Robert Olivares - robertolivares@csu.fullerton.edu
Dylon Gaddi - dylongaddi@csu.fullerton.edu

335 Project 1

Github Link: https://github.com/rolivares14/335project1

Algorithm Output:

output.txt

```
Group 1: [('9:00', '10:15'), ('11:00', '11:30'), ('13:00', '14:00'), ('16:00', '18:30')]
Group 2: [('16:00', '18:30')]
Group 3: [('9:00', '10:00'), ('13:00', '14:00'), ('16:00', '18:30')]
Group 4: [('10:30', '12:00'), ('15:00', '16:00'), ('18:00', '18:30')]
Group 5: [('9:00', '10:15'), ('11:00', '11:30'), ('13:00', '14:00'), ('16:00', '18:30')]
Group 6: [('9:00', '10:00'), ('13:00', '14:00'), ('16:00', '18:30')]
Group 7: [('8:00', '9:00'), ('13:00', '20:00')]
Group 8: [('8:00', '9:00'), ('12:00', '13:00'), ('16:00', '18:00')]
Group 9: [('8:00', '9:00'), ('11:00', '12:00'), ('14:00', '15:00'), ('17:00', '20:00')]
Group 10: [('16:00', '18:00')]
Group 11: [('7:00', '8:00'), ('10:00', '11:00'), ('13:00', '19:00')]
Group 12: [('8:00', '9:00'), ('11:00', '12:00'), ('14:00', '18:00')]
Group 13: [('7:00', '8:00'), ('10:00', '11:00'), ('13:00', '15:00'), ('17:00', '20:00')]
Group 14: [('8:00', '9:00'), ('12:00', '13:00'), ('16:00', '17:00')]
Group 15: [('7:00', '8:00'), ('11:00', '12:00'), ('15:00', '18:00')]
```

Step Count and Math Analysis of Algorithm

1. Helper function- time_to_minutes(): This function takes **O(1)** time for each call. While .split() in python is implemented in **O(n)** time, because the format of the data we are converting is always a string “00:00” of length 5, this function call is actually constant, so it is **O(1)**

2. Converting busy schedules to minutes using time_to_minutes function for each busy time(busy1_minutes and busy2_minutes): This takes **O(n)** for the first list and **O(m)** for the second, **O(n + m)**

3. Merging the busy schedules (merged_busy): This takes **O(n+m)** time.

4. Sorting merged_busy: This takes **O(p log p)** time. Where **p = n + m**, alternatively **O((n + m) log(n + m))**

5. Finding intersection of working periods, Find min free from of both people and max free to of both people, **O(1 + 1)**, constant this input will only ever have 1 array for each person

6. Looping through merged_busy to find free slots and append to common free time: This takes **O(n+m)** time.

7. Converting the free slots back to HH:MM format: This takes **O(p)** time, where p is the number of free slots found, which is at most **p = n+m**.

Adding all these up, the overall time complexity is: **O(n+m) + O(n+m) + O(p log p) + O(1) + O(1) + O(n + m) + O(p) = 3(n+m) + p + p log p + 2**

Step Count: **3n + 3m + (n+m) + (n+m)*log(n+m) + 2; let p = n + m**
Step Count = **4p + p log p + 2**

So, the algorithm belongs to the time complexity of **O(p log p).** Where **p = n + m**, and
n is the size of busy schedule1 and m is the size of busy schedule2

Proof

$$\lim_{p\to\infty} \frac{T(p)}{f(p)} = \lim_{p\to\infty} \frac{p\log p + 4p + 2}{p\log p} = \lim_{p\to\infty} 1 + \frac{4p}{p\log p} + \frac{2}{p\log p}$$
$$= 1 + 0 + 0 = 1$$

Since the limit gives us 1 which is a constant, T(p) belongs to f(p), **p log p + 4p + 2** belongs to **O(p log p)**

Can we do better?

Given that we have to sort the schedules to get our answer, **O(p log p)** is most likely the most optimal solution in terms of time complexity.

The complexity class O(p log p) will remain the same even if p increases. The algorithm scales logarithmically with the size of the input, which is generally quite efficient. Adding more schedules to an individual person would not change the complexity class. Adding more individual people than 2 would also not change the complexity class as it would simply add to the sum of P.