Problem 3440: Reschedule Meetings for Maximum Free Time II

Problem Information

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

Problem Description

You are given an integer

eventTime

denoting the duration of an event. You are also given two integer arrays

startTime

and

endTime

, each of length

n

.

These represent the start and end times of

n

non-overlapping

meetings that occur during the event between time

t = 0

and time

t = eventTime

, where the

i

th

meeting occurs during the time

[startTime[i], endTime[i]].

You can reschedule

at most

one meeting by moving its start time while maintaining the

same duration

, such that the meetings remain non-overlapping, to

maximize

the

longest

continuous period of free time

during the event.

Return the

maximum

amount of free time possible after rearranging the meetings.

Note

that the meetings can

not

be rescheduled to a time outside the event and they should remain non-overlapping.

Note:

In this version

, it is

valid

for the relative ordering of the meetings to change after rescheduling one meeting.

Example 1:

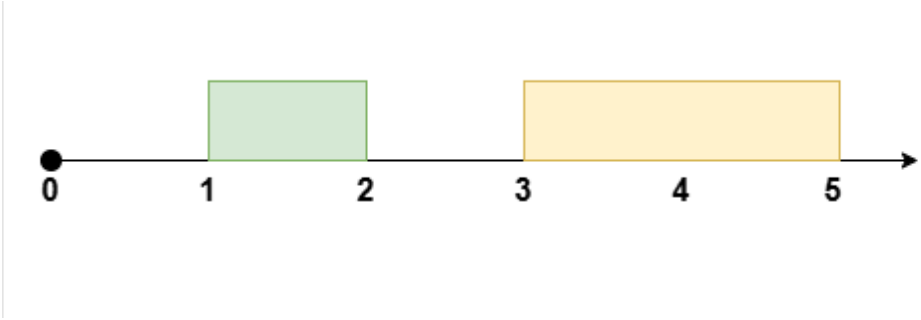
Input:

eventTime = 5, startTime = [1,3], endTime = [2,5]

Output:

2

Explanation:



Reschedule the meeting at

[1, 2]

to

[2, 3]

, leaving no meetings during the time

[0, 2]

.

Example 2:

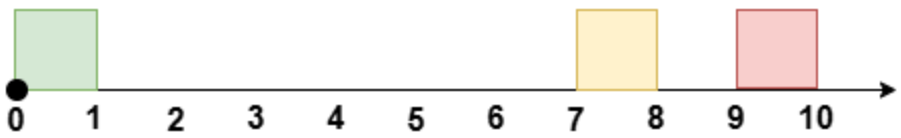
Input:

eventTime = 10, startTime = [0,7,9], endTime = [1,8,10]

Output:

7

Explanation:



Reschedule the meeting at

[0, 1]

to

[8, 9]

, leaving no meetings during the time

[0, 7]

.

Example 3:

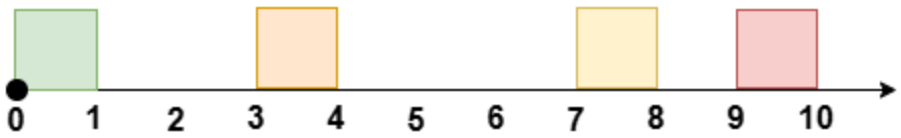
Input:

eventTime = 10, startTime = [0,3,7,9], endTime = [1,4,8,10]

Output:

6

Explanation:



Reschedule the meeting at

[3, 4]

to

[8, 9]

, leaving no meetings during the time

[1, 7]

.

Example 4:

Input:

eventTime = 5, startTime = [0,1,2,3,4], endTime = [1,2,3,4,5]

Output:

0

Explanation:

There is no time during the event not occupied by meetings.

Constraints:

1 <= eventTime <= 10

9

n == startTime.length == endTime.length

2 <= n <= 10

5

0 <= startTime[i] < endTime[i] <= eventTime

endTime[i] <= startTime[i + 1]

where

i

lies in the range

[0, n - 2]

.

Code Snippets

C++:

```
class Solution {
public:
int maxFreeTime(int eventTime, vector<int>& startTime, vector<int>& endTime)
{

}
};
```

Java:

```
class Solution {
public int maxFreeTime(int eventTime, int[] startTime, int[] endTime) {
```

```
}
}
```

Python3:

```
class Solution:
def maxFreeTime(self, eventTime: int, startTime: List[int], endTime:
List[int]) -> int:
```

Python:

```
class Solution(object):
def maxFreeTime(self, eventTime, startTime, endTime):
"""
:type eventTime: int
:type startTime: List[int]
:type endTime: List[int]
:rtype: int
"""
```

JavaScript:

```
/**
* @param {number} eventTime
* @param {number[]} startTime
* @param {number[]} endTime
* @return {number}
*/
var maxFreeTime = function(eventTime, startTime, endTime) {

};
```

TypeScript:

```
function maxFreeTime(eventTime: number, startTime: number[], endTime:
number[]): number {

};
```

C#:

```
public class Solution {
public int MaxFreeTime(int eventTime, int[] startTime, int[] endTime) {

}
}
```

C:

```
int maxFreeTime(int eventTime, int* startTime, int startTimeSize, int*
endTime, int endTimeSize) {

}
```

Go:

```
func maxFreeTime(eventTime int, startTime []int, endTime []int) int {

}
```

Kotlin:

```
class Solution {
fun maxFreeTime(eventTime: Int, startTime: IntArray, endTime: IntArray): Int
{

}
}
```

Swift:

```
class Solution {
func maxFreeTime(_ eventTime: Int, _ startTime: [Int], _ endTime: [Int]) ->
Int {

}
}
```

Rust:

```
impl Solution {
pub fn max_free_time(event_time: i32, start_time: Vec<i32>, end_time:
Vec<i32>) -> i32 {
```

```
}
}
```

Ruby:

```
# @param {Integer} event_time
# @param {Integer[]} start_time
# @param {Integer[]} end_time
# @return {Integer}
def max_free_time(event_time, start_time, end_time)

end
```

PHP:

```
class Solution {

/**
* @param Integer $eventTime
* @param Integer[] $startTime
* @param Integer[] $endTime
* @return Integer
*/
function maxFreeTime($eventTime, $startTime, $endTime) {

}
}
```

Dart:

```
class Solution {
int maxFreeTime(int eventTime, List<int> startTime, List<int> endTime) {

}
}
```

Scala:

```
object Solution {
def maxFreeTime(eventTime: Int, startTime: Array[Int], endTime: Array[Int]):
Int = {
```

```
}
}
```

Elixir:

```
defmodule Solution do
@spec max_free_time(event_time :: integer, start_time :: [integer], end_time
:: [integer]) :: integer
def max_free_time(event_time, start_time, end_time) do

end
end
```

Erlang:

```
-spec max_free_time(EventTime :: integer(), StartTime :: [integer()], EndTime
:: [integer()]) -> integer().
max_free_time(EventTime, StartTime, EndTime) ->
.
```

Racket:

```
(define/contract (max-free-time eventTime startTime endTime)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Reschedule Meetings for Maximum Free Time II
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
class Solution {
public:
int maxFreeTime(int eventTime, vector<int>& startTime, vector<int>& endTime)
{

}
};
```

Java Solution:

```
/**
* Problem: Reschedule Meetings for Maximum Free Time II
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

class Solution {
public int maxFreeTime(int eventTime, int[] startTime, int[] endTime) {

}
}
```

Python3 Solution:

```
"""
Problem: Reschedule Meetings for Maximum Free Time II
Difficulty: Medium
Tags: array, greedy

Approach: Use two pointers or sliding window technique
Time Complexity: O(n) or O(n log n)
Space Complexity: O(1) to O(n) depending on approach
"""

class Solution:
def maxFreeTime(self, eventTime: int, startTime: List[int], endTime:
```

```
List[int]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class Solution(object):
def maxFreeTime(self, eventTime, startTime, endTime):
"""
:type eventTime: int
:type startTime: List[int]
:type endTime: List[int]
:rtype: int
"""
```

JavaScript Solution:

```
/**
* Problem: Reschedule Meetings for Maximum Free Time II
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

/**
* @param {number} eventTime
* @param {number[]} startTime
* @param {number[]} endTime
* @return {number}
*/
var maxFreeTime = function(eventTime, startTime, endTime) {

};
```

TypeScript Solution:

```
/**
* Problem: Reschedule Meetings for Maximum Free Time II
* Difficulty: Medium
```

```
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

function maxFreeTime(eventTime: number, startTime: number[], endTime:
number[]): number {

};
```

C# Solution:

```
/*
* Problem: Reschedule Meetings for Maximum Free Time II
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/

public class Solution {
public int MaxFreeTime(int eventTime, int[] startTime, int[] endTime) {

}
}
```

C Solution:

```
/*
* Problem: Reschedule Meetings for Maximum Free Time II
* Difficulty: Medium
* Tags: array, greedy
*
* Approach: Use two pointers or sliding window technique
* Time Complexity: O(n) or O(n log n)
* Space Complexity: O(1) to O(n) depending on approach
*/
```

```
int maxFreeTime(int eventTime, int* startTime, int startTimeSize, int*
endTime, int endTimeSize) {

}
```

Go Solution:

```
// Problem: Reschedule Meetings for Maximum Free Time II
// Difficulty: Medium
// Tags: array, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

func maxFreeTime(eventTime int, startTime []int, endTime []int) int {

}
```

Kotlin Solution:

```
class Solution {
fun maxFreeTime(eventTime: Int, startTime: IntArray, endTime: IntArray): Int
{

}
}
```

Swift Solution:

```
class Solution {
func maxFreeTime(_ eventTime: Int, _ startTime: [Int], _ endTime: [Int]) ->
Int {

}
}
```

Rust Solution:

```
// Problem: Reschedule Meetings for Maximum Free Time II
// Difficulty: Medium
// Tags: array, greedy
//
// Approach: Use two pointers or sliding window technique
// Time Complexity: O(n) or O(n log n)
// Space Complexity: O(1) to O(n) depending on approach

impl Solution {
pub fn max_free_time(event_time: i32, start_time: Vec<i32>, end_time:
Vec<i32>) -> i32 {

}
}
```

Ruby Solution:

```
# @param {Integer} event_time
# @param {Integer[]} start_time
# @param {Integer[]} end_time
# @return {Integer}
def max_free_time(event_time, start_time, end_time)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer $eventTime
* @param Integer[] $startTime
* @param Integer[] $endTime
* @return Integer
*/
function maxFreeTime($eventTime, $startTime, $endTime) {

}
}
```

Dart Solution:

```
class Solution {
int maxFreeTime(int eventTime, List<int> startTime, List<int> endTime) {

}
}
```

Scala Solution:

```
object Solution {
def maxFreeTime(eventTime: Int, startTime: Array[Int], endTime: Array[Int]):
Int = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec max_free_time(event_time :: integer, start_time :: [integer], end_time
:: [integer]) :: integer
def max_free_time(event_time, start_time, end_time) do

end
end
```

Erlang Solution:

```
-spec max_free_time(EventTime :: integer(), StartTime :: [integer()], EndTime
:: [integer()]) -> integer().
max_free_time(EventTime, StartTime, EndTime) ->
.
```

Racket Solution:

```
(define/contract (max-free-time eventTime startTime endTime)
(-> exact-integer? (listof exact-integer?) (listof exact-integer?)
exact-integer?)
)
```