Project report

Algorithm 3

Group 1

1. Name

1) Chanran Kim (shining04@csu.fullerton.edu)
2) Vinh Nguyen (vinhgod123@csu.fullerton.edu)
3) Brian Alvarez (briandalvarez@csu.fullerton.edu)
1) Christopher Contreras (CContreras71@csu.fullerton.edu)

2. Pseudocode

Problem: Finding the right time when all the group members are free to meet up for a meeting.

Inputs:

- An array of integers Busy_Schedule
- An array of integers Working_period
- An array of integers Duration

Outputs: A list of lists of the best times for all group members to attend a meeting.

Constraints and Assumptions:

- The time has to be in military 24 hour times
- Meeting should fits within all member's active period
- Output time intervals should be in ascending order

Psecudocode:

```
Function GroupScheduleMatching(busy_schedules, working_periods, duration):
    merged_busy_intervals = MergeBusySchedules(busy_schedules,
working_periods)
    free_intervals = FindFreeIntervals(merged_busy_intervals, working_periods)
    available_intervals = FilterByDuration(free_intervals, duration)
    return available_intervals
```

```
Function MergeBusySchedules(busy_schedules, working_periods):
    all_intervals = []
    for i = 0 to len(busy_schedules):
        all_intervals += busy_schedules[i]
        all_intervals.append([working_periods[i][0], working_periods[i][1]])

    all_intervals.sort(by start_time)
    merged_intervals = [all_intervals[0]]

    for i = 1 to len(all_intervals):
        last_interval = merged_intervals[-1]
        current_interval = all_intervals[i]
        if current_interval[0] <= last_interval[1]:
            last_interval[1] = max(last_interval[1], current_interval[1])
        else:
            merged_intervals.append(current_interval)

    return merged_intervals


Function FindFreeIntervals(merged_intervals, working_periods):
    free_intervals = []
    earliest = max([wp[0] for wp in working_periods])
    latest = min([wp[1] for wp in working_periods])
    start_time = earliest

    for interval in merged_intervals:
        if interval[0] > start_time:
            free_intervals.append([start_time, interval[0]])
        start_time = interval[1]

    if start_time < latest:
        free_intervals.append([start_time, latest])

    return free_intervals


Function FilterByDuration(free_intervals,):
    return [interval for interval in free_intervals if (interval[1] - interval[0]) >= duration]
```

3. Proving efficiency of the pseudocode

Time Complexity:

1. Components: Call three functions: MergeBusySchedules, FindFreeIntervals, and FilterByDuration. The call functions will determine their complexity.
2. Combining intervals: loop through n schedules to all_intervals O(m). Sort all)intervals O(m log m). merging intervals: iterate through sorted all_intervals O(m)
3. total time complexity of MergeBusySchedules: O(m) + O(m log m) + O(m) = O(m log m)
4. merged_intervals is M. scan through n working periods O(n). literate through merged intervals time: O(m)
5. Total time complexity of FindFreeIntervals: O(n) + O(m) = O(n+m)
6. FilterByDuration: iterate through free_intervals to filter by duration time O(k) K is the number of intervals
7. Total time complexity of FilerByDuration: O(m)
8. Final Overall Time Complexity: O(m log m) + O(n + m) + O(m) = O(m log m + n)

Space Complexity:

1. GroupScheduleMatching: space for all intervals, MergeBusySchedules: all_intervals and merged_intervals are O(m). FindFreeIntervals: space for free_intervals to O(m), FilterByDuration: space for filtered list O(m)
2. Total Space Complexity: O(m)