Previous Next

452. Minimum Number of Arrows to Burst Balloons

May 7, 2019 | 10.9K views

Average Rating: 4.47 (15 votes)

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 $x_{start} \le x \le x_{end}$. 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
```

Solution

Approach 1: Greedy.

Greedy algorithms

Greedy problems usually look like "Find minimum number of *something* to do *something*" or "Find maximum number of *something* to fit in *some conditions*", and typically propose an unsorted input.

> The idea of greedy algorithm is to pick the *locally* optimal move at each step, that will lead to the *globally* optimal solution.

The standard solution has $\mathcal{O}(N \log N)$ time complexity and consists of two parts:

- Figure out how to sort the input data ($\mathcal{O}(N \log N)$ time). That could be done directly by a sorting or indirectly by a heap usage. Typically sort is better than the heap usage because of gain in space.
- Parse the sorted input to have a solution ($\mathcal{O}(N)$ time).

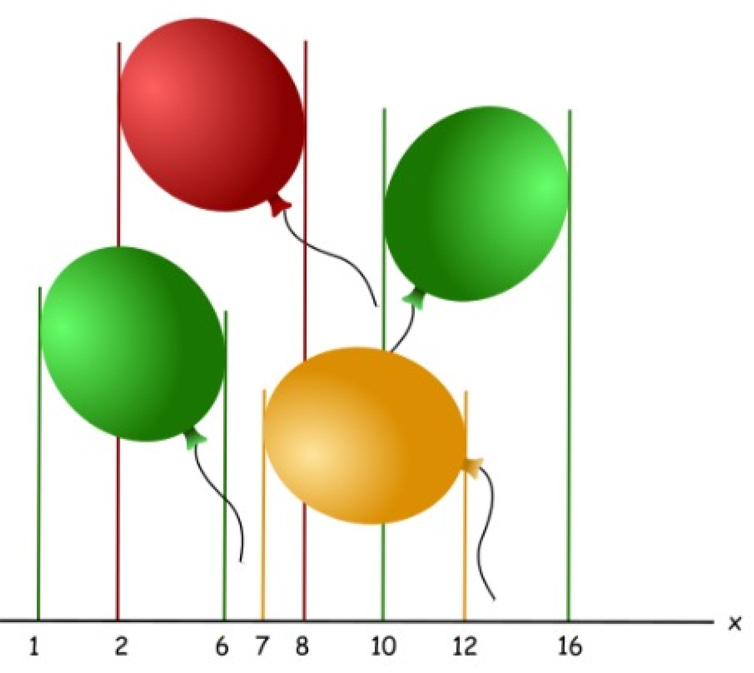
Please notice that in case of well-sorted input one doesn't need the first part and the greedy solution could have $\mathcal{O}(N)$ time complexity, here is an example.

> How to prove that your greedy algorithm provides globally optimal solution?

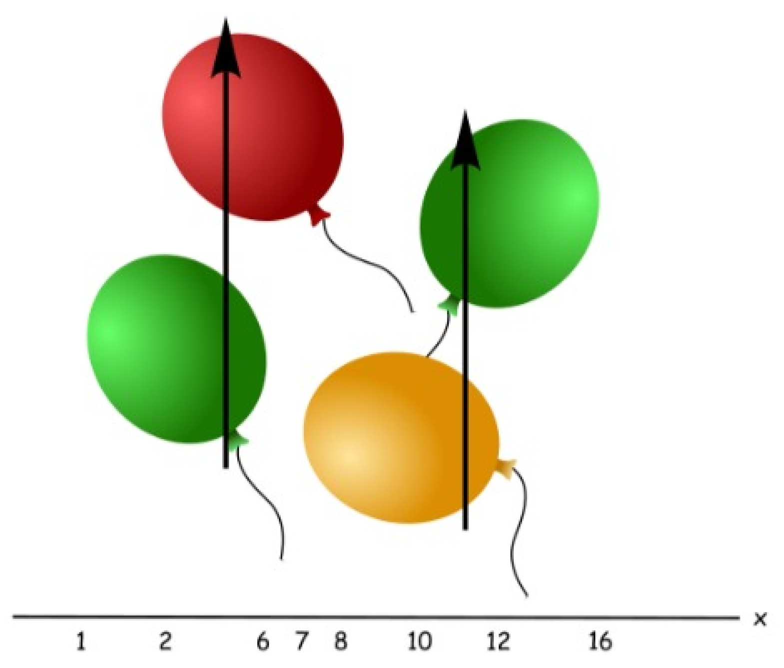
Usually you could use the proof by contradiction.

Intuition

Let's consider the following combinations of the balloons.



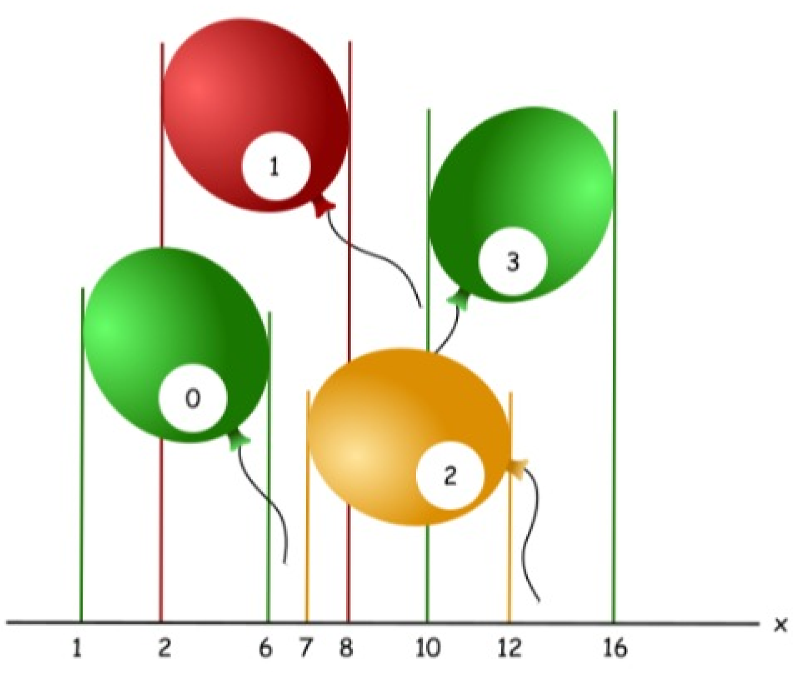
That's quite obvious that two arrows is enough to burst them all, let's figure out how to compute this result with the help of greedy algorithm.



Let's sort the balloons by the end coordinate, and then check them one by one. The first balloon is a green one number `0`, it ends at coordinate `6`, and there is no balloons ending before it because of sorting.

The other balloons have two possibilities :

- To have a start coordinate smaller than `6`, like a red balloon. These ones could be burst together with the balloon `0` by one arrow.
- To have a start coordinate larger than `6`, like a yellow balloon. These ones couldn't be burst together with the balloon `0` by one arrow, and hence one needs to increase the number of arrows here.



> That means that one could always track the end of the current balloon, and ignore all the balloons which end before it. Once the current balloon is ended (= the next balloon starts after the current balloon), one has to increase the number of arrows by one and start to track the end of the next balloon.

Algorithm

Now the algorithm is straightforward :

- Sort the balloons by end coordinate `x_end`.
- Initiate the end coordinate of a balloon which ends first : `first_end = points[0][1]`.
- Initiate number of arrows: `arrows = 1`.
- Iterate over all balloons:
  - If the balloon starts after `first_end`:
    - Increase the number of arrows by one.
    - Set `first_end` to be equal to the end of the current balloon.
- Return arrows.

Implementation

C++ Java Python Copy

```python
class Solution:
    def findMinArrowShots(self, points: List[List[int]]) -> int:
        if not points:
            return 0

        # sort by x_end
        points.sort(key = lambda x : x[1])

        arrows = 1
        first_end = points[0][1]
        for x_start, x_end in points:
            # if the current balloon starts after the end of another one,
            # one needs one more arrow
            if first_end < x_start:
                arrows += 1
                first_end = x_end

        return arrows
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N \log N)$ because of sorting of input data.
- Space complexity : $\mathcal{O}(1)$ since it's a constant space solution.

Rate this article:

Previous Next

Comments: 7 Sort By

Type comment here... (Markdown is supported)

Preview Post

antonio2 7 May 8, 2019 5:02 AM

Apologizes, but this question is not targeted at the balloon scenario specifically... I am still working on learning how to develop proofs for greedy algorithms myself, but you mention:

> Usually you could use the proof by contradiction.

Read More

6 Share Reply

SHOW 4 REPLIES

piyush121 184 May 28, 2020 8:23 AM

We can sort it from start position as well:

```
class Solution {
    public int findMinArrowShots(int[][] points) {
        if(points.length == 0) return 0;
```

Read More

3 Share Reply

LeoShi817 44 April 9, 2020 6:14 AM

This is my attempt to solve this question. The idea is same.

```
    public int findMinArrowShots(int[][] points) {
        if (points.length == 0) {
            return 0;
```

Read More

1 Share Reply

samwisegamgee 133 January 26, 2020 7:14 AM

The explanation is not complete. It doesn't explain why you have to do the last line - 'Set first_end to be equal to the end of the current balloon.'

Consider this example: [1,10],[2,3],[8,9],[7,8]. I know the algorithm will work. But why??

1 Share Reply

eefiasfira 597 November 27, 2019 4:55 AM

Refactored given solution to make it more succinct and readable

```
class Solution {
  public int findMinArrowShots(int[][] points) {
    if (points.length == 0) return 0;
```

Read More

1 Share Reply

swapnil2927 45 May 18, 2020 11:03 AM

A part of the complexity analysis is obviously wrong here. Java's Arrays.sort uses merge sort for objects. So, space complexity should be O(N).

0 Share Reply

EshwarMolugu 0 February 10, 2020 12:02 PM

The solution fails for the input [[1,8],[2,6],[7,12]]. The answer should be 2 but the above solution returns 1.

0 Share Reply

SHOW 1 REPLY