920. Meeting Rooms

* [Description](http://www.lintcode.com/en/problem/meeting-rooms/#description)
* [Notes](http://www.lintcode.com/en/problem/meeting-rooms/#note)
* [Testcase](http://www.lintcode.com/en/problem/meeting-rooms/#testcase)
* [Judge](http://www.lintcode.com/en/problem/meeting-rooms/#judge)

[Discuss](http://www.lintcode.com/discuss)

Accepted

Total Runtime: 4020 ms

100% test cases passed.

Share your acceptance to your friends!

Take a look at the time distribution of your submission!

[View](http://www.lintcode.com/submission/13904122/)

Wrong code got passed?  
Provide more test data to LintCode!

Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), determine if a person could attend all meetings.

Have you met this question in a real interview?

Yes

**Example**

Given intervals = [[0,30],[5,10],[15,20]], return false.

[http://www.lintcode.com/en/problem/meeting-rooms/#](http://www.lintcode.com/en/problem/meeting-rooms/)

*/\**

*\* To change this license header, choose License Headers in Project Properties.*

*\* To change this template file, choose Tools | Templates*

*\* and open the template in the editor.*

*\*/*

**package** javaapplication24;

**import** java.util.ArrayList;

**import** java.util.List;

***/\*\****

***\****

***\* @author Usuario***

***\*/***

**public** **class** JavaApplication24 {

**public** **static** **class** Interval {

**public** **int** start, end;

**public** Interval(**int** start, **int** end) {

**this**.start = start;

**this**.end = end;

       }

    }

**public** **static** **boolean** canAttendMeetings(List<Interval> intervals) {

*// Write your code here*

**for**(**int** i =0; i<intervals.size()-1; i++)

        {

**for**(**int** j =i+1; j<intervals.size(); j++)

            {

                Interval a = intervals.get(i);

                Interval b = intervals.get(j);

**if**(b.start >a.start && b.start < a.end)

                {

**return** **false**;

                }

            }

        }

**return** **true**;

    }

**public** **static** **void** main(String[] args) {

*// TODO code application logic here*

*// [[0,30],[5,10],[15,20]]*

        List<Interval> inter = **new** ArrayList();

        inter.add(**new** Interval(0,30));

        inter.add(**new** Interval(5,10));

        inter.add(**new** Interval(15,20));

        System.out.println( canAttendMeetings(inter));

    }

}

--------------MI SOLUCION CON QUICK SORT---------------------------------

***/\*\****

***\* Definition of Interval:***

***\* public classs Interval {***

***\*     int start, end;***

***\*     Interval(int start, int end) {***

***\*         this.start = start;***

***\*         this.end = end;***

***\*     }***

***\* }***

***\*/***

**public** **class** Solution {

***/\*\****

***\* @param intervals: an array of meeting time intervals***

***\* @return: if a person could attend all meetings***

***\*/***

**void** quicksort(List<Interval> vector, **int** primero, **int** ultimo)

        {

**int** i, j, central;

**int** pivote;

            central = (primero + ultimo) / 2;

            pivote = vector.get( central).start;

            i = primero;

            j = ultimo;

**do**

            {

**while** (vector.get(i).start < pivote) i++;

**while** (vector.get(j).start > pivote) j--;

**if** (i <= j)

                {

                    Interval temp;

                    temp = vector.get(i);

*//vector[i] = vector[j];*

*//vector[j] = temp;*

                    vector.set(i, vector.get(j));

                    vector.set(j, temp);

                    i++;

                    j--;

                }

            } **while** (i <= j);

**if** (primero < j)

            {

                quicksort(vector, primero, j);

            }

**if** (i < ultimo)

            {

                quicksort(vector, i, ultimo);

            }

        }

**public**  **boolean** canAttendMeetings(List<Interval> intervals) {

**if**(intervals == **null** || intervals.isEmpty()) **return** **true**;

         quicksort(intervals, 0, intervals.size()-1);

**for**(**int** i =1; i<intervals.size(); i++) {

**if**(intervals.get(i).start < intervals.get(i-1).end) {

**return** **false**;

             }

         }

**return** **true**;

     }

}

------------------------SOLUCIONES DE OTROS-------------------

[awangdev](https://github.com/awangdev)/**[LintCode](https://github.com/awangdev/LintCode)**

|  |
| --- |
| E |
|  | 1521097808 |
|  | tags: Sort, Sweep Line |
|  |  |
|  | - 注意接头点要考虑所有开会结会的情况，不要恰巧漏掉相接的点 |
|  | - 开会的是超人。瞬间移动接上下一个会议 |
|  |  |
|  | #### 方法1: |
|  | 找是否有overlap. priorityQueue 按照start time排序好以后, 比较current和peek: current.end > peek.start? |
|  |  |
|  | #### 方法2: Sweep line |
|  | - class Point{pos, flag}, PriorityQueue排序。计算count |
|  | - 跟 Number of Airplanes in the Sky 是一个类型的题目 |
|  |  |
|  |  |
|  |  |
|  | ``` |
|  | /\* |
|  | Given an array of meeting time intervals consisting of start and end times [[s1,e1],[s2,e2],...] (si < ei), |
|  | determine if a person could attend all meetings. |
|  |  |
|  | For example, |
|  | Given [[0, 30],[5, 10],[15, 20]], |
|  | return false. |
|  |  |
|  | Hide Company Tags Facebook |
|  | Hide Tags Sort |
|  | Hide Similar Problems (H) Merge Intervals (M) Meeting Rooms II |
|  |  |
|  | \*/ |
|  |  |
|  | /\*\* |
|  | \* Definition for an interval. |
|  | \* public class Interval { |
|  | \* int start; |
|  | \* int end; |
|  | \* Interval() { start = 0; end = 0; } |
|  | \* Interval(int s, int e) { start = s; end = e; } |
|  | \* } |
|  | \*/ |
|  | /\* |
|  | Thoughts: |
|  | Cannot have overlap -> sort by the interval.start using priority queue |
|  | \*/ |
|  | // Check over lap, sorting by priority queue |
|  | class Solution { |
|  | public boolean canAttendMeetings(Interval[] intervals) { |
|  | if (intervals == null || intervals.length == 0) { |
|  | return true; |
|  | } |
|  | PriorityQueue<Interval> queue = new PriorityQueue<Interval>(new Comparator<Interval>() { |
|  | public int compare(Interval a, Interval b) { |
|  | return a.start - b.start; |
|  | } |
|  | }); |
|  |  |
|  | // Sort |
|  | for (Interval interval: intervals) { |
|  | queue.offer(interval); |
|  | } |
|  |  |
|  | // Compare tail to head |
|  | while (!queue.isEmpty()) { |
|  | Interval head = queue.poll(); |
|  | Interval next = queue.peek(); |
|  | if (next != null && head.end > next.start) { // overlap happens |
|  | return false; |
|  | } |
|  | } |
|  |  |
|  | return true; |
|  | } |
|  | } |
|  |  |
|  | /\*\* |
|  | \* Definition for an interval. |
|  | \* public class Interval { |
|  | \* int start; |
|  | \* int end; |
|  | \* Interval() { start = 0; end = 0; } |
|  | \* Interval(int s, int e) { start = s; end = e; } |
|  | \* } |
|  | \*/ |
|  |  |
|  | /\* |
|  | Thought: |
|  | Use scan line. |
|  | Note: special care for edge point: make sure to process all connecting point before shuouting the result. |
|  | \*/ |
|  |  |
|  | //use scan line |
|  | public class Solution { |
|  | class Point { |
|  | int pos, flag; |
|  | public Point(int pos, int flag) { |
|  | this.pos = pos; |
|  | this.flag = flag; |
|  | } |
|  | } |
|  | public boolean canAttendMeetings(Interval[] intervals) { |
|  | if (intervals == null || intervals.length == 0) { |
|  | return true; |
|  | } |
|  |  |
|  | // Prepare sweep line points |
|  | PriorityQueue<Point> queue = new PriorityQueue<Point>(2, new Comparator<Point>() { |
|  | public int compare(Point p1, Point p2) { |
|  | return p1.pos - p2.pos; |
|  | } |
|  | }); |
|  | for (int i = 0; i < intervals.length; i++) { |
|  | queue.offer(new Point(intervals[i].start, 1)); |
|  | queue.offer(new Point(intervals[i].end, -1)); |
|  | } |
|  |  |
|  | int count = 0; // count how many meeting happening concurrently |
|  | while (!queue.isEmpty()) { |
|  | Point p = queue.poll(); |
|  | count += p.flag; |
|  |  |
|  | // For all point marked on x (overlap with point), check the flag status |
|  | while (!queue.isEmpty() && p.pos == queue.peek().pos) { |
|  | p = queue.poll(); |
|  | count += p.flag; |
|  | } |
|  | if (count > 1) { |
|  | return false; |
|  | } |
|  | } |
|  |  |
|  | return true; |
|  | } |
|  | } |