Project 1

Jake Harvanchik
harvanchik@csu.fullerton.edu

https://github.com/harvanchik/335-project-1

After analyzing my own code for the project, it's clear that the algorithm's efficiency class is O(n^2), where n represents the number of meetings or schedule intervals for each person in the group. Here's a breakdown of my analysis:

The ***updateSchedule*** function copies the person's schedule and daily activities, both of which have an O(n) complexity since they iterate over the schedule once. So, the total complexity here is O(n + n) = O(n).

The ***mergeSchedules*** function combines the schedules of two persons. It iterates through both schedules once, resulting in a nested loop structure. Hence, the complexity of this function is O(n^2) in the worst case.

The ***sortedSchedules*** function identifies possible availabilities in the merged schedule by iterating over it once, giving it a complexity of O(n).

The ***matchedAvailabilities*** function filters availabilities based on the provided duration, iterating over the list of possible availabilities once, resulting in a complexity of O(n).

Overall, the most significant factor contributing to the algorithm's time complexity is the ***mergedSchedules*** function with its O(n^2) complexity. This means that the overall time complexity of the algorithm is O(n^2).

To improve efficiency, I could optimize the ***mergedSchedules*** function. One approach would be to sort both people's schedules based on their start times, reducing the complexity to O(n * log(n)). However, it's important to note that this would require additional memory for sorting. Importantly, increasing the value of n wouldn't change the complexity class in this case.

In conclusion, my current algorithm has a time complexity of O(n^2). While I recognize the potential for optimization by sorting schedules, I also see that this would introduce some trade-offs.