Name: Bryant Martinez

Email: Bryantmartinez322@csu.fullerton.edu

REPORT:

Problem:

Analogy for the Problem:
Assume there are at least two persons in your class project group. You want to schedule a meeting with another group member. You are provided with (a) a schedule of members' daily activities, containing times of planned engagements. They are not available to have a meeting you during these periods; (b) the earliest and latest times at which they are available for meetings daily. Your schedule and availabilities are provided too.

Input:

Five different types on inputs were required.

- **Person1_Schedule (AKA *BUSY_SCHEDULE*):** A integrated list of time for when **Person1** is available during the day.
- **Person1_DailyAct (*AKA WORK_PERIOD*):** List of planned engagement for **Person1.**
- **Person2_Schedule (AKA *BUSY_SCHEDULE*):** A integrated list of time for when **Person2** is available during the day.
- **Person2_DailyAct: (*AKA WORK_PERIOD*):** List of planned engagement for **Person2.**
- **Duration:** Required length of meeting.

Output:

- **Available_Slots:** The index of the available time periods in which both people area free from their schedules.

Constraints and Assumptions:

1. Time Format: Time must be in a consistent format (e.g., 24-hour).

2. Duration : Meeting duration must be a positive value.

3. Non-Overlapping Activities: Daily activities must not overlap.

4. Input Validity: All input must be formatted correctly and contain valid time ranges.

5. Single Day Scheduling: The scheduling is for a single day only.

Assumptions:

1. Participation: Both persons will attend if a meeting time is found.

2. Independence: Each person's schedule is independent of the other's.

3. Time Zone Consistency: Both persons are in the same time zone.

4. Accurate Input: Users provide accurate schedules without error-checking.

PSEUDOCODE:

```
function FindAvailableSlots(Person1_Schedule, Person1_DailyAct, Person2_Schedule,
Person2_DailyAct, Duration):
  initialize Available_Slots as empty list  // Initialize empty list for available slots

  // Iterate through each time slot for Person 1
  for each slot1 in Person1_Schedule:
    // Ensure no conflict with daily activities
    if slot1 is not in Person1_DailyAct:
      // Iterate through each time slot for Person 2
      for each slot2 in Person2_Schedule:
        // Ensure no conflict with daily activities
        if slot2 is not in Person2_DailyAct:
          // Check for overlapping time
          if Overlap(slot1, slot2, Duration):
            add overlapping time to Available_Slots  // Add overlapping time to available
slots

  return Available_Slots  // Return the list of available slots

// Function to check if two slots overlap
function Overlap(slot1, slot2, Duration):
  Start1, End1 = slot1  // Get start and end of slot1
  Start2, End2 = slot2  // Get start and end of slot2

  // Check if slots overlap and can fit the duration
  if Start1 < End2 and End1 > Start2 and (End1 - Start1 >= Duration) and (End2 - Start2 >=
Duration):
    return True  // Return True if overlap is valid
```

```
   return False  // Return False if no valid overlap

// Sample input for testing
initialize Person1_Schedule as [[7:00, 8:30], [12:00, 13:00], [16:00, 18:00]]
initialize Person1_DailyAct as [[10:00, 11:00]]  // Example daily activities
initialize Person2_Schedule as [[9:00, 10:30], [12:20, 13:30], [14:00, 15:00], [16:00,
17:00]]
initialize Person2_DailyAct as [[11:00, 12:00]]  // Example daily activities
initialize Duration as 30  // Minimum time for meeting in minutes

// Call the function to find available meeting times
Available_Slots = FindAvailableSlots(Person1_Schedule, Person1_DailyAct,
Person2_Schedule, Person2_DailyAct, Duration)

// Print the available slots
print Available_Slots  // This will print the result
```

Big(O) Notation:

CODES:	**EFFICIANCY CLASS:**	**EXPLANATION:**
return hours * 60 + minutes	**O(1)**	Constant time for converting time formats.
return f'{hours:02}:{minutes:02}'	**O(1)**	Constant time for converting time formats.
start, end = map(time_to_minutes, working_period	**O(1)**	Constant time for initial setup.
for busy in busy_schedule:	**O(n)**	Linear time based on the number of busy slots n.
if current < busy_start and (busy_start - current) >= min_duration:	**O(1)**	Constant checks within the loop.
for slot1 in common_times:	**$O(n^2)$**	Quadratic for nested loops comparing free slots.
for slot2 in member_times:	**$O(n^2)$**	Quadratic for nested loops comparing free slots.

Person1_Schedule	**O(n)**	Linear time based on the number of entries in Person 1's schedule n.
Person2_Schedule	**O(n)**	Linear time based on the number of entries in Person 2's schedule n.
Person1_DailyAct	**O(n)**	Linear time based on the number of entries in Person 1's daily activities n.
Person2_DailyAct	**O(n)**	Linear time based on the number of entries in Person 2's daily activities n.
Available_Slots = []	**O(1)**	Constant time for initializing an empty list.

Proving Efficiency Class Using Limits

The overall complexity is:
O(n^2)(due to the nested loop through Person 1's and Person 2's schedules)

f(n)=c1·n2+c2·n+c3

g(n)=n2

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{C_1 * n^2 + C_2 * n + C_3}{n^2}$$

Simplify the limit expression,

$$\lim_{n\to\infty}\left(C_1 + \frac{C_2}{n} + \frac{C_3}{n^2}\right)$$

Evaluate the limit

$$As\ n \to \infty, \frac{C_2}{n} \to 0$$
$$As\ n \to \infty, \frac{C_3}{n^2} \to 0$$

Thus,

$$\lim_{n\to\infty}\left(C_1 + \frac{C_2}{n} + \frac{C_3}{n^2}\right) = C_1 + 0 + 0 = C_1$$

Since, C_1 ***is a constance, we can conclude that,***

$$\lim_{n\to\infty}\frac{f(n)}{g(n)} = C_1 > 0$$

Because of this,

$$f(n) = O(n^2)$$