Problem 3709: Design Exam Scores Tracker

Problem Information

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

Problem Description

Alice frequently takes exams and wants to track her scores and calculate the total scores over specific time periods.

Implement the

ExamTracker

class:

ExamTracker()

: Initializes the

ExamTracker

object.

void record(int time, int score)

: Alice takes a new exam at time

time

and achieves the score

score

.

long long totalScore(int startTime, int endTime)

: Returns an integer that represents the

total

score of all exams taken by Alice between

startTime

and

endTime

(inclusive). If there are no recorded exams taken by Alice within the specified time interval, return 0.

It is guaranteed that the function calls are made in chronological order. That is,

Calls to

record()

will be made with

strictly increasing

time

.

Alice will never ask for total scores that require information from the future. That is, if the latest

record()

is called with

time = t

, then

totalScore()

will always be called with

startTime <= endTime <= t

.

Example 1:

Input:

["ExamTracker", "record", "totalScore", "record", "totalScore", "totalScore", "totalScore", "totalScore"]

[[], [1, 98], [1, 1], [5, 99], [1, 3], [1, 5], [3, 4], [2, 5]]

Output:

[null, null, 98, null, 98, 197, 0, 99]

Explanation

ExamTracker examTracker = new ExamTracker();

examTracker.record(1, 98); // Alice takes a new exam at time 1, scoring 98.

examTracker.totalScore(1, 1); // Between time 1 and time 1, Alice took 1 exam at time 1, scoring 98. The total score is 98.

examTracker.record(5, 99); // Alice takes a new exam at time 5, scoring 99.

examTracker.totalScore(1, 3); // Between time 1 and time 3, Alice took 1 exam at time 1, scoring 98. The total score is 98.

examTracker.totalScore(1, 5); // Between time 1 and time 5, Alice took 2 exams at time 1 and 5, scoring 98 and 99. The total score is

98 + 99 = 197

.

examTracker.totalScore(3, 4); // Alice did not take any exam between time 3 and time 4. Therefore, the answer is 0.

examTracker.totalScore(2, 5); // Between time 2 and time 5, Alice took 1 exam at time 5, scoring 99. The total score is 99.

Constraints:

1 <= time <= 10

9

1 <= score <= 10

9

1 <= startTime <= endTime <= t

, where

t

is the value of

time

from the most recent call of

record()

.

Calls of

record()

will be made with

strictly increasing

time

.

After

ExamTracker()

, the first function call will always be

record()

.

At most

10

5

calls will be made in total to

record()

and

totalScore()

.

Code Snippets

C++:

```
class ExamTracker {
public:
ExamTracker() {

}

void record(int time, int score) {

}

long long totalScore(int startTime, int endTime) {

}
};

/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker* obj = new ExamTracker();
* obj->record(time,score);
* long long param_2 = obj->totalScore(startTime,endTime);
*/
```

Java:

```
class ExamTracker {

public ExamTracker() {

}

public void record(int time, int score) {

}

public long totalScore(int startTime, int endTime) {

}
}
```

```
/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker obj = new ExamTracker();
* obj.record(time,score);
* long param_2 = obj.totalScore(startTime,endTime);
*/
```

Python3:

```
class ExamTracker:

def __init__(self):


def record(self, time: int, score: int) -> None:


def totalScore(self, startTime: int, endTime: int) -> int:



# Your ExamTracker object will be instantiated and called as such:
# obj = ExamTracker()
# obj.record(time,score)
# param_2 = obj.totalScore(startTime,endTime)
```

Python:

```
class ExamTracker(object):

def __init__(self):


def record(self, time, score):
"""
:type time: int
:type score: int
:rtype: None
"""


def totalScore(self, startTime, endTime):
```

```
"""
:type startTime: int
:type endTime: int
:rtype: int
"""



# Your ExamTracker object will be instantiated and called as such:
# obj = ExamTracker()
# obj.record(time,score)
# param_2 = obj.totalScore(startTime,endTime)
```

JavaScript:

```
var ExamTracker = function() {

};

/**
* @param {number} time
* @param {number} score
* @return {void}
*/
ExamTracker.prototype.record = function(time, score) {

};

/**
* @param {number} startTime
* @param {number} endTime
* @return {number}
*/
ExamTracker.prototype.totalScore = function(startTime, endTime) {

};

/**
* Your ExamTracker object will be instantiated and called as such:
* var obj = new ExamTracker()
* obj.record(time,score)
```

```
* var param_2 = obj.totalScore(startTime,endTime)
*/
```

TypeScript:

```
class ExamTracker {
constructor() {

}

record(time: number, score: number): void {

}

totalScore(startTime: number, endTime: number): number {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* var obj = new ExamTracker()
* obj.record(time,score)
* var param_2 = obj.totalScore(startTime,endTime)
*/
```

C#:

```
public class ExamTracker {

public ExamTracker() {

}

public void Record(int time, int score) {

}

public long TotalScore(int startTime, int endTime) {

}
}
```

```
/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker obj = new ExamTracker();
* obj.Record(time,score);
* long param_2 = obj.TotalScore(startTime,endTime);
*/
```

C:

```
typedef struct {

} ExamTracker;


ExamTracker* examTrackerCreate() {

}

void examTrackerRecord(ExamTracker* obj, int time, int score) {

}

long long examTrackerTotalScore(ExamTracker* obj, int startTime, int endTime)
{

}

void examTrackerFree(ExamTracker* obj) {

}

/**
* Your ExamTracker struct will be instantiated and called as such:
* ExamTracker* obj = examTrackerCreate();
* examTrackerRecord(obj, time, score);

* long long param_2 = examTrackerTotalScore(obj, startTime, endTime);
```

```
* examTrackerFree(obj);
*/
```

Go:

```
type ExamTracker struct {

}


func Constructor() ExamTracker {

}


func (this *ExamTracker) Record(time int, score int) {

}


func (this *ExamTracker) TotalScore(startTime int, endTime int) int64 {

}


/**
* Your ExamTracker object will be instantiated and called as such:
* obj := Constructor();
* obj.Record(time,score);
* param_2 := obj.TotalScore(startTime,endTime);
*/
```

Kotlin:

```
class ExamTracker() {

fun record(time: Int, score: Int) {

}

fun totalScore(startTime: Int, endTime: Int): Long {
```

```
}

}

/**
* Your ExamTracker object will be instantiated and called as such:
* var obj = ExamTracker()
* obj.record(time,score)
* var param_2 = obj.totalScore(startTime,endTime)
*/
```

Swift:

```
class ExamTracker {

init() {

}

func record(_ time: Int, _ score: Int) {

}

func totalScore(_ startTime: Int, _ endTime: Int) -> Int {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* let obj = ExamTracker()
* obj.record(time, score)
* let ret_2: Int = obj.totalScore(startTime, endTime)
*/
```

Rust:

```
struct ExamTracker {

}
```

```
/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl ExamTracker {

fn new() -> Self {

}

fn record(&self, time: i32, score: i32) {

}

fn total_score(&self, start_time: i32, end_time: i32) -> i64 {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* let obj = ExamTracker::new();
* obj.record(time, score);
* let ret_2: i64 = obj.total_score(startTime, endTime);
*/
```

Ruby:

```
class ExamTracker
def initialize()

end


=begin
:type time: Integer
:type score: Integer
:rtype: Void
=end
def record(time, score)
```

```
end


=begin
:type start_time: Integer
:type end_time: Integer
:rtype: Integer
=end
def total_score(start_time, end_time)

end



end

# Your ExamTracker object will be instantiated and called as such:
# obj = ExamTracker.new()
# obj.record(time, score)
# param_2 = obj.total_score(start_time, end_time)
```

PHP:

```
class ExamTracker {
/**
*/
function __construct() {

}

/**
* @param Integer $time
* @param Integer $score
* @return NULL
*/
function record($time, $score) {

}

/**
* @param Integer $startTime
* @param Integer $endTime
* @return Integer
```

```
*/
function totalScore($startTime, $endTime) {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* $obj = ExamTracker();
* $obj->record($time, $score);
* $ret_2 = $obj->totalScore($startTime, $endTime);
*/
```

Dart:

```
class ExamTracker {

ExamTracker() {

}

void record(int time, int score) {

}

int totalScore(int startTime, int endTime) {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker obj = ExamTracker();
* obj.record(time,score);
* int param2 = obj.totalScore(startTime,endTime);
*/
```

Scala:

```
class ExamTracker() {

def record(time: Int, score: Int): Unit = {
```

```
}

def totalScore(startTime: Int, endTime: Int): Long = {

}

}

/**
* Your ExamTracker object will be instantiated and called as such:
* val obj = new ExamTracker()
* obj.record(time,score)
* val param_2 = obj.totalScore(startTime,endTime)
*/
```

Elixir:

```
defmodule ExamTracker do
@spec init_() :: any
def init_() do

end

@spec record(time :: integer, score :: integer) :: any
def record(time, score) do

end

@spec total_score(start_time :: integer, end_time :: integer) :: integer
def total_score(start_time, end_time) do

end
end

# Your functions will be called as such:
# ExamTracker.init_()
# ExamTracker.record(time, score)
# param_2 = ExamTracker.total_score(start_time, end_time)

# ExamTracker.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang:

```
-spec exam_tracker_init_() -> any().
exam_tracker_init_() ->
.

-spec exam_tracker_record(Time :: integer(), Score :: integer()) -> any().
exam_tracker_record(Time, Score) ->
.

-spec exam_tracker_total_score(StartTime :: integer(), EndTime :: integer())
-> integer().
exam_tracker_total_score(StartTime, EndTime) ->
.


%% Your functions will be called as such:
%% exam_tracker_init_(),
%% exam_tracker_record(Time, Score),
%% Param_2 = exam_tracker_total_score(StartTime, EndTime),

%% exam_tracker_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket:

```
(define exam-tracker%
(class object%
(super-new)

(init-field)

; record : exact-integer? exact-integer? -> void?
(define/public (record time score)
)
; total-score : exact-integer? exact-integer? -> exact-integer?
(define/public (total-score start-time end-time)
)))

;; Your exam-tracker% object will be instantiated and called as such:
;; (define obj (new exam-tracker%))
;; (send obj record time score)
```

```
;; (define param_2 (send obj total-score start-time end-time))
```

Solutions

C++ Solution:

```
/*
* Problem: Design Exam Scores Tracker
* Difficulty: Medium
* Tags: array, search
*
* 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 ExamTracker {
public:
ExamTracker() {

}

void record(int time, int score) {

}

long long totalScore(int startTime, int endTime) {

}
};

/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker* obj = new ExamTracker();
* obj->record(time,score);
* long long param_2 = obj->totalScore(startTime,endTime);
*/
```

Java Solution:

```
/**
* Problem: Design Exam Scores Tracker
* Difficulty: Medium
* Tags: array, search
*
* 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 ExamTracker {

public ExamTracker() {

}

public void record(int time, int score) {

}

public long totalScore(int startTime, int endTime) {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker obj = new ExamTracker();
* obj.record(time,score);
* long param_2 = obj.totalScore(startTime,endTime);
*/
```

Python3 Solution:

```
"""
Problem: Design Exam Scores Tracker
Difficulty: Medium
Tags: array, search

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 ExamTracker:

def __init__(self):


def record(self, time: int, score: int) -> None:
# TODO: Implement optimized solution
pass
```

Python Solution:

```
class ExamTracker(object):

def __init__(self):


def record(self, time, score):
"""
:type time: int
:type score: int
:rtype: None
"""


def totalScore(self, startTime, endTime):
"""
:type startTime: int
:type endTime: int
:rtype: int
"""




# Your ExamTracker object will be instantiated and called as such:
# obj = ExamTracker()
# obj.record(time,score)
# param_2 = obj.totalScore(startTime,endTime)
```

JavaScript Solution:

```javascript
/**
* Problem: Design Exam Scores Tracker
* Difficulty: Medium
* Tags: array, search
*
* 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
*/


var ExamTracker = function() {

};

/**
* @param {number} time
* @param {number} score
* @return {void}
*/
ExamTracker.prototype.record = function(time, score) {

};

/**
* @param {number} startTime
* @param {number} endTime
* @return {number}
*/
ExamTracker.prototype.totalScore = function(startTime, endTime) {

};

/**
* Your ExamTracker object will be instantiated and called as such:
* var obj = new ExamTracker()
* obj.record(time,score)
* var param_2 = obj.totalScore(startTime,endTime)
*/
```

TypeScript Solution:

```
/**
* Problem: Design Exam Scores Tracker
* Difficulty: Medium
* Tags: array, search
*
* 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 ExamTracker {
constructor() {

}

record(time: number, score: number): void {

}

totalScore(startTime: number, endTime: number): number {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* var obj = new ExamTracker()
* obj.record(time,score)
* var param_2 = obj.totalScore(startTime,endTime)
*/
```

C# Solution:

```
/*
* Problem: Design Exam Scores Tracker
* Difficulty: Medium
* Tags: array, search
*
* 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 ExamTracker {

public ExamTracker() {

}

public void Record(int time, int score) {

}

public long TotalScore(int startTime, int endTime) {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker obj = new ExamTracker();
* obj.Record(time,score);
* long param_2 = obj.TotalScore(startTime,endTime);
*/
```

C Solution:

```
/*
* Problem: Design Exam Scores Tracker
* Difficulty: Medium
* Tags: array, search
*
* 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
*/




typedef struct {

} ExamTracker;
```

```
ExamTracker* examTrackerCreate() {

}

void examTrackerRecord(ExamTracker* obj, int time, int score) {

}

long long examTrackerTotalScore(ExamTracker* obj, int startTime, int endTime)
{

}

void examTrackerFree(ExamTracker* obj) {

}

/**
* Your ExamTracker struct will be instantiated and called as such:
* ExamTracker* obj = examTrackerCreate();
* examTrackerRecord(obj, time, score);

* long long param_2 = examTrackerTotalScore(obj, startTime, endTime);

* examTrackerFree(obj);
*/
```

Go Solution:

```
// Problem: Design Exam Scores Tracker
// Difficulty: Medium
// Tags: array, search
//
// 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

type ExamTracker struct {
```

```
}


func Constructor() ExamTracker {

}


func (this *ExamTracker) Record(time int, score int) {

}


func (this *ExamTracker) TotalScore(startTime int, endTime int) int64 {

}


/**
* Your ExamTracker object will be instantiated and called as such:
* obj := Constructor();
* obj.Record(time,score);
* param_2 := obj.TotalScore(startTime,endTime);
*/
```

Kotlin Solution:

```
class ExamTracker() {

fun record(time: Int, score: Int) {

}

fun totalScore(startTime: Int, endTime: Int): Long {

}

}

/**
* Your ExamTracker object will be instantiated and called as such:
```

```
* var obj = ExamTracker()
* obj.record(time,score)
* var param_2 = obj.totalScore(startTime,endTime)
*/
```

Swift Solution:

```
class ExamTracker {

init() {

}

func record(_ time: Int, _ score: Int) {

}

func totalScore(_ startTime: Int, _ endTime: Int) -> Int {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* let obj = ExamTracker()
* obj.record(time, score)
* let ret_2: Int = obj.totalScore(startTime, endTime)
*/
```

Rust Solution:

```
// Problem: Design Exam Scores Tracker
// Difficulty: Medium
// Tags: array, search
//
// 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

struct ExamTracker {
```

```
}


/**
* `&self` means the method takes an immutable reference.
* If you need a mutable reference, change it to `&mut self` instead.
*/
impl ExamTracker {

fn new() -> Self {

}

fn record(&self, time: i32, score: i32) {

}

fn total_score(&self, start_time: i32, end_time: i32) -> i64 {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* let obj = ExamTracker::new();
* obj.record(time, score);
* let ret_2: i64 = obj.total_score(startTime, endTime);
*/
```

Ruby Solution:

```
class ExamTracker
def initialize()

end


=begin
:type time: Integer
:type score: Integer
```

```
:rtype: Void
=end
def record(time, score)

end


=begin
:type start_time: Integer
:type end_time: Integer
:rtype: Integer
=end
def total_score(start_time, end_time)

end



end

# Your ExamTracker object will be instantiated and called as such:
# obj = ExamTracker.new()
# obj.record(time, score)
# param_2 = obj.total_score(start_time, end_time)
```

PHP Solution:

```
class ExamTracker {
/**
*/
function __construct() {

}

/**
* @param Integer $time
* @param Integer $score
* @return NULL
*/
function record($time, $score) {

}
```

```
/**
* @param Integer $startTime
* @param Integer $endTime
* @return Integer
*/
function totalScore($startTime, $endTime) {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* $obj = ExamTracker();
* $obj->record($time, $score);
* $ret_2 = $obj->totalScore($startTime, $endTime);
*/
```

Dart Solution:

```
class ExamTracker {

ExamTracker() {

}

void record(int time, int score) {

}

int totalScore(int startTime, int endTime) {

}
}

/**
* Your ExamTracker object will be instantiated and called as such:
* ExamTracker obj = ExamTracker();
* obj.record(time,score);
* int param2 = obj.totalScore(startTime,endTime);
*/
```

Scala Solution:

```
class ExamTracker() {

def record(time: Int, score: Int): Unit = {

}

def totalScore(startTime: Int, endTime: Int): Long = {

}

}

/**
* Your ExamTracker object will be instantiated and called as such:
* val obj = new ExamTracker()
* obj.record(time,score)
* val param_2 = obj.totalScore(startTime,endTime)
*/
```

Elixir Solution:

```
defmodule ExamTracker do
@spec init_() :: any
def init_() do

end

@spec record(time :: integer, score :: integer) :: any
def record(time, score) do

end

@spec total_score(start_time :: integer, end_time :: integer) :: integer
def total_score(start_time, end_time) do

end
end

# Your functions will be called as such:
# ExamTracker.init_()
```

```
# ExamTracker.record(time, score)
# param_2 = ExamTracker.total_score(start_time, end_time)

# ExamTracker.init_ will be called before every test case, in which you can
do some necessary initializations.
```

Erlang Solution:

```
-spec exam_tracker_init_() -> any().
exam_tracker_init_() ->
.

-spec exam_tracker_record(Time :: integer(), Score :: integer()) -> any().
exam_tracker_record(Time, Score) ->
.

-spec exam_tracker_total_score(StartTime :: integer(), EndTime :: integer())
-> integer().
exam_tracker_total_score(StartTime, EndTime) ->
.


%% Your functions will be called as such:
%% exam_tracker_init_(),
%% exam_tracker_record(Time, Score),
%% Param_2 = exam_tracker_total_score(StartTime, EndTime),

%% exam_tracker_init_ will be called before every test case, in which you can
do some necessary initializations.
```

Racket Solution:

```
(define exam-tracker%
(class object%
(super-new)

(init-field)

; record : exact-integer? exact-integer? -> void?
(define/public (record time score)
)
```

```
; total-score : exact-integer? exact-integer? -> exact-integer?
(define/public (total-score start-time end-time)
)))

;; Your exam-tracker% object will be instantiated and called as such:
;; (define obj (new exam-tracker%))
;; (send obj record time score)
;; (define param_2 (send obj total-score start-time end-time))
```