Curtis Quan-Tran
Student Email: cquantran@csu.fullerton.edu

Pseudocode

```
Function groupScheduler(mySchedule, myDailyAct, otherSchedules, otherDailyActs, duration) {
   // Combine all busy times from everyone's schedules
   combinedBusyTimes = []

   for each time slot in mySchedule:
      add time slot to combinedBusyTimes

   for each schedule in otherSchedules:
      for each time slot in schedule:
         add time slot to combinedBusyTimes

   // Sort combinedBusyTimes
   sort combinedBusyTimes by start time

   // Merge overlapping busy times
   mergedBusyTimes = []
   add combinedBusyTimes[0] to mergedBusyTimes

   for i from 1 to size of combinedBusyTimes:
      // Check if mergedBusyTimes' end time is earlier than combinedBusyTimes' Start time
      if combinedBusyTimes[i] overlaps with the last time in mergedBusyTimes:
         //  merge by replacing the mergedBusyTime's end time with the latest max end time.
         merge them (update end time) using the max of the two end times.
      else:
         add combinedBusyTimes[i] to mergedBusyTimes

   // Find the earliest and latest working hours for everyone
   earliestStart = later start between myDailyAct and otherDailyActs[0]'s start time
   latestEnd = earlier end between myDailyAct and otherDailyActs[0]'s end time

   // Find free times between merged busy times
   availableTimes = []
   previousEnd = earliestStart

   for each busy time in mergedBusyTimes:
      if there is free time between previousEnd and this busy time:
         if free time is long enough (greater than or equal to duration):
            add that free time to availableTimes
      update previousEnd to the end of this busy time

   //  Check if there is free time after the last busy period
   if there is enough free time between the end of the last busy time and latestEnd:
      add that free time to availableTimes

   return availableTimes }
```

Proving the Efficiency of Pseudocode Using Step Count

Step 1: Combining all busy times

```
combinedBusyTimes = []                                          // 1

for each time slot in mySchedule:                               // n + 1
    add time slot to combinedBusyTimes                          // n

for each schedule in otherSchedules:                            // s + 1
    for each time slot in schedule:                             // t + 1
        add time slot to combinedBusyTimes                      // t
```

The nested loop iterates over differently sized vectors, therefore it's not going to have n^2 operations.

f(n) = 1 + ((n + 1) + n) + ((s + 1) + s(2t + 1))
f(n) = 1 + (2n + 1) + (2s+2st + 1)
f(n) = (n + s + st) **=>** **Worst Case Time Complexity is O(n + s + st)**

Step 2: Sorting CombinedBusyTimes

Calling std::sort for the *combinedBusyTimes* vector has a **worst case time complexity of O(n log n).**

Step 3: Merge overlapping busy times

```
mergedBusyTimes = []                                                        // 1
add combinedBusyTimes[0] to mergedBusyTimes                                 // 1

for i from 1 to size of combinedBusyTimes:                                  // (n - 1) + 1
    if combinedBusyTimes[i] overlaps with the last time in mergedBusyTimes: // 1
        merge them (update end time) using the max of the two end times.    // 1
    else:
        add combinedBusyTimes[i] to mergedBusyTimes                         // n
```

f(n) = (((n-1) + 1) + 1 + 1 + n) + 1 + 1 = 2n + 1 + 1 + 1
f(n) = 2n+3 **=>** **Worst Case Time Complexity is O(n)**

Step 4: Find the earliest and latest working hours for everyone

```
earliestStart = later start between myDailyAct and otherDailyActs[0]'s start time   // 1
latestEnd = earlier end between myDailyAct and otherDailyActs[0]'s end time         // 1
```

Calling std::max and std::min would have the **worst time complexity of O(1)** because we are just comparing two values.

f(n) = 1 + 1
f(n) = 2 **=>** **Worst Case Time Complexity is O(1)**

Step 5: Find free times between merged busy times

```
availableTimes = []                                                          // 1
previousEnd = earliestStart                                                  // 1

for each busy time in mergedBusyTimes:                                       // n + 1
    if there is free time between previousEnd and this busy time:            // 1
        if free time is long enough (greater than or equal to duration):     // 1
            add that free time to availableTimes                             // n
    update previousEnd to the end of this busy time                          // n

// Check if there is free time after the last busy period
if there is enough free time between the end of the last busy time and latestEnd:   // 1
    add that free time to availableTimes                                     // n

return availableTimes                                                        // 1
```

f(n) = 1 + 1 + ((n + 1) + 1 + 1 + n + n) + 1 + n + 1
f(n) = 2 + (3n + 3) + 1 + n + 1
f(n) = 4n + 7 **=>** **Worst Case Time Complexity is O(n)**

Overall Time Complexity of the groupScheduler Function
f(n) = O(n + s + st) + O(n log n) + O(n) + O(1) + O(n)

Thus, the **overall worst case time complexity is O(n log n)** for the groupScheduler function.