Problem 2054: Two Best Non-Overlapping Events

Problem Information

Difficulty: Medium
Acceptance Rate: 0.00%
Paid Only: No

Problem Description

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

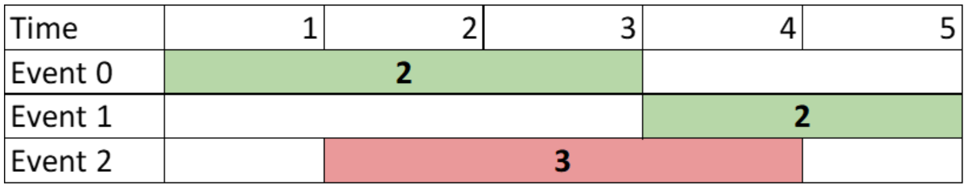
t

, 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]]

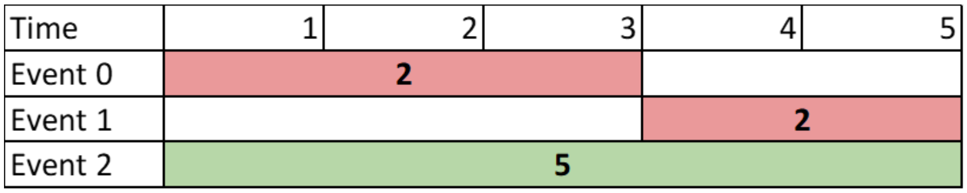
Output:

4

Explanation:

Choose the green events, 0 and 1 for a sum of 2 + 2 = 4.

Example 2:



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

Input:

events = [[1,3,2],[4,5,2],[1,5,5]]

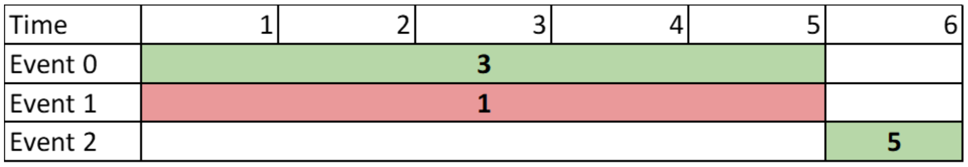
Output:

5

Explanation:

Choose event 2 for a sum of 5.

Example 3:



Time	1	2	3	4	5	6
Event 0			3			
Event 1			1			
Event 2						5

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

Code Snippets

C++:

```
class Solution {
public:
int maxTwoEvents(vector<vector<int>>& events) {

}
};
```

Java:

```
class Solution {
public int maxTwoEvents(int[][] events) {

}
}
```

Python3:

```
class Solution:
def maxTwoEvents(self, events: List[List[int]]) -> int:
```

Python:

```
class Solution(object):
def maxTwoEvents(self, events):
"""
:type events: List[List[int]]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number[][]} events
* @return {number}
*/
var maxTwoEvents = function(events) {

};
```

TypeScript:

```
function maxTwoEvents(events: number[][]): number {

};
```

C#:

```
public class Solution {
public int MaxTwoEvents(int[][] events) {

}
}
```

C:

```
int maxTwoEvents(int** events, int eventsSize, int* eventsColSize) {

}
```

Go:

```
func maxTwoEvents(events [][]int) int {

}
```

Kotlin:

```
class Solution {
fun maxTwoEvents(events: Array<IntArray>): Int {

}
}
```

Swift:

```
class Solution {
func maxTwoEvents(_ events: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_two_events(events: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby:

```
# @param {Integer[][]} events
# @return {Integer}
def max_two_events(events)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $events
* @return Integer
*/
function maxTwoEvents($events) {

}
}
```

Dart:

```
class Solution {
int maxTwoEvents(List<List<int>> events) {

}
}
```

Scala:

```
object Solution {
def maxTwoEvents(events: Array[Array[Int]]): Int = {

}
}
```

Elixir:

```
defmodule Solution do
@spec max_two_events(events :: [[integer]]) :: integer
def max_two_events(events) do

end
end
```

Erlang:

```
-spec max_two_events(Events :: [[integer()]]) -> integer().
max_two_events(Events) ->
.
```

Racket:

```
(define/contract (max-two-events events)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Two Best Non-Overlapping Events
* Difficulty: Medium
* Tags: array, dp, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public:
int maxTwoEvents(vector<vector<int>>& events) {

}
};
```

Java Solution:

```
/**
* Problem: Two Best Non-Overlapping Events
* Difficulty: Medium
* Tags: array, dp, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

class Solution {
public int maxTwoEvents(int[][] events) {
}
}
```

Python3 Solution:

```
"""
Problem: Two Best Non-Overlapping Events
Difficulty: Medium
Tags: array, dp, sort, search, queue, heap

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(n) or O(n * m) for DP table
"""

class Solution:
def maxTwoEvents(self, events: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxTwoEvents(self, events):
"""
:type events: List[List[int]]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Two Best Non-Overlapping Events
* Difficulty: Medium
* Tags: array, dp, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/
```

```
/**
* @param {number[][]} events
* @return {number}
*/
var maxTwoEvents = function(events) {

};
```

TypeScript Solution:

```
/**
* Problem: Two Best Non-Overlapping Events
* Difficulty: Medium
* Tags: array, dp, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

function maxTwoEvents(events: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Two Best Non-Overlapping Events
* Difficulty: Medium
* Tags: array, dp, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

public class Solution {
public int MaxTwoEvents(int[][] events) {

}
}
```

C Solution:

```
/*
* Problem: Two Best Non-Overlapping Events
* Difficulty: Medium
* Tags: array, dp, sort, search, queue, heap
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(n) or O(n * m) for DP table
*/

int maxTwoEvents(int** events, int eventsSize, int* eventsColSize) {

}
```

Go Solution:

```
// Problem: Two Best Non-Overlapping Events
// Difficulty: Medium
// Tags: array, dp, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

func maxTwoEvents(events [][]int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxTwoEvents(events: Array<IntArray>): Int {

}
}
```

Swift Solution:

```
class Solution {
func maxTwoEvents(_ events: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Two Best Non-Overlapping Events
// Difficulty: Medium
// Tags: array, dp, sort, search, queue, heap
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(n) or O(n * m) for DP table

impl Solution {
pub fn max_two_events(events: Vec<Vec<i32>>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} events
# @return {Integer}
def max_two_events(events)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $events
* @return Integer
*/
function maxTwoEvents($events) {

}
}
```

Dart Solution:

```
class Solution {
int maxTwoEvents(List<List<int>> events) {

}
}
```

Scala Solution:

```
object Solution {
def maxTwoEvents(events: Array[Array[Int]]): Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_two_events(events :: [[integer]]) :: integer
def max_two_events(events) do

end
end
```

Erlang Solution:

```
-spec max_two_events(Events :: [[integer()]]) -> integer().
max_two_events(Events) ->
.
```

Racket Solution:

```
(define/contract (max-two-events events)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```