Solution 1

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

#include <cmath>
#include <vector>

using namespace std;

struct StringMeeting {
  string start;
  string end;
};

struct Meeting {
  int start;
  int end;
};

vector<Meeting> updateCalendar(vector<StringMeeting> calendar,
                               StringMeeting dailyBounds);
vector<Meeting> mergeCalendars(vector<Meeting> calendar1,
                               vector<Meeting> calendar2);
vector<Meeting> flattenCalendar(vector<Meeting> calendar);
vector<StringMeeting> getMatchingAvailabilities(vector<Meeting> calendar,
                                                int meetingDuration);
int timeToMinutes(string time);
string minutesToTime(int minutes);

// O(c1 + c2) time | O(c1 + c2) space - where c1 and c2 are the respective
// numbers of meetings in calendar1 and calendar2
vector<StringMeeting> calendarMatching(vector<StringMeeting> calendar1,
                                       StringMeeting dailyBounds1,
                                       vector<StringMeeting> calendar2,
                                       StringMeeting dailyBounds2,
                                       int meetingDuration) {
  vector<Meeting> updatedCalendar1 = updateCalendar(calendar1, dailyBounds1);
  vector<Meeting> updatedCalendar2 = updateCalendar(calendar2, dailyBounds2);
  vector<Meeting> mergedCalendar =
      mergeCalendars(updatedCalendar1, updatedCalendar2);
  vector<Meeting> flattenedCalendar = flattenCalendar(mergedCalendar);
  return getMatchingAvailabilities(flattenedCalendar, meetingDuration);
}

vector<Meeting> updateCalendar(vector<StringMeeting> calendar,
                               StringMeeting dailyBounds) {
  vector<StringMeeting> updatedCalendar;
  updatedCalendar.push_back({"0:00", dailyBounds.start});
  updatedCalendar.insert(updatedCalendar.end(), calendar.begin(),
                         calendar.end());
  updatedCalendar.push_back({dailyBounds.end, "23:59"});
  vector<Meeting> calendarInMinutes;
  for (int i = 0; i < updatedCalendar.size(); i++) {
    calendarInMinutes.push_back({timeToMinutes(updatedCalendar[i].start),
                                 timeToMinutes(updatedCalendar[i].end)});
  }
  return calendarInMinutes;
}

vector<Meeting> mergeCalendars(vector<Meeting> calendar1,
                               vector<Meeting> calendar2) {
  vector<Meeting> merged;
  int i = 0;
  int j = 0;
  while (i < calendar1.size() && j < calendar2.size()) {
    Meeting meeting1 = calendar1[i];
    Meeting meeting2 = calendar2[j];
    if (meeting1.start < meeting2.start) {
      merged.push_back(meeting1);
      i++;
    } else {
      merged.push_back(meeting2);
      j++;
    }
  }
  while (i < calendar1.size())
    merged.push_back(calendar1[i++]);
  while (j < calendar2.size())
    merged.push_back(calendar2[j++]);
  return merged;
}

vector<Meeting> flattenCalendar(vector<Meeting> calendar) {
  vector<Meeting> flattened = {calendar[0]};
  for (int i = 1; i < calendar.size(); i++) {
    Meeting currentMeeting = calendar[i];
    Meeting previousMeeting = flattened[flattened.size() - 1];
    if (previousMeeting.end >= currentMeeting.start) {
      Meeting newPreviousMeeting = {
          previousMeeting.start, max(previousMeeting.end, currentMeeting.end)};
      flattened[flattened.size() - 1] = newPreviousMeeting;
    } else {
      flattened.push_back(currentMeeting);
    }
  }
  return flattened;
}

vector<StringMeeting> getMatchingAvailabilities(vector<Meeting> calendar,
                                                int meetingDuration) {
  vector<Meeting> matchingAvailabilities;
  for (int i = 1; i < calendar.size(); i++) {
    int start = calendar[i - 1].end;
    int end = calendar[i].start;
    int availabilityDuration = end - start;
    if (availabilityDuration >= meetingDuration) {
      matchingAvailabilities.push_back({start, end});
    }
  }

  vector<StringMeeting> matchingAvailabilitiesInHours;
  for (int i = 0; i < matchingAvailabilities.size(); i++) {
    matchingAvailabilitiesInHours.push_back(
        {minutesToTime(matchingAvailabilities[i].start),
         minutesToTime(matchingAvailabilities[i].end)});
  }
  return matchingAvailabilitiesInHours;
```

```
}

int timeToMinutes(string time) {
  int delimiterPos = time.find(":");
  int hours = stoi(time.substr(0, delimiterPos));
  int minutes = stoi(time.substr(delimiterPos + 1, time.length()));
  return hours * 60 + minutes;
}

string minutesToTime(int minutes) {
  int hours = minutes / 60;
  int mins = minutes % 60;
  string hoursString = to_string(hours);
  string minutesString = mins < 10 ? "0" + to_string(mins) : to_string(mins);
  return hoursString + ":" + minutesString;
}

```