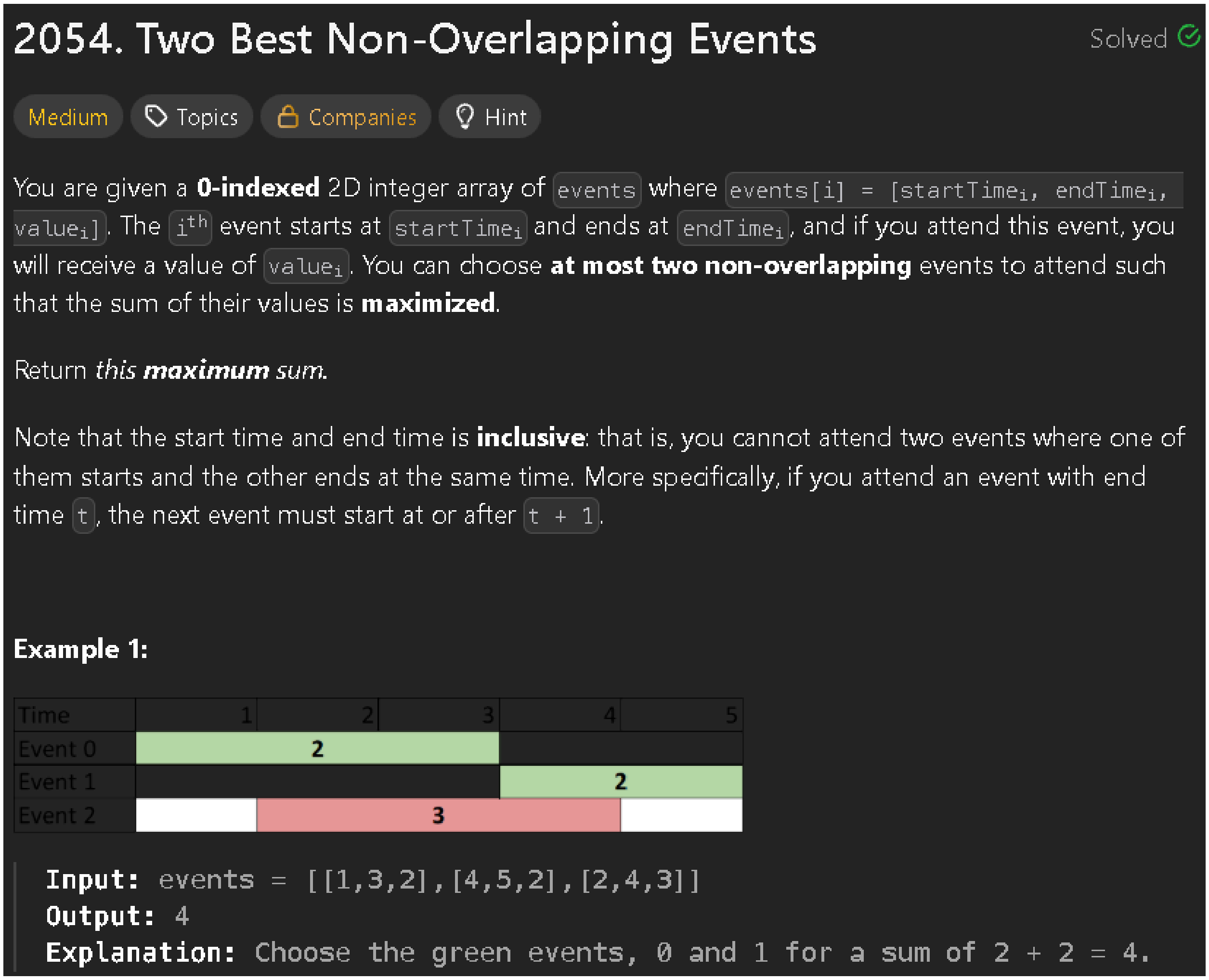


2054. Two Best Non-Overlapping Events

Solved ✓

You are given a **0-indexed** 2D integer array of `events` where `events[i] = [startTime_i, endTime_i, value_i]`. The i^{th} event starts at `startTime_i` and ends at `endTime_i`, and if you attend this event, you will receive a value of `value_i`. You can choose **at most two non-overlapping** events to attend such that the sum of their values is **maximized**.

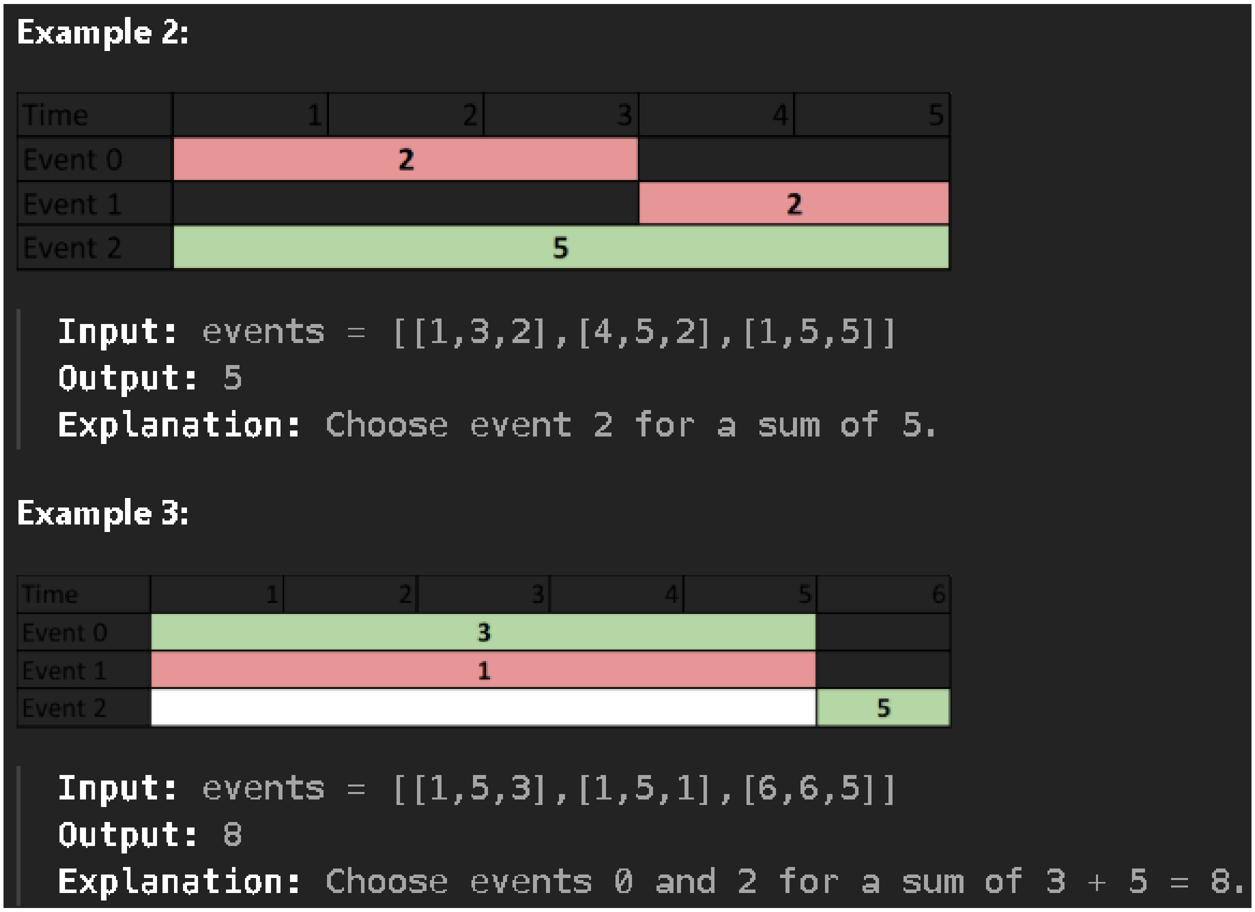
Return *this* ***maximum*** *sum.*

Note that the start time and end time is **inclusive**: that is, you cannot attend two events where one of them starts and the other ends at the same time. More specifically, if you attend an event with end time `t`, the next event must start at or after `t + 1`.

Example 1:

Time	1	2	3	4	5
Event 0	🟩	🟩 2	🟩		
Event 1				🟩 2	🟩
Event 2		🟥	🟥 3	🟥	

```
Input: events = [[1,3,2],[4,5,2],[2,4,3]]
Output: 4
Explanation: Choose the green events, 0 and 1 for a sum of 2 + 2 = 4.
```



Example 2:

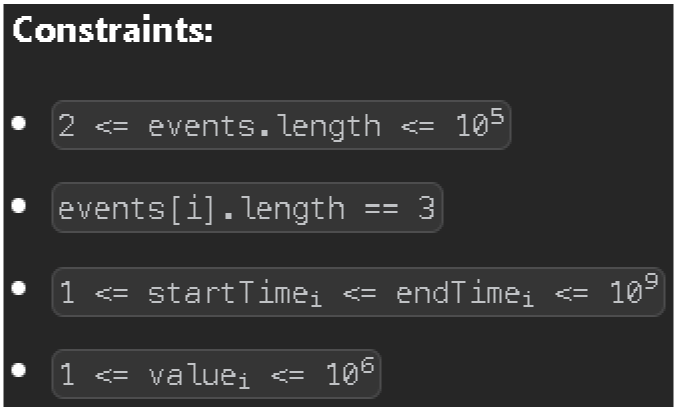
<table>
<tr><td>Time</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td></tr>
<tr><td>Event 0</td><td colspan="3">🟥 2</td><td></td><td></td></tr>
<tr><td>Event 1</td><td></td><td></td><td></td><td colspan="2">🟥 2</td></tr>
<tr><td>Event 2</td><td colspan="5">🟩 5</td></tr>
</table>

> **Input:** events = [[1,3,2],[4,5,2],[1,5,5]]
> **Output:** 5
> **Explanation:** Choose event 2 for a sum of 5.

Example 3:

<table>
<tr><td>Time</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr>
<tr><td>Event 0</td><td colspan="5">🟩 3</td><td></td></tr>
<tr><td>Event 1</td><td colspan="5">🟥 1</td><td></td></tr>
<tr><td>Event 2</td><td colspan="5">□</td><td>🟩 5</td></tr>
</table>

> **Input:** events = [[1,5,3],[1,5,1],[6,6,5]]
> **Output:** 8
> **Explanation:** Choose events 0 and 2 for a sum of 3 + 5 = 8.



Constraints:

- `2 <= events.length <=` 10^5
- `events[i].length == 3`
- `1 <=` $startTime_i$ `<=` $endTime_i$ `<=` 10^9
- `1 <=` $value_i$ `<=` 10^6

Python:

```
class Solution:
    def maxTwoEvents(self, events):
        events.sort(key=lambda x: x[1])
        n = len(events)

        endT = [0] * n
        best = [0] * n
        for i in range(n):
            endT[i] = events[i][1]
            best[i] = events[i][2]
            if i > 0:
                best[i] = max(best[i], best[i - 1])

        ans = 0

        for i in range(n):
            st, _, val = events[i]

            l, r = 0, n - 1
            idx = -1
            while l <= r:
                mid = (l + r) // 2
                if endT[mid] < st:
                    idx = mid
                    l = mid + 1
                else:
                    r = mid - 1

            if idx != -1:
                ans = max(ans, val + best[idx])
            ans = max(ans, val)

        return ans
```

JavaScript:

```
var maxTwoEvents = function(events) {
    events.sort((a, b) => a[1] - b[1]);

    const n = events.length;
    const endT = new Array(n);
    const best = new Array(n);

    for (let i = 0; i < n; i++) {
        endT[i] = events[i][1];
        best[i] = events[i][2];
        if (i > 0) best[i] = Math.max(best[i], best[i - 1]);
    }

    let ans = 0;

    for (let i = 0; i < n; i++) {
        const st = events[i][0];
        const val = events[i][2];

        let l = 0, r = n - 1, idx = -1;
        while (l <= r) {
            const mid = Math.floor((l + r) / 2);
            if (endT[mid] < st) {
                idx = mid;
                l = mid + 1;
            } else {
                r = mid - 1;
            }
        }

        if (idx !== -1) ans = Math.max(ans, val + best[idx]);
        ans = Math.max(ans, val);
    }

    return ans;
};
```

Java:

```
import java.util.*;

class Solution {
    public int maxTwoEvents(int[][] events) {
        Arrays.sort(events, (a, b) -> a[1] - b[1]);

        int n = events.length;
        int[] endT = new int[n];
        int[] best = new int[n];

        for (int i = 0; i < n; i++) {
            endT[i] = events[i][1];
            best[i] = events[i][2];
            if (i > 0) best[i] = Math.max(best[i], best[i - 1]);
        }

        int ans = 0;

        for (int i = 0; i < n; i++) {
            int st = events[i][0];
            int val = events[i][2];
            int l = 0, r = n - 1, idx = -1;
            while (l <= r) {
                int mid = (l + r) >>> 1;
                if (endT[mid] < st) {
                    idx = mid;
                    l = mid + 1;
                } else {
                    r = mid - 1;
                }
            }

            if (idx != -1) ans = Math.max(ans, val + best[idx]);
            ans = Math.max(ans, val);
        }

        return ans;
    }
}
```