Previous Next

57. Insert Interval

May 14, 2019 | 34.5K views

Average Rating: 4.63 (19 votes)

Given a set of *non-overlapping* intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:

```
Input: intervals = [[1,3],[6,9]], newInterval = [2,5]
Output: [[1,5],[6,9]]
```

Example 2:

```
Input: intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]
Output: [[1,2],[3,10],[12,16]]
Explanation: Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].
```

NOTE: input types have been changed on April 15, 2019. Please reset to default code definition to get new method signature.

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.

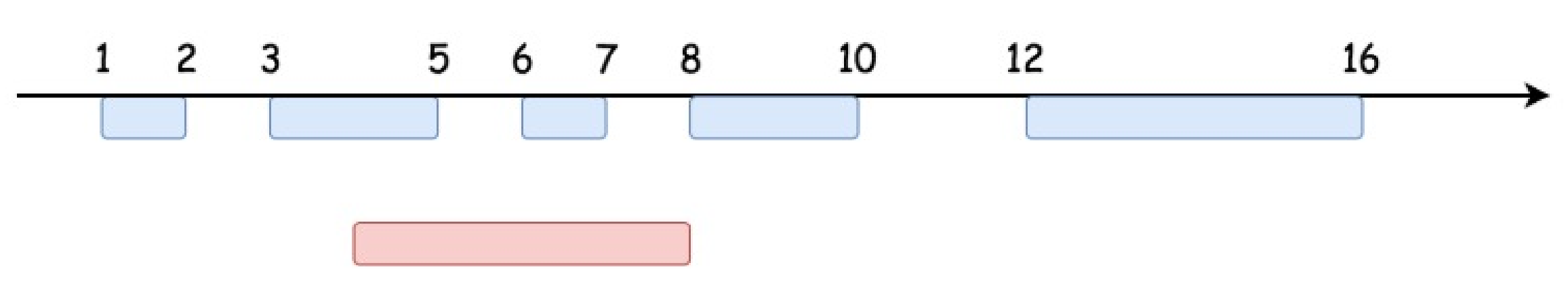
> How to prove that your greedy algorithm provides globally optimal solution?

Usually you could use the proof by contradiction.

Intuition

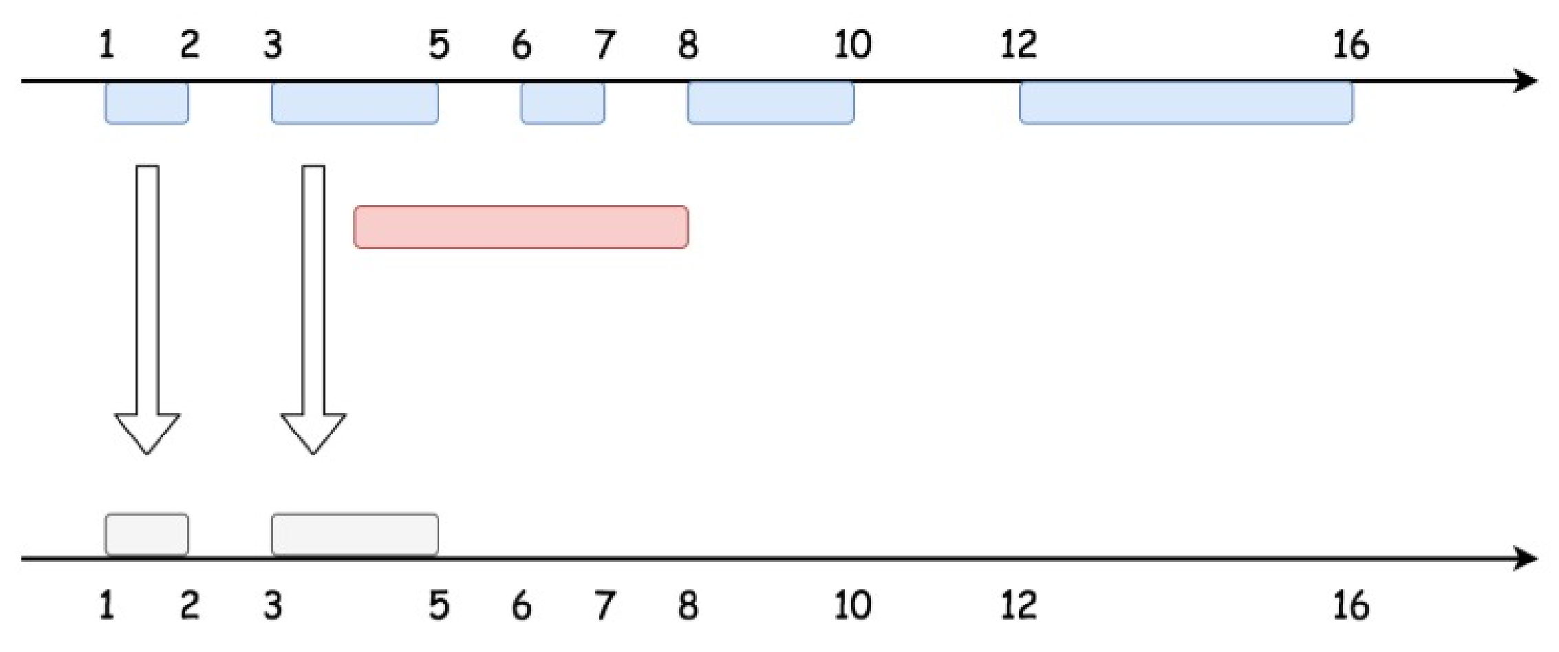
Here we have an example of a greedy problem with a well-sorted input, and hence the algorithm time complexity should be $\mathcal{O}(N)$.

Let's consider the following intervals

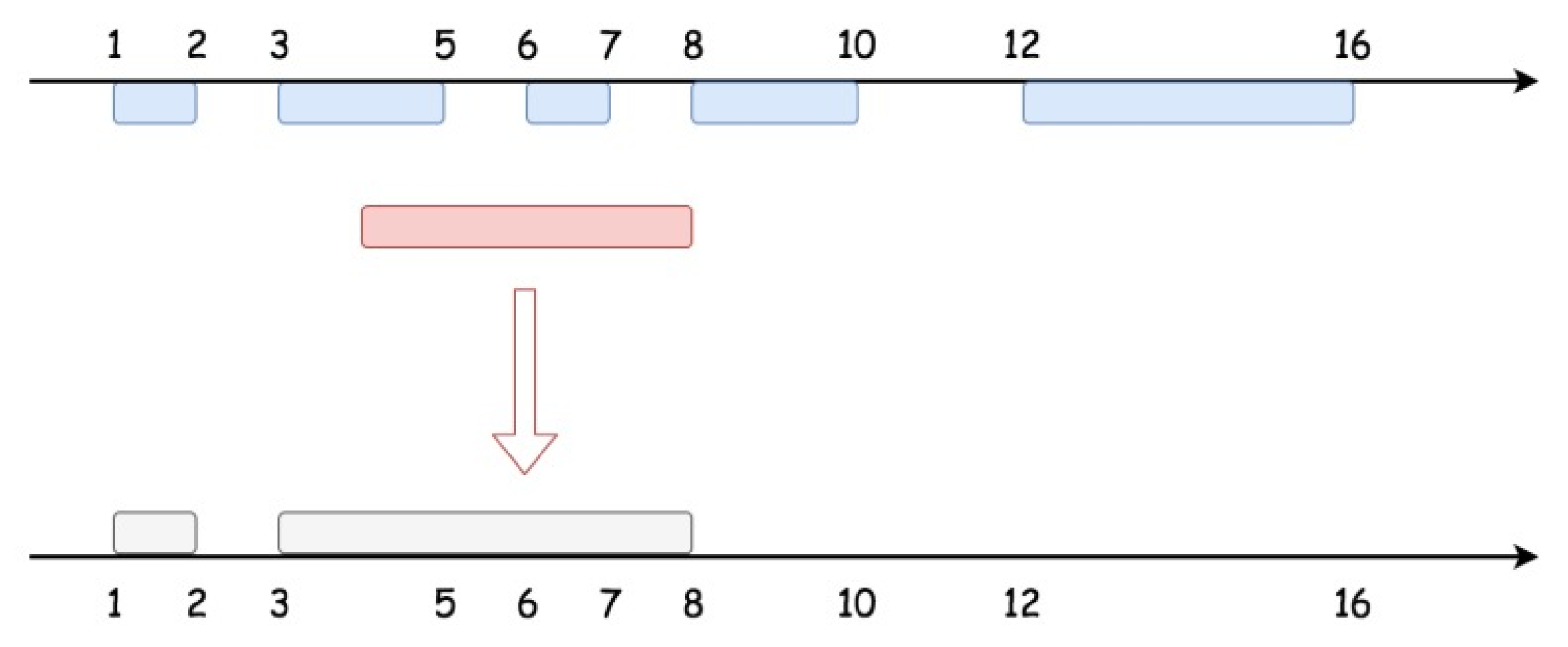


The straightforward one-pass strategy could be implemented in three steps.

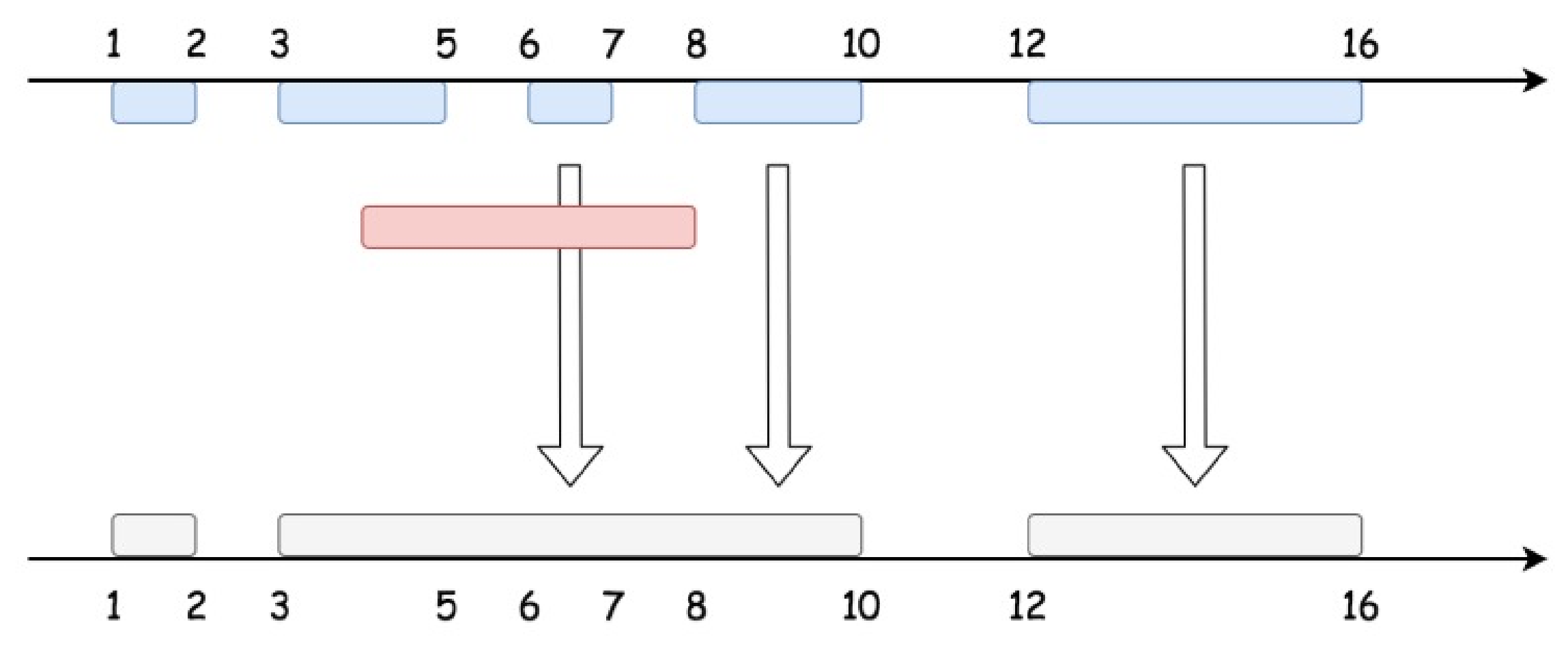
1 . Add to the output all the intervals starting before `newInterval`.



2 . Add to the output `newInterval`, merge it with the last added interval if needed.



3 . Add the next intervals one by one, merge if needed.



Basically, the same strategy as here, with an additional care to add the new interval in its proper position in order not to destroy the well-sorted input.

Algorithm

Here is the algorithm :

- Add to the output all the intervals starting before `newInterval`.
- Add to the output `newInterval`. Merge it with the last added interval if `newInterval` starts before the last added interval.
- Add the next intervals one by one. Merge with the last added interval if the current interval starts before the last added interval.

Implementation

Java Python Copy

```
class Solution:
    def insert(self, intervals: 'List[Interval]', newInterval: 'Interval') -> 'List[Interval]':
        # init data
        new_start, new_end = newInterval
        idx, n = 0, len(intervals)
        output = []

        # add all intervals starting before newInterval
        while idx < n and new_start > intervals[idx][0]:
            output.append(intervals[idx])
            idx += 1

        # add newInterval
        # if there is no overlap, just add the interval
        if not output or output[-1][1] < new_start:
            output.append(newInterval)
        # if there is an overlap, merge with the last interval
        else:
            output[-1][1] = max(output[-1][1], new_end)

        # add next intervals, merge with newInterval if needed
        while idx < n:
            interval = intervals[idx]
            start, end = interval
            idx += 1
            # if there is no overlap, just add an interval
            if output[-1][1] < start:
```

Complexity Analysis

- Time complexity : $\mathcal{O}(N)$ since it's one pass along the input array.
- Space complexity : $\mathcal{O}(N)$ to keep the output.

Rate this article: ★★★★★

Previous Next

Comments: 13

Sort By

Type comment here... (Markdown is supported)

Preview Post

hello_world_cn ★ 279 February 5, 2020 1:45 PM Report

A concise version:

```
class Solution {
  public int[][] insert(int[][] intervals, int[] newInterval) {
    // init data
```

Read More

11 Share Reply

SHOW 2 REPLIES

Ek-Do-Teen ★ 48 March 7, 2020 10:34 AM

Ummm... why is this a hard problem?

28 Share Reply

SHOW 2 REPLIES

girishiitj ★ 115 September 24, 2019 4:47 PM

We can find the first and last overlapping interval in O(logN) using binary search.

11 Share Reply

SHOW 3 REPLIES

hardfault ★ 256 January 11, 2020 6:53 AM Report

Come on! This shouldn't be tagged as hard. #351 **Android Unlock Pattern** is medium difficulty compared to this? xD

17 Share Reply

SHOW 3 REPLIES

knapsack ★ 6 October 4, 2019 8:45 PM

Is there a reason why we couldn't use binary search for this problem to search for the overlapping areas? This would then reduce the problem to O(logn)

4 Share Reply

SHOW 4 REPLIES

joseclaris6500 ★ 1 May 14, 2019 10:26 PM

Can I get an explaination of what this does: " def insert(self, intervals: 'List[Interval]', newInterval: 'Interval') -> 'List[Interval]':"?

Thank you

1 Share Reply

SHOW 2 REPLIES

arihant1467 ★ 55 May 15, 2019 11:51 PM

If we get an unsorted input, then it will make a complete interview question.

1 Share Reply

SHOW 1 REPLY

vaidygogetit ★ 6 May 17, 2019 8:40 AM Report

This solution will fail for this test case.
[[1,20],[1,40],[1,60]]
[2,8]

Output - [[1,20],[1,40],[1,60]]

Read More

0 Share Reply

SHOW 1 REPLY

user2766a ★ 0 May 30, 2020 5:07 AM Report

Hi, I was a bit misled by the problem statement where it simply says to insert a new interval and merge if necessary. I took this to mean that we had to merge the intervals in place after performing the insert. In Python,
this would be an O(N^2) operation with O(1) space. Would this be clarified in an interview setting, as it is not clear here.

Read More

0 Share Reply

Username1604 ★ 42 May 21, 2020 6:04 PM

didn't get the hard tag on that, I assumed there exists some superclever thing behind the lock, and not this one.

0 Share Reply

‹ 1 2 ›