Project 1

Group 9:

Xander Lahti (xanderla@csu.fullerton.edu)
Xareni Merino Rita (xmerino@csu.fullerton.edu)
T e Yen Lee (leeteyen91@csu.fullerton.edu)
Juan Martinez Vasquez (diegomarvas@csu.fullerton.edu)

Pseudocode for Algorithm 3:

```
Function convertToMilitaryTime(time string)
 {
        hours = first 2 characters (time string) as int
         minutes =last 2 characters(time string) as int

         Return (hours * 60 + minutes)
}
```

```
Function convertToMilitaryTime(military time int)
{
        Hour = totalmin /60
        Minutes = totalmin % 60

                If hours < 10
                {
                        hour (String)= "0" + hours
                }
                Else
                {
                        hour (String) = hours
                  }
                 if minutes < 10
                {
                         minute (String) = "0" + minutes
                }
```

```
        Else
        {
                minute (String) = minutes
        }

    Return hour (String) + ":" + minute (String)
}
```

```
Function combineBusySchedules(firstSchedule, secondSchedule)
{
     combinedSchedules = firstSchedule + secondSchedule
    Sort combinedSchedules ( start times )
    finalMerged = EMPTY

     For ( n currently in combinedSchedules)
    {
          If (finalMerged  EMPTY OR lastElement(finalMerged).end < current.start)
        {
                finalMerged += combinedSchedules[n];
        }
        Else
        {
         lastElement(finalMerged).end = MAX(lastElement(finalMerged).end, current.end)
        }
    }

    Return finalMerged;
}
```

```
Function getAvailableSlots(mergedSchedule, dailyAvailability, meetingDuration)
{
 availableSlots = EMPTY
 startOfDay = dailyAvailability.start
 endOfDay = dailyAvailability.end

    If (mergedSchedule  !EMPTY AND
    (mergedSchedule[0].start - startOfDay >= meetingDuration))
    {

        (startOfDay, mergedSchedule[0].start) += availableSlots
    }

    //Check for gaps between meetings
```

```
        For( ii FROM 1 TO LENGTH(mergedSchedule) - 1 )
        {
                If (mergedSchedule[i].start - mergedSchedule[i - 1].end >= meetingDuration)
                {
                        (mergedSchedule[i - 1].end, mergedSchedule[i].start) += availableSlots
                }
        }
         // Check for gaps after the last meeting
         If ( mergedSchedule  !EMPTY AND
        (endOfDay - mergedSchedule[LAST_INDEX].end >= meetingDuration) )
        {
                 (mergedSchedule[LAST_INDEX].end, endOfDay) += availableSlots
        }

        // Checking if they fall within daily availability
        finalAvailableSlots = EMPTY
         For each slot in availableSlots
        {
                If( slot.start >= startOfDay AND slot.end <= endOfDay)
                        slot += finalAvailableSlots
        }
                Return finalAvailableSlots
}
```

```
Function findAvailableMeetingTimes(firstSchedule, secondSchedule, dailyAvailability1,
dailyAvailability2, meetingDuration)
{
 // Convert times in schedules to military time
 militarySchedule1 = EMPTY
        For each time in firstSchedule
        {
                start = convertToMilitaryTime(time.start)
                 end = convertToMilitaryTime(time.end)
                  (start, end) += militarySchedule1
        }
 militarySchedule2 = EMPTY
        For each time in secondSchedule
        {
                start = convertToMilitaryTime(time.start)
                 end = convertToMilitaryTime(time.end)
                (start, end) += militarySchedule2
        }
 // Combine the two schedules
```

```
 combinedSchedule = combineBusySchedules(militarySchedule1, militarySchedule2)

//Finding overlapping daily availabilities of the two people

 startAvailability = MAX(convertToMilitaryTime(dailyAvailability1.start),
                         convertToMilitaryTime(dailyAvailability2.start))
endAvailability = MIN(convertToMilitaryTime(dailyAvailability1.end),
                         convertToMilitaryTime(dailyAvailability2.end))

 // Get available slots using start and end overlapping availabilities
availableSlots = getAvailableSlots(combinedSchedule, (startAvailability, endAvailability),
meetingDuration)

// Convert available slots back to string format for output
resultSlots = EMPTY
        For (each slot in availableSlots )
        {
                startString = convertTimeToString(slot.start) endString =
        convertTimeToString(slot.end)
                 (startString, endString) += resultSlots
        }

Return resultSlots;
```

Proven Efficiency of the pseudocode “Step Counts”

Input size: 1

```
Function convertToMilitaryTime(time string)
 {
        hours = first 2 characters (time string) as int
         minutes =last 2 characters(time string) as int

         Return (hours * 60 + minutes)
}
```

Complexity: O(1)

Input Size: 1

```
Function convertToMilitaryTime(military time int)
{
        Hour = totalmin /60
        Minutes = totalmin % 60

                If hours < 10
                {
                        hour (String)= "0" + hours
                }
                Else
                {
                        hour (String) = hours
                  }
                 if minutes < 10
                {
                         minute (String) = "0" + minutes
                }
                Else
                {
                        minute (String) = minutes
                }

        Return hour (String) + ":" + minute (String)
}
```

Complexity O(1)

Input size : (n+m)

```
Function combineBusySchedules(firstSchedule, secondSchedule)
{
        combinedSchedules = firstSchedule + secondSchedule O(n+m)
        Sort combinedSchedules ( start times ) O(n+m) log(n+m)
        finalMerged = EMPTY

        For ( n currently in combinedSchedules) O(n+m)
        {
                If (finalMerged  EMPTY OR lastElement(finalMerged).end < current.start)
                {
                        finalMerged += combinedSchedules[n];
                }
                Else
                {
                lastElement(finalMerged).end = MAX(lastElement(finalMerged).end, current.end)
                }
        }

        Return finalMerged;
}
```

Complexity: O(n+m)log(n+m)

Input Size: k (k <=n+m)

```
Function getAvailableSlots(mergedSchedule, dailyAvailability, meetingDuration)
{
 availableSlots = EMPTY
 startOfDay = dailyAvailability.start
 endOfDay = dailyAvailability.end

        If (mergedSchedule  !EMPTY AND O(1)
        (mergedSchedule[0].start - startOfDay >= meetingDuration))
        {

                (startOfDay, mergedSchedule[0].start) += availableSlots
        }

        //Check for gaps between meetings
        For( ii FROM 1 TO LENGTH(mergedSchedule) - 1 ) O(k)
        {
                If (mergedSchedule[i].start - mergedSchedule[i - 1].end >= meetingDuration)
                {
                        (mergedSchedule[i - 1].end, mergedSchedule[i].start) += availableSlots
```

```
                }
        }
         // Check for gaps after the last meeting
         If ( mergedSchedule  !EMPTY AND O(1)
        (endOfDay - mergedSchedule[LAST_INDEX].end >= meetingDuration) )
        {
                 (mergedSchedule[LAST_INDEX].end, endOfDay) += availableSlots
        }

        // Checking if they fall within daily availability
        finalAvailableSlots = EMPTY
         For each slot in availableSlots O(m)
        {
                If( slot.start >= startOfDay AND slot.end <= endOfDay)
                        slot += finalAvailableSlots
        }
                Return finalAvailableSlots
}
```
Complexity: O(k+m)

Function findAvailableMeetingTimes:

Convert schedules to int (military time) function: O(n)
Convert schedules to int (military time) function: O(m)
Combine Schedules function: O(n+m) log(n+m)
Get Available Slots function: O(k)

O(n+m+(n+m)log(n+m)+m+k)

Overal Time Complexity : O(n+m)log(n+m)