Problem 3709: Design Exam Scores Tracker

Problem Information

Difficulty: Medium
Acceptance Rate: 43.32%
Paid Only: No
Tags: Array, Binary Search, Design, Prefix Sum

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 <= 109` * `1 <= score <= 109` * `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 `105` 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)
```