Ethan Wu
Haroutyun Chamelian

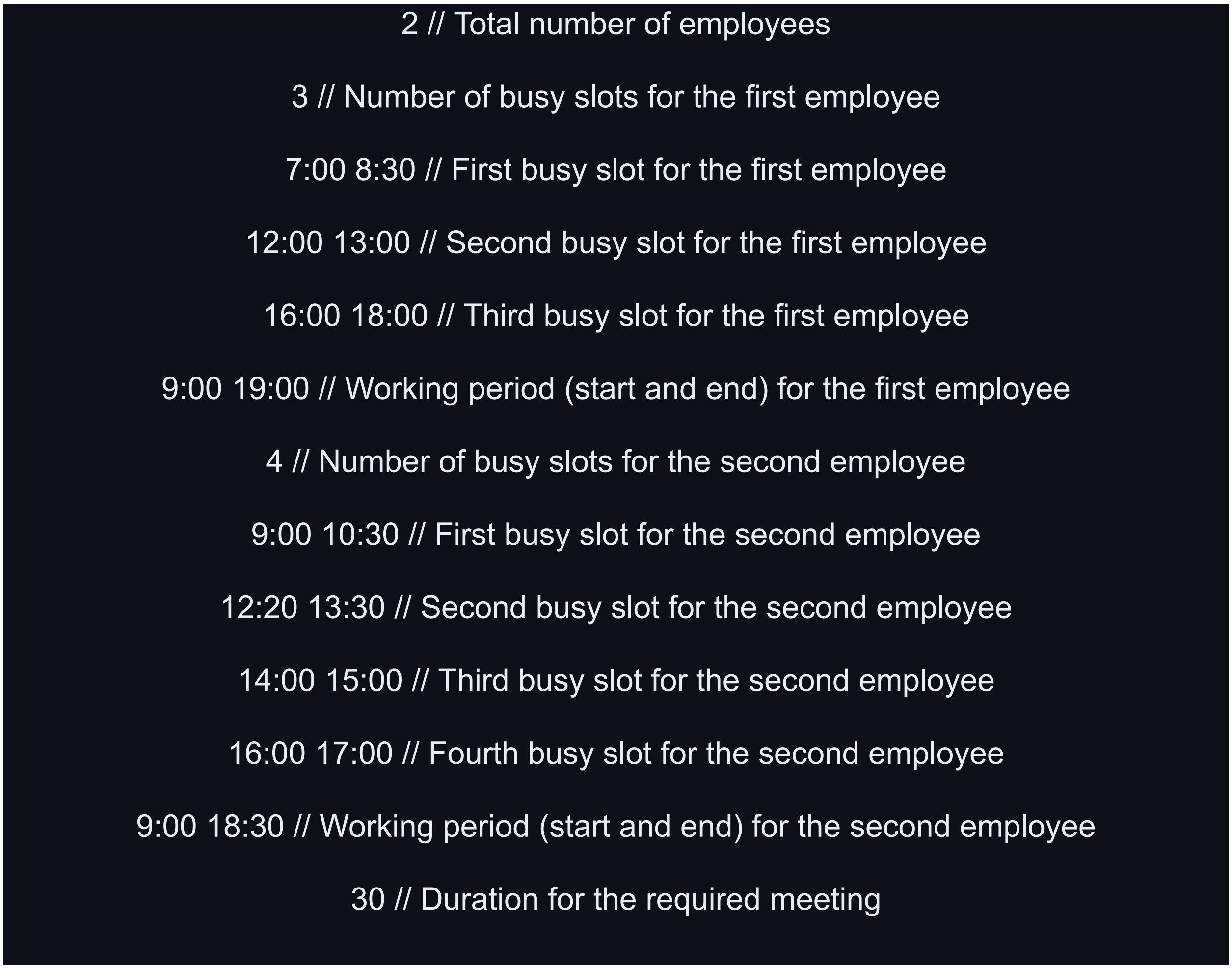
Project one

https://github.com/ethanawu/CPSC-335-Project-1

CASES

How input works

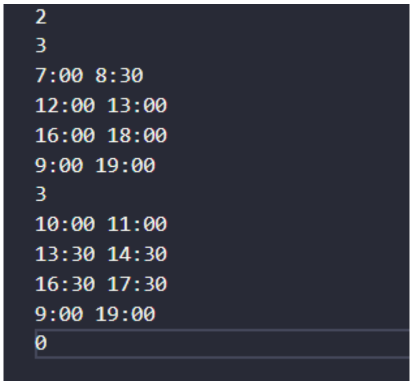
EXAMPLE



```
2 // Total number of employees

3 // Number of busy slots for the first employee

7:00 8:30 // First busy slot for the first employee

12:00 13:00 // Second busy slot for the first employee

16:00 18:00 // Third busy slot for the first employee

9:00 19:00 // Working period (start and end) for the first employee

4 // Number of busy slots for the second employee

9:00 10:30 // First busy slot for the second employee

12:20 13:30 // Second busy slot for the second employee

14:00 15:00 // Third busy slot for the second employee

16:00 17:00 // Fourth busy slot for the second employee

9:00 18:30 // Working period (start and end) for the second employee

30 // Duration for the required meeting
```

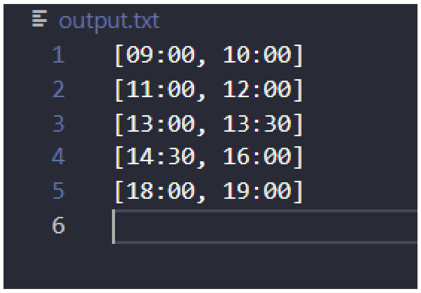
Case 1: Completely Overlapping Working Hours, No Meeting Duration

Input



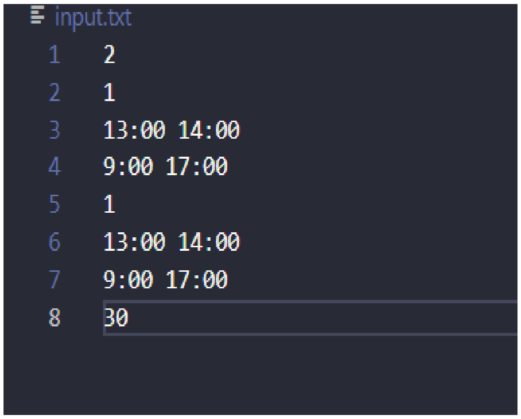
```
2
3
7:00 8:30
12:00 13:00
16:00 18:00
9:00 19:00
3
10:00 11:00
13:30 14:30
16:30 17:30
9:00 19:00
0
```

Output



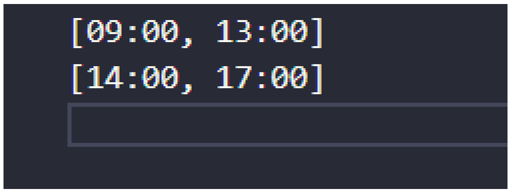
```
[09:00, 10:00]
[11:00, 12:00]
[13:00, 13:30]
[14:30, 16:00]
[18:00, 19:00]
```

Case 2: Completely Overlapping Working Hours, No Available Slots



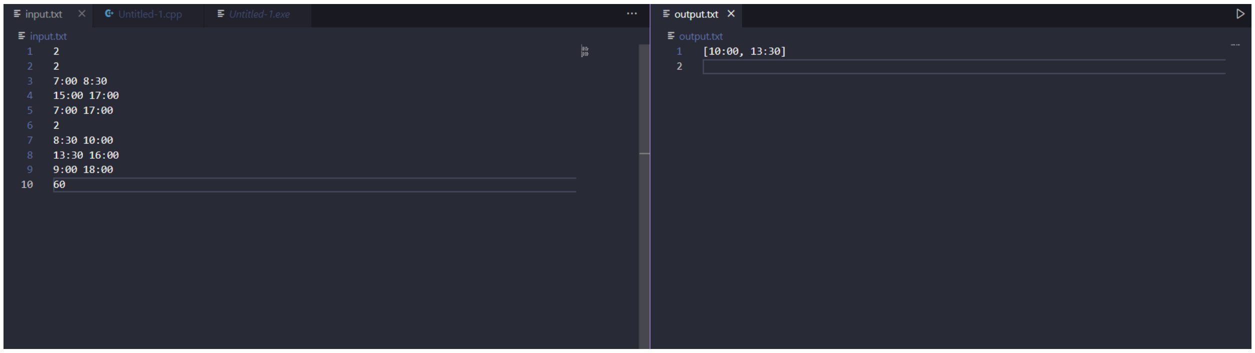
```
2
1
13:00 14:00
9:00 17:00
1
13:00 14:00
9:00 17:00
30
```

Output



```
[09:00, 13:00]
[14:00, 17:00]
```

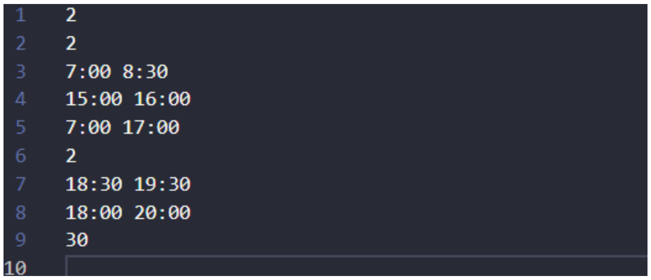
Case 3: Partially Overlapping Working Hours



```
2
2
7:00 8:30
15:00 17:00
7:00 17:00
2
8:30 10:00
13:30 16:00
9:00 18:00
60
```

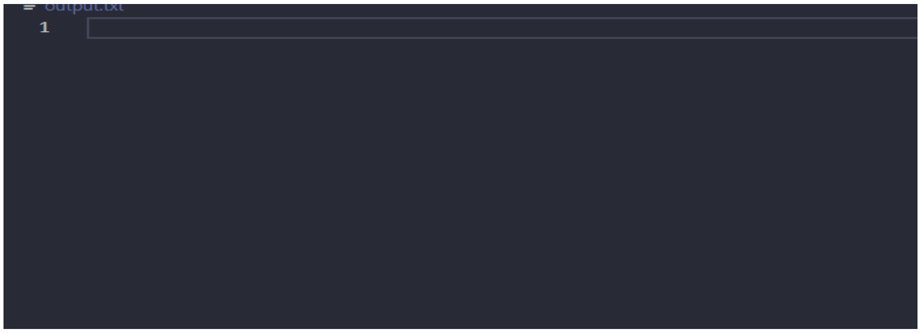
```
[10:00, 13:30]
```

Case 4: No Overlapping Working Hours

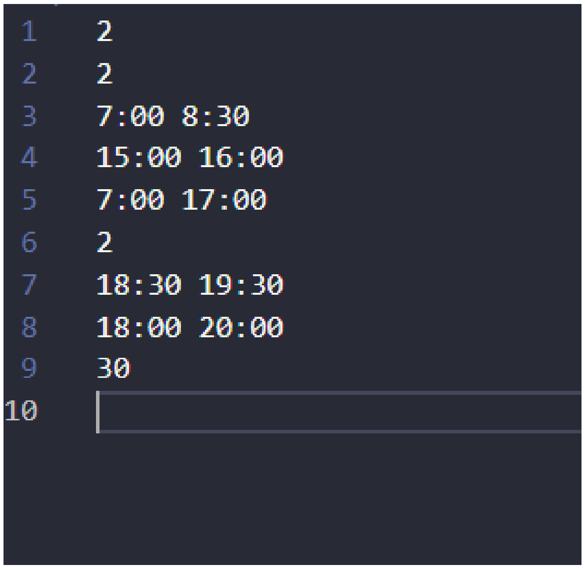


```
2
2
7:00 8:30
15:00 16:00
7:00 17:00
2
18:30 19:30
18:00 20:00
30
```

Output

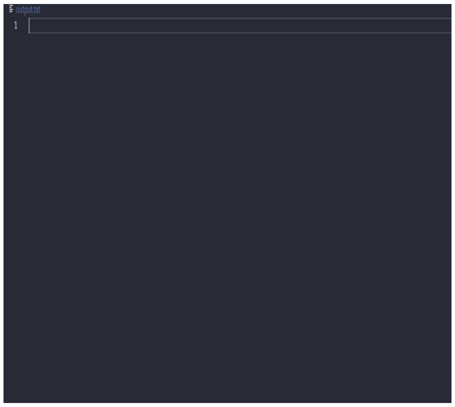


Case 5: Completely Overlapping Working Hours, Fully Busy

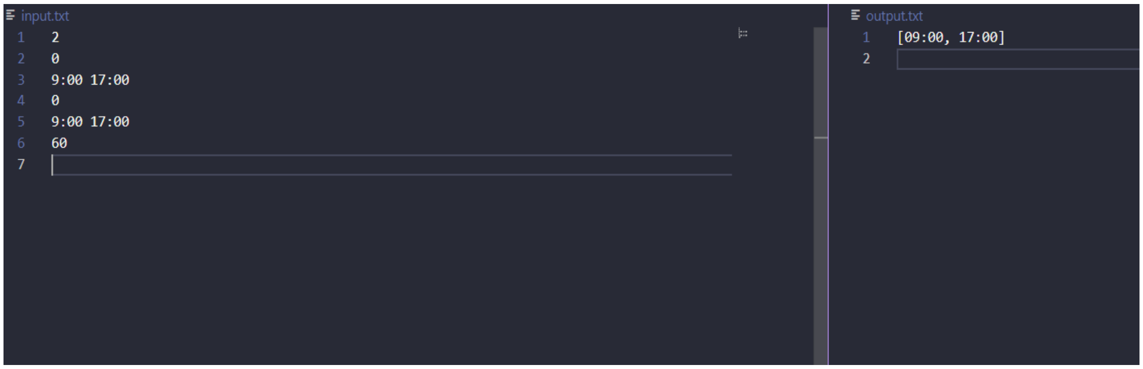


```
2
2
7:00 8:30
15:00 16:00
7:00 17:00
2
18:30 19:30
18:00 20:00
30
```

Output



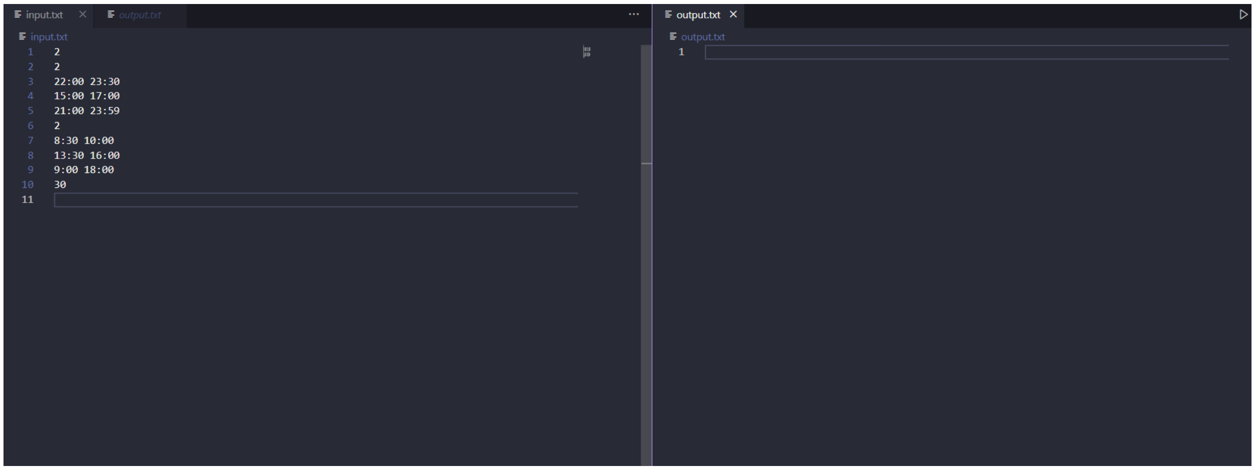
Case 6: One Employee Completely Free



```
2
0
9:00 17:00
0
9:00 17:00
60
```

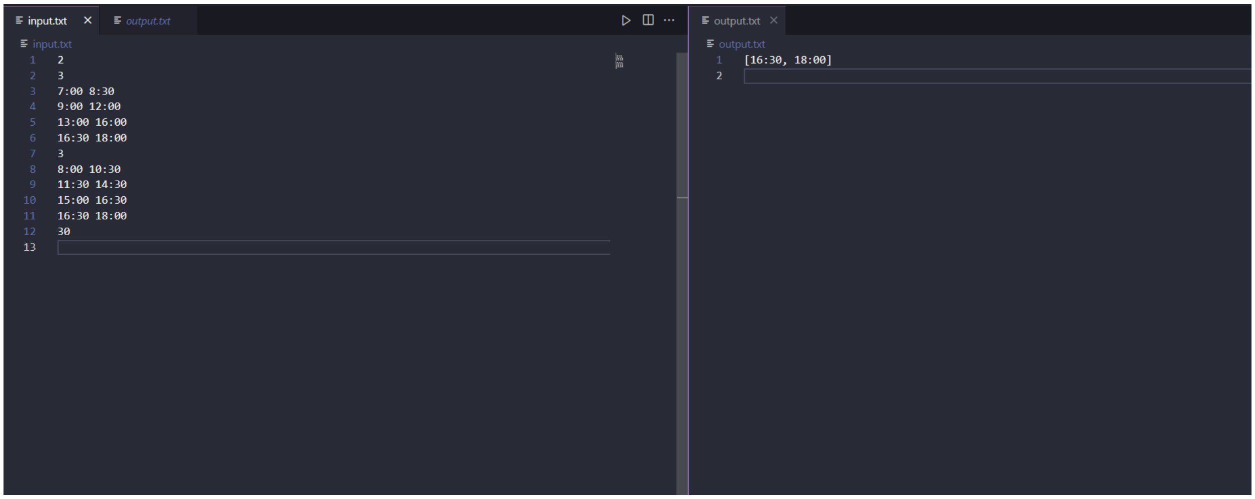
```
[09:00, 17:00]
```

Case 7: One Employee Works Night Shift



```
2
2
22:00 23:30
15:00 17:00
21:00 23:59
2
8:30 10:00
13:30 16:00
9:00 18:00
30
```

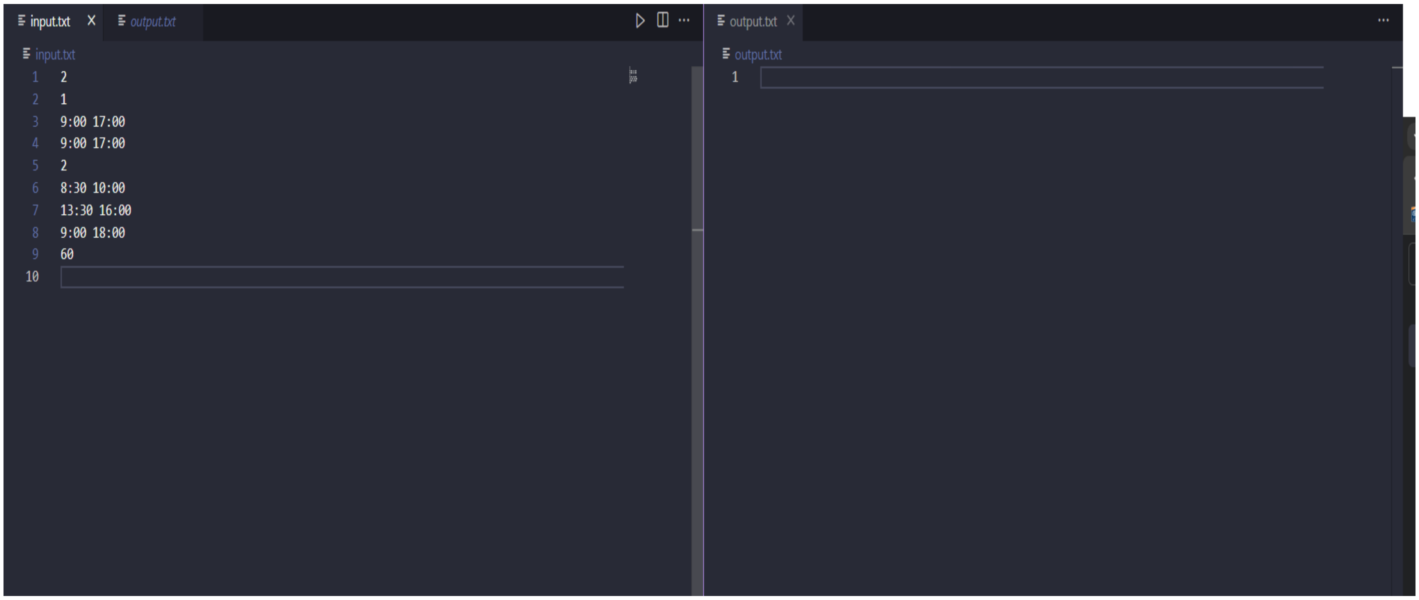
Case 8: Both Employees Free Only in the Evening



```
2
3
7:00 8:30
9:00 12:00
13:00 16:00
16:30 18:00
3
8:00 10:30
11:30 14:30
15:00 16:30
16:30 18:00
30
```

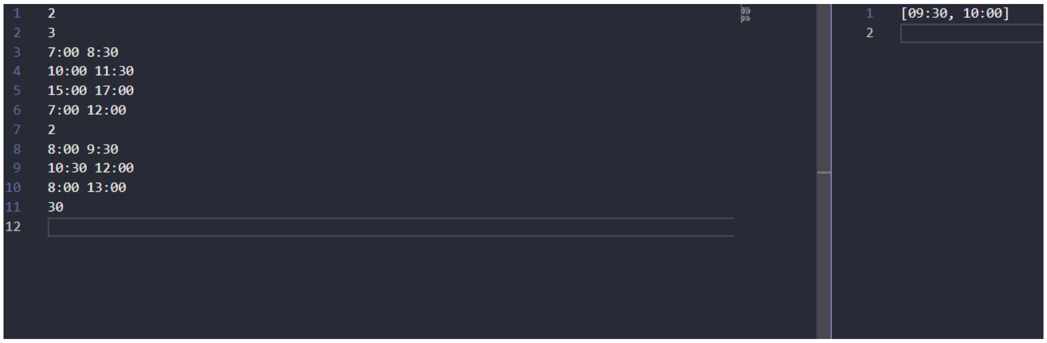
```
[16:30, 18:00]
```

Case 9: One Employee is Always Busy



```
2
1
9:00 17:00
9:00 17:00
2
8:30 10:00
13:30 16:00
9:00 18:00
60
```

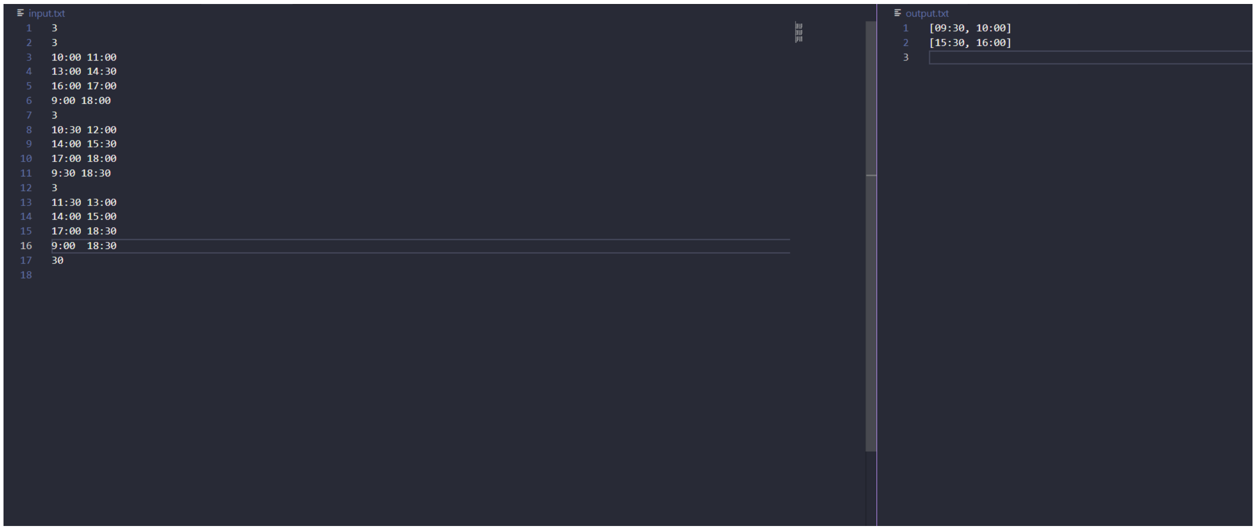
Case 10: Only Morning Hours Overlap



```
2
3
7:00 8:30
10:00 11:30
15:00 17:00
7:00 12:00
2
8:00 9:30
10:30 12:00
8:00 13:00
30
```

```
[09:30, 10:00]
```

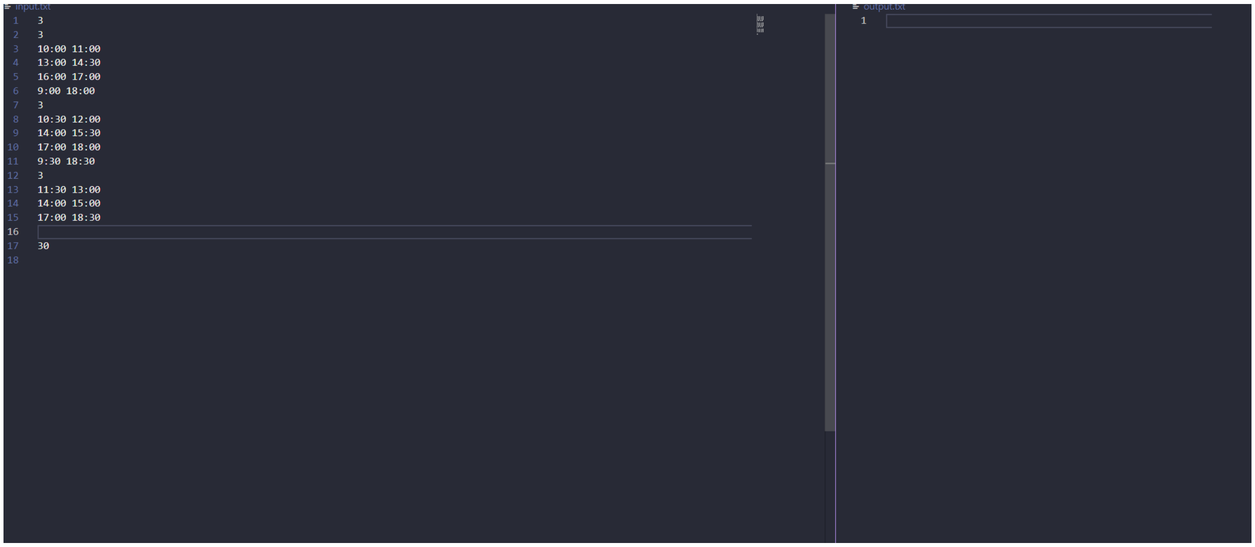
Case 11: three workers



```
3
3
10:00 11:00
13:00 14:30
16:00 17:00
9:00 18:00
3
10:30 12:00
14:00 15:30
17:00 18:00
9:30 18:30
3
11:30 13:00
14:00 15:00
17:00 18:30
9:00  18:30
30
```

```
[09:30, 10:00]
[15:30, 16:00]
```

Case 12 : three workers but worker 3 has no hours.



```
3
3
10:00 11:00
13:00 14:30
16:00 17:00
9:00 18:00
3
10:30 12:00
14:00 15:30
17:00 18:00
9:30 18:30
3
11:30 13:00
14:00 15:00
17:00 18:30

30
```

Analysis

Functions

1. stringToMinutes
2. minutesToTime

- These are both O(1), they just perform conversions and do not rely on input size.
    - Step counts for string is

3. matchingGroupSched
    - We Begin with initializing Start_time and end_time which is o(1) (both are 1 step = 2 steps total)
    - We begin with outer loop, it iterates through each employee which would ne o(n) which is n steps
        - This has an inner loop of o(m) which is n steps that iterates through busy slots
    - Initializes busy_start as well as busy_end and updatedAvailableSlots which is O(1) each takes 1 step so 3 steps total.
    - Initializing updatedAvailableSlots: 1 step
    - Initializing available_start which is 1 step and avaialble_end which as 1 step making a total of 2 steps
    - Another nested loop which now iterates through the available times as well as will be denoted as o(k) this may also be o(m) because the available times may match the busy times, one may be more or less so we will take the worst case.
    - Steps for updatedAvailableslots and conditional checks result to 5 steps
    - Assigning everything else in the rest of the function each take one step which does not affect time complexity.
4. Main Function

    a. There are no loops that in the main function that affect time complexity,

    i. There are multiple steps that read and write

    Reading numEmployees 1 step

    Loop to read employee data n steps

    Loop to read busy slots m steps

Reading start 1 step

Reading end 1 step

Adding to schedule 1 step

Reading working start 1 step

Reading workingEnd 2 steps

Assigning to working_period 1 step

- ★ SUMMARY There are a total of three loops, One for employees, one for busy schedules, and lastly one for available time denoted by O(n),O(m), andO(k) respectively.
  - Combining these together we get the time complexity of O(n * m * k) and taking in consideration of worst case the final time complexity would be denoted as O(n* m^2)

Can we do better? What changes do you think can be made to your algorithm to increase its time complexity/efficiency? Will an increase in n change the complexity class? n is the number of persons in the group

Our project currently takes into account an increase of n, so no an increase in n will NOT change the complexity class.