Solution 1

```swift
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

class Program {
    // O(c1 + c2) time | O(c1 + c2) space
    func calendarMatching(_ calendar1: [[String]], _ dailyBounds1: [String], _ calendar2: [[String]], _ dailyBounds2: [String], _ meetingDuration: Int) -> [[String]] {
        let updatedCalendar1 = updateCalendar(calendar1, dailyBounds1)
        let updatedCalendar2 = updateCalendar(calendar2, dailyBounds2)

        let mergedCalendar = mergeCalendars(updatedCalendar1, updatedCalendar2)
        let flattenedCalendar = flattenCalendar(mergedCalendar)

        return getMatchingAvailabilities(flattenedCalendar, meetingDuration)
    }

    func updateCalendar(_ calendar: [[String]], _ dailyBounds: [String]) -> [[Int]] {
        let lowerBound = ["0:00", dailyBounds[0]]
        let upperBound = [dailyBounds[1], "23:59"]
        var updatedCalendar = [[String]]()

        updatedCalendar.append(lowerBound)
        updatedCalendar.append(contentsOf: calendar)
        updatedCalendar.append(upperBound)

        return updatedCalendar.map { $0.map { timeToMinutes($0) } }
    }

    func mergeCalendars(_ calendar1: [[Int]], _ calendar2: [[Int]]) -> [[Int]] {
        var i = 0
        var j = 0
        var merged = [[Int]]()

        while i < calendar1.count, j < calendar2.count {
            let meeting1 = calendar1[i]
            let meeting2 = calendar2[j]

            if meeting1[0] < meeting2[0] {
                merged.append(meeting1)
                i += 1
            } else {
                merged.append(meeting2)
                j += 1
            }
        }

        while i < calendar1.count {
            merged.append(calendar1[i])
            i += 1
        }

        while j < calendar2.count {
            merged.append(calendar2[j])
            j += 1
        }

        return merged
    }

    func flattenCalendar(_ calendar: [[Int]]) -> [[Int]] {
        let firstEntry = calendar[0]
        var flattened = [[Int]]()
        flattened.append(firstEntry)

        for currentMeeting in calendar {
            if let previousMeeting = flattened.last, let currentStart = currentMeeting.first, let currentEnd = currentMeeting.last, let previousStart = previousMeeting.first, let previousE
                if previousEnd >= currentStart {
                    let newPreviousMeeting = [previousStart, max(previousEnd, currentEnd)]
                    flattened[flattened.count - 1] = newPreviousMeeting
                } else {
                    flattened.append(currentMeeting)
                }
            }
        }

        return flattened
    }

    func getMatchingAvailabilities(_ calendar: [[Int]], _ meetingDuration: Int) -> [[String]] {
        var matchingAvailabilities = [[Int]]()

        for i in 1 ..< calendar.count {
            let start = calendar[i - 1][1]
            let end = calendar[i][0]

            let availabilityDuration = end - start
            if availabilityDuration >= meetingDuration {
                matchingAvailabilities.append([start, end])
            }
        }

        return matchingAvailabilities.map { $0.map { minutesToTime($0) } }
    }

    func timeToMinutes(_ string: String) -> Int {
        let separatedComponents = string.split(separator: ":").map { Int($0) }

        if let hours = separatedComponents[0], let minutes = separatedComponents[1] {
            return (hours * 60) + minutes
        }

        return 0
    }

    func minutesToTime(_ minutes: Int) -> String {
        var hours = (Double(minutes) / 60)
        hours = hours.rounded(.down)

        let mins = minutes % 60

        let hoursString = "\(Int(hours))"
        let minsString = mins < 10 ? "0" + "\(mins)" : "\(mins)"

        return hoursString + ":" + minsString
    }
}

```