Minimum Number of Arrows to Burst Balloons

There are a number of spherical balloons spread in two-dimensional space. For each balloon, provided input is the start and end coordinates of the horizontal diameter. Since it's horizontal, y-coordinates don't matter and hence the x-coordinates of start and end of the diameter suffice. Start is always smaller than end. There will be at most 10^4 balloons.

An arrow can be shot up exactly vertically from different points along the x-axis. A balloon with x_{start} and x_{end} bursts by an arrow shot at x if $x_{start} \leq x \leq x_{end}$. There is no limit to the number of arrows that can be shot. An arrow once shot keeps travelling up infinitely. The problem is to find the minimum number of arrows that must be shot to burst all balloons.

Example:

Input:
`[[10,16], [2,8], [1,6], [7,12]]`

Output:
`2`

Explanation:
`One way is to shoot one arrow for example at x = 6 (bursting the balloons [2,8] and [1,6]) and another arrow at x = 11 (bursting the other two balloons).`

Solution 1

First, we sort balloons by increasing points.end (if ends are the same, then by increasing of points.start). Every time arrow shot points.end, say, points[i].second. If next balloon.start <= points[i].second, it is also shot, thus we continue.

```
    int findMinArrowShots(vector<pair<int, int>>& points) {
        int count = 0, arrow = INT_MIN;
        sort(points.begin(), points.end(), mysort);
        for(int i = 0; i<points.size(); i++){
            if(arrow!=INT_MIN && points[i].first<=arrow){continue;} //former arro
w shot points[i]
            arrow = points[i].second; // new arrow shot the end of points[i]
            count++;
        }
        return count;
    }
    static bool mysort(pair<int, int>& a, pair<int, int>& b){
        return a.second==b.second?a.first<b.first:a.second<b.second;
    }
```

written by westwatermelon original link here

Solution 2

```
public int findMinArrowShots(int[][] points) {
 if(points==null || points.length==0 || points[0].length==0) return 0;
 Arrays.sort(points, new Comparator<int[]>() {
  public int compare(int[] a, int[] b) {
   if(a[0]==b[0]) return a[1]-b[1];
   else return a[0]-b[0];
  }
 });

 int minArrows = 1;
 int arrowLimit = points[0][1];
 for(int i=1;i<points.length;i++) {
  int[] baloon = points[i];
  if(baloon[0]<=arrowLimit) {
   arrowLimit=Math.min(arrowLimit, baloon[1]);
  } else {
   minArrows++;
   arrowLimit=baloon[1];
  }
 }
 return minArrows;
}
```

written by fabrizio3 original link here

Solution 3

```
public class Solution {
    public int findMinArrowShots(int[][] points) {
        if(points == null || points.length < 1) return 0;
        Arrays.sort(points, (a, b)->(a[0]-b[0]));
        int result = 1;
        int end = points[0][1];
        for(int i = 1; i < points.length; i ++) {
            if(points[i][0] > end) {
                result ++;
                end = points[i][1];
            } else {
                end = Math.min(end, points[i][1]);
            }
        }
        return result;
    }
}
```

written by yuejn93 original link here

From Leetcoder.