# Mid-Term Project: Implementing Algorithm

Fall 2023 CPSC 335.07 - Algorithm Engineering
Instructor: Rakesh Singh


## Abstract

Develop an algorithm for the problem, analyze its time and space complexity, implement the code for the algorithm of your choice, test your implementation, and describe your results.


## The Problem: Matching Group Schedules

The group schedule matching takes two or more arrays as input. The arrays represent slots that are already booked and the login/logout time of group members. It outputs an array containing intervals of time when all members are available for a meeting for a minimum duration expected.

Mathematical notation of the problem.

*Hint: you will be good even if you don't understand the mathematical notations of the problem.*

| *Group Schedule Problem* |
|---|
| **input:** arrays $m$ of related elements comprising the time intervals and an array $d$, representing the daily active periods of all members. U is a global set of all arrays. The problem can be represented as: $$U = \sum_{i=1}^{n} m_i$$ **output:** a set of an array, $r$, such that $r \subseteq U$ |

The group schedule matching takes the following inputs:

1. **Busy_Schedule:** An array list that represents the person's existing schedule (they can't plan any other engagement during these hours)
   *Hint: Array may be 2D or maybe a list, ArrayList. It's up to you how you want to implement the input.*
2. **Working_period**: Daily working periods of group members. (login, logout)
   *Just two entries [login, logout]*
3. **Minimum Duration: D (in minutes):** It outputs a list containing intervals of time when all members are available for a meeting for the minimum duration of the meeting required.

**An analogy for the question:**

Assume you and your group members provide your schedules and daily availability. The goal is to find a time slot when all of you are free for a meeting, considering the provided schedules and the minimum duration required for the meeting.

**Sample Input**

Enter person1_Schedule =[[ '7:00', '8:30'], ['12:00', '13:00'], ['16:00', '18:00']]
person1_DailyAct = ['9:00', '19:00']

Enter person2_Schedule = [[ ‘9:00’, ’10:30’], [’12:20’, ’13:30’], [’14:00’, ’15:00’], [’16:00’, ’17:00’ ]]
person2_DailyAct = [‘9:00’, ’18: 30’]

Enter duration_of_meeting =30

**Sample output**
[[’10:30’, ’12:00’], [‘13:30’, ’14:00’] [’15:00’, ’16:00’], [’18:00’, ’18:30’]]

*Note: The goal is to find time slots when all attendees are free for a meeting for a minimum of D minutes.*

## Implementation

Have following files

1. “project1_starter” that defines functions for the algorithm described above. You will need to develop and write the functions. Describe how to run your program in the ReadMe file
2. “Input.txt” contains the sample input test cases. Use these sample cases to run your program to see whether your algorithm implementations work correctly. Have a new line character separating the sample test cases (10). **At least 2 must be edge cases.**
3. “Output.txt” – load the sample test case result to output.txt.

## To Do

1. Create a Readme file and include your name(s) and email address(es). The Readme file should also contain instructions on how to run your program.
2. Study the sample input and output above. Write your own complete and clear code for an algorithm to solve this problem.
3. Analyze your code for the algorithm and its big-O efficiency class for time and space.
4. Implement your algorithm using either Python or C++.
5. Run your code using different data inputs

Finally, produce a brief written project report ***in PDF format***. Your report should include the following:

1. Your names, CSUF email address(es), and an indication that the submission is for project 1.
2. A screenshot showing the output of your code for a minimum of 10 test cases defined by yourself. **At least 2 must be edge cases.**
   *Note: First, write edge cases and give a* ***heading****.*
3. A brief proof argument for the **time and space** complexity of your algorithm and its big-O efficiency class.

## Grading Rubric

The suggested grading rubric is below.

1. Algorithm design and implementation = 55 points, divided as follows:
    a. Clear and complete code (full points for Optimal solution, -15 for brute force solution) = **30** points
    b. Complete and clear README.md file = **5** points
    c. Successful compilation = **10** points
    d. Produces accurate result = **10** points
2. Analysis = 45 points, divided as follows
    a. Report document = **30** points
    b. Correct input cases, including correct edge cases = **10** points
    c. Comments on possible improvements or ambiguous code = **5** points

**NOTE: Ensure your submissions are your own works. Your submissions may be checked for similarities using software.**

## Submitting your code

Submit your project as a zip folder with the following format <team_member1_member2>.zip to the Project 1 link on Canvas. It allows for multiple submissions.
Include the following files in the zip folder:

- Readme
- Input.txt
- Project1_starter.py or project1_starter.cpp
- Output.txt
- Report.pdf

## Deadline

The project deadline is Friday, October 27, 11:59 p.m. on Canvas.

Godspeed…!!!
You got this.