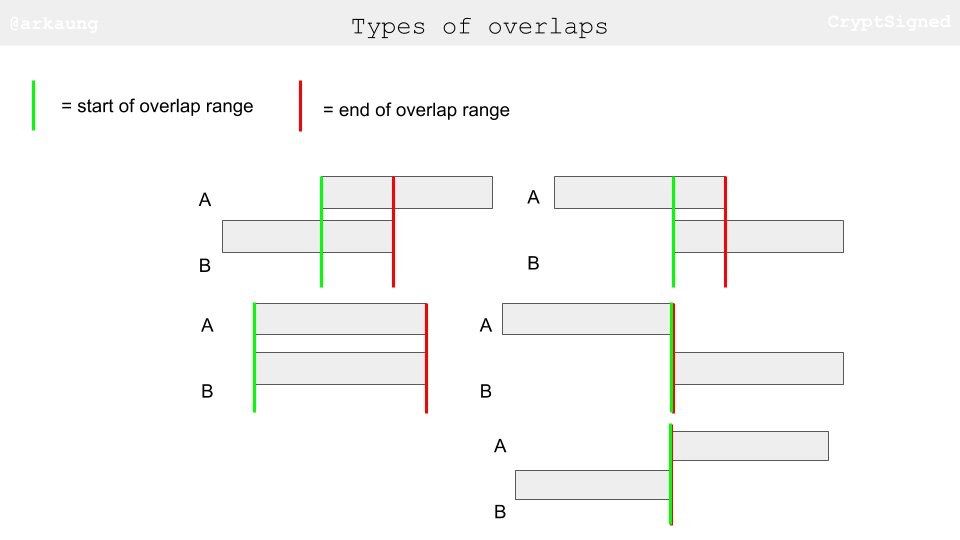
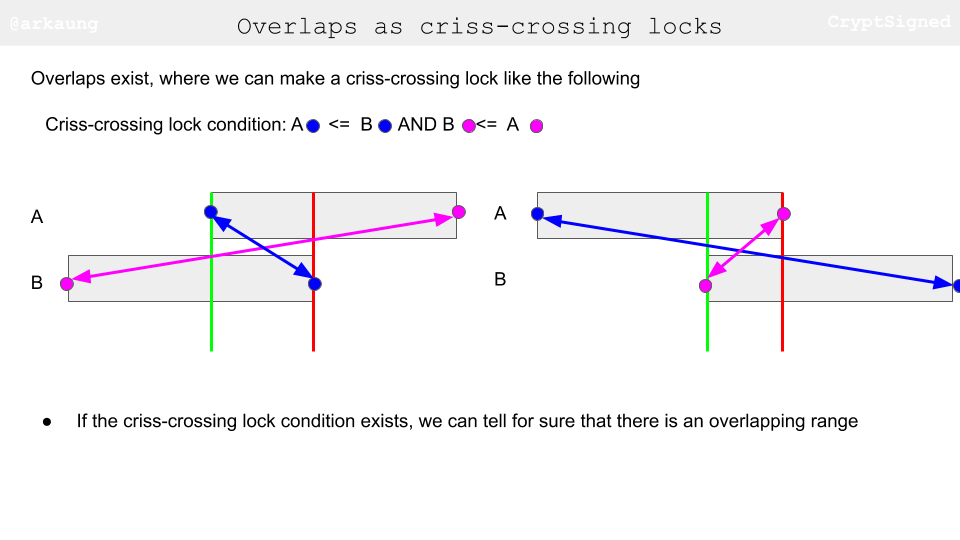
<https://leetcode.com/problems/interval-list-intersections/discuss/647482/Python-Two-Pointer-Approach-%2B-Thinking-Process-Diagrams>

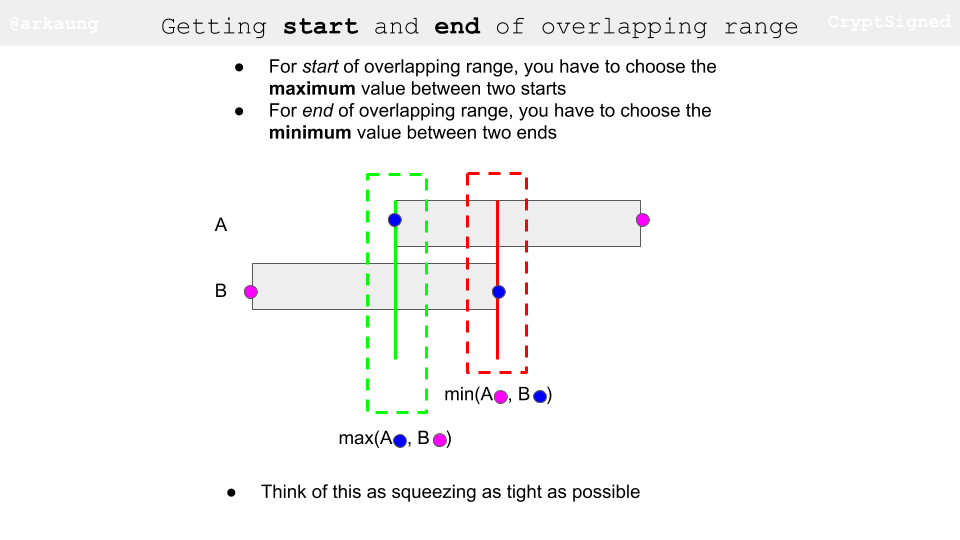
Two Pointer Approach + Thinking Process Diagrams

I love this problem. Even though it is a two pointer problem, making everything concise is quite hard!

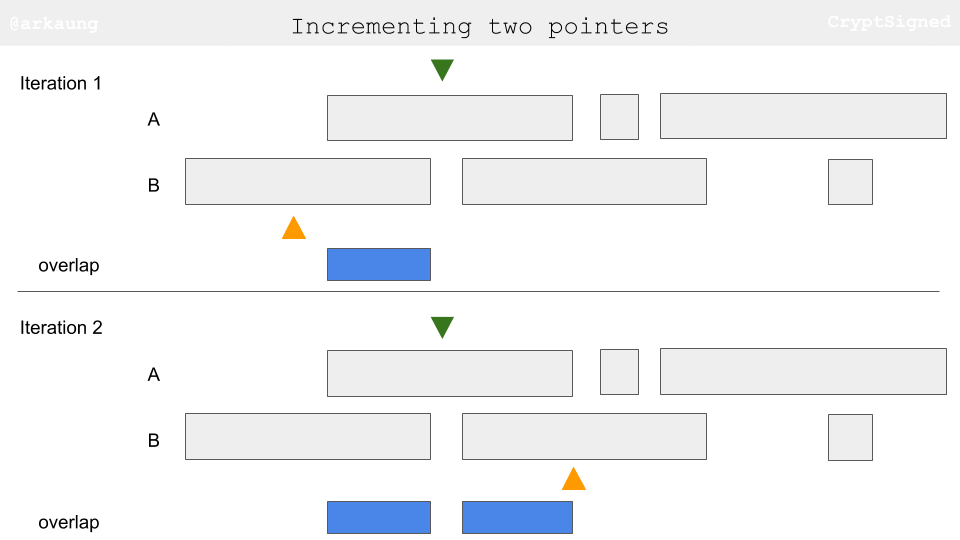
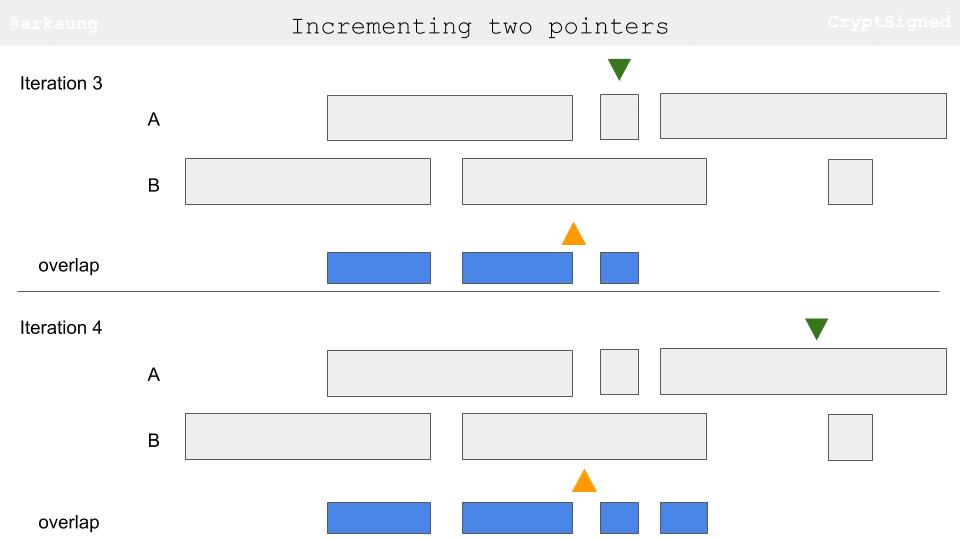
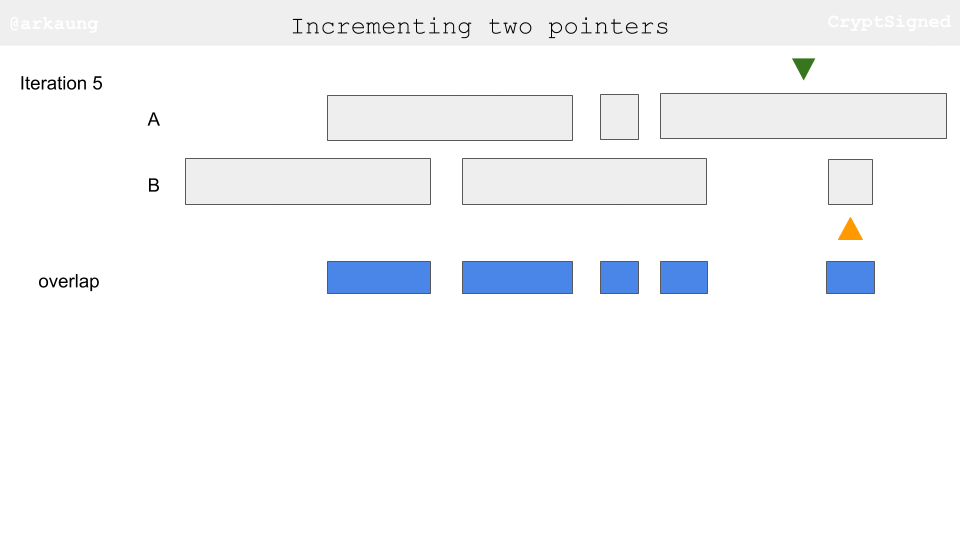
Let's break down this problem into two parts:  
**First part** - Finding overlapping range

Let's see possible types of overlaps  


What's the condition for two ranges to have an overlapping range? Check out the figures below. The way I think is: if we can have a criss-crossing lock condition satisfied, we have essentially found an overlapping range.  


After we have made sure that there is an overlapping range, we need to figure out the start and end of the overlapping range. I think of this as trying to squeeze the overlapping range as tight as possible (pushing as far *right* as possible for start and pushing as far *left* as possible for end)  


**Second part** - Incrementing pointers

The idea behind is to increment the pointer based on the end values of two ranges. Let's say the current range in A has end value smaller than to equal to end value of the current range in B, that essentially means that you have exhausted that range in A and you should move on to the next range. Let's try to visually think about this. When you are going through the images, keep track of end values of the ranges and how the little pointer triangles progess.  
  
  


**Python**

1. class Solution:

2. def intervalIntersection(self, A: List[List[int]], B: List[List[int]]) -> List[List[int]]:

3. i = 0

4. j = 0

5.

6. result = []

7. while i < len(A) and j < len(B):

8. a\_start, a\_end = A[i]

9. b\_start, b\_end = B[j]

10. if a\_start <= b\_end and b\_start <= a\_end: # Criss-cross lock

11. result.append([max(a\_start, b\_start), min(a\_end, b\_end)]) # Squeezing

12.

13. if a\_end <= b\_end: # Exhausted this range in A

14. i += 1 # Point to next range in A

15. else: # Exhausted this range in B

16. j += 1 # Point to next range in B

17. return result