Problem 2140: Solving Questions With Brainpower

Problem Information

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

Problem Description

You are given a

0-indexed

2D integer array

questions

where

questions[i] = [points

i

, brainpower

i

]

.

The array describes the questions of an exam, where you have to process the questions

in order

(i.e., starting from question

0

) and make a decision whether to

solve

or

skip

each question. Solving question

i

will

earn

you

points

i

points but you will be

unable

to solve each of the next

brainpower

i

questions. If you skip question

i

, you get to make the decision on the next question.

For example, given

questions = [[3, 2], [4, 3], [4, 4], [2, 5]]

:

If question

0

is solved, you will earn

3

points but you will be unable to solve questions

1

and

2

.

If instead, question

0

is skipped and question

1

is solved, you will earn

4

points but you will be unable to solve questions

2

and

3

.

Return

the

maximum

points you can earn for the exam

.

Example 1:

Input:

questions = [[3,2],[4,3],[4,4],[2,5]]

Output:

5

Explanation:

The maximum points can be earned by solving questions 0 and 3. - Solve question 0: Earn 3 points, will be unable to solve the next 2 questions - Unable to solve questions 1 and 2 - Solve question 3: Earn 2 points Total points earned: 3 + 2 = 5. There is no other way to earn 5 or more points.

Example 2:

Input:

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

Output:

7

Explanation:

The maximum points can be earned by solving questions 1 and 4. - Skip question 0 - Solve question 1: Earn 2 points, will be unable to solve the next 2 questions - Unable to solve questions 2 and 3 - Solve question 4: Earn 5 points Total points earned: 2 + 5 = 7. There is no other way to earn 7 or more points.

Constraints:

1 <= questions.length <= 10

5

questions[i].length == 2

1 <= points

i

, brainpower

i

<= 10

5

Code Snippets

C++:

```
class Solution {
public:
long long mostPoints(vector<vector<int>>& questions) {

}
};
```

Java:

```
class Solution {
public long mostPoints(int[][] questions) {

}
}
```

Python3:

```
class Solution:
def mostPoints(self, questions: List[List[int]]) -> int:
```

Python:

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

JavaScript:

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

};
```

TypeScript:

```
function mostPoints(questions: number[][]): number {
```

```
};
```

C#:

```
public class Solution {
public long MostPoints(int[][] questions) {

}
}
```

C:

```
long long mostPoints(int** questions, int questionsSize, int*
questionsColSize) {

}
```

Go:

```
func mostPoints(questions [][]int) int64 {

}
```

Kotlin:

```
class Solution {
fun mostPoints(questions: Array<IntArray>): Long {

}
}
```

Swift:

```
class Solution {
func mostPoints(_ questions: [[Int]]) -> Int {

}
}
```

Rust:

```
impl Solution {
pub fn most_points(questions: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby:

```
# @param {Integer[][]} questions
# @return {Integer}
def most_points(questions)

end
```

PHP:

```
class Solution {

/**
* @param Integer[][] $questions
* @return Integer
*/
function mostPoints($questions) {

}
}
```

Dart:

```
class Solution {
int mostPoints(List<List<int>> questions) {

}
}
```

Scala:

```
object Solution {
def mostPoints(questions: Array[Array[Int]]): Long = {

}
}
```

Elixir:

```
defmodule Solution do
@spec most_points(questions :: [[integer]]) :: integer
def most_points(questions) do

end
end
```

Erlang:

```
-spec most_points(Questions :: [[integer()]]) -> integer().
most_points(Questions) ->
.
```

Racket:

```
(define/contract (most-points questions)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```

Solutions

C++ Solution:

```
/*
* Problem: Solving Questions With Brainpower
* Difficulty: Medium
* Tags: array, dp
*
* 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:
long long mostPoints(vector<vector<int>>& questions) {

}
};
```

Java Solution:

```
/**
* Problem: Solving Questions With Brainpower
* Difficulty: Medium
* Tags: array, dp
*
* 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 long mostPoints(int[][] questions) {

}
}
```

Python3 Solution:

```
"""
Problem: Solving Questions With Brainpower
Difficulty: Medium
Tags: array, dp

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 mostPoints(self, questions: List[List[int]]) -> int:
# TODO: Implement optimized solution
pass
```

Python Solution:

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

```
"""
```

JavaScript Solution:

```
/**
* Problem: Solving Questions With Brainpower
* Difficulty: Medium
* Tags: array, dp
*
* 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[][]} questions
* @return {number}
*/
var mostPoints = function(questions) {

};
```

TypeScript Solution:

```
/**
* Problem: Solving Questions With Brainpower
* Difficulty: Medium
* Tags: array, dp
*
* 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 mostPoints(questions: number[][]): number {

};
```

C# Solution:

```
/*
* Problem: Solving Questions With Brainpower
* Difficulty: Medium
* Tags: array, dp
*
* 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 long MostPoints(int[][] questions) {

}
}
```

C Solution:

```
/*
* Problem: Solving Questions With Brainpower
* Difficulty: Medium
* Tags: array, dp
*
* 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
*/

long long mostPoints(int** questions, int questionsSize, int*
questionsColSize) {

}
```

Go Solution:

```
// Problem: Solving Questions With Brainpower
// Difficulty: Medium
// Tags: array, dp
//
// 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 mostPoints(questions [][]int) int64 {

}
```

Kotlin Solution:

```
class Solution {
fun mostPoints(questions: Array<IntArray>): Long {

}
}
```

Swift Solution:

```
class Solution {
func mostPoints(_ questions: [[Int]]) -> Int {

}
}
```

Rust Solution:

```
// Problem: Solving Questions With Brainpower
// Difficulty: Medium
// Tags: array, dp
//
// 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 most_points(questions: Vec<Vec<i32>>) -> i64 {

}
}
```

Ruby Solution:

```
# @param {Integer[][]} questions
# @return {Integer}
```

```
def most_points(questions)

end
```

PHP Solution:

```
class Solution {

/**
* @param Integer[][] $questions
* @return Integer
*/
function mostPoints($questions) {

}
}
```

Dart Solution:

```
class Solution {
int mostPoints(List<List<int>> questions) {

}
}
```

Scala Solution:

```
object Solution {
def mostPoints(questions: Array[Array[Int]]): Long = {

}
}
```

Elixir Solution:

```
defmodule Solution do
@spec most_points(questions :: [[integer]]) :: integer
def most_points(questions) do

end
end
```

Erlang Solution:

```
-spec most_points(Questions :: [[integer()]]) -> integer().
most_points(Questions) ->
.
```

Racket Solution:

```
(define/contract (most-points questions)
(-> (listof (listof exact-integer?)) exact-integer?)
)
```