Deacon Steiner COSC3020

Lab 1 Explanation

METHOD:

I had to think about the best ways to determine overlaps, because i could see visually where there would be overlaps. I eventually decided to sort the intervals by start time. I realized this would be useful while reading through all of the documents on lambda functions we were provided. From there it was as simple as comparing the start time of one meeting to the end time of the previous one.

APPROACH:

My function takes in a list of intervals, sorts them using a lambda function as a key in order to sort by start times. Then compares each meeting start time to the end time of the prior meeting, returning false if there is overlap. If the program cycles through all meetings without returning early then there must not be any overlap and so returns true.

REASONING:

My method works because once it is sorted you can check to see if any meeting starts before the previous one finishes. It is efficient because Python's sort() method runs in O(nlog(n)) time and the check itself is O(n) time. So the worst case is only O(nlog(n)) and if the list is already sorted then the function becomes O(n).

REFLECTION:

I used these three test cases:
interval1 = [[0, 30], [5, 10], [15, 20]]
interval2 = [[7, 10], [2, 4]]
interval3 = [[5, 7], [15, 30], [0, 5]]

The first two are the cases from the example. I used these to check my solution against an interval set with a known answer.
The final set ensured that the end point checking was non-inclusive. This was discussed in lab as something that should return True.

For further review of my code or any tests not included in the submission, you can see the full project here:
https://github.com/DeaconSteiner/cosc-3020/tree/main/Lab01